UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Control de movimiento para una máquina de medición por coordenadas

T E S I S

QUE PARA OBTENER EL GRADO DE:
INGENIERO EN COMPUTACIÓN

PRESENTA

JOSUÉ ZAVALETA IRIGOYEN

DIRECTOR DE TESIS: M.I. BENJAMÍN VALERA OROZCO

MEXICO, D.F. Noviembre de 2004.

Índice

Capítulo 1

Conceptos generales

El capítulo uno es una revisión de los conceptos necesarios para adentrarse al problema planteado en la introducción. La medición por coordenadas es un área de alta especialización que agrupa una gran cantidad de disciplinas y técnicas. En particular, para automatizar el movimiento de la MMC del CCADET UNAM, se requiere el estudio previo de los siguientes aspectos: la teoría de control digital y los sistemas de lazo cerrado, la importancia de las MMC's como instrumentos de medición de alta exactitud, los principios de funcionamiento en los transductores de posición comúnmente empleados en MMC's y el principio de medición por contacto mecánico. En forma adicional, es común que la interfase de medición para una MMC resida en una computadora de manera que se ofrezca la capacidad de software de medición mencionada en la introducción. De esta forma describimos los conceptos básicos de programación bajo ambiente Windows que serán de utilidad en la implementación del presente proyecto de tesis.

1.1. Control digital

El control desempeña un papel de una importancia en constante aumento en la mayoría de los aspectos de nuestra moderna forma de vida, al ser un proceso mediante el cual a una variable de un sistema se le establece un valor deseado denominado valor de referencia.

Hasta el surgimiento de los sistemas digitales el único elemento de cálculo con que contaba la Ingeniería de Control eran los computadores analógicos electrónicos. Pero el desarrollo de la electrónica y de los computadores digitales llevó a cambiar rápidamente la concepción. Aunque los objetivos siguen siendo los mismos, la teoría de control analógico difiere sustancialmente del control digital, lo cual conlleva beneficios y perjuicios.

1.1.1. Control digital y control analógico

La década de los años 50 es considerada como parteaguas en la historia del control digital, pues es en estos años donde aparecen las primeras computadoras dedicadas al control de procesos, siendo muy grandes en cuanto a volumen y consumo, y generalmente su fiabilidad no era muy grande [17].

Los primeros computadores digitales fueron usados en sistemas de control de procesos extremadamente complejos. Con la reducción constante de los precios y tamaño hoy se implementan reguladores digitales individuales por lazo de control.

Poco a poco se fueron presenciando avances en varios campos y con diversos tiempos, uno de ellos fue el propio conocimiento del proceso. Los progresos han sido lentos pero constantes, asimismo, en años más recientes se han potenciado por la facilidad en la recolección de datos y su posterior análisis, asociado a esto están las técnicas de medición que se sofistican día a día, al haber cada vez más sensores inteligentes, incluso incorporando computadores a bordo.

En los últimos años se ha incrementado el uso de controladores digitales en sistemas de control, los cuales son utilizados para alcanzar un óptimo desempeño. La tendencia actual de controlar los sistemas dinámicos en forma digital en lugar de analógica, se debe principalmente a la disponibilidad de computadoras digitales de bajo costo y a las ventajas de trabajar con señales digitales en lugar de señales en tiempo continuo.

Los controladores digitales solamente operan sobre números. Una de sus funciones más importantes, consiste en la toma de decisiones, es decir, en la resolución de problemas.

Los computadores digitales son usados también como herramienta para el análisis y diseño de los sistemas automatizados, ya que cuentan con elementos mucho más poderosos que en el pasado, gracias al constante progreso, especialmente con los avances de la tecnología en la integración a muy alta escala.

Los controladores digitales son muy versátiles debido a que pueden manejar ecuaciones de control no lineales que involucran cálculos complejos u operaciones lógicas. Se pueden utilizar también con controladores digitales una amplia variedad de leyes de control que las que se pueden usar con controladores analógicos.

Otra de sus funciones es que mediante la edición de un nuevo programa, las operaciones que se estén ejecutando se pueden cambiar por completo, esta característica es en particular importante, si el sistema de control va a recibir información o

instrucciones de operación desde algún centro de cálculo donde se hacen análisis económicos y estudios de optimización.

Como características básicas del control digital se pueden mencionar las siguientes:

1) No existe límite en la complejidad del algoritmo, lo que sucedía anteriormente con los sistemas analógicos.
2) Facilidad de ajuste y cambio. En el control analógico, implica en el mejor de los casos una sustitución de componentes, o una modificación del controlador completo.
3) Exactitud y estabilidad en el cálculo, debido a que no existen derivas u otras fuentes de error.
4) Uso del computador con otros fines (alarmas, archivo de datos, administración, entre otros.)
5) Costo contra número de lazos. No siempre se justifica un control digital ya que existe un costo mínimo que lo hace inaplicable para un número reducido de variables.
6) Tendencia al control distribuido o jerárquico. Se ha pasado de la idea de usar un único controlador o computador para toda una planta, a la idea de distribuir los dispositivos inteligentes por variable o grupos, e ir formando estructuras jerárquicas.

No siempre los sistemas de control constituyen procesos continuos. Muchas veces son sistemas dinámicos en los cuales una o más variables pueden diferir solamente en ciertos instantes, y por lo tanto las salidas o respuestas de estos sistemas no son funciones continuas en el tiempo, sino funciones discontinuas o discretas.

En la práctica se presentan los sistemas digitales porque alguna medición o señal de control se envía en forma intermitente o periódica. Un proceso discreto conlleva entonces a un muestreo de la señal de entrada para poder procesarla y producir una señal digital de salida.

El muestreo es una característica inherente de muchos sistemas físicos y el comportamiento de estos puede describirse mediante datos discretos o modelos digitales. Para abordar las ventajas y características de los sistemas de datos discretos, es necesario examinar las de los sistemas de control de datos continuos, los cuales tienen muchos años de existencia. La figura 1.1 muestra un diagrama de bloques que contiene los elementos básicos de un sistema de control de datos continuos formado por un solo lazo.

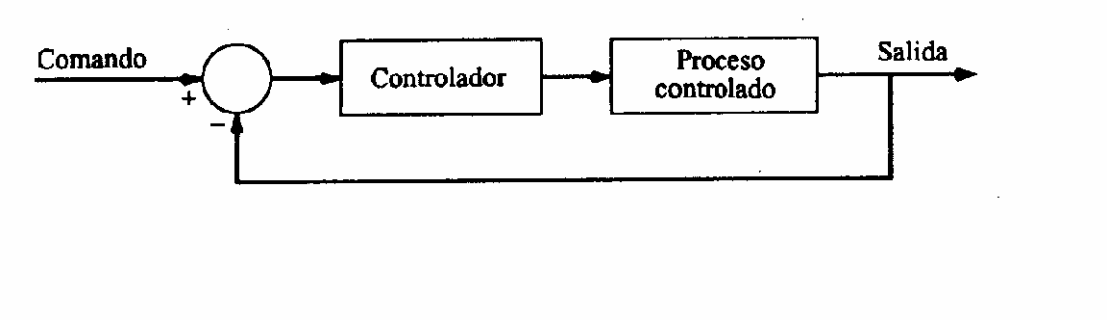


Figura 1.1. Sistema de control continuo en lazo cerrado.

En general el controlador es un circuito electrónico que trabaja con una señal analógica y cuya salida es otra señal del mismo tipo. La ventaja del controlador analógico es que el sistema trabaja en tiempo real y puede tener un ancho de banda muy grande. Esto es equivalente a tener una frecuencia de muestreo infinita, de modo que el efecto del controlador siempre esta presente.

Las desventajas del controlador analógico es que sus elementos se encuentran montados de una manera rígida, de modo que las características del controlador están fijas de antemano, lo que dificulta modificar el diseño. Además, los componentes analógicos exhiben una susceptibilidad mayor a problemas de ruido.

Por ello, el análisis y diseño de sistemas de control de datos discretos y digitales han sufrido transformaciones importantes. Estos sistemas han logrado colocarse en la industria debido a dos razones principales, en primer lugar, gracias a los progresos realizados en computadoras digitales de control, en microprocesadores (MP) y en segundo lugar por los procesadores digitales de señal (PDS).

Así, en el otro extremo tenemos el control digital, el cual permite que los componentes digitales tengan menor susceptibilidad al envejecimiento y a las variaciones de las condiciones ambientales, son menos sensibles al ruido y a las perturbaciones; los procesadores digitales tienen un tamaño y peso menor, pueden fabricarse en MP y PDS de un solo circuito integrado muy versátiles y de gran capacidad para aplicaciones de control, siendo los costos de los MP y PDS cada vez menores, además de que estos permiten una flexibilidad mayor en la programación. Es así, que el control digital demuestra ser más confiable, proporcionando una mejor sensibilidad a variaciones en parámetros.

El programa que caracteriza a un controlador digital puede modificarse con facilidad para dar cabida a cambios en el diseño sin necesidad de efectuar cambios en el hardware. Los componentes digitales en forma de circuitos integrados, partes electrónicas, transductores y codificadores son mucho más confiables y tienen una presentación más robusta y compacta. Es por estas razones que

muchas aplicaciones de sistemas de control analógico están pasando al dominio digital.

Sin embargo, los sistemas de control digital también llegan a presentar desventajas como las siguientes:

1) Limitaciones en la velocidad de cálculo y en la resolución de la señal debido a la longitud de palabra finita del procesador digital.
2) La longitud de palabra finita del procesador digital con frecuencia da origen a inestabilidades en el sistema, mismas que se manifiestan como ciclos límite en sistemas de lazo cerrado.
3) La limitación en la velocidad del cálculo provoca retrasos en el lazo de control, los cuales pueden provocar inestabilidad en sistemas de lazo cerrado.

La figura 1.2 muestra los elementos básicos de un sistema de control digital.

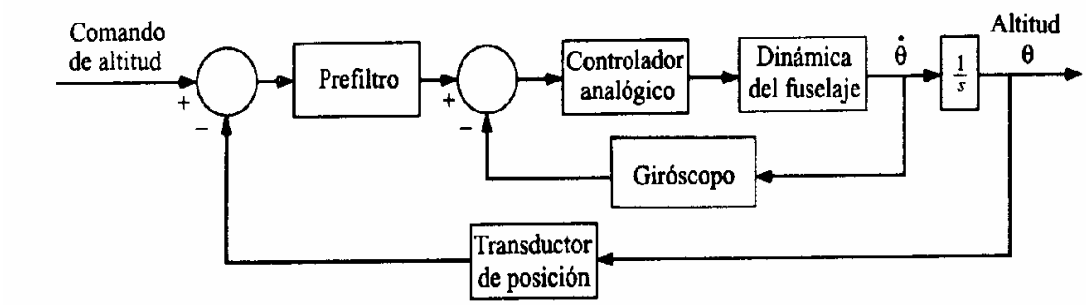


Figura 1.2. Sistema de control digital en lazo cerrado.

Los microprocesadores de propósito especial y los PDS para aplicaciones en sistemas de control se han vuelto algo común. Estos dispositivos compactos y de bajo costo han mejorado con mucho el desempeño y factibilidad de los sistemas de control digital [14, 18].

1.1.2. Control de lazo cerrado para motores CD

La figura 1.3 presenta el diagrama de bloques de un sistema de control de lazo cerrado para motores de C.D [8, 12, 19].

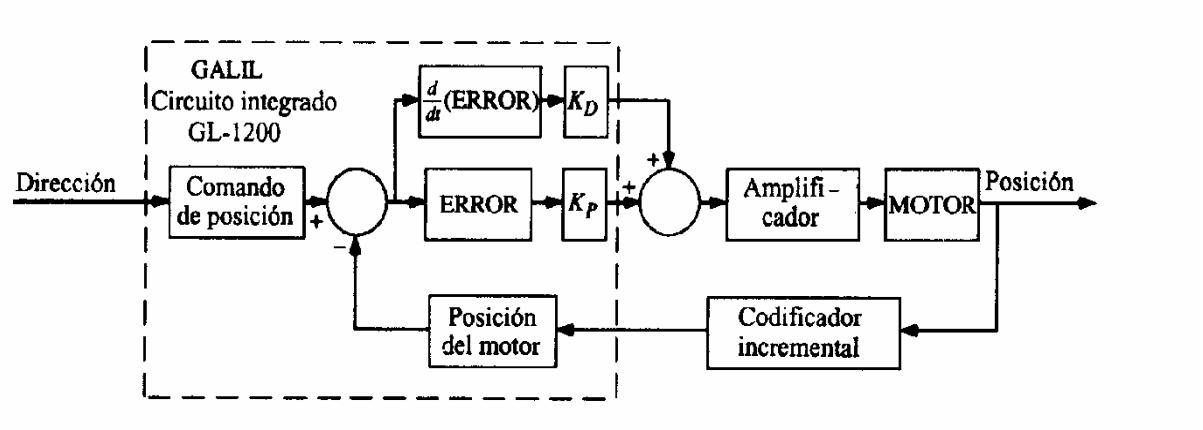


Figura 1.3. Control continuo de lazo cerrado para motores CD.

El controlador acepta como entrada a la posición, la cual se retroalimenta por un codificador incremental de dos canales, y su salida es una señal proporcional al error en la posición y a la derivada de este.

La figura 1.4 ilustra un sistema de control de motor similar al de la figura anterior, pero con un controlador diferente, donde el controlador es un microprocesador de circuito integrado de propósito específico diseñado con controles del tipo proporcional, integral y derivativo (PID).

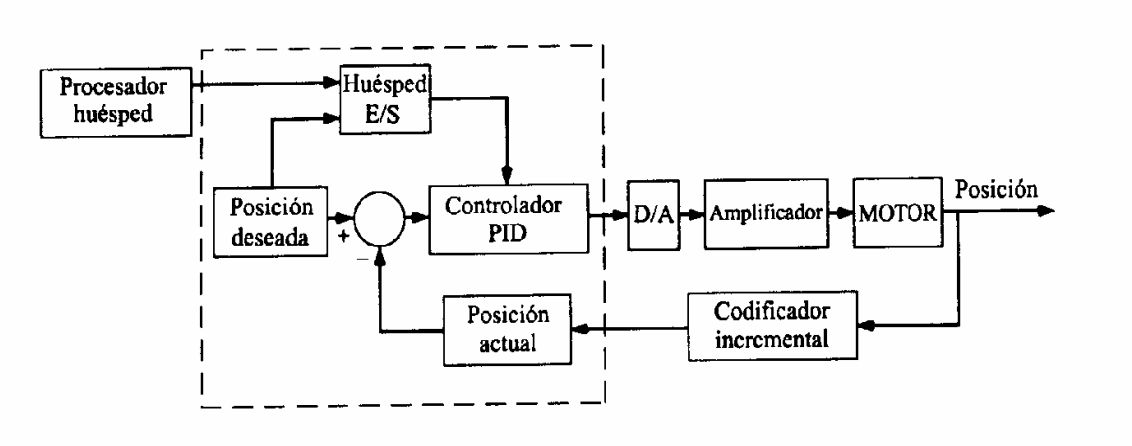


Figura 1.4. Control discreto de lazo cerrado para motores CD.

Los parámetros del controlador PID pueden programarse para procesar los objetivos del desempeño relacionados con el control de la posición y velocidad de un sistema de motor de C.D.

Los dispositivos procesadores de las figuras 1.3 y 1.4 pueden ser reemplazados por un PDS, lo que llevaría a una mejora en el desempeño del sistema. La razón de esto es que los PDS tienen la capacidad de realizar cálculos con una velocidad mayor que la de los microprocesadores, además de que pueden programarse con la finalidad de dotar de características de filtrado con pendientes de

corte muy grandes, imposibles de obtener con los procesadores convencionales.

Así, uno de los controladores más utilizados en el diseño de sistemas de control de datos continuos es el controlador proporcional-integral-derivativo (PID). La figura 1.5 presenta el diagrama de bloques de un controlador de datos continuos PID que actúa sobre una señal de error $e(t)$.

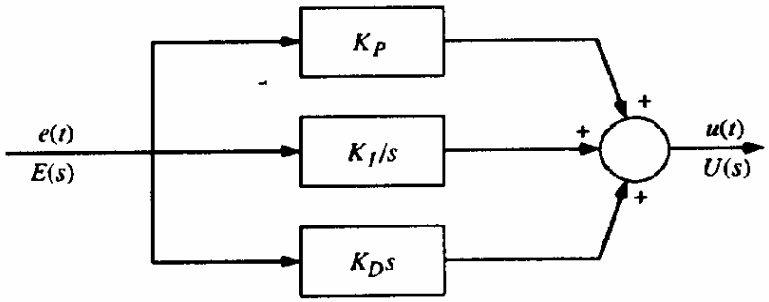


Figura 1.5. Controlador PID continuo.

El control proporcional multiplica $e(t)$ por una constante K_P, el control integral multiplica la integral con respecto al tiempo de $e(t)$ por una constante K_I y el control derivativo genera una señal igual a K_D veces la derivada con respecto al tiempo de $e(t)$. La función del control integral es proporcional una acción que disminuya el área bajo $e(t)$, lo que conduce a la reducción del error de estado estacionario. El control derivativo proporciona una acción anticipativa que reduce el sobreimpulso y las oscilaciones de la respuesta en ele tiempo. El mismo principio del mismo control PID es aplicable al control digital. En este último, el control proporcional se implanta con una constante proporcional K_P.

En general, existen muchas vías para implantar de manera digital la integración y la derivación. Las funciones de transferencia se presentan a continuación:

Integración rectangular

$$D_I(z) = K_I \frac{T}{z-1} \tag{1.1}$$

Integración rectangular hacia delante

$$D_I(z) = K_I \frac{Tz}{z-1} \tag{1.2}$$

Integración con la transformación bilineal

$$D_I(z) = K_I \frac{T}{2}\frac{z+1}{z-1} \tag{1.3}$$

El método más común para aproximar la derivada de *e (t)* en $t=T$ y que origina una función de transferencia físicamente realizable es:

$$\left.\frac{d\,e(t)}{dt}\right|_{t=T} = \frac{e(kT) - e\left[(k-1)T\right]}{T} \tag{1.4}$$

Si se toma la transformada *Z* en ambos lados de la última ecuación y se incluye la constante proporcional K_D, la función de transferencia del controlador derivativo digital es:

$$D_D(z) = K_P \frac{z-1}{Tz} \tag{1.5}$$

La figura 1.6 muestra el diagrama de bloque del controlador digital PID.

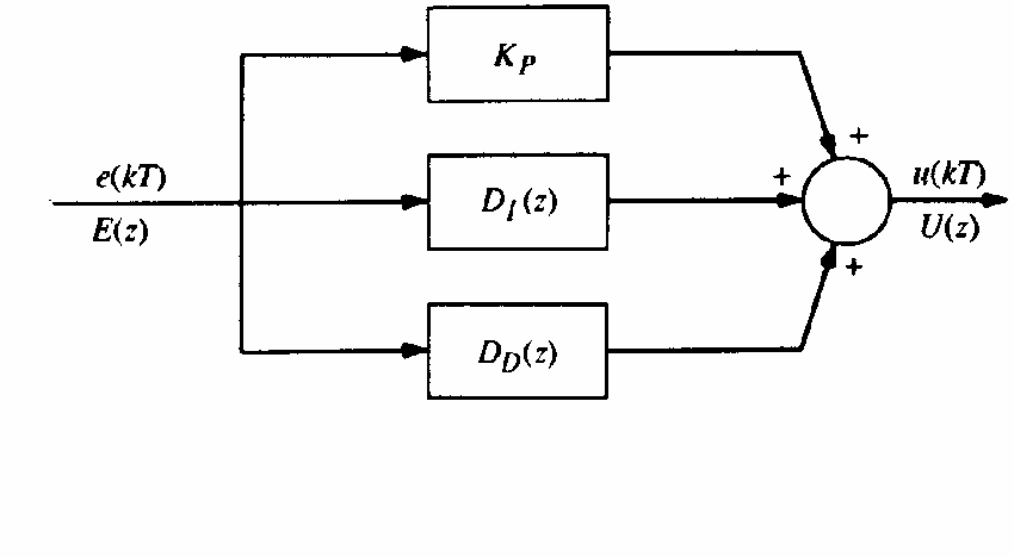


Figura 1.6. Diagrama a bloques de un controlador PID digital.

La siguiente función de transferencia representa el controlador digital PID, donde K_P es la constante del control proporcional, D_I (*z*) es el control integral que puede ser modelado por una de las funciones de transferencia de las ecuaciones 1.1 y 1.3 mencionadas anteriormente, y D_D *(z)* es el control derivativo modelado por la ecuación 1.5. Si se emplean los tres esquemas de integración rectangular, las funciones de transferencia para el controlador digital PID se sintetizan como sigue [14, 18]:

Integración rectangular con diferencias hacia atrás

$$D(z) = \frac{(K_P T + K_D)z^2 + (K_I T^2 - K_P T - 2K_D)z + K_D}{Tz(z-1)}$$

Integración rectangular con diferencias hacia delante

$$D(z) = \frac{(K_P T + K_D + K_I T^2)z^2 - (K_P T + 2K_D)z + K_D}{Tz(z-1)}$$

Integración con la transformación bilineal

$$D(z) = \frac{(2K_P T + K_I T^2 + 2K_D)z^2 - (K_I T^2 + 2K_P T - 4K_D)z + 2K_D}{2Tz(z-1)}$$

Si $K_I = 0$;

$$D(z) = \frac{(K_P T + K_D)z - K_D}{Tz}$$

En consecuencia, el controlador digital PID tiene un polo en $z=0$ y otro en $z=1$. Existen así, dos ceros que pueden ser reales o complejos conjugados.

$$D(z) = K_P + D_I(z) + D_D(z)$$

La importancia del control digital radica en que muchas personas por sus actividades necesitan trabajar con sistemas de control, aunque sea sólo como usuarios. Por lo tanto, para el ingeniero los conocimientos del control digital le son útiles en su formación por ser una disciplina variada que extrae conocimientos de muy distintas ramas del saber. Su estudio puede resultar benéfico al reunir unos temas que se han considerado por separado y aplicarlos a un problema común.

