Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Plataforma de práctica para PLC simulado

Autor: Álvaro Gómez Díaz

Tutor: David Muñoz de la Peña Sequedo

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

Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Plataforma de práctica para PLC simulado

Autor:
Álvaro Gómez Díaz

Tutor:
David Muñoz de la Peña Sequedo
Catedrático

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

Agradecimientos

Quiero agradecer este proyecto a todas las personas que estuvieron ayudándome y guiándome durante mi vida y que estuvieron ahí en los momentos más necesarios. Sin ellos esto no sería posible.

Álvaro Gómez Díaz

Sevilla, 2020

Resumen

Este trabajo tiene como objetivo facilitar a los estudiantes practicar con la programación de PLCs, el control de sistemas continuos simulados en Matlab y diseño de SCADAS mediante la interconexión de diferentes plataformas (Vijeo, Matlab, SoMachine...) con licencia disponible o gratuita y protocolos de comunicación abiertos como OPC y Modbus, haciendo así que se puedan dar prácticas de forma no presencial o complementar alguna práctica presencial ya existente. En el proyecto se ha intentado maximizar la flexibilidad y facilidad a la hora de implementar las plataformas en los ordenadores de los estudiantes, de forma que no se necesite comprar ningún tipo de hardware o software.

Abstract

This work aims to facilitate students to practice with the programming of PLCs, the control of continuous systems simulated in Matlab and SCADA design through the interconnection of different platforms (Vijeo, Matlab, SoMachine ...) with available or free license and open communication protocols such as OPC and Modbus, thus enabling non-face-to-face practices to be given or to complement an existing face-to-face practice. The project has tried to maximize flexibility and ease when it comes to implementing the platforms on the students' computers, so that it is not necessary to buy any type of hardware or software.

Índice Abreviado

Índice

1 Planteamiento

Este proyecto ha sido planteado con el fin de ayudar a la realización de prácticas con PLCs, haciendo uso de un PLC simulado en un ordenador personal, así evitando tener que hacer uso de un dispositivo físico de estas características, por lo que el estudiante o usuario que quiera practicar y familiarizarse con estos dispositivos podrá hacerlo de forma remota. Esto será realmente positivo en situaciones como las que se han vivido durante la cuarentena a causa de la pandemia por el Covid-19, donde al estudiante le resultaba imposible acudir a su respectiva universidad a realizar dichas prácticas. A pesar de que se ha comprobado que esta herramienta se puede utilizar, es recomendable que se siga optando por hacer las practicas presenciales si es posible, ya que el proceso de prueba de forma remota es más lento y pueden surgir otros problemas que sean ajenos a la práctica con los PLCs.

1.1 Objetivos

Nuestro objetivo principal será crear una plataforma para prácticas de control de sistemas continuos con Scada usando un PLC gratuito y comunicación con MATLAB® por OPC. Como hemos comentado antes, esto servirá para que el estudiante pueda aprender sobre el funcionamiento de los PLCs y familiarizarse con los lenguajes que se usan en éstos.

Para conseguir esto tendremos que realizar una serie de tareas. La primera será representar un sistema a controlar por el PLC, este tendrá que tener una variable de entrada y otra de salida. Para representar este sistema utilizaremos un archivo de Simulink MATLAB® . La siguiente tarea será crear un controlador mediante un programa de Schneider, empresa fabricante de PLCs. Este controlador intercambiará información con el sistema de MATLAB® enviándole la acción de control necesaria y recibiendo tanto el set-point como la respuesta del sistema a dicha acción de control. También habrá que, como bien se ha comentado antes, comunicar el controlador con el sistema a controlar y sincronizarlos de forma que tengan el mismo tiempo de simulación. Por último, habrá que visualizar como se desarrolla el sistema durante la simulación, para ello se creará una interfaz HMI (Scada) para que el usuario pueda interactuar con el sistema y observar su desarrollo.

2 Pruebas realizadas

Antes de dar con los programas idóneos y con la arquitectura necesaria para lograr los objetivos planteados, se han realizado un conjunto de pruebas hasta encontrar el modelo y los programas más convenientes para nuestra plataforma. Dividiremos estas pruebas en lo referente al apartado de control del sistema, a la comunicación de ambos programas, la sincronización de éstos y la visualización del mismo.

2.1 Control

Para el control, se empezó probando el programa de Schneider llamado SoMachine 4.3. Una vez se descargue el programa tendremos unos 21 días de prueba, aunque se podrá usar de forma ilimitada una vez nos registremos en la página web de Schneider [5], que se puede descargar en el enlace que se encuentra más abajo, es posible simular para un amplio número de controladores Logic como por ejemplo los dispositivos M221, M238, M241, M251, M258 o HMI, Drive o Motion. También dispone de varios tipos de lenguajes para usarlos en los POUs (bloques programables), entre ellos lenguajes como LD, ST, SFC, FBD, CFC u otras menos usados a la hora del aprendizaje con los PLCs como IL. Aparte de los POUs, también se pueden crear acciones y transiciones para éstos con cualquiera de los lenguajes que se han comentado antes. Adicionalmente, cuenta con variables globales y locales para todos los POUs y bastantes herramientas para ayudar a la hora de realizar el control como pueden ser alarmas, operadores o fórmulas.
https://schneider-electric.box.com/s/35g7f6zz5mr8spyka0695hge7j2l3qdy

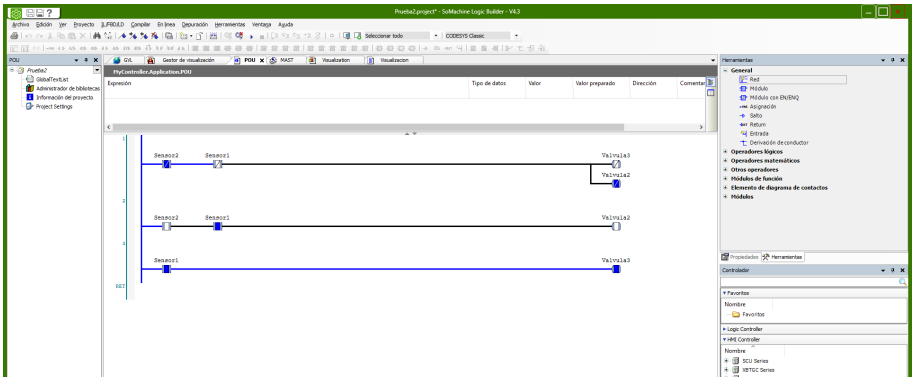


Figura 2.1 SoMachine 4.3.

Para poder simular con este programa sin usar un PLC físico, es decir, simulando en nuestro propio ordenador, nos hará falta pasar a modo simulación. Una vez hecho esto compilaremos y simularemos nuestro programa sin ningún tipo de problema.

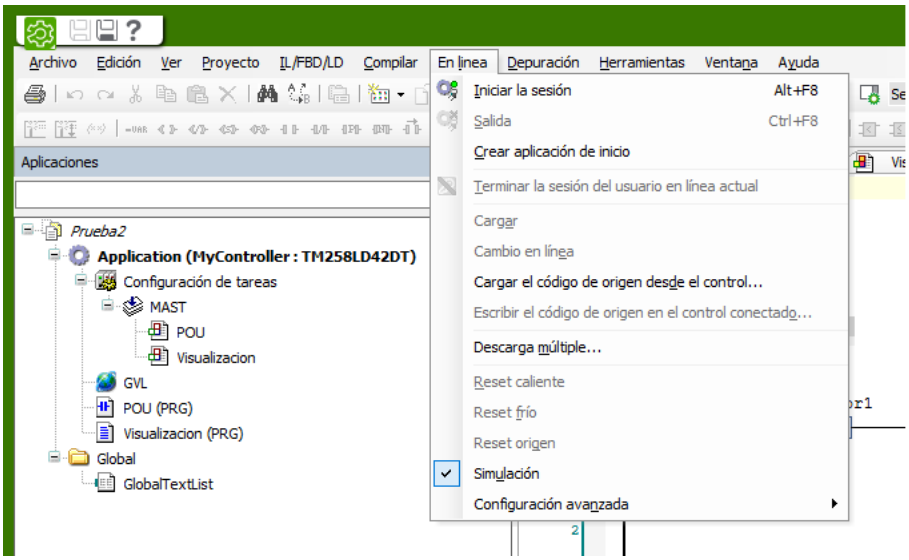


Figura 2.2 Simulación sin PLC SoMachine 4.3.

Fijándonos en el manual, vemos como se puede conectar mediante OPC con programas como CodeSys. Sin embargo, no nos es posible comunicarnos con Vijeo debido a que nos hará falta la licencia del mismo. También no nos es posible comunicarnos con MATLAB® mediante Modbus, como se puede observar en el libro [4]. Es por esto último que se decide utilizar otro programa.

Después de este programa, se probó a usar otro programa de Schneider llamado EcoStruxure Machine Basic, el cual sí era posible comunicarlo con MATLAB® (se hablará en el siguiente apartado). Usamos la versión gratuita que se consigue una vez te registras y tiene un tiempo ilimitado. Esta versión gratuita tiene algunas limitaciones como se puede ver en la siguiente página [2] entre ellas algunos comandos no disponibles en la depuración o la imposibilidad de cambiar el programa estando en línea. Dentro de este programa se pueden usar diferentes tipos de lenguajes, sin embargo es más limitado que SoMachine 4.3, ya que solo se pueden usar los lenguajes SFC, LD e IL, aunque con posibilidad de usar bloques de ST en LD.

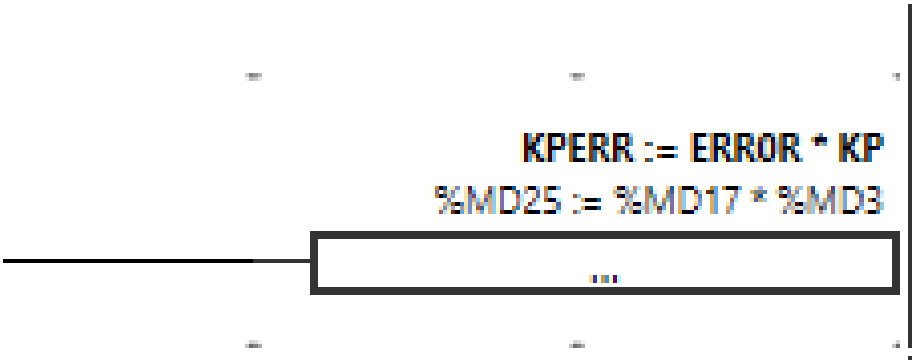


Figura 2.3 Ejemplo de bloque de ST en LD.

En el programa de EcoStruxure se pueden usar diferentes objetos entre ellos variables de software simples, como bits de memoria y palabras, direcciones de las entradas y salidas digitales o analógicas, variables internas del controlador, como palabras y bits del sistema, funciones predefinidas del sistema o bloques de funciones, como temporizadores y contadores. Aparte de la tarea maestra también se pueden crear tareas periódicas las cuales podrán tener un periodo de entre 1 a 255 ms. Por la parte de los controladores a usar, solo tendremos la posibilidad de simular con controladores M221. Toda esta información queda recogida en la siguiente manual colgado en la web por Schneider [3]

Al principio se intentó usar para controlar una de estas funciones predefinidas que creaba PID directamente, sin embargo, esta opción solo era posible usarla si se tiene consigo un PLC de forma física.

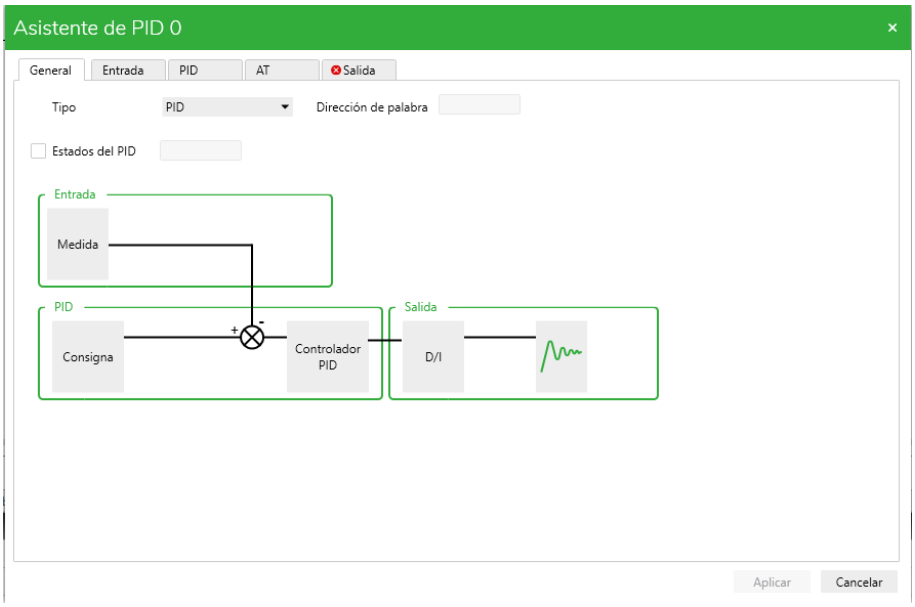


Figura 2.4 Herramienta PID.

Al no poder usar esta herramienta, fuimos a crear un controlador en LD utilizando bloques de ST. Para probar su funcionamiento, primero creamos un control P para controlar un sistema con un tiempo de muestreo de 0.2s (este tiempo de muestreo indicará cada cuanto se envía la acción de control), para esto recogíamos tanto la variable a controlar como la referencia y restábamos una con la otra. A continuación, guardábamos en una variable el resultado anterior y en otra el valor de nuestra constante Kp (en este caso de valor 7), para luego multiplicarlas entre ellas y obtener así la acción de control. Aquí se puede ver el código del programa.

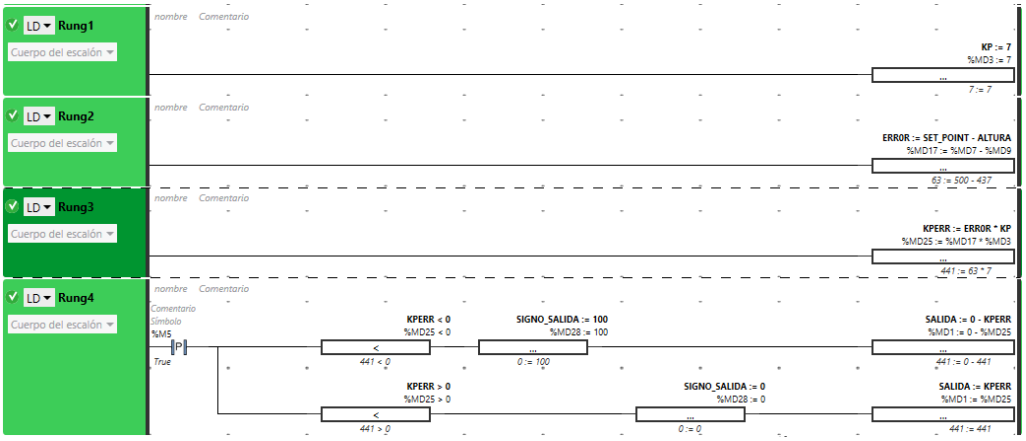


Figura 2.5 Código control Kp.

