Revista Iberoamericana de Automática e Informática Industrial
CEA comité español de automática
ISSN: 1697-7912. Vol. 4, Núm. 3, Julio 2007, pp. 86-95
http://riai.isa.upv.es

ESTRUCTURA ABIERTA DE SOFTWARE PARA UN ROBOT INDUSTRIAL

Emanuel Slawiñski*, José Francisco Postigo, Vicente Mut, Darío Carestía, Federico Castro

Instituto de Automática (INAUT). Universidad Nacional de San Juan.
Av. Libertador San Martín 1109 (oeste). J5400ARL. San Juan, Argentina.
**e-mail: slawinski@inaut.unsj.edu.ar*

Resumen: En este trabajo se realiza el diseño, implementación y experimentación de una arquitectura "abierta" de software para el control de un robot industrial. En esta forma se pueden probar rápidamente algoritmos de control e incorporar nuevos dispositivos (sensores, actuadores, dispositivos de mando) facilitando tareas de investigación y enseñanza en el área de robótica, con un bajo tiempo de implementación y reutilización.

Palabras Clave: objetos, robot manipulador, software abierto.

1. INTRODUCCIÓN

El control automático se ha convertido en parte importante e integral de los procesos industriales modernos. Los avances, tanto en investigación básica como aplicada en control automático, brindan los medios para lograr el funcionamiento óptimo de sistemas dinámicos, mejorar la calidad y reducir los costos de producción.

La robótica como parte de la automática es hoy un área de intensa investigación en todos sus campos. Asimismo, en la industria ocupa un lugar destacado en el proceso de modernización e innovación productiva (UNECE and IFR, 2005). La incorporación de robots en industrias de fabricación resulta en productos de mejor calidad y menor precio. Para ello, el diseño de una estructura de software abierta (William, 1994), (Frederick and Albus, 1997), tiene una importancia relevante.

La principal característica de una estructura de software abierta para robótica es la definición de la interfase de los componentes y la estructura interna básica. En el área industrial, uno de los trabajos más importantes fue realizado a partir de 1992, dentro del marco del proyecto europeo OSACA (*Open System Architecture for Controls within Automation systems*, http://www.osaca.org). Se realizan esfuerzos similares en Japón, a través de OSEC bajo IROFA (Sawada and Akira, 1997), y en Estados Unidos a través de OMAC (*Open, Modular Architecture Control*, cuya página web es http://www.omac.org). Los investigadores tienen como objetivo desarrollar un sistema de control abierto incluyendo el modelo de referencia de componentes, la interfase de aplicación general y la estructura que hace que todos los componentes trabajan juntos. Hasta el momento, los desarrollos obtenidos no fuerzan a todos los fabricantes a conformar un estándar único. En cambio, estos desarrollos utilizan recursos existentes para realizar funciones usuario-específicas.
Por otra parte, existen varios paquetes de software comerciales para el control de robots sobre plataformas Windows, como por ejemplo *Advanced Robotics Interface for Applications* (ARIA) usado en los robots fabricados por *ActivMedia Robotics* (http://www.activrobots.com), el software *BotController* desarrollado por *MobotSoft* (http://www.mobotsoft.com), usado en la programación de los conocidos robots *Khepera* y *Koala*; entre otros. Aunque estos paquetes de software para robots tienen muchas prestaciones para los robots hacia los cuales fueron diseñados, los mismos son librerías de software potentes pero 'cerrados', que circunscriben sus aplicaciones a sus propios robots. Lo cual imposibilita su utilización para desarrollar software de robótica de acuerdo al hardware que posea un centro de investigación.

El objetivo de este trabajo es diseñar e implementar una estructura de software "abierta" con componentes reutilizables, que haga de nexo entre el hardware de un robot industrial y un algoritmo de control en particular, de forma tal que la implementación de éste último se realice en un corto tiempo. La aplicación de esta estructura de software bajo entornos Windows NT, la cual puede ser extendida a otros robots, esta destinada principalmente para trabajos de investigación y formación de recursos humanos en el área de robótica.

Para cumplir con los requerimientos establecidos, se desarrolló un sistema de software compuesto de dos programas: un programa realiza la adquisición de datos de los distintos sensores que posee el robot, y es el encargado de entregar los valores de las acciones de control a cada uno de los motores. Las variables correspondientes a cada uno de los sensores se colocan en una región de memoria compartida, del mismo modo, las acciones de control se leen de la misma región de memoria. Un segundo programa se encarga de leer los sensores, ejecutar el algoritmo de control y escribir las acciones de control en memoria compartida. De esta manera, se logra separar la ejecución del algoritmo de control de la transmisión de señales entre software y componentes de hardware. Esta separación permite reducir el tiempo de implementación para algún algoritmo de control en particular, debido a que dicho controlador se encuentra aislado de la adquisición y transmisión de señales desde y hacia el robot manipulador.

