UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE UN SISTEMA DE CONTROL PARA SERVO MOTOR CON INTERFASE USB

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Ingeniero Eléctrico Electrónico

PRESENTAN

JULIO ALEJANDRO PINZÓN AGUIRRE y
ARMANDO FIGUEROA ORTIZ

DIRECTOR DE TESIS: M.I. BENJAMÍN VALERA OROZCO

MEXICO, D.F. 2007

Índice temático

Introducción

Objetivo

Desarrollar hardware y software para el control de posición y velocidad de un servo motor sobre la base de un microcontrolador con interfase a una PC mediante el puerto USB.

Definición del problema

El bus serie universal (USB, *Universal Serial Bus*) es una interfase flexible y de gran velocidad para conectar dispositivos periféricos a las computadoras. Las computadoras modernas incorporan al menos dos puertos USB en reemplazo a los antiguos medios de comunicación como puerto serie y paralelo. El puerto USB no sólo ha sido utilizado por dispositivos estándar comerciales como cámaras multimedia o impresoras; también ha sido introducido como parte integral de algunos microcontroladores dirigidos a los desarrolladores de aplicaciones de control. Al utilizar un microcontrolador USB, las aplicaciones de control se ven incrementadas en desempeño al contar con un medio de comunicación mediante el cual una PC puede actuar sobre el mundo real.

En el Laboratorio de Metrología del Centro de Ciencias Aplicadas y Desarrollo Tecnológico, CCADET UNAM, hemos experimentado la evolución de los estándares de comunicaciones RS232 hacia la interfase USB. Es por ésta razón que actualmente dirigimos los esfuerzos para adaptar nuestros desarrollos tecnológicos a las nuevas características de cómputo. Tal es el caso del control de movimiento en servomotores eléctricos que actualmente utilizan diversos proyectos de desarrollo en

instrumentación y que ineludiblemente deben incorporar el estándar USB en reemplazo del RS232 en extinción.

Entorno actual

En el año 2004 se experimentó con hardware y software para el control de movimiento en motores CD dirigidos a posicionar una Máquina de Medir por Coordenadas, MMC. Las MMC son uno de los instrumentos de medición geométrica de mayor precisión y exactitud que existen en el área de la metrología dimensional. Una MMC determina posiciones espaciales al posicionar su dispositivo sensor, llamado *palpador de contacto*, en los 3 ejes coordenados. La posición espacial es registrada por transductores de longitud, comúnmente codificadores ópticos, en el momento en que se registra un contacto mecánico entre el palpador y la pieza de trabajo. El movimiento en los ejes coordenados de una MMC es proporcionado por servomotores bajo el control centralizado de una PC, como se muestra en la figura 1.

Figura 1. MMC comercial.

Los experimentos realizados en esa fecha utilizaron un sistema de control electrónico sobre la base de un microcontrolador, una PC, interfase de comunicación serie RS232 y un prototipo de MMC de un solo eje coordenado, figura 2. El trabajo desarrollado fue reportado en la tesis “Control de movimiento para una máquina de medición por coordenadas” por el alumno Josué Zavaleta Irigoyen de la Facultad de Ingeniería UNAM [19]. En éste desarrollo, se prescindió de la MMC pero los resultados proporcionaron la pauta para extender el sistema electrónico y software utilizados a una MMC real.

Figura 2. Prototipo experimental para el control de movimiento en motores CD con aplicación en MMC.

Descripción del problema a resolver

Los antecedentes descritos y la sustitución del estándar RS232 por el USB, nos han conducido hacia el desarrollo de un sistema de control para servomotor con aplicación a la transmisión de movimiento en una MMC. Entonces el problema consiste en desarrollar hardware y software para controlar, en lazo cerrado, la velocidad y posición de un motor CD de forma completamente automática, centralizada en una PC y explotando las ventajas que ofrece la interfase USB. En nuestro concepto un sistema electrónico, sobre la base de microcontrolador, actúa sobre el servomotor en respuesta a los comandos enviados por una PC mediante el puerto común USB. Se contempla la manipulación de dispositivos tales como codificadores ópticos y motores CD de mediana potencia. En forma adicional, el sistema a desarrollar deberá contemplar la interfase con un palpador mecánico de contacto y un joystick. El palpador mecánico se simulará mediante un interruptor de dos posiciones. Mediante el joystick el usuario podrá establecer la velocidad deseada del servomotor, como se acostumbra en las MMC

comerciales. El esquema general del sistema planteado se muestra en la figura 3.

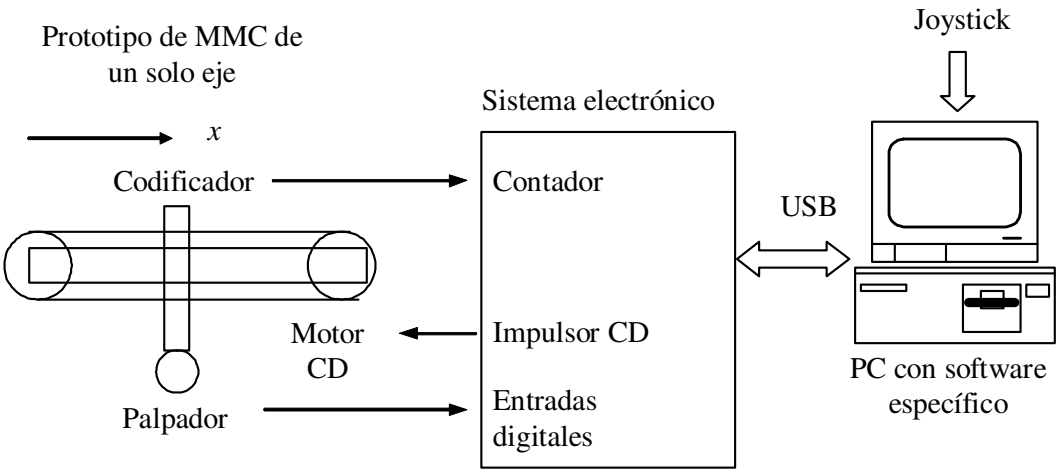

Figura 3. Esquema general del proyecto.

En un futuro, el presente proyecto servirá como pauta para el desarrollo de una MMC real.

Relevancia y justificación

La relevancia del presente proyecto de tesis radica en el apoyo a la creación de infraestructura de desarrollo propio que puede ser utilizada en la prestación de servicios ofrecidos por el Laboratorio de Metrología o que puede ser ofertada como una transferencia tecnológica. Adicionalmente, el proyecto apoya el desarrollo de un proyecto ambicioso en donde se pretende construir un instrumento de medición MMC de alta exactitud y gran capacidad de automatización. En el proyecto participan alumnos y académicos realizando labores de desarrollo tecnológico.

Por otra parte, y analizando el mercado de los controladores comerciales para servomotores, observamos que abundan los sistemas para PC sobre la base de comunicación por puerto serie, puerto paralelo y cavidades para PC (ISA, PCI). En el presente proyecto buscamos introducir la interfase USB como aspecto de relevancia, ya que en el mercado comercial no están ampliamente difundidos.

En un futuro, se busca innovar en el desarrollo de una MMC con interfase USB, situación que hasta el momento no es contemplada comercialmente y como alternativa a las interfases tradicionales IEEE-488.

Alcance y limitaciones

Las complicaciones implícitas al desarrollo de un proyecto orientado a automatizar un instrumento de gran exactitud como lo es la MMC, limitan nuestro trabajo a una propuesta de sistema para el control de un servomotor. La automatización completa de una MMC implica no sólo el desarrollo electrónico y de computación propuesto, sino también el diseño de mecanismos de precisión que posicionen la MMC en los niveles de exactitud deseados. De esta manera, nos limitamos a desarrollar un prototipo electrónico y de programación que sea fácilmente transportado a una MMC como parte de una etapa posterior de integración de partes.

Por otra parte la infraestructura a utilizar ya ha sido adquirida y por lo tanto no existe posibilidad de evaluación de diferentes opciones debido a que la experiencia de los académicos del Laboratorio de Metrología ha orientado el proyecto a su situación actual.

Método

El método a emplear en el presente proyecto de tesis consiste en integrar un sistema de control de movimiento realimentado orientado a una MMC empleando técnicas de control digital sobre la base de una PC y el microcontrolador PIC16C765 con interfase USB. La figura 4 muestra el esquema propuesto.

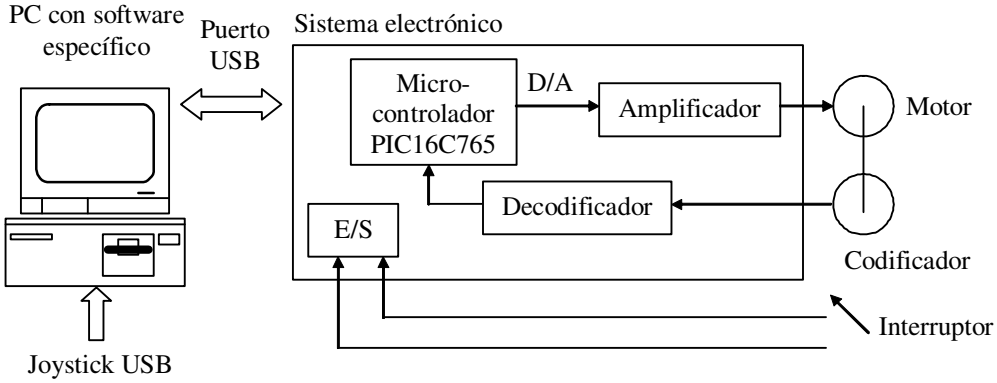

Figura 4. Esquema del sistema de control para un servomotor con interfase USB.

En la figura 4 el sistema electrónico recibe comandos específicos para ejecutar acciones de control de posición, control de velocidad y lectura de estado enviadas por el software específico en la PC mediante la interfase USB. El micro-controlador interpreta los comandos enviandos y regresa a la PC el resultado de la operación. Asimismo, el

microcontrolador implementa un algoritmo PID que genera señales PWM mediante el convertidor D/A. La señal PWM posteriormente es adecuada en potencia por el amplificador y aplicada al motor CD. Las señales en cuadratura que genera el codificador óptico lineal, solidario al motor, son procesadas por el decodificador de manera que sea interpretada la lectura de posición y se cierra el lazo de control. El palpador, simulado por un interruptor de dos posiciones, genera dos señales eléctricas on/off cuando es desplazado en la dirección de palpación por el servomotor y hace contacto con una pieza de trabajo. Tales señales son recibidas por el controlador en sus entradas para interruptores con el fin de validar una lectura de longitud en el momento preciso en que ocurre el contacto mecánico. Los comandos de control son ingresados por el operador en una interfase de medición amigable. En particular se requiere de un control de palanca o joystick que tradicionalmente es usado en las MMC's.

Debido a la carencia actual de un diseño mecánico de precisión que nos permita transmitir el movimiento de los motores CD a los ejes coordenados en una MMC, la implementación del esquema de la figura 4 se limita a desarrollar un prototipo que simule las condiciones reales. La validación del prototipo nos conducirá a la implementación en una MMC real.

Resultados esperados

Desarrollar un sistema hardware y software para el control de un servomotor gobernado por una PC e interfase USB que sea fácilmente integrado a una máquina de medir por coordenadas. Se espera que tenga una exactitud en el posicionamiento de ±2 cuentas del codificador óptico, interfase con un palpador de contacto con dos sentidos de palpación e interfase con joystick.

Resumen de la tesis

El primer capitulo contiene una breve introducción al estándar de comunicaciones USB enfocado al desarrollo de dispositivos periféricos para PC, se explican las razones del por que usar esta interfase y se presentan los beneficios de su uso tanto para los desarrolladores como para los usuarios. Se considera que la interfase no es perfecta y se describen las desventajas más características. Una vez que se ha optado por usar USB, está el dilema de saber cual microcontrolador usar que tenga el soporte para la interfase. De esta manera, se proporciona una

breve introducción a la arquitectura general de los chips USB y la descripción de los más usados que son fabricados por los distintos distribuidores de importancia. En seguida, en el presente proyecto es indispensable tratar con los principios básicos para la obtención del modelo electromecánico de un motor CD. El primer paso es obtener la función de transferencia de la planta. Una vez obtenida, el siguiente paso es conseguir el sistema en lazo cerrado para después hacer la discretizacion de la planta y asi diseñar el control digital de movimiento del motor CD. Una herramienta indispensable en el presente proyecto es la teoría de los codificadores. En el tercer apartado se proporciona un resumen del principio de funcionamiento de los codificadores así como los tipos existentes. Por ultimo, el cuarto apartado contiene la descripción de las componentes más importantes del microcontrolador PIC16C765 de MicroChip, usadas en el proyecto, ya que fue el microcontrolador base para el presente trabajo de tesis.

En el segundo capitulo se presentan los siguientes aspectos del trabajo realizado específicamente para el presente proyecto de tesis. En primer lugar el esquema general del sistema propuesto para el control de un servomotor. Después, una descripción del sistema electrónico desarrollado para el control y finalmente la descripción de la interfase de operación para PC desarrollada en Visual C++ 6.0. El capítulo contiene la implementación del algoritmo de control que reside en el microcontrolador del sistema electrónico. El software de operación para PC es descrito tanto en el ámbito del desarrollador, describiendo las clases usadas; como en el ámbito de usuario, describiendo la pantalla con sus botones y controles de la cual se obtienen e introducen valores. El software desarrollado es concebido como una “terminal USB” de donde se envían y reciben comandos USB para manejar u observar el estado del sistema electrónico.

Finalmente, el capítulo tres contiene los resultados obtenidos al operar el sistema de control desarrollado, las conclusiones derivadas del presente proyecto y a partir de lo anterior, las posibles líneas de trabajo a futuro.

Capítulo 1

Antecedentes

En el presente capítulo se revisan los conceptos fundamentales en el desarrollo del proyecto de tesis. En primer lugar, una breve introducción al estándar de comunicaciones USB enfocado al desarrollo de dispositivos periféricos. Posteriormente se mencionan los elementos fundamentales para comprender el control digital de movimiento en motores CD. Se realiza una breve introducción a los codificadores ópticos utilizados como transductores de posición en el control de movimiento de motores CD. Finalmente, se describe en forma general el microcontrolador utilizado en el presente trabajo de tesis.

1.1. Desarrollo de periféricos USB

USB es una probable solución para comunicar una computadora con dispositivos afuera de ella. USB es una especificación de las empresas Compaq, Intel, Microsoft y NEC, que describe un canal serie que soporta una gran variedad de periféricos de media y baja velocidad, con soporte integral para transferencias en tiempo real (isócronas) como voz, audio y vídeo comprimido, y que permite mezclar dispositivos y aplicaciones isócronas y asíncronas. Por lo tanto, entre los dispositivos USB más característicos se pueden citar teclados, ratones, joysticks, tabletas gráficas, monitores, modems, impresoras, escáneres, CD-ROMs, dispositivos de audio (como micrófonos o altavoces digitales), cámaras digitales y otros dispositivos multimedia [15].

Varios sistemas operativos soportan USB como es: Win98, Win2000 y MacOS 8.5 y superior asi como Linux, NetBSD y FreeBSD.

1.1.1. Descripción de la interfase USB

Nivel fisico

A nivel físico, USB utiliza un cable de 4 conductores para transmitir una señal diferencial (D+ y D-) y alimentación (VBus = 5V y GND) por medio de conexiones punto a punto. Los dispositivos de baja velocidad van obligatoriamente equipados con un cable de longitud adecuada (hasta unos 3 m, dependiendo de sus características eléctricas), mientras que los de alta velocidad pueden ir equipados con un cable o utilizar cables independientes de hasta 5 m (también dependiendo de sus características eléctricas).

La comunicación es bi-direccional y semi-dúplex. Los dispositivos disponen de un transmisor diferencial, receptores diferencial de Salida/Entrada y resistencias de terminación con los que pueden transmitir y detectar varios estados eléctricos distintos en la línea.

Hubs

Dentro de la arquitectura USB, unos elementos esenciales y especiales son los *hubs* (concentradores), que proveen conectividad ya que los dispositivos no se conectan entre sí directamente, sino cada uno a un hub. Detectan la conexión y desconexión de dispositivos y si son alta o baja velocidad, generan alimentación hacia los dispositivos e incorporan la terminación de las líneas. Como a los hubs se conectan los dispositivos en estrella, la topología USB se denomina *Estrella en Niveles*. USB permite hasta 6 niveles, y en el nivel 0 (raíz o root) se encuentra el *Controlador USB*, que controla todo el tráfico de información en el bus.

Protocolo

El protocolo de nivel físico se basa en *tokens* (testigos). El controlador USB transmite tokens que incluyen la dirección del dispositivo destino, y el dispositivo que detecta su dirección en el token responde y lleva a cabo la transferencia de datos con el controlador. De esta manera, el controlador USB maneja la parte más compleja del protocolo, generando los tokens de transferencias de datos a 12 Mbps o a 1,5 Mbps, y controlando la conexión lógica entre el sistema y las funciones internas de cada dispositivo. El controlador USB también maneja el consumo en el bus a través de las funciones suspender/continuar, por medio de las cuales controla los modos

reposo/activo de los dispositivos. Esta arquitectura permite el diseño de dispositivos extremadamente simples y de bajo costo.

USB divide el tiempo en espacios de 1 ms denominados tramas, durantes las cuales se llevan a cabo las comunicaciones a través de transacciones, las cuales se componen a su vez de paquetes.

1.1.2. Tipos de transferencias

USB soporta 4 tipos de transferencias de datos:

1. Control. Para configuración y control de dispositivos y para manejo del bus. USB desarrolla tres tipos de transacciones para controlar el flujo de información:

- **Transacción de configuración (Setup)**. En la que se envía al dispositivo un paquete que especifica la operación a ejecutar.
- **Cero o más transacciones de datos**. En las que se transfieren los paquetes de datos en el sentido indicado por la transacción de configuración.
- **Transacción de estado**. En la que el receptor informa del estado final de la operación.

2. Isócrono, para transmisión de información con ancho de banda y latencia garantizadas, necesario para aplicaciones como audio, telefonía y vídeo. Permite una comunicación periódica y continua entre el sistema y el dispositivo.

3. Interrupción. Para transferencias de pocos datos, no periódicas, de baja frecuencia pero con unos ciertos límites de latencia. USB contempla las transferencias de interrupción que aseguran una transacción (paquete) dentro de un periodo máximo e incorporan la detección de errores y retransmisión de datos. La información útil por paquete puede oscilar entre 1 y 64 bytes para dispositivos de alta velocidad y entre 1 y 8 bytes para dispositivos de baja velocidad. El sistema puede asignar como máximo el 90% del tiempo de trama para transferencias isócronas y de interrupción. Si el sistema no puede garantizar tiempo suficiente como para manejar una nueva conexión de interrupción (transmitir un nuevo paquete dentro del periodo máximo requerido), simplemente no se establece la conexión.

4. Bulto. Para transferencias de grandes cantidades de datos con dispositivos asíncronos, como impresoras, escáneres, cámaras de fotos (foto fija), etc. Las transferencias de bulto sólo son utilizables por dispositivos full-speed. Se procesan por medio de un mecanismo "good effort", en el que el sistema aprovecha cualquier ancho de banda disponible y en el momento en que esté disponible. Se puede

utilizar el tiempo de trama reservado y no consumido por transferencias de control (10%). Incorporan mecanismos de control de errores para garantizar la entrega de datos.

Estos cuatro tipos de transferencias están disponibles como interfaces software que el sistema pone a disposición de los manejadores de dispositivo, estando los manejadores obligados a comunicarse con los dispositivos, única y exclusivamente a través de estas 4 interfaces de programación

1.1.3. Beneficios

Beneficios para usuarios

Los beneficios, según los usuarios, son que es fácil de usar, rápido y fiable en la transferencia de datos, flexible, de bajo costo y bajo consumo de energía. Sin emabargo existen otros beneficios que se mencionan a continuación [3].

Una interfaz para varios dispositivos. USB es suficientemente versátil para ser usada por varios tipos de periféricos. En lugar de tener diferentes tipos conexiones y mantenimiento de hardware para cada periférico, una interfase sirve a varios.

Configuración automática. Cuando se conecta a un periférico USB, Windows automáticamente detecta el periférico y carga el manejador apropiado. La primera vez que el dispositivo se conecta, Windows le pedirá el usuario insertar el disco con el manejador, pero para las próximas ocasiones la instalación es automática. Entonces eliminamos la necesidad de localizar y correr un programa de instalación o reiniciar el sistema antes de usar el periférico.

No usar opciones. Los periféricos USB no tienen opciones a seleccionar tales como dirección de puertos y solicitar interrupciones. Las solicitudes de interrupción (IRQ) están disponibles en pequeños suministros en la PC, y no se tienen que estar localizando uno para un nuevo periférico. Esta es una razón suficiente para elegir USB.

Recursos de hardware libres para otros dispositivos. Usando USB para varios periféricos es posible librarse de líneas de interrupción (IRQ) de los periféricos que lo requieran. La computadora dedica una serie de puertos y una solicitud de interrupción (IRQ) a la interfaz USB, pero más allá de esto, los periféricos individuales no requieren recursos adicionales. En contraste, cada periférico que no use USB requiere dedicarle un puerto, frecuentemente una línea de solicitud de interrupción, y algunas veces una ranura de expansión (para una tarjeta de puerto paralelo, por ejemplo).

Fácil de conectar. Con USB no hay necesidad de abrir la computadora para añadir una tarjeta de expansión para cada periférico. Una computadora típica tiene por lo menos dos puertos USB. Se puede expandir el número de puertos conectando un hub a un puerto existente. Cada hub tiene puertos adicionales para conectar más periféricos o hubs.

Cableado sencillo. El cable de conexión USB puede medir a lo más cinco metros. Con un hub un dispositivo puede conectarse a más de treinta metros con la computadora. Comparados con los conectores típicos como RS-232 y puerto paralelo son pequeños y compactos

Versatilidad. La interfase USB es la más rápida y flexible para conectar dispositivos a la computadora. Las nuevas computadoras tienen cuando menos un par de puertos USB. La interfase además es versátil pues es usada en periféricos estándar tales como teclados, unidades de almacenamiento y dispositivos más especializados.

Un dispositivo USB puede usar cuatro tipos de transferencias y tres velocidades. Además, el dispositivo puede responder a una serie de requerimientos para que la PC aprenda acerca del dispositivo y a establecer comunicación con el. En la PC, cualquier dispositivo debe tener un driver de bajo nivel para manejar la comunicación entre las aplicaciones y los drivers USB del sistema.

Conectado en "caliente". Se pueden conectar o desconectar periféricos cuando se quiera este o no el sistema conectado sin dañar la PC o periféricos. El sistema operativo detecta cuando el dispositivo esta conectado y listo para usarse.

No se requiere de una fuente de poder externa. La interfase USB incluye una fuente de poder y línea de tierra de 5 volts desde la computadora. Un periférico que requiera arriba de 500 mA tendrá que usar toda la potencia que tiene el bus, y es solo entonces cuando se necesita una fuente externa. Así muchos dispositivos pueden escoger entre incluir la fuente de 5 volts o usar la inconveniente fuente externa.

Velocidad. USB soporta tres tipos de velocidad: *high-speed* 480 Mbits por segundo, *full-speed* de 12Mbites por segundo y *low-speed* de 1.5 Mbits por segundo. Todas las PC equipadas con USB soportan las velocidades baja y completa. La velocidad alta (high-speed) fue añadida en la versión 2.0 y requiere hardware equipado con USB 2.0 en la tarjeta madre o una tarjeta de expansión. Ahora, la velocidad baja fue incluida por dos razones. Por que algunas veces los periféricos pueden construirse a bajo costo, por ejemplo, para los

ratones y otros dispositivos se requieren cables flexibles y los cables a baja velocidad lo son porque ellos no requieren de mucho cuidado. La full-speed es comparable o mejor que la velocidad conseguida por el puerto serie y el paralelo y puede servir para remplazarlos.

