FORO DOCENTE DEL AREA MECANICA DE LAS INGENIERIAS
FoDAMI
UNIVERSIDAD NACIONAL DEL SUR BAHIA BLANCA

I CAIM 2008
Primer Congreso Argentino de Ingeniería Mecánica
Octubre 2008

NUEVA INTERFASE HOMBRE MÁQUINA PARA LA PLATAFORMA DEL ROBOT GANTRY- UBA

Andrés Mauro *[1], Ricardo M. Ramos *[2] y Mauricio Anigstein*[3]

Depto. de Ing. Mecánica – Facultad de Ingeniería. Universidad de Buenos Aires

Av. Paseo Colon 850, (1010) Buenos Aires, Argentina

e-mail: amauro, rramos, manigst@fi.uba.ar

RESUMEN

El estudio de la Robótica se incorporó no hace mucho a las carreras de grado de ingeniería mecánica y electrónica de la UBA. Para las tareas de enseñanza, investigación y desarrollo de aplicaciones, es de suma utilidad contar con manipuladores robóticos. Actualmente se dispone de un robot industrial de 6 ejes de la firma ABB + un robot para tareas planas construido en el Laboratorio de Robótica por alumnos y docentes investigadores de la Cátedra: el Gantry UBA de 3 ejes + 1 eje de dos posiciones. Se trata de equipamientos complementarios, necesarios para distintas actividades de investigación y capacitación de alumnos.

El robot Gantry actual, es el resultado de sucesivos proyectos finales de graduación relacionados. La idea es que siempre siga siendo una máquina en evolución, en la que en sucesivos trabajos se incorporen mejoras aprovechando que su controlador es abierto (a diferencia del robot industrial), por haber sido diseñado en el laboratorio.

En este artículo se describirán las características mecánicas y del sistema de control del robot Gantry. Con respecto a este último, la necesidad de controlar dispositivos electromecánicos con una PC, orientó el diseño hacia el uso del sistema operativo QNX. La utilización de lenguajes de bajo nivel como C y C++ bajo la plataforma QNX permitió desarrollar algoritmos de control en tiempo real.

En el último trabajo realizado se encaró el desarrollo de una nueva interfase hombre maquina. Con este objetivo, se realizaron mejoras en el lenguaje de programación del robot a nivel usuario y se utilizó el software Photon bajo QNX para incorporarle capacidades gráficas. Se desarrollaron aplicaciones con esta nueva interfase con resultados exitosos, que muestran que se facilitó la utilización y programación del robot por parte del usuario.

Palabras Claves: manipuladores robóticos, controlador, sistemas de tiempo real.

1. INTRODUCCIÓN

En el laboratorio de Robótica, se había desarrollado inicialmente en un ambiente de programación orientado a objetos C++, el control de ejes robóticos bajo el sistema operativo DOS [1]. Gracias a la reusabilidad y extensibilidad del lenguaje C++ se pudo implementar ya bajo la plataforma QNX [2] la construcción de clases más complejas que forman la plataforma actual del robot Gantry. QNX permitió la implementación de tiempos de muestreo del orden de los 5 mseg requeridos para procesar el algoritmo de control. Algunos trabajos recientes abonan el concepto usado en el diseño del Gantry. Chen *et al.* [3], realizan el control de un motor de corriente continua haciendo hincapié en la solución PC in the loop y la utilización del conocido software Matlab y Simulink como ambiente de trabajo para resolver los algoritmos de control del motor. Se destaca la no pérdida de tiempo en implementación del control en programación de más bajo nivel para el usuario del sistema. El sistema implementado por Chen *et al.* constituye una plataforma de aprendizaje para aplicar diferentes estrategias de control. Sin embargo, cuando el cálculo es altamente complejo como es el caso de un robot, donde dentro del control es necesario desarrollar la generación de trayectoria, la cinemática directa / inversa y resolver la dinámica del robot [4], esta solución se ve limitada porque los tiempos de cómputo son demasiado altos para poder hacer control en tiempo real. Por otro lado, la forma tradicional de realizar el control utiliza varios procesadores. Un sistema típico se compone de una computadora de propósito general que hace de host y una placa DSP donde se ejecutan los programas de control. Costescu *et al.* [5] hacen una descripción, discusión y selección del sistema operativo QNX para hacer control en tiempo real con una PC (PC in the loop) con una tarjeta MultiQ y confrontan la relación costo/perfomance de un sistema con dos procesadores (el primero un DSP sobre una PC, el segundo una PC usando QNX), concluyendo que es de menor costo realizar la implementación PC in the Loop a igual o mejor perfomance.

