Capítulo 4

Control clásico aplicado a la planta

La primera estrategia que se prueba esta basada en un control clásico Proporcional, Integral y Derivativo. Más de la mitad de los controladores industriales que se usan hoy en día utilizan esquemas de control PID (*Proportional-Integral-Derivative*) o PID modificado. Debido a esto se ha pensado que un buen punto de partida para empezar a aplicar estrategias de control es empezar por los controladores más utilizados en la industria.

Durante este capítulo se hará un pequeño recordatorio a la teoría de control clásica para pasar luego a diseñar una estrategia de control que sea adecuada para el proceso que tiene lugar en la planta que se vio en el capítulo anterior.

Se explicará el funcionamiento de una herramienta de MATLAB muy útil para el diseño de controladores PID: *Simulink PID Controller Blocks.* Se utilizará este bloque de Simulink para empezar a diseñar la estrategia de control que más tarde implementaremos en un PID en Unity Pro.

Para terminar con este capítulo se realizarán una serie de ensayos para comprobar el funcionamiento del controlador diseñado con la planta.

4.1. Teoría control clásico

La utilidad de los controles PID estriba en que se aplican en forma casi general a la mayoría de los sistemas de control. En particular, cuando el modelo matemático de la planta no se conoce y, por lo tanto, no se pueden emplear métodos de diseño analíticos, es cuando los controles PID resultan más útiles. En el campo de los sistemas para control de procesos, es un hecho bien conocido que los esquemas de control PID básicos y modificados han demostrado su utilidad para aportar un control satisfactorio, aunque tal vez en muchas situaciones especificas no aporten un control óptimo.

El controlador proporcional-integral-derivativo (PID) ha sido ampliamente usado en los procesos industriales por ser simple, robusto y confiable.

$$C(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s\right) \tag{4.1.1}$$

El controlador consta de tres componentes sumadas, una reacciona al error en la medición entre el valor deseado o referencia y la variable a controlar, otro término reacciona ante la integral del error en el caso de que tengamos un error en estado estacionario que no se corrige con el primer término (el proporcional); vamos a intentar eliminar el error estacionario integrándolo con lo cual cada vez que pase el tiempo se hará más grande y por lo tanto la acción integral será mayor; y por último tenemos otro término, correspondiente a la derivada del error que reacciona ante los cambios en la pendiente del error; si de repente el error empieza a crecer rápidamente tenemos este término ir incrementando este término correctivo. Con estos tres término diseñamos un controlador PID.

Para mejorar la respuesta al escalón de un sistema mediante la aplicación de un controlador PID, las especificaciones de desempeño como el error en estado estacionario y sobreoscilación, se deben minimizar. Un controlador que consiste sólo en una ganancia se llama Proporcional (P). La velocidad a la cual la salida puede responder a la señal de error depende de la ganancia del controlador. Por lo tanto aumentando dicha ganancia, el tiempo de subida del sistema puede ser disminuido, permitiendo que la salida siga a la entrada más rápidamente. Sin embargo, esto agrega el problema de provocar un aumento de la sobreoscilación, causando oscilaciones en la salida, atentando contra la estabilidad del sistema. Una forma de reducir el tiempo de subida sin aumentar el porcentaje de sobreoscilación es agregar un término derivativo (D) al controlador P, dado que la derivada de la señal de error provee información acerca de cómo el error va cambiando con respecto al tiempo. De este modo, el controlador puede estimar valores futuros de la señal de error y compensar adecuadamente. Para eliminar el error de estado estacionario, se agrega un término integral (I). Este término le da la habilidad al controlador PID de recordar el pasado, permitiendo también dar una salida distinta de cero para una entrada nula. Así, dicho controlador permite tener un error de estado estacionario igual a cero.

Los controladores PID son los más conocidos y más usados en la industria debido a su estructura simple y un desempeño robusto en un intervalo amplio de condiciones de operación.

4.1.1. Diseño del algoritmo

Lo primero es elegir la estructura adecuada: P, PI, PD o PID.

Otro aspecto importante es elegir la forma: ideal o paralela, con saturación de la salida, con *anti-windup* del integrador, con una transferencia suave entre un modo de funcionamiento manual y automático una transición sin saltos.

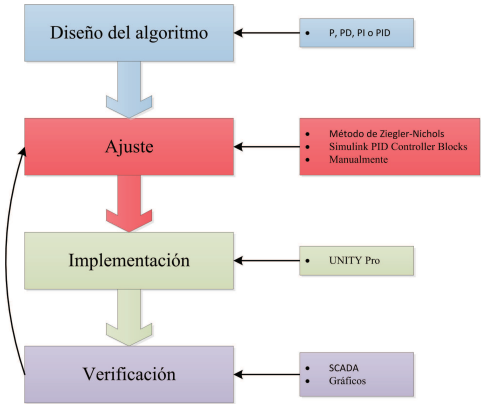


Figura 4.1.1: Diagrama de flujo de trabajo para el diseño de un PID.

4.1.2. Sintonización de controladores PID

Diseñar un controlador consiste en llegar a un balance entre la estabilidad del sistema, rechazo de perturbaciones, tiempos de subida, sobreoscilación, etc. El reto está en cómo ajustar el controlador para una planta inestable.

Como casi todos los controladores PID se ajustan en el sitio, se han propuesto muchos tipos diferentes de reglas de sintonización, que permiten llevar a cabo una sintonización delicada y fina de los controladores PID en el sitio. Asimismo, se han desarrollado métodos automáticos de sintonización y algunos de los controladores PID poseen capacidad de sintonización automática en línea.

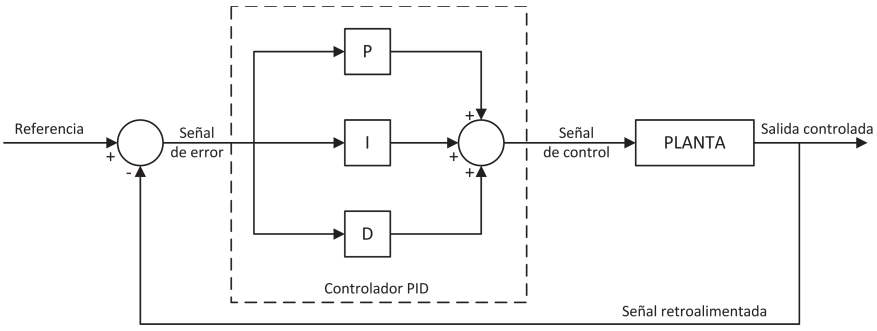


Figura 4.1.2: Esquema controlador PID de una planta.

Si se puede obtener un modelo matemático de la planta, es posible aplicar diversas técnicas de diseño con el fin de determinar los parámetros del controlador que cumpla las especificaciones del transitorio y del estado estacionario del sistema en lazo cerrado. Sin embargo, si la planta es tan complicada que no es fácil obtener su modelo matemático, tampoco es posible un método analítico para el diseño de un controlador PID. En este caso, se debe recurrir a procedimientos experimentales para la sintonía de los controladores PID.

El proceso de seleccionar los parámetros del controlador que cumplan con las especificaciones de comportamiento dadas se conoce como sintonía de controlador.

Ziegler y Nichols sugirieron reglas para sintonizar los controladores PID (esto significa dar valores a K_p, T_i *y* T_d) basándose en las respuestas escalón experimentales o en el valor de K_p que produce estabilidad marginal cuando solo se usa la acción de control proporcional. Las reglas de Ziegler-Nichols, que se presentan a continuación, son muy convenientes cuando no se conocen los modelos matemáticos de las plantas. Por supuesto, estas reglas se pueden aplicar al diseño de sistemas con modelos matemáticos conocidos. Tales reglas sugieren un conjunto de valores de K_p, T_i *y* T_d que darán una operación estable del sistema. No obstante, el sistema resultante puede presentar una gran sobreoscilación en su respuesta escalón de forma que resulte no aceptable. En tales casos se necesitará una serie de ajustes finos hasta que se obtenga el resultado deseado. De hecho, las reglas de sintonía de Ziegler-Nichols dan una estimación razonable de los parámetros del controlador y proporcionan un punto de partida para una sintonía fina, en lugar de dar los parámetros K_p, T_i *y* T_d en un único intento.

Reglas de Ziegler-Nichols para sintonizar controladores PID

Ziegler y Nichols propusieron reglas para determinar los valores de la ganancia proporcional K_p, del tiempo integral T_i y del tiempo derivativo T_d, basándose en las características de respuesta transitoria de una planta dada. Tal determinación de los parámetros de los controladores PID o sintonía de controladores PID se puede realizar mediante experimentos sobre la planta.

Hay dos métodos denominados reglas de sintonía de Ziegler-Nichols: el primero y el segundo método. A continuación se hace una breve presentación de estos dos métodos.

