Plataforma de Control Remoto Aplicada a un Sistema de Levitación Neumática

José Antonio Pérez García de Castro

Tutor: D.Manuel G. Ortega Linares

Departamento de Ingeniería de Sistemas y Automática
Universidad de Sevilla

Septiembre de 2005

Índice

Capítulo 1

Introducción

En los últimos años se ha observado un vertiginoso crecimiento en el campo de las Tecnologías de la Información, y especialmente en el desarrollo de Internet, posibilitando una conexión sencilla entre dos puntos cualesquiera del planeta para intercambio de todo tipo de datos. Fruto de este avance y de las posibilidades que representa, la enseñanza a distancia está evolucionando a grandes pasos.

Un aspecto destacado es la realización de prácticas a distancia, que por lo general se basan en la ejecución de simulaciones. En este caso los resultados se consiguen trabajando con modelos programados, por lo que se conoce como laboratorio virtual.

Últimamente hay una tendencia creciente a poder trabajar sobre equipos reales a través de la red, provocando el desarrollo de los laboratorios remotos o de telepresencia.

El objetivo central del proyecto consiste en desarrollar una plataforma adecuada para el control remoto de equipos, a través de la cual poder realizar ensayos desde un ordenador exterior al laboratorio.

A la hora de definir la estructura para el control remoto, se han tenido en cuenta una serie de características que han marcado su desarrollo:

- Existencia de un control de acceso a los equipos, con gestión de permisos.
- Diseño escalable, permitiendo añadir nuevos equipos de una forma sencilla.
- Posibilidad de utilizar controladores ya diseñados o programados por el propio usuario.
- Almacenamiento de la información de los experimentos para su análisis.
- Acceso remoto basado en interfaz http, accesible desde un navegador Web, evitando instalar una aplicación específica en el equipo cliente.

La plataforma se va a utilizar para hacer accesible vía red un sistema de levitación neumática, equipo que se ha desarrollado en paralelo a la ejecución de este proyecto con motivo del "Premio OMRON de Iniciación a la Investigación y Automáticaçoncedido en las XXVI Jornadas de Automática celebradas en septiembre de 2005.

1.1. Descripción del sistema de levitación neumática

La figura 1.1 muestra una imagen general del sistema de posicionamiento neumático que se ha usado para probar la plataforma para el control remoto desarrollada.

Figura 1.1: Foto general del sistema de levitación neumática

El objetivo es controlar la posición de una bola que es sustentada a cierta altura gracias a una corriente vertical de aire proporcionada por el motor ventilador. El sistema de visión calcula la posición y la envía a través del ordenador al autómata que controla la frecuencia de salida del variador de velocidad que hace girar el motor, cerrándose así el lazo de control

El soplador es un motor centrífugo de corriente alterna que proporciona una corriente de aire en la que queda suspendida la bola. La velocidad del flujo de aire varía con la frecuencia de la entrada del motor, conectada al variador de frecuencia F7 de Omron, que se ha establecido para que su frecuencia de salida, controlada mediante una entrada analógica de 0 a 10 Voltios, varíe linealmente entre 20 y 30 Hz.

El sistema de visión F150 de Omron se ha configurado para que cuando llegue una petición para efectuar una nueva medida, procese la imagen capturada por la cámara calculando la altura del centro de la pelota respecto a la boca del soplador. Desde el PC se envían los comandos adecuados para solicitar el valor de la altura en cada iteración del bucle de control.

En el ordenador reside la aplicación que controla el funcionamiento del equipo, desarrollada en LabVIEW, desde donde se gestionan las comunicaciones entre el sistema de visión y el autómata, que se efectúan a través de los puertos serie.

El autómata proporciona la señal de control que fija la frecuencia de funcionamiento del motor, generada en función del controlador empleado y los datos recibidos del ordenador correspondientes a la referencia marcada y la altura actual.

En [17] se detallan los distintos controladores ensayados, que siguen varias configuraciones:

PID Se han programado dos tipos de controladores, ambos íntegramente contenidos en el autómata. Uno utiliza la instrucción PID del juego de instrucciones de la serie CJ de los autómatas Omron, mientras que el otro se ha programado.

Borroso Se ejecuta en el autómata.

H_∞ Se calcula en Matlab al cargar el controlador, transfiriéndose al PLC, donde se ejecuta en tiempo real.

GPC El control se calcula en cada iteración en Matlab, enviando al autómata la señal de control resultante para que la ejerza.

La aplicación de control permite elegir entre los distintos controladores para actuar en el equipo, enviando al autómata una trama de configuración para que active la sección correspondiente al control seleccionado. También se encarga de la recogida y almacenamiento de datos para su análisis posterior, almacenándolos en un fichero.

El objetivo del proyecto aplicado al sistema de levitación automática se centra en conseguir seleccionar entre los distintos controladores programados desde un equipo remoto y modificar en línea los parámetros del control.

La arquitectura remota está basada en un servidor Web con soporte para generación dinámica de código que interactúa con una base de datos, desde donde se toman los parámetros para el funcionamiento del equipo.

1.1.1. Listado de los elementos principales del sistema

Autómata programable CJ1M

- CPU 160 E/S 5Kpasos 32KW (CJ1MCPU11)
- Fuente de alimentación 100 a 240Vca 5Vcc 2,8A Relé (CJ1WPA202)
- Módulo 2 Salidas analógicas (CJ1WDA021NL)

Sensor de visión F150

- Unidad procesadora de imágenes V3 NPN (F150C10E3)
- Cámara CCD (F150S1A)
- Consola de programación (F150KP)
- Cable conexión cámara (F150VS)
- Cable conexión monitor (F150VM)
- Lente Pentax

Variador de frecuencia

- Trifásico 400V 0.4KW Control vectorial (CIMRF7Z20P41A)

PC Pentium 4 a 3.2Ghz En el PC se ha instalado el siguiente software básico para la aplicación, si bien no es necesario tenerlo instalado en el equipo cliente para ejercer el control remoto.

- CX-Programmer ver. 5.0
- Labview 7.1
- Matlab 6.5
- Microsoft Visual C++ 6.0

1.2. Estructura del documento

La memoria se divide en los siguientes capítulos:

1. **Introducción.**

   El presente capítulo. Describe los objetivos del proyecto y el sistema empleado en su desarrollo.

2. **Descripción de la solución adoptada.**

   Muestra la solución que se ha tomado para resolver el problema del control remoto, detallando la arquitectura de la plataforma. Se indican también otras posibles soluciones.

3. **Comunicación con el autómata.**

   Para poder modificar los parámetros del autómata en línea hay que conseguir establecer comunicación con él, punto que se trata en este capítulo.

4. **Base de datos para el control remoto.**

   Se detalla la programación de la base de datos empleada como elemento de conexión entre el funcionamiento local y el acceso remoto.

5. **Acceso SQL desde LabVIEW 7.1 .**

   Al estar la aplicación local de control desarrollada en LabVIEW, es necesario poder acceder a la base de datos MySQL, para tomar los parámetros de la misma y reflejar los resultados en ella.

6. **Modificaciones en la aplicación local de control.**

   En este capítulo se muestran las adaptaciones que hay que hacer en la aplicación local para que pueda funcionar con la plataforma para el control remoto.

7. **Características para mejorar la monitorización y el control remoto.**

   Se desarrollan los elementos que se han empleado para mejorar la supervisión del equipo y de operación remota, como es la posibilidad de enviar controladores programados por el propio usuario, usar applets para monitorizar el estado del sistema o incluir las imágenes de la cámara Web en la interfaz remota.

8. **Desarrollo de la interfaz para operar remotamente el equipo.**

   Capítulo en el que se muestra la instalación de los servidores necesarios y la programación de la página Web que hace de interfaz remota.

9. **Conclusiones.**

Por último se incluye una sección de anexos con un manual de funcionamiento del autómata CJ1M, los ficheros fuente de las librerías y applets programados y el código de los archivos que componen la página Web para el control remoto.

Capítulo 2

Descripción de la solución adoptada

Al comienzo del capítulo se describe la estructura adoptada para desarrollar el control remoto del sistema de posicionamiento neumático. Esta arquitectura es la que se detalla en los capítulos posteriores.

En la parte final se comentan posibles alternativas a la arquitectura adoptada y las diferencias existentes.

2.1. Arquitectura de la solución

La solución escogida se centra en el uso de una base de datos que sirva para la gestión de los equipos y los usuarios que pueden acceder a ellos y un servidor Web que sirva de punto de entrada de los equipos clientes a la base de datos.

Los equipos clientes acceden al servidor y se comunican con él mediante peticiones http. El servidor actúa de enlace entre la red exterior y los equipos locales, por un lado accede a los datos de funcionamiento del equipo que residen en la base de datos y por otro ofrece una página Web dinámica. El usuario que se conecta trabaja en el equipo a través de una página Web en un navegador.

El servidor es el único punto directamente accesible desde el exterior. Además de poder controlar el equipo, es necesario que se pueda supervisar el funcionamiento del sistema, para comprobar la correcta evolución de los ensayos realizados. Para ello, se utilizan applets[1] de supervisión, que periódicamente monitorizan el estado del equipo. La información del equipo se transmite estructurada en un documento XML, que debe ser interpretado por el applet. Otros elementos como las imágenes de una cámara Web facilitan la supervisión remota del

[1]Programa en Java que se ejecuta en un navegador

equipo, haciendo presente el entorno de trabajo de la planta real a través de la red.

Según esta estructura, los equipos que se vayan a hacer accesibles remotamente deben funcionar tomando los datos de entrada de la base de datos y reflejando los resultados en la misma.

En el esquema de la figura 2.1 se resume la estructura de la plataforma utilizada para el control remoto.

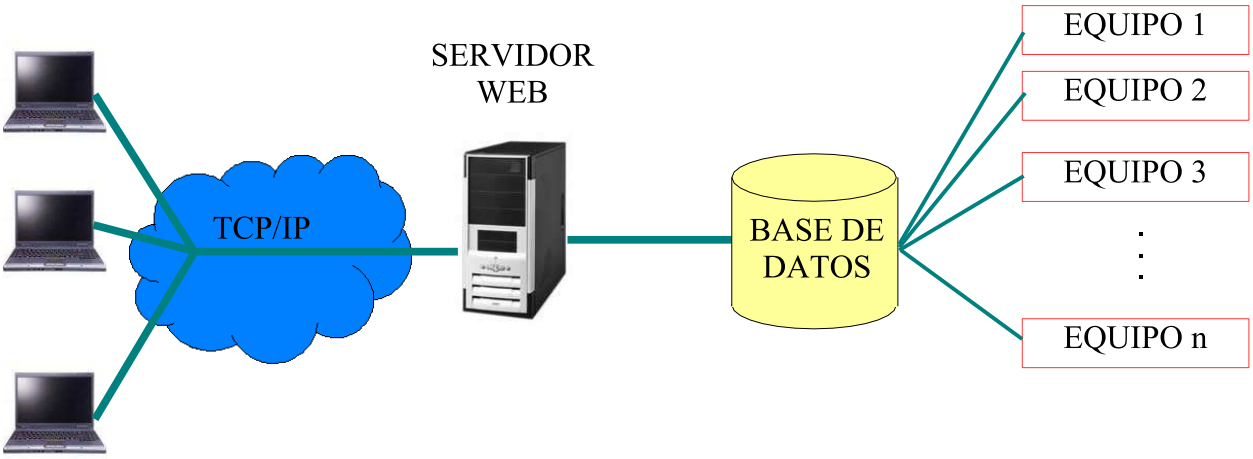


Figura 2.1: Arquitectura de la plataforma de control remoto

2.2. Solución aplicada al sistema de levitación neumática

La solución planteada se ha llevado a cabo a través del sistema de levitación neumática. Aunque según la estructura diseñada los servidores se encuentran en equipos distintos a los de las plantas con las que se trabajan, al ser un único sistema el que se va a controlar remotamente, se han agrupado los servidores Web y de la base de datos en el mismo ordenador que gestiona el funcionamiento del equipo Levineu[2].

En la figura 2.2 puede verse un esquema del funcionamiento del equipo incluyendo la parte correspondiente al control local y la arquitectura para controlarlo remotamente, en la que se muestra las comunicaciones existentes entre cada uno de los elementos.

Básicamente el funcionamiento remoto se centra en tomar los datos de control de la base de datos y reflejar los resultados ella.

En los siguientes capítulos se desarrolla la solución para el sistema de levitación neumática. En primer lugar se solventa el problema de las comunicaciones con el autómata para poder cambiar los parámetros del control durante la ejecución. A continuación se desarrolla la estructura de información que se ha programado en la base de datos. El siguiente paso es poder comunicar la aplicación local de control con la base de datos, para lo que se

[2]Nombre asociado al sistema de levitación neumática

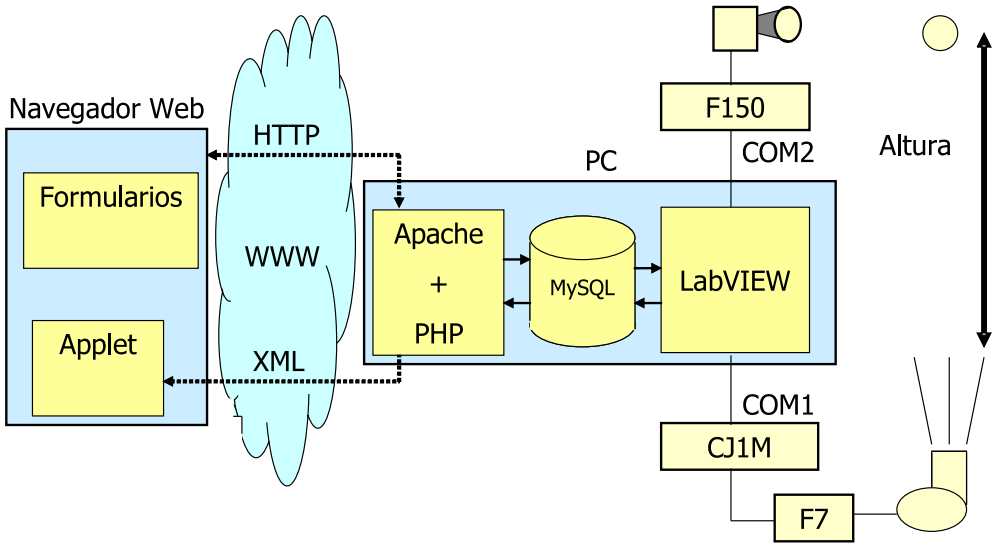


Figura 2.2: Estructura completa del equipo Levineu

ha desarrollado un módulo de acceso SQL en LabVIEW, incluyendo las consultas necesarias para que se lean los parámetros de la base de datos. En los últimos capítulos se desarrolla la interfaz usada para acceder remotamente, basada en un página Web que incluye elementos de monitorización, como las imágenes de las cámaras, y permite actuar sobre el equipo.

2.3. Alternativas

Existen otras posibilidades para llevar a cabo el control remoto que se han probado, pero que finalmente se descartaron en favor de usar una base de datos y un servidor Web para acceder a ella.

2.3.1. Servidor OPC

OPC[3] es un protocolo de intercambio de datos muy usado en el control remoto de procesos industriales.

La estructura sería similar a la planteada en la solución llevada a cabo, sustituyendo la base de datos por un servidor OPC, que realizaría las mismas funciones. En [5] se muestra el desarrollo de una plataforma HTTP-OPC para el control de una planta solar.

Como servidor OPC se ha utilizado el Data Socket Server que se distribuye con LabVIEW. La ventaja que presenta respecto a la base de datos es que la comunicación con el servidor OPC es muy sencilla puesto que LabVIEW incluye módulos para comunicaciones según el estándar OPC. No obstante, una vez desarrollada la librería para acceso SQL, los

[3]OLE for Process Control

pasos a dar serían similares en ambos casos, salvo que las consultas SQL se sustituyen por comunicaciones con el Data Socket Server.

Las figuras 2.3, 2.4 y 2.5 se muestran ejemplos de cómo se introducirían las lecturas y actualizaciones de los objetos OPC en el Data Socket Server desde LabVIEW. Son fragmentos de la aplicación de control original del sistema de levitación neumática, modificada para adaptarla al control y la monitorización remota.

En la figura 2.3 se establece la conexión con el Data Socket Server, indicando el nombre del objeto que se quiere leer o escribir y el modo de acceso, que en este caso es lectura/escritura.

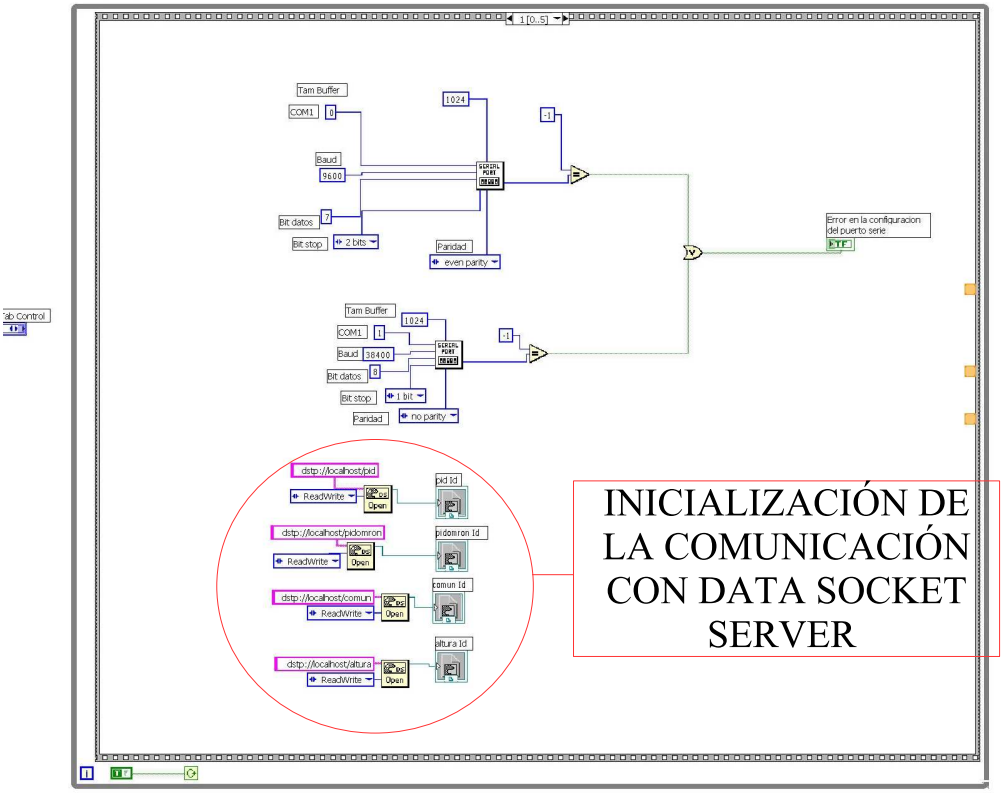


Figura 2.3: Conexión de con el servidor

En la figura 2.4 se copian en el servidor los parámetros de uno de los controladores. Los parámetros se agrupan en un clúster que forman el objeto que reside en el servidor. Se suben los datos del controlador en caso de que el equipo se esté controlando localmente, para poderlo monitorizar desde el exterior mediante una conexión al servidor.

Por último, en la figura 2.5 se lee del servidor el objeto que contiene los parámetros de uno de los controladores y se asignan a las variables de la aplicación. Esta operación se ejecuta cuando el equipo se controla remotamente.

El motivo por el que se descartó esta opción fue que la aplicación no era capaz de realizar todas las conexiones con el servidor, debido a la gran cantidad de parámetros que se desean controlar remotamente y a la necesidad de un refresco relativamente bajo (se trabaja

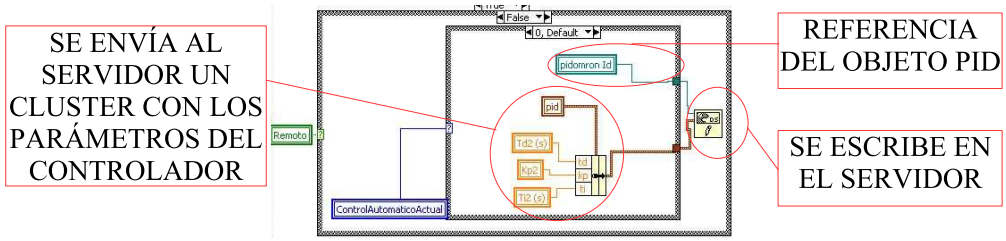


Figura 2.4: Escritura de los parámetros del controlador PID en el servidor

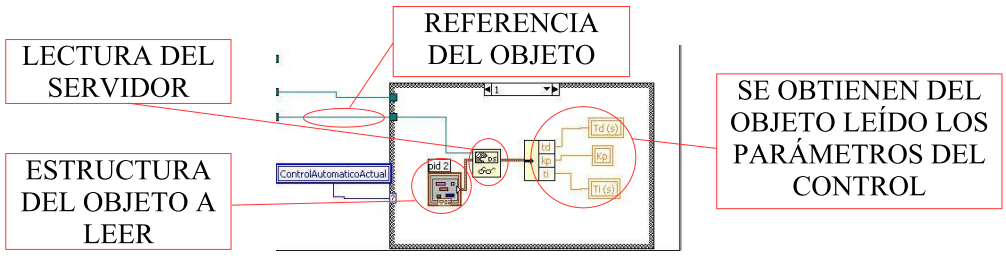


Figura 2.5: Lectura de los parámetros del controlador PID del servidor

con un tiempo de muestreo cercano a 150 ms). OPC está orientado a procesos industriales, que evolucionan de una forma más lenta que el equipo con el que se ha trabajado.

El resultado era que la aplicación daba menos prioridad a las consultas al servidor y los parámetros tardaban a veces cinco o diez segundos en refrescarse desde que se habían modificado.

2.3.2. Remote Panel Control de LabVIEW

La opción más sencilla en cuanto a coste y esfuerzo es utilizar el Remote Control Panel incorporado en la distribución de LabVIEW. Mediante esta herramienta, una vez desarrollada la aplicación local, se puede integrar en un control ActiveX y controlarla vía Web.

LabVIEW incluye un servidor Web que es utilizado para acceder a las aplicaciones exportadas. Para poder desde un equipo cliente acceder al control del *vi* es necesario que tenga instalado el Run Time Enviroment de LabVIEW que National Instruments distribuye gratuitamente.

Para poder acceder al control remoto de un aplicación creada en LabVIEW hay que desarrollar una página HTML que incluya el objeto con la imagen del *Front Panel* y colocarla en el directorio *www* de la instalación de LabVIEW.

En la página, donde se quiera incluir la imagen del *Front Panel*, hay que añadir las siguientes líneas de código, cambiando *controlconcx.vi* por el nombre del *vi* que se desea exportar e indicando en las opciones *WIDTH* y *HEIGHT* el tamaño que se quiere tenga la imagen en la página:

```
<OBJECT ID="LabVIEWControl"
CLASSID="CLSID:A40B0AD4-B50E-4E58-8A1D-8544233807AC" WIDTH=922
HEIGHT=594
CODEBASE="ftp://ftp.ni.com/support/labview/runtime/windows/7.1/LVRunTimeEng.exe">

<PARAM name="LVFPPVINAME" value="controlconcx.vi"> <PARAM
name="REQCTRL" value=false> <EMBED
SRC=".LV_FrontPanelProtocol.rpvi71" LVFPPVINAME="controlconcx.vi"
REQCTRL=false TYPE="application/x-labviewrpvi71" WIDTH=922
HEIGHT=594
PLUGINSPAGE="http://digital.ni.com/express.nsf/express?openagent&code=ex3e33&">
</EMBED>

</OBJECT>
```

LabVIEW también presenta una herramienta que ayuda a la generación de la página Web. Se puede acceder a ella a través del menú *Tools* seleccionando *Web Publishing Tool*.

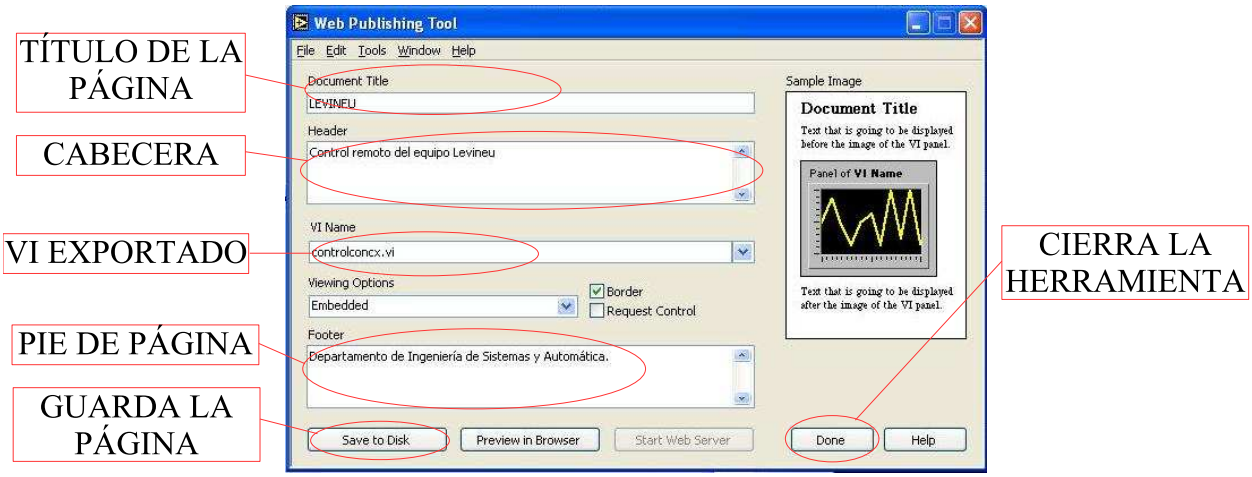


Figura 2.6: Herramienta para la generación de la página Web

Una vez generado el documento HTML, para que se pueda controlar el *vi* desde el exterior tiene que estar abierto y el servidor web de LabVIEW habilitado. Para arrancar el servidor Web se selecciona *Options* del menú *Tools*. Se escoge en el menú desplegable la opción *Web Server:Configuration* y se marca la opción *Enable Web Server* seleccionando el puerto en el que se quiere que funcione el servidor.

En las figuras 2.7 y 2.8 se observa el aspecto de la página generada y cómo se puede controlar el funcionamiento del *vi* a través de un navegador.

LabVIEW también ofrece una herramienta para desde la aplicación local gestionar los clientes que se encuentran conectados. Se puede acceder a través del menú *Tools* en la opción *Remote Panel Connection Manager*. Se muestra en la figura 2.9

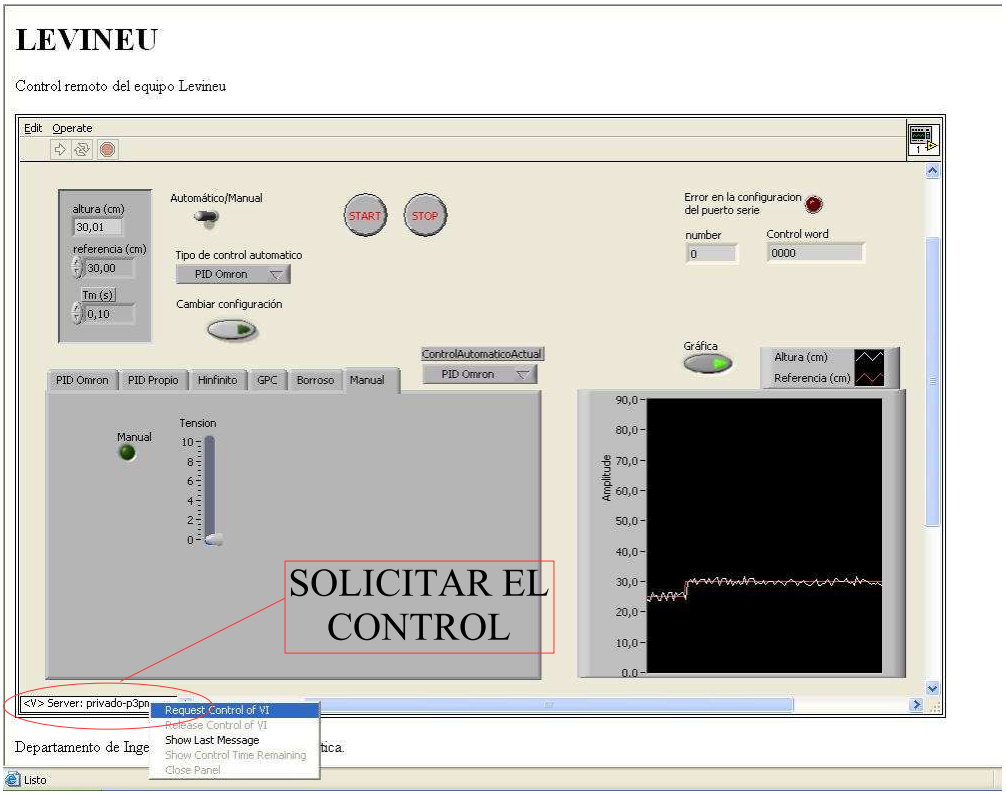


Figura 2.7: Control del equipo a través del Remote Panel Control. Solicita el control del vi

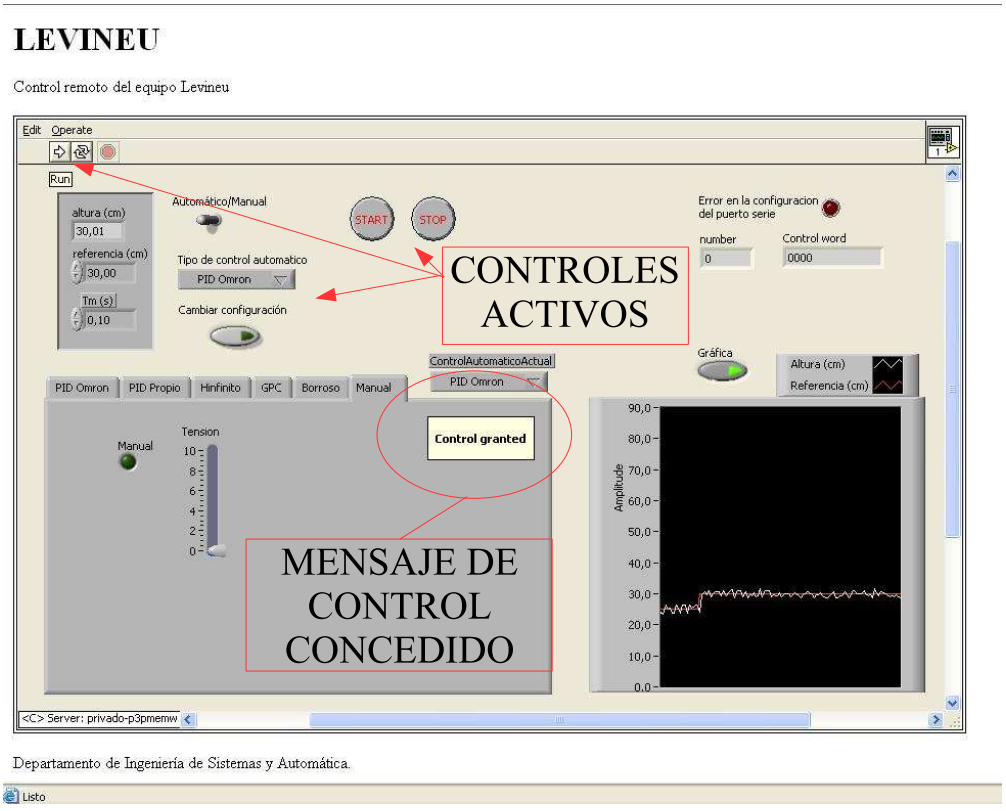


Figura 2.8: Control del equipo a través del Remote Panel Control. Control concedido

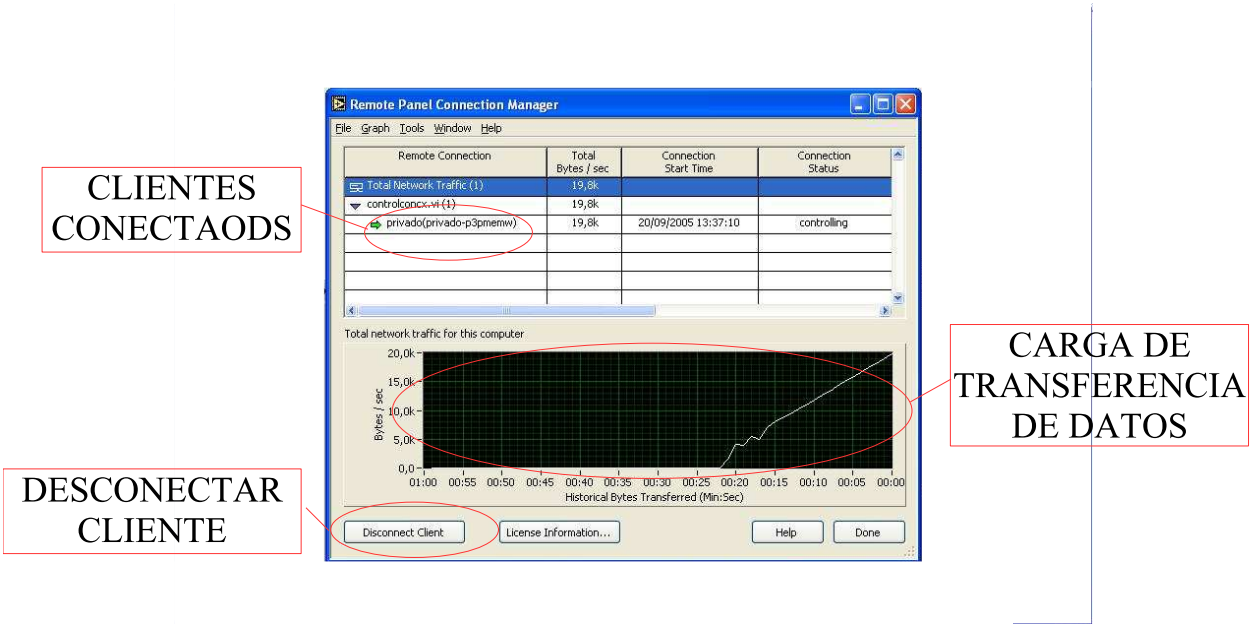


Figura 2.9: Herramienta de gestión de conexiones

A pesar de la funcionalidad y el sencillo uso del *Remote Panel Control*, se ha descartado como solución definitiva por los siguientes motivos:

- Mediante este mecanismo se exporta el *Front Panel* completo, mientras que hay controles que no tienen sentido que se modifiquen remotamente, pues son utilizados para el desarrollo y depuración. Además, el alto número de elementos presentes hace que la interfaz generada sea de gran tamaño, teniendo mucho peso dentro de la página. Una solución a estos problemas sería generar dos interfaces distintas para la aplicación, una para usarla localmente y otra con únicamente los controles accesibles remotamente y el resto ocultos.

- La solución es válida para equipos que localmente se controlen mediante el uso de LabVIEW; Sin embargo, se pretende que la plataforma permita incluir diversos equipos a los que acceder remotamente, que no todos tienen porqué estar controlados con LabVIEW. En cualquier caso, si todos los equipos lo usasen, sería necesario que todos tuviesen una IP pública para que se pudiese acceder al servidor correspondiente desde un equipo fuera de la red del laboratorio. Con la estructura planteada hay un único equipo que debe tener acceso al exterior.

- Para gestionar las conexiones establecidas es necesario estar presente en el equipo. Mediante el uso de la base de datos se pueden desarrollar métodos para la gestión del uso del equipo que se puedan ejercer también remotamente.

Capítulo 3

Comunicación con el autómata

Este capítulo se centra en el desarrollo de los módulos para las comunicaciones entre el PC y el autómata CJ1M de OMRON utilizado en el sistema de levitación neumática, necesarias para poder modificar los parámetros del control ejercido en el PLC en línea. En los anexos se incluye un manual de funcionamiento y programación del autómata, a modo de guía para su uso. Para mayor detalle deben consultarse los manuales.

3.1. Comunicaciones en los autómatas de la serie CJ de Omron

El autómata CJ1M se va a emplear como controlador de un sistema, presentando la posibilidad de modificar los parámetros del control en línea, por lo que es necesario poder establecer comunicaciones con el autómata desde el software que se utilice como interfaz de control. No se dispone de ningún módulo de comunicaciones, por lo que la única posibilidad es utilizar el puerto serie disponible de la CPU, que también se usa para conectarlo al dispositivo de programación.

Para comunicar con dispositivos externos existen dos posibilidades: en modo sin protocolo o usando el protocolo Host Link de Omron.

