



**UNIVERSIDAD
CARLOS III DE MADRID**

CONTROL INTELIGENTE

CONTROL DE ÁNGULO PITCH DE UN HELICÓPTERO CON DIFFERENTIAL EVOLUTION

**Profesores: Luis Enrique Moreno Lorente
Luis Santiago Garrido Bullón
Dorin Sabin Copaci**

CONTROL DE ÁNGULO PITCH DE UN HELICÓPTERO CON DIFFERENTIAL EVOLUTION

El objetivo de este trabajo será optimizar los parámetros de un controlador PID mediante el método de optimización *Differential Evolution* para conseguir controlar el ángulo *Pitch* de un helicóptero de manera que pueda seguir a una señal de referencia.

1. Modelo del helicóptero en Simulink

El modelo utilizado en el desarrollo de este trabajo es *rct_helico.slx*, disponible entre la librería de modelos de Simulink. Este modelo se muestra en la figura 1.

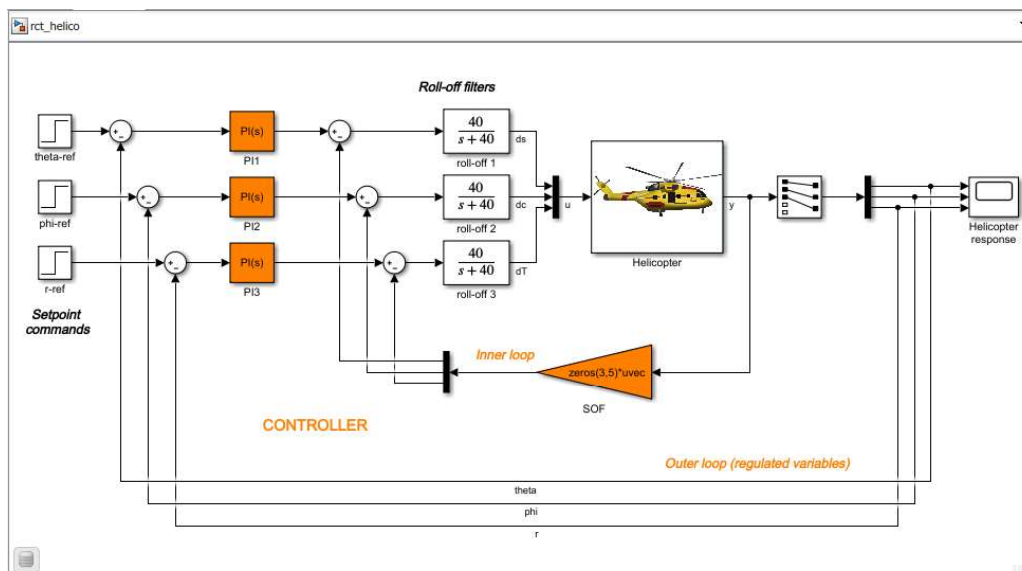


Figura 1: Modelo rct_helico.slx

Si se visita la ayuda de Matlab sobre el modelo [1], se especifica que *theta* en el modelo corresponde al ángulo *Pitch*, *phi* al ángulo *Roll* y, finalmente, *r* modela la tasa de variación del ángulo *Yaw*.

Por simplificar el problema de control, se reduce el objetivo del proyecto a controlar únicamente el ángulo *Pitch*. Controlar las otras dos variables consistiría en aplicar el mismo proceso que se detallará a continuación a dichas variables. También, se va a intentar usar un controlador PID en lugar del PI que sugiere el modelo original. Por tanto, el modelo de partida ahora es el de la figura 2, nombrado como *rct_SOF.slx*.