Fiabilidad. La fiabilidad resulta del diseño de hardware y del protocolo de transferencia de datos. Las especificaciones de hardware para los manejadores USB, receptores y cables eliminan mucho del ruido que de otra manera causaría errores. Aun más, el protocolo USB habilita la detección de errores y notifican el envío para que se pueda retransmitir. La detección, notificación y retransmisión son hechas en hardware y no requieren ninguna programación o intervención del usuario.

Bajo costo. La interfaz USB es la mas compleja y joven, además de que sus componentes y cables son baratos. Un dispositivo con USB es igual o mas barato que con una vieja interfase. Para varios periféricos de bajo costo la baja velocidad es una opción que puede reducir los costos futuros.

Beneficio para desarrolladores

Los avances han hecho a los usuarios estar ansiosos de usar periféricos USB debido a que muchos de los desarrollos con esta interfase resultan fáciles, por ejemplo, los cables estándar y chequeo automático de errores definidos por USB hace que los desarrolladores no se preocupen por especificaciones en las características del cable o revisar errores en el software. Los beneficios para los desarrolladores resultan de la flexibilidad construida dentro del protocolo USB, el soporte en el chip controlador y el sistema operativo y el hecho de que la interfase no es controlada por un único distribuidor. Aunque los usuarios no están concientes de ello, los resultados son los bajos costos, pocos problemas y variados y ricos periféricos. Otros beneficios se explican a continuación.

Flexibilidad. Los cuatro tipos de transferencia y los tres tipos de velocidad hacen posibles muchos tipos de periféricos. Hay varios tipos de transferencia para intercambio de pequeños y grandes bloques de datos, con o sin supresión de tiempo. Para los datos que no pueden tolerar retrasos, USB garantiza un porcentaje de transferencia o tiempo máximo entre transferencia. Estas propiedades son especialmente bienvenidas en Windows, el cual accede a los periféricos en tiempo real. El sistema operativo, los manejadores y el software pueden introducir inevitablemente retrasos, pero USB hace posible llevar a cabo las transferencias muy cerca del tiempo real. Sin parecido con otras interfaces, USB no asigna funciones especificas o

hace otras suposiciones acerca de cómo la interfase debe usarse. Para la comunicación con dispositivos comunes tales como impresoras y modems, hay clases de USB con los requerimientos para dispositivos definidos y protocolos. Esto salva a los desarrolladores de reinventarlos de nuevo.

Soporte del Sistema Operativo. Windows 98 fue el primer sistema operativo de Windows que soportaba USB, y sus sucesores incluyendo Windows 2000, ME lo soportan también. Otras computadoras y sistemas operativos soportan USB como las MAC, Linux, NetBSD y FreeBSD. Ahora, en cada sistema operativo pueden variar diversas cosas. El nivel de soporte varía. En el nivel fundamental, un sistema operativo que soporte USB debe hacer tres cosas:

- Detectar cuando un dispositivo ha sido conectado o removido.
- Comunicarse nuevamente con el dispositivo conectado para encontrar como intercambiar datos con el.
- Proveer de un mecanismo que habilite el software manejador de la PC para comunicarse con el periférico.

En un nivel mas alto el sistema operativo debe incluir manejadores que hagan posible acceder a los dispositivos llamando funciones soportadas por el sistema operativo. Si el sistema operativo no incluye este manejador, el vendedor debe proveer uno.

Microsoft ha añadido clases de manejadores en cada Windows. Estos dispositivos para los cuales se añaden manejadores son llamados *Dispositivos de Interfaz Humana* (HID Human Interface Device) como teclados, ratones, palancas, dispositivos de audio, modems, cámaras, escáneres, impresoras, memorias e inclusive el chip usado en el presente proyecto. Las aplicaciones se basan en las funciones llamadas *Interfaz de Aplicación de Programa* (API).

Los manejadores USB usan el nuevo modelo *Win32 Driver Model* (WDM) el cual define la arquitectura de los manejadores que corren bajo Windows 98, 2000, Me y futuras ediciones. Esto apunta a que los desarrollos estén soportados en todos los sistemas operativos con un único manejador.

1.1.4. Desventajas

Desafíos. Desde la perspectiva de los usuarios, falta soporte para el viejo hardware y sistemas operativos, velocidad y limite de distancia que hace al USB impractico para algunos usos, así como problemas con algunos productos [3].

Falta de soporte del hardware heredado. Las viejas computadoras y periféricos no tienen puertos USB. Si se quiere conectar un periférico que no tiene USB a un puerto USB, una solución es convertidor que traslade entre USB y la vieja interfase. Varias fuentes tienen convertidores para usarse con periféricos con RS-232, RS-485 y el puerto paralelo. De cualquier modo, la solución por el convertidor es muy usada solo para periféricos que usan el protocolo convencional soportado por el manejador convertidor. Por ejemplo, el convertidor a puerto paralelo solo soporta impresoras pero no otros periféricos.

Si se quiere usar un periférico USB con una PC que no tenga USB, la solución es añadirle USB. Se requieren dos cosas: Un controlador USB y un sistema operativo que soporte USB. El hardware esta disponible en tarjetas de expansión que se conectan en una ranura de la PCI. Pocos periféricos tienen manejadores para usarse con ediciones como Windows 95. Si el hardware no encuentra los requerimientos mínimos del Windows 98, debe actualizarse. La actualización del hardware muchas veces cuesta más que actualizar el sistema operativo. Pero si no es la solución cambiar el sistema es muy complicado convertir de una interfaz antigua (puerto paralelo, RS-232, etc) a USB ya que el código para el control de los periféricos y el controlador normalmente reside en la PC. Así que convertir no es una opción normal cuando la PC tiene una interfase heredada.

Los usuarios ocasionalmente corren aplicaciones en viejos sistemas operativos tales como MS-DOS. Pero los manejadores que las aplicaciones que Windows 98 usan para comunicar con los dispositivos USB son específicos de Windows. Sin el manejador, no hay manera de acceder al periférico USB. Aunque si bien es posible escribir un manejador para DOS, la realidad es que pocos periféricos proveen de uno.

Limite de velocidad. USB es versátil pero no esta diseñado para hacer todo. La velocidad alta de USB lo hace competitivo con la interfaz IEEE-1394 que transmite a 400 Mbits por segundo, pero esta ultima puede ser mas rápida aun, esto es a 3.2 Gigabytes por segundo.

Limite de distancia. USB fue diseñado como un bus de escritorio, para periféricos que realmente estén a la mano, a lo más 5 metros. Otras interfaces como la RS-232, RS-485 y Ethernet permiten longitudes mas largas. Se puede incrementar la longitud de alcance a 30 metros por ejemplo pero se necesitan 5 hubs cada cinco metros, usando 6 segmentos de cable. Si se quiere extender más allá de esto es mejor un convertidor a RS-232 u otra interfaz.

Comunicación usuario – usuario. El que un sistema USB sea de escritorio quiere decir que tiene una computadora principal que maneja las comunicaciones. Ningún periférico puede comunicarse con otro directamente. Todas las comunicaciones son desde y a la computadora principal. Otras interfaces con la IEE-1394 permiten la comunicación directamente entre periféricos.

USB provee de una solución parcial con USB *On-The-Go* introducida en el 2001 en un suplemento de la especificación 2.0. USB *On-The-Go* define una computadora principal con reducidas capacidades, conveniente para usarse con dispositivos embebidos que necesitan conectarse a un solo periférico USB.

Complejidad del protocolo. Para programar periféricos USB se necesita saber bien acerca de los protocolos USB. El chip controlador maneja mucho de las comunicaciones automáticamente pero aun así debe ser programado. Estos chips varían en cuanto es lo que soportan. En el lado de la PC, para escribir un manejador es necesario familiarizarse con el protocolo USB y la responsabilidad que tiene manejador.

En contraste, algunas viejas interfaces pueden conectarse con circuitos muy simples y muy básicos protocolos. Por ejemplo, el puerto paralelo de la PC es solo una serie de entradas y salidas digitales. Cualquiera puede conectar circuitos de entrada y salida básicos tales como retardadores, interruptores, convertidores analógico digital y no se requieren chips del lado del periférico ni tampoco manejadores en el lado de la PC.

Evolución del soporte del sistema operativo. Windows incluye clases de manejadores que habilitan las comunicaciones con algunos dispositivos. Esto es bueno si el diseño para el dispositivo usara uno de estos manejadores que proveen. En muchos casos, se tiene que usar o adaptar un manejador que provee el vendedor del chip controlador. Varios vendedores ofrecen herramientas que hacen fácil el trabajo de escribir el manejador USB.

Honorarios. El Forum USB provee de especificaciones, documentos, software y mucho más. Todo es gratis, pero nadie puede desarrollar software USB sin pagar una licencia. De cualquier modo, alguien que venda un dispositivo USB debe obtener un *Vendor ID*, que cuesta alrededor de $1500 o también si uno se inscribe por un año puede ser gratis pero la suscripción cuesta alrededor de $2500.

USB contra IEEE-1394. La otra interfase elegida por nuevos periféricos es IEEE-1394. USB y IEEE-1394 son aproximadas, la

IEEE-1394 es mas rápida y flexible, es la mejor para video y otras cosas donde la velocidad es esencial o donde una PC no este disponible. El problema es que es muy costosa. USB es la mejor para periféricos típicos tales como teclados, impresoras, escáneres, discos duros y, en general, para aplicaciones de baja o moderada velocidad. Mientras que en USB un *host* (PC) controla las comunicaciones con varios periféricos y, por tanto, es más complejo, así también la electrónica de los periféricos puede ser relativamente simple y barata. En IEEE-1394 usa el modelo *peer-to-peer*, donde los periféricos se pueden comunicar con otros directamente, el resultado es una interfase más flexible pero la electrónica de los periféricos es más compleja y cara. La versión IEEE-1394 trabaja a 400 Mbits por segundo y es treinta veces más rápida que las versiones 1.x de USB que trabajan a 12 Mbits por segundo. La versión USB 2.0 trabaja a 480 Mbits por segundo pero la IEEE-1394b llega a trabajar a 3.2 Gbits por segundo que es seis veces mas.

1.1.5. Seleccionando el dispositivo electrónico

USB también ha tenido avances para los desarrolladores. Los desarrolladores ahora incluyen los diseños de hardware, los componentes y diseño de los circuitos [3].

Un controlador de periféricos USB tiene, por supuesto, un puerto USB y circuitos que soportan la comunicación con el *host* (PC). Un transmisor USB provee la interfase de hardware al bus. Los circuitos que se comunican de esta manera con el transmisor reciben el nombre genérico de *serial interface engine* (SIE). La SIE maneja la recepción y envío de datos, o sea, envía los datos que están disponibles así como guarda cualquier dato recibido. Una típica SIE hace lo siguiente:

- Detecta paquetes que llegan.
- Envía paquetes.
- Detecta y genera *Start-of-Packet*, *End-of-Packet*, *Reset* y devuelve señales.
- Codifica y decodifica datos en el formato requerido en el bus.
- Verifica y genera valores CRC.
- Decodifica y genera Paquetes IDs.
- Convierte entre datos serie de USB a datos paralelos en registros o memoria.

Buffers de datos USB. Un controlador USB debe tener buffers para guardar datos que fueron recientemente recibidos y para datos que están listos para enviarse al bus. Algunos chips como el NET2888 de Netchip usan registros mientras otros como el EZ-USB de Cyprees reservan una porción de datos de memoria para los buffers.

La recepción y transmisión de datos es estructurada como FIFO (*first in, first out buffers*). Esto es, cada que un dato es recibido es acomodado en último lugar. Un apuntador interno a la siguiente localidad se incrementa automáticamente cuando el *firmware* lee o escribe al FIFO. En algunos chips como el Cyprees el buffer esta en la memoria de datos y el firmware debe seleccionar explícitamente cada localización para leer o escribir. No hay un apuntador que incremente automáticamente cuando el firmware lee o escribe en el buffer. Los bytes que se transmiten van en orden desde la menor dirección hasta la mayor y los bytes recibidos se guardan como llegan, desde la dirección baja hasta la alta.

CPU. La unidad central de procesamiento (CPU) controla las acciones del chip ejecutando las instrucciones del firmware almacenadas en el chip. Cada CPU soporta un catalogo de instrucciones que incluye lenguaje maquina para movimiento de datos, operaciones lógicas y matemáticas así como saltos en el programa. Una instrucción habilita al CPU para comunicarse con la SIE. El CPU puede estar basado en un microcontrolador de propósito general tal como el 8051 o también desarrollado para el diseño de aplicaciones especificas para USB.

Memoria de programa. La memoria de programa tiene el código que el CPU ejecuta. Puede almacenarse en memorias de varios tipos: ROM, EPROM, EEPROM, Flash EPROM o RAM. El rango de almacenamiento puede llegar hasta los kbytes aunque hay casos en que llegan a almacenar Mbytes. El nombre del código almacenado en una memoria de programa es firmware, este termino indica que la memoria no es volátil y que no es fácil cambiar el código del programa cargado en la RAM, ni editado ni vuelto a salvar en disco.

Memoria de datos. La memoria de datos provee de almacenamiento temporal durante la ejecución del programa. El contenido de la memoria de datos puede incluir datos recibidos desde el puerto USB, datos enviados al puerto USB, valores usados en cálculos o cualquier otra cosa que el chip necesite recordar. Esta memoria es usualmente RAM. La capacidad de almacenamiento esta entre los 128 y 1024 bytes.

Registros. Son también de almacenamiento temporal. Están localizados en el CPU y se accede a ellos mediante instrucciones definidas. En algunos CPU el acceso es más rápido que en otros. Un controlador USB típicamente tiene un registro de estado y uno de control los cuales tienen información acerca de las terminales habilitadas, número de bytes recibidos, el número de bytes listos para transmitir, información de errores y otra información acerca de cómo se ha usado el chip y el estado actual de datos recibidos y transmitidos.

Otras I/O. Casi todos los controladores tienen posibilidad de tener varias interfases aparte de la USB, o sea, incluyen entradas y salidas de propósito general que se pueden conectar a otros circuitos. Un chip puede soportar otras interfases tales como la asíncrona RS-232 o la sincronía como la IC, Mricowire y SPI. Otros chips son de propósito especial, por ejemplo, USA1321 de Philips que contiene un convertidor analógico digital para usarse en bocinas USB y otros dispositivos de audio.

Simplificando el proceso de desarrollo. Un simple y rápido proyecto USB es uno que use un controlador con las siguientes características:

- La arquitectura y lenguaje de programación debe ser familiar.
- Detallada y bien organizada documentación del hardware.
- Buena documentación, firmware de ejemplo libre de fallas.
- Un sistema de desarrollo que sea fácil de cargar y depurar errores en el firmware.
- Manejador disponible, ya sea que este en Windows o se tenga una buena documentación para el vendedor u otra fuente conveniente.

Chips diseñados para USB. Algunos chips son diseñados específicamente para aplicaciones USB. En lugar de añadir capacidades USB a la arquitectura existente, estos diseños son optimizados para USB desde el principio. Dos ejemplos de este tipo son los chips de Cyprees y los de ScanLogic.

La familia M8 de Cyprees es barata y tiene un óptimo conjunto de instrucciones. La serie enCore es de baja velocidad cada uno con un puerto USB y de 8 a 16 líneas de propósito general de entrada/salida. Otras de la serie M8 tienen más entradas/salidas y soportan la full-speed.

El SL11R de ScanLogic contiene BiosRom que soporta las cuatro transferencias USB. La ROM ejecuta el firmware desde una memoria externa paralela o cargando el código desde la EEPROM a la RAM. El chip tiene 32 líneas de entrada/salida de propósito general.

Chips basados en las familias populares

Estar familiarizados con alguna familia y que esta añada las capacidades USB da una mejor idea de cómo empezar un proyecto pues ya es conocida la arquitectura y las instrucciones. Como ejemplo esta el 8051 de la serie FX2 de Cyprees, también el AVR de Atmel, el PIC de Microchip y el 68HC05/8 de Motorota. La tabla 1.1 muestra algunos ejemplos.

Compañía	Chip de ejemplo
AMD	AM186
Atmel	AT76C711
Cyprees	AN2121 (EZ-USB series)
Infineon	C541U
Microchip Technology	16C7x5
Mitsubishi	7640, 7532/36
Motorota	68HC05JB3/4, 68HC08JB8, MPC850
Standard Microsystems	USB97C100
ST Microelectronics	ST7261

Tabla 1.1. Chips USB.

Breve descripción de algunos chips

Cyprees enCore. Estos chips son de simple diseño y baratos. Fueron proyectados para transferencias de pequeños bloques a baja velocidad. Ejemplos de su uso incluyen periféricos estándar como ratones y palancas también en dispositivos especializados en unidades de adquisición de datos y controladores.

Cypress EZ-USB. La familia EZ-USB es notable por dos razones, es compatible con el 8051 y los chips soportan una flexible y diferente forma de almacenar el firmware. En lugar de almacenar el firmware en el chip, una EZ-USB puede almacenar el firmware en el host, el cual se carga en el chip cuando de enciende. La ventaja de esto es que es fácil de actualizar el firmware pues se guarda la nueva versión en el host y el driver lo manda al chip en el siguiente encendido. No hay que reemplazar el chip o usar un programa especial.

NetChip NET2888. El chip NET2888 no contiene un CPU de propósito general o memoria. Este tiene solamente un controlador USB y una interfaz a un bus de datos genérico, con el cual se puede conectar a cualquier CPU que tenga un bus complementario.

USBN9603 de Nacional Semiconductor. Este chip requiere una interfaz a un microcontrolador con un bus de datos paralelo, una interfaz Microware o solo cuatro I/O controladas enteramente en el firmware.

PDIUSBD11/12 de Philips Semiconductors. Philips semiconductors ofrece opciones adicionales para un control USB sencillo en los chips PDIUSBD11 y PDIUSBD12. Los dos chips son similares excepto por sus buses de datos externos ambos chips son full-speed

1.2. Control digital de movimiento para motores CD

En el control de movimiento para motores CD es necesario primero comprender la situación física. De esta manera se pueden diseñar e implementar sistemas controladores de alguno o varios parámetros que definen el movimiento en un motor y por lo tanto en su carga mecánica.

1.2.1. Principios básicos

Un sistema electromecánico para el posicionamiento en esencia puede ser modelado y comprendido como el caso de un motor CD impulsando una carga inercial. La figura 1.1 muestra el modelo electromecánico de un motor CD [16].

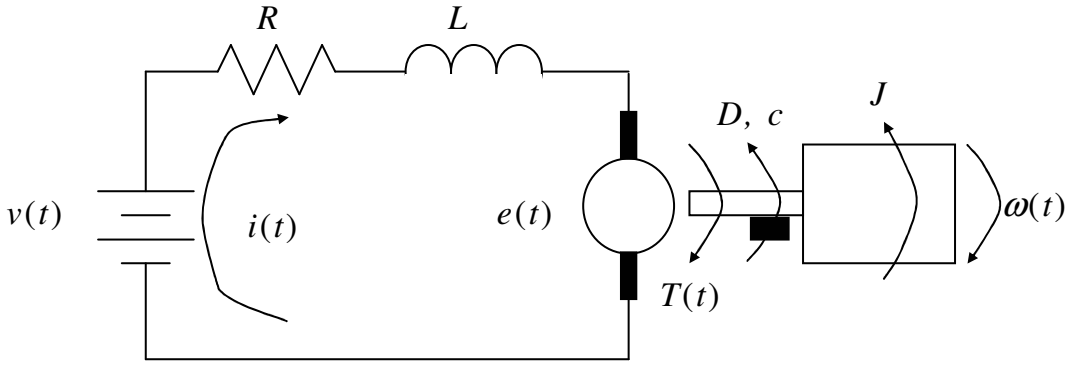

Figura 1.1. Modelo electromecánico del motor CD.

En donde los parámetros eléctricos son:

$v(t)$ es el voltaje aplicado en (V).
R es la resistencia en la armadura en (Ω).
L es la inductancia en la armadura (H).

$e(t)$ es la fuerza electro- motriz contraria en (V).
$i(t)$ es la corriente en la armadura en (A).

Y los parámetros mecánicos son:

$T(t)$ es el torque aplicado en (N m).
D es el coeficiente viscoso de fricción en (N m s)
c es el coeficiente de fricción en (N m)
J es la inercia en (kg m² / s²)
$\omega(t)$ es la velocidad angular en (rad / s)

Aplicando las leyes de Kirchhoff a la figura

$$v(t) = Ri(t) + L\frac{di(t)}{dt} + e(t)$$

Para un motor CD la fuerza electromotriz contraria es proporcional a la velocidad angular,

$$e(t) = k_e\omega(t)$$

En donde k_e es la constante de fuerza electromotriz en (N m / A). Sustituyendo,

$$v(t) = Ri(t) + L\frac{di(t)}{dt} + k_e\omega(t)$$

En Laplace

$$V(s) = R\ I(s) + L\ s\ I(s) + k_e\Omega(s) \qquad (1.1)$$

Por otra parte, aplicando las leyes de Newton a la figura,

$$T(t) = J\frac{d\omega(t)}{dt} + D\omega(t) + c$$

Para un motor CD la torca es proporcional a la corriente de armadura,

$$T(t) = k_t\ I(t)$$

En donde k_t es la constante de torca del motor en (N m / A). Sustituyendo,

$$k_t i(t) = J\frac{d\omega(t)}{dt} + D\omega(t) + c$$

En Laplace

$$k_t I(s) = J\ s\ \Omega(s) + D\ \Omega(s) \qquad (1.2)$$

De las ecuaciones (2.1) y (2.2) es fácil demostrar que la función de transferencia en modo de velocidad del sistema es

$$\frac{\Omega(s)}{V(s)} = \frac{k_t}{LJs^2 + (RJ + LD)s + (RD + k_e k_t)} \tag{1.3}$$

En donde el denominador conforma la ecuación característica, EC,

$$EC = LJs^2 + (RJ + LD)s + (RD + k_e k_t) \tag{1.4}$$

Encontrando las raíces de (1.4),

$$s_1, s_2 = \frac{-(RJ + LD) \pm \sqrt{(RJ + LD)^2 - 4LJ(RD + k_e k_t)}}{2LJ} \tag{1.5}$$

Desarrollando el radical de (1.5)

$$\sqrt{(RJ + LD)^2 - 4LJ(RD + k_e k_t)} = \sqrt{(RJ)^2 + 2RJLD + (LD)^2 - 4RJLD - 4JLk_e k_t}$$

$$\sqrt{(RJ + LD)^2 - 4LJ(RD + k_e k_t)} = \sqrt{(RJ)^2 - 2RJLD + (LD)^2 - 4JLk_e k_t}$$

$$\sqrt{(RJ + LD)^2 - 4LJ(RD + k_e k_t)} = \sqrt{(RJ - LD)^2 - 4JLk_e k_t}$$

Asumiendo $LD \approx 0$

$$\sqrt{(RJ + LD)^2 - 4LJ(RD + k_e k_t)} = \sqrt{(RJ)^2 - 4JLk_e k_t}$$

Completando el binomio cuadrado

$$\sqrt{(RJ + LD)^2 - 4LJ(RD + k_e k_t)} = \sqrt{(RJ)^2 - 4JLk_e k_t + \frac{4L^2 {k_e}^2 {k_t}^2}{R^2} - \frac{4L^2 {k_e}^2 {k_t}^2}{R^2}}$$

$$\sqrt{(RJ + LD)^2 - 4LJ(RD + k_e k_t)} = \sqrt{\left[RJ - \frac{2Lk_e k_t}{R}\right]^2 - \frac{4L^2 {k_e}^2 {k_t}^2}{R^2}}$$

Asumiendo $\frac{4L^2 {k_e}^2 {k_t}^2}{R^2} \approx 0$

$$\sqrt{(RJ + LD)^2 - 4LJ(RD + k_e k_t)} = RJ - \frac{2Lk_e k_t}{R}$$

Sustituyendo el radical anterior y $LD \approx 0$ en (1.5)

$$s_1, s_2 = \frac{-RJ \pm \left(RJ - \frac{2Lk_e k_t}{R}\right)}{2LJ} \tag{1.6}$$

Para s_1 de (1.6)

$$s_1 = \frac{-RJ \pm \left(RJ - \frac{2Lk_e k_t}{R} \right)}{2LJ} = \frac{-\frac{2Lk_e k_t}{R}}{2LJ} = -\frac{k_e k_t}{RJ} \tag{1.7}$$

Para s_2 de (1.6)

$$s_2 = \frac{-RJ \pm \left(RJ - \frac{2Lk_e k_t}{R} \right)}{2LJ} = \frac{-2RJ + \frac{2Lk_e k_t}{R}}{2LJ} = \frac{-2R^2 J - 2Lk_e k_t}{2RLJ} \tag{1.8}$$

Asumiendo $2Lk_e k_t \approx 0$

$$s_2 = \frac{-2R^2 J}{2RLJ} = -\frac{R}{L}$$

Con las raíces s_1 y s_2 en las ecuaciones (1.7) y (1.8), la función de transferencia (1.3) se puede escribir de la siguiente manera

$$\frac{\Omega(s)}{V(S)} = \frac{k_t}{\left(s + \frac{k_e k_t}{RJ} \right)\left(s + \frac{R}{L} \right)} = \frac{k_t}{(s - s_1)(s - s_2)}$$

De otra forma, la función de transferencia en modo de velocidad es

$$\frac{\Omega(s)}{V(S)} = \frac{k_t}{(s + a)(s + b)} \tag{1.9}$$

En donde

$$a = -s_1 = \frac{k_e k_t}{RJ} \tag{1.10}$$

$$b = -s_2 = \frac{R}{L} \tag{1.11}$$

Por otra parte, la velocidad angular, $\omega(t)$, es la derivada de la posición angular, $\theta(t)$, es decir

$$\omega(t) = \frac{d\theta(t)}{dt}$$

En Laplace

$$\Omega(s) = s\ \Theta(s) \tag{1.12}$$

Sustituyendo (1.12) en (1.9) se obtiene la función de transferencia en modo de posición

$$\frac{\Theta(s)}{V(s)} = \frac{k_t}{s(s + a)(s + b)} \tag{1.13}$$

Con a y b determinadas por las ecuaciones (1.10) y (1.11) respectivamente.