Con este controlador intentamos controlar un sistema simple: $\frac{1}{s+1}$. Este sistema se simuló en Simulink (más tarde se verá como se comunican ambos programas) y quedó así.

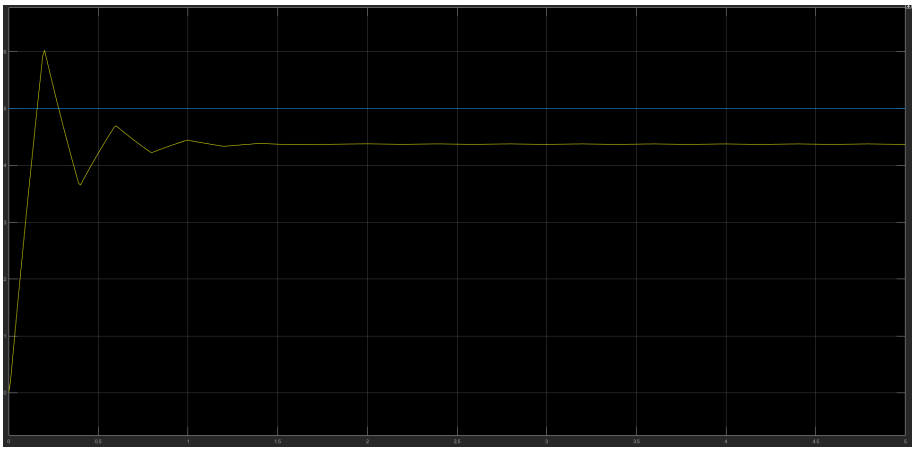


Figura 2.6 Control P.

Si se utiliza un controlador PID directamente en simulink usando el bloque de PID discreto con los mismos valores mencionados anteriormente, se puede observar bastantes similitudes entre ambas gráficas.

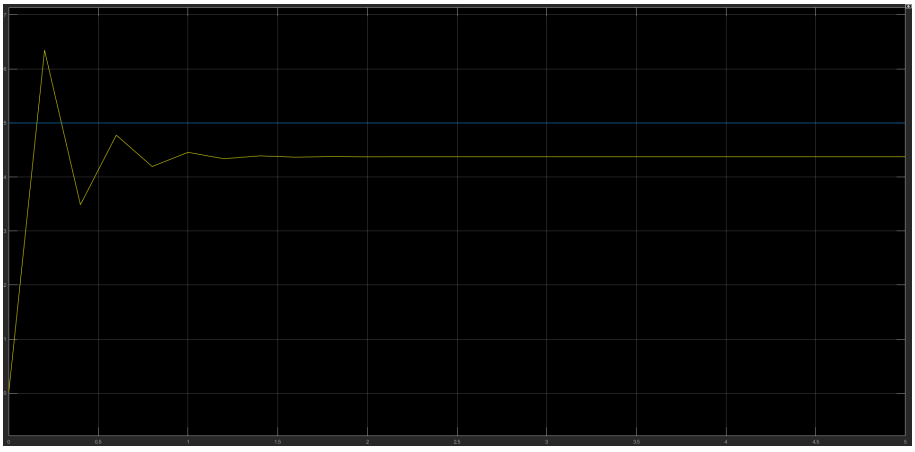


Figura 2.7 Control P usando bloque PID de Simulink.

Por lo tanto, para arreglar este error en régimen permanente decidimos utilizar un control por PI. Como consecuencia de esto, se tuvieron que añadir más líneas de código a nuestro POU. Primero, se creó una nueva variable llamada Ki (la cual tendrá un valor de 12) y otra llamada error acumulado. Este error acumulado sería igual al error acumulado en el ciclo anterior más el error actual multiplicado por el tiempo de muestreo. Una vez hecho esto, cambiaremos el valor de la acción de control igualándola al error multiplicado por Kp más el error acumulado por Ki. Aquí el pseudocódigo del programa y una imagen de como sería la gráfica de la simulación.

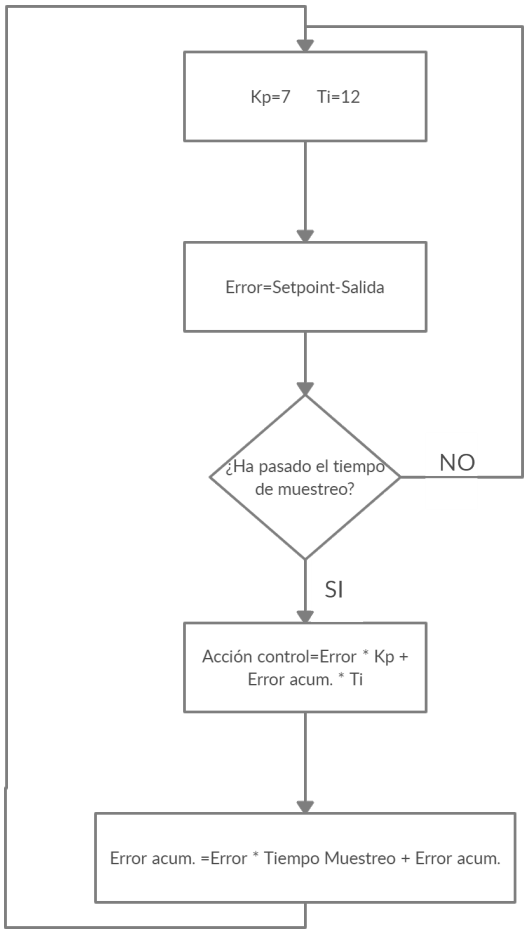


Figura 2.8 Control PI.

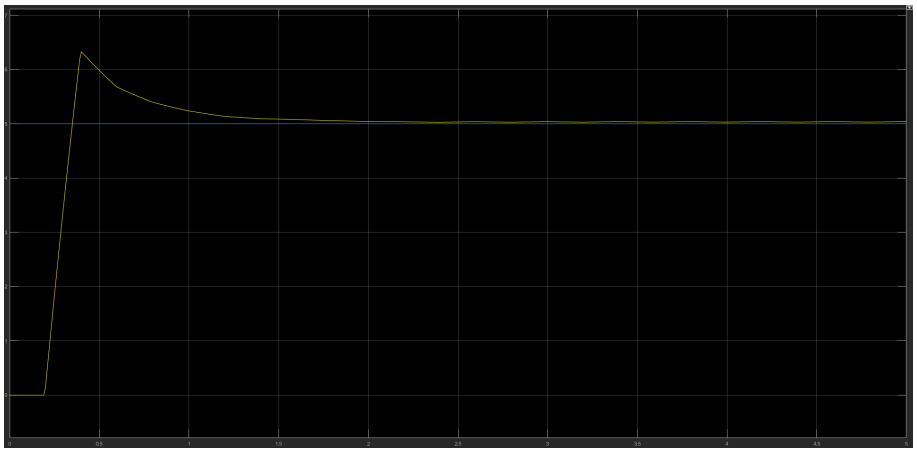


Figura 2.9 Diagrama control PI.

Ahora lo volvemos a comparar con el control usando el bloque PID de Simulink y con los mismos valores en las constantes.

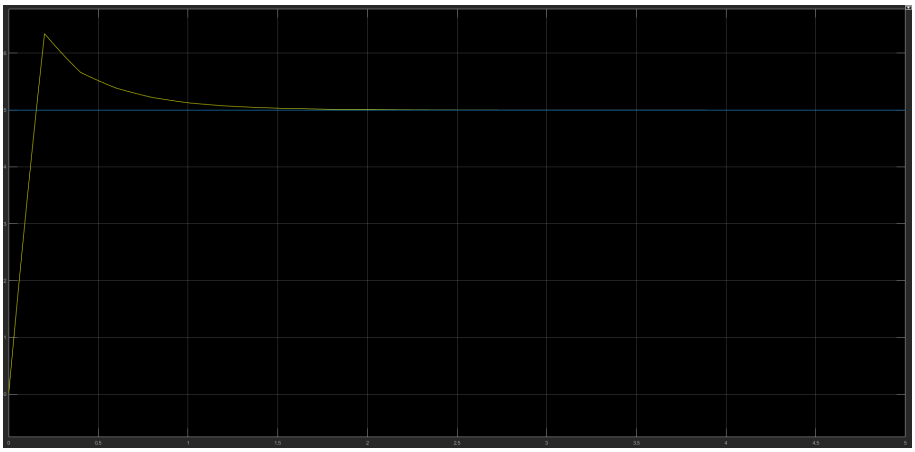


Figura 2.10 Control PI usando bloque PID de Simulink.

Como se puede apreciar, la respuesta del sistema sobreoscila demasiado al principio. Para solucionar este problema tendremos dos opciones. Una de ellas es limitar el efecto de la integral, esta herramienta se le llama compensación anti windup. Sobre todo, este tipo de compensación es usada en sistemas reales donde la salida puede llegar a saturarse y, al no poder aumentar más la salida, el error acumulado no pare de incrementarse. Para hacer esto, limitaremos los valores máximos y mínimos del error acumulado (-500, 500). Consecuentemente, la gráfica nos quedará así.

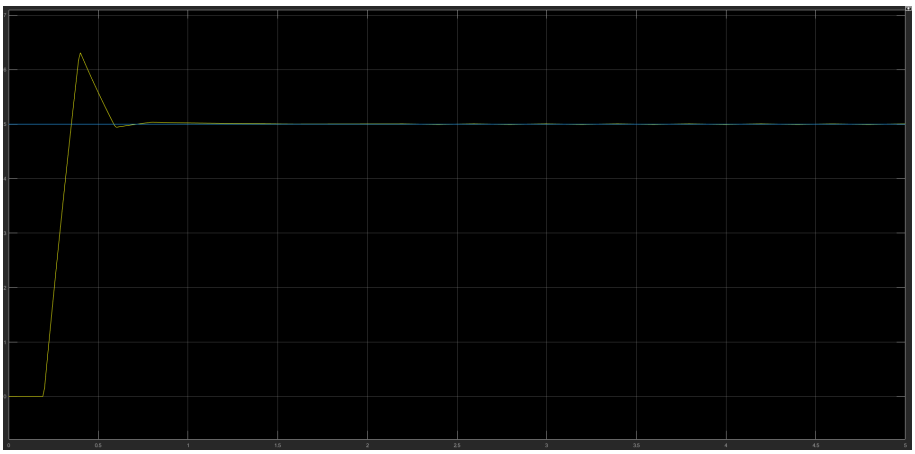


Figura 2.11 Control PI con anti wind up.

La otra forma es variar ese tiempo de muestreo, que como habíamos dicho antes era el tiempo que nos indicará cada cuanto cambia la acción de control que se envía al sistema. En el anterior caso era de 0.2s pero si se baja ese tiempo a 0.1s se puede ver una mejora en el control.

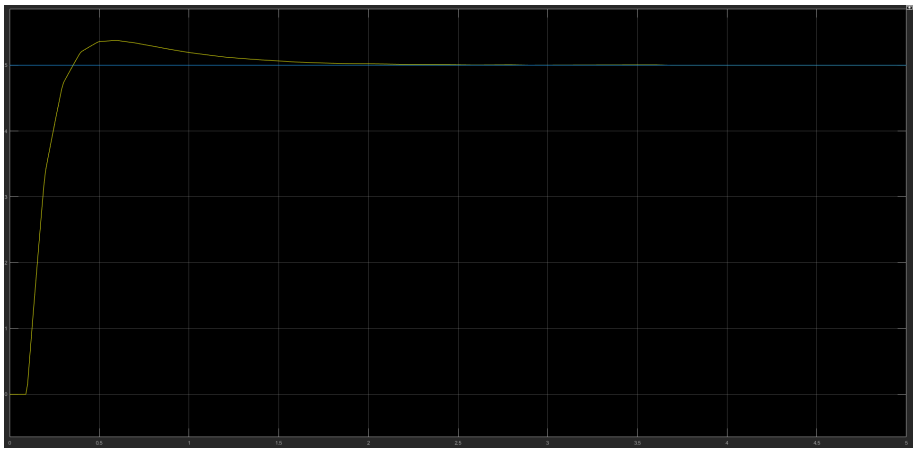


Figura 2.12 Control PI con tiempo de muestreo menor.

Volvemos a mostrar como sería usando el bloque PID de Simulink con el mismo tiempo de muestreo y datos.

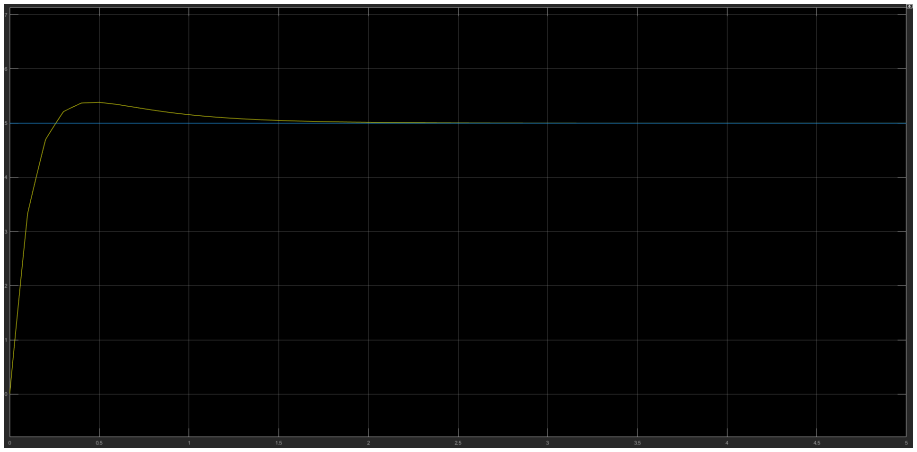


Figura 2.13 Control PI con tiempo de muestreo menor usando bloque PID de Simulink.

Por último, juntando ambos conceptos, el sistema se controla mucho mejor, aquí la gráfica.

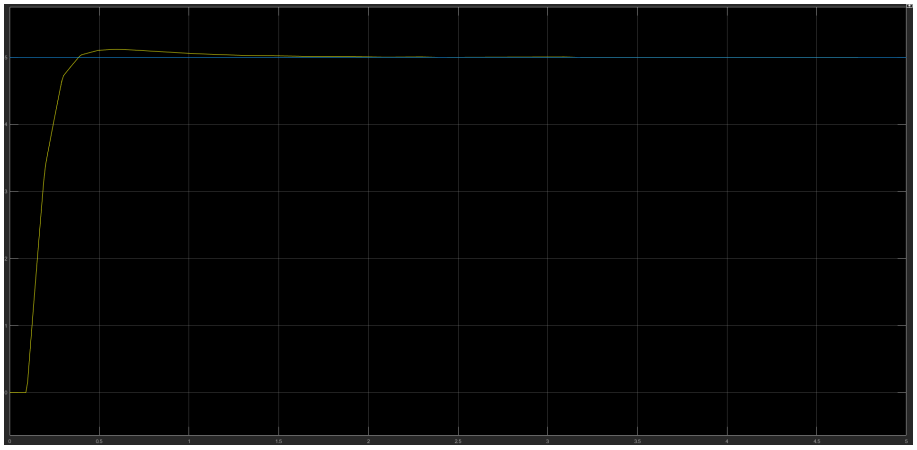


Figura 2.14 Control PI con tiempo de muestreo menor y anti windup.

