DISEÑO E IMPLEMENTACIÓN DE UNA MÁQUINA PARA EL RUTEO Y PERFORACIÓN DE PCB

WILLIAM GONZÁLEZ COQUEL

HÉCTOR FABIO REYES CARVAJAL

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

FACULTAD DE INGENIERÍAS

PROGRAMA DE INGENIERÍA ELECTRÓNICA Y MECATRÓNICA

CARTAGENA D.T Y C

2014

DISEÑO E IMPLEMENTACIÓN DE UNA MÁQUINA PARA EL RUTEO Y PERFORACIÓN DE PCB

WILLIAM GONZÁLEZ COQUEL

HÉCTOR FABIO REYES CARVAJAL

Trabajo de grado para optar al título de Ingeniero Electrónico y Mecatrónico

Director:

ING. JOSÉ LUIS VILLA RAMÍREZ

Ingeniero Electrónico

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

FACULTAD DE INGENIERÍAS

PROGRAMA DE INGENIERÍA ELECTRÓNICA Y MECATRÓNICA

CARTAGENA D.T Y C

2014

Nota de aceptación

_____

_____

_____

_____

_____

_____

_____

Firma del jurado

_____

Firma del jurado

Cartagena D.T. y C., Diciembre de 2014

AGRADECIMIENTOS

Agradezco primero que todo a Dios por darme la sabiduría, fortaleza, paciencia y perseverancia para alcanzar este gran logro en mi vida. A mis padres en especial a mi madre por todo ese esfuerzo, apoyo incondicional, por siempre creer en mí y por motivarme para alcanzar mis metas. A mi hermana que siempre me tendió la mano incondicionalmente y por los sacrificios que hizo para que yo pudiera ser un profesional.

Gracias a mis profesores y todas esas personas que compartieron sus conocimientos, en especial a mi tutor de grado José Luis Villa Ramírez quien con su visión y persistencia pudo impulsarnos a materializar esta gran idea y a su vez me enseño a ver el mundo desde otra perspectiva; Agradezco a William Castellar Lora quien con su experiencia, ingenio, y perseverancia fue de gran ayuda para el desarrollo de este proyecto; Y en general agradezco a todos aquellos que aportaron su granito de arena a lo largo de este camino y que gracias a ellos hoy soy un profesional y una mejor persona.

William Enrique Gonzalez Coquel

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

Las Tarjetas de Circuitos Impresos (PCB por su significado en inglés) son cruciales para la manufactura de todo tipo de dispositivos electrónicos y genera ventas mundiales por el valor de 1 billón de dólares al año (Gutierrez, Rodriguez, & Garzon, 2008). En la actualidad existen diferentes formas de fabricar una PCB. En particular a nivel de prototipo y de circuitos de baja complejidad se realiza de forma manual o mediante otros mecanismos que de una u otra forma incluyen intervención humana. Dichos procesos pueden resultar altamente riesgosos debido a los procedimientos con que se trabaja (técnica del planchado, de la tinta indeleble).

Uno de los factores limitantes de crecimiento del sector de la electrónica en Colombia ha sido la baja calidad de los productos nacionales, lo que está directamente relacionado con el atraso tecnológico en el sector y una baja capacidad de (Gutierrez, Rodriguez, & Garzon, 2008) inversión.

La industria de manufactura de PCB tiene un amplio historial de exposición a grandes cantidades de productos químicos peligrosos tales como formaldehído, dimetilformamida y el plomo, y muy a pesar de que los avances tecnológicos han demostrado que muchas de estas sustancias son tóxicos-reproductivos y cancerígenos, siguen siendo utilizadas en gran parte de los países de Asia. A raíz de esto se han eliminado muchos productos utilizados para la fabricación de PCB pero aun así los métodos químicos siguen siendo los más utilizados.

Hay diferentes formas de fabricar una PCB, los diferentes métodos de fabricación, son: ataque químico o mecanizado químico (Chemical etching or chemical machining), ataque electroquímico o fresado químico (Electrochemical etching or chemical milling) y por último el grabado mecánico mediante fresado (Mechanical etching by milling), ésta última no requiere de ningún agente químico durante el proceso (Khandpur, 2006).

El fresado de circuitos es un proceso llevado a cabo por una máquina de tipo fresadora con movimiento en los ejes X, Y o X, Y, Z. Éstas máquinas comúnmente trabajan en conjunto con computadoras que controlan sus movimientos a partir de un patrón inicial que sería el diseño deseado sobre la placa de circuito. Dicho diseño es inicialmente hecho en un software que luego lo traduce a un lenguaje conocido por la computadora que controla la máquina. A partir de este punto no habrá más intervención humana que colocar la placa sobre la máquina y luego al finalizar el proceso retirarla.

Las máquinas de fresado tipo CNC han tenido nuevo auge en años recientes debido a la posibilidad de implementarlas en arquitecturas abiertas y de bajo costo (Suh, Kang, Chung, & Stroud, 2008). Su versatilidad reside en la posibilidad de tener prototipos de bajo costo, con un impacto ambiental muy limitado y con un tiempo de fabricación aceptable. Este proceso es adecuado para el diseño y desarrollo de las tarjetas de prototipo.

En estas máquinas, la información del diseño es suministrada en el formato Gerber (conocido como formato RS-274X), que luego se utiliza en el software y la máquina de creación de prototipos. En este trabajo se ha seleccionado este método debido a que somete al usuario al menor riesgo posible.

El objetivo general de éste trabajo de grado es diseñar e implementar una máquina de bajo costo para la elaboración de PCB.

Para lograrlo se han planteado los siguientes objetivos específicos:

- Definir la arquitectura del sistema, teniendo en cuenta su viabilidad y fiabilidad.
- Diseñar un traductor de un fichero Gerber en coordenadas X, Y y Z para la máquina CNC.

- Elaborar una herramienta software con interfaz gráfica, capaz de reconocer el fichero Gerber y de generar la trayectoria del efector final, aparte de monitorear las principales variables del sistema.
- Diseñar e implementar el sistema de movimiento de la máquina CNC
- Modelar, diseñar, simular e implementar un sistema de control de posición para cada uno de los tres ejes de movimiento en el espacio.
- Validar la máquina desarrollada con tres tipos de PCB de diferente complejidad.

En este documento, el lector encontrará que en el capítulo 2 se describe de forma detallada la metodología de diseño implementada en éste proyecto, con el fin de mostrar una serie de pasos y decisiones que normalmente se llevan a cabo antes de entrar en la etapa de diseño de cualquier proyecto. Más adelante, en los capítulos 3, 4 y 5 se describe de forma detallada la etapa de diseño del hardware mecánico y electrónico, incluyendo la etapa de diseño de software.

El capítulo 6 muestra los resultados obtenidos de la ejecución de este proyecto y en el capítulo 7 se presentan las conclusiones a la que se llegó luego de terminar la etapa aplicativa de este proyecto.

2 METODOLOGÍA

La literatura de dispositivos Mecatrónicos propone diferentes metodologías de diseño (Silva, 2008), (Bishop, 2006), (Melón, Ballester, & Navarro, 2001). Al analizar cada una de estas metodologías y al aplicarla al objetivo de este trabajo, se elaboró la metodología de la Figura 1 en la cual se tiene en cuenta todas las fases de diseño desde los parámetros de diseño hasta su implementación, pruebas y ajustes.

Teniendo la metodología de la Figura 1, donde a partir de los parámetros de diseño se estudia un grupo de soluciones, se realiza un análisis preliminar de cada solución verificando que cumpla con los parámetros de diseño. De todas las posibles soluciones se escoge la más adecuada y a partir de ésta se empieza a realizar el diseño detallado de la electrónica, mecánica, software y comunicación. En las siguientes secciones se detallan los parámetros de diseño establecidos para éste proyecto y la aplicación de la metodología para el hardware y el software del proyecto.

2.1 Parámetros de diseño

El objetivo de la máquina es que sirva para fabricar PCBs, mediante el método de fresado mecánico, a un bajo costo. Por tanto se definen las siguientes premisas para el diseño:

- Debe poder hacer varios diseños sobre una misma placa virgen.
- Área de trabajo de 1000 cm^2.
- Alta precisión, en lo posible del orden de 50 µm.
- Buena velocidad de avance.
- Debe tardar la menor cantidad de tiempo posible en realizar el trabajo.
- Debe poder construirse con un bajo presupuesto.
- Debe poder trabajar con la mínima intervención de un operario.

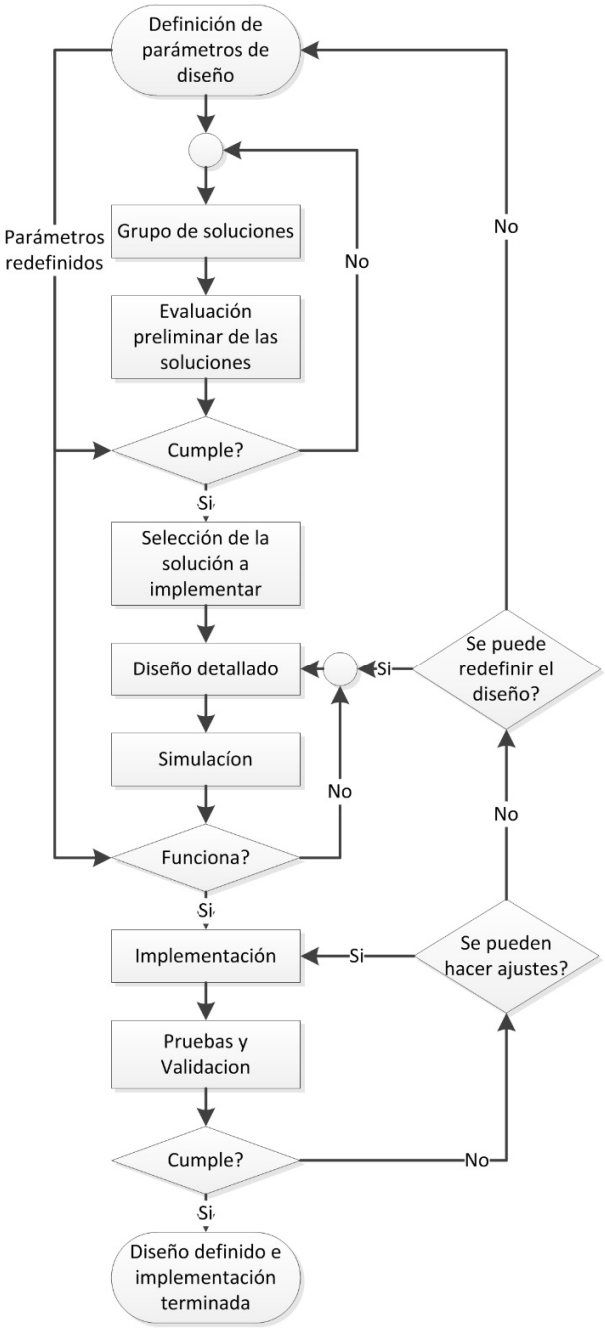


Figura 1. Metodología aplicada (Elaboración Propia)

La máquina debe poder fabricar un diseño PCB, los cuales hoy en día no son de gran tamaño, se piensa en la posibilidad de hacer varios diseños sobre una misma placa virgen, pues estas se encuentran en gran variedad de tamaños en el mercado. Por esta razón, se concluye que el área de trabajo de la maquina seria de 1000 cm^2. También se estipula que la máquina debe tener una alta precisión, en lo posible del orden de 50 µm con una

velocidad de avance media, para ofrecer mayor precisión al sistema y lograr un compromiso con el tiempo de impresión de la PCB. Además de esto se busca que la maquina sea fácil de utilizar para cualquier persona, sin que exista la necesidad de leer un manual muy extenso para poder manejar el software o para montar las placas de cobre vírgenes sobre la máquina, todo esto con el objetivo de que el usuario vea lo sencillo que es y no tenga la necesidad de hacer el PCB a mano por no saber usar la máquina.

La máquina debe poder construirse con un bajo presupuesto, de esta manera se busca tener una buena precisión y poder tener un diseño competitivo con respecto a los sistemas que ya existen.

El proceso de fabricación de un PCB debe poder realizarse en la menor cantidad de pasos posibles. Debe poder realizarse con la mínima intervención de un operario después de haber cargado los datos que le indiquen a la máquina que hacer.

2.2 Arquitectura del Sistema Mecánico

En esta sección se detalla la aplicación de la metodología al hardware mecánico de éste proyecto.

2.2.1 Mecánica

El trabajo de rutear y perforar PCB, finalmente será realizado por la parte mecánica, comandada por la parte electrónica y su software. Esto implica que si algo no funciona bien en la mecánica, simplemente la maquina no realizará un buen trabajo. Es por ello que se hace necesario seleccionar con cuidado tanto la forma de la máquina, como los elementos que la forman.

2.2.1.1 Estructura mecánica

La estructura mecánica es una parte muy importante en la máquina, pues depende de la configuración de ésta misma, si la maquina cumplirá o no las especificaciones de diseño. Teniendo en cuenta que el trabajo de la maquina requiere movimientos en el espacio que obedecen a coordenadas cartesianas, se empezó por analizar las maquinas ya existentes que trabajan de forma similar, como las fresadoras.

Existe gran variedad de fresadoras, en su gran mayoría el efector final solo tiene movimiento vertical, y es la mesa de trabajo la que se mueve de forma transversal y longitudinal, esto implica que su área de trabajo es pequeña respecto al espacio que ocupa. Una mesa móvil resulta inadecuada para la finalidad de este proyecto, puesto que el material sobre el cual se trabajará ocupara un área grande, y utilizar una mesa de trabajo móvil implica que ésta debe ocupar un espacio mayor al área de trabajo. Por otro lado existen fresadoras especiales, hechas para trabajar con piezas de gran tamaño.

Entre los tipos de fresadoras de gran tamaño se tiene:

- Fresadoras de pórtico (Figura 2).
- Fresadoras de puente móvil (Figura 3).

Ambas con una estructura mecánica, que se asemeja a la de un puente grúa, hechas para trabajar en piezas de gran tamaño. La diferencia entre ambas es que en la primera, la herramienta solo tiene movimiento vertical y transversal, es decir que el puente no se mueve. El movimiento longitudinal lo sigue realizando la meza de trabajo. Mientras que en la fresadora de puente móvil la mesa está siempre estática y es el efector final quien tiene toda la movilidad tanto vertical y transversal como longitudinal. La ventaja de la fresadora de puente móvil sobre la otra es que esta puede trabajar en piezas mucho más largas.

Analizando desde el punto de vista del área de trabajo, para poder utilizar la estructura de una fresadora de pórtico sería necesario ocupar mucho más espacio del que ocuparía una fresadora de puente móvil con la misma área de trabajo, por tanto la opción seleccionada a implementar es la fresadora de puente móvil.

Figura 2. Fresadora pórtico (Correanayak, s.f.).

Figura 3. Fresadora de puente móvil (Correanayak, s.f.).

2.2.1.2 Sistema de guías

Todo mecanismo de traslación requiere un sistema de guías que mantenga la trayectoria del elemento móvil y, si es necesario, que soporte la carga del mismo. Para esta tarea existen diversas soluciones, pero la solución que ofrecen los diferentes fabricantes son las siguientes:

- Ejes guía con cojinetes de deslizamiento o bujes (Figura 4).
- Ejes guía con rodamientos lineales de bolas recirculantes. (Shaft guidance systems with linear ball bearings) (Figura 5A).
- Perfiles de riel guía. (profile rail guides) (Figura 5B).
- Rieles guía de precisión. (precision rail guides) (Figura 5C).

Los cojinetes de deslizamiento no son más que elementos de rozamiento a los cuales se les reduce la fricción mediante lubricación, lo cual quiere decir que requiere constante mantenimiento. Pero también es importante la selección de materiales tanto del buje como de la barra guía, para que el coeficiente de fricción entre ambos sea el más bajo posible. Por lo general se usan barras de acero inoxidable pulidas y cojinetes de bronce, puesto que el coeficiente de fricción entre ambos materiales es aproximadamente (μ=0.18). Como son elementos de rozamiento, el cojinete requiere cierta holgura para que pueda mantenerse lubricada la superficie de contacto entre el cojinete y el eje guía, por tanto es posible que exista un pequeño juego.

Figura 4. Cojinetes de deslizamiento en bronce (Sankyo Oilless Industry, Inc, s.f.).

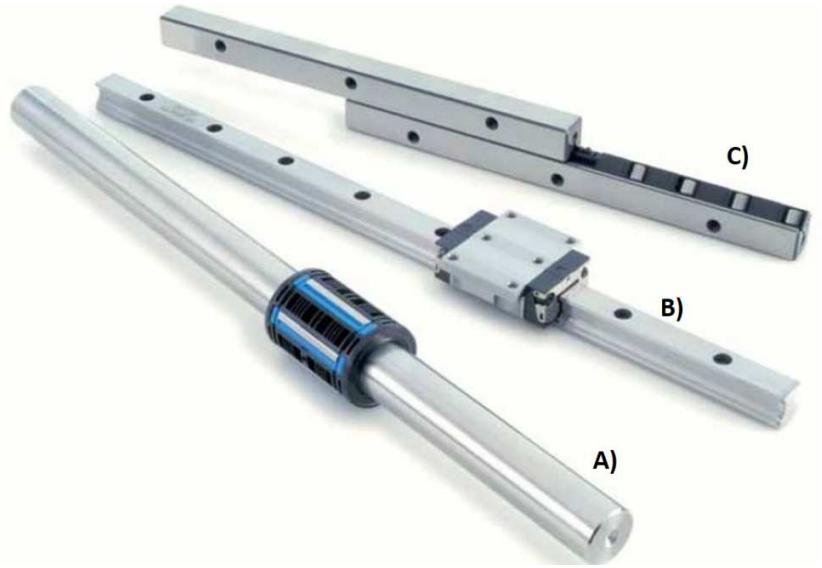


Figura 5. Diferentes sistemas de guías lineales con elementos de rodadura (SKF, Grupo, 2011).

Los tres sistemas de guiado lineal con elementos de rodadura, son de alta precisión, soportan cargas elevadas y trabajan desde velocidades muy bajas hasta velocidades altas, con ciertas diferencias.

Los dos primeros sistemas tienen cierta ventaja sobre el último, ya que este tiene un recorrido limitado, mientras que el elemento rodante en los otros dos es de recorrido ilimitado. No obstante los rieles guía de precisión, son el sistema de guías lineales con mayor capacidad de carga. En la Figura 6. Se muestra una comparación de los tres sistemas en cuanto a velocidad, precisión y capacidad de carga.

Teniendo nuevamente en cuenta los parámetros iniciales del diseño, la mejor opción es una solución de bajo costo que conserve las características necesarias para el buen funcionamiento de la máquina. Inicialmente se pensó que el sistema de barra guía y cojinete de deslizamiento podría ser una solución que conservara las características necesarias para el desarrollo del proyecto, además de ser la opción más económica. Aun así se pensó en la posibilidad de tener que cambiar el sistema de guías por otro. Y ya que los tres sistemas con elementos de rodadura son de alta precisión y alta capacidad de carga, solo se puede seleccionar un sistema que sea más económico respecto a los otros

dos. En este caso la opción seleccionada fue el sistema de ejes guía con rodamientos lineales de bolas recirculantes.

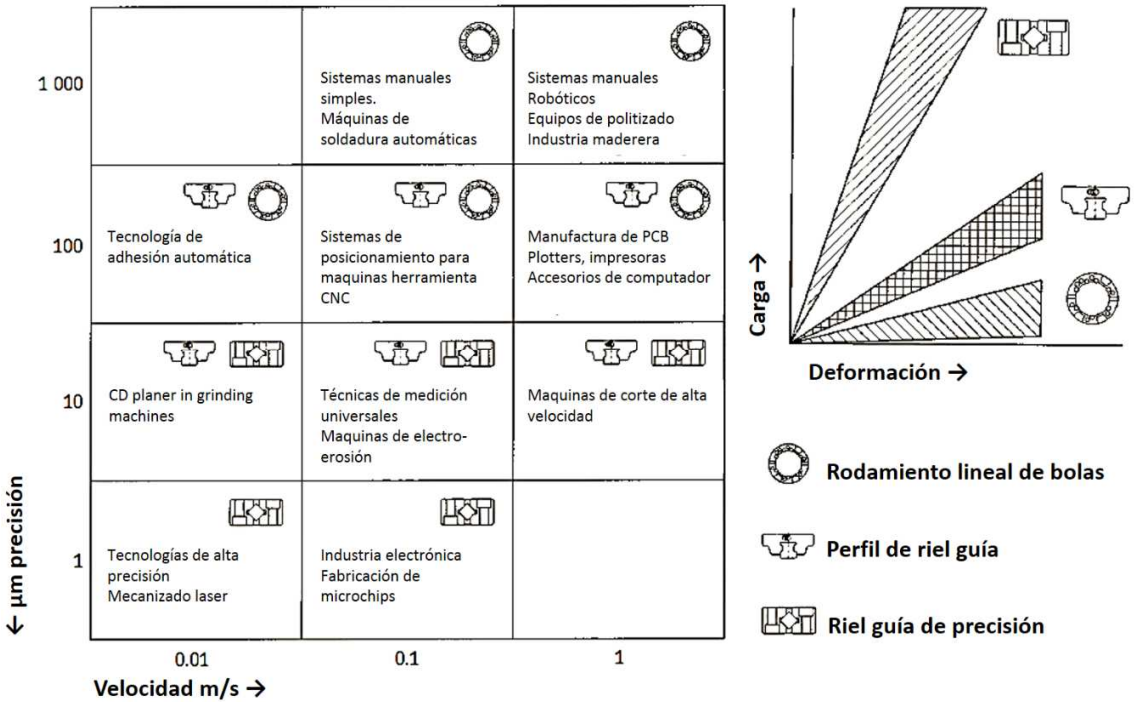


Figura 6 Comparación entre sistemas de guías lineales (SKF, Grupo, 2011).

Existen dos opciones, barras guía con cojinetes de deslizamiento y barras guía con rodamientos de bola. Ambos sistemas trabajan con ejes como elemento de guía, y ya que es mucho más económico fabricar un cojinete de deslizamiento, se seleccionaron las dimensiones apropiadas de tal manera que en caso de tener que remplazar el cojinete por un rodamiento, no fuese necesario cambiar las barras.

2.2.1.3 Mecanismo de movimiento

Tomando en cuenta los parámetros de diseño, el tipo de estructura mecánica y el tipo de movimiento, se determinó que era necesario utilizar un mecanismo de transformación de movimiento rotatorio en movimiento lineal, que es el tipo de movimiento que requiere la estructura mecánica establecida anteriormente.

Se investigó las diferentes soluciones existentes para el problema del mecanismo de transformación de movimiento, de lo cual se obtuvieron las siguientes soluciones:

- Mecanismo piñón-cremallera.
- Mecanismo piñón-cadena.
- Mecanismo polea-correa.
- Mecanismo tornillo-tuerca.

El mecanismo piñón-cremallera hace la transformación del movimiento mediante dos elementos dentados, un piñón o rueda dentada y una barra dentada comúnmente llamada cremallera. Este mecanismo puede utilizarse de dos formas, fijando la cremallera en una superficie y haciendo que el piñón gire libremente y se desplace sobre la cremallera o fijando el eje de rotación del piñón y haciendo que la cremallera se desplace según el giro del piñón.

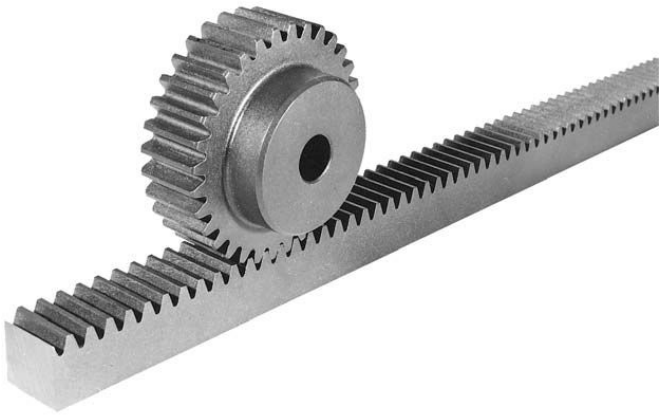


Figura 7. Mecanismo piñón cremallera (Morse Cross, s.f.).

El mecanismo piñón-cadena se utiliza comúnmente para transmitir un movimiento de rotación de una rueda dentada a otra mediante una cadena, y multiplicar o reducir la fuerza necesaria para ejercer el movimiento, pero si la distancia entre los piñones es larga y si se fija un elemento a una sección de la cadena entre ambos piñones, se obtiene un mecanismo de traslación accionado por un movimiento rotatorio.

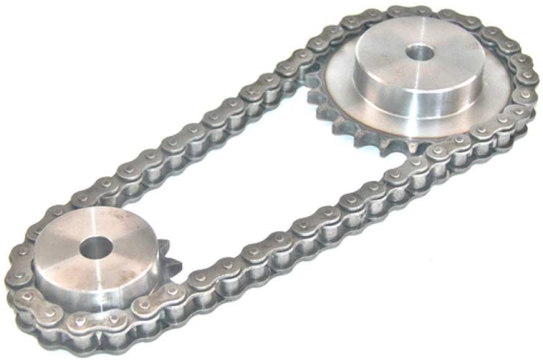


Figura 8. Mecanismo piñón cadena (Surtirodamientos, s.f.).

El mecanismo polea correa funciona de la misma forma que el mecanismo Piñón cadena, diferenciándose únicamente en los elementos mecánicos que usa para la transmisión del movimiento.

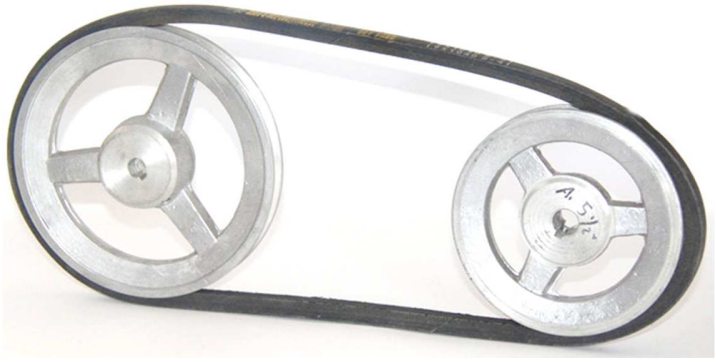


Figura 9.Mecanismo polea correa (Surtirodamientos, s.f.).

El mecanismo tornillo-tuerca consiste en una varilla roscada y un elemento con un agujero roscado o tuerca. El mecanismo se utiliza comúnmente fijando la tuerca en el elemento a desplazar y haciendo girar el tornillo sobre su propio eje mientras se encuentra acoplado a la tuerca, de esta manera el movimiento giratorio del tornillo hace que la tuerca se desplace de forma longitudinal. También es posible fijar el tornillo y hacer girar la tuerca sobre el mismo, lo cual producirá el mismo efecto.

Luego de realizar el análisis de ventajas y desventajas de cada mecanismo se llegó a la conclusión que la mejor solución es el mecanismo tornillo tuerca. En la Tabla 1 se muestra las ventajas y desventajas de cada uno de los mecanismos.

Figura 10. Husillo de bolas con tuerca de bolas.

Tabla 1. Ventajas y desventajas de los mecanismos de traslación.

MECANISMO	VENTAJAS	DESVENTAJAS
Piñón-cremallera	• Económico a distancias largas. • Silencioso.	• Costoso a distancias cortas. • Presenta desgaste, que con el tiempo se traduce en error de precisión. • Mecanismo reversible. • Requiere mantenimiento constante.
Piñón-cadena	• Económico.	• Ruidoso. • Las cadenas son flexibles, lo que se traduce en error de precisión. • Mecanismo reversible. • Requiere mantenimiento constante.
Correa-polea	• Silencioso. • Económico. • No requiere mantenimiento.	• Presenta deslizamientos, lo que se traduce en error de precisión. • Mecanismo reversible.
Tornillo tuerca	• Silencioso. • Relativamente económico a distancias cortas. • De alta precisión. • Mecanismo no reversible (para ángulos de avance grandes). • Requiere poco mantenimiento.	• Costoso a distancias grandes. • Mecanismo reversible (para ángulos de avance pequeños).

El mecanismo tornillo tuerca tiene algunas variaciones como son el tornillo de potencia y el tornillo a bolas. Ambas opciones son muy eficientes y tienen una alta ventaja mecánica, lo que quiere decir que pueden someterse a cargas axiales muy elevadas; pero el tornillo a bolas tiene ventajas sobre el tornillo de potencia, ya que este último es un mecanismo de fricción mientras que el tornillo a bolas posee cojinetes internos que reducen la fricción al máximo. Reduciendo de esta manera la potencia necesaria en los actuadores para ejercer el movimiento y aumentando la eficiencia del mismo. También son mucho más precisos, pero fabricar un tornillo de potencia es mucho más económico que comprar un tornillo a bolas y existen formas de reducir el error de un tornillo de potencia, por lo cual inicialmente esta fue la opción que se consideró a utilizar en este proyecto.

Por razones que se explican en la sección 3.4.1 del Capítulo 3. Se decidió que en lugar de usar tornillos de potencia, se usarían tornillos a bolas.

2.2.1.4 Actuadores

El campo de los actuadores es muy amplio debido a que existen muchos tipos de actuadores que se clasifican según el principio físico de funcionamiento. Pero en este tipo de aplicaciones para bajas cargas, se utilizan actuadores eléctricos.

De los actuadores eléctricos existe gran variedad, de los cuales se tiene:

- Motores Paso a Paso.
- Motores DC
- Motores AC.

Los motores paso a paso son muy utilizados en este tipo de aplicaciones puesto que se puede llegar a la precisión requerida solamente calculando el número de pasos necesarios en el motor. Pero para una mayor precisión es necesario un mayor número de pasos, esto

eleva el costo del motor y uno de los objetivos de este proyecto es la economía del mismo, es decir, que sea de bajo costo. Por otra parte, los motores paso a paso no tienen en cuenta la posición real del efector final, y su arquitectura no permite un buen control en lazo cerrado.

Los motores AC por lo general son de gran tamaño, puesto que son fabricados para maquinas más grandes y ocuparían demasiado espacio. Estos funcionan con corriente alterna y su control se hace mediante frecuencia, lo cual lo hace mucho más complejo

Los motores DC son fabricados en gran variedad de tamaños y por lo general son de bajo costo. Estos funcionan con corriente continua y se controlan mediante variación de ancho de pulso o PWM, lo cual hace que su control sea relativamente fácil. Estos constituyen una buena alternativa a los motores paso a paso en cuestiones de precisión, si se les complementa con un buen sistema de control. Por estas razones se decidió usar motores DC como actuadores en éste proyecto.

Hasta éste punto se ha investigado, evaluado y seleccionado varias soluciones para los diferentes problemas que representa la construcción de la parte mecánica a partir de los parámetros iniciales del diseño. Tal y como expone la metodología implementada en este proyecto, el siguiente paso a seguir es el diseño detallado, lo cual implica seleccionar las referencias y/o características especificas de los elementos mecánicos a utilizar mediante algunos cálculos. Se realizaran simulaciones de la estructura mecánica para determinar cuál será el mejor diseño final y se realizaran pruebas para validar el funcionamiento.

Agrupando las soluciones anteriormente seleccionadas en una única solución para el hardware mecánico, se tiene una muestra previa de la maquina en la Figura 11.

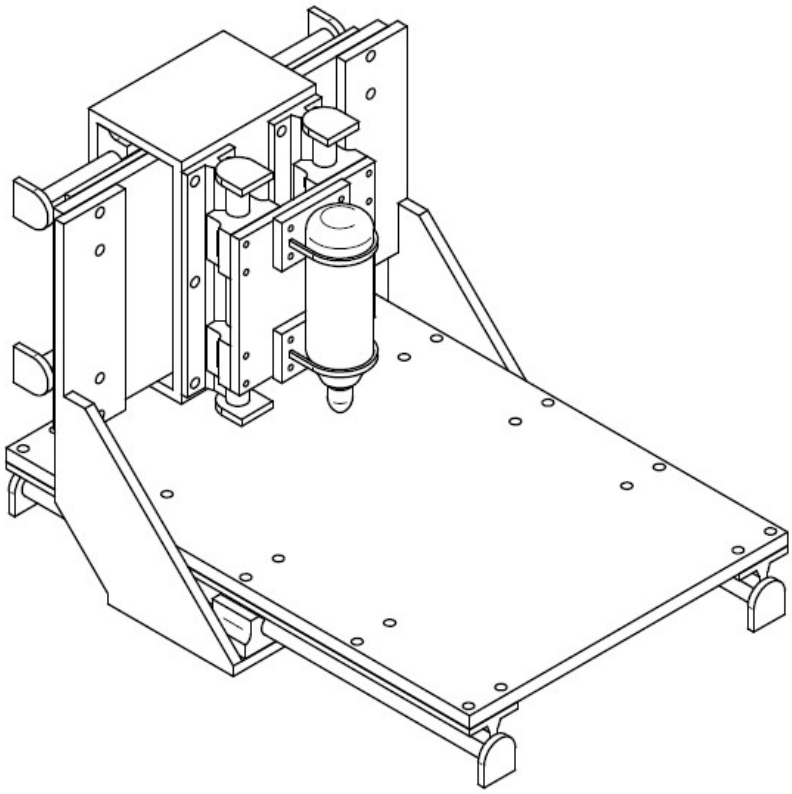


Figura 11. Modelo previo de la máquina.

2.3 Arquitectura del control electrónico

La electrónica está muy relacionada con el tipo de actuador a utilizar y la forma de cómo se va a comunicar el computador con la máquina que ya está establecido en los parámetros de diseño que es por USB. En la sección 2.2.1.4 se escogió el motor DC por las diferentes ventajas que ofrece y porque cumple con los parámetros de diseño.

El control de las maquinas CNC basadas en arquitectura estándar se encuentra en el computador, donde el control trabaja con un sistema operativo en tiempo real en el cual se procesan todas las señales de posición provenientes de los actuadores y se obtienen las salidas de control que alimentan a una etapa de potencia que a su vez alimenta a los actuadores. Este tipo de arquitectura tiene una desventaja y es que el protocolo USB por ser una comunicación serial requiere que el sistema trabaje a una mayor velocidad para

que se pueda realizar un buen control de posición, de lo contrario se perdería información valiosa para el control.

Como uno de los parámetros de diseño es que la maquina trabaje por USB, se requiere que el control sea implementado por Hardware externo y dejar el computador que solo suministre la posición a la cual se debe ubicar el efector final y visualizar la posición en la que se encuentra este mismo.

Por la forma como trabaja el USB, se requiere que el transmisor/receptor de comunicación sea digital y trabaje a una alta frecuencia en el orden de los 100MHz para que pueda trabajar a la par con el USB del computador y no se pierdan datos durante la transmisión.

Por otra parte, la mejor forma de variar la velocidad de un motor DC es por medio de un PWM, es decir cada controlador necesita de una salida PWM capaz de poder suministrar la potencia suficiente al motor sin que ésta se dañe y al mismo tiempo debe tener la capacidad de poder cambiar su sentido de giro.

Para que el control pueda saber la posición en la que se encuentra el actuador, se requiere de un encoder el cual nos dirá la posición del eje del actuador en cualquier instante de tiempo. Por lo tanto la electrónica debe ser capaz de poder decodificar esta señal y suministrarla al control de cada eje.

Otro requisito de la electrónica, es que la maquina debe reconocer la posición inicial y final de cada eje de movimiento por medio de un sensor o final de carrera, esto con el fin de proteger al mecanismo y al actuador.

Hasta el momento se sabe que la electrónica debe ser capaz de procesar las señales del USB, obtener señales por medio de tres encoder y decodificarla, poder suministrar tres

señales PWM y a la vez poder controlar la posición de los actuadores y leer constantemente los seis finales de carrera que tiene la máquina.

Por funcionalidad, mantenimiento y organización, se estipula un diseño electrónico modular, de tal forma que cada módulo realice una tarea específica. Se tienen tres ejes mecánicos (X, Y, Z), cada eje usa un módulo controlador. Otro módulo se encarga de la comunicación y suministra la información necesaria a cada uno de los módulos controladores. Este diseño modular se detalla en la Figura 12.

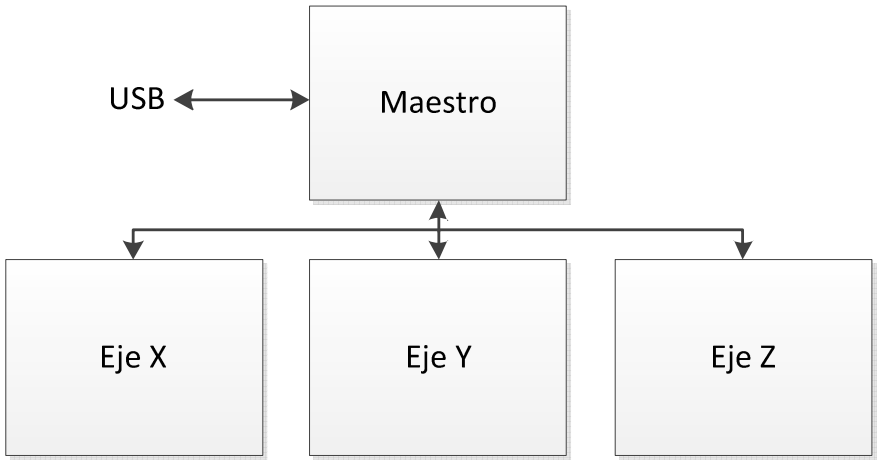


Figura 12. Diagrama modular de la electrónica. Fuente: (Elaboracion, propia)

Teniendo un diseño modular es más fácil plantear la electrónica debido a que cada módulo debe cumplir con una tarea específica. Para los tres ejes (X, Y, Z), el modulo funciona de la misma manera, lo que varía es la señal que recibe y envía cada uno. Estos módulos serán llamados Esclavos. El módulo que se encarga de la comunicación USB y de enviar y recibir la información necesaria de los tres módulos Esclavos, será llamado Maestro.

2.3.1 Módulo Maestro

Una vez definido en la Figura 12 el modo de trabajo de la electrónica, se puede apreciar que el módulo del Maestro se encarga de hacer la interfaz de comunicación entre cada controlador y la aplicación software del computador.

El modulo tiene una sección que se encarga de la comunicación USB, la longitud de esta trama es de 64 bytes. La información es procesada, y luego la trama USB es divida en pequeñas tramas de información, que se envían a cada módulo Esclavo. Una función adicional del Maestro es la de accionar el efector final, debido a que el control de este consta únicamente de una señal de encendido y apagado. Por el tipo de señales con las que se trabaja, se hace necesario que la electrónica del Maestro sea digital; entre la variedad de dispositivos digitales se encuentran la lógica cableada, los microprocesadores y los Microcontroladores.

La lógica cableada tiene la desventaja de que la elaboración del diseño se vuelve más complejo y que el hardware es más voluminoso, debido a que son dispositivos lógicos muy básicos. A diferencia de los microprocesadores que tienen todos éstos dispositivos lógicos internamente y solo es cuestión de programar su conexión. Los microprocesadores son una buena opción, pero son costosos y requieren un diseño en descripción de hardware más complejo.

Otra opción es la de utilizar Microcontroladores, ya que hoy en día existen muchas referencias para diferentes tipos de aplicaciones, son muy sencillos de programar y de configurar, y además poseen módulos de funciones específicas (ADC, PWM, USB, SSP, etc.); de estos se puede agregar que son muy asequibles en el mercado local y son de bajo costo. Debido a estas razones, se decidió utilizar Microcontroladores para el modulo Maestro.

Hasta el momento solo falta definir el protocolo de comunicación entre los módulos Maestro y Esclavo de la Figura 12. Existen diferentes protocolos de comunicación los cuales pueden ser tipo serie o paralelo, los tipo paralelo transmiten más de un bit a la vez pero la desventaja es que requieren muchos cables o pines para la comunicación. Los tipo serie envían un bit a la vez y requieren menos pines o cables para la comunicación. Por

ésta razón se utilizara un protocolo de comunicación serie para tener un hardware más compacto.

La comunicación serie puede ser síncrona o asíncrona. En la comunicación serial asíncrona, cada dispositivo debe generar su propia señal de reloj, es decir el emisor y el receptor no se encuentran sincronizados. Éste modo de comunicación tiene bajo rendimiento en la transmisión, no es apto para trabajar a altas velocidades y en caso de errores se pierde siempre una cantidad pequeña de información. Mientras que en la comunicación serial síncrona, se necesitan dos líneas, una línea sobre la cual se transmitirán los datos y otra que transmitirá los pulsos de reloj (la señal de reloj la genera el emisor y es enviada a cada receptor). El tipo de comunicación serial síncrono posee un alto rendimiento en la transmisión y es apta para altas velocidades de transmisión. Por ésta razón el protocolo de transmisión a utilizar en éste proyecto será serial síncrono para que todos los módulos Esclavos trabajen de manera sincronizada y a una alta velocidad de transmisión.

La mayoría de los Microcontroladores tienen el hardware necesario para realizar la comunicación serial síncrona, como es el caso del módulo SSP (Synchronous Serial Port). El módulo SSP se puede configurar para trabajar en uno de dos modos, en el modo SPI (Serial Peripheral Interface) o en el modo I2C (Inter-Integrated Circuit).

La comunicación SPI, incluye una línea de reloj, dato entrante, dato saliente y un pin de “chip select”, que conecta o desconecta la operación del dispositivo con el que se desee comunicar, esto permite multiplexar las líneas de reloj entre los Esclavos. En la Figura 13 se puede apreciar el diagrama de conexión de la comunicación SPI entre un Maestro y tres Esclavos.

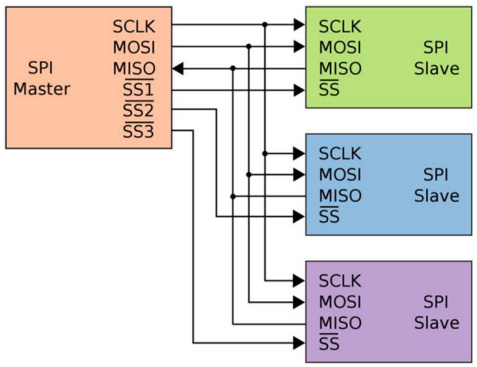


Figura 13. Comunicación SPI, diagrama de conexión entre un Maestro y tres Esclavos.

En la comunicación I2C, la principal característica es que utiliza dos líneas para transmitir la información: una para los datos y otra para la señal de reloj. Los dispositivos conectados al bus I2C tienen una dirección única para cada uno. El dispositivo Maestro inicia la transferencia de datos y además genera la señal de reloj. En la Figura 14 se puede apreciar el diagrama de conexión de la comunicación I2C entre un Maestro y tres Esclavos.

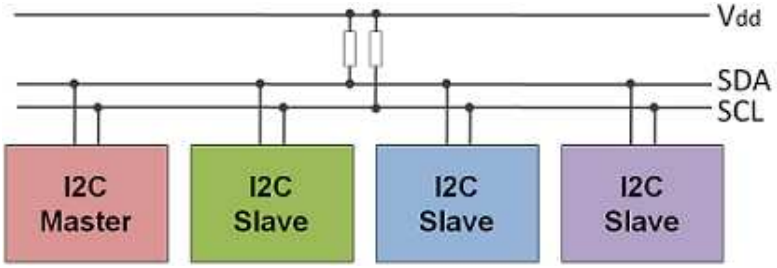


Figura 14. Comunicación I2C, diagrama de conexión entre un Maestro y tres Esclavos.

Al comparar estos dos modos de comunicación, se puede concluir que para éste proyecto ambos cumplen las expectativas de comunicación más sin embargo se optó por utilizar la comunicación I2C, ya que requiere menos líneas de comunicación y por ser un protocolo más estructurado al incluir direcciones en los receptores Esclavos. Pese a que el protocolo SPI ofrece ventajas ante el I2C como transmisiones a alta velocidades y comunicación "Full-Duplex", estas no son características primordiales en éste proyecto.

Una vez evaluada cada una de las soluciones de la electrónica, se desarrolló un diagrama de bloques del módulo Maestro, que se puede apreciar en la Figura 15.

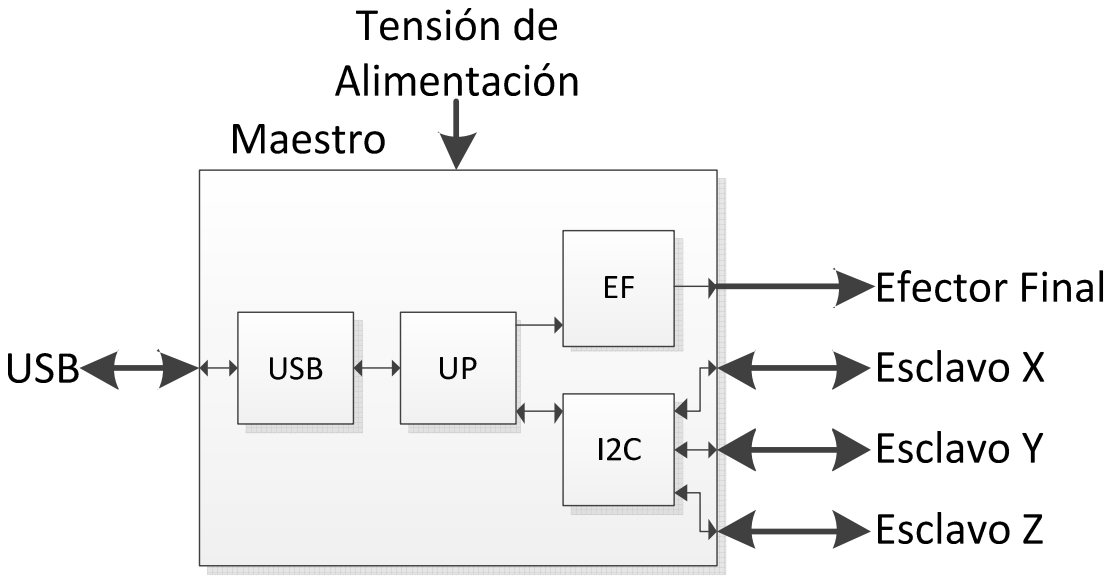


Figura 15. Diagrama de bloque interno del módulo Maestro.

2.3.2 Módulo Esclavo

Para el control de posición es necesario que se haga una retroalimentación de la posición del eje del motor en cualquier instante de tiempo. Para ello se hace necesario que el sistema cuente con un transductor rotativo o encoder los cuales pueden ser incrementales o absolutos. El encoder incremental entrega una cantidad de pulsos digitales por cada revolución o vuelta del eje. Mientras que el encoder absoluto memoriza una posición dada o programada.

La selección de un buen encoder es de vital importancia para el control de posición debido a que la resolución de éste es directamente proporcional a la precisión de posicionamiento de cada eje. Para éste proyecto, se utiliza un encoder tipo incremental debido a que son más económicos y son de fácil adquisición en comparación a los encoder absolutos. Su resolución depende mucho del paso del husillo a bolas.

El control debe ser complementado con un buen sistema de calibración de posición, puesto que los errores del sistema mecánico se van incrementando durante el funcionamiento de la máquina, provocando un pequeño desplazamiento de la referencia el cual no puede ser corregido mediante el encoder. Por esta razón es necesario usar sensores de finales de carrera para poder determinar físicamente la referencia del sistema. Una función adicional de estos sensores es la de evitar colisiones en los extremos del sistema mecánico, logrando así proteger al motor de posibles daños eléctricos y mecánicos.

Por otra parte, el control se puede realizar de dos maneras, puede ser digital o análogo. Como las variables de entrada y de salida del sistema son digitales, la electrónica digital es menos sensible al ruido, es más compacta por lo tanto se requiere menos hardware para su implementación y es más precisa. Para este proyecto el control será de forma digital.

En la sección 2.3.1 se mencionó que la comunicación entre el Maestro y los Esclavos se va a realizar con el protocolo I2C, por lo tanto los módulos Esclavos deben tener la capacidad de poder manejar éste protocolo.

Al ver las facilidades que se tiene al trabajar con Microcontroladores, se debe seleccionar un dispositivo que sea capaz de trabajar con la comunicación I2C, que tenga dos salidas PWM, capacidad para leer el encoder incremental y los finales de carrera.

Una desventaja que tienen los Microcontroladores, es que sus salidas son de baja potencia, por tanto no son capaces de suministrar la tensión y corriente suficientes para mover al motor DC, por esta razón es necesario implementar una etapa de potencia la cual amplifique la tensión y corriente de la señal PWM. Normalmente esto se logra mediante un puente H. En la Figura 16 se desarrolló un diagrama de bloques del módulo Esclavo.

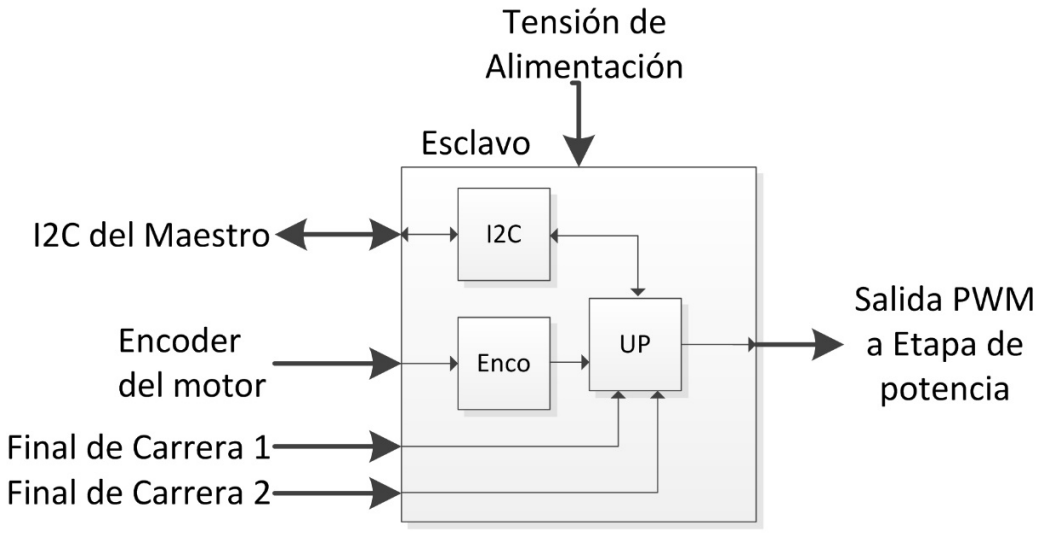


Figura 16. Diagrama de bloque interno del módulo Esclavo.

2.4 Arquitectura del Software Intérprete

2.4.1 Software Intérprete e Interfaz Grafica

Una de las finalidades de este proyecto es poder controlar las posiciones del efector final desde una computadora para elaborar PCB a través del fresado mecánico. Para ello se hace necesario en todo tipo de aplicación controlada por computador, una interfaz hombre máquina (HMI, Human Machine Interface).

Uno de los parámetros de diseño es que la interfaz gráfica trabaje con la información proporcionada por un archivo denominado Gerber, exportado por la mayoría de software de diseño electrónico, el cual contiene la información necesaria para la generación de un PCB. Pero esta información antes de ser enviada a la máquina, debe ser procesada. Por lo cual se ha optado por desarrollar esta aplicación en un software de programación que sea más matemático.

Existen varias opciones de software matemáticos orientados a la programación, pero la mejor opción en este caso es Matlab, ya que tiene muchos aplicativos y herramientas de procesamiento de imagen que podrían ser útiles a la hora de procesar la información contenida en el archivo Gerber.

Hasta éste punto, se conoce la información que contiene el archivo Gerber. Por tanto se ha planteado una posible arquitectura de funcionamiento del software intérprete, la cual se muestra en la Figura 17.

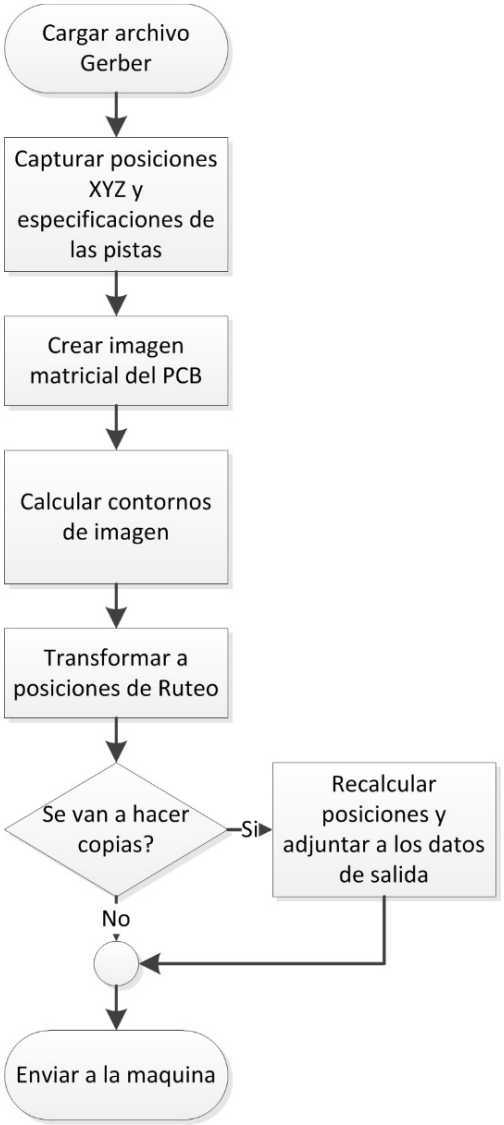


Figura 17. Arquitectura general propuesta para software interprete.

Teniendo en cuenta esta arquitectura, en el capítulo 4 se describe el desarrollo de una aplicación software que procesa la información del archivo Gerber y se comunica con la máquina.

3 MECANISMO

3.1 Rodamientos lineales de bolas recirculantes

Los rodamientos lineales son elementos indispensables en este proyecto puesto que la maquina realiza movimientos de traslación en los que interviene la fricción y se requiere que ésta sea lo menor posible, disminuyendo de esta manera las cargas a las cuales deberá ser sometido el actuador y por consiguiente la potencia necesaria en el antes mencionado.

Comúnmente los rodamientos lineales de bolas poseen una alta capacidad de carga, alta precisión y pueden trabajar a altas velocidades. Por tanto no se hace necesario tener en cuenta estos criterios para la selección de dichos elementos, puesto que las cargas y velocidades presentes serán bajas. Para éste proyecto se tomaron como criterios de selección el tamaño, precio y asequibilidad.

De manera estándar se encuentran en el mercado rodamientos lineales con un diámetro interno de 12mm en adelante. También se pueden encontrar rodamientos con diámetro interno de 5 y 8 mm pero estos son para aplicaciones muy específicas, por tanto son más difíciles de conseguir, sin mencionar que su precio es elevado.

De los rodamientos lineales de bolas se encuentran dos tipos, abiertos (ver Figura 18B) y cerrados (ver Figura 18A). Los rodamientos lineales abiertos se utilizan en aplicaciones bajo condiciones de alta carga y se hace necesario que el eje guía este correctamente soportado en toda su longitud. Mientras que los rodamientos lineales cerrados se utilizan

en aplicaciones normales en condiciones de cargas bajas y el eje guía es una barra que se soporta solo en sus extremos.

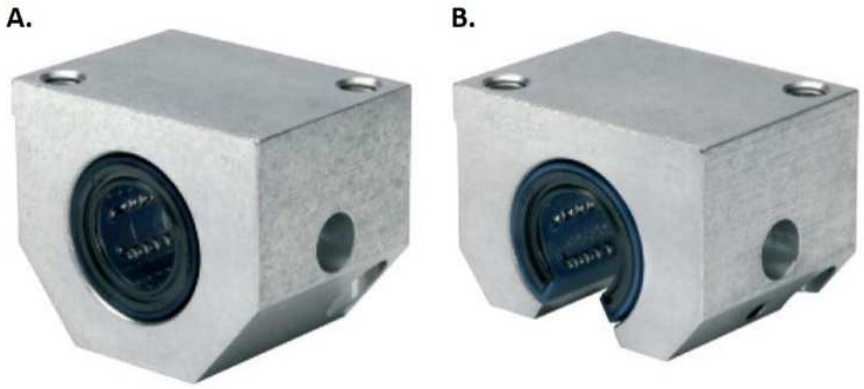
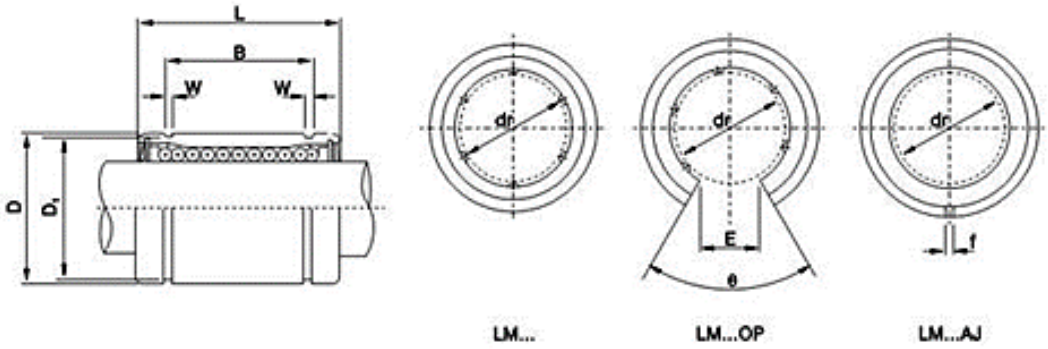


Figura 18. A) rodamiento lineal cerrado. B) Rodamiento lineal abierto.



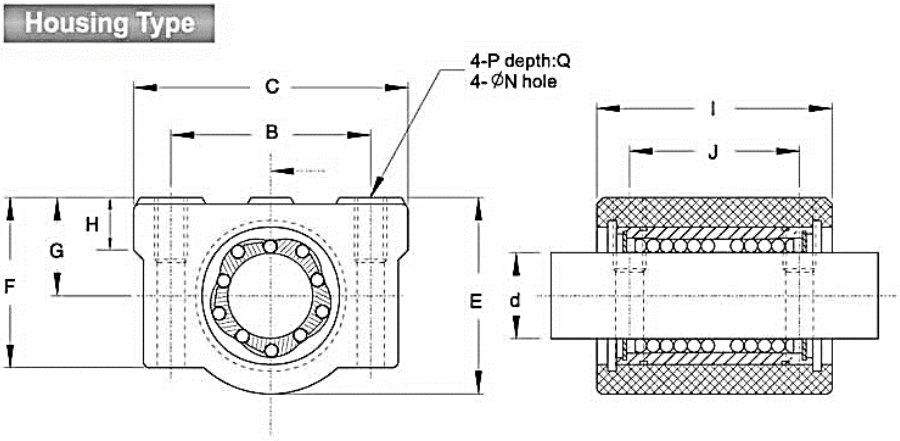
Major dimensions and tolerance											Ratings load		Type
D (mm)	Tolerance (mm)	L (mm)	Tolerance (mm)	B (mm)	Tolerance (mm)	W (mm)	D1 (mm)	f (mm)	E (mm)	θ	Dynamic C(N)	Static Co(N)	
8	0 -0.009	12	0 -0.12	-	-	-	-	-	-	-	88	127	LM4UU
10		15		10.2		1.1	9.6	-	-	-	167	206	LM5UU
12	0 -0.011	19	0 -0.2	13.5	0 -0.2	1.1	11.5	1	-	-	206	265	LM6UU
15		17		11.5		1.1	14.3	1	-	-	176	216	LM8UU
15		24		17.5		1.1	14.3	1	-	-	274	392	LM8UU
19	0 -0.013	29		22		1.3	18	1	6.8	80℃	372	549	LM10UU
21		30		23		1.3	20	1.5	8	80℃	510	784	LM12UU
23		32		23		1.3	22	1.5	9	80℃	510	784	LM13UU
28		37		26.5		1.6	27	1.5	11	80℃	774	1180	LM16UU

Figura 19. Tabla de datos de algunas referencias de rodamientos lineales chinos.

Bajo tales circunstancias se decidió que para éste proyecto, se usarían rodamientos lineales de 12mm (ref. LM12UU en la Figura 19) para el eje Z, con el fin de no aumentar demasiado la masa de la estructura y de reducir la cantidad de espacio utilizado, puesto

que unos rodamientos de mayor diámetro necesitarían soportes de mayor tamaño. Y rodamientos lineales de 16 mm (ref. LM16UU en la Figura 19) para los ejes X y Y, puesto que estos son mucho más económicos y ofrecen mayor robustez a la máquina. Los rodamientos seleccionados son del tipo cerrados, puesto que las cargas que manejaran son relativamente bajas, por lo cual no se hace necesario el uso de un riel especial.

Las dimensiones de los soportes para los rodamientos lineales de referencia LM12UU se muestran en la Figura 20.



Material: Aluminum Alloy — Unit : mm

Model No.	Dimension											load(kgf)		Wt.(g)
	d	B	C	E	F	G	H	I	J	N	P	Ca	Coa	
SC8UU	8	24	34	22	18	11	6	30	18	3.4	M4	28	40	52
SC10UU	10	28	40	26	21	13	8	35	21	4.3	M5	38	56	92
SC12UU	12	30.5	42	28	24	15	8	36	26	4.3	M5	52	80	102

Figura 20. Dimensiones soporte de rodamiento lineal Ref. SC12UU.

Para los rodamientos lineales de referencia LM16UU, también existe una referencia de soporte, la cual es SC16UU. Pero en lugar de utilizar dicha referencia de soporte, se utilizaron soportes de fabricación nacional por razones económicas. Las dimensiones de dichos soportes se expresan en la Figura 21.

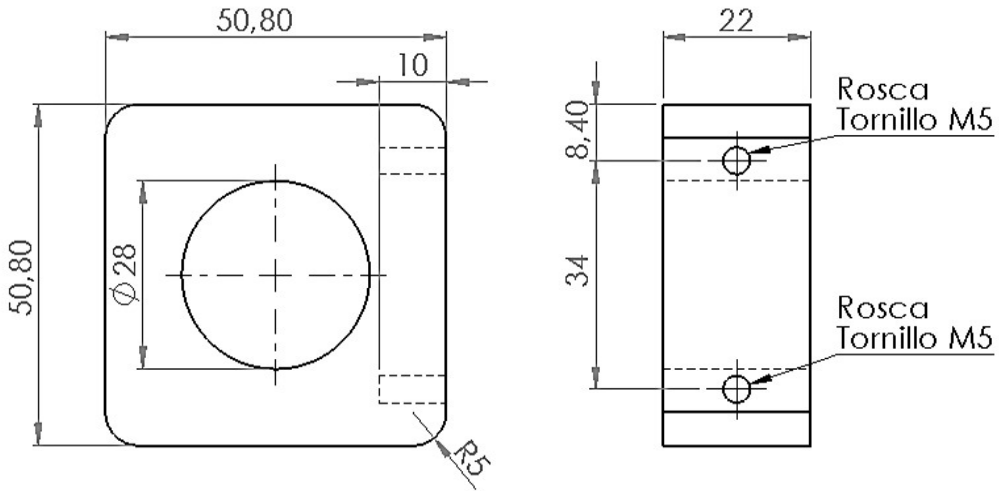


Figura 21. Dimensiones soporte de fabricación nacional para rodamiento lineal LM16UU.

3.2 Barras Guía

Las barras guía son los elementos que ofrecerán una superficie uniforme de contacto con los elementos rodantes. Pero su verdadera función es la de proporcionar a dichos elementos una trayectoria rectilínea de movimiento y soportar las cargas de las partes móviles de la máquina que pueden afectar no solo el funcionamiento de la misma, sino que también dañar otros elementos mecánicos.

Las barras guía utilizadas en este proyecto tienen dimensiones que van designadas por el área de trabajo deseada y por los rodamientos lineales, además de otros elementos mecánicos que por su tamaño le suman longitud para que no se vea afectada el área de trabajo. El material escogido para dichas barras es acero plata que tiene una alta resistencia, además de que es el material más comúnmente utilizado para tales aplicaciones.

En la Figura 22. Se puede apreciar dos unidades de traslación lineal marca SKF, con diferentes tipos de rodamientos, camisas y barras guía. Los cuales se utilizan para aplicaciones de baja carga (A) y alta carga (B). En este proyecto, Las barras guía son

soportadas únicamente en los extremos, como se muestra en el inciso A., debido a razones explicadas en el capítulo 2.2.1.2

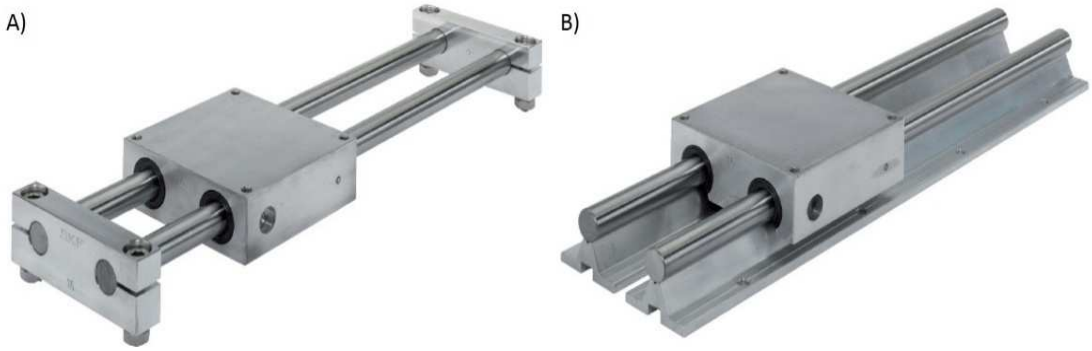


Figura 22. A) Barras guía soportadas en sus extremos. B) Barras guía soportadas de manera uniforme desde la parte inferior.

A continuación se muestra los resultados de un análisis de elementos finitos, hecho a las barras guía usada en éste proyecto, realizado con la herramienta SimulationXpress del software SolidWorks.

