UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELÉCTRICA

"CONTROL DE UN MANIPULADOR CON DOS GRADOS DE LIBERTAD BASADO EN MICROCONTROLADORES PIC, UTILIZADO COMO ROTULADOR SOBRE MADERA"

TESIS

Que para obtener el Titulo de

INGENIERO EN ELECTRÓNICA

Presenta

Ángel Carreón Silva

Asesor de Tesis

M. I Salvador Ramírez Zavala

Morelia, Michoacán, Noviembre del 2011

Agradecimientos

Agradezco el desarrollo de esta tesis a:

Mi asesor de tesis M. I. Salvador Ramírez Zavala por el apoyo, la paciencia, su experiencia y por compartir sus conocimientos durante las clases y durante el desarrollo de este proyecto.

A los profesores de la facultad que transmitieron sus conocimientos y resolvieron todas las dudas que se tenían en todo momento.

A todos mis amigos con los que compartí experiencias, clases, conocimientos, que me apoyaron y con los que salí adelante dentro de la Facultad: Alan Jiménez R., Miguel Palmerín A., Alejandro Aguilar A., Alejandro Barajas, Francisco López, Víctor Vergara y Manuel Gutiérrez.

A Claudia Juárez V. por todo el apoyo y la motivación que me otorgó durante el desarrollo de esta tesis.

Dedicatoria

A mis Padres José Luis Carreón Flores y Ma. Esthela Silva Chagolla por haberme permitido una formación Académica, por haber inculcado el valor de la educación desde siempre y por haber apoyado mi formación.

A mis Abuelos Bernardo Carreón y Ma. Casimira Flores por su gran apoyo y sabiduría compartida desde el tiempo que he vivido con ellos, por su paciencia, su lucha constante y su gran respeto por la vida.

A todos mis tíos y tías, especialmente a mis tías Irma y Carmen, por su gran apoyo constante, por sus consejos y atenciones que tienen cada momento que lo necesito.

A mis Hermanos José Manuel, Luis Miguel y Víctor Hugo por todo su apoyo, sus consejos y por sus grandes ejemplos a seguir.

A mi tío Gaudencio Silva quien fue un ejemplo de liderazgo y sabiduría. ✝

Resumen

En esta tesis se presenta el control de un manipulador, para el control de movimiento en los ejes X, Y, Z el cual consta con dos grados de libertad y es utilizado como rotulador sobre madera. El manipulador se basa en una estructura de aluminio tipo caballete, en forma de U, sobre la cual descansan los ejes X, Y y el efector final o herramienta, montado sobre el eje Z. Para el controlador del manipulador se utilizan Microcontroladores de la familia de Microchip, en específico los PIC18Fxx31 por su amplia variedad de aplicaciones para el control de motores, ya que poseen 2 módulos importantes para aplicaciones con motores, dichos módulos son el control PWM e Interfaz para la codificación por cuadratura, el primer módulo es importante porque permite una muy versátil estructura de salidas PWM para disparar circuitos de potencia o drivers para manejar motores de todo tipo, y la interfaz para la codificación de cuadratura que permite la medición de velocidad y posición del encoder. El sistema está basado en una interfaz de interacción con el usuario final, mediante una computadora la cual envía y recibe información de velocidad y posición para cada uno de los servomecanismos de los ejes que mueven al manipulador, esta información es leída mediante un microcontrolador maestro, que es quien controla la acción de 3 Microcontroladores esclavos que a su vez controlan cada uno un eje del manipulador, con ello, se establece una sincronización de los 4 Microcontroladores para establecer las acciones de cada uno de ellos, lo cual se verá en la imagen a procesar en la estructura. Para medir la posición y velocidad de los motores, se utilizan Encoders de la familia HEDS de HP acoplados a la flecha de los motores y cada uno con una resolución de 256 pulsos por vuelta. También se utiliza un esquema de conmutación bipolar PWM a través de un convertidor de CD a CD tipo puente construido a base de MOSFET’s para el manejo de los servomecanismos, dicho puente actúa como el driver del motor, recibe señales PWM y manda un valor de voltaje a las terminales del motor.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Símbolos y Abreviaturas

K	Kilo
M	Mega
G	Giga
mm	Milímetros
W	Watts
Hz	Hertz
t	Tiempo
d/dt	Derivada con respecto del tiempo
Kid	Ganancia integral digital
Kpd	Ganancia proporcional digital
Ω	Ohms
R	Resistencia
ω	Velocidad angular
mA	Miliamperes
CNC	Computer Numéric Control
MDF	Medium Density Fiberboard

Capítulo 1

Introducción

1.1 Antecedentes

A lo largo de la historia, el hombre se ha sentido fascinado por máquinas y dispositivos capaces de imitar las funciones y los movimientos de los seres vivos. Los griegos tenían una palabra específica para denominar a estas máquinas: *autómatos*, de la cual se deriva la palabra actual *autómata*, que se define como una máquina que imita la Figura y movimientos de un ser animado [Barrientos, Aracil, 1997].

Posteriormente, en el siglo XXI se introdujo la palabra *Robot* procedente de la palabra checa *robota*, que significa "trabajo obligatorio"; fue empleado por primera vez en la obra teatral R.U.R. (Robots Universales de Rossum).

El término Robótica se refiere a la ciencia o arte relacionada con la inteligencia artificial (para razonar) y con la ingeniería mecánica (para realizar acciones físicas sugeridas por el razonamiento). Este término fue acuñado en 1942 por el bioquímico, escritor y divulgador científico norteamericano de origen ruso Isaac Asimov en su novela corta *Runaround.*

En la actualidad, los robots comerciales e industriales son ampliamente utilizados porque realizan tareas de forma más exacta o más barata que los humanos. También se les utiliza en trabajos demasiado sucios, peligrosos o tediosos para los humanos. Los robots son muy utilizados en plantas de manufactura, montaje y embalaje, en transporte, en exploraciones en la Tierra y en el espacio, cirugía, armamento, investigación en laboratorios y en la producción en masa de bienes industriales o de consumo.

La robótica como parte de la automatización ha ocupado un lugar destacado en el proceso de la modernización de algunas industrias, como la automotriz, la siderurgia y la maquiladora, donde robots manipuladores son comúnmente empleados en tareas repetitivas y de precisión, así como en actividades peligrosas que podrían lastimar operadores humanos.

El robot industrial ha generado una multitud de definiciones, de ellas, la adoptada por el Robot Institute of América (Instituto Norteamericano de Robots) es la que tiene actualmente mayor aceptación [Barrientos, Aracil, 1997, Cloy, 1993]: "Un robot industrial es un manipulador reprogramable con funciones múltiples, diseñado para mover materiales, partes, herramientas o dispositivos especializados a través de movimientos programados variables para el desempeño de una gran diversidad de tareas". Las palabras clave que distinguen a los robots de otras máquinas conocidas son: *"manipulador,* y *"reprogramable"*. La manipulación es el acto de sujetar un objeto, cambiar su posición y orientación en el espacio.

1.2 Objetivo

El objetivo principal de esta tesis es el de controlar un manipulador de dos grados de libertad basado en microcontroladores PIC a partir de una estructura física ya establecida, para ser utilizado como rotulador de figuras en 2 dimensiones sobre madera.

Desarrollar una interfaz gráfica donde el usuario pueda dibujar el modelo a reproducir en el torneado sobre madera, esto sobre una computadora para interpretar los datos dibujados en coordenadas que puedan ser leídas por los microcontroladores.

Aplicar las capacidades que tienen los microcontroladores utilizadas para el manejo y control de velocidad y posición de los motores de corriente directa.

1.3 Justificación

Hoy en día las industrias tienen una gran cantidad de máquinas, manipuladores, actuadores, sensores, grandes avances tecnológicos han hecho que las industrias optimicen sus recursos para mejorar sus servicios o productos, es por ello que se pretende lograr los alcances que tienen los manipuladores de las grandes industrias, en específico las que se dedican al rotulado de piezas como el vidrio, al plástico, madera, o las que se dedican al corte de piezas de diferentes materiales etc. Muchas de las veces, esos manipuladores son de origen extranjero, y son adquiridas a precios bastante elevados, como por ejemplo la Flexicam VL,

Flexicam Pro [TENDU, 2011], o la infoTEC Modelo 1008G [INFOTEC, 2011], esto significa una inversión de las empresas que se recupera en un periodo relativamente largo, dependiendo de las capacidades de producción y ventas de la empresa. Otra de las desventajas es que el encargado de su utilización se encarga únicamente de mandar los datos y supervisar su funcionamiento, no conoce el diseño y la estructura interna del equipo, es por eso que se propone un trabajo de esta naturaleza para optimizar recursos en las industrias.

Actualmente existen gran variedad de Microcontroladores de propósito general, así como también existen Microcontroladores con funciones para hacer tareas específicas, tal como los capaces de manejar y manipular motores, desde medir la posición y velocidad hasta manejar señales PWM con las condiciones adecuadas para su control, dichas condiciones pueden ser; manejo de señales PWM en modo complementario, manejo de tiempos muertos, entre otras, lo que permite ahorrar espacio en circuitería externa y reducir el tamaño de los sistemas.

También es un buen punto de partida para dar seguimiento y crecimiento al área de robótica en la Facultad de Ingeniería Eléctrica, puede ser una excelente opción para dar a conocer más aspectos de la robótica dentro de la carrera de Ingeniería en Electrónica de la Facultad, que se necesita para extender el alcance de conocimientos del egresado de Ingeniería Electrónica.

1.4 Metodología

El movimiento punto a punto es la manera más sencilla de especificar el movimiento de un robot, cuya metodología consiste en determinar una serie de puntos en el espacio de trabajo del manipulador, los cuales son calculados para satisfacer una cierta aplicación, ya sea trayecto o posición fija. El problema de control consistió en hacer pasar el manipulador por dicha cantidad de puntos y rotularlos para formar una figura deseada.

Una manera más general para especificar el movimiento de un manipulador es la llamada trayectoria continua (o trayectoria simplemente), en donde se determina una trayectoria o

curva dentro del espacio de trabajo y el problema de control consiste en hacer pasar el manipulador por dicha trayectoria, y además controlar su velocidad.

El cerebro de un manipulador generalmente tiene tres funciones principales:

a) Iniciar y terminar los movimientos del manipulador en los puntos deseados de acuerdo a la secuencia predefinida.

b) Almacenar información de la secuencia anterior en la memoria.

c) Realizar la interfaz con el entorno o ambiente en que opera el manipulador, es decir, con la computadora.

Se van a utilizar los siguientes elementos importantes en el manipulador:

- Motores de CD (Actuadores).
- Encoders ópticos (Sensores).
- Microcontroladores (Controladores).
- Computadora (Interfaz).

Primeramente se hará el control de 1 eje, se hará el control de posición y de velocidad, para posteriormente, reproducirlo sobre los demás ejes.

Los motores de CD se escogen por su rapidez y bajo costo, son los que se utilizarán para hacer girar los ejes del manipulador.

Los encoders ópticos son los que detectarán la velocidad y posición de los motores.

Los microcontroladores serán capaces de medir esta información enviada por los sensores y harán el cálculo de la velocidad y posición para hacer el control correspondiente con base en esa información, así como la comunicación entre ellos para sincronizar los movimientos de los ejes.

La Computadora será el enlace entre el usuario final y el sistema, a través de ella, se podrán ejecutar los comandos requeridos para hacer el trazado correspondiente enviando información a los microcontroladores correspondientes.

1.5 Descripción de los capítulos

En el capítulo 1 se presentan algunos antecedentes referentes al proyecto, se plantean los objetivos del proyecto, los alcances que se pretenden, y su justificación, así como a metodología que se llevará a cabo para alcanzar los objetivos deseados.

En el capítulo 2 se explica un marco teórico acerca de los manipuladores, diferentes tipos que existen, las características de cada uno de ellos, también se muestran las características del manipulador que se utiliza en este trabajo y un esquema general del cómo se va a realizar.

En el capítulo 3 se describe el hardware necesario para el manipulador, que tipo de manipulador es y cómo está armado, así como el tipo de controlador diseñado. Cómo va a actuar y que métodos de control se le van a aplicar.

En el capítulo 4 se describen y analizan los algoritmos de medición y control de velocidad y posición de los motores para lograr el control de cada uno de los servomecanismos, así como también se describe el software de cada uno de los módulos, tanto de los microcontroladores, como de la interfaz.

En el capítulo 5 se muestran las pruebas realizadas al manipulador, cómo responde ante entradas escalón, respuestas en lazo abierto, respuestas de los controladores de posición y velocidad y los detalles que se fueron presentando al hacer dichas pruebas, así como los resultados del sistema en conjunto, el grabado sobre madera.

En el capítulo 6 se establecen las conclusiones a las que se llegaron con este trabajo, y algunos trabajos futuros para los cuales se podría dar seguimiento.

Capítulo 2

Estructura general de los manipuladores

2.1 Introducción

En este capítulo se presentan los tipos de manipuladores que existen, así como algunos ejemplos que hay en el mercado, también se presenta la estructura general del manipulador empleado, así como la arquitectura de cada uno de sus elementos.

2.2 Geometría de los manipuladores

Un sistema de robot industrial automatizado consta de los siguientes elementos:

- Manipulador o brazo mecánico articulado.
- Controlador
- Actuadores electromecánicos o hidráulicos.
- Elemento terminal (herramienta, aprehensor o efector final).
- Sensores de información.

Cuando un robot manipulador efectúa una tarea programada, se realizan una serie de actividades que resumen de la siguiente manera:

- Interpretación de la tarea programada en lenguaje que pueda interpretar el robot.
- Generación de trayectorias deseadas.
- Ejecución de los algoritmos de control.

Los algoritmos de control ocupan la parte más baja de la escala y en general son transparentes al usuario. Sin embargo, estos algoritmos de control son lo que permitirán obtener un buen desempeño al ejecutar la tarea programada. La gran mayoría de los robots industriales son diseñados para realizar tareas en las cuales el manipulador se desplaza

libremente dentro de su área de trabajo. Por ejemplo, en las tareas de pintado, traslado de objetos de un punto a otro, etc. A este tipo de problemas se les denomina control de movimiento.

Existen 4 categorías en las que caen este tipo de manipuladores, las cuales se muestran en la Figura 2.1, cuyas características definen sus movimientos básicos.

- Coordenadas cartesianas (Tres ejes prismáticos PPP). Figura 2.1a).
- Coordenadas angulares (Tres ejes rotacionales RRR). Figura 2.1d).
- Coordenadas cilíndricas (Un eje rotacional y dos ejes prismáticos RPP). Figura 2.1b).
- Coordenadas esféricas (Dos ejes rotacionales y un eje prismático RRP). Figura 2.1c).

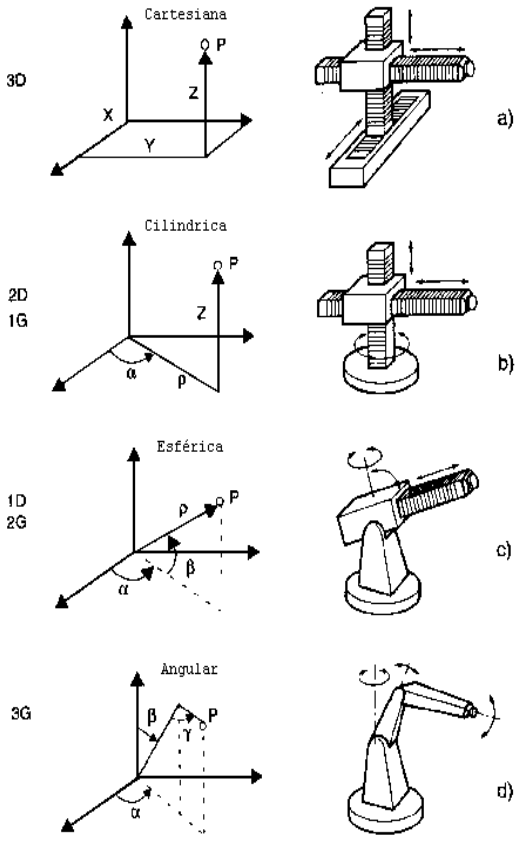


Figura 2.1 Diferentes geometrías de los manipuladores. a) Cartesiana, b) Cilíndrica, c) Esférica, d) Angular.

2.2.1 Geometría cartesiana.

En un robot móvil, el manipulador casi siempre trabaja más allá del borde del la base, y debe ser capaz de ir desde el suelo, hasta por encima de la altura del robot.

Como se muestra en la Figura 2.1a, el manipulador se mueve solamente en línea recta sobre los tres ejes del espacio, y combinando estos movimientos, se pueden hacer movimientos rectos y curvos, pero solo en 2D.

2.2.2 Geometría cilíndrica.

La geometría cilíndrica, permite hacer más movimientos sobre el espacio, en este tipo de geometría, el brazo se mueve en dos ejes de forma rotacional, y en un eje se mueve de forma lineal o recta.

2.2.3 Geometría esférica.

Para el caso de la geometría esférica, el manipulador se mueve en dos ejes en línea recta, mientras que sobre el otro eje, se puede mover en forma rotacional, como se muestra en la Figura 2.1b.

2.2.4 Geometría polar o angular.

La geometría polar o angular tiene la característica principal de poder mover el brazo mecánico sobre los 3 ejes de coordenadas en forma rotacional, lo que permite un amplio rango de movimiento, es decir su rango es una esfera.

Para el caso de la geometría cartesiana cuyos ejes, en la mayoría de los casos son mutuamente ortogonales, es decir, perpendiculares entre sí, y en vista de que la geometría es simple, cada grado de dirección de movilidad corresponde a un grado de libertad en el espacio, y entonces este espacio resulta natural para representar los movimientos en el espacio.

Los manipuladores cartesianos son ampliamente utilizados en la industria ya que se emplean para tornear piezas, para cortar materiales, como dibujadores, y como perforadores. Los actuadores que usan los manipuladores para moverse generalmente son motores eléctricos.

2.3 Manipuladores comerciales

Los manipuladores comerciales también son llamados comúnmente fresadoras, o también llamadas CNC por sus siglas en inglés "*Computer Numeric Control*".

Algunos ejemplos de manipuladores industriales que tienen la geometría cartesiana, se describen a continuación.

La CNC Flexicam VL [TENDU, 2011] que se muestra en la Figura 2.2 es ideal para el procesamiento y corte de metacrilatos, PVC (Polycloruro de Vinilo), aluminio, metales no ferrosos, madera, compuestos sintéticos, otros materiales y que requieran una solida estructura.

La base de la fresadora está diseñada para soportar un trabajo intensivo y libre de vibraciones.

Figura 2.2 Flexicam VL Router

La CNC Stealth mostrada en la Figura 2.3 se usa para el procesamiento y corte de metacrilatos. Al igual que la CNC Flexicam VL. En contraste con la serie VL, que utiliza motores paso a paso, la fresadora CNC Stealth está equipada con motores servo AC. Estructura totalmente en acero de 12 mm de espesor (10 mm en el puente), mecanizada y libre de tensiones.

La CNC infoTEC-1008G [INFOTEC, 2011] que se muestra en la figura 2.4 es de construcción monolítica, de alta rigidez y precisión de parámetros, se utilizan en el grabado de materiales, en estudios, laboratorios joyeros, etc.

Figura 2.3 Flexicam Stealth.

El Tamaño del área de trabajo para las máquinas de grabado es usado para pequeños proyectos artísticos. Este tipo de máquinas permiten por ejemplo:

- Grabado de proyectos en 3D.
- Grabado de máquinas en forma de termoconformado.
- Grabado de las placas que tienen los datos de las máquinas.
- Grabado tarjetas.
- Grabado inscripciones, logotipos, signos etc.

Figura 2.4 infoTEC Modelo 1008G

Estas son sólo algunas, de la gran variedad de maquinaria, en específico de manipuladores que existe en el mercado y que las industrias adquieren para satisfacer las necesidades que requieren.

2.4 Estructura general del manipulador

La estructura General del manipulador que se presenta en este trabajo consta de una geometría tipo cartesiana con dos grados de libertad. En la Figura 2.5 se muestra el diagrama de bloques de la estructura general del manipulador, el cual costa de un controlador maestro, el cual comanda las acciones de tres controladores individuales para las coordenadas X, Y, Z.

Se pretenden utilizar Microcontroladores de la familia de Microchip por sus diferentes ventajas que ofrecen, tal como la velocidad a la que trabajan, son de bajo costo, algunos contienen codificador de cuadratura, para medir pulsos de velocidad y posición, son fáciles de programar, son conocidos y otras características importantes.

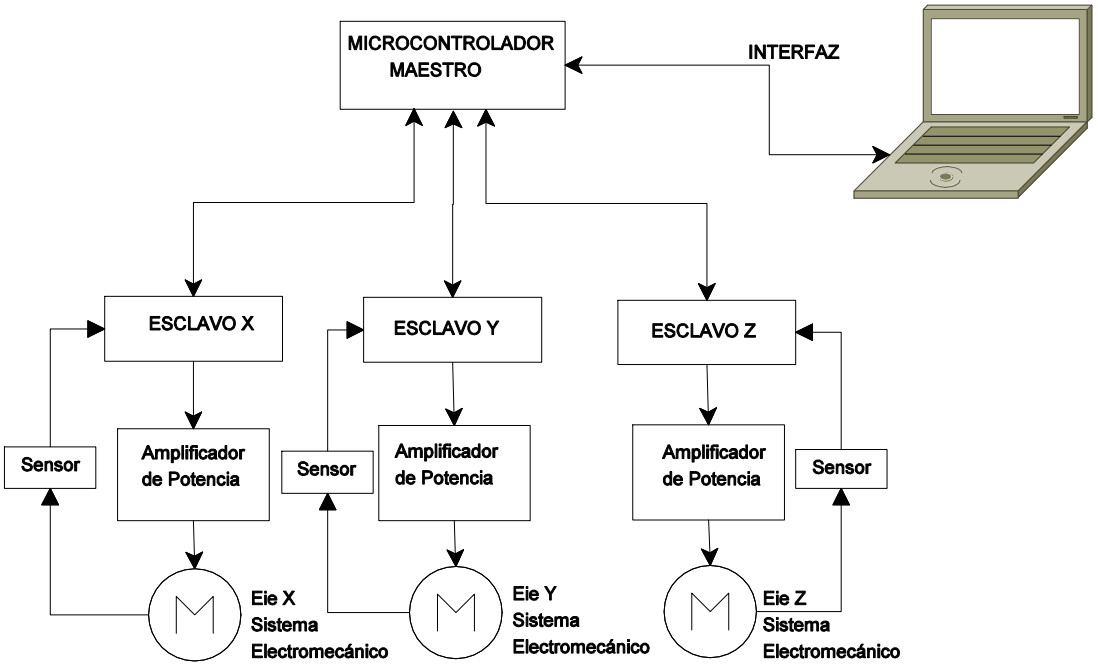


Figura 2.5 Diagrama de bloques General del manipulador.

El esquema general se presenta con un microcontrolador maestro Figura 2.5, que se encarga de comunicarse con la computadora, que a su vez le enviará la información de los trazados

a seguir para hacer el rotulado de las figuras, tres microcontroladores esclavos que son los encargados de hacer el control en lazo cerrado de cada uno de los motores que mueven los ejes X, Y, Z para los movimientos y trazos y envían la señal que indica que se está en la posición de origen de cada eje al microcontrolador maestro.

En seguida se explica brevemente la función que tiene cada elemento del diagrama de bloques de la Figura 2.5.

Interfaz.- Es en este caso es una Computadora personal, se encarga de hacer la comunicación con el usuario final, por este medio se envían las coordenadas para hacer el fresado de las figuras, también es la encargada de hacer las transformaciones de los comandos de posición que se ingresan en forma de coordenadas X, Y, Z a pulsos referentes a la posición de cada motor y a la velocidad con la que se debe mover cada uno de ellos.

Microcontrolador maestro.- Este microcontrolador es el encargado de recibir los datos del PC para enviar los comandos de posición y velocidad a los Microcontroladores esclavos, el Microcontrolador maestro envía dichas señales de control a estos Microcontroladores para ejecutar cada uno su parte de control en lazo cerrado, también son encargados de recibir de los Microcontroladores esclavos las señales que indican que ya se está en la posición de referencia en cada eje, esperar los siguientes comandos de velocidad y posición y mandar las señales de trazado.

Microcontroladores esclavos. Son los encargados de hacer el control en lazo cerrado de los motores para los ejes X, Y, Z de forma independiente, realizar los movimientos de cada eje, y finalizar dichos movimientos.

Amplificador de potencia.- Se trata de un convertidor de CD a CD tipo puente completo, construido a base de MOSFETs. Este amplificador recibe la señal PWM del microcontrolador esclavo, y la convierte a los niveles adecuados de Voltaje y corriente para activar el motor de CD correspondiente a cada servomecanismo.

Sistema electromecánico.- El sistema electromecánico está construido por un motor de CD, una caja de engranaje reductora de velocidad, un tornillo para transformar el movimiento rotacional a lineal y la carga. Para obtener el posicionamiento mecánico se emplean motores de CD que por sus características permiten manipularlos con gran facilidad y precisión en el control. Se utiliza una caja de engranaje como enlace mecánico

para la transmisión de la fuerza motriz y reductora de velocidad del motor, dado que la mayoría de los motores de CD son de alta velocidad y par reducido.

Sensores.- Los sensores proporcionan al microcontrolador esclavo una señal digital correspondiente a la posición y al sentido de la flecha del motor, así como la posición de referencia o la posición cero del manipulador. El sensor para la posición de referencia es un sensor magnético que se encuentra colocado en el origen de los respectivos ejes de movimiento.

En este capítulo se describieron algunos tipos de manipuladores que existen en el mercado, también se describió de manera general la estructura del manipulador que se va a desarrollar, así como también se mencionaron los componentes que conforman el manipulador, y como se emplean.

En el siguiente capítulo se especifican con más detalle los componentes utilizados, así como las partes físicas de la base del manipulador, la conexión de cada uno de ellos, los tipos de acoplamientos de carga que hay y el tipo de acoplamiento empleado para el manipulador.

Capítulo 3

Constitución del hardware del manipulador

3.1. Introducción

En este capítulo se presenta como está constituido el manipulador, todo el hardware que lo compone, la estructura mecánica y toda la circuitería que conforma cada uno de los controladores.

3.2. Estructura del hardware

La estructura del hardware del manipulador es una estructura que ya está definida, es decir, se partió de una base ya construida, con todos los elementos electromecánicos ya diseñados, motores, convertidores CD/CD, tal como se muestra en la Figura 3.1, sólo se tomaron los parámetros de dichos elementos para poder hacer las operaciones necesarias de los controladores del manipulador. Los servomecanismos que se utilizaron tienen reductor de velocidad, y aumentan el par de carga, esto para soportar el peso de cada eje.

3.3. Sistema electromecánico

Dentro de la estructura interna de los manipuladores, generalmente se encuentran los elementos motrices, engranajes y transmisiones que soportan en movimiento el peso de las partes que constituyen el brazo robot.

Estas partes son: la base o pedestal de fijación, el cuerpo y el brazo. Los tres elementos del brazo están relacionados entre sí mediante articulaciones, que pueden ser rotacionales o prismáticas.

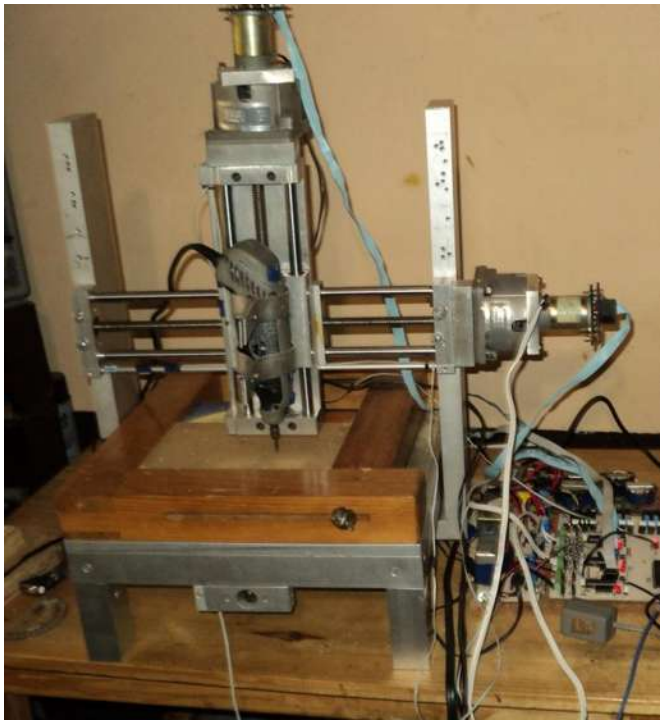


Figura 3.1 Base del manipulador.

Para nuestro caso, el sistema electromecánico para cada eje consta de un motor de CD con un encoder montado en la flecha, acoplado a una caja de engranes reductora y un tornillo tipo sin fin para transformar el movimiento rotacional a lineal. La Figura 3.2 muestra el diagrama de bloques del manejador de servomecanismo:

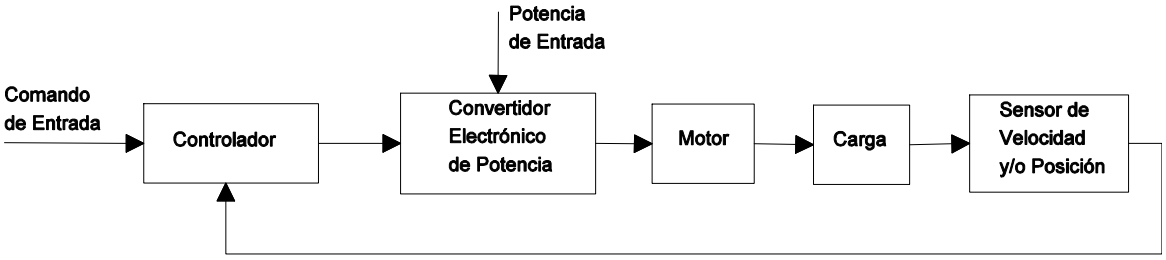


Figura 3.2 Manejador de servomecanismo.

Existen varios tipos de mecanismos de acoplamiento, por ejemplo:

- Manejador con acoplamiento directo.
- Manejador con acoplamiento por tornillo.
- Manejador con acoplamiento tangencial.

- Cualquiera de los anteriores con reducción de engranaje.

Dichos manejadores se analizan enseguida, ya que dos de ellos se requieren para plantear el tercero que es el que utiliza el manipulador.

3.3.1. Manejador con acoplamiento directo y reducción de engranaje

Este tipo de manejador se muestra en la Figura 3.3 [Ramírez, 1998]

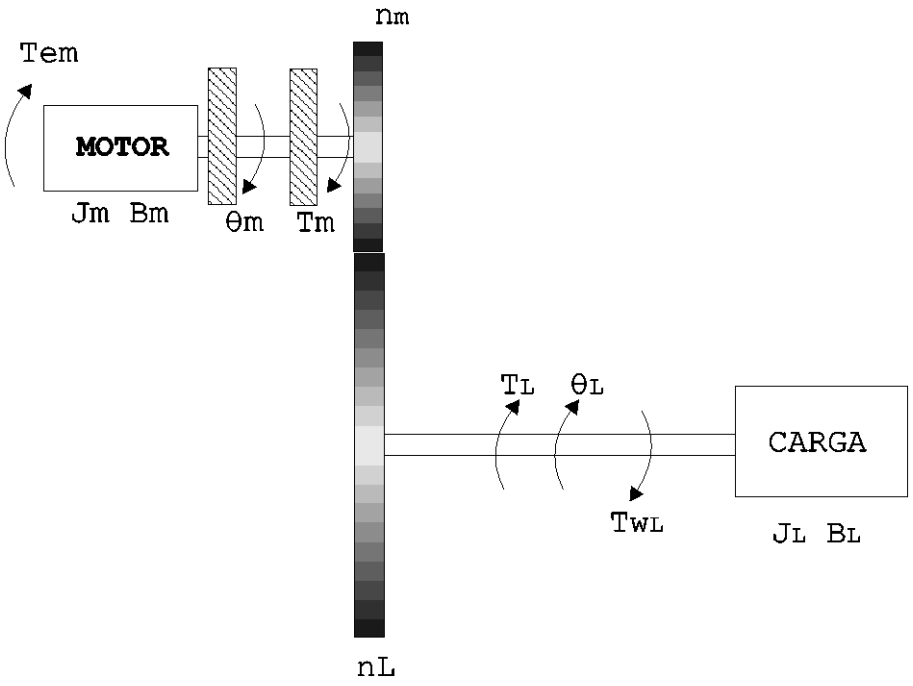


Figura 3.3 Manejador con acoplamiento directo y reducción de engranaje.

Considerando una eficiencia del 100 % en los engranes, esto es: $P_{ent}=P_{sal}$, los pares en ambos lados del engranaje están relacionados por la ecuación (3.1).

$$\frac{T_m}{T_L} = \frac{\omega_L}{\omega_m} = \frac{\theta_L}{\theta_m} = \frac{n_m}{n_L} = a \qquad (3.1)$$

Donde:

$T_{ml,}$ T_L son los pares del motor y carga respectivamente.

$\omega_{m,}$ ω_L son las velocidades angulares del motor y la carga respectivamente

$\Theta_{m,}$ Θ_L son las posiciones angulares del motor y la carga respectivamente

n_m, n_L son los números de dientes del engrane del lado del motor y del lado de la carga.

a es la relación de transformación.

