UNIVERSIDAD LASALLISTA BENAVENTE

ESCUELA DE INGENERIA EN COMPUTACION

CON ESTUDIOS INCORPORADOS ALA

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
CLAVE: 8793-16

"CONTROL DE MOTORES DE PASOS POR COMPUTADORA"

T E S I S :

QUE PARA OBTENER EL TITULO DE:
INGENIERO EN COMPUTACION

P R E S E N T A:
JOSE DE JESUS RICO JIMENEZ

ASESOR:
ING. NOE DE JESUS VELA AGUIRRE

CELAYA GTO **2003**

A

ÍNDICE

INTRODUCCIÓN

CAPÍTULO I. ANTECEDENTES DE DESARROLLO 1

1.1 CONTROL 1

1.1.1 Tipos de sistemas de control 1

1.1.2 Términos básicos de los sistemas de control 2

1.2 MOTORES 4

1.2.1 Motores paso a paso 4

1.2.2 Tipos de motores paso a paso y su funcionamiento 5

1.2.2.a Motores paso a paso de reluctancia variable 6

1.2.2.b Motores paso a paso de imán permanente 9

1.2.3 Servomotor 16

1.2.3.a Funcionamiento del servomotor 17

1.2.4 Motor paso a paso vs Servomotor 19

CAPÍTULO II. DIAGRAMA A BLOQUES DEL PROYECTO 21

2.1 DETERMINACIÓN DEL DIAGRAMA A BLOQUES 21

2.1.1 ¿Que es un diagrama a bloques, cuáles son sus componentes y para qué sirve en sistemas de control? 21

2.1.2 Determinación del tipo de sistema para desarrollar el diagrama a bloques 23

2.1.3 Diseño del diagrama a bloques del proyecto 25

2.2 ANÁLISIS DE LAS SALIDAS Y ENTRADAS DEL SISTEMA 26

2.2.1 Determinación de que salidas producirá el sistema 26

2.2.2 Determinación de las entradas que aceptará el sistema 27

2.3 ANÁLISIS DE LOS PROCESOS DEL SISTEMA 28

2.3.1 Determinación de los procesos que realizará el sistema 28
2.3.2 Análisis y determinación de los métodos para llevar a cabo los procesos necesarios 30

CAPÍTULO III. DISEÑO 43
3.1 DISEÑO DE LA APLICACIÓN 43
3.1.1 Diseño del software de la perforadora 44
3.2 USO DE LA ETAPA DE POTENCIA 67
3.2.1 Funcionamiento del driver 67
3.2.2 Funcionamiento del controlador 75
3.2.3 Multiplexación 86
3.3 CONEXIÓN DEL MOTOR 89
3.4 DISEÑO DEL GRAFICADOR 93

CAPÍTULO IV. PRUEBAS Y RESULTADOS 98
4.1 PRUEBAS Y RESULTADOS DEL SOFTWARE 98
4.2 PRUEBAS Y RESULTADOS DEL HARDWARE 100
4.3 PRUEBAS Y RESULTADOS DEL GRAFICADOR 101

CONCLUSIONES

BIBLIOGRAFÍA

INTRODUCCIÓN

Con el avance acelerado de la Ciencia y Tecnología cada vez es más fácil automatizar los procesos, y es muy importante porque que en la actualidad en cualquier empresa se requiere que sus procesos se realicen más rápido, con mayor precisión y a bajo costo. La computadora es una herramienta muy útil en la realización de procesos, que evita errores costosos, permite la simulación y aplicación de los procesos industriales como medición, control de movimiento, procesamiento de señales, entre otros adecuándose fácilmente y permitiendo al usuario manejarlos y controlarlos mas rápidamente. En el presente trabajo se tratan de automatizar los procesos por medio de una interacción del software y hardware de la computadora con las maquinas que realizan estos procesos. Se hace referencia a la construcción del software y hardware para manejar motores de pasos, de tal forma que se tenga una gran precisión en trabajo, bajando costos de fabricación y reduciendo sus tiempos.

Por medio de la automatización se maximiza la seguridad ya que el personal no interactúa directamente con las máquinas sino que la computadora se encarga de interpretar las ordenes que les dan los usuarios y así se tiene una relación indirecta máquina-hombre, además de que eleva el nivel de calidad de la producción.

El trabajo esta dedicado al tratamiento de motores de pasos desde la computadora, se verá cada tipo de motor de pasos que existe y sus

características de funcionamiento. Se enfatizará en la resolución de problemas por medio del uso de motores de pasos controlados por la computadora.

Se analizará la forma de construir el software y el hardware necesario para cada tipo de motor. El software se enfocará a la resolución del problema de manera que realice el trabajo más rápido, con mayor precisión y que presente una interfaz fácil de manejar para el usuario, sin necesidad de una extensa capacitación, además debe ser flexible y adaptarse para realizar diferentes funciones. El hardware se enfocará a la creación y desarrollo del controlador, es la parte que maneja físicamente al motor, y el Driver, que es el manejador de la potencia; también se tratará de construir de la forma más simple y con el menor costo, dando así varias alternativas de construcción.

CAPÍTULO I

ANTECEDENTES DE DESARROLLO

1.1 CONTROL

En la actualidad, el control automático es una de las partes más importantes en el avance de la ingeniería y la ciencia; es utilizado en sistemas de vehículos espaciales, robots y en los procesos industriales modernos y de manufactura. Es esencial en operaciones industriales como el control de movimiento, presión, temperatura, viscosidad y flujo en las de procesos. Mejora la productividad, aligera la carga de las operaciones manuales repetitivas y rutinarias; además, todo ingeniero debe tener amplio conocimiento de este campo.

1.1.1 TIPOS DE SISTEMAS DE CONTROL

En los sistemas existen dos tipos de control: ***en lazo cerrado*** y ***en lazo abierto.***

Los **sistemas de control en lazo cerrado** son sistemas de control realimentados, la salida afecta a la acción de control. En estos sistemas se alimenta al controlador con la señal de error de actuación, que es la diferencia entre la señal de entrada y la señal de realimentación, con el fin de reducir el error y llevar la salida de sistema a un valor deseado.

En los **sistemas de control en lazo abierto** la salida no afecta la acción de control, no se mide la salida ni se realimenta para compararla con la entrada. A cada acción de referencia le corresponde una condición operativa fija; por lo tanto, la precisión del sistema depende de la calibración. Tiene una desventaja

importante: ante la presencia de perturbaciones, no realiza la tarea deseada. En la práctica, el control en lazo abierto sólo se usa si se conoce la relación entre la entrada y la salida y si no hay perturbaciones internas y externas.

Comparando los sistemas de lazo cerrado y los de lazo abierto, una ventaja del control en lazo cerrado es que el uso de la realimentación vuelve la respuesta del sistema relativamente insensible a las perturbaciones externas y a las variaciones internas en los parámetros del sistema. Ahora, desde el punto de vista de la estabilidad, el de lazo abierto es más fácil de desarrollar, porque la estabilidad del sistema no es problema importante; en los de lazo cerrado, la estabilidad es función principal del sistema. En general, una combinación adecuada de controles en lazo abierto y en lazo cerrado es menos costosa y ofrecerá un desempeño satisfactorio del sistema.

1.1.2 TÉRMINOS BÁSICOS DE LOS SISTEMAS DE CONTROL

En los sistemas de control deben definirse ciertos términos básicos:

- **La variable controlada y variable manipulada.** La variable controlada es la cantidad o condición que se mide y controla. La variable manipulada es la cantidad o condición deseada que el controlador utiliza para ajustar el valor de la variable controlada. Normalmente, la variable controlada es la salida del sistema. Al controlar el sistema se mide la variable controlada del sistema y se le aplica la variable manipulada para corregir y limitar una desviación del valor medido a partir de un valor deseado.
- **Plantas.** Son la parte de un equipo, o un conjunto de partes de una máquina que funcionan juntas, su propósito es ejecutar una operación particular. Son cualquier objeto físico a controlar, por ejemplo: un dispositivo mecánico, un horno de calefacción, un reactor o una nave espacial.

- **Procesos.** Son las operaciones, con una serie de cambios graduales que suceden uno al otro en una forma relativamente fija y que conducen al resultados o propósitos determinados. Puede ser una operación artificial o voluntaria progresiva que consiste en una serie de acciones o movimientos controlados, sistemáticamente dirigidos hacia un resultado o propósito determinado. Son las operaciones que se van a controlar.
- **Sistemas.** Son una combinación de componentes relacionados entre sí que actúan juntos o realizan un objetivo determinado. Pueden ser sistemas físicos, biológicos, económicos y similares.
- **Perturbaciones.** Son señales que tienden a afectar negativamente al valor de la salida del sistema. Si una perturbación se genera dentro del sistema se denomina interna, y si se genera fuera del sistema es externa y se toma como una entrada.

1.2 MOTORES

Esta sección presenta los tipos de motores utilizados en los sistemas de control, como son ***motores de pasos*** y ***servomotores***, detallando su funcionamiento y sus tipos. Además se presenta una comparación entre ambos.

1.2.1 MOTORES PASO A PASO

Los motores paso a paso, o también llamados motores de pasos, son motores que giran en un ángulo proporcional a la codificación de tensiones que se aplican a sus bobinas en forma secuencial. Llevar un control en todo momento de esta codificación permite realizar movimientos angulares (en sentido horario y antihorario) muy precisos, dependiendo del número de pasos por vuelta o ángulo de paso del motor, que pueden ser desde 0.72°, 1.8°, 3.6°, 7.5°, 15°, 30° y aún de 90°. Los ***grados por paso*** pueden ser calculados dividiendo 360° entre el número de pasos que se necesitan para completar una revolución. La ***velocidad*** del motor ésta en función directa a la frecuencia de variación de las codificaciones en las entradas de las bobinas.

Estos motores, como todos, constan de dos partes, que son: el ***rotor*** y el ***estator***. Todas las bobinas en el motor son parte del estator, y el motor es, o un imán permanente, o como en los motores de reluctancia variable, un bloque dentado de un material ligeramente magnetizado. Un ***controlador*** maneja externamente la conmutación del motor, de tal manera que puede ser frenado en una posición fija o que sea rotado en una dirección o en otra. La mayoría de los motores de pasos pueden ser fasados en frecuencias de audio, permitiendo girarlos bastante rápido.

Los motores paso a paso se pueden encontrar en algunos equipos electromecánicos como son las impresoras de matriz de punto, floppy's entre otros.

Hay 3 parámetros que caracterizan a un determinado motor de pasos que son:

- ***Voltaje***, que normalmente está impreso directamente en la unidad o que se especifica en la hoja de datos del motor. Este voltaje se debe respetar para evitar producir calor excesivo o acortar la vida del motor.
- ***Resistencia por bobinado***, que determina la corriente del motor, afecta la curva de torque y la velocidad máxima de operación.
- ***Grados por paso***, este es el más importante porque habilita al motor para una determinada aplicación dependiendo de la precisión que se requiera. Los grados por paso comúnmente se refieren a la resolución del motor.

1.2.2 TIPOS DE MOTORES PASO A PASO Y SU FUNCIONAMIENTO

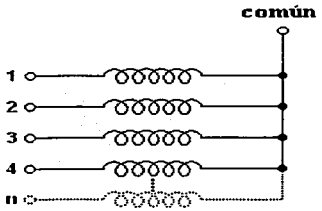
Existen dos categorías básicas de motores de pasos: la de ***imán permanente*** y de ***reluctancia variable*** (aunque también hay ***híbridos***, que se pueden operar como unipolares o bipolares de imán permanente). Una forma de distinción es que los motores de magneto permanente tienden seguir la inercia en el giro debido a la perdida de torque, mientras que los de reluctancia variable casi giran libremente (pero también tienden a equivocarse ligeramente por la magnetización residual en el rotor). Los de reluctancia variable tienen 3 ó 4 bobinas, mientras que los de imán permanente usualmente tienen 2 bobinas independientes, con o sin derivación central. Las bobinas de derivación central son usadas en motores unipolares.

Los motores menos precisos normalmente giran 90° por paso, mientras que los de más alta resolución giran hasta 0.72° por paso. Con el controlador apropiado la mayoría de los motores de imán permanente e híbridos pueden ser girados o rotados con ***medios pasos*** y algunos controladores manejan pasos fraccionales más pequeños llamados ***micropasos***.

Para los motores paso a paso de imán permanente y de reluctancia variable, si sólo una bobina es energizada, el rotor pasa a un ángulo fijo y se queda en ese ángulo hasta que el torque excede el torque de frenado (o también llamado de sostenimiento) del motor y en ese punto el motor girará, intentando detenerse en cada punto de equilibrio sucesivo.

- **Motores paso a paso de reluctancia variable**

Son los motores más simples de controlar de todos los tipos de motores paso a paso. Su secuencia de manejo es simplemente energizar cada una de las bobinas en orden, una después de otra como se muestra en la tabla de la figura 1.1.



bobina_1	1	0	0	0	0	1	0	0	0	0
bobina_2	0	1	0	0	0	0	1	0	0	0
bobina_3	0	0	1	0	0	0	0	1	0	0
bobina_4	0	0	0	1	0	0	0	0	1	0
⋮										
bobina_n	0	0	0	0	1	0	0	0	0	1

tiempo →

Figura 1.1 Configuración de bobinas de un motor de pasos y sus secuencias de energización

Estos motores tienen una sola terminal, la cual es común para las otras terminales, con la cual se combinan para energizar cada una de las bobinas. Este tipo de motor se siente como un motor DC (corriente directa) cuando el rotor es girado con la mano, gira libremente y no se pueden sentir los pasos. Estos motores no están magnetizados permanentemente como ocurre con los unipolares y bipolares que se verán mas adelante.

Para explicar mas afondo el funcionamiento de estos motores de pasos si tomamos como referencia uno de 30° por paso con reluctancia variable mostrado en la figura 1.2, el motor tendrá 3 bobinas conectadas como se muestra en la figura, con una terminal común que va a todas las bobinas. La terminal común se conecta al suministro positivo y se energizan las bobinas en secuencia.

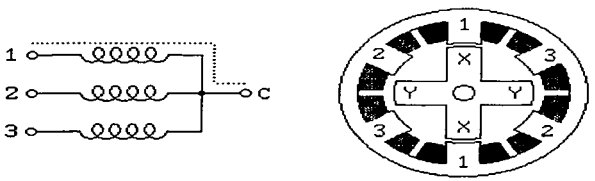


Figura 1.2[1] Motor de reluctancia variable con un paso de 30°

En este motor se tienen 4 dientes en el rotor, y 6 polos en el estator, con cada bobina envuelta alrededor de 2 polos opuestos. Si la bobina 1 es energizada, los dientes del rotor marcados con una **X** son atraídos a los polos de esta bobina, como se muestra en la figura 1.2 en las zonas más obscuras. Si se corta la corriente en la bobina 1 y se energiza la 2, el rotor girará 30° en sentido horario (dirección de las manecillas del reloj), debido a que los polos marcados con **Y** se alinean con los polos de la bobina 2 como se muestra en la figura 1.3.

[1] http://www.cs.uiowa.edu/~jones/step/

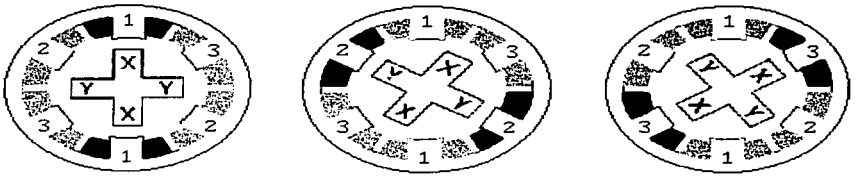


Figura 1.3 Secuencia de posicionamiento del rotor de un motor de pasos de reluctancia variable con un paso de 30°

Para rotar el motor continuamente se deben energizar las 3 bobinas en forma secuencial. Ahora con lógica binaria mostrada en la tabla 1.1, si tiene un **1** significa que estará energizada la bobina y si tiene **0** estará desactivada. En la tabla 1.1 se muestra una secuencia de control que hará girar el motor que se tomó como referencia en la figura anterior, dando 24 pasos que equivalen a 2 revoluciones o 720°.

				90°			180°			270°			360°			450°			560°			640°			720°
Bobina 1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
Bobina 2	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
Bobina 3	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0

Tiempo

Tabla 1.1 Lógica binaria para la secuencia de giro

También hay motores paso a paso de reluctancia variable con 4 y 5 bobinas y con 5 o 6 conexiones. Se utiliza el mismo principio para manejar la secuencia de estos motores que el de 3 bobinas, pero se debe llevar un orden correcto al energizar las bobinas para que el motor gire de manera debida.

Si se usan más polos en el estator y más dientes en el motor, se tendrán motores con un ángulo de paso más pequeño. Las caras de cada polo y su

correspondiente rotor finamente dentado permite ángulos tan pequeños como unos cuantos grados.

- **Motores paso a paso de imán permanente**

De estos motores se deben diferenciar los ***unipolares*** de los ***bipolares***. Esta clasificación se da dependiendo del tipo de bobinas que se encuentran devanadas sobre los estatores y del modo de crear el campo magnético giratorio.

En los motores ***unipolares***, cada bobina se encuentra dividida en dos, mediante una derivación central conectada a una terminal de alimentación. Las derivaciones centrales de las bobinas se conectan al suministro positivo y las dos terminales de cada bobina son conectadas alternadamente a tierra para revertir la dirección del campo magnético provisto por esa bobina, pero la corriente siempre circula en el mismo sentido, en vez de invertir la polaridad como lo hacen los bipolares que se verán mas adelante.

Para explicar más a fondo el funcionamiento de estos motores, se toma como ejemplo un motor paso a paso de imán permanente mostrado en la figura 1.4, con 30° por paso.

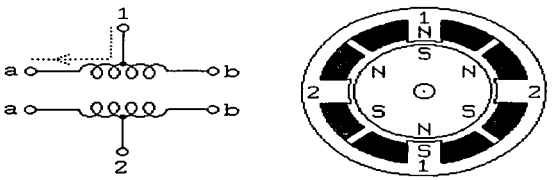


Figura 1.4[2] Motor unipolar con un paso de 30°

[2] http://www.cs.uiowa.edu/~jones/step/

La bobina 1 está distribuida entre el polo de arriba y el de abajo (zona más obscura de la figura 1.4), mientras que la bobina 2 está entre el polo izquierdo y el derecho. El rotor es un imán permanente con 6 polos, 3 al norte y 3 al sur (refiriéndose a polaridad magnética), arreglados alrededor de su circunferencia.

Si se quiere más resolución angular, el rotor debe tener más polos proporcionalmente. El motor paso a paso de imán permanente más común es el de 30° aunque también se pueden conseguir fácilmente los de 15° y 7.5° grados por paso, y los de mayor resolución son de 1.8°. En los híbridos de 3.6 y 1.8 grados por paso, siendo el de mayor resolución el de 0.72°.

La corriente que fluye desde la derivación central de la bobina ***1*** a la terminal ***a*** causa que el polo de arriba sea un polo norte, mientras que el de abajo sea un polo sur. Esto atrae al rotor a la posición mostrada en la figura 1.4. Si se desactiva la bobina ***1*** y se energiza la ***2***, el rotor girará 30°, o un paso.

Para girar continuamente el motor, se debe aplicar energía a las 2 en secuencia. Si se utiliza lógica binaria donde 1 es suministrar corriente a una bobina, tenemos dos secuencias de flujo válidas que giran el motor en sentido horario 24 pasos ó 2 revoluciones.

Como se puede apreciar el la taba 1.2, los dos medios, ***a*** y ***b***, de una bobina nunca son energizados al mismo tiempo. En la primera secuencia (tabla 1.2) sólo se energiza una bobina a la vez y utiliza menos potencia. La segunda secuencia (tabla 1.3) energiza las dos bobinas a la vez, lo que generalmente producen un torque de 1.4 veces más grande que el torque de la primera secuencia, pero utiliza el doble del poder.

Bobina_1a	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
Bobina_1b	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
Bobina_2a	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
Bobina_2b	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0

tiempo →

Tabla 1.2 Secuencia de alimentación de una sola bobina para un motor unipolar

Bobina_1a	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
Bobina_1b	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
Bobina_2a	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
Bobina_2b	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

tiempo →

Tabla 1.3 Secuencia de alimentación de dos bobinas para un motor unipolar

Las figuras 1.5 y 1.6 muestran cómo se realizaría fisicamente el movimiento del rotor dentro del motor paso por paso de las secuencias anteriores.

