*II CAIM 2010*
*Segundo Congreso Argentino de Ingeniería Mecánica*
*San Juan - Noviembre 2010*

# SOBRE LOS SISTEMAS DE TIEMPO REAL Y EL CUMPLIMIENTO DE LOS TIEMPOS DE LAS TAREAS EN UN MANIPULADOR ROBÓTICO

Andrés Mauro [*1] y Mauricio Anigstein [2]

***Laboratorio de Robótica – Departamentos de Ingeniería Mecánica y de Electrónica***
***Facultad de Ingeniería – Universidad de Buenos Aires***
***Av. Paseo Colón 850 (1063) Buenos Aires, Argentina***
*- E-mail: amauro, manigst@fi.uba.ar*

**RESUMEN**

Los sistemas de Tiempo Real se utilizan en una enorme variedad de aplicaciones. Control de tráfico aéreo, señalización y seguimiento de ferrocarriles, centrales nucleares, distribución de energía eléctrica, redes telefónicas, pilotos automáticos, vehículos espaciales, cadenas de producción, control de manipuladores robóticos y máquinas-herramienta, son sólo algunos ejemplos. Este tipo de sistemas debe interactuar con los usuarios, el medio y responder a los diversos eventos externos asegurando un tiempo de respuesta máximo. La exactitud de los resultados obtenidos es tan importante como el tiempo que se tarda en obtenerlos.

El Laboratorio de Robótica de la FIUBA cuenta con un robot para tareas planas, desarrollado para enseñanza e investigación por alumnos y docentes de la cátedra: el Gantry UBA de 3 ejes + 1 eje de dos posiciones. El control del Gantry está distribuido en varios procesos organizados jerárquicamente que realizan sus tareas de manera cooperativa y concurrente utilizando un único procesador y bajo la plataforma del sistema operativo de tiempo real QNX.
Un proceso clave del sistema, es el que genera la trayectoria que debe realizar la herramienta. Este proceso calcula y actualiza en tiempo real, las referencias que debe seguir otro proceso: el que controla el movimiento de cada eje. El resultado de la cooperación entre procesos debe permitir ejecutar trayectorias no sólo con la forma deseada, sino también en el tiempo deseado. La organización del software correspondiente debe asegurar que los tiempos de ejecución de las tareas efectivamente se cumplan. Sin embargo, la utilización de un sistema de tiempo real es condición necesaria pero no suficiente para lograrlo. En el artículo se analiza este tema, se muestran los errores encontrados en la estructura del software de control anterior y cómo estos errores afectaban el correcto funcionamiento del sistema. Finalmente se explican las modificaciones realizadas para repararlo.

**Palabras Claves:** robots, sistemas de tiempo real, generación de trayectoria, control.

## 1. INTRODUCCIÓN

El robot Gantry UBA fue construido en el Laboratorio de Robótica por alumnos y docentes investigadores de la cátedra [1], [2], [3], [4] y es el resultado de sucesivas tesinas de graduación relacionadas. Es un Robot tipo pórtico (Gantry), con 3 ejes servocontrolados que permiten posicionar y orientar la herramienta en forma arbitraria en un plano. Un cuarto eje de dos posiciones permite el desplazamiento vertical de la herramienta, que consiste en una pinza accionada por un solenoide que puede tomar pequeñas piezas. Relatar resumidamente la evolución del controlador del Gantry UBA será útil para motivar el objetivo de este trabajo y resaltar una capacidad clave que debe tener un robot.

Inicialmente se había desarrollado el control de sus ejes bajo el sistema operativo DOS utilizando el lenguaje de programación orientado a objetos C++ [1]. En esta versión del controlador primero se calculaban todas las referencias que componían la totalidad de la trayectoria y se las guardaba en un archivo. El tamaño del archivo que contenía las referencias estaba limitado a la cantidad de memoria disponible en la PC. Para ejecutar el movimiento, se leía este archivo y se enviaban las sucesivas referencias al control. Es decir, la generación de las trayectorias no se realizaba en tiempo real de ejecución. Esta forma de trabajo hace imposible la interacción del manipulador con el ambiente de trabajo, reduce la capacidad de ejecución de movimientos, restándole flexibilidad y versatilidad, y pierde la característica clave de un manipulador robótico.

La ejecución de una tarea involucra el seguimiento de una trayectoria cuya generación usualmente está determinada por eventos externos que pueden provenir de sensores, dispositivos u otras máquinas que formen parte del ambiente de trabajo con los cuales el robot debe interactuar. La ocurrencia de estos eventos no se puede predecir por lo tanto las trayectorias no pueden ser planificadas sino que deben ser generadas en tiempo real a medida que se ejecuta el movimiento. Por otro lado, el controlador debe asegurar los ***tiempos*** de ejecución de las distintas tareas para poder realizar los movimientos recorriendo las trayectorias en forma y tiempo, cumpliendo con los requerimientos tanto de posición como de velocidad. El DOS no es un sistema operativo adecuado para ser utilizado bajo estas exigencias [5], por lo que la solución implementada se adecuó a los medios y recursos disponibles en aquel momento.