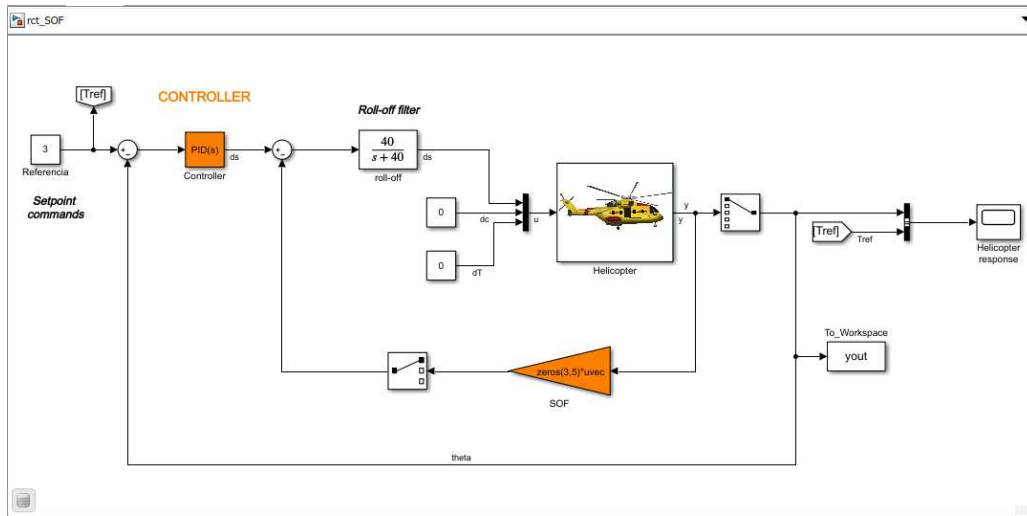


Figura 2: Modelo rct_SOF.slx

Como elementos interesantes del modelo se destacan:

- **Bloque roll-off:** Constituye un filtro paso bajo con frecuencia de corte en 40 rad/s para limitar ruido y controlar que la dinámica del sistema no sea excesivamente rápida.
- **Controlador PID:** Para conseguir que el ángulo *Pitch* siga a la señal de referencia.
- **Bloque SOF:** Proporciona un lazo de realimentación estática cuyo objetivo es dotar de estabilidad al sistema ayudando a evitar respuestas con *overshoot*.

2. Pruebas y resultados

Para la optimización de los parámetros del controlador se utilizó el código proporcionado por el profesor, disponible en Aula Global:

- **Archivo Pid_DIiff_Evol_3.m.** Será el archivo a ejecutar. Permite fijar los parámetros de configuración del algoritmo, entre ellos: Número de parámetros a optimizar (D), valores mínimo y máximo de dichos parámetros en la población inicial (XVmin y XVmax), número de individuos de la población (NP), número de iteraciones o número de generaciones (itermax), factor de amplitud de diferencias (F), constante de cruce (CR) y valor de función de coste a alcanzar (VTR). Para el desarrollo de todo el trabajo se eligen los siguientes valores (tabla 1):

VTR	D	XVmin	XVmax	NP	itermax	F	CR
1.e-6	3	[0,0,0]	[2,2,2]	30	20	0.7	0.5

Tabla 1: Parámetros del Differential Evolution

- **Archivo tracklsq3.m.** Función que, recibiendo unos parámetros determinados del controlador, simulará el modelo de Simulink con ellos y establecerá mediante una función de coste cómo de buenos son esos parámetros. Esta función se ha modificado para que rechaze los parámetros de PID negativos o mayores de 100 y, en la medida de lo posible, rechaze los parámetros del PID que hagan que la salida del sistema tenga máximos locales (para evitar salidas abruptas que se comprobó eran muy casuales). La función de coste a optimizar consiste en la diferencia en régimen permanente de la señal de referencia y la salida del sistema.

