Grado en Ingeniería Electrónica Industrial y Automática

Curso 2019/2020

CONTROL INTELIGENTE

Reporte 1

Control de un Motor Diesel con un sistema PID

Emanuel David Nuñez Sardinha

13 de Enero de 2020

Introducción

El objetivo de este proyecto es el ajuste de un controlador PID (Proporcional Integral Derivativo) para un motor diesel simulado. Nos concentramos en el proceso de diseño del controlador, influencia de parámetros y modificaciones para mejorar comportamiento en escenarios específicos.

Se realiza una presentación del problema, donde se explica el modelo, respuesta predeterminada y modificaciones hechas antes del proceso de optimización.

En el estudio previo, se propone un modelo inicial para nuestra función de costo, se justifica la decisión de no utilizar el componente diferencial del PID desde el punto de vista teórico y práctico, y se genera un conjunto de datos para entender intuitivamente e ilustrar los cálculos posteriores.

En optimización discutimos el proceso de desarrollo y configuración de los algoritmos utilizados: fmincon y Global Search. Posteriormente se ejecuta este algoritmo cambiando la función de costo para obtener otras respuestas.

Presentación del problema

El modelo base escogido es el modelo de motor diesel encontrado en los ejemplos de Matlab (*Sldemo_enginewc*). El modelo es un sistema no lineal en circuito cerrado, que toma como entrada la velocidad deseada, y computa una velocidad de motor a partir de múltiples factores, incluyendo torques de arrastre, combustión, limitaciones de las válvulas y otras variables físicas simuladas (Figura 1).

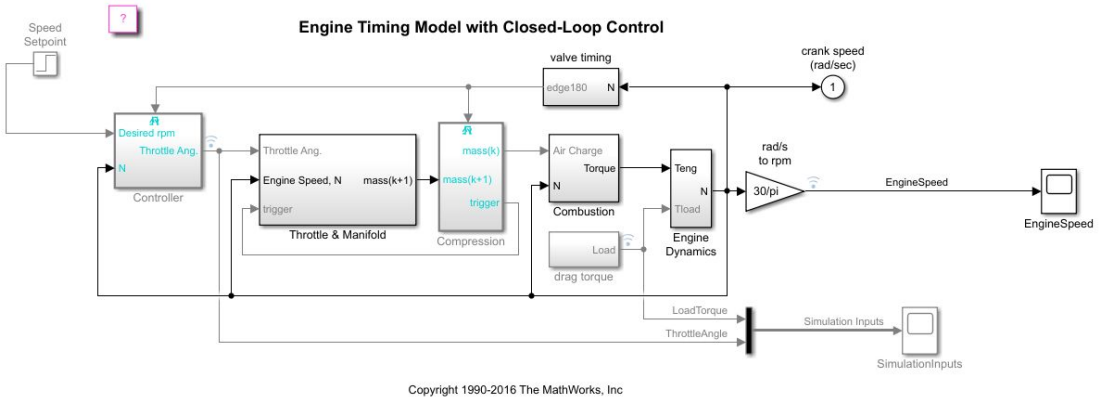

Figura 1: Modelo original de motor diesel

A diferencia de otros modelos, el sistema de control predeterminado del sistema incorpora un control discreto PI (Proporcional e Integral), sin un componente diferencial (Figura 2). Sin cambiar a otro controlador, solo se dispone de la constante integradora y proporcional para manipular.

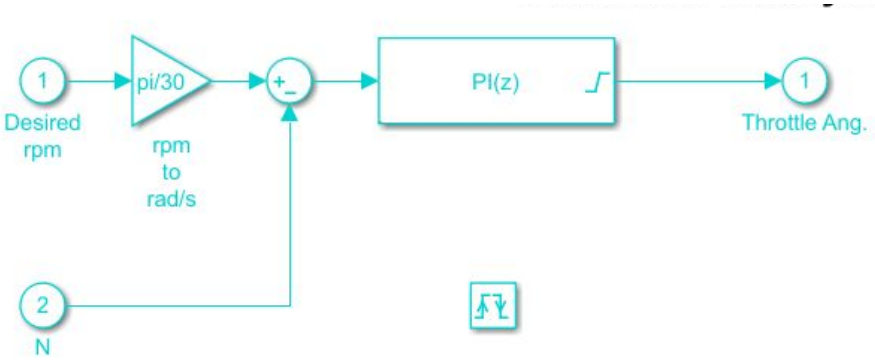

Figura 2: Interior del controlador

Previamente al análisis, se agregaron salidas nuevas al sistema en la señal de referencia y en la respuesta final (Figura 3). También se cambiaron las propiedades de modelado del sistema, de forma que los saltos temporales sean definidos explícitamente en lugar de

calculados por el sistema, y así obtener una respuesta más consistente al evaluar señales con diferentes configuraciones internas. El modelo fue guardado como “EngineTimingModel.slx”

