Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de la Telecomunicación

Aplicación para el control de una carga programable para una máquina de inducción usando un PLC.

Autor: Javier Becerra Pérez

Tutores: Daniel Rodríguez Ramírez

Manuel Garrido Satue

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación para el control de una carga programable para una máquina de inducción usando un PLC.

Autor:
Javier Becerra Pérez

Tutores:
Daniel Rodríguez Ramírez
Profesor titular
Manuel Garrido Satue
Profesor sustituto interino

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo Fin de Grado: Aplicación para el control de una carga programable para una máquina de inducción usando un PLC.

Autor: Javier Becerra Pérez

Tutores: Daniel Rodríguez Ramírez
Manuel Garrido Satue

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi familia

A mis profesores

Agradecimientos

A mis padres, Antonio y Carmen, por su infatigable afán de darnos a sus hijos la mejor educación personal y académica. Por brindarme la oportunidad de hacer esto posible con vuestro esfuerzo y por vuestra infinita paciencia. Sin vosotros esto no hubiera sido posible.

A mi hermana Irene, por permitirme aprender de su experiencia para que mi camino fuese más ameno.

A mi familia, por el constante apoyo que he recibido siempre.

A mis amigos y compañeros, por el apoyo, por las largas jornadas de estudio, por las risas, por los buenos ratos que hemos pasado y por los no tan buenos. En especial a Ángel, Pablo y Edgar, por ser uno de los pilares fundamentales y a todos mis compañeros de la Escuela.

A Kivi.

Gracias a todos.

Javier Becerra Pérez

Alumno del grado de Ingeniería de las Tecnologías de Telecomunicación

Sevilla, 2022

Resumen

El objeto de este proyecto es la creación y configuración de una herramienta digital basada en *MATLAB* y sus librerías de comunicaciones industriales *OPC DA* para el control de un controlador lógico programable *SIEMENS S7-300* para la generación de cargas programadas para una máquina de inducción del laboratorio de *Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla*.

Abstract

The purpose of this project is the creation and configuration of a digital tool based on *MATLAB* and its *OPC DA* industrial communications libraries for the control of a *SIEMENS S7-300* programmable logic controller for the generation of programmed loads for an induction machine of the *Laboratory of Systems and Automation Engineering of the School of Engineering of the University of Seville.*.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Cuando partas hacia Ítaca pide que tu camino sea largo
y rico en aventuras y conocimiento.

K. Kavafis, «Ítaca»

Con el creciente desarrollo de la llamada *Industria 4.0*, la necesidad de crear sistemas industriales conectados ha demandado el desarrollo de tecnologías de comunicación que converjan los campos IT-OT en las industrias, entendiéndose que todas las máquinas de campo necesitan estar conectadas a sistemas de adquisición de datos que faciliten su diagnóstico, compartan información sobre los procesos y permitan recolectar estos datos para su posterior tratamiento.

El objeto de este proyecto es, haciendo uso del estándar de comunicaciones industriales *OPC (OLE for Process Control)*, crear una plataforma de control para un controlador lógico programable S7-300 de SIEMENS sobre el entorno de creación de aplicaciones de MATLAB (AppDesigner). Esta aplicación sirve como interfaz hombre-máquina para realizar cambios de referencia sobre un motor de corriente continua encargado de generar cargas para un motor de inducción, oponiéndose a su giro.

La aplicación consta de varios modos de funcionamiento que se detallarán en capítulos posteriores, así como todos los componentes del sistema, su guía de desarrollo e instalación, su puesta en marcha y, finalmente, el uso de la aplicación.

2 ARQUITECTURA DEL SISTEMA

No es el conocimiento, sino el acto de aprendizaje; y no la posesión,
sino el acto de llegar a ella, lo que concede el mayor disfrute.

Carl Friedrich Gauss.

Antes de comenzar a definir la arquitectura del sistema máquina-aplicación, es necesario asentar la base teórica sobre la que se sustenta e introducir al lector en las tecnologías utilizadas. A continuación, se desarrollarán estos conceptos y después, entraremos en mayor profundidad.

2.1. Descripción general del sistema

A continuación, se describe la arquitectura del sistema completo formado desde los dispositivos de campo hasta la aplicación de usuario.

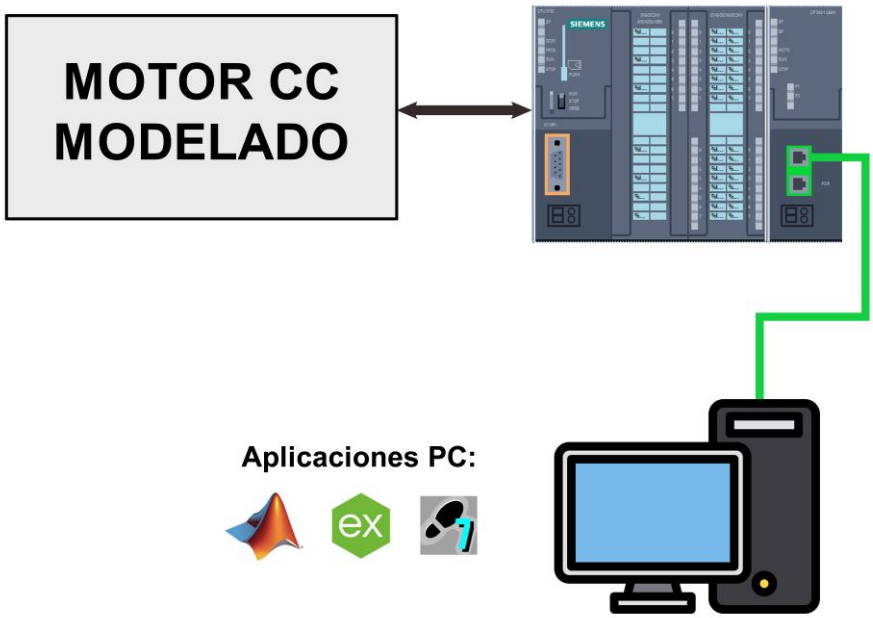


Figura 2-1. Diagrama del sistema.

En la **Figura 2-1** se observa el diagrama del sistema completo, formado por:

- **MOTOR CC MODELADO**: Este es el modelo discretizado de un motor de corriente continua que será ejecutado por un segundo controlador lógico programable (en adelante, PLC 2). El motor de corriente continua se ejecuta en un segundo PLC que simula su comportamiento.

- **CPU S7-313C-5BG04-0AB0:** Es el PLC encargado de realizar el control sobre el motor de corriente continua y estará conectado por su tarjeta de entradas/salidas analógicas al PLC simulador del motor de corriente continua. A su vez, haciendo uso del BUS-K (bus interno sobre el rack), el PLC está conectado a una tarjeta de comunicaciones CP-343 1-Lean.

- **CP 343-1 Lean:** Tarjeta de comunicación que proporciona al PLC una interfaz de comunicaciones tipo Ethernet Industrial (TCP/IP) que permitirá la comunicación sobre ethernet con el PC.

- **PC:** Ordenador común con sistema operativo Windows 10. Debe tener instalado MATLAB, KEPServer y STEP 7.

2.2. Descripción hardware

2.2.2 PLC SIEMENS CPU S7-313C-5BG04-0AB0

La CPU S7-313C-5BG04-0AB0 ofrece las siguientes características:

Tabla 2-1. Características técnicas CPU 313C-5BG04

Información general	
Versión firmware	V3.3
Tensión de alimentación	
Valor nominal (DC)	24 V
Rango admisible, límite inferior (DC)	19,2 V
Rango admisible, límite superior (DC)	28,8 V
Tensión de carga L+	
Entradas digitales	24 entradas
• Valor nominal (DC)	24 V
• Protección contra inversión de polaridad.	Sí
Salidas digitales	16 salidas
• Valor nominal (DC)	24 V
• Protección contra inversión de polaridad.	No

Intensidad de entrada	
Consumo (valor nominal)	650 mA
Consumo (en marcha en vacío)	150 mA
Intensidad de cierre	5 A
Entradas digitales	24 entradas
• de la tensión de carga L+ (sin carga), máx.	80 mA
Salidas digitales	16 salidas
• de la tensión de carga L+, máx.	50 mA
Memoria de trabajo	128 kbytes
Direcciones predeterminadas de los canales integrados	
Entradas digitales	124.0 a 124.7
Salidas digitales	124.0 a 125.7
Entradas analógicas	752 a 761
Salidas analógicas	752 a 755

Figura 2-2. CPU SIEMENS 313-5BG04-0AB0

2.2.3 PLC SIEMENS CPU S7-313C-2 DP 6CF03-0AB0

La CPU S7-313C-2 DP 6CF03-0AB0 ofrece las siguientes características:

Tabla 2-2. Características técnicas CPU 313C-2 DP 6CF03-0AB0

Información general	
Versión firmware	V2.6
Tensión de alimentación	
Valor nominal (DC)	24 V
Rango admisible, límite inferior (DC)	20,4 V
Rango admisible, límite superior (DC)	28,8 V
Tensión de carga L+	
Entradas digitales	16 entradas
• Valor nominal (DC)	24 V
• Protección contra inversión de polaridad.	Sí
Salidas digitales	16 salidas
• Valor nominal (DC)	24 V
• Protección contra inversión de polaridad.	No

Figura 2-3. CPU SIEMENS 313-2 DP 6CF03-0AB0

Dado que el intercambio de información entre el PLC1 y PLC2 se da a través de señales analógicas, esta CPU deberá ir acompañada de dos módulos de expansion que se detallarán a continuación.

2.2.4 Módulo expansión SIEMENS SM331 – AI 8x12 BIT

El módulo de expansión SIEMENS SM331 – AI 8x12 BIT proporciona al PLC2 8 entradas analógicas con resolución de 12 bits.

Este módulo de expansión puede trabajar con señales de corriente o de tensión con las siguientes configuraciones:

Tabla 2-3. Configuración de entradas analógicas del módulo de expansión SM331

Configuración de entradas	
Rangos en tensión	
[0 V; +10V]	
• Impedancia de entrada	100 kΩ
[+1 V; +5 V]	
• Impedancia de entrada	100 kΩ
[-1 V; +1 V]	
• Impedancia de entrada	100 kΩ
[-5 V; +5 V]	
• Impedancia de entrada	100 kΩ
[-10 V; +10 V]	
• Impedancia de entrada	100 kΩ
[-50 mV; +50 mV]	
• Impedancia de entrada	100 kΩ
[-500 mV; +500 mV]	
• Impedancia de entrada	100 kΩ
Rangos en corriente	
[0 mA; +20 mA]	
• Impedancia de entrada	50 Ω
[-20 mA; +20 mA]	
• Impedancia de entrada	50 Ω
[+4 mA; +20 mA]	
• Impedancia de entrada	50 Ω

En próximos capítulos se mostrará la configuración de la tarjeta y cómo se lleva a cabo.

2.2.5 Módulo expansión SIEMENS SM332 – AO 4x12 BIT

El módulo de expansión SIEMENS SM332 – AI 4x12 BIT proporciona al PLC2 4 salidas analógicas con resolución de 12 bits.

Este módulo de expansión puede trabajar con señales de corriente o de tensión con las siguientes configuraciones:

Tabla 2-4. Configuración de salidas analógicas del módulo de expansión SM332

Configuración de salidas	
Rangos en tensión	
[0 V; +10V]	
• Impedancia de carga	1 kΩ
[+1 V; +5 V]	
• Impedancia de carga	1 kΩ
[-10 V; +10 V]	
• Impedancia de carga	1 kΩ
Rangos en corriente	
[0 mA; +20 mA]	
• Impedancia de carga	500 Ω
[-20 mA; +20 mA]	
• Impedancia de carga	500 Ω
[+4 mA; +20 mA]	
• Impedancia de carga	500 Ω

En próximos capítulos se mostrará la configuración de la tarjeta y cómo se lleva a cabo.

2.2.6 Procesador de comunicaciones CP 343-1 LEAN

El procesador de comunicaciones CP 343-1 Lean permite conectar sistemas de automatización SIMATIC S7-300 a Industrial Ethernet (TCP/IP) y soporta PROFINET IO[1] entre equipos SIEMENS.

A continuación se muestra una tabla con sus principales características:

Figura 2-4. Procesador de comunicaciones CP 343-1 Lean

Tabla 2-5. Datasheet del procesador de comunicaciones CP 343-1 Lean

Tensiones	
Tensión de alimentación	24 V
Tensión del bus	5 V
Interfaces de comunicación	
Número de puertos	2
Interfaz 1 – Industrial Ethernet/PROFINET	Puerto RJ45
Interfaz 2 – Industrial Ethernet/PROFINET	Puerto RJ45
Tasa de transmisión	10 … 100 Mbit/s

2.2.7 Fuente de alimentación 6EP1 331-1SL11 24V-2A

Para dar alimentación a todos los equipos del rack de automatización, se ha utilizado la siguiente fuente de alimentación, montada sobre el carril DIN.

Para este proyecto, se podría utilizar indistintamente cualquier otra fuente de alimentación de 24 V/2A.

[1] PROFINET IO (Process Field Network): Estándar de comunicación basado en Ethernet Industrial para el intercambio de datos de forma segura en tiempo real entre equipos en el mismo bus de campo. Fue estandarizado con respecto a las normas IEC 61158 e IEC 61784.

Figura 2-5. Fuente de alimentación 6EP1 331-1SL11 24V – 2A

2.3. Descripción del software

Para este proyecto será necesario utilizar los siguientes paquetes de software.

2.3.2 Siemens STEP 7

STEP 7 es un software de programación de controladores lógicos programables (PLC) de la gama SIMATIC-S7 300/400.

En este proyecto, este software ha sido utilizado para configurar ambos PLCs, sus módulos de expansión y su procesador de comunicaciones, así como todas las funciones implementadas para el control y simulación del motor de corriente continua.

Figura 2-6. Icono del software STEP 7.

2.3.3 Kepware KEPServerEX v6.4

KepServerEX es el software encargado de la comunicación entre el PLC1 y la tarjeta de red del PC a través de sus drivers de comunicaciones sobre TCP/IP y aloja el servidor OPC para nuestra aplicación.

En este software será necesario crear todos los tags OPC que vayamos a leer desde el cliente OPC.

Figura 2-7. Icono del software KEPServerEX.

2.3.4 MATLAB App Designer 2020b

La aplicación de control, configuración y adquisición de datos de este proyecto ha sido diseñada utilizando *App Designer*.

App Designer permite crear aplicaciones con interfaz de usuario programada con código orientado a objetos que es ejecutado sobre un entorno *MATLAB*, brindándo la posibilidad de crear una aplicación independiente con *MATLAB Compiler* y *Simulink Compiler*.

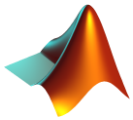


Figura 2-8. Icono del software MATLAB.

2.4. Conexionado del hardware

A continuación se detallan todas las conexiones del montaje completo:

Los componentes del sistema anteriormente mencionados en el apartado 2.2 han sido montados sobre dos carriles DIN[2].

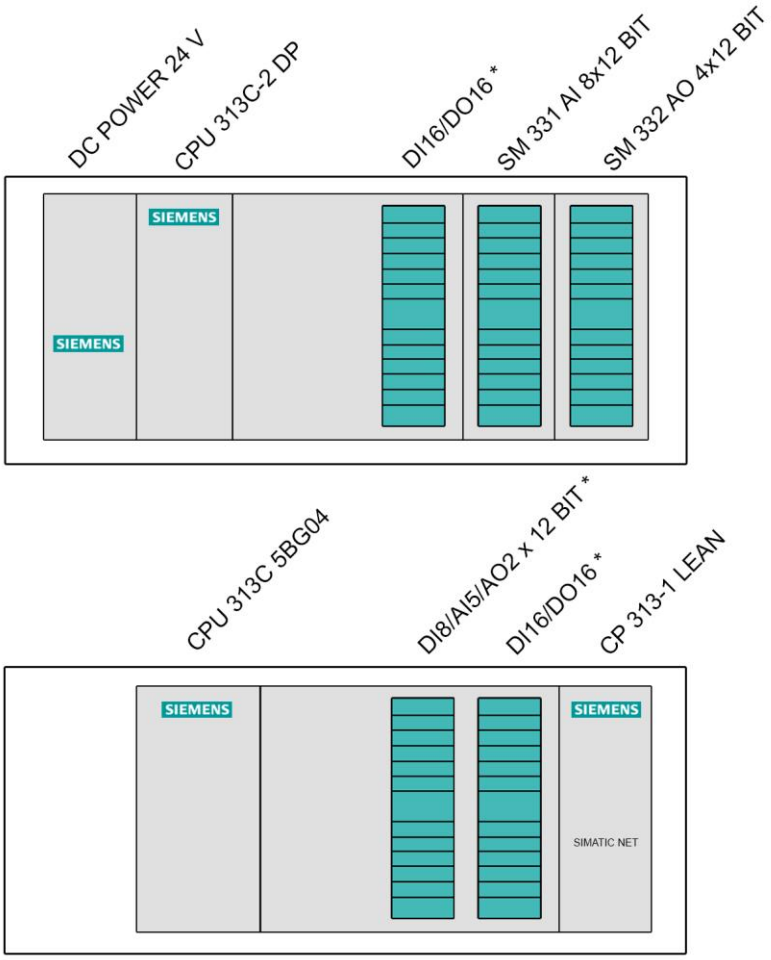


Figura 2-9. Diagrama de montaje sobre los carriles DIN.

*Tarjeta de E/S incorporada en la CPU.

[2] Carril DIN: Perfil metálico normalizada con dimensiones 35 mm x 7.5 mm (EN 50022, BS 5584, DIN 46277-3, NFC 63015, DIN 3) comúnmente utilizado para el montaje de elementos eléctricos de protección y mando en armarios eléctricos industriales y viviendas.

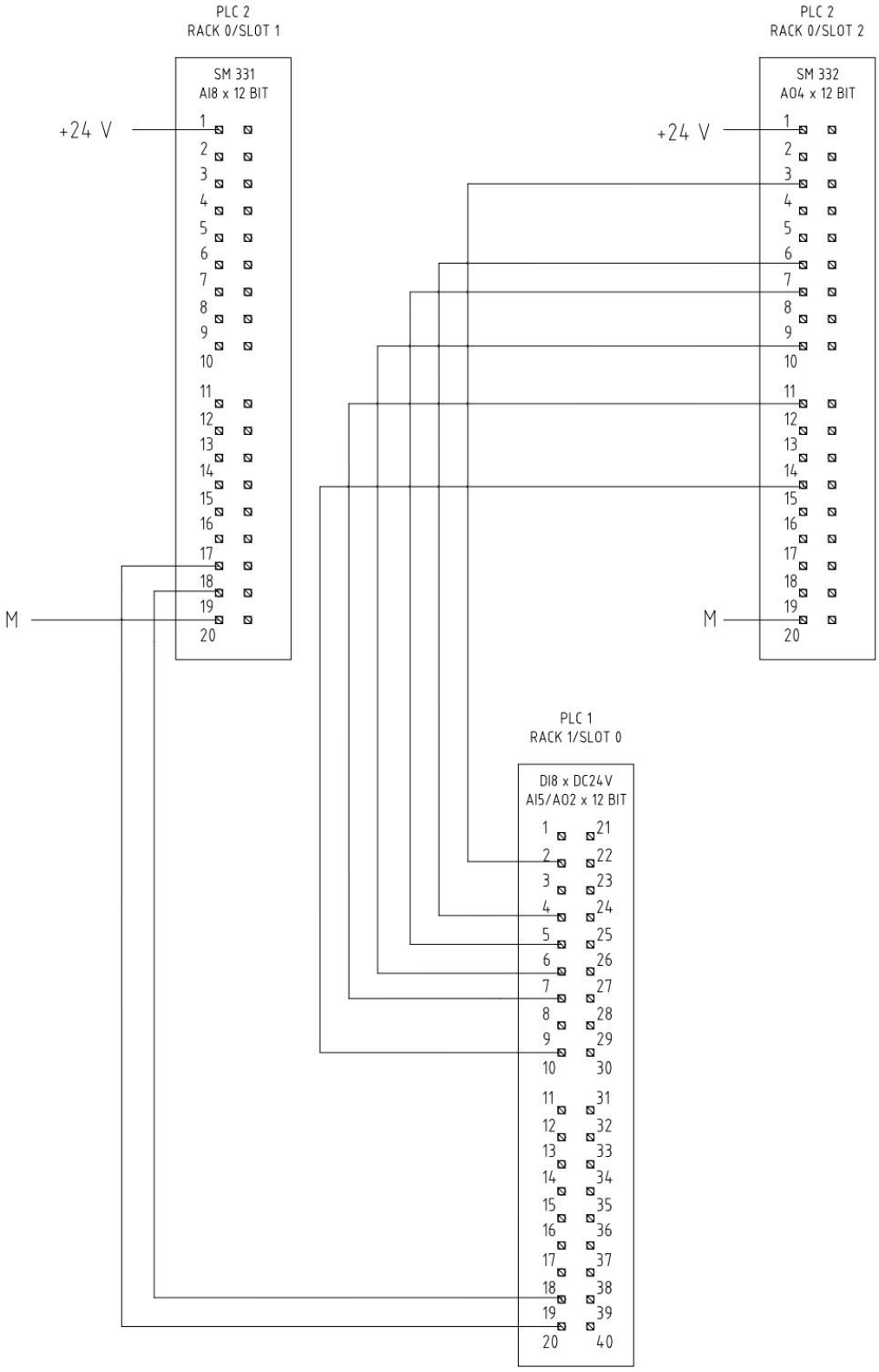


Figura 2-10. Esquema eléctrico de conexión de los módulos de E/S.

Como se puede observar en la **Figura 2-10**, los módulos de expansión SM331 y SM332 necesitan ser alimentados externamente.

En esta parte del proyecto, hemos de hacer especial incapié en que un fallo a la hora de realizar este cableado puede llevar a la inutilización de cualquiera de los módulos de expansión de E/S o incluso, la CPU.

Con esto tendremos conectados el controlador con el modelo simulado del motor de corriente continua. A continuación, conectaremos un cable ethernet desde el puerto 1 del procesador de comunicaciones SIMATIC NET hasta la tarjeta de red del PC donde se alojará el servidor OPC y la aplicación de control y configuración.

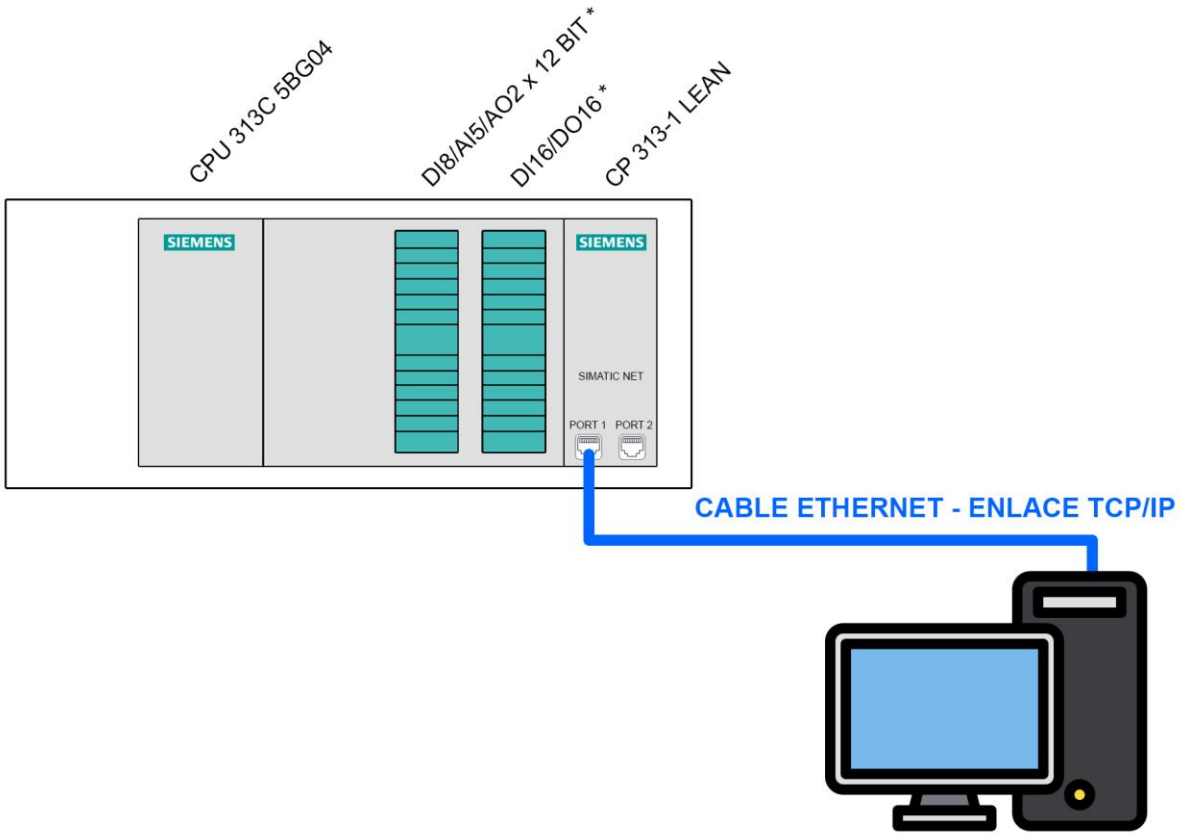


Figura 2-11. Diagrama de conexión SIMATIC NET - PC.

Con esto habremos completado la configuración hardware del sistema.

En el próximo capítulo de este proyecto se detallará la configuración de ambos PLCs, sus módulos de expansión de E/S, procesador de comunicaciones y el servidor OPC.