Para mayor claridad, el código se muestra a continuación:

```

1   function F = tracklsqmin(pid,y)
3   Kp = pid(1);
4   Ki = pid(2);
5   Kd = pid(3);
7   if Kp<0 || Ki<0 || Kd<0 || Kp>100 || Ki>100 || Kd>100
8       F = 10000000000;
9       return;
10  else
11      set_param('rct_SOF/Controller','P',num2str(Kp));
12      set_param('rct_SOF/Controller','I',num2str(Ki));
13      set_param('rct_SOF/Controller','D',num2str(Kd));
14      sim('rct_SOF',[0 50]);
15      max = findpeaks(yout.signals.values);
17      for i = 1:length(max)
18          dif = 3 - max(i);
19
20          if dif>0.05
21              F = 10000000000;
22              return;
23          end
24      end
25      F=(abs(yout.signals.values(length(yout.signals.values))-3));
26      return;
27  end
28  end

```

- **Archivo devec3.** Función que implementa el algoritmo *Differential Evolution*, con las etapas de inicialización de la población, mutación, cruce y selección. Este archivo no sufre ninguna modificación respecto del original a lo largo del trabajo.

El tiempo de simulación fijado para el modelo es de 50s y el *solver* elegido el ode45 con paso variable.

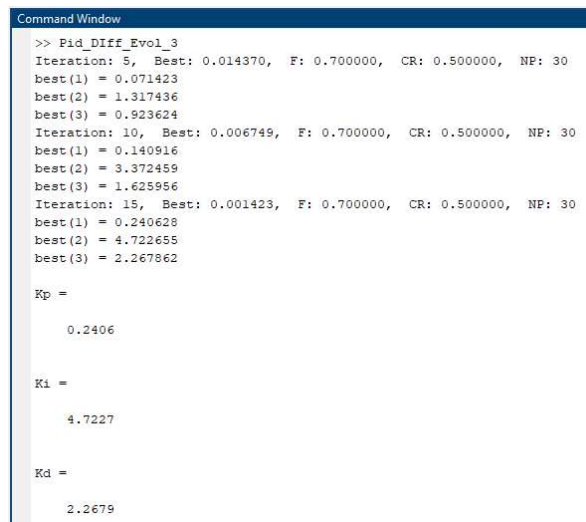
2.1. Controlador PID

En primer lugar, se intenta controlar sistema únicamente con un controlador PID, prescindiendo del lazo de control interno que incluye el bloque SOF. Para ello, el modelo se mantiene tal y como se mostró en la figura *rect_SOF.slx*, donde el bloque SOF consiste en una matriz de ceros, es decir, no tiene ningún efecto.

Se ejecuta el algoritmo optimizador varias veces y se obtienen los siguientes resultados:

- **Ejecución 1:** $K_p = 0.2406$, $K_i = 4.7227$ y $K_d = 2.2679$.

Para esta primera ejecución se muestra como ejemplo en la figura 3 cómo se presentan los datos en la ventana de comandos tras la ejecución del algoritmo.



```
Command Window
>> Pid_Diff_Evol_3
Iteration: 5, Best: 0.014370, F: 0.700000, CR: 0.500000, NP: 30
best(1) = 0.071423
best(2) = 1.317436
best(3) = 0.923624
Iteration: 10, Best: 0.006749, F: 0.700000, CR: 0.500000, NP: 30
best(1) = 0.140916
best(2) = 3.372459
best(3) = 1.625956
Iteration: 15, Best: 0.001423, F: 0.700000, CR: 0.500000, NP: 30
best(1) = 0.240628
best(2) = 4.722655
best(3) = 2.267862

Kp =
    0.2406

Ki =
    4.7227

Kd =
    2.2679
```

Figura 3: Presentación de resultados en la ventana de comandos

Se introducen estos valores en el PID del modelo de Simulink y se ejecuta. En el *scope* se puede visualizar ahora el resultado del control implementado, que corresponde con la figura 4, donde la señal azul es la señal de referencia y la amarilla la salida del sistema.