Este trabajo se organiza de la siguiente manera: la sección 2 describe los grupos de usuarios para los cuales se diseñó el sistema de software. En la sección 3 se describe brevemente el robot manipulador industrial utilizado. En la sección 4 se describe la estructura del sistema de software desarrollado. En la sección 5 se muestran experimentos realizados con un robot industrial para mostrar el desempeño y funcionalidad del sistema desarrollado. Finalmente, en la sección 6 se brindan las conclusiones de este trabajo.

2. USUARIOS

El presente trabajo esta dirigido a usuarios que pueden ser clasificados en distintos niveles según el modo de utilización del software:

Nivel 1: Aquellos que hacen uso del software sin necesidad de realizar una modificación del mismo, como por ejemplo:

- Alumnos de grado.
- Alumnos que realizan su Maestría o Doctorado.

Nivel 2: Quienes desean llevar a la práctica el desarrollo teórico de un algoritmo de control en particular, siendo necesario sólo la modificación del algoritmo implementado, haciendo uso del resto del sistema. Los usuarios de este grupo deben tener un conocimiento mínimo de las estructura de datos y funcionamiento del sistema para realizar la modificación adecuadamente. Ellos pueden ser:

- Alumnos de Maestría o Doctorado.
- Investigadores.

Nivel 3: Son aquellos que deseen realizar su propia implementación de software, haciendo uso solamente de la adquisición de señales de los distintos sensores.

Nivel 4: Son aquellos que desean realizar la incorporación de uno o más sensores y/o actuadores. Este tipo de usuario necesita poseer un conocimiento amplio de las estructuras de datos y funcionamiento del sistema.

3. ROBOT INDUSTRIAL BOSCH SR-800

El manipulador robótico BOSCH SR-800 es un brazo industrial del tipo SCARA. Este tipo de manipuladores tiene sus articulaciones dispuestas en configuración de coordenadas cilíndricas. Se usan para proveer movimientos rápidos y suaves, especialmente en operaciones de ensamble como las que requieren inserción de objetos en orificios. Cuenta con cuatro articulaciones o grados de libertad a saber:

- movimiento de hombro (articulación o eje 1 - rotacional)
- movimiento de codo (articulación o eje 2 - rotacional)
- desplazamiento de muñeca (articulación o eje 3 - prismática)
- rotación de muñeca (articulación o eje 4 - rotacional)

Todas estas articulaciones son verticales. Las articulaciones 1, 2 y 4 se mueven en el plano horizontal, mientras que el movimiento en el plano vertical es provisto por la articulación 3. Cabe destacar que la articulación 4 está desacoplada del resto gracias a un sistema de correas dentadas dispuesto para tal fin. La Figura 1 muestra la configuración del manipulador, además de sus dimensiones físicas.

El manipulador SCARA SR-800 posee una Unidad de Control Riho, provista por el fabricante. La misma esta compuesta por 4 servo-amplificadores destinados a comandar las 4 articulaciones independientes del robot y de una CPU encargada de ejecutar el algoritmo de control de posición con lazo interno de velocidad.

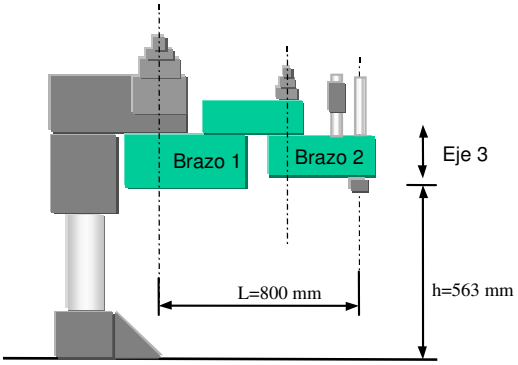

Figura 1. Dimensiones del manipulador BOSCH SR-800.

A los fines de realizar estudios aplicando diferentes algoritmos de control sobre este sistema, se aisló la CPU provista por el fabricante ("sistema cerrado"), la cual se reemplazó por un sistema de control basado en PC para obtener un "sistema abierto". Dicha PC posee placas de adquisición de datos Microaxial AD/DA-Q12 para la conexión con el manipulador. En la Figura 2 se aprecia un diagrama en bloques del robot industrial incluyendo sus componentes principales.