3.1.1. Comunicación serie sin protocolo

Se basa en el uso de las instrucciones RXD y TXD en los programas de usuario. Simplemente hay que configurar en el autómata (ver manual de uso del autómata de los anexos para hacerlo con CX-Programmer 5.0) el carácter de inicio (opcional) y el código de

fin de trama. En la misma configuración hay que indicar que el puerto va a ser usado sin protocolo y no usando Host Link.

Las instrucciones TXD y RXD tienen tres operandos: la dirección de memoria en la que se guardan los datos en la recepción o de la que se leen para la transmisión, un canal de control y el número de caracteres a enviar o recibir (sin contar los códigos de inicio y fin).

La palabra de control está formada por 4 octetos, con los siguientes valores para el caso de recepción:

- El primer octeto va siempre a cero.
- El segundo indica qué puerto serie es por el que se transmite. Para el puerto de la CPU se utiliza el valor 0. Para los puertos de una unidad de comunicaciones serie se emplearían los valores 1 y 2. En este caso siempre se utilizará el valor 0, puesto que es el único puerto del que se dispone.
- El tercero se utiliza para la monitorización de las señales DR y CS[1]. 0 indica que no se realice ninguna monitorización, 1 monitorización de CS, 2 monitorización de DR y 3 de ambas señales. Cuando se monitoriza alguna de las señales se utiliza el bit 15 de la primera dirección de recepción. Cuando se monitorizan las dos, el 15 es CS y el 14 DR. En cualquier caso, si se está monitorizando no se guardan los datos recibidos.
- El cuarto indica el orden de los bytes a transmitir o recibir. 0 para enviar primero los bytes más significativos y 1 para enviar primero los menos significativos.

Por lo tanto, para una recepción por el puerto serie de la CPU y guardando primero los bytes más significativos, la palabra de control debe tener el valor 0. Para poder recibir algún dato debe estar activa la bandera de receptor listo, que corresponde a la dirección A39206.

La palabra de control para la transmisión, es igual cambiando el significado del tercer octeto, que en este caso sirve para el control de las señales ER y RS[2]. El valor 1 indica que se controla el valor de la señal RS, el 2 para ER y el 3 para ambas. Se utiliza el bit 15 de la primera dirección de trasmisión cuando se controla alguna de las señales, y el bit 15 para RS y el 14 para ER cuando se controlan las 2.

La palabra de control para una transmisión normal debería tener también el valor 0. El bit que indica que el transmisor está listo es el A39205.

3.1.2. Protocolo Host Link

Host Link es un protocolo que implementan los equipos Omron para las comunicaciones, no sólo a través del puerto serie. La ventaja de usar Host Link, es que en el autómata no hay que programar nada, sino simplemente configurar el puerto serie para que trabaje

[1] DR y CS son las entradas del puerto RS-232 Data set Ready y Clear to Send
[2] ER y RS son las salidas del puerto RS-232 Data terminal Ready y Request to Send

en modo Host Link (ver el apartado de Selecciones del manual de uso del autómata). Las tramas Host Link que llegan al PLC las interpreta directamente sin necesidad de indicarlo específicamente en el programa de usuario.

Formato de una trama Host Link:

Figura 3.1: Trama Host Link

@ Código de inicio de una trama Host Link. Un octeto con el carácter @ (40 en hexadecimal).

Unidad Número de unidad a la que va dirigida la trama Host Link en BCD. Ocupa dos octetos, siendo los valores posibles de 00 a 31. El número de unidad es un parámetro de configuración del autómata (para ver cómo configurarlo consultar el manual de los anexos).

HC Código de cabecera, que identifica el tipo de comando que se encapsula en la trama Host Link. 2 octetos.

Datos Parámetros correspondientes al comando que se envía.

FCS Código de comprobación de errores en la trama. Son dos octetos que se corresponden con la división en 2 caracteres de un valor de 8 bit calculado a partir de los caracteres que forman la trama. Este valor se calcula aplicando el OR exclusivo a los caracteres que forman la trama Host Link antes del FCS.

Terminador Formado por el carácter * y CR (retorno de carro).

El tamaño máximo de una trama Host Link es de 131 caracteres, si se supera dicho valor hay que dividir la trama. Para indicar que una trama está dividida, se incluye al final de cada fracción únicamente un * en lugar del terminador completo. En el último elemento de la trama sí se incluye el terminador completo. La cabecera, el número de unidad y el código del comando sólo van incluidos en la primera trama. Las sucesivas fracciones pueden tener como máximo 128 caracteres de longitud. En todos los fragmentos se incluye un campo FCS calculado a partir de los caracteres del propio fragmento.

Hay dos tipos de comandos que pueden enviarse usando el protocolo Host Link:

Comandos en modo C Son comandos de comunicaciones especializadas para Host Link, que se envían desde un ordenador al autómata.

Comandos FINS Son comandos de comunicaciones de un servicio de envío de mensajes, que pueden usarse para comunicaciones en varias redes. Pueden ser enviados por un autómata o por un ordenador. Están orientados a las comunicaciones dentro de redes formadas por autómatas, aunque también se pueden utilizar para comunicar un ordenador y un PLC conectados directamente mediante Host Link.

De los dos tipos, los más sencillo son los comandos en modo C, aunque sólo sirven para una comunicación desde el ordenador al PLC. Como para la aplicación que se va a usar es suficiente con estos comandos, puesto que va a ser siempre el ordenador el que envíe datos o solicite los datos al PLC, se van a explicar con más detalle estos.

Los comandos en modo C forman un método de comando/respuesta para comunicaciones a través del puerto serie entre un ordenador que emite los comandos y un autómata que envía las respuestas. Este sistema permite desarrollar distintas operaciones de control como leer o escribir de zonas de memoria de E/S o cambiar el modo de funcionamiento del autómata.

En el caso concreto para el que se va a usar el autómata, lo más interesante es la posibilidad de escribir directamente en la memoria del PLC o leer de ella.

La lista completa de comandos en modo C y su uso y funcionalidad puede consultarse en el *Manual de Referencia de Comandos de Comunicaciones*,documento W342 de la librería técnica de Omron, centrándose este documento en los comandos de escritura y lectura de memoria.

Los comandos de lectura de memoria tienen el mismo formato, cambiando únicamente el código de cabecera empleado según la zona de memoria de la que se lea:

RR Lectura de zona CIO.

RL Lectura de zona LR.

RH Lectura de zona HR.

RD Lectura de zona DM.

RJ Lectura de zona AR.

RE Lectura de zona EM.

La parte de datos de la trama contiene cuatro octetos para determinar la dirección de comienzo de la lectura en decimal y otros cuatro para indicar el número de palabras que se quieren leer.

La respuesta del autómata consiste en una trama Host Link con el código de cabecera de la petición. En la parte de datos primero se incluyen dos octetos con un código de respuesta y cuatro con el valor de las zonas de memoria solicitadas. Los códigos de respuesta posibles son (en hexadecimal):

00 Ejecución normal.

13 Error de FCS.

14 Longitud del comando incorrecta.

15 El número de palabras excede el área.

18 Error en la longitud de la trama.

21 No ejecutado debido a error de la CPU.

Hay que tener en cuenta que si se solicita la lectura de más de 30 palabras, la respuesta llegará en diversas tramas debido a la fragmentación.

Los comandos de escritura posible son:

WR Escritura en zona CIO.

WL Escritura en zona LR.

WH Escritura en zona HR.

WD Escritura en zona DM.

WJ Escritura en zona AR.

WE Escritura en zona EM.

La parte de datos está formada por cuatro octetos para indicar la dirección de comienzo de escritura en decimal y un número de octetos múltiplo de cuatro con los datos a almacenar en la escritura (cada cuatro octetos se corresponden con una palabra de la zona de memoria). Si se supera el tamaño de 131 caracteres de la trama Host Link hay que fragmentarla, teniendo en cuenta que los datos correspondientes a una misma palabra de memoria deben ir en la misma trama.

En este caso la respuesta del PLC será una trama Host Link en la que en el campo de datos únicamente se incluyen los dos octetos del código de respuesta, que puede tomar los siguientes valores (en hexadecimal):

00 Ejecución normal.

01 No ejecutable en modo RUN.

13 Error de FCS. A3 cuando es trama múltiple.

14 Error en la longitud del comando. A4 cuando es trama múltiple.

15 Zona de memoria fuera de los límites. A5 cuando es trama múltiple.

18 Error de longitud de trama. A8 cuando es trama múltiple.

21 No ejecutado debido a error de la CPU.

3.2. Desarrollo de las comunicaciones desde LabVIEW

La aplicación de control local se ha desarrollado en LabVIEW 7.1. Para poder modificar los parámetros del control o enviar la altura de la bola es necesario poder comunicarse con el autómata desde LabVIEW. De los posibles modos de comunicaciones, se ha decidido aprovechar el protocolo Host Link de Omron, puesto que no se necesita controlar el funcionamiento del puerto serie del autómata simplificando en cierta medida el programa a ejecutarse en él.

Se han programado dos bloques que reproducen el protocolo Host Link en LabVIEW, teniendo en cuenta el formato de trama descrito en el apartado 3.1.2:

CalcFCS.vi Calcula el código de comprobación de trama (Frame Check Secuence), usado para detectar errores en las tramas enviadas. En la figura 3.2 se muestra el diagrama del módulo. La entrada es una cadena de caracteres con la trama completa (salvo el FCS) y la salida es otra cadena consistente en dos caracteres con el código calculado.

Se repite un bucle tantas veces como caracteres existan en la cadena de entrada. Cada carácter se representa mediante dos cifras hexadecimales, usándose cada una de ellas para calcular el XOR consecutivo con los caracteres anteriores, mediante el uso de registros de desplazamiento. Cada cifra se desglosa en los cuatro bits que la codifican en binario, haciéndose la operación XOR con cada uno de ellos. Tras recorrer la cadena entera, con cada grupo de cuatro bits se obtiene la codificación de un número de 0 a 15, que se representa en hexadecimal. El código FCS está formado por los dos caracteres hexadecimales resultados de las sucesivas operaciones XOR.

TramaHL.vi Construye la trama Host Link para ser enviada al autómata. Como entrada tiene la parte de datos de la trama y la unidad correspondiente al autómata que se va a enviar la trama. La salida es una cadena de caracteres con la trama formada y fraccionada en caso necesario. La trama la forma concatenando el carácter de comienzo de trama (@), dos caracteres con el número de unidad correspondiente y la entrada con los datos de la trama. Antes de añadir el FCS y el terminador de trama se calcula si es necesario fraccionar la trama porque exceda el tamaño máximo de 131 caracteres (127 si no se tienen en cuenta el código FCS y el terminador de trama que hay que añadir después). Cuando es necesario fraccionar la trama, hay que tener en cuenta que no se puede dividir por cualquier parte.

Las comunicaciones mediante protocolo Host Link se van a usar para enviar al autómata una trama de configuración, los valores de los parámetros cuando cambien, y el valor de la altura en cada iteración del bucle de control. Esto quiere decir que todas las tramas que se van a enviar al PLC desde LabVIEW son del tipo de escritura en una determinada zona de memoria, a través del uso de comandos en modo C. Los comandos de escritura tienen un formato similar en todos los casos, estando constituida la parte de datos por dos bytes para identificar el comando, cuatro para indicar la dirección de comienzo de escritura y una serie de grupos de cuatro bytes con el valor deseado para cada una de las palabras de la zona de memoria en las que se quiere escribir. A la hora de dividir la trama, hay que hacerlo de tal forma que toda la información de

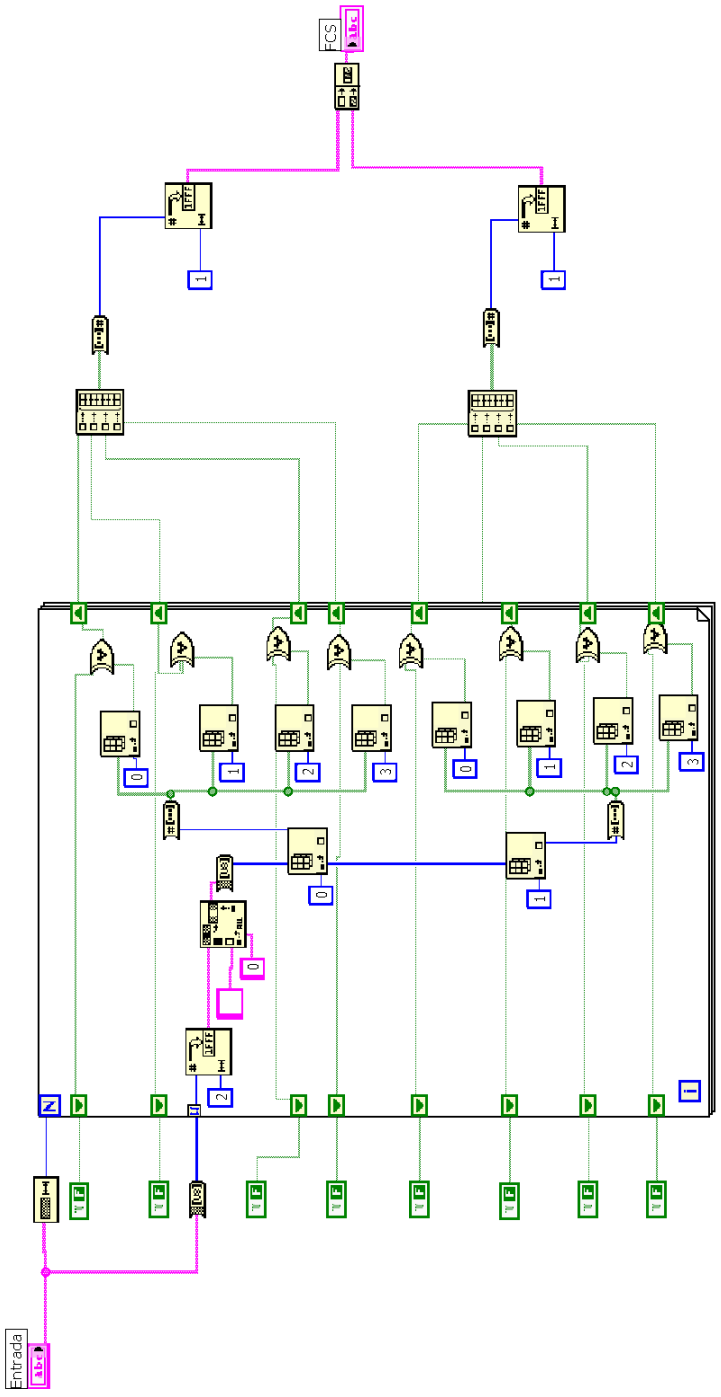


Figura 3.2: calcFCS.vi Block Diagram

una palabra de memoria vaya en la misma trama.

Por lo tanto, si hay que fraccionar la trama, la primera parte se tiene que terminar con los 125 primeros caracteres: 5 con la @, el número de unidad y el identificador del comando, y 120 con unidades de palabras. Si se tomase la siguiente palabra habría que tomar 129 caracteres y excederían el máximo posible cuando se usa tramas divididas (128 caracteres). Los siguientes trozos de la trama contienen únicamente datos a escribir en memoria, por lo que su tamaño debe ser múltiplo de cuatro, en este caso el tamaño mayor posible es 124, puesto que a cada fragmento de la trama hay que añadir tres caracteres con los correspondientes FCS y delimitadores de trama. El último fragmento lleva el terminador de trama.

La estructura del diagrama se muestra en la figura 3.3. Inicialmente se concatena el carácter de inicio de trama, el número de unidad y la parte de datos, para después analizar si se excede el tamaño máximo de trama. Si no se necesita fragmentar la trama se añade el campo FCS, calculado mediante FCS.vi, y el terminador de cadena. En caso contrario (el que se muestra en la figura), se toman los primeros 125 caracteres y se añade el FCS y el delimitador de trama. Con el resto de la cadena, si supera los 124 caracteres máximos que pueden tener las sucesivas partes de datos de las tramas, se entra en el bucle, en el que se extraen los primeros 124 formando los distintos fragmentos, hasta que el resto que quede de la cadena sea menor de 124 caracteres. Al último fragmento se le añade el terminador de trama. La salida se forma concatenando cada uno de los fragmentos de trama obtenidos.

Estos dos bloques forman el soporte básico para la comunicación entre el PC y el autómata. Sin embargo, es necesario que los datos que se envíen tengan un significado y sean correctamente interpretados desde ambos lados. Se ha desarrollado un protocolo de comunicación propio soportado sobre Host Link, adecuado para poder enviar los parámetros al autómata. A la hora de determinar el formato para el envío de los parámetros se ha buscado una estructura adecuada para poder alternar entre los distintos controladores programados fácilmente y adaptarse al número de parámetros de cada uno.

En el autómata se ha reservado una zona para recibir comunicaciones, que se divide en una área de configuración y otra de parámetros. Desde el PC, se escribe directamente sobre esta zona de memoria los datos que se necesitan transmitir. La primera palabra constituye una trama de configuración, en la que se indica el tipo de controlador activo, si ha habido cambio en la configuración de los parámetros del controlador y el estado encendido o apagado del sistema. Tras la trama de configuración, se envían los parámetros, cada uno de ellos formados por una cadena de 10 caracteres. Como parámetros se envían: la altura, el tiempo de muestreo y los propios del controlador, en caso de que haya cambio de configuración del control. El orden de los parámetros se establece inicialmente cuando se programa un nuevo controlador, de forma que el orden sea conocido a la hora de enviarlos en el PC y leerlos desde el autómata.

En el autómata se interpreta la trama de configuración y se adaptan los parámetros al formato adecuado (generalmente coma flotante) para ser usados en el control. Así, por ejemplo, si el controlador activo es el PID, se tomarán del área designada para el envío de parámetros los cinco primeros, correspondientes a la altura, el tiempo de muestreo, la

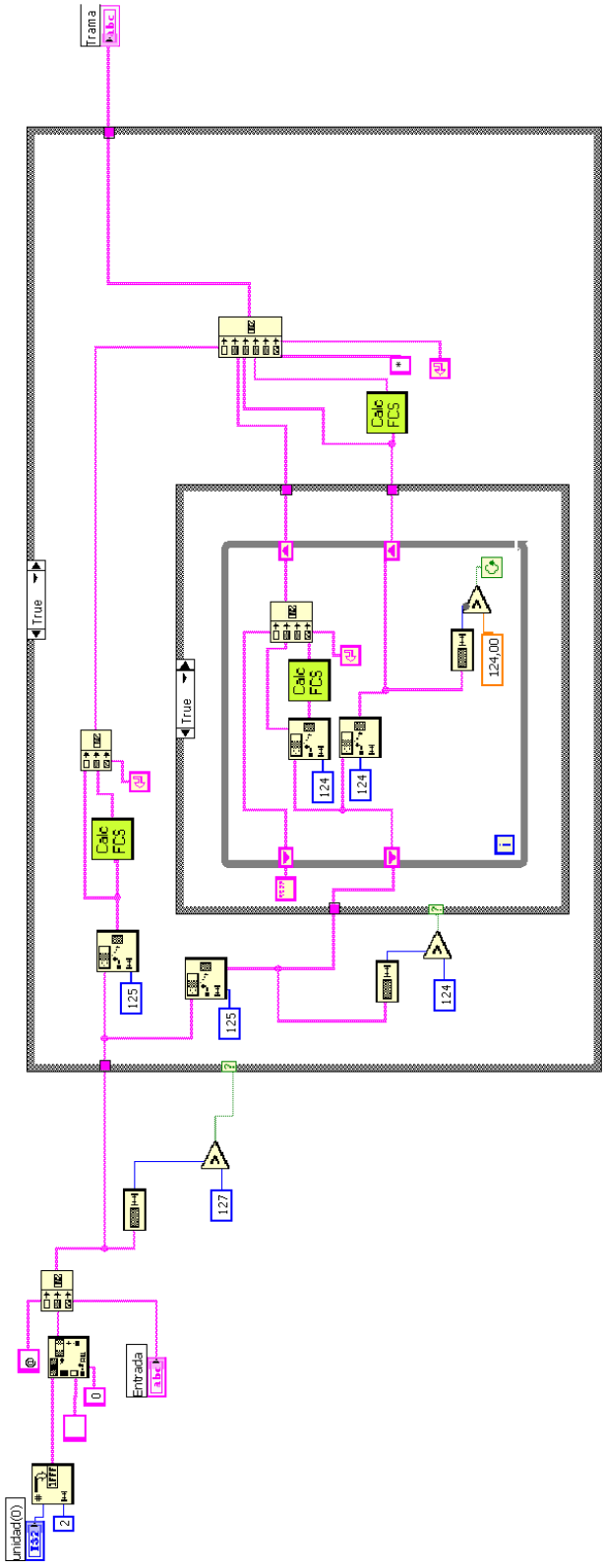


Figura 3.3: TramaHL.vi Block Diagram

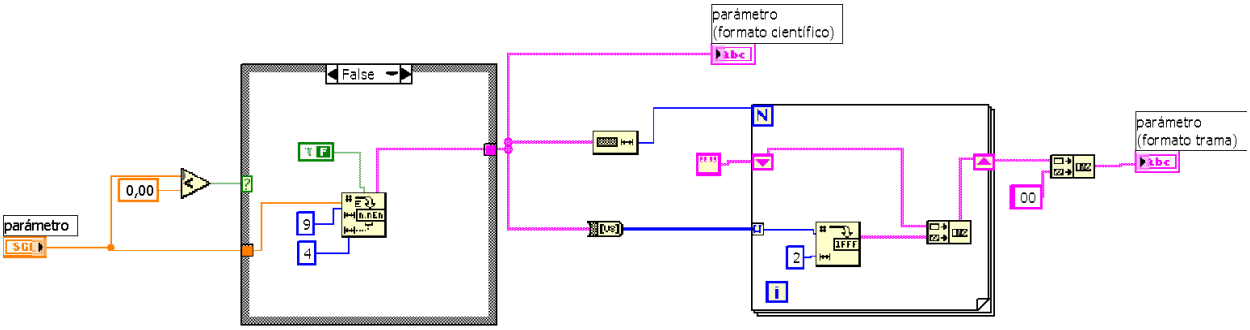


Figura 3.4: Param_hl.vi Block Diagram

ganancia del controlador y los tiempos integral y derivativo. Estos valores leídos se almacenan en las variables correspondientes, en este caso *altura, tm, Kp, Ti, Td.*

Para facilitar el envío de parámetros, se ha programado un nuevo subvi de nombre *Param_hl.vi* que a partir de un valor flotante crea la cadena correspondiente que representa el valor del parámetro para ser incluida en la trama Host Link a enviar. En la figura 3.4 se muestra el esquema básico, en el que puede verse que simplemente se convierte el valor del parámetro a una cadena en formato científico con una longitud de 9 caracteres, formando después la trama con los códigos hexadecimales de cada caracteres, de acuerdo con el formato de los comandos en modo C de escritura. Al final de los 9 caracteres se añade uno más con valor 00 para abarcar una palabra completa, de forma que el décimo byte de cada parámetro en la memoria del autómata queda con valor 0. Esto es usado a la hora de convertir los parámetros a su formato adecuado, ya que marca el final del parámetro.

Capítulo 4

Base de datos para el control remoto

En este capítulo se trata la estructura desarrollada en la base de datos para el control remoto del equipo y su aplicación al sistema de levitación neumática. Previamente se realiza una breve introducción a las bases de datos.

4.1. Introducción a las bases de datos

Una base de datos es un conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su uso posterior.

Los Sistemas Gestores de Bases de Datos (SGBD) permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Las aplicaciones más usuales son para la gestión de empresas e instituciones, aunque también son útiles en entornos científicos para almacenar la información experimental, como es el caso del presente proyecto.

Existen diversos modelos para la administración de los datos de una base de datos. El más utilizado en la actualidad es el modelo relacional. Sus fundamentos se basan en el desarrollo del álgebra y cálculo relacional por Frank Codd. Su idea fundamental es el uso de relaciones. Cada relación consiste en una tabla compuesta por filas o registros , también llamadas tuplas. Las columnas de la tabla se corresponden con los campos de la relación.

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia. Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario no habitual de la base de datos. La información puede ser recuperada o almacenada mediante "consultas"que ofrecen una amplia flexibilidad y poder para administrarla.

El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, Lenguaje Estructurado de Consultas, un estándar implementado por los principales

motores o sistemas de gestión de bases de datos relacionales.

4.2. Programación de la base de datos

Se ha utilizado un sistema gestor de bases de datos basado en lenguaje SQL, concretamente un servidor MySQL, que ha sido elegido debido a que su uso está ampliamente extendido y a sus múltiples ventajas, entre las que destacan:

- Acceso simultáneo por varios usuarios y aplicaciones.
- Seguridad.
- Lenguaje estandarizado.
- Potencia.
- Múltiples API's para distintas plataformas.
- Permite conexiones entre máquinas diferentes.
- Rapidez.

Entre la información que hay que almacenar en la base de datos se pueden distinguir dos aspectos claramente: La información relacionada con la gestión de usuarios y la relativa a los equipos.

4.2.1. Gestión de usuarios

Para controlar quién accede al control de un determinado equipo es necesario almacenar los datos de las personas a las que se le permite. Para validar la identidad del usuario que se conecta, se requiere una pareja de nombre de usuario y clave de acceso.

A cada usuario hay que asociar un conjunto de permisos, que determinen a qué equipos puede acceder para controlarlos o monitorizarlos.

4.2.2. Gestión de parámetros de los equipos

Para controlar un equipo de un laboratorio de forma local, simplemente hay que dar el valor correcto a los parámetros que definen el control que se está ejerciendo. Sin embargo, para un equipo que se opere remotamente hace falta información adicional que permita diferenciar el equipo entre el resto y monitorizar aspectos de su estado.

Para diferenciar cada equipo es necesario un nombre o identificador que debe ser único. Junto a este identificador hay que guardar una serie de datos, necesarios para operar y monitorizar el equipo remotamente, como pueden ser:

- Su estado, para poder ver remotamente si el equipo está libre para acceder a su control, o si está ocupado.
- Un indicador para ver si está activo o detenido.
- Datos de los tiempos en los que se accede al equipo, por si se quiere controlar el tiempo máximo que un usuario puede usar un equipo o por si no se ha abandonado correctamente el control del equipo.
- Identificador del usuario que posee el control del sistema.
- Identificador del usuario que administra el equipo, pudiendo bloquear el acceso al resto de usuarios.
- Parámetros que definen el control que se ejecuta en el equipo. En el caso de que exista más de un control posible, como ocurre en el sistema de levitación neumática del presente trabajo, pueden distinguirse entre parámetros comunes a todos los controles, como puede ser el tiempo de muestreo y las señales de entrada y salida, y parámetros propios del control, como la ganancia de un controlador proporcional.

4.2.3. Modelo relacional

Teniendo en cuenta los apartados 4.2.1 y 4.2.2, el modelo relacional que describe la estructura de la base de datos que se necesita para gestionar remotamente el equipo se corresponde con el esquema de la figura 4.1, en la que, por claridad, no se han incluido los atributos de las distintas entidades.

Puede verse que el modelo está formado por las entidades usuario, equipo, control, parámetro y común. Teniendo en cuenta este modelo, se ha programado una base de datos formada por las siguientes tablas, en las que se indican los atributos que forman cada una y la consulta SQL que la genera:

Usuario Información para controlar el acceso.

- nombre: Nombre del usuario.
- apellido1: Primer apellido.
- apellido2: Segundo apellido.
- id: Identificador único del usuario.
- login: Nombre de usuario para el acceso.
- pwd: Contraseña para validar la validez del usuario. Se almacena la codificación md5 de la contraseña.

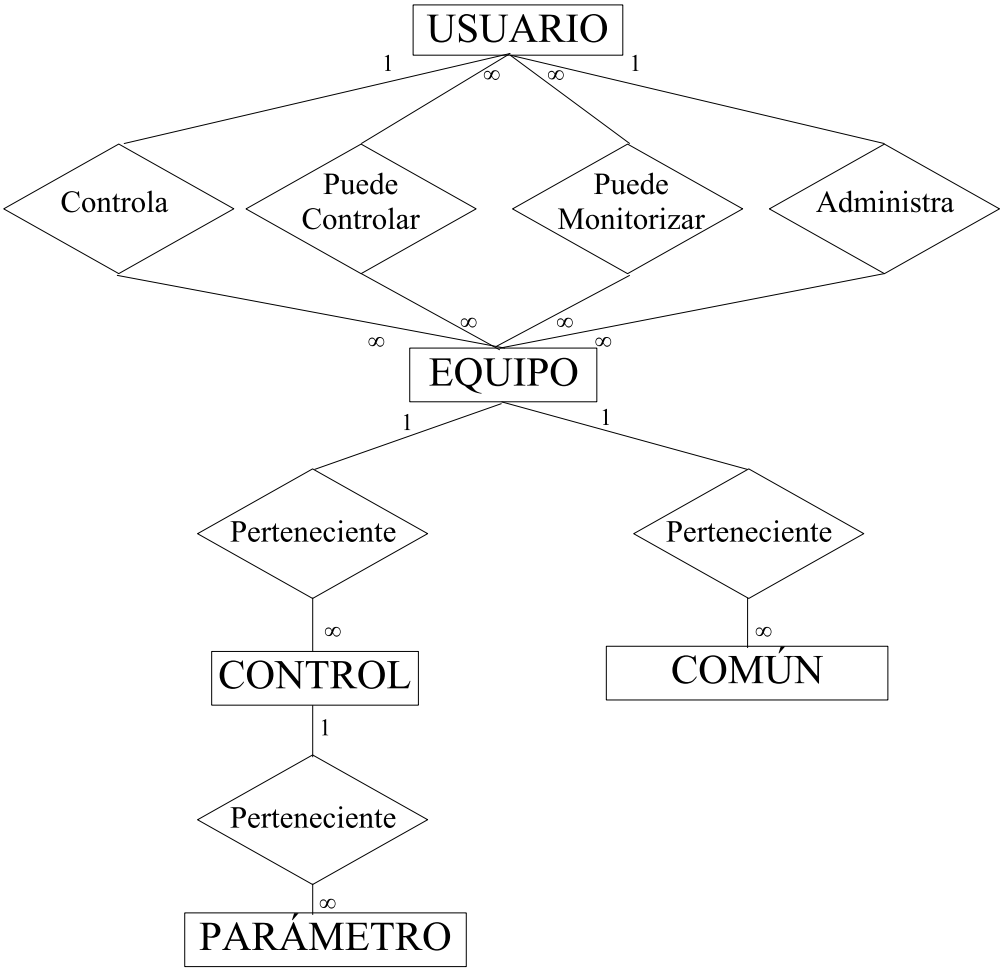


Figura 4.1: Modelo relacional de la base de datos

```
CREATE TABLE ‘usuario‘ (
  ‘nombre‘ varchar(20) NOT NULL default ’’,
  ‘apellido1‘ varchar(20) NOT NULL default ’’,
  ‘apellido2‘ varchar(20) default NULL,
  ‘id‘ int(11) NOT NULL default ’0’,
  ‘login‘ varchar(10) NOT NULL default ’’,
  ‘pwd‘ varchar(70) NOT NULL default ’’,
  PRIMARY KEY  (‘id‘)
);
```

Equipo Contiene los datos principales de cada equipo.

- id: Identificador numérico del equipo.
- nombre: Nombre para reconocerlo.
- descrip: Una breve descripción.
- estado: Indica si está libre para ser controlado, está ocupado por otro usuario o está bloqueado su acceso.
- id_admin: Identificador del usuario que administra el equipo.
- id_control Identificador del usuario que tiene actualmente el control del equipo.
- activo: Indica si está detenido o en ejecución.
- ult_acceso: Tiempo en que se accedió al equipo por última vez para controlarlo, con el fin de manejar las sesiones mal cerradas.

```
CREATE TABLE ‘equipo‘ (
  ‘id‘ smallint(6) NOT NULL auto_increment,
  ‘nombre‘ varchar(20) NOT NULL default ’’,
  ‘descrip‘ tinytext,
  ‘estado‘ smallint(6) NOT NULL default ’0’,
  ‘id_admin‘ int(11) NOT NULL default ’0’,
  ‘id_control‘ int(11) default NULL,
  ‘activo‘ char(2) NOT NULL default ’no’,
  ‘ult_acceso‘ timestamp(14) NOT NULL,
  PRIMARY KEY  (‘id‘)
);
```

Común Parámetros comunes dentro de un mismo sistema.

- nombre: Nombre del parámetro.
- id_eq: Identificador del equipo al que pertenece.
- valor: Valor que posee el parámetro en el instante actual.

```
CREATE TABLE ‘comun‘ (
  ‘nombre‘ varchar(10) NOT NULL default ’’,
```

```
  `valor` float NOT NULL default '0',
  `id_eq` smallint(6) NOT NULL default '0',
  PRIMARY KEY  (`nombre`,`id_eq`)
);
```

Control Controladores existentes en el equipo.

- nombre: Nombre del control.
- id_eq: Identificador del equipo al que pertenece.
- id: Identificador del control.

```
CREATE TABLE `control` (
  `id` smallint(6) NOT NULL default '0',
  `nombre` varchar(10) NOT NULL default '',
  `id_eq` smallint(6) NOT NULL default '0',
  PRIMARY KEY  (`id`)
);
```

Parámetro Parámetros propios de cada controlador.

- nombre: Nombre del parámetro.
- id_cont: Identificador del control al que pertenece.
- valor: Valor que posee el parámetro en el instante actual.

```
CREATE TABLE `parametro` (
  `nombre` varchar(10) NOT NULL default '',
  `valor` float NOT NULL default '0',
  `id_cont` smallint(6) NOT NULL default '0',
  PRIMARY KEY  (`nombre`,`id_cont`)
);
```

Permcont Permisos para controlar un equipo.

- equipo: Identificador del equipo.
- user: Identificador del usuario que tiene acceso al control.

```
CREATE TABLE `permcont` (
  `equipo` smallint(6) NOT NULL default '0',
  `user` int(11) NOT NULL default '0',
  PRIMARY KEY  (`equipo`,`user`)
);
```

Permmon Permisos para monitorizar un equipo.

- equipo: Identificador del equipo.

- user: Identificador del usuario que tiene acceso para monitorizar.

```
CREATE TABLE ‘permmon‘ (
  ‘equipo‘ smallint(6) NOT NULL default ’0’,
  ‘user‘ int(11) NOT NULL default ’0’,
  PRIMARY KEY  (‘equipo‘,‘user‘)
);
```

4.3. Datos de Levineu a almacenar

En esta sección se indican los parámetros presentes en la base de datos para el control remoto correspondiente al sistema de levitación neumática (Levineu). En los anexos se incluye el script SQL que genera la base de datos descrita con los elementos incluidos.

4.3.1. Parámetros comunes

tm Tiempo de muestreo en segundos.

ref Referencia marcada en cm.

altura Altura de la bola respecto a la boca del soplador en cm.

selector Indicador del control seleccionado (0=PID_Instr, 1=PID_Bloque, 2=HInf, 3=GPC, 4=Borroso 5=DLL).

start Inicio de un ensayo.

stop Fin de un ensayo.

detener Detener la aplicación de control local.

4.3.2. Controladores

PID_Instr Controlador PID mediante instrucción.

PID_Bloque Controlador PID mediante bloque de función.

Borroso Controlador borroso.

HInf Controlador H_∞.

GPC Control predictivo generalizado.

DLL Control mediante librería dinámica.

4.3.3. Parámetros del control PID_Bloque

kp Ganancia.

td Tiempo derivativo en segundos.

ti Tiempo integral en segundos.

4.3.4. Parámetros del control PID_Instr

kp Ganancia.

td Tiempo derivativo en segundos.

ti Tiempo integral en segundos.

4.3.5. Parámetros del control Borroso

cLe Centro del conjunto borroso de la izquierda de la variable error.

wLe Pendiente del conjunto borroso de la izquierda de la variable error.

cCe Centro del conjunto borroso central de la variable error.

wCe Pendiente del conjunto borroso central de la variable error.

cRe Centro del conjunto borroso de la derecha de la variable error.

wRe Pendiente del conjunto borroso de la derecha de la variable error.

cLde Centro del conjunto borroso de la izquierda de la variable derivada del error.

wLde Pendiente del conjunto borroso de la izquierda de la variable derivada del error.

cCde Centro del conjunto borroso central de la variable derivada del error.

wCde Pendiente del conjunto borroso central de la variable derivada del error.

cRde Centro del conjunto borroso de la derecha de la variable derivada del error.

wRde Pendiente del conjunto borroso de la derecha de la variable derivada del error.

cNGu Centro del conjunto borroso negativo grande de la salida.

cNPu Centro del conjunto borroso negativo pequeño de la salida.

cCCu Centro del conjunto borroso cero de la salida.

cPPu Centro del conjunto borroso positivo pequeño de la salida.

cPGu Centro del conjunto borroso positivo grande de la salida.

