ESCUELA POLITECNICA NACIONAL
FACULTAD DE INGENIERIA ELECTRICA

"ADQUISICION DE DATOS, SUPERVISION Y CONTROL PID UTILIZANDO CONTROLADORES LOGICOS PROGRAMABLES"

LUIS GIOVANNI PERRASO BASANTES

TESIS PREVIA A LA OBTENCION
DEL TITULO DE INGENIERO
EN ELECTRONICA Y CONTROL

QUITO, NOVIEMBRE 1995

Certifico que el presente trabajo ha sido elaborado en su totalidad por el señor Luis Giovanni Perraso Basantes.

ING. JORGE MOLINA M.

Director de Tesis

CONTENIDO

3. SISTEMA DE CONTROL PID

4. SOFTWARE DESARROLLADO PARA LA ADQUISICION DE DATOS, SUPERVISION Y CONTROL

5. RESULTADOS Y CONCLUSIONES

1. INTRODUCCION

1.1 INTRODUCCION

El desarrollo de los paquetes computacionales ha sido fundamental dentro del ambiente industrial. Hoy en día se dispone de software amigables e interactivos que permiten el monitoreo fácil y rápido de los parámetros más importantes de un determinado proceso al usuario. Sin embargo este tipo de software de adquisición de datos, en muchos casos no satisface las necesidades muy particulares de un proceso industrial determinado o en otros casos resulta ser demasiado general y muy costoso.

Se propone entonces, la creación e implementación de un programa computacional sencillo y práctico, que puede ser realizado por el propio usuario y que permitirá convertir una computadora personal en una estación de trabajo tipo SCADA básico para la adquisición de datos, supervisión y control PID de un proceso, con la ayuda de una tarjeta electrónica para tal propósito.

El trabajo de tesis que a continuación se presenta pretende implementar un sistema de adquisición de datos, supervisión y control, utilizando un controlador lógico programable como elemento fundamental en un Sistema de Control Digital Directo (DDC); específicamente como un dispositivo capaz de ejecutar un algoritmo de control Proporcional, Integral y Derivativo (PID) para cambiar las características dinámicas de un proceso cualquiera.

La tesis, en su Capítulo I, comienza con una introducción, una breve historia de la evolución tecnológica en la industria de los PLC's y el planteamiento de los objetivos y alcance que tendrá el presente trabajo.

En el Capítulo II se realiza una descripción del hardware a utilizar, con una breve revisión del tipo de controlador lógico programable, de los módulos que éste dispone y del tipo de interface implementado para comunicación.

El Capítulo III contiene el desarrollo teórico del algoritmo de control PID implementado en un PLC sin función PID así como también una descripción de la manera de ejecutar el control PID en un PLC que si dispone de la rutina para tal efecto.

En el Capítulo IV se desarrolla todo el sistema de adquisición de datos, supervisión y control PID basado en la interface Allen-Bradley 1784-KR. Se incluye las funciones destinadas a la comunicación con el PLC y las funciones de presentación gráfica al usuario de los datos supervisados.

La tesis finaliza en el Capítulo V con una aplicación práctica del sistema implementado. Se incluye varias pruebas realizadas con distintos tipos de plantas, sus resultados y conclusiones.

1.2 ANTECEDENTES

El desafío constante que toda industria tiene planteado para ser competitiva es el desarrollo, utilización y adaptación de nuevas tecnologías para conseguir una mayor productividad.

Muchos procesos de fabricación se realizan en ambientes nocivos para la salud, con gases tóxicos, ruidos, temperaturas muy altas o muy bajas; otros procesos en cambio, requieren que un determinado parámetro (temperatura, presión, etc.) sea mantenido constante para garantizar igual y una buena calidad en el producto terminado. Todo esto llevó a pensar en la posibilidad de desarrollar ciertas tareas repetitivas, peligrosas o de precisión a través de dispositivos inteligentes que remplazaran al hombre. Nace entonces el concepto de máquina y con ella la AUTOMATIZACION.

El constante desarrollo de la ciencia y tecnología ha hecho que la automatización sufra un proceso de transformación en su estructura, pasando en su inicio de enormes sistemas de control con gran consumo de energía y alta probabilidad de falla a sistemas mucho más pequeños, de menor disipación de energía y alta confiabilidad.

De estos equipos electrónicos, destinados al control de procesos industriales, el elemento más difundido actualmente es sin duda el controlador lógico programable o PLC (Programmable Logic Controller).

Los predecesores de los autómatas programables con memoria fueron por una parte los "Controladores programables por conexiones" cuyas funciones se realizaban cableando los módulos lógicos, y por otra parte, los ordenadores de proceso con sus controladores binarios, de los que se imitaron dichos sistemas de control para la simplificación del nivel operativo.

El controlador lógico programable o computador industrial es un equipo electrónico de operación digital que usa una memoria interna programable para el almacenamiento de funciones específicas destinadas a controlar mediante módulos de entrada y salida analógicos o digitales, varios tipos de procesos.

El PLC es muy utilizado debido a que permite en forma rápida y sencilla cambiar el comportamiento dinámico de un proceso. Es suficientemente confiable en su operación, permite encontrar en forma rápida las fallas en el sistema de control del cual forme parte, es altamente versátil, físicamente pequeño, tiene facilidad para la comunicación con una computadora y su introducción dentro de un sistema de control es, en la mayoría de los casos, económicamente justificable.

En la actualidad, con el aumento del tamaño de los procesos y su complejidad, es necesario obtener un control más sofisticado para conseguir un mejor funcionamiento de la planta y lograr los beneficios adicionales que ello implica.

Es vital entonces, que un sistema con PLC disponga de hardware compatible para aplicaciones industriales, además de software que permita, de manera rápida y precisa la visualización de los datos adquiridos; y de modo amigable, la comunicación hombre-máquina.

Actualmente, en un nivel jerárquico superior se utiliza el control supervisor tipo SCADA (Supervisory Control And Data Acquisition) basado en PLC's para el monitoreo de una planta.

1.3 OBJETIVOS Y ALCANCE

El trabajo de tesis que a continuación se desarrolla estudiará e implementará un sistema de control digital directo para controlar un proceso cualquiera utilizando como elemento fundamental un controlador lógico programable.

El PLC como elemento principal de control en este sistema, ejecutará un algoritmo digital de control para cambiar las características dinámicas del proceso estudiado. Dicho algoritmo de control constituye un controlador Proporcional Integral y Derivativo (PID), implementado en base a sumas y diferencias en un PLC que no dispone de rutina para tal efecto. También se estudiará la manera de programar la instrucción PID, en un PLC con rutina PID y se observará su comportamiento con distintos tipos de plantas.

El funcionamiento del controlador lógico programable será supervisado mediante el desarrollo e implantación de un sistema de adquisición de datos, que permitirá controlar y monitorear las variables más importantes del proceso estudiado.

Independientemente del tipo trabajo ejecutado por el PLC, la función del sistema de adquisición de datos, desarrollado en lenguaje "C", será la de monitorear e incluso modificar cualquier dato procesado en el PLC.

La Fig. 1-1 muestra a manera de bloques, una generalización del sistema que se pretende desarrollar.

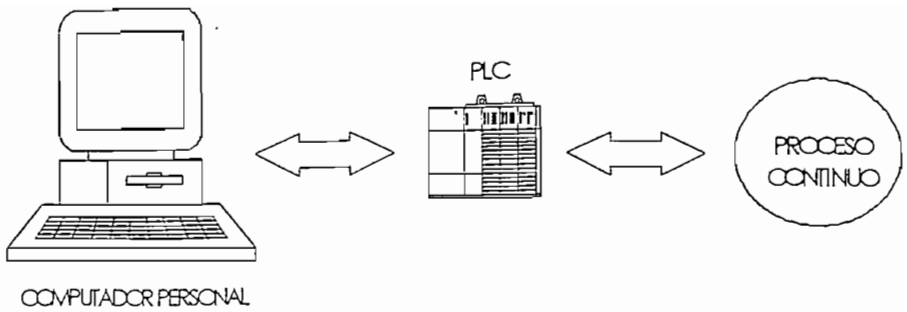


Fig. 1-1: Esquema del sistema a implementar.

El controlador lógico programable controla directamente un proceso continuo determinado, utilizando un algoritmo de control digital programado en su unidad central de procesamiento. Paralelamente a esto, el computador personal se encarga de adquirir datos en tiempo real y supervisar el comportamiento del PLC y del proceso de una manera gráfica. Eventualmente, y si las necesidades del proceso así lo requieran, el computador personal está en capacidad de transferir datos hacia el PLC con el propósito de cambiar ciertas características dinámicas del sistema.

La capacidad de los controladores lógicos programables de interconectarse en red mediante una interface RS-485 (Allen-Bradley DH-485) y el desarrollo de un software adecuado, permitirá conseguir un sistema de control tipo SCADA básico, que posibilite la adquisición de datos, supervisión y el control de todos y cada uno de los PLC's conectados en la red.

2. DESCRIPCION DEL SISTEMA A IMPLEMENTARSE

2. DESCRIPCION DEL SISTEMA A IMPLEMENTARSE

2.1 EL CONTROLADOR LOGICO PROGRAMABLE.

2.1.1 INTRODUCCION

La automatización tal y como hoy la concebimos, surgió de la utilización de relés, temporizadores, contadores, pulsadores, válvulas, levas y otros elementos basados en principios electromecánicos. Con ello se podía implementar cualquier aplicación particular con un diseño adecuado. Al aumentar la complejidad de las tareas a realizar, los paneles de maniobra eran cada vez más grandes, la probabilidad de avería aumentaba y el mantenimiento se hacía más difícil. La aparición de los semiconductores primero, y los circuitos integrados electrónicos posteriormente, generó un relevo generacional que trajo consigo la sustitución de los relés por dispositivos electrónicos de funciones específicas.

Los sistemas de automatización que utilizan tarjetas electrónicas basadas en compuertas lógicos físicas se caracterizaban por su mayor velocidad de respuesta, menor consumo de potencia y menor espacio, entre otras cualidades, pero aún conservan el problema de falta de flexibilidad de los sistemas construidos con relés debido a que su lógica es alambrada físicamente.

De otro lado, desde 1960 varios computadores fueron instalados en plantas de producción de vidrio para realizar tareas de control de hornos. Su uso evidenció las ventajas de los sistemas de "mando por programa" para aplicaciones industriales, como son la flexibilidad, facilidad de diseño y mantenimiento; lo que marcó el inicio de la era de la automatización usando lógica "alambrada" por programa (software).

La aparición de los microprocesadores da inicio a la era de los microcomputadores y minicomputadores, y se genera una gran migración de estos equipos hacia la industria

para realizar tareas de automatización y control, permitiendo un mayor grado de desarrollo de soluciones a problemas complejos, reducción de costos, ahorros en la instalación y aumento de la productividad.

Sin embargo, el uso de estos equipos evidenció que su diseño era inapropiado para operar en ambientes industriales. Los problemas encontrados más importantes fueron: sistemas operativos no orientados hacia tareas de mando y control, diseño de tarjetas y circuitos electrónicos con bajo nivel de rechazo de interferencias electromagnéticas, que los hace muy sensibles a tales perturbaciones y a fallas frecuentes, circuitos inadecuados para el manejo de señales de entrada/salida y desarrollo incipiente de programas y dispositivos de apoyo orientados a la automatización y control.

La falta de estandarización de los lenguajes de programación y de los sistemas de comunicaciones entre microcomputadores, la necesidad de usar lenguaje ensamblador para trabajar en tiempo real en procesos de alta velocidad de respuesta, los requerimientos de atmósfera controlada en los salones de microcomputadores y el desarrollo de programas de automatización implementados sobre sistemas operativos no orientados para este propósito, generaron muchos inconvenientes, que desalentaron su uso en aplicaciones industriales. Las ventajas de los mandos "programados" eran relativas como para generalizar su utilización. A finales de la década de los setenta, salen al mercado los primeros "Controladores Programables" los cuales reemplazaban a paneles de relés y estaban basados en microprocesadores. Su diseño electrónico y mecánico era muy robusto, de fácil instalación y adecuado para operar en la industria en un ambiente electromagnéticamente hostil. Estos nuevos equipos inicialmente utilizaron los "esquemas de contactos" (Ladder Logic) para su programación, por su facilidad de manejo por parte de los encargados del mantenimiento y diseño en las fábricas.

Los diseños posteriores de los ya denominados "Controladores Lógicos Programables" (Programmable Logic Controller) también podían ser programados con

"lógica de contactos" e incluían nuevos lenguajes de programación como la "lista de instrucciones" y el "esquema de funciones".

Inicialmente, los PLC contenían un pequeño set o conjunto de instrucciones. Posteriormente, el avance industrial y la gran demanda que tuvieron estos dispositivos obligaron también su desarrollo tecnológico: mayor capacidad de memoria, mayor número de entradas y salidas, variedad de instrucciones y sobre todo, mayor velocidad de procesamiento.

Paralelamente, los sistemas operativos, las tarjetas y circuitos electrónicos también evolucionaron para mejorar la seguridad de operación, el alcance y la programación de los PLC's. A nivel de los circuitos electrónicos, lo más relevante es el desplazamiento de los microprocesadores y en su lugar, el uso de microcontroladores.

Una de las cualidades más importantes de los microcontroladores es la reducción de la sensibilidad a interferencias electromagnéticas con respecto a los microprocesadores, lo que posibilita su uso en ambientes industriales con menor probabilidad de ser perturbados.

En los últimos años, el desarrollo de los autómatas programables ha estado ligado al desarrollo de los microcontroladores y las nuevas versiones de PLC están implementadas sobre microcontroladores cada vez más potentes.

En la década de los ochenta aparecen en el mercado autómatas de diferentes características y precios adaptables a las distintas gamas de prestaciones. A nivel industrial, su uso se vuelve más confiable y común por su diseño especial para operar en estos ambientes.

En la actualidad, los PLC's son más flexibles y cómodos de manejar que sus predecesores y se tiende a una estandarización industrial de estos dispositivos.

En 1978, la Asociación de Fabricantes de Equipos Eléctricos de los Estado Unidos (NEMA), formuló una estandarización para los controladores programables. NEMA STANDARD ICS3-1978 part ICS3-304, define un controlador programable como:

> "Un aparato electrónico de operación digital que usa una memoria programable para el almacenamiento interno de instrucciones para funciones específicas tales como: secuencias lógicas, temporización, conteo, aritméticas; para controlar, mediante módulos de entrada/salida analógicos o digitales, varios tipos de máquinas o procesos".

Hoy en día, debido a su tamaño reducido, la facilidad de programación, la cual puede ser realizada por personal eléctrico o electrónico sin conocimientos de informática, la posibilidad de almacenar los programas para su posterior y rápida utilización, la facilidad para modificar los programas, la sencillez de montaje y la modularidad de su diseño, que permite crecer en la medida que el sistema lo exija, ha convertido al PLC en un elemento esencial en los sistemas de automatización y control, por lo que es muy común encontrarlo realizando diversas tareas en la industria moderna. Desde el punto de vista técnico puede afirmarse que el PLC simplificó la industria y la dotó de modernas herramientas para su desarrollo.

2.1.2 ESTRUCTURA DE UN PLC

En la Fig. 2-1 se muestran los componentes básicos de un controlador lógico programable.

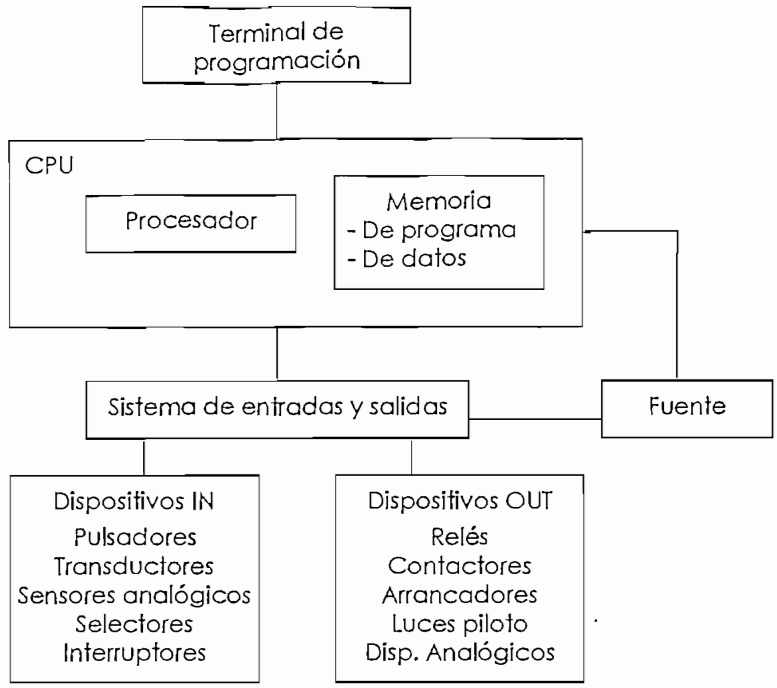


Fig. 2-1: Estructura básica de un PLC.

UNIDAD CENTRAL DE PROCESAMIENTO (CPU)

La unidad central de procesamiento incluye al procesador o microcontrolador y a la memoria. Constituye la parte principal del PLC. Generalmente un PLC dispone de solo un procesador.

Procesador:

Su principal función es el control y gobierno de las actividades del PLC. Evalúa cíclicamente el programa en forma secuencial. El tiempo en el que el procesador completa un ciclo de operación se denomina "Scan time". Típicamente la exploración del proceso puede dividirse en dos partes:

- Exploración de entradas y salidas.

- Exploración del programa en memoria.

La exploración de entradas y salidas implica la lectura de todas las entradas y la actualización de todas las salidas. La exploración del programa en memoria implica la ejecución, paso a paso, de todas las instrucciones dadas en el programa del usuario y en el orden que estas han sido ingresadas.

Memoria:

El sistema de memoria de un controlador lógico programable es básicamente un arreglo de bits accesibles aleatoriamente, cada uno de los cuales es identificado por una única dirección. El módulo de memoria contiene el programa del usuario y la tabla de datos de cada una de las instrucciones ingresadas en dicho programa.

El tamaño de la memoria es usualmente especificado en bytes o words, y puede variar en tamaño desde 256 bytes hasta 128 kwords.
Los tipos de memoria actualmente encontrados en los PLC's son:

- Read only memory: ROM, PROM, EPROM, EEPROM.
- Read-write memory: CMOS, RAM, CORE.

TERMINAL DE PROGRAMACION

Es un dispositivo conectado PLC con el propósito de introducir el programa del usuario o editarlo. Incluso, puede quedar permanentemente conectado para propósitos de monitoreo del proceso. Entre otros tenemos:

- Terminal de programación manual con display de cuarzo líquido (Hand Held Terminal).
- Terminal de programación de tubos de rayos catódicos propio de cada fabricante.

- Computador personal con el respectivo software e interface para la comunicación con el PLC.

SISTEMA DE ENTRADAS Y SALIDAS (E/S)

Es la parte que adapta las señales lógicas provenientes de la CPU en señales compatibles con el proceso y viceversa. Los módulos de entrada/salida pueden ser:

- Discretos o digitales: Son usados con elementos de control ON-OFF tales como pulsantes, finales de carrera, interruptores de presión, etc. Para el caso de entrada, o contactores, relés, electroválvulas, etc. Para el caso de las salidas. Los módulos discretos de salida por lo general están constituidos por triacs, transistores o por relés. Se dispone, típicamente, de módulos de 4, 8 o 16 circuitos (puntos).
- Analógicos: Convierten las señales de corriente o voltaje provenientes de procesos continuos, en un valor numérico reconocible por la CPU. Los módulos de entrada analógicos poseen una resolución de 8 hasta 16 bits.
- Especiales: Transductores de presión, termopares, visualizadores, módulos de comunicación, etc.

2.1.3 PRINCIPIO DE FUNCIONAMIENTO DE UN PLC.

Un controlador lógico programable realiza continuamente un ciclo de barrido o exploración (scan) que consiste en:

- Lectura de las entradas.
- Ejecución del programa: calcula las nuevas salidas en función de las entradas, la secuencia y lógica de las instrucciones.
- Actualización de las salidas.

En el siguiente esquema (Fig. 2-2) se muestra un ciclo básico de barrido de un PLC. El tiempo que demora en este ciclo es conocido como SCAN TIME y depende del modelo de PLC disponible (aproximadamente alrededor de los 20 ms).

El ciclo de operación consiste de dos partes: el scan del programa y el scan de las entradas y salidas. En el I/O scan, los datos asociados con salidas externas son transferidos del archivo de datos de salida a los terminales de salida (estos datos fueron actualizados en el scan anterior del programa). Adicionalmente los terminales de entrada son examinados y transferidos sus datos a los archivos correspondientes. El I/O scan y el scan del programa son independientes. Cualquier cambio de estado que ocurra en los terminales externos durante el scan del programa no son considerados sino hasta el próximo I/O scan.

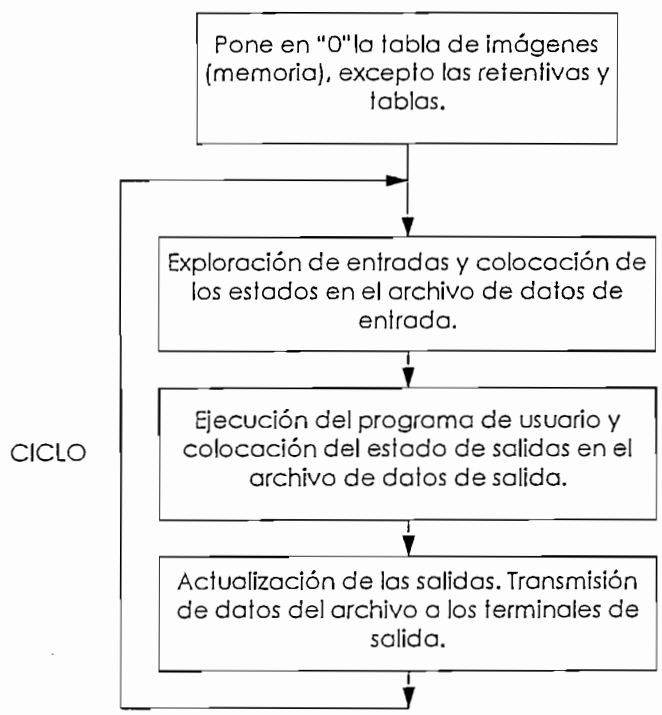


Fig. 2-2: Ciclo básico de barrido (Scan Time) de un PLC.

2.1.4 CONTROLADOR MODULAR SLC 500

El controlador modular SLC 500 ofrece flexibilidad en la configuración de sistemas, mayor poder de procesamiento y amplia capacidad de módulos de entrada y salida de datos. Seleccionando adecuadamente los módulos constitutivos de este tipo de PLC se puede diseñar un controlador programable específico para una determinada aplicación. Dispone de:

- Cuatro unidades de procesamiento central: posee gran funcionalidad; soporta gran variedad de módulos de entrada y salida.
- Cuatro diferentes tipos de chasis: proporciona flexibilidad en las opciones de expansión.
- Fuentes de alimentación: soporta alimentación AC y DC.
- Opciones de comunicación: soporta comunicación DH-485, RS-232.

La Tabla 2-1 muestra las características principales de los controladores lógicos programables modulares Allen-Bradley SLC 5/01, 5/02, 5/03 y 5/04.

Tabla 2-1: Características de los procesadores SLC-500.

ESPECIFICACION	SLC 5/01	SLC 5/02	SLC 5/03	SLC 5/04
Memoria de programa	1K de instrucciones o 4K de palabras de datos	4K de instrucciones o 16K de palabras de datos	12K de instrucciones y 4K de palabras de datos	20K de instrucciones y 4K de palabras de datos
Capacidad de I/O	256 discretas	480 discretas	960 discretas	960 discretas
Scan time típico	8 ms/K	4.8 ms/k	1 ms/k	0.9 ms/k
Ejecución de XIC	4 µs	2.4 µs	0.44 µs	0.37 µs
Programación	APS o HHT	APS o HHT	APS	APS
Comunicación	Receptor DH-485	Receptor o iniciador DH-485	Receptor o iniciador DH-485 y RS-232	DH+ o RS-232
Consumo a 5 Vdc	350 mA	350 mA	500 mA	650 mA
Instrucción PID	No disponible	Disponible	Disponible	Disponible
Interrupción de tiempo STI	No disponible	Seleccionable	Seleccionable	Seleccionable

2.2 *MODULO DE ENTRADAS Y SALIDAS ANALOGICAS ALLEN-BRADLEY 1746-NIO4I.*

2.2.1 CARACTERISTICAS.

El módulo analógico Allen-Bradley 1746-NIO4I provee 2 canales de entrada analógicos y 2 canales de salida analógicos en el mismo módulo. Cada canal de entrada puede ser seleccionado por el usuario para voltaje o corriente mientras que el canal de salida es únicamente por corriente.

Para obtener la máxima capacidad del módulo analógico es imperativo una instalación adecuada del mismo. Las siguientes secciones describen con más detalle el módulo analógico y la forma adecuada de instalarlo en un sistema Allen-Bradley SLC 500.

Las especificaciones técnicas del módulo analógico constan en el apéndice D.

2.2.2 CONFIGURACION DEL MODULO

El módulo analógico Allen-Bradley 1746-NIO4I posee pequeños interruptores seleccionables por el usuario para permitir configurar los canales de entrada por corriente o por voltaje. Estos interruptores están colocados sobre la propia tarjeta del módulo y etiquetados "1" y "2" para controlar el modo de funcionamiento de los canales de entrada 0 y 1 respectivamente. El interruptor en la posición "ON" configura el canal para una entrada por corriente mientras que en la posición "OFF" lo configura para una entrada por voltaje.

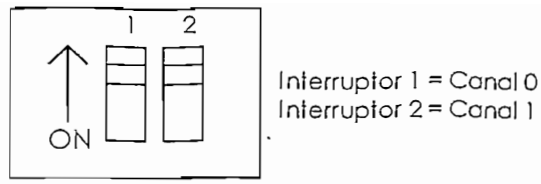


Fig. 2-3: Interruptores para configuración del módulo analógico A-B 1746-NIO4I.

2.2.3 SELECCION DEL SLOT

Dos factores determinan la ranura dentro del bastidor donde debe ubicarse el módulo analógico: la temperatura ambiente y el ruido eléctrico. Las siguientes recomendaciones pueden ser tomadas en cuenta para la ubicación del módulo:

- Colocar el módulo lejos de voltajes AC o voltajes DC elevados, y
- Lejos de la ranura que contiene la fuente de poder, si está utilizando un sistema modular.

2.2.4 CONSIDERACIONES DEL CABLEADO

Se tomarán en cuenta las siguientes características:

- Todos los terminales analógicos comunes (ANL COM) están eléctricamente unidos dentro del módulo pero no están conectados al chasis del módulo.

- Por seguridad, los voltajes en los terminales IN+ e IN- deben permanecer dentro de ±20V con respecto al terminal ANL COM para garantizar un adecuado funcionamiento del canal de entrada. Esta consideración es válida para la operación de los canales de entrada por voltaje y por corriente.

- La resistencia de carga para los canales de salida por corriente debe permanecer entre 0 y 500Ω.

2.2.5 DIAGRAMA DE CONEXIONES

En la Fig. 2-4 se muestra la manera de realizar las conexiones tanto para los terminales de entrada como para los de salida del módulo analógico Allen-Bradley 1746-NIO4I. Nótese que los terminales correspondientes a las entradas no usadas (terminales 3, 4 y 5) deben ir puenteados, no así los terminales de las salidas no utilizadas (terminales 10 y 11).

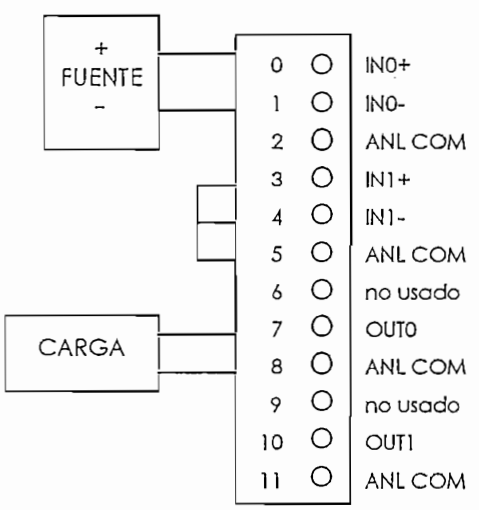


Fig. 2-4: Diagrama de conexiones del módulo analógico A-B 1746-NIO4I.

2.2.6 MINIMIZACION DEL RUIDO ELECTRICO

Las entradas del módulo analógico emplean filtros digitales de alta frecuencia que reducen significativamente los efectos del ruido eléctrico en las señales de entrada. Sin embargo debido a la variedad de aplicaciones y ambientes en los cuales el módulo analógico puede ser instalado y operado, es imposible asegurar que todo el ruido ambiental sea removido por los filtros de entrada.

En este sentido, para reducir los efectos del ruido ambiental en las señales de entrada, se recomienda:

- Instalar el sistema SLC-500 en un gabinete tipo NEMA. Se debe comprobar que el sistema tenga una adecuada instalación a tierra.
- Se recomienda usar cable BELDEN #8761 para la instalación.
- La instalación debe ser independiente de cualquier otra instalación.
- Se puede lograr una inmunidad adicional al ruido realizando la instalación con tubería metálica.

2.2.7 DIRECCIONAMIENTO DEL MODULO ANALOGICO

Cada canal de entrada del módulo analógico 1746-NIO4I es direccionado como una palabra simple en la tabla de imágenes de entrada y cada canal de salida, de la misma manera, es direccionado como una palabra simple en la tabla de imágenes de salida. El módulo analógico usa en total 2 palabras de entrada y 2 palabras de salida.

Las señales de entrada convertidas desde el canal 0 y 1 son direccionadas como palabras 0 y 1 del slot donde el módulo reside. De igual forma, las señales de salida de los canales 0 y 1 son direccionadas como palabras de salida 0 y 1 del slot donde residen.

En la Fig. 2-5 se ilustra el direccionamiento del módulo analógico.

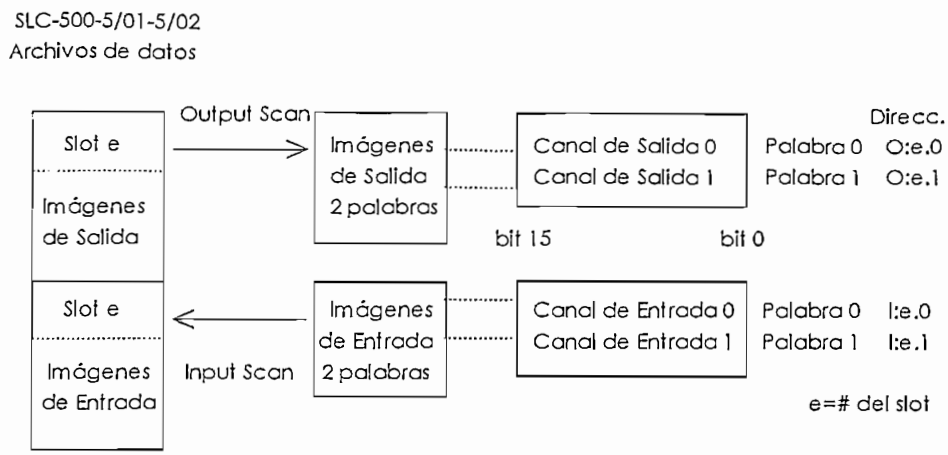


Fig. 2-5: Direccionamiento del módulo analógico.

2.2.8 DIRECCIONAMIENTO A NIVEL DE BITS

Los siguientes mapas de bits, Fig. 2-7 y Fig. 2-6, muestran el direccionamiento a nivel de bits para las entradas y salidas analógicas. La resolución del conversor de entrada es 16 bits o 1 palabra, mientras que la resolución del conversor de salida es 14 bits.

Los 2 bits menos significativos (O:e.0/0 y O:e.0/1) de la palabra de salida no tienen efecto en el valor de la salida.

Fig. 2-6: Direccionamiento a nivel de bits de los canales de salida analógicos.

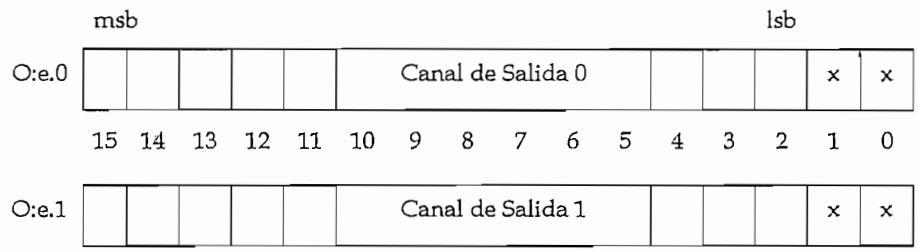
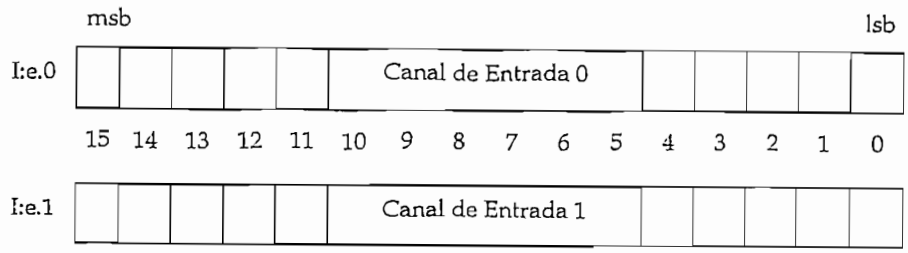


Fig. 2-7: Direccionamiento a nivel de bits de los canales de entrada analógicos.



2.2.9 ACTUALIZACION DE DATOS

Los datos de las entradas y salidas analógicas son actualizados por el procesador solamente una vez durante cada exploración del programa del usuario.

Si una aplicación requiere que el procesador actualice los datos analógicos más frecuentemente que una vez por "scan", se debe usar la instrucción Immediate Input (IIM) o Immediate Output (IOM), las cuales actualizan típicamente 16 bits o un canal analógico en 1 milisegundo.

2.2.10 CONVERSION DE LOS DATOS ANALOGICOS DE ENTRADA.

Las entradas analógicas convierten las señales de voltaje y corriente en valores binarios de 16 bits, usando el complemento de 2 para los valores negativos.

La Tabla 2-2 muestra algunos rangos típicos de corriente y voltaje de entrada utilizados, su representación decimal equivalente y su resolución.

Tabla 2-2: Rangos típicos de entrada y su representación decimal equivalente.

Rango Voltaje/Corriente	Representación Decimal	Número de bits significativos	Resolución por LSB
-10 a +10V -1LSB	-32768 a +32767	16 bits	305.176μV
0 a 10V -1LSB	0 a 32767	15 bits	305.176μV
0 a 5V	0 a 16384	14 bits	305.176μV
1 a 5V	3277 a 16384	13.67 bits	305.176μV
-20mA a +20mA	-16384 a +16384	15 bits	1.2207μA
0 a +20mA	0 a 16384	14 bits	1.2207μA
4 a +20mA	3277 a 16384	13.67 bits	1.2207μA

Para determinar el voltaje o corriente aproximados que un valor numérico de entrada representa se debe usar las siguientes relaciones:

DETERMINACION DEL VOLTAJE DE ENTRADA

$$\frac{10V}{32768} \cdot \text{Valor numérico de entrada} = \text{Voltaje de entrada(V)}$$

Ec. 2.2-1

DETERMINACION DE LA CORRIENTE DE ENTRADA

$$\frac{20mA}{16384} \cdot \text{Valor numérico de entrada} = \text{Corriente de entrada (mA)}$$

Ec. 2.2-2

Cuando la entrada I1:1.0/0 se habilite, la rung es verdadera y se realiza la operación de movimiento del valor 32767 a la tabla de imágenes de salida correspondiente. Al finalizar el barrido de todo el programa, este valor es transferido al módulo analógico donde es convertido al valor de voltaje o corriente apropiado dependiendo del tipo de módulo usado.

Si en el próximo barrido del programa la rung es falsa, la instrucción de movimiento no se realiza. A menos que otra rung transfiera datos a la tabla de imágenes de salida, el valor previo es retenido, es decir el valor 32767 es retenido en la tabla de imágenes de salida y posteriormente transferido al módulo analógico al finalizar el scan del programa.

EJEMPLO DE PROGRAMACION NO RETENTIVA.

En la Fig. 2-9 se muestra un ejemplo de programación no retentiva del módulo analógico.
Cuando la entrada discreta esté activa, el valor 32767 es transferido al módulo analógico. Si la entrada se desactiva, el valor transferido es 0.

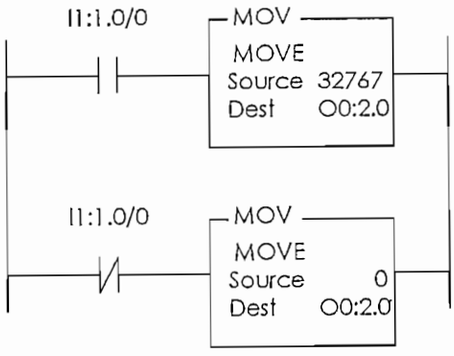


Fig. 2-9: Ejemplo de programación no retentiva.

Los canales de entrada del módulo analógico incorporan circuitos de acondicionamiento de señal con el propósito de rechazar el ruido eléctrico que puede introducirse en las señales de entrada analógicas. El acondicionamiento es realizado filtrando la señal a través de un filtro digital.

2.2.16 FILTROS DE LOS CANALES DE ENTRADA

Los canales de entrada del módulo analógico incorporan circuitos de acondicionamiento de señal con el propósito de rechazar el ruido eléctrico que puede introducirse en las señales de entrada analógicas. El acondicionamiento es realizado filtrando la señal a través de un filtro digital.

La respuesta de frecuencia del módulo puede observase en la Fig. 2-10. Señales con frecuencia superior a 10 Hz sufren una atenuación superior a 3 dB. Dentro de esta característica de respuesta de frecuencia del módulo analógico se encuentran normalmente sensores tales como de temperatura, presión, flujo y otro tipo de sensores cuyas señales no tengan variaciones rápidas. Del diagrama de frecuencia se puede observar que señales de 60 Hz en la entrada positiva, con respecto a la entrada negativa, son atenuadas en aproximadamente 55 dB.

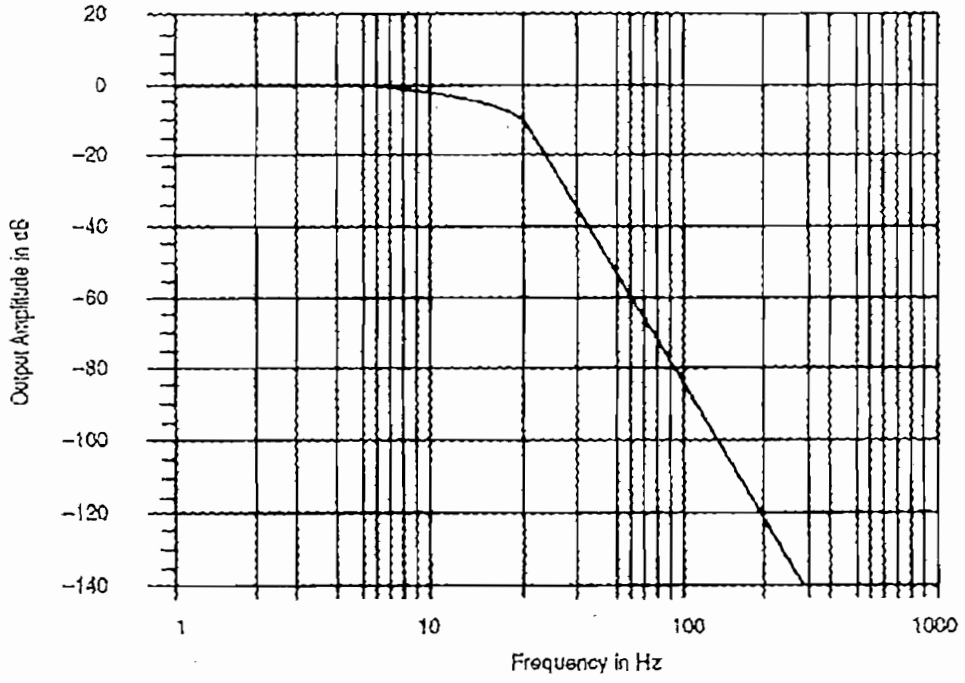


Fig. 2-10: Respuesta de frecuencia del módulo analógico.

El efecto del filtro respecto al tiempo puede observarse analizando la respuesta paso del módulo mostrada en la Fig. 2-11. Esta respuesta no tiene sobreimpulso y tiene un tiempo de estabilización rápido. Independientemente de la magnitud de la entrada, el tiempo de estabilización (al 95%) es de 60 ms.

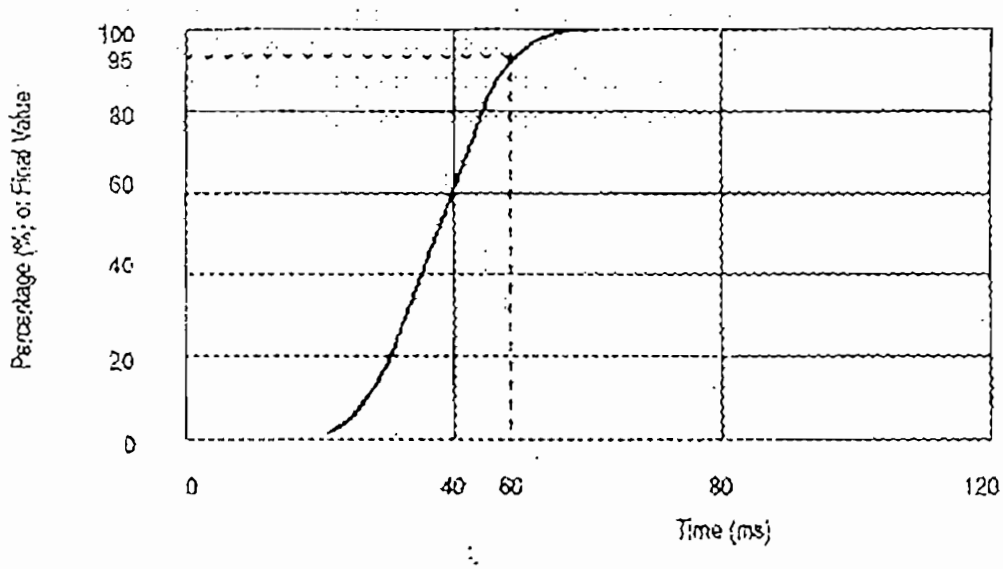


Fig. 2-11: Respuesta paso del módulo analógico.

Por ejemplo, si una entrada instantáneamente cambia de 0 a 10 V, el valor convertido por el módulo analógico después de 60 ms es 9.5 V. Dentro de este tiempo, el módulo actualiza los valores de entrada en memoria aproximadamente cada 512 µs.

2.3 DESCRIPCION DE LA RED DH-485.

2.3.1 QUE HACE LA RED DH-485

La red DH-485 fue diseñada para usuarios con necesidades menos sofisticadas, aplicaciones sin tiempos críticos donde el costo de PLC y redes grandes es prohibitivo y donde las aplicaciones no requieren de su capacidad.

La red DH-485 es una red que permite la transferencia de información entre varios dispositivos en una planta o proceso. La red monitorea los parámetros del proceso, estado del proceso, los parámetros de un dispositivo, estado del dispositivo y soporta programas de aplicación para la adquisición y el monitoreo de datos, la programación, supervisión y control.

La red DH-485 se caracteriza por ofrecer:

- Interconexión de hasta 32 dispositivos.
- Capacidad multi-master, es decir, existe la posibilidad de disponer de más de un dispositivo iniciador de la comunicación (dispositivo maestro).
- Control de acceso token-passing.
- Habilidad de añadir o remover nodos en la red sin romperla.
- Longitud máxima 4000 pies (1219 m).

2.3.2 PROTOCOLO DE LA RED DH-485.

El protocolo de comunicación utilizado para el control de los mensajes transmitidos sobre la red DH-485, soporta dos clases de dispositivos: iniciadores de la comunicación y dispositivos de respuesta. Todos los dispositivos que inician la comunicación poseen un "espacio" para realizarla. Para determinar cual dispositivo tiene el derecho a iniciar la comunicación, se utiliza un algoritmo token passing.

2.3.3 PROCESO DE COMUNICACION EN LA RED

Un nodo que tenga habilitada su comunicación puede enviar cualquier paquete válido de información a la red DH-485. Cada vez que un nodo recibe la habilitación para comunicarse (token), se le permite solamente una transmisión más dos intentos. Luego de que un nodo envía un paquete de información, éste intenta dar la señal de

2.3.5.1 *CONEXION DE LA INTERFACE 1784-KR A UNA RED DH-485*

La Fig. 2-12 muestra en que consiste esta clase de conexión utilizando 3 controladores lógicos programables y una estación de programación (computador personal). Esta configuración requiere la interface 1784-KR y 3 acopladores de enlace (1747-AIC):

- Un controlador SLC-500 es conectado a cada uno de los acopladores de enlace con un cable 1747-C11.
- La interface 1784-KR es conectada a la red en cualquier acoplador de enlace, como se muestra en la figura.

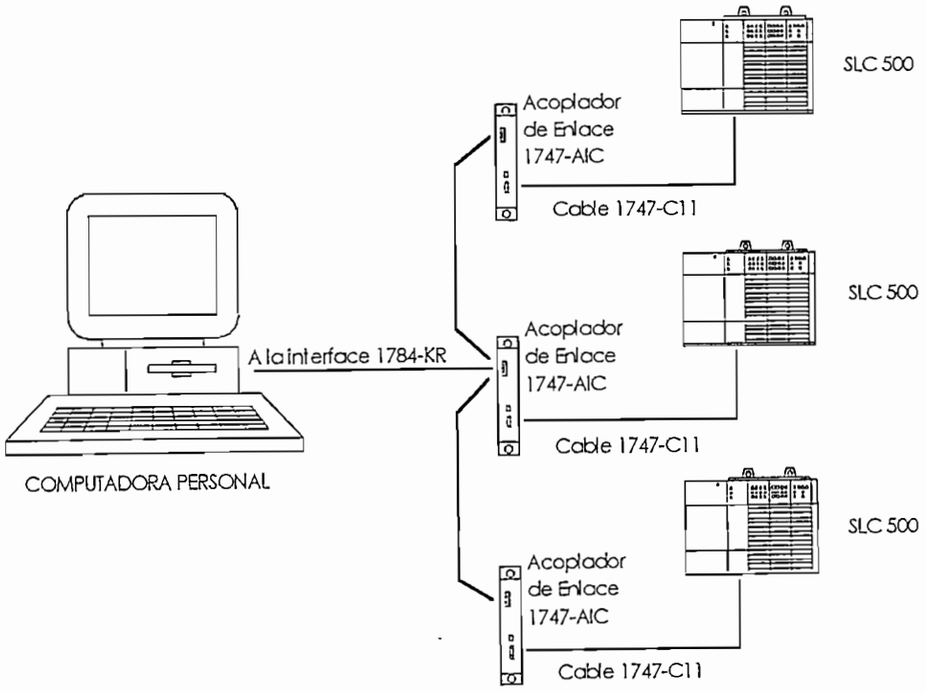


Fig. 2-12: Comunicación a varios SLC 500 a través de la red DH-485.

	G-AU).
Salidas	RS-485 / protocolo DH-485.
Interrupciones de hardware	IRQ2, IRQ3, IRQ4, IRQ5.
Baud rate	300, 1200, 2400, 4800, 9600, 19200 baud.

2.3.5.2 CONEXION PUNTO-PUNTO DE LA INTERFACE 1784-KR EN LA RED DH-485.

La Fig. 2-13 muestra en ejemplo de enlace punto-punto. Este consiste de un controlador programable SLC-500 y la estación de programación. Esta configuración requiere la interface 1784-KR y un acoplador de enlace. El SLC-500 es conectado al acoplador de enlace con cable 1747-C11. La interface 1784-KR es conectada directamente al acoplador de enlace.

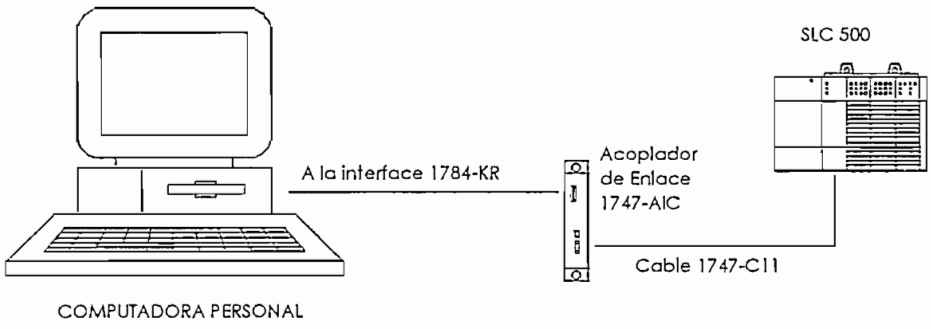


Fig. 2-13: Comunicación a un SLC 500 usando el enlace punto a punto DH-485.

2.4 INTERFACE ALLEN-BRADLEY 1784-KR.

2.4.1 FUNCION DE LA INTERFACE A-B 1784-KR

Con el uso de una computadora IBM PC XT/AT o compatible se puede construir una estación de adquisición de datos dentro de la red Allen-Bradley DH-485. Cuando se combina con la interface 1784-KR y el software 6001-F2E, una computadora personal puede conseguir información vital de un proceso.

Además, una computadora puede convertirse en un terminal de programación de controladores lógicos programables, cuando es usada la interface 1784-KR y el software 1747-PA2E (APS) evitando así el uso de un 1747-PIC.

2.4.2 ESPECIFICACIONES Y CARACTERISTICAS DE LA INTERFACE 1784-KR.

Conector de salida	Terminal de 6 posiciones (Phoenix MSTBA-1.5/6-G-AU).
Salidas	RS-485 / protocolo DH-485.
Interrupciones de hardware	IRQ2, IRQ3, IRQ4, IRQ5.
Baud rate	300, 1200, 2400, 4800, 9600, 19200 baud.
Longitud máx. de línea.	4000 pies (1219m).
Alimentación y potencia	+5Vdc, 1.8A.
Voltaje de aislamiento	500 Vdc.
Temperatura de operación	0 a 60° C.
Humedad	5 a 95% no condensada.

La interface 1784-KR permite a computadoras IBM XT/AT o compatibles comunicarse sobre la red Allen-Bradley DH-485. Esto incluye comunicación a la familia de controladores programables Allen-Bradley SLC-500.

La interface 1784-KR es una tarjeta electrónica de simple slot que reside en cualquier slot de expansión disponible en una computadora.

A continuación se mencionan las características más importantes de la interface Allen-Bradley 1784-KR.

- Utiliza un half-slot del computador para realizar la interface a la red DH-485.
- Proporciona una interface directa al bus del PC.
- El coprocesador que esta interface dispone reduce el trabajo del PC liberándolo de las actividades de comunicación token-passing.
- Conexión a múltiples controladores programables SLC-500 mediante comunicación en la red DH-485.
- Conexión a un solo SLC-500 mediante enlace punto a punto.
- Programación de controladores lógicos programables SLC-500 para lo cual se requiere la utilización del software 1747-PA2E (APS).
- Adquisición de datos en la red DH-485, incluyendo controladores programables SLC-500, requiriendo para el efecto la disponibilidad del software 6001-F2E.
- Se puede seleccionar la dirección de memoria, el nivel de interrupción y la dirección de I/O lo que posibilita el uso de la interface con otras tarjetas instaladas en la computadora.

2.4.3 OPERACION DE LA INTERFACE 1784-KR.

La interface 1784-KR provee una comunicación directa entre computadores IBM XT/AT o compatibles y la red DH-485. Dicha interface es una tarjeta electrónica inteligente que soporta el protocolo DH-485 requerido para actuar como un nodo en la red DH-485, liberando a la computadora principal de esta tarea.

La interface 1784-KR es uno de los tres componentes necesarios para comunicarse en la red DH-485. Dependiendo de la aplicación se requiere:

- La interface Allen-Bradley 1784-KR.
- El cable apropiado de interconexión.
- Un correcto paquete de programación para la aplicación (1747-PA2E APS o 6001-F2E DH-485 Standard Driver).

Los drivers incluidos en el módulo proveen el adecuado nivel de señal para ser utilizado en la red DH-485.

Para proteger al computador, la interface 1784-KR posee un pórtico con un voltaje de aislamiento de 500Vdc entre el cable de comunicación y el computador.

Si se utilizan otras tarjetas con la interface 1784-KR pueden ocurrir conflictos con la dirección del bus o los niveles de interrupción, si éstos no son seleccionados adecuadamente.

2.4.4 INSTALACION DE LA INTERFACE 1784-KR.

Las siguientes instrucciones indican la manera de instalar la interface 1784-KR.

1. Apague la computadora y retire el cable de poder.
2. Remueva todos los cobertores necesarios para permitir el acceso a los slots de expansión.
3. Remueva los tornillos del slot de expansión.
4. Seleccione la dirección de memoria, la dirección de I/O y el nivel de interrupción con los mircro interruptores ensamblados en la tarjeta. La Fig. 2-14 muestra los interruptores utilizados para seleccionar cada una de estas opciones.

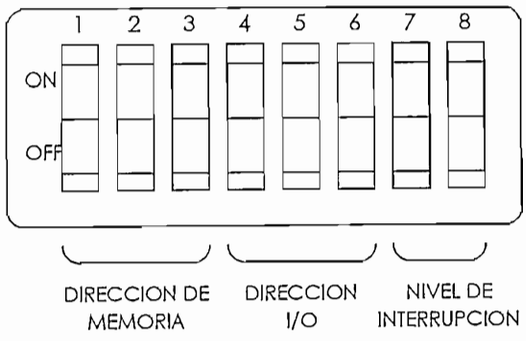


Fig. 2-14: Micro interruptores de selección.

Tabla 2-4: Interruptores para direccionamiento de memoria.

	POSICION DEL INTERRUPTOR		
DIREC. DE MEMORIA	1	2	3
C0000H-C7FFFH	OFF	OFF	OFF
C0000H-C7FFFH	OFF	OFF	ON
C8000H-CFFFFH	OFF	ON	OFF
C8000H-CFFFFH	OFF	ON	ON
D0000H-D7FFFH	ON	OFF	OFF
D8000H-DFFFFH	ON	OFF	ON
E0000H-E7FFFH	ON	ON	OFF

Los parámetros seleccionados por defecto son los siguientes:

- Dirección de memoria física: C0000H-C7FFFH
- Dirección I/O: 300H-307H
- Nivel de interrupción: IRQ 2/9

Para seleccionar parámetros distintos a los fijados por defecto, es necesario ingresar el comando que se indica a través del siguiente ejemplo:

Si se quiere seleccionar:

Dirección de memoria física: d0000H
Dirección I/O: 360H
Nivel de interrupción: IRQ 3

se debe ingresar el siguiente comando en DOS antes de correr cualquier software con la interface:

Set AB_OAKLAND = md000,b360,i3

donde:

- m debe preceder al segmento de memoria base en HEX.
- b debe preceder a la dirección de I/O en HEX
- i debe preceder al nivel de interrupción.

Los valores pueden ser ingresados en cualquier orden. Solamente los valores que no sean los seleccionados por defecto deben ser ingresados.

Si se está usando una computadora con monitor EGA se recomienda no usar la dirección de memoria por defecto (C0000H-C7FFFH). Seleccione la dirección de memoria en cualquier otro valor.

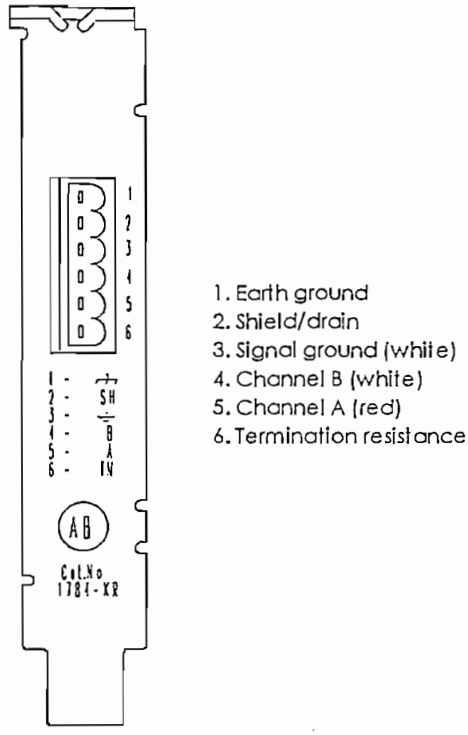


Fig. 2-15: Vista frontal de la interface Allen-Bradley 1784-KR.