Con el objetivo de mejorar el diseño del controlador del robot Gantry UBA para que pueda ejecutarse desde el sistema de ventanas nativo de QNX, el Photon, y además mejorar el ambiente operativo para facilitar la programación, ejecución y monitoreo de las tareas, se desarrolló una nueva versión del software de control y su interfase gráfica de usuario.

El artículo se organiza de la siguiente manera. En la sección 2 se hace una descripción muy general del robot. En la sección 3 se describe el hardware. En la sección 4 se describe la arquitectura y el software de control detallando las mejoras incorporadas a la versión anterior. Finalmente, la sección 5 muestra su funcionamiento.

2. DESCRIPCIÓN GENERAL DEL ROBOT

Es un Robot tipo pórtico (Gantry), Figura 1, con 3 ejes servocontrolados que permiten posicionar y orientar la herramienta en forma arbitraria en un plano, dentro de un área de trabajo de 450x400 mm. Un cuarto eje permite el desplazamiento vertical (mediante un electroimán) de la herramienta, para acercarla y alejarla del plano de trabajo. Los actuadores son motores de corriente continua de imán

permanente con codificadores ópticos incrementales montados sobre sus ejes. El primer motor tiene además un tacogenerador.

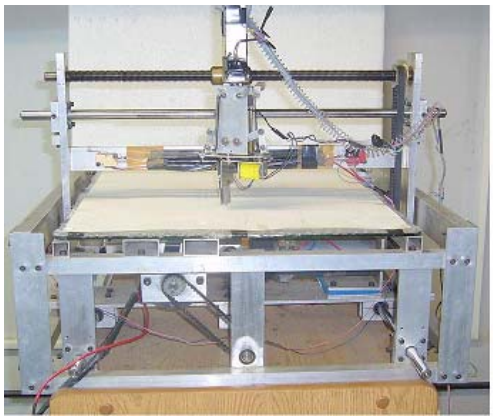

Figura 1 Fotografía del Robot Gantry UBA

Los dos primeros ejes son prismáticos, y permiten el desplazamiento del cabezal en un plano horizontal, mediante transmisiones motor-tornillo sinfín. El tercer eje es vertical y de rotación. Controla la orientación de la herramienta mediante un motor con un reductor planetario (1:200). La estructura base o mesa es una estructura prismática hueca de aluminio. La herramienta es una pinza cuya apertura está accionada con un solenoide. El cierre se logra con un resorte. La Figura 2 corresponde a los últimos dos eslabones del robot, donde puede observarse el montaje del solenoide para el desplazamiento vertical, un mecanismo de palancas para multiplicar la carrera de descenso y resortes para el ascenso, y el diseño de la pinza.

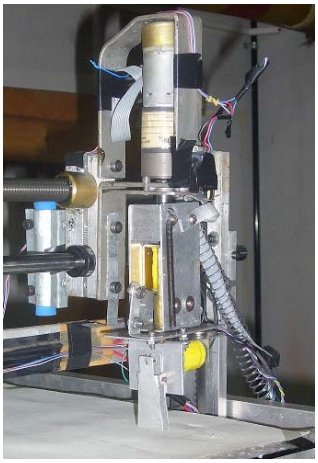

Figura 2 Los dos últimos eslabones del robot

3. DESCRIPCIÓN DEL HARDWARE

El hardware de control está compuesto por dos placas encargadas de adquirir los datos provenientes del robot. Estas tarjetas constituyen la interfase entre la PC y el robot Gantry [6]. En la Figura 3 puede verse el diagrama en bloques general del hardware.

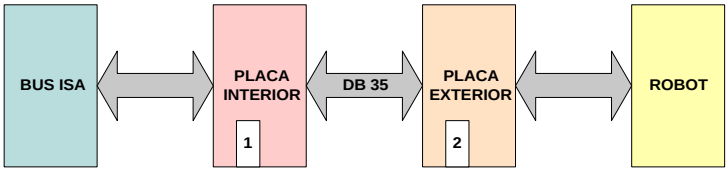

Figura 3 Diagrama en bloques general del hardware

La placa interior a la PC está conectada a un BUS ISA de 8 bits. Se encarga de controlar los conversores y contadores, leer y escribir los datos provenientes de la PC, escribir las salidas analógicas, leer las entradas y escribir las salidas digitales, y establecer la dirección donde la PC podrá trabajar con el robot. La placa exterior es la encargada de multiplexar las entradas y demultiplexar las salidas digitales. Contiene la etapa de potencia para el motor 3 y para el control de la pinza. Tiene un bloque de rectificación y regulación que genera las tensiones necesarias para su funcionamiento. También posee un bloque de atenuadores para las entradas analógicas.