La ecuación (1.13) describe el comportamiento de la posición para una entrada de voltaje para cualquier tipo de motor. La ecuación (1.13) tiene tres polos: uno en el origen y dos en $-a$ y $-b$ respectivamente. Sin embargo, para motores pequeños sólo existen dos polos: uno en el origen y otro en $-a$. Por lo tanto la ecuación (1.13) se puede simplificar aún más de la siguiente manera

$$G_p(s) = \frac{\Theta(s)}{V(s)} = \frac{k}{s(s+a)} \tag{1.14}$$

En la ecuación (1.14) k_t es reemplazada por una constante k con fines de simplificación en la notación. Entonces en la ecuación (1.14), $G_p(s)$, es utilizada en éste trabajo para representar la función de transferencia en modo de posición para un sistema electro-mecánico como el de la figura. La planta entonces se puede modelar a través del siguiente diagrama de bloques de la figura 1.2.

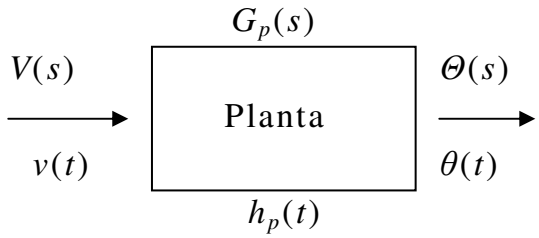

Figura 1.2. Diagrama a bloques de la planta.

En donde $h_p(t) = L^{-1}\{G_p(s)\}$ es la respuesta al impulso de la planta.

1.2.2. Sistema en lazo cerrado

Una vez obtenida la función de transferencia de la planta, el sistema en lazo cerrado que representa el esquema de control de posición de la figura es el mostrado en la figura 1.3 [10, 11].

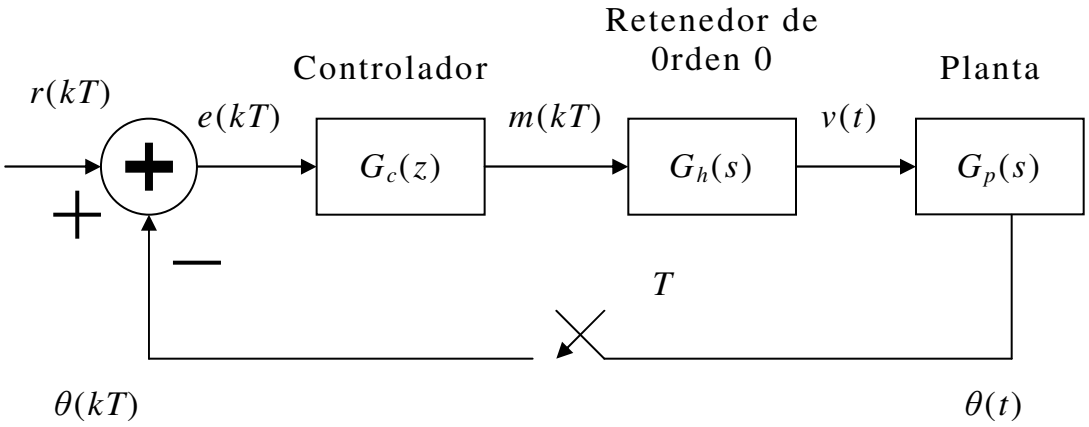

Figura 1.3. Sistema en lazo cerrado para el control de posición.

Que contiene tanto elementos discretos como continuos en el tiempo. En la figura 1.3:

$r(kT)$: es la señal de referencia discreta o "set point" especificado por el operador.
$e(kT)$: es la señal discreta de error.
$\theta(kT)$: es la posición angular discreta actual de la planta.
$\theta(t)$: es la posición angular continua actual de la planta.
$m(kT)$: es la señal de alimentación de voltaje discreta.
$v(t)$: es la señal de voltaje de alimentación continua.
$G_c(z)$: es la función de transferencia discreta del controlador digital.
$G_h(s)$: es la función de transferencia continua del retenedor de orden cero que convierte la entrada discreta en salida continua en el tiempo.
$Gp(s)$: es la función de transferencia continua de la planta.

Considerando los elementos anteriores, el propósito del trabajo consiste entonces en diseñar un controlador digital que proporcione elementos para el control de posición angular en la planta introduciendo el menor error posible.

El sistema híbrido de la figura 1.3 puede ser representado por el siguiente sistema completamente discreto equivalente, mostrado en la figura 1.4.

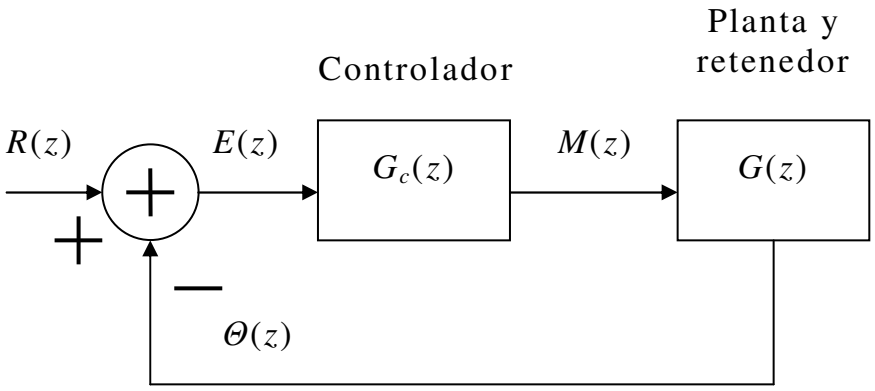

Figura 1.4. Sistema discreto en lazo cerrado para el control de posición.

En donde $G(z)$ representa la respuesta conjunta tanto del retenedor como de la planta en el tiempo discreto, procedimiento conocido como "discretización de la planta".

1.2.3. Discretización de la planta

La discretización de la planta consiste en obtener la respuesta conjunta de la planta y del controlador de orden cero en un sistema discreto $G(z)$ dado por

$$G(z) = Z\{\, G_h(s)G_p(s)\,\} \tag{1.15}$$

Para el retenedor de orden cero [10]

$$G_h(s) = \frac{1-\exp(-s)}{s}$$

Para la planta

$$G_p(s) = \frac{k}{s(s+a)}$$

Sustituyendo $G_h(s)$ y $G_p(s)$ en (1.15)

$$G(z) = Z\left\{\frac{1-\exp(-s)}{s}\,\frac{k}{s(s+a)}\right\}$$

Para una transformación que incluye el término $(1-\exp(-s))/s$

$$G(z) = k(1-z^{-1})Z\left\{\frac{k}{s^2(s+a)}\right\} \tag{1.16}$$

Resolviendo la transformada de (1.16)

$$Z\left\{\frac{k}{s^2(s+a)}\right\} = \frac{1}{a^2}\,Z\left\{\frac{a^2}{s^2(s+a)}\right\}$$

$$Z\left\{\frac{k}{s^2(s+a)}\right\}=\left\{\frac{1}{a^2}\,\frac{\left[(aT-1+\exp(-aT))+(1-\exp(-aT)-aT\exp(-aT))z^{-1}\right]z^{-1}}{(1-z^{-1})^2(1-\exp(-aT)z^{-1})}\right\}$$

1.2.4. Controlador digital

El esquema de control PID ha sido usado de manera exitosa en muchos sistemas de control industrial por más de medio siglo. El principio básico del esquema de control PID es que actúa sobre la variable a ser manipulada (error actuante, $e(t)$) a través de una combinación apropiada de las tres acciones de control: acción de control proporcional (donde la acción de control es proporcional al error); la acción de control integral (donde la acción de control es proporcional a la integral del error) y la acción de control derivativa (donde la acción de control es proporcional a la derivada del error) [1].

Desde un punto de vista práctico, el término proporcional reduce el tiempo de subida, el término integral reduce el error de estado estacionario, y el término derivativo reduce el sobrepaso, como se muestra en la figura 1.5 para un sistema realimentado.

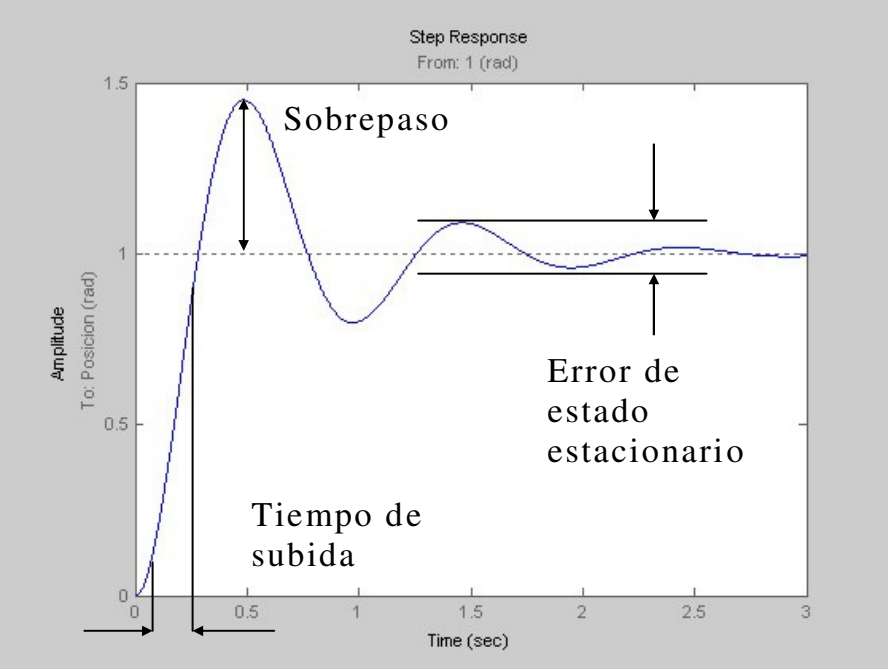

Figura 1.5. Controlador digital.

La acción de control PID en controladores analógicos en el tiempo está dada por

$$m(t)=K\left[e(t)+\frac{1}{T_i}\int_0^t e(t)dt+T_d\frac{de(t)}{dt}\right]$$

En donde

K es el término proporcional.
T_i es el término integral.
T_d es el término derivativo.

La acción de control PID en controladores digitales, como el usado en este trabajo, en el dominio Z está dada por

$$G_c(z) = \frac{M(z)}{E(z)} = K_p + \frac{K_i}{1 - z^{-1}} + K_d(1 - z^{-1})$$

Desarrollando la última expresión

$$G_c(z) = \frac{M(z)}{E(z)} = \frac{(K_p + K_i + K_d)z^2 - (K_p + 2K_d)z + K_d}{z^2 - z} \tag{1.17}$$

En donde

K_p es la constante proporcional.
K_i es la constante integral.
K_d es la constante derivativo.

A partir de la figura 1.4 es fácil demostrar la siguiente función de transferencia en lazo cerrado

$$G_l(z) = \frac{\Theta(z)}{R(z)} = \frac{G_c(z)G(z)}{1 + G_c(z)G(z)} \tag{1.18}$$

que representa la salida en posición angular $\Theta(z)$ de la planta a la entrada especificada en el set point $R(z)$.

1.2.5. Sintonización del controlador

La sintonización del controlador consiste en seleccionar los parámetros K_p, K_d y K_i en el sentido de mejorar el desempeño del sistema realimentado. En la práctica se busca reducir el tiempo de retardo, reducir el sobrepaso y reducir el error de estado estacionario.

Por otra parte, la simulación en MatLab para la búsqueda de los parámetros del controlador, es de gran ayuda y facilita la implementación en hardware, ver anexo 5.3.

No obstante que existen técnicas analíticas (Ziegler –Nichols [11]) para especificar los parámetros del controlador, nosotros optamos por sintonizar los parámetros manualmente (prueba y error) para observar y experimentar diversas situaciones. Si bien nuestro procedimiento podría ser evaluado cualitativamente y sintonizado heurística mente, en principio nos planteamos los siguientes parámetros de diseño:

- Tiempo de subida cercano a un segundo.

- Sobrepaso nulo.
- Error en estado estacionario nulo.

Parámetros que son seleccionados a partir de nuestra propia experiencia.

1.3. Codificadores ópticos

Un codificador óptico digital es un dispositivo que convierte el movimiento en una secuencia de pulsos digitales. Ya sea mediante la cuenta de un solo bit o por el desciframiento de un paquete de bits, los pulsos pueden ser convertidos a un sistema de posición relativa o absoluta. Los codificadores tienen configuraciones lineales y angulares, pero el tipo más común es el angular. Los codificadores angulares están fabricados en dos formas básicas: el codificador absoluto, donde una sola palabra digital corresponde a cada posición rotacional de la flecha; y el codificador incremental, el cuál produce pulsos digitales conforme gire la flecha, permitiendo mediciones de la posición relativa de la flecha. La mayoría de los codificadores angulares están compuestos de un disco codificado de cristal o plástico que contiene un patrón radial colocado fotográficamente el cuál es organizado en pistas. Cada que una línea radial en cada pista interrumpe el rayo entre el foto-emisor y el detector, se producen los pulsos digitales como se muestra en la figura 1.6 [14].

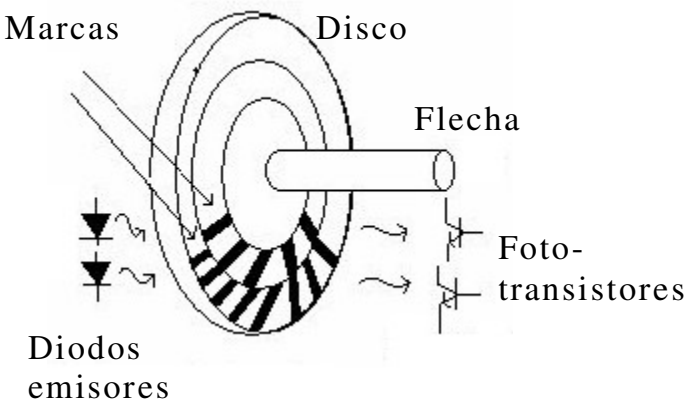

Figura 1.6. Codificador angular incremental.

1.3.1. Codificadores absolutos

El disco óptico de los codificadores absolutos esta diseñado para producir una palabra digital que distingue n posiciones distintas de la flecha. Por ejemplo, si hay ocho pistas, el codificador es capaz de producir 256 distintas posiciones de una resolución angular de 1.406 (360/256) grados. Los tipos más comunes de codificación numérica

usados en los codificadores absolutos son los códigos gray y binario. Para ilustrar la acción de un codificador absoluto, el código gray y el código natural binario de una pista patrón para un codificador de 4 bits, están mostrados en las figuras 1.7 y 1.8 respectivamente.

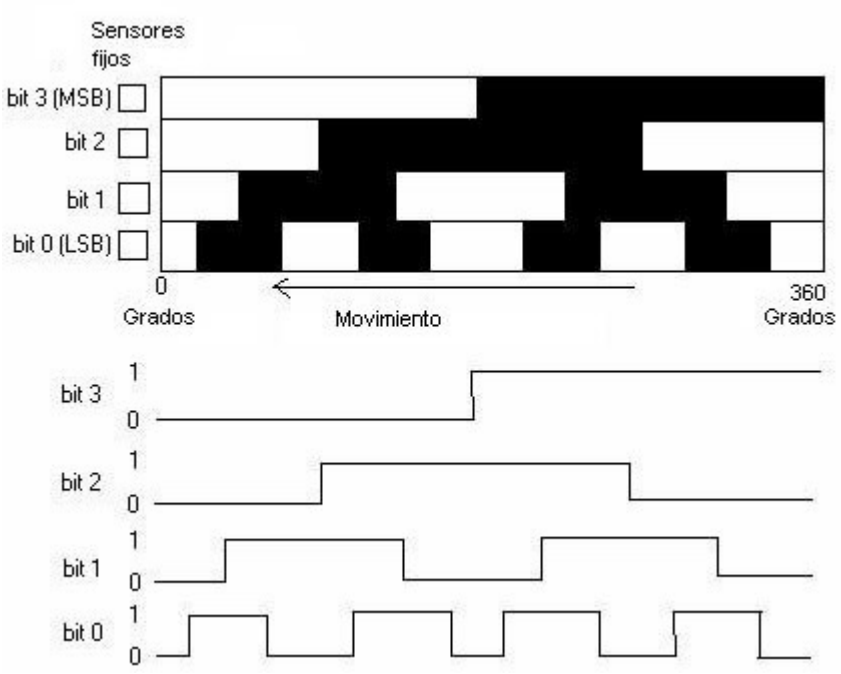

Figura 1.7. Código Gray de 4 bits.

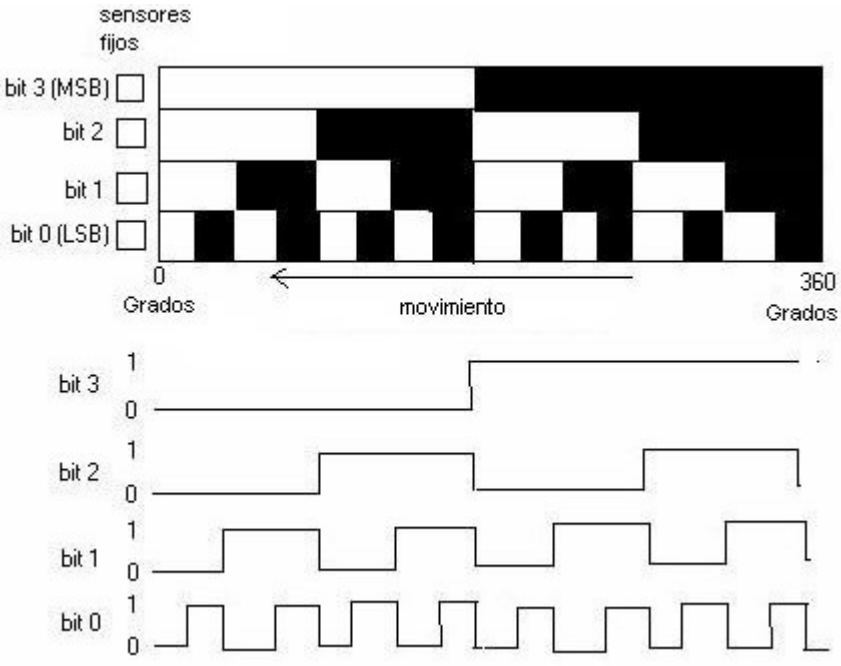

Figura 1.8. Código binario de 4 bits.

Los patrones lineales y los diagramas de tiempo asociados son lo que los fotorreceptores registran en la medida que las pistas circulares del disco rotan con la flecha.

Los códigos binarios de salida para ambos esquemas de codificación están mostrados en la tabla 1.2.

Código decimal	Rango de rotación (grad.)	Código binario	Código Gray
0	0-22.5	0000	0000
1	22.5-45	0001	0001
2	45-67.5	0010	0011
3	67.5-90	0011	0010
4	90-112.5	0100	0110
5	112.5-135	0101	0111
6	135-157.5	0110	0101
7	15.75-180	0111	0100
8	180-202.5	1000	1100
9	202.5-225	1001	1101
10	225-247.5	1010	1111
11	247.5-270	1011	1110
12	270-292.5	1100	1010
13	292.5-315	1101	1011
14	315-337.5	1110	1001
15	337.5-360	1111	1000

Tabla 1.2. Códigos Gray y binario natural.

El código Gray esta diseñado de tal manera que solo una pista (un bit) cambiara de estado por cada transición en la cuenta, a diferencia del código binario donde múltiples pistas (bits) cambian en ciertas transiciones de la cuenta. Estos efectos pueden ser vistos claramente en la tabla 1.2. Para el código Gray, la incertidumbre durante una transición es solo una cuenta, a diferencia del código binario, donde la incertidumbre puede ser de múltiples cuentas.

Como el código gray provee datos con una mínima incertidumbre y el código binario natural es la opción preferida par la interfase directa con computadoras, otros dispositivos digitales, un circuito para convertir de gray a binario es recomendable.

1.3.2. Codificador incremental

El codificador incremental, algunas veces llamado codificador relativo, es más simple en diseño que el codificador absoluto. Consiste de dos pistas y dos sensores cuyas salidas son llamadas canal A y canal B. Conforme la flecha rota, los trenes de pulsos ocurren en esos canales a una frecuencia proporcional a la velocidad de la flecha, y la relación de fase entre las señales proporciona la dirección de rotación.

El disco con el código patrón y las señales de salida A y B están ilustradas en la figura 1.9. Por el conteo del número de pulsos y conociendo la resolución del disco, el movimiento angular puede ser medido. Los canales A y B son usados para determinar la dirección de rotación evaluando que canal "encabeza" al otro. Las señales desde los dos canales están desfasadas un cuarto de ciclo una con respecto a la otra y son conocidas como señales en cuadratura.

En algunas ocasiones un tercer canal de salida, llamado INDEX, produce un pulso por revolución, el cual es usado para el conteo de revoluciones. También es usado como referencia para definir una posición de origen del sistema.