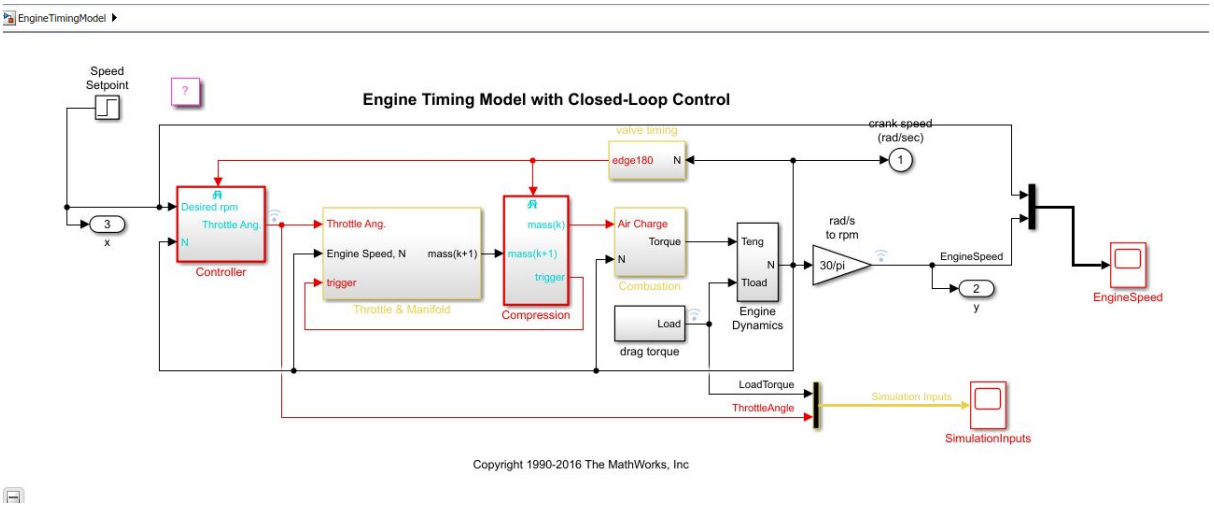

Figura 3: Modelo con salidas *x* e *y* adicionadas.

La respuesta del sistema en condiciones iniciales para una entrada step se muestra en la Figura 4. Podemos observar como el sistema presenta irregularidades anormales respecto a los modelos de respuesta comunes.

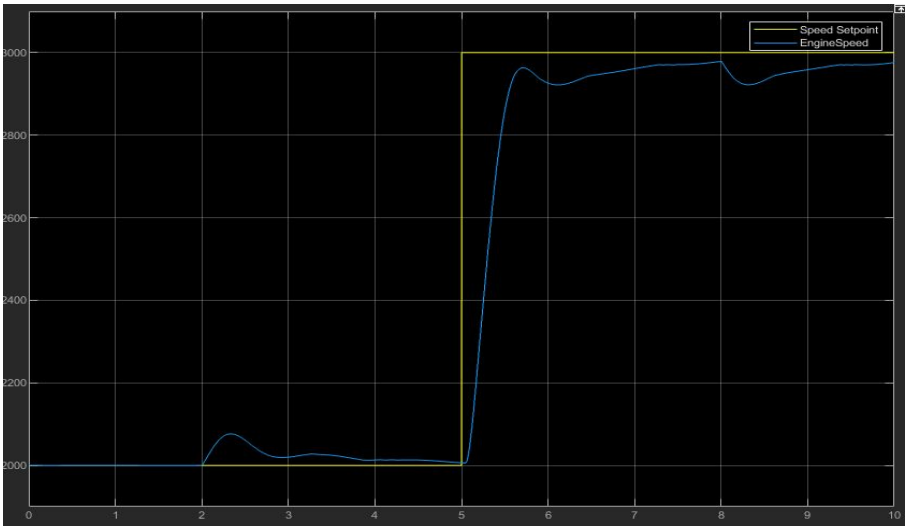

Figura 4: Respuesta predeterminada a impulso

Estudio previo

Desarrollo de función costo

El paso preliminar para encontrar los valores ideales para el controlador es el desarrollo de una función de costo apropiada para el sistema. Esta función debe depender de los valores del controlador utilizado, y debe regresar un valor proporcional al desempeño del controlador en el sistema. Como explicamos previamente, el sistema utiliza un controlador PI discreto, con valores por defecto Kp = 0,0723 y Ki = 0.0614 (Figura 5).

