UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MÁSTER EN INGENIERÍA INDUSTRIAL

DISEÑO DE UN SISTEMA DE CONTROL BASADO EN PC MEDIANTE SOFTWARE TWINCAT 3 DE UNA PLANTA DE GENERACIÓN DE ENERGÍA ELÉCTRICA DE 160MW SIMULADA MEDIANTE HIL E IMPLEMENTADA EN PYTHON

AUTOR: GUILLERMO JOAQUÍN ALITE CEREZUELA

TUTOR: JUAN MANUEL HERRERO DURÁ

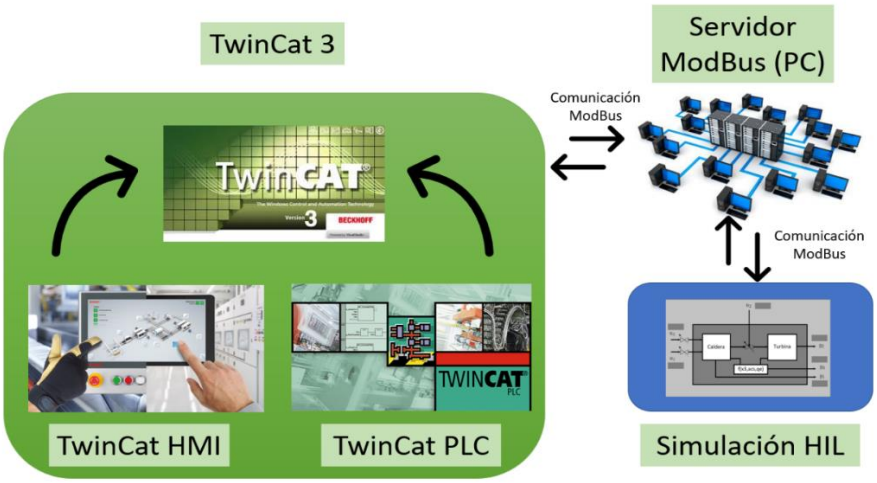
Curso Académico: 2019-20

RESUMEN

El objetivo de este TFM es el diseño del sistema de control de una planta de generación de energía eléctrica, compuesta por una caldera, una turbina y tres válvulas regulables, que pueden llegar a generar hasta 160 MW. El modelo que se ha utilizado para la simulación está disponible en la literatura (*benchmark*), el cual se implementa con *Python*. Esta simulación se ejecuta mediante *HIL (Hardware In the Loop)* a tiempo real, lo que significa que este modelo calculará las salidas del proceso exactamente en un tiempo de muestreo establecido dentro del HIL. Para la comunicación entre el proceso simulado y el sistema de control se ha utilizado el protocolo *ModBus*, con una estructura cliente-servidor.

La parte principal de este proyecto es el diseño de un sistema de control basado en PC, mediante el software *TwinCat 3*. Que permite convertir cualquier PC compatible, basado en Windows, en un controlador de tiempo real que puede integrar PLC, Motion, HMI, C++ y Matlab en un solo software y una sola CPU. Para este trabajo se ha creado una instancia de PLC, que se encargará de la comunicación con el servidor mediante *ModBus* y de calcular las acciones de control utilizando PIDs. Este PLC ha sido combinado con una pantalla táctil (HMI), con el objetivo de permitir a los usuarios acceder a los datos y eventos del sistema, así como la posibilidad de cambiar algunos parámetros del mismo.

Finalmente se ha llevado a cabo un diseño óptimo de los parámetros de los PIDs mediante el uso de algoritmos genéticos en *Matlab/Simulink.*

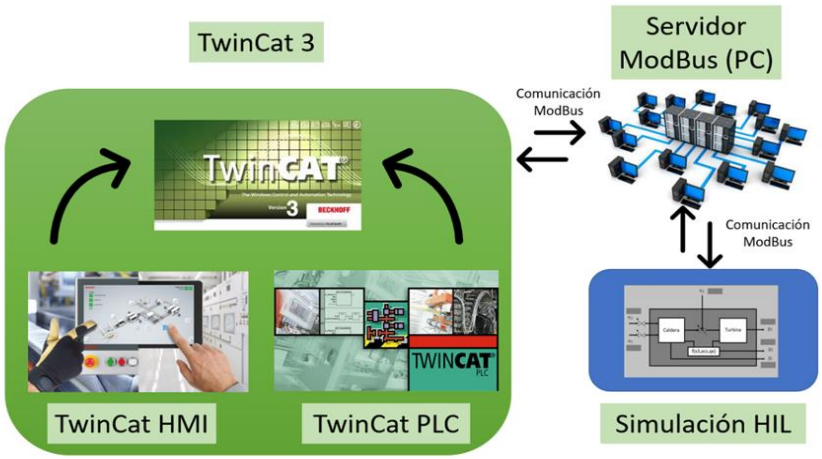


RESUM

L'objectiu d'aquest TFM és el disseny del sistema de control d'una planta de generació d'energia elèctrica, consistent en una caldera, una turbina i tres vàlvules regulables, que poden generar fins a 160 MW. El model que s'ha utilitzat per a la simulació està disponible en la literatura (*benchmark),*que està implementada amb *Python*. Aquesta simulació s'executa amb HIL *(Hardware In the Loop) en* temps real, el que significa que aquest model calcularà les sortides del procés exactament en un temps de mostreig fixat en el HIL. El protocol *Modbus*,amb una estructura client-servidor, s'ha utilitzat per a la comunicació entre el procés simulat i el sistema de control.

La part principal d'aquest projecte és el disseny d'un sistema de control basat en PC, utilitzant el programari *TwinCat* 3. Li permet convertir qualsevol PC basat en Windows compatible en un controlador en temps real que pot integrar PLC, motion, HMI, C++ i Matlab en un sol programari i una sola CPU. Per a aquest treball s'ha creat una instància PLC, que s'encarregarà de comunicar-se amb el servidor utilitzant *Modbus* i calculant les accions de control utilitzant pids. Aquest PLC s'ha combinat amb una pantalla tàctil (HMI), amb l'objectiu de permetre als usuaris accedir a les dades i esdeveniments del sistema, així com la possibilitat de canviar alguns paràmetres del sistema.

Finalment, s'ha dut a terme un disseny òptim dels paràmetres del PIDs mitjançant l'ús d'algorismes genètics en *MATLAB/Simulink.*

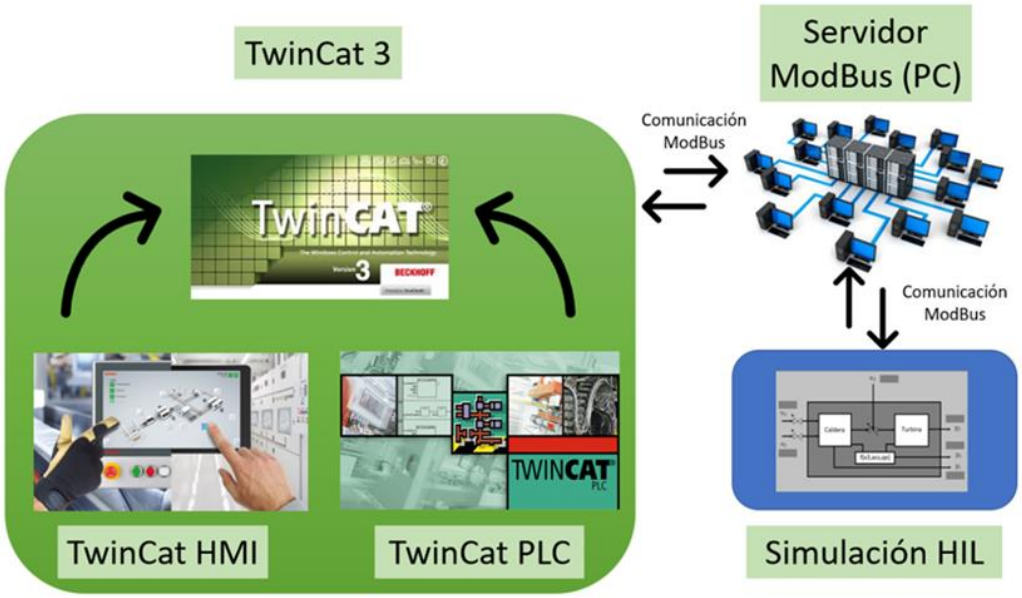


ABSTRACT

The objective of this TFM is the design of the control system of an electric power generation plant, consisting of a boiler, a turbine and three adjustable valves, which can generate up to 160 MW. The model that has been used for simulation is available in the literature (*benchmark),* which is implemented with Python. This simulation runs using HIL *(Hardware In the Loop)* in real time, which means that this model will calculate the process outputs exactly at a sampling time set within the HIL. *ModBus* protocol, with a client-server structure, has been used for communication between the simulated process and the control system.

The main part of this project is the design of a PC-based control system, using *TwinCat* 3 software. It allows any compatible Windows-based PC to be turned into a real-time controller that can integrate PLC, Motion, HMI, C++ and *Matlab* into a single software and a single CPU. For this work a PLC instance has been created, which will take care of communicating with the server using Modbus and calculating the control actions using PIDs. This PLC has been combined with a touch screen (HMI), with the aim of allowing users to access the data and events of the system, as well as the possibility to change some parameters of the system.

Finally, optimal design of parameters of PIDs has been carried out using genetic algorithms in *Matlab/Simulink.*



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MÁSTER EN INGENIERÍA INDUSTRIAL

DISEÑO DE UN SISTEMA DE CONTROL BASADO EN PC MEDIANTE SOFTWARE TWINCAT 3 DE UNA PLANTA DE GENERACIÓN DE ENERGÍA ELÉCTRICA DE 160MW SIMULADA MEDIANTE HIL E IMPLEMENTADA EN PYTHON

DOCUMENTO Nº1:

MEMORIA DEL PROYECTO

AUTOR: GUILLERMO JOAQUÍN ALITE CEREZUELA

TUTOR: JUAN MANUEL HERRERO DURÁ

Curso Académico: 2019-20

Índice documento nº1

Índice de ilustraciones

Índice de tablas

Índice de código en python

1 OBJETO DEL PROYECTO

El objetivo principal de este trabajo es diseñar un sistema de control mediante el *software TwinCat 3*, con el fin de obtener un programa que permita realizar distintas funciones, tales como:

- Controlar mediante PIDs la planta de generación de energía que se ha propuesto para este trabajo.
- Interactuar con el programa mediante una pantalla táctil.
- Comunicarse mediante protocolo *Modbus*.
- Realizar un histórico de los datos procesados.

Para poder realizar un proyecto funcional, se requiere de un servidor de *Modbus* y una simulación del proceso de la planta. De manera que el programa que calcula la simulación y el programa de control desarrollado en *TwinCat 3* se comunicarán mediante protocolo *Modbus* a través del servidor.

2 JUSTIFICACIÓN DEL PROYECTO

Este proyecto es de carácter académico, tras cursar el Máster Universitario en Ingeniería Industrial (MUII) y realizar la especialidad en control de procesos, automatización y robótica, el alumno ha de recopilar todos los conocimientos necesarios para abordar un Trabajo Final de Máster (TFM). El objeto de este trabajo es muy interesante ya que en este último año ha tratado en profundidad muchos aspectos relacionados con el diseño de PIDs, distintas estructuras de control y SCADA, por lo que, tras la propuesta del tutor de este documento, se decidió abordar este proyecto.

Además, como cualquier otro trabajo académico, busca enriquecer los conocimientos del alumno, y aplicar los ya estudiados con anterioridad. A lo largo del diseño se han conseguido conocimientos relativos a *TwinCat 3, Python,* servidores *Modbus* y su protocolo de comunicación.

Otra razón para justificar este proyecto es el propio *software TwinCat 3*, es un programa que no se ha utilizado nunca en el departamento, y puede servir como base para futuros proyectos que se realicen sobre esta plataforma. Es un software que difiere con lo que está establecido en la industria. Lo más usual es

encontrar PLCs que controlen los procesos, *TwinCat* se basa en ordenadores con SO *Windows*, de manera que cualquier usuario que tenga conocimientos sobre *Windows* es muy sencillo que se habitúe a estos ordenadores, conocidos como ordenadores industriales. Estos ordenadores tienen las características básicas para ejecutar *Windows* y los programas que ejecute *TwinCat,* y se pueden añadir módulos de todo tipo para añadir funcionalidades. En el caso de que este programa controlará una caldera real, sería necesario un módulo que permitiera la conexión para el protocolo Modbus.

3 ANTECEDENTES

A continuación, se explican los conceptos necesarios para comprender el apartado 4 DESARROLLO DE LA SOLUCIÓN ADOPTADA, para abordarlo se requieren conocimientos sobre:

- Hardware In the Loop (HIL).
- Protocolo de comunicación ModBus TCP.
- Entorno de programación Python.
- Entrono de programación *TwinCat* 3.
- Algoritmos genéticos.

3.1 HARDWARE IN THE LOOP (HIL)

Hardware In the Loop (HIL) es una técnica en la que se simula una planta real mediante modelos matemáticos, obteniendo fácilmente un sistema con el que probar controladores sin necesidad de sensores, actuadores, maquinaria etc… (National Instuments, 2020). El controlador se conecta al HIL, a partir de los valores de salida del proceso simulado, se calculan las acciones de control que utilizará el HIL para actualizar las salidas tal y como lo haría el proceso real.

Ilustración 1 Esquema Hardware In the Loop

Esta técnica es muy útil en la fase de diseño de un sistema de control, permite no solo recrear la planta real mediante una simulación, sino también se pueden crear distintos escenarios para abordar otros tipos de control o simplemente ver cómo reacciona el controlador diseñado si ocurren ciertos sucesos. Además, reduce el coste y tiempo de las pruebas a realizar, ya que no es necesario el arranque de ninguna maquinaria ni requiere esperas de ningún tipo, basta con ejecutar el programa de simulación y activar el controlador.

En este trabajo se ha aprovechado esta técnica para simular una planta de generación de energía de 160 MW, esta planta consiste en una caldera y una turbina, con tres válvulas como variables manipuladas. Este proceso se simula utilizando las ecuaciones 1 a 8 (Tan, Márquez, Tongwen, & Jizhen, 2005) descritas en el apartado 4.1.1 y junto con el método de integración numérica de Euler generan las mismas salidas que la planta real, con la ventaja de que no es necesario tener una caldera y turbinas físicas.

La ejecución del HIL tiene que completarse en un periodo constante, es muy importante que se mida la cantidad de tiempo que tarda en realizar una iteración del proceso. Dado un tiempo de ejecución, el programa debe esperar el tiempo restante hasta que complete el periodo de tiempo diseñado para el HIL.

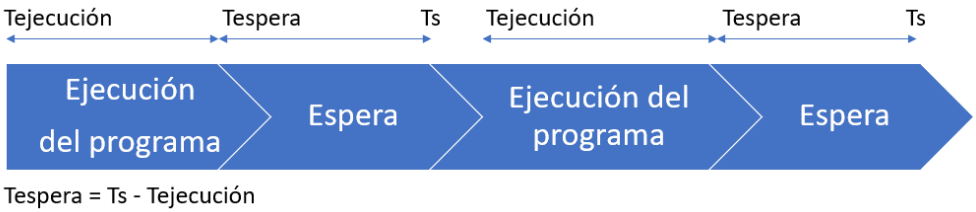


Ilustración 2 Esquema ejecución HIL

Para el cálculo de la simulación se ha utilizado el método de integración numérica de Euler, es una técnica que permite resolver ecuaciones diferenciales siempre y cuando se conozca un valor inicial. El punto de partida sería:

$$\frac{dy}{dt} = f(t, y) \qquad \mathrm{a} \leq \mathrm{t} \leq \mathrm{b} \qquad \text{Euler (1)}$$

$$y(a) = \alpha \quad \mathrm{a} \leq \mathrm{t} \leq \mathrm{b} \qquad \text{Euler (2)}$$

Primeramente, se establece el tamaño de paso al dividir el intervalo [a b] en n partes iguales, siendo t_0 = a y t_n = b, con un tamaño de:

$$h = \frac{t_n - t_0}{n} \qquad \text{Euler (3)}$$

A partir del valor inicial (t_0 = a $y_0 = \alpha$) y conociendo la ecuación diferencial Euler 1 que expresa la pendiente de la función y(t_i), se puede calcular cuánto vale la magnitud y en el siguiente instante t_1. Este proceso se repite hasta alcanzar el instante final deseado t_n = b, de esta manera se consigue aproximar el valor real y al valor de la función y(t_i) en cada instante de tiempo t_i (Arévalo Ovalle, Bernal Yermanos, & Posada Restrepo, 2017). En la Ilustración 2 se ha utilizado la nomenclatura usual en ingeniería, el tamaño de paso h suele asociarse a la variable Ts, es por esto que en las ecuaciones que se han implementado se hace uso de esta variable.

$$y_{i+1} = y_i + f(x_k, y_k)h \qquad \text{Euler (4)}$$

Si se reduce el tamaño del paso, la función simulada mediante Euler será más precisa, ya que dispone de una cantidad mayor de instantes de tiempo en los que se conoce la salida, pero no siempre es conveniente reducir al máximo el tamaño de paso. En tareas a tiempo real el tamaño de paso es igual al tiempo en el que debe completarse la ejecución del programa, como ocurre en este trabajo, además una mayor cantidad de instantes a calcular también supone una mayor carga computacional para el simulador. En la Ilustración 3 se muestra un pequeño ejemplo de la aproximación mediante el método de integración numérica de Euler de una función polinómica, los puntos de la solución calculada por este método se aproximan mucho a los de la función real, pero los puntos intermedios difieren en mayor medida.

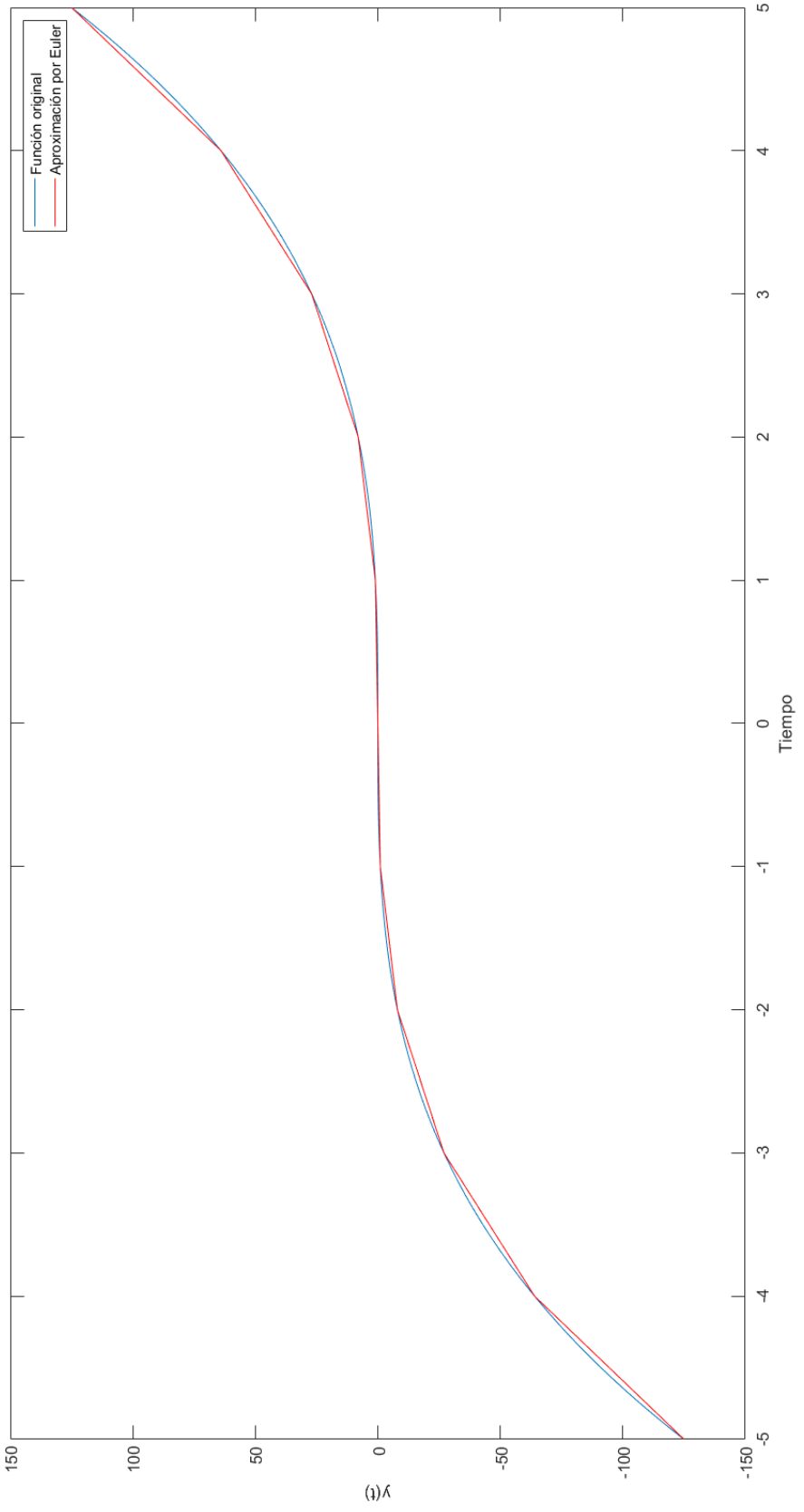


Ilustración 3 Comparación de la función original y la aproximación mediante Euler

3.2 MODBUS TCP

Para realizar la conexión entre la planta simulada y el sistema de control se ha utilizado el protocolo de comunicación *ModBus TCP* que está basado en la estructura cliente/servidor. *ModBus* es un protocolo muy utilizado y estandarizado en la industria, cuenta con distintas versiones de este, en concreto se ha utilizado *ModBus TCP/IP*, que como su propio nombre indica es utilizada en la comunicación vía redes *TCP/IP*.

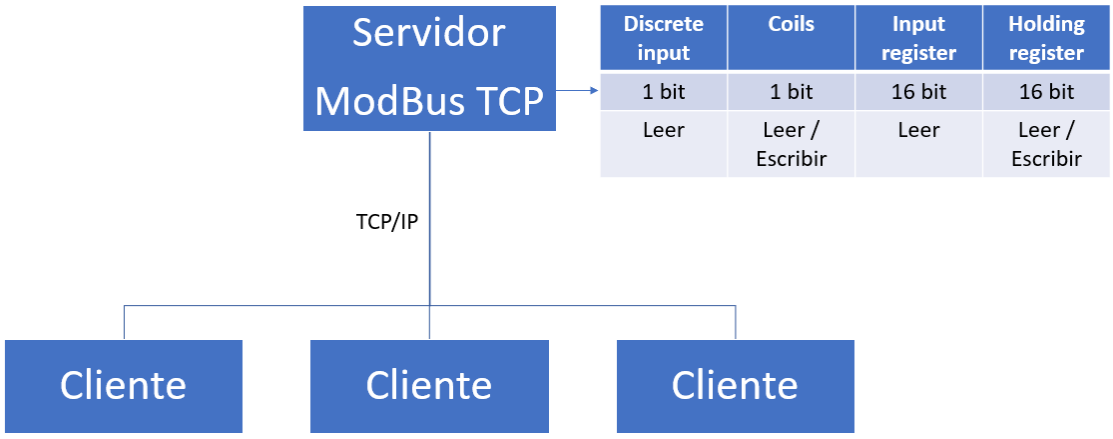


Ilustración 4 Esquema cliente/servidor ModBus TCP

Al servidor se le asigna una dirección IP y un puerto de conexión, el cliente debe conectarse al servidor mediante esta dirección, además debe especificar qué tipo de objeto, de los que ofrece el servidor, quiere acceder y de qué modo. Los objetos disponibles son los que se pueden apreciar en la tabla de la Ilustración 4 (Modbus Organization, 2006) (National Instruments, 2019).

- Discrete input: Entradas digitales de 1 bit:
- Coils: Salidas digitales de 1 bit.
- Input registers: Registros de entrada de 2 Bytes.
- Holding registers: Registros de retención de 2 Bytes.

