



UNIVERSIDAD NACIONAL DEL SUR

**TESIS DE MAGÍSTER
EN INGENIERÍA**

Redes de control en ambientes industriales

Plataforma de evaluación de sistemas distribuidos en tiempo real

Adriana Michelis

BAHÍA BLANCA

ARGENTINA

2009

Prefacio

Esta Tesis se presenta como parte de los requisitos para optar al grado Académico de Magíster en Ingeniería, de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otra. La misma contiene los resultados obtenidos en el ámbito del Departamento de Ingeniería Eléctrica y de Computadoras durante el período comprendido entre el 17/04/07 y el 31/03/09, bajo la dirección del Dr. Ricardo Cayssials y del Dr. Edgardo Ferro, Profesores del Departamento mencionado.

Adriana Michelis

Departamento de Ing. Eléctrica y de Computadoras
Universidad Nacional del Sur
Marzo de 2009



UNIVERSIDAD NACIONAL DEL SUR
Secretaría General de Posgrado y Educación Continua

La presente tesis ha sido aprobada el/..../..... , mereciendo
la calificación de (.....)

Agradecimientos

A mis directores Ricardo y Edgardo, por tanta paciencia y por su ayuda.

A Hugo, por sus consejos y por su perseverancia.

A la UNLPam, por el soporte económico facilitado que hicieron posible en buena parte este estudio.

A mi esposo Cesar y a mis amores Jerónimo, Joaquín, Juan Pedro y Juliana por acompañarme en este proceso.

Contenidos

PREFACIO	III
AGRADECIMIENTOS	V
ABSTRACT	XI
RESUMEN	XIII
PRINCIPALES CONTRIBUCIONES DE ESTA TESIS	XV
ESTRUCTURA DE LA TESIS.....	XVI
CAPÍTULO 1 SISTEMAS DE TIEMPO REAL	17
1.1 INTRODUCCIÓN	17
1.2 CLASIFICACIÓN DE LOS SISTEMAS DE TIEMPO REAL.....	18
1.3 MODELO DE SISTEMAS DE TIEMPO REAL	20
1.4 ALGORITMOS DE PLANIFICACIÓN Y DISCIPLINAS DE PRIORIDADES	21
1.4.1 <i>Factor de Utilización del Procesador</i>	22
1.5 CONCLUSIÓN.....	25
CAPÍTULO 2 SISTEMAS DE CONTROL	27
2.1 INTRODUCCIÓN	27
2.2 CONTROL DIGITAL	29
2.3 DEFINICIÓN DEL PERÍODO DE MUESTREO.....	30
2.4 ROBUSTEZ Y SENSIBILIDAD A LAS PERTURBACIONES	31
2.5 CONCLUSIÓN.....	32
CAPÍTULO 3 REDES DE COMUNICACIÓN EN SISTEMAS DE CONTROL	33
3.1 INTRODUCCIÓN	33
3.2 REDES DE CAMPO	34
3.2.1 <i>Requerimientos de una red de campo</i>	35
3.2.2 <i>Características de tráfico</i>	37
3.2.3 <i>Redes de campo en tiempo real</i>	37
3.2.4 <i>Estándares industriales de redes de campo</i>	38
3.3 CONCLUSIÓN.....	40
CAPÍTULO 4 SIMULACIÓN DE SISTEMAS DE CONTROL EN TIEMPO-REAL	41
4.1 INTRODUCCIÓN	41
4.2 TRUETIME.....	42
4.2.1 <i>Ambiente de simulación</i>	43
4.3 PROTOCOLO OPC.....	51
4.3.1 <i>Toolbox OPC de Matlab®</i>	52
4.3.2 <i>Software Matrikon®</i>	56
4.4 CONCLUSIÓN.....	57
CAPÍTULO 5 PLATAFORMA DE SIMULACIÓN DE SISTEMAS DE CONTROL EN TIEMPO-REAL	59
5.1 INTRODUCCIÓN	59

5.2	ESTRUCTURA DE LA PLATAFORMA.....	60
5.3	MODIFICACIÓN DE LA PLATAFORMA.....	68
5.3.1	<i>Parámetros que definen el desempeño de redes de campo.....</i>	<i>80</i>
5.3.2	<i>Monitoreo de consistencia</i>	<i>82</i>
5.4	CONCLUSIÓN.....	83
CAPÍTULO 6	CASOS DE ESTUDIO.....	85
6.1	INTRODUCCIÓN	85
6.1.1	<i>Control de temperatura de bobinado.....</i>	<i>85</i>
6.1.2	<i>Control distribuido utilizando OPC.....</i>	<i>90</i>
6.1.3	<i>Columna de destilación.....</i>	<i>94</i>
6.2	CONCLUSIÓN.....	103
CAPÍTULO 7	CONCLUSIONES FINALES	105
APÉNDICE I	DETALLES DE CASOS DE ESTUDIO	107
REFERENCIAS	133

Índice de Figuras

<i>Figura 1.1: Tiempo de slot Ts</i>	18
<i>Figura 2.1: Sistema de control continuo en el tiempo</i>	28
<i>Figura 2.2: Sistema de control discreto en el tiempo</i>	29
<i>Figura 3.1: Ubicación de las redes de campo dentro de las redes de comunicación</i>	34
<i>Figura 4.1: Librería de TrueTime. Las salidas Monitor y Diagramador exhiben la asignación de los recursos comunes (CPU, monitores, red) durante la simulación</i>	44
<i>Figura 4.2: Interfaces de los bloques Simulink®. Los puertos Schedule y Monitors proporcionan gráficos de la asignación de los recursos comunes (CPU, monitores, red) durante la simulación</i> 44	
<i>Figura 4.3: Ventanas de diagramador que muestran la asignación de recursos comunes: red (arriba) y nodo controlador (abajo). Una señal alta significa enviar o ejecutarse, una señal media significa esperar, y una señal baja significa ocioso</i>	49
<i>Figura 4.4: Ventana de diálogo en el bloque de red de TrueTime</i>	51
<i>Figura 4.5: Bloques OPC Config Real-Time, OPC Read y OPC Write</i>	52
<i>Figura 5.1: Pantalla gráfica principal de plataforma</i>	61
<i>Figura 5.2: Subpantalla gráfica de la plataforma</i>	61
<i>Figura 5.3: Subpantalla gráfica de la plataforma</i>	62
<i>Figura 5.4a: Pantalla de Simulink® de la plataforma</i>	62
<i>Figura 5.4b: Otra vista de la pantalla de Simulink® de la plataforma</i>	63
<i>Figura 5.5a: Entradas y salidas de los nodos en la red</i>	64
<i>Figura 5.5b: Entradas y salidas de los nodos en la red</i>	65
<i>Figura 5.6: Bloque sensor cableado</i>	65
<i>Figura 5.7: Bloque actuador cableado</i>	66
<i>Figura 5.8: Bloque controlador cableado</i>	66
<i>Figura 5.9: Bloque interferencia cableada</i>	67
<i>Figura 5.10: Ventana Block Parameters</i>	67
<i>Figura 5.11: Elementos del TrueTime Kernel que representa al sensor 1</i>	68
<i>Figura 5.12: Vista interna del bloque controlador₁</i>	80
<i>Figura 5.13: Bloque Mux</i>	80
<i>Figura 5.14: Monitor de consistencia</i>	83
<i>Figura 6.1: Control simple de temperatura de bobinado</i>	86
<i>Figura 6.2: Control en cascada de temperatura de bobinado</i>	87
<i>Figura 6.3: Estructura general del control en cascada</i>	87
<i>Figura 6.4: Bloques sensor, actuador y controlador utilizados en este ejemplo</i>	88
<i>Figura 6.5: Bloques plantas utilizados en este ejemplo</i>	88
<i>Figura 6.6: Elección de elementos con los que se armará el sistema en estudio</i>	89
<i>Figura 6.7: Respuesta del sistema a lazo cerrado usando control en cascada</i>	90
<i>Figura 6.8: Bloques interferencia, sensor y actuador utilizados en este ejemplo</i>	92
<i>Figura 6.9: Bloque planta utilizado en este ejemplo</i>	92
<i>Figura 6.10: Servidor OPC y PLC (con su controlador)</i>	92
<i>Figura 6.11: Elección de elementos con los que se armará el sistema en estudio</i>	93
<i>Figura 6.12: Respuesta a lazo cerrado empleando un esquema de control realimentado de tipo PID, sin interferencia en la red</i>	94
<i>Figura 6.13: Columna de destilación</i>	95
<i>Figura 6.14: Control PI descentralizado de la columna de destilación</i>	96
<i>Figura 6.15: Bloques sensor, actuador y controlador utilizados en este ejemplo</i>	97
<i>Figura 6.16: Bloques plantas utilizados en este ejemplo</i>	97
<i>Figura 6.17: Elección de elementos con los que se armará el sistema en estudio</i>	98
<i>Figura 6.18: Respuesta a lazo cerrado empleando un esquema de control feedback de tipo PI</i> ... 99	
<i>Figura 6.19: Bloques sensor y actuador utilizados en este ejemplo</i>	99
<i>Figura 6.20: Bloques plantas utilizados en este ejemplo</i>	100
<i>Figura 6.21: Servidor OPC y PLC (con su controlador)</i>	101
<i>Figura 6.22: Elección de elementos con los que se armará el sistema en estudio</i>	102
<i>Figura 6.23: Respuesta a lazo cerrado empleando esquema de control feedback de tipo PI</i>	103
<i>Figura I.1: Bloques sensor, actuador y controlador utilizados en este ejemplo</i>	110
<i>Figura I.2: Bloques plantas utilizados en este ejemplo</i>	110

<i>Figura I.3: Bloque controlador1 por dentro.....</i>	<i>112</i>
<i>Figura I.4: Elección de elementos con los que se armará el sistema en estudio</i>	<i>114</i>
<i>Figura I.5: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A1 (no ingresar e-s porque está representada por un bloque retardo en “plataforma.mdl”).....</i>	<i>114</i>
<i>Figura I.6: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A2.....</i>	<i>115</i>
<i>Figura I.7: Bloques interferencia, sensor y actuador utilizados en este ejemplo</i>	<i>116</i>
<i>Figura I.8: Bloque planta utilizado en este ejemplo</i>	<i>116</i>
<i>Figura I.9: Servidor OPC y PLC (con su controlador)</i>	<i>117</i>
<i>Figura I.10: Elección de elementos con los que se armará el sistema en estudio</i>	<i>119</i>
<i>Figura I.11: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta</i>	<i>119</i>
<i>Figura I.12: Bloques sensor, actuador y controlador utilizados en este ejemplo.....</i>	<i>121</i>
<i>Figura I.13: Bloques plantas utilizados en este ejemplo.....</i>	<i>121</i>
<i>Figura I.14: Elección de elementos con los que se armará el sistema en estudio</i>	<i>123</i>
<i>Figura I.15: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A1 (no ingresar e-0.043s porque está representado por un bloque retardo en “plataforma.mdl”).....</i>	<i>124</i>
<i>Figura I.16: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A2 (no ingresar e-0.017s porque está representado por un bloque retardo en “plataforma.mdl”).....</i>	<i>124</i>
<i>Figura I.17: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A3 (no ingresar e-0.017s porque está representado por un bloque retardo en “plataforma.mdl”).....</i>	<i>125</i>
<i>Figura I.18: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A4 (no ingresar e-0.153s porque está representado por un bloque retardo en “plataforma.mdl”).....</i>	<i>125</i>
<i>Figura I.19: Bloques sensor, actuador utilizados en este ejemplo.....</i>	<i>126</i>
<i>Figura I.20: Bloques plantas utilizados en este ejemplo.....</i>	<i>126</i>
<i>Figura I.21: Servidor OPC y PLC (con su controlador)</i>	<i>127</i>
<i>Figura I.22: Elección de elementos con los que se armará el sistema en estudio</i>	<i>130</i>
<i>Figura I.23: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A1 (no ingresar e-0.043s porque está representado por un bloque retardo en “plataforma.mdl”).....</i>	<i>130</i>
<i>Figura I.24: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A2 (no ingresar e-0.017s porque está representado por un bloque retardo en “plataforma.mdl”).....</i>	<i>131</i>
<i>Figura I.25: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A3 (no ingresar e-0.017s porque está representado por un bloque retardo en “plataforma.mdl”).....</i>	<i>131</i>
<i>Figura I.26: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A4 (no ingresar e-0.153s porque está representado por un bloque retardo en “plataforma.mdl”).....</i>	<i>131</i>

Abstract

In most of the industrial applications, it is necessary the verification of the correct operation of the control strategies and automatism before being applied to the real process. This type of applications requires the use of real time systems to guarantee that the results not only are correct from the arithmetical and logical point of view, but that take place before a certain time, named *deadline*. The utilization of real-time systems in this kind of applications includes from industrial controllers (PLC) to data and sensors networks for advanced distributed control systems (DCS).

Also, different constraints may exist that can be opposed to the temporal requirements in such designs: the energy consumption, the size of the implementations, the communications rate of the data links, among others.

On the other hand, the present applications in which strategies of control and automatism are required, demand sophisticated techniques of analysis and verification of operation under different scenes. Many of the operation scenes that has to be considered, involve critical conditions of the process which are not desirable to reproduce in the real application and much less with a control system that is under verification.

At the moment, the distributed systems and networks of sensors are widely used in diverse applications and with different objectives. Although in these applications the temporal requirements rarely demand a treatment of hard real-time, it is necessary to guarantee that temporal consistency of the data is preserved. Consequently, a holistic analysis of the designed control systems and the application is necessary to assure that the design specifications are satisfied for different scenes of operation.

In most of the modern implementations of strategies of control and automatism, simulators of the controllers are used, artificially generating the inputs that come from the applications. Nevertheless, this type of testing methodology does not allow verifying the right dynamics of the controller, because it is not a easy task to generate the same behavior in the inputs of the simulator that when these input are connected to the real sensors. In order to avoid these restrictions in the verification, and in applications that justify it, usually they are developed platforms that emulate the behavior of the specific application. Nevertheless, the use of these platforms is exclusive to the application for which they were designed and consequently its development is generally expensive.

The objective of this thesis is the development of a simulation platform designed for the analysis and simulation of distributed control systems and sensor networks in real-time applications in which strategies of control and industrial automatism are implemented. In order to develop this platform the TrueTime tool, developed by Johan Eker and Anton Cervin and later extended by its research group, was used. This tool allows us the simulation in a Simulink[®]/Matlab[®] environment of the set real time/control features of an implementation in order to verify the disturbances that different control configurations produce on the control applications. The TrueTime tool was modified and extended to incorporate the parameters that, after a deep analysis, were considered the main parameters required to build the most common scenes required in industrial applications. The communication protocol OPC was configured in order to generate all the simulated surroundings of the application on the industrial controller under verification.

Resumen

En la mayoría de las aplicaciones industriales es necesaria la verificación del correcto funcionamiento de las estrategias de control y automatismo antes de ser aplicadas al proceso real. Este tipo de aplicaciones requieren la utilización de sistemas de tiempo real para garantizar que los resultados no sólo son correctos desde el punto de vista lógico-aritmético, sino que se producen antes de un determinado tiempo, denominado *vencimiento*. Los sistemas de tiempo real utilizados en este tipo de aplicaciones incluyen desde controladores industriales (PLC¹), redes de datos y sensores hasta avanzados sistemas de control distribuido (DCS).

Asimismo, existen diferentes restricciones que pueden contraponerse a los requerimientos temporales en tales diseños como ser: el consumo de energía, las dimensiones físicas de las aplicaciones, el alcance y el retardo de los enlaces de datos, entre otras.

Por otro lado, las actuales aplicaciones en donde se requieren estrategias de control y automatismos demandan técnicas sofisticadas de análisis y verificación

¹ PLC: Programmer Logical Controller

de funcionamiento bajo diferentes escenarios. Muchos de los escenarios de funcionamiento que deben ser considerados involucran condiciones críticas del proceso, las cuales no son deseables de reproducir en la aplicación real y menos aún con un sistema de control que está bajo verificación.

Actualmente, los sistemas distribuidos y redes de sensores son ampliamente utilizados en diversas aplicaciones y con diferentes objetivos. Si bien en estas aplicaciones los requerimientos temporales raramente exigen un tratamiento de tiempo real duro, sí resulta necesario garantizar la consistencia temporal de los datos para asegurar que las acciones que se obtengan de dichos datos sean también consistentes. En consecuencia, es necesario un análisis conjunto de los sistemas diseñados y la aplicación que permita confirmar que las especificaciones de diseño se satisfacen para diferentes escenarios de funcionamiento.

En la mayoría de las implementaciones modernas de estrategias de control y automatismos se utilizan simuladores de los controladores y se generan artificialmente las entradas que en la aplicación real provendrían de los sensores. Sin embargo, este tipo de metodología de prueba no permite verificar la correcta dinámica del controlador debido a que no es simple generar el mismo comportamiento en las entradas que cuando éstas están conectadas a los sensores reales. Para evitar estas restricciones en la verificación, y en aplicaciones que lo justifican, se suelen desarrollar plataformas que emulen el comportamiento de la aplicación específica. No obstante, la utilización de estas plataformas es exclusiva a la aplicación para las que fueron destinadas y su costo de desarrollo es generalmente elevado.

Esta tesis tiene como objetivo el desarrollo de una plataforma para el análisis y la simulación de sistemas distribuidos y redes de sensores en tiempo real en aplicaciones donde se implementen estrategias de control y automatismos industriales. Para realizar esta plataforma se utilizó la herramienta TrueTime, creada por Johan Eker y Anton Cervin² y posteriormente ampliada por su grupo de investigación. Esta herramienta permite la simulación en un ambiente de Simulink[®]/Matlab[®] del conjunto tiempo-real/control para verificar las

² Ver <http://www.control.lth.se/attic/truetime/>

perturbaciones que diferentes algoritmos de diagramación producen en las aplicaciones de control. La herramienta TrueTime fue modificada y ampliada para la incorporación de los parámetros que, luego de un profundo análisis, son los adecuados para monitorear y configurar en forma genérica las redes de sensores y sistemas distribuidos anteriormente detallados. El protocolo de comunicaciones OPC³ fue utilizado para generar en el controlador industrial bajo verificación todo el entorno simulado de la aplicación.

Principales contribuciones de esta tesis

El objetivo principal del plan de trabajo propuesto consistió en el diseño e implementación de herramientas de simulación y monitoreo para verificación de la consistencia de datos y estrategias en un sistema distribuido de control en tiempo real. Como resultado final de estos análisis se implementó una plataforma genérica basada en TrueTime y OPC que servirá para futuros estudios de sistemas distribuidos de control.

La herramienta TrueTime permite la simulación de sistemas de tiempo real, tanto monoprocesador como distribuidos, y su utilización requiere un profundo conocimiento de sus comandos. La plataforma propuesta permite que un diseñador pueda simular y verificar su sistema distribuido o red de sensores de manera sencilla mediante la introducción de los parámetros que le son conocidos y no mediante la utilización de funciones específicas de la herramienta.

La incorporación de comunicación mediante protocolo OPC ([15]) permite generar ambientes de simulación de aplicaciones para la verificación de la correcta implementación de estrategias en controladores reales. De esta manera, se posibilita una plataforma útil en el desarrollo y la verificación de estrategias de automatización y control industrial como en la enseñanza de técnicas de automatización de vanguardia.

³ OPC: Ole for Process Control en inglés. Es un protocolo de comunicación que especifica parámetros para comunicación en tiempo real entre diferentes aplicaciones y diferentes dispositivos de control de diferentes proveedores

Estructura de la Tesis

La tesis está organizada como sigue: en el Capítulo 1 se presentan los principales conceptos de *tiempo-real* que serán utilizados en el resto de la tesis. En el Capítulo 2, se describen los conceptos más importantes en que se basa un diseño de un sistema de control. En el Capítulo 3, se describen los principales conceptos de redes de sensores. En el Capítulo 4 se describen por un lado, las principales características modeladas por la librería TrueTime; y por otro lado, se da una leve noción de las características del protocolo de comunicación OPC. En el Capítulo 5, se describen los componentes y estructuras de la plataforma de simulación desarrollada. En el Capítulo 6, se muestran diversos casos de estudio que permiten evaluar las diferentes características de la plataforma. Por último, en el Capítulo 7, se dan las conclusiones finales de todo el trabajo de tesis.

Certifico que fueron incluidos los cambios y correcciones sugeridas por los jurados.

Firma del Director

Capítulo 1

Sistemas de Tiempo Real

1.1 Introducción

Un sistema de tiempo real es un sistema en el cual su buen funcionamiento depende no sólo de los resultados aritmético-lógicos, sino también del tiempo en el cual esos resultados son producidos.

Un caso particular de sistema de tiempo real son las redes de datos industriales que deben satisfacer condiciones de *tiempo-real*: en ellas, un mensaje generado en un nodo transmisor debe alcanzar su destino no sólo sin errores, sino que debe hacerlo dentro de un lapso preestablecido después de su generación; este lapso se denomina *vencimiento*. El retardo del mensaje es un intervalo de tiempo que va desde el instante en el cual es generado en el nodo fuente (está listo para ser transmitido) hasta el instante en el cual el último bit es recibido por el nodo receptor. Esto incluye, el tiempo de acceso al medio de transmisión de la red, el tiempo para transmitir el mensaje en el nodo generador y el tiempo para propagarlo a través de la red. Si el retardo es mayor que el vencimiento, el

mensaje se considera perdido y se dice que ha ocurrido una *crisis*. Es obvio entonces que los nodos deben tener garantizado cierto tiempo de acceso al medio. Como consecuencia, para una red dada, debe ser impuesto un límite sobre la longitud del mensaje a ser transmitido, y existirá un máximo tiempo entre el comienzo de dos sucesivos mensajes cuando por lo menos un nodo está listo para transmitir mientras otro está realmente transmitiendo. Ese tiempo es denominado *tiempo de ranura*, e indicado T_s (slot time), ver Figura 1.1. El tiempo T_s está formado por: el tiempo de *sobrecarga*⁴ T_o , usado para funciones de mantenimiento, sincronismo y para determinar cuál nodo adquiere el derecho a transmitir de acuerdo a cierta disciplina de prioridad; el tiempo T_m utilizado para *transmitir el mensaje*; el tiempo T_p para *propagarlo* desde el nodo generador al nodo receptor. Sólo una parte T_d del tiempo T_m es usado para transmitir datos relevantes de la aplicación, el resto $T_m - T_d$ es usado para la transmisión de los bytes del preámbulo, dirección, control, detección de errores, etc.

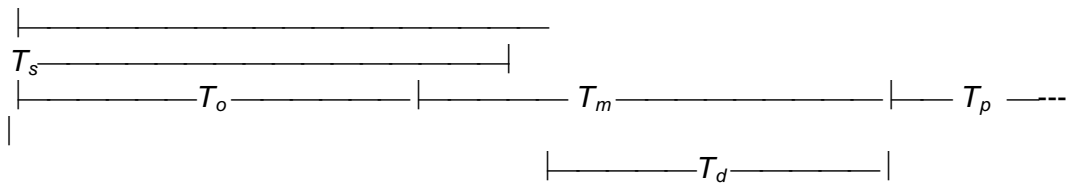


Figura 1.1: Tiempo de slot T_s

1.2 Clasificación de los sistemas de tiempo real

En algunas ocasiones, se pueden ver referencias sobre sistemas de *tiempo-real* cuando sólo se quiere decir que el sistema es rápido. Cabe mencionar que *tiempo-real* no es sinónimo de rapidez; esto significa que no es la latencia⁵ de la respuesta lo que nos enfoca en un sistema de *tiempo-real* (esta latencia a veces está en el orden de los segundos), sino el asegurarse de que la latencia del sistema no es impedimento para resolver el problema al cual el sistema está dedicado.

⁴ Overhead en inglés

⁵ En redes informáticas de datos se denomina **latencia** a la suma de retardos temporales dentro de una red. Un retardo es producido por la demora en la propagación y transmisión de paquetes dentro de la red.

El sistema de *tiempo-real* está definido por el hecho de que el sistema cumple con los vencimientos de todas funciones que debe cumplir, o mensajes que debe transmitir. Si la pérdida de un solo vencimiento es considerada crítica, y en consecuencia produce una falla crítica del sistema, entonces se dice que el sistema es de *tiempo-real duro*. Por otro lado, un proceso de *tiempo-real blando* es aquél que manifiesta una tolerancia ante la pérdida de vencimientos, cuando dicha pérdida está dentro de ciertos límites establecidos (por ejemplo, es posible perder uno de cada 100 ó 1000 vencimientos, o definir que la pérdida de vencimientos no exceda el 10%).

Sin embargo, la definición de un porcentaje en la pérdida de vencimientos que un sistema puede tolerar, muchas veces no es suficiente para una especificación. Por ejemplo, si se especifica que el sistema puede perder 30% de los vencimientos, dicha especificación se satisface si se pierden 3 vencimientos consecutivos y se cumplen los siguientes 7, o si se pierden 300 vencimientos consecutivos y se cumplen los siguientes 700, que obviamente no es lo mismo. Para permitir una mejor definición de las especificaciones temporales de sistemas de *tiempo-real* tolerantes a pérdidas de vencimientos, se propuso la especificación de *tiempo-real débil*. En este tipo de especificación, se define el número y forma en que pueden perderse vencimientos dentro de una determinada ventana de tiempo.

De esta manera, los sistemas de tiempo real pueden clasificarse de la siguiente manera, dependiendo de su tolerancia a la pérdida de vencimientos:

- ***Tiempo-real duro***: es aquel en que se garantiza que ningún vencimiento será perdido. Es utilizado en aplicaciones críticas de control o donde la pérdida de un vencimiento puede tener consecuencias catastróficas.
- ***Tiempo-real blando***: es aquel en que se garantiza que el promedio de pérdidas de vencimientos estará por debajo de un determinado umbral. Es utilizado generalmente en sistemas en donde la pérdida de vencimientos provoca, como mayor consecuencia, una caída en la calidad de servicio.
- ***Tiempo-real débil***: es aquel en que se garantiza que las pérdidas de vencimientos estará acotada en su cantidad y forma dentro de una

determinada ventana de tiempo. Son utilizados para tener una especificación precisa de la pérdida de vencimientos y obtener implementaciones menos sobredimensionadas que los sistemas de *tiempo-real* duros.

1.3 Modelo de Sistemas de Tiempo Real

En un sistema de *tiempo-real* existen uno o varios recursos que deben ser compartidos entre varias entidades que lo requieren, cumpliendo sus respectivos vencimientos. De esta manera, en una red de controladores, el recurso que debe compartirse es el *medio físico* utilizado para la transmisión de datos y las entidades que lo requieren son los controladores que en forma genérica se denominan *odos*. Por otro lado, en un controlador diversas funciones deben ser ejecutadas por el único procesador que el controlador contiene. En estos sistemas de procesamiento, el recurso a compartir está constituido por el *procesador* y las entidades que deben compartir su utilización son las diversas *tareas* que implementan las diversas funcionalidades programadas en dicho controlador.

Para una mejor facilidad de escritura, en lo que sigue de la tesis nos referiremos a los componentes de los sistemas de *tiempo-real* mediante la terminología de los sistemas de procesamiento, esto es: tareas para las entidades que requieren la utilización del recurso y procesador para el recurso.

Un análisis de factibilidad requiere la modelización del recurso y de las entidades para determinar si las restricciones temporales serán satisfechas. La manera más utilizada en la literatura de tiempo-real, modela a un sistema de *tiempo-real* como un conjunto Π de N tareas. La tarea i , simbolizada como τ_i , está caracterizada por los siguientes parámetros:

- **Periodo:** es el mínimo tiempo transcurrido entre dos requerimientos consecutivos de una tarea que desea ser ejecutada. El periodo de la tarea i se representa como T_i .
- **Máximo tiempo de ejecución:** es el máximo tiempo que la tarea requiere la utilización del procesador para ejecutarse completamente. El máximo tiempo de ejecución de la tarea i se representa como C_i .

- **Vencimiento:** es el intervalo de tiempo medido desde la solicitud de ejecución de la tarea en el que la tarea debe ser ejecutada completamente. El vencimiento de la tarea i se representa como D_i .

De esta manera, un sistema de *tiempo-real* está caracterizado por:

$$\Pi = \{\tau_i = (T_i, C_i, D_i), 1 \leq i \leq N\}$$

Debido que existen varias tareas que requieren el uso del procesador y el procesador no puede ejecutar más de una tarea en un determinado instante, debe organizarse la utilización del procesador entre todas las tareas que lo requieran. La organización en la ejecución de las tareas se realiza mediante un *algoritmo de planificación*, el cual es el encargado de implementar una *disciplina de prioridades*.

1.4 Algoritmos de planificación y disciplinas de prioridades

Según lo expuesto por Liu & Layland en [1], un algoritmo de planificación, o simplemente *planificador*, es un conjunto de reglas que determinan cuál es la tarea a ser asignada a un recurso o procesador en un determinado instante de tiempo. En los sistemas de *tiempo-real* compuestos por redes de comunicaciones, el planificador es implementado mediante las reglas que determinan el control de acceso al medio.

Los algoritmos de planificación pueden ser divididos en dos tipos bien diferenciados: los *Estáticos o Fijos* y los *Dinámicos*. En éstos, la noción de prioridad (Fija o Dinámica) es utilizada para ordenar el acceso al procesador. Cuando hay varias tareas que requieren acceder a un recurso, ésto se resuelve asignando el mismo a la tarea con la más alta prioridad ([2]). Cada vez que una tarea solicita la utilización del recurso compartido, se dice que la tarea es *invocada*.

Existen condiciones exactas, necesarias y suficientes para el análisis de la planificabilidad. Éstas pretenden garantizar los requerimientos temporales respetando el acceso compartido al recurso.

Los algoritmos de planificación Estáticos son aquellos en los que la asignación de prioridades se mantiene invariante durante todo el tiempo de funcionamiento del sistema. También se los llama *Algoritmos de Planificación de Prioridades Fijas*. Por otro lado, los algoritmos de planificación Dinámicos son aquellos en los que las prioridades de las tareas se modifican durante el tiempo de funcionamiento del sistema.

En [1], se presentan dos algoritmos muy usados y conocidos en la teoría de *tiempo-real*, como lo son el algoritmo estático *FP (Fixed Priority)* y el algoritmo dinámico *EDF (Earliest Deadline First)*.

Un parámetro muy importante para medir la eficiencia de los planificadores es el *máximo tiempo de respuesta*. Éste es definido como el tiempo entre el pedido de ejecución de una tarea y la finalización de la ejecución de la misma.

En particular, Liu & Layland presentan una serie de suposiciones respecto al entorno, para la adecuada utilización de las condiciones necesarias y suficientes que plantean:

- Un conjunto fijo de tareas periódicas.
- El vencimiento de cada tarea es igual a su período ($T = D$).
- Las tareas deben ser independientes, esto es, no tienen relaciones de precedencia o de recursos.
- Todas las tareas deben terminar de ejecutarse antes de su próxima instancia.
- Ninguna tarea puede desalojarse a sí misma.
- Todas las tareas son completamente desalojadas.
- Los tiempos de ejecución de cada tarea son fijos ($C = \text{fijo}$).
- El único recurso en común es el procesador.

1.4.1 Factor de Utilización del Procesador.

Se define *Factor de Utilización* de una tarea ([1]) a la fracción de tiempo de procesamiento del procesador que necesita una tarea para ser ejecutada completamente. Por consiguiente, el factor de utilización de una tarea τ_i , denominado U_i , queda definido como:

$$U_i = \frac{C_i}{T_i} \quad (1)$$

Si consideramos el caso en que $D_i = T_i$ y dado que C_i/T_i es la fracción de tiempo del procesador utilizado en ejecutar la tarea τ_i , de un conjunto de N tareas, el Factor de Utilización Total del sistema, denominado U , es:

$$U = \sum_{i=1}^N (C_i/T_i) \quad (2)$$

Este factor de utilización indica la carga del sistema. Si dicho valor supera la unidad se dice que el sistema está *sobrecargado*. Esto implica que existirán tareas que no podrán satisfacer su ejecución antes de su vencimiento, debido a que la capacidad del procesador es inferior al requerimiento del sistema. Este es el límite superior para satisfacer el requerimiento de que todas las tareas cumplan sus vencimientos.

Cuando el factor de utilización es igual a la unidad, el sistema se denomina *saturado*. En cambio, cuando este número no alcanza la unidad se dice que el sistema está *relajado*. Sin embargo, sistemas relajados pueden perder vencimientos debido a la política de planificación.

1.4.2 Prioridades Fijas: FP

Bajo las suposiciones realizadas anteriormente, Liu y Layland, en [1], proponen un método de asignación de prioridades para la disciplina de prioridades fijas. Este método, denominado *Periodos Monotónicos Crecientes* (RM^6), está basado en la tasa de arribo de los pedidos de ejecución. A tareas con menor período se les asignan las mayores prioridades.

Esta asignación de prioridades es óptima en el sentido de que ninguna otra regla de asignación de prioridades fijas puede planificar un conjunto de tareas que no puede ser planificado bajo RM . Es decir, si un conjunto de tareas no puede ser planificado bajo RM , entonces no puede ser planificado por ningún otro algoritmo de prioridades fijas.

⁶ RM : Rate Monotonic, en inglés

En [1], se propone una cota superior para analizar la planificabilidad de estos algoritmos de prioridades fijas:

$$U_p = N \left(2^{\frac{1}{N}} - 1 \right) \quad (3)$$

Teniendo en cuenta la ecuación (2), un conjunto de N tareas es planificable bajo RM si se cumple:

$$U \leq U_p \quad (4)$$

Esta condición es suficiente y no necesaria. Esta condición de diagramabilidad es muy conservadora y pueden existir sistemas no planificables mediante prioridades fijas que satisfacen esta cota.

1.4.3 Menor Tiempo al Vencimiento: EDF

En [1], Liu y Layland, plantean una asignación de prioridades dinámicas basadas en los vencimientos de las tareas. A la tarea que en su instancia actual tiene el menor tiempo al vencimiento, se le asigna la mayor prioridad. Por otro lado, a la tarea que en su instancia actual tiene el vencimiento más lejano, se le asigna la menor prioridad.

Este método es óptimo en el sentido de que, si un sistema no es planificable por EDF ⁷ (*Menor Tiempo al Vencimiento*), no lo será por ninguna otra disciplina de prioridades.

Teniendo en cuenta la ecuación (2), un conjunto de tareas es planificable bajo EDF sí y sólo sí:

$$U \leq 1 \quad (5)$$

Esta es una condición necesaria y suficiente para garantizar la diagramabilidad de un sistema de tiempo real diagramado mediante una disciplina de prioridades EDF .

