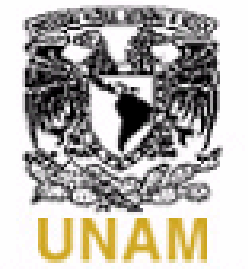


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

AUTOMATIZACIÓN DE UNA CÁMARA DE VIGILANCIA

T E S I S

Que para obtener el Título de

INGENIERO ELÉCTRICO ELECTRÓNICO

P r e s e n t a n

SERGIO ATAYDE DEL MORAL

JOSÉ ANTONIO VILA GARCÍA

DIRECTOR DE TESIS: M.I. ALBERTO FUENTES MAYA

México D.F. 2003

ÍNDICE

- **INTRODUCCIÓN**

Uno de los elementos principales para realizar actividades de vigilancia son los sistemas automatizados. Estos sistemas ofrecen la posibilidad de observar los objetos que se encuentran en el medio ambiente a través de una cámara y por medio de ésta llevar a cabo la selección de un blanco en específico.

Al escuchar el término *cámara de vigilancia*, podemos recordar algunos tipos de cámaras de vigilancia que se ven cotidianamente, en los supermercados, en los bancos, en algunas oficinas, etc. La mayoría de estas cámaras de vigilancia, son cámaras fijas que transmiten una señal de video analógico por cable o inalámbricamente, hacia un monitor donde son desplegados y/o hacia una videograbadora que las captura. También se pueden encontrar cámaras con movimientos constantes, que realizan el barrido de una zona especifica en determinado tiempo, dicho movimiento en la mayoría de los casos es en un plano determinado, esto es, solamente sobre un eje de movimiento. Podemos encontrar también, pero en menor medida cámaras con movimiento en más de un eje, controlados automáticamente por una secuencia predefinida o por medio de un operador humano que determina los movimientos de la cámara.

El título de esta tesis, lleva consigo la idea más general del sistema que se quiere desarrollar, una **cámara de vigilancia automatizada**. Sin embargo, este título es muy amplio y se hace necesario hacer algunas acotaciones y precisiones acerca de lo que se quiere llevar a cabo en el desarrollo de esta tesis.

Nuestro objetivo inmediato es integrar un sistema que opere una cámara de vigilancia con dos grados de libertad: movimiento en ronza (giro sobre un eje perpendicular a la horizontal), movimiento en elevación (giro sobre un eje paralelo a la horizontal) y con la capacidad de hacer acercamientos (zoom).

Otra de las directrices principales en el desarrollo de esta tesis es la de que el sistema implementado, eventualmente sea utilizado en proyectos de seguimiento automático de blancos. Esto es, que se busca integrar el sistema físico (cámara de video, zoom, pedestal, motores, codificadores de posición, controladores de movimiento, computadoras, tarjetas de conversión analógica-digital, manejadores de potencia) capaz de actuar en conjunto con el software especial creado ex profeso para el control automático de movimientos, y mediante el cual se puedan programar tareas y rutinas de vigilancia.

La consecución de los objetivos anteriores, involucra aplicar, con mayor o menor profundidad, varios aspectos de la ingeniería, como son la instrumentación, la programación, el control, el procesamiento de imágenes, etc.

La estructura de esta tesis es la siguiente:

En el **Capítulo I** se plantea el objetivo y aplicaciones del proyecto, así como los requerimientos de software y componentes de hardware necesarios para su desarrollo. Se propone el sistema que habrá de satisfacer los objetivos planteados, y se le subdivide en varios módulos o subsistemas para facilitar las labores de análisis e implementación.

En el **Capítulo II** se exponen generalidades acerca de los dispositivos y componentes que se utilizan comúnmente para llevar a cabo los objetivos de control y automatización planteados en el Capítulo I.

El **Capítulo III** proporciona una revisión detallada de los componentes de hardware seleccionados para implementar los módulos o subsistemas propuestos. Para cada componente se describe su funcionamiento y prestaciones, así como la forma en que interactúa con los demás dispositivos del sistema. Se puede decir que este capítulo revisa la integración física o de *hardware* del sistema

En el **Capítulo IV** se revisa la integración del sistema a nivel de *software*. Se describen los programas desarrollados para la operación del sistema, así como la interfaz visual de usuario. Hacía el final de este capítulo se presenta un algoritmo llamado *Posicionamiento Automático*, que si bien conceptualmente sencillo, permite vislumbrar las capacidades del sistema.

En el **Capítulo V**, habremos de realizar una evaluación de los resultados a los que hemos llegado, tomando como referencia los objetivos que se plantearon al inicio de la tesis. También se hablará sobre las aplicaciones que se le pueden dar al proyecto entero o a alguno de los módulos que lo componen, dando las **CONCLUSIONES** de la tesis.

Al final de la tesis se presenta el código de los programas que se escribieron, así como las traducciones de los manuales, especificaciones y notas de aplicación consultados.

CAPÍTULO I

I.1-OBJETIVO DEL PROYECTO

Desarrollar un sistema que permita a una cámara sobre un pedestal con actuadores, seguir en tiempo real una imagen: CAMARA DE VIGILANCIA. Dichos actuadores (motores) darán movilidad a la cámara sobre dos ejes, uno de los cuales permitirá un barrido de 360° (horizonte), y el otro un barrido de 120° (elevación) pudiendo así cubrir todo un hemisferio como campo de visión.[1] Las imágenes de la cámara serán desplegadas en un monitor dedicado, o en alguna ventana de un monitor de computadora [2]

Para tener una imagen adecuada, incluso cuando el objetivo cambie su distancia respecto a la cámara, ésta contará con un zoom, por lo que los algoritmos de control del sistema deberán coordinar los movimientos de los actuadotes (en velocidad y recorrido) con el valor actual de zoom.

I.2-SISTEMA DE VIGILANCIA

La descripción de un sistema de video-vigilancia en general da la pauta a la elección de componentes. El siguiente diagrama presenta el esquema de un sistema de vigilancia:

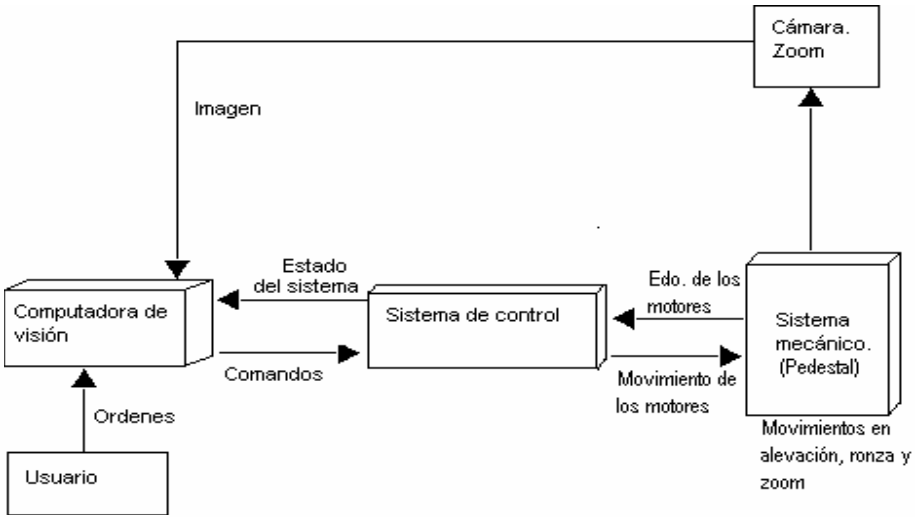


Fig.I.1.-Descripción de funcionamiento

Funcionamiento

Las imágenes capturadas por una cámara analógica/digital de video son convertidas a un formato adecuado para ser manipuladas por una computadora; ya sea para su procesamiento digital o simple despliegue. A través de una interfaz de usuario, los algoritmos programados en dicha computadora, que llamaremos *computadora de visión*,

[1] Evidentemente el "hemisferio de visión" dependerá de la orientación que se le de a la base, utilizándose los términos de horizonte y elevación sólo como los referentes más próximos.
[2] El despliegue de la imagen de la cámara dependerá del grado de autonomía del sistema y la aplicación específica.

ordenarán que los actuadores (motores) entren en acción y re-orienten la cámara para que conserve en su campo de visión el blanco deseado.

La *computadora de visión* estará dedicada exclusivamente a la manipulación digital de las imágenes, esto es, a su despliegue y/o a su análisis e interpretación. Así mismo la *computadora de visión* albergará la interfaz de usuario. Usualmente, se emplea una segunda computadora para que se encargue del control de los motores y el zoom, la cual designaremos como *computadora de control* o *computadora anfitriona*[3].

La *computadora de visión* se comunicará con la de control: le enviará las nuevas coordenadas de orientación y/o el nuevo valor del zoom y recibirá información del estado del sistema. Se va estableciendo así, una relación cliente-servidor en la que cada módulo adicionado es cliente de la etapa previa y servidor de la posterior. Este enfoque modular dota al sistema de flexibilidad: cada módulo puede desarrollarse y modificarse por separado y una vez listo se integra al sistema.

Codificadores de posición e interruptores límite adosados a los motores y al pedestal respectivamente, informarán a la *computadora de control* del estado de las partes móviles del sistema.

Siguiendo la premisa de módulos, la *computadora de control*, a su vez, hará uso de tarjetas adicionales a través de sus ranuras de expansión. De lo anterior se desprende el segundo apelativo de dicha computadora como *computadora anfitriona*, pues éste es un nombre más descriptivo de la verdadera función de esta computadora dentro del sistema.

Las señales de salida que genere el sistema de control, serán las que manejen a los motores a través de una interfaz de potencia.

I.3-MÓDULOS DEL SISTEMA

Comenzaremos la definición del sistema dividiéndolo en tres módulos o subsistemas interactuando entre sí:

• El **sistema mecánico** estará formado por una base que da soporte a la cámara de video y a los elementos móviles de la cámara de vigilancia (engranes, motores interruptores, etc.). Los motores forman parte del sistema mecánico empotrados con el juego de engranes o planetario mismo que nos dará los diferentes movimientos

•El **sistema de control**, será el que nos permita tener movimientos controlados de la cámara en velocidad, aceleración y posición. Este sistema estará formado por una computadora digital, una tarjeta controladora, manejadores de potencia, codificadores de posición, tarjetas CDA/CAD e interruptores. Analizando lo anterior como un sistema de control en general, se tiene que la computadora digital será la que se encargue de obtener los parámetros de entrada del sistema de control: posiciones deseadas. La

[3] El término *computadora anfitriona,* que es el que se utiliza a lo largo de la tesis, se explica y justifica al detallar más adelante la función de esta computadora.

tarjeta controladora se encargado de comparar los valores de *posición deseada* y *posición actual* y en base a dicha comparación generar una señal de error.

Codificadores de posición cerrarán el lazo del sistema, proporcionando como retroalimentación la posición actual de los motores. (Ver figura I.2)

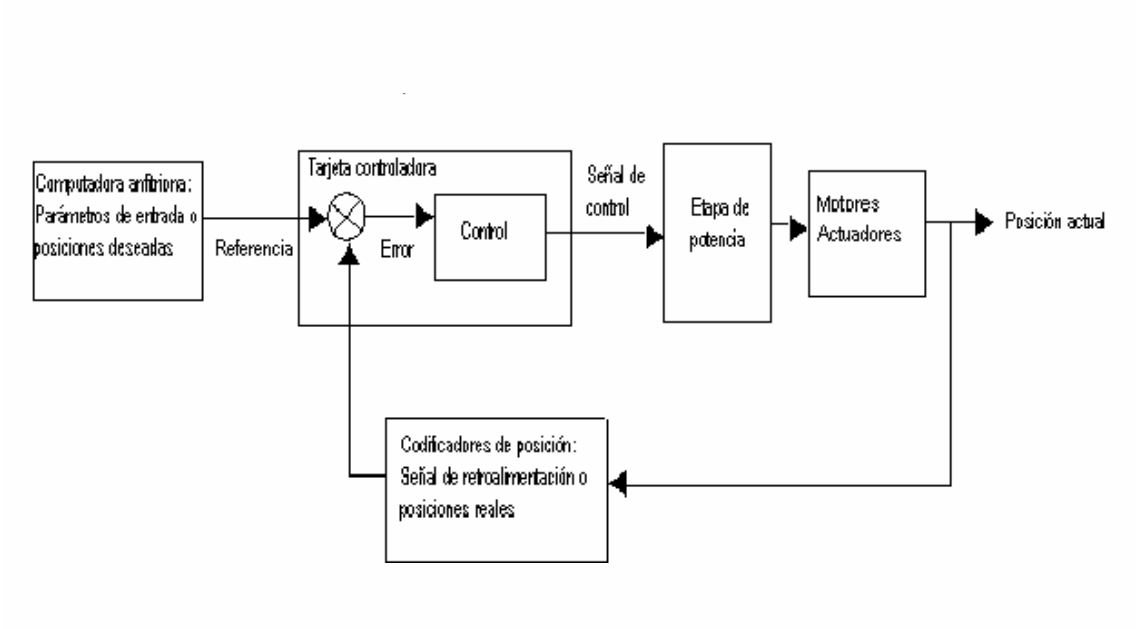


Fig.I.2.- Diagrama de bloques del sistema de control

Existe un modulo que, en función de la señal de control, se encarga de dar la potencia necesaria para los movimientos de los motores.

•El **sistema de video**, será el encargado de la obtención y manejo de la señal de video. Este sistema constará de una cámara de video analógica/digital que transforme señales luminosas en señales eléctricas; un convertidor analógico digital que transforme la señal de video analógica/digital a formato adecuado para su manejo (*Frame Grabber*); una computadora, llamada *computadora de visión,* que haga el manipulación de la señal de video ya en formato digital. Para tener una imagen detallada usualmente se cuenta con un zoom.

La división del sistema de vigilancia en los tres subsistemas recién descritos, facilitará la exposición del funcionamiento del sistema completo. Al estar íntimamente relacionados dichos subsistemas, algunos componentes se podrían ubicar en más de uno. Un claro ejemplo es el de los motores, los cuales desde el punto de vista más general son elementos mecánicos, esto es, elementos de movimiento; pero que también forman parte del sistema de control como lo que denominan algunos autores como actuadores.

Antes de continuar, es conveniente subrayar algo. En los tres subsistemas descritos, se han mencionado dos computadoras y una tarjeta controladora de movimientos. Aún cuando el objetivo inmediato de la tesis podría llevarse acabo con una sola computadora, se quiere recalcar el hecho de que este proyecto busca sentar la

base de proyectos posteriores en los que el procesamiento digital de las imágenes obtenidas será una labor bastante ardua, por lo cual, se busca asignar las tareas de manipulación de imágenes a una sola computadora desahogándola en la medida de lo posible de cualquier otra tarea.

I.4.-REQUERIMIENTOS DE HARDWARE Y SOFTWARE

A continuación se revisan las características requeridas para los componentes de los subsistemas recién definidos.

I.4.1.-Cámara de vídeo analógica.

Las etapas de operación de una cámara pueden dividirse en etapa de adquisición y etapa de formato de salida de vídeo, siendo la segunda la más importante. La etapa de adquisición queda primordialmente definida por el sensor que adquiere la imagen. Las cámaras actuales se basan en chips CCD (Charge Coupled Device) que son matrices bidimensionales de elementos fotosensibles similares a capacitores que almacenan carga eléctrica en función de la intensidad luminosa incidente. Las dimensiones de dicha matriz de sensores (columnas X filas) define la resolución de la cámara. Así, en principio, el sensor CCD puede pensarse como un arreglo de "pixeles-sensores".

El orden y frecuencia en que la matriz de sensores CCD es "barrida" y la forma en que es transducida y codificada la carga de los sensores del CCD, definen la segunda etapa de operación de la cámara, que se encarga de dar formato a la salida de vídeo, ya sea analógica compuesto o Y/C[4].

Establecido todo lo anterior, se propone una cámara con las siguientes características:

*Sensor CCD
*Capacidad de salida de vídeo en formato compatible con la entrada del siguiente dispositivo
*Opción de señales de entrada/salida para sincronización con dispositivos externos (por ejemplo con un digitalizador de imágenes -*Frame Grabber*-).

I.4.2.-Zoom motorizado

Para satisfacer el objetivo de proveer al sistema de la capacidad de realizar acercamientos de las imágenes de interés, y además para poder automatizar dicha capacidad, se requiere de un zoom con las siguientes características:

*motores integrados para modificación de foco y zoom
*potenciómetros integrados para señales de error de zoom y foco

[4] En el capítulo III se dan detalles de estos formatos de video.

*opciones de control de iris
*niveles de acercamiento adecuados para el fin de vigilancia específico

I.4.3-*Frame Grabber*

La señal generada por la cámara de video, es un flujo de información analógica compuesta por "píxeles" que forman líneas, que forman marcos, que forman secuencias completas de vídeo.

Para manipular computacionalmente la señal generada por la cámara, se debe tener acceso a cada uno de los píxeles de cada uno de los marcos. Para ello se utilizan dispositivos como el *Frame Grabber* capaces de muestrear el flujo de entrada de video, y convertirlo en información binaria cuyo almacenamiento se hace en registros de memoria cuya dirección es plenamente conocida y por tanto son directamente accesibles por los algoritmos de procesamiento digital. El *Frame Grabber* puede pensarse como un convertidor analógico digital altamente especializado.

Para los fines del proyecto que procesará imágenes en tiempo real, se requiere un Frame Grabber con las siguientes características:

*Alta velocidad de adquisición de píxeles (para operación en tiempo real)
*Entrada compatible con el formato de salida de la cámara asociada.
*Bus de conexión de amplio ancho de banda como el PCI para altas trasferencias de datos (del orden de decenas de Megabytes por segundo)
*Opciones de señales de entrada/salida para control y sincronización con la cámara "residente".
*Opciones de pre-procesamiento de la imagen adquirida.

I.4.4-Computadora de visión

La información digital a la salida del *Frame Grabber* habrá de ser procesada por esta computadora. Como es bien sabido, el procesamiento digital de imágenes en tiempo real es una labor ardua a nivel de operaciones de cálculo y disposición de memoria. Por ello la computadora de visión habrá de cumplir con especificaciones altas en :

*procesador,
* memoria de trabajo RAM,
*bus frontal,
*capacidad de almacenamiento,

así mismo requerirá de una tarjeta de video compatible con el dispositivo previo de imagen y unidades mínimas de disco-flexible y CD-ROM.

I.4.5.-Computadora anfitriona

Esta computadora no requiere tanto poder de procesamiento como la de visión, pudiendo ser una Pentium genérica con opción de conectores para tarjetas de expansión para bus ISA (como PC/104) o PCI. Está computadora está pensada para actuar como un modulo "traductor", que interpretará las peticiones de la computadora de visión, y la convertirá en comandos inteligibles para una tarjeta controladora conectada a una ranura de expansión.

I.4.6.-Tarjeta controladora de movimiento

La tarjeta controladora de movimiento, es el módulo principal en el control de los motores actuadotes para el sistema propuesto. Su función será generar las señales de control PWM o variaciones de voltaje que manejen, a través de una etapa de potencia, cada uno de los motores del sistema.

Este tipo de tarjetas están basadas en microcontroladores de diseño específico, capaces de controlar velocidad, aceleración y posición del eje de un motor. Las características requeridas para la controladora, son:

*Capacidad para controlar dos ejes (motores)
*Entradas para codificadores de posición
*Compatibilidad con conector ISA o PCI

I.4.7.-Codificadores de posición

Los codificadores de posición son elementos que se sujetan al eje de un motor y tienen la función de proveer una señal codificada al presentarse movimiento relativo entre el eje del motor y el cuerpo del codificador. Con estos pulsos, una tarjeta controladora como la descrita en el inciso anterior, es capaz de llevar acabo el control de posición, velocidad y aceleración de un motor, al establecerse un lazo de control cerrado. Las características del decodificador, deben ser:

*Compatibilidad de las señales de salida con la tarjeta controladora seleccionada
*Resolución adecuado a los parámetros de precisión requeridos
*Tamaño adecuado al motor/actuador al que se va a adosar

I.4.8.-Motores actuadores

Los motores serán los responsables de re-orientar la cámara de visión, y estarán al servicio directo de la tarjeta controladora. De los motores se requiere los siguientes parámetros:

*Tamaño pequeño
*Alto nivel de torca

*Eje capaz de albergar un codificador de posición
*Curvas de operación lineales que faciliten el control

I.4.9.-Manejadores de potencia

Como se mencionó anteriormente, la *tarjeta controladora de movimiento* es responsable de generar señales de control, pero al estar formada por elementos digitales, sólo es capaz de entregar señales de voltaje del orden de volts y corrientes del orden de micro amperes (o inferiores). Por tanto se requiere una etapa de potencia capaz de suministrar las corrientes que exige un motor como el descrito, las cuales son del orden de décimas de ampere hasta un par de amperes. Las características de un manejador de este tipo serán:

*Proveer la máxima potencia que puedan llegar a requerir los motores
*Ser compatibles con la salida de control de la tarjeta controladora

1.4.10.-Tarjetas de conversión Analógica-Digital-Analógica

Las señales de sensado de tipo analógico (por ejemplo la codificación de posición de aquellos elementos móviles que se realice por medio de potenciómetros) deben ser convertidas a un formato digital para los fines de automatización y control establecidos. Por ello se requiere un convertidor Analógico Digital con las siguientes características:

*Ofrecer conversiones Analógicas-Digitales y Digitales-Analógicas para varios canales
*Apilable en una ranura ISA o PCI.
*Velocidades de conversión adecuadas

1.4.11.-Software

El *software*, es por decirlo de alguna forma, el componente intangible que "pega" al *hardware*, que le da vida. Se requiere pues del software adecuado para desarrollar y correr los programas de control del sistema:

*Sistema operativo gráfico para la computadora de visión, pues ésta es la computadora con la que interactuará directamente el usuario
*Una herramienta de programación visual para desarrollar una interfaz de usuario gráfica.
*Sistema operativo tipo MS-DOS para la computadora anfitriona
*Un compilador para programar bajo MS-DOS los componentes de software de control no-gráficos.

I.5.-APLICACIONES

Este sistema, no se concibe como una aplicación terminal, sino como un módulo, completamente funcional, sí, pero dentro de algún otro sistema con un objetivo más grande.

La aplicación directa e inherente al sistema es la de ser un módulo vigilancia asistida por un operador, pero se puede utilizar en otros sistemas que necesiten de control de posición y velocidad Una aplicación que seguiría a este trabajo es la de seguimiento automático de un blanco, al ligar el sistema integrado con el algoritmo adecuado (evidentemente más complejos que los desarrollados en esta tesis).

El sistema puede aplicarse hacia la automatización de procesos industriales en los que una supervisión visual asegure la calidad de un producto, ya sea al comparar las imágenes capturadas con patrones almacenados previamente, o al reconocer patrones de operación fuera de norma. La ejecución de labores visuales con dispositivos automatizados puede ofrecer ventajas sobre su contraparte humana: la visión de máquina no se fatiga, ni es discrecional, ni subjetiva. La visión de máquina responderá siempre igual a una misma condición, así, la eficacia del sistema dependerá sólo de los algoritmos programados y la adecuada conjunción de componentes de hardware.

En robótica, un sistema de seguimiento automático puede sincronizarse con dispositivos tales como brazos autómatas y dirigir sus movimientos en alguna tarea específica.

I.6.- METODOLOGÍA

La integración del sistema de video-vigilancia, constó de su implementación física a nivel de *hardware*, y del desarrollo de los componentes de *software* necesarios para su puesta en marcha y control. Dicha integración, se hizo desarrollando cada uno de los tres subsistemas que se propusieron.

Hay que resaltar el hecho de que esta tesis no partió de cero en el apartado de los componentes, debido a que varios de ellos fueron componentes recuperados de algunos otros proyectos, por lo que se puede hacer una distinción entre componentes dados y elegidos. Pero ya sea que fueran componentes elegidos o dados, se hubo de investigar acerca de las características y funcionamiento de todos ellos, con el fin de lograr su integración y adecuación al sistema que se pretendía implementar.

El procedimiento de integración empleado, puede describirse de la siguiente forma:

1.-Se investigan las características de cada uno de los componentes de un mismo subsistema: se comparan las ventajas/desventajas de un componente respecto a otro que realice la misma tarea y del cual se pueda disponer; se verifica la compatibilidad de un nuevo componente propuesto con el resto de componentes ya elegidos o dados.

2.-Se integran físicamente los componentes del mismo subsistema, lo cual implica investigar con detalle: manuales, notas de aplicación y demás bibliografía auxiliar.

3,-Se desarrolla un componente de software con el que se pueda verificar la correcta integración y funcionamiento del subsistema. Igual que en el inciso anterior se consultan manuales y notas técnicas, además del software de demostración con que los fabricantes de hardware suelen acompañar sus productos (que de seguro no hará exactamente lo que se necesita, pero va a ayudar bastante como ejemplo).

4.-Corroborado el funcionamiento del subsistema, se integra al resto de los módulos ya implementados y probados, haciendo para ello las modificaciones pertinentes el software de control maestro.

5.-Se repiten los pasos 1, 2, 3 y 4 hasta que se considere alcanzado el objetivo final.

CAPÍTULO II

- ***CONCEPTOS GENERALES.***

II.1- MOTORES

En general se puede referir como maquinas eléctricas a los generadores y motores. Los generadores transforman la energía mecánica en eléctrica y los motores son máquinas en que transforman la energía eléctrica en mecánica.

Hablando específicamente de motores, se tienen dos tipos principales: los motores de corriente alterna y de corriente directa. Aún cuando parecen ser muy diferentes estos tipos de motores, el análisis de ellos se puede realizar en base a una misma máquina básica compuesta de un sistema de polos fijos, llamado estator y de una armadura giratoria, conocida como rotor. Del análisis de dicha máquina compuesta deriva directamente el estudio del motor de CD.

Las máquinas eléctricas consisten de un circuito magnético, uno o más circuitos eléctricos y soportes mecánicos, con por lo menos un embobinado. Tal embobinado se energiza desde una fuente de energía eléctrica. El circuito magnético está formado por un hierro, interrumpido por el entrehierro entre el estator y el rotor. Los núcleos magnéticos en el estator sujetos a flujos que sufren rápidas variaciones en el tiempo son usualmente laminados para asegurar bajas pérdidas por corrientes de Eddy y respuestas rápidas. Las vueltas en los embobinados de máquinas pequeñas consisten en alambre redondo. El material conductor más común es cobre, aun cuando el aluminio tiene un uso limitado.

II.1.1.- Motores de CA.

De la máquina compuesta descrita anteriormente, se pueden derivar otra clase de máquinas, que se alimenten por medio de una fuente de corriente alterna. Estas máquinas son llamadas máquinas de corriente alterna, y pueden ser monofásicas, bifásicas o trifásicas, dependiendo del tipo de fuente usada.

Las máquinas trifásicas encuentran una extensa aplicación en grandes establecimientos industriales como generadores y motores. Los motores de dos fases son muy usados como dispositivos de posición en sistemas de control automático. Los motores monofásicos encuentran gran aplicación en el hogar y en la industria como primotores.

Las máquinas de corriente alterna se clasifican también como: síncronas y asíncronas. Las máquinas síncronas derivan su nombre del hecho de que giran a una velocidad angular constante, la cual se llama velocidad angular síncrona.

Las características mas importantes para nosotros en este punto, es el hecho de tener control sobre la velocidad y posición de un motor, por lo cual los motores bifásicos son aquellos que llaman nuestra atención. Analizando estos motores, nos encontramos con que la velocidad de rotación del campo magnético rotatorio, que es la que determina la velocidad de giro del eje del motor, está dada por la siguiente ecuación:

$$N_{sin} = 2f / P$$

Donde el termino N_{sin} es llamado velocidad síncrona del campo magnético rotatorio, la letra f representa a la frecuencia de la excitación, y P son los polos del sistema.

De aquí podemos concluir que la velocidad del eje del motor depende por lo tanto, también de la frecuencia y los polos del sistema. Los polos del sistema están determinados intrínsecamente en la construcción del motor, por lo cual no se puede contar con ellos para realizar un control. La frecuencia es una variable que se puede controlar por medio de osciladores o dispositivos específicos para ello, siendo esto un poco complicado.

Se puede también observar, que los motores bifásicos se usan en sistemas de posicionamiento, y el nuestro no solo involucra posicionamiento sino control de velocidad, y como ya vimos el control de dicha velocidad no es algo sencillo.

Otro punto en contra de la utilización de motores de CA en nuestro sistema, es que de tener un motor bifásico, que es él mas utilizado en sistemas de posicionamiento, necesitaríamos una fuente bifásica de alimentación. Por lo tanto, a continuación veremos el funcionamiento del motor de CD y se verán las ventajas que este motor tiene sobre los de CA para el desarrollo del sistema propuesto en esta tesis.

II.1.2.-Motores de CD. Características y funcionamiento.

Un motor elemental compuesto de un sistema de polos fijos y de una armadura rotatoria es la base con la que se puede explicar el funcionamiento de este motor, el cual se ilustra a continuación

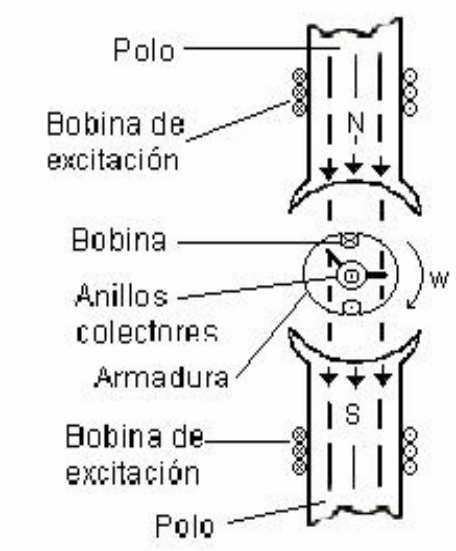


Fig. II.1- Configuración básica del motor de CD(B2).

El número de polos usados en una máquina de CD está gobernado por el valor nominal del voltaje y la corriente de la máquina. Mientras mayor sea el voltaje para un

diámetro dado de la armadura, menor será el número de polos. Esto es necesario para proveer un espacio para el mayor número de barras conmutadoras requeridas para el voltaje más alto, ya que hay usualmente tantos juegos de escobillas, como hay de polos, y el espacio entre juegos de escobillas adyacentes es el mismo que el existente entre los polos adyacentes.

El diagrama esquemático del circuito magnético de una máquina de cuatro polos, muestra las trayectorias aproximadas que toma el flujo debido a la excitación del campo.

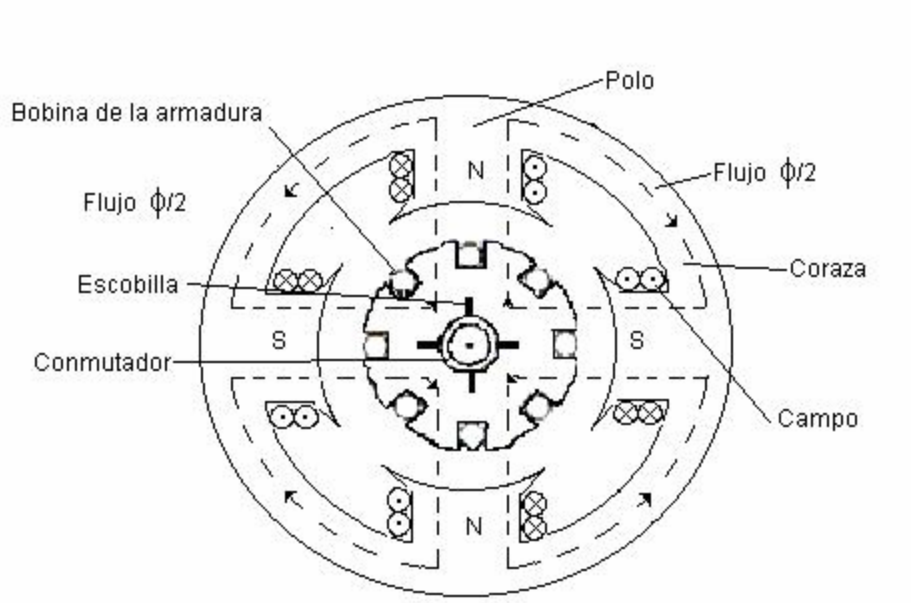


Fig. II.2.- Trayectorias del flujo dentro de una máquina de CD de 4 polos(B2).

Los polos más grandes llevan las bobinas de campo que produce el flujo principal, y los polos pequeños son los polos de conmutación o interpolos que ejercen una Fuerza Electro Motriz (FEM) en oposición de la armadura, con objeto de alcanzar una conmutación si tener prácticamente ninguna chispa en las escobillas. Los polos conmutadores son generalmente omitidos en máquinas pequeñas.

Los polos del campo y los polos conmutadores son una sección transversal rectangular construida de láminas de acero. La construcción del polo laminado minimiza las corrientes parásitas resultantes de las pulsaciones en el flujo del entrehierrro causado por la diferencia en la reluctancia de las ranuras de la armadura y los dientes de la armadura. Los polos del campo están sujetos al yunque que completa el circuito magnético

La máquina de corriente directa usual depende para su operación en la rotación de un embobinado, llamado embobinado de armadura, en un campo magnético producido por un embobinado estacionario conocido como el embobinado de campo. El embobinado de la armadura consiste de un número de bobinas usualmente idénticas colocadas en ranuras uniformemente distribuidas alrededor de la periferia del hierro del rotor, que está construido con laminaciones de acero. Las bobinas están interconectadas a través del conmutador compuesto de un número de barras, algunas veces llamadas segmentos del

conmutador, que están aisladas entre sí. El conmutador gira con la armadura, sirve para rectificar el voltaje inducido y la corriente den la armadura[1].

La armadura tiene una bobina que está colocada en las ranuras de la misma. Los dos extremos de la bobina están conectados a dos anillos deslizantes montados en la flecha. Los anillos deslizantes están conectados a circuitos externos por medio de escobillas de carbón fijas, las cuales hacen contacto continuo con los anillos deslizantes. Es importante señalar que posteriormente, se explicará un poco más a fondo la construcción de un motor de corriente directa.

El campo magnético está establecido por una corriente directa fluyendo en las bobinas de excitación devanadas alrededor de los polos. Aun cuando el campo magnético que enlaza a la armadura es invariable en el tiempo, la magnetización de la armadura rotatoria se invierte dos veces en cada revolución de acuerdo a como las partes de la armadura se someten a la influencia de los polos alternamente.

La armadura gira por medio de un primotor, como ella gira, los enlazamientos de flujo asociados con la bobina cambian. Para expresar mas claramente como se desarrollan estos cambios de flujo, se puede utilizar la ley de Lenz[2], para determinar la polaridad de la fem generada en cada momento, esto se ve mas claramente en la siguiente figura.

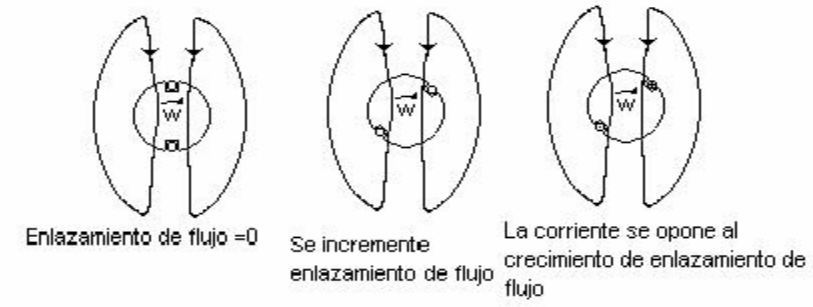


Fig. II.3.- Determinación de polaridades de la FEM generada por medio de la ley de Lenz

Esta figura, explica como el flujo invariante producido por los polos, produce un movimiento en el conductor que se encuentra en el rotor, forzando al rotor a un movimiento giratorio, determinado por la ley de Lenz, y que es la base para el funcionamiento de cualquier motor de corriente directa. Es obvio que esto solo es la base, pues hacen falta ciertas consideraciones, para tener el correcto y completo funcionamiento de esta máquina eléctrica, consideraciones que a pesar de ser tomadas en cuenta en el diseño de un motor de corriente directa, no consideramos pertinente exponer en esta tesis.

Los motores de corriente directa son más aptos que los motores de corriente alterna para tareas que demandan altos grados de control de posición y velocidad.

[1] Referencia: Máquina Eléctricas y Electromecánicas. Mastch (B1)
[2] Referencia: Electricidad y Magnetismo. Jaramillo;Alvarado (B3)

La corriente directa logra una velocidad ajustable del motor en rangos amplios, una salida de potencia mecánica constante, o par constante, una rápida aceleración y desaceleración y respuesta a una señal de retroalimentación[3]. Estas son algunas de las características que debe cumplir el motor a usar, pues se trata básicamente de un control de velocidad, aceleración y posición, con manejo de par y necesidad de rápida respuesta a cambios de aceleración mediante una señal retroalimentada.[4]

Los motores de CD presentan claras ventajas sobre los de AC para los fines del sistema propuesto, por lo que fueron el tipo de motor seleccionado. Más adelante se presentarán los motores de CD que se utilizaron en el sistema de vigilancia.

II.2.- CONTROL DE MOVIMIENTO.

La capacidad de orientar la cámara del sistema hacia el punto de interés con la velocidad requerida es parte esencial del desarrollo de esta tesis. Por ello se plantea la necesidad de tener pleno control sobre la velocidad, aceleración y posición de los motores del sistema. Como ya se estableció anteriormente, estos son motores de corriente directa, por lo cual se revisarán en este apartado métodos de control de motores de CD.

Para controlar motores de corriente directa, se pueden tener básicamente dos tipos de control, mediante la variación directa del voltaje de alimentación y mediante una modulación por ancho de pulso o PWM.

II.2.1.- Control por variación de voltaje

El movimiento de un motor de corriente continua en sentido y velocidad, depende en términos generales, de la polarización aplicada a sus terminales y del valor de tensión de dicha polarización.

Un voltaje o polarización positiva, producirá un movimiento en un sentido y una polarización negativa producirá un movimiento en el sentido opuesto. Esto es, el sentido de giro del motor depende de la polaridad del voltaje que se establezca entre sus terminales. En cuanto a la velocidad, se puede decir que ésta varía proporcionalmente a la magnitud del voltaje suministrado al motor. Esto es, a mayor voltaje de alimentación, mayor será la velocidad con que gire el motor, ésta última delimitada por las características propias del motor, que tendrá un límite en cuanto al voltaje que le puede ser suministrado.

En resumen, la polaridad del voltaje de alimentación nos proporciona el sentido de giro, y la magnitud de este voltaje es la que determina la velocidad del movimiento.

[3]Referencia. Conversión de Energía Electromecánica. Gourishankar (B2)
[4] Cita de *"Why DC?"* Departamento de corriente directa y generadores de General Electric Company Eire 1956"

II.2.2.- Control por Modulación de Ancho de Pulso (PWM)

El control de motores de DC por PWM trata de emular una señal de voltaje constante a partir de una señal periódica de pulsos rectangulares. La variación del ciclo de trabajo (ancho de pulso) de dicha señal periódica determina el valor promedio equivalente.

Consideremos un pulso de voltaje de amplitud A, y de duración t en un periodo T, como se ejemplifica la figura II.4.a :

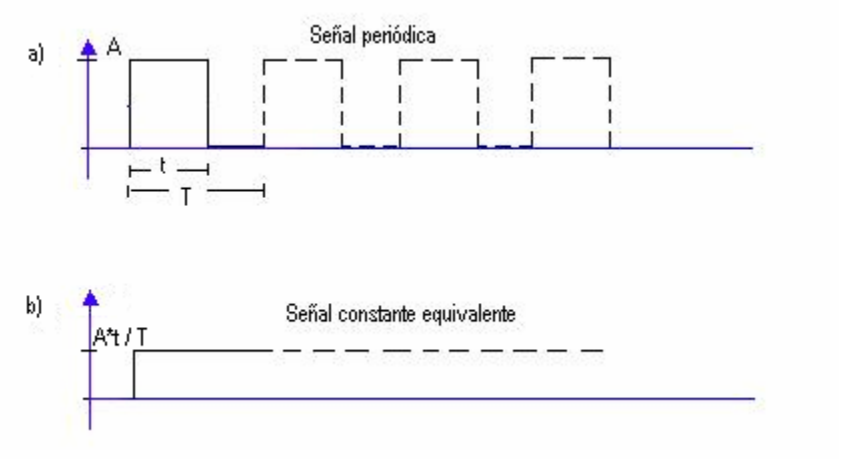


Fig.II.4 Modulación por Ancho de Pulso

el voltaje promedio de este pulso, está dado por la siguiente ecuación:

$$V = \frac{1}{T}\int_0^T A \cdot dt \qquad \text{...(2.1)}$$

De donde se tiene:

$$V = \frac{A}{T}\int_0^t dt \qquad \text{...(2.2)}$$

$$V = \frac{t}{T} \cdot A \qquad \text{...(2.3)}$$

De esta última ecuación (2.3) se puede ver que el voltaje promedio que produce un pulso depende directamente de la duración de éste. Una serie de pulsos de voltaje, repetidos con la suficiente velocidad, producirá sobre un motor de DC el mismo efecto que la señal constante equivalente A*t/ T (figura II.4.b).

Así, incrementando el ancho del pulso (ciclo de trabajo) de la señal periódica, se puede obtener en el motor el efecto equivalente a aumentar su voltaje de alimentación, y de manera contraria, reduciendo el ciclo de trabajo, se tiene el equivalente a un menor voltaje de alimentación.

Este método de controlar la velocidad, ofrece la ventaja de tener una gran vocación digital, pues los dispositivos digitales, son aptos para dar como salida dos niveles lógicos:

alto/encendido o bajo/apagado, además de que la generación de pulsos a altas velocidades es una tarea trivial para dichos dispositivos.

Existen en el mercado microcontroladores de uso específico para el control de motores, capaces de generar las señales de control descritas anteriormente (tanto PWM como variación de voltaje). Fue uno de estos microcontroladores el usado para el control de motores. Dicho microcontrolador se revisará a detalle en el siguiente capítulo.

II.3.- FASE DE POTENCIA.

Hasta este momento se han presentado generalidades de motores y la forma de controlar su movimiento. Como se ha establecido hasta ahora, el sistema propuesto es eminentemente digital, ello implica que las señales de control de motores son generadas por dispositivos digitales y por ello son de muy baja potencia, por lo cual no se pueden conectar directamente a los motores. Para solventar esta limitación, se hace necesaria una etapa de potencia, sobre la cual se tratará en seguida.

Cuando pretendemos controlar cargas con un microcontrolador, encontramos que no podemos realizarlo directamente. Si se quiere controlar desde un circuito digital convencional un dispositivo electromecánico (un motor, un relevador, o un alambre muscular) tal circuito no tendrá la capacidad de suministrar la potencia requerida para accionar dicho dispositivo. Debemos usar un manejador de potencia, que excitado por una señal de control proveniente del circuito digital, provea al dispositivo de los niveles de potencia que requiera.

La manera más sencilla de manejar un elemento electromecánico pequeño con un circuito digital es utilizando un transistor como interruptor. Así el circuito digital sólo prende y apaga el transistor, y el transistor es el que prende y apaga el motor.

La Figura II.5 presenta el circuito base que necesitamos para prender y apagar un motor eléctrico pequeño de corriente directa desde un circuito digital:

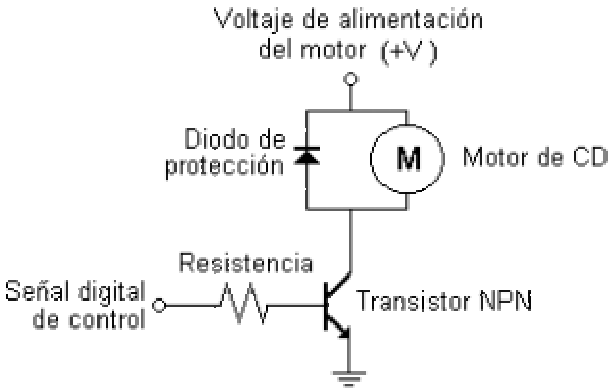


Fig.II.5.-Circuito básico para encender un motor de DC

Los dispositivos electromecánicos como los motores son altamente inductivos, es decir, un motor se opone a la variación del flujo magnético induciendo corrientes a través de sí. Al desenergizar un motor, se alcanzan las variaciones máximas de flujo magnético, lo que trae consigo altas corrientes inducidas y esto puede fácilmente dañar circuitos electrónicos, a menos que se implementen medidas de protección como la mostrada en la Figura II.5 en la que un diodo en paralelo sirve de "desahogo" de la corriente inducida.

Cuando un motor de DC debe moverse en dos sentidos es claro que con un circuito semejante al de la Figura II.5, no es posible hacerlo. El circuito denominado Puente H, está diseñado para realizar este tipo de manejo de los motores. Se ha de describir brevemente y en general el funcionamiento del puente H. Siendo este circuito el requerido en caso de utilizar un control de motores tipo PWM.

Por otro lado, cuando el control de movimiento del motor de CD se realiza por medio de variaciones de tensión, lo pertinente es un amplificador operacional de potencia, de manera que a la entrada de este amplificador se tenga una señal de voltaje de control y a la salida se tenga el voltaje y corriente necesarios para el movimiento del motor.

Para el desarrollo de la etapa de potencia del sistema se utilizaron los dos métodos descritos, un puente H para PWM y un amplificador operacional de potencia. Por lo cual se describirá brevemente el funcionamiento general de ambos (aún cuando finalmente se decidió por el control con PWM que ofreció mayores ventajas).

II.3.1.- Puente H

Es un circuito, diseñado específicamente para funcionar como excitador en motores de DC, y poder dar la libertad de manejar los dos sentidos de giro al motor. Existen distintas maneras de construirlo e implementarlo, con amplificadores operacionales, con transistores de potencia, etc. Revisaremos el principio de funcionamiento general y

posteriormente, se expondrá el puente H que se eligió, que en nuestro caso ya es una tarjeta implementada y adaptada específicamente para la controladora elegida.

II.3.1.1.- Diagrama esquemático del puente H y su funcionamiento.

Un puente H es conceptualmente un arreglo de CUATRO interruptores dispuestos de la siguiente manera:

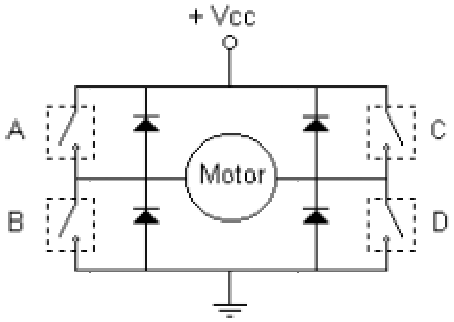


Fig. II.5.-Esquema del puente H

Si se cierran solamente los interruptores A y D la corriente circulará en un sentido a través del motor, y si se cierran solamente los contactos B y C la corriente circulará en sentido contrario. La implementación física de un puente H necesita de cuatro diodos de protección para el motor. Estos diodos pueden ser de preferencia de conmutación rápida.

Estos interruptores (A,B,C y D) pueden implementarse con transistores bipolares, mosfets, jfets, relevadores o cualquier combinación de los anteriores. El punto central es: el puente H se utilizan para que un motor eléctrico de corriente directa funcione en dos sentidos sin necesidad de una fuente de dos polaridades.