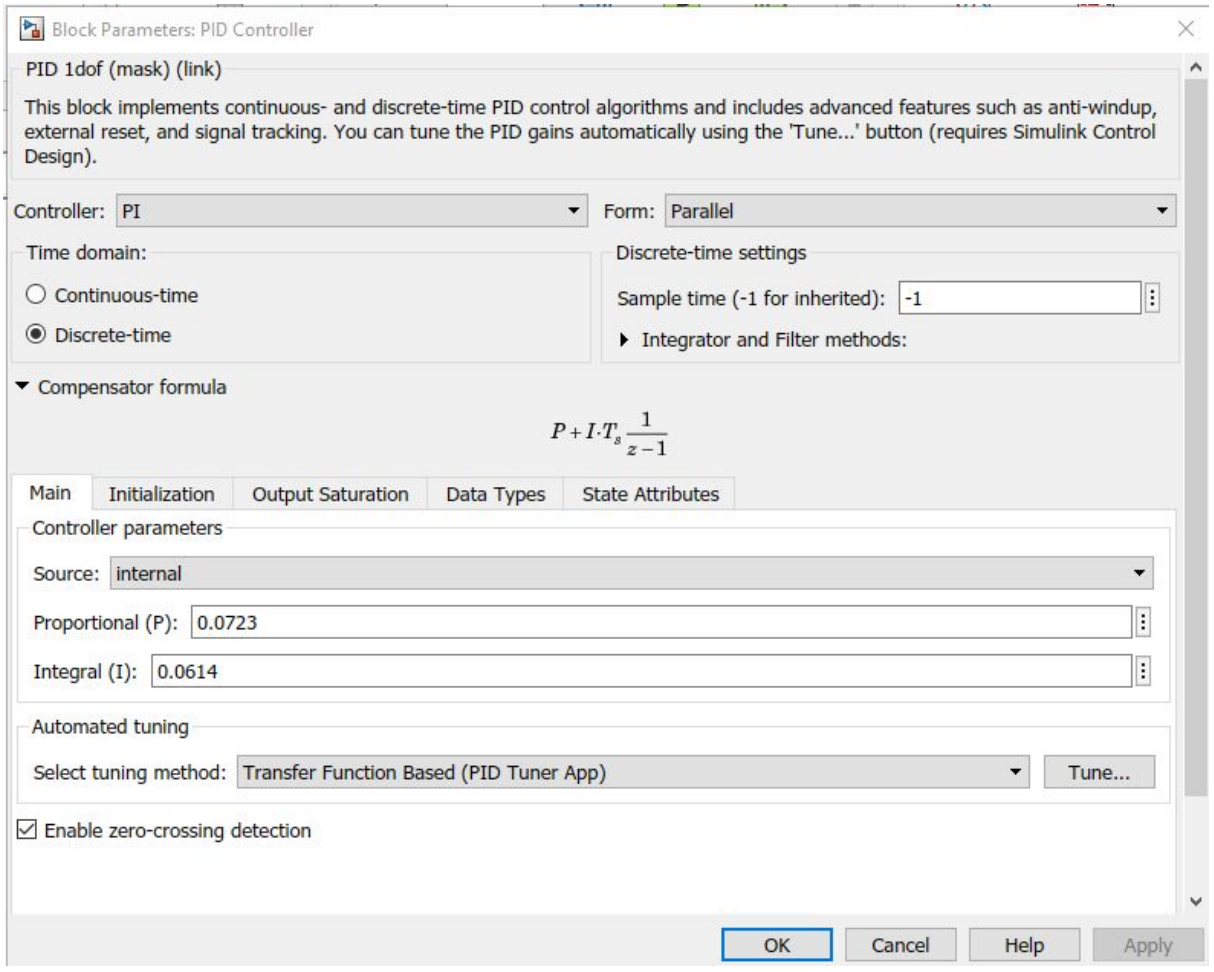

Figura 5: Bloque controlador dentro del subbloque

La función de costo, contenida en la función separada "*CostFunction.m*", recibe como entrada un array de una dimensión con los parámetros Kp y Ki, y retorna un valor escalar correspondiente al costo. Está dividida en 3 partes de acuerdo a su función: la primera parte realiza la configuración inicial del sistema, aplicando los parámetros recibidos al modelo, y si se reciben parámetros de entrada con valores imposibles (Kp o Ki <=0), se retorna un valor de costo extremadamente alto, no realizando simulación alguna. La segunda parte ejecuta la simulación y guarda los valores correspondientes de entrada (x), salida (y) y muestras de tiempo (t). El segmento final calcula el valor costo a partir del error lineal, siguiendo la fórmula:

$$\text{Error lineal} = \Sigma|y\text{-ref\_val}|/\text{no\_muestras}$$

Esta función, junto con la duración de la simulación, pueden ser cambiadas fácilmente para dar prioridad a comportamientos diferentes del sistema. Nótese que aunque dividimos por el número de muestras para normalizar los valores respecto a otras simulaciones, inicialmente configuramos la simulación para utilizar un número fijo de muestras, por lo que este paso solo divide el valor del error por una constante.

Componente Diferencial

En los controladores PID aplicados en ambientes altamente ruidosos, existe la posibilidad de que el componente diferenciador derive una cantidad de ruido significativamente alta, amplificando aún más el ruido. Este comportamiento, como se estudió en el primer laboratorio, es particularmente común en motores, como lo es este caso. Es posible filtrar la señal para disminuir el ruido, pero esto también limitaría la efectividad del componente diferencial.

Se realiza una comprobación inicial de la influencia del componente diferencial, sustituyendo el controlador predeterminado PI por uno PID, manipulando el valor de la nueva constante diferencial y estudiando el efecto que tiene sobre la respuesta del sistema. El sistema modificado fue guardado como "*EngineTimingModelPID.slx*". Hemos evaluado la función costo para las condiciones iniciales de Kp y Ki, y varios valores de Kd (Tabla 1).

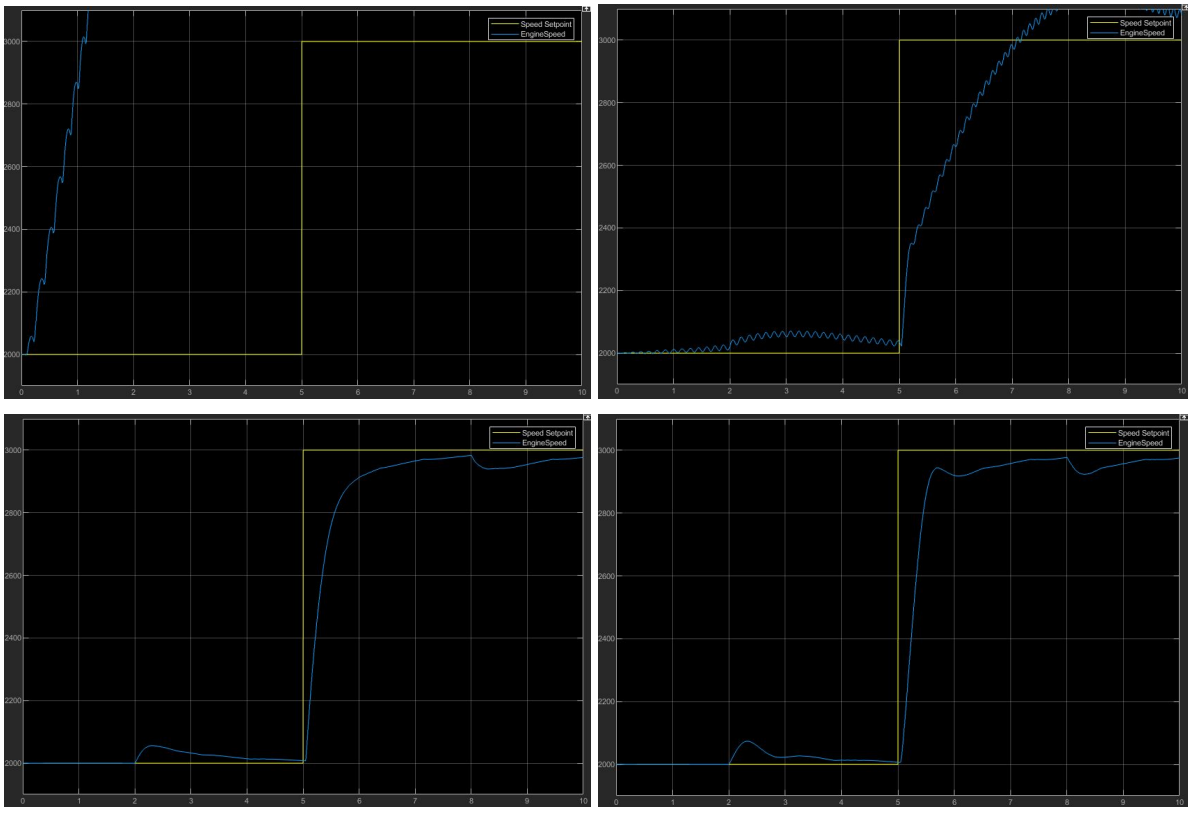

Figura 6: Respuesta para Kp, Ki inicial, varios valores de Kd.