2.5 ACOPLADOR AISLADO DE ENLACE 1747-AIC.

El acoplador aislado de enlace (Insolated Link Coupler) es un dispositivo necesario en la configuración que se pretende implementar. Este dispositivo permite acoplar los datos transmitidos entre el computador y el controlador lógico programable dentro de una red DH-485 de PLC's. Se requiere un acoplador para cada uno de los nodos que se pretenda instalar. El número máximo de dispositivos que pueden ser conectados en red es 31.

Para protección de los dispositivos conectados en la red, el acoplador posee un voltaje de aislamiento de 1500 Vdc.

Solamente uno de los acopladores aislados de enlace, al final de la red, debe tener los terminales 1 y 2 puenteados, esto conecta a tierra la malla del cable de comunicación.

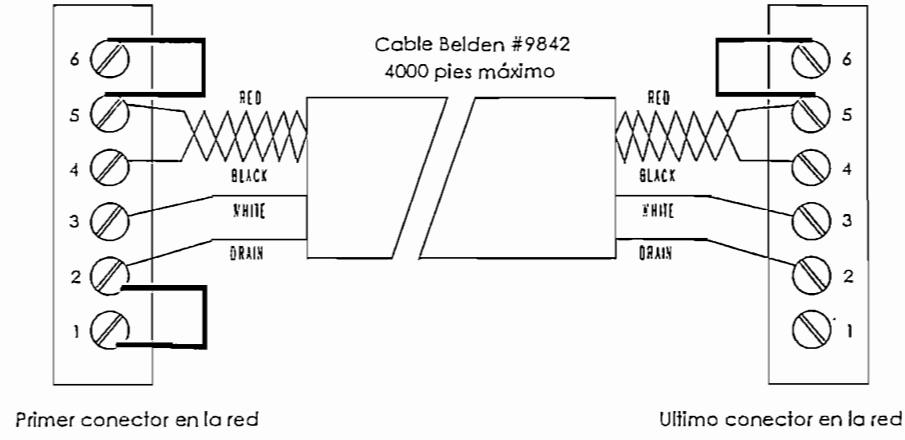


Fig. 2-16: Conexión de los acopladores aislados de enlace en una red de PLC's.

Los acopladores aislados de enlace, tanto al inicio como al final de la red, deben tener los terminales 5 y 6 puenteados para conectar una impedancia terminal, internamente construida, a la red.

Si la interface 1784-KR está al final de la red, se puede puentear los terminales 5 y 6 del bloque de terminales. Esto conecta la impedancia terminal construida en la tarjeta.

2.6 SOFTWARE DE COMUNICACION.

La interface Allen-Bradley 1784-KR permite la comunicación básicamente con dos tipos de software:

- El software Allen-Bradley 6001-F2E.
- El software 1747-PA2E SLC-500 (APS).

El software Allen-Bradley 6001-F2E usado en conjunto con la interface 1784-KR permiten al usuario escribir programas de aplicación propios en lenguaje "C", que establezcan directamente la comunicación con la red DH-485.

Con el software y el módulo, la computadora principal puede actuar como una estación de adquisición y supervisión de datos en la red DH-485.

El software 6001-F2E permite a los usuarios una interface directa al bus de la computadora; incluye para tal efecto un conjunto de funciones en lenguaje "C" utilizadas también para la comunicación en la red DH-485.

Un controlador lógico programable, la red DH-485 y la combinación de la interface 1784-KR y el software 6001-F2E constituyen un paquete para adquisición, supervisión, control y red, fáciles de implementar a bajo costo. Para aquellas aplicaciones propias que no requieran de la capacidad de una red mucho más sofisticada, este paquete constituye una buena solución de conexión en red.

El uso de la interface 1784-KR con el software 1747-PA2E SLC-500 (APS) permite a la computadora principal actuar como un terminal de programación de controladores lógicos programables SLC-500 evitando de esta manera el uso de la interface 1747-PIC.

El 1747-PIC es un dispositivo opcional usado para la comunicación entre un controlador SLC 500 y una computadora personal. Su función es convertir el protocolo RS-232 del computador a protocolo DH-485 del PLC.

Debido a que la interface 1784-KR es un módulo half-slot que puede residir en cualquier slot de expansión disponible en un IBM XT/AT o compatible, se puede implementar en un PC, las conexiones necesarias para usarlo en variadas aplicaciones ya sea como dispositivo de programación, adquisición y supervisión de datos o una sencilla interface con la red DH-485.

2.2.14 DETECCION DE VALORES FUERA DE RANGO

El módulo analógico no provee una señal o bandera para los valores fuera de rango. Sin embargo, si esta característica es crítica para aplicaciones específicas, es posible realizarla mediante programa utilizando las instrucciones de comparación incluidas en el controlador lógico programable.

2.2.15 HABILITACION Y DESHABILITACION DEL MODULO

ENTRADAS CON SLOT DESHABILITADO

El módulo continúa actualizando los valores de entrada al procesador. Sin embargo, el procesador no lee las entradas desde un módulo deshabilitado. Por lo tanto, cuando el procesador deshabilita el módulo analógico, las entradas del módulo aparecen en la tabla de imágenes del procesador retenidas en su último estado. Cuando el procesador las habilita nuevamente, las entradas son recibidas por el procesador en el siguiente barrido del programa.

SALIDAS CON SLOT DESHABILITADO

El procesador puede cambiar los datos de salida analógicos en la tabla de imágenes, sin embargo estos datos no son transferidos al módulo, manteniéndose las salidas en su último estado. Cuando se habilita nuevamente al slot, los datos son transferidos al módulo en el próximo barrido del programa.

Este software de comunicación es usado básicamente en sistemas de adquisición de datos y programas de aplicación desarrollados por el propio usuario. Habitualmente es utilizado en aplicaciones no tan sofisticadas dentro de las industrias:

Automotriz	Metales	Máquinas-herramientas
Farmacéutica	Electrónica	Control
Comidas y bebidas	Petroquímica	Transportación
Plástico	Pulpa y papel	Manejo de energía.
Etc.		

2.6.1 CONFIGURACION

Una configuración típica usando la interface 1784-KR es la siguiente:

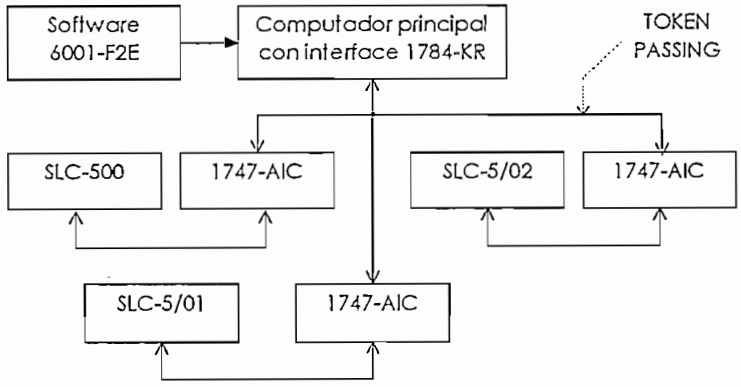


Fig. 2-17: Configuración típica de una red de PLC's utilizando la interface 1784-KR.

Se dispone de una red de tres PLC's interconectados a la computadora principal. Esta computadora posee de la tarjeta electrónica Allen-Bradley 1784-KR y del software Allen-Bradley 6001-F2E, para propósitos de adquisición de datos y supervisión del trabajo de la red. La comunicación entre los PLC's de la red se establece de acuerdo a un algoritmo de comunicación "token passing".

3. SISTEMA DE CONTROL PID

3. SISTEMA DE CONTROL PID

3.1 INTRODUCCION

Los computadores digitales tienen dos usos comunes en las situaciones de control real: control supervisor, donde los controladores analógicos originales se mantienen y el computador digital se usa para proveerlas de las señales de comando; y control digital directo (DDC), donde el computador digital está realmente en el lazo de control.

Partiendo del hecho de que un controlador lógico programable es un computador industrial digital, en la presente sección se implementará un control digital directo utilizando este dispositivo como elemento de control.

Como se conoce, un controlador digital genera salidas discretas en el tiempo de acuerdo a algún algoritmo de control, éste difiere de un controlador analógico en dos importantes aspectos:

- Las entradas al controlador digital deben ser cuantificadas o discretizadas, es decir, se requiere de un conversor analógico a digital y,
- Las salidas en un controlador digital son actualizadas solamente a intervalos discretos de tiempo, y no continuamente.

Es por esto que se requiere de un muestreador en el lado de las entradas y un retenedor en el lado de las salidas.

En contraste a los sistemas de tiempo continuo, los cuales son descritos por un conjunto de ecuaciones diferenciales y analizados mediante la transformada de Laplace; los sistemas de tiempo discreto son descritos por un conjunto de ecuaciones de diferencias y modelados mediante la transformada "Z". En general, cuando

aparecen integrales en la teoría de las ecuaciones diferenciales, aparecen sumas en la teoría de las ecuaciones en diferencias.

La condición de estabilidad para los sistemas discretos se expresa en términos del círculo unitario en el plano complejo, en lugar de expresarse en términos de división del plano a lo largo del eje imaginario.

El intervalo de muestreo, que se supone constante en el tiempo, es la característica distintiva de los sistemas discretos.

3.1.1 ECUACIONES EN DIFERENCIAS

Consideremos el siguiente sistema de control digital en lazo cerrado.

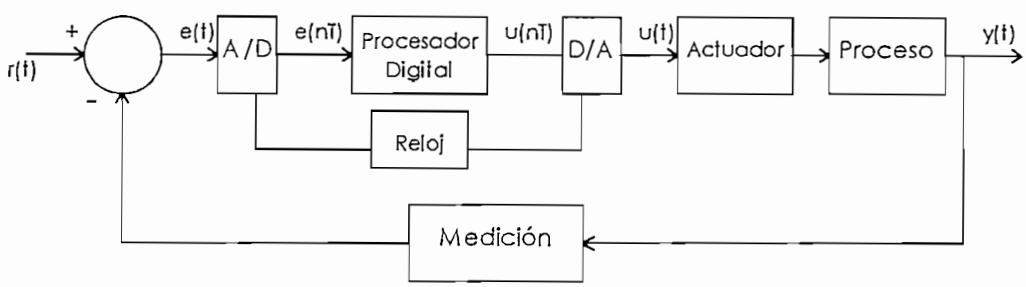


Fig. 3-1: Sistema de control digital en lazo cerrado.

Los muestreos de la señal de error e(t) son tomados a intervalos regulares de tiempo T. Hasta la k-ésima muestra son:

$$e_0, e_1, e_2, \ldots\ldots, e_k$$

De la misma manera, el procesador digital (PLC) produce las señales de salida. Previo al k-ésimo dato de entrada muestreado, estos son:

$$U_0, U_1, U_2, \ldots\ldots, U_{k-1}$$

Para obtener el próximo dato de salida U_k, el procesador digital debe evaluar algún tipo de función. En términos generales, esta función es:

$$U_n = f\,(e_0, e_1, e_2, \ldots\ldots, e_n, U_0, U_1, U_2, \ldots\ldots, U_{n-1})$$

Si se asume que la función es lineal y depende solamente de un número finito de valores anteriores, ésta puede ser escrita de la siguiente manera:

$$U_n = a_1U_{n-1} + a_2U_{n-2} \ldots. + a_kU_{n-k} + b_0e_n + b_1e_{n-1} \ldots. + b_mU_{n-m}$$

Ec. 3-1

que corresponde a una ecuación en diferencias con coeficientes constantes.

3.1.2 LA TRANSFORMADA Z Y LAS ECUACIONES EN DIFERENCIAS.

De la teoría de la transformada Z se tiene:

$$Z\,\{\,f(t)\,\} = F(z) = \sum_{n=0}^{\infty} f(nT)Z^{-n}$$

Ec. 3-2

y

$$Z\,\{\,f(t-T)\,\} = Z^{-1}\,F(z)$$

Ec. 3-3

Si t=nT, entonces:

$$Z\,\{\,f(nT)\,\} = Z\,\{\,f_n\,\} = F(z)$$

Ec. 3-4

y

$$Z\,\{\,f(nT-T)\,\} = Z\,\{\,f_{n-1}\,\} = Z^{-1}F(z)$$

Ec. 3-5

Se puede por lo tanto aplicar la Ec. 3-4 y la Ec. 3-5 a la Ec. 3-1 para obtener:

$$U(z) = [a_1z^{-1} + a_2z^{-2} + \ldots. + a_kz^{-k}]U(z) + [b_0 + b_1z^{-1} + \ldots. + b_mz^{-m}]E(z)$$

Ec. 3-6

$$D(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_k z^{-k}}$$

Ec. 3-7

si k≥m, entonces se puede escribir como:

$$D(z) = \frac{U(z)}{E(z)} = \frac{b_0 z^k + b_1 z^{k-1} + \ldots + b_m z^{k-m}}{z^k + a_1 z^{k-1} + a_2 z^{k-2} + \ldots + a_k} = \frac{b(z)}{a(z)}$$

Ec. 3-8

Una ecuación lineal en diferencias que describe la relación entre los datos muestreados de entrada y la señal de salida en un sistema de control digital, puede ser expresado como una función en el dominio Z y, de igual forma, de una función descrita en el dominio Z se puede encontrar una ecuación en diferencias equivalente.

3.1.3 CONTROL DIGITAL DIRECTO

El control digital directo o DDC (Direct Digital Control) constituye un sistema de control en el cual, un computador digital determina una acción reguladora en tiempo real ejecutando un programa almacenado en su memoria interna.

La Fig. 3-2 muestra el esquema de un DDC en el que se incluye en el computador digital (en este caso PLC), el comparador y el nivel de referencia.

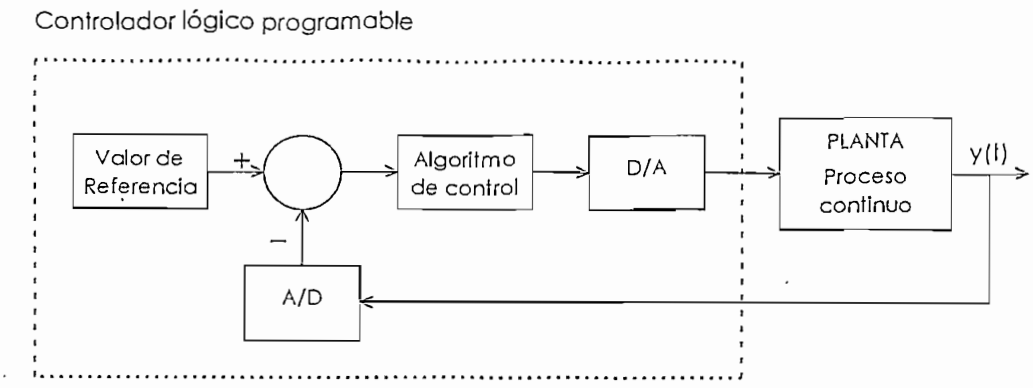


Fig. 3-2: Esquema de un Control Digital Directo.

La salida del proceso y(t) es una señal continua en el tiempo. Esta salida es muestreada y convertida en una señal digital usando un conversor analógico a digital, esta conversión es realizada a intervalos constantes de tiempo T segundos (periodo de muestreo). La señal muestreada medida es entonces procesada usando un algoritmo digital de control, el mismo que como resultado de su cálculo produce una señal de salida digital que posteriormente será convertida en una señal analógica mediante el uso de un conversor digital-analógico.

Por lo general, este tipo de sistemas de control dispone de una señal de reloj para sincronización de todo el proceso y las operaciones de entrada y salida de datos. Esta es una característica esencial de cualquier sistema de control digital.

VENTAJAS DEL CONTROL DIGITAL

La implementación de controladores mediante el uso de algoritmos de control digital tienen muchos beneficios si los comparamos con la contraparte analógica para el control de procesos. Estos beneficios pueden ser agrupados en las siguientes áreas:

A. Flexibilidad.
B. Multiplicidad de funciones.
C. Capacidad de hacer uso de técnicas de análisis y diseño avanzadas.
D. Comunicación con otros dispositivos.

FLEXIBILIDAD

Flexibilidad en el diseño del sistema de control, pudiendo pasar fácilmente de una acción de control a otra, diseñar la ecuación de control que más convenga al proceso, y añadir cómodamente acciones de control en adelanto o en cascada.

Asumiendo que las herramientas de programación convenientes están disponibles, al hacer un control digital es más fácil cambiar las especificaciones del lazo de control que con su contraparte analógica. Esta flexibilidad es una característica tanto de los sistemas pequeños como en sistemas mucho más complejos.

Con un esquema de control digital, el cambio de una señal o variable es una cuestión de reespecificación de datos mientras que en un sistema analógico es posible que se requiera de un cambio total en el cableado de la planta. Esta particularidad es muy evidente sobretodo en el área de los relés y los controles secuenciales donde los controladores lógicos programables prácticamente han sustituido a los sistemas analógicos de relés.

MULTIPLICIDAD DE FUNCIONES

Un controlador digital puede ser usado en muchas otras tareas además de la implementación del algoritmo de control propiamente dicho.

En la Fig. 3-3 se observa que en el caso analógico, el lazo de control requiere además del controlador, una etapa de linealización, amplificación y limitación de la señal procesada. Todas estas etapas son reemplazadas, en el control digital, por un instrumento microprocesado que ejecuta dichas funciones.

Al existir menos elementos, por ende, menos cableado, la confiabilidad del sistema se incrementará.

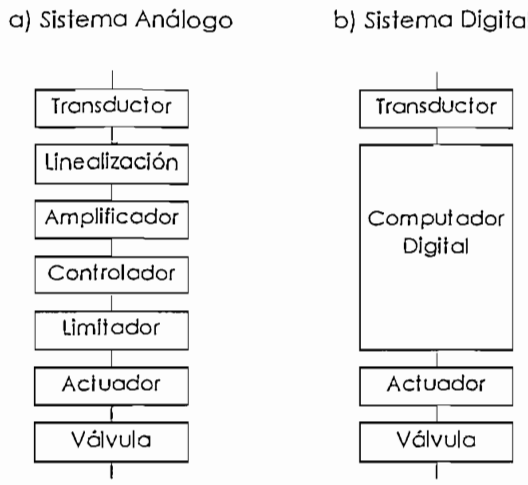


Fig. 3-3: Elementos en un lazo de control típico.

La característica de multiplicidad de funciones se observa también en el hecho de que un dispositivo digital puede manejar varios lazos de control simultáneamente.

TECNICAS AVANZADAS

El control digital posee la capacidad de implementar modernas técnicas de control avanzado que facilitan el control de sistemas donde exista interacción, significantes tiempos de retardo o en sistemas de orden superior donde las técnicas convencionales de control PI/PID no pueden ser satisfactoriamente utilizadas.

COMUNICACION

Una de las ventajas más importantes es la capacidad de intercomunicación con otros dispositivos o computadoras lo que posibilita el acceso a un sinnúmero de otras

actividades propias de estos dispositivos, tales como graficación, almacenamiento, impresión de datos, etc.

Esta característica de comunicación puede incrementar la seguridad del sistema al controlar los niveles de cualquier variable del proceso entre los límites prefijados. Además la interconexión de dispositivos permite la implementación de un sistema de supervisión de datos centralizado.

3.1.4 EL CONTROLADOR PID

Sin lugar a dudas, el sistema de control PID o controlador de tres términos, es uno de los sistemas más difundidos a nivel de procesos industriales. Tienen la capacidad de controlar adecuadamente una amplia gama de procesos dando una buena respuesta transitoria y de estado estable, es fácilmente comprensible y ajustable así como también fácil de implementarlo utilizando técnicas analógicas o digitales.

La acción de control PID está compuesta por la combinación de los efectos de la acción de control proporcional, acción de control integral y acción de control derivativa.

El control proporcional es esencialmente un amplificador con ganancia ajustable en el cual la salida del controlador y la señal de error actuante están relacionadas por:

$$u(t) = Kp \cdot e(t)$$

Ec. 3-9

En un control con acción integral, la variación de la señal de salida del controlador es proporcional a la señal de error actuante, es decir:

$$\frac{\partial u(t)}{\partial t} = k_i \cdot e(t)$$

Ec. 3-10

o, en otras palabras:

$$u(t) = Ki \cdot \int e(t)\,dt$$

Ec. 3-11

Si se duplica el valor del error, la señal de salida del controlador varía dos veces más rápido. Para un error actuante igual a cero, el valor de u(t) se mantiene estacionario.

Es importante notar, que si bien la acción integral elimina el error en estado estable, puede llevar al sistema a respuestas oscilatorias en amplitud lentamente decrecientes o en amplitud creciente, ambas generalmente indeseables.

La acción derivativa se produce cuando el valor de salida del control es proporcional a la velocidad de variación de la señal de error actuante, lo que puede producir una corrección significativa del error antes de que su valor se haga excesivo.

$$u(t) = k_d \frac{\partial e(t)}{\partial t}$$

Ec. 3-12

De este modo, el control derivativo se anticipa al error actuante, inicia una acción correctiva temprana y tiende a aumentar la estabilidad del sistema. Aunque el control derivativo no afecta directamente al error en estado estacionario, añade amortiguamiento al sistema y, por tanto, permite el uso de un valor de ganancia Kp más elevado, lo que produce un mejoramiento en la exactitud del estado estacionario.

Esta acción es usada principalmente en sistemas con excesivos tiempos de retardo o en sistemas de orden mayor o igual a 3.

Resulta muy conveniente, para la implementación de la acción derivativa, reemplazar la expresión $\partial e/\partial t$ por $\partial y/\partial t$ para evitar la derivación de la señal de referencia ($e = y - r$).

3.2 ALTERNATIVAS DE DISEÑO

Existen diversas tipos de sistemas programables que pueden ser utilizados en la fase de tratamiento de la información para la realización de un sistema automático de control, entre estos tenemos:

- Controladores lógicos programables.
- Microcomputadores industriales.
- Tarjetas electrónicas con microprocesadores dedicados.
- Controladores numéricos.

Entre las consideraciones técnicas que pueden ser tomadas para elegir el mejor sistema se mencionan las siguientes[1]:

a. Cantidad de equipos para el proceso:

El proceso puede ser controlado con equipos individuales, en el caso de producciones en serie, o con varios equipos para producciones diversificadas; en este caso, la elección se da por el factor económico del equipo. Si el proceso se da con equipos en serie, los mejores sistemas son los PLC's conectados en red, donde cada equipo o cada máquina es controlada por un PLC.

b. Tipo de ambiente:

Puede ser explosivo o no explosivo. En el caso de sistemas que trabajen bajo ambiente explosivo no se recomienda usar equipos electrónicos directamente, la solución se da con un sistema de comando local neumático controlado desde un computador.

---

[1] Ortega Edgar, "Sistema de transferencia automático de carga y sincronización para la estación Cotopaxi utilizando un controlador lógico programable", EPN-FIE, 1993.

c. Señales de entrada/salida:

El tipo de señal de entrada y salida encontrada en las aplicaciones es muy importante. El voltaje de control para las entradas y salidas se deben unificar de acuerdo a las normas. Todos los métodos de solución son válidos, sin embargo, dentro de la línea de PLC's se tiende más a al estandarización en cuanto a la utilización de los diferentes módulos de E/S.
Cuando se dispone de estructuras modulares en base a PLC's, la expansión del sistema de E/S resulta mucho más fácil.

d. Comunicación:

La comunicación es importante en los sistemas actuales, todo proceso además del diálogo hombre-máquina realizado con luces de señalización y pulsantes requiere comunicaciones con sistemas centrales de administración, supervisión y/o control. Los PLC's y los sistemas de control por computador cumplen esta condición, con la ventaja que los PLC's están hechos para trabajar en ambientes industriales; no así un computador personal que requiere de cuidados especiales en cuanto a la alimentación y perturbaciones eléctricas.

Los sistemas electrónicos dedicados han dejado de ser utilizados mayoritariamente debido a la gran dinámica y versatilidad que se requiere en la producción industrial actual, la falla de un sistema dedicado provoca pérdidas por el tiempo requerido para su reemplazo.

Un controlador numérico se usa casi exclusivamente en maquinaria con tratamiento lógico, control de ejes y sistemas de máquinas-herramientas (troqueladoras, tornos, etc.).

La solución más confiable para casi todo sistema consiste en la utilización de PLC's asociados a máquinas individuales, con la posibilidad de que todos los PLC's se

conecten entre si con un sistema de red en la cual uno de ellos es el "maestro"; todo este sistema se puede enlazar con un computador para supervisión y control y, finalmente con programas informáticos llegar a un sistema de gestión y control desde un computador central.

VENTAJAS DE UN SISTEMA PROGRAMABLE

En un sistema con controladores lógicos programables, la función de mando se establece en un programa almacenado en una memoria, reduciéndose el cableado solamente a la conexión de los sensores y los actuadores, por tanto, las variaciones funcionales, pruebas y puesta en marcha resultan sencillas y fáciles de realizar.

El diseño de sistemas de control de procesos utilizando los controladores lógicos programables, presentan frente a los métodos tradicionales, las siguientes ventajas:

- Al ser un sistema electrónico, es compacto y requiere poco espacio para el montaje.
- Un proyecto se lo realiza en menor tiempo ya que se disminuye la complejidad del cableado del sistema externo.
- Facilidad de modificación. En caso de variación sólo hay que modificar el programa, manteniéndose el cableado.
- Puesta en marcha fácil del proceso.
- Diagnóstico de fallas más simple, por la presencia de auto-test integrados.

Como contraparte a estas ventajas se puede mencionar que el costo inicial de un sistema con PLC es alto, aunque se compensa con el bajo costo de mantenimiento.

Por las características de seguridad y concepción, de acuerdo con las tecnologías electrónicas actuales, el controlador lógico programable es un elemento que puede aplicarse a todo sistema de automatización y control de procesos.

El diseño del controlador PID para la implementación digital en un controlador lógico programable puede ser realizado mediante dos métodos: efectuar un diseño continuo (plano S) del sistema de control de datos muestreados o realizar un diseño discreto (plano Z) del mismo sistema. A continuación se estudiará cada uno de los métodos.

3.2.1 DISEÑO EN EL PLANO "S" DE UN SISTEMA DE CONTROL DE DATOS MUESTREADOS.

El diseño en el plano "S" o diseño continuo de un sistema de control de datos muestreados se lleva a cabo en dos etapas:

- La compensación se diseña de la misma manera que para un sistema de control continuo.
- Se busca que la implementación digital del compensador de el mejor equivalente discreto de la compensación continua.

3.2.1.1 PROCEDIMIENTOS DE DISCRETIZACION.

Consideremos el sistema de control mostrado en la Fig. 3-4.

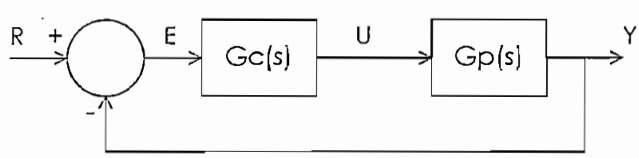


Fig. 3-4: Sistema de control básico.

La implementación digital requiere que la salida Y(t) sea muestreada a una velocidad determinada y que la computadora digital produzca señales de salida u(t) suavizadas

y continuas probablemente por medio de un "zero-order hold", tal como se muestra en la Fig. 3-5

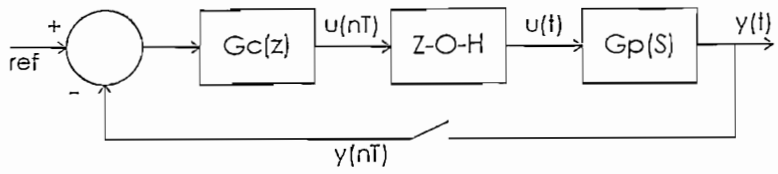


Fig. 3-5: Sistema de control incluido el muestreador y el retenedor.

Es necesario entonces, encontrar el "mejor" Gc(z) para reemplazar a Gc(s), o en otras palabras encontrar el mejor "mapeo de la variable s" en términos de la variable z.

Varias aproximaciones pueden ser adoptadas simplemente haciendo diferentes suposiciones de lo que sucede entre dos datos muestreados. Se pueden considerar tres aproximaciones:

- Integración numérica.
- Mapeo de polos y ceros.
- Equivalente hold.

Una de las ventajas de usar estos métodos radica en el hecho de que el periodo de muestreo no necesita ser escogido hasta después de que el diseño de la acción de control sea llevado a cabo.

3.2.1.1.1 INTEGRACION NUMERICA.

Primero se representa una función de transferencia continua como una ecuación diferencial, luego la ecuación se la pone en forma integral, para posteriormente reemplazando la integral por métodos aproximados llegar a una ecuación en diferencias cuya solución se acerca a la solución de la ecuación diferencial. Esta

integral es aproximada geométricamente por el área de un rectángulo, hacia adelante, hacia atrás o el área de un trapecio (bilineal). El resultado de este método se muestra en la siguiente tabla[2]:

Tabla 3-1: Equivalencias entre los planos S y Z usando integración numérica.

METODO	APROXIMACION
Rectángulo hacia adelante	$s \approx \frac{z-1}{T}$
Rectángulo hacia atrás	$s \approx \frac{z-1}{T \cdot z}$
Trapezoidal o bilineal	$s \approx \frac{2}{T}\left(\frac{z-1}{z+1}\right)$

3.2.1.1.2 MAPEO DE POLOS Y CEROS.

Este método se basa en encontrar por extrapolación la relación entre los planos S y Z. Para una señal continua e(t), los polos de E(z) se relacionan con los polos de E(s) de acuerdo a la relación $z=e^{sT}$. Las reglas de este método son:

1. Todos los polos de E(s) son mapeados de acuerdo a $z = e^{sT}$.
   Si E(s) tiene un polo en $s = -a$, entonces E(z) tiene un polo en $z = e^{-aT}$.
2. Todos los ceros finitos son mapeados por $z = e^{sT}$.
   Si E(s) tiene un cero en $s = -b$, E(z) tiene un cero en $z = e^{-bT}$.
3. Todos los ceros de E(s) en $s = \infty$ son mapeados en E(z) al punto $z = -1$.
4. Se selecciona la ganancia de tal manera que $E(s)\,|_{s=0} = E(z)\,|_{z=1}$.

---

[2] Valencia Javier, "Análisis y diseño de sistemas de control de tiempo discreto asistido por computador, tesis EPN-FIE, 1994.

3.2.1.1.3 EQUIVALENTE HOLD.

Este método consiste en utilizar un retenedor de orden cero, lo que proporciona un buen mecanismo de discretización.

$$G(z) = Z\left[\frac{1-e^{-sT}}{s} G(s)\right]$$

Ec. 3-13

Por lo mencionado en la sección 3.1, este método es usado básicamente para la discretización de la función de transferencia de la planta.

3.2.1.2 DISEÑO DEL CONTROLADOR PID

Experimentalmente se ha comprobado que la mejor aproximación es discretizar la parte integral del controlador PID utilizando la relación bilineal y la parte derivativa utilizando la integración de rectángulo hacia atrás (Tabla 3-1).

El controlador PID implementado posee la configuración mostrada en la siguiente figura.

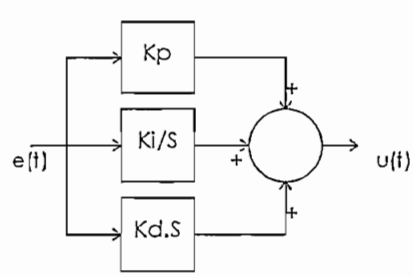


Fig. 3-6: Diseño continuo del controlador PID.

Es decir:

$$Gc(s) = Kp + \frac{Ki}{S} + Kd \cdot S$$

Ec. 3-14

Discretizando con la consideración hecha anteriormente se tiene:

$$Gc(z) = \frac{U(z)}{E(z)} = \frac{bo + b1 \cdot z^{-1} + b2 \cdot z^{-2}}{1 - z^{-1}}$$

Ec. 3-15

como ecuación en diferencias:

$$u(k) = bo \cdot e(k) + b1 \cdot e(k-1) + b2 \cdot e(k-2) + u(k-1)$$

Ec. 3-16

donde:

$$b0 = Kp + \frac{Kd}{T} + \frac{Ki \cdot T}{2}$$

$$b1 = -Kp - \frac{2Kd}{T} + \frac{Ki \cdot T}{2}$$

$$b2 = \frac{Kd}{T}$$

La Ec. 3-16 es el algoritmo digital resultado del diseño en el plano "S" del controlador PID para el sistema de datos muestreados.

3.2.2 DISEÑO EN EL PLANO "Z" DE UN SISTEMA DE CONTROL DE DATOS MUESTREADOS.

3.2.2.1 HERRAMIENTAS PARA EL ANALISIS.

El primer paso en el diseño de sistemas de control de datos muestreados es discretizar los elementos continuos presentes.

Consideremos el siguiente diagrama (Fig. 3-7).

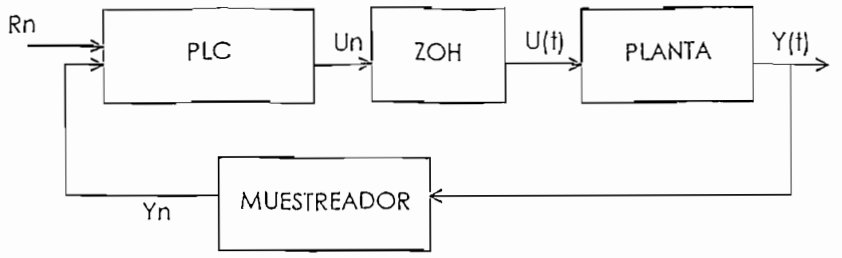


Fig. 3-7: Esquema de control.

Para una planta descrita por una función de transferencia continua Gp(s) y precedida por un Zero-order hold, la función de transferencia discreta está dada por:

$$Gp(z) = \left(1 - z^{-1}\right) Z\left\{\frac{G(s)}{s}\right\}$$

Ec. 3-17

La ecuación anterior permite reemplazar el sistema mixto (sistema continuo y discreto) mostrado en la Fig. 3-8 con el sistema equivalente puramente discreto mostrado en la Fig. 3-9.

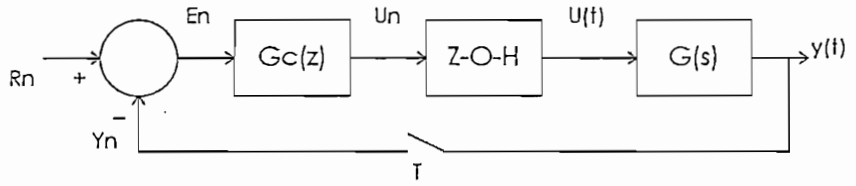


Fig. 3-8: Sistema mixto, continuo y discreto.

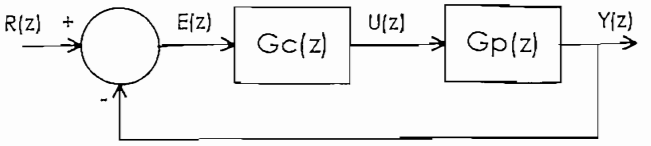


Fig. 3-9: Sistema equivalente puramente discreto.

Una vez que el sistema ha sido discretizado, el análisis y el diseño del compensador es muy similar al análisis con sistemas continuos y las mismas reglas son aplicadas. La función de transferencia en lazo cerrado se obtiene siguiendo las reglas de reducción de los diagramas de bloques de sistemas continuos, esto es:

$$\frac{Y(z)}{R(z)} = \frac{Gc(z)\cdot Gp(z)}{1+Gc(z)\cdot Gp(z)}$$

Ec. 3-18

Las raíces características del sistema en lazo cerrado se obtienen resolviendo la siguiente ecuación:

$$1+Gc(z)\cdot Gp(z)=0$$

Ec. 3-19

y cualquier técnica de diseño discreto (por ejemplo lugar geométrico de las raíces discreto, cancelación de polos y ceros, etc.) puede ser aplicado sin inconvenientes.

3.2.2.2 DISEÑO DEL CONTROLADOR PID

En sistemas continuos, el diseño utiliza controladores del tipo proporcional, integral o derivativo, o la combinación de estas formas para obtener controladores de adelanto de fase, retardo de fase o compensadores adelanto-retraso.

En el campo del control discreto, de manera similar, las leyes simples de control son las siguientes:

a. CONTROLADOR PROPORCIONAL:

$$U_n = Kp \cdot E_n$$

Ec. 3-20

$$Gc(z) = Kp$$

Ec. 3-21

b. CONTROLADOR DERIVATIVO PURO:

$$U_n = Kd\left(E_n - E_{n-1}\right)$$

Ec. 3-22

$$Gc(z) = Kd\left(1 - z^{-1}\right) = Kd\left(\frac{z-1}{z}\right)$$

Ec. 3-23

c. CONTROLADOR INTEGRAL:

$$U_n = U_{n-1} + Ki \cdot E_n$$

Ec. 3-24

$$Gc(z) = \frac{Ki}{1 - z^{-1}} = \frac{Ki \cdot z}{z-1}$$

Ec. 3-25

Con los controladores indicados anteriormente se puede construir un controlador PID discreto de la siguiente manera:

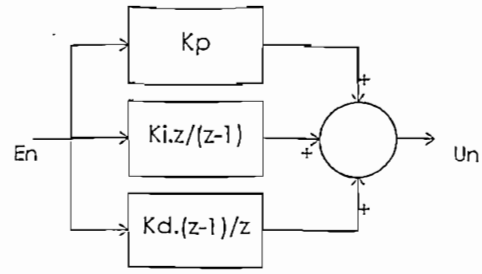


Fig. 3-10: Diseño discreto del controlador PID.

En forma de función de transferencia:

$$Gc(z) = Kp + Kd\left(\frac{z-1}{z}\right) + Ki\left(\frac{z}{z-1}\right)$$

Ec. 3-26

desarrollando:

$$Gc(z) = \frac{U(z)}{E(z)} = \frac{z^2(Kp+Ki+Kd) + z(-Kp-2Kd) + (Kd)}{(z-1)z}$$

Ec. 3-27

$$Gc(z) = \frac{U(z)}{E(z)} = \frac{(Kp+Ki+Kd) + z^{-1}(-Kp-2Kd) + z^{-2}(Kd)}{1-z^{-1}}$$

Ec. 3-28

Convirtiendo la ecuación anterior en una ecuación en diferencias se tiene:

$$u(k) - u(k-1) = (Kp+Ki+Kd)\cdot e(k) + (-Kp-2Kd)\cdot e(k-1) + (Kd)\cdot e(k-2)$$

Ec. 3-29

La Ec. 3-29 constituye la ecuación en diferencias de un controlador PID discreto diseñado en el plano "Z".

3.3 SISTEMA DE CONTROL PID UTILIZANDO PLC's

A continuación se detalla tanto la implementación del controlador PID en un PLC sin rutina PID como también en uno que si dispone dicha instrucción.

3.3.1 SISTEMA DE CONTROL UTILIZANDO PLC SIN RUTINA PID.

En un controlador lógico programable que no dispone de rutina de control PID como es el caso de PLC's con escaso número de funciones, el algoritmo de control PID se lo debe implementar en base a las funciones básicas propias que éste posee.

En el presente trabajo se utiliza un PLC Allen-Bradley de la familia SLC-500. Este PLC es programado con una ecuación en diferencias correspondiente a un controlador PID.

La Ec. 3-16 (diseño continuo del controlador PID) o la Ec. 3-29 (diseño discreto del controlador PID) al ser programada en el PLC posibilita a este dispositivo ejecutar un control del tipo PID en un proceso determinado.

Evaluar directamente dichas ecuaciones en el controlador lógico programable, en muchas ocasiones no resulta conveniente por cuanto al existir una limitación en el rango de la cifras que puede manejar el PLC (-32768, +32767), muchas de las operaciones matemáticas de esas ecuaciones producen resultados fuera del rango permitido. Por esta razón es conveniente, para la implementación final en el PLC, utilizar una modificación matemática que consiste en obtener incrementos de la variable e(k), de la siguiente manera:

Diseño continuo del controlador PID:

$$u(k) = Kp \cdot [\, e(k) - e(k-1) \,] + \frac{Ki \cdot T}{2} [\, e(k) + e(k-1) \,] + \frac{Kd}{T} \cdot [\, e(k) - e(k-1) \,] + \\ + \frac{kd}{T} \cdot [\, e(k-2) - e(k-1) \,] + u(k-1)$$

Ec. 3-30

Diseño discreto del controlador PID:

$$u(k) = Kp \cdot [\, e(k) - e(k-1) \,] + Ki \cdot [\, e(k) \,] + Kd \cdot [\, e(k) - e(k-1) \,] + kd \cdot [\, e(k-2) - e(k-1) \,] + u(k-1)$$

Ec. 3-31

Las ecuaciones así obtenidas evitan el sobrerango de sus términos por lo que la evaluación en el PLC no presenta mayores problemas.

Sin embargo, el hecho de que el PLC solamente pueda manejar **valores enteros** comprendidos en el rango de -32768 a +32767 obliga que la variación de las constantes del controlador PID (Kp, Ki, Kd) únicamente sean valores enteros (mínima variación = 1). Esto evidentemente conduce a una imprecisión en la determinación de los coeficientes de la ecuación en diferencias del controlador PID que sin lugar a dudas produciría un sistema controlado de menor calidad.

Con el propósito de disminuir este inconveniente y permitir una mayor precisión en las constantes del controlador PID, se procede a dividir cada parámetro del PID para un valor constante igual a 100 de tal manera de tener la posibilidad de ingresar dichos parámetros hasta con dos números decimales, así:

Diseño continuo del controlador PID:

$$u(k) = \frac{Kp}{100} \cdot [\, e(k) - e(k-1) \,] + \frac{Ki \cdot T}{20000} [\, e(k) + e(k-1) \,] + \frac{Kd}{T} \cdot [\, e(k) - e(k-1) \,] + \\ + \frac{kd}{T} \cdot [\, e(k-2) - e(k-1) \,] + u(k-1)$$

Ec. 3-32

Diseño discreto del controlador PID:

$$u(k) = \frac{Kp}{100}\cdot[e(k) - e(k-1)] + \frac{Ki}{100}\cdot[e(k)] + \frac{Kd}{100}\cdot[e(k) - e(k-1)\ ] + \frac{kd}{100}\cdot[\ e(k-2) - e(k-1)\ ] + u(k-1)$$

Ec. 3-33

Esta operación puede ser llevada a cabo gracias a la ayuda de la instrucción DDV (doble división) que incluye el PLC Allen-Bradley en su conjunto de operaciones básicas.

Cualquiera de las ecuaciones anteriores puede ser implementada en el controlador lógico programable para él control PID, con sus respectivas consideraciones de diseño.

3.3.1.1 DESCRIPCION DEL PROGRAMA DESARROLLADO.

El programa desarrollado para el control PID en un PLC que no dispone de esta rutina (ver anexo E, programa PID_CONT.ACH o PID_DISC.ACH), lee la entrada del módulo analógico, evalúa una de las ecuaciones en diferencias determinadas anteriormente (Ec. 3-32 o Ec. 3-33) y produce una señal de control analógica de salida. El programa en si es simple y eficaz.

Dispone de una bandera (HABIL) que habilita o deshabilita la ejecución del algoritmo PID. Con HABIL=0, el programa se encuentra en modo MANUAL y el usuario puede cambiar el valor de salida del módulo analógico a su conveniencia. Con HABIL=1, se activa el modo AUTOMATICO y el PLC, en base a sus parámetros, es el que controla la salida analógica.

En su primera parte, el programa actualiza el archivo de datos de entrada y realiza un movimiento de estos datos a la DATA FILE 9 con el solo propósito de permitir la comunicación con el computador. Posteriormente efectúa el cálculo del error, restando el valor de la señal de entrada del valor de referencia.

Luego se determinan las diferencias de los errores actuales con los errores anteriores de acuerdo a la ecuación en diferencias utilizada. Cada factor así obtenido es multiplicado por la constante correspondiente y posteriormente dividido por un factor de 100. Cabe resaltar que luego de cada operación matemática realizada en el PLC, es fundamental efectuar un chequeo de la bandera de overflow (S:0/1) para determinar si existió o no un sobrerango en el resultado de la operación matemática. En caso de existir sobrerango es necesario desactivar el bit de error menor que se genera debido a esta condición. Si no se desactiva esta bandera, cuando se termine la exploración del programa se declarará un error mayor y el PLC entrará en modo FAULT, dejando de realizar el barrido del programa.

Una vez encontrados estos productos son sumados hasta obtener el valor de u(k) resultante. En una etapa final del programa se verifica que el valor de u(k) no sea negativo y se actualiza el archivo de datos de salida con este valor obtenido. Para finalizar, se realiza el movimiento de la señal calculada u(k) a la DATA FILE 9 (para que el computador adquiera este dato), además del proceso de actualización tanto de los valores de la señal de error como de los valores de la señal actuante.

3.3.2 SISTEMA DE CONTROL UTILIZANDO PLC QUE INCLUYE RUTINA PID.

Actualmente muchos de los controladores lógicos programables incluyen, para procesos industriales variados, módulos con rutinas de control PID. La ventaja de estos módulos es que no requieren una programación para su funcionamiento, tan solo necesita una elección adecuada de sus parámetros.

Dentro de la familia Allen-Bradley SLC-500 de controladores lógicos programables, la instrucción PID es solamente aplicable a los procesadores 5/02 y 5/03.

La rutina PID es una instrucción de salida que permite controlar variables físicas tales como temperatura, presión, nivel de líquidos o velocidad de flujo entre otras, usando lazos de proceso.

La instrucción PID normalmente controla un lazo cerrado, usando para el efecto una entrada proveniente de un módulo de entradas analógicas y dando como resultado una salida a un módulo de salidas analógicas.

La instrucción PID puede ser operada en dos modos de funcionamiento: el modo TIMED y el STI. En el modo TIMED, la instrucción actualiza su salida periódicamente a una velocidad seleccionable por el usuario. En el modo STI, la instrucción deberá ser situada en una subrutina de interrupción STI. Esta, entonces, actualiza su salida cada vez que la subrutina STI es leída o explorada. El intervalo de tiempo STI y la velocidad con la que se actualiza el lazo PID deberán ser del mismo orden para que la ecuación se ejecute adecuadamente.

En la Fig. 3-11 se muestra un ejemplo del control PID en una aplicación utilizando la instrucción PID que incluye el PLC.

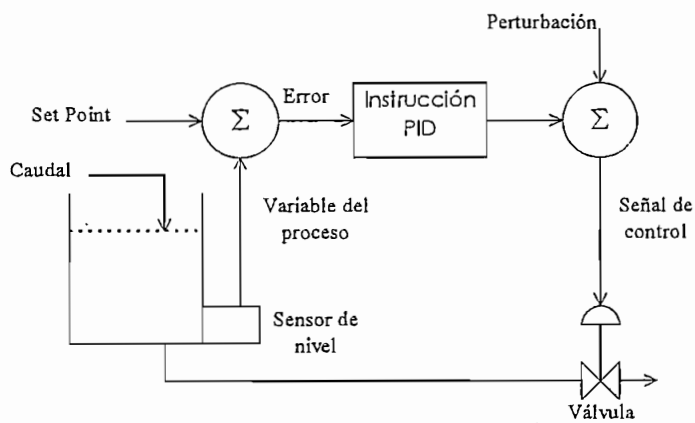


Fig. 3-11: Aplicación típica de un control PID.

Existe la posibilidad de incluir una señal de perturbación como un offset en los cálculos de la instrucción PID.

A. *LA ECUACION PID*

La instrucción PID usa el siguiente algoritmo para evaluar la señal de salida:

$$Salida = K_C \cdot [E + \frac{1}{T_i} \int E \cdot dt + T_D \cdot \frac{\partial (PV)}{\partial t}] + Perturbación$$

Ec. 3-34

Donde: Kc, Ti y Td son los términos del controlador PID.

E, es la señal de error.

PV, constituye la variable del proceso, es decir, la entrada analógica previamente escalada.

Salida, es la señal de control escalada.

Perturbación, es una señal correspondiente a una perturbación.

El rango de las constantes del controlador PID al utilizar esta instrucción es:

Tabla 3-2: Constantes PID permitidas en el PLC con rutina PID.

Término	Rango (Bajo-Alto)	Referencia
K_C	0.1 - 25.5 0.01 - 327.67*	Proporcional
1/Ti	25.5 - 0.1 327.67 - 0.01*	Integral (minutos por repetición)
T_D	0.01 - 2.55 0.01 - 327.67*	Derivativo (minutos)

* Válido para procesador 5/03 con bit RG=1.

El término derivativo está construido de una manera suave, por medio de un filtro pasa bajo.

B. INGRESO DE PARAMETROS

Normalmente se sitúa la instrucción PID en una "rung" sin condiciones lógicas. La salida permanece en su último valor cuando la "rung" es falsa. El término integral es también borrado cuando la "rung" es falsa.

Durante la programación se deben ingresar los siguientes parámetros:

- Bloque de control (Control Block)
- Variable del proceso (Process Variable), y
- Variable de control (Control Variable).

BLOQUE DE CONTROL

Es un archivo que almacena los datos requeridos para procesar la información. La longitud del archivo está fijado en 23 palabras y deberán ser ingresadas como una dirección de un archivo de tipo entero. Por ejemplo, ingresar N10:0 localizará elementos desde N10:0 hasta N10:22.

Se recomienda no escribir en la dirección del bloque de control otros valores, excepto los descritos posteriormente. Si se está re-usando un bloque de datos, el cual fue previamente usado para otros propósitos, es aconsejable borrar los datos primero. Se recomienda también usar un único archivo de datos que contenga el bloque de control del PID (por ejemplo N10:0). Esto impide usos accidentales de las direcciones del bloque de control del PID en otras instrucciones del programa.

VARIABLE DEL PROCESO (PV)

Esta variable es una dirección que almacena el valor de la variable de entrada del proceso. Esta dirección puede ser la localización de la palabra de una entrada analógica donde es almacenado el valor de la entrada del A/D. Este valor puede ser también un valor entero

siempre y cuando se elija un pre-escalamiento del valor de entrada en un rango de 0 a 16383.

VARIABLE DE CONTROL (CV)

Es una dirección que almacena el resultado del procesamiento de la instrucción PID. El valor de salida posee un rango comprendido entre 0 y 16383. Este valor es normalmente un entero que puede ser escalado a un rango particular de la salida analógica para una determinada aplicación.

Después de ingresar los valores adecuados del bloque de control, la variable del proceso y la variable de control, el software de programación APS muestra una pantalla con los siguientes datos:

F1	auto/manual:	AUTO		time mode bit:	1	TM
F2	mode:	STI		auto/manual bit:	0	AM
F3	control:	E=SP-PV		control mode bit:	0	CM
	setpoint (SP):	0		output limiting enabled bit:	0	OL
	process (PV):	0		reset and gain range:	0	RG
	scaled error:	0		scale setpoint flag:	0	SC
	deadband:	0		loop update time too fast:	0	TF
	output (CV):	0	%	derivitive (rate) action:	0	DA
				DB, set when error is in DB:	0	DB
	loop update:	5	[.01 secs]	output alarm, upper limit:	0	UL
	gain:	50	[/10]	output alarm, lower limit:	0	LL
	reset:	25	[/10 m/r]	setpoint out of range:	0	SP
	rate:	10	[100 min]	process var out of range:	0	PV
	min scaled:	0		PID done:	0	DN
	max scaled:	1000				
F4	output (CV) limit:	NO				
	output (CV) min:	0	%			
	output (CV) max:	100	%			

Enter value or press <ESC> to exit monitor
N7:2 =
offline no forces File PID502

TOGGLE AUTO/MN	TOGGLE MODE	TOGGLE CONTROL	TOGGLE OUT LIM
F1	F2	F3	F4

Fig. 3-12: Pantalla para el ingreso de datos en instrucción PID.

[F1] AUTO/MANUAL AM (palabra 0, bit 1).

AUTO indica que el PID está controlando la salida (el bit es desactivado). MANUAL indica que el usuario está ingresando el valor de salida (el bit es activado), es decir, la salida no es determinada por medio de la instrucción PID del PLC. La calibración de los parámetros se recomienda hacerlo en modo manual y posteriormente regresar al modo automático. La limitación en el rango de la variable de salida también es aplicable al modo manual.

[F2] MODE TM (palabra 0, bit 0).

Permite el cambio de modo de operación del PID. En el modo TIMED la salida es actualizada a la velocidad especificada en el parámetro de actualización de lazo.
Cuando se use el modo de operación TIMED el tiempo de exploración del proceso deberá ser al menos 10 veces más rápido que el tiempo de actualización del lazo, para prevenir imprecisiones de tiempo o perturbaciones en el funcionamiento.

El modo STI indica que el PID actualiza la salida cada vez que esta subrutina es barrida. Cuando se selecciona el modo STI, la instrucción PID deberá ser programada en una subrutina de interrupción STI, y la subrutina STI deberá tener un intervalo de tiempo igual al fijado en el parámetro de actualización del lazo PID. La determinación del período STI se lo realiza en la palabra S:30 del status del PLC.

[F3] CONTROL CM (palabra 0, bit 2)

La instrucción PID puede trabajar en dos modos de control: considerando al error como E=PV-SP o como E=SP-PV. En el primer modo (E=PV-SP), el PLC produce un incremento de la señal de salida CV cuando la entrada PV es más grande que el setpoint SP. En el segundo modo de operación (E=SP-PV), la salida CV se incrementa cuando la señal de entrada PV es menor que el setpoint SP.

Set point (palabra 2):

Es el punto deseado de control de la variable de proceso. Se puede cambiar este valor desde el programa de escalera usando instrucciones para el efecto, escribiendo su valor en la tercera palabra del bloque de control (por ejemplo, escribir el valor en N10:2 si el bloque de control es N10:0). Sin escalamiento, el rango del set point es [0 ; 16383]. De otra manera, el rango es: [mínimo valor escalado (palabra 8) ; máximo valor escalado (palabra 7)].

Constante proporcional K_C (palabra 3):

Es la ganancia proporcional. Varía entre 0.1 y 25.5. Se recomienda poner esta ganancia a la mitad del valor necesario para que la salida oscile cuando el término integral y derivativo son puestos a cero. Para un procesador 5/03, el rango válido es [0.01 ; 327.67] (RG=1).

Constante integral Ti (palabra 4):

Es la ganancia integral. Varía entre 0.1 y 25.5 minutos por repetición. Una regla útil es colocar el tiempo integral igual al período natural medido sobre la calibración de ganancia. Note que un valor igual a 1 añadirá el mínimo término integral posible en la ecuación del PID. El rango válido para un procesador 5/03 es 0.01 a 327.67 minutos por repetición (RG=1).

Constante derivativa Td (palabra 5):

Es el término derivativo de la instrucción PID. El rango de ajuste es de 0.01 a 2.55 minutos. Una regla útil es fijar este valor en 1/8 del tiempo integral. Cuando se ejecute la instrucción PID en un procesador 5/03, el rango válido es 0.01 a 327.67 minutos.

Máximo escalado Smax (palabra 7):

Si el set point es leído en unidades de ingeniería (unidades físicas), este parámetro corresponde al valor del set point en unidades de ingeniería cuando la entrada de control es 16383. El rango válido es [-16383 ; +16383]. Para procesador 5/03: [-32768 ; +32767].

Mínimo escalado Smin (palabra 8):

Si el set point es leído en unidades de ingeniería, este parámetro corresponde al valor del set point en unidades de ingeniería cuando la entrada de control es cero. Rango válido [-16383 ; +16383]. Para procesador 5/03: [-32768 ; +32767].

Las palabras Smin y Smax permiten ingresar el valor del set point en unidades de ingeniería. La banda muerta, el error y la variable de proceso PV serán mostrados también en unidades de ingeniería. La variable de proceso PV, se espera también tenga un rango de 0 a 16383.
El uso del Smin y Smax no minimizan la resolución del PID.

Banda muerta DB (palabra 9):

Es un valor no negativo. La banda muerta se extiende encima y abajo del set point por el valor ingresado. La banda muerta es declarada en el cruce por cero de la variable de proceso PV y del set point SP. Esto significa que la banda muerta entra en efecto solamente cuando la variable de proceso PV entra en ella y pasa a través del set point. El rango válido es [0 ; Smax], o [0 ; 16383] cuando no existe escalamiento.

Actualización del lazo (palabra 13):

Corresponde al intervalo de tiempo entre cálculos del PID. Los valores se pueden ingresar en intervalos de 0.01 segundos. Una regla útil es ingresar un tiempo de actualización del lazo 5 o 10 veces más rápido que el período natural de oscilación (determinado poniendo las constantes integral y derivativa a cero e incrementando la ganancia hasta que la salida comience a oscilar). Cuando se opera en el modo STI, este valor debe ser igual al valor del intervalo de tiempo STI fijado en la palabra S:30 del status del PLC. El rango válido es 1 a 2.55 segundos. Para procesador 5/03: 1 a 10.02 segundos.