Resultado del análisis de elementos finitos a las barras Guías del eje Z:

INFORMACIÓN DEL MODELO	**Medidas de la barra**
	Diametro: 12 mm Longitud: 214 mm
	Propiedades volumétricas
	Masa: 0.189487 kg Volumen: 2.40772e-005 m^3 Densidad: 7870 kg/m^3 Peso: 1.85698 N
	Propiedades del material
	Nombre: 1.2210 (115CrV3) Tipo de modelo: Isotrópico elástico lineal Límite elástico: 370 N/mm^2 Límite de tracción: 670 N/mm^2
	Detalles de la carga
	Entidades: 1 cara(s), 1 plano(s) Referencia: Alzado Tipo: Aplicar fuerza Valores: ---, ---, -1000 N

Figura 23. Modelo: Guía eje Z

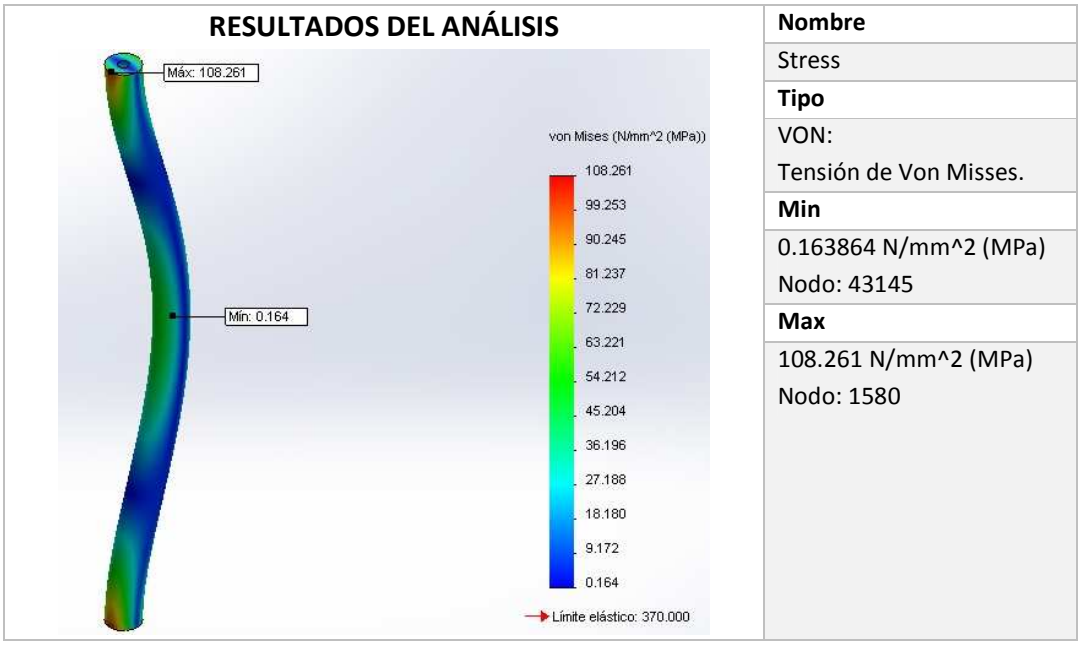


Figura 24. Guía eje Z-SimulationXpress Study-Tensiones-Stress

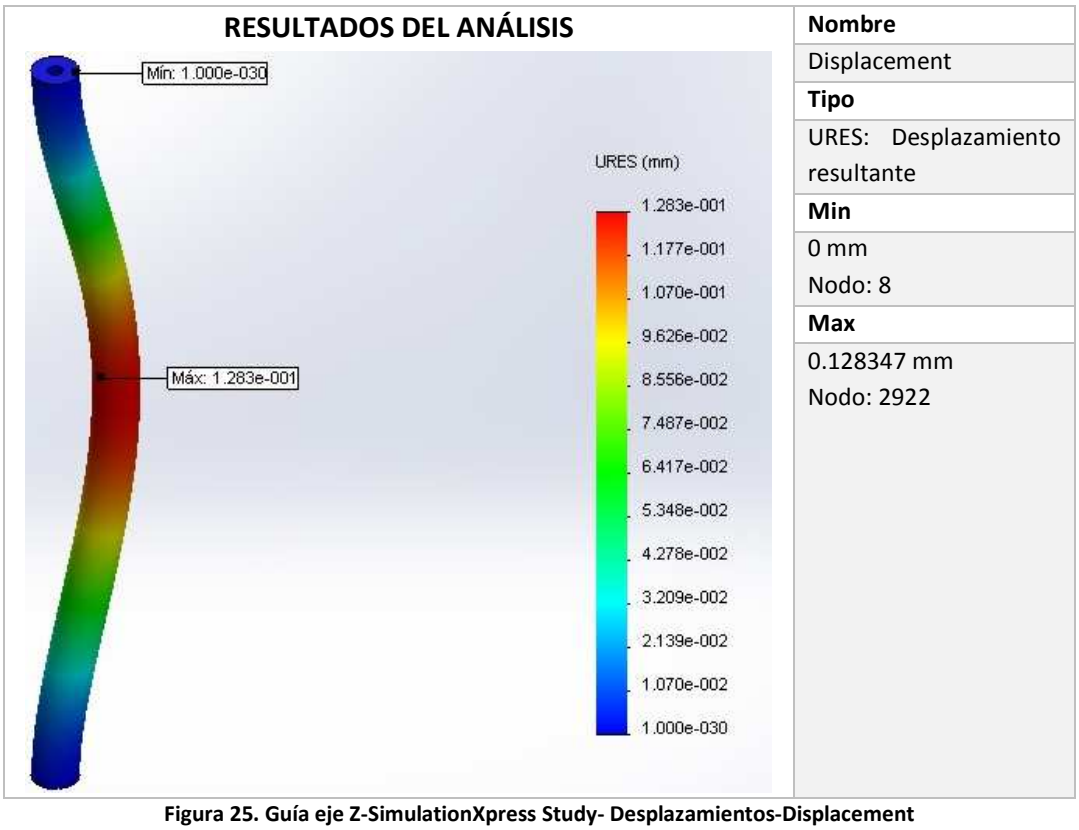


Figura 25. Guía eje Z-SimulationXpress Study- Desplazamientos-Displacement

Resultado del análisis de elementos finitos a las barras Guías del eje X:

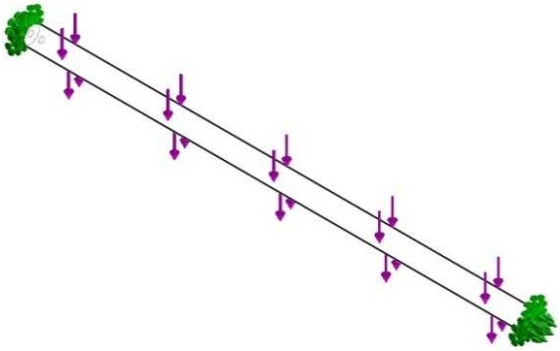
INFORMACIÓN DEL MODELO	
	**Medidas de la barra**
	Diametro: 16 mm Longitud: 457 mm
	Propiedades volumétricas
	Masa: 0.720047 kg Volumen: 9.14926e-005 m^3 Densidad: 7870 kg/m^3 Peso:7.05646 N
	Propiedades del material
	Nombre: 1.2210 (115CrV3) Tipo de modelo: Isotrópico elástico lineal Límite elástico: 370 N/mm^2 Límite de tracción: 670 N/mm^2
	Detalles de la carga
	Entidades: 1 cara(s), 1 plano(s) Referencia: Alzado Tipo: Aplicar fuerza Valores: ---, ---, -1000 N

Figura 26. Modelo: Guía eje X

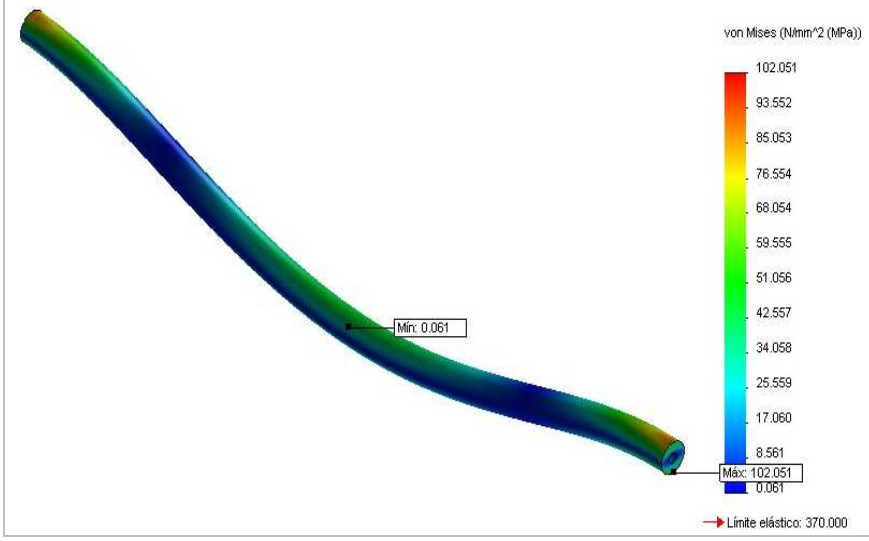
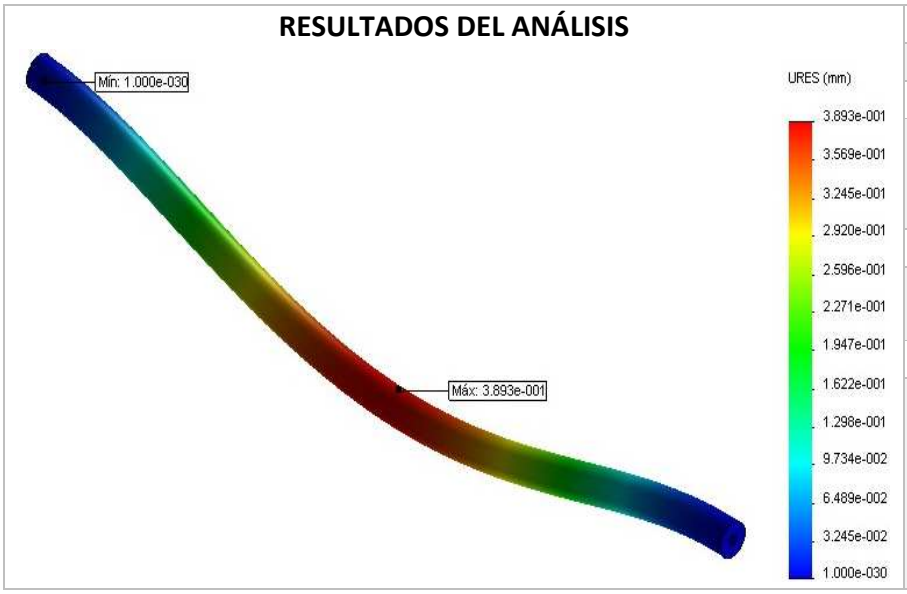
RESULTADOS DEL ANÁLISIS	
	**Nombre**
	Stress
	Tipo
	VON: Tensión de Von Misses.
	Min
	0.0614091 N/mm^2 (MPa) Nodo: 6223
	Max
	102.051 N/mm^2 (MPa) Nodo: 73

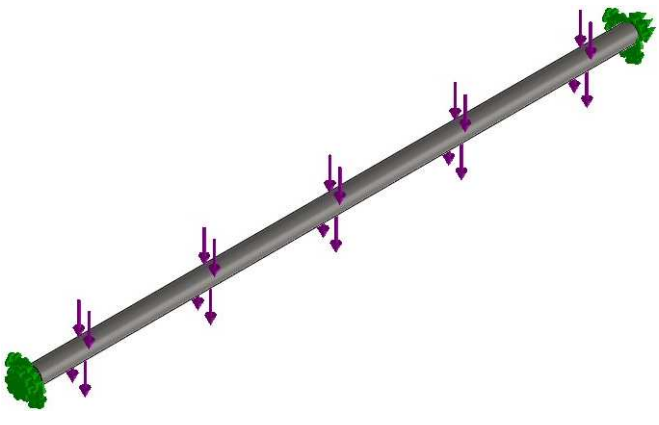
Figura 27. Guía eje X-SimulationXpress Study-Tensiones-Stress



Nombre
Displacement
Tipo
URES: Desplazamiento resultante
Min
0 mm Nodo: 9
Max
0.389341 mm Nodo: 1921

Figura 28. Guía eje X-SimulationXpress Study- Desplazamientos-Displacement

Resultado del análisis de elementos finitos a las barras Guías del eje Y:



Medidas de la barra
Diametro: 16 mm Longitud: 557 mm
Propiedades volumétricas
Masa:0.878283 kg Volumen:0.000111599 m^3 Densidad:7870 kg/m^3 Peso:8.60717 N
Propiedades del material
Nombre: 1.2210 (115CrV3) Tipo de modelo: Isotrópico elástico lineal Límite elástico: 370 N/mm^2 Límite de tracción: 670 N/mm^2
Detalles de la carga
Entidades: 1 cara(s), 1 plano(s) Referencia: Alzado Tipo: Aplicar fuerza Valores: ---, ---, -1000 N

Figura 29. Modelo: Guía eje Y

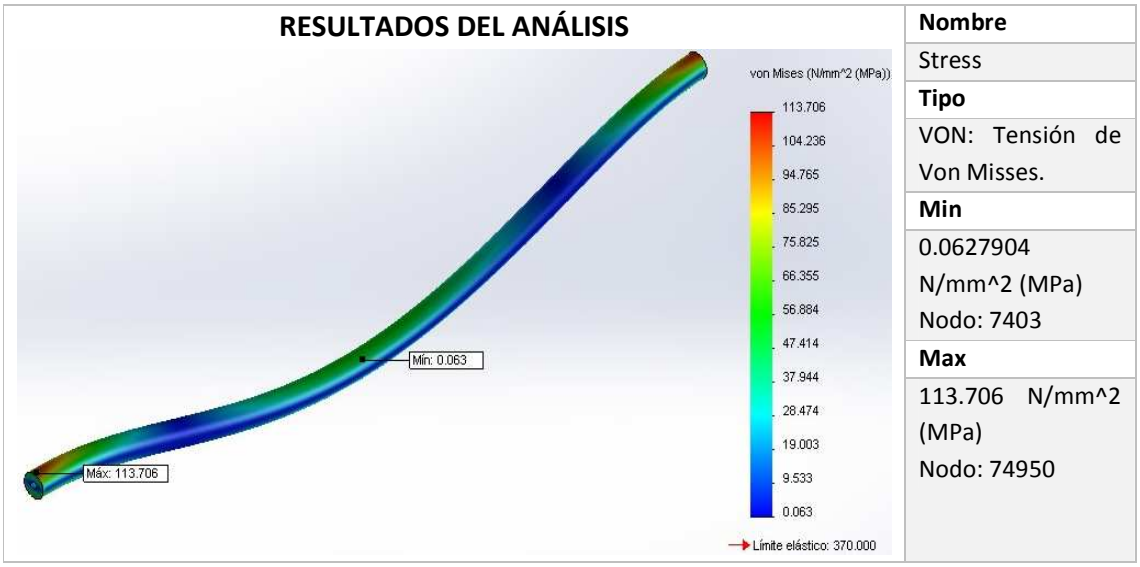


Figura 30. Guía eje Y-SimulationXpress Study-Tensiones-Stress

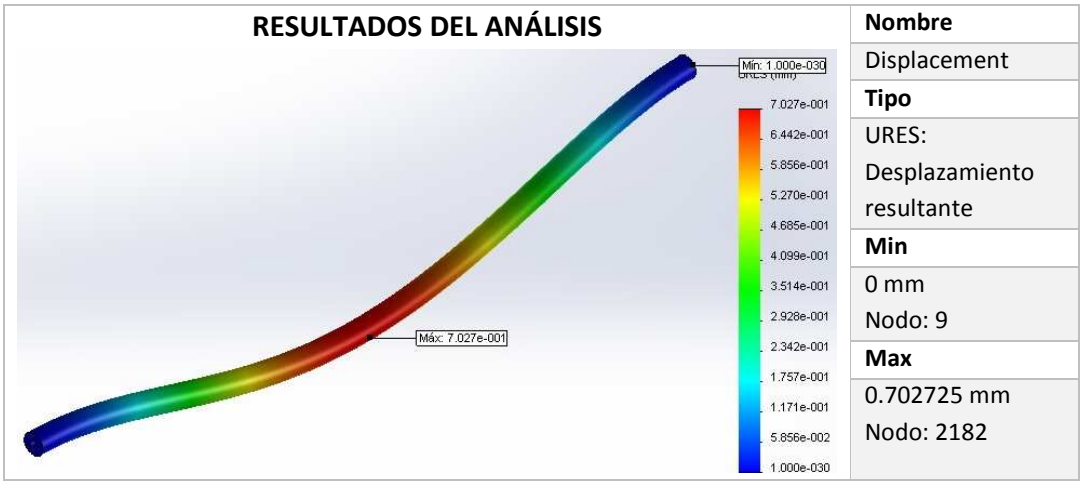


Figura 31. Guía eje Y-SimulationXpress Study- Desplazamientos-Displacement

Como se muestra en el análisis, las cargas generadas sobre las barras no son suficientes para llegar al límite elástico y la deformación del material es insignificante. Cabe recalcar que las cargas a las que se sometió los modelos, son mucho mayores a las cargas reales presentes en cada barra guía. Por tanto se puede decir que las barras están sobredimensionadas para la aplicación, pero esto representa mas una ventaja que un problema, puesto que dado el caso que se presente una carga parecida, la barra o barras serán capaces de soportarlas.

3.3 Tornillo De Potencia

Los tornillos de potencia, son elementos mecánicos que sirven para transformar movimiento rotatorio en movimiento lineal y se utilizan en diversas aplicaciones. Un tornillo de potencia no solo transforma el movimiento, también tiene la capacidad de transformar un pequeño torque en una gran fuerza de arrastre, por ello se hace necesario que el tipo de rosca sea lo suficientemente resistente para soportar dichas cargas.

Para este tipo de aplicaciones se utilizan perfiles de rosca:

- Cuadrada.
- ACME.
- Trapezoidal.

El tipo de rosca cuadrada es la más eficiente para aplicaciones de fuerza, pero es mucho más difícil de fabricar debido a su cara perpendicular. La rosca ACME es mucho más fácil de fabricar debido a su ángulo de 29°, además proporciona al tornillo la capacidad de ejercer fuerza en ambas direcciones, lo cual no se presenta en un tornillo de Rosca trapezoidal que está diseñado para ejercer la fuerza en solo una dirección. La aplicación para la cual es requerido el tornillo de potencia en este proyecto, hace necesario el uso de la fuerza en ambas direcciones pues no solo debe levantar la carga de la herramienta sino que también debe ejercer cierta fuerza al momento de realizar perforaciones. Teniendo en cuenta lo anteriormente explicado, y ya que las cargas a las cuales será sometido dicho tornillo no son tan elevadas, se decidió que el tornillo de potencia tendría un perfil de rosca ACME.

En la Figura 32. Se puede apreciar los diferentes perfiles de rosca y las principales variables dimensionales de las roscas:

- p: paso del tornillo.
- d: diámetro mayor del tornillo.
- dp: diámetro medio o diámetro de paso.

- dr: diámetro menor o de raíz del tornillo.
- α: ángulo de perfil de rosca.

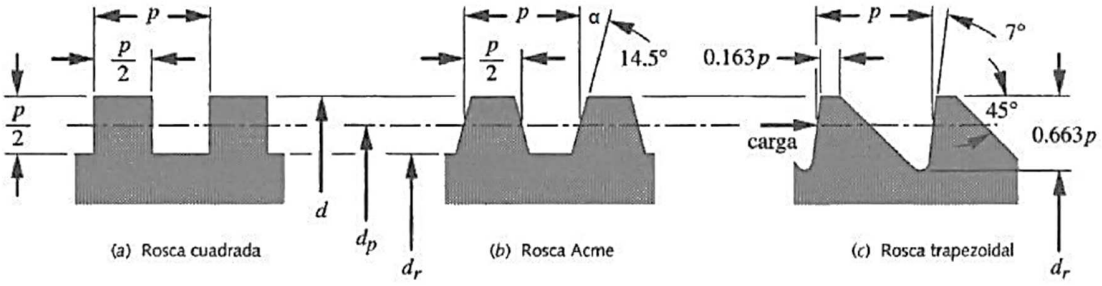


Figura 32. Tipos de rosca para tornillos de potencia

En la se encuentran las principales dimensiones de las Roscas ACME americana estándar.

Tabla 2. Principales dimensiones de roscas ACME americana estándar.

Diámetro mayor *d* (in)	**Diámetro medio d_m (in)**	**Diámetro menor d_r (in)**	**Paso *p* (in)**	**Hilos por pulgada**	**Área de esfuerzo a tracción A_t (in_2)**
0.250	0.219	0.188	0.063	16	0.032
0.313	0.277	0.241	0.071	14	0.053
0.375	0.333	0.292	0.083	12	0.077
0.438	0.396	0.354	0.083	12	0.110
0.500	0.450	0.400	0.100	10	0.142
0.625	0.563	0.500	0.125	8	0.222
0.750	0.667	0.583	0.167	6	0.307
0.875	0.792	0.708	0.167	6	0.442
1.000	0.900	0.800	0.200	5	0.568
1.125	1.025	0.925	0.200	5	0.747
1.250	1.150	1.050	0.200	5	0.950
1.375	1.250	1.125	0.250	4	1.108
1.500	1.375	1.250	0.250	4	1.353
1.750	1.625	1.500	0.250	4	1.918
2.000	1.875	1.750	0.250	4	2.580
2.250	2.083	1.917	0.333	3	3.142
2.500	2.333	2.167	0.333	3	3.976
2.750	2.583	2.417	0.333	3	4.909
3.000	2.750	2.500	0.500	2	5.412
3.500	3.250	3.000	0.500	2	7.670
4.000	3.750	3.500	0.500	2	10.32
4.500	4.250	4.000	0.500	2	13.36
5.000	4.750	4.500	0.500	2	16.80

Debido a que la aplicación del tornillo, más que de fuerza, es de precisión. Se optó por utilizar un tornillo con un paso pequeño, de manera que se obtuviera un avance pequeño con cada vuelta del mismo, dando así al sistema una mayor resolución. Inicialmente se había pensado en un tornillo con un paso que estuviera en un rango de 3mm a 5mm, pero los diámetros para estos valores de paso hacen que el tornillo sea demasiado grande para la aplicación. Por ello, para esta aplicación se seleccionó un tornillo Rosca ACME con un diámetro mayor de 0.625in o 15.875mm Y un paso de 0.125in o 3.175mm (ver en la Tabla 2).

3.3.1 Par de giro

Un tornillo de potencia que se utiliza para elevar y bajar cargas, tiene un par de subida (Ts) y un par de bajada (Tb). El par de subida siempre será mayor, ya que en éste caso la gravedad actúa en contra del movimiento, generando un trabajo negativo, que se suma al trabajo necesario para vencer la fuerza de fricción estática producida por la carga. Mientras que al bajar la carga, la gravedad genera un trabajo positivo que se resta al trabajo generado por la fricción.

De forma general, el par necesario para hacer girar el tornillo es:

$$T = F * D = F * \frac{dp}{2}$$

Donde D es la distancia entre el centro y el diámetro de paso del tornillo. La literatura de (Norton) Propone las siguientes fórmulas para el cálculo de las fuerzas de subida y bajada en un tornillo rosca ACME:

$$Fsu = \frac{W(\mu\pi dp + L\cos\alpha)}{(\pi dp\cos\alpha - \mu L)} \quad Fuerza\ de\ subida$$

$$Fba = \frac{W(\mu\pi dp - L\cos\alpha)}{(\pi dp\cos\alpha + \mu L)} \quad Fuerza\ de\ bajada$$

- W: Carga soportada por el tornillo.
- μ: Coeficiente de fricción entre la tuerca y el tornillo.
- L: distancia de avance de la tuerca con una vuelta del tornillo.

Teniendo así las siguientes ecuaciones para el cálculo de torques:

$$Tsu = \frac{W * dp(\mu\pi dp + L\cos\alpha)}{2(\pi dp\cos\alpha - \mu L)} + Tc$$

$$Tba = \frac{W * dp(\mu\pi dp - L\cos\alpha)}{2(\pi dp\cos\alpha + \mu L)} + Tc$$

(Norton) Incluye el termino Tc, en las fórmulas de torques, que corresponde al torque necesario para hacer girar un collarín de empuje que normalmente se encuentra en aplicaciones donde la tuerca se mantiene en su posición mientras el tornillo es el que se traslada.

$$Tc = \mu_c W\frac{dc}{2}$$

- μ_c: coeficiente de fricción en el collarín.
- dc: diámetro medio del collarín.

La aplicación para la cual se requiere el tornillo de potencia en éste proyecto, requiere que sea la tuerca la que se traslade, por tanto se hace necesario el uso de dos collarines o rodamientos para sujetar el tornillo en su lugar y reducir la fricción de giro. Pero por la forma en que se acoplan los rodamientos al tornillo, toda la carga recae solo sobre el rodamiento inferior, de la misma manera que sucedería con un collarín de empuje. (Ver Figura 33). Por esta razón se empleara el termino Tc de la misma forma.

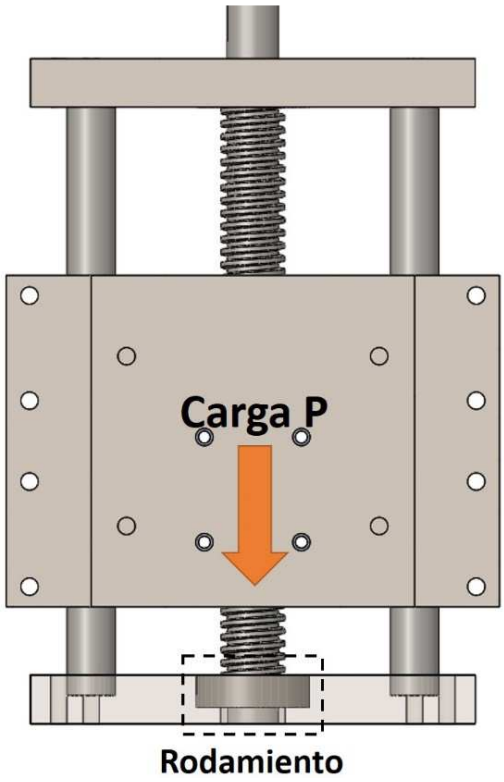


Figura 33. Acción de la carga sobre el rodamiento inferior.

3.3.2 Coeficientes de fricción

A través de los años, los coeficientes de fricción, como otras variables mecánicas, han sido estimados mediante prácticas experimentales. La literatura de (Richard G. Budynas) nos ofrece la siguiente información incluida en la Tabla 3.

Tabla 3. Coeficientes de fricción de pares roscados.

Material del tornillo	**Material de la tuerca**			
	Acero	**Bronce**	**latón**	**Hierro fundido**
Acero, seco	0.15 – 0.25	0.15 – 0.23	0.15 – 0.19	0.15 – 0.25
Acero, aceite para maquina	0.11 – 0.17	0.10 – 0.16	0.10 – 0.15	0.11 – 0.17
Bronce	0.08 – 0.12	0.04 – 0.06	----------	0.06 – 0.09

Bajo las siguientes condiciones:

Tornillo de potencia:

- Material: acero.
- Diámetro mayor (d)= 0.625in – 15.875mm
- Diámetro medio (dp)= 0.563in – 14.3mm
- Avance (L)= 0.125in – 3.175mm
- α=14.5°

Tuerca:

- Material: bronce.
- μ= 0.23.

Carga:

- W= 20N
- Fs= 2

Rodamiento:

- Ref: 6301
- d=37mm
- di=12mm
- dc=25mm
- μ_c=0.01

Conociendo cada una de las constantes mecánicas que hacen parte del sistema, se procede a calcular el par de subida y de bajada para poder dimensionar el motor del eje Z.

Par de subida:

$$Tsu = \frac{2 * 20(0.0143)(\pi(0.23)(0.0143) + (0.003175)\cos(14.5))}{2\left(\pi(0.0143)\cos(14.5) - (0.23)(0.003175)\right)} + (0.01)(20)\frac{0.025}{2}$$

$$= 0.0905$$

$$Tsu = \frac{100}{9.81}(0.0905)[Kg.cm$$

$$Tsu = 0.9221[Kg.cm]$$

Par de bajada:

$$Tba = \frac{2 * 20(0.0143)(\pi(0.23)(0.0143) - (0.003175)\cos(14.5))}{2\left(\pi(0.0143)\cos(14.5) + (0.23)(0.003175)\right)} + (0.01)(20)\frac{0.025}{2}$$

$$= 0.0486$$

$$Tba = \frac{100}{9.81}(0.0486)[Kg.cm$$

$$Tba = 0.4949[Kg.cm]$$

Como Tsu es mayor que Tba, será tomado como valor de torque de referencia para seleccionar el actuador.

3.4 Tornillos De Bolas

Los tornillos de bolas realizan el mismo trabajo que los tornillos de potencia, pero de una manera más eficiente, puesto que utilizan un tren de bolas re-circulantes que permiten la disminución del coeficiente de fricción de la tuerca con el mismo y tienen una capacidad de carga mucho más elevada. Obteniendo así un movimiento más suave, disminuyendo la potencia necesaria para mover la misma carga y disminuyendo las dimensiones necesarias en el tornillo.

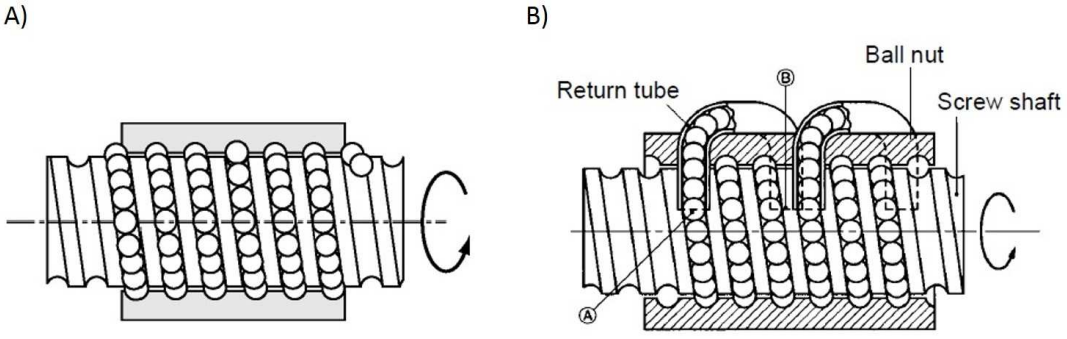


Figura 34. A) tornillo de bolas sin recirculación. B) tornillo de bolas con recirculación.

Como se puede observar en la Figura 34, en el inciso A. se tiene un tornillo de bolas sin tubos de retorno. Al girar el tornillo, las bolas giran y se trasladan en el interior de la rosca disminuyendo la fricción, pero a determinado número de vueltas del tornillo las bolas se salen y se pierden. Por el contrario en el inciso B. se puede observar como las bolas al girar y trasladarse dentro de la rosca llegan al punto A, donde por medio de un tubo de retorno viajan al punto B, permitiendo que las bolas giren y se trasladen continuamente sin que se salgan de la rosca. A esto se le llama sistema de recirculación, de los cuales existen gran variedad. El tipo de recirculación de la tuerca queda a opción de la marca diseñadora.

Los tornillos de bolas no tienen auto-bloqueo debido a su coeficiente de fricción tan bajo, por ello se hace necesario el uso de un dispositivo de frenado para mantener la carga en una posición estática, si el tornillo trabaja en forma vertical. Cuando el tornillo trabaja en forma horizontal no es necesario tal sistema de sujeción puesto que no existe una carga axial que provoque el movimiento del mismo.

La rosca en un tornillo de bolas no obedece el estándar de tornillos de potencia, es decir, la rosca es modificada de manera que se ajuste a forma esférica de las bolas, tal y como se muestra en la Figura 35.

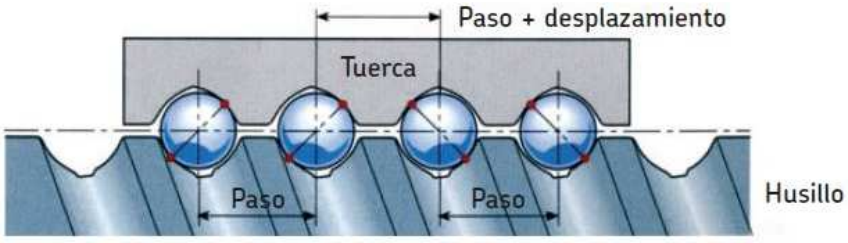


Figura 35. Forma de rosca y esquema de contactos entre un tornillo de bolas y la tuerca.

La forma del perfil de la rosca no es completamente esférica, puesto que necesita canales para mantener lubricado el contacto entre el tornillo, las bolas y la tuerca. Además, como sistema de reducción de juego, la forma de la rosca tanto en el tornillo como en la tuerca hace que los puntos de contacto entre las bolas generen fuerzas que, en el caso de la figura anterior, aprieten las dos mitades de la tuerca. A esta fuerza se le llama precarga.

Las tuercas precargadas tienen una deformación elástica mucho menor que las tuercas sin precarga y se utilizan para aplicaciones de precisión.

3.4.1 Cálculos para la selección

El manual de (Grupo SKF, Diciembre de 2003) proporciona las siguientes fórmulas para la selección de husillos.

Capacidad de carga dinámica

$$L_{10} = \left(\frac{C_a}{F_m}\right)^3$$

Capacidad de carga dinámica requerida

$$C_{req} = F_m (L_{10})^{1/3}{}_{req}$$

Donde:

- L_{10} = Vida [millones de revoluciones].
- C_{req} = Capacidad de carga dinámica requerida [N].
- F_m = Carga media cubica.[N]

Carga media cubica

$$F_m = \frac{F_{min} + 2F_{max}}{3}$$

Donde:

- F_{min} = Carga mínima.
- F_{max} = Carga máxima.

Velocidad critica del tornillo (sin factor de seguridad)

$$n_\sigma = 49 * 10^6 \frac{f_1 d_2}{l^2}$$

Donde:

- n_σ = Velocidad critica [rpm].

- d_2 = Diámetro primitivo [mm].
- l = longitud libre, o distancia entre los 2 rodamientos de apoyo.
- f_1 = Factor de corrección de montaje
- 0.9 ●●——— fijo-libre
- 3.8 ●●———● fijo-soportado
- 5.6 ●●———●● fijo-fijo

Eficiencia teórica

Directa

$$n = \frac{1}{1 + \frac{\pi d_0}{P_h}\mu}$$

Indirecta

$$n' = 2 - \frac{1}{n}$$

Donde:

- μ = Varía entre 0.0065 y 0.006 según el tipo de tornillo.
- d_0 = Diámetro nominal del tornillo [mm].
- P_h = Paso [mm].

Eficiencia práctica

$$n_p = 0.9n$$

Torque de entrada en estado de reposo

$$T = \frac{FP_h}{2000\,\pi\,n_p}$$

Donde:

- T = Torque de entrada [Nm].
- F = Máxima carga del ciclo [N].
- P_h = Paso [mm].
- n_p = Eficiencia práctica.

Torque nominal del motor cuando acelera (tornillo horizontal)

$$T_t = T_f + T_{pr} + \frac{P_h[F + m_L\ \mu_f\ g]}{2000\ \pi\ n_p}\dot{\omega}\sum I$$

Donde:

- T_t = Torque nominal [Nm].
- T_f = Torque por fricción en rodamientos, motor, sellos, etc. [Nm].
- T_{pr} = Torque de Precarga [Nm].
- μ_f = Coeficiente de fricción
- $\dot{\omega}$ = Aceleración angular [rad/s^2].
- m_L = Masa de la carga [kg].
- g = Aceleración de la gravedad [9.8 m/s^2].]

Torque de frenado nominal cuando desacelera

$$T'_t = T_f + T_{pr} + \frac{P_h\ n'[F + m_L\ \mu_f\ g]}{2000\ \pi\ n_p}\dot{\omega}\sum I$$

Donde:

$$\sum I = I_M + I_L + I_S\ l\ 10^{-9}$$

$$I_L = m_L\left(\frac{P_h}{2\ \pi}\right)^2\ 10^{-6}$$

Donde:

- $\mathtt{I}_M$ = inercia del motor [kgm^2].
- $\mathtt{I}_S$ = Inercia del tornillo por metro [kgmm^2/m].
- l = Longitud del tornillo [mm].

Dichas formulas se utilizaron para seleccionar un tornillo de bolas que cumpliera con la necesidad de éste proyecto. Inicialmente se estimó un tiempo de vida de 8 años de trabajo continuo a una velocidad de 250 RPM, moviendo una carga máxima de aproximadamente 20 kg.

$$L_{10} = 8\ años\ \frac{365\ dias}{1\ años}\ \frac{24\ horas}{1\ dias}\ \frac{60\ min}{1\ horas}\ 250\ rev/min$$

$$L_{10} = 1051.2\ [millones\ de\ rev]$$

Ya que la carga es constante, el resultado de F_m = 20 kg.

$$C_{req} = (20 * 9.81)(1051.2)^{1/3}{}_{req}$$

$$C_{req} = 1.99\ [kN]$$

La referencia de tornillo SH 10 x 3 R puede ser utilizada. Sus características son:

- Husillo miniatura SH 10 x 3 R
- Diámetro = 10 mm.
- Paso = 3 mm.
- Coeficiente de carga dinámica = 2.3 kN.
- Inercia = 5.1 kg mm^2/m

Durante la gestión de compra, se determinó que el precio de dicho tornillo es demasiado elevado. Teniendo en cuenta que son 2 tornillos de bolas de diferente longitud, que en total suman aproximadamente 1m, se optó por buscar una solución más económica. Dicha búsqueda dio como resultado un tornillo de bolas de fabricación china, cuya referencia es RM-1605. Dicho tornillo tiene un valor de no más de la mitad del precio del tornillo calculado inicialmente. La referencia RM-1605 es bastante común en el mercado, por tanto es bastante asequible.

Sus principales características son:

- Husillo laminado RM-1605 C7
- Diámetro = 16 mm.
- Paso = 5 mm.
- Coeficiente de carga dinámica = 7.6 kN.

Se realizaron los cálculos para ésta nueva referencia, obteniendo lo siguiente.

$$L_{10} = \left(\frac{2300}{20 * 9.81}\right)^3$$

$$L_{10} = 1611\ [millones\ de\ rev]$$

Con la nueva referencia el tiempo de vida calculado se incrementa en aproximadamente 560 millones de revoluciones, lo cual se traduce en aproximadamente 4.2 años más de trabajo continuo a una velocidad de 250 rpm.

Debido a la procedencia del tornillo de bolas, existe una falta de información en cuanto a sus características, por tanto se realizó una comparación con las referencias de tornillos del manual de SKF para rellenar tales espacios y obtener una aproximación. La referencia más parecida es SD 16 x 5 R y sus características son:

- Husillo miniatura SD 16 x 5 R
- Diámetro nominal = 16 mm.
- Paso = 5 mm.
- Coeficiente de carga dinámica = 7.6 kN.
- u = 0.006.

Eficiencia directa

$$n = \frac{1}{1 + (0.006)\frac{16\,\pi}{5}}$$

$$n = 0.9431$$

Eficiencia Indirecta

$$n' = 2 - \frac{1}{0.9431}$$

$$n' = 0.9396$$

Eficiencia práctica

$$n_p = 0.9(0.9431)$$

$$n_p = 0.8488$$

Torque de entrada en estado de reposo

$$T = \frac{(20 * 9.81)(5)}{2000\,\pi\,(0.8488)}$$

$$T = 0.184\ [Nm]$$

$$T = 1.875\ [kg.cm]$$

Inercias

$$I_L = (20)\left(\frac{5}{2\,\pi}\right)^2 10^{-6}$$

$$I_L = 12.6651\ 10^{-6}\ Kg\ mm^2$$

$$I_s = \frac{5.1}{0.45}$$

$$I_s = 11.3\ Kg\ mm^2$$

$$\sum I = I_M + I_L + I_S\ l\ 10^{-9}$$

Torque nominal del motor cuando acelera (tornillo horizontal)

$$T_t = T_f + T_{pr} + \frac{P_h[F + m_L\,\mu_f\,g]}{2000\,\pi\,n_p}\dot{\omega}\sum I$$

Torque de frenado nominal cuando desacelera

$$T'_t = T_f + T_{pr} + \frac{P_h\,n'[F + m_L\,\mu_f\,g]}{2000\,\pi\,n_p}\dot{\omega}\sum I$$

Los torques nominales de aceleración y deceleración del motor no fueron calculados por falta de datos necesarios, datos tales como las inercias de los motores. Esto no representa un problema para la selección del motor, puesto que los movimientos que realizara la maquina no son demasiado bruscos, por tanto el torque máximo generado en los motores es el torque de entrada en estado de reposo.

Para éste tornillo se utilizaron rodamientos rígidos de bolas de la referencia UC-204, por las dimensiones del tornillo, con soportes de brida de la referencia UCFL-201.

Figura 36. Rodamiento UC-204 con soporte de brida UCFL-201

Generalmente en éste tipo de aplicaciones se utilizan rodamientos para cargas axiales, pero ya que las cargas axiales no son altas, se puede utilizar rodamientos para cargas radiales. Se seleccionó la referencia anteriormente descrita, por su bajo costo y por la facilidad de ensamble que ofrece el soporte de brida.

3.5 Actuadores

Para este tipo de aplicaciones, la selección de los actuadores se realiza a partir de las cargas que debe desplazar dicho actuador. Para el caso particular de éste proyecto, las cargas que debe manejar el actuador son los torques generados en los tornillos de bolas o tornillos de rosca Acme. Debido a las bajas cargas presentadas en los cálculos realizados en los capítulos 3.3 y 3.4, los actuadores fueron seleccionados a partir de valores de velocidad y numero de pulsos por vuelta en el encoder. De esta manera se puede asegurar que el sistema tiene una buena resolución de encoder y una velocidad lo suficientemente rápida para trabajar.

Figura 37. Actuador pololu

Los motores seleccionados para esta aplicación, son de la marca Pololu y tienen una velocidad máxima (en vacío) de aproximadamente 10.500 RPM. La relación de reducción de la caja de engranajes es de 30:1, es decir que la velocidad real de salida es de 350 RPM. Dicho valor en la reducción de la velocidad representa un mejor control del actuador.

El encoder que posee es de 64 pulsos por vuelta acoplado al eje del motor DC, es decir que por cada vuelta de la caja reductora, en el encoder se registran 1920 pulsos. Esto le da al sistema mucha mas resolución para el control de los actuadores.

3.6 Fabricación y montaje

La estructura mecánica utilizada en éste proyecto, está basada en la arquitectura de una fresadora de puente móvil; la cual es apropiada para abarcar grandes áreas de trabajo, que a fin de cuentas es uno de los objetivos de éste proyecto. La arquitectura tipo puente usada en esta aplicación, consta de tres grados de libertad proporcionándole a la maquina la capacidad de moverse a cualquier punto en el espacio.

El mecanismo de movimiento es tipo serie y de configuración cartesiana, por lo que se entiende que cada parte móvil de la maquina se encuentra acoplada directamente sobre otra, el mecanismo de movimiento en la dirección del eje Z se encuentra acoplado directamente al mecanismo de movimiento del eje X y éste último se mueve sobre todo el eje Y.

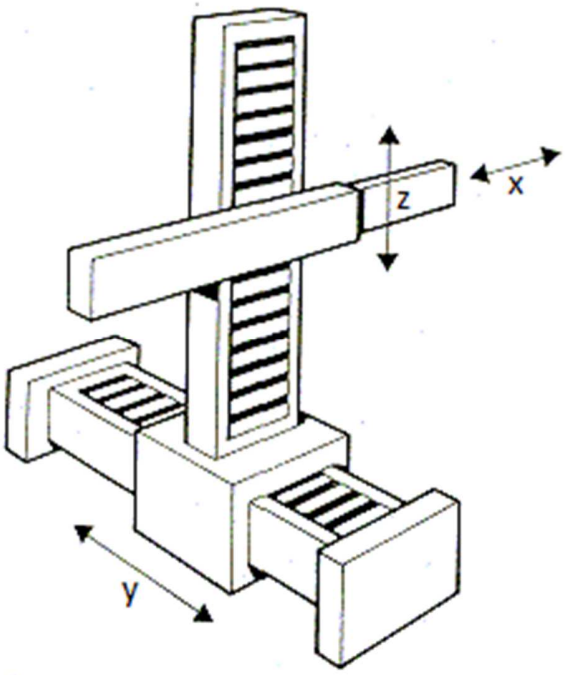


Figura 38. Mecanismo serie de configuración cartesiana.

Cada uno de los ejes de movimiento, lo conforman un sistema de guiado con rodamientos lineales (ver capítulo 3.1) que haría las veces de soporte para las diferentes estructuras móviles, y a su vez un sistema de tornillos y tuercas (ver capítulo 3.3 y 3.4) para transformar el movimiento rotatorio de los actuadores a un movimiento de traslación lineal.

La estructura de la maquina se fabricó en láminas de aluminio recicladas. Las cargas soportadas por la maquina no son muy altas y varían mucho según el material de la estructura, por lo cual esta pudo ser fabricada en otros materiales pero se escogió el aluminio por ser un material poco pesado y con mayor resistencia que otros materiales con una densidad parecida. De esta forma se le proporciona a la estructura mayor robustez y mejor presentación.

Los Ejes de movimiento X e Y poseen el mismo mecanismo de guías lineales y tornillos de bolas, pero el eje Z en lugar de un tornillo de bolas trabaja con un tornillo de potencia de rosca cuadrada.

Para la fabricación, se elaboró un diseño 3D muy básico del conjunto que haría parte del eje Z en el software SolidWorks, donde se hicieron las guías lineales, rodamientos, tornillo de potencia y tuerca, carro de desplazamiento y soportes del efector final; teniendo en cuenta las premisas de diseño. Se decidió fabricar inicialmente la estructura del eje Z para verificar el correcto funcionamiento del mecanismo de movimiento seleccionado.

Luego de la fabricación, se detectaron fallas dimensionales de fabricación en los bujes y el carro de desplazamiento, además hubo fallas en la fabricación del tornillo. Dichas fallas ocasionaban excentricidades en el giro del tornillo y juego en las guías lineales, lo cual se traduce finalmente en un error de posición inaceptable. Por consiguiente se decidió reparar el tornillo y remplazar bujes y carro de desplazamiento.

Las modificaciones realizadas al diseño pueden apreciarse en la Figura 39 y Figura 40.

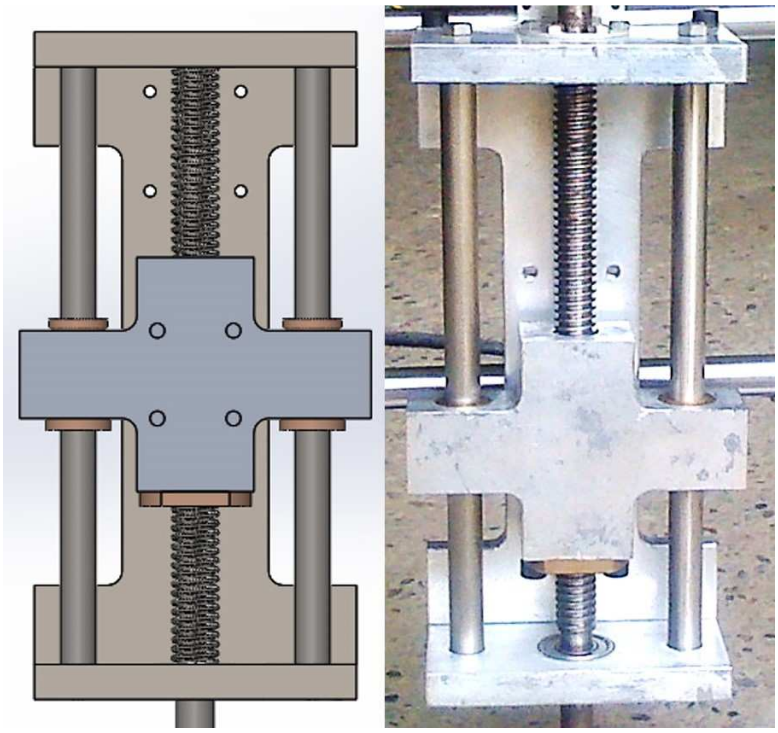


Figura 39. Diseño inicial Eje Z, Ensamble en SolidWorks y Prototipo fabricado.

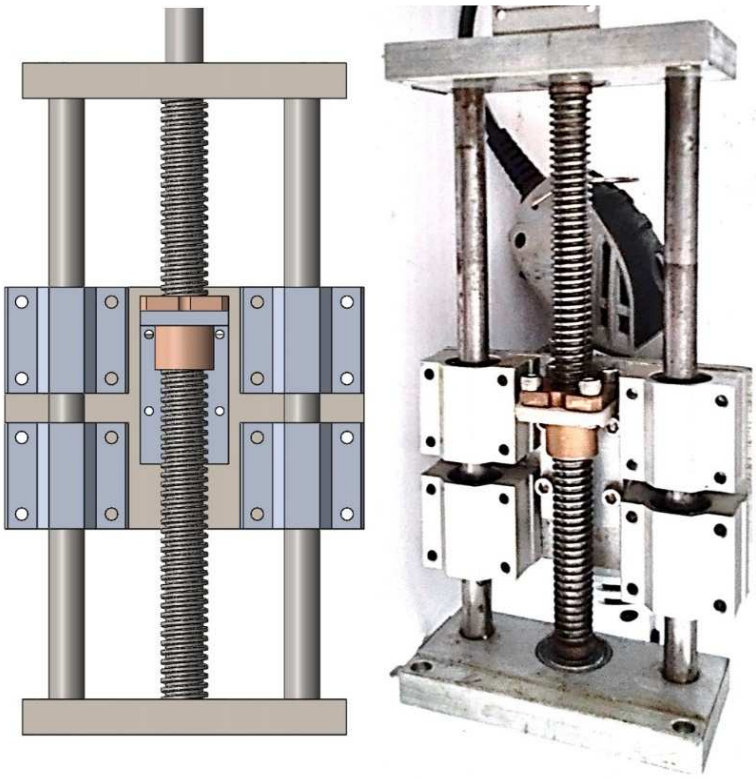


Figura 40. Diseño final Eje Z, Ensamble en SolidWorks y Prototipo fabricado.

Se realizó el mismo procedimiento para el resto de la estructura de la máquina. Para el resto de piezas del mecanismo de movimiento, se tuvo en cuenta la experiencia de la fabricación del eje Z y se optó por comprar tornillos tuercas y rodamientos de bolas.

Para el diseño del puente móvil, se incluyó en la simulación 3D el ensamble inicial del eje Z con masas reales, de esta forma se dimensionaron los soportes del puente, de tal forma que la posición del centro de masa fuera central con respecto a la posición de los rodamientos inferiores. De esta manera las cargas están correctamente balanceadas y así se puede evitar mayor desgaste en rodamientos por centros de masa excéntricos.

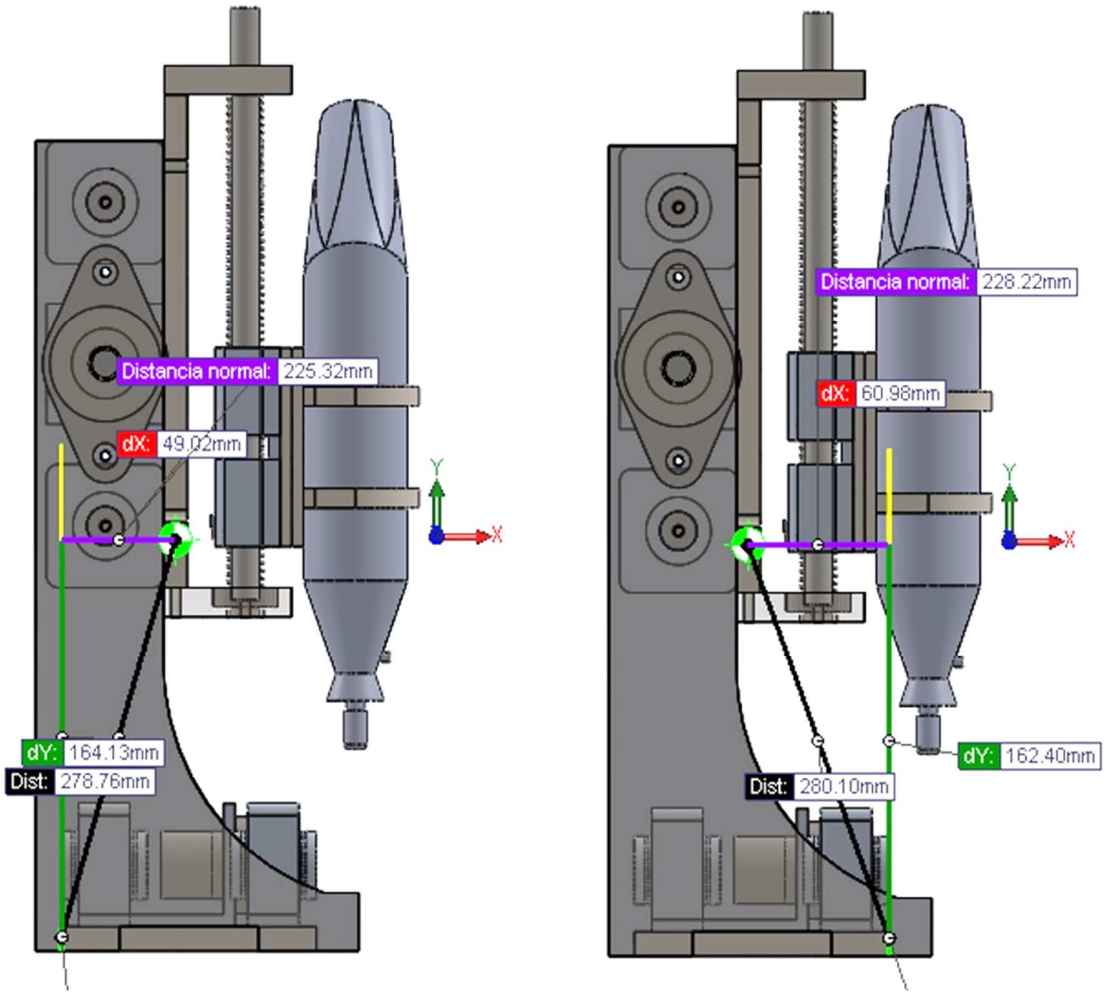


Figura 41. Posición centro de masa del puente.

Las modificaciones realizadas al eje Z, se hicieron luego de la fabricación del puente. Debido a tales modificaciones, hubo una reducción de masa del eje Z. Por tanto la ubicación del centro de masa no es totalmente central. En la Figura 41, se puede apreciar un error de 11,96 mm en el delta X, que corresponde a la distancia entre la posición real del centro de masa y la posición deseada.

Como se puede observar en la Figura 43, la mesa de trabajo de la maquina se encuentra fija y el efector final tiene la movilidad que le proporcionan los tres grados de libertad de la máquina, de esta manera el espacio ocupado por la maquina es aproximadamente un 90% de área de trabajo.

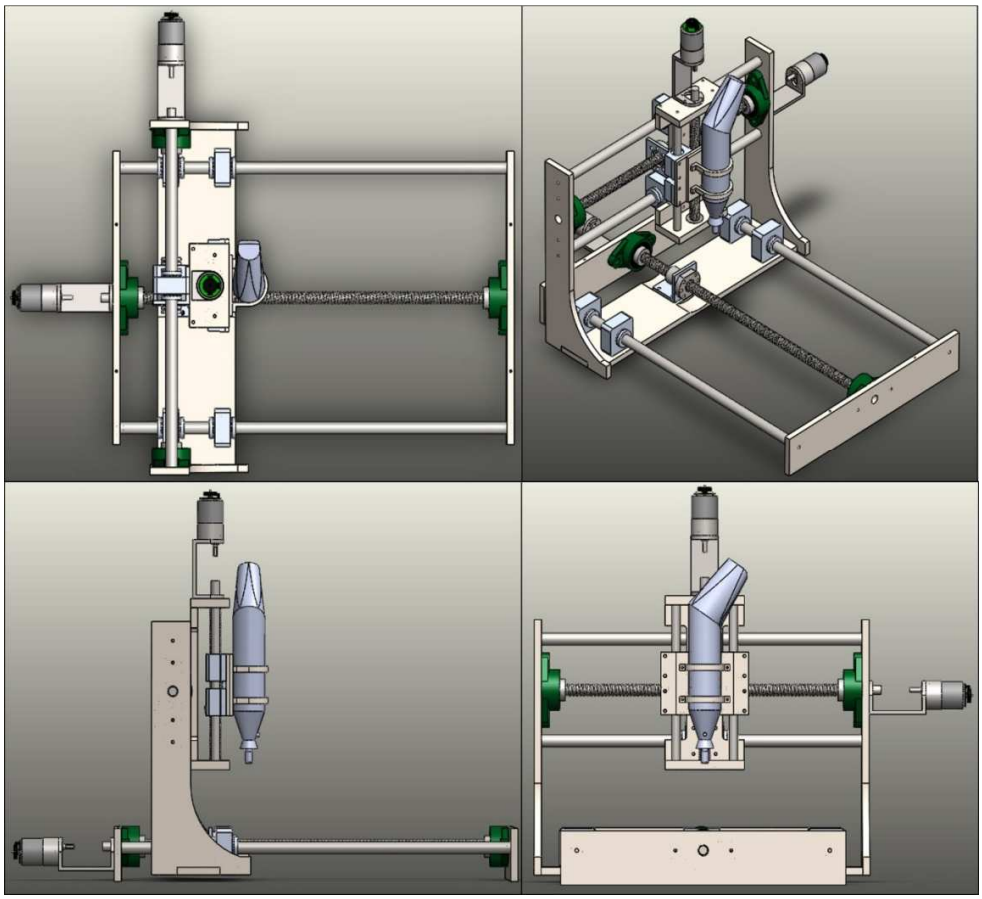


Figura 42. Vistas superior, lateral, frontal e isométrica del diseño final de la máquina.

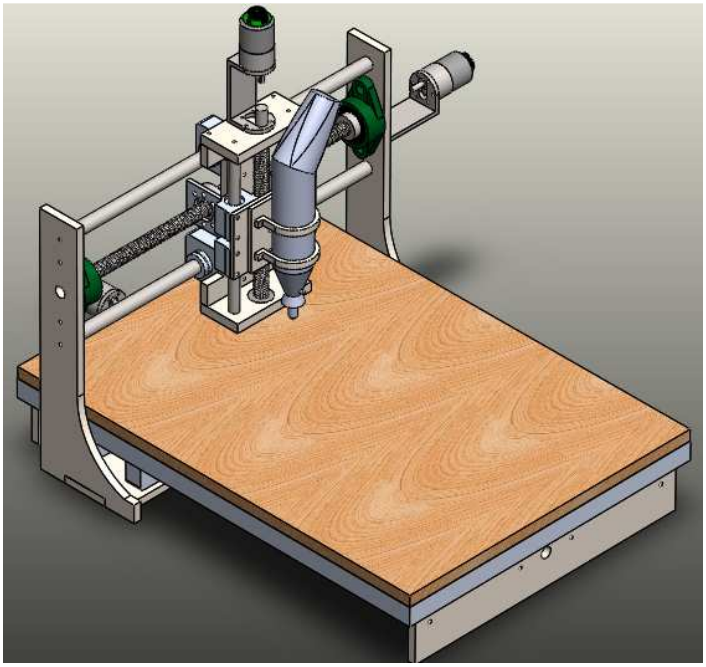


Figura 43. Diseño final de la maquina con mesa de trabajo.

4 SOFTWARE

4.1 Código Gerber

Hoy en día la mayoría de los programas de diseño electrónico, son capaces de crear un archivo Gerber el cual contiene la información necesaria para la fabricación de la placa de circuito impreso. Éste formato de fichero Gerber fue desarrollado por Gerber System Corporation y se estandarizó en la década de los 80´s por la Electronic Industries Association, recibiendo una descripción técnica de RS-274-D. Más tarde se desarrollaría una nueva especificación o formato extendido, descrito como RS-274-X.

Éste formato se diseñó para el control de las impresoras de transparencias (photolite o photoplotter).

El estándar RS-274-X no es más que una nueva versión del estándar RS-274-D que incluye muchos comandos de alto nivel y los controles que permiten al creador del fichero Gerber especificar el photoplotter de una forma muy precisa, mucho más que RS-274-D, que implica pasar una gran cantidad de información crítica por separado del archivo de datos principal.

La información contenida en este formato comprende lo siguiente:

- Formato, unidades y tipo información.
- Aberturas (Formas que se plasman sobre un punto en el plano).
- Definiciones de aberturas específicas (formas de aberturas especiales).
- Declaraciones de control.
- Múltiples capas incrustadas en un único archivo.
- Definiciones de polígonos especiales.

En este capitulo se explicara brevemente el código Gerber, teniendo en cuenta solo las funciones utilizadas en este proyecto.

4.1.1 Cabecera del formato RS-274-X

En la cabecera del RS-274-X se incluye la información fundamental sobre el formato (x,y), supresión de ceros (principio, fin o ninguno) y tipo de coordenadas (absolutos o incrementales), en una sola línea del archivo.

$$\%\ \mathrm{FS} \begin{Bmatrix} \mathrm{L} \\ \mathrm{T} \\ \mathrm{D} \end{Bmatrix} \begin{Bmatrix} \mathrm{A} \\ \mathrm{I} \end{Bmatrix} (\mathrm{Nn})\ (\mathrm{Gn})\ (\mathrm{Xa})\ (\mathrm{Yb})\ (\mathrm{Zc})\ (\mathrm{Dn})\ (\mathrm{Mn}) * \%$$

Donde:

- L = ceros a la izquierda omitidos
- T = ceros finales omitido s
- D = punto decimal explícito (es decir, sin ceros omitidos)
- A = modo de coordenadas absolutas
- I = modo de coordenadas incrementales
- Nn = número de secuencia, donde n es el número de dígitos (poco usado)
- Gn = Código preparatorio de función (poco usado)
- Xa = formato de los datos de entrada (5.5 es max)
- Yb = formato de los datos de entrada
- Zb = formato de los datos de entrada (Z casi nunca se pone)
- Dn = Códigos de dibujo
- Mn = otros código

Por ejemplo:

- La cabecera % FSLAX24Y24 *% indica.
  Declaración: ceros iniciales suprimidos, coordenadas absolutas, formato XY= 2,4.

- La cabecera % FSTIX44Y44 *% indica.
  Declaración: ceros finales suprimidos, coordenadas incrementales, formato XY= 4,4

4.1.2 Unidades embebidas

Los archivos RS-274-X pueden usar coordenadas y dimensiones, tanto de las pistas como de las aberturas, en pulgadas o milímetros. Las declaraciones son:

- % MOIN *% indica pulgadas
- % MOMM *% indica milímetros

4.1.3 Códigos D

Los códigos D, son instrucciones para el photoplotter, que naturalmente incluyen la letra "D". Los tres primeros códigos D controlan el movimiento (X,Y) del plotter.

- D01 (D1): mueve a la posición (X,Y) con el obturador abierto. Es el comando que traza las líneas sobre la PCB.
- D02 (D2): mueve a la posición (X,Y) con el obturador cerrado. Es el comando que cambia la posición de la plumilla, sin trazar líneas.
- D03 (D3): mueve a la posición (X,Y) con el obturador cerrado, y a continuación, abre y cierra el obturador. Conocido como parpadeo en la exposición (flash). Es el comando utilizado para crear las aberturas.

En ésta aplicación, D01 y D02 corresponden a moverse sobre la PCB con el efector final arriba y abajo. D03 corresponde a la instrucción que ubica una forma determinada sobre la pista y además designa la posición de una perforación.

4.1.4 Definiciones básicas de abertura

RS274x incluye varios "estándar", ya que estas aberturas representan más del 90 % de los tipos de flash utilizado:

- Círculo
- Rectángulo
- Elíptico
- Polígono

Todos estos se suponen centrados, y pueden definirse con un agujero redondo o Rectangular si lo desea. Esto es posible, ya que fueron diseñados para los photoplotter,

pero para este proyecto solo es posible crear agujeros redondos, sin posibilidad de cambiar la dimensión del mismo de manera versátil, debido a que seria necesario un sistema para cambiar la proba automáticamente. Por tanto, para éste proyecto solo se tiene en cuenta las dimensiones externas de las definiciones de aberturas.

4.1.4.1 Estándar “Círculos”

% ADD (código) C, ($1) X ($2) X ($3) *%

Donde:

- AD: Descripción de los parámetros de abertura.
- D (código): Código D al que se asigna esta abertura (10-999).
- C: Indica a RS-274-X que se trata de un Círculo.
- $1: Valor (pulgadas o milímetros) del diámetro exterior.
- $2 opcional: Si está presente define el diámetro del agujero
- $3 opcional: Si está presente el agujero será rectangular con un lado $2 y un lado $3.

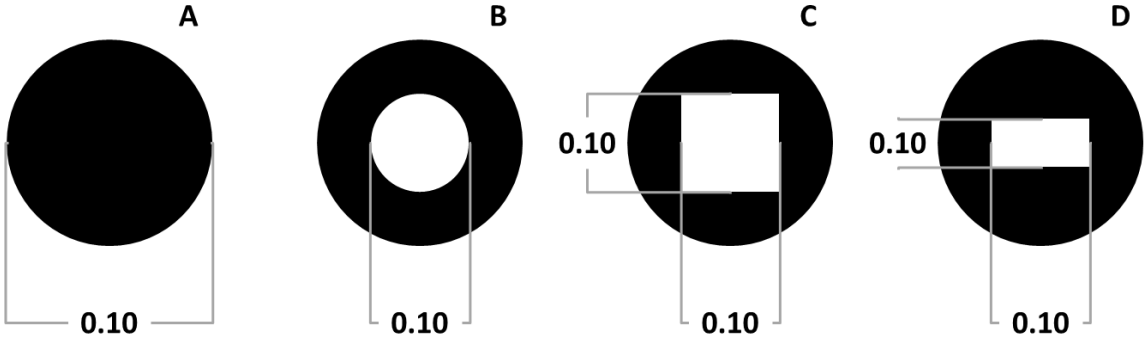


Figura 44. Ejemplos del estándar “circulo”.

