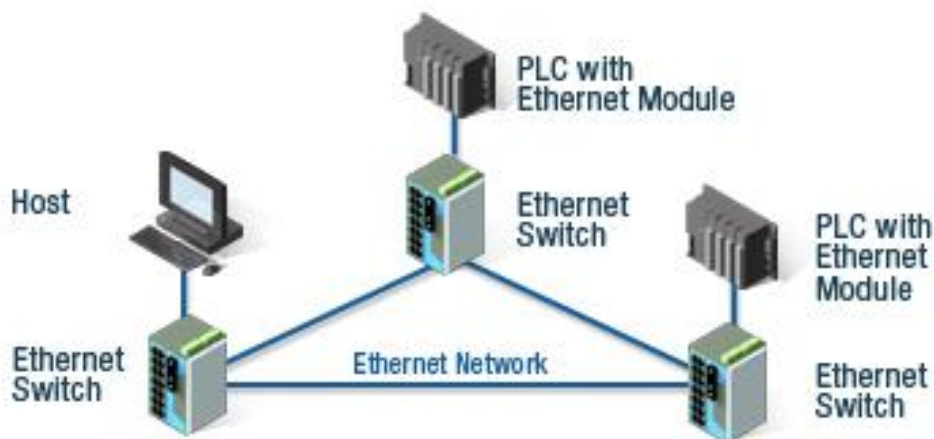


Diseño de un simulador virtual para testeo de algoritmos de control en hardware real



Josu Anta Larrea

07/07/2014

Índice

1.	Introducción	3
1.1	Descripción del sistema	4
1.2	OPC	5
1.3	Omron CX-Server OPC	6
1.4	Matlab/Simulink OPC Toolbox	6
2.	Estado del arte	8
2.1	Revisión histórica	8
2.2	Términos básicos	9
2.3	Sistemas en lazo cerrado	9
2.4	Sistemas en lazo abierto	9
2.5	Comparación entre ambos sistemas	10
2.6	Modelado matemático de sistemas	10
2.7	Sistemas lineales	11
2.8	Mathworks	11
2.9	Omron	12
3.	Trabajo realizado	13
3.1	Búsqueda de alternativas para interconectar un autómata con aplicaciones externas	13
3.2	Soluciones planteadas por Omron	13
3.2.1	CX-Server Lite	14
3.2.2	CX-Server OPC	14
3.2.3	CX-Compolet	15
3.3	Configuración de una red local	15
3.4	Configurar tablas de entrada y salida de un PLC	21
3.5	Real-Time Windows Target	22
3.5.1	Normal mode simulation	23
3.5.2	External mode simulation	24
3.6	OPC Foundation Core Components	25
3.6.1	OPC Configuration Block	26
3.6.2	OPC Read Block	27
3.6.3	OPC Write Block	29

3.7	Combinación de las herramientas Real-Time Windows Target y OPC Toolbox	30
3.8	Configuración DCOM para habilitar la comunicación entre aplicaciones que se ejecutan en distintas máquinas	31
3.8.1	“Configure DCOM to Use Named User Security”	32
3.8.2	“Configure DCOM to Use No Security”	33
3.9	SCADA	34
4.	Resultados	37
4.1	Simulación teórica	38
4.2	“Hardware-in-the-loop simulation”	39
4.2.1	Implementación del algoritmo de control en el autómata	39
4.2.2	Ralentización del simulador para la correcta ejecución en tiempo real	40
4.2.3	Simulaciones a distintas velocidades	41
5.	Conclusiones	47
5.1	Comparación entre los resultados de las distintas simulaciones ralentizadas realizadas	47
5.2	Comparación de los resultados obtenidos con el modelo teórico	49
6.	Bibliografía	51
	Anexo I: Código fuente del algoritmo de control	52

1. Introducción

Nuevas tendencias llevan a nuevas metas, es un constante no parar de encontrar nuevas formas de optimizar y mejorar lo ya existente. En este sentido, la simulación ha traído un importante paso adelante en la manera de trabajar y de anticiparse a lo que pueda suceder, desde la simulación de un examen que le hace la maestra a su alumno para un examen, pasando por la producción de textiles, alimentos, juguetes, construcción de infraestructuras por medio de maquetas, hasta el entrenamiento virtual de los pilotos de combate.

La simulación de procesos es una de las más grandes herramientas de la ingeniería industrial, la cual, se utiliza para representar un proceso mediante otro que lo hace mucho más simple y entendible. Esta simulación es en algunos casos casi indispensable, bien sea dentro de un proceso industrial, bajo unas determinadas condiciones o con unas características u otras. En otros casos no lo es tanto, pero sin este procedimiento se hace más complicado. La simulación permite analizar con detenimiento las características de un proceso para después poder mejorar su actuación⁽¹⁾.

Es precisamente este último detalle el que ha sido el principal impulsor de este proyecto, cuyo objetivo final es el testeo de un controlador. Un controlador es diseñado para realizar una secuencia de actividades de manera que todo el proceso industrial esté automatizado y controlado. Los propios softwares de programación de controladores traen consigo programas de depuración y corrección de errores pero, ¿cómo sabemos si dicho controlador funcionará correctamente en tiempo real? ¿Cómo sabemos que no incurrirá en retrasos debido al tiempo material que necesitamos en realizar una actividad determinada? Es entonces donde surge la necesidad de testear dicho algoritmo de control en tiempo real antes de implementarlo en la planta industrial. Gracias a ello conseguimos obtener resultados certeros a un coste muy bajo. Para ello es para lo que vamos a utilizar un simulador, con el que nos conectaremos con el controlador y simularemos nuestro objetivo en tiempo real. En nuestro caso, el simulador simulará un sistema en lazo cerrado obtenido del libro *Ingeniería de Control Moderna*⁽²⁾, pero bien podría simular una planta industrial o un juego virtual. Esta técnica es conocida como “*hardware-in-the-loop simulation*”.

El resultado de la realización de este proyecto ha sido obtener un software capaz de comunicar de manera eficiente y en tiempo real la simulación de un modelo de cualquier sistema (el sistema descrito en el apartado Descripción del sistema en nuestro caso) con algún programa SCADA y un controlador, con el fin de poder realizar el control de este modelo de la misma manera que se podría controlar un sistema de producción real. El tipo de conexión utilizada ha sido una comunicación vía Ethernet a través de un servidor OPC.

Las herramientas utilizadas para la realización de este proyecto han sido las siguientes⁽³⁾:

- Matlab: software matemático muy potente que gracias a su herramienta Simulink nos permite realizar simulaciones muy certeras de cualquier modelo

matemático. Además es ampliamente conocido y utilizado en entornos universitarios y empresariales.

- Simulink: herramienta de Matlab que ofrece un interfaz gráfico fácil a través de diagramas de bloques, que nos permite construir modelos de manera rápida e intuitiva.
- Autómata CJ1M-CPU11 de Omron: autómata programable donde programaremos nuestro algoritmo de control para luego testarlo en el simulador. Se trata de una versión de autómata bastante potente ya que cuenta con puerto de Ethernet para conectarlo a cualquier red.
- Omron OPC Server: servidor de dominio público con el que es posible conectarse a través de un cliente OPC para adquirir datos, en este caso, del autómata.

En definitiva, el objetivo ha sido realizar la comunicación vía Ethernet entre autómata, servidor OPC y cliente OPC de Matlab/Simulink para obtener valores del modelo e interactuar sobre ellos.

1.1 Descripción del sistema

Se ha escogido un sistema sencillo para testear nuestro controlador. Ha sido preferible comenzar por este tipo de sistema ya que sistemas que no sean lineales siempre llevan a mayores problemas debido a la necesidad de linealizarlos. Teniendo en cuenta que el proyecto incluye una parte de establecer la comunicación entre el simulador y el autómata vía Ethernet, se ha optado por un sistema sencillo que no diera problemas de computación. Un sistema más complejo hubiera requerido mayor conocimiento de los métodos de linealización así como mayor complejidad y dedicación a la parte programación. Como se quería abarcar distintas herramientas como los SCADA, servidores OPC, simulaciones en Simulink... se ha optado por un sistema sencillo. Nuestro sistema escogido ha sido un ejemplo que aparece en el libro *Ingeniería de Control Moderna*⁽²⁾ de Katsuhiko Ogata (página 660). En dicho ejemplo se nos propone diseñar un controlador que cumpla unos determinados requisitos: constante de error estático de velocidad 20 seg^{-1} , margen de fase 50° y margen de ganancia superior a 10dB.

Tal y como se muestra en la resolución del ejemplo el sistema en lazo cerrado se representaría de la siguiente manera:

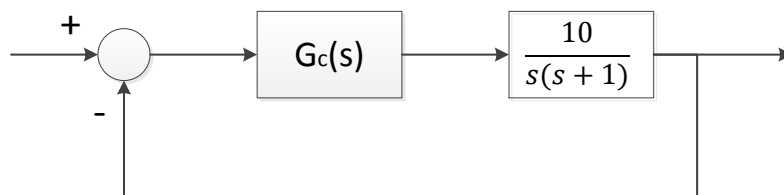


Figura 1 Sistema en lazo cerrado

Donde la función de transferencia del compensador es la siguiente:

$$G_c(s) = 9.5238 \frac{s + 2.9789}{s + 14.1842} = 2 \frac{0.3357s + 1}{0.07050s + 1}$$

Con estos datos se procede a implementar dicho modelo en Simulink y programar el compensador en el autómata. Se han realizado dos modelos, uno íntegramente en Simulink en lazo cerrado para obtener los datos teóricos (Simulación teórica) y otro con *“hardware-in-the-loop”* donde el lazo de control viene implementado en el autómata (*“Hardware-in-the-loop simulation”*). Todos los detalles vienen incluidos en el apartado Resultados.

1.2 OPC

OPC (Object Linking and Embedding (OLE) for Process Control) es un estándar para el intercambio seguro y fiable de los datos en el espacio de la automatización industrial y de otras industrias. Es independiente de la plataforma que se esté utilizando y asegura el flujo continuo de información entre dispositivos de múltiples proveedores, siempre bajo la estructura cliente/servidor OPC. OPC Foundation es el responsable del desarrollo y mantenimiento de esta norma. El estándar OPC es una serie de especificaciones desarrolladas por proveedores de la industria, los usuarios finales y desarrolladores de software. Estas especificaciones definen la interfaz entre clientes y servidores, así como la interfaz entre servidores, incluyendo el acceso a datos en tiempo real, monitorización de alarmas y eventos, acceso a los datos históricos y otras aplicaciones.

Cuando la norma fue lanzada por primera vez en 1996, su propósito era convertir protocolos específicos de comunicación en PLCs (Modbus, Profibus...) en una interfaz estandarizada que permitiera que sistemas como HMI/SCADA pudieran interactuar con un intermediario que convirtiera peticiones de lectura/escritura en las solicitudes específicas del dispositivo y viceversa.

Inicialmente, el estándar OPC se limitaba sólo al sistema operativo Windows. Como tal, las siglas OPC nacieron de OLE (Object Linking and Embedding) para control de procesos. Estas especificaciones, conocidos como OPC clásico, han sido adoptadas de manera generalizada en múltiples industrias como la fabricación, automatización de edificios y energías renovables entre otros. Con la introducción de las arquitecturas orientadas a servicios en los sistemas de fabricación llegaron nuevos retos en seguridad y modelado de datos. La Fundación OPC desarrolló las especificaciones OPC UA para tratar estas necesidades y al mismo tiempo proporcionar una tecnología rica en características de arquitectura de plataformas abiertas, escalables y extensibles⁽⁴⁾.

De esta manera en este proyecto se han utilizado dichos estándares para comunicar hardwares y softwares de distintos fabricantes consiguiendo así una integración muy

eficiente. Tal y como se ha mencionado anteriormente, básicamente, es necesario contar con un servidor OPC y distintos clientes para el desarrollo de dichas aplicaciones. En nuestro caso se ha utilizado un servidor OPC de Omron, especialmente diseñado para la comunicación con autómatas Omron, y un cliente OPC de Matlab/Simulink a través de su herramienta OPC Toolbox.

1.3 Omron CX-Server OPC

La versión comercial de CX-Server OPC incluye el CX-Server OPC Server que permite que los datos recogidos en un PLC de Omron puedan ser accesibles en cualquier cliente OPC v1.0a o v2.0 Data Access (DA). También incluye componentes gráficos ActiveX para crear aplicaciones MMI (Multimodal Interaction Activity) sencillas en Microsoft Excel, Visual Basic, Visual C++ o Visual Studio.

Al ejecutar el programa aparecerá un icono con las letras "CX" en la barra de herramientas en la parte inferior derecha de la pantalla. Al hacer clic en este icono con el botón derecho del ratón muestra un menú desde el que se puede seleccionar un archivo de proyecto de CX-Server (*.cdm). Este archivo de proyecto puede ser nuevo o haber sido creado previamente por una aplicación como CX-Supervisor, CX-Programmer o CX-Server Lite. Si estamos ante un proyecto nuevo deberemos asignar al proyecto los distintos PLCs con los que queremos que el proyecto conste (nombre del PLC, tipo de conexión...), así como los distintos puntos o direcciones de memoria con los que necesitamos trabajar en el proyecto (su ubicación de memoria, tipo de dato...). Una vez seleccionados, los datos de este archivo estarán disponibles para su uso por todos los programas clientes v2.0 OPC (por ejemplo, aplicaciones SCADA, OPC Toolbox de Matlab...). Cuando el servidor OPC CX-Server se cierra recuerda qué archivo se ha cargado previamente, y este archivo se vuelve a cargar automáticamente cada vez que se reinicie. El nombre del servidor OPC para el Servidor OPC CX-Server es *"OMRON.OpenDataServer.1"* (utilizado para la configuración de los clientes OPC)⁽⁵⁾.

1.4 Matlab/Simulink OPC Toolbox

La herramienta OPC de Matlab implementa un enfoque jerárquico orientado a objeto para comunicarse con los servidores OPC utilizando tanto el OPC Data Access como el estándar de acceso a datos históricos. Utilizando las distintas funciones que proporciona dicha herramienta es posible crear clientes OPC para el acceso a distintos datos del servidor OPC a través de Matlab. Para establecer la comunicación con cualquier servidor OPC es necesario incluir en nuestro diagrama de bloques (si estamos trabajando en Simulink) el bloque de configuración del servidor OPC. En las propiedades de este bloque podremos seleccionar el servidor OPC con el que queramos trabajar, así como otros parámetros que deseemos. En nuestro caso es suficiente con seleccionar el servidor y activar la opción de pseudo real-time, ya que queremos simularlo en tiempo real.

Una vez hayamos configurado el servidor sólo nos queda seleccionar los puntos o direcciones de memoria que queremos incluir en nuestra simulación. Tal y como se ha mencionado en el apartado anterior (Omron CX-Server OPC), al ejecutar el servidor le asignamos un proyecto que contiene los datos del autómatas, así como las direcciones de memoria sobre las que queremos trabajar. Una vez conectados al servidor, tendremos acceso a esas direcciones de memoria y podremos seleccionar las que deseemos para trabajar con ellas durante la simulación. Trabajar sobre estas direcciones de memoria es relativamente sencillo, sólo tenemos que incluir los bloques de escritura y lectura que tenemos en la OPC Toolbox. En esos bloques seleccionaremos las direcciones de memoria y de esta manera estarán listas para ser utilizadas durante la simulación⁽⁶⁾.

2. Estado del arte

El control automático ha desempeñado un papel vital en el avance de la ingeniería y la ciencia. El control automático se ha convertido en una parte importante e integral de los procesos modernos industriales y de fabricación. Por ejemplo, el control numérico se ha convertido en pieza esencial de un gran número de máquinas-herramienta de las industrias de manufactura. Pero también es crítico en el control de presión, temperatura, humedad, viscosidad y flujo en las industrias de proceso. Se trata de un campo que la mayoría de ingenieros y científicos deberían controlar en cierta medida ya que los avances en la teoría y la práctica del control automático proporcionan los medios para conseguir un comportamiento óptimo de los sistemas dinámicos o mejorar la productividad por ejemplo⁽²⁾.