Por ejemplo, la implementación del puente H con transistores sería la siguiente:

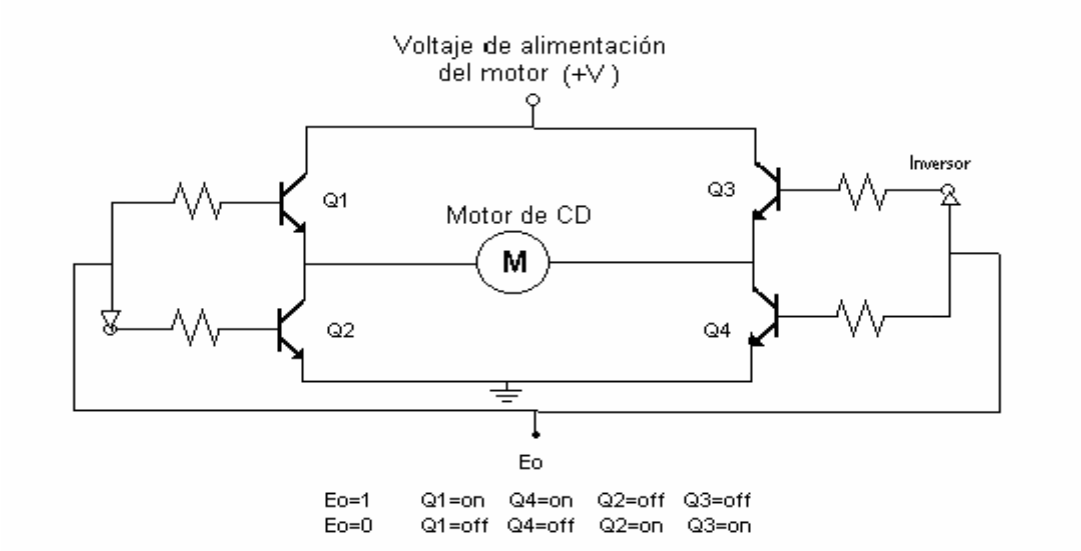


Fig.II.6.-Implementación con transistores de un Puente H

Esta implementación requiere únicamente de una señal de control para determinar el sentido de giro; se pueden tener una implementación con dos señales de control, dos combinaciones lógicas de las señales permiten elegir los dos sentidos de giro del motor.

II.3.2- El amplificador operacional de potencia.

Como se apuntó anteriormente el control de movimiento de un motor de DC se puede hacer variando directamente el valor de su alimentación. Dada la naturaleza digital del sistema propuesto, los requerimientos de este tipo de control son los siguientes:

- un sistema digital que produzca la señal de control en formato binario

- un convertidor digital a analógico con capacidad de excursiones de voltaje positivas y negativas (para posibilidad de movimiento en dos sentidos) para convertir la señal de control binaria a una señal de control analógica.

- una etapa de potencia que a partir de la señal de control analógica, genere la señal de potencia necesaria para accionar a los motores.

Este último requerimiento puede satisfacerse empleando un amplificador operacional de potencia en modo seguidor como el propuesto en la figura II.7

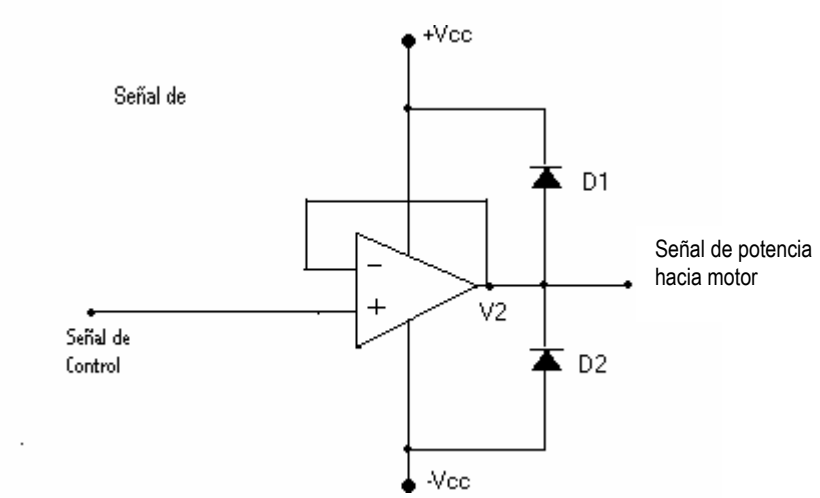


Fig. II.7.-Amplificador operacional de potencia

Las características de dicho amplificador son que tenga: capacidad de manejar la potencia máxima que requiera el motor; un nivel de *offset* despreciable comparado con los rangos de tensión necesarios para mover el motor; un CMMR grande que asegure que el movimiento del motor no se verá afectado por ruidos externos.

En la figura II.7 se pueden observar también un par de D1 y D2 a la salida del operacional. Estos diodos, como ya se mencionó, se colocan como protección por la naturaleza inductiva de la carga (motor).

II.4- CODIFICADORES

Parte imprescindible del control de los motores es conocer de manera permanente la posición angular de sus ejes de manera rápida y eficiente. Esto implica incluir sensores de posición en los motores, con el fin de conocer su ubicación, y colocar límites físicos (interruptores) que sirvan de referencia para establecer posiciones absolutas. Estos dispositivos que proporcionan información sobre la ubicación de las partes móviles del sistema, permiten la retroalimentación al sistema, cerrando el laso de control.

Actualmente no existe un método infalible y universal para calcular la posición, sino que, por el contrario, existen una serie de métodos basados en diversas técnicas que intentan resolver el problema. En la mayoría de los casos reales, la solución adoptada pasa por el empleo de varios de estos métodos.

En lo referente a la tarea de incluir sensores de posición, que nos permitan saber posiciones relativas de los motores, existen diferentes técnicas, y aun cuando nosotros utilizaremos un codificador incremental, resulta importante mencionar brevemente algunos otros métodos que existen, hablando claro está, de sensores de posición.

II.4.1.- Codificadores Incrementales.

Los codificadores incrementales o codificadores ópticos se utilizan fundamentalmente para el cálculo de la posición angular. Básicamente constan de un disco transparente, el cual tiene una serie de marcas opacas colocadas radialmente y equidistantes entre sí; de un elemento emisor de luz (como un diodo LED) y de un elemento fotosensible que actúa como receptor. El eje cuya posición angular se va a medir va acoplado al disco.

El funcionamiento es el siguiente: cuando el sistema comienza a funcionar el emisor de luz empieza a emitir; a medida que el eje vaya girando, se producirán una serie de pulsos de luz en el receptor, correspondientes a la luz que atraviesa los huecos entre las marcas. Llevando una cuenta de esos pulsos es posible conocer la posición del eje. Sobre este esquema básico es habitual encontrar algunas mejoras. Por ejemplo, se suele introducir otra franja de marcas por debajo, desplazada de la anterior, para poder controlar el sentido del giro; además suele ser necesario el empleo de una marca de referencia que nos ayudará a saber si hemos completado una vuelta. Realmente los codificadores incrementales miden la velocidad de giro, pero podemos derivar la posición angular. Como es lógico, la resolución de este tipo de sensores depende directamente del número de marcas que podamos poner físicamente en el disco.

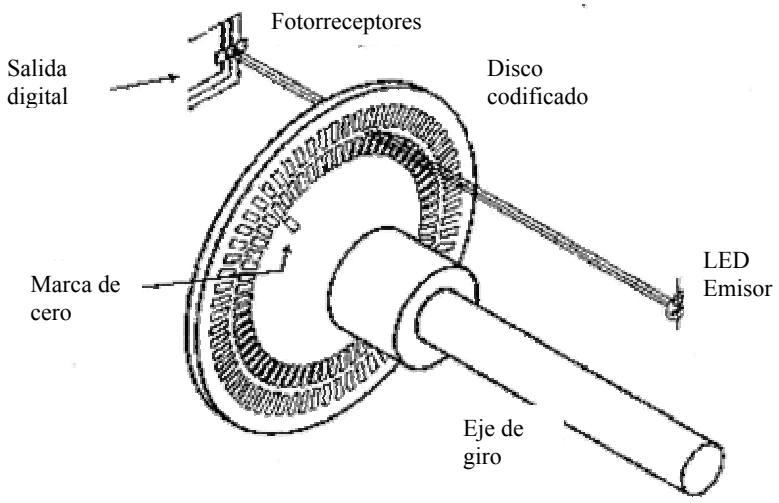


Fig. II.8.-Codificador Incremental (E9)

II.4.2.- Codificadores Absolutos

La función de este tipo de dispositivos es similar a la de los anteriores, medir la posición angular. Sin embargo en este caso lo que se va a medir no es el incremento de esa posición, sino la posición exacta. La disposición es parecida a la de los codificadores incrementales. También se dispone de una fuente de luz, de un disco graduado y de un fotorreceptor.

La diferencia estriba en la graduación o codificación del disco. En este caso el disco se divide en un número fijo de sectores (potencia de 2) y se codifica cada uno con un código cíclico (normalmente un código de Gray); este código queda representado en el disco por zonas transparentes y opacas dispuestas radialmente, como se puede apreciar en la figura. No es necesaria ninguna mejora para detectar el sentido del giro, ya que la codificación de los distintos sectores angulares es absoluta.

La resolución de estos sensores es fija y viene dada por el número de anillos que posea el disco, o lo que es lo mismo, el número de bits del código utilizado. Normalmente se usan códigos de 8 a 19 bits. Tanto los codificadores absolutos como los incrementales pueden presentar problemas debido a la gran precisión que es necesaria en el proceso de fabricación. Además son dispositivos especialmente sensibles a golpes y vibraciones.

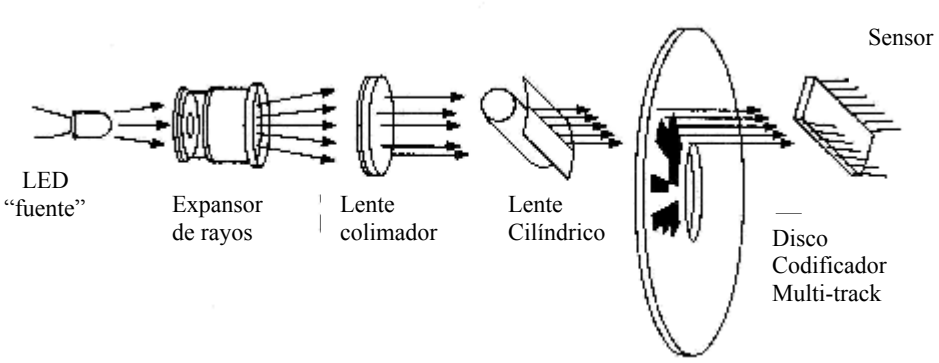


Fig. II.9.-Codificador Absoluto (E9)

II.4.3.- Potenciómetro

Los potenciómetros son unos dispositivos capaces de medir la posición angular y pequeños desplazamientos de posición lineal. Según el tipo de posición a medir tendremos dos tipos distintos de dispositivos pero la idea básica es común. Constan de una resistencia a través de la cual hay una determinada diferencia de potencial. Además hay un contacto

unido a la resistencia pero que se puede deslizar a su alrededor; este elemento es conocido como *wiper*. El *wiper* se conecta físicamente al elemento cuyo movimiento vamos a medir. Cuando este elemento se mueva el *wiper* se irá moviendo "a lo largo" de la resistencia y la tensión de salida en él irá cambiando. Si medimos está tensión de salida, podremos determinar cuanto se ha desplazado el *wiper*, y por lo tanto sabremos cuanto se ha desplazado el elemento que pretendemos controlar.

II.5.-PEDESTAL MECÁNICO

A fin de que la cámara de video del sistema pueda orientarse satisfaciendo los requerimientos ya establecidos, se necesita un pedestal mecánico que tendrá las siguientes funciones:

-servir de soporte físico al conjunto cámara-zoom,

-albergar a los motores actuadores,

-contar con la estructura mecánica que transforme el movimiento de los ejes de cada motor en movimientos de la cámara en los ejes de elevación y ronza (giro).

Las consideraciones mecánicas generales que se pueden hacer respecto a un pedestal están fuera del alcance y objetivo de está tesis. Por ello sólo se revisarán, hasta el capitulo III, las características particulares del pedestal usado, enfocándonos esencialmente en el reacondicionamiento que se le efectuó, pues el pedestal fue uno de los elementos que, como ya se mencionó, se "reciclaron" para esta proyecto.

II.6.- COMPUTADORAS

El sistema de vigilancia propuesto, contempla la utilización de dos computadoras para labores de control y procesamiento. Los objetivos inmediatos de está tesis podrían cubrirse con una sola computadora, pero, como ya se menciono en la introducción, buscamos implementar un sistema que eventualmente sea capaz de desarrollar tareas tan complejas y demandantes en recursos como es, por ejemplo, el seguimiento automático de blancos.

II.6.1.- Computadora anfitriona

Hemos nombrado *computadora anfitriona* a la computadora directamente involucrada con el control de movimiento de los motores. Se prefirió llamarla así y no *computadora de control* en virtud de que alberga a las tarjetas que resuelven casi totalmente el grueso del control de los motores[5]. Está computadora no requiere tener características sobresalientes, pues se contempla que es suficiente que corra con sistema

[4] Los pormenores se presentan en el siguiente capítulo.

operativo MS-DOS. Puertos paralelo y serial, así como ranuras de expansión, son los pocos requisitos que debe cumplir esta computadora.

La *computadora anfitriona* se planea para los objetivos inmediatos de este trabajo como un relevo, un traductor, un intermediario.

II.6.2.-Computadora de visión.

La computadora de visión realizará el manejo de las imágenes que se obtengan a través del sistema de captura de video, y así mismo, correrá una interfaz visual de usuario para el control del sistema.

Las funciones que esta computadora tendrá, imponen requerimientos y prestaciones relativamente elevadas. Al tener que manipular imágenes, ya sea meramente desplegándolas, o eventualmente efectuándoles algún procesamiento digital, la *computadora de visión* requerirá de una buena cantidad de recursos para cumplir con las exigencias de operación en tiempo real. La acotación puntual de dichos recursos dependerá del tipo de algoritmos que se pretenda ejecutar y de la eficiencia con que estén implementados.

II.7.- SISTEMA DE CAPTURA DE VIDEO

En este apartado revisaremos los conceptos generales referentes a la captura de video y su adecuación para manipulación digital.

II.7.1.-Cámaras CCD

Las cámaras que se usan en el campo del procesamiento digital de imágenes constan primordialmente de dos partes: la unidad de adquisición de imagen y la unidad de salida de imagen. La unidad de adquisición queda definida por el tipo de sensor que registra la señal luminosa que conforma la imagen. La unidad de salida es la encargada de interpretar los valores del sensor y convertirlos en una señal eléctrica de video conforme a un protocolo, estándar o no estándar, el cual define entre otros parámetros, el número de líneas horizontales y verticales por cuadro, así como el número de cuadros por segundo.

Los protocolos estandarizados son por ejemplo, los reconocidos por organismos internacionales como el CCIR[6] para Europa y EIA[7] para América y que se ajustan a monitores analógicos. En tanto que los protocolos no estandarizados tienen una orientación hacia cámaras digitales

[6]Comité Consultatif International des Radiocommunications: Comité Consultivo Internacional de Radiocomunicaciones
[7]Electronics Industries Association: Asociación de Industrias Electrónicas

II.7.1.1.-Adquisición de imagen.

Las cámaras CCD (Charge Coupled Device) se basan en un chip que consiste de un arreglo bidimensional[8] de sensores sensibles a la luz, donde cada uno de estos sensores puede verse como un píxel-sensor. La dimensión de tal arreglo de sensores definirá la resolución de la imagen que se obtenga de la unidad de salida de imagen.

Cada píxel-sensor del chip CCD transforma la luz (longitudes de onda entre 400nm y 1000nm para el espectro visible) en una carga, la cual a su vez es transformada en un voltaje. Los fotones incidentes en el píxel-sensor generan electrones que se acumulan en el sensor, el cual funciona como un capacitor y entrega un voltaje acorde a la cantidad de carga acumulada.

Además de la luz incidente, la temperatura ambiente genera electrones espurios, la llamada "corriente oscura", lo cual origina efectos no deseados (ruido). Como resultado, un chip CCD se puede "llenar" de electrones aún sin ser expuesto a ninguna fuente de luz. Este efecto no deseado se contrarresta adicionando sistemas de enfriamiento a los chips CCD, o en el caso de tomas estáticas, obteniendo varios cuadros y promediándolos para reducir el ruido (considerando que el ruido sea un evento aleatorio de media cero).

II.7.1.2.-Parámetros de importancia de la cámara CCD

El **tiempo de exposición** del chip CCD es el parámetro que determina la cantidad de tiempo que los sensores de un chip CCD están expuestos a la señal luminosa. El método tradicional de controlarlo para cámaras analógicas convencionales es con el uso de un obturador mecánico, pero en el caso de las cámaras CCD, el dispositivo que controla la exposición es un "obturador" electrónico también conocido como integrador.

Así mismo, para la tecnología CCD en vez de usarse el término "tiempo de exposición" se usa el término "tiempo de integración" el cual es más descriptivo, tomando en cuenta la naturaleza capacitiva de los sensores CCD. El obturador electrónico puede ofrecer tiempos de integración seleccionados "manualmente", o un AEL[9] que adapta automáticamente el tiempo de integración en función de la intensidad luminosa que se está recibiendo.

Si la intensidad de luz recibida es muy baja/alta, resultando en imágenes pobres/saturadas, la calidad de la imagen se puede manipular variando la ganancia que es parte de la generación de la señal, o modificando el tiempo de integración.

[8] Existen también las cámaras de escaneo progresivo que cuentan sólo con una hilera de sensores, que realizan un barrido para obtener una imagen bidimensional.
[9] AES: Automatic Electronic Shutter

Otro de los parámetros importantes para un chip CCD es su **resolución** expresada en el número efectivo de pixeles. Una resolución típica para los chips CCD es por ejemplo de 756x581 (columnas x líneas). El número total de pixeles suele ser mayor, y se explica debido a que el chip CCD tiene algunas líneas y columnas de "pixeles ciegos" conocidos como el área de oscuridad óptica

La **respuesta espectral** es otro parámetro de un chip CCD que nos indica el rango de frecuencias (o longitudes de onda) a las cuales el chip es sensible. La figura 1 presenta una característica típica para un chip CCD de respuesta en el visible (línea continua).

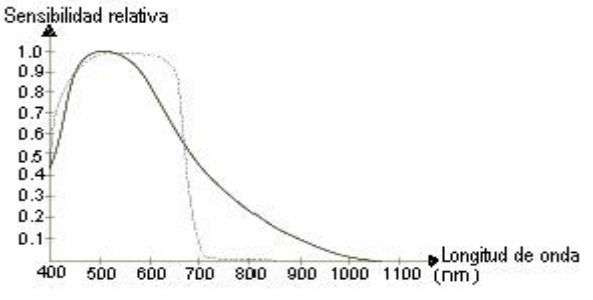


Fig. II.10.-Respuesta Espectral de un sensor CCD

Los chips CCD no tienen la "habilidad" inherente para discriminar el color (esto es diferenciar las distintas longitudes de onda dentro de su espectro de respuesta). Por lo tanto, en las cámaras a color la luz debe separarse en sus colores primarios verde, rojo y azul (esquema RGB[10]) por medio de filtros y prismas. La separación de los componentes de color para cámaras CCD sigue dos esquemas principales. El primero es separar los componentes RGB por medio de un prisma que dirige cada componente de color a un chip CCD separado. Las cámaras con está tecnología se llaman cámaras 3CCD, presentando alta calidad pero también alto costo.

Otra aproximación al color más simple y barata, utiliza una estructura de pequeños filtros, llamados filtros de mosaico, la cual distribuye cada componente de color de manera alternada en los pixeles de un solo chip, esto es, se tienen tres pixeles "vecinos", cada uno con los filtros necesarios que le hacen responder sólo a la componente de luz roja, verde y azul alternativamente.

II.7.2.-Unidad de salida de una cámara de video

La unidad de salida de una cámara genera la señal eléctrica apropiada para que el dispositivo subsiguiente (tarjeta de video, monitor analógico, monitor digital, *frame grabber*) realice con ella las operaciones adecuadas. Dado que primordialmente la señal de salida de una cámara va a un monitor analógico, ésta suele apegarse a alguno de los siguientes estándares internacionales: PAL (Phase Alternation Line) y SECAM (Séquentiel Couleur à Mémoire) para CCIR; y RS-170 y NTCS para EIA.

[10] RGB por las siglas en ingles Red Green Blue : Rojo Verde Azul

La unidad de salida se encarga de interpretar el nivel de carga en cada uno de los "pixeles" del chip CCD y asignarles una señal (que contiene información de tono e intensidad), la cual es "montada" en una señal portadora junto con pulsos de sincronía vertical y horizontal que indican el comienzo de un nuevo cuadro (o medio cuadro[11]) y el comienzo de una nueva línea respectivamente. El estándar de video determinará: el número de señales necesarias para codificar toda la información de video (y audio si fuera el caso); el número de pixeles[12] por línea; las líneas por cuadro y los cuadros por segundo.

II.7.2.1.-Escanéo progresivo y "trenzado"

Para formar una imagen en un monitor CRT (Catodic Ray Tube), ya sea éste digital o analógico, un grupo de cañones de electrones (tres para el monitor a color - uno para cada color primario -) controlados por la señal de video, proyectan un haz de electrones contra una pantalla luminiscente situada en el extremo opuesto de los tubos. Los haces de electrones "barren" la pantalla y así forman la imagen.

Existen dos tipos de sistemas de barrido de imagen que difieren en la técnica que utilizan para "pintar" una imagen completa en la pantalla. Los monitores compatibles con señales de televisión generalmente usan la técnica de "trenzado", en tanto los monitores digitales de las computadoras, usan la técnica de barrido progresivo. Este par de formatos son incompatibles el uno con el otro; es necesario que uno de ellos sea convertido al formato del otro para poder hacer operaciones comunes con ambos.

El barrido trenzado, consiste en que cada imagen, referida como cuadro, es dividida en dos sub-imágenes, referidas como campos. Un campo está formado por las líneas horizontales "pares" de la imagen y el otro por las "nones". La señal de video controla los haces de electrones para que impresionen todo un campo non en la pantalla, seguido de un campo par, esto es, se forma una imagen completa en dos barridos.

El barrido progresivo (o no entrelazado) consiste en formar la imagen de forma corrida; se barren todas las líneas horizontales en una sola pasada.

II.7.2.2.-Forma de onda típica de video analógico

En términos generales, una señal de video analógica estándar (que es la que entrega comúnmente la cámara CCD) es de tipo compuesto, lo cual significa que en una sola señal eléctrica se "montan" todos los parámetros que describen a las imágenes. Dicha señal combina la información de brillantez (*luma*), la información de color (*chroma*), e

[11] Dependiendo de si la señal es "trenzada " o no .(Ver sección de estándares de video)
[12]Se hace referencia al "píxel-imagen" del cuadro de salida a diferencia del "píxel-sensor" del chip CCD.

información de sincronización en una sola señal. En la figura se muestra una señal compuesta NTCS[13] que representa una línea horizontal de color blanco.

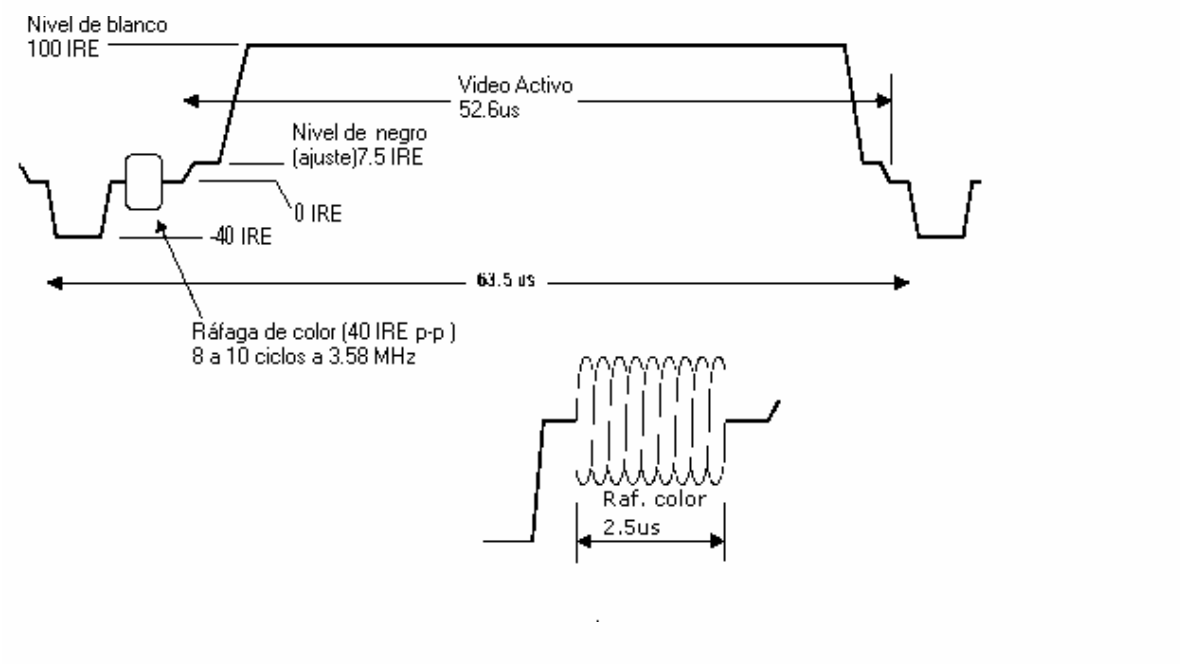


Fig II.11.-Forma de onda de video compuesto NTSC

Cada línea esta conformada por una porción activa que indica la intensidad y color de cada píxel de la línea, y una porción "inactiva" o de recuperación horizontal, en la cual el haz de electrones (inactivo) se reposiciona al inicio de la siguiente línea, y se envían las referencias de brillo y color. La brillantez está dada por la amplitud instantánea de la porción de video activo. La unidad de medida de la amplitud se da en términos de IRE's. IRE es una medida "arbitraria" en la que, por ejemplo, 140IRE=1V p-p. En la figura, también se establece por ejemplo el nivel de referencia del negro a 7.5 IRE

La información de color se agrega sobre el nivel de brillantez y es una onda senoidal cuya diferencia de fase respecto a la fase de la ráfaga de color (enviada antes de la porción activa) determina el color. Esto queda representado en forma esquemática en la siguiente figura:

[13] El formato Y/C separa la señal de luminiscencia de la de color, intentando así darles más integridad.

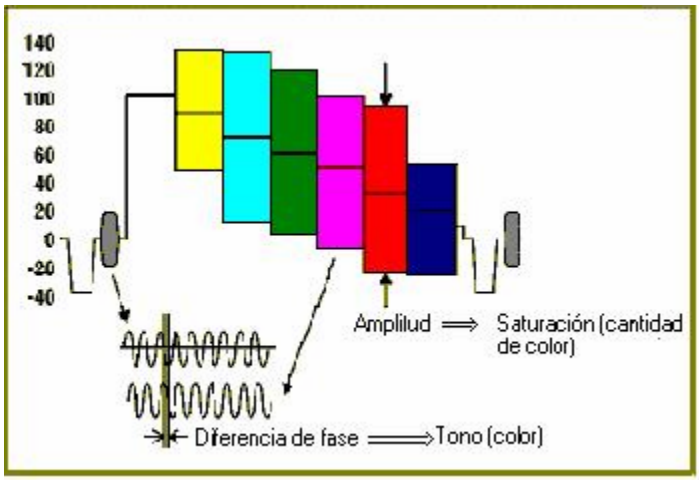


Fig.* II.12.-*Forma de onda de video compuesto: barras de color

II.7.2.3.-Tabla de formatos de video

Se presenta un resumen de los principales formatos de video analógico.

Tabla 1. Frecuencias típicas para formatos de video de TV y computadoras

Formato de video	NTSC	PAL	HDTV/SDTV	VGA	XGA
Descripción	**Formato de Televisión para Norteamérica y Japón**	**Formato de Televisión para la mayoría de Europa y Sudamérica**	**Alta Definición(HD)/ Definición Estándar(SD) para Televisión Digital**	**Video Graphics Array (PC)**	**Extended Graphics Array (PC)**
Resolución Vertical (líneas visibles por cuadro)	Aprox. 480 (525 líneas totales)	Aprox. 575 (625 líneas totales)	1080 o 720 o 480; 18 formatos distintos	480	768
Resolución horizontal (píxeles visibles por línea)	Determinado por el ancho de banda, va de 320 a 650	Determinado por el ancho de banda, va de 320 a 720	1920 o 704 o 640; 18 formatos distintos	640	1024
Frecuencia horizontal(kHz)	15.734	15.625	33.75-45	31.5	60
Cuadros por segundo	29.97	25	30-60	60-80	60-80
Ancho de banda	4.2	5.5	25	15.3	40.7

II.7.3.-*Frame Grabber*

Cuando la señal de video que que entrega una cámara de video tiene un formato *analógico*, no es apta para ser desplegada en un monitor de PC y mucho menos para realizar con ella procesamiento digital más complejo.

A fin de superar la dificultad anterior, se hace uso de un *Frame Grabber* o *Tarjeta Adquisidora de Imagen*, la cual "captura" las imágenes en formato analógico y las convierte a un formato digital para manipulación (digital) posterior. La Figura II.13 presenta un esquema del conjunto CCD-*Frame Grabber*:

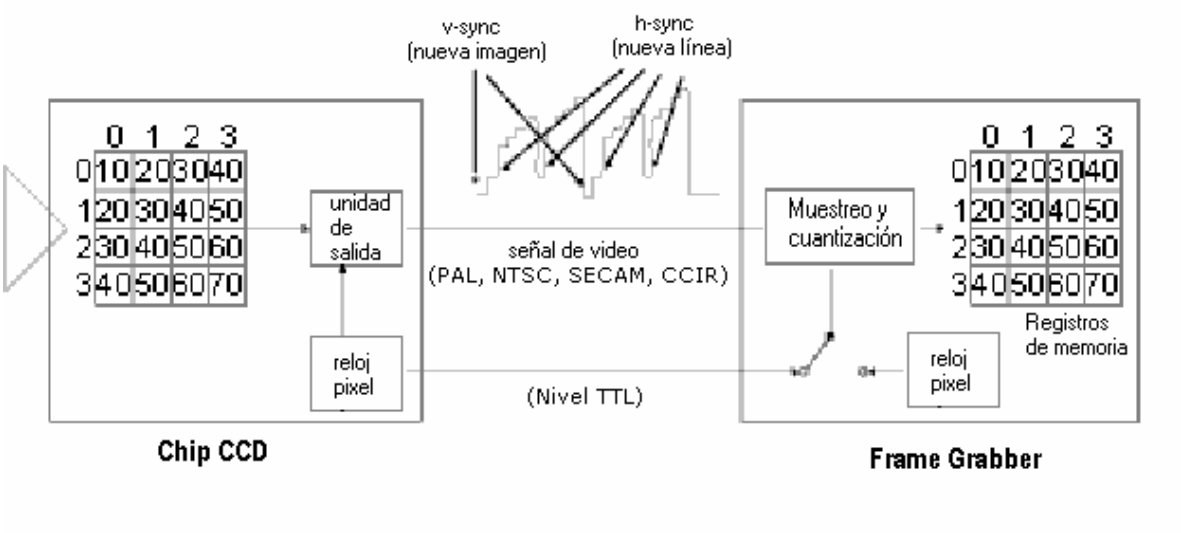


Fig. II.13.-Esquema del conjunto CCD-FrameGrabber

La tarea del *Frame Grabber* (también conocido como digitalizador) es muestrear y cuantizar la señal de video analógico y almacenar el resultado en la llamada *memoria intermedia de cuadro*[14] .El muestreo, recordemos, es el proceso de transformar una señal analógica continua en el tiempo en una señal discreta en el tiempo, en tanto la cuantización se refiere a "mapear" un rango continuo de valores de amplitud, a un conjunto limitado de valores.

La frecuencia de muestreo del *Frame Grabber* está dada por el llamado reloj de píxel (*píxel clock*) y ésta determinará la resolución espacial de la imagen digitalizada y a su vez estará en función del formato de la señal analógica de video de entrada y del formato de la señal digital de video de salida.

[14] *Frame Buffer*: memoria residente en el mismo *Frame Grabber* o en el sistema.

Los píxeles digitalizados son almacenados en un *buffer* de imagen tipo FIFO, que es capaz de almacenar cuando menos, un cuadro digitalizado completo. Este *buffer* es más usado en tanto el bus que comunica al *Frame Grabber* y a la PC tenga un ancho de banda estrecho. No obstante los sistemas de bus modernos como el PCI son tan rápidos que el *Frame Grabber* requiere apenas un buffer de unos cuantos kilobytes para compensar algunas irregularidades en el flujo de datos hacia la computadora a la que está conectado, de tal forma que el video digitalizado puede ser enviado con prontitud a la memoria de trabajo para procesamiento, o ser dirigido directamente a la tarjeta gráfica para ser visualizado en tiempo real.

II.8.-COMUNICACIÓN ENTRE DISPOSITIVOS DIGITALES

Los puertos de E/S constituyen el medio por el cual el microprocesador de una computadora se comunica con su entorno. Existen puertos para cada interacción de la unidad de procesamiento principal con sus dispositivos auxiliares.

En este apartado revisamos conceptos generales acerca de comunicación entre dispositivos digitales.

II.8.1.- Comunicación Serial

La comunicación serial, establece entre dos dispositivos un intercambio de datos bit por bit. Esto significa que cada palabra, cada byte de información, se transmite descomponiéndolo en sus bits constitutivos y enviando sucesivamente cada uno de ellos.

La comunicación serial se presenta en dos formas básicas para los microcontroladores y microprocesadores: síncrona y asíncrona. En la síncrona, los dispositivos comunicados se sincronizan a nivel de palabra, esto es, se establece el instante preciso en que las palabras deben ser transmitidas/recibidas. Por otro lado, en la comunicación serial asíncrona, la recepción/transmisión de cada palabra es aleatoria, y lo que se sincroniza es el instante preciso en que cada bit es transmitido/recibido.

Para establecer cualesquiera de los dos tipos de comunicación serial, es necesario adoptar un código que permita la transferencia de datos. Hablando de bits, se tienen dos estados que pueden definir su estado: 0 o 1, alto o bajo. Cada palabra de un código consta de *n* bits, de donde el número de palabras del código es igual a 2^n-1. A este número de bits se le puede agregar bits de paridad para detección y/o corrección de errores.

II.8.1.1- Comunicación Serial Asíncrona

La comunicación serial asíncrona comúnmente se resuelve con un dispositivo UART (Universal Asynchronous Receiver/Transmitter) que recibe en forma paralela un byte de información a transmitir y lo envía en forma serial utilizando alguno de los

protocolos de transferencia serial estándar como el RS-232, usado de forma genérica por los puertos seriales de las computadoras.

Este protocolo maneja dos tipos de códigos llamados "marca" y "espacio", el estado alto de voltaje, se considera como el uno lógico, y el estado bajo es considerado 0 lógico. La diferencia entre los dos códigos radica en cual de los estados es el estado que se mantiene "estable", esto es, cual de los estados es el que se tiene cuando no hay transmisión de datos.

A la comunicación serial asíncrona la definen los siguientes parámetros: velocidad de transmisión (bits por segundo), longitud de palabra, bits de paro y de paridad. De esta manera, al recibir el bit de inicio, se sabe cuantos bits subsecuentes constituyen la palabra de datos y cuantos bits son sólo de control.

Se transmiten siempre el mismo número de bits, esto es, un número de bits idéntico para cada palabra transmitida. El inicio de la transmisión de los datos es totalmente asíncrona, pues no se sabe el momento exacto en el que comienza la transmisión, pero una vez que ha comenzado, se sabe cuanto durará. Esto debido, a que el primer bit que se envía siempre es un bit de inicio que hace que cambie el estado "estable" de la comunicación, después se envía cierto número de bits que es la palabra que contiene la información, y posteriormente se pueden incluir bits de paridad para seguridad, y se termina la transmisión con un bit de paro[15], que es del mismo valor lógico que el bit de arranque. Los datos son transmitidos del bit menos significativo (LSB) al bit más significativo (MSB). Obviamente el bit menos significativo es precedido por el bit de arranque. Los bits de paridad o detección de errores son agregados después del MSB. Y el último en transmitir es el bit de paro. El bit de paro indica la separación entre palabra y palabra

Se puede observar que la comunicación es asíncrona, en el sentido de que no se sabe cuando se tendrá transmisión de datos, pero síncrona en cuanto a la duración de cada bit de la palabra. No se sabe cuanto espacio habrá entre palabra y palabra, pero sí se sabe cuanto durará la transmisión de cada palabra.

II.8.2.- Comunicación Paralela

El método de transferencia paralela entre dispositivos digitales, se basa en múltiples líneas físicas de datos, lo que permite enviar/recibir simultáneamente un conjunto de varios bits.

El puerto paralelo se utiliza generalmente para manejar impresoras. Sin embargo, dado que tiene un conjunto de entradas y salidas digitales, se puede emplear para leer datos y controlar dispositivos en general. De esta manera, el puerto paralelo se puede utilizar como una interfaz de entrada/salida que funcione subordinado a rutinas de software.

[15] El parámetro "bit de paro" toma comúnmente los valores 1,1.5 y 2

El puerto paralelo se identifica por su dirección de I/0 (entrada/salida) base y se identifica ante sistemas MS-DOS por el número LPT. Las direcciones estándar para el puerto paralelo son 03BCh, 0378h y 0278h.

El puerto consiste en tres registros de 8 bits ubicados en direcciones adyacentes del espacio de I/O de la PC. Los registros se definen relativos a la dirección de I/O base y son:

IOBase+0: registro de datos
IOBase+1: registro de estado
IOBase+2: registro de control.

El **registro de datos** permite escribir (o leer si el puerto es bidireccional) un dato en los pines 2 al 9 del conector del puerto (ver Figura II.14). Si se lee el registro, se verifica el ultimo dato escrito (o el último dato recibido si el puerto es bidireccional). La corriente que pueden entregar los pines es de 2.6 mA y pueden "drenar" un máximo de 24 mA.

El **registro de estado** conformado por cinco bits, es únicamente de lectura.

El **registro de control** conformado por 4 bits, es sólo de escritura.

En seguida se presenta una tabla con la descripción general y correspondencia entre los bits de los registros mencionados y los pines del conector del puerto paralelo:

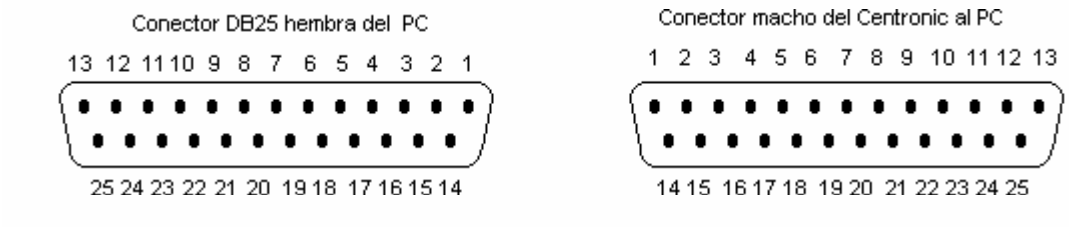


Fig.II.14.-Disposición del puerto paralelo

DB25	Señal	Registro	Tipo	Descripción
1	Control 0	C0 -	Salida	Bajo por mas de 0.5 µs para habilitar a la impresora, para recibir los datos enviados
2	dato 0	D0	Salida	
3	dato 1	D1	Salida	
4	dato 2	D2	Salida	
5	dato 3	D3	Salida	
6	dato 4	D4	Salida	
7	dato 5	D5	Salida	
8	dato 6	D6	Salida	

9	dato 7	D7	Salida	MSB de datos
10	Estado 6	S6+	Entrada	Pulso bajo de aprox. 11µs, indica que se recibieron datos en impresora y que está preparada para recibir más datos
11	Estado 7	S7 -	Entrada	En estado alto nos indica que a impresora está ocupada.
12	Estado 5	S5+	Entrada	En alto indica que no hay papel
13	Estado 4	S4+	Entrada	En alto para impresora seleccionada
14	Control 1	C1 -	Salida	En bajo, el papel se mueve una línea luego de la impresión.
15	Estado 3	S3+	Entrada	En bajo indica error
16	Control 2	C2+	Salida	Si enviamos un pulso en bajo la impresora se reinicia
17	Control 3	C3 -	Salida	En bajo seleccionamos impresora
18-25	Ground			

Tabla. Características de pines del puerto paralelo para controlar la impresora.

Las direcciones I/O de los registros del puerto paralelo se almacenan en una tabla, de manera que se pueda obtener acceso a ellos por medio de distintos lenguajes como Pascal, Quick Basic, Windows, C, Delphi, etc. En sistemas Windows la gestión del puerto se complica pues el sistema puede negar el acceso a la dirección del puerto.

En este proyecto se manipuló el puerto paralelo a través de Delphi 5.0, utilizando instrucciones de ensamblador. Se usaron las instrucciones básicas *mov*, *in* y *out*[16].

[16] Consultar el código principal del proyecto CAM_VIG_1.0 en el **Anexo de Programación**

CAPÍTULO III

● *SISTEMAS MECÁNICO, DE CONTROL Y DE VIDEO*

En este capítulo se revisarán los componentes específicos utilizados para la integración del sistema en lo referente a *hardware* para los módulos mecánico, de control y de visión. Para cada componente se revisa con detalle sus características y se describe la forma en que interactúa con el resto de los componentes del sistema.

III.1.-SISTEMA MECÁNICO

III.1.1.- Pedestal Mecánico

El pedestal mecánico es el soporte físico del conjunto cámara-zoom-codificador y de los motores. Así mismo, es la estructura mecánica que transforma los movimientos de los ejes de cada motor en movimientos de la cámara de video en los ejes de ronza y elevación (Figura III.1).

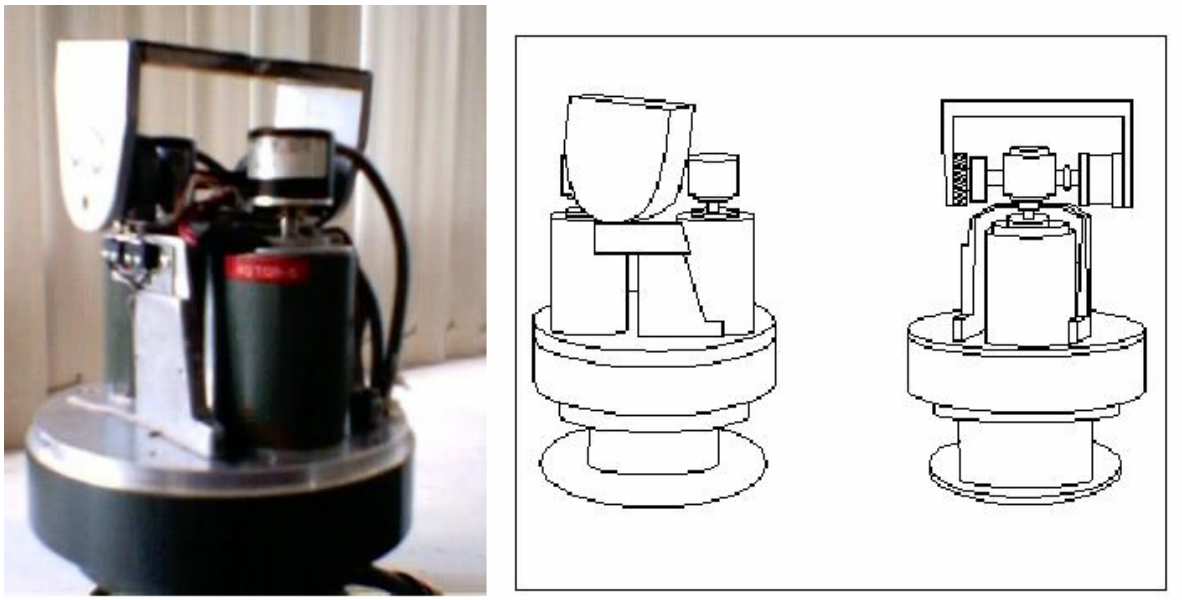


Fig. III.1.-Pedestal: Fotografía y Vistas

Este pedestal está diseñado para convertir los movimientos de dos motores en movimientos de elevación y ronza. Dicho pedestal fue diseñado originalmente para dar soporte a dos motores de AC sin codificadores. Estos motores de AC, eran de dimensiones más pequeñas y con un eje de menor calibre que los motores de DC utilizados.

El pedestal cuenta con un sistema de engranes o planetario, para convertir los movimientos del eje en movimientos de todo el conjunto mecánico. Un tornillo de plástico sinfin, colocado sobre los ejes de los motores, es el encargado de transmitir el movimiento giratorio al sistema de engranes. Cada uno de los ejes tiene un tornillo sinfin idéntico, pero el sistema al que mueve cada tornillo es diferente, pues se requiere un movimiento diferente para cada motor.

Para nuestra aplicación era necesario acoplar los motores de DC con el pedestal y con el sistema mecánico del mismo, por lo cual se hicieron algunas modificaciones físicas al sistema. La parte mas importante, fue la de acoplar el tornillo de plástico sinfin a los ejes de los motores. Como se mencionó, el largo y ancho de los ejes de los motores de DC que se usaron, eran mayores que los motores originales de AC del pedestal. Por ello se realizaron cortes a los ejes, a fin de reducir su longitud y rebajar su diámetro, de forma que el tornillo sinfin embonara adecuadamente.

Otro requerimiento que planteaba el usar el pedestal ya descrito, era el de ajustarlo al mayor tamaño de los motores de DC que usaríamos, agregando el hecho de que los codificadores de posición aumentan a lo largo el espacio de alojamiento para el conjunto motor-codificador. Para esto, se realizó un desbastado circular de unos 3mm de profundidad en la parte del pedestal que soporta los motores, y posteriormente se desbastó el pedestal por la parte de arriba para ajustar el motor, haciendo de esta manera que los motores quedaran "aprisionados" entre el soporte bajo y la parte superior del pedestal. Fue necesario realizar nuevas perforaciones a la base para asegurar los motores por medio de tornillos. Todo este trabajo mecánico, fue hecho observando que los nuevos motores de DC mantuviera el mismo centro que los motores anteriores de AC, para no modificar el sistema mecánico.

III.1.2.- Motores Maxon

Para orientar el pedestal en elevación y ronza, de modo que el campo de visión de la cámara sea todo un hemisferio, se hace uso de un par de motores de DC de la marca Maxon[1]. Estos motores brindan altas prestaciones para labores de automatización como son:

-baja inercia, lo que permite aceleraciones y cambios de sentido rápidos

-saturación magnética casi nula, lo que implica constante velocidad-torca lineal, que se traduce en control más simple y preciso

-alto nivel de torca en relación a sus dimensiones: 40W para un tamaño de 9cm de largo y 6cm de diámetro

[1] Ver hoja de especificaciones en anexo **Características del sistema.**

Fig. III.2.- Motor Maxon una vez instalado en el pedestal

Toda una serie de parámetros y gráficas características definen el desempeño del motor, estando entre las más descriptivas las constantes de torca y velocidad, y la línea torca-velocidad, las cuales se presentan a continuación.

III.1.2.1.-Constante de velocidad

La constante de velocidad $\mathbf{K_n}$, relaciona directamente la velocidad del motor **n** (en rpm), con el voltaje **U** (en volts) inducido en su embobinado, tal que:

$$\mathbf{n = K_n * U}$$

y para el motor utilizado es de 158 rpm/V.

III.1.2.2.-Constante de torca

La constante de torca $\mathbf{K_m}$, relaciona directamente la torca mecánica **M** que proporciona el motor, con la corriente **I** que está consumiendo:

$$\mathbf{M = K_m * I}$$

y para el motor utilizado es de 60.3 mNm/A.