Dado que en este proyecto la comunicación por *ModBus* se utiliza para comunicar el sistema de control y el HIL, se utilizarán *Holding registers* con capacidad para

escribir 16 bits, en este registro se representará un valor real codificado en el intervalo [0 65535]. Se ha establecido que los registros a utilizar corresponden con las direcciones 1-6 *holding registers*. El sistema de control leerá las entradas de los registros 1-3 y escribirá el valor de las salidas simuladas en 4-6. Por el contrario, el sistema de control lee las salidas simuladas 4-6 y escribe las acciones de control 1-3.

Ilustración 5 Lectura y escritura de registros

3.3 PROGRAMACIÓN EN PYTHON

Python es un lenguaje de programación de propósito general y código abierto, que se puede ejecutar en diversas plataformas (Linux, iOS, Windows…) y destaca por su sencillez y limpieza en el código.

Ilustración 6 Logotipo de Python (Python, 2020)

En este trabajo se ha utilizado la versión 2.7 de *Python* (Python, 2019) para Windows, junto con dos librerías para programar el servidor *ModBus* y el HIL:

- Librería para funciones de tiempo: *"time"* (Python, 2020).
- Librería para comunicación *ModBus: "pyModbusTCP"* (Lefebvre, 2018).

Es un lenguaje con una curva de aprendizaje muy rápida y con muchas referencias en internet, esto permite que el programador pueda comenzar con códigos sencillos y acabar programando códigos mucho más complejos en poco

tiempo. Al principio de este proyecto se utilizaba un programa HIL, con una entrada y una salida, a partir del correcto funcionamiento de este sistema simplificado, se fue implementando poco a poco el código final que simula la planta de generación con tres entradas y tres salidas.

El servidor *ModBus* es un programa relativamente sencillo, el cual tan solo utiliza la librería de *ModBus TCP* para iniciar un servidor con las cualidades explicadas en el apartado anterior. El programa HIL es más complejo, tiene que comunicarse mediante *ModBus* y además ejecutarse en un periodo constante, es por estas dos características que el HIL utiliza ambas librerías mencionadas anteriormente.

La librería "*time"* de Python permite al usuario utilizar funciones relacionadas con el tiempo, se puede detectar la hora del sistema en el que está ejecutándose el programa, parar durante un periodo de tiempo la ejecución del programa o detectar en que instante de tiempo se ha ejecutado una línea de código, entre otras funciones. Esta última característica es muy útil para el HIL, permite conocer el tiempo de ciclo de este, es esta función la que se usa para que el HIL se ejecute a tiempo real Ilustración 2. El cálculo del tiempo de ejecución se implementa con la función *time* y el tiempo de espera con la función *timer*.

La librería que contiene las funciones para la comunicación mediante *ModBus TCP* no viene en el paquete inicial de *Python,* para poder utilizarla hay que descargarla (Lefebvre, 2018). Seguidamente hay que instalarla siguiendo estos pasos:

1. Abrir la ventana de comandos de Windows como administrador (cmd).
2. Mediante el comando "cd" ir a la carpeta donde está el archivo de la librería.
3. Ejecutar el comando: python setup.py install

Si no aparece ningún error la librería se ha instalado correctamente y se pueden utilizar sus funciones. De estas librerías se han utilizado las funciones *ModbusServer* para crear el servidor, ModbusClient para crear el cliente, *read_holding_registers* y *write_single_register* para leer/escribir en los registros del servidor.

3.4 ENTORNO TWINCAT 3

TwinCat 3 es una plataforma de control desarrollada por la compañía Beckhoff (Beckhoff, 2020) que sigue la norma IEC61131, a diferencia del control convencional, este software utiliza PC con sistema operativo Windows, en vez de los usuales PLCs. Beckhoff ofrece un producto conocido como "PC industrial", es una gama de ordenadores con las funcionalidades básicas para ejecutar *TwinCat*, existe un amplio catálogo de "PCs industriales", se debe escoger el que proporcione las funcionalidades requeridas para el programa que se ha desarrollado en esta plataforma. Sin embargo, no impide que *TwinCat* se ejecute en ordenadores de terceros, es decir, con cualquier ordenador con SO Windows puede ejecutarse un programa de *TwinCat* siempre y cuando estén instalados los programas adecuados.

El entorno de programación se puede dividir en dos bloques:

- eXtended Automation Engineering (XAE).
- eXtended Automation Runtime (XAR).

Al instalar el paquete básico de *TwinCat 3* se instalan estos dos softwares, el primero (XAE) proporciona el entorno de programación propiamente dicho, en el que se puede crear proyectos y escribir el código que se necesite ejecutar, el segundo de ellos (XAR) proporciona un entorno en el que los proyectos de *TwinCat* pueden ser cargados, administrados o ejecutados en tiempo real. Esta es una de las cualidades más importantes de *TwinCat*, la capacidad de ejecutar su código a tiempo real. Cada programa de *TwinCat* puede ser asignado a una tarea, que se ejecuta al nivel de las tareas de Windows, además permite asignar a cada núcleo del procesador las distintas tareas, si se quiere que una tarea sea ejecutada en un núcleo específico, permitiendo explotar todo el potencial del procesador. El esquema de funcionamiento de XAE y XAR se muestra a continuación:

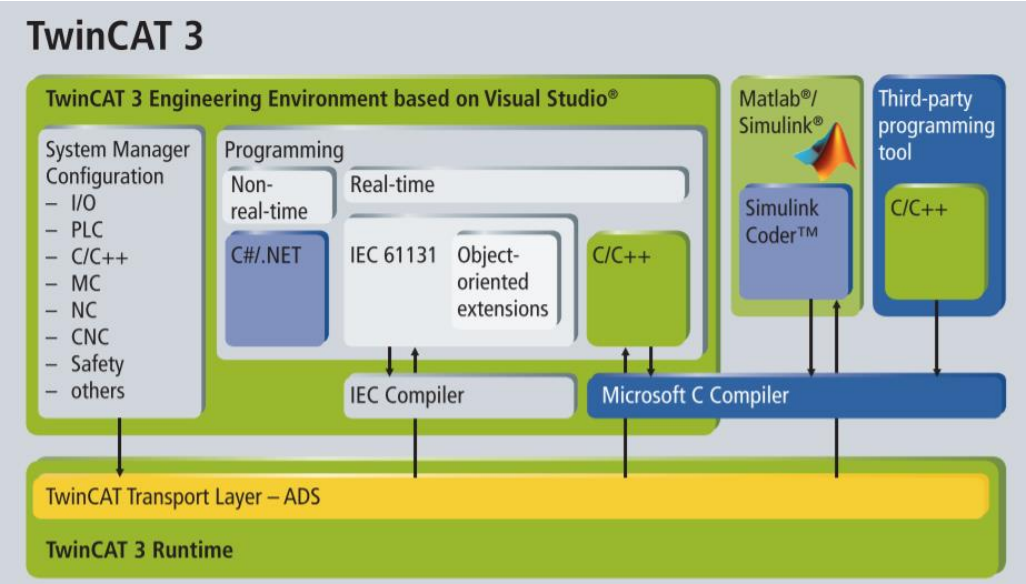


Ilustración 7 Esquema de funcionamiento de TwinCat 3 (Beckhoff, 2020)

El XAE contiene los lenguajes típicos de programación que se podría encontrar en un PLC, texto estructurado, diagrama de contactos, SFC… Pero también soporta otros lenguajes como C# y C/C++, también tiene compatibilidad con *Matlab/Simulink* y otros programas de terceros. Este entorno de programación está integrado en *Microsoft Visual Studio* (Microsoft, 2020), en la versión 2013, esto beneficia al usuario, ya que no tiene que acostumbrarse a un nuevo entorno, si ya ha tratado con *Visual Studio*, dado que la estructura que usa el XAE es muy similar.

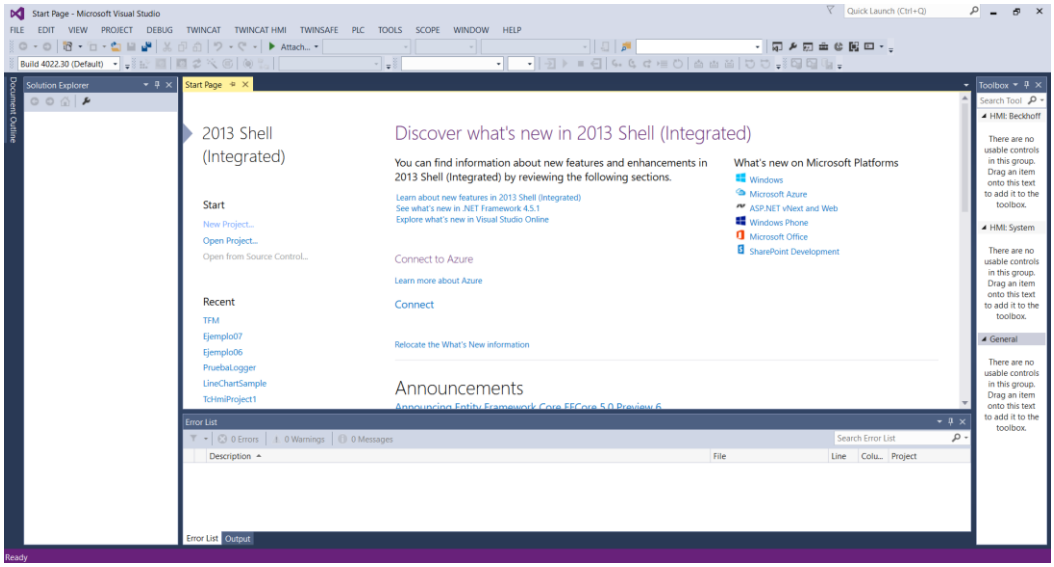


Ilustración 8 Página de inicio TwinCat 3 XAE

El XAR es un componente menos visual, se puede comprobar su estado y configuración en la barra desplegable de la barra de Windows.

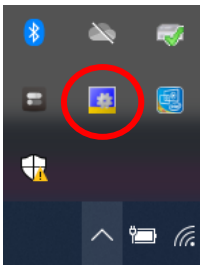


Ilustración 9 XAR de TwinCat 3

Es el encargado de gestionar las tareas que se han programado en el XAE y coordinarlas con las tareas de Windows. Tiene dos modos: Config y Run, en el modo Config el *Runtime* no está activo, y se puede cambiar la configuración de las tareas mediante el XAE, una vez se ha programado todo, hay que cargar el programa en el XAR, y cambiar su modo a Run, en este punto el dispositivo ya puede ejecutar su tarea, no permite cambios en la configuración hasta que se vuelva al modo Config. Como se ha mencionado antes cada tarea, que a su vez puede contener distintos programas, es totalmente configurable mediante el XAE (tiempos de ejecución, prioridad, núcleo en el que se ejecuta…), y permite distribuirlas como se muestra en la siguiente imagen:

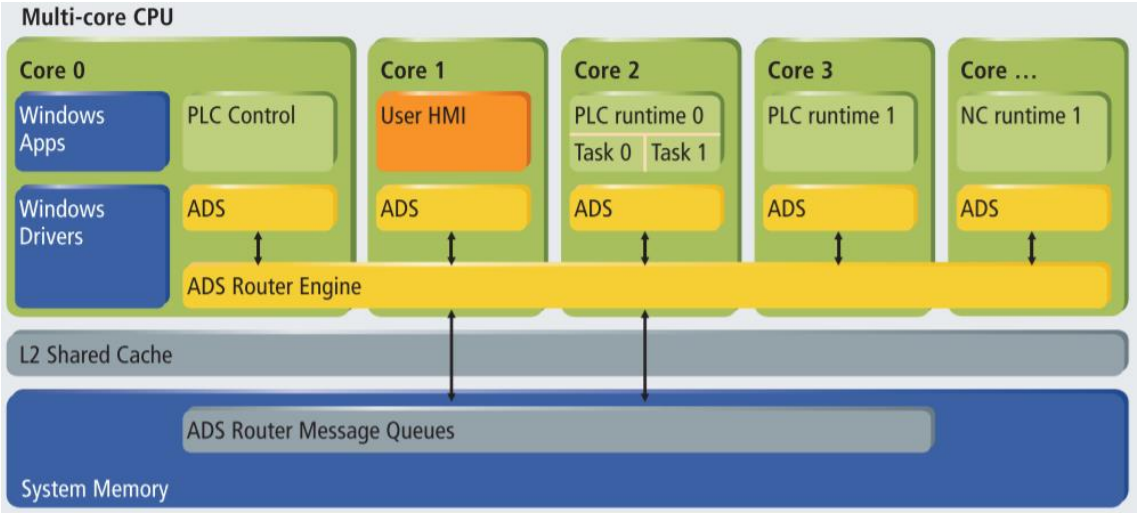


Ilustración 10 Ejemplo de distribución de tareas en el XAR (Beckhoff, 2020)

Los programas de *TwinCat* funcionan por módulos, es decir puedes utilizar o no los módulos disponibles, cuantos más se usen, el precio de implementación en

un PC real aumenta. Beckhoff propone un sistema de licencias en el que solo se paga por los módulos utilizados, las licencias de los módulos básicos que se usan en este proyecto son: TC3 ADS y TC3 PLC.

ADS es el protocolo de comunicación interno de *TwinCat*, se utiliza para comunicar los dos softwares principales, XAE y XAR. La licencia de PLC es necesaria ya que se ha programado todo el código del sistema de control dentro de este módulo.

Además de estas dos librerías, se ha utilizado: TC3 Controller Toolbox (Beckhoff, 2020), TC3 HMI Engineering, TC3 HMI Server (Beckhoff, 2020),TC3 Modbus TCP (Beckhoff, 2020) y TC3 EventLogger (Beckhoff, 2020). A diferencia de las anteriores estos paquetes no vienen por defecto en *TwinCat* y es necesario ir a la página de descargas de Beckhoff para descargarlos e instalarlos. Una vez instalados basta con importar la librería adecuada dentro del XAE en el apartado *References*, y activar la licencia obtenida para cada una de ellas. Para este trabajo no se ha adquirido ninguna licencia ya que este trabajo es de carácter académico, lo que se ha adquirido son licencias de prueba gratuitas, que tienen una duración de siete días y se pueden renovar sin limitaciones.

Todas las librerías disponen de un manual en el que se explica cómo instalarlas y su funcionamiento:

- TC3 Controller Toolbox: Utilizada para implementar PID en el módulo PLC de *TwinCat.* Manual: (Beckhoff, 2019).
- TC3 HMI Engineering: Proporciona el entorno de programación de HMIs. Manual: (Beckhoff, 2020)
- TC3 HMI Server: Servidor para la conexión entre el módulo PLC y HMI. Manual: (Beckhoff, 2018)
- TC3 Modbus TCP: Comunicación mediante protocolo *Modbus TCP*. Manual: (Beckhoff, 2020)
- TC3 EventLogger: Proporciona un detector de eventos. Manual: (Beckhoff, 2020)

3.5 ALGORITMOS GENÉTICOS

Los algoritmos genéticos permiten optimizar la solución de un problema, realiza sucesivas iteraciones generando en cada una de ellas una población de soluciones, en la siguiente iteración el algoritmo utiliza las soluciones anteriores como "padres" para generar un nuevo conjunto de soluciones que se aproximen más al objetivo definido, de ahí viene el nombre de "algoritmo genético" (MathWorks, 2020). En este caso se ha aplicado para obtener los valores óptimos de los parámetros de los PIDs programados. El pseudocódigo que siguen estos algoritmos es el siguiente (EHU, 2020):

1. INICIO Algoritmo genético
2. Generar una población inicial
3. Calcular la función de coste de cada individuo
4. WHILE NOT Fin DO
5. FOR población/2 DO
6. Seleccionar dos individuos de la población anterior con probabilidad asociada al valor del individuo en la función de coste
7. Cruzar los individuos obteniendo nuevos descendientes.
8. Evaluar los descendientes con la función de coste
9. Insertar los nuevos descendientes en la población
10. END FOR
11. IF la población ha convergido THEN
12. FIN = TRUE
13. END IF
14. END WHILE
15. END (Bramlette, 1991)

Este algoritmo está ya implementado en *Matlab/Simulink* mediante la función *ga()* de la librería *Global Optimization Toolbox* (MathWorks, 2020). Para su ejecución en *Matlab* es necesario tener el sistema de control y la simulación del proceso en un archivo *Simulink*, de esta manera el algoritmo genético es capaz de acceder al control y de hacer las pruebas necesarias para obtener los parámetros óptimos. La mejora que se obtiene de este algoritmo se mide

utilizando una función de coste, compuesta por la integral del error absoluto de las salidas respecto de las referencias (IAE), y de la variación de las acciones de control. Este índice de coste se le asigna la letra J y se normaliza para que el control original resulte en un valor de la función de coste de J=1, cualquier valor de J que obtenga el algoritmo y sea J<1, habrá mejorado el control en comparación con el original.

Ilustración 11 Procedimiento de cálculo del algoritmo genético

4 DESARROLLO DE LA SOLUCIÓN ADOPTADA

A partir de las ideas explicadas en el apartado 3 ANTECEDENTES, se ha diseñado una solución que permite el control de la planta de generación simulada, en los siguientes apartados se explica detalladamente todos los procesos e interacciones programados utilizando las herramientas disponibles en este trabajo:

- Entorno de programación *Python.*
- Entorno de programación *TwinCat 3.*
- *Matlab/Simulink.*

En la Ilustración 12 se muestra un esquema simplificado del funcionamiento del sistema de control que se ha diseñado, en el que encontramos por un lado el entorno de *TwinCat,* con los módulos PLC y HMI, que realizan las labores de control, por otro lado, el entorno de *Python* en el que se simula el proceso. Además, se ha creado un servidor, también en *Python,* para realizar la comunicación del proceso simulado y el sistema de control.

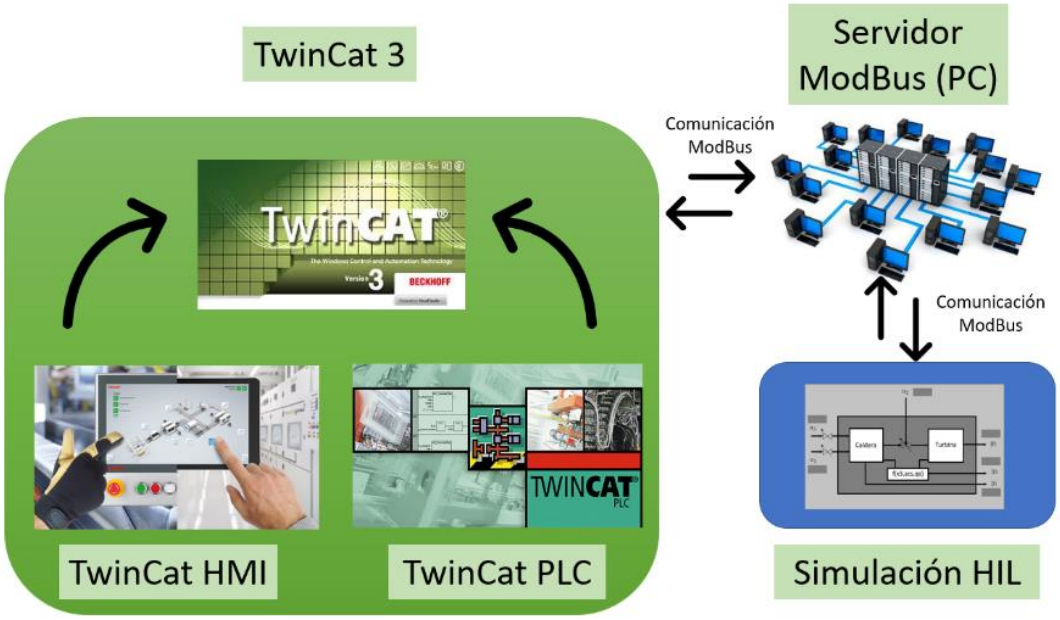


Ilustración 12 Esquema de la solución adoptada (VirtualCable, 2016) (Beckhoff, 2020).

4.1 ENTORNO DE PYTHON

En el entorno de *Python* se pueden distinguir dos programas con funcionalidades distintas, los nombres de estos programas son:

- Main_calderaNL.py (Modelo simulado HIL).
- Modbus_tcp_server.py (Servidor ModBus).

El primero de ellos corresponde a la simulación de las ecuaciones de la planta mediante HIL en tiempo real, y el segundo inicia un servidor ModBus en el ordenador que se ejecuta, al cual se pueden conectar clientes con la posibilidad de lectura y escritura de sus registros.

4.1.1 MODELO SIMULADO MEDIANTE HIL

Las ecuaciones que se han implementado han sido obtenidas del documento "Analysis and control of a nonlinear boiler-turbine unit" (Tan, Márquez, Tongwen, & Jizhen, 2005), consiste en un sistema caldera-turbina, en el cual existe la posibilidad de manipular tres válvulas y de medir tres salidas, el esquema de este proceso aparece en la Ilustración 13.

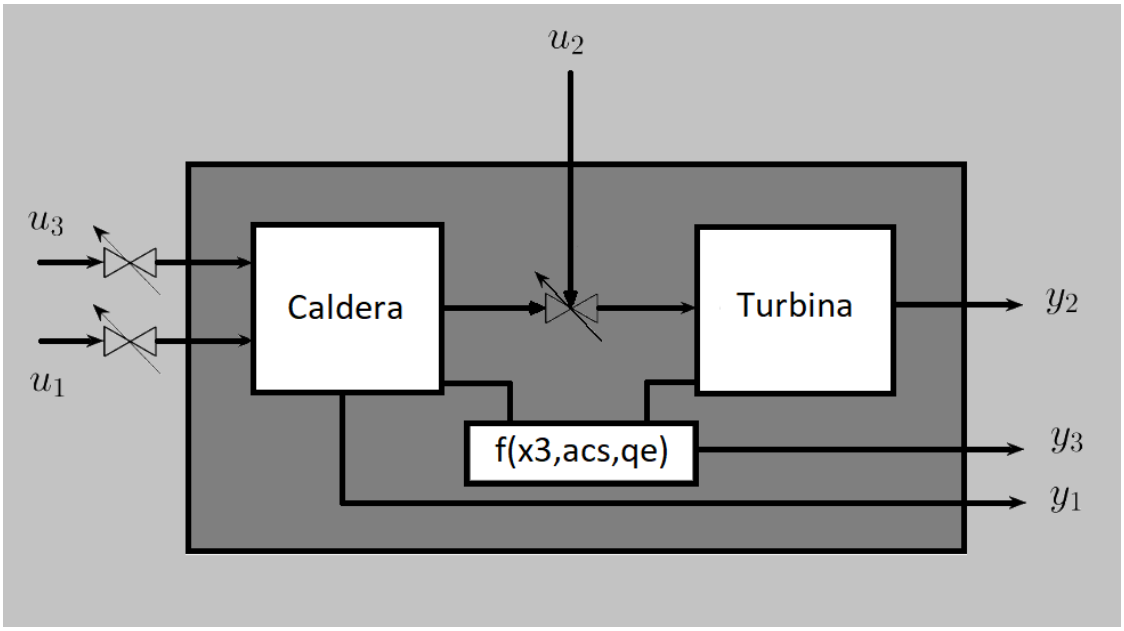


Ilustración 13 Esquema caldera-trubina

Las entradas nombradas como u1, u2 y u3 corresponden a las tres válvulas y representan: apertura de la válvula para el control de caudal de combustible [‰] (u1), apertura de la válvula para el control de caudal de vapor [‰] (u2) y apertura de la válvula para el control de caudal de agua en la caldera [‰] (u3). Por otro lado, las salidas denominadas como y1, y2 e y3 corresponden a: presión de la caldera [Kg/cm^2] (y1), potencia eléctrica de salida [MW] (y2) y nivel de agua de la caldera [m] (y3).