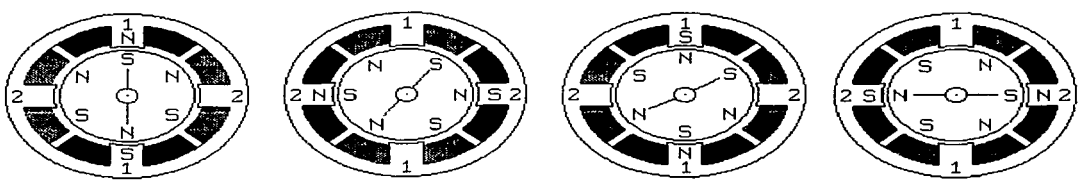


Figura 1.5 Movimiento del rotor de un motor de pasos de acuerdo a la secuencia de la tabla 1.2

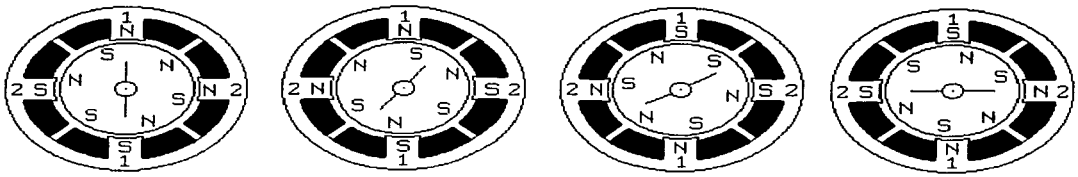


Figura 1.6 Movimiento del rotor de un motor de pasos de acuerdo a la secuencia de la tabla 1.3

Las posiciones de los pasos producidos por las dos secuencias anteriores son las mismas. La combinación de estas secuencias permite medios pasos, parando el motor alternadamente en las posiciones indicadas por una u otra secuencia. Obsérvese la siguiente secuencia de la tabla 1.4 y la figura 1.7, que producen medios pasos:

Bobina_1a	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1
Bobina_1b	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0
Bobina_2a	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1
Bobina_2b	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0

tiempo

Tabla 1.4 Secuencias para producir medios pasos en un motor unipolar

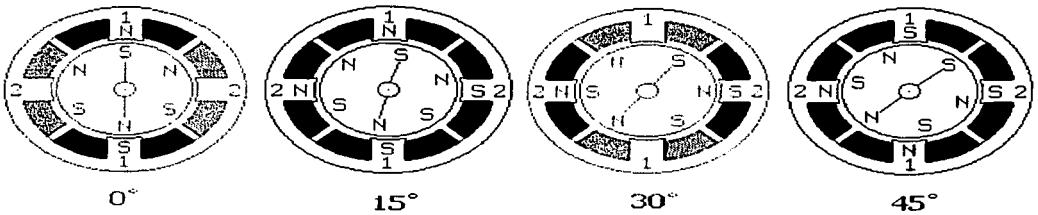


Figura 1.7 Movimiento del rotor en medios pasos de un motor unipolar

Los motores ***bipolares*** son construidos con el mismo mecanismo que los unipolares, pero las 2 bobinas están conectadas de manera más simple sin derivación central. Para que el motor funcione, la corriente que circula por las bobinas cambia de sentido en función de la tensión que se aplica, de esta forma una misma bobina puede tener distinta polaridad (bipolar) en sus extremos. Así la dificultad de manejar los motores bipolares esta en controlar la alimentación y cambiar la polaridad y el ritmo de las bobinas, de manera que el motor funcione correctamente. La figura 1.8 muestra la forma en la que el motor es conectado, nótese que el motor es igual al mostrado anteriormente en la figura 4.

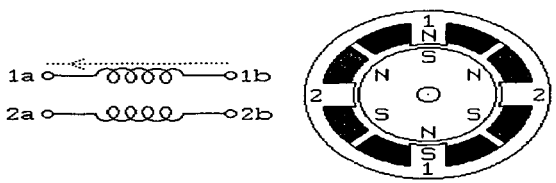


Figura 1.8[3] Motor bipolar con un paso de 30°

[3] http://www.cs.uiowa.edu/~jones/step/

Los motores bipolares requieren de un ***circuito manejador*** (driver) más complejo que los unipolares, llamado puente-H, para cada una de las bobinas. El puente-H permite que la polaridad de poder aplicado a cada terminal de cada bobina sea controlada independientemente. Obsérvese la tabla 1.5 que muestra un ejemplo de las secuencias de control, se usará (+) y (-) para indicar la polaridad de la energía aplicada en cada terminal.

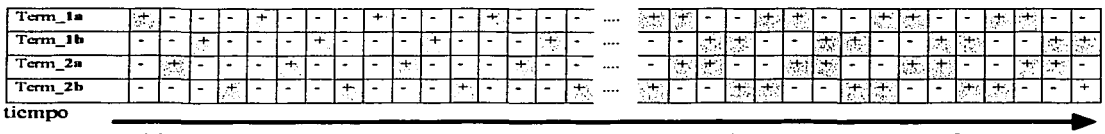


Tabla 1.5 Secuencias de control para un motor bipolar con un paso de 30°

También existen motores ***multifase***, que son una clase de motor de imán permanente y se conectan todas las bobinas del motor en una serie cíclica, con una derivación entre cada par de bobinas en el ciclo, las más comunes son las de 3 fases y 5 fases. Su control requiere ½ del puente-H para cada terminal. Como todas las bobinas son energizadas en cada punto en el manejo del ciclo proveen de un torque mayor. Pueden tener una resolución de 0.72 grados por paso (o 500 pasos por revolución). Con un motor de 5 fases, hay 10 pasos a repetir en un ciclo de pasos, como se muestra en la tabla 1.6.

Terminal 1	+	+	+	-	-	-	-	-	+	+	...	+	+	+	-	-	-	-	-	+	+
Terminal 2	-	-	+	+	+	+	+	-	-	-	...	-	-	+	+	+	+	+	-	-	-
Terminal 3	+	-	-	-	-	-	+	+	+	+	...	+	-	-	-	-	-	+	+	+	+
Terminal 4	+	+	+	+	+	-	-	-	-	-	...	+	+	+	+	+	-	-	-	-	-
Terminal 5	-	-	-	-	+	+	+	+	+	-	...	-	-	-	-	+	+	+	+	+	-

tiempo

Tabla 1.6 Secuencias de control para un motor bipolar de 5 fases

Se puede apreciar que sólo una terminal cambia de polaridad. Este cambio desactiva la energía de una bobina conectada a una terminal (porque ambas terminales de la bobina en cuestión son de la misma polaridad) y se aplica energía a una bobina que estaba desactivada anteriormente.

Si el motor es como se muestra en la figura 1.9, la secuencia de control anterior producirá 2 revoluciones.

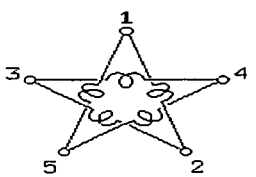


Figura 1.9[4] Esquema de un motor bipolar de 5 fases

El circuito ***puente-H*** mencionado anteriormente es usado para manejar motores de pasos bipolares. Normalmente los motores bipolares tienen 4 conductores, conectados a dos bobinas aisladas en el motor. Una característica de

[4] http://www.cs.uiowa.edu/~jones/step/

los circuitos puente-H es que tienen "frenos" eléctricos que pueden ser aplicados para reducir la velocidad y aún para detener el motor de su rotación libre cuando no se está moviendo bajo control del circuito. En la figura 3.13 del capitulo 3 apartado 3.1, se muestra el diagrama de un circuito puente–H y una tabla de su operación.

1.2.3 SERVOMOTOR

Un servomotor es un motor eléctrico de precisión en el que se pueden controlar su velocidad y/o posición. Consiste de un pequeño motor DC (corriente directa), un mecanismo de retroalimentación (potenciómetro o también llamado resistencia variable) conectado al motor por medio de engranajes y un circuito de control que compara la posición del motor deseada, y lo mueve según se necesite. Éste puede ser llevado a posiciones angulares específicas al enviar una señal codificada.

Los servomotores son usados para posicionar superficies de control como el movimiento de palancas, pequeños ascensores, timones, en radio control como en los robots y en modelismo, que se utilizan en aviones, coches y vehículos a escala a control remoto.

Los tipos de servomotores más comunes son los motores ***con escobillas*** (de corriente directa o alterna), y también hay motores ***sin escobillas*** (también llamados "brushless", de corriente directa o alterna).

En la figura 1.10 se muestran las piezas que componen al servomotor, como son la tarjeta controladora, el motor DC, el juego de engranajes, la flecha y la resistencia variable, y como están organizadas dentro de la caja. También se pueden observar 3 cables de conexión externa, uno para la alimentación de Vcc,

otro para la tierra (GND) y el de control (normalmente de color rojo, negro y blanco, respectivamente, pero pueden cambiar según el fabricante), este último es el cable por el cual se le pide al servomotor en que posición acomodarse, ya sea 0 ó 180 grados.

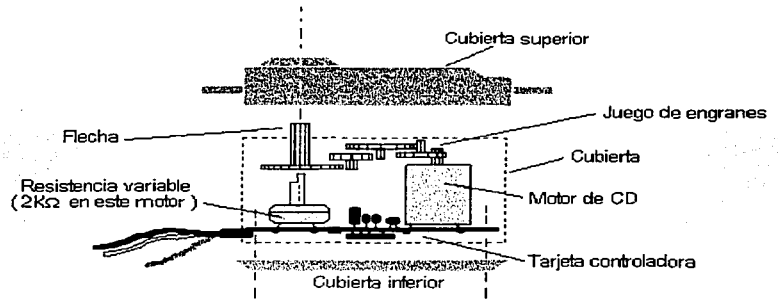


Figura 1.10[5] Componentes de un servomotor

- **Funcionamiento del servomotor**

Como se puede ver en la figura 1.10 el potenciómetro esta sujeto a la flecha, y sirve para medir hacia donde esta rotada en todo momento, permitiendo así a la circuitería de control supervisar el ángulo actual del servomotor. Si el eje está en el ángulo correcto, el motor no se mueve. Si el circuito detecta que el ángulo no está correcto el motor volverá a la dirección correcta, hasta llegar a donde el ángulo es correcto. El eje del servomotor es capaz de llegar al rededor de 180 grados y hasta en algunos casos 210, pero varía según el fabricante. De esta forma, si se tiene que reacomodar la flecha, la tarjeta controladora le dice al motor DC cuántas vueltas girar para llegar a la posición que se le ha pedido.

[5] http://www.creaturoides.com/srvesp.htm

La cantidad de voltaje aplicado al motor es proporcional a la distancia que éste necesita viajar. Si el eje necesita viajar una distancia grande el motor regresará a toda velocidad; pero si éste necesita regresar sólo una pequeña cantidad, el motor correrá a una velocidad más lenta. A esto se le llama control proporcional.

Para comunicar el ángulo se utiliza el cable de control. El ángulo está determinado por la duración de un pulso que se aplica al alambre de control, por medio de PCM (por sus siglas en ingles: Pulse Codified Modulation, Modulación Codificada de Pulsos). El servomotor espera ver un pulso cada 20 milisegundos. La longitud del pulso determina los giros del motor. Por ejemplo, suponiendo que si el pulso es menor a 1.5 ms, entonces el motor se acercará a los 90 grados (llamada posición neutra), si el pulso es menor de 1.5ms, entonces el motor se acercará a los 0 grados, y si el pulso es mayor, el eje se acercará a los 180 grados. De esta forma, la duración del pulso indica el ángulo del eje.

Todo el tiempo debe haber una señal de pulsos presente en ese cable. Si se deja de enviar pulsos por más de 50 ms (dependiendo del servomotor), éste podría caerse (perdiendo su torque), y sólo la fricción sostendría el brazo. Si por alguna razón se necesita tener el servomotor prendido y no se le puede generar pulsos entonces se aterriza el cable de control.

Para cada servomotor, los tiempos reales dependen de lo que indique el fabricante del motor en la hoja de datos, es muy importante analizar ésta antes de usar el servomotor. Se debe realizar una prueba preliminar para encontrar exactamente el periodo y la duración de los pulsos que mejor funcionen. Esta prueba se puede llevar a cabo mediante un osciloscopio y un generador de señales.

Al servomotor también se puede mover a otro ángulo diferente de 180 ó 120 grados. Por ejemplo, si se le quiere mover a 30 grados se debe calcular la longitud de ancho de pulso.

En 0 grados => 1 ms,
en 120 grados => 2ms,
con relación lineal tenemos que:
en 30 grados => 1.6 ms.

Así, si se le envían pulsos de 1.6 ms, se incrementará su posición en 30 grados. El servomotor también es capaz de corregir un error en caso de que una fuerza externa intente bloquearlo, es decir, si el brazo se mueve externamente, ese servomotor dará entradas a su motor interno para corregir el error.

Actualmente existen nuevos servomotores los cuales son llamados "super servos". De este tipo de servomotor hay uno que toma el error del pulso y lo regenera a los intervalos más cortos; por consiguiente, controla al servo con una señal PWM (por sus siglas en ingles: Pulse Width Modulation, Modulación de Ancho de Pulso) para mayor eficiencia. También tienen mayor torque de espera y error de posición más bajo, pero estas características hacen mucho más costoso al servomotor. Otro servomotor de este tipo integra un microprocesador internamente. Se pueden programar en un punto central variable, a través de volumen, dirección y velocidad de operación. Se utiliza una unidad de programación para darle estas características al servomotor.

1.2.4 MOTOR PASO A PASO vs. SERVOMOTOR

Para determinadas aplicaciones se debe hacer una elección entre usar motores de pasos o servomotores. Ambos tipos de motores ofrecen posicionamiento de precisión, pero son muy diferentes en cuanto a su

funcionamiento. Dependiendo de la aplicación se deben considerar ciertas condiciones, por ejemplo, el posicionamiento de un motor de pasos depende de la geometría del rotor, mientras que el posicionamiento en un servomotor depende de la resistencia variable y el circuito de retroalimentación.

Debido a que los servomotores están formados por engranajes y un mecanismo de retroalimentación, son en algunos casos muy complicados de manejar, costosos y ocupan demasiado espacio; esto, para algunas aplicaciones, es muy importante. Sin embargo, los motores de pasos se comportan de una forma diferente a los servomotores; estos no pueden correr libremente porque tienen que avanzar paso por paso. También difieren en su torque, los motores de pasos generan grandes torques a bajas velocidades, mientras que los servomotores no producen grandes torques a bajas velocidades. Además, los motores a pasos tienen un torque de sostenimiento, del cual los servomotores carecen, permitiendo al rotor mantenerse en su posición cuando no está girando. Esto es muy útil en aplicaciones donde el motor se prende y apaga continuamente, mientras la fuerza que actúa contra el motor aún está presente y esto elimina la necesidad de usar mecanismos de frenado.

Los motores de pasos pueden ser usados en simples sistemas de control de lazo abierto. Si un motor de pasos en sistemas de control de lazo abierto es sobreforzado en su rotación, la posición conocida del rotor se pierde y el sistema tiene que ser reinicializado, pero los servomotores no están sujetos a este problema.

CAPÍTULO II

DIAGRAMA A BLOQUES DEL PROYECTO

2.1 DETERMINACIÓN DEL DIAGRAMA A BLOQUES

Todo sistema de control tiene uno o varios componentes. Se utiliza un ***diagrama a bloques*** para mostrar las funciones que lleva a cabo cada componente en la ingeniería de control. Este capítulo esta dedicado a analizar y determinar el tipo de sistema que se va ha utilizar en el desarrollo del proyecto, por medio de su diagrama a bloques. En el diagrama a bloques se definirán las entradas, salidas y procesos del sistema.

2.1.1 ¿QUÉ ES UN DIAGRAMA A BLOQUES, CUÁLES SON SUS COMPONENTES Y PARA QUÉ SIRVE EN SISTEMAS DE CONTROL?

Un diagrama a bloques de un sistema es una representación gráfica de las funciones que lleva a cabo cada componente y el flujo de señales. El diagrama muestra las relaciones existentes entre los diversos componentes. A diferencia de la representación matemática puramente abstracta, un diagrama a bloques tiene la flexibilidad de indicar en forma más realista el flujo de las señales del sistema real.

El diagrama utiliza ***bloques funcionales*** para enlazar una con otra todas las variables del sistema. El boque funcional es un símbolo para representar la operación matemática que el bloque, para producir la salida, hace sobre la señal de

entrada. Las ***funciones de transferencia*** de los componentes, por lo general, se introducen en los bloques correspondientes, los cuales se conectan mediante flechas para indicar la dirección de flujo de las señales, la señal sólo puede pasar en dirección de la flecha. En la figura 2.1(a) se muestra un ejemplo sencillo de la función de transferencia de un diagrama a bloques y la dirección hacia donde viajan las señales.

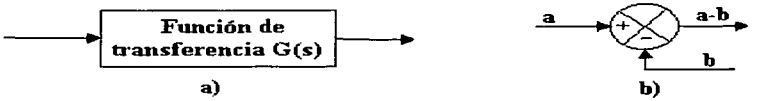


Figura 2.1 Representación de la función de transferencia y un punto de suma

La punta de flecha que señala al bloque indica la señal de entrada, y la que se aleja del bloque indica la señal de salida. Las dimensiones de la salida del bloque son las dimensiones de la señal de entrada multiplicadas por las dimensiones de la función de transferencia en el bloque.

Las ***ventajas*** de la representación de un sistema mediante diagrama a bloques estriban en que es fácil formar el diagrama de todo el sistema con sólo conectar los bloques de los componentes de acuerdo con el flujo de señales y en que es posible evaluar la contribución de cada componente al desempeño general del sistema. Es posible dibujar varios diagramas a bloques diferentes para un sistema, depende del punto de vista del análisis.

Existen otros elementos importantes dentro de los diagramas a bloques que son el ***punto de suma*** y ***punto de ramificación***. En la figura 2.1(b) se muestra un punto de suma, básicamente es un círculo con una cruz e indica la operación de suma. El signo más o menos en cada punta de flecha indica si la señal debe sumarse

o restarse; las cantidades a sumar o restar deben tener las mismas unidades. El punto de ramificación es aquel a partir del cual la señal de un bloque va de modo concurrente a otros bloques o puntos de suma.

A continuación, la figura 2.2 muestra un diagrama a bloques de un sistema de lazo cerrado. Como se puede observar, la salida C(s) se realimenta al punto de suma, en donde se compara con la entrada de referencia R(s). La salida del bloque C(s) se obtiene multiplicando la función de transferencia G(s) por la entrada al bloque, E(s). Cualquier sistema de control lineal puede representarse mediante un diagrama a bloques formado por puntos de suma, bloques y puntos de ramificación.

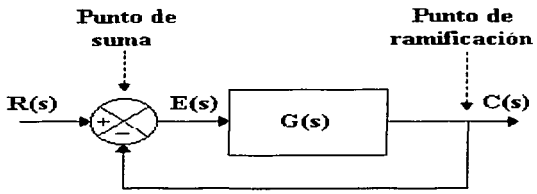


Figura 2.2 Diagrama a bloques de un sistema en lazo cerrado

2.1.2 DETERMINACIÓN DEL TIPO DE SISTEMA PARA DESARROLLAR EL DIAGRAMA A BLOQUES

El control de motores de pasos se lleva a cabo directamente desde la computadora y no necesita retroalimentación; por lo tanto, es un sistema de control en lazo abierto. Debido a que se conoce la relación de la salida con respecto de la entrada, no se requiere estar midiendo la salida para compararla con la entrada. El sistema debe funcionar de manera correcta en condiciones libres de perturbaciones.

Este sistema realiza el control en la computadora, indicando la posición del motor y conociéndola en todo momento según se haya programado; luego se manda la salida programada al controlador/driver, el cual envía las señales al motor, generando la potencia requerida, como se muestra en la figura 2.3.

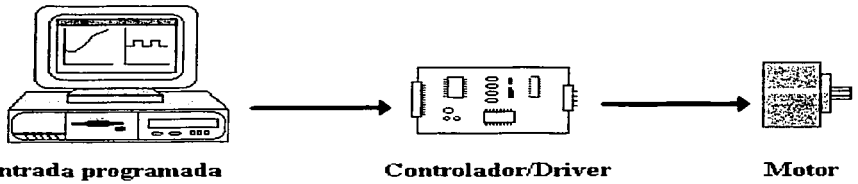


Figura 2.3 Representación física del flujo de señales

De esta manera se puede dibujar un diagrama a bloques general del sistema. Como ya se dijo anteriormente, será de lazo abierto debido a que no tiene retroalimentación. Quedaría como se muestra en la figura2.4.

Sistema de control de motores.

Figura 2.4 Diagrama a bloques general del sistema

De esta forma, la entrada programada será generada por el software, el cual mandará la información al puerto paralelo, que es donde se conectará el circuito controlador/driver.

2.1.3 DISEÑO DEL DIAGRAMA A BLOQUES DEL PROYECTO

Como el diagrama anterior no define los procesos que se deben realizar para mantener el control de los motores, por esto se debe tomar a la computadora como un sistema de control individual y definir su diagrama a bloques, el cual después se unirá con el anterior. La computadora es la que realmente realiza las funciones de control, los demás dispositivos sólo se encargan de adecuar las señales para que el motor funcione de manera correcta.

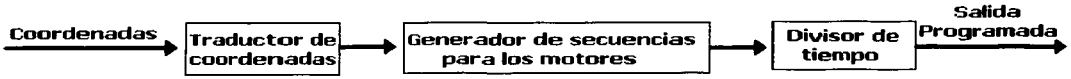
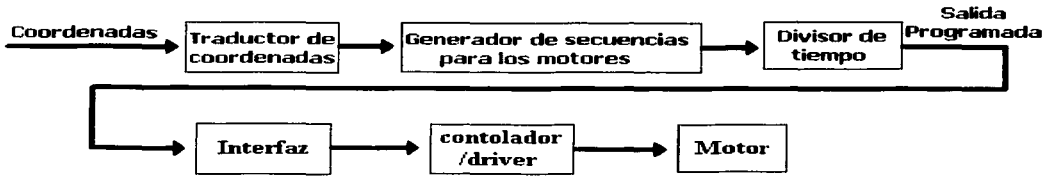


Figura 2.5 Flujo de información dentro de la computadora

El diagrama de la figura 2.5 muestra el flujo de información dentro de la computadora, recibiendo como entrada principal las coordenadas que serán traducidas a las secuencias requeridas para cada motor, determinando los pulsos que se les mandarán en un tiempo especifico.