3 CONFIGURACIÓN DEL HARDWARE

No quiero creer, quiero saber.
Carl Sagan

Este capítulo está dedicado a explicar detalladamente cómo se ha llevado a cabo la configuración del hardware descrito anteriormente, así como la configuración del servidor OPC de KEPServerEX.

3.2. Configuración de equipos SIEMENS con STEP7.

En primera instancia, iniciaremos STEP7 (Simatic Manager) y crearemos un nuevo proyecto. Para ello, una vez iniciada la aplicación, nos dirigiremos a la pestaña *Archivo -> Nuevo.*

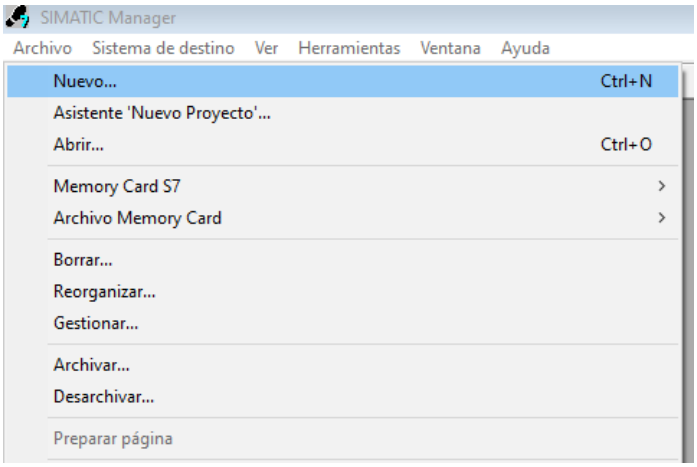


Figura 3-1. Ventana de creación de nuevo proyecto en STEP7.

Esta acción nos abrirá una nueva ventana *Nuevo Proyecto*, donde tendremos que indicar la ubicación (ruta) donde se alojará nuestros archivos de proyecto y el nombre que le queramos poner.

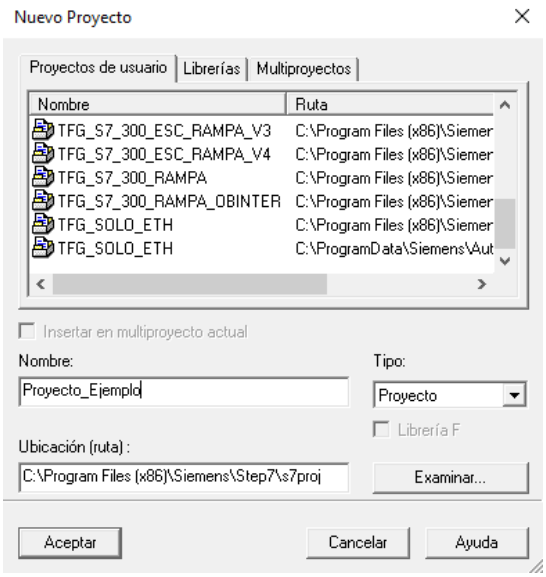


Figura 3-2. Ventana de proyectos de usuario en STEP7.

Una vez completados los campos necesarios, pulsaremos el botón *Aceptar*.

Con esto, habremos creado nuestro proyecto y ya podemos comenzar a realizar la configuración del hardware.

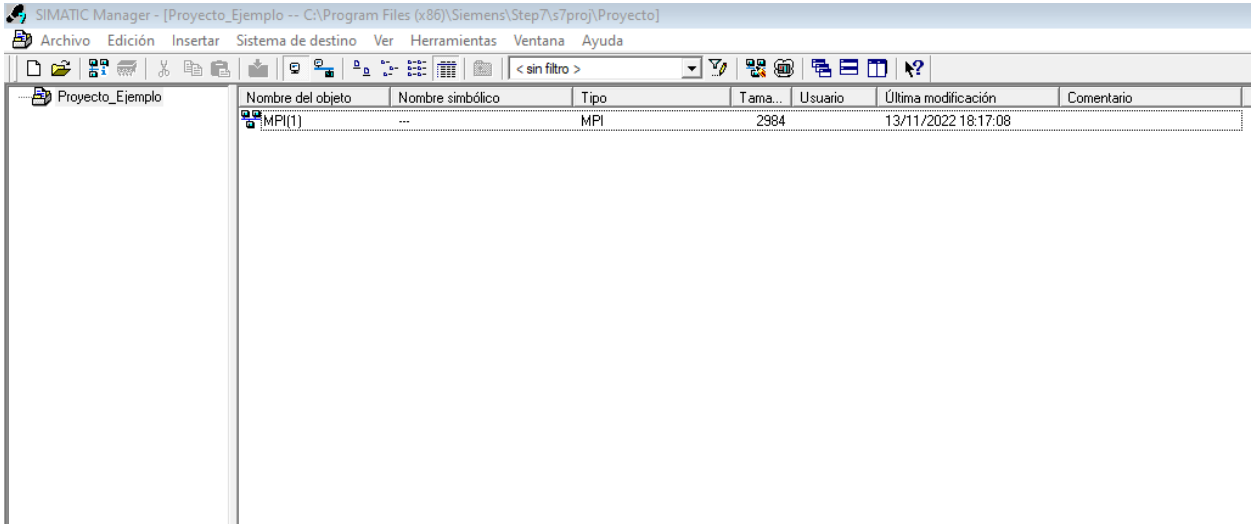


Figura 3-3. Ventana principal de proyecto en STEP7.

Para comenzar, tendremos que añadir en nuestro proyecto el hardware que vamos a utilizar. Para ello, haremos click derecho sobre *Proyecto_Ejemplo* y esto abrirá un menú desplegable con las siguientes opciones:

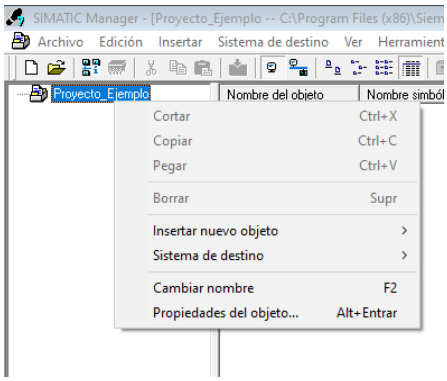


Figura 3-4. Menu de proyecto en STEP7.

Una vez desplegado el menú, haremos click en *Insertar nuevo objeto -> SIMATIC 300*. A continuación, observaremos que se ha añadido una nueva entrada con nombre *SIMATIC 300(1)* en la lista de objetos del proyecto.

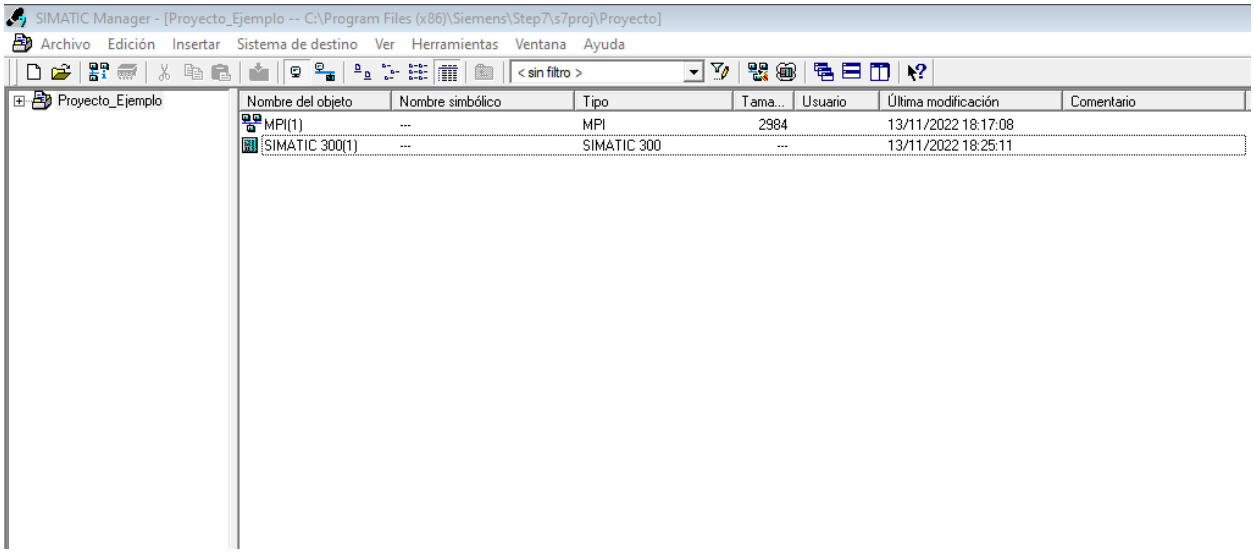


Figura 3-5. Vista componentes de proyecto en STEP7.

Para configurar el hardware del rack, haremos doble click sobre *SIMATIC 300(1)->Hardware* y se abrirá una nueva ventada de configuración hardware *HW Config*.

En esta nueva ventana añadiremos en primer lugar un *Perfil de soporte*, y posteriormente iremos añadiendo el PLC1 con todos sus módulos de expansión de E/S y el procesador de comunicaciones.

Para añadir el *Perfil de soporte* desplegaremos en el menú de la derecha, *SIMATIC 300 ->BASTIDOR 300 -> Perfil soporte* y bastará con arrastrar el elemento deseado hasta la sub-ventana en blanco.

Después de hacer esto, aparecerá una tabla enumerada correspondiente al *Perfil de soporte* y a cada slot disponible.

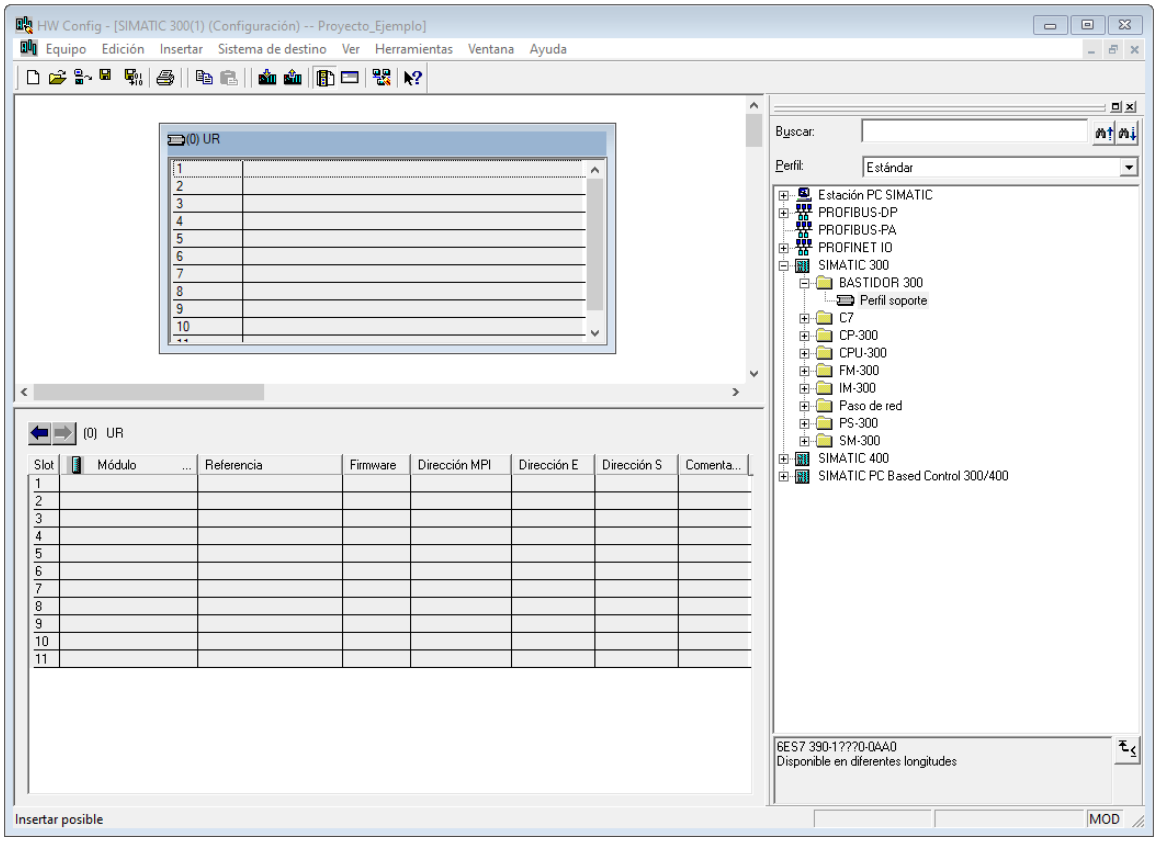


Figura 3-6. Ventana de configuración hardware en STEP7.

Una vez añadido el *Perfil de soporte,* iremos arrastrando en la posición correspondiente el PLC1 (*CPU-300 -> CPU 313C -> 6ES7 313-5BG04-0AB0 -> V3.3)* en el slot 2 (primer slot disponible para la CPU) y el procesador de comunicaciones (*CP-300 -> Industrial Ethernet -> CP 343-1 Lean -> 6GK7 343-1CX10-0XE0 -> V3.0*).

La posición que ocupe cada dispositivo en el perfil virtual debe corresponder con la posición (o *slot*) que ocupen físicamente.

Al arrastrar el procesador de comunicaciones CP 343-1 Lean (SIMATIC NET), aparecerá una nueva ventana de propiedades de la interfaz ethernet del módulo.

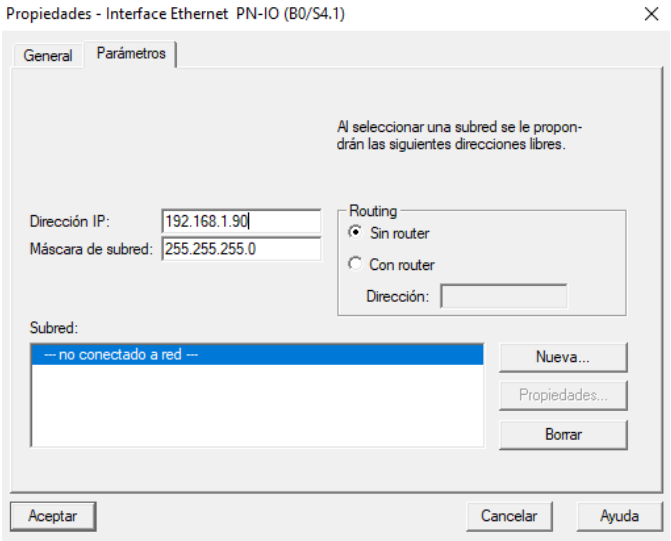


Figura 3-7. Ventana de propiedades del módulo CP-343 en STEP7.

En este caso, asignaremos a esta interfaz la dirección *IP: 192.168.1.90* . Antes de cerrar esta ventana, será necesario añadir la interfaz a una subred.

Crearemos una subred ethernet haciendo click en el botón *Nueva*, le asignaremos un nombre y cerraremos ambas ventanas pulsando *Aceptar*.

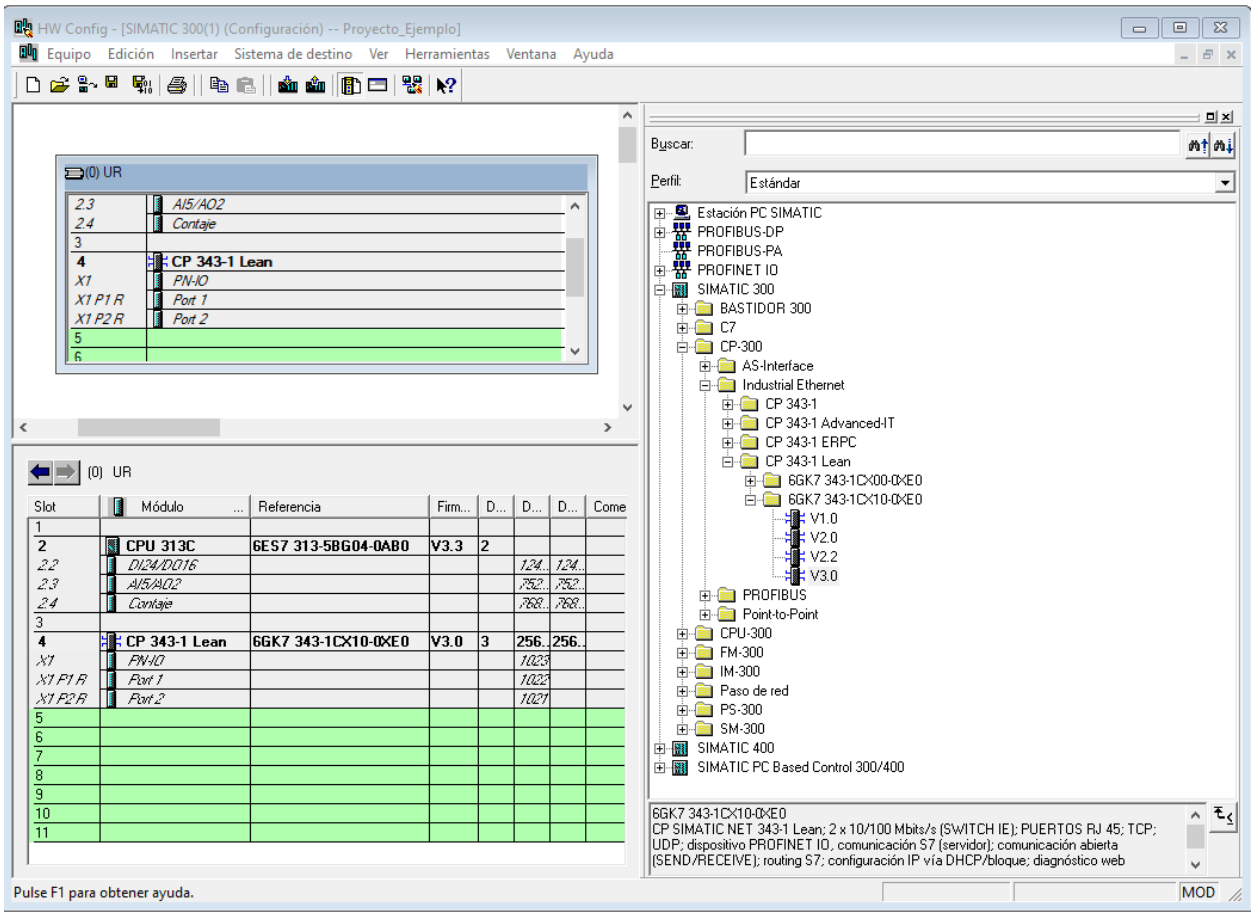


Figura 3-8. Ventana de configuración hardware en STEP7.

Una vez añadidos el PLC1 y el procesador de comunicaciones, configuraremos las entradas y salidas analógicas integradas en este PLC. Para ello, haremos doble click en la fila correspondiente al slot 2.3 y se abrirá una nueva ventana de configuración.

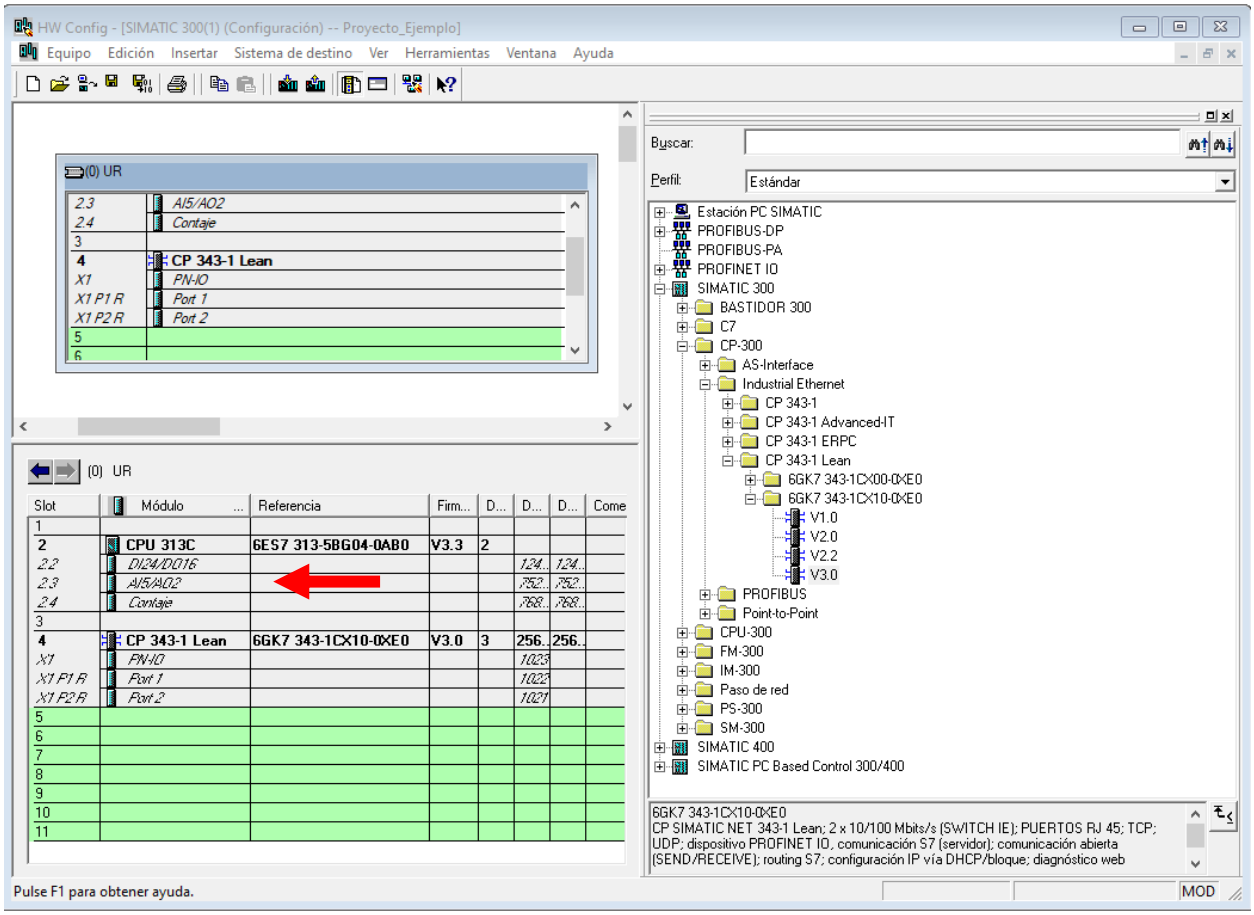


Figura 3-9. Ventana de configuración hardware en STEP7.

En esta nueva ventana podremos configurar el rango de direcciones que emplearemos para direccionar las entradas/salidas analógicas, el tipo de entrada a usar (tensión/corriente) y su rango de trabajo, y esta misma configuración para las salidas.

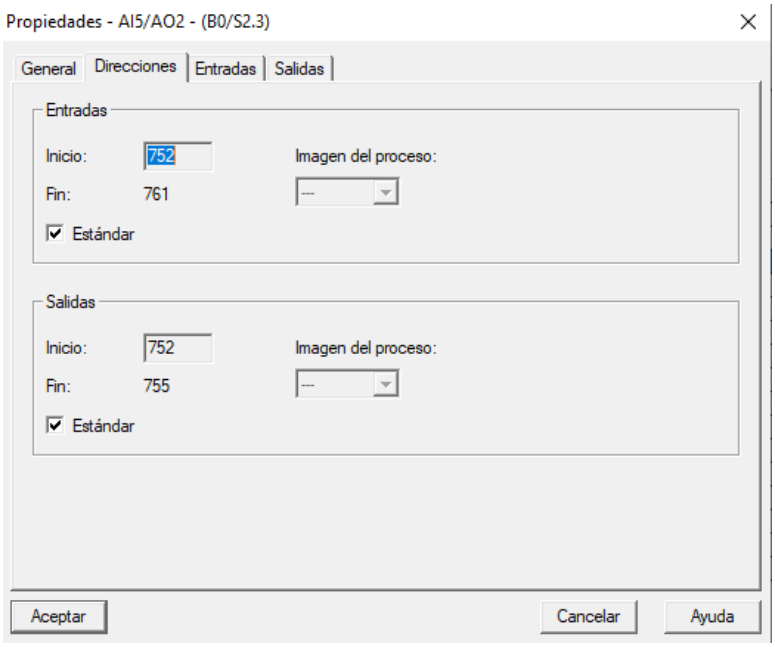


Figura 3-10. Ventana de propiedades de E/S analógicas en STEP7.

En nuestro caso, mantendremos las direcciones de las entradas y salidas analógicas con su configuración por defecto, es decir:

Las entradas/salidas analógicas serán accesibles desde el programa apuntando a las siguientes direcciones:

Tabla 3-1. Direccionamiento de entradas y salidas analógicas del PLC1.