Tabla 4. Código y descripción de aberturas A, B, C, D.

	CÓDIGO	DESCRIPCIÓN
A	% ADD21C, .100 *%	Diámetro de 0.10 en D21.
B	% ADD22C, .100 X.050 *%	Diámetro 0.10, con agujero de 0.05 en D22
C	% ADD23C, .100 X.050X.050 *%	Diámetro 0.10, con orificio cuadrado de 0.05 x 0.05 en D23
D	% ADD24C, .100 X.050X.025 *%	Diámetro 0.10, con orificio rectangular de 0.05 x 0.025 en D24

4.1.4.2 Estándar “Rectángulos”

% ADD (código) R, ($1) X ($2) X ($3) X ($4) *%

Donde:

- AD: Descripción de los parámetros de abertura.
- D(código): Código D al que se asigna esta abertura (10-999).
- R: Indica a RS-274-X que se trata de un rectángulo.
- $1: Valor (pulgadas o milímetros) de la longitud en X.
- $2: Valor de la altura en Y.
- $3 opcional: Si está presente define el diámetro del agujero
- $4 opcional: $2 y $3 representan el tamaño de un agujero rectangular.

%ADD22R,0.020X0.040*% Indica que la abertura D22 es un rectángulo Sólido de 0.02x0.04

Figura 45. Ejemplos del estándar rectángulo.

4.1.4.3 Estándar Obround (ovalada o elíptica)

% ADD (código) O, ($1) X ($2) X ($3) X ($4) *%

Donde:

- AD: Descripción de los parámetros de abertura.
- D (código): Código al que se asigna esta abertura (10-999).

- O: Indica que se trata de una forma ovalada o elíptica.
- $1: Valor (pulgadas o milímetros) de la longitud en X.
- $2: Valor de la altura Y.
- $3 opcional: Si está presente define el diámetro del agujero.
- $4 opcional: $2 y $3 representan el tamaño de un agujero rectangular.

%ADD22O,0.020X0.04X0.005X0.010*% Indica una elipse vertical de 0.02 de ancho y 0.04 de alto, con un orificio rectangular de 0.05x0.01.

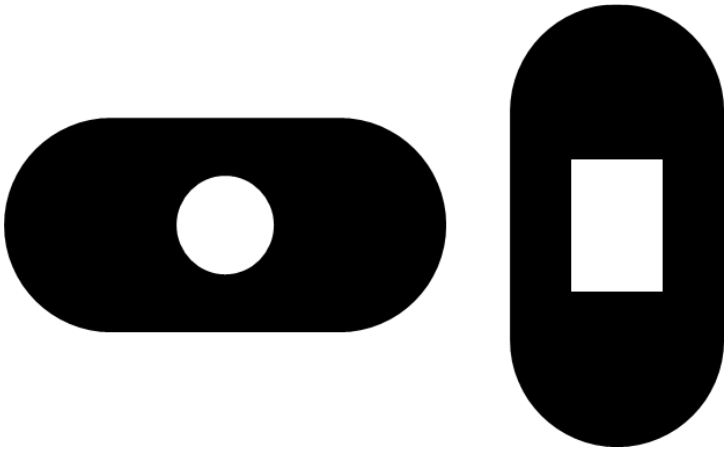


Figura 46. Ejemplos del estándar obround (ovalada o elíptica).

4.2 Software En Matlab

La interfaz hombre maquina se realizó con ayuda del Software MATLAB como herramienta de programación y cálculos integrados, para desarrollar una aplicación que tuviera la capacidad de interpretar el código Gerber y traducirlo a un lenguaje conocido por la máquina.

Inicialmente se hizo un programa muy básico para interpretar las instrucciones en código Gerber y mostrarlas en una gráfica, con el objetivo de verificar la correcta interpretación del código. Ver Figura 47.

Luego de estudiar en detalle cómo funciona el código Gerber, al ver el resultado arrojado por el programa intérprete y al hacer un análisis de cómo debía trabajar la máquina para obtener el resultado deseado, se inició un periodo de análisis para saber cuál sería el mejor método para transformar dicha información en la requerida, puesto que como se muestra en la Figura 47, las trayectorias obtenidas del código Gerber por el interprete, en sí no son útiles si no se utiliza toda la información brindada por el código para generar unas trayectorias nuevas.

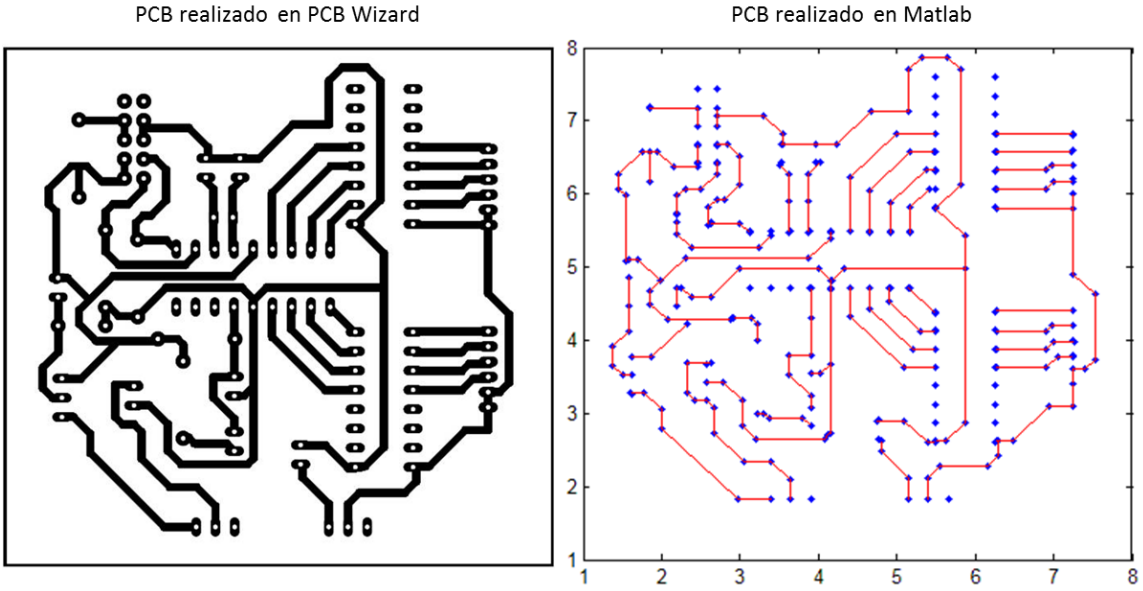


Figura 47. PCB realizado en PCB Wizard y el resultado obtenido del intérprete de Gerber.

Bajo este criterio se tomó la decisión de crear una imagen a partir el código Gerber, para luego procesarla de tal manera que sus contornos sirvieran como trayectoria final para la máquina. En este capítulo se mostrara de forma detallada el funcionamiento del software intérprete y generador de trayectorias.

En la Figura 48 se muestra el diagrama de flujo general del proceso que se lleva a cabo para generar la imagen y luego obtener una trayectoria final.

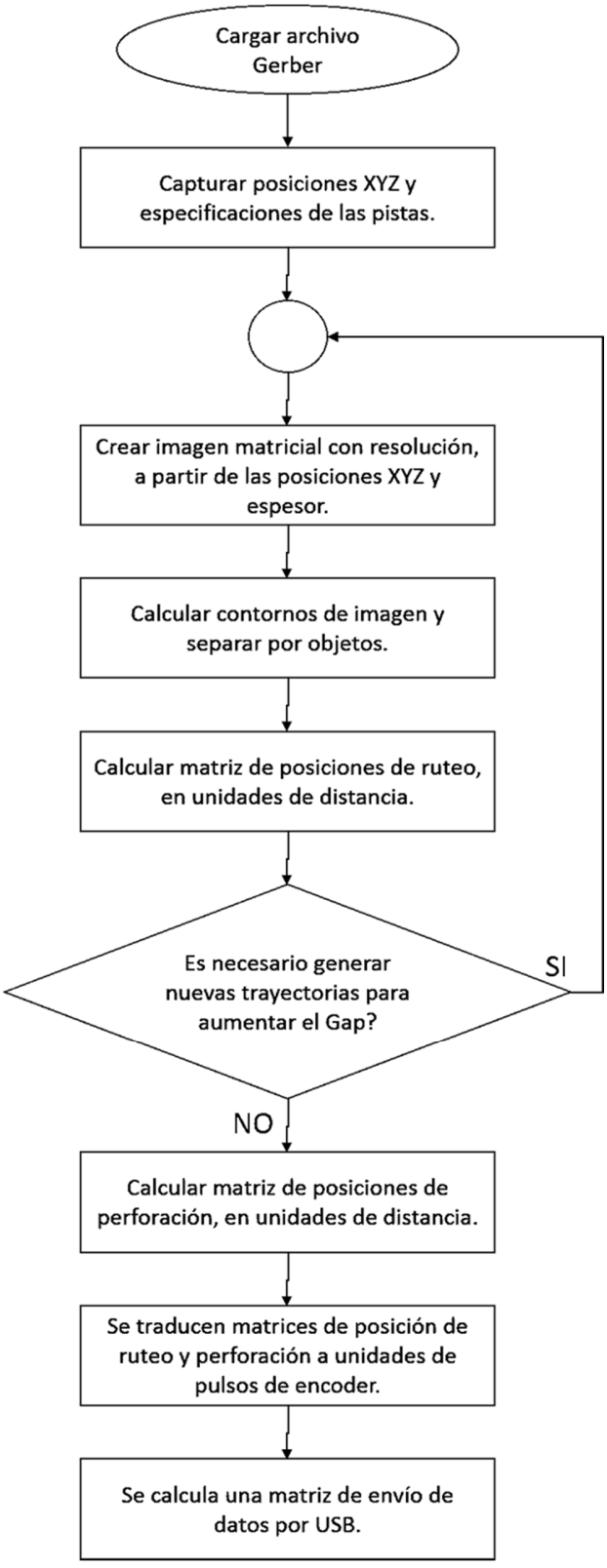


Figura 48.- Diagrama de flujo del proceso para generar datos de la maquina

4.2.1 Interprete del código Gerber

La tarea de interpretar el código Gerber, dentro de la aplicación desarrollada en Matlab, está constituida por varias funciones que trabajan de forma anidada.

El proceso inicia cargando los archivos Gerber principales donde se encuentra almacenada la información principal del circuito impreso y el otro contiene la información del tamaño de la placa. El programa analiza los datos de una sección del código Gerber llamada cabecera q contiene información general sobre el formato del archivo (Ver capítulo 4.1.1). Luego analiza las definiciones de abertura para ver que figuras deben ser generadas en las posiciones de los agujeros (ver capítulo 4.1.4) y además conocer sus dimensiones. Por ultimo captura y genera en vectores diferentes las posiciones (X,Y) contenidas en el Gerber y la acción del efector final definida por la instrucción (D01, D02, D03, etc. Ver capítulo 4.1.3).

El programa define el tamaño de la PCB a partir de los datos del archivo Gerber, luego de esto utiliza los datos de posiciones de ruteo y perforación para generar los vectores de posición (X,Y) y los vectores de acción del efector final. Luego de generar los vectores de posición (X,Y), se convierten a datos con resolución para que la imagen generada sea mucho más fina y se puedan obtener contornos más suaves.

4.2.2 Creación de imágenes

Ya teniendo la información del Gerber, necesaria para la creación de la imagen, el programa genera una imagen en blanco y negro. Inicialmente, el programa sitúa las formas correspondientes a las aberturas en sus respectivas posiciones. Luego, evalúa las coordenadas de las pistas según los vectores (X,Y), determinando si las pistas a generar son horizontales, verticales o con inclinación.

Los valores contenidos en la matriz son “255” y “0” y corresponden a los colores blanco y negro respectivamente. El vector (X,Y) y el vector de códigos “D” (D01, D02 y D03) junto con la resolución, nos indica una posición dentro de la matriz de la imagen.

El algoritmo que genera la imagen de una pista, busca una posición en la matriz que corresponda a la posición donde debe iniciar una pista. Al ubicarse en la posición deseada, una línea de código cambia los valores iniciales de la matriz en el área correspondiente a dicha pista, desde su inicio (x1,y1) hasta su final (x2,y2) con un ancho correspondiente a la cantidad de pixeles designada tanto por el espesor de la pista como por la resolución de la matriz. El resultado obtenido es una figura rectangular, en el caso de las pistas verticales y horizontales.

En el caso de las pistas con un ángulo de inclinación, el algoritmo es más complejo, puesto que depende del ángulo para determinar a qué pixeles cambiara su valor. Para este caso, el algoritmo hace una interpolación entre los puntos inicial y final. De esta manera, determina qué pixeles trazan la línea que cruza por esos puntos. Una vez hallado el pixel, el algoritmo procede a modificar una hilera de pixeles de forma horizontal o vertical, hacia los costados. La cantidad de pixeles se determina por el ángulo de inclinación de la pista.

Al crear la imagen de una pista con inclinación de la forma antes descrita, quedan espacios blancos indeseados, en forma triangular. Esto se corrige modificando los límites de la gráfica y colocando formas circulares en las posiciones determinadas. Tales espacios, también se presentan cuando una pista cambia de dirección, es decir, en las intercepciones. Se cubren los espacios solo colocando círculos.

El resultado final luego de todo el proceso anteriormente descrito es una matriz en blanco y negro que representa las pitas del circuito (ver Figura 49).

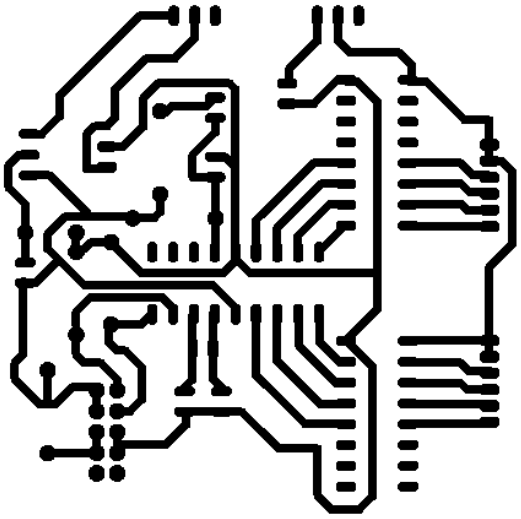


Figura 49. Matriz principal en blanco y negro.

4.2.3 Procesamiento de imágenes

De las imágenes generadas a partir de la información obtenida del código Gerber, se entiende que las figuras de color negro serán las pistas de cobre en la PCB. Pero la placa virgen está recubierta completamente de cobre, por tanto para obtener el mismo diseño, lo que la maquina debe hacer es eliminar el cobre indeseado, es decir, trazar una trayectoria alrededor de la pista. Por esta razón, el programa utiliza funciones propias de Matlab para separar la imagen por objetos, donde cada pista constituye un objeto. Y además almacenar en un vector una serie de posiciones que forman el contorno del objeto. Éstas posiciones pueden enviarse a la maquina como trayectoria final, pero los datos arrojados son muchos, por esta razón se elaboró un algoritmo para vectorizar los datos. Así las líneas rectas tienen, en lo posible, solo un punto inicial y un punto final. Las líneas curvas siguen teniendo más datos pero en cantidades reducidas. La función utilizada en este caso es “Bwboundaries()” de Matlab.

La mayoría de los circuitos poseen una dimensión llamada “Gap” que corresponde a la brecha de aislamiento de una pista, es decir la distancia perpendicular entre la pista y el cobre que la rodea. Tal dimensión puede ser variable, y el diámetro de la fresa siempre es

el mismo, por tanto para lograr que el gap sea variable se hace necesario realizar varios ciclos de fresado en el que se amplíe tal dimensión hasta el valor deseado. Para ejecutar éste proceso se incrementa gradualmente el ancho de las pistas en la matriz principal y se recalcula el contorno, generando así una nueva trayectoria que aumentara el Gap que deja el fresado (ver Figura 50).

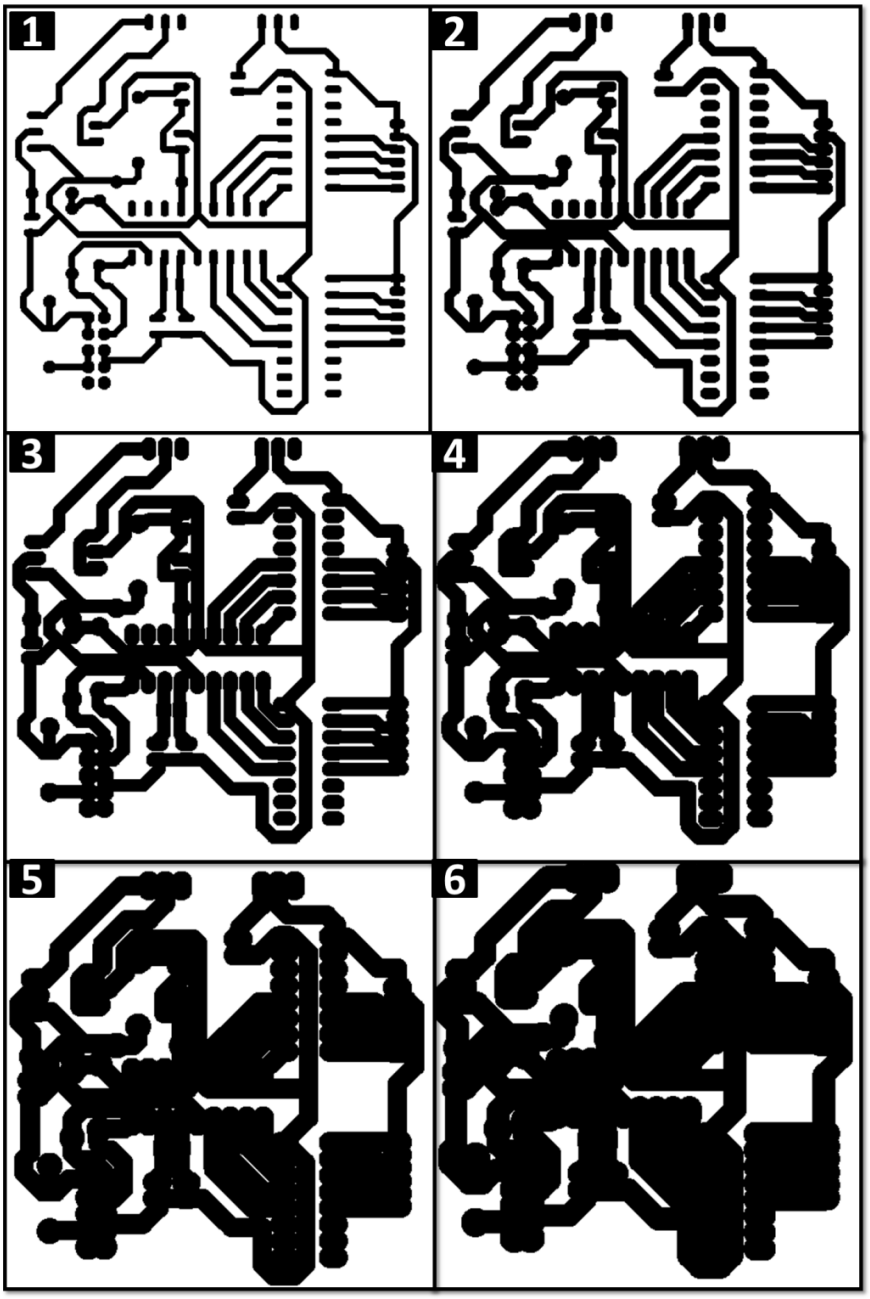


Figura 50. Modificaciones de la matriz principal para ampliar GAP.

En la Figura 51 se puede apreciar varios contornos en color azul, generados uno por uno a partir de las modificaciones de la matriz principal. Los asteriscos rojos representan la posición de un agujero y por tanto de una definición de abertura (ver capítulo 4.1.4).

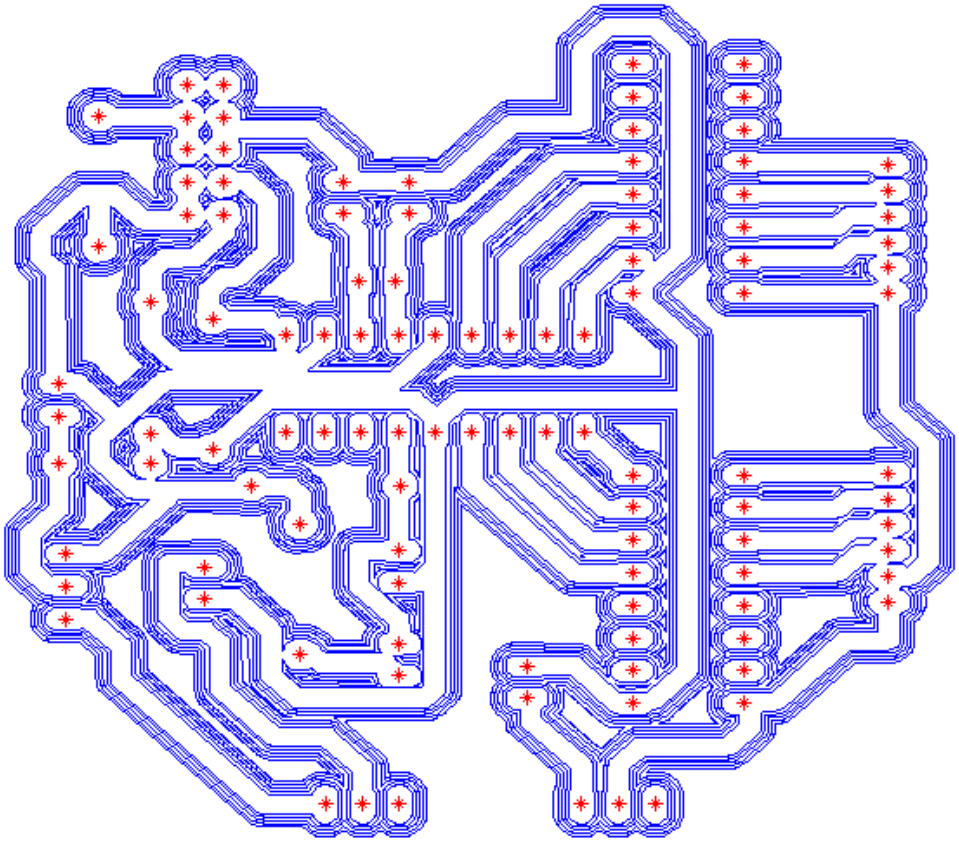


Figura 51. Contornos generados a partir de las modificaciones de la matriz principal.

4.2.4 Matriz de posiciones

Ya teniendo las trayectorias en el plano (X,Y), antes de poder enviar los datos a la máquina, es necesario que la maquina sepa en qué posiciones el efector final debe subir o bajar. Por lo cual se hace necesario incluir los movimientos en el eje Z. para esta tarea se desarrolló un algoritmo que a partir de los códigos de abertura y las posiciones de inicio y final de una pista, define la posición Z y la agrega a la Matriz final de posiciones (X,Y,Z).

4.2.5 Interfaz grafica

Uno de los objetivos de este proyecto es que la interfaz sea intuitiva para el usuario. Por tanto se elaboró un diseño bastante sencillo, donde el usuario tiene una ventana con una barra de menú como cualquier otro software, donde puede manejar opciones de archivo para importar el archivo Gerber al programa, opciones de configuración del software y opciones de copiado para ejecutar el mismo trabajo varias veces sobre una misma placa.

En la ventana principal se proporciona información visual del diseño sobre el cual se trabajara, una tabla donde se muestra las posiciones que el efector final ira tomando mientras la maquina ejecuta su trabajo y un área donde se muestra la posición actual de la máquina. La información de la posición actual proviene de los controladores. Por último, en la parte inferior de la ventana se tienen cuatro botones para un control sencillo de la máquina. Un botón “Rutear” para iniciar el trabajo de fresado, un botón “Perforar” para iniciar el trabajo de perforación, un botón de “Pausa” y un botón de “Detener”. De tal manera que el usuario pueda iniciar, pausar o detener el proceso en el momento que lo necesite (ver Figura 52).

Dentro de las opciones del menú “maquina” se encuentra la opción de “configurar copias de PCB”, “Configurar parámetros de la maquina”, “Comenzar desde línea” y “Control manual”. En la ventana de configuración de copias se puede configurar el tamaño de la placa que se tiene y el número de copias deseado. El programa acomoda automáticamente la posición de las copias y con el botón de vista previa se puede visualizar como quedará la placa con las copias (ver Figura 53).

En la ventana de configuración de parámetros de la maquina se pueden modificar algunas variables del sistema, tales modificaciones solo pueden ser realizadas por los autores del proyecto, por esta razón es necesario ingresar una contraseña para habilitar tal función. En el menú “Modo de ejecución” se puede elegir entre “Modo seguro” y “Modo

avanzado", en modo seguro se tiene acceso a las opciones principales del software. La opción de modo avanzado muestra una ventana para ingresar la contraseña que habilita la opción de configuración de parámetros de la máquina. la contraseña es "UTBCNC2014PCBROUTER".

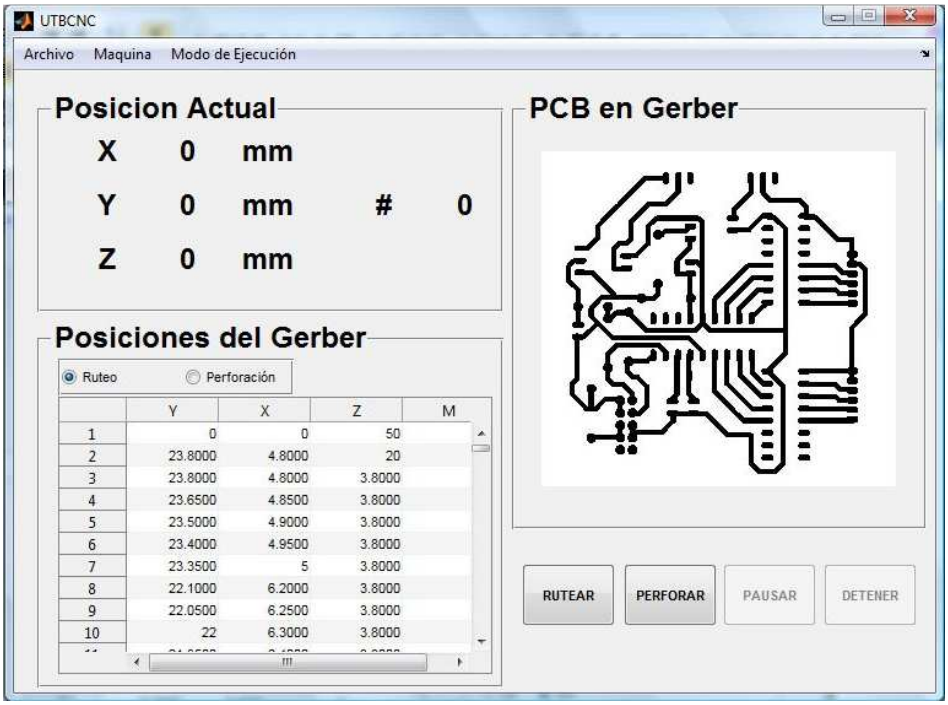


Figura 52. Interfaz gráfica del software.

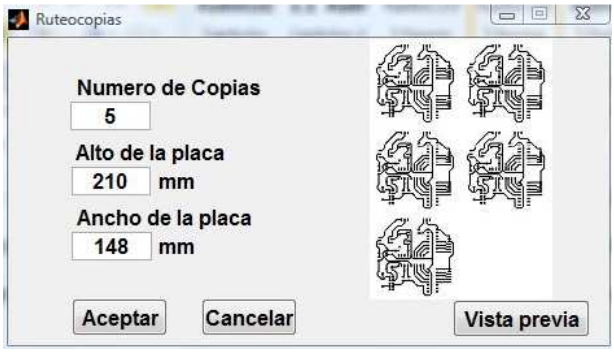


Figura 53. Ventana de configuración de copias.

De los parámetros modificables en la ventana de configuración de la Figura 54, se pueden modificar los parámetros del encoder, motor, tornillos de bolas y rosca ACME, algunos parámetros del software, fresas, etc.

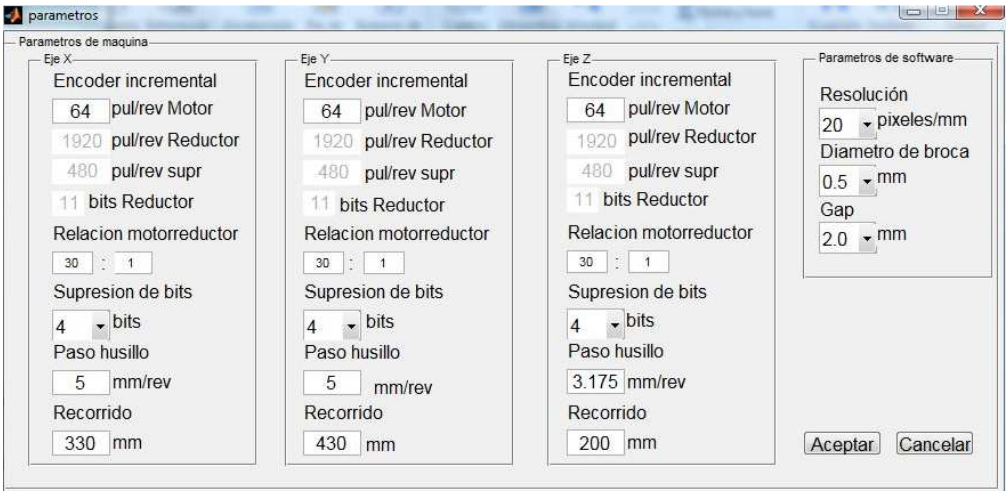


Figura 54. Ventana de parámetros de configuración.

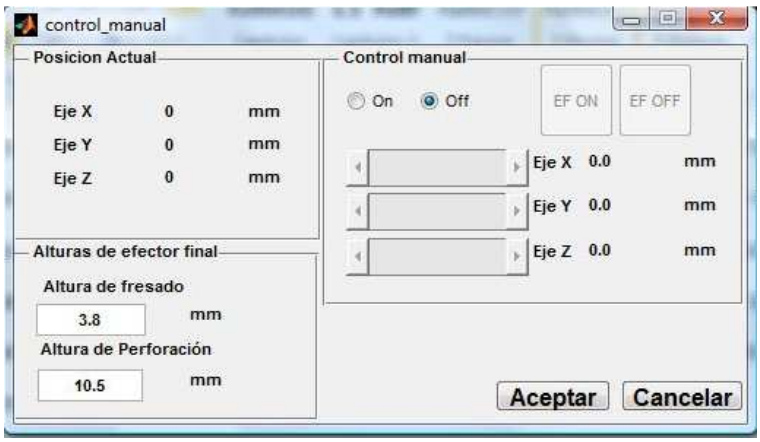


Figura 55. Ventana de control manual.

La ventana de control manual se elaboró con el objetivo de mover la maquina manualmente para calibrar las alturas de trabajo del eje Z con respecto a las tareas de ruteo y perforación. Con esta ventana de control manual se puede controlar todas las funciones móviles de la máquina de forma independiente, por tanto se le dio también la utilidad de calibración de la mesa, la cual requiere que la distancia entre la broca y la mesa sea la misma sobre cualquier punto. La opción de control manual facilita la tarea de realizar tal procedimiento.

La ventana de configuración de línea inicial es útil cuando por alguna razón el usuario ha decidido detener el proceso. Al iniciarlo nuevamente, no tendrá que realizar todo el

trabajo hecho anteriormente, sino que podrá seleccionar un punto cercano al último procesado por la máquina. El número ingresado corresponde a la línea descrita en la tabla de posiciones de la ventana principal. Una vez configurada la posición inicial, la maquina comenzara a trabajar desde dicha posición en la tabla, hasta el fin de la misma. Las posiciones anteriores al punto configurado no serán procesadas por la máquina.

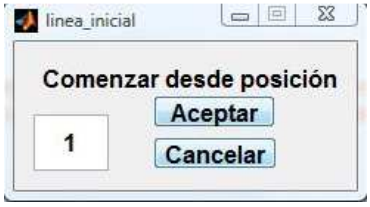


Figura 56. Ventana de configuración de línea inicial.

El procedimiento de uso de cada una de las opciones del software se explica detalladamente en el manual técnico del usuario.

4.2.6 Comunicación USB

El programa en Matlab envía una trama de 64 Bytes hacia el Maestro, donde se envía la información de cada uno de los controladores y efector final. Una vez que se envía la información hacia el Maestro, el programa le pide una trama de datos de la misma longitud para actualizar la aplicación con los datos de la máquina.

Ya obtenida la trama de la máquina, se calcula el error de la posición entre el valor deseado y la posición en ese instante de tiempo. Si este error es menor de cierto valor en los tres ejes, se procede a enviar la siguiente posición a la cual debe ir el efector final.

Éste proceso se repite cada 50 milisegundos aproximadamente y se hace para poder actualizar en el menor tiempo posible la posición de cada uno de los ejes en la interfaz gráfica del programa. Mientras la maquina no llegue a la posición deseada se sigue enviando la misma posición.

5 CONTROLADORES

En las secciones anteriores se hablo acerca del mecanismo de movimiento que corresponde a toda la estructura mecánica de la máquina y la forma de cómo se moverá el efector final, luego se hablo acerca del software en Matlab el cual interpreta el código Gerber y crea los vectores de movimiento del efector final de la máquina para luego enviarlas por el protocolo USB.

El turno ahora es para aquellos dispositivos que se encargan de mover el mecanismo y de llevar al efector final a la posición dada por el software en Matlab. Para ello, como se mencionó en la Figura 12, se implementaron cuatro controladores, uno que trabaja como interfaz de comunicación y los otros tres se encargan de controlar la posición de cada eje de movimiento (X,Y,Z) de los cuales se habla a continuación.

5.1 Maestro

Como se mencionó en la sección 2.3.1, el modulo Maestro se encarga de comunicar el software con la máquina. En las siguientes secciones se detalla el diseño y la implementación del mismo.

5.1.1 Comunicación

La arquitectura software del Maestro es bastante simple, pues éste trabaja solo como una interfaz de comunicación entre el pc y los Esclavos que corresponden a los controladores de cada motor en la máquina. En la Figura 57, se puede apreciar el diagrama de flujo de éste controlador, donde el Maestro comienza su tarea inicializando algunas variables necesarias para la ejecución del programa, por ejemplo algunas variables contadoras para ciclos “for” y vectores de tránsito de información. Luego de finalizar la inicialización de las variables, los puertos de entrada/salida y módulos de hardware, el Maestro procede a

entrar en un ciclo infinito dentro del cual estará siempre al pendiente de si existe o no un dato en el módulo de comunicaciones USB. Si hay alguna información en dicho puerto, la tarea del Maestro es capturar y analizar dicha información.

La información almacenada por el maestro consta de una trama de 64 Bytes, de la cual se habla más detalladamente en la sección 5.1.1.1. De dicha trama se analiza siempre el primer valor, denominado cabecera. Otra tarea del Maestro es la de enviar y recibir información a través del protocolo I2C, del cual se habla más detalladamente en la sección 5.1.1.2.

La información transmitida por el I2C siempre será la contenida en la trama USB de entrada, capturada por medio del módulo USB, pero la cantidad de datos transmitidos varía según el valor de cabecera de la trama, pues el Maestro la descompone y envía solo la información necesaria a cada Esclavo. El tamaño de la trama de datos principal entre el Maestro y los Esclavos para éste proyecto es de 4 Bytes donde el principal dato que se envía es el Set Point de posición de dicho controlador.

Una vez el Maestro envía la trama a cada controlador Esclavo, éste le pide las posiciones actuales para luego armar una trama de retorno y enviarla a la aplicación de Matlab por medio del módulo USB. Con éste sistema de comunicación, se envía y se recibe la información desde el computador hacia cada controlador y viceversa.

Para la comunicación USB, se tomó como punto de partida, el trabajo publicado en Matlab Central “Comunicación entre MATLAB y PIC de MICROCHIP usando puerto USB” (Pool, 2010); en el cual se establece la comunicación USB entre un microcontrolador PIC y una aplicación realizada en Matlab. La parte de la comunicación se adoptó en éste proyecto y lo que se hizo fue modificar el programa para que pudiese trabajar con la comunicación I2C principalmente. En la Figura 57, se puede apreciar el diagrama de flujo del software implementado en el microcontrolador Maestro.

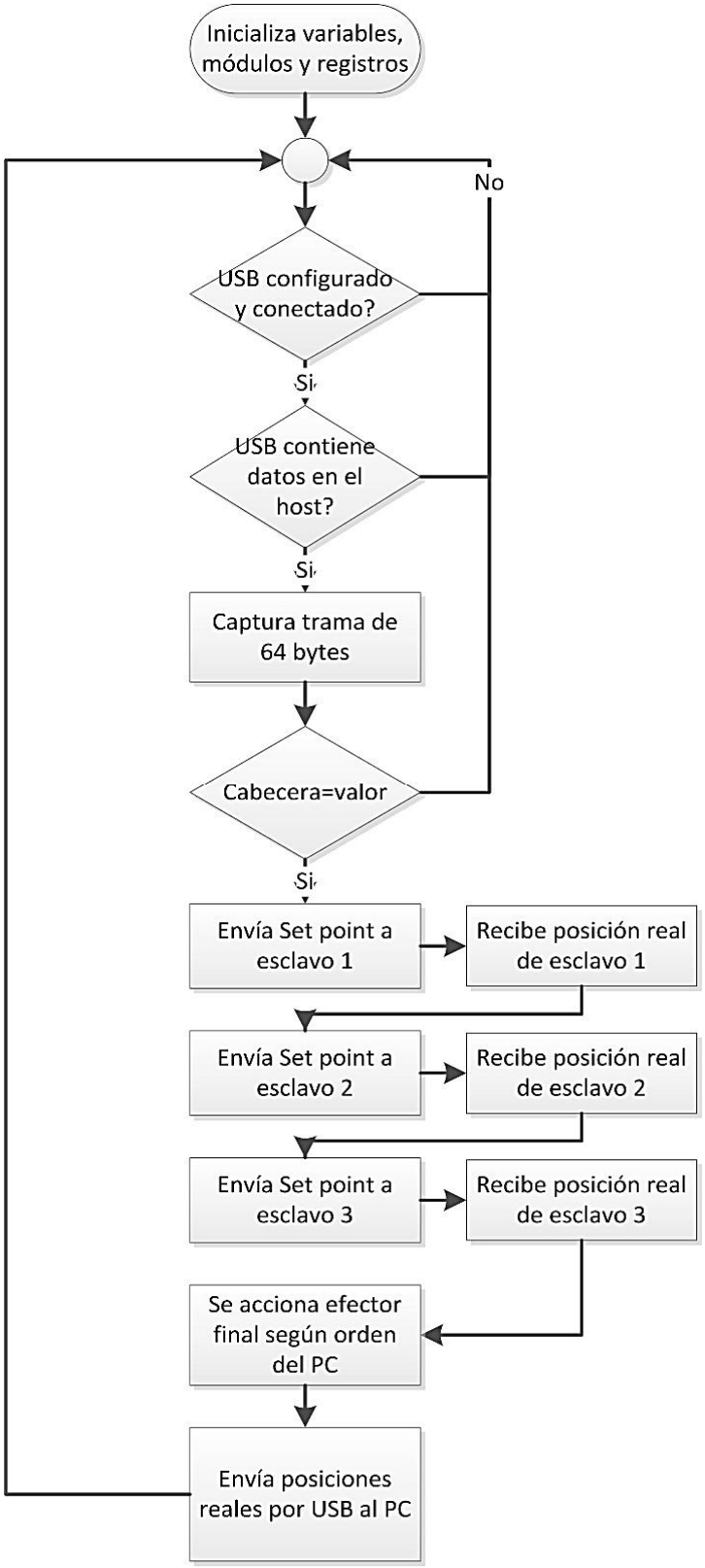


Figura 57. Diagrama de flujo del software del Microcontrolador Maestro.

El mismo software se utilizó durante el proceso de caracterización, con algunas modificaciones, debido a que solo se necesita comunicar con un Esclavo, y la cantidad de datos enviados y recibidos por el I2C es mayor (50Bytes); también se modificó la secuencia de envío de datos. El diagrama de flujo del programa utilizado para la caracterización de cada eje de movimiento se muestra en la Figura 58.

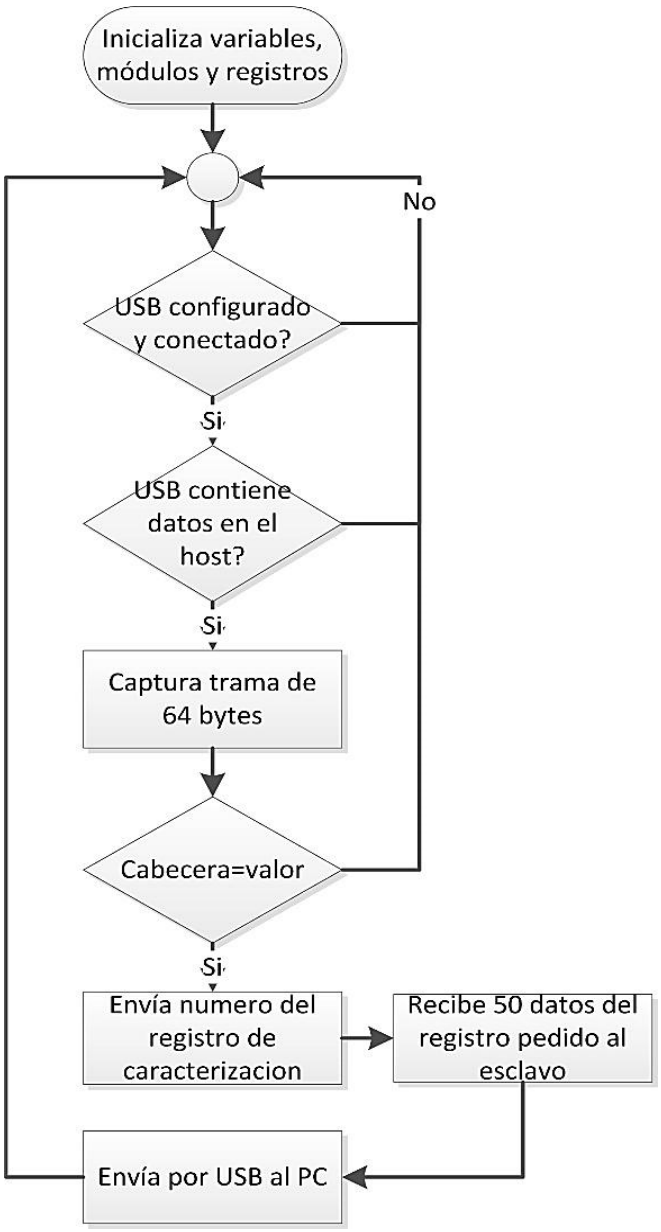


Figura 58. Diagrama de flujo de software del Microcontrolador Maestro para Caracterización

La implementación de los diagramas de flujo anteriores en el microcontrolador, se realizó en el lenguaje C y se utilizó el compilador PIC CCS C (el mismo lenguaje y compilador que se utilizó en el trabajo publicado en Matlab Central (Pool, 2010)). El código del programa de caracterización se detalla en el ANEXO 1.

5.1.1.1 Trama de datos del protocolo USB

Para que la comunicación entre el computador y el microcontrolador Maestro sea exitosa, es necesario que éstos se encuentren sincronizados, es decir que ambos sepan que dato corresponde a cada posición de la trama. Por ello se implementó un orden para los datos como una trama subdividida, de tal manera que al llegar los datos al Maestro, éste envíe a cada Esclavo los datos que le corresponden, o en el caso contrario, que sea el Maestro el que envíe los datos al computador en un orden que éste entienda a que corresponden dichos datos.

Ya que la comunicación USB envía y recibe siempre una trama de 64 Bytes, la trama se subdividió en 6 partes. La primera parte corresponde a la cabecera de la trama y a un byte de configuración cada uno con un tamaño de un byte. La cabecera es un valor que le indica al Esclavo cuantos datos tiene que recibir del maestro; el segundo byte corresponde a configuración, éste es un registro que tiene 255 posibles combinaciones. Su uso se centra en funciones adicionales que pueda tener cada uno de los controladores para futuras modificaciones (por ejemplo: calibración, cambios en las constantes del controlador, cambio en la precisión, etc).

La segunda, tercera y cuarta parte de la trama USB, son tramas de 20 Bytes cada una, en la cual se envía la información a cada Esclavo. Esto quiere decir que el envío máximo de datos a cada Esclavo por cada trama USB recibida es de 20 Bytes. La principal función de estas tramas es el envío de la posición en valor de encoder o Set-Point a cada Esclavo. Se deja una alta brecha para futuras modificaciones que se deseen implementar en el

sistema, como es el caso del envío de la información según la función del byte de configuración mencionado anteriormente.

La quinta parte de la trama USB, corresponde a un byte con el cual se dará el mando de encendido o apagado del efector final. Y por último se tiene una sexta parte de la trama que corresponde a un Byte el cual hasta el momento no se ha implementado y queda disponible para futuras aplicaciones en la máquina.

La distribución de la trama USB recibida en el Maestro se detalla en la Tabla 5. Para éste proyecto, se detalla la distribución de los 20 bytes recibidos de la trama USB en la Tabla 6, la cual es la misma para los controladores Y y Z. Los bytes que no se utilizan, se envían como 0.

Tabla 5. Distribución de la trama de 64 Bytes recibida por el Maestro del USB

Cabecera y configuración 2 Bytes	**Eje X 20 Bytes**	**Eje Y 20 Bytes**	**Eje Z 20 Bytes**	**Efector Final 1 Byte**	**No se implementa 1 Byte**

Tabla 6. Distribución de una sección de la trama recibida.

Eje X (20 Bytes)																			
SP MSB	SP LSB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Para éste proyecto, los datos indispensables en la comunicación USB son las instrucciones de posición de cada uno de los ejes, representada con dos bytes (6 bytes en total); el byte de cabecera, el byte de configuración y el byte del efecto final. Quedando de esta forma 9 bytes ocupados en la trama de 64 Bytes.

Hasta el momento se ha hablado acerca de la recepción de datos por parte del Maestro y como éste envía los datos a cada controlador Esclavo. Ahora se procede hablar de cómo el Maestro recibe los datos de los Esclavos y como lo envía al Software Matlab.

La trama enviada por el Maestro y recibida en el computador, al igual que su contraparte, contiene datos de posición, pero éstos no corresponden a instrucciones sino a la posición real de la máquina y se envían para que el computador pueda saber la posición del efector final de la máquina. La sincronización de los datos se conserva enviando y recibiendo los datos siempre en el mismo orden. En la Tabla 7 se puede apreciar la trama de 64 Bytes enviada desde el Maestro, donde solo se envían los datos de posición de cada uno de los controladores. Los bytes que no se utilizan, se envían como 0.

Tabla 7. Trama enviada al PC por USB desde el modulo Maestro.

Pos eje X MSB	Pos eje X LSB	Aux eje X	Pos eje Y MSB	Pos eje Y LSB	Aux eje Y	Pos eje Z MSB	Pos eje Z LSB	Aux eje Z	55 bytes restantes

5.1.1.2 Trama de datos del protocolo I2C

La trama que envía el Maestro y recibe el Esclavo, se desarrolló lo mas dinámica posible, debido a que la cantidad de datos recibidos en los Esclavos puede variar según la información que se necesite enviar (posición, configuración, calibración, etc.).

La cabecera que recibe el Maestro por la transmisión USB, le da conocimiento de cuantos bytes se deben transmitir a cada Esclavo. Teniendo esta información, el Maestro puede reducir la cantidad de datos que envía a los Esclavos, aumentando de esta manera la velocidad de la comunicación y disminuyendo el tiempo de la ejecución de la interrupción I2C en el Esclavo, lo cual es de suma importancia para no afectar el control de velocidad y de posición.

La trama de datos que el Maestro recibe del Esclavo es mucho más simple, pues esta solo contiene los datos de posición a transmitir, pero conserva el mismo dinamismo que la enviada. Esto es posible gracias a que en dicha trama, el Maestro incluye el valor de la cabecera, con el objetivo de utilizar la misma, para indicarle al Esclavo cuantos datos debe

recibir. Al conocer el Esclavo el valor de dicha cabecera, ya no es necesario que éste le informe al Maestro cuantos datos enviará. Los datos que puedan sobrar en la transmisión simplemente no se utilizan.

Durante el desarrollo del proyecto se ejecutaron muchos procedimientos donde fue necesario enviar y recibir muchos datos a través de la trama y las cantidades variaban según el procedimiento, por esta razón fue de mucha utilidad el dinamismo dado a la trama.

En la Tabla 8 se aprecia la trama enviada desde el Maestro al controlador Esclavo, cuando la cabecera es igual a 4, la misma trama se recibe en el Maestro pero con los valores de posición reales obtenidos desde el registro contador del módulo del encoder. Hay que tener en cuenta que el esclavo además de recibir el número de bytes de la cabecera, también recibe dos bytes adicionales que no se detallan en la trama de la Tabla 8 y son el byte de dirección y el byte de cabecera.

Tabla 8. Trama enviada por el Maestro a cualquier de los Esclavos cuando la cabecera es igual a 4.

Eje (Cabecera 4)			
SP MSB	SP LSB	X	X

5.1.2 Efector Final

El efector final es el encargado de remover el cobre de la baquelita por medio de una fresa. Para ello se utilizo un Mototool Bauker modelo MP170 (Bauker), el cual puede girar a una velocidad entre las 8000 y 32000 RPM suficientes para el fresado del cobre, la potencia del motor es de 170W, se alimenta a una tensión de 110VAC y el consumo de corriente es de 1,6 A aproximadamente.

Figura 59. Mototool Bauker modelo MP170 usado como efector final.

La electrónica del efector final es la más sencilla de todas. No es más que una salida tipo relé con el cual se hace el encendido y el apagado del efector final

La orden de encendido y apagado del relé se hace desde la aplicación de Matlab, esta información viaja en la misma trama de 64 bytes del USB, donde el Microcontrolador Maestro la interpreta y por medio del pin 19 del PIC se alimenta la base del transistor 2N3904 con el cual se accionar la bobina del relé.

La corriente de operación del relé debe ser superior a la corriente del Mototool. Para este caso se seleccionó un relé comercial de 10Amps. El circuito esquemático del accionamiento del efector final se puede apreciar en la Figura 60, el cual fue elaborado en Proteus.

La tensión de alimentación es de 12VDC por ser la tensión de operación de la bobina del relé. Las dos señales de entrada provienen de la tarjeta del Maestro (terminal 1 y 2 de entrada, ver Figura 60). El montaje se implementó en una PCB diferente para mantener el diseño modular, tal como se puede apreciar en la Figura 61. En la Tabla 9 se describen los terminales de entrada y salida de dicho circuito.

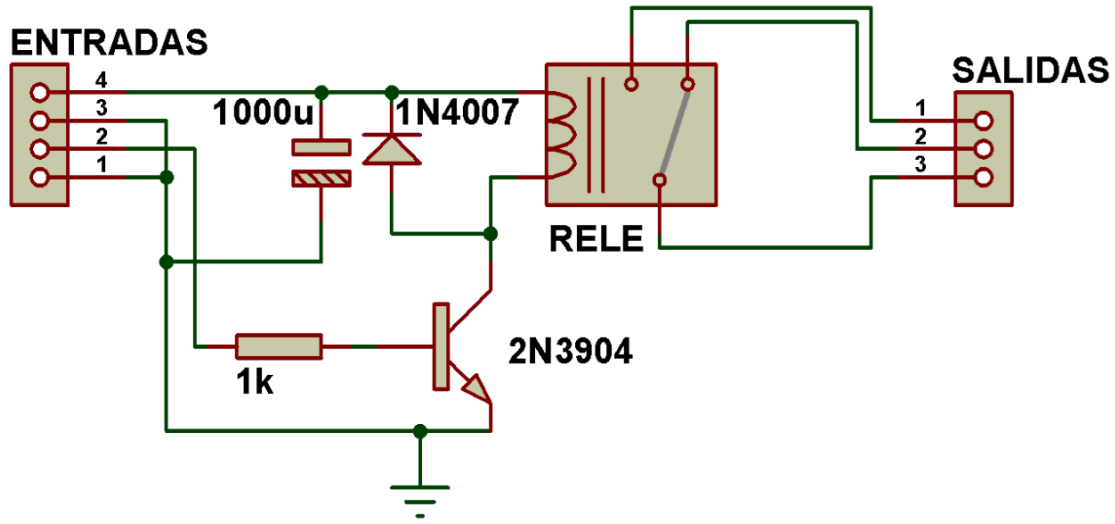


Figura 60. Circuito esquemático de accionamiento del efector final.

Figura 61. Implementación del accionamiento del efector final.

Tabla 9. Terminales de entrada y de salida del esquemático de la Figura 61.

Nombre del terminal	Característica
V+	Entradas de alimentación de 12VDC proveniente de la fuente.
Tierra	
Señal de activación	Entrada de activación de 5VDC del relé.
Tierra	
Común	Señal de salida tipo relé. NC, normalmente cerrado; NA, normalmente abierto.
NC	
NA	

5.1.3 Hardware

Como se mencionó en la sección 2.3.1, la electrónica utilizada en este proyecto está basada en Microcontroladores. Para la selección del microcontrolador se analizaron diferentes referencias de la familia Microchip con modulo USB, de las que se destacan las referencias de la Tabla 10, de la cual se seleccionó el microcontrolador PIC18F4550 por tener todos los módulos de hardware, la capacidad de memoria de datos, memoria de programa y los pines de entrada y salida que se requieren para esta aplicación; además por ser utilizado en el proyecto "Comunicación entre MATLAB y PIC de MICROCHIP usando puerto USB" (Pool, 2010).

Tabla 10. Características principales del PIC18F4550 (Microchip Technology Inc., 2009).

Device	Program Memory		Data Memory		I/O	10-Bit A/D (ch)	CCP/ECCP (PWM)	SPP	MSSP		EUSART	Comparators	Timers 8/16-Bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI	Master I^2C™			
PIC18F2455	24K	12288	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F2550	32K	16384	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F4455	24K	12288	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3
PIC18F4550	32K	16384	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3

Los módulos utilizados en el Microcontrolador se detallan en la Tabla 11. La configuración de cada uno de los módulos se explica en el código del programa del microcontrolador,

Tabla 11. Módulos utilizados en Microcontrolador PIC18F4550 (Microchip Technology Inc., 2009).

Modulo utilizado	Función del módulo del Microcontrolador
Puerto serial síncrono Maestro (Master Synchronous Serial Port - MSSP)	El módulo MSSP es una interfaz serial útil para comunicarse con otros dispositivos periféricos o Microcontroladores. El módulo MSSP puede operar en uno de dos modos: • Interfaz periférica serial SPI (Serial Peripheral Interface). • Inter-circuitos integrados I2C (Inter-Integrated Circuits). Éste soporta el modo Maestro, multi-Maestro y el modo Esclavo. El módulo I2C se utilizó para la comunicación con los módulos Esclavos.
Bus Serie Universal (Universal Serial Bus - USB)	El modulo USB es una interfaz serial de alta velocidad, muy útil para comunicarse con el computador o dispositivos que cuenten con este puerto. El modulo USB se utilizó para la interacción entre la máquina y el computador.

Para el correcto funcionamiento del microcontrolador es necesario usar otros dispositivos electrónicos, como son el cristal de 20MHz con el cual se genera la señal de reloj del microcontrolador, de capacitores para reducción de ruido y absorción de variaciones de voltaje en la línea de entrada, reguladores de voltaje para evitar altos valores de alimentación que puedan dañar el microcontrolador, un diodo a la salida del efector final para evitar que retornen señales que puedan dañar los puertos entrada/salida del micro; y borneras para la interconexión entre las diferentes tarjetas electrónicas y la alimentación.

Para el correcto funcionamiento de la comunicación I2C, se agregaron dos resistencias Pull-Up de 1KΩ. El valor de estas resistencias depende de la velocidad a la que se haga la comunicación. Por medio de estas resistencias se polarizan los transistores de salida del módulo I2C.

En el caso del USB es necesario colocar un capacitor de 470pF, además se debe configurar la señal de reloj con la cual trabajara el modulo USB en el microcontrolador (Microchip Technology Inc., 2009). Éste módulo trabaja a una velocidad de 48MHz, por lo tanto se deben configurar los fusibles de programación, dicha configuración se detalla en el código del microcontrolador Maestro.

En la Figura 62 se puede apreciar el circuito esquemático del módulo Maestro con todos sus componentes, el cual fue elaborado en Proteus.

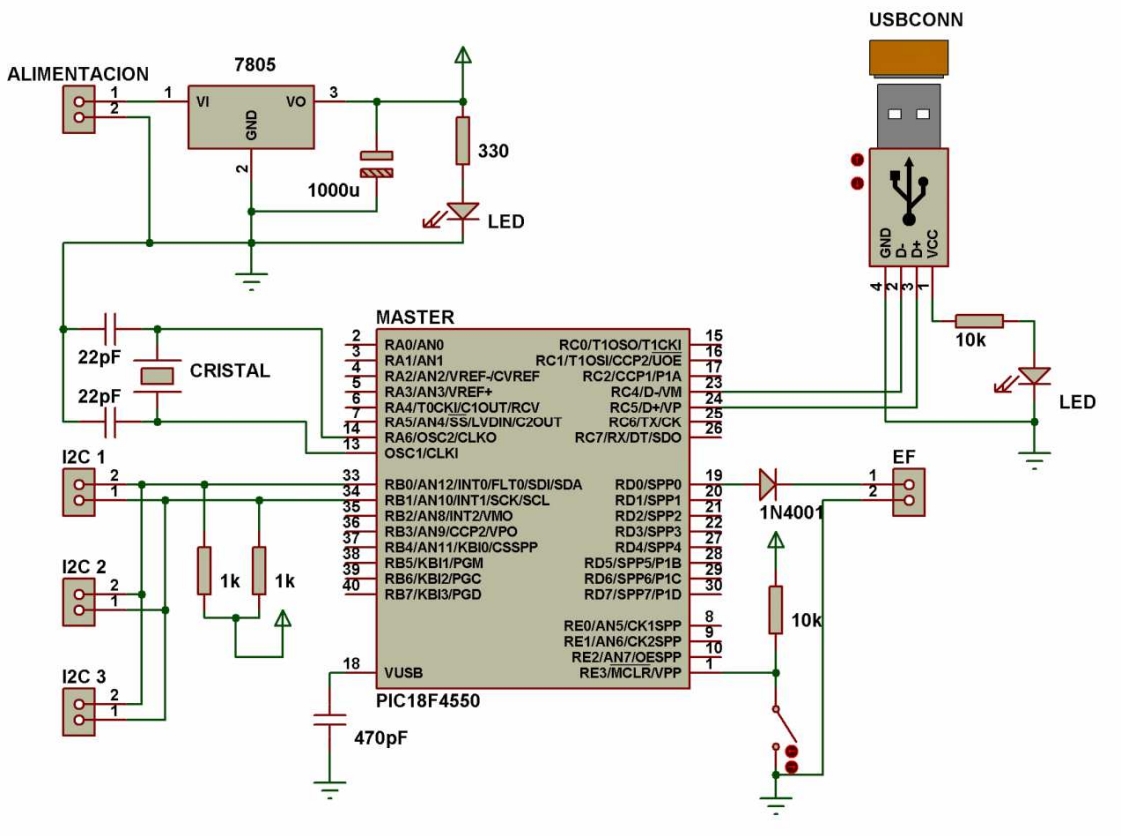


Figura 62. Esquema de circuito del módulo Maestro.

En la Figura 63 se muestra una foto del módulo Maestro implementada en éste proyecto, el cual concuerda perfectamente con el diagrama de bloques de la Figura 16. En la Tabla 12 se describen los terminales de entrada y salida de dicho modulo. La tensión de alimentación de entrada varía entre 7VDC y 35VDC que son los parámetros entre los cuales trabaja el regulador de entrada (Fairchild Semiconductor).

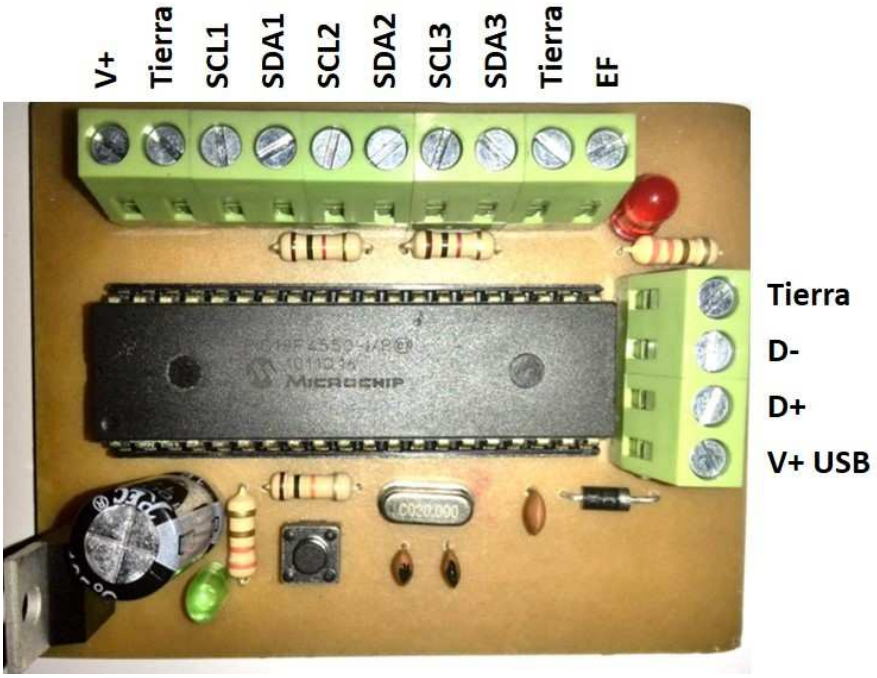


Figura 63. Tarjeta modulo Maestro (vista superior)

Tabla 12. Terminales de entrada y de salida del módulo Maestro.

Nombre del terminal	Característica
V+	Entradas de alimentación del módulo Maestro proveniente de la fuente.
Tierra	
SCL1	Terminal I2C del Esclavo X.
SDA1	
SCL2	Terminal I2C del Esclavo Y.
SDA2	
SCL3	Terminal I2C del Esclavo Z.
SDA3	
Tierra	Señal de salida hacia tarjeta del Efector Final.
EF	
Tierra	Señales del Puerto USB.
D-	
D+	
V+ USB	

5.2 Esclavo

El controlador esclavo, permite el control de movimiento de cada eje basado en comandos de posición desde la aplicación de Matlab, es lo último en el sistema de la Figura 12. El controlador debe ser capaz de controlar la velocidad, desde velocidades elevadas para el fresado mecanizado de alta velocidad (mm/min) hasta bajas velocidades para el mecanizado de alta precisión (mm/min). Además, también debe garantizar una precisión razonable y tener robustez frente a las perturbaciones.

Para cumplir con el requisito antes mencionado, se considera comúnmente un control de lazo cerrado, donde la velocidad real y los datos de posición son monitoreados y retroalimentados al controlador. El lazo de control de cada uno de los ejes de la máquina forma una estructura en cascada, está conectado con un lazo de posición y de velocidad dispuestos en serie, como se muestra en la Figura 64. El lazo de control de velocidad está situado en el interior del controlador, el lazo de control de posición rodea al lazo de control de velocidad.

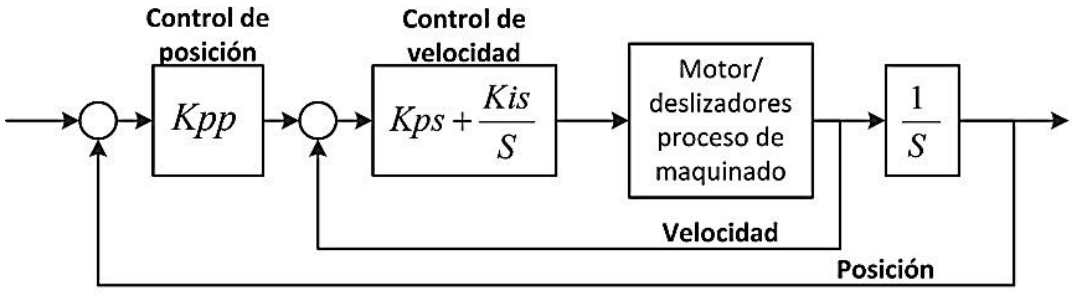


Figura 64. Estructura para lazo cerrado de control en cascada (Suh, Kang, Chung, & Stroud, 2008).

En la arquitectura de control tipo cascada es fácil ajustar las características de cada lazo de control. Sin embargo, es necesario primero garantizar la estabilidad del lazo de control interior para que todo el lazo de control sea estable y para reducir al mínimo la dependencia entre el lazo exterior y el lazo interior. Para lograr el propósito anteriormente mencionado, el lazo interior debe ajustarse para tener una respuesta del

sistema más rápida que el lazo exterior, mediante la regulación de las ganancias de cada uno de los lazos de control.

Como la velocidad y la posición real del sistema se retroalimenta por medio de un sensor al circuito de control, el motor que se utiliza se controla continuamente para minimizar el error de velocidad y el error de posición. En el sistema de husillo de máquinas herramientas, se hace la realimentación de la velocidad para mantener una velocidad de rotación regular. La señal de realimentación se genera generalmente en dos formas; con un tacómetro o con un encoder el cual genera impulsos.