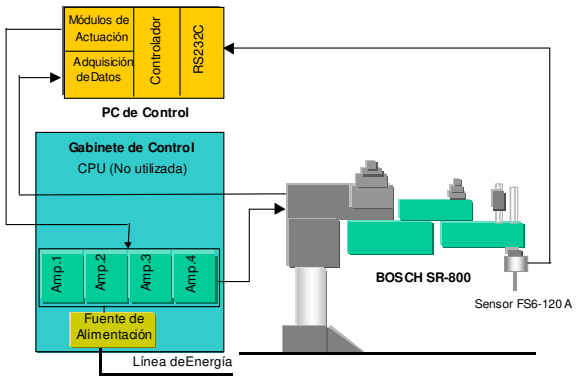

Figura 2. Diagrama de control del manipulador BOSCH SR-800.

El robot posee un sensor de fuerza modelo FS6-120 A, el cual permite medir fuerzas y momentos en 3 ejes (X, Y, Z), enviando dichas lecturas vía RS232C hacia la PC de control.
El sensor está compuesto principalmente por un conjunto de bandas extensiométricas encargadas de medir la fuerza aplicada sobre el extremo operativo del robot manipulador, una placa electrónica acondicionadora de señales (tensión y filtrado) provenientes de dichas bandas, un microprocesador y una interfase de conexión RS232C.

Por otra parte, se adicionó al hardware del robot una cámara para adquirir imagen. La cámara utilizada es una webcam comercial de uso general, marca Genius modelo VideoCAM Messenger. Esta cámara posee una resolución de 640x480 (300K) píxeles y se conecta a la PC mediante el puerto USB. Mediante esta cámara, se obtiene la posición (en coordenadas de píxel) de objetos presentes en el espacio de trabajo del robot industrial.

4. DISEÑO DEL SOFTWARE

4.1 Requerimientos funcionales del sistema

En base a los objetivos de este trabajo, los requerimientos funcionales del sistema son:

- Software de control para el robot manipulador industrial SCARA BOSCH SR-800, el cual debe permitir implementar y evaluar diversos algoritmos de control utilizando información de sensores de fuerza, posición y características de la imagen almacenando todas las variables relevantes en archivo para su posterior análisis.
- La arquitectura del software desarrollado debe ser flexible y abierta para permitir la adición de nuevos componentes como sensores, actuadores, dispositivos de mando, etc.

4.2 Diseño con reutilización

En muchas disciplinas de ingeniería, el proceso de diseño se basa en la reutilización de componentes, por ejemplo en ingeniería mecánica o eléctrica, no se llevan a cabo diseños en los que cada componente tenga que ser diseñado en forma aislada, como el caso de tornillos, tuercas, etc. La ingeniería del software en las últimas décadas viene haciendo sus mayores esfuerzos para imitar estas técnicas, tratando de encapsular las unidades de software para su posterior reutilización (Sommerville, 2002). Se plantea una arquitectura basada en objetos para el manejo de los distintos dispositivos o componentes hardware como sensores, actuadores, dispositivos de mando, etc. De esta forma se encapsulan los datos y las tareas internas de cada dispositivo. Además, la estructura de los objetos diseñados esta basada en un temporizador que corre en un hilo independiente (los hilos son el corazón de la capacidad multitarea de Windows encargándose de ejecutar el código almacenado en los procesos) y funciones miembro que abren, leen, escriben y cierran el dispositivo. El diseño realizado de los componentes de software, correspondiente a diferentes dispositivos, puede ser utilizado como soporte de nuevos componentes de software para algún nuevo componente de hardware.

4.3 Sistema operativo y entorno de programación

Se utiliza el sistema operativo Windows (se recomienda trabajar con Windows NT, por ejemplo

Windows 2000 Server). Aunque este sistema operativo no es de tiempo real, el mismo dispone de un conjunto de características que justifican su utilización en este proyecto. Windows NT es el sistema operativo más utilizado del mercado, lo cual brinda gran disponibilidad de utilidades y drivers, además es un sistema operativo multitarea (permite la ejecución de múltiples procesos e hilos en forma simultánea) y permite utilizar máquinas que contienen más de un procesador asignándolos a distintos hilos (Russinovich and Solomon, 2004).