Las coordenadas son interpretadas por el traductor convirtiéndolas en la información que determina qué motor girará y cuántos grados. El generador se encarga de producir las secuencias que determinan el movimiento de cada motor, según le ordene el traductor. El divisor de tiempo será el encargado de mandar los pulsos a cada motor en el tiempo que le corresponda, dando un tiempo específico para cada uno, en el cual se mandará la señal de control determinada por el generador de secuencias. De esta manera se obtiene la salida programada. El diagrama general quedaría como se muestra en la figura 2.6.



Sistema de control de motores.

Figura 2.6 Diagrama a bloques general del sistema

2.2 ANÁLISIS DE LAS SALIDAS Y ENTRADAS DEL SISTEMA

2.2.1 DETERMINACIÓN DE QUE SALIDAS PRODUCIRÁ EL SISTEMA

De acuerdo con los diagramas anteriores se pueden definir las salidas del sistema. Comenzando con el diagrama de la figura 2.6, el traductor de coordenadas dará como salida dos datos: el número del motor y los grados que va a girar. De una coordenada a otra, el traductor puede mandar desde un par de datos hasta n, dependiendo de la distancia que se tenga que recorrer. El generador de secuencias recibirá cada par de datos, transformándolos en la señal de control para el motor en cuestión; ésta será su salida, un número que convertido a binario representará los pulsos que alimentarán al motor. El divisor de tiempo será el encargado de enviar la señal de control que reciba del generador de secuencias, dando un tiempo determinado para cada motor.

En el diagrama de la figura 2.4, la entrada programada será el conjunto de señales de control en su determinado tiempo, que serán enviadas por el puerto paralelo hacia un demultiplexor, siendo estos dos elementos la interfaz de comunicación de la computadora con los dispositivos externos. El controlador y el driver recibirán las señales y las adecuarán de manera que el motor pueda funcionar correctamente, dando como salidas pulsos con voltajes y corrientes específicos según el motor a utilizar. Por último, cada motor dará como salida el movimiento deseado por medio de los giros de su rotor.

2.2.2 DETERMINACIÓN DE LAS ENTRADAS QUE ACEPTARÁ EL SISTEMA

Como las salidas de cada bloque se convierten en entradas no ha mucho que definir, sólo resta decir que la principal entrada, que aquí se define como un conjunto de coordenadas, puede ser no sólo (x,y) ó (x,y,z), sino también se podrán mandar más datos, por ejemplo (u,v,w,x,y,z), en los que supone manejará 6 motores, no necesariamente tiene que ser un plano cartesiano. El traductor necesitará esta información para determinar la distancia que se deben mover los motores, comparando la posición deseada con la posición actual, determinado cuantos pasos debe dar el motor y en que dirección. Por otra parte, la entrada programada será la información que arroje la computadora a los dispositivos externos.

2.3 ANÁLISIS DE LOS PROCESOS DEL SISTEMA

2.3.1 DETERMINACIÓN DE LOS PROCESOS QUE REALIZARÁ EL SISTEMA

El primer proceso a realizar es la **generación de coordenadas**. Este proceso consiste en formar una matriz multidimensional que contenga las posiciones hacia donde se moverá cada motor. La dimensión estará determinada por el número de movimientos y el número de motores que se van ha utilizar. La tabla 2.1 muestra un ejemplo de una matriz de coordenadas en donde se manejarán 4 motores.

Mov. \ motor	1	2	3	4
1	0	0	0	0
2	10	15	0	0
3	10	15	10	10
4	10	15	0	10
5	32	5	0	10
6	32	5	10	20
7	32	5	0	20
8	7	28	0	20
9	7	28	10	20

Tabla 2.1. Ejemplo de una matriz de coordenadas para manejar 4 motores

Aquí se muestra una simulación de cuatro motores de una perforadora, dos motores para las coordenadas (x , y) , uno para el taladro y otro para bajar y subir el taladro. Realizaría 3 perforaciones en (10,15), (32,5) y (7,28); en este nivel de abstracción no son importantes las unidades de medición, ya que esto dependerá de la aplicación.

Después que se tiene la matriz de coordenadas deseada, el siguiente proceso es la **traducción de coordenadas**. En este proceso se determina qué motores y cuánto deben girar, este proceso es tal vez el más importante, pues aquí es donde se da la coordinación, precisión y rapidez con la que se van a mover los motores. Se debe tener mucho cuidado al realizar la aplicación, por lo que se recomienda hacer muchas pruebas para evitar errores cuando se ponga en marcha el sistema.

Una vez que se ha definido qué motor se va a mover, el siguiente proceso es la **generación de secuencias**, estos dos últimos procesos estarán interactuando constantemente. En este proceso se deben tomar en cuenta dos condiciones: si el motor será controlado por un circuito integrado (especial para cada tipo de motor) o si la PC llevará el control total.

Para el primer caso, el proceso de generación de secuencias se facilita, debido a que el circuito integrado genera los pulsos de control para cada bobina del motor; así, lo único que se tiene que generar desde la PC es un conjunto de pares de bits; en tales pares un bit será para el sentido en que el motor girará y el otro para darle los pulsos necesarios para moverlo, estas serán las dos señales que necesita el circuito integrado del controlador/driver que estará conectado a la computadora por medio del puerto paralelo . En la tabla 2.2 se presenta un ejemplo de las secuencias de control manejadas con los 8 bits del puerto paralelo.

BIT	sec 1	sec 2	sec 3	sec 4	sec 56	sec 6	sec 7	sec 8
0	1	1	0	0	1	0	1	0
1	1	0	0	0	0	0	1	0
2	0	1	1	0	0	1	1	1
3	0	1	0	0	0	0	1	0
4	0	0	1	0	1	0	1	0
5	0	0	1	0	1	0	0	0
6	1	0	0	1	1	0	0	1
7	0	0	0	1	1	0	0	0

Tabla 2.2 Secuencias de ejemplo manejando 8 bits del puerto paralelo

Otro proceso importante es el **divisor de tiempo**. Este se puede llevar a cabo desde el software o en el hardware en caso de multiplexación por circuiteria. La computadora puede determinar el tiempo entre cada pulso que se manda a los motores para dar la velocidad deseada y también el control de tiempo en la multiplexación, en el desarrollo de la aplicación se detallará este punto.

2.3.2 ANALISIS Y DETERMINACIÓN DE LOS MÉTODOS PARA LLEVAR A CABO LOS PROCESOS NECESARIOS

Para llevar a cabo los procesos que se mencionaron anteriormente es necesario definir los métodos que ayuden a solucionar el problema de manera más rápida y eficiente. Debido a que el control de motores puede ser útil en muchas áreas y todas requieren una solución especifica, aunque algunas sean parecidas, sólo se pueden tomar como referencia los trabajos que ya se tienen; se tomarán sólo dos problemas a resolver, que son controlar los motores de una perforadora y un cortadora simples.

Primero se analizará el problema de la perforadora. El primer paso es generar la información necesaria sobre cada una de las perforaciones que realizará, creando la ***matriz de coordenadas***, ésta se puede obtener de dos maneras dependiendo del software: una forma sería desde un ambiente gráfico colocando cada una de las perforaciones por medio del ratón y otra forma más simple es llenando una tabla que será la matriz de coordenadas introduciendo la información por medio del teclado. Esto se explicara detalladamente en el apartado del desarrollo de la aplicación.

La matriz de coordenadas para la perforadora contiene los siguientes datos: *tipo de movimiento y coordenadas del motor 1, 2, 3 y 4*; el tipo de movimiento se

agrega a esta matriz para que el traductor de coordenadas sepa de qué manera se deben comportar los motores, que para este caso todos los movimientos serán en línea recta. La tabla 2.3 se tomará como ejemplo para explicación.

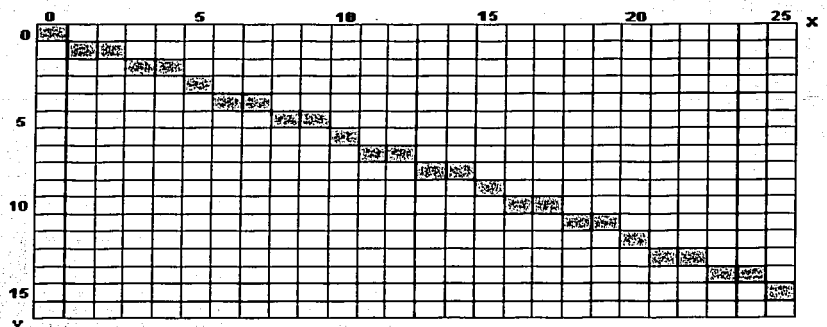
Movimiento	Motor 1	Motor 2	Motor 3	Motor 4
R	0	0	0	0
R	25	15	0	0
R	25	15	30	10
R	25	15	0	0
R	22	54	0	0
R	22	54	30	10
R	22	54	0	0
R	6	23	0	0
R	6	23	30	10

Tabla 2.3 Ejemplo de una matriz de coordenadas con movimientos en línea recta, para una perforadora

El traductor de coordenadas leerá el primer renglón de la matriz para saber el tipo de movimiento y las coordenadas a las que se debe mover, que de acuerdo a la matriz anterior es un movimiento en línea recta y todos los motores se deben mover a las posiciones iniciales de **0**; si los motores se encuentran ya en esas posiciones no se enviarán las secuencias de control y por lo tanto no se moverá ninguno. Después se leerá el segundo reglón y ahora se moverán los motores 1 y 2 al mismo tiempo, que para este caso de la perforadora son los ejes **X** y **Y** respectivamente.

En este ejemplo la matriz de coordenadas indica el movimiento de los motores en pares, ya que primero se moverán los motores **1** y **2** que representan los ejes **X** y **Y** y después los motores **3** y **4** que uno representa el motor del taladro y otro el motor que lo bajará y subirá.

La figura 2.7 representa el movimiento de estos motores.



X	Y
0.6	1
1.2	2
1.8	3
2.4	4
3	5
3.6	6
4.2	7
4.8	8
5.4	9
6	10
6.6	11
7.2	12
7.8	13

X	Y
8.4	14
9	15
9.6	16
10.2	17
10.8	18
11.4	19
12	20
12.6	21
13.2	22
13.8	23
14.4	24
15	25

Figura 2.7 Representación grafica del movimiento de los motores de los ejes Y y X, para el movimiento en línea recta

Como se puede apreciar en la figura 2.7, el sistema de coordenadas es el que manejan la mayoría de los lenguajes de programación, en el cual el eje positivo Y es hacia abajo. Para el movimiento en línea recta se utiliza la siguiente ecuación de la recta llamada *forma dos puntos*:[7]

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}\,(x - x_1)$$

[7] FUENLABRADA, Samuel, ***Matemáticas III Geometría Analítica***. México, editorial McGraw-Hill, 1995, p. 45.

Con esta fórmula se puede determinar la ecuación de una recta que pasa por dos puntos, que siguiendo con el ejemplo serán $\mathbf{P_1(\ 0,\ 0\ )}$ y $\mathbf{P_2(\ 25,\ 15\ )}$. Ahora se tiene que despejar la fórmula y sustituir valores, pero es muy importante saber qué variable será la dependiente y se determina sacando la distancia entre los puntos de cada eje, para el eje **X** será $\mathbf{(\ x_2 - x_1\ ) = (\ 25 - 0\ ) = 25}$ y para el eje **Y** será $\mathbf{(y_2 - y_1\ ) = (\ 15 - 0) = 15}$; así la variable dependiente será la de la distancia menor y la independiente la de la distancia mayor, en este caso **X** es la dependiente y se le darán valores a **Y**, como se observa en la tabla de la 2.7 anterior.

En el ejemplo no se especifican las unidades de desplazamiento de los motores y en caso de que no se puedan manejar fracciones se utilizará el método de redondeo, por ejemplo, cuando $\mathbf{Y = 1}$ la variable dependiente será $\mathbf{X=0.6}$ y se redondea a $\mathbf{X=1}$, después $\mathbf{Y = 2}$ y $\mathbf{X = 1.2}$; por lo tanto, se redondea y queda X = 1 como se ve en la gráfica de la figura 2.7; de manera que con valores de fracción menores o igual a **0.5** se trunca la fracción y con valores mayores se redondea al entero siguiente, es muy recomendable aplicar el redondeo sólo al número de pasos y no a la distancia para mayor precisión.

Y el último paso en la traducción de coordenadas es convertir la distancia que se moverá el motor a su equivalente en pasos y esto dependerá del desplazamiento que genere el motor cada que da un paso. Pro ejemplo, si el rotor tiene un engrane que desplaza 3 cm/rev y su resolución es de 30° por paso de manera que desplaza 0.25 cm por paso; ahora, si se quiere mover 7 cm el motor tendrá que dar 28 pasos, y ahora sí se pueden generar las secuencias de control para que el motor gire 7 cm o 28 pasos.

Para la cortadora, la matriz de coordenadas tendrá los siguientes datos: *el tipo de movimiento y las coordenadas de los motores 1 y 2, el ángulo inicial, el ángulo final, y las coordenadas de los motores 3 y 4,* los ángulos sólo se aplican al movimiento elíptico. Igual que en la perforadora tendrá movimientos en línea recta,

pero además podrá moverse en línea curva elíptica y una combinación de estos dos movimientos. La tabla 2.4 se tomará como ejemplo para explicación de la cortadora.

Movimiento	Motor 1	Motor 2	θ_1	θ_2	Motor 3	Motor 4
R	0	0	0	0	0	0
CR	20	25	3	1	50	10
R	30	25	0	0	50	8
CR	35	35	5	7	50	3
R	22	54	0	0	50	4
R	22	54	0	0	50	10
CR	22	54	1.3	3.8	50	7
CR	6	23	6.1	1	50	10
CR	6	23	8.2	5.4	50	0

Tabla 2.4. Ejemplo de una matriz de coordenadas con movimientos en línea recta y curva elíptica, para una cortadora

La matriz de coordenadas anterior indica primero que todos los motores deben posicionarse en 0, y después realizará un corte circular y recto a la vez (CR), para los motores 1 y 2 movimiento elíptico y recto para 3 y 4, ya que *los motores 1 y 2 serán para los ejes X y Y, el motor 3 será para girar el disco de la sierra y el 4 para la cabeza de la sierra.* El movimiento en línea recta se realiza igual que para la perforadora. Para el movimiento elíptico se necesita la posición de inicio, la posición final, el ángulo de inicio y el ángulo final de manera que se pueda trazar una elipse como la de la figura 2.8, tomando como base la segunda línea de la matriz anterior.

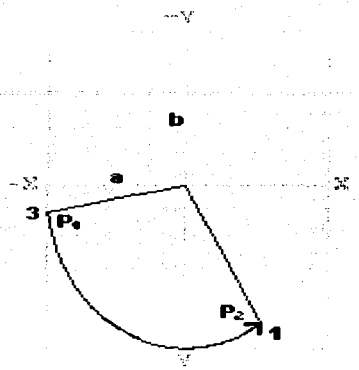


Figura 2.8 Representación gráfica del movimiento de los motores de los ejes X y Y para el movimiento en curva elíptica de 3 radianes a 1

Se puede apreciar la línea más obscura como la trayectoria que seguirán los motores de los ejes **X** y **Y**, comenzando en el punto $\mathbf{P_1(\ 0,\ 0\ )}$ con un ángulo de inicio de **3** radianes y terminando en el punto $\mathbf{P_2(\ 20,\ 25\ )}$ con un ángulo final de **1** radian. Debido a este sistema de coordenadas un movimiento angular positivo será en sentido horario y el movimiento anteriormente descrito será negativo o en sentido antihorario, ya que va de 3 radianes a 1 radian; para determinar el sentido sólo se tiene que observar si los ángulos van de mayor a menor es sentido *antihorario* y si van de menor a mayor es *horario*. Si se quiere trazar la otra parte de la elipse en sentido antihorario no se toma de **1** radian hacia **3** radianes sino que se tomará de $\mathbf{(2\ \pi_{rad.} + 1_{rad.}) = 7.2831}$ radianes hacia **3** radianes; véase la figura 2.9.

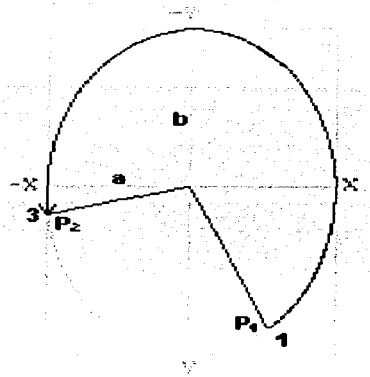


Figura 2.9 Representación grafica del movimiento de los motores de los ejes X y Y para el movimiento en curva elíptica de 1 radianes a 3 en sentido antihorario

La trayectoria de esta elipse se puede determinar mediante las siguientes *ecuaciones paramétricas:*[8]

$$x = a \cos \theta$$
$$y = b \operatorname{sen} \theta$$

Donde **a** es el radio de la circunferencia menor y **b** el radio de la circunferencia mayor, esto es porque una elipse se puede representar dentro de dos circunferencias una *circunferencia principal* y *una menor*, como se ve en la figura 2.10, donde **b** es la distancia de **O** hasta **B** y representa el radio de la circunferencia menor y **a** el la distancia de **O** hasta **A** y que es el radio de la circunferencia

[8] FUENLABRADA, Samuel, ***Matemáticas III Geometría Analítica***, México, editorial McGraw-Hill, 1995, p. 189.

principal. Para obtener la trayectoria de la elipse o parte de la elipse se le dan valores a θ en radianes y se mantienen **a** y **b** constantes.

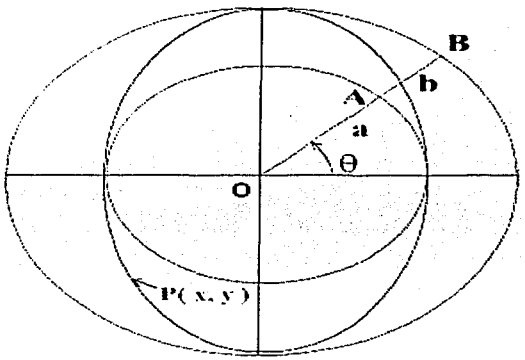


Figura 2.10 Gráfica de la representación de una elipse determinada mediante dos circunferencias

Para el proceso de *generación de secuencias* primero se analizarán las secuencias de salida al puerto paralelo, que éstas son independientes del tipo de aplicación, ya que para llegar a esta etapa ya se tiene la información de que motores y cuanto se van a mover. Si se quiere manejar de uno a cuatro motores en la aplicación se utilizaran los *8 bits* de datos del puerto paralelo, enviando las señales de control por pares, *un bit para el pulso de control y otro para el sentido de giro del motor*. La tabla 2.5 muestra las secuencias de control para cuatro motores unipolares.

BIT	Al motor 1	al 2	al 3	al 4	al 1	al 2	al 3	al 4
0	1	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0
3	0	1	0	0	0	0	0	0
4	0	0	1	0	0	0	1	0
5	0	0	1	0	0	0	0	0
6	0	0	0	1	0	0	0	1
7	0	0	0	1	0	0	0	0

Tabla 2.5 Secuencias de ejemplo para manejar 4 motores, uno por envío

Se puede apreciar en la secuencia anterior que es muy fácil manejar de 1 a 4 motores ocupando los 8 bits del puerto paralelo porque no se tiene que hacer ninguna multiplexación. El primer envío de señales es para el primer motor porque así lo indica el ***bit 0*** y el ***bit 1*** para el sentido del giro de este motor, en este caso en sentido de las manecillas del reloj. El segundo envío va para el motor 2 indicado por el ***bit 2*** y el ***bit 3*** para su correspondiente giro, en sentido horario, así sucesivamente hasta el cuarto motor. Después el quinto envío es para el motor ***1*** otra vez, pero ahora es en sentido antihorario ya que el ***bit 1*** tiene un ***0***, los demás envíos son para los motores 2,3 y 4 respectivamente y con sentido antihorario. De esta manera también se pueden mover más de un motor en un solo envío, como se muestra en la tabla 2.6.

BIT	Al motor 1 y 3	al 1,2 y 4	a los 4	a los 4	al 1 y 4	al 2 y 4	al 3	al 1 y 4
0	1	1	1	1	1	0	0	1
1	1	0	0	1	0	0	0	1
2	0	1	1	1	0	1	0	0
3	0	1	0	1	0	0	0	0
4	1	0	1	1	0	0	1	0
5	1	0	0	1	0	0	0	0
6	0	1	1	1	1	1	0	1
7	0	0	0	1	1	0	0	0

Tabla 2.6 Secuencias de ejemplo para manejar 4 motores, de 1 a 4 en un solo envío

Como se puede ver en el primer envío se moverán el motor 1 y 3 en sentido horario. En el segundo envío los motores 1 y 4 se moverá en sentido antihorario y el motor 2 en sentido horario. En el tercer envío los cuatro motores se moverán en sentido antihorario y en el cuarto envío en sentido antihorario. Después, en el quinto envío el motor 1 en sentido antihorario y el motor 4 en sentido horario.

Ahora, para manejar 5 motores o mas se tiene que hacer una multiplexación de señales, par hacer más eficiente este proceso se deben dividir los 8 bits que se pueden utilizar en 6 bits para las señales que van hacia los circuitos integrados y 2 bits para las señales de multiplexación; es importante recalcar que la multiplexación se lleva a cabo en el hardware, y será un arreglo de circuitos parte del controlador/driver, este arreglo se explica detalladamente en la etapa de potencia. La tabla 2.7 muestra un ejemplo de cómo podrían ser las secuencias.