Valor escalado del proceso (palabra 14):

Es un dato solamente de lectura y corresponde al valor escalado de la variable del proceso PV (entrada analógica). Sin escalamiento, el rango de este parámetro es [0 ; 16383]. Con escalamiento el rango es [mínimo escalado (palabra 8) ; máximo escalado (palabra 7)].

Error escalado (palabra 15):

Dato solamente de lectura. Este es el error escalado seleccionado por el parámetro de modo de control (tecla de función F3). Rango: [máximo escalado ; - máximo escalado] o, [16383 ; -16383] cuando no existe escalamiento.

Salida CV% (palabra 16):

Muestra la salida actual de 0 a 16383 en términos de porcentaje (rango: 0-100%). Si se selecciona el modo AUTO con la tecla de función F1, este valor es solamente de lectura. Cuando se seleccione el modo MANUAL y se use el monitor de datos del APS, se puede

cambiar la salida CV% y el cambio será aplicado a CV. Cuando no se esté usando el software APS se debe escribir directamente en CV un valor comprendido entre 0 y 16383.

[F4] OUTPUT (CV) LIMIT (OL) (palabra 0, bit 3):

Cambia entre SI y NO. Seleccione SI, si se desea limitar la salida a los valores mínimo y máximo.

Tabla 3-3: Tecla de función [F4] Output (CV) Limit.

Salida CV%	SI(1). Límite de la salida CV seleccionado.	NO(0). Límite de la salida CV no seleccionado.
min	El valor ingresado será el porcentaje de salida mínima que la variable de control CV puede obtener. Si CV es menor que el valor mínimo ocurre lo siguiente: - CV será fijado en el valor ingresado, y - La alarma de salida será seteada (bit LL bajo límite).	El valor ingresado determinará cuando la alarma de salida de límite inferior será activado. Si CV cae por debajo de este mínimo valor, la alarma de salida (bit LL de límite bajo) será activada.
max	El valor ingresado será el porcentaje máximo de salida que la variable de control CV puede obtener. Si CV excede este máximo valor, ocurrirá lo siguiente: - CV será fijado en el valor ingresado, y - La alarma de salida (bit UL de límite superior será activada.	El valor ingresado determinará cuando la alarma de salida de límite superior será activada. Si CV excede este máximo valor, la alarma de salida (bit UL de límite superior) será activada.

C. BANDERAS DE LA INSTRUCCION PID

La columna de la derecha del display del APS (Fig. 3-12) muestra varias banderas asociadas con la instrucción PID. A continuación se describen aquellas banderas.

Bit TM. Modo del PID (palabra 0, bit 0):

Especifica el modo del PID. Este es activado (1) cuando el modo TIMED está en efecto y es desactivado (0) cuando el modo STI está en operación. Este bit puede ser cambiado con instrucciones para el efecto desde el programa de escalera.

Bit AM. Auto/Manual (palabra 0, bit 01):

La operación automática es especificada cuando es desactivado este bit, mientras que la operación manual es fijada cuando es activado el bit. Al igual que el bit anterior, esta opción puede ser cambiada desde el programa de escalera.

Bit CM. Modo de control (palabra 0, bit 02):

Es desactivado cuando el control es de la forma E=SP-PV y activado cuando es de la forma E=PV-SP. Este bit puede ser alterado desde el programa de escalera por el usuario.

Bit OL. Habilitación de limitación en la salida (palabra 0, bit 03):

Es activado cuando se ha seleccionado limitar la variable de control usando la tecla de función F4. Se puede alterar el estado de este bit desde el programa de escalera.

Bit de aumento del rango RG. (Solo para procesador 5/03. Palabra 0, bit 4):

Cuando está activado, este bit causa que el valor de la ganancia proporcional y del término integral (minutos por repetición) se incrementen en un factor de 10.

Por ejemplo: un valor integral de 1 indica que un valor integral de 0.01 minutos/repetición será aplicado al algoritmo PID. Un valor de ganancia de 1 indica que el error será multiplicado por 0.01 y aplicado a la ecuación del PID.
Cuando este bit es desactivado, permite al término integral y a la ganancia proporcional ser evaluados en las mismas unidades que el PID de un procesador 5/02 (multiplicados por 0.1). Es importante mencionar que un cambio de este bit no afecta al término derivativo en absoluto.

Bandera de escala del set point SC (palabra 0, bit 5):

Es desactivado cuando el valor de escalamiento del set point es especificado.

Tiempo de actualización del lazo muy rápido TF (palabra 0, bit 6):

Es activado por el algoritmo del PID si el tiempo de actualización del lazo especificado no puede ser alcanzado por el programa debido a limitaciones en el tiempo de barrido. En este caso, el problema se corrige actualizando el lazo del PID a una velocidad más baja o moviendo la instrucción PID a un rutina de interrupción STI. Las ganancias derivativa e integral estarán en error si la instrucción opera con este bit activado.

Bit DA. Acción Derivativa (Solo para procesador 5/03. Palabra 0, bit 7):

Cuando este bit está activo, causa que los cálculos derivativos sean evaluados sobre el error en lugar de PV. Cuando está desactivado, permite que los cálculos derivativos sean evaluados en la misma forma que en la instrucción PID del procesador 5/02 (donde la derivación es efectuada sobre PV).

Bit DB. Error en banda muerta (palabra 0, bit 8):

Es activado cuando la variable del proceso PV está dentro del rango de la banda muerta.

Alarma de salida.- Límite superior UL (palabra 0, bit 9):

Es activado cuando la salida de control CV calculada excede el límite superior de CV.

Alarma de salida.- Límite inferior LL (palabra 0, bit 10):

Es activado cuando la salida de control CV calculada es menor que el límite inferior de CV.

Set point fuera de rango SP (palabra 0, bit 11):

Se activa cuando el setpoint excede el máximo valor escalado o es menor que el mínimo valor escalado, es decir cuando el valor de referencia está fuera de rango.

Variable del proceso fuera de rango PV (palabra 0, bit 12):

Se activa cuando la variable del proceso no escalada excede 16383 o es menor que 0.

Bit DN. Ejecución de la instrucción PID (palabra 0, bit 13):

Es activado, durante el barrido, cuando el algoritmo PID es procesado. La instrucción PID es procesada a la velocidad de actualización del lazo.

PID habilitado EN (palabra 0, bit 15):

Es activado mientras la rung de la instrucción del PID es verdadera.

D. CARACTERISTICAS DEL BLOQUE DE CONTROL

El bloque de control tiene una longitud fija de 23 palabras y debe ser programado como un archivo entero. Las banderas de la instrucción PID (palabra 0) y los demás parámetros están localizados de la siguiente forma:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	word
EN		DN	PV	SP	LL	UL	DB	DA	TF	SC	RG	OL	CM	AM	TM	0
* Código de error del PID (Msbyte)																1
* Set point SP																2
* Ganancia K_c																3
* Término Ti																4
* Término Td																5
* Perturbación																6
* Set point max. (Smax)																7
* Set point min. (Smin)																8
* Banda muerta																9
USO INTERNO NO CAMBIAR																10
* Salida máxima																11
* Salida mínima																12
* Actualización del lazo																13

Variable del proceso escalada		14
Error escalado SE		15
Salida CV% (0-100%)		16
LSW Suma integral	5/03 MSW Suma integral	17
MSW Suma integral	5/03 LSW Suma integral	18
USO INTERNO NO CAMBIAR		19
		20
		21
		22

- Los bits 0/7 y 0/4 son aplicados únicamente al procesador 5/03
- Los bits OL, CM, AM, TM pueden ser alterados en el programa de escalera

E. CODIGOS DE ERROR

Cuando ocurre un error en el procesamiento de la instrucción PID se declara un código de error 0036 en el archivo de status del PLC. Este código abarca las condiciones de error que se detalla a continuación. Cada error tiene asignado un código único que aparece en la segunda palabra del bloque de control (Código de error del PID).

CODIGO 11H

5/02: Se ha ingresado un tiempo de actualización del lazo mayor a 255 o igual a 0.

5/03: Tiempo de actualización del lazo mayor a 1024 o igual a 0.

Corrección: Cambiar el tiempo de actualización del lazo: $0 < t \leq 255$.

CODIGO 12H

5/02: El valor de la ganancia proporcional es mayor a 255 o igual a 0.

5/03: Ganancia proporcional menor a 0.

Corrección: Cambiar la ganancia proporcional: $0 < Kc \leq 255$.

CODIGO 13H

5/02: El valor de la ganancia integral es mayor a 255.

5/03: Ganancia integral menor a 0.

Corrección: Cambiar la ganancia integral: $0 \leq Ti \leq 255$.

CODIGO 14H

5/02: El valor de la ganancia derivativa es mayor a 255.

5/03: Ganancia derivativa menor a 0.

Corrección: Cambiar la ganancia derivativa: $0 \leq Td \leq 255$.

CODIGO 21H

Solamente 5/02: Setpoint escalado máximo (Smax) mayor a 16383 o menor a -16383.

Corrección: Cambiar setpoint escalado máximo: $-16383 \leq Smax \leq 16383$.

CODIGO 22H

Solamente 5/02: Setpoint escalado mínimo (Smin) mayor a 16383 o menor a -16383.

Corrección: Cambiar setpoint escalado mínimo: $-16383 \leq Smin \leq Smax \leq 16383$.

CODIGO 23H

Setpoint escalado mínimo (Smin) mayor a setpoint escalado máximo (Smax).

Corrección: Cambiar setpoint escalado: $-16383 \leq Smin \leq Smax \leq 16383$.

Para 5/03: -32768 a +32767.

CODIGO 31H

Si se está escalando el setpoint y $Smin > SP > Smax$, o cuando sin escalamiento $0 > SP > 16383$, entonces durante la ejecución inicial del lazo PID, este error ocurre y el bit 11,

palabra 0 del bloque de control se activa. Sin embargo durante la siguiente ejecución del lazo PID si un setpoint ingresado es inválido, el PID se continua ejecutando usando el setpoint anterior y el bit 11 de la palabra 0 del bloque de control es activado.
Corrección: Cuando se use setpoint escalado verificar: Smin ≤ SP ≤ Smax. Cuando no se use escalamiento: 0 ≤ SP ≤ 16383.

CODIGO 41H

Escalamiento seleccionado: La banda muerta es menor que 0, mayor que Smax-Smin o mayor a 16383 (solo para 5/02).
Sin escalamiento: La banda muerta es menor que 0 o mayor que 16383.
Corrección: Con escalamiento, cambiar la banda muerta a: 0 ≤ banda muerta ≤ Smax-Smin ≤ 16383. Sin escalamiento: 0 ≤ banda muerta ≤ 16383.

CODIGO 51H

Límite superior de salida menor que 0 o mayor que 100.
Corrección: Cambiar el límite superior a: 0 ≤ límite superior de salida ≤ 100.

CODIGO 52H

Límite inferior de salida menor que 0 o mayor que 100.
Corrección: Cambiar el límite inferior a: 0 ≤ límite inferior de salida ≤ límite superior de salida ≤ 100.

CODIGO 60H (solo para 5/02)

El lazo PID está siendo ingresado por segunda ocasión: el lazo del PID fue interrumpido por una interrupción I/O la cual es a su vez interrumpida por la interrupción STI del PID.
Corrección: Modificar el programa.

F. *LA INSTRUCCION PID Y EL ESCALAMIENTO DE LAS ENTRADAS Y SALIDAS ANALOGICAS*

Para la instrucción PID, la escala numérica para la variable del proceso así como para la variable de control es 0 a 16383. Para usar unidades físicas (tales como PSI, grados, etc.) se debe primero escalar el rango de las entradas y salidas analógicas dentro de la escala numérica adecuada. Para hacer esto se puede usar la instrucción SCL y seguir los pasos descritos a continuación.

Para escalar tanto la variable de entrada como de salida se debe primero determinar el rango decimal en el que se encuentran dichas variables. Por ejemplo una entrada analógica de 4 a 20 mA tiene un rango decimal de 3277 a 16384. Luego, con estos valores se procede al escalamiento.

Una vez escalada la entrada y salida analógica se puede ingresar los valores máximo y mínimo en unidades físicas de la aplicación o proceso a controlar. Por ejemplo si una entrada analógica de 4-20 mA representa una presión de 0-300 PSI, los valores de 0 y 300 corresponden a los parámetros mínimo y máximo respectivamente. De esta manera, la variable del proceso, el error, el setpoint y la banda muerta serán mostrados en unidades físicas en la pantalla de monitoreo de datos del PID (programa APS).

La siguientes tablas muestran los parámetros de la instrucción de escalamiento SCL que deben ser ingresados de acuerdo al rango de las variables de entrada y salida analógicas.

Tabla 3-4: Parámetros de la instrucción SCL para una entrada analógica

PARAMETRO	4 a 20 mA	0 a 5V	0 a 10V
Rate/10000	12499	10000	5000
Offset	-4096	0	0

Tabla 3-5: Parámetros de la instrucción SCL para una salida analógica

PARAMETRO	4 a 20 mA	0 a 5V	0 a 10V
Rate/10000	15239	10000	19999
Offset	6242	0	0

G. NOTAS DE CONSIDERACION

- RANGO DE LAS ENTRADAS Y SALIDAS

Como se mencionó anteriormente, la variable del proceso posee un rango de trabajo de 0 a 16383. Si el valor de esta variable es menor que 0 (bit 15 activo), se fija un valor igual a cero y se activa la bandera de fuera de rango (bit 12 de la palabra 0 del bloque de control). Si la variable del proceso es mayor que el valor máximo (bit 14 activo), se fija un valor de 16383 en esta variable, activándose además el bit de fuera de rango.

- BANDA MUERTA

La banda muerta ajustable permite seleccionar una banda de error alrededor del punto de consigna o setpoint dentro de la cual la salida permanece invariable. Dentro de esta banda, el procesador del PLC considera que no existe error en el control. Ingresar un valor de cero, deshabilita esta característica. Posee las mismas unidades que se seleccionó para el setpoint.

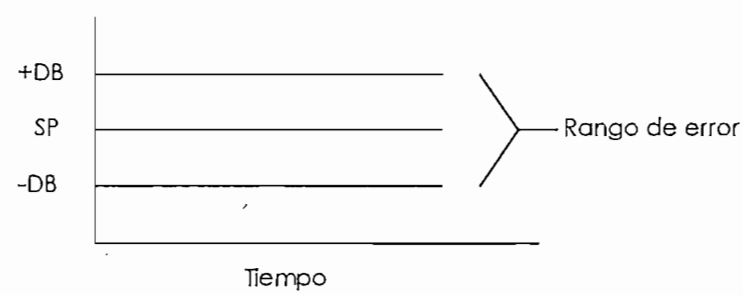


Fig. 3-13: Banda muerta.

- Estado de la rung del PID

Cuando la rung del PID es falsa, la variable controlada permanece en el último valor calculado.

3.3.2.1 DESCRIPCION DEL PROGRAMA DESARROLLADO

El programa implementado para el control PID en un PLC que dispone de esta rutina (anexo E, programa PID502.ACH) consta de dos partes: un programa principal y una subrutina de interrupción por tiempo (subrutina STI).

En el programa principal se realiza el movimiento de los datos de entrada y salida a la DATA FILE 9 para permitir la comunicación con la computadora.

En la subrutina STI, en su primera parte, se realiza una actualización del archivo de entradas analógicas, a continuación se procede al escalamiento de la entrada analógica al rango permitido por la instrucción PID. Posteriormente se ejecuta el algoritmo PID y se escala su resultado para finalmente actualizar el archivo de datos de salida.

3.3.2.2 CALIBRACION DEL PID

Si bien es cierto, existen muchos métodos de calibración de un controlador PID, el fabricante recomienda el siguiente procedimiento utilizado en aplicaciones no críticas:

1. Una vez creado el programa de escalera, asegúrese de tener adecuadamente escalada la entrada analógica dentro del rango permitido para la variable del proceso PV así como también la variable de control CV escalada en el rango permitido para la salida analógica.

2. Conecte el proceso a controlar al módulo de entradas y salidas analógicas y cargue el programa al procesador del PLC manteniéndolo en el modo de programación.

3. Ingrese los siguientes valores: Valor inicial del setpoint, ganancia integral igual a cero, ganancia derivativa igual a cero, ganancia proporcional igual a 1 y un factor de actualización del lazo igual a 5.
   Seleccione el modo de operación del algoritmo PID: modo STI o TIMED. Cuando se selecciona el modo STI, asegúrese de que el tiempo de actualización del lazo sea igual al intervalo de tiempo STI del status del PLC.
   Ingrese, de ser necesario, los valores de límite de salida, alarma de salida, escalamiento Smax y Smin y perturbación.

4. Coloque la instrucción PID en el modo MANUAL y cambie la operación del procesador al modo RUN.

5. Ajuste manualmente el proceso controlando el valor CO%.

6. Coloque la instrucción PID en el modo AUTO

7. Ajuste la ganancia proporcional mientras observa la relación entre la salida y el setpoint.
   Cuando se usa un procesador 5/02, el ajuste de ganancia puede interrumpir el proceso. Para evitar esto, cambie al modo MANUAL, realice el cambio de ganancia y regrese al modo AUTO. Cuando se usa un procesador 5/03, los cambios de ganancia no interrumpen el proceso y no es necesario pasar al modo MANUAL.

8. Cuando el proceso comience a comportarse de manera oscilatoria, mida el tiempo de un ciclo. Esto es, obtenga el periodo natural del proceso:
   periodo natural $\cong 4 \times$ tiempo muerto
   Regrese al modo MANUAL y pare el proceso.

1. Colocar los valores de Smin y Smax de la aplicación.
2. Escribir 50 en CO%.
3. Escribir 60 en CO% e inmediatamente arrancar un cronómetro.
4. Observar la variable del proceso PV. Cuando ésta empiece a cambiar, pare el cronómetro. Este tiempo conseguido es el tiempo muerto.
5. Multiplique el tiempo muerto por 4. Este valor es el periodo natural aproximado. Por ejemplo: si el tiempo muerto es 3 s, entonces el periodo natural es 3x4=12 s.
6. Divida el valor obtenido en el paso anterior por 10. Utilice este valor como el tiempo de actualización del lazo. En el ejemplo: 1.2 segundos. Por tanto un valor de 120 deberá ser ingresado en el tiempo de actualización del lazo.
7. Ingrese un valor de setpoint, la constante integral en 0, la constante derivativa en 0 y la proporcional en 1.
8. Retorne al paso 4 del procedimiento para calibración del PID.

3.3.3 PRINCIPALES BANDERAS DEL PLC UTILIZADAS.

En la realización del control PID se utilizaron algunas banderas tanto para el PLC sin rutina PID como para el PLC que si dispone de esta instrucción. A continuación se revisan brevemente dichas banderas.

S:4 Reloj (Solo lectura).

Esta bandera es encerada cuando el procesador pasa al modo RUN o TEST o cuando se lo enciende en el modo RUN y, es incrementada cada 10 ms.
Se puede utilizar cualquier bit de esta palabra individualmente en el programa del usuario como una base de tiempo precisa. La aplicación que use este bit debe ser evaluada a una velocidad dos veces más rápida. Por ejemplo, el bit S:4/3 cambia cada 80 ms dando como resultado un periodo de 160 ms. (Fig. 3-14). Para mantener

precisión en la aplicación, la instrucción que utilice este bit deberá ser evaluada al menos cada 79.999 ms.

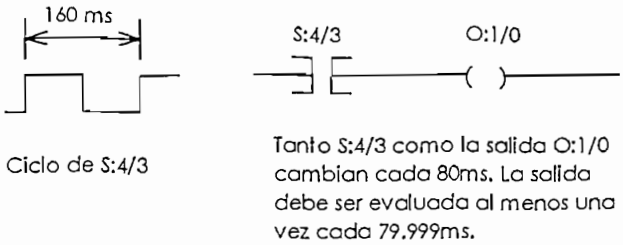


Fig. 3-14: Ejemplo del bit de reloj.

La siguiente tabla muestra todos los bits que pueden ser utilizados como base de tiempo dentro del programa del usuario.

Tabla 3-6: Bits de reloj disponibles en el PLC.

BIT	PERIODO
S:4/0	20 ms
S:4/1	40 ms
S:4/2	80 ms
S:4/3	160 ms
S:4/4	320 ms
S:4/5	640 ms
S:4/6	1280 ms
S:4/7	2560 ms

S:5 Bit de error menor.

Estos bits son activados por el procesador para indicar que un error menor ha ocurrido en el programa del usuario. Un error menor puede convertirse en un error mayor al finalizar el barrido del programa. Uno de estos errores es:

S:5/0 Overflow (Lectura/escritura).

Cuando este bit es activado por el procesador indica que un desbordamiento matemático ha ocurrido en el programa. Si este bit se encuentra activado hasta un instrucción END, ocurre un Error Mayor. Para evitar esto, se debe verificar el estado de este bit luego de cada instrucción matemática.

S:1/13 Error mayor (Lectura/escritura).

Este bit es activado por el procesador cada vez que un error mayor es encontrado. El procesador entonces entra en una condición de falla. La palabra S:6 contiene un código usado para diagnosticar la condición de falla. Cuando se desactiva este bit, el modo del procesador cambia de falla a programación, siendo necesario reingresar al modo RUN para comenzar la operación normal.

3.3.4 PRINCIPALES INSTRUCCIONES UTILIZADAS

A continuación se describe el funcionamiento de algunas de las principales instrucciones utilizadas en la implementación de un controlador PID en un PLC Allen-Bradley SLC-500 con y sin instrucción PID.

ADD: Suma.

La instrucción ADD añade un valor "source A" a otro valor "source B" y el resultado lo almacena en "destination". Los bits de estado aritmético cambian de la siguiente manera:

- Bit de Carry: el PLC lo activa (1) si se genera un carry, caso contrario lo desactiva (0).
- Bit de Overflow: lo activa si se genera un overflow en el resultado. En este caso se produce una activación de la bandera de "Error Menor (S:5)" y se coloca un valor de -32768 o +32767 en el resultado.
- Bit de Cero: activo si el resultado es cero.
- Bit de Signo: activo si el resultado es negativo.
- Registro matemático: no se altera.

SUB: Resta.

Se usa la instrucción SUB para restar un valor "source B" de otro valor "source A" y situar el resultado en la localidad "destination". Las banderas aritméticas cambian de la siguiente manera:

- Bit de Carry: activo si se genera un borrow o "llevo".
- Bit de Overflow: similar a la instrucción ADD.
- Bit de Cero: activo si el resultado es cero.
- Bit de Signo: activo si el resultado es negativo.
- Registro matemático: no cambia.

MUL: Multiplicación.

Esta instrucción multiplica un valor (source A) por otro valor (source B) y el resultado es almacenado en una localidad de destino.
Los bits o banderas de estado aritmético sufren alteraciones con esta instrucción de la siguiente manera:

- El procesador siempre borra la bandera de Carry (desactiva).

- Bit de Overflow: el procesador activa esta bandera si ocurrió un overflow en la localidad de destino, caso contrario es desactivada. Cuando existe overflow, la bandera de "Error menor" es activada y un valor de -32768 o +32767 es colocado en la localidad de destino.
- Bit Zero: el procesador activa esta bandera si el resultado de la instrucción es cero, caso contrario la desactiva.
- Bit de signo: el PLC activa esta bandera si el resultado de la multiplicación es negativo, caso contrario permanece desactivada.
- Registro matemático: Contiene un valor entero con signo de 32 bits resultado de la multiplicación. Este resultado es válido en overflow.

DIV: División

La instrucción DIV es utilizada para dividir un valor (source A) por otro (source B). El cociente redondeado de la división, es situado en la localidad de destino. Si el residuo es mayor o igual a 0.5, ocurre el redondeo del resultado. El cociente no redondeado es almacenado en la palabra más significativa del registro matemático, mientras que el residuo es colocado en la palabra menos significativa del mismo registro.
De la misma manera que con la instrucción anterior, los bits o banderas de estado aritmético son alterados de la siguiente forma:

- Bit de Carry: el procesador siempre lo desactiva.
- Bit de Overflow: el procesador lo activa si existe una división por cero o se ha detectado un overflow, caso contrario permanece desactivo. Igualmente en overflow, se activa la bandera de "Error menor" y se coloca un valor igual a 32767 en la localidad de destino.
- Bit de cero: el PLC lo activa si el resultado de la división es cero. En caso de que la bandera de overflow este activada, posee un valor no definido.
- Bit de signo: el PLC lo activa cuando el resultado de la división es negativo.

- Registro matemático: el cociente no redondeado es situado en la palabra más significativa, mientras que el residuo es situado en la palabra menos significativa.

DDV: Doble división

El contenido del registro matemático es dividido para el valor ingresado en esta instrucción y el resultado es situado en la localidad de destino. Si el residuo es mayor o igual a 0.5, ocurre un redondeo del resultado. El cociente no redondeado es situado en la palabra más significativa del registro matemático mientras que en la menos significativa se almacena el residuo.
Los bits de estado aritmético cambian de la siguiente manera:

- Bit de Carry: desactivado.
- Bit de Overflow: activo si la división es por cero o si el resultado es mayor a +32767 o menor a -32768. En esta condición, el bit de "Error menor" es activado y se coloca un valor de 32767 en el resultado.
- Bit de cero: activo si el resultado es cero.
- Bit de signo: activo si el resultado es negativo. No definido en condición de overflow.
- Registro matemático: inicialmente contiene el dividendo de la instrucción DDV. Una vez ejecutada la instrucción, el cociente no redondeado es colocado en la palabra más significativa y el residuo en la menos significativa.

OSR: One shot rising.

La instrucción OSR es una instrucción de entrada retentiva que posibilita que un evento cualquiera ocurra solamente una vez. Esta instrucción se usa cuando se requiera que un evento empiece basado en el cambio de estado de la rung de falso a verdadero.

IIM: Immediate Input with Mask.

Esta instrucción permite actualizar los datos antes del barrido normal de las entradas. Cuando la instrucción IIM es habilitada, el barrido del programa es interrumpido y los datos de una entrada específica son transferidos a través de una máscara al archivo de datos de entrada, permitiendo que estos datos estén disponibles para las instrucciones siguientes a la IIM en el programa.

IOM: Immediate Output with Mask.

Esta instrucción actualiza las salidas antes del barrido normal de las salidas. Cuando esta instrucción está habilitada, al igual que en las instrucción anterior, la exploración del programa es interrumpida para transferir los datos de salida a través de una máscara. Luego la exploración del programa continua.

SCL: Scale Data.

Cuando esta instrucción es verdadera, el valor "source" es multiplicado por el valor "rate". El valor resultante redondeado es sumado al valor "offset" y almacenado en la dirección de destino.
Cada vez que un overflow ocurra en la dirección de destino el bit de error menor S:5/0 deberá ser desactivado para prevenir que un error mayor sea declarado y ocasione una falla del procesador.

Estado de los bits aritméticos:

- Bit de Carry: no utilizado.
- Bit de Overflow: Lo activa cuando un overflow es detectado, caso contrario lo desactiva. En overflow, se activa también el bit de error menor y se almacena un

valor de 32767 o -32768 en la localidad de destino. La presencia de overflow es verificada antes y después de que el valor del "offset" sea añadido.

- Bit de Cero: El procesador lo activa cuando la localidad de destino es cero.
- Bit de Signo: El procesador lo activa cuando la localidad de destino tiene signo negativo.
- Registro matemático: no cambia.

4. SOFTWARE DESARROLLADO PARA LA ADQUISICION DE DATOS, SUPERVISION Y CONTROL

4. SOFTWARE DESARROLLADO PARA LA ADQUISICION DE DATOS, SUPERVISION Y CONTROL.

4.1 CARACTERISTICAS.

Un sistema de adquisición de datos y supervisión es utilizado en cualquier tipo de proceso industrial para proveer a los operadores de suficiente información y control de todo el sistema o de una parte de él, de una manera segura y económica.

Un sistema de supervisión de datos se lo podría definir como un control remoto de unidades localizadas a distancia que permite al operador disponer de información suficiente como para determinar el estado del proceso y facilitar las acciones a tomarse, sin que sea necesario estar presente en el lugar donde se originan los datos.

En un sistema de control supervisor, la computador supervisa el proceso controlado, de la misma manera en que el operador lo haría de manera manual. Con esta forma de control, por tanto, los lazos de control son mantenidos virtualmente intactos, añadiéndose únicamente elementos para permitir la supervisión.

El sistema consta de un controlador lógico programable, que controla directamente el proceso, y de una computadora central encargada de supervisar todos los parámetros procesados por el PLC. Básicamente las funciones que debe realizar la computadora principal son las siguientes:

- Adquisición de datos
- Supervisión de los mismos.
- Almacenamiento de la información adquirida.

CONFIGURACION DE LA COMPUTADORA PRINCIPAL

Para diseñar, desarrollar e implementar un sistema "SCADA" básico con PLC's es indispensable que la computadora principal disponga de ciertas características para el efecto, como son:

- El software 6001-F2E para lenguaje "C" y MSDOS 3.x o superior.
- Microsoft "C" 5.1 o Borland Turbo "C" 2.0; o versiones superiores.
- Compilación en el modo "memory large".
- Funciones de lectura-escritura incluidas en las librerías de aplicación del software 6001-F2E.
- La interface A-B 1784-KR deberá ser compatible con el PC.
- La interface requiere una interrupción H/W (2, 3, 4 ,5) y un slot de 8K en la memoria del PC._

Antes de ejecutar el programa ASC-PID.EXE es indispensable ejecutar previamente el programa START.BAT. Este archivo, inicializa los parámetros de comunicación que la tarjeta Allen-Bradley 1784-KR necesita para transferir datos desde y hacia el PLC.

Para un mejor rendimiento en lo que se refiere a la adquisición de datos, es recomendable utilizar un computador 386 o superior, de tal manera que el proceso de adquisición de datos y presentación de los mismos, se lleve a cabo de una manera más rápida.

Debido a que el software Allen-Bradley 6001-F2E no ha sido probado bajo ambiente Windows, se recomienda no utilizar el programa ASC-PID.EXE desde Windows.

4.2 SOFTWARE ALLEN-BRADLEY 6001-F2E

El software Allen-Bradley 6001-F2E usado en conjunto con la tarjeta 1784-KR de la misma casa, permite escribir programas de aplicación que se comunican directamente con la red DH-485, es decir, posibilitan la transferencia de información entre una computadora personal y los controladores lógicos programables u otros dispositivos situados en la red DH-485.

Se usa para el efecto, un conjunto estándar de funciones predefinidas en lenguaje "C", que admiten la comunicación con diversas estaciones en la red de trabajo DH-485.

Mediante estas funciones se puede definir paquetes propios de mensajes o utilizar paquetes de mensajes predefinidos que se incluyen en el software. Posteriormente se analizará con más detalle todos los tipos de funciones que se disponen.

Esta combinación de software y hardware permite usar los programas de aplicación para adquisición de datos desde las estaciones "token passing" tales como la familia de controladores programables SLC-500 y desde estaciones esclavas "non-token passing" tales como el 2755 DM6 Bar Code Reader.

El SLC-500, la red DH-485 y la combinación de la interface 1784-KR y el software 6001-F2E ofrecen un amplio y económico paquete de control y red. Estos productos constituyen una solución para aquellas aplicaciones que no requieran la capacidad de redes sofisticadas.

4.2.1 CARACTERISTICAS DEL SOFTWARE A-B 6001-F2E.

A continuación se menciona algunas características del software A-B 6001-F2E usado con la interface A-B 1784-KR:

- El software 6001-F2E posee un driver que posibilita la comunicación con la tarjeta A-B 1784-KR.

- El software 6001-F2E es usado solamente con una interface A-B 1784-KR. El uso de múltiples interfaces en el PC no es factible.

- El software 6001-F2E provee un conjunto de funciones enlazables en lenguaje "C", usadas en los programas de aplicación, para establecer la comunicación con varios dispositivos en la red DH-485.

- Se soporta compiladores de Microsoft "C" v5.1 y Borland Turbo "C" v2.0 o superiores.

- El 6001-F2E soporta solamente la versión "large memory" de Microsoft C y Borland Turbo C.

- El software 6001-F2E ha sido diseñado para uso con aplicaciones desarrolladas bajo sistema operativo DOS (v3.0 o superior).

- Programas que requieran comunicación interactiva entre el 6001-F2E y otras aplicaciones, tales como base de datos, no han sido probadas.

- Operaciones bajo ambiente Windows no han sido probados.

- El 6001-F2E no admite múltiples comandos de salida. Se debe recibir una señal de réplica antes de enviar otro comando.

- El driver 6001-F2E no retorna un código de error local si la tarjeta 1784-KR detecta una dirección igual a la suya en un nodo de la red. La estación que tiene la primera dirección permanece en línea mientras que la otra estación sale de línea. Es por eso importante chequear las direcciones duplicadas en la red DH-485.

- Si se envía desde el software mensajes con un formato incorrecto ocurre una condición de "timeout" en la aplicación. Puesto que no hay un código de error indicando un formato de mensaje incorrecto, se debe chequear el formato del mensaje primero.

- El 6001-F2E no soporta mensajes fuera del enlace (mensajes enviados a través de un puente a otra red).

- Se incluye librerías de aplicación para ejecutar los comandos más comunes de diagnóstico de lectura-escritura en la tabla de datos de los controladores programables de la familia Allen-Bradley SLC-500.

4.2.2 FUNCIONES DISPONIBLES

El software A-B 6001-F2E está constituido por funciones escritas en lenguaje "C" que permiten a una computadora personal comunicarse con cualquier nodo de la red DH-485. La siguiente tabla lista estas funciones.

Tabla 4-1: Funciones incluidas en el software.

Si se desea	Use la función
Ejecutar funciones de inicialización (se requiere antes de cualquier intento de comunicación, para que ésta se lleva a cabo).	Open_StdDrv
Transmitir datos sobre la red DH-485 a cualquier estación DH-485.	Send_StdDrv
Liberar todos los servicios de comunicación antes de que la aplicación termine.	Close_StdDrv

4.3 INSTALACION DEL SOFTWARE 6001-F2E

La instalación del software 6001-F2E se la realiza simplemente copiando los archivos que incluye el diskette de instalación en el directorio de trabajo.
Para una correcta instalación del software es indispensable una previa instalación adecuada de la tarjeta A-B 1784-KR.

ARCHIVOS DE LIBRERIA ESTANDAR

Para construir un programa de aplicación cualquiera, es necesario usar los siguientes archivos incluidos en el software A-B 6001-F2E.

Tabla 4-2: Archivos de librería estándar.

ARCHIVO	CONTENIDO
L_MSKR.LIB	Librería bajo el modelo "large memory" para enlazar programas escritos en Microsoft C v5.1 o superior.
L_TCKR.LIB	Librería bajo el modelo "large memory" para enlazar programas escritos en Borland C v2.01 o superior.
STDDRV.H KRDEFS.H	Archivos de encabezado que contienen definiciones y declaraciones requeridas para compilar las aplicaciones.
START485.EXE	Programa para inicializar el módulo 1784-KR. Es necesario ejecutarlo antes de las aplicaciones.

ARCHIVOS DE LIBRERIAS DE APLICACION.

Los siguientes archivos contienen librerías con funciones de aplicación que permiten usar rutinas predefinidas en el software.

Tabla 4-3: Librerías de aplicación.

ARCHIVO	CONTENIDO
L_MSAPKR.LIB	Módulo de aplicaciones compatible con el modelo "large memory" de Microsoft C v5.1 o superior.
L_TCAPKR.LIB	Módulo de aplicaciones compatible con el modelo "large memory" de Borland C v2.01 o superior.

4.4 PLANIFICACION DEL PROGRAMA DE APLICACION

Si se requiere crear un programa de aplicación específico es necesario conocer ciertos aspectos relacionados con la planificación de los mismos.
En esta sección se estudiarán los archivos de inclusión o encabezado necesarios para la compilación, todas las funciones que dispone el software y algunas consideraciones sobre la programación.

4.4.1 ARCHIVOS DE INCLUSION

Los archivos de inclusión o encabezado contienen declaraciones acerca del tipo de driver que se está usando. El software 6001-F2E usa los siguientes archivos de inclusión:

- KRDEFS.H y,
- STDDRV.H

Además de incluir los archivos de encabezado en la aplicación, es necesario enlazar el programa de aplicación con las librerías apropiadas.

Cuando el programa implementado requiera la utilización de múltiples archivos construidos en un "Project", se debe situar los archivos de inclusión KRDEFS.H y STDDRV.H en el archivo principal. Es importante mencionar que solamente un archivo fuente por programa ejecutable puede referir a cada archivo de inclusión. Cuando se requiera que archivos fuente adicionales, dentro de un programa ejecutable, se refieran a estos archivos, se debe hacer lo siguiente:

- Duplicar los archivos KRDEFS.H y STDDRV.H bajo diferentes nombres.

- Borrar las siguientes líneas del archivo duplicado KRDEFS.H:
  int max_umsg = Max_Umsg;
  int max_smsg = 16;

- Borrar del archivo STDDRV.H duplicado todas las líneas desde la primera ocurrencia de /*.....*/ bajo el encabezado del prototipo hasta el final del archivo.

Los nuevos archivos de inclusión así obtenidos son los que deberán ser utilizados en todos los restantes archivos fuente que hagan referencia al software 6001-F2E.

4.4.2 FUNCIONES DEFINIDAS

Las siguientes funciones ya definidas permiten al programa de aplicación comunicarse con los dispositivos en la red DH-485:

Open_StdDrv()
Inicializa el software de comunicación 6001-F2E. Se debe usar esta función cada vez que se necesite abrir la comunicación con el PLC.

Appl_StdDrv()

Esta función permite usar rutinas predefinidas en las librerías de aplicación en un programa creado por el usuario. En la Tabla 4-6 se especifican estas rutinas.

Send_StdDrv()

Permite dar formato a los comandos no previstos en la librería de aplicación. Esta función posibilita el envío de paquetes de mensajes definidos por el usuario a dispositivos "token-passing" o a dispositivos esclavos.

Get_StdDrv()

Recupera un string en ASCII que describe un mensaje de error en la red.

Close_StdDrv()

Termina la comunicación. Se debe usar esta función cada vez que se desee cerrar la comunicación con el PLC.

Cuando se desarrolle un programa de aplicación se deberá comenzar siempre con la función Open_StdDrv() para inicializar la comunicación y terminar con la función Close_StdDrv() para cerrar la misma. Una vez efectuada la comunicación, el programa podrá hacer uso de las funciones Appl_StdDrv(), Send_StdDrv() o una combinación de las dos, dependiendo de la aplicación.

4.4.3 CONSIDERACIONES SOBRE LA PROGRAMACION

Es necesario recordar las siguientes consideraciones cuando se escriba un programa de aplicación:

- Se deberá usar solamente el software Borland Turbo C (v.2.01 o superior) o Microsoft C (v.5.0 o superior) para la compilación y enlace de cualquier programa de aplicación.

- Si se utiliza las funciones Appl_StdDrv() o Send_StdDrv(), se debe siempre enlazar el programa de aplicación con las siguientes librerías:

Tabla 4-4: Librerías para enlazar un programa de aplicación.

SI SE USA	ENLAZAR CON
Borland Turbo C	L_TCKR.LIB
	L_TCAPKR.LIB
Microsoft C	L_MSKR.LIB
	L_MSAPKR.LIB

4.5 USO DE LAS FUNCIONES DEFINIDAS EN EL SOFTWARE

A continuación se estudia detalladamente la manera de utilizar cada una de las funciones definidas en el software A-B 6001-F2E. Se incluye el formato y los parámetros para cada una de las funciones.

4.5.1 FUNCION Open_StdDrv()

La función Open_StdDrv() permite inicializar la comunicación entre la computadora personal y el controlador lógico programable. Para esto se debe usar el siguiente formato y parámetros.

FORMATO PARA Open_StdDrv():

```
status = Open_StdDrv(device,0,0,0,(unsol_msg*)NULL,0,0,0);
```

PARAMETROS PARA Open_StdDrv():

En la siguiente tabla se resume los parámetros necesarios para ejecutar la función.

Tabla 4-5: Parámetros de la función Open_StdDrv().

PARAMETRO	TIPO	DESCRIPCION
device[]="KR:0"	char	Asigna el tipo de driver "KR:" y el canal de comunicación "0". El driver 6001-F2E soporta solamente un canal de comunicación con la 1784-KR.
(unsol_msg*)NULL	struct	Es un puntero nulo.
0	N/A	Este parámetro es ignorado pero es necesario incluirlo en la función. Se escribe un cero por cada parámetro ignorado.

Cuando la función Open_StdDrv() es llamada, retorna un valor que determina el estado de la operación e indica si ésta fue satisfactoria (valor 1) o no (valor diferente de 1). En el anexo B se incluye una lista de los códigos de error y su significado.

4.5.2 FUNCION Appl_StdDrv()

La función Appl_StdDrv() da formato a la información de las distintas rutinas que soporta (ver Tabla 4-6) y la transmite sobre la red DH-485. La función se usa con un símbolo que identifica la rutina seleccionada y con una estructura de datos correspondiente a la información a enviar. Los símbolos de las rutinas y la estructura de datos de la información a transmitir son definidas en el archivo de inclusión STDDRV.H

FORMATO PARA Appl_StdDrv():

status = Appl_StdDrv(SYMBOL, SD_FB*);

PARAMETRO	TIPO	DESCRIPCION
SYMBOL	int	Corresponde a un identificativo de la rutina que se pretende ejecutar (ver Tabla 4-6).
SD_FB	struct	Inicializa el bloque de datos.

PARAMETROS PARA Appl_StdDrv():

En la siguiente tabla se muestran las rutinas disponibles (en la librería de aplicación) que se pueden usar con Appl_StdDrv().

Tabla 4-6: Rutinas soportadas por el software 6001-F2E.

RUTINA	SIMBOLO
Prueba de comunicación.	PLCx_DLB
Lectura de los contadores de comunicación	PLCx_DRD
Lectura del estado de un dispositivo.	PLCx_DS
Borrar los contadores de comunicación.	PLCx_RC
Lectura en archivo de datos N9.	PLC2_URD
Escritura en archivo de datos N9.	PLC2_UWR
Escritura en archivo tipo BIT (B3).	PLC2_UBW

En la sección 4.7.4 (comandos básicos) se incluye información sobre las rutinas soportadas por la función Appl_StdDrv().

A continuación se detalla la estructura de datos correspondiente al bloque de información transmitido.

DHP_MSG.dev=device; (tipo char)

La variable "device" indica al driver la interface de comunicación y el canal. Ponga esta variable en "KR:0".

DHP_MSG.stat=&io_stat[0]; (tipo unsigned int)

La variable io_stat sirve para dos propósitos. Cuando las rutinas de la librería de aplicación son llamadas, retornan el status antes de que cualquier tipo de réplica sea recibida desde el dispositivo remoto.
Cuando la función Appl_StdDrv() satisfactoriamente inicia el envío de cierta información requerida (status=1), la variable io_stat[0] es puesto en 0. Cuando un mensaje de réplica es recibido o una réplica de "timeout" ocurre, io_stat[0] es puesto en un valor más grande que 0 (normalmente 1). Si io_stat[0] es normal, io_stat[1] contendrá la longitud del buffer de datos de réplica.

Si io_stat no es igual a 1, quiere decir que un error ha ocurrido. El formato es como sigue:
El byte menos significativo (EXT STS) de io_stat[0] contiene los errores locales, tales como el "timeout". El byte más significativo (STS) de io_stat[0] contiene los errores de la red DH-485. Si el byte mas significativo de io_stat[0] es igual a F0H (indicando el estado DH-485 extendido), el byte menos significativo de io_stat[0] contendrá un código que indica el estado extendido de la red DH-485. En la Tabla 4-8 se indica la manera de leer la variable io_stat[0].

DHP_MSG.L_R=Loc_Rem; (tipo int)

Poner esta variable en 0.

DHP_MSG.dst=&destination; (tipo unsigned char)

Dirección de destino DH-485. La variable "destination" es la dirección DH-485 de la red donde se desea que el mensaje sea enviado.

DHP_MSG.dta=&dt_addr; (tipo unsigned int)

Dirección de la tabla de datos. Dependiendo de la rutina, la variable dt_addr es un valor de 2 bytes o un string, que describe la dirección de la tabla de datos donde éstos van a ser leídos o escritos.

DHP_MSG.len=size; (tipo int)

Cuando los datos son leídos, la variable *size* indica cuantos bytes de datos van a ser leídos. Cuando los datos son escritos, la variable *size* indica cuantos bytes deberían ser copiados del buffer de datos y escritos en la estación remota.

DHP_MSG.buf=&d_buff[0]; (tipo unsigned char o int)

Buffer de datos de aplicación. Los datos de aplicación pueden tomar muchas formas. Si se está usando valores en bytes, d_buff puede ser definido como un arreglo tipo char, la variable "size" es definida entonces como una relación uno a uno. Si se está usando enteros con signo, los datos son automáticamente guardados en bytes con formato intercambiado, el valor de la variable "size" está definido por una relación dos a uno.

DHP_MSG.TO=timeout; (tipo unsigned int)

La variable timeout es el número de segundos que se desea esperar por una réplica. Transcurrido este tiempo, se declara un error por pérdida de la comunicación.

Cuando se establece comunicación con un dispositivo SLC-500, usando los comandos de lectura y escritura en el archivo de datos N:9, es necesario crear de antemano, un archivo de datos N:9 en el SLC-500. El PLC usa este archivo de datos para la comunicación en la red DH-485. La dirección de la tabla de datos (definida como dt_addr en la función Appl_StdDrv()) será interpretada por el SLC-500 como un offset (en palabras) en el archivo de datos N:9.

4.5.3 FUNCION Send_StdDrv()

Si no se desea utilizar los comandos incluidos en la librería de aplicación, se usa la función Send_StdDrv() para enviar mensajes formateados por el usuario. La función Send_StdDrv() transmite datos sobre la red DH-485 a una estación cualquiera DH-485. Dependiendo de la aplicación se puede utilizar esta función de dos maneras.

- Comunicación a dispositivos token-passing DH-485 tales como los controladores de la familia SLC-500.
- Comunicación con dispositivos slave-only DH-485.

Cualquier transmisión que no se complete normalmente es abortada por un "timeout" o por un código de error local indicando el problema.

FORMATO PARA Send_StdDrv()

```
status = Send_StdDrv( device, &io_stat[0], &cmd_buff[0],
                      pass_thru, &reply_buff[0], timeout, 0, 0 )
```

PARAMETROS PARA Send_StdDrv().

Los parámetros se detallan a continuación.

device[0]="KR:0"; (tipo char)

La variable "device" deberá coincidir con la variable "device" usada en la función Open_StdDrv(), es decir, "KR:0".

Io_stat[2]; (tipo unsigned int)

Tiene características similares a la variable usada en la función Appl_StdDrv().

Cmd_buff[...]; (tipo unsigned char)

El parámetro cmd_buff es un buffer que contiene los mensajes a ser enviados a la estación remota. Se usa el siguiente formato:

LEN TYP DST SRC CMD STS TNS TNS DATA

El campo LEN contiene la longitud del paquete entero, incluyendo LEN. El campo TYP es el tipo de mensaje. Se debe poner este valor en 0 o 5 dependiendo del tipo de mensaje. El campo DST es el destino DH-485 donde los mensajes serán enviados. El campo SRC es la dirección DH-485 de la tarjeta 1784-KR. Este campo puede ser puesto en 0.

Pass_thru=1; pass_thru=0; (tipo int)

El parámetro pass_thru es una opción de los mensajes de réplica. Cuando pass_thru es puesto en 1, todos los mensajes de réplica son puestos en el buffer. Cuando el parámetro pass_thru es puesto en 0, solamente los datos requeridos (no todos) son

puestos en el buffer. La longitud de los mensajes de réplica es retornada en la variable io_stat[1].

Reply_buff[...]; (tipo unsigned char)

El parámetro reply_buff dice al programa donde pone el mensaje de réplica en la aplicación. El mensaje de réplica es copiado en el buffer usando el siguiente formato:

LEN TYP DST SCR CMD STS TNS TNS DATA

timeout=5; (tipo unsigned int)

El parámetro timeout es el número de segundos que la aplicación espera por un mensaje de réplica.

0; (sin tipo N/A)

Los parámetros 7 y 8 no son usados, pero es necesario incluirlos en la función. Se debe escribir un cero en ambos parámetros.

En el anexo B se muestran los posibles valores retornados por esta función y su significado.

4.5.4 FUNCION Get_ErrMsg()

La función Get_ErrMsg() suministra un string con mensajes de error en la transmisión de datos. Si un error ocurre, se debe llamar a la función Get_ErrMsg() con el código de error para conocer el mensaje indicativo del problema.

FORMATO PARA Get_ErrMsg()

Get_ErrMsg(err, ret_msg);

PARAMETROS PARA Get_ErrMsg():

La siguiente tabla muestra dichos parámetros.

Tabla 4-7: Parámetros de la función Get_ErrMsg().

PARAMETRO	TIPO	DESCRIPCION
err	unsigned int	Es una copia del valor del código de error retornado en io_stat[0].
ret_msg	char	Es un puntero a un buffer tipo char de al menos 80 caracteres de longitud. Contiene el mensaje de error que corresponde al código del parámetro err. El mensaje de error es retornado en este buffer.

La siguiente tabla explica como leer el parámetro io_stat[0] para tomar los bytes STS y EXT STS.

Tabla 4-8: Lectura de io_stat[0].

Si los bytes STS y EXT STS de io_stat[0] son:	DESCRIPCION
STS = 0 EXT STS = 0	No hay error.
STS = F0 EXT STS = X	Existe un mensaje de réplica con información en el status extendido. El código en EXT STS indica el mensaje (ver anexo B).
STS = X EXT STS = Y	Mensaje de réplica: X = contiene los errores remotos retornados por dispositivos en la red DH-485, Y = contiene un código de error local (ver anexo B).

4.5.5 FUNCION Close_StdDrv()

La función Close_StdDrv() libera todo tipo de enlace y comunicación, debe ser usada cuando una aplicación haya terminado.

FORMATO PARA Close_StdDrv()

status = Close_StdDrv(device)

PARAMETROS PARA Close_StdDrv()

La siguiente tabla muestra los parámetros.

Tabla 4-9: Asignación de parámetros para finalizar la comunicación.

PARAMETRO	TIPO	DESCRIPCION
device[]="KR:0"	char	El parámetro device deberá coincidir con el usado en la función Open_StdDrv().

Cuando la función Close_StdDrv() es utilizada, retorna un valor que indica si la operación fue satisfactoria o no. Un valor distinto de 1 indica que un error ha ocurrido (en el anexo B se listan los códigos de error y su significado).

4.5.6 EVITANDO REPLICAS DE MENSAJES PERDIDOS

Para prevenir réplicas de mensajes perdidos, al utilizar la función Send_StdDrv(), se prevé la función Get_tns(). Usa el siguiente formato:

x = Get_tns()

Esta función retorna un valor del tipo entero sin signo (unsigned int). Se debe situar dicho valor en los dos bytes del campo TNS antes de llamar a la función Send_StdDrv().

4.6 COMPILANDO, ENLAZANDO Y CONFIGURANDO UN PROGRAMA DE APLICACION.

En esta sección se indica los pasos a seguir para que un programa de aplicación pueda comunicarse con otros dispositivos en la red. Se debe compilar, enlazar y configurar el programa de aplicación previamente.

4.6.1 COMPILANDO Y ENLAZANDO EL PROGRAMA DE APLICACION.

Para compilar y enlazar el programa de aplicación, se recomienda:

1. Crear el programa de aplicación e incluir los archivos de encabezado KRDEFS.H y STDDRV.H en el directorio de trabajo.

2. Compilar el programa de aplicación usando la opción "large memory" del compilador Microsoft C o Borland Turbo C.

3. Enlazar el programa con el archivo de librería adecuado (incluido con el software).

4. Copiar el programa .EXE generado y el archivo START485.EXE en el directorio de trabajo.

5. Si los micro interruptores de la tarjeta 1784-KR han sido cambiados (no son los de defecto), es necesario ingresar un comando en el DOS para configurarlo adecuadamente (sección 2.4.4).

4.6.2 CONFIGURANDO EL PROGRAMA DE APLICACION

Para configurar el programa es necesario escribir START485 en el prompt del DOS, seguido por las opciones de comunicación deseadas. Por ejemplo:

START485 a0 m31 cM b9600

donde:

Tabla 4-10: Parámetros para configuración del programa.

VARIABLE	DESCRIPCION
a	Corresponde a la dirección DH-485 de la tarjeta 1784-KR. Rango 0-31.
m	Es la dirección del nodo máximo en la red DH-485. Rango 0-31
c	Corresponde a la categoría de operación. Escribir siempre "M".
b	Es el baud rate de la comunicación en la red DH-485. Valores válidos: 300, 1200, 2400, 4800, 9600, 19200 baudios.

Esta operación inicializa la tarjeta 1784-KR. Ahora la interface y el driver están listos para ser utilizados.

4.7 ESPECIFICACION DEL MENSAJE CON LA FUNCION Send_StdDrv().

El formato del paquete de información depende si se envía datos a:

- Dispositivos token-passing (tales como el SLC-500).
- Dispositivos slave-only DH-485.

Esta sección contiene el formato del paquete de información necesario y una explicación de cada campo del paquete. Además contiene los comandos soportados en dicha función.

Los cuadros de las siguientes páginas muestran todos los posibles campos en la función Send_StdDrv(). Los bloques no sombreados indican campos que **siempre están incluidos** en el paquete de información. Los bloques sombreados indican campos que pueden estar incluidos en el paquete dependiendo del comando particular.

4.7.1 COMUNICACION CON DISPOSITIVOS TOKEN-PASSING.

La Fig. 4-1 muestra el formato de la información a ser enviada para comunicación con dispositivos token-passing, tales como los procesadores SLC-500.

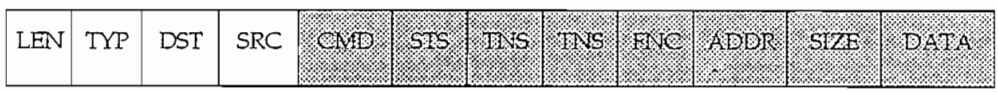


Fig. 4-1: Formato del paquete de información para comunicación con procesadores SLC-500.

La siguiente tabla explica el contenido de los campos anteriores.

Tabla 4-11: Comunicación con procesadores SLC-500.

CAMPO	DESCRIPCION
LEN	Longitud del paquete de mensajes (incluyendo LEN).
TYP	Código que identifica el formato del paquete de información (Es siempre 0).
DST	Dirección de destino DH-485 de la información.
SRC	Dirección del dispositivo que envía la información.
CMD	Byte de comando.

CAMPO	DESCRIPCION
STS	Estado de la información.
TNS	Código de comunicación (2 bytes).
FNC	Byte usado con CMD. Define una función específica.
ADDR	Campo de 2 bytes que contiene la dirección de la localización de memoria donde el comando va a comenzar la ejecución.
SIZE	Especifica el número de bytes de datos a ser transferidos por un mensaje.
DATA	Datos enviados.

4.7.2 COMUNICACION CON DISPOSITIVOS SLAVE-ONLY.

La Fig. 4-2 muestra el formato de la información para comunicación con dispositivos slave-only en la red DH-485.

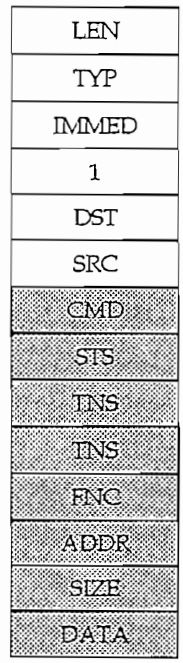


Fig. 4-2: Paquete de información para comunicación con dispositivos SLAVE-ONLY.

La Tabla 4-12 describe los parámetros del paquete de información que se requieren para la comunicación con dispositivos slave-only.

Tabla 4-12: Comunicación con dispositivos Slave-only.

CAMPO	DESCRIPCION
LEN	Longitud del paquete de mensajes (incluyendo LEN).
TYP	Código que identifica el formato del paquete de información. Para dispositivos slave-only es siempre 5.
IMMEDIATE BLOCK	Bloque del dispositivo de destino.
1	Constante.
DST	Dirección de destino DH-485 de la información.
SRC	Dirección del dispositivo que envía la información.
CMD	Byte de comando.
STS	Estado de la información.
TNS	Código de comunicación (2 bytes).
FNC	Byte usado con CMD. Define una función específica.
ADDR	Campo de 2 bytes que contiene la dirección de la localización de memoria donde el comando va a comenzar la ejecución.
SIZE	Especifica el número de bytes de datos a ser transferidos por un mensaje.
DATA	Datos enviados.