- Primer método. En el primer método, la respuesta de la planta a una entrada escalón unitario se obtiene de manera experimental. Si la planta no contiene integradores ni polos dominantes complejos conjugados, la curva de respuesta escalón unitario puede tener forma de S, como se observa en la figura 4.1.3. Este método se puede aplicar si la respuesta muestra una curva con forma de S. Tales curvas de respuesta escalón se pueden generar experimentalmente o a partir de una simulación dinámica de la planta. La curva con forma de S se caracteriza por dos parámetros: el tiempo de retardo L y la constante de tiempo T. El tiempo de retardo y la constante de tiempo se determinan dibujando una recta tangente en el punto de inflexión de la curva con forma de S y determinando las intersecciones de esta tangente con el eje del tiempo y con a línea *c(t)*=K. En este caso, la función de transferencia *C(s)/U(s)* se aproxima mediante un sistema de primer orden con un retardo del modo siguiente:

$$\frac{C(s)}{U(s)} = \frac{Ke^{-Ls}}{Ts+1} \tag{4.1.2}$$

- Segundo método. En el segundo método, primero se fija $T_i = \infty$ y $T_d = 0$. usando solo la acción de control proporcional, se incrementa $K_p$desde 0 hasta un valor crítico K_{cr}, en donde la salida presente oscilaciones sostenidas. (Si la

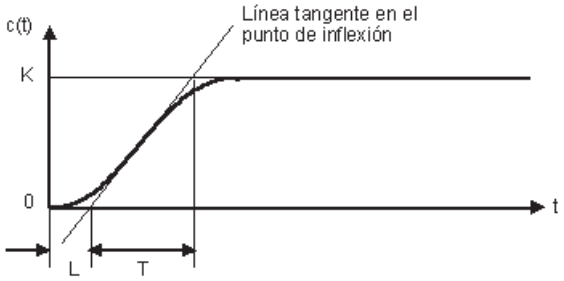


Figura 4.1.3: Curva de respuesta en forma de S.

Tipo de controlador	K_p	T_i	T_d
P	$\frac{T}{L}$	∞	0
PI	$0{,}9\frac{T}{L}$	$\frac{L}{0{,}3}$	0
PID	$1{,}2\frac{T}{L}$	$2L$	$0{,}5L$

Cuadro 4.1: Regla de sintonía de Ziegler-Nichols basada en la respuesta escalón de la planta (primer método).

salida no presenta oscilaciones sostenidas para cualquier valor que pueda tomar K_p, entonces este método no se puede aplicar). Así, la ganancia crítica K_{cr} y el periodo P_{cr} correspondiente se determinan experimentalmente (Figura 4.1.4). Ziegler-Nichols sugirieron que se establecieran los valores de los parámetros K_p, T_i *y* T_d de acuerdo con la fórmula que se muestra en la Tabla 4.2.

Ziegler y Nichols sugirieron establecer los valores de K_p, T_i *y* T_d de acuerdo con la fórmula que se muestra en la Tabla 4.1.

Las reglas de sintonía de Ziegler-Nichols se han usado ampliamente para sintonizar controladores PID en sistemas de control de procesos en los que no se conoce con precisión la dinámica de la planta. Durante muchos años tales reglas de sintonía han demostrado ser muy útiles. Por supuesto, las reglas de sintonía de Ziegler-Nichols se pueden aplicar a plantas cuya dinámica se conoce. En estos casos, hay disponibles muchos métodos analíticos y gráficos para el diseño de controladores PID además de las reglas de sintonía de Ziegler-Nichols.

Por último remarcar que un controlador se va a implementar normalmente como un sistema muestreado, el controlador no tiene una recepción continua de la señal

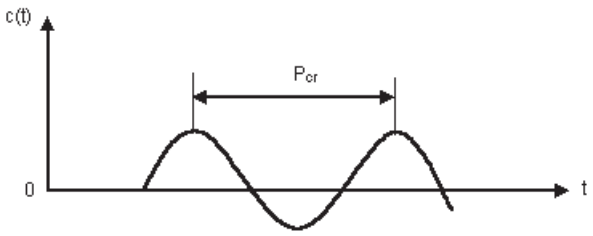


Figura 4.1.4: Oscilación sostenida con periodo P_{cr}.

Tipo de controlador	K_p	T_i	T_d
P	$0{,}5K_{cr}$	∞	0
PI	$0{,}45K_{cr}$	$\frac{1}{1{,}2}P_{cr}$	0
PID	$0{,}6K_{cr}$	$0{,}5P_{cr}$	$0{,}125P_{cr}$

Cuadro 4.2: Regla de sintonía de Ziegler-Nichols basada en la ganancia critica K_{cr} y periodo critico P_{cr} (Segundo método).

del sistema, ese tiempo de recepción variará en función de la planta con la que se trabaje.

4.2. Simulink PID Controller Blocks

Aparece en las últimas versiones de Simulink (dentro de Simulink Control Design). Esta herramienta nos va a dar una interfaz gráfica de usuario que nos va a facilitar realizar los cambios de estructura del controlador e incluso añadir elementos más avanzados a nuestro controlador. Se puede lograr una buena configuración para el seguimiento de referencia y para el rechazo de perturbaciones. El bloque contiene un regulador PID estándar en un bucle de realimentación y añade un pre-filtro para la señal de referencia. El pre-Filtro ayuda a producir una respuesta más suave transitoria a los cambios del punto de referencia.

Para poder utilizar el bloque abrimos la biblioteca de bloques de Simulink y escogemos un bloque PID como se puede observar en la Figura 4.2.1. Aparecen dos bloques de controlador PID, uno con una sola entrada que sería el error, por lo que la diferencia entre la señal a controlar y la referencia habría que efectuarla previamente y otro bloque donde aparecen dos entradas, una para la referencia y otra para la señal a controlar por lo que no sería necesario efectuar ninguna operación previa a dicho bloque. Esa es la única diferencia entre ambos bloques.

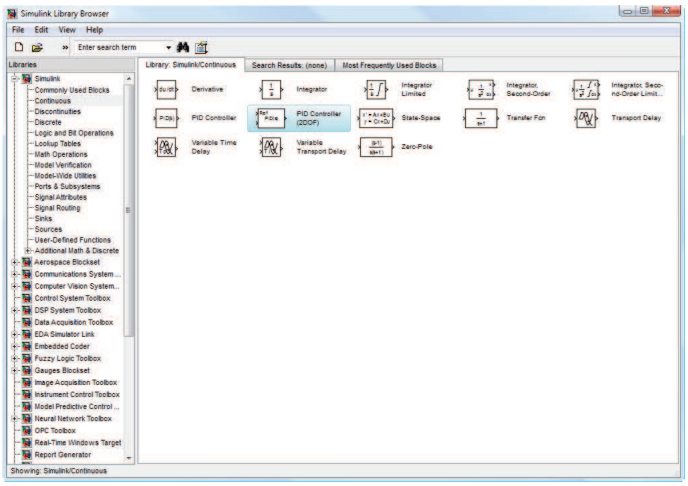


Figura 4.2.1: Biblioteca de bloques de Simulink.

El problema es encontrar el conjunto de valores que cumplan con los requisitos. Realizarlo manualmente es engorroso, puesto que hay que hacer numerosas pruebas para conseguir valores óptimos. Las reglas existentes como Ziegler-Nichols no son válidas para todos los sistemas.

El flujo de trabajo para el diseño típico con el sintonizador PID implica las tareas siguientes:

1. Iniciar el sintonizador de PID. Al iniciarse, el software calcula automáticamente un modelo de planta lineal del modelo de Simulink y diseña un controlador inicial.
2. Se sintoniza el controlador PID ajustando manualmente en función de los criterios de diseño. El sintonizador calcula los parámetros del PID que estabilizan robustamente al sistema.
3. Se exportan los parámetros del controlador diseñado al bloque del controlador PID y se verifica el rendimiento del controlador en Simulink.

Haciendo doble clic sobre el bloque PID aparece la siguiente imagen:

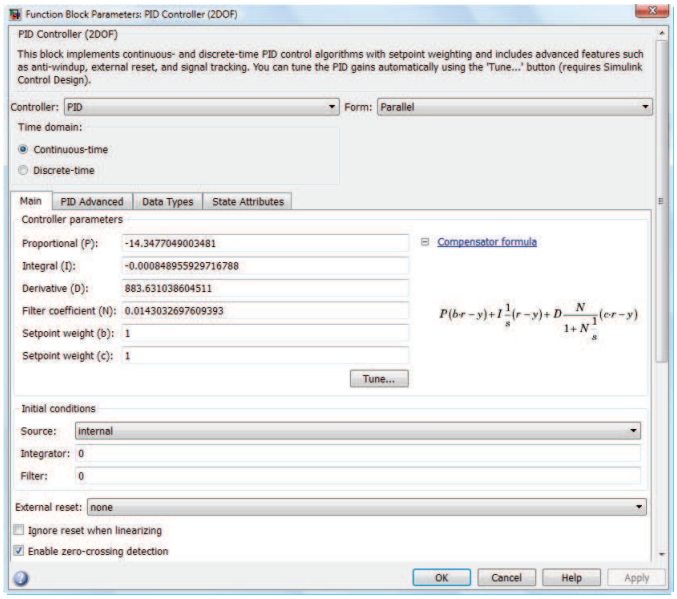


Figura 4.2.2: Parámetros del bloque de Simulink: Controlador PID

Donde encontramos los parámetros de configuración del PID, como la forma del controlador paralelo o ideal, los términos proporcional, integral y derivativo, etc. No hay que efectuar cambios en el modelo al variar la estructura del controlador, estos se hacen automáticamente. Si miramos debajo de la máscara del bloque, ésta irá cambiando automáticamente en función de los parámetros que se configuren.