1.2. Máquina de medición por coordenadas

La necesidad de elaborar piezas con las dimensiones ideales, así como mejorar su ajuste entre sí y su funcionamiento, dieron la pauta a los pioneros de la fabricación mecánica para utilizar la información dimensional obtenida durante el proceso de medición de una pieza, mejorando las condiciones y la productividad en el proceso de fabricación.

La necesidad práctica de medir piezas con exactitud dio origen al desarrollo de un sistema de instrumentos de medición de precisión tales como el calibrador, el micrómetro, las galgas de altura y otros dispositivos. De todas las herramientas utilizadas para medir las dimensiones de una amplia gama de piezas, las Máquinas de Medición por Coordenadas (MMC) son las que presentan mayor capacidad. La figura 1.7 muestra una MMC del tipo puente.

El sistema de coordenadas es utilizado para describir los movimientos de una máquina de medición, y permite localizar características en relación con otras características de una misma pieza. El sistema por coordenadas se parece mucho a un plano de alzado en el que la combinación de una letra a un extremo del plano con un número al otro extremo, y alzados a lo largo del plano, es lo que describe cada localización en un mapa. Esa combinación letra/número/alzado es lo que se conoce como una coordenada y representa un lugar completo en relación con los otros.

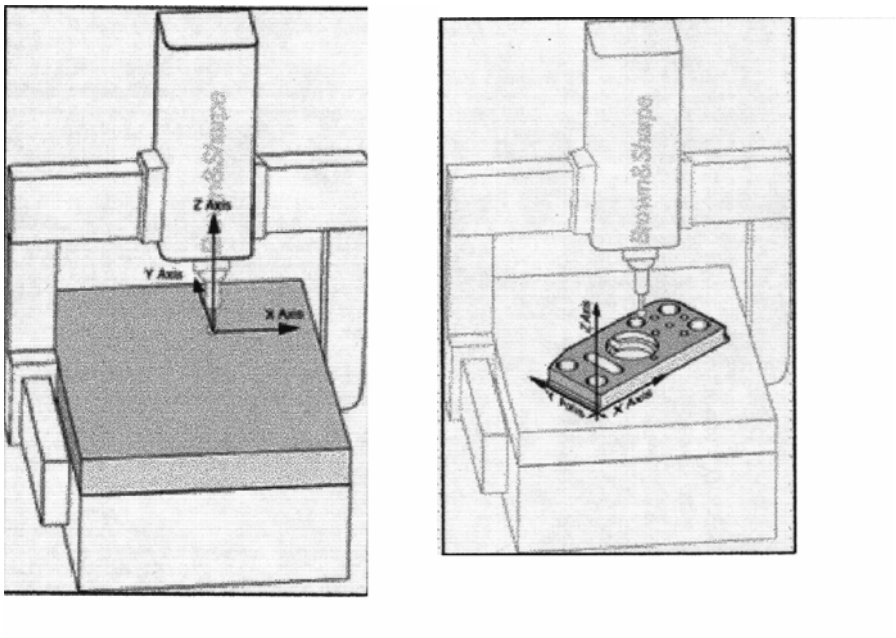


Figura 1.7. MMC del tipo puente.

Las MMC han sido consideradas como un punto de inflexión en la tecnología de máquinas tridimensionales, ya que adquieren información dimensional detallada desplazando un palpador a través de las superficies de la pieza de trabajo. Por lo general las MMC toman los datos palpando la pieza con un sensor adjunto en los ejes de medición de la máquina. El sensor puede ser sólido o electrónico y funcionar con un accionador. Aunque el palpador del sensor es muy preciso, una vez que el sensor se ha adjuntado a la MMC, el posicionamiento del palpador en el sistema de la máquina de medición por coordenadas se tiene que determinar antes de medir, puesto que es la circunferencia la que palpa la pieza, el centro y el radio del sensor se determina midiendo una esfera muy precisa. La mayoría de las MMC adquieren los datos utilizando un palpador que detecta el contacto con puntos individuales de la pieza de trabajo. Esta técnica de medida de puntos individuales puede recoger datos por lo general, a velocidades máximas de 50 a 60 puntos por minuto, lo que la hace considerablemente más eficiente que los instrumentos manuales de medición.

Lo que hace valiosa la metrología por coordenadas como herramienta de control de procesos, es que puede ser usada para medir con exactitud objetos en un amplio margen de tamaños y configuraciones geométricas, y discernir la relación entre diferentes propiedades de una pieza de trabajo. Esta flexibilidad, y la velocidad de operación de la medición por coordenadas comparada con las técnicas de superficies planas y de galgas fijas, significa que los resultados de la medición pueden ser utilizados para refinar de una manera económica aplicaciones de procesos de fabricación, además de analizar las tendencias del mismo.

La dificultad con la metrología por coordenadas, a pesar de su indudable utilidad y valía, ha sido que las rutinas de medición debían ser frecuentemente realizadas fuera de línea por técnicos especialmente entrenados, que eran retirados de las operaciones de fabricación. Las inspecciones fuera de línea de este tipo se convirtieron más en una herramienta de control y aseguramiento de calidad para verificar montajes o descubrir piezas que no cumplen especificaciones antes de ser enviadas a los usuarios, que en un verdadero control del proceso.

Con la introducción de las técnicas de control estadístico de procesos, los técnicos de calidad pueden usar los datos dimensionales adquiridos por las MMC para estudiar tendencias de no-conformidad y ofrecer este análisis al personal de fabricación, el cual usa esta información para corregir las variaciones del proceso que producían dichas inconformidades.

Durante las dos últimas décadas se ha producido una tendencia hacia la integración de la velocidad, flexibilidad y exactitud de las MMC con las operaciones de talleres. Uno de los factores que ha impulsado esta tendencia es la creciente demanda de una producción con tolerancias tan ajustadas que hasta las más pequeñas divergencias de proceso pueden producir piezas inutilizables.

Las mediciones se han ido integrando cada vez más con el proceso de mecanizado. Por ejemplo, en las líneas de transferencia se ha unido físicamente a la línea una estación de medida. En otros casos, la integración de la medida ha sido facilitada por algunos tipos de mecanismo de transferencia, tales como vehículos guiados, carros, dispositivos de manipulación o transportadores aéreos, que transfieren las piezas desde la máquina-herramienta hasta la MMC para su inspección.

Las ventajas de las inspecciones en un taller, ubicando la inspección dimensional en las proximidades de las operaciones de fabricación, se ejerce un mayor control sobre el proceso, a diferencia de cuando los datos son adquiridos en una localización remota.

Además, la medición integrada en un taller es realizada normalmente por los operadores de las máquinas, lo que reduce la necesidad de inspectores, asimismo, las mediciones en un taller pueden ser empleadas para compilar bases de datos del comportamiento de las máquinas, las herramientas, las piezas, las paletas y las fijaciones durante el proceso real de mecanizado.

Las MMC utilizadas en estas aplicaciones tienen la ventaja de ser flexibles, es decir, con un simple cambio de programa de software pueden ser utilizadas para inspeccionar piezas con diferentes dimensiones, así como fijaciones, paletas y herramientas.

Con el software, las MMC miden los datos de las piezas a partir de la impresión de la parte, y establecen el Sistema de Coordenadas de la Pieza y matemáticamente lo ponen en relación son el Sistema de Coordenadas de la Máquina, así el alineamiento es el proceso que se lleva a cabo para relacionar los dos sistemas de coordenadas.

Existen dos tipos de sistemas de coordenadas en el mundo de la medición. El primero se conoce como Sistema de Coordenadas para la Máquina. Aquí los ejes *X*, *Y* y *Z* se refieren a los movimientos de la máquina. Si se toma una perspectiva desde la parte frontal de la máquina, el eje *X* va de izquierda a derecha, el eje *Y* va hacia delante y hacia atrás, y el eje *Z* va de arriba a bajo, verticalmente perpendicular a los otros dos ejes. El segundo sistema de coordenadas se conoce como Sistema de Coordenadas para la Pieza. Los tres ejes tienen relación con los datos o características de la pieza. Antes de la introducción de software para llevar a cabo mediciones por coordenadas, las piezas se alineaban de forma paralela a los ejes de la máquina de manera que los Sistemas de Coordenadas para la Máquina y la Pieza estaban paralelos el uno con el otro. Esto requería mucho tiempo y los resultados no eran precisos.

El obstáculo principal en la precisión de las medidas de una pieza es la variación de la temperatura y el efecto que esta tiene. Una primera solución para este problema fue el aislamiento de una MMC tradicional para protegerla de gradientes térmicos y acondicionar el ambiente dentro de un recinto para mantener la temperatura constante. En la práctica, esta solución encerró a la MMC en un ambiente de laboratorio, que si bien fue una buena solución en algunas circunstancias, tiene el inconveniente de sacrificar la capacidad de producción a cambio de la exactitud, además, encerrarla en un ambiente controlado presenta algunas dificultades en la carga y descarga de las piezas.

Un nuevo enfoque es la compensación de la dilatación y distorsión térmicas con una mezcla de soluciones hardware y software que disponen de una red de sensores ubicados en lugares

críticos de la estructura de la máquina. Los sensores leen la temperatura en la estructura de la máquina y un algoritmo extrapola los valores de la dilatación y la distorsión. Con estos datos el software es capaz de compensar el estado térmico actual de la máquina, de tal manera que la influencia de las variaciones de temperatura prácticamente se cancelen. El resultado es que es posible hacer inspecciones dimensionales con una exactitud comparable con la de laboratorio, aunque se trabaje en un taller.

En la medida en que las piezas fabricadas se hacen más precisas y las tolerancias requeridas son menores, es preciso adquirir y analizar más datos para poder determinar la viabilidad del proceso de fabricación de piezas. Las MMC con funciones de barrido proporcionan los medios para ello. El barrido es simplemente una forma de recoger automáticamente un conjunto de coordenadas de puntos para definir con exactitud la forma de la pieza de trabajo.

La función de barrido se consideró en algún momento exclusiva de los fabricantes de alta tecnología, porque las MMC capaces de realizar el barrido y el software para manejarlo eran costosos y de difícil adquisición. Sin embrago, hoy los avances en software y en tecnología de sensores han hecho las máquinas MMC considerablemente más asequibles.

Los nuevos sensores combinan elementos de las tecnologías óptica, de video y láser en dispositivos que pueden barrer rápidamente formas complejas y superficiales, y recoger con exactitud datos dimensionales.

Algunos de estos sensores pueden leer hasta 20 000 puntos por segundo con extrema exactitud. Combinados con estos sensores se dispondrá de poderosas máquinas matemáticas que analizarán rápidamente la gran cantidad de datos dimensionales que estos sistemas pueden generar.

Actualmente las MMC que llegan al mercado son cada vez más rápidas, más precisas y más asequibles que sus predecesoras de hace solamente unos pocos años. El mayor valor de la metrología en un taller no es solamente la posibilidad de integración con el proceso, sino la unión que permite entre el objetivo de diseño y la función de fabricación, este es el salto cualitativo que dará continuidad a las mejoras de calidad que son una parte integral del diseño, la ingeniería y la fabricación [2, 6, 7].

1.3. Medición de posición usando MMC

Una MMC es un instrumento que integra sensores de longitud y contacto sobre una estructura mecánica de alta precisión. Los sensores proporcionan las señales de información para que el

software de la máquina sea capaz de realizar mediciones. En primer lugar, los ejes coordenados del sistema mecánico deben contar con el medio adecuado para el registro de su desplazamiento. El medio más común lo conforman los codificadores ópticos de alta resolución. Por otra parte, el contacto mecánico entre el palpador de la MMC y la pieza en inspección debe ser registrado con gran exactitud evitando que el mismo contacto deforme o desplace la pieza.

1.3.1. Codificadores ópticos

Los codificadores ópticos son dispositivos que se emplean para medir la posición y la velocidad de un motor. Estos dispositivos constan de un disco transparente con una serie de marcas opacas colocadas radialmente y equidistantes entre sí y un sistema de iluminación en el que la luz es dirigida a un fotorreceptor, figura 1.8. La flecha del motor se acopla al disco transparente generando pulsos a medida que la luz atraviesa cada marca en el disco, llevando una cuenta de estos pulsos en un dispositivo contador. El conteo es la indicación del movimiento y este se incrementa y se decrementa según el sentido del giro en el motor [5].

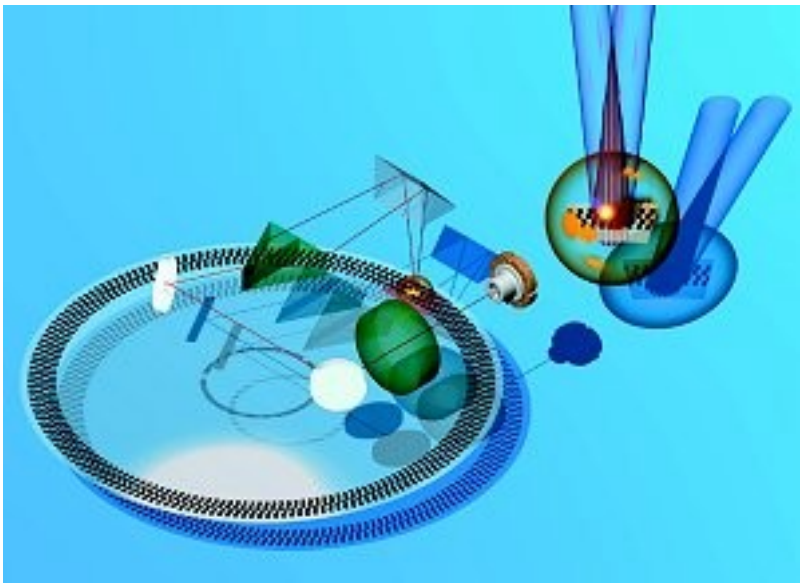


Figura 1.8. Estructura de un codificador óptico.

Los codificadores ópticos generalmente cuentan con dos salidas o canales (A y B) en cuadratura para determinar la dirección del movimiento, figura 1.9. Esto permite contar las transiciones y observar el estado del otro canal durante el movimiento para determinar si el canal A adelanta al canal B o el canal B al canal A obteniendo así el sentido del giro. La siguiente figura muestra a la señal en cuadratura de cada canal y su desfasamiento entre las dos señales.

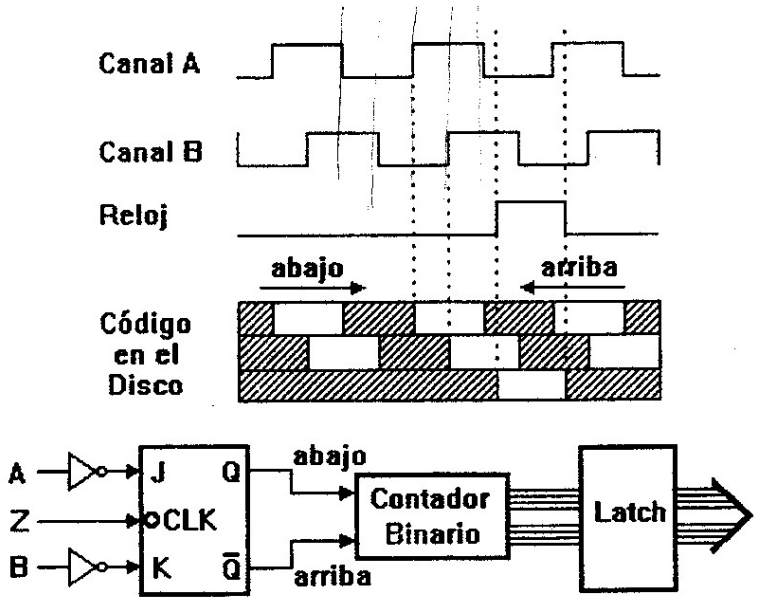


Figura 1.9. Señales en cuadratura de un codificador óptico.

La resolución de este tipo de dispositivos depende directamente del número de marcas con que cuente el disco. Un método para aumentar esta resolución es, no solamente contabilizar los flancos de subida de los trenes de pulsos, sino contabilizar también los de bajada, incrementando así la resolución del captador, pudiendo llegar, con ayuda de circuitos adicionales, a factores de multiplicación por 2 y por 4. El codificador empleado en el prototipo construido es de dos canales y de 100 cuentas por revolución, con lo cual, al aumentar la resolución por 4, se obtiene una resolución de 400 cuentas por revolución. Sin embargo, en una MMC real como la desarrollada por el Laboratorio de Metrología mostrada en las figuras 2 y 3, los codificadores son lineales con resolución de 2μm. No obstante, el principio de operación de los codificadores angulares, como el utilizado en el prototipo, y de los codificadores lineales, como el utilizado en una MMC, son prácticamente idénticos.

1.3.2. Medición por contacto

Cuando la esfera del palpador entra en contacto con la pieza de trabajo, la fuerza de contacto entre ellos crece gradualmente. Una señal de disparo $p_i(t)$ se genera cuando la fuerza excede el umbral en el interruptor eléctrico integrado al sistema de sensado. La señal se utiliza por el software en la PC para registrar las coordenadas espaciales en el instante de disparo. Sin embargo, después del instante de contacto, el palpador continua viajando en la dirección del movimiento debido a la inercia mecánica del sistema e inclusive viaja más allá del instante de disparo. La distancia recorrida entre el instante de contacto y el instante de disparo constituye el error de pre-recorrido. Tal error se debe a la fuerza de doblamiento natural

del palpador y de los interruptores eléctricos. Adicionalmente el software en la PC debe proporcionar un medio de control autónomo para retirar el palpador de la pieza de trabajo en sentido opuesto al del viaje del palpador.

Cuando el sistema ha generado la señal de disparo, las coordenadas registradas corresponden al centro de la esfera del palpador, **C**. No obstante, el punto exacto de contacto es **M**, como se muestra en la figura 1.10. La dirección de movimiento, *x*-, y el instante de disparo, $t+\delta t$, son conocidos precisamente por el software de control en la PC que originó el movimiento y por el interruptor que accionó la señal de disparo. Si se asume que la dirección de movimiento es normal a la tangente de la superficie de la pieza de trabajo y que ocurre en un solo eje coordenado, por ejemplo en x, la compensación de los errores por radio del palpador, *r*, y por el pre-recorrido en *x*, ed_x, es la siguiente

$$\mathbf{M}_x = x(t+\delta t) + ed_x - r \tag{1.6}$$

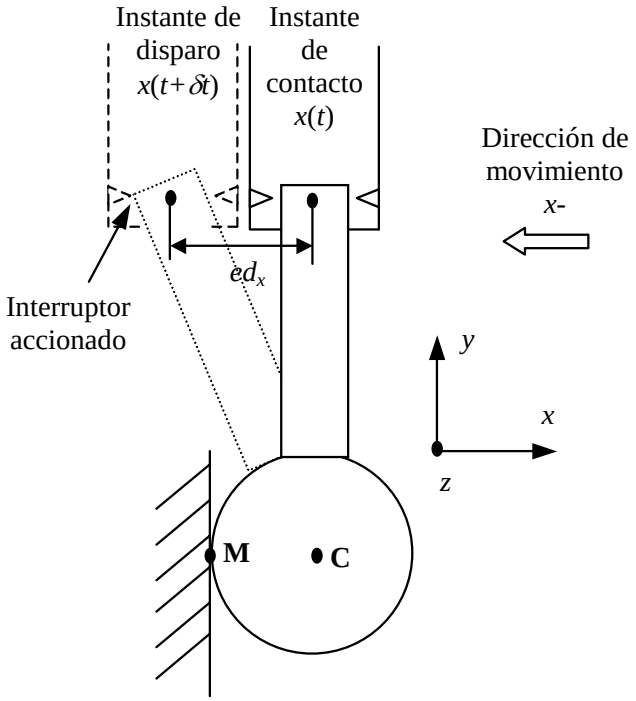


Figura 1.10. Modelo simple de medición por contacto.

Para el caso de la figura 1.10. De forma similar se pueden deducir las relaciones para palpaciones en los restantes ejes con sus respectivas direcciones de movimiento. El error por pre-recorrido es del tipo determinístico y se puede obtener de forma experimental. Otros modelos más realistas para la compensación de errores incluyen direcciones de movimiento no normales a la tangente de la superficie, combinaciones de movimiento en los tres ejes coordenados ó estrategias especiales de palpación [21, 24, 25].

Es importante resaltar que el esquema propuesto en la figura 4 proporciona todos los elementos para resolver la compensación planteada en la ecuación (1.6). En particular, los signos de los errores son proporcionados por la dirección de movimiento y la lectura en el instante del disparo es proporcionada por los interruptores del palpador conectados a las entradas $p_i(t)$.

1.4. Programación en ambiente Windows

Debido a la disponibilidad de las computadoras personales con sistema operativo Windows y a la gran diversidad de equipo periférico que soportan estos equipos, nuestro proyecto contempla la implementación de una interfase de usuario usando como herramienta de desarrollo Visual C++ 6 para Windows 00/XP. A continuación, una descripción de utilidad en la creación de aplicaciones Windows.

1.4.1. Introducción

Actualmente las computadoras se han convertido en una herramienta imprescindible de trabajo para miles de usuarios. Debido a esto se hace necesario que los programas de computadora se comporten de un modo confiable y predecible, para ello es importante entender cómo la programación orientada a objetos y manejada por eventos contribuye a un formato bien estructurado y modular de un programa.

Ante esta necesidad Windows es considerado como el entorno más popular de interfase gráfica de usuario, siendo un entorno multitarea basado en ventanas que se corresponden con programas, es decir, permite ejecutar programas especialmente escritos para dicho entorno y programas escritos para MS-DOS.