2.1 Revisión histórica

El primer trabajo significativo en control automático fue un regulador de velocidad centrífugo de James Watt para el control de la velocidad de una máquina de vapor, en el siglo XVIII. Minorsky, Hazen y Nyquist, entre muchos otros, aportaron trabajos importantes en las etapas iniciales del desarrollo de la teoría de control. En 1922, Minorsky trabajó en controladores automáticos para el guiado de embarcaciones y mostró que la estabilidad puede determinarse a partir de las ecuaciones diferenciales que describen el sistema. En 1932, Nyquist diseñó un procedimiento relativamente simple para determinar la estabilidad de sistemas en lazo cerrado a partir de la respuesta en lazo abierto a entradas sinusoidales en estado estacionario.

Durante la década de los cuarenta, los métodos de la respuesta en frecuencia (diagramas de Bode por ejemplo) hicieron posible que los ingenieros diseñaran sistemas de control lineales en lazo cerrado que cumplieran los requisitos de comportamiento. Después vendría el desarrollo del lugar de las raíces que junto con los métodos de la respuesta en frecuencia forman el núcleo de la teoría de control. Esta teoría nos conduce a sistemas estables que cumplen en cierto modo las restricciones introducidas, pero lejos de ser el óptimo. Por ello, a partir de la década de los cincuenta se puso énfasis en buscar el sistema óptimo para nuestros requerimientos.

En las plantas modernas se han llegado a introducir sistemas con múltiples entradas y salidas, lejos de la teoría clásica de control, que sólo admitía una entrada y una salida. Todo ello implica la introducción de gran cantidad de ecuaciones que no fue posible resolver hasta la década de los sesenta, donde surge la teoría de control moderna. Fue gracias a la creación de computadores digitales cuando se empezaron a analizar sistemas complejos en el dominio del tiempo.

Ahora que los computadores digitales se han hecho más baratos y compactos, se usan como parte integral de los sistemas de control. Las aplicaciones recientes de la teoría de

control moderna incluyen sistemas fuera del ámbito de las ingenierías, como sistemas biológicos, biomédicos, económicos o socioeconómicos.

2.2 Términos básicos

Antes de profundizar en aspectos técnicos es conveniente resaltar ciertos términos básicos utilizados frecuentemente en el control moderno.

En primer lugar es muy importante diferenciar los términos *variable controlada* y *variable manipulada*. La *variable controlada* es el parámetro que se mide y se controla. Normalmente corresponde con la salida de nuestro sistema. Cuando hablamos de controlar un sistema nos referimos a medir el valor de la variable controlada actuando sobre la variable manipulada. La *variable manipulada*, por tanto, será el parámetro que hay que modificar para obtener el valor deseado en la variable controlada.

Cuando se habla de una *planta* nos podemos referir a una parte del equipo así como a un conjunto de elementos que forman una máquina. Los *procesos*, en cambio, son un conjunto de operaciones que aportan cambios y que se suceden de forma relativa para obtener un resultado determinado.

Al utilizar el término *sistema* nos referimos a una combinación de componentes que actúan juntos y realizan un objetivo determinado. Estos sistemas pueden verse afectados y actuar de distinta manera si están sometidos a *perturbaciones* o *realimentaciones*. Las *perturbaciones* son señales que afectan negativamente (normalmente) a la salida del sistema. Pueden ser tanto externas (fuera del sistema) como internas (dentro del sistema). El *control realimentado*, por último, se define como la operación que tiende a reducir la diferencia entre la salida del sistema y una entrada definida como referencia. Comparando dichos valores y utilizando la diferencia como medio de control se consigue el objetivo de controlar el sistema.

2.3 Sistemas en lazo cerrado

Los sistemas realimentados se denominan *sistemas de control en lazo cerrado*. En estos sistemas, el controlador es alimentado por una señal que se denomina error con el objetivo de reducir dicho error y llevar a la salida un valor más próximo a lo requerido. Ese error es la diferencia entre la señal de entrada y la realimentación (puede ser la señal de salida o una función de la señal de salida).

2.4 Sistemas en lazo abierto

Son sistemas donde la salida no está comunicada con el controlador, por tanto no tiene ningún efecto sobre él. Es decir, no existe tal comparación entre la referencia y el error. De esta manera, a cada entrada le corresponde una salida determinada y el resultado dependerá en gran medida del calibrado del sistema. Estos sistemas son

incapaces de actuar ante una perturbación y, como consecuencia, en dichos casos no consiguen realizar la tarea requerida. Estos sistemas sólo se usan en caso de que conocer la relación entre la entrada y la salida.

2.5 Comparación entre ambos sistemas

La mayor ventaja de los sistemas en lazo cerrado es que los sistemas realimentados son casi insensibles a las perturbaciones. De esta manera es posible utilizar componentes baratos y poco precisos para obtener un control aceptable del sistema. Desde el punto de vista de la estabilidad, los sistemas en lazo abierto son mucho más sencillos, ya que la estabilidad del problema no acarrea ningún problema importante. En el lazo cerrado, en cambio, la estabilidad puede conducir a oscilaciones constante o variables. Los sistemas en lazo cerrado son muy útiles cuando hay variaciones impredecibles en el sistema, pero son a su vez más caros y consumen mayor potencia debido al uso de mayor número de componentes.

2.6 Modelado matemático de sistemas

Todo estudio de un sistema de control tiene un trabajo previo de modelización y análisis de las características de dicho sistema. Un modelo matemático está constituido por un conjunto de ecuaciones que representan el sistema con la máxima precisión posible. Cabe mencionar que no existe un único modelo matemático para un determinado sistema, un sistema puede representarse de muchas maneras dependiendo del enfoque que le demos. Muchos sistemas (mecánicos, eléctricos, térmicos...) se describen por medio de ecuaciones diferenciales. Estas ecuaciones se obtienen a partir de leyes físicas que dominan el sistema, como las leyes de Newton en mecánica o las leyes de Kirchhoff en sistemas eléctricos.

Los modelos matemáticos pueden tener formas de lo más diversas dependiendo del sistema que sea, así como en función de las circunstancias a las que esté sometido dicho sistema. Unas veces conviene utilizar un modelo matemático, mientras otras es preferible otro. En análisis de respuesta transitoria de sistemas lineales por ejemplo, es preferible la representación mediante la función de transferencia. Una vez definido el modelo matemático apropiado, se utilizan diversos métodos analíticos y computacionales para su estudio y resolución.

Al desarrollar un modelo matemático es importante que haya un balance entre la simplicidad y precisión del mismo. Simplificaciones en nuestro modelo implican el ignorar ciertas propiedades físicas, esto es, al trabajar con un problema no lineal muchas veces habrá que ignorar o linealizar las posibles no linealidades que nos encontremos en el problema. Si los parámetros donde hemos realizado suposiciones tienen un efecto

despreciable sobre el resultado final, nuestro modelo se ajustará a los resultados obtenidos en el estudio experimental. En caso contrario tendremos un problema.

Siempre es conveniente empezar a partir de un modelo matemático simplificado. Una vez obtengamos una solución aceptable iremos complicando dicho modelo. Es muy importante tener en cuenta que unos parámetros que apenas afectan a frecuencias bajas pueden ser determinantes a altas frecuencias, por lo que es conveniente probar nuestro modelo matemático simplificado tanto en bajas como altas frecuencias.

2.7 Sistemas lineales

Son aquellos sistemas que respetan el principio de superposición. Este principio determina que la respuesta de un sistema ante la entrada de dos funciones distintas es la suma de las respuestas de cada función. De esta manera es bastante sencillo calcular problemas complejos mediante la resolución de los problemas simples en los que se puedan descomponer. Una ecuación diferencial es lineal si sus coeficientes son constantes o son función sólo de la variable independiente. Los sistemas formados por parámetros invariantes con el tiempo se representan mediante ecuaciones diferenciales de coeficientes constantes y se denominan sistemas lineales de coeficientes constantes. Si los coeficientes, en cambio, varían con el tiempo, estaremos ante un sistema lineal variante en el tiempo.

2.8 Mathworks

MathWorks es una empresa líder en desarrollo de software de cálculo matemático. Ingenieros y científicos de todo el mundo confían en sus productos para acelerar el ritmo de sus investigaciones, innovación y desarrollo.

MATLAB, el lenguaje del cálculo técnico, es un entorno de programación para el desarrollo de algoritmos, el análisis y la visualización de datos y el cálculo numérico. Simulink es un entorno gráfico para la simulación y el diseño basado en modelos de sistemas dinámicos e integrados multidominio. La empresa produce casi 100 productos adicionales para tareas especializadas, como análisis de datos y procesamiento de imágenes.

Los productos de MathWorks se utilizan en sectores como automoción, aeroespacial, comunicaciones, electrónica y automatización industrial, como herramientas básicas de investigación y desarrollo. También se utilizan para el modelado y la simulación en campos cada vez más técnicos, como los servicios financieros y la biotecnología. El software de MathWorks permite diseñar y desarrollar una gran variedad de productos avanzados como sistemas de automoción, de aviónica y control de vuelo aeroespacial, de telecomunicaciones y otros equipos electrónicos, maquinaria industrial y dispositivos médicos, entre otros. Las soluciones de MathWorks se utilizan en más de 5.000 facultades

y universidades de todo el mundo para la enseñanza y la investigación en una amplia gama de disciplinas técnicas⁽⁷⁾.

2.9 Omron

La Corporación Omron es una compañía japonesa, uno de los líderes mundiales en el campo de la automatización. Establecida en 1933 y dirigida por su presidente Yoshihito Yamada, Omron tiene más de 37.000 empleados en más de 36 países trabajando para proporcionar productos y servicios en varios campos, incluyendo automatización industrial, industria de componentes electrónicos y atención sanitaria. En su trayectoria, Omron ha dejado una serie de innovaciones tecnológicas que han revolucionado nuestras vidas, formando parte de las primeras innovaciones del mundo: conectores de proximidad sin contacto, señales de tráfico automáticas, máquinas de venta de billetes y un completo sistema de automatización en estaciones de trenes, así como un equipamiento automatizado en diagnósticos de células cancerígenas. Omron realiza contribuciones significativas en una extensa variedad de campos tales como la automatización industrial, aplicaciones caseras y equipamientos de oficinas, automóviles, sistemas financieros y sociales, y cuidados en la salud.

Omron aplica sus principales competencias en tecnologías de detección y control a través de diversas operaciones a escala mundial. En Automatización Industrial Omron ofrece el soporte necesario para sus innovaciones en el saber hacer, gracias a los componentes de control de alta calidad unidos a las tecnologías de detección y de control. En Europa, Omron está representado por tres principales compañías:

- Industrial Automation Business.
- Electronic Components Business.
- Health Care Business.

Estas empresas, que trabajan con una amplia red de distribuidores locales y proveedores de soluciones, cubren los principales países de Europa, la antigua Unión Soviética, Oriente Medio y África⁽⁸⁾.

3. Trabajo realizado

1. Búsqueda de alternativas para interconectar un autómatas con aplicaciones externas
2. Soluciones planteadas por Omron
3. Configuración de una red local
4. Configurar tablas de entrada y salida de un PLC
5. Real-Time Windows Target
6. OPC Foundation Core Components
7. Combinación de las herramientas Real-Time Windows Target y OPC Toolbox
8. Configuración DCOM para habilitar la comunicación entre aplicaciones que se ejecutan en distintas máquinas

3.1 Búsqueda de alternativas para interconectar un autómatas con aplicaciones externas

Antes de abordar cualquier tipo de simulación lo más importante es establecer una comunicación entre el autómatas y una aplicación externa, bien sea Microsoft Excel, Visual Basic o Matlab. Para ello se ha realizado un trabajo de búsqueda en la red con muy poco éxito, ya que todas las opciones que nos planteaban eran a través de programación dirigida a objetos muy avanzada que se nos escapaba de las manos. Estas opciones consistían en programar por nuestra cuenta todas las comunicaciones y los protocolos necesarios.

Teniendo en cuenta estas dificultades a nivel de programación se planteó una opción interesante que consistía en los servidores y clientes OPC. Era una manera de evitar la programación pura y dura. Se trata de utilizar unos softwares comerciales que están programados bajo un protocolo estándar (OPC) preparado específicamente para la comunicación entre aplicaciones de distintos proveedores.

Esta opción resultaba muy interesante ya que no era necesario tener grandes conocimientos de programación para realizar la comunicación. Pero estas vías de comunicación tienen un problema, es necesario comprar distintas licencias, tanto para los clientes OPC como para los servidores OPC. Careciendo de dichas licencias y softwares se decidió ponerse en contacto con *Omron Electronics* para plantearles nuestro problema y recoger las soluciones propuestas por su servicio técnico. La razón de acudir a *Omron Electronics* es bien sencilla, íbamos a trabajar con un autómatas de la marca Omron y nadie mejor que ellos para darnos soluciones al respecto.

3.2 Soluciones planteadas por Omron

Tras una reunión con los responsables del servicio técnico de Omron las propuestas o soluciones planteadas fueron tres:

- Utilización del servidor CX-Server Lite.
- Utilización del servidor CX-Server OPC.
- Utilización del software CX-Compolet.

3.2.1 CX-Server Lite

CX-Server Lite permite que los datos de un PLC Omron sean fácilmente accesibles desde Microsoft Excel, Visual Basic, Visual C++ o Visual Studio. CX-Server Lite incluye a su vez un paquete llamado CX-Server Runtime que proporciona una comunicación completa con los PLCs de las series C, CV y CS1. Esto permite el uso compartido de definiciones de símbolos y datos con otros productos de Omron como CX-Programmer y CX-Supervisor.

CX-Servidor Lite también incluye controles y componentes gráficos ActiveX que se pueden arrastrar y soltar en un documento de Microsoft Excel, lo que permite crear aplicaciones MMI (Multimodal Interaction Activity) sencillas. Por tanto se trata de una herramienta muy sencilla de utilizar y a su vez muy visual, ya que incluye controles gráficos para controlar las direcciones de memoria del autómata.

A la hora de configurar los programas Microsoft Excel, Visual Basic, Visual C++ o Visual Studio para la utilización de dicho servidor, es necesario crear previamente un proyecto de CX-Server (*.cdm). Este archivo de proyecto puede ser nuevo o haber sido creado previamente. Si estamos ante un proyecto nuevo deberemos asignar al proyecto los distintos PLCs con los que queremos que el proyecto conste (nombre del PLC, tipo de conexión...), así como los distintos puntos o direcciones de memoria con los que necesitamos trabajar en el proyecto (su ubicación en la memoria, tipo de dato...)⁽⁹⁾.

3.2.2 CX-Server OPC

La versión comercial de CX-Server OPC incluye el CX-Server OPC Server que permite que los datos recogidos en un PLC de Omron puedan ser accesibles desde cualquier cliente OPC v1.0a o v2.0 Data Access (DA). También incluye componentes gráficos ActiveX para crear aplicaciones MMI (Multimodal Interaction Activity) sencillas en Microsoft Excel, Visual Basic, Visual C++ o Visual Studio.

Al ejecutar el programa aparecerá un icono con las letras "CX" en la barra de herramientas en la parte inferior derecha de la pantalla. Al hacer clic en este icono con el botón derecho del ratón muestra un menú desde el que se puede seleccionar un archivo de proyecto de CX-Server (*.cdm). Este archivo de proyecto puede ser nuevo o haber sido creado previamente por una aplicación como CX-Supervisor, CX-Programmer o CX-Server Lite. Si estamos ante un proyecto nuevo deberemos asignar al proyecto los distintos PLCs con los que queremos que el proyecto conste (nombre del PLC, tipo de conexión...), así como los distintos puntos o direcciones de memoria con los que necesitamos trabajar en el proyecto (su ubicación en la memoria, tipo de dato...). Una vez seleccionados, los datos de este archivo estarán disponibles para su uso por todos los programas clientes v2.0 OPC (por ejemplo, aplicaciones SCADA, OPC Toolbox de Matlab...). Cuando el servidor OPC CX-Server se cierra recuerda qué archivo se ha cargado previamente, y este archivo se vuelve

a cargar automáticamente cada vez que se reinicia el programa. El nombre del servidor OPC para el Servidor OPC CX-Server es "OMRON.OpenDataServer.1" (utilizado para la configuración de los clientes OPC)⁽⁵⁾.