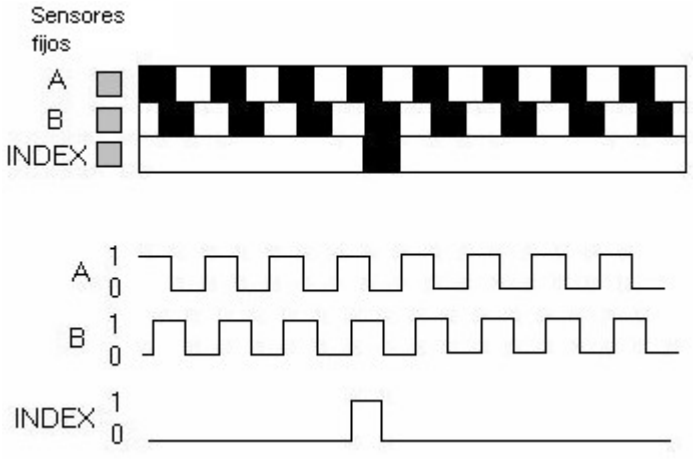

Figura 1.9. Codificador incremental.

La figura 1.9 ilustra dos pistas separadas para los canales A y B, pero una configuración más común usa una sola pista con los sensores A y B desfasados ¼ de ciclo sobre la pista para proporcionar la misma señal patrón. Un disco codificado con una sola pista es más simple y más barato de fabricar.

Las señales en cuadratura A y B pueden ser descodificadas para proporcionar la dirección de rotación como se muestra en la figura 1.10. Descodificando las transiciones de A y B y usando circuitos

lógicos secuénciales, se pueden proveer tres diferentes resoluciones de los pulsos de salida: 1X, 2X y 4X.

La resolución 1X solo proporciona un pulso por cada ciclo en una de las señales A o B, la resolución 4X proporciona un pulso en todas las transiciones en las señales A y B proporcionando 4 veces la resolución 1X. La dirección de rotación es determinada por el nivel de una señal durante la transición de la segunda señal. Por ejemplo, en el modo 1X, A = $\downarrow$ con B = 1 implica un pulso de, y B = $\downarrow$ con A = 1 implica un pulso de.

Si solo tenemos un canal de salida A o B, podría ser imposible determinar la dirección de rotación. Además, si la fleche vibra en el momento en que ocurre una transición en la señal podría producir pulsos incorrectos.

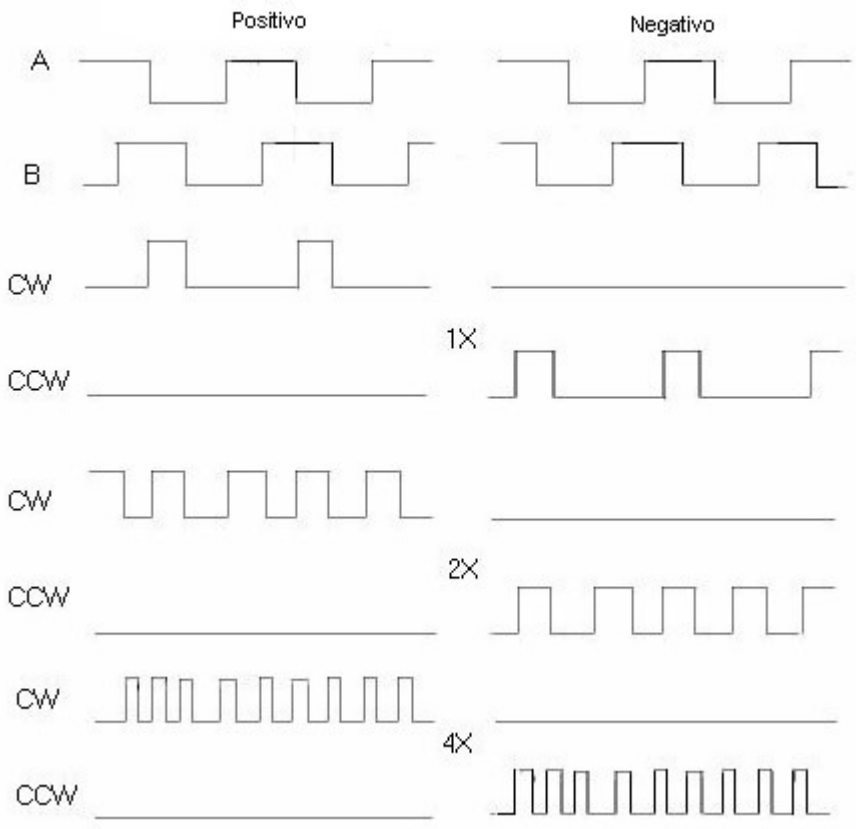

Figura 1.10. Detección del sentido e incremento de la resolución.

1.4. Microcontrolador PIC16C765

Los PIC16C7x5 son dispositivos de bajo costo, alto desempeño, CMOS, microcontroladores de 8 bits en la familia de medio rango PIC16CXX [9].

Emplean una avanzada arquitectura RISC. La familia PIC16C745/765 tiene un aumento de características, ocho niveles de stack y múltiples interrupciones externas e internas. Las instrucciones separadas y los bus de datos de la arquitectura Hardvard permite un ancho de 14 bits para la instrucción con los 8 bits separados de datos. Las dos etapas de la instrucción por *pipeline* permite que todas las instrucciones se ejecuten en un único ciclo, excepto para saltos los cuales requieren dos ciclos. Maneja un total de 35 instrucciones.

El PIC16C745 tiene 22 I/O, mientras el PIC16C765 tiene 33 I/O. Los dos tienen una RAM de 256 bytes, además de tres temporizadores, dos módulos de Captura/Comparación/PWM y dos puertos seriales. El USB 1.1 provee la comunicación del bus. El USART (Universal Synchronous Asynchronous Receiver Transmiter) es también conocido como la Interfase de Comunicación Serie, SCI. El PIC16C745 tiene cinco canales de alta velocidad de 8 bits para un convertidor A/D mientras que el PIC16C765 ofrece 8 canales. La resolución de 8 bits es ideal para aplicaciones de bajo costo con interfaces analógicas (control de termostato, censado de temperatura).

Los PIC16C745/765 tienen la característica especial que reducen componentes externos, reduciendo el costo, aumentando la fiabilidad del sistema y reduciendo el consumo de energía. Hay cuatro opciones para los osciladores, de la cual EC es para una fuente de reloj regulada externa, E4 es para una fuente de reloj regulada externa con PLL habilitado, HS es para cristales y osciladores de alta velocidad y H4 para cristales y osciladores de alta velocidad con PLL habilitado. El modo *SLEEP* provee de un ahorro de consumo. El usuario puede despertar al chip mediante alguna interrupción externa, interna o un reset.

La alta fiabilidad del Watchdog Timer (WDT) con un oscilador RC dedicado al chip, provee una detección contra el software trabado y también de una manera de despertar el chip del estado SLEEP.

La versión de empacado CERDIP, el cual se borra mediante rayos UV, es ideal para actualización de código y programas piloto así como desarrollo de prototipos. Mientras que las OTP (One Time Programmable) es recomendable una vez que el código esta listo y además, para quienes necesitan la flexibilidad de actualizaciones de código frecuentemente y pequeños volúmenes de aplicación.

Los PIC745/765 están basados en la arquitectura Harvard en la cual el programa y los datos son accedidos desde memorias separadas usando buses separados. Por tanto, el opto-código es de 14 bits y las instrucciones se llevan a cabo en un ciclo. En las dos etapas de

pipeline se superponen tanto la ejecución de una instrucción y la búsqueda de la siguiente. Consecuentemente la mayoría de las instrucciones se ejecutan en un ciclo de 166.667 ns a 24 MHz.

Ambos, PIC745/765 son direccionados directa o indirectamente. Todas las funciones especiales de los registros, incluyendo el contador del programa, es mapeado en la memoria de datos. También contienen un ALU de 8 bits y un registro de trabajo, la ALU que es capaz de realizar operaciones lógicas y aritméticas. Típicamente un operando esta en el registro de trabajo W (de 8 bits) mientras el otro operando esta en un registro de la memoria de datos o es una constante. Dependiendo de la operación, la ALU afecta los valores de carry (C), digit carry (DC) and zero (Z) en el registro de estado.

1.4.1. Organización de la memoria

Memoria de programa. Los PIC16C745/765 cuentan con un contador de programa de 13 bits capaz de direccionar un espacio de memoria de programa de 8K x 14. El rango de direcciones es 0000h – 1FFFh para todos los dispositivos. El vector de reset es al 0000h y el vector de interrupción es al 0004h, ver figura 1.11.

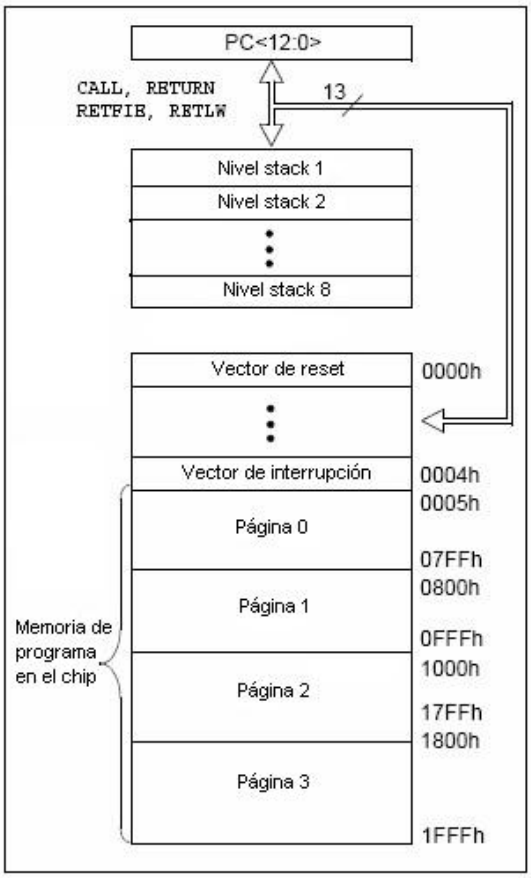

Figura 1.11. Memoria de programa.

Memoria de datos. La memoria de datos es partida en múltiples bancos los cuales contienen los registros de propósito general (GPR) y los registros de funciones especiales (SFR). Mediante los bits RP0 y RP1 del registro de estado son seleccionados estos bancos, ver figura 1.12

Figura 1.12. Memoria de datos.

Registros de funciones especiales. Estos registros son usados por el CPU y módulos periféricos para controlar las operaciones deseadas por el dispositivo. Estos registros están implementados en una RAM. Están clasificados en dos partes: núcleo y periféricos. Los primeros están relacionados con el corazón de las funciones mientras los segundos con las características de operación.

Registro de estado. Este registro el estado aritmético de la ALU, el estado Reset y los bits para la selección de bancos (RP0, RP1) para los datos de memoria. Por ejemplo, aquí se encuentran los bits de estado Z, DC, C los cuales son afectados mediante alguna operación lógica o de otro tipo.

Registro de opción. En este registro se puede leer y escribir, contiene varios bits de control para configurar el pre-escalador TMR0/WDT, la interrupción externa INT, el timer0 TMR0 y los *pull-ups* del PORTB.

Registro INTCON. Contiene varias banderas para el registro TMR0, cambios en el puerto RB y los pines de interrupción externa RB0/INT.

Registro PIE1. Este registro contiene los bits para las interrupciones periféricas.

Registro PIR1. Contiene las banderas para las interrupciones periféricas,

Registro PIE2.Este registro contiene los bits para las interrupciones periféricas del CCP2.

Registro PIR2. Este registro contiene las banderas de la interrupción del CCP2.

Registro PCON. El registro de control de poder (PCON) contiene las banderas que permiten la diferenciación entre *Power-on Reset*

(POR), *Brown-out* Reset (BOR), *Watchdog* Reset (WDT) reset externo (MCLR).

PCL y PCLATH. El contador del programa es de 13 bits. El byte bajo viene desde el registro PCL, el cual es escribible y leíble. Los bits altos (8 al 12) no son leíbles pero se puede escribir en ellos indirectamente a través del registro PCLATH. En cualquier Reset, los bits altos del PC pueden ser limpiados.

Stack. Los PIC16C745/765 tienen un *stack* en hardware de 8 x 13 bits. El espacio del *stack* no es parte ni del programa ni de la memoria y además, el *stack pointer* no es escribible ni leíble. El contador del programa es puesto en el *stack* cuando es ejecutada una instrucción CALL o una interrupción que cause salto.

El *stack* opera como un buffer circular, esto es, una vez que se ha usado ocho veces, la novena vez se sobrescribe el valor que había sido escrito en la primera vez, la décima vez se escribe donde estaba la segunda y así sucesivamente.

Paginado del programa de memoria

Los PIC16CXX son capaces de direccionar un bloque continuo de palabras de 8k de la memoria de programa. Las instrucciones GOTO Y CALL proveen de solo 11 bits de dirección para permitir saltar en cualquier página de la memoria de programa de 2k. Cuando se ejecutan las instrucciones CALL o GOTO, los dos bits mas altos de la dirección son proveídos por el PCLATH <4:3>. Cuando se hacen estas instrucciones el usuario debe estar seguro de que ha sido programada esa parte de la página y entonces queda archivada. Si se regresa de la instrucción CALL o una interrupción, los 13 bits del contador del programa (PC) son puestos en el *stack*. La manipulación del PCLATH <4:3> no es requerida para las instrucciones de regreso.

Direccionamiento indirecto. Registros INDF y FSR

El registro INDF no es un registro físico, por lo cual, usar este registro se esta usando el direccionamiento indirecto. Escribiendo en el registro seleccionador de archivos (FSR) un valor, accedemos a esa parte de la memoria. Leyendo el registro INDF leemos indirectamente una dirección. Obtenemos un direccionamiento de 9 bits usando los 8 bits del FSR concatenado un bit (el 7) del IRP.

1.4.2. Puertos de Entrada/Salida

Algunos de estos puertos están multiplexados con una función alternativa de las características del periférico en el dispositivo.

Cuando una función esta habilitada, en general no debe funcionar ese pin como de propósito general I/O.

Puerto A y el registro TRISA

El puerto a es un retenedor de 6 bits. El pin RA4/T0CK1 es una entrada *Schimitt-Trigger* y una salida abierta de drenaje. Los otros pines RA del puerto tienen niveles de entrada TTL e impulsores de salida CMOS. Todos los pins tienen bits de dirección de datos (registros TRIS), con los cuales se puede configurar estos pines como entrada o salida.

Poniendo un bit en el registro TRISA ponemos la salida correspondiente al impulsor en modo de alta impedancia. Limpiando un bit ponemos el contenido del match de salida en el pin o pines seleccionados. El pin RA4 esta multiplexado con el modulo de reloj del timer0.

En los PIC16C745/765, el puerto A esta multiplexado con una entrada analógica y un voltaje de referencia analógico. La operación de cada pin es ya sea poniendo en alto o limpiando algún bit en el registro de control analógico digital ADCON1.

El registro TRISA controla las direcciones de los pines RA.

Puerto B y el registro TRISB

El puerto B es de 8 bits de ancho y bidireccional. Poniendo bit en el registro TRISB tenemos la correspondiente salida en alta impedancia. Limpiando un bit en este registro ponemos el contenido del latch en el pin o pines correspondientes.

Cada pin tiene una correspondiente *weak pull-up*, limpiando el bit RBPU podemos cambiar todas las pull ups. Estas *weak pull-up* son apagadas automáticamente cuando el puerto es configurado como salida. Las *pull-up* son deshabilitadas cuando se prende el dispositivo o en un reset.

Cuatro pines del puerto B <7:4> tienen la característica de hacer una interrupción si están configurados como entrada. El pin RB0/INT es para la interrupción externa y se configura usando el bit INTEDG (registro de opción <6>).

Puerto C y el registro TRISC

El puerto C es bidireccional de 5 bits. Cada pin es configurado como entrada o salida de acuerdo con el registro TRISC. Este puerto esta multiplexado con varias funciones. Cada pin tiene impulsores de

entrada *Schimitt trigger*. Hay que tener cuidado de cómo están configurados los pines.

Puerto D y el registro TRISD.

El puerto D es de 8 bits con entrada *Schimitt trigger*. Cada pin esta individualmente configurado como entrada o salida. Puede ser configurado como puerto paralelo esclavo por medio del bit de control PSPMODE (TRISE<4>). Este puerto no lo tiene el PIC16C745.

Puerto E y el registro TRISE.

El puerto E tiene tres pines. RE0/RD/AN5, RE1/WR/AN6 y RE2/CS/AN7 los cuales son configurados individualmente como entrada o salida. También tienen entrada *Schimitt trigger*.

Cuando esta en alto el bit PSPMODE (TRISE <4>) el puerto funciona como entrada y los bits <2:0> del TRISE deben estar en alto, esto es, configurados como entradas digitales. Y además, el registro ADCON1 debe estar configurado como entrada/salida digital. En este modo, las entradas son TTL.

Los pines de este puerto pueden ser multiplexados con una entrada analógica. La operación de estos pines esta en el registro ADCON1. Cuando se usaran los pines como entrada analógica, deben estar a 0.

También este puerto se puede usar como un paralelo esclavo. Como nota a resaltar, este puerto esta configurado por default como de entrada analógica.

El Timer0

Tiene las siguientes características:

- Reloj/contador de 8 bits.
- Se puede leer y escribir.
- Prescalador programable por software de 8 bits.
- Selección de reloj interno o externo.
- Interrupción por desborde desde FFh a 00h.
- Selección de corte para reloj externo.

El modo reloj es seleccionado limpiando el bit TOSC (registro de opciones <5>). En este modo, el timer0 incrementara cada ciclo de instrucción.

El modo contador es seleccionado poniendo en alto el bit TOSC (registro de opciones <5>). En este modo, el timer0 se incrementara ya sea en pulso de subida o de bajada que entra por el pin

RA4/TOCK1. Limpiando el bit TOSE (registro de opciones <4>) se configura como pulso de subida.

La interrupción es generada cuando hay un desborde desde el FFh al 00h. Este desborde pone en alto el bit TOIF (INTCON<2>). La interrupción puede ser enmascarada limpiando el bit TOIE (INTCON<5>). El bit TOIF puede ser limpiado por software antes de que vuelva a ocurrir otra interrupción.

El pre-escalador es compartido tanto por el timer0 como por el *watchdog*, si esta asignado a uno, no lo esta para el otro y viceversa. No se puede leer y ni escribir en el.

Los bits PSA y PS (registro de opciones <2:0>) determinan el del pre-escalador y el radio.

El Timer1

El timer1 es un reloj/contador de 16 bits partido en dos registros de 1 bytes. El registro TMR1 se incrementa desde el 0000h hasta el FFFFh. Una interrupción es generada por un desborde y se prende la bandera TMR1IF (PIR1<0>). Esta interrupción puede ser habilitada o deshabilitada limpiando o alzando el bit TMR1IE (PIE1<0>).
Puede operar en uno de los dos modos:

- Como temporizador
- Como contador

El modo de operación es seleccionado por el bit TMR1CS (T1CON<1>). En modo temporizador, se incrementa el ciclo de instrucción mientras que en modo contador se incrementa en pulso de subida.
El timer1 puede ser habilitado/deshabilitado alzando/limpiando el bit de control TMR1ON (T1CON<0>).
El modo contador es seleccionado poniendo en alto el bit TMR1CS y se incrementa el timer en cada pulso de subida que entra en el pin RC1/T1OSCI/CCP2 cuando el bit T1OSCEN esta en alto o cuando entra por el pin RC0/T1OSC0/T1CKI cuando el bit T1OSCEN esta en bajo.

El Timer2

El timer2 es de 8 bits con un pre-escalador y un post-escalador. Puede ser usado también en modo PWM. El registro TMR2 es escribible, leíble y es limpiado por cualquier reset.
Hay tres opciones para el pre-escalador 1:1, 1:4 y 1:16 los cuales se seleccionan por medio de los bits T2CKPS <1:0> . El periodo de 8 bits se escribe en el registro PR2. El timer2 se incrementa desde el 0000h hasta el valor que tenga el PR2. La inicialización del timer2 es mediante el bit de control TMR2ON (T2CON<2>).

Capítulo 2

Sistema propuesto

En el presente capítulo se describe el sistema propuesto para el control de un servomotor. El sistema consta de hardware y software desarrollados a partir de herramientas básicas para controlar un motor CD al cual le fue adaptado un codificador incremental. El sistema incluye un sistema electrónico sobre la base del microcontrolador PIC16C765, software de operación desarrollado en Visual C para la operación del sistema mediante una PC e interfase USB entre el sistema electrónico y la PC.

2.1. Esquema general

En forma general, una MMC es un dispositivo electromecánico con tres grados de libertad, es decir, tres ejes de movimiento con un codificador lineal incremental y motor eléctrico impulsor por cada eje. El desplazamiento en su dispositivo sensor, llamado *palpador mecánico de contacto*, es registrado en el espacio mediante los tres codificadores ópticos y generado por tres motores eléctricos, como se muestra en la figura 2.1.

Cuando el palpador entra en contacto con algún objeto a medir, se genera una señal de disparo de referencia que indica el instante preciso en el cual se deben registrar las lecturas de los tres codificadores para validar una medición. En la mayoría de las MMC comerciales se utiliza un sistema electrónico que comanda los algoritmos de control y medición mientras que una computadora implementa la interfase con el usuario [17, 18].

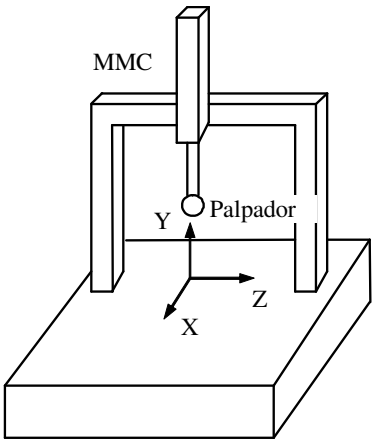

Figura 2.1. Esquema de una MMC.

Con esta idea en mente, en el presente trabajo de tesis se desarrolló un sistema simple que emula una MMC en un solo eje cartesiano pero que fácilmente puede ser extendido a los tres ejes de la figura 2.1. El esquema adoptado en el presente proyecto consiste básicamente de los siguientes componentes:

- Un motor CD de 55 oz-in de torque y 1000rpm/V.
- Un codificador angular incremental de 100X4=400 cpr (cuentas por revolución).
- Un sistema electrónico para el control del motor en lazo cerrado con interfase USB.
- Una computadora personal con software desarrollado específicamente para esta aplicación.

El esquema general se muestra en la figura 2.2.

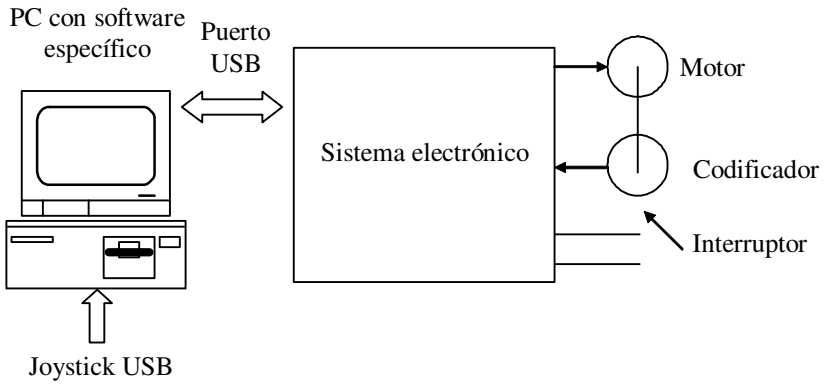

Figura 2.2. Esquema general del sistema para el control de un servomotor.

En el esquema adoptado el sistema electrónico carece completamente de elementos para interactuar con el operador humano.

En este sentido, el software en la PC y joystick implementan la interfase con el operador.

Además del control sobre el motor, el esquema propuesto contempla una interfase con un interruptor que simula el palpador mecánico de contacto, como el que se dispone en las MMC comerciales.

2.1.1. Sistema electrónico

El hardware desarrollado para esta aplicación esta sobre la base de un microcontrolador PIC16C765 con interfase USB. En esta aplicación, el microcontrolador maneja dispositivos periféricos para el control de movimiento en lazo cerrado de un motor CD, un decodificador óptico y entradas digitales para la interfase con palpador de contacto, como se muestra en la figura 2.3 [7].