Para desarrollar programas, Windows proporciona rutinas que permiten la utilización de componentes como menús, cuadros de diálogo y barras de desplazamiento, entre otros. Así existe la manipulación de cualquier objeto que tenga contacto directo con el programador.

Una aplicación típica de Windows presenta una o más pantallas llenas de objetos con los cuales interactúa el usuario para determinar el flujo de un programa. En el sentido visual más sencillo, los objetos de programación son sus formas de aplicación y los controles que se trazan sobre ellas.

Los objetos y eventos de la programación están íntimamente relacionados. Los eventos tienen lugar como resultado de la acción del usuario o del código del programa, o pueden ser activados por el sistema. La mayoría de los objetos responden a un número de eventos generados por el usuario, desde la opresión de una tecla

hasta un clic de ratón. Además, los objetos pueden reconocer eventos del sistema, incluyendo los eventos de temporizador y de carga, activando el primero a intervalos especificados y el segundo cuando un objeto, como una forma, es cargado por primera vez en la memoria [3, 4, 9].

Microsoft Visual C++ se presenta como un entorno de programación en el que se combinan la programación orientada a objetos y el sistema de desarrollo diseñado especialmente para crear aplicaciones gráficas para Windows.

Las aplicaciones Windows son sencillas de utilizar, sin embargo, el desarrollo de las mismas no es fácil. Por ello, Visual C++ incluye varias herramientas que lo convierten en un generador de programas C++ y un conjunto completo de clases, permitiendo crear las aplicaciones para Windows y manejar sus componentes según la naturaleza de los objetos.

Para el desarrollo de aplicaciones para Windows, Visual C++ se centra en dos tipos de objetos: ventanas y controles; ambos permiten diseñar sin programar una interfaz gráfica para una aplicación.

Para realizar una aplicación se crean ventanas a veces llamados formularios, y sobre ellas se dibujan otros objetos llamados controles. Es decir, cada objeto (ventanas y controles) está ligado a un código que permanece inactivo hasta que se presenta un suceso que lo activa.

Visual C++ es un paquete cuya función principal es desarrollar aplicaciones y sus características más sobresalientes son [3]:

1) Una biblioteca de clases, MFC (Microsoft Foundation Class), que da soporte a los objetos Windows tales como ventanas, cajas de diálogo, controles, así como a los objetos GDI (Graphic Device Interface) tales como lápices (pens), pinceles (brushes), fuentes (fonts), y mapas de bits (bitmaps). A su vez, los objetos se comunican entre sí mediante mensajes, también soportados por las MFC [9].
2) Un entorno de desarrollo integrado (editor de texto, compilador, depurador, explorador de código fuente, administrador de proyectos, etc.).
3) El editor de textos le ayuda a completar cada una de las sentencias visualizando la sintaxis correspondiente a las mismas.
4) Asistentes para el desarrollo de aplicaciones como AppWizard, editores de recursos (editor de menús, de diálogos, de tablas de cadenas de caracteres, de tablas de aceleradores y de objetos gráficos como mapas de bits o íconos), ClassWizard, ControlWizard, WizardBar y ATL COM

Wizard (Activate Template Library Component Objet Model Wizard).

5) Galería de objetos incrustados y vinculados (OLE - Object Linking and Embedding). Esto es, software autocontenido en pequeñas y potentes unidades o componentes software para reutilizar en cualquier aplicación. Asimismo, Visual C++ proporciona soporte para diseñar componentes software a medida.
6) Visualización y manipulación de datos de otras aplicaciones Windows utilizando controles OLE.
7) Una interfaz para múltiples documentos (MDI - Multiple Document Interface) que permite crear una aplicación con una ventana principal y múltiples ventanas de documento.
8) Personalización de AppWizard.
9) Cabeceras precompiladas que reducen el tiempo de compilación.
10) Editar y continuar. Durante una sesión de depuración, se pueden realizar modificaciones en el código de la aplicación sin tener que salir de dicha sesión, recompilar y reiniciar la depuración. Los cambios son recompilados y aplicados a la aplicación que se está ejecutando.
11) Nuevas clases para la programación de hilos (threads), para implementar páginas HTML, etc.
12) Creación y utilización de bibliotecas dinámicas (DLL - Dynamic Link Libraries).
13) A partir de la versión 4.2, Visual C++ integra la biblioteca estándar de C++ e incorpora soporte para la incorporación de aplicaciones para internet; forma parte de este soporte la tecnología de componentes activos (ActiveX).
14) Soporte para el estándar COM (Component Object Model - modelo de objeto componente; en otras palabras componente software) al que pertenecen los componentes activos (ActiveX o formalmente controles OLE).
15) Objetos de acceso a datos (DAO) que permiten acceder a bases de datos a través del motor de Access o de controladores ODBC.
16) OLE DB como un proveedor de datos y objetos ADO (ActiveX Data Objects – objetos ActiveX para acceso a datos), como tecnología de acceso a datos, para satisfacer los nuevos escenarios demandados por las empresas, tales como los sistemas de información basados en la Web.

17) Soporte para aplicaciones que interaccionen con Internet a través de la API para Internet de Windows (biblioteca WinInet).

18) Incorporación de nuevas definiciones. Por ejemplo, el tipo bool y sus constantes asociadas true y false, los tipos _variant_t (COleVariant), _bstr_t y _com_ptr_t, la directriz #import, etc.

Cuando se da la combinación de estas características, se dispone de un potente sistema de desarrollo que permite diseñar eficientemente aplicaciones sofisticadas.

1.4.2. Características de una aplicación

El proceso de desarrollo de una aplicación Visual C++ se divide en dos fases: creación del esqueleto de la aplicación y desarrollo de la aplicación.

Así, cuando se crea una aplicación Visual C++, lo primero que se hace es generar con AppWizard un conjunto de archivos de partida, que forman la base de una aplicación genérica que se denomina "esqueleto de una aplicación". La figura 1.11 ilustra gráficamente el proceso mencionado.

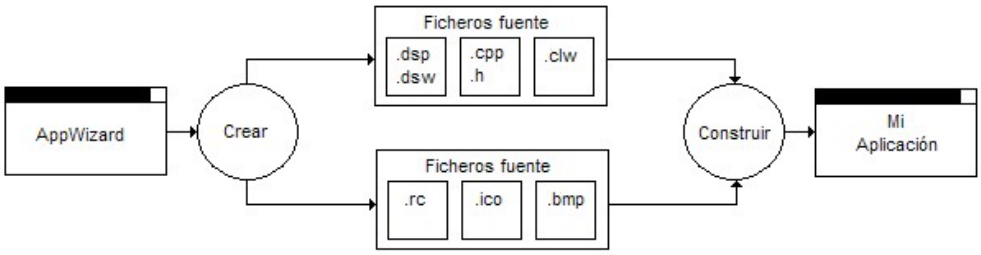


Figura 1.11. Proceso de creación de una aplicación.

La fase de desarrollo de una aplicación Windows incluye el diseño de la interfaz de usuario (edición de los archivos de recursos), la edición de los archivos fuente, la compilación, el enlace, la prueba y la depuración de la aplicación. Estas actividades son interactivas y entrelazadas por lo que no procede enumerar los pasos a seguir. La figura 1.12 ejemplifica lo mencionado.

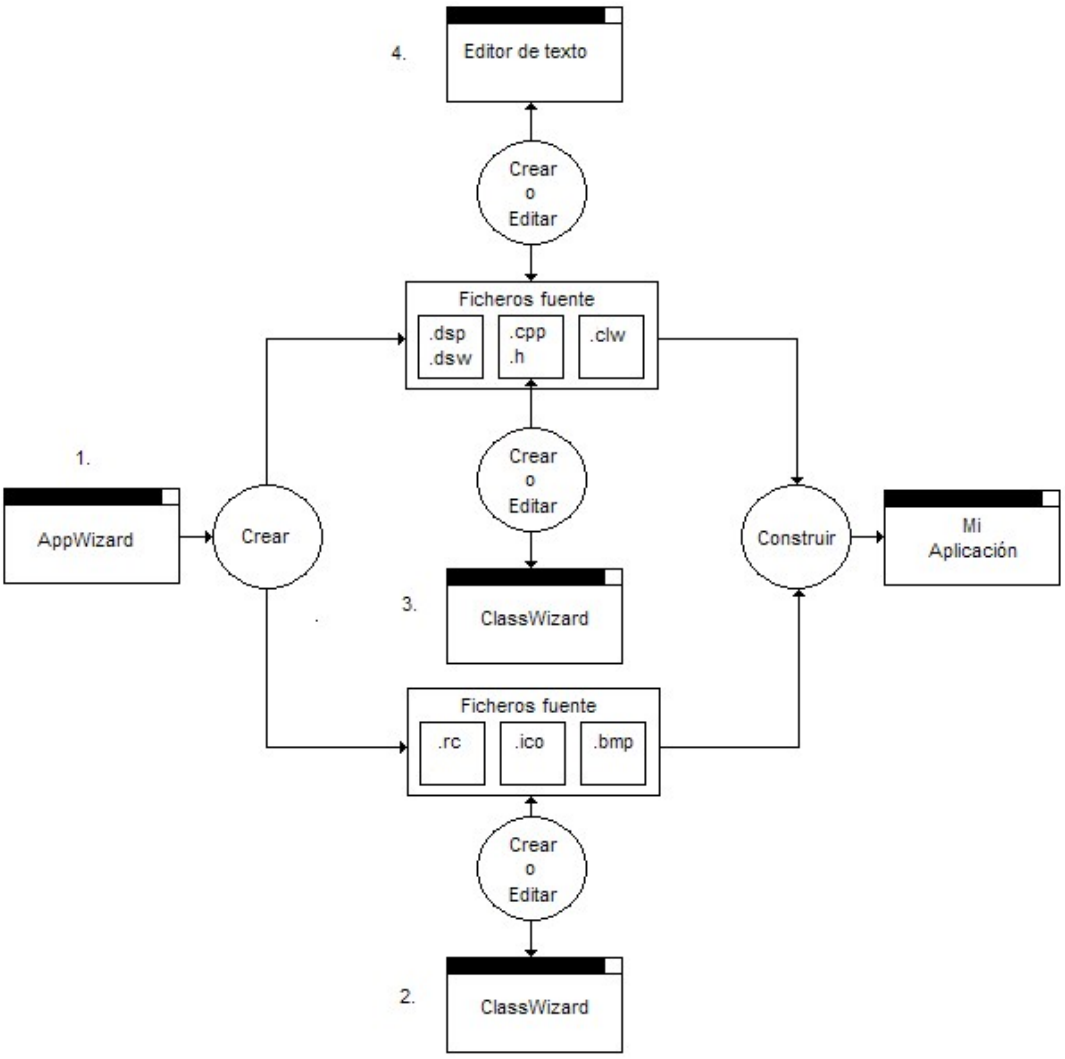


Figura 1.12. Fases de desarrollo de una aplicación.

En pocas palabras, el orden normal del desarrollo de la aplicación es el siguiente:

1) Crear el esqueleto de la aplicación con AppWizard.
2) Utilizar los editores de recursos para construir la interfaz del usuario.
3) Utilizar ClassWizard para crear el esqueleto de los controladores de eventos (funciones de tratamiento de mensajes) relativos a los objetos de la interfaz del usuario.
4) Utilizar el entorno de desarrollo para editar el código correspondiente a cada uno de los controladores de eventos asociados con los objetos de la interfaz del usuario.

1.4.3. Comunicaciones por el puerto serie

Son diversas las formas a través de las cuales una PC puede comunicarse con el exterior, además de la pantalla y el teclado. Razón por la cual, toda PC que se precie de serlo debe contar con

una serie de puertos de entrada/salida para poder conectarse a otros periféricos [3, 15, 23].

Existen dos puertos principales de los que dispone la PC, los de tipo serie y paralelo, entre los que hay una serie de distinciones:

1) Los puertos serie siempre han sido bidireccionales, mientras que los puertos paralelos son sólo los de la última generación.
2) En el puerto serie se trasmite la información en serie bit a bit, mientras que en el puerto paralelo la información pasa a través de ocho hilos simultáneamente, de manera que se transmite de byte a byte.
3) El puerto paralelo es menos susceptible de ruidos e interferencias que el de serie.
4) La comunicación por puerto serie exige menos conexiones que el paralelo.
5) Mientras que el puerto paralelo funciona con niveles lógicos de 0v y +5v para cero y uno respectivamente, el puerto serie funciona a -12v y +12v para el cero y el uno.

Estas diferencias irreconciliables entre los puertos serie y paralelo los hacen incompatibles entre sí, destinándose cada uno de ellos a funciones determinadas. Sin embargo, existen adaptadores de niveles y señales que permiten la interconexión entre ellos sin ningún problema.

Debido a la importancia que ambos puertos tienen para la comunicación de la PC, se hace necesaria la explicación de dichos puertos, sin embargo, cabe mencionar que para efectos de esta tesis al puerto serie se le dará mayor énfasis.

El puerto paralelo es el que se utiliza para la comunicación con la impresora. Su conexión exterior se realiza mediante un conector DB25 macho, pero no se usan todos los pines, siendo gran parte correspondientes a masa o negativo.

En un sistema PC se pueden conectar hasta tres puertos paralelos, si bien lo más normal es tener sólo uno o dos. Las direcciones I/O y los IRQ son los siguientes:

- Puerto paralelo 1 : I/O = 378 IRQ = 7
- Puerto paralelo 2 : I/O = 278 IRQ = 5
- Puerto paralelo 3 : I/O = 3BC IRQ = 7

Las señales más usuales en el puerto paralelo son:

- D0…D7: Son los 8 bits utilizados para la transmisión de datos.
- Busy: Esta señal la manda la impresora o el aparato que está conectado al puerto para indicar que está ocupado.

- Acknowledge: Señal de reconocimiento del puerto al aparato.
- Ready: Señal de “todo listo” del aparato al puerto paralelo.
- Reset: Señal del puerto al aparato para su inicialización.
- Paper Out: Condición de aviso de la impresora sobre la falta de papel.

Una de las aplicaciones de los puertos paralelos es el conexionado de dos computadoras para su comunicación mediante los programas Interlnk, LapLink, etc. Debido a las características de los puertos actuales estos es posible.

Si bien en el pasado los puertos eran auténticas interfaces Centronics (el Interface Centronics es el estándar utilizado para las impresoras), hoy día son PIA’s (Programmable Interface Adapter), de manera que los pines pueden ser definidos como entrada o como salida; en resumen pueden mandar datos o recibirlos.

Durante mucho tiempo el puerto paralelo ha sido del tipo SPP (Stándar Parallel Port), sin embargo son de reciente aparición el EPP (Enhacement Parallel Port) y el ECP (Enhacement Controller Port), mediante las cuales la velocidad de transmisión ha aumentado de manera importante.

El otro gran camino de salida del PC es el puerto serie, en el cual los datos viajan de bit a bit sobre un único cable de forma asíncrona, es decir, imprevisible; lo que obliga a procesar los datos conforme llegan al buffer de recepción del puerto.

Desde el principio de las computadoras, el puerto serie ha sido por excelencia el más utilizado para la comunicación, a pesar de ser el más lento. Actualmente, los puertos serie han cambiado mucho y sus velocidades, sin ser como las del puerto paralelo han incrementado su velocidad.

El motivo de la lentitud del puerto serie es que utiliza sólo dos hilos para la comunicación, uno para mandar datos y otro para recibir, de manera que es necesario perder tiempo enviando señales de sincronismo para que funcionen de manera fiable.

El circuito encargado de generar las señales y los sincronismos se denomina UART, y siempre ha sido el menos utilizado, puesto que nunca ha sido lo bastante veloz. La UART más utilizada ha sido la 8250, con la cual no se conseguían velocidades adecuadas.

Cabe señalar que en años recientes han aparecido las 16550AF, que permiten gran velocidad de transferencia y mayor fiabilidad.

El direccionamiento de los puertos serie, conocidos como COM es el siguiente:

- COM1 : I/O = 3F8 IRQ = 3

- COM2 : I/O = 2F8 IRQ = 4
- COM3 : I/O = 3E8 IRQ = 3
- COM4 : I/O = 2E8 IRQ = 4

La conexión al puerto serie se realiza mediante conector DB9 o DB25 macho. Normalmente, en el puerto serie existe una gran cantidad de señales lo que justifica el DB25, pero sólo unas pocas son las imprescindibles para la comunicación, de manera que caben en el DB9.

Las señales más importantes son:

- Tx: Transmisión de datos.
- Rx: Recepción de datos.
- RTS: Request-to-Send, petición de envío de datos.
- CTS: Clear-to-Send, listo para enviar datos.
- DSR: Data-Send-Ready, envío de datos listos.
- DTR: Data-Terminal-Ready, terminal de datos preparada.
- RI: Ring Indicator, indicador telefónico.

Existen diversos protocolos de comunicación serie, pero todos ellos tienen la misma estructura:

- 1 bit de comienzo ó “Start”.
- 7 u 8 bits de datos.
- 1 ó 2 bits de parada ó “Stop”.
- 1 bit de paridad.

A pesar de todos estos bits, lo más normal es utilizar el formato 8N1, es decir, 8 bits de datos, ninguna paridad y 1 bit de parada.

La paridad consiste en contar todos los bits que van circulando en ese momento y ver si el número de unos lógicos es par o impar (“Odd” ó “Even”). Si es par, se escribirá un uno, y si es impar se escribirá un cero. Posteriormente, la paridad se consulta para ver si todo es correcto y así detectar posibles errores de transmisión.

Respecto a la velocidad, esta se mide en baudios, que equivalen a bits/segundo. De esta forma, una transmisión a 9600 baudios equivale a 9600 bits por segundo, pero no estamos hablando sólo de los bits de datos, ya que también se incluyen los de paridad, comienzo, stop y los espacios entre paquete y paquete, por ello la transmisión serie tiene menos rendimiento.

Con las antiguas UART, se conseguían hasta 19200 baudios, con la 16550AF se pueden obtener 115200 baudios. La 16550AF es de difícil localización y puede llegar hasta los 4Mbits por segundo.

Aunque se le ha considerado como el sistema más lento, el puerto serie se utiliza hoy en día para la comunicación por MODEM, ya que la línea telefónica dispone de dos hilos, y ese es el número mínimo para la comunicación serie.

Capítulo 2

Circuitos electrónicos para el control de movimiento

El control de motores juega un papel clave en la construcción y modernización de máquinas de medición por coordenadas, ya que es el principio del movimiento de los ejes de la máquina. A medida que los movimientos se automatizan, es necesaria una fiabilidad y una precisión mayor; esto, se logra a través de controladores digitales formados básicamente por circuitos electrónicos. Existen en el mercado una gran diversidad de fabricantes de controladores de motores, ya sean motores a pasos, motores AC, o DC según las necesidades de los usuarios.

Para la máquina de medición por coordenadas construida en el Laboratorio de Metrología del CCADET UNAM se cuenta con un módulo de control de motores PIC-SERVO por cada eje, un codificador incremental retroalimentado por cada motor y un adaptador de puerto serie para el control de su movimiento.

2.1. Módulo de control

El módulo de control de movimiento, PIC-SERVO KAE-T0V4-DPS, está formado principalmente por dos dispositivos PIC que son los encargados de enviar y recibir la información hacia y desde el motor [11].

2.1.1. Descripción

La tarjeta de control PIC-SERVO KAE-T0V4-DPS es un sistema completo de servo-control que incluye un servo controlador, un amplificador, una interfase de comunicaciones, una interfase para codificador óptico incremental, interfase para interruptores de límite y una entrada con pre-amplificador auxiliar, figura 2.1. Esta

tarjeta está diseñada para soportar hasta 32 controladores que pueden ser conectados directamente a un sólo puerto serie estándar (con un convertidor RS232 ó RS485).

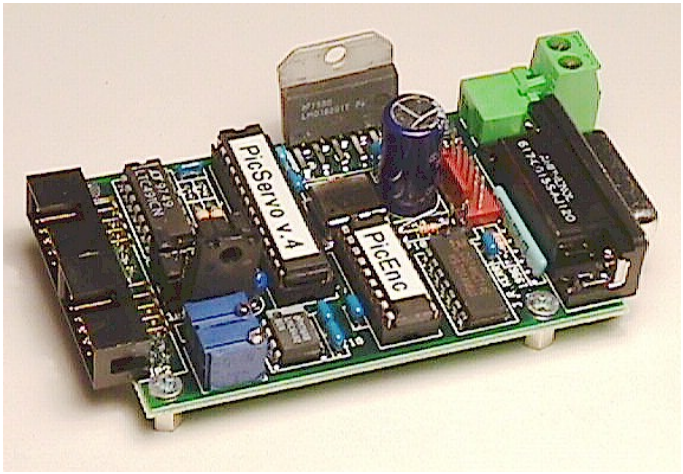


Figura 2.1. PIC-SERVO KAE-T0V4-DPS.

Los dispositivos del PIC-SERVO KAE-T0V4-DPS forman el núcleo principal del módulo. El PIC-SERVO es un microcontrolador PIC16C73 programado con un controlador PID y una interfaz de comandos seriales. El PIC-ENC es un microcontrolador PIC16C54 programado como un codificador incremental de 16 bit. Ambos dispositivos están basados en la serie de microcontroladores PIC16CXX [16].

El microcontrolador PIC16C73 es un dispositivo con 28 pines que generalmente opera con +5V y es compatible con la lógica de dispositivos TTL y CMOS. La tabla 2.1 resume las funciones de cada uno de los pines.