III.1.2.3.-Línea velocidad-torca

La curva Velocidad-Torca de la figura III.3 describe la velocidad n del motor en función de la torca M a un voltaje constante U, y arroja el siguiente resultado importante:

-la velocidad del motor se reduce linealmente cuando la torca aumenta

así que, entre más rápido gira el motor, menor es la torca que puede proporcionar. La curva velocidad-torca, está definida por dos puntos; la velocidad del eje del motor sin carga y la torca que hace que el motor se detenga. Conforme el voltaje U aplicado cambia, varían los dos puntos mencionados, pero se describen líneas velocidad-torca paralelas, esto es, todas tienen la misma pendiente. Tal pendiente se denomina *gradiente velocidad-torca* $\Delta n/\Delta M$

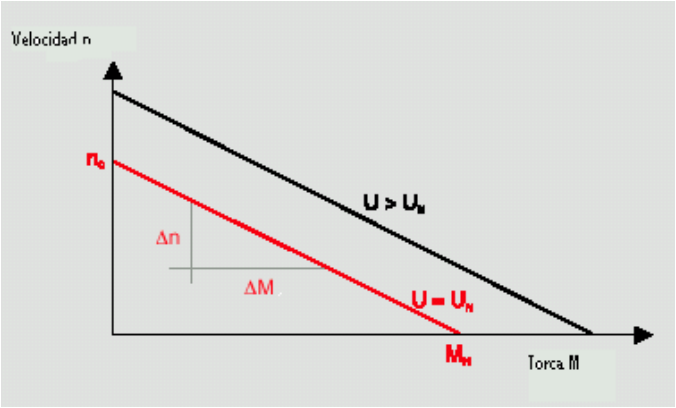


Fig.III.3.-Línea Velocidad-Torca

Los parámetros y curvas presentadas sirven para hacer cálculos que caractericen al sistema mecánico (ver anexo **Características del Sistema)**.

III.2.- SISTEMA DE CONTROL

En este apartado se revisan los componentes que participan de manera directa en el control de movimiento de los motores del pedestal. Comenzamos hablando de los microcontroladores LM628 y LM629 que realizan el grueso del control de los motores. Después se describe la tarjeta 4I27 que integra dichos microcontroladores con el hardware periférico necesario para su gestión a través de una computadora. Por último se detalla el codificador de posición *Accu-Codder* que proporciona la señal de retroalimentación en base a la cual el microcontrolador LM62X realiza el control de movimiento

III.2.1.-Microcontroladores LM628 y LM629

Los microcontroladores LM628 y LM629 son microcontroladores de *National Semiconductors* diseñados específicamente para el control de motores por medio de variación de voltaje y de PWM respectivamente. Dichos microcontroladores, en conjunción con los codificadores de posición, son la base del control de los motores del sistema[2].

El microcontrolador LM629[3] tiene dos modos de operación; velocidad y posición. En el modo de velocidad se programan la tasa de aceleración y velocidad, y una vez que se manda el comando de comenzar el movimiento, el motor empieza a acelerar a la tasa programada hasta alcanzar la velocidad especificada, que se conservará en tanto no se envíe el comando de parar. Este modo de operación se utilizó sólo en un par de funciones[4] .

En el modo de posición, que es el que se usa principalmente, los parámetros que se cargan al LM629 son los coeficientes de un filtro PID dentro de un lazo de control y cuyos valores determinan la magnitud de la fuerza restauradora que llevará/mantendrá al eje del motor en la posición deseada. También se cargan los parámetros de trayectoria: Posición Deseada, Velocidad Máxima y Aceleración. Con estos 3 valores el microcontrolador LM629 calcula un perfil de velocidad trapezoidal como el mostrado:

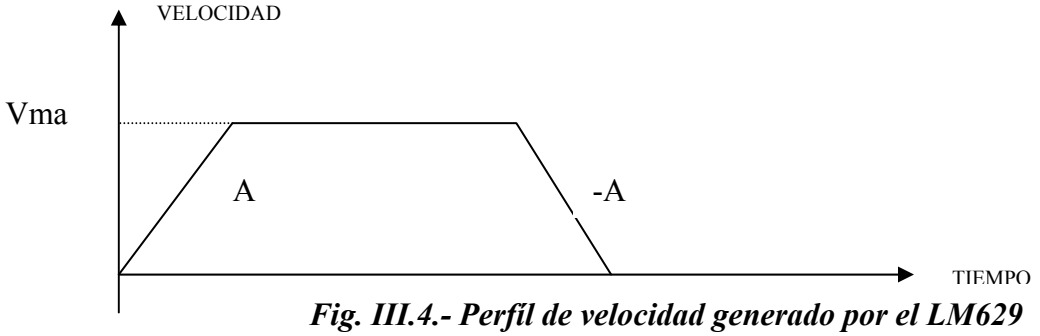


Fig. III.4.- Perfil de velocidad generado por el LM629

[2] A partir de este punto se hará referencia sólo al LM629, pues a pesar de haberse probado también el LM628, el que se uso finalmente fue el LM629. No obstante el modo de operación de ambos es el mismo, variando sólo en la señal de control que entregan.
[3] Ver hojas de especificaciones en el anexo **Características del sistema.**
[4] Ver **Capítulo IV**: función de inicialización RI y función de *joystick* RJ.

En este perfil que describe la velocidad del eje del motor, se acelera a una tasa constante A hasta alcanzar una velocidad máxima, la cual se conserva hasta que los cálculos del LM629 indican que es tiempo de empezar a desacelerar (también a una tasa constante -A), de modo que se alcance la posición deseada y la velocidad cero al mismo tiempo.

La integral bajo la curva corresponde al recorrido total del eje del motor.

El LM629 realiza su acción de control por medio de un filtro PID (proporcional, integral, derivativo) digital que trae implementado, y sobre el cual se pueden cambiar los parámetros de ganancia de cada acción a fin de lograr un sistema estable y una mejor respuesta del servomecanismo. Estas acciones se detallan a continuación.

III.2.1.1.-Acción de control del microcontrolador LM629

El LM629 basa su acción de control en las señales que recibe del codificador de posición y en un filtro digital PID. El siguiente diagrama servirá para explicar el proceso de control del LM629:

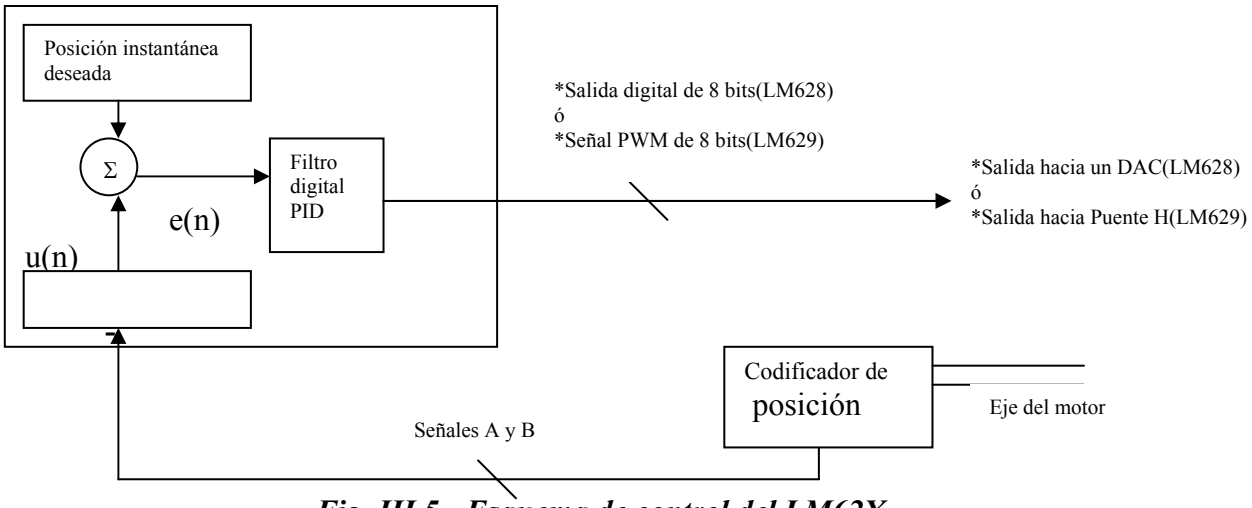


Fig. III.5.- Esquema de control del LM62X

Con los datos cargados de 32 bits de aceleración, velocidad y posición final, el LM629 genera una posición deseada *instantánea* que sirve de valor de referencia para el lazo de control, el cual se actualiza cada periodo de muestreo (que es de 256μs para el cristal de 8MHz con que opera el LM629). La posición real se resta de la deseada, lo cual es una operación de 32 bits que se trunca a los 16 bits menos significativos para generar la señal de error e(n).

La señal de error es procesada para generar la salida u(n) de la siguiente forma:

$$u(n) = Kp*e(n) + Ki*\sum_{0}^{n} e(n) + Kd*[e(n') - e(n'-1)] \quad .....(2.4)$$

donde Kp, Ki, Kd son las constantes de los términos proporcional, integral, y derivativo respectivamente. Todos los términos de (2.4) son de 16 bits (excepto el termino integral que se satura a 24 bits pero de los cuales se toman sólo los 16 más significativos), por lo que al multiplicarse y sumarse generan un producto de 32 bits de los cuales se toman los 8 bits más significativos para formar la salida de control[5].

El término e(n') representa el error en el último tiempo de *muestreo derivativo* y el término e(n'-1) representa el error en el penúltimo tiempo de *muestreo derivativo*. El tiempo de *muestreo derivativo* es programable y es un múltiplo del tiempo de muestreo (desde 1 hasta 256). El término derivativo actúa en cada tiempo de muestreo, pero se actualiza sólo cada tiempo de *muestreo derivativo.*

El término integral por su parte, se compara con un valor programable il. El menor de ellos es el que contribuye a la suma u(n).

Algunas anotaciones breves pueden hacerse respecto a la acción de cada término en el filtro. El término Kp*e(n) contribuye con una fuerza directamente proporcional al error; entre más grande es el error mayor es esta fuerza y más rápido se puede alcanzar el valor de referencia. Pero un efecto contraproducente de está rapidez, es que genera sobrepaso y eventualmente oscilaciones. Es por ello que para compensar el efecto proporcional se tiene un término derivativo que actúa a modo de amortiguador. El término derivativo kd*[e(n)-e(n-1)] nos muestra que entre más rápido se reduzca el error -digamos por una acción proporcional demasiado grande- mayor es el efecto de amortiguamiento que reduce el posible sobrepaso.

Por su parte la acción integral tiene el fin de contrarrestar torcas que de forma continua actúan sobre el eje y tienden a sacarlo de la posición deseada.

El microcontrolador LM629 cuenta con una serie de 23 comandos con los cuales, en términos generales, se programa su funcionamiento, se conoce su estado de operación y se configuran las interrupciones hacia la computadora anfitriona. Se presenta una lista y explicación de los comandos en el *Anexo* **Características del Sistema**, esta lista, se encuentra detallada dentro de las hojas de especificaciones del LM629.

[5] La señal de control que entrega el LM628 es un código signado de 8 bits , en el que una salida de 0xFF representa el máximo valor positivo de la escala, 0x80 es el cero de la escala y 0x00 es el máximo valor negativo de la escala.
La señal de control que entrega el LM629 es una señal PWM de 8 bits, lo que significa que el ciclo de trabajo varía de 0 a 100% en pasos de 0.39% aproximadamente

Los microcontroladores LM629 son el "alma" de la tarjeta controladora 4I27A, pues son estos los encargados de hacer los cálculos que generan los perfiles de velocidad de los motores en base a los parámetros programados. Siendo la tarjeta controladora 4I27A, una tarjeta que integra dos microcontroladores LM628 y facilita su operación a través del bus ISA.

II.2.2-Codificador de posición *Accu-Codder*

El requisito primordial para la acción de control del LM629 recién descrito, es conocer la posición actual del eje del motor cuya velocidad, aceleración y posición está controlando.

Para el seguimiento de la posición real del eje del motor se requiere un codificador de posición incremental en cuadratura, para empatar con la señal de entrada requerida por los microcontroladores LM629. El codificador utilizado, es un *Accu-Coder* de 4096 líneas, 2 canales de salida A y B en cuadratura, y una salida tipo índice.

Lo anterior significa que por cada revolución del eje al cual esté sujeto el codificador, este último entregará 4096 formas de onda cuadradas por cada canal y un pulso índice por cada revolución. Por convención, si la dirección del eje es positiva la señal A adelantará a la B por 90° grados eléctricos y si la dirección es negativa B adelantará a A por 90 grados eléctricos.

Debido a que las dos señales cuadradas A y B están desfasadas 90 grados una de otra, en cada periodo existen 4 cambios de flanco (Figura II.7), por lo que los microcontroladores LM629 de la tarjeta controladora interpretan las 4096 líneas como 16,384 pulsos (4096*4=16,384). Esto significa que se puede tener una precisión o resolución en el direccionamiento de los ejes de 0.383 miliradianes(2Π / 16,384=0.383E-3).

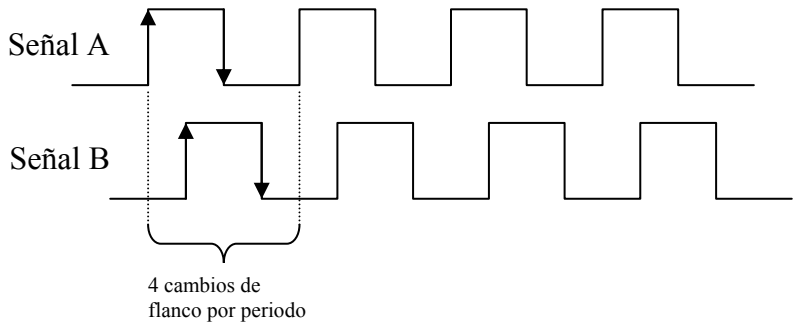


Figura II.7.-Salidas del codificador de posición incremental Accu-Codder

Fig. II.8.- Codificador ACCU CODER

III.2.3-Controladora 4I27

La tarjeta controladora 4I27 de *MESA Electronics* realiza control de motores de CD por variación de ancho de pulso PWM. Esta tarjeta, es una tarjeta de expansión que se comunica con su *computadora anfitriona* a través de un conector PC/104 (bus ISA), y cuya gestión se realiza por medio de los registros I/O cuya dirección base se configura por medio de un juego de *jumpers*.

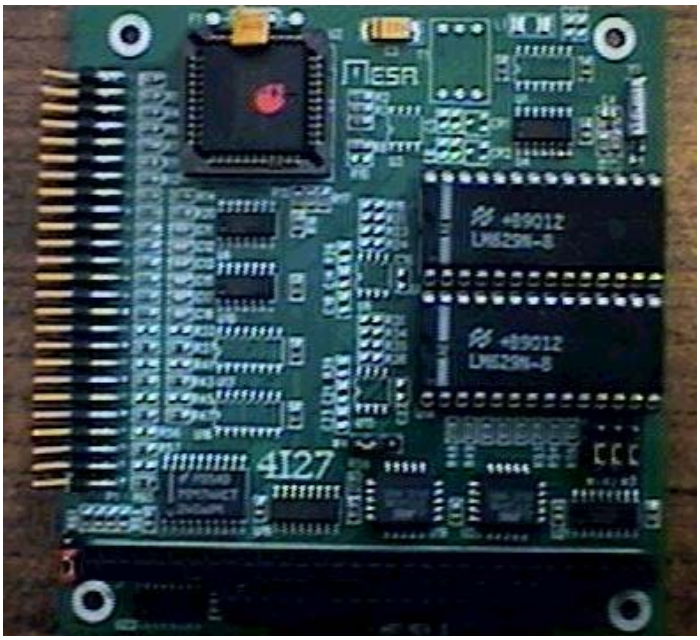


Fig. II.6.- Tarjeta controladora 4I27

La tarjeta controladora 4I27[6] integra dos microcontroladores LM629 con el hardware periférico necesario para facilitar la comunicación entre estos y la computadora anfitriona.

La 4I27 también ofrece tres puertos paralelos: un primer puerto de salida para habilitar los motores y seleccionar el número y tipo de interrupción asignada a la 4I27; un segundo puerto de entrada para sensores varios (por ejemplo de sobretemperatura); y un tercer puerto I/O configurable para usos múltiples.

Las características de control de la 4I27A quedan definidas por las capacidades microcontroladores LM629 que ya se han revisado.

III.2.4-Etapa de potencia

En el **Capítulo II** se expuso que el control de movimientos de un motor de CD puede realizarse por PWM y por variación de voltaje. El haber preferido el control por medio del microcontrolador LM629 y no del LM628, obedeció directamente al hecho de que la etapa de potencia relacionada con el PWM nos resulto más conveniente. Y fue más conveniente el control PWM porque requirió sólo una fuente de polarización de 12 volts que se derivó de la misma fuente de la computadora anfitriona.

Haber elegido el control por variaciones de voltaje, requería de una etapa de potencia consistente de un amplificador operacional de potencia con dos fuentes de polarización. Tal etapa de hecho se armó, y funcionó correctamente en conjunción con el microcontrolador LM628, pero tuvo que polarizarse con una fuente externa "extra" para la excursión negativa de voltaje[7].

Así, el dispositivo manejador de potencia utilizado finalmente fue el puente H. Se utilizó el manejador de potencia 7I25 que integra 2 puentes H (un canal para cada motor).

III.2.4.1.- Manejador de potencia 7I25: puente H bicanal

La razón principal para utilizar este manejador de potencia, es porque está diseñado ex profeso para actuar directamente con la tarjeta controladora 4i27[8].

[6] Ver hojas de especificaciones en el anexo **Características del sistema**
[7] A pesar de que una fuente de computadora cuenta con voltajes de +/- 12 volts, la polarización de -12 volts entrega un máximo de 0.5 amperes.
[8] Ver hojas de especificaciones en el anexo **Características del sistema**

Fig.II.9.- Tarjeta 7I25: puente H bicanal

Esta tarjeta está diseñada para realizar el manejo de motores de DC por modulación por ancho de pulso, con mediación de una tarjeta de control y la retroalimentación dada por los codificadores de posición. En la tarjeta, se tienen pines de entrada para alimentar la señal en cuadratura de los codificadores.

El 7I25[9] es un excitador de puente H de dos canales con 150 watts por canal, diseñado para ser usado con las tarjetas 6I27, 4I17, 4I43 de *MESA Electronics* ó tarjetas de control de servomotores, ventaja ya señalada anteriormente. El 7I25 cuanta con circuitos integrados de protección térmica que genera alertas de sobretemperatura a 145°, que pueden ser leídas por la tarjeta controladora. Así mismo, está establecida una temperatura de apagado para evitar daño en el puente H en caso de una sobre carga, está temperatura de apagado es de 170°. Además cada canal de salida cuanta con un *LED* que da una rápida indicación de la magnitud y polaridad de la señal presente en dicho canal.

El rango de alimentación de la tarjeta 7I25 es de 12 a 55 Vdc, y no debe exceder de los 60V en ningún instante. Además la tarjeta cuenta con un fusible a 10 A, un disipador integrado con una resistencia térmica de 2° por watt; los interruptores están conectados directamente a un disipador de una resistencia térmica de 2°C por watt.

[9] Ver hojas de especificaciones en el anexo **Características del sistema**

III.2.5.-Computadora Anfitriona

Como ya se había mencionado, la gestión de los recursos de las tarjetas de expansión[10], ha de hacerse a través de una *computadora anfitriona*.

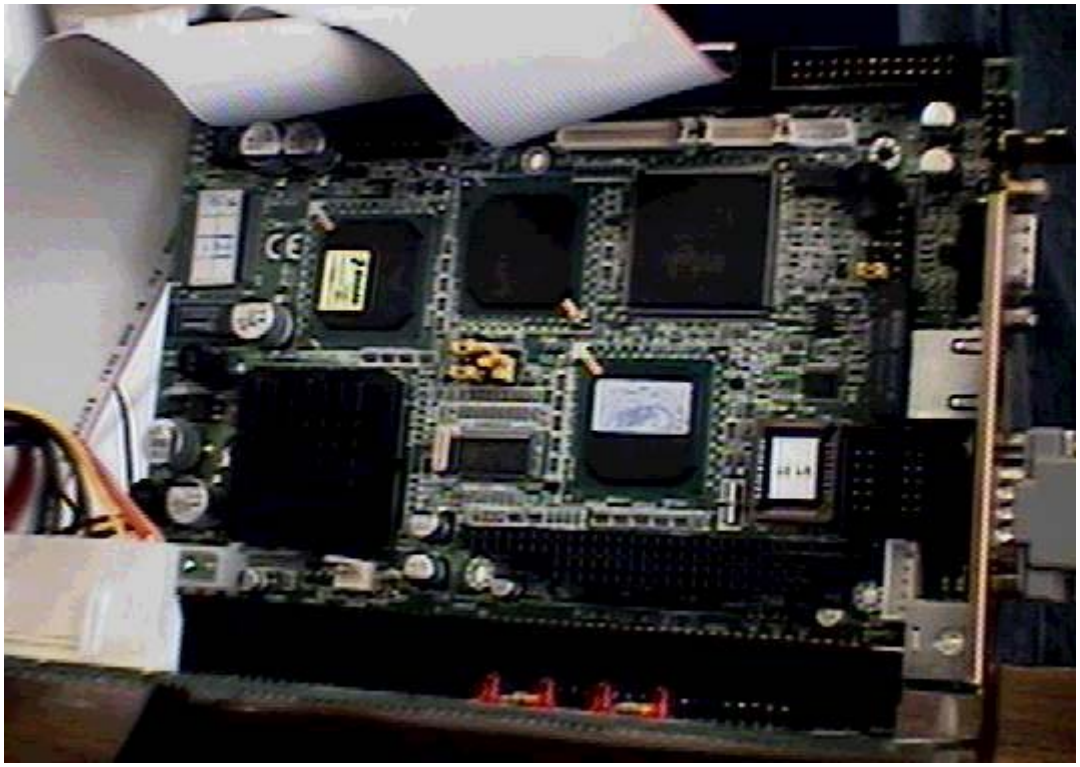


Fig. II.10.- Computadora Advantech

La "computadora anfitriona" es una computadora industrial *Advantech* pequeña y de bajo consumo a 166MHz y con 128 Mbytes en RAM. Como principales características de la computadora se tiene su conector PC/104 y la opción de un temporizador programable tipo *watchdog*.

El conector PC/104 permite la expansión del sistema a través de módulos. Justamente se utiliza el término "computadora anfitriona", porque ésta alberga a través del conector PC/104 a la controladora 4I27 a la que llegarán comandos y datos por medio del bus ISA.

Por su parte, el temporizador *watchdog* ofrece ventajas en tareas de automatización en las que alguna falla desestabiliza o paraliza al sistema. El *watchdog* se inicializa a través de un programa-verificador (escrito por el usuario) que escribe en el puerto I/O 443h el valor máximo en segundos que debe esperar la computadora una "reescritura" a dicho puerto antes de considerar que ha ocurrido un error. Si la computadora entra en estado inestable, el programa-verificador es incapaz de rescribir al puerto 443h a tiempo, y se levanta una

[10] Que son, a saber, la controladora 4I27 que ya se revisó, y la tarjeta convertidora ML16-P que se revisará en la siguiente sección.

bandera que reinicia la computadora o la manda a una rutina de interrupción con el fin de normalizar el funcionamiento

La *computadora anfitriona* cuenta además con un par de puertos seriales, puerto paralelo y 4 ranuras de expansión ISA, en una de las cuales se alberga también la tarjeta convertidora ML16-P que se revisa enseguida

III.2.6.-Tarjeta Convertidora ML16-P

Es bien sabido que al manipular una cámara que incorpora un zoom, conforme el nivel de dicho zoom es más alto, pequeños movimientos de la cámara generan cambios cada vez más significativos en el campo de visión. Dicho de otra forma, conforme aumenta el nivel de zoom de una cámara, sus movimientos deben ser más cortos en amplitud y más suaves en velocidad.

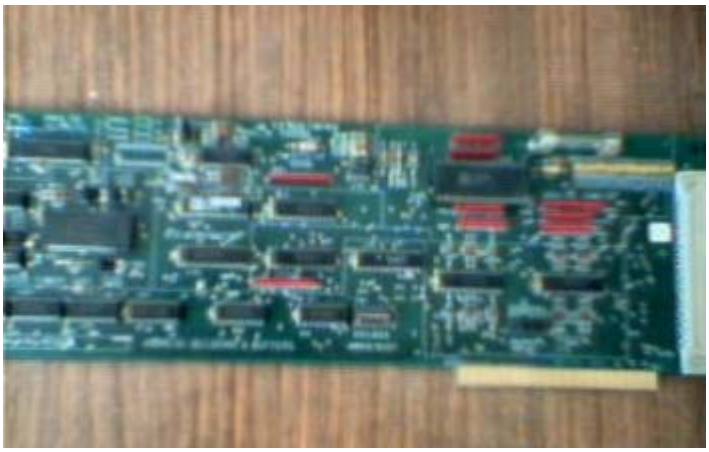


Fig. III.11.- Tarjeta convertidora ML16-P

Por lo anterior, es necesario el conocimiento preciso del valor actual del zoom para así poder adecuar automáticamente la velocidad, aceleración y amplitud de los movimientos del pedestal. .

El zoom motorizado empleado, cuenta con el codificador de posición necesario para conocer su estado, pero dicho codificador es de tipo potenciómetro, por lo que se requirió un convertidor analógico-digital que interpretara el estado analógico de dicho potenciómetro a un formato digital.

El ML16-P es una tarjeta convertidora Analógica- Digital con las siguientes características:

-ocho canales de conversión analógica-digital de 8 bits de resolución
-rangos de conversión de ±128mV, ±5V,0-255mV y 0-10V.
- un máximo de 16,600 conversiones por segundo
-interfaz con su computadora anfitriona por medio de una ranura ISA

Se utilizó un canal de conversión para la señal de error del zoom, con posibilidad para desarrollos posteriores de asignar otro canal de conversión a la señal de error de foco, y así coordinar los valores de zoom y foco. Otro par de canales de la convertidora se usaron para agregar un *joystick* de control[11]

III.3.-SISTEMA DE VIDEO

En este momento hablaremos sobre los elementos relacionados con la adquisición, conversión y despliegue de imágenes, al que hemos definido como el Sistema de video. Tales elementos son: la cámara CCD, el zoom, el *Frame Grabber* y la computadora de visión.

III.3.1.-Cámara CCD

Se utilizó una cámara CCD, que como ya se dijo, se basa en un arreglo bidimensional de sensores sensibles a la luz integrados en un chip.

La cámara usada es del proveedor *The Imaging Source*, modelo DFK 4003/N con las siguientes características principales:

Tabla III.1.- Características principales de la cámara CCD.

Dispositivo de imagen (sensor)	Sony ICOX058AK 1/3" interline CCD
Tamaño de la imagen	768(H) X 494(V)
Tamaño de píxel	6.35(H) X 7.40(V)µm
Señal de salida	NTSC
Disparador electrónico	On/off, 1/60 a 1/30,000 seg
Gamma	0.45
Base del lente	Base CS (base C con adaptador)
Alimentación de energía	7 a 15 VCD
Consumo de energía	Máx. 2.4W
Temperatura de operación	-5° C a +40° C
Dimensiones	52mm X 62mm X 42mm
Peso	160g

[11] Ver en el siguiente capitulo la función RJ que maneja el pedestal por medio de un joystick.

Fig. III.12.- Cámara CCD.

Una característica importante de la cámara CCD es que su disparador electrónico se ajusta automáticamente a las condiciones de luz, variando el tiempo de integración y asemejando así un auto iris que permite obtener imágenes claras (ni pobres ni saturadas) en un amplio rango de condiciones de iluminación.

La cámara entrega una señal de video en formato analógico NTSC[12], ya sea en video compuesto o Y/C[13].

El estándar NTSC establece una resolución horizontal de 525 líneas y una frecuencia de 59.94 campos por segundo; un campo es el conjunto de líneas horizontales pares o de líneas horizontales nones. Los campos pares y nones son desplegados secuencialmente hasta entrelazar el cuadro completo. De esta manera, un cuadro completo es formado por dos campos entrelazados y es desplegado cada 1/30 segundos.

Por tanto, el formato NTSC supone un escaneo trenzado de la imagen. Este escaneo trenzado es sólo una de las incompatibilidades de formato entre el manejo y despliegue de imágenes analógico y el digital, pues este último requiere un escaneo progresivo en el que todo un cuadro se "dibuja" en una sola "pasada". Tal incompatibilidad se resuelve con el uso del *Frame Grabber* que también se revisa en este capitulo.

Como se puede ver en la figura III.12, la cámara no tiene integrado un lente que concentre y haga convergir la luz al chip CCD, pero cuenta con una basa CS al que se adosó el zoom motorizado que se describe en seguida.

[12] Nacional Television System Committee : Comité Nacional de Sistemas de Televisión, establece el estándar de video de televisión al que se apega EUA y Japón

[13] El formato Y/C se refiere a la separación de la imagen en luminiscencia (Y) y color (C), mientras que el formato compuesto, como su nombre lo dice mezcla estas dos propiedades de la imagen en una sola señal.

III.3.2-Zoom Motorizado

El zoom utilizado es de la marca COMPUTAR[14] modelo H10Z1218AMPS. Cuenta con zoom y foco motorizados; distancia focal variable desde 12mm hasta 120mm (10x aprox.); foco de 1.5 a ∞ y capacidad de señal de entrada para auto iris.

Los motores que mueven al zoom y al foco tienen adosados un par de potenciómetros de 5k ohms cuyo voltaje en la línea central varía conforme dichos motores cambian de posición:

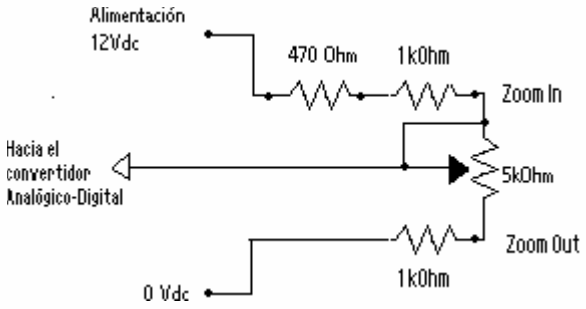


Fig. III.13.- Circuito usado para conocer la posición del zoom.

De está forma, la línea central del potenciómetro entrega una señal analógica proporcional a la posición actual del zoom[15]. Tal señal se alimenta a uno de los canales de conversión de la tarjeta convertidota ML16-P. Cada vez que el zoom es modificado mediante la interfaz "CAM_VIG_1.0", la computadora anfitriona lee el registro de conversión de la ML16-P, le da el formato adecuado y lo envía a la computadora de visión.

La convertidora ML16-P ofrece una conversión digital de 8 bits para una variación del voltaje de entrada analógico de 0 a 10V. Sin embargo, el rango de valores digitales del zoom no es de 256, sino de 200, dado que las caídas de potencial a través de las resistencias que preceden al potenciómetro reducen la excursión de voltaje de entrada a unos 8 volts.

Las características a detalle del iris, foco y zoom se presentan en el anexo **Características del sistema**.

En cuanto a la disposición física del conjunto CCD-Zoom, se optó por asegurar el zoom al pedestal y no la cámara, dado que el zoom es más pasado y grande. Esto se puede observar en la siguiente imagen:

[14] Ver anexo **Características del sistema.**
[15] Aunque la interfaz de usuario "CAM_VIG_1.0" también controla el acercamiento/alejamiento del foco, no se consideró necesario digitalizar la señal de su potenciómetro.

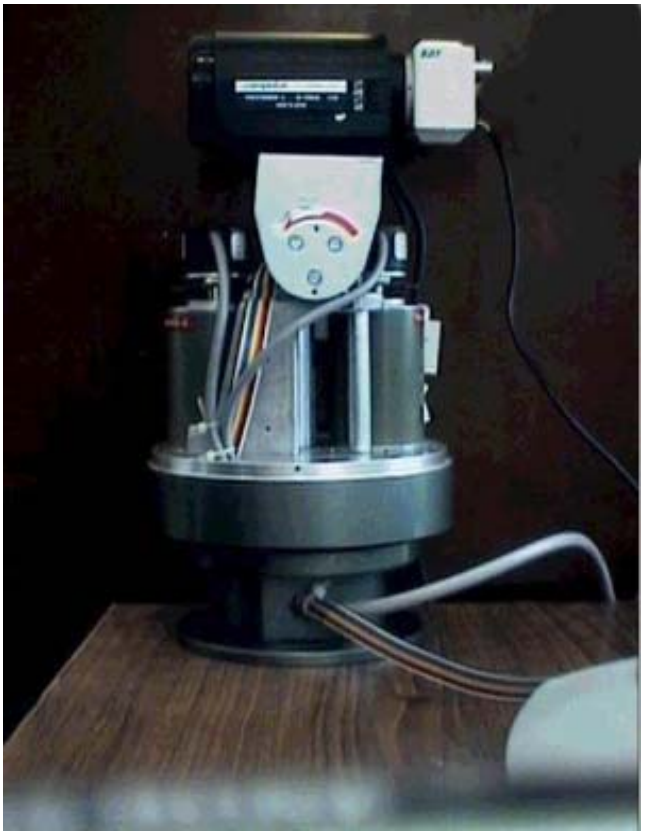


Fig III.14.- Zoom y cámara montados sobre el pedestal.

Los motores de zoom y foco son de 8 V y consumen 50mA aproximadamente, por lo que para moverlos se implementó una pequeña etapa de potencia con un par de puentes H. Se utilizó el circuito integrado L293B, capaz de proporcionar hasta 1A por canal. Este circuito funciona en base a comparadores internos que permiten el *switcheo* de la alimentación Vcc. Cuatro líneas de control determinan qué motor se mueve y en qué sentido, como indica el diagrama siguiente:

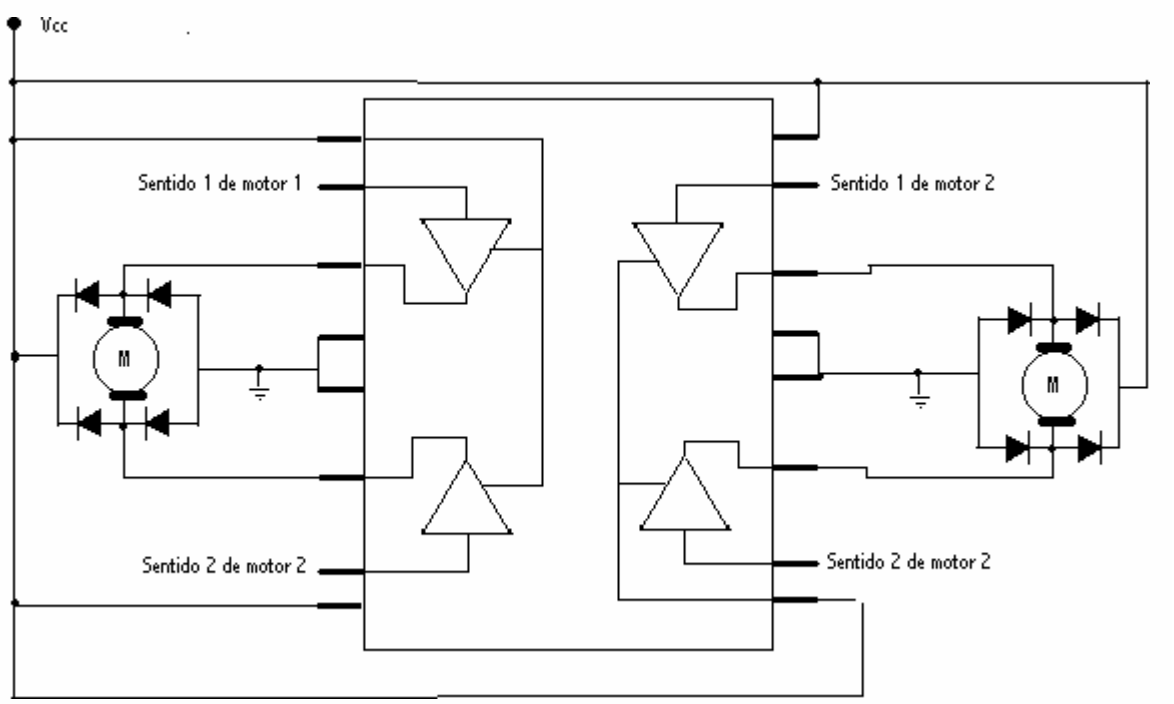


Fig. III.15.-Diagrama de la etapa de potencia (puentes H L293B) para el zoom y foco.

En el diagrama se pueden ver los diodos de seguridad y los de conmutación de corriente. El par de líneas de control Sentido 1 y Sentido 2 determinan en estado del motor correspondiente. Cuando se tiene un uno lógico (5V) en una de ellas y en la otra un cero lógico (0V) el movimiento es en un sentido, y cuando se tiene la combinación contraria, el sentido es inverso. En caso de que las dos entradas tengan el mismo valor lógico ya sea uno o cero, no se presenta movimiento en el motor.

El circuito integrado tiene líneas de habilitación para cada puente y permite tener valores diferentes de alimentación y de uno lógico. Sin embargo para simplificar el alambrado el nivel de uno lógico y el voltaje de alimentación de los motores se igualaron y las líneas de habilitación se dejaron fijas a uno lógico. El acceso a las cuatro líneas de control de zoom y foco motores se realiza a través del puerto paralelo.

III.3.3.-Digitalizador de imagen: *Frame Grabber*

Como ya se mencionó en el capítulo II, el *Frame Grabber* o tarjeta adquisidora de imagen, captura las imágenes en formato analógico y las convierte a formato digital para permitir "manipular" la señal de video mediante una computadora. En términos generales, el *Frame Grabber* muestrea y cuantiza la señal de video analógico y transfiere el paquete de datos resultante a la tarjeta de video o a la memoria principal de la computadora.

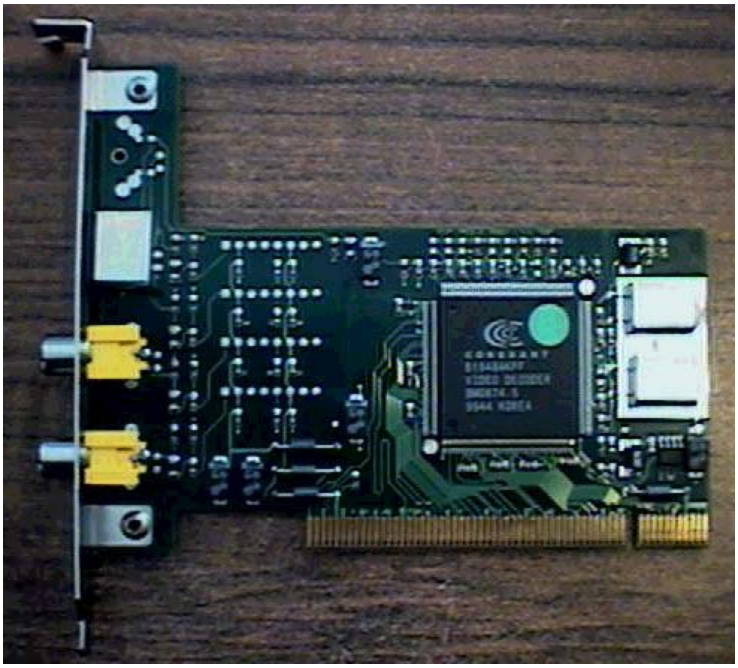


Fig. III.16.- Frame Grabber

El *Frame Grabber* utilizado es del proveedor *The Imaging Source*, modelo DFG/LC1. La siguiente tabla resume sus características más importantes:

Tabla III.2.- Características principales del Frame Grabber DFG/LC1.

Fuentes de video de entrada S-VHS(Y/C) y CSB (video compuesto).
Resolución en formato NTSC de 640x480 píxeles a 60Hz o PAL/CCIR de 768x576 píxeles a 50 Hz
Transferencia de datos a tarjeta VGA o a la memoria del sistema a través del Bus PCI: Formato de salida RGB de 24 o 32 bits para desplegar en monitor (modos *Bitmap* a memoria principal o *DirectDraw* a zona visible/no-visible de la tarjeta VGA) o formatos YUV4:2:2, YUV4:1:1 y Y8 (monocromático) exclusivos para enviar a la memoria principal del sistema y realizar procesamiento digital ulterior.
DLL API de programador (*falcon.dll* para Windows 95, 98 y NT) para acceso al juego completo de funciones para configuración y manejo de las capacidades del *Frame Grabber* (ver Anexo Características del Sistema para un resumen de las funciones principales).

Capa no destructiva[16] : la imagen se digitaliza al área no-visible de la tarjeta VGA, pero sólo se refrescan las zonas de la ventana de salida que contengan un color clave específico. Las demás zonas no son refrescadas, permitiendo esto conservar en la misma ventana de salida imágenes generadas por otra(s) instancia(s).
Filtro de interpolación horizontal y vertical (un paso para resolución vertical, tres pasos para resolución horizontal).
Memoria EEPROM de 64 bytes que pueden servir como identificación de la tarjeta

Tabla III.2.- Características principales del Frame Grabber DFG/LC1 (continuación)

El hecho de que tanto la cámara CCD como el *Frame Grabber* sean del mismo proveedor permitió que los dispositivos, al ser compatibles, operaran en conjunto sin problema.

III.3.4.- Computadora de Visión

El *Frame Grabber* establece los siguientes requerimientos mínimos para la computadora de visión que lo alberga: bus PCI a 33MHz; procesador Pentium a 100MHz; sistema operativo Windows 95 en adelante y tarjeta de video PCI VGA. Tales características son apenas las mínimas para desplegar las imágenes en tiempo real.

Sin embargo como se ha establecido ya, este proyecto busca sentar la base de hardware y software para proyectos ulteriores basados no sólo en la adquisición y despliegue de imágenes, sino en un procesamiento digital arduo, como pueden ser, por ejemplo, el seguimiento automático de blancos y el reconocimiento de patrones. Por ello las características de la computadora de visión deben ser tan holgadas como sea posible (bien sabemos que tratándose de procesamiento digital nunca se tienen suficientes recursos).

Las siguientes son las características de la terminal que se uso como computadora de visión y que pensamos puede ser suficiente para correr adecuadamente un algoritmo de seguimiento automático:

- Procesador Pentium a 700 MHz
- 128 Mbytes en RAM
- Bus PCI a 133 MHz
- 20 Gbytes en disco duro
- Tarjeta de video VGA
- Monitor VGA

[16] *Non-destructive Overlay*: Función disponible sólo si la tarjeta VGA soporta esta característica y su área no visible está integrada a la propia tarjeta.

Las imágenes salvadas[17] ocupan bastante memoria en disco duro por lo cual se recomienda uno de buen tamaño para almacenarlas y se recomienda una unidad de *CD-Writter* para poder liberar la memoria del disco duro.

III.4.-DESCRIPCIÓN GLOBAL DE FUNCIONAMIENTO DEL SISTEMA

A modo de recapitulación la Figura III.16 presenta en forma esquemática la interconexión de los componentes que conforman al sistema de vigilancia y que ya se han descrito.

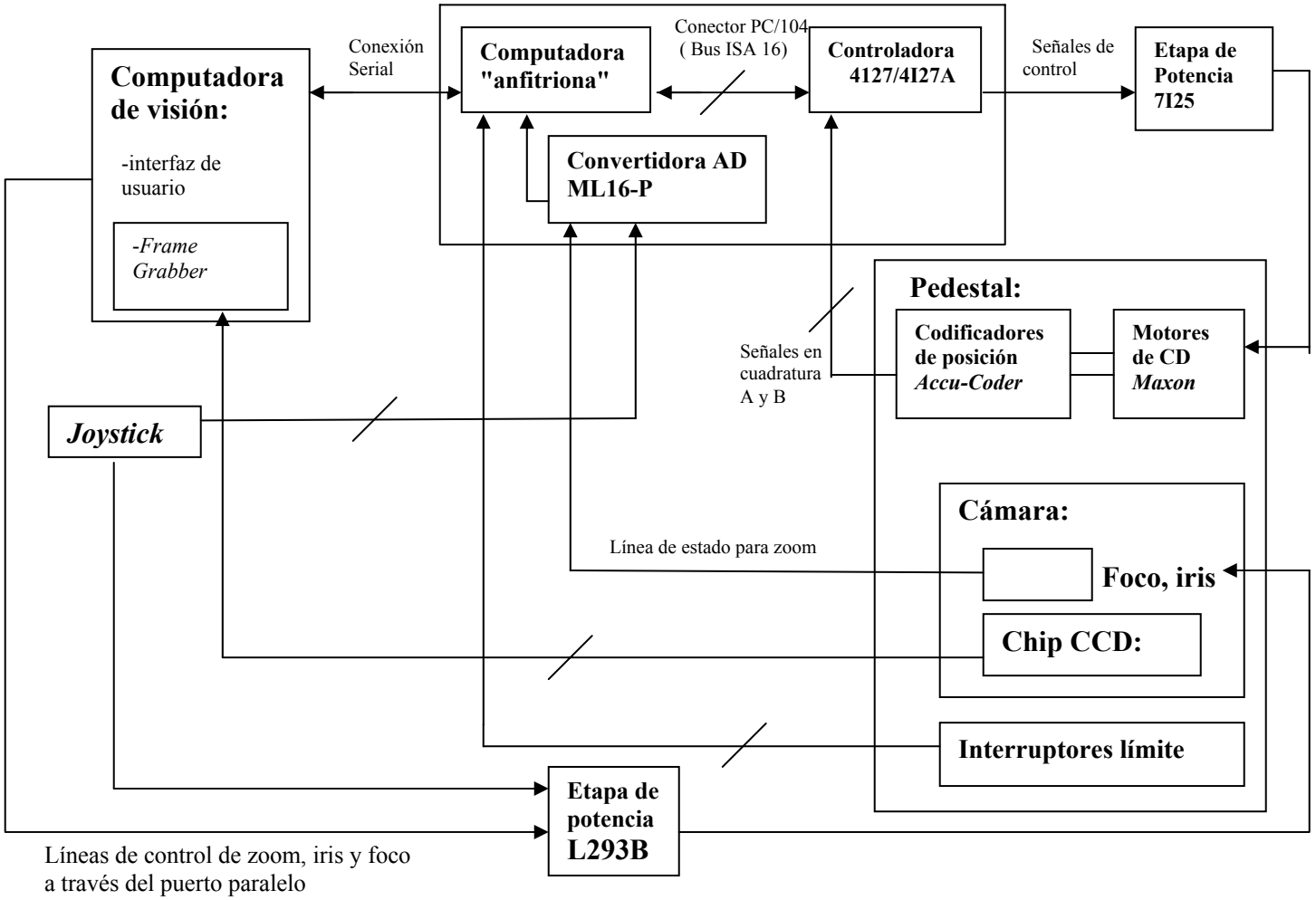


Fig.III.17.- Esquema de interconexión de componentes

A estas alturas de la tesis, estamos en posición de describir el funcionamiento del sistema, pero esta vez (a diferencia de lo que se hizo en el capítulo I) usando nombres y apellidos:

[17] Y eventualmente de las secuencias de video digital capturadas.

Las operaciones realizadas por el operador a través de la *interfaz de usuario*[18] residente en la *computadora de visión*, determinan una serie de comandos que son enviados a la *computadora anfitriona* a través de una *conexión serial*. La *computadora anfitriona* dirige dichos comandos hacia la *tarjeta controladora de movimiento 4I27* a través del bus ISA.

La *controladora 4I27* es el dispositivo que se encarga del manejo de *los motores de DC maxon* que orientan al pedestal. A ella llegan las señales en cuadratura de los codificadores de posición incrementales *Accu-Codder* conectados a los ejes de los motores. En base a los pulsos de los codificadores, los *microcontroladores LM629* en la 4I27 pueden conocer la posición actual de los ejes y realizar los cálculos pertinentes para alcanzar la aceleración, velocidad y posición objetivo requeridos.

La etapa de potencia dada por el *manejador 7I25* se encarga de proveer a los motores de la potencia necesaria para su funcionamiento. Dicho manejador es controlado por la señal de salida de la *controladora 4I27*, siendo tal señal un PWM de 8 bits

La *cámara CCD* montada en el pedestal envía una señal se salida de video analógico hacia la *computadora de visión* a través del *Frame Grabber DFG/LC1* que se encarga de digitalizar dicha señal y hacerla apta para que la *computadora de visión* la maneje; ya sea para almacenarla, enviarla a la tarjeta de video, o manipularla con algún algoritmo de procesamiento digital.

El zoom, y foco de la cámara se controlan digitalmente a través del *puerto paralelo* de la computadora de visión a través de la *interfaz de potencia L293B*. La señal de un *codificador de posición tipo potenciómetro* adosado al zoom, se hace pasar por la *convertidora analógica-digital Ml16-P* a fin de conocer permanentemente el estado del zoom y adecuar los movimientos del pedestal en velocidad y amplitud.