Para mejorar su funcionamiento, se implementó una segunda versión del controlador, esta vez utilizando como software de base el sistema operativo de tiempo real QNX [6]. Costescu et al. [7] 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. QNX permitió desarrollar un controlador en tiempo real, distribuido en varios procesos organizados jerárquicamente que realizan sus tareas de manera cooperativa y concurrente utilizando un único procesador. Para que el controlador del robot Gantry pudiera ejecutarse desde el sistema de ventanas nativo de QNX, el Photon, y para mejorar el ambiente operativo facilitando la programación, ejecución y monitoreo de las tareas, se desarrolló una tercera versión del software de control que le agregó capacidad gráfica a través de una interfase hombre máquina [4].

### 1.1. Objetivo del artículo

Mediciones recientes mostraron que, aún trabajando con un sistema operativo de tiempo real, el sistema de control presentaba fallas en el cumplimiento de los ***tiempos*** de los movimientos. El objetivo de este artículo es el de analizar las causas de estos errores y las modificaciones hechas para corregirlos. En la sección 2 se expone la teoría que se aplicó para construir el generador de trayectoria. En la sección 3 se describe la estructura general del software del sistema. En la cuarta sección se muestran las mediciones realizadas y las fallas encontradas; además se discute y explica por qué el controlador del Gantry, a pesar de estar trabajando con un sistema operativo adecuado, presentaba estas fallas en su funcionamiento. En la quinta sección se explica las modificaciones realizadas y se muestran los nuevos resultados obtenidos.

## 2. GENERACIÓN DE TRAYECTORIA

En este apartado se explica el método utilizado en el robot Gantry para generar las trayectorias que describen el movimiento de la herramienta. La generación automática de la trayectoria hace la descripción del movimiento fácil para el usuario, dejando que el sistema realice los cálculos necesarios [8]. Para programar una tarea el operador especifica una secuencia de objetivos (posición y orientación) a los cuales la herramienta del robot deberá acercarse. Para pasar exactamente por un objetivo deberá detenerse en el mismo [9]. El algoritmo generador de trayectoria calcula los puntos intermedios a seguir entre los objetivos ingresados por el operador. Estos puntos intermedios son las ***referencias*** para el algoritmo que realiza el proceso de control de los actuadores de los ejes.

La Figura 1 muestra un conjunto de posiciones en el espacio de coordenadas para una variable genérica θ en función del tiempo, unidas por una función lineal por tramos. Por la estructura cinemática del Gantry, linealidad en el espacio de los ejes significa linealidad en el espacio cartesiano y no existen singularidades. En cada uno de los segmentos la velocidad es constante, sin embargo los cambios bruscos de velocidad que se muestran no son admisibles, requerirían aceleraciones (torques en los actuadores) infinitas.

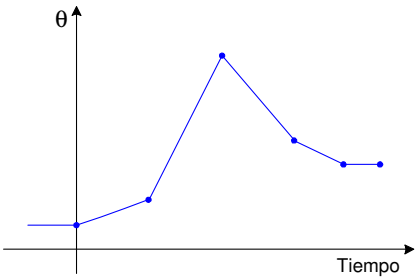



*Figura 1 Función lineal por tramos*

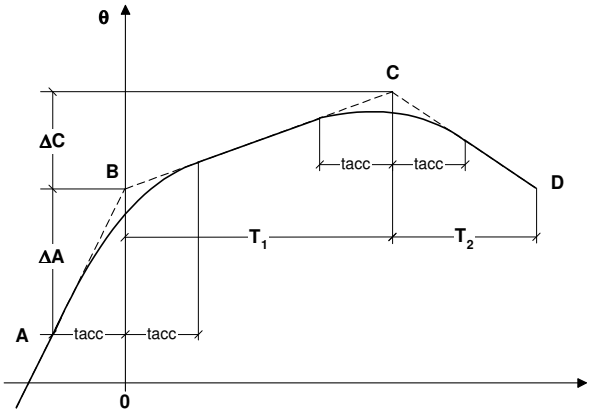



*Figura 2 Transición al siguiente segmento*

Con el fin de garantizar la continuidad en la posición y la velocidad, los sucesivos segmentos lineales de la trayectoria se unen definiendo zonas alrededor de cada uno de los puntos de transición, en las cuales la posición será una función polinómica del tiempo, ya que dentro de estas zonas la aceleración no será nula. Hay que tener en cuenta el tiempo necesario para acelerar al manipulador desde el reposo hasta alcanzar su máxima velocidad, tacc. Las zonas de aceleración tendrán una duración de 2tacc. En un instante de tiempo tacc anterior al tiempo necesario para alcanzar el punto B, el manipulador estará moviéndose a