4. ARQUITECTURA Y SOFTWARE DE CONTROL. MEJORAS

El controlador del robot es un ejemplo típico de un sistema de control en tiempo real. Su respuesta debe ser predecible ante eventos críticos. Debe asegurar los requerimientos de tiempo para las distintas tareas. Debe ser estable ante situaciones de sobrecarga y garantizar el cumplimiento de las tareas críticas. Debe presentar alta confiabilidad y ser adaptable ante cambios en la configuración, en las especificaciones o en el estado del sistema [7].

4.1. La solución elegida y el sistema operativo QNX

Los avances del hardware y la tecnología de software, han hecho de la PC una plataforma viable para la implementación de sistemas de control con un único procesador. Actualmente está disponible en computadoras de propósito general la capacidad de procesamiento necesaria para el cómputo de algoritmos de control complejos en tiempo real. Por otro lado, se seleccionó el sistema operativo QNX ya que cumple con los requerimientos necesarios para utilizarlo en sistemas de control en tiempo real [3,4]. Permite el desarrollo de aplicaciones distribuidas, basándose en una estructura de microkernel (10 Kb) sobre el cual se ejecutan un conjunto de tareas que otorgan servicios con compatibilidad POSIX y UNIX. Mediante la inclusión o exclusión de ciertos administradores de recursos, puede reducirse para ser usado en sistemas embebidos, o ampliado para usar muchos procesadores manteniendo transparencia.

4.2. Diseño y Mejoras

Para obtener las estructuras de datos y definir la arquitectura general del sistema, se realizó la división en tareas concurrentes utilizando DARTS (Design Approach for Real Time Systems) como

metodología de diseño [7]. La arquitectura de procesamiento de los controladores además de ser distribuida, se encuentra organizada en forma jerárquica [8].

El software de control anterior estaba distribuido en tres procesos: *Gentra, Tcontrol y Monitor*. El primero generaba la trayectoria a seguir en tiempo real calculando las sucesivas referencias para el control. El segundo realizaba el control propiamente dicho interactuando con el hardware del robot. El tercero monitoreaba los eventos de comunicación. Para utilizar el controlador se debían ejecutar los tres procesos en consolas distintas, siguiendo una secuencia determinada. Toda la información debía ingresarse con el teclado. Los datos del movimiento que se estaba realizando no eran visibles en forma gráfica, y para poder obtener datos de las trayectorias, éstas debían ser ejecutadas por el robot.

El actual diseño permite ejecutar los procesos necesarios en forma automática desde el Photon. Navegar el sistema de archivos para buscar los archivos de movimiento que pueden residir en cualquier directorio. Visualizar la trayectoria deseada gráficamente en pantalla sin necesidad de ejecutar el programa de movimiento. Visualizar gráficamente en pantalla y en tiempo real, la trayectoria real que el robot está ejecutando una vez que se lo pone en marcha y manejar la teach con el mouse. El software del sistema está compuesto por cinco procesos que realizan sus tareas de manera cooperativa y concurrente: *Setup*, *Phgentra*, *Phtcontrol*, *Phdibujo* y *phgantry*. El software del controlador fue desarrollado en C++, y en el caso de la interfase de usuario, se utilizó como herramienta de diseño y programación el *Photon Application Builder (PhAB)*. Se desarrollaron cuatro clases de objetos:

Program: incluye las funciones y variables encargadas de interpretar el pseudo código creado para la descripción de la tarea a realizar por el robot. Las instrucciones básicas son las siguientes:

MOVE(X,Y, Θ): permite realizar movimientos a las coordenadas indicadas.
STOP: movimiento a la última posición deseada.
COORD(X,Y, Θ): modifica la terna de referencia del robot
FRAME(P1,P2,P3): permite definir un nuevo sistema de referencia mediante la definición de tres punto. P1 y P2 definen la orientación del sistema de referencia y P3 el origen.
UP, DOWN: sube y baja la herramienta.
OPEN, TAKE: abre y cierra la pinza.
pmt(P,V): permite cambiar parámetros del controlador y del generador de trayectoria.
END: indica el fin del programa.