Otro dato muy importante acerca del programa de control EcoStruxure, es que cada variable que se vaya a

usar tiene que estar separada de otras, ya que si está se vuelve negativa o excede sus limites puede provocar cambios en otras que se encuentren cercanas a ella. Por ejemplo, como se puede observar en la siguiente imagen, al cambiar el valor de %MD5 y ponerlo negativo se cambia el valor de %MD6

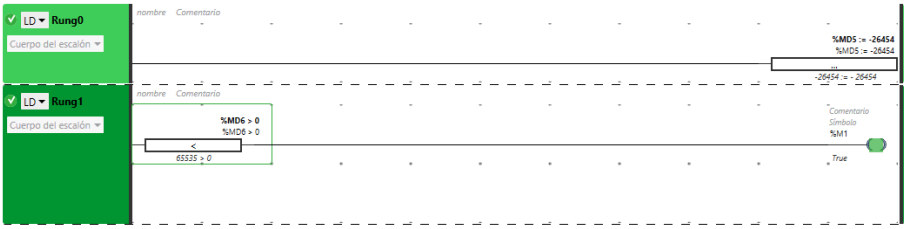


Figura 2.15 Solapamiento de valores entre variables adyacentes en la memoria.

2.2 Comunicación

Al principio del trabajo se comentó que ambos programas se iban a comunicar mediante OPC. Sin embargo, en la bibliografía que teníamos a nuestra disposición no había información al respecto de como conectar ambos programas mediante este método. Es por esto y porque se revisó información de otras pruebas anteriores, como las que se pueden ver en los siguientes libros [6] [1], que se decidió escoger como método de comunicación Modbus TCP/IP usando para esto funciones ya incluidas en el mismo Matlab. Este método es fácil de usar y no hace falta instalación aparte alguna lo cual es idóneo para nuestro proyecto, ya que éste va orientado a estudiantes y cuanto más fácil sea la preparación previa a la practica más tiempo se podrá dedicar a la misma.

Para utilizar este tipo de comunicación hemos usado las siguientes tres funciones:

1. Función **modbus**: sirve para crear un objeto MODBUS que se usará para comunicar ambos programas. Esta función recibirá los siguientes argumentos:

   - Transport: Este argumento habrá que rellenarlo con el tipo de transporte que en este caso será "tcpip"
   - DeviceAddress: Habrá que añadir la dirección IP con la que se quiere comunicar, en este caso al simular desde nuestro propio ordenador utilizaremos la dirección 127.0.0.1.
   - Port: Este argumento indicará el puerto remoto con el que se comunicará, este argumento es opcional ya que por defecto se añade el puerto 502, que es el puerto que está reservado para comunicación MODBUS.

2. Función **read**: esta función se usará para leer un valor usando el objeto MODBUS antes creado con la función **modbus**. Los argumentos que utiliza esta función son los siguientes:
   - M: Hace referencia al objeto MODBUS que ya fue creado anteriormente.
   - Target: Es el tipo de dato que vamos a leer, en este caso se pondrá en este argumento 'holdingregs' que hace referencia a los registros de retención de EcoStruxure.
   - Address: El tercer argumento es la dirección del valor por el que se empieza a leer, por ejemplo si se quiere leer el dato MW7 se deberá poner 7 en este apartado.
   - Count: El cuarto argumento es el número de datos a leer, si se añade un 8 a este argumento se leerán los 8 valores empezando por el primer valor dado por el argumento address.
   - Precision: El quinto argumento es el formato del dato que esta siendo leído. Este puede ser uint32, double, int16...

3. Función **write**: Esta función tiene como objetivo escribir un dato usando el objeto MODBUS antes creado. Los argumentos que utiliza son los siguientes:
   - M: Tiene la misma función que en read, sirve para añadir el objeto MODBUS con el que vamos a trabajar.
   - Target: Tiene la misma función que en read, sirve para indicar el tipo de dato que se va a escribir.
   - Address: El tercer argumento hace referencia a la dirección donde se va a escribir (al igual que en la funcion read)
   - Values: El cuarto argumento es la variable o constante que se va mandar como dato
   - Precision: El quinto argumento indica el formato del dato que esta siendo enviado. Puedes ser uint32, int, double...

A pesar de esto, se encuentran ciertos inconvenientes a la hora de la comunicación por medio de Modbus:

1. Su memoria es limitada, ya que como bien se ha dicho es un registro de 16-bits por lo que como mucho se podrá enviar datos de valor máximo de 65535 (2^{16}).
2. No es posible enviar datos decimales, solo se podrán enviar y recibir datos enteros.
3. Tampoco es posible enviar datos con valores negativos.

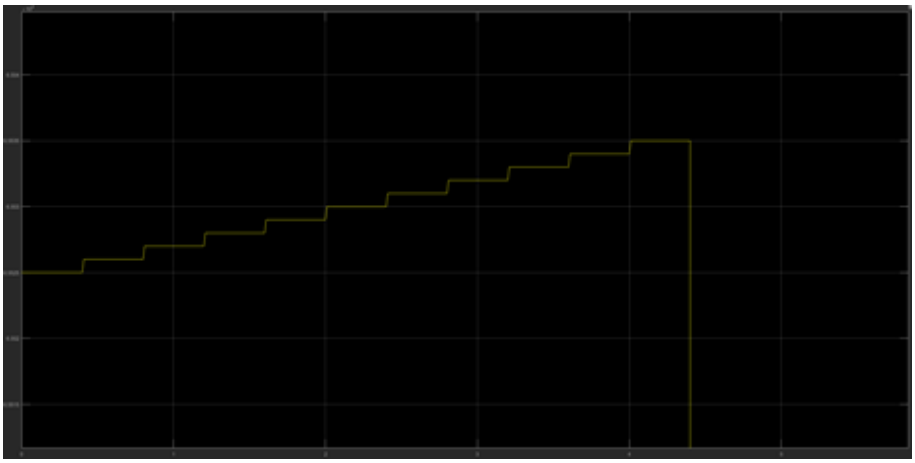


Figura 2.16 Demostración de limite en 2^{16}.

En realidad, a la hora de las pruebas de control, el problema que más nos afectará será que no se puedan enviar datos negativos. A pesar de esto, hemos conseguido evitar este problema cambiando el signo de la acción de control cuando este sea negativo y pasando ese signo como una variable independiente. Esta variable se pasará al programa de Simulink, demostrando que el valor de la acción de control es positivo si esta variable tiene el valor de 100 y negativo si tiene el valor de 1. Aquí se puede ver como programamos ambos programas para solucionar este problema como hemos indicado anteriormente.

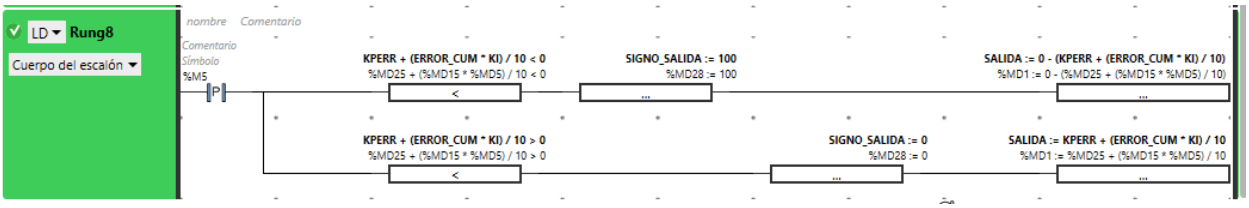


Figura 2.17 Cambio de signo en EcoStruxure.

```
if (signo==100)
    K=-MVi/100;
else
K=MVi/100;
end

end
```

Figura 2.18 Cambio de signo en MATLAB® .

Adicionalmente, como se puede ver en las imágenes antes mencionadas, multiplicamos por 100 los valores que les pasamos al controlador y los redondeamos. De esta forma, nos quitamos el problema de no poder usar decimales. Una vez recibida la acción de control, la dividiremos entre 100 para volver así a sus verdaderas dimensiones.

2.3 Sincronización

Esta es una parte importante del proyecto ya que es necesario que tanto el controlador como la plataforma donde se simule el sistema (en este caso MATLAB®) sigan los mismos tiempos de simulación, ya sea este tiempo el tiempo real o el tiempo local de uno de los dos. De no ser así, nos encontraremos problemas a la hora de controlar nuestro sistema. Estas pruebas que vienen a continuación se hicieron con los programas EcoStruxure y MATLAB® . Para la comunicación se usó un programa de MATLAB® implementado en un bloque dentro de un archivo de Simulink. En este programa se mandaba una señal con forma de escalón en el segundo 1 de la simulación y luego, 0.1s después, se recogía la misma señal proveniente del programa de EcoStruxure. Según lo que en teoría tendría que pasar es que se devolviera ese escalón 0.1 s después de haberlo enviado. Sin embargo, como se puede ver en la imagen esto no fue así.

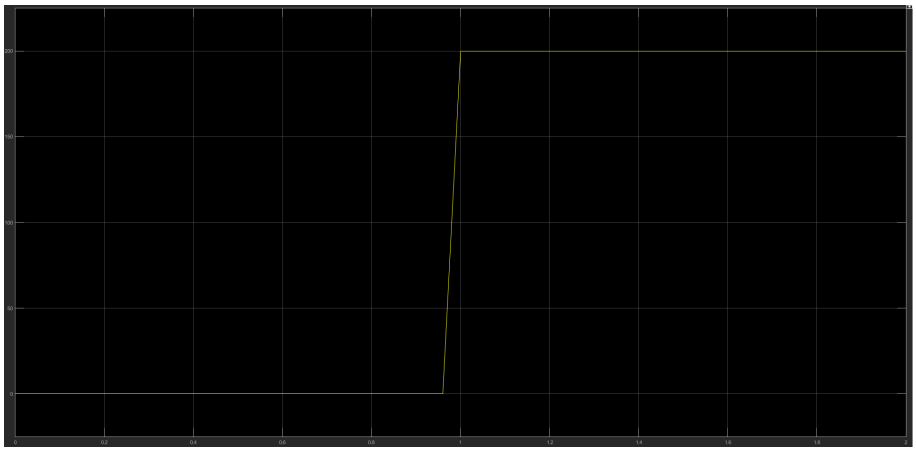


Figura 2.19 Escalón devuelto por EcoStruxure.

Esto es debido a que, aunque cada vez que el bloque con el programa manda y recibe datos cada 0.1s, dicho bloque se ejecuta un número de veces que depende del tiempo total de simulación y del step size que tenga Simulink. Aquí en la siguiente imagen se puede ver cuantas veces se ejecuta dicho bloque mediante una variable contador y se demuestra como ambos elementos condicionan cuantas veces se ejecuta dicho bloque.

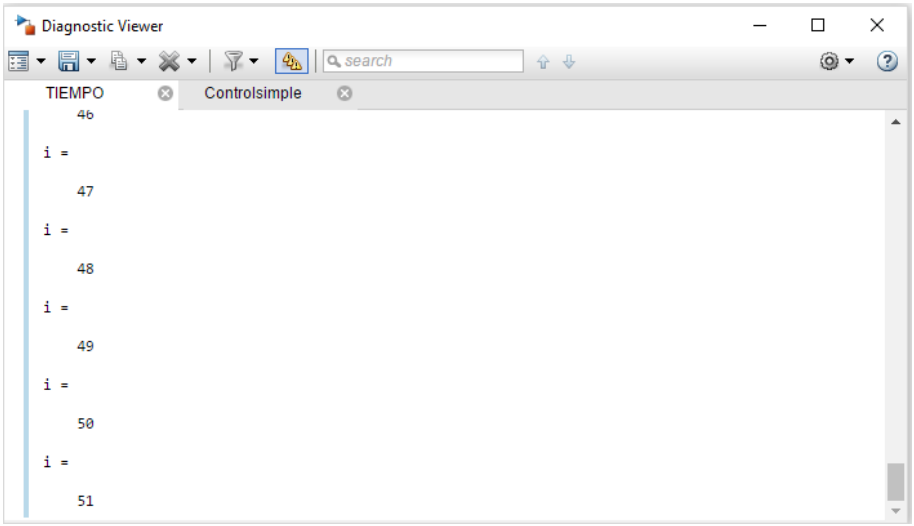


Figura 2.20 Contador de ciclos en Simulink.

Seguidamente, se probó usando el bloque de Simulink llamado Zero Order Hold. Este bloque obliga a una señal a poder cambiar solo cada cierto tiempo, es decir, bloquea el valor de una señal durante cierto tiempo especificado por el propio bloque. Esto nos ayudaría a mandar la variable a controlar cada cierto tiempo al controlador y también a retocar indirectamente el step-size del archivo de Simulink.

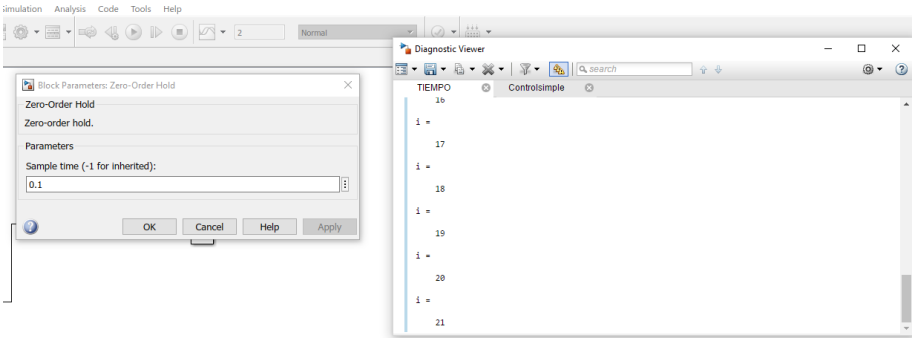


Figura 2.21 Relación entre Zero-order Hold y número de ciclos.

Después de esto, se probó a controlar un sistema simple $\frac{1}{s+1}$ cambiando la acción de control del controlador cada cierto tiempo mientras que se mandaba la variable a controlar de forma ininterrumpida (tantas veces como se ejecutaba el bloque del programa de comunicación). Para esto se usó cierta variable del sistema que se activaba cada 1s, para comprobar su correcto funcionamiento se usó un cronómetro mientras se guardaba en una variable cuantas veces se activaba dicha variable del sistema durante 3 minutos. Una vez comprobado que la variable del sistema funcionaba correctamente, se usó para controlar el sistema y se mandó la acción de control cada 1s. En la imagen se puede ver el resultado de esta acción de control.