BIT	Al motor 1	al 5	al 7 y 9	Al 11, 12 y 13	al 4	al 11 y 12	al 3	al 7,8 y 9
0	1	0	1	1	1	1	0	1
1	1	0	1	0	0	1	0	1
2	0	1	0	1	0	1	0	1
3	0	0	0	0	0	0	0	1
4	0	0	1	1	0	0	1	1
5	0	0	1	0	0	0	0	1
6	0	1	0	1	1	1	0	0
7	0	0	1	1	0	1	0	1

Tabla 2.7 Secuencias de ejemplo para manejar de 5 a 12 motores, solo 3 por envío, con señales multiplexadas

En la tabla anterior los bits 6 y 7 son las señales de multiplexación. En el primer envío las señales de multiplexación indican que se manejaran los tres primeros motores y en este caso solo el motor 1 girará en sentido horario. En el segundo envío los bits 6 y 7 indican, como si fueran 01 en binario, que se manejarán los motores 4, 5 y 6, por lo tanto, girará el motor 5 en sentido antihorario. En el tercer envío se indica que se manejarán los motores 7, 8 y 9 respectivamente y solo se moverán el 7 y 9 en sentido horario. En el cuarto envío se manejarán los motores 10, 11 y 12 que se moverán en sentido antihorario. Después en el quinto envío se moverá solo el motor 4 en sentido antihorario. Para entender mejor la multiplexación observese la tabla 2.8, se maneja un número en binario de dos bits y el **0** indica los primeros tres motores el **1** los siguientes tres y así sucesivamente.

Bit 7	Bit 6	Motores
0	0	1,2 y 3
0	1	4,5 y 6
1	0	7,8 y 9
1	1	10, 11 y 12

Tabla 2.8 Tabla de multiplexación para el manejo de 12 motores

También se podrían manejar mas de 12 motores pero, además de que la mayoría de las aplicaciones con motores no utilizan mas de 8 motores, no es muy recomendable ya que se pierde eficiencia. Para manejar más de 12 motores se reservan 4 bits para multiplexación, de esta manera se manejarán solo 2 motores por envío. En la tabla 2.9 se presenta una secuencia de ejemplo.

BIT	Al motor 1	al 3 y 4	al 15	al 4	al 1	al 2	al 3	al 4
0	1	1	1	0	1	0	1	1
1	1	1	0	0	0	0	0	1
2	0	1	0	1	0	1	0	0
3	0	1	0	1	0	0	0	0
4	0	1	1	0	0	1	0	0
5	0	0	1	0	0	0	1	0
6	0	0	1	1	0	0	0	1
7	0	0	0	1	0	1	0	0

Tabla 2.9 Secuencias de ejemplo para manejar de 13 a 32 motores, solo 2 por envío, con señales multiplexadas

En la tabla anterior los bits 4,5,6 y 7 son los bits de multiplexación. En el primer envío se indica que se manejarán los motores 1 y 2, pero sólo se moverá el motor 1 en sentido horario. En el segundo envío se manejarán los motores 3 y 4 y se moverán en sentido horario. En el tercer envío se manejarán los motores 15 y 16 pero solo se moverá el 15 en sentido antihorario . En el tercer envío se manejaran los motores 25 y 26 pero solo se moverá el 26 en sentido horario.

Para un segundo caso en que no se tienen circuitos integrados para generar los pulsos de control que van directo a las bobinas del motor, se tienen que generar los pulsos desde la computadora e irán directo al driver para darles la potencia necesaria, ahora no se tienen que enviar pares de bits, lo que se tiene que hacer es convertir la secuencia de control en los pulsos de control que van hacia las bobinas

de cada uno de los motores, uno por uno de forma multiplexada. Una vez que se sabe en que sentido y cuantos pulsos girará el motor y que para cada motor se haya creado una secuencia de control, si el motor no se moverá se debe excluir esta. Por ejemplo para manejar motores unipolares con 4 bobinas, se toman los 4 primeros bits para los pulsos de control y los 4 restantes para la multiplexación, de tal forma que se pueden manejar hasta 16 motores, en la tabla se 2.10 presenta un ejemplo de cómo se enviarían las señales del puerto paralelo al driver.

BIT	envío 1	envío 2	envío 3	envío 4	envío 5	envío 6	envío 7	envío 8
0	1	0	0	1	1	0	0	0
1	0	0	0	0	0	0	1	0
2	0	0	1	0	0	1	0	1
3	0	1	0	0	0	0	0	0
4	0	1	1	0	1	0	1	0
5	0	0	1	1	1	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

Tabla 2.10 Ejemplo de secuencias para cuando no se utilizan circuitos integrados para generar estas

El primer envío es para el motor 0, el segundo es para el motor 1 y los siguientes para 3, 2, 3, 0, 1 y 0 respectivamente. La secuencia de pulsos se maneja desde el software, ya sea que se tengan definidas como matrices o que se consulte una base de datos que contengan las secuencias especificas para cada tipo de motor.

CAPÍTULO III

DISEÑO

3.1 DISEÑO DE LA APLICACIÓN

El ***software*** debe realizar los procesos necesarios para llevar a cabo el control del motor, estos procesos fueron definidos en la sección 2.1 y en la figura 2.6 se muestra el diagrama a bloques. El software debe ser muy flexible, es decir, que se debe ajustar a los motores y al graficador, permitiendo una configuración especifica para cada tipo de motor dependiendo de sus características y su comportamiento con el graficador.

Como se explico en la sección 2.4.1 el primer proceso a realizar es el de la ***generación de coordenadas***, para este proceso el software debe proveer de una interfaz (de preferencia gráfica) para comunicarse con el usuario, esta debe permitir que se introduzcan los datos, ya sea mediante el teclado o ratón de la computadora, si es por teclado se debe tener una tabla en la cual el usuario teclee las coordenadas necesarias para formar la *matriz de coordenadas* de la que se hablo en la sección 2.4.1, y si es por ratón se debe proveer de un lienzo, de preferencia cuadriculado para dar una referencia a las coordenadas, para que el usuario con solo hacer un clic o arrastrar el ratón proponga las coordenadas deseadas, y el software debe crear la *matriz de coordenadas*.

También es importante recalcar que el software debe dar las unidades de medición dependiendo de las que maneje en el graficador.

Una vez que se tiene la matriz de coordenadas el software debe permitir el salvado de la información, de manera que si se tiene que realizar el mismo trabajo

varias veces solamente se haga referencia a un archivo para leer su información y ejecutar este trabajo ***n*** veces.

Cuando ya se ha salvado la información el siguiente paso será la traducción de coordenadas, para determinar que motores y cuanto deben girar dependiendo de su configuración. Como se explico anteriormente este es un proceso muy importante y se debe tener cuidado para no cometer errores costosos. Después el software debe generar las secuencias de acuerdo al tipo de motor y por ultimo se le debe de asignar los tiempos según sea necesario.

3.1.1 DISEÑO DEL SOFTWARE DE LA PERFORADORA

El software de la perforadora se realizó con una combinación de programación en ***java jdk1.3.1***, ***Visual Basic*** y ***C++***. Como todo software bien estructurado este debe contar un formulario MDI, y este a su vez debe contener una barra de herramientas con las diferentes opciones para manejar la perforadora. En la figura 3.1 se muestra el formulario MDI y su respectiva barra de herramientas.

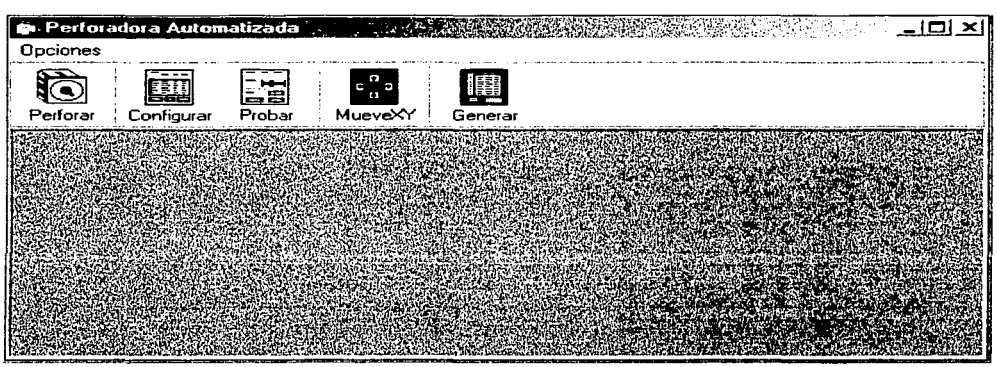


Figura 3.1 Formulario MDI del sistema Perforadora Automatizada

Como se puede apreciar en la figura 3.1 la barra de herramientas cuenta con cinco iconos, de los cuales el primero muestra el formulario que permite ***leer*** y ***ejecutar*** un archivo con extensión **.Perf**, que contenga una matriz de coordenadas, de manera que se realice el proceso de perforación ***n*** veces. Para abrir un archivo se utiliza el cuadro de dialogo estándar de Windows, como el de la figura 3.2.

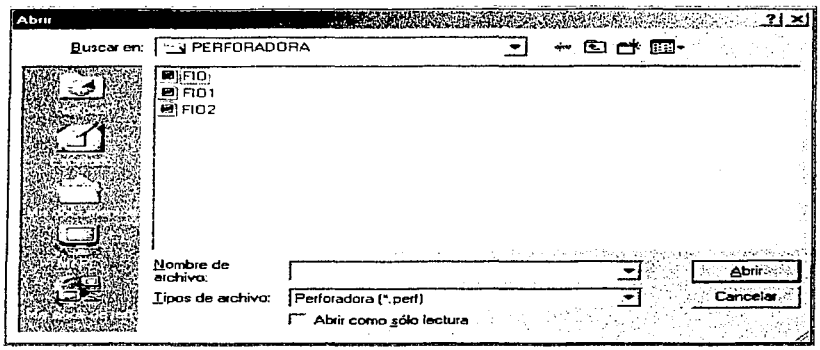


Figura 3.2 Dialogo Abrir archivo

Después de que se ha seleccionado el archivo se muestra una pantalla con la información de la matriz de coordenadas en un *Listbox*, esta pantalla también contiene dos botones, uno que sirve para abrir otro archivo y otro que ejecuta las perforaciones de la matriz de coordenadas cargada actualmente, se puede ejecutar una y otra vez. La figura 3.3 muestra la pantalla de este formulario.

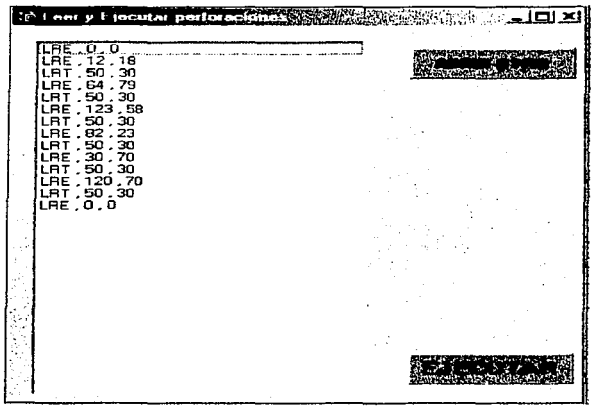


Figura 3.3 Formulario Leer y ejecutar perforaciones

Como se puede ver la matriz de coordenadas es diferente a la que se propuso en la sección 2.4, debido a que se debe optimizar tanto los procesos como el almacenamiento en memoria física y en la memoria RAM, de manera que el programa sea lo más eficiente posible. En esta matriz de coordenadas se eliminan los ceros, y se da un identificador de movimiento ya sea **LRE** para mover los ejes (motores 1 y 2) a la posición que se indica después de éste ó **LRT** que indica que se moverán el taladro y su elevador (motores 3 y 4) a la posición correspondiente.

Lo primero que realiza el código de este formulario es leer la configuración de los motores que esta contenida en un archivo llamado "motConfig.Txt", este contiene la configuración para cada motor a utilizar, la edición de este archivo será detallada más adelante. El siguiente código realiza esta operación.

Código para leer la configuración de los motores.

```
1:      Private Sub Form_Load()
2:      Dim m(3, 2) As Double
3:      Dim x1, x2, x3 As Double
```

```
4:          a = 0
5:          On Error GoTo señal
6:          Open App.Path & "\motConfig.txt" For Input As #1   ' Abre el archivo para recibir los datos.
7:            While Not EOF(1)
8:               Input #1, x1, x2, x3
9:               If a > 3 Then
10:                Close #1
11:                MsgBox "ERROR de configuración: número de motores erróneo"
12:                Form5.Show vbModal
13:                End
14:              End If
15:              m(a, 0) = x1
16:              m(a, 1) = x2
17:              m(a, 2) = x3
18:              a = a + 1
19:           Wend
20:           Close #1
21:           For i = 0 To a - 1
22:            dp(i) = m(i, 0) * m(i, 1) / 360
23:           Next
24:        If m(0, 2) > m(1, 2) Then ret(0) = m(0, 2) Else ret(0) = m(1, 2)
25:     If m(2, 2) > m(3, 2) Then ret(1) = m(2, 2) Else ret(1) = m(3, 2)
26:     Exit Sub
26:     señal:
27:        MsgBox Err.Description
28:        Err.Clear
29:     End Sub
```

De la línea 6 a la 20 se abre el archivo y se leen sus datos renglón por renglón hasta el fin del archivo y se almacenan en una matriz de 4x3, que en el código se muestra como **m(3,2),** porque son 4 motores que van de 0..4 y 3 datos de cada motor que van del 0..2. El primer dato de cada motor es la precisión del motor llamada ***ángulo/paso***, el segundo dato es el avance por vuelta medido en milímetros y llamado ***mm/rev***, y el ultimo es el tiempo de espera antes de mandar la

siguiente secuencia de control al motor de manera que este sea rápido y que trabaje de manera correcta, es decir, que la frecuencia de pulsos no afecte el funcionamiento del motor provocando movimientos no deseados. La tabla 3.1 muestra un ejemplo de esta matriz.

	Ángulo/paso	mm/rev	Retardo/paso
Motor 1	1.875	20.25	100000
Motor 2	7.5	48	100000
Motor 3	15	144	100000
Motor 4	1.8	30.22	100000

Tabla 3.1 Matriz de configuración para los motores utilizados en la aplicación

De la línea 21 a la 23 se inicializa un vector llamado **dp** con la información de la distancia por paso de cada motor. Y las líneas 24 y 25 sirven para determinar que retardo se utilizará para cada par de motores, debido a que se moverán de dos en dos, se hace una comparación del retardo del motor 1 con el del 2 y el del 3 con el del 4 almacenado estas comparaciones en un vector llamado **ret**.

El siguiente paso es la ejecución de las perforaciones. Cuando ocurre un evento *click* en el botón ejecutar se manda llamar a un procedimiento llamado **ClacDist,** en este procedimiento se calculan las distancias y se manda llamar a al procedimiento de secuencias llamado **sec** que a su vez manda llamar al procedimiento para enviar los datos al puerto paralelo llamado **envia**. A continuación se presenta el código del procedimiento **calcDist**.

Código del procedimiento ClacDist

```
1:    Private Sub CalcDist()
2:    Dim X, Y, , xa, ya, px, py, n1, n2 As Double
```

```
3:      Dim posX1, posY1, posX2, posY2 As Integer
4:      Dim mov As String
5:      xa = 0                  'Variable que sirve como referencia para la posición anterior de X
6:      ya = 0                  'Variable que sirve como referencia para la posición anterior de Y
7:      posX1 = 0               'Posición actual del eje X
8:      posY1 = 0               'Posición actual del eje Y
9:      posX2 = 0               'Posición actual del eje X del Taladro
10:     posY2 = 0               'Posición actual del eje Y del Taladro
11:     L.Clear
12:     Open App.Path & "\" & MDIForm1.CD1.FileTitle For Input As #1        'Abre el archivo
13:       Input #1, linea, X, Y, p                                  'Lee la primera linea para configuración
14:       While Not EOF(1)                                          'Ciclo para leer el archivo linea por linea
15.         Input #1, mov, X, Y                                     'Lee la matriz de coordenadas desde el archivo
16:         L.AddItem mov & ", " & X & ", " & Y                     'Muestra la matriz de coordenadas en el listbox
17:         If mov = "LRE" Then                                     'Inicializa el modo de funcionamiento
18.           g = 0                                                 'para trabajar con los ejes X-Y
19.           h = 0
20:         Else
21:           If mov = "LRT" Then                                   'Inicializa el modo de funcionamiento
22.             g = 1                                               'para trabajar con los ejes X-Y del Taladro
23:             h = 2
24:           End If
25:         End If
26:       For a = g To h                                            'Ciclo que activa el modo de funcionamiento
27:       Select Case a
28:         Case 2                                      'En en este caso se mandan a la posición 0 a Y y X
29:           X = 0                                     'Para subir el taladro y se inicializan las posiciones
30:           Y = 0                                     'Anteriores
31:           xa = posX2
32:           ya = posY2
33:         Case 1                                      ' En este caso se bajara el taladro y se inicializan
34:           xa = posX2                                'las posiciones anteriores.
35:           ya = posY2
36:         Case 0                                      'En este caso se moverán los ejes Y y X
37:           xa = posX1
38:           ya = posY1
39:       End Select
40:       n1 = (X - xa)   'distancia a recorrer en X
```

```
41:         n2 = (Y - ya)   'distancia a recorrer en Y
42:         Select Case a
43:         Case 1, 2
44:          px = Round((n1 / dp(2)), 0)   'Número de pasos a recorrer en eje X Taladro
45:          py = Round((n2 / dp(3)), 0)   'Número de pasos a recorrer en eje Y Taladro
46:         Case 0
47:          px = Round((n1 / dp(0)), 0)   'Número de pasos a recorrer en eje X
48:          py = Round((n2 / dp(1)), 0)   'Número de pasos a recorrer en eje Y
49:         End Select
50:         If (Abs(px) - Abs(py)) > 0 Then              'Checar si es mayor la distancia en X
51:          mayor = "Mayor x"                           'Para graficar se dan valores a X
52:          If px > 0 Then inc = 1 Else inc = -1  'Para mover en sentido (-)el incremento –1 y (+)  +1
53:          pxa = 0
54:          pya = 0
55:          For i = 0 To px Step inc                    'Ciclo para graficar la linea recta
56:           If (xa = posX1) And (ya = posY1) Then
57:            sec pxa, pya, i, Round((py / px) * i, 0), 0  'Manda al procedimiento sec enviándole las
58:           Else                                  ' posiciones de pulso anteriores y actuales a los ejes X-Y
59:            sec pxa, pya, i, Round((py / px) * i, 0), 4  'sec para los ejes X-Y del Taladro
60:           End If
61:           pxa = i                                  'Actualiza la X anterior
62:           pya = Round((py / px) * i, 0)            'Actualiza la Y anterior
63:          Next
64:         Else
65:          If  (Abs(py) - Abs(px)) >= 0 Then       'En caso de que sea mayor o igual a  la distancia
66:           mayor = "Mayor  y o igual"              ' en Y .Para graficar se dan valores a Y
67:           If py > 0 Then inc = 1 Else inc = -1
68:           pxa = 0
69:           pya = 0
70:           For i = 0 To py Step inc
71:            If (xa = posX1) And (ya = posY1) Then
72:                    sec pxa, pya, Round((i / py) * px, 0), i, 0
73:            Else
74:                    sec pxa, pya, Round((i / py) * px, 0), i, 4
75:            End If
76:            pxa = Round((i / py) * px, 0)
78:            pya = i
79:           Next
```

```
80:             End If
81:           End If
82:           Select Case a
83:           Case 1, 2                    'Caso para el Taladro se
84:             xa = xa + (px * dp(2))     'Actualiza la distancia real en la que se encuentra el exe Y y X,
85:             ya = ya + (py * dp(3))     'debido a que en la otra actualización se pierde precisión
86:           Case 0                       'Caso para los ejes X-Y
87:             xa = xa + (px * dp(0))
88:             ya = ya + (py * dp(1))
89:           End Select
90:           Select Case a
91:             Case 1, 2                  'Caso para el Taladro se actualizara las posiciones anteriores
92:               posX2 = xa               'Con la mínima perdida de precisión
93:               posY2 = ya
94:             Case 0                     'Caso para los ejes X-Y se actualizara las posiciones anteriores
95:               posX1 = xa               'Con la mínima perdida de precisión
96:               posY1 = ya
97:           End Select
98:         Next
99:       Wend
100:      Close #1      ' Cierra el archivo.
101:      p1.Pulso 0    'Desactiva el puerto
102:  End Sub
```

Se dice que un código muy extenso es un *código ciego*, debido a que es muy difícil llevar la secuencia de los pasos a realizar y por tanto es difícil de entender, pero los comentarios son muy útiles y esenciales dentro de todo programa, se explicará el código lo mejor posible.

De la línea 1 a la 11 se declaran e inicializan las variables que se utilizarán, obsérvense los comentarios que indican para que sirve cada variable. En la línea 12 se abre el archivo a procesar, en la línea 13 se lee el primer renglón del archivo como referencia. De la 14 a la 99 se encuentra el ciclo que leerá línea por línea el archivo e irá ejecutando la información que reciba. De la línea 17 a la 25 se

inicializa el modo de funcionamiento, ya sea **LRE** para mover los ejes X-Y ó **LRT** para mover los motores X-Y del taladro.

De la 26 a la 98 se encuentra el ciclo que activa el funcionamiento. Se tiene un *select case* con los tres casos posibles, en el primero se pretende elevar el taladro después de que se ha bajado para perforar, inicializando en cero las posiciones a las que se desea subir el motor que serían (0,0) y se actualizan las posiciones anteriores de Y y X. En las líneas 40 y 41 se calculan las distancias a recorrer en Y y X, se resta la distancia anterior de la distancia deseada. Una vez que se tiene las distancias a recorrer el siguiente paso es convertir estas distancias al número de pasos que dará cada motor, esto se realiza de la línea 43 a la 49 en las que se tienen dos casos, el primer caso (líneas 44 y 45) convierte el número de pasos por dar según la configuración del par de motores para el taladro y el segundo caso (líneas 47 y 48) convierte el número de pasos para los motores de los ejes X-Y, estos cálculos se hacen redondeando el resultado de la división de la distancia deseada entre la distancia por paso de cada motor, esta ultima distancia leída del archivo de configuración, se debe tomar en cuenta que con el redondeo se pierde precisión pero este problema se resuelve más adelante.