El sistema de control se puede clasificar en cuatro tipos según la ubicación a la que este unido el sensor de posición o encoder (al eje del motor, directamente al efector final) los cuales son: en lazo abierto, lazo semi cerrado, lazo cerrado y el lazo hibrido (Suh, Kang, Chung, & Stroud, 2008).

El lazo abierto no tiene retroalimentación y se puede aplicar en el caso en el que la precisión del control no es alta y se utiliza un motor paso a paso con el cual se puede conocer la posición por medio de la secuencia de movimiento. Debido a que el lazo abierto no necesita un detector y un circuito de realimentación, la estructura es muy simple. Además, la precisión del sistema está directamente influenciada por la exactitud del motor paso a paso, del tornillo de bola, y la transmisión.

El lazo semi-cerrado, es el mecanismo de control más popular y tiene la estructura que se muestra en la Figura 65. En éste tipo, el sensor de posición está unido al eje del motor y detecta la rotación del motor. La precisión de la posición del eje es directamente proporcional a la exactitud del tornillo de bolas, del acople y de la caja de engranajes si la tiene.

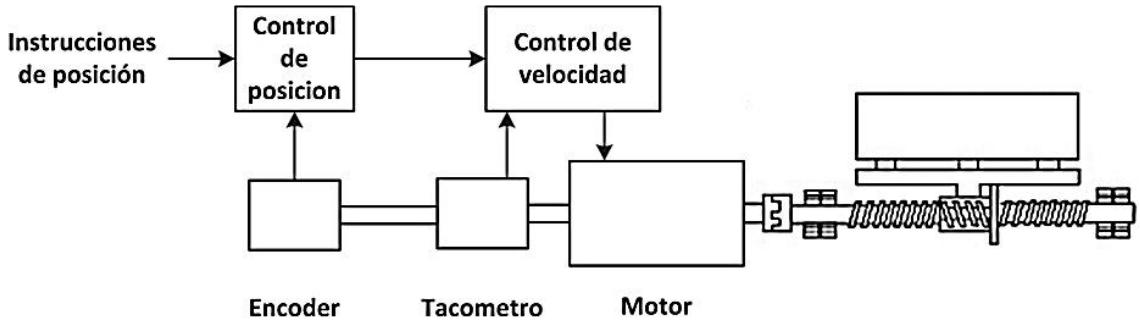


Figura 65. Lazo de control semi-cerrado (Suh, Kang, Chung, & Stroud, 2008).

Como el rendimiento del lazo semi-cerrado depende de la precisión del tornillo de bolas, del acople y de la caja de engranajes si la tiene; también es posible aumentar la exactitud de la posición a través de la compensación de paso y compensación de la holgura. Sin embargo, la cantidad de holgura puede variar de acuerdo con el peso, la ubicación del efector final y la acumulación de errores de paso del tornillo de bolas que según la temperatura.

Una forma de aumentar la exactitud y la precisión del sistema, es aplicando el lazo cerrado de control de la Figura 66, en donde el detector de posición está fijado a la mesa de la máquina y el error de posición real se retroalimenta al sistema de control. El lazo cerrado y el lazo semi-cerrado son muy similares y se diferencian en la ubicación del detector de posición.

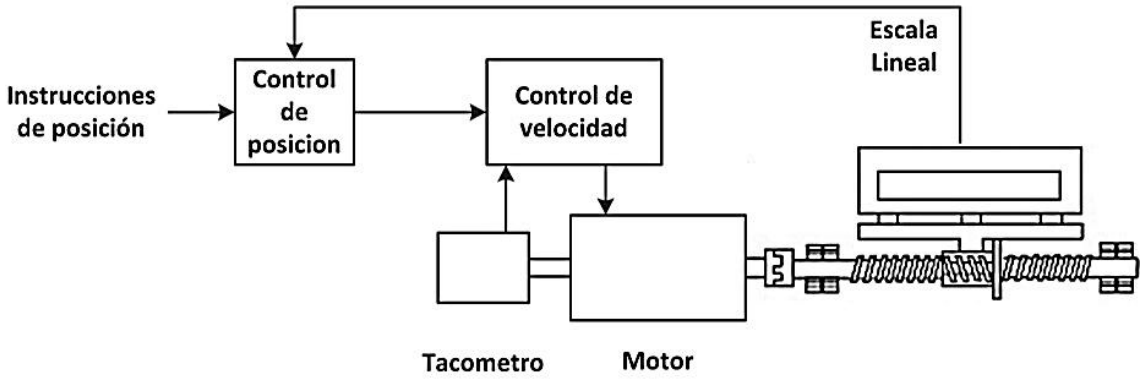


Figura 66. Lazo de control cerrado (Suh, Kang, Chung, & Stroud, 2008).

Y por último se tiene el lazo híbrido de control mostrado en la Figura 67. En el lazo híbrido, hay dos tipos de lazo de control; el lazo semi-cerrado, donde se detecta la posición del eje de un motor, y el lazo cerrado, el cual se basa en una escala lineal o sensor lineal fijado a la mesa.

Con el lazo semi-cerrado, es posible que el control sea de alta ganancia porque la máquina no está incluida en el sistema de control. El lazo cerrado aumenta la precisión al compensar el error que el bucle semi-cerrado no puede controlar. Debido a que el lazo cerrado se usa para compensar el error de posición, se comporta bien a pesar de la baja ganancia. Al combinar el lazo cerrado y el lazo semi-cerrado, es posible obtener una alta precisión con alta ganancia.

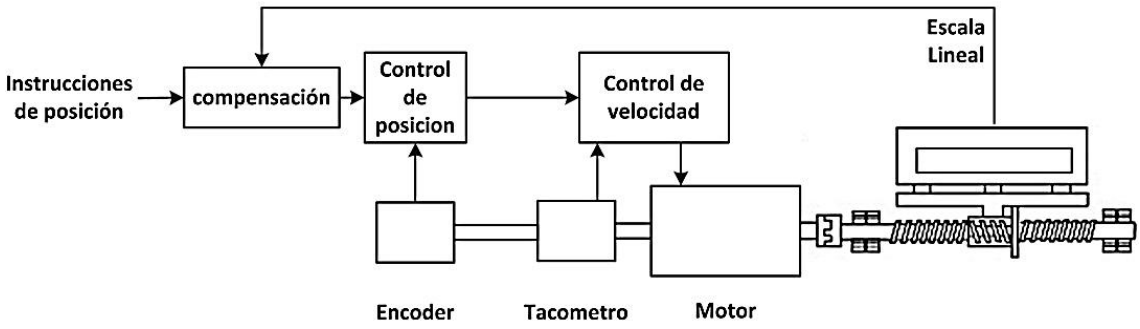


Figura 67. Lazo de control hibrido (Suh, Kang, Chung, & Stroud, 2008).

Para éste proyecto el encoder está unido al eje del motor, por lo tanto se tiene un control de lazo semi-cerrado. Hay que tener en cuenta que la velocidad se calcula en base a la lectura del encoder dentro de un mismo intervalo de tiempo, esta velocidad calculada es la que entra a compensar el error de velocidad con la salida del controlador de posición. En la Figura 68 se puede apreciar el lazo de control implementado que es muy similar al de la Figura 65.

En la siguiente sección se detalla el hardware del controlador, posteriormente se explica la manera de cómo se obtuvo el modelo del sistema, también se detalla el control de velocidad y de posición; y por último la implementación en el microcontrolador.

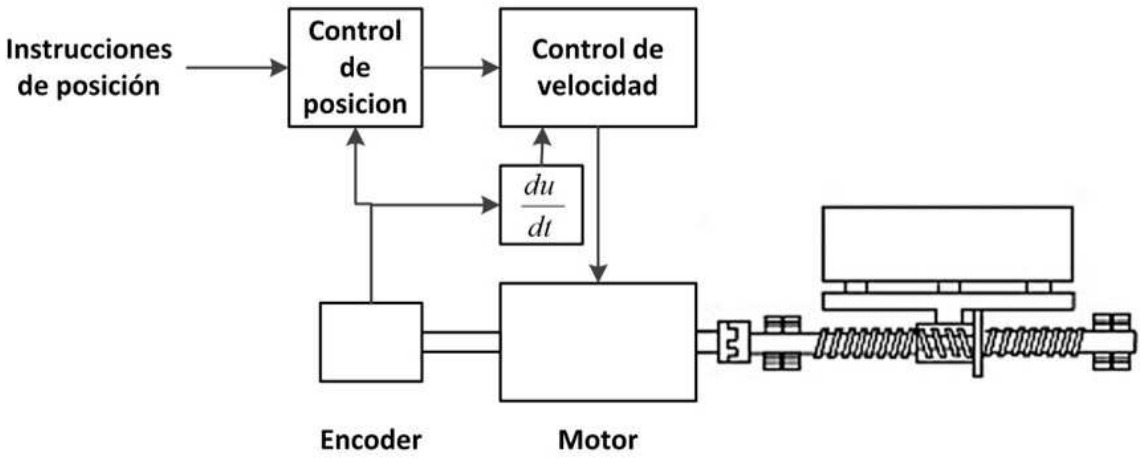


Figura 68. Lazo de control implementado.

5.2.1 Hardware

El hardware del Esclavo es más complejo, debido a que se encarga de controlar la posición de cada eje de movimiento. Como se explicó en la sección 2.3.2 se utiliza un microcontrolador el cual es capaz de comunicar el modulo Esclavo con el Maestro y también de controlar la posición de cada eje.

Para la selección del microcontrolador, se analizaron diferentes referencias de fabricantes recomendadas para este tipo de aplicaciones en las que se destacan los de la familia Microchip de la Tabla 13, de la cual se seleccionó el microcontrolador PIC18F4431 por tener todos los módulos de hardware, por la capacidad de memoria de datos y de programa, y los pines de entrada y salida que se requieren para esta aplicación.

Los módulos utilizados en el Microcontrolador se detallan en la Tabla 14. Además de esto se hizo necesario implementar la interrupción RB del microcontrolador para poder detectar las señales provenientes de los finales de carrera. La configuración de cada uno de los módulos y de la interrupción se explica en el código del programa del microcontrolador.

Tabla 13. Características principales del PIC18F4431 (Microchip Technology Inc., 2010).

Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	CCP	SSP		EUSART	Quadrature Encoder	14-bit PWM (ch)	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)				SPI	Slave I^2C™				
PIC18F2331	8192	4096	768	256	24	5	2	Y	Y	Y	Y	6	1/3
PIC18F2431	16384	8192	768	256	24	5	2	Y	Y	Y	Y	6	1/3
PIC18F4331	8192	4096	768	256	36	9	2	Y	Y	Y	Y	8	1/3
PIC18F4431	16384	8192	768	256	36	9	2	Y	Y	Y	Y	8	1/3

Tabla 14. Módulos utilizados en Microcontrolador PIC18F4431 (Microchip Technology Inc., 2010).

Modulo utilizado	Función del módulo del Microcontrolador
Módulo de retroalimentación de movimiento (Motion feedback module)	El módulo de retroalimentación de movimiento es un periférico de propósito especial diseñado para aplicaciones de retroalimentación de movimiento. El módulo se compone de dos sub-módulos de hardware: • Módulo de captura de entrada IC (Input capture module). • Interfaz de codificadora de cuadratura QEI (Quadrature encoder interface module). El módulo QEI se utilizó para leer las señales provenientes del encoder.
Módulo PWM de control de potencia (Power control PWM module)	El módulo PWM de control de potencia simplifica la tarea de generar múltiples salidas PWM sincronizadas para usarlas en el control de motores y aplicaciones de conversión de potencia.
Puerto serial síncrono (Synchronous Serial Port)	El módulo SSP es una interfaz serial útil para comunicarse con otros dispositivos periféricos o Microcontroladores. El módulo SSP puede operar en uno de dos modos: • Interfaz periférica serial SPI (Serial Peripheral Interface). • Inter-circuitos integrados I2C (Inter-Integrated Circuits). El módulo I2C se utilizó para la comunicación con el modulo Maestro.

Como se mencionó en la sección 2.3.1, cada Esclavo tiene una dirección única para la comunicación I2C. La dirección utilizada para cada uno de los módulos Esclavos se detalla en la Tabla 15.

Tabla 15. Direcciones hexadecimales de los módulos Esclavos.

Modulo Esclavo	Dirección hexadecimal
Eje X	0xB0
Eje Y	0xC0
Eje Z	0xA0

Para el correcto funcionamiento del microcontrolador es necesario usar otros dispositivos electrónicos, como son el cristal de 20MHz con el cual se genera la señal de reloj del microcontrolador, de capacitores para reducción de ruido y absorción de variaciones de voltaje en la línea de entrada, reguladores de voltaje para evitar altos valores de alimentación que puedan dañar el microcontrolador, diodos a la salida de los PWM para evitar que retornen señales provenientes de la etapa de potencia y que puedan dañar los puertos entrada/salida del micro; y borneras para la interconexión entre las diferentes tarjetas electrónicas y la alimentación.

El circuito esquemático del módulo Esclavo con todos sus componentes se puede apreciar en la Figura 69, el cual fue elaborado en Proteus.

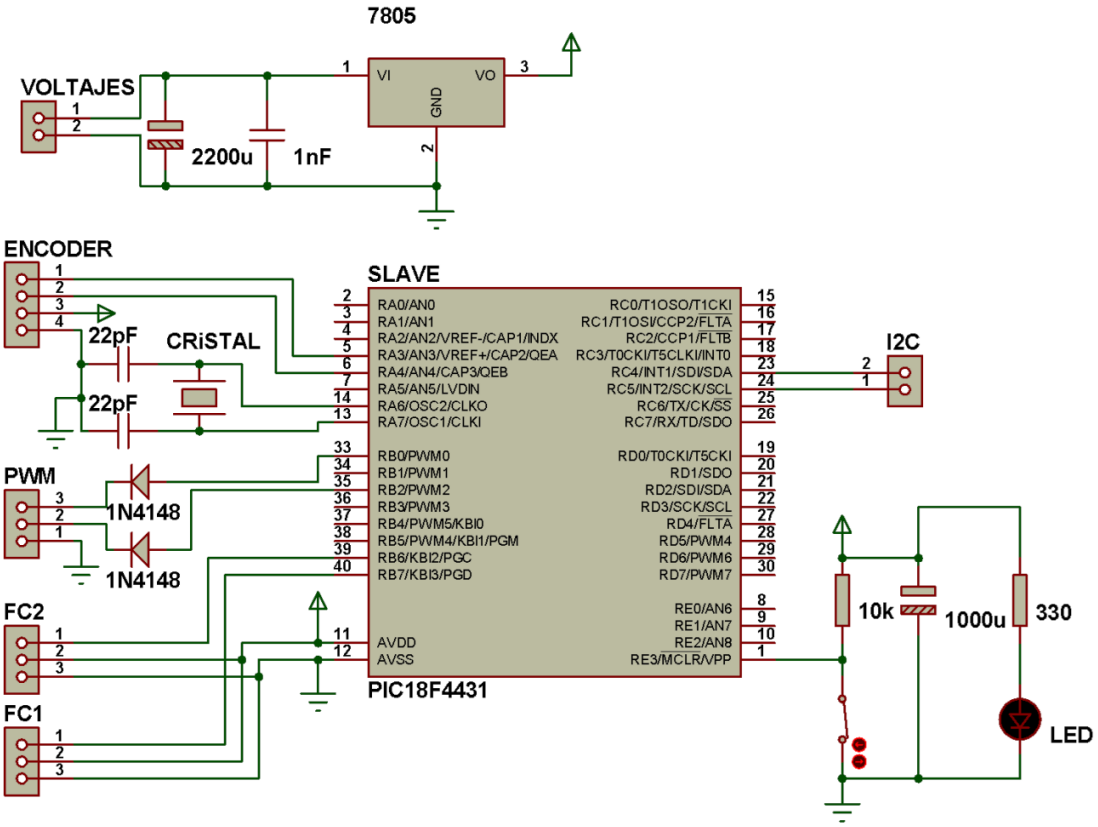


Figura 69. Esquema de circuito del módulo Esclavo.

En la Figura 70 se muestra una foto del módulo Esclavo implementada en éste proyecto, el cual concuerda perfectamente con el diagrama de bloques de la Figura 16. En la Tabla 16 se describen los terminales de entrada y salida de dicho modulo. La tensión de alimentación de entrada varía entre 7VDC y 35VDC que son los parámetros entre los cuales trabaja el regulador de entrada (Fairchild Semiconductor)

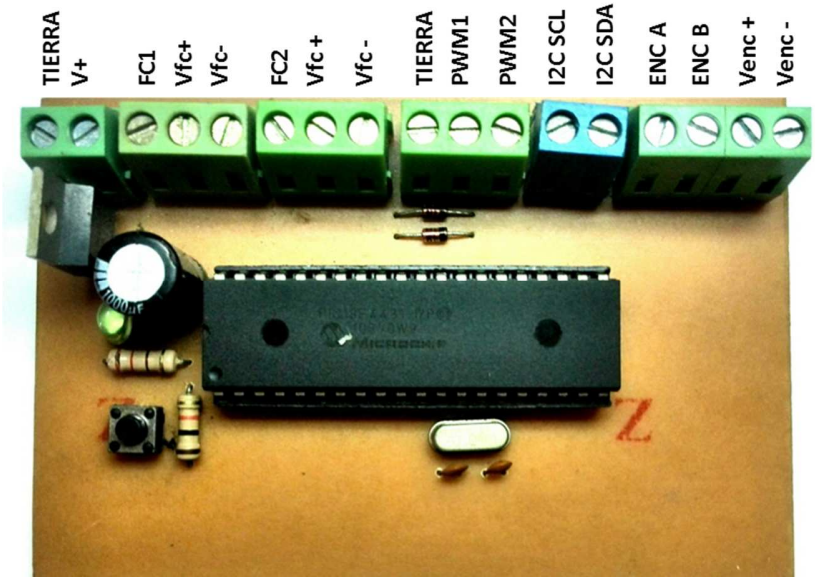


Figura 70. Tarjeta modulo Esclavo (vista superior).

Tabla 16. Terminales de entrada y de salida del módulo Esclavo.

Nombre del terminal	Característica
Tierra	Entradas de alimentación del módulo Esclavo proveniente de la fuente.
V+	
FC1	Entrada normalmente en alto proveniente del final de carrera 1. Cuando el final de carrera detecta que la maquina llega a su posición, la señal se convierte en bajo.
Vfc+	Señal de alimentación del final de carrera 1 la cual es de 5VDC
Vfc-	
FC2	Entrada normalmente en alto proveniente del final de carrera 2. Cuando el final de carrera detecta que la maquina llega a su posición, la señal se convierte en bajo.

Vfc+	Señal de alimentación del final de carrera 2 la cual es de 5VDC
Vfc-	
Tierra	Tierra común con la etapa de potencia
PWM1	Salida PWM1 hacia la etapa de potencia
PWM2	Salida PWM2 hacia la etapa de potencia
I2C SCL	Señal de reloj de la comunicación I2C
I2C SDA	Señal de datos de la comunicación I2C
ENCA	Entrada de encoder A
ENCB	Entrada de encoder B
Venc+	Salida de 5VDC de la alimentación del encoder
Venc-	

5.2.1.1 Finales de carrera

Cada eje de movimiento (X,Y,Z), tiene dos finales de carrera los cuales detectan cuando el carro ha llegado al inicio o final de su recorrido. Estos sensores además de proteger al mecanismo de colisiones en sus extremos, también son usados para la calibración del sistema. Para éste proyecto se utilizan finales de carrera ópticos los cuales requieren de una fuente de alimentación para su funcionamiento.

Los finales de carrera constan básicamente de un sensor óptico que envía una señal en alto o en bajo al ser interrumpido un haz de luz entre el emisor y el receptor. Para éste proyecto se utiliza el sensor óptico de ranura S525 de la Figura 71. Cuando el haz de luz se interrumpe entonces la señal de salida que se obtiene es en alto.

El funcionamiento del sensor no es el mas adecuado para la aplicación puesto que al estar constantemente sin enviar una señal, el controlador no sabe si hay o no un sensor conectado. Por esta razón el funcionamiento del sensor se cambia por medio de un circuito, de tal manera que el sensor envíe siempre una señal en alto dando aviso de que está conectado, y que al interrumpirse el haz de luz la señal de salida sea en bajo.

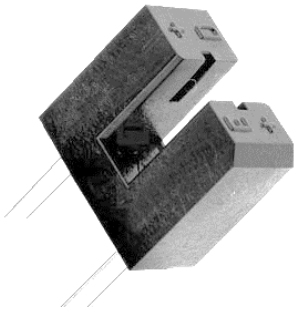


Figura 71. Sensor óptico de ranura S525.

El sensor de ranura consta de un diodo emisor de infrarrojo y un fototransistor. El circuito se alimenta con 5VDC por lo tanto es necesario de una resistencia (R2) para limitar la corriente del emisor. Al colector del fototransistor se debe conectar una resistencia (R5) a 5VDC para su polarización y el emisor a tierra. Mientras el haz de luz no se interrumpa, la salida siempre será en bajo. Para invertir la señal se utiliza un transistor NPN de referencia 2N3904, su configuración es en emisor común. El circuito se puede apreciar en la Figura 72, donde también se puede observar que cuenta con un diodo LED que indica la alimentación del final de carrera.

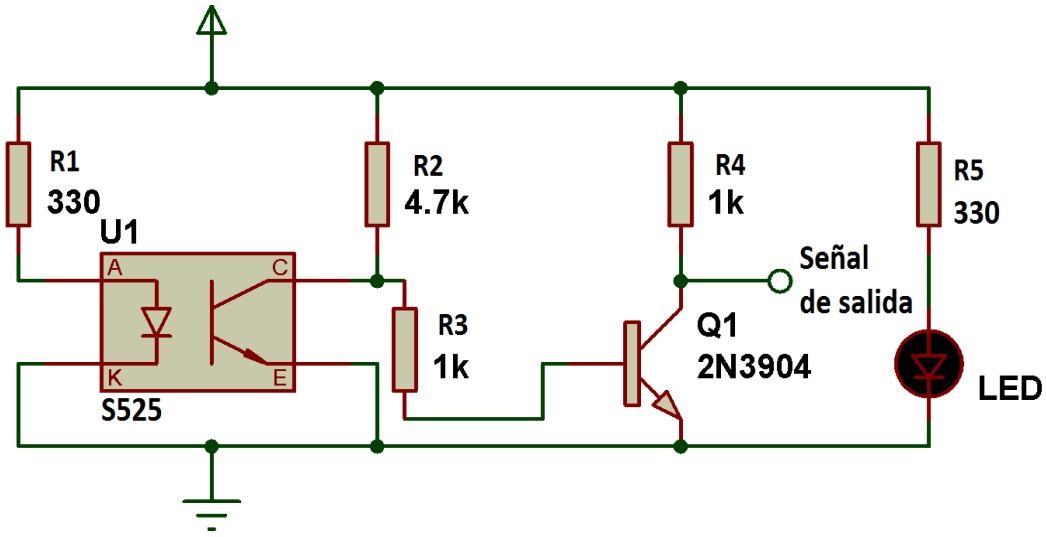


Figura 72. Circuito final de carrera con sensor óptico.

El circuito esquemático y el PCB se realizaron con ayuda del software Proteus. La tarjeta con el montaje de todos sus componentes se aprecia en la Figura 73.

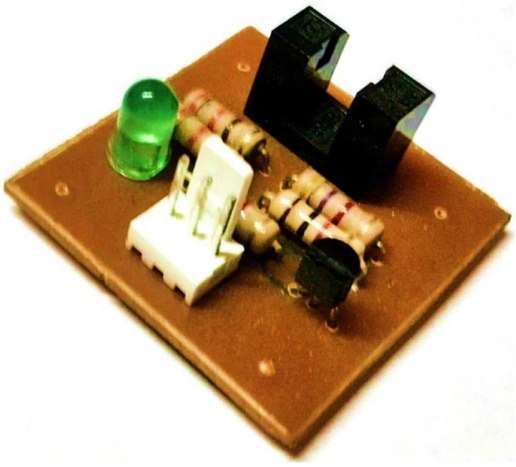


Figura 73. Tarjeta de final de carrera con sensor de ranura.

5.2.1.2 Encoder

Tal y como se mencionó en la sección 2.3.2, el encoder a utilizar en esta aplicación es de tipo incremental de cuadratura. El encoder incremental de cuadratura proporciona normalmente dos formas de ondas cuadradas y desfasadas entre si 90° eléctricos, los cuales por lo general son “canal A” y “canal B”. Con la lectura de un solo canal se puede obtener la posición y velocidad de rotación, mientras que si se capta también la señal “B” es posible discriminar el sentido de giro de rotación en base a la secuencia de datos que producen ambas señales. En la Figura 74 se puede apreciar la forma de onda del canal A con respecto al canal B.

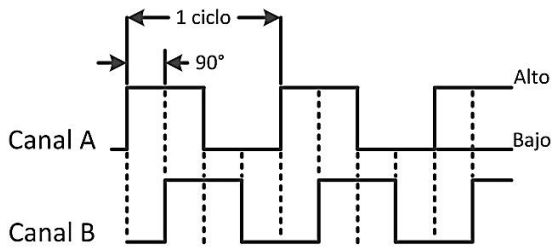


Figura 74. Señales de encoder incremental Canal A y B.

El actuador utilizado en éste proyecto, tiene un encoder de efecto Hall y salida digital tal como se puede apreciar en la Figura 75. Los sensores de efecto Hall están formados por un imán en forma de rueda que gira a la par del eje del motor, el sensor se ubica en una

parte fija y al detectar el campo magnético del imán se genera un cambio de estado en su salida. Éste tipo de sensores por ser magnéticos, pueden trabajar en diferentes entornos y no necesitan mantenimiento o limpieza, también resulta mucho más sencillo su montaje sobre el eje del motor debido a su tamaño reducido.

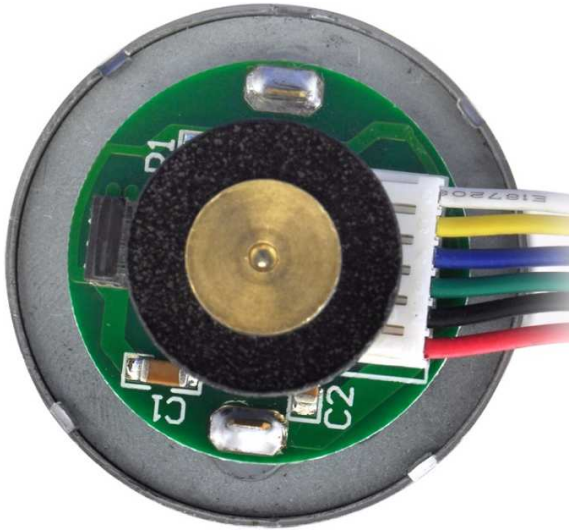


Figura 75. Encoder incremental de efecto hall montado sobre el eje del actuador (Pololu Robotics & Electronics, s.f.).

Tabla 17. Terminales de conexión del encoder y del motor (Pololu Robotics & Electronics, s.f.).

Color de cable	Función
Rojo	Terminal V+ del motor
Negro	Terminal V- del motor
Verde	Tierra del encoder o GND
Azul	Vcc del encoder (3.5 - 20 V)
Amarillo	Salida de encoder A
Blanco	Salida de encoder B

En la Tabla 17 se detallan las señales de entrada y de salida del motor y el encoder. Las señales A y B son las que entran al módulo QEI del microcontrolador por medio de los pines 5 (QEA) y 6 (QEB), el modulo lee ambas señales y si la señal A adelanta a la señal B en 90 grados, incrementa el valor de un contador de 16 bits por cada pulso del encoder. Si la señal B adelanta a la señal A, se disminuye el valor del contador por cada pulso de encoder. El nombre del contador de 16 bits en el microcontrolador es POSCNT y está

representado por dos registros de 8 bits cada uno, donde se almacena el mas y menos significativo (POSCNTH y POSCNTL).

La configuración del módulo del encoder, se detalla en el código del microcontrolador Esclavo; es el mismo para cada uno de los controladores.

5.2.1.3 Etapa de potencia

La etapa de potencia trabaja como una extensión del módulo de Esclavo y se encarga de amplificar la señal que controla el motor. Por lo cual solo trabaja con dos señales de entrada, correspondientes a las señales PWM del control, y dos señales de salida, que son el resultado de amplificar las señales de entrada y se conectan directamente al motor.

Ésta etapa consta principalmente de un circuito integrado puente H de la referencia L298 (ver Figura 76) el cual es un driver de motor con doble puente H, conectado en configuración paralela para brindar una mayor potencia a la señal de salida (ST Microelectronics, 2000).

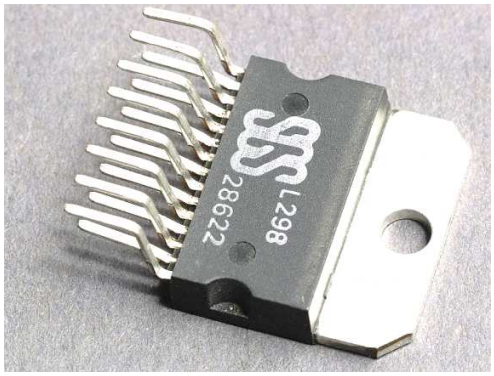


Figura 76. Circuito Integrado puente H L298 (Wikimedia).

En la Figura 77 se puede ver el esquema del circuito utilizado para la etapa de potencia con el integrado L298 y los componentes adicionales necesarios para su funcionamiento, tales como reguladores de tensión para alimentarlo con 5V ya que la lógica interna de funcionamiento trabaja a una tensión menor. Los diodos 1N4007 son para la protección

del L298 ante las tensiones generadas en la bobina del motor por efecto de la conmutación del PWM y del cambio en el sentido de giro, el cual fue elaborado en Proteus.

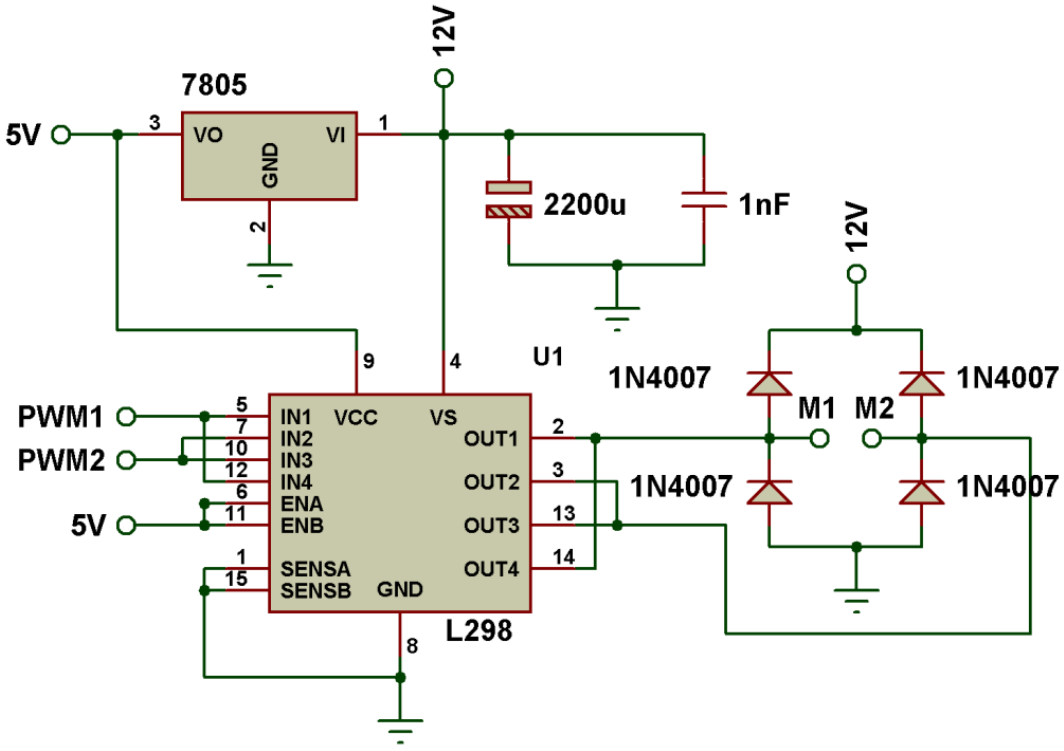


Figura 77. Circuito esquemático de la etapa de potencia.

En la Figura 78 se muestra una foto de la etapa de potencia implementada en éste proyecto y en la Tabla 18 se describen los terminales de entrada y salida de dicha etapa. La tensión de alimentación de entrada varía entre 7VDC y 35VDC que son los parámetros entre los cuales trabaja el regulador de entrada (Fairchild Semiconductor); y están dentro del rango de tensión del L298 (ST Microelectronics, 2000). Para este proyecto, los motores trabajan a 12VDC por lo tanto la tensión de entrada debe ser igual a éste valor para no dañar al motor (Pololu Robotics & Electronics, s.f.).

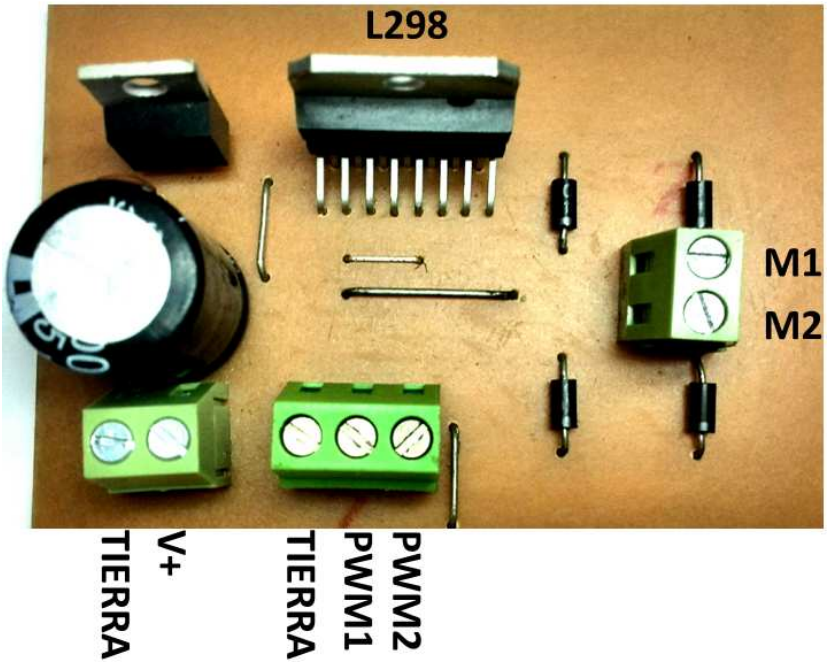


Figura 78. Etapa de potencia del controlador.

Tabla 18. Terminales de entrada y de salida de la etapa de potencia del módulo Esclavo.

Nombre del terminal	Característica
Tierra	Señal de alimentación de la etapa de potencia proveniente de la fuente.
V+	
Tierra	Tierra común con el módulo del Esclavo
PWM1	Entrada PWM1
PWM2	Entrada PWM2
M1	Salida hacia el motor 1
M2	Salida hacia el motor 2

5.2.2 Modelo Del Sistema

Para diseñar cualquier sistema de control, se hace necesario conocer su modelo matemático que explique el funcionamiento del sistema. Éste modelo matemático es de vital importancia ya que con él se conoce el tiempo de respuesta del sistema, el tipo de controlador a utilizar y su sintonización.

Hay diferentes formas de obtener el modelo del sistema, una es caracterizando cada uno de sus componentes hasta obtener la función de transferencia que represente el sistema, ésta opción resulta algo engorrosa debido a que se desconoce muchas de las constantes mecánicas del sistema. Otra manera de estimar el modelo, es conociendo la respuesta del sistema a una entrada escalón en lazo abierto (Coughran). A partir de los resultados obtenidos se puede determinar la función de transferencia que caracteriza el sistema.

Para caracterizar cada eje de movimiento, se realizó una serie de pruebas en lazo abierto. Por lo tanto fue necesario desarrollar un programa en el microcontrolador para el proceso de caracterización, en el cual se programó una serie de variaciones de ciclo de trabajo. También se desarrolló una aplicación en Matlab que fuera capaz de comunicarse con el microcontrolador, con el fin de poder obtener éstos datos y luego poder procesarlos para hallar el modelo del sistema.

5.2.2.1 Software de caracterización

Para el proceso de caracterización, se desarrolló un programa en el microcontrolador, el cual contiene valores por defecto que corresponden al ciclo de trabajo del PWM, cuyo valor máximo del registro al 100% es de 16383. La señal de salida se envía a la etapa de potencia del controlador a la cual se conecta el motor. Éste comienza a moverse a diferentes velocidades según el valor del registro y a la vez se captura la posición del encoder cada 20 ms para cada uno de los ejes X, Y, y Z. En la Figura 79 se puede apreciar el diagrama de flujo del software del microcontrolador para la caracterización.

Cada vez que se hace la lectura del encoder, la información es almacenada en diferentes arreglos 50 bytes; el tamaño del arreglo debe ser menor que la máxima cantidad de bytes que se transmiten por el USB el cual es de 64 Bytes. La máxima cantidad de datos con la que se hizo la caracterización está limitada al tamaño de la memoria de datos del

microcontrolador. Por esta razón la cantidad de información que se puede almacenar es de 600 bytes; por ser registro del encoder de 2 bytes, se obtendría un total de 300 datos representados en dos arreglos de 300 bytes cada uno, el cual, uno es para los bits más significativos del registro del encoder y el otro para los menos significativos.

Después de ser almacenada la información del encoder, el computador le hace la solicitud al Maestro y éste último se la hace al Esclavo teniendo como resultado el envío de una trama de 50 bytes. La solicitud se hace un total de 12 veces, hasta que los 300 datos de posición de 2 bytes estén en el computador.

El código del programa implementado en el microcontrolador maestro y esclavo para la caracterización se encuentra en el ANEXO 2 y ANEXO 3, respectivamente.

El programa del microcontrolador, en éste punto, solo es capaz de enviar señales al motor para que éste se mueva, y capturar los datos de posición. Pero para visualizar y utilizar dichos datos se hace necesario acceder a estos datos desde el computador, por esta razón se desarrolla la aplicación en Matlab que únicamente se encarga de comunicarse con los microcontroladores para pedir y recibir dichos datos. Luego de esto, se lleva a cabo un ciclo de cálculos del cual obtiene la curva final de “velocidad vs tiempo”.

La solicitud de los datos desde la aplicación en Matlab se debe hacer después que se termina la caracterización lo cual demora 6 segundos. La aplicación después que recibe las 12 tramas de 50 bytes, une los bits más significativos con los menos significativos para obtener los 300 datos de posición. Al tener un tiempo de muestreo de 20 ms, se calcula la velocidad a la que gira el motor.

La Figura 80 muestra un diagrama de flujo de la aplicación desarrollada en Matlab.

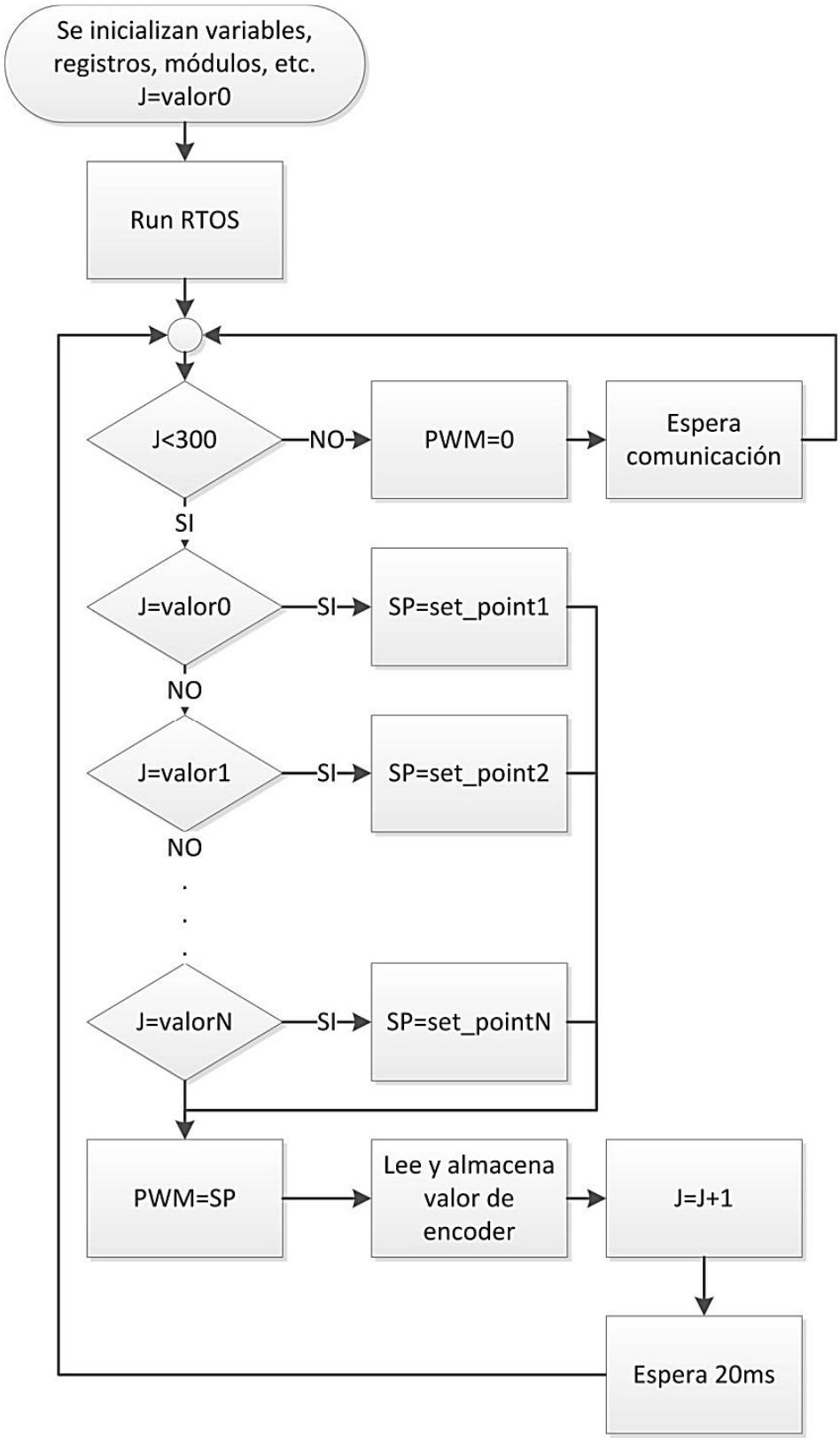


Figura 79. Diagrama de flujo del software del microcontrolador para la caracterización.

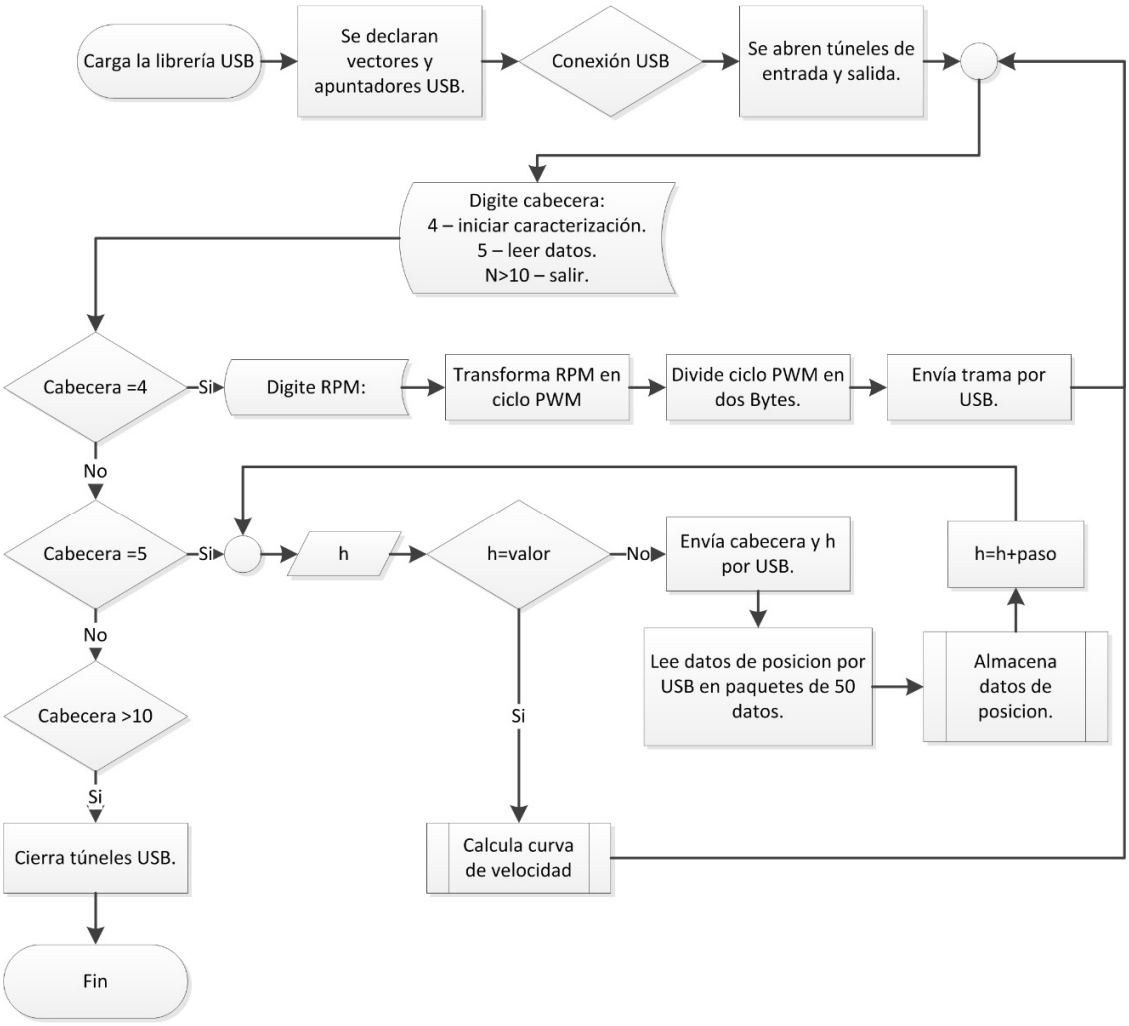


Figura 80. Diagrama de flujo aplicación en Matlab para caracterización.

El código de la aplicación en Matlab se encuentra en el ANEXO 1.

5.2.2.2 Respuesta a entrada escalón

Al ejecutar el software de caracterización de la sección 5.2.2.1 se obtuvo la curva de "velocidad vs tiempo" para cada eje de movimiento (X,Y,Z). Donde se pudo apreciar que el sistema no tiene una buena respuesta a bajas velocidades, también se observó que cuando el ciclo de trabajo se hace más pequeño el sistema se detiene lo cual representa un problema cuando se esté trabajando con el control.

Se realizaron varias curvas de caracterización para cada eje, donde se suprimieron las zonas donde el sistema no es lineal, esto con el fin de poder hallar un mejor modelo que represente cada eje de movimiento. En la Figura 81, Figura 82 y Figura 83 se puede apreciar una de las respuestas a la entrada escalón para cada eje de movimiento.

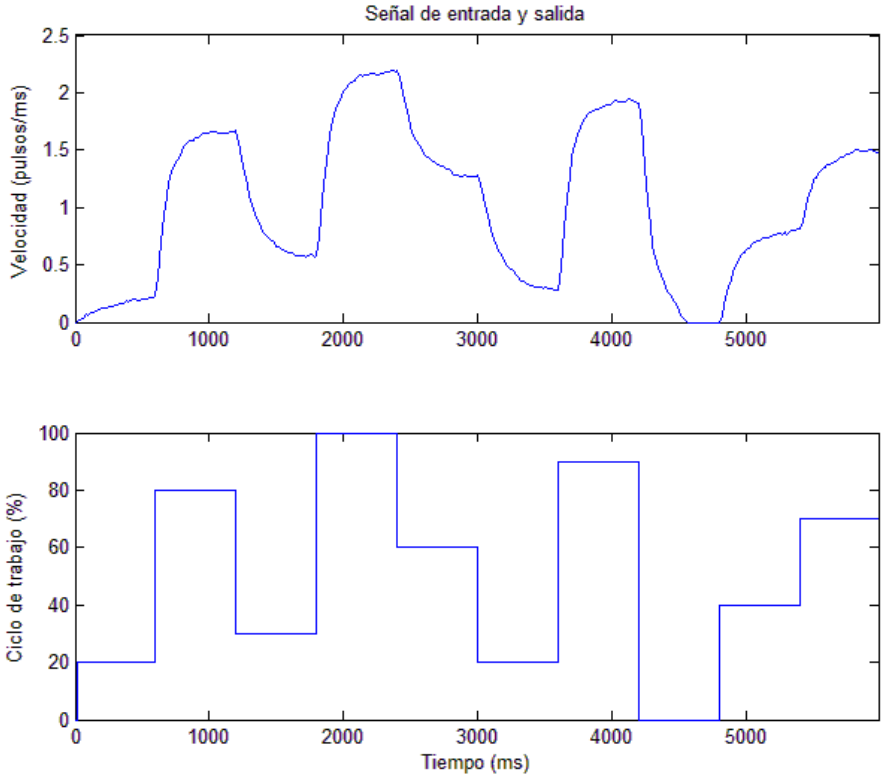


Figura 81. Respuesta a entrada escalón del eje X.

Al hacer varias pruebas de caracterización con cada uno de los ejes, se determinó que para ciclos de trabajo menores del 21% en el eje X, 32%, en el eje Y, 25% cuando el movimiento es de subida en el eje Z y 18% cuando el movimiento es de bajada en el eje Z, el sistema se detiene. Estos valores se tienen en cuenta al momento de programar el control de cada eje.

Luego de tener las curvas de caracterización para cada eje de movimiento, se procede a hallar los modelos matemáticos que representen cada uno de los sistemas, esto se realiza en la siguiente sección.

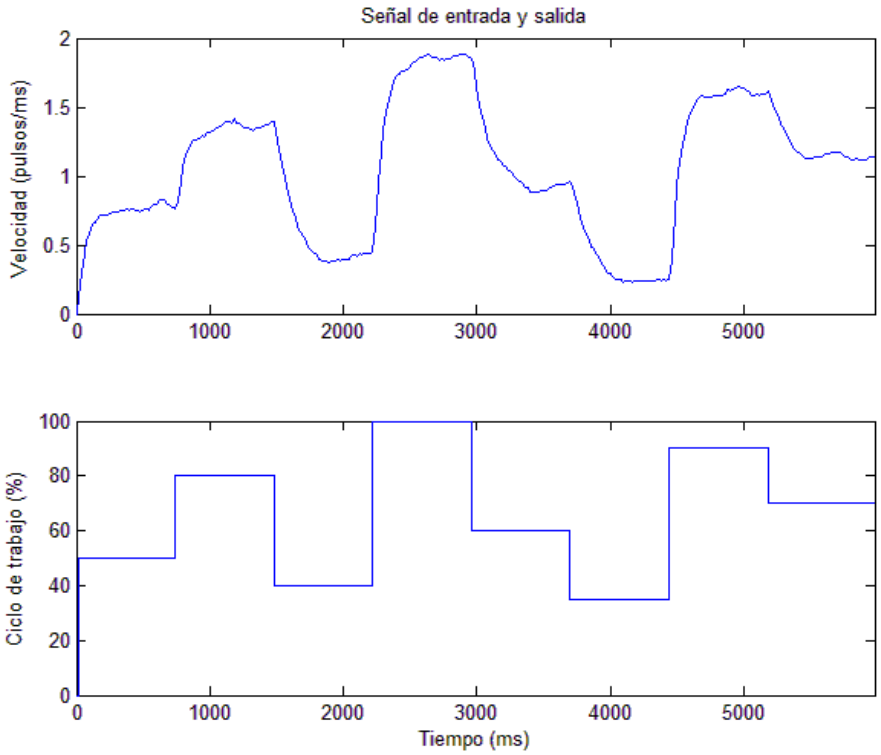


Figura 82. Respuesta a entrada escalón del eje Y.

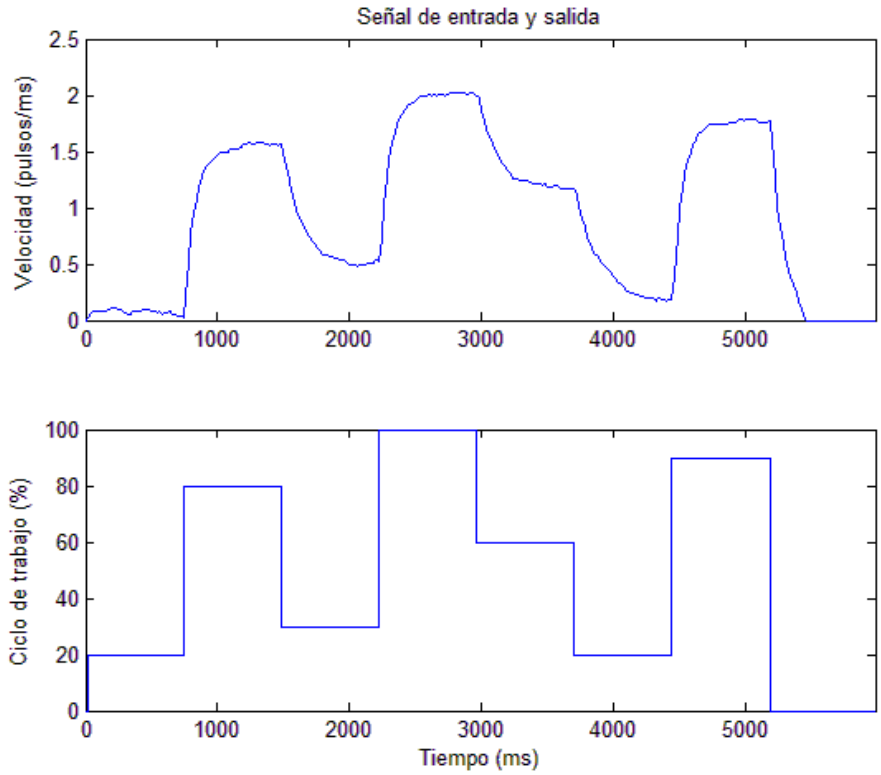


Figura 83. Respuesta a entrada escalón del eje Z.

5.2.2.3 Estimación del modelo

Teniendo las curvas de velocidades que caracterizan al sistema "motor-mecanismo", se inicia un proceso de estimación del modelo matemático o función de transferencia que representa a dicho sistema. Para ello se utiliza la herramienta IDENT del software MATLAB.

Los datos de entrada y salida del sistema se ingresan a la ventana de la herramienta IDENT y se especifica que están en el dominio del tiempo. Luego en la Figura 85 se especifican los nombres de las variables de entrada y salida, el tiempo inicial que en éste caso es cero "0", el tiempo de muestreo de los datos que es de 20 ms para cada uno de los ejes.

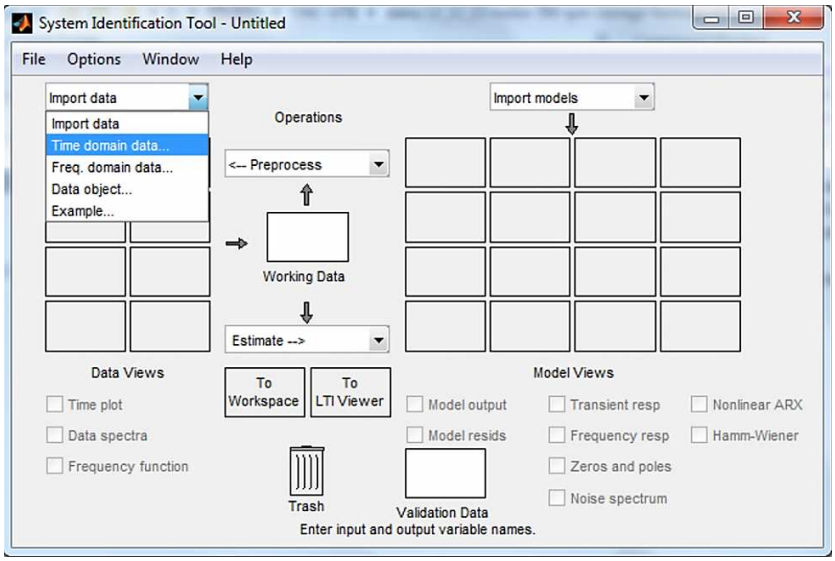


Figura 84. Ventana de la herramienta IDENT de MATLAB

Las unidades de los datos de entrada están pulsos para la posición y en pulsos por milisegundos en el caso de la velocidad. La referencia de tiempo del aplicativo IDENT, está dado en segundos; conociendo las unidades de los datos por analizar, se procede a colocar como 20 (que corresponde al tiempo de muestreo en ms) en el atributo "Sampling Interval" en la ventana de la Figura 85, esto con el fin de que el modelo obtenido este en las unidades de ms.

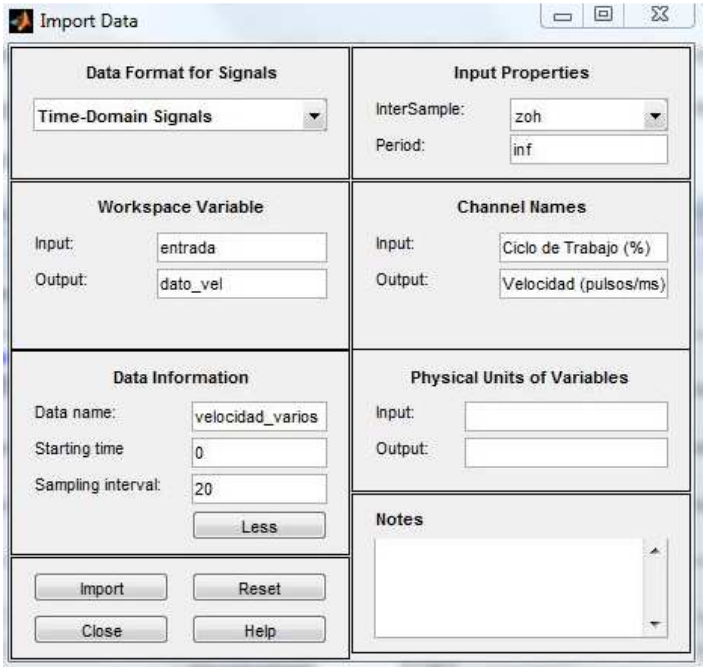


Figura 85. Ventana para importar datos.

Luego se selecciona en la ventana, el tipo de estimación que se quiere realizar a los datos importados, para este caso se selecciona “Transfer Function Models” o modelo de función de transferencia, ver Figura 86.

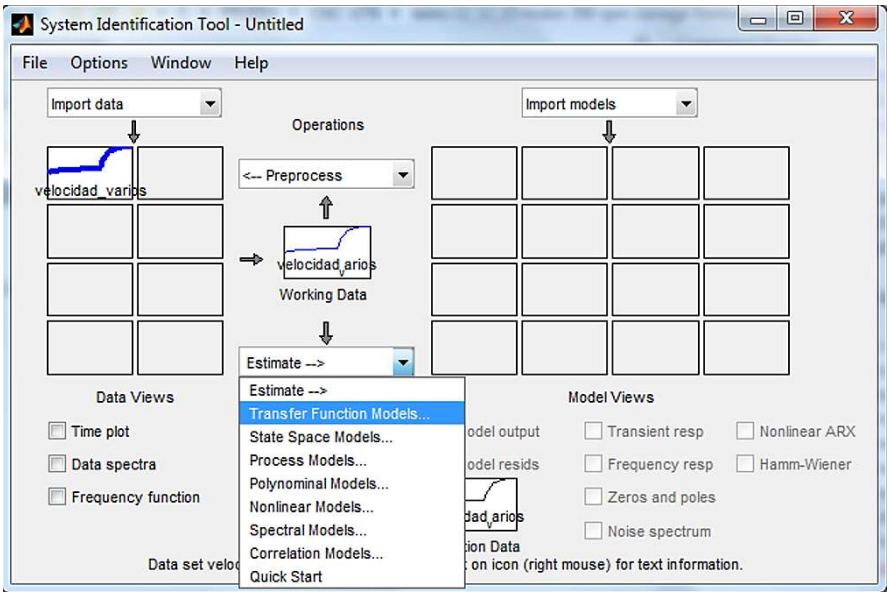


Figura 86. Ventana de herramienta IDENT. Selección de estimación.

La herramienta nos ofrece la opción de seleccionar el número de polos y ceros deseados en la función transferencia ver Figura 87. Se escoge trabajar con 1 polo y 0 ceros para obtener un sistema de primer orden. La herramienta realiza los cálculos necesarios y arroja un modelo matemático o función de transferencia que actúe lo más parecido al sistema en lazo abierto.

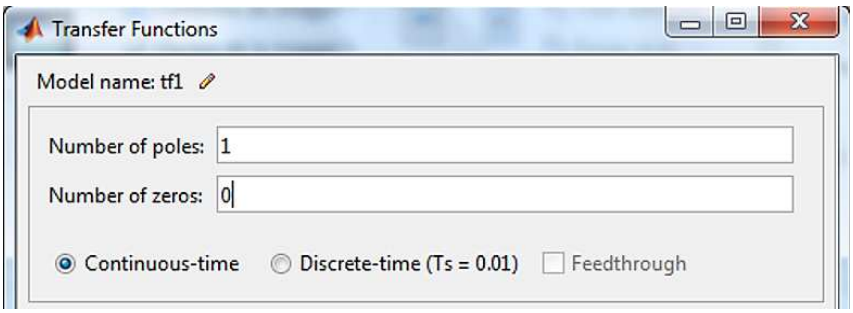


Figura 87. Selección de polos y ceros.

En la Figura 88 se puede apreciar una comparación de los datos reales con los datos de la función de transferencia estimada.

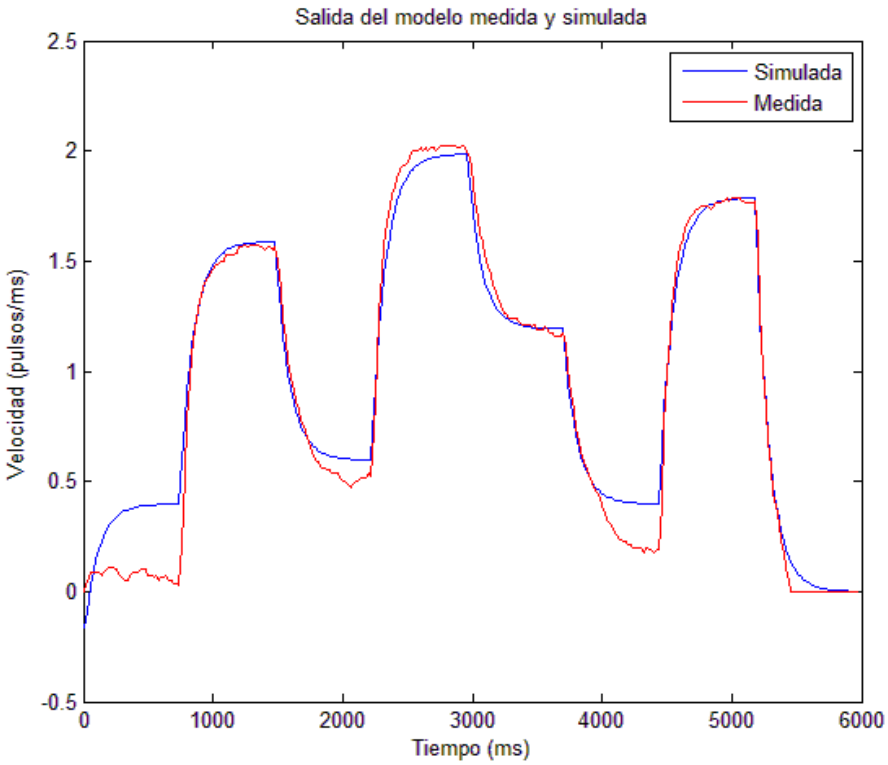


Figura 88. Modelo estimado Vs datos del sistema.

Cada función de transferencia está representada en un modelo de primer orden de la forma $F(s) = \frac{Kp}{\tau s+1}$ Donde Kp es la ganancia del sistema y τ es el tiempo de respuesta. Por lo tanto para el eje X se obtiene:

$$Fz(s) = \frac{0{,}01986}{107{,}066s + 1}$$

El resultado anterior fue el obtenido para la caracterización del motor que proporciona movimientos al eje Z. El procedimiento explicado anteriormente se ejecutó de la misma forma para los ejes X y Y, obteniendo de esta manera las siguientes funciones de transferencia.

$$Fx(s) = \frac{0{,}02140}{107{,}781s + 1}$$

$$Fy(s) = \frac{0{,}01684}{78{,}0031s + 1}$$

Luego de obtener el modelo del sistema, se procede a la elaboración del control de velocidad el cual se explica en la siguiente sección.

5.2.3 Control de velocidad

El lazo de control de velocidad, es un control tipo proporcional integral. En la Figura 64 se determinó el lazo de control en cascada donde el lazo de control de velocidad esta contenido dentro del lazo de control de posición. También se dijo que el lazo interior debe tener una mayor ganancia que el lazo exterior. En la Figura 89 se puede apreciar el diagrama de bloques del lazo de control de velocidad.