Todo este proceso está implementado en una tarea HIL que se ejecuta constantemente en un periodo de Ts=0.5s, ya que utilizando la librería *"Time"* de *Python*, se mide al inicio y al final el tiempo transcurrido para ejecutar esta tarea, posteriormente se espera el tiempo restante hasta que haya transcurrido 0.5s, el esquema de este programa se muestra en la Ilustración 14. Los cuadrados verdes representan pasos que solo se ejecutan al inicio del programa, mientras que los azules se repiten de manera cíclica, este bucle acaba cuando se detiene el programa.

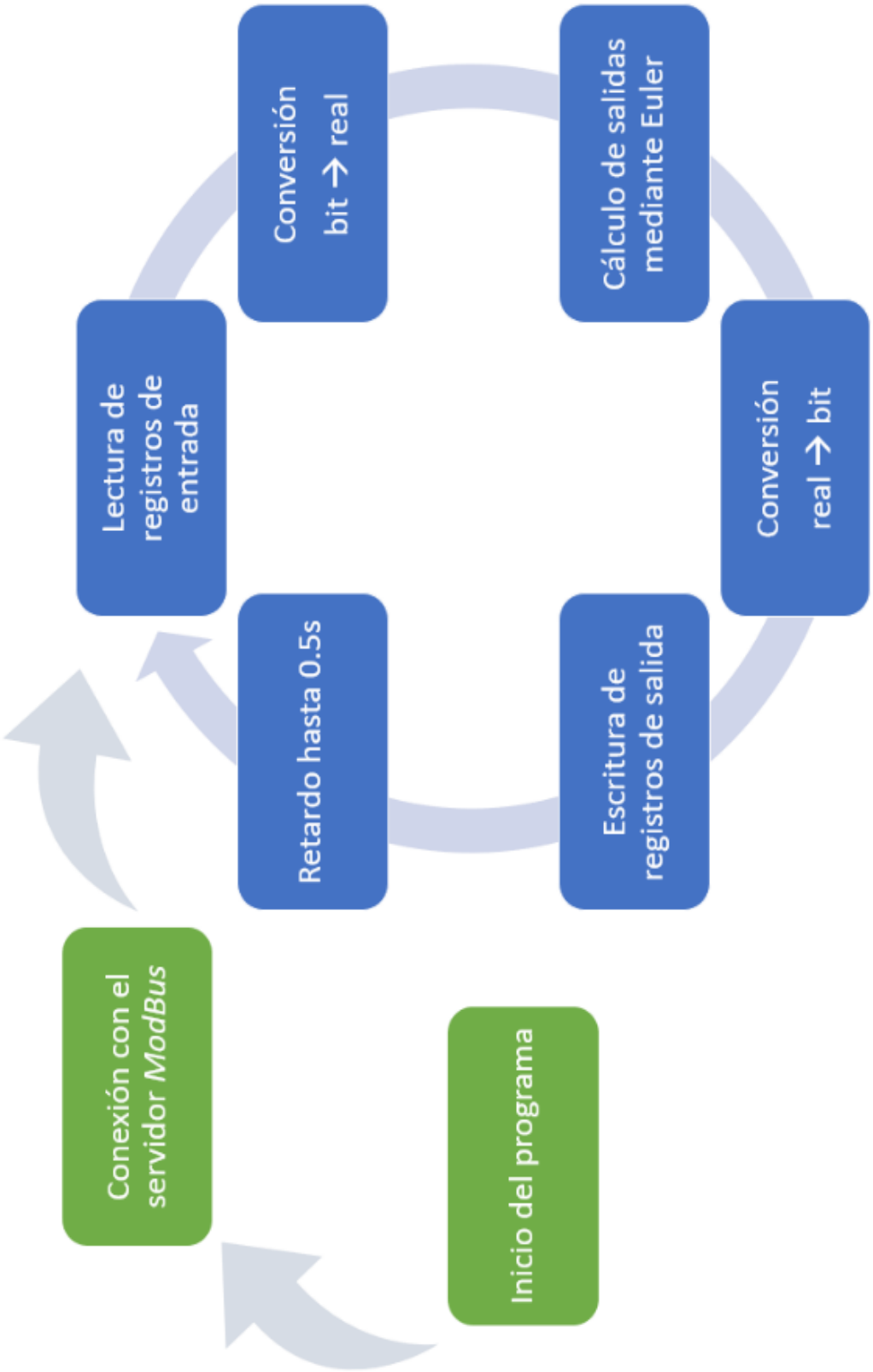


Ilustración 14 Flujograma de Main_calderaNL.py

Para poder realizar el cálculo de las variables de estado y salidas actualizadas, es necesario tener un valor de u1, u2 y u3, que se obtiene al leer los registros del servidor *ModBus*, en concreto para estas variables los registros son el 1, 2 y 3. Seguidamente se realiza la conversión de bits (nº de bits = 16) en el rango [0 65535] a un número real en el intervalo [0 1], que son los valores máximos y mínimo que pueden adoptar las acciones de control.

```
Ts=0.5
time_in = time.time()
u1_entera = c.read_holding_registers(1, 1)
u1=(float(u1_entera[0])/65535.0)
u2_entera = c.read_holding_registers(2, 1)
u2=(float(u2_entera[0])/65535.0)
u3_entera = c.read_holding_registers(3, 1)
u3=(float(u3_entera[0])/65535.0)
```

Código 1 Lectura de entradas y conversión bit/real

Mediante el método de integración numérica de Euler se han programado las siguientes ecuaciones:

$$\dot{x}_1 = -0.0018u_2x_1^{\frac{9}{8}} + 0.9u_1 - 0.15u_3 \quad (1)$$

$$\dot{x}_2 = (0.073u_2 - 0.016)x_1^{\frac{9}{8}} - 0.1x_2 \quad (2)$$

$$\dot{x}_3 = (141u_3 - (1.1u_2 - 0.19)x_1)/85 \quad (3)$$

$$y_1 = x1 \quad (4)$$

$$y_2 = x2 \quad (5)$$

$$y_3 = 0.05\left(0.13073x_3 + 100a_{cs} + \frac{q_e}{9} - 69.975\right) \quad (6)$$

$$a_{cs} = \frac{(1 - 0.001538x_3)(0.8x_1 - 25.6)}{x_3(1.0394 - 0.0012304x_1)} \quad (7)$$

$$q_e = (0.854u_2 - 0.147)x_1 + 45.59u_1 - 2.514u_3 - 2.096 \quad (8)$$

```
y1_out=y1+(-0.0018*u2*(y1**(9.0/8.0))+0.9*u1-0.15*u3)*Ts
y2_out=y2+((0.073*u2-0.016)*(y1**(9.0/8.0))-0.1*y2)*Ts
x3_out=x3+((141.0*u3-(1.1*u2-0.19)*y1)/85.0)*Ts
acs=((1.0-0.001538*x3)*(0.8*y1-25.6))/(x3*(1.0394-0.0012304*y1))
qe=(0.854*u2-0.147)*y1+45.59*u1-2.514*u3-2.096

y3_out=0.05*(0.13073*x3+100.0*acs+qe/9.0-67.975)
```

Código 2 Ecuaciones 1-8 implementadas mediante el método de integración numérica de Euler.

En el Código 2 se pueden observar las variables de estado x1, x2 y x3, en este modelo x1 y x2, representan exactamente lo mismo que y1 e y2, sin embargo x3 no se corresponde con la salida y3, en este caso la variable de estado x3 representa la densidad del fluido [Kg/cm^2], esto es debido a que la salida y3 es una función tal que y3=f(acs,qe,x3), las variables acs y qe representan la calidad del vapor y la tasa de evaporación respectivamente.

Con el valor en coma flotante de las acciones de control se realiza el cálculo de las salidas actualizadas del sistema, y se procede a escribir en los registros correspondientes del servidor, los valores de y1, y2 e y3. De nuevo es necesario una conversión para estos valores, pero en este caso se busca convertir el valor real a un valor comprendido entre 0 y 65535.

```
y1_entera=int((y1/(250.0))*65535.0) #Rango supuesto 0/250 Kg/cm^2
y2_entera=int((y2/(160.0))*65535.0) #Rango supuesto 0/160 MW

y3_entera=int(((y3+1.0)/2.0)*65535.0) #Rango supuesto -1/1 m
```

Código 3 Conversión valor de salida/bits

Finalmente, como se observa en el Código 4, se escribe en las salidas en los registros 4, 5 y 6 y se espera el tiempo correspondiente hasta cumplir los 0.5s.

```
regs = c.write_single_register(4, y1_entera)
regs = c.write_single_register(5, y2_entera)
regs = c.write_single_register(6, y3_entera)
tnow=time.time()
delaytime = Ts-(tnow-time_in)
timer = Timer(delaytime, HILTAsk)

timer.start()
```

Código 4 Escritura del valor de salida y retardo de espera

4.1.2SERVIDOR Y COMUNICACIÓN MODBUS

Se ha habilitado un servidor *ModBus TCP* con el objetivo de ser el punto de unión entre el proceso simulado y el sistema de control en *TwinCat*, en el que se ha

establecido la siguiente relación de variables y registros:

Variable	Registro de 16 bits
U1	1
U2	2
U3	3
Y1	4
Y2	5
Y3	6

Tabla 1 Relación de variables y registros del servidor ModBus

Aplicando esta relación el proceso simulado lee de los registros 1, 2 y 3, calcula las nuevas salidas, y las escribe en los registros 4, 5 y 6. Por el contrario el sistema de control lee los registros 4, 5 y 6, calcula las acciones de control, y escribe en los registros 1, 2 y 3. Este protocolo tiene una estructura cliente servidor, donde tanto la simulación como el sistema de control son clientes interactuando con el servidor. Utilizando las librerías de *ModBus TCP* se inicia el servidor en el entorno de *Python,* en este caso se ubica en la ip local (127.0.0.1) y en el puerto 504.

```
from pyModbusTCP.server import ModbusServer

server = ModbusServer(host='localhost', port=504)
server.start()
```

Código 5 Inicio del servidor ModBus TCP

Para la lectura/escritura de datos se utiliza las siguientes funciones de la librería de *ModBus TCP* de *Python.*

- ModBusClient().
- Host().

- Port().
- Read_holding_registers().
- Write_single_register().

Las tres primeras sirven para iniciar un cliente del servidor *ModBus* y para indicar la dirección de este, como se puede ver en Código 6. Las dos últimas son las que se utilizan para leer y escribir procesos en la simulación.

```
c = ModbusClient()
#c.debug(True)
c.host(SERVER_HOST)
c.port(SERVER_PORT)
```

Código 6 Inicio del cliente ModBus

4.2 ENTORNO TWINCAT

El proyecto final que se ha implementado en *TwinCat 3* está compuesto por dos módulos principales, PLC y HMI (Pantalla táctil). Cada uno tiene unas funciones específicas:

- PLC
    - Comunicación mediante protocolo *ModBus* con el servidor.
    - Cálculo de acciones de control mediante PIDs.
    - Escritura de datos en un fichero de texto.
    - Operaciones intermedias para lograr las funciones anteriores.
- HMI
    - Arranque/parada del sistema de control.
    - Control manual/automático.
    - Muestra información relativa a la planta y los controladores.
    - Modificar los parámetros de los controladores.

Estos dos módulos no son independientes, con la pantalla táctil se puede interactuar, de manera que acciones como pulsar un botón o cambiar un parámetro influyen en el código del módulo PLC, esta conexión se establece con el programa complementario de *TwinCat HMI Server*, el cual guarda y actualiza los símbolos (variables que se han declarado en el PLC y HMI).

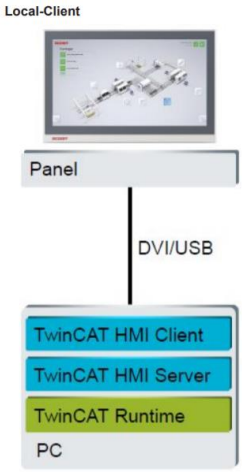


Ilustración 15 Esquema de comunicación HMI y PLC (Beckhoff, 2020)

En este caso se ha programado todo en el mismo PC, cliente HMI, servidor HMI y *Runtime* (PLC), pero podría establecerse en distintos dispositivos cada uno de ellos si se necesita un control más descentralizado.

4.2.1 PROGRAMACIÓN MÓDULO PLC

El primero de los módulos que se ha programado es el de PLC, formado por una tarea que contiene dos programas, implementados en texto estructurado, además permite configurar la duración de la tarea que se ejecuta a tiempo real. El árbol de contenidos del PLC se muestra en la Ilustración 16.

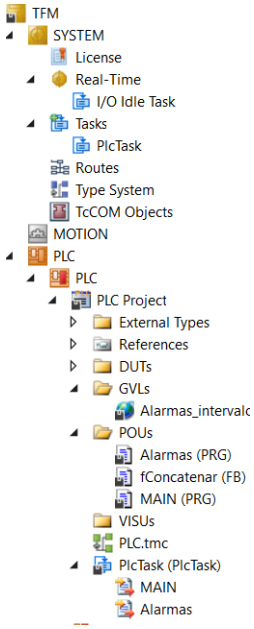


Ilustración 16 Árbol de contenido del módulo PLC

Dentro del apartado POUs es donde se ubican los dos programas que ejecuta la tarea *PlcTask*, esta tarea tiene un periodo de 1s, este tiempo se especifica en el apartado de *SYSTEM/Tasks/PlcTask.*

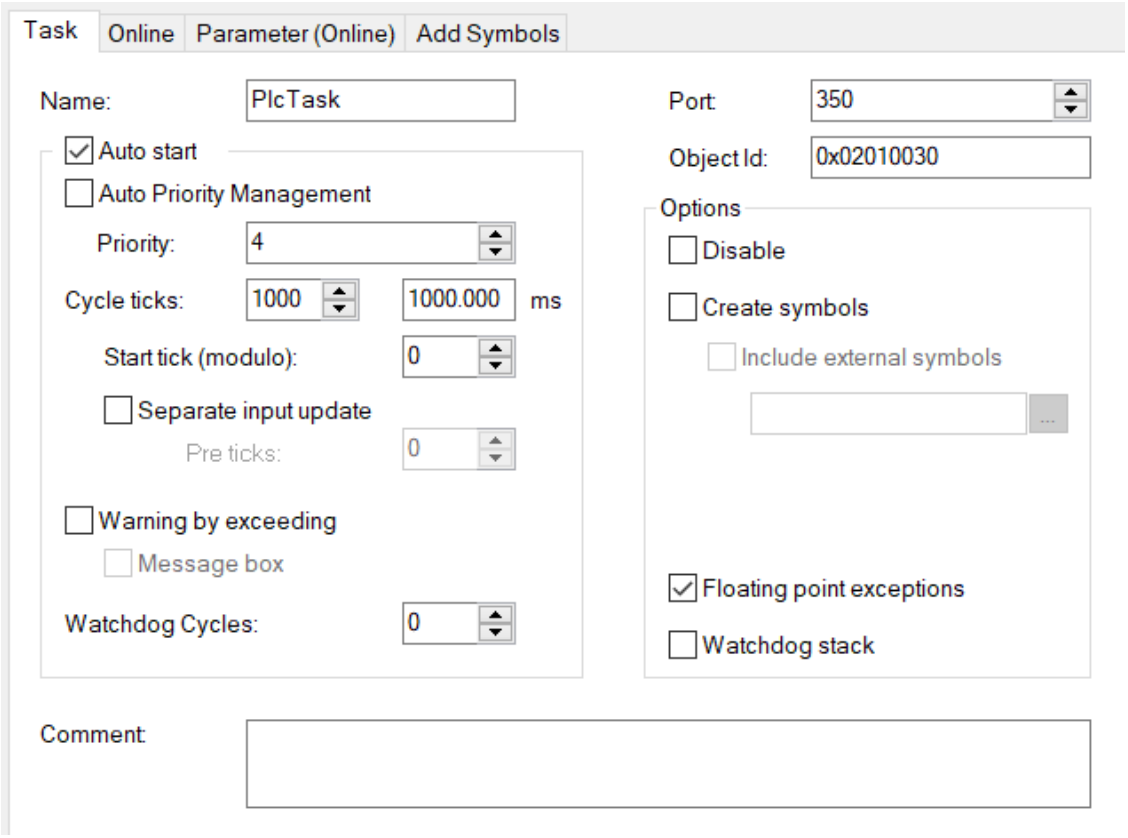


Ilustración 17 Asignación de tiempo de ejecución TwinCat

El programa que contiene la mayor parte del sistema de control es MAIN(PRG), en la Ilustración 18 se muestra un flujograma con los pasos que va siguiendo el programa. Los pasos marcados con el fondo verde indican que son procedimientos iniciales que solo se van a ejecutar una vez (Declaración de variables, apertura de fichero de texto…), las acciones con el fondo azul son el núcleo del programa que se repite constantemente en un periodo de 1s (Comunicación mediante Modbus, cálculo de acciones de control…), cuando se decide parar el sistema de control se ejecutan los pasos con el fondo naranja, estas últimas también se ejecutan una sola vez (Cierre del archivo de texto).

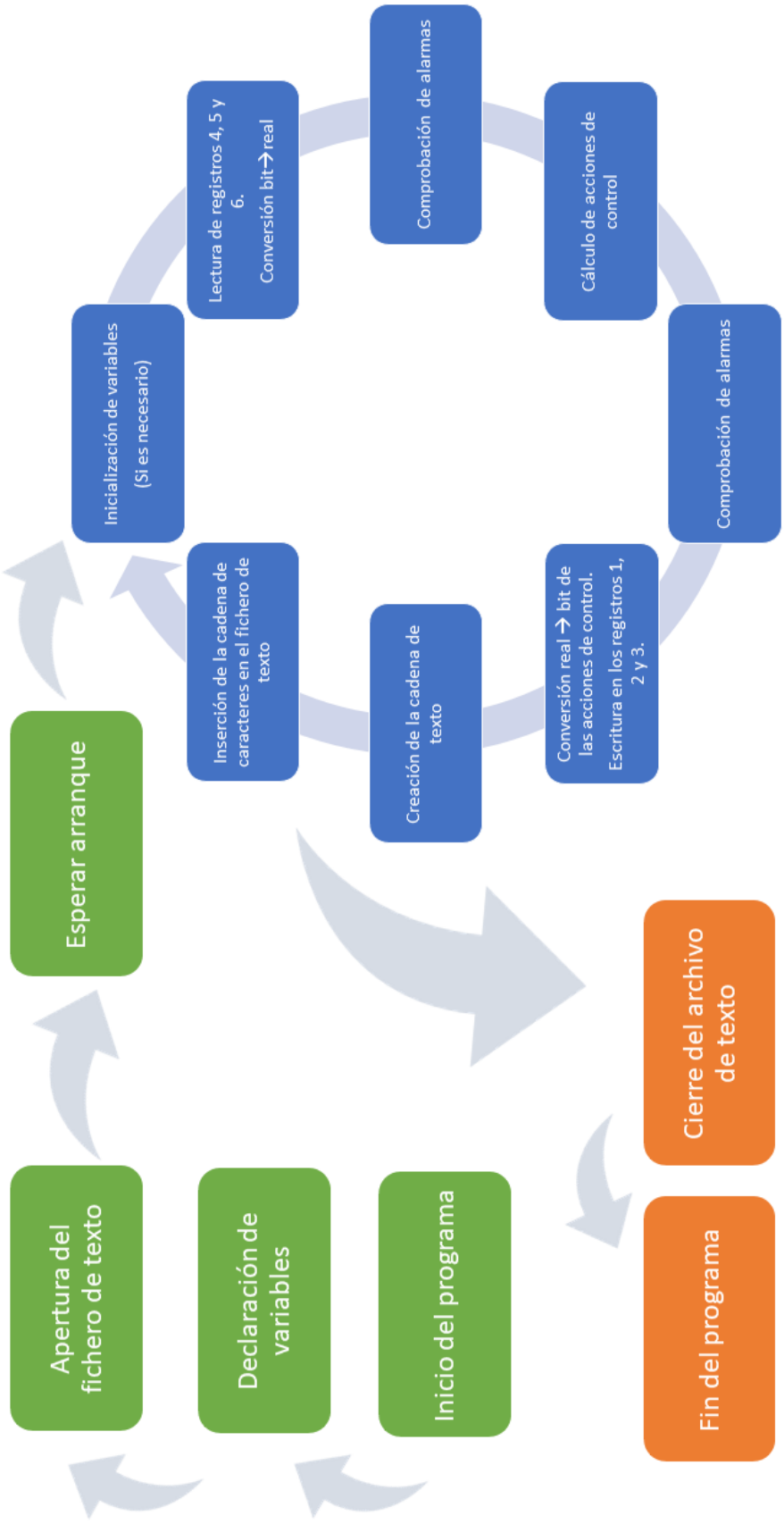

Esperar arranque
Apertura del fichero de texto
Declaración de variables
Inicio del programa
Inicialización de variables (Si es necesario)
Lectura de registros 4, 5 y 6. Conversión bit→real
Comprobación de alarmas
Cálculo de acciones de control
Comprobación de alarmas
Conversión real → bit de las acciones de control. Escritura en los registros 1, 2 y 3.
Creación de la cadena de texto
Inserción de la cadena de caracteres en el fichero de texto
Cierre del archivo de texto
Fin del programa

Ilustración 18 Flujograma MAIN(PRG)

1. Declaración de variables:

Todas las variables relativas al control, escritura en el fichero de texto y comunicación mediante protocolo ModBus son declaradas.

```
PROGRAM MAIN
VAR
    bReadRegsBusy      : BOOL;
    bReadRegsError     : BOOL;
    nReadRegsErrorId   : UDINT;
    nReadRegsCount     : UDINT;
    nQuantityr          : WORD:=3;
    nMBAddrr            : WORD:=4;
    arrDatar            : ARRAY [1..3] OF WORD;
    FB_MBReadRegs: FB_MBReadRegs;

    bWriteRegsBusy          : BOOL;
    bWriteRegsError         : BOOL;
    nWriteRegsErrorId       : UDINT;
    nMBAddrw               : WORD := 1;
    arrDataw               : ARRAY [1..3] OF WORD;
    FB_MBWriteRegs: FB_MBWriteRegs;

    nCounter : INT:=1;
    y1 :REAL;
    y2: REAL;
    y3: REAL;
    aPoints1 :ARRAY[1..2,1..1000] OF ST_Point;
    aPoints2 :ARRAY[1..2,1..1000] OF ST_Point;
    aPoints3 :ARRAY[1..2,1..1000] OF ST_Point;
    aPoints4 :ARRAY[1..2,1..1000] OF ST_Point;
    aPoints5 :ARRAY[1..2,1..1000] OF ST_Point;
    aPoints6 :ARRAY[1..2,1..1000] OF ST_Point;
    bInit :BOOL:=TRUE;
```

Ilustración 19 Declaración de variables

2. Apertura del fichero de texto:

Se ejecuta la función fbFileOpen de la librería *Utilities* de *TwinCat,* que abre el fichero de datos históricos en modo APPEND, de manera que no borra los datos que ya tiene el fichero y continúa escribiendo a continuación del último.

```
bFileOpen := TRUE;
    fbFileOpen(
    sNetId:= ,
    sPathName:= 'C:\Programas Twincat\TFM\Datos_historico.txt',
    nMode:= FOPEN_MODEAPPEND OR FOPEN_MODEBINARY,
    ePath:= PATH_GENERIC,
    bExecute:= bFileOpen,
    tTimeout:= T#2S,
    bBusy=> bBusyFileOpen,
    bError=> bErrorFileOpen,
    nErrId=> nIdFileOpen,
    hFile=> hFichero);
```

Ilustración 20 Apertura del fichero de texto en modo APPEND

3. Esperar el arranque:

El programa se queda esperando a que se inicie el sistema de control desde la pantalla táctil pulsando el botón de arranque.