Se adicionó al sistema un *joystick* para orientar la cámara y ajustar los valores de zoom y foco[19]. Las señales analógicas generadas por un par de potenciómetros dentro del *joystick* son digitalizadas por la tarjeta ML16-P a fin de ser interpretadas por la *computadora anfitriona* y que está última sea capaz de enviar los comandos de movimiento adecuados a la tarjeta controladora 4I27.

[18] Interfaz que se revisa en el siguiente capítulo, al igual que los demás programas desarrollados
[19] La Interfaz de Usuario cuenta con un conjunto de componentes suficientemente completo para manejar al sistema, el *joystick* es sólo una opción de control.

CAPÍTULO IV

● COMUNICACIÓN ENTRE DISPOSITIVOS Y PROGRAMAS DE CONTROL

En este capítulo, se revisa la comunicación entre los diversos dispositivos y, a la par que se describen las capacidades del sistema, se presentan pormenores de los programas desarrollados para manejarlo.

IV.1.- PUERTOS Y REGISTROS I/O

En términos generales, la comunicación entre una computadora y los dispositivos que se le conectan, se realiza a través de los llamados registros I/O, ya sea que estén "alambrados" y se tenga acceso a ellos por medio de hardware (por ejemplo los puertos serial y paralelo) o mediante una localidad de memoria accesible a través de un bus como el PCI o ISA.

Como se puede ver en la figura III.17, se tienen dos conexiones entre la computadora anfitriona y el resto del sistema: una con la computadora de visión y otra con la tarjeta controladora 4i27. Con estos dos elementos, la computadora anfitriona mantiene una comunicación bidireccional continua.

La comunicación con la computadora de visión se realiza a través del puerto serie y la comunicación con la controladora se realiza a través del bus ISA. En el **Capítulo I** se trató brevemente a cerca de la comunicación serie, a continuación se dan detalles de las conexiones y protocolos de comunicación implementados

IV.1.1.-Conector PC104 (Bus ISA): Conexión Anfitriona-Controladora 4I27

La *computadora anfitriona* (*Avantech*) cuenta con un conector PC/104 de 50 pines (hembra), que permite la expansión adicionándole diferentes tarjetas. La tarjeta de control 4i27 es compatible con el conector PC/104[1] de 50 pines (macho), permitiendo de esta manera establecer una conexión física de los componentes.

La tarjeta controladora 4i27 requiere de 16 localidades de memoria continuas para sus registros I/O[2]: dirección BASE+00, hasta la BASE+0F en hexadecimal, donde el valor de BASE se configura con *jumpers*. Estos registros comprenden los registros de control y datos (principales y auxiliares) para cada eje, así como puertos de entrada y salida y un par configurable por el usuario.

Así, la tarjeta controladora recibe comandos y datos de la computadora anfitriona (de acuerdo a la tabla de comandos especificada en el *Anexo Características del sistema*) por medio del bus ISA a través de registros I/O.

[1] La especificación PC104 establece protocolos y dimensiones para módulos de expansión compactos principalmente utilizados en aplicaciones industriales
[2] Ver hoja de especificaciones del 4i27. REGISTER MAP

IV.1.2.- Comunicación Serial

En este proyecto, se utiliza transmisión/recepción asíncrona por medio de los puertos COM1 para comunicar a las computadoras de visión y anfitriona. La conexión establecida es una conexión serie bidireccional simple, esto es, se están alambrando sólo 3 líneas de los puertos seriales:

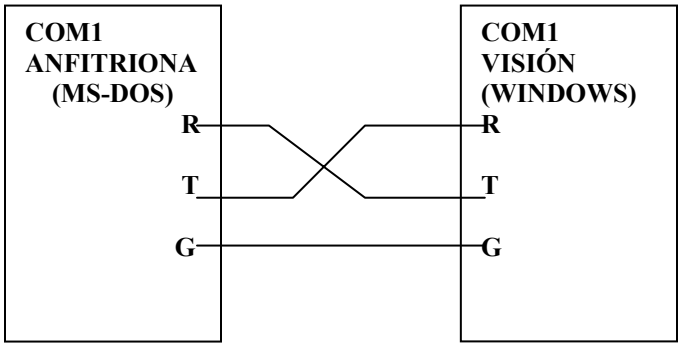


Fig.IV.1.-Conexión serial entre computadoras

Dado que la computadora anfitriona opera con sistema MS-DOS, su puerto serie se manipuló directamente a través de sus registros I/O[3]. Por otro lado, en la computadora de visión (que corre bajo Windows) el puerto serie se operó a través de la HyperTerminal[4] (para propósitos de pruebas) y posteriormente, para la interfaz de usuario, a través de un componente *freeware*[5] desarrollado para Delphi.

IV.2.- PROGRAMA DE COMUNICACIÓN DE LA COMPUTADORA ANFITRIONA CON LA TARJETA CONTROLADORA 4I27.

En base a los protocolos de comunicación y la lista de comandos suministrados por el fabricante del circuito LM628 se desarrolló un programa para establecer la comunicación entre la computadora anfitriona y la tarjeta controladora 4I27, que se detalla a continuación.

IV.2.1- Programa Básico de Comunicación entre la Computadora Anfitriona y la Tarjeta Controladora: "4I27_18.C".

Se realizó un programa básico (**4I27_18.C)** de comunicación entre la computadora "anfitriona" y la controladora 4i27. Este programa fue realizado en el lenguaje de programación "C" e incluye las funciones básicas de control de la tarjeta como son:

[3] Ver anexo "Registros del puerto serie"
[4] Accesorio de Windows para comunicaciones
[5] Se trata del componente ASYNC32 v1.25 de *Varian Software*

reiniciarla; programar los parámetros del filtro PID para el control de movimiento de motores; definir posición de referencia cero; programar parámetros de aceleración, velocidad y posición del movimiento de los motores; detener movimientos; establecer *breakpoits*; consultar registros de status, posición actual de los ejes de los motores, y estado de los puertos paralelos A y B. Este programa está íntimamente ligado con el programa de comunicación serial entre las computadoras.

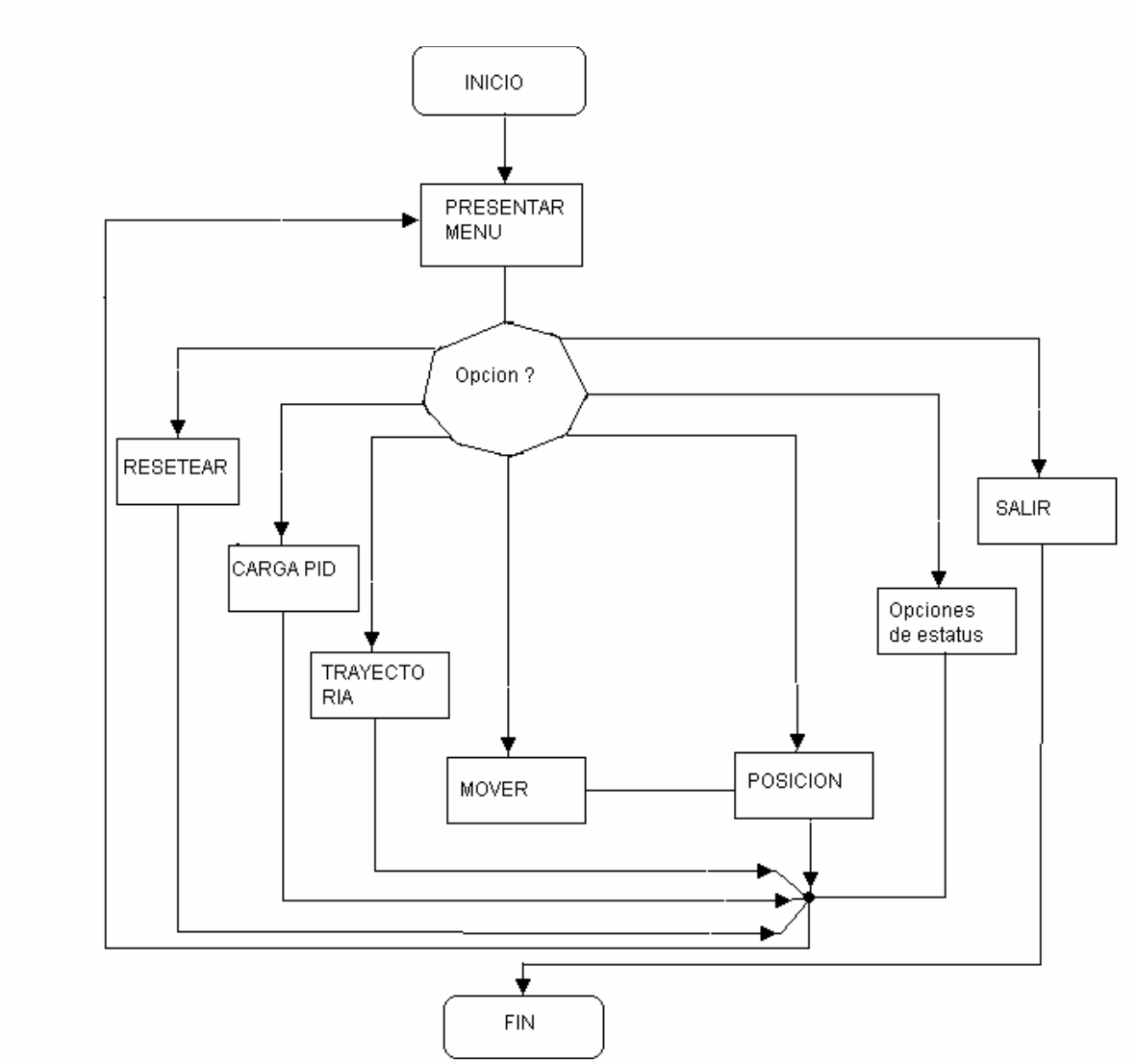


Fig. IV.2.- Diagrama de flujo del programa 4I27_18.C

Este programa se utilizó durante el proyecto para hacer correcciones y pruebas al control de motores, pues se ejecuta y manipula completamente a través de la computadora anfitriona. Dicho programa sirvió de base para el programa final MOTORES.C que se manipula ya por el puerto serie y cuyas características se detallan más adelante.

Se realiza una descripción breve de las opciones del menú de 4I27_18.C[6]:

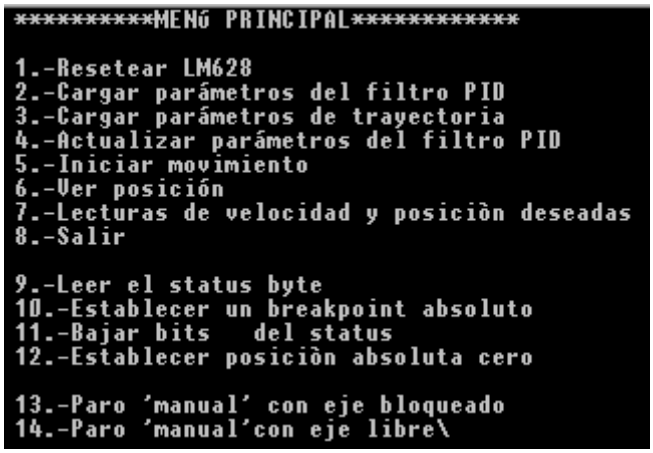


FIG. IV.3.- Menú de entrada del programa 4i27_18.C

1.-Resetear el LM629: reinicia al microcontrolador especificado, llevando todos los registros de trabajo y *buffers* intermedios a cero y definiendo la posición actual del eje del motor se como *Home*. Además se establecen las opciones de operación por defecto[7] .

2.- Cargar parámetros del filtro PID: permite cambiar los valores de cada una de las acciones de control. Después de responder a la pregunta de confirmación de cambios, se presentan los valores posibles que pueden tomar los parámetro y el usuario escogerá el valor adecuado. Los parámetros son las constantes proporcional, integral y derivativa del filtro. Así mismo se solicita el tiempo de muestreo derivativo Td que es un múltiplo del tiempo de muestreo Ts e indica cada cuanto se actualizara el término derivativo.

3.-Cargar parámetros de trayectoria: opción que opera de forma similar a la anterior variar los parámetros de aceleración, velocidad y posición.

4.-Actualizar parámetros del filtro PID: La opción dos carga los valores ingresados en un buffer del microcontrolador LM629, está opción transfiere dichos valores a los registros de trabajo con los que se realizan los cálculos de trayectoria.

5.-Iniciar movimiento: esta opción es usada una vez que se tienen cargados los parámetros del filtro y de trayectoria, y es la orden para comenzar el movimiento del motor. Inmediatamente después que se ha seleccionado esta opción, se visualiza una pantalla que despliega la posición actual (en pulsos de codificador) del eje del motor seleccionado.

6.-Ver posición: opción para conocer la posición absoluta en pulsos de codificador del eje actual.

[6] Ver código fuente en el **anexo de programación**
[7] Consultar las hojas de especificación del LM629 en el "Anexo de **características del sistema**"

7.-Lecturas de velocidad y posición deseadas: el controlador LM628 genera dos valores instantáneos de velocidad y posición deseadas que operan como valores de referencia o *setpoint* dentro del lazo de control. Con este comando se accede a dichos valores que se encuentran en los propios registros de trabajo.

8,.-Salir : opción para abandonar el programa.

9.-Lee status byte: opción que regresa un valor hexadecimal, que se interpreta de acuerdo a las notas de aplicación del LM628, incluidas en el *ANEXO1*.

10.- Establecer *breakpoint* absoluto: permite establecer un valor absoluto de pulsos que al ser alcanzado por el eje actual levantará una de las banderas del byte de status.

11.-Bajar bits del status: permite bajar todas las banderas enmascarables del byte de estatus.

12.-Establecer posición absoluta cero: establece la posición actual del motor como el valor de referencia cero.

Las opciones 13 y 14 sirven para detener el motor actual, la diferencia entre ellas es que una deja el eje del motor sin libertad de movimiento (lo "amarra" a la posición de paro) y la otra sí permite que se pueda mover manualmente.

IV.3.-PROGRAMA DE COMUNICACIÓN SERIAL ENTRE LA COMPUTADORA ANFITRIONA Y LA DE VISIÓN: "MOTORES.C".

Para establecer la comunicación serial entre las computadoras de visión y anfitriona, se desarrolló el programa "**MOTORES.C**[8]" (residente en la anfitriona) que cuenta con todas las capacidades del programa 4I27_18.C recién descrito. MOTORES.C realiza funciones de movimiento más complejas, las cuales fueron programadas utilizando los comandos base de 4I27_18.C .Las adiciones son las siguientes:

- responde a comandos enviados por la computadora de visión a través del puerto serial

-lee interruptores adosados al pedestal para establecer ceros absolutos en ronza y en elevación y poder así realizar una rutina de inicialización.

-opción de controlar el movimiento del pedestal por medio de un *joystick*

-a través del puerto serie envía a la computadora de visión la posición actual de los ejes de ronza y elevación, así como la posición del zoom. Esta última es conocida al acceder al

[8] Ver **Anexo de programación**

registro de conversión de un Convertidor Analógico Digital[9] al que llega como entrada la señal de un potenciómetro integrado al zoom[10].

IV.3.1.-Características del Programa "MOTORES.C"

Como ya mencionó, la comunicación serial entre las computadoras es asíncrona bidireccional simple y sin *Handshaking* pues no se considera necesario, dado que el mismo programa de recepción de comandos efectuará la labor de validar la sintaxis de los mismos.

Por tanto las primeras líneas del programa están destinadas a la configuración de la comunicación serial asíncrona: 8 bits de datos; velocidad de 19,200; 1 bit de paro; ningún control de flujo.

Una vez establecida la comunicación serial, se tiene control de los motores desde la computadora de visión. Para facilitar la comunicación y reducir la posibilidad de errores, se crearon comandos de sintaxis sencilla que se explican más adelante. Dichos comandos, pueden realizar las mismas acciones que el programa "4i27_18.C" y algunas más como ya se mencionó.

Tras configurar la comunicación, "MOTORES.C" despliega en el monitor de la anfitriona el carácter R .Así mismo se desplegarán todos los comandos recibidos de la computadora de visión. Cabe señalar que este despliegue de información tiene principalmente fines de depuración, pues la computadora anfitriona bien puede operar sin un monitor dedicado.

En seguida, se detallará la función de cada comando, su sintaxis y un ejemplo. En dicha sintaxis se define 'm', como un valor entre 0,1 y 2[11], que determina el motor al cual se esta haciendo referencia; 0 indica que se accederá a los registros de control del motor0;1 indica que se accederá a los registros de control del motor1 y 2 indica que se cargará la misma información a los registros de control de ambos motores. Se define 'x' como un número dígito y 's' como el signo + o -.

- **RRm**. Instrucción de Reiniciar (R). Se puede aplicar a cualquiera de los dos motores o a los dos simultáneamente. Envía un mensaje a la pantalla de haber realizado la acción.

  Ejemplo:
  RR2 Reinicia los dos microcontroladores

- **RFmxxxPxxIxxDxx**. Instrucción cargar Filtro (F,P,I,D). Se puede aplicar a cualquiera de los dos motores o a los dos simultáneamente. Los primeros tres dígitos determinan el tiempo de muestreo derivativo Td, los dos dígitos que

[9] Este convertidor esta integrado en la tarjeta MPL-16 apilada en el bus ISA de la anfitriona.
[10] Detalles en la sección "Zoom motorizado" del **Capítulo III**
[11] El motor 0 da el movimiento de ronza, y el motor 1 el movimiento de elevación.

siguen a las letras P, I, D, son los valores de los términos proporcional, integral y derivativo respectivamente para cada una de las acciones del filtro de control .

Ejemplo

RF0001P11I07D04 Carga los parámetros del filtro con los valores 11 para acción proporcional, 7 para acción integral, 4 para acción derivativa

- **RTmAxx.xxxVxxxPsxxxxxxxx**[12]. Instrucción cargar Trayectoria (P,A,V,P). Se puede aplicar a cualquiera de los dos motores o a los dos simultáneamente. La aceleración esta dividida en parte entera y parte fraccionaria, tal como lo indican los dígitos que suceden a la letra A. Una vez que ya se han cargado los valores de aceleración y velocidad, se puede cambiar la posición de la siguiente manera:

  RTPsxxxxxxxx

Ejemplos:

RT1A00.100V14P+10000000 Carga parámetros de aceleración, velocidad y posición

RTP-200000 Carga únicamente la posición.

- **RV** Instrucción Ver posición (V). Con esta instrucción se podrá ver el punto en el que se encuentran los ejes de los motores. Esta instrucción espera cualquier carácter para regresar al estado de listo (R_)

- **RMm** Instrucción de iniciar Movimiento (M). Se puede aplicar a cualquiera de los dos motores o a los dos simultáneamente. Comienza el movimiento del o los motores indicados, y pasa directamente a la pantalla de ver posición, para visualizar la posición en cada instante. Para regresar al estado de listo (R_) es necesario enviar cualquier carácter.

-

Ejemplo:

RM2 Mueve a los dos motores de acuerdo los parámetros de trayectoria previamente cargados.

- **RHm** Instrucción de establecer referencia cero o *Home* (H). Se puede aplicar a cualquiera de los dos motores o a los dos simultáneamente. Esta instrucción establece como punto cero el lugar actual donde se encuentren los ejes de los motores. **Ejemplo:**

RH1 Pone en cero el contador de posición del motor uno.

- **RPm** Instrucción de Paro (P). Se puede aplicar a cualquiera de los dos motores o a los dos simultáneamente. Esta instrucción detiene los motores en

[12] Las unidades de Aceleración son en pulsos_de_codificador/tiempo_de_muestreo² y puede ser una fracción
Las unidades de Velocidad son pulsos_de_codificador/tiempo_de_muestreo valor siempre entero
Las unidades de posición son en pulsos_de_codificador valor siempre entero

movimiento, y "amarra" los ejes del motor en la posición en que se halla recibido la orden de paro.

Ejemplo

RP0 Detiene el motor cero.

- **RS** Instrucción Salir (S). Esta instrucción es para salir del programa.

- **RI** Inicializar. Comienza una secuencia, en la que se ordena a la controladora operar ambos motores en modo de velocidad de forma que sigan en movimiento hasta encontrar los límites (interruptores)"inferiores" en ronza y elevación. *(Ver Fig. IV.3.- Interruptores límite de elevación)*

Una vez que un motor encuentra su límite inferior, se detiene y "espera" a que el otro motor encuentre su límite inferior. Cuando los dos límites inferiores se han alcanzado se reinicia el movimiento de ambos motores en modo de velocidad en el sentido opuesto, a fin de buscar los límites (interruptores) superiores. Cuando este segundo par de interruptores ha sido alcanzado, se conoce ya el recorrido total (en pulsos de codificador) en ronza y elevación y se ordena que los motores, operando ya en modo de posición, vayan a la mitad de su respectivo recorrido.

Ejemplo:

RI Los motores comenzarán con la secuencia de inicialización, llegando a los interruptores límite, y encontrando el punto que servirá de referencia, 60° en elevación y 180° en ronza

Fig. IV.4.- Interruptores límite de elevación.

-**RZ** Enviar posición del Zoom(Z). Indica a la computadora anfitriona leer el estado del zoom y enviarlo a la computadora de visión. La lectura de la posición del zoom es conocida al acceder al registro de conversión de un Conversor Analógico Digital a uno de cuyos canales se tiene conectada la señal de voltaje analógico de un potenciómetro adosado al zoom. Este comando se ejecuta automáticamente en cuanto el zoom cambia de posición, por lo tanto la computadora de visión siempre tiene conocimiento del estado real del zoom.

-**RJ** Comando de operar el pedestal por medio de un *joystick* .

Otra característica importante del programa "MOTORES.C" es la de enviar a la computadora de visión la posición actual de los ejes de ronza y elevación. Dicha posición se refresca continuamente mientras alguno de los motores este en movimiento, por lo tanto, igual que con el zoom, la computadora de visión siempre conoce la posición real de los ejes de ronza y elevación.

A continuación se presenta la interfaz de usuario, residente en la computadora de visión, con la cual interactúa el programa "MOTORES.C".

IV.4.- INTERFAZ DE USUARIO

El desarrollo del software para el proyecto, requirió de la implementación, en la computadora de visión, de una interfaz de usuario que permitiera el control del sistema de una manera intuitiva, fácil y a la vez completa. Esto es, dicha interfaz integra los elementos necesarios para acceder a todas las capacidades del sistema en dos niveles: el usuario neófito puede operar fácilmente el sistema con sólo un *clic* de ratón; en tanto el desarrollador cuenta con opciones para efectuar tareas más elaboradas a fin de verificar el funcionamiento óptimo del sistema.

La interfaz mencionada se realizó con Delphi5 usando las paletas y componentes que incluye la instalación típica, a excepción del componente para establecer la comunicación serial y el componente para acceder al *Frame Grabber* y poder desplegar la imagen de la cámara. El nombre del proyecto Delphi es "**CAM_VIG_1.0**" [13]

Y como una imagen dice más que mil palabras, se presenta a continuación la ventana de la interfaz de usuario y se explica en el siguiente apartado la función de cada uno de sus bloques y componentes.

[13] Con la respectiva pléyade de archivos y códigos que genera un proyecto en Delphi, incluyendo el ejecutable. Ver **Anexo de programación**.

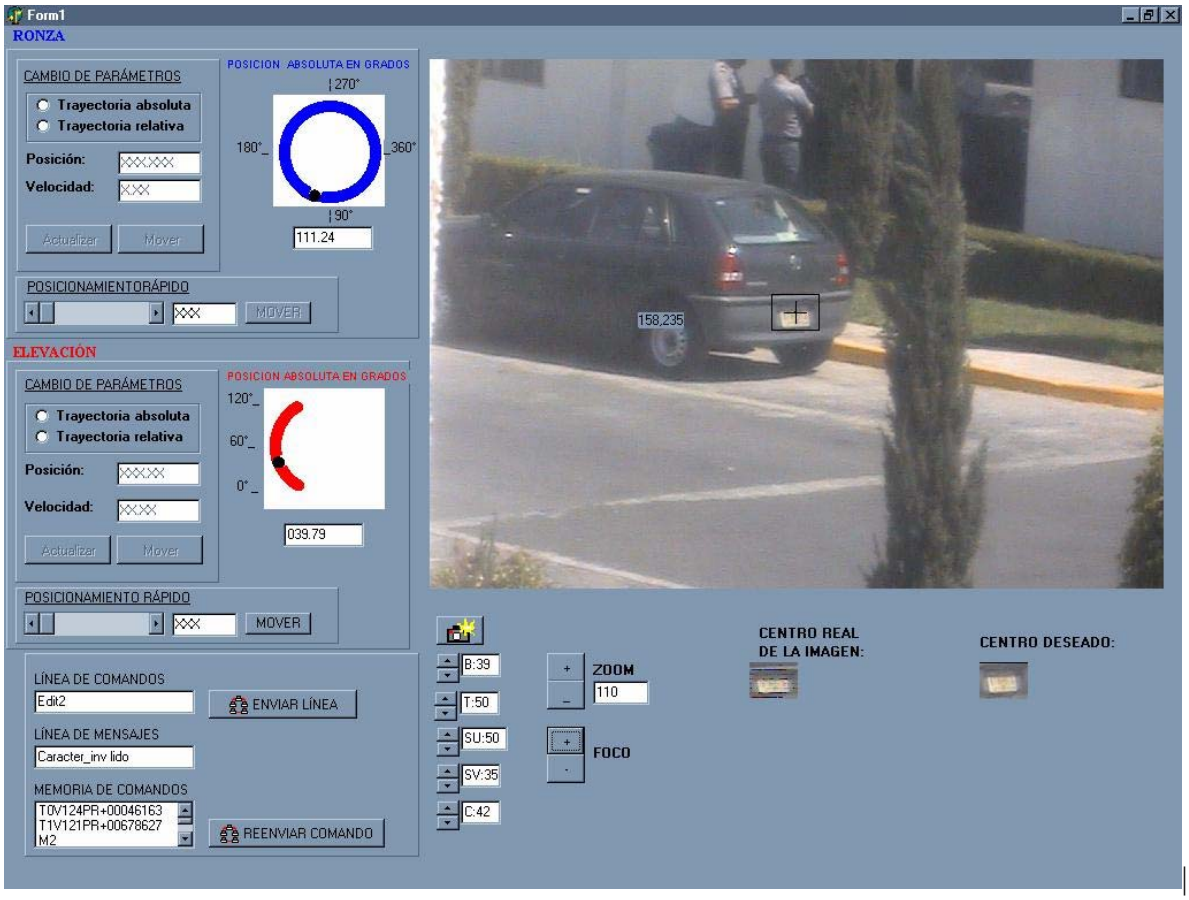


Fig.IV.4.-Ventana de Interfaz de Usuario: bloques y componentes de bloque

IV.4.1.-Interfaz de Usuario: bloques y componentes de bloque

Como se puede ver en la figura anterior, la distribución de componentes se divide en varios bloques:

- BLOQUE DE RONZA: Este permite el posicionamiento "manual"[14] del pedestal en el eje de ronza. Presenta opciones de movimiento absoluto y relativo, posicionamiento rápido, y visualizador gráfico y numérico de la posición actual de ronza.

[14] Se utiliza el término "manual" para diferenciar este tipo de posicionamiento del *posicionamiento automático de un sólo clic* que se describe más adelante

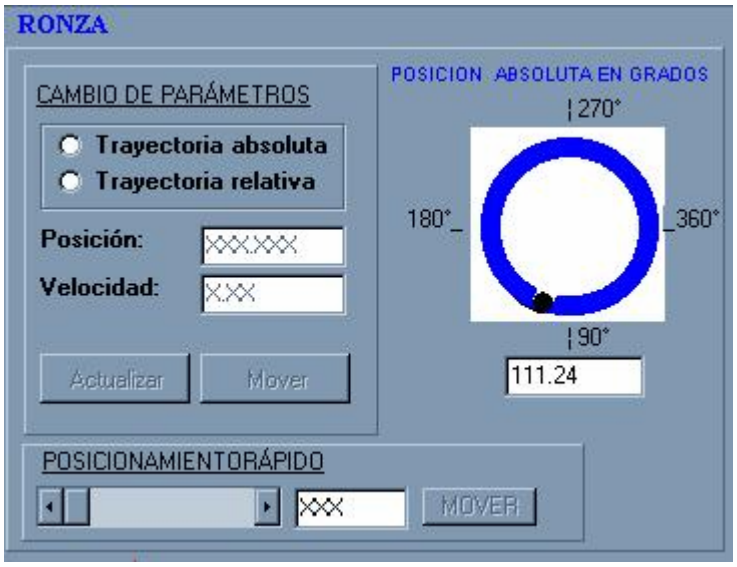


Fig.IV.5.-Bloque de control de ronza

Las unidades de posición de trayectoria absoluta y relativa se dan en grados de ronza[15]. El programa realiza una comprobación de rangos, de forma que no se puede realizar movimientos que estén fuera del rango de 0 a 358° absolutos (los 2 grados que faltan para completar 360° se dieron como "colchón" para compensar el recorrido que se pierde por los interruptores de ronza). La velocidad, se ingresa en grados de ronza por segundo, permitiéndose un valor máximo de 7.5°/seg.[16]. El programa despliega mensajes de error si un comando pretende rebasar los rangos de posiciones absolutas.

El botón "Actualizar" envía los parámetros de trayectoria deseada a la controladora 4I27 a través de la computadora anfitriona, pero el movimiento se inicia hasta pulsar el botón "Mover".

La barra de deslizamiento en el fondo de nombre "*POSICIONAMIENTO RÁPIDO*" permite establecer rápidamente una trayectoria absoluta en ronza. El botón "MOVER" inicia la trayectoria elegida con la barra.

- BLOQUE DE ELEVACIÓN: Este permite el posicionamiento "manual"[17] del pedestal en el eje de elevación. Presenta opciones de movimiento absoluto y relativo, posicionamiento rápido, y visualizador gráfico y numérico de la posición actual de elevación.

[15] Los grados que manejan ronza y elevación son grados geométricos normales
[16] Esta restricción la estableció el voltaje de alimentación del puente H.
[17] Igual que en la nota 13

Fig.IV.6.-Bloque de control de elevación

Aplican las mismas anotaciones que para el bloque de ronza, con la salvedad de que el rango de posición en elevación es de 0 a 120° y la velocidad máxima es de 4.5°/seg.

-BLOQUE DE EDICIÓN DE COMANDOS: Debajo del bloque de elevación se tiene un panel con los componentes necesarios para editar y reenviar los comandos anteriores o generar y enviar nuevos comandos. También se tiene un campo donde se despliegan mensajes de texto enviados por la anfitriona.

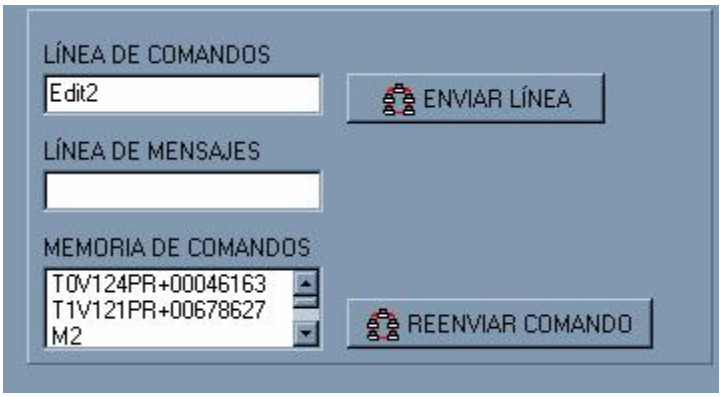


Fig.IV.7.-Bloque de edición de comandos

La memoria de comandos almacena TODOS los comandos enviados a la anfitriona, ya sea que se hayan generado automáticamente o que hayan sido editados manualmente. La línea de mensajes presenta información sobre el estado de inicialización del pedestal y mensajes de error. Los mensajes de error sólo son generados por los comandos editados manualmente (el factor humano), pues los construidos automáticamente tienen una sintaxis invariablemente correcta.

Un doble clic sobre un elemento de la memoria de comandos lo carga a la línea de comandos donde puede ser editado.

-BLOQUE DE IMAGEN: Presenta los componentes con que se manipula la sección de video del sistema, a saber:

- *Lienzo*: En esta sección de la pantalla se despliega (a una resolución de 620x480 pixeles) en tiempo real la imagen capturada por la cámara CCD y convertida por el F*rame Grabber* .Un clic de ratón sobre el lienzo posiciona automáticamente la cámara, a fin de que la zona *clickeada* quede centrada. El algoritmo que hace este posicionamiento automático coordina el valor actual de zoom con la diferencia de píxeles, en ronza y elevación, entre el centro deseado y el actual, para así generar automáticamente las trayectorias (en posición y velocidad) adecuadas[18].

  El lienzo también despliega la posición actual (x,y) del cursor. La parte de imagen en la mira rectangular del centro define el recuadro "*Centro real de la imagen"*

Fig.IV.8.-Lienzo

- *Ajustes de imagen* Presenta una serie de controles para ajustar el brillo, contraste, tinte y saturación de la imagen desplegada.

- *Control de zoom y foco* Modifican el valor del foco y zoom del juego de lentes acoplado a la cámara CCD. Así mismo se refresca el registro de posición del

[18] Más sobre este algoritmo en la sección *Posicionamiento automático con un "clic"* de este capítulo.

zoom, pues es vital conocer su posición exacta para el algoritmo de *posicionamiento automático*.

- *Icono de guardar imagen* Al presionar este botón se archiva en formato BMP la imagen actual del lienzo

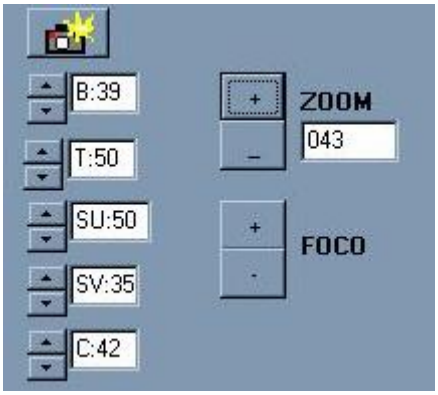


Fig.IV.9 Bloque de ajuste de imagen e icono de guardar imagen

- *Ventanas de comparación* Este par de pequeños *lienzos "Centro deseado de la imagen"* y *"Centro real de la imagen"* al ser comparados permiten evaluar visualmente el desempeño del algoritmo de posicionamiento automático

Fig.IV.9 Ventanas de comparación

IV.5.-POSICIONAMIENTO AUTOMÁTICO CON UN "CLIC"

Como se mencionó en la sección "Interfaz de Usuario", el sistema tiene la capacidad de reorientar automáticamente la cámara, de modo que la región del lienzo sobre la que se da un "clic" quede centrada en la imagen del lienzo. A continuación se revisa a detalle tal característica, así como los principios en que se basa el algoritmo, que es, a decir verdad, bastante elemental, pero permite apreciar las capacidades del sistema en coordinación con el software adecuado.

IV.5.1.-Ejemplo de operación del *Posicionamiento Automático con un Clic*

Se da un "clic" en la llanta trasera de un coche blanco**....**

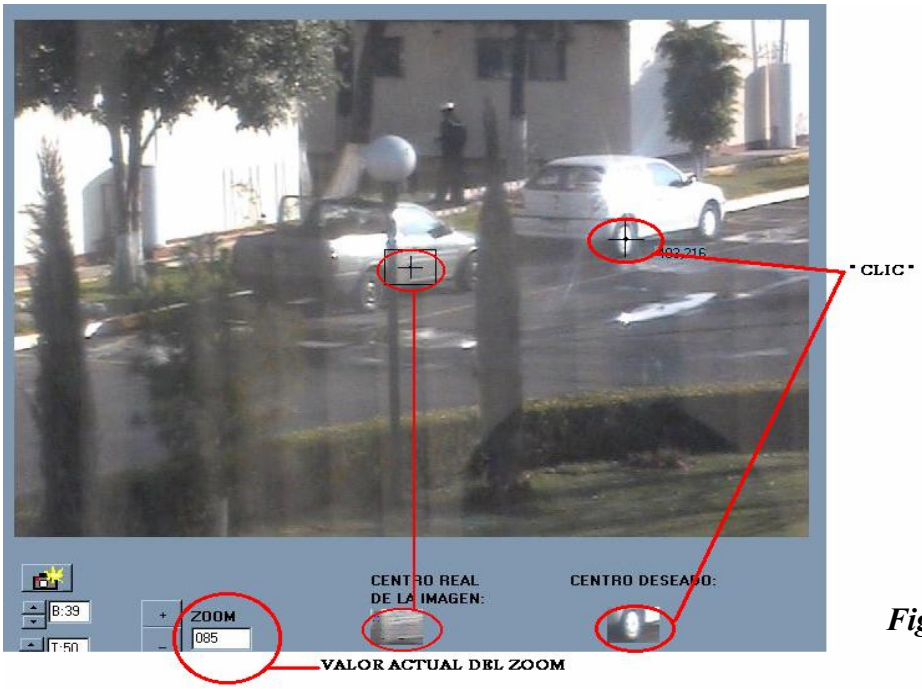


***Fig.*IV.10**

....el posicionamiento automático reorienta la cámara para que la región "cliqueada" vaya al centro**...**

***Fig.*IV.11**

...se puede evaluar visualmente el desempeño del posicionamiento automático al comparar el recuadro *Centro Deseado* (imagen estática capturada al momento del clic) con el recuadro *Centro Real De La Imagen* .Cabe notar que las imágenes de estos dos recuadros

no pueden ser 100% idénticas[19], pues el recuadro *Centro Deseado* corresponde a una perspectiva que se pierde cuando la cámara se reorienta y el recuadro *Centro Real* corresponde a la nueva perspectiva de reorientación

IV.5.2.-Principios de operación del Posicionamiento Automático con un Clic.

El principio del posicionamiento automático es muy simple, y consiste en obtener la correspondencia entre el recorrido en píxeles apreciado en el lienzo de la ventana de usuario, con el recorrido en grados en ronza y elevación en función del valor de zoom. Antes de explicar la forma en que se obtuvo dicha correspondencia, se establecerán las siguientes consideraciones:

1.-La imagen desplegada en el lienzo de la interfaz de usuario, puede considerarse como la proyección en un plano de la imagen acombada capturada por la lente del zoom[20]. Esto tiene como efecto que los píxeles periféricos de la imagen representan más distancia real que los píxeles del centro, pues son la proyección de arcos mayores que los del centro. Dicho de otra manera, la imagen está "más apretada" en las zonas más alejadas del centro.

2.-A menor zoom el campo de visión se "abre", y los píxeles representan distancias reales más grandes; por el contrario, a mayor zoom, el campo se "cierra" y los píxeles representan distancias reales menores.

Por ejemplo, dado que el zoom utilizado es de 12mm 120mm, podemos establecer que, de manera aproximada, un objeto que aparenta una longitud de 20 píxeles con el zoom totalmente retraído, aparentará una longitud de 200 píxeles con el zoom totalmente desplegado.

Estas consideraciones se resumen en la figura siguiente:

[19] A menos que se haya dado clic sobre el centro mismo del lienzo, caso en el que las imágenes serían idénticas, pero debido al hecho de que el pedestal no se habría movido.

[20] O en términos llanos: los lentes del zoom son curvos y el sensor CCD es plano.

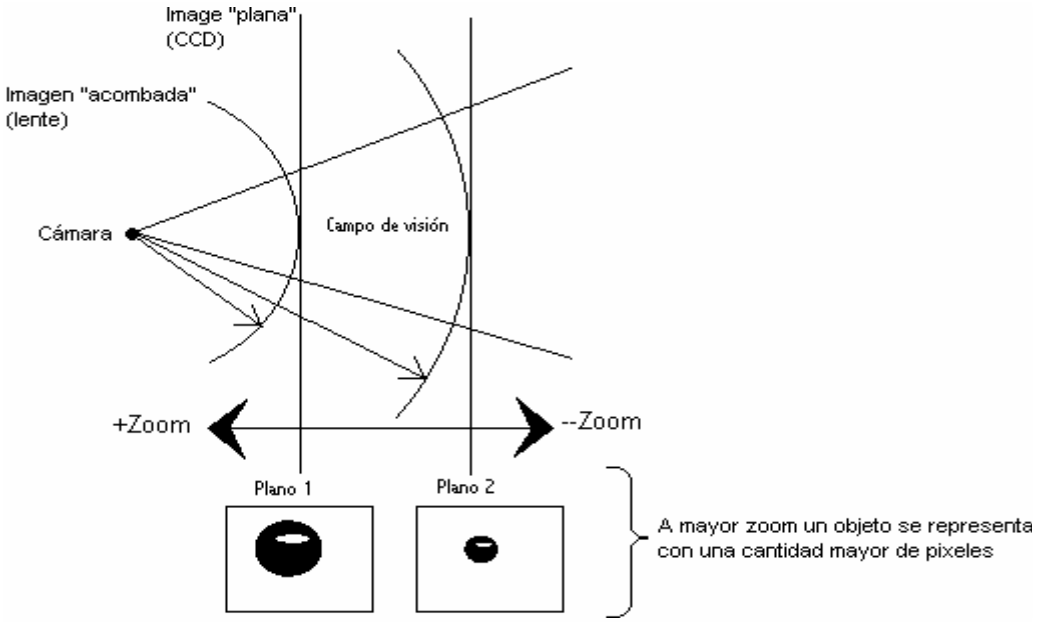


Fig.IV.13.-Consideraciones de posicionamiento automático

De las consideraciones anteriores (que fueron las que se contemplaron) la más importante es la segunda: la de encontrar las correspondencias Píxel/Grado_Ronza y Píxel/Grado_Elevación respecto al valor de zoom. Dichas correspondencias se caracterizaron de forma experimental siguiendo el siguiente procedimiento:

1.-Se lleva el zoom a una posición fija Z_i.

2.-Se establece un punto dentro del campo visual como referencia. Este punto referencial correspondió al centro de la imagen.

3.-Se realizan movimientos de amplitud conocida (medida en grados) en ronza y elevación.

4.-Se mide (con ayuda del cursor de posición del lienzo de usuario) el recorrido en píxeles que experimenta el punto de referencia.

5.-Se obtienen la relación Píxel/Grado_Ronza y Píxel/Grado_Elevación

(Los pasos 3 4 y 5 se repiten tantas veces como se crea conveniente, variando en cada iteración el valor de la amplitud del movimiento)

6.- Se promedian todos los valores Píxel/Grado_Ronza y todos los valores Pixerl/Grado_Elevación, para obtener sólo dos valores finales que caractericen el valor de zoom en la posición Zi

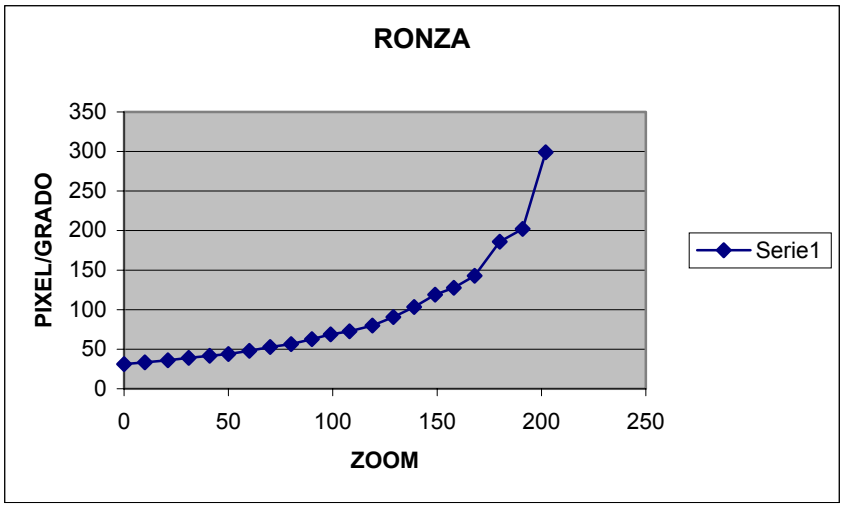
7.- $i = i + 1$

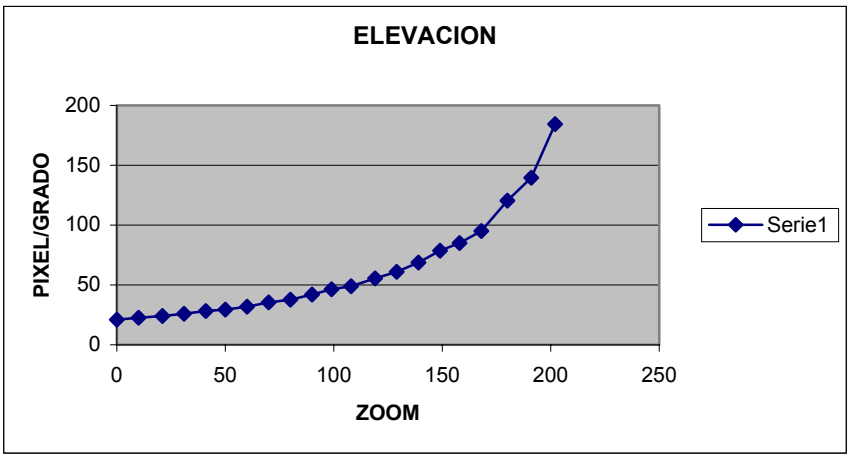
Así, siguiendo el procedimiento ya descrito, se caracterizó el zoom del 0 al 100% en pasos de 5%, esto es, para 20 posiciones distintas y 5 recorridos por posición por eje (200 mediciones en total).

Las posiciones intermedias de zoom se ajustaron con interpolaciones lineales.

Dado que el procedimiento involucró promedios, y dado que el algoritmo tiene un buen desempeño, la primera consideración del "acombamiento" de la imagen no se caracterizó, pero se hará en trabajos posteriores para afinar más el reposicionamiento automático.

Se presentan las gráficas de resultados obtenidas para la caracterización[21]:





[21] Las unidades de ZOOM en las gráficas se dan directamente en el número de cuentas (200 en total) que entregaba el convertidor analógico digital (Ver sección "Zoom Motorizado **Capítulo III**"). Para convertir a porcentaje sólo se divide entre 2.

Con esto no queda presentar en el siguiente capítulo, las conclusiones del trabajo llevado a cabo.

CAPÍTULO V

- ***CONCLUSIONES***

V.1.- CONCLUSIONES

Visualizamos este proyecto como una tesis robusta en cuanto a temas de ingeniería se trata, con varios aspectos involucrados en su desarrollo y consideramos que se ha llegado a formar un sistema bien fundamentado y flexible para aplicaciones posteriores. El trabajo que se ha realizado ha sido el de darle forma a un sistema completo de vigilancia a partir de los componentes aislados que se tenían inicialmente; haciéndolos trabajar de manera conjunta y creando un software que nos permita manejarlo de manera sencilla.

Se logró implementar un sistema de video-vigilancia que permite movimientos de una cámara en dos ejes, realizando así dicha vigilancia en un hemisferio de visión. Hemisferio definido por dos rangos en los movimientos de la cámara: 358 grados en el horizonte o ronza y 120 grados en elevación

Las velocidad de posicionamiento máximas resultaron de 7.5°/seg en ronza y 4.5°/seg en elevación, pero esta limitación se puede salvar fácilmente aumentando la capacidad de voltaje de la fuente de alimentación.

En cuanto al control de los movimientos del pedestal, se tiene que el error de posicionamiento máximo es del orden de +/-0.02 grados, el cual se presenta a las máximas velocidades de posicionamiento y para recorridos del orden de grados. La precisión anterior, es, considerando la aplicación directa del sistema, suficiente para un buen desempeño, pues aún cuando se tiene aplicado todo el zoom (que es cuando los movimientos deben ser más finos), los movimientos en ronza y elevación significativos son del orden de decenas de grado y el error típico es muy inferior al máximo.

Los errores de posicionamiento se atribuyen primordialmente al acoplamiento entre el sistema mecánico (Pedestal y Engranes) y los motores.

Se desarrollo una interfaz de usuario de tipo visual para control del sistema, lo que la hace su operación bastante sencilla. Tal interfaz, da control sobre todas las capacidades del sistema y da información sobre el estado del mismo.