Los programas se desarrollaron con el lenguaje de programación Visual C ++ (Zaratian, 1998). Para la construcción de cada aplicación, se utilizó la librería de clases MFC (Microsoft Foundation Class Library) en conjunto con la plataforma SDK (Software Development Kit). También se utilizaron los asistentes AppWizard y ClassWizard, que provee Visual C++ para trabajar con MFC (Sohar, 1998).
El asistente ClassWizard simplifica algunas tareas como la creación de nuevas clases, definición de manejadores de mensajes, sobrescribir las funciones de las clases de MFC o asignar variables a controles de cuadros de dialogo (Gurenwich and Gurenwich, 1998).
Los distintos objetos en los programas, se implementan en clases y se crean al inicio del programa, pero no comienzan a trabajar hasta que se llama su función de activación. En particular, los objetos de sensores o actuadores poseen una función asociada, que se ejecuta en un hilo aparte en forma periódica, con un periodo de muestreo apropiado para cada dispositivo. Estas no son funciones miembro de las clases, sino que se implementan en temporizadores multimedia mediante funciones de la API de Windows, y brindan la mejor resolución (1 milisegundo) posible para el sistema operativo utilizado (Richter, 1997).
Por otra parte, la sincronización de los programas se realiza mediante objetos de sincronización de Windows, los mismos se crean y utilizan con funciones de la API (Richter, 1997).

4.4 Diseño de la estructura de software

La estructura se diseña de forma tal de conseguir módulos independientes para la interfase gráfica, los componentes software de cada componente o dispositivo hardware y el algoritmo de control. La figura 3 muestra un diagrama en bloques del sistema de software desarrollado. Las diversas tareas son divididas y asignadas a dos procesos o programas, que se comunican entre si, y trabajan en forma cooperativa. Las tareas de comunicación con sensores y actuadores, adecuación de señales y sincronización del ciclo de barrido de control son resueltas por el programa llamado de aquí en adelante *Programa de Tiempo Crítico*, mientras que el algoritmo de control corre en el programa llamado *Programa de Control*.

Figura 3. Diagrama en bloques del sistema de software desarrollado.

La función de procesamiento del algoritmo de control se puede modificar fácilmente, lo cual permite escribir controladores en un corto período. Dicha función se ejecuta periódicamente a intervalos del tiempo de barrido del programa (establecido desde la interfase gráfica al usuario), y permite el acceso rápido para leer cualquiera de las variables de los sensores (las cuales son actualizadas en cada instante de muestreo) y escribir las acciones de control.
Para agregar nuevas clases para el manejo de dispositivos, se pueden tomar como referencia las clases de objetos desarrolladas para los sensores-actuadores del robot industrial utilizado. Por otra parte, la interfase gráfica está aislada en objetos dedicados para tal fin, separándose de la funcionalidad del programa.

A continuación, se describirán brevemente las características principales de cada programa de la estructura de software desarrollada

4.4.1 Programa de Tiempo Crítico

Este programa es el responsable de comunicarse con los sensores y actuadores a través del hardware de adquisición y control, mantener actualizados todos los datos leídos de sensores en una estructura de memoria compartida, y de sincronizar al programa de control para que procese esos datos en cada periodo de muestreo.

Este programa posee cuatro clases de objetos, los cuales son:

A) Objeto motor: este objeto es el encargado de aplicar las acciones de control establecidas por el controlador sobre los motores del robot industrial

mediante los conversores D/A de la placa de adquisición de datos dispuesta sobre la PC de control.

B) Objeto cámara web: este objeto se conecta con el driver provisto por el fabricante (puede ser cualquier CámaraWeb) para capturar imágenes a la tasa establecida a través del puerto USB del PC de control. Además, se extrae a partir de la imagen cruda en formato HSV, la posición y área en píxeles de los objetos presentes en el espacio de trabajo del robot industrial. Se utiliza detección por color y una segmentación basada en separar conjuntos de píxeles contiguos (Parker, 1997), (González, 1993). Para la implementación en software se utilizaron las librerías IPL de procesamiento de imagen provista por Microsoft (Intel Corporation, 2000).

C) Objeto posición: es el encargado de medir la lectura de los encoders de cada articulación del robot industrial a través de los puertos de entrada de la placa de adquisición de datos situada en la PC de control.

D) Objeto fuerza: este objeto lee la medición de fuerza realizada por el sensor FS6-120 comunicándose con el mismo mediante el puerto serie RS-232C del PC de control.

Por otra parte, se implementa también un temporizador multimedia que funciona como reloj del sistema con una frecuencia (frecuencia de muestreo) seleccionada por el usuario al momento del inicio de la ejecución del control. Por otra parte, el programa posee una interfaz de usuario formada por un cuadro de dialogo, en el cual se encuentra un conjunto de controles gráficos que permiten seleccionar los sensores a utilizar, configurar sus parámetros, seleccionar el tiempo de barrido de todo el sistema de control, y comenzar o detener su ejecución.