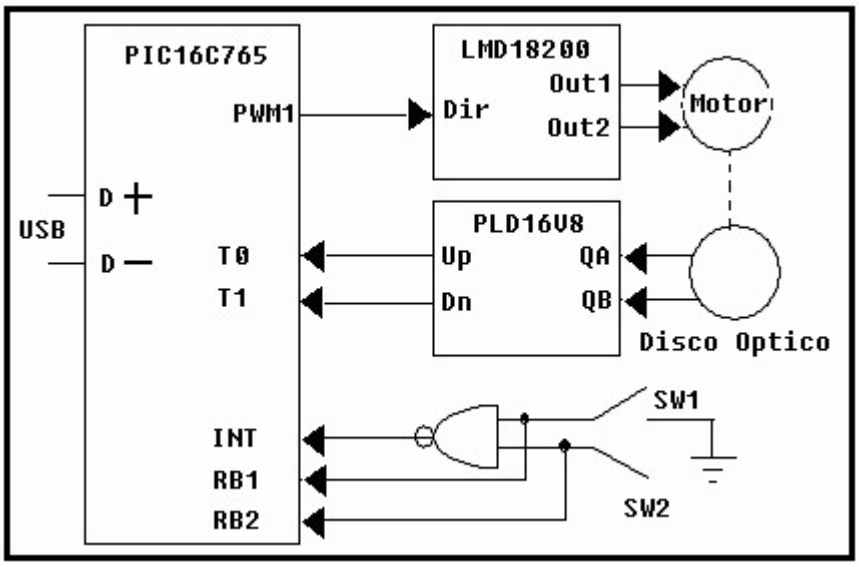

Figura 2.3. Sistema electrónico

El microcontrolador genera una modulación por ancho de pulso (PWM1) para controlar el movimiento del motor. La señal es introducida a una etapa de potencia (Puente H LMD18200) para cumplir con los requerimientos de corriente y voltaje necesarios en el motor. La señal PWM1 es resultado del cálculo para un controlador PID; en esta configuración 50% de ciclo de trabajo equivale a 0% en movimiento en la posición angular; 100% del ciclo de trabajo a 100% de la velocidad angular en sentido positivo y 0% del ciclo de trabajo a 100% en la velocidad angular en sentido negativo.

Los incrementos angulares en el codificador óptico son usados para emular el desplazamiento del decodificador lineal de una MMC real. En este prototipo el codificador óptico es de 400 cuentas por revolución. Las señales en cuadratura QA y QB son procesadas por un

dispositivo lógico programable (PLD) el cual genera dos señales de pulsos: cuentas arriba y cuentas abajo, Up y Dn respectivamente [5]. Estas señales se introducen en dos contadores de 24 bits, T0 y T1 del microcontrolador para guardar la posición angular. La posición actual es el resultado de la diferencia T0-T1.

Debido a que el palpador de contacto es el dispositivo más común de sensado en MMC reales, el esquema propuesto incluye un interruptor de contacto que genera dos señales de disparo SW1 y SW2 cuando se simula una palpación con algún objeto a medir [17, 18]. Estas señales entran a una compuerta lógica NAND para generar una señal de interrupción INT. Cuando se activa la señal INT se entra a la rutina de interrupción la cual discrimina sobre cual señal de disparo ocurrió y por lo tanto se registra el instante de medición. En esta situación, la medición puede llevarse a cabo cuando el palpador viaja por un eje en dos diferentes direcciones (x-positiva o x-negativa). El esquema de palpación y los perfiles de movimiento (posición y velocidad) preferidos se muestran en la figura 2.4.

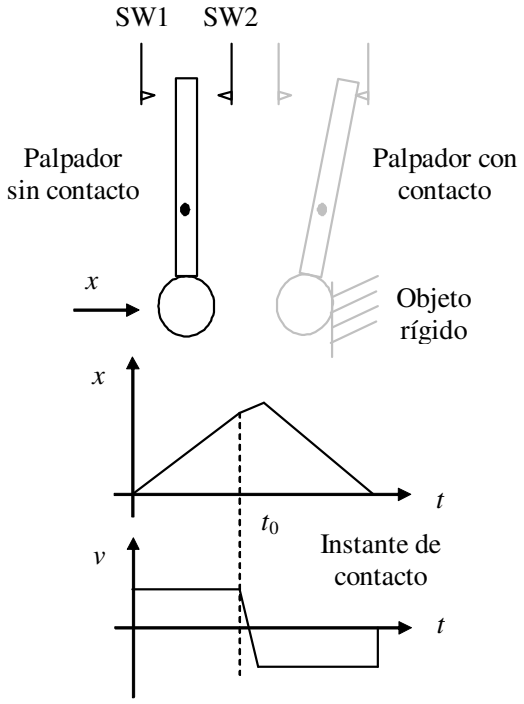

Figura 2.4. Medición con palpador de contacto.

Cuando la esfera del palpador hace contacto con un objeto externo interpuesto en el camino, la fuerza de contacto aumenta gradualmente. La señal de disparo es generada cuando la fuerza de contacto supera un umbral en el interruptor eléctrico del sistema sensor. Esta señal es usada por el sistema electrónico para registrar la posición de contacto exactamente en el tiempo dado. Sin embargo, después del tiempo de

contacto, el palpador continua en movimiento por inercia, pero el sistema electrónico provee de un mecanismo automático para retirar el palpador de la posición de contacto con el cuerpo rígido y poner al sistema listo para otra nueva medición.

En el esquema propuesto, la interfase USB es utilizada para enviar y recibir datos del microcontrolador (PIC16C765). En este sentido, la PC lleva a cabo procesos de alto nivel constituyéndose como la interfase con el usuario. En el software se tienen dos maneras de enviarle datos al microcontrolador. La primera es por medio del teclado, esto es, introduciendo los valores hacia donde se quiera llevar el motor. Para usar esta forma es necesario introducir el valor de la posición requerida. La segunda forma es por medio de una palanca de juego o *joystick*. El joystick genera ya sea 65536 si el movimiento es hacia la derecha, 32768 para la posición central o cero si el movimiento es a la izquierda. Visual C++ tiene implementada las librerías de la *HID* que permiten obtener los valores que generó el joystick [12]. De esta manera se elabora una rutina de desplazamiento del servomotor mediante el propio desplazamiento del joystick.

2.1.2. Interfase de operación

El entorno de desarrollo de VC++ 6.0 ofrece muchas posibilidades al programador, desde crear aplicaciones de diversos formatos y características además de la creación de librerías DLL, iconos, bitmaps, cursores, etcétera [4, 6, 13].

Visual C++ tiene incluida la biblioteca MFC que es un conjunto de clases predefinidas de C++. Estas clases son ahora un estándar de desarrollo para aplicaciones Windows y poseen clases que facilitan la programación en C++.

La interfase de operación para el presente proyecto de tesis es una aplicación sobre la base de *diálogos*. Las ventanas de diálogos son uno de los elementos más utilizados para la interacción con el usuario. Un diálogo es una ventana especial que contiene en su interior ventanas hijas que son controles identificados por un número único, además, cuando un diálogo se despliega, se convierte en una ventana exclusiva pues deja inhabilitada cualquier operación con el resto de la aplicación.

La ventana principal es un grafico sencillo, figura 2.5. En la esquina superior izquierda se tiene la casilla para introducir el *setpoint* o *posición objetivo* a la que se desea posicionar el servomotor. Hacia abajo la casilla que indica la *Posición actual* donde se encuentra el servomotor. Enseguida se encuentra el recuadro donde aparece la *Posición de contacto* en que se origino una interrupción por contacto y

la última casilla indica la *Bandera de contacto* que es cero cuando ocurre la interrupción y en otro caso es uno. En la parte inferior, indicado por el letrero *Velocidad 10 bits* se despliega la velocidad en que se encuentra el sistema cuando se utiliza el joystick. En la parte superior a la derecha la caja *Velocidad* permite establecer una velocidad deseada mediante teclado. Las ultimas tres casillas *k1*, *k2* y *k3* son utilizadas para realizar experimentos de exactitud y precisión al introducir las constantes del controlador PID

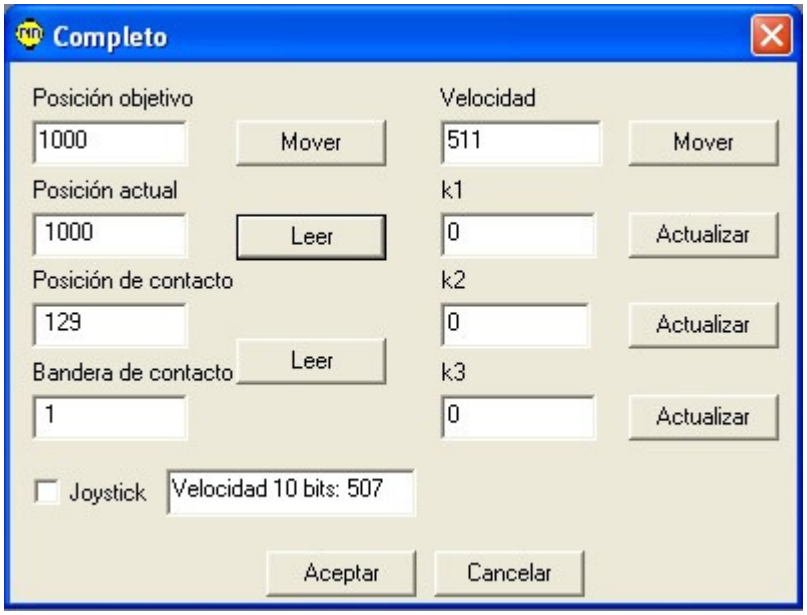

Figura 2.5. Interfase de operación.

El siguiente segmento de pseudo-código muestra la estructura básica del programa de la aplicación.

```
Inicio
Agregar la clase CUSBDEVICE para la comunicación USB
Crear los cajones de diálogo de la  aplicación IDD_COMPLETO_DIALOG
Definir el tipo de variable a utilizar en los cajones de diálogo
Construir el cuerpo de las funciones de cada botón creando objetos de
la clase CUSBDEVICE para la comunicación USB
If Button1
  Almacenar el valor del set point con una resolución de 24 bits
  If conexión USB
     Mandar datos a la tarjeta
  Else
    Desplegar mensaje de error en la conexión
If Button2
  If conexión USB
    Recibir el valor de la posición del motor leyendo el estado
    de los Timers del PIC

  Else
    Desplegar mensaje de error en la conexión
If Button3
  Almacenar el valor de la constante k1 con una resolució
  de 24 bits
```

```
   If conexión USB
     Mandar datos a la tarjeta
   Else
     Desplegar mensaje de error en la conexión
If Button4
   Almacenar el valor de la constante k2 con una resolución
   de 24 bits
   If conexión USB
     Mandar datos a la tarjeta
   Else
     Desplegar mensaje de error en la conexión
If Button5
   Almacenar el valor de la constante k3 con una resolución
   de 24 bits
   If conexión USB
     Mandar datos a la tarjeta
   Else
     Desplegar mensaje de error en la conexión
If Button6
   Almacenar el valor de la velocidad y el sentido de giro
   de la flecha del motor con una resolución de 10 bits
   If conexión USB
     Mandar datos a la tarjeta
   Else
     Desplegar mensaje de error en la conexión
If Button7
   If conexión USB
     Recibir el valor de la posición del motor en caso de existir
     una interrupción externa
   Else
     Desplegar mensaje de error en la conexión
Fin
```

2.2. Sistema electrónico

El sistema electrónico constituye el elemento actuador y transductor de las variables físicas involucradas en el servomotor. Su elemento central es el microcontrolador PIC16C765 gobernando una serie de dispositivos electrónicos que adecuan las señales a los requerimientos eléctricos del prototipo. El anexo 5.1 muestra el diagrama electrónico completo. A continuación, una descripción detallada del sistema electrónico desarrollado específicamente para éste proyecto.

2.2.1. Descripción del hardware

La figura 2.6 muestra la parte central del sistema electrónico, consta de un microcontrolador PIC17C765, el cual tiene soporte para la interfase USB. Opera con un oscilador de 6MHz. El conector USB es de cuatro pines (V, GND, D-, D+) que proporcionan la alimentación de 5 V que requiere la tarjeta. Las entradas D- y D+ son para entrada y salida de datos.

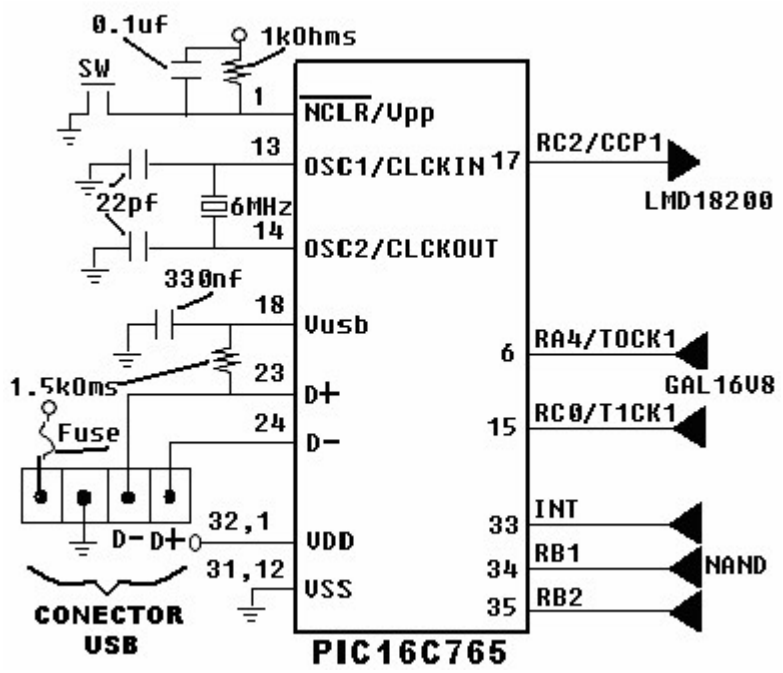

Figura 2.6. Arquitectura del microcontrolador PIC16C765.

La señal PWM es de 10 bits de resolución y su salida es el pin17 (CCP1) hacia el chip LMD18200. El LMD18200 es un puente H de 3 A para operar hasta con 55 V, figura 2.7. Tiene aplicaciones tales como en mecanismos de posición y velocidad, motores a pasos, plotters, impresoras, etcétera. Es necesario usar una fuente de 12 V para aumentar la potencia de la señal PWM y así controlar el motor.

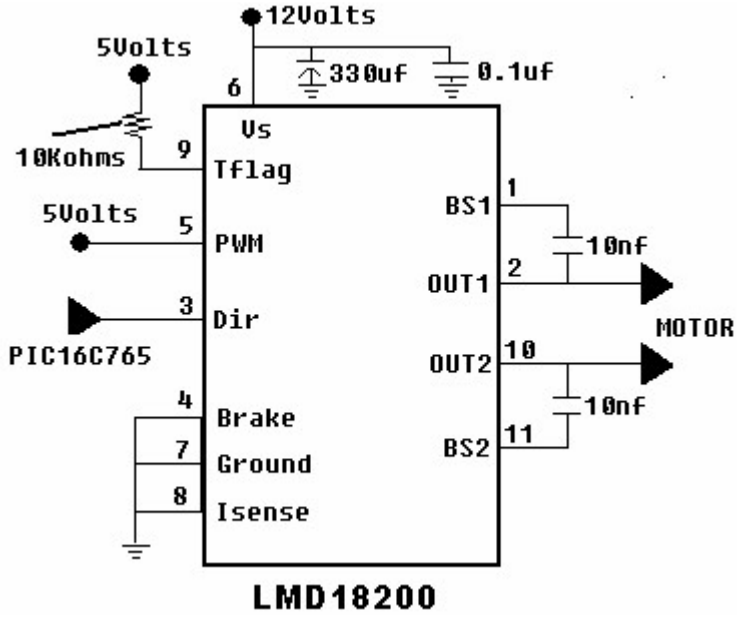

Figura 2.7. Arquitectura del puente H LMD18200.

Una vez que se ha desplazado el motor, el sensor HEDS 9200 solidario a la estructura del motor envía las señales en cuadratura QA y QB al circuito GAL16V8. La GAL16V8 es un arreglo lógico programable de 64x32 con tecnología ECMOS reprogramable y reconfigurable con borrado de alta velocidad. Un programa implementando en WINCUPL en este dispositivo realiza la decodificación de los pulsos enviados por el sensor y genera señales para su conteo "CountUp-CountDown"[2]. El programa implementa el circuito decodificador de la figura 2.8. El código en WINCUPL de la implementación se muestra en el anexo 5.2.

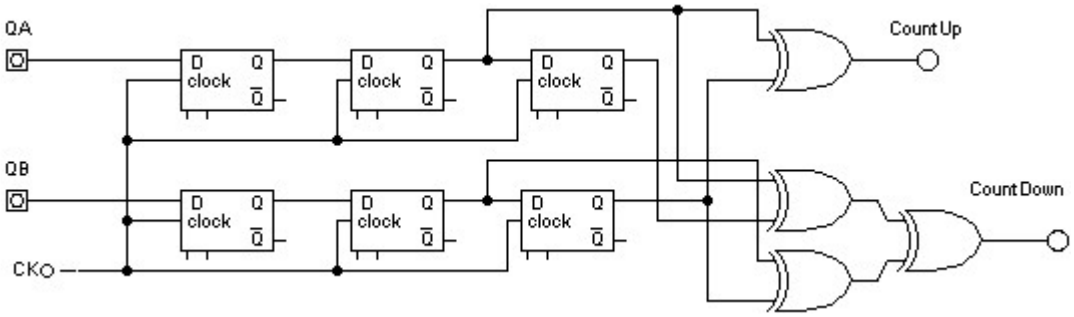

Figura 2.8. Circuito decodificador de señales en cuadratura.

Una vez realizado el descifrado, se envían los datos de vuelta al microprocesador a sus pines 6 y 15 para impulsar los contadores de 24 bits y decodificar la posición angular, figura 2.9.

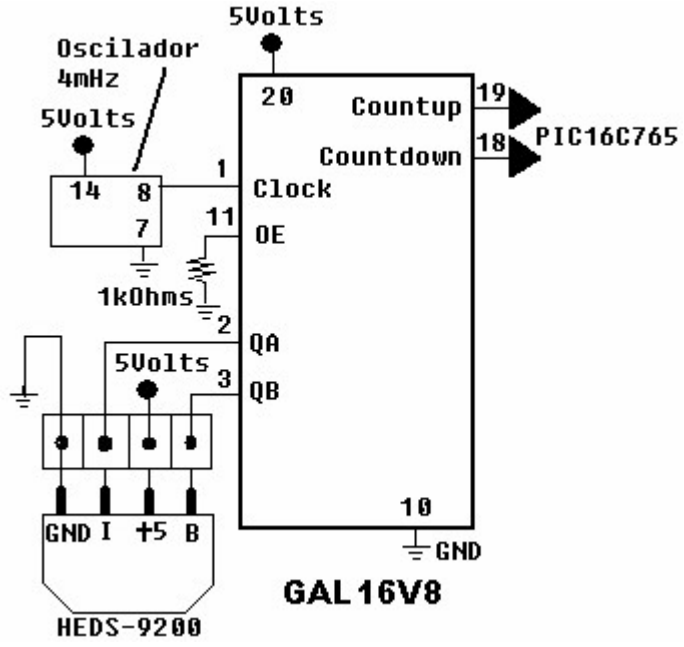

Figura 2.9. Arquitectura del arreglo GAL16V8.

Al desplazar los ejes coordenados de una MMC y al tocar el palpador con un objeto bajo medición, el sistema debe realizar las tres

siguientes tareas: reconocer el sentido de palpación, validar la lectura en el instante preciso de contacto y retirar el palpador en el sentido opuesto a la palpación. La solución que se plantea en el software de control del microcontrolador es el reconocimiento del accionar de dos señales de un interruptor SW1 y SW2 para posteriormente realizar un retroceso rápido del motor a una posición anterior y contraria a la dirección de movimiento implícita a SW1 o SW2. Para emular este suceso, mediante un sencillo dispositivo que consta de una NAND (74LS00) se provoca una interrupción externa en el microcontrolador y su servicio de atención que ejecuta las tres tareas anteriores. El diagrama se muestra en la figura 2.10.

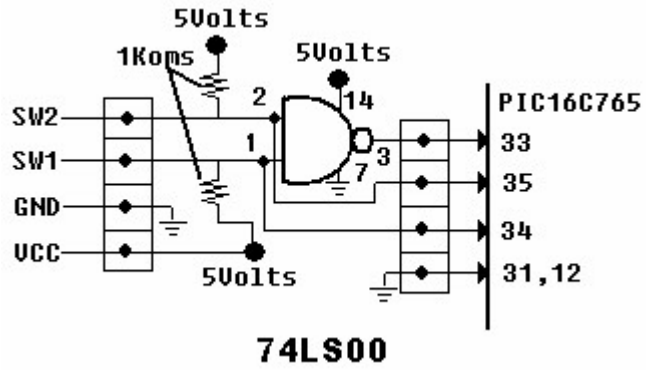

Figura 2.10. Interfase con palpador.

2.2.2. Algoritmo de control

El software en el sistema permite el control del movimiento del motor y los algoritmos de medición. El código fuente esta escrito en lenguaje ensamblador para el PIC16C765 y se omite en el presente trabajo debido a su gran longitud [8]. En lugar del listado completo del código fuente, la presente sección describe en pseudo-código las porciones más significativas del programa. El software hace uso extensivo de módulos periféricos integrados al microcontrolador en un esquema basado fuertemente en interrupciones.

Estructura del programa

El programa es descrito con el siguiente pseudo-código:

```
Establecer el tipo de PIC utilizado
Establecer las variables utilizadas y el tamaño que ocupan en memoria
Declaración del módulo que provee la  interfase USB
Establecer el proceso de la rutina del servicio de interrupción
Establecer el cuerpo principal del programa
Declarar valores de variables
Establecer Timers 0, 1 y 2 como contadores
Establecer el módulo PWM al 50%
Habilitar interrupciones

Iteración
IF se recibe dato USB
```

```
      Llama a subrutina Procesar datos
ELSE
      regresar a Iteración
Rutinas de interrupciones
```

Las variables que se utilizan en el programa son de 48, 24 y 8 bits; cada localidad de memoria tiene una capacidad de 8 bits. La aritmética considerada es de 24 bits debido a que una MMC real tiene aproximadamente las siguientes características:

Resolución = 2μm = 0.002mm.
Alcance = 500 mm.

Lo cual resulta en contadores de al menos $500/0.002=250{,}000\approx2^{18}$, es decir por lo menos 18 bits para cubrir el alcance de medición con la resolución especificada.

La interfase para la aplicación USB esta almacenada en las siguientes funciones:

- InitUSB. Esta función habilita el periférico USB y la interrupción USB por reset y además prepara al dispositivo para el proceso de enumeración.
- PutUSB. Esta función envía datos hacia la computadora.
- GetUSB. Esta función recibe datos desde la computadora.
- ServiceUSBInt. Esta función maneja todas las interrupciones generadas por el periférico USB.

La figura 2.11 muestra el esquema del software de interfase USB.

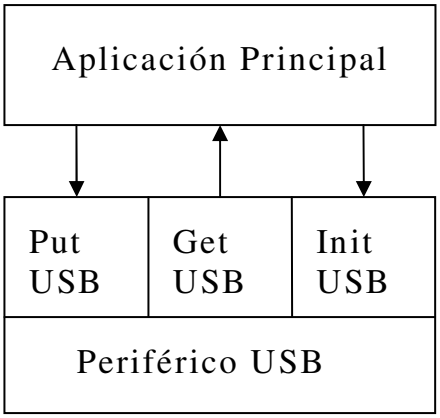

Figura 2.11. Interfase USB.