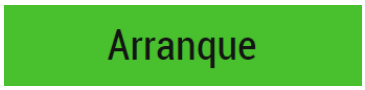


Ilustración 21 Botón de arranque/parada del sistema de control

4. Inicialización de variables:

Durante el primer ciclo del bucle se inicializan los vectores de datos que representan las gráficas de la pantalla táctil a 0, ya que no hay información para rellenarlas. También se establecen los valores de la estructura de datos perteneciente a cada PID, es decir su tiempo de ciclo, ganancia, tiempo de integración, límites máximos y mínimos, así como otros parámetros de configuración del PID.

```
IF bCounter THEN
    FOR nCounter:=1 TO 1600 DO
        //Salidas
        aPoints1[1,nCounter].x:= nCounter;
        aPoints1[1,nCounter].y:= 0.0;
        aPoints1[1,1].y:= 108.0;

        aPoints2[1,nCounter].x:= nCounter;
        aPoints2[1,nCounter].y:= 0.0;
        aPoints2[1,1].y:= 66.65;

        aPoints3[1,nCounter].x:= nCounter;
        aPoints3[1,nCounter].y:= 0.0;
        aPoints3[1,1].y:= 0.0;
        //Entradas
        aPoints4[1,nCounter].x:= nCounter;
        aPoints4[1,nCounter].y:= 0.0;
        aPoints4[1,1].y:= 0.34;

        aPoints5[1,nCounter].x:= nCounter;
        aPoints5[1,nCounter].y:= 0.0;
        aPoints5[1,1].y:= 0.69;

        aPoints6[1,nCounter].x:= nCounter;
        aPoints6[1,nCounter].y:= 0.0;
        aPoints6[1,1].y:= 0.433;
    END_FOR
```

Ilustración 22 Iniciación de los vectores de datos de los gráficos HMI

El vector de datos que corresponde a los gráficos HMI, contiene en cada una de sus ochocientas celdas, una variable declarada como una estructura de datos con dos valores reales, que definen las coordenadas x e y del punto que representa esa celda. La pantalla táctil lee el valor de este vector y representa en un gráfico los 800 puntos, cada uno en su coordenada x e y correspondiente, en todos los casos el eje x representa el tiempo en segundos, mientras que el eje y representa el valor de la variable en sus unidades correspondientes. En la Ilustración 23 se muestra como ejemplo como quedaría representado el vector de datos que hace referencia a la salida 3, donde los valores de x son [1 800] y los de y son constantes e igual a cero.

x	1	2	3	…	800
y	0	0	0	…	0

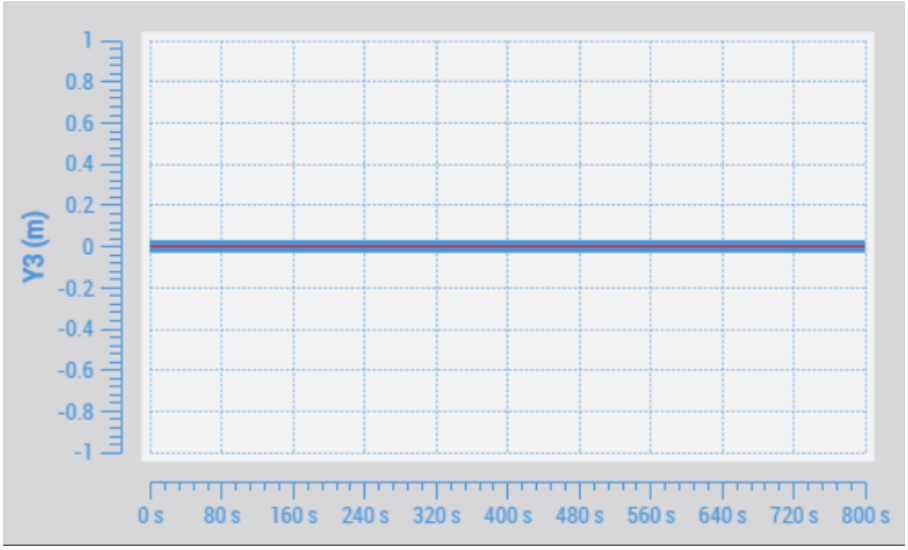


Ilustración 23 Representación del vector de datos del gráfico HMI

En la Ilustración 24 también se observa una variable de tipo estructura, *stParams1,* hace referencia a todos los parámetros que utiliza el PID para el cálculo de la acción de control y su funcionamiento. Todos los PIDs se han programado con un tiempo de tarea de 1s, exactamente el mismo tiempo que tarda en ejecutarse la tarea de tiempo real de *TwinCat,* la diferencia entre los

controladores son los valores de ganancia, tiempo de integración y límites de saturación de cada uno.

```
IF bInit THEN
    bInit:=FALSE;

    stParams1.tCtrlCycleTime:= T#1S;
    stParams1.tTaskCycleTime:= T#1S;
    stParams1.fKp:= kp1;
    tn1 := LREAL_TO_TIME(tn1r*1000);
    stParams1.tTn:= tn1;
    str1 :=TIME_TO_STRING(tn1);
    stParams1.fOutMaxLimit:= 0.66;
    stParams1.fOutMinLimit:= -0.34;
    stParams1.bARWOnIPartOnly:=FALSE;
```

Ilustración 24 Iniciación de los parámetros del PID1

5. Lectura de los registros 4, 5 y 6 del servidor ModBus:

Para la lectura de registros mediante protocolo *ModBus* en *TwinCat,* se ha utilizado el siguiente bloque de función FB_MBReadRegs. En este bloque hay que especificar la dirección del servidor (Ip y puerto), además de cuantas palabras se quieren leer y a partir de que registro, en este caso se quieren leer los registros 4, 5 y 6 (nQuantity = tres registros) y empezando en la posición 4 (nMBAddr = 4), todas estas características se muestran en la Ilustración 25.

```
FB_MBReadRegs(
    sIPAddr:= '127.0.0.1',
    nTCPPort:= 504,
    nUnitID:= 255,
    nQuantity:= nQuantityr,
    nMBAddr:= nMBAddrr,
    cbLength:= SIZEOF(arrDatar),
    pDestAddr:= ADR(arrDatar),
    bExecute:= TRUE,
    tTimeout:= T#1S,
    bBusy=> bReadRegsBusy,
    bError=> bReadRegsError,
    nErrId=> nReadRegsErrorId,
    cbRead=> nReadRegsCount);
```

Ilustración 25 Lectura del servidor ModBus desde Twincat 3

Se leen los valores que tienen las salidas mediante el código que aparece en la Ilustración 25 y se almacenan en el vector arrDatar. En estos registros el valor

que aparece es un número entre [0 65535] por lo que hay que convertirlo a real utilizando los límites establecidos. Y1 → [0 250], Y2 →[0 160], Y3 → [-1 1]. Por último, el valor de cada una de las salidas es asignado al componente y del vector que representa el gráfico HMI.

```
y1:=(arrDatar[1]/65535.0)*250.0;
aPoints1[1,nCounter].y:=y1;
y2:=(arrDatar[2]/65535.0)*160.0;
aPoints2[1,nCounter].y:=y2;
y3:=(arrDatar[3]/65535.0)*2.0-1.0;
aPoints3[1,nCounter].y:=y3;
```

Ilustración 26 Asignación de la salida al vector que representa el gráfico HMI.

6. Comprobación de alarmas:

Una vez convertidas las salidas a números reales, se comprueba mediante operaciones lógicas si las salidas están desviadas de la referencia un ±10%, en el caso de que así sea, se actualiza el valor de la variable binaria correspondiente y se activará una alarma en la pantalla táctil, haciendo saber al usuario que esté consultando la pantalla que alguna de las salidas está fuera del rango establecido.

7. Cálculo de las acciones de control

El siguiente paso es calcular la acción de control utilizando la referencia establecida y el valor de salida que se ha leído, a continuación, se explica cómo funciona el bloque PID1, pero los demás funcionan exactamente igual.

```
FB_CTRL_PI1(
    fSetpointValue:= ref1,
    fActualValue:= y1,
    fManSyncValue:=u1m-0.34 ,
    bSync:=FALSE,
    eMode:=eMode1 ,
    bHold:=FALSE,
    fOut=> u1,
    bARWactive=> ,
    eState=> ,
    eErrorId=> ,
    bError=> bErrorPID1,
    stParams:=stParams1 );
```

Ilustración 27 Código referente al PID1

La Ilustración 27 muestra las distintas variables de salida y entrada del bloque PI

que proporciona la librería *Controller Toolbox* de *Twincat*, se establece la referencia y el valor actual de la variable con los dos primeros parámetros (fSetPointValue y fAtualValue). El tercer parámetro fManSyncValue, tiene que ver con el parámetro eMode, el PID se inicializa en modo automático (eMode = eCTRL_MODE_ACTIVE), y funcionará de manera normal, calculando constantemente la salida y asignándola a fOut. Pero existe la posibilidad, desde la pantalla táctil, de cambiar el PID a modo manual (eMode = eCTRL_MODE_MANUAL), en este estado el PID asignará a fOut el mismo valor que aparezca en fManSyncValue, sin importar la discrepancia entre la salida y la referencia. Esta variable adquiere el valor que el usuario desee, siempre y cuando esté dentro de los límites admisibles.

Las variables bSync y bHold no han sido utilizadas en este proyecto, el resto de las variables de salida aportan información sobre el PID y los posibles errores que ocurran. En la última posición se asigna la estructura de parámetros inicializada en el paso 4.

8. Comprobación de alarmas:

En este caso se comprueba si los valores de las acciones de control están por encima del 90% o por debajo del 10% del grado de apertura, ya que en este estado se tiene poco rango de maniobra, y no se desea que los actuadores saturen. Si se diera alguno de estos casos, se actualiza la variable binaria correspondiente a la alarma y aparece un mensaje en la pantalla táctil.

9. Escritura en los registros 1, 2 y 3 del servidor *ModBus*

Después de que el PID calcule la acción de control, su valor es asignado a la coordenada y del vector que representa esta acción de control, para su posterior representación en la pantalla táctil. También se realiza su conversión al intervalo [0 65535] para ser escrito en el registro del servidor *ModBus*.

Con todas las acciones de control calculadas, se utiliza el bloque FB_MBWriteRegs. En la Ilustración 28 se muestra el código que utiliza esta función, cuando detecte un flanco de subida en la variable bExecute, escribirá los

datos en los registros 1, 2 y 3.

```
FB_MBWriteRegs(
    sIPAddr:= '127.0.0.1',
    nTCPPort:= 504,
    nUnitID:= 255,
    nQuantity:= 3,
    nMBAddr:= nMBAddrw,
    cbLength:= SIZEOF(arrDataw),
    pSrcAddr:= ADR(arrDataw),
    bExecute:= TRUE,
    tTimeout:= T#1S,
    bBusy=> bWriteRegsBusy,
    bError=> bWriteRegsError,
    nErrId=> nWriteRegsErrorId);
```

Ilustración 28 Escritura del servidor ModBus desde Twincat 3

10. Creación de la cadena de caracteres para escritura en el fichero de texto:

En este paso se concatenan los datos que se quieren escribir en el fichero de texto y se les da un formato, de manera que cuando el usuario quiera consultar estos datos sea sencillo de comprender. Los datos se escriben según el orden que especifica la siguiente tabla. Para realizar este paso se ha creado la función fConcatenar que realiza todos los pasos para conseguir esta estructura de datos.

U1: valor	U2: valor	U3: valor	Y1: valor	Y2: valor	Y3: valor	Fecha y hora

```
u1:0.337 u2:0.690 u3:0.443 y1:108.003 y2:66.649 y3:-0.000  2020/6/29 12:30
u1:0.332 u2:0.690 u3:0.450 y1:108.007 y2:66.649 y3:-0.000  2020/6/29 12:30
u1:0.343 u2:0.690 u3:0.483 y1:107.996 y2:66.654 y3:-0.001  2020/6/29 12:30
u1:0.363 u2:0.689 u3:0.537 y1:107.977 y2:66.656 y3:-0.002  2020/6/29 12:30
```

Ilustración 29 Ejemplo de escritura de datos en el fichero de texto

11. Inserción de la cadena de caracteres en fichero de texto:

Para este apartado se usa la función fbFilePuts, la cual escribe en el fichero de datos que se ha abierto con anterioridad en modo APPEND, escribiendo a continuación del último dato, la nueva cadena de caracteres que se ha creado. Cuando detecte un flanco de subida en bExecute, escribirá en el fichero hFile (hFile = Handler proporcionado por la función FileOpen) la cadena de caracteres strFile, que tendrá como estructura la explicada en el paso anterior.

```
fbFilePuts(
    sNetId:= ,
    hFile:= hFichero,
    sLine:= strFile,
    bExecute:= TRUE,
    tTimeout:= T#2S,
    bBusy=> ,
    bError=> ,
    nErrId=> );
```

Ilustración 30 Función FilePuts

12. Volver al paso 5:

El programa se repetirá desde la lectura de los registros 1, 2 y 3 hasta el paso anterior, donde se escribe en el fichero de texto. Solo cuando un usuario pulse el botón de parada, el proceso continuará a su paso final. Existe la posibilidad de volver al paso 4 Inicialización de variables si se requiere un cambio en los parámetros de ganancia o tiempo de integración de los PIDs.

13. Cerrar el archivo de texto:

Cuando el programa vaya a finalizar por orden de un usuario, primero ejecuta el comando fbFileClose, el cual cierra el archivo de texto que se ha abierto en modo APPEND al principio del programa, esto es necesario para un correcto almacenamiento de los datos en el fichero de texto, si no se ejecuta este comando y el programa de control se cierra, existe la posibilidad de pérdida de datos.

Estos trece pasos son todas las funcionalidades que se han incorporado al programa MAIN, pero como se ha mencionado antes existe otro programa, que será el encargado de las alarmas. Antes de pasar al programa ALARMAS, en la Tabla 2 se muestran todas las alarmas diseñadas, así como su texto, id y severidad.

Evento	Id	Mensaje	Severidad
U1_90	1	Apertura de la válvula 1 >= 90%	Warning

U2_90	2	Apertura de la válvula 2 >= 90%	Warning
U2_90	3	Apertura de la válvula 3 >= 90%	Warning
U1_10	4	Apertura de la válvula 1 <= 10%	Warning
U2_10	5	Apertura de la válvula 2 <= 10%	Warning
U3_10	6	Apertura de la válvula 3 <= 10%	Warning
SP1	7	Cambio de referencia 1 efectuado	Info
SP2	8	Cambio de referencia 2 efectuado	Info
SP3	9	Cambio de referencia 3 efectuado	Info
Kp1	10	Cambio de Kp1 efectuado	Info
Kp2	11	Cambio de Kp2 efectuado	Info
Kp3	12	Cambio de Kp3 efectuado	Info
Ti1	13	Cambio de Ti1 efectuado	Info
Ti2	14	Cambio de Ti2 efectuado	Info
Ti3	15	Cambio de Ti3 efectuado	Info
Admin	16	Modo administrador activado.	Info
Adminlogout	17	Modo administrador desactivado.	Info
Y1_10plus	18	La salida 1 está más de un 10% desviada por encima del SP1	Warning
Y1_10minus	19	La salida 1 está más de un 10% desviada por debajo del SP1	Warning

Y2_10plus	18	La salida 2 está más de un 10% desviada por encima del SP2	Warning
Y2_10minus	19	La salida 2 está más de un 10% desviada por debajo del SP2	Warning
Y3_10plus	18	La salida 3 está más de un 10% desviada por encima del SP3	Warning
Y3_10minus	19	La salida 3 está más de un 10% desviada por debajo del SP3	Warning

Tabla 2 Alarmas programadas

Todas estas alarmas hay que declararlas como una EventClass, y luego ir definiendo cada una de ellas. Debido a que algunas de estas alarmas se activan utilizando datos del programa MAIN, se ha creado una lista de variables global (GVL), con este recurso se puede interactuar con la misma variable desde dos programas distintos.

```
VAR_GLOBAL
    bu1_90 : BOOL := FALSE;
    bu2_90 : BOOL := FALSE;
    bu3_90 : BOOL := FALSE;
    bu1_10 : BOOL := FALSE;
    bu2_10 : BOOL := FALSE;
    bu3_10 : BOOL := FALSE;
    by1_10plus :BOOL :=FALSE;
    by1_10minus :BOOL :=FALSE;
    by2_10plus :BOOL :=FALSE;
    by2_10minus :BOOL :=FALSE;
    by3_10plus :BOOL :=FALSE;
    by3_10minus :BOOL :=FALSE;
END VAR
```

Ilustración 31 Declaración de la lista de variables global

Finalmente, en la Ilustración 32 se muestra el flujograma del programa ALARMAS(PRG), como en apartados anteriores los cuadros verdes representan acciones que tan solo se ejecutan una vez, mientras que las que tienen fondo azul son las tareas que se repiten en cada ejecución.

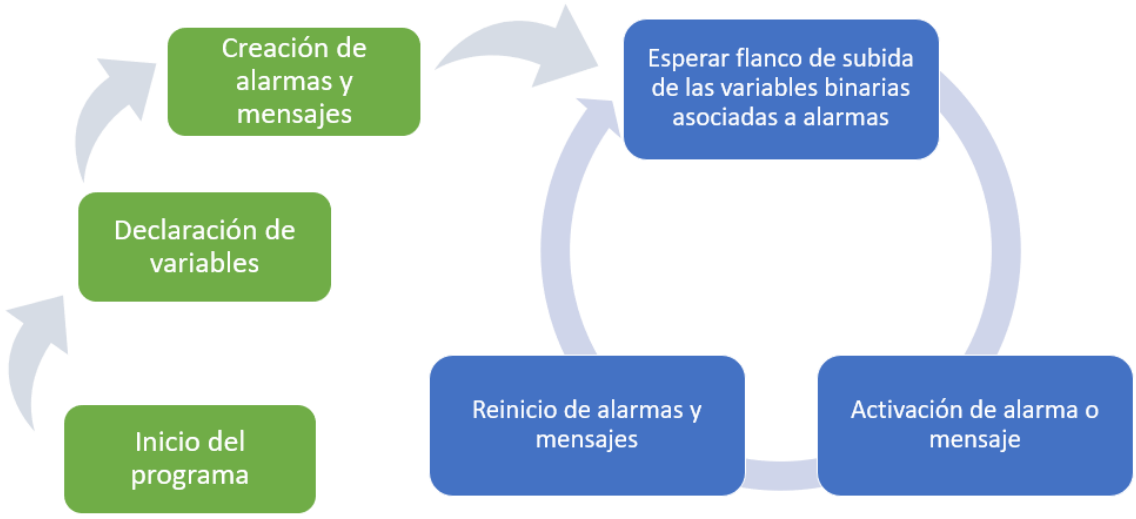


Ilustración 32 Flujograma ALARMAS(PRG)

1. Declaración de alarmas:

El primer paso es declarar las variables que funcionarán como alarmas o mensajes, para ello se usan los tipos de variable FB_TCAlarm y FB_TCMessage.

```
fby1_10plus : FB_TcAlarm;
fby1_10minus : FB_TcAlarm;
fby2_10plus : FB_TcAlarm;
fby2_10minus : FB_TcAlarm;
fby3_10plus : FB_TcAlarm;
fby3_10minus : FB_TcAlarm;
fbu1_90 : FB_TcMessage;
fbu2_90 : FB_TcMessage;
fbu3_90 : FB_TcMessage;
fbu1_10 : FB_TcMessage;
fbu2_10 : FB_TcMessage;
fbu3_10 : FB_TcMessage;
```

Ilustración 33 Declaración de alarmas y mensajes

2. Iniciación de alarmas y mensajes:

Utilizando el comando CreatEx, se inicializan las variables de alarmas y mensajes, este paso solo se ejecuta la primera vez que el programa funciona.

```
fbu1_90.CreateEx(TC_EVENTS.EventClass.u1_90, 0);
fbu2_90.CreateEx(TC_EVENTS.EventClass.u2_90, 0);
fbu3_90.CreateEx(TC_EVENTS.EventClass.u3_90, 0);
```

Ilustración 34 Inicialización de alarmas y mensajes

3. Espera a condición de alarma:

La mayoría del tiempo este programa de alarmas simplemente está observando si alguna de las variables binarias correspondientes a las alarmas y mensajes se activa, si este es el caso ejecuta el comando Send() para mensajes y Raise() para alarmas, además las variables binarias son reiniciadas para poder detectar de nuevo si alarma se sigue cumpliendo.

```
IF bLogout THEN
    bLogout := FALSE;
    fbLogout.Send(0);
END_IF

IF by1_10plus THEN
    by1_10plus :=FALSE;
    fby1_10plus.Raise(0);
END_IF
```

Ilustración 35 Activación del mensaje y alarma

Volviendo al paso 10, se menciona la función fConcatenar, esta función se ha creado para dar el formato a los datos que van a ser escritos en el fichero de texto, utiliza la función *CONCAT* de *TwinCat,* para unir dos variables de tipo *string*, en una sola. Para obtener el valor de la fecha y hora se ha utilizado la función NT_GetTime de la librería *Utilities,* esta función devuelve una estructura de datos de tipo *Word*, utilizando la codificación dada en (Beckhoff, 2020), se puede transformar cada uno de los campos de la estructura en números enteros que representen la fecha con el formato AAAA/MM/DD HH:MM.

```
str := CONCAT(STR1:='u1:' , STR2:= StringData[1] );
str := CONCAT(STR1:=str , STR2:= 'u2:' );
str := CONCAT(STR1:=str , STR2:= StringData[2] );
str := CONCAT(STR1:=str , STR2:= 'u3:' );
str := CONCAT(STR1:=str , STR2:= StringData[3] );
str := CONCAT(STR1:=str , STR2:= 'y1:' );
str := CONCAT(STR1:=str , STR2:= StringData[4] );
str := CONCAT(STR1:=str , STR2:= 'y2:' );
str := CONCAT(STR1:=str , STR2:= StringData[5] );
str := CONCAT(STR1:=str , STR2:= 'y3:' );
str := CONCAT(STR1:=str , STR2:= StringData[6] );
str := CONCAT(STR1:=str , STR2:= ' ' );
str := CONCAT(STR1:=str , STR2:= year );
str := CONCAT(STR1:=str , STR2:= month );
str := CONCAT(STR1:=str , STR2:= day );
str := CONCAT(STR1:=str , STR2:= ' ' );
str := CONCAT(STR1:=str , STR2:= hour );
str := CONCAT(STR1:=str , STR2:= minute );
Stringconcat:=CONCAT(STR1:= str, STR2:= '$0D');
```

Ilustración 36 Función fConcatenar

A cada valor se le añade un indicador (u1, u2, u3, etc…) para que al leer el fichero de datos sea más intuitivo, para acabar la función se añade la fecha y hora y el símbolo "$0D", que se interpreta como un salto de línea en el fichero de texto.