3.2.3 CX-Compolet

CX-Compolet se trata de un software optimizado para las nuevas series CJ2 que cuentan con una conexión Ethernet/IP. Además de dicha conexión se utiliza para transferir aplicaciones .NET al autómat. Por tanto el entorno de ejecución que soporta es la estructura .NET, el entorno de desarrollo es Visual Studio y los lenguajes soportados son Visual Basic y Visual C#. Esto limita muchas las opciones que nos da CX-Compolet y teniendo en cuenta que nuestra aplicación externa no es ni Visual Basic ni Visual C, esta opción fue desestimada desde el principio⁽¹⁰⁾.

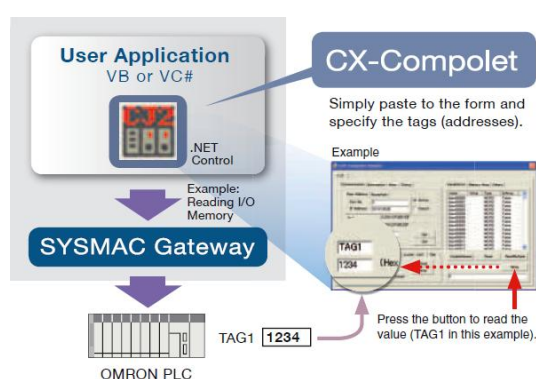


Figura 2 Estructura de comunicación CX-Compolet

De esta manera, teniendo en cuenta que CX-Server Lite está preparado para aplicaciones en Microsoft Excel, Visual Basic, Visual C++ o Visual Studio y CX-Compolet para aplicaciones .NET, la única opción posible era introducirnos en el mundo de la comunicación vía el estándar OPC. Tal y como se ha mencionado anteriormente, este tipo de comunicación consta de clientes y servidores que intercambian información entre ellos (OPC).

3.3 Configuración de una red local

Para la comunicación entre el autómat y el PC vía Ethernet es necesario crear una red local a la que ambos estén conectados de manera que puedan intercambiar información. Para la configuración de la red local es necesario establecer las direcciones IP del ordenador y el autómat, teniendo en cuenta que ambos han de tener la misma máscara de subred. En nuestro caso utilizaremos la máscara de subred 255.255.255.0, tal y como se muestra en la Figura 9. Estos números nos indican que las direcciones IP que tengamos dentro de esta red compartirán los tres primeros números variando el cuarto. Este cuarto número es conocido como nodo. De esta manera podemos asignar hasta 254 IPs distintas (la 0 y la 255 están reservadas para usos más avanzados)⁽¹¹⁾.

Habiendo seleccionado la máscara de subred que deseamos utilizar, se procede primero a la configuración de la dirección IP de nuestro autómata⁽¹²⁾. Para ello es necesario contar con un cable serie y conectarnos al PC vía serie. Una vez que estemos conectados, ejecutamos el programa CX-Programmer, creamos un nuevo proyecto con nuestro autómata y establecemos una conexión toolbus (vía serie). Nos conectamos al autómata *online* a través de la herramienta *trabajar online*. Una vez conectados con el autómata, lo ponemos en modo *program* (pestaña PLC -> modo de operación -> programa). Véase la Figura 4.

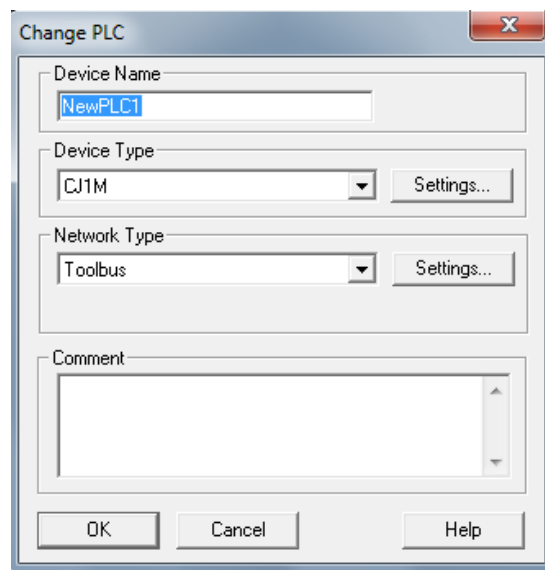


Figura 3 Selección de un nuevo PLC para conexión vía serie

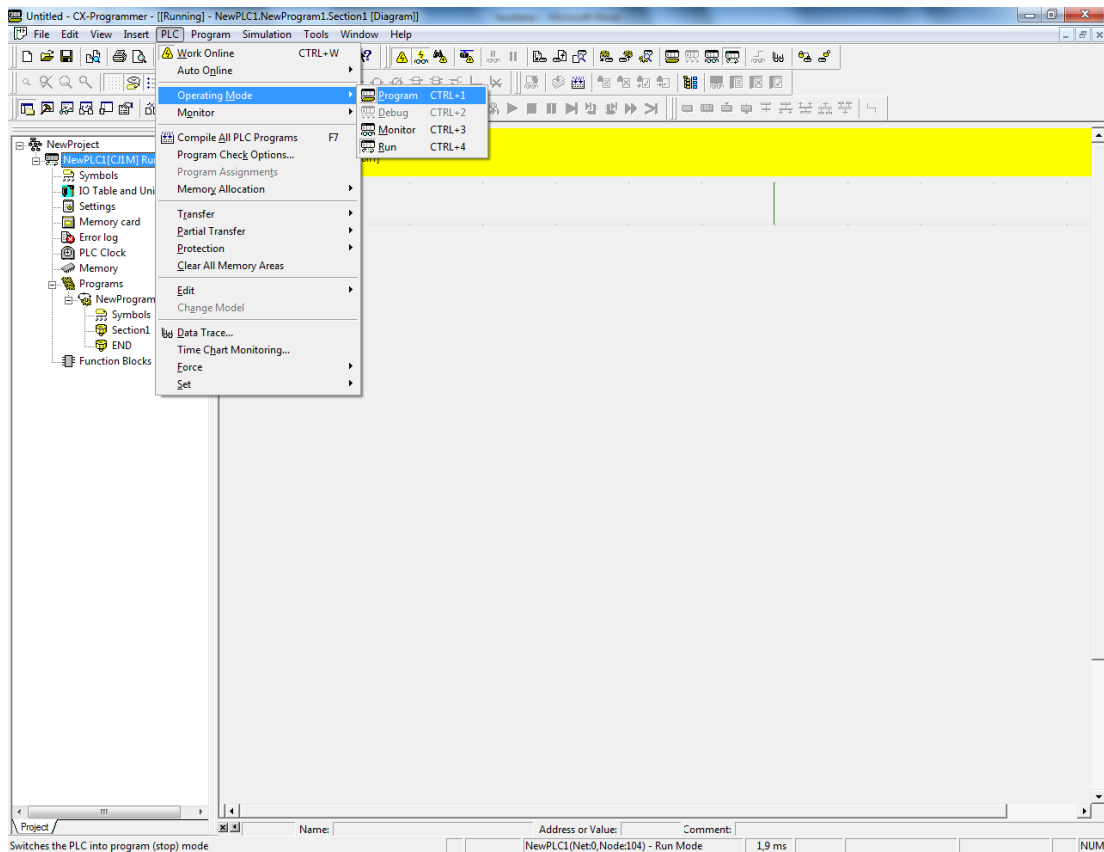


Figura 4 Configurar el PLC para ponerlo en modo programa

Una vez hayamos puesto nuestro autómatas en modo *program* abrimos el menú que aparece en la parte izquierda de la pantalla y configuramos nuestra tabla de entradas y salidas (Figura 5). Dentro de la pestaña bastidor principal podremos ir introduciendo uno a uno todos los hardwares que contiene nuestro autómatas. Elegimos el primer hueco y con el click derecho escogemos añadir unidad. Se desplegará un menú donde debemos elegir el tipo de unidad que deseamos añadir. En nuestro caso deseamos insertar un adaptador de comunicaciones tipo Ethernet (Communications Adapter -> CJ1W-ETN11 (Ethernet Unit)). A continuación vienen los pasos más importantes, ya que la configuración que introduzcamos tiene que coincidir con la escogida mediante hardware.

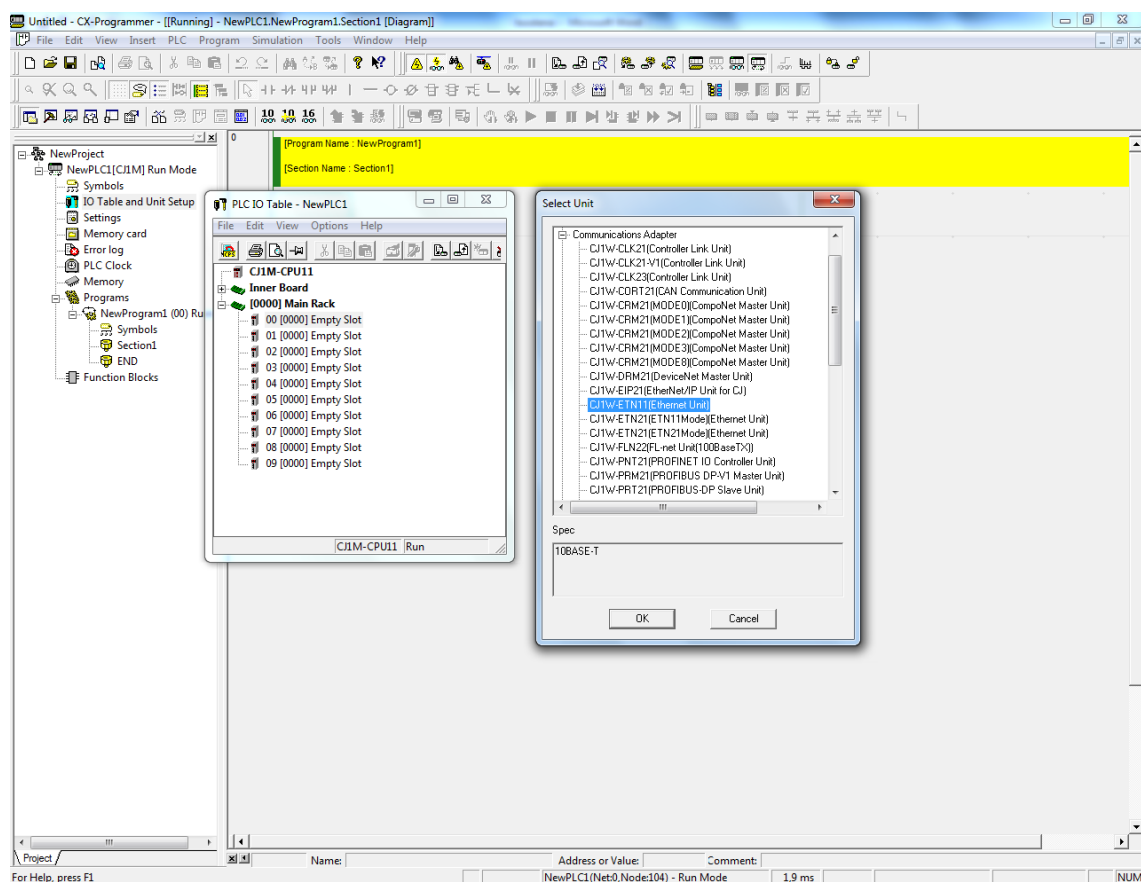


Figura 5 Introducir una nueva entrada de Ethernet

Todos los autómatas de Omron traen consigo unas ruedas de configuración manual que se denominan en su conjunto el *rotoswitch* (Figura 6). La primera de ellas sirve para definir el número de unidad y las dos siguientes para definir el número de nodo (en escala hexadecimal). Una vez que hayamos escogido el tipo de adaptador, deberemos introducir el número de unidad (tiene que coincidir con el número que aparece en la primera rueda del *rotoswitch*). Una vez introducido el número de unidad se nos creará el tipo de comunicación seleccionado. Haciendo click en dicha comunicación entraremos en el menú de editar, donde podremos introducir la dirección IP del autómata, así como la máscara de subred que deseemos utilizar (Figura 7). En nuestro caso la máscara de subred será 255.255.255.0 y la dirección IP que vamos a asignar será del tipo 10.10.10.XXX, la dirección 10.10.10.104 en concreto.

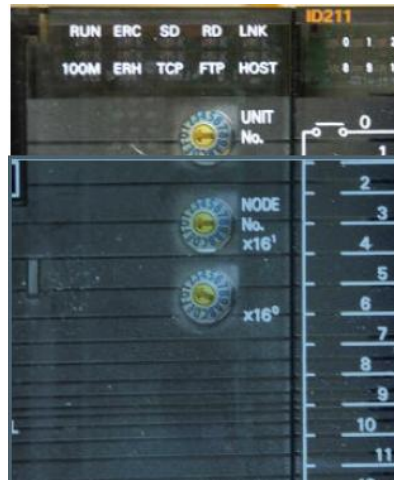


Figura 6 Rotoswitch de un PLC

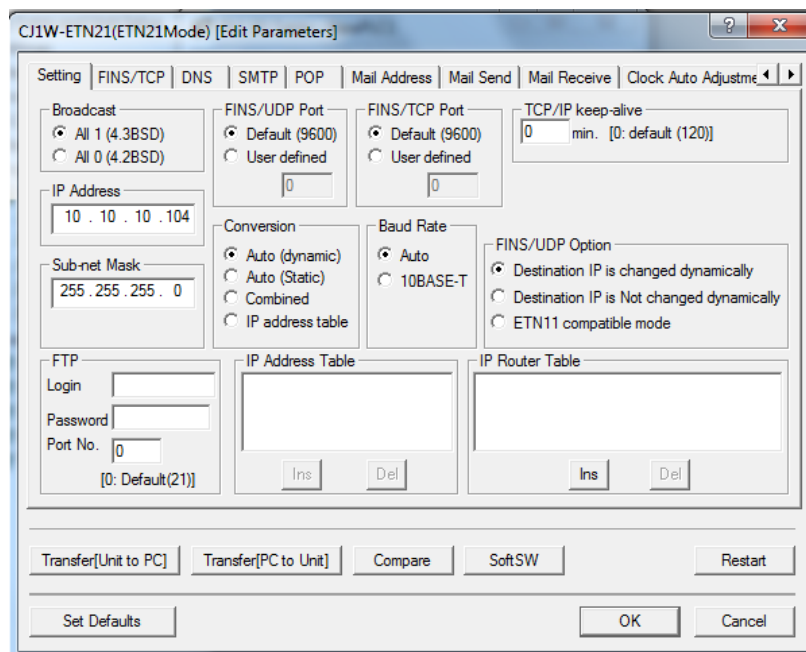


Figura 7 Editar parámetros de la comunicación Ethernet del PLC

Una vez realizados todos estos pasos, dentro del menú de tablas de entrada y salida, seleccionamos opciones y después transferir al PLC (Figura 8).