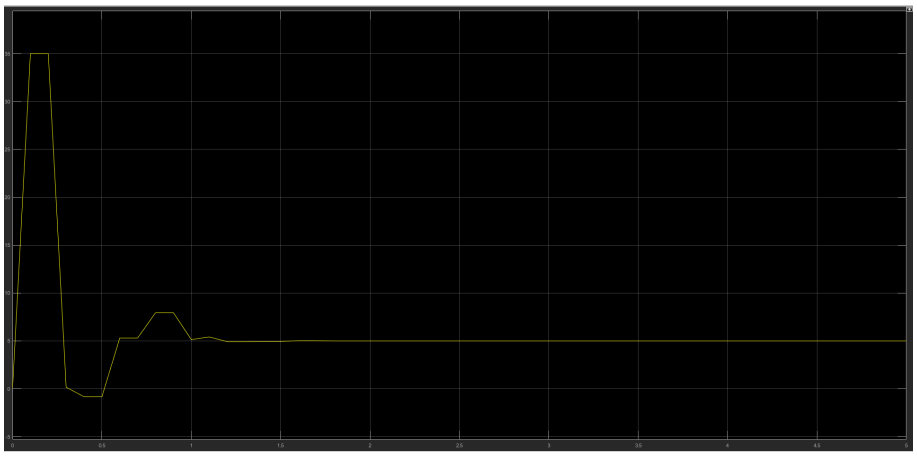


Figura 2.22 Acción de control en Simulink con tiempo de muestreo de 1 segundo en tiempo real.

Esto es debido a que el tiempo de simulación en simulink no es el mismo que el tiempo real, y los pasos que da durante la simulación, es decir, el step size no es constante, por lo que 1s en tiempo real, para Simulink puede ser 600 ms o incluso 200 ms. Por último, se decidió optar por utilizar el tiempo local de simulación de Simulink, ya que de esta forma, aunque no se simulase en tiempo real, se pudiera controlar el sistema. Por lo tanto, se hizo que el programa de comunicación de Matlab, aparte de enviar la variable a controlar y el setpoint, también enviase el tiempo local de simulink. Aquí se pueden ver los resultados.

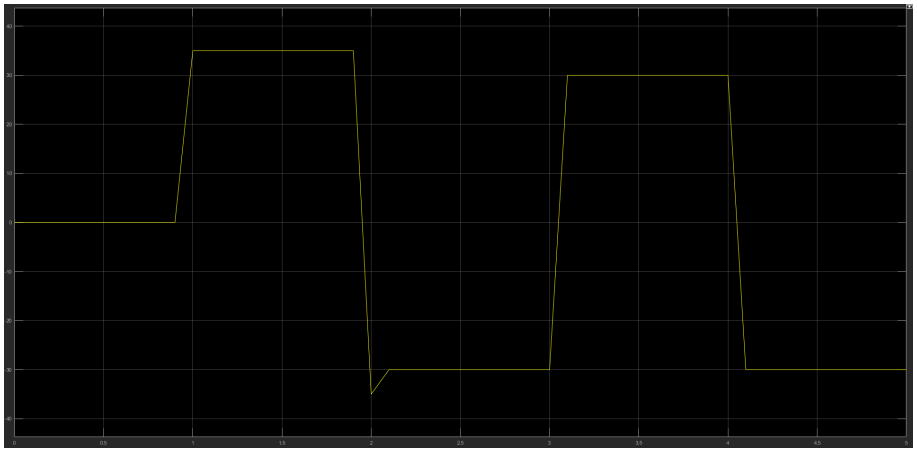


Figura 2.23 Acción de control en Simulink con tiempo de muestreo de 1 segundo con tiempo local del mismo programa.

Se puede observar que ahora cambia al pasar ese segundo que le hemos dicho pero que tarda bastante en hacer ese cambio y esto es a causa del step size del programa de simulink. Si el step size es muy grande por ejemplo 0.1, y se manda la acción de control cada 1s ésta tardará en reflejarse en simulink hasta el siguiente ciclo después de mandársela. Por lo que lo único que nos queda es cambiar el step size disminuyéndolo lo suficiente para poder controlar mejor así el sistema. Aquí se muestra otra prueba que se hizo en la cual se enviaba escalones cada 0.1 s teniendo un step size de 0.01.

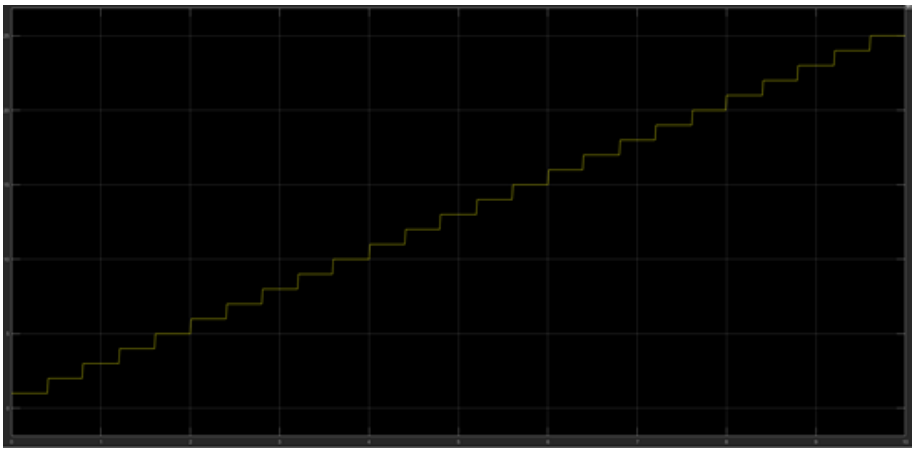


Figura 2.24 Escalones de 0.1s con step-size de 0.01.

Una vez visto esto, se observa que se envían los datos tal y como se quería. Por lo tanto, se pasa a implementar dicho método de sincronización en nuestro sistema, pero en vez de cambiar el step size usando el bloque Zero-order Hold. A continuación se ve tanto la acción de control como la respuesta del sistema.

Figura 2.25 Acción de control final.

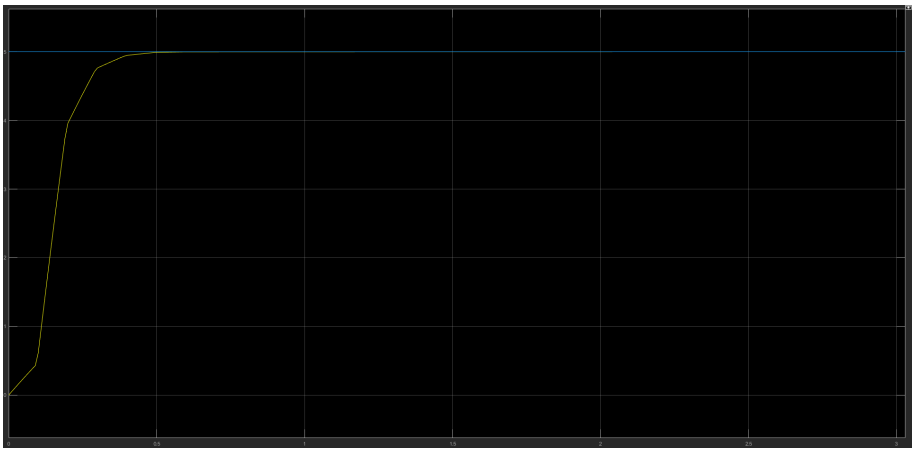


Figura 2.26 Respuesta del sistema final.

2.4 HMI

La primera visualización que se probará será la que viene incluida en el mismo programa de SoMachine 4.3. Como ya se puede uno imaginar, esta visualización no tiene tantas herramientas como Vijeo Designer, pero podría servirnos para una práctica o para simplemente familiarizarnos con este tipo de programas de visualización. Aquí una imagen de como se vería la visualización de un sistema de control por relé de un tanque de agua.

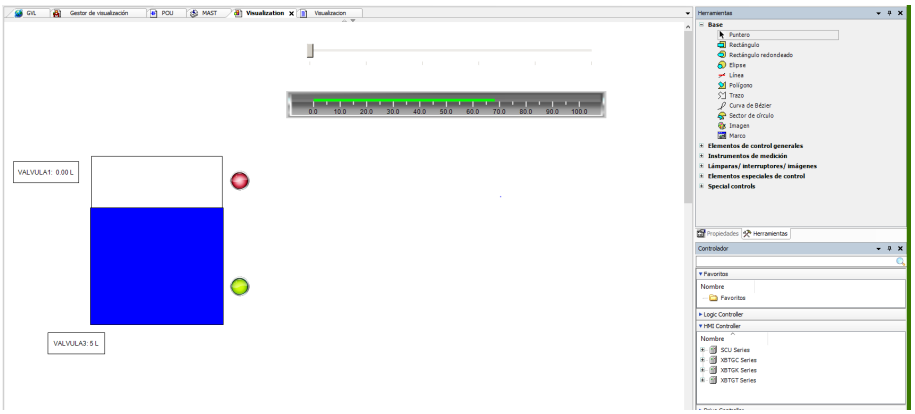


Figura 2.27 Visualización en SoMachine 4.3.

A pesar de que encajaría con lo que estamos buscando, al no poder usar el programa de SoMachine 4.3 debido a que no lo podemos comunicar con MATLAB® , nos imposibilita también el uso de la visualización en el mismo programa. La siguiente opción para crear nuestra interfaz HMI fue usando el programa Vijeo Designer. Este programa era muy versátil y tenía muchas herramientas para la visualización ya que el programa estaba orientado a hacer eso exclusivamente. Se hicieron algunas pruebas representando diferentes sistemas a controlar.

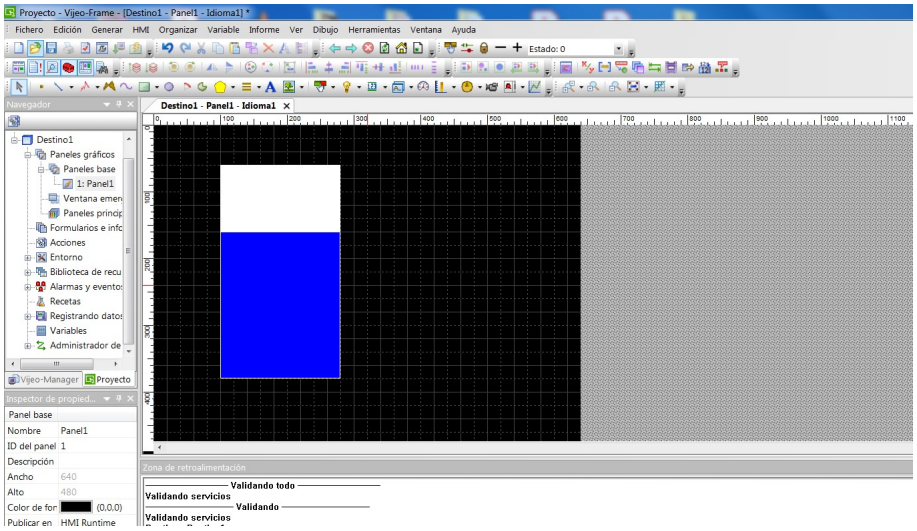


Figura 2.28 Interfaz de Vijeo Designer.

Sin embargo, después de recoger información de la web nos dimos cuenta de que nos hacía falta una licencia para conectar con el programa de EcoStruxure, por lo tanto tuvimos que rechazar esta opción. Por último, recurrimos a una herramienta ya incluida en MATLAB® llamada MATLAB® GUI. Esta herramienta nos permite crear una interfaz con la que puede interactuar el usuario. Esta interfaz puede contener diferentes elementos como pueden ser botones, sliders, gráficas, menús despegables y recuadros de texto editable.

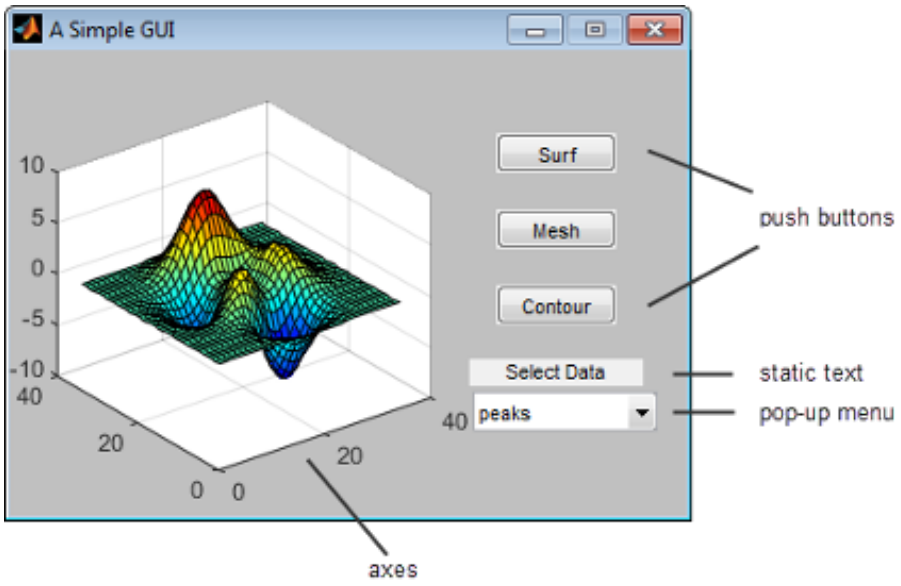


Figura 2.29 Interfaz de MATLAB® GUI.

Esta herramienta no tiene tantos elementos como puede tener el programa nombrado anteriormente, Vijeo Designer, sin embargo, incluye todo lo necesario para la visualización que se pide. Para crear esta interfaz usando MATLAB® GUI habrá que escribir en la ventana de comando la palabra GUIDE. Una vez hecho esto nos dará la opción de crear una interfaz nueva o editar una ya creada anteriormente. Después podremos añadir todos los elementos que se precise a nuestra interfaz de usuario. Una vez hayamos terminado de crear nuestra interfaz, se creará una serie de funciones asociadas a la misma. Esta serie de funciones indicarán que se hará cuando se abra la interfaz o cuando el usuario interactúe con la misma, es decir, si por ejemplo se quiere que se ejecute un programa en el momento que el usuario pulse en un botón, entonces iremos a la función 'Callback' de ese elemento y añadiremos la función que queramos que se haga.

Al principio, al intentar usar este tipo de herramienta, se intentó implementar directamente en el archivo de Simulink con un bloque de la función en Matlab. Sin embargo, esto no nos servía ya que lo que hacía era crear una nueva interfaz por cada ciclo del archivo de simulink, por lo que no nos dejaba interactuar con la interfaz. Aquí se veía como lo intentamos implementar en el archivo de Simulink.

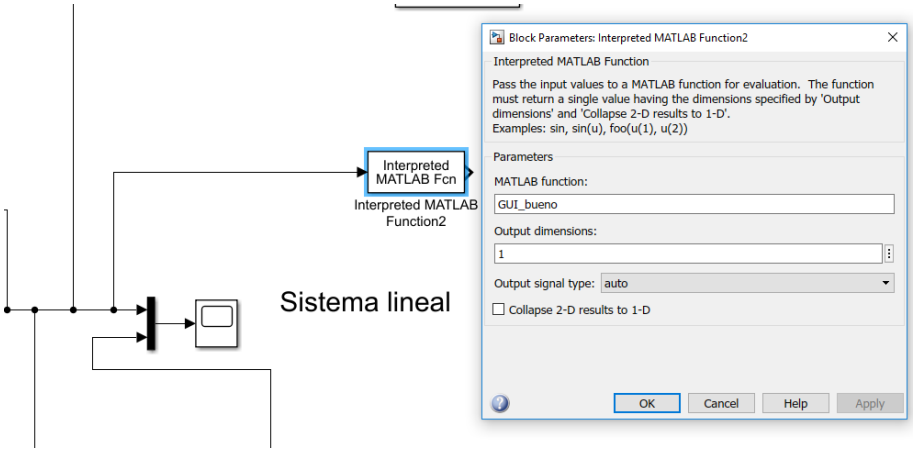


Figura 2.30 MATLAB® GUI en bloque de Simulink.