velocidad constante y pasando por el punto definido como A, como se muestra en la Figura 2. Recién en ese instante es necesario conocer las coordenadas del punto C para empezar a evaluar el movimiento desde el punto B hacia el C. Es también necesario que el robot mueva todos sus ejes al mismo tiempo y que todos ellos lleguen simultáneamente a la posición de destino, es decir, que el movimiento sea coordinado. Por lo tanto, conociendo el valor de las coordenadas de cada eje (i) correspondientes a los puntos B y C, y la máxima velocidad permitida de cada eje, se calcula $T_1$ ***= tiempo total para recorrer el segmento*** B – C como el máximo valor entre: los tiempos que tarda cada uno de los ejes en recorrer el segmento, dos veces el tiempo de aceleración y eventualmente un tiempo proporcionado por el usuario (que requiera el proceso):

$$T_1 = \max\left\{ \frac{|\theta_{Ci} - \theta_{Bi}|}{v_{i\,max}} \; ; 2\,t_{acc} \; ; T_{usuario} \right\} \qquad (1)$$

Obsérvese que no es necesario conocer las coordenadas del punto objetivo siguiente D, hasta tanto el tiempo transcurrido t sea igual a $T_1$-$t_{acc}$. Cuando esto ocurre, se inicia la transición al siguiente segmento calculando T2 con la Ecuación (1) (***tiempo total para recorrer el segmento*** C – D), y haciendo las siguientes asignaciones:

T1 = T2
A = X  Posición actual.
B = C
t = -tacc  Resetear el tiempo.

**3. ARQUITECTURA DEL SOFTWARE DE CONTROL**

El software del sistema 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) [10]. Está distribuido en cinco procesos organizados jerárquicamente, que realizan sus tareas de manera cooperativa y concurrente. Dos de ellos son esenciales para el funcionamiento del robot: el proceso que genera en tiempo real la trayectoria que debe realizar la herramienta, y el proceso encargado de realizar el control de los ejes interactuando con el hardware. El resto realiza tareas auxiliares de configuración, monitoreo y supervisión.

El proceso generador de trayectoria lee e interpreta una instrucción, resuelve el problema inverso, calcula el tiempo $T_1$ necesario para alcanzar la siguiente posición y genera las sucesivas referencias que serán enviadas al proceso que realiza el control. Para que el sistema funcione en tiempo real, se debe asegurar el cumplimento de las metas de tiempo para las diferentes tareas. En particular, el generador de trayectoria no sólo debe generar una nueva referencia a intervalos regulares de tiempo, sino que debe garantizar que esas referencias se correspondan con el tiempo realmente transcurrido. Finalmente, al cabo de un tiempo t igual a $T_1$ – tacc, debe ir en busca de la siguiente instrucción y así repetir el ciclo hasta la última instrucción.

El proceso de control, recibe las referencias del generador de trayectoria, las compara con las posiciones reales de los ejes leyendo los sensores de posición y calcula y envía la tensión de control para los actuadores. Este proceso debe funcionar en todo momento, aun cuando el robot no se mueva (no reciba nuevas referencias del generador de trayectoria). Debe rechazar ruidos, perturbaciones externas sobre el manipulador y eventualmente esfuerzos provocados por el peso propio. Su objetivo no es cumplir tiempos de movimiento sino seguir las referencias del generador ***asegurando la estabilidad del sistema.***

## 4. MEDICIONES REALIZADAS. FALLAS

Uno de los parámetros fundamentales para el funcionamiento del controlador, es el tiempo de muestreo, $t_s$. En las primeras versiones, se fijó en 5 milisegundos tanto para el generador de trayectoria como para el control. Posteriormente se incrementó a 10 ms. sólo para el proceso de generación, ya que sin afectar la calidad de la trayectoria real se descarga la tarea del controlador. En cualquier caso, la organización y los algoritmos básicos de los procesos de generación de trayectoria y de control no fueron modificados.