PIN	SIMBOLO	DESCRIPCION
1	MCLR	Reset pin, activo en bajo. Se conecta a Vcc para un reset automático al encenderse
2	CUR_SENSE	Entrada analógica del sensor o para uso de entrada analógica en general. (0 - +5V)
3	ADDR_OUT	Esta salida esta en nivel alto al encenderse y disminuye su valor cuando la dirección del chip se programa a un valor único. Se usa en conjunto con la entrada ADDR_IN del siguiente PIC-SERVO en la misma red de trabajo RS485.
4	LIMIT1	Entrada del interruptor de límite
5	ADDR_IN	El pin debe ser llevado a un valor bajo para habilitar la comunicación. Normalmente ligado al pin ADDR_OUT del PIC-SERVO que le antecede en la misma red RS485
6	INDEX	Normalmente se conecta a la salida INDEX del codificador incremental y se usa para propósitos de Residencia
7	LIMIT2	Entrada del interruptor de límite
8	GND	GND
9	OSC1	Conectado a un reloj fuente de 20 MHz o a un lado de un cristal de 20 MHz
10	OSC2	Conectado al otro lado del cristal e 20 MHz
11	DIR	Bit de salida del control de dirección. Se usa con la salida del PWM para controlar un servo amp.
12	AMP_EN	Salida que habilita el amplificador. Se establece el valor en alto o bajo para habilitar el servo amplificador.

13	PWM	20 Khz. (aprox.) de señal cuadrada con variación de porcentaje en ciclo de trabajo. Se usa con DIR para controlar un servo amplificador
14	ENC_SEL / PWR_SENS	Esta salida se conecta al ENC_SEL en el PIC-ENC para seleccionar el byte alto o bajo del contador
		de 16 bit. Lógica 1 selecciona el byte alto, lógica 0 el byte bajo. Este in es monitoreado como
		entrada para detectar la presencia de alimentación del motor conectándolo a la fuente del motor
		a través de un divisor de voltaje. De otra forma debe ser ligado a una fuente de +5V a través
		de una resistencia de 10 K
15	ENC_RES	Se conecta al pin ENC_RES del PIC-ENC. Establecer este valor en alto inicializa el codificador a cero
		Establecer este valor en bajo permite el conteo normal.
16	XMT_EN	Esta salida puede conectarse al pin ENABLE del driver RS485 para habilitar la salida si se usa una
		Línea de repuesta compartida. No se usa si una conexión es punto a punto (RS232).
17	TX	Salida de transmisión serial. Se conecta a la entrada de transmisión del driver RS232 o RS485
18	RX	Entrada de recepción serial. Se conecta a la salida de recepción del driver RS232 o RS485
19	GND	GND
20	Vcc	+ 5V
21-28	RB0-7	Entrada de datos del codificador. Se conecta a las salidas del PIC-ENC

Tabla 2.1. Descripción de los pines en el PIC16C73.

El microcontrolador PIC16C54 es un dispositivo con 18 pines, el cual, usualmente opera con +5V y es también compatible con la lógica de dispositivos TTL y CMOS. La tabla 2.2 resume las funciones de cada uno de los pines

PIN	SIMBOLO	DESCRIPCIÓN
1	ENC_SEL	Cuando esta entrada se encuentra en un nivel bajo, la salida serán los 8 bits menos significativos
		de los 16 bits de conteo en los RB0-7, si se encuentra en un nivel alto, la salida serán los 8 bits
		más significativos.
2	ENC_RES	Cuando esta entrada se encuentra en un nivel alto, el valor del contador interno será cero, y un
		valor cero será la salida en el RB0-7. Si la entrada es un nivel bajo, permite un conteo normal.
3	Not Used	Este pin se liga conecta a Vcc.
4	MCLR	Reset pin, activado en bajo. Se conecta a Vcc para un reset automático al encenderse.
5	GND	GND
6 - 13	RB0-7	Salida del contador del codificador.
14	Vcc	+5V
15	OSC2	Conectado al otro lado del cristal e 20 MHz
16	OSC1	Conectado a un reloj fuente de 20 MHz o a un lado de un cristal de 20 MHz
17	CH_A	Entrada del codificador del canal A
18	CH_B	Entrada del codificador del canal B

Tabla 2.2. Descripción de los pines en el PIC16C54.

El PIC-SERVO involucra un tipo de comunicación *full-duplex* (RS232-RS485) basada en una red de control modular (NMC por sus siglas en inglés), que en sentido estricto es un protocolo Maestro/Esclavo, donde los comandos son enviados en forma de paquete al módulo de control desde una computadora que cumple la función de host (anfitrión) tomando el papel de maestro. De esta

manera, el módulo tomando su papel de esclavo responde a la computadora con un paquete de estado que contiene información básica de las condiciones del módulo de control, así como los datos de posición de motor e información del tipo de módulo empleado y su versión.

Para lograr la comunicación entre el host y el módulo, es necesario establecer los parámetros de velocidad de baudios en la comunicación, la cual es de 19,200 pudiendo modificarse hasta 115,200. El protocolo de comunicación usa un bit de comienzo, un bit de paro y no utiliza paridad.

Los paquetes de comando son transmitidos por el host a través de una línea de comandos, y a su vez los paquetes de estado son recibidos a través de una línea de estado que es compartida por todos los módulos de la red. Debido a que la línea de comando no se comparte con ningún dispositivo, el puerto de comunicaciones del host puede ser de tipo estándar RS232. Sin embargo, los puertos esclavos, deben de tener la capacidad de deshabilitar sus transmisores para prevenir colisiones de datos en la línea compartida de estado. En el esquema de la figura 2.2 se puede apreciar de forma general la configuración de la comunicación.

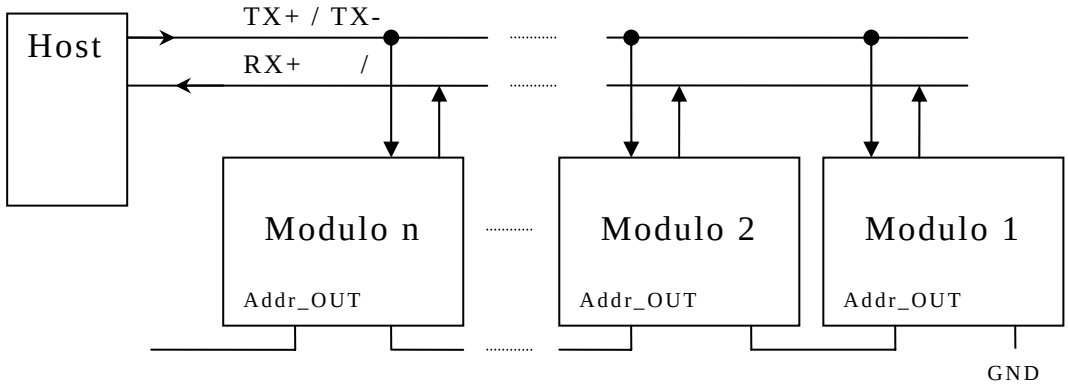


Figura 2.2. Configuración de múltiples módulos.

Los paquetes de comando tienen la siguiente estructura:

- Byte de Cabecera (siempre 0xAA)
- Byte Dirección de Módulo (0 – 255)
- Byte Comando
- Byte(s) de dato(s) Adicional(es)
- Byte Checksum (suma de 8 bit desde el byte de dirección hasta el byte de dato)

Es importante resaltar las funciones que cumplen los bytes en los paquetes de comando. Así, el byte de cabecera es utilizado para

indicar el comienzo de un paquete de comando, por ello, cada módulo ignora cualquier dato hasta que aparece el byte de cabecera.

El byte de dirección es el identificador del módulo al cual se le da la instrucción de ejecución, de esta manera la dirección puede ser individual o por grupo de módulos.

El byte de comando se divide en dos partes: nibble alto y nibble bajo. El nibble bajo contiene el valor del comando (0-15), y el nibble alto contiene el número de bytes de datos adicionales que requiere dicho comando (0-15), siendo el host el encargado de asegurarse que el número de datos adicionales en cada comando coincida con el número de bytes realmente enviados. El byte de datos adicionales contiene información específica que requieren algunos comandos.

Una vez que el módulo recibe un paquete completo y el byte de dirección coincide con la dirección de dicho módulo, se verifica que el byte checksum sea correcto, de ser así, inmediatamente comienza el proceso del comando. Si existe un error en el byte de checksum, el comando simplemente no se ejecuta, sin embargo, el paquete de estado si es enviado al módulo.

Los paquetes de estado tienen la siguiente estructura:

- Byte de estado
- Bytes de datos adicionales (programables)
- Byte de checksum

Cabe mencionar que el byte de estado contiene información básica de las condiciones del módulo, incluyendo si el comando anterior presentó un error en el byte de checksum.

El número de bytes de datos adicionales es programable. El contenido de información del byte de estado depende de la programación de lectura y escritura del estado de los comandos. Cada módulo de la red envía sólo el byte de estado sin datos adicionales en el momento en que se enciende o se inicia algún módulo.

Un comando que se envía al controlador PIC-SERVO es alojado en un buffer interno hasta que el ciclo actual del servo finaliza (aproximadamente tarda 0.51 milisegundos), es entonces cuando es ejecutado y el byte de datos es recibido.

Ningún comando debe ser enviado al módulo de control hasta que se recibe el paquete de estado, esto con la finalidad de prevenir sobre-escrituras en el buffer de comandos y colisiones en la línea de estado. En el caso de que el host envíe un comando antes de recibir un paquete de estado, todos los módulos esclavos existentes en la red deshabilitaran toda transmisión de datos en progreso y esperarán

otro comando proveniente del host. Esto asegura que el host puede demandar la atención de los módulos esclavos de la red.

El comando de referencia describe la información contenida en los paquetes de comando y en los paquetes de estado.

Cuando dos o más módulos están conectados a una misma red NMC, se les asigna una dirección. Esto se logra a través del uso de los pin ADDR_IN y ADDR_OUT en cada controlador compatible NMC. El pin ADDR_OUT de un controlador es ligado al pin ADDR_IN del controlador adyacente en la red. Normalmente, el pin de ADDR_IN del controlador tiene una ubicación más lejana al host por lo que es ligado a través del GND, y el pin ADDR_OUT del controlador es el que se encuentra más cerca al host.

Las direcciones son asignadas utilizando el siguiente procedimiento:

1) En el momento de arranque, todos los módulos asumen una dirección de 0x00, y cada uno será colocado en su pin de ADDR_OUT en un nivel alto. Además los módulos de comunicaciones serán invalidados completamente hasta que el pin ADDR_IN se encuentra en un nivel bajo. Si los pin ADDR_OUT y ADDR_IN son ligados, todos los módulos serán invalidados excepto el módulo más alejado del host.
2) El host comienza enviando un comando de dirección establecido al módulo 0, cambiando esta dirección por el valor 1. La mayor falla que puede presentarse es el cambio de valores de alto a bajo.
3) El siguiente módulo en línea es habilitado con una dirección 0. El host envía un comando al módulo 0 para cambiar esta dirección por un valor de 2.
4) Estos procesos son continuos hasta que a todos los módulos les haya sido asignada una dirección.

Inicialmente el host tiene la tarea de dar un direccionamiento cada vez que la red NMC es iniciada o reiniciada. El host también puede utilizar este mecanismo para verificar que el número de módulos presentes sea apropiado, y que sus tipos coincidan con los esperados para una aplicación particular. Una vez establecida la dirección, todas las demás operaciones pueden ser ejecutadas.

Cada módulo de control tiene dos direcciones: una dirección individual y una dirección grupal. Cuando la dirección individual es iniciada o reiniciada le es asignado el valor 0x00, y a la dirección grupal se le asigna el valor 0xFF. Ambos tipos de direcciones son asignadas con el mismo comando de direccionamiento. La dirección individual puede tener un valor entre 0 y 255, pero la dirección grupal está restringida a valores entre 128 y 255.

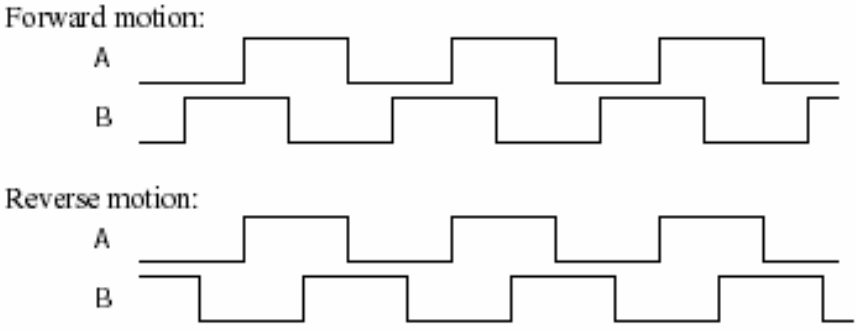
El objetivo de la dirección grupal es enviar un sólo comando para varios controladores al mismo tiempo. Mientras la dirección individual de todos los controladores debe ser única, la dirección grupal puede compartir un grupo de controladores. Cuando un paquete de comando es enviado a la dirección grupal de la red NMC, el comando podrá ejecutar todos los módulos que coincidan con la dirección grupal.

El problema de que cada módulo envíe como respuesta un paquete de estado a un grupo de comando suele resolverse con la diferenciación entre los miembros del grupo y los líderes. Cuando la dirección grupal para un módulo es establecida, el comando de dirección será especificado si el módulo es de los líderes, de los miembros o del grupo. Si un módulo es de uno de los miembros del grupo y recibe un comando de grupo, será ejecutado el comando pero no reenviado a un paquete de estado. Si un módulo es el líder de este grupo y recibe un comando de grupo, se reenviará a un paquete de estado y se ejecutará el comando.

Si un grupo de módulos comparten la misma dirección grupal, sólo uno puede ser declarado líder de grupo.

En ocasiones es necesario enviar un comando a un grupo con ausencia de líder. En este caso, no se recibirá información de los módulos, y el host podría esperar por lo menos 0.51 milisegundos antes de enviar otro comando para conservar la sobre-escritura de los comandos previos.

La salida de un codificador incremental de doble canal es una señal cuadrada de 50% de trabajo de +90° ó -90° fuera de fase, dependiendo de cual es la dirección de rotación del motor, ver figura 2.3



.

Figura 2.3. Señales en cuadratura.

Por ejemplo, un codificador de 500 líneas puede producir cuatro señales por línea como límite (dos en el canal A, dos en el canal B), para un total de 2000 líneas por revolución como límite. El PIC-ENC ejecuta en un tiempo crítico la tarea de detección de transiciones límite de incremento o decremento de un contador

interno de 16 bit. El ENC_SEL de entrada es usado para especificar si byte alto o bajo del contador aparece en el pin de salida RBO-7.

2.1.2. Programación

Las operaciones necesarias para la iniciación y/o reiniciación de la red son:

1) Establecer o fijar los parámetros de comunicación del puerto a 19,200 baudios, un bit de comienzo, un bit de paro y sin paridad.
2) Enviar una cadena de 16 bytes nulos (0x00) para rellenar los buffer de comando, y descartar los bytes recibidos en el buffer de recepción.
3) Usar el comando de dirección establecido para asignar sólo una dirección individual para cada módulo. Hasta este punto, todas las direcciones grupales deben ser establecidas en 0xFF, y no declarar ningún líder en el grupo.
4) Verificar que el número de módulos encontrado coincida con el número esperado.
5) Para los diferentes módulos del controlador, habrá diferentes tipos y versiones de números (PIC-SERVO= tipo 0). Usar el comando de estado lector para leer el tipo y versión de los números de cada módulo, y verificar que coincidan con los tipos y versiones esperadas.
6) Enviar un comando de baudio establecido a la dirección grupal 0xFF para cambiar el índice del baudio por el valor deseado.
7) Cambiar el baudio del host para que coincida con el índice especificado.
8) Para obtener cada uno de los módulos individuales se verifica que estos estén operando en un nuevo índice de baudio.
9) Usar el comando de dirección establecido para asignar alguna dirección grupal que sea necesaria.

Con lo hasta ahora expuesto se podrá iniciar el envío de un módulo de comando específico a un módulo individual, y se podrá usar el comando de dirección establecida para reasignar la dirección grupal.

El PIC-ENC fue diseñado específicamente para usarse con PIC-SERVO pero también puede ser usado de manera individual. Sin embargo, el PIC-ENC no es del todo apropiado porque no verifica los datos, y por lo tanto los PIN RBO-7 debe ser leído dos veces para asegurar que los pin sean leídos mientras están en transición, o mientras el byte bajo esté rodando sobre el byte alto. En general el procedimiento para leer el PIC-ENC es el siguiente:

1) Establecer ENC_SEL para leer el byte alto.
2) Esperar 2.8 microsegundos para cambiar la información.
3) Leer el byte alto.
4) Bajar ENC_SEL para leer el byte inferior.
5) Esperar 2.8 microsegundos para cambiar la información.
6) Leer el byte inferior.
7) Leer nuevamente el byte inferior y comparar.
8) Si el byte inferior tiene un valor diferente, se irá a 6.
9) Establecer ENC_SEL para releer el byte superior.
10) Esperar 2.8 microsegundos para cambiar la información.
11) Releer el byte superior y comparar.
12) Si el byte superior tiene un valor diferente, se irá a 3.

Los pasos 6 y 7 pueden ser ejecutados en menos de 2 microsegundos, el índice máximo del codificador. El procedimiento total puede ser ejecutado en menos de 500 microsegundos, que es el tiempo que tomaría dar una vuelta al byte inferior en el índice máximo del codificador. En suma, el contador de 16 bit será enrollado automáticamente y deberá ser leído lo suficiente para detectar ciertas condiciones.

En general, la posición o velocidad del motor son controladas por el servo, es decir, el servo manda una señal para saber la posición actual del motor y sus posiciones posteriores, para ello usa un filtro de control calculando una salida que hará la diferencia de las posiciones, donde la posición error puede llegar a ser menor. Dos parámetros establecidos controlarán el movimiento del motor: los parámetros de la trayectoria deseada (posición meta, velocidad, aceleración), y los parámetros del filtro de control.

El filtro de control usado por el PIC-SERVO es un proporcional integral derivativo (Filtro PID). La salida del amplificador del motor es la suma de los tres componentes: uno proporcional para el error de posición, uno proporcional al cambio en el error de posición, y uno proporcional para lo acumulado en el error de posición.

El filtro de control PID opera en la posición de comando, y la posición actual de cada señal del servo produce una salida que se calcula de la siguiente manera:

Output=Kp(pos_error)-Kd(pos_error-prev_pos_error)+Ki(integral_error)

El término pos_error es simplemente la posición de comando actual menos la posición actual. El prev_pos_error es el error de

posición previo a la señal del servo. Kp, Ki y Kd están en el servo para ser programados y optimizar la ejecución de su motor particular.

El integral_error es la dirección total del pos_error dividido en 256. Para mantener el crecimiento del potencial integral_error, la dirección total es atado a un límite de integración específico. Por el establecimiento temporal la integración límite es cero.

El valor actual de salida PWM es (0-255) y la dirección del bit es dada por:

PWM = min(abs(output/256), output_limit)) – current_limit_adjustment

Dir = 0 if output>0, Dir = 1 if output < 0

La escala de salida PWM es limitada por un usuario definido output_limit. Por ejemplo, si alguien está usando un motor 12v por 24v, querrá establecer el output_limit a 255/2, ó 127. Además, el valor final de PWM es reducido por un current_limit_adjustment. Una operación normal es current_limit_adjustment=0. Si el motor actual está indicado por el valor de A/D y el usuario excede el límite especificado, current_limit_adjustment es incrementado a 1 por cada señal del servo, hasta un valor máximo de *min(abs(output/256), output_limit)*. Si el motor actual está bajo el límite especificado, current_limit_adjustment está en decremento por 1, bajo un mínimo valor de 0.

Este arreglo incremental es usado en vez de un arreglo proporcional debido a la no linealidad de algunos proyectos en progreso, y de hecho pueden ser usados con un amplificador externo que provee un valor binario.

La señal PWM tiene una onda cuadrada de 19.53 KHz de variación que funciona con un valor PWM de 255 correspondiente al 100% y un valor de 0 correspondiente a 0%.

Un último parámetro de control es el especificado por el usuario para el límite del error de posición. Si *abs(pos_error)* llega a ser más largo que este límite, la posición servo será invalidada. Esto es útil para desactivar automáticamente el servo ante una colisión o prueba de condición, dicha condición puede ser usada para establecer in límite intencional al motor.

La selección óptima de los parámetros de control PID puede ser hecha analíticamente o típicamente, ya que son elegidos a través de experimentos. El procedimiento puede ser el siguiente:

1) Primero se establece la posición Kp y la integral Ki para 0. Se conserva el incremento derivativo de Kd, hasta que el motor se activa y entonces retrocede un bit pequeño. El eje

del motor puede estar más lento cuando el valor Kd es incrementado.

2) Con el establecimiento del valor máximo de Kd, empieza a incrementarse Kp y las pruebas de movimiento del comando hasta que el motor comienza a dirigirse a la meta, es entonces cuando retrocede un bit pequeño. Los movimientos de prueba pueden ser pequeños movimientos con una aceleración y velocidad amplia, esto causará un perfil trapezoidal para brincar a la posición meta en una sola marca, dando la respuesta exacta del motor.

3) Dependiendo de la dinámica del sistema, el motor puede tener un estado de error estable con Kp y Kd se establecerá por encima. Si este es el caso, primero se establece un valor para IL de 16000 y entonces comienza a incrementarse el valor de Ki hasta que el estado de error estable es reducido a un nivel y tiempo aceptables. Al incrementarse Ki se introducirá alguna dirección en la posición. El mejor valor para Kp será algún acuerdo entre la dirección y el establecimiento del tiempo.

4) Finalmente, se reduce el valor de IL al mínimo y será cancelado cualquier estado de error estable.