4.3.6. Parámetros del control H_∞ y GPC

Ambos controles se encuentran en fase de desarrollo al finalizar este proyecto, por lo que no se han incluido aún los parámetros para ser controlados remotamente. Se pueden usar, pero no modificar sus parámetros. Cuando se haya culminado su correcta programación, se pueden añadir fácilmente. Simplemente hay que decidir los parámetros que se pueden modificar remotamente y añadirlos a la base de datos. En el capítulo 6 se indican las modificaciones que hay que realizar en la aplicación de control para que funcionen tomando los parámetros de la base de datos.

Capítulo 5

Acceso SQL desde LabVIEW 7.1

La aplicación que gestiona el control del equipo está diseñada en LabVIEW, por lo que para que sea compatible con la estructura desarrollada hay que poder acceder a la base de datos MySQL con LabVIEW. Existe un módulo para comunicarse con bases de datos basadas en SQL, que puede adquirirse pagando la licencia correspondiente. Sin embargo, se ha aprovechado la API de C para programar clientes MySQL. Se ha utilizado una DLL que implementa el cliente y distintos *subvis* que facilitan el uso de la DLL en caso de desconocer el lenguaje SQL, necesario para realizar las consultas.

Hay que tener en cuenta el formato utilizado en LabVIEW para almacenar los distintos tipos de datos en memoria y especialmente de las tablas de cadenas de caracteres, ya que es el tipo de datos correspondiente a los resultados de las consultas SQL que devuelven algún valor, como por ejemplo un SELECT. Si no se realiza bien la conversión de los tipos de datos usados en C a los de LabVIEW, se producirán errores durante la ejecución debido a acceso a zonas de memoria no permitidos.

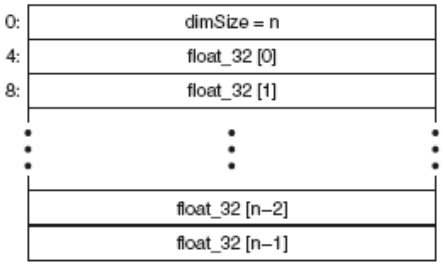
5.1. Manejo de los distintos tipos de datos en LabVIEW

A la hora de programar librerías para llamarla desde LabVIEW hay que conocer cómo se tratan los tipos de datos, para poder trabajar con ellos.

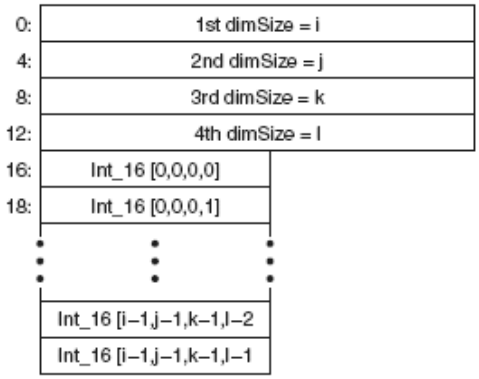
- Los datos booleanos se almacenan en una posición de 8 bits, cuyo valor es 0 si es false y cualquier otro número en caso de ser true.
- Los enteros de tipo Byte, Long y Word tienen un formato de 8, 16 y 32 bits respectivamente.

- Los tipos numéricos de coma flotante siguen el formato IEEE de 32 bits los de precisión simple y de 64 los de doble y 80, en plataformas Windows y Linux, los tipos de precisión extendida.

- LabVIEW trabaja con las tablas mediante el uso de punteros a punteros, a los que llama *handle*. Contienen el tamaño de cada dimensión en un entero de 32 bits, seguido de los datos. El tratamiento de los arrays en una DLL en C es importante, puesto que en las tablas no se incluye la dimensión y hay que adaptar los tipos correctamente. En la figura 5.1 se incluyen dos ejemplos de cómo se almacenan tablas en LabVIEW.



(a) Array de floats de una dimensión



(b) Array de enteros de cuatro dimensiones

Figura 5.1: Ejemplos de arrays en LabVIEW

- Las cadenas de caracteres se almacenan mediante un puntero a una estructura que contiene un valor de cuatro bytes con la longitud de la cadena, seguido de tantos bytes como indique el tamaño, como puede verse en la figura 5.2

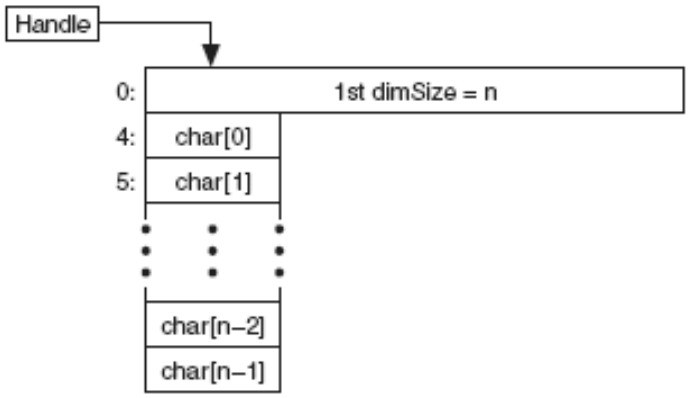


Figura 5.2: Ejemplo de cadena en LabVIEW

- Las estructuras[1], se almacenan atendiendo al orden de sus elementos. El orden se puede modificar haciendo clic con el botón derecho del ratón en el *cluster* y seleccionando *Cluster Order* en el menú contextual. Con el *cluster* se trabaja en LabVIEW mediante

[1]Clusters en LabVIEW

punteros a punteros, que apuntan a la zona de memoria donde se almacena. Dentro de la estructura, los escalares se almacenan directamente y los arrays y cadenas mediante el *handle*.

Las principales diferencias entre el tratamiento de datos que se hace en C y en LabVIEW se encuentran en las tablas y cadenas, por lo que no hay mayores problemas con las librerías sencillas. Sin embargo, para construir librerías complejas, como es el caso de el cliente MySQL objeto de este capítulo, es necesario usar las funciones de gestión de LabVIEW, disponibles en la librería *labview.lib* situada en el directorio *c:/Archios de Programas/National Instruments/LabVIEW 7.1/cintools*. La librería hay que usarla cuando se quiere, entre otras cosas, crear, liberar o cambiar de tamaño tablas o cadenas.

En el archivo *extcode.h* del directorio *cintools* se encuentran las definiciones de tipo adecuadas para trabajar con los de LabVIEW desde un programa en C. En concreto, las que interesan para la librería utilizada:

```
typedef int int32; typedef char* CStr; typedef char uChar;

typedef struct {        /* Cadena de caracteres en LabVIEW */
    long    cnt;        /* numero de bytes que siguen */
    char    str[1];     /* cnt bytes */
    } Lstr, **LStrHandle; /* Cadena y handle de cadena */
```

Además se van a usar arrays de cadenas de una y dos dimensiones:

```
typedef struct {       /* Array 1D de cadenas */
    int32 dimSize;
    LStrHandle String[1];
    } A1D,**A1DHdl;    /* Array  y handle de array */

typedef struct {       /* Array 2D de cadenas  */
    int32 dimSizes[2];
    LStrHandle String[1];
    } A2D, **A2DHdl;
```

5.1.1. Funciones de gestión de memoria de LabVIEW

Son muchas las funciones de gestión de memoria, de forma que aquí solo se van a comentar las usadas en el desarrollo de la librería. Para más información puede consultarse el manual *Using External Code in LabVIEW*.

Mediante las distintas funciones se puede:

- Verificar punteros y handles.
- Asignar y liberar punteros y handles.
- Manipular las propiedades de los handles.
- Operar sobre la memoria (mover bloques, eliminarlos o intercambiarlos)

De todas las funciones, las que interesan para la librería son las que permiten crear un nuevo handle y cambiar su tamaño:

UHandle DSNewHandle (int32 size)

Devuelve un handle nuevo del tamaño indicado en el parámetro *size*. Si ocurre algún error devuelve NULL.

MgErr DSSetHandleSize(UHandle h,int32 size)

Cambia el tamaño del bloque de memoria indicado por *h*. Se pueden producir errores si no hay suficiente memoria o si el handle no está en la zona de memoria indicada.

5.2. API de C para acceso MySQL

El sistema de gestión de base de datos MySQL ofrece múltiples API's para la creación de clientes que accedan a las bases de datos. Se ha usado la API para C para programar la DLL que ofrece las funciones que tratan con el servidor MySQL desde LabVIEW. En el sitio oficial *www.mysql.org* puede encontrarse la documentación que detalla las funciones existentes y su uso. Aquí sólo se van a tratar las usadas para crear los clientes.

El código de la API de C se distribuye con MySQL. Se incluye en la librería *mysqlclient* permitiendo a programas en C acceder a las bases de datos. De hecho, la mayoría de clientes incluidos en la distribución en código fuente de MySQL se han escrito en C, por lo que pueden servir de ayuda o guía a la hora de programar nuevas aplicaciones en C.

5.2.1. Tipos de datos

Los tipos de datos definidos en la API más importantes se enumeran a continuación:

MYSQL Estructura que representa un manejador de una conexión a una base de datos. Se usa en la mayoría de funciones.

MYSQL_RES Estructura que contiene los resultados de peticiones que devuelven filas.

MYSQL_ROW Representación de datos de una columna.

5.2.2. Funciones principales

En primer lugar hay que iniciar una estructura MYSQL para ser usada con el fin de establecer una conexión y acceder a ella. Para ello se utiliza la función

*MYSQL *mysql_init(MYSQL *mysql)*

Si *mysql* es NULL, la función asigna, inicializa y devuelve un nuevo objeto. Si es distinto de NULL, se inicializa el objeto y se devuelve su dirección.

Una vez inicializado un objeto MYSQL, se puede establecer la conexión con una base de datos. Se utiliza

*MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user,*
*const char *passwd, const char *db, unsigned int port,*
*const char *unix_socket, unsigned long client_flag)*

El parámetro MYSQL tiene que haber sido inicializado llamando a *mysql_init*. Los parámetros *host, user, passwd y db* representan las características de la conexión que se quiere establecer, en concreto son cadenas con el nombre del host, que puede ser una dirección IP, en el que se encuentra la base de datos, el usuario que accede y su contraseña y el nombre de la base de datos en la que se establece la conexión. Los parámetros *port, unix_socket y client_flag* se dejan normalmente a los valores 0, NULL y 0, usándose los parámetros por defecto.

Una vez establecida la conexión, se pueden realizar consultas a la base de datos, mediante la función

*int mysql_query(MYSQL *mysql, const char *query)*

Los parámetros de entrada son el identificador de la conexión y una cadena de caracteres con la petición SQL a realizar. Devuelve 0 si no ha habido ningún problema al realizar la petición.

Para acceder a los datos de la última consulta[2] hay que llamar a

*MYSQL_RES *mysql_store_result(MYSQL *mysql)*

El resultado es una estructura MYSQL_RES. A partir de esta estructura puede saberse el número de campos o filas que contiene la respuesta, respectivamente a través de

*unsigned int mysql_num_fields(MYSQL_RES *result)*

[2]No todas las consultas devuelven datos. Por ejemplo, las peticiones de tipo UPDATE modifican la base de datos pero no tienen resultado.

*my_ulonglong mysql_num_rows(MYSQL_RES *result)*

Para acceder a los datos concretos de la respuesta de *mysql_store_resul* se utiliza la función

*MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)*

Que devuelve la siguiente fila del resultado, en forma de tabla de cadenas. Llamando sucesivamente a la función se recorre el resultado entero hasta que se recibe un NULL, indicativo de que no quedan más filas por recorrer.

Con estas funciones básicas se pueden realizar las consultas necesarias en los clientes que accedan a la base de datos para el control remoto.

5.3. Librería para clientes MySQL en LabVIEW

Se ha creado una DLL en Microsoft Visual C++ 6.0, con dos funciones exportadas:

- *sql_select*. Realiza consultas de tipo SELECT. Para leer los datos existentes en la base de datos
- *sql_update*. Realiza consultas de tipo UPDATE. Actualiza los valores de la base de datos.

La DLL de acceso MySQL necesita hacer uso de dos librerías:

- *libmysql.lib* Librería que contiene las funciones de la API de C para MySQL.
- *labview.lib* Para utilizar las funciones de gestión de memoria de LabVIEW.

Para incluir ambas librerías en el proyecto y que pueda ser compilado, se selecciona en $Project->AddtoProject->Files$ y se escogen las librerías indicadas[3]. *libmysql.lib* se encuentra en el directorio *dir_raiz/lib/opt/*, siendo dir_raiz el directorio en el que se encuentran los archivos de MySQL. La librería *labview.lib* se encuentra en el directorio *cintools* de LabVIEW.

Para poder usar la librería generada hay que copiar la librería *labmysql.dll* incluida en la distribución de MySQL en el directorio *c:/Archivos de Programa/National Instruments/LabVIEW 7.1/.*

El código generado se basa en la librería para conectar LabVIEW con bases de datos ODBC de Performance Microwave y que puede descargarse de forma gratuita en la dirección *http://performancemicrowave.com*. Esta librería se distribuye con licencia GPL por lo que

[3]El tipo de archivo debe ser Librería (*.lib)

puede ser modificada libremente, siempre que se distribuya con el programa generado el código modificado. En los anexos se incluye un listado de los ficheros que forman la DLL:

- *stcdll32.cpp*. Fichero con el código fuente de las funciones exportadas y las utilizadas por éstas.
- *stcdll32.h*. Fichero con las definiciones de tipos utilizados y los prototipos de las funciones.
- *stcdll32.def* Indica las funciones que se exportan.

5.3.1. Función para consultas SELECT

*long WINAPI sql_select(DB_LOGIN *db, CStr table, CStr condition, A1DHdl fields, char *distinct, A2DHdl values, CStr debug)*

La función inicialmente intenta establecer la conexión con la base de datos según los parámetros indicados en *db*. Si no hay problemas al conectarse a la base de datos, se genera la petición SQL teniendo en cuenta la tabla y campos seleccionados y la condición indicada. Devuelve los resultados en forma de array de cadenas a través del parámetro de entrada y salida *values*.

Parámetros de entrada:

- *db*. Estructura con las características de la conexión: base de datos, host, usuario y contraseña. En LabVIEW se conecta a un cluster con dichos parámetros.
- *table*. Tabla de la base de datos a la que se quiere hacer la consulta.
- *condition*. Cláusula WHERE de la petición. Condición que deben cumplir los registros de la tabla para formar parte de la salida.
- *fields*. Campos de la tabla seleccionada que se quiere que formen parte de la salida.
- *distinct*. Indica si se quiere que las filas que forman el resultado sean únicas, sin repetición.
- *values*. Tabla multidimensional de cadenas, que se pasa como parámetro para almacenar en ella el resultado de la consulta.
- *debug*. Cadena en la que se incluye información útil para la depuración de los programas. Refleja los errores que se puedan producir en la consulta a la base de datos.

Parámetro de salida:

- Devuelve 0 si no ha habido ningún problema. -1 si no se ha podido establecer la conexión. -2 si hay problemas al realizar la consulta, como error en la sintaxis.

5.3.2. Función para consultas UPDATE

*long WINAPI sql_update(DB_LOGIN *db, char table[], A2DHdl values, A1DHdl fields, A1DHdl where, char debug[], long *num_rows)*

La función inicialmente intenta establecer la conexión con la base de datos según los parámetros indicados en *db*. Si no hay problemas al conectarse a la base de datos, se realizan las actualizaciones indicadas a través de los parámetros de entrada.

Parámetros de entrada:

- *db*. Estructura con las características de la conexión: base de datos, host, usuario y contraseña. En LabVIEW se conecta a un cluster con dichos parámetros.
- *table*. Tabla de la base de datos que se va a actualizar.
- *values*. Tabla de cadenas, que contiene los valores de las actualizaciones que se van a hacer. *fields*. Campos de las tablas seleccionada que se van a actualizar.
- *where*. Cláusula WHERE de la petición. Condición que deben cumplir los registros de la tabla que se actualizan.
- *debug*. Cadena en la que se incluye información útil para la depuración de los programas. Refleja los errores que se puedan producir en la consulta a la base de datos.
- *num_rows* Número de filas que se han visto afectadas por las distintas actualizaciones.

Parámetro de salida:

- Devuelve 0 si no ha habido ningún problema. -1 si no se ha podido establecer la conexión. -2 si hay problemas al realizar la consulta.

5.4. Módulos de LabVIEW para facilitar el uso de la librería de acceso SQL

5.4.1. Select.vi

El *subvi Select.vi* se encarga de facilitar el uso de la función de la librería para realizar lecturas de la base de datos. En la figura 5.3 puede verse el panel frontal y en la 5.4 el diagrama.

El programa evalúa si hay algún error de entrada, en cuyo caso no se hace nada. Si no hay error, llama a la función que realiza la consulta. Posteriormente evalúa el valor devuelto para ver si ha habido algún tipo de error e indicarlo.

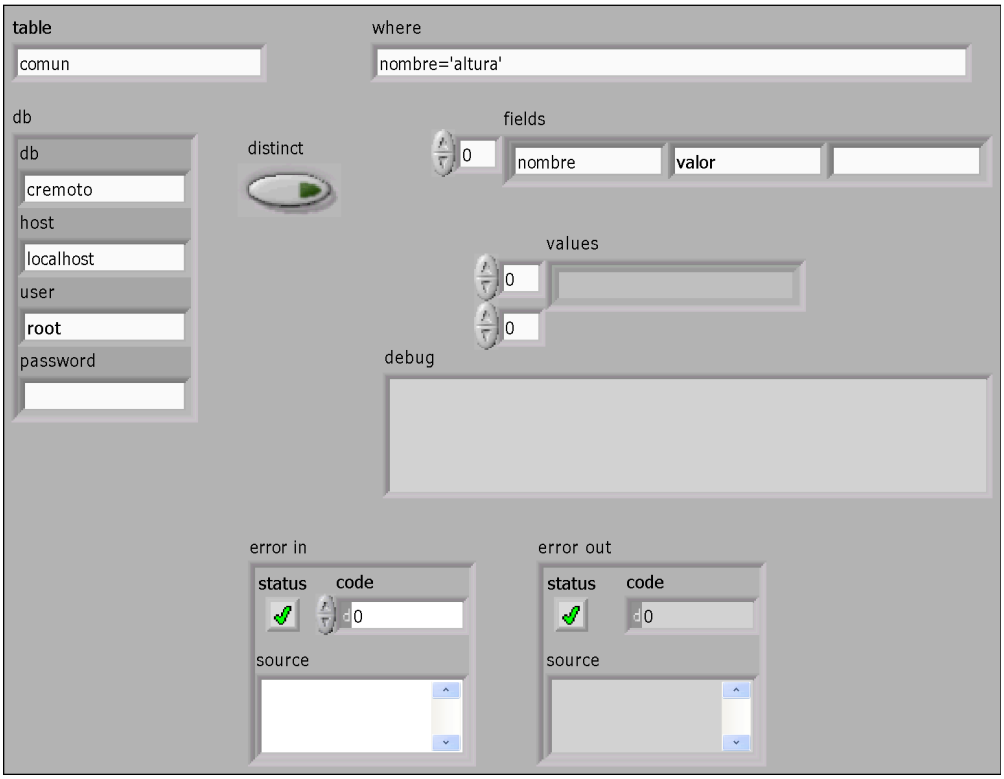


Figura 5.3: Select.vi Front Panel

5.4.2. Update.vi

El *subvi Update.vi* se encarga de facilitar el uso de la función de la librería para realizar actualizaciones en la base de datos. En la figura 5.5 puede verse el panel frontal y en la 5.6 el diagrama.

El programa evalúa si hay algún error de entrada, en cuyo caso no se hace nada. Si no hay error, llama a la función que realiza la consulta. Posteriormente evalúa el valor devuelto para ver si ha habido algún tipo de error e indicarlo.

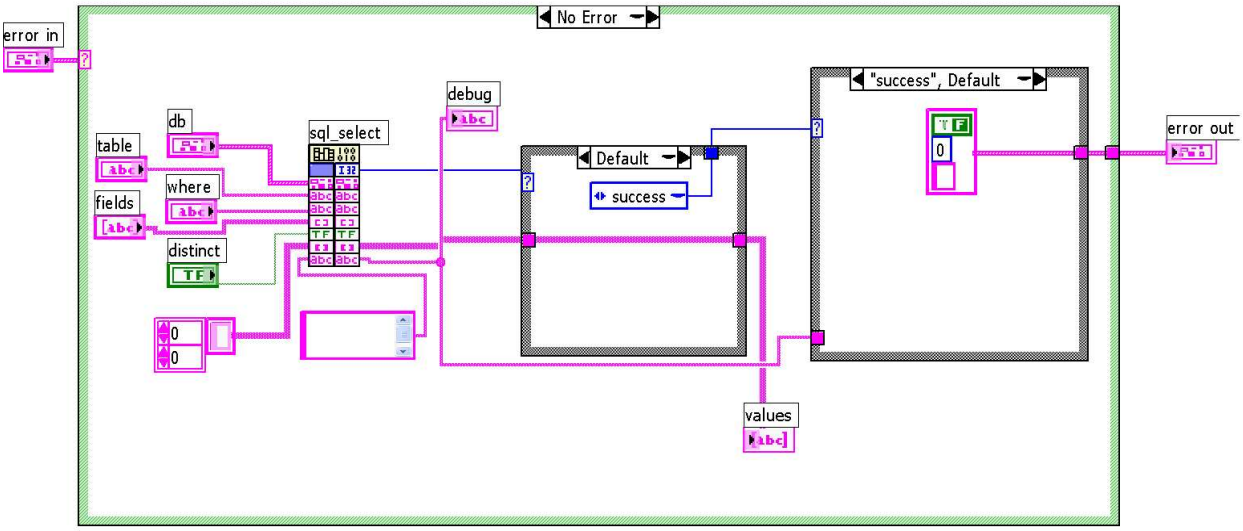


Figura 5.4: Select.vi Block Diagram

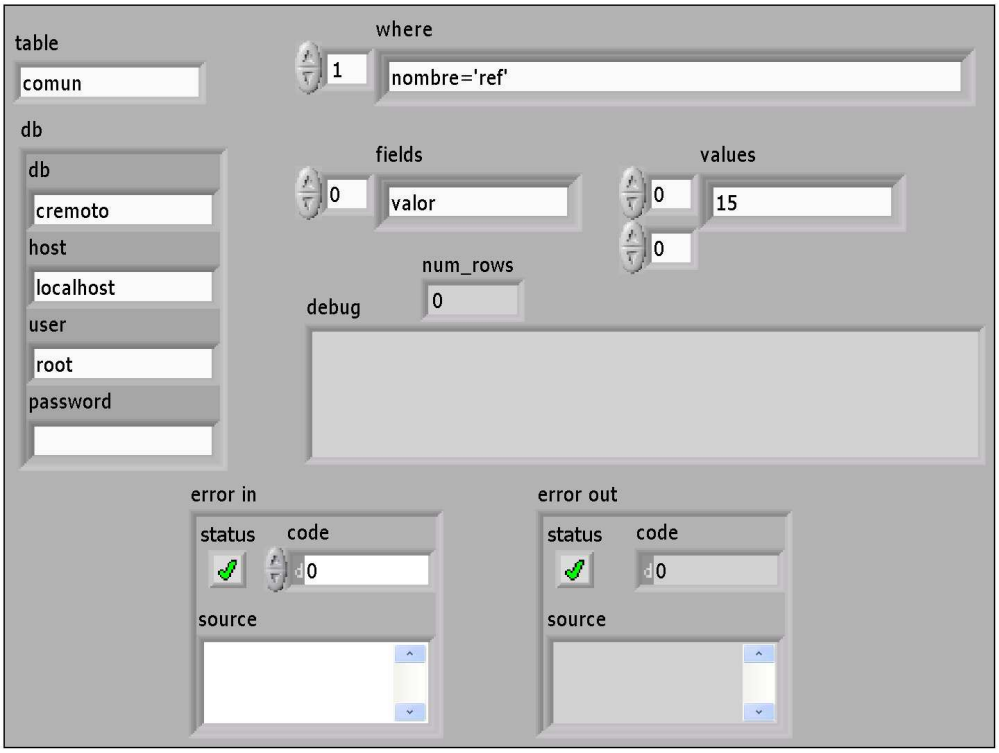


Figura 5.5: Update.vi Front Panel

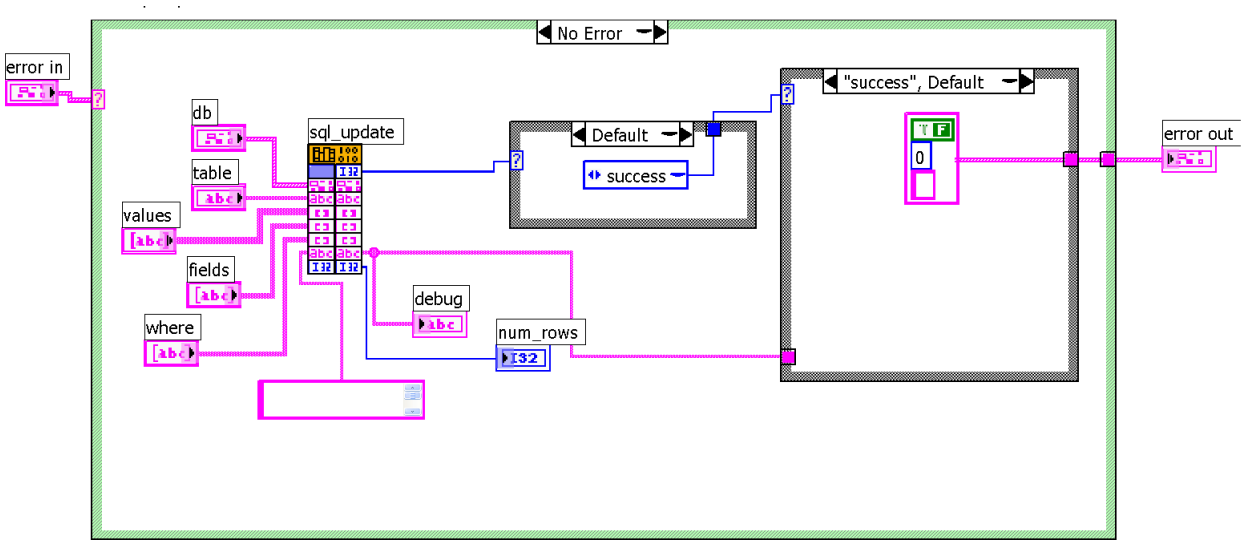


Figura 5.6: Update.vi Block Diagram

Capítulo 6

Modificaciones en la aplicación local de control

Una vez que los parámetros que se van a controlar remotamente se han introducido en la base de datos y que es posible acceder desde LabVIEW a esos datos, hay que realizar modificaciones en la aplicación que controla el funcionamiento del equipo para que interactúe de forma adecuada con la base de datos. Se quiere compaginar el que sea posible controlar el equipo remotamente con que se pueda monitorizar la evolución del sistema desde un equipo exterior cuando se está controlando localmente.

Antes de entrar en las modificaciones se explica brevemente el funcionamiento y la estructura de la aplicación. Posteriormente se introducen los cambios para el control remoto.

6.1. Descripción del funcionamiento de la aplicación de control

La interfaz de control local se ha desarrollado en LabVIEW 7.1 y gestiona las comunicaciones con el sistema de visión y el autómata CJ1M, sirviendo como enlace entre ambos equipos. Ofrece la posibilidad de seleccionar entre los distintos tipos de controladores programados y modificar los parámetros de cada uno de ellos, configurando el autómata de la forma adecuada para que trabaje con el control deseado. Además se encarga de representar la evolución de las señales del sistema, la altura y la referencia, y de almacenar los datos para permitir su posterior análisis.

En las figuras 6.1 a 6.3 se muestran los distintos controles e indicadores que aparecen en el panel frontal de la aplicación y su objetivo.

Figura 6.1: Interfaz de control local (I)

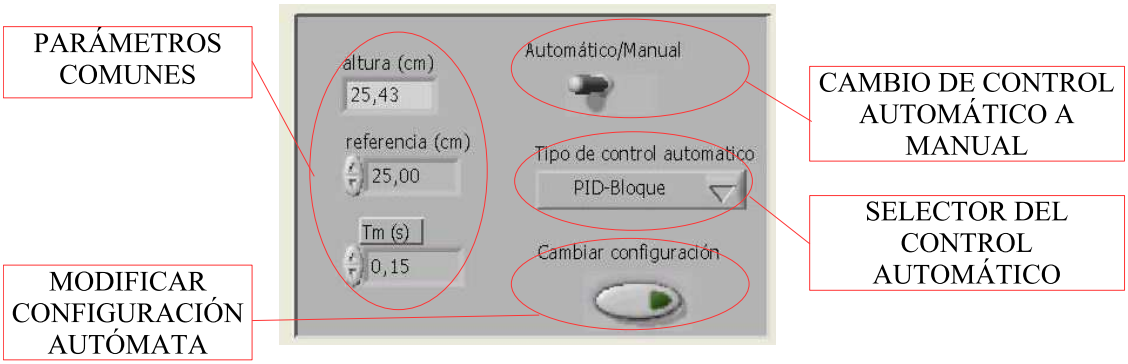


Figura 6.2: Interfaz de control local (II)

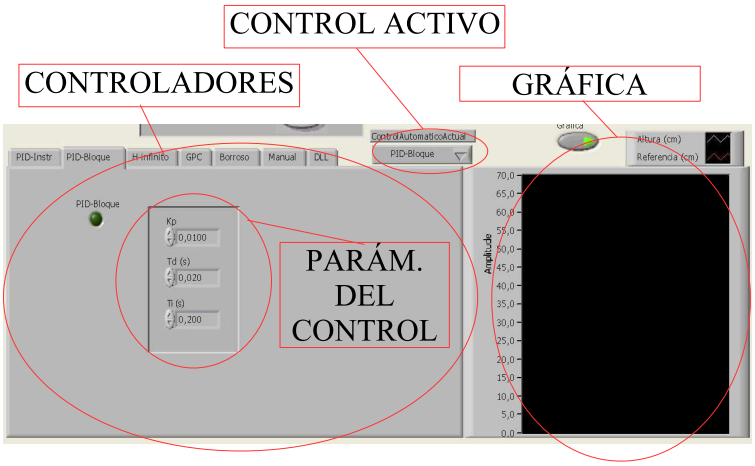


Figura 6.3: Interfaz de control local (III)

El programa consta de un bucle que se repite constantemente mientras que no se pulse el botón para detener la aplicación. El bucle principal consta de un elemento secuencia que consta de siete marcos, cuya estructura se comenta a continuación brevemente:

1. Bucle en el que se espera a que se inicie un experimento o se cierre la aplicación. Es un estado de reposo en el que se permanece mientras el equipo está detenido.
2. Configura los puertos serie para las comunicaciones con el autómata y el sistema de visión.
3. Se ordena comenzar el bucle de programa en el PLC.
4. Bucle de control que se ejecuta cada tiempo de muestreo, en el que se permanece hasta que se detenga el experimento. Se realizan las siguientes acciones y por este orden:
   - Solicitar al controlador del sistema F150 el valor actual de la altura de la bola.
   - Se actualiza el tipo de control automático en caso de cambiarse la configuración.
   - Se prepara la trama de configuración del autómata.
   - Se añaden a la trama los valores de la altura y la referencia.
   - En caso de que se haya cambiado la configuración del control se completa la trama que se va a enviar al PLC con los parámetros nuevos.
   - Se monta la trama Host Link y se envía al PLC.
   - Se almacenan los datos de la altura, la referencia y el tiempo en vectores y se representa en la gráfica.
5. Almacena los datos guardados en un fichero con formato compatible con Matlab.
6. Se ordena parar el bucle de programa en el autómata.
7. Se cierran los puertos de comunicaciones.

6.2. Programación del control remoto en la aplicación

El objetivo de este capítulo es adecuar la aplicación desarrollada para el control del sistema de forma que pueda ser controlada remotamente según la plataforma propuesta como solución. Se ha realizado de forma que se pueda desacoplar el control remoto fácilmente y no afecte a la aplicación original. Además, como se indicó al comienzo del capítulo, debe ser posible monitorizar el funcionamiento del sistema cuando se está controlando localmente.

Por lo tanto, pueden existir las siguientes configuraciones respecto al funcionamiento local o remoto:

- Control local idéntico al que se ejercía antes de realizar las modificaciones. El sistema trabaja totalmente desacoplado de la base de datos, por lo que no se pueden monitorizar

los experimentos que se realizan localmente. Adecuado para cuando se está trabajando con controladores en fase de desarrollo y aún no se han configurado correctamente.

- Control local, pero interactuando con la base de datos para poder monitorizar la evolución del sistema desde un equipo exterior.
- Operación remota. El equipo se controla desde el exterior a través de Internet. La aplicación local debe leer los valores de los parámetros de entrada de la base de datos y actualizar los resultados.

6.2.1. Modificación del panel frontal

En la interfaz se han añadido controles adecuados para la conexión a la base de datos y para alternar entre las configuraciones indicadas en el punto anterior. En la figura 6.4 se muestra la interfaz como ha quedado tras las modificaciones.

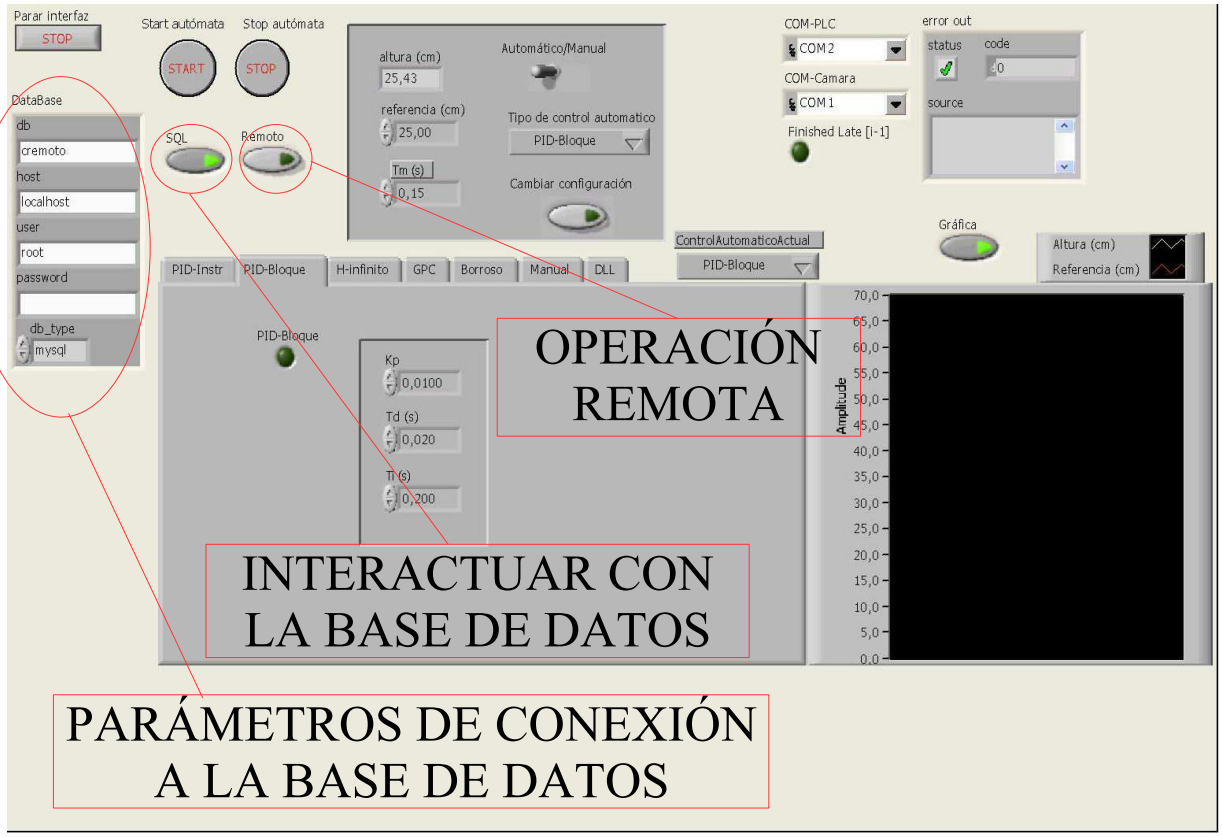


Figura 6.4: Interfaz de control modificada

Si el botón *SQL* está inactivo, la aplicación no realiza consultas a la base de datos y su funcionamiento es el mismo que antes de hacer ninguna modificación. Si está activo, el control local o remoto se diferencia dependiendo del estado del botón de nombre *Remoto*. No tiene sentido activar *Remoto* sin estar *SQL* encendido, pues en ese caso el funcionamiento sería local sin actualizaciones en la base de datos.