### 4.1 Ensayos

Con el objetivo de verificar el correcto funcionamiento del sistema se realizaron varios ensayos ejecutando una serie de archivos de movimiento. Manteniendo el tiempo de muestreo del generador en 10 ms, y mientras el robot ejecutaba cada uno de estos archivos de movimiento, se fueron midiendo los tiempos que tardaba el generador de trayectoria en calcular cada una de las ***referencias***, es decir, el tiempo real de muestreo. También se midieron los tiempos reales totales que tardaba el robot en realizar los movimientos y se los comparó con los tiempos calculados por el generador de trayectoria según el algoritmo implementado. Se muestran los resultados obtenidos tomando como ejemplo la realización del siguiente archivo de movimiento:

```
MOVE(40,40,0)          // X = 40, Y = 40, θ = 0
MOVE(140,40,0)
MOVE(90,126,0)
MOVE(40,40,0)
STOP
END
```

La instrucción MOVE mueve la herramienta a las coordenadas indicadas. La herramienta del robot no alcanzará en general las posiciones indicadas como destino. Para que la herramienta pase exactamente por una posición determinada se debe ejecutar una instrucción MOVE repitiendo la posición anterior, o utilizar la instrucción STOP, que trabaja de la misma manera. La Figura 4 muestra la trayectoria generada para los ejes x e y del robot al ejecutar este movimiento. La herramienta del robot se mueve siguiendo los lados de un triángulo sin pasar exactamente por los puntos intermedios, deteniéndose al alcanzar la última posición. El gráfico del movimiento de cada uno de los ejes se muestra en la Figura 5. Este ensayo se repitió 10 veces para la misma trayectoria y el resultado del Tiempo total medido = 21.60 s. resulta del promedio de las mediciones con diferencias máximas del orden de 50 ms. En tanto el tiempo teórico calculado por el generador de trayectoria es de 19.10 s. El robot siempre tarda un tiempo mayor que el esperado. La Figura 6 muestra en ordenadas la evolución de los tiempos reales de muestreo durante un ensayo de movimiento. Puede verse que el valor real del tiempo de muestreo, además de variar entre muestras, siempre es mayor al valor establecido de 10 ms. En un sistema de tiempo real esto no debe ocurrir, ya que significa que no se están cumpliendo con las metas de tiempo de las tareas.

### 4.2. Mecanismo de comunicación

Para explicar los motivos por los cuales se produce este error hay que estudiar la estructura del proceso que genera la trayectoria y del proceso que realiza el control, analizando la forma en la cual estos dos procesos se comunican e interactúan entre si.