En la siguiente sección se describen los campos de ambas aplicaciones.

4.7.3 CAMPOS DEL PAQUETE DE INFORMACION

A continuación se describen los campos del paquete de información con más detalle.

LEN

Este campo identifica la longitud del paquete de información (incluyendo LEN). El rango permitido es de 0 a 255.

TYP

Es el código que identifica el tipo de comunicación. Se usa 0 para comunicación a procesadores SLC-500 y 5 para comunicación a dispositivos slave-only.

IMMEDIATE BLOCK

El rango permitido es de 0 128-255 dependiendo del dispositivo.

DST y SRC

El byte DST (destino) contiene el número del nodo DH-485 al que va a llegar el mensaje o paquete de información. El byte SRC (origen) contiene el número del nodo DH-485 que origina el mensaje. Valores permitidos para estos bytes son de 0 a 31 (decimal).

CMD y FNC

Juntos los bytes CMD (comando) y el byte FNC (función) definen la actividad a ser ejecutada por el comando en el nodo destinatario. CMD define el tipo de comando y FNC (cuando es usado) define la función específica en el tipo de comando.

STS y EXT STS

Los bytes STS (status) y EXT STS (status extendido) indican el estado de la transmisión de la información. Al enviar un comando, el programa de aplicación deberá poner el valor de STS en cero. En mensajes de réplica, el byte STS puede contener uno de los códigos de estado listados en el anexo B. Si el byte más significativo es igual a F0 (hexadecimal), hay entonces información adicional en el byte EXT STS. Si no es igual a F0H, el byte EXT STS no es usado. Un valor de 0 en STS, en un mensaje de réplica, significa que el comando ha sido ejecutado sin errores.

TNS

El byte TNS (2 bytes) contiene un único identificador de comunicación de 16 bits. Por cada mensaje transmitido por el nodo, el software debe asignar un número único de comunicación de 16 bits. Una simple forma de generar este número es usar la función Get_tns() y almacenar el valor en los dos bytes del nuevo mensaje. Cuando el comando iniciador de un mensaje recibe una réplica de uno de sus comandos, éste puede usar el valor TNS para asociar el mensaje de réplica con su comando correspondiente. Si el valor TNS del mensaje de réplica se iguala el valor TNS del comando, entonces la réplica es apropiada para el comando.
En cualquier instante, la combinación de SRC, CMD y TNS son suficientes para identificar todo el paquete de información transmitido. Al menos uno de estos tres campos en el mensaje o paquete de información actual deberá ser diferente que el campo correspondiente al último mensaje recibido por el comando ejecutor. Si ninguno de estos tres campos es diferente, el mensaje es ignorado. Este proceso se denomina *"detección de mensajes duplicados"*.

ADDR

El campo ADDR (2 bytes) contiene la dirección de una localidad de memoria en donde el comando ejecutor va a iniciar su trabajo. Por ejemplo si el comando ejecutor es una lectura

de datos, ADDR especifica la dirección del primer byte de datos a ser leídos. El primer byte del campo ADDR contiene el byte menos significativo de la dirección y el segundo byte contiene el byte más significativo de la dirección.

SIZE

El byte SIZE especifica el número de bytes de datos a ser transferidos por el comando. Si este campo aparece en un comando de lectura, especifica el número de bytes de datos que el nodo de destino debe retornar en su mensaje de réplica. El valor permitido para SIZE varía con el tipo de comando.

DATA

El campo DATA contiene los datos binarios del programa de aplicación. El número de bytes de datos en el mensaje depende del comando o función que se ejecute.

4.7.4 COMANDOS SOPORTADOS

Los comandos soportados pueden generalmente ser ejecutados por cualquier procesador SLC. Estos comandos son usados exclusivamente con la función Send_StdDrv().

El paquete de información para cada comando es descrito en más detalle en las siguientes páginas. Los valores de CMD y FNC están expresados en notación hexadecimal, los otros valores corresponden a notación decimal.

COMANDO	CMD(hex)	FNC(hex)
SLC500		
Lectura lógica protegida con 3 direcciones de campo.	0F	A2
Escritura lógica protegida con 3 direcciones de campo.	0F	AA
BASICOS		
Borrar los contadores de comunicación.	06	07
Diagnóstico de la comunicación.	06	00
Lectura del contador de comunicación.	06	01
Lectura no protegida.	01	N/A
Escritura no protegida.	08	N/A
Lectura del estado del dispositivo.	06	03

4.7.4.1 BORRAR LOS CONTADORES DE COMUNICACION

Este comando pone en cero todos los contadores de comunicación (número de mensajes transmitidos, recibidos, etc.) en el nodo que se especifique. El comando de lectura de estado del dispositivo proporciona la dirección de inicio de estos contadores.

FORMATO DE ENVIO

CMD 6	STS	TNS	FNC 07

FORMATO DE RECEPCION

CMD 46	STS	TNS

4.7.4.2 DIAGNOSTICO DE LA COMUNICACION

Se puede usar este comando para chequear la integridad de los datos transmitidos en la red. El comando transmite información hasta de 243 bytes. El módulo de recepción deberá enviar o responder a este comando transmitiendo los mismos datos de regreso al nodo original de la transmisión.

FORMATO DE ENVIO

CMD 06	STS	TNS	FNC 00	DATOS 243 bytes max.

FORMATO DE RECEPCION

CMD 46	STS	TNS	DATOS 243 bytes max.

4.7.4.3 LECTURA DEL CONTADOR DE COMUNICACION

Este comando lee hasta 244 bytes de datos de la memoria PROM o RAM de un PLC. Se usa este comando para leer los contadores de comunicación. Cuando se ejecute este comando con procesadores SLC-500, se debe poner el parámetro ADDR en 0 y SIZE en 10 (decimal). En el anexo C se muestran los mensajes de réplica de este comando.

FORMATO DE ENVIO

CMD 06	STS	TNS	FNC 01	ADDR	SIZE

FORMATO DE RECEPCION

CMD 46	STS	TNS	DATA 244 bytes max.

4.7.4.4 LECTURA DEL ESTADO DE UN DISPOSITIVO

La computadora usa este comando para leer un bloque de información del estado de un dispositivo en la red DH-485. La información del estado varía con el tipo de dispositivo. En el anexo C se muestran los mensajes de réplica de varios dispositivos DH-485.

FORMATO DE ENVIO

CMD 06	STS	TNS	FNC 03

FORMATO DE RECEPCION

CMD 46	STS	TNS	DATA 244 bytes max.

4.7.4.5 LECTURA NO PROTEGIDA

Este comando lee palabras de datos de cualquier área de la tabla de datos N:9 del PLC. El parámetro SIZE es utilizado para especificar el número de bytes a ser leídos. Para especificar el número de palabras PLC, SIZE deberá ser un valor par, debido a que las palabras del PLC son de 2 bytes de longitud. La dirección de una palabra deberá ser par.

FORMATO DE ENVIO

CMD 01	STS	TNS	ADDR	SIZE

FORMATO DE RECEPCION

CMD 41	STS	TNS	DATA 244 bytes max.

4.7.4.6 ESCRITURA NO PROTEGIDA

Este comando escribe datos en cualquier área de la tabla de datos N:9 del PLC.

FORMATO DE ENVIO

CMD 08	STS	TNS	ADDR	DATA 244 bytes max.

FORMATO DE RECEPCION

CMD 48	STS	TNS

Es importante que al realizar la comunicación con dispositivo SLC-500 usando los modos de lectura y escritura no protegida, se cree de antemano un archivo de datos 9, tipo entero (N9) en el procesador. El PLC usa este archivo para la comunicación en la red DH-485.

4.7.4.7 LECTURA LOGICA PROTEGIDA

La lectura lógica protegida forma parte del conjunto de comandos permitidos para la función Send_StdDrv(). Esta función posee en su estructura tres direcciones correspondientes a los campos: archivo, elemento y sub elemento.

FORMATO DE ENVIO

	CMD 0F	STS	TNS	FNC A2	BYTE SIZE	FILE NUMBER	FILE TYPE	ELEMENT NUMBER	SUB-ELEM NUMBER
Byte No.	1	2	3,4	5	6	7	8	9	10

A continuación se describe cada campo.

BYTE CMD

Siempre es igual a 0F.

BYTE STS

Indica el estado del mensaje en la transmisión.

BYTE TNS

Es un número único de comunicación de 16 bits. Se usa este campo para igualar un mensaje de réplica a su correspondiente comando, el cual contiene el mismo número.

BYTE FNC

Es igual a A2.

BYTE SIZE

El tamaño de los datos a ser leídos (en bytes), no incluye el campo de dirección u otros bytes de encabezado.

FILE NUMBER

(Bytes 7a-7c). Este byte depende del parámetro FILE TYPE. Direcciona solamente archivos de 0 a 254. Para direccionar más allá de 254, el byte 7a debe igualarse a FF. Esto expande al FILE NUMBER a tres bytes en total. Se usa los bytes 7b y 7c para direccionamiento superior.

FILE TYPE

Es un número que corresponde a un archivo de datos del PLC.

80H - 81H	Reservado
82H	Salida
83 H	Entrada
84H	Status
85H	Bit
86H	Timer
87H	Counter
88H	Control
89H	Enteros (integer)

ELEMENT NUMBER

(Bytes 9a-9c). El byte 9 direcciona elementos de 0 a 254. Para direcciones más altas, se debe poner el byte 9a=FF, esto expande a ELEMENT NUMBER a tres bytes en total. Los bytes 9b y 9c son usados para acceder direcciones superiores.

DATA

Contiene los datos a ser escritos por el comando en la dirección especificada por los bytes anteriores. Primero constan los bytes menos significativos.

FORMATO DE RECEPCION

CMD 4F	STS	TNS	EXT STS solo en error

4.8 PROGRAMA IMPLEMENTADO.

El programa de adquisición de datos, supervisión y control PID (ASC-PID.EXE) fue desarrollado en su totalidad bajo Borland C++ v3.1 para DOS. El sistema de menús fue realizado con la ayuda de los utilitarios de Turbo Visión que incluye este paquete de programación.

El programa fue construido como un "Projetc". Contiene los siguientes módulos:

- FUNC-1.C: Archivo que contiene el código fuente de las opciones de escritura y lectura no protegida (lectura y escritura en el archivo de datos N9).
- FUNC-2.C: Contiene las funciones necesarias para ejecutar las operaciones de lectura y escritura lógica (lectura y escritura en los restantes tipos de archivos de datos).
- FUNC-3.C: Contiene las funciones que establecen la comunicación con el PLC (tanto para lectura y escritura protegidas como lógicas), éstas son, apertura de comunicación, inicialización de la comunicación, ejecución de la comunicación y finalización de la misma. Mediante estas funciones se realizó el enlace con las

subrutinas disponibles en el programa de control del prototipo de nivel de líquidos[1].

- PID.C: Incluye las funciones de graficación de la respuesta en el tiempo de los datos adquiridos del PLC. Constituye la "pantalla" principal del programa.
- AYUDA1.CPP: Incluye las subrutinas de comentarios para ayuda al usuario.
- ADSC-2.CPP y ADSC-1.CPP: Contienen los códigos de algunas de las opciones del menú principal.
- MEN-1.CPP: Contiene el código fuente del sistema de menús utilizado.
- GRAPHAPP.CPP: Código fuente que incluye funciones de graficación (utilitario del Turbo Vision).
- CALC.CPP: Código fuente de la opción "Calculadora" del menú principal (utilitario del Turbo Vision).
- TAN.CPP: Funciones del prototipo de nivel de líquidos.
- FUNC.CPP: Funciones gráficas utilizadas en el archivo TAN.CPP.

Además de los módulos antes mencionados, fue necesario la creación y/o utilización de algunos archivos de encabezado (header files) que contienen definiciones útiles para la compilación de los módulos anteriores:

- CALC.H: Definiciones relacionadas con la opción CALCULADORA del menú principal (Turbo Vision).
- CONT.H y CONT-1.H: Definiciones relativas a las distintas funciones utilizadas.
- GRAPHAPP.H: Definiciones relacionadas con las subrutinas gráficas (Turbo Vision).
- DATOS.H: Define la estructura para transferencia de información entre funciones.
- KRDEFS.H y STDDRV.H: Definiciones relativas al software A-B 6001-F2E.
- KRDEFS-1.H y STDDRV-1.H: Archivos KRDEFS.H y STDDRV.H modificados que posibilitan la construcción del programa en forma modular.

---

[1] Benítez Diego, Diseño y Construcción de un prototipo de control de líquidos, tesis de grado, EPN-FIE, 1994.

El programa principal básicamente está constituido de tres partes:

a) La primera encargada de leer el archivo de configuración creado en la sesión de trabajo anterior. Esto permite inicializar ciertos valores de algunas variables como por ejemplo los niveles prefijados de alarma, el nodo utilizado para la comunicación, etc.
b) La segunda, que es propiamente el programa de adquisición de datos, supervisión y control, y
c) La tercera que guarda en el archivo de configuración, los nuevos valores de ciertos parámetros ingresados en la sesión de trabajo.

El archivo de configuración se denomina CONFIG.PID. Posee valores útiles para la inicialización de algunas variables utilizadas en la ejecución del programa tales como: el tiempo de visualización en pantalla de los datos adquiridos desde el controlador lógico programable, el número de nodo (PLC) que se está supervisando, los niveles superior e inferior de alarma y el estado de la alarma (activada o desactivada), la escala de visualización en el eje Y.

El programa de adquisición de datos, supervisión y control PID se presenta en forma de menús para facilitar su utilización. Los ítems de estos menús pueden ser ejecutados de tres maneras distintas:

- Utilizando el mouse: Se debe posicionar con el mouse en la opción requerida y oprimir el botón izquierdo del mismo.
- Utilizando la tecla de función F10 para acceder al menú y navegar con las teclas del cursor hasta la opción requerida. Una vez seleccionada esta opción presionar la tecla "ENTER".
- Utilizando la combinación <Alt+tecla>, donde "tecla" corresponde a la letra resaltada del ítem que se desee ejecutar.

Es posible también utilizar una combinación de los métodos de acceso anteriores para ejecutar una opción requerida.

Existen además teclas rápidas que permiten ejecutar las distintas opciones que se disponen en el programa. Estas son:

- F1: Accede a la subrutina que visualiza la variación tanto de la señal de salida como la de control cuando se ejecuta un control PID en un PLC que dispone o no de esta instrucción (programas PID_CONT.ACH, PID_DISC.ACH o PID502.ACH).
- F3: Permite activar (modo automático) o desactivar (modo manual) el control PID. En modo automático, el PLC, en base a sus cálculos fija el valor de la señal de control. En modo manual, el usuario, mediante las teclas de cursor (↑ y ↓) determinan el valor de la señal de control.
- F5: Cuando se realice el control en el prototipo de nivel de líquidos, esta opción visualiza el tanque y la variación de su nivel.
- F6: Permite observar el nivel en el tanque y la respuesta en el tiempo a la vez.
- F8: Activa la alarma en los niveles superior e inferior prefijados.
- F9: Desactiva la alarma.
- Ctrl-F6: Opción que permite cambiar el valor máximo de visualización del eje Y.
- Ctrl-F7: Cambia el valor mínimo de visualización del eje Y.
- Ctrl-F8: Presenta una caja de diálogo que posibilita cambiar el tiempo de visualización de las señales.
- Alt-H: Opción de información y ayuda.
- Alt-X: Abandonar el programa.

4.9 SUBRUTINAS DESARROLLADAS.

El programa está construido casi en su totalidad por subrutinas encargadas de cumplir una tarea específica dentro del sistema de adquisición.

Se puede encontrar varios tipos de subrutinas: subrutinas de información, subrutinas de lectura o adquisición de los datos procesados por el controlador lógico programable, subrutinas de escritura de datos en los distintos archivos de datos del PLC, subrutinas de graficación y presentación de los datos adquiridos al usuario, subrutinas de almacenamiento en disco de los datos obtenidos del PLC y otras varias subrutinas de procesamiento de datos que permiten el adecuado funcionamiento del programa en si.

A continuación se describe brevemente la función de cada una de las opciones disponibles en el programa.

1. OPCION "≡"

   Acerca de...: Esta opción muestra datos relativos a la tesis.

   Calculadora: Permite desplegar una calculadora.

   Salir: Termina el programa.

2. OPCION "**RED**"

   Número de nodo: Esta opción permite seleccionar el número de nodo en una red de PLC's al que se desee ingresar o visualizar los datos.

   Evaluar nodo: Muestra el tipo de PLC que está conectado en una red de PLC's DH-485.

3. OPCION "**Lectura PLC**"

   Output: Lee el estado de las salidas O:0 sean éstas analógicas o digitales.

   Input: Lee el estado de las entradas I:1 sean éstas analógicas o digitales.

   Bit: Lee el estado del archivo de datos tipo bit B:3.

Timer: Permite leer tanto el valor prefijado como el acumulado de cualquier temporizador T:4 programado en el PLC.

Counter: Lee el valor prefijado y el valor acumulado de un contador C:5.

Control: Lee los registros de control R:6.

Integer: Lee el archivo de datos entero N:7.

N:9: Lee el archivo de datos enteros N:9.

4. OPCION "**Escritura PLC**"

Bit: Escribe el valor ingresado en el archivo de datos tipo bit B:3.

Timer: Permite escribir tanto el valor prefijado como el valor acumulado de cualquier temporizador T:4 programado en el PLC.

Counter: Escribe el valor prefijado como el valor acumulado de un contador C:5.

Control: Escribe cualquier valor en los registros de control R:6.

Integer: Escribe el valor ingresado en el archivo de datos entero N:7.

N:9: Escribe un valor en el archivo de datos enteros N:9.

5. OPCION "**PID**"

Control PID: Visualiza el comportamiento de la entrada y salida del módulo analógico conectado a un PLC que ejecuta un algoritmo de control PID desarrollado en base a una ecuación de diferencias o utilzando la instrucción para tal efecto.

On/Off: Activa o desactiva el control PID. Permite cambiar de modo manual a automático y viceversa. Opción no válida para el control utilizando la instrucción PID

6. OPCION "**TANQUE**"

Tanque solo: Cuando la planta sea el prototipo de nivel de líquidos, esta opción muestra un gráfico del prototipo en el que se observa la variación del nivel del líquido.

Tanque combinado: Permite observar simultáneamente la respuesta en el tiempo del nivel del líquido y el prototipo.

7. OPCION "**ALARMAS**"

Activar: Activa el nivel de alarma superior e inferior.

Desactivar: Desactiva las alarmas.

Fijar niveles: Permite ingresar los niveles a partir del cual las alarmas entrarán en funcionamiento.

8. OPCION "**PANTALLA**"

Eje Y máximo: Opción que permite cambiar la escala superior del eje Y.

Eje Y mínimo: Cambia la escala inferior del eje Y.

Tiempo: Cambia el tiempo de visualización gráfica.

9. OPCION "**INFORMACION**"

Ayuda: Breve ayuda del programa

Gráfico del proceso: Visualización gráfica del sistema de adquisición de datos, supervisión y control PID.

El listado del programa principal y las subrutinas implementadas consta en el anexo F.

4.10 MENSAJES DE ERROR

Los mensajes de error son cajas de diálogo que muestran el tipo de error que ha ocurrido al acceder a alguna de las opciones del programa o al intentar ingresar valores erróneos, sean estos alfanuméricos o fuera de rango.

Errores que pueden presentarse y su causa.

(02 00) Undeliverable message.

Este error puede ocurrir cuando se pretende evaluar una estación (PLC) que no se encuentre físicamente conectada. Se recomienda chequear las conexiones o verificar que el número de nodo exista.

(10 00) Illegal command or format.

Este tipo de error se presenta cuando se pretende leer o escribir en un elemento no definido en la tabla de datos del PLC. Verifique que el elemento donde se va a efectuar una operación de lectura o escritura, se encuentre definido en el PLC.

(00 19) 1784-KR driver not installed.

Este error ocurre cuando se intenta ejecutar cualquier comando de lectura o escritura en el PLC y previamente no se ha inicializado la comunicación a través de la ejecución del archivo START.BAT. Siempre, antes de ejecutar el programa ASC-PID.EXE es necesario ejecutar el archivo START.BAT que inicializa ciertos parámetros para establecer la comunicación (ver sección 4.6.2).

(00 1D) Reply timeout.

Se declara este error cuando el controlador lógico programable no envía, dentro del tiempo máximo establecido (5 segundos), un mensaje con la réplica al comando último enviado. Esto puede ocurrir por una desconexión física entre el PLC y la tarjeta 1784-KR.

No se puede cargar adaptador gráfico.
Device driver file not found (EGAVGA.BGI)
Graphics hardware not detected.

Estos mensajes de error ocurren cuando el programa, no ha encontrado en el directorio de trabajo, el archivo que habilita las funciones gráficas utilizadas. Verifique que en el directorio de trabajo se encuentre el archivo EGAVGA.BGI.

No se puede abrir el archivo de configuración CONFIG.PID

Este mensaje suele ocurrir cuando, en el directorio de trabajo, no se encuentra el archivo que contiene la información necesaria para ejecutar el programa ASC-PID.EXE. Verifique la presencia del archivo CONFIG.PID en el directorio de trabajo.

Ingresado dato alfanumérico o fuera de rango

Ocurre este tipo de mensaje cuando se ha ingresado un parámetro alfanumérico o su valor no esta dentro del rango permitido. Ingrese correctamente el dato.

5. RESULTADOS Y CONCLUSIONES

5. RESULTADOS Y CONCLUSIONES.

5.1 DESCRIPCION DE LOS SISTEMAS UTILIZADOS

Debido a que el módulo analógico 1746-NIO4I disponible el controlador lógico programable Allen-Bradley SLC 5/02 tiene dos salidas por corriente, fue necesario para las aplicaciones probadas, recurrir a un circuito de acondicionamiento de señal, un circuito que convierta la señal eléctrica de corriente en su equivalente de voltaje.

Para este propósito se diseñó el circuito eléctrico básico de la Fig. 5-1:

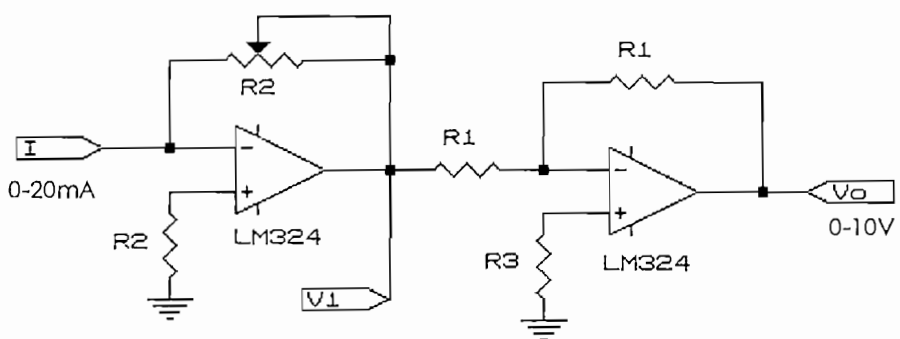


Fig. 5-1: Conversor corriente-voltaje. R1=10kΩ, R2=330Ω, R3=4.7kΩ

El circuito de la figura anterior, convierte la señal de corriente de 0-20 mA proveniente de la salida analógica del módulo, en una señal de voltaje entre 0 y 10 Vdc que será aplicada a la planta analizada. Consta de dos etapas: la primera constituye un conversor corriente a voltaje con ganancia negativa variable, y la segunda un amplificador inversor con ganancia unitaria.

El funcionamiento del sistema de control PID utilizando un controlador lógico programable que dispone o no de la instrucción PID, así como el software desarrollado para la adquisición de datos, supervisión y control fue experimentalmente comprobado con diversos tipos de plantas:

- Circuito eléctrico R-C de primer orden.
- Circuito eléctrico R-C de segundo orden.
- Prototipo de control de nivel de líquidos.
- Servomecanismo Motomatic.

5.1.1 CIRCUITO ELECTRICO R-C DE PRIMER ORDEN

Experimentalmente se efectuó pruebas con un circuito eléctrico resistivo-capacitivo (R-C), equivalente a una planta de primer orden, de las siguientes características:

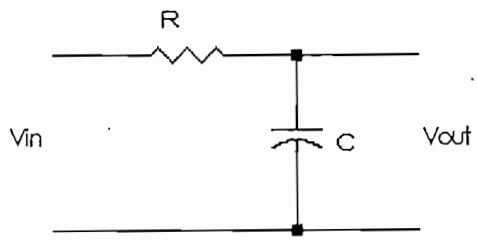


Fig. 5-2: Circuito R-C de primer orden.

La función de transferencia del circuito de la Fig. 5-2 es la siguiente:

$$Gp(s) = \frac{1}{RC \cdot s + 1}$$

Con una resistencia R=10kΩ y un condensador C=1000μF, se tiene:

$$Gp1\ (s) = \frac{1}{10 \cdot s + 1}$$

A continuación se muestra la respuesta del sistema ante una entrada paso de 5V.

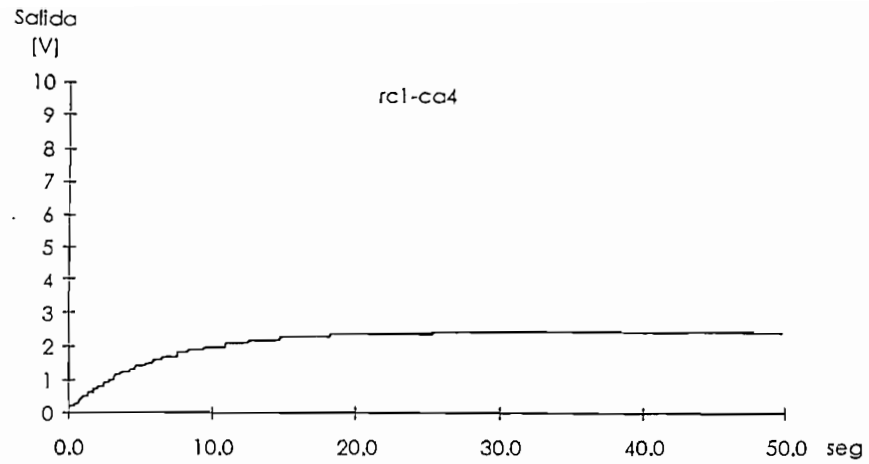


Fig. 5-3: Respuesta de Gp1(s) en lazo cerrado sin compensación.

Como era de esperar, el sistema presenta un error en estado estable del 50% en lazo cerrado.

Ahora, con R=10kΩ y C=470μF, se tiene la siguiente función de transferencia:

$$Gp2\ (s) = \frac{1}{4.7 \cdot s + 1}$$

Se trata de un sistema con una constante de tiempo menor que el sistema anterior. En lazo cerrado, de igual manera, el sistema tiene 50% de error en estado estable.

5.1.2 CIRCUITO ELECTRICO R-C DE SEGUNDO ORDEN

El siguiente circuito eléctrico R-C, equivalente a una planta de segundo orden, fue también objeto de pruebas.

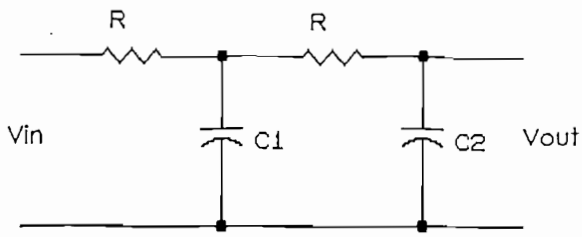


Fig. 5-4: Circuito R-C de segundo orden. R=10kΩ, C1=1000μF, C2=470μF.

La función de trasferencia del circuito de segundo orden correspondiente a la figura anterior es:

$$Gp(s) = \frac{1}{s^2 \cdot (C1 \cdot C2 \cdot R^2) + s \cdot (R \cdot C1 + 2 \cdot R \cdot C2) + 1}$$

Reemplazando R=10kΩ, C1=1000μF y C2=470μF, se tiene:

$$Gp3\ (s) = \frac{1}{47s^2 + 19.4s + 1}$$

A continuación, la respuesta de este sistema, a una entrada escalón de 5V.

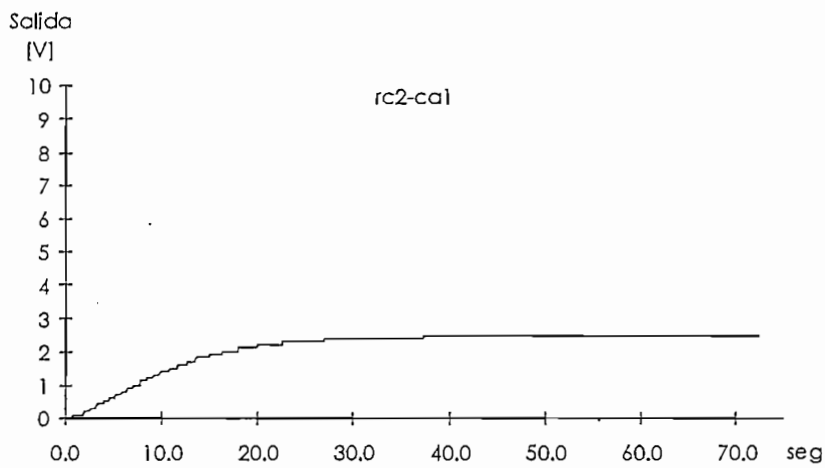


Fig. 5-5: Respuesta del sistema R-C de segundo orden en lazo cerrado sin compensación.

Como predice la teoría, el sistema presenta un error en estado estable del 50%, en lazo cerrado.

5.1.3 BREVE DESCRIPCION DEL PROTOTIPO DE NIVEL DE LIQUIDOS[1]

El comportamiento del algoritmo PID implementado en el controlador lógico programable fue también probado experimentalmente con el Prototipo de Control de Nivel de Líquidos disponible en el Laboratorio de Sistemas de Control.

El prototipo de control de nivel de líquidos es un módulo didáctico que puede responder como un sistema de primero, segundo o tercer orden dependiendo del número de tanques acoplados que se utilicen en una determinada aplicación. Cabe mencionar que el control se realiza solamente sobre la altura o nivel del líquido del último tanque. En él, se sensa indirectamente el nivel del líquido midiendo su presión diferencial.

SENSOR DE PRESION DIFERENCIAL

El módulo utiliza un sensor de presión diferencial de estado sólido, para medir la presión hidrostática en el fondo del último tanque del prototipo. Esta presión hidrostática es directamente proporcional a la altura del líquido que la produce. Se trabaja en una configuración diferencial para evitar la influencia de la presión atmosférica.
El sensor de presión está montado sobre una tarjeta de amplificación que acondiciona la señal de salida del sensor a un voltaje de 0 a 10 Vdc a plena escala.

BOMBA Y ACONDICIONAMIENTO DE SEÑAL

A través de una bomba centrífuga accionada por un motor DC, el agua es enviada desde el reservorio hacia los tanques del prototipo. El circuito implementado para el acondicionamiento de señal controla la velocidad del motor de la bomba en

---

[1] Benítez Diego, "Diseño y Construcción de un Sistema de Control de Nivel de Líquidos", tesis de grado, EPN-FIE, 1994.

proporción directa al voltaje de entrada al circuito, mediante el empleo de un chopper de DC. El voltaje de entrada posee un rango válido de 0 a 10 Vdc.

VALVULAS DE CONTROL DE FLUJO

Los tanques del prototipo se encuentran acoplados mediante estas válvulas de control de flujo que varían el caudal de agua de un tanque a otro. El caudal de agua que sale del último tanque es también controlado por una válvula de desfogue de este tipo.

FUNCION DE TRANSFERENCIA DEL PROTOTIPO

Como resultado de la modelación de la planta y todos sus componentes, se obtuvieron las siguientes funciones de transferencia del prototipo:

a) Sistema de primer orden:

$$Gp4(s) = \frac{0.028}{s + 0.2240}$$

b) Sistema de segundo orden:

$$Gp5(s) = \frac{0.0179}{s^2 + 1.4908s + 0.1419}$$

5.1.4 SERVOMECANISMO MOTOMATIC.

Constituye un motor de corriente continua en el cual se controlará la velocidad, aplicando un voltaje variable a la armadura. Su velocidad es sensada por medio de un tacómetro que produce un voltaje de 0 a 30 Vdc proporcional a su velocidad, por lo que es indispensable calibrar un potenciómetro (K) disponible en este equipo para acondicionar este voltaje al rango de trabajo del módulo analógico (0 a 10 Vdc).

El diagrama de bloques del servomecanismo, experimentalmente se ha determinado como[2]:

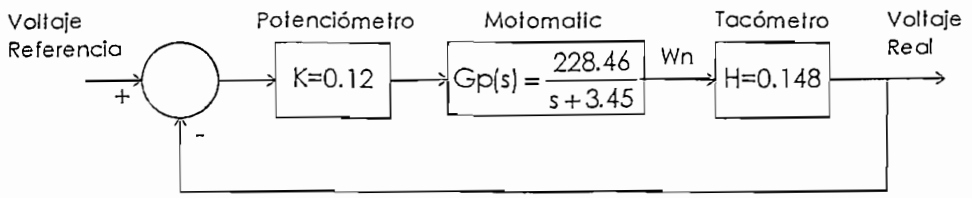


Fig. 5-6: Diagrama de bloques del Motomatic.

Teóricamente se espera un error en estado estable del 49.5% en lazo cerrado. Esto se verifica experimentalmente con la siguiente gráfica elaborada con los datos adquiridos por el PLC, en lazo cerrado, ante una entrada paso de magnitud 7 Vdc y sin ninguna compensación.

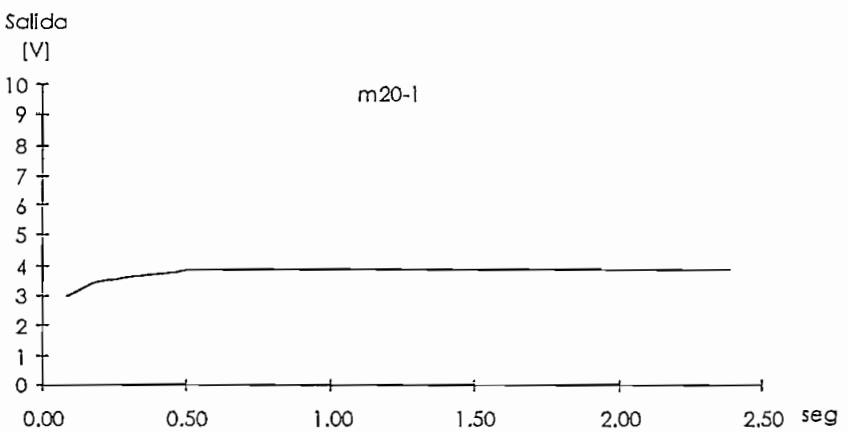


Fig. 5-7: Respuesta paso del Motomatic en lazo cerrado sin compensador.

[2] Bastidas Jorge, "Estudio teórico experimental de un servomecanismo de posición y velocidad", tesis de grado, EPN-FIE, 1981.

5.2 RESULTADOS EXPERIMENTALES EN PLC SIN RUTINA PID

5.2.1 PRUEBA No. 1

Esta prueba consiste en realizar el control sobre la planta R-C de primer orden descrita por la función de transferencia Gp1(s).

A continuación se presentan el diseño teórico del compensador así como también su comportamiento experimental.

DISEÑO DEL CONTROLADOR PID EN EL PLANO "Z".

Se considera que el controlador lógico programable está procesando la ecuación de diferencias correspondiente al algoritmo PID cada 640 mseg (valor dentro del rango permitido por el PLC e inferior a la menor constante de tiempo del sistema).

Discretizando la función de transferencia de la planta Gp1(s) mediante un zero order hold para pasar al dominio "Z", con el periodo de muestreo anterior, es decir, aplicando la ecuación 3.13 a la función de transferencia Gp1(s), se tiene que:

$$Gp1(z) = \frac{0.061995}{z - 0.938005}$$

El controlador que se aplicará, es un Proporcional-Integral (PI) de la siguiente forma:

$$Gc1(z) = \frac{k(z-a)}{z-1}$$

donde: $k = kp+ki$

$a = kp/(kp+ki)$

cancelando el polo en $z = 0.938$, se tiene: $a = 0.938$

Ahora, para lograr un tiempo de establecimiento de aproximadamente 8 segundos, se debe cumplir que: $0.061995k = 0.32$ (análisis realizado simulando la planta en el programa CadControl)

Reemplazando se tiene que:

$$kp = 4.84$$

$$ki = 0.31$$

Para una entrada paso de magnitud 5V, con las constantes anteriores se obtuvo el siguiente resultado:

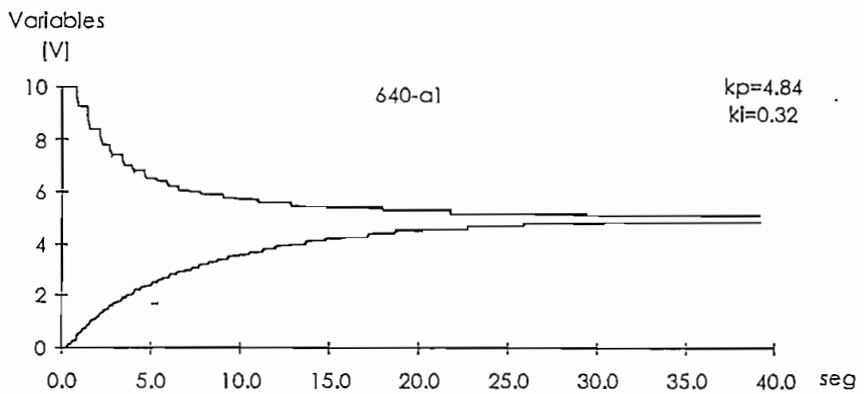


Fig. 5-8: Respuesta del sistema R-C de primer orden utilizando las constantes determinadas teóricamente.

Mp=0% Ess=0% ts (98%)=30.4 seg.

Como se puede observar, el sistema con las constantes teóricas encontradas responde de manera muy lenta.

Para el mismo tipo de entrada con kp=5 y ki=0.8 se lograron mejores resultados:

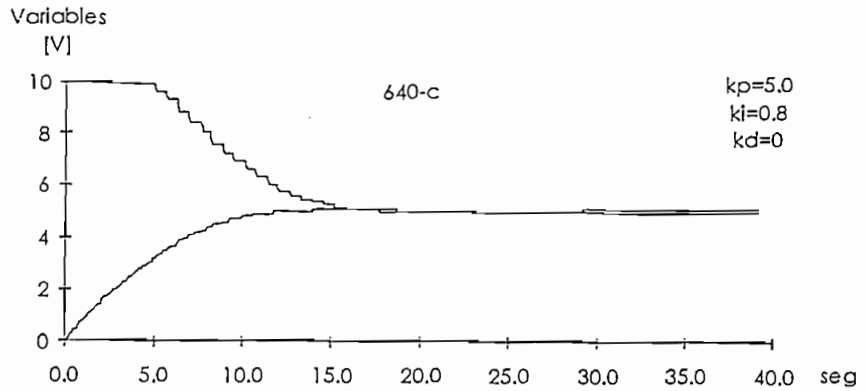


Fig. 5-9: Respuesta a una entrada paso del sistema R-C de primer orden.

Mp=2% Ess=0% ts(98%)=10.6 seg.

A pesar de que el sistema responde con un pequeño sobreimpulso, tiene mejores características que el anterior. El sobreimpulso puede disminuirse con un incremento ligero de la constante kp.

DISEÑO DEL CONTROLADOR PID EN EL PLANO "S".

Para el mismo sistema de primer orden, el controlador PI tiene la siguiente forma:

$$Gcl(s) = \frac{kp \cdot (s + \frac{ki}{kp})}{s}$$

Por cancelación de polos: ki/kp=0.1. Para lograr una estabilización de aproximadamente 8 segundos (simulación en programa CadControl): 0.1kp=0.5

Con las consideraciones anteriores, se tiene:

$$kp = 5.0$$

$$ki = 0.5$$

La siguiente figura muestra los resultados experimentales obtenidos con las constantes determinadas anteriormente, un periodo de muestreo de 640 mseg. y una entrada paso de magnitud 5 Vdc.

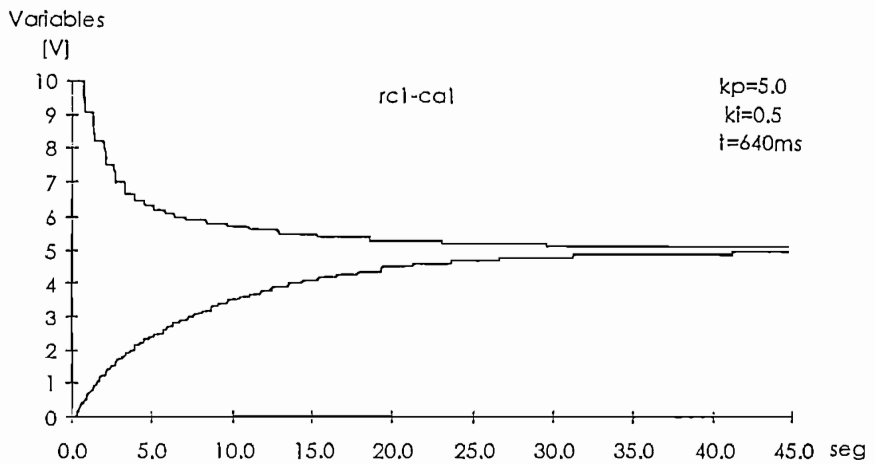


Fig. 5-10: Respuesta del sistema R-C de primer orden. Método continuo.

Mp=0% Ess=0% ts(98%)=31.27 seg.

Mejores resultados fueron conseguidos utilizando las constantes siguientes calibradas directamente en la planta, en base a las referencias teóricas calculadas: Kp=8.0, Ki=1.5

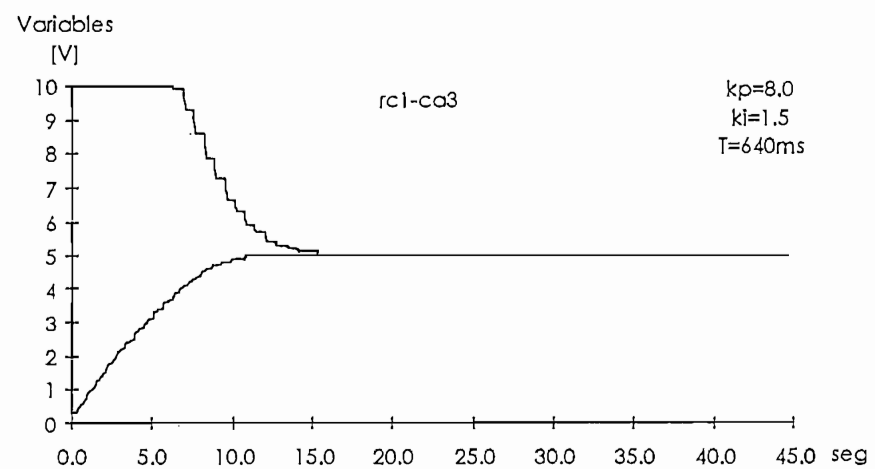


Fig. 5-11: Respuesta a una entrada paso.

Mp=0% Ess=0% ts(98%)=9.97 seg.

Analizando los dos métodos de diseño, se puede observar que los resultados teóricos, en los dos casos, no satisfacen las condiciones de tiempo de establecimiento planteadas, pero si logran conseguir un error en estado estable igual a cero. Los dos métodos dan resultados muy parecidos, requiriéndose una posterior calibración de las constantes, para aproximarse al tiempo de establecimiento planteado.

5.2.2 PRUEBA No. 2

Esta prueba consiste en controlar el circuito R-C de primer orden descrito por la función de transferencia Gp2(s).

DISEÑO DEL CONTROLADOR PID EN EL PLANO "Z"

Discretizando la función de transferencia Gp2(s) mediante un zero-order hold con un periodo de muestreo de 80 mseg, se tiene:

$$Gp2\,(z) = \frac{1.687 \times 10^{-2}}{z - 0.9831}$$

El controlador para esta planta tiene la misma forma que el descrito para la prueba No.1, es decir Gc1(z). Para conseguir un tiempo de estabilización de aproximadamente 8 segundos, realizando una simulación de la planta y el compensador en el programa CadControl, se llega a determinar que Kp=2.06 y Ki=0.03.

Con las constantes y periodo de muestreo anteriores, ante una entrada paso de magnitud 5 Vdc, se obtuvo la siguiente respuesta:

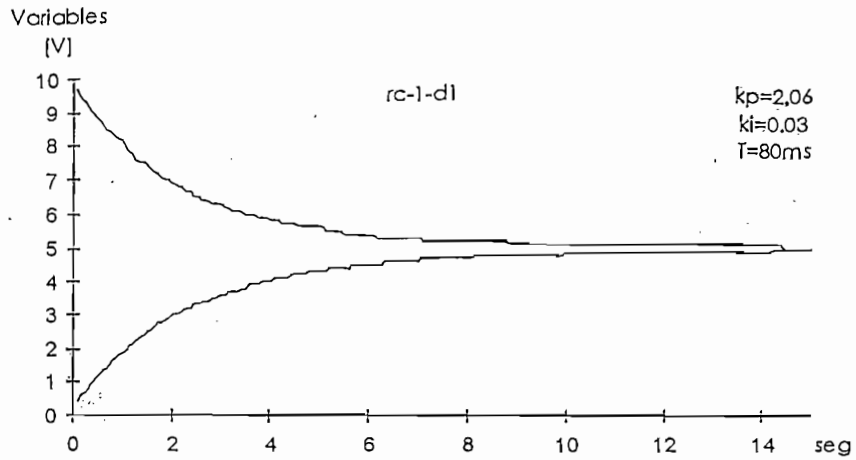


Fig. 5-12: Respuesta paso de Gp2(s) con las constantes teóricas.

Mp=0% Ess=0% ts(98%)=9.96 seg.

Con una calibración ligera de las constantes directamente en el proceso, se consiguió el siguiente resultado.

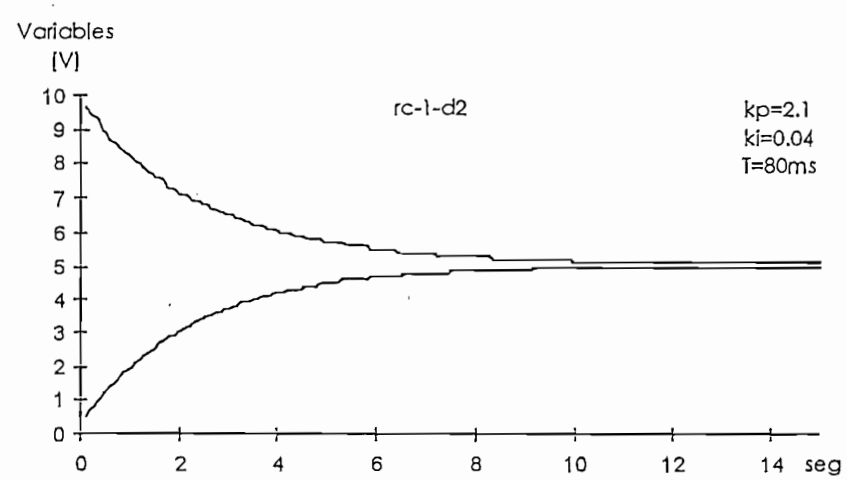


Fig. 5-13: Respuesta de Gp2(s) ante una entrada paso de 5V con compensación.

Mp=0% Ess=0% ts(98%)=7.49 seg.

DISEÑO DEL CONTROLADOR PID EN EL PLANO "S"

El controlador PI a diseñar tiene la forma

$$Gc2(s) = \frac{kp \cdot (s + \frac{ki}{kp})}{s}$$

Realizando la simulación de esta planta, para lograr un tiempo de establecimiento de aproximadamente 8 segundos, se determina que:

Kp=2.35 Ki=0.5

El siguiente resultado fue obtenido con las constantes anteriores y un periodo de muestreo de 80 mseg.

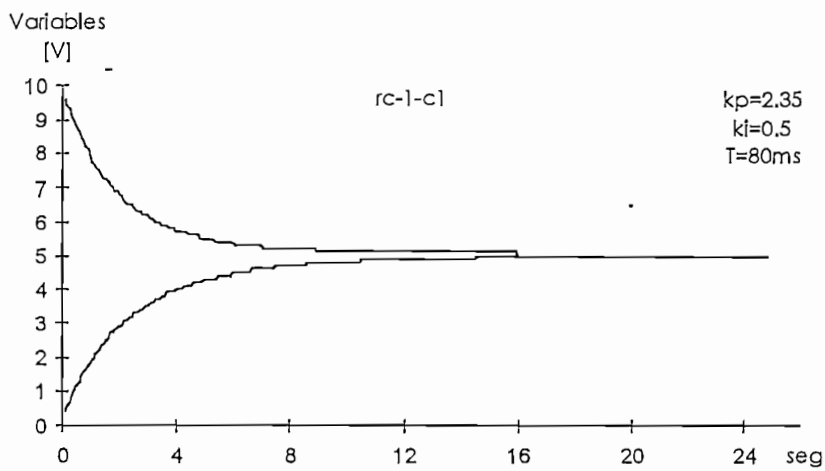


Fig. 5-14: Respuesta de Gp2(s). Diseño del controlador en el plano "S".

Mp=0% Ess=0% ts(98%)=10.47seg.

Mejor resultado se consiguió variando ligeramente las constantes determinadas teóricamente.

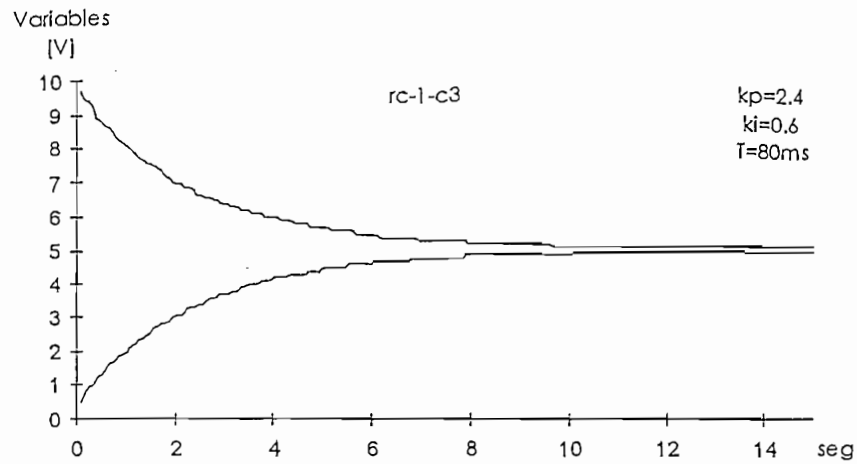


Fig. 5-15: Respuesta de Gp2(s) compensada ante una entrada paso de 5V.

Mp=0% Ess=0% ts(98%)=7.92 seg.

5.2.3 PRUEBA No. 3

A continuación se realiza el diseño teórico en el plano "S" del controlador para el circuito R-C de segundo orden descrito por la función de transferencia Gp3(s).

Debido a las características de este sistema, un controlador PI es suficiente para lograr un error en estado estable cero y un tiempo de establecimiento bueno.

$$Gp3(s) = \frac{1}{47(s+0.35)(s+0.06)}$$

$$Gc3(s) = \frac{kp \cdot \left(s + \frac{ki}{kp}\right)}{s}$$

Cancelando el polo de la planta más cercano al origen, haciendo una simulación en el programa CadControl para lograr un tiempo de establecimiento de aproximadamente 20 seg., se tiene que:

Kp=2 Ki=0.12

La figura que a continuación se muestra corresponde a la respuesta del sistema con las constantes anteriores, una referencia paso de 5V y un periodo de muestreo de 80 mseg.

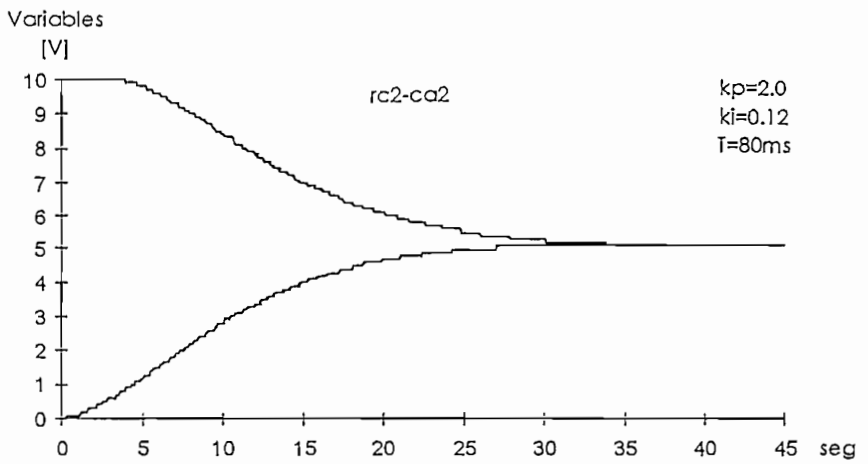


Fig. 5-16: Resultado del sistema R-C de segundo orden con constantes teóricas.

Mp=2% Ess=0% ts(98%)=22.4 seg.

Con un pequeño incremento de la constante proporcional se logra reducir el mínimo sobreimpulso de la respuesta anterior, a pesar de incrementarse el tiempo de establecimiento.

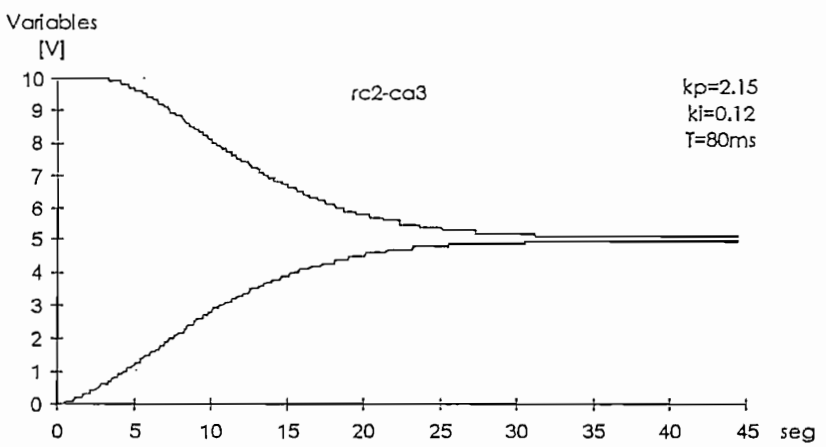


Fig. 5-17: Respuesta del sistema R-C de segundo orden.

Mp=0% Ess=0% ts(98%)=25.63 seg.

5.2.4 PRUEBA No. 4

A continuación, se realiza el diseño en el plano "S" del controlador PI para el sistema de nivel de líquidos usando un solo tanque (sistema de primer orden).

Para cancelar el polo de la planta Gp4(s), se debe cumplir que Ki/Kp=0.224. Basándose en una simulación de la planta, utilizando el programa CadControl, se puede determinar que, para conseguir un tiempo de establecimiento de aproximadamente 20 segundos, la constante proporcional debe ser Kp=7.14. Reemplazando en la relación anterior, Ki=1.6.

Partiendo de un sistema estable en 15.2 cm, se fijó una nueva referencia en 26.6 cm (perturbación de 11.4 cm). La respuesta obtenida considerando las constantes anteriores y un periodo de muestreo de 250 mseg. es la siguiente:

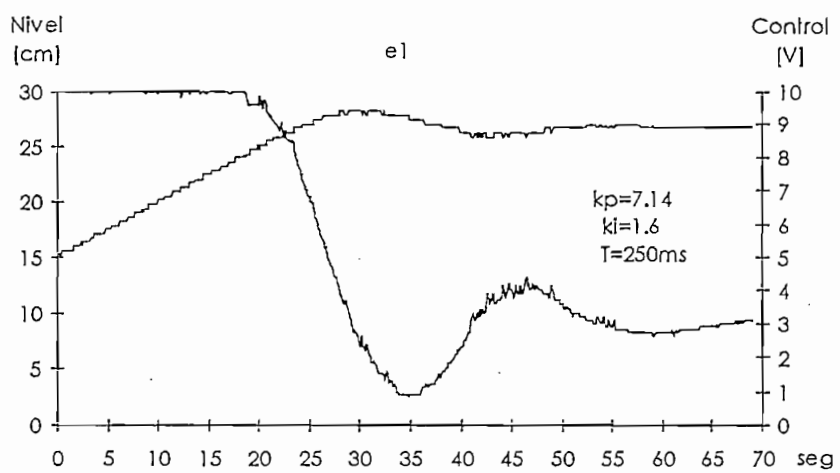


Fig. 5-18: Respuesta del prototipo de nivel utilizando las constantes determinadas teóricamente.