4.4.2 Programa de Control

Este programa tiene la principal tarea de procesar el algoritmo de control. Para ello, debe antes obtener todos los valores de sensores del robot. Se comunica con el *Programa de Tiempo Crítico* para recibir los datos de sensores y para enviar las acciones de control, esto lo realiza en la misma PC a través de memoria compartida.
El *Programa de Control* posee un hilo dedicado al procesamiento del algoritmo de control. Este se crea al iniciar el control y se sincroniza por medio de un objeto de sincronización controlado por el *Programa de Tiempo Crítico.*
La interfaz de usuario esta formada por un cuadro de dialogo, en el cual se encuentra un conjunto de controles gráficos que permiten la visualización de datos, la modificación de los parámetros del control, la inicialización y detención del algoritmo de control.

4.4.3 Programación avanzada bajo Windows NT

En esta sección, nosotros mencionamos algunos aspectos relevantes a tener en cuenta para una programación eficiente, en cuanto al uso de los recursos de hardware disponibles, bajo plataformas Windows NT:

a) Analizar adecuadamente la cantidad de hilos que debe tener cada programa. Emplee la función *CreateThread* de la API para crear cada hilo secundario (el hilo principal corresponde a la función *WinMain*)
b) Evitar operaciones 'bloqueantes' usando las funciones de lectura y escritura a un dispositivo (por ejemplo: comunicación serial, escritura de datos al disco rígido en línea, etc.) en forma asincrónica. Para ello debe utilizar la función de la API *CreateFile* definiendo la estructura OVERLAPPED.
c) No definir variables globales, es conveniente para reutilizar y extender el código a la utilización de objetos.
d) No esperar un evento utilizando las funciones conocidas en la programación estructurada (por ejemplo, la función *while*), sino que debe utilizar las funciones de la API de Windows para esperar eventos, como por ejemplo la función *WaitForSingleObject.*
e) Utilizar memoria compartida a través de las funciones *CreateFileMapping* y *MapViewOfFile* de la API de Windows para compartir datos entre procesos.
f) Incorporar temporizadores multimedia de alta resolución, a través de la función *timeSetEvent* de la API, para establecer el período de muestreo. No usar la función *SetTimer,* ya que ésta genera mensajes que van a una cola de espera. La función *SetTimer* se utiliza principalmente para la actualización en pantalla de datos.
g) Para sincronizar fragmentos o partes de código importantes del sistema, utilizar eventos, los cuales pueden ser definidos mediante la función *CreateEvent* de la API.
h) Para grabar en línea datos al disco rígido, es conveniente utilizar un esquema de doble buffer, donde cuando se llena el primero, se copia al otro buffer, de forma que éste último sea escrito en forma asincrónica al disco quedando liberado el primer buffer para seguir 'juntando' datos en cada período de muestreo.
i) Usar las funciones *SetPriorityClass* y *SetThreadPriority* de la API para establecer la prioridad de un hilo de un proceso dado (Russinovich and Solomon, 2004). . Esta prioridad debe ser elegida adecuadamente de acuerdo a la importancia de la tarea que desarrolla el hilo y a los requerimientos de tiempo y procesamiento que requiere. Por ejemplo, el hilo de control que se ejecuta en cada período de muestreo para ejecutar el

algoritmo de control debe tener una prioridad más elevada que el hilo que graba datos al disco en forma asincrónica.

5. EXPERIMENTACIÓN

Con el fin de evaluar el desempeño del sistema de software desarrollado, se realizan dos experimentos utilizando el robot industrial BOSCH SR-800, los cuales son: (A) implementación de un controlador de impedancia; (B) incorporar un joystick para generar comandos de posición sobre el robot y recibir realimentación de fuerza. A continuación se describen ambos experimentos.

5.1 Experimento A: implementación de un controlador

En este experimento, se implementa y ajusta un controlador de impedancia (Hogan, 1985) , de forma tal que una persona pueda mover fácilmente el robot industrial Bosch SR-800 (de gran peso) para escribir sobre una pizarra (ver la Figura 4).

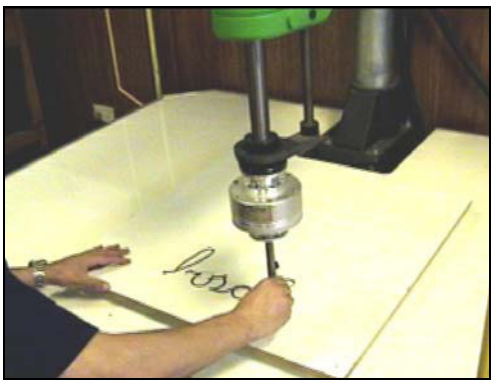

Figura 4. Experimento utilizando un controlador de impedancia.