6.2.2. Modificación del diagrama

Para realizar las modificaciones hay que tener en cuenta la estructura de la aplicación original (ver la sección 6.1), los posibles modos de funcionamiento deseados y la información de la base de datos que hay que leer o actualizar.

Para considerar las distintas configuraciones, el proceso es el mismo cada vez que hay que añadir una nueva consulta a la base de datos en la aplicación.

Se incluye una estructura condicional controlada por el valor del control *SQL*, de forma que si no está activo no se realiza consulta alguna. Si está pulsado, se distingue si el control es remoto o local, realizándose en cada caso las consultas adecuadas.

Si se programase en C, consistiría en añadir bloques como el siguiente, cuando hubiese que trabajar con la base de datos.

```
if ( SQL == true )
{
    if ( remoto == true )
    {
    /*  Codigo con consultas para operaci{\'o}n remota */
    }
    else
    {
    /*  Codigo con consultas para control local    */
    }
}
else
{
/* No se hace nada, para no afectar al comportamiento original */
}
```

Para saber cuándo hay que incluir las consultas a la base de datos simplemente hay que pensar que el funcionamiento es similar cuando se opera local y remotamente, sólo que con la nueva configuración en lugar de leer los parámetros de lo controles hay que leerlos de la base de datos, y en lugar de mostrar los resultados en los indicadores hay que actualizar la base de datos. Como es interesante que localmente también se puedan ver los cambios que se producen remotamente, realmente lo que se hace es tomar los valores de la base de datos, almacenándolos en los controladores, y efectuar las operaciones reflejándolas en los indicadores, cuyo valor se almacena en la base de datos.

A continuación se indican los cambios que se han introducido en la estructura de la aplicación, siguiendo el orden de ejecución que se ha presentado en la sección 6.1:

1. Bucle en el que se espera a que se inicie un experimento o se cierre la aplicación. Si el equipo está siendo controlado remotamente hay que leer de la base de datos los

parámetros *detener y start* y se almacenan en las variables *Parar interfaz y start*. Sólo se leen estos valores porque son los que pueden hacer que se salga del bucle de reposo. Cuando se sale se desactiva el parámetro *start*, para que la próxima vez que se llegue al bucle haya que volverlo a activar para salir de él. En este caso cuando el control es local no se hace nada, simplemente se espera a que se pulse *start* o *Parar interfaz*. En la figura 6.5 puede verse además la estructura indicada que es común a todas las consultas, de forma que sólo se realiza en caso de estar *SQL* activo.

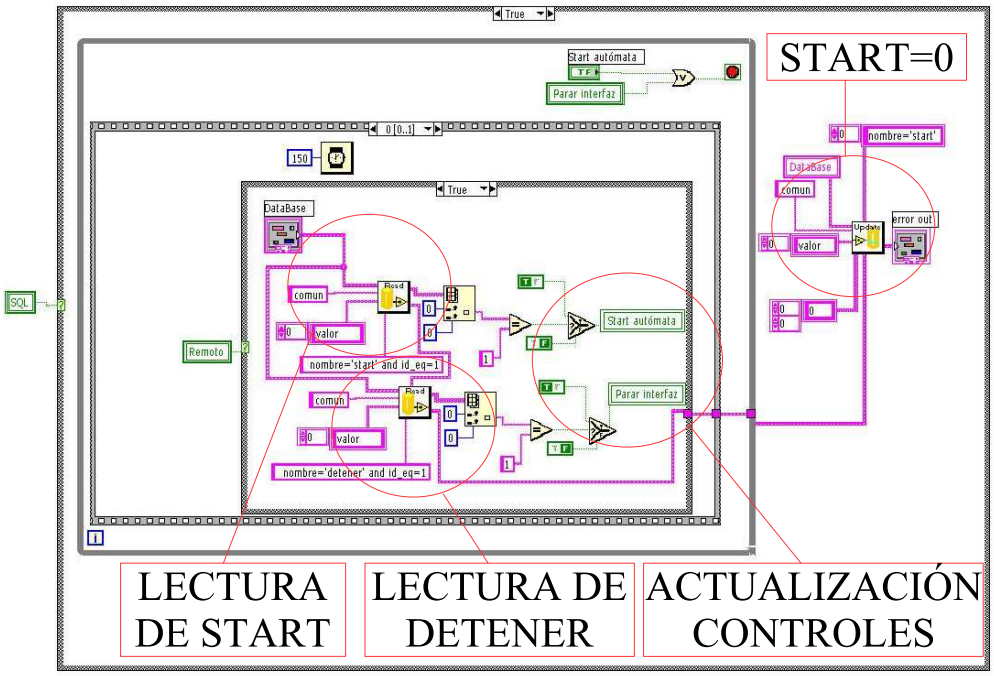


Figura 6.5: Lectura de parámetros remotos para salir del bucle de reposo. Actualización de start al salir

2. Configura los puertos serie para las comunicaciones con el autómata y el sistema de visión. Al configurar los puertos de comunicaciones se aprovecha para poner a uno el parámetro *activo* de la base de datos, con el fin de indicar que en el equipo se está ejecutando un experimento.

3. Se ordena comenzar el bucle de programa en el PLC. Si el equipo se está controlando remotamente, se lee de la base de datos el valor del tiempo de muestreo, ya que en el siguiente marco de la secuencia se comienza el bucle de control que se ejecuta cada vez que pasa el tiempo indicado en la variable *tm*. Si se esperase a leer el tiempo de muestreo con el resto de parámetros, dentro del bucle, no afectaría a la temporización y se ejecutaría el bucle según el último valor existente en *tm*.

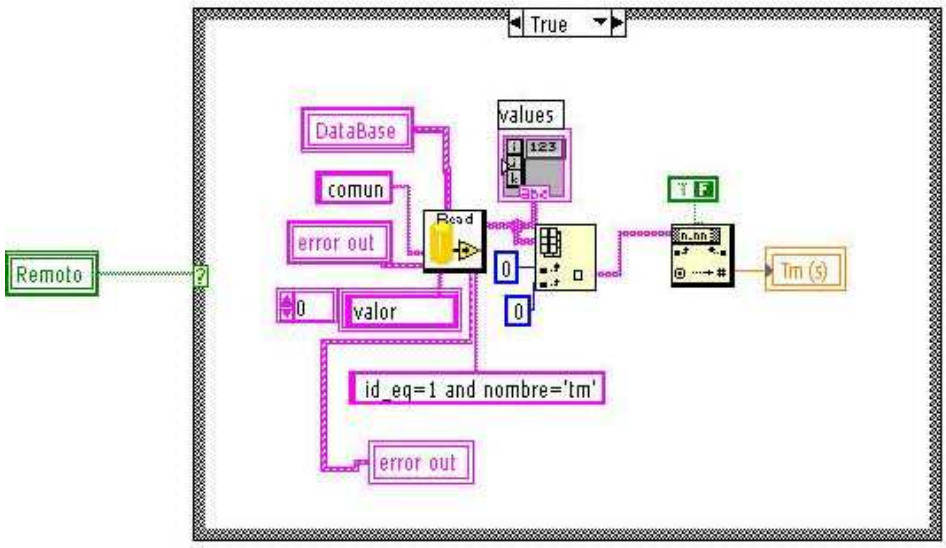


Figura 6.6: Actualización del tiempo de muestreo en caso de control remoto

4. Bucle de control:

   - Se aprovecha el tiempo empleado para la comunicación con el sistema de visión para leer los parámetros del control de la base de datos en caso de control remoto, o actualizarlos en caso de control local.

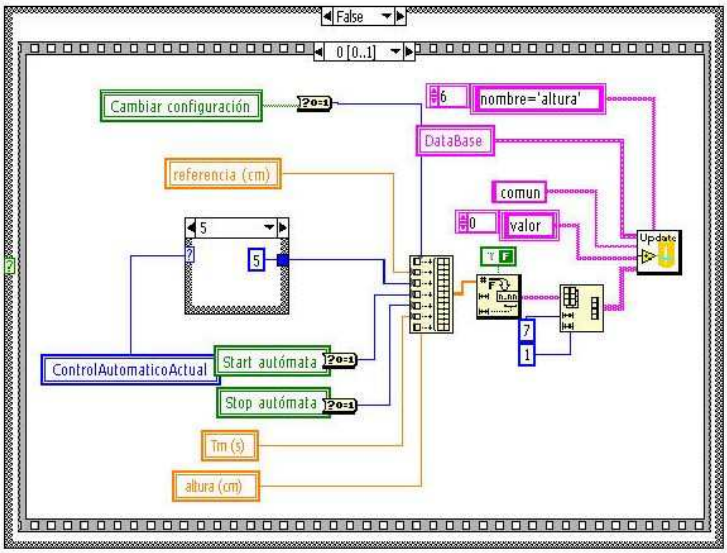


Figura 6.7: Control local. Actualización de los parámetros comunes en la base de datos

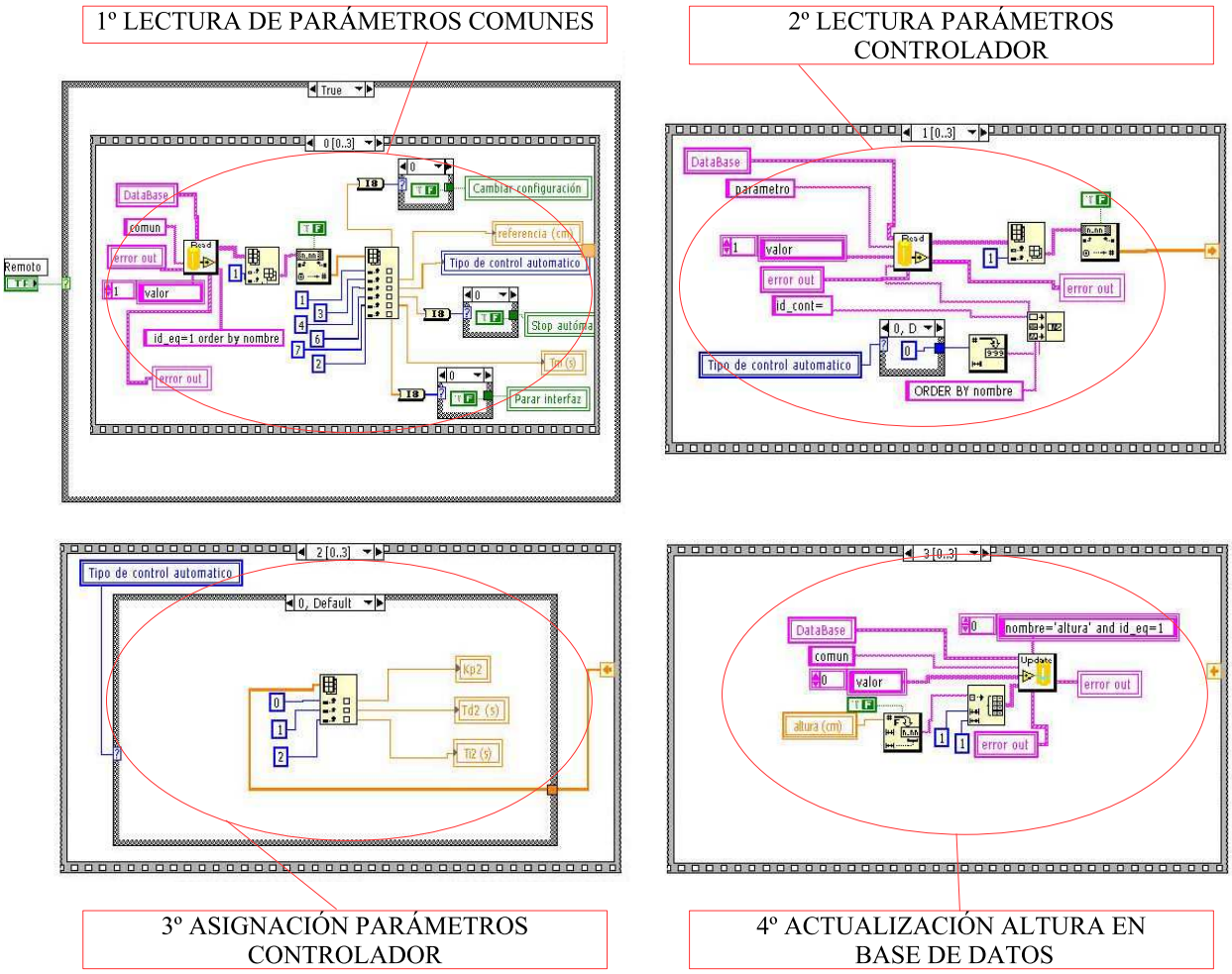
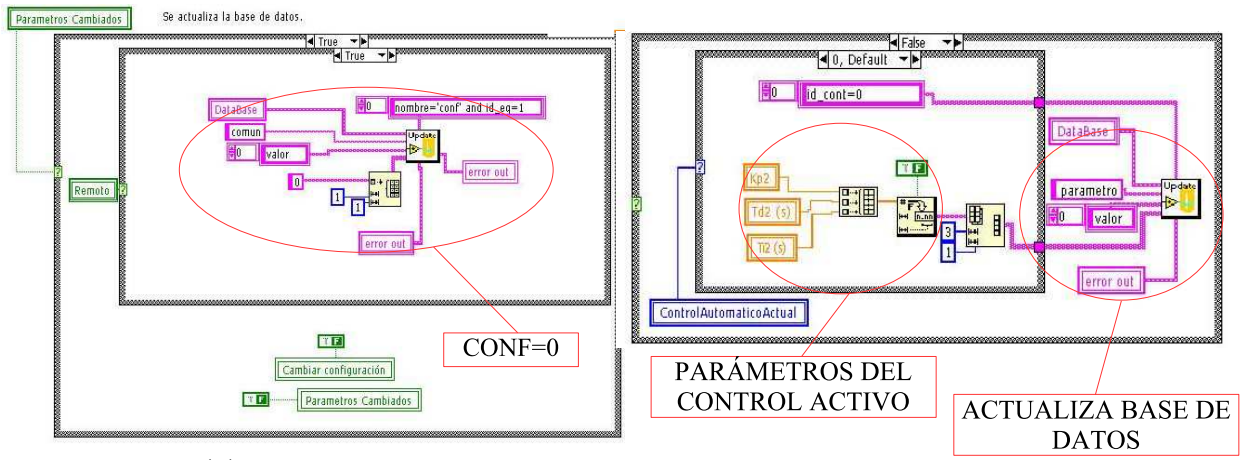


Figura 6.8: Control remoto. Lectura de los parámetros del control y actualización de la altura en la base de datos

- Se actualiza el tipo de control automático en caso de cambiarse la configuración. Sin modificaciones.
- Se prepara la trama de configuración del autómata. Sin modificaciones.
- Se añaden a la trama los valores de la altura y la referencia. Sin modificaciones.
- En caso de que se haya cambiado la configuración del control se completa la trama que se va a enviar al PLC con los parámetros nuevos. Sin modificaciones.
- Se monta la trama Host Link y se envía al PLC. Sin modificaciones.
- Se almacenan los datos de la altura, la referencia y el tiempo en vectores y se representa en la gráfica.

  Al finalizar el bucle, si ha sido una iteración de cambio en la configuración, hay que actualizar los nuevos parámetros en la base de datos si se trata de control local o desactivar el parámetro reservado para realizar un cambio de configuración si el control es remoto. Si no se desactiva al comenzar el bucle nuevamente se volvería a interpretar que es necesario cambiar la configuración sin que hayan cambiado los parámetros.



(a) Control local. (b) Control remoto.

Figura 6.9: Consultas al terminar el bucle de control cuando se ha modificado la configuración del control

5. Almacena los datos guardados en un fichero con formato compatible con Matlab. El módulo *Fichero.vi* que crea el fichero también ha habido que modificarlo. Inicialmente estaba configurado para preguntar a través de un cuadro de diálogo el nombre deseado del fichero en el que guardar los datos. Sin embargo, si el control es remoto no puede existir comunicación con el usuario a través de ventanas emergentes, puesto que aparecen en el ordenador local y no en el que está manejando el usuario. A través de una entrada al *subvi*, se le indica si se está controlando el equipo remotamente, en cuyo caso se guarda el fichero en una ubicación por defecto. Desde la interfaz de control remota habrá que hacer dicho fichero accesible para el usuario que ha realizado el experimento.

6. Se ordena parar el bucle de programa en el autómata. Se actualizan los parámetros *activo* de la base de datos, para indicar que se ha terminado el experimento en el equipo y que se encuentra inactivo. También se desactiva *stop* de manera que cuando se inicie un nuevo experimento no parezca que se ha indicado su fin.

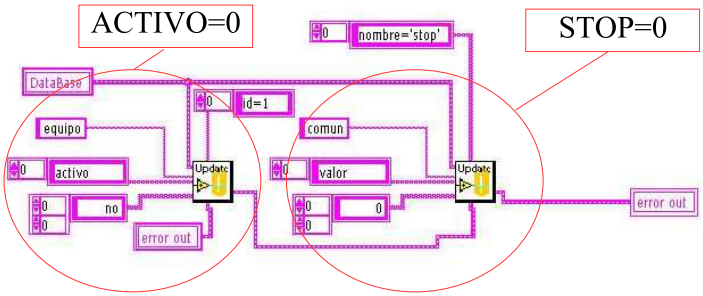


Figura 6.10: Desactivación de los parámetros al acabar un experimento

7. Se cierran los puertos de comunicaciones. Sin modificaciones.

6.3. Integrar un nuevo control

Los pasos a realizar cuando se incluye un nuevo control en la aplicación y se pretende que pueda usarse remotamente son:

- Establecer los parámetros que van a ser accesibles remotamente y crear las correspondientes entradas en la base de datos.
- Al comienzo del bucle de control leer los parámetros, del mismo modo que se hace con el resto de controles. Sería añadir un nuevo caso en el tercer paso de la figura 6.8.
- Al final del bucle de control, incluir el volcado de los parámetros del nuevo controlador en el caso de control local y cambio de configuración. Habría que añadir un nuevo caso en el apartado (b) de la figura 6.9

Capítulo 7

Características para mejorar la monitorización y el control remoto

A continuación se van a describir elementos que se han incluido en la plataforma desarrollada para facilitar la supervisión del equipo y mejorar las características de la operación remota.

En los dos primeros apartados se tratan los aspectos relacionados con el control, que se centran en ofrecer la posibilidad de usar controladores propios del usuario en el equipo y recuperar la información relativa a los experimentos realizados.

En el resto del capítulo se muestran los métodos desarrollados para monitorizar la evolución del sistema desde un equipo externo.

7.1. Uso de controladores programados por el propio usuario

Una de las características más importantes de la interfaz remota de operación es la posibilidad de que el usuario programe un controlador y lo envíe para que se ejecute en el equipo. Esto ofrece versatilidad y cierta libertad a la hora de realizar experimentos, puesto que no hay que limitarse a los ya programados.

Se consigue mediante el uso de una DLL, siendo uno de los controladores automáticos que se pueden seleccionar. Cuando se trabaja con el control basado en una DLL, la señal de control se calcula en el ordenador en cada iteración del bucle de control y se envía al autómata para que se ejerza su valor directamente en la salida analógica que controla la frecuencia de salida del variador.

A pesar de que en el cálculo de la señal de control no interviene el autómata como en la mayoría de controladores programados, es una configuración adecuada por las posibilidades que ofrece, especialmente interesantes para el control remoto.

7.1.1. Librería para programar un controlador

Para la librería que se envía el único requisito que debe cumplir es que contenga una función exportada que cumpla con el prototipo:

double WINAPI Control(double r,double a,double tm)

Donde:

- r es la referencia en el instante actual.
- a es la altura medida de la bola.
- tm es el tiempo de muestreo en segundos.
- Devuelve la señal de control en un rango de 0 a 10V.

Cuando no se ha cargado ninguna librería se utiliza una DLL por defecto que produce una señal de control constante e igual a 0 voltios.

Mediante el uso de variables de tipo *static*, se puede calcular la señal de control teniendo en cuenta los valores de los instantes anteriores de la señal de control y del error.

Desde la interfaz remota se debe gestionar el envío de la librería, almacenándose en una localización determinada para que se pueda cargar de forma correcta.

7.1.2. Carga de la librería en la aplicación local de control

Cuando una aplicación que utiliza una DLL arranca, toma posesión de la librería en modo lectura y hasta que todos los programas que la estén utilizando no se hayan cerrado, no es posible sobreeescribirla.

En nuestro caso, una vez que arranca la aplicación que controla el equipo, se lee la librería que contiene el controlador programado por el usuario. Si durante la ejecución de la aplicación se envía una nueva librería, esta no va a poder sustituir a la anterior hasta que la aplicación se vuelva a cargar. El usuario desde su equipo no puede cerrar y volver a arrancar el programa de control directamente,por lo que es necesario realizarlo de forma automatizada.

Por este motivo se introdujo en la interfaz el botón que detiene la aplicación, de forma que activando dicho control (ya sea local o remotamente) se consigue detener la aplicación.

Sólo falta volverla a arrancar cuando la DLL se haya modificado por la enviada. De esto se encarga el módulo desarrollado *Gestor.vi*.

Gestor.vi

El objetivo de este subvi es permitir la carga de las librerías enviadas por los usuarios.

Cuando se envía una DLL, ésta se guarda en un directorio establecido previamente y con un nombre determinado. Esta librería permanece ahí simplemente a la espera de ser copiada en la destinación correcta para ser usada desde la aplicación. Mientras que permanecen en el fichero temporal no son usadas por ningún programa, por lo que pueden ser sobreescritas sin problema alguno cuando se envía una nueva DLL.

Cuando se envía una nueva DLL por parte del usuario que tiene el control del equipo, se activa en la base de datos el parámetro *detener*, de forma que se para la aplicación. En ese instante, *Gestor.vi* permanece en un bucle de espera, hasta que se desactive el valor *detener*, que ocurre cuando se va a iniciar un nuevo experimento. Una vez que se puede reanudar la aplicación, se copia la librería del directorio temporal en la ubicación indicada para su uso, previa eliminación de la usada la última vez. Por último, mediante el uso de una referencia al vi correspondiente, se vuelve a arrancar la aplicación de control.

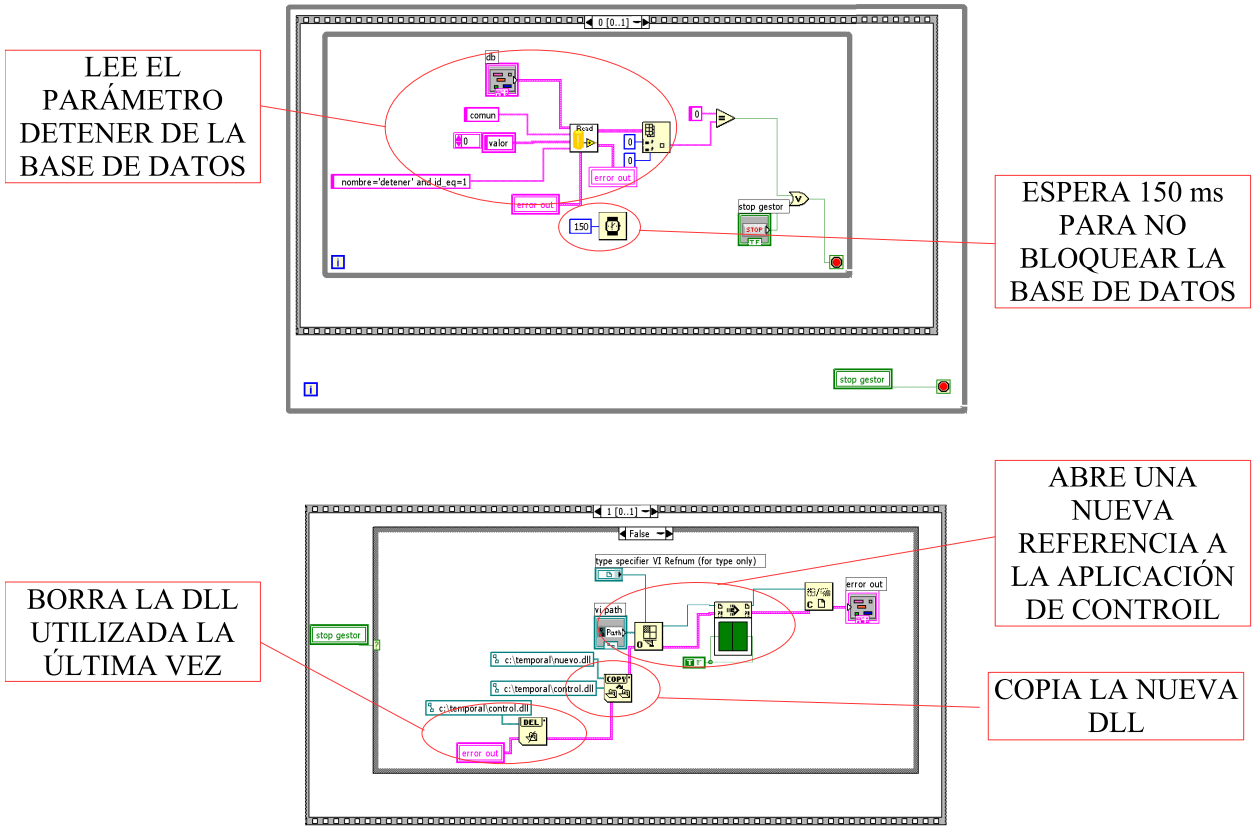


Figura 7.1: Gestor.vi Diagrama

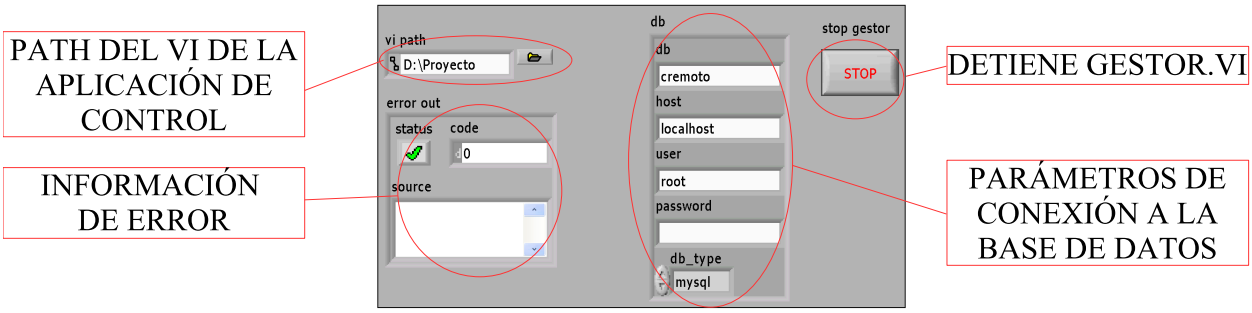


Figura 7.2: Gestor.vi Panel frontal

7.2. Recuperación de la información de los experimentos

Además de poder diseñar nuevos controladores que utilizar en el equipo, es también muy importante poder recuperar la información almacenada de cada experimento para su posterior análisis.

Cuando se está trabajando con control remoto, los resultados se guardan sistemáticamente en un directorio determinado de forma temporal. Desde la interfaz remota se tiene que ofrecer la posibilidad de recuperar el fichero con los datos del último experimento y almacenarlo en la cuenta del usuario.

El usuario debe tener acceso a su cuenta y a los archivos presentes en ella.

7.3. Uso de XML para enviar información para monitorizar el sistema

7.3.1. Breve introducción a XML

El Lenguaje Extensible de Marcas, abreviado XML, describe una clase de objetos de datos llamados documentos XML, que están compuestos por unidades de almacenamiento llamadas entidades. Se utiliza un procesador XML para leer documentos XML y proporcionar acceso a su contenido y estructura. XML ha tenido un gran auge como estándar para el intercambio de datos entre aplicaciones.

Al igual que HTML, se basa en documentos de texto plano en los que se utilizan etiquetas para delimitar los elementos de un documento. Sin embargo, XML define estas etiquetas en función del tipo de datos que está describiendo y no de la apariencia final que tendrán en pantalla, además de permitir definir nuevas etiquetas y ampliar las existentes.

Los documentos XML se dividen en documentos bien formados y válidos.

- Bien formados son aquellos que cumplen las especificaciones del lenguaje XML respecto a las reglas sintácticas. Tienen que cumplir una estructura jerárquica.
- Válidos son los que además de ser bien formados, siguen un estructura y semántica determinada por un DTD o un Schema XML, donde se definen su estructura, elementos y atributos.

Especificaciones básicas que debe cumplir un documento XML bien formado:

- Estructura jerárquica de elementos. Los elementos del documento deben seguir una estructura jerárquica, de forma que una etiqueta se incluya completamente en otra y los elementos se cierren adecuadamente. No se permite, como sí se hace en HTML, abrir una etiqueta dentro de un elemento y cerrarla fuera de ese elemento, la estructura debe ser estrictamente jerárquica.
- Sólo se permite un elemento raíz, que sea el inicio de la jerarquía. El resto de elementos surgirán de este inicial.
- Se permiten etiquetas sin contenido,en cuyo caso se debe cerrar el elemento de la siguiente forma $< ElementoVacio/ >$.
- Los valores de los atributos siempre tienen que estar encerrados por comillas, ya sea simples o dobles.
- Los espacios en blanco, como retornos de carro o tabuladores, se pueden usar para hacer más legible el código, siendo ignorados por los procesadores XML.
- XML se tratan las mayúsculas y minúsculas como caracteres diferentes.

El contenido de un documento XML está formado por entidades:

Elementos. Pueden estar vacíos o tener contenido.

Los elementos con contenido comienzan con una etiqueta $< elemento >$, pudiendo tener atributos, y terminan con $< /elemento >$. El contenido pueden ser caracteres, otros elementos o ambos a la vez.

Los elementos vacíos pueden tener atributos y sólo tienen una etiqueta para delimitar el elemento del formato $< elemento/ >$.

Atributos. Añaden características a los elementos del documento. Los atributos van siempre encerrados entre comillas simples o dobles. Un ejemplo de elemento vacío con atributos:

```
<elemento atributo1="2" atributo2="si"/>
```

Comentarios. Que tienen el formato

```
<!--Comentario en XML -->
```

7.3.2. Estructura del documento XML que representa el estado del sistema

La información del estado del equipo se transmite en forma de documento XML, para que pueda ser analizado por una aplicación cualquiera que se ejecute en el cliente. La estructura del documento generado para describir el estado del sistema de levitación neumática se muestra en la figura 7.3.

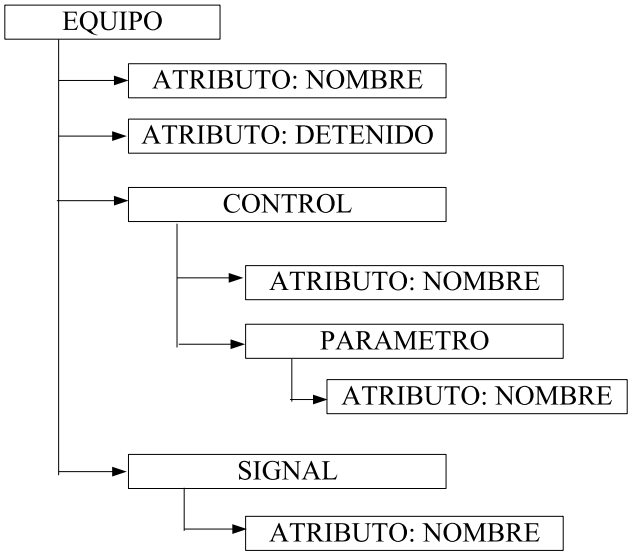


Figura 7.3: Estructura del documento XML

El documento XML debe ser generado dinámicamente en el servidor cuando lleguen las peticiones adecuadas. Aprovechando que el protocolo HTTP permite realizar la transferencia de ficheros en cualquier formato, es posible realizar un programa en el servidor que envíe estructurado en un documento XML los valores de los parámetros que describen el estado del sistema actualizados en tiempo real.

A la hora de construir el documento, existen dos posibilidades:

- Si el equipo está funcionando y se esté realizando un experimento en él, se incluyen en el documento XML los parámetros del control que se usa y los valores de las señales de referencia y salida.

```
<EQUIPO NOMBRE='Levineu' DETENIDO='no'>Sistema de Levitacion
neumatica
    <CONTROL NOMBRE='PID_Bloque'>
        <PARAMETRO NOMBRE='tm'>0.15</PARAMETRO>
        <PARAMETRO NOMBRE='kp'>0.01</PARAMETRO>
        <PARAMETRO NOMBRE='td'>0.02</PARAMETRO>
        <PARAMETRO NOMBRE='ti'>0.2</PARAMETRO>
    </CONTROL>
```

```
    <SIGNAL NOMBRE='ref'>15.0</SIGNAL>
    <SIGNAL NOMBRE='altura'>18.237</SIGNAL>
</EQUIPO>
```

- Si el equipo está en estado de reposo, a la espera de que se inicie un experimento, no se transmiten los valores de las señales de referencia y salida, sólo los parámetros del último controlador empleado.

```
<EQUIPO NOMBRE='Levineu' DETENIDO='si'>Sistema de Levitacion
neumatica
    <CONTROL NOMBRE='PID_Bloque'>
        <PARAMETRO NOMBRE='tm'>0.15</PARAMETRO>
        <PARAMETRO NOMBRE='kp'>0.01</PARAMETRO>
        <PARAMETRO NOMBRE='td'>0.02</PARAMETRO>
        <PARAMETRO NOMBRE='ti'>0.2</PARAMETRO>
    </CONTROL>
</EQUIPO>
```

7.4. Applet para monitorización del sistema

Para reflejar el estado del sistema en cada instante y poder observar su evolución a lo largo de un experimento realizado, se necesita una aplicación en el equipo cliente que periódicamente solicite el documento XML con la información del equipo y la interprete correctamente.

Para evitar realizar instalaciones de software en el ordenador del usuario, teniendo en cuenta que al control remoto se va a acceder a través de un navegador, los más sencillo es utilizar la tecnología Applet de Java para desarrollar la función de supervisión del sistema. En cualquier navegador que tenga habilitado la ejecución de programas Java, podrá ejecutarse el applet. En caso de no tener instalada ninguna máquina virtual de Java, se puede descargar el Java Runtime Enviroment desde el sitio oficial de Sun para Java *http://java.sun.com*.

El objetivo del applet es mostrar la evolución del sistema a partir del documento XML que lo describe, por lo que es necesario poder trabajar con documentos XML desde Java.

7.4.1. Lectura de documentos XML en Java

Para trabajar con datos en XML en Java se utiliza la API JAXP[1], que permite trabajar con los estándares SAX y DOM.

[1] Java API for XML Processing

El estándar DOM[2] genera un árbol jerárquico en memoria del documento o información en XML. Como toda la estructura está representada en memoria, el usuario puede acceder a ella y modificarla.

SAX[3] procesa el documento XML de una manera muy diferente a DOM, ya que lo hace por eventos. SAX procesa la información de un documento XML conforme esta sea presentada, manipulando cada elemento a un determinado tiempo, sin incurrir en uso excesivo de memoria.

Se ha utilizado el API de SAX, ya que no se necesita realizar modificaciones a los documentos XML leídos, sino simplemente interpretarlos.

La API de SAX para Java se define en el paquete *org.xml.sax*.

Se hace uso de la clase *SAXParserFactory* para crear una instancia del analizador del documento utilizado. Para crear un objeto de la clase *SAXParserFactory*, hay que llamar al método estático *newInstance*:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
```

La clase *SAXParser* define varios tipos de métodos para analizar un documento XML. En general se le pasa la fuente de datos XML, que en este caso es la dirección URL en la que encontrar el fichero, y un objeto de la clase *DefaultHandler*.

La clase *DefaultHandler*, implementa la interfaz *ContentHandler*, entre otras, aunque con métodos vacíos. Generalmente se crea una nueva clase que herede de *DefaultHandler* y se sobreescriben los métodos. Estos métodos se llaman cuando ocurren los distintos eventos al procesar el documento XML:

- *startDocument*. Se comienza a analizar el documento XML.
- *endDocument*. Se ha llegado al final del documento.
- *startElement*. Comienzo de un elemento.
- *endElement*. Final de un elemento.
- *characteres*. Se encuentra texto dentro de un elemento XML.

Para crear el objeto *SAXParser* se llama al método *newSAXParser* de la clase *SAXParserFactory*. Una vez creado el objeto, se puede llamar al método *parse* para procesar el documento. En el ejemplo, *dir* es la dirección donde encontrar el documento XML y *this* hace referencia al propio objeto, que hereda de la clase *DefaultHandler*.