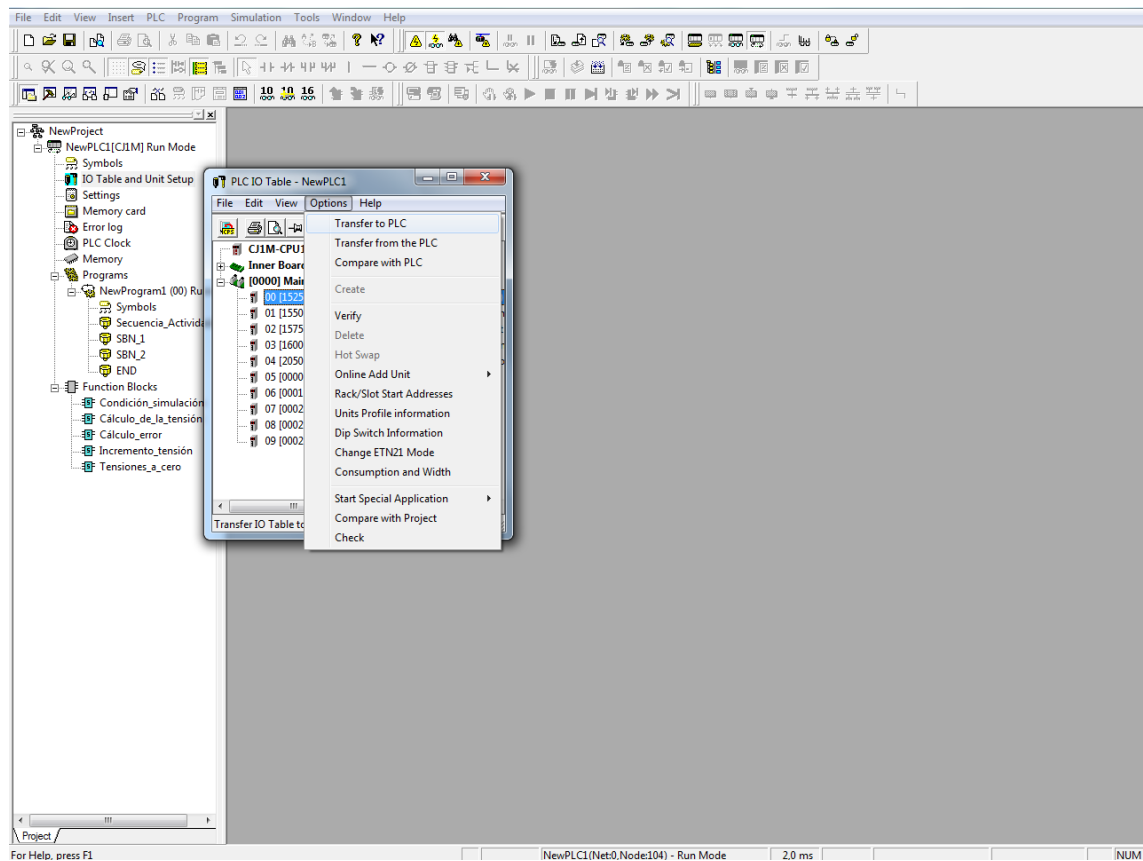


Figura 8 Transferir tabla de entradas y salidas al PLC

Una manera mucho más sencilla de realizar todo este trabajo es dejando que el propio CX-Programmer cree todas las tablas de entrada y salida automáticamente teniendo en cuenta los hardwares que contiene el automático. Los pasos a seguir para esta configuración automática se describen en el apartado Configurar tablas de entrada y salida de un PLC.

Una vez configurada la dirección IP de nuestro automático pasaremos a configurar la dirección IP de nuestro ordenador⁽¹³⁾. Para ello deberemos entrar en el Panel de control de nuestro ordenador y acceder al Centro de redes y recursos compartidos. Dentro de este menú seleccionaremos cambiar la configuración del adaptador. Una vez que accedamos a este último menú, aparecerán las distintas conexiones que tiene nuestro ordenador. Seleccionaremos la conexión de área local y configuraremos sus propiedades. Dentro de estas propiedades escogemos y cambiamos la configuración del protocolo TCP/IPv4. Llegaremos a la siguiente ventana:

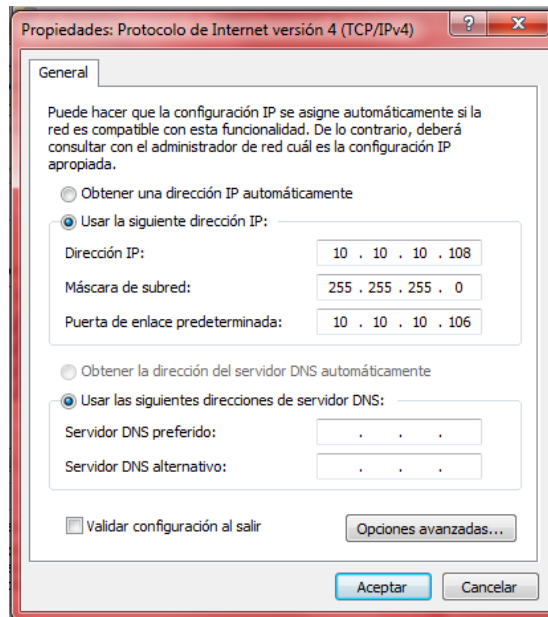


Figura 9 Configuración de la red local

En esta ventana configuraremos la dirección IP del autómata, la máscara de subred y la puerta de enlace. Tal y como se ha mencionado anteriormente, es necesario que todas las máquinas de nuestra red compartan los tres primeros números: 10.10.10.XXX (fijado en la máscara de subred). Por tanto le asignamos una IP a nuestra máquina siempre dentro del dominio mencionado anteriormente. La puerta de enlace o *Gateway* se utiliza cuando utilizamos un *router* para conectarnos al exterior a través de internet, pero para asegurarnos un mejor funcionamiento introducimos la dirección IP de nuestro autómata. Una vez realizada la configuración de la red, si ésta ha sido satisfactoria, deberá aparecer una nueva red en nuestro centro de redes.

Una vez realizados estos pasos, estamos en disposición de conectarnos a nuestro autómata vía Ethernet. Tenemos dos opciones, utilizar un cable RJ45 cruzado y conectarlo directamente con el autómata o utilizar un cable normal RJ45 y conectarnos con el autómata a través de un *switch*. Esta última opción es la que utilizaremos en este proyecto ya que queremos actuar y visualizar el resultado en distintas máquinas al mismo tiempo.

3.4 Configurar tablas de entrada y salida de un PLC

Una manera sencilla y rápida de configurar la tabla de entradas y salidas de un PLC es dejar que el programa CX-Programmer lo haga automáticamente. Puede ser muy útil cuando, por ejemplo, diseñamos el programa que queremos transferir a nuestro autómata y al volcar el programa, por error, volcamos todo lo que tenemos en nuestro programa sin darnos cuenta de que la tabla de entradas y salidas está normalmente vacía en nuestro programa. Es entonces cuando aparece un error en el autómata denominado

error H7. Este error supone que no hay ninguna tabla de entrada y salida en nuestra autómeta por lo que hay que volver a generarlas.

Para hacerlo de una manera rápida hay que seguir los siguientes pasos. Lo primero de todo es conectarnos con el PLC *online* y poner el autómeta en modo programa. Después accedemos al menú de la tabla de entradas y salidas situado a la izquierda de nuestra pantalla. Nos daremos cuenta que todas las entradas y salidas del bastidor principal están vacías. Seleccionamos el bastidor principal y hacemos click en opciones. Seleccionamos la opción crear y a continuación el programa mismo genera las entradas y salidas que detecte en nuestro autómeta. De esta manera podremos apreciar como nuestro bastidor principal se ha llenado de las entradas y salidas existentes y en caso de querer cambiar algún parámetro sólo deberemos de seleccionar dicha entrada o salida, cambiar ese parámetro y después transferirlo al autómeta⁽¹⁴⁾.

3.5 Real-Time Windows Target

El Real-Time Windows Target se trata de una herramienta de Matlab/Simulink que permite ejecutar los modelos y los bloques conectados a tablas de entrada y salida en tiempo real. Por tanto, permite al usuario crear y controlar sistemas en tiempo real para *“rapid prototyping”* o *“hardware-in-the-loop simulations”*. El Real-Time Windows Target consta de dos modos de simulación: modo normal (para operaciones sencillas) y el modo *external* (para una mayor eficiencia de las aplicaciones en tiempo real). Gracias a esta herramienta es posible ejecutar simulaciones en lazo cerrado, visualizar parámetros durante la simulación, controlar el sistema directamente desde Simulink y además, ofrece una gran variedad de módulos de entrada y salida y protocolos de comunicación⁽¹⁵⁾.

Una vez creado el modelo de Simulink y después de simularlo en modo normal, podemos agregar bloques de entradas y salidas del Real-Time Windows Target para representar nuestro hardware. Dependiendo del rendimiento que queramos obtener, podemos optar por simular de nuevo en modo normal o cambiar al modo *external*. Para el modo *external*, se genera un código ejecutable con la herramienta Simulink Coder y el compilador de C que tiene incluido. A continuación, se ejecuta esta aplicación con Simulink en modo *external* gracias a la herramienta *“connect to target”*. La integración entre el modo *external* de Simulink y el software Real-Time Windows Target permite utilizar el modelo de Simulink como una interfaz gráfica de usuario para:

- Visualización de señales a través del bloque Scope de Simulink.
- Cambio de parámetros durante la simulación a través del bloque Parameter dialog box.
- Control en tiempo real.
- *“Hardware-in-the-loop simulations”*.

La herramienta Real-Time Windows Target utiliza un pequeño *“real-time kernel”* que permite ejecutar las aplicaciones en tiempo real mediante la sincronización con el reloj de la CPU. Si estamos en el modo normal los bloques de entradas y salidas serán los únicos

en ejecutarse en tiempo real, en el modo *external*, en cambio, todas las aplicaciones se ejecutan en el *kernel*. Para poder realizar esta sincronización con el reloj de la CPU es necesario instalar el “*real-time kernel*” de Matlab. Para ello, basta con escribir el comando que aparece en la siguiente imagen en el *workspace* de Matlab:

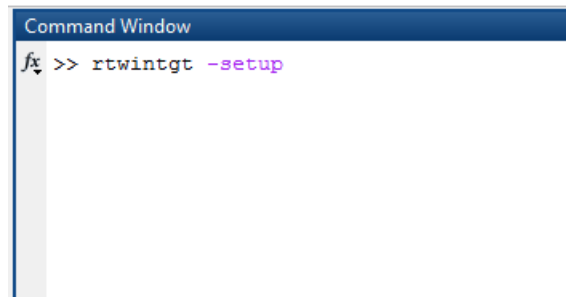


Figura 10 Instalar “*real-time kernel*”

3.5.1 Normal mode simulation

En caso de que la simulación se desee realizar en el ordenador proveedor, la simulación en modo normal puede ser muy apropiada. En este tipo de simulación, Simulink ejecuta el algoritmo de simulación y el “*real-time kernel*” ejecuta los drivers de entrada y salida en tiempo real. De esta manera, tanto el proceso de Simulink como el *kernel* se ejecutan en la misma máquina. En este modo de simulación se puede utilizar tanto el *solver* de paso variable como el *solver* de paso fijo. Al ejecutarse los drivers de entrada y salida en tiempo real, la comunicación con las entradas y salidas se realiza en tiempo real, sincronizando de esta manera toda la simulación con el reloj de la CPU.

Para este tipo de simulaciones no es necesario generar un código, de manera que para ejecutar el modelo basta con especificar el tiempo de duración de la simulación y accionar la tecla de *play*.

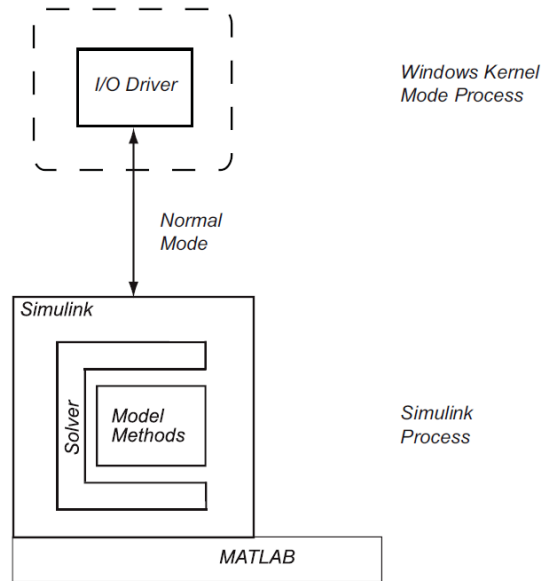


Figura 11 Real-time normal mode simulation

3.5.2 External mode simulation

Para una ejecución más precisa de nuestro modelo la mejor opción es ejecutar nuestro modelo en modo *external*. Este modo *external* incluye el Simulink Coder para generar el código en C y un compilador de dicho código, todo ello para crear un ejecutable que pueda ser ejecutado por el *kernel* en tiempo real. Este tipo de aplicaciones utilizan los parámetros iniciales que estén disponibles en el momento de generar el código, si después se cambian, no los tiene en cuenta. En consecuencia, si incluimos en nuestro modelo componentes que no varían con el tiempo no habrá ningún problema, en caso contrario sí. Además, solo acepta *solver* de paso fijo.

El Simulink Coder genera básicamente el código necesario para la comunicación entre el algoritmo del modelo a simular y los drivers de entrada y salida. Esto permite una total sincronización con reloj de la CPU a tiempo real dejando en manos de Simulink sólo la representación de los resultados que se pueden leer del ejecutable generado.

Para generar dicho código es necesario acceder al menú de código y seleccionar generar código C/C++. Una vez se genere el código hacemos click en “*connect to target*”. Después de que se haya compilado el código podremos empezar la simulación dándole al *play*.

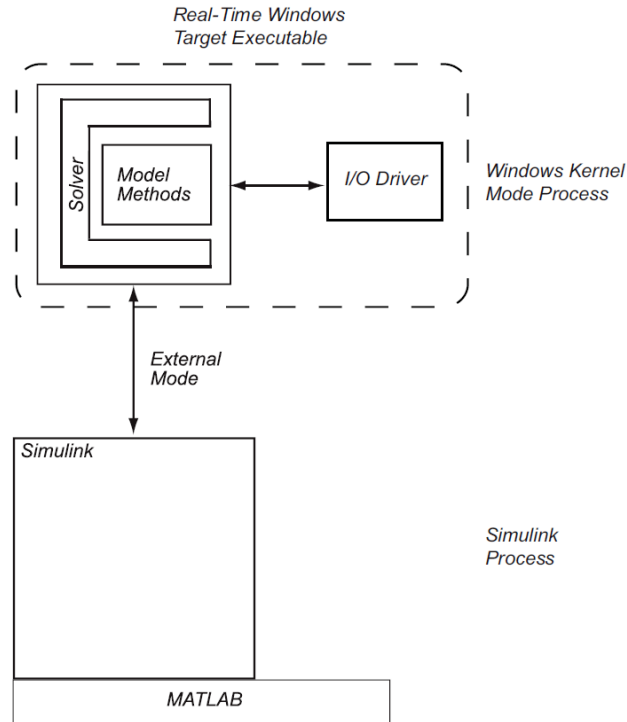


Figura 12 Real-time external mode simulation

3.6 OPC Foundation Core Components

Antes de realizar la comunicación con cualquier servidor OPC de la red es necesario preparar la estación de trabajo (y posiblemente también el servidor OPC) para utilizar la tecnología sobre la cual se sustenta nuestra OPC Toolbox. Esta tecnología permite conectarse con el servidor OPC para obtener e intercambiar información, así como interactuar con el simulador. Para adecuar nuestro espacio de trabajo es necesario instalar el OPC Foundation Core Components. Se trata de un conjunto de herramientas proporcionadas por la OPC Foundation⁽⁴⁾ para establecer conexiones entre los servidores OPC y los ordenadores conectados a la red, así como para acceder a los servidores OPC del ordenador local. Para instalar dicho paquete de herramientas es necesario introducir el siguiente comando en el workspace de Matlab⁽⁶⁾:

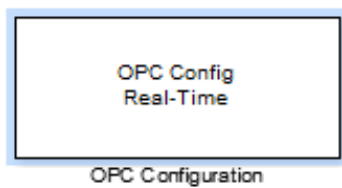
```
Command Window
fx >> opcregister('install')
```

Figura 13 Instalar OPC Foundation Core Components

Una vez instalado el paquete de herramientas para la comunicación entre los servidores y clientes OPC, realizar las comunicaciones es bastante sencillo e intuitivo. Dentro de la OPC Toolbox contamos con 3 principales bloques:

- OPC Configuration Block
- OPC Read Block
- OPC Write Block

3.6.1 OPC Configuration Block



Se trata del bloque de configuración. Es necesario incluir este bloque de configuración cuando vayamos a utilizar el cliente OPC. Este bloque no tiene ningún argumento de entrada ni de salida. Se trata del bloque que configura el cliente OPC, las opciones de ejecución del pseudo real-time y las respuestas a los posibles errores que vayan a aparecer.

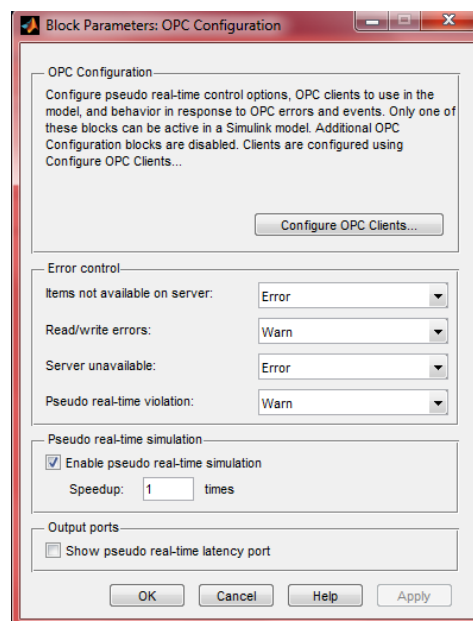


Figura 14 Configuración de los parámetros del OPC Configuration Block

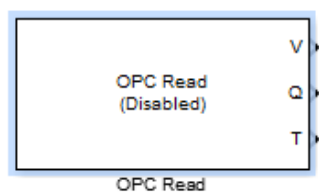
Configure OPC Clients ejecuta el OPC Client Manager para el modelo en concreto. Cada modelo tiene una lista de clientes asociados y estos clientes son utilizados para las operaciones de escritura y lectura. Gracias a esta herramienta seleccionamos el servidor al que deseamos conectarnos.