Por ser un control de velocidad agresivo o de alta ganancia y al tener una salida limitada por el ciclo de trabajo del PWM; se hace necesario implementar un sistema anti Wind-Up para evitar grandes sobre impulsos causados por la referencia del controlador de velocidad y también por el integrador y al momento de digitalizarlo, evitar que estos valores se queden almacenados en el integrador. El lazo de control de velocidad con el anti Wind-Up se puede apreciar en la Figura 90

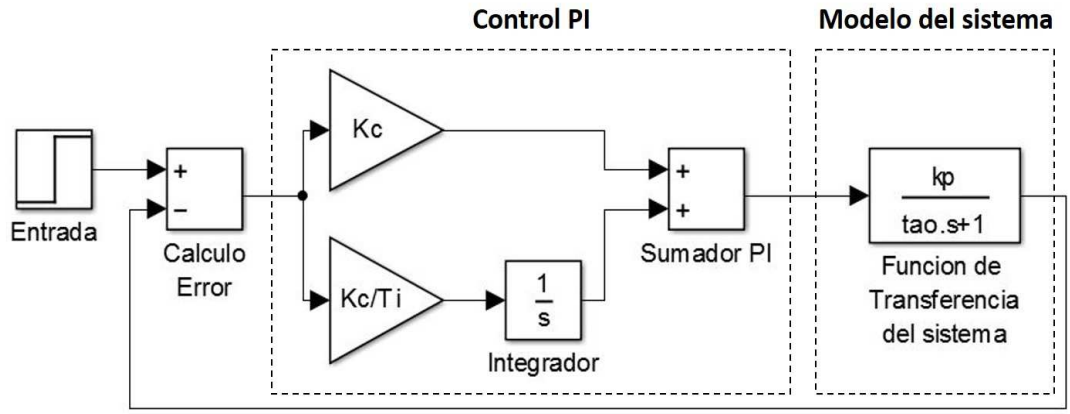


Figura 89. Lazo de control de velocidad con control PI.

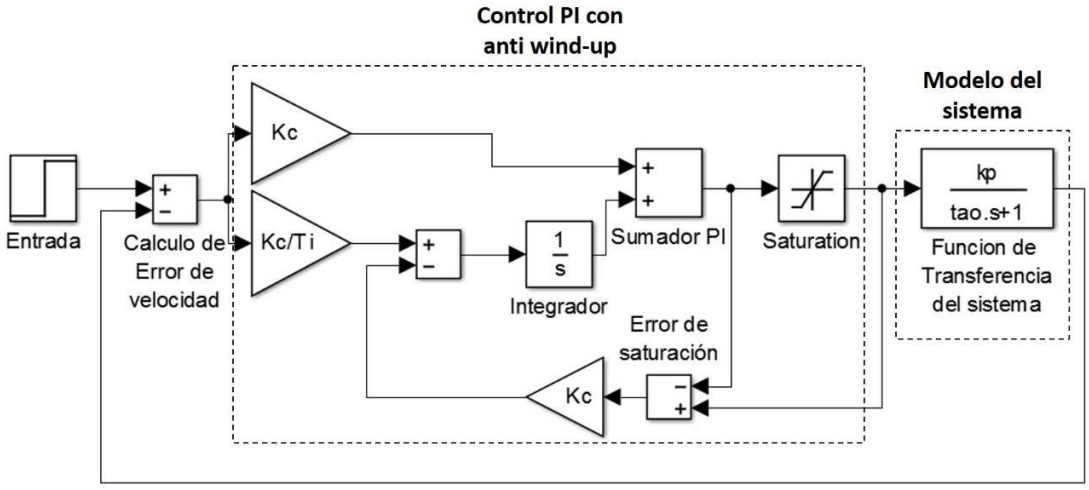


Figura 90. Lazo de control de velocidad con anti Wind-Up.

Una vez definida la estructura del lazo de control de velocidad, se procede a su sintonización del control el cual consiste en la determinación del ajuste de los parámetros (Kc y Ti), para lograr un comportamiento del sistema de control aceptable y robusto. Por la dinámica del sistema, no puede existir sobre-impulsos en la velocidad ni mucho menos en la posición. El método utilizado para la sintonización del control es el método Lambda y se detalla en la sección 5.2.3.1

5.2.3.1 Método Lambda

El método Lambda es uno de los métodos más utilizado para la sintonización de sistemas de control PID, pues ofrece una respuesta en el tiempo no oscilatoria para el tiempo (lambda) requerido por el sistema lo cual es esencial para la aplicación de este proyecto (Coughran).

Anteriormente, durante la estimación del modelo o función de transferencia de la planta, se dedujo un modelo de primer orden el cual representa la dinámica del sistema. El método lambda propone las siguientes ecuaciones de ganancia proporcional, tiempo integral y tiempo derivativo para un sistema de control de primer orden:

$Kc = \frac{\tau}{Kp(\lambda + Td)}$ Ganancia proporcional

$Ti = \tau$ Tiempo integral

$Td =$ Tiempo muerto del sistema

El tiempo derivativo es cero.

$\lambda =$ Lambda

Aplicando el método, se obtuvieron los siguientes parámetros para cada sistema de control:

Tabla 19. Constantes de control de velocidad.

Eje	Lambda	Ganancia Proporcional	Tiempo integral	Kc/Ti
X	125	40,2923	107,7818	0,3738
Y	80	51,4668	78,0031	0,6598
Z	125	43,1283	107,066	0,4028

Una vez sintonizado el control de velocidad PI, se procede hacer la simulación de la respuesta a una entrada escalón de velocidad para conocer la respuesta del modelo y del control en la simulación. Para cada una de las simulaciones, el Set-Point de velocidad se estableció en 1,5 pulsos/ms.

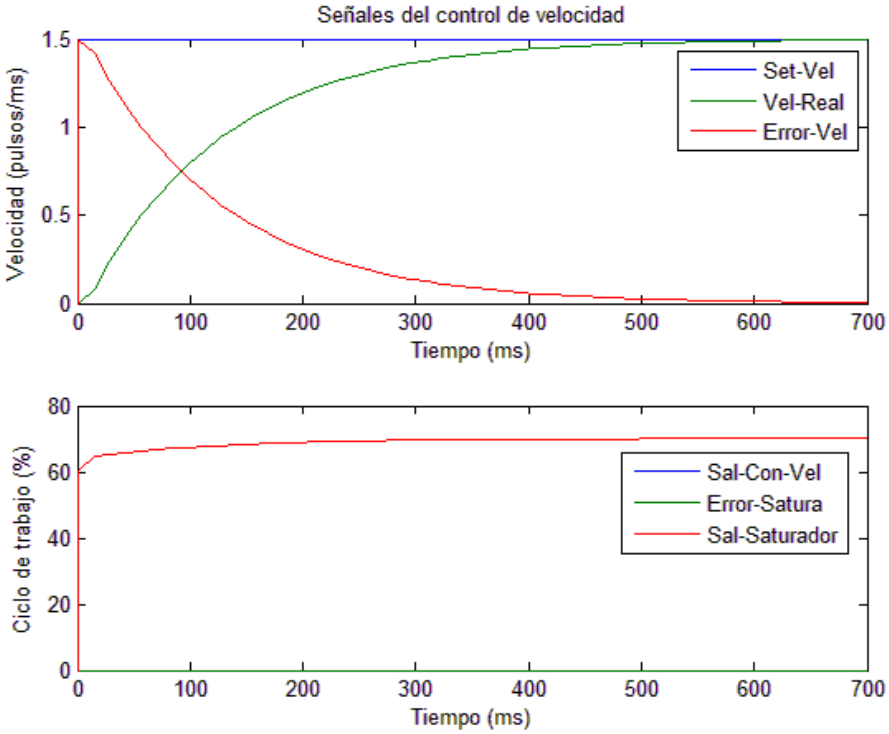


Figura 91. Simulación en Simulink del control de velocidad para el eje X.

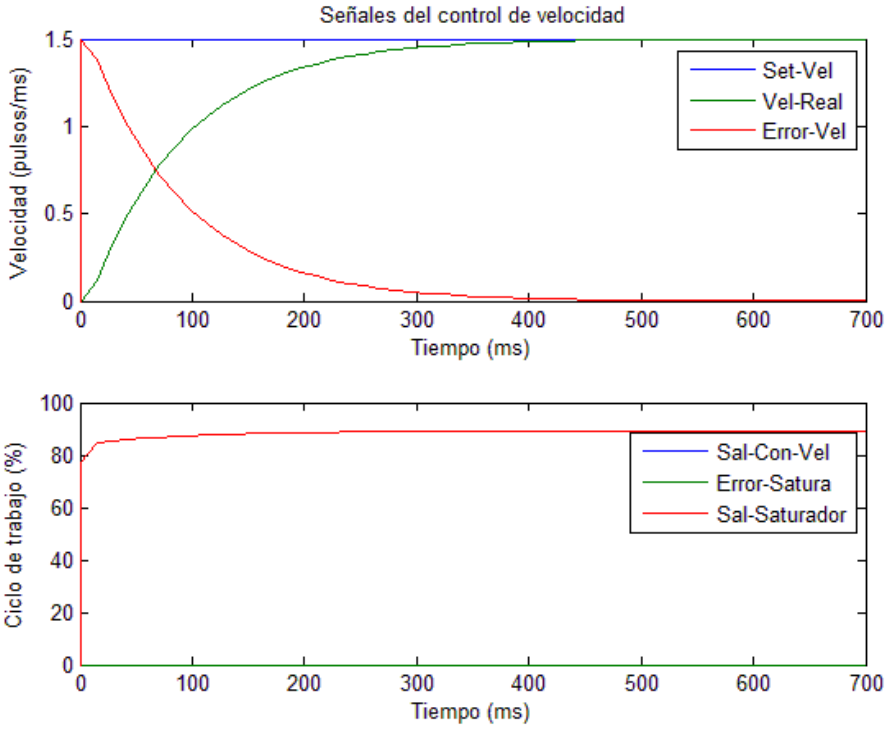


Figura 92. Simulación en Simulink del control de velocidad para el eje Y.

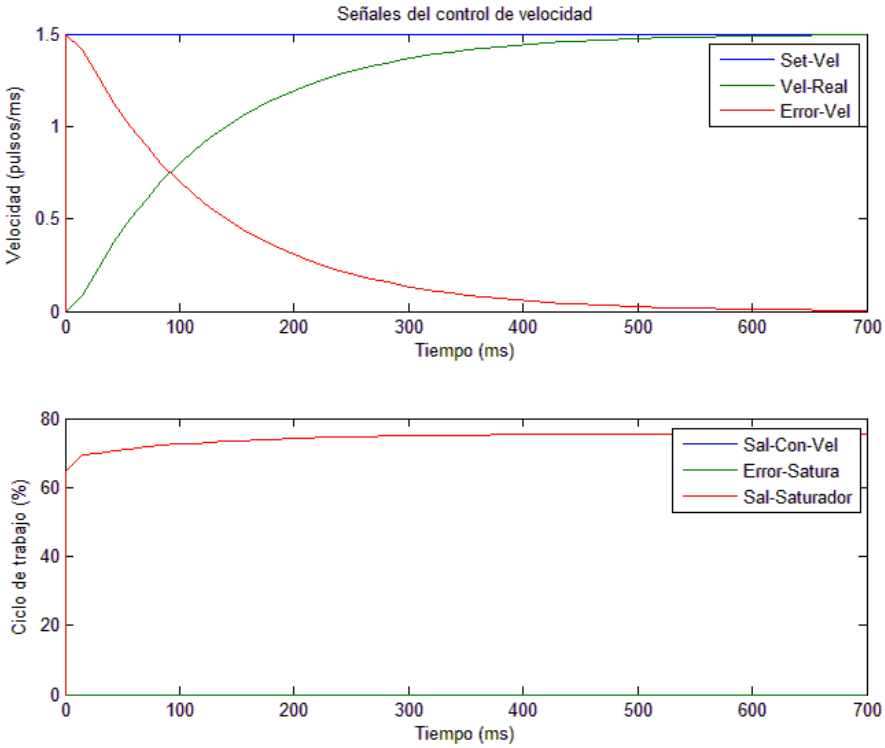


Figura 93. Simulación en Simulink del control de velocidad para el eje Z.

En las respuestas de velocidad vs tiempo de la Figura 91, Figura 92 y Figura 93 se puede observar que no existe sobre impulsos en ninguna de las tres simulaciones. Por otra parte la señal de salida del controlador PI se va ajustando hasta que el error en la velocidad de hace cero, esta característica en la respuesta se debe al integrador de controlador. Tal como se puede apreciar, la simulación del sistema es consistente y a la vez es fácil de implementar en el microcontrolador.

5.2.4 Control de posición

Luego de ser definida la estructura del lazo de control de velocidad y su sintonización con el método Lambda, se procede a elaborar el lazo de control de posición, el cual está formado por un control tipo proporcional y se aprecia en la Figura 94. La salida de este se encarga de alimentar al control de velocidad, por esta razón es un control con baja ganancia. Dada la dinámica de la máquina, la actividad del fresado y de la perforación, es

necesario que el control ofrezca una respuesta amortiguada para que la maquina trabaje dentro de los parámetros de calidad.

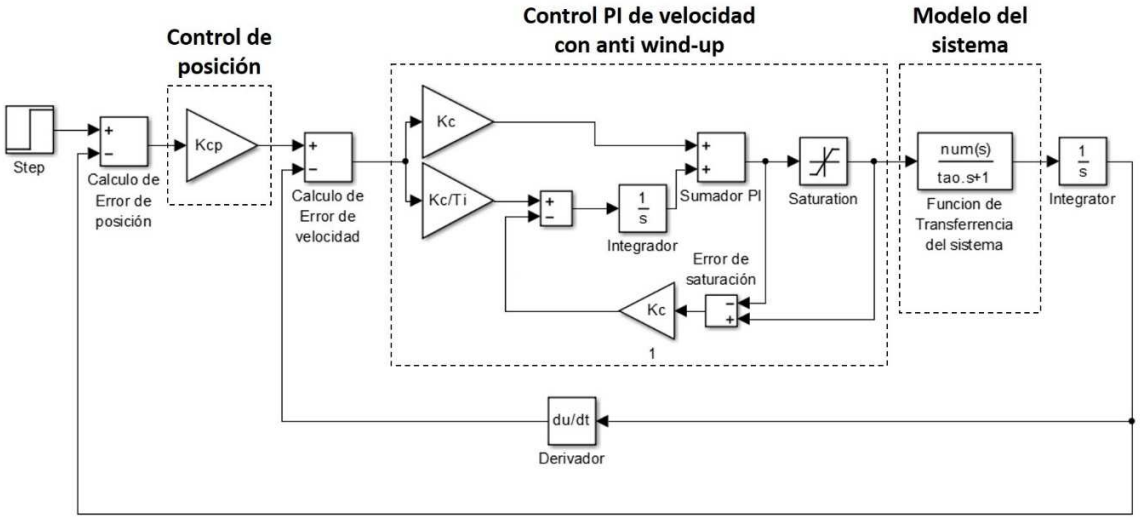


Figura 94. Lazo de control de posición y de velocidad.

Para el cálculo de la ganancia del control de posición de la Figura 95, es necesario hallar la función de transferencia de la planta de posición o G(s). Las funciones de transferencia fueron halladas con ayuda del software Matlab y se detallan en la Tabla 20 para cada uno de los ejes de la máquina.

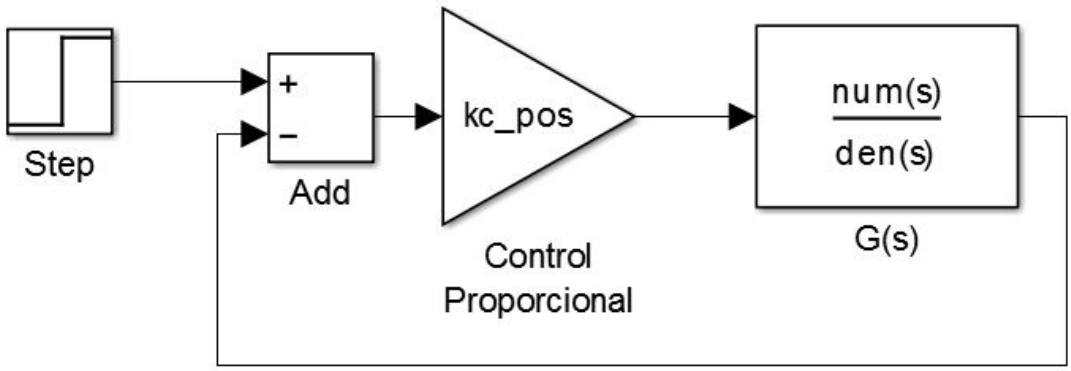


Figura 95. Lazo de control de posición.

Tabla 20. Funciones de transferencia del lazo cerrado de velocidad con integrador.

Eje	Función de transferencia de Posición
X	$G(s) = \frac{92{,}94s + 0{,}8623}{11620s^3 + 200{,}7s^2 + 0{,}8623s}$
Y	$G(s) = \frac{67{,}61s + 0{,}8667}{6084s^3 + 145{,}6s^2 + 0{,}8667s}$
Z	$G(s) = \frac{91{,}71s + 0{,}8565}{11460s^3 + 198{,}8s^2 + 0{,}8565s}$

Teniendo el sistema de la Figura 95 se procede hallar la ganancia del controlador de posición utilizando el método del lugar geométrico de las raíces. Esto con el fin de determinar gráficamente cual es el máximo valor de ganancia que puede tener el controlador de posición para que la respuesta sea amortiguada. Para ello se utilizó la función “rlocus” de Matlab, con la cual se obtuvo la Figura 96, Figura 97 y Figura 98 para cada una de las plantas.

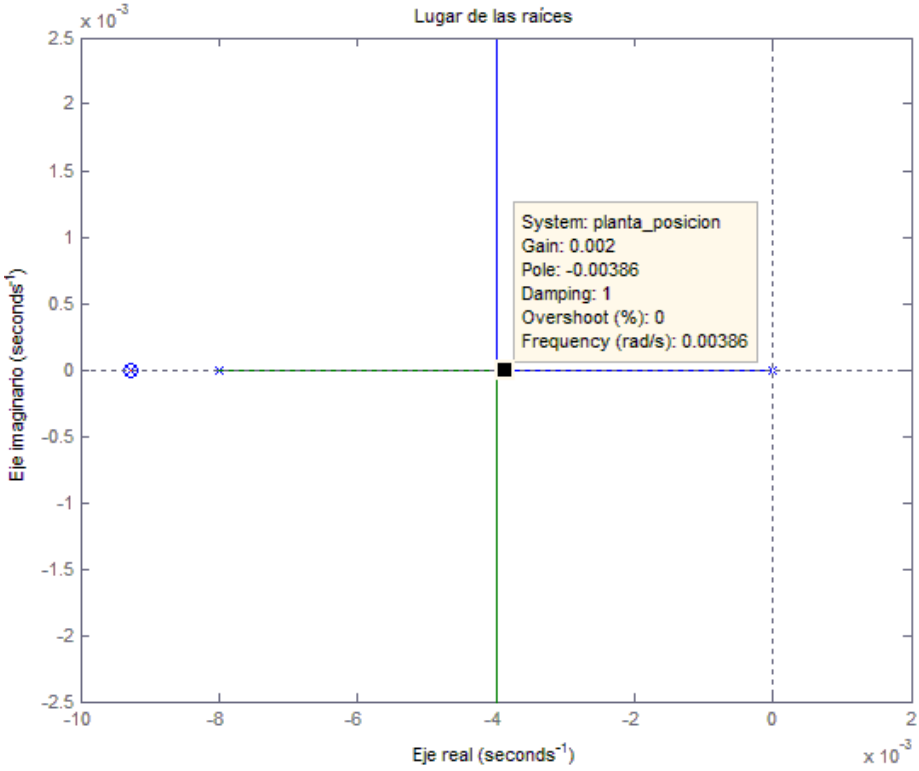


Figura 96. Lugar de las raíces eje X.

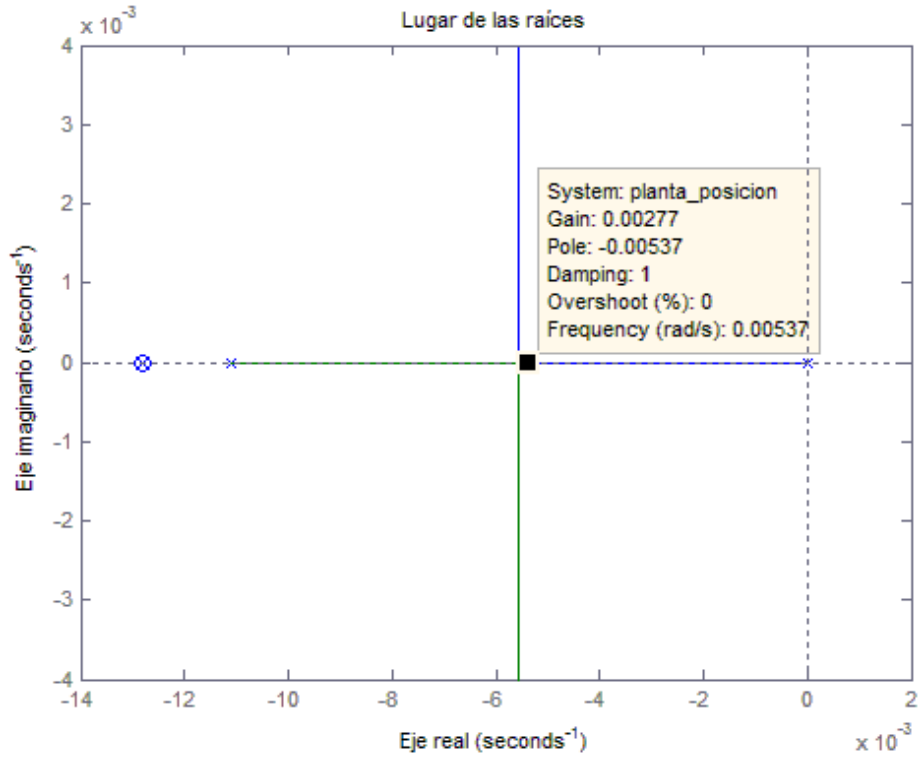


Figura 97. Lugar de las raíces eje Y.

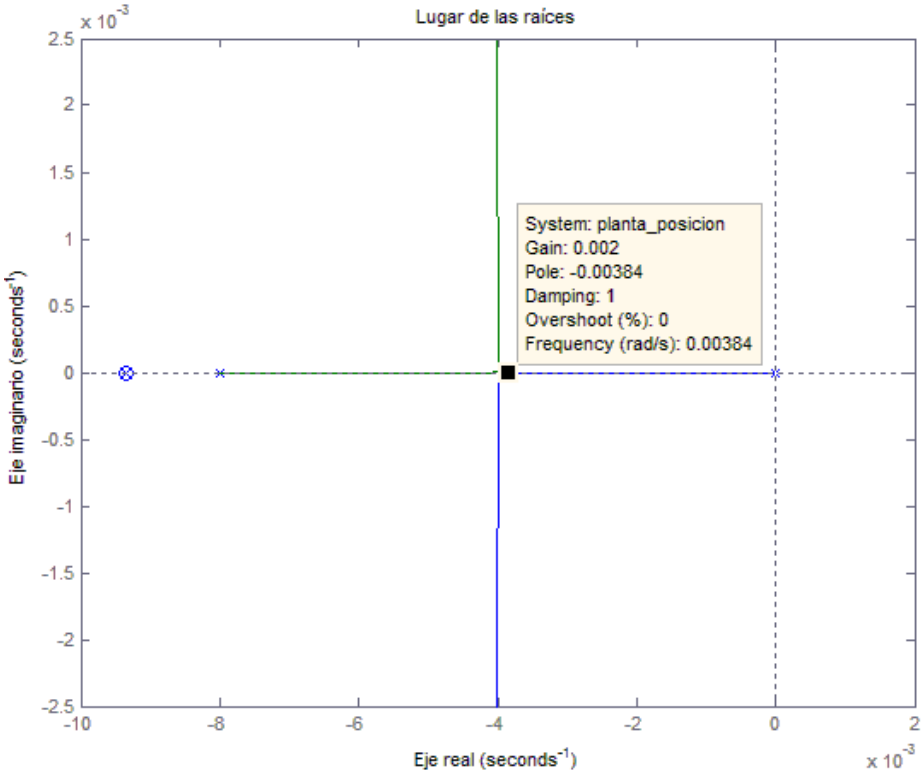


Figura 98. Lugar de las raíces eje Z.

Del lugar geométrico de las raíces para cada uno de los ejes, se puede apreciar que existen tres polos y un cero los cuales están ubicados a la izquierda sobre el eje real, por lo tanto el sistema está dentro de los criterios de estabilidad. La ganancia máxima que puede tener el controlador de posición para que el sistema sea sobre-amortiguado, se detalla en la Tabla 21.

Tabla 21. Máximo valor de constante de control de posición.

Eje	Ganancia Proporcional máxima	Ganancia Proporcional implementada
X	0,002	0,00165
Y	0,00277	0,00165
Z	0,002	0,0015

Al hacer varias pruebas, la ganancia se ajustó en 0,00165 para los ejes X y Y. Para el eje Z se ajustó en 0,0015, debido a la forma en la que está construido y por la tarea que hace (subir y bajar el efector final), por lo tanto no se hace necesario que el motor gire a alta

velocidad. Después de ajustar la ganancia del control de posición proporcional, se procede hacer la simulación de la respuesta a una entrada escalón de posición para conocer el comportamiento del sistema.

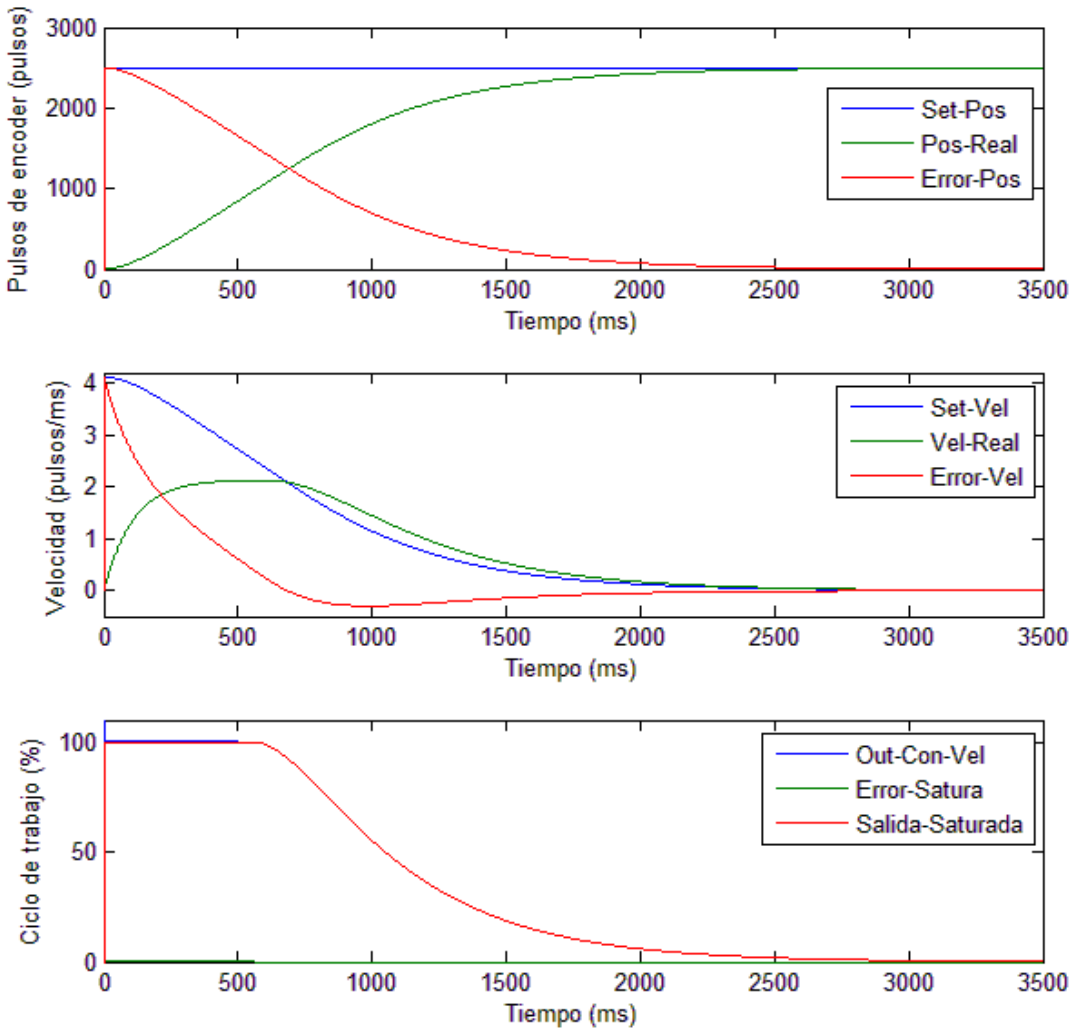


Figura 99. Simulación en Matlab del control de posición para el eje X.

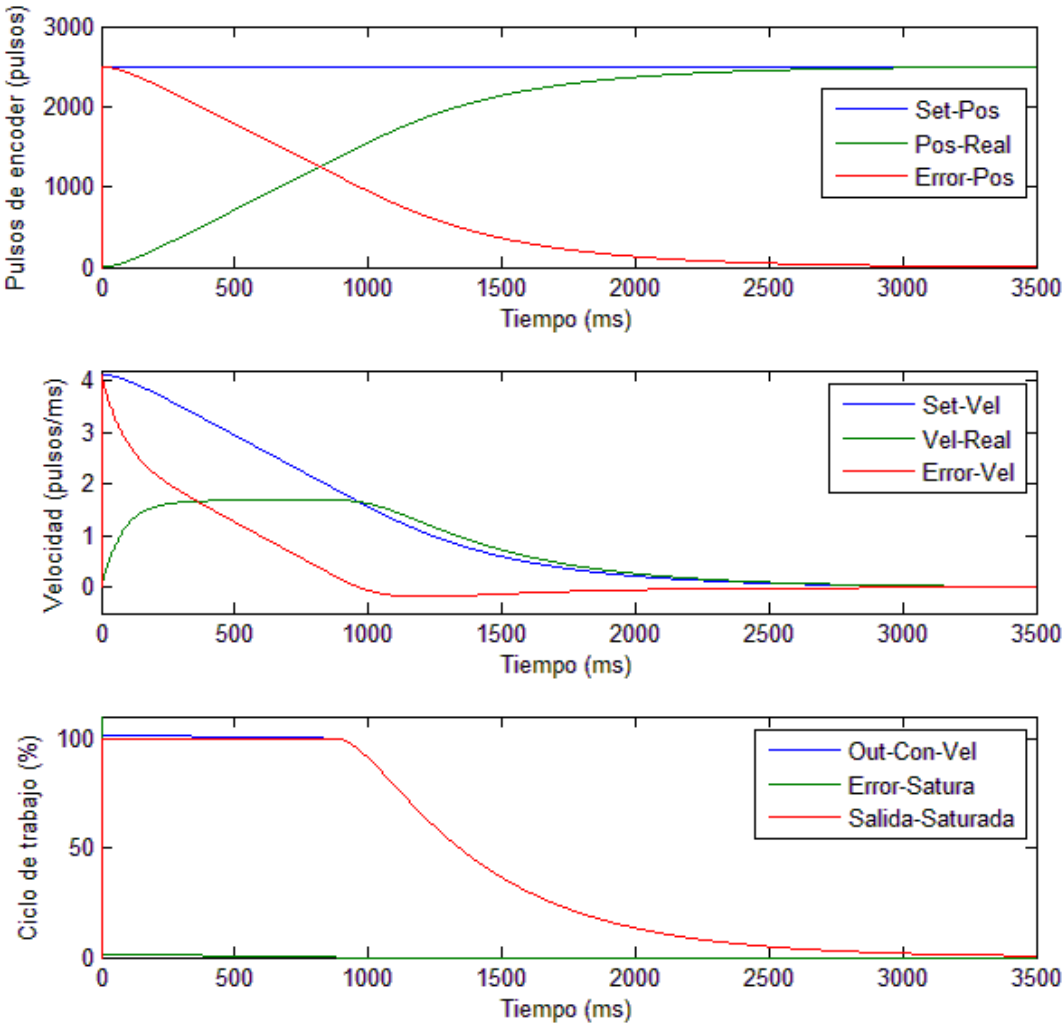


Figura 100. Simulación en Matlab del control de posición para el eje Y.

Para que los ejes X y Y se muevan a la misma velocidad, es necesario que el control de posición tenga la misma ganancia para que a coordenadas iguales de posición, se obtenga el mismo Set-Point de velocidad, por lo tanto a la salda del control de posición también se hace necesario implementar un límite máximo el cual alimenta al lazo de control de velocidad.

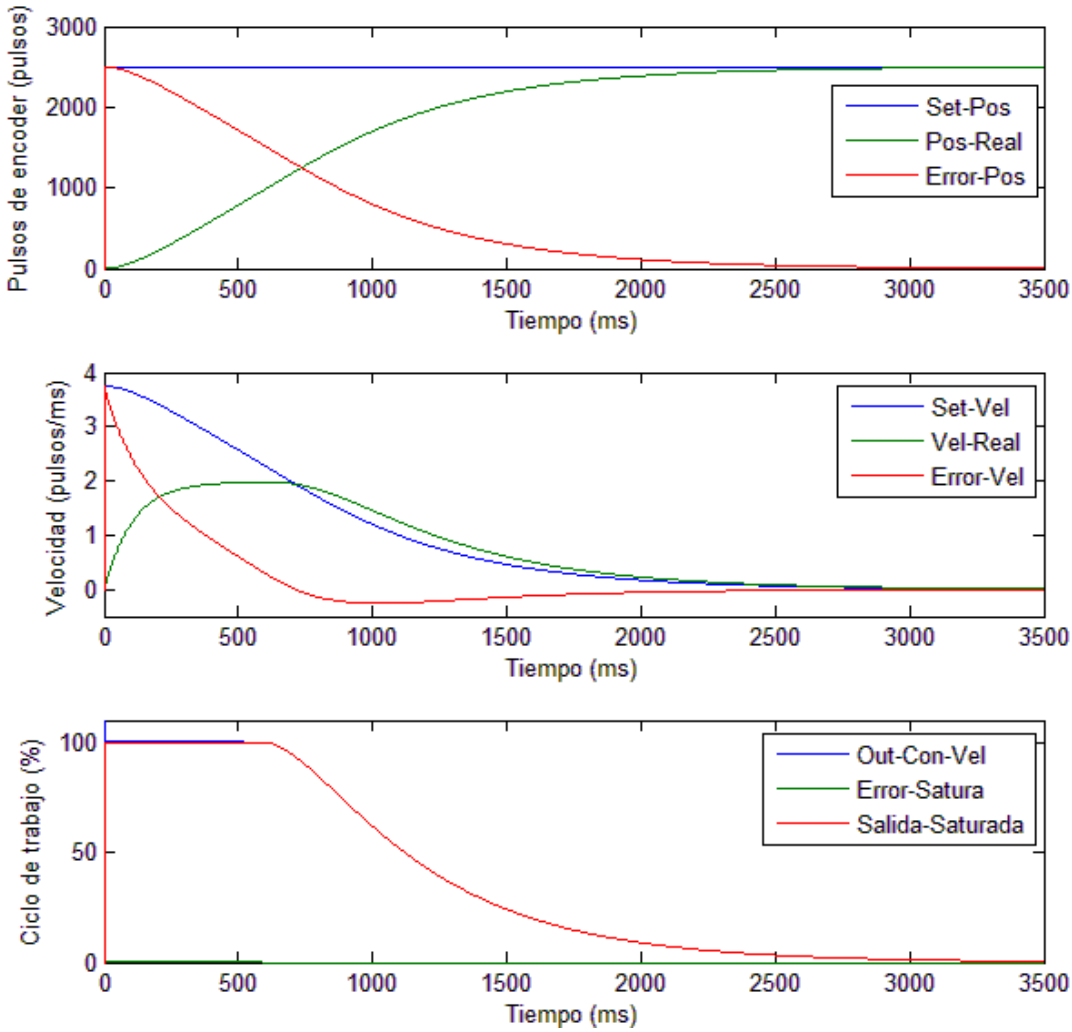


Figura 101. Simulación en Matlab del control de posición para el eje Z.

Una vez diseñado ambos controladores se procede a su implementación en el microcontrolador, la cual se detalla en la siguiente sección.

5.2.5 Implementación del control de velocidad y de posición

La implementación del control se hace por software en el microcontrolador; la arquitectura de software del Esclavo es más compleja, puesto que además de tener implementado un control de velocidad y de posición, también tiene implementado un sistema operativo en tiempo real o RTOS (Breijo) con el cual se controla el tiempo de ejecución del programa haciendo mucho más eficiente el control.

Una vez se inicializan las variables, módulos de hardware, puertos de entrada y salida; cada uno de los Esclavos calibra la posición inicial de la máquina, es decir la posición 0mm en cada uno de los ejes de movimiento. Luego entra en modo RTOS donde se encuentra el programa del controlador y a la espera de cualquier interrupción ya sea por la comunicación I2C o por los finales de carrera.

Para la subrutina de control, se hace necesario digitalizar el lazo de control de posición y de velocidad de la Figura 94. La relación de un control PI discreto viene dado por la transformada Z (Astrom & Wittenmark, 2011):

$$U(z) = E(z)K_c\left[1 + \frac{T}{T_i(1 - z^{-1})}\right]$$

También:

$$\frac{U(z)}{E(z)} = a + \frac{bT}{1 - z^{-1}}$$

Dónde:

$$a = K_c \qquad b = \frac{K_c}{T_i}$$

Para la implementación en el microcontrolador, se tiene la siguiente ecuación que representa al control PI del lazo de velocidad.

$$u_k = aE_k + I_{k-1} + bE_k h$$

Dónde:

- E_k es el error de velocidad.
- I_{k-1} es el valor del integrador de la iteración anterior.
- a es la ganancia del control proporcional y su valor se muestra en la Tabla 19.
- b es la relación $\frac{K_c}{T_i}$ y su valor se muestra en la Tabla 19.
- u_k es la salida del controlador de velocidad.

Para el lazo de control de posición, por ser un control solo proporcional, la ecuación que representa la salida del control es:

$$u_{k\_pos} = K_{c\_pos}E_{k\_pos}$$

Dónde:

- E_{k_pos}es el error de posición.
- K_{c_pos} es la ganancia del control de posición y se muestra en la Tabla 21.

- u_{k_pos} es la salida del controlador de posición.

La implementación del lazo de velocidad y de posición de la Figura 94 en tiempo discreto, se detalla en el código del microcontrolador en el ANEXO 6, donde se puede apreciar el Software de control implementado en el microcontrolador PIC18F4431 de cada eje.

6 PRUEBAS Y RESULTADOS

Como resultado principal de éste proyecto, se diseñó y se construyó una maquina CNC con una arquitectura convencional y modular basada en microcontroladores PIC. El montaje de todos los módulos (módulo de control y de potencia) se puede apreciar en la Figura 102. La máquina opera bajo los valores de posición gobernados desde la aplicación en Matlab y es capaz de fabricar tarjetas de circuitos impresos de buena calidad con una mínima intervención humana. La máquina es capaz de trabajar en un área superior a la de 1000cm^2, área que hace parte de las premisas de diseño de este proyecto.

Figura 102. Maquina CNC con arquitectura funcional diseñada y construida en éste Trabajo de Grado.

La aplicación desarrollada en Matlab, logra interpretar al menos tres tipos de fichero Gerber de software tales como “PCB Wizard”, “Eagle” y ”Proteus”. Cabe resaltar que aunque el Gerber es un estándar, no todos los programas de fabricación de circuitos impresos trabajan con la misma extensión de archivo y en el código pueden existir pequeñas variaciones como es el caso de una coma por punto, entre otros.

Por otra parte la aplicación realizada en Matlab tiene la capacidad de realizar copias (para la producción en masa) en base a las dimensiones de la placa de cobre virgen.

La electrónica de la maquina es capaz de realizar movimientos tan pequeños del orden de los 41.667 micrómetros, esta resolución se obtiene gracias a la resolución del encoder y al paso del tornillo. Es difícil medir el error de todo el sistema, debido a que la caja de engranaje del motor tiene un pequeño juego de valor desconocido.

Las pruebas se realizaron en varios niveles de dificultad, comenzando con circuitos simples de prueba que contienen pistas de diferentes medidas, comenzando con pistas de más de 3mm de espesor.

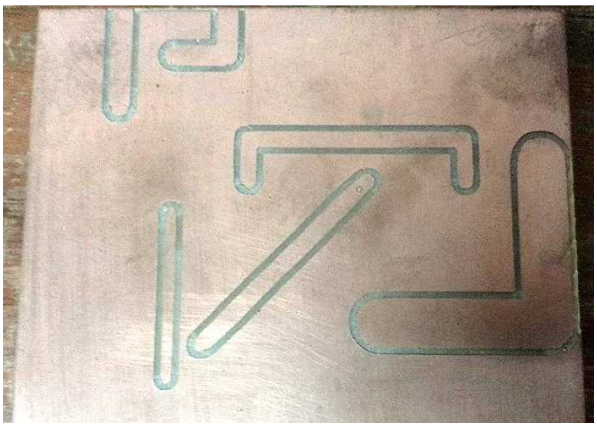


Figura 103. Primer nivel de dificultad, PCB con pistas de más de 3mm de espesor.

En la Figura 103 se puede apreciar el primer resultado obtenido, en comparación con la información ingresada al software. De éste primer resultado se puede apreciar gran precisión en los movimientos y el acabado de la placa. Luego de verificar algunas medidas se llegó a la conclusión que para la primera prueba existe un error de 1mm en el ancho de

las pistas. Se verifica la fresa y se encuentra la punta rota, lo cual provoca un ensanchamiento del área fresada, quitándole 0.5mm al borde de la pista.

También se concluye que la profundidad de fresado es demasiado alta y se calibra la máquina para evitar que vuelva a suceder. Luego de las nuevas pruebas, con las correcciones de la primera, se obtienen mejores resultados. Al verificar las medidas de las pistas se encuentra que el error es insignificante.

Luego se realizaron pruebas con pistas entre 1mm y 3mm de espesor y se obtuvieron los resultados de la Figura 104, donde se puede apreciar que se obtuvieron excelentes resultados en la prueba. El error en el diámetro de las pistas es insignificante y se debe a que resulta difícil calibrar la altura del fresado (para obtener el diámetro exacto de la pista), al ser la broca en "V", entre mayor sea la profundidad de fresado mayor será el diámetro de ruteo por lo tanto se verá afectado el diámetro de la pista.

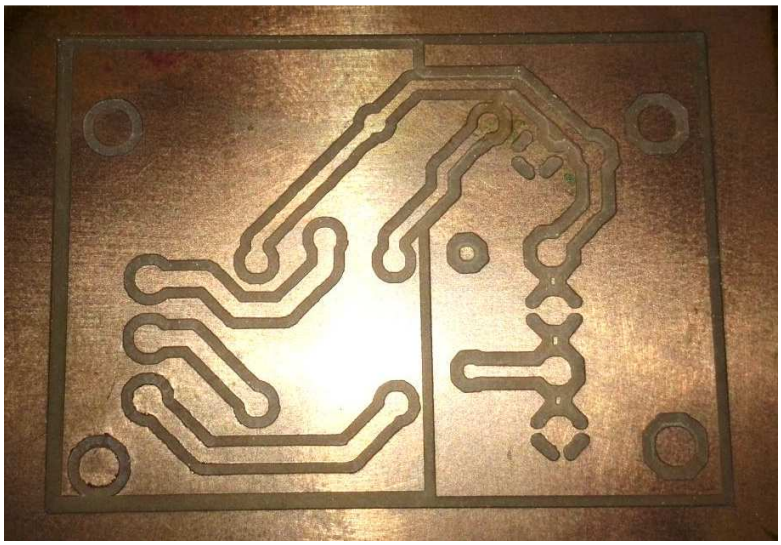


Figura 104. Segundo nivel de dificultad, PCB con pistas de más de 1mm y menos de 3mm de espesor.

Por otra parte las pistas diagonales pueden presentar algunas pequeñas malformaciones debido a que la dinámica del sistema del eje X y eje Y no son iguales, lo que sucede es que un eje llega más rápido a la posición que el otro teniendo como consecuencia estas pequeñas deformaciones que pueden ser despreciable.

Por último se hicieron pruebas con un tercer nivel de dificultad que consiste en hacer pistas de hasta menos de 1mm de espesor; los resultados se pueden apreciar en la Figura 105. De ésta última prueba se observa que tanto la maquina como el software son capaces de hacer circuitos impresos con pistas de menos de 1 mm de espesor. Además de esto es capaz de realizar los agujeros en la misma PCB.

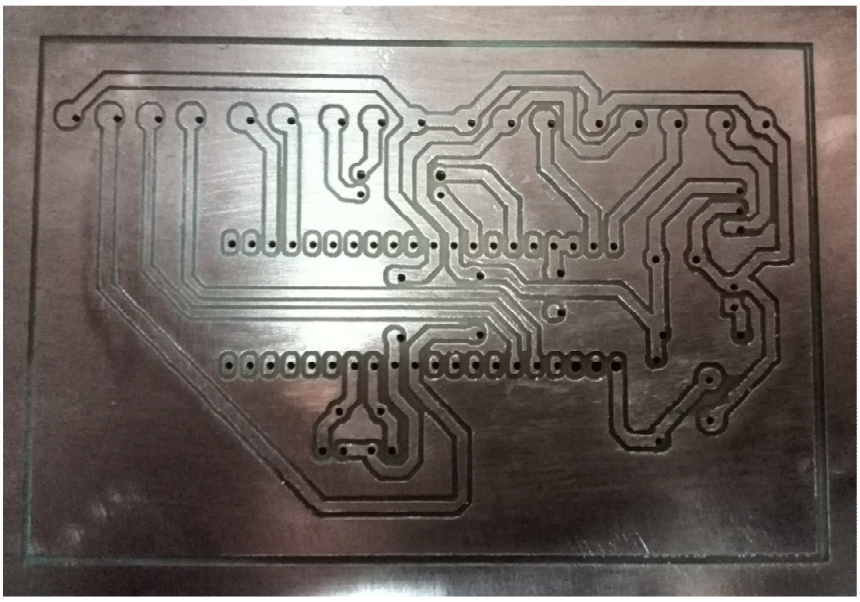


Figura 105. Tercer nivel de dificultad, PCB con pistas de menos de 1mm de espesor.

De la se puede apreciar que algunos agujeros quedaron fuera de la referencia, esto se debe a que primero se realizó la acción de rutero y luego se quitó la PCB de la máquina para observarla a detalle, luego se volvió a colocar pero la referencia no quedo en el mismo lugar teniendo como consecuencias un pequeño corrimiento en todos los agujeros.

También cabe resaltar que los agujeros de la parte inferior del circuito integrado se encuentran removidos debido a que la broca es de un diámetro mayor que el indicado para estos agujeros. Por esta razón se recomienda realizar ambas operaciones de manera continua sin mover la referencia de la PCB en la máquina, esto con el fin de obtener mejores resultados.

Se recomienda que al diseñar un circuito en cualquiera de los softwares con los que trabaja la maquina, se deje un borde de 5mm libre en la placa virgen que corresponden a

la distancia que requiere el sistema de fijación para que no se corra la placa durante la ejecución del trabajo.

El Gap en el programa, debe ser escogido de acuerdo al menor valor de Gap que tenga el PCB, esto se debe a que si hay pistas con un Gap menor que el seleccionado en la ventana de “Parámetros”, los contornos serán omitidos durante procesamiento del programa y estos aparecerán como una sola pista.

7 CONCLUSIONES

Se logró construir de manera funcional una maquina CNC con arquitectura convencional capaz de fabricar circuitos impresos a partir del código Gerber con una mínima intervención humana.

Durante la construcción de la máquina, hubo un pequeño sobrecosto debido a que no se contaba con la experiencia suficiente en el diseño de estas máquinas, por lo tanto hubo piezas que se tuvieron que volver a fabricar para poder obtener buenos parámetros de funcionalidad y de precisión.

Fue necesario encontrar un modelo matemático de cada uno de los ejes de la máquina, para poder sintonizar el control de posición y de velocidad de cada controlador. Esto con el fin de evitar sobre impulsos durante el recorrido del carro en el eje de movimiento; un sobre impulso en la posición puede representar un grave error en la fabricación de PCB´s

Para poder diseñar máquinas de alta precisión, es de vital importancia seleccionar componentes de buena calidad, que manejen una alta precisión y también que la alineación de dichos componentes a la hora del armado sea la mejor; esto con el fin de que el movimiento ofrezca la menor fricción posible y el efector final tenga un

movimiento suave (sin vibraciones) y se obtenga un producto final (en este caso fabricación de PCB´s) con una buena exactitud y precisión.

Con el diseño de la electrónica modular, se obtuvieron una serie de ventajas al momento de solucionar problemas durante el armado de la máquina, ya que solo con intercambiar una tarjera de un controlador por otra se puede determinar el correcto funcionamiento del mismo sin necesidad de llegar a pruebas tan engorrosas con equipos sofisticados antes de dar con el problema. Esto representa una ventaja para el mantenimiento y reparaciones de la máquina.

La arquitectura mecánica y la electrónica de la máquina, se diseñó para que pudiera usarse en otro tipo de aplicaciones de las maquinas CNC con arquitectura convencional, solo basta con modificar o agregar funciones adicionales al programa en Matlab. Aplicaciones como la impresora 3D, perforación y corte de piezas pueden ser implementadas con un mínimo de configuración del hardware y software de toda la máquina.

Antes de empezar a ejecutar la función de ruteo desde la aplicación en Matlab, se hace necesario calibrar la altura de la broca en “V” en el efector final; debido a que la maquina baja siempre a la misma altura, una mala calibración puede dañar a la placa de cobre virgen e incluso puede deteriorar la broca y desajustar la arquitectura mecánica de toda la máquina.

Dada la complejidad del diseño, resulto muy difícil poder simular todo el conjunto en un solo Software de simulación, actualmente en el mercado no existe un software de simulación que sea capaz de simular el control, la comunicación USB y los mecanismos en tiempo real. Por lo tanto la simulación se hizo por sistemas y en diferente software; esto trae inconvenientes a la hora de ensamblar y conectar los diferentes sistemas debido a que hay problemas que solo se pueden identificar al momento de hacer el montaje.

La fabricación de piezas para las máquinas de precisión como son las maquinas CNC, se deben de realizar en talleres especializados y con personal calificado ya que un pequeño error de medida puede generar un grave error en la precisión de la máquina. Este fue el principal problema que se obtuvo cuando se empezaron a fabricar las barras guías, cojinetes de deslizamiento, tornillo y tuerca rosca ACME.

La función de ruteo y perforación se deben realizar de manera continua sin remover la baquelita de los puntos de fijación de la maquina; esto con el fin de que la referencia de ruteo sea la misma para los agujeros y estos queden lo más centrado posibles según el diseño de la PCB.

8 BIBLIOGRAFÍA

- Astrom, K. J., & Wittenmark, B. (2011). *Computer-Controlled Systems: Theory and Design.* Dover Publications.
- Bauker. (s.f.). *Manual de Uso Mini Esmerilador Modelo MP170.* Recuperado el 04 de 05 de 2014, de http://www.bauker.com/manuales/esmeriles_pulidoras/mini_esmerilador_MP170.pdf
- Bishop, R. H. (2006). *Mechatronics: an introduction.* London: CRC Press.
- Breijo, E. G. (s.f.). *Compilador C CCS y Simulador Proteus para Microcontroladores PIC* (2 ed.). Marcombo, S.A.
- Correanayak. (s.f.). *Fresadoras de Puente fijo y mesa móvil*. Recuperado el 4 de Mayo de 2014, de http://www.correanayak.eu/es/gama-de-productos/1183388604-mandrinadoras-puente-fijo-y-mesa-movil/1184959697-serie-fp/)
- Correanayak. (s.f.). *Fresadoras de Puente móvil y mesa fija*. Recuperado el 4 de Mayo de 2014, de http://www.correanayak.eu/es/gama-de-productos/1184960198-mandrinadoras-puente-movil-y-mesa-fija/1184960263-fpm/
- Coughran, M. T. (s.f.). *Lambda Tuning—the Universal Method for PID Controllers in Process Control.* Recuperado el 4 de Mayo de 2014, de http://www.controlglobal.com/assets/13WPpdf/131022-coughran-controllers.pdf
- Elaboracion, propia. (s.f.).
- Fairchild Semiconductor. (s.f.). *LM78XX / LM78XXA Data Sheet.* Recuperado el 04 de 05 de 2014, de http://www.fairchildsemi.com/ds/LM/LM7805.pdf
- Grupo SKF. (Diciembre de 2003). *Husillos de bolas.* Francia: Grupo SKF.
- Gutierrez, M. T., Rodriguez, R. E., & Garzon, A. J. (2008). Prospectiva De La Electrónica En Colombia. *1 Congreso Internacional de Gestión Tecnológica e Innovación*, (pág. 66). Bogota.

- Hood-Daniel, a. J. (2009). *Build Your Own CNC Machine.* United States: Apress.
- Khandpur, R. S. (2006). *Printed Circuit Boards: Design, Fabrication and Assembly.* New York, United States: McGraw-Hill.
- Melón, M. G., Ballester, V. C., & Navarro, T. G. (2001). *Metodología del diseño industrial.* Universidad Politécnica de Valencia.
- Microchip Technology Inc. (2009). *PIC18F2455/2550/4455/4550 Data Sheet.* Recuperado el 04 de 05 de 2014, de http://ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf
- Microchip Technology Inc. (2010). *PIC18F2331/2431/4331/4431 Data Sheet.* Recuperado el 04 de 05 de 2014, de http://ww1.microchip.com/downloads/en/DeviceDoc/39616d.pdf
- Morse Cross. (s.f.). *Piñon cremallera (Rack and pinion)*. Recuperado el 04 de 05 de 2014, de Cross Morse: http://www.cross-morse.co.uk/Spanish/pdf/Steel%20RacksSP.pdf
- Norton, R. L. (s.f.). *Diseño de Maquinas.* Pearson.
- Padilla, C. M. (s.f.). *El Prisma, Control Numérico - CNC*. Recuperado el 4 de Mayo de 2014, de http://www.elprisma.com/apuntes/ingenieria_mecanica/controlnumericocnc/
- Pololu Robotics & Electronics. (s.f.). *30:1 Metal Gearmotor 37Dx52L mm with 64 CPR Encoder*. Recuperado el 04 de 05 de 2014, de http://www.pololu.com/product/1443
- Pool, G. (8 de Abril de 2010). *Comunicacion entre MATLAB y PIC de MICROCHIP usando puerto USB*. Recuperado el 01 de 05 de 2014, de Matlab Central: http://www.mathworks.com/matlabcentral/fileexchange/24417-comunicacion-entre-matlab-y-pic-de-microchip-usando-puerto-usb
- Richard G. Budynas, J. K. (s.f.). *Diseño en ingeniería mecánica de shigley.* Mc Graw Hill.

- Sankyo Oilless Industry, Inc. (s.f.). *Guía de buje*. Recuperado el 04 de 05 de 2014, de https://sites.google.com/a/sankyo-oilless.com/sankyo-oilless-english/download/ep_02#p01
- Silva, C. W. (2008). *Mechatronic systems: Devices, design, control, operation and monitoring.* London: CRC Press.
- SKF, Grupo. (2011). *Unidades y rodamientos lineales para ejes.* Suecia: Grupo SKF.
- ST Microelectronics. (2000). *L298 Data Sheet.* Recuperado el 4 de Mayo de 2014, de http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00000240.pdf
- Suh, S.-H., Kang, S., Chung, D.-H., & Stroud, I. (2008). *Theory and Design of CNC Systems.* London: Springer.
- Surtirodamientos. (s.f.). *Transmision de Potencia*. Recuperado el 04 de 05 de 2014, de http://www.surtirodamientos.com/transmision.html
- Wikimedia. (s.f.). *Imagen L298.* Recuperado el 4 de Mayo de 2014, de http://commons.wikimedia.org/wiki/File:L298_IMGP4533_wp.jpg

ANEXO 1. Programa implementado en Matlab para la caracterización.

```
clear all;
close all;
clc;
loadlibrary mpusbapi _mpusbapi.h alias libreria
data_in = zeros(1,64,'uint8');
data_out = zeros(1,64,'uint8');
vid_pid_norm = libpointer('int8Ptr',[uint8('vid_04d8&pid_000b') 0]);
out_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);
in_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);

[conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm)

if conectado == 1

    [my_out_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, out_pipe,
uint8(0), uint8 (0));
    [my_in_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, in_pipe, uint8
(1), uint8 (0));

    cabecera=0;
    while (cabecera<20)

    cabecera=input('Digite 4 o 5 (o digite >10 para salir): ');
    if cabecera==4

        set_point=input('Digite velocidad en RPM: [0-100]: ');
        giro=input('Digite 1 o 2 para el sentido de giro: ');
        ciclo=uint16(set_point*16383/100);
        datoL=uint8(bitshift(ciclo,0,8));
        datoH=uint8(bitshift(ciclo,-8,8));
        data_out (1)=uint8(cabecera);
        data_out (5)=datoH;
        data_out (6)=datoL;
        data_out (7)=giro;
        calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(64), uint8(64),
uint8(10)) % Se envia el dato al PIC
        [aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64),
uint8(64), uint8(10)); % Se recibe el dato que envia el PIC
        data_out(1:8)
        data_in()
    elseif cabecera==5
        for h=1:12
            data_out (1)=uint8(cabecera);
            data_out (5)=h;
            data_out (6)=0;

            calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(64), uint8(64),
uint8(10))
            data_out(1:8)

            [aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in,
uint8(64), uint8(64), uint8(10));
            data_in()


            if data_out(5)==1
                datoMSB1=data_in(1:50);
            elseif data_out(5)==2
                datoMSB2=[datoMSB1,data_in(1:50)];
            elseif data_out(5)==3
                datoMSB3=[datoMSB2,data_in(1:50)];
            elseif data_out(5)==4
                datoMSB4=[datoMSB3,data_in(1:50)];
            elseif data_out(5)==5
                datoMSB5=[datoMSB4,data_in(1:50)];
            elseif data_out(5)==6
                datoMSB6=[datoMSB5,data_in(1:50)];
            elseif data_out(5)==7
                datoLSB1=data_in(1:50);
            elseif data_out(5)==8
                datoLSB2=[datoLSB1,data_in(1:50)];
```

```
            elseif data_out(5)==9
                datoLSB3=[datoLSB2,data_in(1:50)];
            elseif data_out(5)==10
                datoLSB4=[datoLSB3,data_in(1:50)];
            elseif data_out(5)==11
                datoLSB5=[datoLSB4,data_in(1:50)];
            elseif data_out(5)==12
                datoLSB6=[datoLSB5,data_in(1:50)];
            end
            pause(0.1);
        end

        dato_pos=bin2dec([dec2bin(datoMSB6,8),dec2bin(datoLSB6,8)]);
        dato_vel(1)=0;
        t(1)=0;
        for i=2:300
          dato_vel(i)=((dato_pos(i)-dato_pos(i-1)))*(60)/(1856*0.02);%4331, 1856
          t(i)=(t(i-1)+20);
        end
        figure(1)
        plot(t,dato_vel);
        save 'datos_motor.mat' dato_vel t dato_pos;
     end
    end


    calllib('libreria', 'MPUSBClose', my_in_pipe);
    calllib('libreria', 'MPUSBClose', my_out_pipe);
 end
close all
```

ANEXO 2. Programa implementado en el PIC Maestro para la caracterización.

```
//Declaracion de librerias, FUSES de programacion y variables de programa
#include <MASTER_VARIOS.h>
#define USB_HID_DEVICE     FALSE     //deshabilitamos el uso de las directi
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK  //turn on EP1(EndPoint1) for IN
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK  //turn on EP1(EndPoint1) for OU
#define USB_EP1_TX_SIZE    64       //size to allocate for the tx endpoint
#define USB_EP1_RX_SIZE    64       //size to allocate for the rx endpoint

//Declaracion de librerias
#include <pic18_usb.h>
#include <usb_desc_scope_CNC.h>
#include <usb.c>

//Declaracion de registros
#define  EN_Z    PIN_B5
#define  EN_Y    PIN_B6
#define  EN_X    PIN_B7
#define  LED_ON  output_high
#define  LED_OFF output_low

//Configuracion de modulos
#use i2c(Master,force_hw,sda=PIN_B0,scl=PIN_B1,FAST=400000)

//Declaracion de funciones
void write(BYTE,BYTE);
BYTE read(BYTE);

//Declaracion de variables
int8 dato_in_CNC[64];
int8 dato_out_CNC[64];
int8 i=0;
int8 datos[64];
int8 datos_z[100];

//Funcion principal
void main()
{
   //Inicializamos los arreglos de datos
   for(i=0;i<64;i++)
   {
      datos[i]=0;
      dato_in_CNC[i]=0;
      dato_out_CNC[i]=0;
   }
      for(i=0;i<60;i++)
   {
      datos_z[i]=0;
   }
   usb_init();           // inicializamos el USB
   usb_task();           // habilita periferico usb e interrupciones
   usb_wait_for_enumeration(); // esperamos hasta que el PicUSB sea configu
//Se configuran los puertos y modulos
   SET_TRIS_B(0xff);
   SET_TRIS_D(0x00);
   OUTPUT_D(0x00);
   delay_ms(250);
   do
   {
      if(usb_enumerated())    // si el Pic está configurado via USB
      {
         if (usb_kbhit(1))    // si el endpoint de salida contiene datos del
         {
            dato_in_CNC[0]=0;
            usb_get_packet(1, dato_in_CNC, 64);
            for(i=0;i<64;i++)
            {
               datos[i]=dato_in_CNC[i];
            }
```

```
            if(datos[0]==4)
             {
             //Se envia informacion al Esclavo por i2c y recibe del PC po
                write(4,0xA0);
                usb_put_packet(1, dato_in_CNC, 64, USB_DTS_TOGGLE);
             }
             else if (datos[0]==5)
             {
             //Se envia informacion al Esclavo por i2c y envia al PC por
                write(4,0xA0);
                read(0xA1);
                for(i=0;i<50;i++)
                {
                   dato_out_CNC[i]=datos_z[i];
                }
                usb_put_packet(1, dato_out_CNC, 64, USB_DTS_TOGGLE);
             }
          }
       }
    }while(true);
 }

//Subrutina para leer la trama de datos del i2c
BYTE read(BYTE dir)
{
   BYTE dato_in;
   i2c_start();       //Inicializa la transmisión
   i2c_write(dir);    //Direccion del esclavo a comunicar
   for(i=0;i<49;i++)
   {
      datos_z[i]=i2c_read(1); //Byte recibidos
   }
   datos_z[i]=i2c_read(0);
   i2c_stop();
   return dato_in;
}

//Subrutina para enviar la trama de datos al esclavo por el i2c
void write(BYTE inieje,BYTE dir)
{
   i2c_start();          //Inicializa la transmisión
   i2c_write(dir);       //Direccion del esclavo a comunicar
   i2c_write(datos[0]);  // Numero de Bytes a enviar
   for(i=inieje; i<datos[0]+inieje; i++)
   {
      i2c_write(datos[i]);  //Dato a enviar;
   }
   i2c_stop();
   return;
}
```

ANEXO 3. Programa implementado en el PIC Esclavo para la caracterización.

```
//Declaracion de librerias
#include <18F4431.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

//Declaracion de registros
#byte QEICON = 0xFB6
#byte POSCNTH = 0xF67
#byte POSCNTL = 0xF66
#byte MAXCNTH = 0xF65
#byte MAXCNTL = 0xF64
#byte DFLTCON = 0xF60
#byte VELRH = 0xF69
#byte VELRL = 0xF68
#byte T5CON = 0xB7

//Declaracion de los FUSES de programacion
#FUSES NOWDT       //No Watch Dog Timer
#FUSES WDT128      //Watch Dog Timer uses 1:128 Postscale
#FUSES HS          //Crystal osc <= 4mhz for PCM/PCH , 3mhz to 10 mhz for PC
#FUSES MCLR
#FUSES NOCPB
#FUSES NOPROTECT
#FUSES NOCPD
#FUSES NOBROWNOUT
#FUSES NOLVP
#FUSES NOIESO
#FUSES NOPUT
#FUSES NOWRT
#FUSES NOEBTR
#FUSES NOPWMPIN
#FUSES SSP_RC //#FUSES SSP_RC para i2c en puerto C
#FUSES NODEBUG
#FUSES NOFCMEN
#FUSES HPOL_LOW
#FUSES LPOL_LOW

//Configuracion de modulos
#use delay(clock=20000000)
#use i2c(slave,fast,sda=PIN_C4,scl=PIN_C5,address=0xA0,FORCE_HW)
#use rtos(timer=0,minor_cycle=10ms)//para el tiempo de muestreo

//Declaracion de funciones
void leer(void);
void CONFIGPCPWM(void);
void configencoder(void);
void write(void);
void pid(void);

//Declaracion de variables
int8 curvah[300];
int8 curval[300];
int8 state;
int16 k=0;
int16 L=0;
int8 trama_control[25]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int16 j=0;
int16 SP=0; //Set point de posicion

//Declaracion de tareas del RTOS
#task(rate=20ms,max=10ms)       //Tarea del PID
void pid();

//Interrupcion I2C
#INT_SSP
void ssp_interupt ()
{
```

```
   state = i2c_isr_state();
   if(state < 0x80)     //Maestro esta enviando datos
   {
      leer();//Se ejecuta la funcion leer
   }
   else if(state == 0x80) //Maestro pide un dato
   {
      write();  //Se ejecuta la funcion escribir
   }
}

//Funcion principal
void main()
{
//Se configuran los puertos y modulos
   SET_TRIS_B(0x00);
   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_OFF);
   enable_interrupts(GLOBAL);
   enable_interrupts(INT_SSP);
   disable_interrupts(INT_IC2QEI);
   CONFIGPCPWM();
   configencoder();
   POSCNTH=0;
   POSCNTL=0;
   output_low(PIN_B7);
   set_power_pwm0_duty(0);
   set_power_pwm2_duty(0);
//Se inicializa el arreglo de datos en cero
   for (L=0;L<300;L++){
      curvah[L]=0;
      curval[L]=0;
   }
   rtos_run();//Inicia el RTOS
}

//Funcion que se ejecuta dentro del RTOS y es donde se ejecuta todo el cont
void pid(void)
{
   if (trama_control[1]==4 && trama_control[4]==3)
   {
      set_power_pwm0_duty(0);
      set_power_pwm2_duty(16000);
   }
   if(trama_control[1]==4 && trama_control[4]!=3 && j<300)
   {
      if(j==0){SP=8191;}//50
      if(j==37){SP=13106;}//80
      if(j==74){SP=6553;}//40
      if(j==111){SP=16383;}//100
      if(j==148){SP=9828;}//60
      if(j==185){SP=5734;}//35
      if(j==222){SP=14744;}//90
      if(j==259){SP=11468;}//70
      if(j==240){SP=6553;}//40
      if(j==270){SP=11468;}//70
      SP=make16(trama_control[2],trama_control[3]);
      set_power_pwm0_duty(SP);
      set_power_pwm2_duty(0);
      curvah[j]=POSCNTH;
      curval[j]=POSCNTL;
      j=j+1;
   }
   if (trama_control[1]==4 && trama_control[4]!=3 && j==300)
   {
      trama_control[1]=0;
      j=0;
      set_power_pwm0_duty(0);
```

```
      set_power_pwm2_duty(0);
   }
   output_toggle(PIN_B7);
}

//Subrutina para leer la trama de datos del i2c
void leer(void)
{
   //Direccion
   while(!i2c_poll());
   trama_control[0]=i2c_read();
   //Numero de Byte
   while(!i2c_poll());
   trama_control[1]=i2c_read();
   for(k=2; k<trama_control[1]+2; k++)
   {
      while(!i2c_poll());
      trama_control[k]=i2c_read();
   }
   return;
}

//Subrutina para enviar la trama de datos al maestro por el i2c
void write(void)
{
   if (trama_control[1]==5 && trama_control[2]==1){
      for (k=0;k<50;k++)
      {
         i2c_write(curvah[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==2){
      for (k=50;k<100;k++)
      {
         i2c_write(curvah[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==3){
      for (k=100;k<150;k++)
      {
         i2c_write(curvah[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==4){
      for (k=150;k<200;k++)
      {
         i2c_write(curvah[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==5){
      for (k=200;k<250;k++)
      {
         i2c_write(curvah[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==6){
      for (k=250;k<300;k++)
      {
         i2c_write(curvah[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==7){
      for (k=0;k<50;k++)
      {
         i2c_write(curval[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==8){
      for (k=50;k<100;k++)
      {
         i2c_write(curval[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==9){
      for (k=100;k<150;k++)
      {
```

```
         i2c_write(curval[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==10){
      for (k=150;k<200;k++)
      {
         i2c_write(curval[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==11){
      for (k=200;k<250;k++)
      {
         i2c_write(curval[k]);
      }
   }else if(trama_control[1]==5 && trama_control[2]==12){
      for (k=250;k<300;k++)
      {
         i2c_write(curval[k]);
      }
   }
   return;
}

//Subrutina para la configuracion del modulo PWM
void CONFIGPCPWM(void)
{
   setup_power_pwm_pins(PWM_COMPLEMENTARY,PWM_COMPLEMENTARY,PWM_OFF,PWM_OFF
   setup_power_pwm(PWM_CLOCK_DIV_4|PWM_FREE_RUN,1,0,4095,0,1,0);
   return;
}

//Subrutina para la configuracion del modulo encoder
void configencoder(void)
{
   QEICON  = 0b10111000;//Configuracion del modulo del encoder
   POSCNTH = 0;//Posicion del eje
   POSCNTL = 0;//Posicion del eje
   MAXCNTH = 0xFF;//Maixmo Valor del contador
   MAXCNTL = 0xFF;//Maixmo Valor del contador
   DFLTCON = 0b00111000;//Activar Filtro
   return;
}
```

ANEXO 4. Programa implementado en Matlab para el control final.