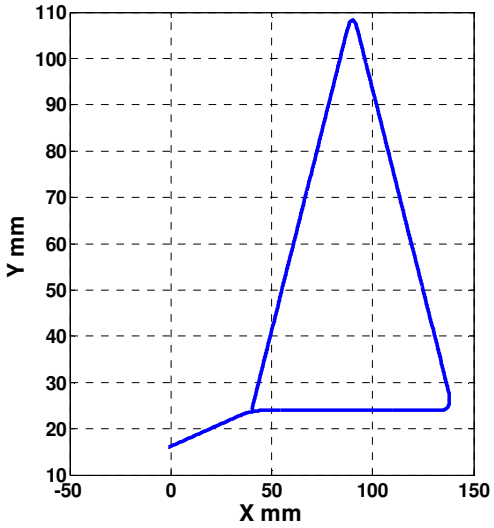



*Figura 4 Trayectoria generada*

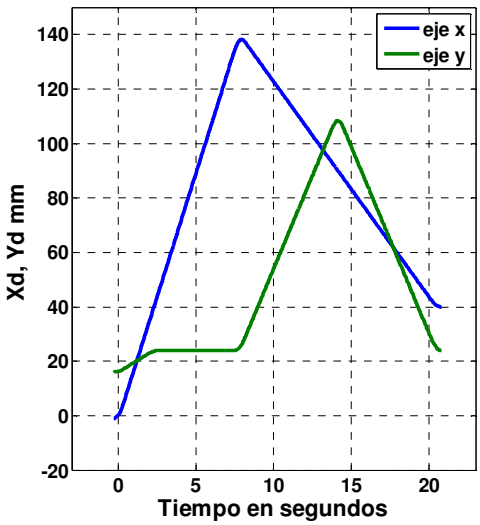



*Figura 5 Evolución de los ejes en función del tiempo*

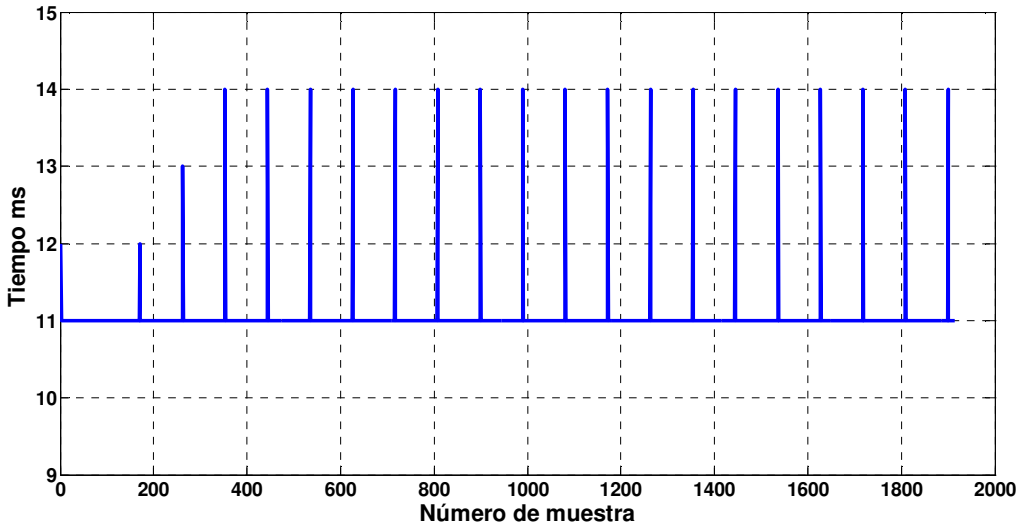



*Figura 6 Evolución del Tiempo de muestreo durante el movimiento*

El mecanismo utilizado para la comunicación entre estos procesos es a través del envío y recepción de mensajes. Un mensaje es un conjunto de bytes que son transmitidos de un proceso a otro. El sistema operativo provee primitivas para manejo de mensajes:

- *Send()*: para el envío de mensajes.
- *Receive()*: para la recepción de mensajes.
- *Reply()*: para enviar una respuestas a los procesos que han enviado los mensajes.
- *Creceive()*: para la recepción condicional de mensajes.

Las primeras tres primitivas además de comunicar información permiten la sincronización y la planificación de los distintos procesos debido a que van cambiando de estado con cada llamada, como puede verse en el diagrama de transición de estados que muestra la Figura 7. En QNX [6] todos los procesos tienen asignada

una prioridad. El planificador de QNX [6] selecciona el siguiente proceso a ejecutarse examinando las prioridades asignadas a cada uno de los procesos en estado READY y decide que proceso se ejecutará cuando:

- Un proceso se desbloquea (vuelve al estado READY, ver Figura 7).
- Finaliza el *timeslice* del proceso en ejecución. Este tiempo es de 100 milisegundos.
- El proceso en ejecución es desalojado por otro proceso de mayor prioridad en estado READY.

Para que un proceso pueda ejecutarse debe estar en estado READY. La cuarta primitiva (*Creceive()*) es ***no bloqueante*** ya que su llamada no produce el bloqueo del proceso que la invoca.

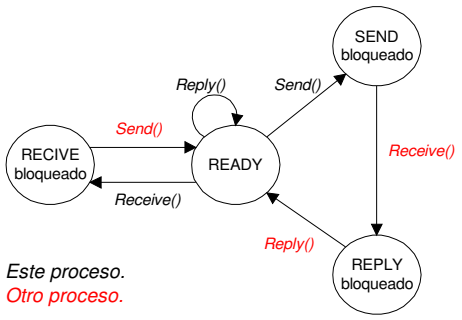



*Figura 7 Diagrama de transición de estados*

### 4.3. Causas de los errores en el cumplimiento de los tiempos

La Figura 8 muestra el diagrama de flujo del proceso generador de trayectoria junto con la secuencia de estados a través de los cuales va atravesando a medida que se ejecuta. La Figura 9 representa la secuencia de ejecución de operaciones del proceso que realiza el control. El concepto central que se utilizó en esa organización, fue tratar de asegurar el tiempo total del movimiento intentando asegurar el tiempo de muestreo en el cálculo de cada referencia. Sin embargo, como surge de las mediciones que mostramos, ese objetivo no se cumple.

Las variaciones de mayor influencia en el comportamiento del sistema se producen cuando el generador de trayectoria ejecuta el ciclo indicado como lazo 2 en la Figura 8. Cada vez que el generador de trayectoria envía las referencias al proceso de control utiliza la primitiva *Send()* y pasa al estado *SEND-bloqueado* (ver Figura 7). Cuando el proceso de control espera a que finalice su temporizador para comenzar un nuevo ciclo, utiliza la primitiva *Creceive()* que al ser no bloqueante, hace que consuma tiempo de procesamiento impidiendo que se ejecuten otros procesos. Esto produce una demora en la ejecución del proceso generador de trayectoria que no le permite alcanzar la meta de tiempo establecida por el período de muestreo. También se puede ver en las Figuras 8 y 9 que ambos procesos inicializan los temporizadores tardíamente. En el generador de trayectoria este temporizador no incluye el tiempo de ejecución del lazo 1, que se suma al tiempo de muestreo cada vez que va a buscar una nueva instrucción. Además, como el tiempo de lectura y de decodificación depende de la instrucción que se ejecute, también la duración de este lazo varía con la instrucción. En el control, el temporizador utilizado para esperar las sucesivas muestras no incluye la operación de actualización de las referencias, la comunicación con otros procesos del sistema ni el envío de la respuesta al generador.