La función CheckSleep permite al dispositivo ser puesto en la modalidad Sleep, es decir, poner al PIC en un modo de espera de bajo poder de energía.

Interrupciones

Los microcontroladores PIC pueden tener muchas fuentes de interrupción. Estas fuentes generalmente incluyen una interrupción por cada módulo periférico, sin embargo algunos módulos pueden generar múltiples interrupciones. Las utilizadas en este programa son:

- Interrupción por desbordamiento del Timer 0.
- Interrupción por desbordamiento del Timer 1.
- Interrupción por desbordamiento del Timer 2.
- Interrupción externa a través del pin INT.

Típicamente, los usuarios pueden desear salvar la información contenida en los registros internos del PIC durante una interrupción, como por ejemplo los registros W y STATUS. La acción de salvar información es conocida como “PUSHing”, mientras que la acción de restaurar información antes de dejar la interrupción es conocida como “POPing”. Estas acciones no son instrucciones del lenguaje ensamblador sino acciones conceptuales. Estas acciones son implementadas por una secuencia de instrucciones dentro del programa.

Dentro del cuerpo principal del programa se establecen valores iniciales de algunas variables que se utilizan para el controlador PID, las distintas interrupciones y para los Timers utilizados como contadores.

Timer0

En el programa, el Timer0 tiene un pre-escalador asignado de 1:1, fuente de reloj externa (la salida del decodificador) y se incrementa en el cambio de estado bajo a alto en el nivel de voltaje, figura 2.12. El Timer0 es usado como contador de las cuentas ascendentes del codificador colocado en el motor.

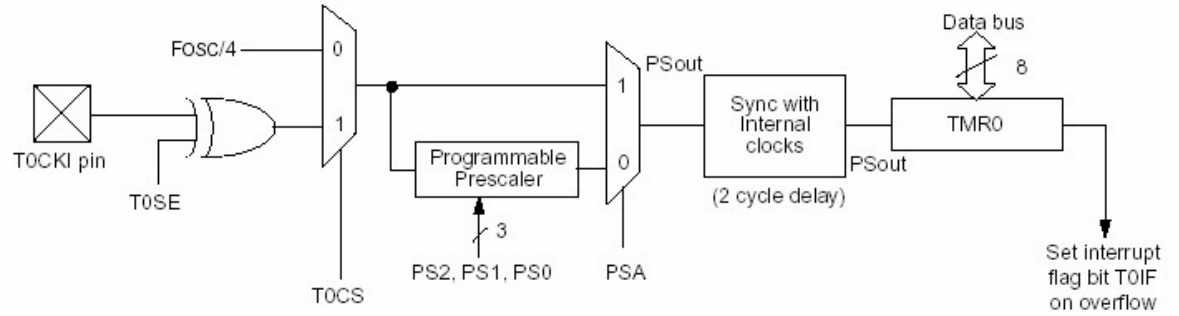

Figura 2.12. Timer0.

Timer1

El módulo del Timer1 es un contador/reloj de 16 bits consistente en dos registros de 8 bits, los cuales pueden ser leídos y sobre-escritos. El par de registros del Timer1 se incrementa de 0000h a FFFFh y después empieza de nuevo en ceros, si la interrupción por desbordamiento es habilitada, es en este momento que se produce. En el modo de reloj, el Timer1 se incrementa con cada ciclo que dure el completarse una instrucción. En el modo de contador, se incrementa con cada flanco de subida de la señal de entrada de reloj externa en el pin T1CKL. También cuenta con un pre-escalador programable de cuatro valores (1:1, 1:2, 1:4 y 1:8), figura 2.13.

En el programa el Timer1 es usado como contador de las cuentas descendentes del decodificador conectado al motor.

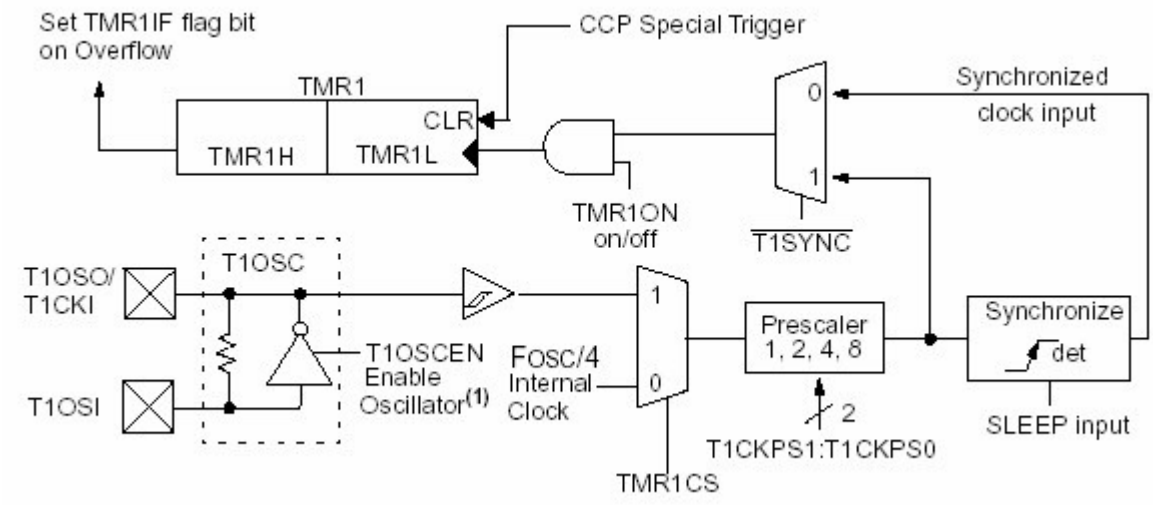

Figura 2.13. Timer1.

Timer2

El Timer2 es un modulo de 8 bits con un pre-escalador, un post-escalador y un registro para establecer periodos (PR2). Usando el pre-escalador y el post-escalador a su máxima capacidad el tiempo de desbordamiento es el mismo que el de un timer de 16 bits. Este Timer puede ser la base de tiempo para el PWM. Cuando se encuentra en el modo de la modulación de ancho de pulso, el pin CCP produce una resolución del PWM de 10 bits en la salida.

Los siguientes son los pasos necesarios para establecer el modo PWM:

- Establecer el periodo del PWM escribiéndolo en el registro PR2.
- Establecer el ciclo de trabajo del PWM.

- Establecer el valor del pre-escalador y habilitar el Timer2.
- Configurar al módulo para la operación PWM.

Un esquema básico del módulo del Timer2 puede apreciarse en la figura 2.14. El Timer2 es utilizado en la aplicación para actualizar el valor del PWM y del PID.

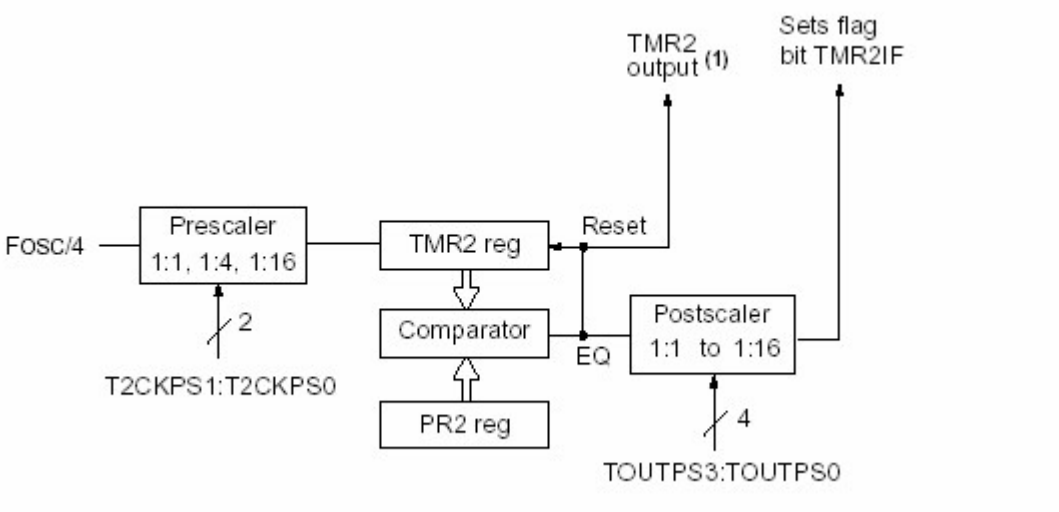

Figura 2.14. Timer2.

De especial interés para el presente proyecto es la actualización del PID en tiempo real. La actualización se lleva a cabo mediante la actuación del PID descrita en la ecuación 1.17. Despejando de esta ecuación el voltaje aplicado a la planta

$$M(z)=(K_p+K_i+K_d)E(z)-(K_p+2K_d)E(z)z^{-1}+K_dE(z)z^{-2}+M(z)z^{-1}$$

Considerando que z^{-n} representa un retraso de n unidades de tiempo y aplicando la transformada inversa a la ecuación anterior

$$m(k)= (K_p+K_i+K_d)e(k)-(K_p+2K_d)e(k-1)+K_de(k-2)+m(k-1) \qquad (2.1)$$

Que resulta ser la ecuación base para la actualización del PID. El anexo 5.3 muestra la simulación en MatLab para la implementación de la ecuación (2.1). Tomando en cuenta que las variables e y m son de 24 bits, la ecuación (2.1) resulta en una serie de tres multiplicaciones y tres sumas de 48 bits.

Dentro de la iteración el PIC espera a que la computadora envíe datos por el puerto USB, una vez que esto ocurre se llama a una subrutina para el procesamiento del dato USB recibido, aquí es donde se lleva a cabo el análisis del tipo de dato enviado desde la computadora para saber que acción debe llevar a cabo el microcontrolador. El siguiente segmento de pseudo-código describe el proceso.

```
Subrutina Procesar dato USB

      IF dato USB = posición
            Set point = dato USB
            Bandera PID = Verdadera
      Else IF dato USB = leer posición
            Dato USB =Ttimer0 -Ttimer1
      Else IF dato USB = constantes PID
            Constants PID = dato USB
      Else IF dato USB = velocidad
            Bandera PID = falso
            PWM = dato USB
      Else IF dato USB = leer posición Contacto
            Dato USB = posición Contacto
            IF bandera Contacto = verdadero
                  Bandera Contacto = falso
      Regresar a Subrutina
```

Rutinas de interrupción

A continuación se presenta un esquema básico de la implementación de las 5 subrutinas de interrupción ocupadas dentro del programa.

```
Timer 0
  Incrementar el Ttimer 0 utilizando 24 bits en las cuentas

Timer 1
  Incrementar el Ttimer 1 utilizando 24 bits en las cuentas

Timer 2

  If bandera PID = verdadera
    Leer el estado del Timer0 y el Timer1
    Determinar la posición del motor: posición = Timer0 -
     Timer1
    error0 = set point - posición
    mul1 = k1 * error
    mul2 =k2 * error1
    mul3 =k3 * error2
    eme0 = mul1 + mul2 + mul3 + eme1
    If eme0 > 1023
      eme0 = 1023
    If eme0 ≤ 0
          me0 = 0
    PWM = eme0
    error2 = error1
    error1 = error0
    eme1 = eme0
  else regresar al programa principal

Interrupción Externa
  Posición de contacto = Timer0 - Timer1
  Bandera PID = falso
  Bandera de contacto = verdadero
  PWM = 50%
  If RB1 = 0
    Set point = Timer0 - Timer1 - Offset
  Else If RB2 = 0
    Set point = Timer0 - Timer1 + Offset
  Bandera PID = verdadero
```

2.3. Software de operación

El sistema electrónico y por lo tanto su actuación sobre el motor es controlada mediante el software llamado “completo”. El programa fue desarrollado específicamente para este proyecto utilizando Visual C++ 6.0 y en particular programando la clase HID que soporta la interfase con los dispositivos USB más comunes. El software desarrollado se puede concebir como una “terminal USB” de donde se envían y reciben comandos USB para comandar o encuestar el estado del sistema electrónico.

2.3.1. Descripción del ámbito usuario

El software que implementa la interfase para el control del servomotor a través de la interfase USB es el mostrado en la figura 2.15. El usuario tiene las siguientes posibilidades en la interfase:

- Botón 1 *Mover*. Especificar la posición objetivo en cuentas.
- Botón 2 *Leer*. Leer la posición actual en tiempo real del sistema.
- Botón 3 *Leer*. Leer la posición de contacto y bandera de contacto.
- Botón 4 *Mover*. Establecer la velocidad y el sentido de giro por teclado.
- Botones 5, 6 y 7 *Actualizar*. Especificar el valor de las tres constantes del controlador PID digital.
- Caja *Joystick*. Establecer la velocidad del sistema mediante una palanca de mandos o Joystick.

Los requisitos mínimos para operar el sistema son:

- Computadora Pentium III, Puerto USB, 256MB RAM, 6MB DD libres, SVGA, Windows 2000/XP, Visual C++ 6.
- Sistema electrónico incluyendo motor CD y codificador.
- Joystick USB.

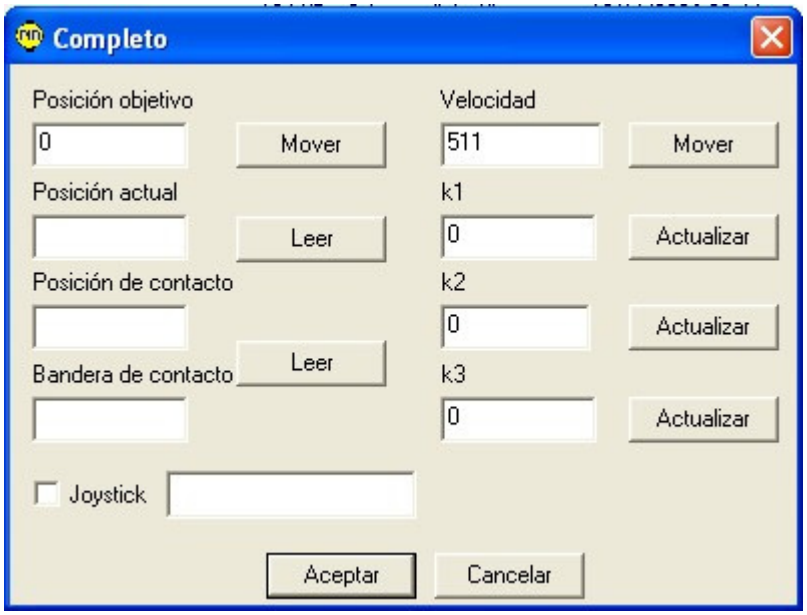

Figura 2.15 Interfase de operación.

2.3.2. Descripción en el ámbito del programador

La presente sección describe la programación utilizada en la implementación en tiempo real de la aplicación "completo".

El programa se desarrolló usando la MFC como una aplicación basada en dialogo, sin documento ni vista.

Debido a la gran extensión del código fuente, en la presente sección sólo se describen las porciones más significativas del programa y se omite el listado completo.

La figura 2.16 muestra las clases en la implementación del programa para el controlador PID.

Figura 2.16. Clases del proyecto.

Clase CAboutDlg

La figura 2.17 muestra la clase CAboutDlg para la implementación de la caja de diálogo "Acerca de ".

Figura 2.17. Clase CAboutDlg.

CAboutDlg(); Constructor por omisión.

Virtual void DoDataExchange(CDataExchange* pDX); Soporte para el intercambio de datos con otras clases DDX/DDV.

Clase CCompletoApp

La figura 2.18 muestra la clase CCompletoApp para la implementación de la instancia de aplicación.

Figura 2.18. Clase CCompletoApp.

CCompletoApp(); Constructor por omisión.

Virtual BOOL InitInstance(); Habilita a la instancia para su ejecución como caja de diálogo.

Clase CCompletoDlg

La figura 2.19 muestra la clase CCompletoDlg que contiene el código principal de la aplicación.

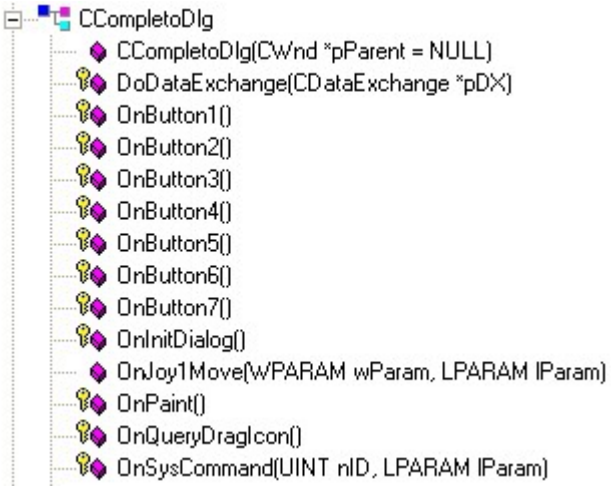

Figura 2.19. Clase CCompletoDlg.

CCompletoDlg(CWnd* pParent = NULL); Constructor por omisión.

virtual void DoDataExchange(CDataExchange* pDX); Soporte para el intercambio de información entre clases DDx/DDV.

virtual void OnButton1(); Inicia la salida de datos hacia la tarjeta, para modificar el valor del set point, a través del puerto USB.

virtual void OnButton2(); Inicia la entrada de datos desde la tarjeta, para leer la posición del motor, a través del puerto USB.

virtual void OnButton3(); Inicia la salida de datos hacia la tarjeta, para modificar el valor de la constante k1, a través del puerto USB.

virtual void OnButton4(); Inicia la salida de datos hacia la tarjeta, para modificar el valor de la constante k2, a través del puerto USB.

virtual void OnButton5(); Inicia la salida de datos hacia la tarjeta, para modificar el valor de la constante k3, a través del puerto USB.

virtual void OnButton6(); Inicia la salida de datos hacia la tarjeta, para modificar la velocidad y el sentido de giro de la flecha del motor, a través del puerto USB.

virtual void OnButton7(); Inicia la entrada de datos desde la tarjeta, para leer la posición del motor cuando ocurre una interrupción externa, a través del puerto USB.

virtual BOOL OnInitDialog(); carga la tarjeta en memoria, inicia un timer auxiliar para el despliegue de lecturas e inicia la tarjeta.

LRESULT OnJoy1Move(WPARAM wParam, LPARAM lParam); Obtiene el desplazamiento en el joystick para convertirlo a un valor de 10 bits y enviarlo al sistema electrónico para establecer la velocidad.

afx_msg void OnPaint(); Soporte para minimizar la aplicación.

afx_msg HCURSOR OnQueryDragIcon();Soporte para minimizar la aplicación.

Afx_msg void OnSysCommand(Unit nID, LPARAM lParam); Procesa el mensaje para desplegar la caja de diálogo “Acerca de “.

Clase CUSBDevice

La figura 2.20 muestra la clase CUSBDevice que contiene el código para la comunicación del microprocesador PIC16C765 con la PC.

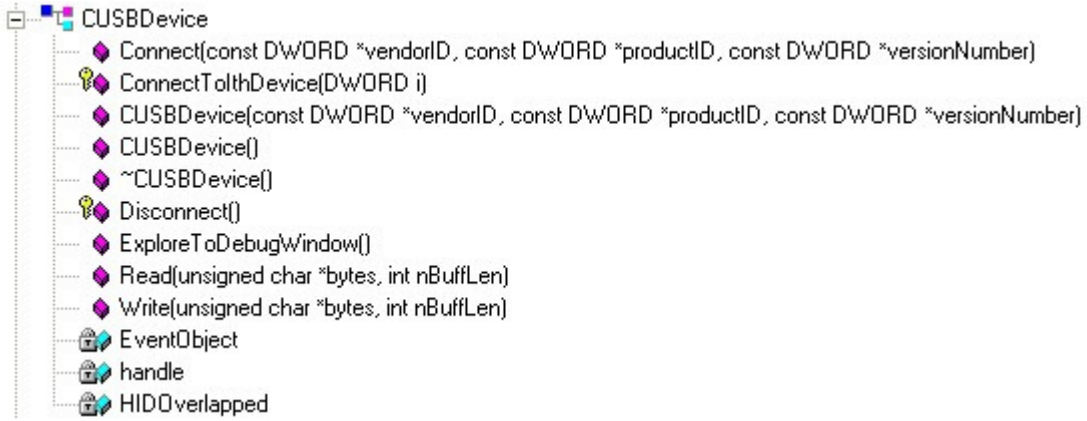

Figura 2.20. Clase CUSBDevice.

bool CUSBDevice::Connect(const DWORD *vendorID, const DWORD *productID, const DWORD *versionNumber); inicia la comunicacion de la PC con el PIC16C765 recibiendo de este los parámetros como son vendor, producto y versión.

HANDLE CUSBDevice::ConnectToIthDevice (DWORD deviceIndex); se conecta con el i-esimo dispositivo HID USB conectado a la computadora.

CUSBDevice::CUSBDevice(const DWORD *vendorID, const DWORD *productID, const DWORD *versionNumber); Se conecta al HID de USB por la combinación de la id del distribuidor, id del producto y la id del numero de versión.

CUSBDevice::CUSBDevice(); Se conecta al HID de USB descrito por la combinación del la id del distribuidor e id del producto. Si el atributo es nulo, conectará el primer dispostivo que satisfaga los requerimientos.

CUSBDevice::~CUSBDevice(); desconecta en caso de error.

void CUSBDevice::Disconnect(); desconecta el dispositivo de la PC.

void CUSBDevice::ExploreToDebugWindow(); Solo enumera el dispositivo y despliega vid, pid en la ventana del depurador.

unsigned long CUSBDevice::Read(unsigned char *bytes, int nBuffLen); lee los bytes alojados en el buffer.

unsigned long CUSBDevice::Write(unsigned char *bytes, int nBuffLen); escribe en el buffer.

Capítulo 3

Resultados y conclusiones

El principal resultado es la implementación en tiempo real de un sistema de control para servomotor ejecutando un algoritmo PID sobre un sistema hardware. Adicionalmente se adquirió experiencia en controladores de posición así como en el manejo de herramientas de programación que pueden ser incluidos en desarrollos recientes del Laboratorio de Metrología del CCADET UNAM.

3.1. Resultados

La figura 3.1 muestra el hardware para el sistema de control digital con operación en tiempo real.

La figura 3.2 muestra el software de operación desarrollado. Es importante destacar la exactitud en el posicionamiento, en el caso de la figura 3.2 (-1000)-(-998) = 2.

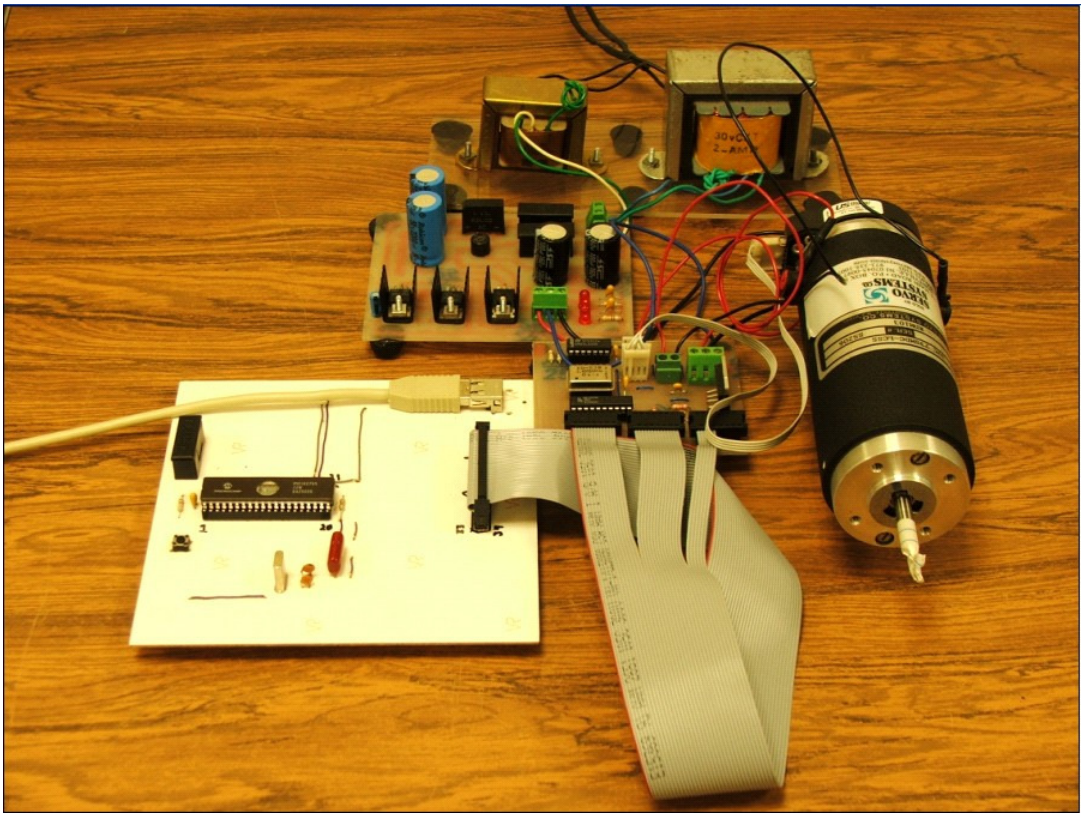

Figura 3.1. Implementación para el sistema de control.