En los parámetros avanzados podemos limitar la salida, por ejemplo entre 0 y 100, también nos permite configurar algún método *anti-windup*.

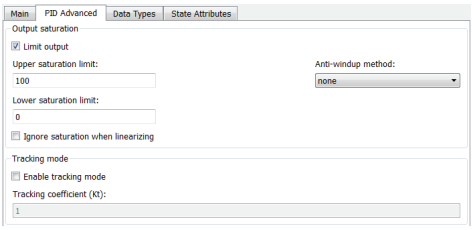


Figura 4.2.3: Parámetros avanzados del bloque PID

Como ya se ha comentado, el principal problema a la hora de diseñar un controlador PID es su sintonización y cómo ya hemos visto existen métodos para la sintonización que nos dan valores aproximados que en muchas ocasiones no son óptimos. Para ayudarnos a la sintonización, dentro del bloque PID, existe un apartado de autosintonización, donde además podremos sintonizar de manera manual los valores que queramos.

Al pulsar el botón de sintonización estamos diciéndole a Simulink que nos ajuste nuestro modelo.

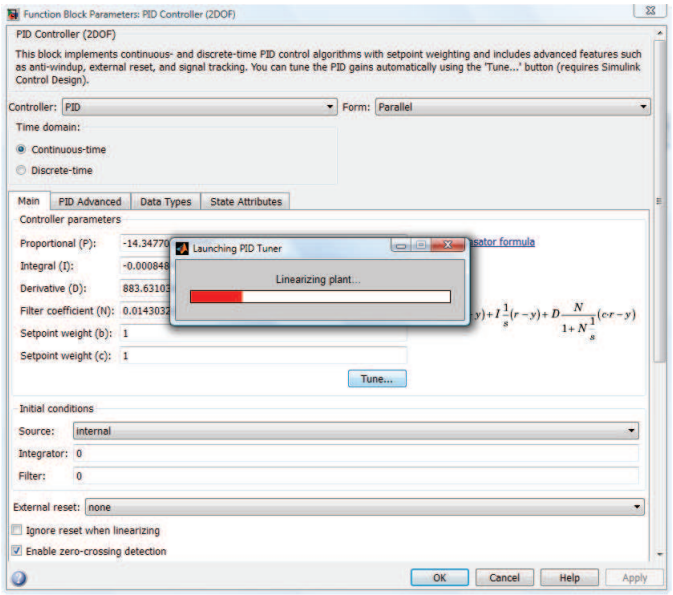


Figura 4.2.4: Linealización de la planta.

Cuando se pulsa la sintonización, el software calcula un modelo de planta lineal. El software identifica automáticamente la entrada y la salida de la planta, y se utiliza el punto de funcionamiento actual de la linealización. La planta puede tener cualquier orden y puede tener retardos de tiempo. El sintonizador PID calcula un controlador PI inicial para lograr un equilibrio razonable entre robustez y rendimiento. Por defecto, la referencia se muestra en el diagrama.

La siguiente figura muestra la respuesta que nos daría el sistema donde podemos ajustar el tiempo de respuesta así como el margen de fase.

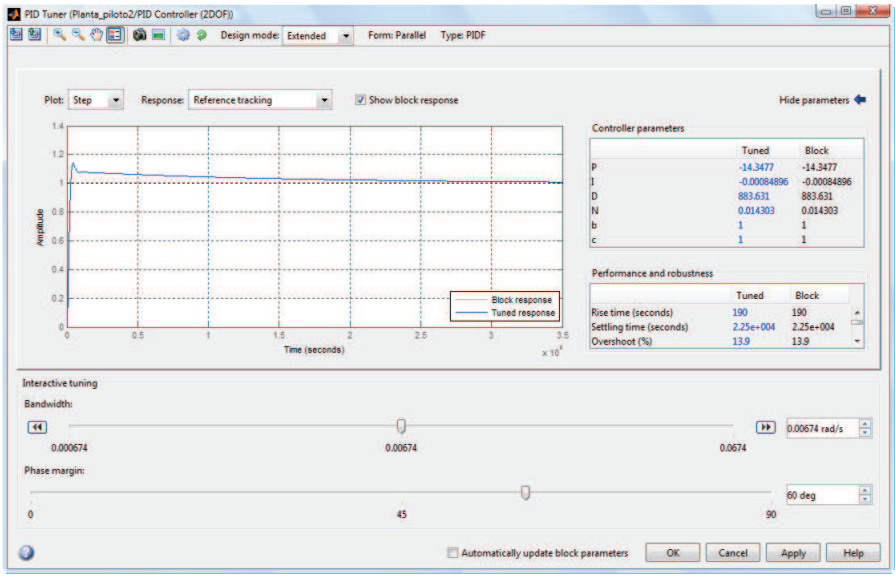


Figura 4.2.5: Ventana de sintonización del PID

Aplicándolo a nuestro sistema nos mostrará los valores de los términos proporcional, integral y derivativo.

En la próxima sección aplicaremos estas técnicas de sintonización para el modelo de la planta. Este bloque nos permitirá tanto sintonizar los controladores PID como verificar su funcionamiento por lo que son muy útiles a la hora de diseñar estrategias de control clásicas.

4.3. Diseño del controlador PID en Simulink para el modelo de la planta

Para realizar una primera sintonización del controlador clásico para el modelo de la planta utilizaremos la función de transferencia de primer orden más retraso aproximada que se ha visto en el capítulo anterior.

$$G(s) = \frac{-0{,}975}{950s + 1} e^{-31{,}25s}$$

Utilizaremos las reglas de sintonización de Ziegler-Nichols para diseñar un controlador PID. Para identificar los parámetros necesarios aplicamos un escalón unitario a la entrada del sistema para obtener su respuesta. Los parámetros que se obtuvieron se muestra en la Tabla 4.3.

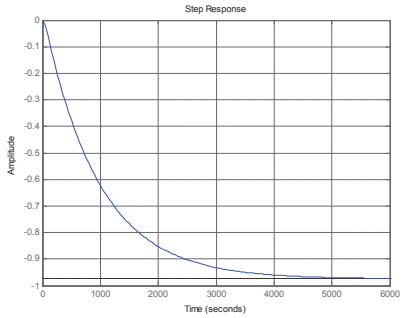


Figura 4.3.1: Respuesta ante una entrada escalón unitario del sistema de primer orden.

	K_p	T_i	T_d
PID	$1{,}2\frac{T}{L} = -37{,}415$	$2L = 62{,}5\,s$	$0{,}5L = 15{,}625\,s$

Cuadro 4.3: Parámetros Ziegler-Nichols PID.

Para poder probar el funcionamiento del controlador PID, se diseñará un controlador en Simulink. Utilizaremos el modelo de la planta que se vio en el capítulo anterior y se le añadirá un bloque PID (2DOF), tal y como se muestra en la Figura 4.3.2.

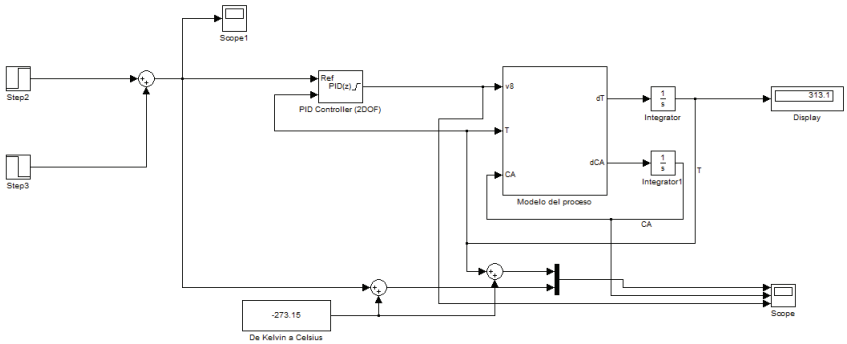


Figura 4.3.2: Modelo de la planta con un controlador PID.

Únicamente tendremos que abrir el bloque PID de Simulink haciendo "doble clic" sobre el bloque y configurar los parámetros del PID obtenidos por el método de Ziegler-Nichols. Emplearemos la forma ideal de la fórmula del PID. Además para que sea lo más parecido posible a la realidad se limitará la salida del PID en el menú de opciones avanzadas entre los valores 0 y 100, que se corresponderán a la mínima y máxima apertura de la válvula.