Después de esto, se buscó información sobre como acceder a las variables y datos de un archivo de Simulink en proceso. Esto resultaba complicado ya que no se disponían de muchas herramientas debido a que se

obtienen los datos de las simulaciones en Simulink normalmente cuando ya ésta ha terminado. enviándolos al Workspace. A pesar de esto, se consiguió acceder a estos datos utilizando las funciones **get_param** y **set_param**. Con estas funciones podríamos obtener el valor de uno de los bloques de Simulink y si se añadía el argumento 'RuntimeObject' lo podíamos ver en ese instante. También se intentó poner en bucle las funciones anteriores (**get_param** y **set_param**) para que no parasen de mandar y recibir información, pero una vez entraba en dicho bucle infinito la simulación se detenía hasta que dicho bucle terminase.

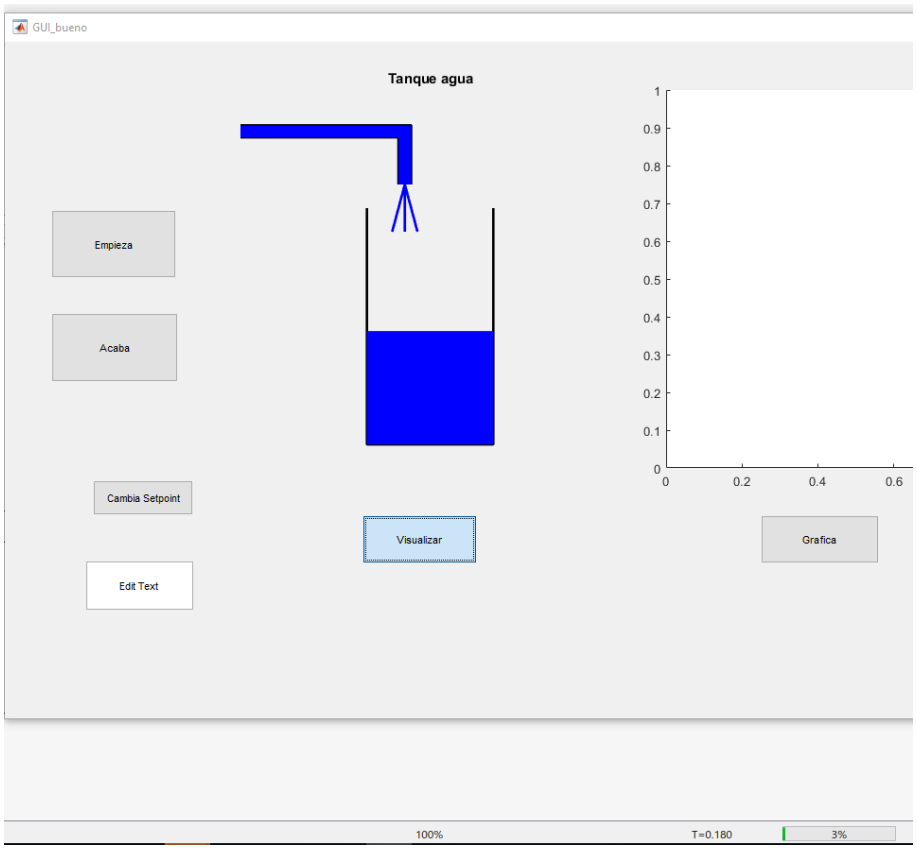


Figura 2.31 Simulación pausada a causa del bucle en MATLAB® GUI.

Por lo tanto, no tuvimos más opción que refrescar los datos de la interfaz del MATLAB® GUI cada vez que el usuario presionase algún botón, sin usar bucle alguno para no pausar la simulación en ningún momento.

3 Arquitectura final

Para crear nuestra plataforma, después de haber hecho las pruebas necesarias, tomaremos la siguiente arquitectura.

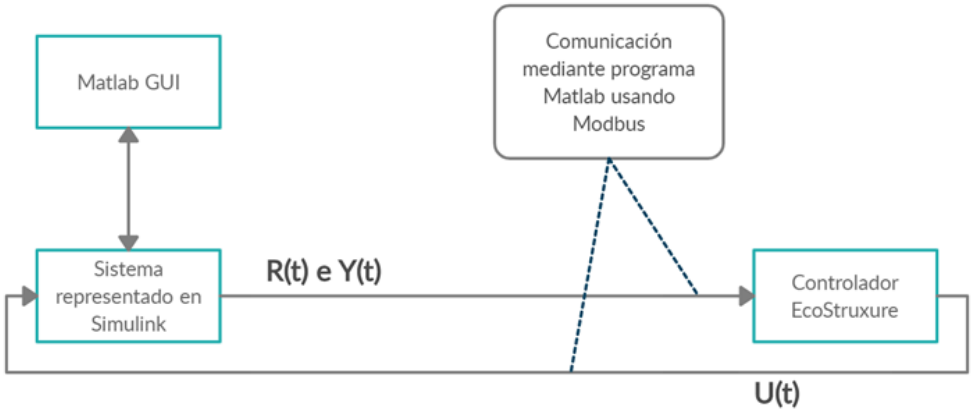


Figura 3.1 Arquitectura del proyecto.

Como se puede ver en la imagen, utilizaremos en total dos programas diferentes: MATLAB® y EcoStruxure. En el primero representaremos el sistema a simular con la herramienta de Simulink, para ello pondremos su ecuación o función de transferencia. Después de esto, añadiremos la función de comunicación que conecta EcoStruxure con MATLAB® , esta función tendrá como entradas el tiempo local de Simulink (t), el set point o referencia (R(t)) y la variable a controlar (Y(t)) y como salida tendrá la acción de control (U(t)) que se le pasará a la función de transferencia. Por lo tanto, nuestra arquitectura dentro del archivo de Simulink quedaría tal como se puede observar en la siguiente imagen

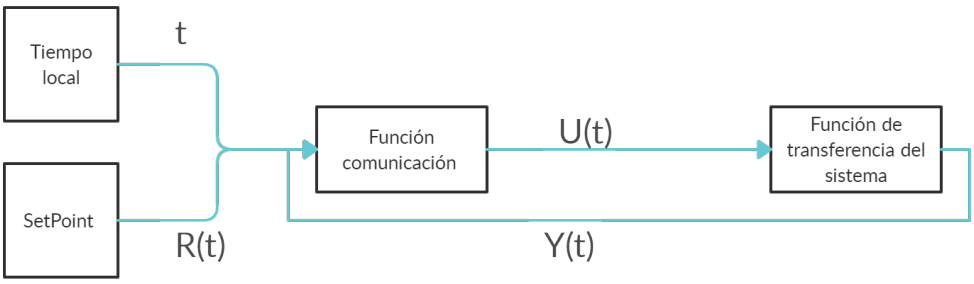


Figura 3.2 Representación del archivo de Simulink.

Adicionalmente, pondremos un step size del archivo de Simulink relativamente bajo, lo recomendable es que sea en torno a 0.01 para su correcto funcionamiento. Para hacer esto iremos a los parámetros de Simulink, cambiaremos a Variable Step-size y en max step size el 0.01 que se ha comentado antes. También se podrá usar el bloque Zero Order Hold para cambiar el step size indirectamente.

Como se puede ver en el diagrama anterior, el archivo de Simulink le enviará el tiempo local de la simulación al programa de EcoStruxure, el cual lo guardará en una variable, pero será el programa del EcoStruxure el que decidirá cuando mandar la acción de control, es decir, decidirá el tiempo de muestreo. Esto lo hace mediante un código simple en LD que se compone de una variable y un bloque de comparación en ST. En el código se ve cuando el tiempo local que se nos pasa es múltiplo de nuestro tiempo de muestreo. Para ello usamos la función REM que ve el resto de una división, si este resto es cero significa que es múltiplo por lo cual se cambiará la acción de control. Aquí se ve el código.

Figura 3.3 Tiempo de muestreo usando tiempo local de Simulink.

Después de haber hecho esto, se dará pie a mandar la acción de control mediante el flanco de subida de la variable %M5, como se puede ver en la siguiente imagen.

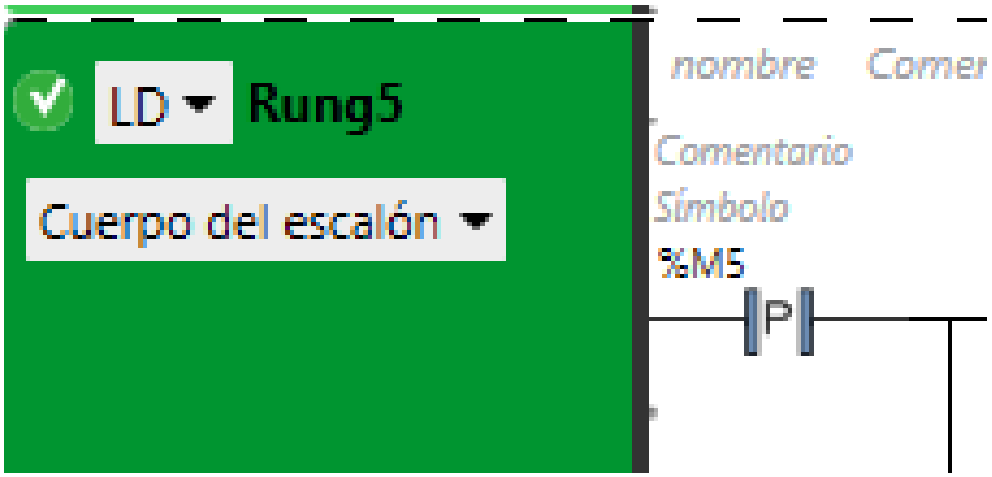


Figura 3.4 Flanco de subida vairable %M5.

Además, también se debe añadir a la línea del error acumulado un bloque de comparación para que no se coja error acumulado antes de empezar a controlar.

Figura 3.5 Bloque de comparación tiempo local>0.

Todo esto se hace debido a que Simulink, como ya bien antes dijimos, no trabaja en tiempo real, por lo que necesariamente tenemos utilizar el tiempo local del programa y mandárselo al controlador del EcoStruxure para poder tener la misma escala de tiempo en ambos programas. Por último, volvemos a poner el código de implementación final.

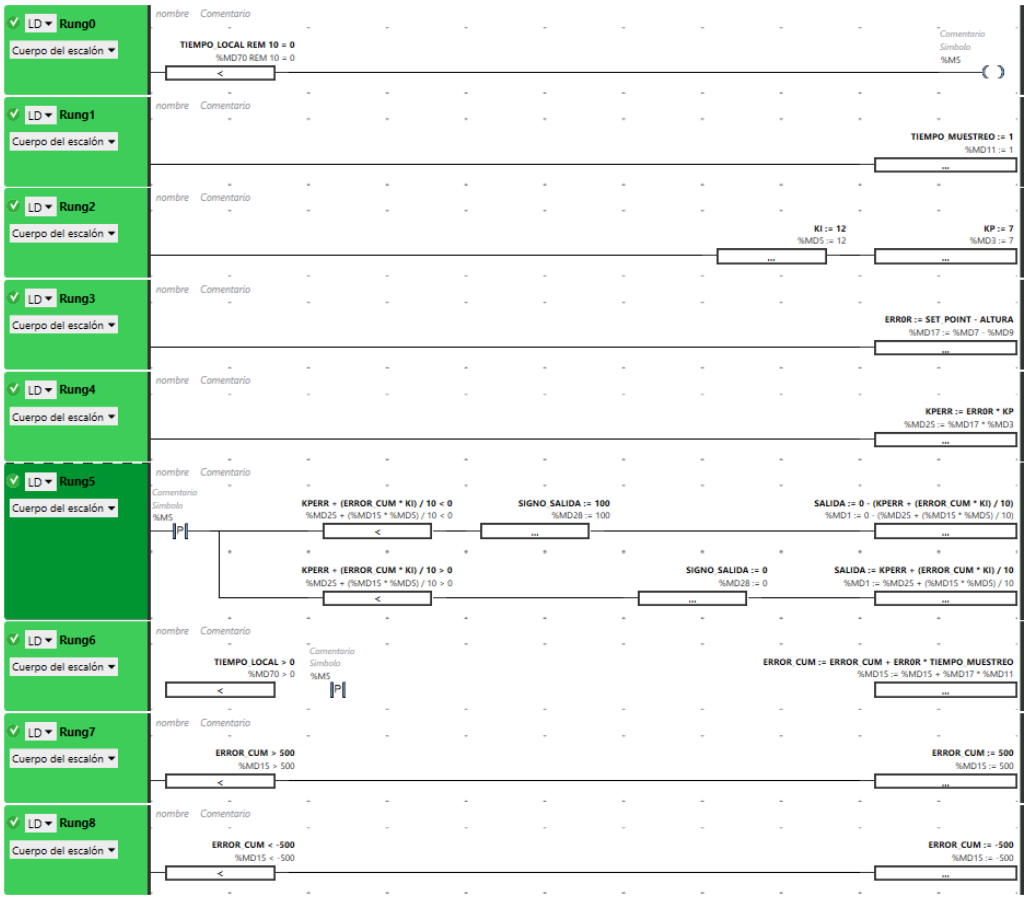


Figura 3.6 Código de controlador.

4 Guía instalación

4.1 Instalación de EcoStruxure

Para empezar dicha instalación utilizaremos el siguiente enlace de descarga para obtener el ejecutable de dicho programa:
https://download.schneider-electric.com/files?p_enDocType=Software+-+Released&p_File_Name=MachineExpertBasic_V1.0_SP2_build64454.exe&p_Doc_Ref=Machine_Expert_Basic

Una vez hecho esto, pasaremos a instalar el programa como se puede ver en la siguiente imagen Fig 5.9

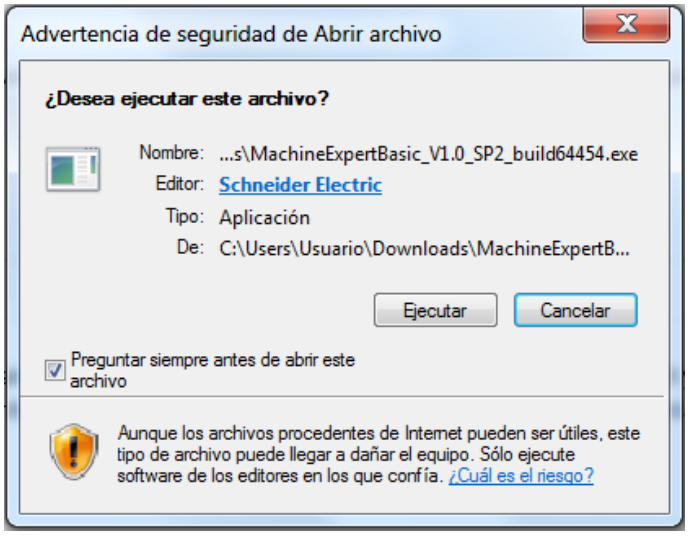


Figura 4.1 Ejecutable EcoStruxure.