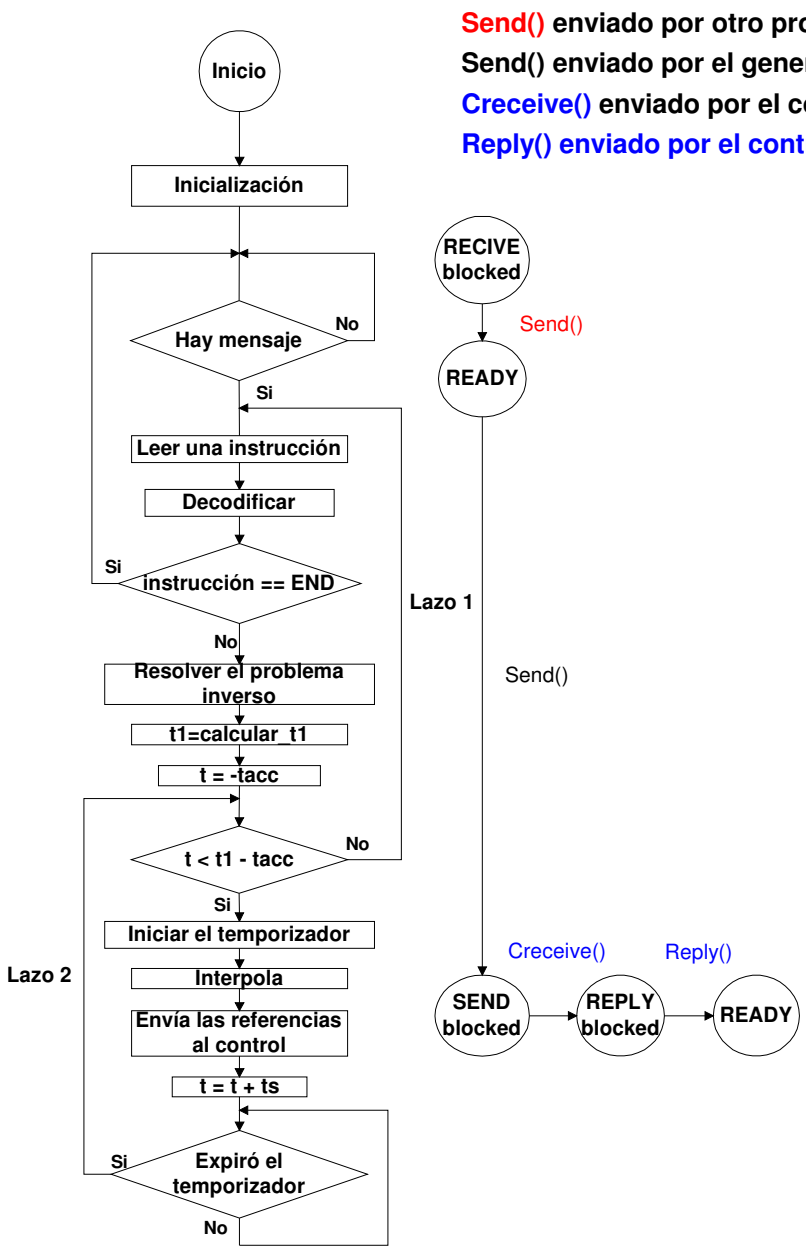



*Figura 8 Diagrama de flujo del generador de trayectoria*

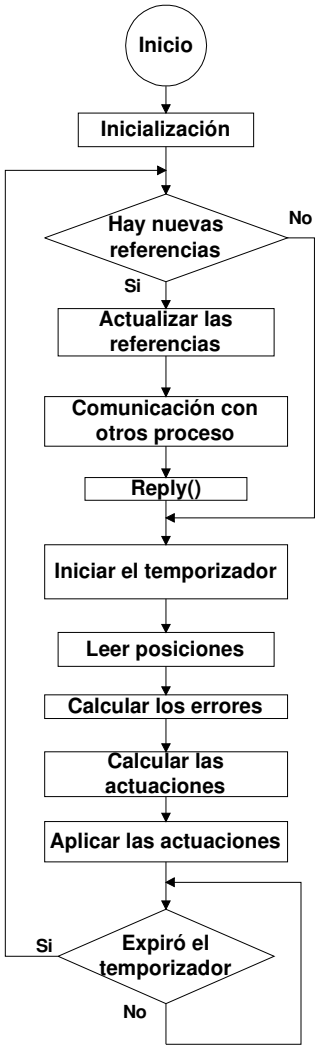
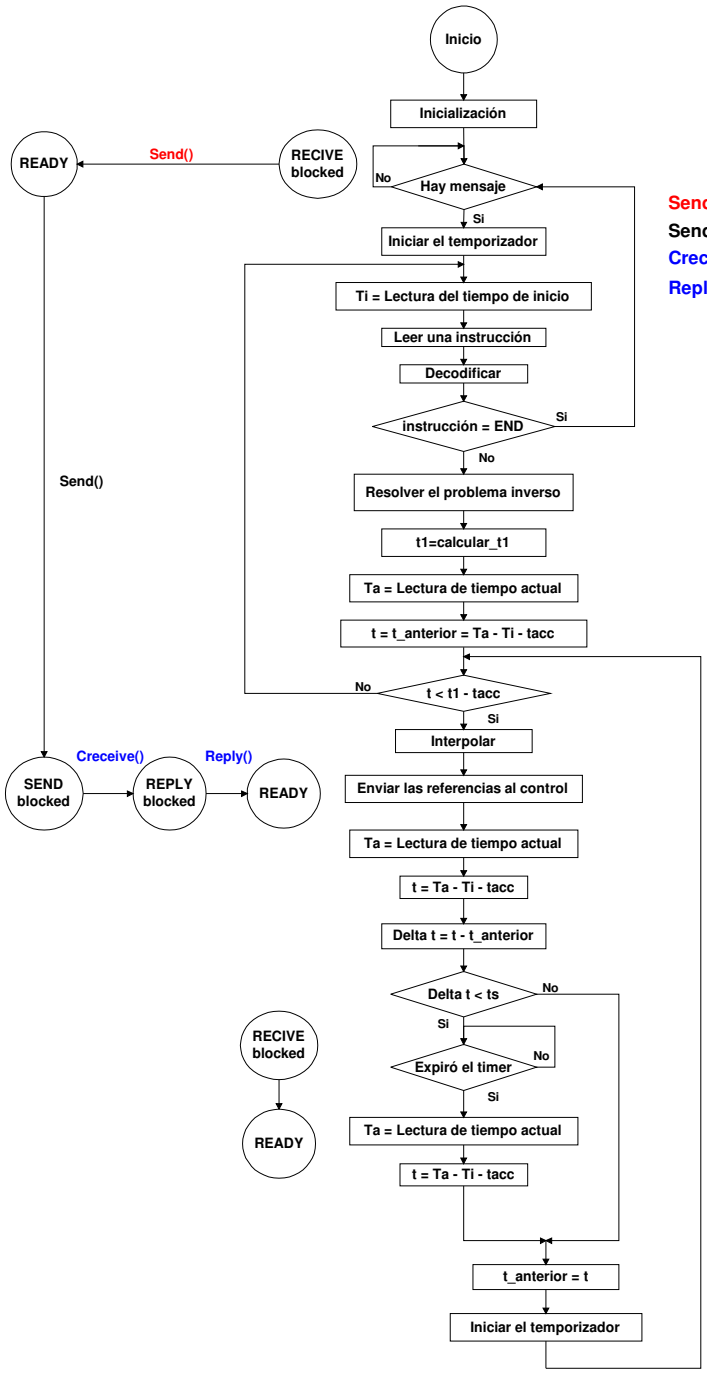



*Figura 9 Diagrama de flujo del control*

## 4. MODIFICACIONES REALIZADAS. MEDICIONES.




**Send()** enviado por otro proceso.