Para llevar a cabo el movimiento de los motores de manera adecuada se utiliza el método para graficar una línea recta, para la ecuación de la línea recta se debe tener una variable dependiente y una independiente, estas variables servirán para mover el par de motores en curso identificándolos con Y y X. Para determinar cual será la variable independiente se compara la distancias del eje X con la del eje Y y a la que sea mayor se le darán valores, y por tanto la otra será la variable dependiente. Este proceso de selección se realiza en la línea 50 para saber si X es la variable independiente, en caso de que lo sea el siguiente paso es conocer el incremento que se le irán asignado valores para graficar la línea recta, en la línea 52 se determina si el incremento es -1 ó $+1$, dependiendo de que el número de pasos a

recorrer sea negativo ó positivo respectivamente, para girar el motor en sentido horario ó antihorario.

De la línea 55 a 63 se tiene un ciclo para graficar la línea recta, lo que se hace en este ciclo es darle valores a la variable independiente (en este caso X) que se van incrementando o decrementando según el valor de inc, de manera que se obtengan los valores para graficar la línea recta, estos valores son los que se enviarán al procedimiento **sec**. Antes de enviar los valores se debe determinar a que motores se les enviarán las secuencias, el condicional *if* de la línea 56 a la línea 60 realiza esta operación llamando al procedimiento **sec** enviando como ultimo parámetro el bit en inicial que seria **0** para el primer par de motores ocupando los cuatro primeros bits del puerto paralelo y **4** para el segundo par de motores ocupando los últimos cuatro bits. En este ciclo se utilizan dos variables auxiliares para realizar el recorrido de la posición inicial a la posición final, que son **pya** y **pxa** para los ejes Y y X respectivamente, y en las líneas 61 y 62 se actualizan para el siguiente movimiento.

En la l línea 65 se encuentra el otro caso, cuando la distancia en Y es mayor, por lo tanto de la línea 65 a la 80 se realizan las mismas operaciones que se hacen para cuando la distancia en X es mayor (líneas de la 50 a la 64).

Por ultimo antes de mover el siguiente par de motores a la posición leída desde el archivo se deben actualizar las posiciones actuales de los ejes X-Y en curso, pero ahora de forma real evitando la perdida de precisión. De la línea 82 a la 89 se tienen dos casos, el primero para actualizar a la distancia real de los ejes X-Y del taladro y el segundo para actualizar la distancia de los ejes X-Y, esto se calcula sumándole a la distancia anterior ya registrada la conversión del calculo de la ultima distancia recorrida, que esta ultima es la multiplicación del número de pasos que dio el motor por su distancia por paso. Después simplemente se actualizan las variables de posición actual **posX1** y **posY1** en caso de que se hayan movido los

motores de los ejes X-Y ó **posX2** y **posY2** en caso de que se hayan movido los ejes X-Y del taladro. Y todo el proceso encerrado por el ciclo *while* (líneas de la 14 a la 99) se repetirá para cada renglón leído del archivo.

Ahora se analizará el procedimiento **sec** que sirve para convertir la secuencia de pulsos que se enviaran al puerto paralelo, determinando si avanza un motor y en que sentido.

Procedimiento para generar las secuencias de control y mandar los pulsos al puerto paralelo

```
1:    Private Sub sec(pxa, pya, X, Y, bit)     'las siguentes lineas determinan si avanza un motor
2:    Dim s1a, s1b As Integer                  'y en que sentido
3:    'SENTIDO
4:    s1a = s(bit + 1)      'Variable auxiliar para saber si hay un cambio de sentido en el primer par
5:    s2a = s(bit + 3)      'Variable auxiliar para saber si hay un cambio de sentido en el segundo par
6:    If pxa <> X Then                  'Activa  o desactiva el sentido para el eje X
7:       If (X - pxa) > 0 Then s(bit + 1) = 1 Else s(bit + 1) = 0
8:    End If
9:    If pya <> Y Then                  'Activa o desactiva el sentido para el eje Y
10:      If (Y - pya) > 0 Then s(bit + 3) = 1 Else s(bit + 3) = 0
11:   End If
12:
13:   If (s1a <> s(bit + 1)) Or (s2a <> s(bit + 3)) Then    'Si hay un cambio de sentido se
14:      envia bit                      'activan o desactivan los btis asignados para dar el sentido
15:         For i = 1 To 1000000        'del giro del motor y se le da un retardo de tiempo
16:         Next
17:   End If
18:
19:   'PASO
20:   If pxa <> X Then              'Determina si se mueve un paso el motor del eje X
21:      s(bit) = 1
22:   Else
23:      s(bit) = 0
24:   End If
25:   If pya <> Y Then              'Determina si se mueve un paso el motor del eje Y
26:      s(bit + 2) = 1
27:   Else
```

```
28:         s(bit + 2) = 0
29:       End If
30:
31:       'ENVIAR PULSO
32:       envia bit                'Envia la secuencia en forma de pulso al puerto paralelo
33:       s(bit) = 0               'Desactiva el bit de movimiento del motor X
34:       s(bit + 2) = 0           'Desactiva el bit de movimiento del motor Y
35:       envia bit                'Envia la secuencia en forma de pulso al puerto paralelo con los
36:                                'bits de movimiento desactivados para crear un flanco de bajada
37:       End Sub
```

Este procedimiento recibe como parámetros **pxa** y **pya** que representan las posiciones anteriores de el par de motores a mover, otros dos parámetros que recibe son **Y** y **X** que representan la siguiente posición a la que se moverán el par de motores, y el ultimo parámetro llamada **bit** que indica el bit inicial, este sirve como referencia para mandar la secuencia al par de motores de los ejes X-Y si su valor es **0** y si es **4** manda la secuencia al par de motores del taladro.

El primer paso que realiza este procedimiento es inicializar las variables **s1a** y **s2a** (líneas 4 y 5), estas variables servirán como auxiliares para conocer si hay un cambio de sentido en algún motor, se les asigna el sentido actual que tiene activado cada motor. Después se debe determinar el sentido de cada motor, con un **1** ó un **0** asignado los bits de sentido. Para determinar el sentido primero se analiza si el motor se va mover, es decir, si la diferencia entre la posición anterior y la posición deseada es diferente de **0** (líneas 6 y 9), si se va mover se analiza en que sentido con una la condición de la línea 10 y 7, para los ejes Y y X respectivamente, si la posición deseada menos la posición anterior es mayor que **0** se activa en un sentido y si es menor que **0** se activa en sentido contrario.

El siguiente paso es determinar si hay un cambio de sentido, analizado la condición de que si **s1a** es diferente de el bit de sentido **s(bit+1)** ó que **s2a** es diferente de **s(bit+3)** , si se cumple la condición hay un cambio de sentido y por lo

tanto se activan los bits de sentido mandando llamar el procedimiento **envia,** que es el procedimiento que convierte el vector que simula los 8 bits del puerto llamado **s** y los convierte de binario a decimal de manera que se pueda enviar al puerto paralelo.

El ultimo paso sería enviar los pulsos de movimiento de sus respectivos bits almacenados en el vector **s**, para esto se debe llamar a el procedimiento **envia**. Es importante observar las líneas 32 a la 35, estas líneas simulan el pulso necesario para que el controlador actué, de manera que cuando se llame al procedimiento **envia** en la línea 32 se realice un flanco de subida y cuando se vuelva a llamar en la línea 35 se realice el flanco de bajada, en las líneas 33 y 34 se asigna un **0** a los bits de movimiento para que se pueda dar este flanco de bajada, como se puede ver en la figura 3.4.

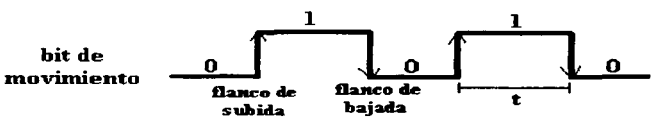


Figura 3.4 Representación de una señal cuadrada en su flanco de subida y bajada

El procedimiento **envia** solamente convierte de binario a decimal el vector s y manda el resultado al puerto paralelo para generar los pulsos que necesita el controlador.

Procedimiento para enviar los pulsos al puerto

```
1:      Private Sub envia(bit)
2.      Dim t As Double
3:      If bit = 4 Then t = ret(1) Else t = ret(0)      'Determinar que retardo se va usar
4:                                  ' este fue calculado en el procedimiento form_load en las lineas 24 y 25
5:      conv = 0
6:      For a = 0 To 7                                  'Convertir  de binario a decimal el vector s
7:         conv = conv + (s(a) * 2 ^ (a))
8:      Next
```

```
9:
10:         p1.Pulso conv     'Envía el dato al puerto paralelo por medio del control 11:ActiveX P
12:
13:         For i = 1 To t                              'Genera un retardo
14:         Next
15:
16:     End Sub
```

El primer paso en este procedimiento es determinar el retardo que se va usar. El segundo paso es convertir de binario a decimal el vector **s**, esto se realiza en las líneas de la 5 a la 8, simplemente va convirtiendo cada bit en su valor decimal elevando el valor del bit a la potencia a, donde a va de 0..7, y los resultados se acumulan en la variable **conv**. Una vez que se ha convertido a decimal se envía el dato contenido en **conv** al puerto paralelo y se le da un retardo de tiempo.

Como ***Visual Basic*** no cuenta con un control para mandar bytes al puerto paralelo, solo tiene la opción para imprimir que es muy diferente, fue necesario crear uno del tipo ActiveX desde ***Visual C++***. El control es muy sencillo ya que solo requiere de dos métodos, uno para asignar el número de puerto al que se van a enviar los bytes y otro para enviarlos, este control cuenta con otros métodos propiedades y características que no son importantes y no se mencionarán.

Código del método "Puerto" para activar el puerto a utilizar del control ActiveX "P"

```
1:      // CCtrlPCtrl::Puerto - puerto paralelo a utilizar;
2:
3:      void CPCtrl::Puerto(int p)
4:      {
5               P_P=p;          'Asigna el número del puerto a la variable P_P
6:      }
```

El método **Puerto** solo almacena su parámetro de tipo entero en la variable P_P, esta variable se utilizará en el método **Pulso**.

Código del método "Pulso" para enviar un byte a un puerto

```
1:      // CCtrlPCtrl::Pulso - Manda el pulso al puerto paralelo
2:
3:      void CPCtrl::Pulso(int n)
4:      {
5:              _outp( P_P , n );       'Comando para enviar un byte "n" a un puerto contenido en la
6:      }                               'variable "P_P"
```

Este método utiliza el comando **_outp(** *Puerto*, *Byte*), que sirve para mandar un byte a un puerto valido.

Y eso es lo que realiza y contiene el formulario para leer y ejecutar un archivo de tipo **Perf**. Siguiendo con el análisis del formulario MDI el segundo icono abre un formulario para crear o modificar la configuración de los motores, este tiene una presentación como se muestra en la figura 3.5.

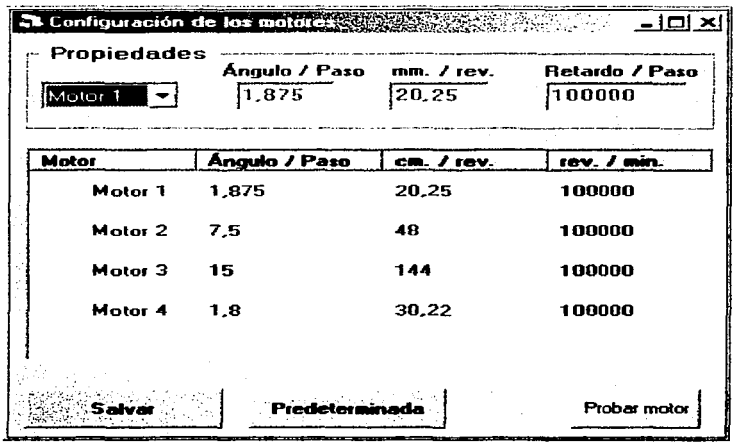


Figura 3.5 Formulario Configuración de motores

En este formulario se utiliza un procedimiento llamado **iniLvw** para almacenar los datos en un *listview*, que servirá como apoyo para crear o modificar la configuración, seleccionando con el ratón, dentro del *listview*, el renglón del motor a modificar su configuración se llenan los *TextBox* de la parte superior del formulario con los datos de ese renglón, de forma que se puedan modificar.

Código del procedimiento iniLvw para leer los datos del archivo motConfig.txt y poder modificarlos

```
1:      Private Sub iniLvw()
2:      Dim x1, x2, x3 As Double
3:      ListView1.ListItems.Clear
4:      a = 0
5:      On Error GoTo señal
6:      Open App.Path & "\motConfig.txt" For Input As #1   'Abre el archivo para recibir los datos.
7:        While Not EOF(1)                                 'Ciclo para leer renglón por renglón  el archivo
8:          Input #1, x1, x2, x3                           'Lee un renglón del archivo
9:          a = a + 1                                                'Se incrementa el contador de motores
10:         ListView1.ListItems.Add a, , "Motor " & a & " "          'Asigna el número de motor y los
11:         ListView1.ListItems.Item(a).ListSubItems.Add 1, , x1     'datos leidos del archivo
12:         ListView1.ListItems.Item(a).ListSubItems.Add 2, , x2     'al listview
13:         ListView1.ListItems.Item(a).ListSubItems.Add 3, , x3
14:       Wend
16:       Close #1
17:     Exit Sub
18:     señal:
19:       MsgBox Err.Description
20:       Err.Clear
21:     End Sub
```

Este procedimiento lee los datos del archivo renglón por renglón y los va almacenando en un *listview* para que sean manipulados.

Una vez que se tiene la configuración deseada el botón salvar guarda la información en el archivo.

Código del Botón salvar para guardar la configuración en el archivo motConfig

```
1:      Private Sub Salvar_Click()
2:        Open App.Path & "\motConfig.txt" For Output As #1        'Abre el archivo para operaciones
3:        For b = 1 To a                                           'de salida.
4:        Write #1, CDbl(ListView1.ListItems.Item(b).ListSubItems.Item(1)),   'Escribe en el archivo
5:                  CDbl(ListView1.ListItems.Item(b).ListSubItems.Item(2)),   ' los datos contenidos 6:
CDbl(ListView1.ListItems.Item(b).ListSubItems.Item(3))      'en el listview para
7:        Next                                                     'actualizar la config..
8:        Close #1   ' Cierra el archivo.
9:      End Sub
```

La información contenida en el *listview* se guarda en el archivo para configuración. Este formulario cuenta con otro botón que abre el formulario para hacer pruebas, que también esta en la barra de herramientas de la MDI como el tercer icono. Este formulario tiene la presentación de la figura 3.6.

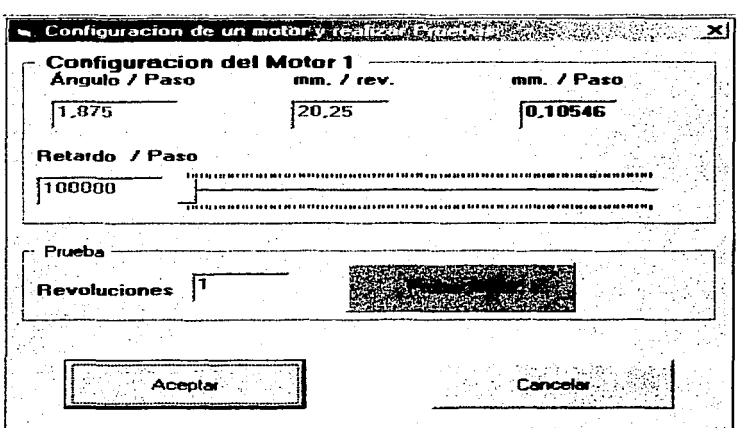


Figura 3.6 Formulario Configuración y pruebas de un motor

Este formulario sirve para probar la configuración de un motor, dado las características y los valores que se quieran probar, en figura 3.6 se pone como ejemplo un motor con un ángulo de paso de **1.875°** que se probará con un avance de **20,25** mm/rev y un retardo de tiempo de **100000**.

El código que realiza el proceso de prueba para un motor cuando ocurre un evento *click* en el botón Probar es el siguiente.

Código del evento click en el botón Probar

```
1:    Private Sub Probar_Click()
2:    Dim t As Double
3:    Dim d, f, g As Integer                    'variables d, f y g para mandar el valor al puerto
4:    Select Case Form5.Combo1.ListIndex        'según el motor que se utilice
5:     Case 0
6:      d = 3                                   'Para enviar 00000011
7:      f = 2                                   'Para enviar 00000010
8:      g = 1                                   'Para enviar 00000001
9:     Case 1
10:     d = 8                                   'Para enviar 00001100
11:     f = 12                                  'Para enviar 00001000
12:     g = 4                                   'Para enviar 00000100
13:    Case 2
14:     d = 48                                  'Para enviar 00110000
15:     f = 32                                  'Para enviar 00100000
16:     g = 16                                  'Para enviar 00010000
17:    Case 3
18:     d = 192                                 'Para enviar 11000000
19:     f = 128                                 'Para enviar 10000000
20:     g = 64                                  'Para enviar 01000000
21:   End Select
22:
23:   t = Text1(3)                              't ➔ almacena el tiempo de retardo
24:   p = Text1(4) * (360 / Text1(0))           'p ➔ almacena el número de pasos por dar
25:                                             ' según el número de revoluciones
26:   For a = 1 To p          'Ciclo para mandar los pulsos de manera que el motor gire en un sentido
27:    p1.Pulso (d)                             'flanco de subida ↑
28:    retardar t                               'Genera un retardo
```

```
29:     p1.Pulso (f)                    'flanco de bajada ↓
30:     retardar t                      'Genera un retardo
31:   Next
32:
33:   p1.Pulso (0)
34:   retardar (t * 100)
35:
36:   For a = 1 To p        'Ciclo para mandar los pulsos para que el motor gire en sentido contrario
37:     p1.Pulso (g)                    'flanco de subida↑
38:     retardar t                      'Genera un retardo
39:     p1.Pulso (0)                    'flanco de bajada ↓
40:     retardar t                      'Genera un retardo
41:   Next
42:   End Sub
```

Este código es muy sencillo ya que solamente se trata de dar una secuencia predeterminada para mover el motor que se desee. De la línea 4 a la 21 se tienen los casos para probar cualquiera de los cuatro motores, ahí se asignan los valores predeterminados que se enviaran al puerto paralelo de manera que se pueda crear un pulso, en las líneas 23 y 24 se almacena el factor de retardo que se utilizará para mover el motor en curso y se calcula el número de pasos dependiendo del ángulo/paso del motor y el número de revoluciones que se desean probar. De la línea 26 a la 31 se encuentra el ciclo que hará que el motor gire según la prueba propuesta, primero mandara el dato al puerto de manera que se genere un flanco de subida, después generara el retardo, luego mandara otro dato al puerto para generar un flanco de bajada y por ultimo le da otro retardo, esto se repetirá desde 1 hasta el número de pasos que debe dar. De la línea 36 a la 41 se repite este ciclo pero modificado para que gire el motor en sentido contrario.

Otro icono que se encuentra en la barra de herramientas de la MDI abre el formulario, que se muestra en la figura 3.7, para mover un motor de los ejes X-Y poco a poco de forma personalizada.

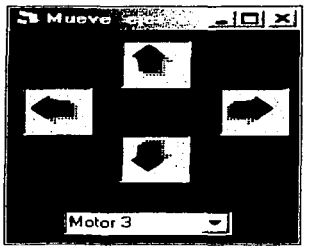


Figura 3.7 Formulario para Mover ejes X-Y manualmente

Este proceso es útil cuando los ejes no están en las posiciones iniciales o para cuando se quiera colocar los ejes en otra posición manualmente.

Código del procedimiento Mover y el evento MouseMove de un objeto Picture

```
1:      Private Sub mover (d1 As Integer, d2 As Integer)
2:      For a = 1 To p
3:        p1.Pulso (d1)                 'Envía el dato para generar flanco de subida
4:        retardar                      'Genera un retardo
5:        p1.Pulso (d2)                 'Envía el dato para generar flanco de bajada
6:        retardar                      'Genera un retardo
7:      Next
8:      End Sub
9:
10:
11:   Private Sub Picture1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
12:         If Button = 1 Then mover 3, 2          'Llama al procedimiento mover
13:   End Sub
```

El procedimiento **mover** genera la secuencia de pulsos para girar el motor y es llamado desde el evento *MouseMove* de cada uno de los cuatro objetos *Picture* que representan la dirección en la que se quiera mover los ejes, con solo mover el ratón con el botón izquierdo presionado como si se estuviera arrastrando el motor

correspondiente comenzará a girar. El procedimiento **mover** recibe dos parámetros, uno para recibir el dato que generará el flanco de subida y otro para el flanco de bajada, dependiendo de que motor se quiera mover.

El último icono de la barra de herramientas de la MDI es el que abre el formulario, que se muestra en la figura 3.8, para generar la matriz de coordenadas, esta se genera por medio del teclado en forma de tabla.