Lo siguiente será escoger el idioma con el que trabajará el programa, en nuestro caso elegiremos el español Fig 4.2

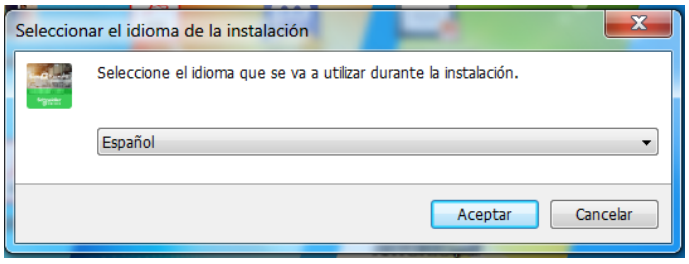


Figura 4.2 Elección de idioma.

Después de esto pulsaremos en siguiente y luego rellenaremos los datos que nos piden Schneider, al solo necesitar la versión gratuita de la aplicación no nos tenemos que preocupar de licencias asi que tampoco importa lo que rellenemos en los siguientes campos. Fig 4.3

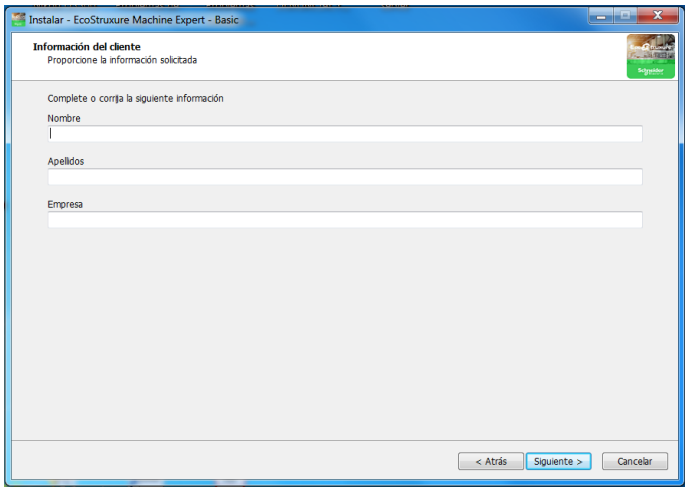


Figura 4.3 Elección de idioma.

Seguidamente continuaremos aceptando los términos y condiciones de uso del programa como se puede ver en Fig 4.4

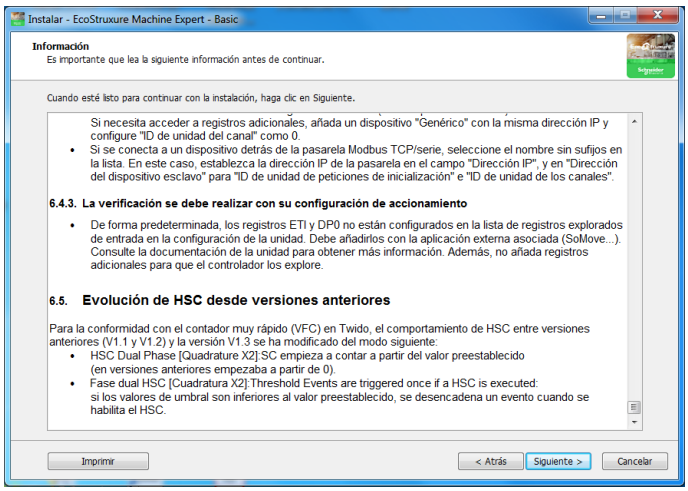


Figura 4.4 Términos y condiciones de uso.

A continuación, seleccionamos la carpeta donde se guardará el programa Fig 4.5, lo instalaremos Fig 4.11 y por último abriremos el programa.

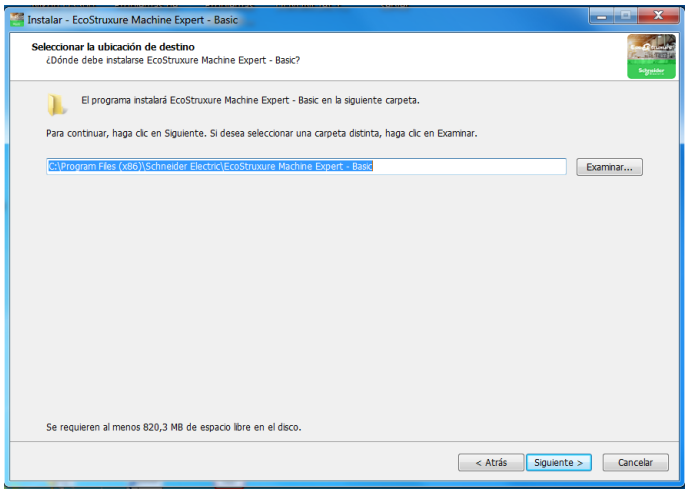


Figura 4.5 Carpeta del programa.

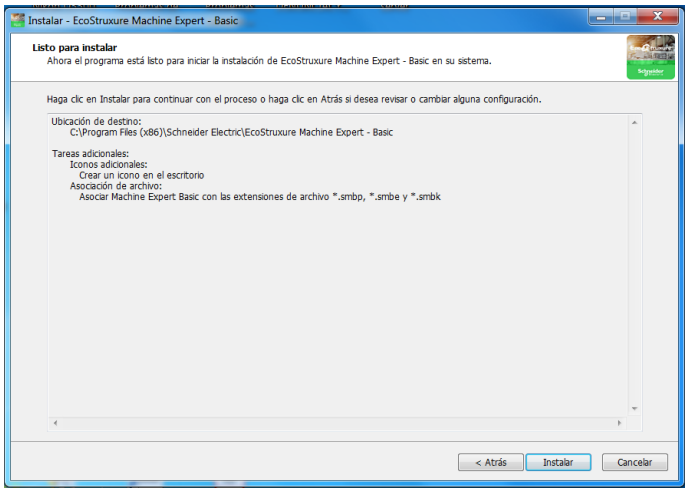


Figura 4.6 Instalación programa.

Una vez dentro del programa, si nos damos cuenta nos dará fallo al ejecutar un proyecto. Para cambiar esto, deberemos irnos al apartado de propiedades y luego al subapartado de protección de la aplicación, como se puede ver en la imagen Fig 4.7

Figura 4.7 Propiedades/Protección del sistema.

En este apartado cambiaremos tanto la protección de lectura como la de escritura desactivándolo ambos como se puede ver en Fig 4.8 y aplicaremos dichos cambios.

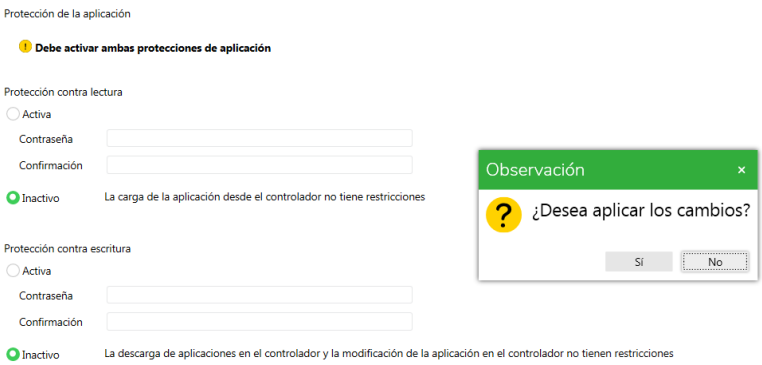


Figura 4.8 Propiedades/Protección del sistema.

4.2 Configuración para la comunicación

Para las funciones relacionadas a Modbus que se usarán en MATLAB® no hará falta instalación alguna ya que dicha librería ya viene incluida en el programa. Para establecer al principio la comunicación deberemos iniciar la simulación en el programa de EcoStruxure (no hace falta darle al play solo con iniciarlo es suficiente) y luego pondremos las dos siguientes lineas de código en la ventana de comando de MATLAB® .

Código 4.1 Ventana de comando de Matlab para establecer la comunicación.

```
global m
m = modbus('tcpip','127.0.0.1',502)
```

Por otra parte, para comunicarse se utilizará un programa de MATLAB® que contendrá las funciones de Modbus que servirá tanto para enviar como para recibir datos. Dicho programa tendremos que incluirlo en nuestro archivo de Simulink donde se encontrará el sistema a simular. Para ello utilizaremos el bloque de Simulink que permite utilizar programas ya creados anteriormente en Matlab, dicho bloque se llama 'Interpreted MATLAB® function'.

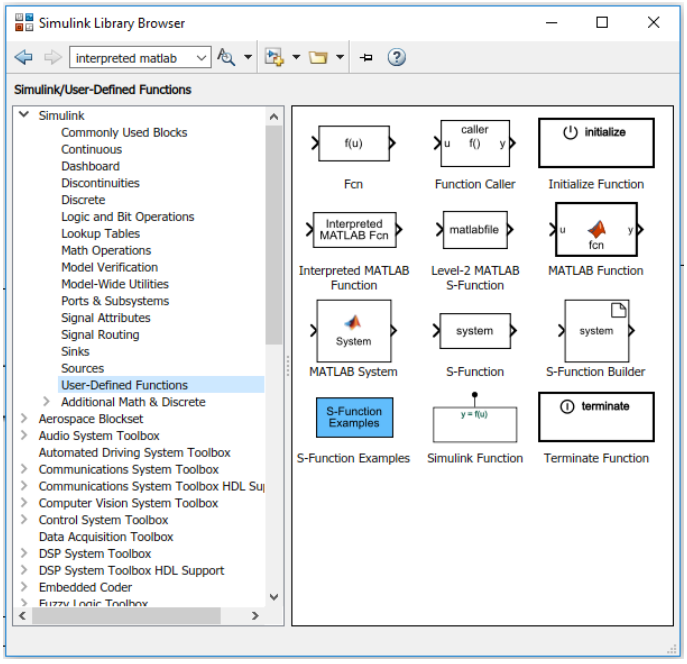


Figura 4.9 Interpreted MATLAB® function.

También deberemos asignar nombres a las variables que vayamos a usar en el EcoStruxure, siempre recordando usar las %MD para poder tener más rango de valores y siempre con un espacio mínimo de un elemento de distancia en la memoria del EcoStruxure. Para asignar esas variables acudiremos a herramientas, dentro de aquí pulsaremos en objetos de memoria y luego en palabras de memoria. Aquí se ve como deberían quedar asignadas las variables.

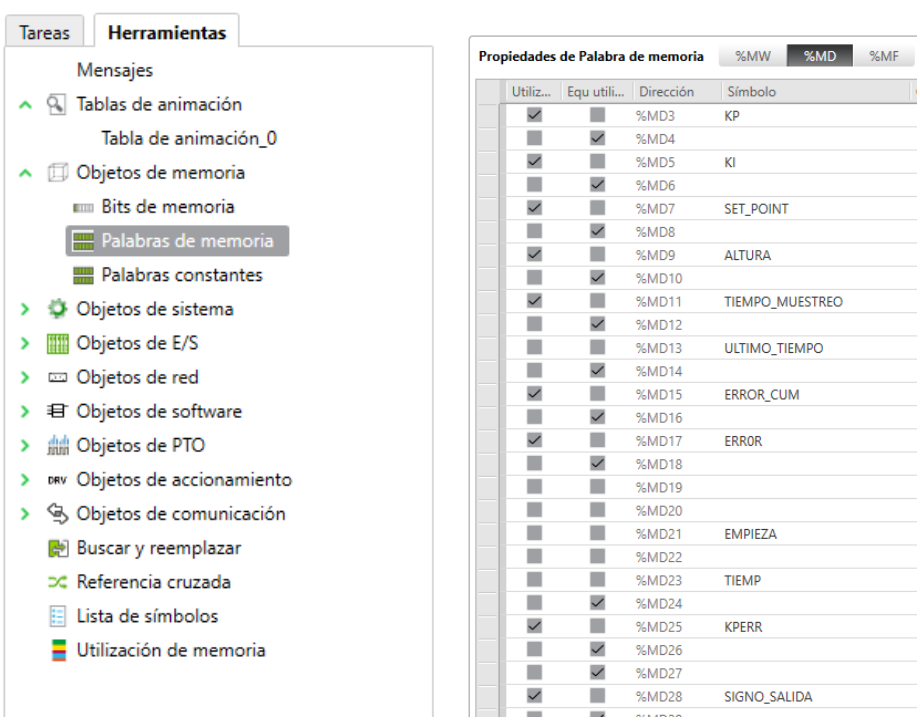


Figura 4.10 Variables asignadas.

Para terminar, deberemos asegurarnos en que la tarea maestra está en la modalidad de exploración normal para que así el tiempo de ciclo sea lo más rápido posible, esto ya viene puesto por defecto. También se podría cambiar para que el tiempo de ciclo del programa tomase el valor que el usuario quiera en el caso de que sea necesario (entre 1 y 150 ms)

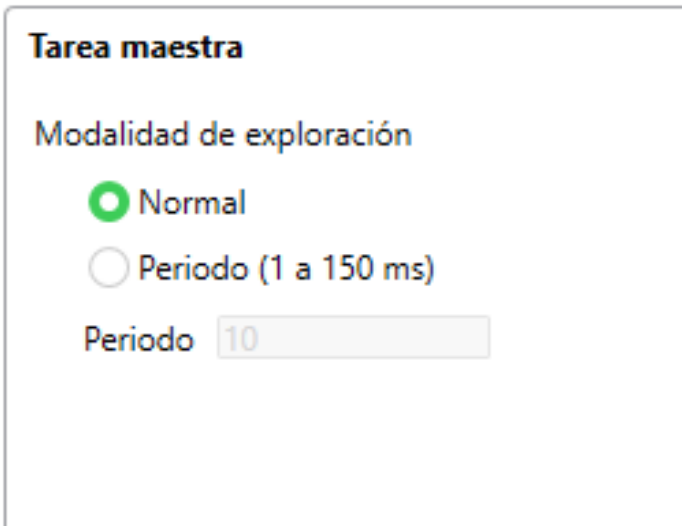


Figura 4.11 Tiempo de ciclo de tarea maestra.

5 Ejemplo de demostración

Como ejemplo de demostración con esta plataforma utilizaremos un sistema basado en un tanque de agua, el cual se controlará para tener un nivel de agua especificado por el usuario. El nivel de agua del tanque variará entre los 0 y 10 metros como se observa en la tabla 2.1. Dicho tanque tendrá una entrada de agua que se podrá regular y una salida fija. En principio, la entrada de agua a través del grifo no tendrá ningún tipo de restricción aunque en un futuro se podría añadir sin ningún tipo de problema. En la imagen siguiente se puede observar como sería dicho sistema.