Mp=5.71% Ess=0% ts (98%)=36.41 seg.

Se puede observar una señal de control un tanto oscilante, un sobreimpulso notorio y un tiempo de establecimiento aceptable.

Resultados más satisfactorios se lograron con las siguientes constantes calibradas directamente sobre la planta y considerando un periodo de muestreo de 500 mseg:

Kp=9.5 Ki=0.75

Fig. 5-19: Resultados en el prototipo de nivel de líquidos. Sistema de primer orden.

Mp=0% Ess=0% ts (98%)=25.35 seg.

5.2.5 PRUEBA No. 5

Esta prueba consiste en controlar la velocidad del servomecanismo Motomatic descrito por el diagrama de bloques de la Fig. 5-6.

Realizando el diseño en el plano "S" del controlador se tiene que, cancelando el polo en $s = -3.45$ y para lograr un tiempo de establecimiento de aproximadamente 1.5 segundos se debe cumplir que Kp=0.65 y Ki=2.27.

Experimentalmente se lograron resultados satisfactorios con Kp=1.0, Ki=2.5 y un periodo de muestreo de 20 mseg., ante una entrada paso de magnitud 7 Vdc.

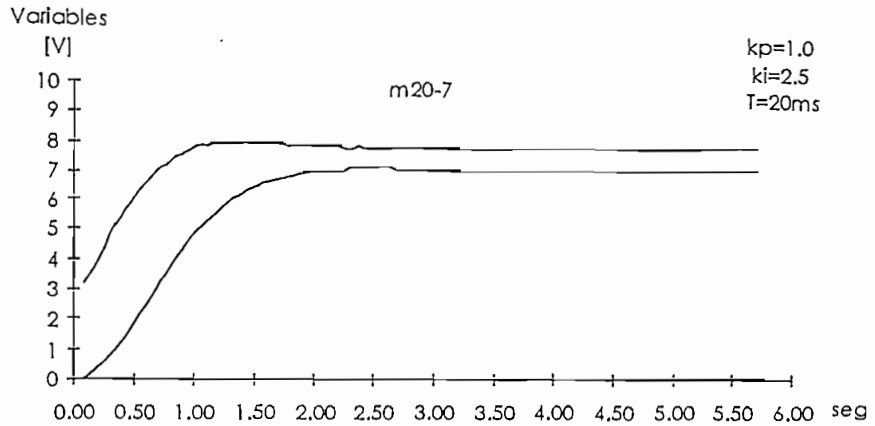


Fig. 5-20: Respuesta paso del Motomatic con compensación.

Mp=1.4% Ess=0% ts(98%)=1.87 seg.

5.3 RESULTADOS EXPERIMENTALES EN PLC CON RUTINA PID

5.3.1 PRUEBA No. 6

Siguiendo el procedimiento descrito en secciones anteriores para encontrar las constantes más adecuadas del controlador PID, programado utilizando la instrucción que incluye el PLC en su conjunto de operaciones básicas, se probó la planta R-C de primer orden descrita por la función de transferencia Gp1(s) con kp=2.0 y ki=0.1, lográndose el siguiente resultado ante una entrada paso de 4.6V y un periodo de muestreo de 20 mseg.

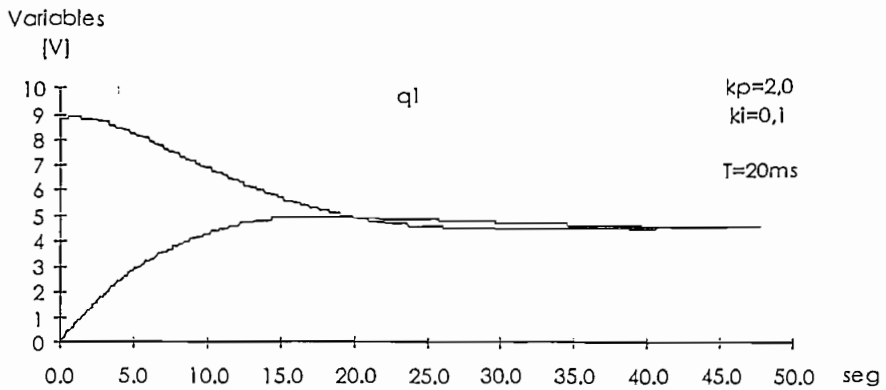


Fig. 5-21: Respuesta del sistema R-C de primer orden utilizando la instrucción PID del PLC

Mp=8.69% Ess=0% ts(98%)=29.6 seg.

Al incrementar la constante proporcional (Kp=10) se observa un sistema más rápido y una saturación de la señal de control. El sistema responde sin sobreimpulso, sin error y tiene un tiempo de establecimiento bueno.

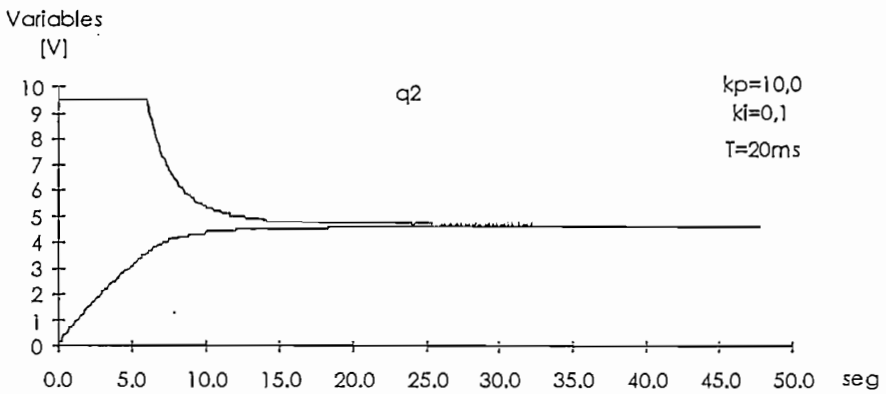


Fig. 5-22: Respuesta a una entrada paso de 4.6 Vdc del sistema R-C de primer orden con otras constantes.

Mp=0% Ess=0% ts(98%)=12.1 seg.

5.3.2 PRUEBA No. 7

De igual forma, el circuito R-C de primer orden descrito por la función de transferencia Gp2(s), fue controlado por la instrucción PID del PLC con las siguientes constantes calibradas directamente sobre la planta. A continuación su resultado ante una entrada paso de magnitud 5 Vdc.

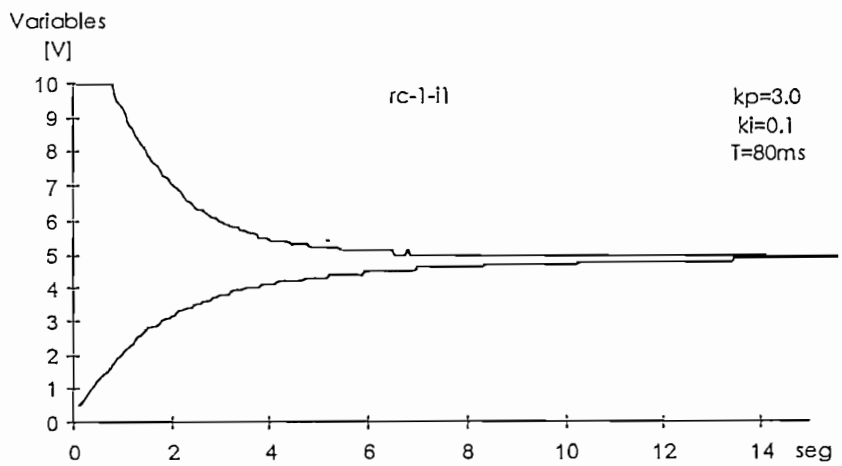


Fig. 5-23: Respuesta de Gp2(s) compensada utilizando instrucción PID.

Mp=0% Ess=0% ts(98%)=13.45 seg.

5.3.3 PRUEBA No. 8

Esta prueba fue realizada con la instrucción PID del PLC en el circuito R-C de segundo orden descrito por la función de transferencia Gp3(s).

Las constantes del controlador PID, utilizando la instrucción para tal efecto, fueron calibradas directamente en la planta. Para un periodo de muestreo de 40 mseg. y con Kp=3, Ki=0.2, se obtuvo el siguiente resultado ante una entrada paso de magnitud 5 Vdc.

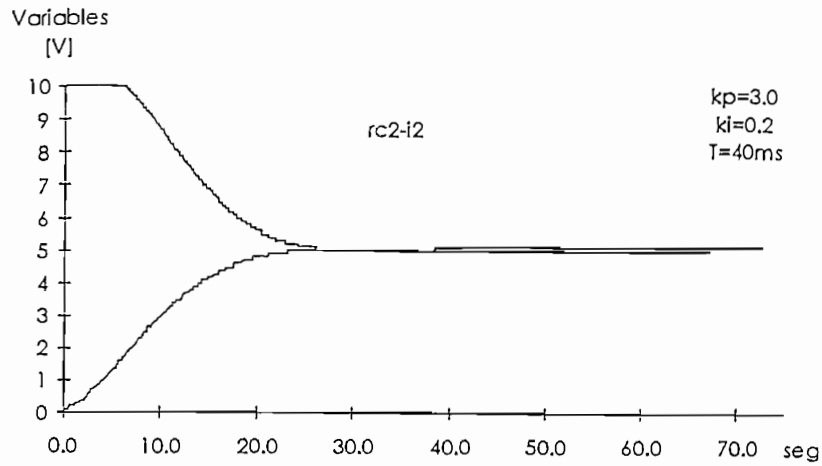


Fig. 5-24: Resultado del sistema R-C de segundo orden utilizando la instrucción PID del PLC.

Mp=0% Ess=0% ts(98%)=21.11 seg.

5.3.4 PRUEBA No. 9

Considerando un sistema de primer orden (un solo tanque) en el prototipo de nivel de líquidos y partiendo de un punto de trabajo estable en 15.2 cm, se dio una perturbación de 11.4 cm (referencia en 26.6 cm), obteniéndose los siguientes resultados con un periodo de muestreo de 250 mseg.

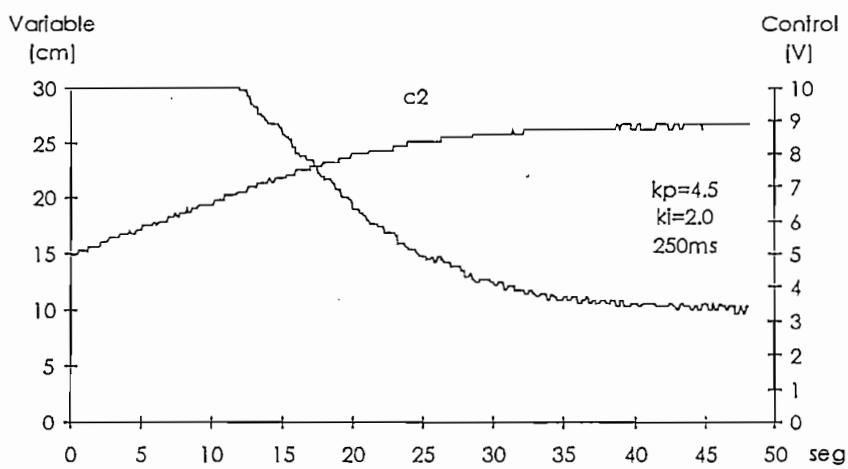


Fig. 5-25: Respuesta del prototipo de nivel de líquidos utilizando la instrucción PID.

Mp=0% Ess=0% ts(98%)=32.1 seg.

5.4 CONCLUSIONES

Del trabajo y de las pruebas realizadas se obtuvieron resultados y conclusiones importantes, entre ellas se puede citar las siguientes:

- Se recomienda no utilizar directamente la ecuación en diferencias obtenida al realizar el diseño del controlador PID, es decir aquella en la que intervienen valores anteriores de la señal de error y el valor actual, debido a que el rango de trabajo para las operaciones matemáticas en el PLC es limitado. Se aconseja utilizar una variación matemática de esta ecuación, es decir, aquella en la que no intervienen las mediciones de la señal de error directamente sino más bien sus incrementos, debido a que estos valores obviamente son mucho más pequeños, razón por la cual las operaciones matemáticas para la determinación de los términos del algoritmo de control PID ejecutado por el PLC resultan no ser numéricamente grandes.

- Para el diseño del controlador PID que se implementa en el controlador lógico programable que no dispone de este tipo de instrucción, se analizaron dos alternativas: realizar un diseño del controlador PID en el plano "S" y realizar un diseño en el plano "Z". Cuando se realiza un diseño en el plano "S" de un sistema de datos muestreados, en los cálculos para la determinación de los coeficientes de la ecuación en diferencias del controlador PID interviene el periodo de muestreo, esto produce que en la realización del algoritmo en el PLC se añada más operaciones matemáticas. Mientras más operaciones matemáticas intervengan, se introduce más error debido al redondeo que realiza el PLC en todos los resultados de las operaciones matemáticos (el PLC no maneja números decimales). Por esto, al realizar un diseño discreto (plano "Z"), en el que obviamente no interviene el periodo de muestreo (en realidad interviene para el diseño en la discretización mediante zero order hold de la planta) se requiere de menos operaciones matemáticas evitando así los errores que pueden introducirse debido al redondeo en dichas operaciones.

- Existe sin embargo una "desventaja" de realizar el diseño en el dominio discreto y "evitar" el uso, en el algoritmo de control, del periodo de muestreo, y es el hecho de que una vez realizado un diseño continuo del sistema de datos muestreados, los valores encontrados para las constantes proporcional, integral y derivativa que teóricamente logran controlar el proceso, determinan una ecuación en diferencias específica. Si por algún motivo, el periodo de muestreo cambia, los coeficientes de la ecuación en diferencias automáticamente cambiarán para seguir controlando el proceso debido a que están en función de Kp, Ki, Kd y del periodo de muestreo. Por tanto el algoritmo de control digital resulta distinto ante cualquier cambio en el periodo de muestreo, no requiriéndose, en la mayoría de los casos, rediseñar el controlador. En el diseño discreto, ocurre totalmente lo contrario. Los coeficientes del algoritmo de control digital son solamente función de las constantes Kp, Ki y Kd (constantes determinadas por un diseño en el dominio discreto para un determinado periodo de muestreo), por ende, para un controlador ya diseñado, cualquier variación del periodo de muestreo no alterará en forma alguna el algoritmo digital de control, pudiendo "posiblemente" perder las condiciones adecuadas de operación. Entonces, en un diseño puramente discreto, una variación del periodo de muestreo provocaría obligatoriamente un rediseño del controlador, si es que se requiere mantener exactamente las condiciones anteriores en la planta.

- El periodo de muestreo que se escoja para realizar el control digital resulta de fundamental importancia para el diseño discreto de las constantes del controlador y fundamentalmente para el adecuado control del proceso. Al realizar un diseño puramente discreto del controlador PID, el periodo de muestreo interviene en la determinación del equivalente discreto de la función de transferencia de la planta (discretizada mediante zero order-hold). Diseñando en el plano "S", el periodo de muestreo, interviene directamente en los coeficientes del algoritmo digital de control. Este periodo de muestreo debe ser escogido siguiendo las normas determinadas en la teoría de control digital y debe estar dentro del rango permitido por el PLC.

- El escalamiento de la ecuación en diferencias fue necesario efectuarlo para permitir que las constantes del controlador PID tengan parte fraccionaria. Se trabajó con un

escalamiento de 100, lo que producía constantes con una precisión de centésimas. Un escalamiento superior (por ejemplo por 1000), produciría constantes con tres números decimales, pero a su vez provocaría que los términos de la ecuación en diferencias crezcan en la misma proporción, De ahí que se produce un conflicto entre el número de decimales que puede tener las constantes del controlador PID y la magnitud de las operaciones matemáticas así como de los términos de la ecuación en diferencias. A mayor escalamiento, más grande son las operaciones matemáticas y los términos de la ecuación en diferencias, lo que puede provocar condiciones de "fuera de rango" en el procesador del PLC. Si bien es cierto, cuando se produce una condición de "fuera de rango" se procede a truncar la magnitud del resultado en -32768 o en +32767, esta condición provoca, como es lógico, errores en la evaluación del algoritmo de control.

- De las pruebas realizadas se puede apreciar que no es posible reducir los tiempos de estabilización más allá de un cierto límite debido fundamentalmente a que al inicio del control existe **saturación** en el sistema. Este comportamiento, en su inicio no lineal, del sistema es uno de los motivos para que los diseños teóricos no se comporten adecuadamente y requieran una calibración directamente sobre la planta. La característica mencionada anteriormente es muy evidente en las pruebas realizadas en el prototipo de nivel de líquidos: ante un cambio de referencia, la bomba se satura entregando su máximo caudal, situación que se mantiene hasta que el nivel real esté próximo al de referencia, momento en el cual el sistema se comporta linealmente (ya no existe saturación) permitiendo al controlador actuar normalmente con los parámetros que éste dispone. Adicionalmente se debe considerar que los diseños teóricos incluidos en este trabajo, se basan en modelos también teóricos de las plantas estudiadas. Dichos modelos son, sin lugar a dudas, solo aproximaciones matemáticas del comportamiento real de la planta.

- Los excelentes resultados obtenidos en el control PID de las plantas estudiadas se deben en buena manera a disponer de un PLC de procesamiento rápido, una adecuada calibración de las constantes del controlador PID, el escogimiento correcto del periodo de muestreo y sobre todo la utilización de un módulo analógico con una muy buena resolución (16 bits para las entradas analógicas y 14

bits para las salidas). Todo esto produce un control digital con muy buenas características.

- Como se puede observar en las gráficas de respuesta incluidas en este capítulo, las señales de control, en general, tienen una tendencia suave, no posee variaciones bruscas, lo que da lugar a un incremento de la vida útil de los actuadores utilizados. Esto es crítico si consideramos que en el campo industrial estos dispositivos son por lo general costosos y requieren de ciertos cuidados.

- Si se compara el control PID realizado en base a un algoritmo digital programado mediante sumas y restas y el control PID utilizando la instrucción que dispone el PLC, se puede determinar varias diferencias tales como: facilidad de programación del controlador PID, el rango de sus constantes, la mínima variación que se puede introducir en las constantes, el espacio de memoria que utiliza. En cuanto a la facilidad de programación, no cabe dudas en afirmar que, la instrucción PID que incluyen ciertos PLC's en su conjunto de comandos básicos, tiene la ventaja de no requerir programación para su funcionamiento sino tan solo necesita de una adecuada calibración de sus constantes. El rango de las constantes al emplear el algoritmo digital de control PID en base a sumas y restas, es más amplio [0-327.67] si lo comparamos con el correspondiente al utilizar la instrucción PID (ver tabla 3-2). Tan solo al utilizar un procesador superior (modelo A-B SLC 5/03) se tiene el mismo rango de trabajo para las constantes. En el caso del algoritmo digital, la mínima variación de todas las constantes es una centésima, no así en el caso de la instrucción PID, en la que se dispone como mínima variación una décima (solo para el término derivativo existe variación de centésimas). Esto es importante por cuanto al tener más precisión en las constantes, se puede lograr una calibración más fina del comportamiento de la planta.

- Se pudo constatar que, al utilizar la instrucción PID del PLC, cualquier calibración de las constantes, en algunos casos producía que la variable controlada sufra perturbaciones graves, es decir, existe un momento luego de la calibración de las constantes, en el cual el sistema se "descontrolaba". Posteriormente a esta perturbación, el PLC lograba controlar satisfactoriamente el sistema. Este

comportamiento particular de la instrucción PID, no se observa de manera alguna en el programa de control PID implementado en base a sumas y restas (ecuación de diferencias).

- Como es bien conocido, los sistemas de control de datos muestreados tienen generalmente un rendimiento inferior que los sistemas de control continuos. Esto es muchas veces explicado por el hecho de que la señal muestreada contiene menos información que la señal continua; sin embargo, en base a los resultados obtenidos, se concluye que el control digital es muy bueno principalmente en sistemas tales como nivel de líquidos, de temperatura, etc.

- Con el desarrollo de este trabajo de tesis, se comprueba la importancia y la potencialidad que tienen los sistemas de adquisición de datos aplicados al control de procesos. Estos sistemas permiten al operador de una planta, entender de manera rápida y efectiva el funcionamiento del proceso y evaluar el estado del mismo en cualquier instante de tiempo sea éste actual o pasado. Gracias a la estación de adquisición de datos (computador personal), es factible realizar un sinnúmero de operaciones adicionales (visualización gráfica, almacenamiento de la información en disco, impresión de resultados, disponibilidad de alarmas, etc.), útiles para analizar el comportamiento del proceso. Como se mencionó en la primera parte de este trabajo de tesis, los controladores lógicos programables son cada vez más utilizados dentro del campo industrial para el control de procesos, debido a las características que estos dispositivos poseen. Los sistemas de adquisición de datos ligados al trabajo de los controladores lógicos programables son sin lugar a dudas muy útiles, por lo que están siendo utilizados para aprovechar de mejor manera las características de una planta y lograr una mayor eficiencia en su producción.

- Debido a que la utilización de paquetes computacionales, disponibles en el mercado, dedicados a la adquisición de datos y supervisión, es en muchos casos económicamente prohibitivo, la alternativa se da en el desarrollo, por parte del propio usuario, de software específico a los requerimientos de su proceso. Esto de ninguna manera limita el crecimiento futuro del proceso; se trata más bien de una

alternativa económica que se adapta a las necesidades actuales y futuras del usuario.

- Se recomienda seguir desarrollando sistemas de adquisición de datos, muy útiles en el campo industrial, y sobre todo investigar más profundamente la potencialidad de la interface Allen-Bradley 1784-KR, de tal manera de conseguir estaciones de adquisición de datos más amigables y funcionales. La característica de la interface 1784-KR de interconexión en redes del PLC's deja abierto un gran campo de investigación y desarrollo para los sistemas de adquisición de datos, supervisión y control en redes de PLC's.

- La característica del control digital de permitir programar cualquier tipo de controlador, deja abierta la posibilidad de implementar otros tipos de controladores digitales que logren de manera más efectiva controlar una determinada planta.

BIBLIOGRAFIA

- Sánchez Ch. Gustavo, "APLICACION DEL CONTROLADOR PROGRAMABLE EN LA OPERACION DE UN MODELO PARA DEMOSTRACION DE PROTECCIONES EN SEP", tesis de grado, EPN-FIE, 1983.
- Ortega C. Edgar, "SISTEMA DE TRANSFERENCIA AUTOMATICO DE CARGA Y SINCRONIZACION PARA LA ESTACION COTOPAXI UTILIZANDO UN CONTROLADOR LOGICO PROGRAMABLE", tesis de grado, EPN-FIE, 1993.
- Cifuentes L. Juan, "ANALISIS TECNICO ECONOMICO Y POSIBILIDADES DE APLICACION DE LOS PLC's", tesis de grado, EPN-FIE, 1993.
- Molina Jorge, "APUNTES DE CONTROL INDUSTRIAL", EPN-FIE.
- Kuo B., "DIGITAL CONTROL SYSTEMS", ed. Van Valkenburg, 1980.
- Kuo B., "SISTEMAS AUTOMATICOS DE CONTROL", ed. Continental, México, 1986.
- Auslander D. - Takahashi Y. , "INTRODUCCION A SISTEMAS Y CONTROL", ed. McGraw-Hill, México, 1974.
- Valencia Javier, "ANALISIS Y DISEÑO DE SISTEMAS DE CONTROL DE TIEMPO DISCRETO ASISTIDO POR COMPUTADOR", tesis de grado, EPN-FIE, 1994.
- B. S. Bennet, "ANALYSIS AND DESIGN OF LINEAR SISO CONTROL SYSTEMS. SAMPLE-DATA CONTROL SYSTEMS", University of Manchester, Institute of Science and Technology, Manchester, 1981.
- Warwick K., Reeds D., "INDUSTRIAL DIGITAL CONTROL SYSTEMS", ed. Peregrinus, serie 29, England, 1988.
- Allen-Bradley, "SLC-500 MODULAR HARDWARE STYLE. INSTALLATION AND OPERATION MANUAL", 1993.
- Allen-Bradley, "SLC 500 ANALOG I/O MODULES. USER'S MANUAL", 1992.
- Allen-Bradley, "STANDARD DRIVER SOFTWARE Cat. No. 6001-F2E. USER'S MANUAL", 1989.
- Allen-Bradley, "PC DH-485 INTERFACE MODULE Cat. No. 1784-KR", 1990.

- Allen-Bradley, "SLC 500 ADVANCED PROGRAMMING SOFTWARE. Cat. No. 1747-PA2E", 1990.
- Bastidas J., "ESTUDIO TEORICO EXPERIMENTAL DE UN SERVOMECANISMO DE POSICION Y VELOCIDAD", tesis de grado, EPN-FIE, 1981.
- Barragán M., "EL LABORATORIO DE SISTEMAS DE CONTROL MOTOMATIC", EPN-FIE.
- Benítez Diego, "DISEÑO Y CONSTRUCCION DE UN SISTEMA DE CONTROL DE NIVEL DE LIQUIDOS", tesis de grado, EPN-FIE, 1994.
- Banda H., Flokes D., "PROGRAMMING IN C", Departament of Mathematics and Computer Science, University of Dundee, 1989.

ANEXOS

A. Manual de usuario de los programas
B. Códigos de error y su significado
C. Réplicas para el comando de lectura de diagnóstico
D. Características técnicas del módulo analógico A-B 1746-NIO4I
E. Listado de los programas del PLC
F Listado del programa ASC-PID.EXE

ANEXO A

MANUAL DE USUARIO DE LOS PROGRAMAS

A. MANUAL DE USUARIO DE LOS PROGRAMAS

A.1. PROGRAMA ASC-PID.EXE

La siguiente información constituye solamente una guía para ejecutar de manera adecuada el programa ASC-PID.EXE. Para mayor información remítase al capítulo 4 de este trabajo de tesis.

Si bien es cierto, el programa dispone de varias opciones que realizan procesos diversos, los pasos que posteriormente se detallan corresponden únicamente a la opción de supervisión de un control PID ejecutado por el PLC.

A.1.1 PROCEDIMIENTO PARA LA ADQUISICION DE DATOS, SUPERVISION Y CONTROL PID DE UN PROCESO DETERMINADO.

- Una vez determinado el proceso que se desea controlar, asegúrese que las señales proporcionadas desde el sensor y hacia el actuador del proceso estén dentro del rango de trabajo permitido (0 a 10 Vdc).
- Conecte correctamente las entradas y salidas del módulo analógico, siguiendo las recomendaciones descritas en la sección correspondiente a este dispositivo, al proceso a controlar.
- Cargue el programa PID502.ACH, PID_CONT.ACH o PID_DISC.ACH con el programa APS, dependiendo del método de diseño del controlador PID que se requiera utilizar:

  PID502.ACH: Control PID ejecutado a través de la instrucción PID que posee el PLC A-B SLC-500 en su conjunto de instrucciones básicas.

  PID_CONT.ACH: Control PID ejecutado en base a la ecuación en diferencias 3-32. Método de diseño en el plano "S" del controlador.

  PID_DISC.ACH: Control PID ejecutado evaluando la ecuación en diferencias 3-33. Método de diseño en el plano "Z" del controlador.
- Ubíquese bajo el directorio de trabajo C:\F2E.

- Ejecute el programa START.BAT para inicializar la interface Allen-Bradley 1784-KR.
- Ejecute el programa de adquisición de datos, supervisión y control PID, ASC-PID.EXE. Se presentará la siguiente pantalla.

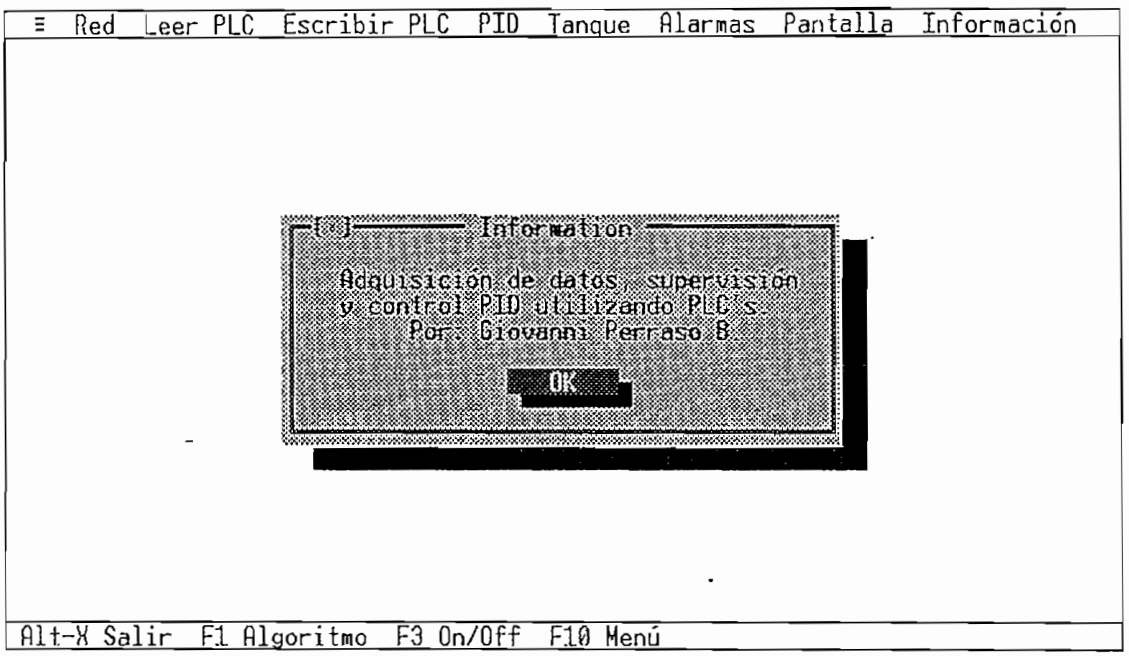


Fig. A-1: Presentación del programa ASC-PID.EXE.

- Una vez ingresado al programa, asegúrese que el número de nodo por defecto (opción RED) coincida con el nodo del PLC que se va a supervisar (por defecto es 1).
- Con ayuda del teclado o el mouse seleccione la opción ALGORITMO del menú principal.
- Si no existe algún inconveniente con la interface Allen-Bradley 1784-KR, el programa comenzará a adquirir las señales de entrada y salida del módulo analógico. En la parte superior derecha se podrá observar el modo de funcionamiento (Auto o Man) y el tipo de programa ejecutado por el PLC.

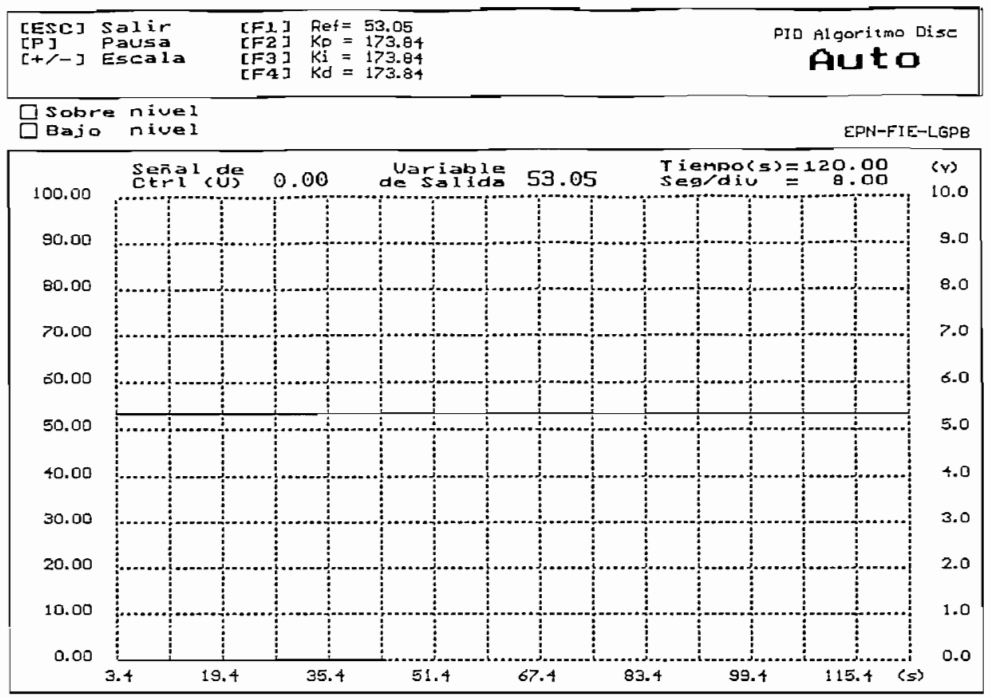


Fig. A-2: Pantalla principal del programa ASC-PID.EXE.

- Para cambiar de modo de funcionamiento, presionando la tecla ESC salga al menú principal y ejecute la opción ON/OFF. Esto cambiará el modo de manual a automático o viceversa.
- Ingrese nuevamente a la opción ALGORITMO. Ahora el programa está en modo automático.
- Cambie las constantes del controlador PID y su referencia presionando las teclas de función correspondientes mostradas en la parte superior de la pantalla.
- La computadora enviará estos datos al PLC y éste ejecutará el programa cargado inicialmente en su memoria, de acuerdo a los parámetros Kp, Ki, Kd y referencia ingresados.
- Para cambiar el rango de visualización presione las teclas "+" o "-". Esta opción hará un "zoom" en la pantalla gráfica.
- Unicamente en modo manual se encuentran habilitadas las teclas de cursor "↑" y "↓". Estas teclas cambiarán manualmente el valor de la señal de control (el PLC no controla el proceso en base al algoritmo programado).

- En cualquier instante del proceso de adquisición de datos, es posible presionar la tecla "P" para realizar una pausa en la visualización de los datos. Esto de ninguna manera suspenderá el trabajo del PLC, únicamente suspende el proceso de adquisición de datos, con el objetivo de analizar los datos adquiridos hasta ese momento. Dentro de esta opción, se puede presionar la tecla "G". El programa pide entonces, el nombre de un archivo en el que van a ser almacenados los datos que muestra el monitor.
- Si su proceso requiere de una escala distinta a la incluida por defecto (0 a 10), seleccione la opción PANTALLA del menú principal y proceda a cambiar los niveles máximo y mínimo de visualización en el eje Y.
- Para cambiar el tiempo de visualización de la respuesta, seleccione la opción "Tiempo" e introduzca el tiempo de visualización que requiera (máximo 150 segundos).
- Existe también una opción para fijar los niveles de alarma máximo y mínimo. Elija para esto, la opción ALARMA del menú principal. Actívela, desactívela o fije los niveles de alarma según su conveniencia. Cuando la variable de salida no se encuentre dentro de los límites de alarma, se producirá una señal visual y audible que indicará la situación al usuario.
- Como ayuda al usuario, existe la opción "AYUDA", que de manera breve da una visión general del programa. En la opción Gráfico del proceso se presenta la siguiente pantalla.

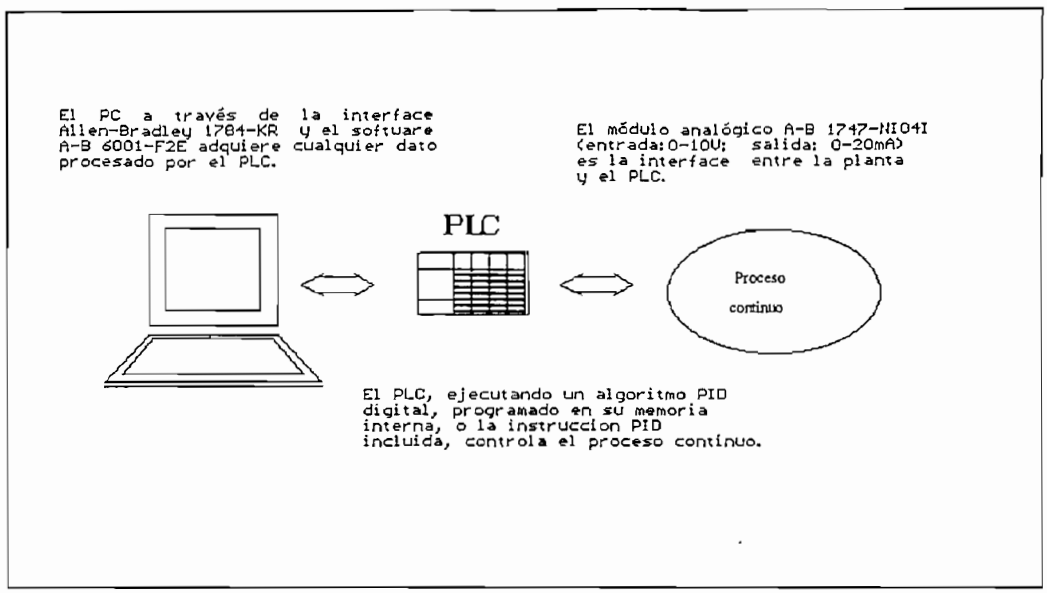


Fig. A-3: Opción "Gráfico del proceso".

A.2. PROGRAMAS DEL PLC

Los programas del PLC (PID_CONT.ACH, PID_DISC.ACH o PID502.ACH), para control PID de un proceso, pueden ser también manejados con el programa APS (Advanced Programming Software). A continuación algunas indicaciones de su forma de trabajo.

A.2.1 PID_CONT.ACH

- Tiene implementado un algoritmo de control PID diseñado en el plano "S".
- En el archivo de programa número 2 se encuentra el programa principal.
- El archivo de programa número 3 es una subrutina STI que contiene el algoritmo de control PID,
- El periodo de muestreo se fija en la palabra 30 del status (S:30).

- N7:1 Entrada del módulo analógico.
- N7:2 Valor de referencia.
- N7:3 Constante Kp multiplicada por 100.
- N7:4 Constante Ki multiplicada por 100.
- N7:5 Constante Kd multiplicada por 100.
- N7:40 Bandera de funcionamiento Manual-Automático. En modo manual, se debe ingresar el valor de salida del módulo analógico en la palabra N7:30.
- N7:41 Bandera que identifica el programa.

A.2.2 PID_DISC.ACH

- Tiene programado un algoritmo de control PID diseñado en el plano "Z".
- El algoritmo de control PID se incluye dentro del programa principal no se encuentra dentro de alguna subrutina.
- Posee los mismas características que el programa anterior, excepto para el periodo de muestreo. Este dato, debe ser cambiado alterando la línea 1 del programa (instrucción XIO S:4/0). Los valores permitidos constan el tabla 3-6.

A.2.3 PID502.ACH

- Programa que presenta un control PID utilizando la instrucción PID que incluye el PLC en su conjunto de instrucciones básicas.
- La instrucción PID ha sido programada en una subrutina de interrupción STI.
- Para más información refiérase al capítulo 3.

ANEXO B

CODIGOS DE ERROR Y SU SIGNIFICADO

Error Codes

Error Codes

This appendix contains local and remote error codes:

Table B.A
Local Error Codes

Error Codes (in Hex)	Message
1	Successful transmission
18	LLC Interface not initialized; SSAP already used or invalid SSAP
19	LLC Interface not initialized; KTLLC driver not installed
1A	Unsuccessful send request; SSAP not active or illegal
1B	Unsuccessful send request; Invalid packet size
1C	Unsuccessful send request; Host cannot access dual port
1D	Reply timeout; No reply data received
1E	Unsuccessful send request; Buffer not available
1F	DH-485 Interface not initialized
23	Bad function parameter
25	Invalid Channel
27	KR transmit timeout
28	Fatal solicited buffer memory allocation
29	Fatal timer buffer memory allocation
2A	Fatal timer buffer memory corrupted
2B	KR channel already open

Appendix
Error Codes

Table B.B
Remote Error Codes

Error Codes (in Hex)	Message
STS	
00	Success
10	Command format incorrect
50	Address problem
60	Disallowed due to command protection
F0	Extended status
EXT STS	
0B	Access denied. Improper privilege
1A	File already open
1B	Procesor in edit mode. Not accessible

ANEXO C

REPLICAS PARA EL COMANDO DE LECTURA DE ESTADO

Diagnostic Replies

Diagnostic Status Reply

The tables below show diagnostic status replies for the following:

- SLC-5/01 (Table C.A)
- APS Software (Table C.B)
- 1784-KR Interface Module (Table C.C)

Table C.A
Diagnostic Status Replies for the SLC-5/01

Byte	Description	Status Reply
1	Mode/Status	00 (Hex)
2	Type Extender	EE (Hex)
3	Extended Interface Type	1F(Hex)
4	Extended Processor Type – for rack type 1747-L51 – for 20-40 I/O	 18 1A
5	Series/Revision – Bits 0-4 – Bits 5-7	 0 (Release 1) 1 (Release 2), etc. 0 (Series A) 1 (Series B), etc.
6-16	Bulletin Number/Name 1747-LP11&LP14 1747-L20 1747-L30 1747-L40	(in ASCII) 5/01 500-20 500-30 500-40
Product Specific Information		
17	Major Error Code Word	(low byte)
18	Major Error Code Word	(high byte)

Byte	Description	Status Reply
19	Processor Mode Status/Control Word – Bit 0-4 mode: – Bit 5 – Bit 6 – Bit 7	 0 – Download 1 – Program 2 – Reserved 3 – Idle due to SUS instruction 4 – Reserved 5 – Reserved 6 – Run 7 – Test Continuous Scan 8 – Test Single Scan 9 – 31 Reserved Forces Active Forces Installed Communication Active
20	Processor Mode Status/Control Word (High Byte) Bit 0 Bit 2 Bit 5	 Protection Power Loss Load Memory Module on Mem Error Major Error – Process halted
21	Program ID	(low byte)
22	Program ID	(high byte)
23	Processor RAM size (in Kbytes)	
24	Bit 0 Bits 2-7:	Directory File Corrupted 00-1F (Program Owner Node Address) 3F (No Program Owner)

Table C.B
Diagnostic Status Reply for APS Software (with COM1 Port DH-485 connection)

Byte	Description	Status Reply (Hex)
1	Mode/Status Byte	00 (no modes)
2	Type Extender	EE
3	Extended Interface Type	20
4	Extended Processor Type	1B
5	Series/Revision: Bits 0-4 Bits 5-7	 0 (Release 1) 1 (Release 2). etc. 0 (Series A) 1 (Series B), etc.
6-16	Bulletin Number/Name= APS	(ASCII)
17-24	Product Specific Information	00

Table C.C
Diagnostic Status Reply for 1784-KR

Byte	Description	Status Reply (Hex)
1	Mode/Status Byte	00 (no modes)
2	Type Extender	FE
3	Extended Interface type	24
4	Not used	00
5	Series/Revision Bits 0-4: Bits 5-7:	 0 (Release 1) 1 (Release 2). etc. 0 (Series A) 1 (Series B), etc.
6-16	Bulletin Number/Name = "1784-KR"	ASCII
17-24	Product Specific Information	00

Diagnostic Read Reply (Diagnostic Counters)

The table below contains diagnostic read reply values for:

- SLC-500
- APS (COM1)
- 1784-KR Interface Module

Table C.D
Diagnostic Read Reply Values for SLC-500, APS, and 1784-KR

This Byte	Means
0	Total packets received, low byte
1	Total packets received, high byte
2	Total messagaes sent, low byte
3	Total messages sent, high byte
4	Retries
5	Retry limit exceeded
6	NAK, no memory sent
7	NAK, no memory received
8	Bad messages received
9	Reserved

ANEXO D

CARACTERISTICAS TECNICAS DEL MODULO ANALOGICO A-B 1746-NIO4I

Analog Module Specifications

This section lists the specifications for the 1746–NI4, NIO4I, NIO4V, NO4 and NO4V analog modules. They include:

- General specifications
- Current and Voltage input specifications
- Current and Voltage output specifications

General Specifications for NI4, NIO4I, NIO4V, NO4I, and NO4V

Description	Specification
SLC Communication Formal	16 Bit Two's Complement Binary
Field Wiring to Backplane Isolation	500 V dc
Update Time	512 µs for all channels in parallel
Recommended Cable	Shielded Belden #8761
Maximum Wire Size	#14 AWG
Terminal Block	Removable
Location	1746 rack
Calibration	Factory Calibrated
Noise Immunity	NEMA Standard ICS 2–230
Environmental Conditions Operating Temperature Storage Temperature Relative Humidity	 0° to +60° C (+32° to +140° F) –40° to +85° C (–40° to +185° F) 5 to 95% (non–condensing)

Catalog 1746–	Input Channels per Module	Output Channels per Module	Backplane Current Draw 5V (max.)	Backplane Current Draw 24V (max.)	External 24 V dc Po Supply Toleranc
NI4	4 differential, voltage or current selectable per channel	NA	25mA	85mA	NA
NIO4I	2 differential, voltage or current selectable per channel	2 current outputs, not individually isolated	55mA	145mA	NA
NIO4V	2 differential, voltage or current selectable per channel	2 voltage outputs, not individually isolated	55mA	115mA	NA
NO4I	NA	4 current outputs, not individually isolated	55mA	195mA	24 ±10% at 195mA (21.6 to 26.4 V dc
NO4V	NA	4 voltage outputs, not individually isolated	55mA	145mA	24 ±10% at 145mA (21.6 to 26.4 V dc

Φ *Required for some applications.*

General Specifications for NI4, NIO4I, and NIO4V

Description	Specification
Converter Resolution	16 bit
Repeatability	±1 LSB
Location of LSB in I/O image word	0000 0000 0000 0001
Non-linearity	0.01%
Common Mode Voltage Range	–20 to +20 volts
Common Mode Rejection at 0 to 10 Hz (min.)	50dB
Common Mode Rejection at 60 Hz (min.)	105dB
Normal Mode Rejection at 60 Hz (min.)	55dB
Channel Bandwidth	10 Hz
Step Response	60msec at 95%
Voltage Input Coding (–10 to +10VDC – 1 LSB)	–32,768 to +32,767
Current Input Coding –20 to +20mA	–16,384 to +16,384
Conversion Method	Delta–Sigma Modulation
Impedance to ANL COM	500 Kohms
Impedance channel to channel	1 Mohms

Current Input Specifications for NI4, NIO4I, and NIO4V

Description	Specification
Input Range (Normal Operation)	–20 to +20 mA
Absolute Maximum Input Current	–30 to +30 mA
Absolute Maximum Input Voltage	±7.5VDC or 7.5 volts AC RMS
Input Impedance	250 Ohms
Resolution	1.22070 µA per LSB
Full Scale	20 mA
Overall Accuracy at +25° C (77° F) (max.)	±0.365% of full scale
Overall Accuracy at 0° to +60° C (32° to 140° F) (max.)	±0.642% of full scale
Overall Accuracy Drift (max.)	±79ppm/° C of full scale
Gain Error at +25° C (77° F) (max.)	±0.323%
Gain Error at 0° to +60° C (32° to 140° F) (max.)	±0.556%
Gain Error Drift (max.)	±67ppm/° C
Offset Error at +25° C (77° F) (max.) (Iin = 0, Vcm = 0)	±7 LSB
Offset Error at 0° to +60° C (32° to 140° F) (max.) (Iin = 0, Vcm = 0)	±14 LSB
Offset Error Drift (max.) (Iin = 0, Vcm = 0)	±0.20 LSB/° C

Voltage Input Specifications for NI4, NIO4I, and NIO4V

Description	Specification
Input Range	–10 to +10 V dc – 1 LSB
Input Impedance	1 Mohms
Resolution	305.176µV per LSB
Full Scale	10 V dc
Overall Accuracy at +25° C (77° F) (max.)	±0.284% of full scale
Overall Accuracy at 0° to +60° C (32° to 140° F) (max.)	±0.504% of full scale
Overall Accuracy Drift (max.)	±63ppm/° C of full scale
Gain Error at +25° C (77° F) (max.)	±0.263%
Gain Error at 0° to +60° C (32° to 140° F) (max.)	±0.461%
Gain Error Drift (max.)	±57ppm/° C
Overvoltage Protection (max. across IN+ to IN– terminals)	either 220 volts AC RMS continuously or 220 volts DC continuously
Offset Error at +25° C (77° F) (max.)	±7 LSB
Offset Error at 0° to +60° C (32° to 140° F) (max.)	±14 LSB
Offset Error Drift (max.)	±0.20 LSB/° C

Current Output Specifications for NIO4I and NO4I

Description	Specification
Converter Resolution	14 bit
Location of LSB in I/O image word	0000 0000 0000 01XX
Non-linearity	0.05%
Conversion Method	R-2R Ladder
Step Response	2.5 ms (at 95%)
Load Range	0 to 500 Ohms
Maximum Load Reactance	100µH
Current Output Coding (0 to +21mA – 1 LSB)	0 to +32764

Output Range (Normal)	0 to +20 mA
Overrange Capability	5% (0 to +21mA – 1 LSB)
Resolution	2.56348µA per LSB
Full Scale	21mA
Overall Accuracy at +25° C (77° F) (max.)	±0.298% of full scale
Overall Accuracy at 0° to +60° C (32° to 140° F) (max.)	±0.541% of full scale
Overall Accuracy Drift (max.)	±70ppm/° C of full scale
Gain Error at +25° C (77° F) (max.)	±0.298%
Gain Error at 0° to +60° C (32° to 140° F) (max.)	±0.516%
Gain Error Drift (max.)	±62ppm/° C
Offset Error at +25° C (77° F) (max.)	±10 LSB
Offset Error at 0° to +60° C (32° to 140° F) (max.)	±12 LSB

ANEXO E

PROGRAMAS DEL PLC

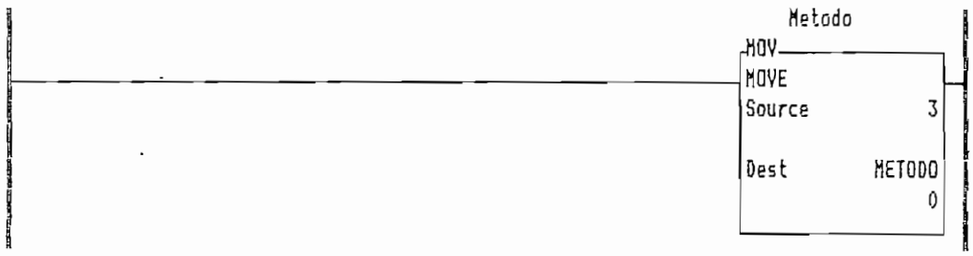
Allen-Bradley Co.
1747 Series Software
APS Release 4.01
Documentation Utility
Program Listing

PID502.ACH

Processor File: PID502.ACH
November 08, 1995 - 22:29

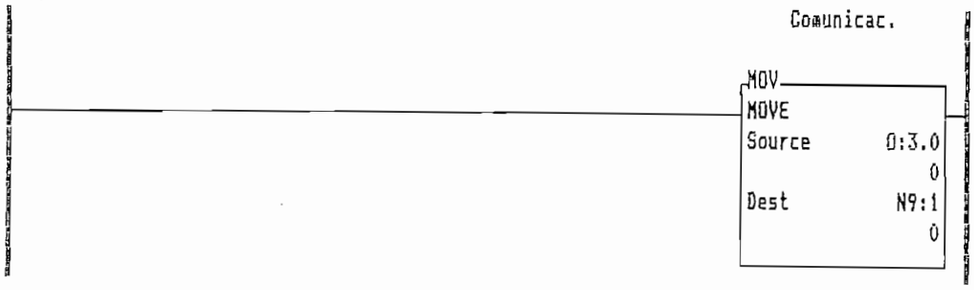
Rung 2:0

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                Adquisición de datos, supervisión y control PID              !
!                Control PID utilizando la instrucción PID del PLC            !
!                Por: Luis Giovanni Perraso Basantes                          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

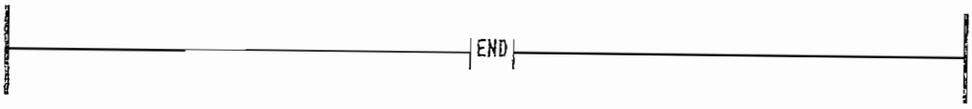


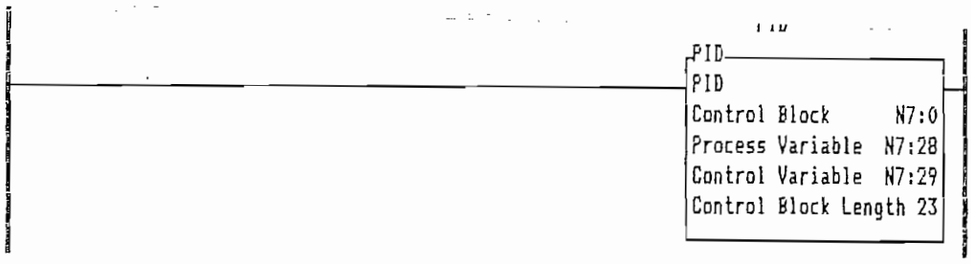
Rung 2:1

Rung 2:2

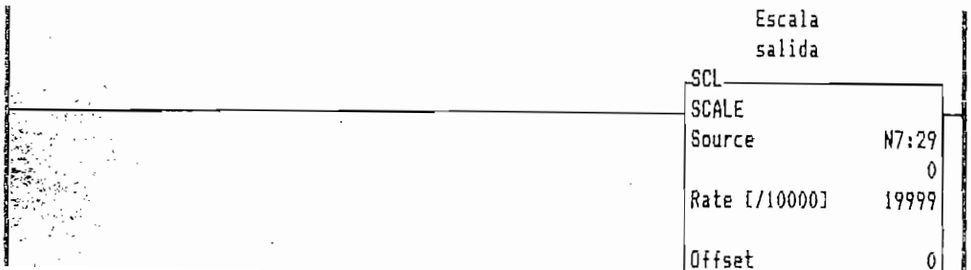


Rung 2:3

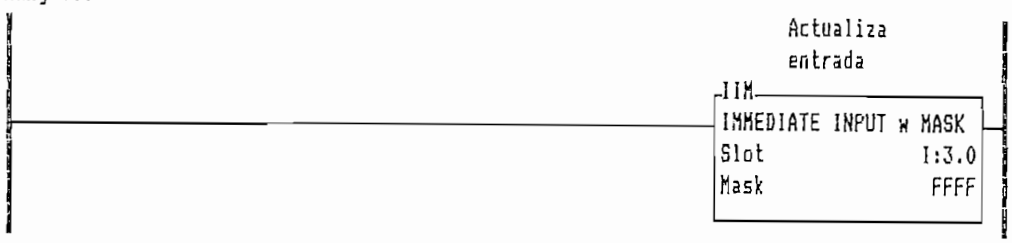




Rung 3:3

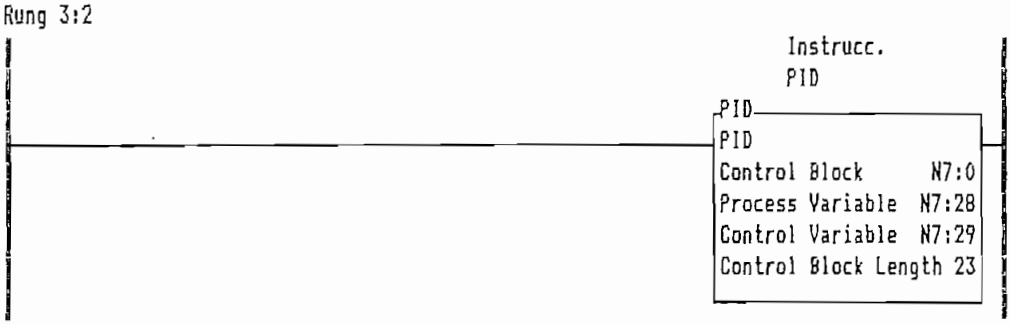


Rung 3:0

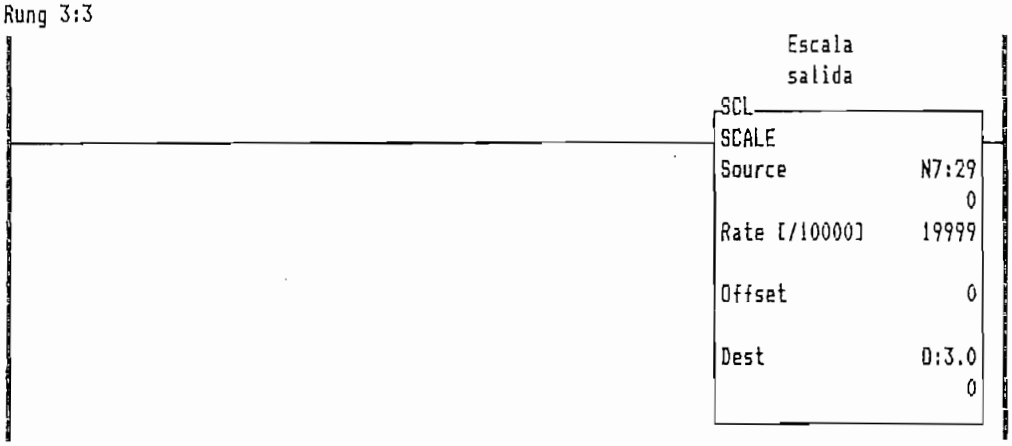


Rung 3:1

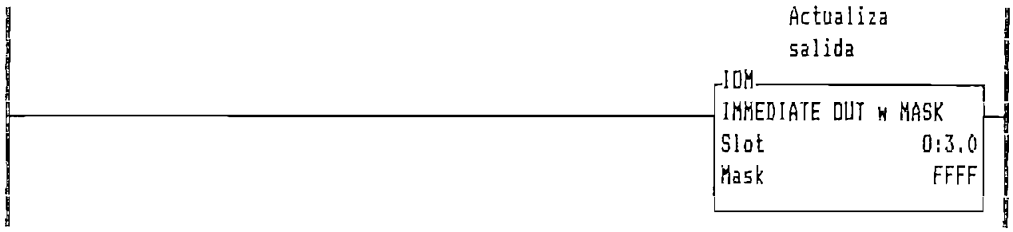
Rung 3:2



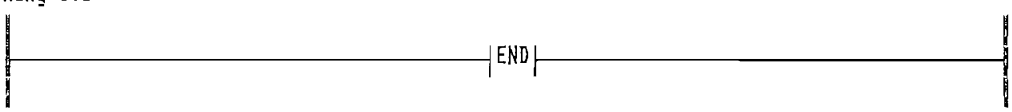
Rung 3:3



Rung 3:4



Rung 3:5

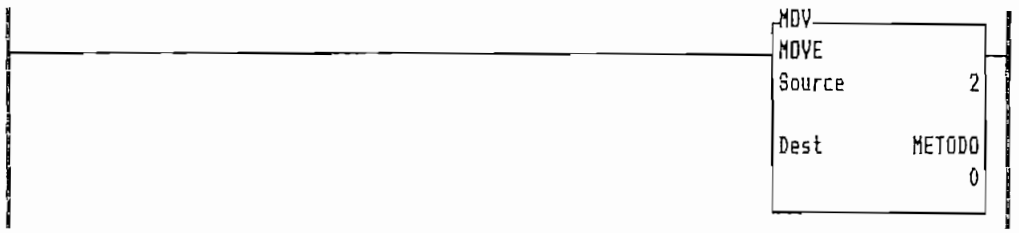


Allen-Bradley Co.
1747 Series Software
APS Release 4.01
Documentation Utility
Program Listing