⁷ EDF : Earliest Deadline First, en inglés

1.5 Conclusión

En este capítulo se presentaron los principales conceptos de *tiempo-real* que serán utilizados en los próximos capítulos de la tesis. Para esto, se hizo referencia al trabajo más destacado en la teoría que trata los sistemas de *tiempo-real* como es el de Liu y Layland [1].

Capítulo 2

Sistemas de Control

2.1 Introducción

El control automático ha jugado un papel vital en el avance de la ciencia y de la ingeniería. Además de su extrema importancia en vehículos espaciales, sistemas de guía de proyectiles, sistemas de piloto automático de aeronaves, sistemas robóticos y otros, el control automático se ha vuelto parte integral e importante de los procesos industriales y de manufactura modernos. Por ejemplo, el control automático resulta esencial en el control numérico de las máquinas herramienta en las industrias manufactureras. También resulta esencial en operaciones industriales como el control de presión, temperatura, humedad, viscosidad, y flujo en las industrias de transformación.

Un sistema de control básico consiste generalmente de una *planta* y de un *controlador*. La planta es el sistema en el cual se encuentran las señales o variables que se desean controlar. El controlador es el que realiza este trabajo en

tres funciones bien diferenciadas, primero lee la información que le proporcionan dichas variables por medio de sensores ubicados estratégicamente en la planta, luego calcula las acciones de control y por último actúa sobre la planta para que el sistema alcance la eficiencia de control deseada.

Para expresar matemáticamente la dinámica tanto de la planta como del controlador se aplican técnicas de modelado. El modelo es una abstracción simplificada usada para predecir el comportamiento del sistema. Es frecuente obtener un modelo analítico del sistema usando leyes que determinan su comportamiento

El modelo de *espacio de estado* es una técnica muy usada en modelado de sistemas. En un sistema de tiempo continuo, un modelo en el espacio de estados consiste de un vector de ecuaciones diferenciales de primer orden, denominado *ecuación de espacio de estado* y una ecuación de salida, las cuales son mostradas en las ecuaciones (6) y (7) respectivamente,

$$\dot{x}(t) = f(x(t), u(t), t) \quad (6)$$

$$y(t) = g(x(t), u(t), t) \quad (7)$$

donde $x(t)$ es denominada variable de estado, $u(t)$ es la variable de entrada, $y(t)$ es la variable de salida, f es la función de estados y g es la función de salida.

Un prototipo típico generalmente encontrado en la teoría clásica de control se muestra en la Figura 2.1. En esta figura, la señal realimentada $y(t)$ es la *señal de salida* del sistema, que es comparada con la *señal de referencia de entrada* $r(t)$, para obtener la *señal error* $e(t)$ que es la que entrará al controlador para luego obtener la *señal de control* $u(t)$ que será aplicada a la planta.

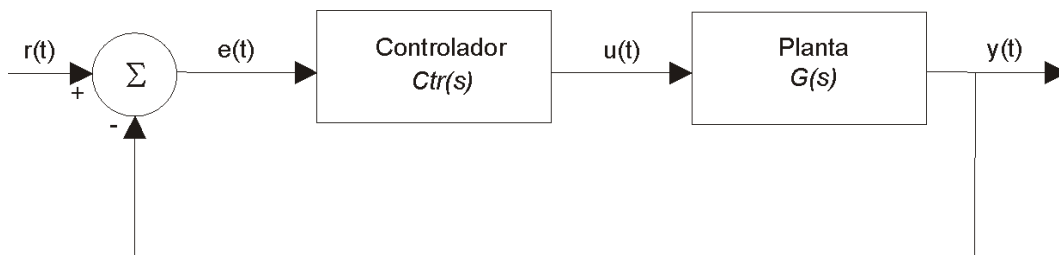


Figura 2.1: Sistema de control continuo en el tiempo

El problema de diseño es especificar la función de transferencia $Ctr(s)$ para cumplir con las características deseadas del sistema a lazo cerrado. Dentro de las características más comunes a tener en cuenta en el diseño, se encuentran la estabilidad y algunas especificaciones para la respuesta al escalón como el *porcentaje de sobrepico*, el *tiempo de establecimiento* y el *error en estado estacionario*.

Dos de las propiedades más importantes de un sistema de control son la *linealidad* y la *invariancia en el tiempo* ([3]). El modelo de un sistema lineal invariante y continuo en el tiempo es expresado en espacio de estado en la ecuación (8),

$$\begin{aligned}\dot{x}(t) &= \mathbf{A} \cdot x(t) + \mathbf{B} \cdot u(t) \\ y(t) &= \mathbf{c} \cdot x(t) + \mathbf{d} \cdot u(t)\end{aligned}\quad (8)$$

donde \mathbf{A} es la matriz del sistema, \mathbf{B} es la matriz de entrada, \mathbf{c} es la matriz de salida y \mathbf{d} es la matriz de realimentación.

2.2 Control Digital

Cuando un controlador es implementado sobre una computadora, se debe usar un modelo discreto. La Figura 2.2 muestra una planta en tiempo continuo controlada por un controlador en tiempo discreto.

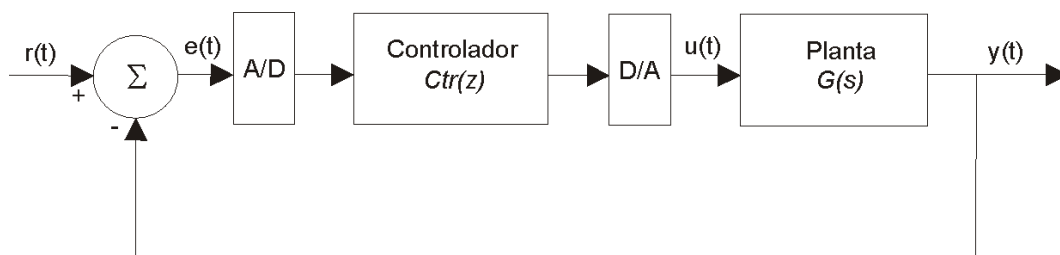


Figura 2.2: Sistema de control discreto en el tiempo

La función de transferencia del controlador $Ctr(z)$ es implementada como software y ejecutada por un procesador. La entrada a $Ctr(z)$ es una secuencia de números obtenida desde un conversor A/D y la salida es otra secuencia de

números que es convertida a una señal de control continua a tramos a través de un conversor D/A .

El modelo de espacio de estados de un sistema de tiempo discreto es expresado por las ecuaciones en diferencias mostradas en la ecuación (9).

$$\begin{aligned} x[k+1] &= \mathbf{\Phi} \cdot x[k] + \mathbf{\Gamma} \cdot u[k] \\ y[k] &= \mathbf{c} \cdot x[k] + \mathbf{d} \cdot u[k] \end{aligned} \quad (9)$$

En [3], se muestra que dado un modelo en espacio de estados de una planta caracterizada por las matrices \mathbf{A} , \mathbf{B} , \mathbf{c} , y \mathbf{d} como el representado en la ecuación (8) y un intervalo de muestreo T , el sistema equivalente en tiempo discreto está dado por las matrices $\mathbf{\Phi}$, $\mathbf{\Gamma}$, \mathbf{c} y \mathbf{d} , donde

$$\mathbf{\Phi} = e^{\mathbf{A}T}, \quad \mathbf{\Gamma} = \int_0^T e^{\mathbf{A}\tau} \cdot \mathbf{B} \cdot d\tau \quad (10)$$

Este modelo discreto obtenido a través de esta transformación *no es una aproximación, sino una descripción exacta del comportamiento de la planta en cada instante de muestreo T .*

Variando el intervalo de muestreo a lo largo del tiempo, el modelo de tiempo discreto deja de ser válido, y por consiguiente mismas acciones producirán efectos diferentes. Un retardo variable durante la actualización de las salidas, producirá un efecto que puede no ser modelado conduciendo a un comportamiento no deseado en el sistema. Con esta tesis se pretende obtener una plataforma de simulación y verificación que permita comprobar que la estrategia de control implementada es tolerante a las perturbaciones producidas por el sistema de control digital.

2.3 Definición del período de muestreo

En control digital la elección del período de muestreo es muy importante. Si se elige muy grande, la señal de tiempo continuo no podrá ser reconstruida. Por otro lado, si se elige muy pequeño, la carga computacional aumentará y posiblemente la actualización de la salida no tendrá lugar.

Existen varias reglas de diseño para determinar un rango de selección para la frecuencia de muestreo, w_s (definida en la ecuación (11)) y por consiguiente también para el período de muestreo ($T = 1/w_s$).

$$w_s = 2 \cdot \frac{\pi}{T} (\text{rad} / \text{seg}) \quad (11)$$

Muchas de esas reglas son basadas en la relación entre la frecuencia de muestreo y el ancho de banda del sistema a lazo cerrado, denominado w_B .

De acuerdo a las reglas en [3], w_s puede ser elegido de la siguiente manera:

- Si la planta (o los sensores) están sujetos a perturbaciones aleatorias cuyo contenido de frecuencia excede el ancho de banda del sistema de control, se elige: $20 \leq w_s/w_B \leq 40$.
- Si la planta no está sujeta a perturbaciones aleatorias, se elige: $5 \leq w_s/w_B \leq 10$.

Una velocidad de muestreo alta puede hacer que el sistema se torne muy sensible a la precisión de los parámetros y a los errores de cuantización.

2.4 Robustez y sensibilidad a las perturbaciones

En la implementación de sistemas de control, el comportamiento logrado puede diferir del esperado debido a varias cuestiones de implementación. Algunas de estas cuestiones son enumeradas a continuación:

- a. Variaciones de los parámetros del sistema. Tanto las tolerancias mecánicas como la calidad del producto, pueden producir diferentes características de funcionamiento en la implementación final. Por lo tanto, un sistema de control debe ser lo suficientemente robusto como para tener en cuenta todas estas variaciones.
- b. Precisión e histéresis de sensores y actuadores. Los sensores y actuadores reales pueden no responder a pequeñas variaciones en sus entradas debido a propiedades de precisión, sensibilidad o histéresis.

- c. Cuantificación de errores de conversores. Conversores A/D y D/A tienen cuantificación de errores debido a la longitud finita de sus representaciones binarias.
- d. Rango dinámico finito de sensores y actuadores. La respuesta tanto de un sensor como de un actuador es limitada a un cierto rango y debe ser tenida en cuenta en el análisis de estabilidad y robustez.

Muchos de estos factores pueden ser reducidos mejorando el hardware utilizado, con la consecuencia del aumento del costo. En este caso se debe tener en cuenta una solución de compromiso.

2.5 Conclusión

En este capítulo se describieron los principales conceptos en que se basa un diseño de un sistema de control. En el resto de la tesis utilizaremos herramientas que nos permitirán simular las plantas y procesos a controlar y las estrategias a implementar mediante los modelos que describen su comportamiento.

Capítulo 3

Redes de Comunicación en Sistemas de Control

3.1 Introducción

Los sistemas de control modernos son utilizados en las más diversas industrias. La complejidad de los procesos y la necesidad de lograr implementaciones más eficientes, económicas y confiables requieren la comunicación y procesamiento de un gran volumen de información. Dicha información puede ser procesada en diferentes niveles del proceso.

La mayoría de las comunicaciones realizadas en el control y administración de un proceso pueden ser clasificadas en cuatro niveles: *instrumentación*, *control*, *ejecución* y *gerenciamiento*.

Cada uno de estos niveles debe satisfacer diferentes requerimientos y por consiguiente existen diferentes redes de comunicación con diferentes características técnicas para cada uno de dichos niveles. Las redes de comunicación a nivel instrumentación, denominadas *redes de campo*⁸, juegan un rol muy importante en la implementación de las estrategias de control y por ese motivo serán tratadas con más detalle en esta tesis.

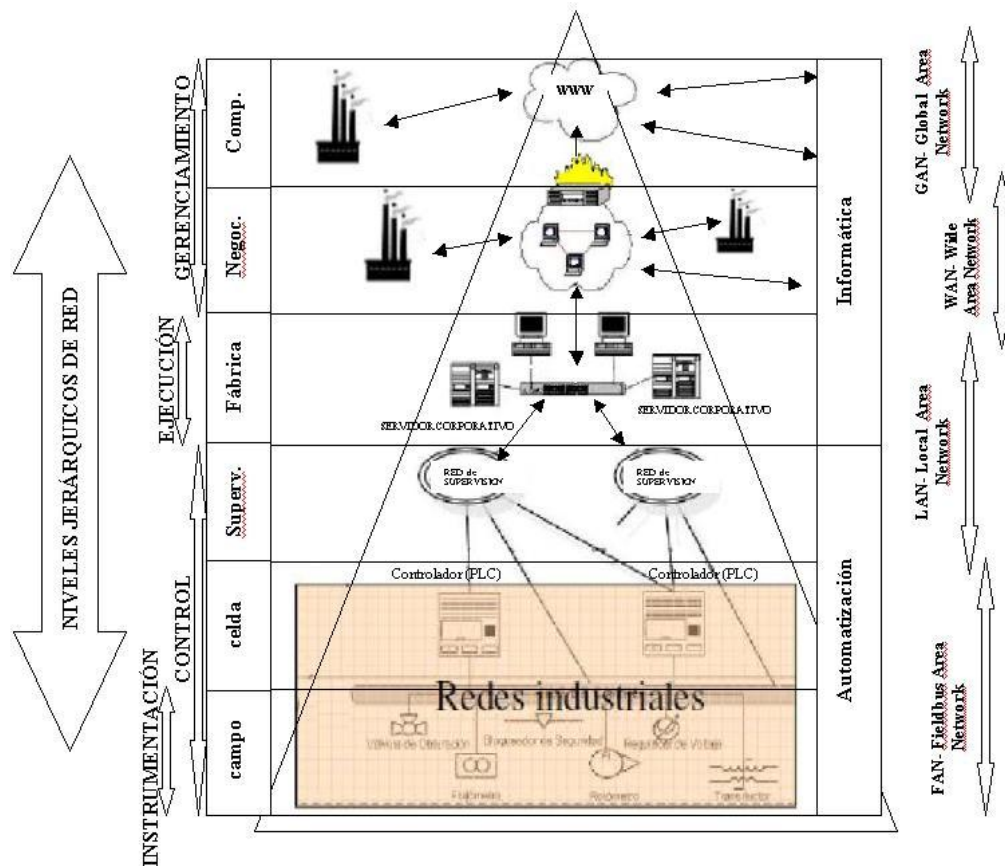


Figura 3.1: Ubicación de las redes de campo dentro de las redes de comunicación

3.2 Redes de Campo

El significado común de una red de campo es “una red para conectar los dispositivos de campo como sensores, actuadores y reguladores de campo como PLCs, reguladores, controladores de dispositivos, etcétera”. Pero esto es

⁸ *Fieldbuses* en ingles

solamente una definición informal. Una red de campo es una clase de sistema de comunicación en *tiempo-real* y está basado en una estructura en capas deducida del modelo OSI⁹ de siete capas ([4]). Sobre cada capa del modelo se puede elegir entre diversas opciones como, por ejemplo, el protocolo de cada uno de los servicios a utilizar. Algunos de los servicios definidos para cada tipo de red de campo son muy similares (permiten la comunicación de datos entre varios dispositivos); pero los protocolos elegidos proporcionan diversas calidades del servicio esencialmente en el término de eficiencia en *tiempo-real* y confiabilidad de las comunicaciones.

Otros objetivos de las redes de campo dependen de los criterios de diseño elegidos por sus diseñadores. Uno de estos objetivos es simplificar el cableado entre dispositivos de campo, mientras que otro es ser el sustento (columna espinal) en la implementación de sistemas de control distribuidos.

3.2.1 Requerimientos de una red de campo

Como se mencionó anteriormente, existen diversos factores por los cuales las redes de campo están siendo utilizadas cada vez con mayor frecuencia en las distintas aplicaciones industriales. Entre las diversas características que brinda una red campo, podemos citar:

- **Facilidad de mantenimiento:** esta característica permite una fácil detección de fallas, reparación y configuración de sus componentes.
- **Modularidad:** permite una eficiente escalabilidad de los dispositivos de campo. El cableado de campo se simplifica y facilita la tarea de agregar nuevos componentes como sensores, controladores y actuadores, y modificar los existentes.
- **Capacidad de conexionado con dispositivos de otros fabricantes:** debido a que los protocolos y servicios cumplen con estándares. De esta manera pueden lograrse mejores y más eficientes estrategias de actualización, innovación y ampliación del equipamiento instalado mediante las diferentes opciones que brinda el mercado.

⁹ El modelo OSI (Open Systems Interconnection) es una propuesta de la organización de estándares internacional (ISO). Define la forma en que se comunican los sistemas *abiertos* de telecomunicaciones basados en un modelo de 7 niveles.

- ***Eficiencia en la comunicación y bajo costo de cableado de sensores y actuadores:*** al permitir una mejor utilización del cableado realizado.

Por otro lado, existen restricciones temporales que las redes de campo utilizadas en ámbitos industriales deben satisfacer. Entre las principales restricciones temporales a satisfacer se encuentran: la *periodicidad*, el *jitter*¹⁰, el *tiempo de respuesta* de la aplicación, y diferentes simultaneidades o coherencias de tiempo de acciones o de datos. Las restricciones de tiempo pueden ser *absolutas* (expresadas con el uso del tiempo) o *relativas* (expresadas relativamente a un evento previo en términos de un intervalo):

- ***Periodicidad:*** la mayor parte del tráfico está representado por el “identificador de datos” el cual debe ser periódicamente transmitido. El período debe ser cumplido para satisfacer las condiciones de muestreo de control digital.
- ***Jitter:*** el período de muestreo en un sensor puede no ser satisfecho en la recepción del mensaje debido a retardos variables en la red de datos. Esta variación en el periodo de recepción del dato se indica como *jitter*.
- ***Tiempo de respuesta:*** el tiempo de respuesta es el transcurrido entre una demanda y una respuesta. En una red de comunicaciones, este tiempo es generalmente definido como el tiempo transcurrido entre la variación de la lectura de un sensor y su recepción en el nodo de procesamiento.
- ***Coherencia de tiempo:*** la coherencia de tiempo es una propiedad relativa a dos o más ocurrencias de eventos las cuales deben suceder en una ventana de tiempo dada. Esto puede ser usado para especificar acciones simultáneas como producciones de valores, transmisiones de datos, disparo de tareas, etcétera. Al existir retardos en la comunicación de datos, la ocurrencia de eventos simultáneos pueden

¹⁰ En transmisión de redes, **jitter**, hace referencia a la variación en el arribo de paquetes transmitidos periódicamente.

no ser procesados como tales y en consecuencia se pierde la coherencia temporal, pudiendo provocar efectos no deseados.

3.2.2 Características de tráfico

Una red de campo debe proporcionar los servicios requeridos por una determinada aplicación, garantizando las restricciones temporales de la misma. Según el estado en que se encuentre la aplicación, puede ser necesario satisfacer diferentes restricciones temporales. Por ejemplo, si un controlador está en funcionamiento normal, las restricciones temporales deben ser satisfechas estrictamente para no provocar incoherencia de los datos procesados. Por otro lado, si el controlador está siendo configurado o en tareas de diagnóstico o mantenimiento, entonces las restricciones temporales a satisfacer pueden relajarse. Existen redes de campo que permiten privilegiar diferentes tráficos de acuerdo a las condiciones temporales que tenga cada uno de los datos. Privilegiar a uno de los tráficos perjudica a los restantes tráficos que comparten la red de comunicaciones.

3.2.3 Redes de campo en tiempo real

En un sistema distribuido en *tiempo-real*, los mensajes transmitidos a través de la red tienen restricciones de tiempo que deben ser satisfechas. Los sistemas en *tiempo-real* deben producir respuestas correctas en un tiempo específico, porque un resultado producido fuera de tiempo es inútil incluso si fue computado correctamente.

Una práctica corriente para construir sistemas de *tiempo-real* centralizados es elegir un PLC robusto y de última generación (los PLC's se han beneficiado de la evolución de los microprocesadores y son cada vez más pequeños, más robustos y con mejor desempeño). Los fabricantes de PLC's proponen una gran variedad de tarjetas modulares de E/S con suficiente número de puertos E/S por tarjeta (8, 16, 24). El usuario sólo debe conectar cada sensor/actuador al correspondiente puerto de E/S y el software instalado en el PLC controlará el proceso según lo deseado. Otra ventaja es que las conexiones son en general estándar como la de 4-20 mA,

que en la práctica, es una de las conexiones estándar para la transmisión de señales analógicas.

Por otra parte, la mayoría de los fabricantes de controladores proponen tarjetas de comunicaciones para interconectar sus dispositivos, permitiendo que ellos intercambien variables. Cada controlador de un PLC, puede leer o escribir regiones de memoria de otros controladores conectados a la red de comunicaciones y el protocolo se asegura que cada cambio sea automáticamente refrescado en las otras memorias.

Sin embargo, los datos almacenados en la memoria de cada controlador deben preservar la coherencia temporal para evitar decisiones globales de control incorrectas. Es por este motivo, que las redes de campo deben garantizar su funcionamiento en *tiempo-real* y permitir la detección de errores en los datos para posibilitar la ejecución de estrategias de recuperación.

Por otro lado, el avance tecnológico ha permitido la evolución de sensores inteligentes que permiten incorporar capacidades de procesamiento y comunicación en los propios sensores de campo. De esta manera, es posible una mayor descentralización y capacidad de procesamiento de información de control.

3.2.4 Estándares industriales de redes de campo

Diversos fabricantes han propuesto diferentes redes de campo. Sin embargo, existen estándares que han sido adoptados por varios de los más importantes fabricantes de equipamiento industrial y por ello resultan ampliamente utilizados. A continuación se listan algunos de los más importantes con las características generales de sus estándares (algunos fabricantes pueden ofrecer características superiores en sus productos) -ciertos de ellos serán accesibles para su simulación en la plataforma propuesta en esta tesis-:

- **Interbus-S:** es un desarrollo especial para usar como el llamado “sensor bus”, que significa en un campo de medición interconectar sensores y actuadores con un camino simple, basado en RS-485. La capacidad de *tiempo-real* está especificada por un tiempo de respuesta menor a 4 ms para 1024 puertos de entrada/salida y 7 ms para 4096 puertos. La velocidad de transmisión de datos es de 300 kbit/s sobre un

bus de 400 m de largo. La máxima extensión permitida es 13 km y la distancia entre nodos 400 m.

- **Profibus (Process Field Bus):** usa DIN 19245 y permite un largo máximo de bus de 1.2 km (4.8 km usando repetidores); los *stubs*¹¹ no deben exceder 0.3 m. La velocidad de transmisión llega hasta 500 kbit/s.
- **Measurement Bus:** está basado en ISO 8482 e ISO 1745 y es definido para 4 cableados RS-485 y hasta 500 m de largo del bus, 5 m entre *stubs* y una velocidad de transmisión de 1 Mbit/s. El modo de operación es por interrogación (*polling*) y asíncrono para 32 nodos. La detección de errores está basada en paridad (DIN 66022), chequeo de bloque (DIN 66219), y chequeo por CRC-16, este último en el caso de código independiente.
- **FIP (Factory Instrumentation Protocol):** es una red de campo abierta, con propiedades de difusión. Cada participante es receptor de mensajes que son datagramas sin reconocimiento. Los marcos de datos (*data frames*) no incluyen direcciones. El administrador del bus tiene que especificar la dirección fuente por medio de un marco de comando (*command frame*). Debe conocer desde la definición de aplicación, qué dirección debe recibir el mensaje; esto significa que todos los participantes tienen que conocer las fuentes que deben observar. El largo máximo es 2 km con 256 estaciones; la velocidad de transmisión de datos es 31.25 kbit/s, 1Mbit/s ó 2.5 Mbit/s. Son permitidos pares trenzados o fibra óptica.
- **CAN-Bus:** es un desarrollo para automóviles que fue definido como “ambiente de *tiempo-real* con interrupciones controladas”. La red de control de área (CAN) es una arquitectura multi-maestro con transmisión asíncrona. El método de acceso es CSMA/CD¹² pero con *bit-wise arbitration*¹³. Esto garantiza que en cualquier caso un mensaje

¹¹ Los **stubs** son los cables que conectan a los nodos profibus con el medio de transmisión principal (el bus)

¹² CSMA/CD: Carrier Sense with Multiple Access and Collision Detection

¹³ *Bit-wise arbitration*: arbitración entre bits (tiene que ver con cómo los nodos CAN pueden acceder al medio compartido)

de alta prioridad va a pasar (pero aún es CSMA/CD). El esquema de direccionamiento es basado en contenidos, debido al uso de identificadores. El tiempo de latencia en mensajes de alta prioridad es de 150 μ s. Pueden ser definidos 2032 mensajes de hasta 8 bytes. La velocidad de transmisión es de 1 Mbit/s sobre 40 m.

3.3 Conclusión

En este capítulo se describieron los principales conceptos de redes de campo. Estas redes son ampliamente utilizadas en la industria y su funcionamiento en *tiempo-real* debe ser cuidadosamente analizado para garantizar que las restricciones temporales de la aplicación serán satisfechas. La plataforma propuesta en esta tesis utiliza herramientas de simulación que permiten modelar el comportamiento de algunas de las redes de campo más utilizadas en la industria.

Capítulo 4

Simulación de Sistemas de Control en Tiempo-Real

4.1 Introducción

Existen varias herramientas computacionales que permiten simular el comportamiento de un sistema a partir de su modelo matemático. Una de las herramientas más difundidas es el software Matlab[®] con su ambiente de simulación Simulink[®], de la firma MathWorks. El ambiente de simulación Matlab[®]/Simulink[®] permite el modelado de diversos sistemas mediante la incorporación o programación de librerías para tal fin.

El diseño tradicional de control usando Matlab[®]/Simulink[®], desatiende a menudo los efectos temporales que se presentan en la puesta en práctica real de los controladores. Hoy en día, las funciones que debe realizar un controlador son

implementadas como tareas en un núcleo de *tiempo-real* y se comunican a menudo con otros nodos a través de una red de campo. Por lo tanto, las restricciones que impone la implementación utilizando un controlador real (la velocidad de procesamiento del controlador y el ancho de banda de red están limitados) deben ser tomadas en cuenta en tiempo de diseño.

Para este propósito fue desarrollado TrueTime, una librería para la simulación de sistemas de control distribuidos en *tiempo-real*. TrueTime hace posible simular el comportamiento temporal de los núcleos en *tiempo-real* que ejecutan tareas del controlador. TrueTime también permite simular modelos simples de protocolos de red y su influencia en lazos de control conectados a través de una red de comunicaciones.

Por otro lado, el protocolo de comunicaciones OPC permite generar ambientes de simulación de aplicaciones, para la verificación de la correcta implementación de estrategias en controladores reales. Para implementar en esta tesis el protocolo OPC se utilizan el toolbox OPC de Matlab[®] y el software MatrikonOPCSimulation de Matrikon[®]. El software del toolbox OPC de Matlab[®] es una colección de funciones que amplían la capacidad del entorno de Matlab[®], y de bloques que extienden el entorno de simulación Simulink[®]. Usando bloques y funciones del toolbox OPC, se pueden adquirir vía OPC datos en tiempo real directamente en el ambiente de Matlab[®] y Simulink[®], y se pueden escribir datos directamente desde Matlab[®]-Simulink[®] al servidor OPC. Por otra parte, por medio del software de la empresa Matrikon, se implementa un servidor OPC. OPC es un sistema de comunicaciones para manejo, almacenaje y transporte de datos, basado en estándares, y que da una solución de bajo costo en comparación con interfaces propietarias.

4.2 TrueTime

TrueTime es una librería desarrollada en Matlab[®]/Simulink[®] para la descripción de sistemas de control de *tiempo-real* ([5]). En TrueTime se puede definir el comportamiento temporal de núcleos multi-tarea de *tiempo-real* que

contienen tareas de control para estudiar los efectos de planificación de CPU y redes sobre aplicaciones de control.

El núcleo de *tiempo-real* simulado está basado en eventos y puede manipular interrupciones externas. Pueden ser definidas diferentes políticas de planificación basadas en vencimientos o basadas en prioridades. Las tareas de control pueden ser implementadas usando funciones programadas en C, en Matlab[®], o usando diagramas de bloques de Simulink[®]. El tiempo de ejecución de las tareas de control puede ser modelado de dos maneras: constante o variante en el tiempo usando distribuciones de probabilidad. Son considerados efectos tales como cambio de contexto y manipulación de interrupciones, así como tareas de sincronización usando eventos y monitores. Con TrueTime también es posible simular el comportamiento temporal de redes de comunicación usadas, por ejemplo, en redes de lazos de control.

TrueTime puede ser utilizado con varios propósitos: para investigar los efectos reales del temporizado no-determinístico sobre aplicaciones de control, para desarrollar esquemas de compensación que ajustan la dinámica del controlador basándose en medidas de las variaciones de tiempo actuales, para experimentar con planificadores flexibles y dinámicos, y para simular sistemas de control basados en eventos.

4.2.1 Ambiente de simulación

TrueTime consiste en una librería con un bloque del núcleo de la computadora y un bloque de la red, entre otros bloques, como se muestra en la Figura 4.1. Los bloques son paso-variable, discretos, *S-Funcions* de Matlab[®] escritas en C++. Las interfaces de tales bloques son mostradas en la Figura 4.2.

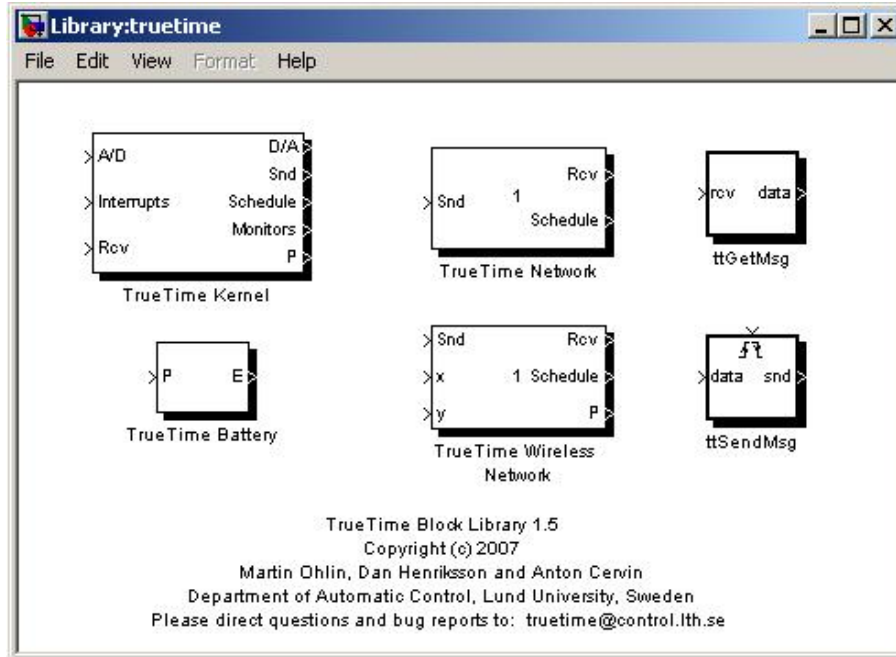


Figura 4.1: Librería de TrueTime. Las salidas Monitor y Diagramador exhiben la asignación de los recursos comunes (CPU, monitores, red) durante la simulación

Se asume que las señales de entrada son discretas, excepto las señales conectadas al puerto *A/D* las cuales pueden ser continuas. Todas las señales de salida son discretas. Los puertos *Schedule* y *Monitors* proporcionan gráficos de la asignación de los recursos comunes (CPU, monitores, red) durante la simulación.

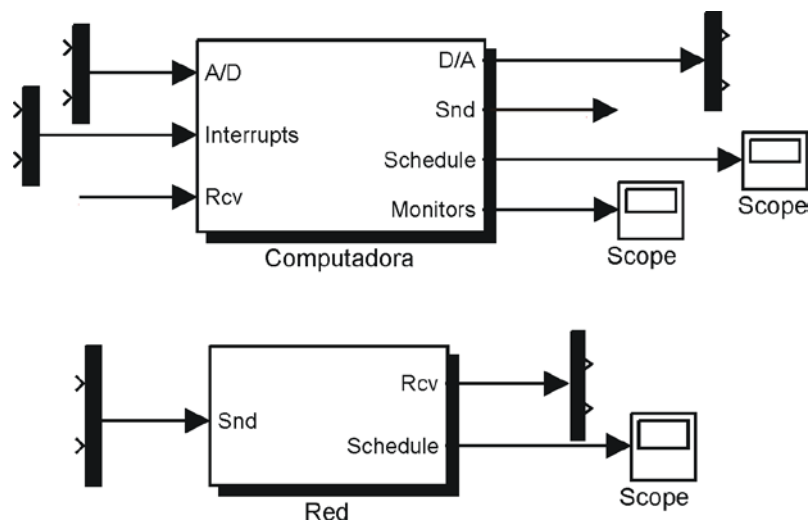


Figura 4.2: Interfaces de los bloques Simulink[®]. Los puertos *Schedule* y *Monitors* proporcionan gráficos de la asignación de los recursos comunes (CPU, monitores, red) durante la simulación

El bloque de la computadora ejecuta tareas definidas por el usuario y manejadores de interrupciones que representan, por ejemplo, tareas de entrada/salida, algoritmos de control e interfaces de red. El tiempo de ejecución simulado en el código se puede modelar como constante, al azar o dato-dependiente. Además, la política de diagramación en *tiempo-real* del núcleo es arbitraria y decidida por el usuario.

El bloque de red distribuye mensajes entre los nodos de computadoras de acuerdo al modelo de red elegido (perfil de red). Son soportados seis de los protocolos de control de acceso al medio más comunes: (CSMA/CD (Ethernet), CSMA/AMP (CAN), Round Robin (Token Bus), FDMA, TDMA (TTP) y Switched Ethernet)¹⁴. También es posible especificar parámetros de red tales como velocidad de transmisión, retardos de pre- y pos-procesamiento, sobrecarga de los marcos, y probabilidad de pérdidas.

Ambos bloques son manejados por eventos, con la ejecución determinada tanto por eventos internos como externos. Los eventos internos son temporales y corresponden a los acontecimientos tales como “un contador de tiempo ha expirado”, “una tarea ha acabado su ejecución”, o “un mensaje ha terminado su transmisión”. Los eventos externos corresponden a las interrupciones externas, por ejemplo “un mensaje llegó a la red” o “el ángulo pasó el grado cero”.