Para realizar la tarea, se añade un marcador en el extremo operativo del manipulador. El experimento consiste en calibrar adecuadamente el controlador de impedancia implementado para que una persona pueda manipular fácilmente un marcador fijado en el extremo operativo del robot para escribir la palabra "*bosch"* sobre una pizarra.

5.1.1 Controlador de impedancia

Con el fin de comprobar la efectividad del diseño de software realizado, esto es, la capacidad para permitir implementar de forma sencilla un algoritmo de control cualquiera sin tener que considerar los detalles respecto del hardware (como por ejemplo, la adquisición de los datos) y de esta forma concentrar la atención en el estudio del algoritmo considerado; se implementó un algoritmo de control de impedancia. La ecuación del controlador es la siguiente:

$$x_{ai}(k) = -a_1 x_{ai}(k-1) - a_2 x_{ai}(k-2) + b_0 f_i(k) + b_1 f_i(k-1) + b_2 f_i(k-2) \quad (1)$$

donde k representa el periodo de muestro k-ésimo, x_{ai} es el ajuste de posición aplicado sobre el eje i generado por la fuerza f_i sobre dicho eje; y los coeficientes a_1 , a_2 , b_0 , b_1 y b_2 definen la impedancia deseada.

El esquema utilizado posee un lazo interno con un controlador PID de posición para cada articulación, como se muestra en la Figura 5. En la misma, x_d es el vector de referencias de posición, q es el vector de posición de las articulaciones del manipulador, x_a es el vector de ajuste y f es el vector que contiene las fuerzas de contacto sobre cada eje.

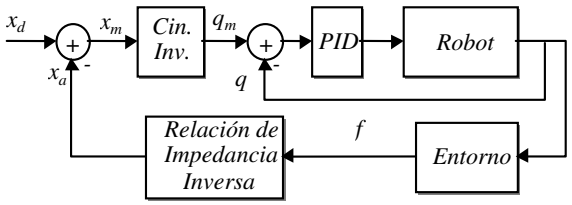

Figura 5. Diagrama en bloques del esquema de control de impedancia con lazo interno de posición.

El ajuste de parámetros de impedancia se realiza en forma independiente en cada uno de los tres ejes coordenados *x, y, z.*

5.1.2 Resultados experimentales

Se ajustaron los parámetros de los controladores de posición e impedancia para conseguir un desempeño adecuado. Esto implica, alta precisión en el eje z, de manera que no se produzcan rebotes en la pizarra, y baja resistencia al movimiento en las direcciones *x* e *y*, para poder desplazar con facilidad el marcador sobre la pizarra. El tiempo de muestreo utilizado es un milisegundo (Tm = 1ms). Por otro lado, los parámetros de los controlares PID de posición e impedancia utilizados en este experimento se muestran en la Tabla 1. Dichos parámetros se calibraron mediante prueba y error.

Tabla 1 Parámetros de control

Eje	PID (P, I, D)	K [N/mm]	D [N/mm.s]	M [N/mm.s^2]
x	(40,7,7)	0.07	0.05	0.001
y	(40,7,7)	0.07	0.05	0.001
z	(40,7,7)	0.15	2	0.0001

La Figuras 6, 7 y 8 muestran la evolución temporal de la posición del extremo operativo, respecto a los ejes *x, y, z,* respectivamente, en contraste con las referencias x_d, y_d, z_d, las cuales se encuentran fijas durante todo el experimento en x_d = 700[mm], y_d = 81[mm], y z_d = -176.3[mm]. También se observa la fuerza en la direcciones *x, y, z*. Debido a que las magnitudes de fuerza y posición poseen diferentes unidades físicas, se debió escalar la fuerza con factores de $0.0071\left[\frac{N}{mm}\right]$, $0.1667\left[\frac{N}{mm}\right]$ y $0.113\left[\frac{N}{mm}\right]$ para los ejes *x, y, z*, respectivamente; para poder contrastar su influencia en la posición del robot en una misma grafica.

Figura 6. Posición, referencia y fuerza según el eje *x*.

Figura 7. Posición, referencia y fuerza según el eje *y*.