Interfaz: incluye todas las funciones para entrada y salida de datos. Se definen las direcciones de los periféricos y los ports correspondientes. Se programan los periféricos para el modo de funcionamiento requerido.

Servo: incluye la generación de trayectoria y los algoritmos de control de los motores. En este objeto se encuentran también las funciones que utiliza el módulo de enseñanza para los movimientos incrementales de los motores.

Rtimer: se encarga de administrar el tiempo. Su base de tiempo es de 1 milisegundo. Se utiliza para muestrear los datos y para el control.

5. FUNCIONAMIENTO

Para utilizar el robot con el nuevo controlador desde *Photon*, se debe ejecutar el archivo *phgantry*. Este archivo es el que se obtiene al compilar el proceso que maneja la interfaz gráfica de usuario, que se encarga de cargar y ejecutar los otros procesos. Durante la etapa de inicialización, el proceso *Setup* inicializa los valores de las placas de adquisición de datos y el Robot se mueve llevando la herramienta a una posición inicial (HOME) donde se detiene. Finalizada la etapa de inicialización, quedan los cuatro procesos ejecutándose concurrentemente. A continuación, se describe el funcionamiento del controlador utilizando como ejemplo el programa de movimiento guardado en el archivo *cuadrado.move*, que se transcribe

```
pmt(tacc,1.500)     // Setea tiempo de aceleración = 1.5 segundos
MOVE punto1         // Posición punto1 = (50,50,0)
MOVE punto1         // Repetir posición = STOP
MOVE(150,50,0)
MOVE(150,150,0)
STOP
pmt(tacc,0.300)     // Setea tiempo de aceleración = 0.3 segundos
MOVE(50,150,0)
MOVE punto1
STOP
MOVE home           // Posición de home
STOP
END
```

Cuando el archivo se abre para su ejecución, es leído por el proceso *Phgantry* para verificar la ausencia de errores y luego por el proceso *Phdibujo* para calcular la trayectoria deseada. Las sucesivas referencias que éste genera son enviadas a *Phgantry* para dibujar la trayectoria deseada en la interfase gráfica, Figura 4. El robot comienza a ejecutar el movimiento al seleccionar el botón *Marcha* de la barra de tareas. En la Figura 5 puede observarse como, mientras el robot ejecuta el movimiento, el software va dibujando, en tiempo de ejecución, la trayectoria real (color azul).

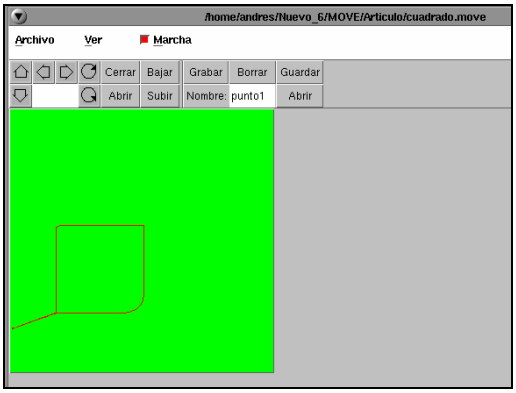

Figura 4 Dibujo de la trayectoria deseada para el archivo cuadrado.move

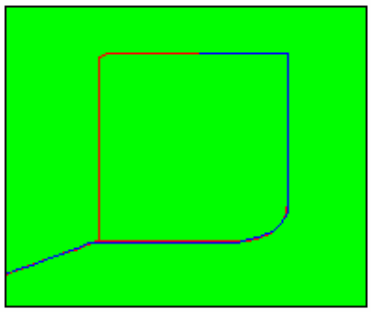

Figura 5 Dibujo de la trayectoria real que el robot ejecuta

En la Figura 6 se pude ver una fotografía del robot ejecutando el movimiento.

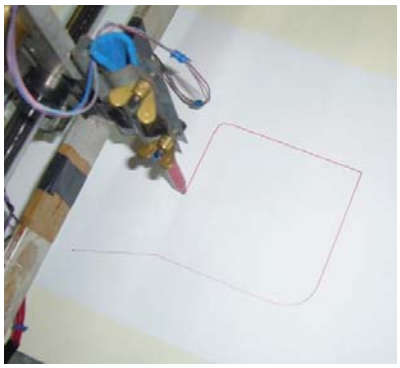

Figura 6 Fotografía del robot ejecutando el movimiento

Los nombres **punto1** y **home** que se utilizan con algunas instrucciones **MOVE** del código *cuadrado.move*, identifican posiciones que fueron enseñadas previamente utilizando el módulo de enseñanza que se muestra en la Figura 7, y grabadas en memoria.