**Send()** enviado por el generador de trayectoria.
**Creceive()** enviado por el control.
**Reply()** enviado por el control.

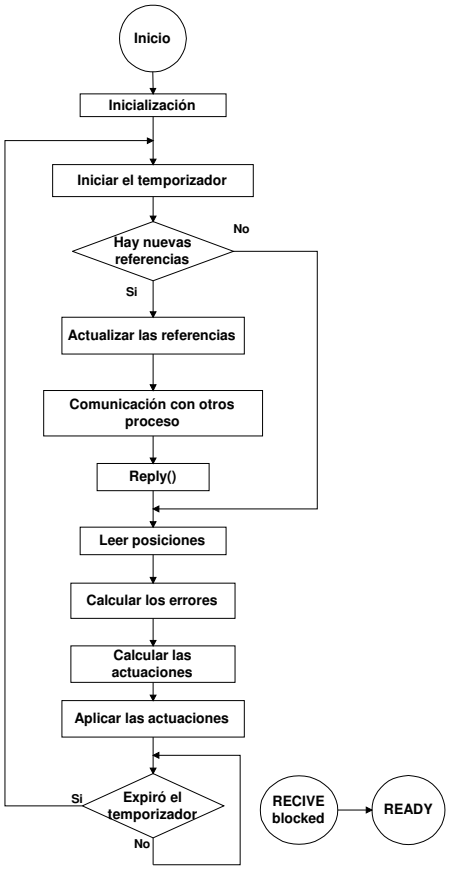



*Figura 10 Diagrama de flujo para el generador de trayectoria*

*Figura 11 Diagrama de flujo para el control*

Los cambios fundamentales realizados fueron:

- Como se observa en los diagramas de flujo (Figura 10 y 11):
  1. Se incorporó en el proceso de generación de trayectoria el control del tiempo total durante la ejecución del movimiento. Si el tiempo en un ciclo supera al establecido para cada intervalo de muestreo, el generador de trayectoria corrige la demora ya que calcula la nueva referencia utilizando el tiempo real transcurrido.
  2. Se modificó el orden en la secuencia de operaciones del generador de trayectoria y del control, para que sus respectivos temporizadores abarquen todas las tareas que realizan.
- Para que las esperas de los temporizadores se hagan con los procesos bloqueados de forma tal que liberen la CPU, se utilizó la primitiva *Receive()* en lugar de *Creceive().*
- Se incrementó la prioridad del generador de trayectoria para asegurar que sea seleccionado cada vez que esté en estado READY. Entonces, siempre que el robot se mueva, es el generador de trayectoria el que fija los tiempos. Como se mencionó en la sección 3, la función del temporizador en el proceso de control es la de establecer un período de muestreo para el control de los ejes que asegure estabilidad, aún cuando el robot no esté realizando ningún movimiento (el generador no se ejecuta).

A continuación se muestran los resultados obtenidos al realizar nuevamente la trayectoria tomada como ejemplo, para evaluar el funcionamiento del sistema con las modificaciones. En la Figura 12 se observa la evolución real de los ejes. Ahora, el tiempo total medido al realizar el movimiento fue igual al teórico, 19.10s. En la Figura 13 se muestra que el intervalo de muestreo es ahora de 10 ms. (salvo en 10 muestras sobre un total de 1908). De todas maneras, la ocurrencia de algunos muestreos con una duración mayor, no afecta el tiempo del movimiento, porque con la nueva organización esta demora se compensa reduciendo el número de referencias enviadas al control y manteniendo así el tiempo total del movimiento.

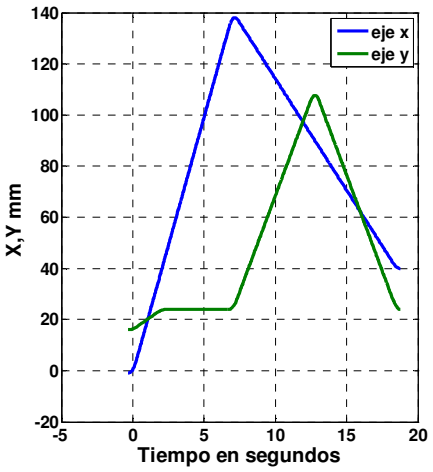



*Figura 12 Evolución de los ejes con la nueva versión*

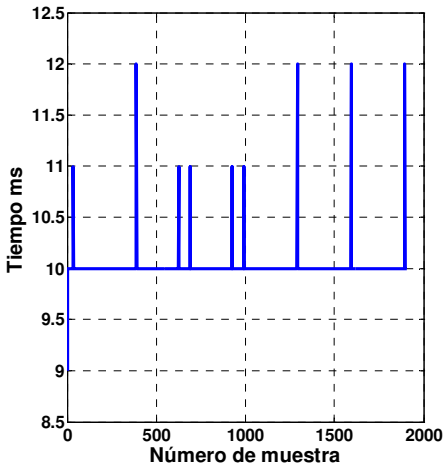



*Figura 13 Evolución del tiempo de muestreo*

## 5. CONCLUSIONES

En el artículo se discute la necesidad de utilizar sistemas operativos de tiempo real para el control de ejes robóticos. Se resalta la importancia del proceso que genera la trayectoria, que debe hacerlo en tiempo real sin la posibilidad de una planificación previa. Se muestra también que la utilización de un sistema operativo adecuado, es condición necesaria pero no suficiente para el correcto funcionamiento del sistema.

Se analizan errores encontrados en una estructura de control anterior y se explican las modificaciones realizadas para repararlo. Se muestra cómo el resultado de la cooperación entre procesos y la organización del software debe permitir ejecutar trayectorias no sólo con la forma deseada sino también en el tiempo deseado.

## 4. REFERENCIAS

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

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

[3] R. 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.

[4] A. Mauro, R. Ramos, M. Anigstein, *Nueva Interfase Hombre Máquina para la Plataforma del Robot Gantry-UBA*, Primer Congreso Argentino de Ingeniería Mecánica, I CAIM 2008. 1-3 Oct 2008, # 058 D. Bahía Blanca 2008.

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

[6] QNX RTOS 4.25 *System Architecture*. Note Control Systems Magazine, Vol 19, number 3, June 1999.

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

[8] J. J. Craig. *Robótica*, Pearson, Prentice-Hall, México. 2006.

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

[10] Photon microGUI 1.13 *Programmer's Guide*.