Entradas	Canal
PIW752	CH 0
PIW754	CH 1
PIW756	CH 2
PIW758	CH 4
Salidas	**Canal**
PQW752	CH 0
PQW754	CH 1

De la misma forma que hemos realizado estas configuraciones, a continuación configuraremos el perfil con el PLC2 y sus módulos de expansión SM331 y SM332 de E/S analógicas. Para realizar esta configuración se ha creado un segundo proyecto de nombre SIM_CC, resultando:

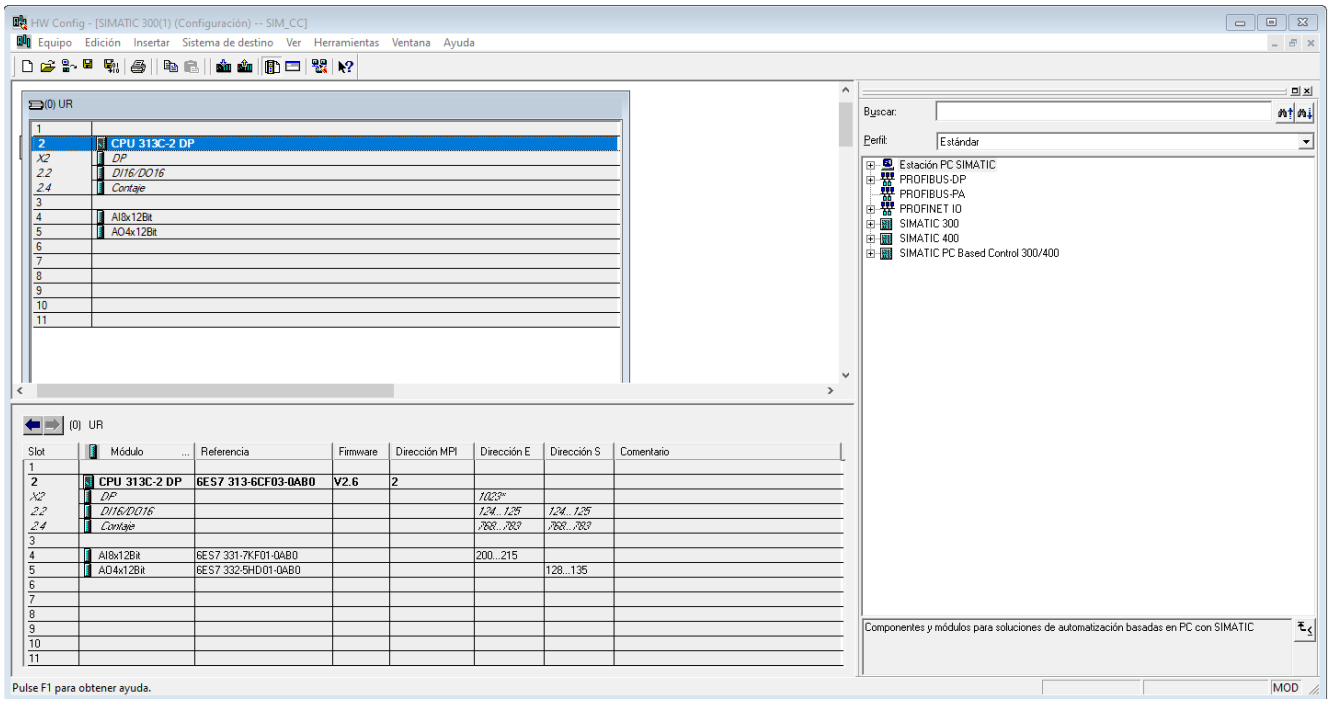


Figura 3-11. Ventana de configuración hardware del PLC2 en STEP7.

Para configurar los módulos de expansión haremos doble click sobre ellos, como se ha mencionado anteriormente.

El módulo SM331 de entradas analógicas queda configurado de la siguiente forma:

Tabla 3-2. Direccionamiento de entradas del módulo SM331.

Entradas	Canal
PIW200	CH 0-1
PIW203	CH 2-3
PIW207	CH 4-5
PIW211	CH 6-7

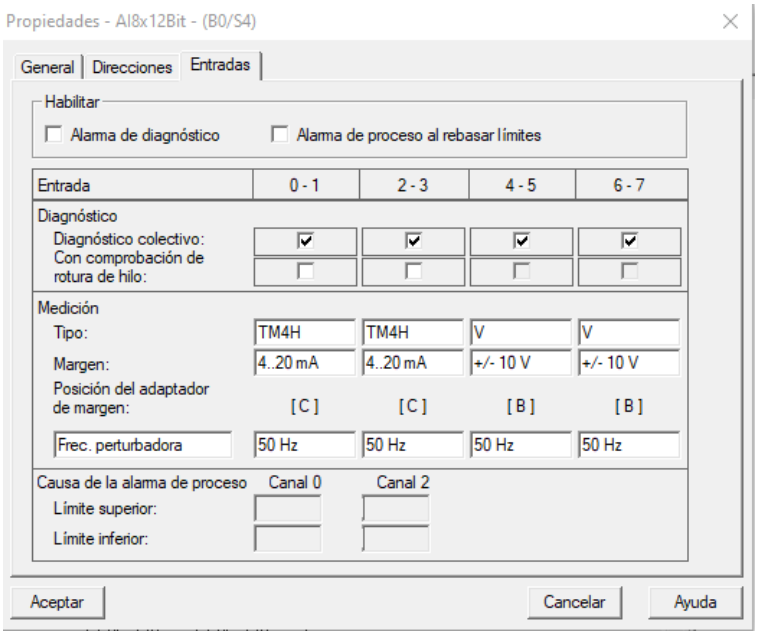


Figura 3-12. Ventana de configuración de entradas analógicas del PLC2 en STEP7.

Nótese que los canales 1,2,3 y 4 han sido configurados con medición en lazo de corriente de rango [4 mA, 20 mA] para dejarlos preparados por si se decide cambiar el modo de funcionamiento del sistema.

Para esta aplicación solo usaremos los canales configurados en tensión de rango [-10 V, +10 V].

Por otro lado, el módulo SM332 de salidas analógicas queda configurado de la siguiente forma:

Tabla 3-3. Direccionamiento de salidas del módulo SM332.

Salidas	Canal
PQW128	CH 0
PQW130	CH 1
PQW132	CH 2
PQW134	CH 3

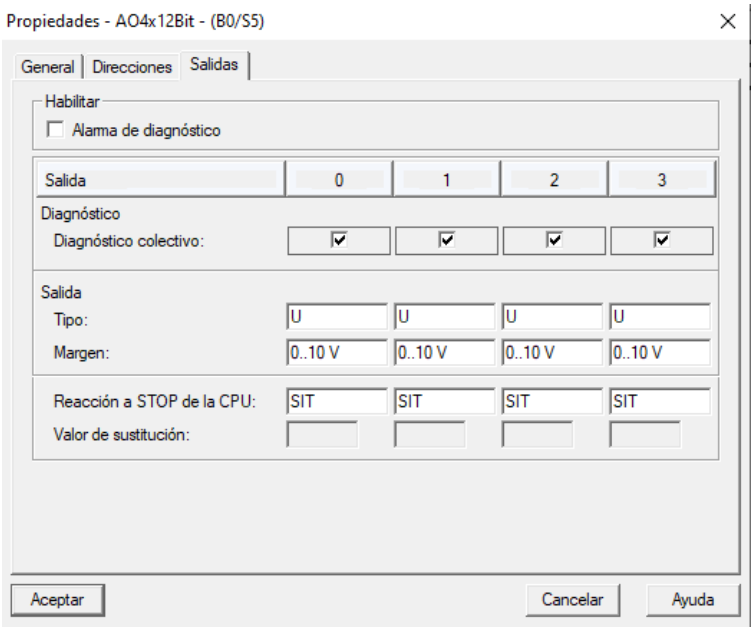


Figura 3-13. Ventana de configuración de salidas analógicas del PLC2 en STEP7..

Dado que el módulo SM332 no tiene posibilidad de trabajar en rango [-10 V, +10 V], lo configuraremos en rango [0 V, +10 V] y el resto de tarjetas de E/S analógicas no usarán escalado bipolar, es decir, solo trabajarán en rango [0 V, +10 V].

Una vez configurados ambos PLC con su periferia (módulos de expansión) será necesario cargar dicha configuración en los dispositivos.

Para ello, necesitaremos un adaptador MPI-USB, como el mostrado en la siguiente figura:

Figura 3-14. PC ADAPTER USB to MPI/DP/PPI - AMSAMOTION

En este caso se ha utilizado un PC ADAPTER MPI to USB de la marca AMSAMOTION (ref: 6ES7 972-0CB20-0XA0).

Conectaremos este adaptador a la interfaz MPI de cada uno de los PLCs y, una vez alimentados los controladores, procederemos a transferir las configuraciones.

Antes de transmitir la configuración, será necesario ajustar/seleccionar la interfaz PG/PC que vayamos a utilizar en STEP7.

A continuación se muestran los pasos a seguir:

Situados en la ventana principal de nuestro proyecto como se muestra en la **Figura *13***, haremos click en *Herramientas -> Ajuste interface PG/PC…* y se abrirá una nueva ventana.

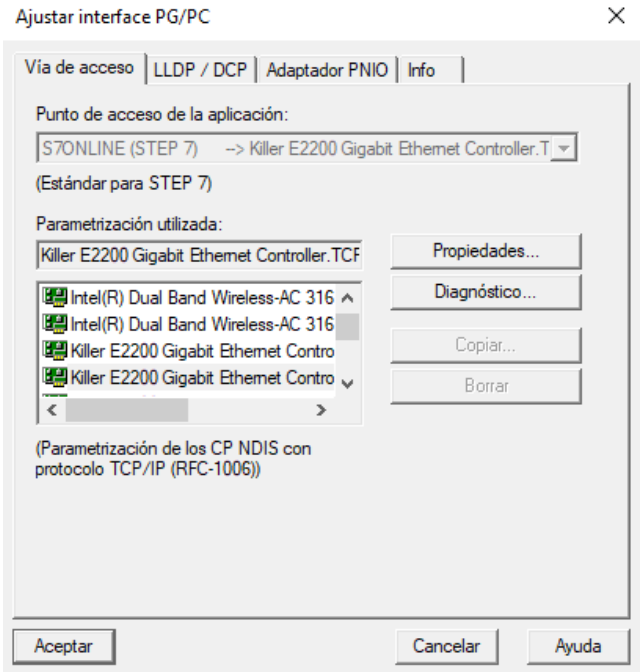


Figura 3-15. Ventana de ajuste de interface PG/PC en STEP7.

A continuación buscaremos en la lista la interfaz *PC ADAPTER.MPI.1,* la seleccionaremos y haremos click en *Aceptar*. Con esto quedará establecido el canal de comunicaciones para configuración entre el PC y el PLC.

Hecho esto, solo queda trasmitir la información al PLC y sus módulos de expansión. Situados en la ventana principal del proyecto, pulsaremos el icono de *Cargar* y todos los archivos del proyecto serán transferidos a los dispositivos.

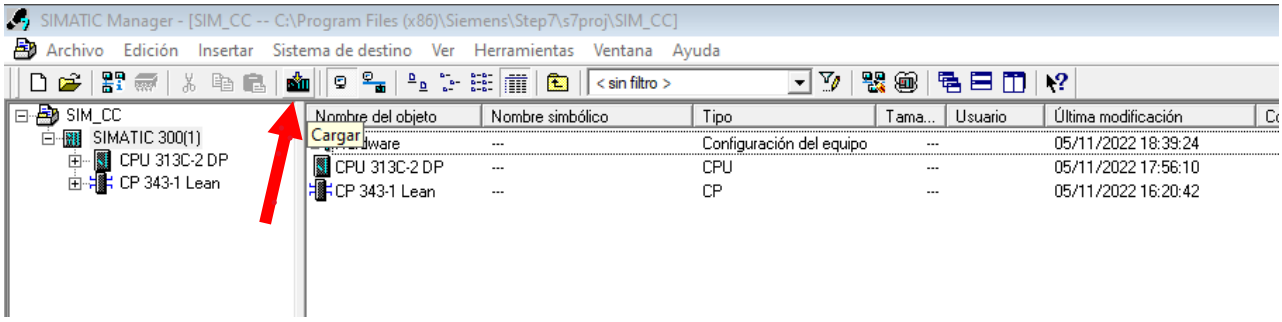


Figura 3-16. Icono de carga de programa en PLC en STEP7.

Durante la carga del programa la CPU pasará a estado *STOP* y una vez terminada la transferencia, nos pedirá confirmación para pasar al estado *RUN*.

Repetiremos este proceso con el PLC2 para transferirle su configuración.

3.3. Configuración de conexión entre KEPServerEX y el controlador.

A continuación configuraremos el servidor OPC y estableceremos la conexión vía TCP/IP entre dicho servidor y el PLC1.

Instalaremos KEPServerEX con su versión de prueba, que nos permitirá hacer uso del servidor OPC sin límites durante, al menos, dos horas. Este software dispone de los drivers necesarios para establecer la comunicación entre el PC y el PLC S7-300 sobre TCP/IP.

El primer paso será iniciar el programa y establecer esta comunicación. Para ello, crearemos un *Nuevo canal* haciendo click derecho sobre *Connectivity*. Al hacer click, se iniciará un wizard de configuración de la comunicación que detallaremos a continuación.

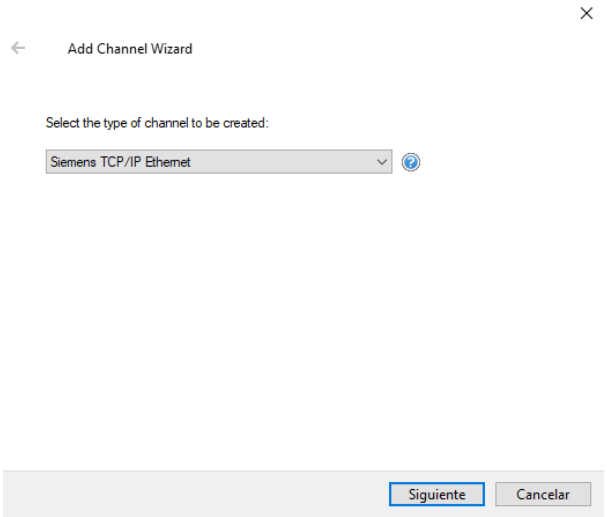


Figura 3-17. Ventana 1 del wizard de configuración de canal de KEPServerEX.

Seleccionaremos el tipo de canal a crear, en nuestro caso, *Siemens TCP/IP Ethernet,* y hacemos click en *Siguiente.*

La siguiente ventana nos permitirá nombrar nuestro objeto (la conexión servidor-PLC). En este ejemplo lo hemos llamado *SiemensTIA.*

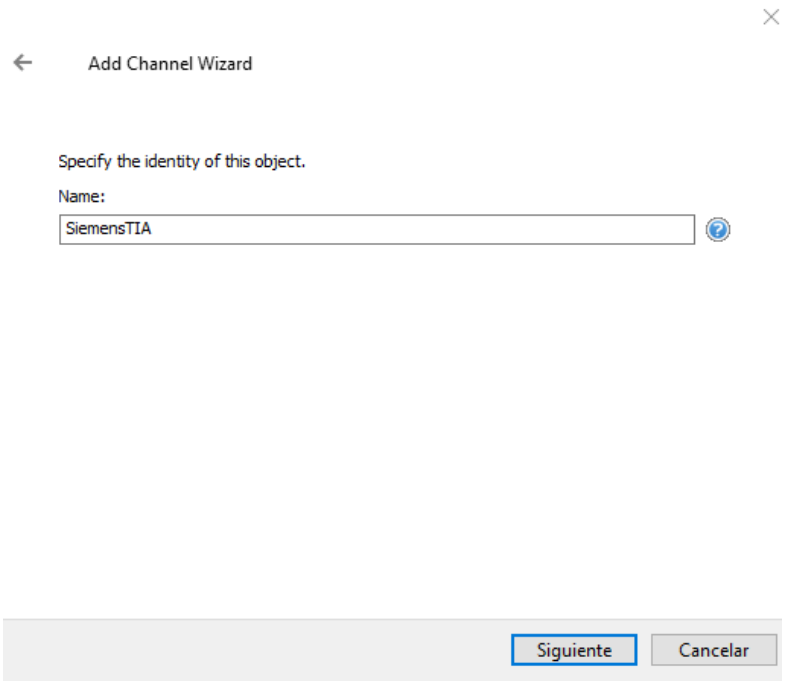


Figura 3-18. Ventana 2 del wizard de configuración de canal de KEPServerEX.

Al pulsar *Siguiente,* pasaremos a la siguiente ventana de selección del adaptador de red.

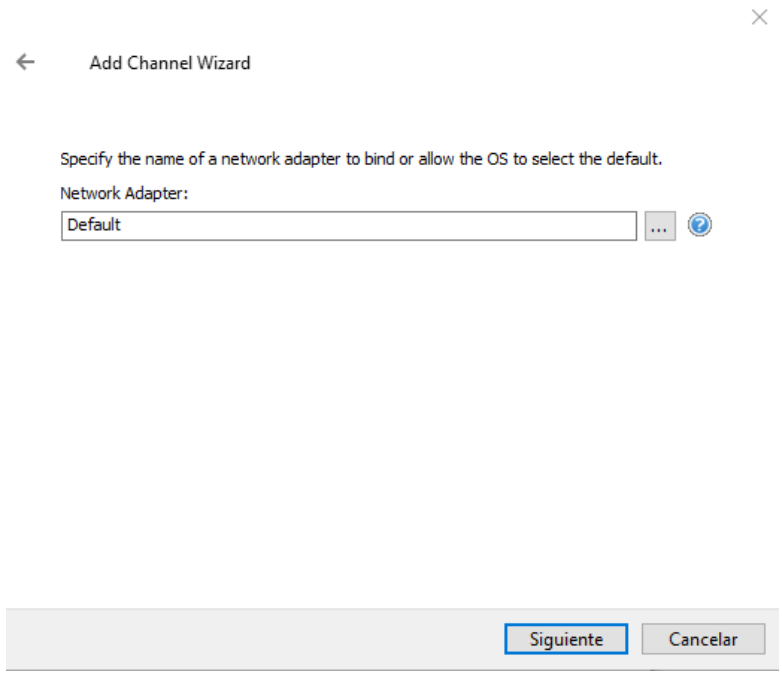


Figura 3-19. Ventana 3 del wizard de configuración de canal de KEPServerEX.

Como el propio wizard indica, podemos indicar el adaptador de red que hará uso del enlace o podemos dejar la opción *Default,* que permitirá al sistema operativo hacer automáticamente esta selección. En este caso se ha seleccionado esta última opción.

Al pulsar *Siguiente*, pasaremos a la ventana de configuración del driver de comunicación.

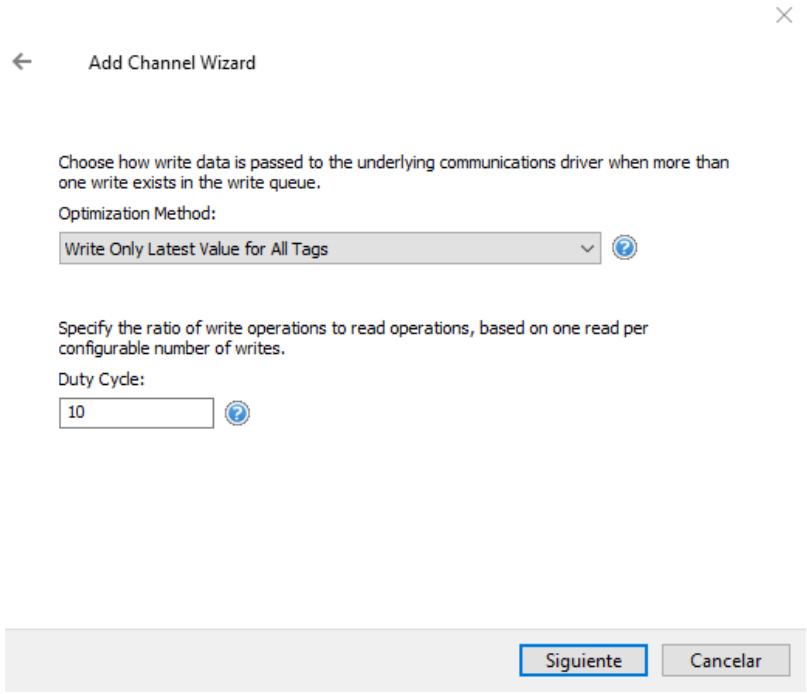


Figura 3-20. Ventana 4 del wizard de configuración de canal de KEPServerEX.

Dado que para nuestra aplicación buscamos tener la máxima tasa de refresco estable posible de las variables, usaremos como método de optimización *Write Only Latest Value for All Tags.*

El *duty cycle* (ratio de nº de lecturas por cada escritura) se ha establecido en un valor igual a *10.* Como nuestra aplicación tendrá mayor número de lecturas que de escrituras, este valor no nos afectará, de acuerdo con la documentación de *KEPServer – Write Optimizations.*

La siguiente ventana nos llevará a la configuración a usar cuando se produzca un valor inválido en notación de coma flotante. Para evitar errores en el controlador, seleccionaremos la opción *Replace with Zero,* lo que hará que si hay que enviar un dato inválido, se reemplazará por *0.*

Al pulsar siguiente, pasaremos a la ventana de resumen de la configuración y pulsaremos *Finalizar* para validar la configuración y establecer la comunicación.

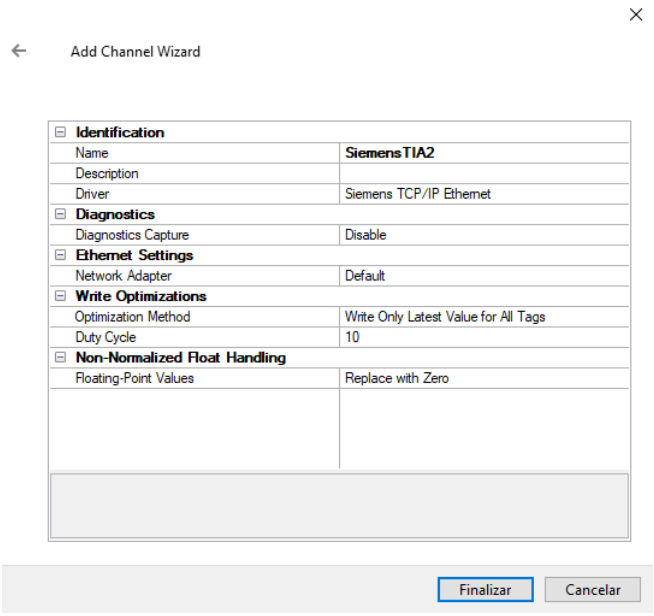


Figura 3-21. Ventana 5 del wizard de configuración de canal de KEPServerEX.

La configuración de las variables a intercambiar entre cliente y servidor OPC las veremos en próximos capítulos.

4 PROGRAMACIÓN DE PLCS PARA CONTROL Y SIMULACIÓN DEL SISTEMA

El conocimiento es poder.

Francis Bacon

En este capítulo vamos a describir cómo se han programado los PLCs involucrados en el sistema, tanto para el control del motor, como para la simulación de este, creando un sistema basado en la técnica *HDL (Hardware-in-the-Loop).*

Definición (Hardware-in-the-Loop):

Se define una simulación Hardware-in-the-Loop (HIL) como una técnica utilizada para el desarrollo y pruebas de sistemas controlados que sustituye el sistema real por un sistema simulado de mismas características.

4.1. Funcionalidades de la aplicación de control

En primer lugar y antes de entrar en detalle sobre la programación de los PLCs, explicaremos cuáles son las principales funcionalidades o modos de control que necesita nuestro sistema.

La aplicación de control tendrá tres modos de funcionamiento que se detallan a continuación:

1. **Modo escalón manual:** Con este modo de funcionamiento, el usuario de la aplicación podrá cambiar la referencia (o set point) de par producido por el motor de continua. El cambio de referencia se hace en forma de escalón.

2. **Modo rampa manual:** El usuario podrá cambiar desde una referencia origen (actual) hasta una referencia de par producido objetivo de manera gradual en un tiempo determinado de cambio. Para esto, el controlador calculará la pendiente entre la referencia origen y la referencia destino para cumplir ese cambio de referencia en el tiempo indicado, aplicando automáticamente saltos de referencia graduales internamente.

3. **Modo programado:** Con este modo de funcionamiento, el usuario podrá programar a través de un fichero *.mat* los cambios de referencia deseados, de manera que el controlador se encargará de ir realizando estos cambios de manera automática en un intervalo de tiempo determinado.

Los modos de funcionamiento indicados anteriormente sólo se encargan de realizar cambios de referencia de manera manual o automática al controlador.

El PLC1 ejecutará un algoritmo de control *PID (Proporcional, integral y derivativo)* que será el encargado de controlar el motor de corriente continua y recibirá los cambios de referencias de la aplicación.