En el eje del motor tenemos:

$$J_m \dot{\omega}_m = \sum T = T_{em} - T_m - B_m \omega_m \tag{3.2}$$

En el eje de la carga

$$J_L \dot{\omega}_L = T_L - T_{WL} - B_L \omega_L \tag{3.3}$$

Así,

$$T_L = J_L \dot{\omega}_L + T_{WL} + B_L \omega_L \tag{3.4}$$

Sustituyendo en la ecuación (3.4) el par en la carga, dado en la ecuación (3.1), se tiene

$$T_m = J_L \dot{\omega}_L a + T_{WL} a + B_L \omega_L a \tag{3.5}$$

Con las relaciones (3.1), poniendo todo en función del eje del motor, se tiene:

$$T_m = J_L \dot{\omega}_m a^2 + T_{WL} a + B_L \omega_m a^2 \tag{3.6}$$

Sustituyendo la ecuación (3.6) en la ecuación (3.2), tenemos:

$$J_m \dot{\omega}_m = T_{em} - J_L \dot{\omega}_m a^2 - T_{WL} a - B_L \omega_m a^2 + B_m \omega_m \tag{3.7}$$

$$(J_m + J_L a^2)\dot{\omega}_m = T_{em} - T_{WL} a - (B_L a^2 + B_m)\omega_m \tag{3.8}$$

Con las relaciones (3.1), poniendo todo en función del eje del motor, se tiene:

$$T_{em} = (J_m + J_L a^2)\frac{\omega_L}{a} + T_{WL} a + (B_m + B_L a^2)\frac{\omega_L}{a} \tag{3.9}$$

Donde:

T_{em} es el par electromagnético del motor

J_m, J_L son los momentos de inercia del motor y carga respectivamente

T_{wL} es el par de trabajo de la carga

ω_L es la aceleración de la carga

B_m, B_L son los factores de fricción viscosa del motor y de la carga, respectivamente

3.3.2. Manejador con acoplamiento por tornillo

Este tipo de manejador se muestra en la Figura 3.4 [Ramírez, 1998]

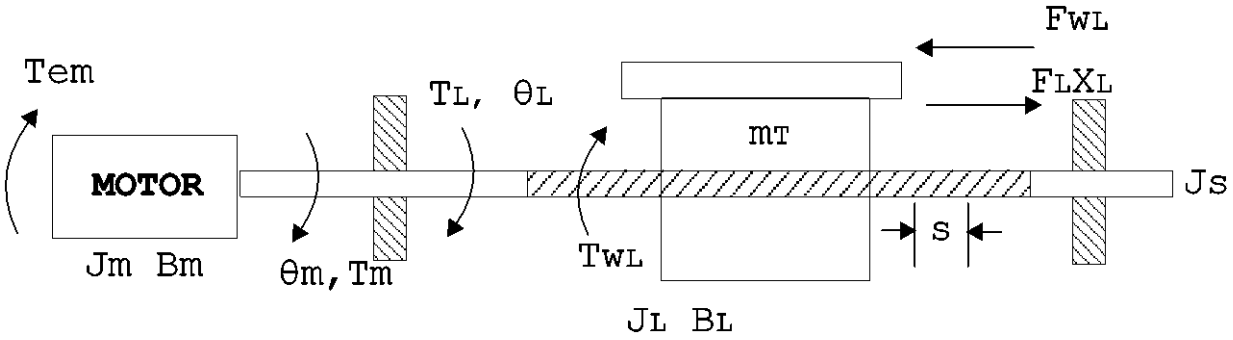


Figura 3.4 Manejador con acoplamiento por tornillo

El par y la fuerza están relacionados por la ecuación (3.10).

$$\frac{T_m}{F_L} = \frac{v_L}{\omega_m} = \frac{x_L}{\theta_m} = \frac{s}{2\pi} = a_1 \qquad (3.10)$$

Donde:

F_L es la fuerza de desplazamiento lineal

v_L es la velocidad lineal

x_L es el desplazamiento lineal

s es el paso de tornillo

a_1 es la relación de transformación

M_T es la masa total a mover

J_s es la inercia del tornillo

T_{wL} es la fuerza de trabajo de la carga

El momento de inercia reflejado en el tornillo por efecto de la carga se puede deducir a partir de la ecuación (3.11).

$$W = T_m \theta_m = F_L x_L \tag{3.11}$$

Con lo cual:

$$(J_L \ddot{\theta}_m)\theta_m = (M_T \ddot{x}_L)x_L \tag{3.12}$$

Así, la inercia reflejada de la carga está dada por la ecuación (3.13).

$$J_L = \frac{M_T \ddot{x}_L x_L}{\ddot{\theta}_m \theta_m} \tag{3.13}$$

Aplicando la ecuación (3.10) en la ecuación (3.13), se obtiene:

$$J_L = \frac{M_T a_1 \ddot{\theta}_m \theta_m a_1}{\ddot{\theta}_m \theta_m} = M_T a_1^{\,2} = M_T \left(\frac{s}{2\pi} \right)^2 \tag{3.14}$$

Si la transmisión de potencia es del 100 %, tenemos

$$T_{WL} \omega_m = F_{WL} v_L \tag{3.15}$$

Así,

$$T_{WL} = F_{WL} \frac{v_L}{\omega_m} = F_{WL} a_1 = F_{WL} \frac{s}{2\pi} \tag{3.16}$$

Si lo observamos todo como una carga acoplada directamente al motor, tenemos:

$$(J_m + J_s + J_L)\dot{\omega}_m = T_{em} - a_1 F_{WL} \tag{3.17}$$

Y utilizando las relaciones (3.10) en la ecuación (3.17) se tiene:

$$T_{em} = \frac{\dot{v}_L}{a_1}(J_m + J_s + a_1^{\,2} M_T) + a_1 F_{WL} \tag{3.18}$$

Donde

$\dot{v}_L$ Es la aceleración lineal de la carga.

3.3.3. Manejador con acoplamiento por tornillo y reducción de engranaje

Este tipo de manejador se muestra en la Figura 3.5 [Ramírez, 1998]

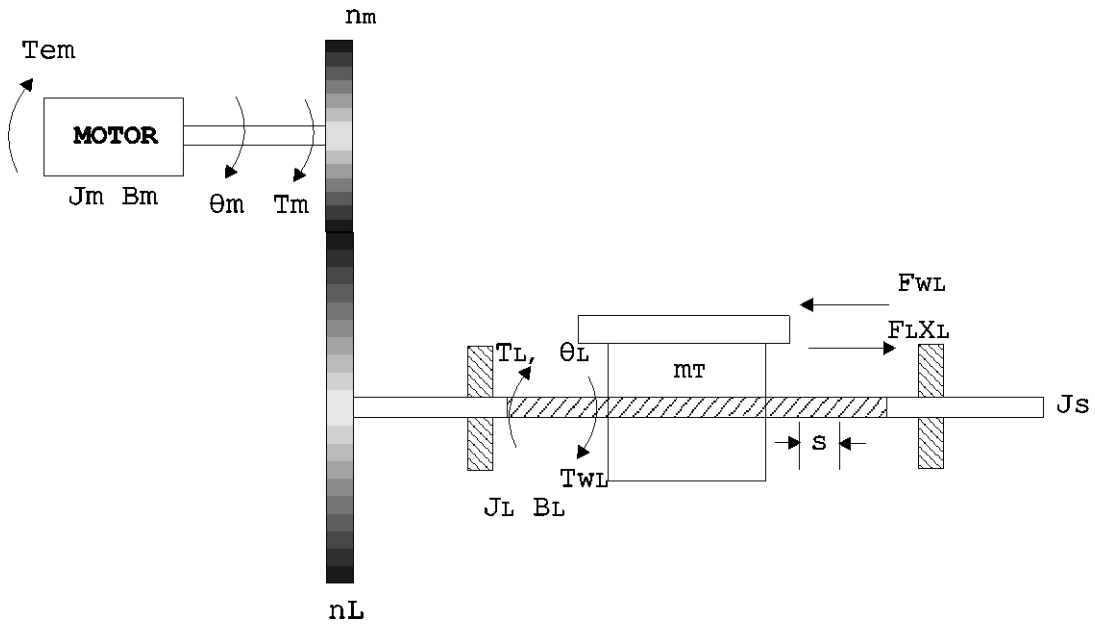


Figura 3.5 Manejador con acoplamiento por tornillo y reducción de engranaje.

Considerando una eficiencia del 100% en los engranes, es decir, $P_{ent}=P_{sal}$, los pares en ambos lados del engranaje están relacionados por la ecuación (3.1). Para el tornillo, se aplican las relaciones de la ecuación (3.10).

El par electromagnético para el caso de acoplamiento con reducción de engranaje está dado por la ecuación (3.9). La inercia de la carga para el manejador con acoplamiento por tornillo con reducción de engranaje está dada por la suma de la inercia del tornillo y la de la carga (dada por la ecuación (3.14)). Aplicando las relaciones (3.10) y la ecuación (3.16), (En donde se cambian los subíndices m por L para ajustar la nomenclatura a la de la Figura 3.5) en la ecuación (3.9) obtenemos:

$$T_{em} = (J_m + (J_S + J_L)a^2)\frac{\dot{v}_L}{a_1 a} + F_{WL}a_1 a + (B_m + B_L a^2)\frac{v_L}{a_1 a} \qquad (3.19)$$

Las ecuaciones (3.9), (3.18) y (3.19) se pueden aplicar para obtener el par electromagnético del motor a utilizar, de acuerdo al tipo de manejador empleado. La potencia mecánica del motor está dada por:

$$P_m = \omega_m T_{em} \tag{3.20}$$

Mientras que la potencia eléctrica del motor de CD está dada por la ecuación (3.21)

$$P_e = e_a i_a \tag{3.20}$$

Donde:

e_a es el voltaje inducido en la armadura

i_a es la corriente en la armadura

En estado estable, se tiene que:

$$P_e \cong P_m$$

De manera que:

$$P_e \cong \omega_m T_{em} \cong e_a i_a \tag{3.22}$$

Esto es, la potencia eléctrica del motor se puede estimar si se conoce el par electromagnético y la velocidad del motor en estado estable.

3.4. Estructura del manipulador

La configuración mecánica del manipulador, como ya se había mencionado es del tipo cartesiana, como se observó en la Figura 2.1a, con dos grados de libertad; esto quiere decir que se puede desplazar a través de los ejes X, Y. Este tipo de estructura, y la relación con sus elementos proporciona una configuración mecánica con parámetros que debemos conocer para lograr los movimientos de cada eje.

Cada uno de los ejes, permite un movimiento independiente, regulado por un motor de CD con encoder óptico montado en la flecha, acoplado a cada eje a través de un reductor de engranaje y tornillo sin fin, esto permite transformar el movimiento rotacional a lineal, este acoplamiento se muestra en la Figura 3.6.

El acoplamiento entre los ejes está hecho de una estructura de aluminio en forma de "U". Ambos ejes están montados sobre una base de aluminio, como se muestra en la Figura 3.1,

donde el eje Y se encuentra debajo de la base, mientras que el eje X está en la parte de arriba.

Figura 3.6 Acoplamiento de los ejes X e Y del manipulador.

Cada motor es alimentado por su propio amplificador de Potencia. El efector terminal se posiciona en cualquier punto sobre un plano de 30cm x 30cm, con una velocidad de desplazamiento entre un rango de 0 y 0.04 cm/seg (10 rev/seg en el motor), y una resolución de $3.9x10^{-6}$ cm/rev. En la Tabla 3.1 se resumen las especificaciones del manipulador empleado.

Tabla 3.1 Especificaciones del manipulador empleado.

Tipo de coordenadas	Cartesianas
Grados de libertad	2
Elementos motrices	Servomotores Eléctricos de CD
Área de trabajo	30 cm x 30 cm
Velocidad máxima de desplazamiento lineal por eje	0.04 cm/seg
Velocidad de desplazamiento lineal de cada motor	10 rev/seg
Controladores	PIC18F2431, PIC18F4550
Peso del motor con encoder	275 gr
Peso del Acoplamiento entre el motor y el engranaje	150 gr
Peso de la caja de engranaje	400 gr
Peso del acoplamiento entre el engranaje y el tornillo	150 gr

Peso del tornillo de acoplamiento	1.925 Kg
Peso de cada uno de los ejes	2.9 Kg
Peso de la estructura de aluminio en "U"	1.425 Kg
Peso total del manipulador	9.575 Kg
Relación de engranaje	1:75
Voltaje del motor de CD	24 V
Potencia del motor de CD	5 W
Diámetro del motor de CD	3.8 cm
Longitud del tornillo	38 cm
Diámetro del tornillo	1.16 cm
Paso del tornillo	0.3 cm/rev

En seguida se presentan los cálculos realizados para obtener los parámetros mecánicos del eje Y, ya que es el más crítico porque soporta el mayor peso del manipulador. En la ecuación (3.19), ya que el producto de las relaciones de transformación ($a_1 a$) es muy pequeño, podemos despreciar el par debido a la fuerza de trabajo de la carga ($F_{WL} a_1 a$). Tomando en cuenta que la fricción del manipulador es muy pequeña, la podemos despreciar.

Y si utilizamos las relaciones (3.1) y (3.10) obtenemos:

$$T_{em} = (J_m + (J_S + J_L)a^2)\omega_m \tag{3.23}$$

Ahora, para la obtención del par electromagnético dado por la ecuación (3.23), partimos de que se tienen motores de CD de 24 volts y 5 watts de potencia. Para obtener la inercia del motor consideramos el motor como un cilindro sólido, donde la inercia del motor está dada por la ecuación (3.24).

$$J_m = \frac{mD^2}{8} = \frac{wr^2}{2g} \tag{3.24}$$

Donde:

M es la masa del motor = w/g [Kg-seg^2/m]

D es el diámetro del motor = 3.18×10^{-2} m.

w es el peso del motor = 0.275 Kg.

r es el radio del motor = 1.9×10^{-2} m.

g es la constante gravitacional = 9.8×10^{-2} m/seg^2.

La inercia del motor será de:

$$J_m = \frac{(0.275Kg)(0.019m)^2}{2(9.8m/s^2)} = 5.065\times10^{-6}\ Kg\cdot m\cdot s^2$$

La inercia del tornillo de acoplamiento está dada por:

$$J_s = \frac{\pi L\rho r^4}{2} \qquad (3.25)$$

Donde:

L es la longitud del tornillo = 0.38 m.

ρ es la densidad del acero = 7750 Kg/m^3.

r es el radio del tornillo = 0.0058 m.

La inercia del tornillo será de:

$$J_s = \frac{\pi}{2}(0.38m)(7750Kg/m^3)(0.0058m)^4 = 5.23\times10^{-6}Kg\cdot m^2$$

La inercia de la carga está dada por la ecuación (3.14), donde la masa total para el eje Y es la suma de las masas que soporta el motor y está dada por:

M_{TOTAL} = (Peso del tornillo del eje + Peso de la estructura de aluminio

+ Peso de la estructura del eje X+ Peso del motor de orientación con carga)/g

Así, la masa total será:

$$M_{TOTAL} = \frac{(1.9\,Kg + 1.425\,Kg + 2.9Kg + 0.45Kg)}{9.8\,m/s^2} = 0.68112\,Kg \cdot seg^2/m$$

Aplicando la ecuación (3.14) tenemos que:

$$J_L = (0.68112\,Kg \cdot seg^2/m)\left(\frac{0.3\,m/rev}{2\pi\,rad/rev}\right)^2 = 1.55272 \times 10^{-3}\,Kg \cdot m \cdot seg^2$$

3.5. Controladores de los mecanismos

Los controladores de los servomecanismos se hicieron mediante el microcontrolador PIC18F2431 de la familia microchip, el cual opera con un reloj externo, con base en un cristal oscilador de alta velocidad (20 MHz). El diagrama de bloques de la Figura 3.7 muestra la estructura de cada servomecanismo del manipulador. El microcontrolador se encarga de medir la velocidad y posición de calcular la acción de control de su motor correspondiente. El microcontrolador se encarga también de hacer la comunicación con el microcontrolador maestro PIC18F4550 que se encarga de transmitir y recibir los comandos de desplazamiento.

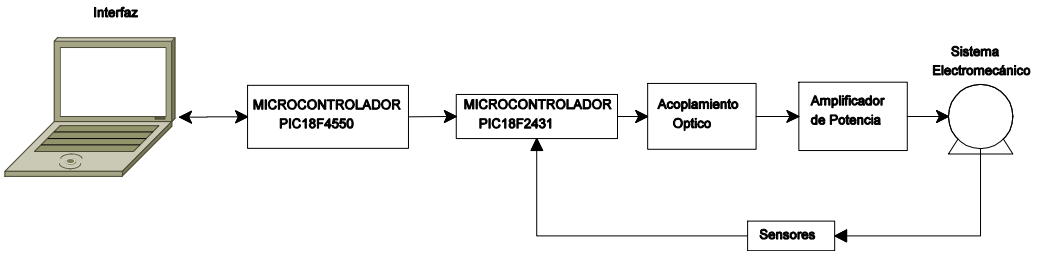


Figura 3.7 Diagrama de bloques de cada servomecanismo.

3.6. Microcontrolador PIC18F2431

La Figura 3.8 muestra el diagrama de pines del microcontrolador PIC18F2431 [Microchip, 2011], el cual contiene 16Kb de memoria de programa, 768 Bytes de memoria RAM interna, 256 Bytes de memoria EEPROM interna, 2 módulos de Captura/Comparación/PWM, 1 Módulo de comunicación EUSART, 1 Módulo de comunicación SSP I2C/SPI, 4 timers, 1 de 8 bits y 3 de 16 bits, 5 canales de conversión

Analógico a Digital de 10 bits cada uno, 8 módulos de control de potencia PWM, Detección de movimiento/Interfaz de Cuadratura para Encoder.

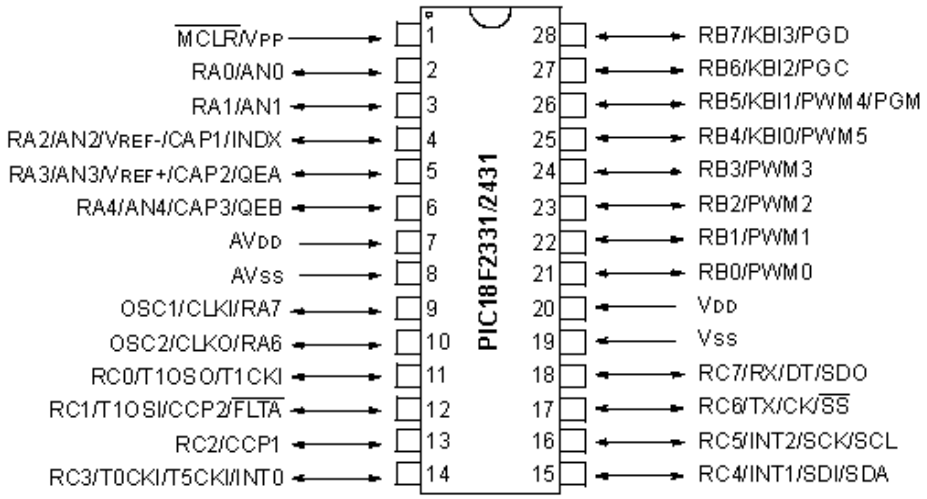


Figura 3.8 Diagrama de pines del microcontrolador PIC18F2431

El microcontrolador PIC18F2431 opera de la siguiente manera: Los pulsos enviados por los sensores de posición son recibidos por el microcontrolador a través de los canales INDX, QEA y QEB que corresponden a los pines 4, 5 y 6 respectivamente, del modulo de Interfaz de codificación por cuadratura., el canal QEA captura la señal del encoder correspondiente al canal A, el canal QEB captura la señal del encoder correspondiente al canal B, y el canal INDX captura la señal del encoder correspondiente al canal I.

3.6.1. Módulo de control PWM

Luego de calcular la acción de control, el microcontrolador envía la señal PWM necesaria para activar el motor a través de los pines RB0, RB1 del microcontrolador esclavo, mediante el modulo de control de potencia PWM.

Este módulo especial en los Microcontroladores de la serie 18Fxx31, es un modulo bastante útil para el manejo de motores de todo tipo, en la Figura 3.9 se muestra el diagrama de bloques del módulo de Potencia PWM.

Este módulo contiene cuatro generadores de ciclo de trabajo, numerados del 0 al 3, el módulo cuenta con ocho pines de salida PWM, numerados del 0 al 7 en el que las ocho

salidas PWM se agrupan en pares de salida pares y salidas numeradas impares. En el modo complementario, las salidas PWM pares siempre debe ser el complemento de las salidas PWM impares correspondientes. Por ejemplo, PWM0 será el complemento de PWM1, WM2 será el complemento de PWM3 y así sucesivamente.

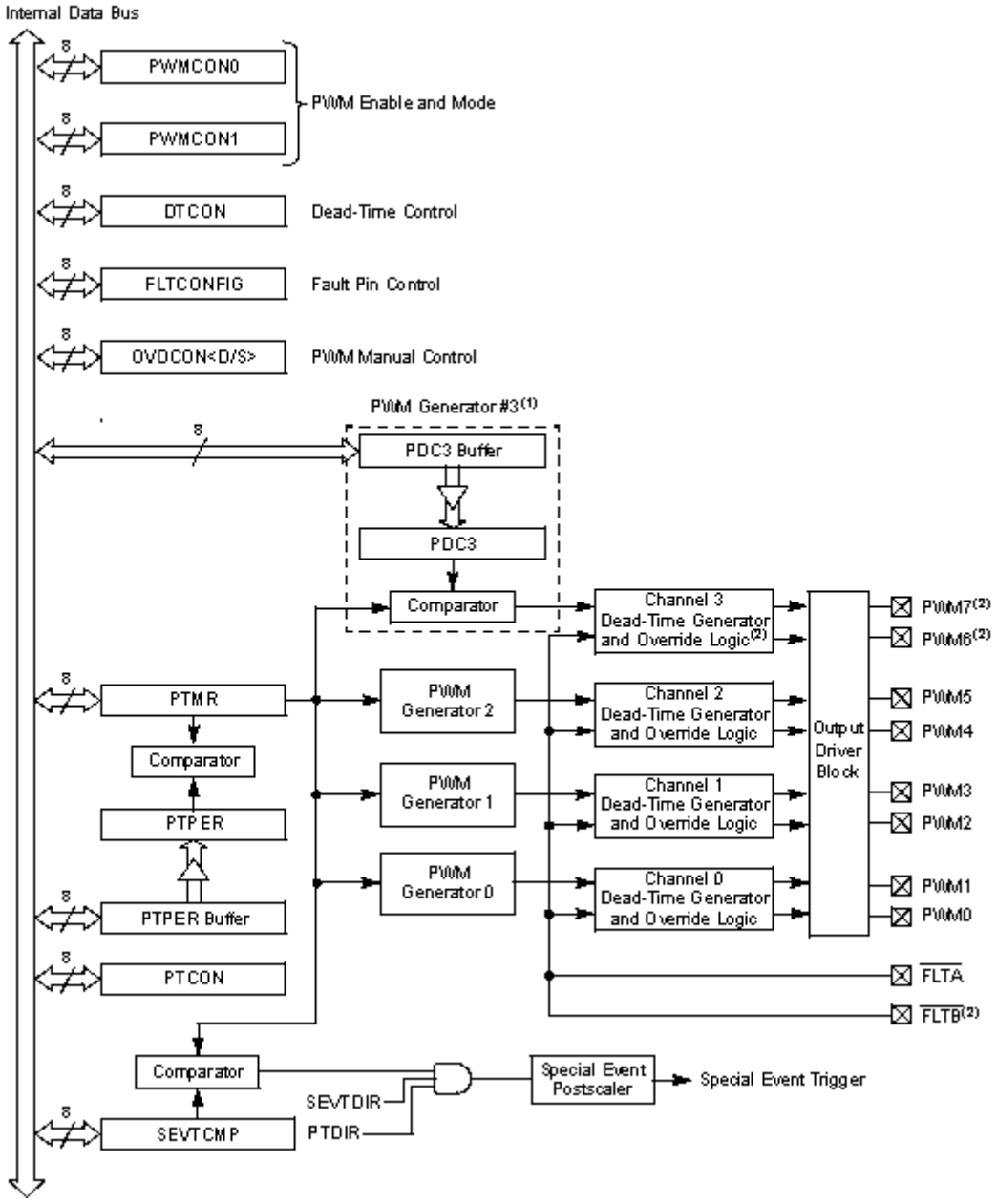


Figura 3.9 Diagrama de Bloques del módulo de potencia PWM del PIC18F2431.

En la Figura 3.10, se muestra el diagrama de bloques del funcionamiento del PWM en modo de salidas pares complementarias.

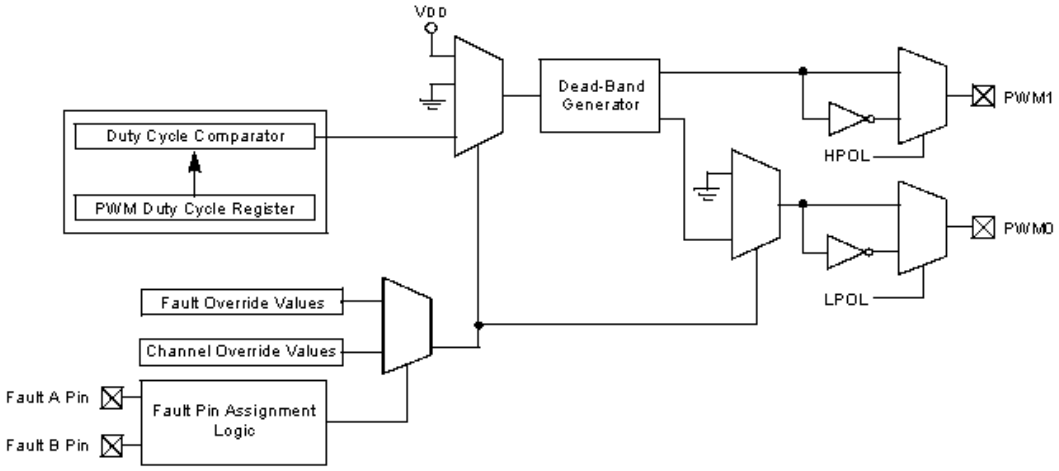


Figura 3.10 Diagrama de bloques del módulo PWM en modo de salidas pares complementarias.

Otra de las características y ventajas que ofrece el microcontrolador, en particular el módulo de control de potencia PWM, es la generación de tiempos muertos, para evitar cruces al momento de las transiciones de subida y bajada de los pulsos generados por las salidas PWM, lo que trae como consecuencia calentamientos en los drivers de potencia.

La Figura 3.11 muestra un diagrama de tiempos para la opción de generación de banda tiempos muertos.

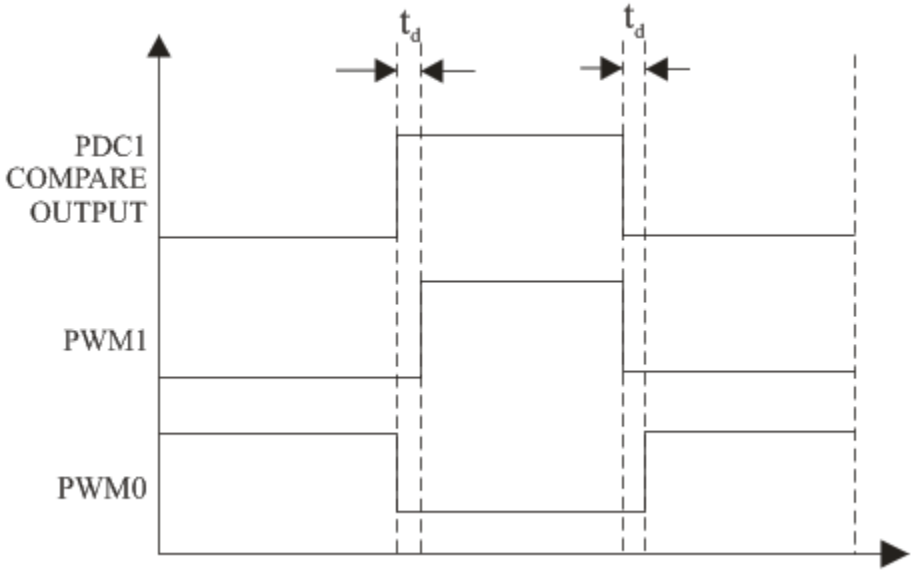


Figura 3.11 Diagrama de tiempos muertos para dos salidas PWM complementarias

3.6.2. Módulo de interfaz para la codificación de cuadratura QEI

Este es otro módulo especial para esta serie de Microcontroladores. El módulo de Interfaz para la Codificación de Cuadratura, la cual consta de 3 entradas correspondientes a los canales A, B e I del encoder óptico, Las señales generadas por el encoder son trenes de pulsos de una frecuencia que depende de la velocidad del motor.

La Figura 3.12 muestra el diagrama simplificado de la interfaz para la codificación de Cuadratura.

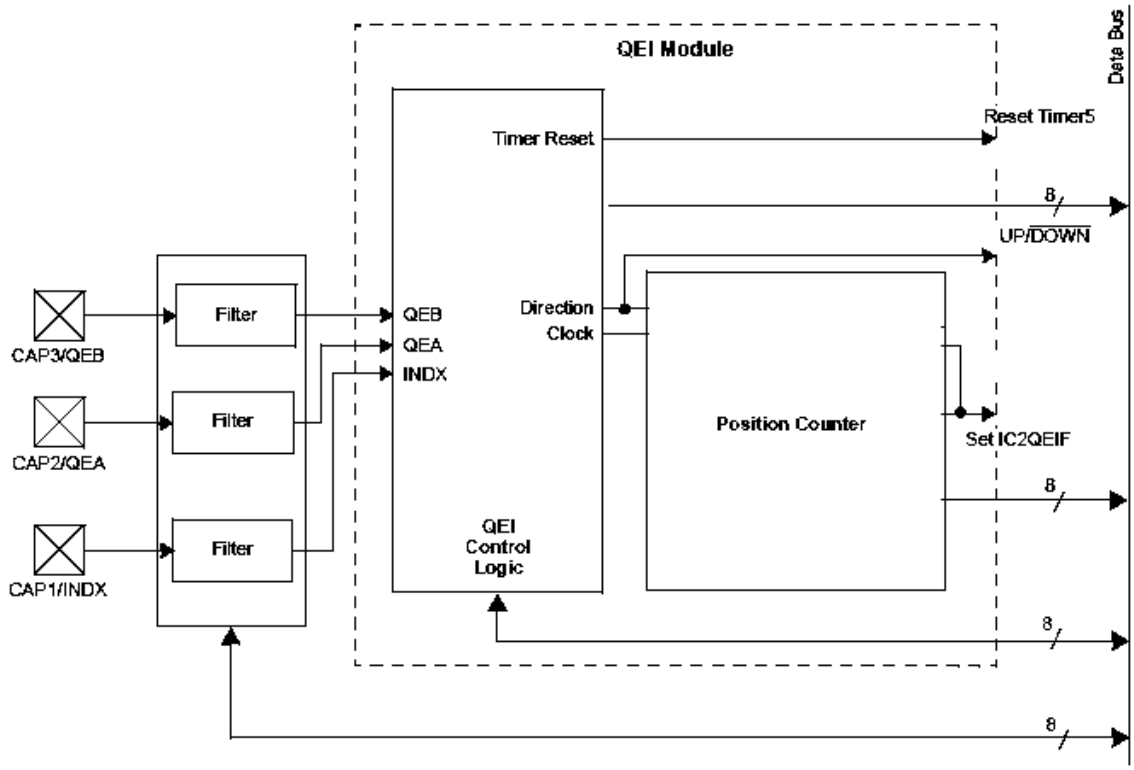


Figura 3.12 Diagrama de Bloques del módulo de Interfaz para la Codificación de Cuadratura.

Como se puede observar, otra de las grandes ventajas que ofrece este módulo es que ya tiene filtros digitales internos, que eliminan el ruido externo generado por interferencias, así como eliminan los rebotes de los pulsos generados por los canales del encoder.

Uno de los registros importantes es el contador POSCNT, de 16 bits, dividido en dos bytes POSCNTH, y POSCNTL que corresponden a la parte alta y parte baja respectivamente, del registro, este contador es el que cuenta los pulsos del encoder. Además la interfaz contiene

el bit UP/DOWN que establece el sentido de giro del motor, también posee una bandera de interrupción IC2QEIF que se activa en el momento de que la flecha del motor da una revolución.

3.7. Microcontrolador PIC18F4550

El otro microcontrolador empleado es el PIC18F4550 [Microchip, 2011] que se encarga de hacer la comunicación con la PC, cuyas características son:

32Kbytes de Memoria Flash, Máximo número de instrucciones simples: 16384, 2048 bytes de Memoria RAM, 256 bytes de memoria EEPROM, 35 pines de Entradas / Salidas, Número de entradas A/D: 13, 2 canales de Captura/Comparación/PWM, 1 canal de Captura/Comparación/PWM Mejorado, Comunicación serial síncrona SSP, SPI/I2C, Módulo de comunicación serial Asíncrona EUSART, 2 comparadores analógicos, 3 temporizadores, uno de 8 bits y 2 de 16 bits, Módulo de Comunicación Serie Universal (USB).

3.7.1. USB en PIC18F4550

Los microcontroladores PIC18F2455/2550/4455/4550 incorporan un módulo USB, el cual permite desarrollar periféricos sencillos tanto de baja velocidad (1.5 Mb/s) como de alta velocidad (12Mb/s), y que solo requieran transferencias de control y de interrupción. Permiten el uso de hasta 32 puntos finales aunque comúnmente solo utilizaremos 3 (endpoints 0, 1 y 2). Dicho módulo consta de una serie de registros que configuran y regulan su funcionamiento, así como de un SIE (Serial Interface Engine) encargado de la generación del CRC y la sincronización de las señales D+ y D-. Cada vez que se produce una transmisión o recepción de datos en el bus, se genera una interrupción en el PIC ante la cual la rutina de atención debe responder gestionando todos los aspectos de bajo nivel de la especificación USB. De esta manera para la aplicación principal que ejecuta el microcontrolador el manejo del protocolo USB es transparente.

3.7.2. Comunicación I2C

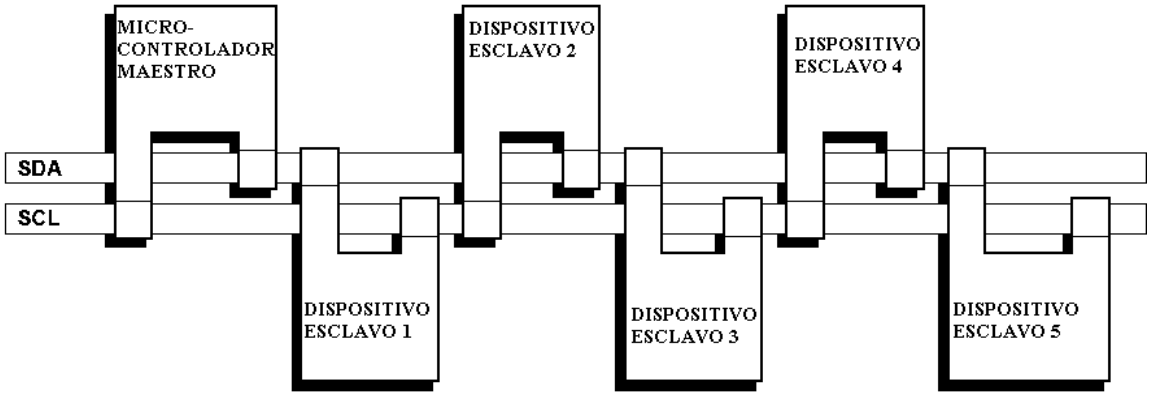


Figura 3.13 Esquema de conexión del protocolo I^2C [I2C Phillips, 2000].

Las líneas de conexión en este tipo de comunicación serial son llamadas SDA (System Data) y SCL (System Clock) éste par de líneas de conexión (llamado también bus) son bidireccionales y están conectadas al positivo de la fuente de 5 volts mediante resistencia de "pull-up" de forma que en reposo están a nivel alto. La línea SDA es donde se transmiten los datos de manera bidireccional y en la línea SCL se transmiten pulsos para sincronizar la transmisión, también es necesario que se comparta la línea de referencia (o tierra).

En el bus existen circuitos maestros (que generan la señal de SCL y controlan la comunicación) y esclavos que responden a peticiones del maestro.

El dato en SDA debe estar estable durante el periodo ALTO de reloj. SDA sólo puede cambiar mientras SCL se encuentre a nivel BAJO. La excepción a esta regla son condiciones de INICIO y PARO. La condición de INICIO se da cuando la señal de reloj se encuentra en alto mientras la señal SCL se encuentra en alto mientras se produce un cambio de la señal de SDA de un estado alto a un estado bajo. La condición de PARO se da cuando la señal de SCL se encuentra en alto mientras la señal de SDA cambia de un estado bajo a un estado alto.

En la transmisión de los datos es necesario que el maestro identifique con que otro dispositivo se está comunicando para esto cada dispositivo conectado tiene un número

identificador (o dirección) y cada que se inicia la comunicación con un dispositivo se debe enviar esta dirección, la dirección está formada por 7 bits según el dispositivo conectado.

3.8. Amplificador de potencia

El amplificador de potencia es un convertidor electrónico de CD a CD en modo de conmutación, el cual convierte el voltaje de entrada de CD no regulado a un voltaje de salida CD a CD regulado. Este amplificador es activado por interruptores controlados a una frecuencia de conmutación mucho mayor que la frecuencia de la línea. El diagrama de bloques de esta etapa se muestra en la Figura 3.14

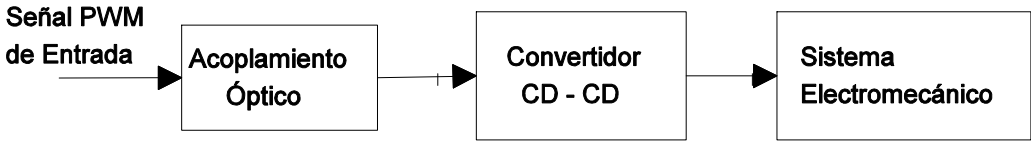


Figura 3.14 Diagrama de bloques de la etapa del amplificador de potencia.