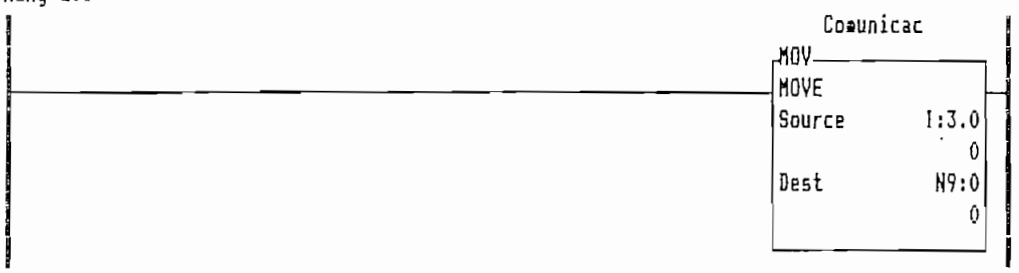
PID_CONT.ACH

Processor File: PID_CONT.ACH
November 08, 1995 - 22:06

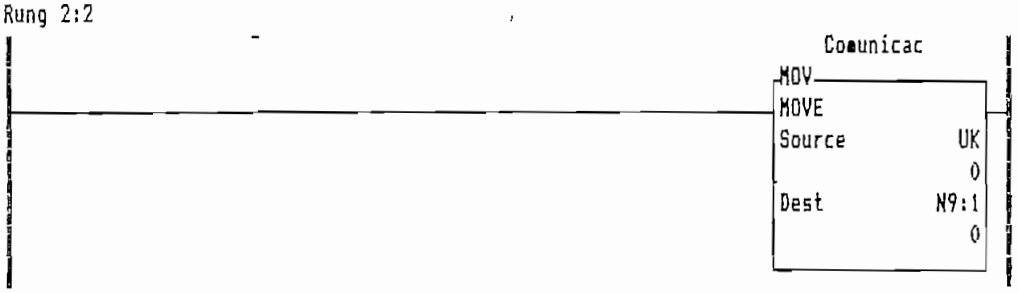
Rung 2:0



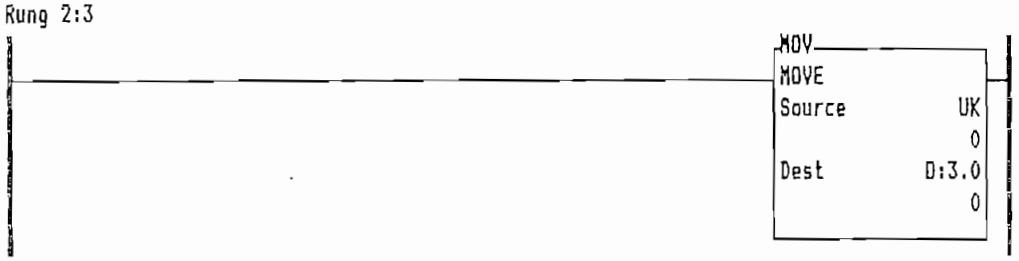
Rung 2:1



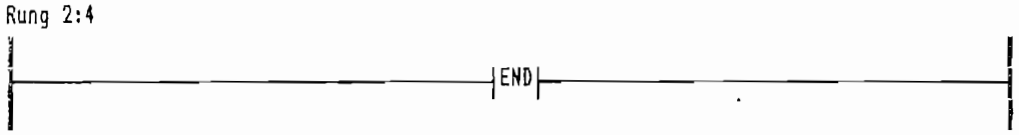
Rung 2:2



Rung 2:3



Rung 2:4



Rung 3:0

```
******************************************************************************
*            Adquisición de datos, supervisión y control PID                 *
*            Algoritmo de control PID. Diseño en plano "S"                   *
*            Por: Luis Giovanni Perraso Basantes                             *
******************************************************************************
|     Auto/Man                                                              |
|                                                                        X  |
| EQU                                                                    B3 |
|-EQUAL                 |-----------------------------------------------( )-|
| Source A      HABIL   |                                                0  |
|                   0   |                                                   |
| Source B          1   |                                                   |
```

Rung 3:1

```
|                                                     Actualiza             |
|                                                     archivo de            |
|                                                     entrada               |
|   X                                                                       |
|   B3                                              IIM                     |
|---] [----------------------------------------------IMMEDIATE INPUT w MASK-|
|    0                                              Slot            I:3.0   |
|                                                   Mask            FFFF    |
```

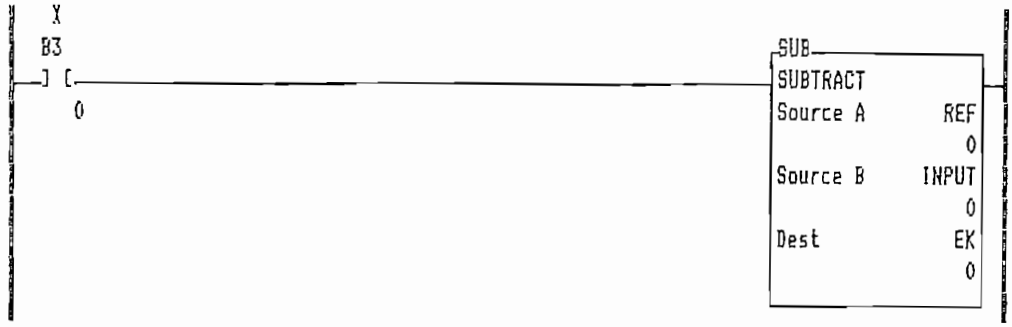
Rung 3:2

```
|                                                       Comunicac           |
|   X                                                                       |
|   B3                                                  MOV                 |
|---] [-------------------------------------------------MOVE              --|
|    0                                                  Source      I:3.0   |
|                                                                       0   |
|                                                       Dest        N9:0    |
|                                                                       0   |
```

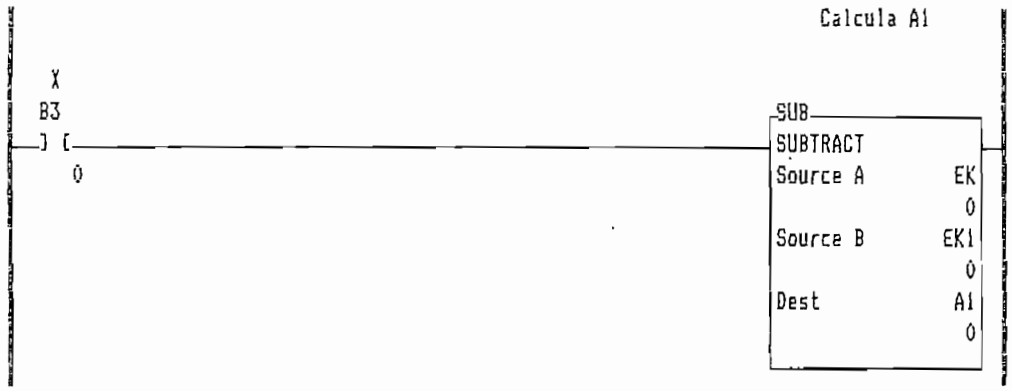
Rung 3:3

```
|                                                       Solo para           |
|                                                       visualizar          |
|   X                                                                       |
|   B3                                                  MOV                 |
|---] [-------------------------------------------------MOVE              --|
|    0                                                  Source      I:3.0   |
|                                                                       0   |
|                                                       Dest        INPUT   |
|                                                                       0   |
```

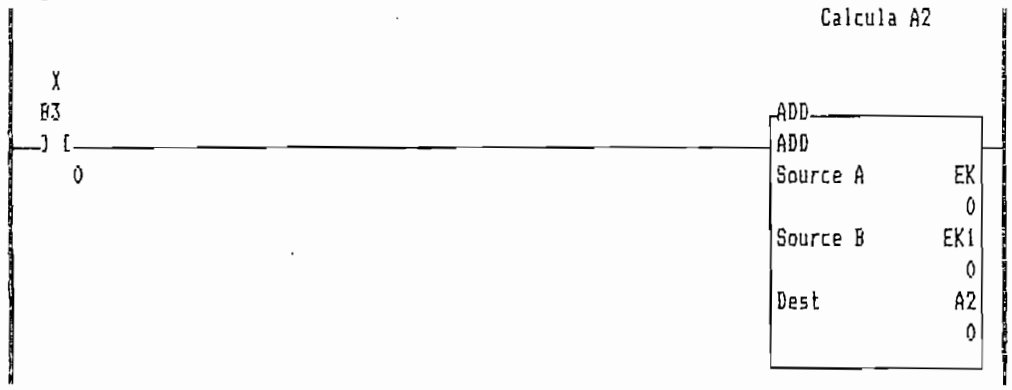
Rung 3:4



Rung 3:5

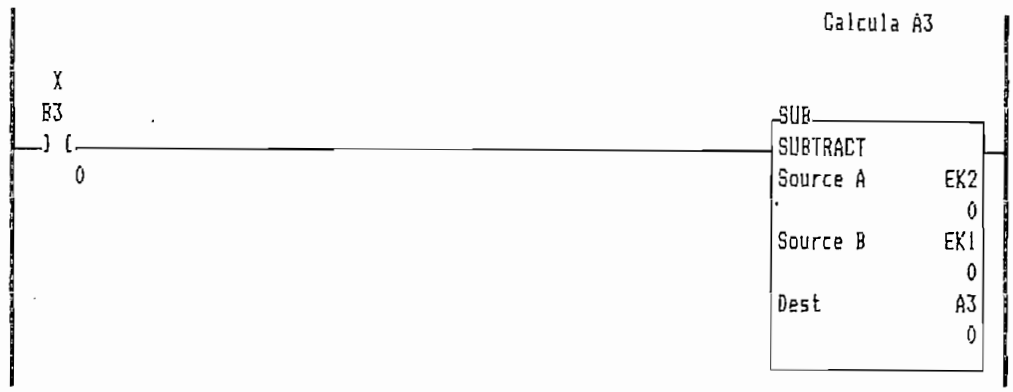


Rung 3:6

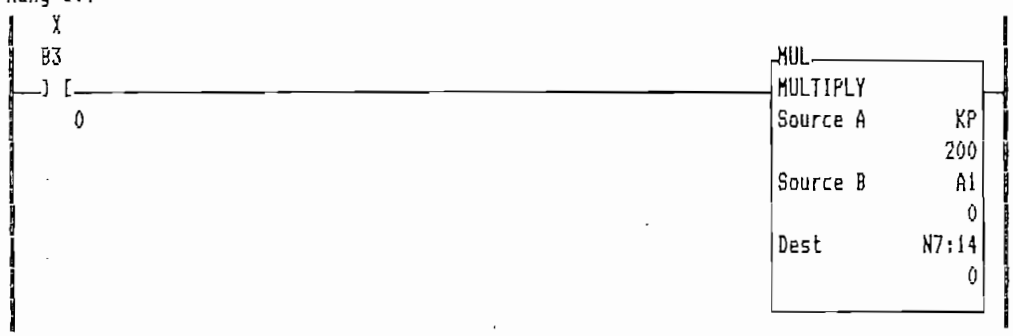


Rung 3:7

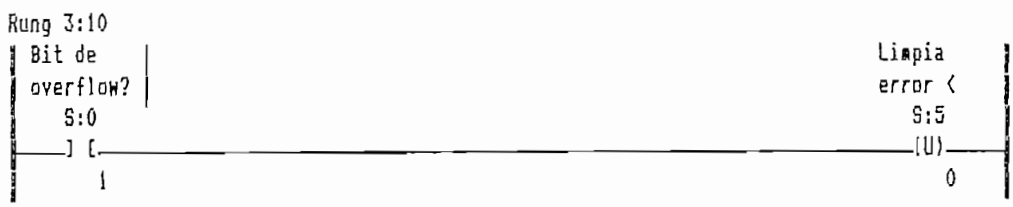
Rung 3:8



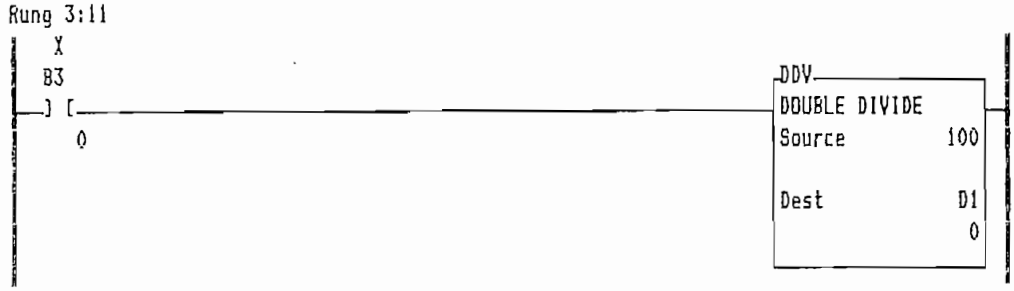
Rung 3:9



Rung 3:10

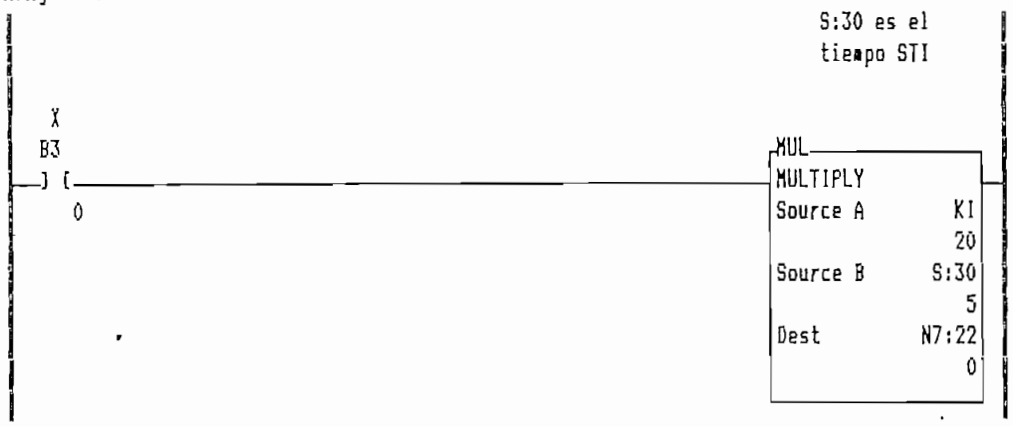


Rung 3:11



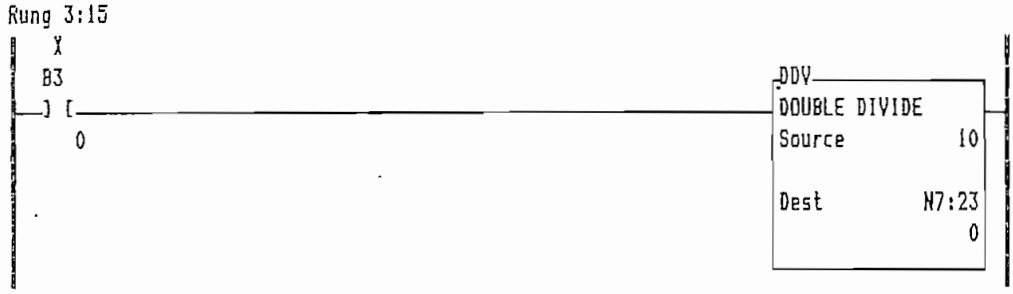
Rung 3:12

Rung 3:13

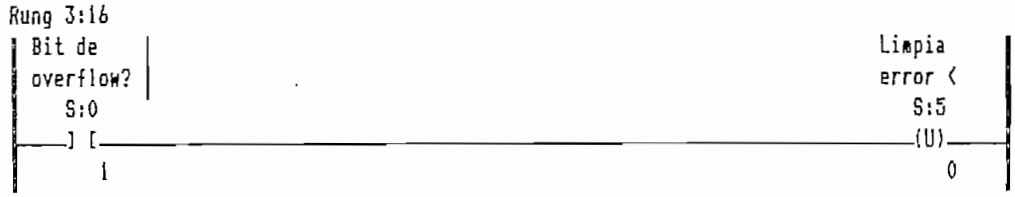


Rung 3:14

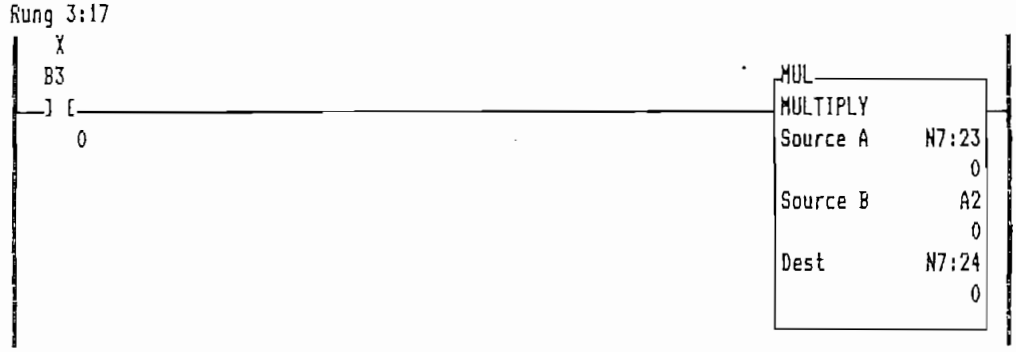
Rung 3:15



Rung 3:16

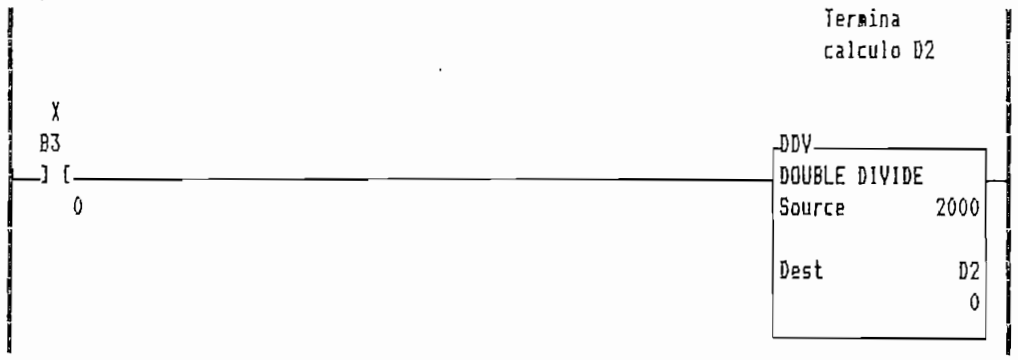


Rung 3:17

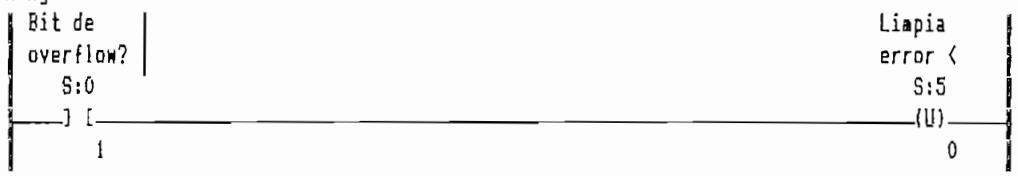


Rung 3:18

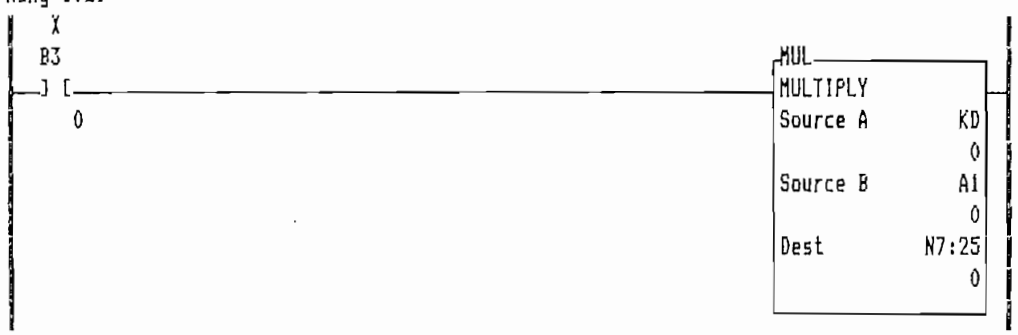
Rung 3:19



Rung 3:20

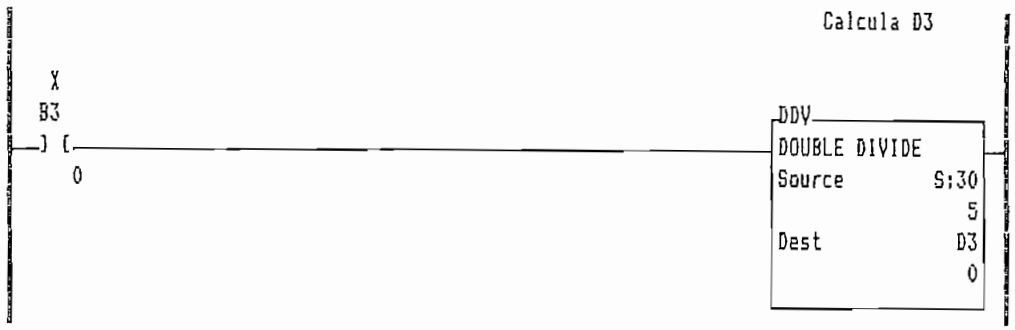


Rung 3:21

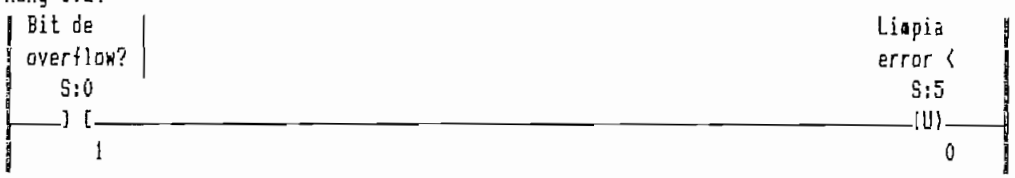


Rung 3:22

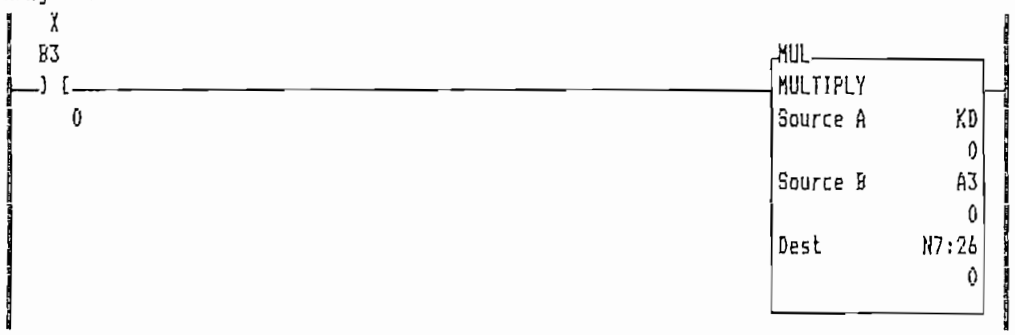
Rung 3:23



Rung 3:24

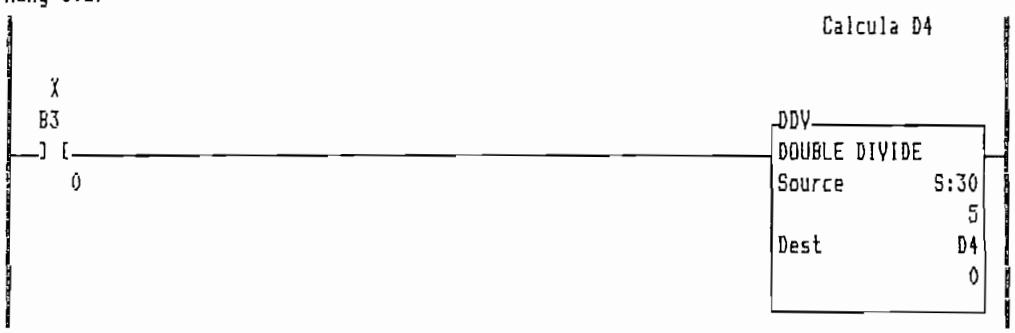


Rung 3:25



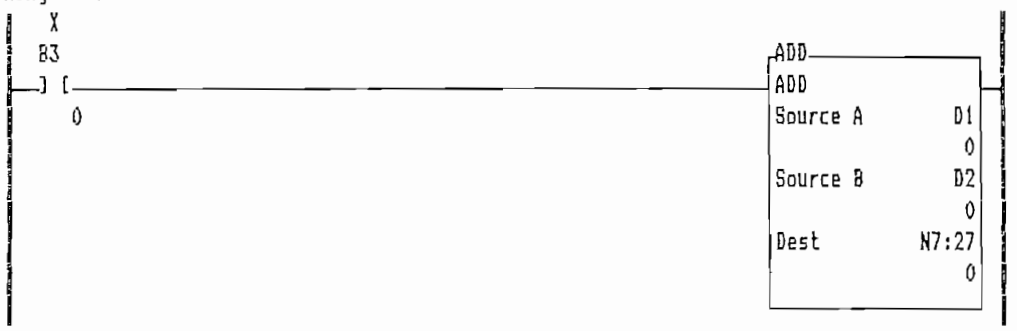
Rung 3:26

Rung 3:27



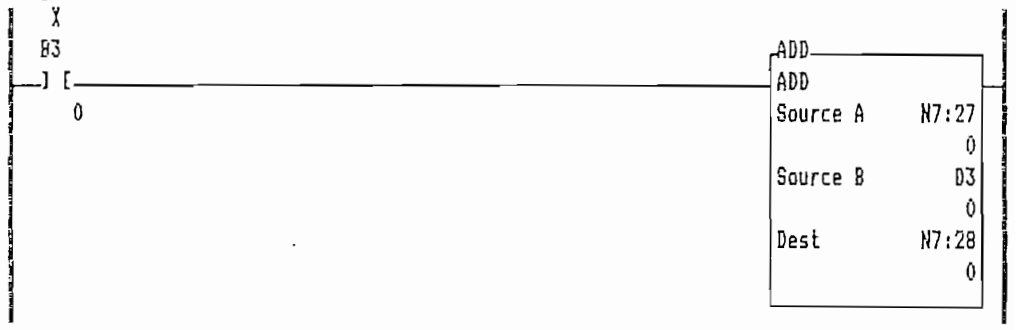
Rung 3:28

Rung 3:29



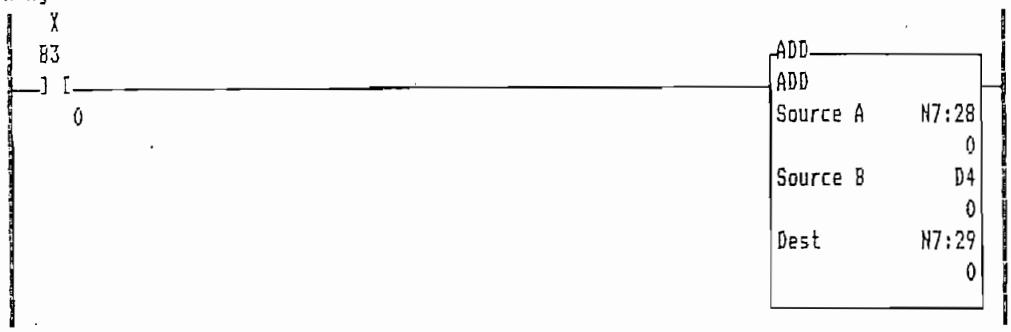
Rung 3:30

Rung 3:31



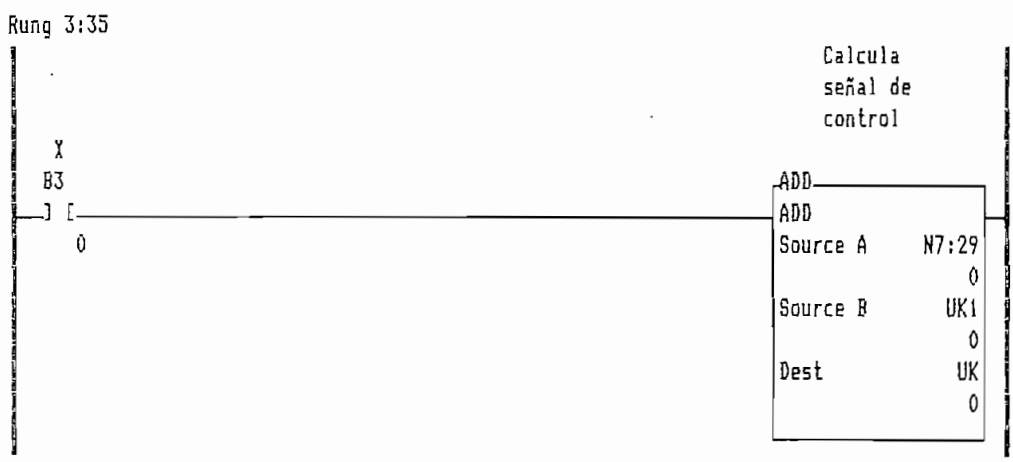
Rung 3:32

Rung 3:33



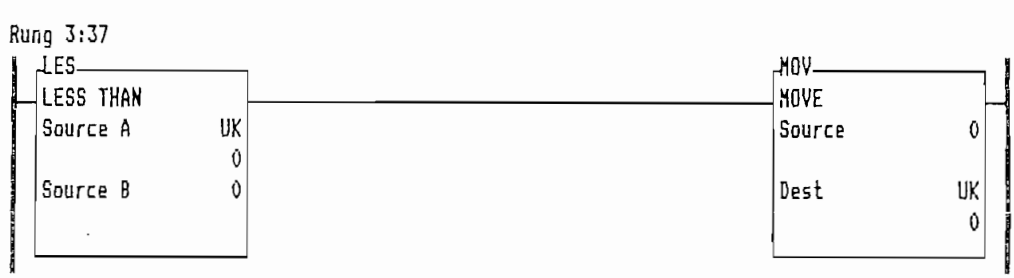
Rung 3:34

Rung 3:35

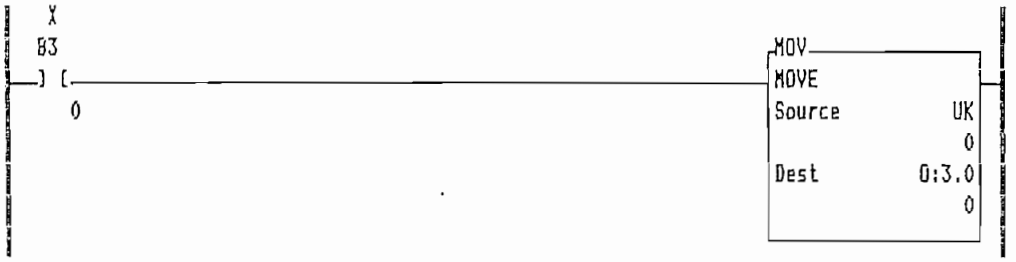


Rung 3:36

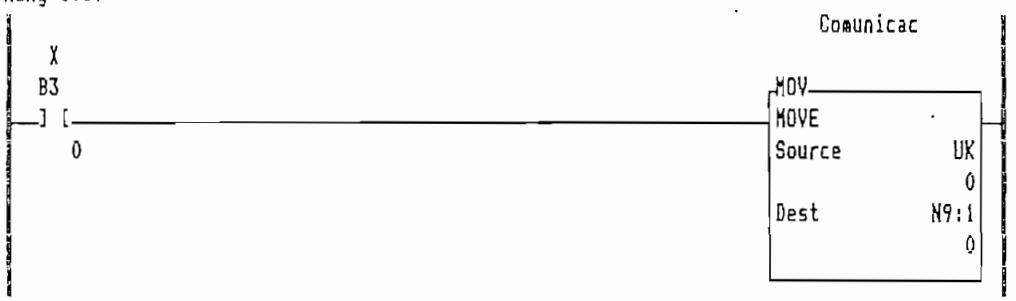
Rung 3:37



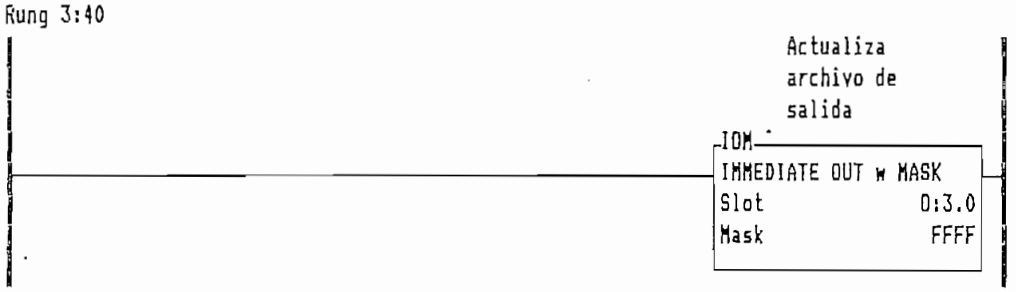
Rung 3:38



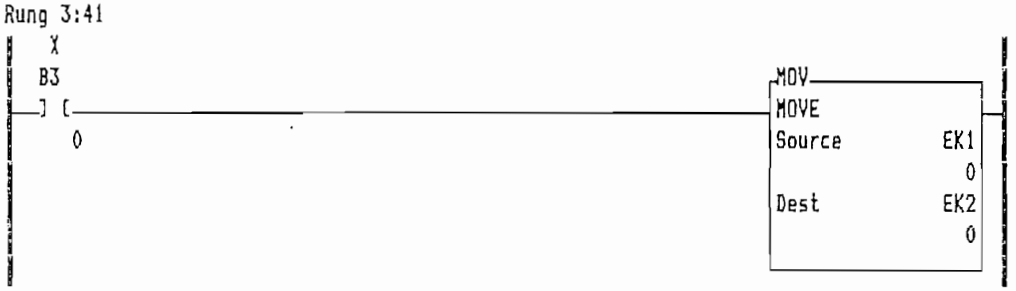
Rung 3:39



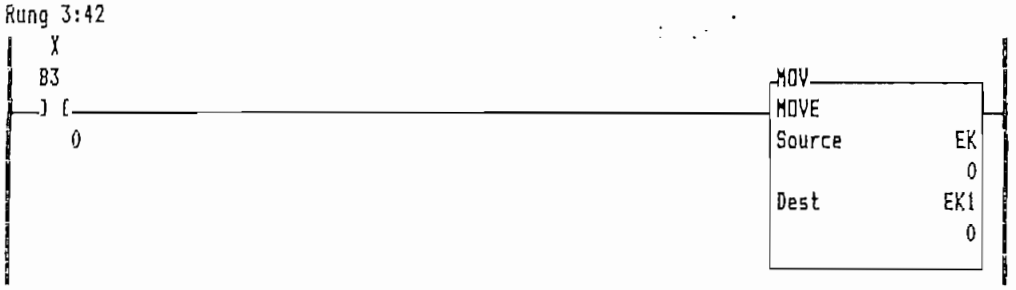
Rung 3:40



Rung 3:41

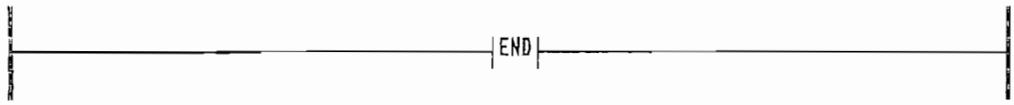


Rung 3:42



Rung 3:43

Rung 3:44



Allen-Bradley Co.
1747 Series Software
APS Release 4.01
Documentation Utility
Program Listing

PID_DISC.ACH

Processor File: PID_DISC.ACH
November 08, 1995 - 21:28

Rung 2:0

```
*************************************************************************
*        Sistema de adquisicion de datos, supervision y control PID     *
*              Algoritmo de control PID. Diseño en plano "Z"            *
*              Por: Luis Giovanni Ferraso Basantes                      *
*************************************************************************
|      Auto/Man        Reloj                                             |
|                                                                   X    |
| EQU                  S:4      B3                                  B3   |
|-EQUAL              |---] [------[OSR]---------------------------( )---|
| Source A     HABIL     3         1                                0    |
|                  0                                                     |
| Source B         1                                                     |
```

Rung 2:1

```
|                                                   Metodo              |
|                                                   discreto            |
|                                                  MOV                  |
|--------------------------------------------------MOVE-----------------|
|                                                  Source            1  |
|                                                  Dest         METODO  |
|                                                                    0  |
```

Rung 2:2

```
|                                          IIM                          |
|------------------------------------------IMMEDIATE INPUT w MASK-------|
|                                          Slot              I:3.0      |
|                                          Mask               FFFF      |
```

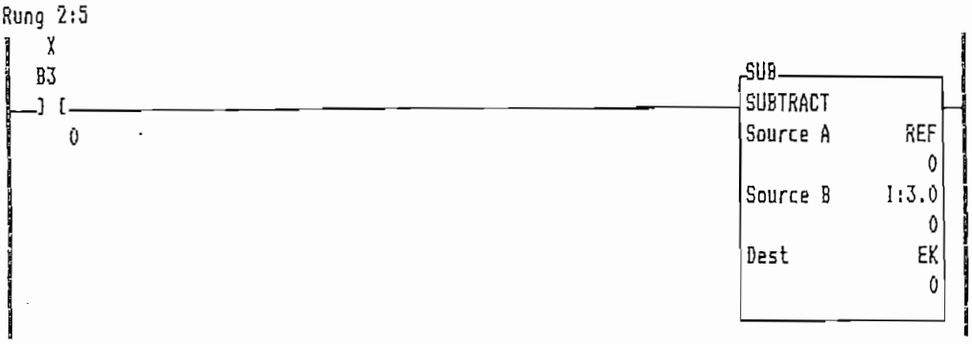
Rung 2:3

```
|                                                  Comunicac.           |
|                                                  F2E                  |
|                                                  MOV                  |
|--------------------------------------------------MOVE-----------------|
|                                                  Source         I:3.0 |
|                                                                    0  |
|                                                  Dest           N9:0  |
|                                                                    0  |
```

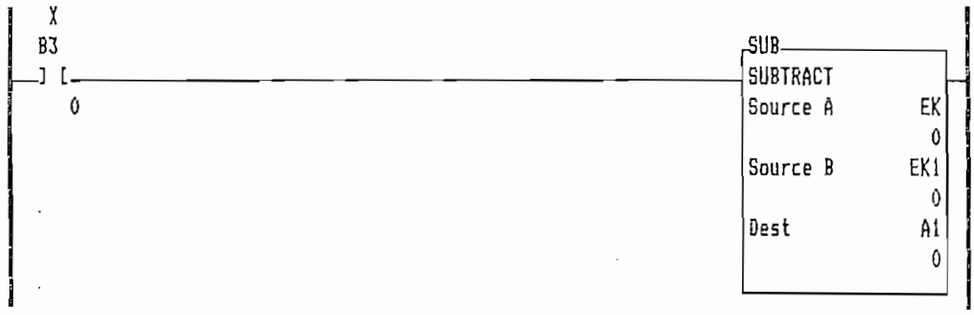
Rung 2:4

```
|                                                  MOV                  |
|--------------------------------------------------MOVE-----------------|
|                                                  Source         I:3.0 |
|                                                                    0  |
|                                                  Dest          INPUT  |
|                                                                    0  |
```