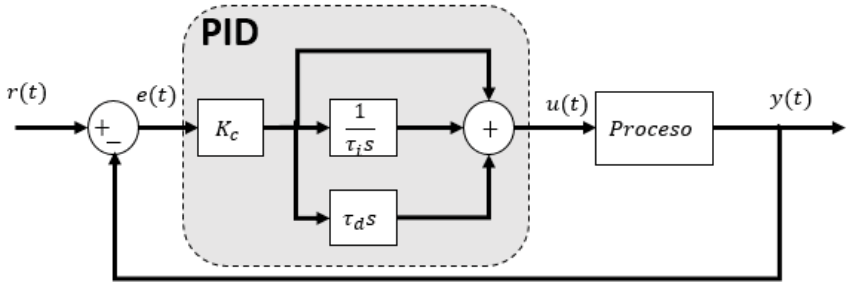


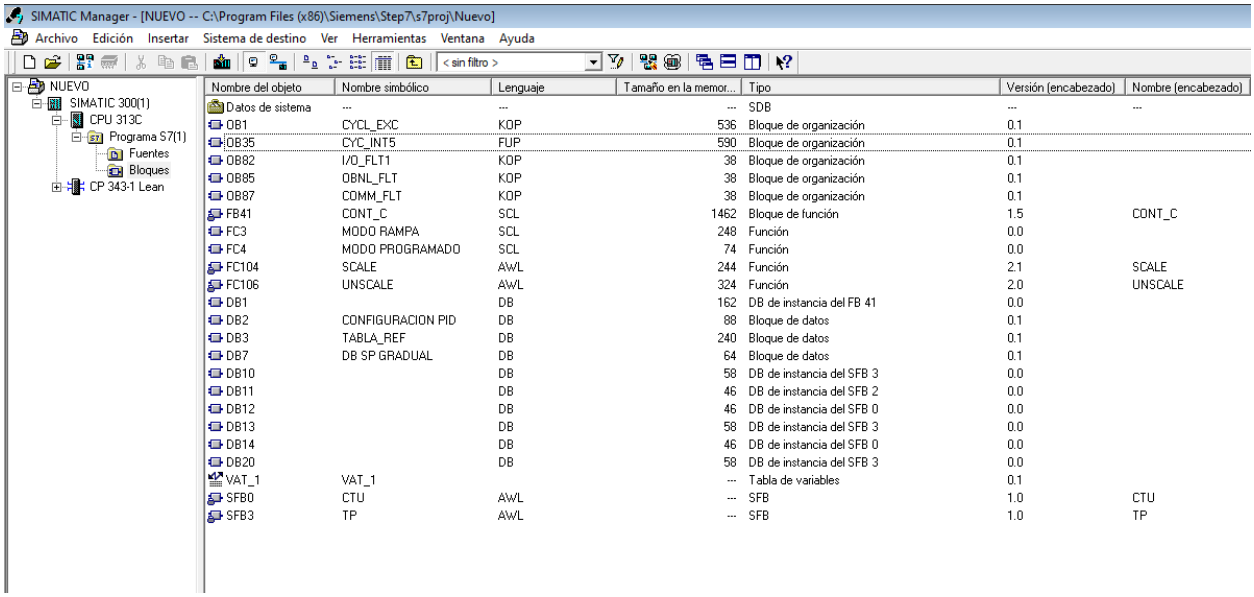
Figura 4-1. Diagrama PID típico.

4.2. Bloque de función de regulación continua (CONT_C)

Para empezar a programar el controlador el primer paso será, una vez abierto *STEP7,* crear un *OB*[3] *cíclico*. Los OB cíclicos son bloques de organización de instrucciones que se ejecutan cíclicamente en el PLC cada *t* unidades de tiempo.

A continuación se detallarán los pasos a seguir para la creación de un OB Cíclico:

Nos situamos en la ventana principal de nuestro proyecto y en la parte izquierda de la ventana, desplegaremos el menú *SIMATIC 300(1) -> Programa S7(1) -> Bloques,* llegando a una ventana similar a la mostrada en la siguiente figura:



Nombre del objeto	Nombre simbólico	Lenguaje	Tamaño en la memor...	Tipo	Versión (encabezado)	Nombre (encabezado)
Datos de sistema	---	---	---	SDB	---	---
OB1	CYCL_EXC	KOP	536	Bloque de organización	0.1	
OB35	CYC_INT5	FUP	590	Bloque de organización	0.1	
OB82	I/O_FLT1	KOP	38	Bloque de organización	0.1	
OB85	OBNL_FLT	KOP	38	Bloque de organización	0.1	
OB87	COMM_FLT	KOP	38	Bloque de organización	0.1	
FB41	CONT_C	SCL	1462	Bloque de función	1.5	CONT_C
FC3	MODO RAMPA	SCL	248	Función	0.0	
FC4	MODO PROGRAMADO	SCL	74	Función	0.0	
FC104	SCALE	AWL	244	Función	2.1	SCALE
FC106	UNSCALE	AWL	324	Función	2.0	UNSCALE
DB1		DB	162	DB de instancia del FB 41	0.0	
DB2	CONFIGURACION PID	DB	88	Bloque de datos	0.1	
DB3	TABLA_REF	DB	240	Bloque de datos	0.1	
DB7	DB SP GRADUAL	DB	64	Bloque de datos	0.1	
DB10		DB	58	DB de instancia del SFB 3	0.0	
DB11		DB	46	DB de instancia del SFB 2	0.0	
DB12		DB	46	DB de instancia del SFB 0	0.0	
DB13		DB	58	DB de instancia del SFB 3	0.0	
DB14		DB	46	DB de instancia del SFB 0	0.0	
DB20		DB	58	DB de instancia del SFB 3	0.0	
VAT_1	VAT_1		---	Tabla de variables	0.1	
SFB0	CTU	AWL	---	SFB	1.0	CTU
SFB3	TP	AWL	---	SFB	1.0	TP

Figura 4-2. Ventana de bloques del proyecto en STEP7.

En esta ventana aparecerán toda la información del programa actual que contenga el PLC.

Haciendo click derecho debajo del listado podremos añadir nuevos bloques a nuestro programa. Insertaremos un OB haciendo click en *Insertar nuevo objeto -> Bloque de organización*.

[3] OB: Organization Block

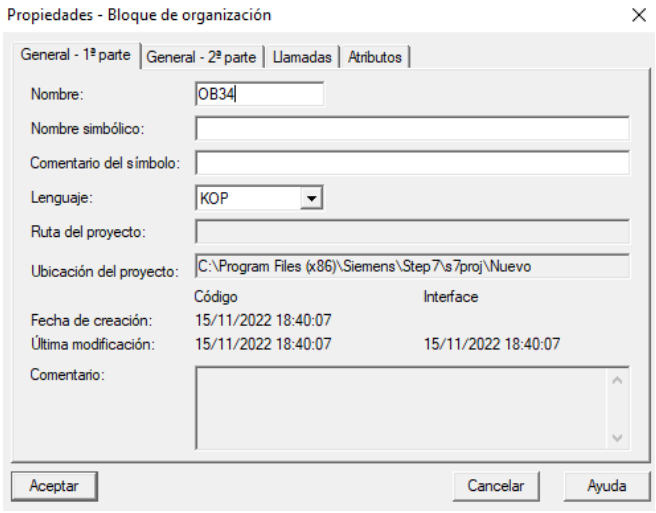


Figura 4-3. Ventana de creación de un bloque de organización en STEP7.

En nuestro caso, utilizaremos el OB35. Este OB no se ha elegido arbitrariamente, si no que por el contrario, es uno de los cuatro OB cíclicos de los que dispone la CPU.

Para ver los OB cíclicos de los que dispone la CPU utilizada, haremos click derecho sobre el nombre de esta, mostraremos las *Propiedades del objeto* y nos moveremos hasta la pestaña de *Alarmas cíclicas*.

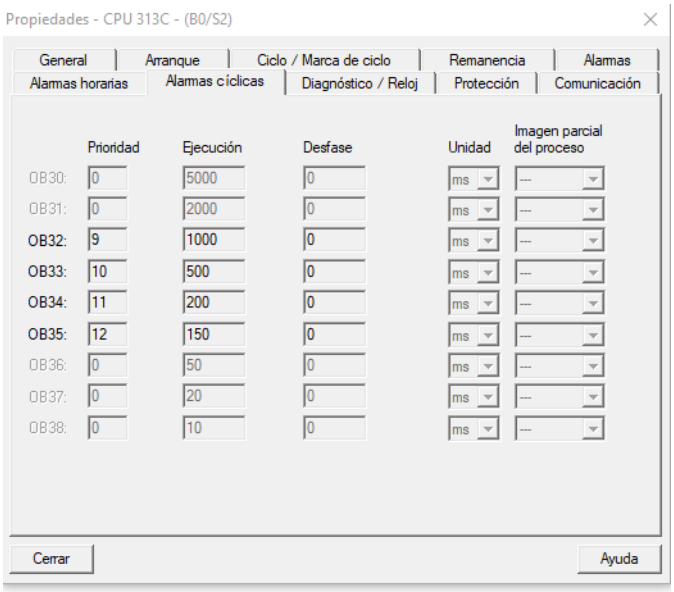


Figura 4-4. Ventana de OBs cíclicos en STEP7.

En la **Figura *30*** podemos ver el listado de OBs cíclicos de la CPU, además de editar su tiempo de ciclo de ejecución. Como se ha comentado, hemos usado el OB35 que se ejecuta cada 150 ms.

Este bloque de organización contendrá tres bloques de función correspondientes al *algoritmo de control*, al *modo rampa* y al *modo programado*.

En este apartado nos centraremos en explicar el bloque de función CONT_C.

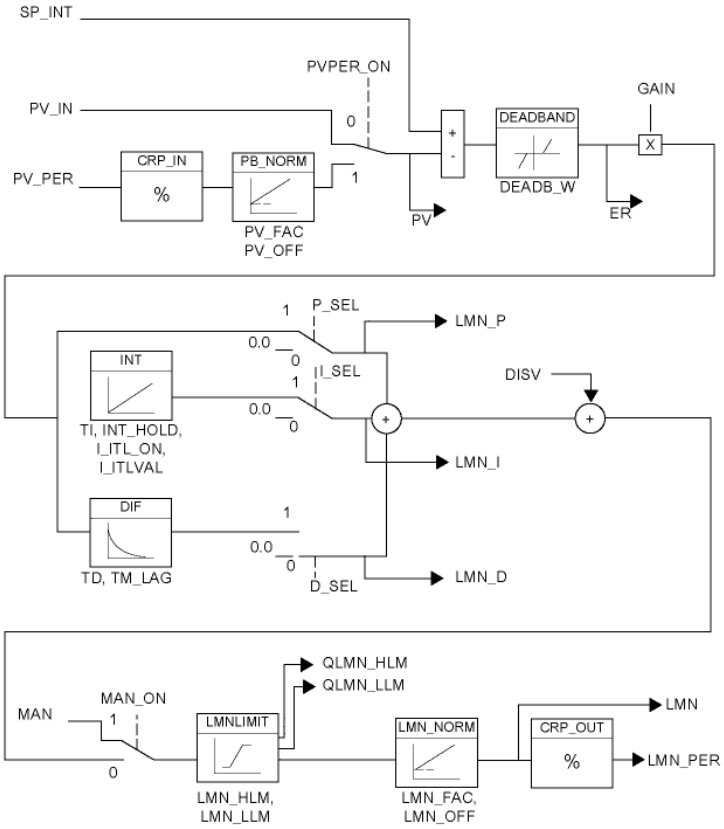


Figura 4-5. Diagrama de bloques de la función CONT_C.

Este bloque de función contiene el algoritmo de control proporcionado por SIEMENS para realizar controles tipo P, PI, PD y PID. Nuestra tarea será, una vez añadido este bloque de función (FB41) al OB35, configurarlo correctamente.

El bloque tiene los siguientes argumentos de entrada y salida:

Tabla 4-1. Parámetros de entrada del bloque de función CONT_C.

Parámetro	Tipo de datos	Por defecto	Descripción
COM_RST	BOOL	FALSE	Rearranque completo El bloque tiene una rutina de inicialización que se procesa cuando está activada la entrada "COM_RST".
MAN_ON	BOOL	TRUE	Conectar a modo manual Si está activada la entrada "Conectar a modo manual", está interrumpido el lazo de regulación. Como valor manipulado se fuerza un valor manual.
PVPER_ON	BOOL	FALSE	Conectar valor real de periferia. Si debe leerse el valor real de la periferia, debe interconectarse la entrada PV_PER con la periferia y activarse la entrada "Conectar valor real de periferia".

P_SEL	BOOL	TRUE	Conectar acción P En el algoritmo PID pueden conectarse y desconectarse individualmente las acciones PID. La acción P está conectada si está activada la entrada "Conectar acción P".
I_SEL	BOOL	TRUE	Conectar acción I En el algoritmo PID pueden conectarse y desconectarse individualmente las acciones PID. La acción I está conectada si está activada la entrada "Conectar acción I".
INT_HOLD	BOOL	FALSE	Congelar acción I La salida del integrador puede congelarse. Para ello se ha de activar la entrada "Congelar acción I".
I_ITL_ON	BOOL	FALSE	Inicializar acción I La salida del integrador puede inicializarse a la entrada I_ITLVAL. Para ello se ha de activar la entrada "Inicializar acción I".
D_SEL	BOOL	FALSE	Conectar acción D En el algoritmo PID pueden conectarse y desconectarse individualmente las acciones PID. La acción D está conectada si está activada la entrada "Conectar acción D".
CYCLE	TIME	T#1s	Tiempo de muestreo El tiempo entre las llamadas del bloque debe ser constante. La entrada "Tiempo de muestreo" indica el tiempo entre las llamadas del bloque.
SP_INT	REAL	0.0	Consigna interna La entrada "Consigna interna" sirve para ajustar un valor de consigna.
PV_IN	REAL	0.0	Entrada de valor real En la entrada "Entrada de valor real" puede parametrizarse un valor de puesta en servicio, o interconectarse un valor real externo en formato en coma flotante.
PV_PER	WORD	W#16#0000	Valor real de periferia El valor real en formato de periferia se interconecta con el regulador en la entrada "Valor real de periferia".
MAN	REAL	0.0	Valor manual La entrada "Valor manual" sirve para establecer un valor manual mediante función de manejo/visualización (interfaz hombre máquina)

GAIN	REAL	2.0	Ganancia proporcional La entrada "Ganancia proporcional" indica la ganancia del regulador.
TI	TIME	T#20s	RESET TIME / Tiempo de acción integral La entrada "Tiempo de acción integral" determina el comportamiento temporal del integrador.
TD	TIME	T#10s	Tiempo de diferenciación (acción derivativa) La entrada "Tiempo de diferenciación" determina el comportamiento temporal del diferenciador.
TM_LAG	TIME	T#2s	Tiempo de retardo de la acción D El algoritmo de la acción D contiene un retardo que puede parametrizarse en la entrada "Tiempo de retardo de la acción D".
DEADB_W	REAL	0.0	Ancho de zona muerta La diferencia de regulación se conduce por una zona muerta. La entrada "Ancho de zona muerta" determina el tamaño de la zona muerta.
LMN_HLM	REAL	100.0	Límite superior del valor manipulado El valor manipulado tiene siempre un límite superior y uno inferior. La entrada "Límite superior del valor manipulado" indica la limitación superior.
LMN_LLM	REAL	0.0	Límite inferior del valor manipulado El valor manipulado tiene siempre un límite superior y uno inferior. La entrada "Límite inferior del valor manipulado" indica la limitación inferior.
PV_FAC	REAL	1.0	Factor de valor real La entrada "Factor de valor real" se multiplica por el valor real. La entrada sirve para la adaptación del margen de valor real.
PV_OFF	REAL	0.0	Offset del valor real La entrada "Offset del valor real" se suma con el valor real. La entrada sirve para la adaptación del margen de valor real.
LMN_FAC	REAL	1.0	Factor del valor manipulado La entrada "Factor del valor manipulado" se multiplica por el valor manipulado. La entrada sirve para la adaptación del margen de valor manipulado.

LMN_OFF	REAL	0.0	Offset del valor manipulado La entrada "Offset del valor manipulado" se suma al valor manipulado. La entrada sirve para la adaptación del margen de valor manipulado.
I_ITLVAL	REAL	0.0	Valor de inicialización de la acción I La salida del integrador puede ponerse en la salida I_ITL_ON. En la entrada "Valor de inicialización de la acción I" está el valor de inicialización.
DISV	REAL	0.0	Magnitud perturbadora Para control anticipativo de la magnitud perturbadora, ésta se conecta en la entrada "Magnitud perturbadora".

Tabla 4-2. Parámetros de salida del bloque de función CONT_C.

Parámetro	Tipo de datos	Por defecto	Descripción
LMN	REAL	0.0	Valor manipulado En la salida "Valor manipulado" se saca en formato en coma flotante el valor manipulado que actúa efectivamente.
LMN_PER	WORD	W#16#0000	Valor manipulado periferia El valor manipulado en formato de periferia se interconecta con el regulador en la salida "Valor manipulado periferia".
QLMN_HLM	BOOL	FALSE	Alcanzado el límite superior del valor manipulado El valor manipulado tiene siempre un límite superior y un límite inferior. La salida "Alcanzada limitación superior del valor manipulado" indica la superación de la limitación superior.
QLMN_LLM	BOOL	FALSE	Alcanzado el límite inferior del valor manipulado El valor manipulado tiene siempre un límite superior y un límite inferior. La salida "Alcanzado el límite inferior del valor manipulado" indica la superación de la limitación inferior.
LMN_P	REAL	0.0	Acción P La salida "Acción P" contiene la componente proporcional de la magnitud manipulada.
LMN_I	REAL	0.0	Acción I La salida "Acción I" contiene la componente integral de la magnitud manipulada.

LMN_D	REAL	0.0	Acción D La salida "Acción D" contiene la componente diferencial de la magnitud manipulada.
PV	REAL	0.0	Valor real Por la salida "Valor real" se emite el valor real que actúa efectivamente.
ER	REAL	0.0	Error de regulación Por la salida "Error de regulación" se emite la diferencia o error de regulación que actúa efectivamente.

Para añadir este bloque de función al OB35, se ha creado un nuevo segmento KOP[4] (lenguaje de contactos o LADDER) y buscamos el *FB41 (CONT_C) en Librerías -> Standard Library -> PID Control Blocks -> FB41 CONT_C ICONT*. Bastará con arrastrar el FB deseado hasta nuestro segmento ladder y este se añadirá al programa.

[4] KOP: Abreviatura del alemán *Kontaktplan*, en español *Diagrama de Escalera* (conocido como LD o LADDER).

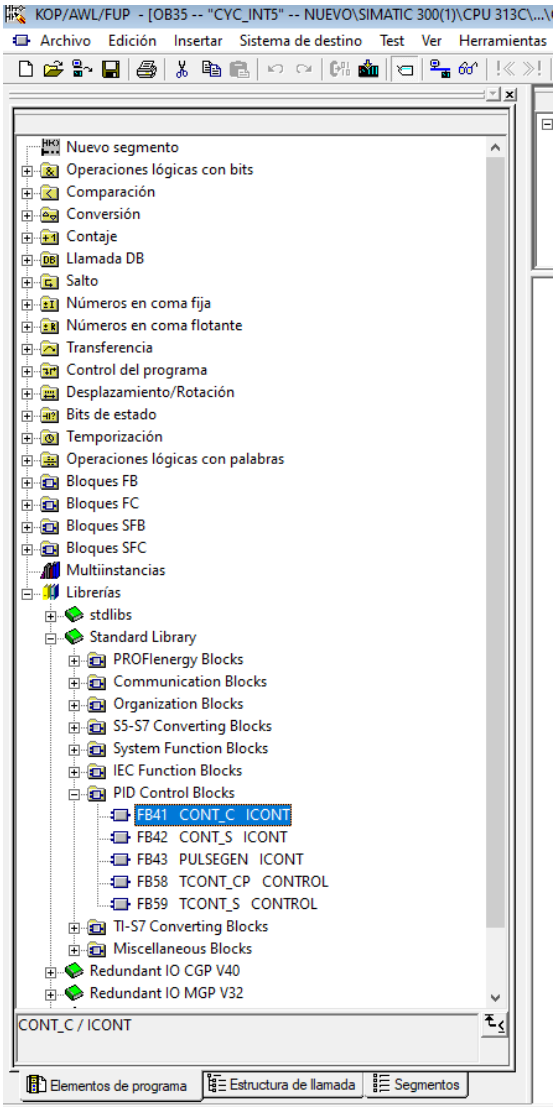


Figura 4-6. Función CONT_C las librerías del programa.

Con el objetivo de organizar nuestro código y hacerlo más legible para los usuarios, gran parte de las variables involucradas en el control de nuestro sistemas han sido organizadas en DBs[5].

4.2.2 Creación de un bloque de datos (DB)

De manera análoga a la creación de un OB vista en este mismo capítulo, se creará un DB (*Insertar nuevo objeto -> Bloque de datos*).

Nuestro programa contiene tres bloques de datos creados para organizar variables y varios bloques de datos de instancias, que son creados por el software de manera automática como consecuencia de la inserción en el programa de un bloque de función.

El *OB2 – CONFIGURACIÓN PID* contiene varias variables utilizadas por OPC para pasar datos de configuración al bloque de función de control continuo.

[5] DB (Data Block): Bloque de datos que nos permite crear una estructura de variables de diferentes tipos. Los datos son de lectura y escritura y pueden ser accedidos desde cualquier parte del programa, ya sea desde otro bloque o función.

Dirección	Nombre	Tipo	Valor inicial
0.0		STRUCT	
+0.0	PID_RESET	BOOL	FALSE
+0.1	PID_MANUAL	BOOL	FALSE
+0.2	PID_PERIF_ENT	BOOL	FALSE
+0.3	PID_P_ON	BOOL	TRUE
+0.4	PID_I_ON	BOOL	FALSE
+0.5	PID_D_ON	BOOL	FALSE
+0.6	PID_CONG_CONTROL	BOOL	FALSE
+2.0	PID_T_MUESTREO	TIME	T#10MS
+6.0	PID_SP	REAL	0.000000e+000
+10.0	PID_PROCESS_VALUE	REAL	0.000000e+000
+14.0	PID_KP	REAL	1.000000e+000
+18.0	PID_TI	TIME	T#0MS
+22.0	PID_TD	TIME	T#0MS
+26.0	PID_RETARDO	TIME	T#0MS
+30.0	PID_DEAD_BAND	REAL	0.000000e+000
+34.0	PID_LIM_SUP	REAL	1.000000e+002
+38.0	PID_LIM_INF	REAL	0.000000e+000
+42.0	PID_VALOR_ACCONTROL	REAL	0.000000e+000
+46.0	PID_VALOR_MANUAL	REAL	0.000000e+000
+50.0	DB_FALSO	BOOL	FALSE
+50.1	DB_TRUE	BOOL	TRUE
=52.0		END_STRUCT	

Figura 4-7. Vista DB PID.

Para acceder a las variables del DB se utilizará el siguiente direccionamiento:

- **Ejemplo de acceso a la variable booleana PID_RESET** -> DB2.DBX0.0
- **Ejemplo de acceso a la variable real PID_KP** -> DB2.DBD14

Una vez añadida la función CONT_C al OB35 y creado el DB2, añadiremos a la llamada de la función las variables de entrada correspondientes, quedando:

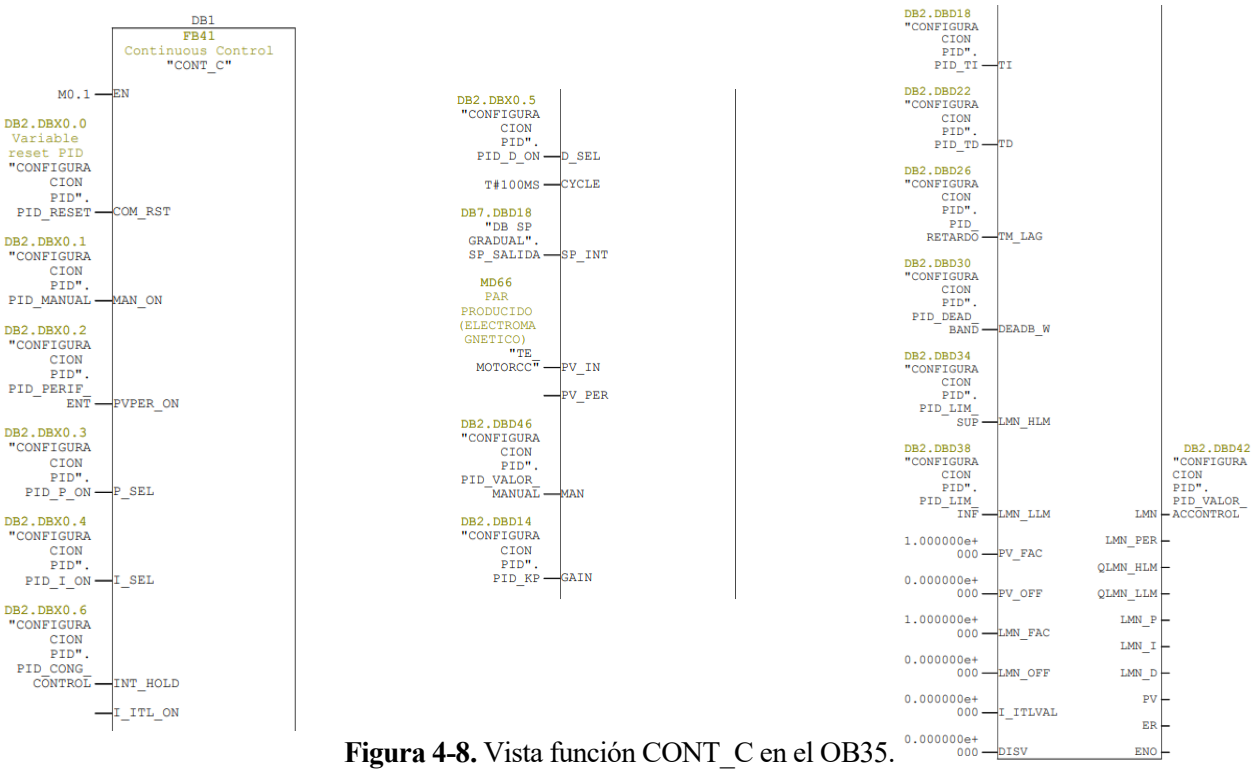


Figura 4-8. Vista función CONT_C en el OB35.

4.3. Función Modo Rampa

Como se ha indicado a principio del capítulo, uno de las funcionalidades del sistema es el *Modo Rampa*. Este modo permite cambiar automáticamente la referencia de manera gradual entre una referencia de par origen, destino y un tiempo objetivo.

Para codificar esta función en el controlador, se ha creado un nuevo bloque de función en lenguaje SCL (Structured Control Language)[6].