Figura 7 Módulo de enseñanza y grabación

Este módulo permite mover los ejes del robot en forma independiente estableciendo el valor del desplazamiento deseado en el cuadro de texto y seleccionando el botón correspondiente al eje que se quiera mover. También se puede bajar o subir el cabezal, y abrir o cerrar la pinza. Se pueden ver también los botones que permiten asociar nombres a posiciones, grabarlas y guardarlas en un archivo de posiciones para ser usadas luego en un archivo de movimiento. El proceso Phgentra, a partir de la lectura de un archivo de movimiento, o a través de los datos ingresados por el operador desde el módulo se enseñanza, genera la trayectoria deseada para obtener las sucesivas referencias con las cuales realizar el control. La función *control.trayecto()* pertenece a la clase servo. Esta función lee y decodifica la instrucción en curso, resuelve el problema inverso y calcula el tiempo de movimiento, como se muestra en el siguiente fragmento del código que decodifica la instrucción MOVE(X,Y, Θ).

```
prog->leer_inst();                              // Lee y decodifica la instrucción
switch( prog->tipo_inst )
{
      case 'M':                                 // MOVE (x,y,z)
      mensaje.pos_vel.instruccion = MOVE;

      if( prog->tflag == 0 )            // tflag sirve para hacer move a posiciones almacenadas
                                        // en memoria con el módulo de enseñanza.
      {
            aux1 = cos(coord.z);
            aux2 = sin(coord.z);
      //*************************************************************************************//
      //          Resuelve el problema inverso donde:                                        //
      //                                    a: Posición donde se encuentra                   //
      //                                    b: Posición a la que se dirigía                  //
      //                                    c: Nueva posición a la que se debe mover         //
      //          y calcula el tiempo de movimiento                                          //
      //*************************************************************************************//
      // Eje 3
            posi[3] = a3 = lazo3.posd;
            b3 = c3;
            posf[3] = c3 = coord.z + prog->pz;
      // Eje 1
            posi[1] = a1 = lazo1.posd;
            b1 = c1;
            posf[1] = c1 = aux1*prog->px - aux2*prog->py + coord.x + radio*sin(c3);
      // Eje 2
            posi[2] = a2 = lazo2.posd;
            b2 = c2;
            posf[2] = c2 = aux2*prog->px + aux1*prog->py + coord.y - radio*cos(c3);
            t1 = calc_t1();             // Calcula el tiempo del movimiento
            control_treal();
}
```

Para realizar las trayectorias deseadas en el plano, los ejes deben moverse al mismo tiempo en forma sincronizada. El tiempo en el que se realizará el movimiento, t1, lo calcula la función *calc_t1()*. El tiempo de aceleración (tacc) es el que se necesita para cambiar el sentido de rotación de los actuadores girando a máxima velocidad, sin superar los torques máximos. Se fija una cota mínima para este tiempo. La función *control_treal()* también pertenece a la clase servo y realiza varias funciones: controla los tiempos, invoca al interpolador, arma y envía el mensaje al proceso *Phtcontrol*:

```
t = -tacc;
while( t < (t1-tacc) )                  // t = tiempo transcurrido
{                                       // t1 = tiempo para realizar el movimiento
      t2->run();                        // Corre un timer seteado en 7 mseg.
                                        // Cuando expira envía un mensaje tipo
                                        // proxy.
      interpolador(t,t1,tacc,a1,b1,c1);
      mensaje.pos_vel.tiempo = t;
      mensaje.pos_vel.posd1 = lazo1.posd = posd;
      mensaje.pos_vel.veld1 = lazo1.veld = veld;
      interpolador(t,t1,tacc,a2,b2,c2);
      mensaje.pos_vel.veld2 = lazo2.veld = veld;
      mensaje.pos_vel.posd2 = lazo2.posd = posd;
      interpolador(t,t1,tacc,a3,b3,c3);
      mensaje.pos_vel.veld3 = lazo3.veld = veld;
      mensaje.pos_vel.posd3 = lazo3.posd = posd;
      mandar_mensaje( 0, mensaje, 0);   // Manda el mensaje con la posición deseada
                                        // al control de GANTRY.
      t += ts;                          // Suma un tiempo de muestreo par la próximo
                                        // secuencia ts = 7ms
      while( !t2->stop() );             // Espera a que se concluya el tiempo de
                                        // muestreo de la secuencia. Espera la recepción del proxy.
}
```