4.2.2DISEÑO DEL HMI

Como ya se ha mencionado anteriormente, la pantalla táctil es el elemento que permite al usuario interactuar con el sistema de control y la planta de generación, el objetivo es que sea sencilla de utilizar y de entender, además se busca que las funcionalidades más críticas solo sean accesibles por usuarios capacitados, es decir la pantalla táctil no solo ha de ser funcional y sencilla, sino que la seguridad es un elemento muy importante. Sus funcionalidades se pueden dividir en dos: Información y control. El resultado final de la pantalla táctil se muestra en la Ilustración 37, la cual se ha subdividido en tres regiones, dos de información y una de control

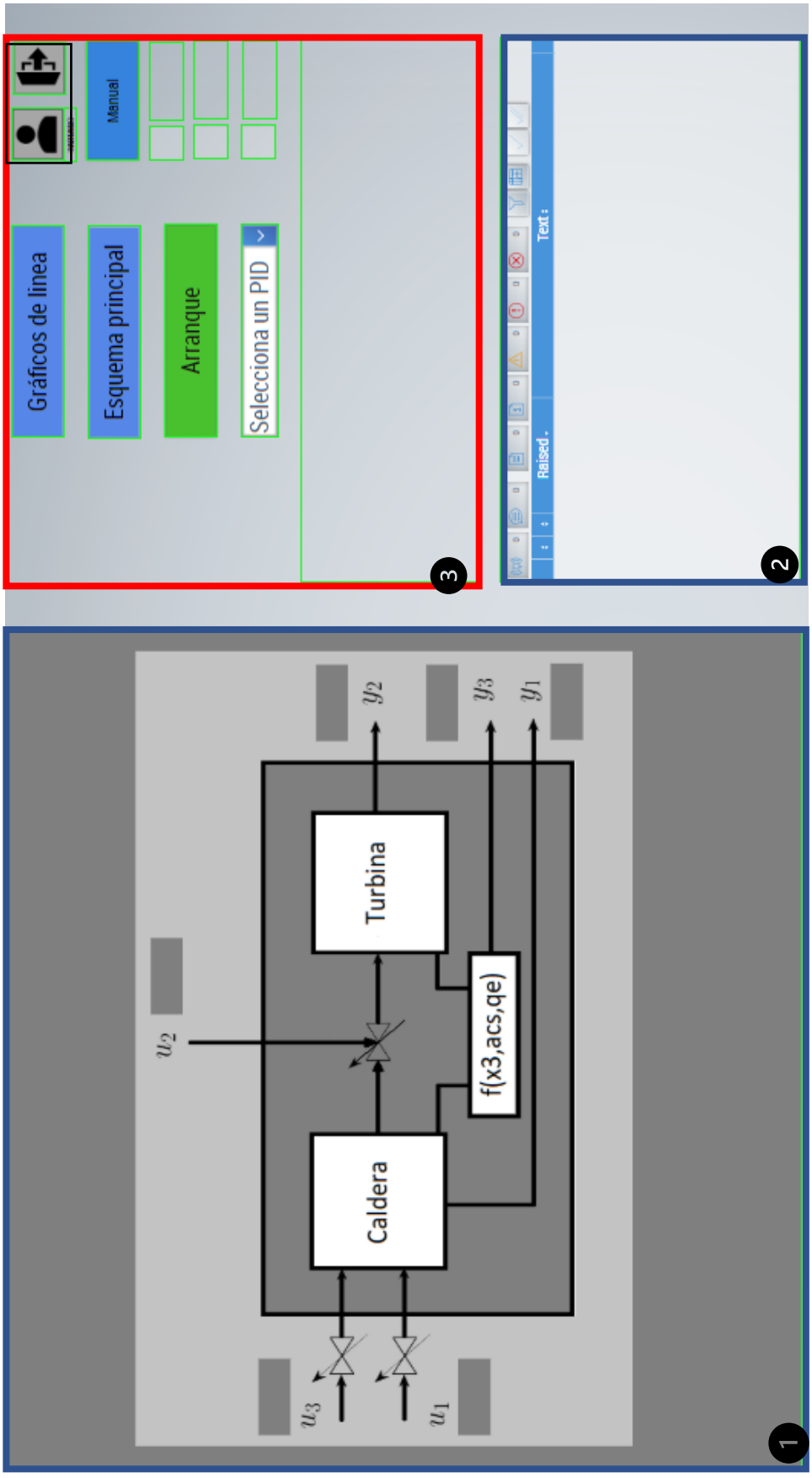


Ilustración 37 Pantalla táctil HMI (Azul = Información / Rojo = Control)

El apartado de información muestra el estado actual de las variables del proceso y los eventos que están ocurriendo (Cuadrados azules). Por otro lado, existen distintos controles (Botones, barra desplegable, textbox…) que permiten escribir en las variables del programa MAIN. Por ejemplo, el botón "Arranque", cambia el valor de la variable binaria "bExecutePrgm", si esta variable es verdadera el sistema de control se inicia.

Esta pantalla ha sido desarrollada utilizando la extensión *Twincat HMI Engineering,* que mediante una interfaz gráfica permite diseñar los objetos de información y control, así como sus interacciones de manera muy sencilla.

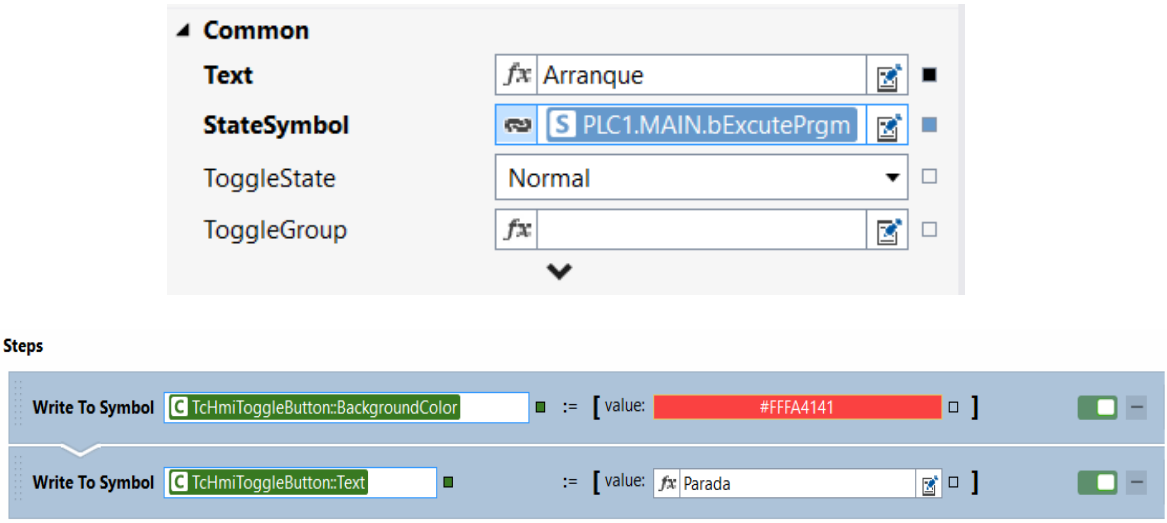


Ilustración 38 Acción al pulsar el botón de arranque

Se han establecido dos tipos de usuario que pueden acceder al sistema: OPERARIO y ADMINISTRADOR. Por defecto el sistema se inicia con el usuario OPERARIO, pero pulsando el control con el icono de una persona (Rectángulo negro de la Ilustración 37) se accede a una página de inicio de sesión, con la opción de cambiar al modo administrador. Existen distintos controles que son accesibles en modo ADMINISTRADOR, mientras que en modo OPERARIO no está permitido su uso. El árbol de contenidos de este módulo se encuentra en la siguiente ilustración.

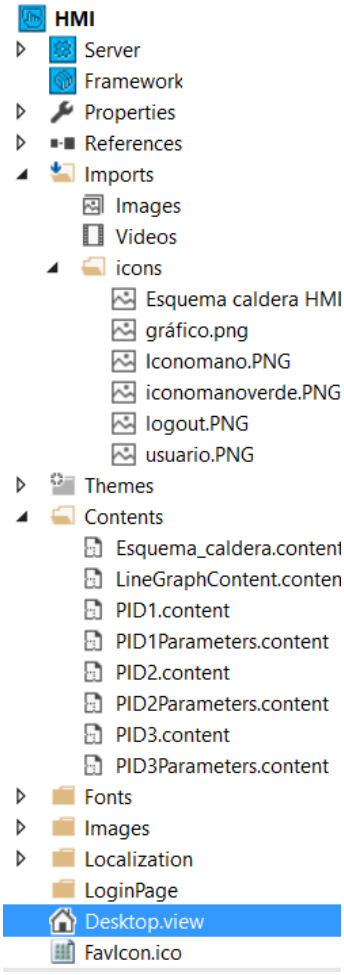


Ilustración 39 Árbol de contenidos del módulo HMI

La pantalla principal y todos sus objetos están contenidos en el archivo Desktop.view, pero existen otras pantallas (Archivos *Content*) que irán apareciendo cuando se les llame desde algún control disponible en la pantalla principal. La pantalla se ha dividido en tres partes:

- Información 1: Hace referencia a los distintos esquemas informativos que se puede tener en la pantalla táctil.
- Información 2: Alarmas y mensajes que pueden ser leídos en el *EventLogger.*
- Control: Objetos que permiten al usuario cambiar parámetros de la pantalla táctil o del sistema de control.

1. Información 1:

Este apartado hace referencia al rectángulo azul número uno, de la Ilustración

37. Esta zona contiene un bloque que se denomina *Region,* su cualidad principal es poder mostrar los archivos *Content*, y no cuenta con ningún control, por defecto aparece Esquema_caldera.content.

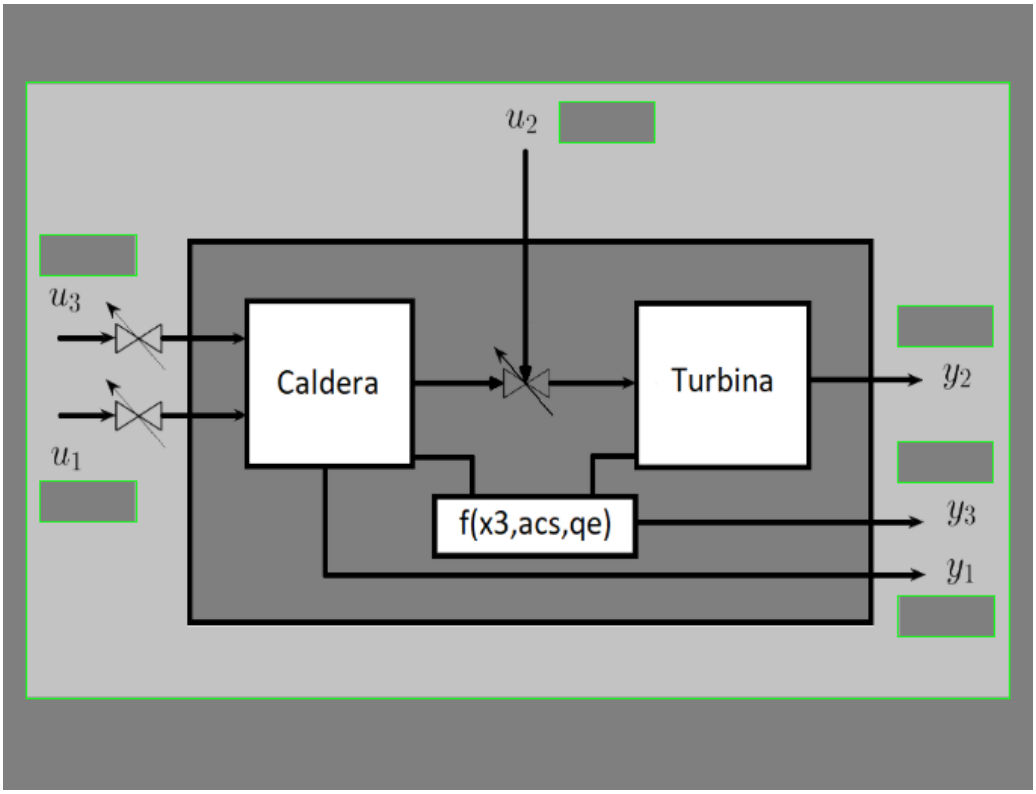


Ilustración 40 Esquema_caldera.content

En la Ilustración 40 se ve un esquema simplificado del proceso simulado, al lado de cada una de las entradas y salidas, se ha dispuesto un indicador en el que aparece el valor actual de la variable. Estos valores son siempre visibles tanto para el OPERARIO como para el administrador.

La segunda posibilidad de esta región es mostrar gráficos de línea de los últimos 800s de la planta. Estos gráficos muestran los valores de los vectores aPoints, como están estructurados y su inicialización está explicado en los pasos 4 y 5 del apartado 4.2.1. Existe la posibilidad de ver las seis variables a la vez si se pulsa el control "Gráficos de línea" del rectángulo número tres de la Ilustración 37, que cambiará el contenido de la *Region* a LineGraphContent.content. Si solo se desea ver cómo está funcionando uno de los lazos de control, se puede mostrar los gráficos de cada lazo por separado (PID1.content / PID2.content / PID3.content).

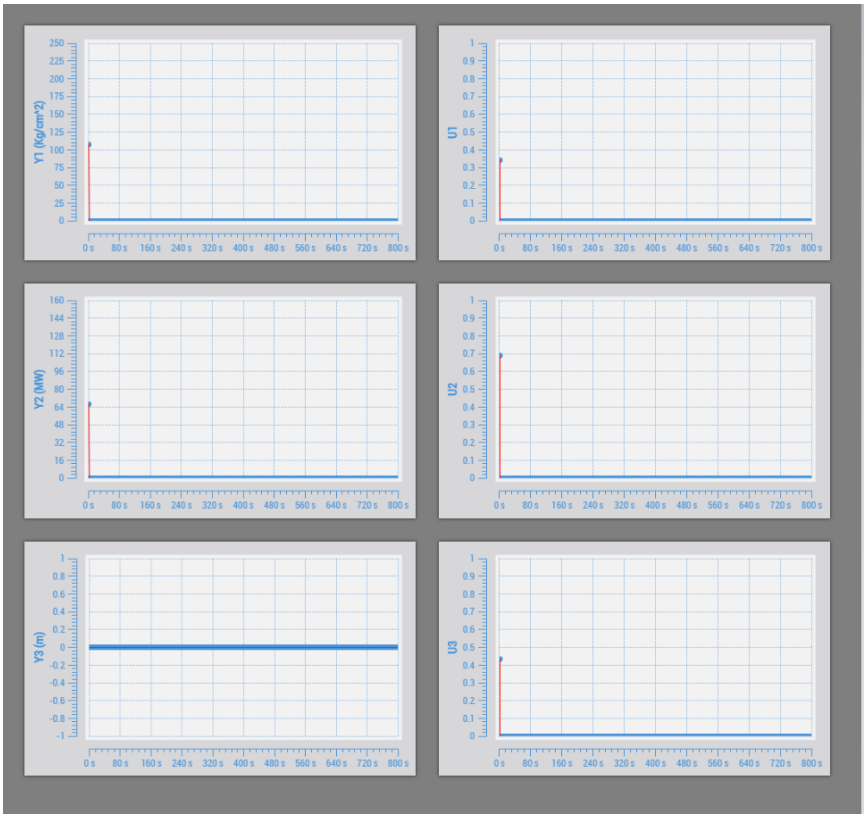


Ilustración 41 LineGraphContent.content

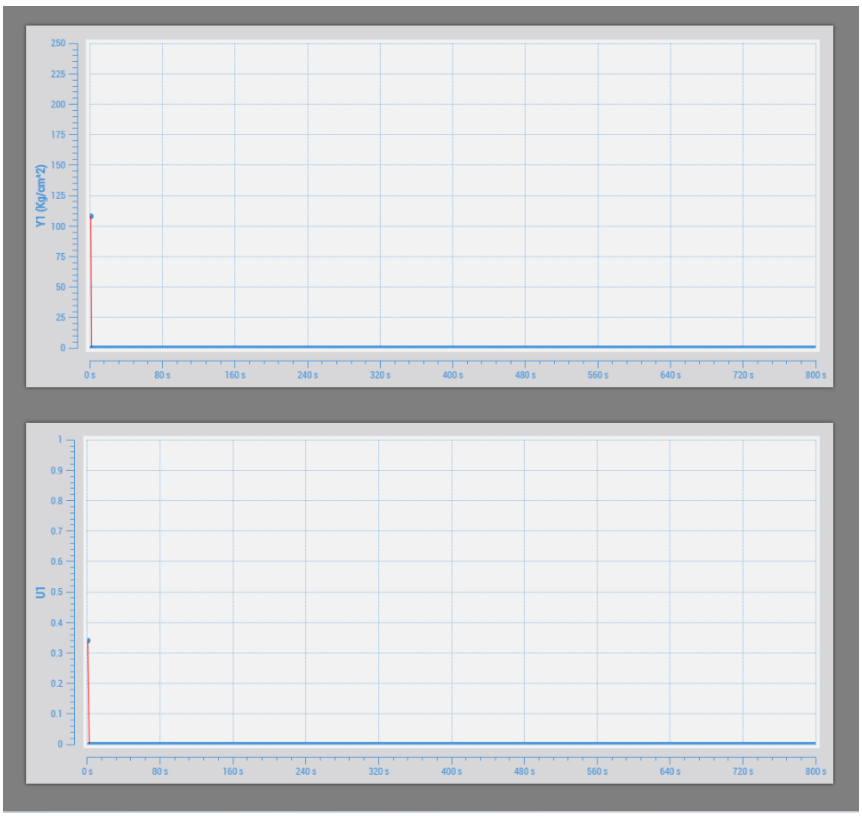


Ilustración 42 PID1.content

2. Información 2:

Este bloque se ha separado del anterior porque tiene una funcionalidad distinta, el rectángulo número dos de la Ilustración 37 es un indicador de alarmas y mensajes, el bloque se denomina *EventLogger*. La información que aparece en esta sección se declara y ejecuta en el módulo PLC, donde previamente se han creado las alarmas y mensajes y se ejecutan a lo largo del funcionamiento del sistema de control (Las alarmas están definidas en la Tabla 2). Cuando se cumplan las condiciones para que una de las alarmas/mensajes se active. aparecerá en esta ventana de información.

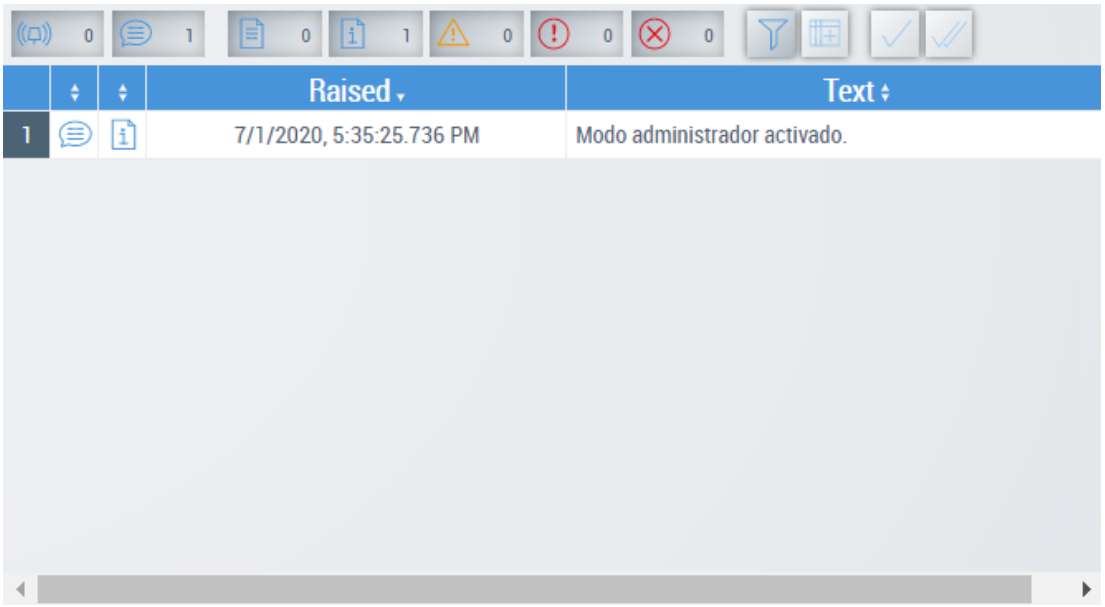


Ilustración 43 EventLogger

El *EventLogger* muestra el tipo de mensaje que se ha activado en la primera y segunda columna, Info/Warning, la comulna de *Raised* hace referencia al momento en el que se ha enviado el mensaje, y en la última columna se ubica el texto asociado a cada una de las alarmas o mensajes.

3. Controles:

Finalmente, en este apartado se van a mostrar los distintos controles disponibles para el usuario. Se puede interactuar con la pantalla mediante tres tipos de objetos: botones, ventana desplegable y *textbox*.

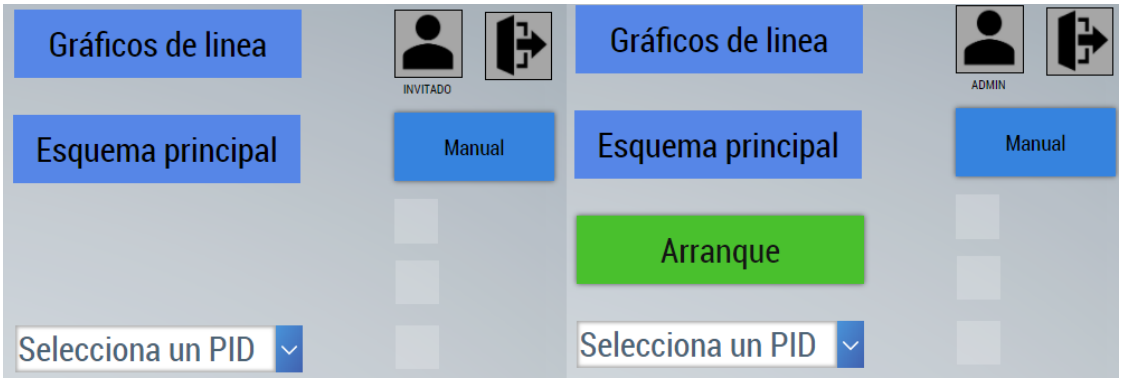


Ilustración 44 Controles de la pantalla táctil

Los botones "Gráficos de línea" y "Esquema principal" son siempre visibles y accesibles, y permiten cambiar el contenido de la *Region* explicada en el apartado 1 Información 1.

- Gráficos de línea → LineGraphContent.content
- Esquema principal → Esquema_caldera.content

Ambos botones están esperando a ser presionados por el usuario, cuando sucede esta acción se ejecuta el siguiente código:

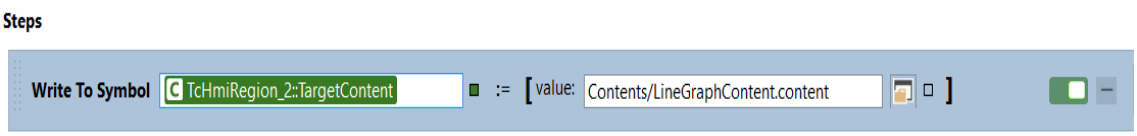


Ilustración 45 Cambio de contenido de la Region (Infomación 1)

En la Ilustración 44 se muestra la diferencia principal entre el usuario OPERARIO (Izquierda) y el ADMINISTRADOR (Derecha), el botón de arranque tan solo está disponible en el modo ADMINISTRADOR, es decir, solo es posible arrancar/parar el sistema de control en este modo. A pesar de que no sea observable en el modo OPERARIO, si el control se inicia y después se cambia de usuario, el control no sufre ningún cambio y sigue funcionando con normalidad, pero habrá que cambiar de usuario de nuevo si se desea detenerlo. Este control es un *ToggleButton* que está ligado a la variable bExecutePrgm, por lo que una vez pulsado no cambiará el estado de esta variable hasta que se vuelva a pulsar.

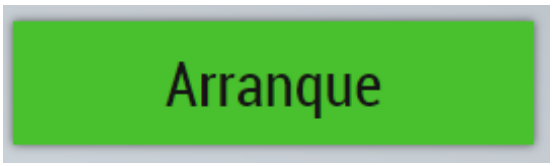


Ilustración 46 Estado inicial (bExecutePrgm = FALSE)

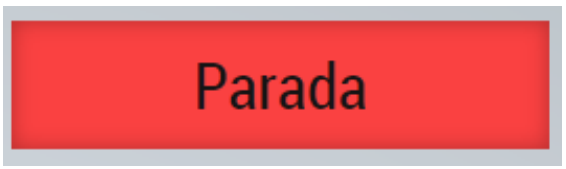


Ilustración 47 Estado final (bExecutePrgm = TRUE)

A continuación, está la ventana desplegable, este control maneja otro objeto de tipo *Region* distinto al explicado con anterioridad. El contenido que va a mostrar al pulsar sobre la opción deseada es: PID1Parameters.content, PID2Parameters.content y PID3Parameters.content.