4.3.2 Creación de un bloque de función en SCL

Para crear un bloque de función se han seguido los siguientes pasos:

En primer lugar nos situaremos en la ventana principal del proyecto y nos moveremos hasta el apartado *SIMATIC 300(1) -> CPU 313C -> Programa S7(1) -> Fuentes*. Una vez en esta ventana, haremos click derecho e insertamos un nuevo objeto *Insertar nuevo objeto -> Fuente SCL.*

Una vez hecho esto se abrirá el editor SCL y podemos empezar a escribir nuestro nuevo bloque de función RAMPAS_SCL.

El código SCL de la función es el siguiente:

Código 4.1 Función *RAMPA_SCL* para MODO RAMPA en SCL.

```
FUNCTION FC3:BOOL

VAR_TEMP // variables temporales
    SUBIR: BOOL;
    BAJAR: BOOL;
END_VAR

VAR_INPUT // variables de entrada
    ORIGEN_SP : REAL;
    DESTINO_SP : REAL;
    ACTUAL_SP : REAL;
    SALTO_SP : REAL;
END_VAR

VAR_OUTPUT // variables de salida
    SALIDA_SP : REAL;
    END_VAR
```

[6] SCL (Structured Control Language): Lenguaje de programación ST "Structured Text" definido en la norma DIN EN-61131-3 (IEC 61131-3).

```
    // Área de instrucciones
    IF((ORIGEN_SP < DESTINO_SP) AND (DESTINO_SP - ORIGEN_SP > SALTO_SP)) THEN
        SUBIR := TRUE;
        BAJAR := FALSE;
    ELSIF((ORIGEN_SP > DESTINO_SP) AND (ORIGEN_SP - DESTINO_SP > SALTO_SP) )
THEN
        SUBIR := FALSE;
        BAJAR := TRUE;
    ELSE
        SUBIR := FALSE;
        BAJAR := FALSE;
        SALIDA_SP := DESTINO_SP;
    END_IF;

    IF((ACTUAL_SP < DESTINO_SP) AND SUBIR) THEN
        SALIDA_SP := ACTUAL_SP + SALTO_SP;
    END_IF;

    IF((ACTUAL_SP > DESTINO_SP) AND BAJAR) THEN
         SALIDA_SP := ACTUAL_SP - SALTO_SP;
    END_IF;
    ;
    FC3 := TRUE;
END_FUNCTION
```

Una vez escrito el código de la función, compilaremos el código y volveremos a transmitir el programa al PLC.

De la misma manera que se añadió la función CONT_C en el OB35, ahora añadiremos una nueva sección KOP dentro del OB35 que incluya la llamada a la función RAMPA_SCL, quedando:

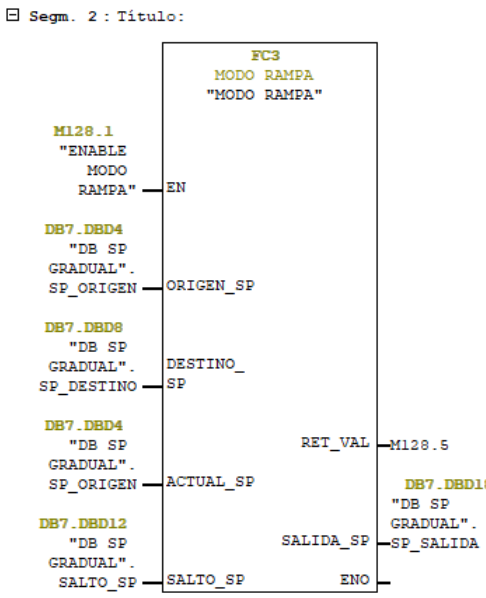


Figura 4-9. Bloque de función FC3 – MODO RAMPA.

Nótese que en la llamada a la función FC3 dentro del OB35 ya han sido incluídas las variables de entrada y salidas dentro de un nuevo DB.

El DB utilizado es el DB7, con nombre DB SP GRADUAL. La estructura de este DB es la siguiente:

Dirección	Nombre	Tipo	Valor inicial
0.0		STRUCT	
+0.0	ANCHO_ESCALON	TIME	T#200MS
+4.0	SP_ORIGEN	REAL	0.000000e+000
+8.0	SP_DESTINO	REAL	0.000000e+000
+12.0	SALTO_SP	REAL	0.000000e+000
+16.0	CAMBIO_SP	BOOL	FALSE
+18.0	SP_SALIDA	REAL	0.000000e+000
+22.0	SP_UPDATE	BOOL	FALSE
+22.1	TIMER_ACTIVO	BOOL	FALSE
+22.2	SUBIR	BOOL	FALSE
+22.3	BAJAR	BOOL	FALSE
+22.4	SEGUIR_SUBIENDO	BOOL	FALSE
+22.5	SEGUIR_BAJANDO	BOOL	FALSE
+24.0	VALOR_ACTUAL	REAL	0.000000e+000
=28.0		END_STRUCT	

Figura 4-10. Vista DB SP GRADUAL.

NOTA: Este DB contiene variables no utilizadas con el fin de que sean utilizadas en futuras versiones del control o para depuración.

4.4. Función Modo Programado

La función *Modo Programado* ha sido programada en un bloque SCL y se encarga de leer los valores de un array de variables de tipo REAL que contienen los valores de las referencias programadas. Esta lectura se produce secuencialmente cada cierto intervalo de tiempo.

Código 4.2 Función *MODO_PROGRAMA* en SCL.

```
FUNCTION FC4: VOID
VAR_INPUT
    PUNTERO: INT;
END_VAR
VAR_OUTPUT
    REF_SALIDA: REAL;
END_VAR
BEGIN
REF_SALIDA := "TABLA_REF".REF[PUNTERO]; //ESCRITURA DEL VALOR LEIDO EN DB A LA SALIDA
END_FUNCTION
```

Esta función será llamada desde el OB35 con el mismo procedimiento visto en apartados anteriores. La llamada a la función dentro del OB queda de la siguiente manera:

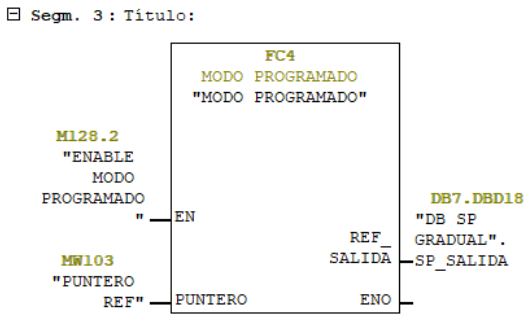


Figura 4-11. Vista de la llamada a la función MODO PROGRAMADO en el OB35.

En este caso, tanto la entrada ENABLE del bloque de función como el puntero has sido reccionados a variables que no se encuentran en un DB.

La salida, dado que el objeto de la función es ir aplicando cambios automáticos de referencia, debe ser la variable SP_SALIDA, o lo que es lo mismo, la entrada de SET POINT del bloque de función de control continuo (CONT_C).

4.5. Bloque de organización principal

El bloque de organización principal del programa (*OB1)* contiene el resto de operaciones a realizar por el programa, como son incrementar el valor del puntero de la función *modo programa*, evitar su desbordamiento y escalar las entradas y salidas analógicas.

A continuación se detalla cada segmento ladder del programa principal:

4.5.2 Segmento 1

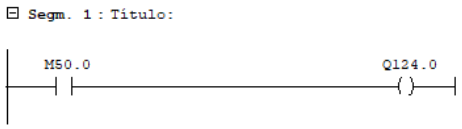


Figura 4-12. Segmento 1 en el OB1.

El contacto M50.0 es una marca de ciclo configurada del PLC, es decir, se ha configurado para se active automáticamente dando un pulso cada 0.1 segundos (f = 10 Hz).

Las marcas cíclicas son 8 señales digitales que podemos activar en nuestro PLC y que nos proporcionan las siguientes salidas:

Tabla 4-3. Bits de marcas de ciclo.

Bit de marca	Duración del período (s)	Frecuencia (Hz)
0	0,1	10
1	0,2	5
2	0,4	2,5
3	0,5	2
4	0,8	1,25
5	1,0	1
6	1,6	0,625
7	2,0	0,5

Para configurar la palabra de memoria reservada para las marcas de ciclo, lo haremos desde la configuración de la CPU, en el apartado de *Ciclo/Marca de ciclo* como se muestra a continuación:

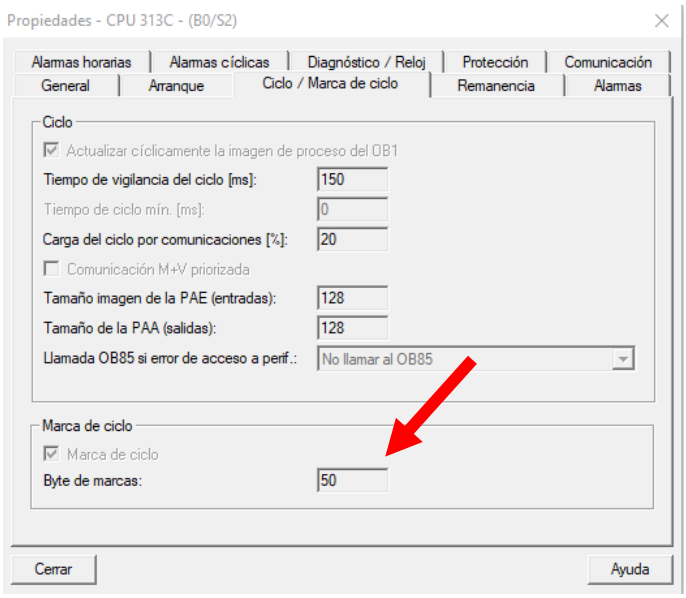


Figura 4-13. Ventana de configuración de marcas de ciclo en STEP7.

4.5.3 Segmento 2

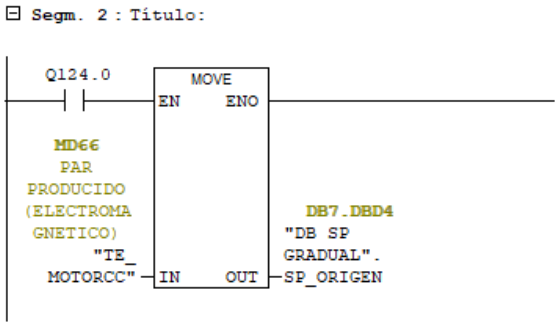


Figura 4-14. Segmento 2 en el OB1.

El segmento 2 mueve cada 0.1s el valor de MD66 hasta DB7.DBD4.

La doble palabra de memoria MD66 contiene el valor escalado de 0 a 100 de la entrada analógica correspondiente al PAR PRODUCIDO por el motor de corriente continua.

4.5.4 Segmento 3

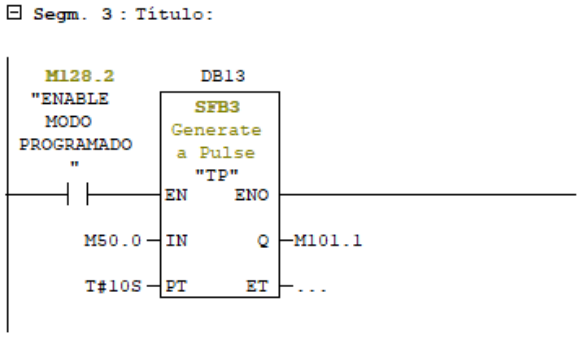


Figura 4-15. Segmento 3 en el OB1.

El segmento 3 genera un pulso en M101.1 cada PT segundos. El bit M101.1 se utilizará como variable de incremento del índice que recorre el array del modo programado.

Este segmento solo se ejecutará si el bit ENABLE MODO PROGRAMADO está activo, es decir, el sistema funciona en *modo programado*.

4.5.5 Segmento 4

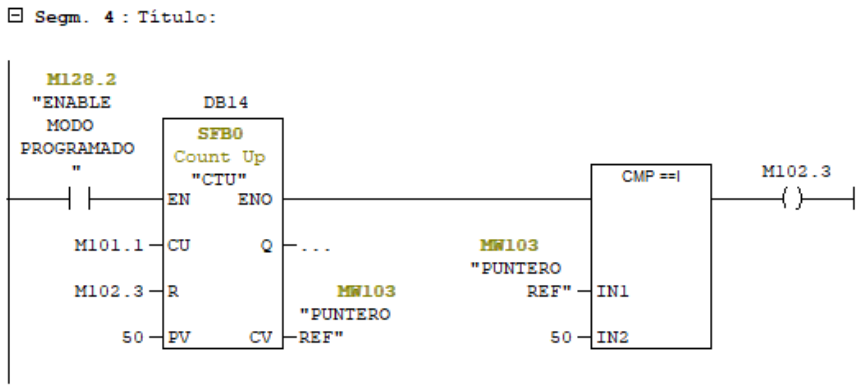


Figura 4-16. Segmento 4 en el OB1.

El segmento 4 realiza dos operaciones.

- **Función contaje ascendente (CTU):** La variable de incremento del contador es la marca M101.1 vista en el segmento 3. La salida de este contador (CV) es una variable de tipo INT (de 1 a 50) que apunta al PUNTERO REF (MW103) utilizado para recorrer el array del modo programado.
- **Función comparación (CMP ==I):** Si la variable PUNTERO REF (MW103) es igual a 50 (tamaño del array de referencias), se activará la marca M102.3 y se reiniciará el índice de contaje (PUNTERO REF). Esto hace que el modo programado se pueda ejecutar cíclicamente y no se produzcan errores por acceso del índice fuera del array.

4.5.6 Segmento 5

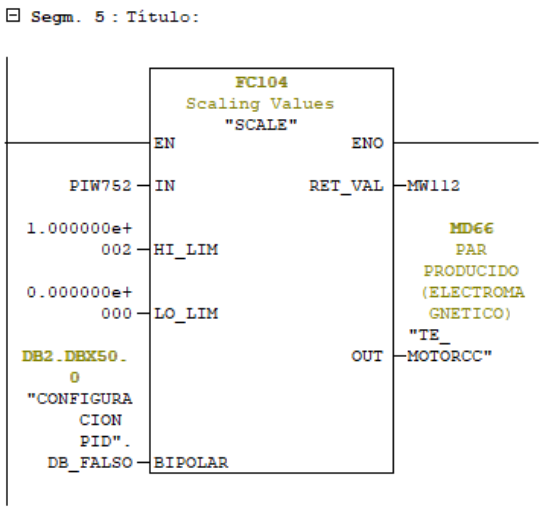


Figura 4-17. Segmento 5 en el OB1.

El segmento 5 escala la entrada analógica PIW752 a valores tipo REAL de entre 0.0 y 100.0.

El valor REAL de salida de la función de escalado es movido a la variable MD66 como PAR PRODUCIDO (ELECTROMAGNÉTICO) “TE_MOTORCC”.

Esta entrada analógica está conectada a una de las salidas analógicas del PLC2, encargado de simular el comportamiento del motor de corriente continua.

4.5.7 Segmento 6

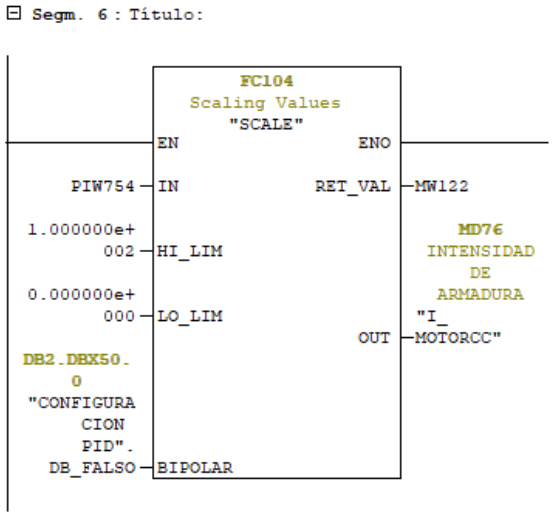


Figura 4-18. Segmento 6 en el OB1.

El segmento 6 escala la entrada analógica PIW754 a valores tipo REAL de entre 0.0 y 100.0.

El valor REAL de salida de la función de escalado es movido a la variable MD76 como INTENSIDAD DE ARMADURA “I_MOTORCC”.

Esta entrada analógica está conectada a una de las salidas analógicas del PLC2, encargado de simular el comportamiento del motor de corriente continua.

4.5.8 Segmento 7

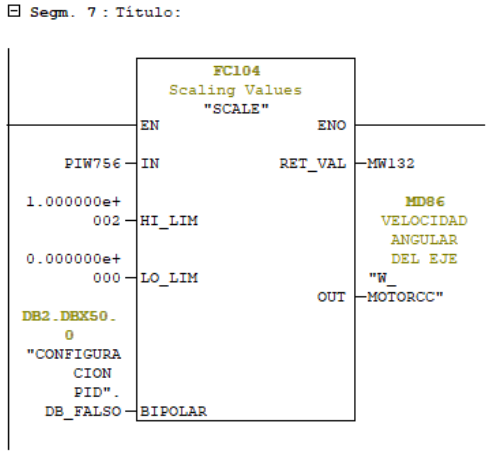


Figura 4-19. Segmento 7 en el OB1.

El segmento 7 escala la entrada analógica PIW756 a valores tipo REAL de entre 0.0 y 100.0.

El valor REAL de salida de la función de escalado es movido a la variable MD86 como VELOCIDAD ANGULAR DEL EJE "W_MOTORCC".

Esta entrada analógica está conectada a una de las salidas analógicas del PLC2, encargado de simular el comportamiento del motor de corriente continua.

4.5.9 Segmento 8

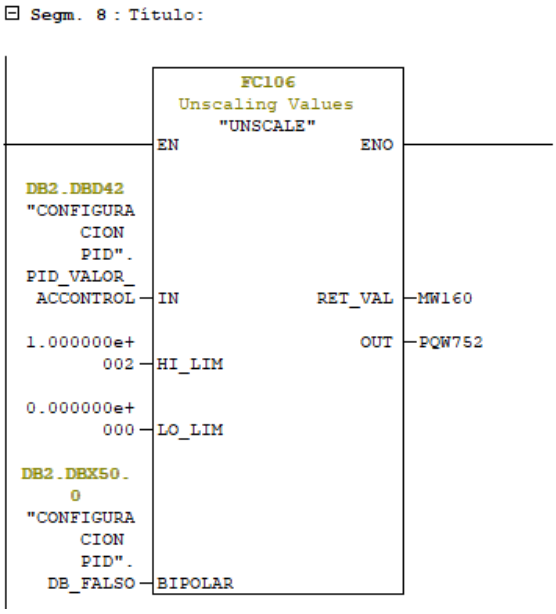


Figura 4-20. Segmento 8 en el OB1.

El segmento 8 desescala valores tipo REAL de entre 0.0 y 100.0 de la salida de la acción de control del bloque de función de control continuo (CONT_C) a valores del rango de salida en tensión de PQW752.

Esta salida analógica está conectada a una de las entradas analógicas del PLC2, encargado de simular el comportamiento del motor de corriente continua.

Con esto, concluímos la configuración del PLC1 (controlador del motor de corriente continua).

4.6. Modelo para simulación del motor de corriente continua

A continuación, se mostrará el modelo del motor de corriente continua utilizado para la simulación del sistema.

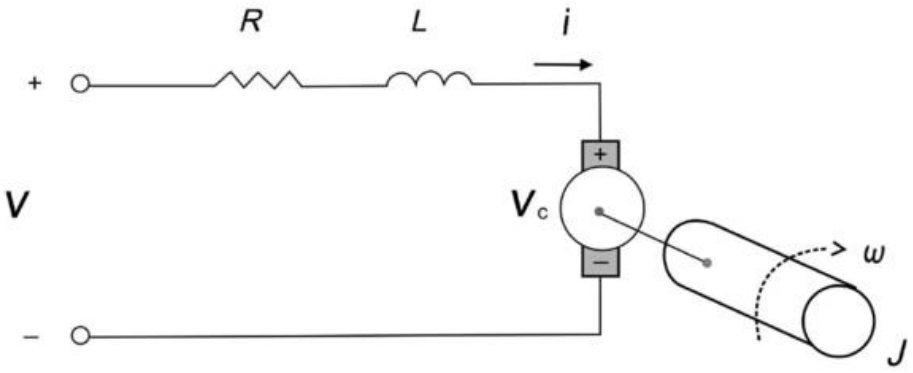


Figura 4-21. Modelo general de un motor de corriente continua.

El modelo lineal del motor de corriente se describe por las siguientes ecuaciones:

$$Vk(t) = Ri(t) + L\frac{di(t)}{dt} + Vcem \quad (1)$$

$$Tm(t) = J \cdot \frac{d\omega(t)}{dt} + B \cdot \omega(t) \quad (2)$$

$$Vcem(t) = Ka \cdot \omega(t) \quad (3)$$

$$Tm(t) = Km \cdot i(t) \quad (4)$$

Discretizando el modelo y despejando, llegaremos a las ecuaciones a usar en el simulador del motor de corriente continua:

$$i = \frac{La \cdot \frac{i}{\Delta t} - Vcem + Vk}{Ra + \frac{La}{\Delta t}} \quad (5)$$

$$\omega = \frac{Te - Text + \frac{Kj}{\Delta t} \cdot \omega}{\frac{Kj}{\Delta t} + KB} \quad (6)$$

$$Vcem = KB \cdot \omega \quad (7)$$

$$Te = Ke \cdot i \quad (8)$$

Donde,

- *Vk* es la tensión inducida al motor de corriente continua.
- *i* es la intensidad de armadura.
- *Vcem* es la tensión contraelectromotriz.
- *Te* es el par producido por el motor de corriente continua.
- *Text* es el par externo ejercido por el motor de inducción sobre el motor de corriente continua.
- *ω* es la velocidad angular del eje.
- *DeltaT* es el tiempo de integración.

- *Ra* es la resistencia modelo.
- *La* es la inductancia modelo.
- *Ke* es la constante de torque expresada en N·m/A.
- *Kb* es la constante de fuerza contraelectromotriz expresada en V·s.
- *B* es el coeficiente de fricción expresado en N·m·s.
- *J* es el momento de inercia expresado en Kg/m^2.

4.7. Función SCL de simulación del motor de corriente continua

La función *SIM_CC* ha sido programada en un bloque SCL y se encarga de simular el comportamiento del motor de corriente continua a partir de las ecuaciones vistas anteriormente. Esta función se ejecuta en un OB cíclico.

Código 4.2 Función *MODO_PROGRAMA en SCL.*

```
FUNCTION_BLOCK FB1
VAR_INPUT
    K_RA : REAL; //Resistencia modelo
    K_LA : REAL; //Inductancia modelo
    K_KE : REAL; //Constante de torque N*m/A
    K_KB : REAL; //Constante de fuerza contraelectromotriz V*s
    K_B : REAL;  //Coeficiente de fricción N*m*s
    K_J : REAL;  //Momento de incercia Kg/m2
    TEXT_K : REAL ; //Par externo ejercido por el motor de induccion sobre el
motor de cc.
    V_K : REAL;  //Tension inducida
    DELTA_T : REAL; //Tiempo de integración
END_VAR
VAR_OUTPUT
    VCEM_K : REAL; //Tensión contraelectromotriz
    TE_K : REAL; //Par producido
    W_K : REAL; //Velocidad angular del eje
    I_K : REAL; //Intensidad de armadura
END_VAR

VAR_TEMP
END_VAR
```

```
    I_K := ((K_LA*I_K/DELTA_T) - VCEM_K + V_K)/(K_RA + (K_LA/DELTA_T));
    W_K := (TE_K - TEXT_K + (K_J/DELTA_T)*W_K)/((K_J/DELTA_T) + K_B);
    VCEM_K := K_KB * W_K;
    TE_K := K_KE*I_K
    ;
END_FUNCTION_BLOCK
```

En este bloque de función se ha implementado el modelo discretizado del sistema lineal del motor de corriente continua.

Este bloque será llamado desde el OB35 del programa del PLC2, como se muestra a continuación:

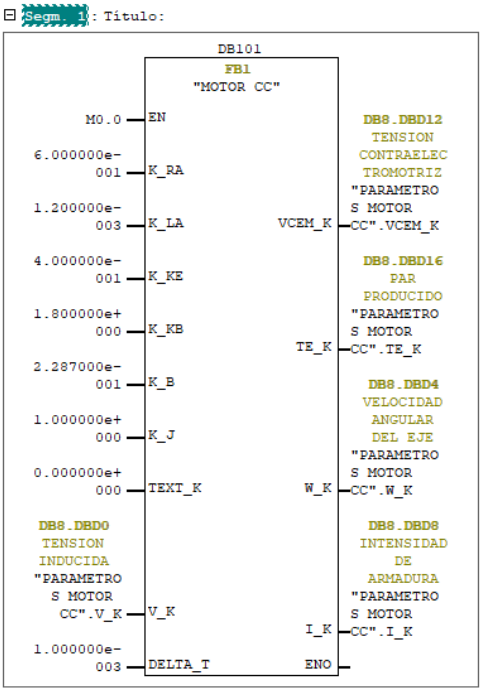


Figura 4-22. Llamada a la función MOTOR CC.

Este OB se ejecuta cíclicamente cada 100 ms. Los resultados de esta función serán observados a través de la aplicación de usuario que se mostrarán en próximos capítulos de este proyecto.