Mediante el algoritmo desarrollado, la herramienta del robot no alcanzará en general las posiciones indicadas como destino. Si se desea que el robot pase exactamente por una determinada posición se debe repetir la posición como se hizo en las líneas 2 y 3 del código *cuadrado.move*. El resultado se pude observar en las Figuras 4, 5 y 6. La instrucción STOP trabaja de la misma manera. Obsérvese que al aumentar el tiempo de aceleración, aumenta la distancia entre la posición dada como destino y la posición por donde el robot pasa realmente. El proceso *Phtcontrol* cada 7ms recibe las referencias generadas por el proceso *Phgentra* y realiza el control interactuando con el hardware del sistema.

```
 while( control.mensaje.tipo != SALIR )
{
      pid = Creceive( 0, &control.mensaje, sizeof( control.mensaje ) );
      if( pid != -1 )               // Hay mensaje de movimiento
      {
            posd1 = control.mensaje.pos_vel.posd1;
            veld1 = control.mensaje.pos_vel.veld1;
            posd2 = control.mensaje.pos_vel.posd2;
            veld2 = control.mensaje.pos_vel.veld2;
            posd3 = control.mensaje.pos_vel.posd3;
            Reply( pid, &(control.reply), sizeof(control.reply) );
            timer.run();
            control.pos();          // Determina la posición real de cada eje
            controlar();
            while( !timer.stop() );
      }
      else                          // No hay mensaje de movimiento
      {
            timer.run();
            control.pos();
            controlar();
            while( !timer.stop() );
      }

}
```

Debido a que la primitiva *Creceive(pid_t pid, void_far *msg, unsigned nbytes)* es no bloqueante, si el tipo de mensaje recibido es de movimiento, toma las nuevas referencias para realizar el control,

mientras que, si no se reciben referencias, se sigue realizando el control del robot con las últimas referencias recibidas, con lo cual el robot permanece detenido en la última posición alcanzada. La función c*ontrol.pos()* pertenece a la clase servo y es la encargada de acceder al hardware para leer los contadores y determinar la posición real de cada eje, y la función *controlar()* determina, con las posiciones deseadas recibidas del generador de trayectoria y las reales calculadas por *control.pos()*, los errores en cada uno de los ejes, calcula las actuaciones y las aplica accediendo al hardware para escribir los DACs.

6. CONCLUSIONES

Con la disponibilidad de sistemas operativos de tiempo real diseñados para correr sobre el hardware de una PC, se logró garantizar que las soluciones basadas en estos sistemas también tuvieran una respuesta igual o mejor que los sistemas multiprocesador, siendo además, más flexibles, menos complejas y costosos, y más fáciles de administrar y mantener.

REFERENCIAS

[1] Ramos R., Yamamoto J., Rendo F. y Anigstein M. (1994). "Interfase de programación orientada a objetos para el control de ejes robóticos", *XIV Simposio Nacional de Control Automático. AADECA*, Bs As.

[2] R. Ramos, C. Costas, C. S. Kang, D. S. Son y M. Anigstein, "Procesamiento Distribuido en tiempo Real. Aplicación a un controlador robotico.", Rpic ´97 San Juan, 224-229 1997

[3] Y. Chen and J. Naugton, "An under graduate Laboratory Platform for Control System Design simulation and implementation", *Control Systems Magazine June 2000*.

[4] R. P. Paul, *Robot Manipulators: Mathematics, Programming, and Control*., The MIT Press. Cambridge, Massachussets, 1981.

[5] N. Costescu, D. Dawson and M Loffler, *Qmotor 2.0 A real time PC Base Control Environment*, Application Note Control Systems Magazine, Vol 19, number 3, June 1999.

[6] Ricardo Ramos, E. Panciera, M. Lehmann, Hardware de Control para la Plataforma del Robot Gantry UBA, *XVIII Congr. Argentino Control Automático*, AADECA 2002, # 6, Buenos Aires, 2002

[7] Gabriel A. Wainer, "Sistemas de Tiempo Real, Conceptos y Aplicaciones", *NUEVA LIBRERÍA.* ISBN 950-9088-86-2, 1997.

[8] W. E. Snyder. *Industrial Robots. Computer Interfacing and Control.* Prentice-Hall, N. J., 1985.