Consideramos que con la integración de este sistema, se logró sentar una buena base para proyectos posteriores, como lo establecía una de las directrices iniciales de está tesis. Así, el sistema es capaz de ser provisto de una mayor autonomía, permitiendo por ejemplo, el seguimiento automático de objetos. Para ejemplificar lo anterior se presentó el *posicionamiento automático con un clic*, siendo ésta, una función que pone de manifiesto algunas de las capacidades del sistema operando bajo el *software* adecuado.

V.2.- APLICACIONES

La aplicación más directa de este proyecto la lleva implícita el título de la tesis, y es la de operar como un sistema de video-vigilancia. Cámara de vigilancia que puede ser utilizada en espacios de dimensiones grandes como puede ser el lobby de un edificio, un supermercado, un estacionamiento, y en general espacios en los que se requiera una vigilancia periférica.

Con esta cámara de vigilancia, se puede cubrir todo un hemisferio de visión, permitiendo al sistema establecerse como sistema central de vigilancia; esto es, que se ubique en el centro de un lugar y que partir de está posición, realice sus tareas de vigilancia.

El sistema tiene movimiento en dos ejes, que es algo poco común entre los sistemas del mercado, además de tener un zoom que permite realizar acercamientos de hasta 10x aproximadamente.

Hay que destacar que como centro de vigilancia el sistema presenta ventajas mayores (y claro, mayor costo) a las que se pueden obtener de un sistema convencional[1]. La principal de ellas (y de la que se derivan la mayoría de las que podamos mencionar) es la de ser un sistema controlado totalmente[2] de manera digital, pues ello permite tanta flexibilidad y versatilidad de operación, como flexible e ingenioso sea el software de control asociado: la captura y almacenamiento de imágenes en formato digital; el movimiento preciso del pedestal y su posicionamiento automático con un clic son tan solo algunos ejemplos.

Sin embargo, pese a su orientación eminentemente digital el sistema tiene la opción de captura de video en formato analógico a través de una señal de video compuesta.

En relación a las aplicaciones o proyectos que tomen como base el sistema implementado, se tiene el que consideramos el más significativo y apropiado: un sistema de seguimiento automático de blancos en tiempo real. Esto es, un sistema que permita detectar los diferentes cuerpos en movimiento dentro de una secuencia de imágenes (video), "enganchar" alguno de ellos y seguirlo. Se ha implementado ya el sistema físico, el "cuerpo" capaz de realizar esta tarea, se requiere por tanto del algoritmo capaz de realizar el procesamiento de imágenes y que como resultado del procesamiento, comande los movimientos de dicho "cuerpo".

Aún cuando sabemos que desarrollar tal algoritmo no es una tarea simple, consideramos que se ha formado una base robusta para que se realice este proyecto, y una muestra es el algoritmo de posicionamiento automático que se explicó en el capítulo anterior, que a pesar de no compararse en complejidad con uno de seguimiento automático,

[1]Por sistema convencional se refiere a sistemas cuyos elementos de control se reducen a botones, *Joysticks* y potenciómetros; sistemas cuya operación está orientada a supervisión humana constante.

[2] A excepción del iris del zoom que se planea controlar eventualmente, cuando se desarrollen algoritmos en los que el factor iluminación juegue un papel importante.

sí nos da una clara idea de lo que este sistema es capaz de hacer invirtiéndole más trabajo a nivel de *software*.

En cuanto a los módulos "reciclables" del sistema, se tiene que el de control de motores puede encontrar aplicación en una infinidad de tareas. La automatización de motores es algo que se lleva a cabo en muchos sectores de la industria y se tiene que el módulo implementado tiene la capacidad de posicionarse en el lugar deseado con un error mínimo.

- ***BIBLIOGRAFÍA***

(B1) Máquinas Eléctricas y Electromagnéticas
Matsch, Leander W.
Ed. Alfaomega
1990

(B2) Conversión de Energía Electromecánica
Gourishankar, Vembu
Ed. Representaciones y servicios de ingeniería, S.A.
1975

(B3) Electricidad y Magnetismo
Jaramillo Morales, Gabriel A.
Alvarado Castellanos, Alfonso A.
Ed. Trillas
1990

(B4) Ingeniería de Control Moderna
Ogata, Katsuhiro
Ed. Prentice Hall
1993

(B5) Guía de Desarrollo Delphi 5
Teixeira, Steve.
Pacheco, Xavier.
Ed. Prentice Hall
2000

Bibliografía Electrónica.

Motores.
(E1) *http://www.maxonmotor.com/index_a.cfm*
(E2) *http://www.chiba-seimitsu.co.jp/koae.htm*
(E3) *http://www.nanotec.de/de/zweiphasen*
(E4) *http://www.eadmotors.com/index_products.html*

Codificadores.
(E5) *http://www.chibaprecision.com/ene.htm*
(E6) *http://www.servotek.com/infocntr.htm*
(E7) *http://www.encodersproducts.com/americas/rounds15.html*
(E8) *http://www.mcg-net.com/images/cx.pdf*
(E9) *http://www.dccia.ua.ex/dccia/inf/asignaturas/ROB/optativos/Sensores/internos.html*

Computadoras.
(E10) *http://www.dell.com/la/sv/es/dhs/products*
(E11) *http://www.techdepot.com/productus.asp?productid=2074626&affid=673*
(E12) *http://www.shopping.hp.com*
(E13) *http://www.compuprice.com.mx*
(E14) *http://www.advantech.com*
(E15) *http://pc104.org/products/index.html*

Tarjetas, componentes electrónicos e información en general
E16 *http://www.mechatronics.mech.nortwestern.edu*
E17 *http://www.fucrum.ru/Documents*
E18 *http://www.mesanet.com*
E19 *http://www.rock2000.com*
E20 *http://www.orreily.com/reference/dictionary/terms/B/Bus.html*

ANEXOS

- **ANEXO CARACTERÍSTICAS DEL SISTEMA**

INDICE DE REFERENCIA

LM628/LM629 CONTROLADOR DE MOVIMIENTO DE PRECISIÓN

DESCRIPCIÓN GENERAL

El LM628/LM629[1] son procesadores dedicados para control de movimiento, diseñados para ser usados con una amplia variedad de servo motores de DC , y otros servomecanismos que ofrezcan una señal de retroalimentación incremental en cuadratura. Estos procesadores realizan las tareas de cálculo necesarias en tiempo real, para un alto desempeño en el control digital de movimiento. La comunicación entre el LM628/LM629 y su procesador "anfitrión" se facilita por medio de un conjunto de comandos de alto nivel. El LM628 proporciona una salida de 8 bits que puede manejar un DAC (Convertidor Digital a Analógico) de 8 o 12 bits. Los componentes requeridos para construir un servo sistema, se reducen a un actuador/motor de DC, un codificador incremental, un DAC, un amplificador operacional y el LM628. Un sistema basado en el LM629 es similar, excepto que proporciona una salida PWM (Modulación por Ancho de Pulso) de 8 bits que puede manejar directamente a un puente H.

CARACTERÍSTICAS

*Registros de aceleración, velocidad y posición de 32 bits.
*Filtro PID digital con coeficientes programables de 16 bits
*Intervalo de muestreo derivativo programable
*Salida a DAC de 8 o 12 bits(LM628)
*Salida PWM signada de 8 bits (LM629)
*Generador de perfil de velocidad trapezoidal
*La velocidad, posición objetivo, y parámetros del filtro PID pueden se modificados en movimiento
*Modos de operación de posición y velocidad
*Capacidad de interrupciones en tiempo real hacia el procesador anfitrión
*Interfase de comunicación paralela asíncrona de 8 bits entre el LM62X y el procesador anfitrión
*Interfase de entrada para codificador incremental en cuadratura con señal índice (indicación de vuelta).

Figura 1 diagrama de bloques

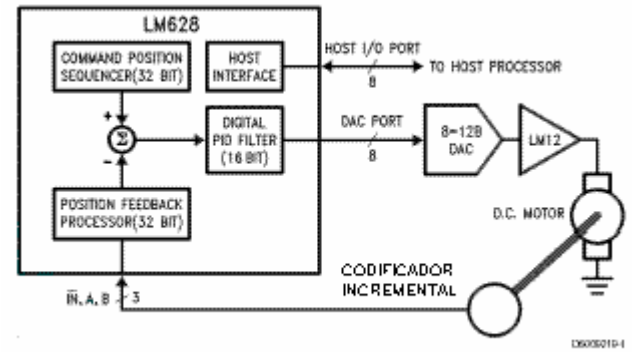
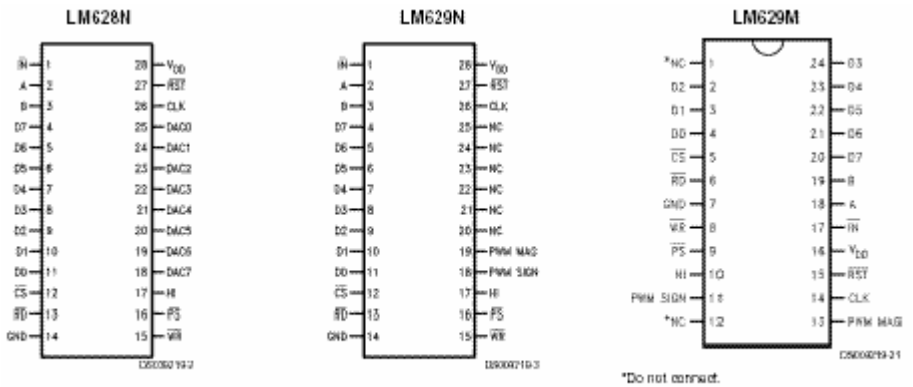


FIG 1. DIAGRAMA DE BLOQUES

DESCRIPCIÓN DE PINES

Figura 2 descripción de pines

[1] Para hacer referencia a cualesquiera de los dos controladores se usará la sintaxis LM62X.



Order Number LM629M-6, LM629M-8, LM628N-6, LM628N-8, LM629N-6 or LM629N-8
See NS Package Number M24B or N28B

VALORES DE OPERACIÓN MÁXIMOS

Voltaje en cualquier pin respecto a tierra	-0.3 a 7.0 volts
Temperatura ambiente de almacenamiento	-65.0ºC a 150ºC
Disipación de potencia máxima	605 mW
Temperatura de operación	-40.0ºC a +85.0ºC

Frecuencia de reloj :

LM628N-8 y LM628N-8	1.0MHz a 8.0 MHz
Voltaje de alimentación	4.5 a 5.5V

CARACTERÍSTICAS ELÉCTRICAS DC

V_{DD} y T_A para rangos operativos; f_{CLK}=6 MHz

SIMBOLO	PARÁMETRO	CONDICIONES	Limites de pueba		UNIDAD
			MIN	MAX	
I_{DD}	Corriente de alimentación	Salidas abiertas		110	mA
VOLTAJES DE ENTRADA					
V_{IH}	Voltaje lógico de entrada 1		2.0		V
V_{IL}	Voltaje lógico de entrada 0			0.8	V
I_{IN}	Corriente de entrada	$0 \le V_{IN} \le V_{DD}$	-10	10	µA
VOLTAJE DE SALIDA					
VOH	1 lógico	IOH = -1.6 mA	2.4		V
VOL	0 lógico	IOL = 1.6 mA		0.4	V
IOUT	Tri-state. Corriente drenada	$0 \le V_{OUT} \le V_{DD}$	-10	10	µA

CARACTERÍSTICAS ELÉCTRICAS AC

V_{DD} y T_A para rangos operativos; f_{CLK}=6 MHz; C_{CARGA} = 50pF, Señal de prueba a la entrada $t_r = t_f$= 10 ns

Intervalo de tiempo	**T#**	***Limites de prueba***		**Unidades**
		Min	**Max**	
TIEMPOS DE CODIFICADOR E ÍNDICE (ver FIG 2)				
Ancho de pulso Motor-Fase	T1	$16/f_{clk}$		µA
Espacio de tiempo por estado	T2	$8/f_{clk}$		µA
Pulso índice subida y estable	T3	0		µA
TIEMPO DE RELOJ Y REINICIO (CLOCK Y RESET)(Ver FIG 3)				
Ancho de pulso de reloj	T4	78 ó 57		ns
Periodo del reloj	T5	166 ó 125		nA
Ancho de pulso reinicio	T6	$8/f_{clk}$		ns
TIEMPO DE LECTURA DEL BYTE DE ESTATUS (Ver FIG 4)				
Tiempo subida/estable de Chip-select	T7	0		ns
Tiempo de establecimiento Port-select	T8	30		ns
Tiempo estable de Port-select	T9	30		ns
Tiempo de acceso a lectura de dato	T10		180	ns
Tiempo de captura de lectura de dato	T11	0		ns
Tiempo Alto a Hi-Z	T12		180	ns
TIEMPO DE ESCRITURA DEL BYTE DE COMANDO(Ver FIG 5)				
Tiempo subida/estable de Chip-select	T7	0		ns
Tiempo de establecimiento Port-select	T8	30		ns
Tiempo estable de Port-select	T9	30		ns
Bit “ocupado” de retraso	T13			ns
Ancho de pulso /WR	T14	100		ns
Tiempo de acceso a escritura de dato	T15	50		ns
Tiempo de captura de escritura dato	T16	120		ns

Intervalo de tiempo	**T#**	**Limites de prueba**		**Unidades**
		Min	**Max**	
TIEMPO DE LECTURA DE PALABRA(Ver FIG 6)				
Tiempo subida/estable de Chip-select	T7	0		ns
Tiempo de establecimiento Port-select	T8	30		ns
Tiempo estable de Port-select	T9	30		ns
Tiempo de acceso a lectura de dato	T10		180	ns
Tiempo de captura de lectura de dato	T11	0		ns
Tiempo Alto a Hi-Z	T12		180	ns
Bit “ocupado” de retraso	T13			ns
Tiempo de recuperación de lectura	T17	120		
TIEMPO DE ESCRITURA DE PALABRA (Ver FIG7)				
Tiempo subida/estable de Chip-select	T7	0		ns
Tiempo de establecimiento Port-select	T8	30		ns
Tiempo estable de Port-select	T9	30		ns
Bit “ocupado” de retraso	T13			ns
Ancho de pulso /WR	T14	100		ns
Tiempo de acceso a escritura de dato	T15	50		ns
Tiempo de captura de escritura dato	T16	120		ns
Tiempo de recuperación de escritura	T18	120		ns

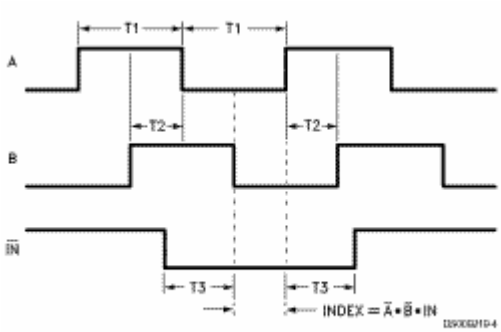


FIG 2. TIEMPO DE CUADRATURA DEL CODIFICADOR

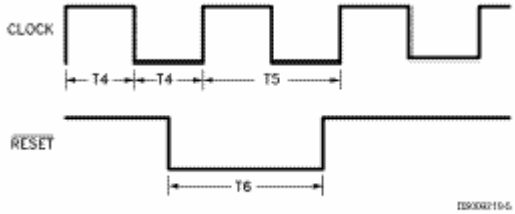


FIG 3. TIEMPO DE RELOJ Y RESET

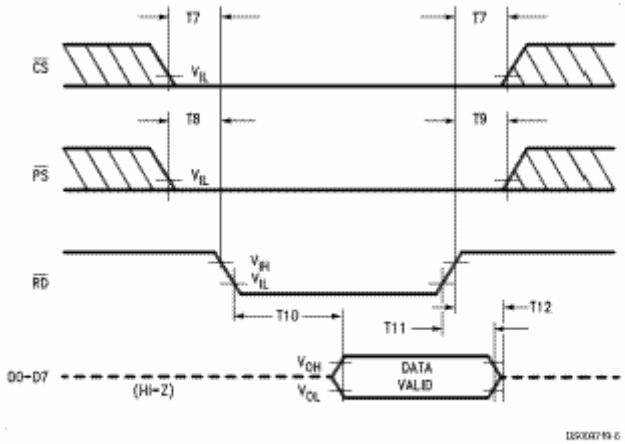


FIG 4. TIEMPO DEL BYTE DE ESTATUS

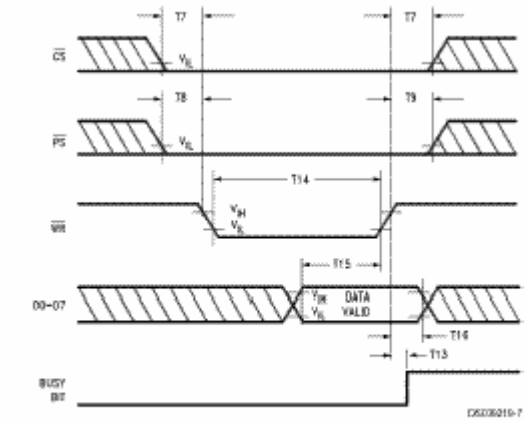


FIG 5. TIEMPO DE ESCRITURA DEL BYTE DE COMANDO

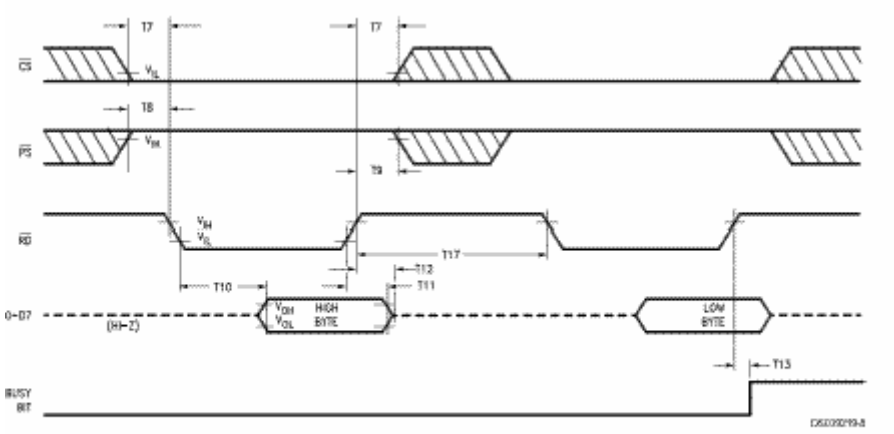


FIG 6. TIEMPO DE LECTURA DE DATO

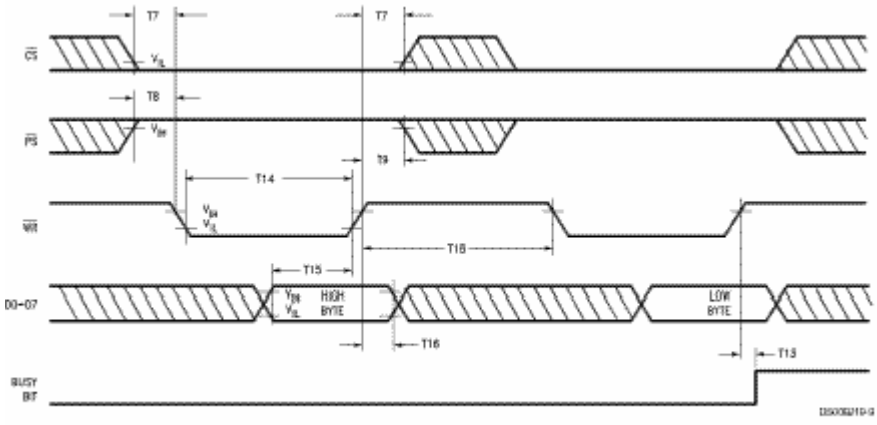


FIG 7. TIEMPO DE ESCRITURA DE DATO

Pin 1,ĪN: entrada del pulso índice.-Recibe el pulso índice (opcional) del codificador. Debe llevarse a estado alto si no se va a usar.

Pines 2 y 3,A y B: entrada de señales del codificador.-Reciben las dos fases en cuadratura (A y B) del codificador de posición incremental. Por convención, cuando el motor está girando hacia delante , la señal en el pin 2 adelante a la señal en el pin 3 por 90 grados. Las señales en dichos pines tardan 8 periodos de reloj en estabilizarse y ser propiamente reconocidas por el LM62X . Gracias al método de decodificar las señales en cuadratura A y B, se tiene una precisión de 4:1 respecto a las líneas del codificador incremental, esto es , si el codificador cuenta con 1000 líneas por vuelta, la decodificación logra 4000 pulsos por vuelta. La tasa máxima de captura de pulsos es de 1 MHz para la frecuencia de reloj de 8 MHz, lo que significa, por ejemplo, que un codificador de posición que entregue 10000 pulsos por vuelta, puede seguirse hasta 100 revoluciones por segundo(10000*100=1MHz).

Pines 4 al 11, D0-D7: puerto paralelo de comunicación con el procesador anfitrión.-Este es un puerto de datos bidireccional que comunica al LM62X con el procesador de la computadora anfitriona. Se usa para escribir datos y comandos al LM62X, y para leer el status byte y datos (como la posición) del LM62X. La figura de este puerto es controlada por ¯CS (pin 12), ¯ PS (pin 16), ¯ RD (pin 13), ¯ WR (pin 15).

Pin 12, ¯CS: entrada Chip Select.-Se usa para seleccionar al LM62X en operaciones de lectura y escritura.

Pin 13, ¯RD: pin de lectura.-Usado para indicar leer el status y datos del LM62X

Pin14, GND: tierra.- Pin de retorno de la alimentación

Pin 15, ¯WR: pin de escritura.-Usado para indicar escribir comandos y datos al LM62X

Pin 16, ¯ PS: selector de puerto.-Define si D0-D7 actuará como puerto de comandos o puerto de datos: se selecciona puerto de comandos con un nivel bajo, y puerto de datos con un nivel alto.

Pin 17, HI: interrupción al anfitrión.-Esta señal activa-alta, alerta al procesador anfitrión (vía una rutina de interrupción) que una condición de interrupción a tenido lugar.

Pines 18 al 25, DAC0-DAC7.- Puerto de salida que se utiliza en tres diferentes modos:

1.-LM628 (modo de salida de 8 bits):Proporciona una señal d control de 8 bits hacia un DAC. El MSB (Bit Más Significativo) es el pin 18 y el LSB (Bit Menos Significativo) es el pin 25.

2.-LM628(modo de salida de 12 bits):Proporciona dos palabras multiplexadas de 6 bits. La palabra menos significativa es proporcionada primero. El MSB es el pin 18 y el LSB es el pin 23. El pin 24 se usa para demultiplexar las palabras; el pin 24 tiene nivel bajo para la palabra menos significativa.

3.-LM629(salidas de signo y magnitud): Proporciona una señal PWM signada: la "magnitud" a través del pin 19 y el signo a través del pin 18. Los pines 20 al 25 no se usan en el LM629.

Pin 26, CLK: reloj.- Recibe el reloj del sistema.

Pin 27,¯RST: reinicio.- Este pin, activo-bajo durante al menos 8 ciclos de reloj, reinicia en el flanco de subida el LM62X a las condiciones iniciales que se presentan a continuación:

1.-Los coeficientes del filtro y parámetros de trayectoria son llevados a cero
2.-El umbral de error de posición se lleva al valor máximo (7FFF hexadecimal), y ejecuta el comando LPEI
3.-Se enmascara (deshabilita) la interrupción SBPA/SBPR.
4.-Las otras cinco interrupciones son desenmascaradas (habilitadas).
5.-La posición actual de los ejes del motor se define como la posición cero o *Home.*
6.- El intervalo de muestreo derivativo se lleva a 1/2048 del reloj (256µs para un reloj de 8MHz).
7.-La salida a DAC de 8 bits es 80 hexadecimal (media escala = salida cero) , y la salida a DAC de 12 bits es 800 hexadecimal (media escala = salida cero)

Inmediatamente después de liberara el pin de reinicio del LM62X , el registro de status debe contener "00". Si el reinicio es exitoso, el registro de status deberá contener "84"o "C4" hexadecimal después de 1.5ms de liberado el pin de reinicio. Si el contenido del status no ha cambiado a "84" o "C4", deberá realizarse otro reinicio. Para asegurar que se ha llevado a cabo propiamente el reinicio, se ejecuta el comando RSTI. Si el chip se reinicio adecuadamente, el status cambiara de "84" a "80" o de "C4" a "C0".

Pin 28:Vcc: alimentación de voltaje.-Pin de alimentación de voltaje a 5volts.

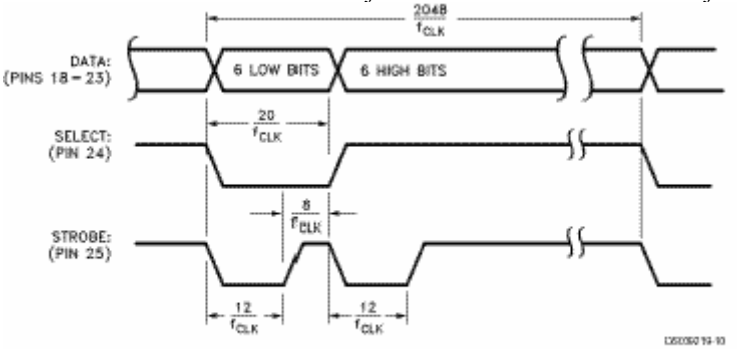


FIG 8. TIEMPO DE SALIDA DE BIT MULTIPLEXADO

TEORÍA DE OPERACIÓN

Introducción

El diagrama a bloques del sistema típico, se presenta en la figura 1. El procesador anfitrión se comunica con el LM62X a través de un puerto I/O para programar un perfil de velocidad trapezoidal y un filtro digital de compensación. La salida del LM628, irá a un DAC para producir una señal analógica que será amplificada por una etapa de potencia que alimentará al motor o servomecanismo, mientras que la salida de un LM629 controlará de forma directa una etapa de potencia basada en puente H. Un codificador incremental proporciona retroalimentación al LM62X para cerrar el lazo de control del motor o servomecanismo.

El generador de perfiles de velocidad trapezoidales calcula la trayectoria tanto en los modos de operación por velocidad y por posición. En operación, l LM62X sustrae la posición actual (posición retroalimentada) de la posición deseada (posición instantánea calculada por el generador de perfil de velocidad) y el error resultante es alimentado al filtro digital para llevar al motor a la posición deseada. La Tabla 1 proporciona un breve sumario de especificaciones del LM62X .

Tabla 1

Rango de posición	-1,073,741,842 a 1,073,741,823 pulsos
Rango de velocidad	De 0 a 1,073,741,823/2E16 pulsos/muestra
Rango de aceleración	0 a 1,073,741,823/2E16 cuentas/muestreo/muestre
Señal de salida para control del motor	LM628:salida de 8 bits paralelos hacia DAC de 8bits o salida multiplexada de 12 bits hacia DAC de 12bits
Modos de operación	Posición y velocidad
Dispositivo de retroalimentación	Codificador incremental (señales en cuadratura A y B y pulso índice Z opcional)
Algoritmo de control	Filtro Proporcional Integral Derivativo PID y límite de integración programable.
Intervalos de muestreo	Término derivativo actualizable en múltiplos de 256µs para un reloj de 8MHz Termino integral y proporcional actualizable cada 256µs para un reloj de 8MHz

Interfase de retroalimentación de posición

La interfase entre el LM62X y el motor es un codificador de posición incremental. El LM62X proporciona tres entradas: dos en cuadratura A y B y un pulso índice de vuelta completa. Las señales en cuadratura son usadas para conocer la posición absoluta del motor. Cada vez que ocurre una transición en alguna de las señales en cuadratura, esto es cada vez que hay un cambio de flanco, el registro de posición interno del LM62X es incrementado o decrementado, según sea el caso. Esto resulta en una resolución de cuatro veces la indicada por las líneas del codificador de posición. Ver la figura 9. Cada una de las señales del codificador está sincronizada con el reloj del LM628.

El pulso índice opcional proporcionado por algunos codificadores, asume un estado lógico bajo una vez por revolución. El usuario de LM62X puede programar que la posición absoluta del motor sea grabada en un registro especial (el registro de índice) cuando ocurra el pulso índice al completarse una revolución (lo cual ocurre cuando las tres señales del codificador tienen simultáneamente un estado bajo).

Si el codificador no proporciona una señal índice, la entrada índice del LM62X puede usarse en conjunción con un interruptor límite y una interrupción, que indique al procesador anfitrión cuándo se ha alcanzado una posición de interés. Si la entrada del pulso índice se lleva permanente a tierra se puede ocasionar un malfuncionamiento del sistema.

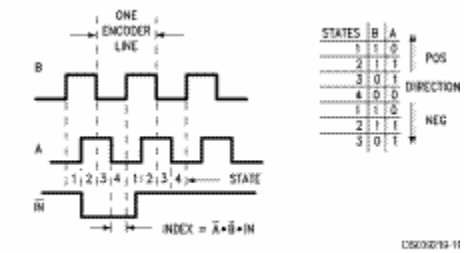


FIG 9. SEÑALES DEL CODIFICADOR EN CUADRATURA

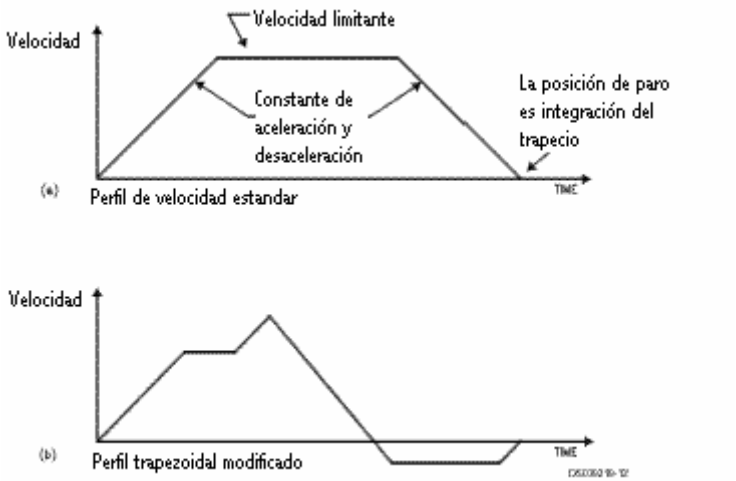


FIG 10. PERFILES DE VELOCIDAD

GENERADOR DE PERFIL DE VELOCIDAD (TRAYECTORIA)

El generador de perfil de velocidad trapezoidal calcula la posición deseada del motor versus el tiempo. En el modo de operación por posición, el procesador anfitrión especifica a aceleración, velocidad máxima y posición final deseada. El LM62X utiliza esta información para controlar el movimiento del motor, acelerándolo a la tasa especificada, hasta alcanzar la velocidad máxima, y desacelerando a la misma tasa. En caso que os cálculos del LM62X lo indiquen, la desaceleración puede comenzar antes de alcanzar la velocidad máxima, para así llegar a la posición deseada sin sobrepaso. En cualquier tiempo durante el movimiento, la velocidad y la posición final pueden ser cambiadas, realizándose las correcciones de calculo "al vuelo".La figura 10a presenta un perfil trapezoidal de velocidad típico sin variación de parámetros. En tanto la figura 10b presenta un perfil de velocidad modificado "al vuelo".

Al operar en el modo de velocidad, el motor acelera hacia la velocidad especificada a la tasa programada, y mantiene esa velocidad hasta que se envíe el comando de parar. La velocidad es mantenida constante aumentando en forma uniforme la posición instantánea deseada. Si hay perturbaciones en el movimiento, la velocidad promedio a largo plazo es constante. Si el motor es incapaz de mantener la velocidad deseada (lo que podría suceder por ejemplo si el rotor se bloqueara), la posición deseada continuaría incrementándose, lo cual incrementaría así mismo el error de posición. Si esta condición pasa desapercibida y el rotor es liberado repentinamente, este podría alcanzar una velocidad muy grande tratando de alcanzar la posición deseada. Esta condición puede ser fácilmente detectable: ver los comandos LPEI y LPES.

Todos los parámetros de trayectoria son valores de 32 bits. La posición es una cantidad signada. La aceleración y velocidad son números enteros con 16b bits de parte entera, y 16 bits de parte fraccionaría. La parte entera de velocidad y aceleración, hacen referencia a pulsos enteros por unidad de tiempo. En tanto las partes fraccionarias hacen alusión a "fracciones" de pulso, lo que no existe físicamente, pero se acumula en un sumador hasta completar pulsos enteros.

El siguiente es un ejemplo numérico de cómo calcular los parámetros de trayectoria. Supóngase un codificador de 500 líneas, y que se desea que el motor acelere a 1 revolución por segundo, hasta alcanzar una velocidad de 600 rpm, y que pare después de haber recorrido 100 revoluciones:

Sea P = posición objetivo (unidades en pulsos o cuentas)
sea R = líneas_del_codificador *4 = (resolución del sistema)
entonces R = 500*4 = 2000
y P=R*número_de_revoluciones= 2000* 100=200,000 cuentas (valor a cargar)
que convirtiendo en valor hexadecimal P=00030D40(código a cargar)
sea V=velocidad (unidades 0=pulso/muestreo)
sea T=tiempo de muestreo en segundos= 341µs para reloj de 6MHz
sea C=factor de conversión =1minuto/60 segundos
entonces V=R*T*C*vel_deseada_en_rpm
V=2000*341E-6*1/60*600rpm
V=6.82 pulsos/muestreo
V(scalado)= 6.82*65536=446,955.52
V(redondeado)=446,956
V(codificado)=0006D1EC
Sea A=aceleración (unidades en pulsos/muestreo/muestreo)
A=R*T*T*aceleración_deseada rev/sec/sec

entonces A=2000*341E-6 * 341E-6*1 rev/sec/sec
y A=2.33E-4 pulsos/muestreo/muestreo
A(escalado)=2.33E-4*65,536=15.24
A(redondeado)=15(valor a cargar)
A(codificado)=0000000F (código hexadecimal a a cargar al LM62X)

Los valores de posición, velocidad y aceleración calculados arriba, deben ser convertidos a código binario para ser cargados al LM62X. Los valores calculados de velocidad y aceleración, deben ser multiplicados por 65,536, lo cual equivale a un corrimiento "hacia la derecha" de 16 bits, de tal forma que tales valores adquieran el formato adecuado de 16 bits de parte entera y 16 bits de parte fraccionaria. Por otra parte el factor de escalamiento de 4 para la resolución del codificador, es resultado del método con el cual el LM62X decodifica las señales en cuadratura del codificador (Ver figura 9).

FILTRO PID DE COMPENSACIÓN

El LM62X utiliza un filtro digital PID para compensar el lazo de control. El motor es mantenido en la posición deseada aplicándole una fuerza restauradora proporcional al error de posición, a la integral del error de posición, y a la derivada del error de posición. La siguiente ecuación en tiempo discreto ilustra el control que realiza el LM62X:

$$U(n)= Kp * e(n) + Ki \sum_{N=0}^{n} e(n) + Kd[e(n`) – e(n`- 1)] \quad \text{.....Eq.1}$$

Donde U(n) es la señal de control del motor en el tiempo de muestreo n; e(n) es el error de posición al tiempo de muestreo n; n` indica un tiempo de muestreo derivativo y n`-1 es el tiempo de muestreo derivativo anterior; Kp, Ki y Kd son las constantes proporcional, integral y derivativa respectivamente, las cuales son programadas por el usuario.

El primer término, el proporcional, proporciona una fuerza restauradora proporcional al error de posición, justo como lo haría un resorte siguiendo la ley de Hooke. El segundo término, el integral, proporciona una

fuerza restauradora que crece con el tiempo, y asegura que el error en estado estático sea cero. Si existiera una carga constante sobre el eje del motor, este término hace posible que el error aún pueda ser cero.

El tercer término, el término derivativo, proporciona una fuerza proporcional a la tasa de variación del error. Actúa como un amortiguador viscoso en un sistema amortiguador-masa (justo como los amortiguadores de un automóvil). El periodo de muestreo asociado con el término derivativo es programable por el usuario; esta característica permite al LM62X manejar un amplio rango de cargas inerciales al proporcionar una mejor aproximación a la derivada continua. En general se puede establecer que periodos de muestreo más largos, son recomendables para operaciones a baja velocidad.

En operación, el algoritmo del filtro recibe una señal de error de 16 bits del nodo de suma del lazo de control. La señal de error se satura a 16 bits para asegurar un comportamiento predecible. Además de ser multiplicada por el término Kp, la señal de error es adicionada a una acumulación ponderada de errores previos (que forman la señal integral), y a una tasa definida por el periodo de muestreo derivativo, también se suma el error previo sustraído del actual y ponderado por el término derivativo Kd. Todas las multiplicaciones de la Eq.1 son operaciones de 16 bits: los resultados se truncan a los 16 bits menos significativos.
La señal integral se mantiene a 24 bits, aunque sólo los 16 bits más significativos son usados. Esta técnica de escalamiento resulta en un rango de valores del parámetro Ki menos sensible. Los 16 bits se corren a le derecha ocho posiciones y multiplicadas por Ki para formar así el término derivativo. El valor absoluto de esta cantidad es comparado con el parámetro il, y el menor de ellos es el que contribuye a la señal de salida de control.

La señal derivativa es multiplicada por el coeficiente Kd cada periodo de muestreo derivativo. Este producto contribuye al control del movimiento del motor cada intervalo de muestreo (el cual siempre es menor o igual al intervalo de muestreo derivativo), independientemente del periodo de muestreo derivativo elegido por el usuario.

Los términos derivativo, proporcional e integral (limitado por il) son sumados para formar una cantidad de 16 bits . Dependiendo del formato de salida (tamaño de palabra de la salida), los 8 o 12 bits más significativos son la señal de salida de control.

OPERACIONES DE LECTURA Y ESCRITURA PARA EL LM62X

El procesador anfitrión al LM62X vía el puerto I/O que lo comunica con el LM62X cuando la entrada Port Select $\overline{\text{PS}}$ (pin 16) tiene un nivel bajo. El código del comando deseado se aplica al puerto de comunicación paralelo y la línea de escritura $\overline{\text{WR}}$ (pin 15) es llevada a nivel bajo. El byte de código del comando es cargado al LM62X en el flanco de subida de $\overline{\text{WR}}$. Al escribir un byte de comando, primero es necesario verificar el bit 0 (*busy bit*) del byte de status. Si tal bit tiene un estado alto, no se podrá escribir ningún comando al LM62X.Este bit no estará en estado alto más de 100µs y comúnmente “caerá” en un tiempo de 15µs a 25µs.
El procesador anfitrión lee el byte de status de una manera similar; dando un nivel bajo a $\overline{\text{RD}}$ (pin 13) cuando $\overline{\text{PS}}$ (pin 16) tiene un estado bajo; la información del status permanece válida en tanto $\overline{\text{RD}}$ tenga un estado bajo.

Leer y escribir datos del/al LL62X (operación opuesta a leer datos y el byte de status) se hace con $\overline{\text{PS}}$ (pin 16) en estado alto. Estas lecturas y escrituras se dan siempre en un par de bytes (de 1 a 7 pares para un sólo comando), siendo el primer byte el más significativo. Cada byte requiere un estrobo de las líneas $\overline{\text{WR}}$ y $\overline{\text{RD}}$ de escritura y lectura respectivamente. Al transferir palabras de datos (pares de bytes), primero es necesario consultar el estado del byte de status y verificar que el *busy bit* está en nivel bajo antes de enviar el par de bytes. Esta operación ha de realizarse cada vez que se envíe un par de bytes. La transferencia de datos se realiza a través de interrupciones internas del LM62X y esto se indica a través del *busy bit*. Si algún comando es enviado mientras el *busy bit* tiene un nivel alto, tal comando será ignorado.

SALIDAS AL MOTOR

La salida a DAC del LM628 puede ser configurada para proporcionar una salida de 8 bits, o una salida multiplexada de 12 bits. La salida a 8 bits puede conectarse directamente a un DAC de 8 bits, en tanto la salida a 12 es multiplexada en palabras de 6 bits. La salida a DAC es signada, por lo que el cero de la escala en el modo de 8 bits es 80 hex y 800 hex en el de 12 bits. Valores menores a este valor ocasionan una torca hacia el sentido definido como negativo y a la inversa valores superiores a los anteriores ocasionarán una torca positiva.
En tanto el LM629 proporciona una salida PWM signada a través de dos pines. Magnitud y sentido. La figura 11 muestra el formato de la salida PWM del LM629.

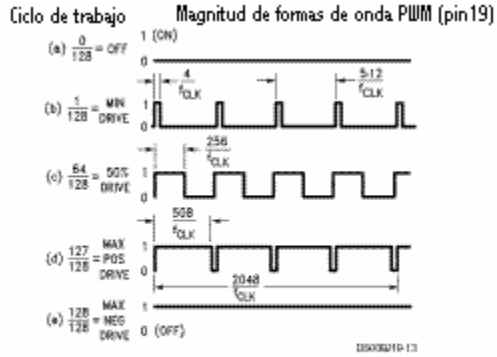


FIG 11. FORMAS DE ONDA DE SEÑAL DE SALIDA PWM

Tabla2 de los comandos de usuario del LM62X

Comando	**Tipo**	**Descripción**	***Hex***	**Bytes de datos**
RESET	Inicializar	Reinicia el LM628	00	0
PORT8	Inicializar	Selecciona salida de 8 Bit	05	0
PORT12	Inicializar	Selecciona salida de 12 Bit	06	0
DFH	Inicializar	Define referencia cero	02	0
SIP	Interrupción	Establece posición índice	03	0
LPEI	Interrupción	Interrupción en error	1B	2
LPES	Interrupción	Para en error	1A	2
SBPA	Interrupción	Establece *breakpoint,*Absoluto	20	4
SBPR	Interrupción	Establece *breakpoint,*Relativo	21	4
MSKI	Interrupción	Mascara de interrupciones	1C	2
RSTI	Interrupción	Reinicia interrupciones	1D	2
LFIL	Filtro	Carga parámetros del filtro	1E	2 a 10
UDF	Fltro	Actualiza el filtro	04	0
LTRJ	Trayectoria	Carga trayectoria	1F	2 a 14
STT	Trayectoria	Comienza movimiento	01	0
RDSTAT	Reporta	Lee byte de estatus	-	1

CONJUNTO DE COMANDOS DE USUARIO

Generalidades
Los siguientes párrafos describen el conjunto de comandos de usuario del LM62X. Algunos de los comandos pueden ser realizados sin argumentos y otros requieren de 1 a 7 pares de bytes como argumento. Como ejemplos el comando STT (de StarT motion o comenzar movimiento) no requiere de ningún argumento, en tanto el comando LFIL (Load FILter o cargar filtro) requiere de 2 10 bytes de datos.
Los comandos se dividen en varias categorías : de inicialización, de interrupción de control, de control de filtro, de control de trayectoria y de reporte de datos. Los comandos se enlistan en la Tabla 2 y se describen a detalle en los siguientes párrafos, junto con la abreviación de cada comando se presenta su nombre, código y cantidad de argumentos que requiere.

Comandos de inicialización

Los siguientes cuatro comandos del LM628 se utilizan para inicializar el sistema:

Comando **RESET**: reinicia el LM62X

Código del comando: 00Hex
Bytes de argumento: Ninguno
Ejecutable en movimiento:Sí

Este comando (y el pin 27 de reinicio por *hardware*) llevan los siguientes parámetros a cero: los coeficientes del filtro y sus *buffers* de entrada, los parámetros de trayectoria y sus *buffers* de entrada, así como la salida de control (la cual se lleva a la mitad de la escala que es 80 Hex para salida a 8 bits, y 800Hex para salida a 12 bits). Este comando también limpia cinco de las 6 máscaras de interrupción (sólo la interrupción SBPA/SBPR está enmascarada), define la salida a DAC a 8 bits, y define la posición actual como el cero absoluto *Home.* El comando RESET que puede ser ejecutado en cualquier momento tarda menos de 1.5 ms en ser ejecutado.

Comando **PORT8**: establece la salida a DAC en 8 bits
Código del comando: 05Hex
Bytes de argumento: Ninguno
Ejecutable en movimiento:No aplica

El tamaño de salida a DAC es de 8 bits tras el RESET, por lo que este comando no es necesario si se piensa utilizar un DAC a 8 bits y aplica sólo para el LM628.

Comando **PORT12**: establece la salida a DAC en 12 bits
Código del comando: 05Hex
Bytes de argumento: Ninguno
Ejecutable en movimiento:No aplica

Establece la salida DAC a 12 bits en 2 palabras multiplexadas de 6 bits cada una y aplica sólo para el LM628.

Comando **DFH :** define posición absoluta cero
Código del comando: 02Hex
Bytes de argumento: Ninguno
Ejecutable en movimiento: Sí

Este comando define la posición inicial como "*Home*" o posición absoluta cero. Si DFH se ejecuta durante el movimiento ello no afectará la posición de paro dek movimiento en curso, a menos que se ejecute también el comando STT.

Comandos de control de interrupción

Los siguientes siete comandos se relacionan con las condiciones que pueden causar una interrupción a la computadora anfitriona. Para que cualquiera de las posibles condiciones de interrupción obtengan servicio del anfitrión vía el pin 17, el bit correspondiente en el registro manipulado por el comando MSKI debe tener un estado lógico alto, esto es, debe estar desenmascarado. La identidad de todas las interrupciones se hace conocer al anfitrión vía la lectura del status byte. Incluso cuando todas las interrupciones estén enmascaradas el status byte seguirá reflejando el estado de cada condición. Esta característica permite diferenciar las labores de información del status, con las de interrupción.

Comando **SIP** : establecer posición del Índice
Código del comando: 03Hex

Bytes de argumento: Ninguno
Ejecutable en movimiento: Sí

Después de que este comando es ejecutado, la posición absoluta correspondiente a la ocurrencia del próximo pulso índice será grabada en el registro índice, y el bit 3 del byte de status deberá ir a un estado alto. La posición es grabada en el momento en que ambas fases del codificador y el pulso índice tienen un estado bajo. Posteriormente el registro índice puede ser leído utilizando el comando RDPI, y así facilitar la definición de una posición *Home* con ayuda de una señal de pulso índice. El usuario también puede disponer que la ocurrencia de un pulso índice ocasione una interrupción a la computadora anfitriona (ver comandos RSTI y MSKI).

Comando **LPEI** : cargar posición de error para interrupción
Código del comando: 1BHex
Bytes de argumento: 2
Rango del argumento: 0000 a 7FFFHex
Ejecutable en movimiento: Sí

Un error de posición excesivo (la salida del nodo de suma del lazo de control) puede indicar un serio problema en el sistema, como un rotor bloqueado. La instrucción LPEI permite que el usuario programe un umbral de error que al ser rebasado levante el bit 5 del byte de status y opcionalmente genere una interrupción hacia la computadora anfitriona. El primer byte de argumento enviado es el más significativo.

Comando **LPES** : cargar posición de error para paro
Código del comando: 1AHex
Bytes de argumento: 2
Rango del argumento: 0000 a 7FFFHex
Ejecutable en movimiento: Sí

Similar al comando anterior, con la diferencia de que se detiene el motor (la señal se salida es llevada a media escala) al rebasarse el umbral de error. Al igual que con el comando LPEI el bit 5 refleja el estado de la posición de error.

Comando **SBPA**: establecer un *breakpoint* absoluto
Código del comando: 20Hex
Bytes de argumento: 4
Rango del argumento: C0000000 a 3FFFFFFFHex
Ejecutable en movimiento: Sí

Este comando habilita al usuario para establecer un punto de ruptura o *breakpoint* en términos de valores absolutos de posición. El bit 6 del byte de status es llevado a nivel alto cuando el *breakpoint* programado es rebasado. Esta condición es útil como indicador para realizar actualizaciones de los parámetros del filtro o trayectoria. También se puede programar interrupción a la computadora anfitriona al alcanzar el *breakpoint.*

Comando **SBPR**: establecer *breakpoint* relativo
Código del comando: 21Hex
Bytes de argumento: 4
Rango del argumento: ver el texto
Ejecutable en movimiento: Sí

Similar al comando anterior con la diferencia que el *breakpoint* es relativo a la posición objetivo actual. El valor de esta posición relativa debe ser tal que al sumarse a la posición objetivo actual, no se sobrepasen los rangos establecidos para posición absoluta.