El nivel de detalle de la simulación es elegido por el usuario – no es a menudo, ni necesario, ni deseable simular la ejecución del código a nivel de instrucción ni la transmisión de mensajes a nivel de bit. Además, TrueTime permite la simulación de la conmutación de contexto y sincronización de tareas usando eventos o monitores.

Bloque Computadora

El bloque de la computadora simula una computadora con un núcleo en *tiempo-real* simple pero flexible, conversores *A/D* y *D/A*, una interfaz de red, y canales externos de interrupción.

¹⁴ CSMA/CD: Carrier Sense with Multiple Access and Collision Detection, CSMA/CA: Carrier Sense with Multiple Access and Collision Avoidance, FDMA: Frequency Division Multiple Access, TDMA: Time Division Multiple Access.

Internamente, el núcleo mantiene varias estructuras de datos que se encuentran comúnmente en un núcleo en *tiempo-real*: una cola de procesos listos, una cola de tiempo, y expedientes (*records*) para las tareas, manejadores de interrupciones, monitores y contadores de tiempo (*timers*) que se han creado para la simulación.

La ejecución de tareas y de los manejadores de interrupciones es definida por funciones de código escritas por el usuario. Estas funciones se pueden escribir en C++ (por velocidad) o como los m-archivos de Matlab[®] (por facilidad de uso). Los algoritmos de control pueden ser también definidos gráficamente usando los diagramas de bloque discretos ordinarios de Simulink[®].

Tareas:

La tarea es la construcción principal en el ambiente de simulación de TrueTime. Se utilizan las tareas para simular tanto actividades periódicas, por ejemplo controladores y tareas de entrada/salida, como actividades aperiódicas, por ejemplo tareas de comunicación y controladores manejados por eventos. Puede ser creado un número arbitrario de tareas para correr en el núcleo TrueTime. Cada tarea es definida por un conjunto de atributos y una función de código. Los atributos incluyen un nombre, un tiempo de finalización (*release*), peor caso de tiempo de ejecución, vencimientos relativos y absolutos, una prioridad (si se utiliza un diagramador de prioridad fija) y un período (si la tarea es periódica). Algunos de estos parámetros, por ejemplo el tiempo de finalización y el vencimiento absoluto, son actualizados constantemente por el núcleo durante la simulación. Otros atributos, por ejemplo período y prioridad, son normalmente mantenidos constantes pero pueden ser modificados a través de primitivas de llamadas al núcleo mientras la tarea se está ejecutando. De acuerdo a [6], es posible asociar dos manejadores de sobrecarga a cada tarea: un *manejador de vencimiento* (disparado si la tarea pierde su vencimiento) y un *manejador de tiempo de ejecución* (disparado si la tarea se ejecuta más tiempo que el peor caso de tiempo de ejecución).

Interrupciones y Manejadores de Interrupciones:

Las interrupciones pueden ser generadas de dos maneras: externamente o internamente. Una *interrupción externa* se asocia a uno de los canales de

interrupción externa del bloque de la computadora. Se acciona la interrupción cuando la señal del canal correspondiente cambia de valor. Este tipo de interrupción se puede utilizar, por ejemplo, para simular los controladores de motores que se muestrean contra la rotación del motor o los controladores distribuidos que se ejecutan cuando las mediciones llegan a la red. Las *interrupciones internas* se asocian a los contadores de tiempo (*timers*). Pueden ser creados contadores de tiempo periódicos y contadores de tiempo paso a paso. La interrupción correspondiente se acciona cuando expira el contador de tiempo. Los contadores de tiempo también son utilizados internamente por el núcleo para implementar los manejadores de sobrecarga.

Cuando ocurre una interrupción externa o interna, un manejador de interrupciones definido por el usuario es diagramado para servir a la interrupción.

Un manejador de interrupciones trabaja de la misma manera que una tarea, pero es diagramado en un nivel de prioridad más alto. Los manejadores de interrupciones realizarán normalmente tareas pequeñas y que consumen poco tiempo, por ejemplo, la generación de un evento o accionar la ejecución de una tarea.

Un manejador de interrupciones está definido por un nombre, una prioridad, y una función de código. Las interrupciones externas también tienen un estado latente durante el cual son insensibles a las nuevas invocaciones.

Diagramación y prioridades:

La ejecución simulada ocurre en tres niveles distintos de la prioridad: el nivel de interrupción (la prioridad más alta), el nivel de núcleo, y el nivel de tarea (la prioridad más baja). La ejecución puede ser apropiativa (es decir con desalojo) o no-apropiativa; esto se puede especificar individualmente para cada tarea y cada manejador de interrupción. En el nivel de interrupciones, un manejador de interrupciones es diagramado de acuerdo a prioridades fijas. Al nivel de la tarea pueden ser usadas prioridades dinámicas. En cada punto de diagramación, la prioridad de una tarea está dada por una función de prioridad definida por el usuario, la cual es función de los atributos de la tarea. Esto hace fácil simular diversas políticas de diagramación. Por ejemplo, una función de prioridad que devuelve un número de prioridad implica una diagramación de prioridad fija,

mientras que una función de prioridad que retorna un vencimiento, implica un diagramador manejado por vencimientos. Las funciones de prioridad predefinidas existen para la mayor parte de los esquemas de diagramación comúnmente usados.

Código:

El código asociado a las tareas y a los manejadores de interrupciones es diagramado y ejecutado por el núcleo mientras progresa la simulación. El código puede interactuar con otras tareas y con el ambiente al principio de cada segmento de código. Este modelo de ejecución hace posible modelar retardos de entrada-salida, bloqueos cuando se accede a recursos compartidos, etc. El tiempo de ejecución simulado de cada segmento es devuelto por la función de código, y puede ser modelado como constante, al azar, o dato-dependiente.

Sincronización:

La sincronización entre las tareas es soportada por los monitores y los eventos. Los monitores se utilizan para garantizar la exclusión mutua al tener acceso a datos comunes. Los eventos se pueden asociar a los monitores para representar variables de condición. Los eventos pueden también estar libres (esto es, no asociados a un monitor). Esta característica se puede utilizar para obtener la sincronización entre las tareas donde no hay implicadas condiciones de datos compartidos.

Gráficos de salida:

Dependiendo de la simulación, diversos gráficos de salida son generados por los bloques de TrueTime. Cada bloque de computadora producirá dos gráficos: un diagramador de la computadora y un gráfico del monitor. El bloque de red producirá un diagramador de la red. El diagramador de la computadora exhibirá el trazo de la ejecución de cada manejador de tareas y de cada tarea durante el curso de la simulación. Si se simula la conmutación de contexto, el gráfico también exhibirá la ejecución del núcleo. En la Figura 4.3, se ve un ejemplo de tal rastro de la ejecución. Si la señal es alta, significa que la tarea está *ejecutándose*. Una señal media indica que la tarea está lista pero no corriendo (*lista*), mientras que una señal baja significa que la tarea está *ociosa*. De manera análoga, el diagramador de la red muestra la transmisión de mensajes sobre la red, con los estados

representando *enviar* (alto), *esperando* (medio), y *ocioso* (baja). El gráfico del monitor muestra qué tareas son las que se mantienen y que esperan en los diversos monitores durante la simulación. La generación de estos trazos de ejecución es opcional y puede ser especificada individualmente para cada tarea, manejador de interrupciones y monitor.

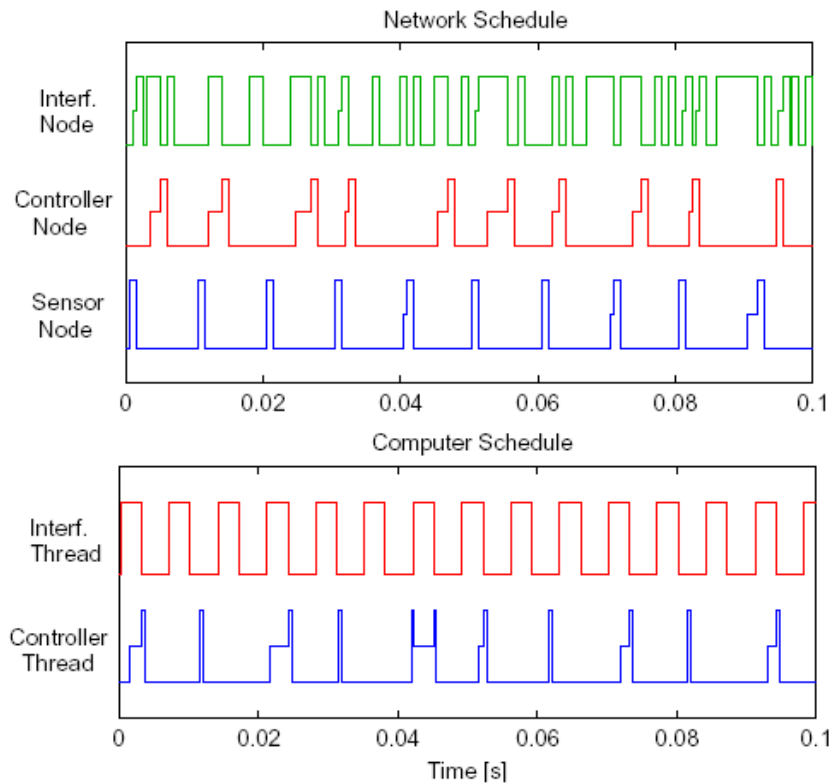


Figura 4.3: Ventanas de diagramador que muestran la asignación de recursos comunes: red (arriba) y nodo controlador (abajo). Una señal alta significa enviar o ejecutarse, una señal media significa esperar, y una señal baja significa ocioso

Bloque de red cableada

El bloque de la red es manejado por eventos y se ejecuta cuando los mensajes se incorporan o salen de la red. Un mensaje contiene la información tanto sobre el nodo de la computadora que envía como el de recepción, los datos arbitrarios del usuario (típicamente señales de medida o señales de control), la longitud del mensaje y los atributos opcionales en *tiempo-real* tales como una prioridad o un vencimiento.

En el bloque de red cableada, es posible especificar la velocidad de transmisión, el protocolo de control de acceso al medio (CSMA/CD (Ethernet), CSMA/AMP (CAN), Round Robin (Token Bus), FDMA, TDMA (TTP) o Switched Ethernet), y un número de otros parámetros (ver Figura 4.4). Un mensaje largo se puede dividir en paquetes que se transmiten en orden, cada uno con una sobrecarga adicional. Cuando la simulación de una transmisión de un mensaje ha terminado, se pone en la cola de recepción del nodo de la computadora destinataria del mensaje (es decir, en el *buffer*), el cual es notificado por una interrupción de hardware.

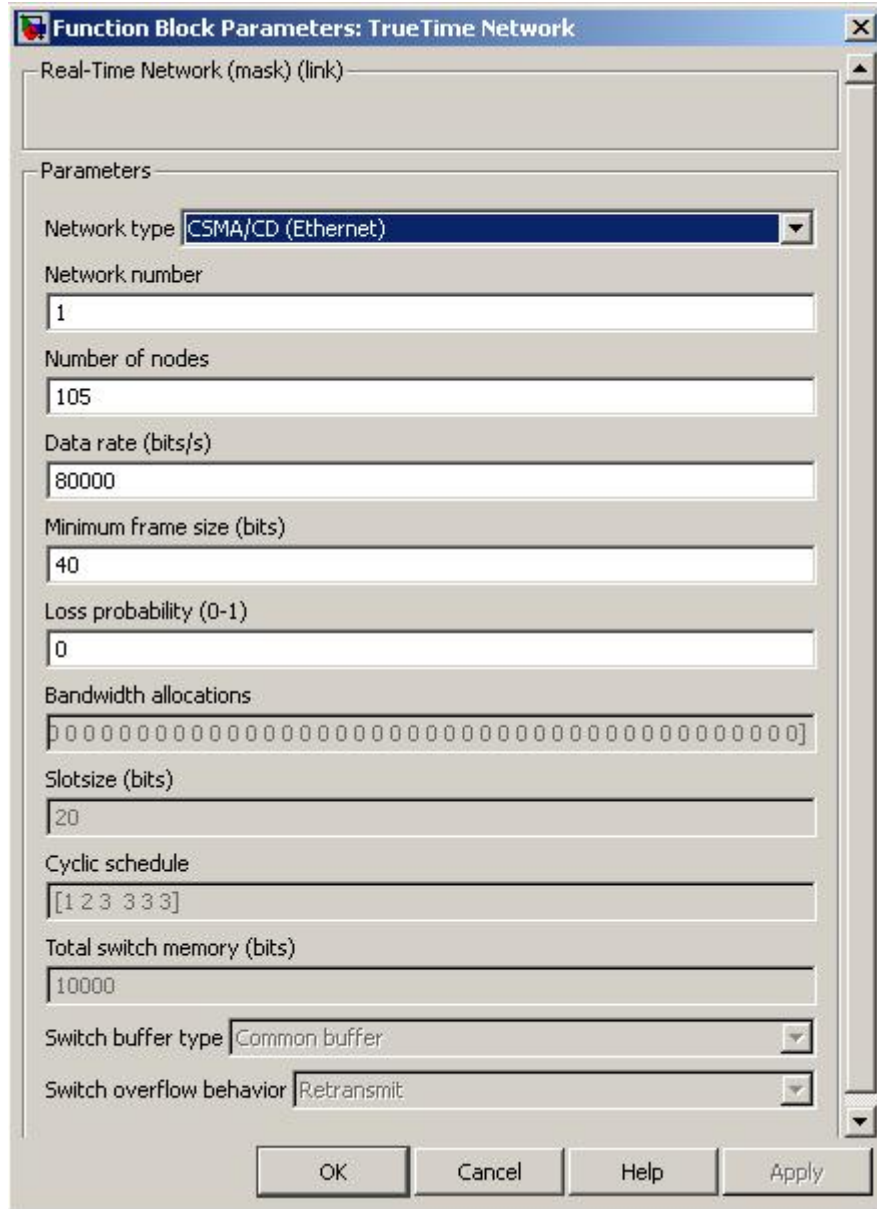


Figura 4.4: Ventana de diálogo en el bloque de red de TrueTime

4.3 Protocolo OPC

Para el efectivo desarrollo de aplicaciones modulares y con diversas funcionalidades para la industria, es necesario que las aplicaciones que se desarrollen tengan una comunicación óptima con los datos del proceso. Aunque en el pasado, esta tarea de integración era ardua y difícil, recientemente (la primera versión data de agosto de 1996) ha aparecido un estándar como OPC

(OLE for Process Control) [11] que permite la interconexión y el intercambio de datos entre dispositivos y aplicaciones software. La aplicación de la interface estándar OPC hace posible la interoperabilidad entre aplicaciones de automatización/control, dispositivos/sistemas de campo y aplicaciones de gestión/oficina. Para implementar el protocolo OPC en esta tesis, se utilizan el toolbox OPC de Matlab® y la aplicación MatrikonOPCSimulation brindado por la empresa Matrikon®.

4.3.1 Toolbox OPC de Matlab®

El software del toolbox OPC de Matlab® es una colección de funciones que amplían la capacidad del entorno de Matlab®, y de bloques que extienden el entorno de simulación Simulink®. Usando bloques y funciones del toolbox OPC, se pueden adquirir vía OPC datos en tiempo real directamente en el ambiente de Matlab® - Simulink®, y se pueden escribir datos directamente desde Matlab® - Simulink® al servidor OPC. En esta tesis se utilizan los bloques *OPC Config Real-Time*, *OPC Read* y *OPC Write*.

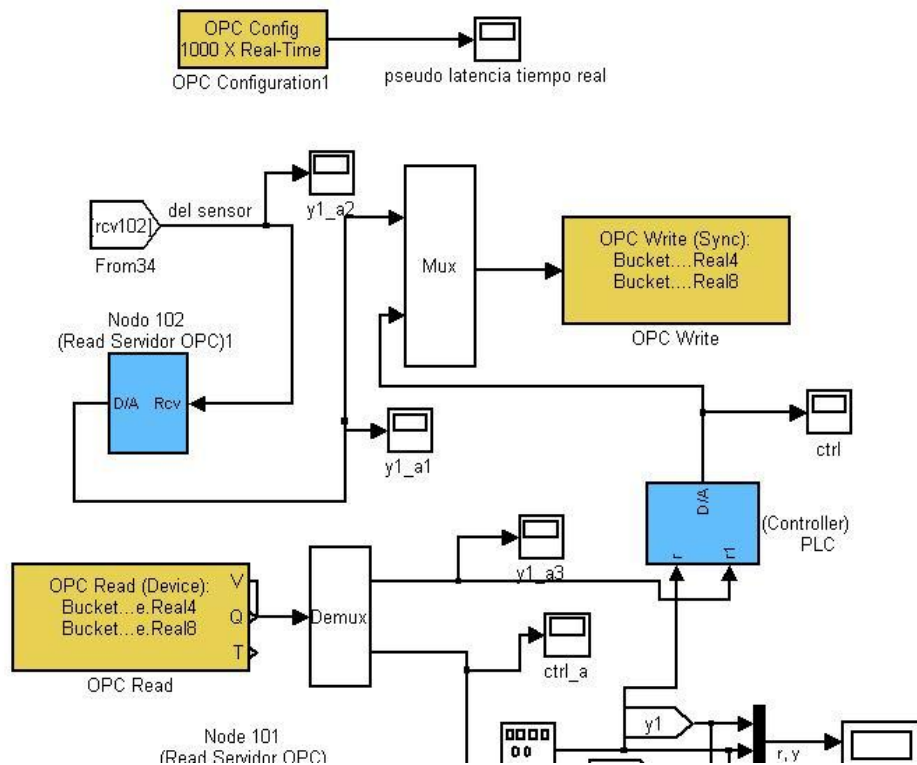


Figura 4.5: Bloques *OPC Config Real-Time*, *OPC Read* y *OPC Write*

Bloque *OPC Config Real-Time*:

El bloque de *OPC Config Real-Time* define los clientes OPC a ser usados en un modelo, configura el comportamiento en pseudo-tiempo real para el modelo, y define el comportamiento en respuesta a los errores OPC y a los eventos. En la Figura 4.6 se muestra la ventana de parámetros de dicho bloque.

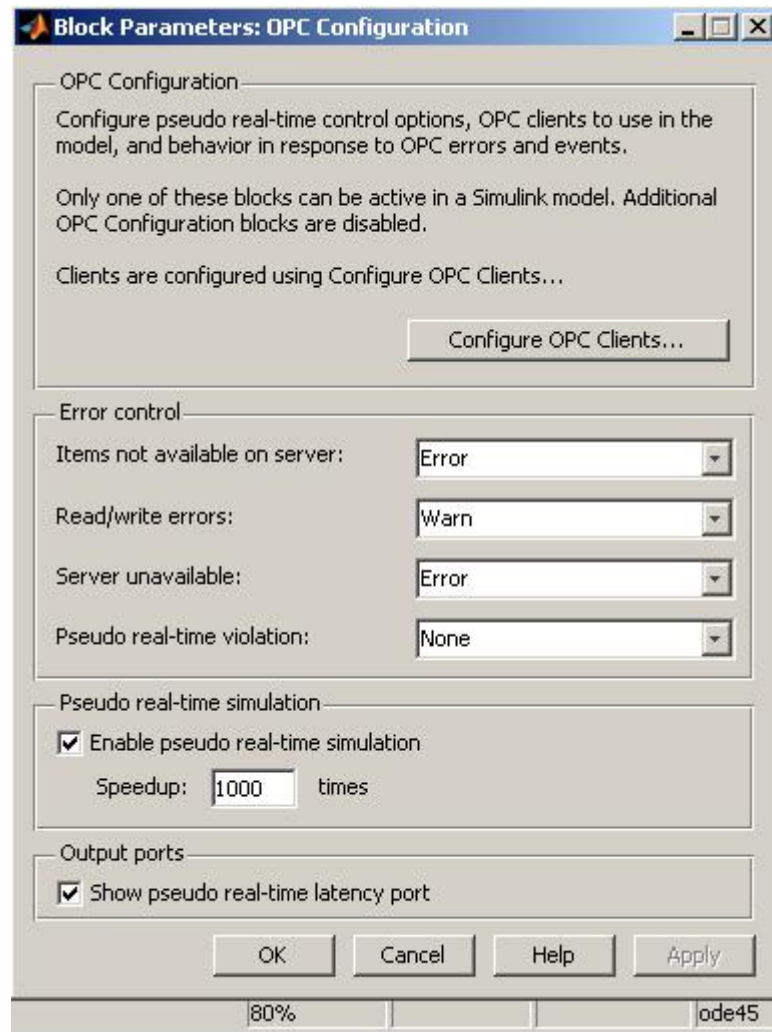


Figura 4.6: Ventana de parámetros del bloque *OPC Config Real-Time* de Matlab®

El bloque *OPC Config Real-Time* no tiene puertos de entrada. Un puerto de salida opcional muestra la latencia del modelo (tiempo de espera en cada paso de simulación para lograr el pseudo comportamiento en tiempo real).

En un mismo modelo, no se puede colocar más de un bloque *OPC Config Real-Time*. Si se intenta hacerlo, aparecerá un mensaje de error, y el segundo bloque *OPC Config Real-Time* será deshabilitado.

Bloque *OPC Read*:

El bloque *OPC Read* lee los datos de uno o más elementos que se hallen en un servidor OPC. La operación de lectura se lleva a cabo de forma sincrónica (del caché o desde el dispositivo) o asincrónica (del dispositivo). En la Figura 4.7 se muestra la ventana de parámetros de dicho bloque.

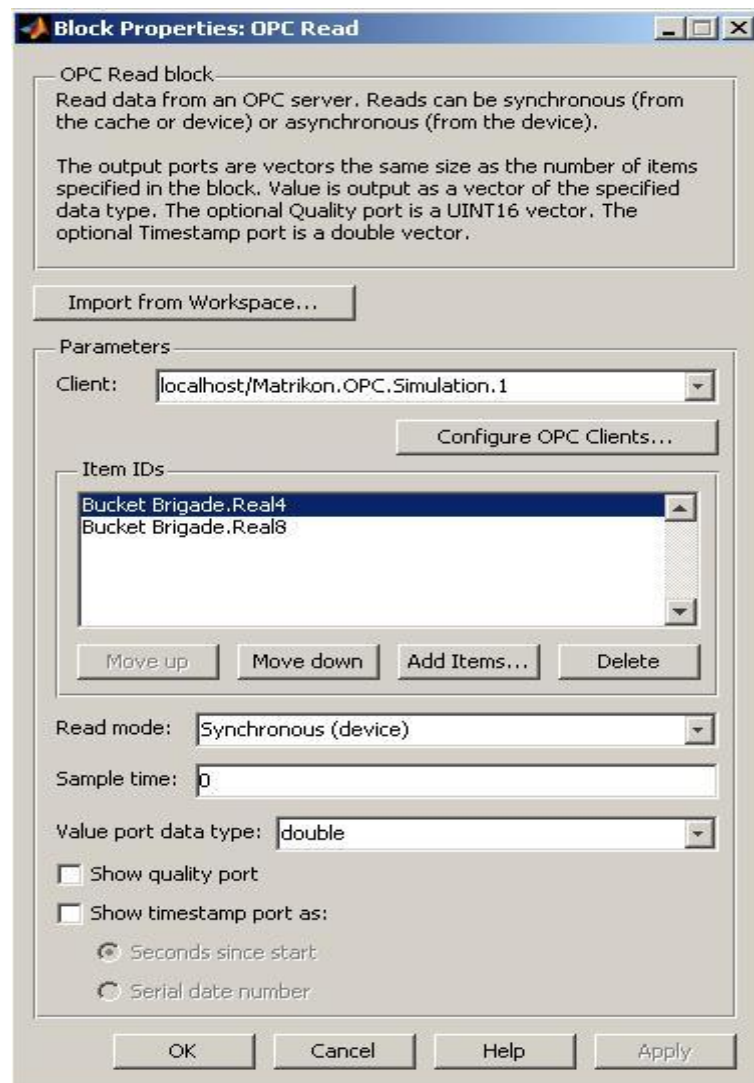


Figura 4.7: Ventana de parámetros del bloque *OPC Read* de Matlab®

Con respecto a las salidas del bloque, en la primera de ellas (*V*) se tienen los valores de los elementos solicitados. Opcionalmente, en salidas adicionales, se tienen los identificadores de salidas de calidad (*Q*) y las marcas de tiempo (*T*), asociados a cada valor de los datos dados en la primer salida. La salida de marca de tiempo puede ser dada como un número de serie de fecha (el tiempo del mundo real), o como el número de segundos desde el inicio de la simulación (tiempo de simulación).

Los valores *V*, *Q*, *T* presentados en los puertos de salida son los últimos datos conocidos para cada uno de los elementos leídos por el bloque. Se utiliza la salida de marca de tiempo para determinar cuándo esa muestra fue cambiada por última vez.

Bloque *OPC Write*:

El bloque *OPC Write* escribe datos en uno o más elementos en un servidor OPC. La operación de escritura se realiza de forma sincrónica o asincrónica. En la Figura 4.8 se muestra la ventana de parámetros de dicho bloque.

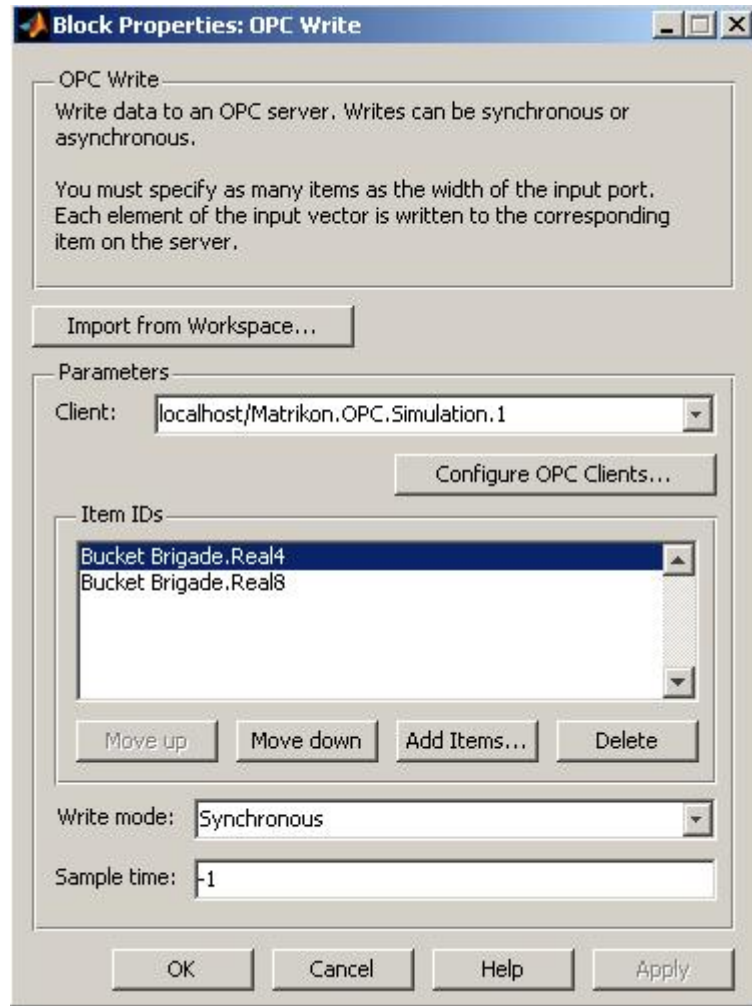


Figura 4.8: Ventana de parámetros del bloque *OPC Write* de Matlab®

Cada elemento del vector de entrada se escribe en su lugar correspondiente dentro de la lista de identificación de ítems definida por el bloque *OPC Write*.

4.3.2 Software Matrikon®

Por medio de la aplicación MatrikonOPCSimulation brindada por la empresa Matrikon®, se implementa un servidor OPC. Dicha aplicación es completamente funcional, y sin restricciones.

El servidor MatrikonOPCSimulation es una herramienta gratuita para ayudar a probar y solucionar problemas de aplicaciones OPC (clientes) y sus conexiones. Probar aplicaciones en servidores OPC reales puede resultar en la pérdida de los datos. El servidor MatrikonOPCSimulation crea un ambiente simulado, con lo

cual, en el caso de surgir algún problema, no se perderán los datos reales del proceso en estudio.

4.4 Conclusión

En este capítulo hemos descrito, por un lado, las principales características modeladas por la librería TrueTime. Esta descripción indica que TrueTime es una librería que permite la simulación de arquitecturas de procesamiento y comunicaciones sofisticadas y que pueden consistir en varios controladores interconectados mediante diversos tipos de redes de comunicaciones. Por lo tanto, se ha utilizado la librería TrueTime para el modelado de las arquitecturas de automatización en donde se verán inmersos los controladores bajo prueba.

Por otro lado, se ha mostrado cómo se implementa en la tesis el protocolo OPC. La incorporación de comunicación mediante este protocolo permite generar ambientes de simulación de aplicaciones para la verificación de la correcta implementación de estrategias en controladores reales.

Capítulo 5

Plataforma de Simulación de Sistemas de Control en Tiempo-Real

5.1 Introducción

En el capítulo anterior se describieron las facilidades que ofrece la librería TrueTime para la simulación de sistemas de control distribuidos en *tiempo-real*. La misma posibilita simular el comportamiento de los núcleos en *tiempo-real* que ejecutan tareas del controlador y también simular modelos simples de protocolos de red y su influencia en lazos de control conectados a través de la red.

Existen varios trabajos en donde se utilizan las diferentes características de TrueTime. En [19], se aplica al análisis de políticas de diagramación. En [20], se aplica al análisis de asignación de periodos a tareas de control que se ejecutan concurrentemente en un mismo controlador. En [16], al análisis y diseño de lazos de control en red con instantes de actuación sincronizados. En [17] y [18], se

utiliza TrueTime para la combinación de diferentes redes y el estudio de comportamiento de sistemas de control en cascada en red.

TrueTime fue concebido para análisis de sistemas de *tiempo-real*. La utilización de esta herramienta para el estudio de redes de campo en sistemas de control requiere un conocimiento profundo de los parámetros y características de *tiempo-real* de los bloques de TrueTime y su programación en Matlab[®] o C++.

La plataforma desarrollada posibilita analizar el comportamiento de redes de campo en sistemas de control, aún sin ser un experto en el manejo de Matlab[®] y/o TrueTime. La plataforma incluye la librería TrueTime para modelar el comportamiento de redes cableadas y temporizaciones de procesamiento y la librería OPC de Matlab[®] para una comunicación eficiente y genérica para controladores de campo. Trabajos anteriores ([11] y [12]) utilizan la comunicación OPC, pero para la simulación de casos particulares de aplicaciones sin la consideración del modelo de la arquitectura distribuida de un sistema de controladores.

5.2 Estructura de la Plataforma

La plataforma ha sido planificada para permitir el diseño e implementación del sistema a simular mediante la interacción del usuario con diversas pantallas de configuración. De esta manera, se pretende que el usuario no requiera conocer en profundidad la utilización de las numerosas funciones y estructuras de datos necesarias en este tipo de plataforma.

En la pantalla principal de la plataforma, el usuario determina la cantidad y tipo de nodos que va a utilizar (Figura 5.1). La pantalla principal permite comenzar con la simulación. Existen diversas subpantallas, algunas aparecen antes de comenzar la simulación (Figura 5.2) y otras mientras la simulación está corriendo (Figura 5.3).

La pantalla de la Figura 5.2 permite al usuario ingresar las funciones transferencia de la planta a simular. La simulación de la carga de comunicaciones producidas por diferentes nodos es configurada mediante la pantalla mostrada en la Figura 5.3. En este último tipo de pantalla pueden establecerse los parámetros

como tiempo de placa a buffer, tiempo de respuesta, etc. para cada nodo sensor, actuador, controlador o interferencia. Estas pantallas fueron realizadas en Guide¹⁵, con la intención de ofrecer al usuario una interfaz más amigable.

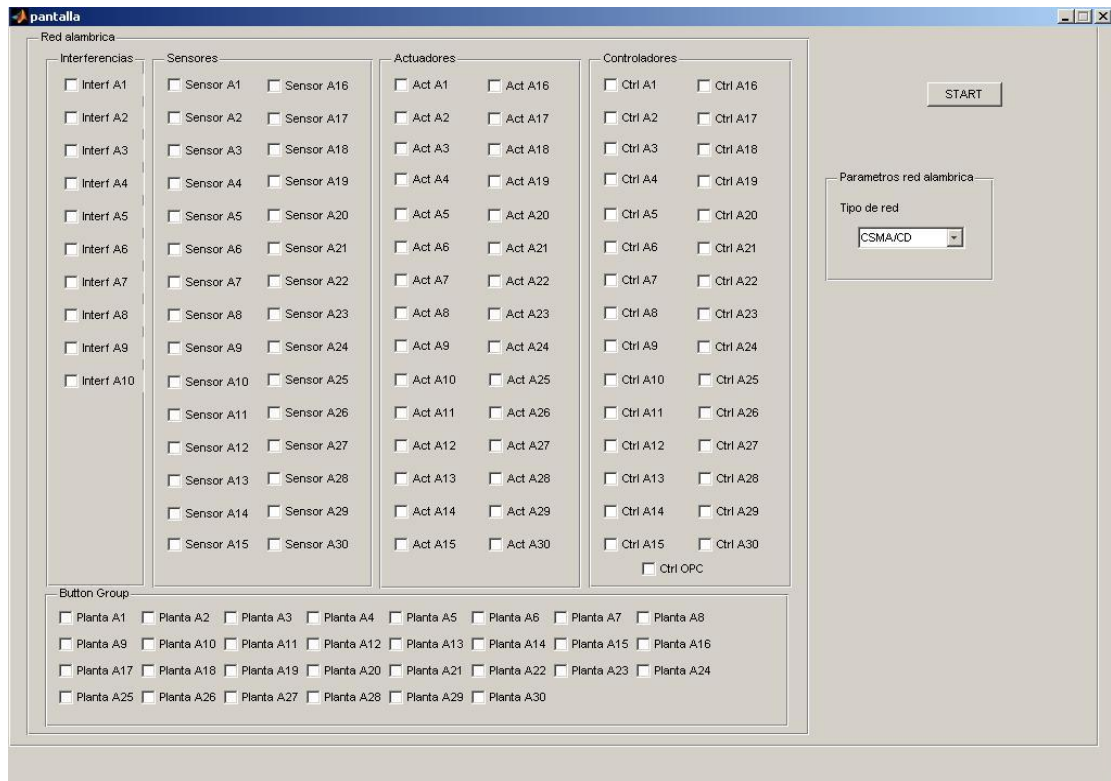


Figura 5.1: Pantalla gráfica principal de plataforma

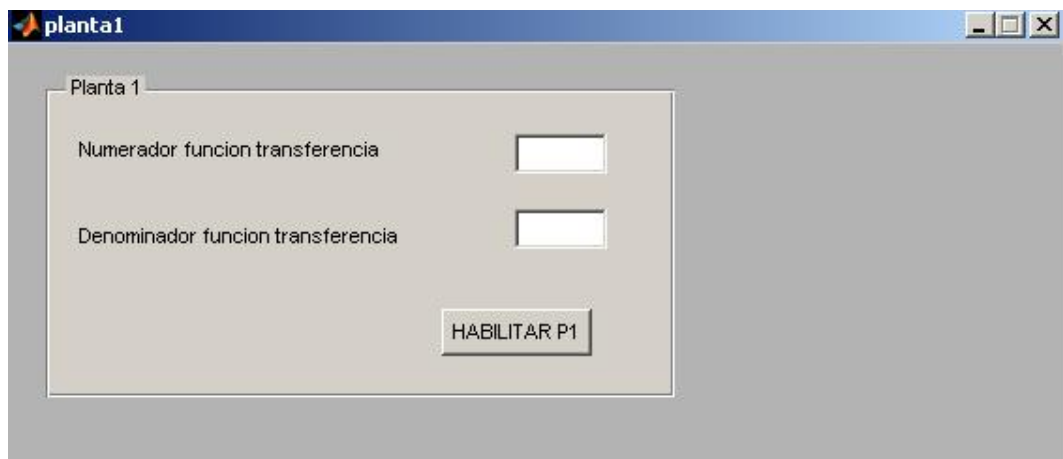


Figura 5.2: Subpantalla gráfica de la plataforma

¹⁵ El ambiente de desarrollo de Interfaz de Usuario en Matlab (Guide) contiene un conjunto de herramientas para crear interfaces graficas muy parecidas a las aplicaciones Windows. Estas herramientas simplifican el proceso de creación y de programación.