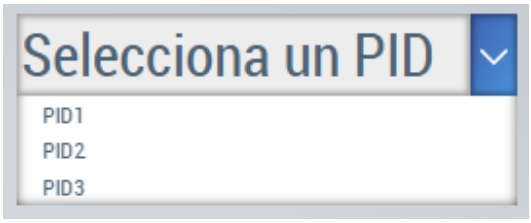


Ilustración 48 Ventana desplegable

Al seleccionar cualquiera de sus elementos aparece el siguiente contenedor de controles:

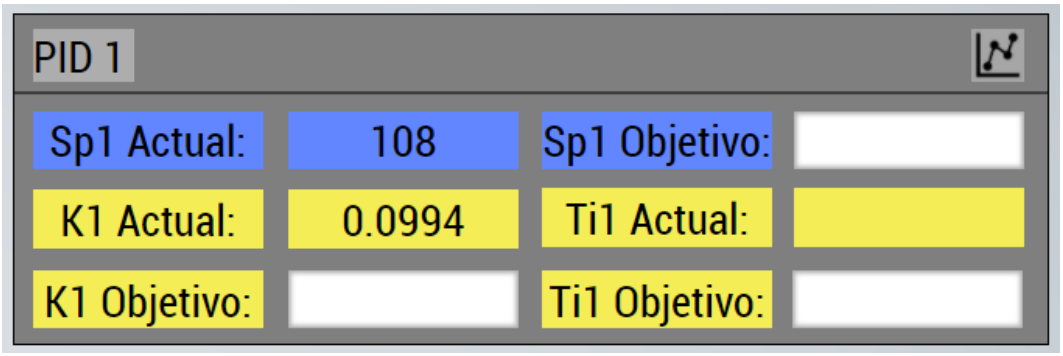


Ilustración 49 Contenedor de controles del PID1 (PID1Parameters.content)

En esta ventana hay indicadores y controles, se puede observar qué PID se muestra y cuál es la referencia, ganancia y tiempo de integraciones actuales. Los controles disponibles son los *textbox* y el botón ubicado en la esquina superior

derecha.

El botón con el icono de una gráfica mostrará en el apartado de Información 1, las gráficas relativas al lazo de control del PID seleccionado, esta pantalla se muestra en la Ilustración 42. Por otro lado, se pueden ver tres rectángulos blancos, en los que el usuario puede introducir los valores deseados para los distintos parámetros. Cualquier usuario puede realizar un cambio en la referencia (*Textbox* → Sp1 Objetivo), pero solo el ADMINISTRADOR, tendrá permiso para modificar el valor de la ganancia y del tiempo de integración.

El uso de *textbox* conlleva un problema, para que este control funcione es necesario introducir un número, y en el caso de que tenga parte decimal, esta debe ir separada por un punto. Para evitar errores se ha programado un código en *JavaScript* que comprueba si el dato introducido en el *textbox* es un número, en el caso de introducir algún carácter o número decimal separado por coma, aparecerá una ventana de alarma y no se actualizará el valor de la variable asociada.

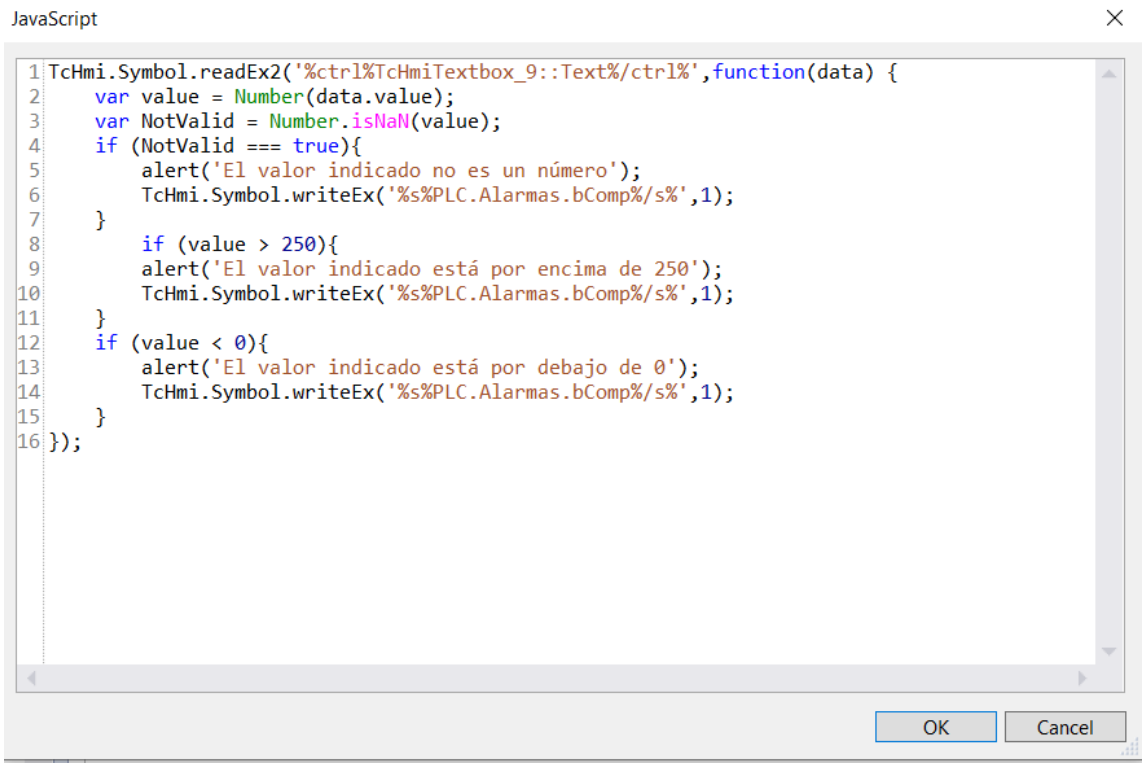


Ilustración 50 Código en JavaScript

Además de comprobar si es un número, evalúa si el número introducido está dentro de un rango, ya que fuera de este no tiene sentido introducir valores para la referencia. Los intervalos establecidos son:

- SP1 → [0 250]
- SP2 → [0 160]
- SP3 → [-1 1]

Los dos botones que aparecen en la Ilustración 51, están relacionados con los usuarios disponibles, OPERARIO y ADMINISTRADOR. El botón con el icono de una persona nos traslada a la página de inicio de sesión (Ilustración 52), en la cual se puede seleccionar el usuario deseado e introducir su contraseña.

- USUARIO: OPERARIO / CONTRASEÑA: operario
- USUARIO: ADMINISTRADOR / CONTRASEÑA: administrador

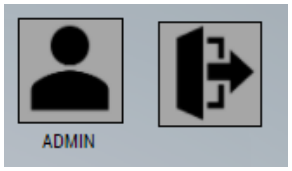


Ilustración 51 Botones de iniciar/cerrar sesión

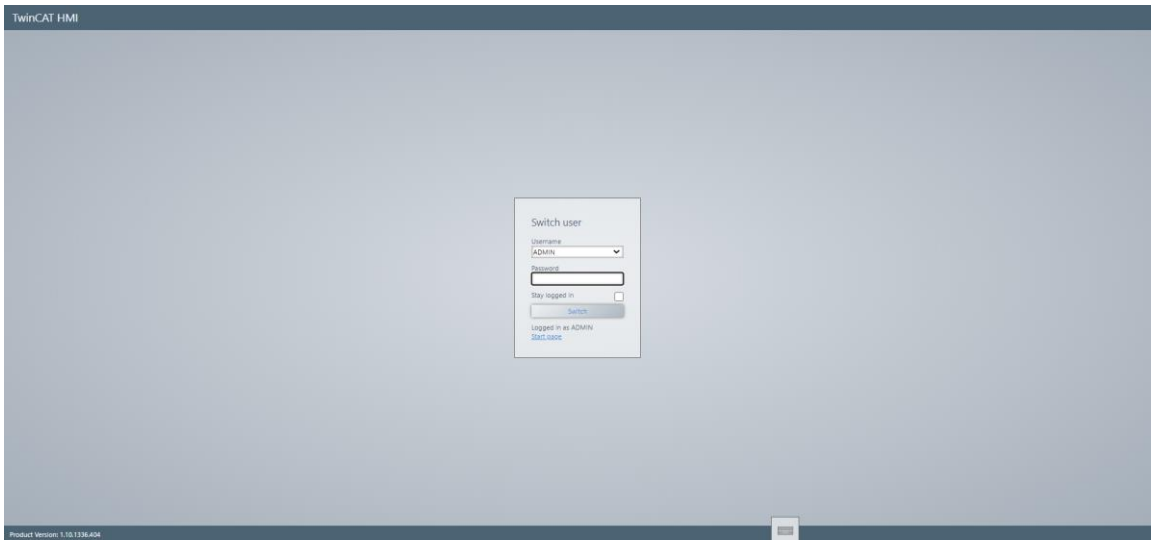


Ilustración 52 Página de inicio de sesión

El segundo botón de la Ilustración 51, automáticamente cierra la sesión con el usuario actual, y lo cambia al usuario por defecto, OPERARIO.

	OPERARIO	ADMINISTRADOR
Cambio de referencia	✔	✔
Cambio de los parámetros del PID	X	✔
Acceso al modo manual	✔	✔
Arranque/Parada	X	✔

Tabla 3 Permisos de OPERARIO y ADMINISTRADOR

El último grupo de controles gestiona el modo de funcionamiento de los PIDs, permite cambiar a modo manual cada uno de los lazos de control, pudiendo así introducir el valor de apertura de cada válvula por separado. La apariencia inicial es la siguiente:

Ilustración 53 Apariencia inicial del control manual

Al pulsar sobre el botón "Manual/Automático" se hacen visibles y se activan los controles de este apartado.

Ilustración 54 Apariencia del control manual

En la Ilustración 54 se ven tres botones con el icono de una mano, inicialmente tienen el fondo gris, lo cual indica que el lazo todavía está en modo automático, al pulsar sobre este icono se activa la *textbox* correspondiente, tras introducir un valor adecuado, el PID pasa a modo manual y como salida se obtiene el valor escrito en este control. Si se vuelve a pulsar el icono de la mano una vez está en verde, el lazo en cuestión vuelve a modo automático y calcula la acción de control a cada iteración del programa. De la misma manera si se pulsa de nuevo el botón "Manual/Automático" y uno de los lazos de control está en modo manual, se cambia a modo automático todos los PIDs.

Como los controles utilizados en este apartado son *textbox* presentan el mismo problema que se ha mencionado anteriormente, y se aplica un código en *JavaScript* similar al de la Ilustración 50. De manera que, si se introduce algún carácter o un numero decimal separado por coma, mostrará un mensaje de error, también se comprueba, en el caso de que el valor introducido sea un número, si el valor está dentro del rango de apertura de las válvulas [0 1], en caso de que esté fuera, también muestra un mensaje de error. En cualquiera de los casos que se produzca un error con este control, no escribirá en la variable asociada al control.

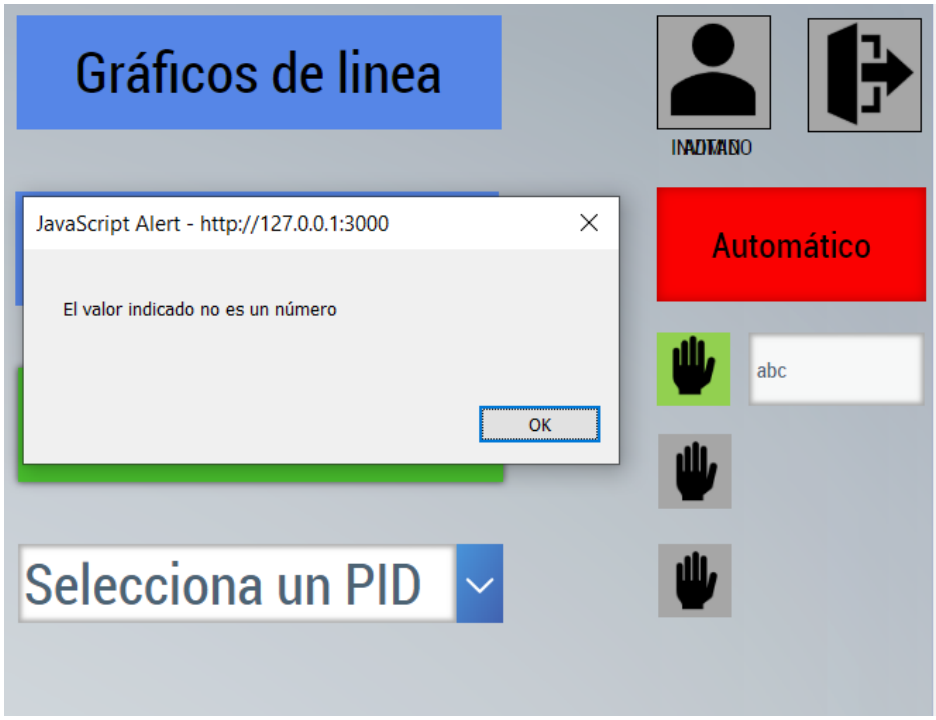


Ilustración 55 Mensaje de error debido a un valor erróneo en el control

Tampoco está permitido cambiar de usuario si el modo manual de alguno de los PIDs está activado, en el caso de pulsar el botón de iniciar/cerrar sesión, se muestra otro mensaje de error, programado también en *JavaScript*, indicando que se debe pasar a modo automático para cambiar de usuario.

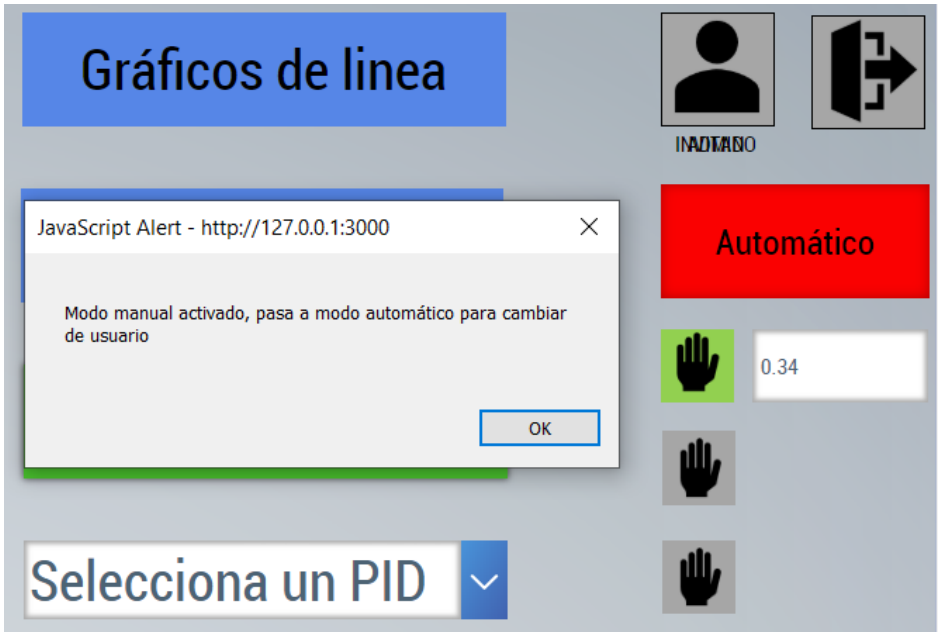


Ilustración 56 Mensaje de error al intentar cambiar de usuario en modo manual

4.3 DISEÑO ÓPTIMO DE PIDs MEDIANTE ALGORÍTMOS GENÉTICOS

El diseño de los PIDs se ha abordado utilizando algoritmos genéticos para minimizar la función de coste, nombrada con la letra J. El punto de partida es un control general sin ningún objetivo concreto, se ha planteado una optimización multiobjetivo a partir de J, buscando el valor mínimo de esta función dentro de unas restricciones establecidas. El diseño original es el mostrado en la Ilustración 57, en la que se representa el sistema de control (Cuadro azul) y la simulación de la planta (Cuadrado rojo).

El sistema de control es un esquema multibucle, que contiene tres PIs con realimentación negativa, y cada uno de ellos controla una de las tres salidas, el emparejamiento es 1-1 2-2 y 3-3. El sistema se sitúa en el punto de funcionamiento número 4 que aparece en el documento "Analysis and control of a nonlinear boiler-turbine unit" (Tan, Márquez, Tongwen, & Jizhen, 2005), mediante un script en *Matlab* se cargan los valores especificados para el punto 4 de funcionamiento en las variables que se ven en la Tabla 4.

Y1_0	108.0 Kg/cm^2
Y2_0	66.65 MW
Y3_0	0.0 m
U1_0	0.34 ‰
U2_0	0.69 ‰
U3_0	0.433 ‰

Tabla 4 Punto de funcionamiento número 4

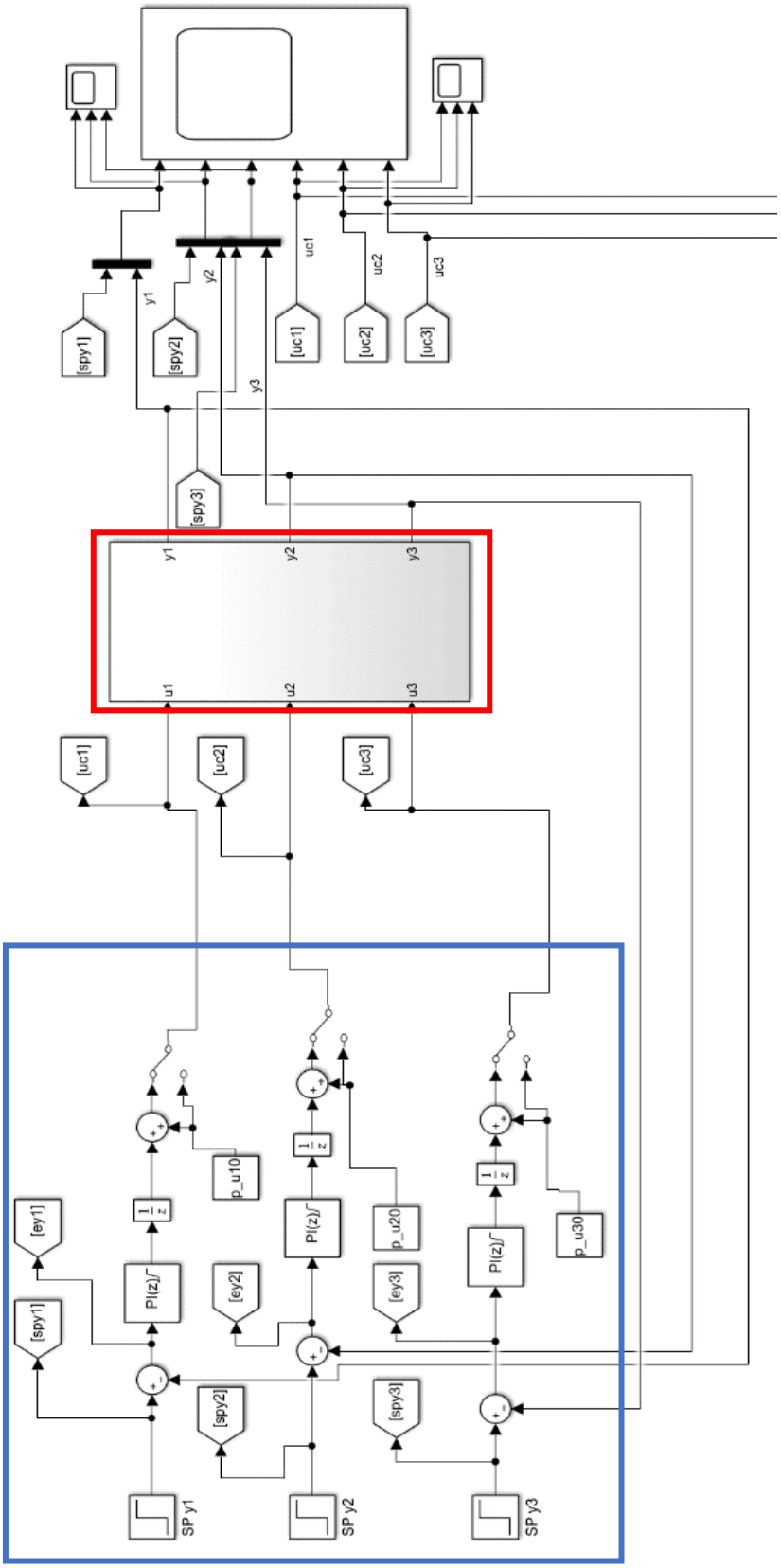


Ilustración 57 Esquema de control en Simulink / Control → Cuadro azul / Simulación de la planta →

Cuadro rojo

Los parámetros iniciales de los controladores son:

Kc1	0.0485
Ti1	40.42
Kc2	0.0197
Ti2	4.38
Kc3	7.25
Ti3	24.9

Tabla 5 Parámetros iniciales de los controladores

Se han configurado los PIs para que las acciones de control estén limitadas en el rango [0 1], debido a que las válvulas tienen limitaciones de apertura. Además, para que la simulación se parezca más al control que se va a realizar en *TwinCat,* se ha añadido el bloque *Unit delay* de *Simulink*, el cual proporciona un retraso de 1s a las acciones de control, tal y como pasa en el control a tiempo real de *TwinCat*.

El otro bloque de este programa es la simulación de la planta de generación, se ha implementado dentro de un *subsystem*, el contenido de este se muestra en la Ilustración 59.

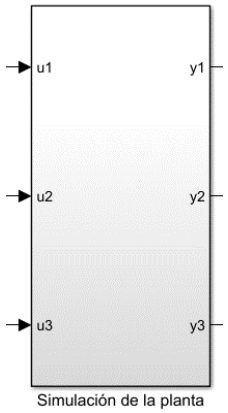


Ilustración 58 Subsytem de simulación de la planta

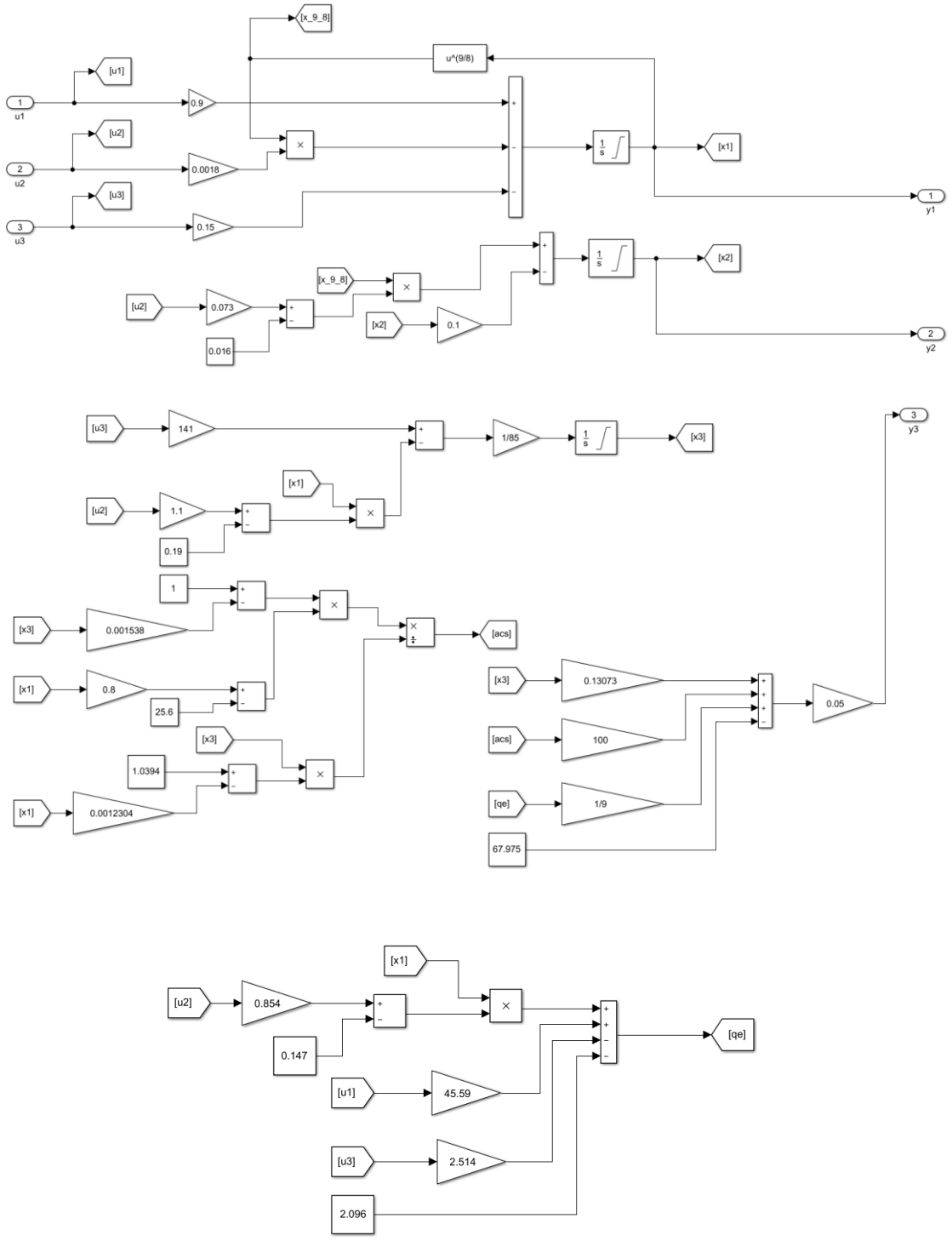


Ilustración 59 Simulación de la planta de generación en Simulink

Todo el contenido de este *subsystem* es equivalente a las ecuaciones 1-8 del apartado 4.1.1, pero en este caso programado con bloques de *Simulink*, las acciones de control u1, u2 y u3 son las entradas del bloque y las salidas y1, y2 e y3 los valores de las magnitudes de salida.