El índice de del servo es aproximadamente de 2 KHz (1.953 KHz, para ser más exactos). Para un sistema con una combinación de inercia amplia, una disminución pequeña inherente y una resolución limitada del codificador, puede ser difícil tener una suficiente disminución en las velocidades bajas porque la interferencia en la digitalización con un valor muy alto de Kd causará la activación o vibración del servo. Afortunadamente, estos sistemas tienen una respuesta lenta y la velocidad del servo puede disminuir considerablemente.

En suma, se tienen un total de ocho parámetros del filtro de control: Posición (Kp), Derivación (Kd), Integral (Ki), Límite de Integración (IL), Límite de Salida (OL), Límite Actual (CL), Límite de Error de Posición (EL), y un Divisor de Velocidad del Servo (SR).

Las funciones de descripción de la señal trapezoidal y velocidad son utilizadas para generar automáticamente una trayectoria regular del motor, siempre y cuando se limite la aceleración y desaceleración del valor programado. Tanto la posición como la velocidad describen modos de operar y calcular dónde puede estar cada señal del servo para identificar la posición del comando actual para el motor. Cuando el filtro de control PID opera con esta posición de comando produce un valor de salida apropiado para PWM.

En el modo de posición, el motor sigue una trayectoria que es conocida como perfil trapezoidal. Cuando empieza un movimiento el motor acelerará al pico de velocidad programado con una aceleración constante, la cual también es programable. Cuando el motor alcance su destino con una máxima velocidad, empezará un proceso de desaceleración hasta lograr una aceleración constante. Cuando el motor alcanza la posición meta el motor controlador continuará al motor del servo para especificar la posición meta. Cuando un perfil de movimiento trapezoidal, el motor podrá empezar siempre desde una velocidad cero y el movimiento terminará en velocidad cero. El modo de posición cambia la velocidad, aceleración o la posición meta antes de que el motor haya alcanzado la posición meta original que causará un daño en el movimiento del motor o en el mecanismo conectado.

Para movimientos pequeños o movimientos con muy baja aceleración, el motor nunca podrá alcanzar el máximo de velocidad, para ello necesita desacelerar. En este caso, la velocidad contra el perfil de tiempo sería triangular en lugar de trapezoidal.

Si las velocidades o aceleraciones del motor necesitan ser dinámicamente cambiadas, el controlador podría ser operado en un modo de velocidad, en el cual la posición de comando se incrementa en cada marca del servo por el perfil de velocidad del generador. Si el motor es lento, la velocidad incrementará en la velocidad de la aceleración hasta alcanzar la velocidad meta. La velocidad meta y la aceleración pueden ser cambiadas en cualquier tiempo.

La posición, velocidad y aceleración están programadas en 32 bit cantidades en unidades del contador del codificador y señales del servo.

El estado del poder del motor (encendido o apagado) puede ser detectado por el suministro del motor conectado al pin ENC_SEL a través de la alta resistencia del divisor del voltaje. El valor del resistor podrá ser seleccionado para que el voltaje máximo del motor se aplique, el voltaje del pin no excederá 5v. Los resistores también podrán ser lo suficientemente grandes para cuando el PIC-SERVO este manejando ENC_SEL como una salida (0 – 5v), lo que nunca tendrá como origen o una inclinación de más de 10ma. Si el poder del motor no es el deseado se une ENC_SEL a +5v a través de un resistor 10k.

El PIC-SERVO indica al monitor automáticamente el estado del ENC_SEL como una entrada (cuando no se está leyendo el PIC-ENC) y deshabilita la posición del servo en la salida para el motor. Esto evita que el motor presente fallas y que deba ser restablecido sin que el host tenga algún conocimiento. El bit del motor del byte de estado indica el estado del motor, y el bit pos_error también

puede establecerse para indicar que el servo ha sido terminado. El host debe asegurarse de monitorear estos bits de estado antes de que inadvertidamente mande comandos de movimiento.

En las acciones de encendido o reinicio se establece lo siguiente:

1) El motor de posición se reinicia en cero.
2) Los valores de velocidad y aceleración son establecidos en cero.
3) Todos los parámetros adquiridos y los valores de límite son establecidos en cero.
4) El divisor de velocidad del servo es establecido en (1.953 KHz velocidad servo).
5) El valor PWM es establecido en cero.
6) El controlador es colocado en PWM y el modo AMP_EN esta establecido en una posición inferior.
7) El estado de la información está sólo en el byte de estado.
8) La dirección individual está establecida en 0x00 y la dirección grupal en 0xFF (grupo líder no establecido).
9) Las comunicaciones son invalidadas dependiendo de un valor inferior de ADDR_IN.
10) La velocidad del baudio está establecida en 19.2 KBaudios.
11) En el byte de estado, las banderas de move_done y pos_error serán establecidas y las banderas home_in_progress se limpiaran.
12) En el byte de estado auxiliar, las banderas pos_wrap, servo_on, accel_done, slew_done y servo_overrun se limpiaran.

2.1.3. Lista de comandos

Posición de reinicio

Valor de comando: 0x0
Numero de bytes de datos: 0
Byte de comando: **0x00**

Descripción:

Reinicia el contador del codificador de 32 bit en 0. También reinicia el comando interno desde la posición cero para prevenir que el motor salte si la posición servo está capacitada. No se debe utilizar este comando si se está ejecutando un movimiento con perfil trapezoidal.

Dirección establecida

Valor de comando: 0x1
Numero de bytes de datos: 2
Byte de comando: **0x21**
Bytes de datos:

1. Dirección individual: 0-0xFF (valor inicial 0x00)
2. Dirección grupal: (valor inicial 0xFF)

Descripción:

Establecer la dirección individual y grupal. Las direcciones grupales son interpretadas como el inicio entre 0x80 y 0xFF. Si un PIC-SERVO está en un grupo líder, se limpia el bit 7 de las direcciones grupales deseadas en la información del segundo bit; el PIC-SERVO establecerá automáticamente el bit 7 después de que se deshabilita al PIC-SERVO como líder del grupo. La primera vez que este comando es usado después de encender o reiniciar también se habilitará la comunicación para el siguiente módulo en la cadena de la red por debajo de la señal de ADDR_OUT.

Definición del estado

Valor de comando: 0x2
Numero de bytes de datos: 1
Byte de comando: **0x12**
Bytes de datos:

1. Punto de estado: (default: 0x00)
   Bit 0: envío de posición (4 bytes)
   1: Envío del valor A/D (1 byte)
   2: Envío de velocidad actual (2 bytes – ningún componente fraccionado) El entero con signo de 16 bit egresado es la velocidad negativa en unidades de cuenta por la señal servo.
   3: Envío del byte de estado auxiliar (1 byte)
   4: Envío de la posición origen (4 bytes)
   5: Envío del dispositivo ID, número de versión (2 bytes) (dispositivo PIC-SERVO ID=0)
   6,7: No usado – Limpiar para cero

Descripción:

Definir la información adicional que será enviada en el paquete de estado a lo largo del byte de estado. Establecer los bits en el primer byte de información lo que causará la correspondiente adición de bytes de información para después ser enviada al byte de estado. El estado de la información siempre será enviado en orden de listado.

Lector de estado

Valor de comando: 0x3

Numero de bytes de datos: 1
Byte de comando**:** **0x13**
Bytes de datos:

1. Punto de estado: (default: 0x00)
   Bit 0: Envío de posición (4 bytes)
   1: Envío del valor A/D (1 byte)
   2: Envío de velocidad actual (2 bytes – ningún componente fraccionado) El entero con signo de 16 bit egresado es la velocidad negativa en unidades de cuenta por la señal servo.
   3: Envío del byte de estado auxiliar (1 byte)
   4: Envío de la posición origen (4 bytes)
   5: Envío del dispositivo ID, número de versión (2 bytes) (dispositivo PIC-SERVO ID=0)
   6,7: No usado – Limpiar para cero

Descripción:

Esta es una versión no permanente de la definición de estado del comando. El paquete de estado retrocede en respuesta a este comando que incorporará la información de los bytes específicos, pero subsecuentemente el paquete de estado incluirá sólo la información de los bytes previamente especificada con el comando de definición de estado.

Cargar trayectoria

Valor de comando: 0x4
Numero de bytes de datos: *n* = 1-14
Byte de comando: **0x*n*4**
Bytes de datos:

1. Byte de control
   Bit 0: Carga de posición en la información (*n* +=4 bytes)
   1: Carga de velocidad en la información (*n* +=4 bytes)
   2: Carga de aceleración en la información (*n* +=4 bytes)
   3: Carga del valor de PWM (*n* +=1 bytes)
   4: Modo del servo – 0 = PWM mode, 1 = posición del servo
   5: Modo de perfil – 0 = perfil trapezoidal, 1 = descripción de velocidad
   6: En la velocidad y modo PWM: 0 = dirección FWD, 1 = dirección REV
   7: Inicio del movimiento

Descripción:

Todos los parámetros de movimiento son establecidos con este comando. El establecimiento de uno de los primeros cuatro bits en el control del byte requerirá una información de bytes adicional que son mandados para enumerarlos. La información de la posición

(rango1 +/- 0x7FFFFFFF) es usada sólo como la posición meta en el modo de perfil trapezoidal. La velocidad de la información (rango 0x00000000 a 0x7FFFFFFF) es usada como la velocidad meta en el modo de descripción de la velocidad o en la velocidad máxima en el modo de descripción trapezoidal. La información de la aceleración (rango 0x00000000 a 0x7FFFFFFF) es usada en el modo de descripción de la velocidad y trapezoidal. El valor de PWM (rango 0-0xFF), es usado sólo cuando la posición del servo no está operando, se envía un valor de PWM directamente al amplificador. El valor de PWM es reiniciado en 0 en alguna condición, la cual automáticamente incapacita la posición del servo. El bit 4 del byte de control especifica ya sea que la posición del servo deba ser usada o si el modo de PWM debe estar iniciado. El bit 5 especifica ya sea que el movimiento de la descripción trapezoidal deba estar iniciado o si la descripción de la velocidad es utilizada. Los movimientos del perfil trapezoidal sólo deben ser iniciados cuando la velocidad del motor marca 0. (El bit 0 del estado del byte indica cuando el movimiento del perfil trapezoidal esta completo, o el modo de velocidad, cuando el comando de velocidad ha sido alcanzado.) El bit 6 indica la velocidad o la dirección de PWM cuando la velocidad o modos de PWM están seleccionados. Si el bit 7 es establecido, el comando será ejecutado inmediatamente. Si el bit 7 esta limpio, el comando de información será amortiguado y será ejecutado cuando el movimiento de inicio del comando es usado.

Iniciar movimiento

Valor de comando: 0x5
Numero de bytes de datos: 0
Byte de comando: **0x05**

Descripción:

Este comando provoca que el motor ejecute el movimiento a la posición cargada previamente por otro comando. Este comando es muy útil ya que puede lograr que varios motores realicen un movimiento con un comando de grupo.

Estableciendo Parámetros de Control

Valor de comando: 0x6
Número de bytes de datos: 14
Byte de comando: **0xE6**
Bytes de datos:
1,2. Posición Kp (0 – 0x7FFF)

1 Mientras la posición puede ser el rango desde -0x7FFFFFFF a 0x7FFFFFFF, la posición meta no debe diferir de la posición actual por más de 0x7FFFFFFF.

3,4. Velocidad Kd (0 – 0x7FFF)
5,6. Posición Ki (0x – 0x7FFF)
7,8. Limite Integración IL (0 – 0x7FFF)
9. Limite de salida OL (0 – 0xFF)
10. Limite Actual CL (0 – 0xFF)
Valor impar: CUR_SENSE proporcional al actual del motor
Valor par: CUR_SENSE proporcional negativo al actual del motor
11, 12. Limite de error de posición EL (0- 0x3FFF)
13. Índice de división del servo SR (1 – 0xFF)
14. Compensación del amplificador (0 - 0xFF)

Descripción:

Establece todos los parámetros de control del servo controlador.

Alto al motor

Valor de comando: 0x7
Número de bytes de información: 1 a 5
Byte de comando: **0x17 ó 0x57**
Bytes de datos:
byte de control de alto
Bit 0: Habilita(1) o deshabilita(0) el amplificador
1: Apaga el motor
2: Detener abruptamente
3: Detener suavemente
4: Detener aquí (no disponible en versión 1)
5,6,7: Borrar a cero
2 – 5. Posición de alto (solo se requiere si el bit 4 se establece a 1)

Descripción:

Detiene el motor de la forma establecida.

Control I/O

Valor de comando: 0x8
Numero de bytes de datos: 1
Byte de comando: **0x18**
Bytes de datos:
1. Control del byte I/O:
Bit 0: valor de salida de Límite 1
1: Valor de salida de Límite 2
2: Dirección de Límite 1 (0 = output, 1 = input)
3: Dirección de Límite 2 (0 = output, 1 = input)
4,5,6,7: Limpiar para cero

Descripción:

Controla si las señales del limite 1 y 2 son entradas o salidas y establece el valor de salida.

Establecer modo de palpado

Valor de comando: 0x9
Numero de bytes de datos: 1
Byte de comando: **0x19**
Bytes de datos:

1. Control del byte de cabecera
   Bit 0: Captura la posición de palpado cuando el Límite 1 cambia.
   1: Captura la posición de palpado cuando el Límite 2 cambia.
   2: Apaga el motor en el palpado.
   3: Captura el palpado cuando cambia el Index.
   4: Detener abruptamente en el palpado.
   5: Detener suavemente en el palpado.
   6: Captura la posición de palpado cuando un error de exceso ocurre en la posición.
   7: captura la posición de palpado cuando el límite actual ocurre.

Descripción:

Causa que el controlador monitoree las condiciones específicas y captura la posición de palpado cuando uno de los abanderamientos ocurre.

Establecer los parámetros de velocidad de comunicación

Valor de comando: 0xA
Numero de bytes de datos: 1
Byte de comando: **0x1A**
Bytes de datos:

1. Divisor del índice de baudios, BRD
   Valores:

9600	BRD = 129
19200	BRD = 63
57600	BRD = 20
115200	BRD = 10

Descripción:

Establece el índice de comunicación en baudios.

Borrar los bits de error

Valor de comando: 0xB
Numero de bytes de datos: 0
Byte de comando: **0x0B**

Descripción:

Todos los bits de error o de evento permanecerán en el paquete de estado hasta que este comando se mande y se ejecute por el PIC.

Establecer la posición actual como cabecera

Valor de comando: 0xC
Numero de bytes de datos: 0
Byte de comando: **0x0C**

Descripción:

Establece la posición actual como posición de cabecera. Este comando normalmente se usa para un grupo de controladores.

No opera

Valor de comando: 0xE
Numero de bytes de datos: 0
Byte de comando: **0x0E**

Descripción:

No opera pero causa que un paquete de estado sea enviado por el módulo de control.

Reinicio completo

Valor de comando: 0xF
Numero de bytes de datos: 0
Byte de comando: **0x0F**

Descripción:

Reinicia el o los módulos de control al mismo estado de cuando se la aplica voltaje a los módulos. En este comando no se recibe ningún paquete de estado.

Byte de estado

Descripción:

Bit	Nombre	Definición
0	move_done	Cambia su valor a 0 cuando cambia la aceleración o velocidad del motor del motor.
1	cksum_error	Su valor es 1 si hubo un error en el checksum.
2	overcurrent	Su valor es 1 si existió una bandera límite.
3	power_on	Su valor es 1 solo si el voltaje del motor es mayor o igual a 12V.
4	pos_error	Su valor es 1 si el error de posición excede el límite.
5	limit1	Valor del interruptor de limite 1.
6	limit2	Valor del interruptor de limite 2.
7	home_in_progress	Su valor es 1 mientras se busca una posición de cabecera y 0 cuando la captura.

2.2. Adaptador para puerto serie

El adaptador para puerto serie utilizado para lograr la comunicación entre la computadora host y los módulos de control es el Z232-485 de la figura 2.4 [10].

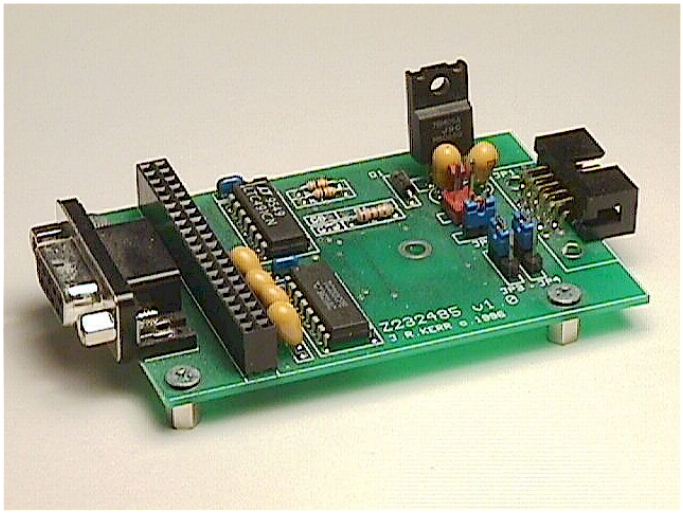


Figura 2.4. Adaptador para puerto serie.

Este convertidor de puerto serie es un adaptador full-duplex RS232-RS485, con un conector compatible al puerto COM de las PC y al conector NMC de la red de Módulos (PIC-SERVO). Este convertidor está diseñado para soportar hasta 32 módulos NMC con índices de velocidad desde 19,200 hasta 115,200 baudios.

Para realizar la conexión con la PC es necesario un cable con un conector hembra -macho DB9 para la conexión con la tarjeta Z232-485. Para la conexión de la tarjeta con los módulos de control es necesario un cable de 10 hilos plano IDC estándar tomando en cuenta que la longitud del cable no debe exceder los 3 metros de longitud.

Para utilizar este dispositivo como un convertidor convencional RS232, los jumpers JP3 y JP4 deben estar en la posición 1-2 para que las señales RS232 se conviertan en RS485 y viceversa.

La alimentación de voltaje del convertidor se logra mediante el cable plano de 10 hilos, para ello debe haber un jumper en el conector JP5, de otra forma, el convertidor debe ser alimentado por separado. En el caso de que los módulos de control NMC y el convertidor Z232-485 compartan el voltaje, debe también haber un conector en la posición del jumper JP6.

Aunque los módulos de control pueden comunicarse a índices de velocidad de 115,200 baudios, no todos los puertos PC COM lo hacen. Además, algunos procesadores no pueden soportar esta velocidad de datos y pueden perder información al momento de procesarla por lo que es recomendable establecer el índice a 19,200

baudios. Otro inconveniente es que algunos drivers de puertos COM no tienen la suficiente ampliación de señal para operar a 115,200 baudios. En la tabla 2.3 se presentan las definiciones de los pines de los conectores en la tarjeta Z232-485.

Conector de Network:
JP1

Pin	Definición
1	Z232-485 XMT +
2	Z232-485 XMT -
3	Z232-485 RCV +
4	Z232-485 RCV -
5	Sin conexión
6	GND
7	Vcc (7.5 - 12 vdc)
8	GND
9	Vcc (7.5 - 12 vdc)
10	GND

Conector de voltaje:
JP6

Pin	Definición
1	GND
2	7.5 - 12 vdc

Conector RS232: P1

Pin	Definición
1	Sin conexión
2	XMT
3	RCV
4	Sin conexión
5	GND
6	Sin conexión
7	Sin conexión
8	Sin conexión
9	Sin conexión

Jumpers:

Jumper	Descripción
JP3, JP4	Jumper en las posición 1-2 para uso normal Jumper en la posición 2-3 para uso con una tarjeta de procesador central (opcional)
JP5	Interconexión de voltaje. Insertando este conector, la tarjeta se

	conecta al voltaje de la red en el conector JP1.

Tabla 2.3. Conectores en el adaptador serie.

Para la conexión de múltiples controladores a una tarjeta adaptador puerto serie se sigue la configuración de la figura 2.5.

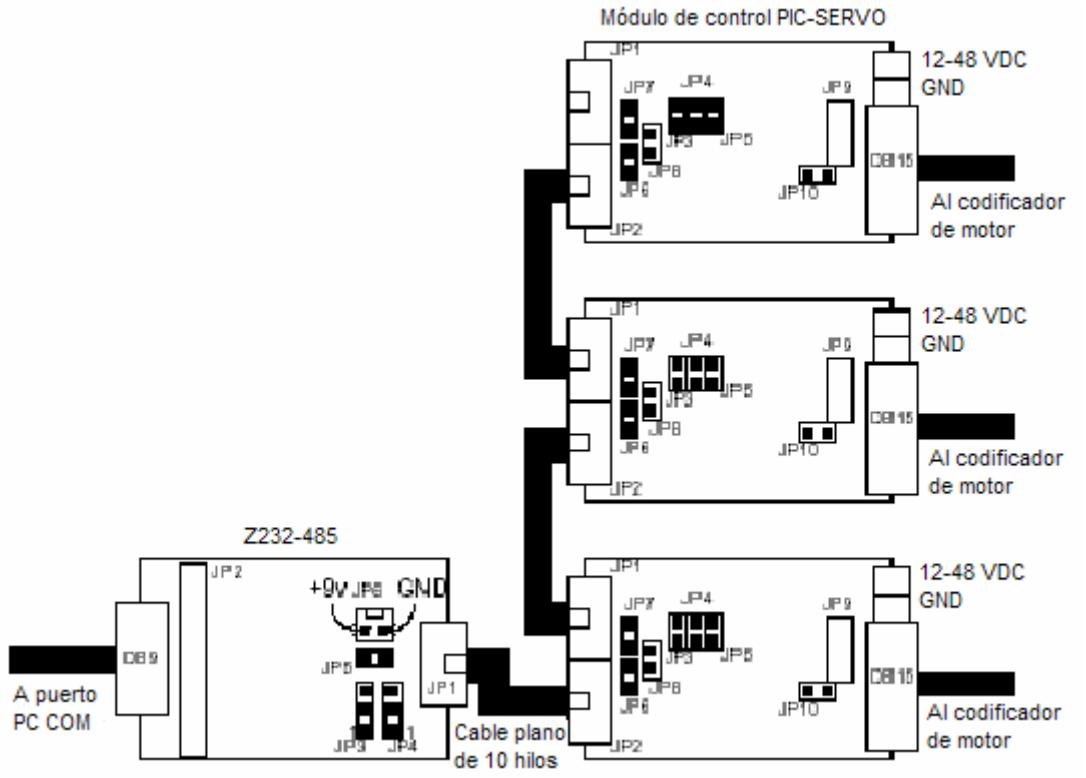


Figura 2.5. Conexión de múltiples controladores.