4.8. Bloque de organización principal del PLC2

El OB1 del programa del PLC2 contiene únicamente el escalado y desescalado de las entradas/salidas analógicas. A continuación se muestra este programa:

4.8.2 Segmento 1

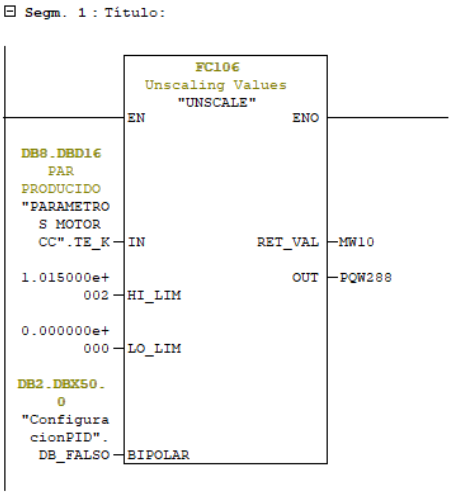


Figura 4-23. Segmento 1 del OB1 en el PLC2.

El segmento 1 desescala valores tipo REAL de entre 0.0 y 100.0 de la salida de par producido de la función *SIM_CC* a valores del rango de salida en tensión de PQW288.

Esta salida analógica está conectada a una de las entradas analógicas del PLC1, encargado del control del sistema simulado del motor de corriente continua.

4.8.3 Segmento 2

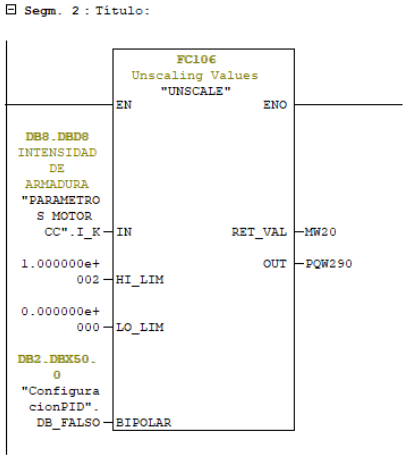


Figura 4-24. Segmento 2 del OB1 en el PLC2.

El segmento 2 desescala valores tipo REAL de entre 0.0 y 100.0 de la salida intensidad de la armadura de la función *SIM_CC* a valores del rango de salida en tensión de PQW290.

Esta salida analógica está conectada a una de las entradas analógicas del PLC1, encargado del control del sistema simulado del motor de corriente continua.

4.8.4 Segmento 3

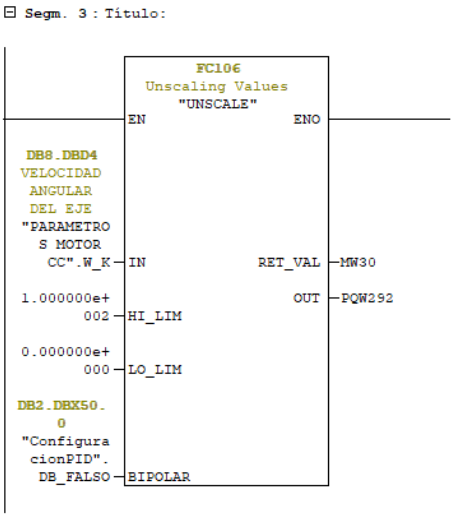


Figura 4-25. Segmento 3 del OB1 en el PLC2.

El segmento 3 desescala valores tipo REAL de entre 0.0 y 100.0 de la salida de velocidad angular del eje de la función *SIM_CC* a valores del rango de salida en tensión de PQW292.

Esta salida analógica está conectada a una de las entradas analógicas del PLC1, encargado del control del sistema simulado del motor de corriente continua.

4.8.5 Segmento 4

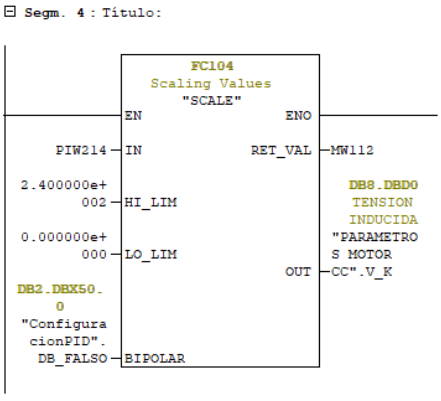


Figura 4-26. Segmento 4 del OB1 en el PLC2.

El segmento 4 escala la entrada analógica PIW214 a valores tipo REAL de entre 0.0 y 240.0.

El valor REAL de salida de la función de escalado es movido a la variable “PARAMETROS MOTORCC”.V_K

Esta entrada analógica está conectada a una de las salidas analógicas del PLC1, encargado del control del sistema simulado del motor de corriente continua.

5 APLICACIÓN DE CONFIGURACIÓN Y ADQUISICIÓN DE DATOS

Aunque llegues último en una carrera, siempre estarás por delante de quienes nunca se atrevieron a correrla.

Anónimo

Una vez configurados ambos PLC, será necesario en primer lugar, crear los tags OPC en el servidor a partir de las direcciones de memoria utilizadas en el PLC1. Con los tags creados en el servidor, podremos empezar a crear la aplicación con *MATLAB App Designer* y sus librerías OPC.

5.1. Creación y test de TAGs OPC en el servidor

A continuación se detalla el proceso de creación de etiquetas OPC en el servidor.

Para crear una etiuqueta o variable en el servidor OPC, hemos de situarnos sobre la conexión y el dispositivo que hemos creado en capítulos anteriores.

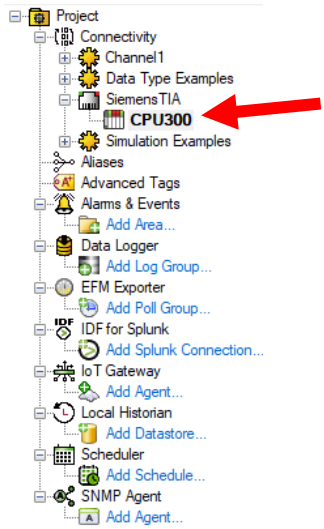


Figura 5-1. Desplegable de configuración del servidor OPC.

Una vez situados sobre el controlador, crearemos un nuevo tag haciendo click sobre el icono de la barra superior *New tag*.

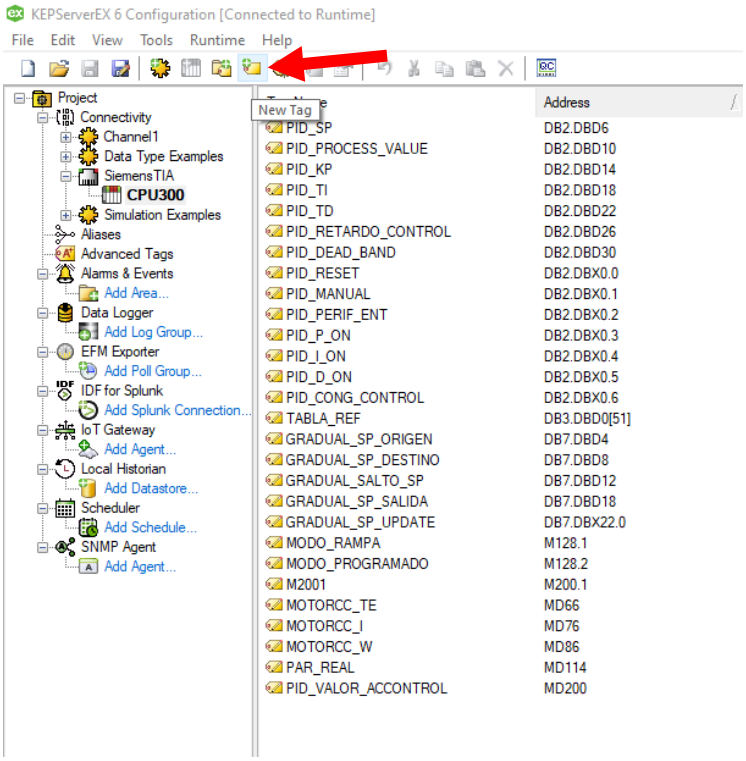


Figura 5-2. Icono de creación de nuevas etiquetas en KEPServerEX.

Esta acción nos llevará a una ventana de creación de la variable.

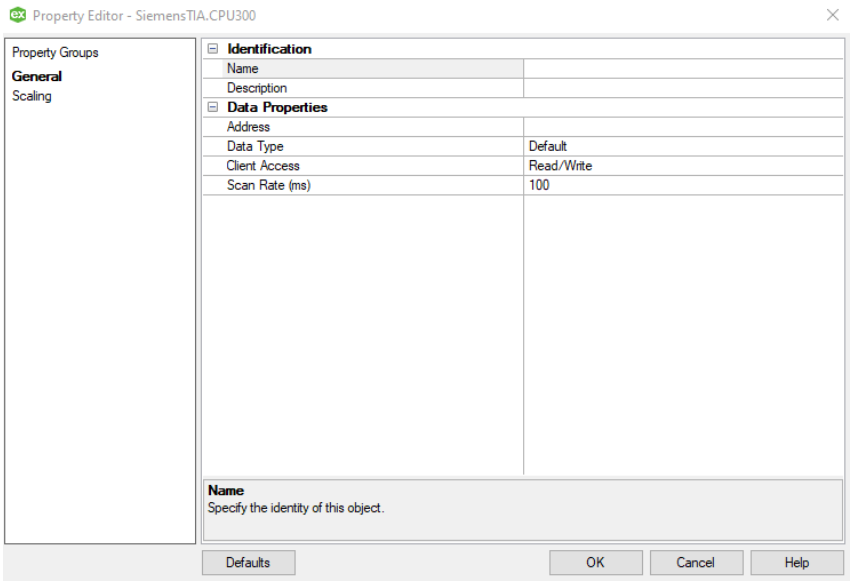


Figura 5-3. Ventana de creación de nuevas etiquetas en KEPServerEX.

Ahora bastará con poner un nombre a la variable, poner su dirección (la misma que tiene en el PLC), configurar el tipo de dato y su scan rate.

Ejemplo 5.1. *Creación de la variable GRADUAL_SP_SALIDA*

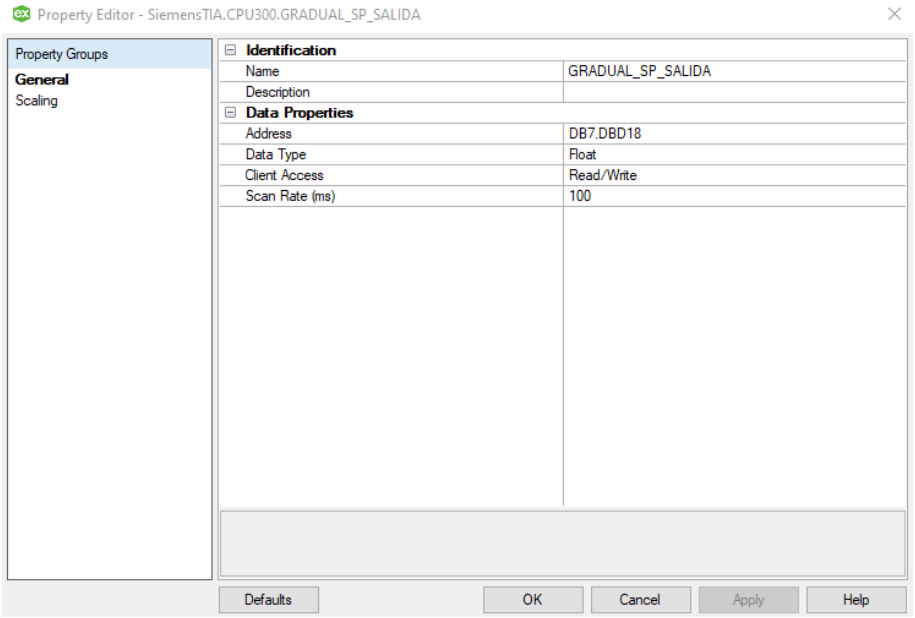


Figura 5-4. Ventana de propiedades de nuevas etiquetas en KEPServerEX.

Observamos que, dado que KEPServerEX dispone de drivers de comunicación para controladores de SIEMENS de serie S7-300, también reconocerá su direccionamiento. Así podemos con la misma sintaxis, apuntar directamente a la variable con la misma dirección que tiene en el PLC.

En este ejemplo se ha creado la variable ***GRADUAL_SP_SALIDA*** con dirección ***DB7.DBD18***. Esta variable en el PLC es de tipo REAL. Las variables de tipo REAL las configuraremos en el servidor OPC como FLOAT.

El campo *Client Access* lo dejaremos por defecto, dado que la serie S7-300 de SIEMENS solo soporta la versión *OPC DA (Data Access)*, lo que implica que nuestras variables no podrán ser protegidas contra escritura. Esto no ocurre para controladores que permitan OPC UA, donde sí podremos definir el nivel de acceso para cada variable.

El *scan rate* ha sido configurado en *100 ms*. Este valor es el mínimo posible para conseguir una comunicación estable.

Este proceso lo repetiremos para cada una de las variables que queramos añadir al servidor OPC.

Una vez configurado el tag, haremos click en *OK* y comprobaremos que la variable ha sido creada y estamos leyendo valores. Para hacer esto se recomienda utilizar la herramienta OPC Quick Client integrada en KEPServerEX.

Para iniciar esta herramienta haremos click en el siguiente icono:

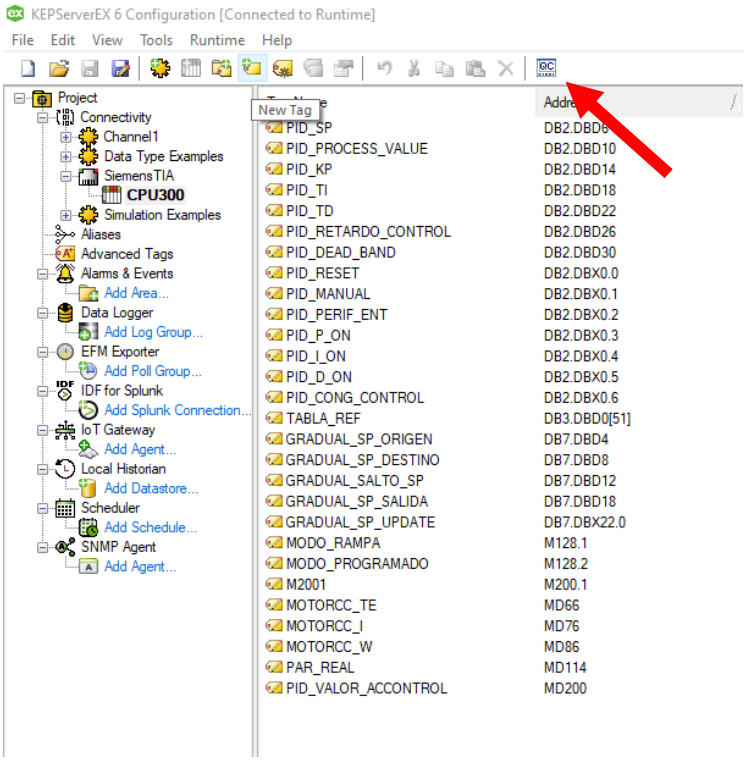


Figura 5-5. Icono de OPC Quick Client en KEPServerEX.

Una vez iniciado *OPC Quick Client*, iremos a la carpeta *SiemensTIA.CPU300* y observaremos las etiquetas creadas con su *Item ID* (muy importante conocer este ID para configurar el cliente OPC de MATLAB). En la columna *Value* observaremos el valor actual de la variable. Si este se muestra *Unknown* es denota que se ha producido un error en la lectura.

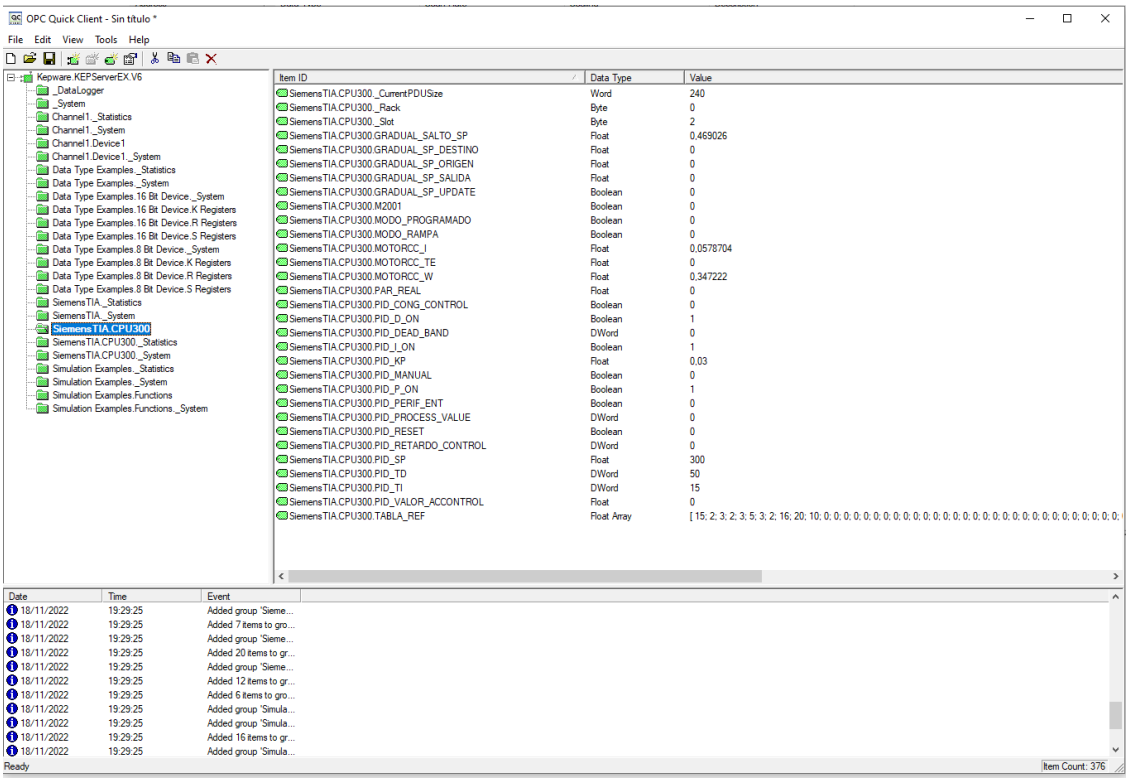


Figura 5-6. Ventana de visualización de tags en OPC Quick Client.

Una vez comprobado que la comunicación entre el servidor OPC y el PLC se ha establecido correctamente y se han creado todos los tags necesarios, podremos pasar a la creación de la aplicación de usuarios en *App Designer*.

5.2. Introducción a la aplicación

5.2.2 ¿Qué es MATLAB App Designer?

MATLAB App Designer es una extensión del software MATLAB que permite crear aplicaciones profesionales con una interfaz intuitiva para los usuarios.

Las aplicaciones creadas con *App Designer* serán ejecutadas sobre MATLAB, lo que nos permitirá usar todas sus librerías, así como su potencia para cálculos.

En este proyecto, la aplicación creada con App Designer brindará al usuario una herramienta para configuración rápida del controlador de motor de continua sin necesidad de utilizar el software de programación del PLC, además de incluir otras funcionalidades, como son la visualización de datos del experimento realizado en gráficas o exportarlos en formatos .xlsx o .mat.

5.2.3 Diagrama de flujo de la aplicación

A continuación, se muestra el diagrama de flujo de la aplicación a nivel usuario, desde el inicio de la aplicación hasta su cierre.

Se recomienda a la hora de utilizar la aplicación, seguir este flujo de trabajo y configuración, ya que, asegurará el correcto funcionamiento de ella.

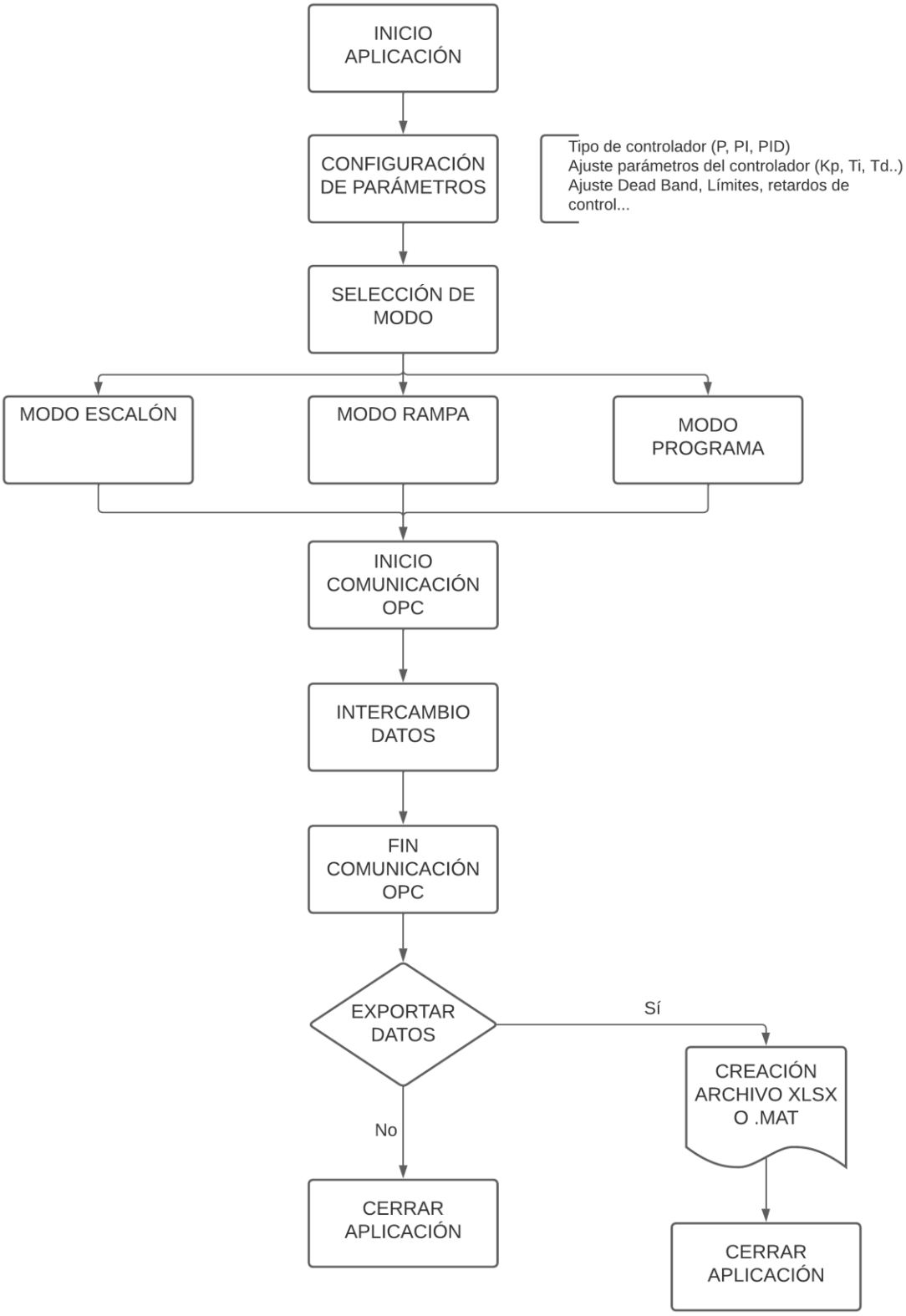


Figura 5-7. Diagrama de flujo de uso de la aplicación de control.

5.2.4 Pantalla principal de la aplicación

En la siguiente figura se muestra la pantalla principal de la aplicación. Esta pantalla muestra los siguientes elementos:

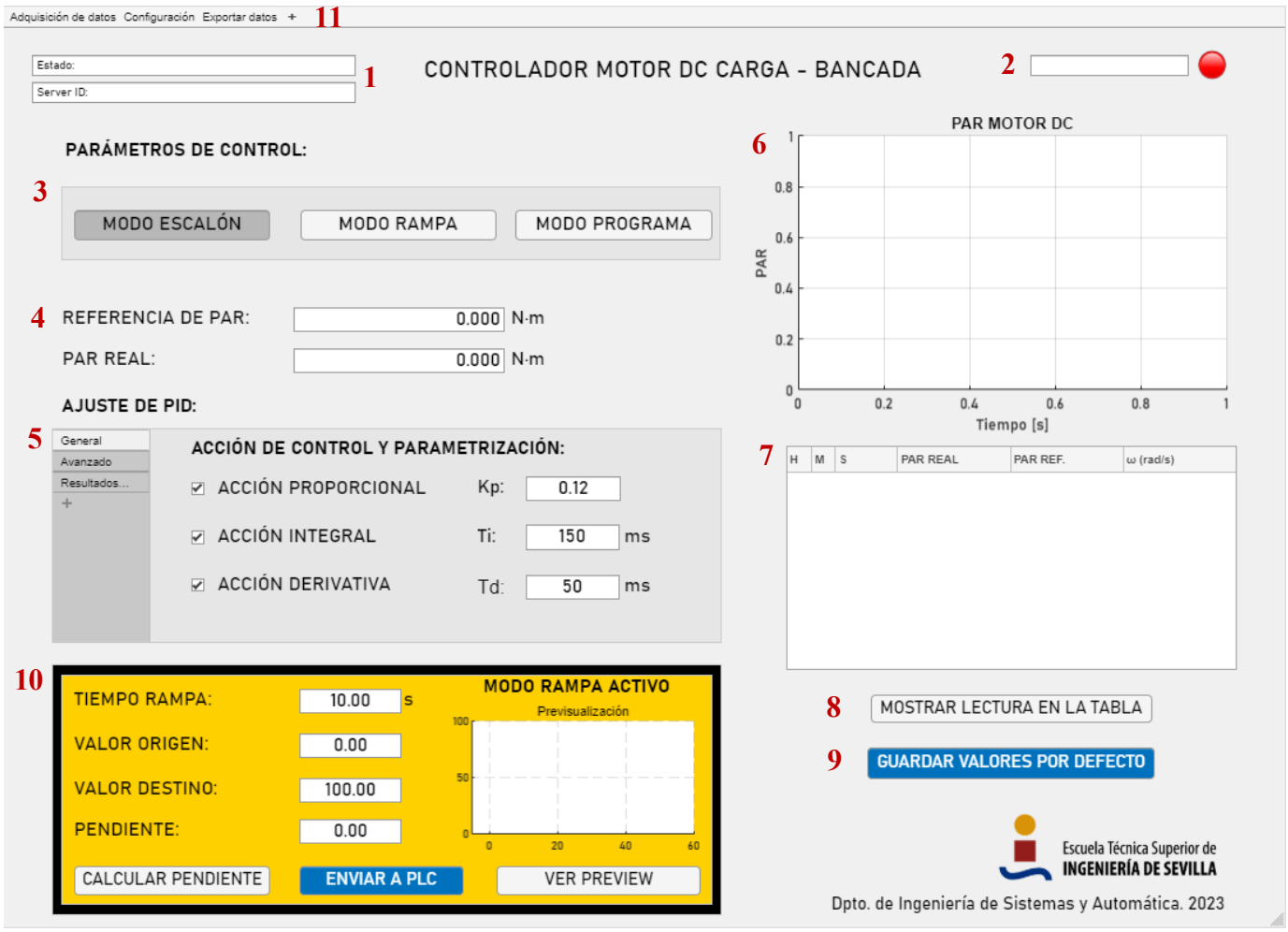


Figura 5-8. Ventana principal de la aplicación de control.