3.8.1. Convertidor CD a CD puente completo

El convertidor de CD a CD utilizado para regular el voltaje en las terminales de la armadura del motor de CD es un convertidor de CD a CD tipo puente completo [Mohan/Undeland/Robbins, 2002], cuya topología se muestra en la Figura 3.15.

Este convertidor está compuesto por dos pares de interruptores, A y B, con su diodo en antiparalelo. Cada par de interruptores son conmutados de tal forma que cuando uno de ellos está en estado de apagado, el otro se encentra en estado de encendido. En la práctica para lograr esto, se necesita introducir un intervalo de tiempo pequeño en la señal de encendido llamado “tiempo de blanqueo”, “tiempo de retardo” o “tiempo muerto”, con el fin de evitar cortocircuitos en la entrada de alimentación del convertidor de CD.

- PWM con conmutación de voltaje bipolar, donde los pares de interruptores (T_{A+}, T_{B-}) y (T_{A-}, T_{B+}) son tratados de manera que siempre hay un par en estado de conducción, mientras que el otro par esta en abierto.

- PWM con conmutación de voltaje unipolar, donde el pares de interruptores (T_{A+}, T_{A-}) es controlado de manera independiente que el par (T_{B+}, T_{B-}).

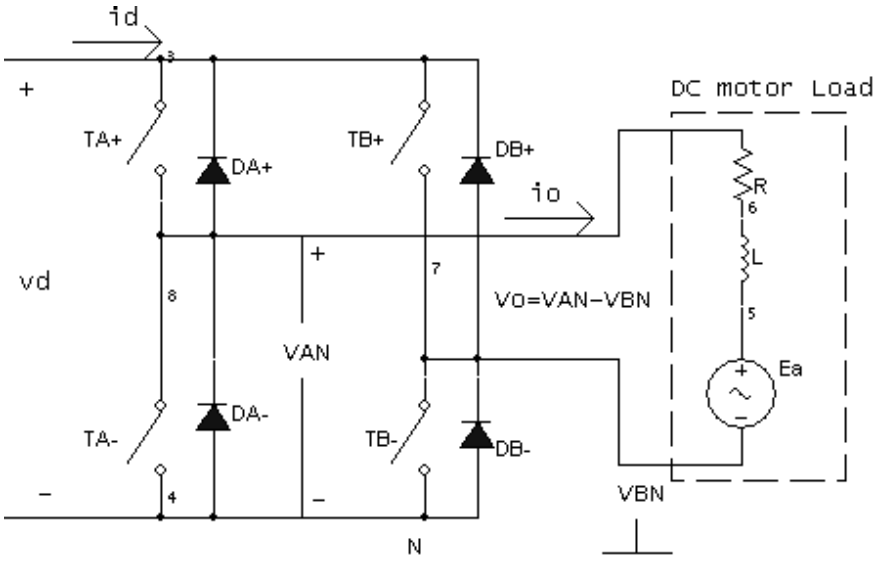


Figura 3.15. Topología del convertidor de CD a CD tipo puente completo.

El esquema de control utilizado es el de conmutación de voltaje bipolar, en el cual, la señal de control generada por el módulo de potencia PWM del microcontrolador es aplicada directamente al par de interruptores, mientras que para controlar el otro par, se utiliza la misma señal, pero complementaria, es decir, invertida, el esquema produce la siguiente salida mostrada en la Figura 3.16.

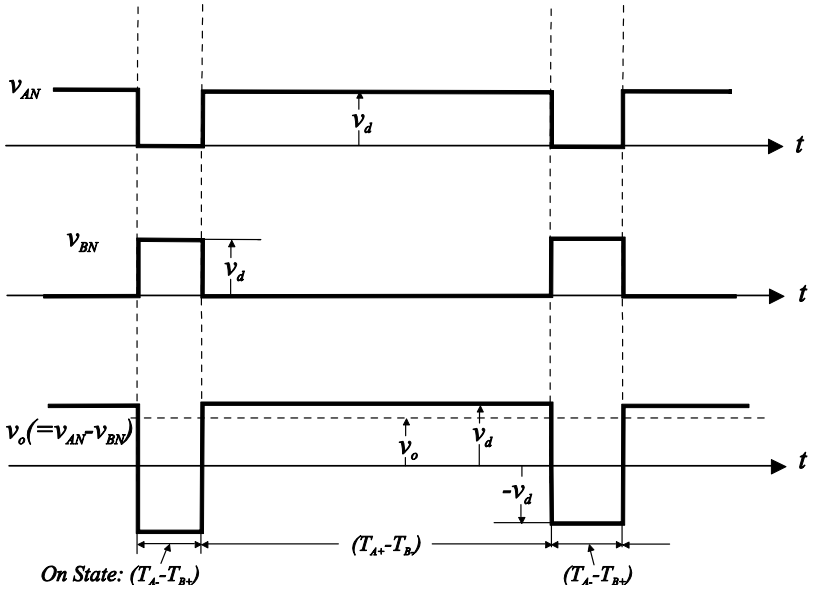


Figura 3.16 Esquema de control del PWM con conmutación en modo bipolar.

Se eligieron MOSFET'S de potencia IRF740 [Fairchild, 2011], cuyas características son las siguientes:

- Diodo antiparalelo integrado.
- Corriente Máxima I_D=10 Amp.
- Corriente pulsante Máxima I_{DM}=10 Amp.
- Voltaje en corte V_{DSS}=400 V.
- Tiempo de encendido $t_{d(ON)}$=35 ns.
- Tiempo de apagado $t_{d(OFF)}$=90 ns.
- Potencia de disipación Máxima P_D=125 Watts.
- Resistencia entre Gatillo y compuerta R_{GS}=1 MΩ.

3.9. Circuitos de acoplamiento óptico

Usualmente cuando se controlan dispositivos de voltajes elevados mediante Microcontroladores, hay el riesgo de dañar los equipos de control, para evitar este tipo de detalles es por medio de aislamiento óptico entre las etapas de control y de potencia, esto es, aislar las tierras de la circuitería lógica, y la circuitería de potencia. El transformador puede cumplir exactamente con esta función.

Para nuestro caso, utilizamos acoplamiento óptico entre las etapas de control y de potencia, de este modo, la única conexión entre los dos módulos, será la luz del diodo incidiendo en un detector con salida Schmitt trigger.

3.9.1. Optoacopladores

El optoacoplador utilizado es el NTE3090 [NTE, 2002], el cual es un circuito integrado que presenta las siguientes características:

- Tiempos de conmutación t_{ON}, t_{OFF} menores a 4 µs.
- Voltaje inverso del LED emisor V_R=6 V.

- Corriente máxima en el LED I_f=60 mA.
- Corriente de salida I_O=50 mA.
- Voltaje de aislamiento Máximo V_{ISO}=7500 V.
- Potencia máxima de disipación en el led P_D=250mW.

La Figura 3.17 muestra la configuración de los optoacopladores para implementar la interfaz con el acoplamiento óptico entre los circuitos de control y los circuitos de potencia.

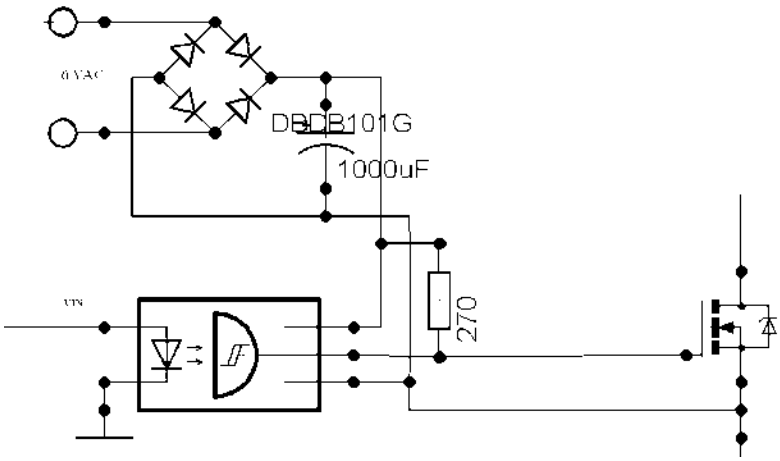


Figura 3.17 Optoacoplador de salida Disparador Schmitt.

Como se observa en la Figura 3.17, se incluye una compuerta NOT a la entrada del optoacoplador, esto es para evitar la negación de la señal de salida del optoacoplador, a la salida, y no tener problemas con los tiempos muertos.

3.10. Sensores

Los controladores en lazo cerrado necesitan medir el parámetro que van a controlar, para este caso, se necesitan medir velocidad de la flecha del motor de CD, así como su posición, para ello se emplea un encoder acoplado a la flecha. Adicionalmente, para fijar la referencia cero de posición, se utilizan sensores magnéticos que mandan la señal de referencia cero.

3.10.1. Encoder óptico

El encoder óptico se encarga de medir la posición y la velocidad del motor de CD, para ello se utiliza el de la serie HEDS-5310 [Hewlett Packard, 1984], este dispositivo es un encoder de alta resolución, de fácil armado y alta confiabilidad que consta de 3 partes. El encapsulado, un disco metálico de 28mm de diámetro, con 256 ranuras alrededor del disco, y un módulo del emisor de luz a través del disco ranurado, y una placa de fase hacia los lentes del detector. La luz es enfocada en un par de detectores (uno por cada canal), que generan señales de salida en forma de onda cuadrada , defasadas un cuarto de ciclo, y un pulso opcional llamado índice, el cual genera un pulso por revolución de la flecha del motor. La alineación de la luz, y la configuración del fotodetector incrementan la confiablidad, afectada por la reducción de la sensibilidad de las ranuras del disco y la degradación de los LED's.

Las señales de salida y la fuente de alimentación del encoder son conectadas a través de un conector de 10 terminales en cable múltiple de 60 cm de longitud.

El disco ranurado se montó en la flecha del motor de CD como se muestra en la Figura 3.18.

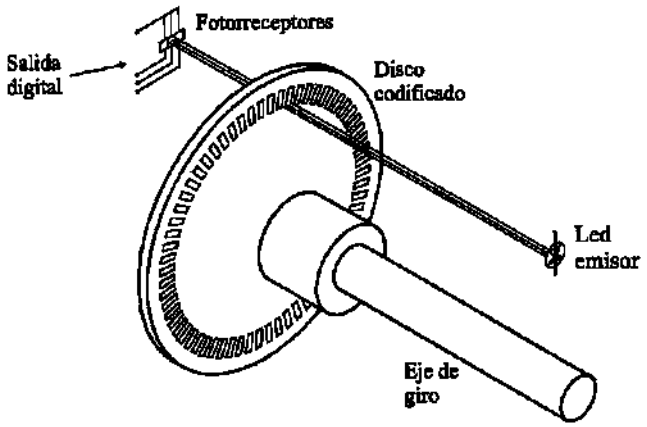


Figura 3.18 Montaje del encoder sobre la flecha del motor de CD.

La operación del sensor está basada en la generación de un tren de pulsos determinado por la interrupción de la luz entre el LED y el fotodetector. El diagrama del encoder se muestra

en la Figura 3.19.Los trenes de pulsos generados por el encoder son enviados a los canales QEA, QEI e INDX del microcontrolador.

La señal índice se obtiene a partir de un pulso que se genera una vez por cada vuelta de la flecha del motor.

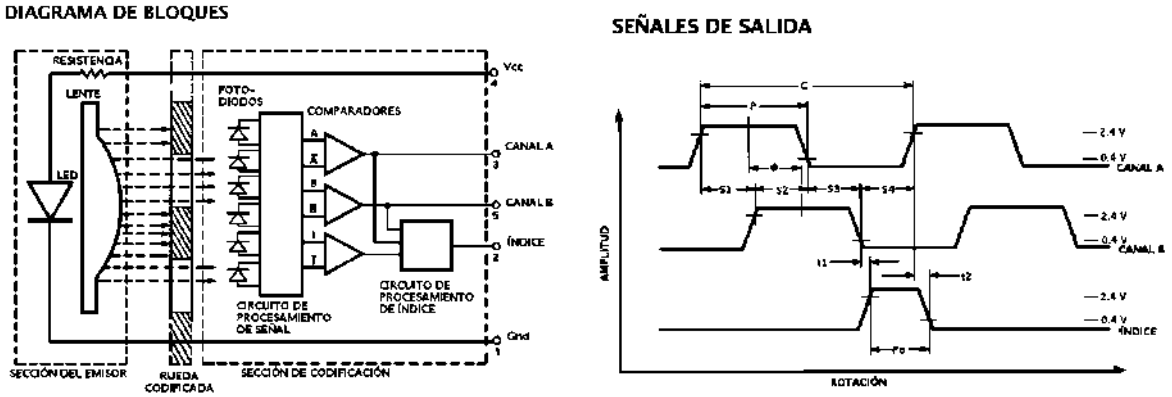


Figura 3.19. Diagrama esquemático del HEDS-5310 y señales producidas por el encoder.

3.10.2. Sensor magnético

Los sensores utilizados para medir las posiciones de referencia X=0, Y=0 son sensores magnéticos tipo reed switch de 1 ampere de capacidad y el cual se muestra en la Figura 3.20. Para el manipulador, estos interruptores magnéticos, están colocados en los orígenes de los respectivos ejes del movimiento. En cada servomecanismo, hay una lámina de imán que es la que activa el sensor magnético cuando ya está en la posición de origen, la salida digital del interruptor magnético es enviada a una de las entradas del microcontrolador.

La Figura 3.21 muestra el circuito driver implementado para cada servomecanismo, así como el disparador de la herramienta basado en un TRIAC en la figura 3.22 y el sistema completo en la figura 3.23. Los diagramas esquemáticos correspondientes a dicho circuito se pueden encontrar en el Apéndice B.

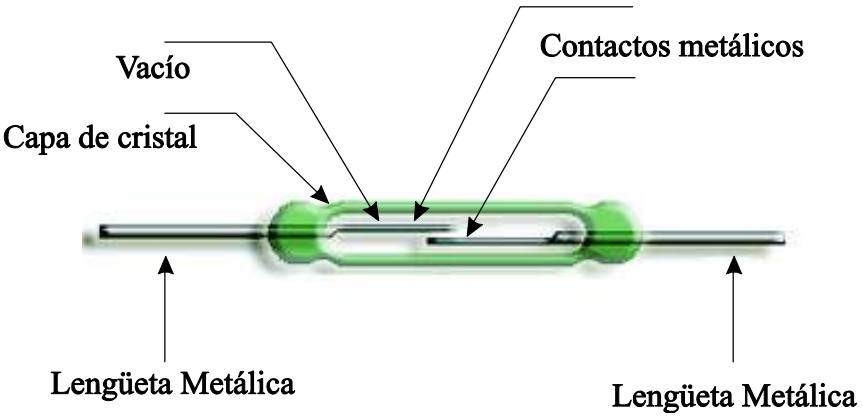


Figura 3.20 Sensor magnético.

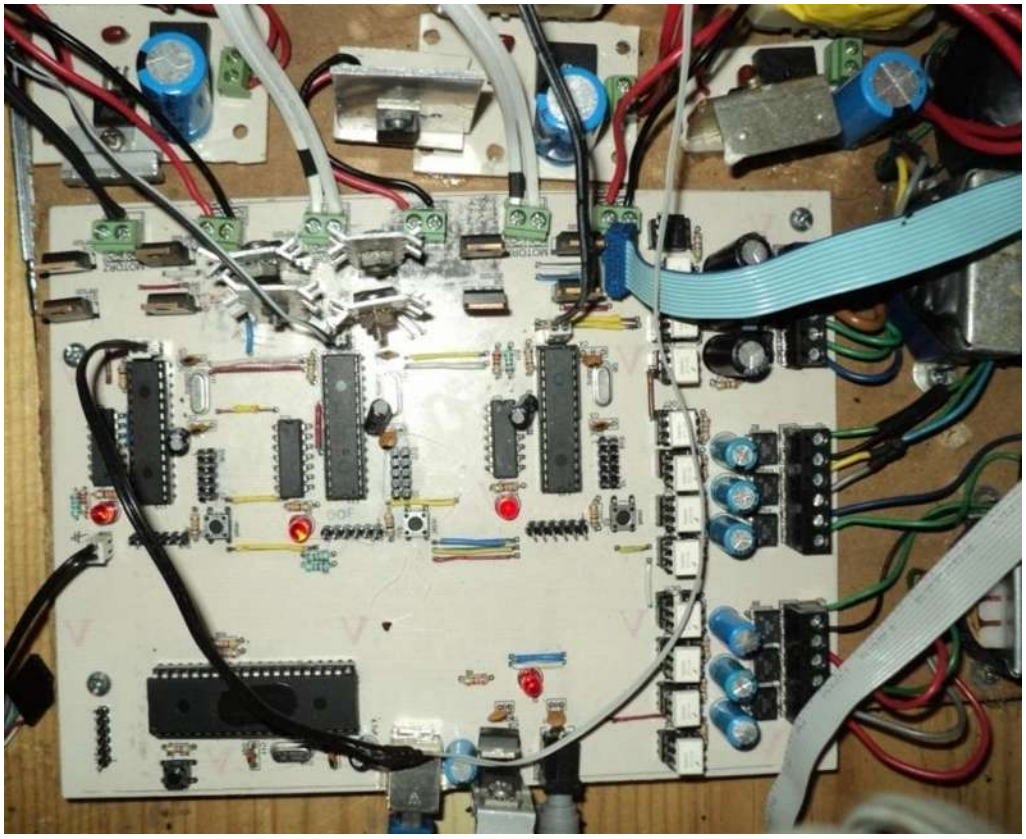


Figura 3.21 Circuito driver para el sistema manipulador.

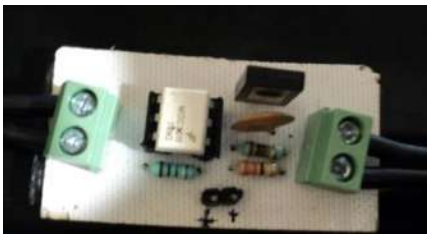


Figura 3.22 Circuito driver para el disparo de la herramienta.

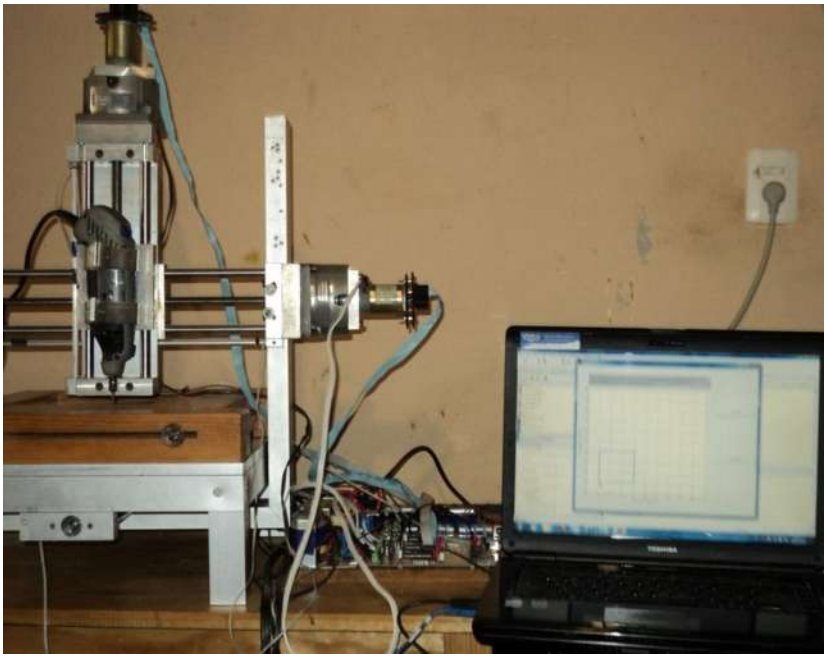


Figura 3.23 Sistema completo del manipulador.

En este capítulo se presentaron los componentes físicos del manipulador, el hardware y la circuitería con la que trabaja, se dio una breve descripción de cada componente a utilizar y de la estructura de los acoplamientos que tiene el manipulador.

En el siguiente capítulo se presentan los algoritmos tanto de medición y control de velocidad y posición de los motores de CD, así como también se presenta todo el software tanto de los microcontroladores como de la interfaz para convertir las coordenadas X, Y, Z a pulsos con los que puedan trabajar los Microcontroladores esclavos para establecer la posición y velocidad a la que deben moverse.

Capítulo 4

Algoritmos de medición y control de posición y velocidad y software del manipulador.

4.1. Introducción

En el presente capítulo se presenta la descripción de los métodos de medición de posición y velocidad, así como también se describen los algoritmos de control de posición y velocidad de los motores de CD para el movimiento de los ejes a través de los servomecanismos.

4.2. Medición de posición y velocidad

Las variables fundamentales que se necesitan para controlar el manipulador son posición y velocidad, dichas variables se toman mediante el sensor óptico descrito en la sección 3.10.1, en seguida se muestran los métodos de medición de dichas variables.

4.2.1. Medición de posición

La posición de la flecha del motor se determina mediante la interfaz de codificación de cuadratura QEI del PIC18F2431, a través de los trenes de pulsos que entregan los sensores ópticos acoplados a la flecha del motor, dichos trenes de pulsos son leídos por la interfaz QEI y por cada pulso, aumenta un contador dentro del microcontrolador. De esta manera, se pueden detectar 1024 pulsos por vuelta con un encoder de 256 ranuras, debido a que la interfaz QEI aumenta la cuenta del contador tanto en las subidas como en las bajadas de los pulsos, de esta manera, como las entradas QEA y QEB están desfasadas, así, se pueden leer cuatro pulsos por cada periodo del tren de pulsos, teniendo hasta 1024 pulsos por cada revolución del motor.

La entrada INDX genera una interrupción por cada vuelta del motor para hacer un reajuste de los pulsos generados por cada revolución, y evitar errores por sobreflujo en la cuenta de la variable contador. Se usa una entrada adicional RB4 para detectar la posición cero de la señal producida por el sensor magnético. En la Figura 4.1 se muestran estas señales, y las transiciones que aumentan el contador.

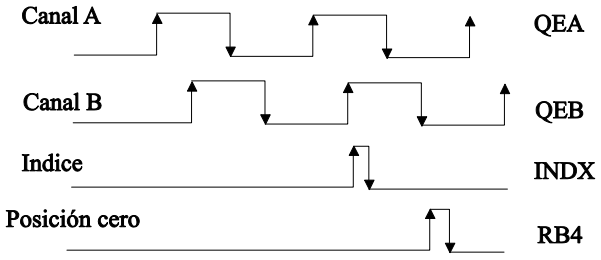


Figura 4.1 Señales para medición de posición.

Para la detección de la dirección de giro del motor con el encoder utilizado, se utiliza una bandera de lectura incluida en la interfaz QEI (UP/DOWN), si ese bit esta en alto, el sentido es en dirección de las agujas del reloj, si está en bajo, el sentido de giro es al contrario de las agujas del reloj.

Recordando que el encoder se encuentra montado en la flecha del rotor y tomando en cuenta que el motor está acoplado a la carga por medio del tornillo, (con un paso de tornillo de S=0.3 cm/rev) y una caja de engranes reductora de velocidad, (1:75) como se muestra en la Figura 4.2, se plantean las siguientes relaciones:

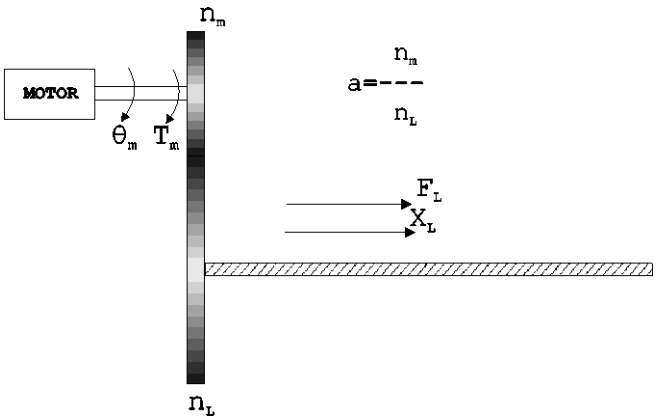


Figura 4.2 Acoplamiento entre el motor y el tornillo.

De la Figura 4.2, observamos que si X_L es el desplazamiento lineal se tiene que:

$$número\ de\ vueltas = \frac{X_L}{S}\left[\frac{cm}{cm/rev}\right] \tag{4.1}$$

Y si Θ_L es el desplazamiento rotacional en el tornillo, tendremos también que:

$$número\ de\ vueltas = \frac{\theta_L}{2\pi}\left[\frac{rad}{rad/vuelta}\right] \tag{4.2}$$

Igualando (4.1) y (4.2)

$$\theta_L = \frac{X_L 2\pi}{S} \tag{4.3}$$

Y dado que

$$\theta_m = \frac{\theta_L}{a} \tag{4.4}$$

Se obtiene que la posición angular del motor en función del desplazamiento lineal del manipulador está dada por:

$$\theta_m = \frac{2\pi}{a}\frac{X_L}{S} \tag{4.5}$$

Donde:

S= desplazamiento del tornillo por cada vuelta del rotor (paso del tornillo [cm/rev])

X_L= Distancia recorrida del tornillo [cm]

Θ_m=posición angular del eje del motor [rad]

Θ_L=posición angular del eje de la carga [rad]

a=relación de reducción de los engranes [adimensional]

4.2.2. Medición de velocidad

La velocidad se calcula también con base en los trenes de pulsos entregados por el encoder, tomando en cuenta adicionalmente el instante en el que estos pulsos son capturados por la interfaz QEI del microcontrolador.

El diagrama de pulsos de la Figura 4.3 ayuda a comprender la forma de medición de velocidad. Cuando el sentido es sentido horario, la cuenta POSCNT aumenta en uno por cada transición de los canales A y B. Por cada periodo de muestreo, se lee la cantidad de pulsos actuales, y se resta la cantidad de pulsos leídos del periodo de muestreo anterior, y esa es la velocidad medida. En caso de que el rotor este girando en sentido contrario, las variables que llevan la cuenta actual y la cuenta anterior, se intercambian de valor, para así evitar números negativos a la hora de hacer la resta, como lo muestran las ecuaciones (4.6) y (4.7).

Para cuando UP/DOWN=1:

$$\omega = \frac{(número\ de\ \ pulsos\ \ actual) - (número\ de\ \ pulsos\ \ anterior)}{T_s} \left[\frac{pulsos}{seg} \right] \qquad (4.6)$$

Para cuando UP/DOWN=0:

$$\omega = \frac{(número\ de\ \ pulsos\ \ anterior) - (número\ de\ \ pulsos\ \ actual)}{T_s} \left[\frac{pulsos}{seg} \right] \qquad (4.7)$$

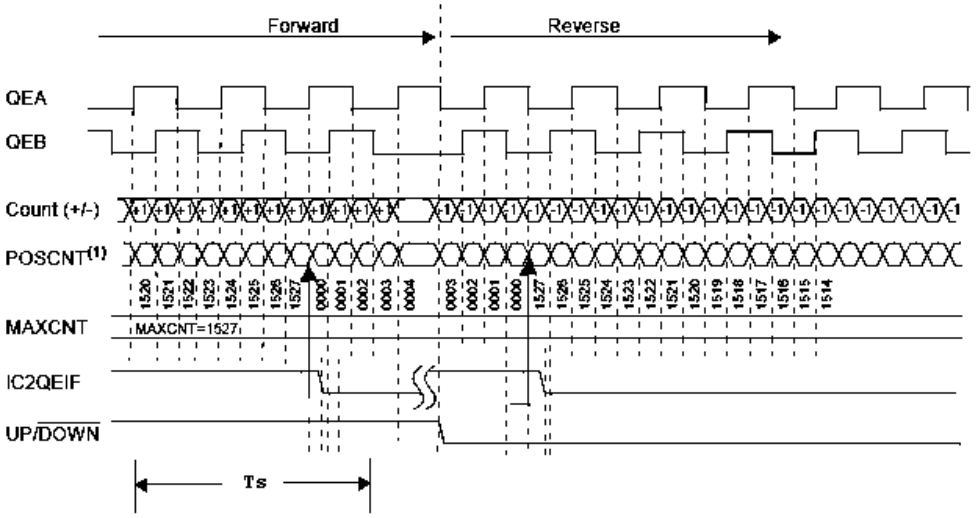


Figura 4.3 Diagrama de pulsos de velocidad y posición

4.3. Control de posición y velocidad

Desde el punto de vista de control, el movimiento de un brazo robótico se suele efectuar en dos fases de control. La primera es el control de movimiento durante el cual el brazo se mueve desde una posición u orientación inicial, hasta una posición u orientación final deseada a lo largo de una trayectoria planificada. La segunda fase es el control del movimiento del efector final del brazo que interacciona dinámicamente utilizando la información de retroalimentación de los sensores para complementar la tarea. Esto requiere un control de velocidad para suavizar la trayectoria planificada para la herramienta terminal.

4.3.1. Algoritmo de control de posición

En la Figura 4.4 se muestra el diagrama de bloques del sistema de de control de posición para cada servomecanismo del manipulador.

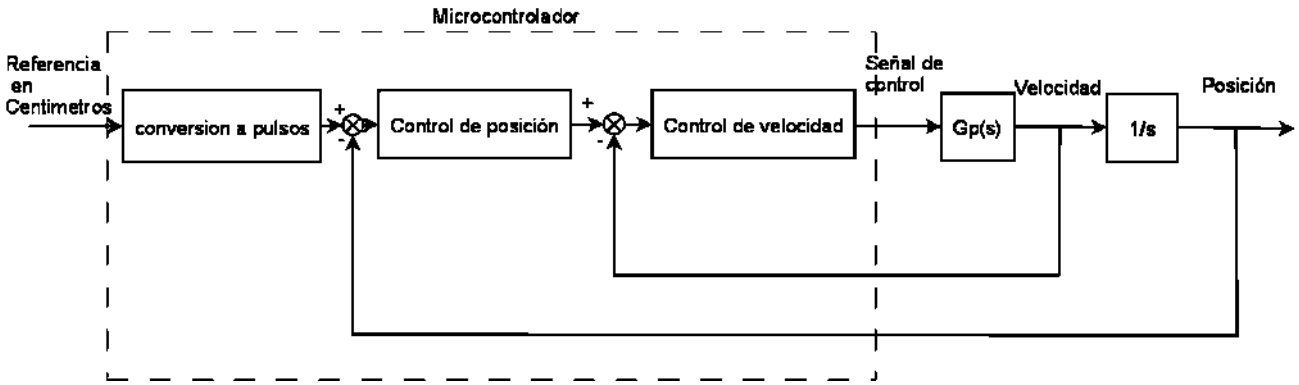


Figura 4.4 Diagrama de bloques del sistema de posición

Se puede apreciar que la parte dentro de la línea punteada es la que el microcontrolador va a realizar y que se trata de la parte digital del control. El bloque Gp(s) es la planta que incluye la etapa de aislamiento, la etapa de potencia, el motor y la reducción por los engranes.

El algoritmo de control de posición propuesto se basa en la idea de Takashi Kenjo de su libro Power Electronics [Kenjo, 1994], en done se propone una tabla constante que relaciona la velocidad con el error de posición.

La posición y velocidad de la flecha del rotor están relacionadas por la ecuación (4.8):

$$\theta = \int \omega \, dt \tag{4.8}$$

Esto implica que para controlar la posición de requiere controlar la velocidad y la relación entre el par *Tm* proporcionado por el motor y la velocidad ω del rotor está dada por la ecuación dinámica Newtoniana:

$$(J_m + J_l)\frac{d\omega}{dt} + D\omega = Tm \tag{4.9}$$

Donde:

J_m = momento de inercia del motor.

J_l = momento de inercia de la carga.

D = coeficiente de amortiguamiento viscoso.

Para la acción de control de la posición se plantean tres etapas:

1) Aceleración: El motor es acelerado por un par constante Tm generado por la máxima corriente permisible. En este caso la solución a la ecuación de la ecuación (4.9) está dada por:

$$\omega = \frac{Tm}{D}\left[1 - \exp\left(\frac{-Dt}{J_m + J_l}\right)\right] \tag{4.10}$$

La relación de ω con respecto al tiempo se muestra en la Figura 4.5.

2) Velocidad constante: Si la derivada de la ecuación (4.9) es cero, la solución a la ecuación es:

$$\omega = \omega_{CS} = \frac{Tm}{D} \tag{4.11}$$

Donde: ω_{CS} es la velocidad que se mantiene constante, en la fase anterior

1) Desaceleración El motor es desacelerado aplicando una corriente negativa que proporciona un par negativo. Si se logra el término D_ω la ecuación dinámica (4.9) puede ser cambiar a:

$$(J_m + J_l)\frac{d\omega}{dt} = -Tm \tag{4.12}$$

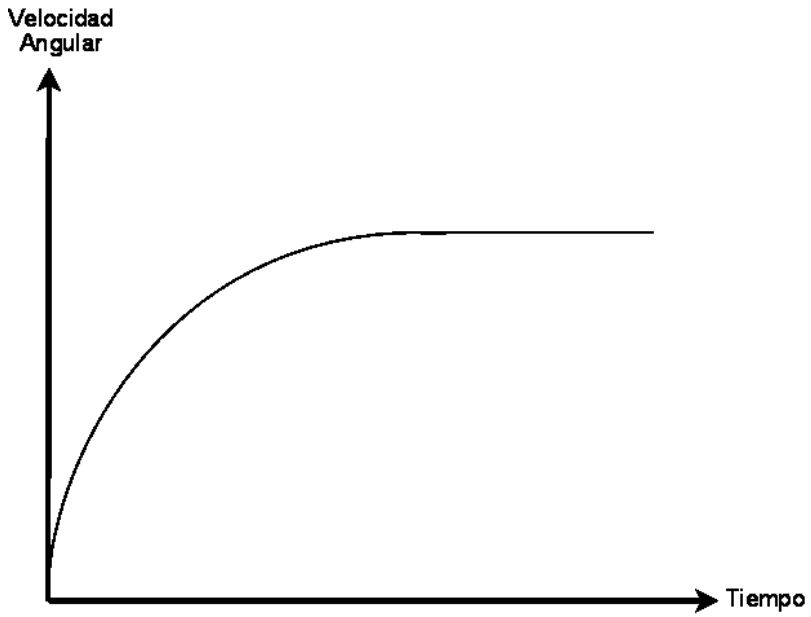


Figura 4.5 Curva de aceleración.

La solución a la ecuación es:

$$\omega = \omega_{CS} - \frac{Tm}{J_m + J_l} t \tag{4.13}$$

De esta manera el perfil de velocidad para el control de posición esta mostrado en la Figura 4.6. Este perfil está dividido en cinco regiones: aceleración, velocidad constante, desaceleración, ajuste y paro.