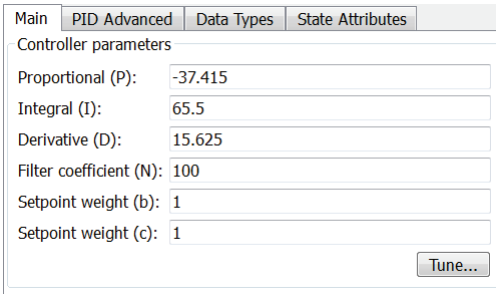


Figura 4.3.3: Pantalla de configuración de los parámetros del PID.

Puesto que el método de Ziegler-Nichols no nos proporciona un ajuste muy fino, más bien nos da unos valores para empezar y ajustarlos en planta, vamos a aprovechar la herramienta de autosintonización que trae Simulink para obtener unos valores más cercanos a los óptimos.

Pulsamos el botón de sintonización (*tune*) del bloque y aplicamos los parámetros que nos proporciona (ver Tabla 4.4). Simularemos 6000 segundos. La respuesta que obtenemos a un cambio de la referencia es la que se muestra en la Figura 4.3.4.

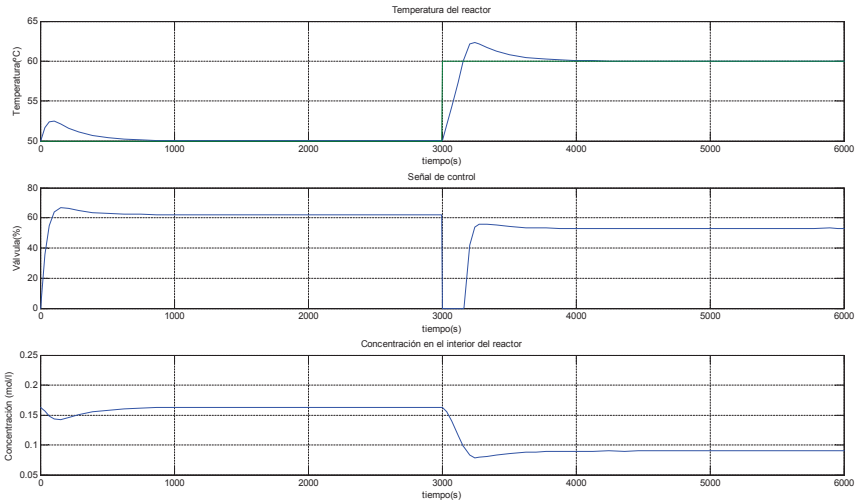


Figura 4.3.4: Resultado de utilizar la autosintonización de Simulink.

	K_p	T_i	T_d
PID	$-19{,}957$	$0{,}00408\,s$	$0\,s$

Cuadro 4.4: Parámetros Ziegler-Nichols PID.

Simulink nos proporciona los parámetros para un PI. Si queremos podemos modificar el tiempo de respuesta del sistema para hacerlo más rápido o lento, según queramos.

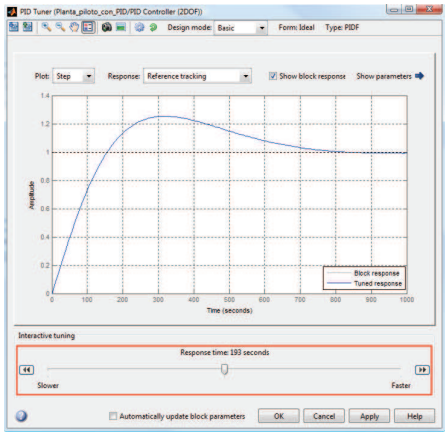


Figura 4.3.5: Pantalla para modificar el tiempo de respuesta.

Modificamos el tiempo de respuesta disminuyéndolo, obteniendo la respuesta de la Figura 4.3.6.

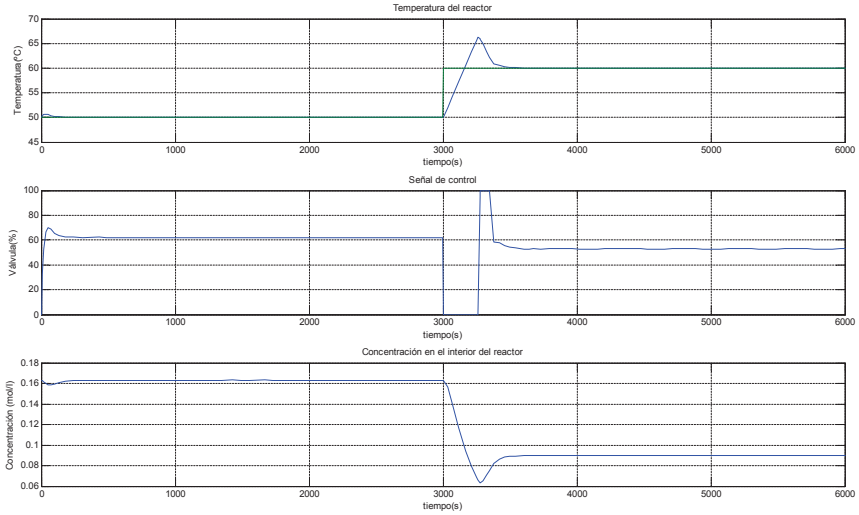


Figura 4.3.6: Respuesta del sistema tras modificar el tiempo de respuesta.

Vemos en la Figura 4.3.7 como varía la respuesta del sistema al aumentar y disminuir el tiempo de respuesta. Si aumentamos el tiempo de respuesta, conseguimos que la señal alcance la referencia con menos sobreoscilación pero tarda más tiempo, mientras que si disminuimos el tiempo de respuesta, la señal alcanza la referencia ante pero con más sobreoscilación.

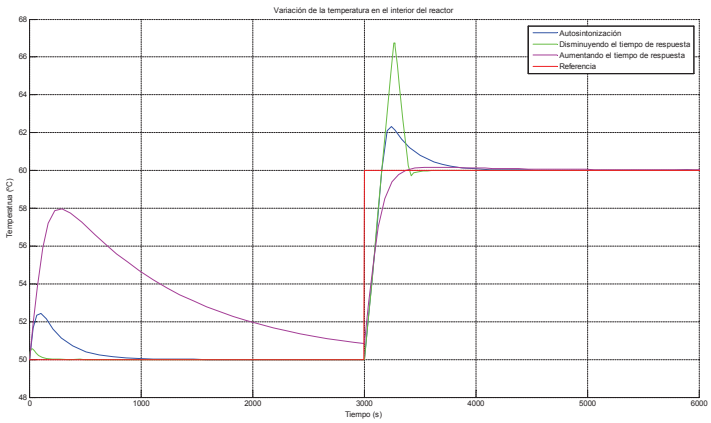


Figura 4.3.7: Distintas respuestas variando el tiempo de respuesta dado por la autosintonización.

Para la implementación en un sistema real, sabemos que será un sistema que muestreado. Se ha diseñado el PID como una función de transferencia continua pero la implementación será en un sistema que se ejecutara cada cierto tiempo y además ese procesador tampoco se va a ejecutar con un número con coma fija (no flotante). Para ajustar el controlador hay que realizar una conversión a tiempo discreto. Para realizar la conversión del controlador a tiempo discreto será necesario:

- Conversión a tiempo discreto.
- Escalado de coma fija.
- Implementación en código de ese controlador.

Sin embargo, mediante la sintonización del PID con Simulink podremos realizar todos estos ajustes automáticamente. Seleccionamos el dominio del tiempo discreto y escribimos el tiempo de muestreo que creamos oportuno.

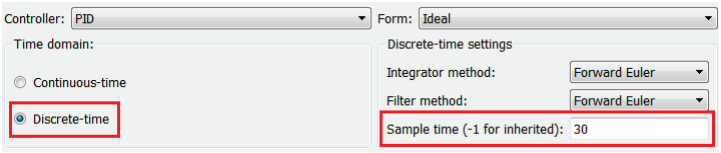


Figura 4.3.8: Opciones para discretizar el controlador.

4.3.1. Simulaciones

Hasta ahora los ejemplos mostrados han sido en el mismo punto de trabajo. En esta sección se mostrarás los resultados obtenidos al variar el punto de trabajo con los mismos parámetros del controlador PID. Las simulaciones comenzarán con una referencia de 49ºC que a los 25 minutos cambia a 60ºC y por último a los 80 minutos vuelve a camiar hasta los 40ºC. El tiempo total de las simulaciones es de 150 minutos. Los parámetros seleccionados se han elegido de los métodos de sintonización vistos en el apartado anterior.