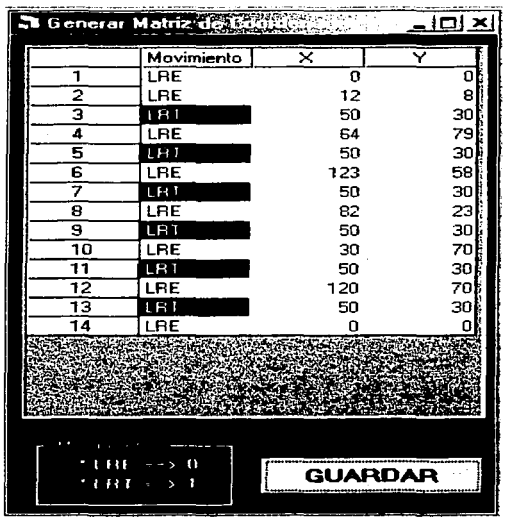


Figura 3.8 Formulario para generar la matriz de coordenadas desde el teclado

Este formulario utiliza un *FlexGrid* como apoyo para crear la matriz de coordenadas. Este objeto se maneja en forma de tabla , en la columna de movimiento solo acepta un **0** ó un **1** para representar los dos tipos de **movimiento** que se pueden realizar, **LRE** para movimiento de los ejes X-Y en línea recta y

LRT para movimiento de los ejes X-Y del taladro en línea recta; en las columnas **Y** y **X** solo acepta números reales positivos y estas representan las coordenadas. No se incluye el código debido a que solo se realiza la captura de los datos y se mandan guardar a un archivo que estos procesos son muy parecidos a los ya explicados anteriormente.

El proceso de generación de coordenadas también se puede llevar a cabo de forma grafica, que seria una interfaz más amigable para el usuario pues con solo arrastrar el ratón a una posición en la pantalla puede ir generándolas coordenadas, que después serán almacenadas en un archivo para que se pueda leer y ejecutar.

Un programa extra que genera la matriz de coordenadas programado en ***Forte for java*** es el que se muestra en la figura 3.9, este presenta una interfaz gráfica que permite agregar y eliminar perforaciones por medio del ratón.

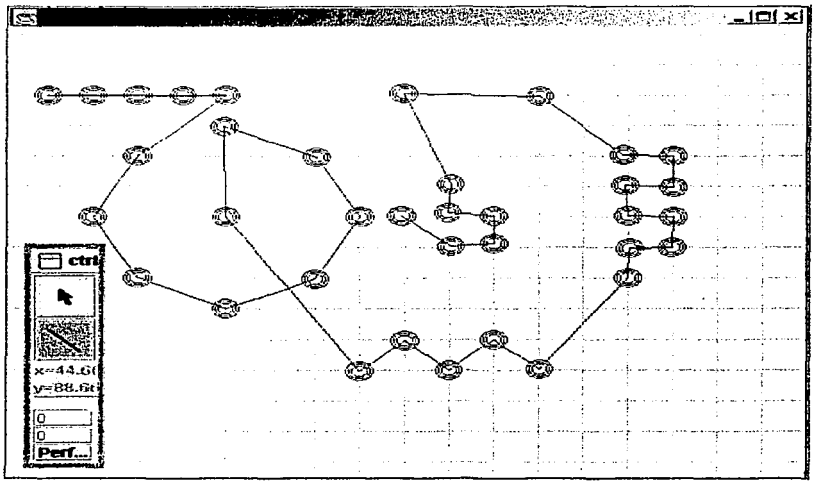


Figura 3.9 Formulario para generar la matriz de coordenadas desde un ambiente grafico

El programa muestra una cuadricula como apoyo para posicionar las perforaciones además dibuja círculos y unas líneas rectas simulando las perforaciones y la trayectoria que seguirán los ejes para realizar estas. Cuenta con una pequeña barra de herramientas, con un botón de selección y otro de edición además de mostrar las coordenadas actuales del ratón, en el modo de edición aparecen dos *Textbox* para posicionar una perforación desde el teclado y en el modo de selección aparecen las opciones de *nuevo* y *salvar*. Este programa es ***fácil*** y ***rápido*** de usar, pero no se incluye su código debido a que este es un poco extenso y en su mayor parte esta dedicado al control de gráficos.

3.2 USO DE LA ETAPA DE POTENCIA

En esta sección se describe el funcionamiento de los **drivers** y los **controladores** de los motores, que forman la etapa de potencia, y que en la mayoría de los casos estos van unidos en una misma placa.

3.2.1 FUNCIONAMIENTO DEL DRIVER

Debido a que los niveles de potencia TTL del puerto paralelo de la computadora no son los adecuados para manejar los motores de pasos, ya que existen motores que requieren como mínimo un potencial de 5V y el puerto maneja niveles máximos de 5V, aunque parecería que si manejaría un motor de 5V, pero no es posible debido a que la corriente requerida por el motor no sería la correcta. Para adaptar estos niveles de voltaje y corriente que el motor necesita, se define una primera etapa en la cual se debe proteger el puerto paralelo de la PC por medio de optoacopladores, que se encargan de aislar la salida y convertirla en una señal idéntica pero en un circuito separado totalmente, ya que solo se comunican las señales por medio de luz, todo esto para que en caso de un corto circuito o una descarga en el controlador/driver que pueda quemar sus componentes no llegue a afectar al puerto. La segunda etapa consiste en buffer de corriente, que permite manejar las bobinas del motor, aquí es donde entran en juego los transistores (que se recomienda un arreglo Darlington) con los cuales se puede manejar hasta 360W. Se debe colocar un optoacoplador por cada bit a transmitir y un transistor por cada bobina del motor. La figura 3.10 muestra estas etapas en un **driver** *para motores de pasos unipolares* y sugiere el uso de algunos elementos.

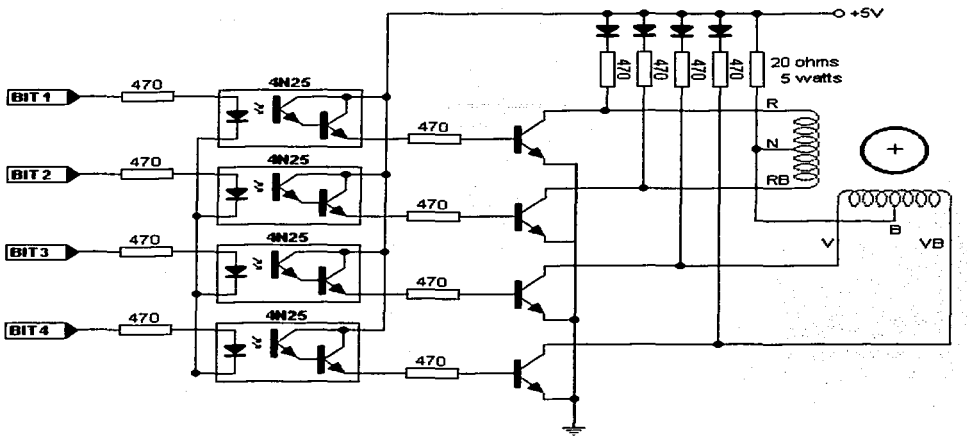


Figura 3.10 Driver para motores de pasos unipolares

En la figura 3.10 se puede apreciar que se utilizan optoacopladores 4N25 (ó también se pueden usar el 4N35, MOC3010, MOC3011 entre otros), resistencias de 470 Ohms y transistores 2N3053.

Para entender como funcionan los optoacopladores se analizará su composición. Un optoacoplador combina un dispositivo semiconductor formado por un fotoemisor, un fotoreceptor y entre ambos hay un espacio por donde se transmite la luz. Todos estos elementos se encuentran dentro de un encapsulado que por lo general es del tipo miniDIP de 6 terminales, obsérvese la figura 3.11.

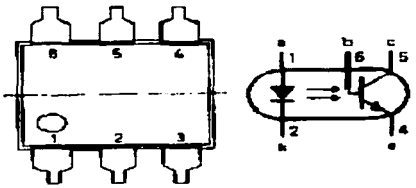


Figura 3.11 Optoacoplador de encapsulado miniDIP de 6 terminales

Se tiene una señal de entrada que se le aplica al diodo emisor de rayos infrarojos (IRED) y la salida es tomada por el fotorreceptor (tiristores o transistores), de manera que se convierte una señal eléctrica en una señal luminosa modulada y se vuelve a convertir en una señal eléctrica. Cuando se aplica una tensión sobre los terminales del diodo IRED, este emite un haz de rayos infrarrojo que se transmite a través de una pequeña guía-ondas de plástico o cristal hacia el fotorreceptor. La energía luminosa que incide sobre el fotorreceptor hace que este genere una tensión eléctrica a su salida. Este responde a las señales de entrada, que serán los pulsos de tensión que se enviarían al controlador. Existen 3 tipos de optoacopladores:

- ***Fototransistor***: se compone de un optoacoplador con una etapa de salida formada por un transistor BJT.

- ***Fototriac***: se compone de un optoacoplador con una etapa de salida formada por un triac.

- ***Fototriac de paso por cero***: optoacoplador en cuya etapa de salida se encuentra un triac de cruce por cero. El circuito interno de cruce por cero conmuta al triac sólo en los cruce por cero de la corriente alterna.

A continuación se presentan la tabla 3.2 con algunos ejemplos de los optoacopladores que son usados para la construcción de los drivers.

Chip	Fabricante:	Descripción:
MOC3010	PHILIPS, TEXAS INSTRUMENT	Optoacoplador dc diodo a triac
MOC3011	MOTOROLA	Optoacoplador dc diodo a triac
MOC3020	-------	Optoacoplador dc diodo a transistor.
MOC3021, MOC3061 y MOC3031	-------	Octoacoplador dc diodo a triac
MOC3040, MOC3041, MOC3063 y MOC5010	-------	Optoacoplador
MCT2	--------	Optoacoplador c/sal a transistor
CNY17-2	---------	Optoacop. salida a transistor.
4N25, 4N26, 4N27, 4N28, 4N29 4N35 y 4N38	---------	Optoacopl.2500V

Tabla 3.2 Otoacopladores utilizados para la construcción drivers

Siguiendo con el análisis del diver de la figura 3.10 los transistores ***Darlington*** funcionan como un interruptor de la corriente con la que se alimentarán las bobinas del motor. Un transistor Darlington es una conexión de dos transistores de unión bipolar de manera que puedan operar como un transistor con superbeta, obsérvese la figura 3.12.

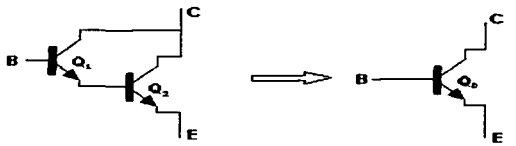


Figura 3.12 Esquema de un transistor Darlington

Una característica importante es que el transistor compuesto actúa como una sola unidad, con una ganancia de corriente, que es el producto de las ganancias de los transistores individuales. La polarización de un circuito Darlington se lleva a cabo de la forma en que se muestra en la figura 3.13.

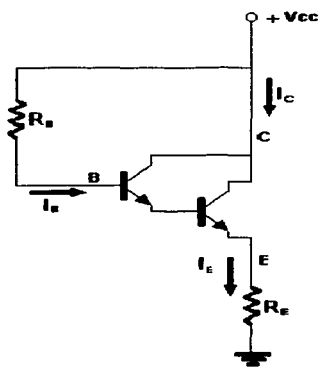


Figura 3.13 Polarización del circuito Darlington

La corriente se puede calcular a partir de la siguiente formula a), donde β_D es la ganancia de corriente que posee el transistor, y sus voltajes formula b).

$$I_B = \frac{V_{CC} - V_{BE}}{R_B + \beta_D R_E} \qquad \textbf{a)}$$

$$V_E = I_E R_E \quad y \quad V_B = V_E + V_{BE} \qquad \textbf{b)}$$

Por ultimo se conecta una resistencia y un diodo a cada terminal, este ultimo para evitar las FEM (Fuerzas Electro-Motrices), o bien, corrientes inversas producidas por el motor debido a su movimiento y sus cambios de sentido. Como se pudo observar es muy fácil construir un driver para los motores de pasos unipolares, a continuación se analizará un *driver para motores de pasos bipolares*. Las unidades bipolares requieren de un circuito driver más complejo debido a que están diseñados con bobinas separadas que necesitan ser manejadas en una y otra dirección (la polaridad debe ser invertida durante su operación) para que tenga un correcto funcionamiento. También utilizan un patrón binario, que como los unipolares se utiliza como 1 y 0, para estos se utiliza (+) y (-).

Un circuito conocido como ***puente-H*** mostrado en la figura 3.14 es utilizado para construir el driver.

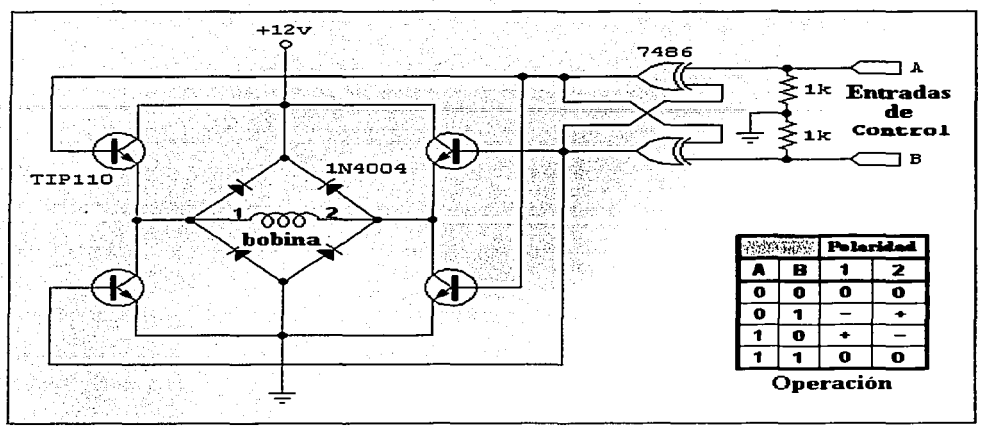


Figura 3.14[1] Conexión del circuito Puente-H

Para efecto de mayor rapidez y evitar errores costosos a la hora de construir un driver se recomienda comprar los chips con los circuitos necesarios como el arreglo de transistores Darlington y el puente-H. Para el arreglo de transistores existen muchos chips que lo manejan, unos muy comunes en el control de motores son el **ULN2003** y el **SAA1027**, el primero de estos se muestra su composición en la figura 3.15.

[1] http://www.eio.com/jasstep.htm

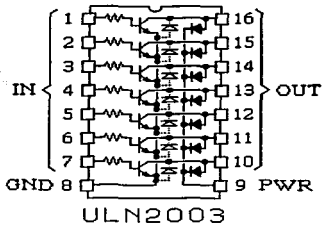


Figura 3.15[2] Esquema del circuito integrado ULN2003

Es un DIP de 16 patillas que cuenta con 7 entradas compatibles con niveles TTL cada una conectada por medio de una resistencia a un transistor Darlington y por lo tanto 7 salidas que manejaran la potencia aplicada a la patilla 9, cada transistor esta protegido por dos diodos, uno que corta al emisor del colector, protegiéndolo de los voltajes invertidos hacia el transistor, y uno conectado del colector a la patilla 9, este diodo protegerá al transistor de los picos inductivos. Usado para aplicaciones donde cada bobina del motor se maneja por debajo de 500 miliamperes.

Otros chips muy recomendables son los de la familia de Alegro MicroSystems como el **UDN2540B** mostrado en la figura 3.16.

[2] http://www.cs.uiowa.edu/%7Ejones/step/circuits.html

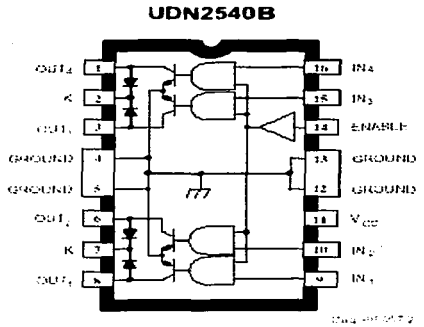


Figura 3.16[3] Esquema del circuito integrado UDN2540B

Este chip al igual que el **A2540SLB** combinan compuertas AND lógicas y proveen de una interfaz entre los circuitos que procesan señales de bajo nivel y cargas de hasta 360 W. Maneja cuatro transistores Darlington con un diodo interno que corta las cargas inductivas, manejando un voltaje de hasta 50 V. Las entradas lógicas son compatibles con TTL y sistemas lógicos de 5V CMOS; y esta compuesto de 16 patillas en un DIP.

3.2.2 FUNCIONAMIENTO DEL CONTROLADOR

Para llevar a cabo la lógica de control que se explico en el capitulo I, se podría manejar desde la computadora, pero no es muy recomendable, es por eso

[3] http://www.allegromicro.com/sf/2540/

que en esta sección se analizará la forma de construir un controlador y algunos de los diferentes chips que existen para el control de motores de pasos.

Un controlador básico para motores de pasos puede ser el que se muestra en la figura 3.16, las salidas de éste circuito pueden ser conectadas a las entradas del circuito de la figura 3.10 y el motor unipolar se conecta a las cuatro salidas del circuito mostrado en la figura 3.10, para proveer la solución de un controlador/driver completo. La desventaja del circuito de la figura 3.17 es que esta limitado a mover el motor en un solo sentido; este circuito podría ser más útil en aplicaciones donde el motor no necesite cambiar de dirección.

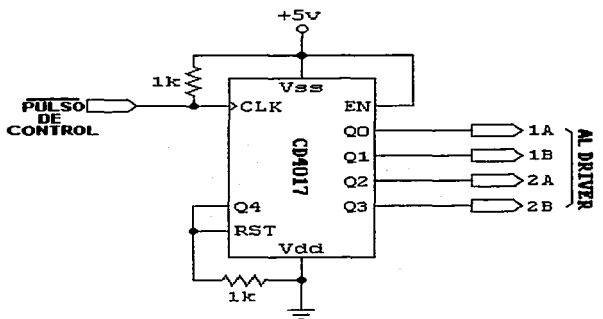


Figura 3.17[4] Controlador para motores de pasos, para girarlo en un solo sentido

A continuación se presenta un circuito que resuelve el problema del cambio de sentido del giro del motor, llamado controlador bidireccional de dos fases, véase la figura 3.18.

[4] http://www.cio.com/intro

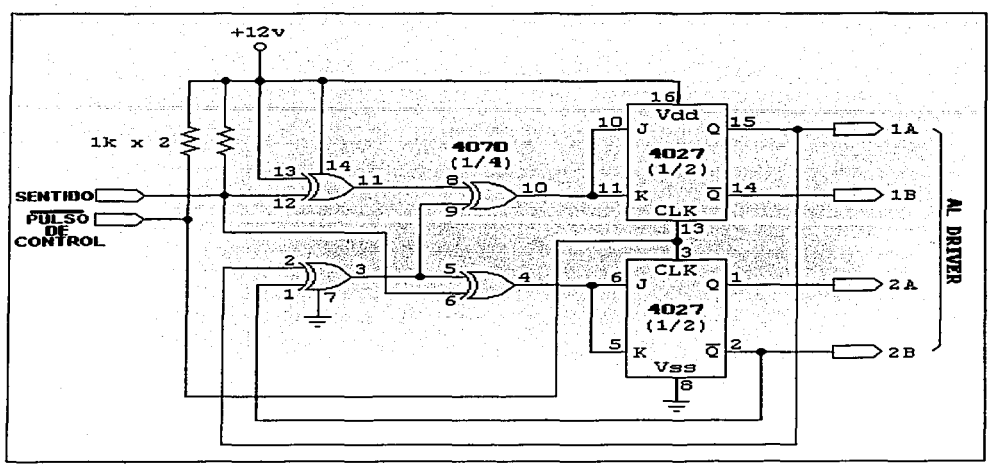


Figura 3.18 Controlador bidireccional de dos fases para motores de pasos

Existen varios circuitos controladores de motores de pasos los cuales usan lógica discreta y que funcionan sin ningún problema, a continuación se presenta otro de estos en la figura 3.19.

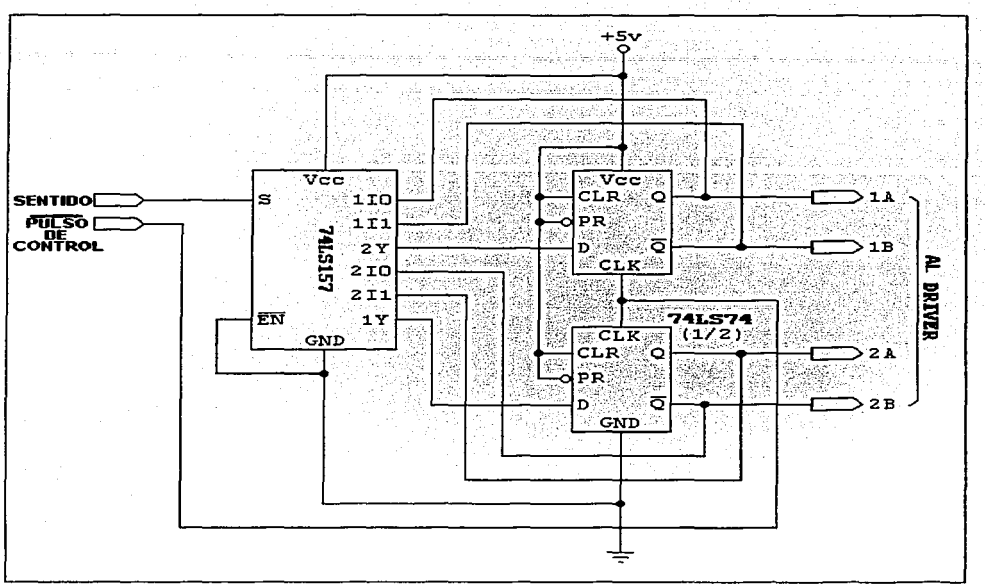


Figura 3.19[5] Controlador bidireccional para motores de pasos que utiliza lógica discreta

Aunque es muy fácil construir un circuito controlador/driver es conveniente comprar un chip que integre este en un solo encapsulado, porque permite ahorrar espacio y tiempo en el diseño de la aplicación, además de que elimina algunos posibles errores de conexión que podrían hacer inconsistente el circuito y que no funcione de manera adecuada; por este motivo se analizarán algunos de los chips.