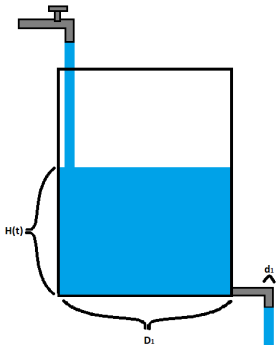


Figura 5.1 Sistema de tanque de agua.

Para dicho sistema se utilizarán ciertos valores para sus dimensiones y variables. Volvemos a recalcar que $q(t)$ tendrá no tendrá valor máximo en principio.

Tabla 5.1 Valores de parámetros.

Definición	Notación	Valor
Radio tanque	R_1	1 m
Radio válvula de salida	r_1	0.5 m
Área tanque	A_1	3.14 m^2
Área válvula de salida	a_1	0.785 m^2
Altura tanque	$h(t)$	0 a 10 m
Gravedad	g	9.8 m/s^2
Caudal de entrada	$q(t)$	- m^3

Seguidamente pasamos a ver como sería la ecuación que representaría nuestro sistema a controlar. Dicha ecuación hace referencia a la variación de la altura en función de la entrada y de la salida de agua del tanque

$$A_1 \frac{dh(t)}{dt} = q(t) - (a_1 \cdot \sqrt{2g})\sqrt{h(t)} \tag{5.1}$$

Como se puede observar a simple vista es una ecuación no lineal, así que a continuación pasaremos a linealizarla en torno a un punto de operación. En este caso el punto de operación del tanque de agua será:

$$q_0 = (a_1 \cdot \sqrt{2g})\sqrt{h_0} \tag{5.2}$$

Donde en este caso, intentaremos controlarlo en torno a la altura de h_0 = 5 m, por lo tanto usando la ecuación anterior nos quedaría que q_0 sería:

$$q_0 = (0.785 \cdot \sqrt{2 \cdot 9.8}) \cdot \sqrt{5} = 7.78m^3 \tag{5.3}$$

Por otro lado, nuestras variables incrementales serán las siguientes:

$$q(t) = q_0 + \triangle q(t); \tag{5.4}$$
$$h(t) = h_0 + \triangle h(t); \tag{5.5}$$

Seguimos con las ecuaciones de las variables incrementales

$$A_1 \frac{d\triangle h(t)}{dt} = \triangle q(t) - \frac{(a_1 \cdot \sqrt{2g})}{2\sqrt{h_0}} \triangle h(t) \tag{5.6}$$

Usando la transformada de Laplace en estas ecuaciones nos quedarán tal que así:

$$A_1(s \cdot H(s) - h_0) = Q(s) - \frac{(a_1 \cdot \sqrt{2g})}{2\sqrt{h_0}} H(s) \tag{5.7}$$

$$\frac{H(s)}{Q(s)} = \frac{1}{A_1 \cdot s + \frac{(a_1 \cdot \sqrt{2g})}{2\sqrt{h_0}}} \tag{5.8}$$

Que intercambiando las constantes por sus respectivos valores tendríamos la siguiente función de transferencia que definiría nuestro sistema.

$$\frac{H(s)}{Q(s)} = \frac{1}{3.14 \cdot s + 0.778} \tag{5.9}$$

5.1 Representación en Matlab

Una vez visto como será nuestro sistema tenemos que plasmarlo en el programa de MATLAB® . Para ello usaremos la herramienta de Simulink. Por una parte, pondremos nuestro sistema sin linealizar y por otra nuestro sistema linealizado.

5.1.1 Sistema sin linealizar

Por la parte del sistema sin linealizar, nos fijaremos en la ecuación (5.1) para representarlo en la herramienta de Simulink. Primero crearemos este sistema usando los bloques que nos brinda Simulink, entre ellos el bloque de integrador, el de ganancia y el de raíz cuadrada. Quedándonos así el sistema:

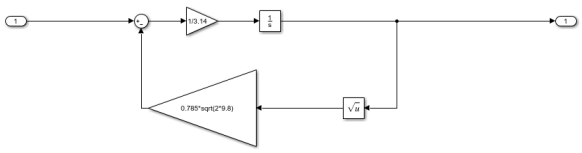


Figura 5.2 Bloque del sistema sin linealizar.

Después de esto, añadiremos a este subsistema que hemos creado en Simulink más bloques de funciones, entre ellos un bloque de constante que dará la altura a la que se tendrá que quedar el nivel del agua del tanque. También se puede observar un bloque con una función de MATLAB® la cual sirve para comunicarse con el controlador del otro programa, se explicará el funcionamiento de esta función más tarde. Aparte también usaremos bloques Scopes y Muxes para verificar el funcionamiento del programa. Adicionalmente, utilizaremos un bloque de ganancia que nos servirá para la visualización de nuestro sistema más adelante.

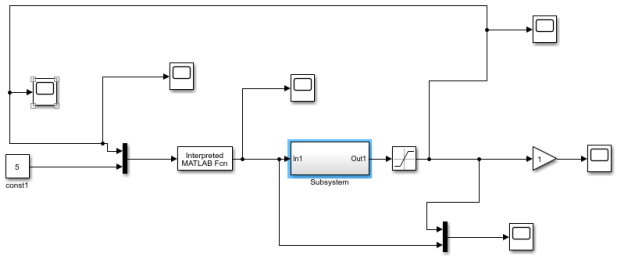


Figura 5.3 Programa simulink del sistema sin linealizar.

5.1.2 Sistema linealizado

Por otra parte, el sistema linealizado será representado utilizando el bloque de función de transferencia y fijándonos en la ecuación (5.9).

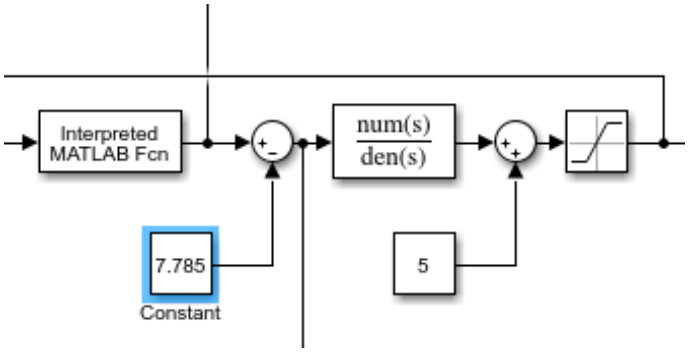


Figura 5.4 Programa Simulink del sistema linealizado.

Como se puede observar, se utilizan los mismo bloques que para el sistema sin linealizar, sin embargo se añaden los valores del punto de funcionamiento, estos valores son los dados por la ecuación (5.3).

5.2 Control

Para el control utilizaremos un controlador de tipo PI, como se veía en el apartado de pruebas realizadas. Aquí se recuerda la ecuación de dicho controlador:

$$Kp + Ki \cdot T_s(\frac{1}{z-1}) \tag{5.10}$$

Los valores de dichas constantes serán los siguientes:

$$Kp = 20 \tag{5.11}$$

$$Ki = 50 \tag{5.12}$$

Aparte también añadiremos una compensación anti wind up, al igual que cuando se probó anteriormente en las pruebas realizadas. Para realizar dicha compensación añadiremos un límite de 200 y -200 al error acumulado, de esta forma no se disparará la acción de control y no se producirá tanta sobreoscilación. También cambiaremos el signo de la acción de control si ésta es negativa volviéndola a poner positiva, a la vez que hacemos esto cambiaremos también el valor de la variable que indicará el signo de la acción de control al archivo de Simulink. Como se dijo en los apartados anteriores, pasaremos el tiempo local de Simulink al controlador y este enviará cada 100 ms la acción de control.

Una vez realizado todos estos pasos, nos quedaremos con el siguiente programa en EcoStruxure:

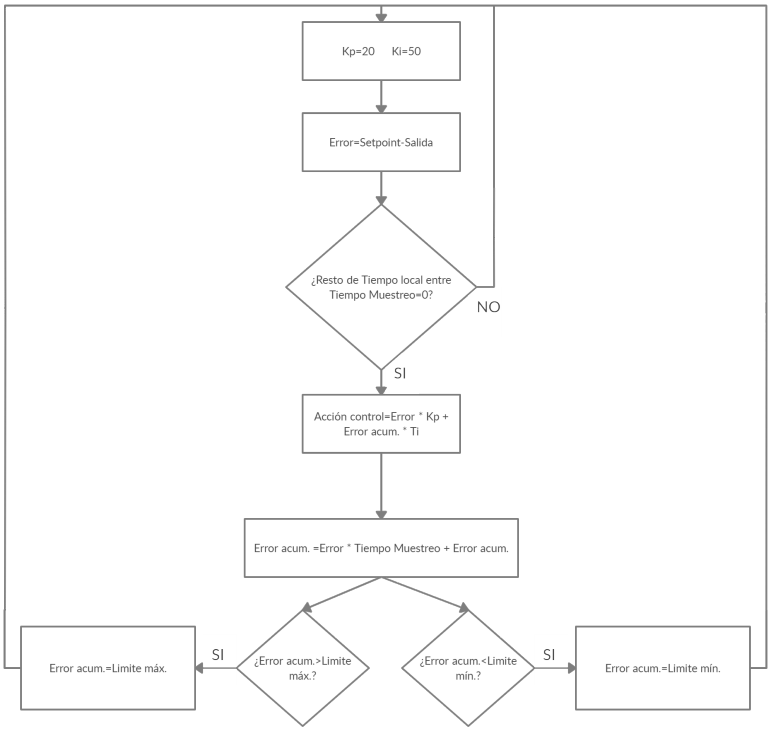


Figura 5.5 Pseudocódigo del programa.

POU

Tarea maestra

1 - Nuevo POU

Tarea maestra

Rung0

Variables utilizadas:

%M5

%MD70 TIEMPO_LOCAL

Rung1

Variables utilizadas:

%MD11 TIEMPO_MUESTREO

Rung2

Variables utilizadas:

%MD3 KP

%MD5 KI

Rung3

Leyenda:

1 %MD17 := %MD7 - %MD9

Variables utilizadas:

%MD7 SET_POINT

%MD9 ALTURA

%MD17 ERR0R

Rung4

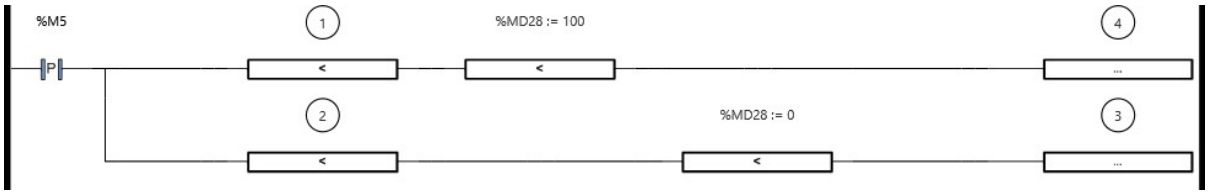
Leyenda:

```
1    %MD25 := %MD17 * %MD3
```

Variables utilizadas:

```
%MD3                KP
%MD17               ERR0R
%MD25               KPERR
```

Rung5



Leyenda:

```
1    %MD25 + (%MD15 * %MD5) / 10 < 0
2    %MD25 + (%MD15 * %MD5) / 10 > 0
3    %MD1 := %MD25 + (%MD15 * %MD5) / 10
4    %MD1 := 0 - (%MD25 + (%MD15 * %MD5) / 10)
```

Variables utilizadas:

```
%M5
%MD1                SALIDA
%MD28               SIGNO_SALIDA
```

Rung6

Leyenda:

```
1    %MD15 := %MD15 + %MD17 * %MD11
```

Variables utilizadas:

```
%M5
%MD15               ERROR_CUM
%MD70               TIEMPO_LOCAL
```

Rung7

Variables utilizadas:

```
%MD15               ERROR_CUM
```

Rung8

Variables utilizadas:

%MD15	ERROR_CUM

SÍMBOLOS

Utilizado	Dirección	Símbolo	Comentario
X	%MD1	SALIDA	
X	%MD3	KP	
X	%MD5	KI	
X	%MD7	SET_POINT	
X	%MD9	ALTURA	
X	%MD11	TIEMPO_MUESTREO	
X	%MD15	ERROR_CUM	
X	%MD17	ERR0R	
X	%MD25	KPERR	
X	%MD28	SIGNO_SALIDA	
X	%MD70	TIEMPO_LOCAL	

Por último, aquí se puede observar la repuesta del sistema y la acción de control.

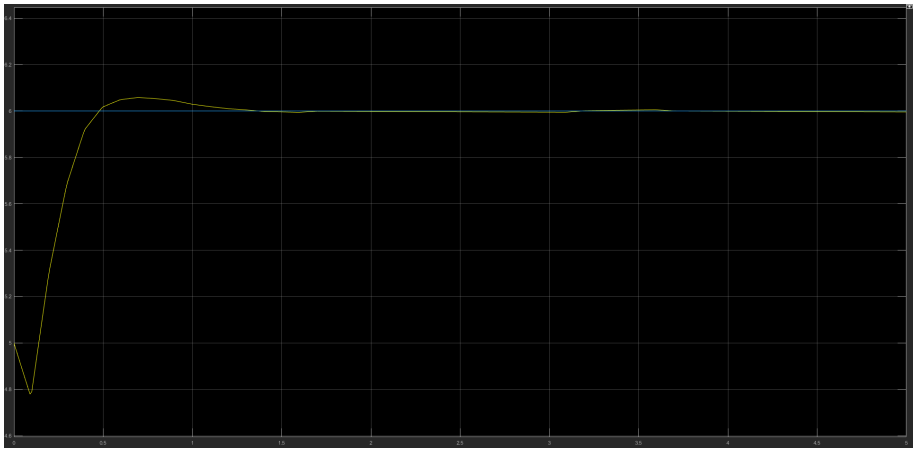


Figura 5.6 Respuesta final.

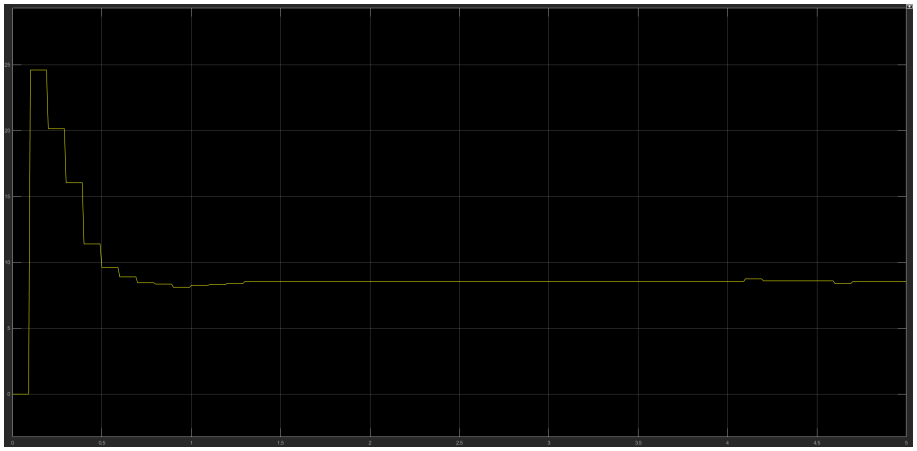


Figura 5.7 Acción control final.