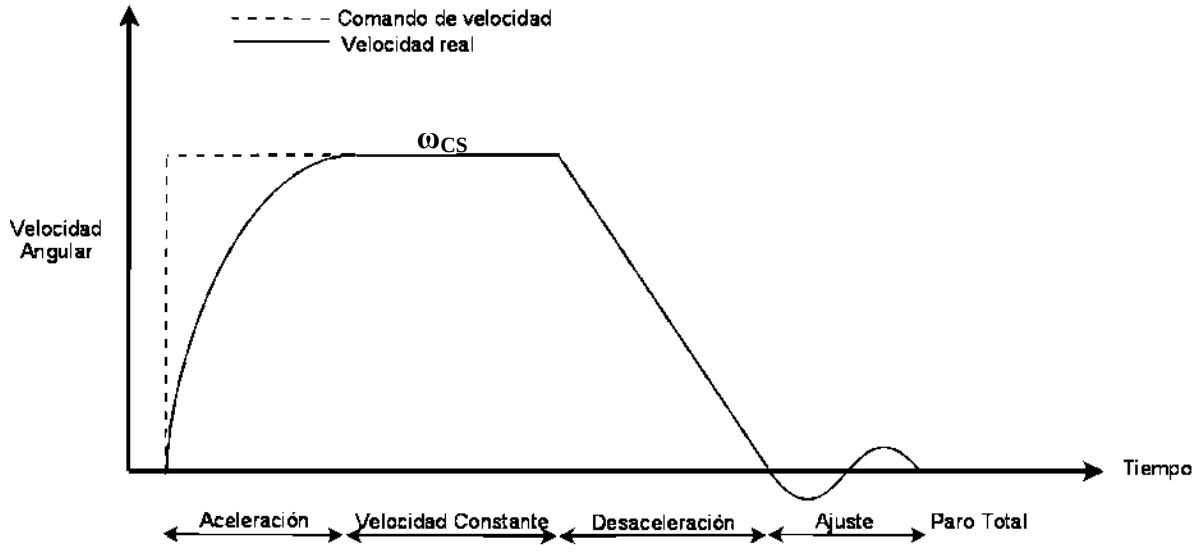


Figura 4.6 Perfil de velocidad

Aceleración: el contador de error (error de posición) ordena un valor máximo de velocidad hacia el cual el rotor es acelerado de acuerdo a la dinámica del motor y el controlador de velocidad empleado.

Velocidad constante: Cuando el rotor alcanza la velocidad máxima deseada se mantiene constante esta velocidad, por medio del controlador de velocidad.

Desaceleración: Para frenar al rotor con aceleración constante, el comando se debe dar de acuerdo con la siguiente ecuación:

$$\omega_C = \sqrt{(2q\beta)} \tag{4.14}$$

Donde: q = error de posición (la diferencia al valor deseado de posición dado en pulsos del encoder montado en la flecha del motor).

β = aceleración negativa o razón de desaceleración en pulsos/seg^2.

Ajuste: Idealmente el motor se debería parar en la posición deseada y no debería viajar más allá de esa posición. En la práctica, sin embargo, existe la posibilidad de que el rotor vaya más allá de la posición deseada, debido a que los comandos de velocidad no son dados en forma de función lineal ideal, sino en forma discontinua debido a la cuantización digital. El sistema de control se diseña para que el rotor eventual se detenga en el objetivo después de varios ciclos de oscilación alrededor de la referencia.

Paro: Se alcanza cuando el rotor se detiene en la posición deseada.

Para lograr el perfil de velocidad deseado, mostrado en la Figura 4.6, el microcontrolador genera los comandos de la velocidad que dependen del error de posición, de acuerdo a ecuación (4.14). Así estos comandos se calculan de acuerdo a las reglas, las cuales se muestran gráficamente en la Figura 4.7.

i) si $$0 < q < q_f \rightarrow \omega_{ref} = \sqrt{2\beta_{\max} q} \tag{4.15}$$

ii) si $$-q_f < q < 0 \rightarrow \omega_{ref} = \sqrt{2\beta_{\max} q} \tag{4.16}$$

iii) si $$q_f < q < q_{\max} \rightarrow \omega_{ref} = \omega_{\max} \tag{4.17}$$

iv) si $$-q_{\max} < q < -q_f \rightarrow \omega_{ref} = -\omega_{\max} \tag{4.18}$$

Donde:

q_f = valor del error a partir del cual comienza la región de frenado (este valor define indirectamente el tiempo de frenado).

ω_{ref} = valor deseado de velocidad.

B_{max} = valor digital que representa la desaceleración máxima utilizada.

q_{max} = error de posición máximo.

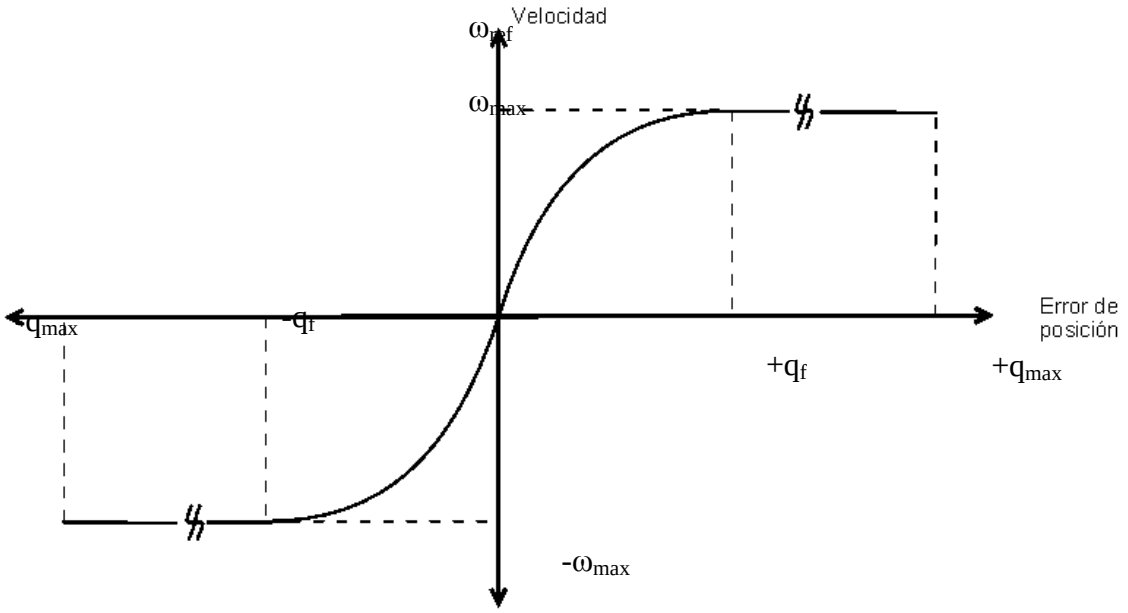


Figura 4.7 Relación de error de posición y los comandos de velocidad.

4.3.2. Algoritmo de control de velocidad

Para el controlador de velocidad se implementó un control IP Digital. El lazo de control de velocidad se muestra en la Figura 4.8 en su forma analógica [Ogata, 1996]. Nótese que sólo el término de control integral incluye la entrada de referencia $\omega_{ref.}$

La función de transferencia de lazo cerrado de la Figura 4.8 está dada por la ecuación (4.19).

$$\frac{\omega(s)}{\omega_{ref}(s)} = \frac{\frac{K_i}{s}G_p(s)}{1+(K_p+\frac{K_i}{s})G_p(s)} \qquad (4.19)$$

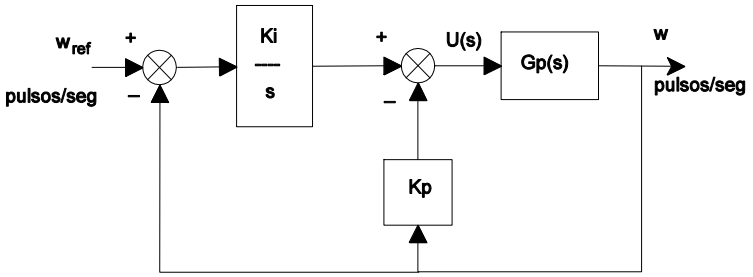


Figura 4.8 Lazo de control de velocidad.

Donde:

K_i = Ganancia integral analógica

K_p = Ganancia proporcional analógica

ω_{ref} = Velocidad de referencia

ω = Velocidad de salida

Una ventaja de utilizar un esquema de control IP es que elimina automáticamente la saturación del término integral, porque la integración para automáticamente cuando la salida está limitada (No hay necesidad del antiwindup). Otra ventaja es que es útil en la supresión de correcciones excesivas en sistemas de control de procesos, esto es, se pueden evitar grandes cambios de señales de control debido a las acciones de control proporcional cuando hay un cambio súbito en la entrada, la idea básica del controlador IP es evitar grandes señales de control (que pueden producir el fenómeno de saturación) dentro del sistema.

El diagrama de bloques completo, que incluye el control de posición y velocidad es el mostrado en la Figura 4.9.

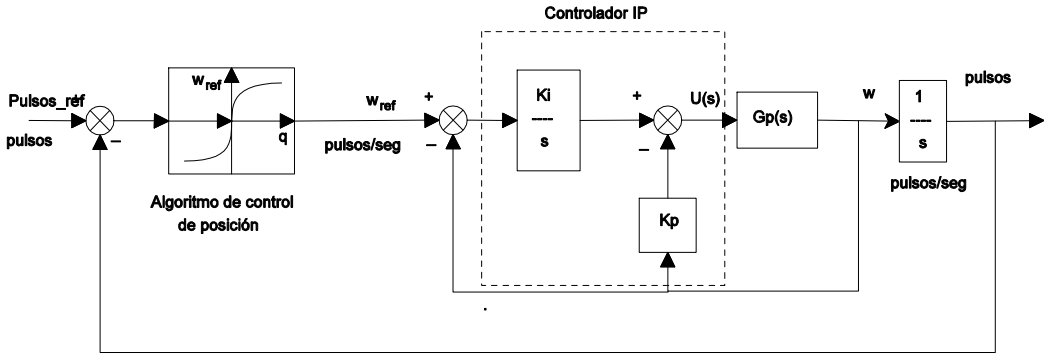


Figura 4.9 Diagrama de bloques del sistema de control de posición para cada servomecanismo.

4.3.3. Controlador IP digital

Para obtener la ecuación del control IP en tiempo discreto, se considera la diferencia hacia atrás en la ecuación de control *u(kT),* esto es la diferencia entre la entrada de control actual y la entrada de control anterior *u(kT)* y *u((k-1)T),* como se muestra en la ecuación (4.20).

$$\Delta u(kT) = u(kT) - u((k-1)T) \tag{4.20}$$

La acción de control IP en controladores analógicos es:

$$u(kT) = Kp\left[e(t) + \int_0^t e(t)dt \right] \tag{4.21}$$

Donde:

e(t) es la entrada al controlador (señal de error actuante).

u(t) es la salida del controlador (señal de control)

Kp es la ganancia proporcional analógica

Ti es el tiempo integral (o tiempo de reajuste)

Para obtener la función de transferencia pulso del controlador IP digital se pude discretizar la ecuación (4.21), al aproximar el término integral mediante la sumatoria trapezoidal, se obtiene:

$$u(kT) = Kp\left[e(kT) + \frac{T}{T_i}\left(\frac{e(0)+e(T)}{2} + \frac{e(T)+e(2T)}{2} + \ldots + \frac{e((k-1)T)+e(kT)}{2} \right) \right] \tag{4.22}$$

Sustituyendo la ecuación (4.22) en la ecuación (4.26), se obtiene:

$$\Delta u(kT) = \left(Kp - \frac{KpT}{2T_i} \right)\left[e(kT) - e((k-1)T) \right] + \frac{KpT}{T_i} e(kT)$$

$$\Delta u(kT) = K_{pd}\left[e(kT) - e((k-1)T) \right] + K_{id} e(kT) \tag{4.22}$$

Donde se han empleado las siguientes relaciones para las ganancias digitales:

$$K_{id} = \frac{K_p T}{T_i} = K_i T \tag{4.23}$$

$$\Delta u(kT) = K_{pd} = \left(K_p - \frac{K_p T}{2T_i} \right) = K_p - \frac{1}{2} K_{id} \tag{4.24}$$

$$e(kT) = \omega_{ref}(kT) - \omega(kT) \tag{4.25}$$

Al sustituir la ecuación (4.25) en la ecuación (4.22) se obtiene:

$$\Delta u(kT) = K_{pd}\left(\omega_{ref}(kT) - \omega_{ref}((k-1)T) - \omega(kT) + \omega((k-1)T)\right)$$

$$+ K_{id}\left(\omega_{ref}(kT) - \omega(kT)\right) \tag{4.26}$$

Este esquema de control PI en forma de velocidad dado por la ecuación (4.26) se puede modificar para hacer frente a grandes cambios súbitos en el punto de ajuste. Puesto que la acción de control proporcional produce grandes cambios en la salida del controlador cuando la señal que entra a éste presenta un cambio súbito grande, para suprimir dichos cambios en la salida del controlador el término proporcional se puede modificar haciendo que la acción proporcional no depende de la entrada ω_{ref}, sino únicamente de la salida ω.

$$u(kT) = -K_{pd}\left(\omega(kT) - \omega((k-1)T)\right) + K_{id}\left(e(kT) + u((k-1)T)\right) \tag{4.27}$$

Esta ecuación es la ecuación del controlador de velocidad, y es la implementada dentro del microcontrolador. Aplicando la transformada Z a la ecuación (4.27), nos da como resultado

$$(1 - z^{-1})U(z) = -K_{pd}(1 - z^{-1})\omega(z) + K_{id}(\omega_{ref}(z) - \omega(z))$$

$$U(z) = -K_{pd}\omega(z) + K_{id}\frac{\omega_{ref}(z) - \omega(z)}{(1 - z^{-1})} \tag{4.28}$$

La ecuación (4.28) nos da el esquema de control PI en forma de velocidad, donde:

$$K_{pd} = K_p - \frac{K_{id}}{2} = \text{Ganancia proporcional digital} \tag{4.29}$$

$$K_{id} = \frac{K_p T}{T_i} = K_i T = \text{Ganancia integral digital} \tag{4.30}$$

El diagrama de bloques de la Figura 4.8 muestra la realización de un esquema de control PI digital en forma de velocidad.

La función de transferencia en lazo cerrado está dado por

$$\frac{\omega(z)}{\omega_{ref}(z)} = \frac{K_{id}G_p(z)}{(1-z^{-1})(1+G_p(z)K_{pd}) + K_{id}G_p(z)} \tag{4.31}$$

Para encontrar las ganancias del controlador IP se requiere primeramente conocer el modelo matemático de $G_p(s)$. En seguida se describe el modelo de $G_p(s)$ utilizando el método empleado para hacer su estimación.

4.3.3.1. Obtención del modelo del sistema.

Tomando en cuenta que el controlador utiliza una alta frecuencia de conmutación a 10 KHz, se puede utilizar una aproximación del conjunto: Convertidor de CD a CD, motor de CD y sistema mecánico. La función de transferencia de un motor de CD, con una entrada de voltaje y una salida de velocidad, se puede aproximar mediante un modelo de segundo orden, sin embargo, dado que la constante de tiempo eléctrica del motor es mucho menor que su constante mecánica, se puede utilizar la aproximación de primer orden para diseñar el controlador de velocidad. Así, el conjunto se puede representar mediante:

$$G_p(s) = \frac{\omega(s)}{U(s)} = \frac{A}{\tau s + 1} = \frac{K}{s + \alpha} \tag{4.32}$$

Donde

ω= Velocidad angular en pulsos/seg

U= Comando de velocidad, en forma digital, enviado por el microcontrolador

A= Ganancia de CD (valor final al que se aproxima la respuesta al escalón unitario)

τ= Constante de tiempo del conjunto $G_p(s)$

α= valor del polo del conjunto $G_p(s)$

Los parámetros a obtener son las ganancias de CD y la constante de tiempo del conjunto $G_p(s)$. Este modelo se estimó, para cada servomecanismo, aplicando por medio del microcontrolador una entrada escalón (Comando digital u=70, constante), capturando la salida de velocidad en pulsos/seg, y comparando con la respuesta obtenida $\omega(t)$.

Dado que se trata del mismo motor de CD para cada servomecanismo, y se está capturando la salida de velocidad, se aproxima un modelo similar para los tres servomecanismos.

La Figura 4.10 muestra la respuesta al escalón de cada uno de los motores.

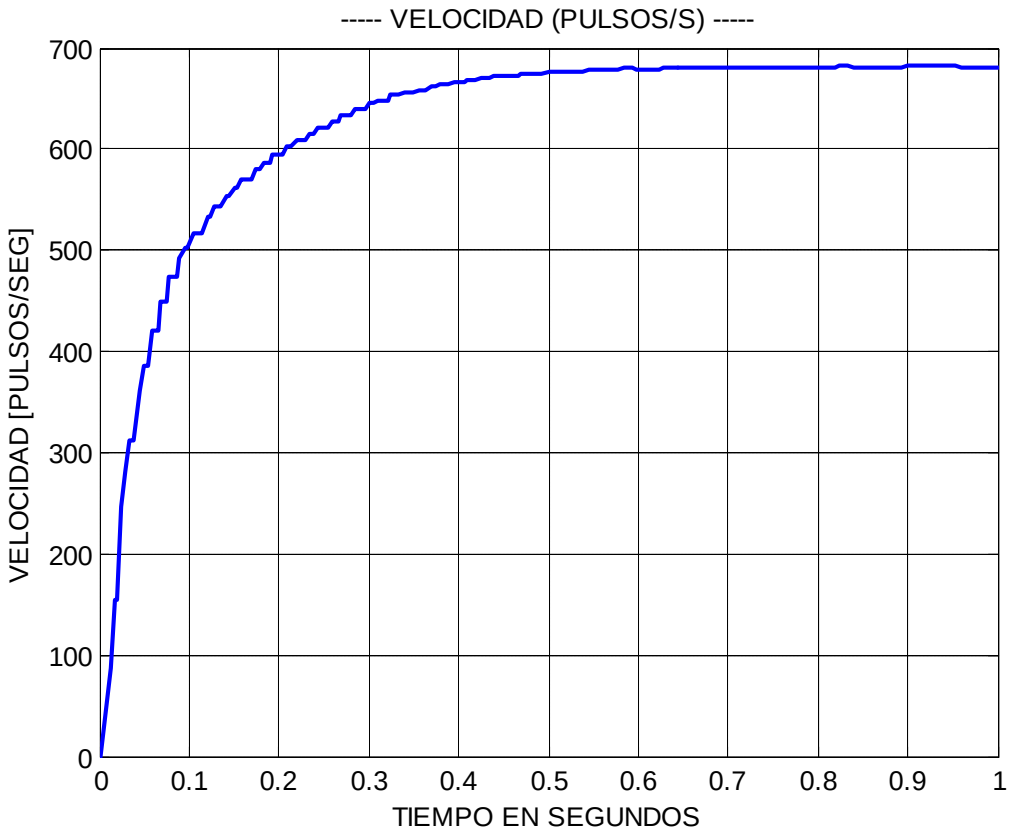


Figura 4.10 Respuesta al escalón del motor de CD

.El modelo aproximado de primer orden se obtiene tras el empleo del siguiente algoritmo.

De la ecuación (4.32), para encontrar el parámetro A, se sabe que la entrada escalón es de u=70, y de la grafica se observa que la velocidad en estado estable es de aproximadamente ω_{ss}=680 pulsos/s, por lo tanto:

$$A = \frac{\omega_{ss}}{u} = \frac{680}{70} = 9.715$$

Ahora, para encontrar el valor de la constante de tiempo aproximada, sabemos que la salida alcanza su valor de estado estable en 5 constantes de tiempo, y que una constante de tiempo ocurre en el 63.2% del valor en estado estable, así, haciendo un zoom a la Figura 4.10, obtenemos que dicho valor es:

$$\tau = 0.067$$

Por lo tanto, obtenemos el siguiente modelo aproximado:

$$G_p(s) = \frac{9.715}{0.067s+1} = \frac{145}{s+14.88} \tag{4.33}$$

Transformando a digital mediante el método Tustin a través de Matlab, la función de transferencia en modo digital es:

$$G_p(z) = \frac{0.6986}{z+0.9283} \tag{4.34}$$

Para un periodo de m muestreo de Ts=0.005 segundos.

Para validar este modelo, se aplican varias entradas escalón al motor y se comparan con la respuesta en lazo abierto de simulación, como se observa en la Figura 4.11.

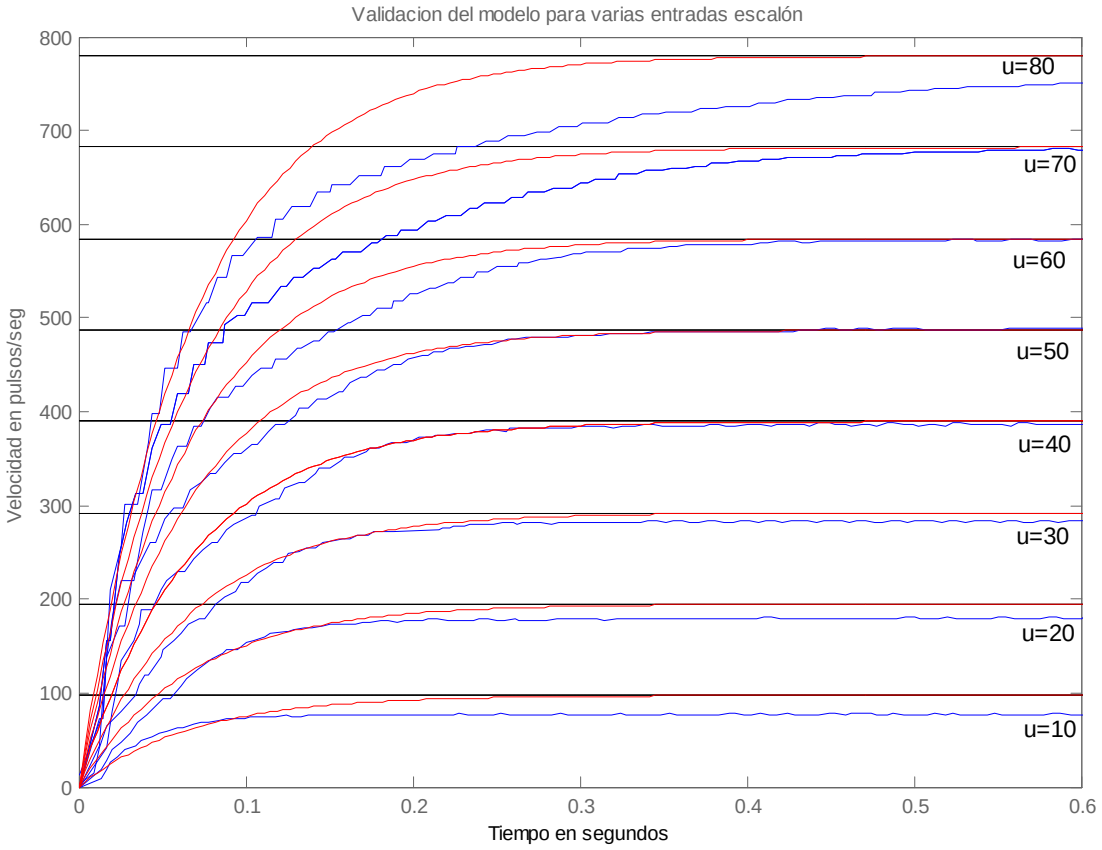


Figura 4.11 Validación del modelo para varias entradas escalón

4.3.3.2. Diseño del controlador IP digital

Para diseñar las ganancias del controlador IP es necesario conocer la función de transferencia de lazo cerrado del controlador IP dado por la ecuación analógica (4.19) y discreta (4.31). Donde el sistema mecánico Gp(s) está dado por la ecuación (4.31).

Para la acción de control analógica, la función de transferencia de lazo cerrado queda:

$$G(s)=\frac{\omega(s)}{\omega_{ref}(s)}=\frac{\frac{Ki}{s}\frac{A}{\tau s+1}}{1+\left(K_p+\frac{Ki}{s}\right)\frac{A}{\tau s+1}} \tag{4.35}$$

Por lo tanto reduciendo la ecuación (4.35) tenemos:

$$G(s)=\frac{\omega(s)}{\omega_{ref}(s)}=\frac{\frac{KiA}{s(\tau s+1)}}{1+\frac{(K_pAs+KiA)}{s(\tau s+1)}}=\frac{K_iA}{s(\tau s+1)+K_pAs+K_iA}=\frac{K_iA}{s^2\tau+s(1+K_pA)+K_iA}$$

$$G(s)=\frac{\frac{K_iA}{\tau}}{s^2+s\left(\frac{1+K_pA}{\tau}\right)+\frac{K_iA}{\tau}} \tag{4.36}$$

Donde:

K_i = Ganancia integral analógica.

K_p = Ganancia proporcional analógica.

A = Ganancia de CD de cada servomecanismo.

τ = Constante de tiempo del conjunto $G_p(s)$.

Del modelo analógico de la función de transferencia (ecuación 4.36) se pueden obtener las ganancias Ki y Kp analógicas, comparando la ecuación (4.36) con la función de transferencia general de segundo orden dada por la ecuación (4.37).

$$G(s)=\frac{\omega_n{}^2}{s^2+2\xi\omega_n s+\omega_n{}^2} \tag{4.37}$$

Comparando la ecuación (4.36) con la ecuación (4.37), se tiene que:

$$\omega_n{}^2=\frac{K_iA}{\tau} \qquad\qquad 2\xi\omega_n=\frac{1+K_pA}{\tau} \tag{4.38}$$

Por lo tanto:

$$K_i = \frac{\omega_n{}^2 \tau}{A} \tag{4.39}$$

$$K_p = \frac{2\xi\omega_n \tau - 1}{A} \tag{4.40}$$

Donde:

ω_n = Frecuencia natural no amortiguada

ξ = Relación de amortiguamiento

ω_n y ξ se pueden seleccionar utilizando las relaciones que definen el sobreimpulso máximo y el tiempo de establecimiento ts (2% de tolerancia) del sistema general de segundo orden y que están dados por las siguientes ecuaciones.

Sobreimpulso máximo:

$$M_p = e^{\frac{-\pi\xi}{\sqrt{1-\xi^2}}} \tag{4.41}$$

$$\text{Tolerancia: } = e^{-\xi\omega_n t_s} \tag{4.42}$$

Así, para una tolerancia del 2% se tiene:

$$t_s \approx \frac{4}{\xi\omega_n} \tag{4.43}$$

$$\omega_n \approx \frac{4}{t_s \xi} \tag{4.44}$$

Se puede calcular de la ecuación (4.41) en términos de un sobreimpulso especificado:

$$\xi \approx \sqrt{\frac{\left[\ln(M_p)\right]^2}{\left[\ln(M_p)\right]^2 - \pi^2}} \tag{4.45}$$

Para un sobreimpulso menor al 5%, se requiere un factor de amortiguamiento ξ=0.7. Con este valor del factor de amortiguamiento se puede plantear un tiempo de establecimiento deseado, (el cual se escoge para que la respuesta del motor sea más rápida que en lazo abierto) y calcular de la ecuación (4.44) la ω_n necesaria y las ganancias del controlador.

Aplicando este procedimiento se obtuvieron las ganancias analógicas y digitales para los motores de CD de cada eje mostradas en la Tabla 4.1.

Tabla 4.1 Ganancias del controlador de velocidad para los ejes X, Y y Z

	t_s	ω_n	K_i analógico	K_p analógico	K_{id} digital	K_{pd} digital
EJE Y	250 ms	22.63	5.6304	0.1732	0.028152	0.159124
EJE X	250 ms	22.63	3.5099	0.1069	0.017549	0.098125
EJE Z	250 ms	22,63	3.2509	0.10074	0.0162	0.09264

La Figura 4.12 muestra la respuesta en lazo cerrado del controlador IP diseñado, aplicando una referencia de ω=682, y comparando con la respuesta en lazo abierto para una entrada escalón de u=70.

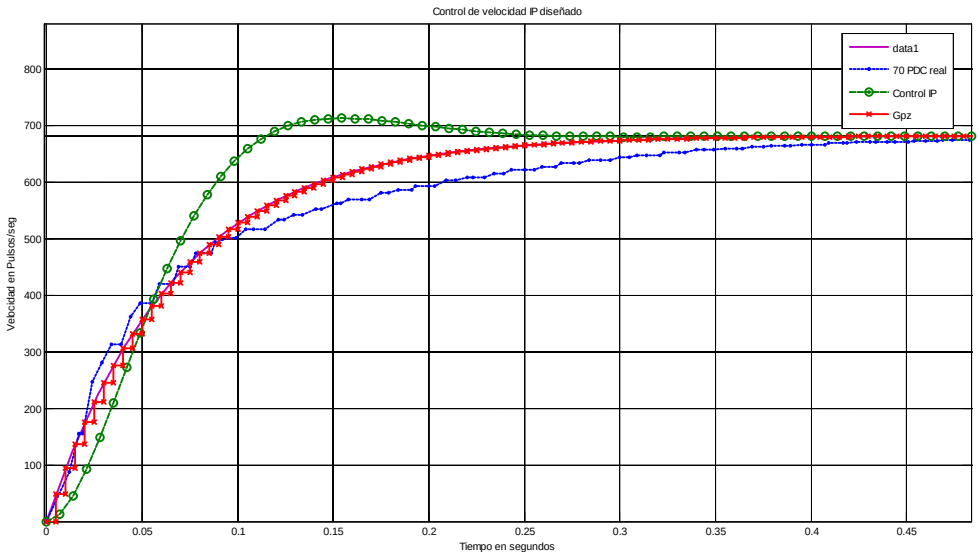


Figura 4.12 Respuesta del controlador IP diseñado

4.4. Software del manipulador

El software del manipulador consta de 5 programas que en conjunto definen el funcionamiento del sistema manipulador, 3 programas corresponden al software de los microcontroladores esclavos, otro programa corresponde al microcontrolador maestro, y el último de ellos es el programa desarrollado para la interfaz entre el usuario y la computadora. Todos los programas se comunican para sincronizar y ejecutar los movimientos de cada uno de los ejes, el microcontrolador maestro recibe los puntos de la interfaz, y los transfiere a cada esclavo para su correspondiente ejecución.

La rutina principal de la interfaz se encarga de tomar los puntos que ingrese el usuario desde el mouse de la computadora, posteriormente guarda los datos en un arreglo de una serie de coordenadas X-Y de longitud igual al número de puntos ingresados, una vez tomados estos datos, el programa los procesa para enviar posición y velocidad al microcontrolador maestro en un arreglo de 2n-1, donde n es el número de puntos.

El microcontrolador maestro Recibe el arreglo de posición y velocidad desde la interfaz de Matlab y a su vez se encarga de enviar los correspondientes datos de posición y velocidad a cada uno de los microcontroladores esclavo que controlan los ejes, dicha comunicación se hace mediante el protocolo i2c, explicado en la sección 3.7.2.

El software de los microcontroladores esclavos consta de los controladores de posición y velocidad de cada eje, dicho software está diseñado con las ganancias calculadas en la sección 4.3.3.2, el mismo procedimiento se aplicó a los ejes X y Z.

4.4.1. Cálculo de la posición y velocidad para cada servomecanismo

Para el caso del manipulador utilizado, se requiere el cálculo de una trayectoria recta que une a dos puntos en el plano P, como se muestra en la figura 4.13.

Para calcular el movimiento de cada eje durante el movimiento de un punto a otro primeramente se calculan los desplazamientos, después, con base en estos desplazamientos se calculan las velocidades.

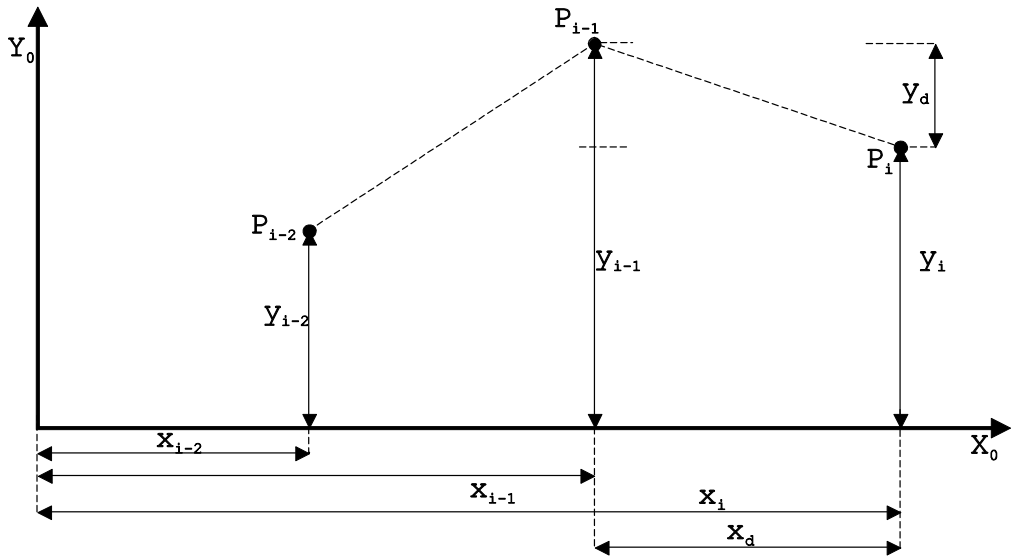


Figura 4.13 Seguimiento de trayectorias entre puntos.

Cálculo de los desplazamientos parciales. Primero se calculan los desplazamientos parciales de los ejes X e Y, los cuales están dados por las ecuaciones 4.46 y 4.47 respectivamente.

$$x_d = x_i - x_{i-1} \tag{4.46}$$

$$y_d = y_i - y_{i-1} \tag{4.47}$$

Los desplazamientos parciales de los ejes X e Y son convertidos a pulsos de su correspondiente encoder mediante la ecuación 4.48.

$$num.\ pulsos = \frac{X_L(1024)}{aS} \tag{4.48}$$

Donde a es la relación de reducción del engrane (a=1:75), S representa el desplazamiento parcial de cada servomecanismo (0.03cm/rev), y X_L es el desplazamiento parcial de cada eje (x_d e y_d, respectivamente).

La ecuación (5.34) puede arrojar un resultado mayor de 32767, lo cual proporciona una resolución de 0.1cm, que es el tamaño de una palabra de 16 bits con signo manejada en el microcontrolador, por lo cual antes de calcular el número de pulsos mediante la ecuación 4.48, se obtiene un factor entre el cual se divide el desplazamiento para que quede dentro

de este rango, como se muestra en la ecuación 4.49. Este factor es el número de veces que se repite el error de posición antes de llegar a la posición deseada.

$$\begin{aligned} |desplazamiento| \leq 0.1cm &\rightarrow \text{Factor de repetición = 1} \\ 0.1cm < |desplazamiento| \leq 1cm &\rightarrow \text{Factor de repetición = 10} \\ 1cm < |desplazamiento| \leq 11cm &\rightarrow \text{Factor de repetición = 100} \\ 11cm < |desplazamiento| &\rightarrow \text{Factor de repetición = 1000} \end{aligned} \tag{4.49}$$

Para lograr que los ejes X e Y alcancen su posición final al mismo tiempo describiendo una trayectoria recta, se aplica lo siguiente: si el desplazamiento de un eje es diferente al del otro, al eje con menor desplazamiento se le asigna una velocidad proporcionalmente menor que el otro. De acuerdo con la ecuación de velocidad lineal (4.50) y el requerimiento del tiempo de desplazamiento del eje X sea igual al Y, se obtienen las relaciones (4.51), (4.52) y (4.53):

$$V_x = \frac{x_d}{t_x} \qquad V_y = \frac{y_d}{t_y} \tag{4.50}$$

$$t_x = t_y = \frac{x_d}{V_x} = \frac{y_d}{V_y} \tag{4.51}$$

$$\frac{x_d}{y_d} = \frac{V_x}{V_y} \tag{4.52}$$

Para que se cumpla la relación 4.52, es necesario que haya un factor de velocidad dado por la ecuación 4.53, dicho factor de velocidad es el que interpreta el microcontrolador (tiene que ser un un número mayor a la unidad).

$$factor\ de\ velocidad = \frac{desplazamiento\ mayor}{desplazamiento\ menor} \tag{4.53}$$

Una vez calculada la posición (numero de pulsos o error de posición), y el factor de velocidad, dados por las ecuaciones (4.46), (4.47), (4.48), y (4.53) respectivamente, se obtienen las velocidades de los ejes X e Y, éstos valores son transmitidos al microcontrolador maestro que se encargará de sincronizar los microcontroladores esclavos de cada eje y definir su trayectoria a seguir.