La figura 2.5 es un esquema general de la interconexión de tres módulos de control con un adaptador para puerto serie para el control de movimiento de los motores de los tres ejes de la máquina de medición por coordenadas.

Capítulo 3

Implementación para el control de movimiento

El sistema de medición por coordenadas y medición por contacto orientado a una MMC, es operado mediante un programa desarrollado específicamente para esta aplicación llamado “MMC Control”. El programa implementa la interfase de usuario y soporta los dispositivos periféricos que complementan el sistema con operación en tiempo real y mínima intervención por parte del usuario. En este capítulo se explicará el programa en dos enfoques fundamentales. El primero constituye un manual para el usuario que desea utilizar el sistema para la medición por contacto así como medición por posicionamiento. El segundo es una guía orientada al programador para facilitar el seguimiento del software que implementa el algoritmo. No se incluye el código fuente debido a su extensión. En su lugar, se describen las clases que conforman el proyecto de programación.

3.1. Introducción general

Los aspectos de interés en el control de movimiento de máquinas de múltiples ejes coordenados son:

- Procesamiento en tiempo real de algoritmos PID para el control de posición y velocidad usando perfiles de aceleración preferidos.
- Soporte para la programación de algoritmos específicos en la generación de perfiles de movimiento y ajuste de parámetros.

- Compatibilidad con transductores de posición del tipo codificador incremental lineal.
- Entradas y salidas digitales de propósito general.
- Soporte para el control de uno hasta 3 ejes coordenados.

En el esquema propuesto, figura 3.1, la planta representa uno de los múltiples ejes coordenados a controlar. El sistema electrónico realimentado descrito en la sección 2.1 implementa un algoritmo PID que genera tensiones de alimentación, $e(t)$ y procesa posiciones, $x(t)$, o velocidades, $v(t)$ para cerrar el lazo de control. Las entradas auxiliares, $p_i(t)$ i=1, 2, son utilizadas en la compensación de errores, principalmente en errores por pre-recorrido como se describió en la sección 1.3.2. Una computadora gobierna hasta tres sistemas electrónicos como el de la figura mediante un único enlace de comunicaciones RS232 descrito en la sección 2.2. En éste esquema, la computadora conforma un control maestro y los sistemas electrónicos constituyen controles esclavos. El programa en Visual C++ 6 gobierna los algoritmos de control de movimiento y delega a los sistemas electrónicos los algoritmos de control PID de posición o velocidad. El software en la PC proporciona parámetros que representan trayectorias de movimiento y el sistema electrónico las ejecuta.

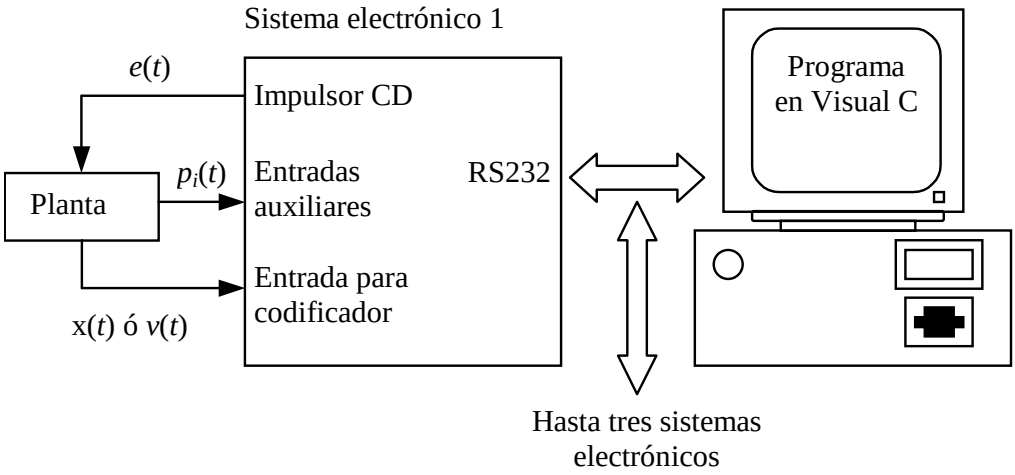


Figura 3.1. Sistema para el control de movimiento en máquinas con múltiples ejes coordenados.

Por otra parte, el software se desarrolló en el Centro de Ciencias Aplicadas y Desarrollo Tecnológico CCADET, por parte del Laboratorio de Metrología, con el objeto de crear un prototipo electrónico con movimiento automatizado para una maquina de medición por coordenadas. La plataforma de desarrollo utilizada fue Visual C++ 6.0, debido a la robustez que presenta este lenguaje para crear una interfase amigable tanto con el usuario final como con el

programador, además de que es posible interactuar directamente con dispositivos externos.

Algunas de las tareas principales del software son las siguientes:

- Medir objetos en tiempo real por posicionamiento.
- Medir objetos en tiempo real por el método de contacto.
- Controlar y registrar la posición del palpador del prototipo mecánico orientado a una máquina de medición por coordenadas.

Los requerimientos mínimos para ejecutar y compilar el programa son los siguientes:

- Computadora Pentium III, 64MB en RAM, 40GB HD, Puerto serie RS232, Windows 2000/XP.
- Joystick.
- Circuitos electrónicos para el control de movimiento.
- Fuentes de alimentación, 12V/4A, 5V/1A.
- Prototipo mecánico de máquina de un eje.
- Visual C++ 6.

3.2. Esquema utilizado

El prototipo de MMC desarrollado en el Laboratorio de Metrología del CCADET UNAM se adapta en forma no tan simple al esquema de la figura 3.1 como se muestra en la figura 3.2.

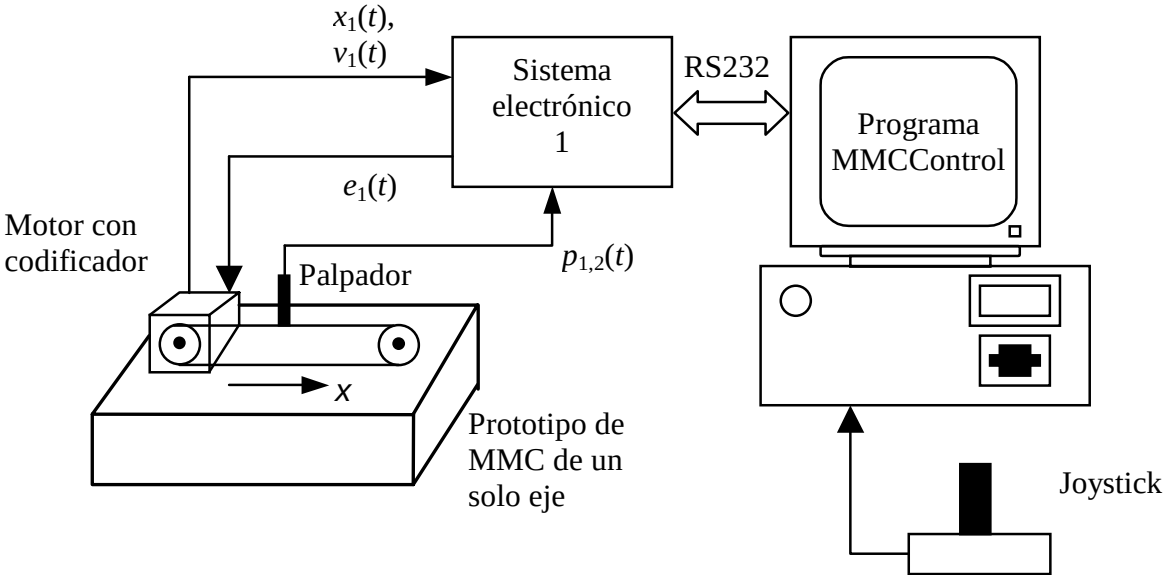


Figura 3.2. Esquema utilizado.

En la figura 3.2, se utiliza un sistema electrónico para el eje coordenado *x* del prototipo de MMC. Se utiliza una señal de tensión $e_1(t)$ para el motor CD que impulsa el palpador en el ejes coordenados *x*. La MMC genera una señale de posición $x_1(t)$ o velocidad $v_1(t)$ mediante un codificador ópticos incremental. Las dos señales $p_{1,2}(t)$ se utilizan para detectar el error de pre-recorrido en dos sentidos de palpación para el eje *x* positivo y negativo. Adicionalmente, el sentido de palpación es utilizado para compensar el error por radio del palpador, ya sea sumando o restando el radio a la medición registrada.

Dependiendo del modo de operación deseado en la MMC, se pueden presentar dos casos de control de movimiento:

- Control posicional. El operador desea posicionar la máquina espacialmente con fines de evaluación o de programación de rutinas repetitivas.
- Control de velocidad. Es utilizada con fines de medición por contacto con la pieza mecánica. En un principio el operador tiene el control de la máquina bajo el control de velocidad de desplazamiento. Posteriormente, al hacer contacto el palpador con la pieza de trabajo, se desarrolla un algoritmo en la computadora para incrementar la exactitud de la medición y disminuir la influencia natural del contacto mecánico al deformar la pieza de trabajo. El algoritmo fue descrito en la sección 1.3.2.

En la figura 3.2 una computadora controla los circuitos electrónicos para el control de movimiento descritos en el capítulo 2. Dicho enfoque ha demostrado su confiabilidad y seguridad en el control de movimiento de múltiples ejes coordenados. En la computadora reside el software desarrollado especificamente para esta aplicación. El software es el encargado de implementar la interfase de usuario para el control de movimiento y principalmente el algoritmo de medición por contacto descrito en la sección 1.3.2. Una interfase mediante Joystick le proporciona un elevado nivel de comodidad al usuario que controla el movimiento del prototipo mecánico.

El prototipo mecánico de la figura 3.3 se desarrolló sobre la base de las dos funciones sustanciales de cualquier MMC:

- Control de posición. Permitir el posicionamiento del palpador de dos formas: automática al especificar la posición objetivo y semi-automática mediante palancas de mando comandadas por el operador a una velocidad especifica. En el primer caso se requiere que el controlador sea puesto en modo posicional y en el segundo caso se requiere el modo de control de velocidad.

- Medición por contacto. Permitir la medición por contacto mecánico entre el palpador y la pieza de trabajo, como se describió en la sección 1.3.2.

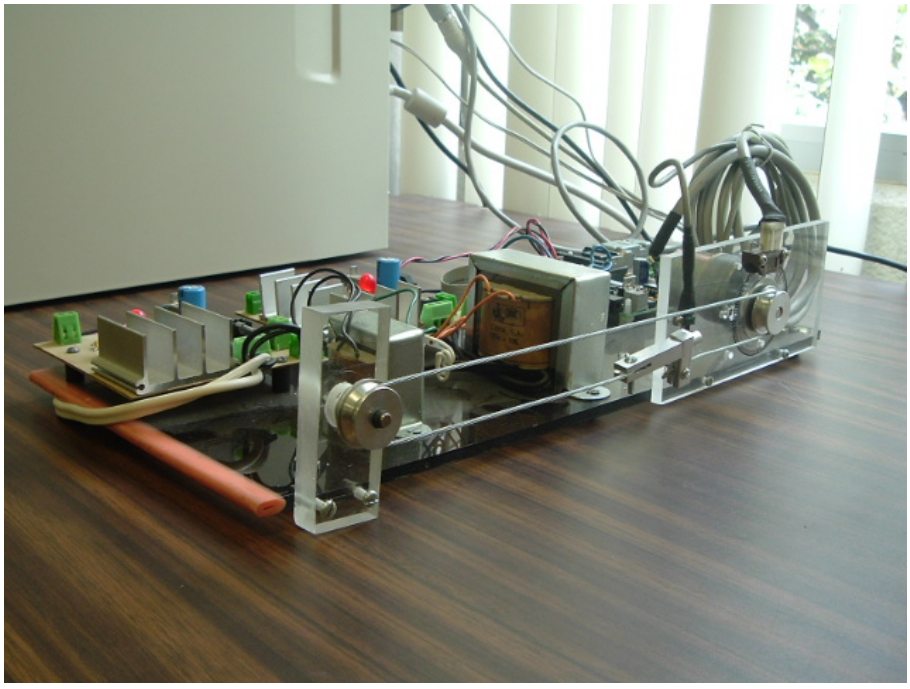


Figura 3.3. Prototipo mecánico.

En el esquema planteado, las características principales de la aplicación para PC que controla el movimiento son las siguientes:

Proyecto sobre la base de documento único

Únicamente es posible tener una ejecución del programa a la vez, debido a que solo se controla un instrumento de medición. La clase base es la Clase Forma en la cual se alojan todos los componentes del programa.

Uso de temporizadores de alta velocidad para el controlador PID posicional

Con el objeto de actualizar la posición del palpador sobre el eje coordenado, es necesario enviar una instrucción al controlador para que regrese el contador del PIC, esto se hace mediante el uso de un temporizador el cual ejecuta la instrucción cuatro veces por segundo y para así poder desplegarla en la aplicación y el usuario pueda ver el posicionamiento del palpador sobre el eje.

Uso de controles Active-X

Para realizar una comunicación con dispositivos externos es necesaria la implementación de este tipo de controles, ya que la

comunicación se hace mediante el puerto de comunicaciones RS232 y los controles son relacionados para interactuar enviando y recibiendo datos por este puerto.

Manejo de variables del tipo VARIANT y SAFEARRAY

Debido a que se emplea el puerto de comunicaciones, los datos son enviados en forma serial; esto provoca que, tanto las instrucciones como los datos recibidos deben ser encapsulados en arreglos del tipo SAFEARRAY y el tipo de variable debe ser VARIANT ya que este tipo de variable es muy robusta y puede ser convertida y manejada según las necesidades del programador.

3.3. Descripción en el ámbito del operador

La siguiente descripción tiene por objeto orientar al usuario final en la operación del software de medición. Se describen las características de la aplicación, su principal función y el modo correcto para su uso. La interfase de usuario presenta varios componentes para realizar el movimiento del eje coordenado. A continuación se explica cada a uno a detalle.

3.3.1. Pantalla principal

En la pantalla principal, como se muestra en la figura 3.4, aparecen los siguientes elementos: en la parte superior un menú de opciones con tres de las opciones más comunes, dos contenedores que agrupan funciones de medición, una caja de texto mostrando la posición actual del mecanismo y un área de datos recibidos por el puerto serie utilizada con fines de depuración.

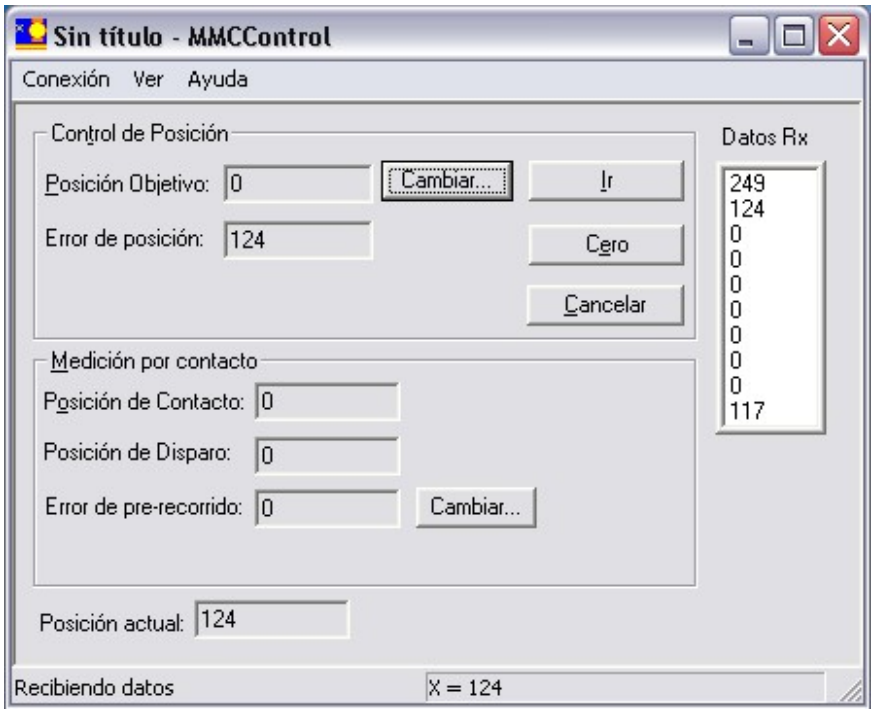


Figura 3.4. Pantalla principal.

La totalidad de las opciones del programa son agrupadas en un menú con las siguientes tres categorías.

Conexión. Corresponde a la función para salir de la aplicación, figura 3.5.

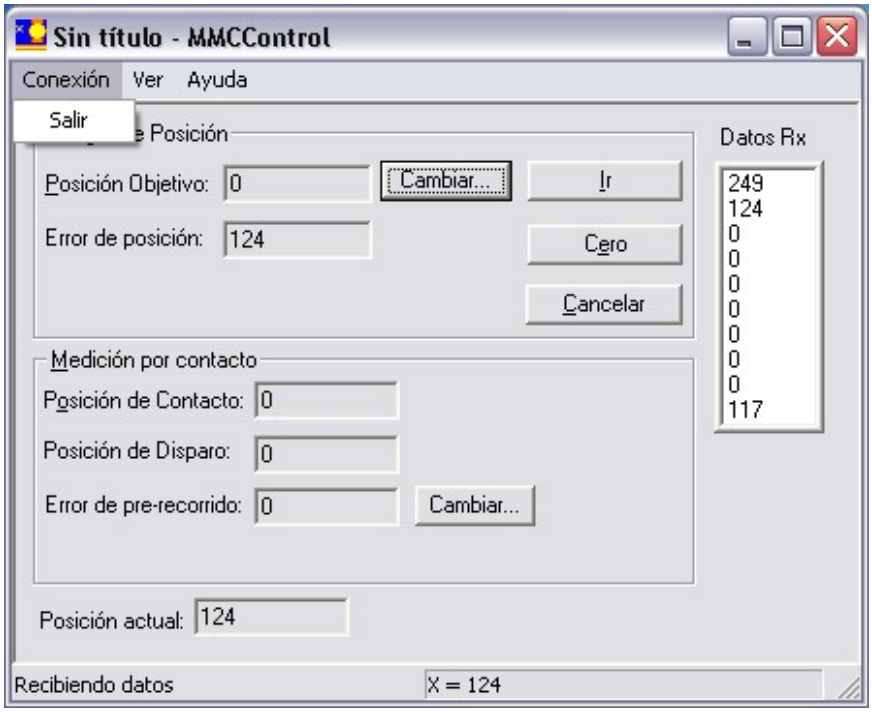


Figura 3.5. Función del menú “Conexión”.

Ver. Simplemente habilita y deshabilita la barra de estado que muestra la posición actual del mecanismo, figura 3.6.

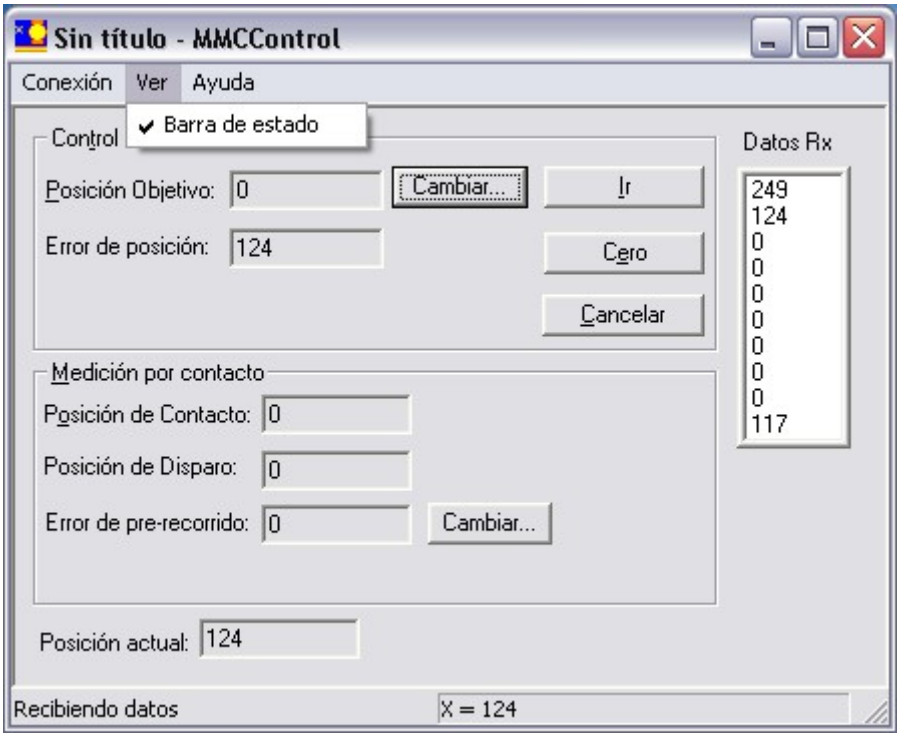


Figura 3.6. Función del menú “Ver”.

Ayuda. Muestra la caja de diálogo “Acerca de...”, figuras 3.7 y 3.8.