5.3 Comunicación

Para comunicar ambos programas usaremos el protocolo Modbus mediante las funciones de MATLAB® , las cuales ya fueron probadas en apartados anteriores. Primero se creará la conexión una vez se haya puesto en marcha el programa EcoStruxure usando la función **modbus**. Después de esto, pondremos una función dentro de nuestro archivo en Simulink el cual contendrá las funciones necesarias para la comunicación.
Dicho programa, como ya hemos comentado anteriormente, tendrá tres entradas: altura, setpoint y tiempo local. Estas variables las multiplicaremos por 100 y seguidamente las redondearemos para poder enviarlas usando el protocolo MODBUS, de esta forma enviaremos datos enteros y a la vez cogeremos hasta las centésimas de las variables de entrada. Una vez hecho esto, recogeremos el valor de la acción de control y su signo. Por último, cambiaremos el signo de la acción de control en función del valor de la variable signo y reescalaremos a su dimensión original la acción de control dividiéndola entre 100.

La frecuencia de ejecución de esta función depende del step size del archivo de Simulink. Este step size puede cambiarse mediante el bloque Zero Order Hold, como se explicó anteriormente. Aquí el código de dicha función:

Código 5.1 Función lectura_escritura.

```
function[K]=lectura_escritura2(entrada)
```

```
global m
entrada(1)=entrada(1)*100;
entrada(1)=round(entrada(1));
entrada(2)=entrada(2)*100;
entrada(2)=round(entrada(2));
entrada(3)=entrada(3)*100;
entrada(3)=round(entrada(3));
empieza=1;
write(m,'holdingregs',7,entrada(2),'int32');
write(m,'holdingregs',9,entrada(1),'int32');
write(m,'holdingregs',70,entrada(3),'int32');
write(m,'holdingregs',21,empieza,'int32');
MVi=read(m,'holdingregs',1,1,'int32');
signo=read(m,'holdingregs',28,1,'int32');

if (signo==100)
    K=-MVi/100;
else
K=MVi/100;
end

end
```

Aquí se puede ver como se le llama a la función de la comunicación en el archivo de Simulink.

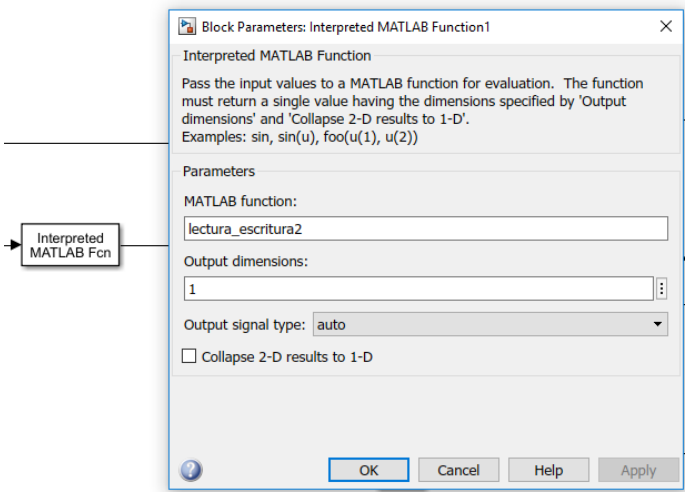
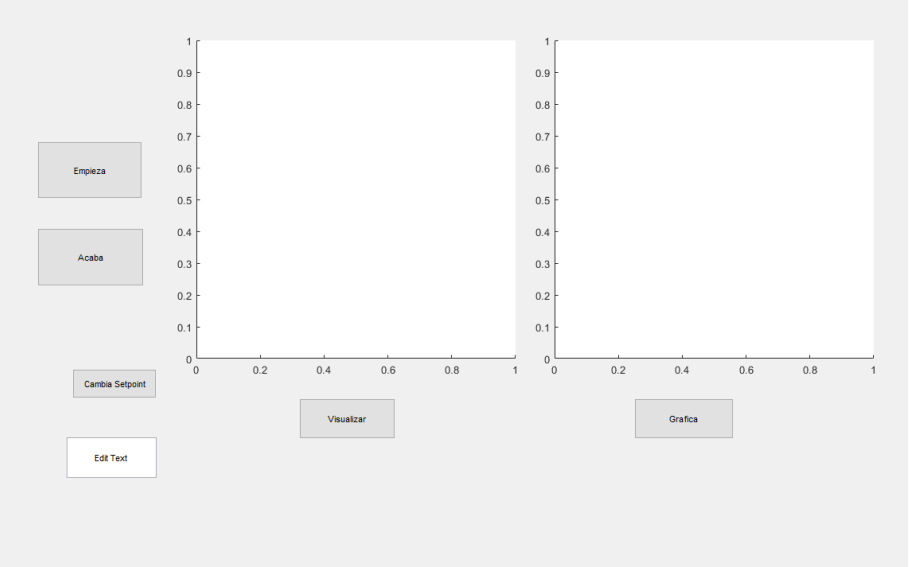


Figura 5.8 Bloque de comunicación en Simulink.

5.4 Visualización

En nuestra interfaz añadiremos en total dos gráficas, una para dibujar el tanque de agua y otra para dibujar la gráfica de todo el proceso una vez este haya terminado. Adicionalmente, crearemos un texto estático para introducir el valor del setpoint y cinco botones sin retención, sus funciones serán las siguientes: Iniciar la simulación, terminar la simulación, cambiar el setpoint una vez se introduzca dicho valor en el texto editable, refrescar el dibujo del tanque y dibujar la gráfica del desarrollo de la respuesta del sistema. Nuestra interfaz por lo tanto quedaría así:



Después de haber creado nuestra interfaz, iremos a las funciones asociadas a ésta que se crearán automáticamente. Dentro de éstas, cambiaremos las llamadas o 'Callbacks' de algunos elementos de la interfaz.
Por una parte, editaremos la llamada del botón 'Empieza' de modo que cuando este se pulse empiece la simulación. Para ello, usaremos la función **set_param()**, esta función cambia los parámetros de un bloque o de un modelo especifico. Los argumentos de dicha función son los siguientes, **set_param(Object,ParameterName,Value)**. *Object* hace referencia al modelo que vamos a editar, *ParameterName* al parámetro que se quiere cambiar y *Value* el valor nuevo que se le quiere dar. Para la llamada del botón empieza usaremos esta función con los siguientes argumentos para iniciar la simulación **set_param()**

Código 5.2 Empieza_Callback.

```
% --- Executes on button press in Empieza.
function Empieza_Callback(hObject, eventdata, handles)
% hObject   handle to Empieza (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

set_param('pruebaconexionbuena','SimulationCommand','start');
```

Para la llamada del botón 'Acaba' usaremos la misma función cambiando el último argumento y poniendo Stop, para que así se acabe la simulación.

Código 5.3 Acaba_Callback.

```
% --- Executes on button press in Acaba.
function Acaba_Callback(hObject, eventdata, handles)
% hObject   handle to Acaba (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
set_param('pruebaconexionbuena','SimulationCommand','stop');
```

En la llamada del botón 'Cambia Setpoint', primero leeremos lo que haya escrito el usuario en el texto editable utilizando la función **get()**, lo guardaremos en una variable llamada *res* y por último volveremos a usar la función **set_param** para cambiar el valor del bloque de simulink con el valor del setpoint.

Código 5.4 Cambiaref_Callback.

```
% --- Executes on button press in Cambiaref.
function Cambiaref_Callback(hObject, eventdata, handles)
% hObject   handle to Cambiaref (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
res=get(handles.edit1,'String');
load_system('pruebaconexionbuena');
find_system('Name','pruebaconexionbuena');
open_system('pruebaconexionbuena');
set_param('pruebaconexionbuena/const1','Value',res{1});
```

En la llamada del botón 'Grafica', representaremos con una función el desarrollo de la simulación en la segunda gráfica (axes2), para ello usaremos la información que nos devuelve uno de los Scopes que se encuentra dentro de nuestro modelo de Simulink. Solo comentar también que para obtener dicha información almacenada en el Workspace hará falta usar la función **evalin()**.

Código 5.5 Grafica_Callback.

```
% --- Executes on button press in Grafica.
function Grafica_Callback(hObject, eventdata, handles)
% hObject   handle to Grafica (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
ScopeData = evalin('base', 'ScopeData');
axes(handles.axes2);
plot(ScopeData.time(:),ScopeData.signals.values(:));
```

Por último, dentro de la llamada del botón 'Visualizar', representaremos con un dibujo como varía la altura del tanque en tiempo real, para ello primero usaremos la función **get_param**. Esta función tiene el objetivo contrario a la función **set_param**, ya que ésta en vez de editar un parámetro, recoge el valor de uno. Como primer argumento pondremos el parámetro del que queremos recoger su valor y como segundo argumento pondremos *'RuntimeObject'*, de esta forma se recogerá el valor que tiene en ese momento dicho parámetro durante la simulación. En nuestro caso recogeremos el valor de un bloque que tendrá la información de la altura del tanque de agua. A continuación, utilizaremos otro subprograma para representar dicho tanque llamado *water*.

Código 5.6 Visualizar_Callback.

```
% --- Executes on button press in Visualizar.
function Visualizar_Callback(hObject, eventdata, handles)
% hObject   handle to Visualizar (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
% end
axes(handles.grafica);
 rto = get_param('pruebaconexionbuena/Gain','RuntimeObject');
 water(rto.OutputPort(1).Data);
```

Este subprograma llamado *water* recibirá como argumento la altura del tanque de agua. El subprograma dibujará líneas en la gráfica para representar el tanque de agua y creará un rectángulo de color azul con dicha altura para representar el nivel del agua. Cada vez que se vuelva a ejecutar el subprograma se borrará el rectángulo anterior antes de crear el siguiente para que así no haya solapamiento entre uno y otro.

Código 5.7 Función water.

```
function water(altura)
delete(findobj('type', 'patch'));
```

```
line([0 1.25 1.25],[14 14 12],'LineWidth',2,'Color','black');
line([0 1.35 1.35],[14.5 14.5 12],'LineWidth',2,'Color','black');
patch([0 1.35 1.35 1.25 1.25 0 0], [14.5 14.5 12 12 14 14 14.5],'blue','
    EdgeColor','none');
line([1.3 1.2],[12 10],'LineWidth',2,'Color','blue');
line([1.3 1.3],[12 10],'LineWidth',2,'Color','blue');
line([1.3 1.4],[12 10],'LineWidth',2,'Color','blue');
line([1 1 2 2], [11 1 1 11],'LineWidth',2,'Color','black');
axis([0 3 0 16]);
patch([1 1 2 2 1], [1 altura+1 altura+1 1 1],'blue','EdgeColor','none');
title("Tanque agua")
axis off;
drawnow;
```

Por último, vemos como quedaría la visualización una vez simulado el sistema.

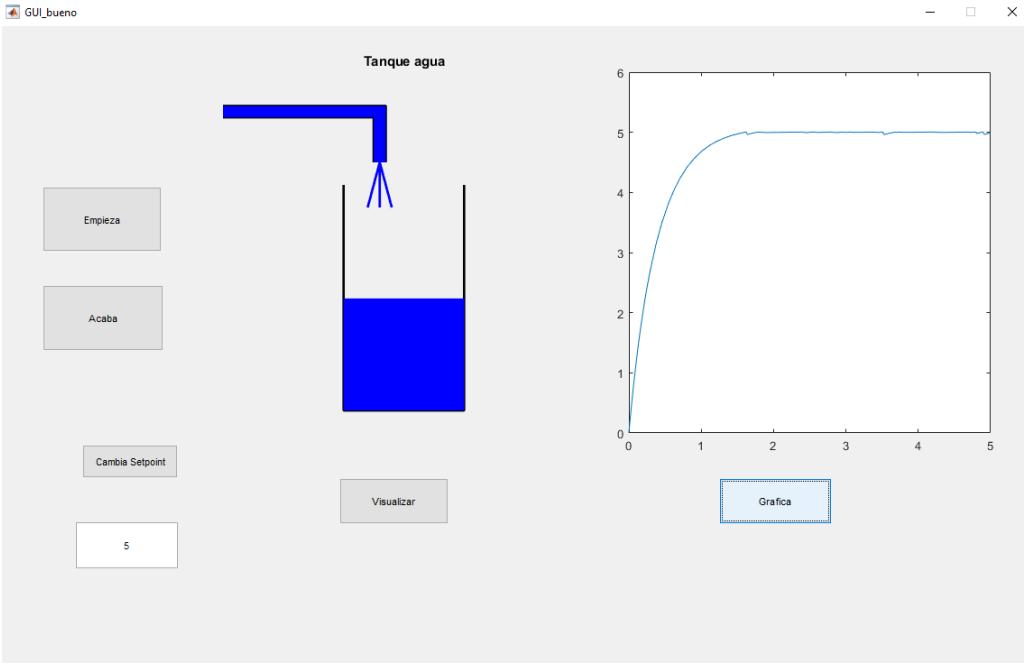


Figura 5.9 HMI una vez simulado.

6 Conclusiones

Una vez terminado este proyecto, sacamos las siguientes conclusiones que aparecen aquí:

1. La simulación finalmente no es en tiempo real sino en tiempo local de Simulink, esto no era nuestro objetivo principal pero es la única opción que había si queremos representar el sistema en MATLAB® Una posible mejora sería utilizar un software de MATLAB® llamado Simulink Desktop Real-time. Este software permitiría simular en tiempo real el sistema, sin embargo el software es de pago y no tiene versión de prueba. Debido a esto, no se pudo usar para este proyecto, pero en el caso de llegar a usarse para la escuela se podría plantear su compra.
2. Su simulación es relativamente lenta al tener que usar un step size pequeño para su correcto funcionamiento, por lo que se haría el proceso de aprendizaje más tedioso que si se estuviese haciendo la práctica en físico.
3. Las herramientas que se han usado para la visualización (MATLAB® GUIDE) son bastantes limitadas. Como mejora, podría conseguirse una licencia para usar Vijeo y tener más elementos que usar para la visualización al mismo tiempo que facilitarla. Adicionalmente, la conexión entre el programa EcoStruxure y Vijeo Designer es simple y no resultaría complicado para el alumnado realizarla.
4. Se pueden presentar problemas ajenos al aprendizaje con el PLC, como pueden ser problemas de uso de memoria del PLC simulado, de comunicación entre ambos dispositivos o de tipos de datos enviados entre sí. Es por esto que sigue siendo más recomendable hacer las prácticas con un PLC físico.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Javier García Caballero y Antonio Nuevo García Juan Manuel Escaño González, *Integración de sistemas de automatización industrial*, 1ª ed., Paraninfo, 2019.

[2] Schneider, *Licencias ecostruxure machine expert - basic*.

[3] _____, *Manual ecostruxure machine expert - basic*.

[4] _____, *Manual somachine*.

[5] _____, *Registro de somachine*.

[6] Juan Manuel Escaño González y José María Maestre Torreblanca, *Sistemas de medida y regulación*, 1ª ed., Paraninfo, 2018.