INTERFAZ DE USUARIO

Ventana principal del software:

```
function varargout = UTBCNC(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @UTBCNC_OpeningFcn, ...
                   'gui_OutputFcn',  @UTBCNC_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 UTBCNC is made visible.
function UTBCNC_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 UTBCNC (see VARARGIN)
% Choose default command line output for UTBCNC
handles.output = hObject;
set(handles.figure1,'CloseRequestFcn',[]);
%Se Agregan los directorios
path(path,'library\');
path(path,'temp\');
path(path,'data\');
path(path,'library\script');
path(path,'library\hmi\');
path(path,'library\function\');

handlesmadre=handles;
save data\manejadores.mat handlesmadre;
pausa=0;
stop=0;
ultimalinea=1;
save data\detener.mat stop pausa ultimalinea;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes UTBCNC wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = UTBCNC_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

% --------------------------------------------------------------------
function op_cerrar_Callback(hObject, eventdata, handles)
% clear all
% clc
set(handles.figure1,'CloseRequestFcn','closereq');
close('UTBCNC');

% --------------------------------------------------------------------
function op_abrir_gerber_Callback(hObject, eventdata, handles)
% Se carga el archivo Gerber
[filename pathGerber]= uigetfile({'*.gb1';'*.GBL';'*CADCAM Bottom Copper.TXT'},'Seleccione
el archivo gerber PCB');%carga el archivo Gerber
```

```
if filename~=0 %cuando se pulsa en (Aceptar filename=1) (Cancelar filename=0)
    % Se cargan los parametros de GAP, Broca, Resolucion
    delete('data\datos_copias.mat');
    load parametros.mat;
    % Se ejecuta el script de load_file donde se hace todo el procedimiento
    load_file;%llamado a Load_file
end

% --------------------------------------------------------------------
function op_ruteo_copias_Callback(hObject, eventdata, handles)
%set(handles.figure1,'CloseRequestFcn',[])
%set(handles.figure1,'CloseRequestFcn','closereq')
set(handles.figure1,'visible','off');
Ruteocopias;%llamado a opcion de menu Ruteocopias 'configuracion de copias PCB'

% --- Executes on button press in rutear_machine.
function rutear_machine_Callback(hObject, eventdata, handles)
set(handles.menu_archivo,'enable','off');
set(handles.menu_maquina,'enable','off');
set(handles.menu_modo_ejecucion,'enable','off');
set(handles.perforar_machine,'enable','off');
set(handles.op_ruteo_copias,'enable','off');
opc=questdlg('¿Desea enviar estos datos a la maquina?','Enviar datos?','Si','No','No');
if strcmp(opc,'No')
    set(handles.menu_archivo,'enable','on');
    set(handles.menu_maquina,'enable','on');
    set(handles.menu_modo_ejecucion,'enable','on');
    set(handles.perforar_machine,'enable','on');
    set(handles.pausa_machine,'enable','off');%deshabilita boton pausar
    set(handles.stop_machine,'enable','off');%deshabilita boton detener
    return;
elseif strcmp(opc,'Si')
    operacion='r';
    script_encoder;
    %% Ejecutamos el script_usb el cual se encarga de transmitir los datos
    cabeza=4;
    set(handles.pausa_machine,'enable','on');%habilita boton pausar
    set(handles.stop_machine,'enable','on');%habilita boton detener
    script_usb;
    if exist('mensaje')==1
        status=close(mensaje);
    end
    set(handles.op_ruteo_copias,'enable','on');
    set(handles.menu_archivo,'enable','on');
    set(handles.menu_maquina,'enable','on');
    set(handles.menu_modo_ejecucion,'enable','on');
    set(handles.perforar_machine,'enable','on');
    set(handles.pausa_machine,'enable','off');%deshabilita boton pausar
    set(handles.stop_machine,'enable','off');%deshabilita boton detener
end
guidata(hObject, handles);

% --------------------------------------------------------------------
function menu_archivo_Callback(hObject, eventdata, handles)

% --------------------------------------------------------------------
function menu_maquina_Callback(hObject, eventdata, handles)

% --- Executes on button press in pausa_machine.
function pausa_machine_Callback(hObject, eventdata, handles)
pausa=1;
save data\detener.mat pausa -append;
set(handles.pausa_machine,'enable','off');%deshabilita boton pausar
set(handles.stop_machine,'enable','off');%deshabilita boton detener
% --- Executes on button press in stop_machine.
function stop_machine_Callback(hObject, eventdata, handles)
stop=1;
ultimalinea=1;
save data\detener.mat stop ultimalinea -append;
set(handles.pausa_machine,'enable','off');%deshabilita boton pausar
```

```
set(handles.stop_machine,'enable','off');%deshabilita boton detener
%guidata(hObject, handles);

% --------------------------------------------------------------------
function menu_modo_ejecucion_Callback(hObject, eventdata, handles)

% --------------------------------------------------------------------
function op_modo_seguro_Callback(hObject, eventdata, handles)
set(handles.op_parametros,'enable','off');

% --------------------------------------------------------------------
function op_modo_avanzado_Callback(hObject, eventdata, handles)
% hObject    handle to op_modo_avanzado (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1,'visible','off');
password;

% --------------------------------------------------------------------
function op_propiedades_gerber_Callback(hObject, eventdata, handles)
% hObject    handle to op_propiedades_gerber (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --------------------------------------------------------------------
function op_parametros_Callback(hObject, eventdata, handles)
% hObject    handle to op_parametros (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1,'visible','off');
parametros;

% --- Executes on button press in perforar_machine.
function perforar_machine_Callback(hObject, eventdata, handles)
set(handles.menu_archivo,'enable','off');
set(handles.menu_maquina,'enable','off');
set(handles.menu_modo_ejecucion,'enable','off');
set(handles.rutear_machine,'enable','off');
set(handles.op_ruteo_copias,'enable','off');
opc=questdlg('¿Desea enviar estos datos a la maquina?','Enviar datos?','Si','No','No');
if strcmp(opc,'No')
    set(handles.menu_archivo,'enable','on');
    set(handles.menu_maquina,'enable','on');
    set(handles.menu_modo_ejecucion,'enable','on');
    set(handles.rutear_machine,'enable','on');
    set(handles.pausa_machine,'enable','off');%deshabilita boton pausar
    set(handles.stop_machine,'enable','off');%deshabilita boton detener
    return;
elseif strcmp(opc,'Si')
    operacion='p';
    script_encoder
    %% Ejecutamos el script_usb el cual se encarga de transmitir los datos
    cabeza=4;
    script_usb;
    if exist('mensaje')==1
        close(mensaje);
    end
    set(handles.op_ruteo_copias,'enable','on');
    set(handles.menu_archivo,'enable','on');
    set(handles.menu_maquina,'enable','on');
    set(handles.menu_modo_ejecucion,'enable','on');
    set(handles.rutear_machine,'enable','on');
    set(handles.pausa_machine,'enable','off');%deshabilita boton pausar
    set(handles.stop_machine,'enable','off');%deshabilita boton detener
end
guidata(hObject, handles);

% --- Executes when selected object is changed in uipanel_botones_tabla.
function uipanel_botones_tabla_SelectionChangeFcn(hObject, eventdata, handles)
 if hObject==handles.radiobutton_rutear
```

```
     set(handles.tabla_machine,'enable','on','data',handles.matrizcopiado_dec);
 elseif hObject==handles.radiobutton_perforar
     set(handles.tabla_machine,'enable','on','data',handles.matriz_agu_dec);
 end

% --------------------------------------------------------------------
function op_linea_Callback(hObject, eventdata, handles)
% hObject    handle to op_linea (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1,'visible','off');
linea_inicial;

% --------------------------------------------------------------------
function op_manual_Callback(hObject, eventdata, handles)
% hObject    handle to op_manual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1,'visible','off');
control_manual;
```

Ventana de control manual

```
function varargout = control_manual(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @control_manual_OpeningFcn, ...
                   'gui_OutputFcn',  @control_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 control_manual is made visible.
function control_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   command line arguments to control_manual (see VARARGIN)

% Choose default command line output for control_manual
handles.output = hObject;
set(handles.figure1,'CloseRequestFcn',[]);
load parametros.mat;
set(handles.altura_fresado,'string',config_ejes.z.recorrido-altura_rut);
set(handles.altura_perforacion,'string',config_ejes.z.recorrido-altura_perf);
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes control_manual wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = control_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 pushbutton_EF_ON.
function pushbutton_EF_ON_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_EF_ON (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load matriz_ruteo.mat;
matriz_manual(1,4)=1;
save data\matriz_ruteo.mat matriz_manual -append;

% --- Executes on button press in pushbutton_EF_OFF.
function pushbutton_EF_OFF_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_EF_OFF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load matriz_ruteo.mat;
matriz_manual(1,4)=0;
save data\matriz_ruteo.mat matriz_manual -append;


% --- Executes on slider movement.
function sliderX_Callback(hObject, eventdata, handles)
% hObject    handle to sliderX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
load parametros.mat;
load matriz_ruteo.mat;
matriz_manual(1,2)=get(hObject,'value');
set(handles.text_posx_man,'string',matriz_manual(1,2));
save data\matriz_ruteo.mat matriz_manual -append;

% --- Executes during object creation, after setting all properties.
function sliderX_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function sliderY_Callback(hObject, eventdata, handles)
% hObject    handle to sliderY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
load parametros.mat;
load matriz_ruteo.mat;
matriz_manual(1,1)=get(hObject,'value');
set(handles.text_posy_man,'string',matriz_manual(1,1));
save data\matriz_ruteo.mat matriz_manual -append;

% --- Executes during object creation, after setting all properties.
```

```
function sliderY_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function sliderZ_Callback(hObject, eventdata, handles)
% hObject    handle to sliderZ (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
load parametros.mat;
load matriz_ruteo.mat;
matriz_manual(1,3)=get(hObject,'value');
set(handles.text_posz_man,'string',config_ejes.z.recorrido-matriz_manual(1,3));
save data\matriz_ruteo.mat matriz_manual -append;

% --- Executes during object creation, after setting all properties.
function sliderZ_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderZ (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes when selected object is changed in uipanel_manual.
function uipanel_manual_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in uipanel_manual
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none was selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
load manejadores.mat;
load parametros.mat;

if hObject==handles.radiobutton_on_manual
     set(handles.sliderX,'enable','on');
     set(handles.sliderY,'enable','on');
     set(handles.sliderZ,'enable','on');

     set(handles.pushbutton_EF_ON,'enable','on');
     set(handles.pushbutton_EF_OFF,'enable','on');

%      set(handles.rutear_machine,'enable','off');
%      set(handles.pausa_machine,'enable','off');
%      set(handles.perforar_machine,'enable','off');
%      set(handles.stop_machine,'enable','off');

     set(handles.sliderX,'max',config_ejes.x.recorrido);
     set(handles.sliderY,'max',config_ejes.y.recorrido);
     set(handles.sliderZ,'max',config_ejes.z.recorrido);
     set(handles.sliderX,'value',config_ejes.x.recorrido-0.5);
     set(handles.sliderY,'value',0.5);
     set(handles.sliderZ,'value',0.5);
     matriz_manual=[0.5,config_ejes.x.recorrido-0.5,0.5,0];
     save data\matriz_ruteo.mat matriz_manual -append;
     save data\leer_hobject.mat hObject;
     cabeza=4;
```

```
        script_usb_manual;
        if conectado==0
            set(handles.radiobutton_on_manual,'value',0);
            set(handles.radiobutton_off_manual,'value',1);
            set(handles.sliderX,'enable','off');
            set(handles.sliderY,'enable','off');
            set(handles.sliderZ,'enable','off');
            set(handles.pushbutton_EF_ON,'enable','off');
            set(handles.pushbutton_EF_OFF,'enable','off');
        end
    elseif hObject==handles.radiobutton_off_manual
        set(handles.sliderX,'enable','off');
        set(handles.sliderY,'enable','off');
        set(handles.sliderZ,'enable','off');
        set(handles.pushbutton_EF_ON,'enable','off');
        set(handles.pushbutton_EF_OFF,'enable','off');
        save data\leer_hobject.mat hObject;
    end

    % --- Executes on button press in pushbutton_aceptar.
    function pushbutton_aceptar_Callback(hObject, eventdata, handles)
    % hObject    handle to pushbutton_aceptar (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    load manejadores.mat;
    load parametros.mat
    altura_rut=config_ejes.z.recorrido-str2double(get(handles.altura_fresado,'string'));
    altura_perf=config_ejes.z.recorrido-str2double(get(handles.altura_perforacion,'string'));
    save data\parametros.mat altura_rut altura_perf -append;

    set(handles.figure1,'CloseRequestFcn','closereq');
    set(handlesmadre.figure1,'visible','on');
    close('control_manual');

    % --- Executes on button press in pushbutton_cancelar.
    function pushbutton_cancelar_Callback(hObject, eventdata, handles)
    % hObject    handle to pushbutton_cancelar (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    load manejadores.mat;
    set(handles.figure1,'CloseRequestFcn','closereq');
    set(handlesmadre.figure1,'visible','on');
    close('control_manual');

    function altura_fresado_Callback(hObject, eventdata, handles)
    % hObject    handle to altura_fresado (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of altura_fresado as text
    %        str2double(get(hObject,'String')) returns contents of altura_fresado as a double

    % --- Executes during object creation, after setting all properties.
    function altura_fresado_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to altura_fresado (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: edit 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

    function altura_perforacion_Callback(hObject, eventdata, handles)
    % hObject    handle to altura_perforacion (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of altura_perforacion as text
```

```
%        str2double(get(hObject,'String')) returns contents of altura_perforacion as a
double

% --- Executes during object creation, after setting all properties.
function altura_perforacion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to altura_perforacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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
```

Ventana de configuración de línea inicial

```
function varargout = linea_inicial(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @linea_inicial_OpeningFcn, ...
                   'gui_OutputFcn',  @linea_inicial_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 linea_inicial is made visible.
function linea_inicial_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 linea_inicial (see VARARGIN)

% Choose default command line output for linea_inicial
handles.output = hObject;
set(handles.figure1,'CloseRequestFcn',[]);
load detener.mat;
set(handles.edit1,'string',(ultimalinea));
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes linea_inicial wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = linea_inicial_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;

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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 pushbutton_aceptar.
function pushbutton_aceptar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_aceptar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load manejadores.mat
load detener.mat
ultimalinea=str2double(get(handles.edit1,'string'));
save data\detener.mat ultimalinea -append;
set(handles.figure1,'CloseRequestFcn','closereq');
set(handlesmadre.figure1,'visible','on');
close('linea_inicial');

% --- Executes on button press in pushbutton_cancelar.
function pushbutton_cancelar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_cancelar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load manejadores.mat
set(handles.figure1,'CloseRequestFcn','closereq');
set(handlesmadre.figure1,'visible','on');
close('linea_inicial');
```

Ventana de Configuracion de parámetros:

```
function varargout = parametros(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @parametros_OpeningFcn, ...
                   'gui_OutputFcn',  @parametros_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 parametros is made visible.
function parametros_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 parametros (see VARARGIN)

% Choose default command line output for parametros
handles.output = hObject;
set(handles.figure1,'CloseRequestFcn',[]);
if exist('parametros.mat')==2
    load parametros.mat;
    set(handles.lista_resolucion,'value',resolucion_value);
    set(handles.lista_diam_broca,'value',diam_broca_value);
    set(handles.lista_gap,'string',gap_string);
    set(handles.lista_gap,'value',gap_value);

    set(handles.edit_encoder_x,'string',config_ejes.x.encoder);
    set(handles.edit_pulsered_x,'string',config_ejes.x.pulsered);
    set(handles.edit_enc_supr_bits_x,'string',config_ejes.x.enc_supr_bits);
    set(handles.edit_numbits_x,'string',config_ejes.x.numbits);
    set(handles.edit_rel1_x,'string',config_ejes.x.rel(1));
    set(handles.edit_rel2_x,'string',config_ejes.x.rel(2));
    set(handles.popupmenu_supr_bits_x,'string',config_ejes.x.supr_bits.cadena);
    set(handles.popupmenu_supr_bits_x,'value',config_ejes.x.supr_bits.value);
    set(handles.edit_paso_husillo_x,'string',config_ejes.x.paso);
    set(handles.edit_recorrido_x,'string',config_ejes.x.recorrido);

    set(handles.edit_encoder_y,'string',config_ejes.y.encoder);
    set(handles.edit_pulsered_y,'string',config_ejes.y.pulsered);
    set(handles.edit_enc_supr_bits_y,'string',config_ejes.y.enc_supr_bits);
    set(handles.edit_numbits_y,'string',config_ejes.y.numbits);
    set(handles.edit_rel1_y,'string',config_ejes.y.rel(1));
    set(handles.edit_rel2_y,'string',config_ejes.y.rel(2));
    set(handles.popupmenu_supr_bits_y,'string',config_ejes.y.supr_bits.cadena);
    set(handles.popupmenu_supr_bits_y,'value',config_ejes.y.supr_bits.value);
    set(handles.edit_paso_husillo_y,'string',config_ejes.y.paso);
    set(handles.edit_recorrido_y,'string',config_ejes.y.recorrido);

    set(handles.edit_encoder_z,'string',config_ejes.z.encoder);
    set(handles.edit_pulsered_z,'string',config_ejes.z.pulsered);
    set(handles.edit_enc_supr_bits_z,'string',config_ejes.z.enc_supr_bits);
    set(handles.edit_numbits_z,'string',config_ejes.z.numbits);
    set(handles.edit_rel1_z,'string',config_ejes.z.rel(1));
    set(handles.edit_rel2_z,'string',config_ejes.z.rel(2));
    set(handles.popupmenu_supr_bits_z,'string',config_ejes.z.supr_bits.cadena);
    set(handles.popupmenu_supr_bits_z,'value',config_ejes.z.supr_bits.value);
    set(handles.edit_paso_husillo_z,'string',config_ejes.z.paso);
    set(handles.edit_recorrido_z,'string',config_ejes.z.recorrido);

else
    set(handles.lista_resolucion,'value',1);
    set(handles.lista_diam_broca,'value',1);
    set(handles.lista_gap,'string',{'0.1';'0.2';'0.3';'0.4';'0.5';'0.6';'0.7';'0.8';'0.9';'1.0'});
    set(handles.lista_gap,'value',1);

    set(handles.edit_encoder_y,'string','1');
%     set(handles.popupmenu_supr_bits_y,'string',{'1'});
%     set(handles.popupmenu_supr_bits_y,'value',1);
    set(handles.edit_paso_husillo_y,'string','1');
    set(handles.edit_recorrido_y,'string','1');

    set(handles.edit_encoder_x,'string','1');
%     set(handles.popupmenu_supr_bits_x,'string',{'1'});
%     set(handles.popupmenu_supr_bits_x,'value',1);
    set(handles.edit_paso_husillo_x,'string','1');
    set(handles.edit_recorrido_x,'string','1');

    set(handles.edit_encoder_x,'string','1');
%     set(handles.popupmenu_supr_bits_x,'string',{'1'});
%     set(handles.popupmenu_supr_bits_x,'value',1);
    set(handles.edit_paso_husillo_x,'string','1');
```

```
    set(handles.edit_recorrido_x,'string','1');
end

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes parametros wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = parametros_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;

function lista_resolucion_Callback(hObject, eventdata, handles)
% hObject    handle to lista_resolucion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function lista_resolucion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lista_resolucion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function lista_diam_broca_Callback(hObject, eventdata, handles)
% hObject    handle to lista_diam_broca (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
variable=get(handles.lista_diam_broca,'value');
switch (variable)
    case 1

set(handles.lista_gap,'string',{'0.1';'0.2';'0.3';'0.4';'0.5';'0.6';'0.7';'0.8';'0.9';'1.0'});
        if (get(handles.lista_gap,'value'))>10
            set(handles.lista_gap,'value',1.0);
        end
    case 2

set(handles.lista_gap,'string',{'0.2';'0.4';'0.6';'0.8';'1.0';'1.2';'1.4';'1.6';'1.8';'2.0'});
        if (get(handles.lista_gap,'value'))>10
            set(handles.lista_gap,'value',1.0);
        end
    case 3
        set(handles.lista_gap,'string',{'0.3';'0.6';'0.9';'1.2';'1.5';'1.8';'2.1'});
        if (get(handles.lista_gap,'value'))>7
            set(handles.lista_gap,'value',1.0);
        end
    case 4
        set(handles.lista_gap,'string',{'0.5';'1.0';'1.5';'2.0'});
        if (get(handles.lista_gap,'value'))>4
            set(handles.lista_gap,'value',1.0);
        end
    case 5
        set(handles.lista_gap,'string',{'1.0';'2.0'});
        if (get(handles.lista_gap,'value'))>2
            set(handles.lista_gap,'value',1.0);
        end
end

% --- Executes during object creation, after setting all properties.
function lista_diam_broca_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lista_diam_broca (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit 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


function lista_gap_Callback(hObject, eventdata, handles)
% hObject    handle to lista_gap (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function lista_gap_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lista_gap (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_recorrido_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_recorrido_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valid_edit_par;
% Hints: get(hObject,'String') returns contents of edit_recorrido_y as text
%        str2double(get(hObject,'String')) returns contents of edit_recorrido_y as a double

% --- Executes during object creation, after setting all properties.
function edit_recorrido_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_recorrido_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_paso_husillo_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_paso_husillo_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valid_edit_par;
% Hints: get(hObject,'String') returns contents of edit_paso_husillo_y as text
%        str2double(get(hObject,'String')) returns contents of edit_paso_husillo_y as a double

% --- Executes during object creation, after setting all properties.
function edit_paso_husillo_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_paso_husillo_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_encoder_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_encoder_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valid_edit_par;
% Hints: get(hObject,'String') returns contents of edit_encoder_y as text
%        str2double(get(hObject,'String')) returns contents of edit_encoder_y as a double

% --- Executes during object creation, after setting all properties.
function edit_encoder_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_encoder_y (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_encoder_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_encoder_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valid_edit_par;
% Hints: get(hObject,'String') returns contents of edit_encoder_x as text
%        str2double(get(hObject,'String')) returns contents of edit_encoder_x as a double

% --- Executes during object creation, after setting all properties.
function edit_encoder_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_encoder_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_paso_husillo_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_paso_husillo_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valid_edit_par;
% Hints: get(hObject,'String') returns contents of edit_paso_husillo_x as text
%        str2double(get(hObject,'String')) returns contents of edit_paso_husillo_x as a double

% --- Executes during object creation, after setting all properties.
function edit_paso_husillo_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_paso_husillo_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_recorrido_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_recorrido_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valid_edit_par;
% Hints: get(hObject,'String') returns contents of edit_recorrido_x as text
%        str2double(get(hObject,'String')) returns contents of edit_recorrido_x as a double

% --- Executes during object creation, after setting all properties.
function edit_recorrido_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_recorrido_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_recorrido_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_recorrido_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valid_edit_par;
% Hints: get(hObject,'String') returns contents of edit_recorrido_x as text
%        str2double(get(hObject,'String')) returns contents of edit_recorrido_x as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit_recorrido_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_recorrido_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_paso_husillo_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_paso_husillo_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valid_edit_par;
% Hints: get(hObject,'String') returns contents of edit_paso_husillo_x as text
%        str2double(get(hObject,'String')) returns contents of edit_paso_husillo_x as a double

% --- Executes during object creation, after setting all properties.
function edit_paso_husillo_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_paso_husillo_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_encoder_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_encoder_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valid_edit_par;
% Hints: get(hObject,'String') returns contents of edit_encoder_x as text
%        str2double(get(hObject,'String')) returns contents of edit_encoder_x as a double

% --- Executes during object creation, after setting all properties.
function edit_encoder_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_encoder_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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 pushbutton_aceptar.
function pushbutton_aceptar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_aceptar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
T=0.08;%100ms tiempo de muestreo
load manejadores.mat;
resolucion_value=(get(handles.lista_resolucion,'value'));
resolucion=10*resolucion_value;
gap_value=get(handles.lista_gap,'value');
gap_string=get(handles.lista_gap,'string');
gap=str2double(gap_string(gap_value));
diam_broca_value=get(handles.lista_diam_broca,'value');
switch(diam_broca_value)
    case 1
        diam_broca=0.1;
    case 2
        diam_broca=0.2;
    case 3
        diam_broca=0.3;
    case 4
        diam_broca=0.5;
    case 5
```

```
        diam_broca=1.0;
end

config_ejes.x.encoder=str2double(get(handles.edit_encoder_x,'string'));
config_ejes.x.pulsered=str2double(get(handles.edit_pulsered_x,'string'));
config_ejes.x.enc_supr_bits=str2double(get(handles.edit_enc_supr_bits_x,'string'));
config_ejes.x.numbits=str2double(get(handles.edit_numbits_x,'string'));
config_ejes.x.rel(1)=str2double(get(handles.edit_rel1_x,'string'));
config_ejes.x.rel(2)=str2double(get(handles.edit_rel2_x,'string'));
config_ejes.x.supr_bits.value=get(handles.popupmenu_supr_bits_x,'value');
config_ejes.x.supr_bits.cadena=get(handles.popupmenu_supr_bits_x,'string');
config_ejes.x.paso=str2double(get(handles.edit_paso_husillo_x,'string'));
config_ejes.x.recorrido=str2double(get(handles.edit_recorrido_x,'string'));

config_ejes.y.encoder=str2double(get(handles.edit_encoder_y,'string'));
config_ejes.y.pulsered=str2double(get(handles.edit_pulsered_y,'string'));
config_ejes.y.enc_supr_bits=str2double(get(handles.edit_enc_supr_bits_y,'string'));
config_ejes.y.numbits=str2double(get(handles.edit_numbits_y,'string'));
config_ejes.y.rel(1)=str2double(get(handles.edit_rel1_y,'string'));
config_ejes.y.rel(2)=str2double(get(handles.edit_rel2_y,'string'));
config_ejes.y.supr_bits.value=get(handles.popupmenu_supr_bits_y,'value');
config_ejes.y.supr_bits.cadena=get(handles.popupmenu_supr_bits_y,'string');
config_ejes.y.paso=str2double(get(handles.edit_paso_husillo_y,'string'));
config_ejes.y.recorrido=str2double(get(handles.edit_recorrido_y,'string'));

config_ejes.z.encoder=str2double(get(handles.edit_encoder_z,'string'));
config_ejes.z.pulsered=str2double(get(handles.edit_pulsered_z,'string'));
config_ejes.z.enc_supr_bits=str2double(get(handles.edit_enc_supr_bits_z,'string'));
config_ejes.z.numbits=str2double(get(handles.edit_numbits_z,'string'));
config_ejes.z.rel(1)=str2double(get(handles.edit_rel1_z,'string'));
config_ejes.z.rel(2)=str2double(get(handles.edit_rel2_z,'string'));
config_ejes.z.supr_bits.value=get(handles.popupmenu_supr_bits_z,'value');
config_ejes.z.supr_bits.cadena=get(handles.popupmenu_supr_bits_z,'string');
config_ejes.z.paso=str2double(get(handles.edit_paso_husillo_z,'string'));
config_ejes.z.recorrido=str2double(get(handles.edit_recorrido_z,'string'));

save data/parametros.mat config_ejes resolucion resolucion_value gap gap_value gap_string
diam_broca diam_broca_value '-append';
set(handles.figure1,'CloseRequestFcn','closereq');
set(handlesmadre.figure1,'visible','on');
close('parametros');

% --- Executes on button press in pushbutton_cancelar.
function pushbutton_cancelar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_cancelar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load manejadores.mat;
set(handles.figure1,'CloseRequestFcn','closereq');
set(handlesmadre.figure1,'visible','on');
close('parametros');

function edit_rel1_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_rel1_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valor1=str2double(get(hObject,'string'));
valor2=str2double(get(handles.edit_rel2_y,'string'));
valor3=str2double(get(handles.edit_encoder_y,'string'));
script_relacionmotor;
if setear==1;
   set(handles.edit_pulsered_y,'string',num2str(pulsosreductor));
   set(handles.edit_numbits_y,'string',num2str(numerobits));
   set(handles.popupmenu_supr_bits_y,'string',cadena_supresion);
end

% --- Executes during object creation, after setting all properties.
function edit_rel1_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_rel1_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_rel2_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_rel2_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valor2=str2double(get(hObject,'string'));
valor1=str2double(get(handles.edit_rel1_y,'string'));
valor3=str2double(get(handles.edit_encoder_y,'string'));
script_relacionmotor;
if setear==1;
   set(handles.edit_pulsered_y,'string',num2str(pulsosreductor));
   set(handles.edit_numbits_y,'string',num2str(numerobits));
   set(handles.popupmenu_supr_bits_y,'string',cadena_supresion);
end

% --- Executes during object creation, after setting all properties.
function edit_rel2_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_rel2_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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 selection change in popupmenu_supr_bits_x.
function popupmenu_supr_bits_y_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_supr_bits_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
pulsosreductor=str2double(get(handles.edit_pulsered_y,'string'));
cadenasupresion=str2double(get(handles.popupmenu_supr_bits_y,'string'));
valorsupresion=get(handles.popupmenu_supr_bits_y,'value');
supresion=cadenasupresion(valorsupresion);
pulsosconsupr=pulsosreductor/supresion;
set(handles.edit_enc_supr_bits_y,'string',num2str(pulsosconsupr));
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu_supr_bits_x contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from popupmenu_supr_bits_x

% --- Executes during object creation, after setting all properties.
function popupmenu_supr_bits_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu_supr_bits_x (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

function edit_enc_supr_bits_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_enc_supr_bits_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_enc_supr_bits_x as text
%        str2double(get(hObject,'String')) returns contents of edit_enc_supr_bits_x as a double

% --- Executes during object creation, after setting all properties.
function edit_enc_supr_bits_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_enc_supr_bits_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_rel1_x_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit_rel1_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valor1=str2double(get(hObject,'string'));
valor2=str2double(get(handles.edit_rel2_x,'string'));
valor3=str2double(get(handles.edit_encoder_x,'string'));
script_relacionmotor;
if setear==1;
   set(handles.edit_pulsered_x,'string',num2str(pulsosreductor));
   set(handles.edit_numbits_x,'string',num2str(numerobits));
   set(handles.popupmenu_supr_bits_x,'string',cadena_supresion);
end

% --- Executes during object creation, after setting all properties.
function edit_rel1_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_rel1_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_rel2_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_rel2_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valor2=str2double(get(hObject,'string'));
valor1=str2double(get(handles.edit_rel1_x,'string'));
valor3=str2double(get(handles.edit_encoder_x,'string'));
script_relacionmotor;
if setear==1;
   set(handles.edit_pulsered_x,'string',num2str(pulsosreductor));
   set(handles.edit_numbits_x,'string',num2str(numerobits));
   set(handles.popupmenu_supr_bits_x,'string',cadena_supresion);
end
% Hints: get(hObject,'String') returns contents of edit_rel2_y as text
%        str2double(get(hObject,'String')) returns contents of edit_rel2_y as a double

% --- Executes during object creation, after setting all properties.
function edit_rel2_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_rel2_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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 selection change in popupmenu_supr_bits_y.
function popupmenu_supr_bits_x_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_supr_bits_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
pulsosreductor=str2double(get(handles.edit_pulsered_x,'string'));
cadenasupresion=(get(handles.popupmenu_supr_bits_x,'string'));
valorsupresion=get(handles.popupmenu_supr_bits_x,'value');
supresion=str2double(cadenasupresion(valorsupresion));
pulsosconsupr=pulsosreductor/supresion;
set(handles.edit_enc_supr_bits_x,'string',num2str(pulsosconsupr));
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu_supr_bits_y contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from popupmenu_supr_bits_y

% --- Executes during object creation, after setting all properties.
function popupmenu_supr_bits_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu_supr_bits_y (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

function edit_enc_supr_bits_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_enc_supr_bits_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit_enc_supr_bits_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_enc_supr_bits_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_rel1_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_rel1_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valor1=str2double(get(hObject,'string'));
valor2=str2double(get(handles.edit_rel2_z,'string'));
valor3=str2double(get(handles.edit_encoder_z,'string'));
script_relacionmotor;
if setear==1;
   set(handles.edit_pulsered_z,'string',num2str(pulsosreductor));
   set(handles.edit_numbits_z,'string',num2str(numerobits));
   set(handles.popupmenu_supr_bits_z,'string',cadena_supresion);
end

% --- Executes during object creation, after setting all properties.
function edit_rel1_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_rel1_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_rel2_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_rel2_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valor2=str2double(get(hObject,'string'));
valor1=str2double(get(handles.edit_rel1_z,'string'));
valor3=str2double(get(handles.edit_encoder_z,'string'));
script_relacionmotor;
if setear==1;
   set(handles.edit_pulsered_z,'string',num2str(pulsosreductor));
   set(handles.edit_numbits_z,'string',num2str(numerobits));
   set(handles.popupmenu_supr_bits_z,'string',cadena_supresion);
end

% --- Executes during object creation, after setting all properties.
function edit_rel2_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_rel2_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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 selection change in popupmenu_supr_bits_x.
function popupmenu_supr_bits_z_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_supr_bits_x (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
pulsosreductor=str2double(get(handles.edit_pulsered_z,'string'));
cadenasupresion=str2double(get(handles.popupmenu_supr_bits_z,'string'));
valorsupresion=get(handles.popupmenu_supr_bits_z,'value');
supresion=cadenasupresion(valorsupresion);
pulsosconsupr=pulsosreductor/supresion;
set(handles.edit_enc_supr_bits_z,'string',num2str(pulsosconsupr));
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu_supr_bits_x contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from popupmenu_supr_bits_x

% --- Executes during object creation, after setting all properties.
function popupmenu_supr_bits_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu_supr_bits_x (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

function edit_enc_supr_bits_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_enc_supr_bits_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_enc_supr_bits_x as text
%        str2double(get(hObject,'String')) returns contents of edit_enc_supr_bits_x as a double

% --- Executes during object creation, after setting all properties.
function edit_enc_supr_bits_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_enc_supr_bits_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_numbits_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_numbits_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit_numbits_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_numbits_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_numbits_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_numbits_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit_numbits_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_numbits_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_numbits_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_numbits_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit_numbits_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_numbits_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_pulsered_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_pulsered_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_pulsered_x as text
%        str2double(get(hObject,'String')) returns contents of edit_pulsered_x as a double

% --- Executes during object creation, after setting all properties.
function edit_pulsered_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_pulsered_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_pulsered_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_pulsered_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit_pulsered_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_pulsered_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function edit_pulsered_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_pulsered_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit_pulsered_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_pulsered_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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
```

Ventana de contraseña

```
function varargout = password(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @password_OpeningFcn, ...
                   'gui_OutputFcn',  @password_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 password is made visible.
function password_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 password (see VARARGIN)

% Choose default command line output for password
handles.output = hObject;
set(handles.figure1,'CloseRequestFcn',[]);
handles.UD=get(handles.pass,'userdata');
handles.asterisc='';
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes password wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = password_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;

function pass_Callback(hObject, eventdata, handles)
% hObject    handle to pass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function pass_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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 pushbutton_aceptar.
function pushbutton_aceptar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_aceptar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if exist('AMPW2.mat')==2
    load manejadores.mat;
    load AMPW2.mat;
    PWA=get(handles.pass,'userdata');
    if strcmp(PWA,AMPW)==1
        set(handlesmadre.op_parametros,'enable','on');
        set(handles.figure1,'CloseRequestFcn','closereq');
        set(handlesmadre.figure1,'visible','on');
        close('password');
    else
        warndlg('Contraseña incorrecta.');
    end

else
    warndlg('Falta el fichero AMPW2.mat.');
end

% --- Executes on button press in pushbutton_cancelar.
function pushbutton_cancelar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_cancelar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load manejadores.mat;
set(handles.figure1,'CloseRequestFcn','closereq');
set(handlesmadre.figure1,'visible','on');
close('password');

% --- Executes on key press with focus on pass and none of its controls.
function pass_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to pass (see GCBO)
% eventdata  structure with the following fields (see UICONTROL)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)

key = get(handles.figure1,'currentkey');

switch key

    case 'backspace'
        lastletter=size(handles.UD);
        if lastletter>0
            handles.UD(lastletter(2))='';
            handles.asterisc(lastletter(2))='';
            set(handles.pass,'userdata',handles.UD);
            set(handles.pass,'string',handles.asterisc);
            guidata(hObject, handles);
        end
    case 'return'
        if exist('AMPW2.mat')==2
            load manejadores.mat;
            load AMPW2.mat;
            PWA=get(handles.pass,'userdata');
            if strcmp(PWA,AMPW)==1
                %set(handlesmadre.op_config_perfiles,'enable','on');
                set(handlesmadre.op_parametros,'enable','on');
                load manejadores.mat;
                set(handles.figure1,'CloseRequestFcn','closereq');
                set(handlesmadre.figure1,'visible','on');
                close('password');
            else
                warndlg('Contraseña incorrecta.');
            end
```

```
        else
            warndlg('Falta el fichero AMPW2.mat.');
        end
    case 'control'
    case 'shift'
    case 'capslock'
    case 'escape'
    case 'downarrow'
    case 'uparrow'
    case 'leftarrow'
    case 'rightarrow'
    case 'alt'
    case 'windows'
    case 'delete'

    otherwise
        caracter=get(handles.figure1,'currentcharacter');
        handles.UD=cat(2,handles.UD,caracter);
        handles.asterisc=cat(2,handles.asterisc,'*');
        set(handles.pass,'userdata',handles.UD);
        set(handles.pass,'string',handles.asterisc);
        guidata(hObject, handles);
end
```

Ventana de configuración de copias:

```
function varargout = Ruteocopias(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Ruteocopias_OpeningFcn, ...
                   'gui_OutputFcn',  @Ruteocopias_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 Ruteocopias is made visible.
function Ruteocopias_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Ruteocopias
handles.output = hObject;
set(handles.figure1,'CloseRequestFcn',[]);
load manejadores.mat;
load parametros.mat;
if exist('datos_copias.mat')==2
    load datos_copias.mat;
    set(handles.alto_placa,'enable','on','string',num2str(altoplaca));
    set(handles.ancho_placa,'enable','on','string',num2str(anchoplaca));
    set(handles.num_copias,'enable','on','string',num2str(numerocopias));
    if numerocopias==1
        axes(handles.grafica_preview);
        img_pcb_copy=imread('pcb_copy.jpg');
        imshow(handlesmadre.rutmat)
    else
        axes(handles.grafica_preview);
```

```
        img_pcb_copy=imread('pcb_copy.jpg');
        imshow(img_pcb_copy);
    end
else

set(handles.alto_placa,'enable','off','string',num2str(ceil(handlesmadre.tamrutmat(1))));

set(handles.ancho_placa,'enable','off','string',num2str(ceil(handlesmadre.tamrutmat(2))));
end
script_preview;
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = Ruteocopias_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

function num_copias_Callback(hObject, eventdata, handles)
% hObject    handle to num_copias (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if str2double(get(hObject,'string'))>0 && isnan(str2double(get(hObject,'string')))~=1
    set(handles.alto_placa,'enable','on');%cuando edita num_copias se habilita alto_placa
para la edicion
    set(handles.ancho_placa,'enable','on');%cuando edita num_copias se habilita alto_placa
para la edicion
else
    h_warning=warndlg('El numero de copias debe ser mayor a 0. Solo debe escribir
numeros.');
    set(hObject,'string','1');
end
% Hints: get(hObject,'String') returns contents of num_copias as text
%        str2double(get(hObject,'String')) returns contents of num_copias as a double

% --- Executes during object creation, after setting all properties.
function num_copias_CreateFcn(hObject, eventdata, handles)
% hObject    handle to num_copias (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function alto_placa_Callback(hObject, eventdata, handles)
% hObject    handle to alto_placa (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function alto_placa_CreateFcn(hObject, eventdata, handles)
% hObject    handle to alto_placa (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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

function ancho_placa_Callback(hObject, eventdata, handles)
% hObject    handle to ancho_placa (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ancho_placa as text
%        str2double(get(hObject,'String')) returns contents of ancho_placa as a double
```

```
% --- Executes during object creation, after setting all properties.
function ancho_placa_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ancho_placa (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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 pushbutton_aceptar.
function pushbutton_aceptar_Callback(hObject, eventdata, handles)
load manejadores.mat;
load parametros.mat;
if exist('datos_copias.mat')==2
    load datos_copias.mat;
end

if(str2double(get(handles.num_copias,'string')))>1
    script_copias;
    if guardar==1

        matrizcopiado_dec(:,1:2)=matrizcopiado(:,1:2)/resolucion;
        matrizcopiado_dec(:,3:4)=matrizcopiado(:,3:4);
        set(handlesmadre.tabla_machine,'data',matrizcopiado_dec);%modifica el valor mostrado
en tabla_machine en la GUI principal
        save data\matriz_ruteo.mat matrizcopiado;
        imwrite(img_pcb_copy,'temp\pcb_copy.jpg');
%         axes(handlesmadre.grafica_copiaspcb);%indica que se graficara en grafica_copiaspcb
%         imshow(img_pcb_copy);
        set(handlesmadre.figure1,'visible','on');
        mensaje=msgbox('Copia realizada con exito.');%muestra un msj de exito
        save data\datos_copias.mat altoplaca anchoplaca numerocopias;
        set(handles.figure1,'CloseRequestFcn','closereq');
        close ('Ruteocopias')
        %close(mensaje);
    end
else
    altoplaca=ceil(handlesmadre.tamrutmat(1));
    anchoplaca=ceil(handlesmadre.tamrutmat(2));
    numerocopias=1;

    set(handles.alto_placa,'enable','off','string',num2str(altoplaca));%cuando edita
num_copias se habilita alto_placa para la edicion
    set(handles.ancho_placa,'enable','off','string',num2str(anchoplaca));%cuando edita
num_copias se habilita alto_placa para la edicion
    set(handles.num_copias,'enable','off','string',num2str(numerocopias));
    matrizcopiado_dec(:,1:2)=handlesmadre.matmachine(:,1:2)/resolucion;
    matrizcopiado_dec(:,3:4)=handlesmadre.matmachine(:,3:4);
    set(handlesmadre.tabla_machine,'data',matrizcopiado_dec);
    matrizcopiado=handlesmadre.matmachine;
    save data\matriz_ruteo.mat matrizcopiado;
%     axes(handlesmadre.grafica_copiaspcb);
%     imshow(handlesmadre.rutmat);
    set(handlesmadre.figure1,'visible','on');
    save data\datos_copias.mat altoplaca anchoplaca numerocopias;
    set(handles.figure1,'CloseRequestFcn','closereq');
    close ('Ruteocopias');
end

% --- Executes on button press in pushbutton_preview.
function pushbutton_preview_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_preview (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load parametros.mat;
script_preview;
```

```
% --- Executes on button press in pushbutton_cancelar.
function pushbutton_cancelar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_cancelar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load manejadores.mat;
set(handles.figure1,'CloseRequestFcn','closereq');
set(handlesmadre.figure1,'visible','on');
close('Ruteocopias');
```

FUNCIONES

Circulomatri.m:

```
function
[rutmat,auxpad]=circulomatri(rutmat,apertura,num_list_apertura,d1,d2,aux_num_list_apertura,e
jexx,ejeyy,res,tempad,limite_pos1,limite_pos2)

diam=round(d1/2*res);
forma=apertura.forma(num_list_apertura);
diam2=round(d2*res);

diam1=diam*2;
if forma=='C'
    diam2=diam1;
    diams2=diam;
end
if forma=='O'
    diams2=round(diam2/2);
end
if forma=='R'
    diams2=round(diam2/2);
end

if num_list_apertura==aux_num_list_apertura
    auxpad=tempad;
else
    [auxpad]=PADS(forma,diam1,diam2,res);
end
 [ypos,xpos]=find(auxpad==0);

yypos=ejeyy-diams2+ypos;
xxpos=ejexx-diam+xpos;

for i=1:length(ypos)
    if xxpos(i)<=limite_pos2 && xxpos(i)>=1 && yypos(i)<=limite_pos1 && yypos(i)>=1
            rutmat(yypos(i),xxpos(i))=0;
    end
end
```

Contornospcb.m:

```
%Esta Funcion lee la imagen en rutmat, identifica los objetos en la imagen
%los separa y saca los cornotos de la misma. Esta funcion devuelve la matriz donde estan
%las coordenadas X Y y la ultima columna indica el numero del objeto o
%pista en este caso
```

```
function [bordesrutmat]=contornospcb(rutmat)

B=bwboundaries(~rutmat);
save data\B.mat B;
j=0;
tolerancia=1;
tolerancia2=80;
for k=1:length(B)
    temp=B{k};
    x1=temp(1,1);
    x2=temp(2,1);
    y1=temp(1,2);
    y2=temp(2,2);
    j=j+1;
    bordesrutmat(j,1)=temp(1,1);
    bordesrutmat(j,2)=temp(1,2);
    bordesrutmat(j,3)=k;
    j=j+1;
    for i=1:length(temp)-4%   algoritmo Hector
        difx1=temp(i+2,1)-temp(i,1);
        difx2=temp(i+4,1)-temp(i+2,1);
        dify1=temp(i+2,2)-temp(i,2);
        dify2=temp(i+4,2)-temp(i+2,2);
        if difx1~=difx2 || dify1~=dify2
            dis=((x2-temp(i+1,1))^2+ (y2-temp(i+1,2))^2)^0.5;
            if dis>tolerancia
                x2=temp(i+1,1);
                y2=temp(i+1,2);
%                     plot(y2,x2,'*')
                    bordesrutmat(j,1)=temp(i+1,1);
                    bordesrutmat(j,2)=temp(i+1,2);
                    bordesrutmat(j,3)=k;
                    j=j+1;
            end
% %          end
        elseif difx1==dify1 && difx2==dify2 || difx1==-dify1 && difx2==-dify2
            dis=((x2-temp(i+1,1))^2+ (y2-temp(i+1,2))^2)^0.5;
            if dis>tolerancia2
                x2=temp(i+1,1);
                y2=temp(i+1,2);
                    bordesrutmat(j,1)=temp(i+1,1);
                    bordesrutmat(j,2)=temp(i+1,2);
                    bordesrutmat(j,3)=k;
                    j=j+1;
            end
         end
    end
    bordesrutmat(j,1)=temp(1,1);
    bordesrutmat(j,2)=temp(1,2);
    bordesrutmat(j,3)=k;
end

end
```

Conver.m:

```
%Esta funcion recibe toda la informacion del Gerber y saca
%el vector "ejex" el "ejey", la accion del efector final "ruteo" (D01, D02 y D03)
%saca lo que se llama la posicion de rueda en la variable "inst"
%tambien obtiene "config" que es donde se guarda la informacion de cabecera
%y manda un codigo de "error" en caso de que exista en el codigo
%def_apertura: define todas las definiciones de apertura ADD10, ADD11..
% datospistas=[ejex,ejey,ruteo,diam1,diam2,posiciona_lista_aperutra]
% Nota: ruteo indica si es pista, punto o agujero
function [datospistas,apertura]=conver(dato,fuente)
j=1;
```

```
[apertura,formato,poscord] = encabezado_gerber(dato);
formato_cab(1,:)=formato(1,:);
apertura_cab(1)=apertura;
save('data\cabecera.mat','formato','apertura_cab');
%%[datospistas,dato123]=coordenadasgerber(dato,apertura,formato);
%%
vectorx=0;
vectory=0;
j=1;
tempj=1;
tam=size(dato);
%tamapertura=size(apertura.forma);
for i=poscord:tam(1)
    pos=cell2mat(dato(i,1));
    size_pos=size(pos);
    %cordenadas X Y
    temnull=strmatch(pos(size_pos(2)-3:size_pos(2)-1),apertura.cod); %busca los D.. en
cod_abertura
    tem=isempty(temnull); %si no lo encuentra tem=1
    if tem==1
        if (pos(1)=='X')||(pos(1)=='Y')||(strcmp(pos(1:3),'D02'))%%%% ojo nueva edicion del
programa
            dato123(j,1)=dato(i,1);
            [vectorx,vectory,rut]=validcor(pos,vectorx,vectory);
            datospistas(j,1)=str2double(vectorx)/(10^mod(formato(1,3),10));
            datospistas(j,2)=str2double(vectory)/(10^mod(formato(1,4),10));
            datospistas(j,3)=str2double(rut);
            %se divide entre 1000 por ser de pcbwizard, se debe
            %estandarizar para los demas softwares
% % % % % % % % %             datospistas(j,4)=apertura.dim1(1,tempj)/1000;
% % % % % % % % %             datospistas(j,5)=apertura.dim2(1,tempj)/1000;
            datospistas(j,4)=apertura.dim1(1,tempj);
            datospistas(j,5)=apertura.dim2(1,tempj);
            datospistas(j,6)=tempj;%Se guarda la posicion de la lista de apertura segun el
codigo de apertura
            j=j+1;
        end
    else
        tempj=temnull; %Si no es nulo guarda la posicion en temj
    end
end
%%
save('data\tempp.mat','dato123','datospistas');
%%
%pasar coordenadas de las unidades del gerber a milimetros o pulgadas segun
%el sistema de coordenadas
if formato(1,5)==1 %si el sistema de coordenadas es MOIN
    datospistas(:,1)=datospistas(:,1)*25.4;
    datospistas(:,2)=datospistas(:,2)*25.4;
    datospistas(:,4)=datospistas(:,4)*25.4;
    datospistas(:,5)=datospistas(:,5)*25.4;
     apertura.dim1=apertura.dim1*25.4;%OJO esto tambien se esta guardando en datospista y
esta siendo usada
     apertura.dim2=apertura.dim2*25.4;%
end

end
```

CoordenadaCNC.m:

```
function [matmachine]=coordenadaCNC(machineaux1)
load parametros.mat;
subida_ref=50;
```

```
bajada_rut=altura_rut;%99.95;%99,3 para ruteo con fresa en V rota 90.6 para broca normal
partida
subida_rut=80;

%las posiciones de referencia del eje Z se colocan en valores altos porque
%la referencia 0 del eje se encuentra en la parte superior por cuestiones
%de lectura del encoder.

size_ma=length(machineaux1);%tamaño machineaux
num_pist=max(machineaux1(:,3));
longitud_matf=(2*num_pist+size_ma)+2;
matrizfinal=zeros(longitud_matf,4);
%tamaño regido por 2 posiciones de referencia al inicio y al final
%2 posiciones faltantes por pista (llegada arriba, bajada para rutear,
%subida al finalizar), la posicion de ruteo ya esta en machineaux1

%inicializo valores de la matriz
matrizfinal(1,3)=subida_ref;%posicion aribba de referencia
matrizfinal(1:(2*num_pist+size_ma),4)=1;%efector final encendido para ruteo
matrizfinal((2*num_pist+size_ma+3):longitud_matf-1,4)=1;%efector final encendido para
perforacion
matrizfinal((2*num_pist+size_ma+2),3)=subida_ref;%posicion aribba de referencia, para cambio
de ruteo a perforacion (usar esta posicion para colocar una pausa en la maquina y cambiar
broca)
matrizfinal(3:(2*num_pist+size_ma),3)=bajada_rut;%posicion abajo de ruteo
matrizfinal(2,1)=machineaux1(1,1);
matrizfinal(2,2)=machineaux1(1,2);
matrizfinal(2,3)=subida_rut;

im=3;
for i=1:size_ma
    matrizfinal(im,1:2)=machineaux1(i,1:2);
    matrizfinal(im,3)=bajada_rut;
    indice=machineaux1(i,3);
    if i~= size_ma
        if indice~=machineaux1(i+1,3)
            matrizfinal(im+1,1:2)=machineaux1(i,1:2);
            matrizfinal(im+1,3)=subida_rut;%ir a posicion aribba de referencia
            matrizfinal(im+2,1:2)=machineaux1(i+1,1:2);
            matrizfinal(im+2,3)=subida_rut;
            im=im+2;
        end
    else
        matrizfinal(im+1,1:2)=machineaux1(i,1:2);
        matrizfinal(im+1,3)=subida_rut;
        im=im+1;
    end
    im=im+1;
end
%%
 figure(36)
tem=max(machineaux1(:,3));
 hold on
j=2;
for i=1:tem
    while machineaux1(j,3)==i
        plot(machineaux1(j-1:j,2),machineaux1(j-1:j,1),'+')
        plot(machineaux1(j-1:j,2),machineaux1(j-1:j,1))
        if j<length(machineaux1(:,3))
            j=j+1;
        else
            i=i+1;
        end
    end
    j=j+1;
end
matmachine=matrizfinal;
end
```

CoordenadaCNC_agujeros.m:

```
function [matmachine_out]=coordenadaCNC_agujeros(ejex,ejey,ruteo1)
    %funcion coordenadaCNC_agujeros modificado 2/8/2014:

    %en esta modificacion la matriz de salida solo contiene las posiciones
    %de los agujeros y omite por completo las posiciones del ruteo, para
    %separar los datos y poder realizar una pausa antes de perforar.

    %se aumenta el tamaño de filas de matmachine_in en 2 posiciones
    %faltantes por agujero (llegada arriba, bajada para perforar, subida al
    %finalizar), la posicion de perforacion se incluye en ruteo1
    load parametros.mat;

    bajada_perf=altura_perf;%posicion predeterminada de bajada del eje Z
    subida_perf=60;%posicion predeterminada de subida del eje Z

    pos_agu=find(ruteo1==3);%posiciones de los agujeros
    num_agu=length(pos_agu);%numero de agujeros
    pos_ex_agu=(3*num_agu);%cantidad de movimientos para hacer los agujeros

    h=1;
    %inicializo en cero las nuevas filas de matmachine que van desde una
    %posicion despues de la ultima que tenia anteriormente mas la cantidad
    %de movimientos necesarios para hacer todos los agujeros
    matmachine_out(1:pos_ex_agu+1,1:4)=0;

    for j=1:3:pos_ex_agu
        %se mantiene la posicion en X y Y mientras el ese Z sube y baja, el
        %efector final se mantiene encendido.

        %posicion de llegada con eje Z arriba
        matmachine_out(j,2)=ejex(pos_agu(h));
        matmachine_out(j,1)=ejey(pos_agu(h));
        matmachine_out(j,3)=subida_perf;
        matmachine_out(j,4)=1;
        %posicion estatica con bajada de eje Z
        matmachine_out(j+1,2)=ejex(pos_agu(h));
        matmachine_out(j+1,1)=ejey(pos_agu(h));
        matmachine_out(j+1,3)=bajada_perf;
        matmachine_out(j+1,4)=1;
        %posicion estatica con subida de eje Z
        matmachine_out(j+2,2)=ejex(pos_agu(h));
        matmachine_out(j+2,1)=ejey(pos_agu(h));
        matmachine_out(j+2,3)=subida_perf;
        matmachine_out(j+2,4)=1;

        %plot(ejex(pos_agu(h)),ejey(pos_agu(h)),'r*')
        h=h+1;
    end

end
```

Encabezado_gerber.m:

```
%Funcion load_data_gerber
%{
Version 3.0
```

```
Creado antes de 07/10/2010 - 7:25pm
Actualizado 15/05/2014 - 5:00pm
Desarrollado por:   William Gonzalez Coquel
                    Hector Fabio Reyes
%La funcion encabezado_gerber, se encarga de leer el encabezado del archivo
%Gerber y la almacena en las variables formato, unidades, y en la
estructura de apertura
%}
function [apertura,formato,poscord] = encabezado_gerber(dato)
%% Se identifica la cabecera del Gerber
%Declaracion de variables auxiliares
jd=1;
fin=0;
i=1;
%Ciclo que se repite hasta que se declara el encabezado
while(fin<1)
    pos=cell2mat(dato(i,1));
    size_pos=size(pos);
    %Se lee la cabecera y se guarda en la variable formato
    if size_pos(2)>2
        if(pos(1:3)=='%FS')
            cabecera=pos; %dato(i,1);
            if cabecera(4)=='L'
                formato(1,1)=1; %Ceros a la izquierda omitidos
            elseif cabecera(4)=='T'
                formato(1,1)=2; %Ceros finales omitidos
            elseif cabecera(4)=='D'
                formato(1,1)=3; %Sin ceros omitidos
            else
                formato(1,1)=4; %Modo de dato no valido, muestra error
            end
            if cabecera(5)=='A'
                formato(1,2)=1; %Modo de coordenadas absoluto
            elseif cabecera(5)=='I'
                formato(1,2)=2; %Modo de coordenadas incrementales
            else
                formato(1,2)=3; %Modo de coordenadas no valido, muestra error
            end
            acab=strfind(cabecera,'X'); %partiendo de que el primer numero siempre esta
completo
            bcab=strfind(cabecera,'Y');
            ccab=strfind(cabecera,'*');
            formato(1,3)=str2double(cabecera(acab+1:bcab-1));
            formato(1,4)=str2double(cabecera(bcab+1:ccab-1));
        %Se lee las unidades de los datos y se alamcena en la variable unidades
        elseif (pos(1:3)=='%MO')
            %Si los datos estan en pulgadas
            if(pos(1:5)=='%MOIN')
                formato(1,5)=1;
            %Si los datos estan en milimetros
            elseif(pos(1:5)=='%MOMM')
                formato(1,5)=2;
            %Si no se es clara las unidades
            else
                formato(1,5)=3;
            end

        elseif (pos(1:4)=='%ADD')
            def_apertura(jd,:)=dato(i,1);
            jd=jd+1;
            %Primero se identifican todas las aperturas y luego se procesan en
            %el bloque de codigo siguiente y se almacena en la estructura
            %apertura
        elseif (pos(1)=='X')||(pos(1)=='Y')
            fin=2;
            poscord=i-1;% esta es la linea del fichero gerber donde comienzan las posiciones
de ruteo
        end
    end
    i=i+1;
end
```

```
%% Se Interpreta la apartura %ADD
%Este segmento de codigo se encarga de desglosar los ADD... donde se define
%los diametros de los circulos o lineas entre otros. La forma, diametro,
%codigo, se almacena en la estructura apertura.
tam=size(def_apertura);
for i=1:tam(1)
    pos=cell2mat(def_apertura(i,1));
    apertura.cod(i,1:3)=pos(4:6);
    apertura.forma(i,1)=pos(7);
    if apertura.forma(i,1)=='C'
        temp=strfind(pos,'*');
        tempcoma=pos(9:temp(1)-1);
        poscoma=strfind(tempcoma,',');
        if isempty(poscoma)
            apertura.dim1(i)=str2double(tempcoma);
            apertura.dim2(i)=0;
        else
            tempcoma(poscoma)='.';
            apertura.dim1(i)=str2double(tempcoma);
            apertura.dim2(i)=0;
        end
    elseif apertura.forma(i,1)=='O'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%modificacion por variacion en eagle%%%%%%%%%%
        if strfind(pos(1:9),',')==8
            temp=strfind(pos,'X');
            tempcoma=pos(9:temp(1)-1);
            poscoma=strfind(tempcoma,',');

            if isempty(poscoma)
                apertura.dim1(i)=str2double(tempcoma);
            else
                tempcoma(poscoma)='.';
                apertura.dim1(i)=str2double(tempcoma);
            end
            temp1=strfind(pos,'*');
            tempcoma=pos(temp(1)+1:temp1(1)-1);
            poscoma=strfind(tempcoma,',');
            if isempty(poscoma)
                apertura.dim2(i)=str2double(tempcoma);
            else
                tempcoma(poscoma)='.';
                apertura.dim2(i)=str2double(tempcoma);
            end
        elseif pos(7:8)=='OC'% variacion de eagle (octagono OC8) sustituimos por un circulo
            apertura.forma(i,1)='C';
            temp=strfind(pos,'*');
            tempcoma=pos(11:temp(1)-1);%notar que la variacion de eagle (OC8) modifica la 
posicion donde comienza la dimension
            poscoma=strfind(tempcoma,',');
            if isempty(poscoma)
                apertura.dim1(i)=str2double(tempcoma);
                apertura.dim2(i)=0;
            else
                tempcoma(poscoma)='.';
                apertura.dim1(i)=str2double(tempcoma);
                apertura.dim2(i)=0;
            end
        end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    elseif apertura.forma(i,1)=='D'
        load readme.mat;
        long_readme=size(readme);

        for busqueda=1:long_readme(1)
            readme_mat=cell2mat(readme(busqueda,:));
            long_readme_mat=size(readme_mat);
            if long_readme_mat(2)>0
                tempforma=strfind(readme_mat(1,2:long_readme_mat(2)),apertura.forma(i))
```

```
                if isempty(tempforma)==0
                    if strcmp(readme_mat(1:tempforma(1)-1),apertura.cod(i,:))==1
                        pos_units=strfind(readme_mat(1,:),'th');
                        if isempty(pos_units)==0
                            units=1/1000;%thou a inch para que todo quede en las mismas
unidades
                        elseif isempty(pos_units)==1
                            pos_units=strfind(readme_mat(1,:),'mm');
                            if isempty(pos_units)==0
                                units=1/25.4;%milimetros a inch para que todo quede en las
mismas unidades
                            elseif isempty(pos_units)==1
                                'ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR
ERROR'
                            end
                        end

                        pos_W=strfind(readme_mat(1,1:pos_units(1)),'W');
                        pos_H=strfind(readme_mat(1,1:pos_units(2)),'H');
                        W=str2double(readme_mat(1,(pos_W+2):(pos_units(1)-1)))*units;
                        H=str2double(readme_mat(1,(pos_H+2):(pos_units(2)-1)))*units;
                        apertura.forma(i)='O';
                        apertura.dim1(i)=W;
                        apertura.dim2(i)=H;
                    end
                end
            end
        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     elseif apertura.forma(i,1)=='R'
        temp=strfind(pos,'X');
        tempcoma=pos(9:temp(1)-1);
        poscoma=strfind(tempcoma,',');

        if isempty(poscoma)
            apertura.dim1(i)=str2double(tempcoma);
        else
            tempcoma(poscoma)='.';
            apertura.dim1(i)=str2double(tempcoma);
        end

        temp1=strfind(pos,'*');
        tempcoma=pos(temp(1)+1:temp1(1)-1);
        poscoma=strfind(tempcoma,',');
        if isempty(poscoma)
            apertura.dim2(i)=str2double(tempcoma);
        else
            tempcoma(poscoma)='.';
            apertura.dim2(i)=str2double(tempcoma);
        end

    end
end
end
```