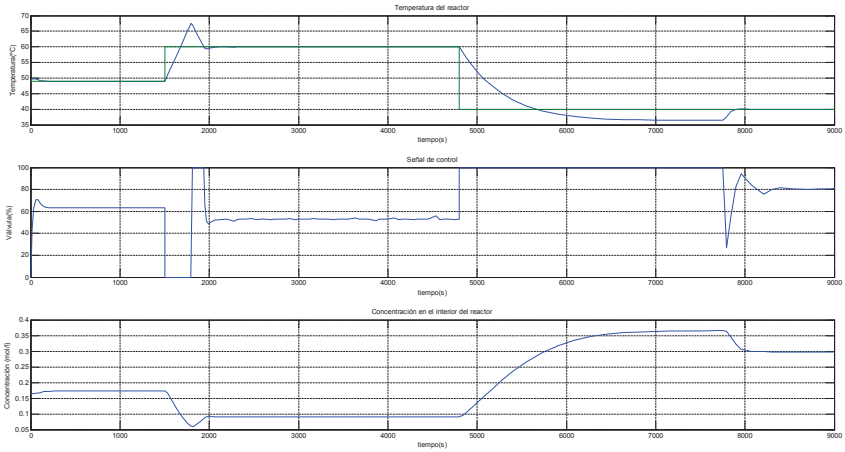


Figura 4.3.9: Resultados simulación 1.

	$D = K_p$	$I = \frac{1}{T_i}$	$D = T_d$
PID	-58.98	0.016	0

Cuadro 4.5: Parámetros del controlador PID en Simulink. Simulación 1.

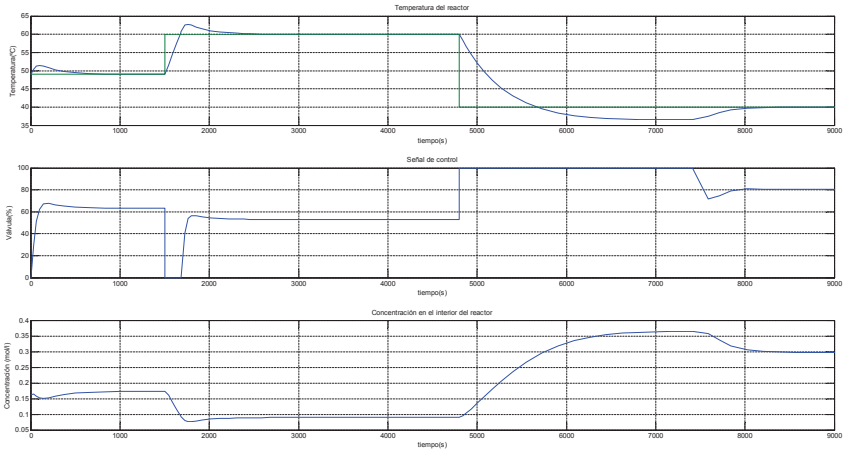


Figura 4.3.10: Resultados simulación 2.

	$D = K_p$	$I = \frac{1}{T_i}$	$D = T_d$
PID	-19.957	0.004	0

Cuadro 4.6: Parámetros del controlador PID en Simulink. Simulación 2.

Como era de esperar, debido a la no linealidad del sistema, el comportamiento del controlador PID en diferentes puntos de trabajo varía. La ganancia que tiene

el sistema cuando la temperatura crece es muy diferente a la ganancia del sistema cuando ésta desciende. No es posible sintonizar un controlador PID o en nuestro caso PI que funcione adecuadamente en todos los puntos de trabajo posibles.

4.4. Implementación del controlador PID en Unity

En esta sección se utilizarán los datos obtenidos para la sintonización de un controlador PID de la sección anterior para la implementación de un controlador PID en el PLC industrial de la planta piloto.

4.4.1. Introducción al software Unity Pro

Uno de los objetivos de este proyecto es implementar controladores en sistemas reales. Puesto que la planta piloto cuenta con un autómata programable Modicon M340 se utilizará el software Unity Pro de *Schneider Electric*. Es el software común de programación, depuración y mantenimiento para las gamas de PLCs Modicon M340, Atrium, Premium y Quantum, de fácil uso y que cumple con la norma IEC-1131, lo que nos permitirá ahorrarnos tiempo de desarrollo y aumentando la calidad del trabajo. La configuración de los distintos módulos del PLC de la planta se puede ver en [5]. Utilizaremos este software para programar los distintos controladores.

Unity nos facilita un conjunto bastante completo de funcionalidades de herramientas, permitiendo máxima flexibilidad y facilidad a la hora de diseñar una aplicación.

Para poder trabajar con el software Unity y el modelo hecho en Simulink habrá que tener en cuenta una serie de pasos que hay que seguir para que la plataforma Unity-Simulink vía OPC funcione correctamente.

Para empezar habría que tener configurados todos los módulos del bastidor del PLC de la planta. Como ya se ha indicado, esta configuración se realiza en [5].

Puesto que la ventaja principal de utilizar la plataforma Unity-Simulink es no tener que estar físicamente con la planta piloto, hay que elegir la modalidad de funcionamiento: estándar o en simulación. En la primera, el software Unity se conectará con el PLC y habrá que configurar la manera de establecer la conexión. El PLC de la planta está conectado vía *ethernet* al ordenador por lo que conectará con la IP del autómata que haya configurado en las opciones de dirección del PLC. En el segundo modo, el software se conectará con el simulador que trae y asignará al simulador un IP en función de la IP del equipo en el que ejecute, es decir, si por ejemplo no estamos conectados a ninguna red, tomará por defecto la IP 127.0.0.1, sin embargo, si estamos conectados a una red local tomará la IP que la red local haya asignado al equipo. Este dato es importante ya que para establecer la conexión por OPC será necesario conocer la IP del PLC, bien la real del autómata o la del simulador.

Aunque la mayoría de las pruebas se desarrollaron bajo el sistema operativo Windows Vista de 32 bits con el Unity Pro XL 4.0 sin ningún tipo de problemas, se ha detectado una incompatibilidad entre Windows 7 de 64 bits y la misma versión de Unity. Al intentar conectar el modo simulación del Unity aparece el mensaje de la Figura 4.4.1.

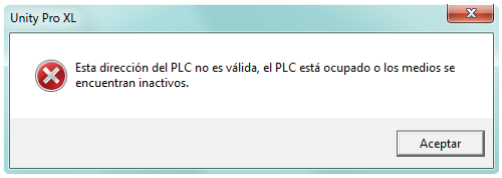


Figura 4.4.1: Pantalla de error al conectar el simulador de Unity Pro.

Mostrando además el panel del simulador del PLC como aparece en la Figura 4.4.2.

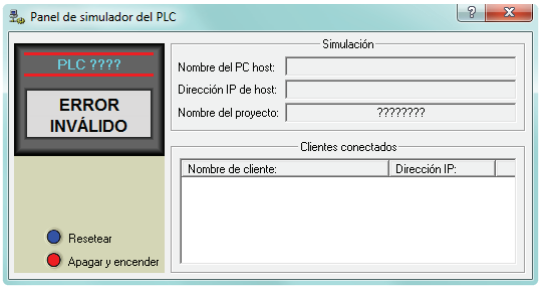


Figura 4.4.2: Panel de simulador del PLC cuando se produce el error.

El error impide al simulador tomar una IP que imposibilita la conexión de Unity con el mismo. Este error es producido por una incompatibilidad entre el sistema Aero de Windows 7 y Unity, para solventarlo basta con desactivar el sistema Aero en el menú de Personalizar, pulsando click derecho en el escritorio, y elegir entre todos los temas "Windows 7 Basic". De este modo se soluciona el problema y el simulador funciona correctamente.

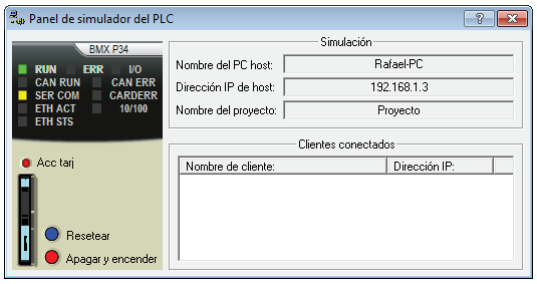


Figura 4.4.3: Panel de simulador del PLC una vez cargado un proyecto.

4.4.1.1. Programación de la aplicación

Para programar la aplicación que se encargará de controlar la planta en Unity se han utilizado tres de los cinco lenguajes de programación disponibles y que cumplen con la norma IEC 1131-3. Se ha utilizado SFC para la estructura del programa, LD para todas las transiciones y algunas secciones y FBD para la programación de los controladores. El funcionamiento básico de la aplicación es inicializar las variables que sean necesarias, esperar el inicio de la simulación en MATLAB, elección y modo de funcionamiento del controlador, y por último paro de la simulación. Se ha desarrollado esta aplicación para poder aplicar las estrategias de control clásicas que hemos diseñado para Simulink pero aplicándolas en un autómata programable industrial, lo que nos proporcionará mejores datos de cómo se comportará el controlador en la realidad.