Error control define las acciones que ha de realizar Simulink cuando aparezcan unos errores específicos. Las respuestas que puede dar Simulinks son tres: parar la simulación, generar una advertencia pero continuar con la simulación o ignorar dichos errores.

Pseudo real-time simulation permite configurar la simulación en tiempo real. Cuando seleccionamos el modo pseudo real-time intentamos aproximar lo más cerca posible la ejecución de nuestro modelo al reloj de la CPU, no se trata de una sincronización perfecta con el reloj pero sí muy parecida. El *display* de *speedup* define cuánto más rápido del tiempo real se quiere ir, por ejemplo, si dicho parámetro lo fijamos en 3, nos llevará 5 segundos la ejecución de 15 segundos de simulación.

Pseudo real-time latency port nos incluye un puerto de salida al bloque de configuración. La latencia es el tiempo de espera necesario en cada paso de ejecución del modelo.

3.6.2 OPC Read Block



El bloque OPC Read lee uno o más puntos de un servidor OPC. Dicho servidor OPC será aquel que esté configurado en el bloque de configuración. La operación de lectura de datos se puede dar de una manera síncrona (lectura desde caché o desde el dispositivo) o asíncrona (únicamente desde el dispositivo). Las salidas de este bloque son V (valores del punto solicitado) y dos salidas opcionales Q (identificador de calidad) y T (marcas de tiempo asociadas a cada valor de las salidas adicionales). Este último puede ser bien el tiempo real que ha pasado desde el inicio de la simulación o bien el tiempo de simulación empleado. Es necesario que este bloque vaya acompañado de un bloque de configuración del OPC para su correcta ejecución.

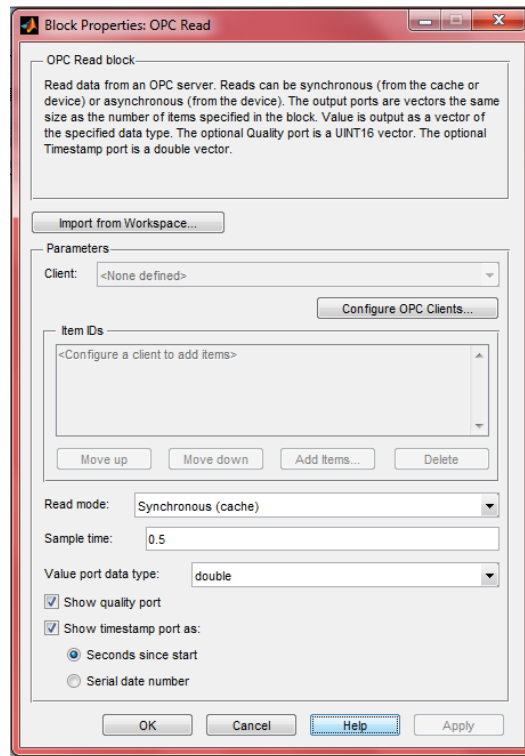


Figura 15 Configuración de los parámetros del OPC Read Block

Import from Workspace permite importar los ajustes realizados en el bloque OPC Read al workspace de Matlab.

Client define el cliente asociado a dicho bloque de lectura. Utilizando la pestaña *Configure OPC Clients* accedemos al OPC Client Manager y podemos realizar los ajustes mencionados en el bloque OPC Configuration Block.

Item IDs muestra los puntos que van a ser leídos desde un servidor determinado. Utilizando la pestaña *Add Items* podemos añadir tantos puntos como tengamos fijados en el servidor. Con *Delete Items* los eliminamos de la lista. Las pestañas *Move Up* y *Move Down* sirven para reordenar los puntos de lectura. El orden determina el orden en que se mostrarán los datos a la salida del bloque.

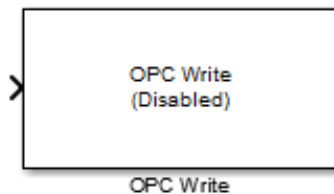
Read mode define el modo de lectura que se desea utilizar. Puede ser asíncrono, síncrono desde caché o síncrono desde dispositivo. El modo síncrono tiene un riesgo algo mayor pero normalmente los resultados son más fiables que el asíncrono.

Sample time define el tiempo de muestreo del bloque. Nos indica la frecuencia con la que leerá los datos de nuestro servidor.

Value port data type define el tipo de dato a la salida. El servidor OPC es el responsable de convertir todos los datos al tipo de dato requerido en la salida.

Show quality port y Show timestamp port nos permite elegir si deseamos que dichos puertos de salida estén visibles en el bloque. Tal y como se ha mencionado anteriormente dichos puertos son opcionales, por tanto podemos ocultarlos si los deseamos.

3.6.3 OPC Write Block



El bloque OPC Write escribe datos en uno o más puntos de un servidor OPC. Al igual que en el bloque OPC Read la escritura se puede realizar de manera síncrona o asíncrona. Cada elemento de entrada se escribe en la dirección de memoria correspondiente definida en la lista de puntos del bloque OPC Write. De la misma manera que en los bloques de lectura, estos bloques de escritura también han de ir acompañados de un bloque de configuración del OPC.

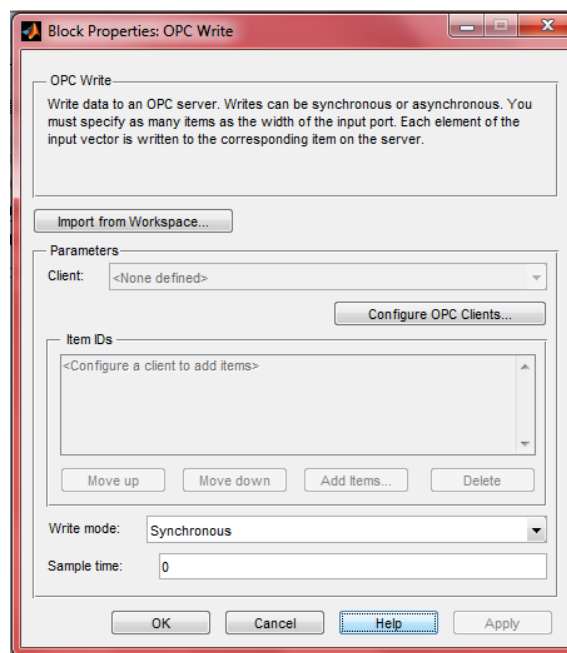


Figura 16 Configuración de los parámetros del OPC Write Block

Import from Workspace permite importar los ajustes realizados en bloque OPC Write al workspace de Matlab.

Client define el cliente asociado a dicho bloque de escritura. Utilizando la pestaña Configure OPC Clients accedemos al OPC Client Manager y podemos realizar los ajustes mencionados en el bloque OPC Configuration Block.

Item IDs muestra los puntos que van a ser escritos en un servidor determinado. Utilizando la pestaña *Add Items* podemos añadir tantos puntos como tengamos fijados en el servidor. Con *Delete Items* los eliminamos de la lista. Las pestañas *Move Up* y *Move Down* sirven para reordenar los puntos de escritura. El orden determina el orden en que se mostrarán los datos a la salida del bloque.

Write mode define el modo de escritura a utilizar. Las opciones disponibles son la escritura asíncrona y la síncrona. Al igual que en el bloque de lectura el modo síncrono tiene un riesgo algo mayor pero normalmente los resultados son más fiables que el asíncrono.

Sample time define el tiempo de muestreo del bloque. Nos indica la frecuencia con la que escribirá los datos en nuestro servidor.

3.7 Combinación de las herramientas Real-Time Windows Target y OPC Toolbox

Una vez investigadas las herramientas Real-Time Windows Target y OPC Toolbox de Matlab/Simulink, que nos permiten realizar simulaciones en tiempo real y comunicarnos con un hardware externo a través de un servidor OPC, llega el momento de combinar ambas herramientas. Tal y como se ha descrito unas cuantas veces se quiere realizar una “*hardware-in-the-loop simulation*” para testear nuestro algoritmo de control. Para ello debemos ejecutarlo en tiempo real y poder interactuar con el control mientras ejecutamos nuestro modelo en el simulador.

Por lo tanto, una vez diseñado nuestro modelo matemático debemos aplicar dos pequeños cambios para ejecutarlo en tiempo real e interactuar con el controlador. La primera opción que se ha tomado ha sido la de optar por ejecutar el modelo en modo *external* para obtener una mayor eficiencia en la simulación en tiempo real (tal y como se ha explicado en el apartado External mode simulation). Ello conlleva la necesidad de generar un código en C/C++ y compilarlo, pulsar el botón “*connect to target*” y ejecutar la simulación.

Este método ha resultado ser problemático ya que al ejecutar el Simulink Coder para generar el código en C resulta que necesita un archivo denominado *opcslread.tlc* que la versión de Matlab R2013b no incluye. Investigando en la red se ha encontrado información sobre los archivos *.tlc, llegando a la conclusión de que varios archivos *.tlc han sido eliminados en las últimas versiones de Matlab haciendo imposible la ejecución de nuestro modelo en el modo *external*. Este archivo, necesario para la ejecución, tiene que ver sólo y exclusivamente con el cliente OPC de Matlab. En otras aplicaciones donde no se utilice la herramienta OPC el modo *external* funciona perfectamente, pero es cuando implementamos la herramienta OPC donde surge el problema. El cliente OPC de Matlab R2013b no incluye el archivo *opcslread.tlc* necesario para la generación del código en C⁽¹⁶⁾.

De esta manera la única opción que queda es utilizar el pseudo real-time que ofrece la herramienta OPC de Matlab, ya que al utilizar la comunicación OPC no podemos utilizar los bloques de entradas y salidas necesarios para la ejecución en tiempo real en modo normal.

Solucionado el problema de la ejecución en tiempo real se introducen los bloques necesarios para adaptar el modelo para interactuar con el servidor OPC. De la misma manera que se ha explicado en apartados anteriores, es necesario introducir el bloque de configuración del OPC y determinar sus características principales (OPC Configuration Block). Muchas veces conviene no dar ninguna respuesta a las advertencias ya que éstas ralentizan la simulación considerablemente. Al habilitar el pseudo real-time se puede elegir la velocidad que le queremos dar a la simulación. Introduciendo un 1 en la ventana de *speedup* sincronizaremos la simulación con el reloj aproximadamente, obteniendo una simulación en tiempo real. Después se introducen los bloques de entradas y salidas teniendo en cuenta las entradas y salidas de nuestro modelo. En dichos bloques seleccionamos los datos del servidor que van a actuar como entradas y salidas. También es importante tener en cuenta que variando el *sample time* cambia el tiempo de simulación. Cuanto más pequeño sea el *sample time*, más tardará en simular el modelo. En nuestro caso el *sample time* que más aproxima la simulación al tiempo real es de 0,1. Esto implica que nuestro modelo leerá y escribirá los datos cada 0,1s o 100ms.

Una vez realizadas estas configuraciones nuestro modelo está listo para simular y obtener los resultados pertinentes.

3.8 Configuración DCOM para habilitar la comunicación entre aplicaciones que se ejecutan en distintas máquinas

La configuración DCOM es utilizada cuando deseamos conectarnos a un servidor OPC que no se encuentra en la misma máquina, esto es, el cliente OPC se encuentra en un ordenador y el servidor en otro. Para comunicarse e intercambiar información entre ellos es necesario cambiar la configuración DCOM de cada máquina⁽⁶⁾.

DCOM es una arquitectura basada en la estructura cliente-servidor para permitir la comunicación entre dos aplicaciones que se ejecutan en equipos distintos. El OPC Data Access y el OPC Historical Data Access utilizan DCOM para la comunicación entre el cliente OPC (por ejemplo, el cliente OPC Toolbox) y el servidor OPC. Para utilizar con éxito DCOM, los dos equipos tendrán que compartir una configuración de seguridad común, de modo que las dos aplicaciones se han concedido los derechos necesarios para comunicarse uno con el otro. Para conectarse satisfactoriamente a servidores OPC utilizando la OPC Toolbox de Matlab, es necesario configurar los permisos de DCOM entre el equipo cliente (en el que MATLAB está instalado) y el equipo servidor (donde se ejecuta el servidor OPC).

Para realizar la configuración de los permisos DCOM existen dos métodos, uno más seguro que el otro. El primero, y más seguro de los dos, es una configuración bajo nombre de usuario ("*Configure DCOM to Use Named User Security*"). El segundo en cambio es una configuración sin seguridad ("*Configure DCOM to Use No Security*"). Este último caso sólo es recomendable en el caso de que la red que se esté utilizando sea totalmente segura y de confianza, ya que nuestros equipos serán vulnerables a intrusiones malignas de otros usuarios que estén conectados a la misma red.

3.8.1 "Configure DCOM to Use Named User Security"

Para configurar los permisos DCOM bajo nombre de usuario, habrá que asegurarse de que tanto la máquina del servidor como la máquina cliente tienen un usuario en común que tenga derechos de acceso a la configuración DCOM en el servidor y en los equipos cliente.

Para configurar los permisos DCOM de la máquina donde se encuentra el servidor OPC se han de seguir los siguientes pasos. Para más detalles se recomienda consultar la guía del usuario del cliente OPC de Matlab⁽⁶⁾ (Capítulo 1. Introducción -> Instalar OPC Toolbox software -> Configurar DCOM):

- 1) Crear un nuevo usuario local.
- 2) Seleccionar Inicio -> Panel de control -> Sistema y seguridad -> Herramientas administrativas -> Servicios de componentes.
- 3) Entrar en Equipos -> Mi PC -> Configuración DCOM.
- 4) Buscar el servidor OPC en la lista de configuración DCOM.
- 5) Escogemos editar sus propiedades.
- 6) En la pestaña *general* verificar que la autenticación está puesta *por defecto o conectar*.
- 7) En la pestaña *seguridad* editar los parámetros de los permisos de inicio y activación -> el usuario creado ha de tener permisos de ejecución local y activación local.
- 8) Dentro de la misma pestaña editar los permisos de acceso -> el usuario creado ha de tener permiso de acceso local.
- 9) En la pestaña *identidad* escoger este usuario e introducir el nombre de usuario y contraseña del usuario creado anteriormente.

Una vez realizada la configuración en la máquina donde se encuentra el servidor OPC, hay que configurar los permisos DCOM de la máquina donde se está ejecutando Matlab y su cliente OPC. De la misma manera que en el apartado anterior, a continuación se enumeran los pasos a seguir, pero para más detalles se recomienda siempre consultar la guía del usuario del cliente OPC de Matlab⁽⁶⁾ (Capítulo 1. Introducción -> Instalar OPC Toolbox software -> Configurar DCOM):

- 1) Crear el mismo usuario local que se ha creado en la máquina donde tenemos instalado el servidor OPC.

- 2) Seleccionar Inicio -> Panel de control -> Sistema y seguridad -> Herramientas administrativas -> Servicios de componentes.
- 3) Entrar en Equipos -> Mi PC y entrar en propiedades utilizando el click derecho.
- 4) Entrar en la pestaña *propiedades predeterminadas* y asegurarse de que está seleccionado *Habilitar COM distribuido en este equipo*, el nivel de autenticación predeterminado esté en *conectar* y el nivel de suplantación predeterminado esté en *identificar*.
- 5) Entrar en la pestaña *seguridad COM*.
- 6) En los permisos de acceso -> editar valores predeterminados -> asegurarse de que el usuario creado tiene permisos tanto de acceso local como remoto.
- 7) En los permisos de acceso -> editar límites -> asegurarse de que el usuario creado tiene permisos tanto de acceso local como remoto.
- 8) Dentro de la misma pestaña editar los permisos de inicio y activación -> editar valores predeterminados y asegurarse de que el usuario creado tiene todos los permisos.
- 9) Realizar lo mismo dentro de editar límites de los permisos de inicio y activación.
- 10) Al finalizar, seleccionamos aceptar y aparecerá una advertencia advirtiéndole de que se va a cambiar la configuración DCOM -> seleccionamos aceptar.