Kd	Costo
1	3.6963e+03
0.1	117.3606
0.01	60.1863
0.001	58.3832
0	58.3754

Tabla 1: Respuesta para Kp, Ki inicial, varios valores de Kd.

Como se esperaba, se puede apreciar como el parámetro Kd solo tiene un efecto negativo en la efectividad del sistema. Causa oscilaciones violentas apreciables en la respuesta en valores altos. Para valores pequeños, tiene un efecto de filtro paso bajo, limitando la respuesta rápida a impulsos nuevos. Se ha proseguido con el modelo original con el controlador PI discreto ("*EngineTimingModel.slx*") para el resto de las optimizaciones, reduciendo el problema a dos dimensiones.

Datos preliminares

Es conveniente tener una idea general del efecto de los parámetros en el comportamiento del sistema. Para ello, podemos desarrollar un estudio sencillo: Generamos una muestra pequeña de valores de constantes para el rango de funcionamiento esperado para el diseño, y calculamos el valor de costo. Para un rango de ambos valores de [0,2], obtenemos la gráfica presentada en la Figura 5.a, Notamos valores relativamente altos de costo para valores de Kp cercanos a 0 que distorsionan la gráfica. Graficando sin estos puntos con el comando: `surf(x(:,2:end),y(:,2:end),z(:,2:end))`,con lo que obtenemos la Figura 5.b.

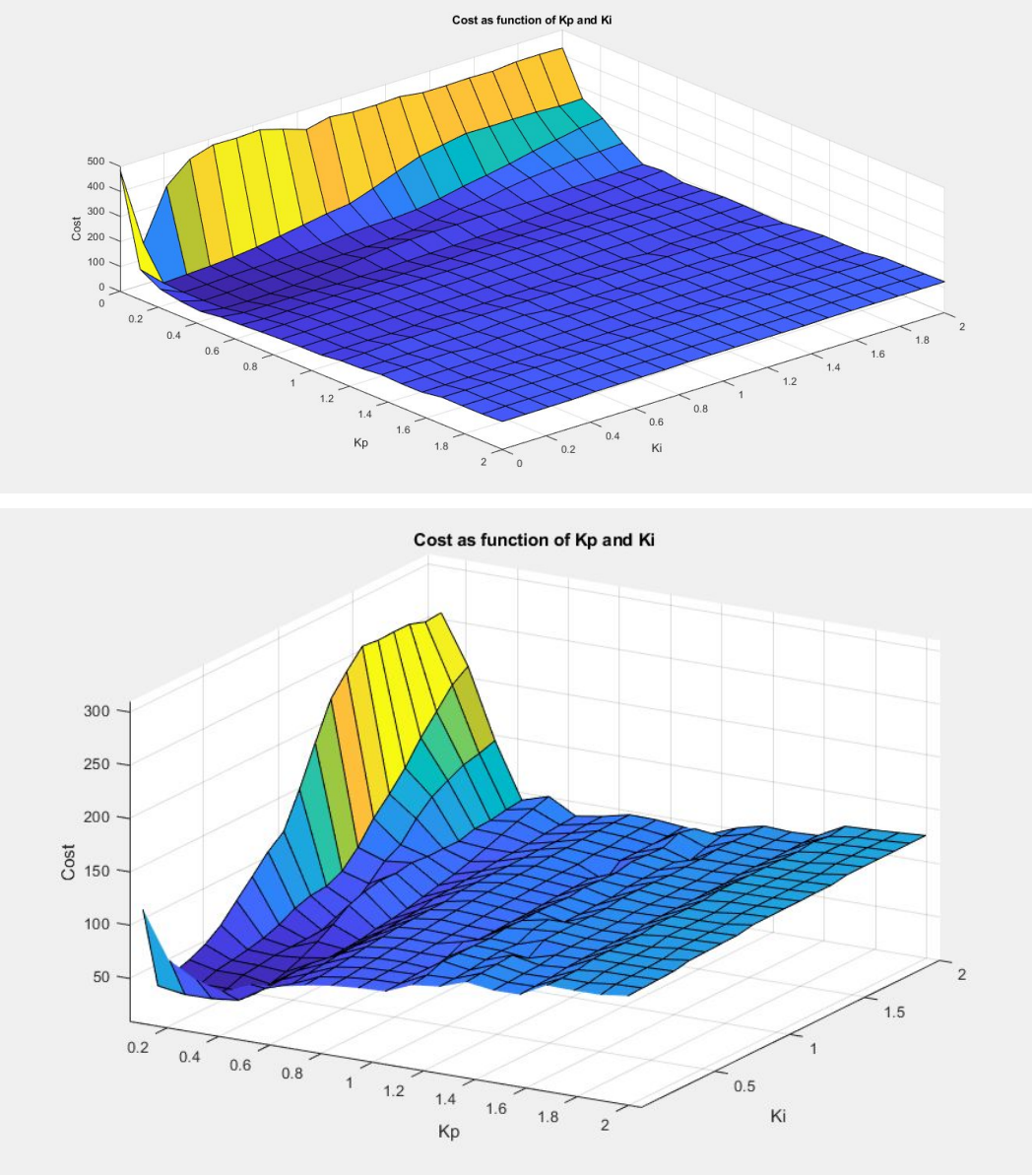

Figura 5: Inicial (a), después de remover valores extremos (b)

Podemos observar que la función costo parece tener un mínimo global en el rango para un valor aproximado de Kp=0.2 y Ki=0.1. Dado que el tiempo necesario para recalcular estos puntos es bastante elevado, los valores se han guardado en un archivo externo (Costs.xls). Este archivo es leído cuando la función de salida se ejecuta. El código original para su generación se mantiene en el archivo "*main.m*", pero ha sido comentado para prevenir su ejecución.