4.4.2. Software de la interfaz gráfica

Cuando se utilizan manipuladores como máquinas de uso general, uno de los retos es lograr una comunicación eficiente y adecuada entre la interfaz, el usuario final y el sistema, de tal forma que el usuario pueda dirigir al robot para cumplir las tareas deseadas.

El software de la interfaz para manipulador se desarrolló como una interfaz de usuario GUI sobre la plataforma del software del Matlab 7 [MATLAB Getting Started,2011]. Matlab es una herramienta de análisis numérico que posee una gran variedad de toolbox o herramientas para aplicaciones específicas, con este programa se pueden desarrollar herramientas poderosas para simulación de sistemas y procesamiento de datos, además de que posee un lenguaje propio de programación, como C o Fortran.

El software de la interfaz actúa como enlace entre el sistema y el usuario, dicho software se encarga de calcular y enviar al microcontrolador maestro, los comandos de posición y velocidad que debe adquirir cada uno de los microcontroladores esclavos para mover cada eje. Las funciones que desarrolla el software son: El control de la interfaz gráfica para la interacción con el usuario, el cálculo del desplazamiento entre los servomecanismos de los ejes X e Y, asi como sus respectivas velocidades y la comunicación con el microcontrolador maestro. En la figura 4.12 se muestra el diagrama de flujo del software de la interfaz. Primeramente se requiere configurar el puerto serial para llevar a cabo la emulación del puerto USB para la comunicación con el microcontrolador Maestro (PIC18F4550), se da de alta el puerto “COM4” (aunque varía dependiendo del puerto utilizado en su momento), dicho bus de datos se comunica con el microcontrolador maestro, a una máxima velocidad de transmisión de 921600 baudios, y utilizando datos de 8 bits, sin control de flujo.

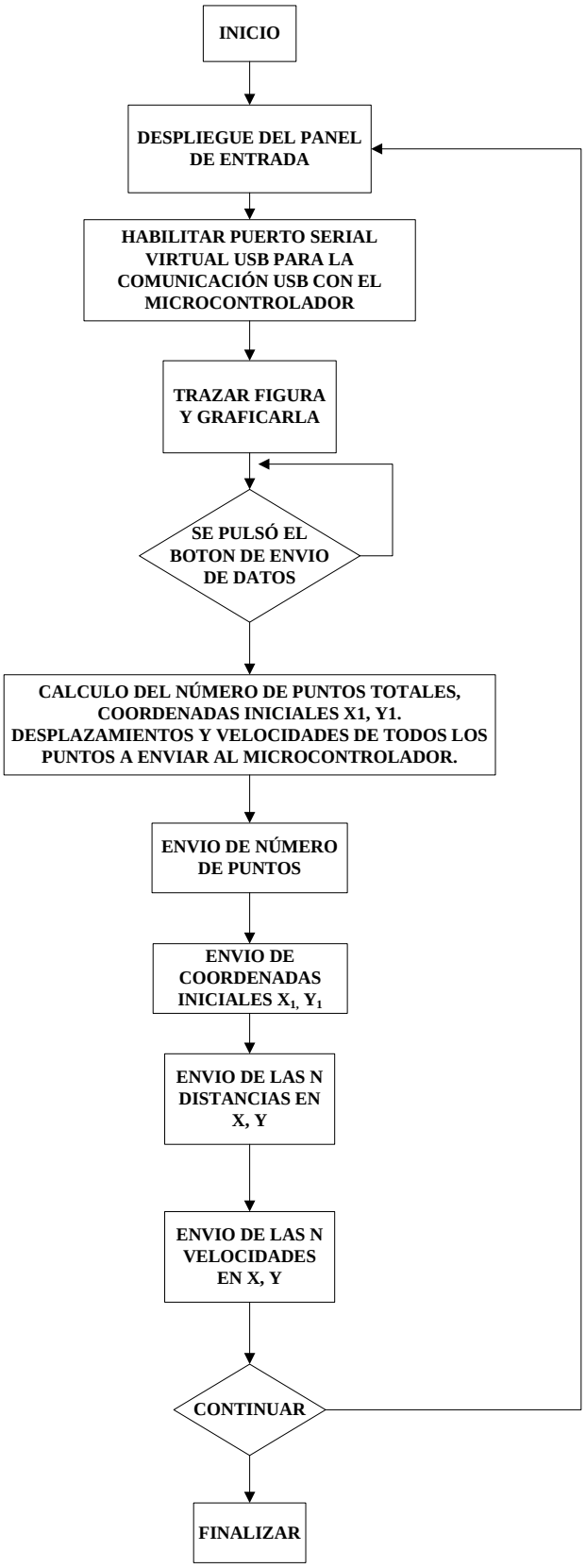


Figura 4.14 Diagrama de flujo del programa principal en Matlab.

Una vez configurado el puerto, se despliega una pantalla de visualización en la cual se pueden trazar varios puntos con la ayuda del mouse para formar una figura deseada, como lo muestra la figura 4.15.

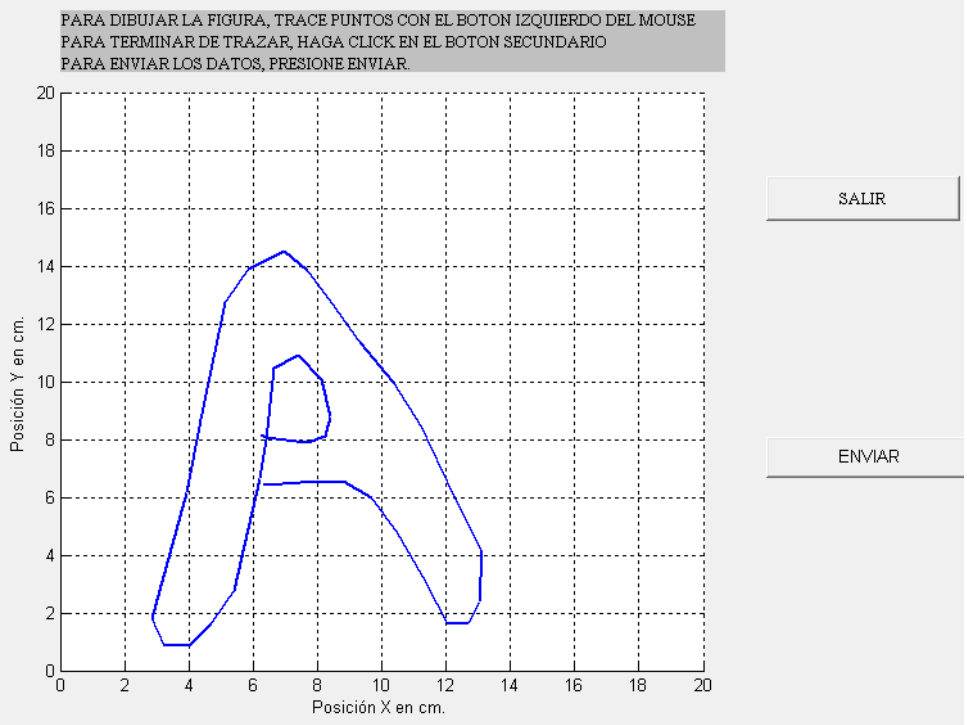


Figura 4.15 Panel de la interfaz gráfica realizada en Matlab.

Mediante esta interfaz el usuario tiene acceso a dos botones que son: envío de datos y salir, con dichos botones se puede controlar el envío de datos, para trazar la figura como lo marcan las instrucciones, el usuario deberá tocar los puntos con el botón primario del mouse (Botón Izquierdo), y para terminar la figura, bastará con dar clic en el botón secundario (Botón Derecho). Existe un límite máximo de puntos que se pueden enviar, esto para no saturar el límite de memoria interna del microcontrolador, para esto, se limita a un máximo de 256 puntos (255 líneas), y como se envían parte alta y parte baja, esto es un total de 512 bytes.

Para representar lo dicho anteriormente, se presenta un ejemplo, en el cual se quiere realizar el trazo mostrado en la figura 4.16; esta figura consta de 11 puntos (10 líneas) unidos para formar una estrella.

Al trazar esta figura con el mouse, el programa almacena un arreglo con el número de puntos:

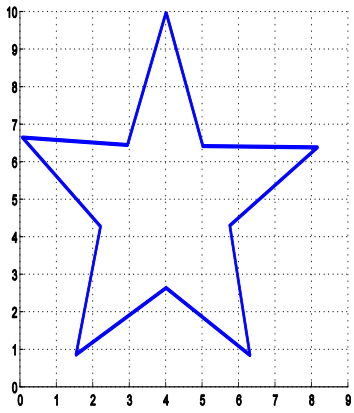


Figura 4.16 Figura de ejemplo

```
xy =

[   151   400   630   575   815   501   400   294     6   220   153;

    86   265    86   431   639   642   999   645   665   428    86  ]
```

El programa hace un escalamiento de 1 a 100 porque la resolución del sistema es de una decima de milímetro, es decir que una distancia de 200 equivale a 2 centímetros.

El primer renglón almacena los puntos en el eje x, y el segundo renglón, los puntos en el eje y, con estos datos, el programa calcula las distancias entre los puntos que forman la estrella, aplicando las relaciones 4.46 y 4.47 obteniendo así:

```
lxy =

[   249   230    55   240   314   101   106   288   214    67;

   179   179   345   208     3   357   354    20   237   342  ]
```

Después de esto el programa calcula las respectivas velocidades de cada eje, aplicando las relaciones 4.49, a 4.53, obteniendo:

```
velxy =

[   480   480   -77   480  -480  -136  -144  -480   433   -94;

    345  -374   480   416     5   480  -480    33  -480  -480 ]
```

Recordando que el signo negativo en la velocidad indica un cambio de sentido en la dirección del motor. Estos dos últimos arreglos son los que se envían desde la interfaz al microcontrolador maestro, además del número de líneas, en este caso n=10 y el primer punto antes de empezar a trazar, en este caso x=151 e y=86;

4.4.3. Software de los microcontroladores

Las rutinas de los controladores para cada uno de los servomecanismos fueron desarrollados en microcontroladores PIC18F2431 y el PIC18F4550 en lenguaje C, bajo el compilador CCS (Custom Computer Services) [CCS Reference Manual, 2011]. Se efectúa una comunicación bidireccional entre un maestro y tres esclavos como se observa en la figura 4.12 el PIC18F4550 actúa como maestro y los PIC18F2431 actúan como esclavos. A continuación se describen el funcionamiento de cada uno de los programas desarrollados:

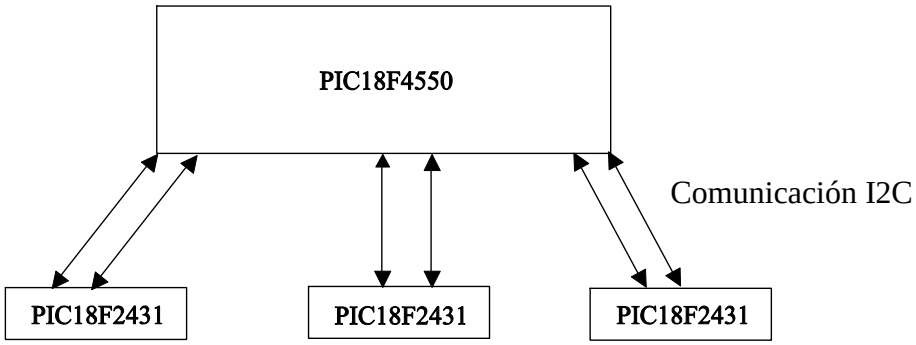


Figura 4.17 Comunicación bidireccional entre microcontroladores.

4.4.3.1. Software del microcontrolador maestro.

El software del microcontrolador maestro se encarga principalmente de sincronizar las tareas de los microcontroladores esclavos para ejecutarse en tiempos precisos, dicha sincronización se logra mediante el envío de comandos de inicio y solicitud de comandos de finalización, de cada uno de los puntos a seguir, esto es muy útil porque no se hacen movimientos a destiempo o movimientos desiguales. El diagrama de flujo del programa principal maestro se muestra en la figura 4.18. Este algoritmo se divide en dos etapas, la primera es la etapa de recepción de datos de la interfaz, y la segunda etapa es la encargada de la comunicación con los esclavos.

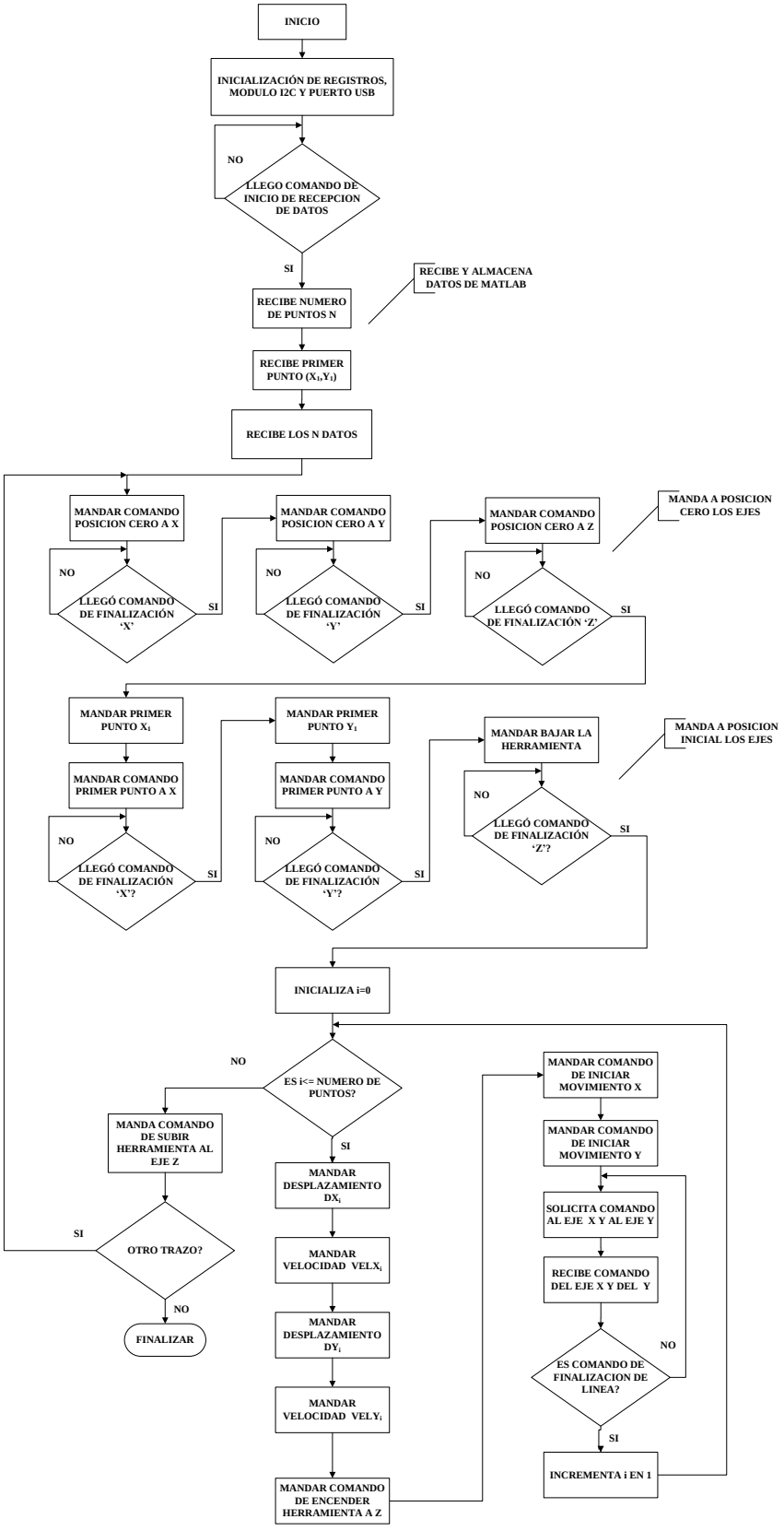


Figura 4.18 Diagrama de flujo del programa para el microcontrolador maestro.

La etapa de recepción de datos consiste en esperar hasta que el programa de la PC (interfaz con el usuario) le envíen los arreglos de posición y velocidad, el número de puntos y los puntos iniciales en X y en Y; una vez que se reciben todos estos datos, se procede a la siguiente etapa de comunicación con los esclavos.

En la segunda etapa se programo en forma de máquina de estados. En el primer estado, el microcontrolador maestro envía a los 3 microcontroladores esclavos el comando para ir a la posición de origen (HOME_POSITION en el programa, Apéndice A.2), un comando a la vez para cada uno de los ejes; una vez enviado, el microcontrolador maestro espera respuesta de finalización, es decir que cada servomecanismo está en su posición de origen, después de que los tres ejes se encuentran en el origen pasa al siguiente estado. En el segundo estado se envía el punto de partida para el trazado de la figura deseada, dicho punto es la distancia del origen al punto x_1 y al punto y_1, así como el desplazamiento que recorrerá el eje z para situar la herramienta sobre la superficie a trazar, una vez alcanzado el punto de inicio de la figura por los 3 ejes el maestro recibe información de cada esclavo indicando que ya está en la posición inicial (FIRST_POINT en el programa Apéndice A.2). En el tercer estado se envía el número de líneas a trazar, se envían todos los datos de posición y velocidad a los microcontroladores esclavos dentro de un ciclo del tamaño igual al número de líneas a trazar, y se inicia el movimiento (NUMBER_OF_POINTS y START_ROUTING, Apéndice A.2). Una vez trazadas todas las líneas, el microcontrolador maestro manda comando para apagar la herramienta y subirla, y así posteriormente esperar recibir una instrucción de terminación de cada uno de los ejes para indicar que ya se ha trazado la figura. El cuarto estado consiste en enviar una vez más el comando de posición cero u origen, para regresar todos los ejes a la posición de cero y terminar el trazado.

4.4.3.2. Software de los microcontroladores esclavos

Las funciones que realiza cada uno de los microcontroladores esclavos son:

- Comunicación con el microcontrolador maestro mediante el bus I2C.
- Medición de la posición y velocidad del rotor.
- Cálculo de la acción de control de velocidad..

- Cálculo del factor de repetición y posicionamiento del eje para un valor deseado de posición.

Comunicación con el microcontrolador maestro. Esta comunicación se hace mediante el bus i2c [I2C Phillips, 2000], estableciendo una comunicación bidireccional en la que hay un control de información que llega y se envía, se esperan comandos de inicio de movimientos y se envían comandos de finalización, esto para que no haya confusión entre microcontroladores. Los microcontroladores esclavos generan una interrupción cada vez que se reconoce un byte en el modulo i2c, una vez que se lee un byte, se interpreta para indicar la acción a realizar, por ejemplo, el byte START_HOME (START_HOME = 100 Apéndice A.2) es el byte que indica que el microcontrolador irá a su posición de origen, ahora por ejemplo, el byte DISTENCE (DISTANCE = 124 Apéndice A.2) indica que el siguiente byte entrante es la distancia a recorrer por el microcontrolador, esto permite identificar los datos de velocidad y posición deseadas, así como del número de puntos.

Medición de la posición y velocidad del rotor. Consiste en medir los pulsos por cada periodo de muestreo del microcontrolador, así como el desplazamiento recorrido por el eje en función de los pulsos medidos por el encoder del motor.

El encoder del motor tiene una resolución de 256 pulsos por vuelta, y como se están usando los canales A y B del encoder, por cada pulso se tienen 4 aumentos en la cuenta del registro QEICON, como se explicó en la sección 4.2.1, así por cada vuelta del motor se tendrá un total de 1024 pulsos. Y como el tornillo tiene un factor de reducción de (1:75) quiere decir que por cada vuelta del motor, el tornillo gira 1/75 de vuelta, entonces para lograr una vuelta del tornillo, se requiere girar 75 veces la flecha del motor, ahora el factor de desplazamiento del tornillo es S=0.3 cm/rev_tornillo, o en revoluciones del motor es S=0.004 cm/rev_motor, y dado que una vuelta del motor quivale a 1023 pulsos, se tendrá una resolución en pulsos de $S=3.91\times10^{-6}$ cm/pulso, por lo tanto, para un desplazamiento de 1 cm se requiere 1/S pulsos, 1/S = 255750 pulsos. Es así como se hace la medición de la posición.

La velocidad se mide restando las posiciones actual y anterior leidas en la interrupción del timer, por ejemplo, si en un periodo de muestreo T0 se mide un desplazamiento de 768 pulsos, y en el siguiente periodo de muestreo T1 se miden 980 pulsos, la velocidad en ese

instante es ω=980-768=212 pulsos/Ts. Como se explica en la sección 4.2.2, si el valor del bit UP/DOWN es 1, significa que el motor gira en otro sentido, entonces la cuenta va en descenso, por lo que el valor anterior es mayor siempre que el valor actual, por ejemplo, en un periodo de muestreo T0 se mide un desplazamiento de 600 pulsos y en el periodo de muestreo actual T1 se miden 300 pulsos, entonces la cuenta sería negativa, es por ello que se invierte la resta y el resultado nos dará siempre positivo ω=600-300=300 pulsos/Ts.

Cálculo de la acción de control y velocidad. Esta etapa se encuentra en la misma rutina de medición de la velocidad y posición, en la cual se calcula la acción de control mediante el controlador IP de velocidad explicado en la sección 4.3.3 y así, hacer la actualización de los comandos del PWM para lograr la velocidad.

Cuando recibe un comando de velocidad se fija la referencia de velocidad y la dirección de giro del motor, la referencia de velocidad permanece así hasta que se reciba un nuevo comando. Cuando se recibe un comando de posición se fija la referencia de posición he inicia la rutina para llegar a la posición de referencia.

Cálculo del factor de repetición y posición del eje. Se calcula un factor de repetición y se ejecuta tantas veces hasta lograr el desplazamiento deseado de cada eje, como se explica en 4.4.1.

En la figura 4.19 se presenta el diagrama de flujo que describe el algoritmo para cada microcontrolador esclavo.

De acuerdo al diagrama de flujo de la figura 4.19 se describe el funcionamiento de los manejadores de los ejes X, Y y Z, dicho funcionamiento se basa en una serie de estados en los que se ejecutan los movimientos.

En el primer estado, se espera un comando de ir a posición de origen (START_HOME, Apéndice A.3.), los tres microcontroladores esclavos esperan dicho comando, y una vez que llega, el microcontrolador mueve el eje a posición cero, después de alcanzar dicho punto, pasa al segundo estado y ahora se espera otro comando, que es el de ir al primer punto (FIRTS_POINT Apéndice A.3), mientras que los ejes X e Y, recibe el número de líneas a trazar, los datos posición y velocidad y se mueven para llegar al primer punto, el eje Z sitúa la herramienta sobre la superficie a trazar.

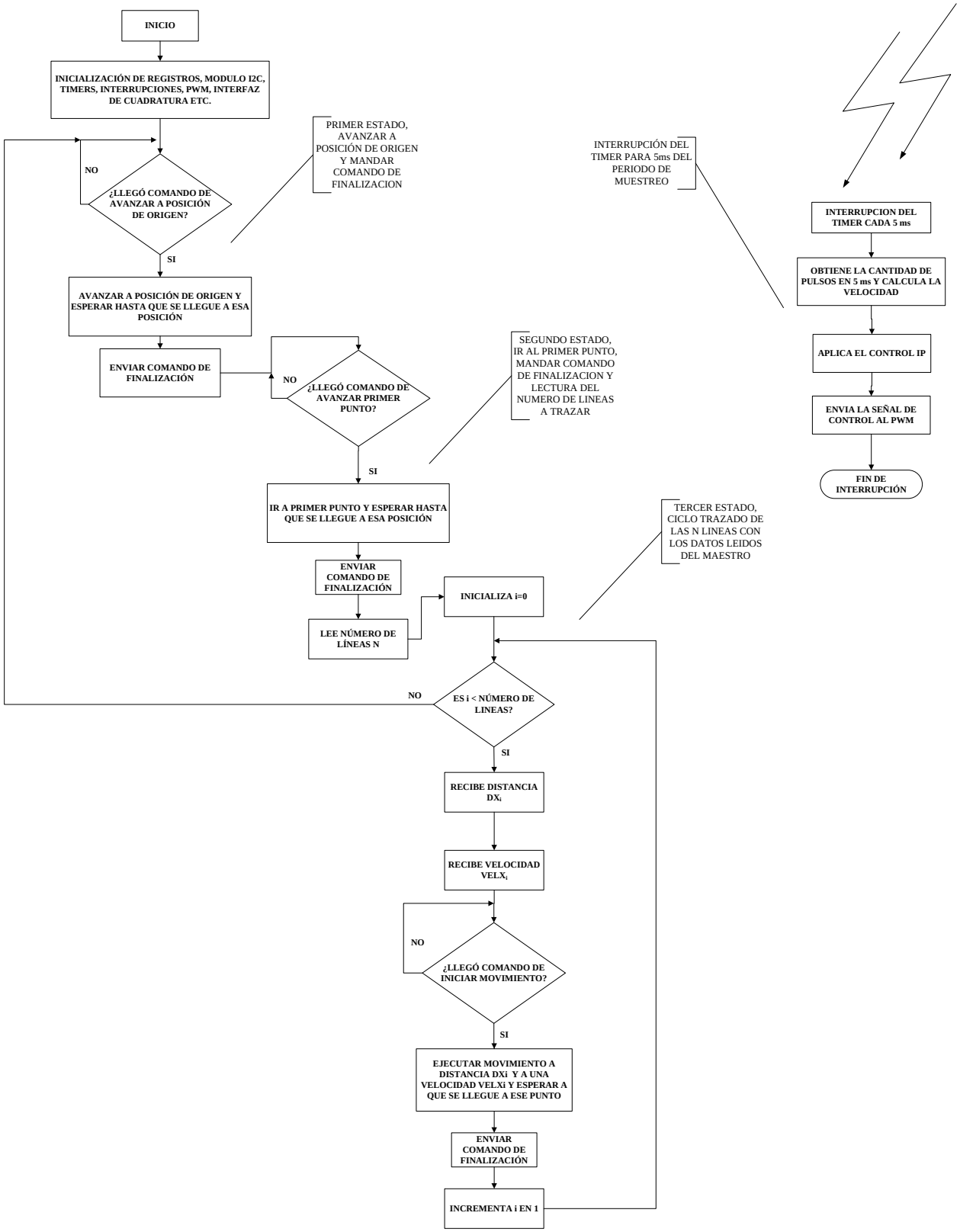


Figura 4.19 Diagrama de flujo del programa para los ejes X e Y.

En el tercer estado se espera a recibir el comando de comenzar con el trazado, esto dentro de un ciclo que se repite tantas veces como el número de líneas a trazar y sólo para los ejes X e Y, igualmente que con el primer punto, llegan datos de velocidad y posición y su respectivo comando, esto se hace para las n líneas que contiene la figura. Para el eje Z, durante todo el ciclo se enciende la herramienta para rotular la figura deseada sobre la superficie, una vez que se rotulan las n líneas, el eje Z espera comando de fin de línea o fin de figura (LINEA_OK, Apéndice A.3.3) para con ello, subir la herramienta y terminar el rotulado. En la figura 4.20, se muestra el diagrama de flujo para el eje Z.

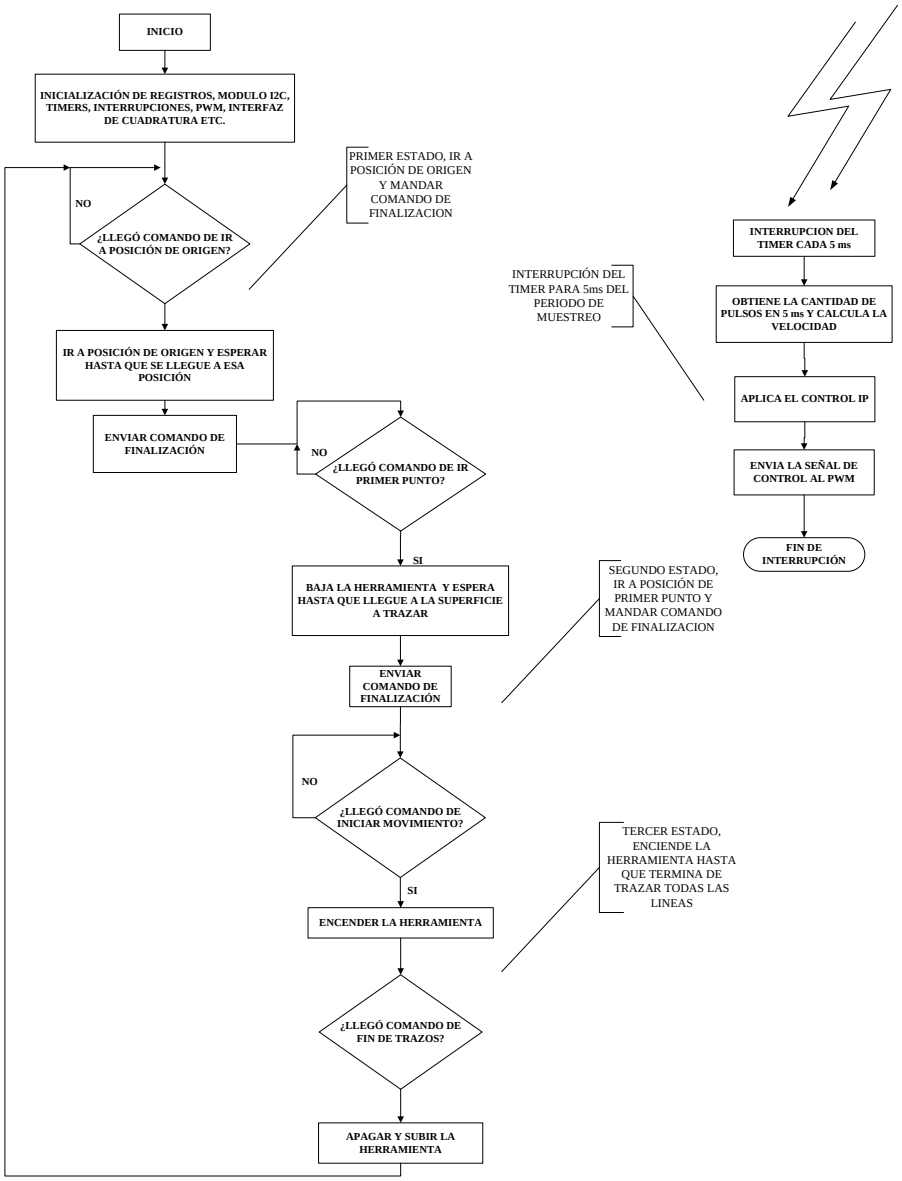


Figura 4.20 Diagrama de flujo del programa del eje Z.

En este capítulo se describieron y analizaron los algoritmos de control utilizados para el control de posición y velocidad de los servomecanismos, así como también se describieron los métodos utilizados para medir posición y velocidad, que son útiles para hacer el control de dichas variables, y se describió el software implementado para la Interfaz gráfica y para los microcontroladores, tanto el maestro como los esclavos.

En el siguiente capítulo se presentan las pruebas realizadas a los controladores de velocidad y posición para cada uno de los servomecanismos, así como el sistema completo del manipulador.

Capítulo 5

Pruebas y resultados

5.1. Introducción

En el presente capítulo se presentan las pruebas y resultados de los servomecanismos por separado, se probaron los controladores de posición y velocidad para los servomecanismos de los ejes X, Y, Z, cada uno con su respectiva carga acoplada, caja de engranes y tornillo. En seguida se enlista la forma general del procedimiento de pruebas que se utilizó para el manipulador.

- Pruebas de Hardware
- Pruebas de Software
- Pruebas de cada uno de los servomecanismos individuales

Una vez comprobados los controladores de los servomecanismos, se obtuvieron pruebas del sistema completo del manipulador

- Pruebas de la interfaz.
- Pruebas del sistema completo. (Perforador y Rotulador).

5.2. Pruebas de hardware

Las pruebas individuales de cada una de las tabletas del hardware se realizaron de acuerdo a lo siguiente:

- Control del convertidor de CD a CD.
- Tarjeta Electrónica del driver para el manipulador.

La figura 5.1 muestra las señales PWM que disparan los mosfets vistas en el osciloscopio, esto prueba la funcionalidad del módulo PWM.

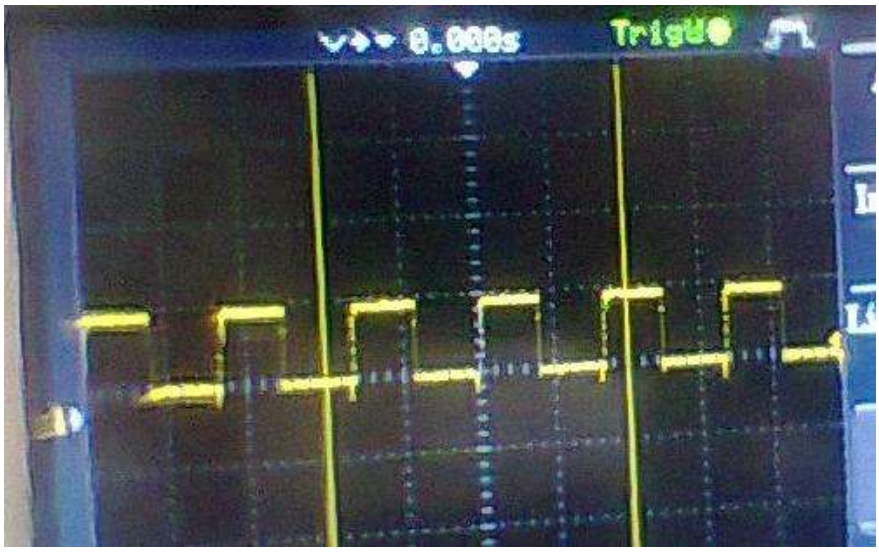


Figura 5.1 Señal PWM al 50% del ciclo de trabajo.

5.3. Pruebas de software

El software del manipulador se probó de la siguiente manera:

- Prueba de medición de Velocidad y Posición.
- Prueba del controlador de velocidad.
- Prueba del controlador de Posición.
- Pruebas del manipulador como rotulador.

5.4. Reportes de las pruebas.

Las graficas mostradas a partir de la Figura 5.1 a la 5.12 ilustran las pruebas realizadas del software.

Inicialmente, se probó el funcionamiento de la tableta del driver que contiene desde el microcontrolador, hasta los circuitos de acoplamiento y convertidor de CD a CD, para ello se realizaron pruebas en lazo abierto de cada uno de los servomecanismos con todo y la carga acoplada, para dichas pruebas se aplicaron entradas escalón u=60 para cada uno de los servomecanismos por separado.

5.4.1. Pruebas en lazo abierto de los servomecanismos

En las Figuras 5.2, 5.3 y 5.4 se muestran la respuestas de lazo abierto para los servomecanismos del eje X, Y y Z, con caja de engranes y tornillo acopladas, para una entrada tipo escalón.