Tanto la máquina del cliente como la del servidor OPC están configuradas para utilizar el mismo nombre de usuario cuando el servidor intente establecer una conexión con el cliente OPC.

3.8.2 “Configure DCOM to Use No Security”

Es muy importante recordar que este tipo de configuración sólo se debe utilizar en caso de que el usuario esté completamente seguro de que la red a la que esté conectado es totalmente segura. La desactivación de la seguridad DCOM significa que cualquier usuario de la red puede lanzar cualquier objeto COM al equipo local poniendo en riesgo la seguridad de nuestro equipo.

En este caso, los pasos a seguir para la configuración de los permisos DCOM es la misma para ambas máquinas (la máquina donde está instalado el servidor y la máquina donde se encuentra el cliente). Los pasos a seguir son los siguientes, pero tal y como se ha mencionado en el apartado anterior para más detalles se recomienda siempre consultar la guía del usuario del cliente OPC de Matlab⁽⁶⁾ (Capítulo 1. Introducción -> Instalar OPC Toolbox software -> Configurar DCOM):

- 1) Asegurarse de que el usuario tipo *guest* está habilitado en la máquina.
- 2) Seleccionar Inicio -> Panel de control -> Sistema y seguridad -> Herramientas administrativas -> Servicios de componentes.
- 3) Entrar en Equipos -> Mi PC y entrar en propiedades utilizando el click derecho.
- 4) Entrar en la pestaña *propiedades predeterminadas* y asegurarse de que está seleccionado *Habilitar COM distribuido en este equipo*, el nivel de autenticación

predeterminado esté en *ninguno* y el nivel de suplantación predeterminado esté en *anónimo*.

- 5) Entrar en la pestaña *seguridad COM* -> permisos de acceso -> editar límites -> asegurarse de que el usuario anónimo tiene permisos tanto de acceso local como remoto.
- 6) Dentro de la misma pestaña editar los permisos de inicio y activación -> editar valores predeterminados y asegurarse de que el usuario anónimo tiene todos los permisos.

Ahora tanto el cliente como el servidor están configurados de manera que cualquiera pueda acceder a un objeto COM en cualquiera de las máquinas.

Una vez realizada la configuración de los permisos DCOM, la manera de comunicarse es bien sencilla. Cuando configuremos el cliente OPC para conectarnos al servidor, por defecto, busca servidores en el equipo local (*localhost*). *Localhost* es una manera de llamar a la IP que tiene asignada nuestro equipo local. Si en vez de *localhost* introducimos la dirección IP que tiene el ordenador donde está instalado el servidor, comenzará a buscar servidores OPC dentro de dicha máquina. Seleccionamos el servidor que deseamos y continuamos con los pasos descritos en el apartado OPC Foundation Core Components.

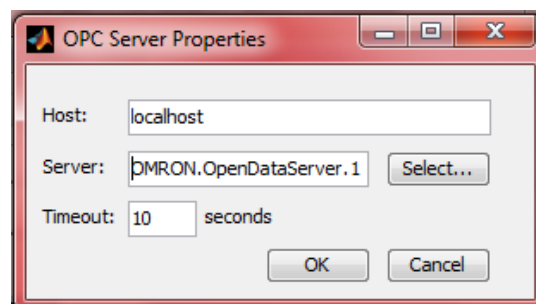


Figura 17 Selección del servidor OPC

3.9 SCADA

SCADA, acrónimo de “*Supervisory Control And Data Acquisition*”, es un software para ordenadores que permite controlar y supervisar procesos industriales a distancia. Facilita la retroalimentación en tiempo real con los dispositivos de campo (sensores y actuadores), y controla el proceso automáticamente. Provee de toda la información que se genera en el proceso productivo (supervisión, control calidad, control de producción, almacenamiento de datos...) y permite su gestión e intervención.

Los primeros SCADA eran simplemente sistemas de telemetría que proporcionaban datos periódicos de las distintas condiciones medidas a través de sensores. Estos sistemas ofrecían una sencilla oportunidad de monitorizar y controlar el sistema. Muchas empresas

vieron la necesidad de realizar funciones de control más complejos y de esta manera surgió la industria de los SCADA, con el objetivo de atender dichas necesidades. Hoy en día los proveedores de SCADA están diseñando sistemas que están pensados para resolver las necesidades de cada empresa con módulos de software específicamente diseñados para cada empresa. Estos sistemas normalmente suelen estar integrados dentro de la estructura que genera la información que se quiere tratar, esto es, vienen incluidos dentro de la propia máquina. Convirtiéndose de esta manera en un recurso importante de información, dejando atrás la funcionalidad opcional que desempeñaban.

Todo SCADA se diseña con el objetivo de mejorar la comunicación entre el operario y la máquina y facilitar la monitorización de los parámetros sobre los que se desee actuar. Para ello los SCADA han de cumplir un mínimo de funcionalidades. La primera de ellas es la adquisición y almacenado de datos para recoger y procesar toda la información. También tienen que ser capaces de representar gráficamente las variables del sistema, así como ejecutar acciones de control sobre ellas. Cumplen el principal objetivo de supervisar la evolución de las variables y gestionan a su vez una base de datos con dichas evoluciones. Por último, tienen que ser capaces de explotar los datos adquiridos y alertar de todos los cambios que ocurran en el sistema (sean normales o imprevistos)⁽¹⁷⁾.

La razón por la que se ha decidido incluir una interfaz operario/máquina (HMI) en este proyecto ha sido la necesidad de facilitar la utilización del software al usuario. Cabe mencionar que la palabra HMI y SCADA no son exactamente el mismo término aunque sí muy parecidos. Los sistemas SCADA cumplen todas las funcionalidades de un HMI, pero además cuenta con una función de supervisión de las variables que carecen los HMI. Con este software se nos permite crear una pantalla dinámica y fácil de comprender para el usuario. Se ha intentado realizar una pantalla lo más intuitiva posible para que el usuario pueda introducir los datos necesarios y pueda observar los resultados claramente. A continuación, en la Figura 18, se muestra el resultado obtenido.

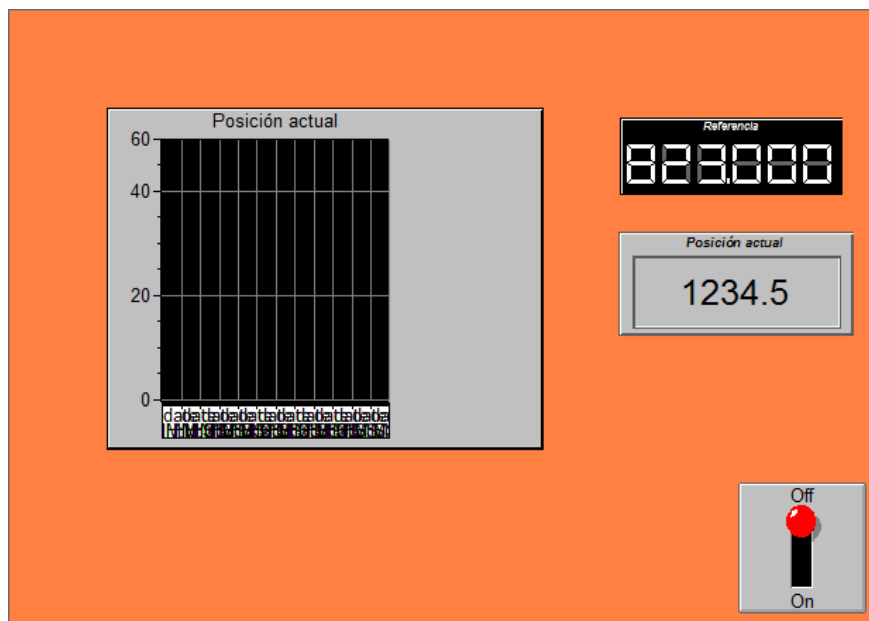


Figura 18 SCADA diseñado para nuestro sistema

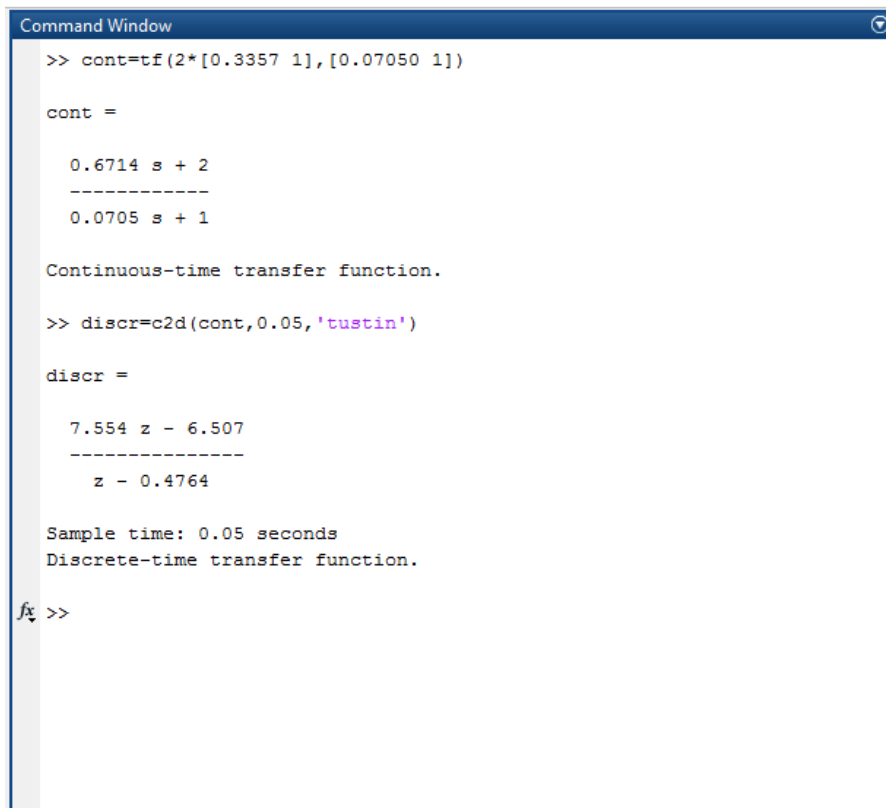
Dicha pantalla contiene un botón de activación de nuestro autómatas. Cuando dicho botón se ponen en ON se ejecuta la subrutina introducida en el autómatas. Además tiene unos cuadros donde se muestran la referencia (objetivo de nuestro sistema) y la posición que tiene nuestro sistema en todo momento. Por último, incluye una representación gráfica de la evolución de la posición del sistema en todo momento.

Para realizar dicha pantalla es necesario introducir el autómatas con el que vamos a trabajar (de la misma manera que se muestra en la Figura 3) y adjudicar los puntos o direcciones de memoria con los que vamos a trabajar. Una vez tengamos nuestro proyecto con los datos del PLC y las direcciones de memoria con las que vamos a trabajar introduciremos los distintos dibujos gráficos que deseemos. Después de introducir nuestros controles gráficos accedemos a las propiedades de los mismos para configurar y asignarles las direcciones de memoria seleccionadas en el proyecto. También podremos editar otros parámetros visuales como los colores, número de decimales en los *displays*...

Para desarrollar la pantalla se ha utilizado el software *CX-Supervisor*. Este software cuenta con dos herramientas. La primera de ellas es el desarrollador (*Developer*) que nos permite diseñar nuestra pantalla, así como adjudicar el autómatas y las direcciones de memoria. Una vez creada nuestra pantalla, se compila y se ejecuta la pantalla creada en la herramienta *runtime*. Esta herramienta nos permite ejecutar la pantalla e interactuar con nuestro autómatas. Todo este proceso es fácil e intuitivo. El mundo de los sistemas SCADA es bastante sencillo de manejar y además permite facilitar el control de nuestro sistema para cualquier persona ajena al desarrollo de dicho sistema.

4. Resultados

Tal y como estaba planteado el problema en el libro (apartado Descripción del sistema) y teniendo en cuenta que nuestro autómata trabaja en discreto es necesario discretizar primero la función de transferencia del controlador. Para ello se ha utilizado el siguiente comando de Matlab que nos transforma una función de transferencia en modo continuo a su correspondiente en modo discreto.



```
Command Window

>> cont=tf(2*[0.3357 1],[0.07050 1])

cont =

    0.6714 s + 2
    -----
    0.0705 s + 1

Continuous-time transfer function.

>> discr=c2d(cont,0.05,'tustin')

discr =

    7.554 z - 6.507
    -----
    z - 0.4764

Sample time: 0.05 seconds
Discrete-time transfer function.

fx >>
```

Figura 19 Transformación de una función de transferencia de modo continuo a discreto

El número 0.05 define el *sample time* con el que va a trabajar nuestro autómata. Este *sample time* tiene que coincidir con la frecuencia a la que se ha calculado el controlador. Como se mencionaba en el problema del libro (apartado Descripción del sistema) la frecuencia tenía que ser de 20 seg^{-1} o mejor dicho, 0.05 segundos. La palabra *tustin*, en cambio, define el método que se va utilizar para la conversión de dicha función. El comando *c2d* de Matlab es la función encargada de transformar la función de continuo a discreto. Una vez obtenida la función de transferencia del compensador en discreto se procede a la programación del simulador y el controlador.

4.1 Simulación teórica

La simulación teórica consiste en implementar nuestro modelo íntegramente en Simulink y obtener los resultados *off-line*. Utilizaremos los datos obtenidos para contrastarlos con los obtenidos en la simulación con “*hardware-in-the-loop*”. Para ello, introducimos nuestro modelo en discreto en Simulink. Con el cambio realizado en la función de transferencia del compensador implementamos el nuevo lazo de control obtenido con el siguiente resultado:

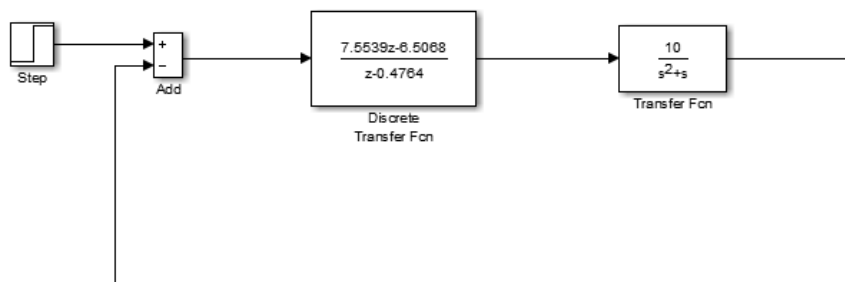


Figura 20 Lazo de control en modo discreto

Con este modelo se han conseguido los resultados del modelo teórico (modelo que se ha descrito en el apartado Descripción del sistema). La figura que se muestra a continuación muestra la respuesta que tiene nuestro sistema ante una entrada escalón de una amplitud determinada (25 en este caso).

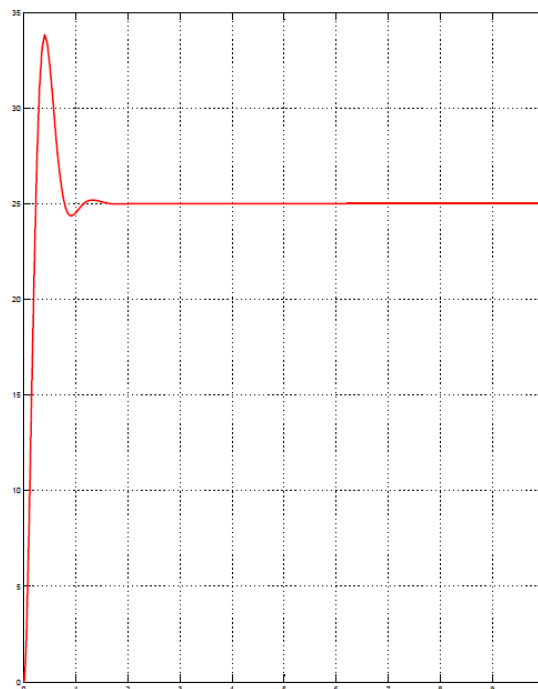


Figura 21 Simulación teórica

Los datos obtenidos para la realización de esta figura muestran que se trata de un sistema totalmente controlado. Después de un segundo y medio, el sistema está totalmente controlado y ajustado al objetivo. Tiene un sobreimpulso al comienzo, pero los siguientes picos son muy pequeños y cercanos al objetivo, por tanto se puede considerar que el controlador funciona correctamente.