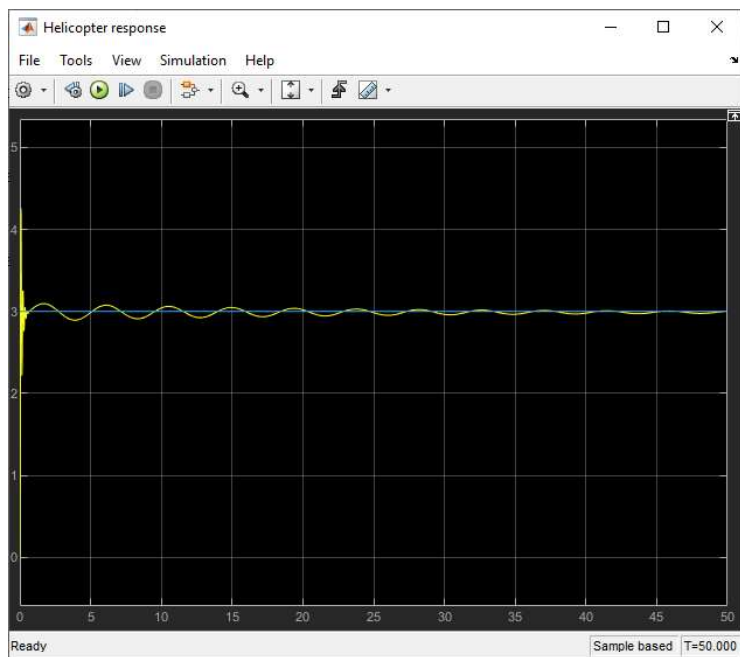


Figura 4: Ejecución 1 algoritmo DE sin bloque SOF

- **Ejecución 2:** $K_p = 3.2135$, $K_i = 14.1736$ y $K_d = 2.2544$.
Con estos nuevos valores, el resultado obtenido es el de la figura 5.

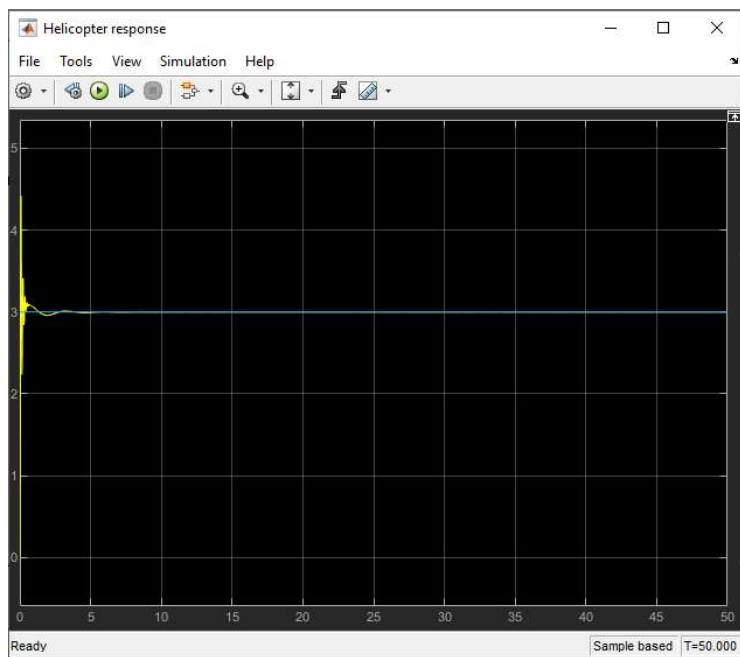


Figura 5: Ejecución 2 algoritmo DE sin bloque SOF

- **Ejecución 3:** $K_p = 2.4474$, $K_i = 10.9629$ y $K_d = 2.4106$.
En esta última prueba, los resultados son los mostrados en la figura 6.