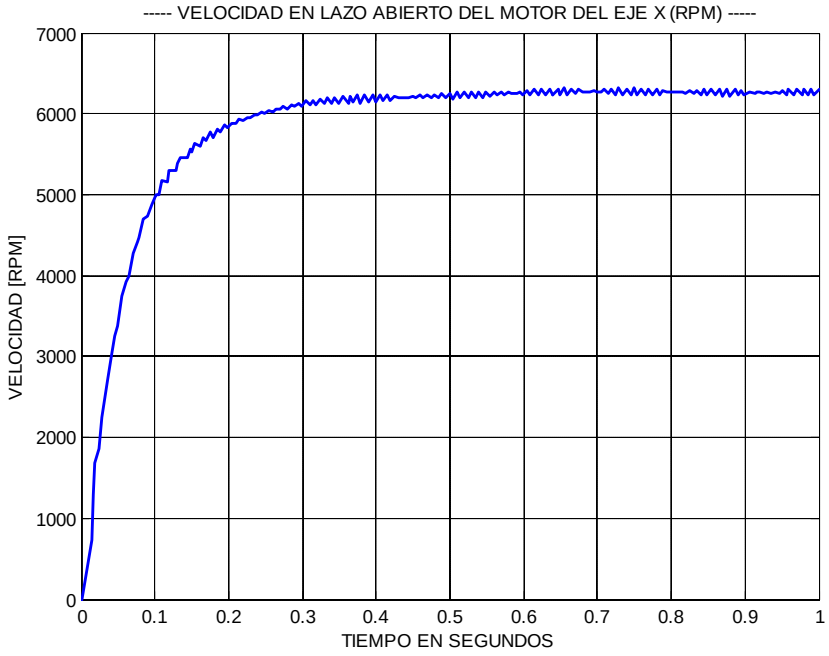


Figura 5.2 Respuesta en lazo abierto del servomecanismo del eje X.

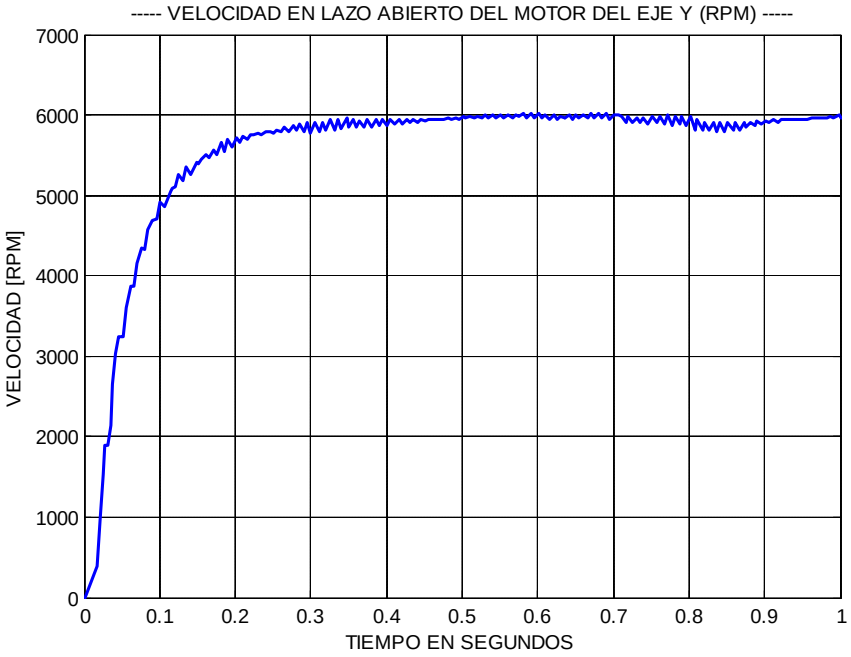


Figura 5.3 Respuesta en lazo abierto del servomecanismo del eje Y.

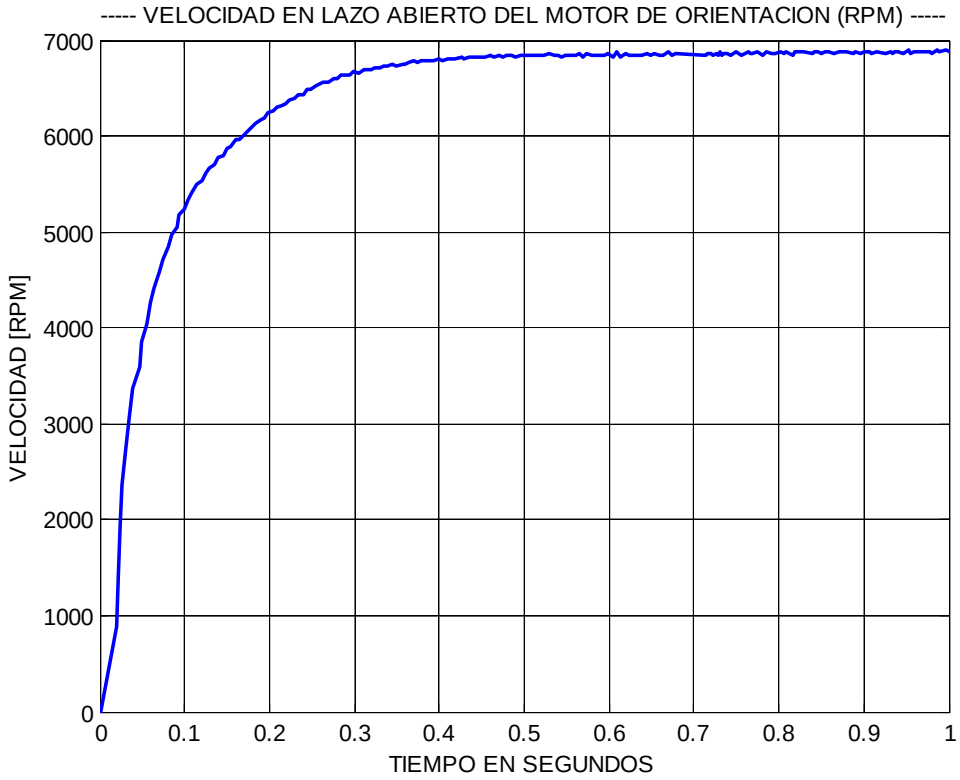


Figura 5.4 Respuesta en lazo abierto del servomecanismo del eje Z.

Como puede observarse, el servomecanismo del eje Z y el servomecanismo del eje X, presentan mayor velocidad en estado estable que el servomecanismo del eje Y, esto se debe a que la carga en ambos es mucho menor que la carga en el eje Y, debido a que la carga en el eje Y es prácticamente toda la estructura del manipulador.

5.4.2. Pruebas de los controladores de velocidad y posición

Se presentan en las figuras 5.5, 5.6 y 5.7 las pruebas del controlador diseñado para cada uno de los servomecanismos, X, Y y Z respectivamente, todas para una señal de referencia en revoluciones por minuto de ω=3515 RPM en cada eje.

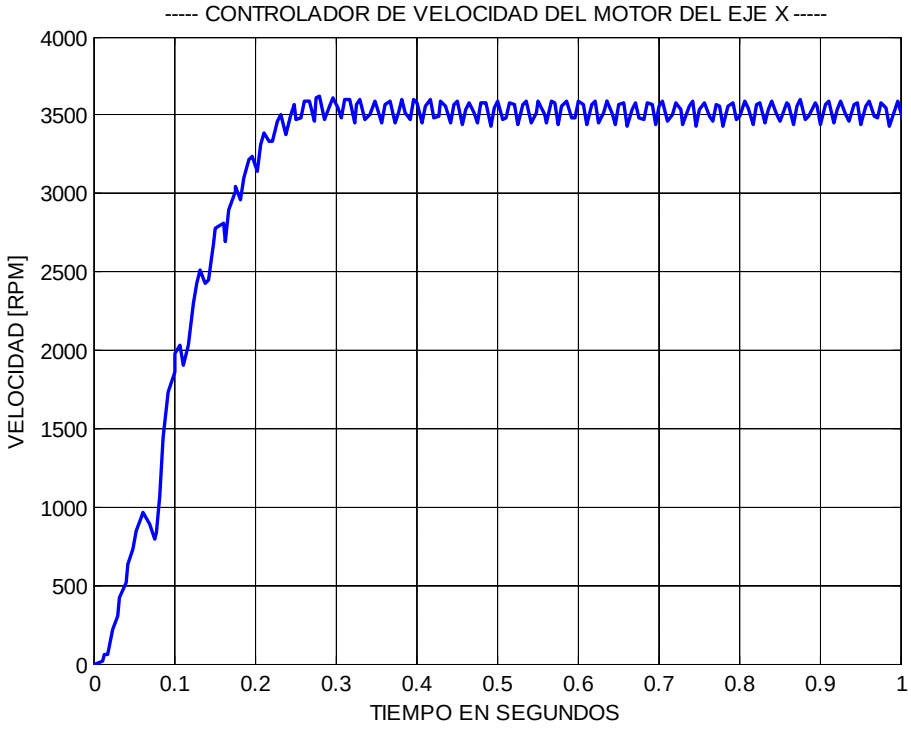


Figura 5.5 Respuesta en lazo cerrado del servomecanismo del eje X.

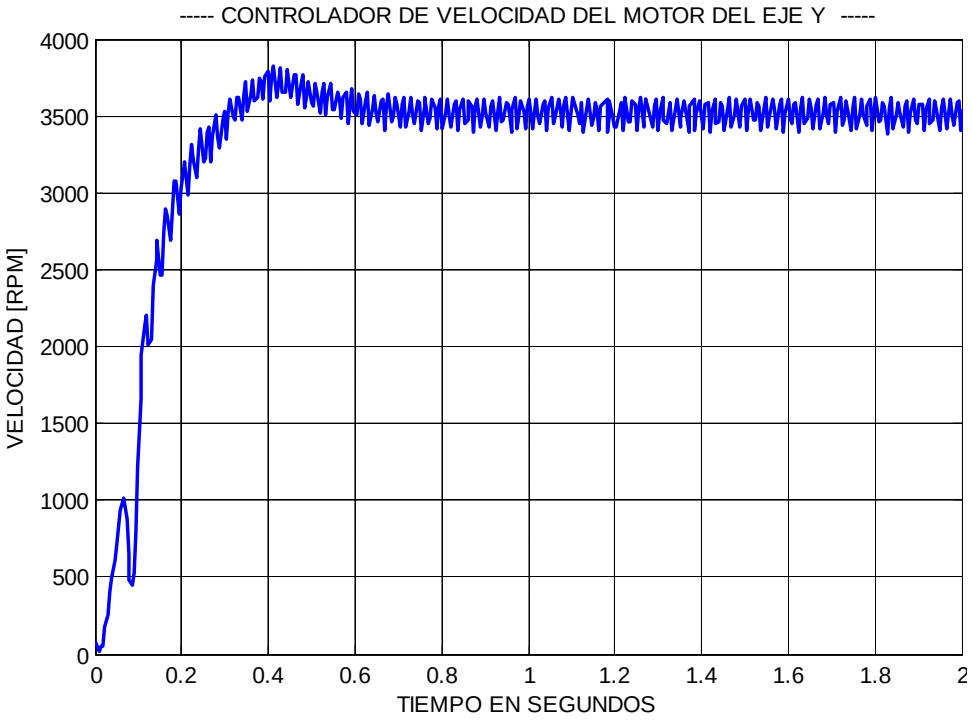


Figura 5.6 Respuesta en lazo cerrado del servomecanismo del eje Y.

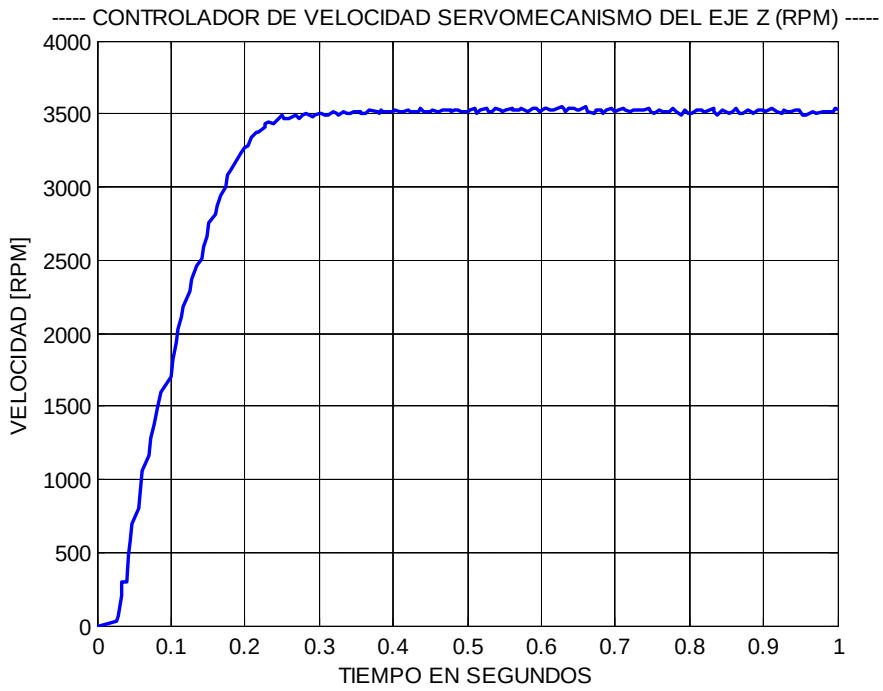


Figura 5.7 Respuesta en lazo cerrado del servomecanismo del eje Z.

En las Figuras 5.6 y 5.7 se observa un pequeño disturbio al comienzo del arranque, esto se debe a los momentos de inercia de las cargas de los respectivos ejes; como los ejes X e Y son los que tienen mayor peso en la carga, el motor hace un esfuerzo más grande que si no la tuviera al momento del arranque. En el servomecanismo del eje Z no se alcanza a distinguir esta variación debido a que el eje Z no tiene mucha carga.

En las figuras 5.8 a 5.13 se muestran las pruebas realizadas al control de posición, para cada servomecanismo, con una entrada de referencia de posición en pulsos de 32767 en cada eje, también se muestran los perfiles de velocidad deseados para cada uno de los ejes.

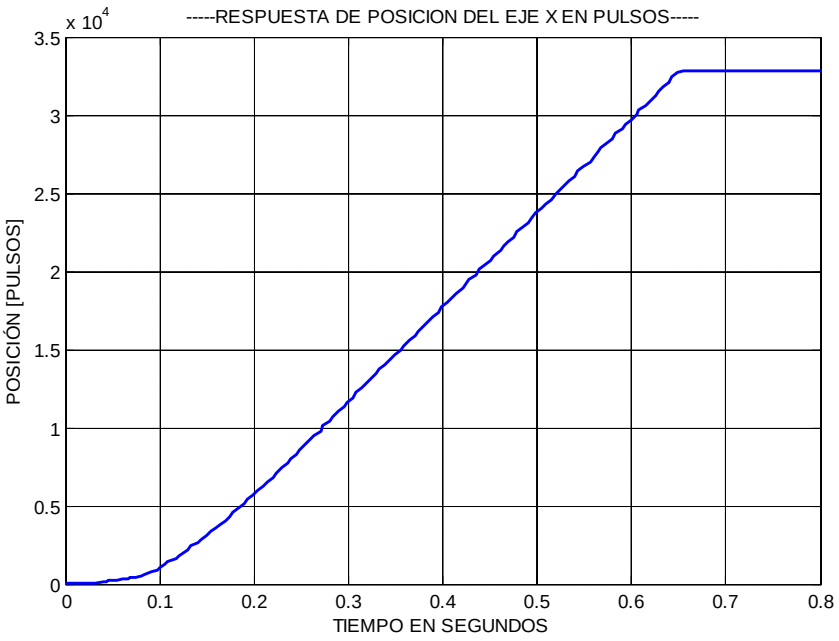


Figura 5.8 Respuesta en lazo cerrado a un comando de posición para el eje X.

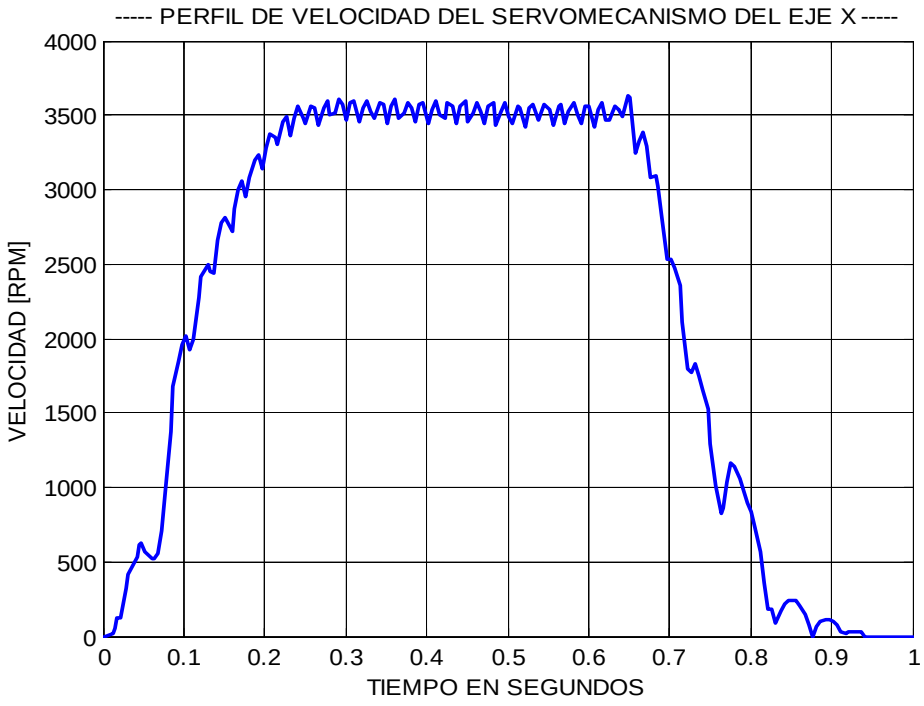


Figura 5.9 Perfil de velocidad del control de posición, servomecanismo del eje X.

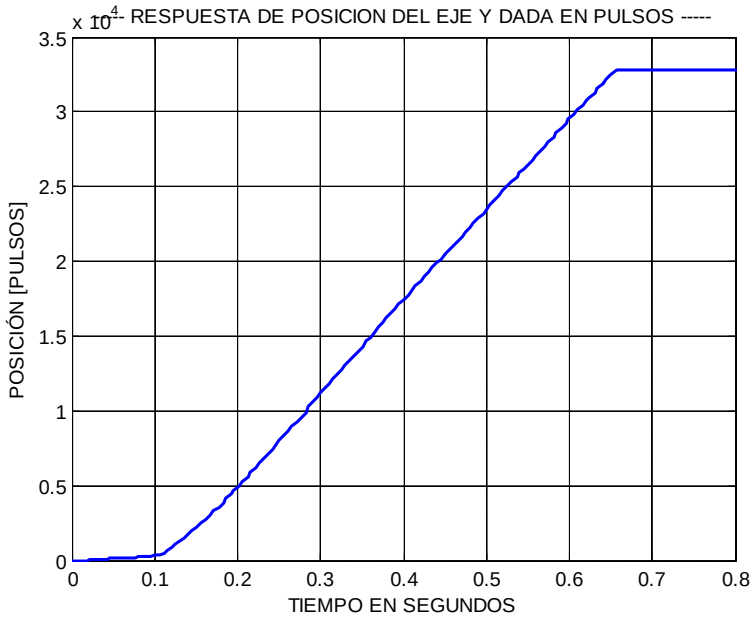


Figura 5.10 Respuesta en lazo cerrado a un comando de posición para el eje Y.

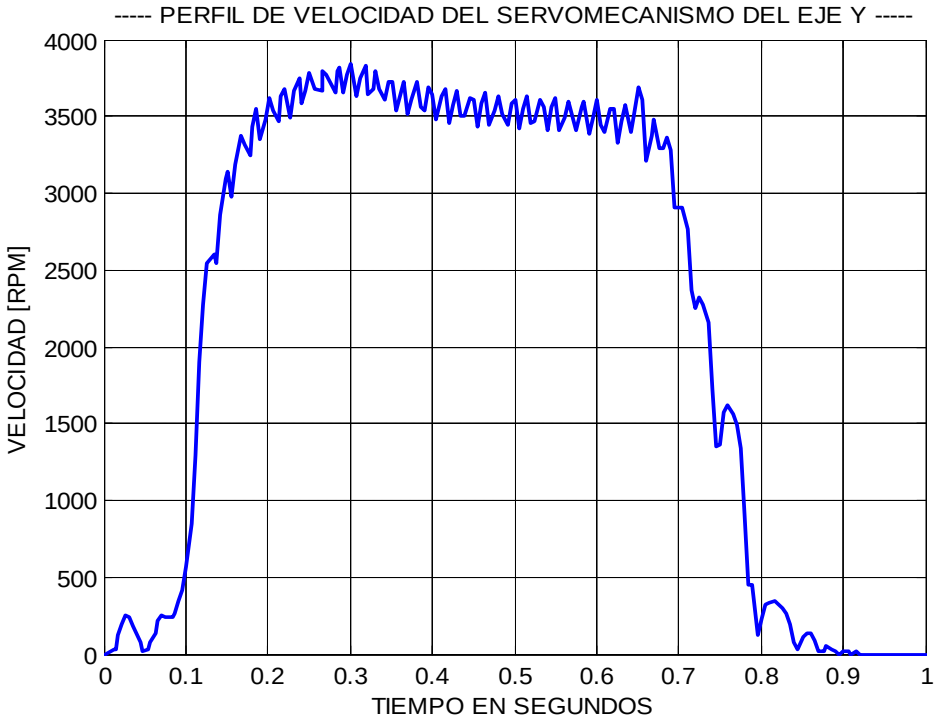


Figura 5.11 Perfil de velocidad del control de posición, servomecanismo del eje Y.

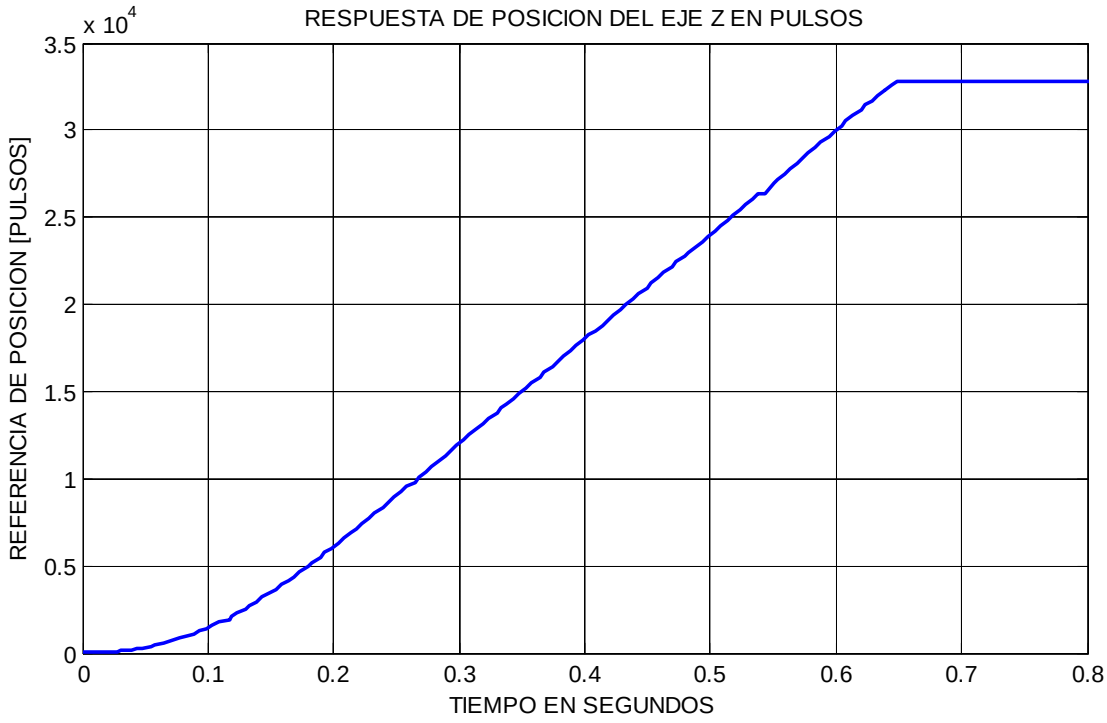


Figura 5.12. Respuesta en lazo cerrado a un comando de posición para el eje Z

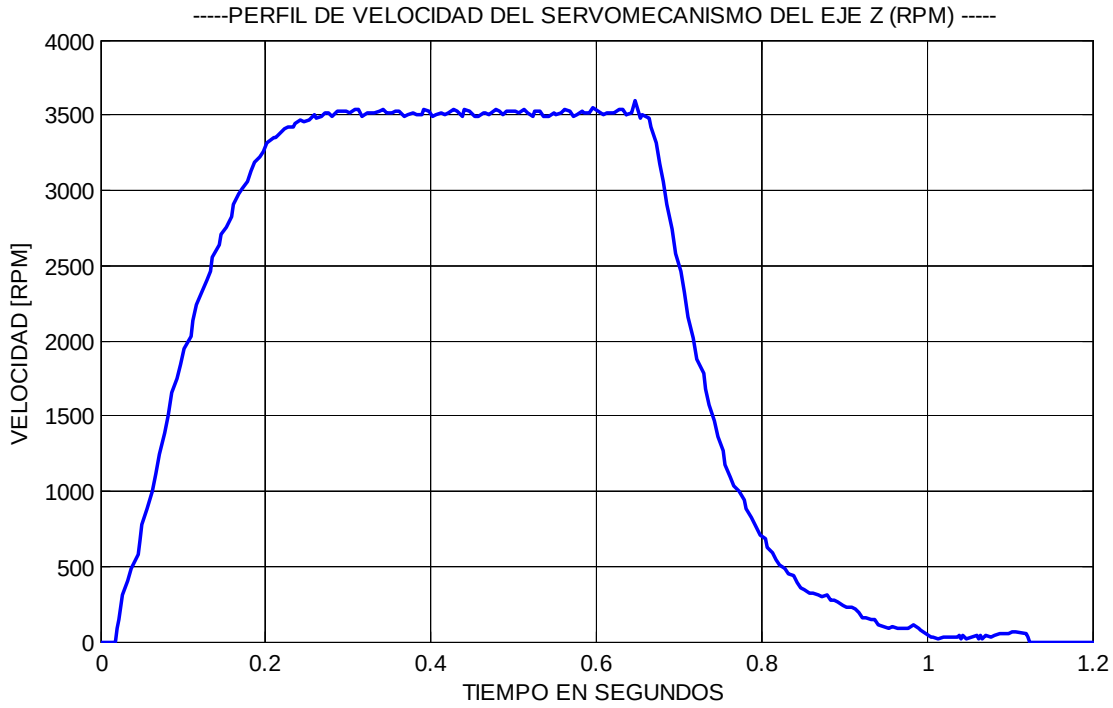


Figura 5.13 Perfil de velocidad del control de posición, servomecanismo del eje Z.

En todas las pruebas de posición para los ejes X, y Z, se utilizó una referencia de posición en pulsos de 32767, esto es debido a que como se está trabajando con una variable de tipo entero con signo, el máximo comando de posición que se le puede enviar al microcontrolador es de 32767, pero se tiene la posibilidad de extenderlo uno más grande, sabiendo que con los 32767 pulsos el eje en cuestión se mueve alrededor de 0.04 cm; para lograr esto se toma un factor de repetición, es decir, repetir el comando tantas veces hasta alcanzar la posición deseada como se explica en la sección 4.4.1.

Como se puede observar, el tiempo de establecimiento de posición es el mismo para todos los servomecanismos, lo que dice que el control de posición es muy eficiente, además es posible establecer un comando de posición mayor al que puede soportar el microcontrolador para moverse por toda el área de trabajo.

5.5 Pruebas del manipulador usado como rotulador

Se probó el sistema completo trabajando como rotulador, trazando una figura deseada desde la interfaz gráfica en Matlab, y observando cómo se reproducía dicha figura por el manipulador trabajando como rotulador. En la figura 5.14, se muestra la figura a trazar y en la figura 5.15 se muestra dicha figura trazada sobre una superficie de madera tipo MDF (Fibra de Media Densidad), observando así una reproducción idéntica a la deseada, comprobando con esto el funcionamiento del sistema manipulador y observando que la precisión es bastante buena porque ejecuta el movimiento en un punto y lo termina en el mismo punto al tratarse de una figura cerrada.

El tipo de fresa utilizada fue de 1/16 pulgadas, aunque al utilizar una fresadora de más bajo diámetro (1/32 pulgadas), se obtuvieron resultados bastante malos, esto debido a que la fresadora pequeña, no desbasta de manera uniforme sobre la madera MDF, y lo que consigue es abrirla.

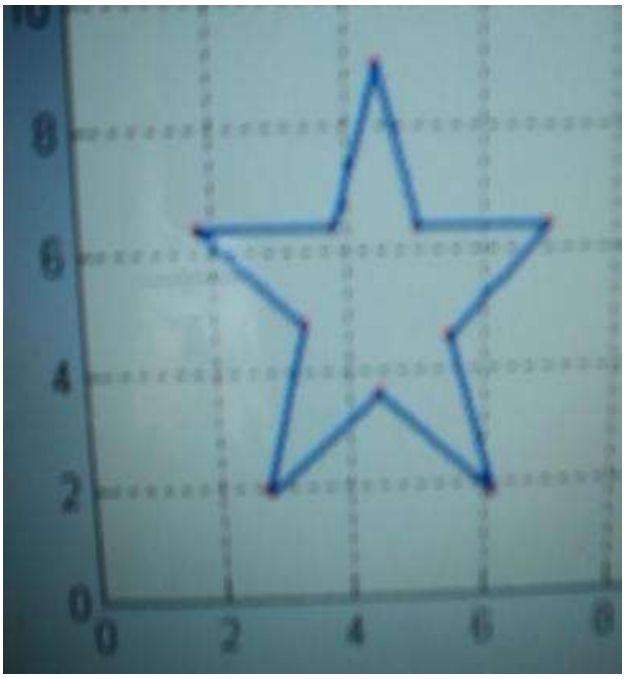


Figura 5.14 Figura a trazar por el manipulador.

La figura que se reproduce (Figura 5.15) es idéntica a la deseada, con ello es posible establecer la precisión del sistema para trazar lo que se desee en una superficie de madera, con esta prueba comprobamos que el sistema funciona y se da pie para poderlo continuar en aplicaciones o trabajos futuros.

Figura 5.15 Figura trazada sobre una superficie de madera.

La capacidad del sistema manipulador permite también hacer trazos de letras mediante líneas continuas, prueba de ello se muestra en la figura 5.16, donde se dibujó un nombre rotulado sobre la superficie de madera, utilizando una copia del tipo de letra cursiva en la

interfaz trazada desde una entrada táctil (Lápiz USB) con un total de 98 puntos que forman líneas rectas como se requiere para el manipulador, y se muestra el resultado rotulado en la figura 5.17. Cabe mencionar que se dibujó el letrero completo partiendo de un tipo de letra cursiva tomado como referencia, observando que además de que la precisión del sistema es buena, el acabado también arroja resultados bastante buenos.

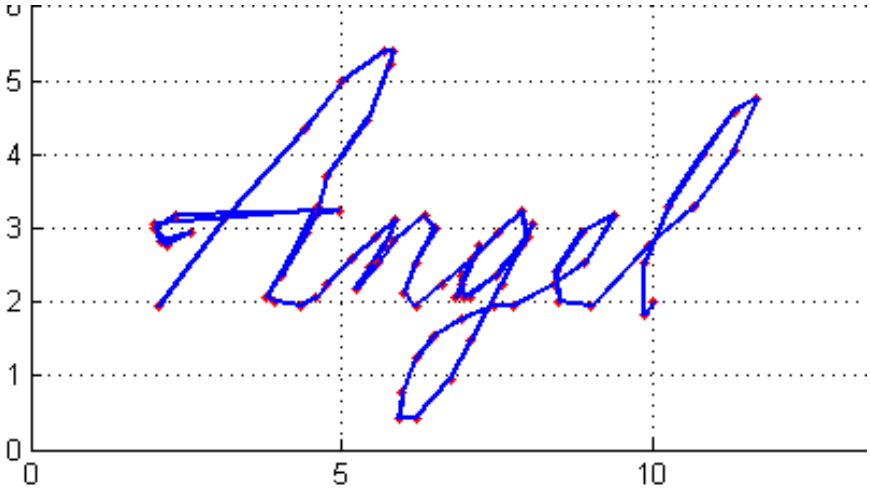


Figura 5.16 Letrero a rotular por el manipulador.

Figura 5.17 Letrero con tipo de letra cursiva, trazado con el manipulador.

Los resultados anteriores muestran el rotulador trazando líneas continuas sin espacios entre unas y otras, es posible también trabajarlo con trazos separados, por ejemplo, en un letrero, trazar letra por letra y no ser un tipo de letra cursivo. Las figuras 5.18 a 5.20 muestran dos ejemplos para ello, un dibujo que contiene diferentes trazos, y en la figura 5.20 un letrero formado por trazos aparte, una letra por trazo, basta con hacer un trazo y repetir el

procedimiento de ese trazo para el siguiente, desde ir a posición de origen hasta terminar el trazo, así para el numero de trazos que contenga.

Figura 5.18 Figura con varios trazos

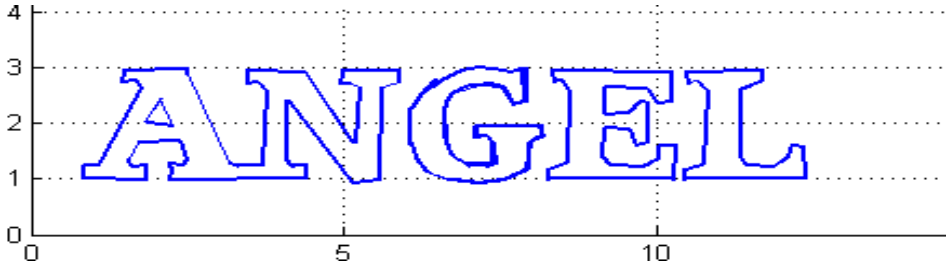


Figura 5.19 Letrero con varios trazos a rotular.

Figura 5.20 Resultado del letrero rotulado

Figura 5.21 Detalle del letrero rotulado.

Como se puede observar a detalle en la figura 5.21, el rotulado del manipulador corresponde a un trazo por cada letra de la palabra a rotular.

En este capítulo se presentaron las pruebas del manipulador, primero por separado cada eje, montado sobre el sistema, se probaron los controladores de posición y velocidad, y después en conjunto como rotulador sobre madera, trabajando los 3 ejes simultáneamente, teniendo buenas respuestas del sistema y mostrando la versatilidad que ofrece.

En el capítulo 6 se presentan las conclusiones que deja este trabajo, así como los trabajos que en un futuro pudieran desarrollarse a partir de la información contenida en esta tesis.

Capítulo 6

Conclusiones

6.1 Conclusiones generales

En este proyecto se planteó como objetivo el control de un manipulador con dos grados de libertad basado en microcontroladores PIC, para ser utilizado como rotulador sobre madera, incluyendo interfaz con el usuario, controladores de velocidad y posición de los servomecanismos.

El desarrollo del trabajo cumplió con los objetivos planteados, rindiendo varios frutos y detalles que en un principio se desconocían, como son el algoritmo de control de posición, el esquema de control de velocidad IP, las capacidades de los microcontroladores 18fxx31 y sus módulos de movimiento.

Se presentaron todos los aspectos electrónicos del sistema, se describió la estructura de la base y como se partió de la estructura mecánica para diseñar las tabletas de los drivers de cada servomecanismo, también se desarrollo el software de los controladores.