Optimizacion

Para este ejercicio de minimización, utilizaremos la función de optimización Global Search. Esta función se basa en ejecutar el minimizador local fmincon desde múltiples puntos diferentes hasta cumplir ciertos criterios de selección. Su ventaja principal comparado con algoritmos similares, en particular Multistart, es su capacidad de rechazar inteligentemente selecciones que sean poco probables de mejorar los resultados, y de ofrecer el mejor desempeño en ordenadores con procesadores de un solo núcleo (el procesador utilizado para realizar los cálculos expuestos). Global Search tiene limitaciones a encontrar soluciones más precisas comparado con otros algoritmos, pero su mayor velocidad permite experimentación en otras situaciones y configuraciones de sistema. Su algoritmo local es fmincon.

Uso de fmincon

Previamente a utilizar Global Search, se estudiará fmincon como solucionador local en puntos específicos para evaluar su desempeño y comparar con nuestros datos de referencia.

La función fmincon permite aplicar un gran número de controles y restricciones a la sección analizada. Para nuestro caso particular, sólo es necesario aplicar las restricciones de límites en el rango de trabajo [0,2] aplicado a las dos constantes Ki y Kp, dejando los otros campos para restricciones en blanco. Por otra parte, para las opciones de ejecución tenemos:

- TolFun: Tolerancia de la función. Iteraciones terminan cuando el algoritmo no sea capaz de disminuir la función costo por este valor. Asignamos 1e-3 (3 decimales de precisión).
- OutputFcn: Función de salida, utilizada para graficar resultados durante la ejecución. Hemos creado la función personalizada “*OutputFMincon.m*”, que imprime las iteraciones sobre la superficie calculada previamente.
- MaxIter: Número de iteraciones a ejecutar. Asignamos 20.
- MaxFunEvals: Máximo número de evaluaciones de función costo por ejecución. Asignamos 100.

Por defecto, fminsearch crea y utiliza información de gradiente de la superficie asignada a optimizar, y su desempeño está limitado por la información de gradientes. Los intentos iniciales de optimizar la función llegaban a la conclusión errónea de que toda posición era un mínimo local (Se puede apreciar en la superficie de la Figura 5 que existen “mesetas” amplias). Esto se puede corregir aumentando la precisión del algoritmo y cambiando la opción ‘FinDiffType' a 'central' para obtener resultados más precisos.

La Figura 6 muestra el resultado para diferentes valores de prueba/puntos de inicio. Podemos observar que el solucionador puede identificar mínimos locales como globales,

pero cuando las condiciones iniciales están cerca del punto que identificamos previamente como candidato a mínimo, ha encontrado una coordenada precisa con un buen candidato a mínimo global.

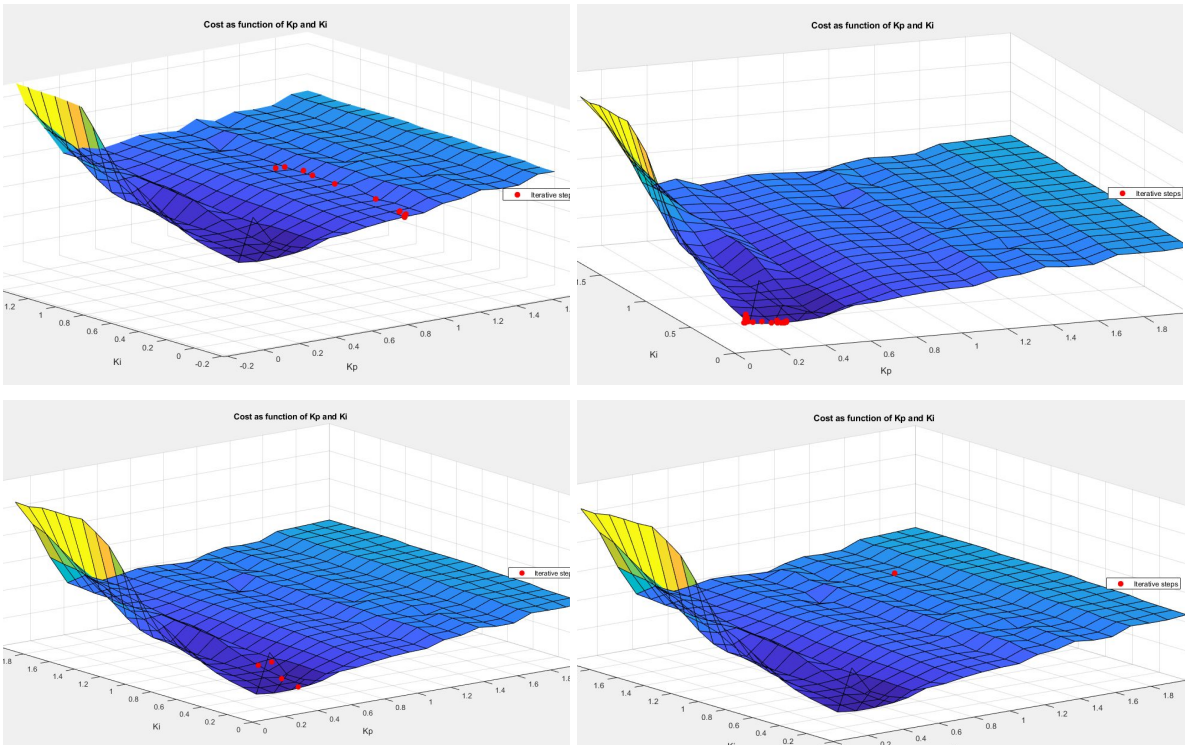

Figura 6: Gráficas con puntos de evaluación a)Kp=1, Ki=1 b)Ki = 0.0723, Kp = 0.0614 (valores predeterminados) c) Ki = 0.5, Kp = 0.5 d) Ki = 1.5, Kp = 1.5