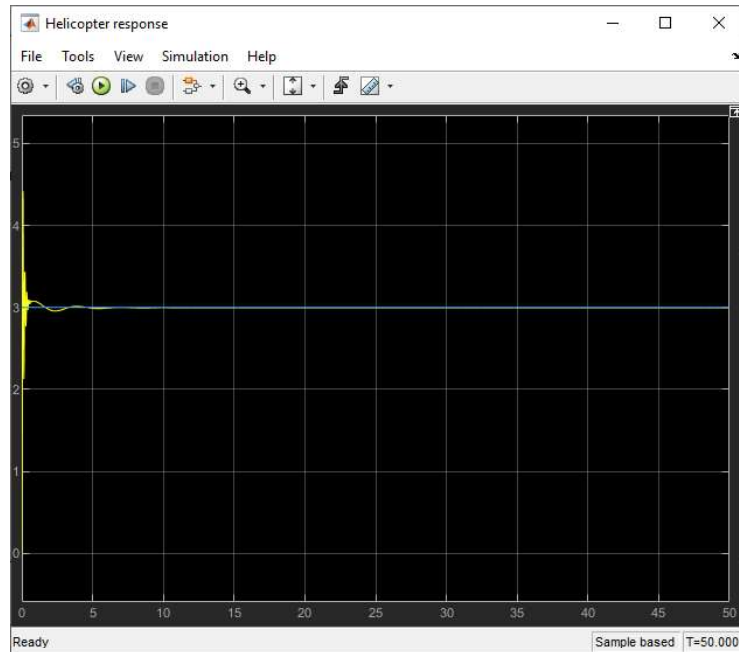


Figura 6: Ejecución 3 algoritmo DE sin bloque SOF

En base a estos resultados se puede concluir que:

- Se consigue que el error en régimen permanente sea 0, es decir, el sistema es capaz de seguir a la señal de referencia.
- Sin embargo, la respuesta del sistema es muy poco realista ya que tiene un tiempo de subida inviable en un sistema real, además de presentar picos en el régimen transitorio, pese a haberlo intentado evitar con la función de coste, como se ha visto anteriormente.

Como última prueba con este controlador se intenta, con los valores de PID de esta última ejecución, a cambiar la señal de referencia por un escalón, obteniéndose la siguiente gráfica (figura 7):

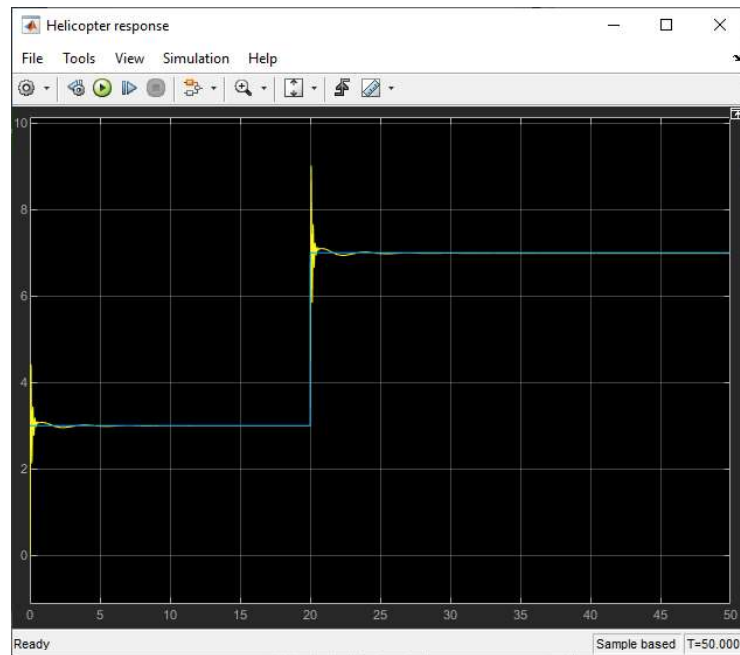


Figura 7: Ejecución 4 algoritmo DE sin bloque SOF

Obteniéndose un resultado como el esperado: se consigue seguir la señal de referencia pero con una dinámica muy rápida y con picos.