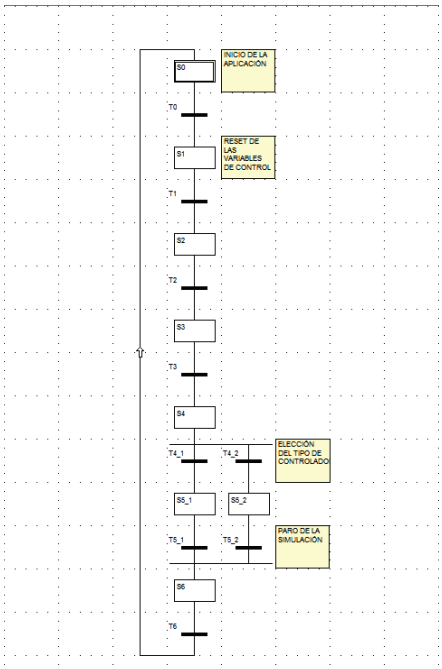
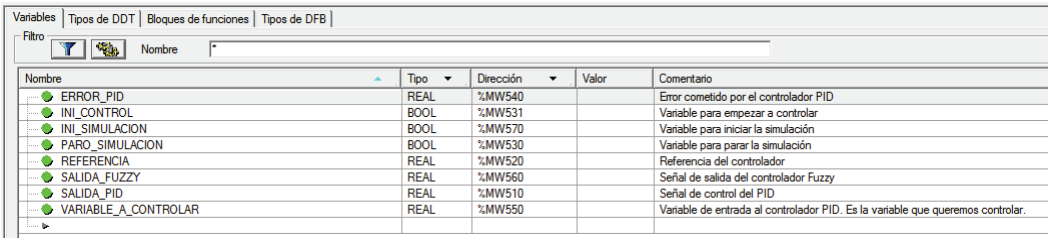


Figura 4.4.4: Estructura de la aplicación en SFC.

Puesto que el sistema a controlar será el modelo de la planta de Simulink que conectaremos con el Unity vía OPC, debemos configurar las variables que utilicemos en ambos programas. Tendremos que guardarlas en una dirección de memoria %MW fija, tal y como se ve en la Figura 4.4.5. Estas direcciones debemos configurarlas manualmente y son muy importantes para el funcionamiento de la plataforma Unity-Matlab. Para conocer más detalles acerca de como se realiza la conexión por OPC consultar el capítulo dedicado a ello.



Nombre	Tipo	Dirección	Valor	Comentario
ERROR_PID	REAL	%MW540		Error cometido por el controlador PID
INI_CONTROL	BOOL	%MW531		Variable para empezar a controlar
INI_SIMULACION	BOOL	%MW570		Variable para iniciar la simulación
PARO_SIMULACION	BOOL	%MW530		Variable para parar la simulación
REFERENCIA	REAL	%MW520		Referencia del controlador
SALIDA_FUZZY	REAL	%MW560		Señal de salida del controlador Fuzzy
SALIDA_PID	REAL	%MW510		Señal de control del PID
VARIABLE_A_CONTROLAR	REAL	%MW550		Variable de entrada al controlador PID. Es la variable que queremos controlar.

Figura 4.4.5: Variables con dirección de memoria %MW fija.

Aunque la aplicación para el control está orientada a la planta piloto puede ser utilizada con cualquier proceso ajustando los parámetros pertinentes. De esta forma se ha conseguido una herramienta muy útil para poder comprobar la eficiencia de las distintas estrategias de control implementadas en un autómata programable.

Como este proyecto se centra en los controladores aplicados en autómatas industriales, nos centraremos en la programación en Unity de los distintos controladores, si se quiere conocer más detalles sobre la programación de la aplicación completa en los anexos se encontrará toda la documentación técnica del programa.

4.4.2. Controlador PID

La mayoría de controladores aplicados a la planta hasta la fecha se han implementado en MATLAB, por eso, en este proyecto queremos centrarnos en la implementación de controladores en sistemas reales como el PLC de la planta.

Unity cuenta con numerosas librerías con multitud de funciones, entre ellas, una función para un controlador PID la PIDFF. Para seleccionar este bloque basta con ir a Herramientas > Explorador de librería de tipos y en el menú del explorador seleccionar en nombre de biblioteca: CONT_CTL.

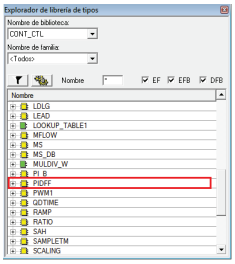


Figura 4.4.6: Menú explorador de biblioteca de tipos de Unity.

El bloque de función PIDFF se basa en el algoritmo de un PID con estructura paralela o mixta. Este bloque cuenta con numerosas funciones:

- Cálculo del componente proporcional, diferencial e incremental.
- Valor real, de consigna y de salida en unidades físicas.
- Acción directa o inversa.

- Componente diferencial para procesar valores o desviaciones.
- Área muerta en desviación.
- Límite superior e inferior de la señal de salida.
- Selección de modalidad manual/automática.
- Modalidad de seguimiento.
- Límite de consigna superior e inferior.

Muchas de estas funciones no se utilizarán en este proyecto pues están orientadas a procesos reales que ocurren en la industria. La industria en un momento determinado se dio cuenta que necesitaba cierto tipo de funciones para ajustar mejor sus controladores y Unity englobó todas estas necesidades en un único bloque. De esta forma con este bloque es posible controlar un amplio abanico de procesos industriales mediante la configuración adecuada de sus funciones.

Para programar el controlador PID se ha elegido el lenguaje FBD para el bloque PIDFF.

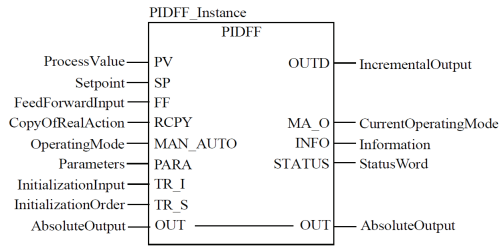


Figura 4.4.7: Representación del bloque PIDFF en FBD.

El bloque cuenta con 8 entradas y 5 salidas. De las entradas PV es el valor del proceso que queremos controlar, SP es la consigna a seguir, es decir la referencia, FF es la entrada de interrupción que no utilizaremos al igual que RCPY. MAN_AUTO es un bit para configurar el bloque en modo automático ("1") o manual ("0"). PARA es una estructura de datos con los parámetros de configuración del bloque PIDFF, esta estructura la veremos más adelante. La entrada TR_I permite introducir un valor inicial al bloque y según en el modo en el que estemos pues lo considerará o lo ignorará. Por último TR_S es el comando de inicialización.

Como salidas tenemos OUT que es la salida del PID, la señal de actuación, mientras que OUTD es el valor incremental. MA_O muestra el modo de funcionamiento en el que trabaja el controlador, mientras que INFO y STATUS muestran información del bloque.

Como ya se ha indicado la estructura de datos PARA permite configurar el bloque por lo que será de vital importancia realizar la configuración correctamente. En la Tabla 4.7 aparece la configuración básica de la estructura que seguiremos a lo largo de todo el proyecto.

Elemento	Valor	Descripción del valor seleccionado
id	automático	Reservado para el ajuste automático.
pv_inf	0	Valor mínimo que el proceso puede tomar.
pv_sup	1000	Valor máximo que el proceso puede tomar.
out_inf	0	Valor mínimo de la salida del PID
out_sup	100	Valor máximo de la salida del PID
rev_dir	1	Acción opuesta del controlador PID
mix_par	0	Controlador PID con estructura mixta
aw_type	0	Sin *anti-windup*
en_rcpy	0	No se utiliza la entrada RCPY
kp	-20	Ganancia proporcional
ti	t#1s[1]	Tiempo integral
td	t#0s	Tiempo derivativo
kd	1	Ganancia diferencial
pv_dev	1	Contribución diferencial en relación al error
bump	1	Modo anti-brusquedad activado
dband	0	Sin área muerta en desviación
gain_kp	1	Reducción de la contribución proporcional
ovs_att	0	Modifica el peso del SP
outbias	0	Introduce un offset a la salida del PID
out_min	0	Límite inferior de la salida
out_max	100	Límite superior de la salida
outrate	100	Límite de modificación de salida por seg.
ff_inf	0	Límite inferior del rango FF
ff_sup	0	Límite superior del rango FF
otff_inf	0	Límite inferior del rango out_ff
otff_sup	0	Límite superior del rango out_ff

Cuadro 4.7: Descripción y valores del parámetro Para_PIDFF.

Como ya se ha comentado, muchos de estos parámetros no se utilizarán durante este proyecto ya que no son necesarios. A continuación se realiza una explicación mas detallada de alguno de ellos:

- dband: define un área en el que, por ejemplo, si hay ruido en esa banda, no se actuará o la actuación sera menor, es decir, si hay ruido evita que se esté actuando continuamente por culpa del ruido. Puesto que se estará trabajando en simulación no será necesario utilizar este parámetro
- ovs_att: modifica el peso de la referencia en el control incremental.
- outbias: introduce un *offset* en la salida del PID, puede utilizarse en la industria para compensar algún actuador (con conocimiento previo).
- outrate: me permite fijar cuánto varia la señal del actuador en un segundo. Por ejemplo, porque el proceso lo requiera o porque el actuador tenga una limitación.