Figura 5.3: Subpantalla gráfica de la plataforma

La plataforma de simulación está implementada mediante el archivo de Simulink® “plataforma.mdl” (Figura 5.4a y 5.4b). Este archivo contiene toda la implementación de la plataforma y debe ser incorporado al ambiente de Simulink®.

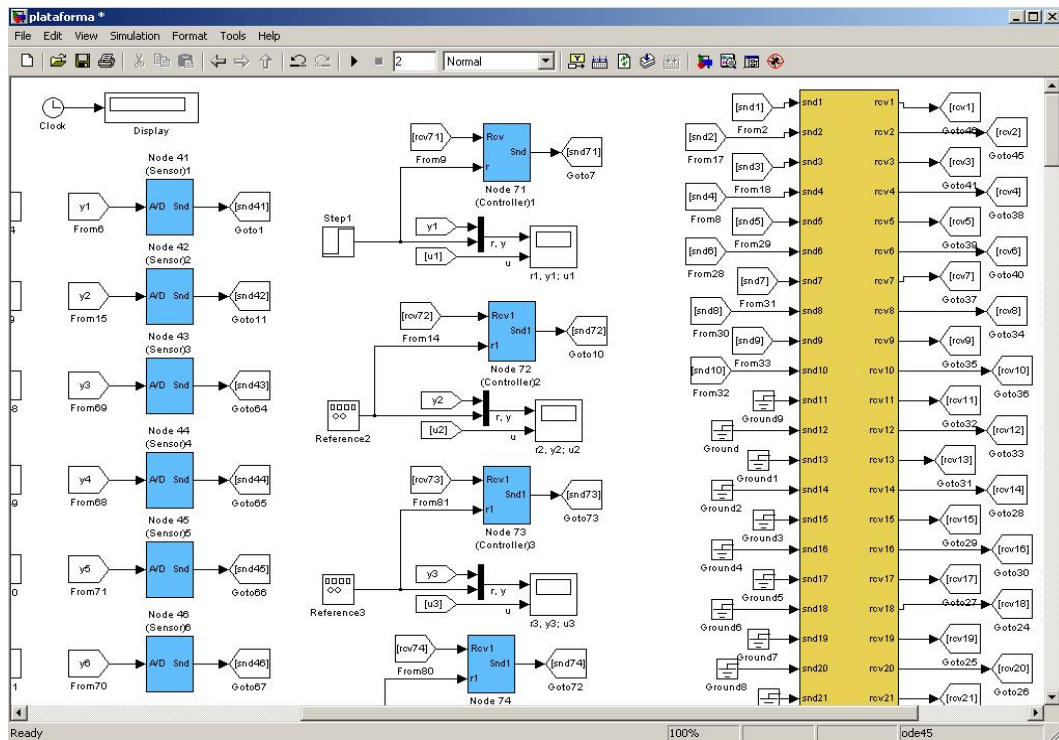


Figura 5.4a: Pantalla de Simulink® de la plataforma

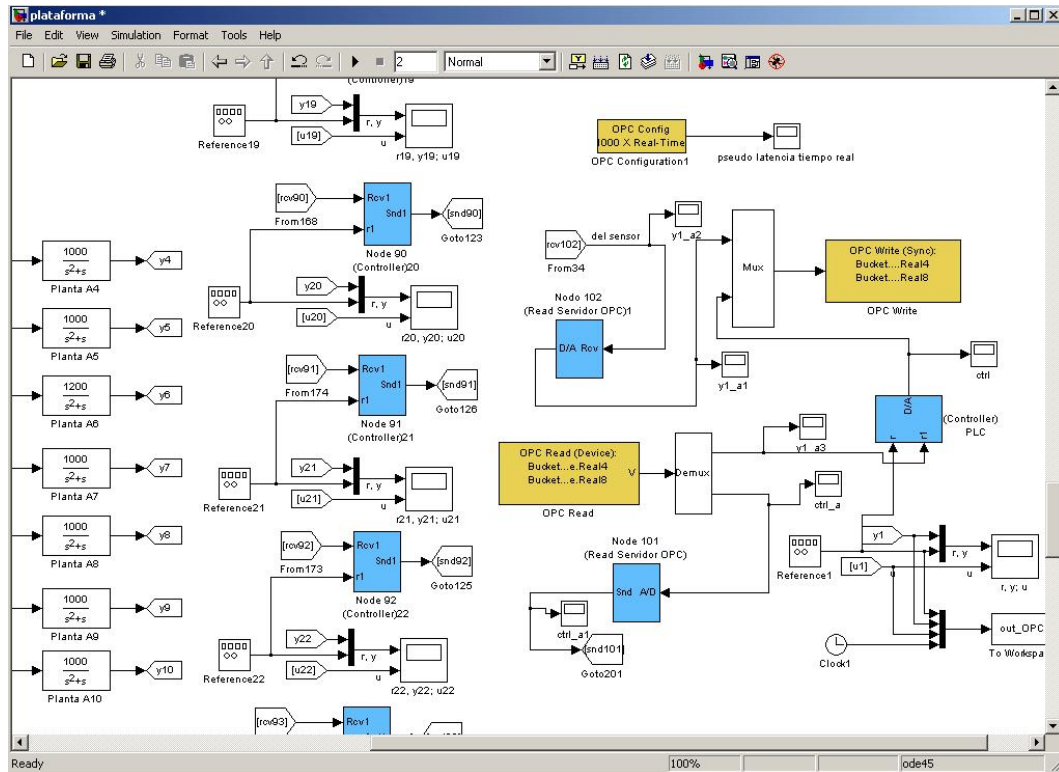


Figura 5.4b: Otra vista de la pantalla de Simulink® de la plataforma

La plataforma consta de un bloque de red cableada (bloque TrueTime Network de TrueTime), bloques sistema (plantas, bloques interferencia, bloques controlador, bloques sensor y bloques actuador: estos últimos cuatro bloques son, al fin y al cabo, bloques *kernel* de TrueTime) y bloques OPC ([13] y [14]). A su vez, ciertos bloques son dependientes de archivos de Matlab® (ubicados en el directorio donde está “plataforma.mdl”), como se detalla a continuación:

- el bloque sensor: “senscode_i.m”, “sensor_i_init.m”, “msgRcvSensor_i.m”.
- el bloque interferencia: “interfcode_i.m”, “interferente_i_init.m”, “msgRcvInterf_i.m”.
- el bloque controlador: “ctrlcode_i.m”, “controller_i_init.m”, “msgRcvCtrl_i.m”, “dummycode_i.m”.
- el bloque actuador: “actcode_i.m”, “actuator_i_init.m”, “msgRcvActuator_i.m”.
- el bloque OPC: “OPC_controller_init.m”, “OPC_ctrlcode.m”, “OPC_msgRcvRead.m”, “OPC_msgRcvRead2.m”, “OPC_pidcalc.m”,

“OPC_read_init.m”, “OPC_read2_init.m”, “OPC_readcode.m”,
 “OPC_readcode2.m”.

A cada bloque sensor, actuador, controlador (salvo el controlador que representa al PLC, que no está en red y el cual se comunica vía OPC) e interferencia cableados, le corresponde un nodo de la red. Cada nodo tiene un puerto de entrada *snd* desde el cual recibe información y un puerto de salida *rcv* en el cual vuelca información. Para los sensores, como salida *rcv* siempre se tendrá un bloque *Terminator*. Para los actuadores como entrada *snd* siempre se tendrá un bloque *Ground* (Figura 5.5b). En los demás casos se debe indicar de dónde proviene la entrada y hacia dónde va la salida (*snd* o *rcv* tendrán la numeración correspondiente a ese nodo, por ejemplo, para el nodo 9 el *From* indica *snd9* y el *Goto*, *rcv9*, como muestra la Figura 5.5a).

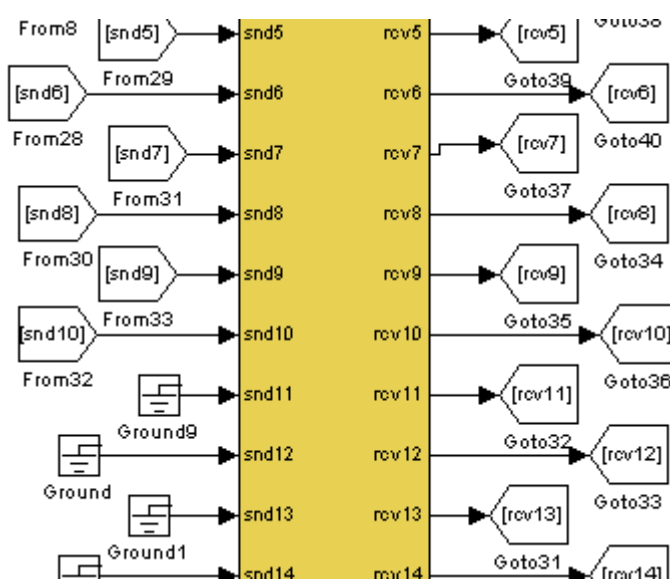


Figura 5.5a: Entradas y salidas de los nodos en la red

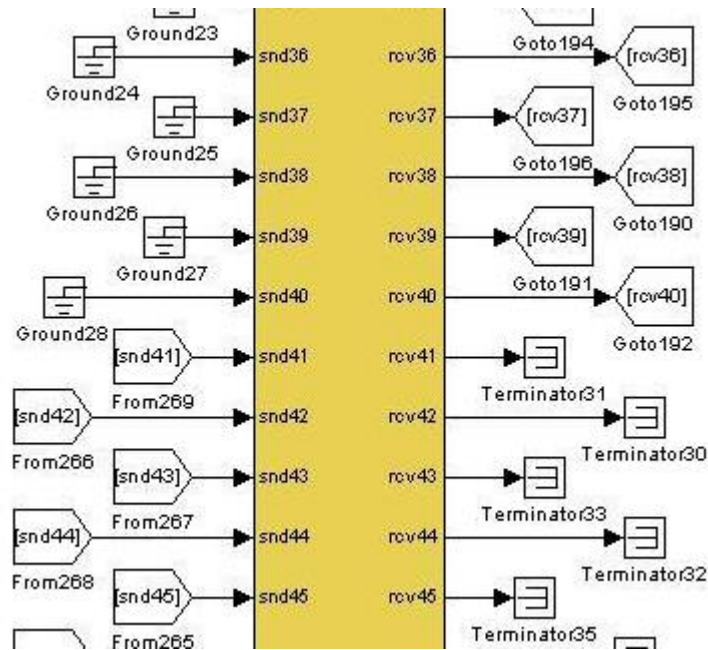


Figura 5.5b: Entradas y salidas de los nodos en la red

El bloque sensor cableado (Figura 5.6) recibe en su entrada *A/D* la señal y proveniente de la planta y la información de salida es enviada al puerto *snd* que le corresponde a ese sensor en la red (por ejemplo, para el sensor 1 como entrada se ve la señal y_1 proveniente de la planta 1 y como salida se ve la señal *snd41* que será la entrada en el nodo 41 de la red).

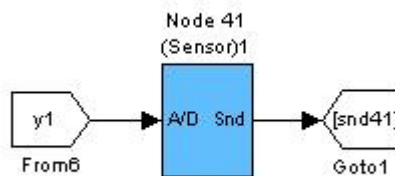


Figura 5.6: Bloque sensor cableado

El bloque actuador cableado (Figura 5.7) recibe en su entrada *rcv* la señal de su puerto de salida *rcv* en la red y como salida *D/A* tiene la señal u que es la entrada a la planta (por ejemplo, para el actuador 1 como entrada se ve la señal *rcv11* que es la salida del nodo 11 en la red y como salida se ve la señal u_1 que es la entrada a la planta 1).

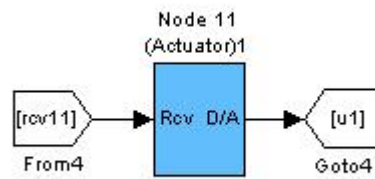


Figura 5.7: Bloque actuador cableado

El bloque controlador cableado (Figura 5.8) recibe en su entrada *rcv* la señal de su puerto salida *rcv* en la red, en su entrada *r* la referencia de la planta; y la información de salida *snd* es enviada al puerto de entrada del nodo correspondiente a ese controlador en la red (por ejemplo, para el controlador 1 como entrada *rcv* se ve la señal *rcv71* proveniente de la salida del nodo 71 en la red, como entrada *r* se ve la referencia tomada para la planta 1 y como salida se ve la señal *snd71* que será la entrada en el nodo 71 de la red).

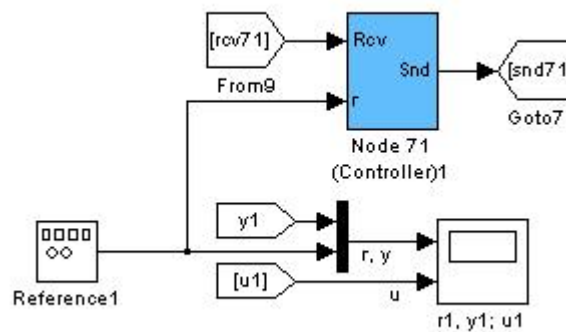


Figura 5.8: Bloque controlador cableado

El bloque interferencia cableada (Figura 5.9) recibe en su entrada *rcv* la señal de su puerto salida *rcv* en la red, y la información de salida *snd* es enviada al puerto de entrada del nodo interferencia en la red (por ejemplo para la interferencia 1 como entrada *rcv* se ve la señal *rcv1* proveniente de la salida del nodo 1 en la red y como salida se ve la señal *snd1* que será la entrada en el nodo 1 de la red).

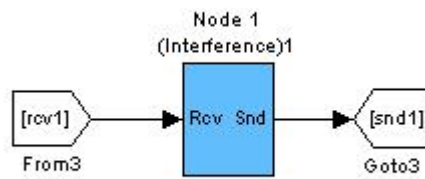


Figura 5.9: Bloque interferencia cableada

Para esos 4 bloques cableados (sensor, actuador, controlador e interferencia), se debe ingresar el nombre de la función de inicio del dispositivo (por ejemplo, para el sensor: `sensor_i_init`; Figura 5.10). El TrueTime Kernel que representa a cada bloque contiene diversos elementos como se puede observar en la Figura 5.11.

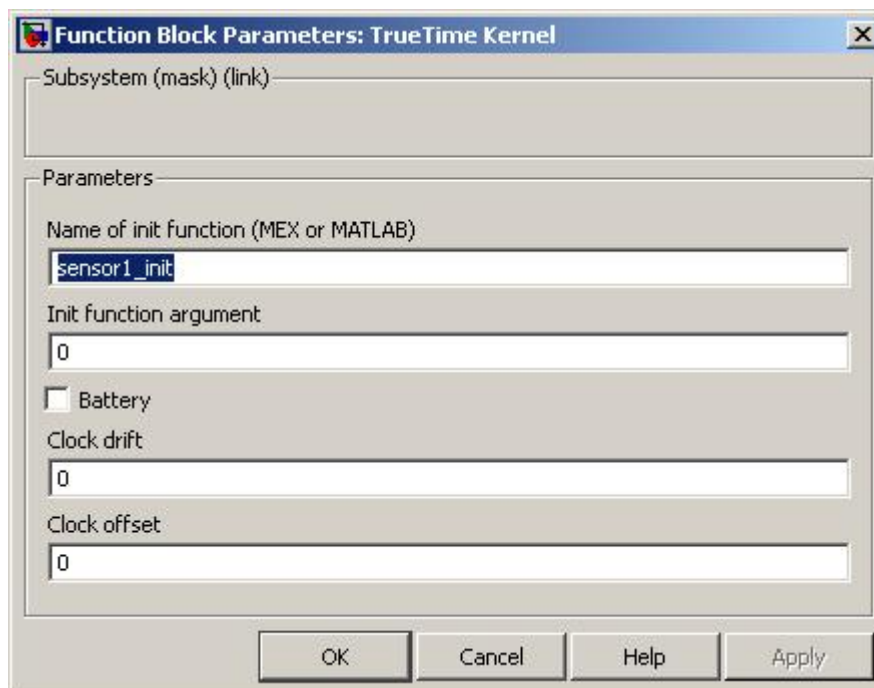


Figura 5.10: Ventana *Block Parameters*

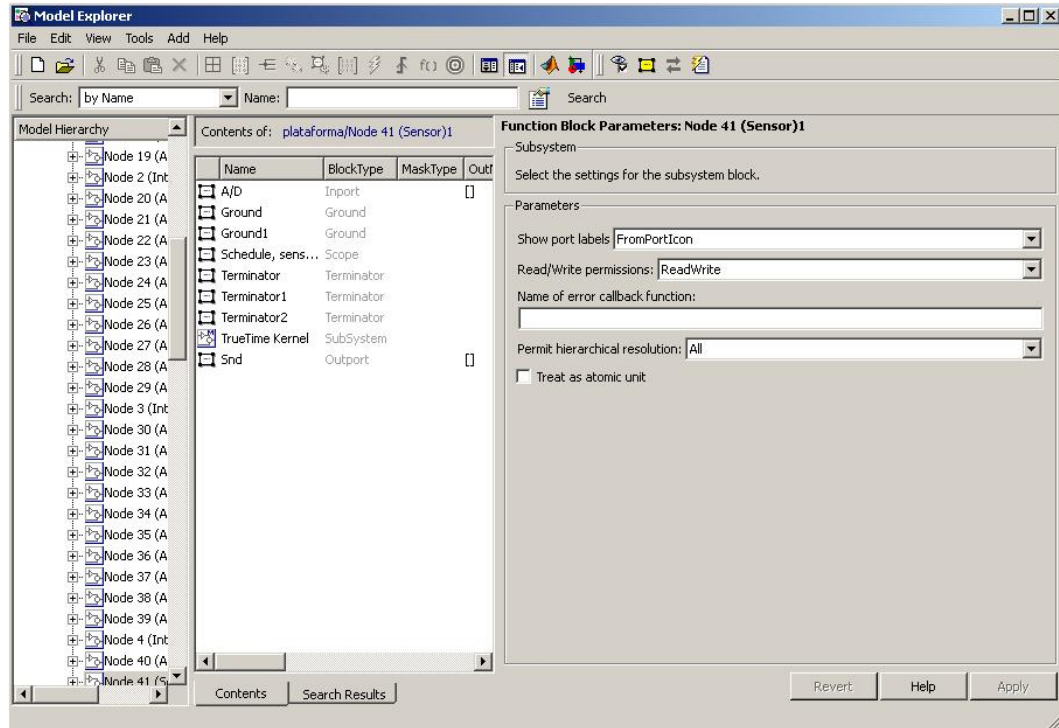


Figura 5.11: Elementos del TrueTime Kernel que representa al sensor 1

5.3 Modificación de la Plataforma

La arquitectura inicial de la plataforma permite la rápida configuración de la mayoría de las aplicaciones que un usuario podría requerir simular. El número de nodos que el usuario puede utilizar está en principio limitado al que permite la pantalla principal (Figura 5.1). En el caso que el usuario requiera agregar algún otro nodo a la red, debe configurar en los cuadros de diálogo correspondientes al TrueTime Network y a las barras *snd* y *rcv* del bloque de red la cantidad deseada de nodos. De esta manera, el bloque de red en la plataforma agregará nodos, los cuales no tendrán nada conectado en sus puertos *snd* y *rcv*. Seguidamente el usuario debe agregar tantos bloques TrueTime Kernel como nodos haya añadido a la red (el usuario determinará si cada bloque agregado representa un sensor, un controlador, un actuador o una interferencia) y, por supuesto, fijar las entradas y salidas necesarias como se explicó anteriormente.

A cada bloque cableado (sensor_i , actuador_i , interferencia_i y controlador_i) se le puede “asociar” su respectivo archivo de Guide (“ $\text{sens}_i.\text{fig}$ ” y “ $\text{sens}_i.\text{m}$ ”, “ $\text{act}_i.\text{fig}$ ”

y “act_i.m”, “interf_i.fig” e “interf_i.m”, “pid_i.fig” y “pid_i.m”). Simplemente se deben agregar esos archivos al directorio donde se encuentra la plataforma (copiar y pegar de uno existente).

En el archivo “sens_i.m” hay que verificar:

- el nombre de la función en las siguientes líneas: `function varargout = sensi(varargin); function sensi_OpeningFcn(hObject, eventdata, handles, varargin); function varargout = sensi_OutputFcn(hObject, eventdata, handles); close sensi.`
- el nombre de las variables tiempo de buffer a placa *tbp_{si}*, tiempo de placa a buffer *tpb_{si}*, tiempo de respuesta *tr_{si}*.

En el archivo “act_i.m” hay que verificar:

- el nombre de la función en las siguientes líneas: `function varargout = acti(varargin); function acti_OpeningFcn(hObject, eventdata, handles, varargin); function varargout = acti_OutputFcn(hObject, eventdata, handles); close acti.`
- el nombre de las variables tiempo de buffer a placa *tbp_{ai}*, tiempo de placa a buffer *tpb_{ai}*, tiempo de respuesta *tr_{ai}*.

En el archivo “interf_j.m” hay que verificar:

- el nombre de la función en las siguientes líneas: `function varargout = interfj(varargin); function interfj_OpeningFcn(hObject, eventdata, handles, varargin); function varargout = interfj_OutputFcn(hObject, eventdata, handles); close interfj.`
- el nombre de las variables tiempo de buffer a placa *tbp_{ij}*, tiempo de placa a buffer *tpb_{ij}*, tiempo de respuesta *tr_{ij}*.

En el archivo “pid_i.m” hay que verificar:

- el nombre de la función en las siguientes líneas: `function varargout = pidi(varargin); function pidi_OpeningFcn(hObject, eventdata, handles, varargin); function varargout = pidi_OutputFcn(hObject, eventdata, handles); close pidi.`

- el nombre de las variables tiempo de buffer a placa tbp_{ci} , tiempo de placa a buffer tpb_{ci} , tiempo de respuesta tr_{ci} .

Como ya se mencionó, a cada bloque sensor, actuador, controlador e interferencia le corresponden archivos de Matlab[®], ubicados en el directorio donde está “plataforma.mdl”. En esos archivos se deben atender los siguientes puntos:

Para un bloque sensor

En sensor_i_init.m:

- En la primera línea de código

```
function sensor1_init
```

debe aparecer el nombre de la función de inicialización ($sensor_i_init$) que es a su vez el nombre del archivo .m

- En la línea de código

```
ttlNitKernel(1, 0, 'prioFP');
```

se indica como primer argumento el número de entradas A/D , como segundo argumento el número de salidas D/A y como tercer argumento la política de diagramación usada ($prioFP$, $prioRM$, $prioEDF$, $prioDM$).

- En la línea de código

```
ttCreatePeriodicTask('sens1_task', offset, period, prio, 'senscode1', data);
```

se debe tener como primer argumento el nombre de la tarea, $sens_i_task$ y como quinto argumento el nombre del archivo .m que tiene el código del sensor, es decir: “senscode_i.m”.

- Para inicializar la red, hay que prestar atención que en la línea de código

```
ttCreateInterruptHandler('nw_handler1', prio, 'msgRcvSensor1');
```

aparezca como primer argumento el nombre del manejador de interrupciones de la tarea correspondiente a ese sensor y como tercer argumento el nombre de la función .m que tiene el código de dicho manejador. Además en la línea de código

```
ttlNitNetwork(1, 41, 'nw_handler1'); % node #41 in the network
```

se debe tener como primer argumento el número de la red, como segundo argumento el número del nodo de red que le corresponde al sensor y como tercer argumento el nombre del manejador de interrupciones de ese sensor.

En `senscodei.m`:

- En la primera línea de código

```
function [exectime, data] = senscode1(seg, data)
```

debe aparecer el nombre de la función, que en este caso es *senscode1* (el mismo nombre lleva el archivo .m que contiene el código de tal sensor)

- En el primer argumento de *ttSendMsg* se indica el nodo al cual el sensor envía la información, por ejemplo con

```
ttSendMsg([1 71], data.y, 80);
```

se envía un mensaje (80 bits) al nodo 71 (controlador 1) de la red 1.

En `msgRcvSensori.m`:

- En la primera línea de código

```
function [exectime, data] = msgRcvSensor1(seg, data)
```

debe aparecer el nombre de la función *msgRcvSensor_i* que es a su vez el nombre del archivo .m que contiene el código del manejador de interrupciones.

Para un bloque interferencia

En `interference_initi.m`:

- En la primera línea de código

```
function interference1_init
```

debe figurar el nombre de la función de inicialización de esa interferencia que es el mismo nombre que tiene el archivo .m que contiene el código (*interference_i_init*).

- En la línea de código

```
ttlNitKernel(0, 0, 'prioFP');
```

se indica como primer argumento el número de entradas *A/D*, como segundo argumento el número de salidas *D/A* y como tercer argumento la política de diagramación utilizada (*prioFP*, *prioRM*, *prioEDF*, *prioDM*).

- En la línea de comando

```
ttCreatePeriodicTask('interf1_task', offset, period, prio, 'interfcode1');
```

debe figurar como primer argumento el nombre de la tarea de esa interferencia y como quinto argumento el nombre del código de dicha interferencia.

- Para inicializar la red, hay que prestar atención que en la línea de código

```
ttCreateInterruptHandler('nw_handler1', prio, 'msgRcvInterf1');
```

figure como primer argumento el nombre del manejador de interrupciones de la tarea correspondiente a esa interferencia y como tercer argumento el nombre de la función .m que tiene el código de dicho manejador. Además en la línea de código

```
ttInitNetwork(1,1, 'nw_handler1'); % node #1 in the network 1
```

se debe tener como primer argumento el número de red, como segundo el nodo de red que le corresponde a la interferencia y como tercer argumento el nombre del manejador de interrupciones de dicha interferencia.

En interfcodei.m:

- En la primera línea de código

```
function [exectime, data] = interfcode1(seg, data)
```

debe figurar el nombre de la función *interfcode_i* que tiene el código de la interferencia (nombre del archivo .m)

- En la línea de código

```
ttSendMsg([1 1], 1, 80); % send 80 bits to myself
```

se indica como primer argumento: el número de red y seguidamente qué nodo de la red ocupa la interferencia, como segundo argumento el dato a enviar y como tercero la longitud de ese dato.

En msgRcvInterf_i.m

- En la primera línea de código

```
function [exectime, data] = msgRcvInterf1(seg, data)
```

debe figurar el nombre de la función que es a su vez el nombre del archivo .m que contiene el código del manejador de interrupciones.

Para un bloque actuador

En actuator_initi.m:

- En la primera línea de código

```
function actuator1_init
```

debe figurar el nombre de la función de inicialización de ese actuador que es el mismo nombre que tiene el archivo .m que contiene el código.

- En la línea de código

```
ttlnitKernel(0, 1, 'prioFP');
```

se indica como primer argumento el número de entradas A/D, como segundo argumento el número de salidas D/A y como tercer argumento la política de diagramación usada (*prioFP*, *prioRM*, *prioEDF*, *prioDM*).

- En la línea de comando

```
ttCreateTask('act1_task', deadline, prio, 'actcode1');
```

debe figurar como primer argumento el nombre de la tarea de ese actuador y como cuarto argumento el nombre del código de dicho actuador.

- Para inicializar la red, hay que prestar atención que en la línea de código

```
ttCreateInterruptHandler('nw_handler1', prio, 'msgRcvActuator1');
```

figure como primer argumento el nombre del manejador de interrupciones de la tarea correspondiente a ese actuador y como tercer argumento el nombre de la función .m que tiene el código de dicho manejador. Y en la línea de código

```
ttlnitNetwork(1, 11, 'nw_handler1'); % node #11 in the network 1
```

se debe tener como primer argumento el número de red, como segundo el número del nodo de red que le corresponde al actuador y como tercer argumento el nombre del manejador de interrupciones de dicho actuador.

En actcode_i.m:

- En la primera línea de código

```
function [exectime, data] = actcode1(seg, data)
```

debe figurar el nombre de la función que es a su vez el nombre del archivo .m del actuador.

En msgRcvActuator_i.m:

- En la primera línea de código

```
function [exectime, data] = msgRcvActuator1(seg, data)
```

debe figurar el nombre de la función que es a su vez el nombre del archivo .m que contiene el código del manejador de interrupciones.

- En la línea de código

```
ttCreateJob('act1_task')
```

se crea el *job* que ejecutará a ese actuador.

Para un bloque controlador (en red)

En controller_init.m:

- En la primera línea de código

```
function controller1_init(arg)
```

debe figurar el nombre de la función de inicialización de ese controlador que es el mismo nombre que tiene el archivo .m que contiene el código correspondiente.

- En la línea de código

```
ttlNitKernel(1, 0, 'prioFP');
```

se indica como primer argumento el número de entradas A/D, como segundo argumento el número de salidas D/A y como tercer argumento la política de diagramación usada (*prioFP*, *prioRM*, *prioEDF*, *prioDM*).

- En la línea de comando

```
ttCreateTask('pid1_task', deadline, prio, 'ctrlcode1', data);
```

debe figurar como primer argumento el nombre de la tarea de ese controlador y como cuarto argumento el nombre del código de dicho controlador.

- En la línea

```
ttCreatePeriodicTask('dummy1', offset, period, prio, 'dummycode1');
```

debe figurar como primer argumento el nombre de la tarea, como segundo el *offset* de inicio, tercero el periodo de invocación, cuarto la prioridad y como quinto argumento el nombre del código de dicha tarea *dummy*.

- Para inicializar la red, hay que prestar atención que en la línea de código

```
ttCreateInterruptHandler('nw_handler1', prio, 'msgRcvCtrl1');
```

figure como primer argumento el nombre del manejador de interrupciones de la tarea correspondiente a ese controlador y como tercer argumento el nombre de la función .m que tiene el código de dicho manejador. Y en la línea de código

```
ttlNitNetwork(1, 71, 'nw_handler1'); % node #71 in the network
```

se debe tener como primer argumento el número de red, como segundo argumento el número de nodo de red que le corresponde al controlador y como tercer argumento el nombre del manejador de interrupciones de dicho controlador.

En ctrlcode;.m:

- En la primera línea de código

```
function [exectime, data] = ctrlcode1(seg, data)
```

debe aparecer el nombre de la función, que en este caso es ctrlcode1 (el mismo nombre lleva el archivo .m que contiene el código de ese controlador)

- En la línea

```
y = ttGetMsg(1); % Obtain sensor value en red 1
```

lee el mensaje que le han enviado por la red. El número entre paréntesis indica en qué red está ubicado ese controlador, que en este caso es la red 1.

- En la línea

```
r = ttAnalogIn(1); % Read reference value
```

lee la entrada del puerto *A/D* (que es el único puerto de entrada que se tiene en este caso)

- En la línea

```
ttSendMsg([1 11], data.u, 80); % Send 80 bits to node 11 (actuador)1 de la red 1
```

figura como primer argumento el número de red seguido del número de nodo en la red que corresponde al actuador, como segundo argumento el dato y como tercer argumento la longitud de dicho dato.

En msgRcvCtrl;.m:

- En la primera línea de código

```
function [exectime, data] = msgRcvCtrl1(seg, data)
```

debe figurar el nombre de la función que es a su vez el nombre del archivo .m que contiene el código del manejador de interrupciones.

- En la línea de código

```
ttCreateJob('pid1_task')
```

se crea el *job* que ejecutará ese controlador.

En dummycode;.m:

- En la primera línea de código

```
function [exectime, data] = dummycode1(seg, data)
```

debe figurar el nombre de la función que es a su vez el nombre del archivo .m de esa tarea.

Para un bloque controlador (en el PLC, comunicado vía OPC)

En OPC_controller_init.m:

- En la primera línea de código

```
function OPC_controller_init(arg)
```

debe figurar el nombre de la función de inicialización de ese controlador que es el mismo nombre que tiene el archivo .m que contiene el código correspondiente.

- En la línea de código

```
ttInitKernel(2, 1, 'prioFP');
```

se indica como primer argumento el número de entradas A/D, como segundo argumento el número de salidas D/A y como tercer argumento la política de diagramación usada (*prioFP*, *prioRM*, *prioEDF*, *prioDM*).

- En la línea

```
ttCreatePeriodicTask('pid_task', deadline, periods, prio, 'OPC_ctrlcode', data);
```

debe figurar como primer argumento el nombre de la tarea y como quinto argumento el nombre del código que ejecuta dicha tarea.

Como se observa el controlador del PLC no está conectado a la red (no hay función de inicialización de red: *ttCreateInterruptHandler* y *ttInitNetwork*), se comunica con todo el sistema a través de OPC.