[5] http://www.eio.com/intro

El primer chip que se analizará es el **UCN4202A** ó **UCN4203A**, que es un controlador/driver diseñado para motores de pasos de imán permanente con rangos de corriente de 500 mA, compatible con circuitería TTL, manejado por un pulso de control esperado con un tiempo mínimo de 1 μs, activado con flanco de subida y que permite cambio de sentido determinando las secuencias de salida (A-B-C-D ó A-D-C-B), obsérvese la figura 3.20 que muestra la conexión de este chip y el motor de pasos.

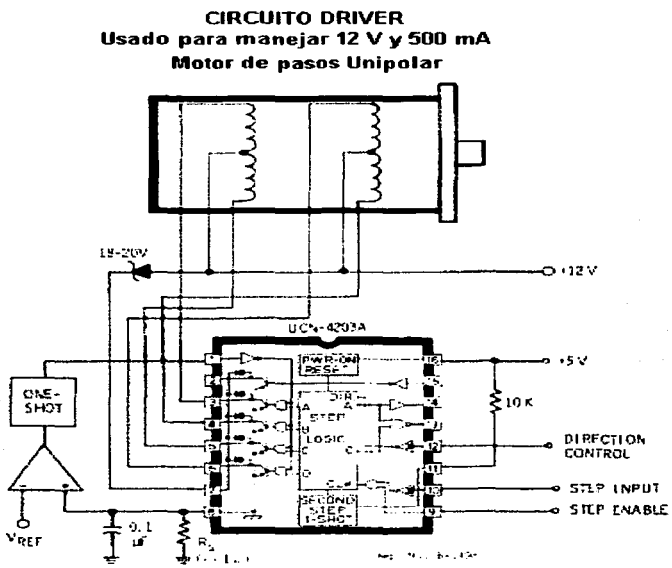


Figura 3.20 Conexión del circuito integrado UCN4203A y el motor de pasos unipolar

Un chip parecido al anterior, muy confiable y fácil de implementar es el **UCN5804B** y **UCN5804LB**, combina lógica de bajo poder CMOS con salidas de alta corriente y alto voltaje. Es un controlador/driver que provee de un control completo y un manejo de motores unipolares de 4 fases y una salida continua de corriente con rangos de 1.25 A por fase y 35 V. La lógica CMOS provee de las secuencias de control con pulso de control y uno de sentido, una función "reset", con tres formatos de paso: ***wave-drive*** (una fase), ***two-phase*** (dos fases) y ***Half-step*** (medios pasos) que son seleccionables externamente y sus entradas son compatibles con CMOS, PMOS, NMOS, TTL O LSTTL que pueden requerir el uso de una resistencia adecuada para una entrada lógica apropiada, la figura 3.21 muestra la conexión de este chip con un motor de pasos de 28 V.

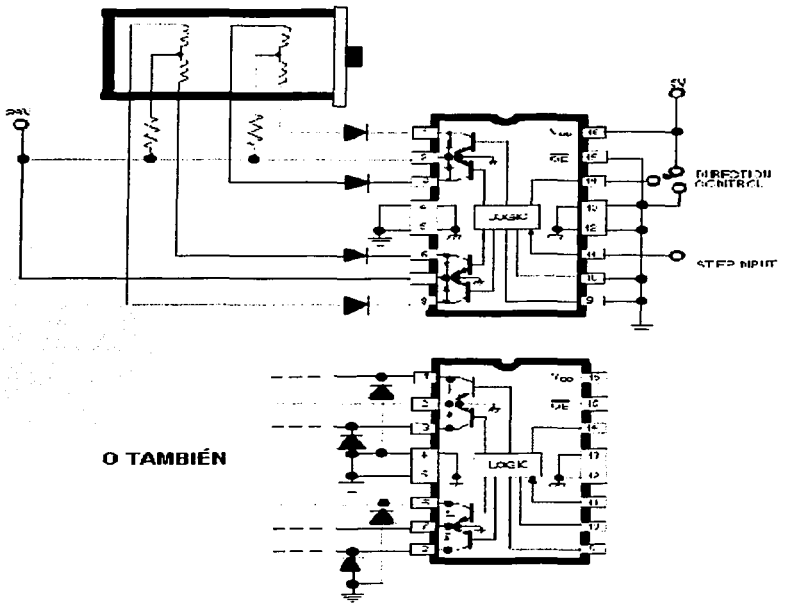


Figura 3.21[6] Conexión del circuito integrado UCN5804B con un motor de pasos unipolar

El formato **wave-drive** consiste en energizar una sola fase del motor a la vez en una secuencia A-B-C-D ó A-D-C-B, éste tiene un consumo mínimo pero genera un torque muy bajo del motor. El formato **Two-phase** energiza dos fases adyacentes en cada posición (AB-BC-CD-DA), éste ofrece un torque mejorado. El formato **Half-step** alterna entre el modo de una fase y el de dos fases (A-AB-B-

[6] http://www.allegromicro.com/sf/5804/

BC-C-CD-D-DA), que provee de una secuencia de ocho pasos. La figura 3.22 muestra las tablas con las secuencias lógicas.

SECUENCIA WAVE-DRIVE

Half Step = L, One Phase = H				
Step	A	B	C	D
POR	ON	OFF	OFF	OFF
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	OFF	OFF	ON	OFF
4	OFF	OFF	OFF	ON

SECUENCIA TWO-PHASE

Half Step = L, One Phase = L				
Step	A	B	C	D
POR	ON	OFF	OFF	ON
1	ON	OFF	OFF	ON
2	ON	ON	OFF	OFF
3	OFF	ON	ON	OFF
4	OFF	OFF	ON	ON

SECUENCIA HALF-STEP

Half Step = H, One Phase = L				
Step	A	B	C	D
POR	ON	OFF	OFF	OFF
1	ON	OFF	OFF	OFF
2	ON	ON	OFF	OFF
3	OFF	ON	OFF	OFF
4	OFF	ON	ON	OFF
5	OFF	OFF	ON	OFF
6	OFF	OFF	ON	ON
7	OFF	OFF	OFF	ON
8	ON	OFF	OFF	ON

Figura 3.22 Tablas de las secuencias lógicas para el integrado UCN5804B

Este tipo de chips es el más recomendable, pero existen otros chip que también son útiles en el control de motores de pasos, la desventaja de estos es que se requiere de 2 chips por motor y la conexión es más compleja. Por ejemplo el **A3952SB**, **A3952SLB** ó **A3952SW** es un chip controlador driver para una sola bobina del motor, la figura 3.23 muestra el diagrama de terminales de éste chip en sus presentaciones de DIP, SOP y SIP respectivamente.

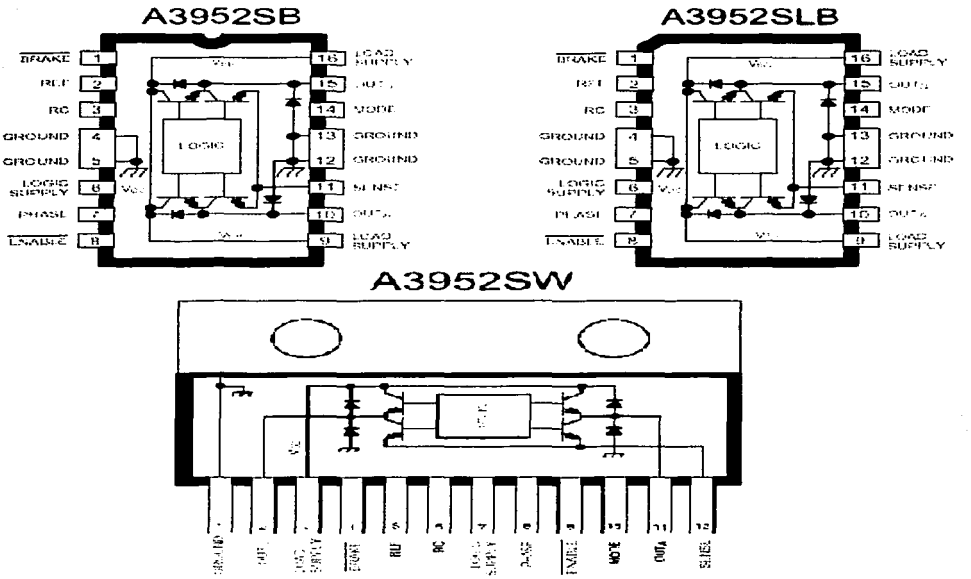


Figura 3.23[7] Esquemas del circuito integrado A3952SB, A3952SLB y A3952SW

Utiliza un control de corriente PWM (Modulación de Ancho de Pulso), permite el control bidireccional, maneja voltajes de hasta 50 V, presente una entrada de "Enable" activada en bajo, la entrada "Phase" controla la polaridad de la carga de corriente y cuando la entrada "Brake" esta en bajo la función de frenado se activa. La figura 3.24 muestra el diagrama de conexión con dos de estos chips al motor.

[7] http://www.allegromicro.com/sf/3952/

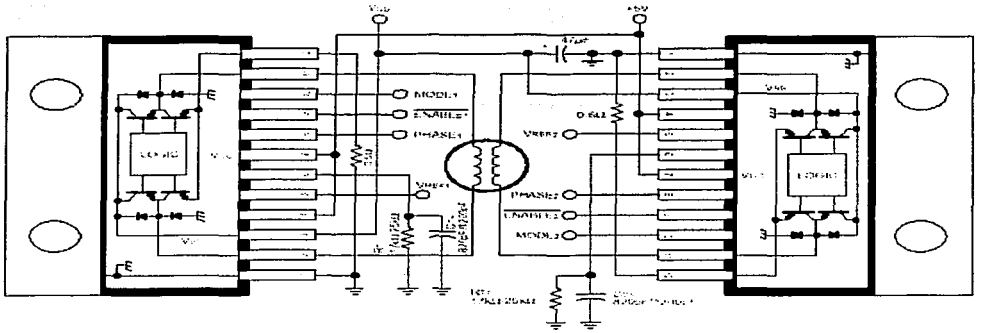


Figura 3.24[8] Diagrama de conexión del circuito integrado A3952SW con un motor de pasos bipolar

La figura 3.25 muestra un circuito controlador driver para 4 motores. Este controlador/driver esta conectado directamente al puerto parallelo, pero esta dividido en dos circuitos por medio de los optoacopladores 4N25, para que en caso de un corto o una descarga no se dañe el puerto de la computadora, las salidas de los optoacopladores van directamente a los circuitos integrados UCN5804B (contolador/dirver para motor unipolar) que darán las secuencias para girar cada motor, en el circuito el las salidas del driver tendrán 12 V.

[8] http://www.allegromicro.com/datafile/3952.pdf

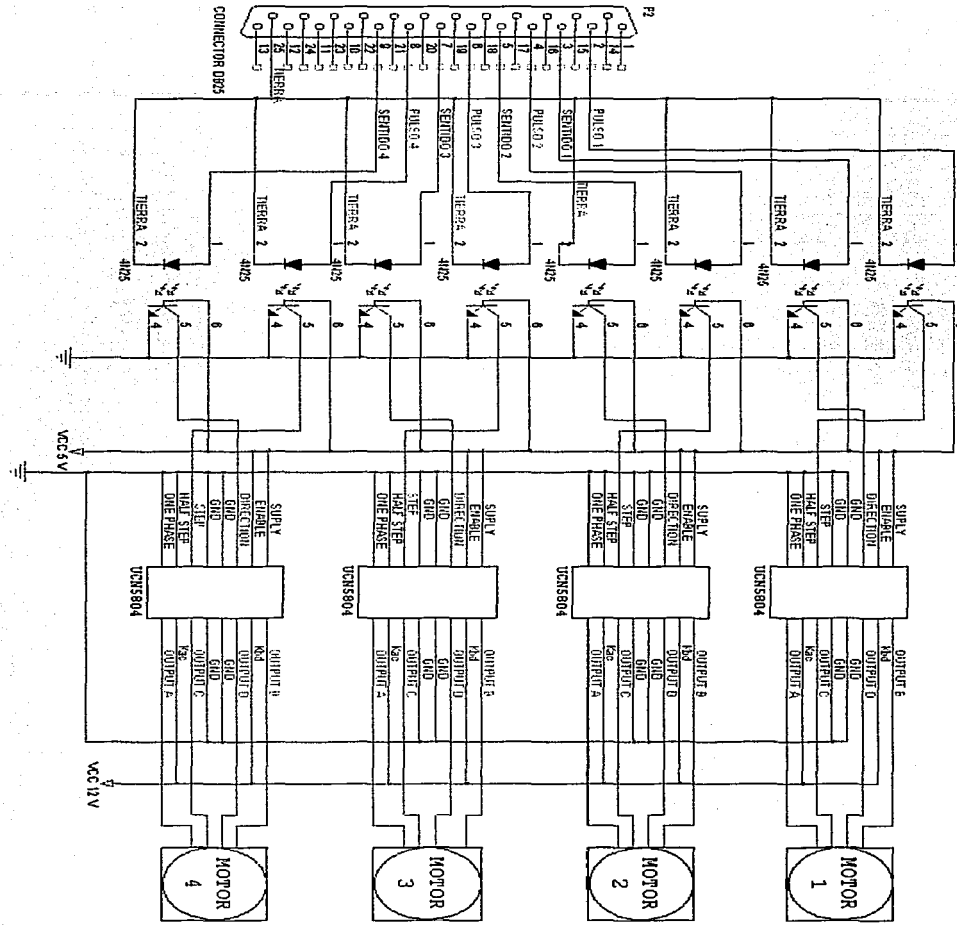


Figura 3.25 Circuito controlador/driver optoacoplado para 4 motores

3.2.3 MULTIPLEXACIÓN

Para manejar mas de 4 motores es necesario hacer un arreglo de multiplexación, la forma de manejar este arreglo se explico en la sección 2.4; ahora se analizará la composición electrónica de este arreglo. Se tienen 2 bits de miltiplexación y 6 de datos, por lo tanto, los motores se manejarán de 3 en 3. La figura 3.26 muestra el circuito de multiplexación que tomaremos como base.

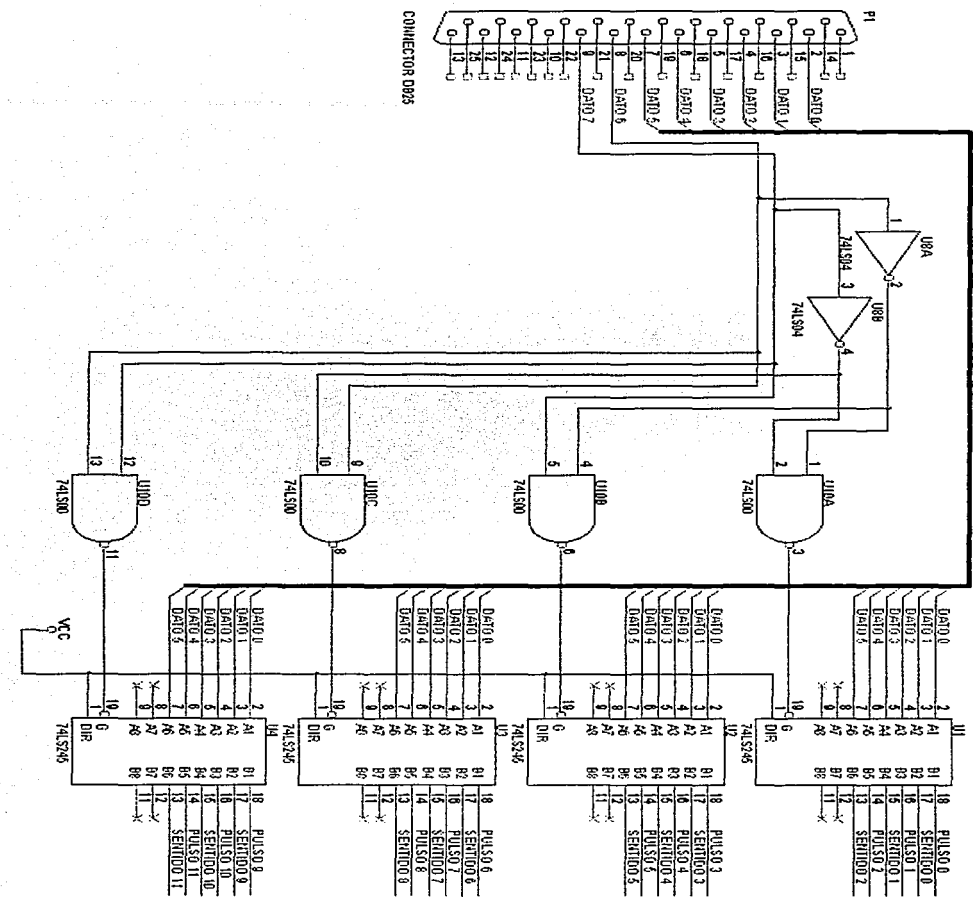


Figura 3.26 Circuito de multiplexación

Los 6 bits de datos se conecta a un bus, el cual llevará las estos a cada uno de los "tranceptores" (74LS245), estos serán los que realicen la multiplexación según las señales que se les suministren. Los bits de multiplexación primero son conectados a un arreglo de NOT (74LS04) y AND (74LS00) de manera que se tengan unas combinaciones como las de la tabla 3.3.

Bit 6	Bit 7	Tranceptor 1	Tranceptor 2	Tranceptor 3	Tranceptor 4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Tabla 3.3 tabla de multiplexación para los 4 tranceptores

El tranceptor 1 esta conectado a la salida de un AND, que a su vez se conecta a la salida de dos NOT para negar las señales. Cuando se tiene un **0** en el *bit 6* y un **0** en el *bit 7*, las entradas del AND serán **1** y **1**, por lo tanto su salida será un **1**. Para el tranceptor 2 esta conectado a un AND, una de las entradas de éste es el *bit 6 negado* y la otra será el *bit 7 directo*. De manera que si se tiene un **0** en el *bit 6* y un **1** en el *bit 7*, las entradas del AND serán **1** y **1**, así su salida será un **1**. Para el tranceptor 3 las entradas al AND serán el *bit 6 directo* y el *bit 7 negado*. Y para el tranceptor 4 las entradas del AND serán el *bit 6* y *bit 7 negados*. Con éste arreglo sólo se activara un tranceptor a la vez.

Al activarse el tranceptor las salidas tendrán el valor que se les alimenten a sus entradas, que en este caso son los 6 bits de datos utilizados por pares, el pulso y sentido de cada motor, que vienen desde el bus. Las salidas de los tranceptores se

conectarán al controlador/driver, que para este caso este último manejara 12 motores, es decir, tendrá integrados 12 controladores con su respectivo driver.

3.3 CONEXIÓN DEL MOTOR

Es muy importante saber como se deben conectar los motores de acuerdo al tipo de motores que son, es por eso que en esta sección se detalla como reconocer el tipo de motor y la forma en que debe conectarse.

Los motores de pasos tienen de 4 a 8 cables, el número de cables y la forma en que se conectan para utilizar el motor dependerá del tipo de motor de pasos que se trate (bipolar, unipolar, híbrido). La tabla 3.4 nos muestra el número de cables según el tipo de motor.

Tipo de Motor	Número de cables o terminales del motor
BIPOLAR	4
UNIPOLAR	5, 6
HÍBRIDO	6, 8

Tabla 3.4 Número de cables para cada tipo de motor

Un motor *bipolar* tiene 4 terminales, y tiene dos bobinas conectadas cada una a un par de terminales, para determinar estos pares se utilizará el multimetro para medir la resistencia, de manera que la combinación de dos cables que marquen una resistencia en el multimetro se aislarán como el par para una bobina y los cables restantes serán para la otra bobina, como se muestra en la figura 3.27.

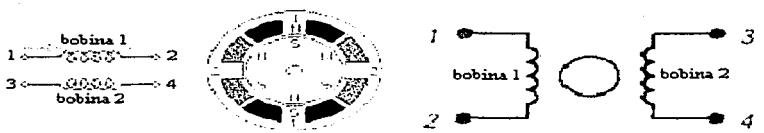


Figura 3.27[9] Conexión de un motor bipolar de 4 terminales

Para averiguar la distribución de los cables a los bobinados y el cable común en un motor de paso *unipolar* de 5 o 6 cables se deben seguir los siguientes pasos:

1. **Para reconocer el cable (ó cables) común**, se debe probar con el multimetro las combinaciones posibles entre las terminales del motor, de esta manera la terminal común será la única que tenga la mitad del valor de la resistencia entre ella y el resto de terminales. En caso de 6 cables, habrá dos comunes que se combinarán para formar una resistencia con otro par de terminales, este par de terminales se combinan para formar una resistencia que será el doble del valor de la combinación entre el común y una de estas. Cuando ya se a reconocido la terminal común esta se aislará de las demás y se conectara a la fuente de alimentación. La figura 3.28 muestra el diagrama de conexión de un motor unipolar de 4 cables.