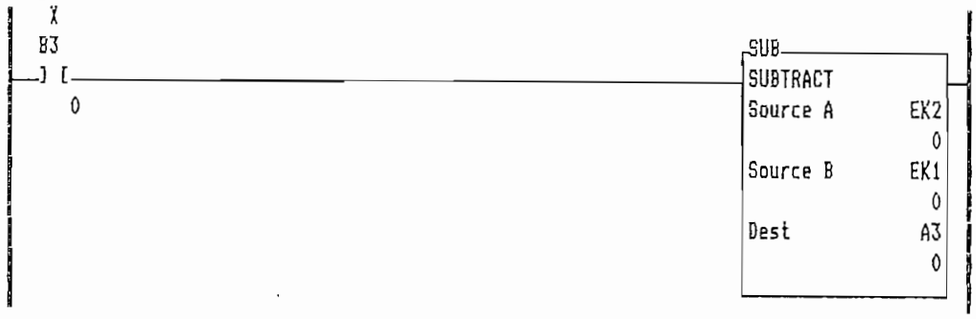
Rung 2:5



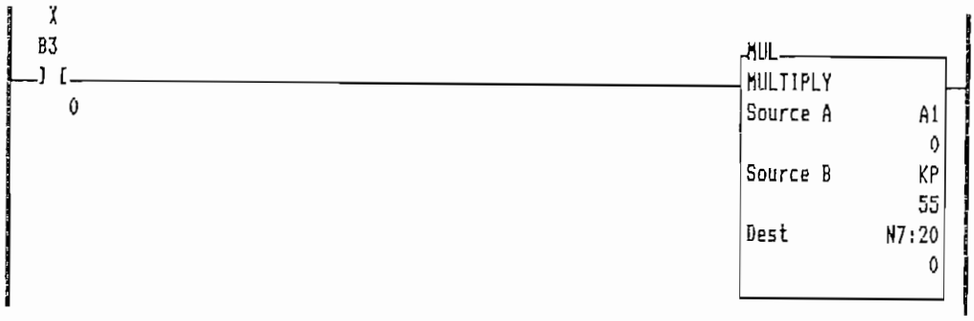
Rung 2:6



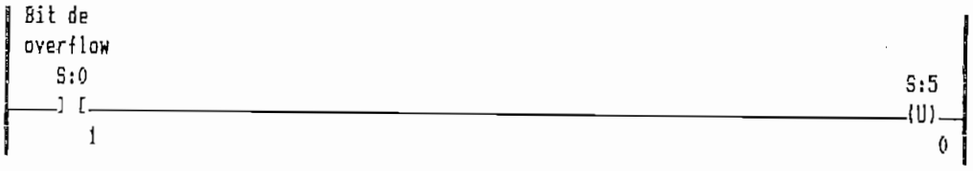
Rung 2:7



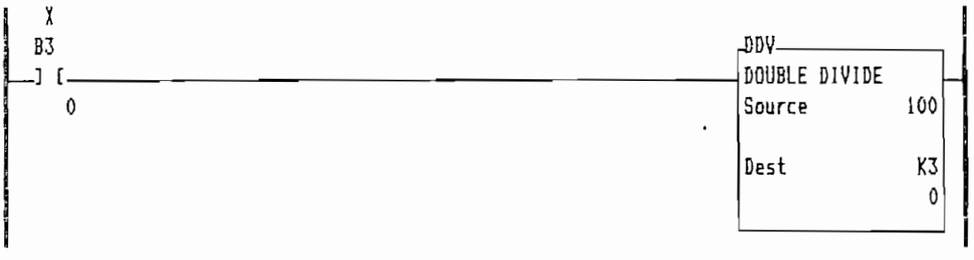
Rung 2:8



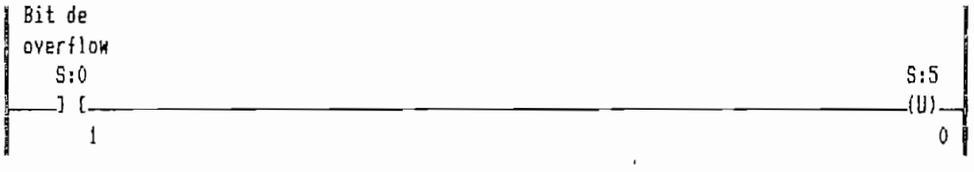
Rung 2:9



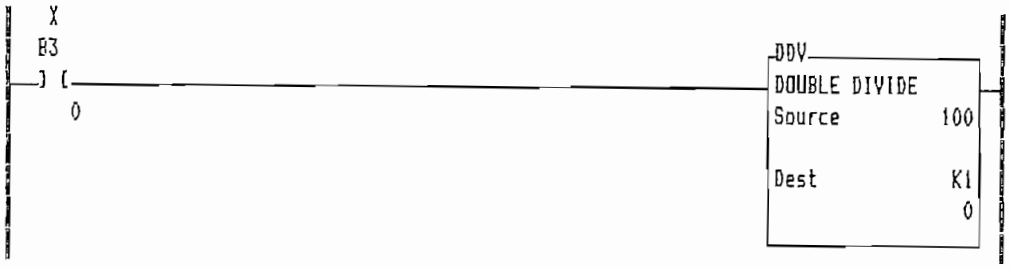
Rung 2:14



Rung 2:15

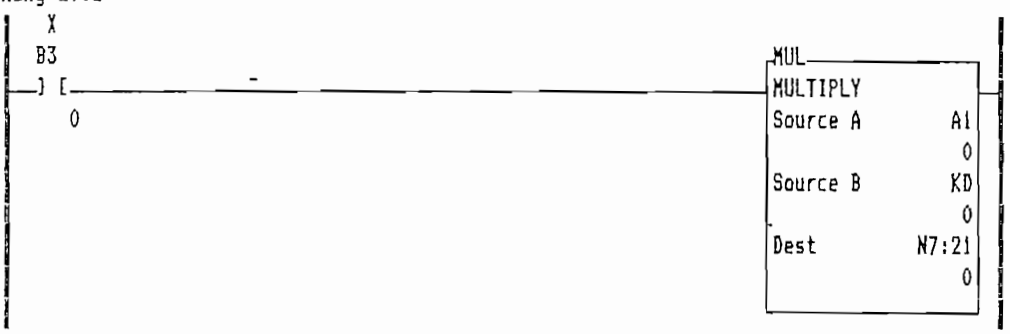


Rung 2:10

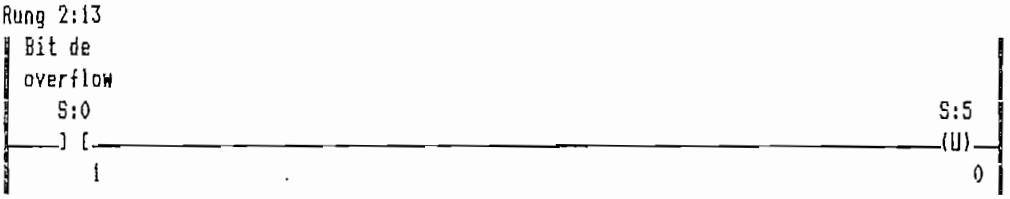


Rung 2:11

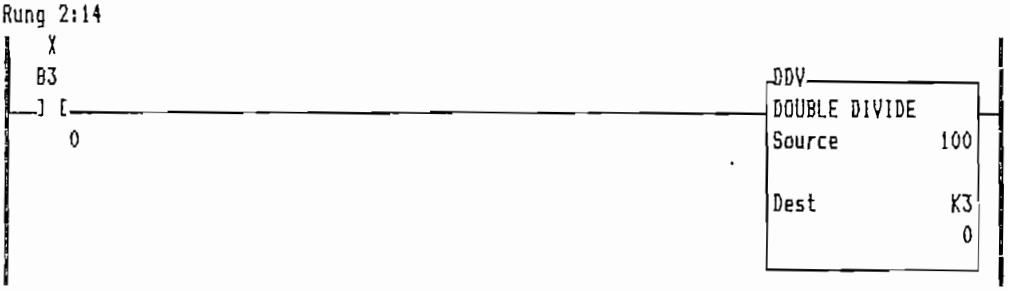
Rung 2:12



Rung 2:13

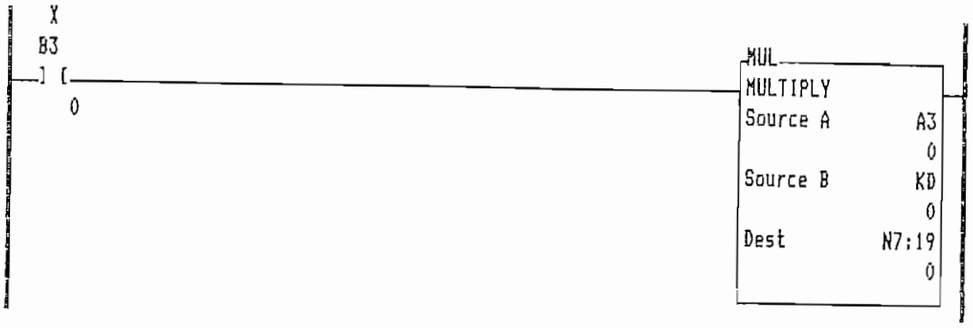


Rung 2:14



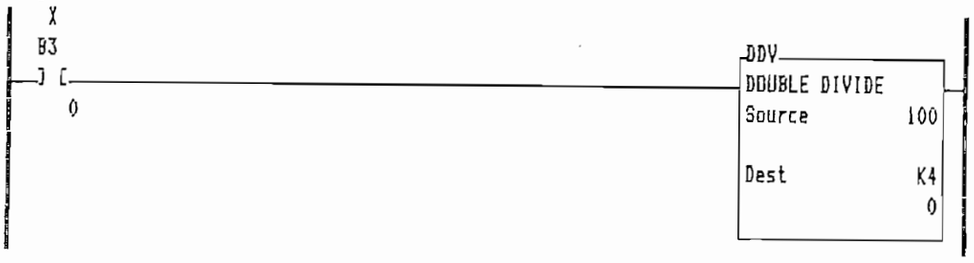
Rung 2:15

Rung 2:16



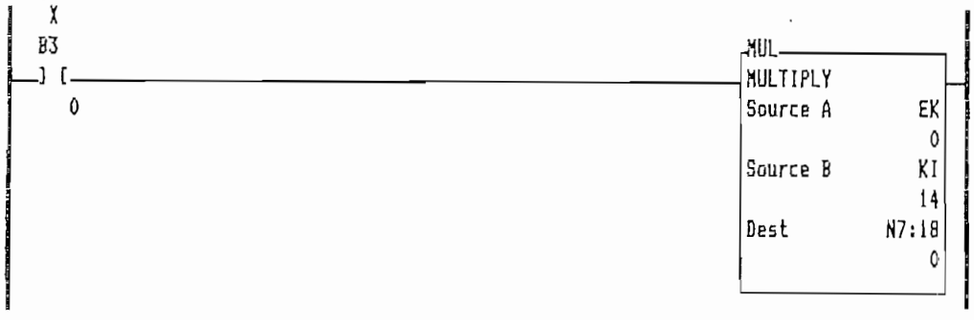
Rung 2:17

Rung 2:18

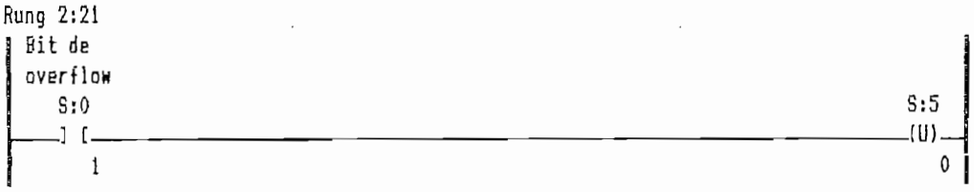


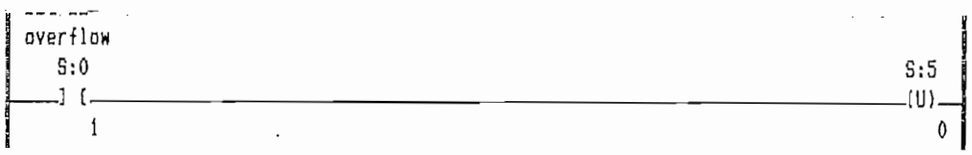
Rung 2:19

Rung 2:20

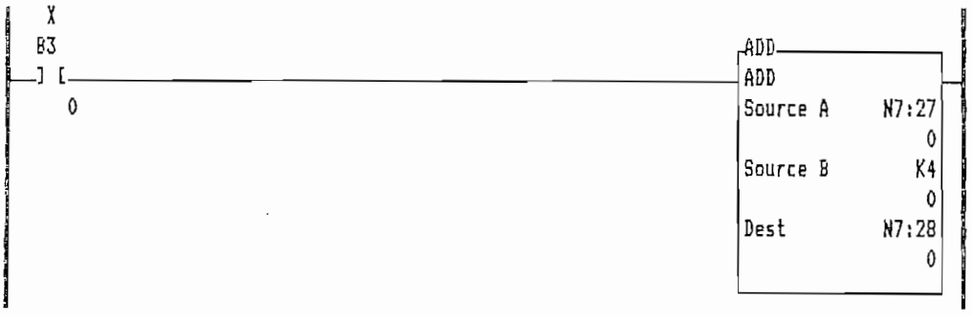


Rung 2:21



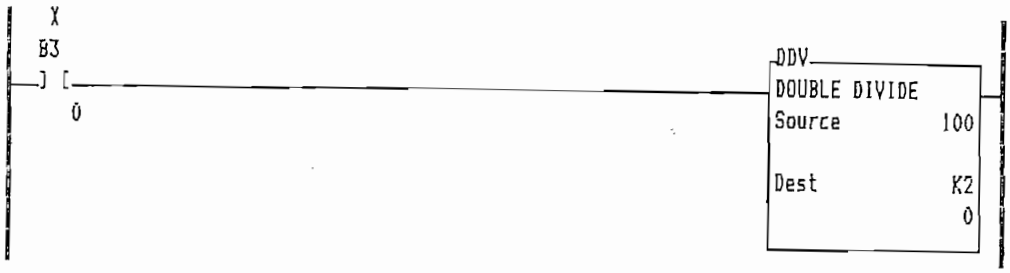


Rung 2:26



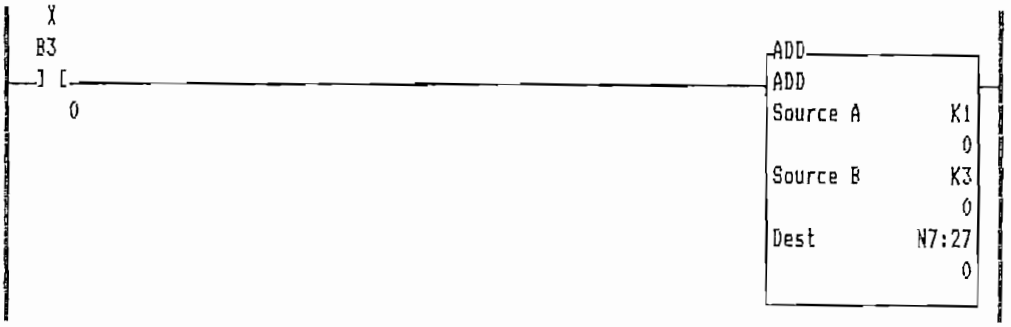
Rung 2:27

Rung 2:22



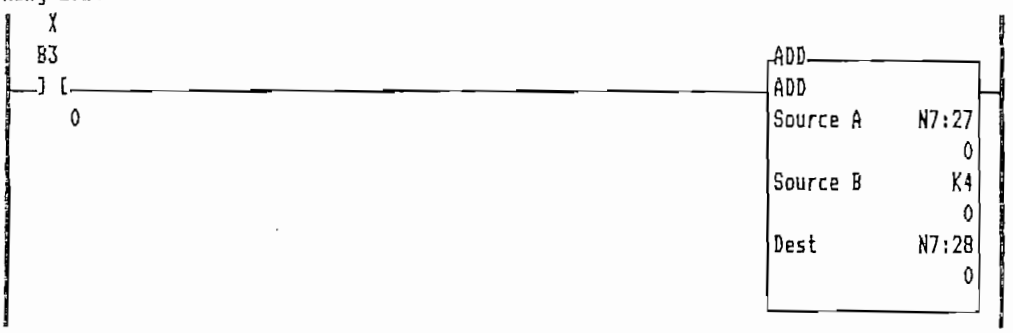
Rung 2:23

Rung 2:24



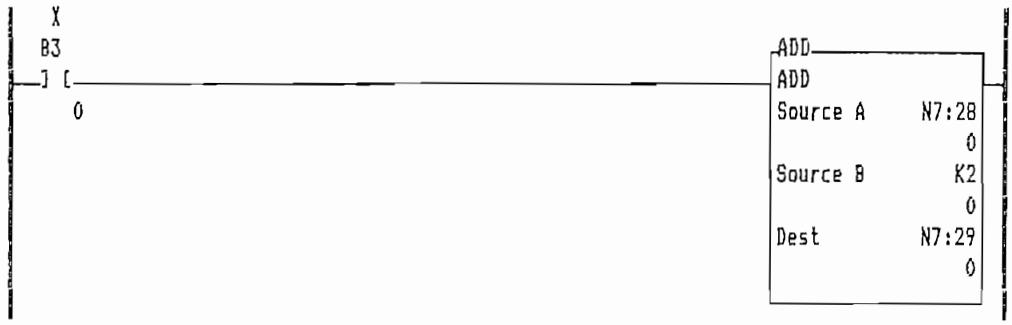
Rung 2:25

Rung 2:26



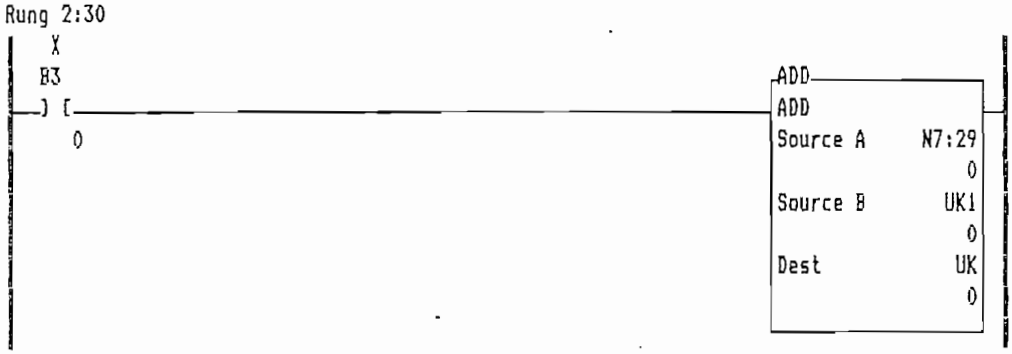
Rung 2:27

Rung 2:28



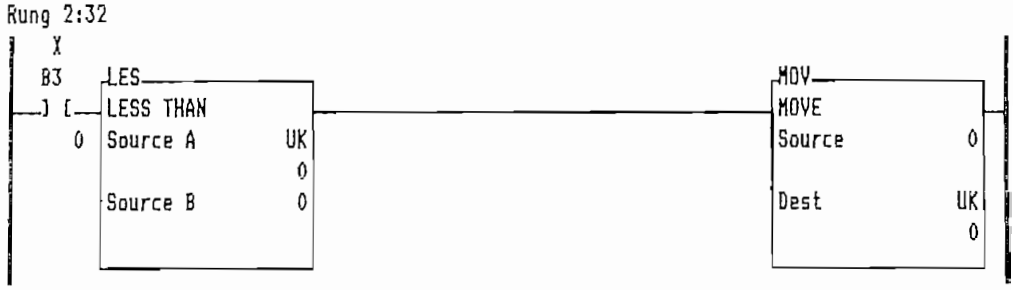
Rung 2:29

Rung 2:30



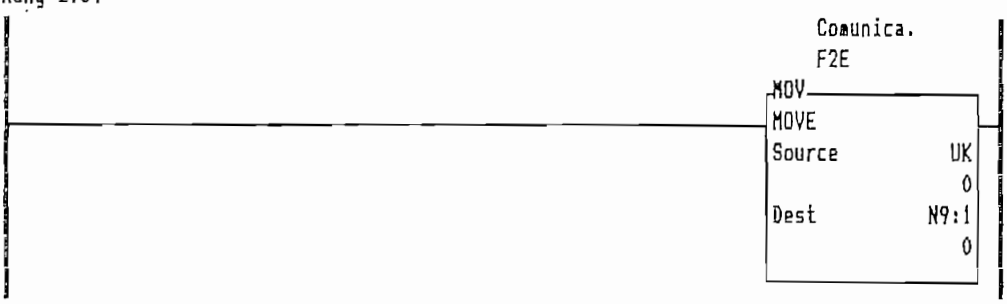
Rung 2:31

Rung 2:32

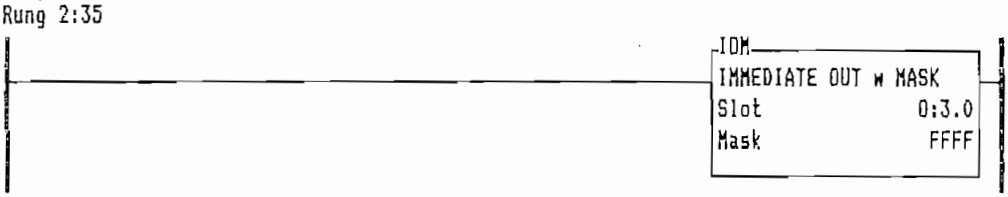


Rung 2:33

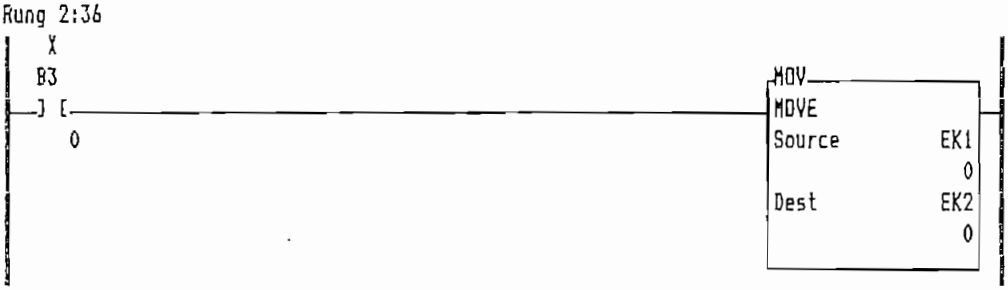
Rung 2:34



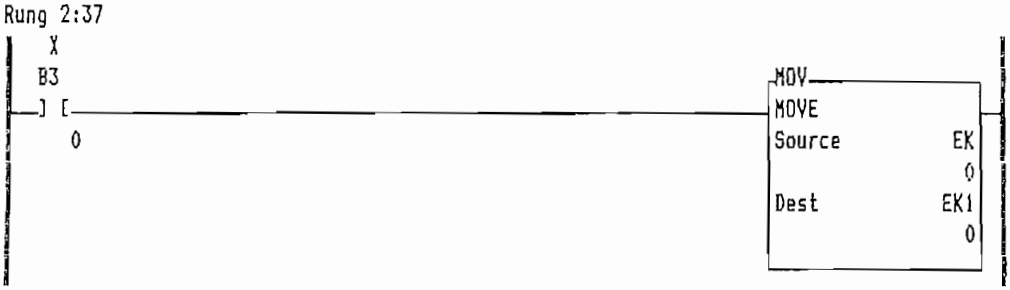
Rung 2:35



Rung 2:36



Rung 2:37



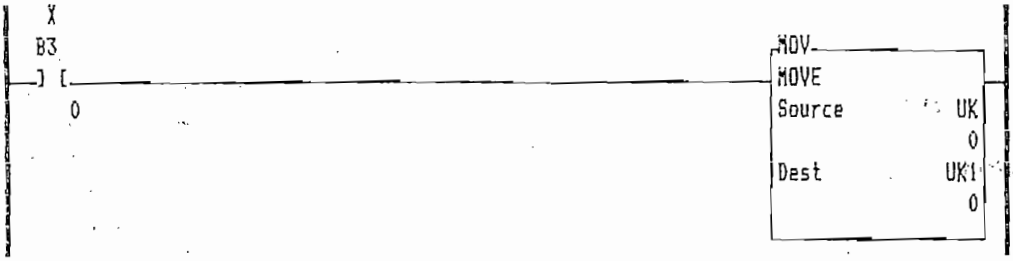
ANEXO F

PROGRAMA ASC-PID.EXE

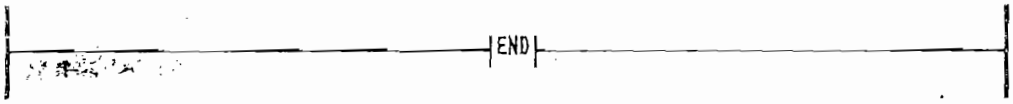
F. LISTADO DEL PROGRAMA ASC-PID.EXE

```
/*****************************************************************************/
/*                ADQUISICION DE DATOS, SUPERVISION Y CONTROL PID            */
/*                UTILIZANDO CONTROLADORES LOGICOS PROGRAMABLES               */
/*                                                                           */
/*   ESCUELA POLITECNICA NACIONAL - FACULTAD DE INGENIERIA ELECTRICA         */
/*                DEPARTAMENTE DE ELECTRONICA Y CONTROL                      */
/*                                                                           */
/*   Autor:    LUIS GIOVANNI PERRASO BASANTES                                */
/*   Director: ING. JORGE MOLINA                                             */
/*                                                                           */
/*   Fecha: Noviembre 1995                                                   */
/*   Modulo: ADSC-1.CPP                                                      */
/*****************************************************************************/

// Definiciones del Turbo Vision
#define Uses_TKeys
#define Uses_TEvent
#define Uses_TRect
#define Uses_TDialog
#define Uses_TButton
#define Uses_TDeskTop
#define Uses_MsgBox
#define Uses_THistory
#define Uses_TInputLine
#define Uses_TLabel

// Archivos de inclusion
#include <tv.h>
#include <stdlib.h>
#include <graphics.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include "cont.h"
#include "datos.h"

// Variables externas
extern char          char_tiempo[9], char_nodo[9], char_kp[9],
                     char_ki[9], char_kd[9];
extern float         tiempo_x;
extern unsigned int  destino;
extern char          err_msg[80];
extern int           activar_alarma;

// Variables globales
char   char1_Esc_logica[9], char2_Esc_logica[9];
int    on_off;
```