En las tres direcciones de movimiento analizadas, se observa que la posición del extremo operativo se aleja de la referencia para acompañar a la fuerza ejercida por el operador.
En la dirección *z* no se observa defasaje temporal apreciable como en las direcciones *x* e *y*. Esto se debe a que los parámetros de impedancia se ajustaron de forma distinta en esta dirección (Ver Tabla 1) para obtener alta precisión de manera que no se produzcan rebotes en la pizarra y poder generar un trazo continuo con el marcador, quedando la componente inercial con un valor mucho menor, con lo que se consigue una respuesta más rápida. El ajuste de parámetros se realizó a través de una interfase grafica amigable diseñada bajo el entorno Windows, la cual permite cambiar fácilmente los parámetros del controlador implementado y grabar al disco nuevos experimentos de forma de comprobar el efecto sobre la respuesta del robot de dichos parámetros.

Figura 8. Posición, referencia y fuerza en *z*.

La Figura 9 muestra la evolución temporal del extremo operativo en 3-D (*x, y, z*). En ella, se puede apreciar el momento en que el operador levanta el marcador y lo regresa a su posición inicial.

Figura 9. Trayectoria en 3D realizada por el extremo operativo del robot industrial.

El análisis de estas graficas demuestra la capacidad del algoritmo de impedancia para acomodar la posición del extremo operativo en función de condiciones restrictivas del medio. Es posible controlar la relación dinámica entre el manipulador y el entorno modificando los parámetros del controlador de impedancia. Se remarca que la implementación del algoritmo de control descrito se realizó rápidamente utilizando la estructura de software diseñada.

5.2 Experimento B: Incorporación de un nuevo dispositivo

El siguiente experimento tiene como objetivo mostrar la flexibilidad del sistema para incorporar un dispositivo nuevo. En este caso, se añade al sistema de control un joystick con realimentación de fuerza, modelo Wingman Strike Force 3-D, fabricado por Logitech. A nivel de software, se desarrolló un componente software para el joystick incorporando al sistema la funcionalidad de este nuevo dispositivo.
Un operador humano maneja el robot industrial entregando consignas de posición mediante el joystick y además, recibe realimentación de fuerza.

El esquema de control utilizado se muestra en la Figura 10, donde G(z) representa la dinámica del joystick, X_d y q_d son las referencias de posición generadas por el operador humano en coordenadas cartesianas y articulares, respectivamente; X y q son la posición del robot en coordenadas cartesianas y articulares. El algoritmo de control de posición que se utiliza es un controlador PID.

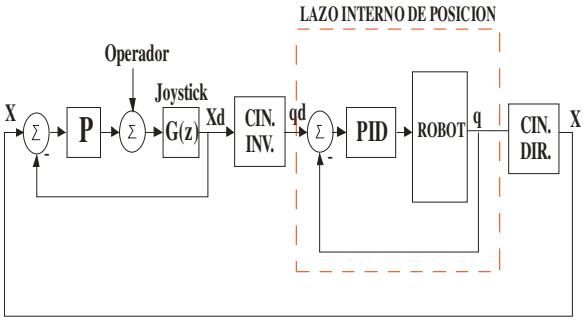

Figura 10. Diagrama en bloque del sistema de control.

Por otra parte, la fuerza de realimentación se genera a partir del error de posición mediante un controlador proporcional (P). El robot se maneja en coordenadas cartesianas, haciendo que el extremo operativo describa una trayectoria decidida en línea por el operador humano.

La condición inicial del robot es x = 645[mm], y = 180[mm], z = -5[mm]. El valor de los parámetros del controlador PID y el tiempo de muestreo utilizados en los experimentos, se indican en cada gráfica.
Las Figuras 11 y 12 muestran la evolución temporal de la posición y referencia de las articulaciones del hombro y del codo, respectivamente del robot industrial.

Figura 11. Posición y referencia de la articulación del hombro del robot en función del tiempo.

Figura 12. Posición y referencia de la articulación del codo del robot en función del tiempo.

Por otro lado, las Figuras 13, 14 y 15 muestran la evolución de la posición cartesiana en función del tiempo sobre los ejes x, y, z respectivamente y la referencia generada por el operador humano mediante un joystick para dichos ejes. La posición sobre el eje z coincide con la evolución de la articulación 3 del robot industrial debido a que dicha articulación es traslacional.

Figura 13. Evolución de la posición y referencia del extremo operativo del robot según el eje *x*.

Figura 14. Evolución de la posición y referencia del extremo operativo del robot según el eje *y*.

Figura 15. Evolución de la posición y referencia del extremo operativo del robot según el eje *z*.

Se observa de las Figuras precedentes que la respuesta del sistema es satisfactoria, logrando un error de seguimiento bajo.
La incorporación de un nuevo dispositivo requirió de la programación de un nuevo objeto (compatible en su estructura con los objetos existentes de dispositivos), lo cual demandó un corto tiempo de desarrollo debido al diseño global de la estructura de software diseñada.