Inicial	Final	Costo	Iteraciones	Evaluaciones de función
Kp=1 Ki=1	Kp = 1.0987 Ki = 1.0016	85.923	7	90
Ki = 0.0723 Kp = 0.0614	Kp = 0.26922 Ki = 0.1336	31.4702	18	173
Ki = 0.5 Kp = 0.5	Kp = 0.3776 Ki = 0.1318	34.7852	9	88
Ki = 1.5 Kp = 1.5	Kp = 1.5 Ki = 1.5	106.4404	0	5

Tabla 1: Resultados de evaluación

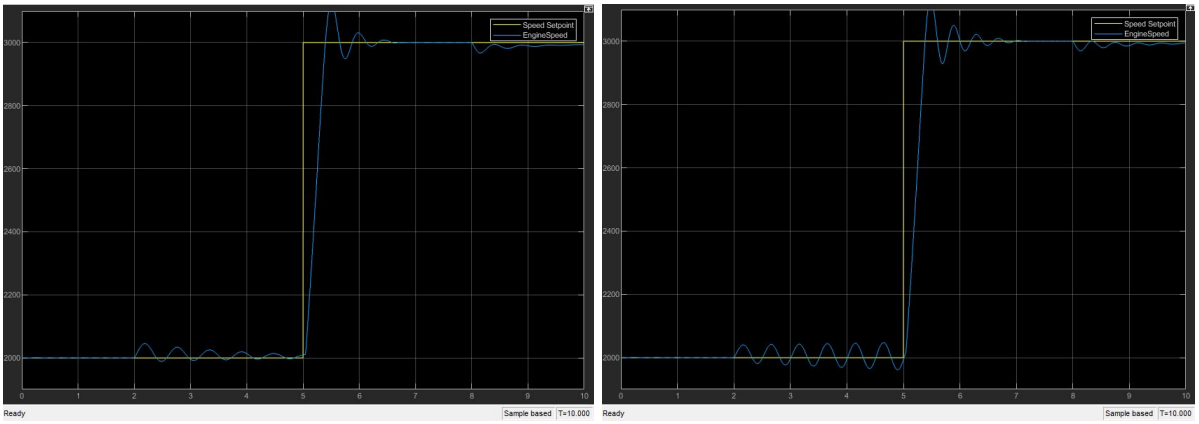

Figura 7: Respuesta de nuestro sistema para: (a) Kp = 0.26922, Ki = 0.1336
(b) Kp = 0.3776, Ki = 0.1318

Notamos que tras el proceso de ejecución el algoritmo ha escogido una respuesta de señal muy rápida/transición corta, pero que resulta en oscilaciones rápidas pequeñas. Más adelante modificaremos nuestra función costo para dar prioridad a respuestas con otras características.

Uso de Global Search

Descripción del algoritmo

Similar a nuestro proceso anterior para escoger puntos de prueba, el algoritmo Global Search escoge puntos dónde aplicar inteligentemente la función fminsearch. El algoritmo ejecuta:

1. **Ejecución de fmincon en x0:** Global Search ejecuta repetidamente fmincon en el punto inicial, para determinar variabilidad. En Matlab, al inicio la ejecución, podemos ver cómo el algoritmo se ejecuta repetidamente en el punto definido por x0.
2. **Generación de puntos de prueba:** GlobalSearch genera puntos de prueba iniciales, utilizados como posibles puntos de arranque para futuras iteraciones de fmincon.
3. **Obtener punto de inicio, etapa 1:** GlobalSearch evalúa los puntos de prueba anteriores, selecciona los mejores valores, y remueve una cantidad *NumStageOnePoints* de los peores puntos.
4. **Configuracion de parametros internos:** Pozos de concentración.
5. **Ejecución de bucle:** Global Search examina los puntos resultantes, parando cuando los algoritmos encuentren un punto final o el tiempo establecido se agote.

Con el sub-algoritmo fmincon configurado correctamente, podemos proceder a la configuración de Global Search. En Matlab creamos un objeto “Global Search” con las opciones de configuración deseadas:

- **NumTrialPoints:** Número de puntos de prueba. Corresponde al número de puntos iniciales para ejecutar fmincon (Segundo paso del algoritmo). Asignamos 50.
- **NumStageOnePoints:** Número puntos a desaparecer en la tercera parte del algoritmo. Quedan un total de *(NumTrialPoints-NumStageOnePoints)*. Asignamos 20, dejando 30 puntos finales.
- **FunctionTolerance:** Tolerancia de la función. Diferencia mínima para distinguir dos valores de la función costo como diferentes. Asignamos 1e-4, aproximadamente 0.01% del valor de la función costo (C=~31.4702) visto en el apartado anterior.
- **MaxTime:** Tiempo máximo de ejecución en segundos. Asignamos 20*60 para limitar ejecución a 20 minutos.
- **PlotFcn:** Función de salida del algoritmo. Asignamos la funcion definida externamente "*OutputGlobalSearch.m".* Existe también el parametro **OutputFcn,** que permite detener externamente la ejecución del algoritmo, funcionalidad que no necesitamos.
- **StartPointsToRun:** Puntos a ejecutar. Asignamos 'bounds' para solo ejecutar los puntos en el rango definido.

Finalmente, en la definición del problema incorporamos las propiedades del fmincon y ejecutamos nuestro algoritmo. Iniciando en un punto donde previamente obtuvimos malos resultados [1.5, 1.5], obtenemos el conjunto [0.2574, 0.1669] con un costo de 33.2190, ligeramente superior al valor más pequeño encontrado previamente (Figura 8). El incremento del valor de costo puede darse por cambio de precisión entre ejecuciones.