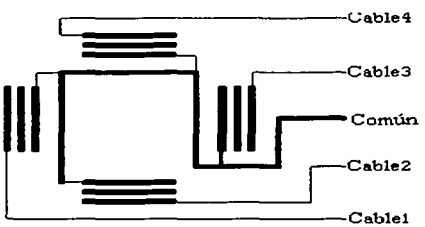


Figura 3.28[10] Conexión de un motor unipolar de 4 terminales

[9] •http://www.doc.ic.ac.uk/~ih/doc/stepper/

2. **Para identificar el orden de los cables de las bobinas**, se debe aplicar voltaje al cable común y manteniendo uno de los cables a tierra, mientras se va poniendo a tierra cada uno de los demás cables de forma alternada y observando los resultados. Siguiendo los siguientes pasos asignado un número (arbitrario) a cada terminal:
   - Se selecciona un cable y se aterriza, en este caso el cable 4 como en la figura 3.29 a).
   - El cable 4 se mantiene en tierra, y después poner el resto de los 3 cables a tierra uno por uno.
   - Se pone otro cable a tierra, y si gira el rotor en sentido horario este será el cable 3, como en la figura 3.29 b).
   - Desconectando el cable 3 de tierra y aterrizando otro cable, de manera que si gira en sentido antihorario este será el cable 1, como en la figura 3.29 c)
   - Por ultimo se desconecta el cable 1 de tierra y se aterriza el ultimo cable, el rotor no va ha girar y este será el cable 2, como en la figura 3.29 d).

Figura 3.29[11] Esquema base para realizar las pruebas de terminales para identificar su orden

[10] http://www.doc.ic.ac.uk/~ih/doc/stepper/others/
[11] •http://www.doc.ic.ac.uk/~ih/doc/stepper/control2/animated/both.html

Para los motores híbridos de 6 y 8 terminales la prueba es similar a las anteriores, para el de 6 terminales se realiza la misma prueba que al unipolar, véase la figura 3.30 a). Pero para el de 8 terminales es un poco diferente, se debe de realizar la prueba con el multimetro a todas las terminales del motor, de manera que la combinación de dos terminales marquen una resistencia en el multimetro, y estas aislarán formando un par para cada una de las 4 bobinas, como se puede observar en la figura 3.30 b).

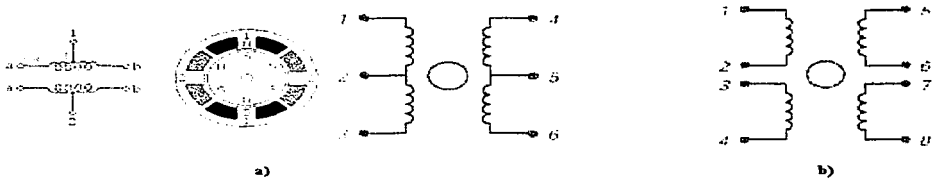


Figura 3.30[12] Conexión de un motor unipolar de 6 y 8 terminales

[12] •http://www.doc.ic.ac.uk/~ih/doc/stepper/control2/animated/both.html

3.4 DISEÑO DEL GRAFICADOR

El graficador es la parte Mecánica del sistema, éste se compone de los ejes X-Y, el taladro y su elevador. La figura 3.31 muestra un esquema general del graficador.

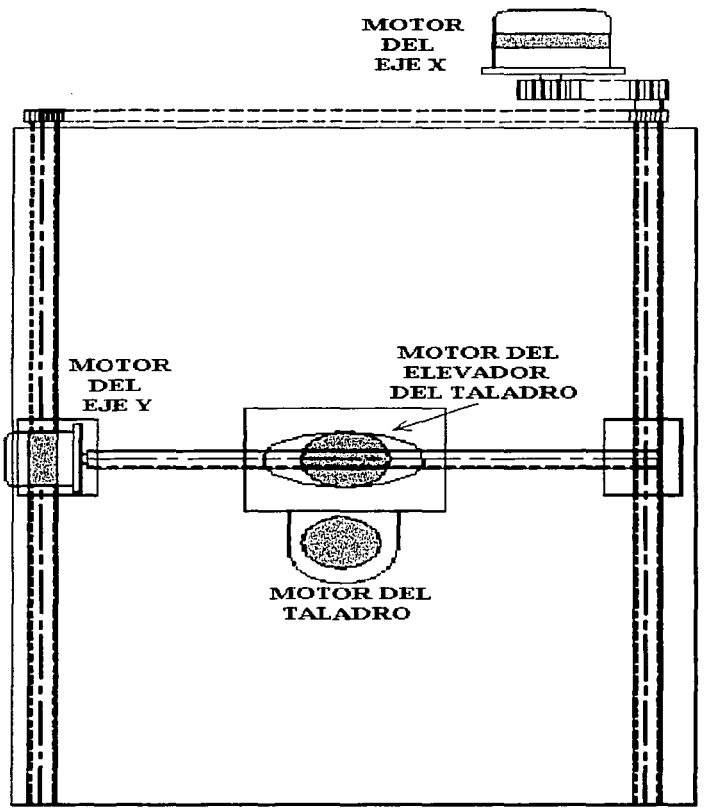


Figura 3.31 Esquema general del graficador

El eje X se compone de un motor de pasos y dos tornillos sin fin de rosca ACME, como el de la figura 3.32, cada uno de éstos utilizado para desplazar las plataformas deslizantes que soportan al eje Y.

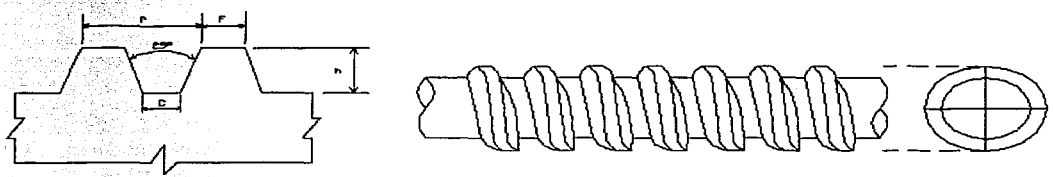


Figura 3.32 Tornillo sin fin de cuerda ACME

Los dos tornillos deben moverse sincronizados y con el mismo desplazamiento, para que esto suceda se le colocan dos poleas dentadas idénticas en cada uno y se conectan mediante una banda, también dentada como se muestra en la figura 3.33, también los tornillos deben ir apoyados en unas placas con rodamientos para evitar la fricción.

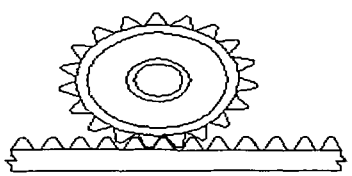


Figura 3.33 Polea y banda dentadas

Las plataformas llevan una tuerca y un placa unidas. La tuerca será roscada de acuerdo al tipo y dimensiones del tornillo, además se les debe maquinar una ranura triangular para que se deslicen sobre una barra cilíndrica lisa, esta barra

servirá como apoyo para que la plataforma no se gire con el tornillo. En la figura 3.34 se muestran cada una de estas piezas.

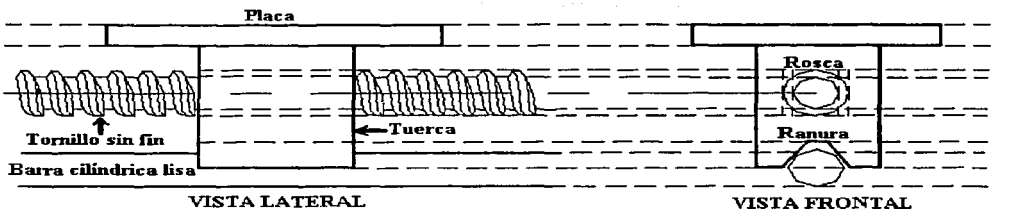


Figura 3.34 Plataforma deslizante

Para el **eje Y** se utiliza sólo un tornillo sin fin y será colocado de la misma manera que en el eje X, la diferencia es de que ahora el motor se conectará directamente con el tornillo, como se aprecia en la figura 3.35.

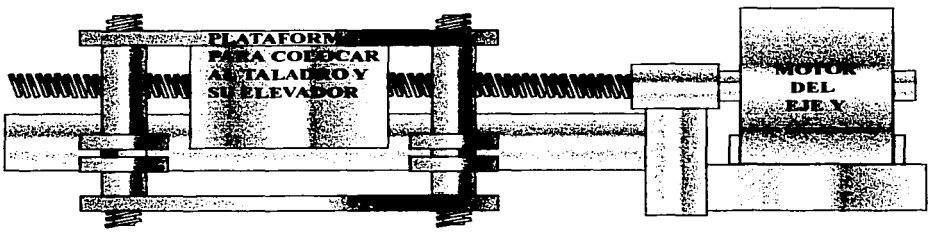


Figura 3.35 Motor tornillo y plataforma para el eje Y

El motor y los apoyos para el tornillo se colocarán en las plataformas del eje X. Este tornillo también lleva una plataforma en donde se instalará el Taladro y su elevador, ver la figura 3.36.

El elevador del taladro se debe manejar como otro eje más, que se tomará como un eje X. Este eje también debe llevar un tornillo sin fin y una plataforma, esta última para instalar el taladro, como en la figura 3.36.

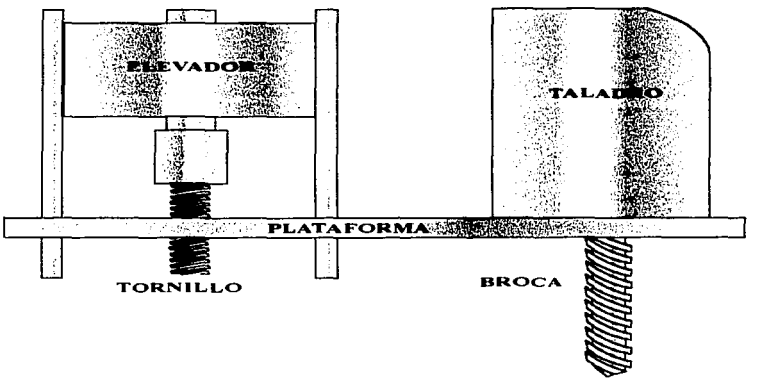


Figura 3.36 Taladro y su elevador

3.4.1 ASPECTOS IMPORTANTES PARA EL DISEÑO DEL GRAFICADOR

Es muy importante definir el diámetro de las poleas dentadas y el número de hilos por pulgada del tornillo, pues de esto en combinación con los grados por paso del motor dependerá el avance de las plataformas. Si el tornillo tiene 6 hilos (ó revoluciones) por pulgada y el motor es de 30° por paso, o 12 pasos por revolución, el eje se moverá con una precisión de 0.0138 pulgadas por paso, para determinar la precisión se utiliza la siguiente formula:

$$\text{Precisión (pulgadas/paso)} = \frac{1}{\text{Pasos/revolución} * \text{Revoluciones/pulgada}}$$

Para definir el área de trabajo se deben cortar los ejes X-Y un poco más grandes, de manera que se deje un espacio en los extremos para que las plataformas nunca choquen con estos. Por ejemplo si se requiere de un área de trabajo de 30x30 pulgadas se le agregan de 2 a 3 pulgadas más a los ejes en cada extremo.

Es muy importante que a la hora de colocar y ajustar las piezas no queden presionadas o en posiciones que ejerzan una resistencia. También es importante limpiar y lubricar las uniones que se rozarán, como la tuerca y el tornillo, para evitar la fricción.

CAPÍTULO IV

PRUEBAS Y RESULTADOS

4.1 PRUEBAS Y RESULTADOS DEL SOFTWARE

Una vez que se ha creado el prototipo el siguiente paso es realizarle las pruebas necesarias para detectar errores o hacerle ajustes de manera que trabaje de la manera más eficiente. Desde el planteamiento del programa se tenia bien definido que era lo que se pretendía realizar, es por eso que se trabajo por bloques utilizando el ***modelo de espiral***. Este es un modelo de proceso de software evolutivo que se utiliza en la construcción de prototipos con características del clásico modelo lineal secuencial, consiste en dividir el número de actividades en tares especificas, se comienza con una tarea principal, después a partir de esta de realizan las demás tareas realizando el mismo proceso de construcción. El proceso de construcción para cada tarea consiste en la planificación, ingeniería construcción, adaptación, pruebas y evaluaciones. En esta sección se analizará solo las dos últimas etapas.

Al realizar la primer tarea que es la generación de coordenadas el programa efectuaba esta tarea como se planteó en el capítulo II en la sección 2.4.2 y la matriz de coordenadas tenia el mismo formato que la tabla 2.4.3. Al ver este resultado se observó que se repetían coordenadas de un motor, por este motivo fue necesario depurar el proceso y hacer ajustes al programa de manera que la matriz de coordenadas quedaría de dimensión **n x 3**, donde las columnas tendrían el tipo de movimiento la posición en X y la posición en Y, como se muestra en la tabla 4.1.

Movimiento	X	Y
LRE	0	0
LRE	25	15
LRT	30	10
LRE	22	54
LRT	30	10
LRE	6	23
LRE	0	0

Tabla 4.1 Matriz de coordenadas optimizada

Fue necesario asignar identificadores para cada tipo de movimiento, son dos movimientos en línea recta, uno para los ejes X-Y llamándolo **LRE** y otro para los ejes X-Y del taladro **LRT**. También fue necesario adaptar el programa de manera que la posición del elevador del taladro y el taladro se inicialicen en cero cada vez que se realice un perforación sin necesidad de indicarlo en la matriz.

La siguiente tarea fue traducir la matriz de coordenadas obtenida, para esto el programa debería realizar la lectura de las coordenadas y traducirlas en movimientos convirtiendo la distancia entre las posiciones dadas al número de pasos de cada motor. En esta prueba solo se tubo un inconveniente, el programa no tenia la flexibilidad para ajustarse a diferentes tipos de motores por lo que fue necesario adicionar un código que permitiera leer, guardar y modificar la configuración de cada motor a utilizar.

Esta tarea se adapto de manera fácil e hizo que el programa se adaptara a los motores y su comportamiento con el graficador, además se le agrego un código para probar la configuración de un motor, de forma que la configuración permita que se realice el trabajo de la manera más adecuada.

La siguiente tarea fue generar las secuencias, aquí solo se agrego un pequeño código que define cada una de las secuencias que se enviaran al puerto paralelo, este fue un código muy sencillo y no se realizaron ajustes relevantes.

Por ultimo antes de enviar las secuencias al puerto se deben asignar los tiempos entre cada una. El prototipo solo definía un tiempo especifico para todos los motores, pero había un problema porque no todos reaccionaban igual, entonces fue necesario ajustar el código de configuración para que el tiempo asignado a cada motor fuera otra característica.

4.2 PRUEBAS Y RESULTADOS DEL HARDWARE

Al tener los elementos necesarios para construir un prototipo del controlador/driver en tablillas *protoboard* se inicio el proceso de construcción. Primero se hicieron pruebas con los 8 bits del puerto paralelo conectados a unos LEDs para probar si el programa estaba realizando sus tareas de manera efectiva. El segundo paso fue probar los chips utilizados de acuerdo a la conexión propuesta en las hojas de datos, esto se realizo en una tablilla *protoboard* independiente y después se reconectaron en la tablilla principal.

Para el chip controlador/driver se diseño un circuito generador de pulsos para probar su funcionamiento antes de adaptarlo al circuito principal, y se le realizaron pruebas con un motor resultando esta prueba exitosa. Para cada elemento del controlador le hicieron pruebas y ajustes por separado y se fueron uniendo al circuito principal. Cuando ya se tiene el circuito completo es necesario hacer mediciones de corriente, voltaje y temperatura, para evitar posibles errores, también es recomendable antes de aplicarle la alimentación al circuito cerciorarse de que no exista algún corto circuito que pueda dañar los elementos de éste. Los resultados en este caso fueron exitosos y no se realizaron ajustes relevantes.

4.3 PRUEBAS Y RESULTADOS DEL GRAFICADOR

La primera prueba que se le realiza al graficador es el avance de la tuerca según el tornillo, se debe observar que no tenga demasiada fricción y que el avance sea optimo.

Después al adaptar los motores se deben alinear perfectamente con el tornillo para no forzar la flecha del motor.

Otro aspecto importante es la colocación de las poleas dentadas, estas deben de estar perfectamente centradas al eje de manera que el giro sea totalmente regulado y no haya variaciones de presión en la banda, ya sea que quede muy floja o muy apretada.

También se debe asegurar cada una de las piezas a unir, para evitar vibraciones. Es muy importante lubricar todos los puntos en los que habrá deslizamientos, como el tornillo, la tuerca y los soportes de los tornillos.

Es muy importante considerar el tipo de material que se va usar para construir el graficador, así que de preferencia las piezas que se van a mover se deben construir de un material ligero y resistente, como el aluminio o plástico como en las poleas dentadas, los soportes y otras piezas que estén fijos pueden ser de cualquier material resistente.

Para el tipo de tornillo a utilizar se recomienda el de rosca ACME, pero si no se consigue también se puede usar el de rosca cuadrada y en último caso el de rosca Americana Estándar. Para los apoyos de los tornillos se recomiendan los rodamientos que evitan la fricción.

CONCLUSIONES

Los motores de pasos son de gran ayuda en procesos que requieren movimientos controlados con gran precisión y rapidez. Estos se ajustan a las necesidades de cada trabajo, y ya que existe gran variedad se puede escoger el más adecuado. Al automatizar los procesos se evitan errores costosos, se aceleran los procesos y se reducen los costos de fabricación.

La iteración de la computadora con los motores de pasos presenta una herramienta poderosa de automatización, ya que no sólo se pueden simular los procesos antes de realizarlos, sino que también se pueden ir ajustando a las necesidades de cada caso, e implementarlo en diferentes giros como cortadoras, perforadoras, tornos, etc. Esta iteración incrementa la seguridad del personal ya se puede manejar a distancia, sin exponer al usuario a daños físicos que se pudieran producir por tener un contacto directo con la máquina a controlar, así la computadora sólo interpreta las ordenes que les da el usuario por medio del software enviando la información al hardware.

El software para el control de motores de pasos es fácil de diseñar y construir, y existen varias alternativas para crearlo como Visual Basic, C, C++, Java y QBASIC, y no presenta grandes requerimientos para su desarrollo y ejecución. Para el desarrollo no sólo se requiere de conocimientos en programación sino que también en geometría analítica, para definir los tipos de movimientos que debe realizar el motor.

El hardware se puede desarrollar de diferentes formas y se debe diseñar de acuerdo a las necesidades que se presenten, tomando en cuenta las

características de los motores y los elementos con los que se cuenten. Se debe tener conocimiento básico de circuitos eléctricos y electrónicos para el desarrollo del controlador/driver.

BIBLIOGRAFÍA

CAMBEROS LOPEZ, Alberto, ***Dibujo de Ingeniería***, 9ª edición, editorial Porrua, México, 1990.

CEBALLOS, Fco. Javier, ***Visual Basic 6 curso de programación***, editorial Alfaomega, México, 1997.

F. COUGHLIN, Robert, y Frederick F. Driscoll, ***Amplificadores operacionales y circuitos integrados lineales***, 4ª edición, editorial Prentice Hall, México, 1995.

FUENLABRADA, Samuel, ***Matemáticas III Geometría Analítica***, editorial McGraw-Hill, México, 1995.

H. HAYT, William, y Jack E Kemerly, ***Análisis de circuitos en Ingeniería***, 5ª edición, editorial McGraw-Hill, México, 1999.

JAMSA, Kris, ***Aprenda C++ paso a paso***, 2ª edición, editorial alfaomega, México, 1999.

L. BOYLESTAD, Robert, y Louis Nashelsky, ***Electrónica: Teoría de circuitos***, 6ª edición, editorial Prentice Hall, México, 1997.

LEMAY, Laura, y Charles L. Perkins, ***Aprendiendo Java en 21 días***, editorial Prentice Hall, México, 1996.

OGATA, Katsuhiko, ***Ingeniería de control moderna***, 3ª edición, editorial Prentice Hall, México 1998.

PERRY, Greg, ***Aprendiendo Visual Basic en 21 días***, editorial Prentice Hall, México, 1999.

OTRAS FUENTES

http://eca.redeya.com/tutoriales/mpp/mpp.htm

http://es.geocities.com/allcircuits/index.htm

http://motioncontrol.com/

http://steppercontrol.com/

http://virtual.umanizales.edu.co/fisica/6/

http://www.allegromicro.com/

http://www.alzanti.com/start.htm

http://www.amc.com/chipdir/index.htm
http://www.ams2000.com/stepping101.html
http://www.anaheimautomation.com/home.htm
http://www.anser.com.ar/motores_paso_a_paso.htm
http://www.astrosyn.co.uk/docs/hybridsteppermotors.pdf
http://www.cenece.com/cenecemotores.htm
http://www.cienciasmisticas.com.ar/electro/mot_pap.html
http://www.cross-automation.com/
http://www.cs.uiowa.edu/%7Ejones/step/
http://www.ctc-control.com/customer/techinfo/idxdocs.asp
http://www.doc.ic.ac.uk/%7Eih/doc/par/
http://www.doc.ic.ac.uk/%7Eih/doc/stepper/control2/vc/single.c
http://www.doc.ic.ac.uk/~ih/doc/stepper/
http://www.eadmotors.com/main.html
http://www.eio.com/stepindx.htm
http://www.euclidres.com/apps/stepper_motor/stepper.html
http://www.fairchildsemi.com/
http://www.micropic.arrakis.es/Motores%20P_P%20bipolar.htm
http://www.motorola.com/
http://www.national.com/
http://www.netmotion.com/
http://www.robotec.itgo.com/
http://www.robotics.com/
http://www.shinano.com/
http://www.ti.com/
http://www.wirz.com/stepper/
http://zeus.uam.mx/labre/motores.htm