En OPC_ctrlcode.m:

- En la primera línea de código

```
function [exectime, data] = OPC_ctrlcode(seg, data)
```

debe aparecer el nombre de la función, que en este caso es *OPC_ctrlcode* (el mismo nombre lleva el archivo .m que contiene el código de ese controlador)

- En las líneas

```
r = ttAnalogIn(data.rChan); % Read reference value
```

```
y = ttAnalogIn(data.yChan); % Read process output
```

lee las entradas de los puertos A/D 1 y 2 respectivamente.

- En la línea

```
ttAnalogOut(data.uChan, data.u); % Output control signal
```

figura como primer argumento el número del puerto D/A y como segundo argumento el valor del dato a enviar.

En OPC_read_init.m:

- En la primera línea de código

```
function OPC_read_init
```

debe aparecer el nombre de la función de inicialización (*OPC_read_init*) que es a su vez el nombre del archivo .m

- En la línea de código

```
ttlNitKernel(1, 0, 'prioFP');
```

se indica como primer argumento el número de entradas A/D, como segundo argumento el número de salidas D/A y como tercer argumento la política de diagramación usada (*prioFP*, *prioRM*, *prioEDF*, *prioDM*).

- En la línea de código

```
ttCreatePeriodicTask('read_task', deadline, period, prio, 'OPC_readcode', data);
```

se debe tener como primer argumento el nombre de la tarea y como quinto argumento el nombre del archivo .m que tiene el código para ejecutar dicha tarea.

- Para inicializar la red, hay que prestar atención que en la línea de código

```
ttCreateInterruptHandler('nw_handler', prio, 'OPC_msgRcvRead');
```

aparezca como primer argumento el nombre del manejador de interrupciones de la tarea correspondiente a ese bloque y como tercer argumento el nombre de la función .m que tiene el código de dicho manejador. Además en la línea de código

```
ttlNitNetwork(1, 101, 'nw_handler'); % node #101 in the network 1
```

se debe tener como primer argumento el número de la red, como segundo argumento el número del nodo de red que le corresponde al bloque y como tercer argumento el nombre del manejador de interrupciones de ese bloque read.

En *OPC_readcode.m*:

- En la primera línea de código

```
function [exectime, data] = OPC_readcode(seg, data)
```

debe aparecer el nombre de la función, que en este caso es *OPC_readcode* (el mismo nombre lleva el archivo .m que contiene el código del bloque que se encarga de enviar los datos calculados por el controlador del PLC desde el servidor OPC a la red)

- La línea

```
data.u=ttAnalogIn(1); % lee del servidor OPC el dato del controlador
```

indica que se está leyendo del puerto 1 del bloque read (en este caso es el dato del controlador que está guardado en el servidor OPC).

- En el primer argumento de *ttSendMsg* se indica el nodo al cual se envía la información, por ejemplo con

```
ttSendMsg([1 11], data.u, 80);
```

se envía un mensaje (80 bits) al nodo 11 (actuador 1) de la red 1.

En OPC_read2_init.m:

- En la primera línea de código

```
function OPC_read2_init
```

debe aparecer el nombre de la función de inicialización (*OPC_read2_init*) que es a su vez el nombre del archivo *.m*

- En la línea de código

```
ttlNitKernel(0, 1, 'prioFP');
```

se indica como primer argumento el número de entradas A/D, como segundo argumento el número de salidas D/A y como tercer argumento la política de diagramación usada (*prioFP*, *prioRM*, *prioEDF*, *prioDM*).

- En la línea de código

```
ttCreateTask('read2_task', deadline, prio, 'OPC_readcode2', data);
```

se debe tener como primer argumento el nombre de la tarea y como cuarto argumento el nombre del archivo *.m* que tiene el código para ejecutar dicha tarea.

- Para inicializar la red, hay que prestar atención que en la línea de código

```
ttCreateInterruptHandler('nw_handler', prio, 'OPC_msgRcvRead2');
```

aparezca como primer argumento el nombre del manejador de interrupciones de la tarea correspondiente a ese bloque y como tercer argumento el nombre de la función *.m* que tiene el código de dicho manejador. Además en la línea de código

```
ttlNitNetwork(1, 102, 'nw_handler'); % node #102 in the network 1
```

se debe tener como primer argumento el número de la red, como segundo argumento el número del nodo de red que le corresponde al bloque y como tercer argumento el nombre del manejador de interrupciones de ese bloque *read2*.

En OPC_readcode2.m:

- En la primera línea de código

```
function [exectime, data] = OPC_readcode2(seg, data)
```

debe aparecer el nombre de la función, que en este caso es OPC_readcode2 (el mismo nombre lleva el archivo .m que contiene el código del bloque que se encarga de recibir los datos que el sensor envía desde la red al servidor OPC)

- En la línea

```
data.u = ttGetMsg(1); % el mensaje se recibe en la red 1
```

se lee desde la red el mensaje que el sensor ha enviado al bloque read para que a través de éste llegue al controlador que está en el PLC.

- En la línea de código

```
ttAnalogOut(1, data.u);
```

se indica que por el puerto 1 de este bloque read2 se envía el dato (en este caso es el dato del sensor que se escribe en el write del servidor OPC para que sea almacenado por el mismo).

Cuando se desee agregar o modificar entradas *A/D* o salidas *D/A* de un bloque Truetime Kernel que representa un controlador, se deberá verificar en la función de iniciación de dicho bloque (“controller1_init.m” por ejemplo para el controlador₁) que en la línea

```
ttlInitKernel(1, 0, 'prioFP'); % nbrOfInputs, nbrOfOutputs, fixed priority
```

aparezcan la cantidad correcta de entradas y salidas. Además sobre los bloques *Mux* y *Demux* se ingresará la cantidad deseada de entradas o salidas, respectivamente (Figura 5.13). En la Figura 5.12 se observa una vista interna del bloque controlador₁ (al *Mux* llega una entrada y en el *Demux* no hay salida, sino un bloque *Terminator* en concordancia con los parámetros de la función *ttlInitKernel*).

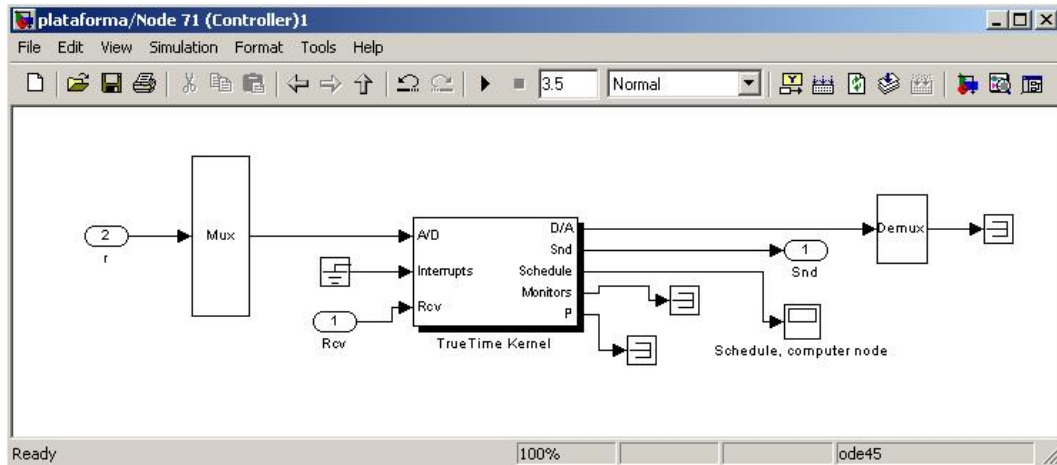


Figura 5.12: Vista interna del bloque controlador₁

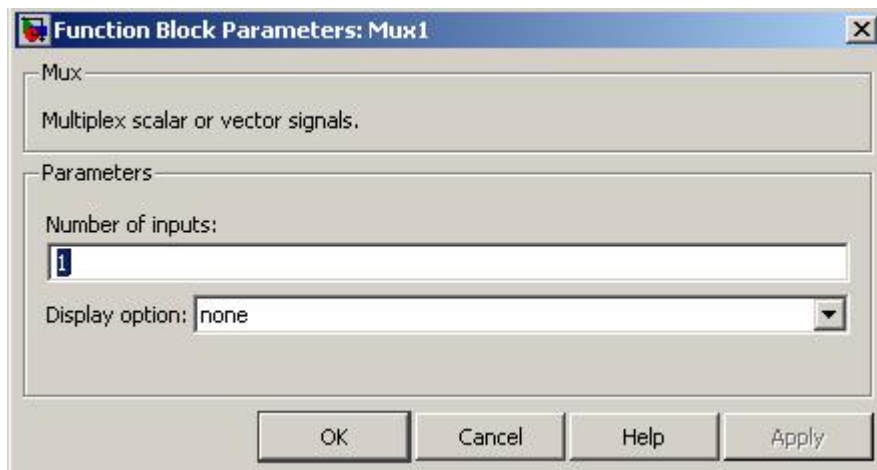


Figura 5.13: Bloque Mux

La anterior es, entonces, la manera en que el usuario puede modificar el número de entradas *A/D* o salidas *D/A* de un bloque TrueTime Kernel.

5.3.1 Parámetros que definen el desempeño de redes de campo

Existen parámetros que definen el desempeño de las redes de campo en el ambiente de simulación de la plataforma. Para cada nodo puede definirse:

- Tasa de arribo de mensajes al buffer de transmisión de cada nodo (*Periodo*): cada sensor, interferencia, etc. implementado a través de tareas periódicas, tiene su propia tasa de arribo de los mensajes al buffer del microprocesador. Se configura dando el valor deseado al parámetro *period* que aparece en su función de inicialización.

- Proceso de arribo de mensajes al buffer: determina la manera de arribo de mensajes al buffer y puede establecerse en: aleatorio, determinístico o cuasi aleatorio (es una mezcla de los dos anteriores). A aquellos elementos que tienen un proceso de arribo de mensajes al buffer determinístico (*prioFP*), se les puede establecer tal parámetro dando el valor deseado a *prio* en su función de inicialización.
- Tiempo de buffer a la placa de red: es el tiempo que un mensaje es retrasado por la interfase de red en el extremo transmisor. Esto se puede utilizar para modelar, por ejemplo, una conexión serial lenta entre la computadora y la interfase de red. Se configura mediante la función *ttSetNetworkParameter* de TrueTime, dándole el valor deseado al parámetro *predelay*.
- Tiempo de servicio: es el tiempo que tarda un transmisor en enviar un mensaje. Para que un transmisor ubicado en un nodo pueda transmitir, tiene que asegurarse que la red esté libre. Eso no depende sólo de ese nodo (por ejemplo si se tuviese Ethernet, el que se encarga de eso es el protocolo CSMA/CD) sino también de si otros nodos están transmitiendo.
- Tiempo de placa de red a buffer: es el tiempo que un mensaje es retrasado por la interfaz de la red en el extremo de recepción. Se configura mediante la función *ttSetNetworkParameter* de TrueTime, dándole el valor deseado al parámetro *postdelay*.
- Tiempo de respuesta: es el tiempo en que el mensaje está disponible a quien le fuera enviado (por ejemplo es el tiempo en que el mensaje está en red -sale de un sensor- y llega a estar disponible en el controlador). Se configura por medio de *set_param* dando el valor adecuado a la variable *rate*.

Algunos de estos parámetros se configuran antes de comenzar la simulación (como Tasa de arribo de mensajes al buffer de transmisión de cada nodo y Proceso de arribo de mensajes al buffer determinístico (*prioFP*)) y otros (Tiempo

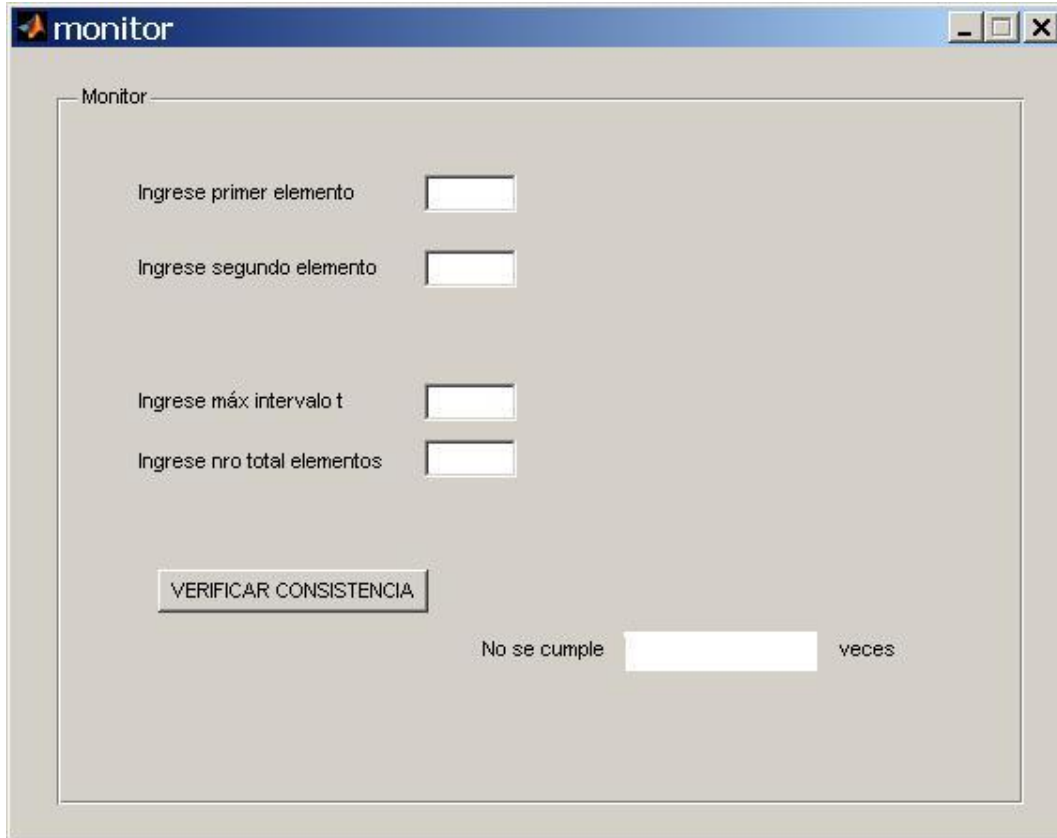
de buffer a la placa de red, Tiempo de placa de red a buffer, Tiempo de respuesta) durante la simulación (Figura 5.3).

5.3.2 Monitoreo de consistencia

Se pretende monitorear la consistencia de datos con respecto al tiempo en que un dato sale de un nodo y el instante en que llega al nodo receptor. El usuario de la plataforma puede verificar, al final de la simulación, un archivo de reporte donde aparecen los instantes en que los datos son transmitidos por cada nodo. A su vez, el monitor le indica al usuario si existió consistencia, es decir si la diferencia entre el instante en que el mensaje llega al nodo receptor (Ej. nodo controlador) y el instante en que sale del nodo emisor (Ej. nodo sensor) fue menor a un determinado intervalo máximo de tiempo.

Para poder utilizar el monitor de consistencia, una vez finalizada la simulación, el usuario debe ejecutar el comando *monitor* desde el *prompt* de Matlab[®]. Se abre entonces la pantalla mostrada en la Figura 5.14. Debe ingresar como primer elemento el nodo emisor (Ej. s1) y como segundo elemento el nodo receptor (Ej. c1). En la casilla *máximo intervalo de tiempo*, se ingresa el valor deseado. En la casilla *número total de elementos*, el número de elementos (sumatoria entre cantidad de sensores, actuadores, controladores e interferencias) que se están utilizando.

Con la opción “Verificar consistencia” aparecerá debajo la cantidad de veces en las que no hubo consistencia de datos.



The image shows a software window titled "monitor" with a standard Windows-style title bar (minimize, maximize, close buttons). The main content area is titled "Monitor" and contains the following elements:

- Four input fields, each preceded by a label:
 - "Ingrese primer elemento" followed by an empty text box.
 - "Ingrese segundo elemento" followed by an empty text box.
 - "Ingrese máx intervalo t" followed by an empty text box.
 - "Ingrese nro total elementos" followed by an empty text box.
- A button labeled "VERIFICAR CONSISTENCIA" located below the input fields.
- Below the button, the text "No se cumple" is followed by an empty text box and the word "veces".

Figura 5.14: Monitor de consistencia

5.4 Conclusión

En este capítulo se describieron los componentes y estructuras de la plataforma de simulación. Se mostró la interfase principal de la plataforma que permite una rápida configuración de la simulación sin requerir conocimientos profundos de las librerías que la conforman.

Por otro lado, se detallaron las diferentes funciones que componen la plataforma para permitir al usuario una rápida localización de los parámetros que debe configurar, de modo de adaptar la plataforma a particularidades que pueden presentarse en simulaciones de sistemas complejos.

Capítulo 6

Casos de estudio

6.1 Introducción

En este capítulo se describen diversos casos de estudio que permiten evaluar las diferentes características de la plataforma.

6.1.1 Control de temperatura de bobinado

El siguiente es un ejemplo implementado en la plataforma y que fue desarrollado en [7]. La Figura 6.1 muestra un esquema de control de temperatura de bobinado en un tren de laminación de chapa en caliente. La regulación precisa de esta temperatura es uno de los factores claves que determinan la calidad de la chapa producida.

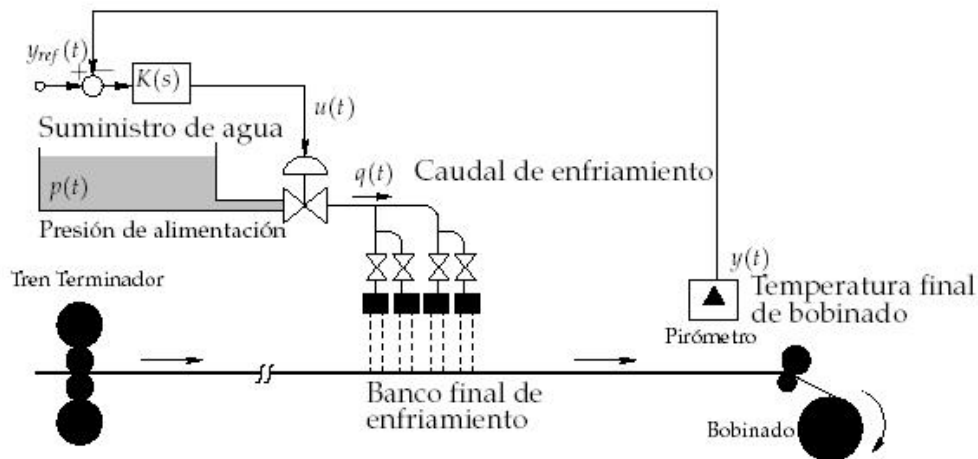


Figura 6.1: Control simple de temperatura de bobinado

En el esquema de la Figura 6.1, la temperatura de bobinado $y(t)$ se controla mediante la regulación del caudal $q(t)$ de agua que alimenta el banco final de enfriamiento. La regulación del caudal $q(t)$ se realiza mediante una válvula alimentada desde un tanque que suministra el agua de enfriamiento al tren de laminación. Una de las dificultades en este sistema de control es que la presión $p(t)$ de alimentación varía. Esta variación afecta el control de temperatura ya que, para una misma señal de control $u(t)$, el caudal de enfriamiento $q(t)$ varía si varía $p(t)$.

Si se puede medir el caudal $q(t)$, una solución para atenuar el efecto de las variaciones de $p(t)$ sobre $y(t)$ es agregar un lazo adicional para regular $q(t)$ al valor de referencia $q_{ref}(t)$, generado por el primer controlador $K_1(s)$ (Figura 6.2), estableciendo de este modo un esquema de control en cascada. La arquitectura de la configuración en cascada se resume en la Figura 6.3.

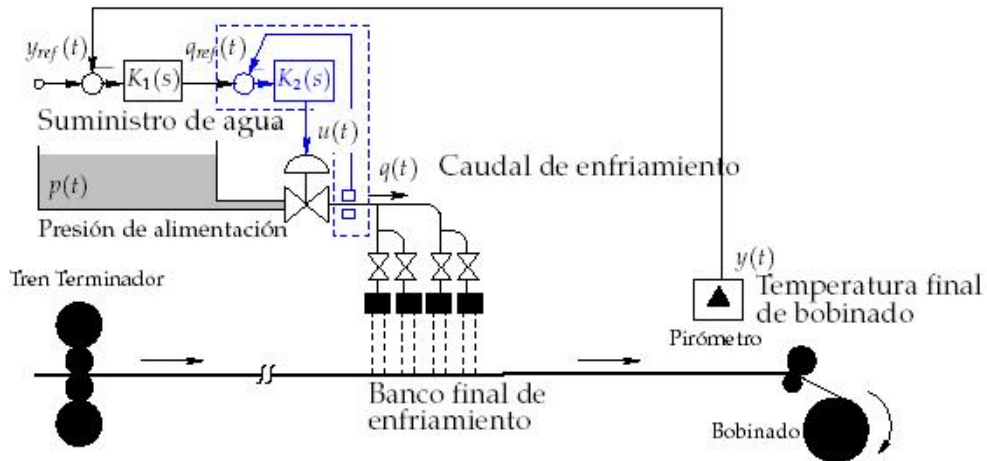


Figura 6.2: Control en cascada de temperatura de bobinado

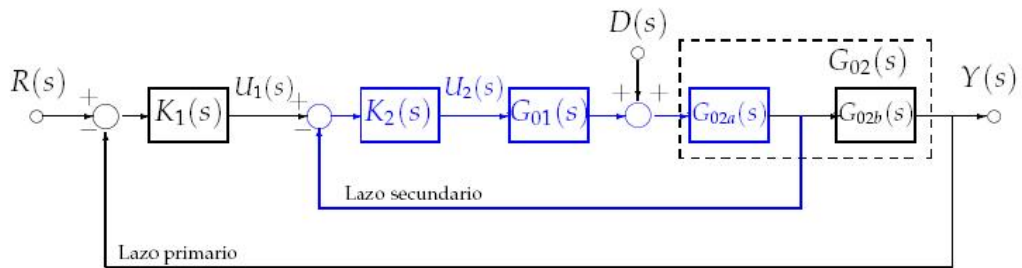


Figura 6.3: Estructura general del control en cascada

En la estructura de la Figura 6.3, la planta se presenta desglosada en dos partes, $G_{01}(s)$ y $G_{02}(s)$, según dónde afecte la perturbación. A su vez, $G_{02}(s)$ puede desglosarse en $G_{02a}(s)$ y $G_{02b}(s)$, dependiendo de cuál sea la señal que puede medirse para construir el lazo secundario.

En [7], la planta considerada es $G_0(s) = G_{01}(s) \cdot G_{02}(s)$, donde

$$G_{01}(s) = \frac{1}{(s+1)} \quad G_{02}(s) = \frac{e^{-s}}{(2s+1)} = G_{02a}(s) \cdot G_{02b}(s) = 1 \cdot \frac{e^{-s}}{(2s+1)} \quad (12)$$

El controlador secundario ve la planta $G_{01}(s) \cdot G_{02a}(s) = 1/(s+1)$. Y su ecuación es:

$$K_2(s) = \frac{8 \cdot (s+1)}{s} \quad (13)$$

Para el controlador primario, los autores de [7] utilizan un predictor de Smith. Este controlador primario ve una planta equivalente:

$$G_{eq}(s) = T_{02}(s) \cdot G_{02b}(s) = \frac{8 \cdot e^{-s}}{(2s+1)(s+8)} \quad (14)$$

Y tiene la ecuación que sigue:

$$K_1(s) = \frac{9 \cdot (s+1/2)}{s} \quad (15)$$

Para implementar el ejemplo en la plataforma, debe revisarse que las conexiones sean las adecuadas para el caso en estudio (Figura 6.4 y 6.5). Luego, al ejecutar la plataforma, se eligen los nodos con los que se trabajará (Figura 6.6).

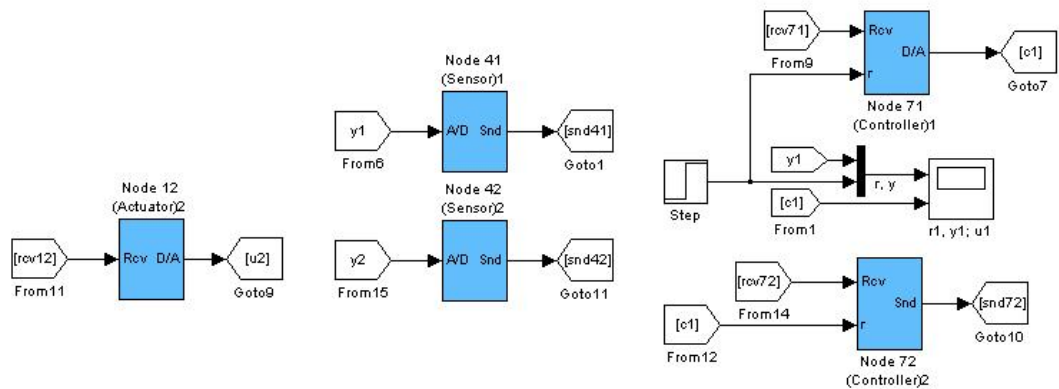


Figura 6.4: Bloques sensor, actuador y controlador utilizados en este ejemplo

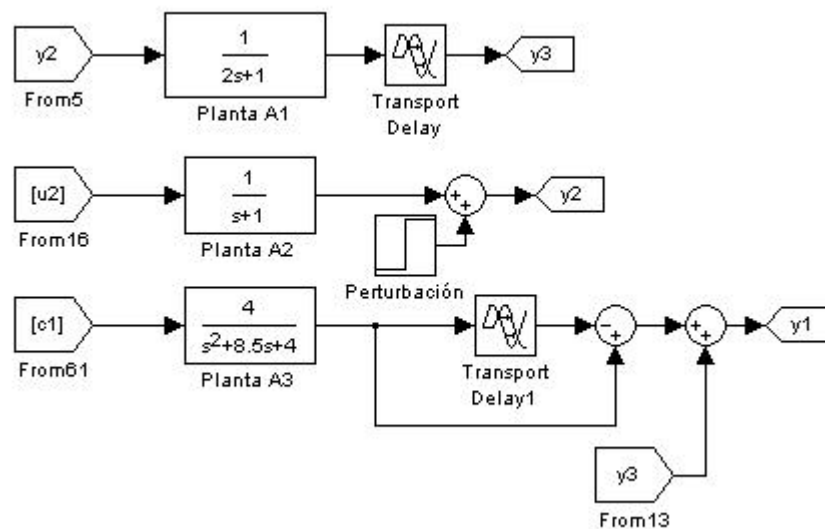


Figura 6.5: Bloques plantas utilizados en este ejemplo

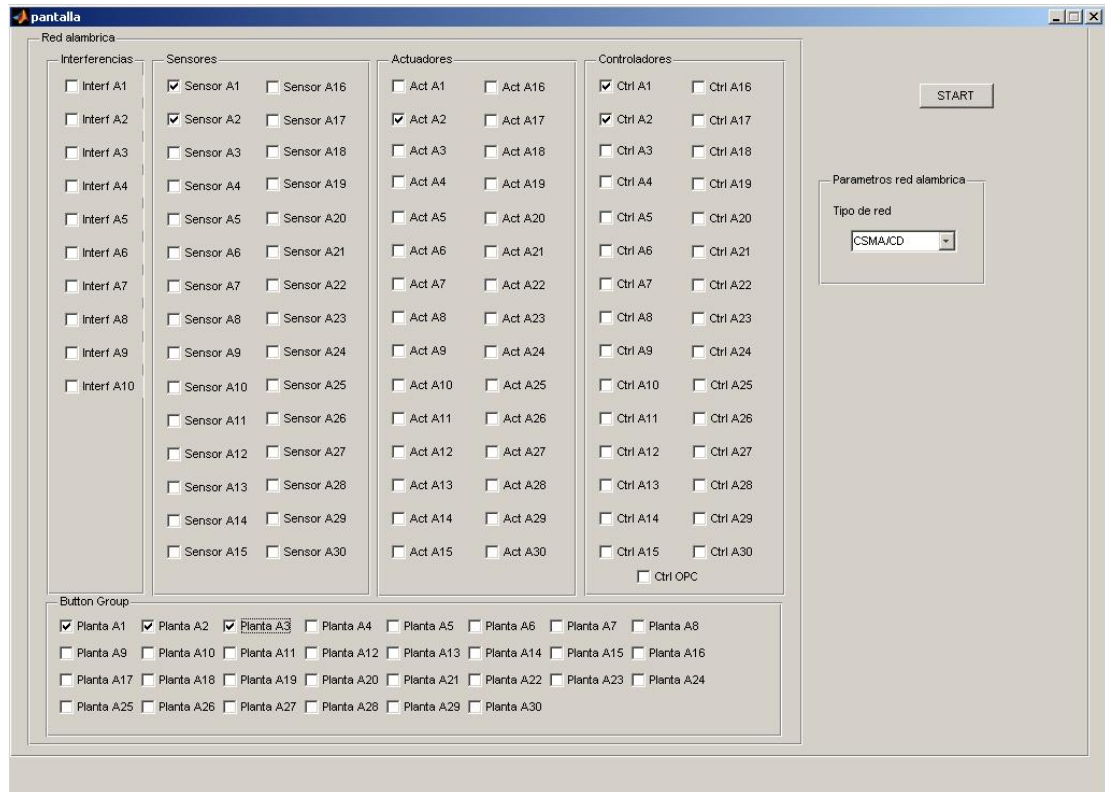


Figura 6.6: Elección de elementos con los que se armará el sistema en estudio

En el apéndice I, se da una explicación detallada de los pasos a seguir para implementar este ejemplo en la plataforma.

En la Figura 6.7 se muestra el desempeño de este esquema de control en cascada, para una entrada de referencia escalón unitario aplicada en $t = 0$ y una perturbación escalón unitario aplicada en $t = 5$. Se ve cómo se atenúa significativamente la perturbación, manteniendo el seguimiento asintótico de la referencia.

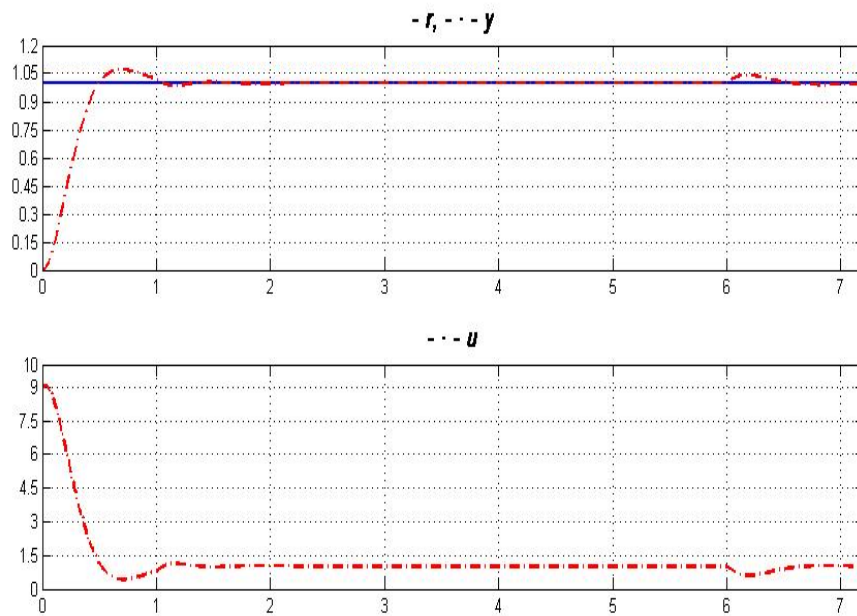


Figura 6.7: Respuesta del sistema a lazo cerrado usando control en cascada

La plataforma permite implementar de manera fácil un esquema de control en cascada de un proceso en tiempo real. Se pueden conectar sensores, actuadores, controladores a la red de campo con sólo elegir los nodos intervinientes en la pantalla principal (Figura 6.6). Esto permite determinar cuáles de ellos formarán parte del lazo de control primario o del lazo de control secundario al verificar las conexiones en el archivo de Simulink[®] donde está implementada la plataforma (“plataforma.mdl”) (Figuras 6.4 y 6.5). En este caso de estudio se muestra un ejemplo obtenido de la referencia consultada y se comprueba que la plataforma lo implementa en forma correcta.

6.1.2 Control distribuido utilizando OPC

El siguiente es un ejemplo implementado en la plataforma, cuyo controlador fue desarrollado por [8]. En este caso, el controlador se encuentra implementado en un PLC real y se comunica con la red (a la cual están conectados sensor, actuador e interferencia) a través del protocolo OPC. Este ejemplo simula el control distribuido (en red) del servo de corriente continua representado por la función de transferencia (16).

$$G(s) = \frac{1000}{s(s+1)} \quad (16)$$

El controlador PID es implementado de acuerdo a las siguientes ecuaciones:

$$P(k) = K \cdot (\beta_r(k) - y(k))$$

$$I(k+1) = I(k) + \frac{K \cdot h}{T_i} (r(k) - y(k))$$

$$D(k) = a_d D(k-1) + b_d (y(k-1) - y(k)) \quad (17)$$

$$u(k) = P(k) + D(k) + I(k)$$

donde $a_d = \frac{T_d}{Nh + T_d}$ y $b_d = \frac{NKT_d}{Nh + T_d}$, ver [9].

Los parámetros del controlador son calculados inicialmente en [8] de tal manera de tener un sistema a lazo cerrado con ancho de banda $\omega_c = 20 \text{ rad/s}$ y muestreo relativo de $\xi = 0.7$. Pero como el controlador está en el PLC, y se comunica con los restantes elementos de la red a través de OPC, se genera un retardo, que debe ser tenido en cuenta. Eso hace que los parámetros del controlador deban ser ajustados para tener en cuenta dicho retardo.

El ejemplo contiene cuatro bloques de computadora, cada uno representado por un bloque TrueTime Kernel. Un nodo sensor, manejado por tiempo, muestrea el proceso periódicamente y envía las muestras sobre la red al servidor OPC. El controlador en el PLC lee ese valor del servidor OPC y luego calcula la señal de control y la escribe en el servidor OPC. El actuador lee, a través de la red, el dato que el controlador escribió en el servidor OPC y lo actúa posteriormente. La simulación también provee de un nodo interferencia que envía tráfico sobre la red perturbando a la misma.

Para implementar el ejemplo en la plataforma, debe revisarse que las conexiones sean las adecuadas para el caso en estudio (Figura 6.8 a 6.10). Luego, al ejecutar la plataforma, se eligen los nodos con los que se trabajará (Figura 6.11).