Rung 2:38



Rung 2:39



```
// Subrutina para lectura de cualquier archivo de datos del PLC
// excepto archivo de datos N:9.

TEvent      event;
int         regreso=1, x, sub, size=2;
char        s[80];
DAT_RET     datos_llegados;

switch (eleccion)
  {
      //    Variable "s" solo de paso
      case 0:               // Se selecciono una OUTPUT
            sub = 0;
            strcpy (s,"Leer en O:0");
            break;
      case 1:               // Se selecciono una INPUT
            sub = 0;
            strcpy (s,"Leer en I:1");
            break;
      case 3:               // Se selecciono un BIT
            sub = 0;
            strcpy (s,"Leer en B:3");
            break;
      case 41:
            eleccion = 4;
            sub = 1;
            strcpy (s,"Leer en T4:PRE");
            break;
      case 42:
            eleccion = 4;
            sub = 2;
            strcpy (s,"Leer en T4:ACC");
            break;
      case 51:
            eleccion = 5;
            sub = 1;
            strcpy (s,"Leer en C5:PRE");
            break;
      case 52:
            eleccion = 5;
            sub = 2;
            strcpy (s,"Leer en C5:ACC");
            break;
      case 6:
            sub = 0;
            strcpy (s,"Leer en R:6");
            break;
      case 7:
            sub = 0;
            strcpy (s,"Leer en N:7");
            break;
      default:
            break;
```

```
  }

TDialog *d = new TDialog(TRect(0,0,50,8), s);
d->options |= ofCentered;

 // Botones
d->insert(new TButton( TRect(25,5,35,7), "~C~ancel", cmCancel, bfNormal) );
d->insert(new TButton( TRect(14,5,24,7),"O~K~", cmOK, bfDefault) );

 // Linea de ingreso del valor
TInputLine *a = new TInputLine ( Trect (16,3,34,4), 20 );
d->insert ( a );
d->insert (new TLabel(TRect(14,2,38,3),"~Ingrese elemento a leer~",a));
d->insert (new THistory(TRect(35,3,38,4),a,cmLec_logica));
strcpy (s,char_Lec_logica);
d->setData(s);
d = (TDialog *) validView(d);
if (d != NULL)
{
 if (deskTop->execView(d) == cmOK)
 {
        d->getData(s);
        for(x=0;s[x];++x)
        {
         if(isdigit(s[x]) || s[x] == '.');
         else
         {
          strcpy (err_msg, "\n\x3 Dato erróneo.\n\x3  Inténtelo nuevamente.");
          messageBox (err_msg, mfError | mfOKButton);
          strcpy (s,char_Lec_logica);
          regreso=0;
          break;
         }
        }
        strcpy (char_Lec_logica,s);
 }
 else regreso=0;       // Se oprimio boton CANCEL
 destroy( d );
}

switch (regreso)
{
 case 1:                // No hay errores en valores y se oprimio OK
  suspend();

  x = registerbgifont(triplex_font);
  if (x < 0)
        {
         strcpy (err_msg,grapherrormsg(x));
         messageBox(err_msg, mfError | mfOKButton);
         resume();
         return;
        }
```

```
    if (graphicsStart() == False)
    {
        strcpy(err_msg,grapherrormsg(graphresult()));
        messageBox(err_msg, mfError | mfOKButton);
        resume();
        break;
    }

    datos_llegados = Lec_log (eleccion,atoi(char_Lec_logica),sub,size,2);

    graphicsStop();
    resume();

    if (datos_llegados.dr_error)
          {
           messageBox (datos_llegados.dr_mensaje,mfError | mfOKButton);
           return;
          }

    break;

   case 0:                  // ingresado dato erroneo
    break;

   default:
    break;
 }
}



/*****************************************************************************/
void TBGIApp::doEsc_logica (int eleccion)
/*****************************************************************************/
{
 // Subrutina para escribir un dato en cualquier archivo de datos del PLC
 // excepto archivo de datos N:9.

 TEvent        event;
 char          buffer_a[20], buffer_b[20];
 int           regreso=1, x, i=0, dato, lugar, sub, size=2;
 DAT_RET       datos_llegados;


 switch (eleccion)
    {
        //      Variable buffer_a solo de paso
//      case 0:                 // Se selecciono una OUTPUT
//              sub = 0;
//              strcpy (buffer_a,"Escribir en O:0");
//              break;
//      case 1:                 // Se selecciono una INPUT
```

```
//                sub = 0;
//                strcpy (buffer_a,"Escribir en I:1");
//                break;
        case 3:                 // Se selecciono un BIT
                sub   = lugar;
                lugar = 0;
                strcpy (buffer_a,"Escribir en B:3");
                break;
        case 41:
                eleccion = 4;
                sub = 1;
                strcpy (buffer_a,"Escribir en T4:PRE");
                break;
        case 42:
                eleccion = 4;
                sub = 2;
                strcpy (buffer_a,"Escribir en T4:ACC");
                break;
        case 51:
                eleccion = 5;
                sub = 1;
                strcpy (buffer_a,"Escribir en C5:PRE");
                break;
        case 52:
                eleccion = 5;
                sub = 2;
                strcpy (buffer_a,"Escribir en C5:ACC");
                break;
        case 6:
                sub = 0;
                strcpy (buffer_a,"Escribir en R:6");
                break;
        case 7:
                sub = 0;
                strcpy (buffer_a,"Escribir en N:7");
                break;
        default:
                break;
   }
 TDialog *d = new TDialog(TRect(0,0,50,8), buffer_a);
 d->options |= ofCentered;

  // Botones
 d->insert(new TButton( TRect(25,5,35,7), "~C~ancel", cmCancel, bfNormal) );
 d->insert(new TButton( TRect(14,5,24,7),"O~K~", cmOK, bfNormal) );

  // Linea de ingreso del valor
 TInputLine *a = new TInputLine( TRect(9,3,17,4), 20 );
 strcpy (buffer_a,char1_Esc_logica);
 a->setData(buffer_a);
 d->insert( a );
 d->insert(new TLabel(TRect(8,2,20,3),"~No.Elemento~",a));
 d->insert(new THistory(TRect(17,3,20,4),a,cmEsc_logica));
```

```
TInputLine *b = new TInputLine( TRect(27,3,36,4), 20 );
strcpy (buffer_b,char2_Esc_logica);
b->setData(buffer_b);
d->insert( b );
d->insert(new TLabel(TRect(22,2,43,3),"~Valor [-32768,32767]~",b));
d->insert(new THistory(TRect(36,3,39,4),b,cmEsc_logica));

d = (TDialog *) validView(d);
if (d != NULL)
{
 if (deskTop->execView(d) == cmOK)
 {
        a->getData(buffer_a);
        b->getData(buffer_b);

        for (x=0;buffer_a[x];++x)
        {
         if (isdigit(buffer_a[x]));
         else
         {
          regreso=0;
          break;
         }
        }

        if (buffer_b[0]=='-')
          i=1;
        if (atol(buffer_b)>=-32768 && atol(buffer_b)<=32767);
        else regreso=0;

        for (x=i;buffer_b[x];++x)
        {
         if (isdigit(buffer_b[x]));
         else
         {
          regreso=0;
          break;
         }
        }

        if (regreso==0)
        {
                strcpy (err_msg,"\x3 Dato alfanumérico \n\x3 o fuera de rango.\n\x3
                  Inténtelo nuevamente.");
          messageBox (err_msg, mfError | mfOKButton);
          strcpy (buffer_a,char1_Esc_logica);
          strcpy (buffer_b,char2_Esc_logica);
        }

        strcpy (char1_Esc_logica,buffer_a);
        strcpy (char2_Esc_logica,buffer_b);
```

```
        lugar = atoi (char1_Esc_logica);
        dato  = atoi (char2_Esc_logica);

   }
   else regreso=0;       // Se oprimio boton CANCEL
   destroy( d );

 }
 switch (regreso)
 {
   case 1:               // No hay errores en valores y se oprimio OK

     datos_llegados = Esc_log (eleccion,lugar,sub,dato,size);

     if (datos_llegados.dr_error)
          {
           messageBox (datos_llegados.dr_mensaje, mfError | mfOKButton);
           return;
          }
     messageBox (datos_llegados.dr_mensaje, mfInformation | mfOKButton);

     break;

   case 0:             // ingresado dato erróneo
     break;
 }
}


/*************************************************************************/
void TBGlApp::doDialogo()
/*************************************************************************/
{
        // Subrutina de informacion y ayuda.

        TDialog* d = new TDialog(TRect(10,4,71,16),"Ayuda");
        if (!d)
          return;

        d->insert(new TStaticText(TRect(2,1,59,3),
                "Sistema de adquisición de datos, supervisión y control PID "
                "utilizando controladores lógicos programables."));
        d->insert(new TStaticText(TRect(2,4,59,7),
                "Para ejecutar el control PID en un controlador lógico "
                "programable, seleccione la opción CONTROL PID del menú"
                " principal."));

        d->insert(new TButton(TRect(25,9,37,11),"~O~K",cmOK,bfDefault));

        deskTop->execView(d);
        destroy(d);
```

```
        TDialog* d1 = new TDialog(TRect(10,1,71,20),"Ayuda");
        if (!d1)
          return;

        d1->insert(new TStaticText(TRect(2,2,59,3),
                "Este programa, únicamente adquiere datos de:"));
        d1->insert(new TStaticText(TRect(16,4,44,13),
                "Seaal de salida        = N9:0 "
                "Seaal de control       = N9:1 "
                "PID ON-OFF             = N7:0 "
                "Referencia             = N7:2 "
                "Constante  Kp          = N7:3 "
                "Constante  Ki          = N7:4 "
                "Constante  Kd          = N7:5 "
                "Auto-Manual            = N7:40 "
                "Metodo de diseao       = N7:41 "));
        d1->insert(new TStaticText(TRect(2,14,59,15),
                "Verifique que sus datos concuerden con los anteriores"));

        d->insert(new TButton(TRect(24,16,36,18),"~O~K",cmOK,bfDefault));
        deskTop->execView(d1);
        destroy(d1);
}


/*************************************************************************/
void TBGIApp::poner_alarma(int eleccion)
/*************************************************************************/
{
 // Subrutina para fijar los niveles de alarma.

 TEvent         event;
 int            x, regreso=1;
 unsigned char s[20];

 TDialog *d = new TDialog(TRect(0,0,50,8), "Nivel de alarma");
 d->options |= ofCentered;

  // Botones
 d->insert(new TButton( TRect(25,5,35,7), "~C~ancel", cmCancel, bfNormal) );
 d->insert(new TButton( TRect(14,5,24,7),"O~K~", cmOK, bfDefault) );

  // Linea de ingreso del valor
 TInputLine *alarm = new TInputLine( TRect(16,3,34,4), 20 );
 d->insert( alarm );

 if (eleccion==1)
 {
  d->insert(new TLabel(TRect(14,2,40,3),"~Nivel Superior de alarma~",alarm));
  strcpy(s,char_alarm_sup);
 }
 if (eleccion==2)
 {
```

```
  d->insert(new TLabel(TRect(14,2,40,3),"~Nivel Inferior de alarma~",alarm));
  strcpy(s,char_alarm_inf);
 }
 d->insert(new THistory(TRect(35,3,38,4),alarm,cmAlarmaSup));

 d->setData(s);
 d = (TDialog *) validView(d);
 if (d != NULL)
 {
  if (deskTop->execView(d) == cmOK)
  {
        d->getData(s);
        for(x=0;s[x];++x)
        {
              if    ((isdigit(s[x]) || s[x]=='.' || s[x]=='-')    &&    atoi(s)<=ejey_max    &&
atoi(s)>=ejey_min);
         else
         {
         strcpy (err_msg,"\x3 Dato alfanumérico o fuera de rango.\n\x3 Inténtelo
                  nuevamente.");
          messageBox(err_msg, mfError | mfOKButton);
          if (eleccion==1)
           strcpy (s,char_alarm_sup);
          if (eleccion==2)
           strcpy (s,char_alarm_inf);
          regreso=0;
          break;
         }
        }

        if ((eleccion==1&&atof(s)>alarma_inf) || (eleccion==2&&alarma_sup>atof(s))
);
        else
         {
         messageBox ("\x3 Error en ingreso de valores. \n\x3 Verifique que: \n\x3 Nivel
                  sup. > Nivel inf.", mfError | mfOKButton);
          if (eleccion==1)
           strcpy(s,char_alarm_sup);
          if (eleccion==2)
           strcpy(s,char_alarm_inf);
          regreso=0;
         }


        if (eleccion==1)
          strcpy (char_alarm_sup,s);
        if (eleccion==2)
          strcpy (char_alarm_inf,s);

  }
  else regreso=0;       // Se oprimio boton CANCEL
  destroy( d );
 }
```

```
 switch (regreso)
 {
  case 1:

        if (eleccion==1)
          alarma_sup=atof(s);

        if (eleccion==2)
          alarma_inf=atof(s);

  case 0:
        break;
 }
}


/*************************************************************************/
void TBGIApp::OnOff()
/*************************************************************************/
{
 // Subrutina para pasar del modo Auto al Manual y viceversa.

 DAT_RET      datos_llegados;
 TEvent       event;

 datos_llegados = abrir_comunicacion ();
 if (datos_llegados.dr_error)
  {
   messageBox (datos_llegados.dr_mensaje,mfError | mfOKButton);
   return;
  }
 //      Lee N7:40. Auto o manual
 ini_log (0x0F,0xA2,2,7,40,0,0);
 datos_llegados = comunicar_log (0);
 if (datos_llegados.dr_error)
  {
   messageBox (datos_llegados.dr_mensaje,mfError | mfOKButton);
   return;
  }
 on_off=datos_llegados.dr_valor[0];

 TDialog *d = new TDialog(TRect(0,0,50,9), "Control PID");
 d->options |= ofCentered;

  // Botones
 d->insert(new TButton( TRect(25,6,35,8),"~C~ancel", cmCancel, bfNormal) );
 d->insert(new TButton( TRect(14,6,24,8),"O~K~", cmOK, bfDefault) );

 if (on_off==1)
 {
  d->insert(new TLabel(TRect(13,3,40,4),"Desactivar control PID..?",d));
 }
```

```
if (on_off==0)
{
  d->insert(new TLabel(TRect(13,3,37,4),"Activar control PID..?",d));
}

d = (TDialog *) validView(d);
if (d != NULL)
{
  if (deskTop->execView(d) == cmOK)
  {
        if (on_off==0)
          {
                //      Activa control. Pone 1 en N7:40
                ini_log (0x0F,0xAA,2,7,40,0,1);
                datos_llegados = comunicar_log (0);
                if (datos_llegados.dr_error)
                 {
                  messageBox (datos_llegados.dr_mensaje,mfError | mfOKButton);
                 }
                else
                 {
                  finalizar();
                  messageBox ("\n\x3 Control activado", mfInformation | mfOKButton);
                 }
          }

          else
          {
                //      Desactiva control. Pone 0 en N7:40
                ini_log (0x0F,0xAA,2,7,40,0,0);
                datos_llegados = comunicar_log (0);
                if (datos_llegados.dr_error)
                 {
                  messageBox (datos_llegados.dr_mensaje,mfError | mfOKButton);
                 }
                else
                 {
                 finalizar();
                        messageBox ("\n\x3 Control desactivado", mfInformation |
mfOKButton);
                 }
          }
   }
   else         // Se oprimio boton CANCEL
   destroy( d );
 }
}



/*************************************************************************/
void TBGIApp::poner_escalaY(int eleccion)
/*************************************************************************/
```

```
{
 // Subrutina para fijar la escala del eje Y.
 //      eleccion = 1 : Nivel maximo
 //      eleccion = 2 : Nivel minimo

 TEvent          event;
 int             x, regreso=1;
 unsigned char s[20];

 TDialog *d = new TDialog(TRect(0,0,50,8), "Escala eje Y");
 d->options |= ofCentered;

  // Botones
 d->insert(new TButton( TRect(25,5,35,7), "~C~ancel", cmCancel, bfNormal) );
 d->insert(new TButton( TRect(14,5,24,7),"O~K~", cmOK, bfDefault) );

  // Linea de ingreso del valor
 TInputLine *ejey = new TInputLine( TRect(16,3,34,4), 20 );
 d->insert( ejey );

 if (eleccion==1)
 {
  d->insert(new TLabel(TRect(15,2,40,3),"~Limite Superior~",ejey));
  strcpy(s,char_ejey_max);
 }
 if (eleccion==2)
 {
  d->insert(new TLabel(TRect(15,2,40,3),"~Limite Inferior~",ejey));
  strcpy(s,char_ejey_min);
 }
 d->insert(new THistory(TRect(35,3,38,4),ejey,cmEscalaY_max));

 d->setData(s);
 d = (TDialog *) validView(d);
 if (d != NULL)
 {
  if (deskTop->execView(d) == cmOK)
  {
        d->getData(s);
        for(x=0;s[x];++x)
        {
         if ( (isdigit(s[x]) || s[x]=='.' || s[x]=='-') );
         else
         {
          strcpy (err_msg, "\x3 Dato erróneo.\n\x3  Inténtelo nuevamente. \n\x3
                                  Mantenido último valor.");
          messageBox(err_msg, mfError | mfOKButton);
          if (eleccion==1)
           strcpy(s,char_ejey_max);
          if (eleccion==2)
           strcpy(s,char_ejey_min);
          regreso=0;
          break;
```

```
          }
        }
        if ( (eleccion==1&&(atof(s)>ejey_min)) || (eleccion==2&&(ejey_max>atof(s))));
        else
        {
            strcpy (err_msg, "\x3 Valor no v lido.\n\x3  Verifique que: \n\x3 Ymax >
Ymin");
          messageBox (err_msg, mfError | mfOKButton);
          if (eleccion==1)
            strcpy(s,char_ejey_max);
          if (eleccion==2)
            strcpy(s,char_ejey_min);
          regreso=0;
        }

        if (eleccion==1)
          strcpy (char_ejey_max,s);
        if (eleccion==2)
          strcpy (char_ejey_min,s);

    }
    else regreso=0;      // Se oprimio boton CANCEL
    destroy( d );
  }

  switch (regreso)
  {
    case 1:
          if (eleccion==1)
            ejey_max=atof(s);
          if (eleccion==2)
            ejey_min=atof(s);

    case 0:
          break;
  }
}
```

```
// ESTRUCTURA PRINCIPAL DEL PROGRAMA

void main()

{
 recuperar();
 TBGIApp bgiApp;
 bgiApp.run();
 guardar();
```

```
}

// FUNCION PARA GUARDAR LOS VALORES UTILIZADOS EN UN ARCHIVO

void guardar(void)
{
 FILE *confi;
 if((confi=fopen("config.pid","wb"))==NULL)
 {
  printf("No se puede abrir archivo de configuración CONFIG.PID");
  getch();
  exit(1);
 }
 fprintf (confi,"%s\n",char_tiempo);
 fprintf (confi,"%s\n",char_nodo);
 fprintf (confi,"%d\n",activar_alarma);
 fprintf (confi,"%s\n",char_alarm_sup);
 fprintf (confi,"%s\n",char_alarm_inf);
 fprintf (confi,"%s\n",char_ejey_max);
 fprintf (confi,"%s\n",char_ejey_min);
 fclose(confi);
}


// FUNCION PARA RECUPERAR LOS PARAMETROS DEL ARCHIVO DE
// CONFIGURACION config.pid

void recuperar(void)
{
 FILE *confi;
 if((confi=fopen("config.pid","rb"))==NULL)
 {
  printf("No se puede abrir archivo de configuración CONFIG.PID");
  getch();
  exit(1);
 }
 fscanf (confi,"%s\n",char_tiempo);
 fscanf (confi,"%s\n",char_nodo);
 fscanf (confi,"%s\n",err_msg);                    // err_msg toma estado de alarma
 fscanf (confi,"%s\n",char_alarm_sup);
 fscanf (confi,"%s\n",char_alarm_inf);
 fscanf (confi,"%s\n",char_ejey_max);
 fscanf (confi,"%s\n",char_ejey_min);
 fclose(confi);

 tiempo_x=atof(char_tiempo);
 destino=atoi(char_nodo);
 alarma_sup=atof(char_alarm_sup);
 alarma_inf=atof(char_alarm_inf);
 activar_alarma=atoi(err_msg);
 ejey_max=atof(char_ejey_max);
 ejey_min=atof(char_ejey_min);
}
```

```
/*****************************************************************************/
/*          ADQUISICION DE DATOS, SUPERVISION Y CONTROL PID                  */
/*                UTILIZANDO CONTROLADORES LOGICOS PROGRAMABLES               */
/*                                                                           */
/*  ESCUELA POLITECNICA NACIONAL - FACULTAD DE INGENIERIA ELECTRICA          */
/*            DEPARTAMENTE DE ELECTRONICA Y CONTROL                          */
/*                                                                           */
/*  Autor:    LUIS GIOVANNI PERRASO BASANTES                                 */
/*  Director: ING. JORGE MOLINA                                              */
/*                                                                           */
/*  Fecha: Noviembre 1995                                                    */
/*  Modulo: ADSC-2.CPP                                                       */
/*****************************************************************************/

// Definiciones Turbo Vision
#define Uses_TEvent
#define Uses_TDialog
#define Uses_TButton
#define Uses_TDeskTop
#define Uses_MsgBox
#define Uses_THistory
#define Uses_TInputLine
#define Uses_TLabel

// Archivos de inclusion
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <ctype.h>
#include <tv.h>
#include "calc.h"
#include "cont-1.h"
#include "datos.h"

// Variables globales
char    char_tiempo[9], char_nodo[9], char_kp[9], char_ki[9], char_kd[9],
        char1_Esc_no_proteg[9], char2_Esc_no_proteg[9],
        char1_Lec_no_proteg[9], char2_Lec_no_proteg[9],char_Evaluar_Nodo[9];
float   tiempo_x=20, val_ref;
int     val_Evaluar_Nodo;

// Variables externas
extern  char            err_msg[80];
extern  unsigned int    destino;

// Funciones utilizadas en este modulo, definidas en otro modulo
extern "C" DAT_RET  plc             (void);
extern "C" DAT_RET  Esc_df9         (int lugar,int dato);
extern "C" DAT_RET  Lec_df9         (int lugar,int numero_datos);
```

```
extern "C" DAT_RET  diag              (int nodo);
extern "C" DAT_RET  Esc_log           (int a,int b,int c,int d,int e);
extern  void        grafico_proceso   (void);



/*************************************************************************/
void TBGIApp::dotiempo()
/*************************************************************************/
{
 // Subrutina para fijar el tiempo de visualizacion
 // Tiempo maximo = 150 segundos

 TEvent       event;
 int          x;
 char         s[20];

 TDialog *d = new TDialog(TRect(0,0,50,8), "Tiempo de visualización");
 d->options |= ofCentered;

  // Botones
 d->insert(new TButton( TRect(25,5,35,7), "Cancel", cmCancel, bfNormal) );
 d->insert(new TButton( TRect(14,5,24,7),"O~K~", cmOK, bfDefault) );

  // Linea de ingreso del valor
 TInputLine *pru1 = new TInputLine( TRect(16,3,34,4), 20 );
 d->insert( pru1 );
 d->insert(new TLabel(TRect(6,2,44,3),"~Ingrese el tiempo~ (150 segundos max.)",pru1));
 d->insert(new THistory(TRect(35,3,38,4),pru1,cmDotiempo));
 strcpy(s,char_tiempo);
 d->setData(s);

 d = (TDialog *) validView(d);
 if (d != NULL)
 {
  if (deskTop->execView(d) == cmOK)
  {
        d->getData(s);
        for(x=0;s[x];++x)
        {
         if((isdigit(s[x]) || s[x] == '.') && atof(s)<=150);
         else
         {
          strcpy (err_msg, "\x3 Ingresado dato alfanumérico o fuera de rango\n\x3
                           Inténtelo nuevamente");
          messageBox(err_msg, mfError | mfOKButton);
          strcpy(s,char_tiempo);
          break;
         }
        }
        strcpy(char_tiempo,s);
        tiempo_x = atof(char_tiempo);
        if (tiempo_x<5) tiempo_x=5;
```

```
    }

    else            // Se oprimio boton CANCEL
    destroy( d );
  }
}

/*************************************************************************/
void TBGIApp::doEsc_no_proteg()
/*************************************************************************/
{
  // Subrutina para escribir datos en archivo de datos N:9 del PLC.

  TEvent        event;
  char          buffer_a[20], buffer_b[20];
  int           regreso=1, x, i=0, dato, lugar;
  DAT_RET       datos_llegados;

  TDialog *d = new TDialog(TRect(0,0,50,8), "Escribir en N:9");
  d->options |= ofCentered;

    // Botones
  d->insert(new TButton( TRect(25,5,35,7), "~C~ancel", cmCancel, bfDefault) );
  d->insert(new TButton( TRect(14,5,24,7),"O~K~", cmOK, bfNormal) );

    // Linea de ingreso del valor
  TInputLine *a = new TInputLine( TRect(9,3,17,4), 20 );
  strcpy (buffer_a,char1_Esc_no_proteg);
  a->setData(buffer_a);
  d->insert( a );
  d->insert(new TLabel(TRect(8,2,20,3),"~No.Elemento~",a));
  d->insert(new THistory(TRect(17,3,20,4),a,cmEsc_no_proteg));

  TInputLine *b = new TInputLine( TRect(27,3,36,4), 20 );
  strcpy (buffer_b,char2_Esc_no_proteg);
  b->setData(buffer_b);
  d->insert( b );
  d->insert(new TLabel(TRect(22,2,43,3),"~Valor [-32768,32767]~",b));
  d->insert(new THistory(TRect(36,3,39,4),b,cmEsc_no_proteg));

  d = (TDialog *) validView(d);
  if (d != NULL)
  {
    if (deskTop->execView(d) == cmOK)
    {
          a->getData(buffer_a);
          b->getData(buffer_b);

          for (x=0;buffer_a[x];++x)
          {
            if (isdigit(buffer_a[x]));
            else
```

```
        {
          regreso=0;
          break;
        }
      }
      if (buffer_b[0]=='-')
        i=1;
      if (atol(buffer_b)>=-32768 && atol(buffer_b)<=32767);
      else regreso=0;

      for (x=i;buffer_b[x];++x)
      {
       if ( isdigit(buffer_b[x]) );
       else
       {
         regreso=0;
         break;
       }
      }

      if (regreso==0)
      {
         strcpy (err_msg, "\x3 Dato alfanumérico \n\x3 o fuera de rango.\n\x3
                                  Inténtelo nuevamente.");
         messageBox (err_msg, mfError | mfOKButton);
         strcpy (buffer_a,char1_Esc_no_proteg);
         strcpy (buffer_b,char2_Esc_no_proteg);
      }

      strcpy (char1_Esc_no_proteg,buffer_a);
      strcpy (char2_Esc_no_proteg,buffer_b);

      lugar = atoi (char1_Esc_no_proteg);
      dato  = atoi (char2_Esc_no_proteg);

      if (regreso)
       {
         datos_llegados = Esc_df9 (lugar,dato);
         if (datos_llegados.dr_error)
           {
                messageBox(datos_llegados.dr_mensaje, mfError | mfOKButton);
                return;
           }
         messageBox(datos_llegados.dr_mensaje, mfInformation | mfOKButton);
       }
   }
   else /* Se oprimio boton CANCEL */
   destroy( d );
  }
}
```

```
/*************************************************************************/
void TBGIApp::doLec_no_proteg()
/*************************************************************************/
{
 // Subrutina para leer datos del archivo de datos N:9 del PLC.

 TEvent        event;
 char          s[80], buffer_a[20], buffer_b[20];
 int           regreso=1, x, numero_datos, lugar;
 DAT_RET       datos;

 TDialog *d = new TDialog(TRect(0,0,50,8), "Leer en N:9");
 d->options |= ofCentered;

  // Botones
 d->insert(new TButton( TRect(25,5,35,7), "~C~ancel", cmCancel, bfNormal) );
 d->insert(new TButton( TRect(14,5,24,7),"O~K~", cmOK, bfDefault) );

  // Linea de ingreso del valor
 TInputLine *a = new TInputLine( TRect(10,3,18,4), 20 );
 strcpy (buffer_a,char1_Lec_no_proteg);
 a->setData(buffer_a);
 d->insert( a );
 d->insert(new TLabel(TRect(8,2,20,3),"~No.Elemento~",a));
 d->insert(new THistory(TRect(18,3,21,4),a,cmLec_no_proteg));

 TInputLine *b = new TInputLine( TRect(26,3,34,4), 20 );
 strcpy (buffer_b,char2_Lec_no_proteg);
 b->setData(buffer_b);
 d->insert( b );
 d->insert(new TLabel(TRect(21,2,43,3),"~Número datos [max.20]~",b));
 d->insert(new THistory(TRect(34,3,37,4),b,cmLec_no_proteg));

 d = (TDialog *) validView(d);
 if (d != NULL)
 {
  if (deskTop->execView(d) == cmOK)
  {
        a->getData(buffer_a);
        b->getData(buffer_b);

        for (x=0;buffer_a[x];++x)
        {
          if (isdigit(buffer_a[x]));
          else
          {
            regreso=0;
            break;
          }
        }

        for (x=0;buffer_b[x];++x)
        {
```

```
 if (isdigit(buffer_b[x]));
 else
 {
  regreso=0;
  break;
 }
}

if (regreso==0)
{
   strcpy(err_msg, "\n\x3 Dato erróneo.\n\x3  Inténtelo nuevamente");
   messageBox (err_msg, mfError | mfOKButton);
   strcpy (buffer_a,char1_Lec_no_proteg);
   strcpy (buffer_b,char2_Lec_no_proteg);
}

strcpy (char1_Lec_no_proteg,buffer_a);
strcpy (char2_Lec_no_proteg,buffer_b);

lugar = atoi (char1_Lec_no_proteg);
numero_datos  = atoi (char2_Lec_no_proteg);
if (numero_datos>20)
  numero_datos=20;

if (regreso)
{
 int errorcode;
 suspend();

 //      Carga tipo de letra a ser utilizada en el programa
 errorcode = registerbgifont(triplex_font);
 if (errorcode < 0)
   {
    strcpy (err_msg,grapherrormsg(errorcode));
    messageBox(err_msg, mfError | mfOKButton);
    resume();
    return;
   }

 if (graphicsStart() == False)
   {
    strcpy(err_msg,grapherrormsg(graphresult()));
    messageBox(err_msg, mfError | mfOKButton);
    resume();
    return;
   }
 datos = Lec_df9 (lugar,numero_datos);
 graphicsStop();
 resume();

 if (datos.dr_error)
   {
         messageBox(datos.dr_mensaje, mfError | mfOKButton);
```

```
                    return;
                }
            }
        }
        else  /* Se oprimio boton CANCEL */
        destroy( d );
    }
}



/*************************************************************************/
void TBGIApp::Nodo()
/*************************************************************************/
{
  // Subrutina para fijar el numero de nodo del PLC donde se adquieren
  // los datos.

  TEvent        event;
  int           x, regreso=1;
  unsigned char s[20];

  TDialog *d = new TDialog(TRect(0,0,50,8), "Dirección Red DH-485");
  d->options |= ofCentered;
    // Botones
  d->insert(new TButton( TRect(25,5,35,7), "~C~ancel", cmCancel, bfNormal) );
  d->insert(new TButton( TRect(14,5,24,7),"O~K~", cmOK, bfDefault) );

    // Linea de ingreso del valor
  TInputLine *nodo = new TInputLine( TRect(16,3,34,4), 20 );
  d->insert( nodo );
  d->insert(new TLabel(TRect(12,2,40,3),"~Ingrese valor entre [1-31]~",nodo));
  d->insert(new THistory(TRect(35,3,38,4),nodo,cmFijar_Nodo));
  strcpy(s,char_nodo);
  d->setData(s);
  d = (TDialog *) validView(d);
  if (d != NULL)
  {
    if (deskTop->execView(d) == cmOK)
    {
        d->getData(s);
        for(x=0;s[x];++x)
        {
          if ( isdigit(s[x]) && atoi(s)<=31 && atoi(s)>=1 );
          else
          {
            strcpy (err_msg, "\x3 Dato erróneo.\n\x3  Inténtelo nuevamente. \n\x3
                                Mantenido último valor.");
            messageBox(err_msg, mfError | mfOKButton);
            strcpy(s,char_nodo);
            regreso=0;
```

```
            break;
          }
        }
        strcpy (char_nodo,s);
   }
   else regreso=0;       /* Se oprimio boton CANCEL */
   destroy( d );
 }
 switch (regreso)
 {
   case 1:
        destino=atoi(s);
   case 0:
        break;
 }
}

/*************************************************************************/
void TBGIApp::doEvaluar_Nodo()
/*************************************************************************/
{
 // Subrutina para comprobar el tipo de PLC conectado en una red de PLC's
 // Nodo=0, constituye la tarjeta 1784-KR

 TEvent        event;
 int           x, regreso=1;
 unsigned char s[20];
 DAT_RET       valor;

 TDialog *d = new TDialog(TRect(0,0,50,8), "Evaluación nodo DH-485");
 d->options |= ofCentered;

  // Botones
 d->insert(new TButton( TRect(25,5,35,7), "~C~ancel", cmCancel, bfNormal) );
 d->insert(new TButton( TRect(14,5,24,7),"O~K~", cmOK, bfDefault) );

  // Linea de ingreso del valor
 TInputLine *a = new TInputLine( TRect(16,3,34,4), 20 );
 d->insert( a );
 d->insert(new TLabel(TRect(12,2,40,3),"~Ingrese valor entre [0-31]~",a));
 d->insert(new THistory(TRect(35,3,38,4),a,cmEvaluar_Nodo));
 strcpy(s,char_Evaluar_Nodo);
 d->setData(s);
 d = (TDialog *) validView(d);
 if (d != NULL)
 {
  if (deskTop->execView(d) == cmOK)
  {
        d->getData(s);
        for(x=0;s[x];++x)
        {
          if ( (isdigit(s[x]))&&(atoi(s)<=31) );
          else
```

```
        {
          strcpy(err_msg, "\n\x3 Dato erróneo.\n\x3  Inténtelo nuevamente.");
          messageBox(err_msg, mfError | mfOKButton);
          strcpy(s,char_Evaluar_Nodo);
          regreso=0;
          break;
        }
      }
      strcpy (char_Evaluar_Nodo,s);
      val_Evaluar_Nodo=atoi(char_Evaluar_Nodo);

      if (regreso)
      {
         valor = diag (val_Evaluar_Nodo);
         if (valor.dr_error)
               {
                messageBox(valor.dr_mensaje, mfError | mfOKButton);
                return;
               }
         messageBox(valor.dr_mensaje, mfInformation | mfOKButton);
      }
  }
   else                  /* Se oprimio boton CANCEL */
   destroy( d );
 }
}


/*****************************************************************************/
void TBGIApp::monitorear()
/*****************************************************************************/
{
 // Subrutina para visualizar los datos adquiridos del modulo analogico
 // y direccionados en el PLC al archivo de datos N:9.

 int            errorcode;
 DAT_RET        datos_llegados;

      suspend();

 //   Carga tipo de letra a ser utilizada en el programa
      errorcode = registerbgifont(small_font);
      if (errorcode < 0)
        {
         strcpy (err_msg,grapherrormsg(errorcode));
         messageBox(err_msg, mfError | mfOKButton);
         resume();
         return;
        }

      errorcode = registerbgifont(triplex_font);
      if (errorcode < 0)
       {
```

```
            strcpy (err_msg,grapherrormsg(errorcode));
            messageBox(err_msg, mfError | mfOKButton);
            resume();
            return;
           }

          if (graphicsStart() == False)
           {
            strcpy(err_msg,grapherrormsg(graphresult()));
            messageBox(err_msg, mfError | mfOKButton);
            resume();
            return;
           }
          datos_llegados = plc();
          graphicsStop();
          resume();
          if (datos_llegados.dr_error)
            messageBox(datos_llegados.dr_mensaje, mfError | mfOKButton);
}



/*************************************************************************/
void TBGIApp::informacion1()
/*************************************************************************/
{
 // Subrutina que muestra un grafico esquematico del sistema implementado

 int              errorcode;
 suspend();

  //     Carga tipo de letra a ser utilizada en el programa
          errorcode = registerbgifont(small_font);
          if (errorcode < 0)
           {
            strcpy (err_msg,grapherrormsg(errorcode));
            messageBox(err_msg, mfError | mfOKButton);
            resume();
            return;
           }

          errorcode = registerbgifont(triplex_font);
          if (errorcode < 0)
           {
            strcpy (err_msg,grapherrormsg(errorcode));
            messageBox(err_msg, mfError | mfOKButton);
            resume();
            return;
           }

          if (graphicsStart() == False)
           {
            strcpy(err_msg,grapherrormsg(graphresult()));
```

```
            messageBox(err_msg, mfError | mfOKButton);
            resume();
            return;
           }

         grafico_proceso();
         getch();

         graphicsStop();
         resume();
}


/*************************************************************************/
void TBGIApp::calculator()
/*************************************************************************/
{
   // Subrutina del TurboVision que despliega una calculadora.

   TCalculator *calc = (TCalculator *) validView(new TCalculator);

   if(calc != 0)
        deskTop->insert(calc);
}


    /*****************************************************************************/
   /*                ADQUISICION DE DATOS, SUPERVISION Y CONTROL PID            */
   /*                UTILIZANDO CONTROLADORES LOGICOS PROGRAMABLES               */
   /*                                                                           */
   /*   ESCUELA POLITECNICA NACIONAL - FACULTAD DE INGENIERIA ELECTRICA         */
   /*                DEPARTAMENTE DE ELECTRONICA Y CONTROL                      */
   /*                                                                           */
   /*   Autor:    LUIS GIOVANNI PERRASO BASANTES                               */
   /*   Director:  ING. JORGE MOLINA                                            */
   /*                                                                           */
   /*   Fecha:  Noviembre 1995                                                  */
   /*   Modulo: AYUDA1.CPP                                                      */
   /*   Pantalla grafica explicativa del proceso implementado                   */
    /*****************************************************************************/

// Archivo de inclusion
#include <graphics.h>


void dibuja_pc(int x, int y, float z)
{
        // Subrutina que grafica una computadora
        // (x,y,z) = vertice superior izquierdo de pantalla y escala

        int  poly[8];
        // Dibuja pantalla
        setcolor (7);
        bar3d (x,y,x+z*60,y+z*60,0,1);
```

```
        bar3d (x+z*8,y+z*8,x+z*60-z*8,y+z*60-z*8,0,1);

        // Dibuja teclado
        line  (x,y+z*65,x+z*60,y+z*65);          // linea horiz.
        line  (x-z*20,y+z*90,x+z*80,y+z*90);     // linea horiz.
        line  (x,y+z*65,x-z*20,y+z*90);          // Linea inclinada izq.
        line  (x+z*60,y+z*65,x+z*80,y+z*90);     // Linea inclinada der.
        line  (x-z*20,y+z*93,x+z*80,y+z*93);     // borde inf.

        // Vertices del teclado. Dibuja teclado
        poly[0]=x+z*2;
        poly[1]=y+z*67;
        poly[2]=x-z*12;
        poly[3]=y+z*86;
        poly[4]=x+z*72;
        poly[5]=y+z*86;
        poly[6]=x+z*58;
        poly[7]=y+z*67;
        setfillstyle (7,8);
        fillpoly (4,poly);
        setfillstyle (1,0);
}

void dibuja_plc(int x1, int y1, int z1)
{
        // Subrutina para graficacion de un PLC.

        setcolor (11);
        bar3d (x1,y1,x1+z1*6,y1+z1*4,2,1);
        setfillstyle (2,11);
        bar3d (x1+z1*2,y1+z1,x1+z1*6,y1+z1*4,0,1);
        setfillstyle (1,0);
        line  (x1,y1+z1,x1+z1*6,y1+z1);
        line  (x1,y1+z1*3,x1+z1*6,y1+z1*3);
        line  (x1+z1*2,y1,x1+z1*2,y1+z1*4);
        line  (x1+z1*3,y1,x1+z1*3,y1+z1*4);
        line  (x1+z1*4,y1,x1+z1*4,y1+z1*4);
        line  (x1+z1*5,y1,x1+z1*5,y1+z1*4);
}

void dibuja_planta(int x2, int y2)
{
        // Subrutina que grafica la "planta".
        setcolor (11);
        ellipse  (x2, y2, 0, 360, 60, 40);
        settextstyle (TRIPLEX_FONT,HORIZ_DIR,1);
        setcolor  (3);
        outtextxy (x2-35,y2-25,"Proceso");
        outtextxy (x2-35,y2-7,"continuo");
        settextstyle (DEFAULT_FONT,HORIZ_DIR,1);
}

void dibuja_flecha(int x3, int y3)
```

```
{
        setcolor (10);
        line (x3,y3,x3+20,y3);              // lineas horiz.
        line (x3,y3+10,x3+20,y3+10);
        line (x3-10,y3+5,x3+2,y3-2);     // Flecha izq.
        line (x3-10,y3+5,x3+2,y3+12);
        line (x3+20,y3-2,x3+30,y3+5);  // Flecha der.
        line (x3+20,y3+12,x3+30,y3+5);
}

void grafico_proceso()
{
        int leyenda1x=60, leyenda1y=140;
        int leyenda2x=230, leyenda2y=320;
        int leyenda3x=350, leyenda3y=150;

        setfillstyle(2,0);
        setcolor (7);
        // Dibuja marco
        bar3d (0,0,getmaxx(),getmaxy(),0,1);
        bar3d (30,80,getmaxx()-30,getmaxy()-30,0,1);
        // Dibuja PC
        dibuja_pc  (110,210,1.2);
        // Dibuja PLC
        dibuja_plc (260,235,10);
        // Dibuja planta
        dibuja_planta (460,260);
        // Dibuja flecha
        dibuja_flecha (205,250);
        dibuja_flecha (350,250);
        // Pone leyendas
        settextstyle (TRIPLEX_FONT,HORIZ_DIR,1);
        setcolor  (14);
        outtextxy (26,20,"SISTEMA DE ADQUISICION DE DATOS, SUPERVISION Y CONTROL
                          PID");
        outtextxy (26,40,"      UTILIZANDO CONTROLADORES LOGICOS PROGRAMABLES");
        setcolor  (7);
        outtextxy (275,205,"PLC");

        settextstyle (SMALL_FONT,HORIZ_DIR,4);
        setcolor  (7);
        outtextxy (30,457,"Por: Giovanni Perraso B.  Director: Ing. Jorge Molina.  EPN-FIE
                           1995.");

        setcolor  (12);
        outtextxy (leyenda1x,leyenda1y  ,"El PC a través de la interface");
        outtextxy (leyenda1x,leyenda1y+10,"Allen-Bradley 1784-KR  y el software");
        outtextxy (leyenda1x,leyenda1y+20,"A-B 6001-F2E adquiere cualquier dato");
        outtextxy (leyenda1x,leyenda1y+30,"procesado por el PLC.");

        outtextxy (leyenda2x,leyenda2y  ,"El PLC, ejecutando un algoritmo PID");
        outtextxy (leyenda2x,leyenda2y+10,"digital, programado en su memoria");
        outtextxy (leyenda2x,leyenda2y+20,"interna, o la instruccion PID");
```

```
        outtextxy (leyenda2x,leyenda2y+30,"incluida, controla el proceso continuo.");

        outtextxy (leyenda3x,leyenda3y   ,"El módulo analógico A-B 1747-NIO4I");
        outtextxy (leyenda3x,leyenda3y+10,"(entrada:0-10V;  salida: 0-20mA)");
        outtextxy (leyenda3x,leyenda3y+20,"es la interface  entre la planta");
        outtextxy (leyenda3x,leyenda3y+30,"y el PLC.");
}


   /*****************************************************************************/
   /*              ADQUISICION DE DATOS, SUPERVISION Y CONTROL PID              */
   /*              UTILIZANDO CONTROLADORES LOGICOS PROGRAMABLES                */
   /*                                                                           */
   /*  ESCUELA POLITECNICA NACIONAL - FACULTAD DE INGENIERIA ELECTRICA          */
   /*              DEPARTAMENTE DE ELECTRONICA Y CONTROL                        */
   /*                                                                           */
   /*  Autor:    LUIS GIOVANNI PERRASO BASANTES                                 */
   /*  Director: ING. JORGE MOLINA                                              */
   /*                                                                           */
   /*  Fecha:  Noviembre 1995                                                   */
   /*  Modulo: MEM-1.CPP                                                        */
   /*                -                                                          */
    /*****************************************************************************/

// Definiciones del Turbo Vision
#define Uses_TKeys
#define Uses_TEvent
#define Uses_TMenuBar
#define Uses_TSubMenu
#define Uses_TMenuItem
#define Uses_TStatusItem
#define Uses_TStatusDef
#define Uses_MsgBox


// Archivos de inclusion
#include <tv.h>
#include <string.h>
#include <conio.h>
#include "cont-1.h"

// Variables globales y externas
int             activar_alarma;
extern int      Tipo;

void TBGIApp::handleEvent(TEvent& event)
{
 char aboutMsg[100];
 TApplication::handleEvent(event);
 switch (event.what)
 {
  case evCommand:
```

```
    switch (event.message.command)
    {
        case cmAboutBox:      //mensaje de informacion inicial
        strcpy   (aboutMsg,"\x3 Adquisición de datos, supervisión y control PID
utilizando                    PLC's. \n\x3 Por: Giovanni Perraso B.");
        messageBox(aboutMsg, mfInformation | mfOKButton);
        break;

        case cmDotiempo:
         dotiempo();
         break;

        case cmTanque:
         Tipo=1;
         PIDI(1,1);
         break;

        case cmTanqueCom:
         Tipo=2;
         PIDI(1,1);
         break;

        case cmAyuda:
         doDialogo();
         break;

        case cmEsc_logicaBit:
         doEsc_logica(3);
         break;

        case cmEsc_logicaTim1:
         doEsc_logica(41);
         break;

        case cmEsc_logicaTim2:
         doEsc_logica(42);
         break;

        case cmEsc_logicaCou1:
         doEsc_logica(51);
         break;

        case cmEsc_logicaCou2:
         doEsc_logica(52);
         break;

        case cmEsc_logicaCon:
         doEsc_logica(6);
         break;

        case cmEsc_logicaInt:
         doEsc_logica(7);
         break;
```

```
case cmLec_logicaInp:
 doLec_logica(1);
 break;

case cmLec_logicaOut:
 doLec_logica(0);
 break;

case cmLec_logicaBit:
 doLec_logica(3);
 break;

case cmLec_logicaTim1:
 doLec_logica(41);
 break;

case cmLec_logicaTim2:
 doLec_logica(42);
 break;

case cmLec_logicaCou1:
 doLec_logica(51);
 break;

case cmLec_logicaCou2:
 doLec_logica(52);
 break;

case cmLec_logicaCon:
 doLec_logica(6);
 break;

case cmLec_logicaInt:
 doLec_logica(7);
 break;

case cmFijar_Nodo:
 Nodo();
 break;

case cmEvaluar_Nodo:
 doEvaluar_Nodo();
 break;

case cmLec_no_proteg:
 doLec_no_proteg();
 break;

case cmEsc_no_proteg:
 doEsc_no_proteg();
 break;
```

```
        case cmDoCalculadora:
          calculator();
          break;

        case cmMonitorear:
          monitorear();
          break;

        case cmInfo1:
          informacion1();
          break;

        case cmAlarmaSup:
          poner_alarma(1);
          break;

        case cmAlarmaInf:
          poner_alarma(2);
          break;

        case cmActivar_alarm:
          activar_alarma=1;
          messageBox("\n\x3 Alarmas activadas", mfInformation | mfOKButton);
          break;

        case cmDesactivar_alarm:
          activar_alarma=0;
          messageBox("\n\x3 Alarmas desactivadas", mfInformation | mfOKButton);
          break;

        case cmOnOff:
          OnOff();
          break;

        case cmEscalaY_max:
          poner_escalaY(1);
          break;

        case cmEscalaY_min:
          poner_escalaY(2);
          break;

        default:
          return;
      }
    break;

    default:
      return;
  }
  clearEvent(event);
}
```

```
TMenuBar *TBGIApp::initMenuBar(TRect r)
{
 r.b.y = r.a.y + 1;
 return new TMenuBar( r,

 *new TSubMenu("~\xF0~",hcNoContext)+
   *new TMenuItem("~A~cerca de...",cmAboutBox,kbNoKey,hcNoContext )+
   *new TMenuItem("~C~alculadora",cmDoCalculadora,kbNoKey,hcNoContext)+
   *new TMenuItem("~S~alir",cmQuit,kbAltX,hcNoContext,"Alt-X")+

 *new TSubMenu("~R~ed",hcNoContext)+
   *new TMenuItem("~N~úmero de nodo",cmFijar_Nodo,kbNoKey,hcNoContext)+
   *new TMenuItem("~E~valuar nodo",cmEvaluar_Nodo,kbNoKey,hcNoContext)+

 *new TSubMenu("~L~eer PLC",hcNoContext)+
   *new TMenuItem("~O~utput",cmLec_logicaOut,kbNoKey,hcNoContext)+
   *new TMenuItem("~I~nput",cmLec_logicaInp,kbNoKey,hcNoContext)+
   *new TMenuItem("~B~it",cmLec_logicaBit,kbNoKey,hcNoContext)+
   *new TMenuItem(
     *new TSubMenu("~T~imer",hcNoContext)+
     *new TMenuItem("~P~reset",cmLec_logicaTim1,kbNoKey,hcNoContext)+
     *new TMenuItem("~A~cumulado",cmLec_logicaTim2,kbNoKey,hcNoContext))+
   *new TMenuItem(
     *new TSubMenu("~C~ounter",hcNoContext)+
     *new TMenuItem("~P~reset",cmLec_logicaCou1,kbNoKey,hcNoContext)+
     *new TMenuItem("~A~cumulado",cmLec_logicaCou2,kbNoKey,hcNoContext))+
   *new TMenuItem("Cont~r~ol",cmLec_logicaCon,kbNoKey,hcNoContext)+
   *new TMenuItem("I~n~teger",cmLec_logicaInt,kbNoKey,hcNoContext)+
   *new TMenuItem("N:~9~",cmLec_no_proteg,kbNoKey,hcNoContext)+

 *new TSubMenu("~E~scribir PLC",hcNoContext)+
   *new TMenuItem("~B~it",cmEsc_logicaBit,kbNoKey,hcNoContext)+
   *new TMenuItem(
     *new TSubMenu("~T~imer",hcNoContext)+
     *new TMenuItem("~P~reset",cmEsc_logicaTim1,kbNoKey,hcNoContext)+
     *new TMenuItem("~A~cumulado",cmEsc_logicaTim2,kbNoKey,hcNoContext))+
   *new TMenuItem(
     *new TSubMenu("~C~ounter",hcNoContext)+
     *new TMenuItem("~P~reset",cmEsc_logicaCou1,kbNoKey,hcNoContext)+
     *new TMenuItem("~A~cumulado",cmEsc_logicaCou2,kbNoKey,hcNoContext))+
   *new TMenuItem("Cont~r~ol",cmEsc_logicaCon,kbNoKey,hcNoContext)+
   *new TMenuItem("I~n~teger",cmEsc_logicaInt,kbNoKey,hcNoContext)+
   *new TMenuItem("N:~9~",cmEsc_no_proteg,kbNoKey,hcNoContext)+

 *new TSubMenu("~P~ID",hcNoContext)+
  *new TMenuItem("~C~ontrol PID",cmMonitorear,kbF1,hcNoContext,"F1")+
  *new TMenuItem("~O~N/OFF",cmOnOff,kbF3,hcNoContext,"F3")+

 *new TSubMenu("~T~anque",hcNoContext)+
  *new TMenuItem("Tanque ~s~olo",cmTanque,kbF5,hcNoContext,"F5")+
  *new TMenuItem("Tanque ~c~ombinado",cmTanqueCom,kbF6,hcNoContext,"F6")+
```

```
 *new TSubMenu("~A~larmas",hcNoContext)+
  *new TMenuItem("~A~ctivar",cmActivar_alarm,kbF8,hcNoContext,"F8")+
  *new TMenuItem("~D~esactivar",cmDesactivar_alarm,kbF9,hcNoContext,"F9")+
  *new TMenuItem(
   *new TSubMenu("~F~ijar niveles",hcNoContext)+
    *new TMenuItem("Nivel ~S~uperior",cmAlarmaSup,kbNoKey,hcNoContext)+
    *new TMenuItem("Nivel ~I~nferior",cmAlarmaInf,kbNoKey,hcNoContext))+

 *new TSubMenu("Pa~n~talla",hcNoContext)+
  *new TMenuItem("Y M~a~ximo",cmEscalaY_max,kbCtrlF6,hcNoContext,"Ctrl-F6")+
  *new TMenuItem("Y M~i~nimo",cmEscalaY_min,kbCtrlF7,hcNoContext,"Ctrl-F7")+
  *new TMenuItem("~T~iempo",cmDotiempo,kbCtrlF8,hcNoContext,"Ctrl-F8")+

 *new TSubMenu("~I~nformación",hcNoContext)+
  *new TMenuItem("~A~yuda",cmAyuda,kbAltH,hcNoContext,"Alt-H")+
  *new TMenuItem("~G~rafico del proceso",cmInfo1,kbNoKey,hcNoContext));
}


TStatusLine *TBGIApp::initStatusLine(TRect r)
{
   r.a.y = r.b.y - 1;
   return new TStatusLine( r,
         *new TStatusDef( 0, 0xFFFF ) +
            *new TStatusItem( "~Alt-X~ Salir", kbAltX, cmQuit)+
            *new TStatusItem( "~F1~ Control PID", kbF1, cmMonitorear)+
            *new TStatusItem( "~F3~ On/Off", kbF3, cmOnOff)+
            *new TStatusItem( "~F10~ Menú",kbF10,cmMenu)
            );
}


void TBGIApp::outOfMemory()
{
   messageBox("Bloqueo de Memoria.", mfError | mfOKButton);
   getch();
}
```

```
/*****************************************************************************/
/*              ADQUISICION DE DATOS, SUPERVISION Y CONTROL PID              */
/*              UTILIZANDO CONTROLADORES LOGICOS PROGRAMABLES                */
/*                                                                           */
/*  ESCUELA POLITECNICA NACIONAL - FACULTAD DE INGENIERIA ELECTRICA          */
/*              DEPARTAMENTE DE ELECTRONICA Y CONTROL                        */
/*                                                                           */
/*  Autor:    LUIS GIOVANNI PERRASO BASANTES                                 */
/*  Director: ING. JORGE MOLINA                                              */
/*                                                                           */
/*  Fecha: Noviembre 1995                                                    */
/*  Modulo: FUNC-1.CPP                                                       */
 /*
                                                                            */
/*      Funciones de lectura y escritura protegida de la Data File 9         */
/*                                                                           */
 /*****************************************************************************/

// Archivos de inclusion
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include "krdefs-1.h"
#include "stddrv-1.h"
#include "datos.h"


// Variables externas
extern unsigned int    destino;
extern char            device[];

// Alusion a funcion
extern void     inicializar       (int lec_esc, int numero_datos,int lugar);
extern DAT_RET adquirir_dato  (void);
extern void     finalizar         (void);
extern void     ini_log           (int cmd,int fnc,int size,int fil_nu, int elem, int sub, int
dato);
extern DAT_RET comunicar_log (int x);
extern DAT_RET abrir_comunicacion (void);
        void    Get_ErrMsg      (unsigned int,char *);


/*****************************************************************************/
DAT_RET  Lec_df9 (int lugar, int numero_datos)
/*****************************************************************************/
{
  // Subrutina para lectura del archivo de datos N:9

  int         i;
  DAT_RET     enviar, datos_llegados;

  enviar.dr_error = 0;
```

```
  datos_llegados = abrir_comunicacion ();
  if (datos_llegados.dr_error)
  {
    strcpy (enviar.dr_mensaje,datos_llegados.dr_mensaje);
    enviar.dr_error=1;
    return enviar;
  }

  inicializar (0,numero_datos,lugar);
  setfillstyle (11,7);
  bar3d (0+20,0+20,getmaxx()-20,75,0,1);
  settextstyle (TRIPLEX_FONT,HORIZ_DIR,5);
  setcolor (14);
  outtextxy    (200,20,"Data File 9");

  do
  {
        datos_llegados=adquirir_dato();
        if (datos_llegados.dr_error)
        {
          strcpy (enviar.dr_mensaje,datos_llegados.dr_mensaje);
          enviar.dr_error=1;
          return enviar;
        }
        else
         {
                for (i = 0;i < numero_datos;i+=2)
                {
                  gotoxy (19,16-numero_datos/4+i/2);
                  printf ("N9:%3d = %6d",lugar+i,datos_llegados.dr_valor[i]);
                }
                for (i = 1;i < numero_datos;i+=2)
                {
                  gotoxy (42,16-numero_datos/4+i/2);
                  printf ("N9:%3d = %6d",lugar+i,datos_llegados.dr_valor[i]);
                }
         }
  }     while (!kbhit());
  finalizar();
  return enviar;
}


/*************************************************************************/
DAT_RET  Esc_df9(int lugar,int dato)
/*************************************************************************/
{
 // Subrutina para escritura de datos en archivo N:9.

 DAT_RET       enviar, datos_llegados;

 enviar.dr_error = 0;
```

```
 datos_llegados = abrir_comunicacion();
 if (datos_llegados.dr_error)
 {
  strcpy (enviar.dr_mensaje,datos_llegados.dr_mensaje);
  enviar.dr_error=1;
  return enviar;
 }

 inicializar (1,dato,lugar);

 datos_llegados = adquirir_dato();
 if (datos_llegados.dr_error)
 {
  strcpy (enviar.dr_mensaje,datos_llegados.dr_mensaje);
  enviar.dr_error=1;
  return enviar;
 }

 finalizar();
 strcpy (enviar.dr_mensaje,"\n\x3 Operación de escritura satisfactoria");
 return enviar;
}


/*****************************************************************************/
DAT_RET diag (int val_Evaluar_Nodo)
/*****************************************************************************/
{
  // Subrutina para identificar el tipo de PLC conectado en un nodo.

  DAT_RET          enviar, datos_llegados;

  datos_llegados = abrir_comunicacion();
  if (datos_llegados.dr_error)
  {
   strcpy (enviar.dr_mensaje,datos_llegados.dr_mensaje);
   enviar.dr_error=1;
   return enviar;
  }

  ini_log (0x06,0x03,2,0,0,0,val_Evaluar_Nodo);

  datos_llegados = comunicar_log(1);       // datos = ENTIRE
  if (datos_llegados.dr_error)
  {
   strcpy (enviar.dr_mensaje,datos_llegados.dr_mensaje);
   enviar.dr_error=1;
   return enviar;
  }
  strcpy (enviar.dr_mensaje,"\x3 Tipo de nodo: ");
  strcat (enviar.dr_mensaje,(unsigned char*)&datos_llegados.dr_valor[9+4]);
  strcat (enviar.dr_mensaje,(unsigned char*)&datos_llegados.dr_valor[9+5]);
```

```
  strcat (enviar.dr_mensaje,(unsigned char*)&datos_llegados.dr_valor[9+6]);
  strcat (enviar.dr_mensaje,(unsigned char*)&datos_llegados.dr_valor[9+7]);
  strcat (enviar.dr_mensaje,(unsigned char*)&datos_llegados.dr_valor[9+8]);
  strcat (enviar.dr_mensaje,(unsigned char*)&datos_llegados.dr_valor[9+9]);
  strcat (enviar.dr_mensaje,(unsigned char*)&datos_llegados.dr_valor[9+10]);
  strcat (enviar.dr_mensaje,(unsigned char*)&datos_llegados.dr_valor[9+11]);
  finalizar();
  enviar.dr_error=0;
  return enviar;
}
```

```
/*****************************************************************************/
/*              ADQUISICION DE DATOS, SUPERVISION Y CONTROL PID              */
/*              UTILIZANDO CONTROLADORES LOGICOS PROGRAMABLES                */
/*                                                                           */
/*  ESCUELA POLITECNICA NACIONAL - FACULTAD DE INGENIERIA ELECTRICA          */
/*              DEPARTAMENTE DE ELECTRONICA Y CONTROL                        */
/*                                                                           */
/*  Autor:    LUIS GIOVANNI PERRASO BASANTES                                 */
/*  Director: ING. JORGE MOLINA                                              */
/*                                                                           */
/*  Fecha: Noviembre 1995                                                    */
/*  Modulo: FUNC-2.C                                                         */
/*   Protected typed logical R&W with 3 address fields (file,
                                                        */
/*  element and sub-element. Use of Send_StdDrv().                           */
/*****************************************************************************/
```

```
// Archivos de inclusion
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "krdefs-1.h"
#include "stddrv-1.h"
#include "datos.h"

// Definiciones para comunicacion
#define  LOOPSIZE    64   /* Size of loop back data */
#define  LEN         0
#define  TYP         1
#define  DEST        2
#define  SRC         3
#define  CMD         4
#define  STS         5
#define  TNSL0       6
#define  TNSH0           7
#define  FNC         8
#define  SIZE        9
```

```
#define  FIL_NU        10
#define  TIPO          11
#define  ELEM          12
#define  SUB           13

// Variables externas
extern char           device[];
extern unsigned int   destino;
extern char           err_msg[];

// Alusion a funcion
extern  void    ini_log             (int cmd, int fnc, int size, int fil_nu,
                                     int elem, int sub, int dato);
extern  DAT_RET comunicar_log (int x);
extern  void    finalizar           (void);
extern  DAT_RET abrir_comunicacion (void);
        void    Get_ErrMsg          (unsigned int,char *);


/*************************************************************************/
DAT_RET Lec_log(int fil_nu, int elem, int sub, int size, int veces)
/*************************************************************************/
{
//      Subrutina para lectura de cualquier archivo de datos del PLC.
//      veces = 1 indica que se quede en lazo do-while 1 sola vez.
//      veces !=1 se debe quedar en lazo do-while hasta que se presione
//                una tecla.

  int       i,dato;
  char  titulo[15], simbolo[5];
  DAT_RET       enviar_datos, datos_llegados;

  switch (fil_nu)
  {
   case 0:    strcpy (titulo,"Salida");
              strcpy (simbolo,"O: ");
              break;
   case 1:    strcpy (titulo,"Entrada");
              strcpy (simbolo,"I: ");
              break;
   case 2:    strcpy (titulo,"Reservado");
              break;
   case 3:    strcpy (titulo,"Bit");
              strcpy (simbolo,"B3:");
              break;
   case 4:    if (sub==1) strcpy (titulo,'Timer Pre.");
              if (sub==2) strcpy (titulo,'Timer Acc. ");
              strcpy (simbolo,'T4:");
              break;
   case 5:    if (sub==1) strcpy (titulo,"Counter Pre.");
              if (sub==2) strcpy (titulo,"Counter Acc. ");
              strcpy (simbolo,"C5:");
              break;
```

```
  case 6:     strcpy (titulo,"Control");
              strcpy (simbolo,"R6:");
              break;
  case 7:     strcpy (titulo,"Entero");
              strcpy (simbolo,"N7:");
              break;
  default:    break;
}

datos_llegados = abrir_comunicacion();
if (datos_llegados.dr_error)
{
 strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
 enviar_datos.dr_error=1;
 return enviar_datos;
}

ini_log (0x0F,0xA2,size,fil_nu,elem,sub,0);

if (veces!=1)
{
      //      Cuadro de titulo
      setfillstyle (11,7);
      bar3d (20,150,getmaxx()-20,205,0,1);
      //      Cuadro de datos
      setfillstyle (1,0);
      if (fil_nu==0 | | fil_nu==1 | | fil_nu==3)
        bar3d (20,210,getmaxx()-20,330,0,1);
      else
        bar3d (20,210,getmaxx()-20,290,0,1);
      //      Titulo
      settextstyle (TRIPLEX_FONT,HORIZ_DIR,5);
      setcolor (14);
      settextjustify (CENTER_TEXT,TOP_TEXT);
      outtextxy (320,148,titulo);
      settextjustify (LEFT_TEXT,TOP_TEXT);
      settextstyle (DEFAULT_FONT,HORIZ_DIR,1);

      //      Numeros de leyenda para bits
      setcolor (8);
      if (fil_nu==0 | | fil_nu==1 | | fil_nu==3)
        {
              for (i=0;i<=15;i++)
              {
              sprintf   (titulo,"%d",i);
              outtextxy (90+i*32,275,titulo);
              }
              outtextxy (50,275,"Bit");
      }
}

do
{
```

```
  datos_llegados = comunicar_log(0);        // datos = DATA_ONLY
  if (datos_llegados.dr_error)
  {
   strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
   enviar_datos.dr_error=1;
   return enviar_datos;
  }

  if ((veces!=1)&&(fil_nu==0 | | fil_nu==1 | | fil_nu==3))
   {
        for (i=1;i<=8;i++)
             {
             gotoxy (8+i*4,19);
             printf ("%d",( (datos_llegados.dr_valor[0]&0x00ff) >>(i-1))&0x1);
             }
        for (i=1;i<=8;i++)
             {
             gotoxy (40+i*4,19);
             printf ("%d",( ((datos_llegados.dr_valor[0]&0xff00)>>8)  >>(i-1))&0x1);
             }
   }

  if (veces!=1)
  {
       gotoxy (32,16);
       printf ("%s %d = %6d",simbolo,elem,datos_llegados.dr_valor[0]);
  }

  } while (!kbhit()&&veces!=1);

  finalizar();
  enviar_datos.dr_error=0;
  return enviar_datos;
}


/*************************************************************************/
DAT_RET Esc_log(int fil_nu, int elem, int sub, int dato, int size)
/*************************************************************************/
{
   // Subrutina para escritura de datos en cualquier archivo de datos
   // del PLC.

   DAT_RET   enviar_datos, datos_llegados;

   datos_llegados = abrir_comunicacion();
   if (datos_llegados.dr_error)
   {
    strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
    enviar_datos.dr_error=1;
    return enviar_datos;
   }
```

```
   ini_log (0x0F,0xAA,size,fil_nu,elem,sub,dato);

   datos_llegados = comunicar_log(0);        // datos = DATA_ONLY
   if (datos_llegados.dr_error)
   {
    strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
    enviar_datos.dr_error=1;
    return enviar_datos;
   }

   strcpy (enviar_datos.dr_mensaje,"Operación de escritura logica satisfactoria");
   finalizar();
   enviar_datos.dr_error = 0;
   return enviar_datos;
}
```

```
   /*****************************************************************************/
   /*              ADQUISICION DE DATOS, SUPERVISION Y CONTROL PID              */
   /*              UTILIZANDO CONTROLADORES LOGICOS PROGRAMABLES                */
   /*                                                                           */
   /*  ESCUELA POLITECNICA NACIONAL - FACULTAD DE INGENIERIA ELECTRICA          */
   /*              DEPARTAMENTE DE ELECTRONICA Y CONTROL                        */
   /*                                                                           */
   /*  Autor:    LUIS GIOVANNI PERRASO BASANTES                                 */
   /*  Director: ING. JORGE MOLINA                                              */
   /*                                                                           */
   /*  Fecha:  Noviembre 1995                                                   */
   /*  Modulo: FUNC-3.C                                                         */
   /*                                                                           */
    /*****************************************************************************/
```

```
// Archivos de inclusion
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include "krdefs-1.h"
#include "stddrv-1.h"
#include "datos.h"

// Definiciones para comunicacion
#define  LOOPSIZE    64   /* Size of loop back data */
#define  LEN         0
#define  TYP         1
#define  DEST        2
#define  SRC         3
#define  CMD         4
#define  STS         5
#define  TNSL0       6
```

```
#define TNSH0           7
#define FNC         8
#define SIZE        9
#define FIL_NU      10
#define TIPO        11
#define ELEM        12
#define SUB         13
#define DATO_L      14
#define DATO_H      15


// Alusion a funcion
void Get_ErrMsg(unsigned int,char *);

// Variables globales y externas
   unsigned char    d_tab[2];
   unsigned char    s_buff[LOOPSIZE], r_buff[LOOPSIZE];
   unsigned int     tns, fct_stat1, io_stat1[2];
   struct SD_FB     DHP_MSG1;
   int              PLC_FCT1;
   extern char      device[];
   extern int       destino;
   extern char      err_msg[];

DAT_RET abrir_comunicacion ()
{
        DAT_RET             enviar_datos;

        fct_stat1 = Open_StdDrv(device,0,0,0,(struct kr_unsol_msg *)NULL,0,0,0);
        if (fct_stat1 != NORMAL)
          {
            Get_ErrMsg(fct_stat1,err_msg);
            enviar_datos.dr_error = 1;
            strcpy (enviar_datos.dr_mensaje,err_msg);
            return enviar_datos;
          }
        enviar_datos.dr_error = 0;
        return enviar_datos;
}

void inicializar (int a, int b, int c)
{
        // lectura  : a = 0, b = numero de datos,  c = lugar
        // escritura: a = 1, b = valor a escribir, c = lugar

        d_tab[0]    = c;    /* Offset into data table (low byte) */
        d_tab[1]    = 0x00;     /* Offset into data table (high byte) */
        DHP_MSG1.dev = device;   /* "KR:0" ASCII string */
        DHP_MSG1.stat= io_stat1;  /* Pointer to user's reply status flag */
        DHP_MSG1.L_R = 0;  /* Indicate on-link addressing */
        DHP_MSG1.dst = (unsigned char*)destino;     /* Pointer to network destination
                                                       address */
        DHP_MSG1.dta = d_tab;    /* Pointer to data table nine offset address */
```

```
        DHP_MSG1.TO  = 5;  /* Time, in seconds, to wait for reply from SLC */

        if (a==0)         // lectura no protegida
        {
          DHP_MSG1.len = b*2;              // 2 x numero de bytes a leer
          DHP_MSG1.buf = r_buff;    // Pointer to buffer where data received
          PLC_FCT1     = PLC2_URD;  // Load selection code for Unprotected Read
                                                command
        }

        if (a==1)         // escritura no protegida
        {
          r_buff[0]      = b&0x00FF;
          r_buff[1]      = (b&0xFF00)>>8;
          DHP_MSG1.len = 2;
          DHP_MSG1.buf = r_buff;
          PLC_FCT1     = PLC2_UWR;
        }
}


DAT_RET  adquirir_dato()
{
        int       i;
        DAT_RET         enviar_datos;

        fct_stat1 = Appl_StdDrv(PLC_FCT1,&DHP_MSG1);     /* Send command */
          if (fct_stat1 != NORMAL)
           {
           Get_ErrMsg(fct_stat1,err_msg);
           fct_stat1 = Close_StdDrv(device);
           strcpy (enviar_datos.dr_mensaje,err_msg);
           enviar_datos.dr_error=1;
           return enviar_datos;
           }

        while (!io_stat1[0]);          /* Wait for reply or timeout */
        if (io_stat1[0] != NORMAL)
         {
           Get_ErrMsg(io_stat1[0],err_msg);
           fct_stat1 = Close_StdDrv(device);
           enviar_datos.dr_error=1;
           strcpy (enviar_datos.dr_mensaje,err_msg);
           return enviar_datos;
         }

        for (i=0;i<31;i++)
          enviar_datos.dr_valor[i] = (r_buff[2*i]&0x00ff) + ( r_buff[2*i+1]<<8 & 0xff00 );

        enviar_datos.dr_error=0;
        return enviar_datos;
}
```

```
void ini_log(int cmd, int fnc, int size, int fil_nu, int elem, int sub, int dato)
{
  unsigned char     tipo;
  unsigned int      tns;

  switch (fil_nu)
  {
    case 0:    tipo=0x82;
               break;
    case 1:    tipo=0x83;
               break;
    case 2:    tipo=0x84;
               break;
    case 3:    tipo=0x85;
               break;
    case 4:    tipo=0x86;
               break;
    case 5:    tipo=0x87;
               break;
    case 6:    tipo=0x88;
               break;
    case 7:    tipo=0x89;
               break;
    default:   break;
  }

  s_buff[TYP]   = 0;                              // Tipo

  if (cmd==0x06 && fnc==0x03)
  {
       s_buff[LEN]  = 9;
       s_buff[DEST] = (unsigned char) dato;
  }

  else
  {
       if  (cmd==0x0F && fnc==0xAA)
          s_buff[LEN]  = 16;
       else
          s_buff[LEN]  = 64;

       s_buff[DEST]   = destino;     // Direcc. Destino
       s_buff[SIZE]   = size;          // Tamaao en bytes de la palabra
       s_buff[FIL_NU] = fil_nu;
       s_buff[TIPO]   = tipo;
       s_buff[ELEM]   = elem;
       s_buff[SUB]    = sub;
  }

  s_buff[SRC]   = 0;                              // Direcc. 1784-KR
  s_buff[CMD]   = (unsigned char) cmd;
  s_buff[STS]   = 0;
```

```
  tns          = get_tns();
  s_buff[TNSL0]  = (char) (tns & 0xff);          // TNS (low byte)
  s_buff[TNSH0]  = (char) ((tns >> 8) & 0xff);   // TNS (high byte)
  s_buff[FNC]    = (unsigned char) fnc;


  if (cmd==0x0F && fnc==0xAA)
  {
   s_buff[DATO_L] = dato&0x00ff;
   s_buff[DATO_H] = (dato&0xff00)>>8;
  }
}


DAT_RET comunicar_log (int x)
{
  //     x = 0   DATA_ONLY
  //     x = 1   ENTIRE

  DAT_RET         enviar_datos;
  int             i;

  fct_stat1 = Send_StdDrv(device,io_stat1,s_buff,x,r_buff,5,0,0);
  if (fct_stat1 != NORMAL)
   {
    Get_ErrMsg(fct_stat1,err_msg);
    strcpy (enviar_datos.dr_mensaje,err_msg);
    enviar_datos.dr_error=1;
    fct_stat1 = Close_StdDrv(device);
    return enviar_datos;
  }

  while (!io_stat1[0]);   // Wait for reply from device or timeout

  if (io_stat1[0] != NORMAL)
   {
    Get_ErrMsg(io_stat1[0],err_msg);
    strcpy (enviar_datos.dr_mensaje,err_msg);
    enviar_datos.dr_error=1;
    fct_stat1 = Close_StdDrv(device);
    return enviar_datos;
   }
  enviar_datos.dr_error=0;

  if (x==0)
  {
   for (i=0;i<31;i++)
     enviar_datos.dr_valor[i] = (r_buff[2*i]&0x00ff) + ( r_buff[2*i+1]<<8 & 0xff00 );
  }
  else
  {
   for (i=0;i<64;i++)
     enviar_datos.dr_valor[i]=r_buff[i];
```

```
  }

  return enviar_datos;
}


void finalizar()
{
 fct_stat1 = Close_StdDrv(device);
}


   /*****************************************************************************/
   /*                ADQUISICION DE DATOS, SUPERVISION Y CONTROL PID            */
   /*                UTILIZANDO CONTROLADORES LOGICOS PROGRAMABLES               */
   /*                                                                           */
   /*  ESCUELA POLITECNICA NACIONAL - FACULTAD DE INGENIERIA ELECTRICA          */
   /*                DEPARTAMENTE DE ELECTRONICA Y CONTROL                      */
   /*                                                                           */
   /*  Autor:    LUIS GIOVANNI PERRASO BASANTES                                 */
   /*  Director: ING. JORGE MOLINA                                              */
   /*                                                                           */
   /*  Fecha: Noviembre 1995                                                    */
   /*  Modulo: PID.C                                                            */
   /*                                                                           */
    /*****************************************************************************/

// Archivos de inclusion
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <bios.h>
#include <dos.h>
#include <string.h>
#include <time.h>
#include <ctype.h>
#include "krdefs.h"
#include "stddrv.h"
#include "datos.h"

// Definiciones para comunicacion
#define  DATALEN     4     /* Para leer 2 data file 9 */
#define  LOOPSIZE    64     /* Size of loop back data  */
#define  ENTER       0x0d    // Definicion para funcion de lectura de caracteres
#define  LEN         0
#define  TYP         1
#define  DEST        2
#define  SRC         3
#define  CMD          4
#define  STS         5
#define  TNSL0       6
#define  TNSH0             7
#define  FNC         8
```

```
#define  SIZE          9
#define  FIL_NU        10
#define  TIPO          11
#define  ELEM          12
#define  SUB           13

// Variables globales */
char            device[] = "KR:0",
                err_msg[80];
unsigned int    fct_stat,destino,
                io_stat[2];
extern float    tiempo_x, alarma_sup, alarma_inf, ejey_max, ejey_min;
extern int      activar_alarma;


//Alusion a funciones
void        Get_ErrMsg          (unsigned int,char *);
extern void   inicializar       (int lec_esc, int numero_datos, int lugar);
extern DAT_RET adquirir_dato    (void);
extern void   finalizar         (void);
extern void   ini_log    (int cmd, int fnc, int size, int fil_nu,
                          int elem, int sub, int dato);
extern DAT_RET comunicar_log        (int x); //x=0 DATA_ONLY
extern DAT_RET abrir_comunicacion (void);


void valores (int x, int y, float z, int decimales, int color)
{
 /* (x,y)=posicion              */
 /* z=valor a presentar         */
 char valor[80];
 int color_anterior;
 color_anterior=getcolor();
 setcolor (color);
 if (decimales==2)
  sprintf (valor,"%.2f",z);
 else
  sprintf (valor,"%.1f",z);
 outtextxy (x,y,valor);
 setcolor(color_anterior);
}


void grilla(float b)
{
 int i,x2=30, color_anterior;
 color_anterior = getcolor();
 setcolor(4);
 setlinestyle (DOTTED_LINE,1,1);
 for (i=30;i<540;i+=30)
 {
  if (i>30)
    line (i,30,i,330);              // grilla vertical
```

```
   if (i<=360)
     line (60,i,510,i);              // grilla horizontal
  }
  setlinestyle (SOLID_LINE,1,1);
  settextjustify (RIGHT_TEXT,0);
  settextstyle (SMALL_FONT,HORIZ_DIR,4);

  for (i=10;i>=0;i--)
   {
     valores (47,x2,(ejey_min+i*(ejey_max-ejey_min)/b/10.0),2,2);
     valores (545,x2,i,1,14);
     x2+=30;
   }
  setcolor (color_anterior);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
}


void imprimir_const(float dato,int decimales,int posy,struct viewporttype *va,char *s)
{
        setviewport (0,0,getmaxx(),getmaxy(),1);
        setviewport (220,posy,290,posy+9,1);
        clearviewport();
        settextstyle (SMALL_FONT,HORIZ_DIR,4);
        settextjustify (0,0);
        setcolor (11);
        outtextxy (2,8,s);
        valores (32,8,dato,decimales,11);
        setviewport (0,0,getmaxx(),getmaxy(),1);
        setviewport ((*va).left,(*va).top,(*va).right,(*va).bottom,1);
}


void escala_x(float tiempo_ant)
{
  int i;
  float valor;
  settextstyle (SMALL_FONT,HORIZ_DIR,4);
  valor = tiempo_x/15.0;
  for (i=0;i<16;i+=2)      // Pone escala de tiempo saltando una division
    valores (70+30*i,343,(float)i*valor+tiempo_ant,1,7);
}


void alarma(int eleccion,struct viewporttype *va )
{
        int color_ant;
        color_ant = getcolor();
        setviewport (0,0,getmaxx(),getmaxy(),1);
        if (eleccion==1)
        {
                setcolor (4);
                outtextxy (65,78,"Û");
```

```
                sound (300);
                delay (25);
                nosound();
                setcolor (0);
                outtextxy (65,78,"Û");
        }
        if (eleccion==2)
        {
                setcolor (4);
                outtextxy (65,90,"Û");
                sound (300);
                delay (25);
                nosound();
                setcolor (0);
                outtextxy (65,90,"Û");
        }
        setviewport ((*va).left,(*va).top,(*va).right,(*va).bottom,1);
        setcolor (color_ant);
}


float leer_flot ( float valor_ant, float minimo, float maximo, int digitos,
                  int xcoord, int ycoord )
{
 /* Declaración de variables. */

 int j, flag, flag1;     /* Variables auxiliares.                    */
 char valor[10];         /* Arreglo que guarda los dígitos leídos.   */
 char c;                 /* Variable auxiliar para lectura de un caracter.*/
 char *puntero;          /* Puntero al arreglo " valor ".            */
 float nuevo_valor;      /* Nuevo valor de la variable.              */

 puntero = &valor[0];    /* Inicialización del puntero al arreglo "valor".*/

 /* Lazo principal de lectura de la variable en forma de string. */

 do
 {
  flag = 1;  /* Bandera que indica si se ha leído un número correcto.    */

  /* Lectura del primer dígito. Si se presiona ENTER se devuelve el valor */
  /* original de la variable.                                             */
  do
  {
   fflush ( stdin );
   gotoxy ( xcoord, ycoord );
   c = getch();
   if ( c == ENTER )
   {
        gotoxy ( xcoord, ycoord );
        return valor_ant;
   }
  } while ( !isdigit(c) && c!='-' );
```

```
gotoxy ( xcoord, ycoord );
printf ("%c",c);
gotoxy ( xcoord + 1, ycoord ); printf ("     ");
valor[0] = c;

/* Lazo de lectura de los dem s d¡gitos. */
for ( j = 1; j < digitos; j++ )
{
  do
  {
       fflush ( stdin ); flag1 = 0;
       gotoxy ( xcoord + j, ycoord );
       c = getch();
       if ( isdigit(c) ) flag1 = 1;
       if ( c == '.' )   flag1 = 1;
       if ( ENTER == c ) flag1 = 1;
  } while ( flag1 == 0 );

  if ( ENTER == c ) break;

  gotoxy ( xcoord + j, ycoord );
  printf ("%c",c);
  valor[j] = c;
}

valor[j] = '\0';

/* Conversión del string le¡do a un número en punto flotante. */
nuevo_valor = atof(puntero);

/* Validación del valor le¡do. Si existe un error despliega un mensaje y*/
/* vuelve a leer el valor.                                          */
if ( (nuevo_valor > maximo) || (nuevo_valor < minimo) )
{
  putch  ('\a'); flag = 0;
  window ( 16,15,74,19 ); textattr (0);
  clrscr ();
  gotoxy ( 12,1 ); printf ("VALOR FUERA DE RANGO");
  gotoxy ( 1,2 );  printf ("————————————————————————————");
  gotoxy ( 2,3 );
  printf ("Ingrese un valor entre %.4f y %.4f ", minimo, maximo );
  gotoxy ( 2,4 ); printf ("Valor ingresado = %.4f",nuevo_valor );
  gotoxy ( 2,5 ); printf ("Presione una tecla para continuar...");
  getch  ();
  clrscr ();
  window (1,1,80,25);
  gotoxy ( xcoord, ycoord );
}

} while ( flag == 0 );

gotoxy ( xcoord, ycoord );
```

```
 return nuevo_valor*1.00001;      /* Retorna el valor leído. */

}


DAT_RET plc (void)
{

  clock_t       inicio1, fin1;
  int           key, auto_man=1, metodo;
  char namef[30];
  float y1[2000], y2[2000], a, b=1.0, tiempo_ant=0,tiempo_nue=0,
                kp,ki,kd,ref,i,i2,punto_a, punto_b, punto_c,dato;
  DAT_RET       enviar_datos, datos_llegados;
  FILE  *fp;
  structviewporttype va;
  structdate        fecha;
  structtime        hora;

  enviar_datos.dr_error=0;

  //    Presentacion de la pantalla
  setfillstyle (11,8);
  setcolor      (0);
  bar3d         (0,0,getmaxx(),getmaxy(),0,1);
  setfillstyle (SOLID_FILL,0);
  setcolor      (7);
  bar3d         (49,99,601,451,0,1);  // Pantalla para graficacion de datos
  bar3d         (49,9,600,64,2,1);          // Pantalla de menu y constantes

  setcolor  (12);
  outtextxy (55,16,"[ESC] Salir");
  setcolor  (7);
  outtextxy (55,26,"[P]   Pausa");
  outtextxy (180,16,"[F1]");
  outtextxy (180,26,"[F2]");
  outtextxy (180,36,"[F3]");
  outtextxy (180,46,"[F4]");
  outtextxy (55,36,"[+/-] Escala");

  setcolor  (7);
  bar3d     (56,69,65,78,0,1);
  bar3d     (56,81,65,90,0,1);
  outtextxy (55,70,"  Sobre nivel");
  outtextxy (55,82,"  Bajo  nivel");

  settextstyle (TRIPLEX_FONT,HORIZ_DIR,1);
  setcolor      (3);
  outtextxy     (26,453,"SISTEMA DE ADQUISICION DE DATOS, SUPERVISION Y CONTROL
                        PID");
  settextstyle (SMALL_FONT,HORIZ_DIR,4);
  outtextxy     (525,82,"EPN-FIE-LGPB");
  settextstyle (DEFAULT_FONT,HORIZ_DIR,1);
```

```
//      Define pantalla para graficos
getviewsettings (&va);

//      Abre comunicacion con PLC
datos_llegados = abrir_comunicacion();

if (datos_llegados.dr_error)
{
  closegraph ();
  enviar_datos.dr_error = 1;
  strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
  return enviar_datos;
}

//      Lee en variable "metodo", el tipo de programa que esta siendo
//      ejecutado por el PLC, de acuerdo a:
//      N7:41 = metodo = 1, entonces, metodo discreto
//      N7:41 = metodo = 2, entonces, metodo continuo
//      N7:41 = metodo = 3, entonces, metodo PID 5/02

ini_log (0x0F,0xA2,2,7,41,0,0);
datos_llegados = comunicar_log(0);
if (datos_llegados.dr_error)
{
  closegraph();
  enviar_datos.dr_error = 1;
  strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
  return enviar_datos;
}
metodo=datos_llegados.dr_valor[0];


//      Lee valor de referencia actual. En PID discretizado, la referencia
//      tiene un rango de [0,32767]. En PID instruccion, tiene un rango de
//      [0,1000]. REF = N7:2

ini_log (0x0F,0xA2,2,7,2,0,0);
datos_llegados = comunicar_log(0);
if (datos_llegados.dr_error)
{
  closegraph();
  enviar_datos.dr_error = 1;
  strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
  return enviar_datos;
}
if (metodo==1 | |metodo==2) ref=(float)(datos_llegados.dr_valor[0]*(ejey_max-
                                                    ejey_min)/32767.0+ejey_min);
if (metodo==3) ref=(float)(datos_llegados.dr_valor[0]*(ejey_max-
                                          ejey_min)/1000.0+ejey_min);
imprimir_const (ref,2,14,&va,"Ref= ");

//      Lee valor de constante Kp actual. En PID discretizado, esta constante
```

```
//      tiene 2 numeros decimales. En PID instruccion, tiene solo 1 decimal
//      Kp = N7:3

ini_log (0x0F,0xA2,2,7,3,0,0);
datos_llegados = comunicar_log(0);
if (datos_llegados.dr_error)
{
  closegraph();
  enviar_datos.dr_error = 1;
  strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
  return enviar_datos;
}
if (metodo==1 | | metodo==2) kp=(float)datos_llegados.dr_valor[0]/100.0;
if (metodo==3) kp=(float)datos_llegados.dr_valor[0]/10.0;
imprimir_const (kp,2,24,&va,"Kp = ");


//      Lee valor de constante Ki actual. En PID discretizado, esta constante
//      tiene 2 numeros decimales. En PID instruccion, tiene solo 1 decimal
//      Ki = N7:4

ini_log (0x0F,0xA2,2,7,4,0,0);
datos_llegados = comunicar_log(0);
if (datos_llegados.dr_error)
{
  closegraph();
  enviar_datos.dr_error = 1;
  strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
  return enviar_datos;
}
if (metodo==1 | | metodo==2) ki=(float)datos_llegados.dr_valor[0]/100.0;
if (metodo==3) ki=(float)datos_llegados.dr_valor[0]/10.0;
imprimir_const (ki,2,34,&va,"Ki = ");


//      Lee valor de constante Kd actual. En PID discretizado, la constante
//      Kd tiene una precision de 2 numeros decimales. En instruccion PID,
//      tambien tiene 2 numeros decimales. Kd = N7:5

ini_log (0x0F,0xA2,2,7,5,0,0);
datos_llegados = comunicar_log(0);
if (datos_llegados.dr_error)
{
  closegraph();
  enviar_datos.dr_error = 1;
  strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
  return enviar_datos;
}
kd=(float)datos_llegados.dr_valor[0]/100.0;
imprimir_const (kd,2,44,&va,"Kd = ");

setcolor (12);
if (metodo==1)   outtextxy (485,27,"PID Algoritmo Disc");
```

```
if (metodo==2)  outtextxy (478,27,"PID Algoritmo Cont");
if (metodo==3)  outtextxy (478,27,"PID Instrucción");
settextstyle (DEFAULT_FONT,HORIZ_DIR,2);


//  Lee N7:40 para comprobar estado del control
//     N7:40=1 Activado(auto);      N7:40=0 Desactivado(manual)
//     Cuando metodo=3, no hay modo manual

if (metodo!=3)
{
 ini_log (0x0F,0xA2,2,7,40,0,0);
 datos_llegados = comunicar_log(0);
 if (datos_llegados.dr_error)
 {
   closegraph();
   enviar_datos.dr_error = 1;
   strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
   return enviar_datos;
 }
 setcolor (13);
 auto_man = datos_llegados.dr_valor[0];
 if (auto_man==1)
       outtextxy (505,50,"Auto");
 if (auto_man==0)
       {
         outtextxy (500,50,"Man");
         settextstyle (DEFAULT_FONT,HORIZ_DIR,1);
         outtextxy (55,60,"[Arr/Aba] REF");
       }
}

setviewport (50,100,600,450,1);
getviewsettings (&va);

settextjustify (RIGHT_TEXT,0);
settextstyle  (DEFAULT_FONT,HORIZ_DIR,1);

setcolor    (2);
outtextxy   (450,15,"Tiempo(s)=");
valores     (500,15,tiempo_x,2,2);
outtextxy   (450,25,"Seg/div  =");
valores     (500,25,tiempo_x/15.0,2,2);


//     Lazo principal del programa

while(key!=0x11b)            // Tecla [ESC] termina el programa
{
 while(bioskey(1)==0)
  {
       grilla       (b);
       setcolor     (14);
```

```
        outtextxy   (134,16,"Señal de");
        outtextxy   (134,25,"Ctrl (V)");
        setcolor    (10);
        outtextxy   (290,16,"Variable ");
        outtextxy   (290,25,"de Salida ");
        settextstyle (SMALL_FONT,HORIZ_DIR,4);
        setcolor    (7);
        outtextxy   (540,14,"(v)");
        outtextxy   (520,343,"(s)");

        tiempo_nue+=tiempo_ant;
        escala_x(tiempo_nue);

        setcolor    (9);
        settextstyle (DEFAULT_FONT,HORIZ_DIR,1);

        //      Variable 'a' sirve para calibrar el tiempo de barrido de una
        //      pantalla. Escala del eje de tiempo. Difiere con cada PC.

        a=450.0*0.085/tiempo_x;

        //      Dibuja linea de referencia
        setcolor (12);
        if (330.0-(30.0*b*(ref*10/(ejey_max-ejey_min)+ejey_min))>=30)
        line (60,330.0-(30.0*b*(ref*10/(ejey_max-ejey_min)+ejey_min)),510,330.0-
                        (30.0*b*(ref*10/(ejey_max-ejey_min)+ejey_min)));

        inicio1=clock();
        for (i=1.0;i<450.0;i+=a)
            {
                i2=i;

                //      Adquiere datos N9:0 y N9:1
                inicializar (0,2,0);
                datos_llegados = adquirir_dato();
                if (datos_llegados.dr_error)
                {
                  closegraph();
                  enviar_datos.dr_error = 1;
                  strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
                  return enviar_datos;
                }

                //      Escalamiento de datos recibidos
                                        y1[i]=datos_llegados.dr_valor[0]*(ejey_max-
ejey_min)/32767.0+ejey_min;
                y2[i]=datos_llegados.dr_valor[1]/32767.0*10.0;


              if (y1[i]<0) y1[i]=0;
                if (y2[i]<0) y2[i]=0;

                if (activar_alarma==1)
```

```
            {
              if (y1[i]>=10.0*(alarma_sup-ejey_min)/(ejey_max-ejey_min))
                    {
                     alarma(1,&va);
                    }
              if (y1[i]<=10.0*(alarma_inf-ejey_min)/(ejey_max-ejey_min))
                    {
                     alarma(2,&va);
                    }
            }

            gotoxy (26,8); printf ("%4.2f",y2[i]);
            gotoxy (44,8); printf ("%4.2f",y1[i]);

            punto_a = i+60.0;
            punto_b = 330.0-(30.0*b*(y1[i]*10/(ejey_max-ejey_min)+ejey_min));
            punto_c = 330.0-(30.0*y2[i]);

            if (punto_b>=30.0) putpixel ((int)punto_a,(int)punto_b,10);
            if (punto_c>=30.0) putpixel ((int)punto_a,(int)punto_c,14);

            if (kbhit())
              break;
          }

       fin1=clock();
       tiempo_ant=(fin1-inicio1)/CLK_TCK;

       clearviewport      ();
       setcolor           (2);
       outtextxy          (450,15,"Tiempo(s)=");
       valores            (500,15,tiempo_ant,2,2);
       outtextxy          (450,25,"Seg/div  =");
       valores            (500,25,tiempo_ant/15.0,2,2);

 }
key=bioskey(0);

if (key==0x4800 && auto_man==0)          // flecha arriba
     {
       ref = ref+0.05*(ejey_max-ejey_min);
       if (ref>ejey_max) ref=ejey_max;

       if (metodo==1 | | metodo==2)        // PID algoritmo
        {
          //    Escribe referencia en N7:2. Rango [0,32767]
          ini_log (0x0F,0xAA,2,7,2,0,(int)((ref-ejey_min)*32767.0/(ejey_max-ejey_min)));
          datos_llegados=comunicar_log(0);
          if (datos_llegados.dr_error)
           {
            closegraph();
            enviar_datos.dr_error = 1;
            strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
```

```
            enviar_datos.dr_error = 1;
            strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
            return enviar_datos;
           }
           //   Escribe referencia en N7:30. Rango [0,32767]
           ini_log (0x0F,0xAA,2,7,30,0,(int)((ref-ejey_min)*32767.0/(ejey_max-ejey_min)));
           datos_llegados=comunicar_log(0);
           if (datos_llegados.dr_error)
            {
             closegraph();
             enviar_datos.dr_error = 1;
             strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
             return enviar_datos;
            }
          }

         if (metodo==3)        // PID instruccion
          {
           //   Escribe referencia en N7:2. Rango [0,1000]
           ini_log (0x0F,0xAA,2,7,2,0,(int)((ref-ejey_min)*1000.0/(ejey_max-ejey_min)));
           datos_llegados=comunicar_log(0);
           if (datos_llegados.dr_error)
            {
             closegraph();
             enviar_datos.dr_error = 1;
             strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
             return enviar_datos;
            }
           //   Escribe referencia en N7:16. Rango [0,16383]
           ini_log (0x0F,0xAA,2,7,16,0,(int)((ref-ejey_min)*16383.0/(ejey_max-ejey_min)));
           datos_llegados=comunicar_log(0);
           if (datos_llegados.dr_error)
            {
             closegraph();
             enviar_datos.dr_error = 1;
             strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
             return enviar_datos;
            }

          }
         imprimir_const (ref,2,14,&va,"Ref= ");
        }

if (key==0x4e2b)                    // signo "+" aumenta escala
     { b+=1;
     }
if ((key==0x4a2d)&&(b>1))           // signo "-" disminuye escala
     { b-=1;
     }
if ((key==6512)||(key==6480))       // p=pausa
     {
        //   Grafica datos leidos hasta el momento de presionar tecla P
        grilla(b);
```

```
escala_x(tiempo_nue);
settextstyle (DEFAULT_FONT,HORIZ_DIR,1);

for (i=1;i<=i2;i+=a)
 {
      punto_a = i+60;
      punto_b = 330.0-(30.0*b*y1[i]);
      punto_c = 330.0-(30.0*b*y2[i]);
      if (punto_b>30) putpixel ((int)punto_a,(int)punto_b,10);
      if (punto_c>30) putpixel ((int)punto_a,(int)punto_c,14);
 }
setcolor (14);
settextstyle (SMALL_FONT,HORIZ_DIR,5);
outtextxy (510,48,"[G]   Grabar");

key=bioskey(0);
if ((key==8807) | | (key==8775))   // Tecla [G] graba datos
 {
      gotoxy (20,10);
      printf ("Ingrese nombre del archivo...");
      scanf("%10s",namef);
      fp=fopen (namef,"w");
      fprintf (fp,"\t SISTEMA DE ADQUISICION DE DATOS, SUPERVISION Y
                        CONTROL\n");
      fprintf (fp,"\t    PID UTILIZANDO CONTROLADORES LOGICOS
                              PROGRAMABLES\n");
      fprintf (fp,"\t\t\tPor: Giovanni Perraso B.\n\n");
      fprintf (fp,"Archivo: %s\t\t",namef);
      getdate(&fecha);
      fprintf (fp,"Fecha: ");
      fprintf (fp,"%d-",fecha.da_year);
      fprintf (fp,"%.2d-",fecha.da_mon);
      fprintf (fp,"%.2d\t\t",fecha.da_day);
      gettime(&hora);
      fprintf (fp,"Hora: ");
      fprintf (fp,"%dh",hora.ti_hour);
      fprintf (fp,"%.2d:",hora.ti_min);
      fprintf (fp,"%.2d\n\n",hora.ti_sec);
      if (metodo==1 | | metodo==2) fprintf (fp,"PID ALGORITMO\n");
      if (metodo==3) fprintf (fp,"PID INSTRUCCION\n");
      fprintf (fp,"Constantes del controlador PID:\n");
      fprintf (fp,"Ref= %.2f\t",ref);
      if (metodo==1 | | metodo==2)
              {
                      fprintf (fp,"Kp = %.2f\t",kp);
                      fprintf (fp,"Ki = %.2f\t",ki);
              }
      if (metodo==3)
              {
                      fprintf (fp,"Kp = %.1f\t",kp);
                      fprintf (fp,"Ki = %.1f\t",ki);
              }
      fprintf (fp,"Kd = %.2f\n",kd);
```

```
                fprintf (fp,"Tiempo total de grabación = %.2f seg.\n\n",tiempo_ant);
                fprintf (fp,"Salida\tControl\n");
                fprintf (fp,"(V)\t(V)\n");
                for (i=1;i<=i2;i+=a)
                 {
                   fprintf (fp,"%3.1f\t",y1[i]);
                   fprintf (fp,"%3.1f\n",y2[i]);
                 }
                fclose (fp);
                sound (1000);
                delay (200);
                nosound ();
              }
             clearviewport();
          }

    if (key==0x3b00)              // Tecla F1 cambia set point
          {
             gotoxy (12,10);
             clearviewport();
             printf ("Ingrese nueva referencia...");
             dato = leer_flot(ref,ejey_min,ejey_max,5,50,10);
             ref  = dato;

             if (metodo==1 || metodo==2)
              {
                  ini_log (0x0F,0xAA,2,7,2,0,(int)((ref-ejey_min)/(ejey_max-
                                  ejey_min)*32767.0));
                  datos_llegados = comunicar_log(0);
                  if (datos_llegados.dr_error)
                  {
                    closegraph();
                    enviar_datos.dr_error = 1;
                    strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
                    return enviar_datos;
                  }
              }

             if (metodo==3)
              {
                  ini_log                    (0x0F,0xAA,2,7,2,0,(int)((ref-ejey_min)/(ejey_max-
ejey_min)*1000.0));
                  datos_llegados = comunicar_log(0);
                  if (datos_llegados.dr_error)
                  {
                    closegraph();
                    enviar_datos.dr_error = 1;
                    strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
                    return enviar_datos;
                  }
              }

             imprimir_const ((int)(ref*100.0)/100.0,2,14,&va,"Ref=");
```

```
        clearviewport();
    }

if (key==0x3c00)              // Tecla F2 cambia KP
    {
        gotoxy (12,10);
        clearviewport();
        printf ("Ingrese nueva constante KP...");
        if (metodo==1 | | metodo==2) dato = leer_flot (kp,0.0,327.6,5,47,10);
        if (metodo==3) dato = leer_flot (kp,0.1,25.5,4,47,10);
        kp = dato;
        if (metodo==1 | | metodo==2)
            {
             ini_log (0x0F,0xAA,2,7,3,0,(int)(kp*100.0));
             datos_llegados = comunicar_log(0);
             if (datos_llegados.dr_error)
              {
               closegraph();
               enviar_datos.dr_error = 1;
               strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
               return enviar_datos;
              }
             imprimir_const ((int)(kp*100.0)/100.0,2,24,&va,"Kp =");
            }
        if (metodo==3)
            {
             ini_log (0x0F,0xAA,2,7,3,0,(int)(kp*10.0));
             datos_llegados = comunicar_log(0);
             if (datos_llegados.dr_error)
              {
               closegraph();
               enviar_datos.dr_error = 1;
               strcpy (enviar_datos.dr_mensaje,datos_llegados.dr_mensaje);
               return enviar_datos;
              }
             imprimir_const ((int)(kp*10.0)/10.0,1,24,&va,"Kp =");
            }
        clearviewport();
    }

if (key==0x3d00)                  // Tecla F3 cambia Ki
    {
        gotoxy (12,10);
        clearviewport();
        printf ("Ingrese nueva constante Ki...");
        if (metodo==1 | | metodo==2) dato = leer_flot(ki,0.0,327.6,5,47,10);
        if (metodo==3) dato = leer_flot(ki,0.0,25.5,5,47,10);
        ki = dato;
        if (metodo==1 | | metodo==2)
            {
             imprimir_const ((int)(ki*100.0)/100.0,2,34,&va,"Ki =");
             ini_log (0x0F,0xAA,2,7,4,0,(int)(ki*100.0));
             datos_llegados = comunicar_log(0);
```