```
SAXParser saxParser = factory.newSAXParser();
saxParser.parse(dir,this);
```

[2]Document Object Model
[3]Simple API for XML

7.4.2. Estructura del applet

El applet está formado por dos clases:

LectorXML Clase utilizada para interpretar un documento XML que sigue la estructura de la figura 7.3. Hereda de la clase *DefaultHandler* implementando los métodos descritos en la sección 7.4.1. Ofrece métodos para acceder a los datos del equipo, parámetros del control y valores de las señales.

Monitor Clase que implementa el applet. Periódicamente solicita el documento XML y, mediante el uso de los métodos de *LectorXML*, representa el estado del sistema. Genera una gráfica con la evolución de la altura y la referencia con las muestras tomadas de las distintas solicitudes.

Si se llama al applet con el parámetro $control = si$, permite modificar el valor de la referencia. De esta forma, el mismo applet se utiliza en caso de monitorización o control.

Llamada al applet con control:

```
<applet code="Monitor" width=550 height=440> <param name="control"
value="si"> </applet>
```

Llamada al applet sin control:

```
<applet code="Monitor" width=550 height=440> <param name="control"
value="si"> </applet>
```

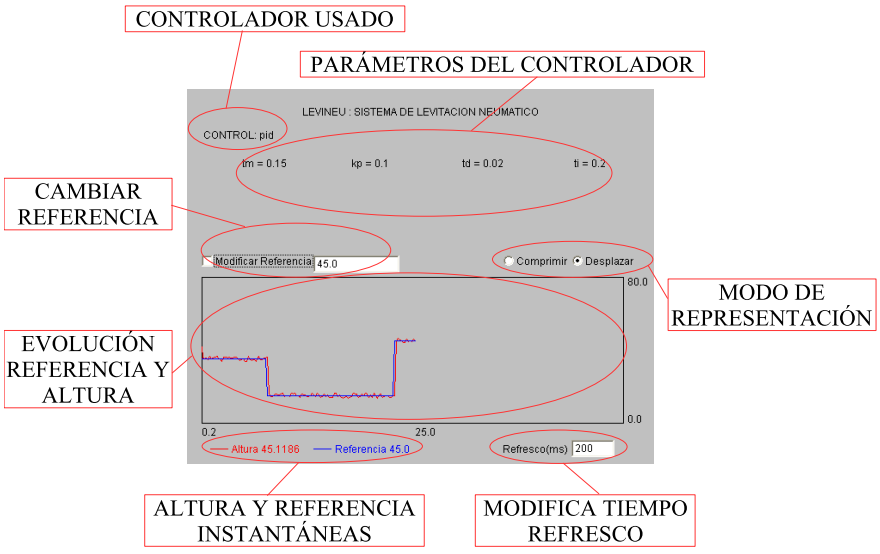


Figura 7.4: Aspecto del applet de monitorización.

7.5. Cámara Web para la supervisión remota

En la actualidad, la existencia de una comunicación visual con el equipo en el que se realizan los experimentos se ha convertido en un aspecto fundamental en un laboratorio remoto. El objetivo de esta sección es permitir integrar en la interfaz remota una cámara Web que ofrezca imagen y sonido del entorno del equipo, para facilitar la supervisión remota del sistema.

Se dispone de un cámara Web de la marca Creative que se muestra en la figura 7.5.

Figura 7.5: Cámara Web

7.5.1. Active WebCam como servidor de las imágenes de la cámara

Se ha escogido el programa Active WebCam de PY Software, cuya versión de prueba puede descargarse de *http://www.pysoft.com/ActiveWebCamMainpage.htm*. La única limitación que presenta es que incluye la leyenda Unregistered en las imágenes exportadas.

Existen diversos procedimientos para observar las imágenes generadas desde la cámara en otro ordenador, entre los que destaca la posibilidad de usar el servidor que incorpora el programa, de manera que los clientes pueden acceder a la cámara simplemente a través de un navegador. Se ha escogido esta opción porque es la más sencilla y transparente al usuario de incorporar las imágenes a la interfaz remota.

Las imágenes se pueden exportar a través de un applet de java o contenidas en un controlador ActiveX. La ventaja de este último es que ofrece una mayor tasa de imágenes por segundo y que permite incorporar audio en la transmisión. Sin embargo, sólo puede visualizarse desde un navegador Internet Explorer.

Se ha configurado el programa para que exporte a través del servidor las imágenes de la cámara Web y las del sistema de visión, con el fin de poder mostrar ambas en la interfaz

remota. Para ello, hay que seguir los siguientes pasos:

- Inicialmente el programa debe buscar las cámaras existentes. Se selecciona la opción *Search Cameras* del menú *File*. La cámara Web debe estar conectada y el sistema de visión encendido y conectado a la entrada de vídeo compuesto de la tarjeta de captura del PC. El programa detectará y mostrará una imagen con las capturas de cada una de las entradas de vídeo. Dado que la tarjeta sintonizadora tiene una entrada para conectar una antena, mostrará tres imágenes. Se selecciona la imagen que sólo muestra ruido y haciendo clic con el botón derecho se selecciona la opción *Show/Hide Current Camera*.

- Se configura el modo de funcionamiento de cada cámara. Haciendo doble clic en una de las imágenes surge un menú de configuración. En la pestaña *device* se puede indicar el uso del micrófono como dispositivo de audio. En la pestaña *Broadcasting* se selecciona la opción de *HTTP Server* y se escoge un puerto libre para ser usado (uno para cada cámara).

- Al realizar cambios y cerrar el menú de configuración, ofrece la posibilidad de actualizar la página web que muestra la imagen de cada cámara. El servidor sólo se va a usar para acceder a las imágenes de las cámaras, incluyendo las sentencias adecuadas en un documento HTML, por lo que no es necesario crear una página Web que resida en el servidor de Active WebCam. Sin embargo, el ayudante para generar la página completa un documento HTML que contiene la llamada adecuada para incluir la cámara con la configuración seleccionada.

- Para que en una página Web que no se encuentra en el servidor del programa aparezcan las imágenes de la cámara, lo más sencillo es copiar del código generado la parte referente a la cámara. En concreto, si se ha escogido la opción del control ActiveX, el código estará formado por un objeto que corresponde a dicho control, cuyos parámetros se pueden cambiar para modificar la apariencia, y un script que inicia el controlador.

```
<OBJECT Name="Pl1" WIDTH="350" HEIGHT="240"
classid="CLSID:66D393D5-4D80-497C-9F4F-F3839E090202"
CODEBASE="http://www.pysoft.com/Downloads/WebCamPlayerOCX.cab"
standby="Loading PY Software player for AWLive files..."
type="application/x-oleobject"> <PARAM NAME="ImagePath"
VALUE="capture2.jpg">
 <PARAM NAME="ShowTopBar" VALUE="0">
 <PARAM NAME="ShowControls" VALUE="0">
 <PARAM NAME="ImagePort" VALUE="8080">
 <PARAM NAME="BackColor" VALUE="#FFFFE3">
</OBJECT>
<script language="JavaScript">
document.Pl1.DocumentURL=document.URL; </script>
```

- Antes de salir, se guardan los datos en una sesión para evitar el realizar de nuevo la configuración completa.

Capítulo 8

Desarrollo de la interfaz para operar remotamente el equipo

El último punto a desarrollar de la solución planteada es permitir que desde el exterior los usuarios tengan acceso al control y monitorización del sistema. El elemento que sirve de intermediario entre la base de datos que rige el comportamiento del equipo y los equipos clientes es un servidor Web, que les presenta la información con el estado del sistema y gestiona el acceso al control del equipo.

El usuario accede a operar en el equipo a través de un navegador, conectándose a la página que se ha programado para actuar de interfaz en el equipo remoto.

8.1. Servidor Web con soporte para generación dinámica de código

El elemento central en la arquitectura montada consiste en un servidor Web, cuyos objetivos fundamentales van a ser dos: hacer accesible desde el exterior una interfaz basada en http para operar en el equipo y controlar el acceso a la base de datos por parte de los usuarios.

Para cumplir los objetivos, el servidor necesita soportar algún lenguaje de programación que se ejecute en el lado del servidor para generar las páginas de la interfaz según el estado del sistema. Además debe estar capacitado para acceder a la base de datos. El servidor Web elegido ha sido Apache, debido a su extenso uso y la posibilidad de configurarlo para procesar scripts basados en PHP.

En este caso, los servidores Web y MySQL residen en el mismo equipo que las apli-

caciones que lo controlan, que en este caso requerían de un sistema operativo Windows para el uso de LabVIEW. Esto impedía plantear el usar equipos Linux para la instalación de los servidores, que suele ser la opción más extendida, especialmente en sistemas basados en Apache+MySQL+PHP. No obstante, existen aplicaciones adecuadas para instalación de cada uno de ellos en Windows, que se pueden encontrar fácilmente en los sitios oficiales. También pueden encontrarse aplicaciones que instalan y configuran el sistema completo de una manera sencilla, como es el caso de AppServ.

8.1.1. Instalación de AppServ

AppServ es un paquete de software que permite instalar en un ordenador, bajo el sistema operativo Windows, de forma sencilla el servidor Web Apache con PHP integrado y un servidor MySQL. Además instala también la herramienta gráfica de gestión de bases de datos phpMyAdmin.

Una vez descargado el archivo ejecutable del AppServ, se pulsa sobre el icono y se inicia la instalación.

En primer lugar se empiezan a preparar los archivos y comienza con la correspondiente pantalla de bienvenida.

Una vez pulsado Next el programa preguntará dónde instalar los archivos correspondientes, se puede dejar por defecto C:/AppServ y se creará una carpeta en nuestro disco C con este nombre.

La siguiente pantalla nos pide que le indiquemos cómo queremos hacer la instalación, si queremos instalar por primera vez el AppServ seleccionamos la instalación Typical y pulsamos sobre Next.

En este punto llegamos a la configuración del servidor Apache, debiendo ingresar en el campo del host la dirección IP pública asignada al equipo, para lograr acceder desde el exterior. En el correo del administrador se puede poner un correo existente, aunque si no se tiene instalado un servidor de correo en el ordenador no se recibirán las incidencias que puedan enviar los usuarios o el propio servidor.

Por último se debe configurar el programa de bases de datos MySQL. Pide un nombre de usuario y su contraseña, que se rellena teniendo en cuenta que esta información es necesaria a la hora de referenciar las bases de datos.

8.2. Desarrollo de la interfaz remota

La interfaz remota constituye el punto de acceso por parte de los usuarios al equipo, que les sirve para observar la evolución del sistema y operar en él. Es accesible a través de un navegador, puesto que el resultado de las consultas que se realizan es código HTML

generado en el servidor según el estado del sistema.

Los archivos que forman la interfaz para controlar remotamente el equipo contienen código PHP, programado para generar un documento HTML, accesible desde cualquier navegador en el equipo cliente. Por tanto, desde el lado del usuario lo que se ve es una página Web normal en la que puede introducir mediante formularios los valores deseados para el funcionamiento del equipo.

8.2.1. Introducción a PHP

PHP es uno de los lenguajes de lado servidor más extendidos en Internet. Se trata de un lenguaje de creación relativamente creciente que ha tenido una gran aceptación debido sobre todo a la potencia y simplicidad que lo caracterizan.

Permite incluir pequeños fragmentos de código dentro de la página HTML y realizar determinadas acciones de una forma fácil y eficaz. Por otra parte, PHP ofrece muchas funciones para la explotación de bases de datos, gestión de archivos, funciones de correo electrónico, tratamiento de imágenes,funciones matemáticas, trabajo con cadenas,...

PHP se escribe dentro de la propia página web, junto con el código HTML y, como para cualquier otro tipo de lenguaje incluido en un código HTML, necesitamos especificar cuáles son las partes del código escritas en este lenguaje. Esto se hace delimitando el código por etiquetas. Hay distintas posibilidades, siendo la utilizada en los archivos de la interfaz remota $<?php$ para indicar el comienzo del script y $?>$ para el final.

El modo de funcionamiento de una página con contenido en PHP se centra en que el servidor va a reconocer la extensión correspondiente a la página PHP y antes de enviarla al navegador va a encargarse de interpretar y ejecutar todo aquello que se encuentre entre las etiquetas correspondientes al lenguaje PHP. El resto, lo enviará sin más, ya que asumirá que se trata de código HTML absolutamente comprensible por el navegador.

La sintaxis de PHP es relativamente parecida a la de C, salvo algunas diferencias. No es el objeto de este proyecto describir la sintaxis de PHP, a la que se puede acceder a través de cualquier manual que se puede encontrar en el sitio oficial. Simplemente indicar algunas diferencias respecto al lenguaje C, con los que se puede comprender el funcionamiento de los ficheros que componen la página para el control del equipo y que se encuentran listados en los anexos.

En PHP las variables se indican usando como carácter inicial el símbolo $. No hace falta declararlas ni indicar el tipo, el procesador las tratará según la operación en la que se encuentren.

Existen variables del sistema, como puede ser $_POST o $_GET, que son tablas que contienen los parámetros en las peticiones de tipo POST y GET respectivamente.

Las tablas pueden ser indexadas mediante cadenas de caracteres, que deben ir entre comillas.

Para concatenar cadenas se utiliza el operador '.'.

8.2.2. Descripción de la página Web

La página se estructura basándose en el uso de marcos. Inicialmente es necesario introducir un nombre de usuario y clave válido. Una vez identificado el usuario, el marco superior presenta el menú de navegación, desde el que se puede acceder a:

Control Permite, en caso de que el equipo esté libre para ser usado, acceder a gestionar la operación del equipo. Se distinguen tres zonas con diferentes objetivos, tal y como se puede apreciar en las figura 8.1.

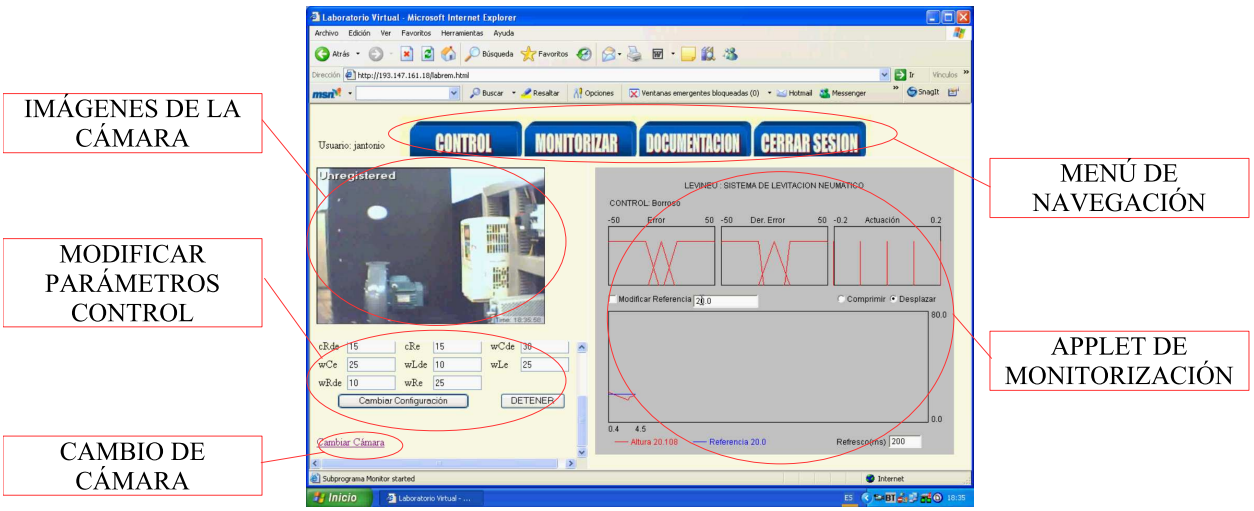


Figura 8.1: Interfaz para el control remoto

En primer lugar se incluye la imagen de la cámara Web para poder observar el comportamiento del equipo como si se estuviese en el propio laboratorio. Existe la posibilidad de cambiar de cámara, mostrando las imágenes capturadas por el sistema de visión.

A su derecha aparece el applet de monitorización que muestra el estado del equipo y la evolución de la altura y la referencia.

El último marco contiene distintos formularios que se utilizan para elegir un control entre las posibilidades existentes y ejecutarlo, pudiendo modificar en línea sus parámetros. Una vez terminado el experimento solicita el nombre del fichero en el que guardar los datos en su cuenta.

Monitorización Muestra las imágenes de las cámaras y el applet, pero sin posibilidad de modificar la referencia. Se puede supervisar la evolución del sistema pero no actuar en él.

Documentación Permite acceder a los archivos guardados en la cuenta de usuario, para recuperar la información de los experimentos realizados.

Gracias a la estructura empleada, si se añade un nuevo control en el equipo, la interfaz remota no hay que modificarla. Simplemente hay que insertar correctamente el nuevo control en la base de datos y sus parámetros, ya que cuando desde el navegador le aparece a un usuario la posibilidad de elegir entre distintos controles, este campo de selección se ha generado tras consultar en la base de datos los controles existentes.

8.2.3. Ficheros que componen la interfaz remota

En la sección de anexos se incluye el código de los archivos programados. Aquí se indica la función de cada uno de ellos:

labrem.html Página principal, que la divide en dos marcos: la cabecera superior y la portada.

cabecera.php Cabecera de la página que inicialmente sirve para introducir el nombre y clave, y una vez identificado presenta el menú de navegación.

portada.html Página de presentación.

sesiones.php Controla la sesión del usuario, no permitiendo acceder a zonas que se necesite identificación sin estarlo.

bdd.php Conecta a la base de datos.

login.php Valida el par nombre de usuario clave.

remoto.php Acceso al control remoto. Divide la página para mostrar los distintos elementos del control remoto.

activex.php Muestra las imágenes de las cámaras según la que se haya seleccionado.

app.php Incluye el applet de monitorización. Según el usuario tenga el control o no incluye la posibilidad de modificar la referencia.

inicio.php Formulario inicial dentro de la interfaz de control. Permite seleccionar entre los controladores existentes o enviar una DLL con un controlador programado por el usuario.

carga.php Detiene la aplicación de control y copia la DLL en el lugar adecuado.

parametro.php Formulario para seleccionar los parámetros del control seleccionado y el tiempo de muestreo.

control.php Permite modificar los parámetros del control en línea y detener el experimento realizado.

fin.php Una vez finalizado el ensayo, sirve para introducir el nombre del archivo en que guardar los datos. Muestra los existentes en la cuenta de usuario, puesto que si se introduce el nombre de uno de los que hay se sobreescribirá.

fichero.php Copia el fichero con los resultados en la cuenta del usuario y vuelve al inicio.

monitoriza.php Acceso a la monitorización. Muestra las cámaras y el applet de monitorización.

ficheros.php Muestra los archivos existentes en la cuenta del usuario.

desc.php Cierra la sesión.

8.2.4. Generación del documento XML

Para que el applet de monitorización funcione es necesario que tenga acceso a un documento XML que describa el estado del sistema en cada instante.

Este documento se debe generar dinámicamente en el servidor. El fichero *levineu.php*, se encarga de ello.

A través del protocolo http se puede enviar cualquier tipo de archivo. Desde el applet se solicita al servidor el archivo levineu.php y la respuesta es el documento XML que se interpreta de forma adecuada en el applet.

Previamente se comprueba que el usuario conectado tenga los permisos adecuados y se realizan las consultas a la base de datos necesarias para generar el documento. A través de una petición de tipo GET se permite además modificar la referencia, siempre y cuando el usuario tenga el control.

Así, ante la petición