4.2 “Hardware-in-the-loop simulation”

4.2.1 Implementación del algoritmo de control en el autómata

Cuando hablamos de “*hardware-in-the-loop*” nos referimos a introducir un hardware dentro de la simulación, bien un sensor, bien un controlador como en este caso. Dentro de la simulación sustituimos el bloque asociado al controlador por las entradas y salidas a nuestro hardware a través de los bloques de la herramienta OPC de Matlab/Simulink (OPC Foundation Core Components). De este modo nuestro modelo de Simulink queda de la siguiente manera:

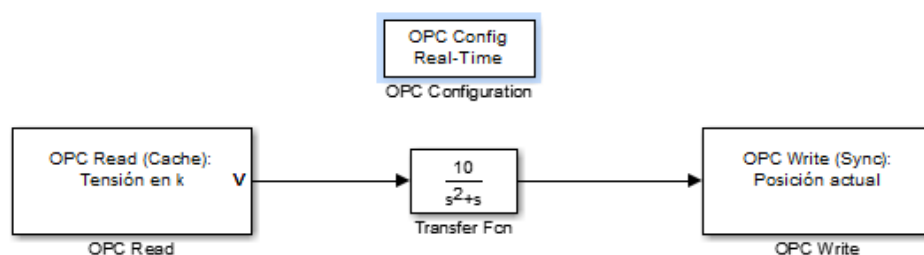


Figura 22 Simulación con “*hardware-in-the-loop*”

Establecemos una conexión vía servidor OPC con nuestro controlador cada 0.1 segundos de manera que garantizamos la simulación a tiempo real. Configurando el OPC en *pseudo real-time* hay veces que no es suficiente para asegurar la simulación en tiempo real. Después de testear distintos *sample times* en los bloques de escritura y lectura, se ha llegado a la conclusión que el *sample times* que asegura una simulación a tiempo real es 0.1 segundos (Combinación de las herramientas Real-Time Windows Target y OPC Toolbox).

En cuanto al algoritmo de control a implementar en el hardware es necesario inicializar los valores de nuestras variables al comienzo de la simulación. Para ello es muy útil crear una subrutina en nuestro autómata de manera que se empiece ejecutar dicha subrutina cuando comience la simulación. De esta manera conseguimos sincronizar simulador y autómata.

En cuanto al algoritmo implementado en el controlador se ha tomado como punto de partida la función de transferencia en discreto:

$$\frac{U(z)}{E(z)} = \frac{7.5539z - 6.5068}{z - 0.4764} = \frac{7.5539 - 6.5068z^{-1}}{1 - 0.4764z^{-1}}$$

$$U(z)(1 - 0.4764z^{-1}) = E(z)(7.5539 - 6.5068z^{-1})$$

$$U(z) - 0.4764 U(z - 1) = 7.5539 E(z) - 6.5098 E(z - 1)$$

$$U(z) = 7.5539 E(z) - 6.5098 E(z - 1) + 0.4764 U(z - 1)$$

Esta última expresión nos define la salida del controlador en un instante determinado. Nos damos cuenta de que es función de la entrada $E(z)$, de la entrada en el instante anterior $E(z - 1)$ y de la salida en el instante anterior $U(z - 1)$. Por tanto esta última expresión será la que implementaremos en nuestro algoritmo de control teniendo en cuenta que los parámetros que tiene que recibir nuestra función son la entrada $E(z)$, la entrada en el instante anterior $E(z - 1)$ y la salida en el instante anterior $U(z - 1)$. Todos estos valores se irán actualizando cada 0.05 segundos, ya que ésta es la frecuencia a la que va a trabajar nuestro controlador.

4.2.2 Ralentización del simulador para la correcta ejecución en tiempo real

En muchas simulaciones se utiliza la ralentización, o a veces la aceleración, para el correcto funcionamiento de la simulación en tiempo real. En nuestro caso, al estar trabajando con un *pseudo real-time*, no se trata de un tiempo real exacto, de manera que ralentizar la simulación facilita la comunicación en tiempo real entre el simulador y el autómatas. Con esta ralentización conseguiremos que la simulación funcione más lenta pero trabajará en tiempo real y la sincronización será perfecta.

Para realizar una ralentización en nuestro simulador es necesario cambiar una serie de parámetros que afectarán a nuestra simulación. Al cambiar la referencia de tiempos la función de transferencia de nuestro sistema variará, al igual que la frecuencia con la que trabajará nuestro controlador.

Para ello se realiza un cambio de variable donde la variable t representa el tiempo real y la variable t' representa la variable en el modelo ralentizado. Dependiendo de cuánto deseemos ralentizar nuestro modelo la relación entre ambas variables variará. Una ralentización bastante grande del modelo se puede considerar una relación de $t' = 100 t$. Si en cambio utilizamos una relación $t' = 10 t$ nuestro sistema se ralentizará bastante menos.

Tal y como se ha mencionado antes, estos cambios de variables implican cambios en el modelo. A continuación se desglosan los cambios que hay que realizar en nuestro modelo una vez realizado el cambio de variables. En este caso se realiza el cambio de variable para el caso $dt' = 100dt$.

En primer lugar se realizan los cambios en la función de transferencia de nuestro sistema:

$$\frac{10}{s(s+1)} = \frac{Y}{U}$$

$$\frac{d^2Y}{dt^2} + \frac{dY}{dt} = 10 U$$

$$\frac{d^2Y}{dt'^2} 100^2 + \frac{dY}{dt'} 100 = 10 U$$

$$(100^2 s^2 + 100s)Y = 10 U$$

$$\frac{10 U}{100(100s^2 + s)Y} = \frac{0.1 U}{(100s^2 + s)Y}$$

Una vez cambiada y corregida nuestra función de transferencia hay que cambiar la frecuencia con la que nuestro controlador actualizará los valores:

$$\Delta t = 0.05s$$

$$\Delta t' = 100 * 0.05 = 5s$$

Realizando dichos cambios en el sistema y en el controlador conseguiremos ralentizar nuestro simulador 100 veces, facilitando de esta manera el correcto funcionamiento del tiempo real así como la sincronización perfecta entre el simulador y el controlador. Para más detalles se puede consultar el algoritmo completo en el Anexo I: Código fuente del algoritmo de control.

4.2.3 Simulaciones a distintas velocidades

Para el testeo del correcto funcionamiento de nuestro controlador se han probado distintas velocidades en nuestro simulador con distintos resultados. En concreto se han realizado cinco distintas simulaciones con las velocidades y resultados indicados a continuación:

1. Simulación para $t' = 100 t$

Se trata de la simulación más ralentizada que se ha realizado. Aplicando un procedimiento análogo al utilizado en el apartado Ralentización del simulador para la correcta ejecución en tiempo real, la función de transferencia que se obtiene es la siguiente:

$$\frac{0.1 U}{(100s^2 + s)Y}$$

La frecuencia a la que trabajará el controlador, en cambio, se define como:

$$\Delta t = 0.05s$$

$$\Delta t' = 100 * 0.05 = 5s$$

Una vez obtenidas la función de transferencia y la frecuencia de trabajo se procede a realizar los cambios pertinentes en nuestro modelo y el algoritmo de control. Una vez ejecutada nuestra simulación se obtiene la siguiente respuesta para una entrada escalón de amplitud 25:

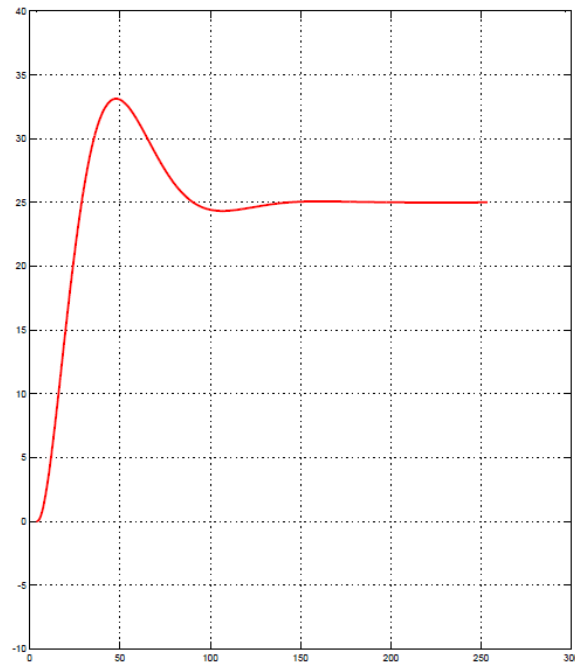


Figura 23 Simulación con $t'=100t$

2. Simulación para $t' = 50 t$

La segunda simulación realizada ha sido el doble de rápido que la primera, tal y como se muestra en la relación de variables. Si en este caso aplicamos el procedimiento descrito en el apartado Ralentización del simulador para la correcta ejecución en tiempo real, llegamos a la siguiente función de transferencia:

$$\frac{0.2 U}{(50s^2 + s)Y}$$

La frecuencia de trabajo del controlador en este caso será:

$$\Delta t = 0.05s$$

$$\Delta t' = 50 * 0.05 = 2.5s$$

Una vez obtenidas la función de transferencia y la frecuencia de trabajo se realizan los cambios necesarios en nuestro modelo y el algoritmo de control. Una vez ejecutada la simulación se obtiene la siguiente respuesta para una entrada escalón de amplitud 25:

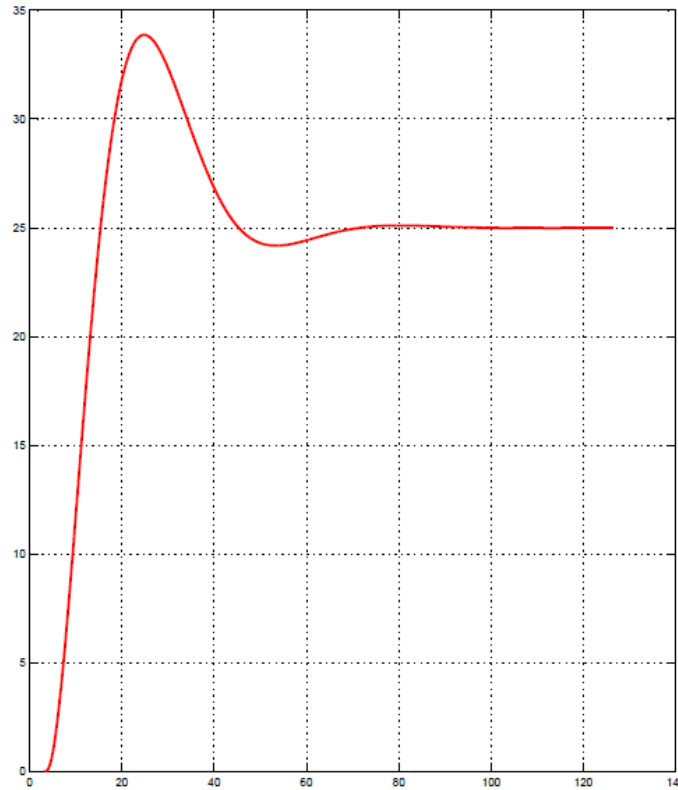


Figura 24 Simulación con $t'=50t$

3. Simulación para $t' = 20 t$

En este caso se ha optado por una simulación que nos proporcione una frecuencia de trabajo igual a un segundo. Su correspondiente relación de variables es $t' = 20 t$. Con el procedimiento descrito en el apartado Ralentización del simulador para la correcta ejecución en tiempo real, se obtiene la siguiente función de transferencia del sistema:

$$\frac{0.5 U}{(20s^2 + s)Y}$$

Tal y como se ha resaltado anteriormente, en este caso la frecuencia a la que trabajará el controlador será de 1 segundo:

$$\Delta t = 0.05s$$

$$\Delta t' = 20 * 0.05 = 1s$$

Definidas la función de transferencia y la frecuencia a la que trabajará el controlador, se realizan los cambios necesarios al igual que en las simulaciones anteriores. Una vez ejecutada la simulación se obtiene la siguiente respuesta para una entrada escalón de amplitud 25:

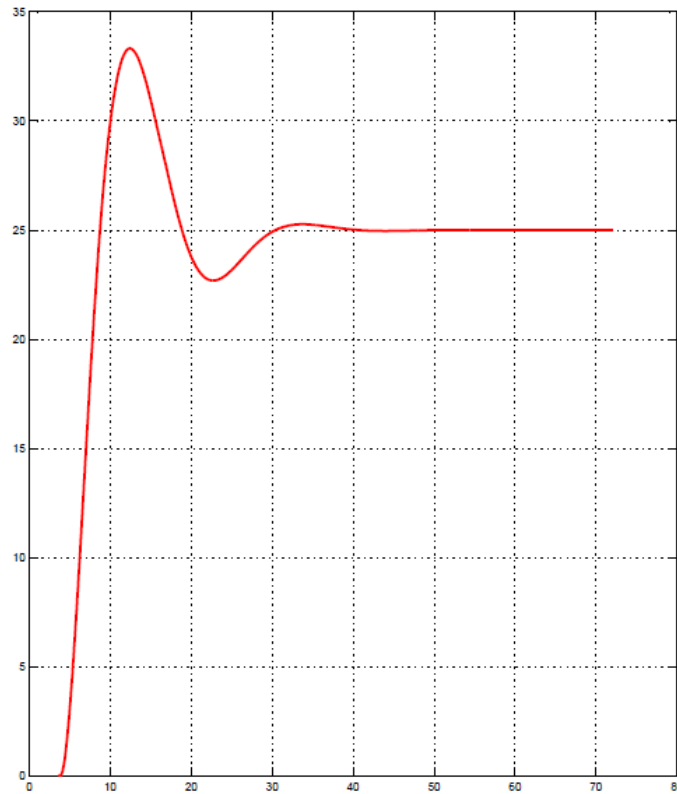


Figura 25 Simulación con $t'=20t$

4. Simulación para $t' = 10 t$

A medida que se reduce la relación entre las variables real e imaginaria, se obtiene una simulación más rápida de nuestro sistema. Con un procedimiento análogo al del apartado Ralentización del simulador para la correcta ejecución en tiempo real, conseguimos la función de transferencia que se muestra a continuación:

$$\frac{U}{(10s^2 + s)Y}$$

Este cambio de variables nos proporciona la siguiente frecuencia de trabajo:

$$\Delta t = 0.05s$$

$$\Delta t' = 10 * 0.05 = 0.5s$$

Estos cambios en nuestro modelo nos llevan a los siguientes resultados para una entrada escalón de amplitud 25:

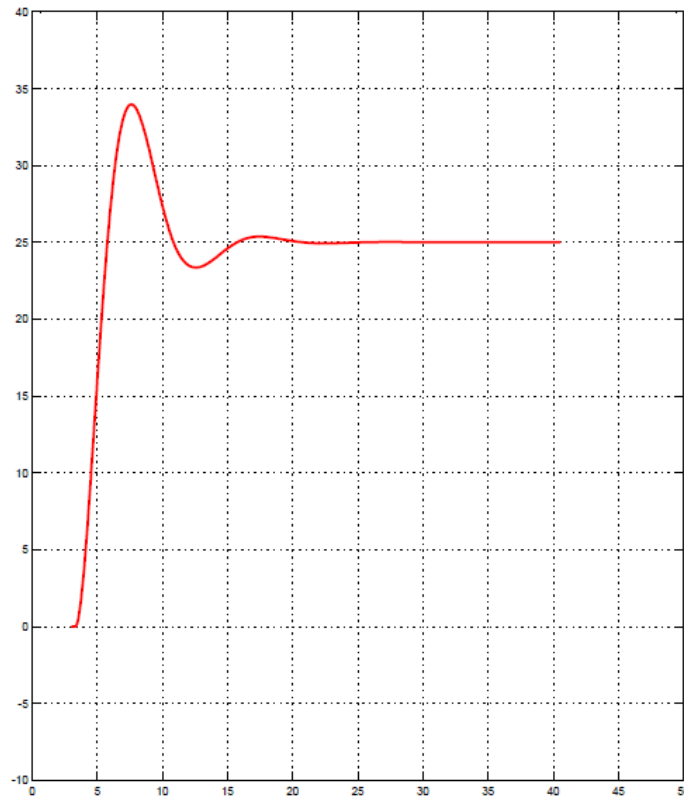


Figura 26 Simulación con $t'=10t$

5. Simulación para $t' = 2t$

Esta última simulación ha sido la simulación más rápida que se ha realizado. Cuanto más rápida sea la simulación, peor es la sincronización entre nuestro simulador y el controlador. Esto queda de manifiesto en los resultados obtenidos. De la misma manera que en el apartado Ralentización del simulador para la correcta ejecución en tiempo real, se obtiene la siguiente función de transferencia:

$$\frac{5U}{(2s^2 + s)Y}$$

En este caso la frecuencia de trabajo del controlador se define como:

$$\Delta t = 0.05s$$

$$\Delta t' = 2 * 0.05 = 0.1s$$

Definidas la función de transferencia y la frecuencia a la que trabajará el controlador, se realizan los cambios necesarios al igual que en las simulaciones anteriores. Una vez ejecutada la simulación se obtiene la siguiente respuesta para una entrada escalón de amplitud 25:



Figura 27 Simulación con $t'=2t$

Como se puede observar claramente en la Figura 27, en esta última simulación el controlador no funciona adecuadamente. Nuestro sistema pasa de ser un sistema controlado a uno oscilatorio. Tal y como se ha explicado anteriormente era necesario ralentizar nuestro modelo para su correcto funcionamiento. En esta frecuencia el controlador deja de controlar nuestro sistema llevándonos a la conclusión de que esta velocidad es demasiado rápida para nuestra comunicación.