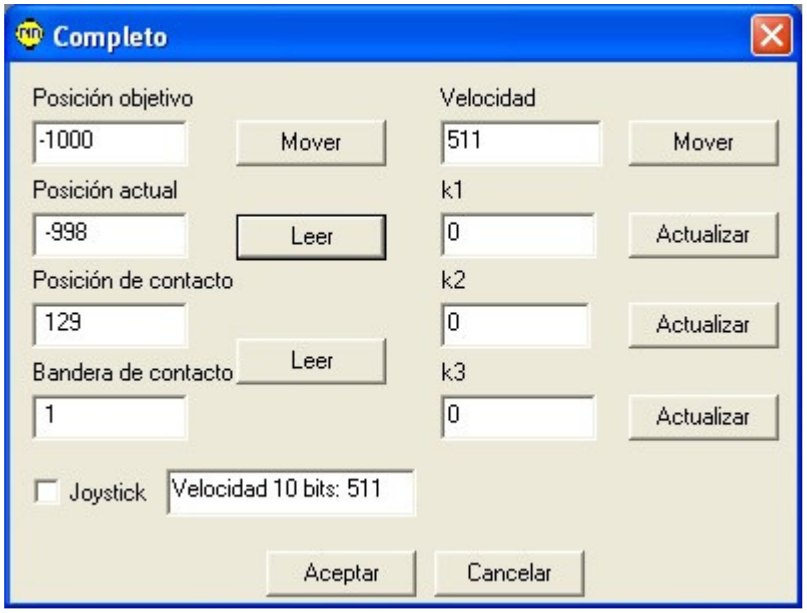

Figura 3.2. Software para el sistema de control.

El desempeño del sistema desarrollado es probado por medio de dos experimentos. El primero esta encaminado a evaluar la exactitud en el posicionamiento. El segundo es usado para evaluar la repetibilidad en el posicionamiento.

3.1.1. Exactitud en el posicionamiento

En el primer experimento, la exactitud de la posición es verificada cuando al sistema se le indica ir a una posición específica. El rango de la posición es de -8 388 608 cuentas a +8 388 607 cuentas. Este rango es resultado de la aritmética de 24 bits usada en el algoritmo del controlador PID.

El experimento es repetido 100 veces. Después de cada nuevo resultado, el sistema es regresado a una posición arbitraria. El resultado indica una máxima desviación de ± 2 cuentas para desplazamientos largos (arriba de 400 cuentas) y una máxima desviación de ± 3 cuentas para desplazamientos unitarios.

3.1.2. Repetibilidad en el posicionamiento

El siguiente experimento es desarrollado para probar la repetibilidad. Primero, al sistema se le especifica una velocidad determinada. Eventualmente, la interrupción externa ocurre simulando un contacto físico entre un dispositivo de prueba y un objeto externo. El experimento es repetido 100 veces con el objeto externo en la misma posición fija para diferentes velocidades. La meta es obtener la misma posición de contacto para cada resultado. La máxima desviación para una posición de contacto se de ± 4cuentas, para un rango de velocidad comprendido entre un ± 25%.

3.1.3. Discusión

Muchas mejoras pueden ser hechas en el sistema propuesto. Primero, un perfil de velocidad no esta implementado debido a que las condiciones reales de operación son desconocidas. La evaluación del desempeño del sistema como una parte de una real Máquina de Coordenadas proveerá las especificaciones para los perfiles de velocidad o aceleración. Segundo, un mejor algoritmo de control de posición puede ser usado. Sin embargo, el algoritmo PID es muy usado en la industria y ha sido probada ampliamente su utilidad y confiabilidad.

3.2. Conclusiones

En este trabajo, un sistema de control para servomotores para Máquinas de Coordenadas ha sido presentado. El sistema fue desarrollado usando herramientas básicas e incluye características adicionales en relación a sistemas comerciales. La primera de estas características adicionales es el algoritmo de la lectura de la posición desarrollado específicamente para máquinas con dispositivos de

prueba. Otra característica es la aritmética de 24 bits del controlador. En este sentido, la aritmética extendida permite trabajar con largas distancias y desplazamientos con alta resolución.

Se desarrollo e implementó un algoritmo para el control de posición con acciones Proporcionales, Derivativas e Integrales operando sobre sistemas en tiempo real. Las siguientes son algunas características del sistema para el control de posición.

- Algoritmo con ejecución en tiempo real. Lo que demuestra su inclusión confiable en sistemas ya desarrollados o por desarrollar.
- Interfase amigable y fácil de utilizar. Una vez analizado el sistema, como se describió en el capítulo 2, su operación es sencilla y no requiere de procedimientos complicados de operación.
- Algoritmo implementado completamente en software. Lo que proporciona sintonizaciones y puesta a punto al variar parámetros software sin la necesidad de alterar el hardware. En forma adicional, su implementación en software nos habilita para desarrollar rutinas preprogramadas de posicionamiento, similares a las máquinas de control numérico.
- Infraestructura simple y barata. Lo que nos garantiza una mínima modificación sobre los sistemas ya desarrollados en el momento de incluir el controlador descrito.
- Plataforma Windows 2000/XP. Lo que nos permite desarrollar instrumentación con interfases fáciles de operar y cooperación de tareas entre diversas aplicaciones Windows.

El método de comunicaciones usado en el sistema desarrollado ha probado su utilidad debido al hecho de que actualmente las computadoras están reemplazando los puertos RS232 y LTP con los puertos USB.

3.3. Trabajo a futuro

El trabajo a futuro puede ser dirigido en dos direcciones. Primero, la implementación en una real máquina de coordenadas con las mejoras y modificaciones necesarias a los algoritmos de control y medición. Finalmente, otra posible línea de desarrollo a futuro es el avanzar en el desarrollo de controladores digitales y mejorar las resoluciones y exactitudes en el posicionamiento.

Anexos

El presente capítulo contiene diversos recursos utilizados en forma adicional para el desarrollo de este proyecto de tesis.

A.1. Diagramas electrónicos

La presente sección contiene información útil para analizar o reproducir el sistema electrónico desarrollado.

A.1.1. Dibujos esquemáticos

Los dibujos esquemáticos del sistema electrónico se subdividieron en los siguientes tres apartados:

- Microcontrolador.
- Interfase con motor y codificador de posición.
- Fuentes de voltaje.

Los cuales se detallan a continuación.

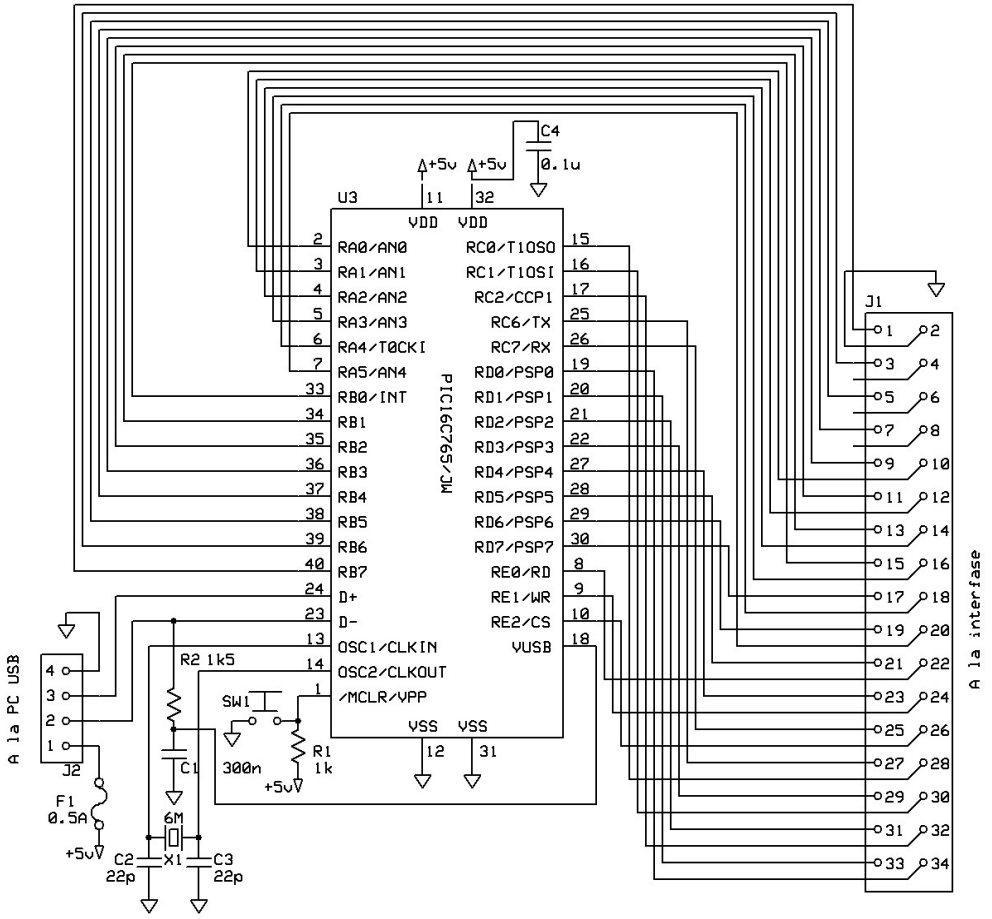

Figura A.1. Circuito esquemático (microcontrolador).

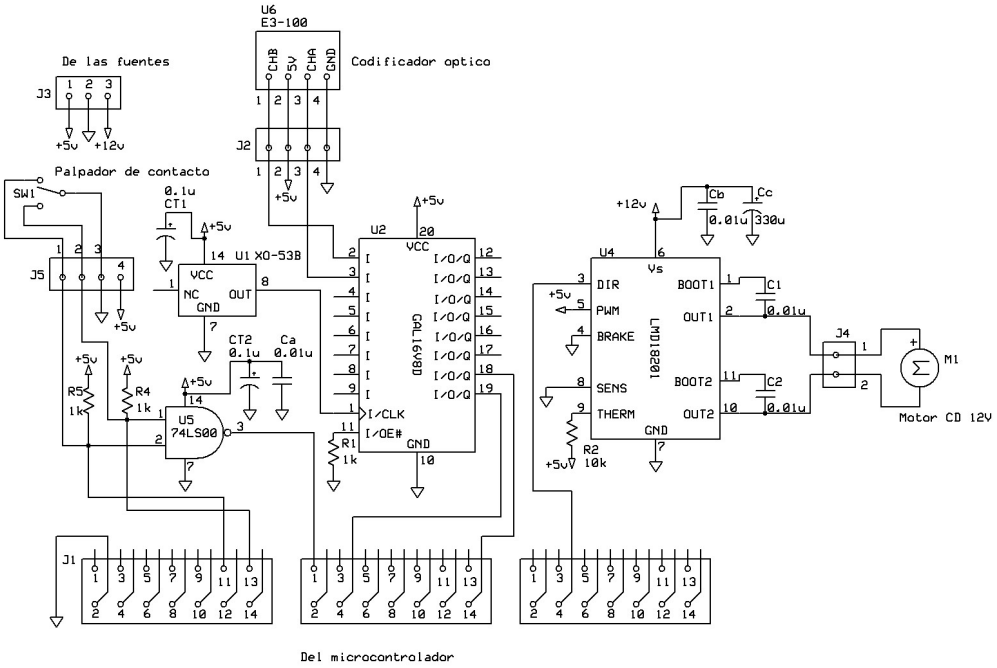

Figura A.2. Circuito esquemático (interfase).

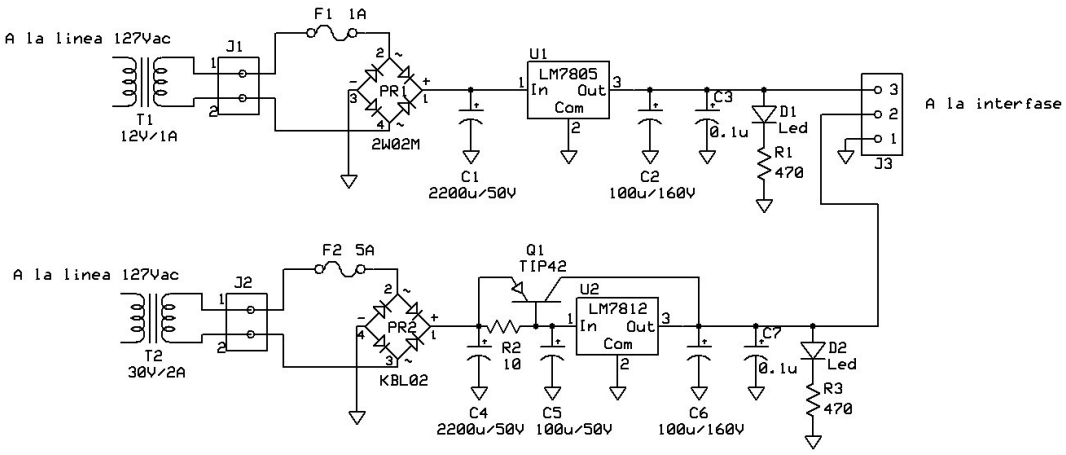

Figura A.3. Circuito esquemático (fuentes).

A.1.2. Circuitos impresos

De igual forma que en la sección anterior, se construyeron circuitos impresos para cada subdivisión del sistema electrónico.

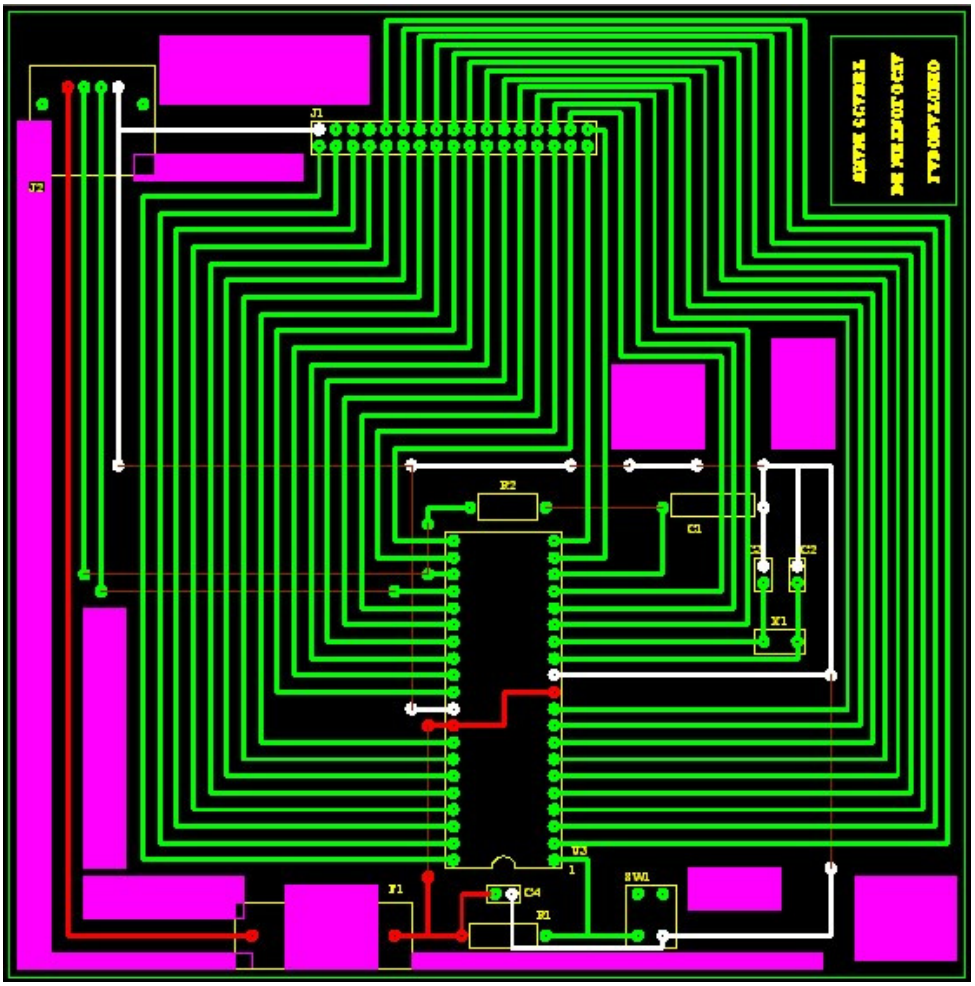

Figura A.4. Circuito impreso (microcontrolador).

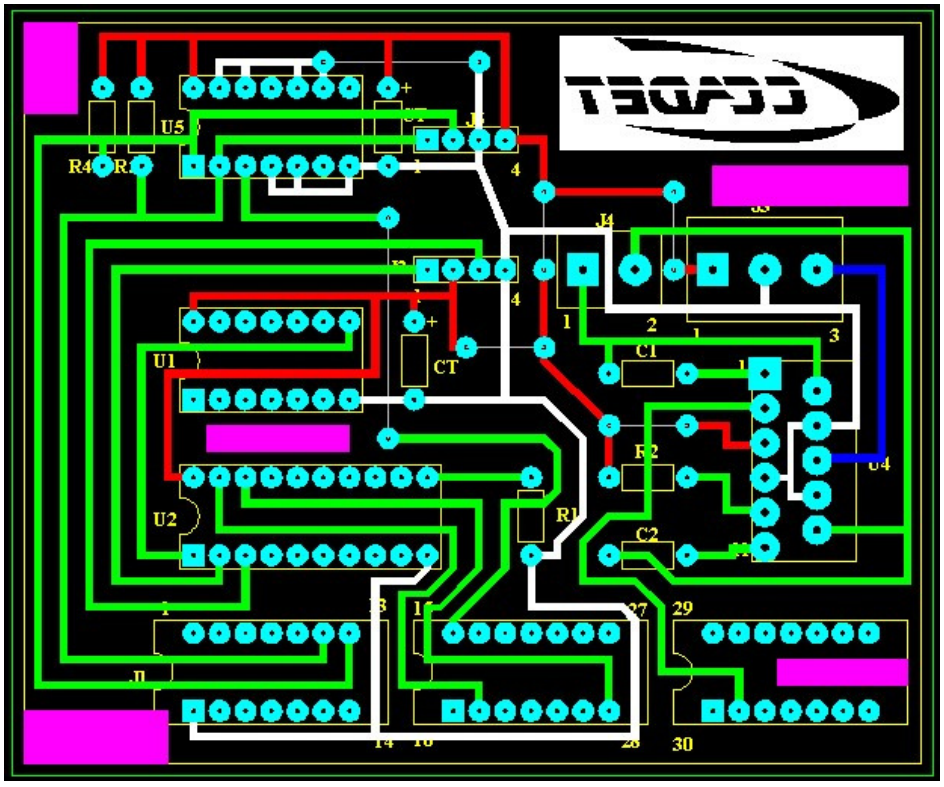

Figura A.5. Circuito impreso (interfase).

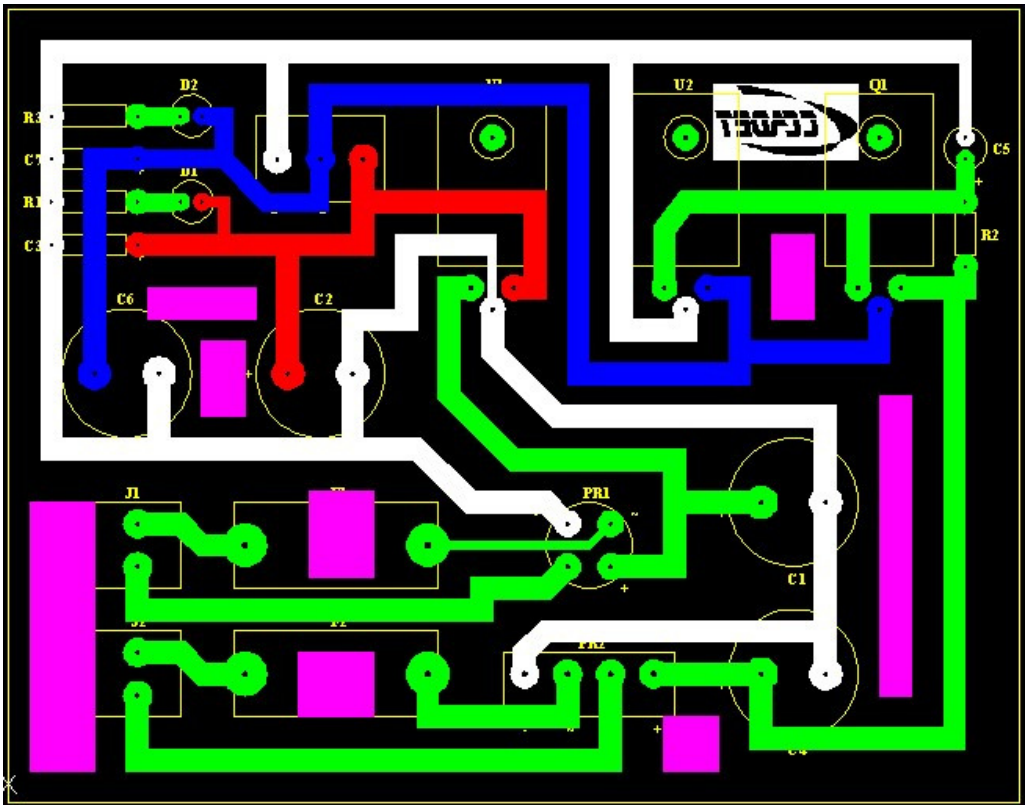

Figura A.6. Circuito impreso (fuentes).

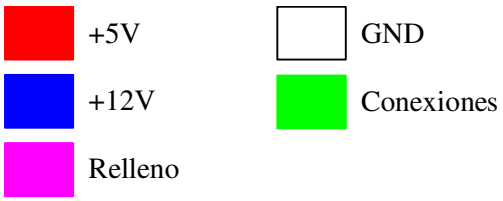

Figura A.7. Circuito impreso (definiciones).

A.1.3. Lista de material

Microcontrolador

Id	Descripción
R1	Resistencia carbón 5% 1KΩ ½ W
R2	Resistencia carbón 5% 1.5kΩ 1/2 W
C1	Capacitor cerámico 300nf
C2	Capacitor cerámico 22pf
C3	Capacitor cerámico 22pf
C4	Capacitor tantalio 0.1µf
X1	Cristal 6MHz
F1	Fusible europeo 1A
F1	Portafusible europeo para circuito impreso
SW1	Push boton normalmente abierto
J1	Header 34 posiciones hilera doble
J1	Conector para cable plano 34 posiciones
J2	Conector USB para circuito impreso
J2	Cable USB
U3	Circuito integrado PIC16C765/JW
U3	Base para circuito integrado 40 posiciones

Tabla A.1. Material para la tarjeta del microcontrolador.