```
http://193.147.161.18/levineu.php
```

La respuesta del servidor será un documento XML con el estado del sistema y que sigue la estructura descrita en el capítulo anterior.

Para además modificar la referencia, que se hace desde el applet, la solicitud debería ser del tipo

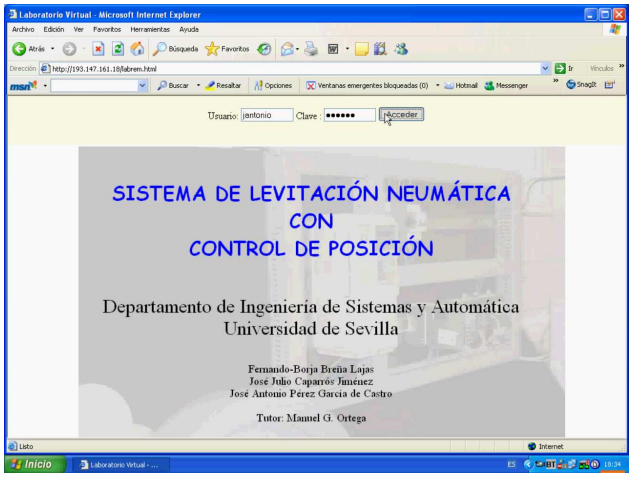
```
http://193.147.161.18/levineu.php?ref=20.5
```

En este caso, si se dispone del control del equipo, la referencia pasaría a ser 20,5.

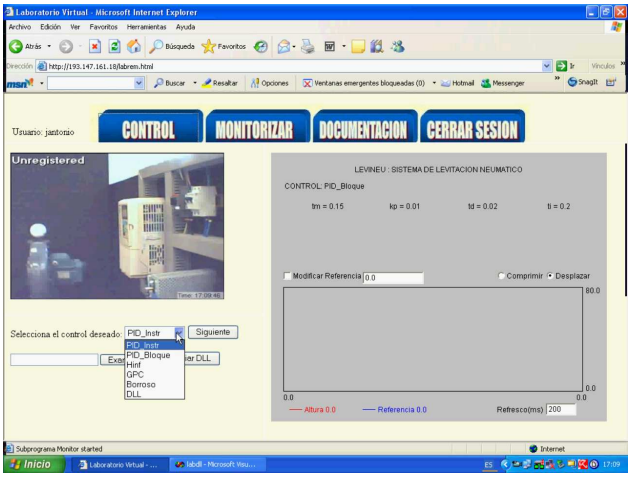
8.3. Pasos para la realización de un experimento remoto

En esta sección se muestran los pasos a dar para realizar pruebas con los distintos controladores de forma remota.

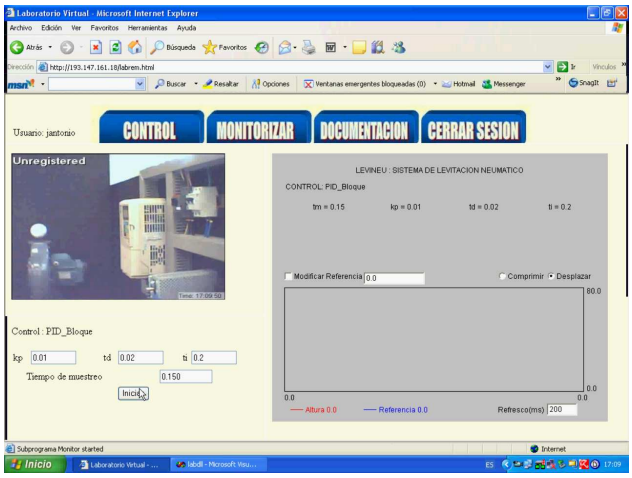
1. Conectarse a través de un navegador a la dirección *http://193.147.161.18/labrem.html*.
2. Introducir un nombre de usuario y contraseña válido.
3. Elegir la opción *control*.
4. Seleccionar el control deseado y pulsar en *Siguiente*. En caso de querer utilizar un control propio programado en una DLL se selecciona el archivo con la DLL y se envía.
5. Indicar el valor deseado de los parámetros e iniciar el ensayo.
6. Es posible modificar los parámetros del control pulsando en *Cambiar configuración*. La referencia se modifica en el applet. Para finalizar el experimento se pulsa en *Parar*.
7. Elegir un nombre para almacenar los datos en un fichero en la cuenta de usuario.
8. Para acceder a la documentación se accede a la pestaña *Documentación* del menú de navegación. Los ficheros se pueden representar fácilmente en Matlab para ser analizados.
9. Para liberar el uso del equipo se selecciona *Cerrar Sesión*.



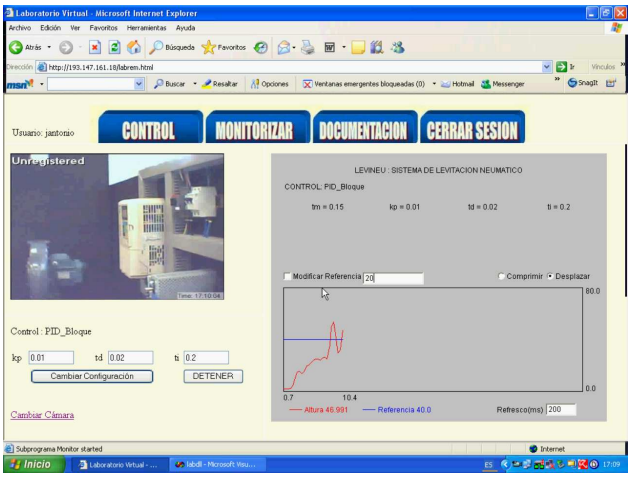
(a) Paso 2



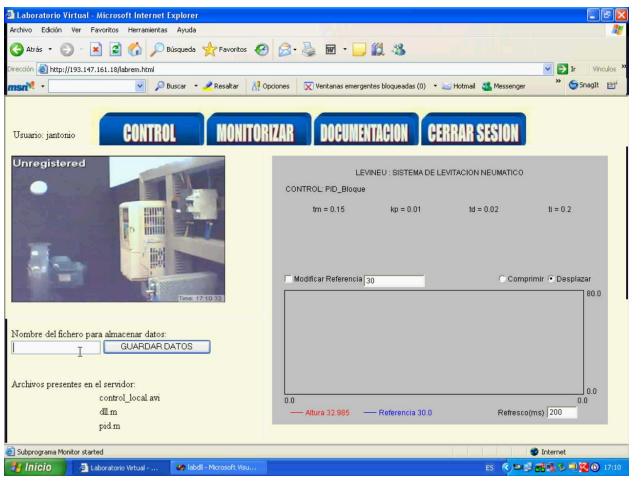
(b) Paso 4



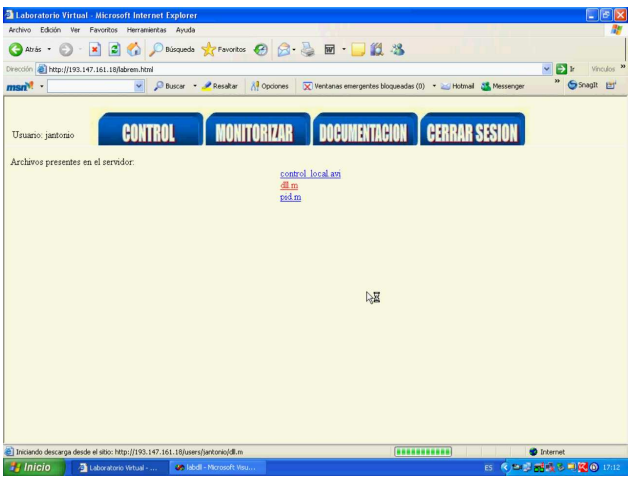
(c) Paso 5



(d) Paso 6



(e) Paso 7



(f) Paso 8

Figura 8.2: Pasos para la ejecución de un experimento remoto

Capítulo 9

Conclusiones

A lo largo de este proyecto se ha pormenorizado la posibilidad de acceder, tanto para monitorización como para control, a un equipo genérico, y en concreto a un Sistema de Levitación Neumática. Para lograr dicho acceso simplemente se necesita un ordenador con conexión a Internet.

Entre las distintas soluciones planteadas se ha usado una base de datos como elemento de conexión entre el funcionamiento local y el remoto, porque ofrece mejores prestaciones en cuanto a velocidad de las conexiones y permite añadir nuevas funcionalidades de forma sencilla. La estructura empleada hace que la plataforma que se ha llevado a cabo esté capacitada para ampliar fácilmente el número de equipos accesibles remotamente. Por otra parte, el hecho de que el elemento central sea una base de datos puede aprovecharse para desarrollar funciones que mejoren la gestión de usuarios y equipos, algo necesario en caso de componer un laboratorio remoto que posibilitara a los alumnos la realización de distintas prácticas.

El protocolo Host Link utilizado en las comunicaciones con el autómata ha permitido modificar los parámetros de funcionamiento del PLC en línea. Este aspecto es fundamental para poder estudiar el comportamiento de los controladores con distintos ajustes desde fuera del laboratorio.

Los módulos para uso de SQL desde LabVIEW ofrecen un acceso sencillo a la base de datos, de forma que las modificaciones que haya realizar en una aplicación de control basada en LabVIEW no requieran amplios conocimientos del estándar SQL.

La interfaz Web, gracias al uso de elementos de monitorización como la cámara o el applet, facilita la supervisión del equipo al realizar los ensayos, haciendo presente el entorno de la planta al usuario. El poder enviar un controlador programado y probarlo en el equipo real ofrece mucha versatilidad a la operación remota.

Durante los ensayos efectuados se ha podido apreciar el efecto de los retardos producidos en las comunicaciones a través de la red TCP/IP. En cualquier caso, dado que los controles se llevan a cabo localmente y remotamente se modifican los parámetros del control, dicho retardo simplemente afecta a la monitorización del sistema. La gráfica generada en la interfaz no refleja completamente la evolución de la esfera, puesto que la diferencia de tiempo entre una muestra y otra depende de los retrasos en la comunicación y, por tanto, es variable. Cuando la diferencia entre una muestra y otra varía demasiado, se pierden puntos resultando líneas rectas en la gráfica. No obstante, esta circunstancia se detecta perfectamente durante la ejecución del experimento y no afecta a su desarrollo, dado que la información almacenada en el fichero de la cuenta contiene todas las muestras pues se genera localmente.

Una situación diferente sería que la señal de control se calculase en el equipo remoto actuando sobre la planta del laboratorio. En este caso un aspecto muy importante a estudiar consistiría en el efecto de los retardos variables de las comunicaciones a través de Internet.

ANEXOS

Apéndice A

Manual de uso y programación del autómata CJ1M de Omron

A.1. Autómata CJ1M de Omron

A.1.1. Elementos

El CJ1M es un autómata modular, del que en el laboratorio se dispone de los siguientes bloques:

- **Fuente de alimentación**. Se conecta a la red eléctrica y proporciona a la CPU la tensión necesaria para su funcionamiento.
- **CPU**. Concretamente se trata de la CPU11, que incluye un puerto serie, un puerto de periférico (puerto específico de OMRON), una ranura para tarjeta de memoria y un DIP configurable con ocho pines. De las distintas CPU de la serie CJ1M, la CPU11 es la más compacta, por lo que no incorpora unidades de E/S.
- **Salidas analógicas**. El módulo CJ1W-DA021 contiene dos salidas analógicas, que pueden funcionar en los rangos 4:20mA, 1:5V, 0:5V, 0:10V ó -10:10V.

Por tanto el autómata constará únicamente del bastidor de la CPU (existe la posibilidad de utilizar bastidores expansores), formado por la fuente de alimentación, la CPU, las salidas analógicas y la tapa final.

A.1.2. Configuración del autómata

Se puede diferenciar entre dos tipos de configuraciones, según se haga a nivel de hardware o de software.

La configuración hardware se refiere a la manipulación de los ocho pines del interruptor DIP presente en el frontal del dispositivo, que se encuentra protegido bajo una pestaña junto a la batería. La función de cada uno de ellos es:

1. Inhabilita la escritura en la memoria de programa del usuario, cuando está activo.
2. Cuando está a ON el programa de usuario se transfiere automáticamente y se ejecuta al conectar la alimentación.
3. No se utiliza.
4. Configura el puerto de periféricos. Si está a ON arranca con los parámetros predeterminados, mientras que si está a OFF lo hace con los definidos en la configuración del PLC.
5. Determina el modo de funcionamiento del puerto serie de la CPU. Si está activo, el puerto serie utiliza el protocolo Toolbus, usando el definido en la configuración si está a OFF.
6. Pin que puede ser utilizado para funciones definidas por el usuario. Al activarse se refleja en el indicador A39512.
7. Configura una copia de seguridad sencilla. Si está a OFF, se verifica el contenido de la tarjeta al mantener pulsado el interruptor de alimentación de la tarjeta durante tres segundos. Si está a ON al arrancar el PLC, leerá de la memoria de la tarjeta a la CPU. Cuando está a ON y se pulsa el botón de la alimentación de la tarjeta durante tres segundos se escribe desde la CPU en la tarjeta.
8. Siempre debe estar a OFF

La configuración software se hace desde una consola de programación o a través del CX-Programmer, pudiéndose configurar el modo de arranque, el modo de funcionamiento de los puertos,... En el apartado A.3.4 se explica como hacerlo usando CX-Programmer.

A.1.3. Área de memoria de E/S

El área de memoria de E/S contiene las áreas de datos a las que se puede acceder mediante los operandos de las instrucciones. Estas áreas de datos incluyen las áreas CIO, de trabajo, de retención, auxiliar, DM, EM, de temporizador, de contador, de indicadores de tarea, de indicadores de condición y de impulsos de reloj, así como los registros de datos y registros de índice.

CIO. No es necesario escribir el acrónimo CIO al especificar una dirección del área CIO. Normalmente, esta área se utiliza para los intercambios de datos, como por ejemplo el refresco de E/S de varias unidades. Los canales que no estén asignados a ninguna

unidad se pueden utilizar como canales y bits de trabajo sólo en el programa. En el caso del equipo del laboratorio, las únicas direcciones que no se pueden usar para otros propósitos van desde CIO 2000 a CIO 2009, que son asignadas a la unidad de salidas analógicas.

Área de trabajo WR. Los canales del área de trabajo sólo pueden utilizarse en el programa. No pueden emplearse para el intercambio de E/S con terminales de E/S externos. Como no se utilizan para otras funciones es conveniente utilizar estos canales antes que los del área CIO.

Área de memoria de datos(DM). El área DM contiene 32.768 canales, con direcciones desde D00000 hasta D32767. Este área de datos se utiliza para el almacenamiento y manipulación general de datos, y el acceso a la misma es exclusivamente mediante canal. Los datos se mantienen al conectar la alimentación del PLC o al cambiar el modo de funcionamiento. Puede realizarse direccionamiento indirecto, en modo binario o BCD. El direccionamiento indirecto en modo binario se indica mediante @D, y el valor de la zona indicada se toma como la dirección del operando. Igual en modo BCD que se indica mediante *D, sólo que la dirección se toma interpretando el valor indicado en BCD. En este segundo caso sólo será posible direccionar la memoria D desde D00000 hasta D09999. Hay que tener en cuenta a la hora de utilizar el área de memoria de datos que hay una parte que se asigna a las unidades de entrada y salida especiales, como el módulo de salidas analógicas del que se dispone en el laboratorio y que si se configura como unidad 00, utiliza cien canales a partir de la dirección D20000.

Área de retención. Contiene 512 canales, cuyas direcciones van desde H000 hasta H511 (bits H00000 hasta H51115). Estos canales sólo se pueden utilizar en el programa.

Área auxiliar. Contiene 960 canales, con direcciones que van desde A000 hasta A959. Estos canales han sido preasignados como indicadores y bits de control para las operaciones de supervisión y control. Las direcciones A000 hasta A447 son de sólo lectura, aunque se podrá leer o escribir en A448 hasta A959 desde el programa o desde un dispositivo de programación. Contiene información de la configuración, para la depuración, de errores de programa, del reloj, de tareas,... Para información detallada de las funciones de cada canal debe consultarse el apéndice B del manual de operación.

Área de temporizador. Existen dos zonas de temporizadores, una para los indicadores de final de cuenta y otra para los valores actuales. Se pueden utilizar hasta 4096 temporizadores, usándose los números T0000 a T4096 para acceder a ambas áreas. Si se utiliza un número de temporizador en un operando que requiere datos de bit, este número es el que accede a los indicadores de finalización del temporizador. Si se utiliza un número de temporizador en un operando que requiere datos de canal, este número es el que accede al valor actual del temporizador. Los indicadores de finalización del temporizador se pueden utilizar tantas veces como sea necesario como condiciones, y los valores actuales del temporizador se pueden leer como datos de canal normales.

Área de contador. Existen 4096 números de contador (C0000 hasta C4095), que se usan de la misma forma que los de temporizador.

Indicadores de condición Estos indicadores incluyen los indicadores aritméticos, que señalan los resultados de la ejecución de instrucciones, y los indicadores de siempre en ON y siempre en OFF. Los indicadores de condición se especifican con símbolos y no con direcciones.

Impulsos de reloj El temporizador interno de la CPU pone en ON y OFF los distintos impulsos del reloj. Estos bits se especifican con símbolos en lugar de con direcciones.

Área de indicador de tarea TK. Los indicadores de tarea, desde TK00 hasta TK31, se corresponden con las tareas cíclicas de 0 hasta 31. Un indicador de tarea se pondrá en ON si la tarea cíclica correspondiente está en estado ejecutable , y en OFF si no ha sido ejecutada o está en espera.

Registros índice IR Estos registros (de IR0 hasta IR15) se utilizan para almacenar direcciones en la memoria del PLC (direcciones de memoria absolutas en RAM), con el objeto de direccionar indirectamente los canales de la memoria de E/S. Los registros de índice pueden ser utilizados individualmente por cada tarea, o bien ser compartidos por todas las tareas.

Registros de datos DR Estos registros (DR0 hasta DR15) se utilizan conjuntamente con los registros de índice. Si se introduce un registro de datos justo delante de un registro de índice, el contenido del primero se sumará a la dirección de memoria del PLC en el registro de índice para desplazar dicha dirección. Los registros de datos pueden ser utilizados individualmente por cada tarea, o bien ser compartidos por todas las tareas.

A.1.4. Área de parámetros

El área de parámetros contiene diversas configuraciones que no pueden especificarse mediante los operandos de las instrucciones y que sólo pueden especificarse con un dispositivo de programación. Estas especificaciones incluyen la configuración del PLC, la tabla de E/S, la tabla de rutas y la configuración de la Unidad de bus de CPU.

A.1.5. Modos de funcionamiento

PROGRAM Se utiliza este modo para editar el programa o realizar operaciones de configuración y transferencia de programas. Todas las tareas se detienen.

MONITOR Este modo se utiliza para realizar pruebas y la edición online. La ejecución de las tareas en este modo es la normal (ver la sección A.1.7).

RUN Modo de ejecución normal.

A.1.6. Instrucciones

Hay una amplia gama de instrucciones que pueden utilizarse en el autómata CJ1M. La documentación disponible es extensa y detallada en el uso de cada una, por lo que en este documento simplemente se va a comentar cómo se pueden localizar las instrucciones necesarias y como encontrar la información existente sobre ellas.

Por lo general una instrucción se compone de una condición de ejecución, una serie de operandos de entrada y de salida y, en algunos casos, condición de ejecución de salida. Dependiendo de la posición de la instrucción, la condición de ejecución puede ser o no obligatoria. Así, las instrucciones de entrada que se conectan directamente a la barra izquierda no necesitan condición de ejecución, sino que indican un inicio lógico que será la condición de ejecución de posteriores instrucciones. Las instrucciones intermedias toman la condición de ejecución y la envían a las siguientes. Por último, las de salida se conectan directamente a la barra derecha, necesitando por lo general una condición de entrada, salvo para algunas instrucciones como END.

Una instrucción se ejecutará siempre que la condición de ejecución sea cierta. Existen variaciones, para que una instrucción se ejecute cuando la condición pase de OFF a ON o de ON a OFF. Una instrucción diferencial ascendente se indica con el símbolo '@' y las descendente con '%'.

Para conocer el uso y funcionamiento de una determinada instrucción simplemente hay que acudir al *Manual de Referencia de Instrucciones*, documento W340 de la librería técnica de OMRON. Concretamente, en la sección 2-3 del manual se encuentran por orden alfabético todas las instrucciones indicando la página en la que se explica cada instrucción.

Para encontrar una instrucción que realice una función deseada se puede acudir a la sección 2-2 donde se listan las instrucciones agrupadas según su función incluyendo un breve resumen para explicando la funcionalidad de la instrucción. Otra forma de buscar una instrucción es a través de la ayuda del software de programación, donde se puede acceder fácilmente a un comentario de cada una de las instrucciones disponibles también agrupadas por su función. En cualquier caso, esta ayuda es insuficiente para usar correctamente las instrucciones pues en la mayoría de los casos se limita a indicar el resultado que produce y las zonas de memoria que se pueden usar en cada uno de los operandos, no obstante sí es útil para encontrar una instrucción determinada y, posteriormente, se puede acudir al *Manual de Referencia de Instrucciones*.

Las funciones en las que se dividen el conjunto de instrucciones de la serie CJ se muestra a continuación para que sirva de guía a la hora de buscar nuevas instrucciones.

- Instrucciones de entrada de secuencia.
- Instrucciones de salida de secuencia.
- Instrucciones de control de secuencia.
- Instrucciones de temporizador y contador.

- Instrucciones de comparación.
- Instrucciones de transferencia de datos.
- Instrucciones de desplazamiento de datos.
- Instrucciones de aumento o disminución.
- Instrucciones matemáticas de símbolos.
- Instrucciones de conversión.
- Instrucciones de operaciones lógicas.
- Instrucciones matemáticas especiales.
- Instrucciones matemáticas de coma flotante.
- Instrucciones de coma flotante de doble precisión.
- Instrucciones de procesamiento de datos de tablas.
- Instrucciones de control de datos.
- Instrucciones de subrutinas.
- Instrucciones de control de interrupción.
- Instrucciones de paso.
- Instrucciones de Unidades de E/S básicas.
- Instrucciones de comunicaciones serie.
- Instrucciones de red.
- Instrucciones de memoria de archivos.
- Instrucciones de visualización.
- Instrucciones de reloj.
- Instrucciones de depuración.
- Instrucciones de diagnóstico de fallos.
- Otras instrucciones.
- Instrucciones de programación de bloques.
- Instrucciones de procesamiento de cadenas de texto.
- Instrucciones de control de tareas.

Instrucciones de movimiento de datos

BSET	(071)	Rellenar Bloque
COLL	(081)	Recogida de Datos
DIST	(080)	Distribución de Datos
MOV	(021)	Mover
MOVB	(082)	Mover Bit
MOVD	(083)	Mover Dígito
MOVL	(498)	Mover Doble
MOVR	(560)	Mover a Registro
MOVRW	(561)	Mover PV del temporizador/contador al registro
MVN	(022)	Mover negado
MVNL	(499)	Mover negado doble
XCGL	(562)	Intercambio doble de datos
XCHG	(073)	Cambiar Dato
XFER	(070)	Transferencia Bloque
XFRB	(062)	Transferir Bits

Figura A.1: Instrucciones de desplazamiento de datos

Así, si por ejemplo se necesitase una instrucción para copiar datos de una dirección de memoria a otra, acudiría a la sección de instrucciones de desplazamiento de datos. De las posibles instrucciones es MOV la más adecuada. Si se acude al manual puede verse que contiene dos operandos, el primero que es el origen y puede ser una constante y el destino en el que se copian los datos del origen. Además se muestran las variaciones que admite, las áreas de programa en las que se puede utilizar, las especificaciones propias de cada operando, una descripción del funcionamiento y un ejemplo de uso ilustrativo. En este caso se trata de una instrucción sencilla, pero la información es muy útil para usar correctamente aquellas instrucciones que incluyen entre los operandos campos de control.

A.1.7. Programas

Hay dos tipos de tareas o programas:

- Tareas cíclicas.
- Tareas de interrupción.

El funcionamiento básico de la CPU del autómata consiste en la ejecución de las tareas cíclicas que estén programadas. Se ejecuta consecutivamente desde la tarea con menor número hasta la de número más alto, para posteriormente pasar al refresco de la E/S y al procesamiento de los periféricos. Una vez completado el ciclo, se comienza a ejecutar nuevamente la primera tarea cíclica. Hay hasta 32 tareas cíclicas, de la 00 a la 31.

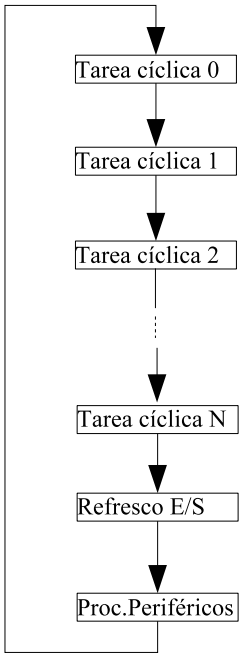


Figura A.2: Ejecución de tareas cíclicas

Las tareas cíclicas tienen distintos estados, que se pueden controlar con las instrucciones TKON y TKOF. Si una tarea está en estado READY, pasa a ejecutarse (estado RUN) cuando obtenga el derecho de ejecución. Si se encuentra en estado STANDBY no se ejecuta cuando le llega el derecho de ejecución. Con la instrucción TKON se pasa una tarea a estado READY y con TKOF a STANDBY. Hay un estado adicional que es el estado inhabilitado o INI, en el que se encuentran todas las tareas cuando el autómata se encuentra en modo Programa, en el que ninguna tarea se ejecuta.

Si se produce una interrupción, se detendrá la ejecución de tareas cíclicas para ejecutar la de interrupción. Tipos de tareas de interrupción:

- Tarea de interrupción de alimentación en OFF (número de tarea de interrupción 1), que se ejecuta cuando se desconecta la alimentación.
- Tareas de interrupción programadas. Como máximo son dos (números 2 y 3) y se ejecutan cada vez que pasa un tiempo fijo.
- Tareas de interrupción de E/S. Es necesario una unidad de entrada de interrupción para poder utilizarlas.
- Tareas de interrupción externas. Asociadas a unidades de E/S especiales o a unidades de bus de CPU. Puede haber hasta 256 tareas de interrupción externas.

Las tareas de interrupción pueden usarse como tareas cíclicas adicionales si se inician con TKON.

A.2. Salidas analógicas. CJ1W-DA021

El módulo de salidas analógicas dispone de dos salidas configurables. Necesita una alimentación de 24 V para funcionar correctamente, para lo que se utilizará la fuente disponible de OMRON, que también se emplea para alimentar el controlador del sistema de visión, dado que la intensidad proporcionada es suficiente como para alimentar ambos sistemas.

El rango de las señales de salida se puede ajustar entre los siguientes valores:

- 4 a 20 mA
- 1 a 5 V
- 0 a 5 V
- 0 a 10 V
- -10 a 10 V

En cualquiera de los casos el fondo de escala es de 4000 cuentas, por lo que la resolución que se puede alcanzar será igual a $\frac{Rango}{4000}$.

A.2.1. Configuración

Para el correcto funcionamiento de la unidad es necesario configurar la unidad, el modo de funcionamiento, y el formato de las salidas.

El número de unidad se asigna mediante dos ruedas numéricas presentes en el frontal del módulo.

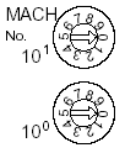


Figura A.3: Selección de unidad

Aparece en el módulo también un selector con dos pines, como puede verse en la figura A.4, que sirve para seleccionar el modo de funcionamiento de la unidad.

Figura A.4: Modo de funcionamiento

El pin 2 debe estar siempre en OFF. Si el pin 1 está en OFF el modo de funcionamiento será el normal, poniéndose a ON para el modo de ajuste, que se usa para la calibración.

Para la configuración y ajuste de la salida la unidad utiliza 10 canales en el área de memoria CIO y 100 en la DM. Los canales concretos que ocupa depende del número de unidad. Si la unidad es la 00, que es la que se utilizará en este caso, la memoria utilizada será de la CIO 2000 a la CIO 2009 y de la DM 20000 a la 20099.

En primer lugar hay que configurar adecuadamente la zona de memoria DM, que sirve para especificar las salidas utilizadas, el rango empleado y el estado de la salida cuando se para la salida. Se va a indicar la configuración suponiendo como unidad la 0, si cambiase el número de unida sería igual pero con dirección inicial DM 20100 para la unidad 1, 20200 para la 2,... Esta zona de memoria permanece con el mismo valor tras la desconexión de la alimentación.

En la posición DM 20000 se utilizan los dos bits menos significativos para indicar qué salida se está utilizando. El bit cero activo indica que la salida 1 se va a emplear. Igual el bit 1 respecto a la salida 2.

En la posición DM 20001 se indica el rango empleado en la salida. Los bits 0 y 1 para la primera salida y los bits 2 y 3 para la segunda. Los significados de cada uno de los valores son:

- 00: -10 a 10 V.
- 01: 0 a 10 V.
- 10: 1 a 5 V ó 4 a 20 mA, dependiendo del cableado.
- 11: 0 a 5 V.

Las siguiente posiciones de memoria se utilizan para especificar el valor de la salida correspondiente cuando la salida se deshabilita. Las posibilidades son dejar el último valor (valor 01), el mínimo (00) o el máximo (02). La posición DM 20002, de la que sólo se usa el octeto menos significativo, configura la salida primera, mientras que la DM 20003 hace lo mismo con la salida segunda.

En segundo lugar, los canales de la zona de memoria CIO se emplean para habilitar las salidas y fijar el valor deseado. En modo de funcionamiento normal, los dos últimos bits de la CIO 2000 sirven para habilitar las salidas. Un 1 en el bit cero habilita la salida primera, mientras que el bit uno hace lo propio con la segunda. En caso de que una salida esté habilitada, los números de cuentas se indican en las posiciones CIO 2001 y CIO 2002 para la primera y segunda respectivamente.

Voltage output 2 (+)	B1	A1	Voltage output 1 (+)
Output 2 (–)	B2	A2	Output 1 (–)
Current output 2 (+)	B3	A3	Current output 1 (+)
N.C.	B4	A4	N.C.
N.C.	B5	A5	N.C.
N.C.	B6	A6	N.C.
N.C.	B7	A7	N.C.
N.C.	B8	A8	N.C.
0 V	B9	A9	24 V

Figura A.5: Esquema de conexiones

A.2.2. Cableado

En la figura A.5, extraída del manual, se muestran los pines de conexión del módulo de salidas analógicas.

Hay que destacar que se necesita introducir una tensión de 24 voltios entre los pines B9 y A9, para alimentación. La salida en caso de que se trabaje en tensión se encuentra entre los pines A1 y A2, mientras que en corriente es entre A3 y A2. Para la salida dos, en tensión es entre los pines B1 y B2, y en corriente entre B3 y B2.

A.2.3. Caso particular

Una vez analizados los elementos que componen el módulo de salidas analógicas, se va a ver la configuración usada en el proyecto, como ejemplo ilustrativo de los pasos a dar para un correcto funcionamiento

Se quiere una salida en tensión de 0 a 10 voltios que se usará para seleccionar la frecuencia de salida del variador de velocidad. Se va a usar para este propósito la salida uno.

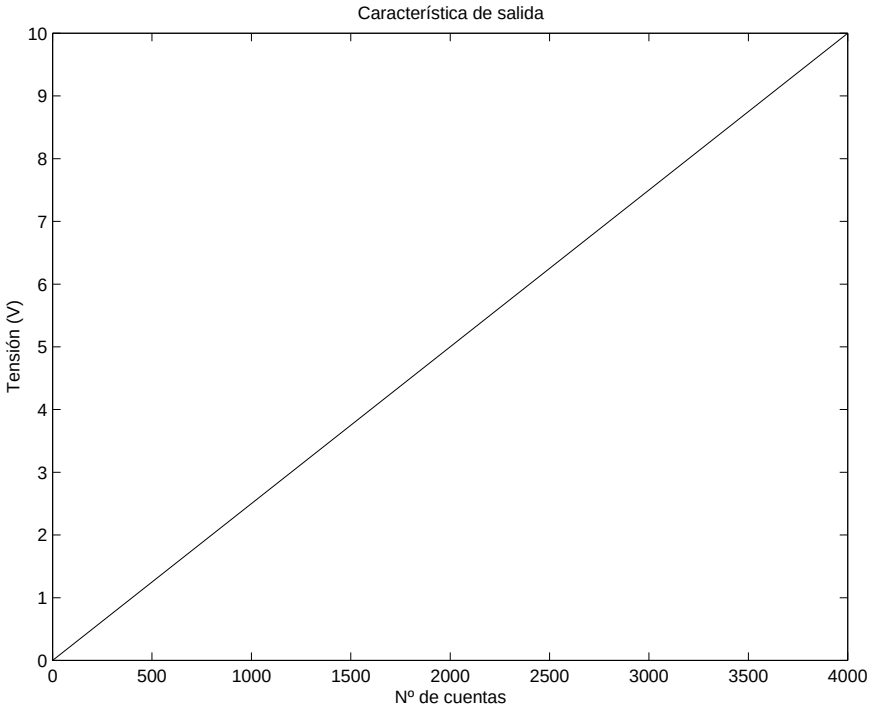


Figura A.6: Característica de salida

1. Seleccionar la unidad. En este caso será la unidad 0. Se colocan las ruedas selectoras de unidad en la posición 00.

2. Seleccionar el modo de funcionamiento como normal (pines 1 y 2 a OFF).

3. Para indicar que la salida 1 va a ser usada, hay que colocar un 0x0001 en la posición DM 20000.

4. Seleccionar el rango de la salida para que funcione de 0 a 10 voltios. Para ello tiene que haber en la dirección DM 20001 un 0x0001. Apagar y volver a conectar la alimentación, para que arranquen con la configuración adecuada.

5. Habilitar la salida, para lo que se pone un 0x0001 en la posición CIO 2000. Los pasos 3 y 4 se pueden hacer una única vez, ya que la zona de memoria DM permanece tras la desconexión de la alimentación. Sin embargo este paso hay que repetirlo siempre que se encienda el PLC, puesto que de lo contrario no se producirá la conversión y el valor de la salida permanecerá constante.

6. Colocar en CIO 2001 el valor de cuenta deseado entre 0 y 4000, correspondiendo el 0 a 0 V y el 4000 a 10 V.

7. Entre A1 y A2 habrá una tensión proporcional al número de cuentas indicado.

A.3. Software de programación. CX-Programmer Ver 5.0

Se dispone del software CX-Programmer Versión 5.0 para la programación del autómata. La instalación es muy sencilla, sólo hay que elegir el idioma, seleccionar el directorio deseado y seguir los pasos indicados. Además de CX-Programmer es necesario instalar para que funcione correctamente el CX-Server, el propio programa de instalación lo indica al final. También pregunta si se quiere instalar la biblioteca de Function Blocks de Omron.

En una primera ojeada a la pantalla del programa se distinguen distintas secciones, como puede verse en la figura A.7:

1. Zona de menús y botones de acceso directo a las distintas funcionalidades del programa, útiles para cambiar rápidamente el modo de funcionamiento del PLC, transferir los programas,...
2. Área de trabajo del proyecto, donde se muestran los distintos elementos del PLC con el que se está trabajando, como las variables definidas, la tabla de memoria, la memoria, la configuración, los programas,... Para acceder a cualquiera de ellos simplemente hay que hacer doble clic sobre el nombre.
3. Zona de edición de programas, bloques de función o símbolos.
4. Ventana de salida.

A.3.1. Creación de un nuevo proyecto

El primer paso a dar para programar un autómata de Omron con el CX-Programmer es crear un nuevo proyecto. Las instrucciones y los elementos del área de trabajo pueden cambiar del PLC con el que se trabaje, en este documento se supondrá siempre que el autómata es el CJ1M, concretamente la CPU 11.

Para crear un proyecto nuevo, se selecciona *Nuevo* en el menú *Archivo*. En la ventana de diálogo resultante, figura A.8, se le da el nombre deseado al dispositivo, se selecciona el tipo de autómata pudiendo configurar los parámetros propios del PLC, como es el tipo de CPU, y se elige el tipo de red para la comunicación con el autómata.

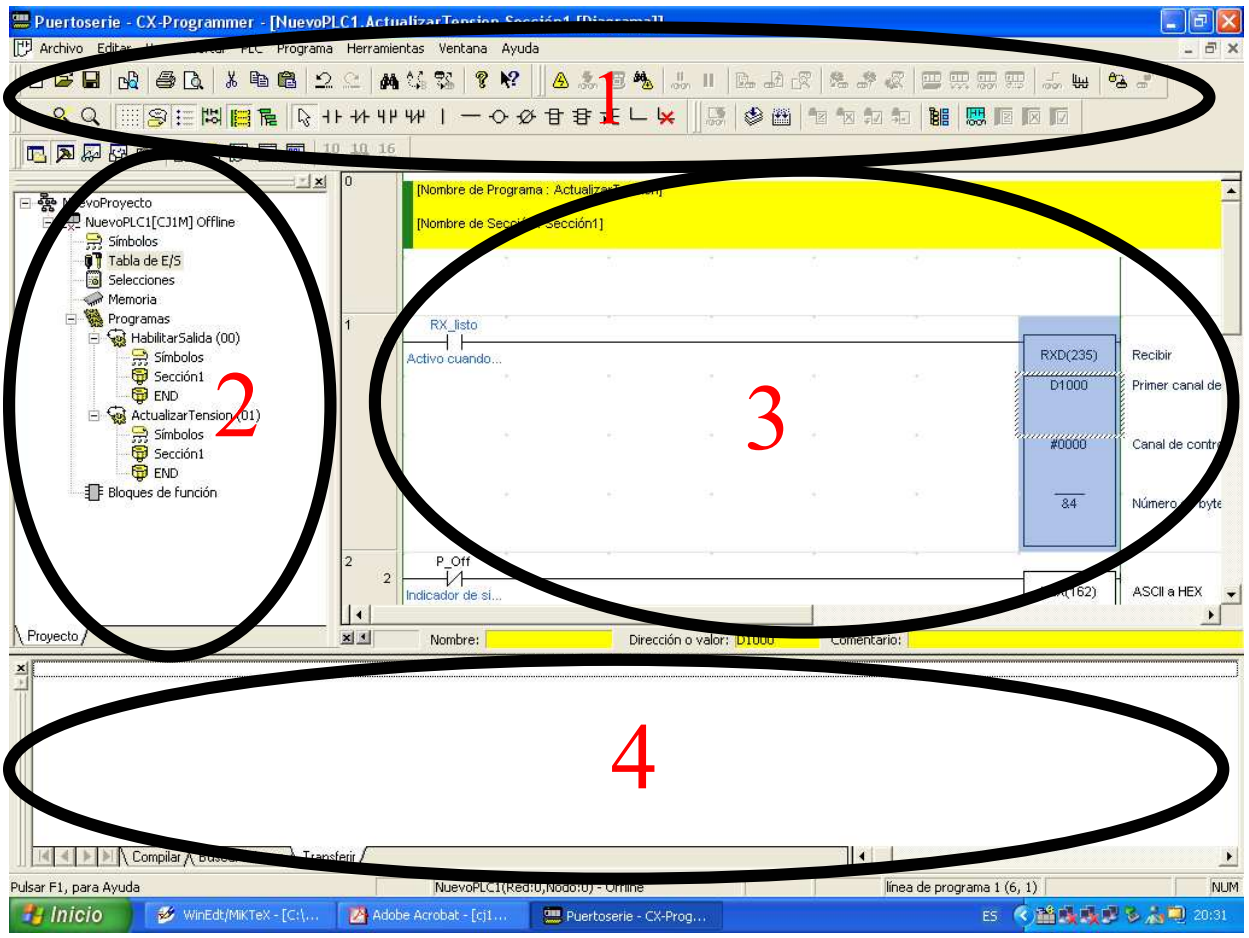


Figura A.7: Pantalla de CX-Programmer

Figura A.8: Inicio de un nuevo proyecto

Para el tipo de red hay distintas posibilidades, ya que es posible comunicarse a través del cable de periféricos (aunque en el laboratorio no se dispone de dicho cable), o a través del puerto serie. Por el puerto serie la comunicación puede realizarse a través del protocolo Host Link, para lo que hay que seleccionar SYSMAC WAY como tipo de red, o configurado como bus de periféricos seleccionando Toolbus. Lo más importante es que la configuración coincida con la del puerto serie del autómata a la hora de arrancar, ya que de lo contrario es imposible la comunicación. Pulsando en el botón *Configurar* junto al tipo de red, en la pestaña *controlador* de la ventana emergente, figura A.10, se indica el puerto del PC que se va a usar y los parámetros de funcionamiento. En caso de usar el protocolo Toolbus, sólo hay que especificar el puerto, porque el formato es siempre ocho bits de datos, sin paridad y un bit de stop, mientras que la velocidad la puede detectar automáticamente. Si se usa Host Link, el formato de trama usado por defecto en general en las comunicaciones de los autómatas de Omron es siete bit de datos, dos bits de parada y paridad par.

En el caso del equipo del laboratorio, el único modo de comunicación con el autómata es el puerto serie de la CPU, puesto que no se dispone de ningún módulo de comunicaciones. Por lo tanto el puerto serie se utiliza también en la aplicación para recibir los datos desde la interfaz de control. Para esta comunicación se decidió emplear modo Host Link. Por lo tanto, lo más sencillo es usar el modo Host Link también para la comunicación con el software de programación. También es posible usar el modo Toolbus, que por otra parte es el modo recomendado por el fabricante, puesto que es el más rápido. Para usarlo hay que modificar el DIP[1] de configuración del autómata que se encuentra tras la pestaña situada en la zona superior. Concretamente, el pin número cinco determina el modo de arranque del puerto serie de la CPU. Si está a ON funciona en modo Toolbus y si está a OFF con la configuración interna que tenga, en este caso modo Host Link. Es más sencillo usar las comunicaciones

[1]Para cambiar alguno de los pines del DIP la alimentación debe estar desconectada, para evitar que descargas dañen el autómata

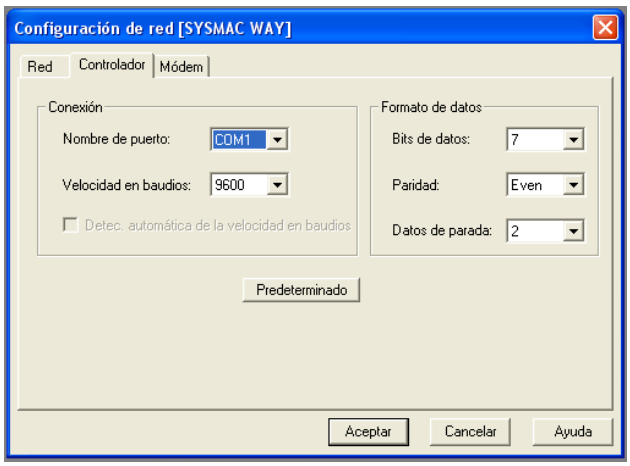


Figura A.9: Configuración del puerto serie para red SYSMAC WAY

Host Link en todos lo casos, puesto que no es necesario apagar y encender el autómata para cambiar la configuración de arranque del PLC cada vez que se quiere pasar de usar la aplicación de control al software de programación.

Una vez definido el autómata, aparece en el área de trabajo un nuevo elemento con el nombre dado al dispositivo, del que surge un árbol con diversos apartados, que cambian se está conectado o no al PLC. Haciendo doble clic en el nombre del autómata se pueden cambiar los parámetros indicados al crearlo. Para el caso del CJ1M y CPU11, los elementos que se encuentran son: Símbolos, Tabla de E/S, Selecciones, Memoria, Programas y Bloques de función. En los siguientes apartados se verá la funcionalidad de cada uno de ellos, a los que se accede con un doble clic.

A.3.2. Símbolos

Un símbolo es una zona de memoria a la que se le da un nombre para que sea más fácilmente identificable. Un símbolo se define mediante el nombre, que debe ser único, el tipo de dato y la zona de memoria a la que hace referencia. Al crear un nuevo dispositivo en el proyecto, aparecen ya definidos muchos símbolos, que no son editables, que se corresponden con distintos indicadores. Se puede crear un nuevo símbolo, haciendo clic con el botón derecho y seleccionando *insertar símbolo* en el menú contextual. Hay que darle un nombre, seleccionar un tipo de datos y asignarle una dirección válida. Si la dirección no es correcta o está fuera de rango no será posible completar la creación del símbolo. Opcionalmente, se puede añadir un comentario para aclarar el significado.

El tipo de dato determina la longitud e interpretación de la dirección a la que se accede a través del símbolo:

- BOOL Dirección de un bit binario, que se usa normalmente para contactos y bobinas.
- CHANNEL Tipo de dato especial que se usa para la compatibilidad con versiones

anteriores. Se trata de una dirección a cualquier dato que puede usarse en lugar de cualquier tipo de dato, excepto NUMBER y BOOL.

- DINT Dirección de un canal binario doble con signo.
- INT Dirección de un canal binario simple con signo.
- LINT Dirección de un canal binario cuádruple con signo.
- NUMBER Valor numérico literal. No es una dirección. El valor puede tener signo o ser de coma flotante. Este tipo de dato puede emplearse para cualquier valor literal o para identificadores del temporizador/contador (en tal caso, sólo se permiten valores enteros sin signo).
- REAL Dirección de un valor de coma flotante y canal doble, en formato IEEE.
- LREAL Dirección de un valor de coma flotante y canal largo, en formato IEEE.
- UDINT Dirección de un canal binario doble sin signo.
- UDINT_BCD Dirección de un canal BCD doble sin signo.
- UINT Dirección de un canal binario simple sin signo.
- UINT_BCD Dirección de un canal BCD simple sin signo.
- ULINT Dirección de un canal binario cuádruple sin signo.
- ULINT_BCD Dirección de un canal BCD cuádruple sin signo.
- WORD Dirección de una cadena de 16 bits.
- DWORD Dirección de una cadena de 32 bits.
- LWORD Dirección de una cadena de 64 bits.

Lo más importante de un símbolo es la dirección a la que está asociada. Para hacer referencia a una zona de memoria generalmente se hace con el nombre de la zona de memoria seguido de la dirección, sin ningún carácter de separación. Casos especiales son las zonas de memoria CIO y DM. La DM se indica mediante una 'd' y la CIO no necesita ningún nombre para ser referenciada, por lo que se indica simplemente el valor de la dirección del área CIO a la que se quiere asociar el símbolo.

La dirección será tratada según el tipo de datos seleccionado. Para los símbolos de tipo booleano, tomará los dos últimos dígitos de la dirección indicada como el bit que se quiere seleccionar, no teniéndolos en cuenta para la dirección, es decir un símbolo de tipo booleano asociado a la dirección 20007 hace referencia al bit 7 de la CIO 200. Esto se podría indicar explícitamente incluyendo un punto entre la dirección y el indicador de bit, así sería equivalente a poner 200.07, siendo así como lo muestra el CX-Programmer en la tabla de símbolos. Hay que tener en cuenta que hay zonas de memoria como la DM a las que es obligatorio acceder por canal, de forma que no se puede acceder a un único bit de la palabra, por lo que no se pueden definir símbolos de tipo booleano.

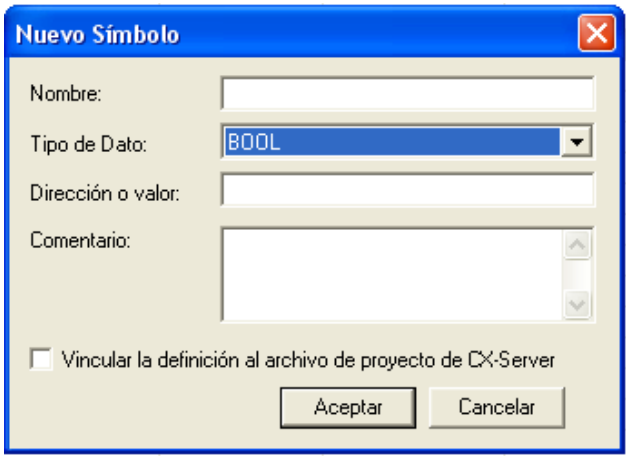


Figura A.10: Creación de un nuevo símbolo

A.3.3. Tabla de E/S

La tabla de E/S es la descripción de la configuración hardware del equipo. Hay que especificar el módulo presente en cada uno de los huecos. La tabla de E/S es muy importante, ya que define las zonas de memoria empleadas por las unidades. En el caso del equipo del laboratorio el único módulo que se tiene es el de saldas analógicas, que se encuentra por tanto en el primer hueco (hueco 00).

Para insertar un nuevo elemento en la tabla se selecciona el hueco correspondiente en el bastidor en el que se va a introducir el módulo, en nuestro caso en el principal y único que tenemos. Es posible tener hasta tres bastidores mediante el uso de unidades especiales de expansión, pudiendo tener hasta diez unidades en cada bastidor. Pulsando con el botón derecho del ratón sale en un menú contextual las posibles unidades entre las que hay que seleccionar la deseada. La unidad de salidas analógicas DA021 se encuentra bajo el conjunto de unidades especiales de entrada y salida (Unidad SIO CS/CJ) entre las salidas analógicas.

Otra posibilidad es transferir la tabla de entrada y salida directamente desde el autómata, para lo que es necesario estar conectado al PLC, que se puede hacer fácilmente mediante accediendo a *Trabajar Online* del menú *PLC*, o mediante el icono de acceso directo presente en la barra de herramientas. Para transferir la tabla desde el PLC, sólo hay que abrir la ventana de E/S y elegir la opción *Transferir del PLC* del menú *Opciones.* Si se elige crear la tabla de E/S a partir de la información que genera el PLC, siempre es conveniente comprobar que la tabla creada se corresponde realmente con la realidad, puesto que puede haber casos en los que algunas unidades especiales las confunda. De cualquier modo, este método es el más cómodo.

La tabla de E/S es un elemento muy importante y su incorrecta configuración suele llevar al autómata a un estado de alarma, generando un error de configuración de entrada y salida. Hasta que no se solucione el problema el autómata no comenzará la ejecución de ningún programa. Por eso, lo primero que se debe hacer al crear un nuevo proyecto es configurar adecuadamente la tabla de E/S o, a ser posible, transferirla desde el propio PLC.

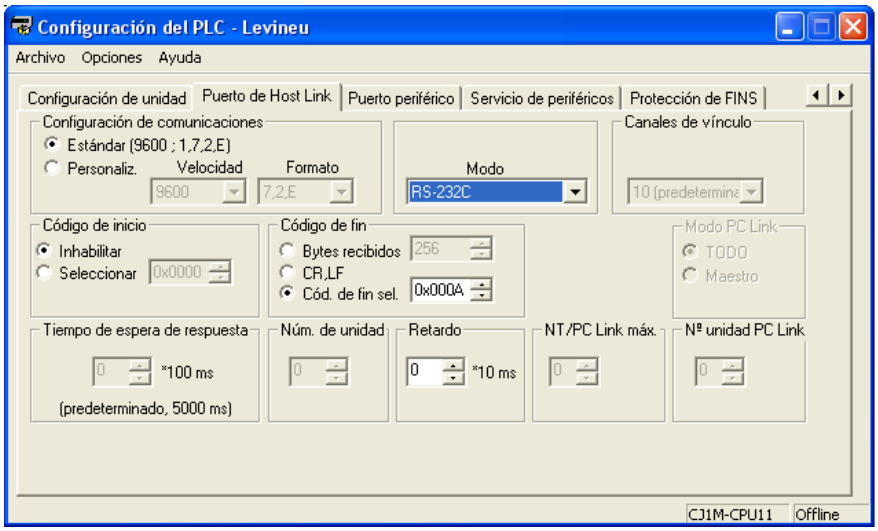


Figura A.11: Configuración del puerto serie del autómata

A.3.4. Selecciones

La tabla de selecciones permite configurar distintos aspectos del PLC, como son el arranque, algunos aspectos de las CPU, las temporizaciones, la actualización de las unidades especiales de entrada y salida, la unidad, el puerto Host Link, el puerto de periféricos, el servicio de periféricos o la protección de FINS.

La parte más importante desde el punto de vista del proyecto que se está desarrollando es la configuración del puerto Host Link, que corresponde con el puerto serie. La configuración que se indique en este apartado, siempre y cuando luego se transfiera al PLC, determinará el modo de funcionamiento del puerto serie cuando el pin cinco del DIP de la CPU del PLC esté a OFF. En el apartado *modo* hay que seleccionar el protocolo de funcionamiento. Si se va a emplear el modo sin protocolo hay que seleccionar modo RS232C e indicar el código de fin empleado o el número de bytes que forman una trama.

Hay que tener especial cuidado al cambiar la configuración del puerto serie, si este se está usando para comunicarse con el software de programación, puesto que podría impedir que la comunicación continuase. Por ejemplo, si se ha seleccionado como tipo de red para la comunicación con el PLC SYSMAC WAY, se estará utilizando el protocolo Host Link. Si una vez conectado al autómata se decide cambiar la configuración del puerto serie a modo sin protocolo, a partir de la transferencia de la tabla de selecciones se perderá totalmente la comunicación. En este caso, a la hora de comunicar con el CX-Programmer se debe emplear Toolbus, poniendo a ON el pin cinco del DIP de la CPU.

A.3.5. Memoria

Este apartado permite observar el valor que toman las distintas zonas de memoria, siempre que se esté trabajando on line. Además, si el PLC está funcionando en modo monitor,

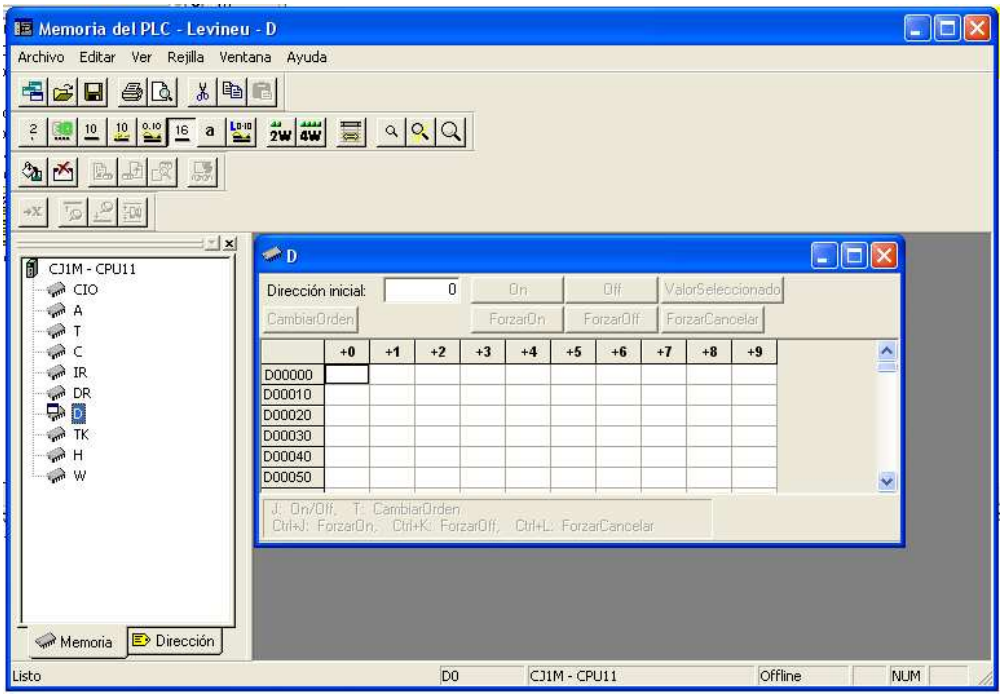


Figura A.12: Memoria

permite modificar los valores. Se puede visualizar la memoria en distintos formatos: binario, decimal codificado en binario, decimal, decimal con signo, hexadecimal, coma flotante, doble coma flotante, doble palabra o cuádruple palabra. Se puede seleccionar mediante el menú *Ver/Visualización* o mediante los iconos correpondientes.

Para monitorizar el contenido de una dirección, primero hay que seleccionar la zona de memoria en la que se encuentra, haciendo doble clic en ella. A continuación se busca la dirección deseada, para lo que se puede modificar el valor del campo dirección inicial. Si se quiere modificar el valor, hay que cambiar el PLC a modo monitor si no lo estaba previamente, y hacer doble clic en la recuadro correspondiente a la dirección que se va a cambiar, surgiendo una ventana de diálogo en la que se marca el valor deseado.

A.3.6. Programas

En esta sección se muestran las tareas que se han programado en el proyecto actual. Pueden existir hasta 32 tares de ejecución normal para el CJ1M, numeradas desde la 00 a la 31, además de las de interrupción. Para insertar una nueva tarea,hay que pinchar con el botón derecho sobre *Programas*, seleccionando *Insertar Programa*. Aperecerá en el árbol de tareas una nueva, sin asignación de número de secuencia. Accediendo a las propiedades a través del menú que surge al hacer clic sobre la tarea con el botón derecho del ratón, puede modificarse el nombre, añadir comentarios o, lo más importante, el tipo de tarea y orden de ejecución.

Junto con la nueva tarea habrá un acceso a los símbolos locales, que se tratan igual

que los del apartado A.3.2, pero que sólo pueden ser utilizados desde la tarea en la que se incluyen. También existirá dos secciones, una vacía y de nombre *Sección1* y otra llamada END y en la que lo único que aparece es una llamada a la instrucción END que debe aparecer al final de cada sección.

Cada sección se programa mediante el empleo de lenguaje de contactos, pudiendo incluir llamadas a bloques de función programados en lenguaje estructurado. En los bloques de función se emplea un lenguaje similar al Pascal.

Apéndice B

Script SQL que genera la estructura e información de la base de datos