En la Figura 4.4.8 aparece el diagrama de la estructura del bloque PIDFF, de esta forma se puede entender mejor su funcionamiento.

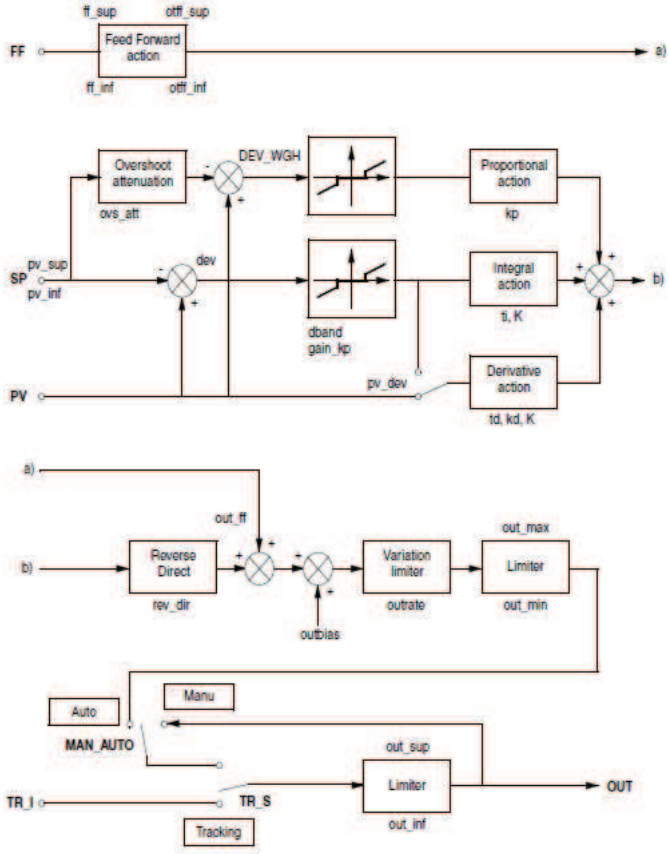


Figura 4.4.8: Diagrama de estructura de controlador PIDFF.

Una vez explicadas las características principales de este bloque PIDFF, se mostrará cómo se ha conectado y qué variables se han utilizado para ello.

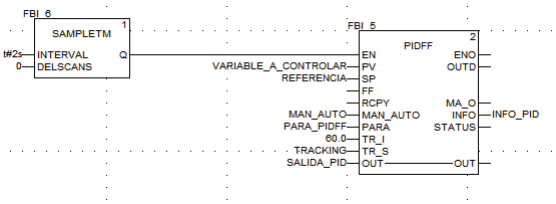


Figura 4.4.9: Bloque PIDFF activado cada 2s.

Nombre	Descripción
VARIABLE_A_CONTROLAR	A través de esta variable el bloque recibe la variable a controlar del proceso que se está ejecutando en Simulink.
REFERENCIA	A través de esta variable el bloque recibe la consigna marcada desde el propio Unity.
MAN_AUTO	Esta variable controla el modo automático y manual del bloque.
PARA_PIDFF	Variable con los parámetros del bloque.
TRACKING	Esta variable permitirá alternar el modo de seguimiento y el automático.
SALIDA_PID	A través de esta variable, la señal de actuación llega hasta el proceso de Simulink que se está ejecutando.
INFO_PID	Mediante esta variable obtenemos información del bloque como la desviación entre el PV y el SP.

Cuadro 4.8: Variables utilizadas en el bloque PIDFF.

La entrada EN (*enable*) del bloque PIDFF está conectado al bloque SAMPLETM que activa su salida según el tiempo que definamos en INTERVAL. Esto permite que el controlador no esté tomando valores y en consecuencia actuando todo el tiempo.

4.4.3. Simulaciones

4.4.3.1. Creación de un SCADA

Una vez explicado cómo se ha programado el bloque PIDFF pasaremos a realizar una serie de simulaciones que nos permitan comprobar el funcionamiento de las estrategias de control clásicas diseñadas. Para tal fin se ha creado un pequeño SCADA en el mismo Unity Pro. Unity Pro también permite crear pantallas para operadores para poder visualizar gráficos de tendencias y variables, de este modo podremos comprobar el funcionamiento de los controladores en tiempo real, sin necesidad de tener que esperar al final del proceso. Además permite la modificación de ciertos parámetros de configuración.

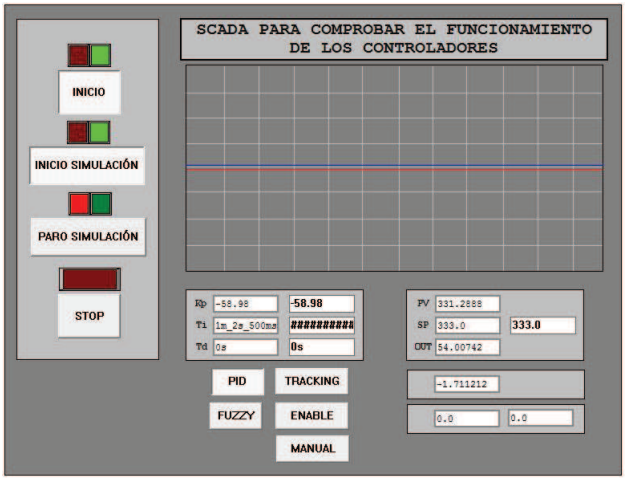


Figura 4.4.10: Imagen del SCADA diseñado.

El SCADA cuenta con una pantalla central donde se muestran las tendencias de la referencia y de la señal, un panel de control a la izquierda para iniciar la aplicación y el inicio/paro de la simulación, varios recuadros que muestran distintos valores de variables y una serie de botones que permiten activar un controlador u otro.

El proceso para crear un SCADA en Unity es muy sencillo, basta con ir al explorador de proyectos y seleccionar "Pantallas de operador", luego pulsar el botón derecho del ratón y seleccionar "Nueva pantalla". Nos aparecerá una pantalla para rellenar las propiedades del SCADA, como el nombre, comentarios y dimensiones.

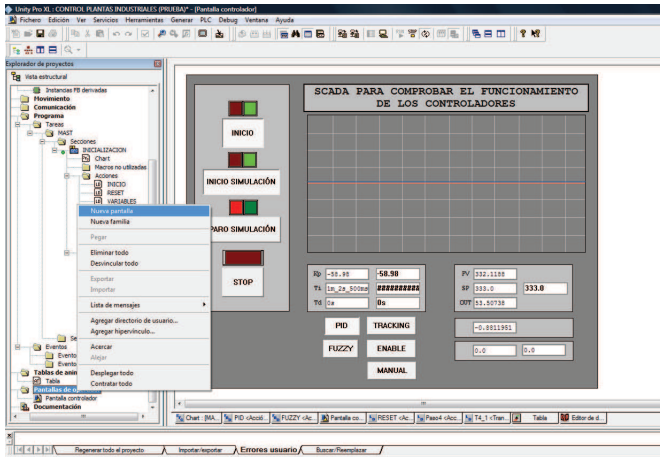


Figura 4.4.11: Creación de un SCADA en Unity.

Luego tendremos que ir eligiendo los elementos que queramos incluir en nuestro SCADA de la barra superior de herramientas y configurarlos de manera apropiada. Variable con la que está relacionada, animación a realizar, colores, etc.

4.4.4. Manual de uso de la plataforma Unity-Matlab

Se explicará los pasos que hay que seguir para poner en marcha una simulación mediante la plataforma Unity-Matlab vía OPC, donde estaremos controlando a través de la aplicación hecha para el PLC el modelo de la planta piloto diseñado en Simulik, todo ello conectado a través del estándar de comunicaciones OPC. Se recomienda leer antes el capítulo Conexión por OPC para tener configurado todo correctamente.

Para empezar hay que regenerar todo el proyecto para comprobar que no hay ningún tipo de error. Una vez hecho esto conectamos el Unity al PLC, bien a un PLC real (modo estándar) o en simulación. Todos las simulaciones que se realizaron para este proyecto fueron hechas en el modo simulación.

Figura 4.4.12: Selección del modo de conexión: estándar o simulador.

Una vez se ha conectado al simulador (aparecerá un icono en la parte derecha de la barra de inicio de Windows), transferimos el programa y pulsamos RUN en la barra de herramientas superior.

Para facilitar el uso del programa se recomienda utilizar el SCADA, aunque también se puede utilizar modificando las diferentes variables creando una tabla de animación.

Hay que pulsar el botón INICIO del SCADA para que arranque el programa, éste quedará a la espera de pulsar el botón INICIO SIMULACIÓN. Pero antes de pulsarlo hay que ejecutar el programa en MATLAB que establecerá la conexión por OPC, se encargará de actualizar variables y de ejecutar el modelo de Simulink. El nombre de este programa es **Conexion_Unity_Matlab.m**, este programa habrá que ejecutarlo en MATLAB. La explicación y programación de este programa se expone en el capítulo titulado Conexión por OPC.