Se destacó también la alta capacidad de los microcontroladores, uno de ellos para la comunicación USB y el otro para la medición de velocidad y posición de motores como elemento básico del controlador de cada servomecanismo. En el diseño de los drivers de cada servomecanismo se incluyó un convertidor de CD a CD con esquema PWM tipo puente completo, se destacó la importancia de la comunicación entre microcontroladores mediante comunicación I2C bidireccional y se describieron los algoritmos de medición de velocidad y posición. Se describió también un algoritmo particular de control de posición que incluye un esquema de control de velocidad IP digital. Se planteó un procedimiento sencillo y práctico para la identificación del modelo de cada uno de los servomecanismos como un modelo discreto, y se diseñaron sus ganancias de cada uno de ellos, logrando buenos resultados, mostrados en las simulaciones.

La estructura del manipulador permite la implementación de otros tipos de controladores sin la necesidad de modificar el hardware.

También se destacó la gran versatilidad del software MATLAB para desarrollo de interfaces gráficas GUI y comunicación Serial/USB. La facilidad de desarrollar interfaces gráficas mediante Matlab ofrece un paquete completo de adquisición de datos para el control de sistemas reales mediante la PC, con Matlab se tiene la capacidad de hacer cálculo numérico, de comunicación mediante puerto serial con Microcontroladores, y de interacción con el usuario mediante interfaces gráficas no solo desde línea de comandos.

El diseño del software modular o por etapas permite un fácil diagnóstico de fallas. El manipulador como rotulador tiene buena precisión, prueba de ello es que al trazar figuras cerradas, el movimiento comenzaba en un punto inicial, y lo terminaba en el mismo punto, las variaciones de error en las líneas para las referencias de posición fueron mínimas, también los trazos fueron líneas rectas, como quedó establecido desde un principio, las rectas fueron logradas debido a los movimientos ejecutados con las velocidades adecuadas para cada eje y así concluir también que la sincronización de velocidades para lograr líneas suaves fue muy buena.

Se realizaron rotulados sobre dos tipos de madera diferentes, madera suave MDF, y madera de pino, sobre la madera MDF se utilizaron puntas fresadoras de 1/16 y 1/32 de pulgada, obteniendo mejores resultados con la punta de 1/16 por tener una superficie más grande y poderse manipular mejor por el sistema, por el contrario, la fresa de 1/32 por ser más pequeña se atora más fácilmente en la superficie de la madera MDF y es más difícil de manipular con el sistema, para evitar esto, se cambia el tipo de madera si se rotulan figuras muy finas, en cambio, si se rotulan figuras de líneas más gruesas o más grandes, se utiliza la punta de fresa mas grande.

Se recomienda utilizar fresas de tamaño medio, entre 1/32 pulgada y 1/8 para tamaños estándar, o 0.8 mm a 1.58 mm para tamaños milimétricos, y para MDF se recomiendan fresas de tamaños mayores a 1/16 de pulgada (1.58 mm).

6.2 Trabajos futuros

Con lo obtenido de este proyecto se proponen los siguientes trabajos futuros:

- Prueba de otros métodos de control de velocidad, para lo cual solamente es necesario transformar la rutina de interrupción del TIMER donde se encuentra la rutina del controlador utilizado.
- Extensión a tres grados de libertad para hacer control de fuerza y poder utilizado como cortador de superficies no lisas.
- Establecer la interpretación de código G, que es un código genérico que utilizan las máquinas de control numérico CNC, y así poder trazar cualquier imagen vectorizada.
- Modificación del código para ser utilizado como perforador de circuitos impresos de manera automática.

Apéndice A

Listado de los programas

A.1. Programa de la interfaz gráfica en Matlab

```
function varargout = interfaz_manipulador(varargin)
% INTERFAZ_MANIPULADOR M-file for interfaz_manipulador.fig
%      INTERFAZ_MANIPULADOR, by itself, creates a new INTERFAZ_MANIPULADOR or raises the existing
%      singleton*.
%
%      H = INTERFAZ_MANIPULADOR returns the handle to a new INTERFAZ_MANIPULADOR or the handle to
%      the existing singleton*.
%
%      INTERFAZ_MANIPULADOR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INTERFAZ_MANIPULADOR.M with the given input arguments.
%
%      INTERFAZ_MANIPULADOR('Property','Value',...) creates a new INTERFAZ_MANIPULADOR or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before interfaz_manipulador_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to interfaz_manipulador_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help interfaz_manipulador

% Last Modified by GUIDE v2.5 18-Jul-2011 19:27:04

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @interfaz_manipulador_OpeningFcn, ...
                   'gui_OutputFcn',  @interfaz_manipulador_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 interfaz_manipulador is made visible.
function interfaz_manipulador_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 interfaz_manipulador (see VARARGIN)

% Choose default command line output for interfaz_manipulador
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are returned to the command line.
function varargout = interfaz_manipulador_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 pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
%INICIALIZA EL PUERTO SERIAL
s = serial('COM4'); %crear el puerto serial
set(s, 'BaudRate', 460800)  %se indica la velocidad de transmicion de datos
```

```
set(s, 'BaudRate', 9600)
set(s, 'DataBits', 7)       %tamaño del buffer 8 bits=1bytes
set(s, 'Parity','none')     % no bits de paridad
set(s, 'StopBits', 1)       % 1 bit de paro
set(s, 'FlowControl', 'none')   % sin control de flujo
set(s, 'TimeOut',10) %segundos maximos para que se llene el buffer
s.InputBufferSize=4;        %palabra a recibir de 5bytes numero decimal
fopen(s) %inicia la conexion serial

flushinput(s);
fprintf(s,'%d',size_of_array);
fprintf(s,'%c',13);
pause(0.01);
fprintf(s,'%d',first_pointx);
fprintf(s,'%c',13);
pause(0.01);
fprintf(s,'%c',first_pointy);
fprintf(s,'%c',13);
pause(0.01);
%
%envio las distancias de X
for i=1:size_of_array
   fprintf(s,'%d',lx(i));
   fprintf(s,'%c',13);
pause(0.001);
end
%envio las distancias de Y
for i=1:size_of_array
   fprintf(s,'%d',ly(i));
   fprintf(s,'%c',13);
    pause(0.001);
end
%envio las velocidades de X
for i=1:size_of_array
   fprintf(s,'%d',velx(i));
   fprintf(s,'%c',13);
   pause(0.001);
end
%envio las velocidades de Y
for i=1:size_of_array
    fprintf(s,'%d',vely(i));
    fprintf(s,'%c',13);
    pause(0.001);
end
clc;  %LIMPIAMOS TERMINAL

fclose(s);           %LIMPIAMOS PUERTO
delete(s);           %LIMPIAMOS VARIABLE
clear s
lxy
guidata(hObject,handles);
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function axes5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes5
global size_of_array lx ly first_pointx first_pointy velx vely
grid on
hold on
axis([0 20 0 20]);
% La lista inicial de puntos es cero
xy = [];
n = 0;
% Loop, picking up the points.
but = 1;
xlabel('Posición X en cm.');
ylabel('Posición Y en cm.');
while but == 1
    [xi,yi,but] = ginput(1);
   % plot(xi,yi,'r.');
    if n>0
        line([xi,xi_last],[yi,yi_last],'LineWidth',2);
    end
    n = n+1;
    xy(:,n) = [xi;yi];
    xi_last=xi;
    yi_last=yi;
end
xy=xy*100;
xy=round(xy);

x_points=xy(1,:);
y_points=xy(2,:);
size_of_array=length(xy)-1;

velx=ones(1,size_of_array)*480;
vely=ones(1,size_of_array)*480;
```

```
for i=1:size_of_array
    lx(i)=x_points(i+1)-x_points(i);
    ly(i)=y_points(i+1)-y_points(i);
%obtengo los factres de velocidad
    if abs(lx(i))>abs(ly(i))
        factor=abs(lx(i))/abs(ly(i));
        vely(i)=480/factor;
    elseif abs(ly(i))>abs(lx(i))
        factor=abs(ly(i))/abs(lx(i));
        velx(i)=480/factor;
    end
%obtengo los datos negativos
    if lx(i) < 0
        lx(i)=-lx(i);
        velx(i)=-velx(i);
    end
    if ly(i) < 0
        ly(i)=-ly(i);
        vely(i)=-vely(i);
    end
end

first_pointx=x_points(1);
first_pointy=y_points(1);

lxy=[lx;ly];        %lxy son las distancias que debe recorrer el motor en X y en Y
velx=round(velx);   %velxy son las velocidades de cada eje
vely=round(vely);   %
hold off

guidata(hObject,handles);

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(gcbf)
```

A.2. Programa del microcontrolador Maestro

```
/*####################################################################
Noviembre, 2011. C_ ANGEL
PROGRAMA DEL MICROCONTROLADOR MAESTRO, ENVIA LOS COMANDOS DE
POSICION Y VELOCIDAD A LOS ESCLAVOS DEL MANIPULADOR
SOFTWARE DESARROLLADO BAJO EL COMPILADOR CCS
####################################################################
*/
#include <18F4550.h>                                                    //LIBRERIAS DEL MICROCONTROLADOR

#fuses HSPLL,NOWDT,NOLVP,USBDIV,PLL5,CPUDIV1,NOPUT,VREGEN,MCLR,NOBROWNOUT,NOPROTECT
#use delay(clock=48000000)                                              //FRECUENCIA 48 MHZ PARA MANEJAR USB

#use i2c(MASTER,SDA=PIN_B0,SCL=PIN_B1,SLOW=50000,FORCE_HW) //MODULO I2C

#include <stdlib.h>                                                     //PARA HACER CONVERSIONES ATOI
#include <usb_cdc.h>                                                    //MODULO USB EN MODO CDC

#bit SSPEN = 0x14.5
#bit SSPOV = 0x14.6
#bit WCOL  = 0x14.7

#define   START_HOME          100
#define START_ROUTING         123
#define FIRST_POINT           135
#define   LISTO               178
#define LINEA_OK              199                                       //COMANDOS DE SINCRONIZACIÓN

#define   DISTANCE            124
#define   VELOCITY            188

#define NUMBER_OF_POINTS      197
#define PUNTOS_LEIDOS         160

void i2c_send(int8 device, int8 data);
void i2c_send16(int8 device, signed int16 data);
void i2c_send_command16(int8 device, int8 command, signed int16 data);
void resetI2C(void);

void i2c_receive(int8 device);
signed int16 get_int16_usb(void);

signed int16 X_ROUTE[128];
signed int16 VELX[128];


signed int16 Y_ROUTE[128];
signed int16 VELY[128];

signed int16 x1=0, y1=0;

signed int16 n=0;
```

```
int8 numBuf=0,bufx=0,bufy=0,bufz=0,i=0,st=0,j=0;

void main()
{
          usb_cdc_init();                         //esta  en  usb_cdc   y  configura  los  baudios, bit paridad etc
          usb_init();

          while(!usb_cdc_connected());           //Espera hasta que haya comunicacion por el puerto

          while(1){
          usb_task();                                                         //REFRESCO DEL USB
                    if(usb_enumerated()){                                     //SI EL DISPOSITIVO ES ENUMERADO
                              n=get_int16_usb();                              //RECIBE EL NUMERO DE PUNTOS

                              x1=get_int16_usb();                             //RECIBE EL PUNTO X1

                              y1=get_int16_usb();                             //RECIBE EL PUNTO Y1

                              for(i=0;i<n;i++){
                                        X_ROUTE[i]=get_int16_usb();   //RECIBE LOS DESPLAZAMIENTOS DE X
                              }
                              //guardo los datos en el arreglo l
                              for(i=0;i<n;i++){
                                        Y_ROUTE[i]=get_int16_usb();   //RECIBE LOS DESPLAZAMIENTOS DE Y
                              }

                              //VELOCIDADES en X
                              for(i=0;i<n;i++){
                                        VELX[i]=get_int16_usb();      //RECIBE LAS VELOCIDADES DE X
                              }
                              //VELOCIDADES EN Y
                              for(i=0;i<n;i++){
                                        VELY[i]=get_int16_usb();      //RECIBE LAS VELOCIDADES DE Y
                              }
                              break;
                    }
          }
          st=0;                                                               //COMIENZA LA MAQUINA DE ESTADOS

          while(1)
     {
                    switch(st){
                              case 0:                                   //Primer evento, manda a todos los motores a home
                              //Z
                                        output_high(PIN_B2);
                                        i2c_send(0x0A,START_HOME);    //Manda comando de POSICION CERO AL EJE Z
                                        numBuf=0;
                                        do{

                                                  i2c_receive(0x0A);            //solicito respuesta
                                        }while(numBuf!='Z');                    //mientras no llegue START_HOME, se queda
                                        delay_ms(150);
                                        output_low(PIN_B2);
                              //X
                                        i2c_send(0x02,START_HOME);              //ENVIA EL COMANDO DE POS. CERO AL EJE X
                                        numBuf=0;
                                        do{
                                                  i2c_receive(0x02);            //Espera hasta LLEGAR A LA PSOICION CERO
                                     }while(numBuf!='X');                 //mientras no llegue al fin de paro, se queda
                              //Y
                                        output_high(PIN_B2);
                                        delay_ms(150);
                                        i2c_send(0x0E,START_HOME);              //ENVIA EL COMANDO DE INICIO AL EJE Y
                                        numBuf=0;
                                        do{
                                                  i2c_receive(0x0E);            //ESPERA HASTA LLEGAR A POSICION CERO Y
                                        }while(numBuf!='Y');
                                        delay_ms(150);
                                        output_low(PIN_B2);
                                        st=1;
                                        break;

                              case 1:                         //SEGUNDO EVENTO, MANDA LOS MOTORES A PRIMER PUNTO

                                        i2c_send_command16(0x02,DISTANCE,x1);             //ENVIA DISTANCIA X1
                                        i2c_receive(0x02);
                                        i2c_send_command16(0x02,VELOCITY,500);            //ENVIA VELOCIDAD CONSTANTE
                                        i2c_receive(0x02);
                                        i2c_send_command16(0x02,NUMBER_OF_POINTS,n);      //ENVIA NUMERO DE PUNTOS
                                        i2c_receive(0x02);
                                        i2c_send(0x02,FIRST_POINT);             //ENVIA COMANDO PARA IR A PRIMER PUNTO
                                        numBuf=0;
                                        do{
                                                  i2c_receive(0x02);            //ESPERA HASTA QUE LLEGUE A PRIMIER PUNTO X
                                         }while(numBuf!='X');           //mientras no llegue a primer punto, se queda
                                        delay_ms(150);

                                        i2c_send_command16(0x0E,DISTANCE,200); //ENVIA DISTANCIA Y1
                                        i2c_receive(0x0E);
                                        i2c_send_command16(0x0E,VELOCITY,500); //ENVIA VELOCIDAD CONSTANTE
                                        i2c_receive(0x0E);
                                        i2c_send_command16(0x0E,NUMBER_OF_POINTS,n);      //ENVIA NUMERO DE PUNTOS A Y
                                        i2c_receive(0x0E);
                                        i2c_send(0x0E,FIRST_POINT);                       //dice a Y ir a primer punto
                                        numBuf=0;
```

```
                                do{
                                        i2c_receive(0x0E);          //ESPERA HASTA QUE LLEGUE A PRIMER PUNTO Y
                                }while(numBuf!='Y');          //mientras no llegue a primer punto, se queda
                                delay_ms(150);

                                i2c_send(0x0A,FIRST_POINT);   //DICE A Z IR A PRIMER PUNTO (BAJAR LA HERRAMIENTA)
                                numBuf=0;
                                do{
                                        i2c_receive(0x0A);          //ESPERA HASTA QUE BAJE LA HERRAMIENTA
                                }while(numBuf!='Z');
                                delay_ms(150);

                                st=2;
                                break;

                        case 2:                                 //TERCER EVENTO, COMIENZA EL TRAZADO
                                for(i=0;i<n;i++){
                                    i2c_send_command16(0x02,DISTANCE,X_ROUTE[i]);   //ENVIA DESPLAZAMIENTO Xi
                                        i2c_receive(0x02);
                                        i2c_send_command16(0x02,VELOCITY,VELX[i]);//ENVIA VELOCIDAD VELXi
                                        i2c_receive(0x02);

                                        i2c_send_command16(0x0E,DISTANCE,Y_ROUTE[i]);//ENVIA DESPLAZAMIENTO Yi
                                        i2c_receive(0x0E);
                                        i2c_send_command16(0x0E,VELOCITY,VELY[i]);  //ENVIA VELOCIDAD VELYi
                                        i2c_receive(0x0E);

                                        i2c_send(0x0A,START_ROUTING);

                                        i2c_send(0x02,START_ROUTING); //ENVIA COMANDOS DE INICIO A LOS EJES X,Y,Z

                                        i2c_send(0x0E,START_ROUTING);

                                        numBuf=0;
                                        bufx=0;
                                        bufy=0;
                                        do{
                                                i2c_receive(0x02);
                                                bufx=numBuf;
                                        i2c_receive(0x0E);          //ESPERA HASTA QUE Y e Y LE RESPONDAN
                                bufy=numBuf;
                                }while((bufx&bufy)!='A');     //MIENTRAS NO TERMINEN AMBOS, UNO SE QUEDA ESPERANDO
                                }
                        i2c_send(0x0A,LINEA_OK);       //ENVIA COMNANDO AL EJE Z DE QUE SE HA TERMINADO EL TRAZO
                                delay_us(2);
                                st=3;                   //FINALIZA.
                                break;
                }
    }
}

void i2c_send(int8 device, int8 data) {
   i2c_start();
   i2c_write(device); //write address
   i2c_write(data);  //write data
   i2c_stop();
}

void i2c_send16(int8 device, signed int16 data) {
   i2c_start();
   i2c_write(device);
   i2c_write((int8)(data>>8));
   i2c_write((int8)data);                               //Parte Baja
   i2c_stop();
}
void i2c_send_command16(int8 device, int8 command, signed int16 data) {
   i2c_start();
   i2c_write(device);
   i2c_write(command);
   i2c_write((int8)(data>>8));
   i2c_write((int8)data);
   i2c_stop();
}
void i2c_receive(int8 device) {

      i2c_start();
      i2c_write(device+1);
      numBuf = i2c_read(0);
      i2c_stop();
   }

void resetI2C() {
          SSPEN = 0;   // disable i2c
          SSPOV = 0;   // clear the receive overflow flag
          WCOL = 0;    // clear the write collision flag
          SSPEN = 1;   // re-enable i2c
}

signed int16 get_int16_usb() {
  char s[7];
  signed int16 j;
  get_string_usb(s, 7);
  j=atol(s);
  return j;
}
```

A.3. Programas de los microcontroladores esclavos

A.3.1. Programa del microcontrolador que controla el eje X

```
/*################################################################
Noviembre, 2011. C_ ANGEL
PROGRAMA QUE CONTROLA EL EJE X DEL MANIPULADOR
SOFTWARE DESARROLLADO BAJO EL COMPILADOR CCS
################################################################
*/
#include <18f2431.h>                                                    //LIBRERIA DEL PIC UTILIZADO

#device high_ints=TRUE                                                  //SE  UTILIZARÁN  INTERRUPCIONES  DE  ALTA
PRIORIDAD
#fuses  HS,NOWDT,NOPROTECT,NOFCMEN                      //BITS DE CONFIGURACION
#use delay(clock=20000000)                                      //CRISTAL DE CUARZO, 20 MHZ

#use i2c(SLAVE,SDA=PIN_C4,SCL=PIN_C5,ADDRESS=0x02,SLOW=50000,FORCE_HW)          //MODULO  I2C,PINES,  VELOCIDAD,
DIRECCIONAMIENTO DEL MICRO

#include "Tabla2.c"                                                    //DEL  ARCHIVO  TABLA.C  PARA  CONOCER  LAS
REFERENCIAS DE VELOCIDAD

#USE FAST_IO(A)                                                                 //
#USE FAST_IO(B)                                                 //DIRECTIVAS QUE OPTIMIZAN LAS ENTRADAS Y SALIDAS DE
LOS PUERTOS A, B Y C
#USE FAST_IO(C)                                                                 //

#byte POSCNTH       = 0xF67                                     // BYTE ALTO DEL REGISTRO DE POSICION CAP2BUFL
#byte POSCNTL       = 0xF66                                     // BYTE BAJO DEL REGISTRO DE POSICION CAP2BUFL

#byte       QEICON   = 0xFB6                            //REGISTRO QEICON DEL CODIFICADOR DE CUADRATURA
#byte       CAP1CON  = 0xF63                            //REGISTRO DE CAPTURA DE PULSOS DEL ENCODER
#bit                UP_DOWN    = QEICON.5                           //BIT DE DIRECCIÓN DEL REGISTRO QEICON, LEE
LA DIRECCIÓN DEL EJE DEL MOTOR

#define             T         0.0025                                        //
#define   Ts        0.005                                         //PERIODO DE MUESTREO
#define   kid       28                                                    //GANANCIA  INTEGRAL  DIGITAL  DEL
MICROCONTROLADOR
#define   kpd       159                                                   //GANANCIA  PROPORCIONAL  DIGITAL
DEL MICROCONTROLADOR

//##########  VARIABLES  ###################################################################
int8 dir,buff[3]={0,0},com=0,state=0,command=0;

//COMANDOS CREADOS PARA EL CONTROL DE LA COMUNICACION I2C Y LECTURA DE DATOS
#define START_HOME          100             //COMANDO DE IR A POSICION DE ORIGEN
#define   START_ROUTING     123             //COMANDO DE COMENZAR TRAZADO
#define FIRST_POINT         135             //COMANDO DE IR A PRIMER PUNTO
#define   LISTO             178             //COMANDO DE TERMINACIÓN (NO UTILIZADO)
#define LINEA_OK            199             //COMANDO DE FIN DE TRAZOS (SE ENVIA SOLO AL EJE Z)
#define DISTANCE            124             //COMANDO DE QUE QUE INDICA RECEPCION DE DISTANCIA
#define VELOCITY            188                             //COMANDO DE QUE SE INDICA RECEPCION DE VELOCIDAD

#define NUMBER_OF_POINTS 197                //COMANDO QUE INDICA RECEPCION DE NUMERO DE PUNTOS
#define PUNTOS_LEIDOS    160                //COMANDO DE QUE YA SE HAN LEIDO LOS PUNTOS (NO UTILIZADO)

//#####  MAS VARIABLES ###########################################################################
char k=0,st=0;
unsigned int16 posAc=0,posAnt=0,conv1=0,conv=0, factor=0,deci_mm=0;
int16 velAc=0,velAnt=0,i=0,j=0;

signed int32 error_prop=0,PROP=0,INTE=0,error_inte=0,u=0;
signed int16 Ref=0,Ref1=0,Posi=0,RefP=0,RefP1=0;

unsigned int16 distancia=0;
signed int16 speed=0,number=0;
int16 n=0;

// ### DEFINICION DE FUNCIONES UTILIZADAS #####
void HOME_POSITION(void);
        //FUNCION PARA IR A POSICION DE ORIGEN
void CONTROL_IP_VELOCIDAD(void);                                                                        //FUNCION  PARA
EL CONTROLADOR IP DIGITAL
void setPDC0(signed int16);
        //FUNCION PARA MANDAR COMANDO PWM A LA SALIDA DE LOS PINES RB0 Y RB1.

void MOVIMIENTO_X(unsigned int16 distancia, signed int16 speed);//FUNCION PARA MOVER EL EJE A CIERTA DISTANCIA Y A CIERTA
VELOCIDAD

/*#############################################################################################*/
// RUTINA DE INTERRUPCIÓN ALTA PRIORIDAD, DEL MODULO I2C, SE ACTIVA EL FLAG CUANDO HAY UN DATO EN EL BUFFER
#INT_SSP HIGH
void ssp_isr ()
{
        state = i2c_isr_state();
        //REVISA EL ESTADO DEL BUS
        if(state < 0x80)     // Master is sending data                                     //SI ESTADO ES MENOR QUE
0x80, SIGNIFICA QUE
         {
                                        //EL MICROCONTROLADOR MAESTRO ESTÁ ENVIANDO DATOS
```

```
                if(state == 0){                                         dir=i2c_read();}                    //EN
EL PRIMER ENVIO SE RECIBE LA DURECCIÓN
                if(state == 1){                                         command=i2c_read();}                //EN
EL SEGUNDO ENVIO RECIBE EL COMANDO DE MOVIMIENTO
                if(state >= 2 && state < 4){  buff[state-2]=i2c_read();}   //DEL  3  AL  4  ESTADO  RECIBE  EL  DATO  DE
POSICION O VELOCIDAD DE 16 BITS
        }
    // Master is requesting data from slave
        if(state == 0x80){
                //SI EL ESTADO ES 0x80, significa que
                switch(command){
                //EL MICROCONTROLADOR MAESTRO ESTÁ SOLICITANDO DATOS
                        case      DISTANCE:
                //SI EL COMANDO INDICA QUE LO QUE LLEGO FUE DISTANCIA
                                        distancia=make16(buff[0],buff[1]);                              //LO
GUARDA EN LA VARIABLE
                                        break;
                        case      VELOCITY:
                //SI EL COMANDO INDICA QUE LO QUE LLEGO FUE VELOCIDAD
                                        speed=make16(buff[0],buff[1]);
        //LO GUARDA EN LA VATIABLE

                                        break;
                        case      NUMBER_OF_POINTS:
        //SI EL COMANDO INDICA QUE LO QUE LLEGO FUE numero de puntos
                                        number=make16(buff[0],buff[1]);
        //GUARDA EL NUMERO DE PUNTOS
                                        n=(unsigned int16)number;
        //HACE UN CATING PARA HACER N POSITIVO

                                        break;
                }
                i2c_write(com);
                        //ESCRIBE COMANDO AL MAESTRO.
        }
}
/*#############################################################################################3#####*/
/*
   RUTINA PRINCIPAL
*/
void main()
{
        int16 period=415;                               //PERIODO DE 415 PARA ESTABLECER EL PWM A 12 KHZ DE FRECUENCIA
    SET_TRIS_A(0b00011101);           //CONFIGURACION DE LOS PINES A0, A2, A3, A4 COMO ENTRADA PARA EL ENCODER
        SET_TRIS_B(0b00010000);                         //CONFIGURACION DEL PIN B4 COMO ENTRADA PARA EL LIMITE DE PARO

        setup_power_pwm_pins(PWM_COMPLEMENTARY,PWM_OFF,PWM_OFF,PWM_OFF);     //CONFIGURACION  DE  LOS  PINES  B0,B1  DEL  PWM
EN MODO COMPLEMENTARIO
        setup_power_pwm(PWM_CLOCK_DIV_4|PWM_FREE_RUN|PWM_DEAD_CLOCK_DIV_4,1,0,period,0,1,7);  //CONFIGURACION  DEL  PWM  A
12 KHZ

                                                                                                        //TIEMPOS  MUERTOS  DE  3
us, PRESCALER DE 4
        //Frequency = Fosc / (4 * (period+1))
        //para 12 KHZ la maxima cuenta de pwm es 1664
        setup_qei(QEI_MODE_X4,QEI_FILTER_ENABLE_QEA | QEI_FILTER_ENABLE_QEB | QEI_FILTER_DIV_1,1023);
        //CONFIGURACIÓN  DEL  MODULO  DE  CUADRATURA,  EN  MODO  X4,  CON  FILTROS  HABILITADOS,  A  UNA  MAXIMA  CUENTA  DE  1023
PULSOS
        setup_timer_5(T5_INTERNAL | T5_DIV_BY_2);   //CONFIGURA EL TIMER5, RELOJ INTERNO, PRESCALADOR DE 2
        set_timer5(53036);                                                              //PARA  ACTIVARSE  CADA  5  ms  POR
MUESTRA
    enable_interrupts(INT_TIMER5);                                      //HABILITA LA INTERRUPCION DEL TIMER5
        enable_interrupts(INT_IC2QEI);                                  //HABILITA  LA  INTERRUPCION  POR  CADA  VUELTA
DEL ROTOR
        enable_interrupts(INT_SSP);                                     //HABILITA LA INTERRUPCION PARA EL MODULO I2C
        enable_interrupts(GLOBAL);                                      //SE HABILITAN LAS INTERRUPCIONES GLOBALES

        disable_interrupts(INT_TIMER5);                                 //DESHABILITO EL TIMER ANTES DE COMENZAR
        setPDC0(0);                                                                             //APAGA     EL
MOTOR
        com='R';                                                                        //COMANDO    INCIAL    A
ENVIAR AL MASTER

        while(1){
        //                delay_ms(1000);
                        switch(st){
                        case 0:                                                                         //PRIMER
ESTADO
                                output_high(PIN_B3);
                                while(command!=START_HOME);             //ESPERAR COMANDO DE IR A POSICION DE ORIGEN
                                enable_interrupts(INT_TIMER5);          //DESPUES  QUE  LLEGA  POSICION  DE  ORIGEN
ACTIVA LA INTERRUPCION DEL TIMER5
                                com=0;                                  //ESTABLECE COM EN CERO
                                output_low(PIN_B3);
                                HOME_POSITION();                        //EL EJE X VA A SU POSICION DE ORIGEN
                                com='X';  //ENVIA COMANDO AL MAESTRO DE QUE HA LLEGADO A SU POSICION DE ORIGEN
                                st=1;                                   //PASA AL ESTADO SIGUIENTE
                                break;
                        case 1:                                         //SEGUNDO ESTADO
                                output_low(PIN_B3);
                                while(command!=FIRST_POINT);  //ESPERAR COMANDO DE IR A PRIMER PUNTO
                                com=0;                                  //ESTABLECE COM EN CERO POR SI EL MASTER SE
QUEDA CON EL DATO ANTERIOR
                                MOVIMIENTO_X(distancia,speed);          //SE  HACE  UN  MOVIMIENTO  A  DISTANCIA  Y
VELOCIDAD DESEADAS
```

```
                                        com='X';                                //SE ENVIA COMANDO DE QUE SE HA LLEGADO AL
PRIMER PUNTO
                                        st=2;                                   //PASA AL SIGUIENTE ESTADO
                                        break;
                                case 2:                                         //TERCER ESTADO
                                        output_high(PIN_B3);
                                        for(j=0;j<n;j++){                         //CICLO IGUAL AL NUMERO DE LINEAS A TRAZAR
                                                while(command!=START_ROUTING); //ESPERA COMANDO DE COMENZAR TRAZADO
                                                output_low(PIN_B3);
                                                com=0;

                                                if(distancia!=0){   //SI LA DISTANCA A RECORRER ES DIFERENTE DE CERO
                                                        MOVIMIENTO_X(distancia,speed);//EJECUTA EL MOVIMIENTO
                                                        output_high(PIN_B3);          //APAGA LED MONITOR
                                                }
                                                else{     delay_cycles(2);}  //SI   LA   DISTANCIA   ES   CERO,   NO   HAY
MOVIMIENTO,
                                                com='A';            //SIMPLEMENTE MANDA COMANDO DE QUE ya ha terminado
                                                command=0;          //ACTUALIZA COMANDO
                                        }
                                        st=0;                           //PASA AL ESTADO CERO PARA EMPEZAR OTRO TRAZO.
                                        break;
                        }                                               //FIN DE LA MAQUINA DE ESTADOS
                }                                                       //FIN DEL WHILE
}                                                                       //FIN DEL PROGRAMA PRINCIPAL

/* RUTINA DE CONVERSION DEL PWM A DATOS DE PORCENTAJE, RANGO DE -100% A 100%
   DONDE EL NUMERO NEGATIVO INDICA SENTIDO CONTRARIO                            */
void setPDC0(signed int16 value){
        int16 DUTY_CYCLE;                                                                                       //CICLO     DE
TRABAJO COMO ENTERA DE 16 BITS
        DUTY_CYCLE=84*value+8400;                                                                       //
        DUTY_CYCLE=DUTY_CYCLE/10;                                                                       //DUTY=8.4*value+840;
        set_power_pwm0_duty(DUTY_CYCLE);                                                        //MANDA DATO A LA SALIDA EL PWM
}

/* RUTINA DE NTERRUPCION POR CADA VUELTA DEL MOTOR          */
#int_IC2QEI
void  IC2QEI_isr(void)
{
        k++;                                    //HA HABIDO UNA VUELTA DEL MOTOR, INCREMENTA FACTOR DE CUENTA
}

/* RUTINA DE NTERRUPCION CADA 5 ms                                                              */
#int_TIMER5
void TIMER5_isr()
{
        posAc=qei_get_count();                          //OBTIENE LA CUENTA ACTUAL DEL ENCODER
        if(UP_DOWN==1){                                         //SI EL MOTOR GIRA SENTIDO HORARIO
                if((posAc-posAnt)>1023){}     //
                else
                        velAc=posAc-posAnt;             //OBTIENE  LA  VELOCIDAD  MEDIANTE  LA  CUENTA  ACTUAL  MENOS  LA
ANTERIOR
        }
        else{                                                           //EN  CAMBIO,  SI  EL  MOTOR  GIRA  SENTIDO
ANTIHORARIO
                if((posAnt-posAc)>1023){}
                else      velAc=posAnt-posAc; //OBTIENE LA VELOCIDAD MEDIANTE LA CUENTA ANTERIOR MENOS LA ACTUAL
        }
        posAnt=posAc;                                           //LA  POSICION  ANTERIOR  LA  ACTUALIZA  COMO  LA  ACTUAL
PARA LA SIGUIENTE MUESTRA

        CONTROL_IP_VELOCIDAD();                         //HACE EL CONTROL DE VELOCIDAD
        set_timer5(53036);                              //VUELVE A PONER EL TIMER A 5 ms
}

/* RUTINA PARA EL CONTROL IP DE VELOCIDAD                                                       */
void CONTROL_IP_VELOCIDAD(void)
{
        Ref1=Ref;                                               //SI LA REFERENCIA ES NEGATIVA, LA ALMACENA
        if(Ref1<=0){Ref1=-Ref1;}
        error_inte=((signed int32)Ref1-(signed int32)velAc);       //CALCULA EL ERROR INTEGRAL
        error_prop=(signed int32)(velAc);                                                       //CALCULA    EL    ERROR
PROPORCIONAL

        INTE=INTE+error_inte*kid;               //CALCULA EL TERMINO INTEGRAL
        PROP=error_prop*kpd;                            //CALCULA EL TERMINO PROPORCIONAL
        u=u+INTE-PROP;                                          //CALCULA LA ACCION DE CONTROL
        u=u/1000;                                               //ESCALA  SOBRE  1000  POR  LAS  GANANCIAS,  ESTAN  EN
FACTOR x1000

        if(u>=100)          u=100;                              //ANTIWINDUP, NO ES NECESARIO POR EL TIPO DE CONTROL
        else if(u<0)        u=0;                            //MAS SIN EMBARGO, PARA EVITAR SOBREIMPULSOS SE PONE

        u=(signed int)u;                                //HACE LA ACCION DE CONTROL ENTERA DE 8 BITS
        if(Ref<0) u=-u;                                 //SI  LA  REFERENCIA  FUE  NEGATIVA,  LA  ACCION  DE  CONTROL  TAMBIEN
LO ES
        if(Ref==0) u=0;                                         //SI  LA  REFERENCIA  ES  CERO,  LA  ACCION  DE  CONTROL  SE
HACE CERO PARA DETENER EL MOTOR
        velAnt=velAc;                                           //SE ACTUALIZA LA VELOCIDAD ANTERIOR COMO LA ACTUAL.

        setPDC0(u);                                                     //EJECUTA LA ACCIÓN DE CONTROL
}

/* RUTINA DE CONTROL DE POSICIÓN O MOVIMIENTO A UNA VELOCIDAD DESEADA     */
```

```
void MOVIMIENTO_X(unsigned int16 distancia, signed int16 speed)
{
                    deci_mm=distancia;                                  //CALCULO LA DISTANCIA EN DECIMAS DE MILIMETRO, P.
EJ. 2cm SON 200 DECIMAS DE CM
                    factor=deci_mm/2;                                   //CALCULA EL FACTOR DE REPETICION

                    if(speed < 0)       RefP=-5120;                     //SI LA VELOCIDAD ES MENOR A 0,LA REFERENICA DE
POSICION EN PULSOS ES NEGATIVA
                    else                        RefP=5120;                      //DE OTRO MODO, LA REFERENCIA EN PULSOS ES
POSITIVA

                    conv1=0;                                                    //VARIABLES DE CUENTA DE POSICION
                    conv=0;

                    RefP1=RefP;                                                                 //VARIABLE      AUXILIAR      PARA
ACTUALIZAR LA REFERENCIA DE VELOCIDAD
                    if(RefP<0){         RefP1=-RefP;}                   //SI LA REFERENICA DE VELOCIDAD ES NEGATIVA, LA
VARIABLE AUXILIAR SE HACE POSITIVA SIN ALTERAR SU VALOR
                    for(i=0;i<factor-1;i++){                //SE EJECUTA EL CONTROL DE POSICION CON UN FACTOR DE REPETICON
-1
                            Posi=0;                                             //LIMPIA EL ERROR DE POSICION
                            k=0;                                                //LIMPIA VARIABLE DE FACTOR DE LA
CUENTA
                            while(Posi<=RefP1){                     //MIENTRAS EL ERROR DE POSICION ES MENOR QUE LA
REFERENCIA, ESTAMOS EN LA ETAPA DE ACELERACCION

                                    conv=(unsigned int16)(POSCNTH*256)+(unsigned int16)POSCNTL;          //    LEE    EL
ESTADO DEL ENCODER
                                    if(UP_DOWN==1)      Posi=conv+1023*k;                       //SI SE PASA DE 1023 LA
CUENTA, EL FACTOR DE CUENTA AUMENTA
                                    else                            Posi=(1023-conv)+1023*k;      //  SI  ESTA  GIRANDO  AL
REVES, DISMINUYE
                                    Ref=speed;
                    //LA REFERENCIA ES LA VELOCIDAD MAXIMA
                                    //if(RefP < 0)      Ref=-speed;
                                    //else                          Ref=speed;
                            }
                    }
                    Posi=0;                                                                     //LIMPIA EL ERROR DE POSICION
                    k=0;                                                                        //LIMPIA VARIABLE DE FACTOR DE LA
CUENTA

                    while((Posi<(RefP1-1000))&& (RefP1>1000)){      //MIENTRAS  EL  ERROR  ESTÉ  DENTRO  DEL  RANGO  DE
POSICION, SIGUE CON AL MISMA VELOCIDAD
                            conv1=conv;
                            conv=(unsigned int16)(POSCNTH*256)+(unsigned int16)POSCNTL;
                            if(UP_DOWN==1)      Posi=conv+1023*k;
                            else                            Posi=(1023-conv)+1023*k;
                            Ref=speed;
                            //if(RefP < 0)      Ref=-speed;
                            //else                          Ref=speed;
                    }
                    while(Posi>((RefP1-1000)||(RefP1<1000))&&(Posi<RefP1)){             //MIENTRAS  EL  ERROR  DE  POSICION
SEA MENOOR QUE EL RANGO ESTABLECIDO,
                            conv=(unsigned int16)(POSCNTH*256)+(unsigned int16)POSCNTL; //HAY UNA DESACELERACION
                            if(UP_DOWN==1)      Posi=conv+1023*k;
                            else                            Posi=(1023-conv)+1023*k;
                            if(RefP < 0)        Ref=-Tabla[RefP1-Posi];
          //LA REFERENICA DE VELOCIDAD DISMINUYE GRADUALMENTE
                            else                            Ref=(Tabla[RefP1-Posi]);
          //SI LA REFERENCIA DE POSICION FUE NEGATIVA, LA REFERENCIA
                    }
                                                                        //DE VELOCIDAD TAMBIEN LA HACE NEGATIVA
                    Posi=0;                                                             //LIMPIA EL ERROR DE POSICION
                    RefP=0;                                                             //LIMPIA LA REFERENCIA DE POSICION
                    velAc=0;                                                    //LIMPIA LAS VELOCIDADES ACTUAL
                    velAnt=0;                                                   //Y ANTERIOR
                    Ref=0;                                                              //PONE REFERENICA DE VELOCIDAD EN CERO
                    k=0;                                                                //PONE FACTOR DE CUENTA EN CERO
                    setPDC0(0);                                                 //APAGA EL MOTOR.
}

/* RUTINA DE POSICION DE ORIGEN */
void HOME_POSITION(void)
{
          while(input(PIN_B4)){
                    Ref=-700;                   //SE QUEDA EN UNA REFERENICA DE VELOCIDAD NEGATIVA HASTA QUE LLEGA A LA
POSICION CERO
          }                                                             //UNA VEZ ACTIVADO EL SENSOR
          Posi=0;                                               //LIMPIA EL ERROR DE POSICION
          RefP=0;                                               //LIMPIA LA REFERENCIA DE POSICION
          Ref=0;                                                //PONE REFERENICA DE VELOCIDAD EN CERO
          k=0;                                                  //PONE FACTOR DE CUENTA EN CERO
}
```