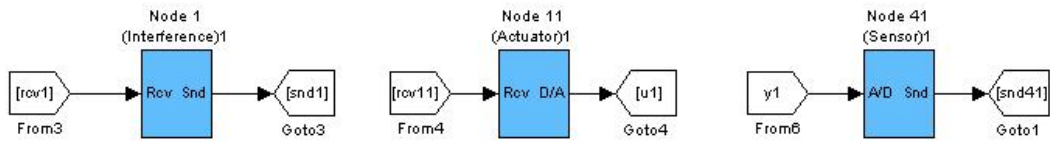


Figura 6.8: Bloques interferencia, sensor y actuador utilizados en este ejemplo

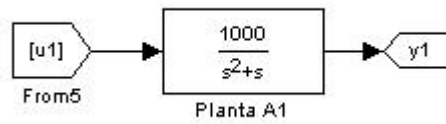


Figura 6.9: Bloque planta utilizado en este ejemplo

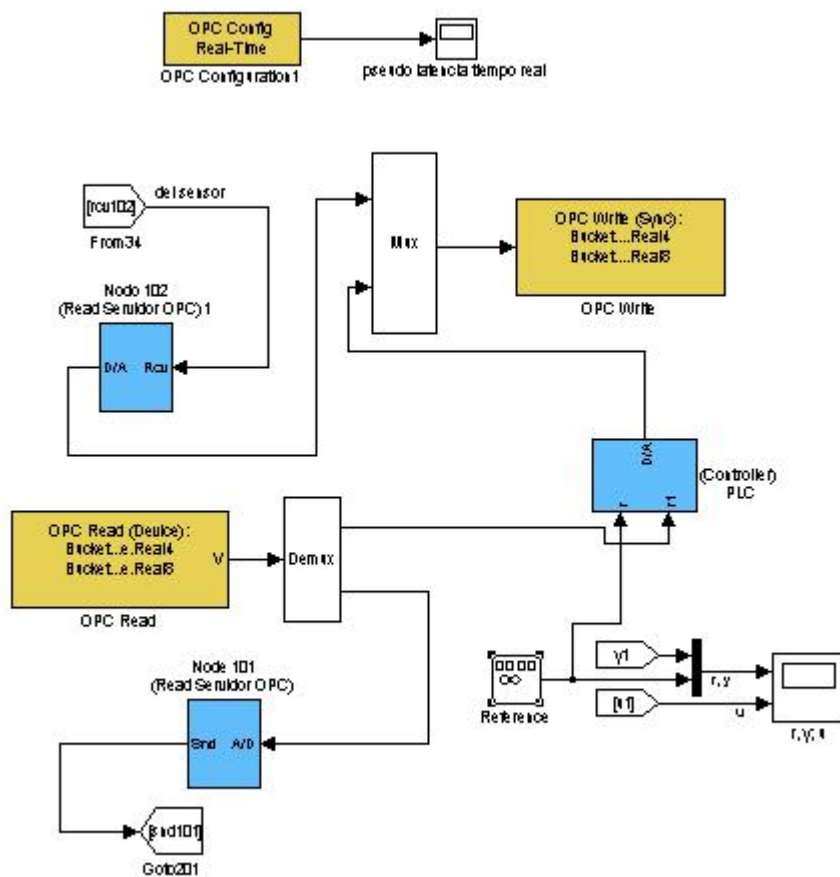


Figura 6.10: Servidor OPC y PLC (con su controlador)

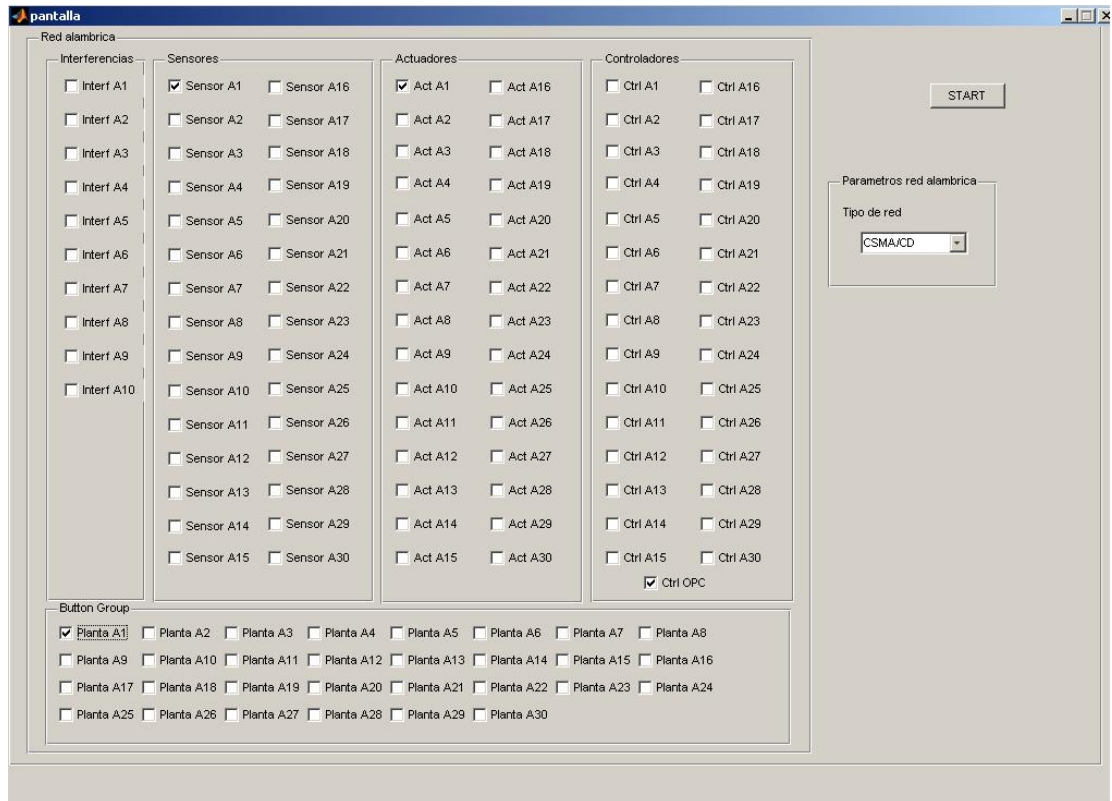


Figura 6.11: Elección de elementos con los que se armará el sistema en estudio

En el apéndice I se da una explicación detallada de los pasos a seguir para implementar este ejemplo en la plataforma.

Se probó el esquema de control para una perturbación unitaria estática, utilizando un controlador PID con valores $K=1.1$, $T_i=1$ y $T_d=0.06$. Se muestra el comportamiento del sistema en la Figura 6.12.

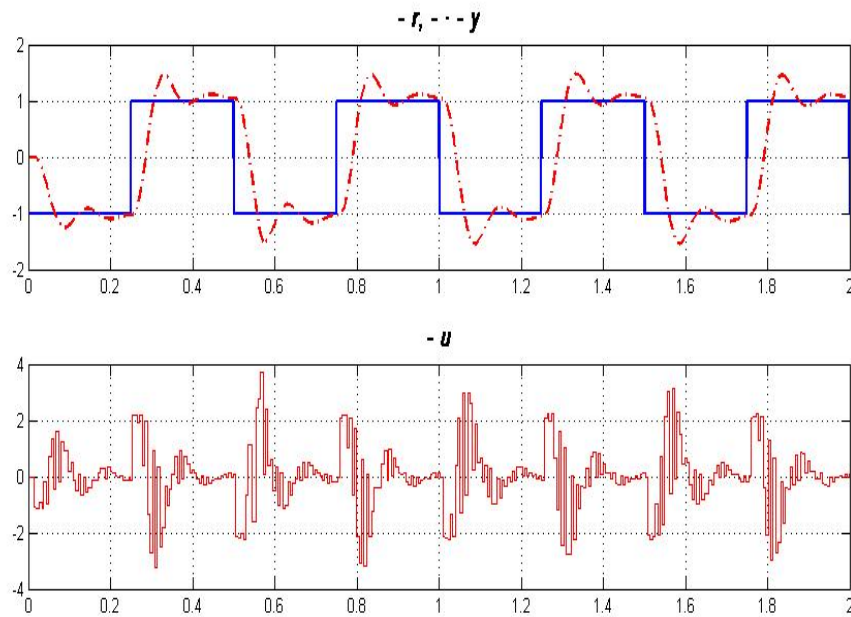


Figura 6.12: Respuesta a lazo cerrado empleando un esquema de control realimentado de tipo PID, sin interferencia en la red

Se utilizó la plataforma para establecer un esquema de control, que se ejecuta en un PLC. El controlador se comunica con el sensor y con el actuador empleando el protocolo de comunicación OPC. Los elementos que forman parte del sistema se eligen a través de la pantalla principal, así como el perfil de red deseado (CSMA/CD, CSMA/AMP, Round Robin, FDMA, TDMA o Switched Ethernet), (Figura 6.11).

6.1.3 Columna de destilación

Las columnas de destilación son equipos muy utilizados en las plantas industriales de procesos químicos. Su propósito es la separación de mezclas de líquidos a partir de las diferentes características de sus componentes.

La columna considerada en este ejemplo es una columna piloto de destilación de mezcla agua-etanol, ilustrada en el esquema de la Figura 6.13. La aplicación como el controlador utilizado fueron descritos en [10].

Con el resto de las variables claves controladas, se dejan dos entradas de control: el caudal de reflujo, $u_1(t)$, para controlar la concentración de etanol en el

destilado, $y_1(t)$; y el caudal de vapor de rehervido, $u_2(t)$, para controlar la composición del producto de fondo, medida por la temperatura del fondo de la columna $y_2(t)$.

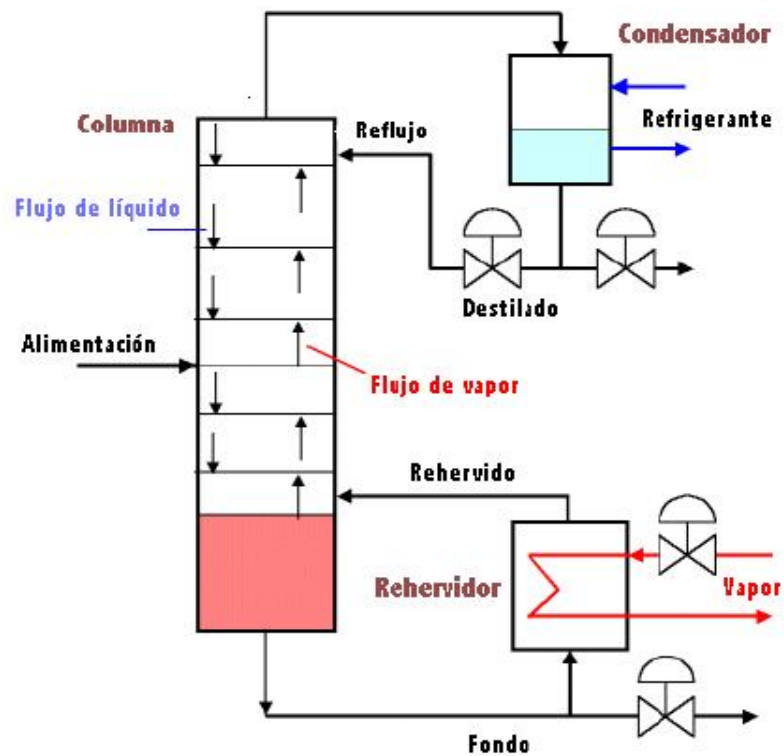


Figura 6.13: Columna de destilación

Un modelo linealizado de la columna está dado por las ecuaciones:

$$\begin{bmatrix} Y_1(s) \\ Y_2(s) \end{bmatrix} = \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} \begin{bmatrix} U_1(s) \\ U_2(s) \end{bmatrix} \quad (18)$$

donde

$$\begin{aligned} G_{11}(s) &= \frac{0.09851 \cdot e^{-0.043s}}{(s + 8.955)} & G_{12}(s) &= \frac{-0.0548 \cdot e^{-0.017s}}{(s + 6.6225)} \\ G_{21}(s) &= \frac{-0.04258 \cdot e^{-0.153s}}{(s + 7.362)} & G_{22}(s) &= \frac{(0.1380 \cdot s + 0.71382) \cdot e^{-0.017s}}{(s^2 + 18.6156 \cdot s + 49.2248)} \end{aligned} \quad (19)$$

Este modelo está escalado con unidades de tiempo *en horas* (es un proceso lento, como muchos procesos químicos).

Se utilizan dos controladores PI, uno conectando y_1 a u_1 , y el otro conectando y_2 a u_2 .

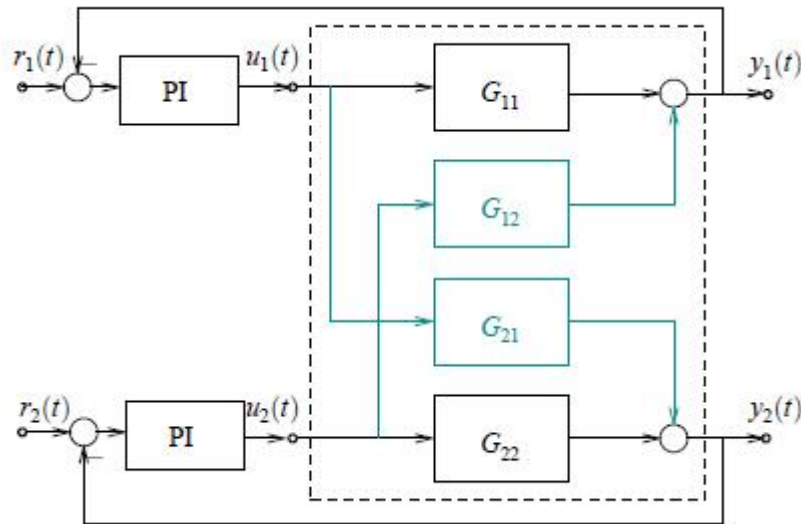


Figura 6.14: Control PI descentralizado de la columna de destilación

Al ignorar las transferencias G_{12} y G_{21} se considera la planta como si fuera dos sistemas SISO¹⁶ separados (no interactuantes). Este enfoque se llama *control descentralizado* ([10]).

Los controladores utilizados en [10] son

$$K_1(s) = 60.09 \cdot \left(1 + \frac{1}{0.1117 \cdot s}\right) \quad K_2(s) = 60.09 \cdot \left(1 + \frac{1}{0.1898 \cdot s}\right) \quad (20)$$

Para implementar el ejemplo en la plataforma, debe revisarse que las conexiones sean las adecuadas para el caso en estudio (Figuras 6.15 y 6.16). Luego, al ejecutar la plataforma, se eligen los nodos con los que se trabajará (Figura 6.17).

¹⁶ SISO: single input single output en inglés

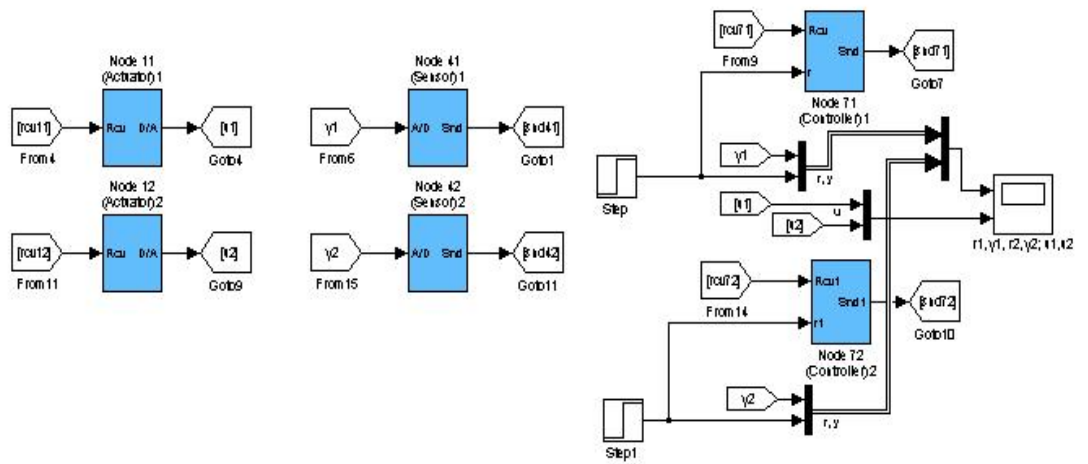


Figura 6.15: Bloques sensor, actuador y controlador utilizados en este ejemplo

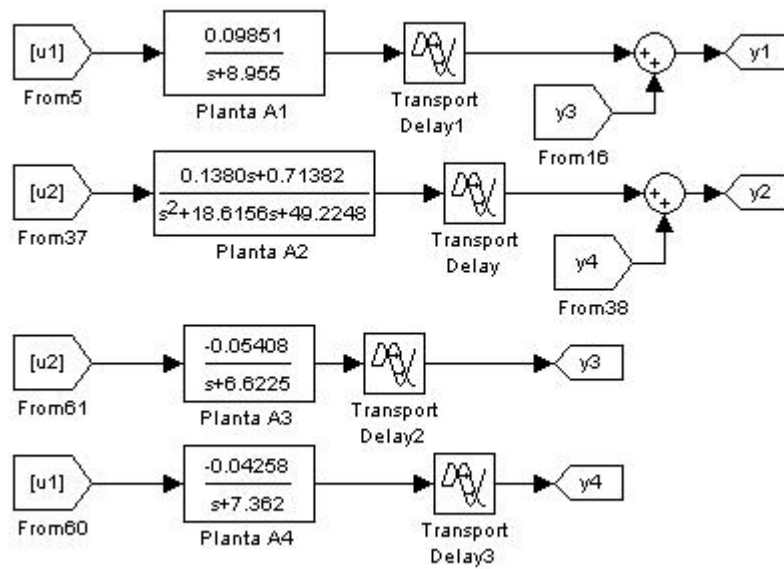


Figura 6.16: Bloques plantas utilizados en este ejemplo

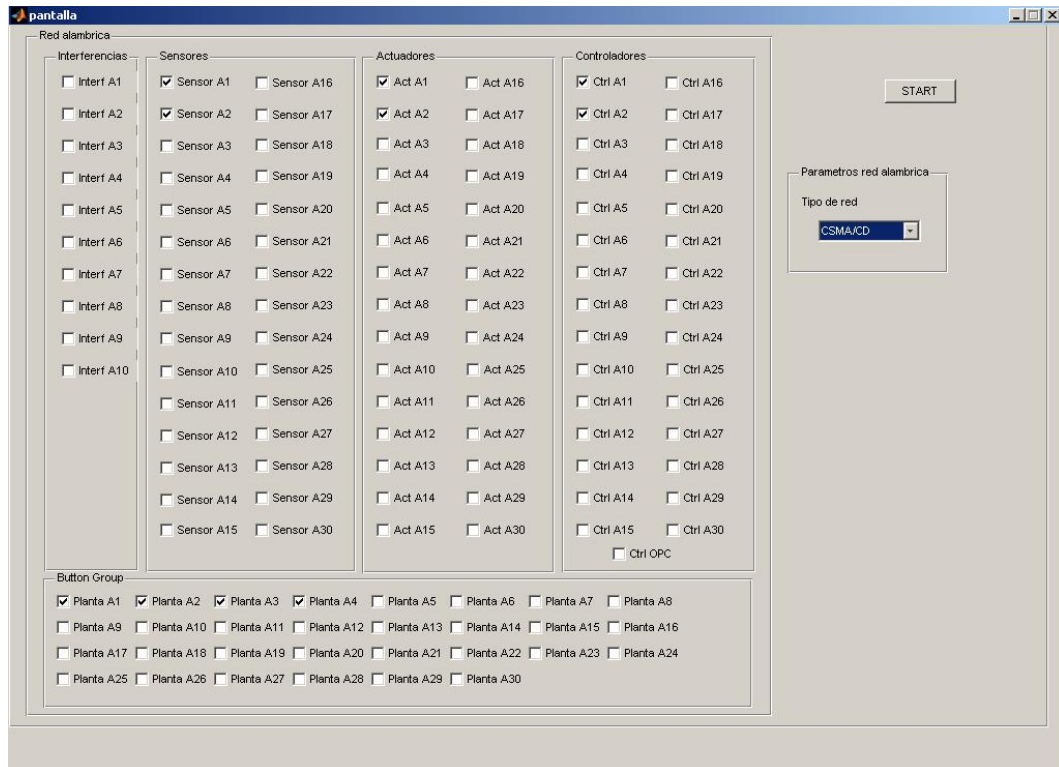


Figura 6.17: Elección de elementos con los que se armará el sistema en estudio

En el apéndice I se da una explicación detallada de los pasos a seguir para implementar este ejemplo en la plataforma.

En la Figura 6.18 se muestra el desempeño de este esquema de control PI, que se obtuvo simulando el sistema a lazo cerrado con el modelo de la planta. Se ve, sin embargo, que los lazos *interactúan* (r_1 afecta a y_2 , y r_2 a y_1).

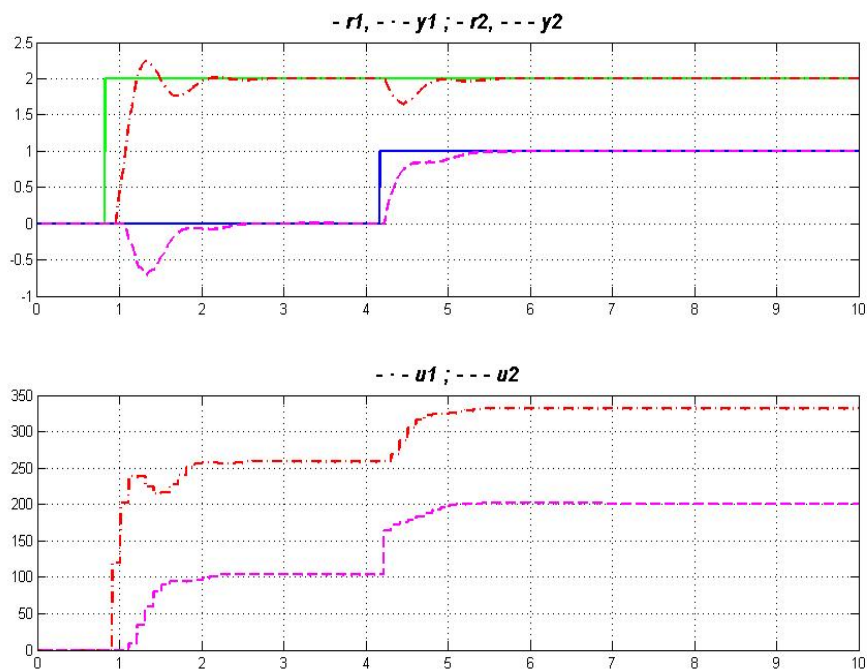


Figura 6.18: Respuesta a lazo cerrado empleando un esquema de control feedback de tipo PI

El ejemplo también puede implementarse sobre un PLC real realizando la comunicación de datos mediante el protocolo OPC. Primero debe revisarse que las conexiones sean las adecuadas para el caso en estudio (Figuras 6.19, 6.20 y 6.21).

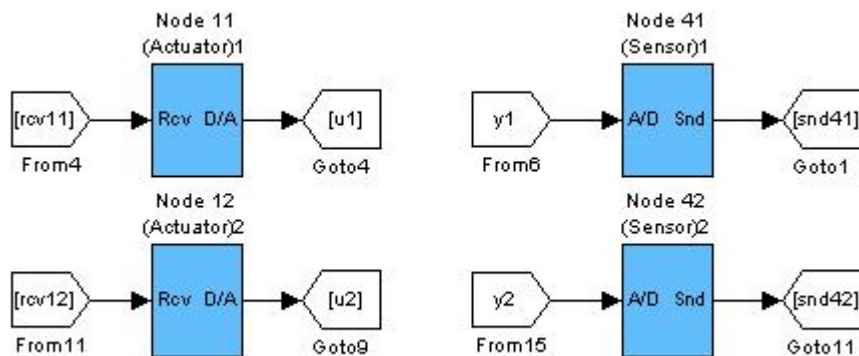


Figura 6.19: Bloques sensor y actuador utilizados en este ejemplo

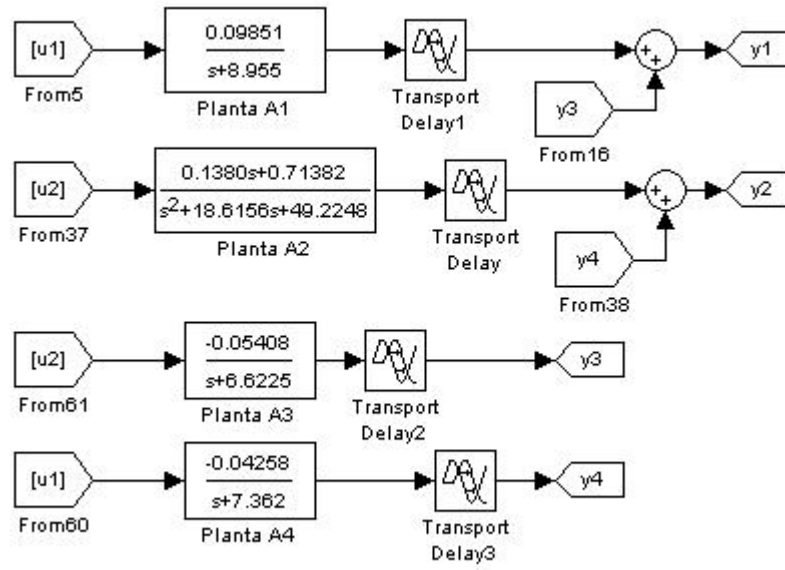


Figura 6.20: Bloques plantas utilizados en este ejemplo

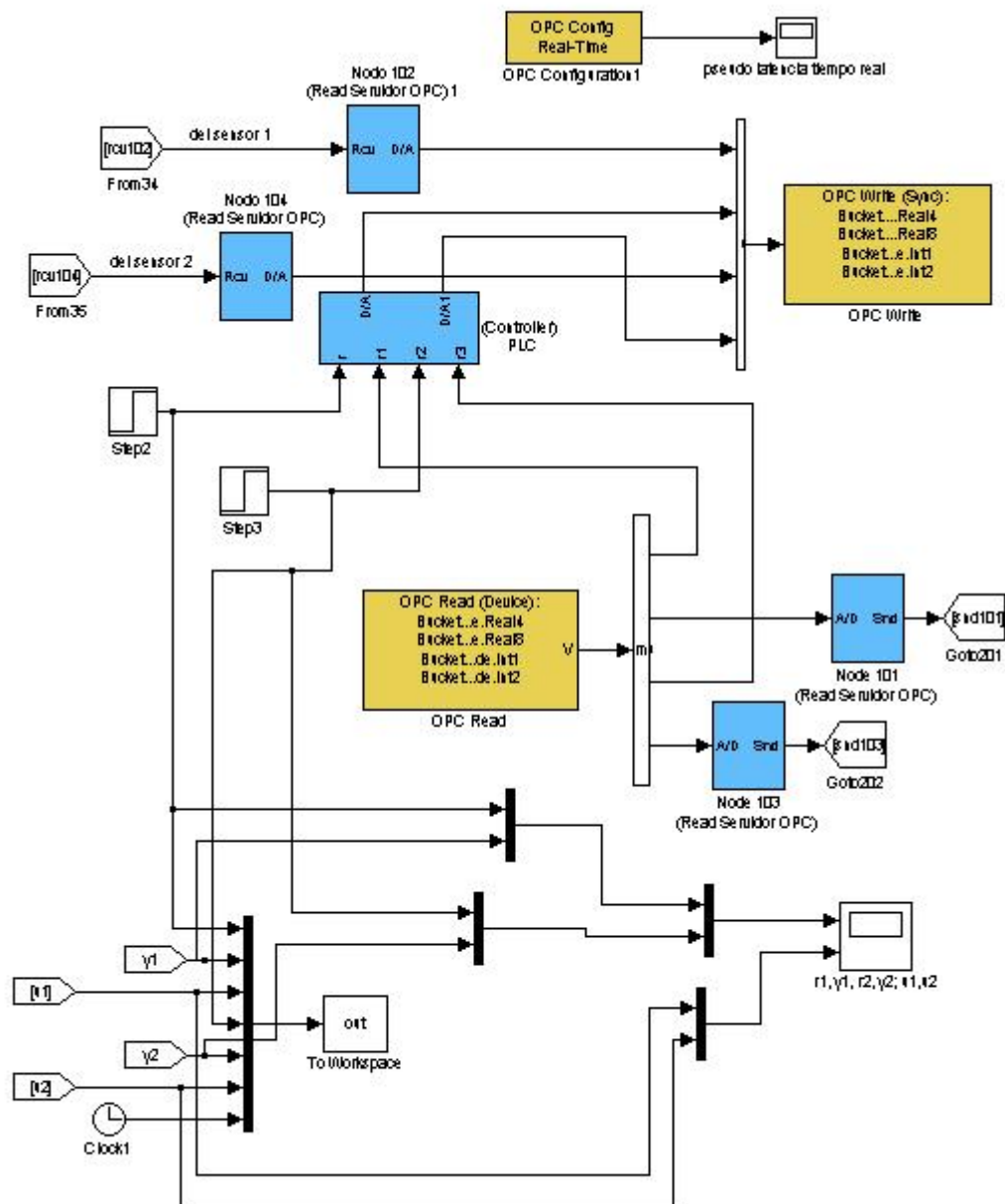


Figura 6.21: Servidor OPC y PLC (con su controlador)

Luego, al ejecutar la plataforma, se eligen los nodos con los que se trabajará (Figura 6.22).

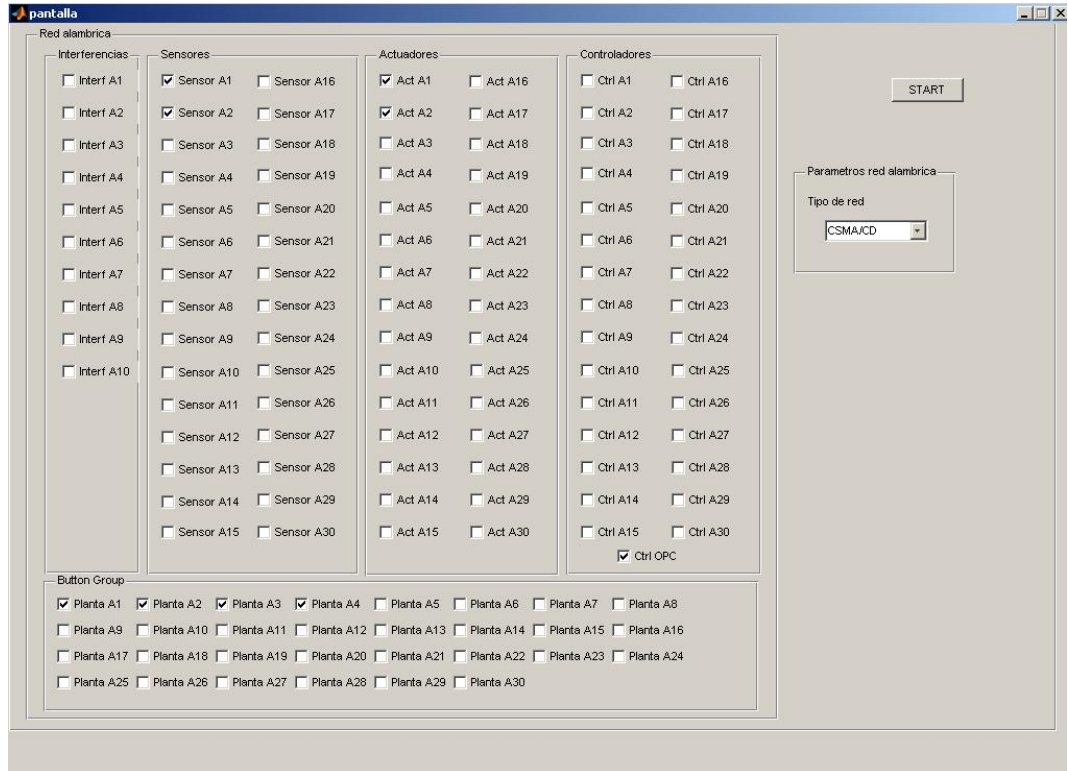


Figura 6.22: Elección de elementos con los que se armará el sistema en estudio

En el apéndice I se da una explicación detallada de los pasos a seguir para poder implementar este ejemplo en la plataforma.

Se probó el esquema de control utilizando controladores PI con valores $K_1=30$, $Ti_1 = 0.1418$ y $Td_1 = 0$ para la acción de control 1 y $K_2 = 30$, $Ti_2 = 0.2364$ y $Td_2 = 0$ para la acción de control 2. Se muestra el comportamiento del sistema en la Figura 6.23.

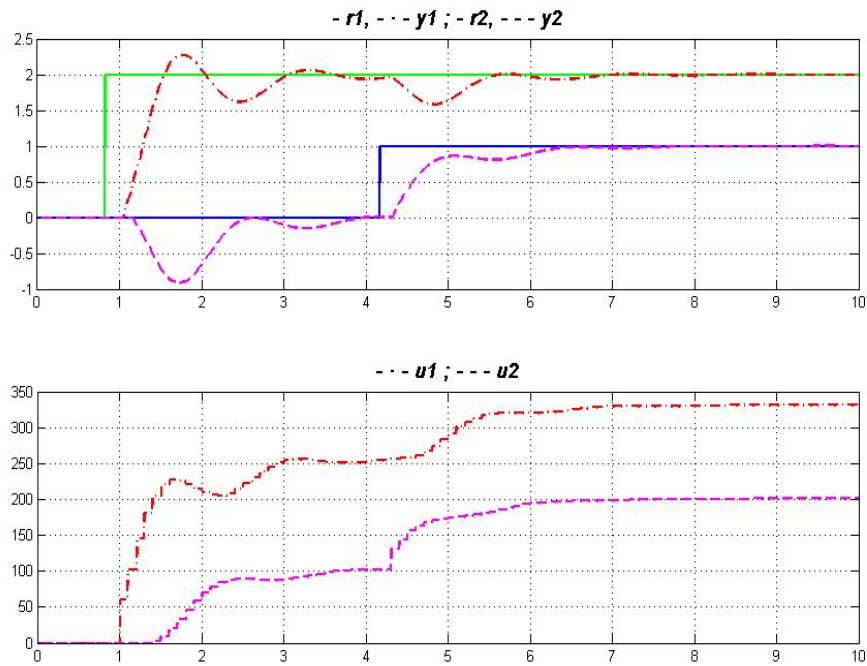


Figura 6.23: Respuesta a lazo cerrado empleando esquema de control feedback de tipo PI

Por medio de la plataforma se implementa un esquema de control descentralizado en tiempo real, con sensores, actuadores y controladores comunicados a través de una red de campo. Los elementos que forman parte del sistema se eligen a través de la pantalla principal, así como el perfil de red deseado (CSMA/CD, CSMA/AMP, Round Robin, FDMA, TDMA o Switched Ethernet) (Figura 6.17 y 6.22 respectivamente), y las conexiones se verifican en el archivo de Simulink[®] donde está implementada la plataforma -“plataforma.mdl”- (Figuras 6.15 y 6.16 para el primer caso y Figuras 6.19, 6.20 y 6.21 para el segundo). En este caso de estudio se muestra un ejemplo obtenido de la referencia consultada y se comprueba que la plataforma lo implementa sin inconvenientes.