Una vez se está ejecutando el programa de conexión de MATLAB aparecerá una ventana del OFS Factory Server, muestra de que la conexión vía OPC se ha establecido correctamente. Pulsaremos el botón INICIO SIMULACIÓN (sin cerrar MATLAB) y el programa del PLC pasará al siguiente estado en el que se requiere la elección de un tipo de controlador. Puesto que en este capítulo estamos viendo controladores clásicos pulsaremos el botón PID para seleccionar el controlador PID. Podremos alternar entre el modo manual y el modo de seguimiento. Una vez hecho todo esto, visualizaremos las señales de referencia y temperatura (en este caso) en la pantalla principal.

Se recomienda fijar los valores de los parámetros del PID al inicio del proceso así como esperar a estar en un punto de equilibrio antes de cambiar de referencia para no obtener datos erróneos.

Para parar la simulación hay que pulsar el botón PARO SIMULACIÓN. Una vez pulsado aparecerán varias gráficas de MATLAB mostrando el valor de la temperatura del reactor, concentración y señal de actuación que ha habido durante todo el tiempo de simulación.

4.4.5. Resultados de las simulaciones

Para la primera prueba que se ha realizado se han utilizado los valores de los parámetros de uno de los PI diseñados en apartados anteriores. Se estabiliza el sistema en un punto de equilibrio y una vez en él se cambia la referencia.

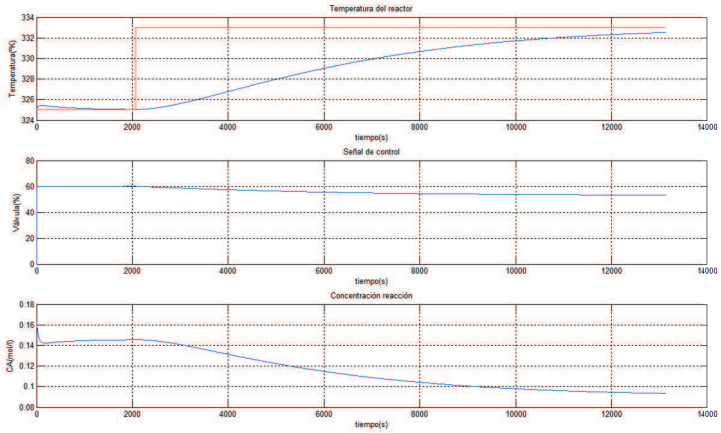


Figura 4.4.13: Prueba 1. $K_p = -58{,}98$ y $T_i = 62{,}25s$.

Después de más de 3 horas de simulación, no se ha alcanzado la referencia marcada, mientras que en la simulación de Simulink con los mismo valores la alcanzaba en poco tiempo. Esto puede deberse a que el PID implementado en el PLC es un sistema real y como tal tiene una serie de limitaciones. Para lograr un mejor ajuste se recurrirá al ajuste "en planta" para lograr una mejor respuesta.

En la siguiente prueba se ha reducido el tiempo de integración a una cuarta parte.

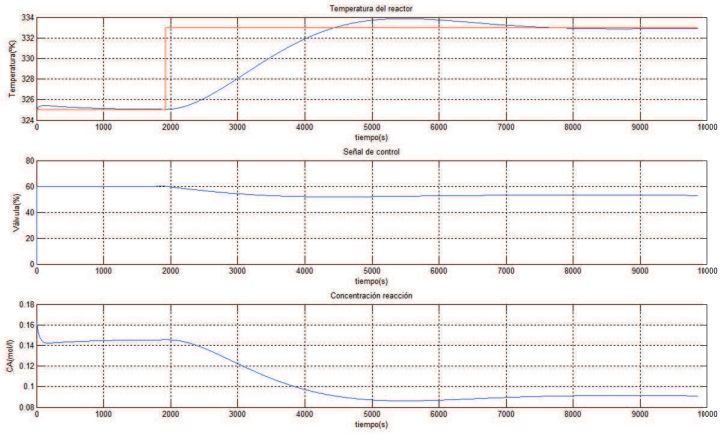


Figura 4.4.14: Prueba 2. $K_p = -58{,}98$ y $T_i = 15s$.

Ahora se alcanza la referencia en menor tiempo, aunque sobreoscila un poco. En función de lo que interese en el proceso se modificaran los parámetros. Si se quiere alcanzar la referencia en poco tiempo habrá que disminuir el tiempo integral pero aumentarán las sobreoscilaciones. Por el contrario, si se aumenta el tiempo integral las sobreoscilaciones desaparecerán pero se tardará más en alcanzar la referencia.

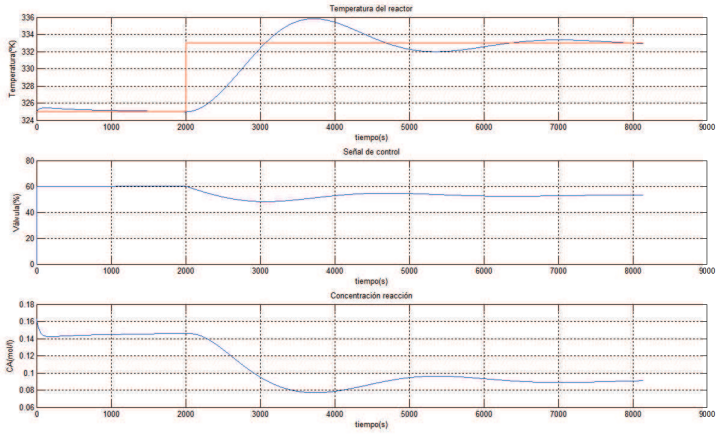


Figura 4.4.15: Prueba 3. $K_p = -58{,}98$ y $T_i = 5s$.

Si disminuimos demasiado el tiempo integral podemos volver inestable el sistema.

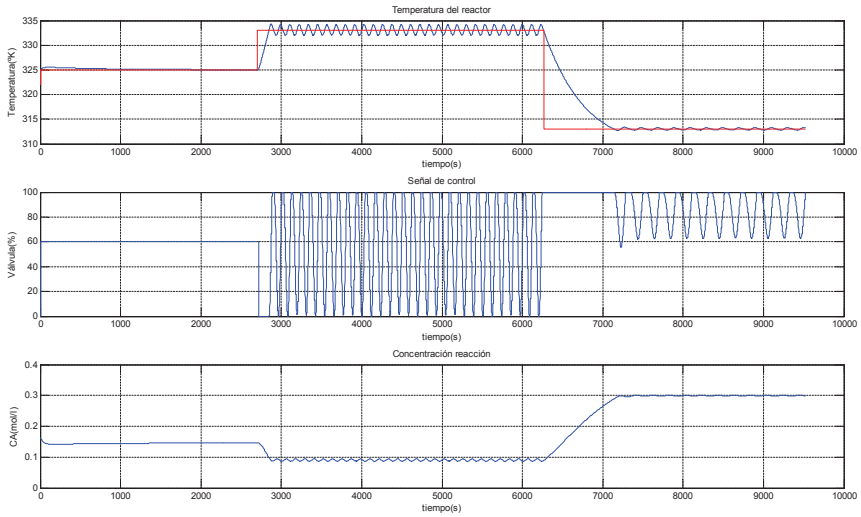


Figura 4.4.16: Respuesta del sistema en diferentes puntos de trabajo con:$K_p = -58{,}98$, $T_i = 4ms$, $T_d = 16s$ y $T_s = 2s$

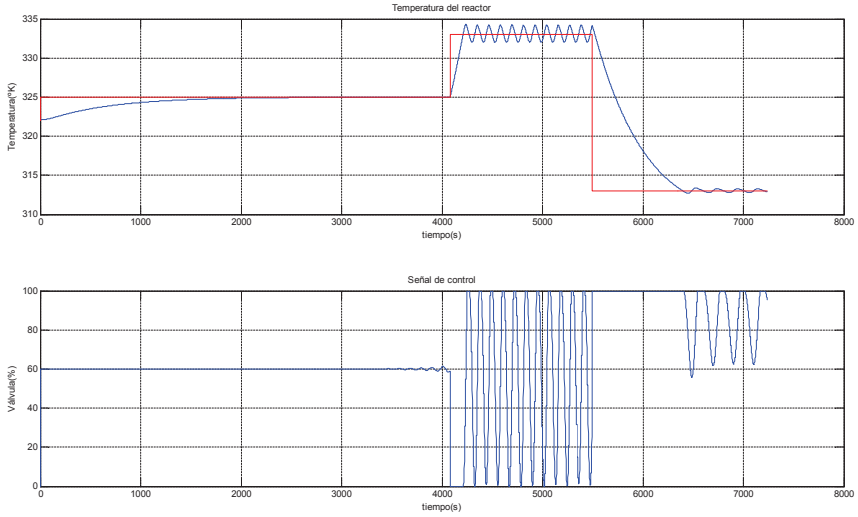


Figura 4.4.17: Respuesta del sistema en diferentes puntos de trabajo con:$K_p = -50$, $T_i = 4ms$ y $T_s = 2s$