A.3.2. Programa del microcontrolador que controla el eje Y

```
/*####################################################################
Noviembre, 2011. C_ ANGEL
PROGRAMA QUE CONTROLA EL EJE Y DEL MANIPULADOR
SOFTWARE DESARROLLADO BAJO EL COMPILADOR CCS
####################################################################
*/
#include <18f2431.h>

#fuses  HS,NOWDT,NOPROTECT,NOFCMEN
#use delay(clock=20000000)

#use i2c(SLAVE,SDA=PIN_C4,SCL=PIN_C5,ADDRESS=0x0E,SLOW=50000,FORCE_HW)

#include "Tabla2.c"

#USE FAST_IO(A)
#USE FAST_IO(B)
#USE FAST_IO(C)

#byte POSCNTH       = 0xF67  // Position register HIGH byte CAP2BUFL
#byte POSCNTL       = 0xF66  // Position register low byte CAP2BUFL

#byte       QEICON   = 0xFB6
#byte       CAP1CON  = 0xF63
#bit                UP_DOWN    = QEICON.5

#define             T         0.0025
#define   Ts        0.005
#define   kid       28
#define   kpd       159

//###################################################
int8 dir,buff[3]={0,0},com=0,state=0,command=0;

//COMANDOS DE HOME, Y LECTURA DE DATOS
#define START_HOME            100
#define   START_ROUTING       101
#define FIRST_POINT           102
#define   LISTO                         103
#define LINEA_OK              104
#define DISTANCE              105
#define VELOCITY              106

#define NUMBER_OF_POINTS 107
#define PUNTOS_LEIDOS    108
//###################################################

char k=0,st=0;
unsigned int16 posAc=0,posAnt=0,conv1=0,conv=0, factor=0,deci_mm=0;
int16 velAc=0,velAnt=0,i=0,j=0;

signed int32 error_prop=0,PROP=0,INTE=0,error_inte=0,u=0;
signed int16 Ref=0,Ref1=0,Posi=0,RefP=0,RefP1=0;

unsigned int16 distancia=0;
signed int16 speed=0,n=0;

//void SEND_TO_LABVIEW(int16);
void HOME_POSITION(void);
void CONTROL_IP_VELOCIDAD(void);
void setPDC0(signed int16);
void MOVIMIENTO_X(unsigned int16 distancia, signed int16 speed);

/*#########################################################*/
#INT_SSP
void ssp_isr ()
{
          state = i2c_isr_state();
          if(state < 0x80)     // Master is sending data
           {
                    if(state == 0){                                          dir=i2c_read();}
                    if(state == 1){                                          command=i2c_read();}

                    if(state >= 2 && state < 4){  buff[state-2]=i2c_read();}    //recibe dato
             }
    // Master is requesting data from slave
          if(state == 0x80){
                    switch(command){
                              case      DISTANCE:
                                        distancia=make16(buff[0],buff[1]);
                                        break;
                              case      VELOCITY:
                                        speed=make16(buff[0],buff[1]);
                                        break;
                              case      NUMBER_OF_POINTS:
                                        n=make16(buff[0],buff[1]);
                                        break;
                    }
                    i2c_write(com);
          }
}
/*#########################################################*/
void main()
```

```
{
          int16 period=276;                                  //Period de 415 para 12 kHz
    SET_TRIS_A(0b00011101);            //Port A I/O Config
          SET_TRIS_B(0b00010000);                            //configuro RB4 como entrada para el limite de paro

          setup_power_pwm_pins(PWM_COMPLEMENTARY,PWM_OFF,PWM_OFF,PWM_OFF);
          setup_power_pwm(PWM_CLOCK_DIV_4|PWM_FREE_RUN|PWM_DEAD_CLOCK_DIV_4,1,0,period,0,1,7); //Frecuencia de 10 KHz

          //Frequency = Fosc / (4 * (period+1) * postscale)
          //para 12 KHZ la maxima cuenta de pwm es 1664

          setup_qei(QEI_MODE_X4,QEI_FILTER_ENABLE_QEA | QEI_FILTER_ENABLE_QEB | QEI_FILTER_DIV_1,1023);

          setup_timer_5(T5_INTERNAL | T5_DIV_BY_2);   //Reloj interno 5MHz  para 5 ms por muestra
          set_timer5(53036);
    enable_interrupts(INT_TIMER5);
          enable_interrupts(INT_IC2QEI);
          enable_interrupts(INT_SSP);     //Activamos la interrupcion para el i2c
          enable_interrupts(GLOBAL);

          setPDC0(0);
          com='R';
//        Ref=-500;
          while(1){
                    switch(st){
                              case 0:
                                        while(command!=START_HOME);
                                        HOME_POSITION();              //voy a home position
                                        com='H';                                //envio comando de que estoy listo
                                        st=1;
                                        break;
                              case 1:
                                        while(command!=FIRST_POINT);
                                        MOVIMIENTO_X(distancia,speed);
                                        com='P';
                                        st=2;
                                        break;
                              case 2:
                                        for(j=0;j<4;j++){
                                                  while(command!=START_ROUTING);
                                                  com=0;
                                                  MOVIMIENTO_X(distancia,speed);
                                                  com='L';
                                                  command=0;
                                        }
                                        st=0;
                                        break;
                    }
          }
}

void setPDC0(signed int16 value){
          int16 DUTY_CYCLE;
          DUTY_CYCLE=5.56*value+556;
//        DUTY_CYCLE=78*value+8400;
//        DUTY_CYCLE=DUTY_CYCLE/10;
          set_power_pwm0_duty(DUTY_CYCLE);
}

#int_IC2QEI
void  IC2QEI_isr(void)
{
          output_toggle(PIN_B2);
          k++;
}

#int_TIMER5
void TIMER5_isr()
{
          output_toggle(PIN_B3);
          posAc=qei_get_count();
          if(UP_DOWN==1){
                    if((posAc-posAnt)>1023){}
                    else
                              velAc=posAc-posAnt;
          }
          else{
                    if((posAnt-posAc)>1023){}
                    else      velAc=posAnt-posAc;
          }
          posAnt=posAc;

          CONTROL_IP_VELOCIDAD();

          set_timer5(53036);
}

void CONTROL_IP_VELOCIDAD(void)
{
/////// CONTROL DE VELOCIDAD /////////
          Ref1=Ref;
          if(Ref1<=0){Ref1=-Ref1;}
          error_inte=((signed int32)Ref1-(signed int32)velAc);
          error_prop=(signed int32)(velAc);
```

```
          INTE=INTE+error_inte*kid;
          PROP=error_prop*kpd;
          u=u+INTE-PROP;
          u=u/1000;
          if(u>=100)          u=100;
          else if(u<0)        u=0;

          u=(signed int)u;
          if(Ref<0) u=-u;
//        if(Ref==0) u=0;
          velAnt=velAc;

          setPDC0(u);
}

void MOVIMIENTO_X(unsigned int16 distancia, signed int16 speed)
{
                    deci_mm=distancia;
                    factor=deci_mm/2;   //50 mm a moverse

                    if(speed < 0)       RefP=-5120;                   //25600;  //5120 para resolucion de 0.2 mm en x y Y,
5954 para eje Z
                    else                          RefP=5120;

                    conv1=0;
                    conv=0;

                    RefP1=RefP;
                    if(RefP<0){         RefP1=-RefP;}
                    for(i=0;i<factor-1;i++){
                              Posi=0;
                              k=0;
                              while(Posi<=RefP1){

                                        conv=(unsigned int16)(POSCNTH*256)+(unsigned int16)POSCNTL;
                                        if(UP_DOWN==1)      Posi=conv+1023*k;
                                        else                          Posi=(1023-conv)+1023*k;
                                        Ref=speed;
                                        //if(RefP < 0)      Ref=-speed;
                                        //else                        Ref=speed;
                              }
                    }
                    Posi=0;
                    k=0;

                    while((Posi<(RefP1-1000))&& (RefP1>1000)){
                              conv1=conv;
                              conv=(unsigned int16)(POSCNTH*256)+(unsigned int16)POSCNTL;
                              if(UP_DOWN==1)      Posi=conv+1023*k;
                              else                          Posi=(1023-conv)+1023*k;
                              Ref=speed;
                              //if(RefP < 0)      Ref=-speed;
                              //else                        Ref=speed;
                    }
                    while(Posi>((RefP1-1000)||(RefP1<1000))&&(Posi<RefP1)){
                              conv=(unsigned int16)(POSCNTH*256)+(unsigned int16)POSCNTL;
                              if(UP_DOWN==1)      Posi=conv+1023*k;
                              else                          Posi=(1023-conv)+1023*k;
                              if(RefP < 0)        Ref=-Tabla[RefP1-Posi]*-Ref/1000;
                              else                          Ref=(Tabla[RefP1-Posi]*Ref/1000);
                    }
                    Posi=0;
                    RefP=0;
                    velAc=0;
                    velAnt=0;
                    Ref=0;
                    k=0;
//                  setPDC0(0);
}

void HOME_POSITION(void)
{
          while(input(PIN_B4)){
                    Ref=-550;
          }
          Posi=0;
          RefP=0;
          Ref=0;
          k=0;
}
```

A.3.3. Programa del microcontrolador que controla el eje Z

```
/*####################################################################
Noviembre, 2011. C_ ANGEL
PROGRAMA QUE CONTROLA EL EJE Z DEL MANIPULADOR
SOFTWARE DESARROLLADO BAJO EL COMPILADOR CCS
####################################################################
*/
#include <18f2431.h>

#fuses  HS,NOWDT,NOPROTECT,NOFCMEN
#use delay(clock=20000000)

#use i2c(SLAVE,SDA=PIN_C4,SCL=PIN_C5,ADDRESS=0x0A,SLOW=50000,FORCE_HW)

#include "Tabla2.c"

#USE FAST_IO(A)
#USE FAST_IO(B)
#USE FAST_IO(C)

#byte POSCNTH       = 0xF67  // Position register HIGH byte CAP2BUFL
#byte POSCNTL       = 0xF66  // Position register low byte CAP2BUFL

#byte       QEICON   = 0xFB6
#byte       CAP1CON  = 0xF63
#bit                UP_DOWN    = QEICON.5

#define             T         0.0025
#define   Ts        0.005
#define   kid       28
#define   kpd       159

#define   dremmel   PIN_B5
//#####################################################
int8 dir,buff[3]={0,0},com=0,state=0,command=0;

//COMANDOS DE HOME, Y LECTURA DE DATOS
#define START_HOME           100
#define   START_ROUTING      101
#define FIRST_POINT          102
#define   LISTO                         103
#define LINEA_OK             104
#define DISTANCE             105
#define VELOCITY             106

#define NUMBER_OF_POINTS 107
#define PUNTOS_LEIDOS    108
//#####################################################

char k=0,st=0;
unsigned int16 posAc=0,posAnt=0,conv1=0,conv=0, factor=0,deci_mm=0;
int16 velAc=0,velAnt=0,i=0,j=0;

signed int32 error_prop=0,PROP=0,INTE=0,error_inte=0,u=0;
signed int16 Ref=0,Ref1=0,Posi=0,RefP=0,RefP1=0;

unsigned int16 distancia=0;
signed int16 speed=0,n=0;

//void SEND_TO_LABVIEW(int16);
void HOME_POSITION(void);
void CONTROL_IP_VELOCIDAD(void);
void setPDC0(signed int16);
void MOVIMIENTO_X(unsigned int16 distancia, signed int16 speed);

/*#########################################################*/
#INT_SSP
void ssp_isr ()
{
          state = i2c_isr_state();
          if(state < 0x80)     // Master is sending data
           {
                    if(state == 0){                                          dir=i2c_read();}
                    if(state == 1){                                          command=i2c_read();}

                    if(state >= 2 && state < 4){  buff[state-2]=i2c_read();}    //recibe dato
            }
    // Master is requesting data from slave
          if(state == 0x80){
                    switch(command){
                              case      DISTANCE:
                                        distancia=make16(buff[0],buff[1]);
                                        break;
                              case      VELOCITY:
                                        speed=make16(buff[0],buff[1]);
                                        break;
                              case      NUMBER_OF_POINTS:
                                        n=make16(buff[0],buff[1]);
                                        break;
                    }
                    i2c_write(com);
          }
}
/*#########################################################*/
```

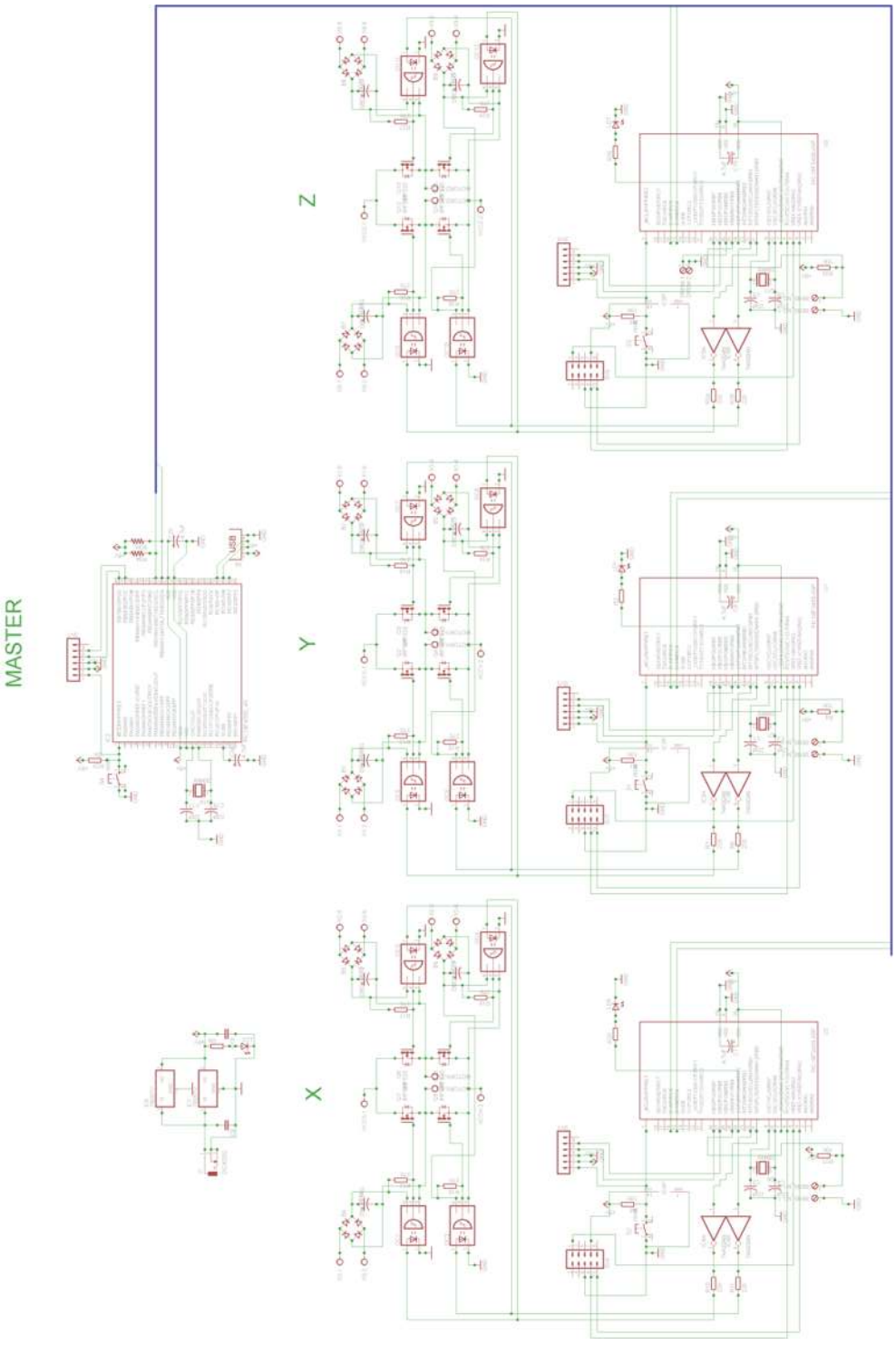
```
void main()
{
          int16 period=276;                                  //Period de 415 para 12 kHz
    SET_TRIS_A(0b00011101);            //Port A I/O Config
          SET_TRIS_B(0b00010000);                            //configuro RB4 como entrada para el limite de paro

          setup_power_pwm_pins(PWM_COMPLEMENTARY,PWM_OFF,PWM_OFF,PWM_OFF);
          setup_power_pwm(PWM_CLOCK_DIV_4|PWM_FREE_RUN|PWM_DEAD_CLOCK_DIV_4,1,0,period,0,1,7); //Frecuencia de 10 KHz

          //Frequency = Fosc / (4 * (period+1) * postscale)
          //para 12 KHZ la maxima cuenta de pwm es 1664

          setup_qei(QEI_MODE_X4,QEI_FILTER_ENABLE_QEA | QEI_FILTER_ENABLE_QEB | QEI_FILTER_DIV_1,1023);

          setup_timer_5(T5_INTERNAL | T5_DIV_BY_2);   //Reloj interno 5MHz  para 5 ms por muestra
          set_timer5(53036);
    enable_interrupts(INT_TIMER5);
          enable_interrupts(INT_IC2QEI);
//        enable_interrupts(INT_RB);
          enable_interrupts(INT_SSP);     //Activamos la interrupcion para el i2c
          enable_interrupts(GLOBAL);

          setPDC0(0);
          com='R';
//        Ref=-500;
          while(1){
                    switch(st){
                              case 0:
                                        while(command!=START_HOME);
                                        HOME_POSITION();              //voy a home position
                                        com='H';                                //envio comando de que estoy listo
                                        st=1;
                                        break;
                              case 1:
                                        while(command!=FIRST_POINT);
                                        MOVIMIENTO_X(100,500);
                                        com='P';
                                        st=2;
                                        break;
                              case 2:
                                        while(command!=START_ROUTING);
                                        output_high(dremmel);
                                        while(command!=LINEA_OK);
                                        output_low(dremmel);
                                        com='L';
                                        st=0;
                                        break;
                    }
          }
}

void setPDC0(signed int16 value){
          int16 DUTY_CYCLE;
          DUTY_CYCLE=5.56*value+556;
//        DUTY_CYCLE=78*value+8400;
//        DUTY_CYCLE=DUTY_CYCLE/10;
          set_power_pwm0_duty(DUTY_CYCLE);
}
#int_RB
void RB_isr(void) //función de interrupción para el limite de paro
{
          if(!input(PIN_B4)){
                    Posi=0;
                    RefP=0;
                    Ref=0;
                    k=0;
                    setPDC0(0);
          }
}

#int_IC2QEI
void  IC2QEI_isr(void)
{
          output_toggle(PIN_B2);
          k++;
}

#int_TIMER5
void TIMER5_isr()
{
          output_toggle(PIN_B3);
          posAc=qei_get_count();
          if(UP_DOWN==1){
                    if((posAc-posAnt)>1023){}
                    else
                              velAc=posAc-posAnt;
          }
          else{
                    if((posAnt-posAc)>1023){}
                    else      velAc=posAnt-posAc;
          }
          posAnt=posAc;

          CONTROL_IP_VELOCIDAD();
```

```
          set_timer5(53036);
}

void CONTROL_IP_VELOCIDAD(void)
{
/////// CONTROL DE VELOCIDAD /////////
          Ref1=Ref;
          if(Ref1<=0){Ref1=-Ref1;}
          error_inte=((signed int32)Ref1-(signed int32)velAc);
          error_prop=(signed int32)(velAc);

          INTE=INTE+error_inte*kid;
          PROP=error_prop*kpd;
          u=u+INTE-PROP;
          u=u/1000;
          if(u>=100)          u=100;
          else if(u<0)        u=0;

          u=(signed int)u;
          if(Ref<0) u=-u;
//        if(Ref==0) u=0;
          velAnt=velAc;

          setPDC0(u);
}

void MOVIMIENTO_X(unsigned int16 distancia, signed int16 speed)
{
                    deci_mm=distancia;
                    factor=deci_mm/2;   //50 mm a moverse

                    if(speed < 0)       RefP=-5954;                   //25600;  //5120 para resolucion de 0.2 mm en x y Y,
5954 para eje Z
                    else                          RefP=5954;

                    conv1=0;
                    conv=0;

                    RefP1=RefP;
                    if(RefP<0){         RefP1=-RefP;}
                    for(i=0;i<factor-1;i++){
                              Posi=0;
                              k=0;
                              while(Posi<=RefP1){

                                        conv=(unsigned int16)(POSCNTH*256)+(unsigned int16)POSCNTL;
                                        if(UP_DOWN==1)      Posi=conv+1023*k;
                                        else                          Posi=(1023-conv)+1023*k;
                                        Ref=speed;
                                        //if(RefP < 0)      Ref=-speed;
                                        //else                        Ref=speed;
                              }
                    }
                    Posi=0;
                    k=0;

                    while((Posi<(RefP1-1000))&& (RefP1>1000)){
                              conv1=conv;
                              conv=(unsigned int16)(POSCNTH*256)+(unsigned int16)POSCNTL;
                              if(UP_DOWN==1)      Posi=conv+1023*k;
                              else                          Posi=(1023-conv)+1023*k;
                              Ref=speed;
                              //if(RefP < 0)      Ref=-speed;
                              //else                        Ref=speed;
                    }
                    while(Posi>((RefP1-1000)||(RefP1<1000))&&(Posi<RefP1)){
                              conv=(unsigned int16)(POSCNTH*256)+(unsigned int16)POSCNTL;
                              if(UP_DOWN==1)      Posi=conv+1023*k;
                              else                          Posi=(1023-conv)+1023*k;
                              if(RefP < 0)        Ref=-Tabla[RefP1-Posi]*-Ref/1000;
                              else                          Ref=(Tabla[RefP1-Posi]*Ref/1000);
                    }
                    Posi=0;
                    RefP=0;
                    velAc=0;
                    velAnt=0;
                    Ref=0;
                    k=0;
//                  setPDC0(0);
}

void HOME_POSITION(void)
{
          while(input(PIN_B4)){
                    Ref=-500;
          }
          Posi=0;
          RefP=0;
          Ref=0;
          k=0;
          setPDC0(0);
}
```

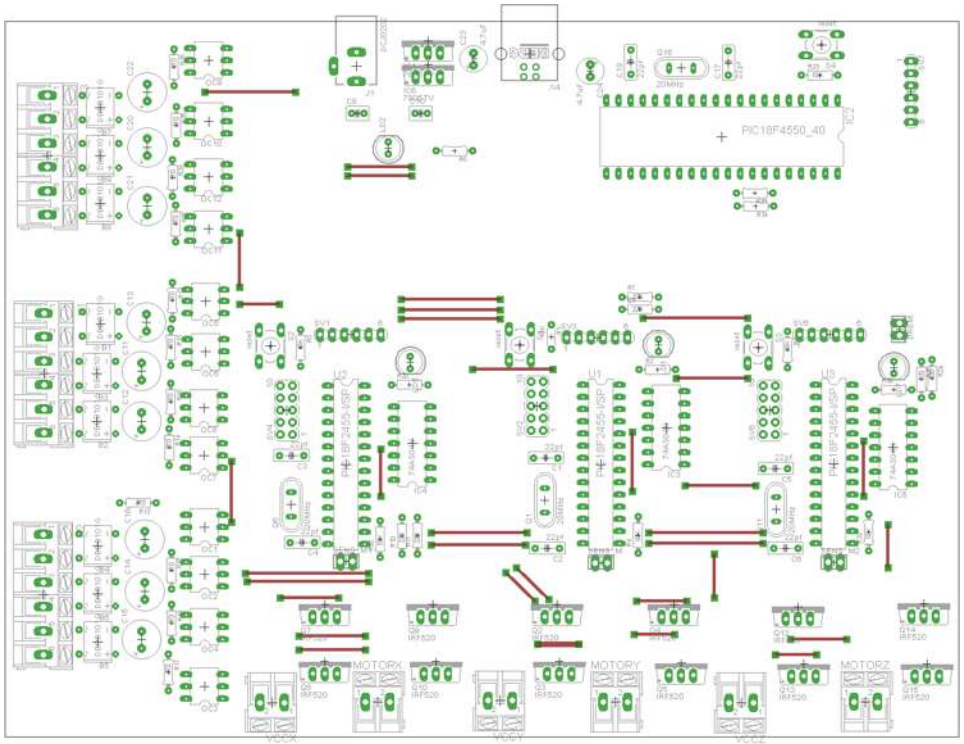
Apéndice B.

Esquemáticos de circuitos impresos

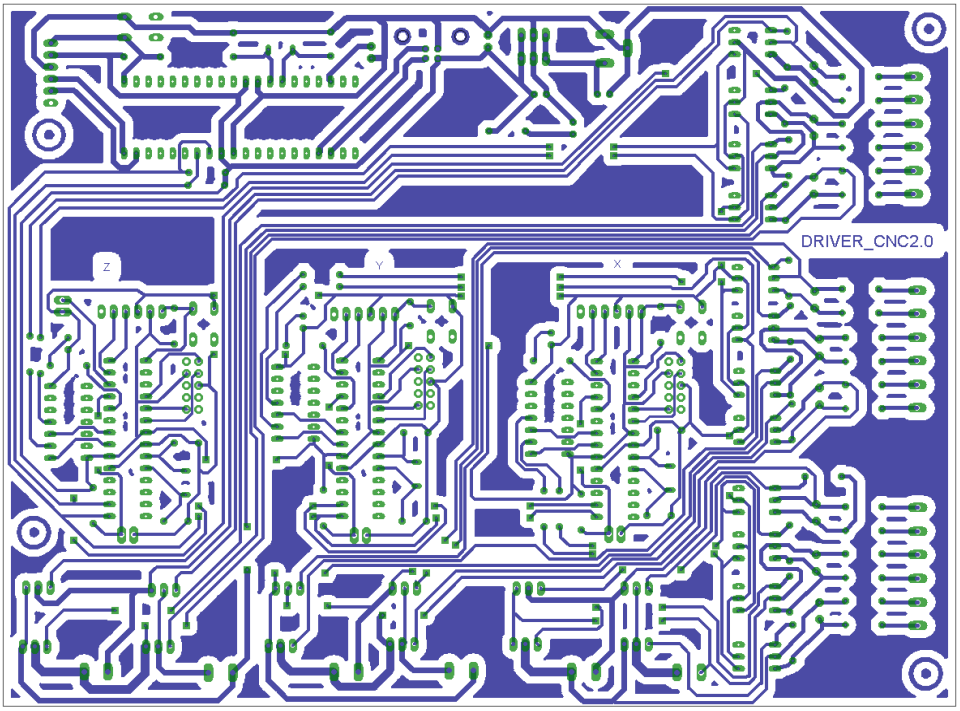
B.1. Esquemático del driver del manipulador



B.1.2. PCB del driver del manipulador

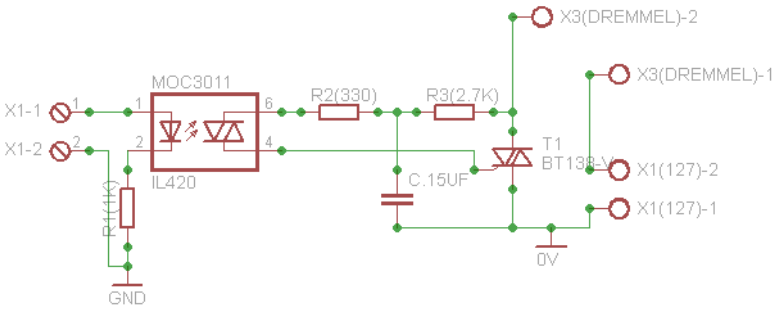


Capa superior.



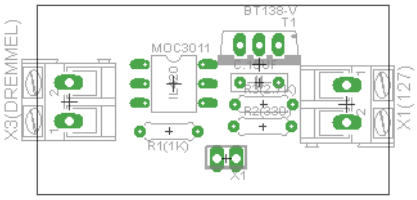
Capa inferior.

B.2.1. Esquemático del disparador de la herramienta



Esquemático.

B.2.2. PCB del disparador de la herramienta



Capa superior.

Capa inferior.

Referencias

[Barrientos, Aracil, 1997]

A. Barrientos, R. Aracil, *Fundamentos de Robótica*, Madrid España: McGrawHill, 1997.

[Cloy, 1993]

D. Mc. Cloy, *Robótica, una Introducción*, México: Limusa, 1993.

[Sandoval, 1997]

R.Sandoval, *Introducción a la programación de Robots*, IT CHihuahua, División de Estudios de Posgrado e Investigación. 1997.

[Mohan/Undeland/Robbins, 2002]

N. M. Mohan & T. M. Undeland & W. P. Robbins, *Power electronics:converters, application and design*, Michigan, E. U, Whiley & Sons, 2002

[Kenjo, 1994]

T. Kenjo, *Power Electronics for the microprocessors Age*, 1994.

[Ogata, 1996]

K. Ogata, *Sistemas de control en tiempo discreto*: Prentice Hall 1996.

[Hewlett Packard, 1984]

28 mm Diameter, Two and three channel incremental optical encoder Heds-5000 series, Technical data, 1984.

[NTE, 2002]

NTE3090 Optoisolator Shmitt Trigger Output, NTE Electronics Inc, Datasheet. 2002

[CCS Reference Manual 2011]

Custom Computers Services, Inc. *C Compiler Reference Manual*, February, 2011

[MATLAB Getting Started, 2011]

The MathWorks, Inc, Getting Started Guide, 2011

[I2C Phillips, 2000]

The I2C Bus Specification, version 2.1, 2000

[TENEDU 2011]

TENEDU, Flexicam, Estados Unidos, 02 de Marzo de 2011, http://www.tendu.com/flexicam.htm

[INFOTEC 2011]

INFOTEC GROUP, INFOTEC, Polonia, 02 de Marzo de 2011, http://www.infotec-group.com/go.live.php/ES-H1503/infotec-2512-p.html

[Microchip 2011]

Microchip Technology Inc. Pagina principal, Estados Unidos, 02 de Marzo del 2011, http://www.microchip.com

[Fairchild 2011]

Fairchild Semiconductor, Pagina principal, Estados Unidos, 10 de Mayo del 2011, http://www.fairchildsemi.com/

[Barrera 2009]

N. Barrera, "Construcción y control de un robot móvil de experimentación", Tesis de Licenciatura Febrero del 2009, Universidad Michoacana de San Nicolás de Hidalgo

[Ramírez 1998]

S. Ramírez, "Diseño y construcción de un manipulador con dos grados de libertad y orientación del efector final", Tesis de Maestría Octubre de 1998, Universidad Michoacana de San Nicolás de Hidalgo

[Sadín 2003]

P. E. Sadín, *Robot Mechanism and Mechanical Devices*, Boston, e E.U: McGrawHil, 2003