Load_data_gerber.m:

```
%Funcion load_data_gerber
%{
Version 3.0
Creado antes de 07/10/2010 - 7:25pm
```

```
Actualizado 15/05/2014 - 5:00pm
Desarrollado por:   William Gonzalez Coquel
                    Hector Fabio Reyes
Funcion load_data_gerber: Esta funcion carga los datos desde el archivo
Gerber y los carga a la varible infor, la variable de salida tamrutmat nos
dice las coordenadas maxima y minimas para "x" y "y", el formato es el
siguiente:
tamrutmat=[tam_y_mas_esp, tam_x_mas_esp, min_valor_y, min_valor_x];
entiendase tam como tamaño y esp como espesor.
%}
%%
function [datospista,apertura,tamrutmat]=load_data_gerber(filename,resolucion)
%Se carga la informacion del Gerber en la Variable gerberfile
    tamname=length(filename);
    if filename(1,tamname-3:tamname)=='.gb1'
        fuente='pcbw';
        gerberfile=importdata(filename);
        %Carga la informacion del Gerber donde estan las dimensiones en la variable
        %infortam
        filenametam=filename;
        filenametam(1,tamname)='0';
        gerbertam=importdata(filenametam);
    %Se obtiene las dimensiones numericas de la variable importam donde estan
    %almacenada
    [datospistatam,apertura]=conver(gerbertam,fuente);

    ejextam=datospistatam(:,1);
    ejeytam=datospistatam(:,2);

    maxx=max(ejextam);
    maxy=max(ejeytam);
    minx=min(ejextam);
    miny=min(ejeytam);
    tamrutmat(1)=maxy-miny+2*datospistatam(1,4); %En y
    tamrutmat(2)=maxx-minx+2*datospistatam(1,4); %En x
    tamrutmat(3)=miny-datospistatam(1,4);      %Minimo valor de y
    tamrutmat(4)=minx-datospistatam(1,4);      %Minimo valor de x
    [datospista,apertura]=conver(gerberfile,fuente);
    datospista(:,1)=ceil((datospista(:,1)-tamrutmat(4))*resolucion);
    datospista(:,2)=ceil((datospista(:,2)-tamrutmat(3))*resolucion);

    %% Provisional para proteus
    elseif isempty(strfind(filename,'CADCAM Bottom Copper.TXT'))==0
        fuente='proteus';
        gerberfile=importdata(filename,' ');
        filenametam=strcat(filename(1,1:tamname-24),' CADCAM READ-ME.TXT');
        readme=importdata(filenametam,'\t');
        save('data\readme.mat','readme');
        [datospista,apertura]=conver(gerberfile,fuente);

        ejextam=datospista(:,1);
        ejeytam=datospista(:,2);

        maxx=max(ejextam);
        maxy=max(ejeytam);
        minx=min(ejextam);
        miny=min(ejeytam);
        tamrutmat(1)=maxy-miny;%+2*datospistatam(1,4); %En y
        tamrutmat(2)=maxx-minx;%+2*datospistatam(1,4); %En x
        tamrutmat(3)=miny;%-datospistatam(1,4);      %Minimo valor de y
        tamrutmat(4)=minx;%-datospistatam(1,4);      %Minimo valor de x
        % agregando resolucion a la matriz
        datospista(:,1)=ceil((datospista(:,1)-tamrutmat(4))*resolucion)+1;
        datospista(:,2)=ceil((datospista(:,2)-tamrutmat(3))*resolucion)+1;
    else
        fuente='otra';
        gerberfile=importdata(filename,' ');
        [datospista,apertura]=conver(gerberfile,fuente);
        ejextam=datospista(:,1);
        ejeytam=datospista(:,2);
```

```
        maxx=max(ejextam);
        maxy=max(ejeytam);
        minx=min(ejextam);
        miny=min(ejeytam);
        tamrutmat(1)=maxy-miny;%+2*datospistatam(1,4); %En y
        tamrutmat(2)=maxx-minx;%+2*datospistatam(1,4); %En x
        tamrutmat(3)=miny;%-datospistatam(1,4);      %Minimo valor de y
        tamrutmat(4)=minx;%-datospistatam(1,4);      %Minimo valor de x
        % agregando resolucion a la matriz
        datospista(:,1)=ceil((datospista(:,1)-tamrutmat(4))*resolucion)+1;
        datospista(:,2)=ceil((datospista(:,2)-tamrutmat(3))*resolucion)+1;

    end
end
```

PADS.m:

```
function [auxpad]=PADS(forma,diam1,diam2,res)
%macro C=1
%macro R=3
%macro O=2
%res=10;%meter resolucion
%forma; meter macro, segun el valor hace referencia a una figura C, R, O
s1=diam1;%meter ancho en x
s2=diam2;%meter alto en y

c=abs(s2-s1);
if forma=='R'%si es un Rectangulo
    auxpad=uint8(zeros(s2,s1)*255);
else
    if (s1<s2 || s1==s2 && forma=='O')
        diam1=0.5*s1;
    elseif (s2<s1 && forma=='O')
        diam1=0.5*s2;
    elseif forma=='C'
        diam1=0.5*s1;
    end

    if forma=='C' || forma=='O'
        if forma=='C'
            auxpad=uint8(ones(s1,s1)*255);
        end
        angulo=0.1:1/res:360;
        x=ceil(diam1+ diam1*cosd(angulo));
        y=ceil(diam1+ diam1*sind(angulo));
            if forma=='O'
                auxpad=uint8(ones(s2,s1)*255);
               if s2>s1
                   auxpad(diam1:diam1+c,1:s1)=0;
                   y1=ceil(c+y);
               elseif s1>s2
                   auxpad(1:s2,diam1:diam1+c)=0;
                   x1=ceil(c+ x);%a
               end
           end
      %comienzo a modificar la matriz
        if forma=='C'%si es un ciculo
            for k=1:length(y)
                    if (x(k)~=0)&&(y(k)~=0)
                        if ceil(diam1)<=x(k)
                            auxpad(y(k),ceil(diam1):x(k))=0;
                        else
                            auxpad(y(k),ceil(diam1):-1:x(k))=0;
                        end
```

```
                    end
                end
            elseif forma=='O'%si es un ovalo
                if s2>s1     %ovalo vertical
                    for k=1:length(y)
                    if x(k)~=0
                        if y(k)~=0
                            if ceil(diam1)<=x(k)
                                auxpad(y(k),ceil(diam1):x(k))=0;
                            else
                                auxpad(y(k),ceil(diam1):-1:x(k))=0;
                            end
                        end
                    end
                    end
                    for k=1:length(y)
                    if x(k)~=0
                        if y1(k)~=0
                            if ceil(diam1)<=x(k)
                                auxpad(y1(k),ceil(diam1):x(k))=0;
                            else
                                auxpad(y1(k),ceil(diam1):-1:x(k))=0;
                            end
                        end
                    end
                    end
                elseif s1>s2 %ovalo horizontal
                    for k=1:length(y)
                    if x(k)~=0
                        if y(k)~=0
                            if ceil(diam1)<=x(k)
                                auxpad(y(k),ceil(diam1):x(k))=0;
                            else
                                auxpad(y(k),ceil(diam1):-1:x(k))=0;
                            end
                        end
                    end
                    end
                    for k=1:length(y)
                    if x1(k)~=0
                        if y(k)~=0
                            if ceil(diam1+c)<=x1(k)
                                auxpad(y(k),ceil(diam1+c):x1(k))=0;
                            else
                                auxpad(y(k),ceil(diam1):-1:x1(k))=0;
                            end
%                             for j=ceil(diam1+c):x1(k)
%                                 auxpad(y(k),j)=0;
%                             end
%                             for j=ceil(diam1):-1:x1(k)
%                                 auxpad(y(k),j)=0;
%                             end
                        end
                    end
                    end
                end
            end
        end
    end
```

Ruteo_mat.m:

```
function [rutmat]=ruteo_mat(resolucion,tamrutmat,ejex,ejey,datospistas,apertura)
%En esta funcion se crea la matriz rutmat en la cual esta la imagen de la
```

```
%pcb cargada desde el archivo Gerber, esta funcion recibe como parametros
%las coordenadas x y y (ejex y ejey), un vector de ruteo que nos indica si
%es inicio o final de pista y los agujeros ( D01, D02 y D03 los D03 no se
%trabajan en esta funcion, recibe un parametro de resolucion, el espesor de
%las pistas entre otros parametros.
% datospistas=[ejex,ejey,ruteo,diam1,diam2,posiciona_lista_aperutra]
%Nota: ruteo indica si es pista, punto o agujero

%en y
limite_posy=ceil(tamrutmat(1)*resolucion);
%en x
limite_posx=ceil(tamrutmat(2)*resolucion);

tammaxx=max(ejex);
tamminx=min(ejex);
tammaxy=max(ejey);
tamminy=min(ejey);
ruteo1=datospistas(:,3);
tam=size(ruteo1);
%Declaro la matriz rutmat de acuerdo con el tamaño del PCB declarado en el
%programa de PCB y respecto a la resolucion del archivo rutmat(y,x)
rutmat=ones(ceil(tamrutmat(1)*resolucion),ceil(tamrutmat(2)*resolucion),'uint8')*255;
aux_num_list_apertura=0;
tempad=ones(1,1);
for i=1:tam(1)
    diam=round((datospistas(i,4)/2)*resolucion);

[rutmat,tempad]=circulomatri(rutmat,apertura,datospistas(i,6),datospistas(i,4),datospistas(i
,5),aux_num_list_apertura,ejex(i),ejey(i),resolucion,tempad,limite_posy,limite_posx);
    aux_num_list_apertura=datospistas(i,6);
    if ruteo1(i)==1
%% Determinamos cual es el mayor para cada coordenada para trabajar las operaciones de forma
organizada
        ax=ejex(i-1);
        ax1=ejex(i);
        ay=ejey(i-1);
        ay1=ejey(i);
        if ax>ax1
            xmax=ax;
            xmin=ax1;
        else
            xmax=ax1;
            xmin=ax;
        end
        if ay>ay1
            ymax=ay;
            ymin=ay1;
        else
            ymax=ay1;
            ymin=ay;
        end
        %% Pimero Evaluamos par alas lineas Verticales
        if ((ax==ax1)&&(ay~=ay1))
        %rutmat(ymin:ymax,ax1-diam:ax1+diam)=0; Esto se modifico a lo siguiente
            if ax1+diam<=limite_posx
                if ax1-diam>=1
                    rutmat(ymin:ymax,ax1-diam:ax1+diam)=0;
                else
                    rutmat(ymin:ymax,1:ax1+diam)=0;
                end
            else
                if ax1-diam>=1
                    rutmat(ymin:ymax,ax1-diam:limite_posx)=0;
                else
                    rutmat(ymin:ymax,1:limite_posx)=0;
                end
            end
        % Luego Evaluamos par alas lineas Verticales
        elseif ((ay==ay1)&&(ax~=ax1))
        %rutmat(ay-diam:ay+diam,xmin:xmax)=0; Esto se modifico a lo siguiente
            if ay+diam<=limite_posy
```

```
            if ay-diam>=1
                rutmat(ay-diam:ay+diam,xmin:xmax)=0;
            else
                rutmat(1:ay+diam,xmin:xmax)=0;
            end
        else
            if ay-diam>=1
                rutmat(ay-diam:limite_posy,xmin:xmax)=0;
            else
                rutmat(1:limite_posy,xmin:xmax)=0;
            end
        end
    % Luego Evaluamos para las lineas diagonales o inclinaciones
    elseif(ax~=ax1 && ay~=ay1)
        m=(ay1-ay)/(ax1-ax);
        alpha=atan(m); %angulo de inclinacion
        theta=0.5*pi-alpha;
%       l3=diam*cos(alpha);
        if (abs(alpha)<= pi/4)% alpha menor 45
            l4=round(abs(diam*sin(alpha)));
            l5=round(abs(diam/cos(alpha)));
            if(m>0)%pendiente positiva
                %mitad superior lado izq
                for x=xmin-l4:xmax-l4
                    y=abs(ceil(m*(x-ax)+ay));
                    for j=y:y+l5
                        if x>=1
                            if j<=limite_posy && j>0
                                rutmat(j,x)=0;
                            else
                                rutmat(limite_posy,x)=0;
                            end
                        else
                            if y<=limite_posy && j>0
                                rutmat(j,1)=0;
                            else
                                rutmat(limite_posy,1)=0;
                            end
                        end
                    end
                end
                %mitad inferior lado der
                for x=xmin+l4:xmax+l4
                    y=abs(ceil((m*(x-ax)+ay)));
                    for j=y:-1:y-l5
                        if x<=limite_posx && x>0
                            if j>=1
                                rutmat(j,x)=0;
                            else
                                rutmat(1,x)=0;
                            end
                        else
                            if j>=1
                                rutmat(j,limite_posx)=0;
                            else
                                rutmat(1,limite_posx)=0;
                            end
                        end
                    end
                end
            elseif(m<0)%pendiente negativa
                %mitad inferior lado izq
                for x=xmin-l4:xmax-l4;
                    y=abs(ceil((m*(x-ax)+ay)));
                    for j=y:-1:y-l5
                        if x>=1
                            if j>=1
                                rutmat(j,x)=0;
                            else
                                rutmat(1,x)=0;
                            end
```

```
                    else
                        if j>=1
                            rutmat(j,1)=0;
                        else
                            rutmat(1,1)=0;
                        end
                    end
                end
            end
            %mitad superior lado der
            for x=xmin+l4:xmax+l4
                y=abs(ceil((m*(x-ax)+ay)));
                for j=y:y+l5
                    if x<=limite_posx && x>0
                        if j<=limite_posy && j>0
                            rutmat(j,x)=0;
                        else
                            rutmat(limite_posy,x)=0;
                        end
                    else
                        if j<=limite_posy && j>0
                            rutmat(j,limite_posx)=0;
                        else
                            rutmat(limite_posy,limite_posx)=0;
                        end
                    end
                end
            end
        end
    else%alpha mayor que 45
        l4=round(abs(diam*sin(theta)));
        l5=round(abs(diam/cos(theta)));
        if(m>0)%pendiente positiva
            %mitad inferior lado der
            for y=ymin-l4:ymax-l4;
                x=abs(ceil(((y-ay)/m)+ax));
                for j=x:x+l5
                    if y>=1
                        if j<=limite_posx && j>0
                            rutmat(y,j)=0;
                        else
                            rutmat(y,limite_posx)=0;
                        end
                    else
                        if j<=limite_posx && j>0
                            rutmat(1,j)=0;
                        else
                            rutmat(1,limite_posx)=0;
                        end
                    end
                end
            end
            %mitad superior lado izq
            for y=ymin+l4:ymax+l4
                x=abs(ceil(((y-ay)/m)+ax));
                for j=x:-1:x-l5
                    if y<=limite_posy  && y>0
                        if j>=1
                            rutmat(y,j)=0;
                        else
                            rutmat(y,1)=0;
                        end
                    else
                        if j>=1
                            rutmat(limite_posy,j)=0;
                        else
                            rutmat(limite_posy,1)=0;
                        end
                    end
                end
            end
```

```
                elseif(m<0)%pendiente negativa
                    %mitad inferior lado izq
                    for y=ymin-l4:ymax-l4;
                        x=abs(ceil(((y-ay)/m)+ax));
                        for j=x:-1:x-l5
                            if y>=1
                                if j>=1
                                    rutmat(y,j)=0;
                                else
                                    rutmat(y,1)=0;
                                end
                            else
                                if j>=1
                                    rutmat(1,j)=0;
                                else
                                    rutmat(1,1)=0;
                                end
                            end
                        end
                    end
                    %mitad superior lado der
                    for y=ymin+l4:ymax+l4
                        x=abs(ceil(((y-ay)/m)+ax));
                        for j=x:x+l5
                            if y<=limite_posy && y>0
                                if j<=limite_posx && j>0
                                    rutmat(y,j)=0;
                                else
                                    rutmat(y,limite_posx)=0;
                                end
                            else
                                if j<=limite_posx && j>0
                                    rutmat(limite_posy,j)=0;
                                else
                                    rutmat(limite_posy,limite_posx)=0;
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end

end
```

Usb_inout.m:

```
function [ output_args ] = usb_inout( input_args )
%Esta funcion envia la informacion por usb a la maquina y la recibe

end
```

Validcor.m:

```
function [coordenadax,coordenaday,accion_rut]=validcor(ejeconver,antejexx,antejeyy)
a=strfind(ejeconver,'X'); %partiendo de que el primer numero siempre esta completo
b=strfind(ejeconver,'Y');
c=strfind(ejeconver,'D');
d=strfind(ejeconver,'D02');
if(isempty(a)) %si no hay X se cumple
    coordenadax=antejexx;
else
    if(isempty(b))
        coordenadax=ejeconver(a+1:c-1);
    else
        coordenadax=ejeconver(a+1:b-1);
    end
end

if(isempty(b))
    coordenaday=antejeyy;
else
    coordenaday=ejeconver(b+1:c-1);
end
val=size(ejeconver);
accion_rut=ejeconver(val(2)-1);
```

SCRIPT

Load_file.m:

```
%Script load_file
%{
Version 3.0
Creado antes de 07/10/2010 - 7:25pm
Actualizado 15/05/2014 - 3:00pm
Desarrollado por:   William Gonzalez Coquel
                    Hector Fabio Reyes
%}
%% Se Carga el Gerber y se obtienen los vectores X y Y
%Se Agrega El Directorio
path(path,pathGerber)
ultimalinea=1;
save data/detener.mat ultimalinea -append;
%Se Visualiza Barra animada del proceso
h=waitbar(0,'Este proceso puede tardar unos segundos','Name','Cargando datos...');
%Se carga el contenido del archivo Gerber en la
[datospista,apertura,tamrutmat]=load_data_gerber(filename,resolucion);
%% temporal
ejex=datospista(:,1);
ejey=datospista(:,2);
ruteo1=datospista(:,3);
%% Este segmento de codigo calcula la rutmat principal y la grafica en axis
[rutmat]=ruteo_mat(resolucion,tamrutmat,ejex,ejey,datospista,apertura); %Funcion que llena
una matriz a partir de archivo gerber
% figure(123);
% imshow(rutmat);

axes(handles.grafica_pcb);%indica que se graficara en grafica_pcb
imshow(rutmat);
% axes(handles.grafica_copiaspcb);%indica que se graficara en grafica_copiaspcb
% imshow(rutmat);
handles.rutmat=rutmat;%almacena la variable en handles
temp_imagen=imresize(rutmat,0.1);
imwrite(temp_imagen,'temp\pcb.jpg');
```

```
clear rutmat
%% Este segmento de codigo calcula el contorno teniendo en cuenta el diametro de la broca
(diam_broca) y la distancia entre las pistas.
%profile on%Momentanea borrar despues

%sumamos el diametro de la broca para obtener el espesor adecuado.
datospista(:,4)=datospista(:,4)+diam_broca;
datospista(:,5)=datospista(:,5)+diam_broca;
[rutmat]=ruteo_mat(resolucion,tamrutmat,ejex,ejey,datospista,apertura); %Funcion que llena
una matriz a partir de archivo gerber
% figure(1);
% imshow(rutmat);

%Muestra barra de espera
waitbar(0.4);
[machineaux1]=contornospcb(rutmat);
clear rutmat
[matmachine]=coordenadaCNC(machineaux1);
%Se empieza a calcular los contornos el contorno obtenido al llamar esta
%funcion por si sola es el contorno de la pista. Pero estos datos no se
%pueden mandar a la maquina por el diametro de la broca ya que el centro de
numfalt_pasadas=(gap/diam_broca)-1;%numero de pasadas faltantes
i=1;
%Muestra barra de espera
waitbar(0.5);
%% Para mas de una linea de contorno en la pista (GAP)
while(i<=numfalt_pasadas)
    datospista(:,4)=datospista(:,4)+diam_broca;
    datospista(:,5)=datospista(:,5)+diam_broca;
    [rutmat]=ruteo_mat(resolucion,tamrutmat,ejex,ejey,datospista,apertura); %Funcion que
llena una matriz a partir de archivo gerber
    %figure(i);
    %imshow(rutmat);
    [machineaux1]=contornospcb(rutmat);
    size(rutmat) %Momentanea borrar despues
    clear rutmat %Probar si es mas rapido quitandolo o dejandolo
    [matmachine_temp]=coordenadaCNC(machineaux1);
    matmachine=[matmachine;matmachine_temp];
    %Se empieza a calcular los contornos el contorno obtenido al llamar esta
    %funcion por si sola es el contorno de la pista. Pero estos datos no se
    %pueden mandar a la maquina por el diametro de la broca ya que el centro de
    i=i+1;
end
%_____________________________________________________________________________
[matmachine_agujeros]=coordenadaCNC_agujeros(ejex,ejey,ruteo1);
%se crea una nueva matriz solo para posiciones de ruteo, para mantener
%datos de ruteo y perforacion separados.
%_____________________________________________________________________________
hold off;
matrizcopiado=matmachine;
save data\matriz_ruteo.mat matrizcopiado matmachine_agujeros;
waitbar(0.9);
%% Finalizalamos el waitbar y liberamos espacio en memoria
%profile viewer%Momentanea borrar despues
%profile off%Momentanea borrar despues
clear ejex ejey espesor filename config matrizcopiado
waitbar(1,h);
close(h);
%%
% Una vez cargado y procesado el Gerber se habilitan los objetos en la ventana
if isfield(handles,'matrizcopiado_dec')==1
handles=rmfield(handles,'matrizcopiado_dec');
end
if isfield(handles,'matriz_agu_dec')==1
handles=rmfield(handles,'matriz_agu_dec');
end

handles.matrizcopiado_dec(:,1:2)=matmachine(:,1:2)/resolucion;
handles.matrizcopiado_dec(:,3)=config_ejes.z.recorrido-matmachine(:,3);
handles.matrizcopiado_dec(:,4)=matmachine(:,4);
handles.matriz_agu_dec(:,1:2)=matmachine_agujeros(:,1:2)/resolucion;
```

```
handles.matriz_agu_dec(:,3)=95-matmachine_agujeros(:,3);
handles.matriz_agu_dec(:,4)=matmachine_agujeros(:,4);
set(handles.tabla_machine,'enable','on','data',handles.matrizcopiado_dec);
set(handles.rutear_machine,'enable','on');%habilita boton
set(handles.perforar_machine,'enable','on');%habilita boton
%set(handles.pausa_machine,'enable','on');%habilit boton
%set(handles.stop_machine,'enable','on');%habilit boton
set(handles.op_ruteo_copias,'enable','on');%habilit opcion en menu de configuracion
set(handles.radiobutton_rutear,'enable','on');
set(handles.radiobutton_perforar,'enable','on');
set(handles.op_linea,'enable','on');
handles.uipanel_botones_tabla=handles.radiobutton_rutear;
% Agregamos las variables resolucion, matmachine, tamrutmat al handles principal y se guarda
como handlesmadre en manejadores.mat
handles.matmachine=matmachine;%almacena la variable en handles
handles.tamrutmat=tamrutmat;%almacena la variable en handles
guidata(hObject, handles);%sentencia que crea el objeto y guarda las variables antes
mencionadas
handlesmadre=handles;%copia handles
save data\manejadores.mat handlesmadre;%almacena handlesmadre en manejadores.mat
% se Libera espacio en memoria
clear tamrutmat%limipa tamrutmat
```

Script_copias.m:

```
%% script_copias
%el algoritmo consiste en hacer primero copias horizontales hasta donde
%pueda luego se hace una copia de forma vertical de las copias horizontales
%hechas anteriormente, es decir, se hacen por ejemplo 3 copias horizontales
%y luego esas 3 se copias de forma vertical por ejemplo otras 3 veces para
%tener un total de 3x3= 9 copias. pero el algoritmo debe ser capaz de hacer
%copias que no llenen todo el espacio, es decir, que si en la placa caben
%solo 9 copias en formato 3 horizontales x 3 verticales, el algoritmo debe
%poder hacer menos de 9 copias y mas de 6, dando lugar a fomarciones que no
%sean cuadradas como se muestra a continuacion:
%x x x
%x x x
%x x
%para ello es necesario saber si el numero de copias deseado es mayor que
%las permitidas
%% Este segmento de codigo recopila los datos necesarios para hacer la copia
altoplaca=str2double(get(handles.alto_placa,'string'));                    %obtiene el
valor de alto_placa
anchoplaca=str2double(get(handles.ancho_placa,'string'));                  %obtiene el
valor de ancho_placa
tambaquelita_alto=handlesmadre.tamrutmat(1);                               %obtiene el
valor de alto de la vaquelita
tambaquelita_ancho=handlesmadre.tamrutmat(2);                              %obtiene el
valor de ancho de la vaquelita
sizerutmat=size(handlesmadre.rutmat);                                      %obtiene filas y
columnas de rutmat de la siguiente forma sizerutmat(1)=alto y sizerutmat(2)=ancho
numerocopias=str2double(get(handles.num_copias,'string'));                 %obtiene el
valor de num_copias
maxcop_alto=floor(altoplaca/tambaquelita_alto);                            %calcula el
valor maximo de copias verticales
maxcop_ancho=floor(anchoplaca/tambaquelita_ancho);                         %calcula el
valor maximo de copias horizontales
img_pcb=imread('pcb.jpg');
tam_pcb=size(img_pcb);
guardar=0;
%%
if isnan(altoplaca)~=1 && isnan(anchoplaca)~=1
```

```
    if altoplaca>0 && anchoplaca>0 && altoplaca<config_ejes.y.recorrido &&
anchoplaca<config_ejes.x.recorrido            %si el valor de alto y ancho de la placa son
diferentes de 0 hace las copias
        maxcop=maxcop_alto * maxcop_ancho;                                      %calcula el
valor maximo de copias en toda la placa
        if numerocopias>maxcop                                                  %si el numero
deseado llega a ser mayor que el permitido
            %% Se ejecuta cuando el numero de copias introducido es mayor al permitido
            handler_msgbox=msgbox(cat(2,'El maximo numero de copias posibles en la placa es
',num2str(maxcop)),'','warn');%lanza un mensaje
            set(handles.num_copias,'string',num2str(maxcop));                    %y modifica
el valor introducido por el usuario al valor maximo permitido
        else
            %% Se ejecuta cuando el numero de copias introducido es menor al permitido
            numnec_alto=(numerocopias/maxcop_ancho);
            %% Se calcula el numero de filas en la matriz de copia
            if numnec_alto<1
                numnec_alto=1;
            else
                numnec_alto=ceil(numnec_alto);
            end
            %% Se crea la imagen para visualizar
            if numerocopias>maxcop_ancho

img_pcb_copy=ones(ceil(tam_pcb(1)*numnec_alto),ceil(tam_pcb(2)*maxcop_ancho),'uint8')*255;
            else

img_pcb_copy=ones(ceil(tam_pcb(1)*numnec_alto),ceil(tam_pcb(2)*numerocopias),'uint8')*255;
            end
            %% Se calcula el numero de copias faltantes
            numfaltcopias=numerocopias - (maxcop_ancho*(numnec_alto-1));%calcula el numero
de copias que hacen que el formato de copiado deje de ser cuadrado
            control_ini=1;

            %% Se empieza a copiar los datos Primero de forma horizontal y luego de forma
vertical
            for nalto=0:(numnec_alto-1)
                %% Se controla el ancho por fila
                if numfaltcopias~=0 && nalto==(numnec_alto-1)
                    numnec_ancho=numfaltcopias;
                else
                    numnec_ancho=maxcop_ancho;
                end
                %% Se empieza a copiar los datos de forma horizontal
                for nancho=0:(numnec_ancho-1)

matcalcopiado=[handlesmadre.matmachine(:,1)+(nalto*sizerutmat(1)),handlesmadre.matmachine(:,
2)+(nancho*sizerutmat(2)),handlesmadre.matmachine(:,3),handlesmadre.matmachine(:,4)];

img_pcb_copy(1+tam_pcb(1)*nalto:tam_pcb(1)+tam_pcb(1)*nalto,1+tam_pcb(2)*nancho:tam_pcb(2)+t
am_pcb(2)*nancho)=img_pcb(:,:);
                    if control_ini==1
                        matrizcopiado=matcalcopiado;
                        control_ini=0;
                    else
                        matrizcopiado=[matrizcopiado;matcalcopiado];
                    end
                end
            end
            guardar=1;

        end
    else%si el valor de alto y ancho de la placa NO son mayores de 0 muestra el mensaje
        msgbox(cat(2,'Introdusca un ancho y alto de la placa mayor a 0 para poder calcular
las copias. Estos valores no deben ser mayores a ancho=',num2str(config_ejes.x.recorrido),'
alto=',num2str(config_ejes.y.recorrido)),'warn');
    end
else
    msgbox('Introduzca solo numeros.','warn');
end
```

Script_decbin.m:

```
entero_USB=fix(variable);
%validar siempre a 1 byte
dcp=variable-entero_USB;% dcp: decimal con punto
dcppota1=uint8(fix(variable));%valor entero
dcppot=dcp*10*10*10*10; %corro la como cuatro digitos a la derecha
dcppota2=uint8(bitshift(uint16(dcppot),-8,8));%H
dcppota3=uint8(bitshift(uint16(dcppot),0,8));%L
```

Script_decstrdec.m:

```
dcppota1=uint8(floor(variable));
decimal1=(variable-floor(variable))*100;
dcppota2=uint8(floor(decimal1));
dcppota3=uint8(floor((decimal1-floor(decimal1))*100));
```

Script_encoder.m:

```
%script_encoder modificado 2/8/2014:
%traduce todos los datos de ruteo y perforacion a valores de encoder. se
%seleccionan los datos a trabajar a partir de la operacion desde donde se
%haga el llamado.
%carga matriz_ruteo.mat que contiene la matriz de agujeros y la matriz de
%ruteo
load matriz_ruteo.mat
load parametros.mat
if operacion=='r'
    %tabla_encoder=matrizcopiado;

tabla_encoder=[round(matrizcopiado(:,1)*(config_ejes.y.enc_supr_bits/(config_ejes.y.paso*res
olucion))),round(((config_ejes.x.recorrido*resolucion)-
matrizcopiado(:,2))*(config_ejes.x.enc_supr_bits/(config_ejes.x.paso*resolucion))),round(mat
rizcopiado(:,3)*(config_ejes.z.enc_supr_bits)/config_ejes.z.paso),matrizcopiado(:,4)];
elseif operacion =='p'
    %tabla_encoder=matmachine_agujeros;

tabla_encoder=[round(matmachine_agujeros(:,1)*(config_ejes.y.enc_supr_bits/(config_ejes.y.pa
so*resolucion))),round(((config_ejes.x.recorrido*resolucion)-
matmachine_agujeros(:,2))*(config_ejes.x.enc_supr_bits/(config_ejes.x.paso*resolucion))),rou
nd(matmachine_agujeros(:,3)*(config_ejes.z.enc_supr_bits)/config_ejes.z.paso),matmachine_agu
jeros(:,4)];
elseif operacion == 'c'
    %tabla_encoder=matriz_manual;

tabla_encoder=[round(matriz_manual(:,1)*(config_ejes.y.enc_supr_bits/(config_ejes.y.paso))),
round(((config_ejes.x.recorrido)-
matriz_manual(:,2))*(config_ejes.x.enc_supr_bits/(config_ejes.x.paso))),round(matriz_manual(
:,3)*(config_ejes.z.enc_supr_bits)/config_ejes.z.paso),matriz_manual(:,4)];
```

```
end
%% Llevamos los valores de encoder a registros de 2 bytes y de paso corregimos la posicion
de x y y en la tabla
tabla_encoder_uint8(:,1)=uint8(bitshift(tabla_encoder(:,2),-8,8));
tabla_encoder_uint8(:,2)=uint8(bitshift(tabla_encoder(:,2),0,8));
tabla_encoder_uint8(:,3)=uint8(bitshift(tabla_encoder(:,1),-8,8));
tabla_encoder_uint8(:,4)=uint8(bitshift(tabla_encoder(:,1),0,8));
tabla_encoder_uint8(:,5)=uint8(bitshift(tabla_encoder(:,3),-8,8));
tabla_encoder_uint8(:,6)=uint8(bitshift(tabla_encoder(:,3),0,8));
tabla_encoder_uint8(:,7)=uint8(tabla_encoder(:,4));
save data\enc_control.mat tabla_encoder tabla_encoder_uint8;
%% Se crea la trama del USB
```

Script_pausa.m:

```
if pausa==1
    ultimalinea=i;
    pausa=0;
    i=filas+1;
    msgbox('Ha pausado el proceso, para continuar pulse el boton RUTEAR',' Proceso
detenido.');
    %apagar efector final y subir dejando en ultimalinea, al
    %continuar regresar a ultima linea
end
```

Script_preview.m:

```
load manejadores.mat;
if(str2double(get(handles.num_copias,'string')))>1
    script_copias;
    if guardar==1;
        axes(handles.grafica_preview);
        imshow(img_pcb_copy);
    end
else
    set(handles.alto_placa,'enable','off');%cuando edita num_copias se habilita alto_placa
para la edicion
    set(handles.ancho_placa,'enable','off');%cuando edita num_copias se habilita alto_placa
para la edicion
    axes(handles.grafica_preview);
    imshow(handlesmadre.rutmat);
end
```

Script_relacionmotor.m:

```
setear=0;
if valor1~=0 && valor2~=0
    relacion=valor1/valor2;
    pulsosreductor=valor3 * relacion;
    numerobits=size(dec2bin(pulsosreductor),2);
    %numerobits=str2double(get(handles.edit_numbits_x));
    cadena_supresion={'2'};
```

```
    %cadena_supresion=ones(numerobits,1);
    for i=2:numerobits-1
       cadena_supresion=[cadena_supresion;num2str(2^i)];
    end
    setear=1;
else
    msgbox('Introduzca solo numeros con valores mayores a 0');
end
```

Script_stop.m:

```
if operacion=='c'
    i=1;
    filas=0;
end
if stop==1
    ultimalinea=1;
    stop=0;

    data_out(5)=tabla_encoder_uint8(i,1); %XH
    data_out(6)=tabla_encoder_uint8(i,2); %XL
    data_out(25)=tabla_encoder_uint8(i,3); %YH
    data_out(26)=tabla_encoder_uint8(i,4); %YL
    data_out(45)=tabla_encoder_uint8(1,5); %ZH
    data_out(46)=tabla_encoder_uint8(1,6); %ZL
    data_out(63)=0; %EF
    data_out(62)=3; %EF
    i=filas+1;
    calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(64), uint8(64),
uint8(10));
    [aa,bb,data_in,dd]=calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64),
uint8(64), uint8(10));
    msgbox('Ha detenido el proceso',' Proceso detenido.');
    %apagar y efector final enviar a origen
end
```

Script_usb.m:

```
%%
% Formato data
% cabeza es una variable que toma su valor en la funcion que llame al
% script, ya sea una funcion de configuracion (203) o de control (202)
sentido_giro=0;
dato_anterior=zeros(1,3);
a=0;
%%

loadlibrary mpusbapi _mpusbapi.h alias libreria
% Los archivos _mpusbapi.c y mpusbapi.dll deben de estar en la misma
% carpeta de trabajo y se obtienen de la descarga del driver en la
% pagina de microchip ("Microchip MCHPFSUSB v2.4 Installer.zip"),
% al instalarse queda ubicado en X:\Microchip Solutions\USB
% Tools\MCHPUSB Custom Driver\Mpusbapi\Dll\Borland_C, en caso de descargar
% una version de driver más reciente, reemplace éstos archivos por los más
% nuevos.
```

```
%libisloaded libreria                % Confirma que la librería ha sido
                                     % cargada
%libfunctions('libreria', '-full')   % Muestra en la línea de comandos las
                                     % funciones de la librería
%libfunctionsview libreria           % Muestra en un cuadro lo mismo que la
                                     % instrucción anterior

data_in = zeros(1,64,'uint8');       % Se declara el vector de datos de entrada (el que se
recibe del PIC)
data_out = zeros(1,64,'uint8');      % Se declara el vector de datos de salida (el que se
envia al PIC)
                                    % TODOS LOS DATOS SE DECLARAN COMO
                                    % UINT8 de lo contrario no hay
                                    % comunicación.

vid_pid_norm = libpointer('int8Ptr',[uint8('vid_04d8&pid_000b') 0]);
out_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);
in_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);
%calllib('libreria','MPUSBGetDLLVersion');
[conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm)

if conectado == 1   % Es importante seguir ésta secuencia para comunicarse con el PIC:
                    % 1. Abrir tuneles, 2. Enviar/Recibir dato
                    % 3. Cerrar tuneles
    mensaje=msgbox('Enviando datos a la maquina, por favor no cierre el software ni
desconecte la maquina durante el proceso',' Ruteando ','ok');
     set(handles.pausa_machine,'enable','on');%habilita boton pausar
     set(handles.stop_machine,'enable','on');%habilita boton detener
    [my_out_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, out_pipe,
uint8(0), uint8 (0)); % Se abre el tunel de envio
    [my_in_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, in_pipe, uint8
(1), uint8 (0)); % Se abre el tunel de recepción

    %utilizar save 'archivo.mat' variables '-append'; para agregar o actualizar
    %datos en el archivo sin borrar los ya contenidos.
    %armamos la trama a enviar por USB
    data_out(1)=cabeza;
    % si la cabeza es igual a 202 (envio de posiciones de ruteo)
    if data_out(1)==4
        load enc_control.mat;
        [filas,columnas]=size(tabla_encoder_uint8);
        load detener.mat;
        i=ultimalinea;
        %%stop=0;
        data_in = zeros(1,64,'uint8');%colocamos el data_in en cero en cada
        data_in(1)=1;
        int_x=1;
        int_y=1;
        int_z=1;

        while i<=filas
            tic

            load detener.mat;
            [conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm)
            if conectado==0
                ultimalinea=i;
                pausa=0;
                i=filas+1;
                save data\detener.mat pausa -append;
                msgbox('Se ha perdido la conexion USB, reconecte y pulse el boton RUTEAR
para continuar.',' Proceso detenido.');
                %apagar efector final y subir dejando en ultimalinea, al
                %continuar regresar a ultima linea
            else

                if (int_x==1 && int_y==1 && int_z==1)

                    pause(0.050);
                    data_out(5)=tabla_encoder_uint8(i,1); %XH
                    data_out(6)=tabla_encoder_uint8(i,2); %XL
```

```
                    data_out(25)=tabla_encoder_uint8(i,3); %YH
                    data_out(26)=tabla_encoder_uint8(i,4); %YL
                    data_out(45)=tabla_encoder_uint8(i,5); %ZH
                    data_out(46)=tabla_encoder_uint8(i,6); %ZL
                    data_out(63)=tabla_encoder_uint8(i,7); %EF
                    data_out(62)=3; %EF
                    cont_a=0;
                    data_out(63);
                    data_out(5:6);
                    calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(64),
uint8(64), uint8(10));
                    [aa,bb,data_in,dd]=calllib('libreria', 'MPUSBRead',my_in_pipe, data_in,
uint8(64), uint8(64), uint8(10)); % Se recibe el dato que envia el PIC
                    int_x=data_in(3);
                    int_y=data_in(6);
                    int_z=data_in(9);
                    data_out;
                    data_in(1:10);
set(handles.text_positionX,'string',num2str((config_ejes.x.recorrido)-
bin2dec([dec2bin(data_in(1),8),dec2bin(data_in(2),8)])*config_ejes.x.paso/config_ejes.x.enc_
supr_bits));

set(handles.text_positionY,'string',num2str(bin2dec([dec2bin(data_in(4),8),dec2bin(data_in(5
),8)])*config_ejes.y.paso/config_ejes.y.enc_supr_bits));

set(handles.text_positionZ,'string',num2str((config_ejes.z.recorrido)-
bin2dec([dec2bin(data_in(7),8),dec2bin(data_in(8),8)])*config_ejes.z.paso/config_ejes.z.enc_
supr_bits));
                            set(handles.text_linea,'string',num2str(i));
                    if bin2dec([dec2bin(data_in(1),8),dec2bin(data_in(2),8)])>60000
                        data_in(1)=255-data_in(1);
                        data_in(2)=255-data_in(2);
                    end
                    if bin2dec([dec2bin(data_in(4),8),dec2bin(data_in(5),8)])>60000
                        data_in(4)=255-data_in(4);
                        data_in(5)=255-data_in(5);
                    end
                    if bin2dec([dec2bin(data_in(7),8),dec2bin(data_in(8),8)])>60000
                        data_in(7)=255-data_in(7);
                        data_in(8)=255-data_in(8);
                    end
                    error_cnc(1)=abs(bin2dec([dec2bin(data_in(1),8),dec2bin(data_in(2),8)])-
bin2dec([dec2bin(tabla_encoder_uint8(i,1),8),dec2bin(tabla_encoder_uint8(i,2),8)]));
                    error_cnc(2)=abs(bin2dec([dec2bin(data_in(4),8),dec2bin(data_in(5),8)])-
bin2dec([dec2bin(tabla_encoder_uint8(i,3),8),dec2bin(tabla_encoder_uint8(i,4),8)]));
                    error_cnc(3)=abs(bin2dec([dec2bin(data_in(7),8),dec2bin(data_in(8),8)])-
bin2dec([dec2bin(tabla_encoder_uint8(i,5),8),dec2bin(tabla_encoder_uint8(i,6),8)]));
                    error_cnc
                    if error_cnc(1)<=4 && error_cnc(2)<=4 && error_cnc(3)<=2
                        i=i+1;
                    end
                    script_stop;
                    script_pausa;
                else
                    int_x
                    int_y
                    int_z
                    if int_x==255
                        msgbox('El eje X se ha detenido, por favor reinicie la maquina.','
Proceso detenido.');
                        stop=1;
                        script_stop;
                    elseif int_y==255
                        msgbox('El eje Y se ha detenido, por favor reinicie la maquina.','
Proceso detenido.');
                        stop=1;
                        script_stop;
                    elseif int_z==255
                        msgbox('El eje Z se ha detenido, por favor reinicie la maquina.','
Proceso detenido.');
                        stop=1;
```

```
                    script_stop;
                end
            end
        end
    end
    save data\detener.mat stop pausa ultimalinea -append;
end
data_out(1)=0;
calllib('libreria', 'MPUSBClose', my_in_pipe);  % Se cierra el tunel de recepción
calllib('libreria', 'MPUSBClose', my_out_pipe); % Se cierra el tunel de envio
else
set(handles.pausa_machine,'enable','off');%deshabilita boton pausar
set(handles.stop_machine,'enable','off');%deshabilita boton detener
msgbox('Por favor conecte la maquina y pulse nuevamente RUTEAR para continuar el
proceso',' Dispositivo desconectado ');
end
```

Script_usb_manual.m:

```
%%
% Formato data
% cabeza es una variable que toma su valor en la funcion que llame al
% script, ya sea una funcion de configuracion (203) o de control (202)
sentido_giro=0;
dato_anterior=zeros(1,3);
a=0;
loadlibrary mpusbapi _mpusbapi.h alias libreria
% Los archivos _mpusbapi.c y mpusbapi.dll deben de estar en la misma
% carpeta de trabajo y se obtienen de la descarga del driver en la
% pagina de microchip ("Microchip MCHPFSUSB v2.4 Installer.zip"),
% al instalarse queda ubicado en X:\Microchip Solutions\USB
% Tools\MCHPUSB Custom Driver\Mpusbapi\Dll\Borland_C, en caso de descargar
% una version de driver más reciente, reemplace éstos archivos por los más
% nuevos.
%libisloaded libreria               % Confirma que la librería ha sido
                                    % cargada
%libfunctions('libreria', '-full')  % Muestra en la línea de comandos las
                                    % funciones de la librería
%libfunctionsview libreria          % Muestra en un cuadro lo mismo que la
                                    % instrucción anterior
data_in = zeros(1,64,'uint8');       % Se declara el vector de datos de entrada (el que se
recibe del PIC)
data_out = zeros(1,64,'uint8');      % Se declara el vector de datos de salida (el que se
envia al PIC)
                                   % TODOS LOS DATOS SE DECLARAN COMO
                                   % UINT8 de lo contrario no hay
                                   % comunicación.
vid_pid_norm = libpointer('int8Ptr',[uint8('vid_04d8&pid_000b') 0]);
out_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);
in_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);
%calllib('libreria','MPUSBGetDLLVersion');
[conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm)
if conectado == 1   % Es importante seguir ésta secuencia para comunicarse con el PIC:
                    % 1. Abrir tuneles, 2. Enviar/Recibir dato
                    % 3. Cerrar tuneles
    mensaje=msgbox('Enviando datos a la maquina, por favor no cierre el software ni
desconecte la maquina durante el proceso',' Ruteando ','ok');
     set(handlesmadre.pausa_machine,'enable','on');%habilita boton pausar
     set(handlesmadre.stop_machine,'enable','on');%habilita boton detener
    [my_out_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, out_pipe,
uint8(0), uint8 (0)); % Se abre el tunel de envio
    [my_in_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, in_pipe, uint8
(1), uint8 (0)); % Se abre el tunel de recepción
```

```
    %utilizar save 'archivo.mat' variables '-append'; para agregar o actualizar
    %datos en el archivo sin borrar los ya contenidos.
    %armamos la trama a enviar por USB
    data_out(1)=cabeza;
    % si la cabeza es igual a 202 (envio de posiciones de ruteo)
    if data_out(1)==4

        data_in = zeros(1,64,'uint8');%colocamos el data_in en cero en cada
        data_in(1)=1;
        int_x=1;
        int_y=1;
        int_z=1;

        while hObject==handles.radiobutton_on_manual && conectado~=0% && int_x==1 &&
int_y==1 && int_z==1
            load leer_hobject.mat;
            load leer_hobject.mat;
            operacion='c';
            script_encoder;

            [conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm)
            if conectado==0
                msgbox('Se ha perdido la conexion USB, reconecte y pulse el boton RUTEAR
para continuar.',' Proceso detenido.');
                %apagar efector final y subir dejando en ultimalinea, al
                %continuar regresar a ultima linea
            else

                if (int_x==1 && int_y==1 && int_z==1)
                    pause(0.050);
                    data_out(5)=tabla_encoder_uint8(1,1); %XH
                    data_out(6)=tabla_encoder_uint8(1,2); %XL
                    data_out(25)=tabla_encoder_uint8(1,3); %YH
                    data_out(26)=tabla_encoder_uint8(1,4); %YL
                    data_out(45)=tabla_encoder_uint8(1,5); %ZH
                    data_out(46)=tabla_encoder_uint8(1,6); %ZL
                    data_out(63)=tabla_encoder_uint8(1,7); %EF
                    data_out(62)=3; %EF
                    data_out(63);
                    data_out(5:6);
                    calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(64),
uint8(64), uint8(10));
                    [aa,bb,data_in,dd]=calllib('libreria', 'MPUSBRead',my_in_pipe, data_in,
uint8(64), uint8(64), uint8(10)); % Se recibe el dato que envia el PIC
                    int_x=data_in(3)
                    int_y=data_in(6)
                    int_z=data_in(9)
                    data_out;
                    data_in(1:10);
                    set(handles.text_positionX,'string',num2str((config_ejes.x.recorrido)-
bin2dec([dec2bin(data_in(1),8),dec2bin(data_in(2),8)])*config_ejes.x.paso/config_ejes.x.enc_
supr_bits));

set(handles.text_positionY,'string',num2str(bin2dec([dec2bin(data_in(4),8),dec2bin(data_in(5
),8)])*config_ejes.y.paso/config_ejes.y.enc_supr_bits));
                    set(handles.text_positionZ,'string',num2str((config_ejes.z.recorrido)-
bin2dec([dec2bin(data_in(7),8),dec2bin(data_in(8),8)])*config_ejes.z.paso/config_ejes.z.enc_
supr_bits));
                    if bin2dec([dec2bin(data_in(1),8),dec2bin(data_in(2),8)])>60000
                        data_in(1)=255-data_in(1);
                        data_in(2)=255-data_in(2);
                    end
                    if bin2dec([dec2bin(data_in(4),8),dec2bin(data_in(5),8)])>60000
                        data_in(4)=255-data_in(4);
                        data_in(5)=255-data_in(5);
                    end
                    if bin2dec([dec2bin(data_in(7),8),dec2bin(data_in(8),8)])>60000
                        data_in(7)=255-data_in(7);
                        data_in(8)=255-data_in(8);
```

```
                    end
                    error_cnc(1)=abs(bin2dec([dec2bin(data_in(1),8),dec2bin(data_in(2),8)])-
bin2dec([dec2bin(tabla_encoder_uint8(1,1),8),dec2bin(tabla_encoder_uint8(1,2),8)]));
                    error_cnc(2)=abs(bin2dec([dec2bin(data_in(4),8),dec2bin(data_in(5),8)])-
bin2dec([dec2bin(tabla_encoder_uint8(1,3),8),dec2bin(tabla_encoder_uint8(1,4),8)]));
                    error_cnc(3)=abs(bin2dec([dec2bin(data_in(7),8),dec2bin(data_in(8),8)])-
bin2dec([dec2bin(tabla_encoder_uint8(1,5),8),dec2bin(tabla_encoder_uint8(1,6),8)]));
                    error_cnc
                else
                    int_x
                    int_y
                    int_z
                    if int_x==255 || int_x==0
                        msgbox('El eje X se ha detenido, por favor reinicie la maquina.','
Proceso detenido.');
                        stop=1;
                        script_stop;
                        hObject=handles.radiobutton_off_manual;
                    elseif int_y==255 || int_y==0
                        msgbox('El eje Y se ha detenido, por favor reinicie la maquina.','
Proceso detenido.');
                        stop=1;
                        script_stop;
                        hObject=handles.radiobutton_off_manual;
                    elseif int_z==255 || int_z==0
                        msgbox('El eje Z se ha detenido, por favor reinicie la maquina.','
Proceso detenido.');
                        stop=1;
                        script_stop;
                        hObject=handles.radiobutton_off_manual;
                    end
                end
            end
        end
    end

    data_out(1)=0;
    calllib('libreria', 'MPUSBClose', my_in_pipe);  % Se cierra el tunel de recepción
    calllib('libreria', 'MPUSBClose', my_out_pipe); % Se cierra el tunel de envio
else
    set(handlesmadre.pausa_machine,'enable','off');%deshabilita boton pausar
    set(handlesmadre.stop_machine,'enable','off');%deshabilita boton detener
    msgbox('Por favor conecte la maquina y pulse nuevamente RUTEAR para continuar el
proceso',' Dispositivo desconectado ');
end
```

Valid_edit_par.m:

```
valor=str2double(get(hObject,'string'));
if valor < 1
    warndlg('Introduzca solo numeros con valores mayores a 0.');
    set(hObject,'string','1');
end
```

ANEXO 5. Programa implementado en el PIC Maestro para el control.

C:\MASTER CCS PARA CONTROL FINAL\MASTER CCS para control final.c

```
//Declaracion de librerias, FUSES de programacion y variables de programa
#include <MASTER_VARIOS.h>
#define USB_EP1_TX_SIZE    64 //size to allocate for the tx endpoint 1 buff
#define USB_EP1_RX_SIZE    64 //size to allocate for the rx endpoint 1 buff

//Declaracion de librerias
#include <pic18_usb.h>
#include <usb_desc_scope_CNC.h>
#include <usb.c>

//Declaracion de registros
#define  EN_Z    PIN_B5
#define  EN_Y    PIN_B6
#define  EN_X    PIN_B7
#define  LED_ON  output_high
#define  LED_OFF output_low

//Configuracion de modulos
#use fast_io(B)
#use fast_io(D)
#use i2c(Master,force_hw,sda=PIN_B0,scl=PIN_B1,fast=400000)

//Declaracion de funciones
void write(BYTE,BYTE);
BYTE read(BYTE);

//Declaracion de variables
int8 dato_in_CNC[64];
int8 dato_out_CNC[64];
int8 i=0;
int8 datos[64];
int8 datos_z[60];
int8 datos_slave[10]={0,0,0,0,0,0,0,0,0,0};

//Funcion principal
void main()
{
   //Inicializamos los arreglos de datos
   for(i=0;i<64;i++)
   {
      datos[i]=0;
      dato_in_CNC[i]=0;
      dato_out_CNC[i]=0;
   }
      for(i=0;i<60;i++)
   {
      datos_z[i]=0;
   }
   usb_init();        // inicializamos el USB
   usb_task();        // habilita periferico usb e interrupciones
   usb_wait_for_enumeration(); // esperamos hasta que el PicUSB sea configu
//Se configuran los puertos y modulos
   SET_TRIS_B(0xff);
   SET_TRIS_D(0x00);
   OUTPUT_D(0x00);
   delay_ms(250);
   do
   {
      if(usb_enumerated())    // si el Pic está configurado via USB
      {
         if (usb_kbhit(1))    // si el endpoint de salida contiene datos del
         {
            dato_in_CNC[0]=0;
            usb_get_packet(1, dato_in_CNC, 64);
            for(i=0;i<64;i++)
            {
               datos[i]=dato_in_CNC[i];
            }
```

```
            if(datos[0]==4)
            {
               output_bit(PIN_D0,datos[62]);
               write(4,0xB0);// Para X
               read(0xB1);
               dato_out_CNC[0]=datos_slave[0];
               dato_out_CNC[1]=datos_slave[1];
               dato_out_CNC[2]=datos_slave[2];
               write(24,0xC0);// Para Y
               read(0xC1);
               dato_out_CNC[3]=datos_slave[0];
               dato_out_CNC[4]=datos_slave[1];
               dato_out_CNC[5]=datos_slave[2];
               write(44,0xA0);// Para Z
               read(0xA1);
               dato_out_CNC[6]=datos_slave[0];
               dato_out_CNC[7]=datos_slave[1];
               dato_out_CNC[8]=datos_slave[2];
               usb_put_packet(1, dato_out_CNC, 64, USB_DTS_TOGGLE);
            }
         }
      }
   }while(true);
 }

//Subrutina para leer la trama de datos del i2c
BYTE read(BYTE dir)
{
   BYTE dato_in;
   i2c_start();         //Inicializa la transmisión
   i2c_write(dir);      //Direccion del esclavo a comunicar
   for(i=0;i<2;i++)
   {
      datos_slave[i]=i2c_read(1); //Byte recibidos
   }
   datos_slave[2]=i2c_read(0);
   i2c_stop();
   return dato_in;
}

//Subrutina para enviar la trama de datos al esclavo por el i2c
void write(BYTE inieje,BYTE dir)
{
   i2c_start();          //Inicializa la transmisión
   i2c_write(dir);       //Direccion del esclavo a comunicar
   i2c_write(datos[0]);  // Numero de Bytes a enviar
   for(i=inieje; i<datos[0]+inieje; i++)
   {
      i2c_write(datos[i]);   //Dato a escribir;
   }
   i2c_stop();
   return;
}
```