```
# Base de datos : ‘cremoto‘
#
# --------------------------------------------------------
#
# Estructura de tabla para la tabla ‘comun‘
#

CREATE TABLE ‘comun‘ (
  ‘nombre‘ varchar(10) NOT NULL default ’’,
  ‘valor‘ float NOT NULL default ’0’,
  ‘id_eq‘ smallint(6) NOT NULL default ’0’,
  PRIMARY KEY  (‘nombre‘,‘id_eq‘)
) TYPE=MyISAM;

#
# Volcar la base de datos para la tabla ‘comun‘
#

INSERT INTO ‘comun‘ (‘nombre‘, ‘valor‘, ‘id_eq‘) VALUES (’tm’, ’0.15’, 1),
(’ref’, ’30’, 1),
(’altura’, ’26.291’, 1),
(’selector’, ’1’, 1),
(’conf’, ’0’, 1),
(’start’, ’0’, 1),
(’stop’, ’1’, 1),
(’detener’, ’0’, 1);

# --------------------------------------------------------
#
# Estructura de tabla para la tabla ‘control‘
#

CREATE TABLE ‘control‘ (
  ‘id‘ smallint(6) NOT NULL default ’0’,
  ‘nombre‘ varchar(10) NOT NULL default ’’,
```

```
  `id_eq` smallint(6) NOT NULL default '0',
  PRIMARY KEY  (`id`)
) TYPE=MyISAM;

#
# Volcar la base de datos para la tabla `control`
#

INSERT INTO `control` (`id`, `nombre`, `id_eq`) VALUES (0, 'PID_Instr', 1),
(1, 'PID_Bloque', 1),
(2, 'Hinf', 1),
(3, 'GPC', 1),
(4, 'Borroso', 1),
(5, 'DLL', 1);

# --------------------------------------------------------
#
# Estructura de tabla para la tabla `equipo`
#

CREATE TABLE `equipo` (
  `id` smallint(6) NOT NULL auto_increment,
  `nombre` varchar(20) NOT NULL default '',
  `descrip` tinytext,
  `estado` smallint(6) NOT NULL default '0',
  `id_admin` int(11) NOT NULL default '0',
  `id_control` int(11) default NULL,
  `activo` char(2) NOT NULL default 'no',
  `ult_acceso` timestamp(14) NOT NULL,
  PRIMARY KEY  (`id`)
) TYPE=MyISAM AUTO_INCREMENT=2 ;

#
# Volcar la base de datos para la tabla `equipo`
#

INSERT INTO `equipo`
(`id`, `nombre`, `descrip`, `estado`, `id_admin`, `id_control`,
 `activo`, `ult_acceso`)
VALUES
(1, 'levineu', 'Sistema de posicionamiento neum{\'a}tico', 0, 1, 1,'no',
 20050824192437);

# --------------------------------------------------------
#
# Estructura de tabla para la tabla `parametro`
#

CREATE TABLE `parametro` (
  `nombre` varchar(10) NOT NULL default '',
  `valor` float NOT NULL default '0',
  `id_cont` smallint(6) NOT NULL default '0',
  PRIMARY KEY  (`nombre`,`id_cont`)
) TYPE=MyISAM;
```

```
#
# Volcar la base de datos para la tabla ‘parametro‘
#

INSERT INTO ‘parametro‘ (‘nombre‘, ‘valor‘, ‘id_cont‘)
VALUES
(’kp’, ’0.0003’, 0),
(’td’, ’5’, 0),
(’ti’, ’0.5’, 0),
(’kp’, ’0.0005’, 1),
(’td’, ’3’, 1),
(’ti’, ’0.3’, 1),
(’cLe’, ’-25’, 4),
(’cRe’, ’25’, 4),
(’cCe’, ’0’, 4),
(’wLe’, ’50’, 4),
(’wCe’, ’50’, 4),
(’wRe’, ’50’, 4),
(’cLde’, ’-25’, 4),
(’cCde’, ’0’, 4),
(’cRde’, ’25’, 4),
(’wRde’, ’50’, 4),
(’wLde’, ’50’, 4),
(’wCde’, ’50’, 4),
(’cNGu’, ’-1.5’, 4),
(’cNPu’, ’-0.75’, 4),
(’cCEu’, ’0’, 4),
(’cPPu’, ’0.75’, 4),
(’cPGu’, ’1.5’, 4);

# --------------------------------------------------------
#
# Estructura de tabla para la tabla ‘permcont‘
#

CREATE TABLE ‘permcont‘ (
  ‘equipo‘ smallint(6) NOT NULL default ’0’,
  ‘user‘ int(11) NOT NULL default ’0’,
  PRIMARY KEY  (‘equipo‘,‘user‘)
) TYPE=MyISAM;

#
# Volcar la base de datos para la tabla ‘permcont‘
#

INSERT INTO ‘permcont‘ (‘equipo‘, ‘user‘) VALUES (1, 1);

# --------------------------------------------------------
#
# Estructura de tabla para la tabla ‘permmon‘
#

CREATE TABLE ‘permmon‘ (
  ‘equipo‘ smallint(6) NOT NULL default ’0’,
  ‘user‘ int(11) NOT NULL default ’0’,
```

```
  PRIMARY KEY  (‘equipo‘,‘user‘)
) TYPE=MyISAM;

#
# Volcar la base de datos para la tabla ‘permmon‘
#


# --------------------------------------------------------
#
# Estructura de tabla para la tabla ‘usuario‘
#

CREATE TABLE ‘usuario‘ (
  ‘nombre‘ varchar(20) NOT NULL default ’’,
  ‘apellido1‘ varchar(20) NOT NULL default ’’,
  ‘apellido2‘ varchar(20) default NULL,
  ‘id‘ int(11) NOT NULL default ’0’,
  ‘login‘ varchar(10) NOT NULL default ’’,
  ‘pwd‘ varchar(32) NOT NULL default ’’,
  PRIMARY KEY  (‘id‘)
) TYPE=MyISAM;

#
#  NOTA: Al crear un usuario se debe crear una carpeta con el nombre del
#        login en el directorio c:/AppServ/www/users donde se almacenan
#        los ficheros de los experimentos remotos
#
```

Apéndice C

Código fuente de la DLL de acceso SQL desde LabVIEW

Ficheros que componen la librería:

- stcdll32.h : Definición de tipos, constantes y funciones.
- stcdll32.cpp : Código de las funciones que realizan el acceso SQL.
- stcdll32.def : Exportación de funciones.

```
/*********************************************************************
 * Fichero: stcdll32.h
 *
 * Definici{\'o}n de tipos, constantes y funciones
 *
 *********************************************************************/

#ifdef __cplusplus // si los ficheros fuente son .cpp
extern "C" {
#endif
#include <mysql.h>

//Variables globales, para no pasarlas entre las funciones
MYSQL mysql; //Gestiona la conexion MySQL
MYSQL *sock;
MYSQL_RES *query_results;  //Resultados de una consulta
MYSQL_ROW row;  // Fila de un resultado

enum {mysql_api=0, odbc=1}; //Tipos de bases de datos, aqui siempre mysql

unsigned short int db_type; //Tipo de la base de datos
#define LOCKED 0
#define UNLOCKED 1
#define QUERY_LEN 4096
int connect_lock = UNLOCKED;//Para no acceder sin inicializar

typedef int int32; //Definicionesde tip de LabVIEW
typedef char* CStr;
typedef char uChar;
typedef struct { //Cadenas en LabVIEW
    long    cnt;
    char    str[1];
    } Lstr, **LStrHandle;

typedef struct {  //Estructura con los datos de una conexion
    LStrHandle db;
    LStrHandle host;
    LStrHandle user;
    LStrHandle password;
    unsigned short int db_type;
    } DB_LOGIN;

typedef struct { //Array de 1D  de cadenas
    int32 dimSize;
    LStrHandle String[1];
    } A1D;
typedef A1D **A1DHdl;

typedef struct { //Array de 2D de cadenas
    int32 dimSizes[2];
    LStrHandle String[1];
    } A2D;
typedef A2D **A2DHdl;

   //Funciones exportadas
```

```
  long WINAPI sql_select(DB_LOGIN *db,CStr table,CStr condition,A1DHdl fields,
                         char *distinct, A2DHdl values, CStr debug);
  long WINAPI sql_update(DB_LOGIN *db,char table[],A2DHdl values,A1DHdl fields,
                         A1DHdl where, char debug[], long *num_rows);
  //Funciones no exportadas

  //anyade una fila a un array de 2D de LabVIEW
  void StringRowAdd(A2DHdl in_array, char *string[], int cols);

  //Abre una conexion con la bbdd
  int open_db(DB_LOGIN *login);

  //Funcione de LabVIEW para gesti{\'o}n de memoria
  long DSNewHandle(int32);
  void DSSetHandleSize(A2DHdl, int32);


#ifdef __cplusplus //si el compilador es C++
}
#endif
```

```
/*************************************************************
 * Fichero: stcdll32.cpp
 *
 * C{\'o}digo de las funciones que realizan el acceso SQL
 *
 *************************************************************/
#include <windows.h>
#include <stdio.h>

#include "stcdll32.h"
#include <math.h>

// Funci{\'o}n DllMain es llamada por windows no por el usuario

BOOL WINAPI DllMain(HANDLE hModule, DWORD dwReason, LPVOID lpReserved)
{

    switch(dwReason)
    {
    case DLL_PROCESS_ATTACH:

        break;
    case DLL_THREAD_ATTACH:

        break;
    case DLL_THREAD_DETACH:

        break;
    case DLL_PROCESS_DETACH:

        break;

    }

    return TRUE; //DLL_PROCESS_ATTACH satisfactorio

}

//Realiza consulta de tipo SELECT
long WINAPI sql_select(DB_LOGIN *db, CStr table, CStr condition, A1DHdl fields,
                       char *distinct, A2DHdl values, CStr debug)
{

  char sql_query[QUERY_LEN];
  int i, num_cols;
  int j=0;


  num_cols = (**fields).dimSize; //Numero de campos pedidos

  db_type = (*db).db_type;  //Tipo de la base de datos, siempre mysql
  if (open_db(db) == -1) {return (-1);}; //Abre la conexion.
                                         //Si hay problemas sale

  strcpy(sql_query, "");
```

```
  /*
   *  Construye la petici{\'o}n SELECT
   */
  if (!*distinct) {strcat(sql_query, "SELECT ");}
  else {strcat(sql_query, "SELECT DISTINCT ");}
  switch (num_cols) {
  case 0:  //Si no hay campos pedidos, se sale
    return (0);
  case 1:  //se incluye en la peticion el campo pedido
    strncat(sql_query,
            (**(**fields).String[0]).str, (**(**fields).String[0]).cnt);
    break;
  default: //se incluyen los caampos pedidos, separados por comas
    strncat(sql_query,
            (**(**fields).String[0]).str, (**(**fields).String[0]).cnt);
    for (i=1; i<num_cols ; i++) {
      strcat(sql_query, ", ");
      strncat(sql_query,
              (**(**fields).String[i]).str, (**(**fields).String[i]).cnt);}
    break;
  }
  strcat(sql_query, " FROM ");  //Clausula FROM, tabla que se consulta
  strcat(sql_query, table);
  if (strlen(condition) > 0) {  //Si hay elementos en el WHERE
    strcat(sql_query, " WHERE ");
    strcat(sql_query, condition);}

  switch (db_type) {//Segun el tipo de base de datos, solo mysql
  case mysql_api:
    if (mysql_query(&mysql, sql_query) != 0) { //Error en la consulta
      sprintf(debug,"%s\nError: %s", sql_query, mysql_error(&mysql));
      return (-2);}
    query_results = mysql_store_result(&mysql);
    //Se anyaden los resultados fila a fila a la tabla values
    while ((row = mysql_fetch_row(query_results))) {
      StringRowAdd(values, row, num_cols); j++;
    }
    mysql_free_result(query_results); //Se libera la estructura con resultados
    mysql_close(&mysql); //Cierra la conexion
    connect_lock = UNLOCKED; //Para no realizar consultas
    return ((long) j); //Devuelve el numero de filas involucradas
  default:
    break;
  }
  mysql_close(&mysql);  //Cierre en caso de problemas
  connect_lock = UNLOCKED;
  return (-2);
}

//Realiza consulta de tipo UPDATE
long WINAPI sql_update(DB_LOGIN *db, char table[],A2DHdl values,A1DHdl fields,
                       A1DHdl where, char debug[], long *num_rows)
{
  char sql_query[QUERY_LEN];
  int i, j, k, l;
```

```
*num_rows = 0;

db_type = (*db).db_type;
if (open_db(db) == -1) {return (-1);};//Abre la conexion. Sale si problemas
/*
 *  Construye la petici{\'o}n UPDATE
 */
k = (**values).dimSizes[0];  //Numero de actualizaciones
if (k != (**where).dimSize) { //Si el numero de actualizaciones
                              //distinto del de condiciones
  sprintf(debug,
          "Error en la llamada: wheres=%d, rows=%d", (**where).dimSize, k);
  return (-2);}
l = (**values).dimSizes[1];
if (l != (**fields).dimSize) { //Si el numero de campos es
                               //distinto del de columnas
  sprintf(debug,
          "Error en la llamada: columns=%d, values=%d",(**fields).dimSize,l);
  return (-2);}
for (i=0; i<k ; i++) {  //Monta la peticion UPDATE para cada actualizacion
  strcpy(sql_query, "UPDATE ");
  strcat(sql_query, table);
  strcat(sql_query, " SET ");
  //Primer campo a actualizar
  strncat(sql_query,
          (**(**fields).String[0]).str, (**(**fields).String[0]).cnt);
  strcat(sql_query, "='");
  //Valor de la actualizacion
  strncat(sql_query,
          (**(**values).String[i*l]).str, (**(**values).String[i*l]).cnt);
  strcat(sql_query, "'");
  //Resto de campos separados por comas
  for (j=1; j<((**fields).dimSize) ;j++) {
    strcat(sql_query, ", ");
    strncat(sql_query,
            (**(**fields).String[j]).str, (**(**fields).String[j]).cnt);
    strcat(sql_query, "='");
    strncat(sql_query,(**(**values).String[j+i*l]).str,
            (**(**values).String[j+i*l]).cnt);
    strcat(sql_query, "'");}
  //Condicion para la actualizacion
  strcat(sql_query, " WHERE ");
  strncat(sql_query, (**(**where).String[i]).str,
          (**(**where).String[i]).cnt);

  switch (db_type) {
  case mysql_api:
    if (mysql_query(&mysql, sql_query) != 0) {  //Realiza la consulta
      sprintf(debug,"%s\nError: %s", sql_query, mysql_error(&mysql));
      return (-2);}
    *num_rows += (long)mysql_affected_rows(&mysql);
    break;

  default:
```

```
      break;
    }
  }

  mysql_close(&mysql);
  connect_lock = UNLOCKED;
  return (0);
}

/* Abre una conexi{\'o}n con una base de datos   */

int open_db(DB_LOGIN *login)
{
  int i;
  char host[80], user[80], passwd[80], db[80];

  connect_lock = LOCKED;
  //Coloca el \0 al final de host
  host[(**(login->host)).cnt] = 0;
  //Copia la cadena LabVIEW login->host a host en C
  for (i=(**(login->host)).cnt; i>0; i--)
    {host[i-1] = (**(login->host)).str[i-1];}
  //Igual con user
  user[(**(login->user)).cnt] = 0;
  for (i=(**(login->user)).cnt; i>0; i--)
    {user[i-1] = (**(login->user)).str[i-1];}
  //Igual con passwd
  passwd[(**(login->password)).cnt] = 0;
  for (i=(**(login->password)).cnt; i>0; i--)
    {passwd[i-1] = (**(login->password)).str[i-1];}
  //Igual con db
  db[(**(login->db)).cnt] = 0;
  for (i=(**(login->db)).cnt; i>0; i--)
    {db[i-1] = (**(login->db)).str[i-1];}

  switch (db_type) {
  case mysql_api:
    mysql_init(&mysql);  //Inicializa la estructura
    if (!(sock= mysql_real_connect(&mysql, host, user, passwd, db,  0, 0, 0))) {
      //Conexion a la base de datos
      connect_lock = UNLOCKED;
      return(-1);}
    break;

  }
  return(0);
}

//Anyade una fila a un array 2D
void StringRowAdd(A2DHdl in_array, char *string[], int cols)
{
  int i, j, num_rows;

  A2D * str_array;
```

```
  i = (*in_array)->dimSizes[0]; //Filas existentes del array
  num_rows = i + 1; //Filas futuras
   //Modifica el tamnyo de la zona e memoria, para una fila mas
  DSSetHandleSize(in_array,sizeof(int32)*2 + num_rows*cols*sizeof(LStrHandle));

  //Para tranbajar con puntero en lugar de handle
  str_array = *in_array;

  str_array->dimSizes[0] = num_rows;
  str_array->dimSizes[1] = cols;

  //Se copia la fila campo a campo
  for(j=0; j < cols; j++)
    { //Si es null el campo , se copia una cadena vacia
      if (string[j] == NULL) {
        //Se reserva un nuevo espacio de memoria para la cadena
        str_array->String[i*cols +j] = (LStrHandle)DSNewHandle(sizeof(int32)
                                        +(strlen("")+1)*sizeof(uChar));
        //Se actualiza el campo cnt de la cadena
        (*(str_array->String[i*cols +j]))->cnt = strlen("");
        //Se copian los caracteres
        strcpy((*(str_array->String[i*cols +j]))->str,"");}
      else {
        //Igual, copiando la cadena.
        str_array->String[i*cols +j] = (LStrHandle)DSNewHandle(sizeof(int32)
                                      + (strlen(string[j])+1)*sizeof(uChar));
        (*(str_array->String[i*cols +j]))->cnt = strlen(string[j]);
        strcpy((*(str_array->String[i*cols +j]))->str,string[j]);}
    }
}
```

```
/*************************************************************************
 * Fichero: stcdll32.def
 *
 * Exportaci{\'o}n de funciones.
 *
 *************************************************************************/
LIBRARY ACCESO_SQL

DESCRIPTION 'DLL para acceso SQL desde LABVIEW'

EXPORTS

    sql_select                  @1
    sql_update                  @2
```

Apéndice D

Archivos Java del applet de monitorización

Archivos:

- LectorXML.java : Interpreta el documento XML que describe el equipo.
- Monitor.java : Periódicamente solicita el documento XML y representa el estado del sistema y su evolución.

```
/*****************************************************
 * Fichero: LectorXML.java
 *
 * Interpreta el documento XML que describe el equipo
 *
 *****************************************************/

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import java.util.*;

public class LectorXML extends DefaultHandler
{
private String c_n;
private int  c_np;
private String c_des;
private Vector par;
private Vector val;
private Vector sig;
private Vector vals;
private String eq_des;
private String eq;
private String desact;
private int elem; //1:equipo 2:control 3:parametro 4:signal

public void lee(String dir)
  {
    par=new Vector();
    val=new Vector();
    sig=new Vector();
    vals=new Vector();
    desact="no";

    SAXParserFactory factory = SAXParserFactory.newInstance();
    try
      {
        SAXParser saxParser = factory.newSAXParser();
        saxParser.parse(dir,this);
      }catch(Exception e){}
  }
//Inicio de un elemento XML
public void startElement(String uri,String localName,String qName,
                         Attributes attributes) throws SAXException
  {
    String elemento=localName;

    if(elemento.equals("")) elemento=qName; //Si no hay espacio de nombrado.
    if(elemento.equals("EQUIPO")){
      elem=1;

      for(int i=0;i<attributes.getLength();i++)
        {
```

```
          String atrib=attributes.getLocalName(i);
          if(atrib.equals("")) atrib=attributes.getQName(i);

          if(atrib.equals("NOMBRE")) eq=attributes.getValue(i);

          else if(atrib.equals("DETENIDO")) desact=attributes.getValue(i);
        }
    }
    else if(elemento.equals("CONTROL")){
      elem=2;
      for(int i=0;i<attributes.getLength();i++)
        {
          String atrib=attributes.getLocalName(i);
          if(atrib.equals(""))
                atrib=attributes.getQName(i);

          if(atrib.equals("NOMBRE"))
                c_n=new String(attributes.getValue(i));

          else if(atrib.equals("N_PAR"))
                c_np=(new Integer(attributes.getValue(i))).intValue();
        }
    }
    else if(elemento.equals("PARAMETRO")){
      elem=3;
      par.add(new String(attributes.getValue(0)));
    }

    else if(elemento.equals("SIGNAL")){
      elem=4;
      sig.add(new String(attributes.getValue(0)));
    }
  }
public void endElement(String naespaceURI,String sName,String qName)
throws SAXException
{
  elem=0;
}

public void characters (char buf[],int offset, int len) throws SAXException
  {
    String cad=new String(buf,offset,len);

    switch(elem){
    case 1:
      eq_des=cad;
      break;
    case 2:
      c_des=cad;
      break;
    case 3:
      val.add(new Float(cad));
      break;
    case 4:
      vals.add(new Float(cad));
```

```
      break;
    }
  }
public String equipo()
  {
    return eq;
  }
public String descEquipo()
  {
    if(eq_des!=null)
      return eq_des;
    else
      return "Sin Descripcion";
  }

public boolean activado()
  {
    boolean act=true;
    if (desact.equals("si"))
      act= false;

    return act;
  }
public String nombreControl()
  {
    return c_n;
  }
public String descControl()
  {
    if(c_des!=null)
      return c_des;
    else
      return "Sin Descripcion";
  }
public int nParamControl()
  {
    return c_np;
  }
public Vector parametros()
  {
    return par;
  }
public Vector parValores()
  {
    return val;
  }
public Vector signals()
  {
    return sig;
  }
public Vector sigValores()
  {
    return vals;
  }
}
```

```
/*************************************************************************
 * Fichero: Montor.java
 *
 * Periodicamente solicita el documento XML y representa el estado
 * del sistema y su evolucion.
 *
 *************************************************************************/

import java.awt.*;
import java.applet.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.*;

/*
    <applet code="Monitor" width=500 height=500>
    <param name="control" value="si">
    </applet>
*/

public class Monitor extends Applet implements Runnable{
  int time[];
  int altura[];
  int referencia[];

  int incMillis;
  long cont;

  float alt;//Ultimo valor de la altura
  float ref;//Ultimo valor de la referencia.

  Label eq_n;

  Label c_n;

  Label p_n[];
  Label p_v[];

  Label s_n[];
  Label s_v[];

  Checkbox comp,desp;
  CheckboxGroup cbg;

  Vector timeV;
  Vector alturaV;
  Vector referenciaV;

  Thread t=null;

  int len;

  long timeini;
```

```
  double xini;//Valor de los ejes inicial y final.
  double xfin;
  double yini;
  double yfin;

  Dimension d;

  String cad;

  boolean control;

  TextField controlT;
  Checkbox enviar;

  TextField incremento;

  boolean borroso;
  float cLe,wLe,cCe,wCe,cRe,wRe,cLde,wLde,cCde,wCde;
  float cRde,wRde,cNGu,cNPu,cCEu,cPPu,cPGu;

public void init(){

  timeV=new Vector();
  alturaV=new Vector();
  referenciaV=new Vector();

  time=new int[1];//Valores iniciales, para llamar a paint la primera vez
  time[0]=0;
  altura=new int[1];
  altura[0]=0;
  referencia=new int[1];
  referencia[0]=0;
  alt=0;
  ref=0;

  xini=xfin=0;
  yini=0;
  yfin=80;

  incMillis=200;

  eq_n=new Label("",Label.CENTER);
  c_n=new Label("");

  add(eq_n);
  add(c_n);

  d =getSize();
  cbg=new CheckboxGroup();
  comp=new Checkbox("Comprimir",cbg,false);
  desp=new Checkbox("Desplazar",cbg,true);
  add(comp);
  add(desp);

  borroso=false;
```

```
  incremento=new TextField("200",10);
  add(incremento);

  if(getParameter("control").equals("si"))
    {
      control=true;
      controlT=new TextField("0.0",20);
      enviar=new Checkbox("Modificar Referencia");

      add(controlT);
      add(enviar);
    }
  else
    {
      control=false;
      controlT=null;
      enviar=null;
    }

  t=new Thread(this);
  t.start();
}

public void run() {
  long sigsondeo=0;

  timeini=System.currentTimeMillis();
  while(true){
    try{
      cont++;

      if(System.currentTimeMillis()<sigsondeo){
        Thread.sleep(sigsondeo-System.currentTimeMillis());
      }
      incMillis=(new Integer(incremento.getText())).intValue();

      sigsondeo=System.currentTimeMillis()+incMillis;

      LectorXML lector=new LectorXML();
      if(enviar!=null && enviar.getState())
        {
          lector.lee(getCodeBase()+"/levineu.php?ref="+controlT.getText());
          enviar.setState(false);
        }
      else
        lector.lee(getCodeBase()+"/levineu.php");

      if(lector.equipo()!=null){

        String equipo=lector.equipo()+" : "+lector.descEquipo();
        eq_n.setText(equipo.toUpperCase());

        c_n.setText("CONTROL: "+lector.nombreControl());

        if ((lector.nombreControl()).equals("Borroso")) borroso=true;
```

```
else borroso=false;

Vector par=lector.parametros();
Vector pval=lector.parValores();
if(par!=null&&pval!=null)
  {
    if (borroso)
      {
        if(p_n != null)/* Quitar las etiquetas que hubiese*/
          for(int i=0;i<p_n.length;i++)
            {
              remove(p_n[i]);
              remove(p_v[i]);
            }
        for(int i=0;i<par.size();i++)
          {
            String parb=(String)par.elementAt(i);
            float valb=((Float)pval.elementAt(i)).floatValue();

            if(parb.equals("cLe")) cLe=valb;
            else if(parb.equals("wLe")) wLe=valb;
            else if(parb.equals("cCe")) cCe=valb;
            else if(parb.equals("wCe")) wCe=valb;
            else if(parb.equals("cRe")) cRe=valb;
            else if(parb.equals("wRe")) wRe=valb;

            else if(parb.equals("cLde")) cLde=valb;
            else if(parb.equals("wLde")) wLde=valb;
            else if(parb.equals("cCde")) cCde=valb;
            else if(parb.equals("wCde")) wCde=valb;
            else if(parb.equals("cRde")) cRde=valb;
            else if(parb.equals("wRde")) wRde=valb;

            else if(parb.equals("cNGu")) cNGu=valb;
            else if(parb.equals("cNPu")) cNPu=valb;
            else if(parb.equals("cCEu")) cCEu=valb;
            else if(parb.equals("cPPu")) cPPu=valb;
            else if(parb.equals("cPGu")) cPGu=valb;


          }
      }
    /*Caso en que no hubiese etiquetas con los par{\'a}metros o
      el n{\'u}mero fuese distinto.*/

    else if(p_n==null || (p_n!=null && p_n.length!=par.size()))
      {
        if(p_n != null)/* Quitar las etiquetas que hubiese*/
          for(int i=0;i<p_n.length;i++)
            {
              remove(p_n[i]);
              remove(p_v[i]);
            }

        p_n=new Label[par.size()];
```

```
          p_v=new Label[pval.size()];

          for(int i=0;i<par.size();i++)
            {

              p_n[i]=new Label((String)par.elementAt(i),Label.RIGHT);

              add(p_n[i]);
              //p_n[i].setBounds(50,160+20*i,d.width/4-60,15);

              p_v[i]=new Label("= "+(Float)pval.elementAt(i),Label.LEFT);
              add(p_v[i]);
              //p_v[i].setBounds(d.width/4,160+20*i,d.width/4-50,15);
            }
        }
      else  /*Simplemente actualizar los valores de las etiquetas.
              Asi se evita parpadeo de los parametros*/
        {
          for(int i=0;i<par.size();i++)
            {
              p_n[i].setText((String)par.elementAt(i));

              p_v[i].setText("= "+(Float)pval.elementAt(i));
            }
        }
    }
  if(lector.activado())
    {

      Vector sig=lector.signals();
      Vector sval=lector.sigValores();

      for (int i=0;i<sig.size();i++)
        {
          if(((String)sig.elementAt(i)).equals("altura"))
            {
              alt=((Float)sval.elementAt(i)).floatValue();
              alturaV.add(new Integer((int)(alt*10)));
            }
          if(((String)sig.elementAt(i)).equals("ref"))
            {
              ref=((Float)sval.elementAt(i)).floatValue();
              referenciaV.add(new Integer((int)(ref*10)));
              if(referenciaV.size()==1 && control)
                controlT.setText(""+ref);
            }
        }

      timeV.add(new Integer
                ((int)((System.currentTimeMillis()-timeini)/100)));

      if(desp.getState()) despl();
      else comp();
    }
  else
```

```
          {
            limpia();
          }

      }
      else Thread.sleep(incMillis);
    }catch(Exception e){cad+=e;}
    repaint();
  }
}

public void paint(Graphics g){
  d =getSize();
  eq_n.setBounds(100,20,d.width-200,15);
  c_n.setBounds(20,40,d.width-40,30);

  int ancho=(d.width-80)/8;
  int anchob=(d.width-60)/3;

  /* En caso de ser el controlador Borroso se dibujan las funciones de
     pertenencia en lugar de mostrar los valores de los par{\'a}metros, porque
     as{\'\i} se ve m{\'a}s claro
  */
  if (borroso)
    {
      int uno=40+d.height/6;
      int cero=d.height/2-40;

      g.drawString("-50              Error",20,85);
      g.drawString("50",5+anchob,85);
      g.drawRect(20,90,anchob,d.height/2-130);

      g.drawString("-50         Der. Error",30+anchob,85);
      g.drawString("50",15+2*anchob,85);
      g.drawRect(30+anchob,90,anchob,d.height/2-130);

      g.drawString("-0.2         Actuaci{\'o}n",40+2*anchob,85);
      g.drawString("0.2",25+3*anchob,85);
      g.drawRect(40+2*anchob,90,anchob,d.height/2-130);

      g.setColor(Color.red);
      g.drawLine(20,uno,(int)(20+(cLe+50)*anchob/100),uno);
      g.drawLine((int)(20+(cLe+50)*anchob/100),uno,
                 (int)(20+(cLe+wLe/2+50)*anchob/100),cero);
      g.drawLine((int)(20+(cCe-wCe/2+50)*anchob/100),cero,
                 (int)(20+(cCe+50)*anchob/100),uno);
      g.drawLine((int)(20+(cCe+50)*anchob/100),uno,
                 (int)(20+(cCe+wCe/2+50)*anchob/100),cero);
      g.drawLine((int)(20+(cRe-wRe/2+50)*anchob/100),cero,
                 (int)(20+(cRe+50)*anchob/100),uno);
      g.drawLine((int)(20+(cRe+50)*anchob/100),uno,anchob+20,uno);

      g.drawLine(30+anchob,uno,(int)(30+anchob+(cLde+50)*anchob/100),uno);
      g.drawLine((int)(30+anchob+(cLde+50)*anchob/100),uno,
                 (int)(30+anchob+(cLde+wLde/2+50)*anchob/100),cero);
```