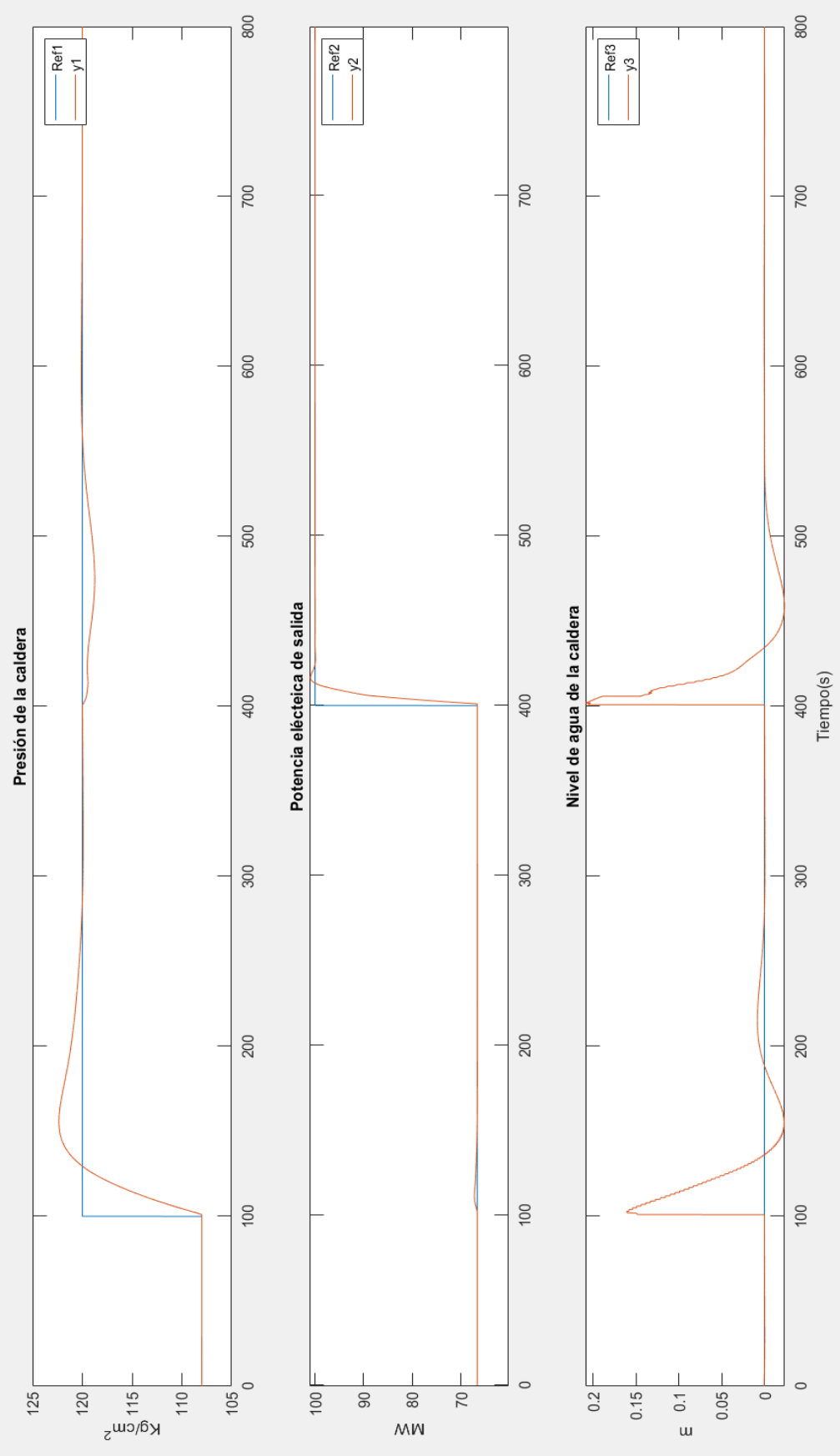


Ilustración 60 Respuesta ante el experimento diseñado

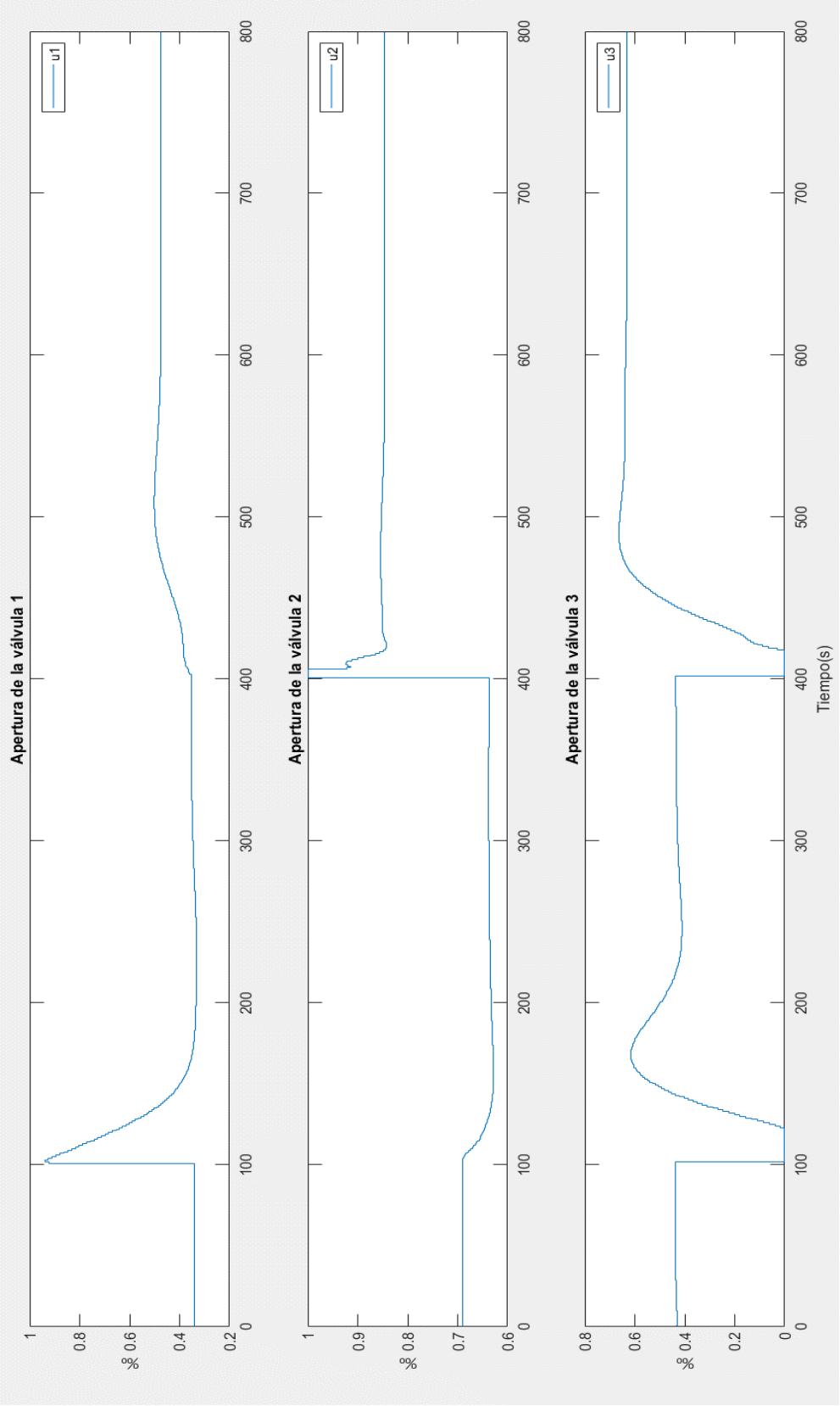


Ilustración 61 Acciones de control calculadas

Se ha programado un experimento con los siguientes cambios de referencia:

Referencia 1	108.0	120.0	T=100s
Referencia 2	66.65	100.0	T=400s

Tabla 6 Cambios de referencia en el experimento diseñado

El resultado de este experimento con el control inicial tiene las señales que se muestran en la Ilustración 60 e Ilustración 61. A partir de este primer experimento de control, se ha diseñado la optimización multiobjetivo mediante el algoritmo genético, los objetivos establecidos son:

- Reducir el valor de la integral del error absoluto de y1, y2 e y3 (IAE).
- Reducir el valor de la integral de la variación de las acciones de control u1, u2 y u3.

Para tener una referencia se calculan estos seis valores con el experimento inicial, mediante los siguientes bloques:

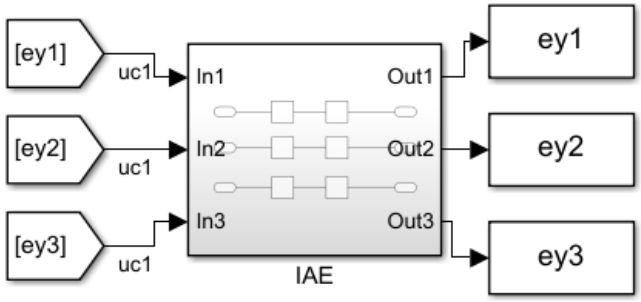


Ilustración 62 Subsystem para el cálculo de los objetivos ey1, ey2 y ey3 mediante IAE

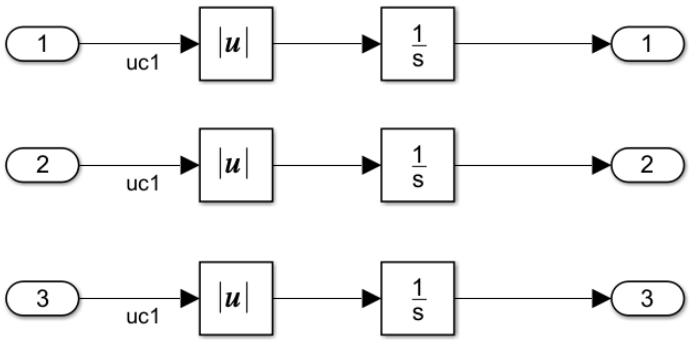


Ilustración 63 Contenido del subsystem IAE de la ilustración anterior

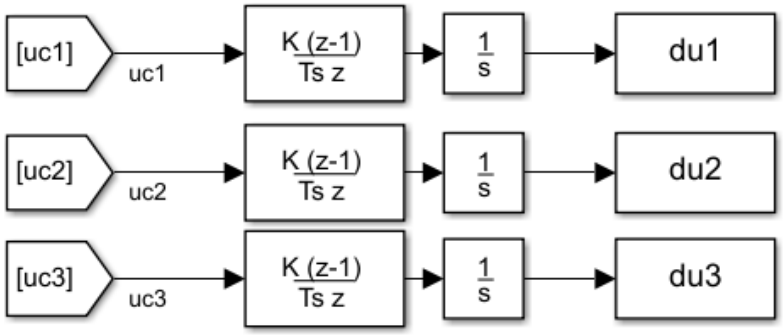


Ilustración 64 Cálculo de los objetivos du1, du2 y du3

Los valores de referencia para los objetivos se calculan mediante los bloques que aparecen en la Ilustración 62, Ilustración 63 e Ilustración 64, y se exportan al *Workspace* de Matlab mediante el bloque de *Simulink To Workspace,* los valores calculados son:

Du1_ref	0.4749
Du2_ref	0.8467
Du3_ref	0.6309
Ey1_ref	446.0795
Ey2_ref	196.5406
Ey3_ref	8.0576

Tabla 7 Valores de referencia para la optimización multiobjetivo

A partir del control inicial y el valor de referencia de los seis objetivos, se ha creado en *Matlab* un *script* en el que se define la función de coste que se pretende optimizar mediante el algoritmo genético. Este *script* se denomina "coste.m", como entrada a la función se pide el valor de los seis parámetros de los PIDs → x = [Kc1 Ti1 Kc2 Ti2 Kc3 Ti3] incluidos en un solo vector de datos, y como salida se obtiene el valor de J calculado para el valor de x proporcionado. Los pasos que realiza la función "coste.m" son:

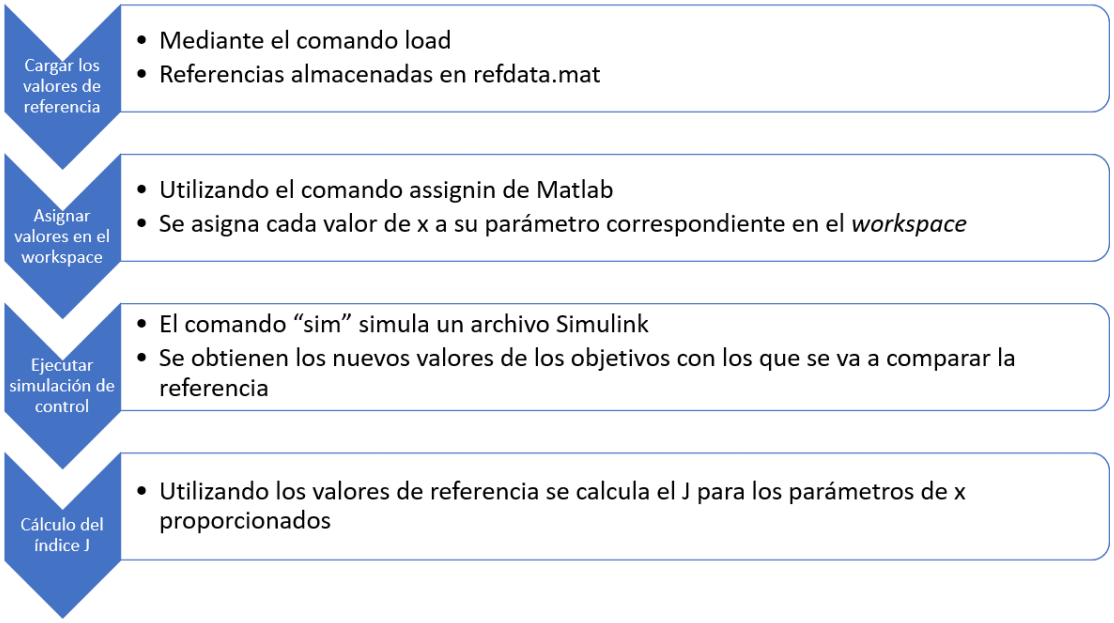


Ilustración 65 Flujograma de la función "coste.m"

```
function J=coste(x)

load ('refdata.mat');
assignin('base','Kc1',x(1));
assignin('base','Ti1',x(2));
assignin('base','Kc2',x(3));
assignin('base','Ti2',x(4));
assignin('base','Kc3',x(5));
assignin('base','Ti3',x(6));

sim('modelo_caldera_LP1')

cte1=1/ey1_ref;
cte2=1/ey2_ref;
cte3=1/ey3_ref;
cte4=1/du1_ref;
cte5=1/du2_ref;
cte6=1/du3_ref;
J=(cte1*ey1+cte2*ey2+cte3*ey3+(cte4*du1+cte5*du2+cte6*du3)*1.75)/(3+3*1.75);
```

Ilustración 66 Función "coste.m"

La función de coste J está definida en la última línea de código que se ve en la Ilustración 66, y está normalizada para que si se ejecuta el control de referencia, la J obtenida sea J=1, cualquier control que obtenga un valor de J<1 significará que es un control que cumple mejor el objetivo propuesto. Por eso las constantes que ponderan cada uno de los objetivos, es la inversa del valor de referencia de ese mismo objetivo. Además, se ha realizado una segunda ponderación, dando

un 75% más de importancia al objetivo relacionado con la variación de las acciones de control. Con esta última ponderación se busca un control que no sea muy agresivo, y que le dé más importancia a variar poco las acciones de control que a corregir rápidamente el error, ya que un control muy brusco puede llegar a inestabilizar el sistema.

Una vez programado el control de referencia y la función de coste, se aplica el algoritmo genético *ga()* que proporciona la *Global Optimization Toolbox* de *Matlab* para encontrar la solución óptima dentro de las restricciones establecidas. Para ello se ha creado el script "algoritmo_genetico.m" con el siguiente flujograma:

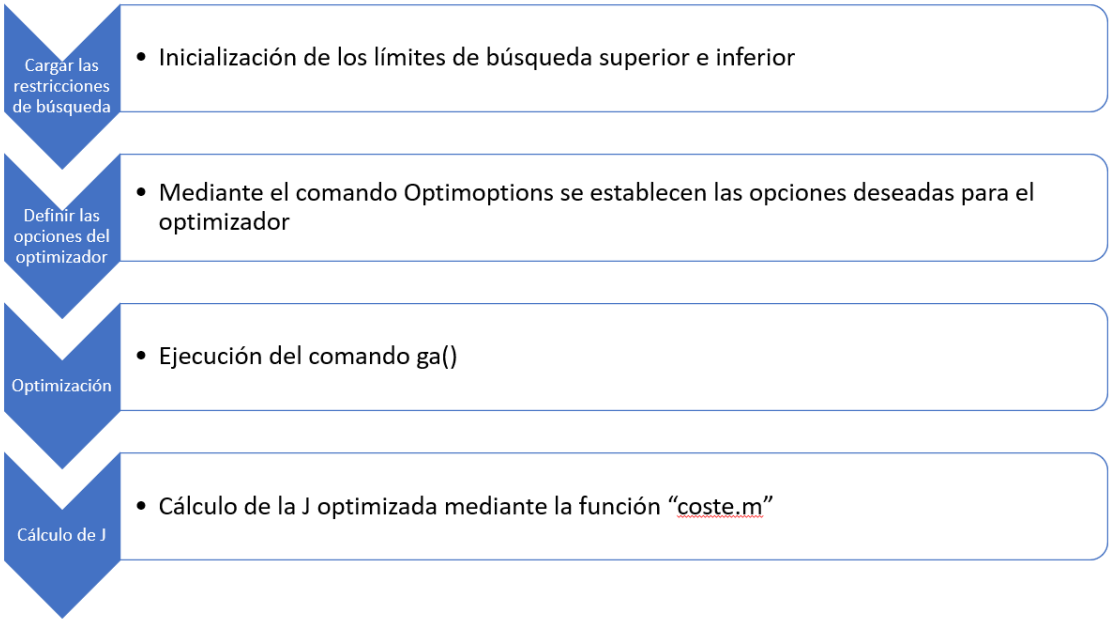


Ilustración 67 Flujograma de "algoritmo_genético.m"

Este código tarda bastante en completarse, calcula una solución inicial, escoge los mejores individuos y comienza el proceso de cruce para generar una nueva población, tras evaluar a los nuevos individuos el proceso comienza de nuevo cruzando los nuevos individuos "mejorados" hasta que converja en una solución, pudiendo llegar a calcular más de 150 generaciones y llegando a tiempos de cálculo de 5h o más.

```
load('refdata.mat');

lb = [0.001,10,0.001,3.9,7,20];
ub = [0.2,100,0.03,50,7.5,30];

fun = @coste;
options = optimoptions(@ga,"Display","iter");

tic
x = ga(fun,6,[],[],[],[],lb,ub,[],options)
toc

J=coste(x)
```

Ilustración 68 Código de "algoritmo_genetico.m"

Tras varias pruebas, optimizador llegó a una solución válida, tardó alrededor de 20000s y proporcionó el siguiente conjunto de valores óptimos para los PIDs:

Kc1	0.2
Ti1	31.5992
Kc2	0.0252
Ti2	4.3842
Kc3	7.5
Ti3	20.0

Tabla 8 Valores óptimos que ha calculado el algoritmo genético

Con una reducción del índice de coste del 10.3%.

```
>> J=coste(x)

J =

    0.8970
```

Ilustración 69 Valor de J para los parámetros óptimos

Para concluir el apartado se muestran las señales obtenidas con los parámetros de la Tabla 8:

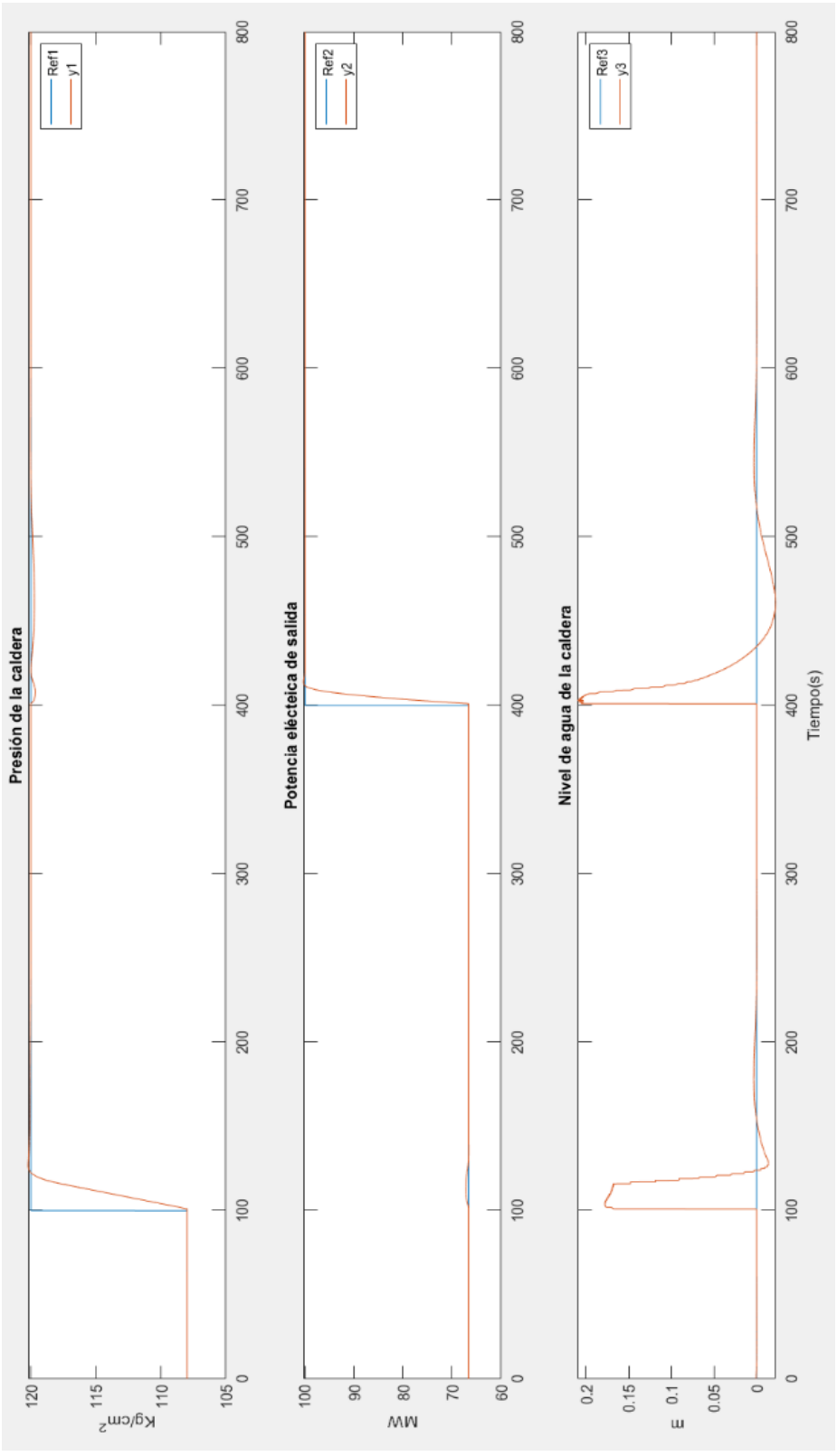


Ilustración 70 Respuesta ante el experimento diseñado con los valores óptimos para los PIDs