1) **Información de la conexión:** El display *Estado* muestra el estado de la conexión del cliente OPC. El display *Servidor* muestra el nombre que identifica al servidor configurado.

2) **Estado de la simulación:** Muestra si se está ejecutando o no el programa.

3) **Selector de modo:** Permite conmutar entre el *modo escalón*, *modo rampa* o *modo programa*.

4) **Información de par:** Mostrada en dos campos, un campo de entrada de datos *Referencia de par* y un segundo campo que muestra el *Par real* producido por el motor de corriente continua.

5) **Ajuste de PID:** Pestaña de ajustes del controlador. Los valores introducidos en esta ventana son cargados al iniciar la comunicación OPC en el controlador.

6) **Gráfica:** En esta gráfica se mostrarán dinámicamente los valores de par referencia y par real durante la ejecución del programa.

7) **Tabla de datos:** Al pulsar el botón 8) *MOSTRAR LECTURA EN LA TABLA,* se mostrarán los valores adquiridos durante la ejecución del programa. Esta tabla muestra una marca temporal que indica cuándo se recibió el dato, el valor de par real, el valor de par referencia y la velocidad angular del eje para ese instante expresada en rad/s.

8) **Botón MOSTRAR LECTURA EN TABLA:** Permite mostrar los datos obtenidos hasta el momento de la pulsación en la tabla.

9) **Botón GUARDAR VALORES POR DEFECTO:** Al pulsar este botón se guardarán todos los valores actuales de los campos de configuración y se establecerán dichos valores como por defecto para ser cargados al iniciar la aplicación.

10) **Modo rampa:** Esta ventana solo aparece si se activa el modo rampa. Nos permite introducir el valor destino de referencia y el tiempo de rampa, calcular la pendiente de la recta de cambio de referencia y enviar los datos al PLC.

11) **Barra superior:** Formada por tres campos desplegables.

    a. **Adquisición de datos:** Desde esta pestaña podremos iniciar la comunicación OPC y detener la comunicación OPC.

    b. **Configuración:** Contiene el botón *Parámetros de OPC*. Al pulsar este botón se abre una ventana emergente que permite configurar la dirección del host y la ID del servidor OPC.

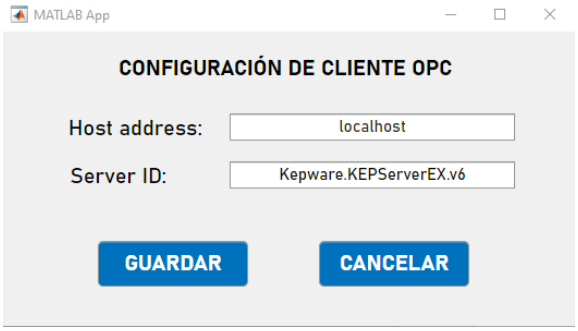


Figura 5-9. Ventana de configuración de cliente OPC de la aplicación de control.

    c. **Exportar datos:** Permite ejecutar el algoritmo de exportación de los datos adquiridos en formato .mat o .xlsx. Los ficheros creados en la exportación se generan en el directorio principal de la aplicación con nombre con formato "XX-MES-AÑO.mat".

5.3. Programación de la aplicación

A la hora de programar aplicaciones en *App Designer* podremos hacer distinción entre dos partes fundamentales de este proceso. Por un lado, diseñar la interfaz visual de la aplicación y por otro, programarla desde la vista de código.

Dado que diseñar la interfaz visual es un proceso trivial arrastrándo componentes de librería a la pantalla de la aplicación, nos centraremos en la vista de código y cómo ha sido programada.

Las aplicaciones de App Designer son programadas con lenguaje orientado a objetos y parte de este código se genera automáticamente al introducir un nuevo elemento gráfico.

5.3.2 Atributos de componentes de la aplicación

A lo largo del desarrollo de esta aplicación se han creado varios atributos de la clase app con el fin de utilizarlos de manera que sean accesibles desde cualquier parte de la aplicación y varias funciones puedan modificarlos.

Estos atributos se listan a continuación y son accesibles utilizando la sintáxis app.[nombre_atributo]:

Código 5.3.1 Atributos de app creados por el usuario.

```
properties (Access = public) % Atributos públicos
    sim = 0; % Atributo de bucle de ejecucion de la app
    enable = 0; % Atributo de modo de funcionamiento de la app
    update = 0; % Atributo para boton Enviar a PLC
end
properties (Access = private) % Atributos privados
    tabla_ref % Atributo con referencias para modo programado
    grap_x = [] % Atributo eje x de la gráfica
    grap_y = [] % Atributo eje y de la gráfica
    grap_y2 = [] % Atributo 2 eje y de la gráfica
    timestamp_hora = [] % Atributo de hora de la marca temporal
    timestamp_min = [] % Atributo de minutos de la marca temporal
    timestamp_seg = [] % Atributo de segundos de la marca temporal
    ref = [] % Atributo de valor de referencia
    i_armadura = [] % Atributo de valor de la intensidad de armadura
    w_eje = [] % Atributo de valor de la velocidad angular
    DialogApp % Atributo ventana emergente configuracion OPC
    main_host = "" % Atributo Host OPC
    main_server_id = "" % Atributo Server ID OPC
    grupo_config % Atributo de grupo OPC para R/W de configuracion
    grupo_adq % Atributo de grupo OPC para R/W de adquisicion datos
end
```

5.3.3 Funciones de componentes de la aplicación

En esta aplicación se han creado dos funciones principales asociadas a componentes de la aplicación.

- **Función *runOPC*:** encargada de la lectura y escritura de las variables OPC durante el ciclo de comunicación con el servidor.
    - El intercambio de información entre cliente y servidor se da en un bucle de ilimitadas iteracciones mientras se cumpla la condición de comunicación activa *app.sim = true.*
    - La función ejecutará un bucle de lectura/escritura según el modo de funcionamiento seleccionado.

Código 5.3.2 Función *runOPC.*

```
function
runOPC(app,~,~,item2,item3,item6,item7,item8,item9,item10,item11,item12,item14,
item15,item17,item18,item19,item21,item22, item30)
 app.sim = 1;
 app.sim_status.Value = "Simulación: STOP";
 %Escritura INICIAL de TAGs
 switch(app.enable)
        case 0
         write(item14,0);
         write(item22,0);
         write(item10,app.checkP.Value); %P ON
         write(item11,app.checkI.Value); %I ON
         write(item12,app.checkD.Value); %D ON
         write(item7,app.KpEditField.Value); %Kp
         write(item8,app.TiEditField.Value); %Ti
         write(item9,app.TdEditField.Value); %Td
         write(item3,app.pid_deadband.Value);
         write(item30,app.tabla_ref.PROGRAMA);
         write(item4,app.pid_limite_sup.Value); %Limite Superior
         write(item5,app.pid_limite_inf.Value); %Limite Inferior
         write(item6,app.pid_retardo_control.Value); %Retardo Control
        case 1
         write(item14,1); %Rampa ON
         write(item22,0);
         write(item10,app.checkP.Value); %P ON
         write(item11,app.checkI.Value); %I ON
         write(item12,app.checkD.Value); %D ON
         write(item7,app.KpEditField.Value); %Kp
         write(item8,app.TiEditField.Value); %Ti
         write(item9,app.TdEditField.Value); %Td
```

```
% item15 = additem(app.grupo, 'SiemensTIA.CPU300.GRADUAL_SP_UPDATE');
% item16 = additem(app.grupo, 'SiemensTIA.CPU300.GRADUAL_SP_ORIGEN');
% item17 = additem(app.grupo, 'SiemensTIA.CPU300.GRADUAL_SP_DESTINO');
% item18 = additem(app.grupo, 'SiemensTIA.CPU300.GRADUAL_SALTO_SP');
          case 2
           write(item14,0);
           write(item22,1);
           write(item10,app.checkP.Value); %P ON
           write(item11,app.checkI.Value); %I ON
           write(item12,app.checkD.Value); %D ON
           write(item7,app.KpEditField.Value); %Kp
           write(item8,app.TiEditField.Value); %Ti
           write(item9,app.TdEditField.Value); %Td
           write(item3,app.pid_deadband.Value);
           write(item30,app.tabla_ref.PROGRAMA);
           write(item6,app.pid_retardo_control.Value);
end
tic
switch(app.enable)
            case 0
              while(app.sim)
                 app.sim_status.Value = "Simulación: RUNNING";
                 write(item19,app.par_referencia.Value);
                 par_refer = app.par_referencia.Value;

                 datosPLC = read(app.grupo_adq,'cache');
                 par_producido = double(datosPLC(1).Value);
                 app.par_real.Value = par_producido;
                 app.referencia_min.Value = par_producido;
                 app.grap_x(end+1,1) = toc;
                 app.grap_y(end+1,1) = par_producido;
                 %app.i_armadura(end+1,1) = double(datosPLC(6).Value);
                 app.w_eje(end+1,1) = double(datosPLC(6).Value);
                 app.timestamp_hora(end+1,1) = datosPLC(1).TimeStamp(1,4);
                 app.timestamp_min(end+1,1) = datosPLC(1).TimeStamp(1,5);
                 app.timestamp_seg(end+1,1) = datosPLC(1).TimeStamp(1,6);
                 app.grap_y2(1,end+1) = par_refer;
                 app.ref(end+1,1) = par_refer;
                 plot(app.grafica, app.grap_x, app.grap_y,app.grap_x,
app.grap_y2);
                 pause(0.2)
```

```
                    end
                case 1
                    while(app.sim)
                       if(app.update == 1)
                            write(item15,1);
                            write(item18,app.salto_referencia.Value);
                            app.update = 0;
                            write(item15,0);
                        end
                    app.sim_status.Value = "Simulación: RUNNING";
                    write(item17,app.par_referencia.Value);
                    par_refer = app.par_referencia.Value;
                    datosPLC = read(app.grupo_adq,'cache');
                    par_producido = double(datosPLC(1).Value);
                    app.par_real.Value = par_producido;
                    app.referencia_min.Value = par_producido;
                    app.grap_x(end+1,1) = toc;
                    app.grap_y(end+1,1)= par_producido;
                    %%app.i_armadura(end+1,1) = double(datosPLC(6).Value);
                    app.w_eje(end+1,1) = double(datosPLC(6).Value);
                    app.timestamp_hora(end+1,1) = datosPLC(1).TimeStamp(1,4);
                    app.timestamp_min(end+1,1) = datosPLC(1).TimeStamp(1,5);
                    app.timestamp_seg(end+1,1) = datosPLC(1).TimeStamp(1,6);
                    app.grap_y2(1,end+1) = par_ref;
                    app.ref(end+1,1) = par_refer;
                    plot(app.grafica, app.grap_x, app.grap_y,app.grap_x,
app.grap_y2);

                    pause(0.2)
                   end
                case 2
                    while(app.sim)
                        app.sim_status.Value = "Simulación: RUNNING";
                        datosPLC = read(app.grupo_adq,'cache');
                        app.par_referencia.Value = double(datosPLC(5).Value);
                        par_producido = double(datosPLC(1).Value);
                        app.par_real.Value = par_producido;
                        app.referencia_min.Value = par_producido;
                        app.grap_x(end+1,1) = toc;
                        app.grap_y(end+1,1) = par_producido;
                        %app.i_armadura(end+1,1) = double(datosPLC(6).Value);
```

```
                        app.w_eje(end+1,1) = double(datosPLC(6).Value);
                        app.timestamp_hora(end+1,1) =
datosPLC(1).TimeStamp(1,4);
                        app.timestamp_min(end+1,1) = datosPLC(1).TimeStamp(1,5);
                        app.timestamp_seg(end+1,1) = datosPLC(1).TimeStamp(1,6);
                        app.grap_y2(1,end+1) = app.par_referencia.Value;
                        app.ref(end+1,1) = app.par_referencia.Value;
                        plot(app.grafica, app.grap_x, app.grap_y,app.grap_x,
app.grap_y2);
                        pause(0.2)
                    end
            end
            app.sim_status.Value = "Simulación: STOP";
        end
```

La función recibe como argumentos todos los items correspondientes a los objetos de los TAGs OPC. Estos items han creados en la función startupFcn(app), que se ejecuta al lanzar la aplicación.

- **Función *updateConfig***: encargada guardar la configuración del host, el ID del servidor y los valores de configuración del controlador en el fichero *save.mat* de configuración por defecto.

Código 5.3.3 Función *updateConfig*.

```
function updateConfig(app,confighost,configserverid)
            app.main_host = confighost; % Leemos valor host actual
            app.main_server_id = configserverid; % Leemos valor server_id actual
            Host = app.main_host;
            ServerID = app.main_server_id;
            Kp = app.KpEditField.Value; % Leemos valor Kp actual
            Ti = app.TiEditField.Value; % Leemos valor Ti actual
            Td = app.TdEditField.Value; % Leemos valor Td actual
            % Guardamos los valores en el archivo save.mat
            save('save.mat','Kp','Ti','Td','Host','ServerID');
        end
```

5.3.4 Funciones asociadas a eventos.

A continuación, se detallan las funciones asociadas a eventos del programa. Estas funciones se ejecutan cuando ocurre un determinado evento, como puede ser el inicio de la aplicación, la pulsación de un determinado botón, el cambio de valor de un campo de entrada, etc…

- **Función *startupFcn***: encargada inicializar los campos de los displays a partir de los valores leídos del fichero de configuración *save.mat*.

Código 5.3.4 Función *startupFcn.*

```
function startupFcn(app)
            app.EstadoConexion.Value = "Estado: No conectado al controlador." ;
            app.IDServer.Value = "Servidor: Esperando a establecer conexión";
            memoria = load('save.mat'); %Cargamos los valores por defecto
            app.tabla_ref = load('programa.mat');
            app.main_host = memoria.Host;
            app.main_server_id = memoria.ServerID;
            app.KpEditField.Value = memoria.Kp;
            app.TiEditField.Value = memoria.Ti;
            app.TdEditField.Value = memoria.Td;
            app.connection_status.Color = [1.00,0.00,0.00] ; % Indicador conx.
            app.sim = 0;
            app.panel_rampa.Visible = 0;
            app.panel_negro.Visible = 0;
end
```

- **Función *run_comSelected:*** encargada de establecer la conexión entre el cliente OPC de MATLAB y el servidor OPC KEPServerEX.
    - Se ejecuta cuando se pulsa el botón de inicio de comunicación OPC.
    - Una vez establecida la comunicación, se crean dos grupos de lectura/escritura de variables, lo que optimiza la lectura y mejora los tiempos. Estos grupos con GRUPO_CONFIGURACIÓN (app.grupo_config) y GRUPO_ADQUISICIÓN (app.grupo_adq).
    - Posteriormente se crean los items para cada variable y se añaden al grupo pertinente.
    - Por último, se llamará a la función runOPC y, una vez finalizado el bucle de intercambio de datos, el cliente se desconecta del servidor.

Código 5.3.5 Función *run_comSelected.*

```
function run_comSelected(app, event)
            da = opcda(string(app.main_host), string(app.main_server_id));
            connect(da);
```

```
            estado_conn = da.Status;
            %Lectura del estado de la conexion con el servidor OPC
            if(estado_conn == "connected")
                estado_conn_esp = "Estado: Conectado al controlador.";
                app.connection_status.Color = [0.00,1.00,0.00] ;
            else
                estado_conn_esp = "Estado: Desconectado del controlador.";
                app.connection_status.Color = [1.00,0.00,0.00] ;
            end
            app.EstadoConexion.Value = estado_conn_esp;
            app.IDServer.Value = "Servidor: " + da.ServerID;

            app.grupo_config = addgroup(da,'GRUPO_CONFIGURACION');
            app.grupo_adq = addgroup(da,'GRUPO_ADQUISICION');

            %% Definimos las variables a leer dentro del GRUPO CONFIGURACION
            item3 = additem(app.grupo_config,
'SiemensTIA.CPU300.PID_DEAD_BAND');
            item4 = additem(app.grupo_config,
'SiemensTIA.CPU300.PID_LIM_SUP');
            item5 = additem(app.grupo_config,
'SiemensTIA.CPU300.PID_LIM_INF');
            item6 = additem(app.grupo_config,
'SiemensTIA.CPU300.PID_RETARDO_CONTROL');
            item7 = additem(app.grupo_config,
'SiemensTIA.CPU300.PID_KP');
            item8 = additem(app.grupo_config,
'SiemensTIA.CPU300.PID_TI');
            item9 = additem(app.grupo_config,
'SiemensTIA.CPU300.PID_TD');
            item10 = additem(app.grupo_config,
'SiemensTIA.CPU300.PID_P_ON');
            item11 = additem(app.grupo_config,
'SiemensTIA.CPU300.PID_I_ON');
            item12 = additem(app.grupo_config,
'SiemensTIA.CPU300.PID_D_ON');
            item14 = additem(app.grupo_config,
'SiemensTIA.CPU300.MODO_RAMPA');
            item22 = additem(app.grupo_config,
'SiemensTIA.CPU300.MODO_PROGRAMADO');
            item30 = additem(app.grupo_config,
```

```
'SiemensTIA.CPU300.TABLA_REF');


            item2 = additem(app.grupo_adq,
'SiemensTIA.CPU300.MOTORCC_TE');


            item15 = additem(app.grupo_adq,
'SiemensTIA.CPU300.GRADUAL_SP_UPDATE');
            item17 = additem(app.grupo_adq,
'SiemensTIA.CPU300.GRADUAL_SP_DESTINO');
            item18 = additem(app.grupo_adq,
'SiemensTIA.CPU300.GRADUAL_SALTO_SP');
            item19 = additem(app.grupo_adq,
'SiemensTIA.CPU300.GRADUAL_SP_SALIDA');
            %item20 = additem(app.grupo_adq, 'SiemensTIA.CPU300.MOTORCC_I');
            item21 = additem(app.grupo_adq,
'SiemensTIA.CPU300.MOTORCC_W');


            %Ejecutamos el proceso de intercambio de datos
runOPC(app,app.grupo_config,app.grupo_adq,item2,item3,item4,item5,item6,item7,it
em8,item9,item10,item11,item12,item14,item15,item17,item18,item19,item21,item22,
item30);
            app.connection_status.Color = [1.00,0.00,0.00] ;
            %Cerramos la conexión
            disconnect(da);
```

- **Función *stop_comSelected:*** encargada de asignar *app.sim = 0* para detener el bucle de comunicación.
    - Se ejecuta cuando se pulsa el botón de detención de comunicación OPC.

Código 5.3.6 Función stop_*comSelected.*

```
function stop_comSelected(app, event)
                app.sim = 0;
end
```

- **Función *checkDValueChanged:*** encargada de habilitar/deshabilitar el campo Td en pantalla.
    - Se ejecuta al pulsar la checkbox de Td.

Código 5.3.7 Función checkDValueChanged.

```
function checkDValueChanged(app, event)
            value = app.checkD.Value;
            if(value)
```

```
                app.TdEditField.Enable = 1;
            else
                app.TdEditField.Enable = 0;
            end
        end
```

- **Función *checkIValueChanged:*** encargada de habilitar/deshabilitar el campo Ti en pantalla.
    - Se ejecuta al pulsar la checkbox de Ti.

Código 5.3.8 Función checkIValueChanged.

```
function checkIValueChanged(app, event)
            value = app.checkI.Value;
            if(value)
                app.TiEditField.Enable = 1;
            else
                app.TiEditField.Enable = 0;
            end
        end
```

- **Función *exportarXLSXSelected:*** encargada de exportar los datos obtenidos en formato .xlsx.
    - Se ejecuta al pulsar la opción de exportar en formato .xlsx.

Código 5.3.9 Función *exportarXLSXSelected.*

```
function exportarXLSXSelected(app, event)
            date = datetime('today');
            filename = string(date) + ".xlsx";
            tdata = table(app.timestamp_hora,app.timestamp_min,app.timestamp_seg,
app.grap_y, app.ref, app.w_eje, 'VariableNames',
{'Horas','Minutos','Segundos','Par Real','Par Referencia','W Eje'});
            writetable(tdata,filename);
end
```

- **Función *exportarMATSelected:*** encargada de exportar los datos obtenidos en formato .mat.
    - Se ejecuta al pulsar la opción de exportar en formato .mat.

Código 5.3.10 Función *exportarMATSelected.*

```
function exportarMATSelected(app, event)
            tdata =
table(app.timestamp_hora,app.timestamp_min,app.timestamp_seg, app.grap_y,
app.ref, app.w_eje, 'VariableNames', {'Horas','Minutos','Segundos','Par
Real','Par Referencia','W Eje'});
            date = datetime('today');
            filename = string(date) + ".mat";
            save(filename,'tdata');
end
```

- **Función *mostrar_tablaPushed:*** encargada de mostrar los valores obtenidos en la tabla de la pantalla principal.
    - Se ejecuta al pulsar el botón *MOSTRAR LECTURA EN LA TABLA*.

Código 5.3.11 Función *mostrar_tablaPushed.*

```
function mostrar_tablaPushed(app, event)
            tdata =
table(app.timestamp_hora,app.timestamp_min,app.timestamp_seg, app.grap_y,
app.ref, app.w_eje, 'VariableNames', {'Horas','Minutos','Segundos','Par
Real','Par Referencia','W Eje'});
            app.tabla_datos.Data = tdata;
end
```

- **Función *ConexionOPCMenuSelected:*** lanza la ventana emergente de configuración de la conexión con el servidor OPC.
    - Se ejecuta al pulsar el botón *Parámetros de OPC*.

Código 5.3.12 Función *ConexionOPCMenuSelected.*

```
function ConexionOPCMenuSelected(app, event)
            app.DialogApp =
pop_up_configuracionOPC(app,app.main_host,app.main_server_id);
end
```

- **Función *calcular_rampaButtonPushed:*** calcula los parámetros necesarios para el modo rampa, como los saltos de referencia necesarios o la pendiente de la rampa.
    - Se ejecuta al pulsar el botón *Calcular rampa*.

Código 5.3.13 Función *calcular_rampaButtonPushed.*

```
function calcular_rampaButtonPushed(app, event)
            n_escalon = (app.tiempo_total.Value*1000/4)/app.ancho_escalon.Value;
            n_escalon_calc = n_escalon;
            if(n_escalon_calc/2 ~= 0)
                n_escalon_calc = n_escalon_calc+1;
            end
            app.n_escalones.Value = n_escalon_calc;
            n_escalon_subida = n_escalon_calc;
            app.salto_referencia.Value = abs((app.referencia_max.Value-
app.referencia_min.Value)/n_escalon_subida);


            x_final = app.tiempo_total.Value;
            x_inicial = 0;
            m = (app.referencia_max.Value - app.referencia_min.Value)/(x_final -
x_inicial);
            app.pendiente.Value = m;
end
```

- **Función *ver_previsualizacionButtonPushed:*** realiza un plot de previsualización de la rampa.
    - Se ejecuta al pulsar el botón *Ver previsualización*.

Código 5.3.14 Función *ver_previsualizacionButtonPushed.*

```
function ver_previsualizacionButtonPushed(app, event)
            x = 0:0.1:(app.tiempo_total.Value +10);
            x_final = app.tiempo_total.Value;
            x_inicial = 0;
            m = (app.referencia_max.Value - app.referencia_min.Value)/(x_final -
x_inicial);
            syms X
            f1 = app.referencia_min.Value;
            f2(X) = (m*X)+app.referencia_min.Value;
            f3 = app.referencia_max.Value;
```

```
            y=f1.*(x<=0)+f2(x).*(x>0 &
x<app.tiempo_total.Value)+f3.*(x>=app.tiempo_total.Value);
            plot(app.preview,x,y)
end
```

- **Función *EnviarAPLCButtonPushed:*** cambia el valor de app.update para indicar al bucle de comunicación que hay que actualizar de manera asíncrona las variables implicadas en la rampa.
    - Se ejecuta al pulsar el botón *Enviar a PLC*.

Código 5.3.15 Función *EnviarAPLCButtonPushed.*

```
function EnviarAPLCButtonPushed(app, event)
            if(app.update==1)
                app.update = 0;
            else
                app.update = 1;
            end
 end
```

- **Función *GuardarValoresPushed:*** guarda en el archivo *save.mat* el valor actual de la configuración del controlador y la información del servidor OPC.
    - Se ejecuta al pulsar el botón *GUARDAR VALORES POR DEFECTO*.