```
    g.drawLine((int)(30+anchob+(cCde-wCde/2+50)*anchob/100),
               cero,(int)(30+anchob+(cCde+50)*anchob/100),uno);
    g.drawLine((int)(30+anchob+(cCde+50)*anchob/100),uno,
               (int)(30+anchob+(cCde+wCde/2+50)*anchob/100),cero);
    g.drawLine((int)(30+anchob+(cRde-wRde/2+50)*anchob/100),cero,
               (int)(30+anchob+(cRde+50)*anchob/100),uno);
    g.drawLine((int)(30+anchob+(cRde+50)*anchob/100),uno,2*anchob+30,uno);


    g.drawLine((int)(40+2*anchob+(cNGu+0.2)*anchob/0.4),cero,
               (int)(40+2*anchob+(cNGu+0.2)*anchob/0.4),uno);
    g.drawLine((int)(40+2*anchob+(cNPu+0.2)*anchob/0.4),cero,
               (int)(40+2*anchob+(cNPu+0.2)*anchob/0.4),uno);
    g.drawLine((int)(40+2*anchob+(cCEu+0.2)*anchob/0.4),cero,
               (int)(40+2*anchob+(cCEu+0.2)*anchob/0.4),uno);
    g.drawLine((int)(40+2*anchob+(cPPu+0.2)*anchob/0.4),cero,
               (int)(40+2*anchob+(cPPu+0.2)*anchob/0.4),uno);
    g.drawLine((int)(40+2*anchob+(cPGu+0.2)*anchob/0.4),cero,
               (int)(40+2*anchob+(cPGu+0.2)*anchob/0.4),uno);

    g.setColor(Color.black);
  }
else  if (p_n!=null) for(int i=0;i<p_n.length;i++)
  {
    p_n[i].setBounds(25+(i%4)*2*ancho+10*(i%4),80+20*(i/4),ancho,15);

    p_v[i].setBounds(25+ancho+(i%4)*2*ancho+10*(i%4),80+20*(i/4),ancho,15);
  }
if (control)
  {
    controlT.setBounds(150,d.height/2-25,100,20);
    enviar.setLocation(20,d.height/2-30);
  }
comp.setLocation(d.width-180,d.height/2-30);
desp.setLocation(d.width-100,d.height/2-30);
//La mitad de abajo  para la informacion grafica.
g.drawRect(20,d.height/2,d.width-60,d.height/2-50);

g.drawString(""+xini,20,d.height-35);

if(len>1 && time[len-1]<d.width-50)
  g.drawString(""+xfin,time[len-1],d.height-35);

else
  g.drawString(""+xfin,d.width-50,d.height-35);

g.drawString(""+yini,d.width-35,d.height-50);
g.drawString(""+yfin,d.width-35,d.height/2+10);

g.drawString("Refresco(ms)",d.width-180,d.height-15);
incremento.setBounds(d.width-100,d.height-30,50,20);

g.setColor(Color.red);
g.drawLine(30,d.height-20,50,d.height-20);
```

```
  g.drawString("Altura "+alt,55,d.height-15);

  for (int i=0;i<len-1;i++)
    g.drawLine(time[i],altura[i],time[i+1],altura[i+1]);

  g.setColor(Color.blue);
  g.drawLine(150,d.height-20,170,d.height-20);
  g.drawString("Referencia "+ref,175,d.height-15);
  for (int i=0;i<len-1;i++)
    g.drawLine(time[i],referencia[i],time[i+1],referencia[i+1]);

}

  /* Grafica desplazandose en el recuadro */
private void despl(){
  int inicial,endtime;
  inicial=((Integer)timeV.elementAt(0)).intValue();
  endtime=((Integer)timeV.elementAt(timeV.size()-1)).intValue();

  while(endtime-inicial>=d.width-60){

    timeV.removeElementAt(0);
    alturaV.removeElementAt(0);
    referenciaV.removeElementAt(0);

    inicial=((Integer)timeV.elementAt(0)).intValue();
  }

  len=alturaV.size();

  time=new int[len];
  altura=new int[len];
  referencia=new int[len];

  for (int k=0;k<len;k++)
    {
      int valor=((Integer)alturaV.elementAt(k)).intValue();
      altura[k]=(int)((800-valor)*(d.height/2-50)/800+d.height/2);
      //800 es el maximo de valor

      valor=((Integer)referenciaV.elementAt(k)).intValue();
      referencia[k]=(int)((800-valor)*(d.height/2-50)/800+d.height/2);

      time[k]=((Integer)timeV.elementAt(k)).intValue()-inicial+20;
    }
  xini=inicial/10.0;
  xfin=endtime/10.0;
}

  /* La grafica se ajusta para el ocupar con el cuadro completo*/

private void comp(){
  int endtime=((Integer)timeV.elementAt(timeV.size()-1)).intValue();
  int inicial=((Integer)timeV.elementAt(0)).intValue();
```

```
  len=alturaV.size();

  time=new int[len];
  altura=new int[len];
  referencia=new int[len];

  for (int k=0;k<len;k++)
    {
      int valor=((Integer)alturaV.elementAt(k)).intValue();
      altura[k]=(int)((800-valor)*(d.height/2-50)/800+d.height/2);

      valor=((Integer)referenciaV.elementAt(k)).intValue();
      referencia[k]=(int)((800-valor)*(d.height/2-50)/800+d.height/2);

      time[k]=(((Integer)timeV.elementAt(k)).intValue()-inicial)*
        (d.width-60)/(endtime-inicial)+20;
    }
  xini=inicial/10.0;;
  xfin=endtime/10.0;
}

  /* Borra los datos existentes en los vectores */

public void limpia(){
  time=new int[1];
  time[0]=0;

  altura=new int[1];
  altura[0]=0;

  referencia=new int[1];
  referencia[0]=0;


  alturaV.clear();
  timeV.clear();
  referenciaV.clear();

  len=0;
  timeini=System.currentTimeMillis();

  xini=xfin=0;
  yini=0;
  yfin=80;
}
}
```

Apéndice E

Código fuente de los archivos que componen la página de acceso remoto

Ficheros de la interfaz remota:

- labrem.html : Página principal, que la divide en dos marcos: la cabecera superior y la portada.
- cabecera.php : Cabecera de la página que inicialmente sirve para introducir el nombre y clave, y una vez identificado presenta el menú de navegación.
- portada.html : Página de presentación.
- sesiones.php : Controla la sesión del usuario, no permitiendo acceder a zonas que se necesite identificación sin estarlo.
- bdd.php : Conecta a la base de datos.
- login.php : Valida el par nombre de usuario clave.
- remoto.php : Acceso al control remoto. Divide la página para mostrar los distintos elementos del control remoto.
- activex.php : Muestra las imágenes de las cámaras según la que se haya seleccionado.
- app.php : Incluye el applet de monitorización. Según el usuario tenga el control o no incluye la posibilidad de modificar la referencia.
- inicio.php : Formulario inicial dentro de la interfaz de control. Permite seleccionar entre los controladores existentes o enviar una DLL con un controlador programado por el usuario.
- carga.php : Detiene la aplicación de control y copia la DLL en el lugar adecuado.
- parametro.php : Formulario para seleccionar los parámetros del control seleccionado y el tiempo de muestreo.

- control.php : Permite modificar los parámetros del control en línea y detener el experimento realizado.
- fin.php : Una vez finalizado el ensayo, sirve para introducir el nombre del archivo en que guardar los datos. Muestra los existentes en la cuenta de usuario, puesto que si se introduce el nombre de uno de los que hay se sobreescribirá.
- fichero.php : Copia el fichero con los resultados en la cuenta del usuario y vuelve al inicio.
- monitoriza.php: Acceso a la monitorización. Muestra las cámaras y el applet de monitorización.
- ficheros.php : Muestra los archivos existentes en la cuenta del usuario.
- desc.php : Cierra la sesión.
- levineu.php : Genera el documento XML que describe el equipo.

```
<!-- Fichero labrem.html    -->

<HTML>
<HEAD>
<TITLE>Laboratorio Remoto</TITLE>
</HEAD>

<FRAMESET ROWS="80,*">
    <FRAME NAME="indice" SRC="cabecera.php" FRAMEBORDER=0 SCROLLING="NO">
    <FRAME NAME="contenido" FRAMEBORDER=0 SRC="portada.html">
</FRAMESET>
</HTML>
```

```
<?php
session_start();
// Fichero cabecera.php
?>
<HTML>
<HEAD>
<TITLE>&Iacute;ndice</TITLE>
</HEAD>
<BODY bgcolor="#FFFFE3" text="#000000" link="#0000FF"
 vlink="#800080" alink="#FF0000" >
<MAP NAME="indice">
  <AREA SHAPE=RECT COORDS="14,14,178,70" HREF="remoto.php" ALT="Control"
   TARGET="contenido" >
  <AREA SHAPE=RECT COORDS="190,14,353,70" HREF="monitoriza.php" ALT="Monitor"
   TARGET="contenido">
  <AREA SHAPE=RECT COORDS="365,14,530,70" HREF="ficheros.php" ALT="Ficheros"
   TARGET="contenido">
  <AREA SHAPE=RECT COORDS="540,14,706,90" HREF="desc.php" ALT="Desconectar"
   TARGET="contenido">
  <AREA SHAPE=DEFAULT NOHREF ALT="Nada">
</MAP>
<?php
if (isset($_SESSION['usuario']))
 {
 ?>
<TABLE WIDTH="100%">
<TR>
 <TD>Usuario: <?php echo $_SESSION['usuario'];?> </TD>
 <TD><IMG SRC="cabecera.jpg" WIDTH=720 HEIGHT=90 BORDER=0 USEMAP="#indice">
 </TD>
</TR>
</TABLE>
<?php
 }
 else
 {
?>
<TABLE  ALIGN=CENTER>
<FORM method="post" action="login.php">
<TR>
 <TD>Usuario: <INPUT TYPE="text" NAME="usu" SIZE=10>
     Clave : <INPUT TYPE="password" NAME="pwd" SIZE=10>
    <input type="submit" value="Acceder">
 </TD>
</TR>
</TABLE>
</FORM>
<?php
}
?>
</BODY>
</HTML>
```

```
<!-- Fichero portada.html -->
<HTML>
<BODY bgcolor="#FFFFE3" BACKGROUND="dibujo.jpg">
<CENTER>
 <br><br>
 <b><FONT FACE="Comic Sans MS, cursive" COLOR=BLUE SIZE=+3>
       SISTEMA DE LEVITACI{\'O}N NEUM{\'A}TICA<br>
       CON <br>
       CONTROL DE POSICI{\'O}N</FONT>
 </b>
 <br><br><br><br>
 <FONT SIZE=+3>Departamento de Ingenier{\'\i}a de Sistemas y Autom{\'a}tica</FONT>
 <br>
 <FONT SIZE=+3>Universidad de Sevilla</FONT>
 <br><br><br><FONT SIZE=+1>
 Fernando-Borja Bre{\~n}a Lajas<br>
 Jos{\'e} Julio Caparr{\'o}s Jim{\'e}nez<br>
 Jos{\'e} Antonio P{\'e}rez Garc{\'\i}a de Castro<br>
 <br>Tutor: Manuel G. Ortega
</FONT>
</BODY>
</HTML>
```

```
<?php
session_start();
// Fichero sesiones.php

if (isset($_SESSION['IP']) ) // si esta registrado
{
 if($_SESSION['IP']!=$_SERVER['REMOTE_ADDR'])
    error ("Falsa IP");

}
else {
    error("No registrado");
}

function error($tipo_error)
{
    echo "Error: ".$tipo_error;
    $_SESSION = array();
    session_destroy();
    die();
}
?>
```

```
<?php
// Fichero bdd.php
if (!($link=mysql_connect("localhost","root","")))
   {
      echo "Error conectando a la base de datos.";
      exit();
   }
   if (!mysql_select_db("cremoto",$link))
   {
      echo "Error seleccionando la base de datos.";
      exit();
   }
?>
```

```
<?php
//Fichero login.php
include("bdd.php");

$usuario = valida($_POST['usu']);
$clave = valida($_POST['pwd']);
$md5clave=md5($clave);
$sql="SELECT * FROM usuario WHERE login='$usuario' AND pwd='".$md5clave."'";

$resultado = mysql_query($sql, $link);

// si el nombre esta
if (mysql_num_rows($resultado))
{
    // creamos la sesion
    session_start();
    $ip = $_SERVER['REMOTE_ADDR'];

    $_SESSION['conectado']=1;
    $_SESSION['usuario'] = $usuario;
    $_SESSION['IP'] = $ip; // graba la IP dentro de la sesion

    mysql_close($link); // cerramos la sesion xq la volvera a abrir en privado

    header("Location: cabecera.php");
}
else
{
    echo "No autorizado";
    mysql_close($link);
    echo '<a href="cabecera.php">Volver</a>';
}


function valida($mensaje) {
    // quita caracteres comprometedores

    $mensaje = str_replace("\'","&#39;",$mensaje);
    $mensaje = str_replace('\"',"&quot;",$mensaje);
    $mensaje = str_replace("\\\\","&#92;",$mensaje);

    if (!ereg('^([0-9a-zA-Z@*#{\~n}{\~N}]{3,10})', $mensaje))
    {
         echo "Error, solo se permiten alfanum&eacute;ricos.";
         die();
    }
return $mensaje;
}
?>
```

```
<?php
include ("sesiones.php");
include ("bdd.php");
//Fichero remoto.php

$sql="SELECT p.equipo FROM permcont p,usuario u,equipo e WHERE p.equipo=1";
$sql.=" AND u.login='".$_SESSION['usuario']."' AND u.id=p.user";
$resultado=mysql_query($sql,$link);

if(mysql_num_rows($resultado))
{
  $sql= "SELECT estado FROM equipo WHERE id=1";
  $resultado=mysql_query($sql,$link);

  $reg=mysql_fetch_array($resultado);

  if ($reg['estado']!=2)
  {
    if ($reg['estado']==0)
    {
        $_SESSION['levineu']=2; //Se le concede el control

        $sql="SELECT id from usuario WHERE login='".$_SESSION['usuario']."'";

        $resultado=mysql_query($sql,$link);
        $reg=mysql_fetch_array($resultado);

        $sql="UPDATE equipo set id_control=".$reg['id']." WHERE id=1";

        mysql_query($sql,$link);

        $sql="UPDATE equipo set estado=1 WHERE id=1";
        mysql_query($sql,$link);
        $control="si";

    }
    else if(isset($_SESSION['levineu']))
    {
        if ($_SESSION['levineu']==2) $control="si";
        else  $control="no";

    }
    else{
     $control="no";

    }

    if($control=="no")
    //Mira a ver si hace mucho que no se controla (Sesion no cerrada)
    {

        $sql="SELECT estado FROM equipo WHERE id=1 AND CURRENT_TIMESTAMP ";
        $sql.="> DATE_ADD( ult_acceso, INTERVAL 10 MINUTE )";
            $res=mysql_query($sql,$link);
```

```
            if(mysql_num_rows($res))
            {
                $_SESSION['levineu']=2; //Se le concede el control

                $sql="SELECT id from usuario WHERE login='";
                $sql.=$_SESSION['usuario']."'";

                $resultado=mysql_query($sql,$link);
                $reg=mysql_fetch_array($resultado);

                $sql="UPDATE equipo set id_control=".$reg['id']." WHERE id=1";

                mysql_query($sql,$link);

                $sql="UPDATE equipo set estado=1 WHERE id=1";
                mysql_query($sql,$link);
                $control="si";
            }

            else  $control="no";

    }
    if ($control=="si")
    {
      $_SESSION['cam']=1;
?>
<HTML>
<HEAD>
  <TITLE>Control Remoto</TITLE>
</HEAD>
<FRAMESET COLS="425,*">
  <FRAMESET ROWS="280,*">
    <FRAME NAME="cam" SRC="activex.php" FRAMEBORDER=0 >
    <FRAME NAME="control" SRC="inicio.php" FRAMEBORDER=0>
  </FRAMESET>
  <FRAME NAME="monitor" SRC="app.php" FRAMEBORDER=0>
  <NOFRAMES>
    <P>Lo siento, pero s{\'o}lo podr{\'a}s ver esta p{\'a}gina
    si tu navegador tiene la capacidad de visualizar
    marcos.</P>
  </NOFRAMES>
</FRAMESET>
</HTML>
<?php
    }
}
else
{

 $control="no";
}

if ($control=="no")
    {
```

```
?>
<TABLE ALIGN=CENTER WIDTH="70%">
<TR ALIGN=CENTER>
  <TD> El equipo Levineu no est{\'a} libre para ser controlado.</TD>
</TR>
<TR ALIGN=CENTER>
  <TD><A href="monitoriza.php">Monitorizar</A><A href="portada.html">Inicio</A>
  </TD>
</TR>
</TABLE>
<?php
    }
}
else
{
?>
<TABLE ALIGN=CENTER WIDTH="70%">
<TR ALIGN=CENTER><TD>Sin permisos para controlar el equipo Levineu.</TD></TR>
<TR ALIGN=CENTER><TD><A href="portada.html">Inicio</A></TD></TR>
</TABLE>
<?php
}
?>
```

```
<?php
include ("sesiones.php");
// Fichero activex.php

if ($_SESSION['levineu']==1 ||$_SESSION['levineu']==2)
{
?>
<HTML>
<HEAD>

<TITLE>
Levineu
</TITLE>

</HEAD>
<?php
if($_SESSION['cam']==1)
{
$_SESSION['cam']=2;
?>
<BODY bgcolor="#FFFFE3" text="#000000" link="#0000FF" vlink="#800080"
 alink="#FF0000" >
<OBJECT Name="Pl1" WIDTH="350" HEIGHT="240"
classid="CLSID:66D393D5-4D80-497C-9F4F-F3839E090202"
CODEBASE="http://www.pysoft.com/Downloads/WebCamPlayerOCX.cab#version=6,1,0,0"
standby="Loading PY Software player for AWLive files..."
type="application/x-oleobject">
<PARAM NAME="ImagePath" VALUE="capture3.jpg">
<PARAM NAME="ShowTopBar" VALUE="0">
<PARAM NAME="ShowControls" VALUE="0">
<PARAM NAME="ImagePort" VALUE="8084">
<PARAM NAME="BackColor" VALUE="#FFFFE3">
</OBJECT>
<script language="JavaScript">
document.Pl1.DocumentURL=document.URL;</script>

<?php
}
else
{
$_SESSION['cam']=1;
?>

<OBJECT Name="Pl2" WIDTH="350" HEIGHT="240"
classid="CLSID:66D393D5-4D80-497C-9F4F-F3839E090202"
CODEBASE="http://www.pysoft.com/Downloads/WebCamPlayerOCX.cab#version=6,1,0,0"
standby="Loading PY Software player for AWLive files..."
type="application/x-oleobject">
<PARAM NAME="ImagePath" VALUE="capture2.jpg">
<PARAM NAME="ShowTopBar" VALUE="0">
<PARAM NAME="ShowControls" VALUE="0">
<PARAM NAME="ImagePort" VALUE="8082">
<PARAM NAME="BackColor" VALUE="#FFFFE3">
</OBJECT>
```

```
<script language="JavaScript">
document.Pl2.DocumentURL=document.URL;
</script>
<?php
}
}

else
  error("No tienes permisos para monitorizar el equipo Levineu");

?>
</BODY>
</HTML>
```

```
<?php
include ("sesiones.php");
// Fichero app.php

if (isset($_SESSION['levineu']))
{
?>
<HTML>
<HEAD>
<TITLE>Monitor</TITLE>
</HEAD>
<BODY bgcolor="#FFFFE3" text="#000000" link="#0000FF" vlink="#800080"
 alink="#FF0000" >
<applet code="Monitor" width=550 height=440>
<?php
    if ($_SESSION['levineu']==2)
    {
?>
<param name="control" value="si">
<?php
    }
    else
    {
?>
<param name="control" value="no">
<?php
    }
?>

</applet>
</BODY>
<?php
}
else error("Sin permiso para monitorizar el sistema Levineu");
?>
```

```
<?php
include ("sesiones.php");
include ("bdd.php");
// Fichero inicio.php

if (isset($_SESSION['levineu']))
{
 if ($_SESSION['levineu']==2)
{


$sql="SELECT nombre,id from control where id_eq=1";

$res=mysql_query($sql,$link);

?>
<HTML>
<HEAD>
<TITLE>Inicio Control</TITLE>
</HEAD>

<BODY bgcolor="#FFFFE3" text="#000000" link="#0000FF" vlink="#800080"
 alink="#FF0000" >
<form action="parametro.php" method="post">
Selecciona el control deseado: <SELECT NAME="control" SIZE="1">
<?php while($reg=mysql_fetch_array($res))
{
?>
    <OPTION VALUE="<?php echo $reg['id'];?>"><?php echo $reg['nombre'];?>
    </OPTION>
<?php
}
?>
</SELECT>
<input type="submit" value="Siguiente">
</form>
<form action="carga.php" method="post" enctype="multipart/form-data">
<input type="file" name="fic">
<input type="submit" value="Enviar DLL">
</form>

</BODY>

</HTML>

<?php
}
else error ("Sin permisos para controlar el sistema Levineu");
}
else error ("Sin permisos para monitorizar el sistema Levineu");
?>
```

```
<?php
// Fichero carga.php

if (!($link=mysql_connect("localhost","root","")))
   {
      echo "Error conectando a la base de datos.";
      exit();
   }
   if (!mysql_select_db("cremoto",$link))
   {
      echo "Error seleccionando la base de datos.";
      exit();
   }

$dir="c:\\temporal\\nuevo.dll";

$fic=$_FILES['fic']['tmp_name'];

copy($fic,$dir);


$sql="UPDATE comun set valor=1 WHERE nombre='detener'";

mysql_query($sql,$link);

header("Location: inicio.php");

?>
```

```
<?php
include ("sesiones.php");
include ("bdd.php");
// Fichero parametro.php
if (isset($_SESSION['levineu']))
{
 if ($_SESSION['levineu']==2)
 {
  $sql="UPDATE comun set valor=0 where nombre='detener' and id_eq=1";
  mysql_query($sql,$link);

  $sql="SELECT nombre from control where id=".$_POST['control'];
  $res=mysql_query($sql,$link);

  if($reg=mysql_fetch_array($res))
   {
?>
<HTML><HEAD>
<TITLE>Par&aacute;metros del Control</TITLE>
</HEAD>

<BODY bgcolor="#FFFFE3" text="#000000" link="#0000FF" vlink="#800080"
alink="#FF0000" >
<form action="control.php" method="post">
Control : <?php echo $reg['nombre'];?>
<br><br>

<?php
  $sql="SELECT nombre,valor from parametro where id_cont=".$_POST['control'];

  $res=mysql_query($sql,$link);
  echo "<TABLE WIDTH=400 BORDER=0></TR><TR ALIGN=LEFT VALIGN=MIDDLE>";
  $cont=0;
  while($reg=mysql_fetch_array($res))
  {
   $cont++;
   echo "<TD>".$reg['nombre']."</TD><TD>";
?>
    <INPUT TYPE=TEXT NAME="<?php echo $reg['nombre'];?>" SIZE=8 value="
                           <?php echo $reg['valor'];?>"></TD>
<?php
  if($cont==3)
  {
    $cont=0;
    echo "</TR><TR ALIGN=LEFT VALIGN=MIDDLE>";
  }
}
?>
</TR>
<TR ALIGN=CENTER>
  <TD COLSPAN=3>Tiempo de muestreo</TD>
  <TD COLSPAN=3><INPUT TYPE=TEXT NAME="tm" SIZE=10 value="0.150"></TD>
   <input type="hidden" name="id_cont" value="<?php echo $_POST['control'];?>">
   <input type="hidden" name="start" value="si">
<TR ALIGN=CENTER>
```

```
  <TD COLSPAN=6><input type="submit" value="Iniciar"></TD>
</TR>
</TABLE>
</form>

<?php
}
?>
</BODY>
</HTML>

<?php
}
else error ("Sin permisos para controlar el sistema Levineu");
}
else error ("Sin permisos para monitorizar el sistema Levineu");
?>
```

```
<?php
include ("sesiones.php");
include ("bdd.php");
// Fichero control.php

if (isset($_SESSION['levineu']))
{
 if ($_SESSION['levineu']==2)
 {

  $sql="SELECT nombre from control where id=".$_POST['id_cont'];
  $res=mysql_query($sql,$link);

  if($reg=mysql_fetch_array($res))
  {

?>
<HTML>
<HEAD>
<TITLE>Control</TITLE>
</HEAD>

<BODY bgcolor="#FFFFE3" text="#000000" link="#0000FF" vlink="#800080"
alink="#FF0000" >
Control : <?php echo $reg['nombre'];?>
<br><br>
<form action="control.php" method="post">
<?php
  $sql="SELECT nombre from parametro where id_cont=".$_POST['id_cont'];

  $res=mysql_query($sql,$link);

  echo "<TABLE WIDTH=400 BORDER=0><TR ALIGN=LEFT VALIGN=MIDDLE>";
  $cont=0;
  $parametros=0;
  while($reg=mysql_fetch_array($res))
  {
   $parametros=1; //Al menos hay un parametro:Cambiar Configuraci{\'o}n.
   $cont++;
   $par=$reg['nombre'];
   $valor=$_POST[$par];

   echo "<TD>".$par."</TD><TD>";
?>
<INPUT TYPE=TEXT NAME="<?php echo $par;?>" SIZE=8 value="<?php echo $valor;?>">
</TD>

<?php
   if($cont==3)
   {
    $cont=0;
    echo "</TR><TR ALIGN=LEFT VALIGN=MIDDLE>";
   }
   $sql="UPDATE parametro set valor=".$valor." WHERE id_cont=";
   $sql.=$_POST['id_cont']." and nombre='".$par."'";
```

```
   mysql_query($sql,$link);
  }
  echo "</TR>";
  if (isset($_POST['tm']))
  {
   $valor=$_POST['tm'];
   $sql="UPDATE comun set valor=".$valor." WHERE nombre='tm'";

   mysql_query($sql,$link);
  }
  $sql="UPDATE comun set valor=1 WHERE nombre='conf'";
  mysql_query($sql,$link);
?>
<input type="hidden" name="id_cont" value="<?php echo $_POST['id_cont'];?>">
<input type="hidden" name="start" value="no">

<TR ALIGN=CENTER>
<?php
  if($par)
  {
?>
 <TD COLSPAN=4>
  <input type="submit" value="Cambiar Configuraci{\'o}n"></TD>
<?php
  }
?>
</form>
<?php
   $sql="UPDATE comun set valor=".$_POST['id_cont']." WHERE nombre='selector'";

   mysql_query($sql,$link);
   if($_POST['start']=="si")
   {
    $sql="UPDATE comun set valor=0 WHERE nombre='stop'";
    mysql_query($sql,$link);

    $sql="UPDATE comun set valor=1 WHERE nombre='start' or nombre='conf'";
    mysql_query($sql,$link);
   }
}
?>
<form action="fin.php" method="post">
<Td COLSPAN=2>
<input type="submit" value="DETENER"></TD></TR></TABLE>
</form>
<br><a href="activex.php" target="cam">Cambiar C{\'a}mara</a>
</BODY>
</HTML>
<?php
}
else error ("Sin permisos para controlar el sistema Levineu");
}
else error ("Sin permisos para monitorizar el sistema Levineu");
?>
```

```
<?php
include ("sesiones.php");
include ("bdd.php");
// Fichero fin.php
if (isset($_SESSION['levineu']))
{
 if ($_SESSION['levineu']==2)
 {

  $sql="UPDATE comun set valor=1 where nombre='stop'";
  $res=mysql_query($sql,$link);
?>
<HTML>
<HEAD>
<TITLE>Fin del Control</TITLE>
</HEAD>

<BODY bgcolor="#FFFFE3" text="#000000" link="#0000FF" vlink="#800080"
 alink="#FF0000" >
<form action="fichero.php" method="post">
Nombre del fichero para almacenar datos:<input type="text" name="fic">
<input type="submit" value="GUARDAR DATOS">
</form>
<br>
Archivos presentes en el servidor:
<br>
<TABLE ALIGN=CENTER>
<?php
  $dir="users/".$_SESSION['usuario']."/*";
  $lista=glob($dir);
  if($lista)
   {
    sort($lista);
    reset($lista);
    foreach($lista as $fichero)
    {
     echo "<TR><TD>".basename($fichero)."</TD></TR>";
    }
   }
?>
</BODY>
</HTML>

<?php
 }
 else error ("Sin permisos para controlar el sistema Levineu");
}
else error ("Sin permisos para monitorizar el sistema Levineu");
?>
```

```
<?php
include ("sesiones.php");
//Fichero fichero.php

if($_POST['fic']!="")
{
 if(file_exists("c:\\temporal\\fichero.m"))
 {
   $fichero="users\\".$_SESSION['usuario']."\\".$_POST['fic'];
   if (file_exists($fichero)) unlink($fichero);

   copy("c:\\temporal\fichero.m",$fichero);
  }
}
header("Location: inicio.php");
?>
```

```
<?php
include ("sesiones.php");
include ("bdd.php");
// Fichero Monitoriza.php

$sql="SELECT p.equipo FROM permmon p,usuario u,equipo e WHERE p.equipo=1 AND ";
$sql.="u.login='".$_SESSION['usuario']."' AND u.id=p.user";

$resultado=mysql_query($sql,$link);

$sql="SELECT p.equipo FROM permcont p,usuario u,equipo e WHERE p.equipo=1 AND";
$sql.=" u.login='".$_SESSION['usuario']."' AND u.id=p.user";

$resultado2=mysql_query($sql,$link);

if(mysql_num_rows($resultado)||mysql_num_rows($resultado2))
{
  $sql= "SELECT estado FROM equipo WHERE id=1";
  $resultado=mysql_query($sql,$link);

  $reg=mysql_fetch_array($resultado);


  if ($reg['estado']!=2)
  {
    if(isset($_SESSION['levineu']))
    {
        if($_SESSION['levineu']==1 || $_SESSION['levineu']==2)
        {
            $monitor="si";
         }
        else $monitor="no";
    }
    else
    {
    $_SESSION['levineu']=1; //Puede monitorizarl
    $monitor="si";
    }
}
else $monitor="no";

if ($monitor=="si")
{
?>
<HTML>
<HEAD>
  <TITLE>Control Remoto</TITLE>
</HEAD>
<FRAMESET COLS="425,*">
    <FRAME NAME="cam" SRC="activex.php" FRAMEBORDER=0>

  <FRAME NAME="monitor" SRC="app.php" FRAMEBORDER=0>
  <NOFRAMES>
    <P>Lo siento, pero s{\'o}lo podr{\'a}s ver esta p{\'a}gina
    si tu navegador tiene la capacidad de visualizar
```

```
    marcos.</P>
  </NOFRAMES>
</FRAMESET>
</HTML>
<?php
}
else
{
?>
<TABLE ALIGN=CENTER WIDTH="70%">
<TR ALIGN=CENTER>
  <TD> El equipo Levineu no est{\'a} libre para ser monitorizado.</TD>
</TR>
<TR ALIGN=CENTER>
  <TD><A href="portada.html">Inicio</A></TD>
</TR>
</TABLE>
<?php
}
}
else
{
?>
<TABLE ALIGN=CENTER WIDTH="70%">
<TR ALIGN=CENTER>
  <TD>Sin permisos para monitorizar el equipo Levineu.</TD>
</TR>
<TR ALIGN=CENTER>
  <TD><A href="portada.html">Inicio</A></TD>
</TR>
</TABLE>
<?php
}
?>
```

```
<?php
include ("sesiones.php");
include ("bdd.php");
// Fichero ficheros.php
?>

<HTML>
<HEAD>
<TITLE>Ficheros Disponibles</TITLE>
</HEAD>

<BODY bgcolor="#FFFFE3" text="#000000" link="#0000FF" vlink="#800080"
 alink="#FF0000" >

Archivos presentes en el servidor:
<br>
<TABLE ALIGN=CENTER>
<?php
$dir="users/".$_SESSION['usuario']."/*";
$lista=glob($dir);
if($lista)
{
 sort($lista);
 reset($lista);
 foreach($lista as $fichero)
 {
?>
<A href="<?php echo $fichero;?>">
             <?php $partes_fic=pathinfo($fichero);
            echo $partes_fic['basename'];?>
</A><br>
<?php
 }
}
?>
</BODY>
</HTML>
```

```
<?php
session_start();
include("bdd.php");
// Fichero desc.php

if (isset($_SESSION['levineu']))
{
    if ($_SESSION['levineu']==2)
     {
       $sql="UPDATE comun set valor=1 where nombre='stop'";
       mysql_query($sql,$link);
       $sql="UPDATE equipo SET estado=0 WHERE id=1";
       mysql_query($sql,$link);
    }
}

$_SESSION = array();
session_destroy();
?>
<TABLE ALIGN=CENTER VALIGN=MIDDLE WIDTH="50%">
<TR ALIGN=CENTER>
  <TD>Sesi{\'o}n cerrada con {\'e}xito.</TD>
</TR>
<TR ALIGN=CENTER>
  <TD><a href="labrem.html" target="_top">Inicio</a></TD>
</TR>
</TABLE>
```

```
<?php
include ("sesiones.php");
include ("bdd.php");
// Fichero levineu.php

if(isset($_SESSION['levineu']))
{
 $sql="SELECT activo from equipo where nombre='levineu'";
 $res=mysql_query($sql,$link);
 $reg=mysql_fetch_array($res);
 $activado=$reg['activo'];

 if($activado=="si")
  echo "<EQUIPO NOMBRE='Levineu'>Sistema de Levitacion neumatico";

 else
 {
  $m="<EQUIPO NOMBRE='Levineu' DETENIDO='si'>Sistema de Levitacion neumatico";
  echo $m;
 }


 $sql="SELECT c.nombre,c.id from control c,comun cm where cm.id_eq=1 ";
 $sql.="and cm.nombre='selector' and c.id=cm.valor";

 $res=mysql_query($sql,$link);
 if($reg=mysql_fetch_array($res))
 {
  $control=$reg['id'];

  echo "<CONTROL NOMBRE='".$reg['nombre']."'>";

  $sql="SELECT nombre,valor from comun where nombre='tm'";
  $res=mysql_query($sql,$link);

  while($reg=mysql_fetch_array($res))
   echo "<PARAMETRO NOMBRE='".$reg['nombre']."'>".$reg['valor']."</PARAMETRO>";

  $sql="SELECT nombre,valor from parametro where id_cont=".$control;
  $sql.=" order by nombre";
  $res=mysql_query($sql,$link);

  while($reg=mysql_fetch_array($res))
   echo "<PARAMETRO NOMBRE='".$reg['nombre']."'>".$reg['valor']."</PARAMETRO>";

  echo "</CONTROL>";
 }
 if($_SESSION['levineu']==2)
 {
  $sql="UPDATE equipo SET ult_acceso = CURRENT_DATE,ult_hora=CURRENT_TIME";
  mysql_query($sql,$link);
 }
 if($activado=="si")
 {
  $sql="SELECT nombre,valor from comun where nombre='altura' or nombre='ref'";
```

```
  $sql.=" order by nombre";
  $res=mysql_query($sql,$link);

  while($reg=mysql_fetch_array($res))
  {
    echo "<SIGNAL NOMBRE='".$reg['nombre']."' MIN='0' MAX='80'>";
    echo $reg['valor']."</SIGNAL>";
  }
  if(($_SESSION['levineu']==2) &&($_GET['ref']!=""))
  {
    $sql="UPDATE comun set valor=".$_GET['ref']." WHERE nombre='ref'";
    mysql_query($sql,$link);
  }
 }
 echo "</EQUIPO>";
}
?>
```

Bibliografía

[1] Lázaro, Antonio Manuel
Programación gráfica para el control de instrumentos
Paraninfo, 2001

[2] Fábrega,Pedro Pablo
PHP 4, Serie Práctica.
Prentice Hall, 2000

[3] DuBois,Paul
Edición Especial MySQL
Prentice Hall, 2001

[4] Naughton,Patrick , Herbert Schildt
Java. Manual de Referencia
McGraw-Hill

[5] Beatriz Pontes, Alfonso Cepeda, Eduardo F.Camacho
Gestión e implementación de una pasarela HTTP-OPC aplicado a una planta solar.

[6] Carlos A.Zuluaga Toro, Carlos G.Sánchez Toro, Elkin A. Rodríguez Ortiz
Laboratorio de automática vía Internet.

[7] Domínguez,M. ,P.Reguera,J.J.Fuertes
Laboratorio remoto para la enseñanza de la automática en la universidad de León.

[8] Dormido, S., J.Sánchez, F. Morilla
Laboratorios virtuales y remotos para la práctica a distancia de la automática.

[9] Guzmán,J.L. ,M. Berenguel, F. Rodríguez
Laboratorio remoto para el control de una maqueta de invernadero.

[10] Marco Casini, Domenico Prattichizzo, Antonio Vicino
The Automatic Control Telelab: a Remote Control Engineering Laboratory.

[11] Rafael Puerto Manchón, Luis Miguel Jiménez García, Óscar Reinoso García, César Fernández Peris
Laboratorio vía Internet para control de procesos.

[12] Rosado,L. ,J.R.Herreros
Nuevas aportaciones didácticas de los laboratorios virtuales y remotos en la enseñanza de la Física.

[13] National Instruments
LabVIEW User Manual
April 2003 Edition

[14] National Instruments
Using External Code in LabVIEW
April 2003 Edition

[15] National Instruments
LabVIEW Data Storage
Application Note 154

[16] Proyecto Fin de Carrera: "Construncción, configuración y modelado de un sistema de levitación neumática".
Autor: Caparrós Jiménez, José Julio .
Tutor: Manuel G. Ortega Linares.
Escuela Superior de Ingenieros. Universidad de Sevilla

[17] Proyecto Fin de Carrera: "Diseño de estrategias de control en un sistema de levitación neumática".
Autor: Breña Lajas, Fernando Borja.
Tutor: Manuel G. Ortega Linares.
Escuela Superior de Ingenieros. Universidad de Sevilla