5. Conclusiones

5.1 Comparación entre los resultados de las distintas simulaciones ralentizadas realizadas

Es muy frecuente realizar ralentizaciones cuando estamos ante simulaciones en tiempo real y con un *hardware* al mismo tiempo. De la misma manera que se ha mencionado anteriormente, estas ralentizaciones se utilizan para sincronizar de la mejor manera posible un *hardware* que opera en tiempo real con un simulador que se pretende ejecutar en tiempo real. Cuanto mayor sea la ralentización mejor será la sincronización en tiempo real y más cerca estará nuestro modelo de parecerse al modelo teórico.

Observando las dos primeras simulaciones realizadas se puede observar que ambas se parecen bastante. Las dos simulaciones tienen un primer sobreimpulso muy parecido, cercano al valor 34 en la amplitud aunque es verdad que la primera simulación (la más ralentizada) tiene algo menos. Después reducen su valor algo por debajo del objetivo y, por último, vuelven a incrementar su valor para acercarse al objetivo. En general se parecen bastante pero es verdad que a medida que se acelera el sistema los picos están algo más descontrolados y éstos se distancian algo más del objetivo (amplitud igual a 25 en nuestro caso).

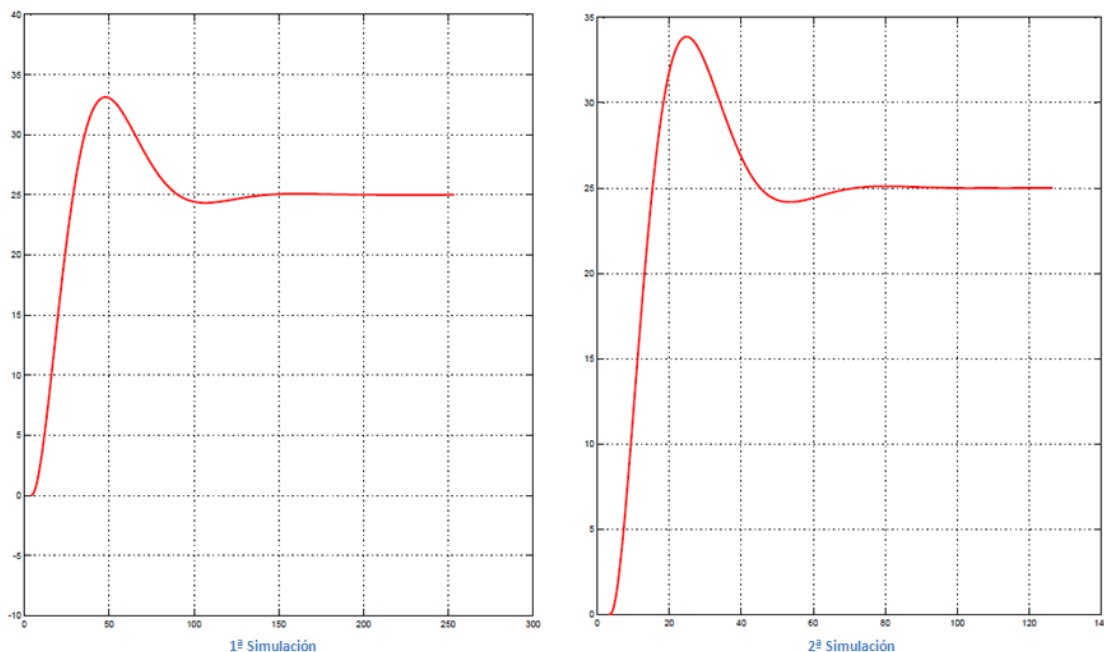


Figura 28 Comparación entre las dos primeras simulaciones

En cuanto al tiempo que necesita el sistema para controlarse queda muy claro que la relación entre una simulación y la siguiente es la mitad. Por tanto si de la primera

simulación a la segunda hemos ralentizado a la mitad (acelerarla al doble de velocidad que la anterior) observamos que el primer pico se da a los 50 segundos en la primera simulación. En la segunda simulación, en cambio, ese primer pico se da después de 25 segundos. Con la estabilidad final sucede algo parecido, en la primera simulación el sistema se estabiliza totalmente en el segundo 200, mientras que en la segunda, podemos considerar que dicha estabilidad se consigue en el segundo 100 (tal y como muestra la Figura 28).

La situación empeora algo con las siguientes simulaciones. En las dos siguientes simulaciones (la tercera y la cuarta) el primer sobreimpulso es parecido a las simulaciones anteriores, cercano a 34. Pero los siguientes picos son más pronunciados, se distancian más del objetivo e incluso aparece otro máximo por encima del objetivo. De esta manera le cuesta más estabilizarse pero al ser sistemas más acelerados que los anteriores la estabilidad se consigue en un menor tiempo. Realizan más oscilaciones pero en un tiempo menor. En la tercera simulación, por ejemplo, el primer pico se consigue a los 13 segundos aproximadamente y la estabilidad, en cambio, a los 42 segundos. Estos tiempos son bastantes más rápidos que los de las simulaciones anteriores, pero durante ese periodo de tiempo el sistema oscila más de lo que hacía en las simulaciones anteriores. Algo parecido sucede con la cuarta simulación, ya que el primer sobreimpulso se da después de 8 segundos y la estabilidad se consigue para los 22 segundos.

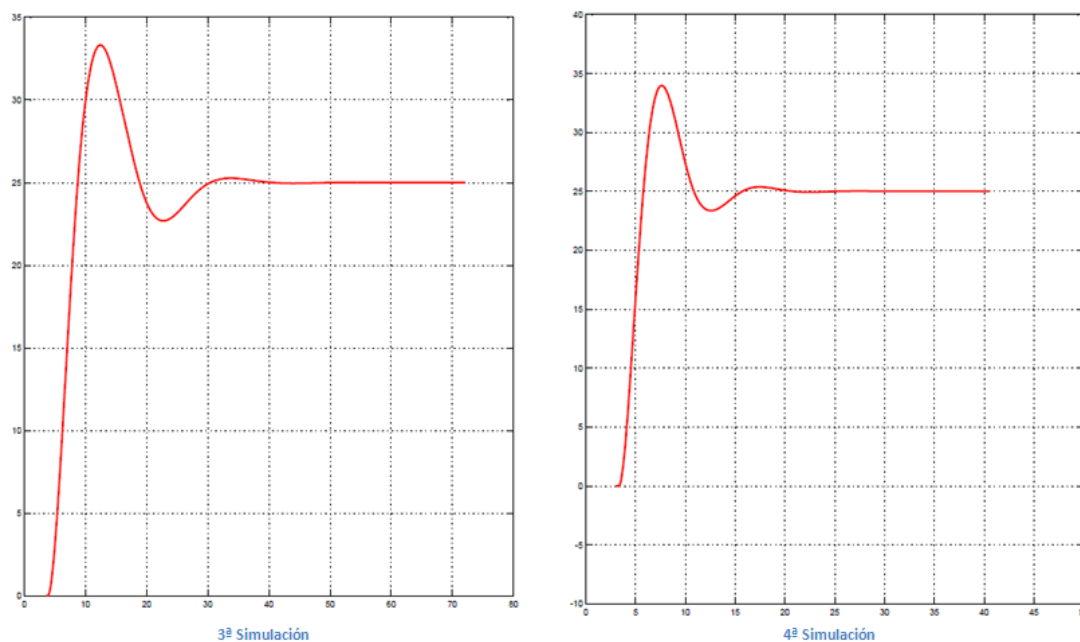


Figura 29 Comparación entre la tercera y cuarta simulación

El último caso es totalmente crítico, ya que nuestro controlador es incapaz de controlar el sistema. Con esta velocidad impuesta nuestro sistema se convierte en un sistema inestable y totalmente aleatorio, está todo el tiempo oscilando libremente como se puede observar en la Figura 27. Por tanto, se llega a la conclusión de que una

ralentización es necesaria para que nuestro sistema funcione correctamente, pero hay que tener cuidado con esa ralentización, ya que si es muy pequeña como el caso $t' = 2 t$ nuestro sistema se vuelve inestable. Deberíamos considerar al menos una relación de ralentización de $t' = 10 t$ para que nuestro sistema funcione de manera correcta.

Otro detalle importante que nos lleva a la conclusión de que la velocidad de ralentización es importante es la varianza que hay entre las cuatro primeras simulaciones. A medida que se incrementa la velocidad de simulación, nuestro controlador tiene mayores dificultades para controlar el sistema. Así, si se desea obtener unos resultados buenos y certeros, es conveniente ralentizar la simulación lo máximo posible. Cuanto más ralenticemos nuestro sistema, más se parecerán nuestros valores obtenidos a los teóricos.

Como ya se ha comentado a lo largo de este estudio, la sincronización entre el simulador y el *hardware* no es algo sencillo. Muchas veces se puede llegar a sistemas inestables si falla la sincronización. Por ello, es muy importante recalcar la necesidad de ralentizar el sistema para que la sincronización sea la mejor posible y obtener resultados fiables y acordes con el modelo teórico.

5.2 Comparación de los resultados obtenidos con el modelo teórico

Atendiendo a los resultados obtenidos en el apartado Simulación teórica se puede apreciar claramente que el tiempo empleado por el modelo teórico es mucho menor al empleado por el resto de simulaciones. Obviamente esto se debe a la ralentización impuesta al resto de simulaciones. Pero incluso teniendo en cuenta dicha ralentización el modelo teórico se resuelve bastante mejor que el modelo con *"hardware-in-the-loop"*.

Si se observan detenidamente los resultados del modelo teórico se puede apreciar que el primer sobreimpulso es parecido al obtenido en las simulaciones ralenticizadas (cercano a 34). Pero atendiendo al primer mínimo, este mínimo está mucho más cerca del objetivo en el modelo teórico que en el resto de simulaciones. Lo mismo sucede con los siguientes máximos y mínimos relativos de la función. Como se ha mencionado anteriormente, la relación de la escala de tiempos entre la simulación teórica y las simulaciones ralenticizadas es correcta. Por ejemplo, comparando la simulación teórica con la primera simulación realizada, nos daremos cuenta que en la simulación teórica el primer sobreimpulso se consigue en el segundo 0.5, mientras que en la primera simulación ralenticizada se consigue en el segundo 50. Teniendo en cuenta que la relación de ralentización es de $t' = 100 t$, son perfectamente correctos los resultados obtenidos.

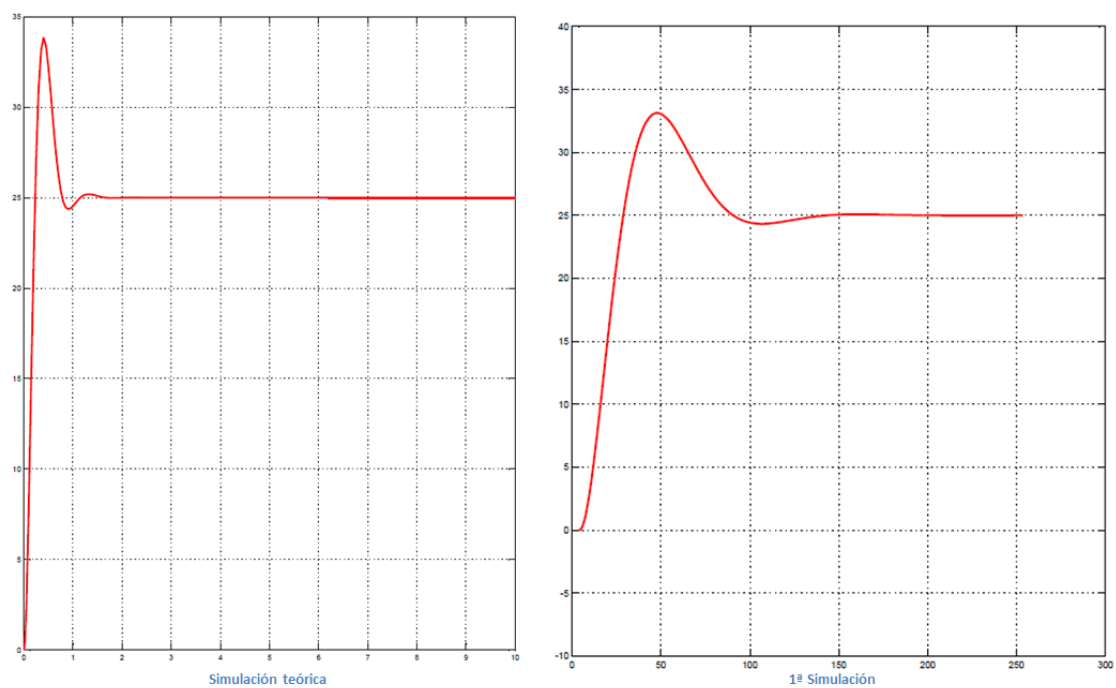


Figura 30 Comparación entre el modelo teórico y la primera simulación

6. Bibliografía

- (1) <http://www.monografias.com/trabajos6/sipro/sipro.shtml>
- (2) Ogata, Katsuhiko (2003). *Ingeniería de Control Moderna*. Madrid: PEARSON EDUCACIÓN, S.A.
- (3) <http://bibing.us.es/proyectos/abreproy/11477>
- (4) <https://opcfoundation.org/>
- (5) *CX-Server OPC User Manual, Getting Started Version 2.0*
- (6) *Matlab OPC Toolbox user's guide*
- (7) http://www.mathworks.es/company/?s_tid=hp_ff_about
- (8) <http://industrial.omron.es/es/company-info/about-omron>
- (9) *CX-Server Lite User Manual, Getting Started Version 2.1*
- (10) https://www.ia.omron.com/data_pdf/cat/cx-compolet_v302-e1_6_1_csm1000258.pdf?id=63
- (11) <http://blog.vermiip.es/2008/03/11/que-es-el-numero-ip-que-significa-ip/>
- (12) <http://www.tknika.net/liferay/es/web/materialeak/6.-opc-ethernet-omron>
- (13) <http://windows.microsoft.com/es-xl/windows/change-tcp-ip-settings#1TC=windows-7>
- (14) <http://www.youtube.com/watch?v=ZldRxu4dBFo>
- (15) *Matlab Real-Time Windows Target user's guide*
- (16) <http://www.mathworks.com/matlabcentral/answers/68726-using-simulink-coder-to-build-models-using-opc-toolbox-blocks-tlc-files-missing>
- (17) De la Rosa Galván, Hernán (2012). *Implementación de un sistema SCADA para la mezcla de dos sustancias en una industria química*. Instituto Politécnico Nacional

Anexo I. Código fuente del algoritmo de control