Comando **MSKI**: máscara de interrupciones
Código del comando: 1CHex
Bytes de argumento: 2
Rango del argumento: ver el texto
Ejecutable en movimiento:Sí

El comando MSKI permite al usuario definir que eventos pueden generar una interrupción al anfitrión. Bit 1 al 6 del byte de status son indicadores de las 6 condiciones candidatas para interrumpir al procesador anfitrión. Al ser interrumpido, el procesador anfitrión puede consultar el status byte para conocer que condición generó la interrupción. Nótese que el comando MSKI es seguido inmediatamente por 2 bytes de datos. Los bits 1 a 6 del segundo byte (el menos significativo) indican cuales interrupciones se habilitarán. Si se escribe un 1 a un bit, se habilita la interrupción respectiva, un cero enmascara la interrupción.
El primer byte de datos enviado no tiene efecto sobre el estado de las interrupciones(Ver la tabla 3 para conocer la posición de las interrupciones).

Tabla3.-Ubicación de las interrupciones

Posición del Bit	**Función**
Bits 15 al 7	No usados
Bit 6	Interrupción por *breakpoint*
Bit 5	Interrupción por error en posición
Bit 4	Interrupción por desbordamiento
Bit 3	Interrupción por pulso índice
Bit 2	Interrupción trayectoria completa
Bit 1	Interrupción por comando Error
Bit 0	No usado

Comando **RSTI** : bajar banderas de interrupción
Código del comando: 1DHex
Bytes de argumento: 2
Rango del argumento: ver el texto
Ejecutable en movimiento:Sí

Cuando alguna de las condiciones potenciales de interrupción ocurre, se utiliza el comando RSTI para bajar la bandera correspondiente en el byte de status. Las banderas de status pueden ser bajadas una a una u todas a la vez. La distribución de las banderas es similar a la del comando MSKI para el segundo byte de datos y los valores del primer byte no tienen ningún efecto en el status byte.

Comandos de control del filtro

Los siguientes dos comandos del LM62X se utilizan para establecer el intervalo de muestreo derivativo, y ajustar los parámetros del filtro PID.

Comando **LFIL** :cargar los parámetros del filtro
Código del comando: 1EHex
Bytes de argumento: 2 a 10
Rango del argumentos:
Palabra de control del filtro: ver el texto
Coeficientes del filtro: 0000 a 7FFFHex (sólo posición)
Límite de integración: 0000 a 7FFFHex (sólo posición)
Ejecutable en movimiento:Sí

Este comando sirve para cargar los parámetros del filtro PID de compensación Kp, Kd, Ki así como el límite de integración il. El límite de integración il establece el valor máximo que puede alcanzar el término integral del filtro. El primer byte del argumento de este comando establece el intervalo de muestreo derivativo como un múltiplo entero, de 1 a 255, del periodo de muestreo. El segundo byte del argumento indica los parámetros del filtro a ser cargados. La tabla 4 indica el orden de estos parámetros. Los bytes de datos enviados posteriormente, indicarán el valor de Kp, Ki, Kd e il (o de los parámetros que se haya indicado que se cambiarán, pero siempre en ese orden) enviando el byte más significativo de cada parámetro primero. Recordar que antes de enviar cualquier par de bytes debe verificarse que el estado del *busy bit* del byte de status tenga un nivel bajo. El comando LFIL cargará los parámetros indicados a una serie de registros tipo *buffer,* esto es no se actualizaran los valores de tales parámetros, en tanto no se halla enviado el comando UDS que se revisa en seguida.

Tabla 4 Palabra de control de modificación de filtro

Posición de Bit	**Función**
Bit 15	Intervalo de muestreo derivativo Bit 7
Bit 14	Intervalo de muestreo derivativo Bit 6
Bit 13	Intervalo de muestreo derivativo Bit 5
Bit 12	Intervalo de muestreo derivativo Bit 4
Bit 11	Intervalo de muestreo derivativo Bit 3
Bit 10	Intervalo de muestreo derivativo Bit 2
Bit 9	Intervalo de muestreo derivativo Bit 1
Bit 8	Intervalo de muestreo derivativo Bit 0
Bit 7	No usado
Bit 6	No usado
Bit 5	No usado
Bit 4	No usado
Bit 3	Cargando el dato kp
Bit 2	Cargando el dato ki
Bit 1	Cargando el dato kd
Bit 0	Cargando el dato il

Comando **UDF** :actualizar los parámetros del filtro
Código del comando: 04Hex
Bytes de argumento: ninguno

Este comando actualiza los valores cargados por medio del comando LFIL. La acción del comando UDF consiste en transferir los parámetros del filtro de los *buffers* a los registros de trabajo que son los que se realizan los cálculos del filtro. Por tanto, en tanto no se ejecute UDF, los cálculos del filtro estarán utilizando los últimos valores actualizados.

Comandos de control de trayectoria

Los siguientes dos comandos de usuario son utilizados para establecer los parámetros de control de trayectoria (posición, velocidad, aceleración), modo de operación (posición o velocidad) y sentido de movimiento (para el modo de velocidad) necesarios para describir el movimiento deseado o para seleccionar el modo de movimiento con paro "manual" y para establecer la sincronización de los cambios en el sistema.

TABLA 5. Intervalo de muestreo del término derivativo, Códigos de selección

Posición del Bit								**Intervalo de muestreo seleccionado**
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	
0	0	0	0	0	0	0	0	256 µs
0	0	0	0	0	0	0	1	512 µs
0	0	0	0	0	0	1	0	768 µs
0	0	0	0	0	0	1	1	1024 µs
...	...	...	...	...	...	...	...	Etc...
1	1	1	1	1	1	1	1	65536 µs

TABLA 6.- Localidades del control de trayectoria

Posición de Bit	**Función**
Bit 15	No usado
Bit 14	No usado
Bit 13	No usado
Bit 12	Hacia delante (modo de velocidad)
Bit 11	Modo de velocidad
Bit 10	Para suavemente (desaceleración prog.)
Bit 9	Para abruptamente(desaceleración máx.)
Bit 8	Apaga el motor
Bit 7	No usado
Bit 6	No usado
Bit 5	La aceleración será cargada
Bit 4	El dato de aceleración es relativo
Bit 3	Velocidad será cargada
Bit 2	El dato de velocidad es relativo
Bit 1	La posición será cargada
Bit 0	El dato de posición es relativo

Comando **LTRJ**: cargar parámetros de trayectoria.
Código del comando: 1Fhex
Bytes de argumento: 2 a 14
Rango del argumentos:
Palabra de control de trayectoria: ver el texto
Posición: C0000000 a 3FFFFFFFHex
Velocidad: 00000000 a 3FFFFFFFHex
Aceleración: 00000000 a 3FFFFFFFHex
Ejecutable en movimiento:Sí

Los parámetros de control de trayectoria que son escritos al LM62X son: aceleración, velocidad y posición. Además se programa si estos parámetros deben ser considerados absolutos o relativos, si el control operará en modo de velocidad o posición. Después de escribir el código del comando, los primeros dos bytes de argumento indican qué parámetros van a ser cambiados. El primer byte escrito es el más significativo. De esta forma, estos dos primeros bytes constituyen la palabra de control que informa al LM62X de la naturaleza y número de los bytes de argumento que resta por enviar. Ver la Tabla 6 para ver la definición de los bits en la palabra de control.

El bit 12 determina la dirección del motor cuando opera en el modo de velocidad. Un uno lógico indica la dirección "positiva".Este bit no tiene efecto si el LM62X opera en el modo de posición.
El bit 11 en estado alto determina si el LM62X opera en modo de velocidad o en modo de posición en estado bajo.
Los bits 8 al 10 indican el modo de paro manual. Estos bits no se proporcionan para que uno simplemente especifique el modo de paro; en el modo de operación de posición, el paro normal siempre es suave y ocurre automáticamente al alcanzarse la posición deseada. Bajo circunstancias excepcionales puede ser deseable intervenir "manualmente" con el proceso de generación de trayectoria al hacer un paro prematuro. En la operación en modo de velocidad, sin embargo, el medio normal de paro es a través de los bits 8 al 10 (generalmente se usará el bit 10). El bit 8 se lleva a nivel alto para parar el motor llevando la salida de control a media escala; el bit 9 se lleva a estado alto para parar el motor abruptamente, esto es aplicándole la máxima desaceleración posible (haciendo de la posición actual la posición deseada); el bit 10 en estado alto indica que el paro será suave a la tasa de desaceleración programada. Los bits 8 al 10 deben usarse de manera exclusiva: sólo uno de ellos debe tener estado alto al mismo tiempo.
Los bits 0 al 5 informan al LM62X si alguno, todos o ninguno de los parámetros de trayectoria serán cargados y si tales datos han de ser interpretados como absolutos o relativos. El usuario informa al LM62X los parámetros escribiendo un 1 lógico en el bit correspondiente. Cualquier parámetro puede ser modificado durante el movimiento, sin embargo, si la aceleración es cambiada, debe esperarse que se complete la secuencia en curso antes de ejecutar el comando STT, o en su defecto realizar un paro manual del motor.
Los bytes de datos que siguen a la palabra de control se envían en pares de bytes que forman palabras de 16 bits. Cada parámetro requiere de dos palabras de 16 bits; el orden del byte y de la palabra es de más significativo a menos significativo. El orden para enviar los parámetros al LM62X corresponde al orden descendente mostrado en la descripción de la palabra de control, esto es, comenzando con el byte más significativo de la aceleración, siguiendo con la velocidad y terminando con el byte menos significativo de la posición.
La aceleración y velocidad son valores positivos de 32 bits, que van de 0 (00000000Hex) a 2E30-1 (3FFFFFFFHex). Los 16 bits bajos de la aceleración y velocidad se definen como la parte fraccionaría; por tanto el bit 16 de estos parámetros es el entero menos significativo (estando los bits numerados del 0 al 31). Para determinar el código para una velocidad deseada, se multiplica la velocidad deseada en pulsos/muestreo por 65,536 y se convierte la cantidad resultante en binario. Las unidades de la aceleración son en pulsos/muestreo/muestreo y su valor no debe exceder el valor cargado para la velocidad. La posición es un valor signado (en complemento a 2) de 32 bits, que va de –2E30 (C0000000Hex) a 2E30-1 (3FFFFFFFHex).
Los datos requeridos son escritos a *buffers* primarios en un esquema de *buffers* dobles; en el que el contenido de los primarios no es transferido a los secundarios (registros de trabajo) hasta que el comando STT es ejecutado. Lo anterior puede ser ventajoso, para enviar datos al *buffer* primario antes de actualizar los parámetros y así poder, por ejemplo, sincronizar el movimiento de varios ejes.

Comando **STT:** comenzar el control de movimiento
Código del comando: 01hex
Bytes de argumento: ninguno
Ejecutable en movimiento:Sí, si la aceleración no ha sido cambiada.

El comando STT es usado para ejecutar la trayectoria deseada, cuyos parámetros han sido cargados con el comando LTRJ. El control de varios ejes con movimiento simultaneo, puede ser logrado cargando primero los parámetros de trayectoria, y después ejecutar el comando STT para cada uno de los ejes al mismo tiempo. Este comando puede ser ejecutado en movimiento, a menos que la aceleración haya sido cambiada y no se haya ejecutado un paro manual. Si STT es ejecutada contraviniendo la advertencia anterior, se generará una interrupción por comando erróneo, y STT será ignorado.

Comandos de reporte de datos

Los siguientes siete comandos de usuario se usan para obtener información de varios de los registros del LM62X. Se reporta la información de status, posición y velocidad. Con excepción del comando RDSTAT, los

datos son leídos del puerto de datos del LM62X después de escribir el comando correspondiente al puerto de comandos.

Comando **RDSTAT** :leer el byte de status
Código del comando: ninguno
Bytes a leer: uno
Rango de los datos: ver el texto
Ejecutable en movimiento:Sí

El comando RDSTAT no es propiamente un comando, pues su ejecución está soportada directamente por hardware, es por ello que no tiene un código asociado. La lectura del status byte se solicita al llevar las líneas ¯CS, ¯PS, ¯RD a cero lógico. Ver la tabla 7 para conocer la definición de bits en el byte status.
TABLA 7. Localidades del Byte de estatus

Posición del Bit	**Función**
Bits 7	Motor apagado
Bit 6	*Breakpoint* alcanzado (INT)
Bit 5	Error de posición excesivo (INT)
Bit 4	Ocurrió desbordamiento (INT)
Bit 3	Pulso índice observado (INT)
Bit 2	Trayectoria completa (INT)
Bit 1	Comando de error (INT)
Bit 0	Bit "ocupado"

El bit 7 es la bandera de motor apagado y se lleva a 1 lógico cuando la salida de control del motor está en media escala (motor parado). El motor es apagado en las siguientes condiciones; reinicio por encendido, reinicio por ejecución del comando RESET, error de posición excesivo (si se ejecuto el comando LPES), o cuando el comando LTRJ es usado para parar "manualmente" el motor. Nótese que cuando el bit 7 es levantado en conjunción con el comando LTRJ para producir un paro manual del motor, el cambio del bit 7 no se efectúa sino hasta que el comando STT se ejecuta. El bit 7 se "limpia" al ejecutar el siguiente STT.
El bit 6 es la bandera de interrupción por *breakpoint* alcanzado, la cual sube cuando los comandos SBPA o SBPR han sido ejecutados y el *breakpoint* asociado ha sido excedido. La bandera opera independientemente del estado de las interrupciones. Este bit se limpia con el comando RSTI.
El bit 5 es la bandera de interrupción por error excesivo y se levanta al ser rebasado el error de posición programado con los comandos LPEI y LPES. La bandera opera independientemente del estado de las interrupciones. El bit 5 se limpia con el comando RSTI.
El bit 4 es la bandera de desborde de posición ocurre. Este desborde significa que el registro de posición excede su capacidad en el sentido positivo o negativo, y da una "vuelta" al contador. Este desborde es común en el modo de velocidad. Está bandera es útil para asegurar la integridad de la información de posición. La bandera opera independientemente del estado de las interrupciones. El bit 4 se limpia con el comando RSTI.
El bit 3 es la bandera de ocurrencia de pulso índice. Se levanta cuando el codificador de posición ha enviado un pulso de vuelta completa y en caso de que el comando SIP haya sido ejecutado se actualiza el registro índice. La bandera opera independientemente del estado de las interrupciones. El bit 3 se limpia con el comando RSTI.
El bit 2 es la bandera de trayectoria completa, y se levanta cuando la trayectoria cargada mediante el comando LTRJ e iniciada por el comando STT ha sido completada. Este bit es el OR lógico de los bits 7 y 10 del registro de señales, ver el comando RDSIGS más abajo. La bandera opera independientemente del estado de las interrupciones. El bit 2 se limpia con el comando RSTI.
El bit 1 es la bandera de comando erróneo, y es levantada cuando el usuario intenta hacer una lectura de datos cuando se espera una escritura, o viceversa. La bandera opera independientemente del estado de las interrupciones. El bit 1 se limpia con el comando RSTI.
El bit 0 es la bandera *busy* que indica si el LM62X está ocupado. En estado alto indica que el LM62X no puede atender ninguna petición de lectura o escritura, y cualquier comando enviado será ignorado. Cualquier lectura realizada con este bit alto, dará el valor de los *buffers* del LM62X que en general no serán el dato esperado. Tales lecturas/escrituras con bit 0 alto, no generarán una interrupción de comando erróneo.

El usuario debe verificar el estado bajo de este bit antes de intentar escribir/leer datos del LM62X y antes de enviar cualquier comando.

Comando **RDSIGS** :leer el registro de señales
Código del comando: 0CHex
Bytes a leer: 2
Rango de los datos: ver el texto
Ejecutable en movimiento:Sí

El registro de señales "internas " del LM62X puede ser leído utilizando este comando. El primer byte leído es el más significativo el byte menos significativo de este registro es una réplica del byte de status (con excepción del bit 0).Ver Tabla 8

Tabla 8 definición de los bits del registro de señales

Posición de Bit	**Función**
Bit 15	Interrupción Host
Bit 14	Aceleración cargada (no actualizada)
Bit 13	UDF ejecutado (filtro no actualizado)
Bit 12	Hacia delante
Bit 11	Modo de velocidad
Bit 10	En objetivo
Bit 9	Apagar hasta error de posición excedida
Bit 8	Modo de salida octavo bit
Bit 7	Motor apagado
Bit 6	*Breakpoint* alcanzado (INT)
Bit 5	Error de posición excesivo (INT)
Bit 4	Ocurrió desbordamiento (INT)
Bit 3	Pulso índice observado (INT)
Bit 2	Trayectoria completa (INT)
Bit 1	Comando de error (INT)
Bit 0	Adquirir próximo índice (SIP ejecutado)

El bit 15 es la bandera de interrupción al procesador anfitrión y se levanta mientras la interrupción al anfitrión (pin 17) este activa (nivel alto). El pin 17 va estado alto si ocurre cualquiera de las 6 condiciones de interrupción (siempre y cuando tales interrupciones no estén enmascaradas). El bit 15 (y el pin 17) se limpian ejecutando el comando RSTI.
El bit 14 es la bandera que se levanta cuando una aceleración ha sido cargada a *buffer,* pero no ha sido actualizada. Este bandera se baja ejecutando el comando STT.
El bit 13 es la bandera que se levanta al realizar el comando UDF. Tal bandera es bajada al término del primer periodo de muestreo en el que fue ejecutado UDF, por tanto es una bandera de muy baja duración.
El bit 12 es la bandera que se levanta si la dirección de movimiento es aquella definida como positiva.
El bit 11 es la bandera que se levanta si el LM62X está en el modo de operación de velocidad, y se baja si está en el modo de operación de posición.
El bit 10 es la bandera que se levanta al completarse la última trayectoria iniciada por el comando STT. Esta bandera baja cuando el próximo comando STT es ejecutado.
El bit 9 es la bandera que se levanta cuando se ha originado un paro debido a un sobrepaso de error de posición si se ejecuto el comando LPES. Esta bandera se baja por medio del comando LPEI.
El bit 8 es la bandera que se levanta cuando el LM62X se ha reiniciado o cuando la salida a DAC programada es de 8 bits. El comando PORT12 baja está bandera.
Los bits 0 al 7 son una réplica del byte de status, con excepción del bit 0 que se levanta cuando el comando SIP es ejecutado.

Comando **RDIP**: leer posición del índice
Código del comando: 09Hex
Bytes a leer: 4
Rango de los datos: C0000000 a 3FFFFFFFHex
Ejecutable en movimiento:Sí

Este comando lee la posición grabada en el registro índice. Leer el registro índice puede ser parte de un sistema de verificación de error: siempre que el comando SIP es ejecutado, la nueva posición índice puede restarse de la anterior y el resultado ser dividido por la resolución del codificador incremental. El resultado siempre debe ser un número entero. El comando RDIP facilita la adquisición los datos necesarios para los cálculos descritos anteriormente. El comando también puede ser usado para identificar/definir la posición *Home* o alguna otra posición especial. Los bytes se leen en orden de mayor a menor orden.

Comando **RDDP**: leer la posición deseada.
Código del comando: 08Hex
Bytes a leer: 4
Rango de los datos: C0000000 a 3FFFFFFFHex
Ejecutable en movimiento:Sí

Este comando lee la posición deseada instantánea calculada por el generador de perfiles de velocidad. Esta posición constituye el valor de referencia (*setpoint*) instantáneo para el nodo de suma del lazo de control.

Comando **RDRP**: leer posición actual
Código del comando: 0AHex
Bytes a leer: 4
Rango de los datos: C0000000 a 3FFFFFFFHex
Ejecutable en movimiento:Sí

Este comando reporta la posición real absoluta del eje del motor. Esta posición es el valor retroalimentado al nodo de suma del lazo de control.

Comando **RDDV** : leer velocidad deseada
Código del comando: 07Hex
Bytes a leer: 4
Rango de los datos: C0000001 a 3FFFFFFFHex
Ejecutable en movimiento:Sí

Este comando lee la fracción entera y fraccionaría de la velocidad instantánea deseada, la cual es usada para generar el perfil de velocidad deseado. Los bytes se leen de mayor a menor orden. Los datos que proporciona este comando están propiamente escalados para comparación numérica con la velocidad programada por el usuario; sin embargo, debido a que los dos bytes menos significativos representan la parte fraccionaría, sólo los dos bytes superiores serían apropiados para la comparación con los datos obtenidos mediante el comando RDRV (ver más abajo). Nótese también que pese a que las cantidades programadas de velocidad se dan sólo en números positivos, los datos regresados por el comando RDDV representan la magnitud y sentido de la velocidad, por lo que pueden tomar valores "negativos".

Comando **RDRV** : leer velocidad real
Código del comando: 0BHex
Bytes a leer: 4
Rango de los datos: C0000 a 3FFFHex
Ejecutable en movimiento:Sí Este comando lee la parte entera de la velocidad instantánea real del motor. La parte fraccionaría de la velocidad no se reporta porque los datos reportados se derivan de la información obtenida del codificador de posición, el cual sólo maneja pulsos enteros. Por tanto para ser comparados con

los datos regresados por RDDV, estos datos deben ser recorridos a la izquierda 16 posiciones. Aplica la misma aclaración de las velocidades negativas que para la instrucción RDDV.

Comando RDSUM : leer término integral
Código del comando: 0DHex
Bytes a leer: 2
Rango de los datos: 00000Hex a ± el valor actual del límite integral actual
Ejecutable en movimiento: Sí

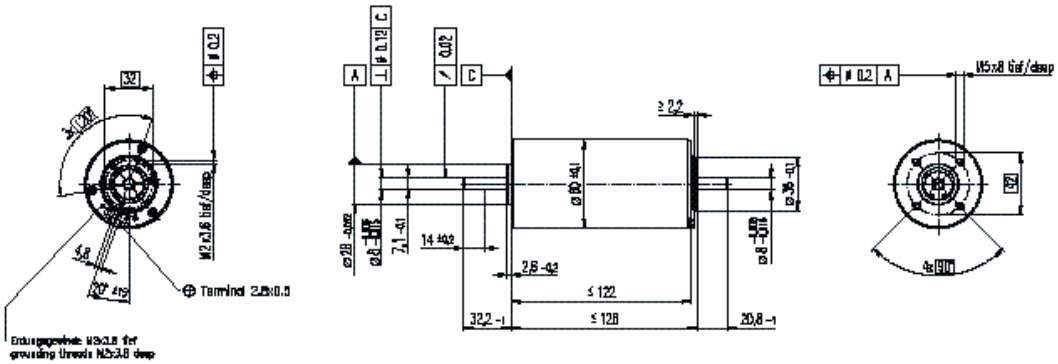
Este comando lee el valor que se ha acumulado en el término integral. La capacidad de leer este valor, puede ser útil para iniciar o afinar un sistema adaptativo.

NOTA FINAL: Para consultar un diagrama de dimensiones físicas de los chips LM62X consultar el PDF de nombre LM628/LM629: *Precition Motor Controller* de *National Semiconductors.* También se podrá consultar algunas configuraciones típicas de conexión del LM62X.

MOTOR MAXON

F 2260 ∅60 mm, Graphite Brushes, 80 Watt

maxon DC motor

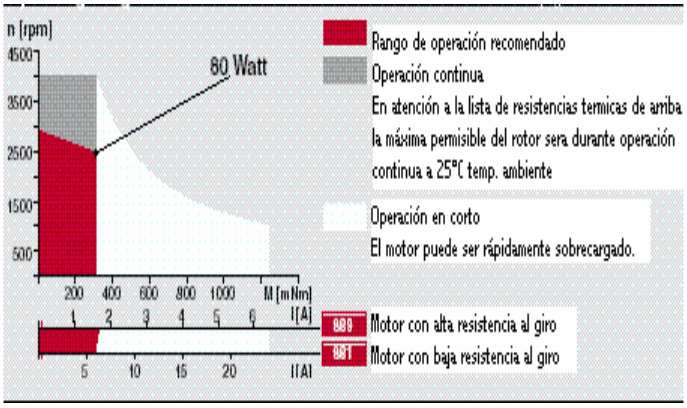


DATOS DEL MOTOR	UNIDADES	VALOR
Rango de potencia asignado	W	40
Voltaje nominal	Volt	24.0
Velocidad sin carga	Rpm	3660
Torca de paro	Nm	0.758
Gradiente velocidad/torca	Rpm/mMm	5.01
Corriente sin carga	MA	225
Corriente de arranque	A	12.6
Resistencia terminal	Ohm	1.91
Velocidad máxima permisible	Rpm	5000
Corriente máxima continua	A	2.16
Torca máxima continua	mNm	130
Potencia máxima de salida a voltaje nominal	W	70.4
Eficiencia máxima	%	73
Constante de torca	mNm/A	60.3
Constante de velocidad	Rpm/V	158
Constante de tiempo mecánica	Ms	35
Inercia del rotor	gcm^2	665
Inductancia terminal	mH	0.63
Resistencia térmica casco-ambiente	K/W	5.0
Resistencia térmica rotor-casco	K/W	2.4
Constante térmica de tiempo	s	76

ESPECIFICACIONES

- Juego de ejes con carga <15N ≤0.1mm
  >15N 0.1-0.5mm
  Juego d ejes para combinaciones de motor con codificaor esta limitado al máximo 0.15mm
- Cojinetes precargados
  Solidez sin carga mínima 15N
- Máxima carga en los cojinetes
  (dinámica) 15N
  radial(5mm) 100N
  Presionado (estático) 400N
  (estático, con flecha suportada) 10000N
- Juego radal de cojinetes 0.05mm
- Rango de temperatura ambiente -20/+100°C
- Temperatura máxima del rotor +125°C
- Número de segmentos del conmutador 26
- Peso del motor 790g
- Los valores listados en la tabla son nominales.

RANGO DE OPERACIÓN.



TARJETA 4I27

TABLA DE CONTENIDO

PRECAUCIONES

RIESGOS

Los servo-motores grandes son capaces de inflingir serios daños a personas y mecanismos asociados con el servo-sistema. En suma, algunos motores usan voltajes de alimentación potencialmente letales.

Cuando un servo-control es configurado por primera vez, un comportamiento impredecible debe ser ESPERADO. La primera vez que se verifique la operación básica del servo –sistema, debe ser verificado con las terminales de conexión de alimentación desconectadas.

Nunca se debe depender de un comando de software para que la 4I27A deshabilite el motor cuando usted u otros podrían estar expuestos a riesgo de que el motor arranque inesperadamente. La alimentación del motor debe ser siempre removida cuando se trabaje en las partes mecánicas del servo-sistema.

ELECTRICIDAD ESTÁTICA

Los circuitos integrados CMOS en el 4I27A pueden ser dañado por exposición a descargas electrostáticas. Las siguientes precauciones deben ser tomadas cuando se maneja el 4I27A para prevenir los daños.

A. Deje el 4i27A en una bolsa antiestática mientras no se necesite.
B. Todo el trabajo debe ser llevado a cabo en una estación antiestática.
C. Equipo de tierra debe ser instalado
D. El manejo debe hacerlo una persona con un brazalete conductivo a través de una resistencia de 1 megaohm a tierra.

INTRODUCCIÓN

GENERAL

El 4I27A es de bajo costo, basado en 2 canales con LM628 DC un sistema de servo-motor control implementado en una tarjeta apilable en un PC/104. El 4I27A est diseñado para un alto desempeño en sistemas de posicionamiento usando servo-motores de DC con codificadores en cuadratura.

La salida por cada canal del 4I27 es de ±10 volts, señal analógica que puede ser

entrada directa para servo-amplificadores.

Las entradas al 4I27A de las señales del codificador en cuadratura y su índice son RS-422. Esto es deseable cuando se requieren cables largos para el codificador.

Las señales de control por cada canal incluyen 3 bits auxiliares I/O. Estos bits I/O pueden ser usados para detectar la temperatura de apagado y la habilitación del servo-amplificador. 8 bits I/O de propósito general están disponibles para cualquier aplicación.

Los LM628's usados en el 4I27a son procesadores digitales de alto desempeño, específicamente diseñados para control de movimiento. El LM628 puede ejecutar una secuencia de movimiento de rampa de subida, mantenerse y rampa de bajada, sin intervención del procesador anfitrión.

Se pueden generar interrupciones y terminar el movimiento, puntos de referencia, pulso de índice, o en respuesta a varias condiciones de error. Las interrupciones están "OR'eadas" en la tarjeta 4I27A, así que se usa un solo sistema de interrupciones. La línea IRQ usada puede ser seleccionada por software desde cualquiera de las 11 interrupciones del bus AT disponibles.

Un filtro digital PID es usado para establecer los parámetros del lazo de retroalimentación para estabilidad y optimo desempeño. Velocidad, posición de objetivo y parámetros del filtro pueden ser cambiados durante el movimiento. La velocidad del reloj del LM628 puede ser reducida para ajustarse a motores grandes que requieren constantes de tiempo mas grandes.

El software de demostración incluye ejemplos de operación en modo de posición para dos ejes, operación en modo de velocidad, y un filtro simple sintonizado que permite la modificación dinámica de los coeficientes del filtro mientras se provee una grafica de la respuesta al escalón del servo-sistema. El código fuente esta provisto para todo el software de demostración.

Una versión del 4I27A para PWM está disponible.

CONFIGURACIÓN DE HARDWARE

GENERAL

El 4I27A tiene sólo tres opciones de configuración de hardware, las direcciones de los puertos I/O, la velocidad del reloj del LM628, y la habilitación de los bits de estado de encendido de los motores. Estas opciones son seleccionadas pon *jumpers* colocados en tres cabezales de pines. En las siguientes partes, cuando se mencionen las palabras "arriba", "abajo", "derecha", e "izquierda"; se asumirá que la tarjeta 4I27 esta orientada con su bus de conexión J1 y J2 en la parte baja de la tarjeta (cerca de la persona que esta haciendo la configuración).

CONFIGURACIÓN PREESTABLECIDA

La tarjeta 4I27A está configurada de la siguiente manera cuando es embarcada de la fábrica.

FUNCIÓN	PREESTABLECIDO	JUMPERS	POSICIÓN
Direcciones del puerto	0200H	W1,W2,W3	abajo, abajo, abajo
Reloj del LM628	8MHz	W4	izquierda
Habilitación del motor	Asegurado	W5	Arriba

DIRECCIONES DEL PUERTO

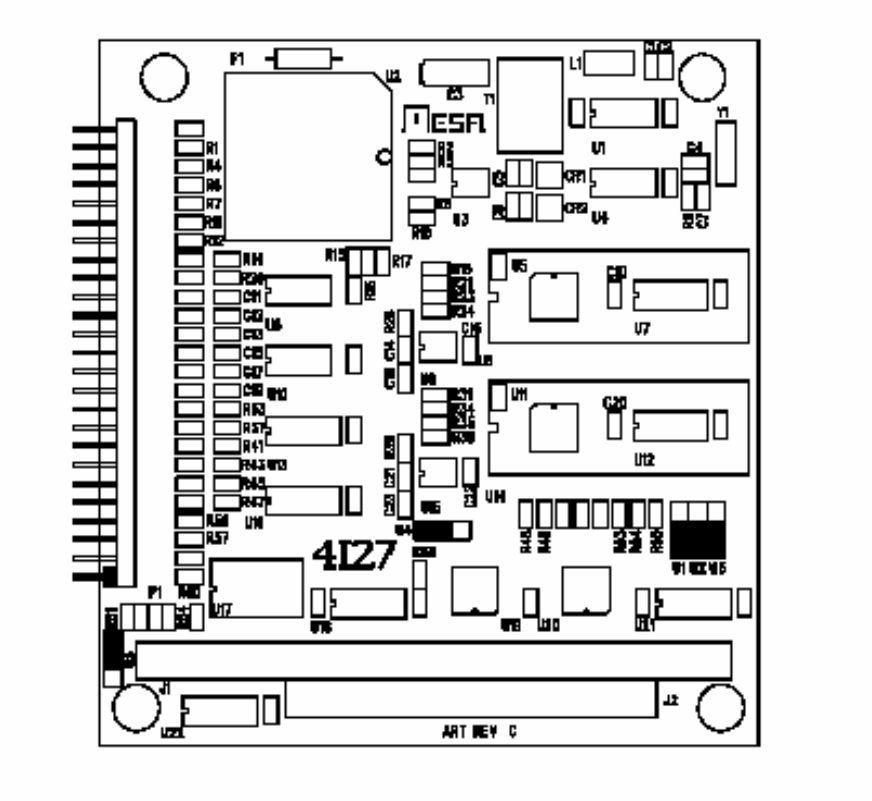
El 4I27A ocupa 16 localidades continuas en el espacio I/O. La dirección base de la tarjeta 4I27A puede ser cualquiera de las ocho localidades diferentes. Estas localidades son seleccionadas con *jumpers* colocados en los bloques W1, W2, y W3.

La tabla siguiente muestra la base de direcciones I/O seleccionadas por los *jumpers*.

DIRECCIÓN	W1	W2	W3
0200H	abajo	abajo	abajo
0210H	abajo	abajo	arriba
0220H	abajo	arriba	abajo
0230H	abajo	arriba	arriba
0240H	arriba	abajo	abajo
0250H	arriba	abajo	arriba
0260H	arriba	arriba	abajo
0270H	arriba	arriba	arriba

CONFIGURACIÓN DE HARDWARE

DIRECCIONES DEL PUERTO, RELOJ Y *JUMPERS* DE HABILITACIÒN DE LOS MOTORES.



CONFIGURACIÓN DE HARDWARE

RELOJ DEL LM628

El LM628 chip de control de motores en el 4I27A usa normalmente un reloj de 8MHz. Este reloj pude ser ajustado a 4MHz si se desea. La máxima tasa de entrada en cuadratura puede ser reducida a 500kHz cuando se selecciona la opción de 4MHz. Moviendo W4 a la posición izquierda se establece el reloj del LM628 a 8MHz. Poniendo W4 a la posición derecha se establece el reloj del LM628 a 4MHz

POLARIDAD PRESTABLECIDA DE HABILITACION DEL MOTOR

Los bits de habilitación del motor en el puerto paralelo A pueden ser establecidos como asegurada o libre en encendido y en el reestablecimiento del sistema. Estos bits pueden ser usados para deshabilitar un servo-amplificador externo en el encendido y prevenir movimientos del motor indeseados.

El puerto paralelo revierte a un estado de entrada cuando se inicializa, así el estatus de estos pines es determinado enteramente por las resistencias de encendido o apagado.

Estableciendo W5 en la posición de arriba se conecta las resistencias a VCC, de esta manera estas señales pueden ser altas para el sistema de reinicio. Este es la conexión preestablecida. Poniendo W5 en la posición de abajo pondrá los bits de habilitación del motor en bajo. Note que estos son resistores de 3.3K y son manejados por entradas CMOS hasta que el puerto paralelo es programado en modo de salida.

CONECTOR DE INTERFASE

GENERAL

El conector de interfase I/O en el 4I27A es un cabezal de 50 pines. El conector sugerido es el AMP PN 1-746285-0. El 4I27A usa entradas de codificador RS-422

ORIENTACIÓN DEL CONECTOR I/O

El conector de 50 pines en el 4I27A tiene el pin uno marcado con un cuadro blanco en la tarjeta. Este corresponde con la franja roja o ensamble típico del cable plano. Si es deseada una polarización mas positiva, los cabezales centrales polarizados IDC deben ser usados. Estos conectores no serán completamente compatibles en la tarjeta 4I27A si se instala hacia atrás. Una sugerencia de conector de 50 pines polarizado al centro con cabezal IDC es el AMP PN 1-746285-0.

INTERFASE DEL CONECTOR DE SALIDA

PIN #	SEÑAL	PIN #	SEÑAL
1	MOTOR1Q B	2	MOTOR1/QB
3	MOTORIQA	4	MOTORI/QA
5	MOTOR0QB	6	MOTOR0/QB
7	MOTOR0QA	8	MOTORO/QA
9	MOTOR1IDX	10	MOTORI/IDX
11	MOTOR0IDX	12	MOTOR0/IDX
13	MOTOR1AOUT	15	MOTOR0AOUT
17	NC	19	NC
21	/MOTOR1ENA	23	/MOTOR0ENA
25	MOTOR1SENSE1	27	MOTOR0SENSE1
29	MOTOR1SENSE0	31	MOTOROSENSE0
33	BIT7	35	BIT6
37	BIT5	39	BIT4
41	BIT3	43	BIT2
45	BIT1	47	BIT0
49	+5 volt power		

Los pines pares del 14 hasta 50 son tierra. Los pares deiferenciales RS-422 son pines nones y pares sucesivos, por ejemplo: MOTOR0QB y MOTOR0/QB (pines 5 y 6 respectivamente) comprenden una pareja diferencial RS-422. Todas las entradas RS-422 en el 4I27A están terminadas en la tarjeta 4I27A con resistencias de 136 ohms.

INSTALACION

GENERAL

Cuando la tarjeta 4I27A ha sido debidamente configurado para su aplicación, ésta puede ser insertada dentro de un conector PC/104. Los seguros deben de ser ajustados para asegurar el 4I27A en su lugar.

OPERACION

MAPA DE REGISTROS

El 4I27A ocupa 16 localidades contiguas de puertos I/O , empezando con la dirección base seleccionada por el usuario.

BASE +00H	92C55 Puerto A	(debe ser salida)
BASE +01H	82C55 Puerto B	(debe ser entrada)
BASE +02H	82C55 Puerto C	(definida por usuario)
BASE +03H	82C55 Puerto de control	
BASE +04H	Motor 0 Puerto Command/Status	
BASE +05H	Motor 0 Puerto de datos	
BASE +06H	Motor 0 Puerto Command/Status	(duplicado)
BASE +07H	Motor 0 Puerto de datos	(duplicado)
BASE +08H	Motor 1 Puerto Command/Status	
BASE +09H	Motor 1 Puerto de datos	
BASE +0AH	Motor 1 Puerto Command/Status	(duplicado)
BASE +0BH	Motor 1 Puerto de datos	(duplicado)
BASE +0CH	Motor 0 y Motor 1 Puerto Command/Status	
BASE +0DH	Motor 0 y Motor 1 Puerto de datos	
BASE +0EH	Motor 0 y Motor 1 Puerto Command/Status (duplicado)	
BASE +0FH	Motor 0 y Motor 1 Puerto de datos	(duplicado)

PUERTO PARALELO

El puerto paralelo 82C55 se proporciona para manipular varios bits del control del motor y para proveer entrada y salida para 8 bits de propósito general. Los bits de control del motor pueden ser usados para cualquier propósito. Los bits de control del motor son activos bajo, y tienen resistencias de *pull-up*, de esta manera están inactivos al encender. Todos los bits no mencionados no son usados y deben ser puestos en estado alto.

OPERACION

BITS DEL PUERTO PARALELO

PUERTO 82C55	BIT	PIN	FUNCION
A	0	23	/Motor 0 habilitación (bit de propósito general)
A	1	21	/Motor 1 habilitación (bit de propósito general)
A	2	N/A	ISEL0
A	3	N/A	ISEL1
A	4	N/A	ISEL2
A	5	N/A	ISEL3
A	6	N/A	IMODE0
A	7	N/A	IMODE1
B	0	31	Motor 0 sense (bit de propósito general. BPG)
B	1	27	/Motor 0 sobre temperatura (BPG)
B	2	29	Motor 1 sense (BPG)
B	3	25	/Motor 1 sobre temperatura (BPG)
B	4	11	Motor 0 índice (también conectado al LM628)
B	5	9	Motor 1 índice (también conectado al LM628)
B	6	N/A	Motor 0 INT status (Rev. C o tarjeta solamente)
B	7	N/A	Motor 1 INT status (Rev. C o tarjeta solamente)
C	0	47	Bit 0 (BPG)
C	1	45	Bit1 (BPG)
C	2	43	Bit2 (BPG)
C	3	41	Bit3 (BPG)
C	4	39	Bit4 (BPG)
C	5	37	Bit5 (BPG)
C	6	35	Bit6 (BPG)
C	7	33	Bit7 (BPG)

OPERACION

OPERACION DE PUERTO PARALELO

Los puertos paralelos en e 4I27A usan un PIA 82C55 estándar. Las líneas de acceso I/O tienen resistencias de 3.3 k para simplificar el contacto de terminación de la interfase, opto-detector o salida de colector abierto.

Los puertos A y B tienen funciones predefinidas, mientras que el puerto C esta disponible para propósito general. El puerto C tiene la característica de ser separado en dos puertos de 4 bits para permitir tanto bit de salida como de entrada en el mismo puerto.

El puerto A es usado como puerto de salida. Este controla los bits de /MOTOR0ENA y /MOTOR1ENA, los bits ISEL que determinan la línea que maneja la IRQ para el 4I27A, y los bits de IMODE que controlan la habilitación de las interrupciones y su modo.

El *jumper* W5 determina si /MOTOR0ENA y /MOTOR1ENA, están asegurados o libres. Antes de establecer al puerto A en el modo de salida, el puerto A puede ser leído para determinar el estado de "apagado" de estos pines.

El Puerto B es usado como puerto de entrada, los bits de MOTOR0SENSE, MOTOR1SENSE, MOTOR0TEMP, MOTOR1TEMP, MOTOR0IDX, MOTOR1IDX, INTO, y INT1 son leídos aquí. El código fuente en Pascal en el archivo 4I27LOW.PAS tiene ejemplos de el establecimiento y uso predefinido del puerto paralelo I/O en el 4I27A.

SELECCIÓN DE INTERRUPCION

Cuando la tarjeta 4I27A usa interrupciones, la línea de IRQ especifica manejada por el 4i27A es determinada por los bits ISEL del puerto A. Los cuatro bits ISEL forman un código binario que corresponde directamente con la línea IRQ seleccionada. Dado que hay otros bits en el 92C55 puerto A, siempre se deberá cambiar los bits ISEL leyendo el puerto, modificando los bits deseados y sobrescribiendo en el puerto. El código fuente en pascal en el archivo 4I27LOW.PAS tiene ejemplos de establecimiento de línea IRQ.

MODO DE INTERRUPCION

Los dos bits IMODE en el 82C55 puerto A determinan el modo de interrupción del 4I27A. Los bits IMODS funcionan de la manera siguiente:

IMODE1	IMODE0	FUNCTION
0	0	Interrupción deshabilitada.
0	1	Modo normal de interrupción.
1	0	Modo compartido de interrupción
1	1	Modo compartido de interrupción + asegurado

El modo compartido de interrupción usa un contolador pullup-activo/pulldown-pasivo que permite compartir las interrupciones entre las tarjetas. La interrupción puede ser

únicamente compartida con otras tarjetas que soporten este modo de interrupciones compartidas.

OPERACION

OPERACIÓN DE INTERRUPCIONES

Las salidas de interrupción en el LM628 están lógicamente "OR'eadas" por la tarjeta 4I27A. Esto significa que una salida de interrupción desde cualquiera de los LM628 puede causar una interrupción en el sistema. 4I27 con Revisión C o tarjetas PC mas grandes pueden leer el estatus de la interrupción de ambos LM628 a través del puerto B. Esto puede ser usado por la rutina de servicio de interrupción para determinar rápidamente cual canal causo la interrupción.

El bus de interrupción es manejado por un *buffer* tri-state controlado por los bits IMODE. Los bits IMODE estarán en estado "bajo" hasta que el 82C55 es programado. Si las interrupciones no son usadas, los bits de IMODE deberán ser "bajos".

CONECTANDO LOS MOTORES

Después de que el 4I27A ha sido configurado e instalado en el sistema, se debe verificar su correcta operación. Los programas 4I27READ, y 4I27TUNE pueden ser usados como verificación rápida de la funcionabilidad del 4I27A.

El primer paso es conectar al 4I27A los codificadores en cuadratura y entonces usar 4I27READ para verificar su funcionabilidad y apropiada dirección de cuenta.

El programa 4I27READ simplemente imprime constantemente la posición de ambos motores (relativa a la posición del programa al inicializar) hasta que se presiona una tecla. Cuando la interfase RS-422 es usada por los codificadores, se debe asegurar no mezclar las salidas A y B.

Para el siguiente paso, es sugerido que los cables de alimentación del motor se dejen desconectados y entonces correr 4I27TUNE. Cuando se corre el 4I27TUNE, se verificará la presencia de la tarjeta 4I27A, y la habilitación del Motor0. Motor 0 será configurado para el modo de posición con la posición inicial puesta en donde se encuentre ubicado Motor0 durante la inicialización.

Las salidas analógicas del 4I27A pueden ser monitoreadas mientras el motor es "movido" manualmente atrás y adelante de su posición original. Si la operación del 4I27A es correcta, las salidas analógicas deberán variar con la posición del eje del codificador de posición.

Los motores deberán ser conectados de tal forma que el manejador del motor tienda a oponerse al movimiento manual del motor.

OPERACION

SINTONIZACIÓN DE SERVO-SISTEMA

Un sencillo programa de sintonización, 4I27ATUNE, es proporcionado con el 4I27A. Este programa permite el ajuste manual de los parámetros del filtro digital, mientras el motor seleccionado está en operación. Además, 4I27TUNE puede grabar la respuesta al escalón del servo-sistema con los parámetros actuales del filtro. 4I27TUNE usa las flechas del teclado para escoger y ajustar los parámetros deseados. Los parámetros seleccionables son los siguientes.

PARÁMETRO	DESCRIPCIÓN	RANGO	
KP	Ganancia	0 a 32767	drive/err
KD	Derivativo	0 a 32767	drive/err(n)-error(n-1)
KI	Integral	0 a 32767	drive/acum..error
IL	Límite integral	0 a 32767	KI límite
SI	Intervalo de muestreo	0 a255	SP/(SI+1)
STEP SIZE	Paso del motor	100 a 10000	Pulsos codificador
VELOCITY	Velocidad	0 a 1073741823	Pulsos/SP
ACCELERATION	Aceleración	0 a 1073741823	Pulsos/SP/SP
MOTOR	Selecciona motor	0 y 1	
TIME/DIV	Selecciona gráfica	10 a200	milisegundos/división

SP = PERIODO DE MUESTREO = 256μseg*(1+SI) con reloj de 8MHz

La siguiente tabla muestra las teclas de funciones del 4I27TUNE.

TECLA	FUNCIÓN
FLECHA ARRIBA	Selecciona parámetro
FLECHA ABAJO	Selecciona parámetro
FLECHA DERECHA	Ajusta parámetro arriba (+10%)
FLECHA IZQUIERDA	Ajusta parámetro abajo (-10%)
PAGINA ABAJO	Ajusta parámetro arriba (+1%)
FIN	Ajusta parámetro abajo (-1%)
PAGINA ARRIBA	Dibuja escala horizontal de tiempo
INSERTA	Realiza y guarda la respuesta al escalón
BORRAR	Limpia gráfica
CTRL.-HOME	Reestablece parámetros a valores iniciales

ALT X	Salir

OPERACION

PROGRAMAS DE DEMOSTRACIÓN

4i27VELO establece al motor 0 y motor 1 en el modo de velocidad y permite establecer su velocidad en RPM.

4I27POS1 establece al motor 0 en modo de posición, y hace algunos movimientos de rampa arriba / rampa abajo. 4I27POS2 establece al motor 0 y 1 en el modo de posición, y realiza movimientos coordinados (XY) con ellos. 4I27POS2 aborta si ambos motores no están presentes.

Los programas 4I27VELO y 4I27POS pueden requerir que los parámetros de filtro y otras constantes preestablecidos sean cambiadas antes de ser ejecutados, para ajustarse a los motores usados y a la localización (registros) asignada a la tarjeta 4i27.

4I27TUNE es un programa manual de sintonización de los parámetros que permite optimizar los parámetros del filtro mientras el motor seleccionado está en operación. 4I27TUNE requiere un adaptador de video EGA, VGA o compatible con Hercules.

Todos los programas de demostración están escritos en Turbo Pascal. El código fuente de los programas de demostración se encuentra en el directorio fuente del disco de distribución del 4I27A.