6.2 Conclusión

En este capítulo se describieron tres ejemplos mediante los cuales se comprueba la versatilidad de la plataforma. La plataforma permite simular de forma sencilla esquemas de control de procesos en tiempo real. Se pueden implementar esquemas de control en cascada, establecer esquemas de control que

se ejecutan en un PLC donde el controlador se comunica con el sensor y con el actuador empleando el protocolo de comunicación OPC (eligiendo además el perfil de red deseado), implementar un esquema de control descentralizado con sensores, actuadores y controladores comunicados a través de una red de campo, entre otras aplicaciones.

Capítulo 7

Conclusiones finales

El objetivo principal del plan de trabajo propuesto consistió en el diseño e implementación de herramientas de simulación y monitoreo para verificación de la consistencia de datos y estrategias en un sistema distribuido de control en tiempo real. Como resultado final de estos análisis se implementó una plataforma genérica basada en TrueTime y OPC (*Ole for Process Control*) que servirá para futuros estudios de sistemas distribuidos de control.

La herramienta TrueTime permite la simulación de sistemas de tiempo real, tanto monoprocesador como distribuidos, y su utilización requiere un profundo conocimiento de sus comandos. La plataforma propuesta permite que un diseñador pueda simular y verificar su sistema distribuido o red de sensores de manera sencilla mediante la introducción de los parámetros que le son conocidos y no mediante la utilización de funciones específicas de la herramienta. Se pueden conectar sensores, actuadores, controladores e interferencias a la red de campo

con sólo elegir los nodos intervinientes en la pantalla principal y determinar cuáles de ellos formarán parte del lazo de control al verificar las conexiones en el archivo de Simulink donde está implementada la plataforma (“plataforma.mdl”). También posibilita utilizar diferentes perfiles de redes de campo (CSMA/CD, CSMA/AMP, Round Robin, FDMA, TDMA o Switched Ethernet) y configurar parámetros que definen el desempeño de dichas redes, así como comprobar que la estrategia de control implementada rechaza en forma correcta las perturbaciones producidas por el sistema de control digital.

La incorporación de comunicación mediante protocolo OPC permite generar ambientes de simulación de aplicaciones para la verificación de la correcta implementación de estrategias en controladores reales. Al admitir esta plataforma la utilización del protocolo OPC para generar en el controlador industrial bajo verificación, todo el entorno simulado de la aplicación, se logra que la misma pueda ser utilizada en la generación de escenarios para la verificación y calibración de estrategias implementadas en controladores industriales sin producir condiciones riesgosas en la planta real.

Apéndice I

Detalles de Casos de Estudio

La utilización de la plataforma requiere tener instalado TrueTime. Para efectuar esto, se copia la carpeta de TrueTime en el directorio deseado (por ejemplo en C:\). Además se debe tener instalado un compilador C++ (por ejemplo Visual Studio C++). Se debe configurar también la variable de entorno TTKERNEL. Para ello, ir a Configuración → Panel de Control → Sistema → Opciones avanzadas → Variables de entorno → Variables de sistema → Nueva. El nombre de la variable será TTKERNEL y su valor, el path donde está ubicada la carpeta *kernel* de TrueTime (por ejemplo, C:\truetime-1.5\kernel). Además, al final del archivo `matlabrc` de Matlab[®], ubicado en el directorio *local* del directorio *toolbox* de Matlab[®] (por ejemplo: C:\Archivos de programa\MATLAB\R2008a\toolbox\local\matlabrc.m) agregar las sentencias:

```
addpath([getenv('TTKERNEL')])
init_truetime;
mex -setup
```

make_truetype;

Ahora, para poder trabajar, se ejecuta Matlab[®] y en línea de comando aparecerá:

Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n?

Se ingresa la opción “y <ENTER>”. Aparece entonces

Select a compiler:

[1] Lcc-win32 C 2.4.1 in C:\ARCHIV~1\MATLAB\R2008a\sys\lcc\bin

[2] Microsoft Visual C++ 6.0 in C:\Archivos de programa\Microsoft Visual Studio

[0] None

Compiler:

Se ingresa la opción “2 <ENTER>”. Aparecen las líneas:

Please verify your choices:

Compiler: Microsoft Visual C++ 6.0

Location: C:\Archivos de programa\Microsoft Visual Studio

Are these correct [y]/n?

Se ingresa la opción “y <ENTER>”. Comienza entonces el proceso de compilación:

Compiling TrueTime kernel block...

...done.

Compiling TrueTime network block...

...done.

Compiling TrueTime wireless network block...

...done.

Compiling TrueTime MEX-functions...

ttAbortSimulation.cpp

ttAnalogIn.cpp

ttAnalogOut.cpp

ttAttachDLHandler.cpp

ttAttachWCETHandler.cpp

ttCallBlockSystem.cpp

.....

ttSleep.cpp

ttSleepUntil.cpp

ttTake.cpp

ttTryFetch.cpp
ttTryPost.cpp
ttWait.cpp
TrueTime compiled successfully!

Verificar luego, dentro de Matlab[®], estar ubicado en el directorio donde se encuentran los archivos de la plataforma.

Y en el archivo “pantalla.m” se debe verificar que en la línea:

```
fid = fopen('C:\truetime-1.5\examples\4_cableada win
           OPC_original\monitor_t.txt','a');
```

aparezca el path correcto en el que se encuentra ubicado el archivo “monitor_t.txt”.

Con respecto al monitor de consistencia, para su correcto funcionamiento se debe verificar que en el archivo “monitor.m” figure el path correcto. Por ejemplo, en “Control de temperatura de bobinado”,

```
fid=fopen('C:\truetime-1.5\examples\4_cableada win OPC_Ej1 (cascada
          serie)\cascada serie completa\monitor_t.txt','r');
```

Caso de Estudio “Control temperatura bobinado”

Esquema de control en cascada

Ubicado en el directorio correspondiente, lo primero que el usuario debe hacer es abrir el modelo de Simulink[®] “plataforma.mdl” y verificar que las conexiones sean las adecuadas para el caso en estudio.

Se trabajará en este caso con el sensor₁, sensor₂, el controlador₁, el controlador₂, la planta A₁, la planta A₂, la planta A₃ y el actuador₂ (ver Figuras I.1 a I.2).

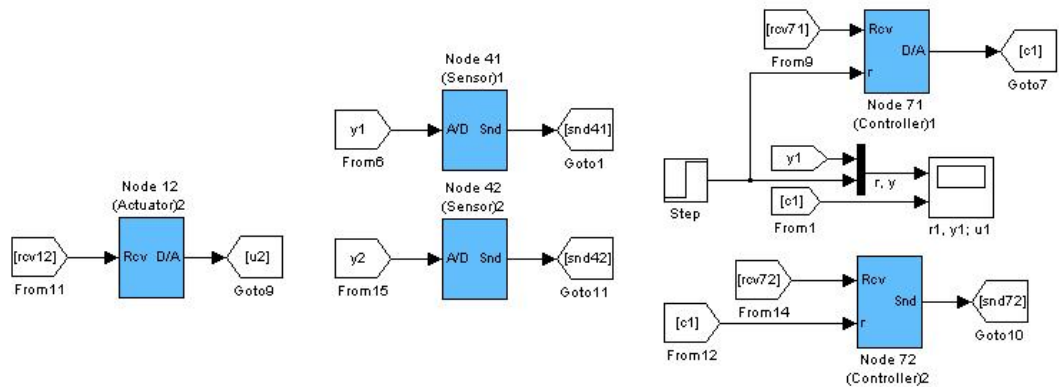


Figura I.1: Bloques sensor, actuador y controlador utilizados en este ejemplo

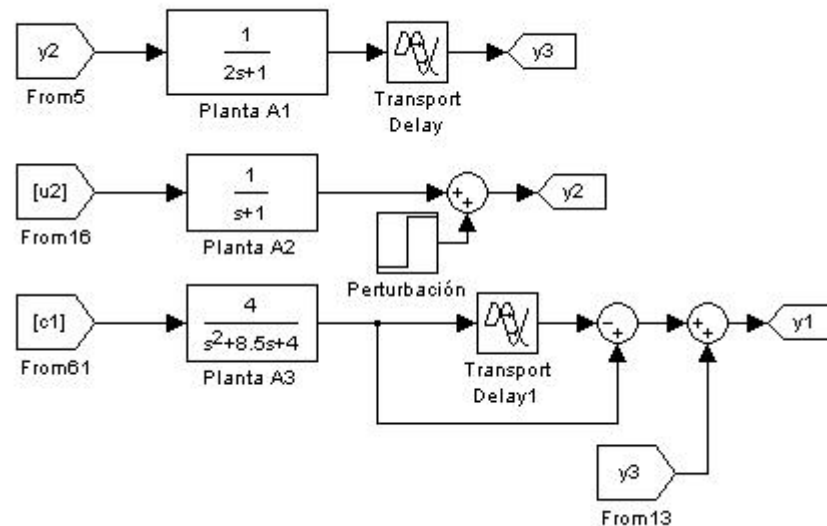


Figura I.2: Bloques plantas utilizados en este ejemplo

Como se ve en la Figura I.1, el sensor₂ lee la salida de la planta 2. Se debe verificar que el sensor₂ envíe los datos al controlador₂, esto se logra configurando en *ttSendMsg* del archivo “senscode2.m” al nodo 72 como al que se envían los datos.

```
ttSendMsg([1 72], data.y, 80); % Send message (80 bits) to node 72 de red 1
```

El nodo 72 es un nodo por medio del cual el controlador recibe los datos del sensor.

Se debe verificar además que la referencia para el controlador₂ sea la señal que calcula el controlador₁. Con ese fin se agrega un bloque *From* con la señal que denominaremos c_1 (se hace doble clic en el bloque *From* y se escribe c_1).

El valor calculado por el controlador₂ es volcado a la red y será leído por el actuador₂. Por ello el usuario debe verificar que en la función

```
ttSendMsg(12, data.u, 80); % Send 80 bits to node 12 (actuador)
```

del archivo “ctrlcode2.m” figure el número de nodo del actuador que debe leer el valor y actuarlo.

En el archivo de inicialización del bloque controlador₂ (“controller2_init.m”) se deben ingresar los valores de los parámetros de sintonización del controlador esclavo, en este caso: $K_{c2}=8$, $Td_{c2}=0$, $Ti_{c2}=1$.

La planta A_2 tendrá como entrada la salida del actuador₂ y como salida una parte de la señal y_2 , ya que la otra parte proviene de la perturbación (ver Figura I.2).

El sensor₁ lee la salida de la planta A_1 denominada y_3 más la señal que viene del predictor (Figura I.2) y esa señal la envía al controlador₁, el cual la recibe por red. Esto se logra configurando en *ttSendMsg* del archivo “senscode1.m” al nodo 71 como al que se envían los datos.

```
ttSendMsg([1 71], data.y, 80); % Send message (80 bits) to node 71 de red 1
```

El controlador₁ utiliza el nodo 71 para recibir los datos del sensor₁.

El controlador₁ envía su valor calculado por el puerto *D/A* al controlador₂, para lograr eso se debe hacer doble clic con el botón izquierdo del mouse sobre el bloque controlador₁, dentro de la plataforma, y acomodar las salidas como se ve en la Figura I.3. También se deben modificar los archivos *.m del controlador, en “controller1_init.m”, verificar que *ttInitKernel* tenga una entrada *A/D* y una salida *D/A*, es decir:

```
ttInitKernel(1, 1, 'prioFP'); % nbrOfInputs, nbrOfOutputs, fixed priority
```

Y en el “ctrlcode1.m”, reemplazar el

```
ttSendMsg([1 11], data.u, 80);
```

por un

```
ttAnalogOut(1, data.u);
```

porque es una salida por puerto *D/A* y no por red.

Para lograr que en el bloque controlador₁ se lea dentro del bloque *D/A*, hacer clic con el botón derecho del mouse sobre el bloque y elegir *Explore*. Sobre *snd* hacer doble clic y cambiarlo por *D/A*, así aparecerá reflejada esa leyenda dentro del bloque.

Además en la red, en el *snd* del nodo 71 hay que colocar un bloque *Ground*.

Los valores del controlador maestro serán: $K_{c1}=9$, $Td_{c1}=0$ y $Ti_{c1}=2$, verificarlo en “controller1_init.m”.

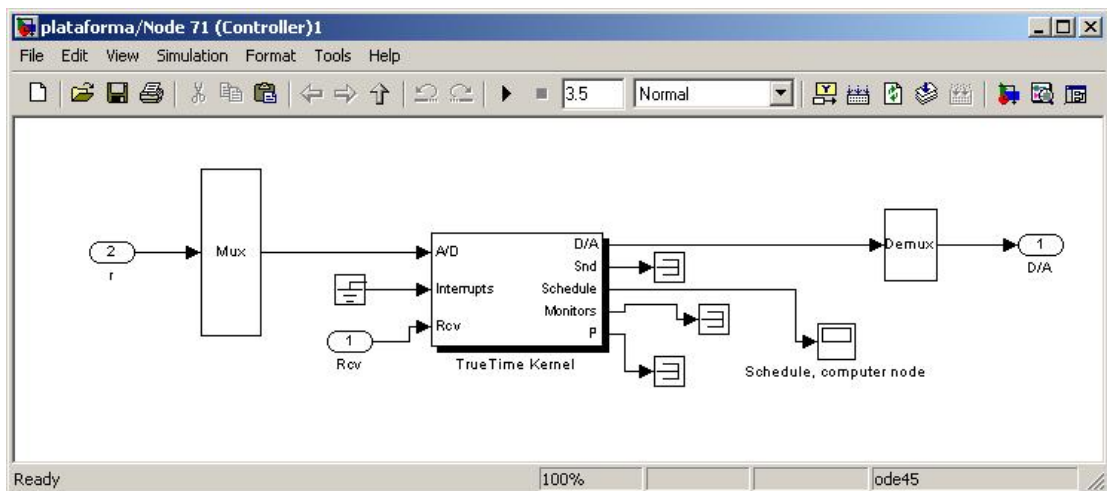


Figura I.3: Bloque controlador1 por dentro

La planta A_1 tendrá como entrada la señal y_2 y como salida la señal y_3 (ver Figura I.2). Hacer doble clic sobre el bloque retardo e ingresar el valor 1.

La planta A_3 tendrá como entrada la salida del controlador c_1 y como salida la señal y_1 (ver Figura I.2). Hacer doble clic sobre el bloque retardo e ingresar el valor 1.

Además para poder ver el comportamiento del sistema se tiene el gráfico $r_1, y_1; u_1$ (ver Figura I.1), pero el usuario debe comprobar que las entradas a dicho gráfico sean las correctas, en este caso la referencia r_1 , el valor recibido por el sensor₁, y_1 y la señal actuada sobre la planta (salida del controlador₂) u_2 .

Una vez verificado todo lo anterior, el usuario debe cerrar “plataforma.mdl”.

Además debe verificar las prioridades de los nodos, porque ello puede afectar la respuesta del sistema (en este caso el sensor₁ tiene prioridad 1, el controlador₁ prioridad 2; el sensor₂ y el actuador₂ tienen prioridad 3 y el controlador₂ prioridad 4).

En línea de usuario introduce el siguiente comando:

```
>> pantalla <ENTER>
```

Aparecerá entonces la pantalla principal, en donde el usuario deberá marcar los elementos con los que se trabajará en este caso (ver Figura I.4)

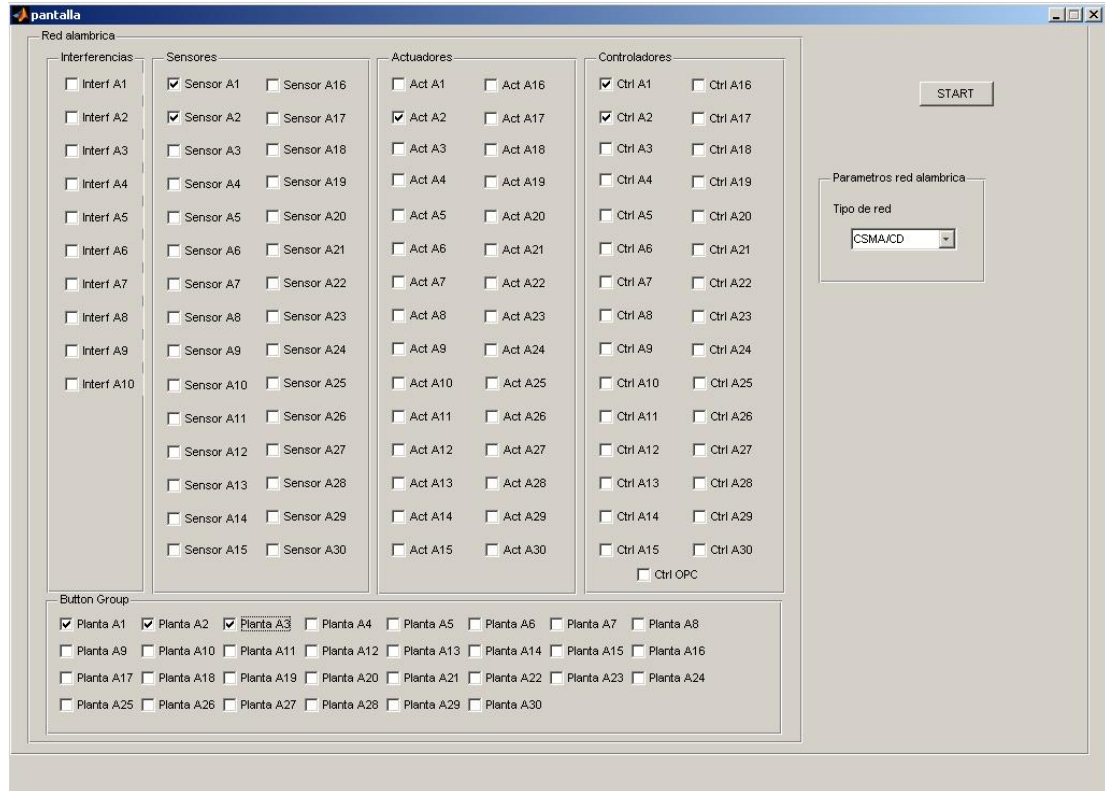


Figura I.4: Elección de elementos con los que se armará el sistema en estudio

Se selecciona sensor₁, sensor₂, el controlador₁, el controlador₂, la planta A₁, la planta A₂, planta A₃ y el actuador₂. Al seleccionar planta A₁ aparece la subpantalla mostrada en la Figura I.5 y al seleccionar planta A₂ aparece la subpantalla mostrada en Figura I.6.

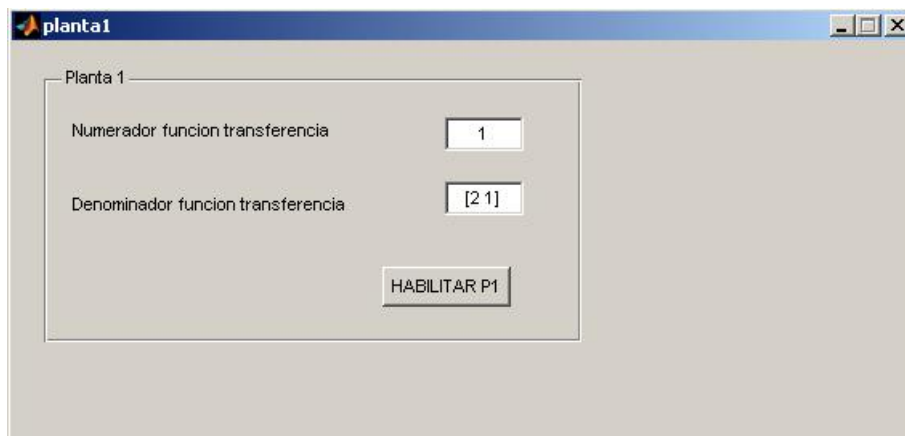


Figura I.5: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A1 (no ingresar e-s porque está representada por un bloque retardo en “plataforma.mdl”)

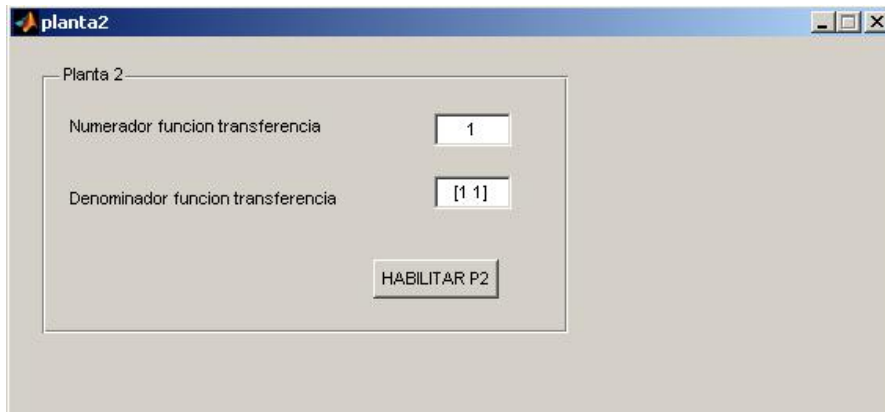


Figura I.6: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A2

Una vez ingresados los vectores que representan numerador y denominador de cada función de transferencia, se clikea sobre el icono “Habilitar P1” o “Habilitar P2”, según sea el caso. Las subpantallas se cierran. Sobre la pantalla principal (Figura I.4) se elige el tipo de red cableada a utilizar y se hace clic sobre el ícono *Start*. Así da comienzo a la simulación.

Aparecen ahora otras subpantallas para que el usuario ingrese valores deseados a los parámetros que definen el desempeño de la red. Una vez ingresados se debe hacer clic sobre el botón “Habilitar” y esos valores serán ingresados durante la corrida.

Cuando la simulación finaliza, se puede hacer doble clic sobre el gráfico correspondiente (ver Figura I.1) para ver la evolución del sistema. Con el botón derecho del mouse se hace clic sobre cada uno de los gráficos (r, y_1 y u_2) y se elige la opción *Autoescale*, así se podrá ver el gráfico completo.

Caso de Estudio: “Control distribuido utilizando OPC”

Ubicado en el directorio correspondiente, lo primero que el usuario debe hacer es abrir el modelo de Simulink “plataforma.mdl” y verificar que las conexiones sean las adecuadas para el caso en estudio.

Se trabajará en este caso con la interferencia 1, el sensor 1, el actuador 1, la planta A_1 , el servidor OPC y el controlador ubicado en el PLC (ver Figuras I.7 a I.9).

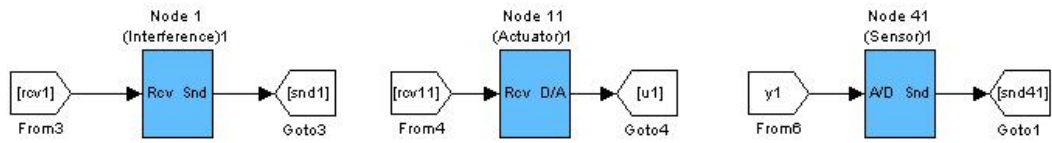


Figura I.7: Bloques interferencia, sensor y actuador utilizados en este ejemplo

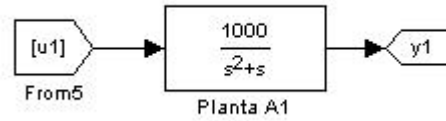


Figura I.8: Bloque planta utilizado en este ejemplo

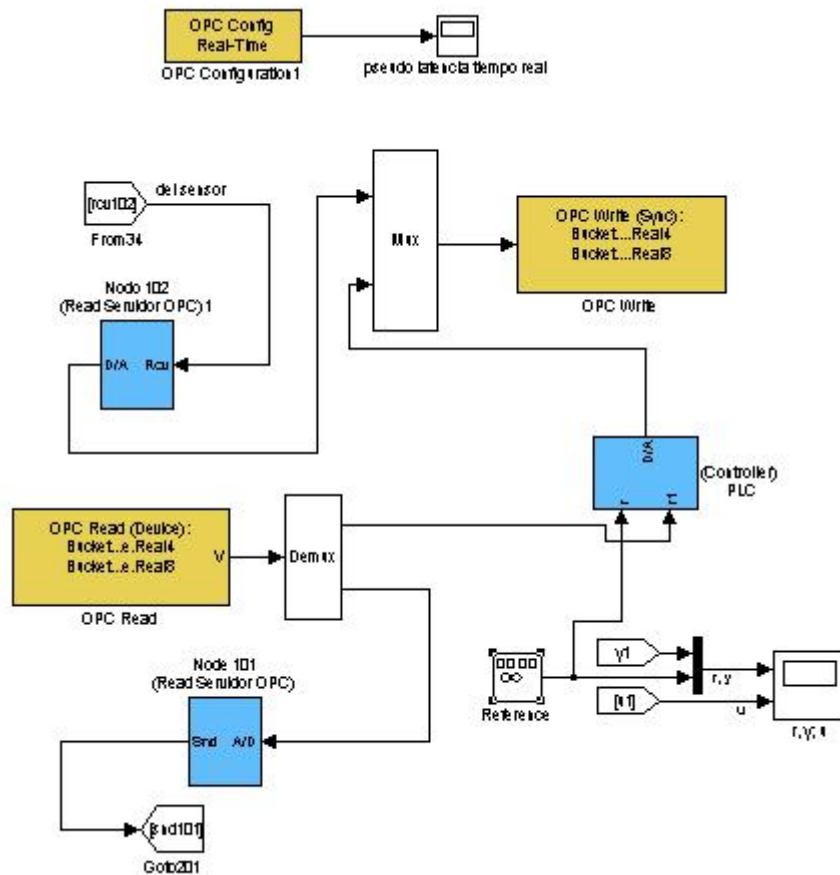


Figura I.9: Servidor OPC y PLC (con su controlador)

Como se ve en la Figura I.7, el sensor₁ lee la salida de la planta 1.

Se debe verificar que el sensor₁ envíe los datos al servidor OPC (y éste lo enviará al controlador ubicado en el PLC), esto se logra configurando en *ttSendMsg* del archivo “senscode1.m” al nodo 102 como al que se envían los datos.

```
ttSendMsg([1 102], data.y, 80); % Send message (80 bits) to node 102 OPC de red 1
```

El servidor OPC recibe datos por medio del nodo 102 y los escribe en él a través de la función *OPC Write*. Luego con la función *OPC Read* el controlador del PLC puede leerlos (entrada *A/D* denominada r_1)

El valor calculado por el controlador es volcado en el servidor OPC a través de la función *OPC Write*, y luego a través de la función *OPC Read* se envía el valor al actuador. Por ello el usuario debe verificar que en la función

```
ttSendMsg(11, data.u, 80); % Send 80 bits to node 11 (actuador)
```

del archivo “OPC_readcode.m” figure el número de nodo del actuador que debe leer el valor y actuarlo.

En el archivo de inicialización del bloque controlador_{OPC} (“OPC_controller_init.m”) se deben ingresar los valores de los parámetros de sintonización del controlador PID, en este caso: $K=1.1$, $Td=0.06$, $Ti=1$.

Además para poder ver el comportamiento del sistema se tiene el gráfico $r, y; u$ (ver Figura I.9), pero el usuario debe comprobar que las entradas a dicho gráfico sean las correctas, en este caso la referencia r , la salida de la planta (valor recibido por el sensor) y_1 y la señal actuada sobre la planta (salida del controlador ubicado en el PLC) u_1 .

Una vez verificado todo lo anterior, el usuario debe cerrar “plataforma.mdl”.

Además debe verificar las prioridades de los nodos, porque ello puede afectar la respuesta del sistema (en este caso el sensor₁ y el actuador₁ tiene prioridad 1, el controlador₁ prioridad 3 y los bloques TrueTime Kernel que representan a los nodos 101 y 102 prioridad 2).

En línea de usuario introduce el siguiente comando:

```
>> pantalla <ENTER>
```

Aparecerá entonces la pantalla principal, en donde el usuario deberá marcar los elementos con los que se trabajará en este caso (ver Figura I.10)

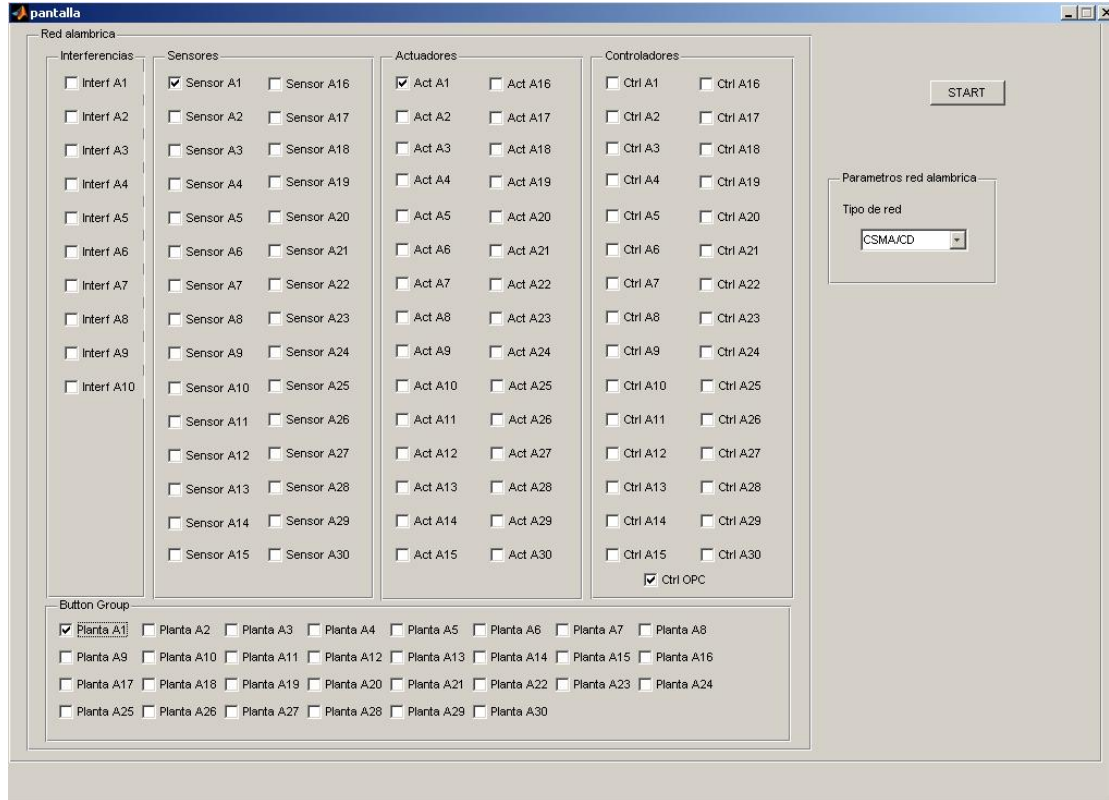


Figura I.10: Elección de elementos con los que se armará el sistema en estudio

Se selecciona sensor_1 , actuador_1 , $\text{controlador}_{\text{OPC}}$ y $\text{planta}_{\text{A1}}$. Al seleccionar esta última aparece la subpantalla mostrada en la Figura I.11.

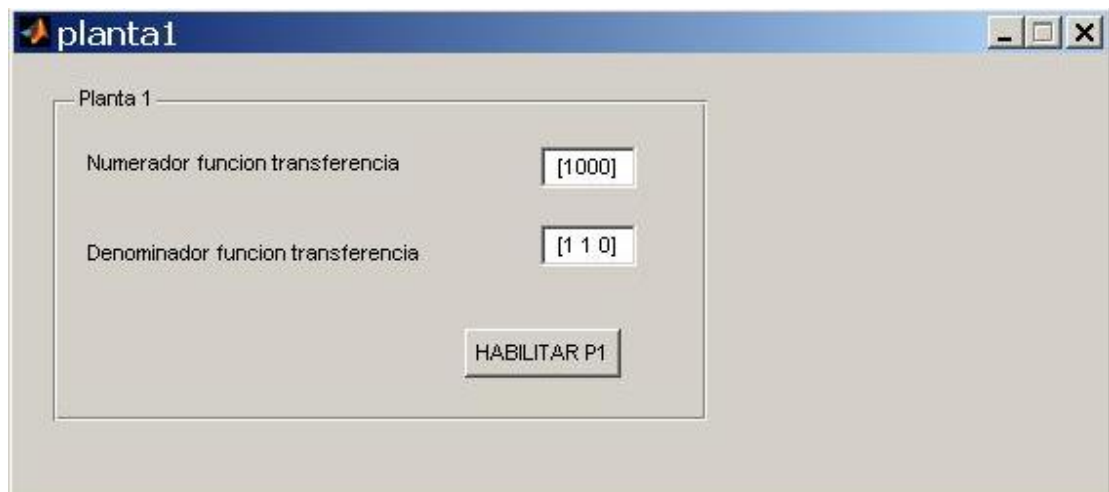


Figura I.11: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta

Una vez ingresados los vectores que representan numerador y denominador de la función de transferencia, se clikea sobre el icono “Habilitar P1”. La subpantalla se cierra. Sobre la pantalla principal (Figura I.10) se elige el tipo de red cableada a utilizar y se hace clic sobre el ícono *Start*. Así da comienzo a la simulación.

Aparecen ahora otras subpantallas para que el usuario ingrese valores deseados a los parámetros que definen el desempeño de la red. Una vez ingresados se debe hacer clic sobre el botón “Habilitar” y esos valores serán ingresados durante la corrida.

Cuando la simulación finaliza, se puede hacer doble clic sobre el gráfico correspondiente (ver Figura I.9) para ver la evolución del sistema. Con el botón derecho del mouse se hace clic sobre cada uno de los gráficos (r,y ; y u) y se elige la opción *Autoescale*, así el usuario podrá ver el gráfico completo.

En el caso de querer correr la simulación agregando interferencia en la red, en la pantalla principal también se debe seleccionar Interf_1 (Figura I.10). Todos los demás pasos son similares al caso sin interferencia.