ANEXO 6. Programa implementado en cada uno de los PIC´s Esclavo para el control.

```
//Declaracion de librerias
#include <18F4431.h>
#DEVICE HIGH_INTS=TRUE //Habilita interrupcion de alta prioridad
#include <stdlib.h>
#include <string.h>
#include <math.h>

//Declaracion de registros
#byte QEICON = 0xFB6
#byte POSCNTH = 0xF67
#byte POSCNTL = 0xF66
#byte MAXCNTH = 0xF65
#byte MAXCNTL = 0xF64
#byte DFLTCON = 0xF60
#byte VELRH = 0xF69
#byte VELRL = 0xF68
#byte T5CON = 0xB7
#byte TRISB = 0xF93

//Declaracion de los FUSES de programacion
#FUSES NOWDT       //No Watch Dog Timer
#FUSES WDT128      //Watch Dog Timer uses 1:128 Postscale
#FUSES HS          //Crystal osc <= 4mhz for PCM/PCH , 3mhz to 10 mhz for PC
#FUSES MCLR
#FUSES NOCPB
#FUSES NOPROTECT
#FUSES NOCPD
#FUSES NOBROWNOUT
#FUSES NOLVP
#FUSES NOIESO
#FUSES NOPUT
#FUSES NOWRT
#FUSES NOEBTR
#FUSES NOPWMPIN
#FUSES SSP_RC //#FUSES SSP_RC para i2c en puerto C
#FUSES NODEBUG
#FUSES NOFCMEN
#FUSES HPOL_LOW
#FUSES LPOL_LOW

//Configuracion de modulos
#use delay(clock=20000000)
#use fast_io(B)
#use i2c(slave,fast=400000,sda=PIN_C4,scl=PIN_C5,address=0xB0,FORCE_HW)
#use rtos(timer=0,minor_cycle=5ms)//para el tiempo de muestreo

//Declaracion de funciones
void leer(void);
void CONFIGPCPWM(void);
void configencoder(void);
void write(void);
void pid(void);
void calibracion(void);

//Declaracion de variables
float ukpos_sat=0;
int8 final_ca=1;
int8 interrupcion=1;
int8 enc_cero=0;
int8 sentido_giro=0;
int8 curvah=0;
int8 curval=0;
int8 curva[3];
int8 i=0;
int8 state;
float cpc=0;
int8 k=0;
float enco32=0;
```

```
float enco32_1=0;
float con_i_1=0;
float con_i=0;
float con_p=0;
float kc=0;
float kc_pos=0;
float SP_pos=0;
float enco_1=0;
float enco=0;
float icon=0;
float vel_1=0;
float dif_vel=0;
float dif_pos=0;
float ukpos=0;
float uk=0;
float saturacion=0;
int16 uk16=0;
int16 offset=3400;// offset de 20.7% ciclo de trabajo
int16 enco32_16=0;
int8 trama_control[25]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

//Declaracion de tareas del RTOS
#task(rate=10ms,max=5ms)                //Tarea del PID
void pid();

//Interrupcion del puerto B, se utiliza para sensar los finales de carrera
#INT_RB  HIGH
void fin_carrera()
{
//PIN_B6 sensor de alante y PIN_B7 sensor de atras
int8 temppuerto=0;
   temppuerto=input_b();
   if(input(PIN_B6)==0 && input(PIN_B7)==1)
   {
      if (final_ca==0){
         interrupcion=0;
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(0);
         while(1){}
      }
   }
   else if(input(PIN_B6)==1 && input(PIN_B7)==0){}
   else if(input(PIN_B6)==0 && input(PIN_B7)==0){}
   else if(input(PIN_B6)==1 && input(PIN_B7)==1){}
   output_b(input_b());
}

//Interrupcion I2C
#INT_SSP
void ssp_interupt ()
{
   state = i2c_isr_state();
   if(state < 0x80)     //Maestro esta enviando datos
   {
      leer();//Se ejecuta la funcion leer
   }
   else if(state == 0x80) //Maestro pide un dato
   {
      write();//Se ejecuta la funcion escribir
   }
}

//Funcion principal
void main()
{
//Se configuracion los puertos y modulos
   SET_TRIS_B(0xC0);
   setup_adc_ports(NO_ANALOGS);
```

```
   setup_adc(ADC_OFF);
   clear_interrupt(int_rb);
   enable_interrupts(GLOBAL);
   enable_interrupts(INT_SSP);
   enable_interrupts(INT_RB);
   disable_interrupts(INT_IC2QEI);
   CONFIGPCPWM();
   configencoder();
   POSCNTH=0;
   POSCNTL=0;
   //Constante control
   kc_pos=0.00165;
   kc=40.29;
   icon=3.738;
   con_i_1=0;
   calibracion();//Se ejecuta la rutina de calibracion
   rtos_run();//Inicia el RTOS
}

//Funcion que se ejecuta dentro del RTOS y es donde se ejecuta todo el cont
void pid(void)
{
   if(trama_control[1]==4)//se recibe posicion
   {
      SP_pos=make16(trama_control[2],trama_control[3]);
      if(enc_cero==0){
         POSCNTH=0;
         POSCNTL=0;
         enc_cero=1;
      }
      enco=make16(POSCNTH,POSCNTL);
      if(enco_1>=65000 && enco<=500){
         cpc=cpc+1;
      }else if(enco_1<=500 && enco>65000){
         cpc=cpc-1;
      }
      enco32=((65535*(cpc)+enco)/4);
      dif_pos=SP_pos-enco32; //Calculo de error de posicion
      if (abs(dif_pos)<=2){
         dif_pos=0;
      }
      ukpos=dif_pos*kc_pos; //Calculo de control de posicion
      vel_1=(enco32-enco32_1)/10; //Calculo de la velocidad real
      enco32_1=enco32;
      enco_1=enco;
      //Limites del SetPoint de velocidad
      if (ukpos>1.44){
         ukpos_sat=1.44;
      }else if(ukpos<-1.44){
         ukpos_sat=-1.44;
      }else{
         ukpos_sat=ukpos;
      }
      dif_vel=ukpos_sat-vel_1; //Calculo de error de velocidad
      //Calculo del control de velocidad
      con_p=dif_vel*kc;
      con_i=con_i_1+icon*dif_vel-saturacion*1.1;
      con_i_1=con_i;
      uk=(con_p+con_i)*163.83; //Salida del control de velocidad
      if (uk > 16383){
         uk16=16383;
         sentido_giro=1;
         saturacion=uk-uk16;
      }
      else if (uk<-16383){
         uk16=16383;
         sentido_giro=2;
         saturacion=uk+uk16;
```

```
      }
      else if (uk <0){
         uk=abs(uk)+offset;
         if (uk>16383){
            uk16=16383;
         }
         else{
            uk16=uk;
         }
         sentido_giro=2;
         saturacion=0;
      }
      else{
         uk=uk+offset;
         if (uk>16383){
            uk16=16383;
         }
         else{
            uk16=uk;
         }
         sentido_giro=1;
         saturacion=0;
      }
      saturacion=saturacion/163.83;
      if (abs(dif_pos)<2){
         sentido_giro=3;
      }
      //Se asigna valor en registro del PWM segun el sentido de giro
      if (sentido_giro==1){
         set_power_pwm0_duty(uk16);
         set_power_pwm2_duty(0);
         }
      else if(sentido_giro==2){
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(uk16);
         }
      else if(sentido_giro==3){
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(0);
         }
      enco32_16=enco32;
      //Se almacena la posicion en las variables curvah y curval para luego
      //enviarlas por i2c
      curvah=enco32_16>>8;
      curval=enco32_16;
   }
}

//Subrutina para leer la trama de datos del i2c
void leer(void)
{
   //Direccion
   while(!i2c_poll());
   trama_control[0]=i2c_read();
   //Numero de Byte a leer
   while(!i2c_poll());
   trama_control[1]=i2c_read();
   for(k=2; k<trama_control[1]+2; k++){
      while(!i2c_poll());
      trama_control[k]=i2c_read();
   }
   return;
}

//Subrutina para enviar la trama de datos al maestro por i2c
void write(void)
{
   curva[0]=curvah;
```

```
   curva[1]=curval;
   curva[2]=interrupcion;
   for (i=0;i<3;i++){
      i2c_write(curva[i]);
   }
   return;
}

//Subrutina para la configuracion del modulo PWM
void CONFIGPCPWM(void)
{
   setup_power_pwm_pins(PWM_COMPLEMENTARY,PWM_COMPLEMENTARY,PWM_OFF,PWM_OFF
   setup_power_pwm(PWM_CLOCK_DIV_4|PWM_FREE_RUN,1,0,4095,0,1,0);
   return;
}

//Subrutina para la configuracion del modulo encoder
void configencoder(void)
{
   QEICON  = 0b10111000;//Configuracion del encoder y sentido
   POSCNTH = 0;//Posicion del eje
   POSCNTL = 0;//Posicion del eje
   MAXCNTH = 0xFF;//Maixmo Valor del contador
   MAXCNTL = 0xFF;//Maixmo Valor del contador
   DFLTCON = 0b00111000;//Activar Filtro
   return;
}

//Subrutina para la calibracion
void calibracion(void)
{
//PIN_B6 sensor de alante y PIN_B7 sensor de atras
//movimiento hacia la izquierda
   if (input(PIN_B6)==1 && input(PIN_B7)==0){//si esta en sensor derecho
      while(input(PIN_B7)==0)
      {  //se mueve hacia la izquierda
         set_power_pwm0_duty(8000);
         set_power_pwm2_duty(0);
      }
      delay_ms(2000);
      //se frena el motor
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
      //se mueve hacia la izquierda lento
      while(input(PIN_B7)==1){
         set_power_pwm2_duty(6000);
         set_power_pwm0_duty(0);
      }
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
   } else if (input(PIN_B6)==0 && input(PIN_B7)==1){//si esta en sensor der
      while(input(PIN_B7)==1)
      {  //se mueve hacia izq LENTO hasta llegar
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(8000);
      }
      set_power_pwm0_duty(0);
      set_power_pwm2_duty(0);
      while(input(PIN_B7)==0)
      {  //se mueve hacia der lento
         set_power_pwm2_duty(0);
         set_power_pwm0_duty(6000);
      }
      delay_ms(2000);
      set_power_pwm0_duty(0);
      set_power_pwm2_duty(0);
      while(input(PIN_B7)==1)
      {  //se mueve hacia izq mas LENTO hasta llegar
```

```
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(5000);
      }
      set_power_pwm0_duty(0);
      set_power_pwm2_duty(0);
   } else if(input(PIN_B6)==1 && input(PIN_B7)==1){//si esta en medio
      while(input(PIN_B7)==1)
      {  //se mueve hacia izq LENTO hasta llegar
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(8000);
      }
      set_power_pwm0_duty(0);
      set_power_pwm2_duty(0);
      while(input(PIN_B7)==0)
      {  //se mueve hacia der lento
         set_power_pwm2_duty(0);
         set_power_pwm0_duty(6000);
      }
      delay_ms(2000);
      set_power_pwm0_duty(0);
      set_power_pwm2_duty(0);
      while(input(PIN_B7)==1)
      {  //se mueve hacia izq mas LENTO hasta llegar
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(5000);
      }
      set_power_pwm0_duty(0);
      set_power_pwm2_duty(0);
   }
   set_power_pwm2_duty(0);
   set_power_pwm0_duty(0);
   //Se inicializa el registro del encoder
   POSCNTH=0;
   POSCNTL=0;
   final_ca=0;
}
```

```
//Declaracion de librerias
#include <18F4431.h>
#DEVICE HIGH_INTS=TRUE //Habilita interrupcion de alta prioridad
#include <stdlib.h>
#include <string.h>
#include <math.h>

//Declaracion de registros
#byte QEICON = 0xFB6
#byte POSCNTH = 0xF67
#byte POSCNTL = 0xF66
#byte MAXCNTH = 0xF65
#byte MAXCNTL = 0xF64
#byte DFLTCON = 0xF60
#byte VELRH = 0xF69
#byte VELRL = 0xF68
#byte T5CON = 0xB7
#byte TRISB = 0xF93

//Declaracion de los FUSES de programacion
#FUSES NOWDT       //No Watch Dog Timer
#FUSES WDT128      //Watch Dog Timer uses 1:128 Postscale
#FUSES HS          //Crystal osc <= 4mhz for PCM/PCH , 3mhz to 10 mhz for PC
#FUSES MCLR
#FUSES NOCPB
#FUSES NOPROTECT
#FUSES NOCPD
#FUSES NOBROWNOUT
#FUSES NOLVP
#FUSES NOIESO
#FUSES NOPUT
#FUSES NOWRT
#FUSES NOEBTR
#FUSES NOPWMPIN
#FUSES SSP_RC //#FUSES SSP_RC para i2c en puerto C
#FUSES NODEBUG
#FUSES NOFCMEN
#FUSES HPOL_LOW
#FUSES LPOL_LOW

//Configuracion de modulos
#use delay(clock=20000000)
#use fast_io(B)
#use i2c(slave,fast=400000,sda=PIN_C4,scl=PIN_C5,address=0xC0,FORCE_HW)
#use rtos(timer=0,minor_cycle=5ms)//para el tiempo de muestreo

//Declaracion de funciones
void leer(void);
void CONFIGPCPWM(void);
void configencoder(void);
void write(void);
void pid(void);
void calibracion(void);

//Declaracion de variables
float ukpos_sat=0;
int8 final_ca=1;
int8 interrupcion=1;
int8 enc_cero=0;
int8 sentido_giro=0;
int8 curvah=0;
int8 curval=0;
int8 curva[3];
int8 i=0;
int8 state;
float cpc=0;
int8 k=0;
float enco32=0;
```

```
float enco32_1=0;
float con_i_1=0;
float con_i=0;
float con_p=0;
float kc=0;
float kc_pos=0;
float SP_pos=0;
float enco_1=0;
float enco=0;
float icon=0;
float vel_1=0;
float dif_vel=0;
float dif_pos=0;
float ukpos=0;
float uk=0;
float saturacion=0;
int16 uk16=0;
int16 offset=5100; // offset de 31% ciclo de trabajo
int16 enco32_16=0;
int8 trama_control[25]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

//Declaracion de tareas del RTOS
#task(rate=10ms,max=5ms)                //Tarea del PID
void pid();

//Interrupcion del puerto B, se utiliza para sensar los finales de carrera
#INT_RB  HIGH
void fin_carrera()
{
//PIN_B6 sensor de alante y PIN_B7 sensor de atras
int8 temppuerto=0;
   temppuerto=input_b();
   if(input(PIN_B6)==0 && input(PIN_B7)==1)
   {
      if (final_ca==0){
         interrupcion=0;
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(0);
         while(1){}
      }
   }
   else if(input(PIN_B6)==1 && input(PIN_B7)==0){}
   else if(input(PIN_B6)==0 && input(PIN_B7)==0){}
   else if(input(PIN_B6)==1 && input(PIN_B7)==1){}
   output_b(input_b());
}

//Interrupcion I2C
#INT_SSP
void ssp_interupt ()
{
   state = i2c_isr_state();
   if(state < 0x80)     //Maestro esta enviando datos
   {
      leer();//Se ejecuta la funcion leer
   }
   else if(state == 0x80) //Maestro pide un dato
   {
      write();//Se ejecuta la funcion escribir
   }
}

//Funcion principal
void main()
{
//Se configuracion los puertos y modulos
   SET_TRIS_B(0xC0);
   setup_adc_ports(NO_ANALOGS);
```

```
   setup_adc(ADC_OFF);
   clear_interrupt(int_rb);
   enable_interrupts(GLOBAL);
   enable_interrupts(INT_SSP);
   enable_interrupts(INT_RB);
   disable_interrupts(INT_IC2QEI);
   CONFIGPCPWM();
   configencoder();
   POSCNTH=0;
   POSCNTL=0;
   //Constantes de control
   kc_pos=0.00165;
   kc=51.4668;
   icon=6.598;
   con_i_1=0;
   calibracion();//Se ejecuta la rutina de calibracion
   rtos_run();//Inicia el RTOS
}

//Funcion que se ejecuta dentro del RTOS y es donde se ejecuta todo el cont
void pid(void)
{
   if(trama_control[1]==4)//se recibe posicion
   {
      SP_pos=make16(trama_control[2],trama_control[3]);
      if(enc_cero==0){
         POSCNTH=0;
         POSCNTL=0;
         enc_cero=1;
      }
      enco=make16(POSCNTH,POSCNTL);
      if(enco_1>=65000 && enco<=500){
         cpc=cpc+1;
      }else if(enco_1<=500 && enco>65000){
         cpc=cpc-1;
      }
      enco32=((65535*(cpc)+enco)/4);
      dif_pos=SP_pos-enco32; //Calculo de error de posicion
      if (abs(dif_pos)<=2){
         dif_pos=0;
      }
      ukpos=dif_pos*kc_pos; //Calculo de control de posicion
      vel_1=(enco32-enco32_1)/10; //Calculo de la velocidad real
      enco32_1=enco32;
      enco_1=enco;
      //Limites del SetPoint de velocidad
      if (ukpos>1.44){
         ukpos_sat=1.44;
      }else if(ukpos<-1.44){
         ukpos_sat=-1.44;
      }else{
         ukpos_sat=ukpos;
      }
      dif_vel=ukpos_sat-vel_1; //Calculo de error de velocidad
      //Calculo del control de velocidad
      con_p=dif_vel*kc;
      con_i=con_i_1+icon*dif_vel-saturacion*1.1;
      con_i_1=con_i;
      uk=(con_p+con_i)*163.83; //Salida del control de velocidad
      if (uk > 16383){
         uk16=16383;
         sentido_giro=1;
         saturacion=uk-uk16;
      }
      else if (uk<-16383){
         uk16=16383;
         sentido_giro=2;
         saturacion=uk+uk16;
```

```
      }
      else if (uk <0){
         uk=abs(uk)+offset;
         if (uk>16383){
            uk16=16383;
         }
         else{
            uk16=uk;
         }
         sentido_giro=2;
         saturacion=0;
      }
      else{
         uk=uk+offset;
         if (uk>16383){
            uk16=16383;
         }
         else{
            uk16=uk;
         }
         sentido_giro=1;
         saturacion=0;
      }
      saturacion=saturacion/163.83;
      if (abs(dif_pos)<2){
         sentido_giro=3;
      }
      //Se asigna valor en registro del PWM segun el sentido de giro
      if (sentido_giro==1){
         set_power_pwm0_duty(uk16);
         set_power_pwm2_duty(0);
         }
      else if(sentido_giro==2){
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(uk16);
         }
      else if(sentido_giro==3){
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(0);
         }
      enco32_16=enco32;
      //Se almacena la posicion en las variables curvah y curval para luego
      //enviarlas por i2c
      curvah=enco32_16>>8;
      curval=enco32_16;
   }
}

//Subrutina para leer la trama de datos del i2c
void leer(void)
{
   //Direccion
   while(!i2c_poll());
   trama_control[0]=i2c_read();
   //Numero de Byte a leer
   while(!i2c_poll());
   trama_control[1]=i2c_read();
   for(k=2; k<trama_control[1]+2; k++){
      while(!i2c_poll());
      trama_control[k]=i2c_read();
   }
   return;
}

//Subrutina para enviar la trama de datos al maestro por i2c
void write(void)
{
   curva[0]=curvah;
```

```
   curva[1]=curval;
   curva[2]=interrupcion;
   for (i=0;i<3;i++){
      i2c_write(curva[i]);
   }
   return;
}

//Subrutina para la configuracion del modulo PWM
void CONFIGPCPWM(void)
{
   setup_power_pwm_pins(PWM_COMPLEMENTARY,PWM_COMPLEMENTARY,PWM_OFF,PWM_OFF
   setup_power_pwm(PWM_CLOCK_DIV_4|PWM_FREE_RUN,1,0,4095,0,1,0);
   return;
}

//Subrutina para la configuracion del modulo encoder
void configencoder(void)
{
   QEICON  = 0b10111000;//Configuracion del encoder y sentido
   POSCNTH = 0;//Posicion del eje
   POSCNTL = 0;//Posicion del eje
   MAXCNTH = 0xFF;//Maixmo Valor del contador
   MAXCNTL = 0xFF;//Maixmo Valor del contador
   DFLTCON = 0b00111000;//Activar Filtro
   return;
}

//Subrutina para la calibracion
void calibracion(void)
{
//PIN_B6 sensor de alante y PIN_B7 sensor de atras
//movimiento hacia adelante
   if (input(PIN_B6)==0 && input(PIN_B7)==1){//si esta en sensor de adelant
      while(input(PIN_B7)==1)
      {  //se mueve hacia atras
         set_power_pwm2_duty(8000);
         set_power_pwm0_duty(0);
      }
      //se frena el motor
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
      //se mueve hacia alante lento
      while(input(PIN_B7)==0)
      {  //se mueve hacia alante
         set_power_pwm0_duty(6000);
         set_power_pwm2_duty(0);
      }
      delay_ms(2000);
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
      while(input(PIN_B7)==1)
      {  //se mueve hacia atras mas lento
         set_power_pwm2_duty(5000);
         set_power_pwm0_duty(0);
      }
   } else if (input(PIN_B6)==1 && input(PIN_B7)==0){//si esta atras
      while(input(PIN_B7)==0)
      {  //se mueve hacia alante LENTO
         set_power_pwm2_duty(0);
         set_power_pwm0_duty(6000);
      }
      delay_ms(2000);
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
      while(input(PIN_B7)==1)
      {  //se mueve hacia ATRAS
         set_power_pwm0_duty(0);
```

```
         set_power_pwm2_duty(5000);
      }
   } else if(input(PIN_B6)==1 && input(PIN_B7)==1){//si esta en medio
      while(input(PIN_B7)==1)
      {  //se mueve hacia atras
         set_power_pwm2_duty(8000);
         set_power_pwm0_duty(0);
      }
      //se frena el motor
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
      //se mueve hacia alante lento
      while(input(PIN_B7)==0)
      {  //se mueve hacia alante
         set_power_pwm0_duty(6000);
         set_power_pwm2_duty(0);
      }
      delay_ms(2000);
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
      while(input(PIN_B7)==1)
      {  //se mueve hacia atras mas lento
         set_power_pwm2_duty(5000);
         set_power_pwm0_duty(0);
      }
   }
   set_power_pwm2_duty(0);
   set_power_pwm0_duty(0);
   //Se inicializa el registro del encoder
   POSCNTH=0;
   POSCNTL=0;
}
```

```
//Declaracion de librerias
#include <18F4431.h>
#DEVICE HIGH_INTS=TRUE //Habilita interrupcion de alta prioridad
#include <stdlib.h>
#include <string.h>
#include <math.h>

//Declaracion de registros
#byte QEICON = 0xFB6
#byte POSCNTH = 0xF67
#byte POSCNTL = 0xF66
#byte MAXCNTH = 0xF65
#byte MAXCNTL = 0xF64
#byte DFLTCON = 0xF60
#byte VELRH = 0xF69
#byte VELRL = 0xF68
#byte T5CON = 0xB7
#byte TRISB = 0xF93

//Declaracion de los FUSES de programacion
#FUSES NOWDT       //No Watch Dog Timer
#FUSES WDT128      //Watch Dog Timer uses 1:128 Postscale
#FUSES HS          //Crystal osc <= 4mhz for PCM/PCH , 3mhz to 10 mhz for PC
#FUSES MCLR
#FUSES NOCPB
#FUSES NOPROTECT
#FUSES NOCPD
#FUSES NOBROWNOUT
#FUSES NOLVP
#FUSES NOIESO
#FUSES NOPUT
#FUSES NOWRT
#FUSES NOEBTR
#FUSES NOPWMPIN
#FUSES SSP_RC //#FUSES SSP_RC para i2c en puerto C
#FUSES NODEBUG
#FUSES NOFCMEN
#FUSES HPOL_LOW
#FUSES LPOL_LOW

//Configuracion de modulos
#use delay(clock=20000000)
#use fast_io(B)
#use i2c(slave,fast=400000,sda=PIN_C4,scl=PIN_C5,address=0xA0,FORCE_HW)
#use rtos(timer=0,minor_cycle=5ms)//para el tiempo de muestreo

//Declaracion de funciones
void leer(void);
void CONFIGPCPWM(void);
void configencoder(void);
void write(void);
void pid(void);
void calibracion(void);

//Declaracion de variables
float ukpos_sat=0;
int8 final_ca=1;
int8 interrupcion=1;
int8 enc_cero=0;
int8 sentido_giro=0;
int8 curvah=0;
int8 curval=0;
int8 curva[3]={0,0,0};
int8 i=0;
int8 state;
float cpc=0;
int8 k=0;
float enco32=0;
```

```
float enco32_1=0;
float con_i_1=0;
float con_i=0;
float con_p=0;
float kc=0;
float kc_pos=0;
float SP_pos=0;
float SP_pos_1=0;
float enco_1=0;
float enco=0;
float icon=0;
float vel_1=0;
float dif_vel=0;
float dif_pos=0;
float ukpos=0;
float uk=0;
float saturacion=0;
int16 uk16=0;
int16 enco32_16=0;
int8 trama_control[25]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

//Declaracion de tareas del RTOS
#task(rate=10ms,max=5ms)                //Tarea del PID
void pid();

//Interrupcion del puerto B, se utiliza para sensar los finales de carrera
#INT_RB  HIGH
void fin_carrera()
{
//PIN_B6 sensor de alante y PIN_B7 sensor de atras
int8 temppuerto=0;
   temppuerto=input_b();
   if(input(PIN_B6)==0 && input(PIN_B7)==1)
   {
      if (final_ca==0){
         interrupcion=0;
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(0);
         while(1){}
      }
   }
   else if(input(PIN_B6)==1 && input(PIN_B7)==0){}
   else if(input(PIN_B6)==0 && input(PIN_B7)==0){}
   else if(input(PIN_B6)==1 && input(PIN_B7)==1){}
   output_b(input_b());
}

//Interrupcion I2C
#INT_SSP
void ssp_interupt ()
{
   state = i2c_isr_state();
   if(state < 0x80)     //Maestro esta enviando datos
   {
      leer();//Se ejecuta la funcion leer
   }
   else if(state == 0x80) //Maestro pide un dato
   {
      write();//Se ejecuta la funcion escribir
   }
}

//Funcion principal
void main()
{
//Se configuracion los puertos y modulos
   SET_TRIS_B(0xC0);
   setup_adc_ports(NO_ANALOGS);
```

```
   setup_adc(ADC_OFF);
   clear_interrupt(int_rb);
   enable_interrupts(GLOBAL);
   enable_interrupts(INT_SSP);
   enable_interrupts(INT_RB);
   disable_interrupts(INT_IC2QEI);
   CONFIGPCPWM();
   configencoder();
   POSCNTH=0;
   POSCNTL=0;
   //Constante control
   kc_pos=0.0015;
   kc=43.1283;
   icon=4.028;
   con_i_1=0;
   calibracion();//Se ejecuta la rutina de calibracion
   rtos_run();//Inicia el RTOS
}

//Funcion que se ejecuta dentro del RTOS y es donde se ejecuta todo el cont
void pid(void)
{
   if(trama_control[1]==4)//se recibe posicion
   {
      SP_pos=make16(trama_control[2],trama_control[3]);
      if(enc_cero==0){
         POSCNTH=0;
         POSCNTL=0;
         enc_cero=1;
      }
      enco=make16(POSCNTH,POSCNTL);
      enco32=(enco)/4;
      dif_pos=SP_pos-enco32; //Calculo de error de posicion
      if (abs(dif_pos)<=2){
         dif_pos=0;
      }
      ukpos=dif_pos*kc_pos; //Calculo de control de posicion
      vel_1=(enco32-enco32_1)/10; //Calculo de la velocidad real
      enco32_1=enco32;
      enco_1=enco;
      //Limites del SetPoint de velocidad
      if (ukpos>1.16){
         ukpos_sat=1.16;
      }else if(ukpos<-1.624){
         ukpos_sat=-1.624;
      }else{
         ukpos_sat=ukpos;
      }
      dif_vel=ukpos_sat-vel_1; //Calculo de error de velocidad
      //Calculo del control de velocidad
      con_p=dif_vel*kc;
      con_i=con_i_1+icon*dif_vel-saturacion*1.1;
      con_i_1=con_i;
      uk=(con_p+con_i)*163.83; //Salida del control de velocidad
      if (uk > 16383){
         uk16=16383;
         sentido_giro=1;
         saturacion=uk-uk16;
      }
      else if (uk<-16383){
         uk16=16383;
         sentido_giro=2;
         saturacion=uk+uk16;
      }
      else if (uk <0){
         uk=abs(uk)+4095;// offset de 25% ciclo de trabajo
         if (uk>16383){
            uk16=16383;
```

```
            }
            else{
               uk16=uk;
            }
            sentido_giro=2;
            saturacion=0;
         }
         else{
            uk=uk+2950; //offset de 18% ciclo de trabajo
            if (uk>16383){
               uk16=16383;
            }
            else{
               uk16=uk;
            }
            sentido_giro=1;
            saturacion=0;
         }
         saturacion=saturacion/163.83;
         if (abs(dif_pos)<2){
            sentido_giro=3;
         }
         //Se asigna valor en registro del PWM segun el sentido de giro
         if (sentido_giro==1){
            set_power_pwm0_duty(uk16);
            set_power_pwm2_duty(0);
            }
         else if(sentido_giro==2){
            set_power_pwm0_duty(0);
            set_power_pwm2_duty(uk16);
            }
         else if(sentido_giro==3){
            set_power_pwm0_duty(0);
            set_power_pwm2_duty(0);
            }
         enco32_16=enco32;
         //Se almacena la posicion en las variables curvah y curval para luego
         //enviarlas por i2c
         curvah=enco32_16>>8;
         curval=enco32_16;
      }
   }

   //Subrutina para leer la trama de datos del i2c
   void leer(void)
   {
      //Direccion
      while(!i2c_poll());
      trama_control[0]=i2c_read();
      //Numero de Byte a leer
      while(!i2c_poll());
      trama_control[1]=i2c_read();
      for(k=2; k<trama_control[1]+2; k++){
         while(!i2c_poll());
         trama_control[k]=i2c_read();
      }
      return;
   }

   //Subrutina para enviar la trama de datos al maestro por i2c
   void write(void)
   {
      curva[0]=curvah;
      curva[1]=curval;
      curva[2]=interrupcion;
      for (i=0;i<3;i++){
         i2c_write(curva[i]);
      }
```

```
   return;
}

//Subrutina para la configuracion del modulo PWM
void CONFIGPCPWM(void)
{
   setup_power_pwm_pins(PWM_COMPLEMENTARY,PWM_COMPLEMENTARY,PWM_OFF,PWM_OFF
   setup_power_pwm(PWM_CLOCK_DIV_4|PWM_FREE_RUN,1,0,4095,0,1,0);
   return;
}

//Subrutina para la configuracion del modulo encoder
void configencoder(void)
{
   QEICON  = 0b10111000;//Configuracion del encoder y sentido
   POSCNTH = 0;//Posicion del eje
   POSCNTL = 0;//Posicion del eje
   MAXCNTH = 0xFF;//Maixmo Valor del contador
   MAXCNTL = 0xFF;//Maixmo Valor del contador
   DFLTCON = 0b00111000;//Activar Filtro
   return;
}

//Subrutina para la calibracion
void calibracion(void)
{
//PIN_B6 sensor de abajo y PIN_B7 sensor de arriba
//movimiento hacia abajo
   if (input(PIN_B6)==0 && input(PIN_B7)==1){//si esta en sensor de abajo
      while(input(PIN_B7)==1)
      {  //se mueve hacia arriba hasta llegar
         set_power_pwm2_duty(8000);
         set_power_pwm0_duty(0);
      }
      //se frena el motor
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
      //se mueve hacia abajo lento
      while(input(PIN_B7)==0)
      {
         set_power_pwm0_duty(6000);
         set_power_pwm2_duty(0);
      }
      delay_ms(2000);
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
      while(input(PIN_B7)==1)
      {  //se mueve hacia arriba mas lento
         set_power_pwm2_duty(5000);
         set_power_pwm0_duty(0);
      }
   } else if (input(PIN_B6)==1 && input(PIN_B7)==0){//si esta arriba
      while(input(PIN_B7)==0)
      {  //se mueve hacia abajo LENTO
         set_power_pwm2_duty(0);
         set_power_pwm0_duty(6000);
      }
      delay_ms(2000);
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
      while(input(PIN_B7)==1)
      {  //se mueve hacia Arriba mas lento
         set_power_pwm0_duty(0);
         set_power_pwm2_duty(5000);
      }
   } else if(input(PIN_B6)==1 && input(PIN_B7)==1){//si esta en medio
      while(input(PIN_B7)==1)
      {  //se mueve hacia arriba hasta llegar
```

```
            set_power_pwm2_duty(8000);
            set_power_pwm0_duty(0);
         }
         //se frena el motor
         set_power_pwm2_duty(0);
         set_power_pwm0_duty(0);
         //se mueve hacia abajo lento
         while(input(PIN_B7)==0)
         {
            set_power_pwm0_duty(6000);
            set_power_pwm2_duty(0);
         }
         delay_ms(2000);
         set_power_pwm2_duty(0);
         set_power_pwm0_duty(0);
         while(input(PIN_B7)==1)
         {  //se mueve hacia arriba mas lento
            set_power_pwm2_duty(5000);
            set_power_pwm0_duty(0);
         }
      }
      set_power_pwm2_duty(0);
      set_power_pwm0_duty(0);
      POSCNTH=0;
      POSCNTL=0;
}
```

ANEXO 7. Manual de Usuario

PROCEDIMIENTO PARA EXPORTAR GERBER

Para poder fabricar un PCB con la máquina, inicialmente deberá tener el diseño del PCB hecho en cualquiera de los tres software probados con la máquina, que corresponden a Eagle, proteus y PCB wizard. A continuación, deberá exportar desde dicho software el archivo Gerber necesario para fabricar la PCB.

Exportar Gerber desde Eagle:

1. Teniendo el proyecto abierto, escoja en el menú “File” la opción “CAM Processor”. Se abrirá una ventana de configuración con una pestaña sin nombre.
   Tenemos la opción de darle nombre a la pestaña, colocando el nombre deseado en el espacio “Section”. Tal acción no se considera verdaderamente importante, a menos que se desee crear un archivo de configuración previa para el Gerber.
2. En el espacio de “Output”, en la lista desplegable “Device”, seleccionamos la opción “GERBER_RS274X”.
3. En la lista de capas, seleccionamos únicamente la capa donde hemos trabajado, generalmente “Bottom”. Seleccionamos también la capa “Pads”.
   Las capas antes mencionadas corresponden a la capa inferior de cobre donde se encuentran las pistas, y los “Pads” o códigos de aberturas, que corresponden a los agujeros.
4. Con el botón “File” seleccionamos el directorio de salida y el nombre del archivo a exportar. Se le puede asignar la extensión que se desee al archivo gerber.
5. El espacio de offset se configura con ambos valores en “0” cero.
6. En el espacio de “Style” se selecciona únicamente las opciones “Pos. Coord” y “Optimize”.
7. Por ultimo haga click en el botón “Process Job”.
   El archivo Gerber se almacenara en el directorio seleccionado.

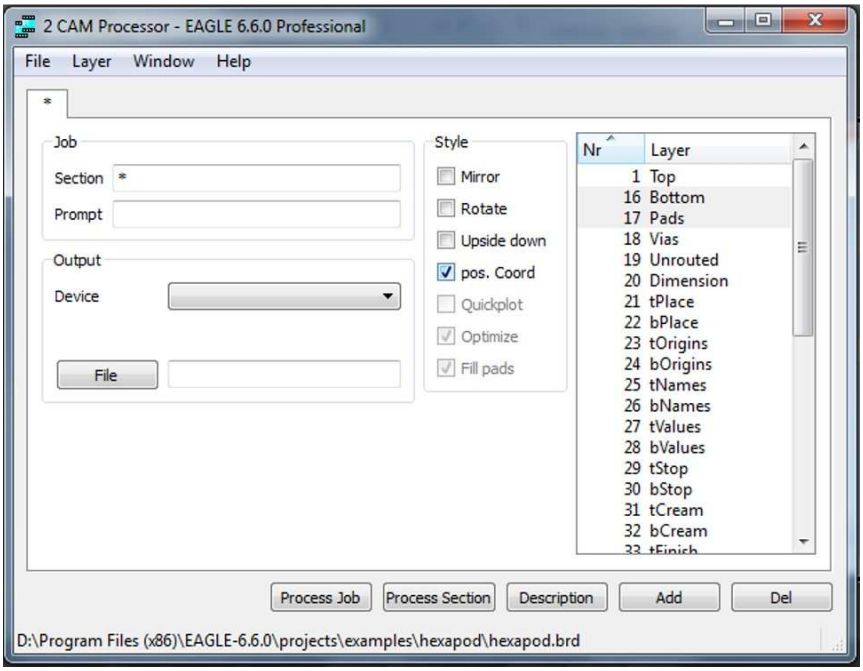


Figura 106. Ventana de "CAM Processor" en blanco.

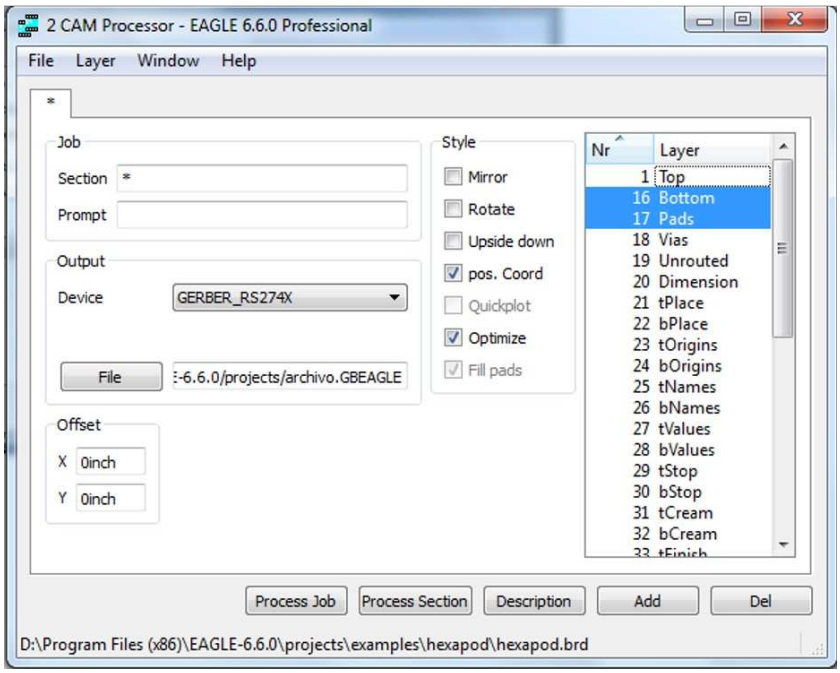


Figura 107. Ventana de "CAM Processor" configurada.

Exportar Gerber desde Proteus:

1. En el menú “output” seleccione la opción “generate Gerber/excellon files”. Aparecerá una ventana preguntando si queremos verificar el diseño de la PCB.
2. Haga click en la opción “Yes”. Aparecerá una ventana de verificación de errores.
3. si no existen errores haga click en “close”. A continuación se abre la ventana principal de configuración para exportar Gerber.
   En la ventana de configuración se configura la capa que se quiere exportar a Gerber, la rotación del diseño, reflejo, unidades y formato del Gerber.
4. Seleccione la capa en la q creo el diseño PCB (generalmente “Bottom copper”). Adicionalmente seleccione la capa “Edge” para que el Gerber contenga la información del tamaño de la placa.
5. Seleccione la rotación que desee (generalmente “X Horizontal”).
6. Seleccione la opción de reflejo “Normal”.
7. Seleccione las unidades que desee (generalmente “thou”).
8. Seleccione como formato la opción “RS247X”.
9. Haga click en “Ok”.
   Revise el directorio fuente del proyecto. En este directorio se exportan los archivos Gerber.

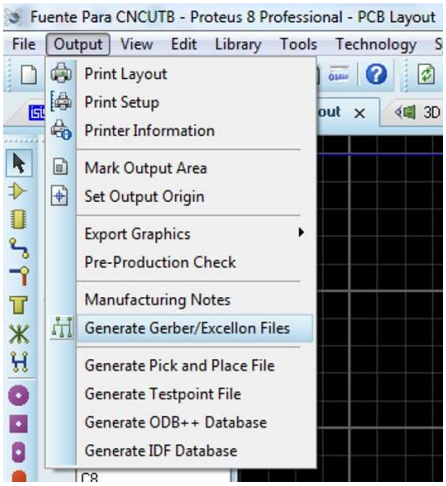


Figura 108. Menú “Output”, Opción “Generate Gerbe/Excellon Files”.

Figura 109. Mensaje de verificación de PCB.

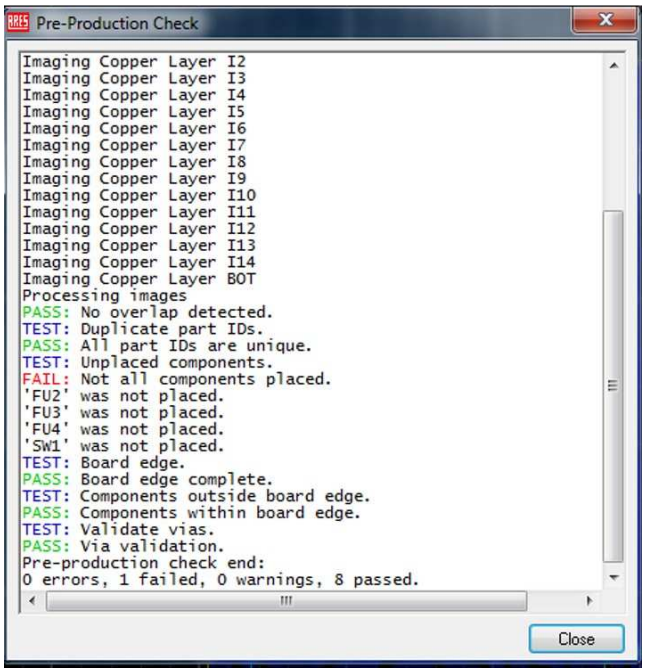


Figura 110. Ventana de resultados de la verificación del PCB.

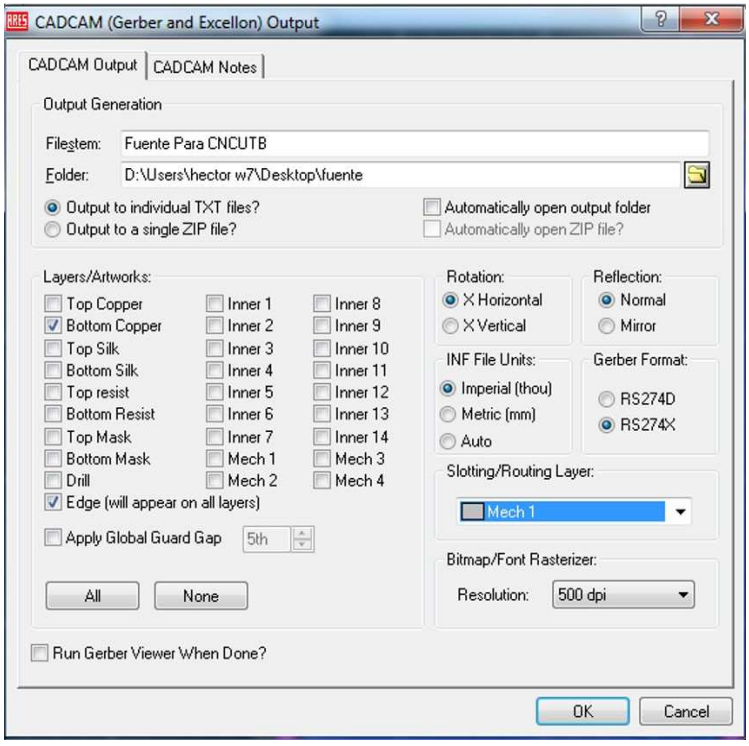


Figura 111. Ventana de configuración del Gerber. (Configuración habitual)

Exportar Gerber desde PCB Wizard:

1. En el menú “Tools”, dentro del submenú “CAD/CAM”, seleccione la opción “Export GERBER”. Inmediatamente aparece la ventana de configuración del Gerber, en la que se puede configurar las capas a exportar en Gerber y el formato de las coordenadas.
2. Seleccione el formato Gerber Extendido “RS274X”.
3. Seleccione las capas “Soldier Side” y “Circuit Board” que contienen la información de las pistas y tamaño de la placa.
4. Configure las coordenadas con Dos (2) unidades, Tres (3) decimales, Métricas (mm), absolutas y sin supresión de ceros “No Zero supression”.
5. Haga click en “Aceptar”. Los archivos serán guardados en el directorio fuente del proyecto.

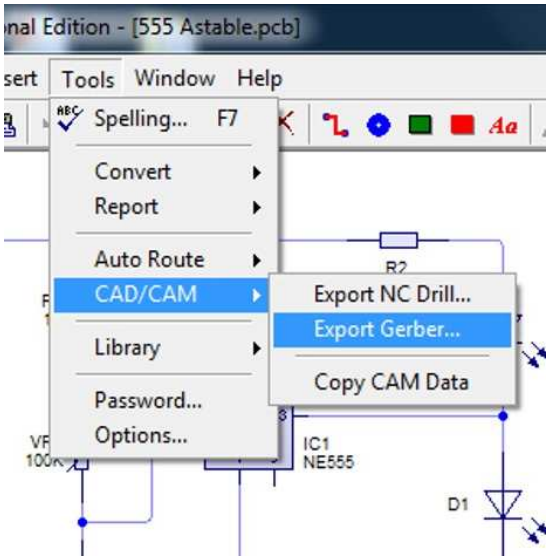


Figura 112. Menú “Tools”, submenú “CAD/CAM”, Opción “Export Gerber”.

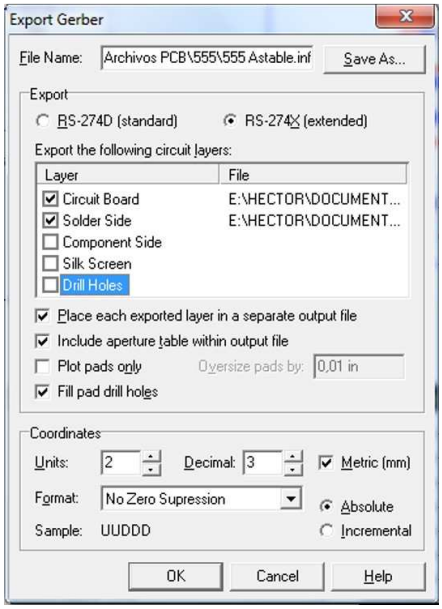


Figura 113. Ventana de configuración del Gerber.

MANEJO DEL SOFTWARE

Una vez se han exportado los archivos Gerber, procedemos a ejecutar el software CNCUTB.exe. Al ingresar al software se visualiza la ventana de la Figura 114. Para comenzar a trabajar, deberá cargar el archivo Gerber en la interfaz de usuario.

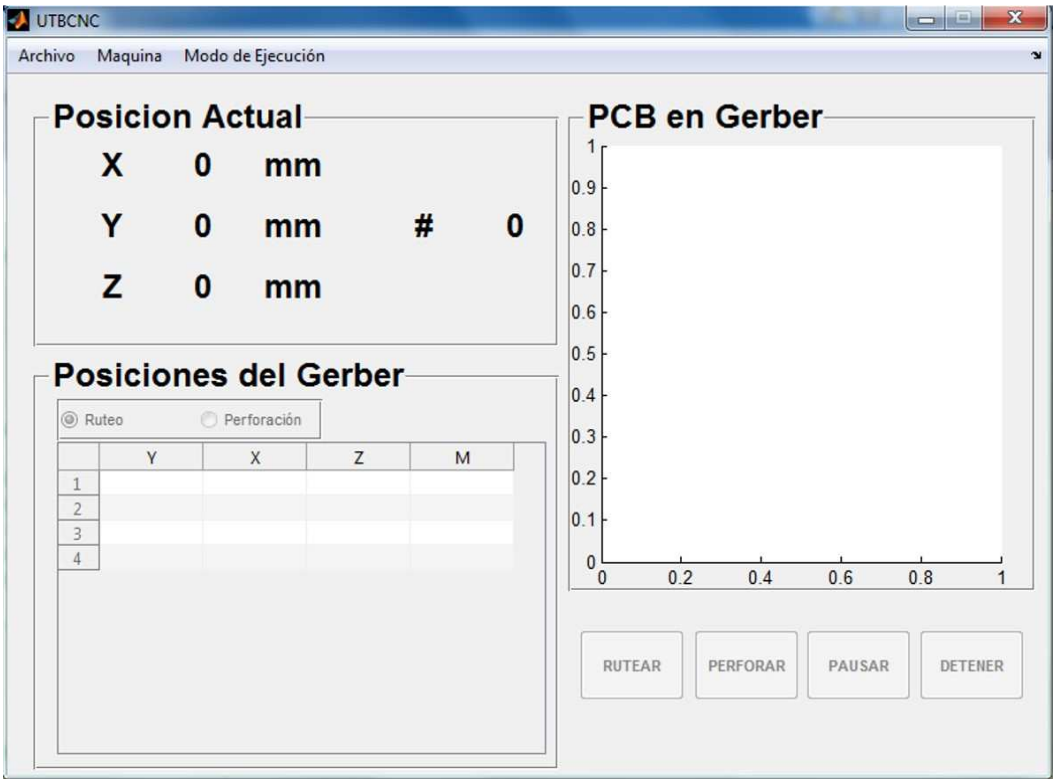


Figura 114. Interfaz gráfica del software de la máquina.

Una vez en el software, realice los siguientes pasos:

1. Ingrese al menú “archivo” y seleccione la opción “abrir archivo Gerber”.
2. En la ventana emergente deberá localizar el archivo Gerber anteriormente generado y seleccionarlo. En la parte inferior de la ventana debe seleccionarse la extensión del archivo Gerber, según sea su procedencia:
    a. (*.gb1): para PCB Wizard.
    b. (*.GBL): Para Altium.
    c. (*CADCAM Bottom Cooper .TXT): Para Proteus.
    d. All files (*.*): Para Eagle y otros softwares.
    La Opción “All files (*.*)” permite seleccionar cualquier otra extensión de archivo, pero es posible que el archivo no sea correctamente interpretado o en su defecto podría ser incompatible con el software.
3. Finalmente, para cargar el archivo, deberá hacer click en el botón “Aceptar”.

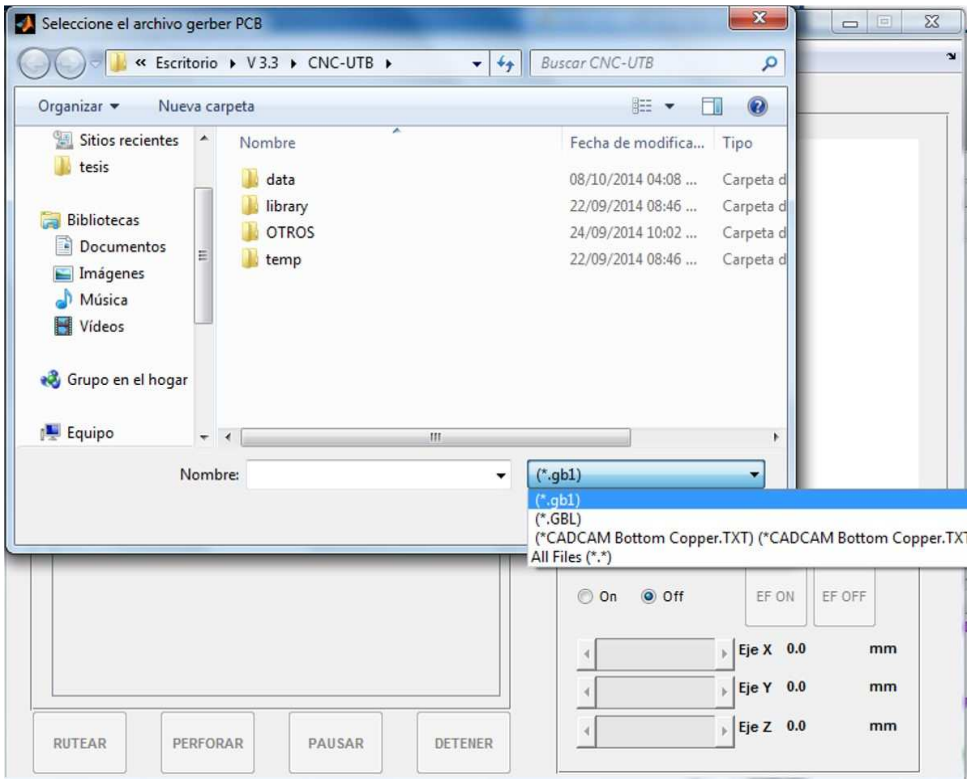


Figura 115. Ventana de selección de archivo.

Una vez cargado el archivo, se muestra en la ventana principal del programa una imagen de la PCB, una tabla que contiene las posiciones (X, Y, Z) en milímetros que será enviada a la máquina y por último se habilitaran algunas opciones del control de la máquina, tales como los botones de “rutear” y “perforar”.

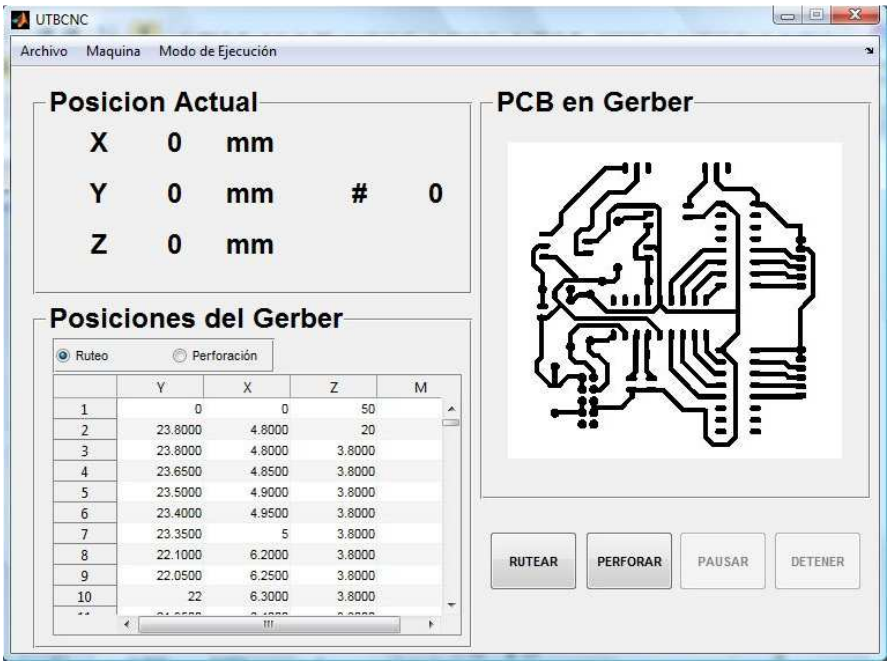


Figura 116. Interfaz gráfica con archivo Gerber cargado.

En el espacio de la tabla puede visualizarse de forma separada las coordenadas de ruteo y perforación, según se seleccione en la parte superior.

Configuración De Copias En Una Misma Placa:

Figura 117. Ventana de configuración de copias.

Si se desea realizar copias del mismo diseño en una placa más grande, haga lo siguiente:

1. Seleccione en el menú “Maquina” la opción “Configurar copias de PCB”. Aparecerá una ventana pequeña.
2. Ingrese la cantidad de copias de quesea realizar.
3. Ingrese el tamaño de la placa que desea utilizar.

4. Haga click en el botón “Vista previa”.
   El software calculara a partir del tamaño de la placa y del diseño si es posible realizar la cantidad de copias deseadas, de no ser posible arrojara un mensaje con el dato exacto de la cantidad de copias posibles para el tamaño de placa anteriormente ingresado y automáticamente modificara dicho valor.
5. Reconfigure la cantidad de copias o el tamaño de la placa, según desee.
6. Haga click en aceptar para terminar el proceso.

Al configurar copias del diseño, la cantidad de posiciones en la tabla (X, Y, Z) aumentara, dependiendo de la cantidad de copias.

Configuración de parámetros de la maquina:

En la ventana de configuración de parámetros de la máquina, existen parámetros fijos que no deben modificarse a menos que el software sea utilizado con un hardware mecánico con especificaciones diferentes a las de este proyecto. Tales como:

- Datos de encoder (Pul/Rev Motor)
- Relación Motor-reductor.
- Paso del husillo.
- Recorrido del eje.
- Supresión de Bits.

Los parámetros antes mencionados, deben configurarse según las características de los motores, encoders y husillos que contenga el hardware mecánico. La supresión de Bits se puede utilizar para disminuir la cantidad de datos capturados por el encoder, dicha configuración debe estar sincronizada con la programación del hardware de control.

El resto de datos, correspondientes a los parámetros de “resolución”, “diámetro de la broca” y “Gap”, se pueden modificar según se desee.

Aumentar la resolución, ocasionará que la imagen creada sea mucho más suave, pero incrementara la cantidad de memoria usada y hará al software más lento. Incremente dicho parámetro con cuidado.

El diámetro de la broca es importante únicamente para el fresado. Ingrese el diámetro real de la fresa.

El parámetro de “Gap” corresponde a la brecha que deja la fresa entre el cobre de una pista y el cobre que la rodea. Se recomienda configurar tal parámetro con la misma dimensión del diámetro de la broca. Incrementar dicho parámetro, aumentara la cantidad de datos que se envían a la máquina y podría demorar mucho más tiempo en ejecutar el trabajo.

COMO TRABAJAR CON LA MÁQUINA

Una vez cargado el archivo Gerber, configuradas copias y parámetros de la máquina, deberá encender y conectar la máquina. Al encender la máquina, esta realiza un procedimiento de pre-calibración de los ejes, deberá esperar a que dicho proceso termine. Una vez terminado el proceso de pre-calibración y antes de comenzar con cualquier proceso de la máquina, verifique que la broca instalada en la maquina corresponde al proceso deseado. En la Figura 118 y Figura 119 se muestran las diferentes brocas para cada proceso.

Figura 118. Fresa de carburo en V (para fresado o ruteo).

Figura 119. Fresas de carburo para perforación.

1. Ubique la placa virgen en los soportes de la mesa. Asegúrese de que está bien ubicada con respecto a la escuadra de la mesa.
2. Ajuste las tuercas de los soportes para sostener la placa. Apretar mucho las tuercas de los soportes provocará una flexión en la placa que perjudicara el acabado de la misma. Se recomienda solo ajustar dichas tuercas.

3. Una vez verificada la información antes mencionada, proceda a pulsar la opción deseada, ya sea “rutear” o “perforar”.

Es importante recalcar que antes de comenzar cualquier trabajo, la maquina ha debido pasar por un proceso de calibración de la mesa y altura de las brocas, de lo contrario podría ocasionar fallas en el funcionamiento de la máquina y obtener malos resultados al fabricar la PCB, además de posibles rupturas de las brocas.

Si durante el funcionamiento de la maquina se llega a detectar cualquier anomalía, proceda a pulsar el botón detener en el software. Si esto no detiene el proceso, proceda a pulsar la parada de emergencia a un lado de la máquina. Esto reiniciara la máquina y deberá reiniciar al proceso que antes estaba ejecutando. Mientras la maquina trabaja, la posición real de la maquina se muestra en la parte superior de la ventana principal del software, incluyendo el número de la línea de posición en la tabla (X, Y, Z). si conoce el número de línea en que la maquina falló y desea continuar el trabajo desde dicha línea, proceda a reiniciar la carga del Gerber, configure las copias necesarias y luego en el menú “Maquina” seleccione la opción “Comenzar desde línea”, donde podrá configurar el número de línea desde el que desea iniciar el trabajo. Se recomienda no iniciar desde la misma posición, sino desde algunas posiciones anteriores, verificando que en dicha línea el valor de la coordenada Z sea mayor a 15 mm, para evitar posibles rupturas de la broca.

CALIBRACIÓN DE LA MESA

Para que la maquina pueda efectuar su trabajo de manera precisa y con un buen acabado, se hace necesario calibrar la mesa para que este bien nivelada, de tal manera que el eje Z se encuentre a la misma altura en cualquier punto de la misma. Para esto, la pesa posee Dos (2) grupos de tornillos, el primer grupo de tornillos fija la mesa a la estructura de la máquina y corresponden a los tornillos más cercanos al centro de la mesa (Tornillos de fijación). El segundo grupo, se utiliza para dar una altura diferente en cada esquina de la mesa y corresponden a los tornillos más cercanos a los extremos (Tornillos de ajuste).

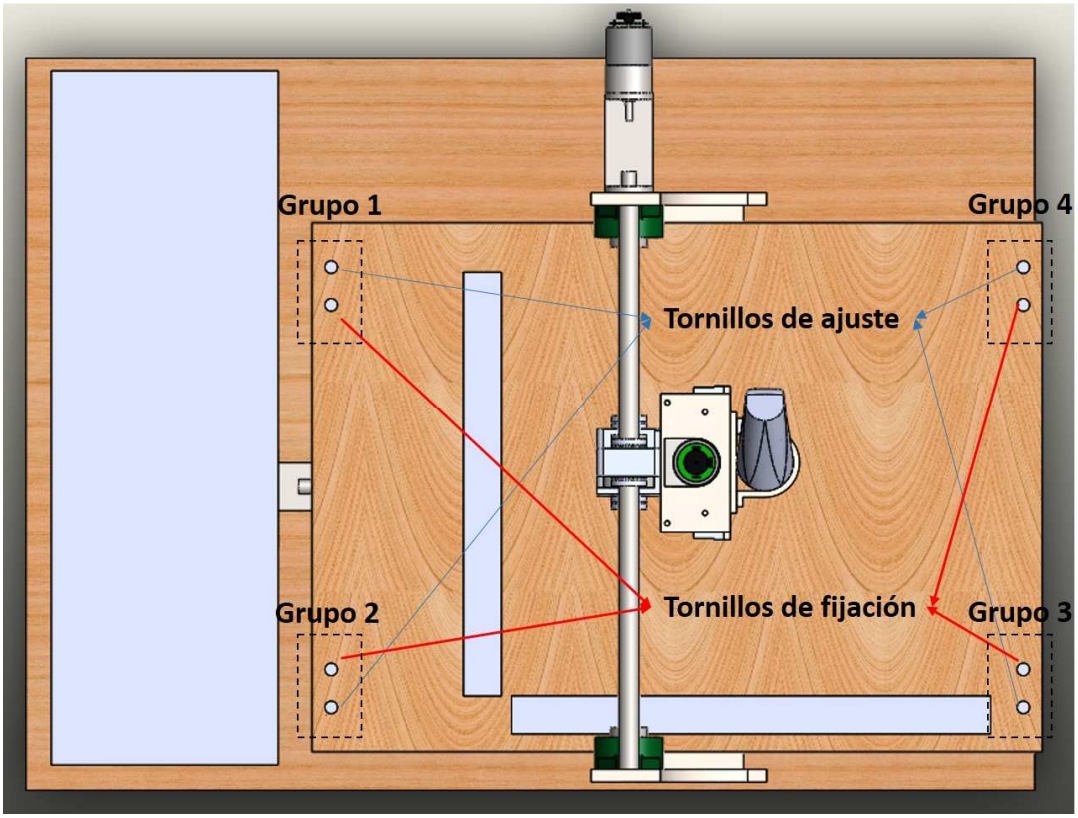


Figura 120. Grupos de Tornillos de calibración de la mesa.

El procedimiento para calibrar la mesa es el siguiente:

1. Teniendo la maquina encendida y conectada al software, haga click en el menú “Maquina”, en la opción “Control Manual”.
2. Suelte todos los tornillos de fijación.
3. Mediante el control manual, mueva la maquina a la posición #1 mostrada en la Figura 121. El eje Z debe quedar lo más cerca posible de la mesa, a una distancia igual al espesor de una placa de PCB. Utilice una placa PCB como un calibrador de holgura para verificar la altura (Ver Figura 125).
4. Utilice el tornillo de ajuste del grupo #1 para ajustar la altura de la mesa en dicha posición.
5. Una vez ajustada la altura, utilice el tornillo de fijación del grupo #1 para fijar la altura de la mesa.
6. Verifique nuevamente la altura de la mesa con la placa PCB usándola nuevamente como calibrador de holgura. Si la altura no es suficiente o es demasiada, utilice nuevamente los tornillos de ajuste y fijación del grupo #1 para corregir.
7. Con el control manual mueva la maquina a la posición #2 mostrada en la Figura 122.
8. Repita los pasos 4, 5 y 6 con el grupo de tornillos #2.

9. Lleve la maquina a las posiciones #3 y #4 (ver Figura 123 y Figura 124). Repita en cada posición el procedimiento explicado anteriormente, esta vez con los grupos de tornillos #3 y #4.
10. Por último, verifique nuevamente la altura del eje Z en los cuatro puntos y si es necesario mueva nuevamente los tornillos de ajuste.

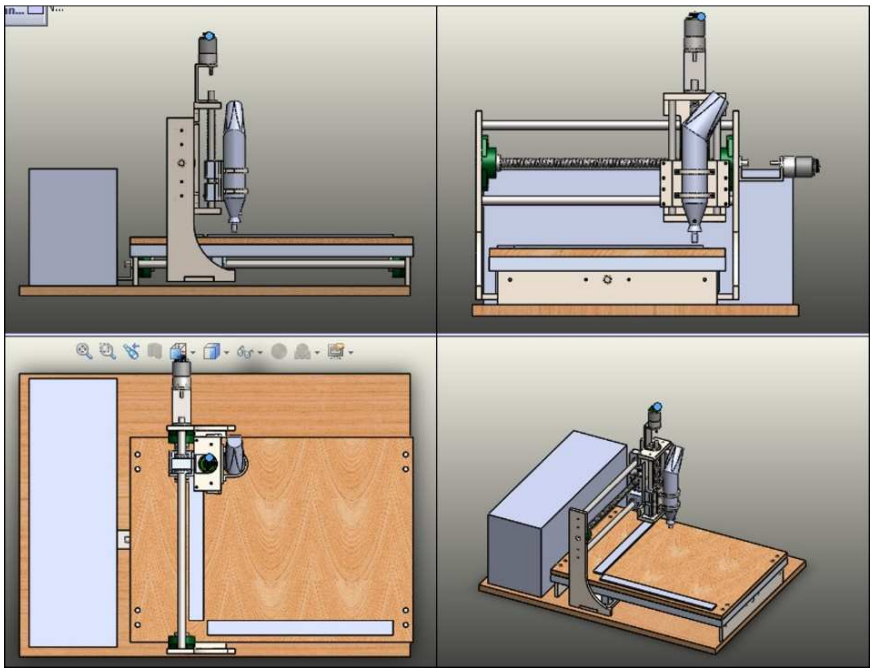


Figura 121. Posición #1.

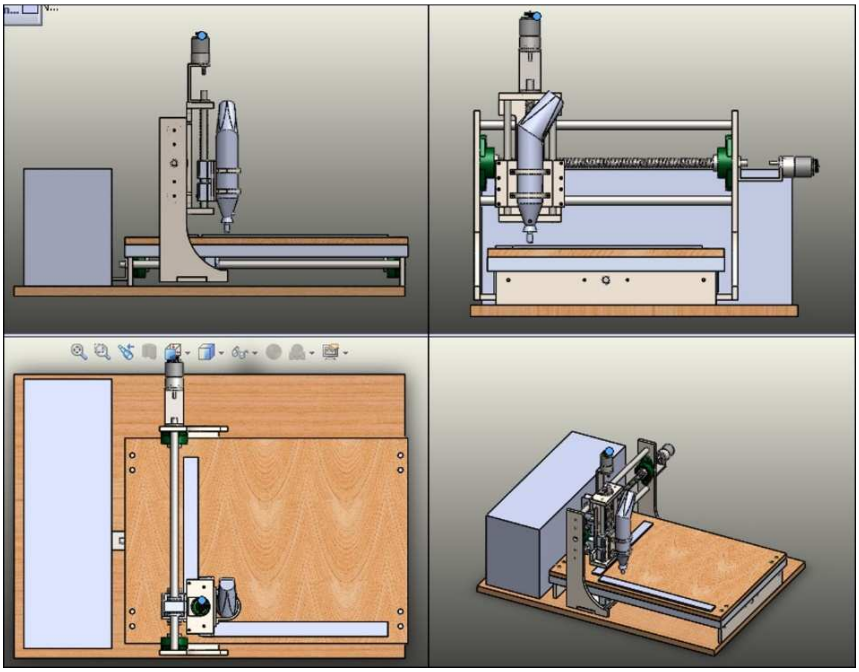


Figura 122. Posición #2.

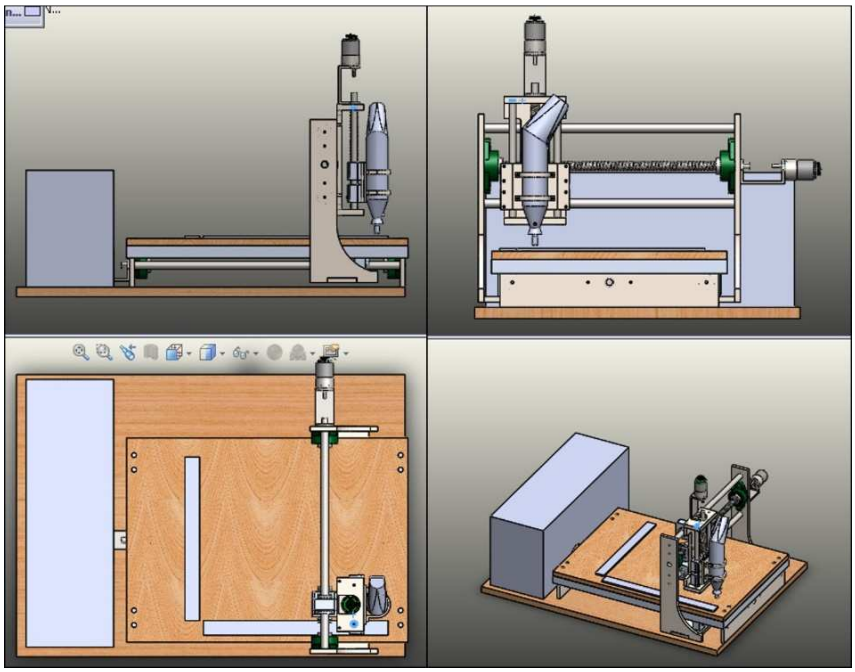


Figura 123. Posición #3.

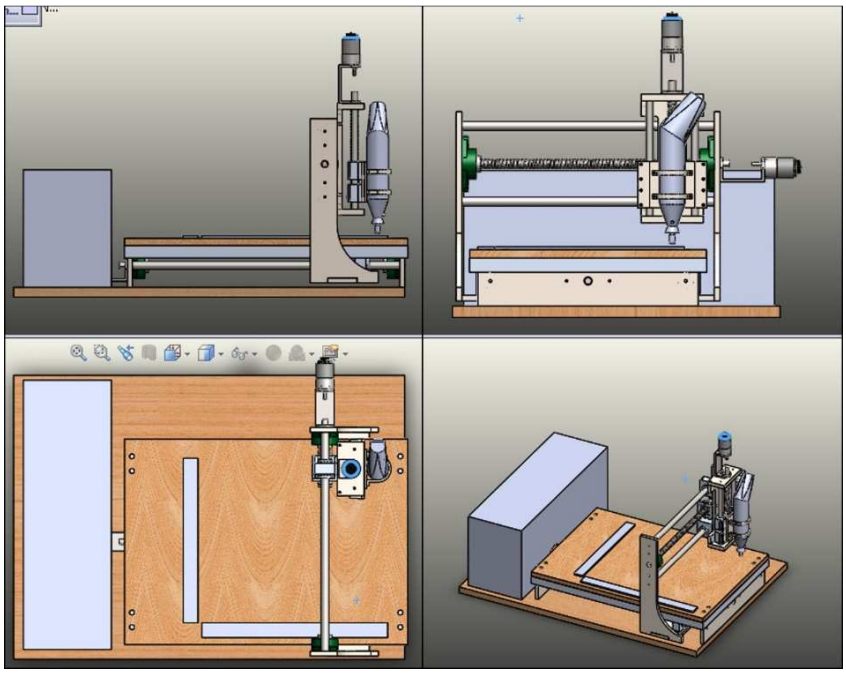


Figura 124. Posición #4.

Figura 125. Uso de PCB virgen como calibrador de holgura.

Se recomienda mover los tornillos de ajuste teniendo los tornillos de fijación flojos, para que la mesa tenga un rango de movimiento. Al ajustar la altura en un punto, apriete siempre el tornillo de fijación y verifique nuevamente la altura, ya que al apretar el tornillo de fijación, puede que cambie la altura ajustada.

CALIBRACIÓN DE LA ALTURA DE LA BROCA

Una vez se ha instalado una broca o fresa en la máquina, se debe calibrar la altura de trabajo, puesto que la fresa o la broca no tienen tope y por tanto no se sabe con exactitud si se encuentra ubicada a la distancia de trabajo correcta. La calibración de la mesa también influye mucho en la altura de trabajo del eje Z, puesto que cada vez que se calibra, tampoco se sabe con exactitud a que altura se encuentra.

Para calibrar las alturas de trabajo del eje Z, debe hacer lo siguiente:

1. Con la maquina encendida y conectada, ingrese al menú “Maquina” y escoja la opción “Control manual”.
   Encontrará tres (3) espacios de trabajo.
2. En el espacio de “Control manual”, escoja la opción “ON”.
3. Con mucho cuidado, utilizando el slider del eje Z, traslade el eje lo más cerca posible a la mesa sin tocarla con la fresa o broca.
4. Con los botones laterales del slider del eje Z, acerque lentamente el eje hacia la superficie de trabajo.

Si se está calibrando una fresa de ruteo, deberá usar una placa virgen como superficie de trabajo. Si por el contrario, se está calibrando una broca de perforación, no es necesario utilizar la placa virgen, basta con la superficie de la mesa.

- Al tocar la superficie, en el caso de una fresa de ruteo, retire la placa y baje unos micrómetros más, aproximadamente 50um.
- Tome el dato de posición en el espacio de trabajo de “Posición actual”.
- Ingrese el dato tomado anteriormente en el espacio de trabajo de “Alturas de efector final”, en el recuadro que le corresponda.
- Escoja la opción “Off” en el espacio de “Control manual”.
- Haga click en aceptar.

La máquina almacenara las alturas configuradas y utilizara dicho dato la próxima vez que cargue un archivo Gerber. Si realiza tal acción con un archivo Gerber ya cargado, deberá volver a cargarlo para actualizar las alturas del eje Z.

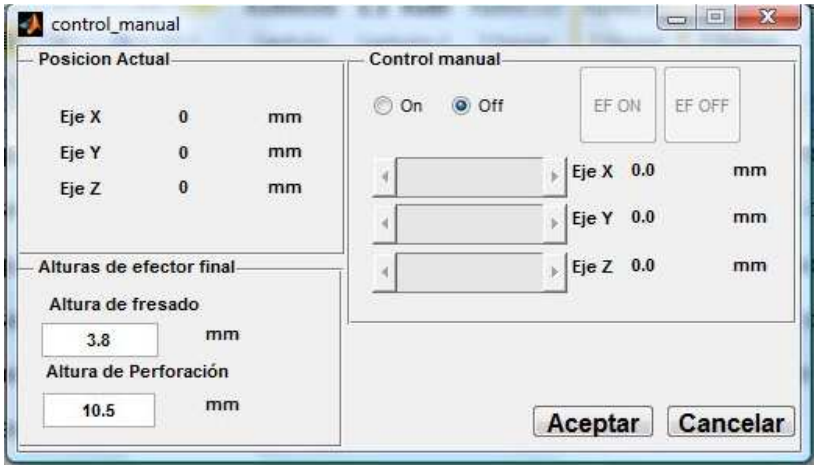


Figura 126. Ventana de control manual y configuración de alturas de fresado y perforación.

CONSEJOS PARA EL MANTENIMIENTO

La máquina depende mucho tanto de la parte mecánica como de la parte electrónica para tener un buen funcionamiento. Por tanto es importante mantenerla limpia de cualquier tipo de residuos metálicos y no metálicos, que pueden generarse mientras la maquina trabaja.

Luego de cada trabajo realizado en la máquina, limpie con un paño limpio y seco la superficie de las barras guía y los husillos, de cualquier residuo generado por la máquina.

Aplique una delgada capa de grasa u otro tipo de lubricante, para proteger las barras guía y husillos de la corrosión. El husillo del eje Z es difícil acceso y constantemente está protegido por la superficie de la mesa, por tanto este puede permanecer más tiempo sin limpieza, solo con una buena lubricación. Una vez al mes retire la mesa y haga limpieza y lubricación al husillo del eje Z.