INFORMACIÓN DE REFERENCIA

ESPECIFICACIONES

	MIN	MAX	UNIDAD	NOTAS
ALIMENTACIÓN:				
Voltaje	4.5	5.5	V	
Corriente de alimentación	---	125	mA	(sin carga ext)
CARGA DEL BUS:				
Capacitancia de entrada	---	19	pF	
Corriente de fuga de entrada	---	5	µA	
Capacitancia de salida150	---	pF		
Corriente de salida casco	12	---	mA	
VALORES DEL PUERTO I/O:				
"Bajo" lógico de entrada	-0.3	0.8	V	
"Alto" lógico de entrada	2.0	5.5	V	
"Bajo" de salida	---	0.4	V	2.5 mA sink
"Alto" de entrada	3.0	---	V	2.5 mA source
SALIDA ANALÓGICA:				
Voltaje de salida	-10	+10	V	5k Ohm carga.
AMBIENTALES:				
Rango de temperaturas de operación				
-I versión	-40	+85	ºC	
-C versión	0	+70	ºC	
Humedad relativa	0	90	%	Sin condensación

MANUAL DEL 7I25 PUENTE H

TABLA DE CONTENIDO

SEGURIDAD

Servo-motores grandes son capaces de infligir serios daños a gente y mecanismos asociados con el servo-sistema. En suma, algunos motores usan voltajes de alimentación potencialmente letales.

Cuando el control de un servo-sistema es configurado por primera vez, un comportamiento impredecible debe de ser ESPERADO. La primera vez que se verifique las operaciones básicas (como la dirección del motor) debe ser verificado con el motor desconectado.

Nunca se dependa de comandos de software para deshabilitar el motor cuando se encuentre expuesta alguna persona a riesgo de que el motor comience inesperadamente su operación. La alimentación del motor debe ser removida siempre que se este trabajando con partes mecánicas del servo-sistema.

PRECAUCIONES DE MANEJO

ELECTRICIDAD ESTÁTICA.

Los circuitos integrados CMOS en el 7I25 pueden resultar dañados por exposición a descargas electrostáticas. Las siguientes precauciones deben de ser tomadas cuando se maneja el 7I25 para prevenir posible daño.

A. Deje el 7I25 en su bolsa antiestática hasta que vaya a ser utilizado.
B. Todo trabajo debe ser realizado en una estación de trabajo antiestática.
C. El equipo donde se instalará el 7I25 debe estar aterrizado.
D. El personal que se encargue de su manejo debe tener un brazalete conductivo aterrizado con una resistencia de 1 M ohm.
E. Evite utilizar textiles sintéticos, en particular el Nylon.

INTRODUCCIÓN

El 7I25 tiene dos canales, maneja 150 watts por canal, diseñado para uso con las tarjetas controladoras de servo-motores 6I27, 4I27, 4i34, o 7I60. El 7I25 proveerá hasta 3 amperes por canal con voltajes desde 12 hasta 48 VDC.

Los bits del controlador para habilitación del motor permanecerán deshabilitados hasta que sean habilitados por software.

El 7I25 tiene una terminal de cable plano de 50 conductores que viene del 6I27 y cuenta con cabezales para conectar codificadores en cuadratura. También tiene cabezales para conectar el MOTO0RSENSE y MOTOR1SENSE. Estas entradas son útiles para interruptores de límite u opto-interruptores.

Los circuitos integrados del puente H usados en el 7I25 pueden generar advertencias de sobre temperatura (a 145°C) que pueden ser leidas por el 6I27. El apagado térmico (a 171°C) previene daños en los circuitos integrados del puente H en caso de una sobrecarga térmica prolongada.

Los LEDs en el 7I25 dan una rápida indicación de la polaridad y magnitud de alimentación que se proporciona al motor. Esto puede simplificar los establecimientos iniciales y la solución de problemas.

GENERAL

El 7I25 tiene un *jumper* configurable que debe ser propiamente ubicado para empatar la aplicación. Cuando las palabras "derecha" e "izquierda" sean usadas, se asumirá que la tarjeta del puente H 7I25 esta orientada con su bloque terminal apuntando hacia la persona que hace la configuración.

CONFIGURACIÓN EN PARO

El 7I25 puede ser configurado para dejar las terminales del motor en corto o dejarlas abiertas cuando no está en operación (/MOTORENA0 o /MOTORENA1 están inactivos). El *jumper* W1 establece el modo en paro para el motor 0 y el *jumper* W2 establece el modo en paro para el motor 1. Cuando un *jumper* está colocado en la posición izquierda, el motor correspondiente estará en corto, cuando el *jumper* está colocado en la posición derecha, el motor estará abierto.

NOTA. Revisión: Los tarjetas 7I25s pueden ser operados sólo en el modo de corto cuando están inactivos. Estas tarjetas tiene los jumpers soldados.

CONECCIÓN DEL CONTROLADOR.

El 7I25 esta hecho para operar con los controladores de servo-motores 6I27, 4I27, 4I34, o 7I60. El 7I25 se conecta al controlador con un cable plano de 50 conductores.

El largo máximo sugerido del cable es de 5 pies. Debe asegurarse que el pin uno del controlador quede conectado al pin uno del 7I25. El pin uno del 7I25 esta en el lado izquierdo de P3 cuando el 7I25 esta orientado en su terminal de mando apuntando hacia usted. Se sugiere usar cabezales hembra polarizados al centro de 50 pines (AMP-PN – 49958-2) para eliminar la posibilidad de conectar mal.

Un cabezal de 5 pines es proveído para cada uno de los codificadores en cuadratura. Un cabezal de 3 pines es provisto para cada entrada MOTORSENSE.

VERIFICACIÓN

Se surgiere que la primera vez que se pruebe el 7I25 sea con una fuente de alimentación limitada en corriente. Una sencilla prueba pude ser completada por medio del programa 6I27TUNE o 4I27TUNE proporcionado con el 6I27 o 4I27.

Primero conecte el 7I25 al controlador con el cable plano del largo apropiado. Después conecte los codificadores del motor al 7I25. Al ejecutar el programa 6I27TUNE o 4I27TUNE se habilitará el puente H para el motor 0, y se habilitará el sevo-control en modo de posición.

Rotar la flecha del motor adelante y atrás deberá causar que el LED del motor 0 se torne rojo o verde con intensidad variante dependiendo de la posición de la flecha.

CONECTOR DE INTERFASE DE SALIDA

PIN #	SEÑAL	PIN#	SEÑAL
1	MOTOR1A (codificador)	25	/MOTOR1 DRIVER SOBRETEMP.
3	MOTOR1B (codificador)	27	/MOTOR0 DRIVER SOBRETEMP.
5	MOTOR0B (codificador)	29	MOTOR1SENSE
7	MOTOR0A (codificador)	31	MOTOR0SENSE
9	MOTOR1IDX	33	BIT7
11	MOTOR0IDX	35	BIT6
13	MOTOR1PWM	37	BIT5
15	MOTOR0PWM	39	BIT4
17	MOTOR1DIR	41	BIT3
19	MOTOR0DIR	43	BIT2
21	/MOTOR1ENA	45	BIT1
23	/MOTOR0ENA	47	BIT0
		49	+5V alimentación.

Todos los pines no numerados están conectados a tierra.
NOTA: el pin 1 está en la parte de arriba del conector.

CONECTOR DEL CODIFICADOR DE SALIDA

La conexión de los codificadores en cuadratura es hecha con conectores de 5 pines P1 y P5. La disposición de los pines es la siguiente.

MOTOR 0 - PI		MOTOR1 – P5	
PIN #	SEÑAL	PIN#	SEÑAL
1	TIERRA	1	TIERRA
2	MOTOR0IDX	2	MOTOR1IDX
3	MOTOR0A	3	MOTOR1A
4	+5V	4	+5V
5	MOTOR0B	5	MOTOR1B

CONECTOR DE SALIDA MOTORSENSE

Dos bits de entrada en del controlador están terminados en el 7I25. Estas entradas están disponibles en cabezales de 3 pines P2 y P4.

MOTOR 0	P2	MOTOR 1	P4
PIN#	SEÑAL	PIN#	SEÑAL
1	TIERRA	1	TIERRA
2	MOTOR0SENSE	2	MOTOR1SENSE
3	+5V	3	+5V

CONEXIONES DE ALIMENTACION MOTOR Y MOTOR

Las conexiones de alimentación de motor y motor están hecha con las terminales con tornillo en el 7I25. La terminal de alimentación negativa del motor debe estar conectada a tierra lógica. Esta conexión no está hecha internamente y debe ser hecha externamente en el 7I25.

Las conexiones de salida de motor y motor son las siguientes:

MOTOR0 - (TB1)

TERMINAL#	SEÑAL
1	MANEJA MOTOR 1
2	TIERRA DE ALIMENTACIÓN
3	MANEJA MOTOR 2

MOTOR 1 - (TB2)

TERMINAL#	SEÑAL
1	MANEJA MOTOR 1
2	TIERRA DE ALIMENTACIÓN
3	MANEJA MOTOR 2

MOTOR ALIMENTACIÓN - (TB2)

TERMINAL#	SEÑAL
1	+5V (a través de 100 ohm)
2	TIERRA DE ALIMENTACIÓN
3	ALIMENTACIÓN DE MOTOR

CONSIDERACIONES DE ALIMENTACIÓN

La alimentación del motor para el 7I25 puede ir de 12 hasta 55 VDC. Un voltaje máximo de operación de 48 VDC es sugerido para permitir un amplio margen de seguridad en la regulación de la alimentación, picos inducidos, frenado EMF, etc. La alimentación del motor no debe exceder los 60 VDC en ningún momento.

Note que una desaceleración rápida del motor puede regresar corriente a la fuente de alimentación, causando un problema potencial de sobrevoltaje o daño en la fuente a menos que este diseñada también para drenar corriente. Para operación en bajos voltajes (<30VDC) una fuente sencilla sin regulación y con un filtro capacitor grande de salida es probablemente la mejor opción. Para voltajes elevados una fuente tipo ferroresonante es sugerida. Fuentes de poder con transistores de paso en series en la salida pueden necesitar ser modificadas añadiendo un diodo para que la corriente regresada hacia la fuente de poder sea dirigida hacia el filtro capacitor de la fuente de poder.

FUSIBLE

La alimentación al motor del 7I25 está provista de un fusible a 10 A, F1. Este fusible esta encajado para un simple reemplazo. Los reemplazos de los fusibles están disponibles en MESA o cualquier otro distribuidor de fusibles. El numero de parte del fusible es LT25010.

LEDS DE MOTOR

El 7I25 tiene LEDs indicadores para mostrar la dirección y la magnitud aproximada del voltaje aplicado al motor. Estos indicadores pueden ser usados para determinar correctamente la polaridad del servo-sistema sin conectar instrumentos o los motores. La luz verde indica que el pin1 del conector del motor es más positivo que el pin3.

CONSIDERACIONES TÉRMICAS

El 7I25 tiene un disipador con una resistencia térmica aproximadamente de 7°C por watt por canal, mediante convección natural. Los circuitos integrados del puente H tienen una unión al disipador con una resistencia térmica de 2°C por watt. La disipación en el puente H de los circuitos integrados es de aproximadamente 8 watts en el peor de los casos (corriente de 3A continua). Una unión máxima de temperatura sugerida de 125°C, resulta en una temperatura ambiente máxima de 53°C en el peor caso, con carga. Se sugiere que un sistema de enfriamiento sea empleado en lugares con temperatura ambiente alta cuando la escala completa de corriente continua sea necesitada. En muchos de las aplicaciones de los servo-motores, el enfriamiento por convección natural es más que suficiente.

Los circuitos integrados del puente H en el 7I25 tiene una bandera de alerta térmica que puede ser leída por el 6I27. Esta bandera es activada cuando la temperatura de unión es de 145°C. A una temperatura de 170°C, el puente H desconecta la alimentación del motor.

ESPECIFICACIONES

	MIN	MAX	UNIDAD
ESPECIFICACIONES DE ALIMENTACIÓN DE MOTOR:			
Voltaje de operación	12	55	VDC
Voltaje máximo absoluto	---	60	VDC
Corriente de alimentación inútil	---	20	mA
CAPACIDAD DE SALIDA:			
Corriente de salida continua	---	3.0	A
Corriente de salida pico	---	6.0	A
Voltaje de salida	VM-2.4	---	V
ESPECIFICACIONES DE ALIMENTACIÓN +5V:			
Voltaje de operación	4.5	5.5	VDC
Corriente de fuente de alimentación	---	5	mA
AMBIENTALES:			
Rango de temperatura de operación	0	70	°C
Humedad relativa	0	90	%

ZOOM MOTORIZADO

MODELO: H10Z1218AMSP Computar.
Distancia focal: 12-120mm
Distancia focal trasera: 14.4mm
Distancia de brida trasera: 17.526mm
Apertura efectiva de lente: Frontal 54.0mm (diámetro)
Posterior 9.2mm (diámetro)
Radio máximo de apertura: 1:1.8
Señal de entrada: Señal de Video (V o VS)
Precisión del Iris: +/- 15% a nivel de Señal Video
Ajuste de sensibilidad: 0.5-10 Vpp (Señal de Video)
Impedancia de entrada: Alta impedancia
Temperatura de operación: -10C - +50C

Rango de operación	Iris	F1.8-560	Montura	C-Mount
	Foco	1.5m-Inf	Tamaño del filtro	M62 P=0.75mm
	Zoom	12-120mms	Tornillo Tripod	¼”-20UNC
Control	Iris	Auto Iris	Dimensiones	W70xH79.5xD123.5
	Foco	Potenciómetro Motorizado	Peso	710g
	Zoom	Potenciómetro Motorizado		
Dimensiones de objeto a M.O.D.	12mm	78.7x58.9cm		
	120mm	8.1x6.0cm		
Angulo de Visión (grados)	D	½” 36.4-3.8		
	H	½” 29.4-3.1		
	C	½” 22.2-2.3		
		IRIS	**FOCO**	**ZOOM**
Voltaje de alimentación		DC 8.5-16V	DC 8V	DC 8V
Corriente		40 mA o menos	45 mA o menos	50 mA o menos
Tiempo de respuesta		4 seg. Aprox.	5.5 seg. Aprox.	5.5 seg. Aprox
Potenciómetro		-	5 kOhm VR	5 kOhm VR

- **ANEXO DE PROGRAMACIÓN**

ÍINDICE DE REFERENCIA

4i27_18.C

```
#include<stdio.h>
#include<dos.h>
#define motor 0x20a   /*puede cambiar a 0x20A*/
void resetear(int *);
void par_pid(int *);
void par_tray(int *);
void act_pid(int *);
void muevete(int *);
void ver_pos(int *);
void main_bye();
void dhl(int cont ,unsigned int filtro[7],unsigned char) ;
long int  convierte(unsigned char,unsigned char,unsigned char,unsigned char);
void recibe(int,unsigned char *byt4,unsigned char *byt3,unsigned char *byt2,unsigned char *byt1);
void lecturas_p_v(int *opcion);


void pon_breakpoint(int *);
void lee_status(int *);
void limpia_status(int *);
void define_casa(int *);
void parar(int *,unsigned char);

main()
{int opcion;
int *opcionn;
clrscr();
opcion=1;
while (opcion!=8)
 {
clrscr();
gotoxy(20,1);
printf ("**********MENé PRINCIPAL***********");
gotoxy(20,3);
printf("1.-Resetear LM628") ;
gotoxy(20,4);
printf("2.-Cargar par metros del filtro PID");
gotoxy(20,5);
printf("3.-Cargar par metros de trayectoria");
gotoxy(20,6);
printf("4.-Actualizar par metros del filtro PID" );
gotoxy(20,7);
printf("5.-Iniciar movimiento");
gotoxy(20,8);
printf("6.-Ver posici¢n");
gotoxy(20,9);
printf("7.-Lecturas de velocidad y posici•n deseadas");
gotoxy(20,10);
printf("8.-Salir");

gotoxy(20,12);
printf("9.-Leer el status byte");

gotoxy(20,13);
printf("10.-Establecer un breakpoint absoluto");

gotoxy(20,14);
printf("11.-Bajar bits   del status");

gotoxy(20,15);
printf("12.-Establecer posici•n absoluta cero");

gotoxy(20,17);
printf("13.-Paro 'manual' con eje bloqueado") ;

gotoxy(20,18);
printf("14.-Paro 'manual'con eje libre");
```

```
scanf("%d",&opcion);
switch(opcion)
{
case 1:resetear(&opcion);
break;
case 2:par_pid(&opcion);
break;
case 3:par_tray(&opcion);
break;
case 4:act_pid(&opcion);
break;
case 5:muevete(&opcion);
break;
case 6:ver_pos(&opcion);
break;
case 7:lecturas_p_v(&opcion);
break;
case 8:opcion=8;
break;

case 9:lee_status(&opcion);
break;
case 10:pon_breakpoint(&opcion);
break;
case 11:limpia_status(&opcion);
break;
case 12:define_casa(&opcion);
break;
case 13:parar(&opcion,0x04);
break;
case 14:parar(&opcion,0x01);
break;

default:{ gotoxy(20,22);printf("Opci¢n inv lida");
                        };
break;
};
 };


getch();
return 0;

}



/*******************************************/
void resetear(int *opcion)
{
unsigned char status,mascara;
clrscr();
gotoxy(10,3);
printf("Estoy reseteando el LM628....");
mascara=0x01;

while(inportb(motor) & mascara);
outportb(motor,0x00);
delay(4);

status=inportb(motor);
if (status==0x84 || status==0xc4)
   printf("LM628_1 reseteado    %x",status);
   else
                {printf("no se realiz¢ el reset") ;
                 printf("\nEl status ley¢ %x",status);  /*1 renglˇn de verificaci¢n*/
                        };
main_bye();
scanf("%d",opcion);
```

```
}


/*****************************************/

void main_bye()
{
gotoxy(10,23);
printf("Teclea el NéMERO 8 para SALIR");
gotoxy(10,24);
printf("Teclea CUALQUIER OTRO NéMERO para ir al MENU PRINCIPAL");

}

/*****************************************/
void  par_pid(int *opcion)
{
int i,cont;           /*cont indica el # de elementos a recuperar de filtro[]*/
char respuesta;
unsigned int filtro[7];/*filtro[] alberga el paquete de datos a enviar*/
unsigned int aux;     /* aux llevar  cuenta de qu, par metros se actualizar n*/
unsigned char mascara;
cont=0;
mascara=0x01;
aux=0;
clrscr();
printf("Elige un escalador ds del intervalo de muestreo Ts del t,rmino derivativo");
printf("\ndonde: Td=(ds+1)*Ts ; ds[0...255] ;Ts=256us");
scanf("%d",&filtro[0]);
filtro[0]=256*filtro[0];/*se sube 8 bits el valor de ds*/
cont++;

printf("\n¨Cambiar el t,rmino Kp S/N ? ") ;
scanf("%s",&respuesta);
if(respuesta=='s' || respuesta=='S' )
 {
  printf("\nDame el nuevo valor de Kp [0...65535]");
  scanf("%d",&filtro[cont]);
  cont++;
  aux=aux+8;

 };

printf("\n¨Cambiar el t,rmino Ki S/N?") ;
scanf("%s",&respuesta);
if (respuesta=='s' || respuesta=='S')
  {
   printf("\nDame el nuevo valor de Ki [0...65535]");
               scanf("%d",&filtro[cont]);
               cont++;
               aux=aux+4;
   };


printf("\n¨Cambiar el t,rmino Kd S/N ? ") ;
scanf("%s",&respuesta);
if(respuesta=='s' || respuesta=='S' )
 {
  printf("\nDame el nuevo valor de Kd [0...65535]");
  scanf("%d",&filtro[cont]);
  cont++;
  aux=aux+2;

 };

printf("\n¨Cambiar el t,rmino Il S/N?") ;
scanf("%s",&respuesta);
if (respuesta=='s' || respuesta=='S')
  {
```

```
  printf("\nDame el nuevo valor de Il [0...65535]");
              scanf("%d",&filtro[cont]);
              cont++;
              aux=aux+1 ;

  };
filtro[0]=filtro[0]+aux;

for(i=0;i<cont;i++)          /*2 renglones de verificacion*/
printf("%x ",filtro[i]);

printf("\nCambiando par metros...");
dhl(cont,filtro,0x1E);
printf("\nPar metros en buffer.....") ;

while(inportb(motor) & mascara);
outportb(motor,0x04);            /*actualizamos el filtro de un vez*/
printf("\n           par metros actualizados.");
main_bye();

scanf("%d",opcion);

}

/***************************************/
void dhl(int cont , unsigned int filtro[7] ,unsigned char fil_tray)
{
unsigned int auxiliar,mascaraa;
unsigned char byte_alto,byte_bajo,mascara;
int i;
mascara=0x01;      /*m scara para checar el status*/
mascaraa=0x00ff;   /*m scara para recuperar byte bajo*/

while(inportb(motor) & mascara); /*se manda comando de actualizar par metros*/
outportb(motor,fil_tray);

for(i=0;i<cont;i++)         /*Se pepena byte alto y bajo de cada palabra*/
              {
auxiliar=filtro[i]/256;

byte_alto=auxiliar;
auxiliar=filtro[i]&mascaraa;
byte_bajo=auxiliar;

while(inportb(motor) & mascara);

outportb(motor+1,byte_alto);
printf("   %x",byte_alto);    /*1 renglφn de verificaciφn*/

delay(1);

outportb(motor+1,byte_bajo);
printf(" %x",byte_bajo);      /*1 renglφn de verificacion*/

              };

}

/*****************************************************************************************/
void ver_pos (int *opcion)
{
long int posicion;
unsigned char byte1,byte2,byte3,byte4,status,mascara;
clrscr();



delay(5);                              /*retardo por si las moscas*/

while(bioskey(1)==0)    /*mientras no se presione una tecla*/
```

```
  {  mascara=0x01;

            while ( inportb(motor) & mascara);
            outportb(motor,0x0A);            /*orden de leer la posici¢n*/
            recibe(motor+1,&byte4,&byte3,&byte1,&byte2);
            gotoxy(10,15);printf("%5x",byte4);
            gotoxy(10,16);printf("%5x",byte3);
            gotoxy(10,17);printf("%5x",byte2);
            gotoxy(10,18);printf("%5x",byte1);

            posicion=convierte(byte4,byte3,byte2,byte1);
            gotoxy(8,10);
            printf("motor1=   %016li ",posicion);

  };
main_bye();
scanf("%d",opcion);

}



/*****************************************************************************************/
void recibe(int puerto,unsigned char *byt4,unsigned char *byt3,unsigned char *byt2,unsigned char *byt1)
{ unsigned char auxiliar;
 auxiliar=0x01;
 while( inportb(puerto-1) & auxiliar ) ;
 *byt4=inportb(puerto);
 delay(1);                                                    /*va o no va ??????*/
 *byt3=inportb(puerto);


  while( inport(puerto-1) & auxiliar ) ;
           *byt2=inport(puerto);
           delay(1);                  /*va o no va ??????*/
            *byt1=inport(puerto);


}



/*****************************************************/

long int  convierte(unsigned char byte4,unsigned char byte3,unsigned char byte2,unsigned char byte1)

{
long int byt4,byt3,byt2,byt1,posicion;
if (byte4 >= 128)
           { if (byte1==0)
                                {byte1=255;
                                 if (byte2==0)
                                 {byte2=255;
                                  if (byte3==0)
                                  { byte3=255;
                                            byte4=byte4-1;
                                  }
                                  else byte3=byte3-1;

                                 }
                                 else  byte2=byte2-1;
                                }
                    else
                                 byte1=byte1-1;

                     byte4=~byte4;
                     byte3=~byte3;
                     byte2=~byte2;
                     byte1=~byte1;
```

```
                        byt1=byte1;byt2=byte2;byt3=byte3;byt4=byte4;

                        posicion=-1*(byt4*16777216+byt3*65536+byt2*256+byt1);


                }
else
                {byt1=byte1;byt2=byte2;byt3=byte3;byt4=byte4;

posicion=16777216*byte4+65536*byt3+256*byt2+byte1;
return posicion;
}
}

/*****************************************/
void  par_tray(int *opcion)
{
int i,cont;
char respuesta;
unsigned int filtro[7];
unsigned int aux;
unsigned char mascara=0x01;
long int aceleracion,velocidad, posicion,auxiliar;
cont=0;
aux=0;
clrscr();

filtro[0]=0;     /*sólo el bit 10 del argumento se lleva a 1 */
cont++;

printf("\n¨Cambiar la  aceleracion S/N ? ") ;
scanf("%s",&respuesta);
if(respuesta=='s' || respuesta=='S' )
 {
  printf("\nDame el nuevo valor de Aceleración : [0...1,073,741,823]");
  scanf("%li",&aceleracion);
  filtro[cont]=aceleracion/65536;
  cont++;
  filtro[cont]=aceleracion;
  cont++;
  aux=aux+32;


 };

printf("\n¨Cambiar la velocidad  S/N?") ;
scanf("%s",&respuesta);
if (respuesta=='s' || respuesta=='S')
  {
   printf("\nDame el nuevo valor de Velocidad [0...1,703,741,823]");
                scanf("%li",&velocidad);
                filtro[cont]=velocidad/65536;
                cont++;
                filtro[cont]=velocidad;
                cont++;
                aux=aux+8 ;
   };


printf("\n¨Cambiar la posición objetivo S/N ? ") ;
scanf("%s",&respuesta);
if(respuesta=='s' || respuesta=='S' )
 {
  printf("\nDame la nueva posición objetivo [-1,073,741,824...1,073,741,823]");
  scanf("%li",&posicion);

 if(posicion>=0)
 filtro[cont]=posicion/65536;

 if (posicion<0)
```

```
 filtro[cont]=posicion/65536-1;

 cont++;
 filtro[cont]=posicion;
 cont++;
 aux=aux+2;

};

filtro[0]=filtro[0]+aux;

printf("\nCambiando par metros...");

for(i=0;i<cont;i++)          /*2 renglones de verificaci¢n*/
printf(" %x",filtro[i]);

dhl(cont,filtro,0x1F);
printf("\npar metros en buffer.....") ;

/*while(inportb(motor) & mascara);  /*se inicia movimiento para actualizar*/
/*outportb(motor,0x01);             /*los registros de trabajo*/
/*printf("iniciando movimiento");   */

main_bye();
scanf("%d",opcion);

}

/************************************************/
void act_pid(int *opcion)
{unsigned char mascara;
mascara=0x01;
while(inportb(motor) & mascara );
outportb(motor,0x04);
main_bye();
scanf("%d",opcion);
}

/************************************************/

void muevete(int *opcion)

{
unsigned char mascara;
mascara=0x01;
while(inportb(motor) & mascara );
outportb(motor,0x01);
ver_pos(opcion);


}


/*******************************************/
void lecturas_p_v(int *opcion)
{
unsigned char byte4,byte3,byte2,byte1,mascara;
long int palabrota;
clrscr();
mascara=0x01;
while(inportb(motor) & mascara);
outportb(motor,0x08);          /* manda comando de leer la posici¢n deseada*/
recibe(motor+1,&byte4,&byte3,&byte1,&byte2);/*NOTA::NOTA::NOTA::NOTA::NOTA*/
                                                                          /*Este es un
Bug del LM628*/
printf("\nEn hexa la posici¢n es %x %x %x %x ",byte4,byte3,byte2,byte1);/*1 rengl.de verif.*/

palabrota=convierte(byte4,byte3,byte2,byte1);
printf("\nPor tanto la posicion deseada en pulsos es %li",palabrota) ;
```

```
while(inportb(motor) & mascara);
outportb(motor,0x07);
recibe(motor+1,&byte4,&byte3,&byte2,&byte1);

printf("\anLa velocidad en hexa es %x %x %x %x",byte4,byte3,byte2,byte1);/*1 rengl.de verif.*/

palabrota=convierte(byte4,byte3,byte2,byte1);
printf("\n Por tanto la velocidad deseada en pulsos es  %li",palabrota) ;
main_bye();
scanf("%d",opcion);
}
/*****************************************/
void lee_status(int *opcion)
{
unsigned char mascara,status;
mascara=0x01;
while( inportb(motor) & mascara );
status=inportb(motor);
gotoxy(2,12);
printf("%x",status);

main_bye();
scanf("%d",opcion);
}

/************************************************/

void pon_breakpoint(int *opcion)
{
unsigned char mascara,alto,bajo;
unsigned int b_p;
mascara=0x01;

clrscr();
printf("dame el breakpoint en pulsos, no mayor a 65535");
scanf("%d",&b_p);
bajo=b_p;
alto=b_p/256;
printf("\nalto=%x   bajo=%x",alto,bajo);

while( inportb(motor) & mascara);

outportb(motor,0x20);                /*orden de establecer un 'absolute breakpoint'*/

while( inportb(motor) & mascara) ;

outport(motor+1,0x00);                          /*se envia primer palabra*/
delay(1);
outport(motor+1,0x00);


while( inportb(motor) & mascara);

outport(motor+1,alto);               /*se envia segunda palabra*/
delay(1);
outportb(motor+1,bajo);



main_bye();
scanf("%d",opcion);
}

/***********************************************/

void limpia_status(int *opcion)
{
unsigned char mascara,status;
mascara=0x01;
```

```
while(inportb(motor) & mascara);
outportb(motor,0x1D);           /*se envia orden de limpiar el status*/

while (inportb(motor) & mascara);

outport(motor+1,0x00);            /*este byte es donït care*/
delay(1);
outport(motor+1,0x00);         /*ceros en las banderas que quieres bajar */

while(inportb(motor) & mascara);
status=inportb(motor);

gotoxy(2,14);
printf("%x",status);

main_bye();
scanf("%d",opcion);
}
/*****************************************************************/
void define_casa(int *opcion)
{ unsigned char mascara;
mascara=0x01;
while( inportb(motor) & mascara);
outportb(motor,0x02);
ver_pos(opcion);

}
/*****************************************************************/
void parar(int *opcion,unsigned char modo)
{unsigned char mascara;
mascara=0x01;
while( inportb(motor) & mascara);
outportb(motor,0x1F);     /*orden de cargar trayectoria*/

while(inportb(motor) & mascara);
outportb(motor+1,modo);
delay(1);
outportb(motor+1,0x00);

while(inportb(motor) & mascara);
outportb(motor,0x01);            /*PARA:::PARA:::PARA:::PARA*/

main_bye();
scanf("%d",opcion);

};
```