Caso de Estudio “Columna de destilación”

Simulación autónoma sin utilización de OPC

Para acceder al ejemplo de este caso de estudio, el usuario debe acceder al directorio correspondiente para abrir el modelo de Simulink[®] “plataforma.mdl” y verificar que las conexiones sean las adecuadas para el caso en estudio.

Se trabajará en este caso con el sensor₁, sensor₂, el controlador₁, el controlador₂, la planta A₁, la planta A₂, la planta A₃, la planta A₄, el actuador₁ y actuador₂ (ver Figuras I.12 a I.13).

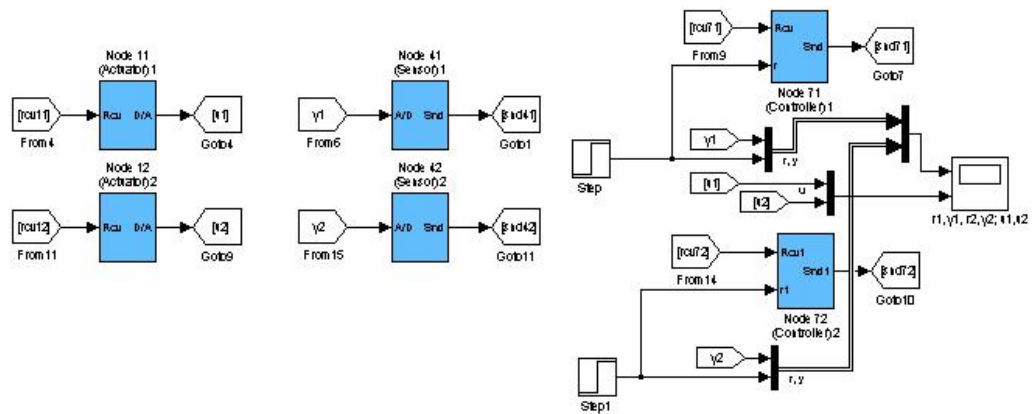


Figura I.12: Bloques sensor, actuador y controlador utilizados en este ejemplo

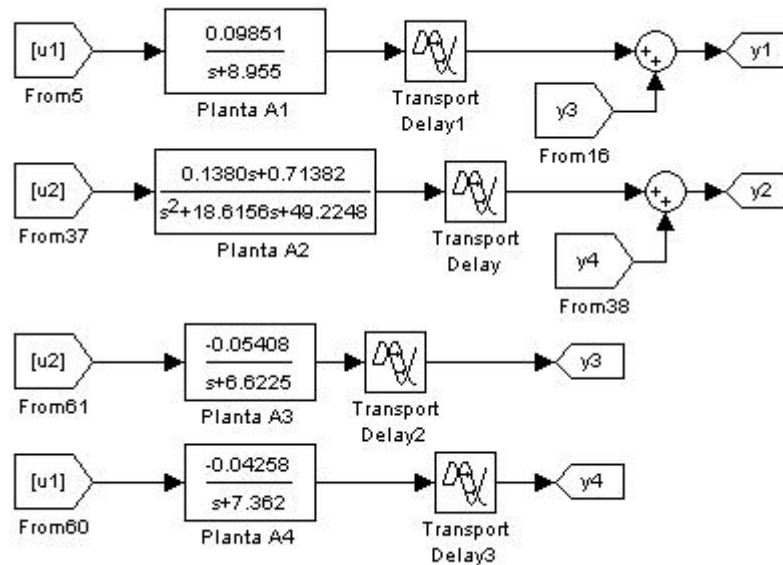


Figura I.13: Bloques plantas utilizados en este ejemplo

Como se ve en la Figura I.12, el sensor₁ lee la salida de la planta 1. Se debe verificar que el sensor₁ envíe los datos al controlador₁, esto se logra configurando en *ttSendMsg* del archivo “senscode1.m” al nodo 71 como al que se envían los datos.

```
ttSendMsg([1 71], data.y, 80); % Send message (80 bits) to node 71 de red 1
```

Así también el sensor₂ lee la salida de la planta 2. Se debe verificar que el sensor₂ envíe los datos al controlador₂, esto se logra configurando en *ttSendMsg* del archivo “senscode2.m” al nodo 72 como al que se envían los datos.

```
ttSendMsg([1 72], data.y, 80); % Send message (80 bits) to node 72 de red 1
```

En el archivo de inicialización del bloque controlador (“controller1_init.m”) se deben ingresar los valores de los parámetros de sintonización del controlador₁, en este caso: $K_{c1}=60.09$, $Td_{c1}=0$, $Ti_{c1}=0.1117$. Además en el “ctrlcode1.m” se debe verificar que los cálculos del controlador sean enviados al actuador₁:

```
ttSendMsg([1 11], data.u, 80); % Send 80 bits to node 11 (actuador)1 en red 1
```

En el archivo de inicialización del bloque controlador₂ (“controller2_init.m”) se deben ingresar los valores de los parámetros de sintonización del controlador₂, en este caso: $K_{c2}=60.09$, $Td_{c2}=0$, $Ti_{c2}=0.1898$. Además en el “ctrlcode2.m” se debe verificar que los cálculos del controlador sean enviados al actuador₂:

```
ttSendMsg([1 12], data.u, 80); % Send 80 bits to node 12 (actuador)2 en red 1
```

La planta A₁ tendrá como entrada la salida del actuador₁ y como salida la señal y_1 , que es la sumatoria de la salida de planta en sí más la señal y_3 (Figura I.13). Hacer doble clic sobre el bloque retardo e ingresar el valor 0.043.

La planta A₂ tendrá como entrada la salida del actuador₂ y como salida la señal y_2 , que es la sumatoria de la salida de planta en sí más la señal y_4 (Figura I.13). Hacer doble clic sobre el bloque retardo e ingresar el valor 0.017.

La planta A₃ tendrá como entrada la salida del actuador₂ y como salida la señal y_3 (Figura I.13). Hacer doble clic sobre el bloque retardo e ingresar el valor 0.017.

La planta A₄ tendrá como entrada la salida del actuador₁ y como salida la señal y_4 (Figura I.13). Hacer doble clic sobre el bloque retardo e ingresar el valor 0.153.

Además para ver el comportamiento del sistema se tiene el gráfico $r_1, y_1, r_2, y_2, u_1, u_2$ (ver Figura I.12). El usuario debe comprobar que las entradas a dicho gráfico sean las correctas, en este caso la referencia r_1 , el valor recibido por el sensor₁: y_1 , la referencia r_2 , el valor recibido por el sensor₂: y_2 , la señal actuada sobre la planta1 (salida del controlador₁) u_1 y la señal actuada sobre la planta2 (salida del controlador₂) u_2 .

El usuario debe verificar las prioridades de los nodos, porque ello puede afectar la respuesta del sistema (en este caso el sensor₁ y sensor₂ tienen prioridad 1, el controlador₁ y controlador₂ prioridad 2; el actuador₁ y el actuador₂ tienen prioridad 1).

Una vez verificado todo lo anterior, el usuario debe cerrar “plataforma.mdl”.

En línea de usuario introduce el siguiente comando:

>> *pantalla* <ENTER>

Aparecerá entonces la pantalla principal, en donde el usuario deberá marcar los elementos con los que se trabajará en este caso (ver Figura I.14)

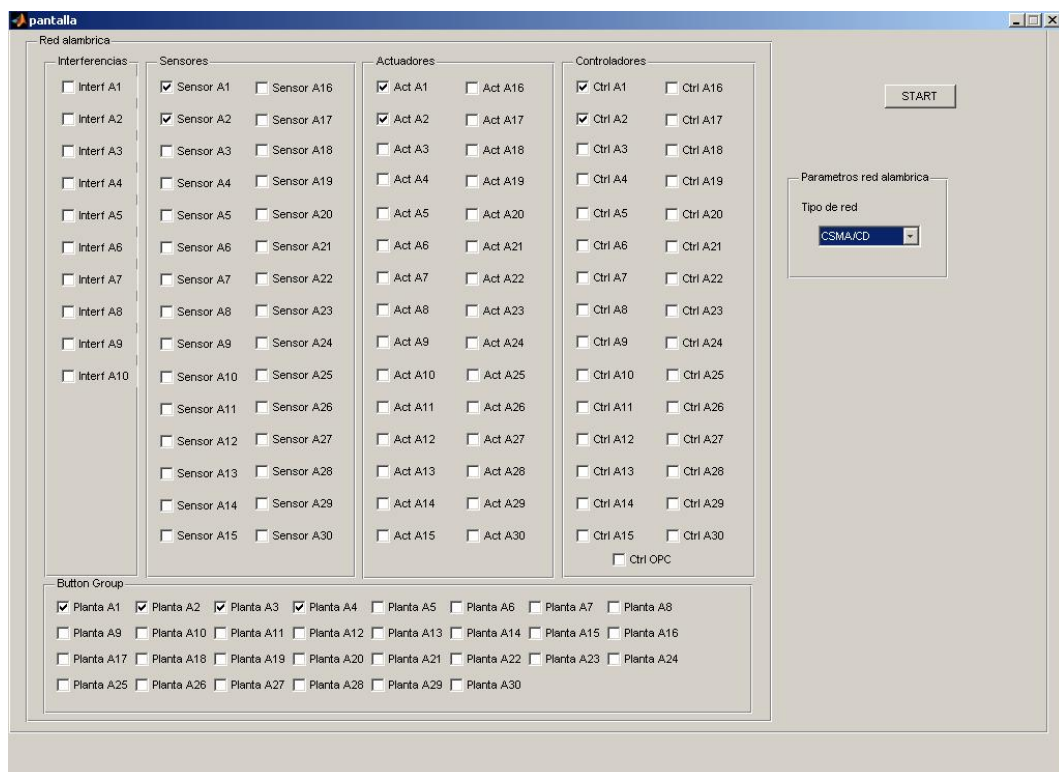


Figura I.14: Elección de elementos con los que se armará el sistema en estudio

Se selecciona sensor₁, sensor₂, el controlador₁, el controlador₂, la planta A₁, la planta A₂, planta A₃, planta A₄, el actuador₁ y el actuador₂. Al seleccionar planta A₁ aparece la subpantalla mostrada en la Figura I.15, al seleccionar planta A₂ aparece la subpantalla mostrada en Figura I.16, al seleccionar planta A₃ aparece la subpantalla mostrada en Figura I.17 y al seleccionar planta A₄ aparece la subpantalla mostrada en Figura I.18.

Figura I.15: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A1 (no ingresar e-0.043s porque está representado por un bloque retardo en “plataforma.mdl”)

Figura I.16: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A2 (no ingresar e-0.017s porque está representado por un bloque retardo en “plataforma.mdl”)

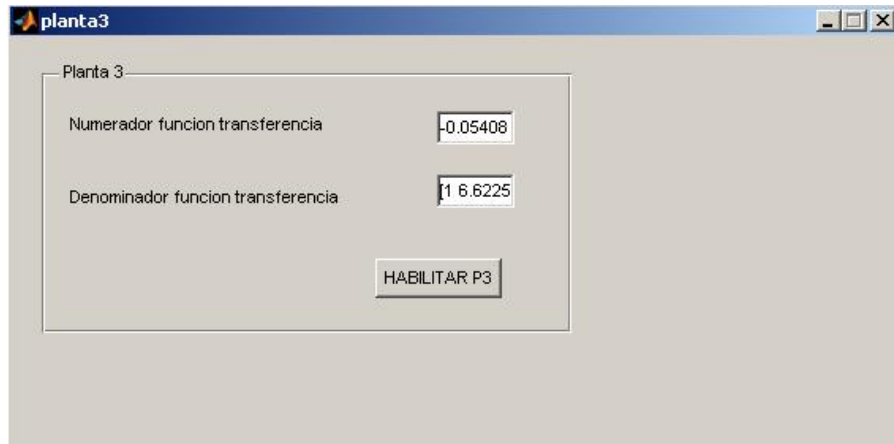


Figura I.17: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A3 (no ingresar e-0.017s porque está representado por un bloque retardo en “plataforma.mdl”)

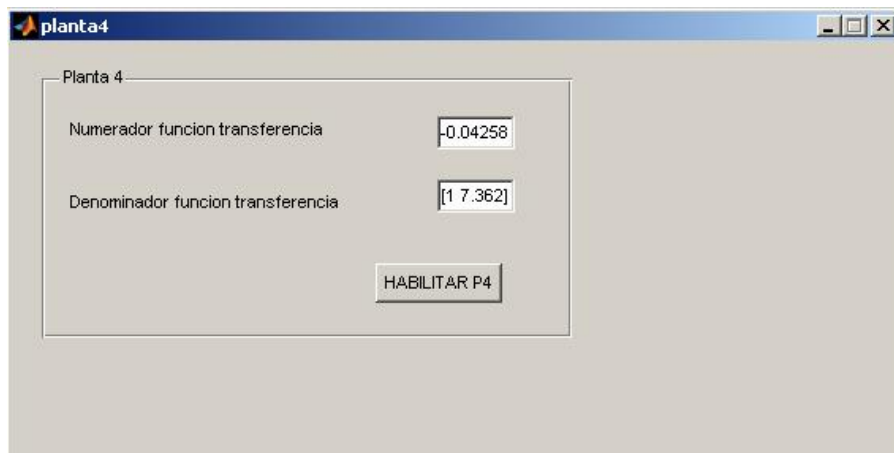


Figura I.18: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A4 (no ingresar e-0.153s porque está representado por un bloque retardo en “plataforma.mdl”)

Una vez ingresados los vectores que representan numerador y denominador de cada función de transferencia, se clikea sobre el icono “Habilitar P1”, “Habilitar P2”, “Habilitar P3” o “Habilitar P4”, según sea el caso. Las subpantallas se cierran. Sobre la pantalla principal (Figura I.14) se elige el tipo de red cableada a utilizar y se hace clic sobre el ícono *Start*. Así da comienzo a la simulación.

Aparecen ahora otras subpantallas para que el usuario ingrese valores deseados a los parámetros que definen el desempeño de la red. Una vez ingresados se debe hacer clic sobre el botón “Habilitar” y esos valores serán ingresados durante la corrida.

Cuando la simulación finaliza, se puede hacer doble clic sobre el gráfico correspondiente (ver Figura I.12) para ver la evolución del sistema. Con el botón derecho del mouse se hace clic sobre cada uno de los gráficos (r_1, y_1 , r_2, y_2 ; y u_1, u_2) y se elige la opción *Autoescale*, así se podrá ver el gráfico completo.

Simulación con comunicación al controlador utilizando OPC

Para acceder al ejemplo de este caso de estudio, el usuario debe acceder al directorio correspondiente para abrir el modelo de Simulink® “plataforma.mdl” y verificar que las conexiones sean las adecuadas para el caso en estudio.

Se trabajará en este caso con el sensor₁, sensor₂, el controlador_{OPC}, la planta A₁, la planta A₂, la planta A₃, la planta A₄, el actuador₁ y actuador₂ (ver Figuras I.19 a I.21).

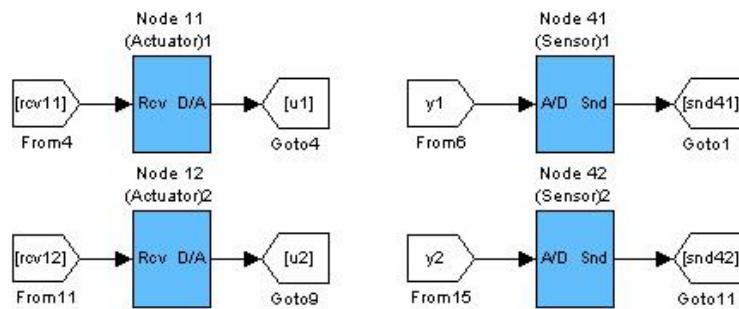


Figura I.19: Bloques sensor, actuador utilizados en este ejemplo

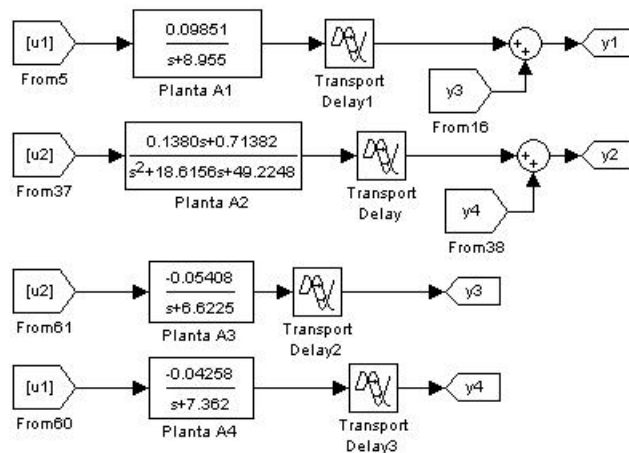


Figura I.20: Bloques plantas utilizados en este ejemplo

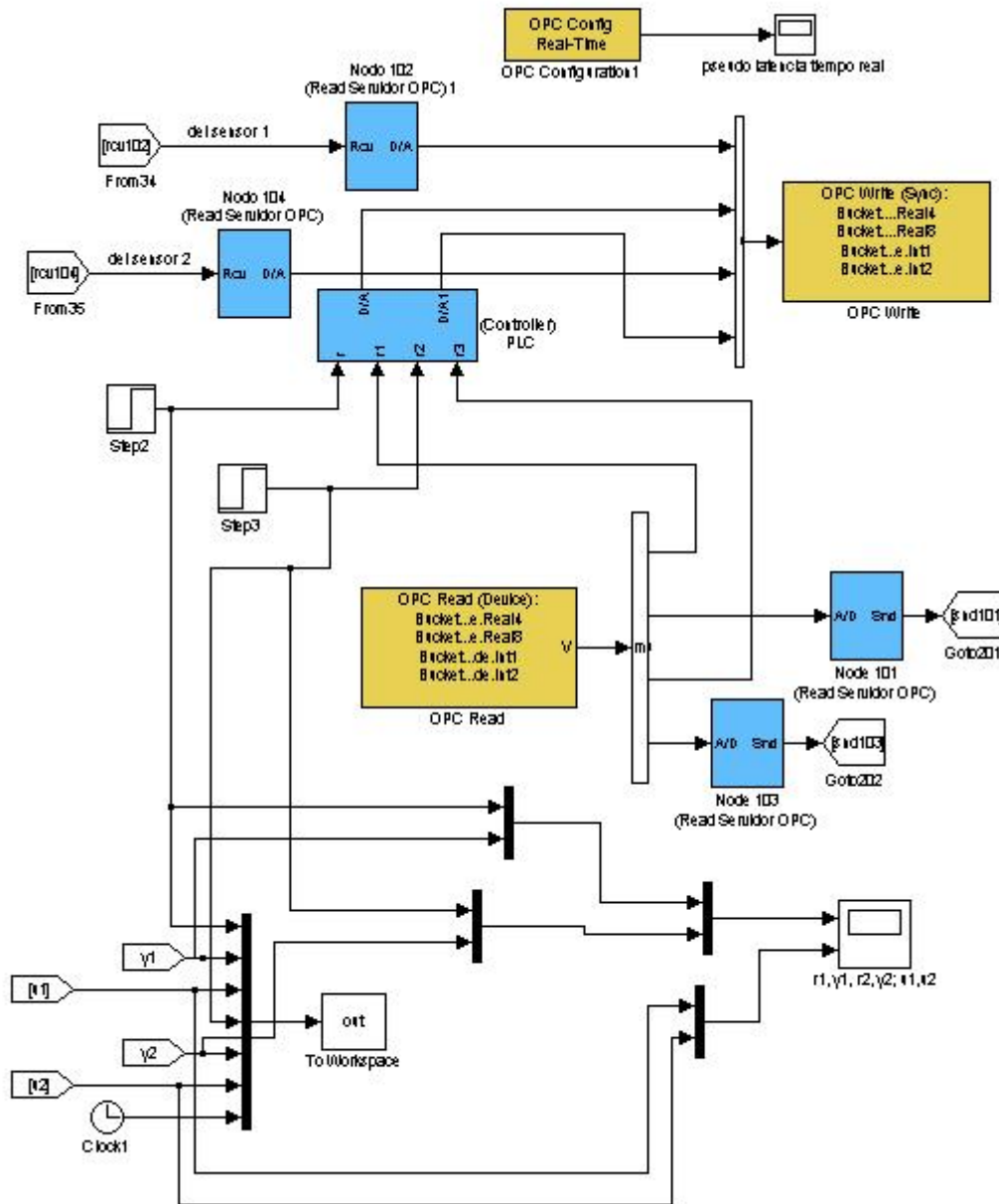


Figura I.21: Servidor OPC y PLC (con su controlador)

Como se ve en la Figura I.19, el sensor₁ lee la salida de la planta 1. Se debe verificar que el sensor₁ envíe los datos al servidor OPC (y éste lo enviará al controlador ubicado en el PLC), esto se logra configurando en *ttSendMsg* del archivo “senscode1.m” al nodo 102 como al que se envían los datos.

```
ttSendMsg([1 102], data.y, 80); % Send message (80 bits) to node 102 OPC de red 1
```

El servidor OPC recibe datos por medio del nodo 102 y los escribe en él a través de la función *OPC Write*. Luego con la función *OPC Read* el controlador del PLC puede leerlos (entrada *A/D* denominada r_1)

Así también el sensor₂ lee la salida de la planta 2. Se debe verificar que el sensor₂ envíe los datos al servidor OPC (y éste lo enviará al controlador ubicado en el PLC), esto se logra configurando en *ttSendMsg* del archivo “senscode2.m” al nodo 104 como al que se envían los datos.

```
ttSendMsg([1 104], data.y, 80); % Send message (80 bits) to node 104 OPC de red 1
```

El servidor OPC recibe datos por medio del nodo 104 y los escribe en él a través de la función *OPC Write*. Luego con la función *OPC Read* el controlador del PLC puede leerlos (entrada *A/D* denominada r_3)

El valor calculado por el controlador es volcado en el servidor OPC a través de la función *OPC Write*, y luego a través de la función *OPC Read* se envía el valor al actuador₁ o al actuador₂ según corresponda.

Por ello el usuario debe verificar que en el archivo “OPC_readcode.m”, en la función

```
ttSendMsg(11, data.u, 80); % Send 80 bits to node 11 (actuador 1)
```

figure el número de nodo del actuador₁ que debe leer el valor y actuarlo; y en el archivo “OPC_readcode3.m”, en la función

```
ttSendMsg(12, data.u, 80); % Send 80 bits to node 12 (actuador 2)
```

figure el número de nodo del actuador₂ que debe leer el valor y actuarlo.

En el archivo de inicialización del bloque controlador_{OPC} (“OPC_controller_init.m”) se deben ingresar los valores de los parámetros de sintonización del controlador PI, en este caso, para controlar el primer lazo: $K_1 =$

30, $Td_1 = 0$, $Ti_1 = 0.1418$ y para controlar el segundo lazo: $K_2 = 30$, $Td_2 = 0$, $Ti_2 = 0.2364$.

La planta A_1 tendrá como entrada la salida del actuador₁ y como salida la señal y_1 , que es la sumatoria de la salida de planta en sí más la señal y_3 (Figura I.20). Hacer doble clic sobre el bloque retardo e ingresar el valor 0.043.

La planta A_2 tendrá como entrada la salida del actuador₂ y como salida la señal y_2 , que es la sumatoria de la salida de planta en sí más la señal y_4 (Figura I.20). Hacer doble clic sobre el bloque retardo e ingresar el valor 0.017.

La planta A_3 tendrá como entrada la salida del actuador₂ y como salida la señal y_3 (Figura I.20). Hacer doble clic sobre el bloque retardo e ingresar el valor 0.017.

La planta A_4 tendrá como entrada la salida del actuador₁ y como salida la señal y_4 (Figura I.20). Hacer doble clic sobre el bloque retardo e ingresar el valor 0.153.

Para poder ver el comportamiento del sistema se tiene el gráfico r_1, y_1 , r_2, y_2 ; u_1, u_2 (ver Figura I.21). El usuario debe comprobar que las entradas a dicho gráfico sean las correctas, en este caso la referencia r_1 , el valor recibido por el sensor₁: y_1 , la referencia r_2 , el valor recibido por el sensor₂: y_2 , la señal actuada sobre la planta1 (salida del controlador_{1 OPC}) u_1 y la señal actuada sobre la planta2 (salida del controlador_{2 OPC}) u_2 .

Una vez verificado todo lo anterior, el usuario debe cerrar “plataforma.mdl”.

En línea de usuario introduce el siguiente comando:

```
>> pantalla <ENTER>
```

Aparecerá entonces la pantalla principal, en donde el usuario deberá marcar los elementos con los que se trabajará en este caso (ver Figura I.22)

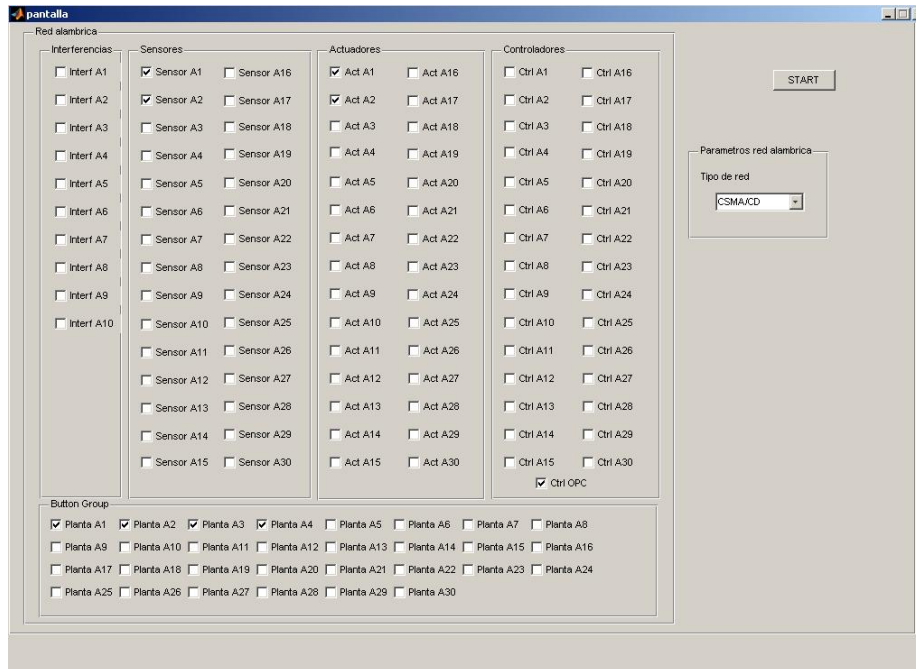


Figura I.22: Elección de elementos con los que se armará el sistema en estudio

Se selecciona sensor₁, sensor₂, el controlador_{OPC}, la planta A₁, la planta A₂, planta A₃, planta A₄, el actuador₁ y el actuador₂. Al seleccionar planta A₁ aparece la subpantalla mostrada en la Figura I.23, al seleccionar planta A₂ aparece la subpantalla mostrada en Figura I.24, al seleccionar planta A₃ aparece la subpantalla mostrada en Figura I.25 y al seleccionar planta A₄ aparece la subpantalla mostrada en Figura I.26.

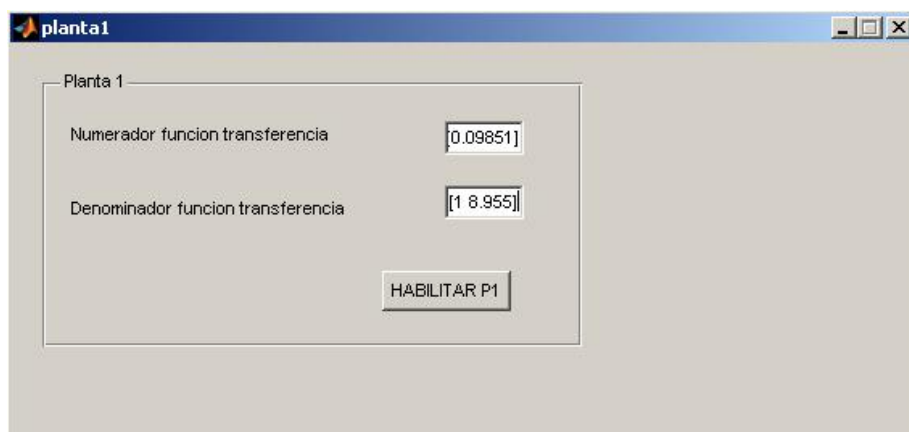


Figura I.23: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A₁ (no ingresar e-0.043s porque está representado por un bloque retardo en “plataforma.mdl”)

The screenshot shows a window titled 'planta2'. Inside, there is a section labeled 'Planta 2'. It contains two input fields: 'Numerador funcion transferencia' with the value '0.1380 0' and 'Denominador funcion transferencia' with the value '1 18.615'. Below these fields is a button labeled 'HABILITAR P2'.

Figura I.24: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A2 (no ingresar e-0.017s porque está representado por un bloque retardo en “plataforma.mdl”)

The screenshot shows a window titled 'planta3'. Inside, there is a section labeled 'Planta 3'. It contains two input fields: 'Numerador funcion transferencia' with the value '-0.05408' and 'Denominador funcion transferencia' with the value '1 6.6225'. Below these fields is a button labeled 'HABILITAR P3'.

Figura I.25: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A3 (no ingresar e-0.017s porque está representado por un bloque retardo en “plataforma.mdl”)

The screenshot shows a window titled 'planta4'. Inside, there is a section labeled 'Planta 4'. It contains two input fields: 'Numerador funcion transferencia' with the value '-0.04258' and 'Denominador funcion transferencia' with the value '1 7.362'. Below these fields is a button labeled 'HABILITAR P4'.

Figura I.26: Se debe ingresar numerador y denominador de la función de transferencia que representa la planta A4 (no ingresar e-0.153s porque está representado por un bloque retardo en “plataforma.mdl”)

Una vez ingresados los vectores que representan numerador y denominador de cada función de transferencia, se clikea sobre el icono “Habilitar P1”, “Habilitar P2”, “Habilitar P3” o “Habilitar P4”, según sea el caso. Las subpantallas se cierran. Sobre la pantalla principal (Figura I.22) se elige el tipo de red cableada a utilizar y se hace clic sobre el ícono *Start*. Así da comienzo a la simulación.

Aparecen ahora otras subpantallas para que el usuario ingrese valores deseados a los parámetros que definen el desempeño de la red. Una vez ingresados se debe hacer clic sobre el botón “Habilitar” y esos valores serán ingresados durante la corrida.

Cuando la simulación finaliza, se puede hacer doble clic sobre el gráfico correspondiente (ver Figura I.21) para ver la evolución del sistema. Con el botón derecho del mouse se hace clic sobre cada uno de los gráficos (r_1, y_1 , r_2, y_2 ; y u_1, u_2) y se elige la opción *Autoescale*, así se podrá ver el gráfico completo.

Referencias

[1] Liu and Layland **“Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”** Journal of ACM 20(1): pp. 46-61, (1973).

[2] Sha, L., Abdelzaher, T., Årzén, K-E., Cervin, A., Baker, T., Burns, A., Caccamo, M., Lehoczky, J. and Mok, A. K. **“Real Time Scheduling Theory: A Historical Perspective”**. *Real-Time Systems* 28(2-3):101-155, (2004).

[3] Vaccaro, R. J., **“Digital Control: A State-Space Approach”**, McGraw-Hill Series in Electrical and Computer Engineering, (1995).

[4] Zimmermann H., **“OS1 reference model. The IS0 model of architecture for open system interconnection”**, IEEE Trans. COM-28 n04, April 80, pp 425-432, (1980).

- [5] Henriksson Dan, Cervin Anton, Årzén Karl-Erik, “**Real-time Control System Simulation with MATLAB/Simulink**”, *Nordic MATLAB Conference*, Copenhagen, Denmark, (2003)
- [6] G. Bollella, B. Brosgol, P. Dibble, S. Furr, J. Gosling, D. Hardin, and M. Turnbull. “**The Real-Time Specification for Java**”, Addison-Wesley, (2000).
- [7] Mazzone Virginia, “**Estructuras Prácticas de Control SISO**”, Universidad Nacional de Quilmes, (2002).
- [8] Ohlin Martin, Henriksson Dan, Cervin Anton, “**TRUETIME 1.5—Reference Manual**”, Manual, Department of Automatic Control, Lund University, Sweden, (2007)
- [9] Åström, K. J. and T. Hägglund, “**PID Controllers: Theory, Design, and Tuning**”, Instrument Society of America, Research Triangle Park, North Carolina, (1995).
- [10] Mazzone Virginia, “**Control PID clásico**”, Universidad Nacional de Quilmes, (2009).
- [11] Zamarreño Jesús M., Cristea Smaranda, Rueda Almudena, Aref Rachid, “**Módulos OPC para control y supervisión de entornos industriales**”, Dpto. de Ingeniería de Sistemas y Automática, Facultad de Ciencias, Universidad de Valladolid, (2005)
- [12] Chvostek T., Foltin M., Farka Ľ., “**The adaptative PID Controller using OPC Toolbox**”, Syprin s. r. o., Žehrianska 10, 851 05 Bratislava, Slovak Republic, Mezinárodní konference Technical Computing Prague (2006)
- [13] MATLAB OPC toolbox help, www.mathworks.com

- [14] Introduction to OPC–Tutorial, www.matrikonopc.com
- [15] OPC FOUNDATION, Specification of OPC, www.opcfoundation.org
- [16] Lozoya Camilo, Martí Pau, Velasco Manel and Fuertes Josep M., **“Analysis and design of networked control loops with synchronization at the actuation instants”**, *Proceeding IECON2008 - 34th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, p.1-1, (2008)
- [17] Herbert Ecker, Misikir Armide, **“Combining the Good Things from Vehicle Networks and High-Performance Networks”**, Master’s Thesis in Electrical Engineering, Halmstad University, Technical report, IDE0705, (2007)
- [18] Xiangdong Xu and Feng Du, **“Simulation of Cascaded Networked Control Systems”**, School of Electrical Engineering, Southwest Jiaotong University, Chengdu 610031, China, (2006)
- [19] Guàrdia Josep, Martí Pau, Velasco Manel and Castañé Rosa, **“Enabling Feedback Scheduling in TrueTime”**, UPC, Research report ESII-RR-06-05, (2006)
- [20] Castañé R., Martí P., Velasco M., Guàrdia J., Yépez J., Ayza J., Villà R., Fuertes J.M., **“Asignación de periodos a tareas de control en función de la predicción de la dinámica de cada planta”**, IX Jornadas de Tiempo Real, Valladolid, Spain, (2006)