Interfase

Id	Descripción
R1	Resistencia carbón 5% 1KΩ ½ W
R2	Resistencia carbón 5% 10kΩ 1/2 W
R4	Resistencia carbón 5% 1KΩ ½ W
R5	Resistencia carbón 5% 1.5kΩ 1/2 W
C1	Capacitor cerámico multicapa 0.01µf
C2	Capacitor cerámico multicapa 0.01µf
CT1	Capacitor tantalio 0.1µf
CT2	Capacitor tantalio 0.1µf
Ca	Capacitor cerámico multicapa 0.01µf
Cb	Capacitor cerámico multicapa 0.01µf
Cc	Capacitor electrolìtico 330µf
U1	Oscilador 2MHz XO-53B

U1	Base para circuito integrado 14 posiciones
U2	PLD GAL16V8D
U2	Base para circuito integrado 20 posiciones
U4	Puente H LMD18201
U5	Compuerta NAND 74LS00
U5	Base para circuito integrado 14 posiciones
J1	3 bases para circuito integrado 14 posiciones
J1	3 Conectores DIP para cable plano 14 posiciones
J2	Header 4 posiciones
J2	Caja 4 posiciones
J3	Conector para circuito impreso 3 posiciones
J4	Conector para circuito impreso 2 posiciones
J5	Header 4 posiciones
J5	Caja 4 posiciones

Tabla A.2. Material para la tarjeta de la interfase con motor y codificador de posición.

Fuentes

Id	Descripción
R1	Resistencia carbón 5% 470Ω ½ W
R2	Resistencia carbón 5% 10Ω 1/2 W
R3	Resistencia carbón 5% 470Ω ½ W
C1	Capacitor electrolìtico 2200µf 50V
C2	Capacitor electrolìtico 100µf 50V
C3	Capacitor tantalio 0.1µf
C4	Capacitor electrolìtico 2200µf 50V
C5	Capacitor electrolìtico 100µf 50V
C6	Capacitor electrolìtico 100µf 50V f
C7	Capacitor tantalio 0.1µf
Q1	Transistor TIP42
Q1	Disipador de calor TO
U1	Regulador de voltaje 7805
U1	Disipador de calor TO
U2	Regulador de voltaje 7812
U2	Disipador de calor TO
J1	Conector para circuito impreso 2 posiciones
J2	Conector para circuito impreso 2 posiciones
J3	Conector para circuito impreso 3 posiciones
D1	Led rojo
D2	Led rojo
F1	Fusible europeo 1A
F1	Portafusible europeo para circuito impreso
F2	Fusible europeo 5A

F2	Portafusible europeo para circuito impreso
PR1	Puente rectificador 2W02M
PR2	Puente rectificador KBL02

Tabla A.3. Material para la tarjeta del las fuentes de voltaje.

A.2. Programación WINCUPL

La implementación del circuito decodificador de las señales en cuadratura se obtuvo mediante el uso del software WINCUPL que permite diseñar y simular arreglos PLD [2].

A.2.1. Código fuente

```
Name     qd ;
PartNo   00 ;
Date     26/08/2005 ;
Revision 01 ;
Designer CCADET;
Company  CCADET ;
Assembly None ;
Location CU ;
Device   G16V8 ;

/* *************** INPUT PINS *********************/
PIN   1  = clock           ; /*                                */
PIN   2  = QA              ; /*                                */
PIN   3  = QB              ; /*                                */

/* *************** OUTPUT PINS *********************/
PIN   18 = countDN         ; /*                                */
PIN   19 = countUP         ; /*                                */

/* *************** PINNODES    *********************/
PINNODE 12 = QA1           ; /*                                */
PINNODE 13 = QA2           ; /*                                */
PINNODE 14 = QA3           ; /*                                */
PINNODE 15 = QB1           ; /*                                */
PINNODE 16 = QB2           ; /*                                */
PINNODE 17 = QB3           ; /*                                */

/* EQUATIONS */

QA1.D=QA;
QA2.D=QA1;
QA3.D=QA2;
QB1.D=QB;
QB2.D=QB1;
QB3.D=QB2;

countUP=(!QA2 & !QA3 $ !QB2 &  QB3) #
        ( QA2 &  QA3 $  QB2 & !QB3) #
        (!QA2 &  QA3 $  QB2 &  QB3) #
        ( QA2 & !QA3 $ !QB2 & !QB3);

countDN=(!QA2 & !QA3 $  QB2 & !QB3) #
        (!QA2 &  QA3 $ !QB2 & !QB3) #
        ( QA2 & !QA3 $  QB2 &  QB3) #
        ( QA2 &  QA3 $ !QB2 &  QB3);
```

A.2.2. Simulación

El código anterior es simulado en WINCUPL para verificar que el circuito corresponda con lo deseado. La figura A.8 muestra el diagrama de tiempos que se obtiene del código fuente anterior. Es importante destacar que el diagrama obtenido corresponde con lo planeado, es decir, las figuras A.8 y 1.10 representan ambas a un decodificador para codificador del tipo incremental.

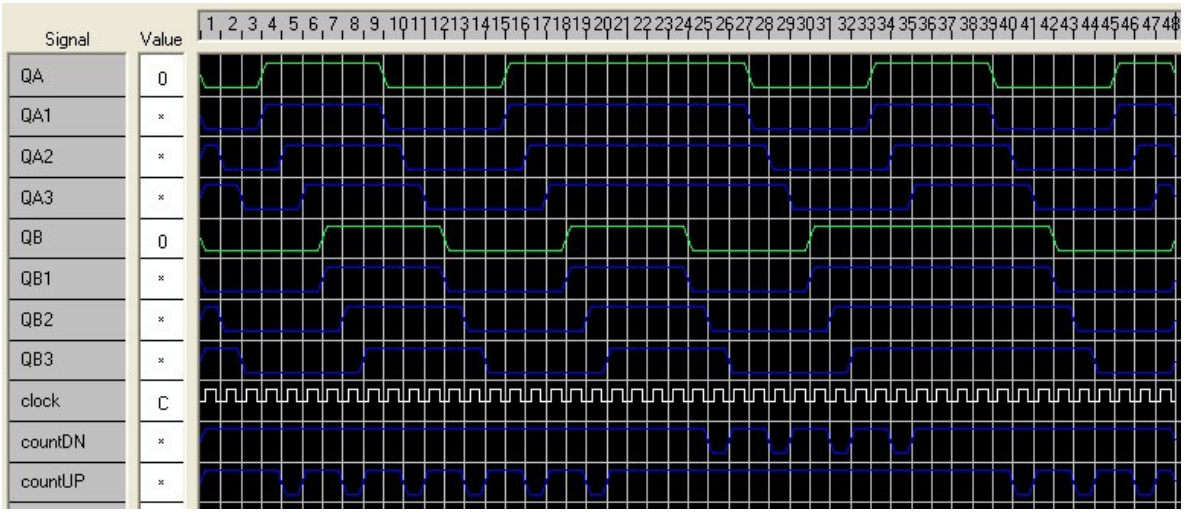

Figura A.8. Simulación en WINCUPL.

A.3. Simulación del PID

El primer paso en la selección de las constantes apropiadas del PID consiste en obtener el modelo de la planta. Una buena aproximación resulta ser el experimento para la respuesta al escalón, como se describe en [16]. La función de transferencia que se obtiene es la siguiente.

$$G_p(s) = \frac{\Theta(s)}{V(s)} = \frac{4.4217}{s(s+3.2922)} \tag{A.1}$$

En donde Θ es la posición angular y V es el voltaje aplicado.

Con esta función de transferencia, el sistema en lazo cerrado de la figura 1.4 se vuelve.

$$G(z) = \frac{\Theta(z)}{M(z)} = \frac{2.2084\times10^{-6} z + 2.206\times10^{-6}}{z^2 - 1.9967 z + 0.9967} \tag{A.2}$$

A continuación se observa la respuesta del sistema con la acción exclusiva del término proporcional unitario.

Sustituyendo (A.2) y la expresión para el controlador PID de (1.17) en la función de transferencia de lazo cerrado de (1.18) con K_p=1, K_d=0 y K_i=0, la respuesta al escalón es la siguiente y se observa gráficamente en la figura A.9.

$$G_l(z) = \frac{\Theta(z)}{R(z)} = \frac{2.208\times10^{-6} z^3 - 2.4\times10^{-9} z^2 - 2.206z}{z^4 - 2.997z^3 + 2.993z^2 - 0.9967z}$$

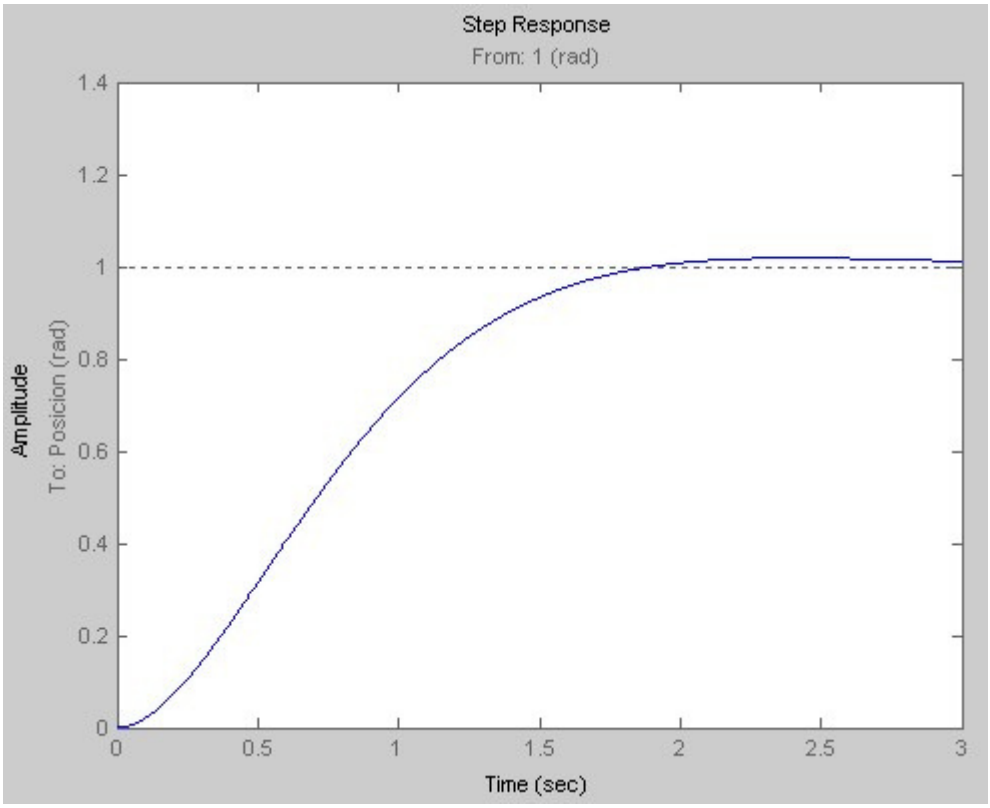

Figura A.9. Respuesta al escalón con K_p=1, K_d=0 y K_i=0.

El segundo paso consiste en reducir el tiempo de subida al incrementar la acción proporcional, es decir, sustituyendo (A.2) y (1.17) en (1.18) con K_p=100, K_d=0 y K_i=0. La respuesta al escalón es la siguiente y se observa gráficamente en la figura A.10.

$$G_l(z) = \frac{\Theta(z)}{R(z)} = \frac{0.0002208z^3 - 2.4\times10^{-7} z^2 - 0.0002206z}{z^4 - 2.996z^3 + 2.993z^2 - 0.9969z}$$

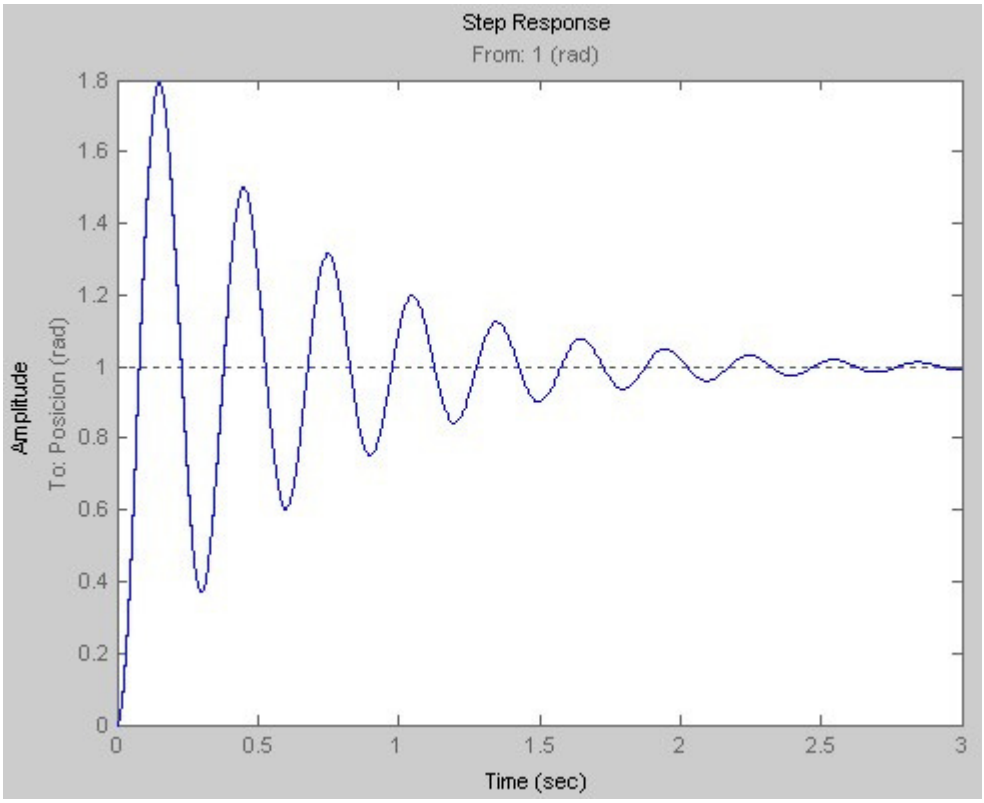

Figura A.10. Respuesta al escalón con K_p=100, K_d=0 y K_i=0.

El tercer paso consiste en reducir el error en estado estacionario al introducir la acción integral, es decir, sustituyendo (A.2) y (1.17) en (1.18) con K_p=100, K_d=0 y K_i=1. La respuesta al escalón es la siguiente y se observa gráficamente en la figura A.11.

$$G_l(z) = \frac{\Theta(z)}{R(z)} = \frac{0.000223z^3 + 1.966\times10^{-6}z^2 - 0.0002206z}{z^4 - 2.996z^3 + 2.993z^2 - 0.9969z}$$

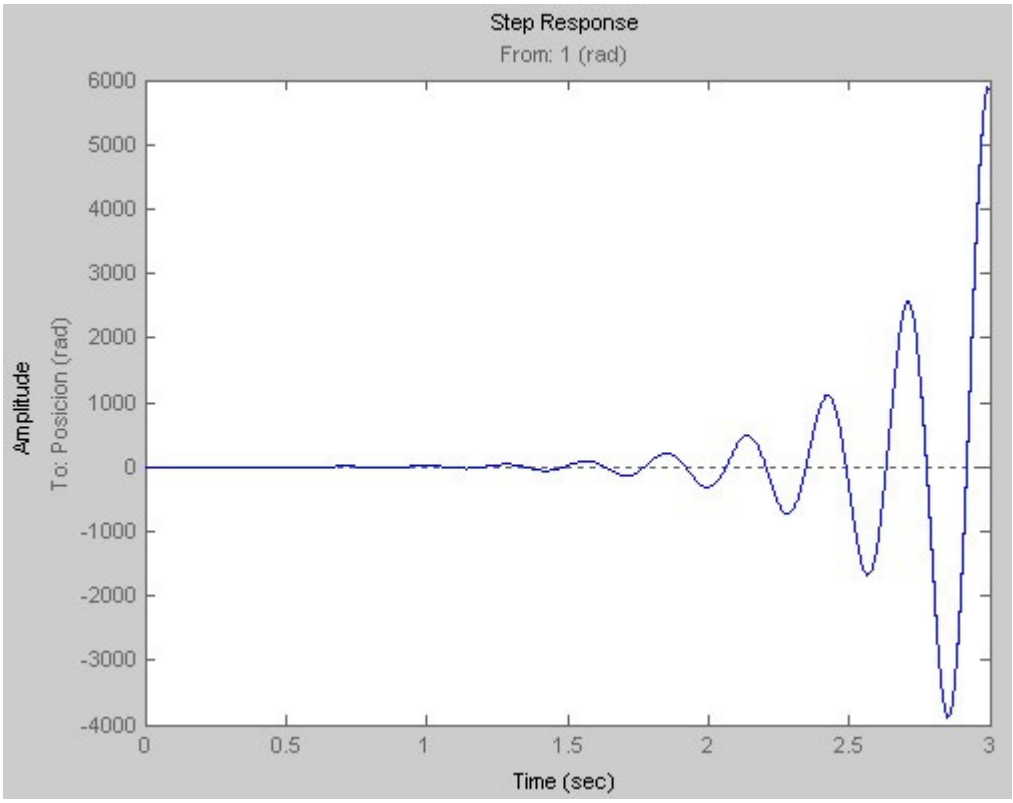

Figura A.11. Respuesta al escalón con K_p=100, K_d=0 y K_i=1.

Como se aprecia en la figura A.11, el sistema presenta inestabilidad, no representa mejora alguna y por lo tanto se descarta el uso de la acción integral.

El cuarto paso consiste en reducir el sobrepaso al introducir la acción derivativa, es decir, sustituyendo (A.2) y (1.17) en (1.18) con K_p=100, K_d=50000 y K_i=0. La respuesta al escalón es la siguiente y se observa gráficamente en la figura A.12.

$$G_l(z) = \frac{\Theta(z)}{R(z)} = \frac{0.1106z^3 - 0.1105\times10^{-6}z^2 - 0.1104z + 0.1103}{z^4 - 2.886z^3 + 2.883z^2 - 1.107z + 0.1103}$$

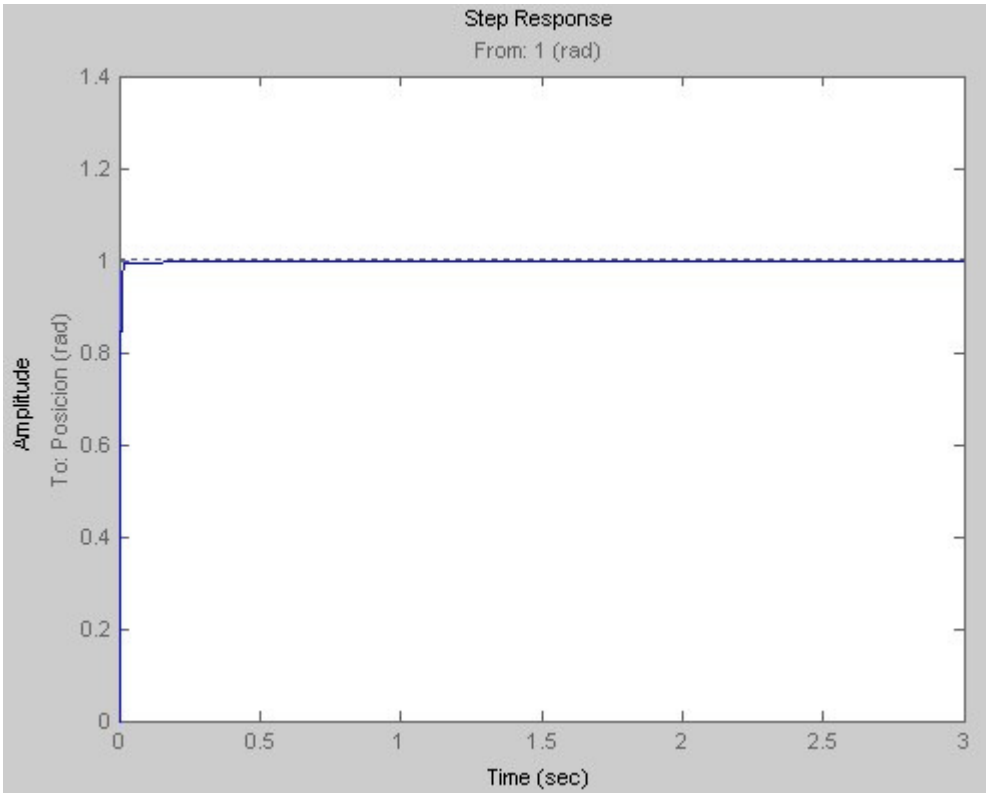

Figura A.12. Respuesta al escalón con K_p=100, K_d=50000 y K_i=0.

La figura A.12 muestra el mejor desempeño que cumple con los criterios de diseño establecidos y por lo tanto se seleccionan las siguientes constantes como punto de referencia en la implementación final del control para servomotor.

$$K_p=100,\ K_d=50000 \text{ y } K_i=0$$

Bibliografía

[1]. K. H. Ang, G. Chong and Y. Li, "PID Control Systems Analysis, Design and Technology" IEEE Trans. Control Systems Technology, vol. 13, no. 4, pp. 559-576, 2005.

[2]. Atmel Corporation, (2006, Nov), "Wincupl" [Online]. http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2759

[3]. J. Axelson, "USB Complete", Lakeview Research, pp. 523, 2001.

[4]. F. J. Ceballos, "Visual C++ Aplicaciones para Win32", Alfaomega, pp. 83, 2004.

[5]. T. L. Floyd, "Fundamentos de Electrónica Digital", Limusa, pp. 389-502, 1996.

[6]. D. J. Kruglinski, G. Shepherd, S. Wingo, "Programming Microsoft Visual C++", Microsoft Press, pp. 1153, 1998.

[7]. D. Lichtel, (2004, May/Jun), "Implementing a USB Equipment Interface Using the Microchip PIC16C745" [Online]. http://www.arrl.org/qex/qx5lichtel.pdf

[8]. A. Macek, (2005, May), "Alan's Home Page" [Online]. http://www.alanmacek.com

[9]. Microchip, "PIC16C745/765 8 Bit CMOS Microcontrollers with USB", Microchip Technology Inc., Chandler, AZ, pp. 165, 2000.

[10]. K. Ogata, "Sistemas de Control en Tiempo Discreto", Pearson Educación, pp. 745, 1996.

[11]. K. Ogata, "Ingeniería de Control Moderna", Pearson Educación, pp. 681-700, 1996.

[12]. D. Panello, (2005, Jun), "Uso del joystick en una aplicación MFC" [Online].

http://www.dcp.com.ar/articulos/art_joystick.htm

[13]. D. Panello, (2005, Jun), "Yerba Mate y Visual C++" [Online]. http://www.dcp.com.ar

[14]. SICK Stegmann, (2006, Nov), "Incremental Rotary Encoders", [Online] http://www.stegmann.com/product/incremental/index.html.

[15]. USB Implementer Forum (2005, Jul), "USB.org" [Online]. http://www.usb.org/home

[16]. B. Valera, S. Padilla, "Implementación de un controlador PID digital", Informe técnico CCADET UNAM, 2002, pp 50.

[17]. Z. H. Xiong and Z. X. Li, "Error Compensation of Workpiece Localization" in Proc. of the 2001 IEEE Intern. Conf. on Robotics and Automation, Seoul, Korea, pp. 2249–2254, 2001.

[18]. Q. Yang, C. Butler and P. Baird, "Error compensation of touch trigger probes" Measurement, vol. 18, no. 1, pp. 47–57, 1996.

[19]. J. Zavaleta, "Control de movimiento para una máquina de medición por coordenadas", Ingeniería en Computación, Facultad de Ingeniería, UNAM, Tesis de licenciatura, México D. F., 6 de diciembre de 2004.