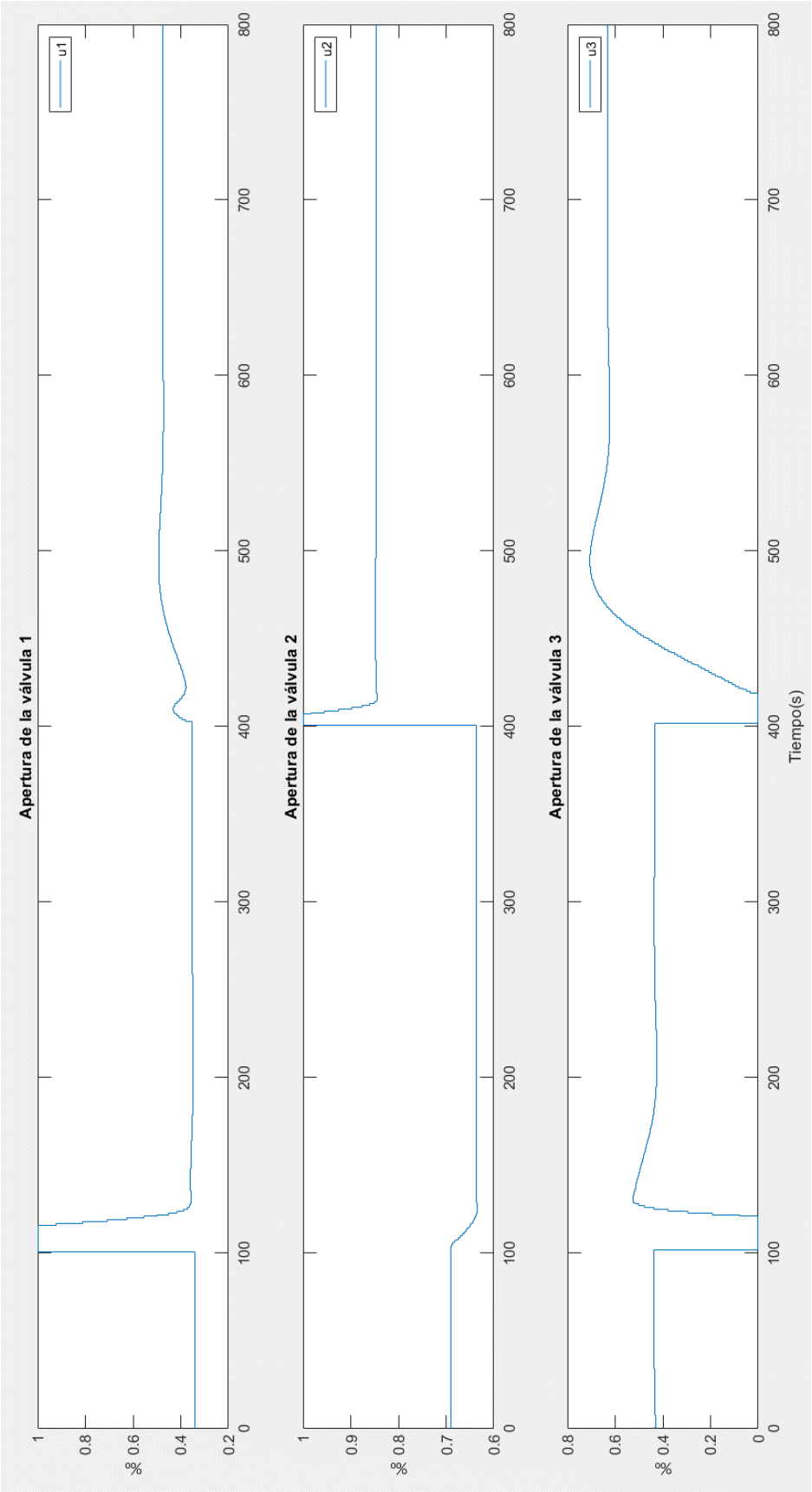


Ilustración 71 Acciones de control calculadas con los valores óptimos para los PIDs

5 CONCLUSIONES

Finalmente, se puede afirmar que se ha diseñado un sistema de control satisfactorio para la planta de generación, cumpliendo los objetivos propuestos para este trabajo. Mediante *HIL* y programado en *Python* se ha creado una simulación que responde tal y como lo haría la planta real, también se ha diseñado un sistema de control que cumple los requisitos establecidos y que combina un módulo PLC y otro HMI, para que los usuarios tengan un acceso sencillo a las funcionalidades del sistema de control, además este sistema de control ha sido optimizado mediante algoritmos genéticos para obtener una combinación de parámetros de los PIDs que cumpliera con los objetivos (mínimo error de posición y mínima variación de las acciones de control) mejor que el control inicial propuesto.

Este trabajo ha proporcionado al alumno conocimiento sobre HIL, programación en *Python*, algoritmos genéticos y *TwinCat 3*, pero es este último *software* el que tiene más relevancia. Se ha aprendido a utilizar un programa que no se ha utilizado a lo largo del máster, con una idea novedosa en el departamento, control en tiempo real basado en PC, esto permite explorar otras posibilidades de control más allá de PLCs y microcontroladores. Este trabajo puede ser útil para futuros TFM o proyectos del departamento en el que se quiera usar *TwinCat 3*, ya que plantea las bases de este programa y su funcionamiento, podría extenderse o modificarse buscando utilizar otros módulos como C/C++ o la combinación con *Matlab/Simulink.*

Para concluir este apartado se muestra el entorno de *TwinCat 3* (HMI) controlando el experimento diseñado para el algoritmo genético, si se compara con la Ilustración 70 e Ilustración 71, se puede ver la gran similitud entre el experimento ejecutado en *Simulink* y el de *TwinCat 3.*

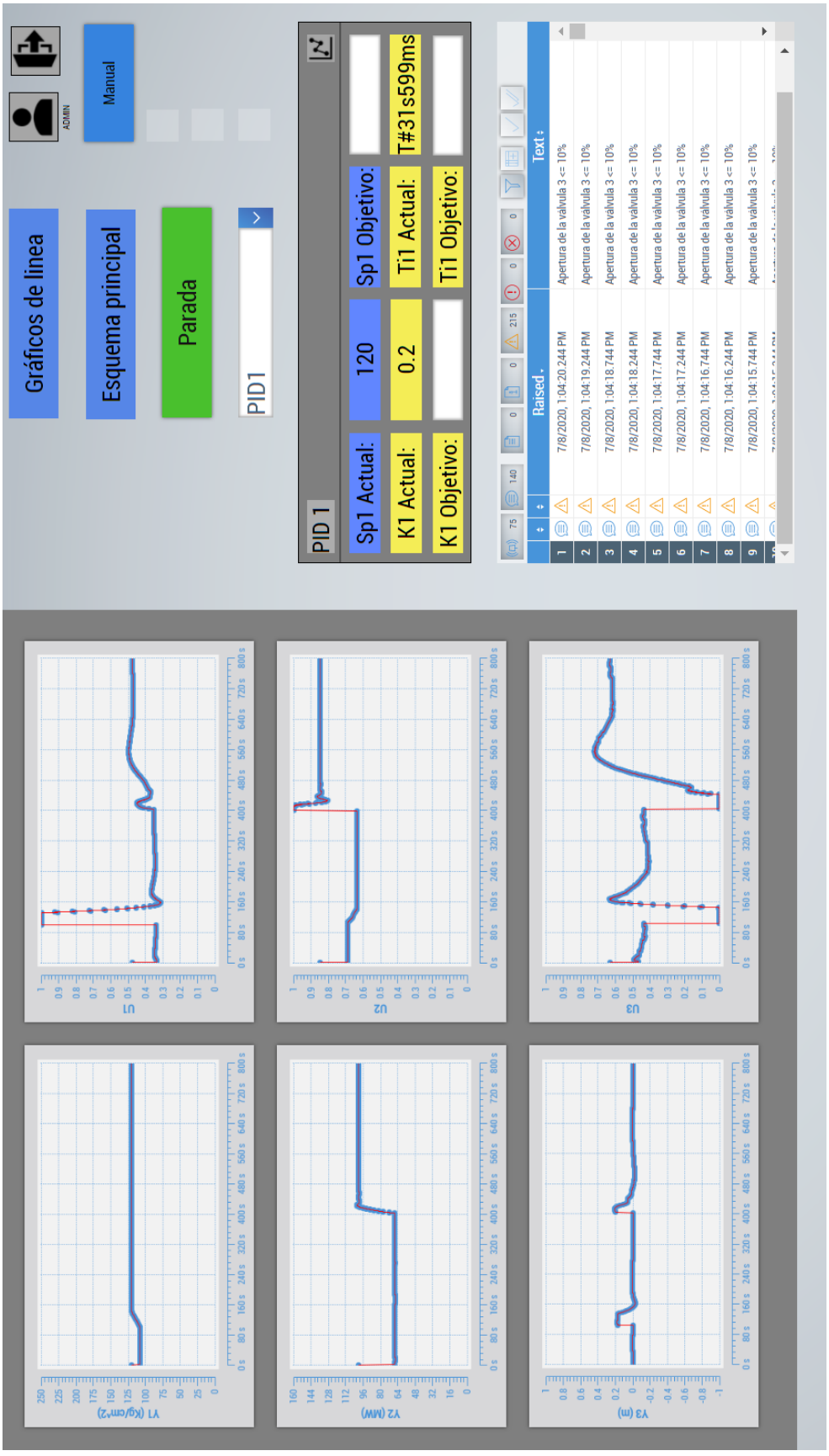


Ilustración 72 Control del experimento diseñado mediante TwinCat 3.

6 BIBLIOGRAFÍA

Arévalo Ovalle, D., Bernal Yermanos, M. Á., & Posada Restrepo, J. A. (2017). *Matemáticas para ingenieria: Métodos numéricos con Python.* Bogotá: Editorial Politécnico Grancolombiano.

Beckhoff. (30 de Octubre de 2018). *Beckhoff.com*. Obtenido de Beckhoff.com: https://download.beckhoff.com/download/document/automation/twincat3/TF2000_TC3_HMI_Server_EN.pdf

Beckhoff. (9 de Julio de 2019). *Beckhoff.com*. Obtenido de Beckhoff.com: https://download.beckhoff.com/download/document/automation/twincat3/TF4100_TC3_Controller_Toolbox_EN.pdf

Beckhoff. (19 de Febrero de 2020). *Bechoff.com*. Obtenido de Beckhoff.com: https://download.beckhoff.com/download/document/automation/twincat3/TF6250_TC3_Modbus_TCP_EN.pdf

Beckhoff. (15 de 06 de 2020). *beckhoff.com*. Obtenido de beckhoff.com: https://www.beckhoff.com/TwinCAT3/

Beckhoff. (7 de Julio de 2020). *beckhoff.com*. Obtenido de beckhoff.com: https://infosys.beckhoff.com/english.php?content=../content/1033/tcplclibcontrollertoolbox/html/tcplclibcontroller_overview.htm&id=

Beckhoff. (22 de Enero de 2020). *Beckhoff.com*. Obtenido de Beckhoff.com: https://download.beckhoff.com/download/document/automation/twincat3/TE2000_TC3_HMI_EN.pdf

Beckhoff. (7 de Julio de 2020). *Beckhoff.com*. Obtenido de Beckhoff.com: https://www.beckhoff.com/

Beckhoff. (7 de Julio de 2020). *Beckhoff.com*. Obtenido de Beckhoff.com: https://download.beckhoff.com/download/document/catalog/TwinCAT_3_Booklet.pdf

Beckhoff. (30 de Enero de 2020). *Beckhoff.com*. Obtenido de Beckhoff.com: https://www.beckhoff.com/TwinCAT-HMI/

Beckhoff. (07 de Julio de 2020). *Beckhoff.com*. Obtenido de Beckhoff.com: https://infosys.beckhoff.com/english.php?content=../content/1033/tcmodbussrv/html/tcmodbussrv_configuration.htm&id=

Beckhoff. (07 de Julio de 2020). *Beckhoff.com*. Obtenido de Becjhoff.com: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_eventlogger/27021602616987915.html&id=

Beckhoff. (29 de Enero de 2020). *Beckhoff.com*. Obtenido de Beckhoff.com: https://download.beckhoff.com/download/document/automation/twincat3/TE2000_TC3_HMI_EN.pdf

Beckhoff. (11 de Mayo de 2020). *Beckhoff.com*. Obtenido de Beckhoff.com: https://download.beckhoff.com/download/Document/automation/twincat3/TC3_EventLogger_EN.pdf

Beckhoff. (7 de Julio de 2020). *Beckhoff.com*. Obtenido de Beckhoff.com: https://infosys.beckhoff.com/english.php?content=../content/1033/tcplclibutilities/html/TcPlcLibUtilities_NT_GetTime.htm&id=

Bramlette, M. (1991). *Initialization, mutation and selection methods in genetic algorithms.*

EHU. (11 de Julio de 2020). *https://www.ehu.eus/es/.* Obtenido de https://www.ehu.eus/es/: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf

Lefebvre, L. (15 de Octubre de 2018). *pypi.org*. Obtenido de pypi.org: https://pypi.org/project/pyModbusTCP/#history

MathWorks. (7 de Julio de 2020). *es.mathworks.com*. Obtenido de es.mathworks.com: https://es.mathworks.com/discovery/genetic-algorithm.html

MathWorks. (7 de Julio de 2020). *es.mathworks.com*. Obtenido de es.mathworks.com: https://es.mathworks.com/products/optimization.html

Microsoft. (7 de Julio de 2020). *Microsoft.com*. Obtenido de Microsoft.com: https://visualstudio.microsoft.com/es/

Modbus Organization. (24 de Octubre de 2006). *Modbus.org.* Obtenido de Modbus.org: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

National Instruments. (17 de Septiembre de 2019). *ni.com*. Obtenido de ni.com: https://www.ni.com/es-es/innovations/white-papers/14/the-modbus-protocol-in-depth.html

National Instuments. (14 de Mayo de 2020). *ni.com*. Obtenido de ni.com: https://www.ni.com/es-es/innovations/white-papers/17/what-is-hardware-in-the-loop-.html

Python. (19 de Octubre de 2019). *Python.org*. Obtenido de Python.org: https://www.python.org/downloads/release/python-2717/

Python. (19 de Junio de 2020). *docs.Python.org*. Obtenido de docs.Python.org: https://docs.python.org/2/library/time.html

Python. (7 de Julio de 2020). *Python.org*. Obtenido de Python.org: https://www.python.org/

Tan, W., Márquez, H., Tongwen, C., & Jizhen, L. (18 de Marzo de 2005). *www.sciencedirect.com.* Obtenido de www.sciencedirect.com: https://www.sciencedirect.com/science/article/pii/S0959152405000326

VirtualCable. (15 de Septiembre de 2016). *VirtualCable.net*. Obtenido de VirtualCable.net: http://www.virtualcable.net/uds-enterprise-enables-the-deployment-of-server-os/

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MÁSTER EN INGENIERÍA INDUSTRIAL

DISEÑO DE UN SISTEMA DE CONTROL BASADO EN PC MEDIANTE SOFTWARE TWINCAT 3 DE UNA PLANTA DE GENERACIÓN DE ENERGÍA ELÉCTRICA DE 160MW SIMULADA MEDIANTE HIL E IMPLEMENTADA EN PYTHON

DOCUMENTO Nº2:

PRESUPUESTO

AUTOR: GUILLERMO JOAQUÍN ALITE CEREZUELA

TUTOR: JUAN MANUEL HERRERO DURÁ

Curso Académico: 2019-20

Índice documento nº2

Índice de tablas

1 INTRODUCCIÓN

El DOCUMENTO Nº2: PRESUPUESTO contiene las tablas utilizadas para el cálculo del coste del proyecto que se ha explicado en el documento anterior, contempla los costes asociados al diseño y optimización del sistema de control, así como el diseño del HIL y las pruebas de funcionamiento. Al final del documento se ha añadido una tabla que muestra los productos de la marca Beckhoff necesarios para la implementación en la planta real y su coste de adquisición. Para las mediciones se ha establecido que un año equivale a 221 días laborables con 1768h/año en jornadas de 8h.

2 CUADRO DE PRECIOS Nº1: MANO DE OBRA

Código	Descripción	Precio (€/h)
MO.1	Graduado en Ingeniería en Tecnologias Industriales	35,00

Tabla 9 Cuadro de precios nº1 mano de obra

3 CUADRO DE PRECIOS Nº2: MATERIALES Y AMORTIZACIONES

Código	Descripción	Precio (€)	Precio (€/año)	Precio €/h
Software				
MAT.S1	Python	x	0	0
MAT.S2	TwinCat 3	1200	600	0,34
MAT.S3	Matlab/Simulink	x	800	0,45
Hardware				
MAT.H1	Ordenador portátil	1055	263,75	0,15

Tabla 10 Cuadro de precios nº2: Materiales y amortizaciones

Se ha considerado una amortización de dos años para las licencias de *TwinCat 3* necesarias para la fase de diseño, y una vida útil de 4 años para el ordenador portátil utilizado para la realización del proyecto.

4 CUADRO DE PRECIOS Nº3: PRECIOS PARCIALES

Código	Unidades	Descripción	Medición	Precio	Importe
Capitulo 1. Diseño del HIL y servidor Modbus mediante Python					
UO1	Ud	UO1: Diseño del HIL para la simulación de la planta y diseño del sevidor para comunicación por Modbus implementado en Python.	1	2.151,18 €	2.151,18 €
		Total Capítulo 1			**2.151,18 €**
Capitulo 2. Diseño del sistema de control mediante TwinCat 3					
UO2.1	Ud	UO2.1: Diseño del sistema de control a tiempor real en TwinCat.	1	4.343,90 €	4.343,90 €
UO2.2	Ud	UO2.2: Optimización de PIDs mediante algoritmos genéticos	1	2.904,96 €	2.904,96 €
		Total Capítulo 2			**7.248,86 €**
Capitulo 3. Redacción de documentos formales					
UO3	Ud	UO3: Tratamiento de la información obtenida a lo largo del trabajo y redacción formal de documentos a presentar.	1	2.151,18 €	2.151,18 €
		Total Capítulo 3			**2.151,18 €**
Capitulo 4. Pruebas de funcionamiento					
UO4	Ud	UO4: Corrección de errores, test de funcionamiento y grabación del video de prueba.	1	1.434,12 €	1.434,12 €
		Total Capítulo 3			**1.434,12 €**

Tabla 11 Tabla de precios parciales

5 CUADRO DE PRECIOS Nº4: PRECIOS DESCOMPUESTOS

Capitulo 1. Diseño del HIL y servidor Modbus mediante Python					
UO1: Diseño del HIL para la simulación de la planta y diseño del sevidor para comunicación por Modbus implementado en Python.					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio**	**Importe**
MO.1	h	Graduado en Ingeniería en Tecnologias Industriales	60,00	35,00	2100,00
MAT.S1	h	Python	60,00	0,00	0,00
MAT.H1	h	Ordenador portátil	60,00	0,15	9,00
	%	Costes directos complementarios	2,00		42,18
		Coste total de la UO1			**2.151,18 €**

Tabla 12 Precios descompuestos. Capítulo 1

Capitulo 2. Diseño del sistema de control mediante TwinCat 3					
UO2.1: Diseño del sistema de control a tiempor real en TwinCat y HMI para control de usuario.					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio**	**Importe**
MO.1	h	Graduado en Ingeniería en Tecnologias Industriales	120,00	35,00	4200,00
MAT.S2	h	TwinCat 3	120,00	0,34	40,72
MAT.H1	h	Ordenador portátil	120,00	0,15	18,00
	%	Costes directos complementarios	2,00		85,17
		Coste total de la UO2.1			**4.343,90 €**

Tabla 13 Precios descompuestos. Capítulo 2.1

UO2.2: Optimización de PIDs mediante algoritmos genéticos					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio**	**Importe**
MO.1	h	Graduado en Ingeniería en Tecnologias Industriales	80,00	35,00	2800,00
MAT.S3	h	Matlab/Simulink	80,00	0,45	36,00
MAT.H1	h	Ordenador portátil	80,00	0,15	12,00
	%	Costes directos complementarios	2,00		56,96
		Coste total de la UO2.2			**2.904,96 €**
		Coste total de la UO2			**7.248,86 €**

Tabla 14 Precios descompuestos. Capítulo 2.2 y coste total Capítulo 2

Capitulo 3. Redacción de documentos formales					
UO3: Tratamiento de la información obtenida a lo largo del trabajo y redacción formal de documentos a presentar.					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio**	**Importe**
MO.1	h	Graduado en Ingeniería en Tecnologias Industriales	60,00	35,00	2100,00
MAT.H1	h	Ordenador portátil	60,00	0,15	9,00
	%	Costes directos complementarios	2,00		42,18
		Coste total de la UO3			**2.151,18 €**

Tabla 15 Precios descompuestos. Capítulo 3

Capitulo 4. Pruebas de funcionamiento					
UO4: Corrección de errores, test de funcionamiento y grabación del video de prueba.					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio**	**Importe**
MO.1	h	Graduado en Ingeniería en Tecnologias Industriales	40,00	35,00	1400,00
MAT.H1	h	Ordenador portátil	40,00	0,15	6,00
	%	Costes directos complementarios	2,00		28,12
		Coste total de la UO4			**1.434,12 €**

Tabla 16 Precios descompuestos. Capítulo 4

6 PRESUPUESTO FINAL

Capítulo 1.....2151.18€

Capítulo 2.....7248.86€

Capítulo 3.....2151.18€

Capítulo 4.....1434.12€

PRESUPUESTO TOTAL DE EJECUCIÓN MATERIAL.....12985.34€

Gastos generales (12%) .....1558.24€

Beneficio industrial (6%) .....779.12€

PRESUPUESTO TOTAL DE INVERSIÓN.....15322.7€

IVA (21%) .....3217.76€

PRESUPUESTO BASE LICITACIÓN.....18540.46€

7 PLANTA REAL

En este presupuesto contempla todo el trabajo relacionado con el diseño y validación del sistema de control mediante simulación de la planta de generación. En el momento de la implantación en la industria, harían falta un PC industrial, módulos de E/S y pagar las licencias de uso de TwinCat 3. Por lo que en este apartado se ha incluido el coste de todos estos elementos y el modelo seleccionado.

Coste del material adicional a adquirir por el cliente					
Costes relacionados con la adquisiciñon de software y hardware necesarios para la implementación en la planta real					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio**	**Importe**
MAT.S1_C	Ud	Licencias de Twincat 3	1,00	1200,00	1200,00
MAT.H1_C	Ud	PC de control (CX1020-0000)	1,00	900,00	900,00
MAT.H2_C	Ud	Terminal analógico de entrada (EL3164)	1,00	100,00	100,00
MAT.H3_C	Ud	Terminal analógico de salida (EL4104)	1,00	100,00	100,00
MAT.H4_C	Ud	Concentrador de EtherCat (EK1100)	1,00	150,00	150,00
	%	Costes directos complementarios	2,00		49,00
Coste total			**2.499,00 €**		

Tabla 17 Cuadro de precios de software y hardware para implementación real