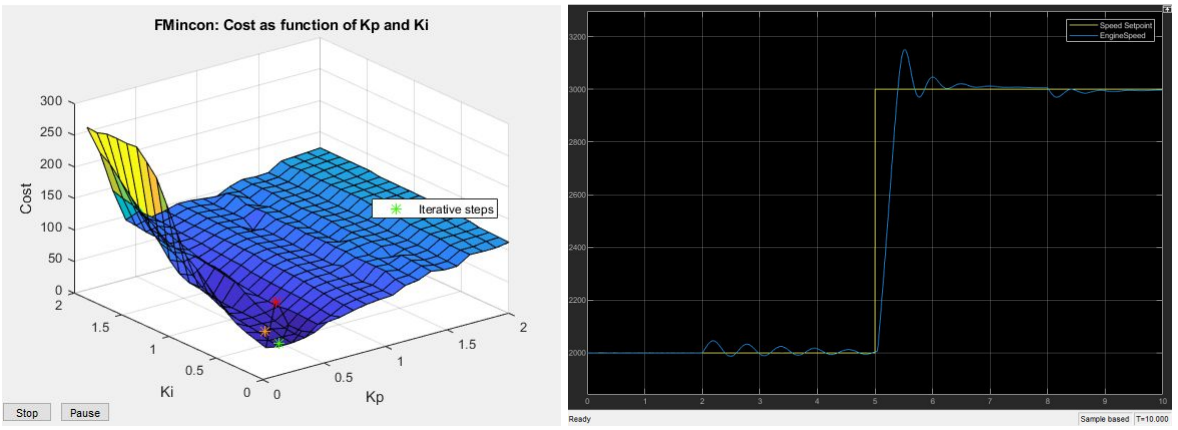

Figura 8: Solución del Global Search desde un punto de inicio lejano a la respuesta.

Con el sistema configurado podemos buscar nuevos puntos. Ejecutando nuevamente para un rango de valores mayor [0,5] para ambas constantes, obtenemos una respuesta similar, con el sistema llegando a Kp = 0.25744,Ki = 0.16691, con un costo de 33.219. Podemos que concluir que el sistema está configurado correctamente y puede encontrar costos pequeños para funciones costo.

Respuestas Específicas

Finalmente, con el solucionador funcionando correctamente, podemos inducir comportamientos particulares en el sistema cambiando la función costo. Notamos que utilizando error lineal, a pesar de su rápida respuesta, ocurren un número considerable de oscilaciones. Utilizando un función de error que penalice las oscilaciones podemos contrarrestarlas.

Para diseñar un controlador que priorice la rapidez de su respuesta, tenemos que diseñar una función costo que se comporte de manera apropiada: penalizando gravemente valores lejanos a nuestro objetivo. Utilizando la función de error cuadrático, castigamos más fuertemente valores grandes de error, logrando en teoría una función con menos oscilaciones. Cambiando nuestra función de error y ejecutando en nuestro rango original, obtenemos el resultado de la figura 9. Cabe notar que la superficie obtenida previamente no corresponde a los valores nuevos, y solo nos interesa el conjunto de constantes y respuesta final.

$$\text{Error Cuadratico} = \text{sqrt}(\Sigma|y\text{-ref\_val}|^2)/\text{no\_muestras}$$

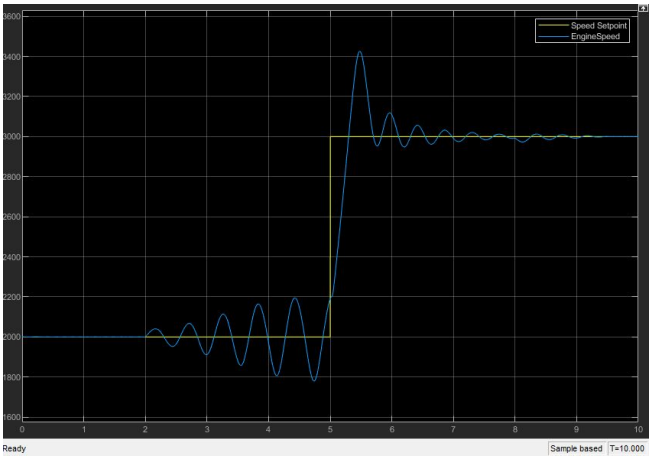

Figura 9. Minimización con error cuadrado Kp =0.3776, Ki=0.4002

Obtenemos un efecto inesperado al minimizar en el área central. Dado que en el momento de transición t=5s aparece un error muy grande en comparación con los alrededores, el sistema se ha optimizado para reducirlo. Como consecuencia, la función utiliza.

Notablemente, se puede observar que el sistema se ha ajustado para alcanzar su oscilación máxima en t=5s, por lo que involuntariamente, hemos optimizado para realizar la transición más rápida posible.

Podemos modificar nuestra función de error para corregir este problema, simplemente ignorando el error en un rango pequeño cercano a t=5s. Hemos excluido el rango [5.0-5.4]s de nuestra fórmula y repetimos la ejecución.