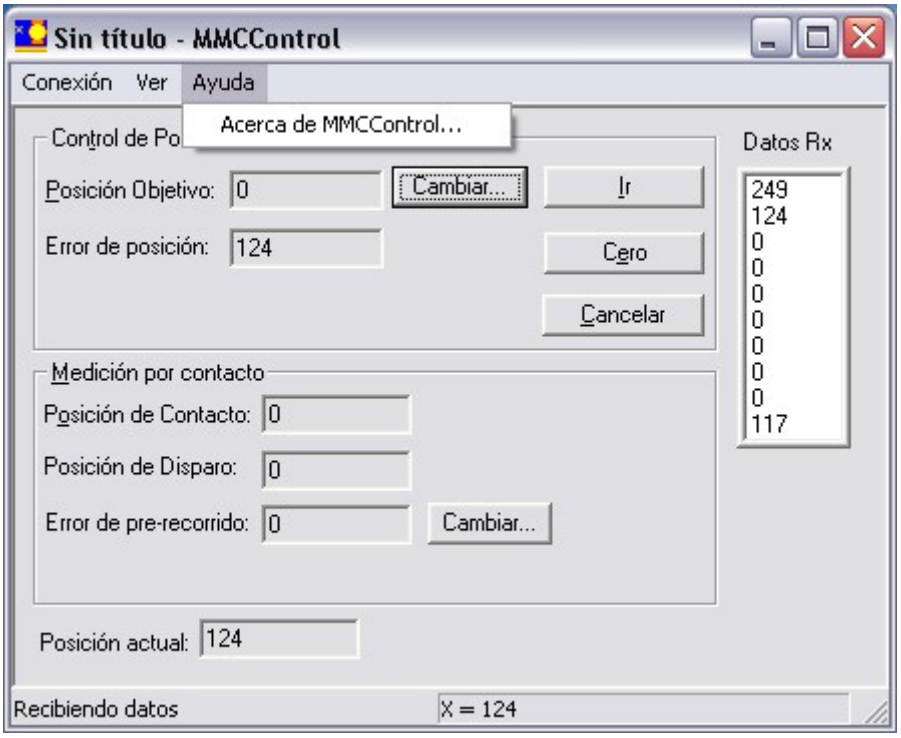


Figura 3.7. Funciones del menú “Ayuda”.

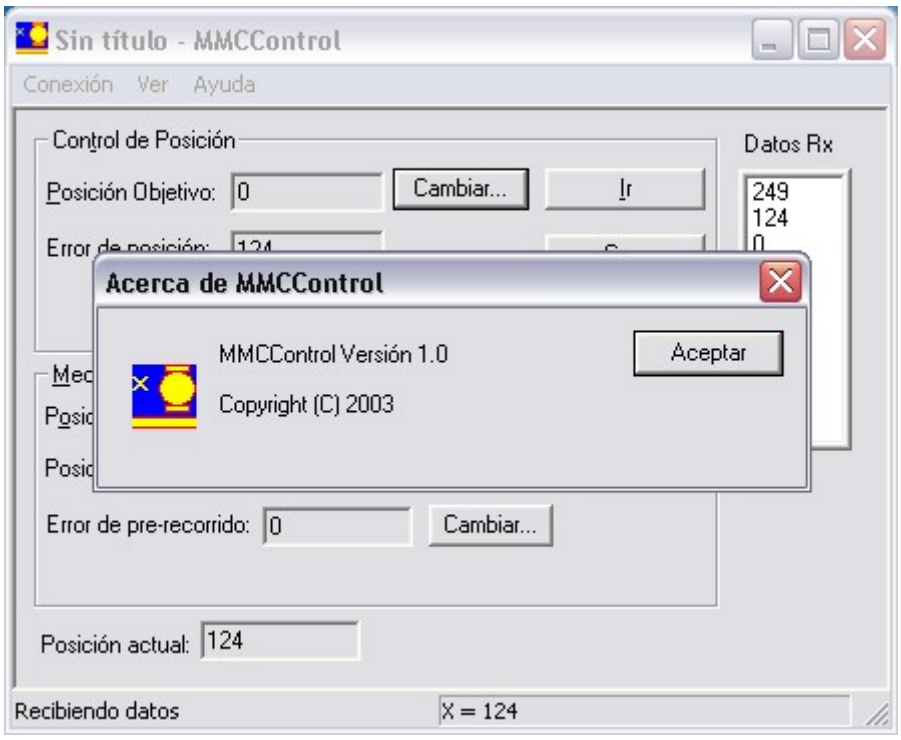


Figura 3.8. Caja de diálogo “Acerca de...”.

La aplicación MMCControl contiene dos contenedores principales llamados: “Control de posición” y “Medición por contacto”.

3.3.2. Contenedor “Control de posición”

Este contenedor (figura 3.9) almacena 4 botones para mover el palpador a una posición específica, es decir, realizar una medición por posicionamiento.

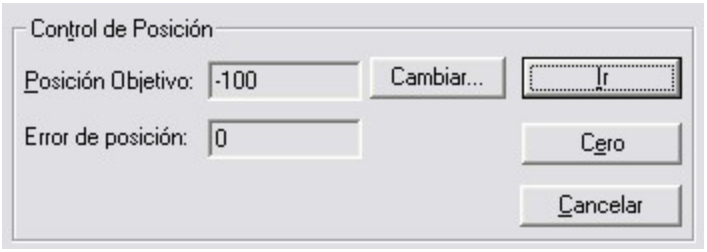


Figura 3.9. Contenedor “Control de posición”.

Para llevar el palpador a una posición deseada, es necesario oprimir el botón “Cambiar..”, al oprimirlo aparecerá una caja de diálogo como lo muestra la figura 3.10. Se inserta la posición objetivo deseada y se oprime el botón “OK”.

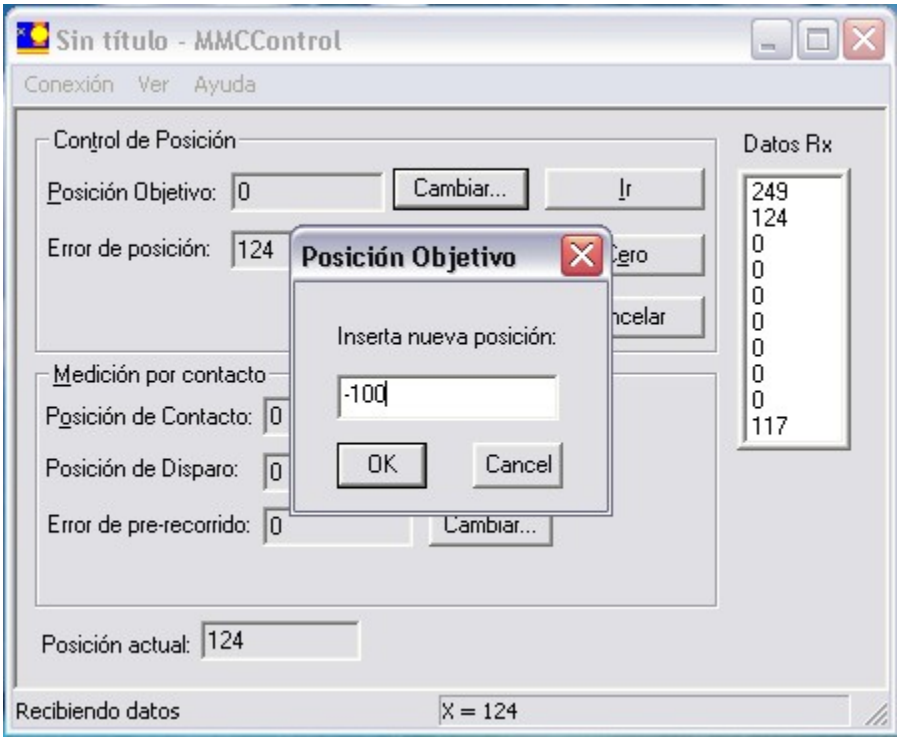


Figura 3.10. Caja de dialogo para cambiar “Posición Objetivo”.

Una vez que en la caja de texto “Posición objetivo” se tiene la posición deseada, se oprime el botón “Ir” y el motor comienza con el movimiento.

El botón “Cero” cambia la posición actual a cero. El botón “Cancelar” detiene el movimiento del motor una vez que este se esta realizando.

En la caja de texto “Error de posición” aparecerá un valor cuando el motor no llegue a la posición objetivo deseada, y marcará solo la diferencia entre la posición objetivo deseada y la posición actual.

3.3.3. Contenedor “Medición por contacto”

Este contenedor (figura 3.11) almacena un botón para cambiar el error de pre-recorrido en el palpador. La información que despliega el contenedor es útil cuando se hace contacto con el objeto a medir, es decir, realizar una medición por contacto como se describió en la sección 1.3.2.

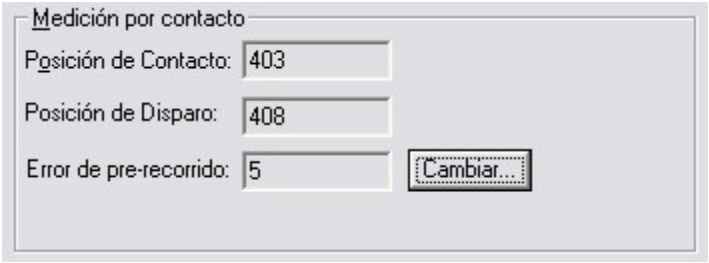


Figura 3.11. Contenedor "Medición por contacto".

Para mover el palpador en una dirección positiva o negativa, puede lograrse con los botones del Joystick, direccionando a la derecha o a la izquierda según la dirección y velocidad deseadas.

Al realizar una medición por contacto tenemos la opción de introducir un valor específico con el objeto de hacer una comparación entre medidas específicas y medidas reales. Es necesario introducir el error de pre-recorrido, esto se hace con el botón "Cambiar" que despliega la caja de diálogo de la figura 3.12.

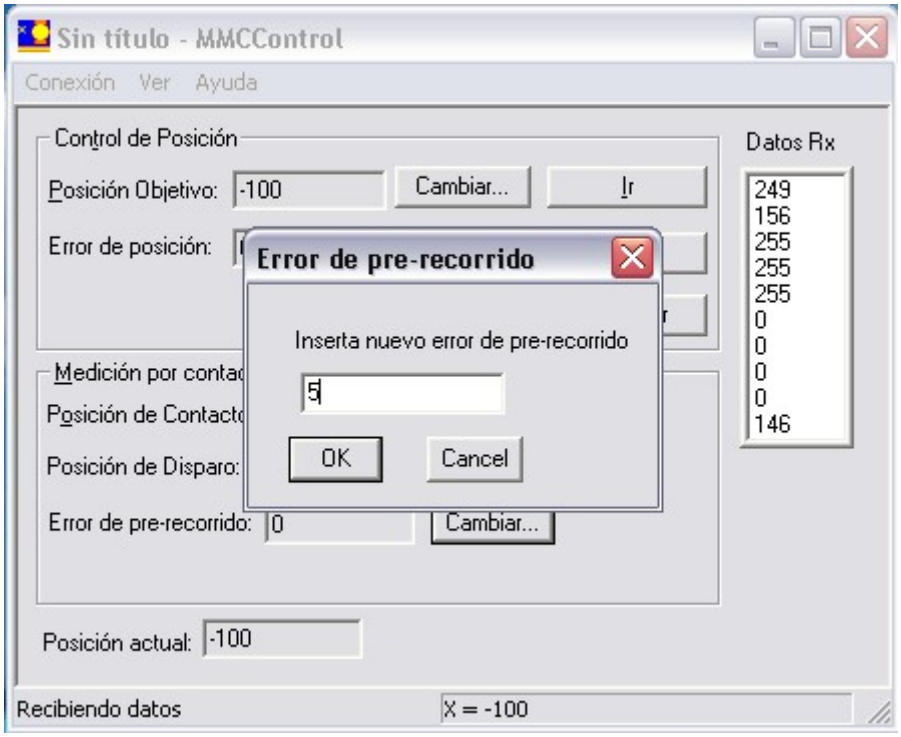


Figura 3.12. Caja de dialogo para cambiar "Error de pre-recorrido".

En la caja de texto "Posición de disparo" aparecerá el valor de la posición cuando el palpador haya cerrado completamente el circuito eléctrico de disparo. En la caja de texto "Posición de contacto" se visualizará la diferencia entre la posición de disparo y el error de pre-recorrido, como se describió en la sección 1.3.2.

La caja de texto posición actual muestra la posición en la que el motor desplaza al palpador y esta se actualiza cuatro veces por segundo.

Por último, la sección “Datos Rx” fue utilizado con fines de depuración. Aquí se muestra el conjunto de bits que recibe el software de los circuitos electrónicos para el control de movimiento.

3.4. Descripción en el ámbito del programador

La figura 3.13 muestra las ocho clases en las cuales se organizó el proyecto de programación. El proyecto se construyó sobre la base de la arquitectura documento-vista, utilizando la MFC y el asistente para la creación de proyectos, APP Wizard de Visual C++ 6.0 [13, 20].

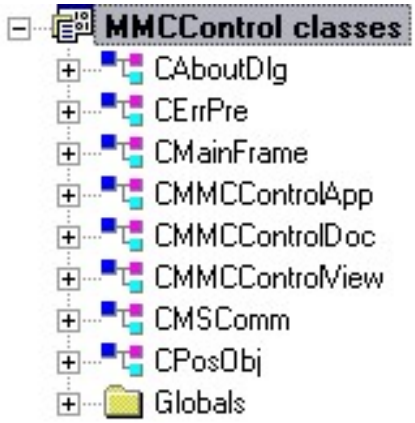


Figura 3.13. Clases empleadas en el software desarrollado.

La siguiente descripción muestra las clases generadas por omisión y creadas especificamente para la aplicación mediante el asistente para proyectos nuevos de Visual C++ 6.0. Las clases soportan principalmente la arquitectura documento vista y la apariencia final de la aplicación.

3.4.1. Clase CAboutDlg

La clase CAboutDlg, figura 3.14, es utilizada para desplegar la caja de diálogo “Acerca de”, que contiene información breve del programa. Los métodos se describen a continuación:

Figura 3.14. Clase CAboutDlg.

CAboutDlg(); Es el constructor por omisión de la clase. No realiza inicializaciones adicionales.

virtual void DoDataExchange(CDataExchange* pDX); Permite el intercambio de datos con otras clases.

3.4.2. Clase CErrPre

La clase CErrPre, figura 3.15, es la encargada de obtener la variable del error de pre-recorrido que introducirá el usuario para que la aplicación procese ese dato.

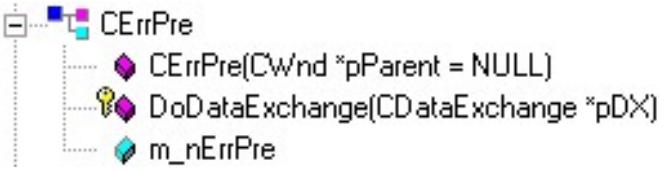


Figura 3.15. Clase CErrPre.

CErrPre(CWnd *pParent = NULL); Es el constructor por omisión de la clase. No realiza inicializaciones adicionales.

virtual void DoDataExchange(CDataExchange *pDx); Aquí se aloja la construcción de los controles de la aplicación (botones y caja de texto).

3.4.3. Clase CMainFrame

La clase CMainFrame es la clase principal utilizada para el despliegue de la aplicación en forma de caja de diálogo, figura 3.16. Los métodos se describen a continuación.

Figura 3.16. Clase CMainFrame.

virtual void AssertValid() const; Valida la clase. Manda al método AssertValid() de la clase CFrameWnd.

CMainFrame(); Es el constructor por omisión.

virtual ~CMainFrame(); El destructor de la clase.

virtual void Dump(CDumpContext& dc) const; Libera el contexto de dispositivo.

afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct); Es un prototipo de manejador de mensajes al crear la estructura.

virtual void OnUpdateLeft(CCmdUI *pCmdUI); Hace modificaciones a las propiedades de la ventana.

virtual OnUpdateViewStatusBar (CCmdUI* pCmdUI); Hace visible las actualizaciones en la barra de estado de la aplicación.

virtual OnViewStatusBar(); Hace visible la barra de estado de la aplicación.

virtual BOOL PreCreateWindow(CREATESTRUCT& cs); Hace modificaciones a las propiedades de la ventana.

3.4.4. Clase CMMCControlApp

La clase CMMCControlApp, figura 3.17, es la encargada de dar la apariencia tridimensional a los objetos y controles que se alojan en la aplicación.

Figura 3.17. Clase CMMCControlApp.

CMMCControlApp(); Es el constructor por omisión de la aplicación. No realiza inicializaciones adicionales a menos que se edite la función.

virtual BOOL InitInstance(); Realiza las siguientes inicializaciones estándar. Inicia la apariencia 3D de los controles. Habilita la conexión entre el documento y sus vistas.

afx_msg void OnAppAbout(); Constituye el procedimiento para atender el evento por desplegar la caja de diálogo “Acerca de...”.

3.4.5. Clase CMMCControlDoc

La clase CMMCControlDoc, figura 3.18, se encarga de la forma de almacenar los datos en archivos, debido a que el programa únicamente controla el movimiento de los motores, no es necesario almacenar datos adicionales.

Figura 3.18. Clase CMMCOntrolDoc.

virtual void AssertValid() const; Valida el documento del programa.

CMMCControlDoc(); Es el constructor por omisión de la clase. Inicializa el formato de archivo para los datos empleados en el documento.

virtual ~CMMCControlDoc(); Es el destructor de la clase.

virtual void Dump(CDumpContext& dc) const; Vacía la estructura de contexto del dispositivo dc.

virtual BOOL OnNewDocument(); Inicializa los datos para crear una nueva estructura del documento para almacenar.

virtual void Serialize(CArchive& ar); Permite el almacenamiento de los datos de memoria con el formato de archivo empleado.

3.4.6. Clase CMMCControlView

La clase CMMCControlView, figura 3.19, presenta los métodos empleados para la operación y control de velocidad y posición del motor. En esta clase es donde se alojan las funciones programadas para el funcionamiento del software. Es a clase principal del programa.

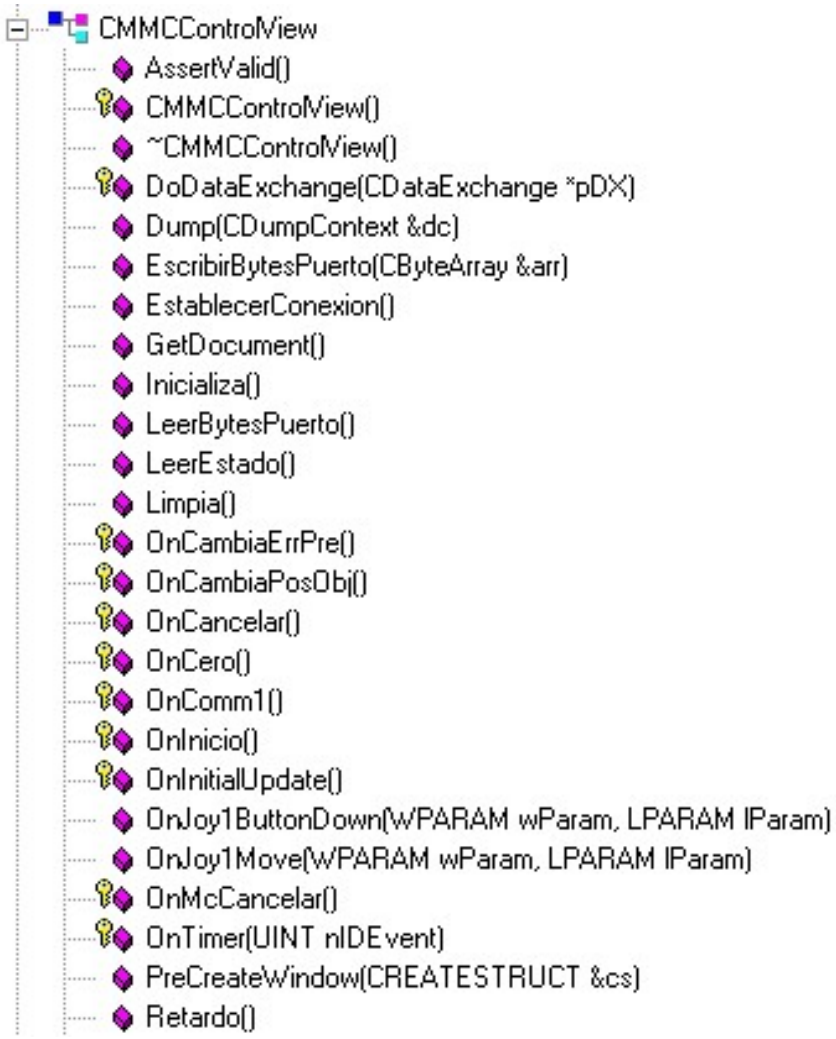


Figura 3.19. Clase CMMCControlView.

virtual void AssertValid() const; Valida la clase, llama a el método de CView AssertValid().

CMMCControlView(); Es el constructor por omisión de la clase. No realiza inicializaciones adicionales.

virtual ~CMMCControlView(); Es el destructor de la clase.

virtual void DoDataExchange(CDataExchange* pDX); Aquí se aloja la construcción de los controles de la aplicación (botones, cajas de texto).

virtual void Dump(CDumpContext& dc) const; Libera el contexto del dispositivo dc.

virtual void EscribirBytesPuerto(CByteArray &arr); Escribe los bytes en el buffer de salida para el puerto de comunicaciones.

virtual void EstablecerConexion(); Abre el puerto de comunicaciones y establece los parámetros de transmisión y recepción de datos.