Finalmente se remarca que bajo el sistema operativo Windows NT, el cual no es un sistema operativo de tiempo real, con una programación eficiente en esta plataforma junto con un hardware de adecuada velocidad logran un sistema satisfactorio para probar algoritmos de control en un entorno amigable y familiar para la mayoría de los usuarios.

6. CONCLUSIONES

En este trabajo se realizó el diseño, implementación y experimentación de una arquitectura de software "abierta" para probar rápidamente algoritmos de control sobre un robot industrial e incorporar fácilmente nuevos dispositivos (sensores, actuadores, dispositivos de mando) con el fin de ayudar en trabajos de investigación y enseñanza en el área de robótica.

El sistema desarrollado se dividió en dos programas que interactúan entre si, separando claramente las tareas del sistema de control. De esta manera, se logra un sistema modular, el cual permite una fácil reutilización y/o modificación de sus partes. El primer programa (Tiempo Crítico) realiza la adquisición y adaptación de las señales provenientes de los sensores tanto internos como externos que posee el manipulador, y envía hacia los actuadores, las acciones de control. El manejo de cada dispositivo se realiza a través de objetos, los cuales lograron ocultar los detalles particulares del manejo de cada dispositivo en particular, facilitando su utilización. Además, dichos objetos brindan el necesario encapsulamiento de datos y funciones para incorporar nuevos dispositivos sin afectar el funcionamiento de los demás. El segundo programa (Programa de Control) realiza el procesamiento del algoritmo de control en cada instante de muestreo. Este programa se encuentra sincronizado por el programa de *Tiempo Critico* mediante objetos de sincronización. Los programas desarrollados se comunican a través de memoria compartida.

El sistema desarrollado cumplió con los objetivos planteados remarcando principalmente el bajo tiempo de desarrollo empleado para implementar un algoritmo de control en particular y para añadir un nuevo dispositivo. Como trabajo futuro se prevé: añadir una webCam más, para trabajar con visión estéreo; conectividad TCP/IP entre el robot industrial y el controlador y la implementación de nuevos algoritmos para detección de objetos a partir de la imagen.

AGRADECIMIENTO

Este trabajo fue parcialmente financiado por el Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina.

REFERENCIAS

Frederick M. P. and J. S. Albus, (1997), *Open-architecture controllers*. IEEE Spectrum, pp. 60-64.

González W., (1993). *Digital Image Processing*. Adison-Wesley Publishing Company.

Gurenwich O. and N. Gurenwich, (1998). *Aprendiendo Visual C++ 5.0 en 21 días*. ISBN: 970-17-0030-9, Editorial Prentice Hall.

Hogan, N., (1985). *Impedance control: An approach to manipulation*. Part 1 – Theory . Journal of Dyn. Syst., Meas. and Control, **vol**. 107.

Intel Corporation, (2000), "Intel Image Processing Library", Document number 663791-005.

Parker J.R., (1997). *Algorithms for Image Processing and Computer Vision*. Wiley Computer Publishing.

Richter J. (1997). *Programación Avanzada en Windows*. ISBN: 84-481-1160-5, Editorial Mc Graw-Hill.

Russinovich M. E. and D. A. Solomon, (2004). *Microsoft® Windows® Internals*. Fourth Edition: Microsoft Windows Server™ 2003, Windows XP, and Windows 2000", ISBN 0-7356-1917-4.

Sawada C. and O. Akira, (1997), "Open controller architecture OSEC-II: architecture overview and prototype systems," in Proc. 6th Int. Conf. Emerging Technologies and Factory Automation, Sept. 1997, pp. 543-550.

Sohar C. (1998). *Aprenda Microsoft Visual C++ Ya*. Editorial Mc Graw-Hill.

Sommerville I., (2002). *Ingeniería de Software*. ISBN: 970-26-0206-8, Editorial Pearson Educación.

United Nations Economic Commission for Europe (UNECE) and International Federation of Robotics (IFR), (2005), "World Robotics - Statistics, Market Analysis, Forecasts, Case Studies and Profitability of Robot Investment", ISBN 92-1-1011000-05.

William E. F., (1994), "What is an Open Architecture Robot Controller?" IEEE Int. Symposium on Intelligent Control, Columbus, Ohio, USA, pp. 27-32.

Zaratian B. (1998). *Microsoft Visual C++ 6.0 Manual del Programador*. ISBN:84-481-2127-9, Editorial Mc Graw-Hill.