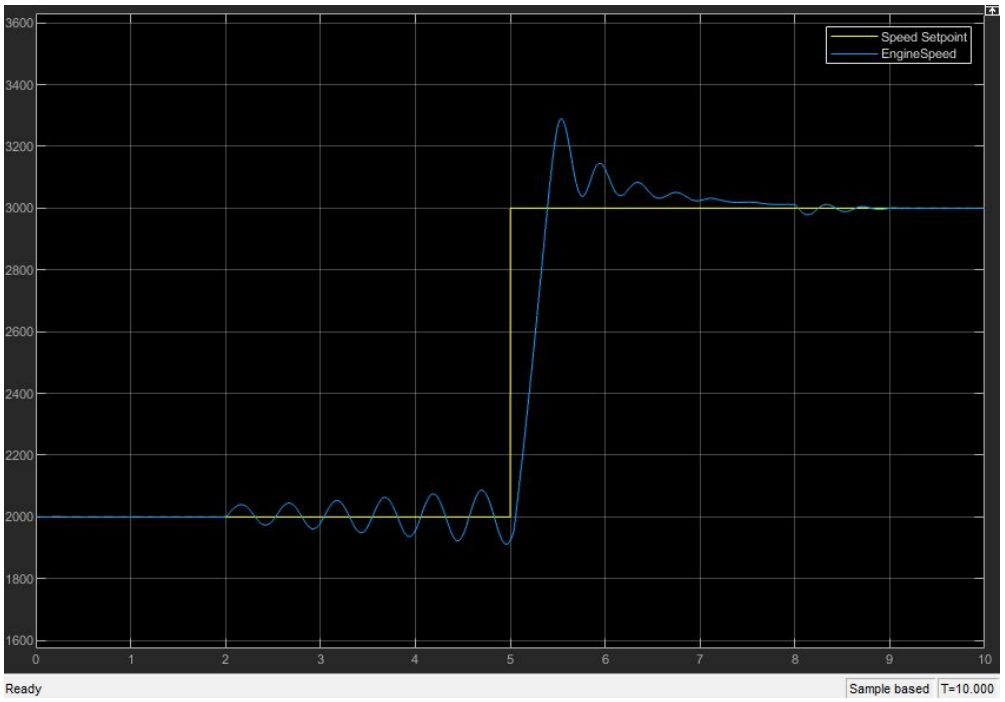

Figura 10: Minimización con error cuadrado ignorando [5-5.4]s. Kp =0.3776, Ki=

Después de experimentar un poco con los parámetros, encontramos los resultados sobresalientes de la Figura 11.

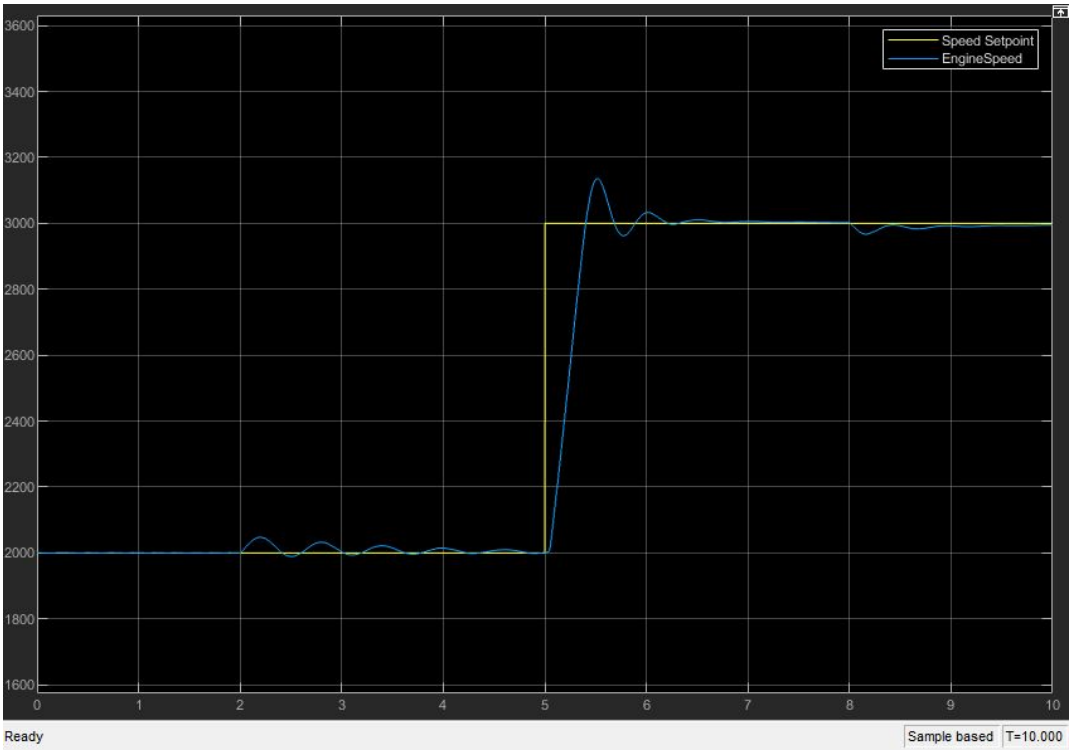

Figura 11: Minimización con error cuadrado, rango [0,0.5s] ignorando [5-6]s.
Kp =0.2434, Ki=0.1446

Conclusion

En el transcurso de este trabajo, optimizamos el controlador incluido en el sistema. Logramos configurar los algoritmos Global Search y fmincon para encontrar mínimos locales dentro de un rango fijo de forma confiable y modificar nuestra función de costo para optimizar cualidades específicas de nuestra señal de respuesta.

Después de configurar nuestros algoritmos de búsqueda, comprobamos su funcionamiento independiente con nuestro análisis previo. Comprobamos su funcionamiento y limitaciones al ejecutarlo sobre áreas grandes y puntos inconvenientes, encontrando las soluciones correctas, pero con tiempos de ejecución bastante mayores. Una vez el algoritmo de optimización está finalizado, los resultados dependen casi exclusivamente de una función de costo desarrollada correctamente. Con la función correcta y rango de búsqueda correcta, podemos obtener resultados extremadamente precisos.

Un aspecto no explorado durante este trabajo fue probar señales alternativas de control, fuera del salto de 2000 a 3000 rpm. Cabe la posibilidad de que un PI optimizado con los valores determinados no presente respuesta óptima fuera de esta área. En este caso, se puede optimizar modificando el bloque generador de señales en simulink con la señal deseada.

Dado la gran gama de herramientas de búsqueda y minimización, utilizar funciones de costo junto con controladores PIDs resulta una herramienta muy poderosa para diseño de controladores. En particular para este sistema, el uso de un controlador PI resulta ideal, ya que al solo tener dos dimensiones, resulta fácil y relativamente rápido de analizar y tratar comportamiento inesperado o disturbaciones.