Código 5.3.16 Función *GuardarValoresPushed.*

```
function GuardarValoresPushed(app, event)
            Kp = app.KpEditField.Value;
            Ti = app.TiEditField.Value;
            Td = app.TdEditField.Value;
            Host = app.main_host;
            ServerID = app.main_server_id;
            save('save.mat','Kp','Ti','Td','Host','ServerID')
end
```

- **Función *SeleccionGruposSelection:*** Funcionamiento de los botones del selector de modo de funcionamiento.
    - Se ejecuta al pulsar uno de los botones del grupo.
    - Si el botón seleccionado es el de *modo rampa,* se muestra la ventana de modo rampa.

Código 5.3.17 Función *SeleccionGruposSlection.*

```
function SeleccionGruposSelectionChanged(app, event)
            value = app.MODO_RAMPA.Value;
            value2 = app.MODO_PROGRAMA.Value;
            if(value)
                app.panel_rampa.Visible = 1;
                app.panel_negro.Visible = 1;
                app.enable = 1;
            elseif(value2)
                app.panel_rampa.Visible = 0;
                app.panel_negro.Visible = 0;
                app.enable = 2;
            else
                app.panel_rampa.Visible = 0;
                app.panel_negro.Visible = 0;
                app.enable = 0;
            end
end
```

5.4. Ejecución con modo escalón

A continuación se muestra un ejemplo de ejecución del programa en modo escalón con los pasos seguidos:

1) Iniciamos la aplicación y ajustamos el controlador con los parámetros deseados.

Figura 5-10. Ventana principal de la aplicación de control sin conexión.

2) Una vez ajustado el controlador, iniciamos la conexión OPC haciendo click en *Adquisición de datos -> Comunicación OPC -> Iniciar.*

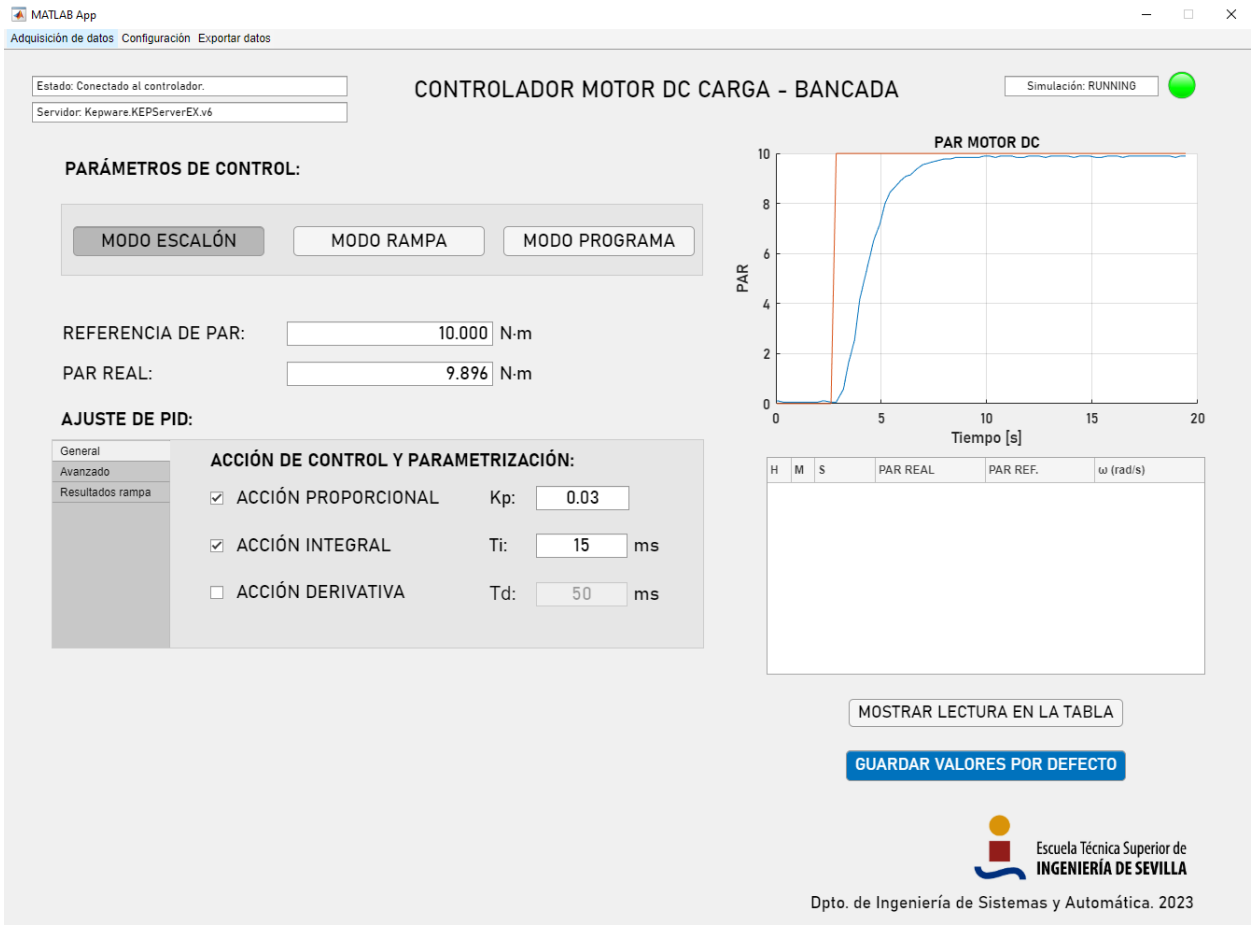


Figura 5-11. Ventana principal de la aplicación de control en modo escalón.

Observamos que los indicadores de conexión han cambiado y la conexión se encuentra activa. Si pulsamos el botón *Mostrar lectura en la tabla* aparecerán los valores obtenidos en la comunicación, como se muestra a continuación.

Figura 5-12. Ventana de la aplicación de control en modo escalón con tabla de datos.

Una vez se quiera finalizar la comunicación, bastará con hacer click en Adquisición de datos -> Comunicación OPC -> Detener y acto seguido se detendrá la comunicación.

A continuación, podremos exportar los datos obtenidos por ejemplo, en formato .mat, haciendo click en *Exportar datos -> Guardar en .xlsx* y hecho esto, se habrá generado en directorio de instalación de la aplicación un archivo con la fecha de exportación y extensión .mat.

Editor - MotorCCDiscreto.m

tdata

2666x6 table

	1 Horas	2 Minutos	3 Segundos	4 Par Real	5 Par Referencia	6 Intensidad
1	14	21	38.2590	0	3	0
2	14	21	38.7710	0.4630	3	1.2731
3	14	21	39.1960	0.6944	3	2.0255
4	14	21	39.6060	1.0417	3	2.9514
5	14	21	40.0190	1.2153	3	3.2986
6	14	21	40.4200	1.3889	3	3.7037
7	14	21	40.8600	1.2731	3	3.4144
8	14	21	41.2500	1.3889	3	3.5880
9	14	21	41.6440	1.4468	3	3.8773
10	14	21	42.5090	1.6204	3	4.2245
11	14	21	42.9160	1.5046	3	4.2245
12	14	21	43.2940	1.6204	3	4.3403
13	14	21	43.6940	1.6204	3	4.4560
14	14	21	44.0790	1.7940	3	4.6875
15	14	21	44.4610	1.6782	3	4.5139
16	14	21	45.0140	1.7361	3	4.5718
17	14	21	45.6160	1.7940	3	4.8032
18	14	21	45.9990	1.9676	3	5.0926
19	14	21	46.5880	1.7940	3	4.6875
20	14	21	46.9810	1.9097	3	4.9769
21	14	21	47.4150	1.9097	3	5.0926
22	14	21	47.8060	2.0255	3	5.2662

Figura 5-13. Vista del archivo .mat exportado en MATLAB.

5.5. Ejecución en modo rampa

A continuación se muestra un ejemplo de ejecución del programa en modo rampa con los pasos seguidos:

1) Iniciamos la aplicación y ajustamos el controlador con los parámetros deseados.

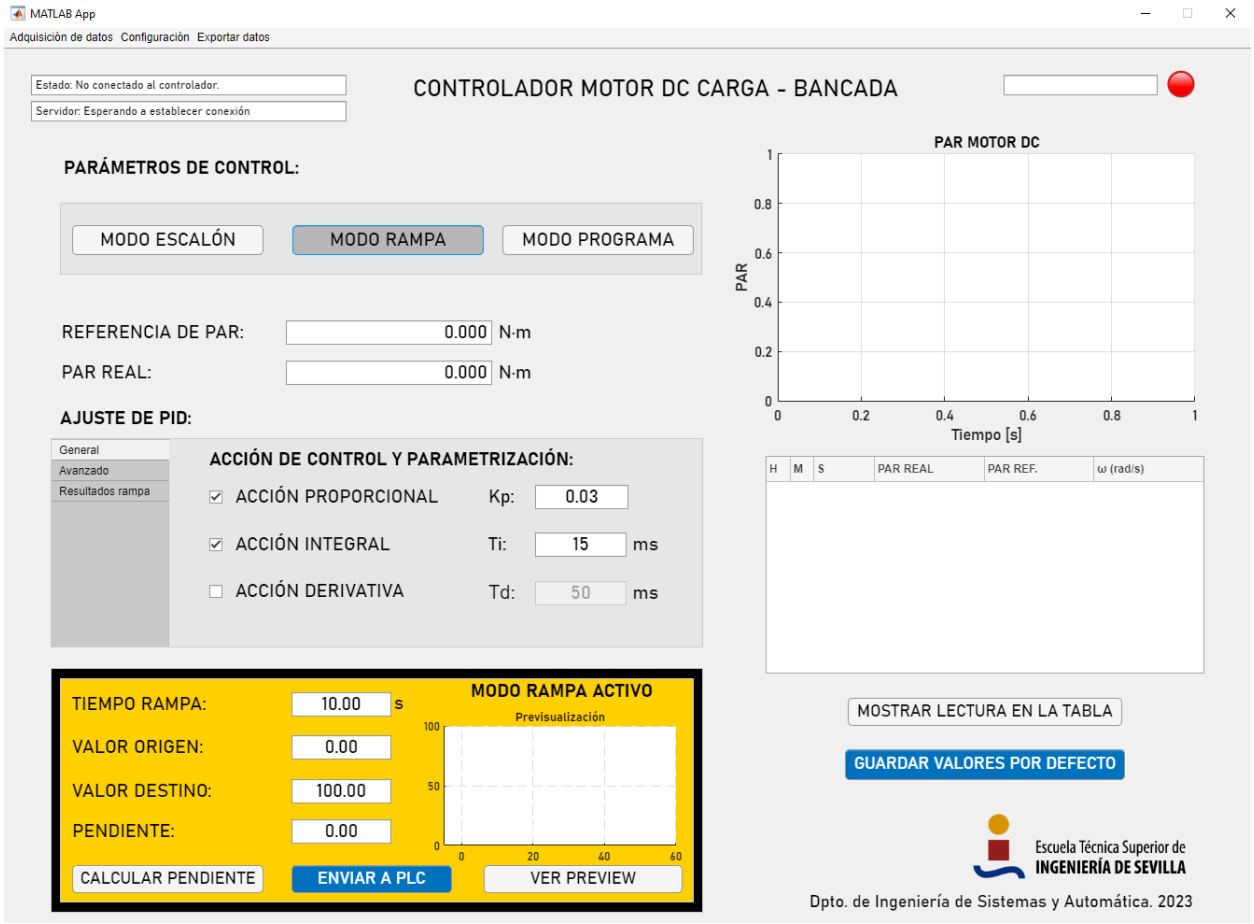


Figura 5-14. Ventana principal de la aplicación de control en modo rampa sin conexión.

1) Una vez ajustado el controlador, iniciamos la conexión OPC haciendo click en *Adquisición de datos -> Comunicación OPC -> Iniciar.*

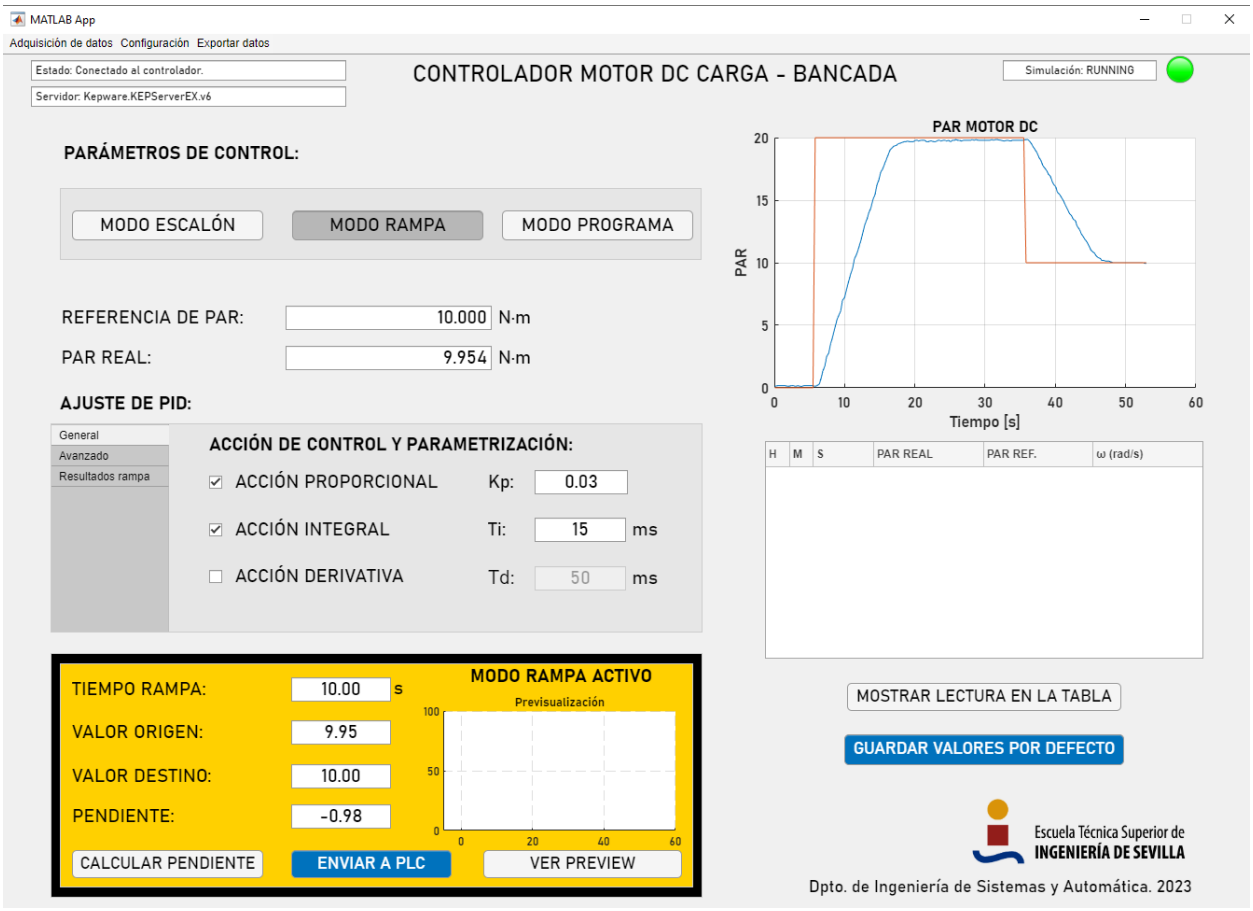


Figura 5-15. Ventana principal de la aplicación de control en modo rampa.

Obsérvese que ha aparecido una nueva ventana de MODO RAMPA ACTIVO que nos servirá para ir ajustando la rampa en función de lo que necesitemos.

Una vez se quiera finalizar la comunicación, bastará con hacer click en *Adquisición de datos -> Comunicación OPC -> Detener* y acto seguido se detendrá la comunicación.

5.6. Ejecución en modo programado

Para trabajar en modo programado, es indispensable inicialmente modificar el fichero *programa.mat* situado en el directorio principal de la aplicación. Este fichero contiene una columna de 51 elementos tipo *double* y cada uno de estos elementos, será un valor de referencia que podemos programar.

Este fichero posteriormente es cargado por nuestro programa y transmitido al inicio como una variable tipo *array* a nuestro controlador, que irá recorriendo dicho array secuencialmente y aplicando los cambios de frecuencia cada intervalo de tiempo.

PROGRAMA

51x1 double

	1	2
1	0	
2	0	
3	5	
4	15	
5	0	
6	5	
7	3	
8	2	
9	16	
10	20	
11	10	
12	0	
13	0	
14	0	
15	0	
16	0	
17	0	
18	0	
19	0	
20	0	
21	0	
22	0	
23	0	
24	0	
25	0	

Figura 5-16. Vista del archivo programa.mat exportado en MATLAB.

El tiempo entre cambios de referencia actual es de *10 segundos* (T#10S), pudiendo ser modificado cambiando el valor de la variable de entrada *PT* del generador de pulsos de contaje en *el segmento 3* del *OB1* del *PLC1*.

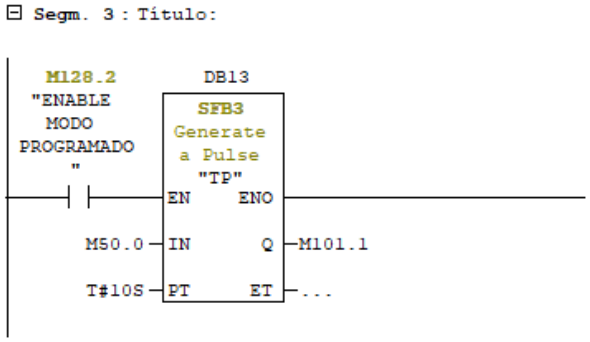


Figura 5-17. Seg. 3 del OB1 del PLC1 para configuración de tiempo de cambio de referencia.

Para modificar el fichero .mat, será necesario abrirlo con MATLAB como veremos a continuación.

A la hora de insertar los valores de referencia, se recomienda dejar siempre los primeros valores a 0 por motivos de pérdida al iniciar la comunicación y también, el último valor del array.

Una vez modificado el fichero, bastará con iniciar la aplicación en modo programado y el controlador irá ejecutando los cambios de referencia automáticamente. Una vez recorridos todos los elementos del array, el controlador empezará de nuevo por el primer índice, haciendo que el programa se ejecute cíclicamente.

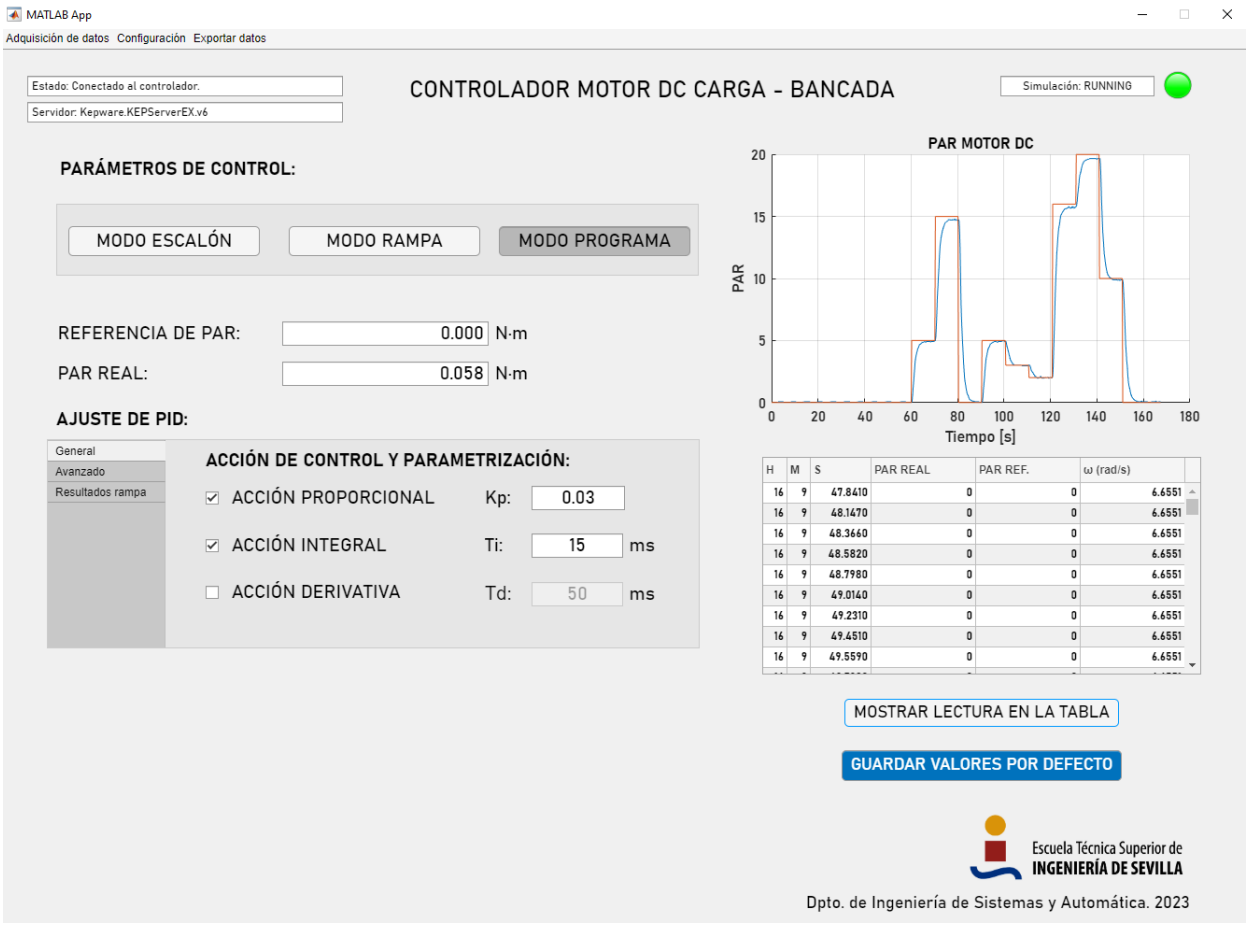


Figura 5-18. Ventana principal de la aplicación de control en modo programado.

Una vez se quiera finalizar la comunicación, bastará con hacer click en *Adquisición de datos -> Comunicación OPC -> Detener* y acto seguido se detendrá la comunicación.

6 RESULTADOS OBTENIDOS Y FUTURAS LÍNEAS DE TRABAJO

Per áspera ad astra.

- Séneca -

Como resultado de este proyecto, se han conseguido establecer las bases y realizar las primeras pruebas de un sistema de control para la bancada de ensayos del laboratorio de Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla, sirviendo este proyecto como punto de partida para futuros trabajos en esta bancada.

6.2. Resultados

Como resultados de este proyecto se han logrado configurar dos controladores lógicos programables SIEMENS S7-300 y su periferia para realizar simulaciones de un motor de corriente continua y su control usando funciones de control continuo.

Además, se ha creado una comunicación OPC entre el PLC encargado del control del motor de corriente continua y una aplicación de usuario desarrollada para el control y la adquisición de datos de este sistema, permitiendo tener control del motor en tiempo cuasi-real y con capacidad de adquirir datos y almacenarlos para su posterior estudio.

H	M	S	PAR REAL	PAR REF.	ω (rad/s)
16	11	15.4110	0.0579	0	7.5231
16	11	15.6340	0.0579	0	7.5231
16	11	15.8530	0.0579	0	7.5231
16	11	16.0700	0.0579	0	7.5231
16	11	16.2780	0.0579	0	7.5231
16	11	16.5080	0.0579	0	7.5231
16	11	16.7270	0.0579	0	7.5231
16	11	16.9440	0.0579	0	7.5231
16	11	17.1570	0.0579	0	7.5231

Figura 6-1. Tabla de datos obtenidos por la aplicación durante un ensayo.

Experimentalmente, se ha obtenido una tasa de refresco de las variables OPC de entorno a los *0,2 segundos*, por lo que se ha hecho especial incapié en optimizar los procesos de lectura/escritura de dichas variables.

6.3. Futuras líneas de trabajo

Como futuras líneas de trabajo, se propone la sustitución del PLC2 por el motor real de corriente continua, así como el diseño e instalación de un armario de control donde situar el controlador y demás elementos de la bancada de ensayos.

REFERENCIAS

[1] Bordóns Alba, Ruiz Arahal, M., & Limón Marruedo, D. (2007). Teoría de sistemas. Universidad de Sevilla, Departamento de Ingeniería de Sistemas y Automática.

[2] Díaz Cárdenas, A. (2017). Diseño e implementación de una carga electrónica, basada en motor de DC para una bancada de ensayos de accionamientos multifásicos. Trabajo Fin de Máster. Universidad de Sevilla, Departamento de Ingeniería de Sistemas y Automática.

[3] Siemens Industry (2006) SIMATIC. Programar en STEP 7. (Edición 03/2006). https://cache.industry.siemens.com/dl/files/056/18652056/att_70833/v1/S7prv54_s.pdf

[4]Diaz.I. (s.f.) Modelado de un motor CC. Área de Ingeniería de Sistemas y Automática, Escuela Politécnica Superior de Ingeniería de Gijón. Universidad de Oviedo. http://isa.uniovi.es/~idiaz/ADSTel/Practicas/ModeladoMotorCC.html

[5] Kepware KEPServerEX (2022) KEPServerEX Manual. https://www.kepware.com/getattachment/e1943820-ef3c-4932-b055-4ef2a80ab863/kepserverex-manual.pdf