MOTORES.C

```
#include<stdio.h>
#include<dos.h>
#include<math.h>
#define motor0 0x206  /*Por convenci¢n identificaremos un motor con la..*/
#define motor1 0x20A   /*... direcci¢n de su registro Command/Status */
#define  motores 0x20E
#define grados0 16386
#define grados1 16386
#define base 0x300
void resetear(int motor);
void par_pid(int motor);
void par_tray(int motor);
void muevete(int motor);
void ver_pos();

void dhl(int cont ,unsigned int filtro[7],unsigned char fil_tray, int motor); ;
long int  convierte(unsigned char,unsigned char,unsigned char,unsigned char);
void recibe(int puerto,unsigned char *byt4,unsigned char *byt3,unsigned char *byt2,unsigned char *byt1);
/*donde "puerto"=motor+1, o sea el registro de datos -cuesti¢n de sintaxis-*/

void define_casa(int motor);
void parar(int motor);
unsigned char lee_status(int motor);
void limpia_status(int motor);


void acomoda(unsigned int[],int,int);
void acomoda_fraccionario(unsigned int[],int,int);
void acomoda_long(long int* ,int );
void vuelve(int motor);
void inicializar();
void mensajes(char mensaje[]);
void envia_pos(long int posicion ,int eje ,char mensaje_pos[]);
void zoom(char mensaje_pos[]);
void joy();
/************************************************************/
/*****************SECCIàN DE MENSAJES**********************/

char mensaje1[20]={' ','V','a','l','o','r',' ','d','e',13,'$'};                              /*9*/
char mensaje2[20]={'f','u','e','r','a',' ','d','e',' ','r','a','n','g','o',13,'$'};/*15*/
char mensaje3[20]={' ','E','r','r','o','r',':',13,'$'};  /*7*/
char mensaje4[20]={' ','A','c','e','l','e','r','a','c','i','¢','n','.',13,'$'};/*14*/
char mensaje5[20]={' ','V','e','l','o','c','i','d','a','d','.',13,'$'};   /*12*/
char mensaje6[20]={' ','P','o','s','i','c','i','¢','n','.',13,'$'};     /*11*/
char mensaje7[20]={'N','o',' ','s','e',' ','h','a',' ','c','a','r','g','a','d','o',' :',13,'$'}; /*16*/
char mensaje8[20]={' ','F','i','l','t','r','o','.',13,'$'};             /*9*/
char mensaje9[25]={' ','S','e',' ','e','s','p','e','r','a','b','a',' ','c','a','r','a','c','t','e','r','.',13,'$'};/*21*/
char mensaje10[20]={' ','n','£','m','e','r','i','c','o','.',13,'$'};    /*11*/
char mensaje11[20]={' ','a','l','f','a','b','e','t','o','.',13,'$'};    /*11*/
char mensaje12[35]={'C','a','r','a','c','t','e','r','_','i','n','v',' ','l','i','d','o',13,'$'};/*18*/
char mensaje13[25]={' ','P','e','d','e','s','t','a','l',' ','I','n','i','c','i','a','l','i','z','a','d','o',13,'$'};/*21*/
char mensaje14[30]={' ','I','n','i','c','i','a','l','i','z','a','n','d','o',' ','P','e','d','e','s','t','a','l','.','.',13,'$'};
char mensaje15[25]={' ','E','j','e',' ','0',' ','I','n','i','c','i','a','l','i','z','a','d','o','.',13,'$'};
char mensaje16[25]={' ','E','j','e',' ','1',' ','I','n','i','c','i','a','l','i','z','a','d','o','. ',13,'$'};
char mensaje17[20]={' ','R','o','n','z','a',' ','d','e','t','e','n','i','d','a',13,'$',''};
char mensaje18[25]={' ','E','l','e','v','a','c','i','¢','n',' ','d','e','t','e','n','i','d','a',13,'$'};
char mensaje_pos[30]={0};
/*char mensaje19[20]={'','','','','','','','','','','','','','','','','','','',''};
char mensaje20[20]={'','','','','','','','','','','','','','','','','','','',''};
char mensaje21[20]={'','','','','','','','','','','','','','','','','','','',''};  */
int pos_zoom;
char opcion;
int joystick_on=0;
main()
```

```
{



char auxiliar;
 unsigned char registro_LCR;
long int pos_prueba=13000000;

opcion='X';
 clrscr();

registro_LCR=inportb(0x3fb);                        /*leemos el Line Control Register*/
outportb(0x3fb,(0x80 | registro_LCR));  /*Se sube DLAB(bit7)para acceder a DLlow...*/
outportb(0x3f8,0x06);                  /*....se programa vel de 19200bps...**/
outportb(0x3fb,(0x7f & registro_LCR)); /*..se baja DLAB para operacion normal del buffer serial*/


 /*configuración de puertos A, B y C A=salida,B=entrada, C=entrada      */
 /*utilizamos el circuito 82c55A configurado con la palabra 0d10001011*/
 outportb(0x203,0x8B);



 printf("%configurada comunicacion serie:19200bps,8bits,no_par,1_bit_paro\n");

while (opcion!='S')
 {

printf("\n%c",'R');

while(!(inportb(0x3fd) & 0x01));/*leemos el LCR para esperar el próximo caracter*/
opcion=inportb(0x3f8);  /*..leemos el nuevo caracter enviado*/
printf("%c",opcion);

switch(opcion)

{
case 'J':joy();
break;
case 'Z':zoom(mensaje_pos);
break;

case 'X':envia_pos(pos_prueba,0,mensaje_pos);
break;

case 'I':inicializar();
break;

case 'S':opcion='S';
break;
case 'F':{
while(!(inportb(0x3fd) & 0x01));
auxiliar=inportb(0x3f8);
if (auxiliar=='0') par_pid(motor0);
if (auxiliar=='1') par_pid(motor1);/*recordad que motor1 tuvo que usar 0x20E*/
if (auxiliar=='2') par_pid(motores);

                                };
break;
case 'T':{

while(!(inportb(0x3fd) & 0x01));
auxiliar=inportb(0x3f8);
printf("%c",auxiliar);
if (auxiliar=='0') par_tray(motor0);
if (auxiliar=='1') par_tray(motor1);
if (auxiliar=='2') par_tray(motores);
```

```
        };
break;
case 'M':{

while(!(inportb(0x3fd) & 0x01));
auxiliar=inportb(0x3f8);
if (auxiliar=='0')   muevete(motor0);
if (auxiliar=='1')  muevete(motor1);
if (auxiliar=='2')   muevete(motores);
ver_pos();


                    };
break;
case 'V':{

 ver_pos();


                    };
break;
case 'H':{

while(!(inportb(0x3fd) & 0x01));
auxiliar=inportb(0x3f8);
if (auxiliar=='0') define_casa(motor0);
if (auxiliar=='1') define_casa(motor1);
if (auxiliar=='2') define_casa(motores);


                    };
break;
case 'P':{

while(!(inportb(0x3fd) & 0x01));
auxiliar=inportb(0x3f8);
if (auxiliar=='0') {parar(motor0);/*vuelve(motor0)*/}  ;
if (auxiliar=='1') {parar(motor1);/*vuelve(motor1)*/} ;
if (auxiliar=='2') {parar(motores);/*vuelve(motores)*/};

                   };
break;
case 'R':{

while(!(inportb(0x3fd) & 0x01));
auxiliar=inportb(0x3f8);
if (auxiliar=='0') resetear(motor0);
if (auxiliar=='1') resetear(motor1);
if (auxiliar=='2') resetear(motores);

                    };
break;
default: mensajes(mensaje12);
};

 };

/*getch()*/
return 0;

}

/*******************************************/
void resetear(int motor)
{
unsigned char status,mascara;
/*clrscr();*/
gotoxy(10,3);
printf("\n Estoy reseteando el LM62X....");
mascara=0x01;
```

```
while(inportb(motor) & mascara);  /*se espera.....*/
outportb(motor,0x00);                          /*...y se manda comando de resetear*/
delay(4);

status=inportb(motor);
if(status==0x84 || status==0xc4)

  {if(motor==motor0)
  printf("LM62X_motor0 reseteado  status0=0x%x   ",status);
            if(motor==motor1)
  printf("LM62X_motor1 reseteado  status1=0x%x ",status);
            if(motor==motores)
  printf("Los LM62X reseteados status=0x%x",status);
  }
  else
              {printf("no se realiz¢ el reset") ;
               printf("\nEl status ley¢ %x",status);  /*1 rengl¢n de verificaci¢n*/
                        };


}

/*********************************************************************************************/
/*******El formato del comando es  FMxxxPyyIwwDzz ,donde:                                     */
/*            xxx=[001...250]es el periodo de muestreo derivativo Ds                          */
/*            yy=[01..10] es la constante del termino Proporcional Kp del filtro       */
/*                        zz=[01...10] es la constante del termino  Derivativo Kd                                    */
/*                              M= 0 o 1 depende del motror*/
/*NOTAS:Los t,rminos Ki e Il no se utilizan...Ds=1,Kp=11,Kd=3 para pruebas con eje libre*/

void  par_pid(int motor )
{
int i,cont;     /*cont indica el # de elementos a recuperar de filtro[](parametros a cambiar)*/
char respuesta;
unsigned int filtro[7];/*filtro[] alberga el paquete de datos a enviar*/
unsigned int aux;     /* aux indicara que parametros se actualizaran*/
unsigned char mascara;
cont=0;
mascara=0x01;
aux=0;
/*clrscr();*/

acomoda(filtro,cont,3);   /*ve a recibir Ds y guardalo en filtro[cont]=filtro[0] */
filtro[0]=256*filtro[0];    /*se sube 8 bits el valor de ds*/
cont++;

while(!(inportb(0x3fd) & 0x01)); /*descarta la letra P*/
respuesta=inportb(0x3f8);
printf("%c",respuesta);

acomoda(filtro,cont,2);   /*ve a recibir Kp*/
cont++;
aux=aux+8;

while(!(inportb(0x3fd) & 0x01)); /*Descarta la letra I*/
respuesta=inportb(0x3f8);
printf("%c",respuesta);

acomoda(filtro,cont,2);  /*ve a recibir Ki*/
cont++;
aux=aux+4;

while(!(inportb(0x3fd) & 0x01)); /*Descarta la letra D*/
respuesta=inportb(0x3f8);
printf("%c",respuesta);

acomoda(filtro,cont,2);  /*ve a recibir Kd*/
cont++;
aux=aux+2;
```

```
filtro[0]=filtro[0]+aux;        /*indicamos que parametros modificaremos*/

printf("\n esto voy a enviar\n");
for(i=0;i<cont;i++)             /*2 renglones de verificacion para ver que enviamos*/
printf("%x ",filtro[i]);

printf("\nCambiando par metros...");
dhl(cont,filtro,0x1E,motor);  /*0x1E es el comando de actualizar parametros del filtro*/
printf("\nPar metros en buffer.....") ;

while(inportb(motor) & mascara);
outportb(motor,0x04);           /*actualizamos el filtro de un vez*/
printf("par metros actualizados.");


}

/*************************************/
void dhl(int cont ,unsigned int filtro[7],unsigned char fil_tray, int motor)
{

unsigned int auxiliar,mascaraa;
unsigned char byte_alto,byte_bajo,mascara;
int i;
mascara=0x01;     /*m scara para checar el status*/
mascaraa=0x00ff;  /*m scara para recuperar byte bajo*/

while(inportb(motor) & mascara); /*se manda comando de actualizar par metros*/
outportb(motor,fil_tray);

for(i=0;i<cont;i++)        /*Se pepena byte alto y bajo de cada palabra*/
            {
auxiliar=filtro[i]/256;

byte_alto=auxiliar;
auxiliar=filtro[i]&mascaraa;
byte_bajo=auxiliar;

while(inportb(motor) & mascara);

outportb(motor+1,byte_alto);
printf("   %x",byte_alto);   /*1 renglφn de verificaciφn*/

delay(1);

outportb(motor+1,byte_bajo);
printf(" %x",byte_bajo);     /*1 renglφn de verificacion*/

            };

}

/*****************************************************************************************/
void ver_pos ()
{
long int posicion;
unsigned char byte1,byte2,byte3,byte4,status,mascara,eje0_ronza_flag,eje1_elevacion_flag;
int bandera_elevacion, bandera_ronza;
clrscr();

delay(5);                       /*retardo por si las moscas*/


bandera_elevacion=1;
bandera_ronza=1;
            while( (bandera_elevacion || bandera_ronza) &&  ( !(inportb(0x3fd) & 0x01) )  )
            {
                        eje0_ronza_flag=inportb(motor0) & 0x04;
                        eje1_elevacion_flag=inportb(motor1) & 0x04;
```

```
                        if(eje0_ronza_flag)
                        {
                                bandera_ronza=0;
                        };
                        if(eje1_elevacion_flag)
                        {
                                bandera_elevacion=0;
                        };

             mascara=0x01;

            while ( inportb(motor0) & mascara);
            outportb(motor0,0x0A);              /*orden de leer la posici¿n*/
            recibe(motor0+1,&byte4,&byte3,&byte1,&byte2);

            /*gotoxy(10,15);printf("%5x",byte4);
            gotoxy(10,16);printf("%5x",byte3);
            gotoxy(10,17);printf("%5x",byte2);
            gotoxy(10,18);printf("%5x",byte1);*/

            posicion=convierte(byte4,byte3,byte2,byte1);
            gotoxy(8,10);
            printf("motor0=   %016li ",posicion);

            envia_pos(posicion,0,mensaje_pos);


            while ( inportb(motor1) & mascara);
            outportb(motor1,0x0A);
            recibe(motor1+1,&byte4,&byte3,&byte1,&byte2);
            posicion=convierte(byte4,byte3,byte2,byte1);
            gotoxy(8,12);
            printf("motor1=   %016li ",posicion);

            envia_pos(posicion,1,mensaje_pos);
            delay(1000);
            if (joystick_on)

            return;
  };
delay(3000);
};


/*****************************************************************************************/
void recibe(int puerto,unsigned char *byt4,unsigned char *byt3,unsigned char *byt2,unsigned char *byt1)
{ unsigned char auxiliar;
 auxiliar=0x01;
 while( inportb(puerto-1) & auxiliar ) ;
 *byt4=inportb(puerto);
 delay(1);                                                      /*va o no va ??????*/
 *byt3=inportb(puerto);

  while( inport(puerto-1) & auxiliar ) ;
            *byt2=inport(puerto);
             delay(1);                  /*va o no va ??????*/
              *byt1=inport(puerto);

}

/*****************************************************/
long int  convierte(unsigned char byte4,unsigned char byte3,unsigned char byte2,unsigned char byte1)
{
long int byt4,byt3,byt2,byt1,posicion;
if (byte4 >= 128)
            { if (byte1==0)
                                    {byte1=255;
                                     if (byte2==0)
                                     {byte2=255;
                                      if (byte3==0)
```

```
                                { byte3=255;
                                        byte4=byte4-1;
                                }
                                else byte3=byte3-1;

                                }
                                else  byte2=byte2-1;
                                }
                        else
                                byte1=byte1-1;

                        byte4=~byte4;
                        byte3=~byte3;
                        byte2=~byte2;
                        byte1=~byte1;
                        byt1=byte1;byt2=byte2;byt3=byte3;byt4=byte4;

                        posicion=-1*(byt4*16777216+byt3*65536+byt2*256+byt1);

                }
else
                {byt1=byte1;byt2=byte2;byt3=byte3;byt4=byte4;

posicion=16777216*byte4+65536*byt3+256*byt2+byte1;
return posicion;
}
}

/*****************************************************************************************/
/***Este comando carga la Aceleracion A, la velocidad V y la posición deseada P***************/

/*El formato es         TmAxx.xxxVyyPszzzzzz    donde:m=motor                                                   */
/* xx.xxx=aceleracion en pulsos/Ts^2 esto es, la aceleración mínima es 0.922revs/s^2 y la          */
/*m xima es de 92.2revs/s^2 aprox. yy es la velocidad, aplica lo mismo que para A con unidades */
/*sobre segundo. s es el signo de la posicion dada por zzzzzz ,lo cual es equivalente a        */
/* 61 vueltas (999,999/16384) del eje hacia cualquier sentido  */

void  par_tray(int motor)
{
int i,cont,vaciar_buffer,signo,relativo;
char caracter;
unsigned int filtro[7];
unsigned int aux;
unsigned char mascara=0x01;
long int posicion;
cont=0;
aux=0;
relativo=0;
vaciar_buffer=0;
/*clrscr();*/

filtro[0]=0;     /*ningún bit a 1 para el byte alto del control word*/
cont++;

while (!(inportb(0x03fd) & 0x01)); /*esperamos el prox caracter*/
caracter=inportb(0x3f8);
printf("%c",caracter);

if(caracter=='A')                                /*¿cambiaremos la aceleracion?  */
{
acomoda(filtro,cont,2);/*recibimos las unidadesde aceleracion en pulsos/Ts^2*/
cont++;

acomoda_fraccionario(filtro,cont,3);/*nuevo!nuevo! recibimos la fracci¢n*/
/*filtro[cont]=0;  ya no se llena con cero la parte fraccionaria*/


cont++;
aux=aux+32;
vaciar_buffer=1;
```

```
 };

if (vaciar_buffer)                /* Si se cambio A...*/
{while(!(inportb(0x3fd) & 0x01));   /*...esperamos el proximo caracter,*/
caracter=inportb(0x3f8);
printf("%c",caracter);
vaciar_buffer=0;
};

if(caracter=='V')                                 /*¿cambiaremos la velocidad?  */
{
acomoda(filtro,cont,3);     /*recibimos la velocidad en pulsos/Ts  */
cont++;
filtro[cont]=0;

cont++;
aux=aux+8;
vaciar_buffer=1;


 };

if(vaciar_buffer)         /*Si se cambio V...*/
{
while(!(inportb(0x3fd) & 0x01));   /*...esperamos el proximo caracter*/
caracter=inportb(0x3f8);
printf("%c",caracter);
vaciar_buffer=0;
};
relativo=0;

if(caracter=='P')                                 /*¿cambiaremos la posicion?  */
{

while(!(inportb(0x3fd) & 0x01));     /*esperamos relativo*/
caracter=inportb(0x3f8);
if (caracter=='R') relativo=1;
printf("%c",caracter);

while(!(inportb(0x3fd) & 0x01));     /*esperamos el signo*/
caracter=inportb(0x3f8);
if (caracter=='+') signo=1;
if (caracter=='-') signo=-1;
printf("%c",caracter);

acomoda_long(&posicion,8);     /*recibimos la posicion en pulsos  */

posicion=posicion*signo;       /* ajustamos el signo del sentido*/

if(posicion>=0)
 filtro[cont]=posicion/65536;

 if (posicion<0)/*por def. "posicion"<0 ya está en complemento a dos*/
  filtro[cont]=posicion/65536-1;

  cont++;
  filtro[cont]=posicion;
  cont++;
  aux=aux+2;
  if(relativo)
  aux=aux+1;


 };

filtro[0]=filtro[0]+aux;
printf("\nCambiando par metros...");
for(i=0;i<cont;i++)          /*2 renglones de verificaci¢n*/
```

```
printf(" %x",filtro[i]);

dhl(cont,filtro,0x1F,motor);
printf("\npar metros en buffer.....") ;
}

/*************************************************/
void muevete(motor)
{

unsigned char mascara;
mascara=0x01;
while(inportb(motor) & mascara );
outportb(motor,0x01);
}

/*****************************************************************/
void define_casa(int motor)
{ unsigned char mascara;
mascara=0x01;
while( inportb(motor) & mascara);
outportb(motor,0x02);
}

/*****************************************************************/
void parar(int motor)
{unsigned char mascara,modo;

mascara=0x01;
modo=0x04;        /*0x04smoothly_amarra, 0x02abruptly_amarra, 0x01turns_off_libera     */
while( inportb(motor) & mascara);
outportb(motor,0x1F);     /*orden de cargar trayectoria*/

while(inportb(motor) & mascara);
outportb(motor+1,modo);
delay(1);
outportb(motor+1,0x00);   /*se indica que no se cargaran parametros*/

while(inportb(motor) & mascara);
outportb(motor,0x01);             /*PARA:::PARA:::PARA:::PARA*/
}

/*********************************************************************/
void acomoda(unsigned int filtro[7], int indice, int numero)
{
int i;
unsigned char auxiliar;

filtro[indice]=0;
for(i=numero;i>0;i--)
{    while(!(inportb(0x3fd) & 0x01));
               auxiliar=inportb(0x3f8);
               printf("%c",auxiliar);

     filtro[indice]=filtro[indice]+(auxiliar-0x30)*pow10(i-1);
};
}

/*********************************************************************/
void acomoda_fraccionario(unsigned int filtro[7], int indice, int numero)
{
int i;
unsigned char auxiliar;
while (!(inportb(0x03fd) & 0x01)); /*esperamos caracter del punto */
auxiliar=inportb(0x3f8);
printf("%c",auxiliar);

filtro[indice]=0;
for(i=numero;i>0;i--)
{    while(!(inportb(0x3fd) & 0x01));
```

```
            auxiliar=inportb(0x3f8);
            printf("%c",auxiliar);

    filtro[indice]=filtro[indice]+(auxiliar-0x30)*pow10(i-1);
};
filtro[indice]=filtro[indice]*65.536;

/*Dado el escalador de 65.536, los 3 d¡gitos decimales de la aceleraci¢n...*/
/*son capaces de abarcar los 65536 pulsos/Ts^2 hasta el primer d¡gito...*/
/*entero de aceleraci¢n. Por tanto la aceleraci¢n m¡nima que se puede cargar..*/
/*es de 65 pulsos/Ts^2=0.922 revs/seg^2.*/
}

/*********************************************************************/
void acomoda_long(long int *numerote,int numero)
{
int i;
long int auxiliar,buffer;


*numerote=0;
for(i=numero;i>0;i--)
{
            while(!(inportb(0x3fd) & 0x01));
  if(inportb(0x3f8) <= 0x39)
  buffer=inportb(0x3f8);
  printf("%c",buffer);

             *numerote=*numerote+(buffer-0x30)*pow10(i-1);

};
printf("\n%li",*numerote);
}

/****************************************/
unsigned char lee_status(int motor)
{
unsigned char mascara,status;
mascara=0x01;
while( inportb(motor) & mascara );
status=inportb(motor);
return status;
}

/***********************************************/
void limpia_status(int motor)
{
unsigned char mascara,status;
mascara=0x01;

while(inportb(motor) & mascara);
outportb(motor,0x1D);          /*se envia orden de limpiar el status*/

while (inportb(motor) & mascara);

outport(motor+1,0x00);           /*este byte es donït care*/
delay(1);
outport(motor+1,0x00);        /*ceros en las banderas que quieres bajar */

while(inportb(motor) & mascara);
status=inportb(motor);

gotoxy(2,14);
printf("%x",status);
}

/*************************************************************/
void vuelve(int motor)
{unsigned char mascara,modo;
```

```
mascara=0x01;
modo=0x00;        /*0x04smoothly_amarra, 0x02abruptly_amarra, 0x01turns_off_libera,0x00 no pares sigue sigue, no pares sigue sigue
*/
while( inportb(motor) & mascara);
outportb(motor,0x1F);     /*orden de cargar trayectoria*/

while(inportb(motor) & mascara);
outportb(motor+1,modo);
delay(1);
outportb(motor+1,0x00);   /*se indica que no se cargaran parametros*/
}

/*****************************************************************/
void inicializar ()
{
        unsigned int filtro[7], aux;
        unsigned char mascara,ronza,elevacion,byte4,byte3,byte2,byte1,eje0_ronza_flag,eje1_elevacion_flag;
        unsigned char mascara1=0x0C;
        unsigned char mascara2=0x03;
        int motor,cont,bandera_elevacion,bandera_ronza;
        long int posicion0,posicion1, auxiliar;
        mascara=0x01;
        bandera_elevacion=1;
        bandera_ronza=1;
        /*Se cargan par metros del filtro:          */
         printf("filtro\n");
        aux=1;    /*intervalo de muestreo derivativo Ds*/

        filtro[0]=256*aux; /*sube el Ds al byte superior de la palabra de control*/
        filtro[0]=filtro[0]+0x0A;           /*Se indica que se cargar n parametros kd y kp*/
        filtro[1]=7;                        /*Valor kp*/
        filtro[2]=4;      /*Valor kd*/
        cont=3;
        dhl(cont,filtro,0x1E,motor0);
        dhl(cont,filtro,0x1E,motor1);       /*Se envian valores del filtro*/
        while( inportb(motor0) & mascara);outportb(motor0,0x04);
        while( inportb(motor1) & mascara);          outportb(motor1,0x04);
                                                    /*Se actualizan par metros del filtro*/

        /*Se carga primera trayectoria en modo de velocidad:          */
        printf("\ntrayectoria hacia primer paro\n");
        aux=0x08; /*Primer byte de control,indica:sentido negativo, modo de velocidad y parar suavemente*/
                        /*bit 15 al 8, 00001000   bit12 dir_pos,bit11_vel_mode, bit10_stop_smooth*/

        filtro[0]=256*aux;
        filtro[0]=filtro[0]+0x28;           /*Se indica cargar aceleración y velocidad*/

        filtro[1]=1; /* Aceleración de  1 pulso/Ts/Ts*/
        filtro[2]=0;
        filtro[3]=0x6a;         /* Velocidad de aprox. 16384 pulsos/seg -1 vuelta de eje-*/
        filtro[4]=0;
        cont=5;
        dhl(cont,filtro,0x1F,motor0);
        dhl(cont,filtro,0x1F,motor1); /* Se mandan valores de trayectoria  */
        while(inportb(motor0) & mascara);outportb(motor0,0x01);
        while(inportb(motor1) & mascara);outportb(motor1,0x01);          /*se actualizan los valores...*/
                                                    /*...de vel. y acel. al ordenarse iniciar el movimiento*/
        printf("\nel puerto C lee \n %x\n",inportb(0x202));
        mensajes(mensaje14);

        /*Primer paro al llegar a interruptores:          */
        while( bandera_elevacion || bandera_ronza )
        {
                elevacion= (inportb(0x202) & 0x0C); /*Verificar si es necesario..*/
                ronza= (inportb(0x202) & 0x03);/*...el puerto C -registro 0x202-   */
                if( elevacion)
                {   while( inportb(motor1) & mascara);
                        parar(motor1);
                        define_casa(motor1);
                        if(bandera_elevacion==1)
```

```
                        {
                        mensajes(mensaje18);
                        }
                        bandera_elevacion=0;
                        /*printf("\nelev %x",elevacion);*/

                };
                if( ronza )

                {   while ( inportb(motor0) & mascara );
                        parar(motor0);
                        define_casa(motor0);

                        if(bandera_ronza==1)
                        {
                        mensajes(mensaje17) ;
                        }
                        bandera_ronza=0;
                /*      printf("\nronz %x",ronza);*/
                };
                delay(150);
};

/*Programamos iniciar el movimiento en sentido contrario -positivo- */
/*Se carga segunda trayectoria en modo de velocidad:          */
printf("trayectoria2");
aux=0x18; /*Primer byte de control,indica:sentido positivo, modo de velocidad y parar suavemente*/
                        /*bit 15 al 8, 00011000   bit12 dir_pos,bit11_vel_mode, bit10_stop_smooth*/

filtro[0]=aux*256;
cont=1;
dhl(cont,filtro,0x1F,motor0);
dhl(cont,filtro,0x1F,motor1); /* Se mandan valores de trayectoria  */
while( inportb(motor0) & mascara);
outportb(motor0,0x01);
delay(50);
while( inportb(motor1) & mascara);
outportb(motor1,0x01);
                                                        /*y se da orden de iniciar el movimiento   */

delay(20000);  /*le damos dos segundo para abrir los interruptores*/
/*Segundo paro al llegar a interruptores:                   */
bandera_elevacion=1;
bandera_ronza=1;
while(bandera_elevacion || bandera_ronza)
{
                elevacion= inportb(0x202) & 0x0C; /*Verificar si es necesario..*/
                ronza= inportb(0x202) & 0x03;/*...el puerto C -registro 0x202-   */
                if(elevacion)
                {               while( inportb(motor1) & mascara);
                                parar(motor1);
                                if(bandera_elevacion==1)
                                mensajes(mensaje17);
                                bandera_elevacion=0;
                };
                if(ronza)
                {
                                while( inportb(motor0) & mascara);
                                parar(motor0);
                                if(bandera_ronza==1)
                                mensajes(mensaje18);
                                bandera_ronza=0;
                };
                delay(150);
 };


printf("\nconociendo posiciones...\n");
while(inportb(motor0) & mascara);
outportb(motor0,0x0A);     /*Se conocer  el recorrido del eje 0 -ronza-*/
```

```
recibe(motor0+1,&byte4,&byte3,&byte1,&byte2);
posicion0=convierte(byte4,byte3,byte2,byte1);

while( inportb(motor1) & mascara);
outportb(motor1,0x0A);    /*Se conocer  el recorrido del eje1 -elevaci¢n-*/
recibe(motor1+1,&byte4,&byte3,&byte1,&byte2);
posicion1=convierte(byte4,byte3,byte2,byte1);

printf("\n recorrido en ronza=%li ",posicion0);
printf("\n recorrido en elevacion=%li",posicion1);
/*Se carga trayectoria del motor0:            */

        aux=0x00; /*Primer byte de control, ning—n bit alto */
                        /*bit 15 al 8, 00001100   bit12 dir_pos,bit11_vel_mode, bit10_stop_smooth*/

        filtro[0]=256*aux;
        filtro[0]=filtro[0]+0x02;           /*Se indica cargar posicion con ese 2*/
        auxiliar=posicion0/2;
        filtro[1]=auxiliar/65536;           /*palabra m s significativo*/
        filtro[2]=auxiliar;       /*palabra menos significativa*/
        cont=3;
        printf("\ntercer trayectoria motor 0");
        dhl(cont,filtro,0x1F,motor0); /* Se mandan valores de trayectoria  */

/*Se carga trayectoria del motor1:            */
        aux=0x00; /*Primer byte de control ning£n bit alto*/
                        /*bit 15 al 8, 00000000   bit12 dir_pos,bit11_vel_mode, bit10_stop_smoothly*/
        filtro[0]=256*aux;
        filtro[0]=filtro[0]+2;   /*Se indica cargar posicion*/
        auxiliar=posicion1/2;
        filtro[1]=auxiliar/65536;           /*palabra m s significativo*/
        filtro[2]=auxiliar;       /*palabra menos significativa*/
        cont=3;
        printf("\ntercer trayectoria motor 1");
        dhl(cont,filtro,0x1F,motor1); /* Se mandan valores de trayectoria  */

        while( inportb(motor0) & mascara);
        outportb(motor0,0x01);
        while( inportb(motor1) & mascara);
        outportb(motor1,0x01);

        delay(10000);           /*se actualizan los valores...*/
                                                /*...de vel. y acel. al ordenarse iniciar el movimiento*/
        mensajes(mensaje13); /*mensaje de inicializando*/
        ver_pos();
/*      limpia_status(motor0);
        limpia_status(motor1) ;*/
                        /*bajamos los bits de trayectoria completa*/
 /*Verificar cu ndo llegan los ejes a la posici¢n de inicializaci¢n*/
bandera_elevacion=1;
bandera_ronza=1;
        while(bandera_elevacion || bandera_ronza)
        {
                eje0_ronza_flag=inportb(motor0) & 0x04;
                eje1_elevacion_flag=inportb(motor1) & 0x04;
                if(eje0_ronza_flag)
                {
                        if(bandera_ronza)
                        mensajes(mensaje15);
                        bandera_ronza=0;
                };
                if(eje1_elevacion_flag)
                {
                        if(bandera_elevacion)
                        mensajes(mensaje16);
                        bandera_elevacion=0;
                };
         };
printf("\n recorrido en ronza=%li ",posicion0);
printf("\n recorrido en elevacion=%li",posicion1);
```

```
}

/*************************************************************************/
void mensajes (char mensaje[])
{
int i=0;

while(mensaje[i] != '$')
                {
                while(!(inportb(0x3fd) & 0x20));
                outportb(0x3f8,mensaje[i]);
                i++;
                } ;
}

/*********************************************************************/
void envia_pos(long int posicion ,int eje ,char mensaje_pos[])
{
unsigned int pos_escalada;
int i,aux1;
char aux2;
if (eje==0)
pos_escalada= (posicion/747.2);/* pensando en 26,900,000 pulsos por vuelta/ronza*/
if(eje==1)
pos_escalada=(posicion/1179.1666); /*pensando en 14,150,000 pulsos/elevacion=120grados*/

for(i=0;i<=4;i++)
  {
   aux1=pos_escalada/pow10(4-i);
   pos_escalada= pos_escalada - aux1*pow10(4-i);   aux2=aux1+0x30;

   if(i < 3)
                mensaje_pos[i+1]=aux2;
   if(i >= 3)
                mensaje_pos[i+2]=aux2;


  };

  if (eje==0)
  mensaje_pos[0]='#';
  if(eje==1)
  mensaje_pos[0]='%';
  mensaje_pos[4]='.';
  mensaje_pos[7]=13;
  mensaje_pos[8]='$';
  mensajes(mensaje_pos);
}

/*********************************************************************/
void zoom(char mensaje_pos[])
{
  int conversion,i,aux1;
  unsigned char lectura,comando;
  char aux2;
  comando=0x12; /*byte de comando que se modifica dependiendo del canal...*/

  outportb(base+2,comando);     /*...mandamos el comando...*/
  delay(20);                    /*...tiempo para que se recupere el puerto I/O.*/
  outportb(base,0x00);     /*Que empiece la conversi¢n*/
  delay(20);                                    /*Tiempo de 1ms (para la advantech 20 cuentas son..*/
                                                /*..apenas 1ms)para terminar la conversi¢n */

  lectura=inportb(base);  /*leemos la conversi¢n*/
  conversion=lectura;
  conversion=250-conversion;
  pos_zoom=conversion;
for(i=0;i<=2;i++)
```

```
  {
   aux1=conversion/pow10(2-i);
   conversion=conversion - aux1*pow10(2-i);
   aux2=aux1+0x30;
   if (aux2=='/')
   aux2='0';
   mensaje_pos[i+1]=aux2;
    };

  mensaje_pos[0]='&';
  mensaje_pos[4]=13;
  mensaje_pos[5]='$';
  mensajes(mensaje_pos);
}

/**********************************/
void joy ()
{
  int joy_elev,joy_ronza,i,cont;
  unsigned char lectura,comando,aux1;
  unsigned int aux3,filtro[7];
  char aux2;
  joystick_on=1;

  while (joystick_on )
  {
            /*****leemos canal 2 =joystick elev*/
            comando=0x14; /*byte de comando que se modifica dependiendo del canal...*/
            outportb(base+2,comando);     /*...mandamos el comando...*/
            delay(20);                    /*...tiempo para que se recupere el puerto I/O.*/
            outportb(base,0x00);     /*Que empiece la conversi¢n*/
            delay(20);                    /*Tiempo de 1ms (para la advantech 20 cuentas son..*/
                                          /*..apenas 1ms)para terminar la conversi¢n */
            lectura=inportb(base);  /*leemos la conversi¢n*/
            joy_elev=lectura;
            /*****leemos canal 3 =joystick ronza*/

            comando=0x16; /*byte de comando que se modifica dependiendo del canal...*/

            outportb(base+2,comando);     /*...mandamos el comando...*/
            delay(20);                    /*...tiempo para que se recupere el puerto I/O.*/
            outportb(base,0x00);     /*Que empiece la conversi¢n*/
            delay(20);                    /*Tiempo de 1ms (para la advantech 20 cuentas son..*/
                                          /*..apenas 1ms)para terminar la conversi¢n */

            lectura=inportb(base);  /*leemos la conversi¢n*/
            joy_ronza=lectura;

            /**********************para ronza*/
            if( (joy_ronza>152) && (joy_ronza<160) )
                        parar(motor0);
            else {
                                    if(joy_ronza >=160)
                                                aux3=0x18; /*direcci•n positiva en modo velocidad*/
                                    else
                                                aux3=0x08; /*direcci•n negativa de velocidad*/

                                    filtro[0]=aux3*256;
                                    filtro[0]=filtro[0]+0x08;   /*se indica s•lo cargar velocidad*/
                                    filtro[1]=144-0.6218*pos_zoom;
                                    filtro[2]=0;
                                    cont=3;
                                    dhl(cont,filtro,0x1F,motor0);

                         };

/**********************para elevacion*/

            if( (joy_elev>150) && (joy_elev<158) )
                        parar(motor1);
```

```
        else {
                        if(joy_elev >=158)
                                aux3=0x18; /*direcci•n positiva en modo velocidad*/
                        else
                                aux3=0x08; /*direcci•n negativa en modo velocidad*/

                        filtro[0]=aux3*256;
                        filtro[0]=filtro[0]+0x08;   /*se indica s•lo cargar velocidad*/
                        filtro[1]=138-0.5323*pos_zoom;
                        filtro[2]=0;
                        cont=3;
                        dhl(cont,filtro,0x1F,motor1);

                };
                muevete(motores);
        if ( inportb(0x3fd) & 0x01)
          {
                aux1=inportb(0x3f8);
                if (aux1=='J')
                joystick_on=0;
          } ;

          ver_pos();
          delay(100);
          zoom(mensaje_pos);

                }


}□
```

CAM_VIG_1.0

```
unit DemoCam;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  CommInt, StdCtrls, Buttons, Spin, ExtCtrls, Gauges, Menus, ComCtrls,DFG_LC1,
  Mask,OleCtrls;


type
  TForm1 = class(TForm)
    Comm1: TComm;
    Timer1: TTimer;
    BitBtn2: TBitBtn;
    Panel1: TPanel;
    Edit1: TEdit;
    Edit3: TEdit;
    Label2: TLabel;
    Label5: TLabel;
    Panel3: TPanel;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit7: TEdit;
    Edit8: TEdit;
    Label6: TLabel;
    Label7: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label14: TLabel;
    Label15: TLabel;
    RadioGroup1: TRadioGroup;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    BitBtn4: TBitBtn;
    RadioGroup2: TRadioGroup;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn1: TBitBtn;
    Label8: TLabel;
    Label11: TLabel;
    Image1: TImage;
    Label12: TLabel;
    Label13: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Image2: TImage;
    Label18: TLabel;
    Label19: TLabel;
    Label20: TLabel;
    Panel5: TPanel;
    Label23: TLabel;
    ScrollBar1: TScrollBar;
    Edit6: TEdit;
    Button1: TButton;
    Panel6: TPanel;
    ScrollBar2: TScrollBar;
    Edit9: TEdit;
    Button2: TButton;
    Label25: TLabel;
    ListBox1: TListBox;
    Label21: TLabel;
    BitBtn7: TBitBtn;
    Memo2: TMemo;
    BitBtn3: TBitBtn;
    Edit2: TEdit;
```

```
    label1: TLabel;
    Label3: TLabel;
    PaintBox1: TPaintBox;
    UpDown1: TUpDown;
    Edit10: TEdit;
    UpDown2: TUpDown;
    UpDown3: TUpDown;
    UpDown4: TUpDown;
    UpDown5: TUpDown;
    Edit11: TEdit;
    Edit12: TEdit;
    Edit13: TEdit;
    Edit14: TEdit;
    Label4: TLabel;
    procedure Comm1RxChar(Sender: TObject; Count: Integer);
    procedure FormCreate(Sender: TObject);
    procedure Comm1Dsr(Sender: TObject);
    procedure Memo2KeyPress(Sender: TObject; var Key: Char);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure RadioButton2Click(Sender: TObject);
    procedure RadioButton1Click(Sender: TObject);
    procedure RadioButton4Click(Sender: TObject);
    procedure RadioButton3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure Edit4Change(Sender: TObject);
    procedure Edit5Change(Sender: TObject);
    procedure Edit7Change(Sender: TObject);
    procedure Edit8Change(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure ListBox1DblClick(Sender: TObject);
    procedure ScrollBar1Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure ScrollBar2Change(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure BitBtn7Click(Sender: TObject);
    procedure UpDown1Changing(Sender: TObject; var AllowChange: Boolean);
    procedure UpDown2Changing(Sender: TObject; var AllowChange: Boolean);
    procedure UpDown3Changing(Sender: TObject; var AllowChange: Boolean);
    procedure UpDown4Changing(Sender: TObject; var AllowChange: Boolean);
    procedure UpDown5Changing(Sender: TObject; var AllowChange: Boolean);
    procedure Timer1Timer(Sender: TObject);
    procedure PaintBox1MouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure PaintBox1DblClick(Sender: TObject);
  private
    { Private declarations }
    LineData: string;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  CONT_MEMO2,paintbox_x,paintbox_y:INTEGER;
  pos0_cambiada,vel0_cambiada,pos1_cambiada,vel1_cambiada : boolean;
   a_ronza,a_elevacion,x_x,y_y,x_xx,y_yy:integer;
  coord_x,coord_y:variant;
  ronza,elevacion:real;

implementation

Uses UnitGrabber;

{$R *.DFM}
Var   Grab        : TGrab;
      Brillo,
```

```
    Saturacion_U,
    Saturacion_V,
    Contraste,
    Tinte       : Integer;

const OnOff: array[0..1] of string = ('Off', 'On');

procedure TForm1.FormCreate(Sender: TObject);
  begin

  Grab:=TGrab.Create;
  Grab.Init;
  Brillo := IS_DEFAULT_BRIGHTNESS;
  UPDOWN1.Position := round(100*Brillo/IS_MAX_BRIGHTNESS);
  EDIT10.TEXT :='B:' + IntToStr(UPDOWN1.Position);
  Grab.Brillo(50);

  Tinte := IS_DEFAULT_HUE;
  UPDOWN2.Position := round(100*Tinte/IS_MAX_HUE);
  EDIT11.TEXT := 'T:' + IntToStr(UPDOWN2.Position);
  Grab.Tinte(128);

  Saturacion_U := IS_DEFAULT_SATURATION_U;
  Saturacion_V := IS_DEFAULT_SATURATION_V;
  UPDOWN3.Position := round(100*Saturacion_U/IS_MAX_SATURATION_U);
  EDIT12.TEXT:= 'SU:' + IntToStr(UPDOWN3.Position);
  UPDOWN4.Position := round(100*Saturacion_V/IS_MAX_SATURATION_V);
  EDIT13.TEXT:='SV:' + IntToStr(UPDOWN4.Position);
  Grab.Saturacion(254, 180);

  Contraste := IS_DEFAULT_CONTRAST;
  UPDOWN5.Position := round(100*Contraste/IS_MAX_CONTRAST);
  EDIT14.TEXT :='C:' + IntToStr(UPDOWN5.Position);
  Grab.Contraste(216);


  Timer1.Enabled := True;
   Comm1.Open;

  end;

procedure TForm1.Comm1RxChar(Sender: TObject; Count: Integer);
  type CharBuf = array[0..9999] of Char;
  var Buffer: ^CharBuf;
     Bytes, P,contador: Integer;

//       posicion_ronza,posicion_elevacion,fantasma:string;
  begin
  for contador := 2 to 100000 do
        p:=0;
    GetMem(Buffer, Comm1.ReadBufSize);
    try
//       Memo1.Lines.add('RxChar signal detected...');
      Fillchar(Buffer^, Comm1.ReadBufSize, 0);

      Bytes := Comm1.Read(Buffer^, Count);
       if Bytes = -1 then
    Memo2.Lines.add('Error reading incoming data...')
  else
  begin
  //  Memo2.Lines.add('Reading ' + IntToStr(Bytes) + ' characters');
    for P := 0 to Bytes - 1 do
    begin
      case Buffer^[P] of

        #0, #10:;
       #35:  begin
        a_ronza:=69;
            end;
       #37: a_elevacion:=69;
```

```
 #13: begin
 //      edit1.text:=(linedata);
   //    fantasma:=edit1.text;
 //        if buffer^[0]='#'then
  //            begin
  //            for contador:=0 to 6 do
    //          posicion_ronza[contador]:=charbuf(contador+1);
      //        edit1.text:=posicion_ronza;
//            edit1.text:=(linedata);

 //             end
   //     else  begin

   if a_ronza = 69 then
   begin
   edit1.text:=(linedata);
   a_ronza:=0;
   linedata:='';

   with image1 do begin
   canvas.pen.width:=10;
   canvas.pen.color:=clblue;
   //canvas.brush.style:=bsclear;
   Canvas.Ellipse(10, 10, 90, 90);

   canvas.pen.width:=10;
   canvas.pen.color:=clwhite;
   canvas.moveto(50,50) ;
   canvas.lineto(x_x,y_y);

   ronza:=strtofloat(edit1.text) ;
   coord_x:= 40*cos(ronza/57.29);
   coord_y:= 40*sin(ronza/57.29);
   x_x:= 50+coord_x;
   y_y:= 50+coord_y ;
   canvas.Pen.style:=pssolid;
    canvas.pen.color:=clblack;
   canvas.lineto(x_x,y_y);
                     end;
     exit;
   end;

   if a_elevacion = 69 then
   begin
   edit3.text:= (linedata);
   a_elevacion:=0;
   linedata:='';

     with image2 do begin
   canvas.pen.width:=10;
   canvas.pen.color:=clred;
   //canvas.brush.style:=bsclear;
   Canvas.arc(10, 10, 90, 90,30,16,30,84);

   canvas.pen.width:=10;
   canvas.pen.color:=clwhite;
   canvas.moveto(50,50) ;
   canvas.lineto(x_xx,y_yy);

   elevacion:=strtofloat(edit3.text) ;
   coord_x:= 40*cos((elevacion+120)/57.29);
   coord_y:= 40*sin((elevacion+120)/57.29);
   x_xx:= 50+coord_x;
   y_yy:= 50+coord_y ;
   canvas.Pen.style:=pssolid;
    canvas.pen.color:=clblack;
   canvas.lineto(x_xx,y_yy);

                     end;
   exit;
```

```
          end;

          Memo2.Lines[0]:=(LineData);
          LineData := '';

            end;

           //      if buffer^[0]='%'then
             //        begin
       //               for contador:=0 to 6 do
         //             posicion_elevacion[contador]:=charbuf(contador+1);
           //           edit3.text:=posicion_elevacion;
       //               edit2.text:=(linedata);
         //            end
           //      else  begin
             //             Memo2.Lines[0]:=(LineData);
               //           LineData := '';
                 //     end;

          else
            begin

             LineData := LineData + CharBuf(Buffer^)[P];
            end;
       end; //case
     end; //for do
   end;

      Application.ProcessMessages;
    finally
    FreeMem(Buffer);
   end;

 End;

 procedure TForm1.Comm1Dsr(Sender: TObject);
   begin
     Memo2.Lines.add('DSR: ' + OnOff[ord(Comm1.DSR)]);
   end;

 procedure TForm1.Memo2KeyPress(Sender: TObject; var Key: Char);
   var S: string;
       Count: Integer;
   begin
     If Key = #13
       Then Begin
             S := Memo2.Lines[Memo2.Lines.Count-1];
             //S := S + #13#10;
     Count := Length(S);
     Count := Comm1.Write(S[1], Count);
   end;
 end;


 procedure TForm1.SpeedButton1Click(Sender: TObject);
   var S: string;
       Count: Integer;
   begin
     S :=listbox1.items[listbox1.itemindex];
//      S := S + #13#10;
     Count := Length(S);
     Count := Comm1.Write(S[1], Count);
   end;

procedure TForm1.BitBtn1Click(Sender: TObject);
   var posicion,velocidad,pos,vel:variant;
       poss,vell,comando:STRING;
       contador,auxiliar,aux_cont: Integer;
begin           //b1
comando:='T0';
```

```
IF vel0_cambiada =true
THEN BEGIN        //b2
       velocidad:=strtofloat(edit5.tExt);
       if velocidad > 7.5   //MÁXIMI REAL LIMITADO POR LA ALIMENTACION DE 12 VOLTS
      then begin       //b3
       Application.MessageBox(
     'Valor fuera de rango: 0<velocidad<7.5°/seg',
     'ERROR DE RANGO DE VELOCIDAD',
     MB_OKCANCEL );
     BitBtn1.enabled:=false;
      EXIT;
          end;         //e3

      if velocidad <0
        then begin     //b4
        Application.MessageBox(
     'Valor fuera de rango: 0<velocidad<7.5°/seg',

     'ERROR DE RANGO',
     MB_OKCANCEL );
     BitBtn1.enabled:=false;
      EXIT;

          end;             //e4
      velocidad:=velocidad*19.2;
      auxiliar:=velocidad;
      vell:=inttostr(auxiliar);
      comando :=comando+'V' ;
        contador:=3-length(vell);
        for aux_cont:=1 to contador do
        comando:=comando+'0';
        comando:=comando+ vell;
        bitbtn4.enabled:=true;

  END;              //e2
       posicion:=strtofloat(edit4.text);
      IF (pos0_cambiada=true and radiobutton2.checked= true) THEN
  BEGIN                 //b4

     if posicion > 358
     then begin            //b5
      Application.MessageBox(
    'Valor fuera de rango:  0<posicion<358°',
    'ERROR DE RANGO DE POSICIÓN ABS.',
    MB_OKCANCEL );
     EXIT;
          end;               //e5

     if posicion <0
        then begin               //b6
        Application.MessageBox(
    'Valor fuera de rango: 0<posicion<355°',
    'ERROR DE RANGO DE POSICION ABS.',
    MB_OKCANCEL );
     EXIT;

          end;                     //e6
        comando :=comando+'PA+';

     END;     //e4

       if (pos0_cambiada=true and radiobutton1.checked= true) then
           BEGIN  //b7
            if (posicion+ronza) > 358
           then begin //b8
           Application.MessageBox(
           'Valor fuera de rango:  0<posicion<358°',

           'ERROR DE RANGO DE POSICIÓN',
```

```
            MB_OKCANCEL );
             EXIT;
             end;       //e8
            if (posicion+ronza) < 0
             then begin      //b9
            Application.MessageBox(
            'Valor fuera de rango:  0<posicion<358°',
            'ERROR DE RANGO DE POSICIÓN RElativa',
              MB_OKCANCEL );
             EXIT;
             end;               //e9

            if posicion < 0 then
            comando:=comando+'PR-'
            else
            comando :=comando+'PR+';
            END ;                   //e7

          if posicion < 0 then
          posicion:=-posicion*74700
          else
          posicion:=posicion*74700;
          poss:=posicion;

          contador:=8-length(poss);
          for aux_cont:=1 to contador do
          comando:=comando+'0';
          comando:=comando+ poss;
           bitbtn4.enabled:=true;

            listbox1.items[cont_memo2]:=comando;
            CONT_MEMO2:=CONT_MEMO2 + 1;
           contador:= Length(comando);
            contador:= Comm1.Write(comando[1], contador);

            //conversiones y  enviar
            pos0_cambiada:=false;
            vel0_cambiada:=false;
            BitBtn1.enabled:=false;

    END;    //e1

procedure TForm1.SpeedButton2Click(Sender: TObject);
  var S: string;
       Count: Integer;
begin
     S := 'R2';
//      S := S + #13#10;
     Count := Length(S);
     Count := Comm1.Write(S[1], Count);
   end;

procedure TForm1.BitBtn2Click(Sender: TObject);
 var S: string;
       Count: Integer;
begin

     S :=EDIT2.TEXT;
     listbox1.items[cont_memo2]:=edit2.text;

     Count := Length(S);
     Count := Comm1.Write(S[1], Count);

     EDIT2.TEXT:='';
     CONT_MEMO2:=CONT_MEMO2 + 1;
end;

  procedure TForm1.RadioButton2Click(Sender: TObject);
begin
edit4.enabled:=true;
```

```
edit5.enabled:=true;
label6.hint:='Mover eje ronza a pos.abs. +XXX.XXX grados';
edit4.text:='+XXX.XXX';
BitBtn1.enabled:=false;
end;

procedure TForm1.RadioButton1Click(Sender: TObject);
begin
edit4.enabled:=true;
edit5.enabled:=false;
label6.hint:='Mover el eje ronza ±XXX.XXX grados ';
edit4.text:='±XXX.XXX';
BitBtn1.enabled:=false;
end;

procedure TForm1.RadioButton4Click(Sender: TObject);
begin
edit7.enabled:=true;
edit8.enabled:=true;
label9.hint:='Mover eje eevación a pos.abs. +XXX.XXX grados';
edit7.text:='+XXX.XXX' ;
BitBtn5.enabled:=false;
end;

procedure TForm1.RadioButton3Click(Sender: TObject);
begin
edit7.enabled:=true;
edit8.enabled:=false;
label9.hint:='Mover el eje elevación ±XXX.XXX grados ';
edit7.text:='±XXX.XXX';
 BitBtn5.enabled:=false;
end;

procedure TForm1.BitBtn4Click(Sender: TObject);
VAR CADENA:STRING;
      CONTADOR: Integer;
begin
     CADENA :='M0';
     CONTADOR:= Length(CADENA);
     CONTADOR:= Comm1.Write(CADENA[1], CONTADOR);
     BitBtn4.enabled:=false;
end;

procedure TForm1.Edit4Change(Sender: TObject);
begin
pos0_cambiada:=true;
BitBtn1.enabled:=true;
end;

procedure TForm1.Edit5Change(Sender: TObject);
begin
vel0_cambiada:=true;
BitBtn1.enabled:=true;
end;

procedure TForm1.Edit7Change(Sender: TObject);
begin
pos1_cambiada:=true;
bitbtn5.enabled:=true;
end;

procedure TForm1.Edit8Change(Sender: TObject);
begin
vel1_cambiada:=true;
bitbtn5.enabled:=true;
end;

  procedure TForm1.BitBtn5Click(Sender: TObject);

  var posicion,velocidad,pos,vel:variant;
```

```
      poss,vell,comando:STRING;
      contador,auxiliar,aux_cont: Integer;

 begin           //b1
comando:='T1';

 IF vel1_cambiada =true
 THEN BEGIN        //b2
       velocidad:=strtofloat(edit8.tExt);
       if velocidad > 4.7   //MÁXIMO falso
      then begin       //b3
       Application.MessageBox(
     'Valor fuera de rango: 0<velocidad de elev.<4.7°/seg',
     'ERROR DE RANGO DE VELOCIDAD DE ELEV.',
     MB_OKCANCEL );
     BitBtn5.enabled:=false;
      EXIT;
          end;         //e3
      if velocidad <0
        then begin     //b4
        Application.MessageBox(
     'Valor fuera de rango: 0<velocidad de elev.<4.7°/seg',
     'ERROR DE RANGO DE VEL. DE ELEV.',
     MB_OKCANCEL );
     BitBtn5.enabled:=false;
      EXIT;

           end;            //e4
      velocidad:=velocidad*30.63; //PORQUE 144/4.7=30.63
      auxiliar:=velocidad;
      vell:=inttostr(auxiliar);
      comando :=comando+'V' ;
       contador:=3-length(vell);
       for aux_cont:=1 to contador do
       comando:=comando+'0';
       comando:=comando+ vell;
       bitbtn6.enabled:=true;
   END;             //e2

       posicion:=strtofloat(edit7.text);
      IF (pos1_cambiada=true and radiobutton4.checked= true) THEN
   BEGIN               //b4

      if posicion > 120
      then begin           //b5
       Application.MessageBox(
     'Valor fuera de rango:  0<posicion<120°',
     'ERROR DE RANGO DE POSICIÓN ABS.',
     MB_OKCANCEL );
      EXIT;
          end;             //e5

      if posicion <0
        then begin             //b6
        Application.MessageBox(
     'Valor fuera de rango: 0<posicion<120º',
     'ERROR DE RANGO DE POSICION ABS.',
     MB_OKCANCEL );
      EXIT;
          end;                 //e6
         comando :=comando+'PA+';

      END;     //e4

       if (pos1_cambiada=true and radiobutton3.checked= true) then
           BEGIN  //b7
            if (posicion+elevacion) > 120
           then begin //b8
           Application.MessageBox(
          'Valor fuera de rango:  0<posicion de elev.<120°',
```

```
            'ERROR DE RANGO DE POSICIÓN DE ELEVACIÓN',
            MB_OKCANCEL );
             EXIT;
             end;       //e8
             if (posicion+elevacion) < 0
             then begin      //b9
            Application.MessageBox(
            'Valor fuera de rango:  0<posicion de elev.<120°',

            'ERROR DE RANGO DE POSICIÓN RELATIVA',
              MB_OKCANCEL );
             EXIT;
             end;               //e9

            if posicion < 0 then
            comando:=comando+'PR-'
            else
            comando :=comando+'PR+';

            END ;                   //e7
          if posicion < 0 then
          posicion:=-posicion*118000//pues 14,160,000/120=118,000
          else
          posicion:=posicion*118000;
          poss:=posicion;

          contador:=8-length(poss);
          for aux_cont:=1 to contador do
          comando:=comando+'0';
          comando:=comando+ poss;
           bitbtn6.enabled:=true;

            listbox1.items[cont_memo2]:=comando;
            CONT_MEMO2:=CONT_MEMO2 + 1;
           contador:= Length(comando);
            contador:= Comm1.Write(comando[1], contador);

            //conversiones y  enviar
            pos1_cambiada:=false;
            vel1_cambiada:=false;
            BitBtn5.enabled:=false;

   END;    //e1

procedure TForm1.BitBtn6Click(Sender: TObject);
VAR CADENA:STRING;
      CONTADOR: Integer;
begin
  CADENA :='M1';
     CONTADOR:= Length(CADENA);
     CONTADOR:= Comm1.Write(CADENA[1], CONTADOR);
end;

procedure TForm1.ListBox1DblClick(Sender: TObject);
begin
edit2.text :=listbox1.items[listbox1.itemindex];
end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
edit6.text:=inttostr(scrollbar1.position)+'°';
button1.enabled:=true;
end;

procedure TForm1.Button1Click(Sender: TObject);

var    posicion:variant;
       poss,comando:STRING;
       contador,aux_cont: Integer;
begin
```

```
 comando:='T0';
  comando :=comando+'PA+';

  posicion:=scrollbar1.position;
  posicion:=posicion*74700;
  poss:=posicion;

  contador:=8-length(poss);
          for aux_cont:=1 to contador do
          comando:=comando+'0';
          comando:=comando+ poss;

  listbox1.items[cont_memo2]:=comando;
  CONT_MEMO2:=CONT_MEMO2 + 1;
  contador:= Length(comando);
  contador:= Comm1.Write(comando[1], contador);

  //**********************
for contador:=1 to 10000000 do
aux_cont:=1;
 comando:='M0';
   listbox1.items[cont_memo2]:=comando;
   CONT_MEMO2:=CONT_MEMO2 + 1;
 contador:= Length(comando);
  contador:= Comm1.Write(comando[1], contador);

   button1.enabled:=false;
end;

procedure TForm1.ScrollBar2Change(Sender: TObject);
begin
edit9.text:=inttostr(scrollbar2.position)+'°';
button2.enabled:=true;
end;

procedure TForm1.Button2Click(Sender: TObject);
var     posicion:variant;
        poss,comando:STRING;
        contador,aux_cont: Integer;
begin

 comando:='T1';
  comando :=comando+'PA+';

  posicion:=scrollbar2.position;
  posicion:=posicion*118000;
  poss:=posicion;

  contador:=8-length(poss);
          for aux_cont:=1 to contador do
          comando:=comando+'0';
          comando:=comando+ poss;

  listbox1.items[cont_memo2]:=comando;
  CONT_MEMO2:=CONT_MEMO2 + 1;
  contador:= Length(comando);
  contador:= Comm1.Write(comando[1], contador);
  //**********************
for contador:=1 to 10000000 do
aux_cont:=1;
 comando:='M1';
   listbox1.items[cont_memo2]:=comando;
   CONT_MEMO2:=CONT_MEMO2 + 1;
 contador:= Length(comando);
  contador:= Comm1.Write(comando[1], contador);
   button2.enabled:=false;
end;
procedure TForm1.BitBtn7Click(Sender: TObject);
    var S: string;
        Count: Integer;
```

```
   begin
     S :=listbox1.items[listbox1.itemindex];
//     S := S + #13#10;
     Count := Length(S);
     Count := Comm1.Write(S[1], Count);
   end;

procedure TForm1.UpDown1Changing(Sender: TObject;
  var AllowChange: Boolean);
begin

EDIT10.TEXT:='B:'+INTTOSTR(UPDOWN1.POSITION);
Brillo := round(IS_MAX_BRIGHTNESS*(UPDOWN1.Position/100));
Grab.Brillo(Brillo);
end;

procedure TForm1.UpDown2Changing(Sender: TObject;
  var AllowChange: Boolean);
begin
 EDIT11.TEXT:='T:'+INTTOSTR(UPDOWN2.POSITION);
 Tinte := round(IS_MAX_HUE*(UPDOWN2.Position/100));
 Grab.Tinte(tinte);
end;

procedure TForm1.UpDown3Changing(Sender: TObject;
  var AllowChange: Boolean);
begin
     EDIT12.TEXT:='SU:'+INTTOSTR(UPDOWN3.POSITION);
    Saturacion_U := round(IS_MAX_SATURATION_U*(UPDOWN3.Position/100));
   Grab.Saturacion(Saturacion_U, Saturacion_V);
end;

procedure TForm1.UpDown4Changing(Sender: TObject;
  var AllowChange: Boolean);
begin
 EDIT13.TEXT:='SV:'+INTTOSTR(UPDOWN4.POSITION);
 Saturacion_V := round(IS_MAX_SATURATION_V*(UPDOWN4.Position/100));
 Grab.Saturacion(Saturacion_U, Saturacion_V);
end;

procedure TForm1.UpDown5Changing(Sender: TObject;
  var AllowChange: Boolean);
begin
 EDIT14.TEXT:='C:'+INTTOSTR(UPDOWN5.POSITION);
 Contraste := round(IS_MAX_CONTRAST*(UPDOWN5.Position/100));
   Grab.Contraste(Contraste);
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
 Grab.Display(0,0,PaintBox1.Canvas);
  Label4.Invalidate;
end;

procedure TForm1.PaintBox1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  Label4.Caption:=IntToStr(X)+','+IntToStr(Y);
   Label4.Top:=Y+56+5;
   Label4.Left:=X+392+5;
   paintbox_x:=X;
   paintbox_y:=Y;
end;

procedure TForm1.PaintBox1DblClick(Sender: TObject);

var
poss,comando:STRING;
posicion,velocidad,pos,vel:variant;
contador,auxiliar,aux_cont,ii: Integer;
```

```
begin
 //******************  Para ronza:

posicion:=(paintbox_x-170)/21;  // (dif_de_pixeles)/pixeles_por_grado
 if (posicion+ronza) > 358
              then begin //b8
              Application.MessageBox(
              'Valor fuera de rango:  Ese punto no se puede centrar',
              'ERROR DE RANGO DE CENTRO',
              MB_OKCANCEL );
               EXIT;
               end;       //e8

if (posicion+ronza) < 0
               then begin      //b9
              Application.MessageBox(
              'Valor fuera de rango:  Ese punto no se puede centrar',
              'ERROR DE RANGO DE CENTRO',
                MB_OKCANCEL );
               EXIT;
               end;               //e9

comando:='T0' ;

 if posicion < 0 then
              comando:=comando+'PR-'
              else
              comando :=comando+'PR+';
 if posicion < 0 then
          posicion:=-posicion*74700
          else
          posicion:=posicion*74700;
          posicion:=int(posicion);
          poss:=posicion;

contador:=8-length(poss);
          for aux_cont:=1 to contador do
          comando:=comando+'0';
          comando:=comando+ poss;
            listbox1.items[cont_memo2]:=comando;
             CONT_MEMO2:=CONT_MEMO2 + 1;

 contador:= Length(comando);
            contador:= Comm1.Write(comando[1], contador);

  for ii:=10000000 downto 0 do
  auxiliar:=100;

//**************************** Para elevacion:
posicion:=(paintbox_y-150)/14;  // (dif_de_pixeles)/pixeles_por_grado
 if (posicion+elevacion) > 120
              then begin //b8
              Application.MessageBox(
              'Valor fuera de rango:  Ese punto no se puede centrar',
              'ERROR DE RANGO DE CENTRO',
              MB_OKCANCEL );
               EXIT;
               end;       //e8

if (posicion+elevacion) < 0
               then begin      //b9
              Application.MessageBox(
              'Valor fuera de rango:  Ese punto no se puede centrar',
              'ERROR DE RANGO DE CENTRO',
                MB_OKCANCEL );
               EXIT;
               end;               //e9

comando:='T1' ;
```

```
 if posicion > 0 then
              comando:=comando+'PR-'
              else
              comando :=comando+'PR+';
 if posicion < 0 then
          posicion:=-posicion*118000
          else
          posicion:=posicion*118000;
          posicion:=int(posicion);
          poss:=posicion;

contador:=8-length(poss);
          for aux_cont:=1 to contador do
          comando:=comando+'0';
          comando:=comando+ poss;
             listbox1.items[cont_memo2]:=comando;
             CONT_MEMO2:=CONT_MEMO2 + 1;

 contador:= Length(comando);
            contador:= Comm1.Write(comando[1], contador);

for ii:=10000000 downto 0 do
  auxiliar:=100;

comando:='M2';
contador:= Length(comando);
            contador:= Comm1.Write(comando[1], contador);
            listbox1.items[cont_memo2]:=comando;
             CONT_MEMO2:=CONT_MEMO2 + 1;
end;

initialization
  CONT_MEMO2:=0;
  pos0_cambiada:=false;
  vel0_cambiada:=false;
  pos1_cambiada:=false;
  vel1_cambiada:=false;
  a_ronza:=0;
  a_elevacion:=0;
  x_x:=50;
  y_y:=50;
  y_yy:=50;
  x_xx:=50;

  ronza:=180;
  elevacion:=60;
end.
```