CMMCControlDoc* GetDocument(); Permite el intercambio de datos que pertenecen al documento de mediciones con otras clases.

virtual void Inicializa(); inicializa y establece los parámetros de la tarjeta controladora del motor.

virtual void LeerBytesPuerto(); Obtiene los bytes del buffer de entrada del puerto de comunicaciones.

virtual void LeerEstado(); Obtiene la posición actual del motor del contador de la tarjeta controladora.

virtual void Limpia(); Limpia las colas de recepcón y transmisión del buffer de comunicaciones.

virtual void OnCambiaErrPre(); Muestra la caja de dialogo para introducir el error de pre-recorrido.

virtual void OnCambiaPosObj(); Muestra la caja de diálogo para introducir la posición objetivo.

virtual void OnCancelar(); Detiene el movimiento del motor cuando se controla la posición.

virtual void OnCero(); Establece los valores de las cajas de texto a cero y envia el comando Reset al controlador electrónico.

virtual void OnComm1(); Determina que evento ocurre en el puerto de comunicaciones y relaciona las funciones a ello.

virtual void OnInicio(); En el contenedor Control de Posición, inicia el movimiento del motor a la posición objetivo introducida previamente.

virtual void OnInitialUpdate(); Establece las características de una aplicación al iniciarse esta, abre el puerto de comunicaciones y verifica si existe algún joystick instalado.

virtual LRESULT OnJoy1ButtonDown(WPARAM wParam, LPARAM lParam); para el motor cuando se ha oprimido el botón número 1 del joystick.

virtual LRESULT OnJoy1Move(WPARAM wParam, LPARAM lParam); mueve el motor a una dirección u otra y con velocidades de palpado.

virtual void OnMcCancelar(); Detiene el movimiento del motor cuando se esta midiendo por contacto.

virtual void OnTimer(UINT nIDEvent); Se establece un temporizador de 250 milisegundos para actualizar la posición actual del contador del codificador.

virtual BOOL PreCreateWindow(CREATESTRUCT& cs); Llama al método de la clase CView PreCreateWindow, para poder crear la estructura de la ventana.

virtual void Retardo(); Establece un retardo al inicializarse el programa para que los datos sean procesados en la tarjeta controladora

3.4.7. Clase CMSComm

Esta clase es predefinida por Visual C++ como librería de métodos para el control de comunicaciones por el puerto serie, figura 3.20. En los métodos que aloja esta clase se encuentran los que envían y reciben datos del buffer del puerto de comunicaciones. Asimismo se encentran las funciones que logran la configuración del puerto, esto es, velocidad de envío, tamaño de bloque de datos, paridad, y la serie de protocolos que existen para lograr la comunicación entre dos dispositivos.

Figura 3.20. Clase CMSComm.

void SetCDHolding(BOOL bNewValue); Establece el estado de la línea portadora, false para desactivar y true para activar.

BOOL GetCDHolding(); obtiene el estado de la línea portadora con las mismas condiciones anteriores

void SetCommID(long nNewValue); Establece una identificación al Puerto de comunicaciones mediante un número.

long GetCommID();Obtiene el valor de la identificación del puerto de comunicaciones.

void SetCommPort(short nNewValue); Establece que Puerto de comunicaciones es el que se va a emplear, depende del equipo y características de la pc.

short GetCommPort(); Obtiene cual es el puerto de comunicaciones que se esta empleando.

void SetCTSHolding(BOOL bNewValue); Establece la propiedad de comunicaciones Clear to Send para poder enviar información (true) o no (false).

BOOL GetCTSHolding(); Obtiene, sí la propiedad Clear to Send esta activa o no.

void SetDSRHolding(BOOL bNewValue); Establece el estado de la propiedad Data Set Ready, false para desactivar, true para activar.

BOOL GetDSRHolding(); Obtiene el estado de la propiedad Data Set Ready.

void SetDTREnable(BOOL bNewValue); Establece el estado de la propiedad de comunicaciones Data Terminal Ready, false para desactivar, true para activar.

BOOL GetDTREnable();Obtiene el estado de la propiedad Data Terminal Ready.

void SetHandshaking(long nNewValue); Establece el valor del protocolo de hardware “saludo” para la comunicación con otro dispositivo.

long GetHandshaking(); Obtiene el valor del protocolo del “saludo”

void SetInBufferSize(short nNewValue); Establece el tamaño de las colas de recepción.

short GetInBufferSize(); Obtiene el tamaño de las colas de recepción.

void SetInBufferCount(short nNewValue); Establece el tamaño de los datos en bytes que se encuentran en la cola de recepción.

short GetInBufferCount(); Obtiene el tamaño de los datos en bytes que se encuentran en la cola de recepción.

void SetBreak(BOOL bNewValue); Establece o borra el estado de la señal break

BOOL GetBreak(); Obtiene el estado de la señal break.

void SetInputLen(short nNewValue); Establece el número de caracteres leídos en el buffer de recepción.

short GetInputLen(); Obtiene el número de caracteres leídos en el buffer de recepción.

void SetNullDiscard(BOOL bNewValue); Determina si caracteres nulos han sido enviados desde el Puerto al buffer de recepción.

BOOL GetNullDiscard(); Obtiene si han sido recibidos caracteres nulos en el buffer de recepción.

void SetOutBufferSize(short nNewValue); Establece el tamaño de las colas de transmisión.

short GetOutBufferSize(); Obtiene el tamaño de las colas de transmisión.

void SetOutBufferCount(short nNewValue); Establece el tamaño de los datos en bytes que se encuentran en la cola de transmisión.

short GetOutBufferCount(); Obtiene el tamaño de los datos en bytes que se encuentran en la cola de transmisión.

void SetParityReplace(LPCTSTR lpszNewValue); Establece el caracter que reemplazará a un caracter no válido cuando ocurre un error de paridad.

CString GetParityReplace(); Obtiene el caractér que reemplaza a otro no válido cuando ocurre un erro de paridad.

void SetPortOpen(BOOL bNewValue); Abre (true) o cierra (false) el puerto de comunicaciones.

BOOL GetPortOpen(); Obtiene el estado del Puerto de comunicaciones.

void SetRThreshold(short nNewValue); Establece el número de caracteres a recibir antes de que se genere un evento OnComm.

short GetRThreshold(); Obtiene el número de caracteres a recibir antes de que se genere un evento OnComm.

void SetRTSEnable(BOOL bNewValue); Establece el valor de la propiedad Request to Send.

BOOL GetRTSEnable(); Obtiene el valor de la propiedad Request To Send.

void SetSettings(LPCTSTR lpszNewValue); Establece los valores de baudios, paridad, bit de dato y bit de parada.

CString GetSettings(); Obtiene el valor de los baudios, paridad, bit de dato y bit de parada.

void SetSThreshold(short nNewValue); Establece el número de caracteres a enviar antes de que se genere un evento OnComm.

short GetSThreshold(); Obtiene el número de caracteres a enviar antes de que se genere un evento OnComm.

void SetOutput(const VARIANT& newValue); Escribe un flujo de datos en el buffer de transmisión.

VARIANT GetOutput(); Obtiene un flujo de datos del buffer de transmisión.

void SetInput(const VARIANT& newValue); Escribe un flujo de datos en el buffer de recepción.

VARIANT GetInput(); Obtiene un flujo de datos del buffer de recepción.

void SetCommEvent(short nNewValue); Establece que evento realiza el control de comunicaciones.

short GetCommEvent(); Obtiene el evento que realiza el control de comunicaciones.

void SetEOFEnable(BOOL bNewValue); Establece si el control de comunicaciones busca un caracter de EndOfFile antes de generar el evento CommEvEOF.

BOOL GetEOFEnable(); Obtiene el estado de la propiedad EOFEnable.

void SetInputMode(long nNewValue); Establece el valor del tipo de datos que se reciben. 0 para recibir datos como texto, 1 para recibir datos como datos binarios.

long GetInputMode(); Obtiene el valor de la propiedad InputMode.

3.4.8. Clase CPosObj

La clase CposObj, figura 3.21, es la encargada de obtener la variable de la posición objetivo que introducirá el usuario como en la posición de contacto, para que la aplicación procese ese dato.

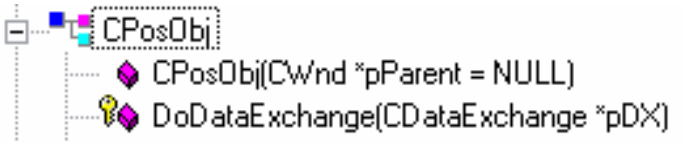


Figura 3.21. Clase CPosObj.

Las clases anteriores se describen de forma breve así como las funciones que en ellas se desarrollan para el funcionamiento de la aplicación. Asimismo se describe el tipo de datos que cada función recibe como parámetro para su proceso.

CPosObj(CWnd *pParent = NULL); Es el constructor por omisión de la clase. No realiza inicializaciones adicionales.

virtual void DoDataExchange(CDataExchange *pDx); Aquí se aloja la construcción de los controles de la aplicación (botones y caja de texto).

Capitulo 4

Resultados y conclusiones

En el capítulo tres se describió el sistema propuesto para el control de movimiento utilizando circuitos electrónicos comerciales, un prototipo de Máquina de Medición por Coordenadas, MMC, de un solo eje y software desarrollado en Visual C++ 6, lo cual proporciona una idea cualitativa de su desempeño. En el presente capitulo describimos resultados cuantitativos obtenidos en el proyecto. No obstante, el desempeño global del sistema debería ser evaluado en una MMC real. Sin embargo, debido a la complejidad en el diseño mecánico de una MMC con palpador de contacto, nos limitamos a evaluar el desempeño en un prototipo que consideramos no esta muy alejado de una situación real. Los aspectos de diseño mecánico no son tema del presente trabajo de tesis ya que sobrepasan nuestras capacidades y objetivos.

Se presentan también las conclusiones finales y una serie de propuestas que pueden orientar el trabajo a futuro con el objetivo de realizar mejoras al sistema.

4.1. Resultados experimentales

En el Laboratorio de Metrología del CCADET se construyó una maquina de medición por coordenadas para realizar experimentos de medición, la cual se espera que en el corto plazo se perfeccione con el apoyo de estudiantes y académicos con el objeto de hacer de ella una máquina que cumpla con las características esenciales, funciones necesarias y un probable mejor rendimiento comparada con una máquina comercial.

Es importante resaltar que seguido de la planeación, diseño y construcción física de la MMC, se desarrolló una aplicación de software que perseguía el fin de lograr la realización de mediciones,

sin embargo, estaba latente un inconveniente, la máquina no estaba automatizada y el usuario se veía en la necesidad de posicionar el palpador de la máquina de manera manual en cada uno de los tres ejes, lo que daba como resultado que la aplicación determinara entonces la posición inexacta del palpador de la MMC.

Ante estas circunstancias nace la idea de la construcción de un prototipo en un eje coordenado para dar el principio de la automatización en el movimiento, es decir, el usuario mediante un joystick comercial y un equipo de cómputo posicionaría el palpador de la máquina, evitando así hacerlo de forma manual y obteniendo una indicación de posición más exacta, llevando a cualquier posición dentro del rango de la MMC el palpador de la máquina con una división de 2μm.

La automatización de la MMC, trajo consigo la necesidad de construir un prototipo para simular el movimiento en un solo eje coordenado (figura 4.1). El prototipo se conformó por diversos dispositivos como lo son: un circuito electrónico para el control PID de movimiento en motores CD (PIC-SERVO Motor Control Board), un circuito electrónico para la interconexión serie RS232 (Z232-485 Serial Port Converter), un motor CD con reducción 12V 300rpm 0.8kg-cm, un codificador óptico angular incremental de 400 cuentas/revolución, un sistema de dos interruptores para el disparo de la medición en un eje, además de la construcción de dos fuentes de voltaje de alimentación 12V/3A 9V/1A, y el desarrollo del software de operación realizado en Visual C++.

Figura 4.1. Prototipo experimental.

El prototipo de máquina con un eje coordenado tuvo la ventaja de permitir la depuración del algoritmo, y de esta manera orientar los diseños de los mecanismos en la MMC.

Este prototipo fue puesto a prueba en diversas ocasiones bajo dos experimentos distintos para comprobar que la eficiencia de cada módulo programado de la aplicación fuera la esperada, esto, para comprobar las capacidades en el control del movimiento y posicionamiento dentro del mismo eje coordenado.

El primer experimento tuvo como objetivo obtener la mejor referencia posible del control de posición y así obtener los valores de control PID, mientras que el segundo experimento tuvo como objetivo probar que la medición por contacto fuera certera. A continuación se explicarán ambos experimentos con mayor detalle.

4.1.1. Experimento 1: Control de posición

El primer experimento realizado para la obtención del control de posición, consistió en llevar el motor a diferentes posiciones sobre el eje coordenado para comprobar que la posición deseada era la que se obtenía en el movimiento, es decir, se trataba de verificar la repetibilidad en el posicionamiento de un mecanismo simple en un eje mediante la operación del software desarrollado en un modo de control de posición [22]. Esto permitió la verificación de una desviación máxima de ±1 cuenta de la posición objetivo.

Debido a que el control de movimiento de motores se hace mediante una tarjeta adquirida, fue necesario introducir valores de ganancia PID, los cuales fueron obtenidos a través de las repeticiones en el experimento realizado. Los valores que se obtuvieron fueron: P = 100, I = 0 y D = 10000. Con estos valores se obtuvo el funcionamiento óptimo en el posicionamiento del motor.

4.1.2. Experimento 2: Control de movimiento

En el segundo experimento se persiguió la comprobación del control de movimiento en una medición experimental que simulara un palpador de contacto, para ello se empleo el software desarrollado en un modo de medición por contacto.

Para el desarrollo del perfil de movimiento se siguió un procedimiento detallado empezando por especificar una posición real de contacto, que pudiera determinarse de manera experimental mediante patrones dimensionales de mayor exactitud acercándose a la precisión. Posteriormente, se procedió a especificar una velocidad de palpación en lazo cerrado, esta velocidad se establece como parámetro en la tarjeta de control iniciando con una aceleración constante hasta llegar a cero una vez que se consiguiera la velocidad objetivo. Para el joystick se programaron dos velocidades; la de posicionamiento y la de palpación. La primera se utiliza cuando el

palpador esta a una distancia considerable del objeto a medir y su objetivo es llevar a una proximidad el palpador, la de palpación o contacto es menor que la anterior, se emplea cuando el palpador se encuentra próximo al objeto a medir hasta hacer contacto con este.

Al alcanzar el palpador la posición de disparo, esto es cuando uno de los interruptores es accionado, se obtiene de manera inmediata el contacto con el interruptor eléctrico en una posición superior a la de contacto. Cuando este momento llega, se frena el sistema con una desaceleración constante hasta alcanzar una velocidad nula.

El sistema refleja cierta inercia lo que provoca que el palpador continúe su viaje hasta una posición de máxima doblez en el interruptor eléctrico aproximadamente. De esta manera, el palpador se retira de la posición de máxima doblez con arranque de aceleración constante, después cero, y finalmente desaceleración constante hasta alcanzar una posición menor a la de contacto, así se libera al palpador de una proximidad con la pieza de trabajo y entonces se obtiene la medición por contacto.

El error por pre-recorrido es calculado como la diferencia entre las posiciones de contacto y disparo, en donde el signo le es asignado de acuerdo con el sentido de palpación que llega indicar el interruptor. La repetibilidad del experimentó que se verificó fue de ±1 cuenta en la posición de contacto.

4.2. Conclusiones

Mediante los experimentos realizados con el prototipo de un eje se comprobó la eficiencia del algoritmo así como los dispositivos adquiridos para la construcción de este. Cabe resaltar que este prototipo cuenta con la ventaja de que no solo puede ser aplicado a MMC’s sino también a Máquinas de Calibración de Escalas Graduadas (MCEG) []. Debido a que es un control de movimiento para uno o más motores, las aplicaciones pueden ser muchas mientras que se emplee un motor o más y estos tengan que llevarse a un a cierta posición.

La aplicación para las Máquinas de Calibración de Escalas Graduadas (MCEG) del prototipo únicamente es para el posicionamiento, mientras que para las Máquinas de Medición por Coordenadas (MMC) permite el control de posición y control de velocidad para realizar mediciones por contacto.

Para hacer la simulación del palpador cuando se realiza una medición por contacto se desarrolló un modelo simple de palpación que consta de un interruptor doble que al ser accionado en uno u otro sentido hace la equivalencia del contacto y representa entonces la posición de disparo. La arquitectura mencionada permite tratar

con los errores que se formarían naturalmente en una medición realizada por una Máquina de Medición pos Coordenadas (MMC).

El proyecto desarrollado y los resultados experimentales obtenidos se presentaron en el XVIII Congreso de Instrumentación de la Sociedad Mexicana de Instrumentación [26].

4.3. Trabajo a futuro

Al haber desarrollado los experimentos surgieron necesidades que nos permitieron determinar cual es al trabajo a futuro para complementar la Máquina de Medición pos Coordenadas (MMC) de tal manera que pueda competir con una máquina comercial.

Primeramente es necesario el desarrollo de un palpador de contacto con cinco sentidos de palpación para que la máquina determine posiciones en los tres ejes coordenados y pueda así medir objetos de forma tridimensional.

El desarrollo de software que permita al operador programar rutinas de medición preestablecidas y con formas determinadas, en forma similar a los instrumentos de control numérico y que pueda realizar rutinas diseñadas por el mismo operador.

Realizar un sistema de calibración para cada eje de la Máquina de Medición pos Coordenadas (MMC) así como en el palpador para poder realizar mediciones más exactas

Desarrollar técnicas de visión por computadora que exploten la exactitud en el posicionamiento de la Máquina de Calibración de Escalas Graduadas (MCEG) construida en el CCADET.

Poco a poco se ha ido perfeccionando este proyecto pero aún existe trabajo por desarrollar y completar así la Máquina de Medición por Coordenadas (MMC). Es necesaria la participación tanto de estudiantes y académicos para lograr dicho objetivo, pero lo más importante es que es desarrollado en la Universidad Nacional Autónoma de México.

Bibliografía

[1]. G. Bermúdez, B. Valera, J. Sánchez, R. Nava, "Interfase para una máquina de medir por coordenadas", *XIV Congreso de Instrumentación SOMI*, Tonantzintla Puebla, 1999, pp 565-569.

[2]. Brown & Sharpe, Introducción a la medición por coordenadas, http://www.bnsh.es/z/i/i13.htm, 2004.

[3]. F. J. Ceballos, *Microsoft Visual C++: Aplicaciones para Win32*, 2ª edición, Ra-Ma, Marzo 2003.

[4]. F. J. Ceballos, *Programación Orientada a Objetos con C++*, Ra-ma, España; 1998.

[5]. US Digital, *E2 Optical Kit Encoder*, (US Digital), pp. 4, (2003).

[6]. M. Ercole, "Máquinas de medir tridimensionales", http://www.metalunivers.com/1pm/pm02/mmt/MMT.htm, 29 de junio de 2004.

[7]. David Genest, Aplique la metrología por coordenadas a su taller, http://www.metalunivers.com/Arees/metrologiadimensional/tutorial/taller.htm, 2004.

[8]. W. J. Grantham and Thomas L. Vincent, *Sistemas de Control Moderno. Análisis y Diseño*, Limusa Noriega Editores, México; 1998.

[9]. Fugene Kain, *The MFC answer Book. Solutions for effective Visual C++ Applications*, Addison-Wesley Publishing Company, 1998.

[10]. J R Kerr, *Z232-485 Serial Port Conver*ter, (J R Kerr), pp. 5, (2003).

[11]. J R Kerr, *PIC-SERVO Motor Control Board*, (J R Kerr), pp. 10, (2003).

[12]. G. Kranklin, D. Powell and A. Emami-Naenini, *Feedback Control of Dynamic Systems*, Addison-Wesley Publishing Company, 1994.

[13]. D. J. Kruglinski, *Progrese con Visual C++*, McGrawHill, 2000.

[14]. B. C. Kuo, *Sistemas de Control Digital*, Compañía Editorial Continental S.A. de C.V., México; 1997.

[15]. Pablo Luna, Comunicaciones mediante puertos en PC 1, http://usuarios.lycos.es/cursoimm/capitulo10.htm, 2004.

[16]. Microchip Technology Inc. ,Pic Microcontrollers PIC 16 C52X, www.microchip.com,2004.

[17]. J. Montbrun D. Fillipo, Introducción al Control Digital, http://prof.usb.ve/montbrun/PS2320/digital/digital.htm

[18]. K. Ogata, *Ingeniería de Control Moderna*, Prentice Hall Hispanoamericana; 1993.

[19]. SMCC, Control Digital para motores D.C. con codificadores, http://www.kelvin.es/formatoshtm/SMCC.htm, 2004.

[20]. B. Strustrup, *El lenguaje de programación C++*, Addison-Wesley Publishing Company, 2000.

[21]. S. H. Suh y S. K. Lee, *Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, "Compensating Probe Radius in Free Surface Modeling with CMM", (IEEE), pp. 222-227, (1994).

[22]. J. Tang, "PID Controller Using the TMS320C31 DSK with On-line Parameter Adjustment for Real-time DC Motor speed and Position Control", *Proceedings of the IEEE International Symposium on Industrial Electronics*, 2001, pp 786-791.

[23]. Oscar G. Tropic, http://www.euskalnet.net/shizuka/rs232.htm, 2004.

[24]. Z. H. Xiong y Z. X. Li, *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, "Error Compensation of Workpiece Localization", (IEEE), pp. 2249-2254, (2001).

[25]. Q. Yang, C. Butler y P. Baird, "Error Compensation of touch trigger probes", *Measurement*, Vol. 18, Num. 1, (1996), pp. 47-57.

[26]. J. Zavaleta, B. Valera, G. Ruíz y J. Sánchez, “Sistema para el control de movimiento en máquinas de múltiples ejes”, *SOMI XVIII Congreso de Instrumentación*, Sociedad Mexicana de Instrumentación, México D. F., 6-10 octubre de 2003, pp ref. BVO1858.