2.2. Controlador PID + bloque SOF

En base a los resultados anteriores, todo parece indicar que la realimentación mediante el bloque SOF es necesaria para conseguir una respuesta estable del sistema.

El bloque SOF, como se ha dicho antes, consiste en una matriz de 3×5 que ha de sintonizarse con los valores adecuados para obtener un comportamiento correcto. Dado que en el caso de este trabajo sólo se está controlando el ángulo θ del helicóptero, sólo es de interés la primera fila de dicha matriz. El resto de elementos podrán tomar un valor de 0.

¿Cómo elegir los valores de la matriz? Se ha hecho uso de la herramienta *slTuner*, que es la utilizada en la ayuda de Matlab del modelo [1] tanto para sintonizar los controladores PI como el bloque SOF del modelo original. Aunque el objetivo del trabajo es sintonizar el controlador PID mediante *Differential Evolution*, se ha de hacer una primera aproximación usando dicha herramienta con el fin de ajustar los valores de la matriz del bloque SOF a un controlador PID que se sepa que es correcto. Una vez ajustados dichos valores, se procederá a ejecutar de nuevo el algoritmo de optimización para obtener los parámetros del PID con la técnica deseada.

El código usado para esa primera aproximación y ajuste de los valores de la matriz SOF se adjunta a continuación (archivo *SOF.m*). Básicamente es el recomendado por Matlab para el ajuste de los parámetros para que el sistema cumpla una serie de condiciones.

```

1 ST0 = slTuner('rct_SOF',{'Controller','SOF'});
  addPoint(ST0,{'Referencia'}) % setpoint commands
3  addPoint(ST0,{'theta'}) % corresponding outputs
  addPoint(ST0,{'u','y'}); % plant input and output
5
  % Less than 20% mismatch with reference model 1/(s+1)
7  TrackReq = TuningGoal.StepTracking({'Referencia'},{'theta'},1);
  TrackReq.RelGap = 0.2;
9
  % Gain and phase margins at plant inputs and outputs
11 MarginReq1 = TuningGoal.Margins('u',5,40);
  MarginReq2 = TuningGoal.Margins('y',5,40);
13
  % Limit on fast dynamics
15 MaxFrequency = 25;
  PoleReq = TuningGoal.Poles(0,0,MaxFrequency);
17
  AllReqs = [TrackReq,MarginReq1,MarginReq2,PoleReq];
19 ST1 = systune(ST0,AllReqs);

```

Se ejecuta el código y, mediante el comando *showTunable(ST1)*, se observa en la ventana de comandos el resultado del ajuste (figura 8).

```

Command Window
>> SOF
Final: Soft = 1.28, Hard = -Inf, Iterations = 86
>> showTunable(ST1)

Block 1: rct_SOF/Controller =

      1      s
Kp + Ki * ---- + Kd * ----
      s      Tf*s+1

with Kp = 2.03, Ki = 4.22, Kd = 0.12, Tf = 0.0452

Name: Controller
Continuous-time PIDF controller in parallel form.

-----

Block 2: rct_SOF/SOF =

D =
      u1      u2      u3      u4      u5
y1  4.59 -0.0003554 2.531e-05 0.6735 -0.002247
y2  2.182e-15 -2.17e-16 -1.482e-16 1.451e-16 -2.079e-18
y3  4.109e-16 3.084e-16 1.343e-17 4.901e-16 7.531e-16

Name: SOF
Static gain.

```

Figura 8: Resultado de la sintonización mediante slTuner

Como se ha dicho, los valores del PID obtenido se ignoran para a continuación obtenerlos mediante *Differential Evolution*. De la matriz obtenida se observa cómo los valores de las dos segundas filas pueden ser sustituidas por 0, como se había comentado anteriormente. Se procede ahora guardar los datos de la matriz en el modelo de Simulink en un callback que se inicialice al ejecutar el modelo con el nombre 'D', y se introduce esta matriz en el bloque SOF.

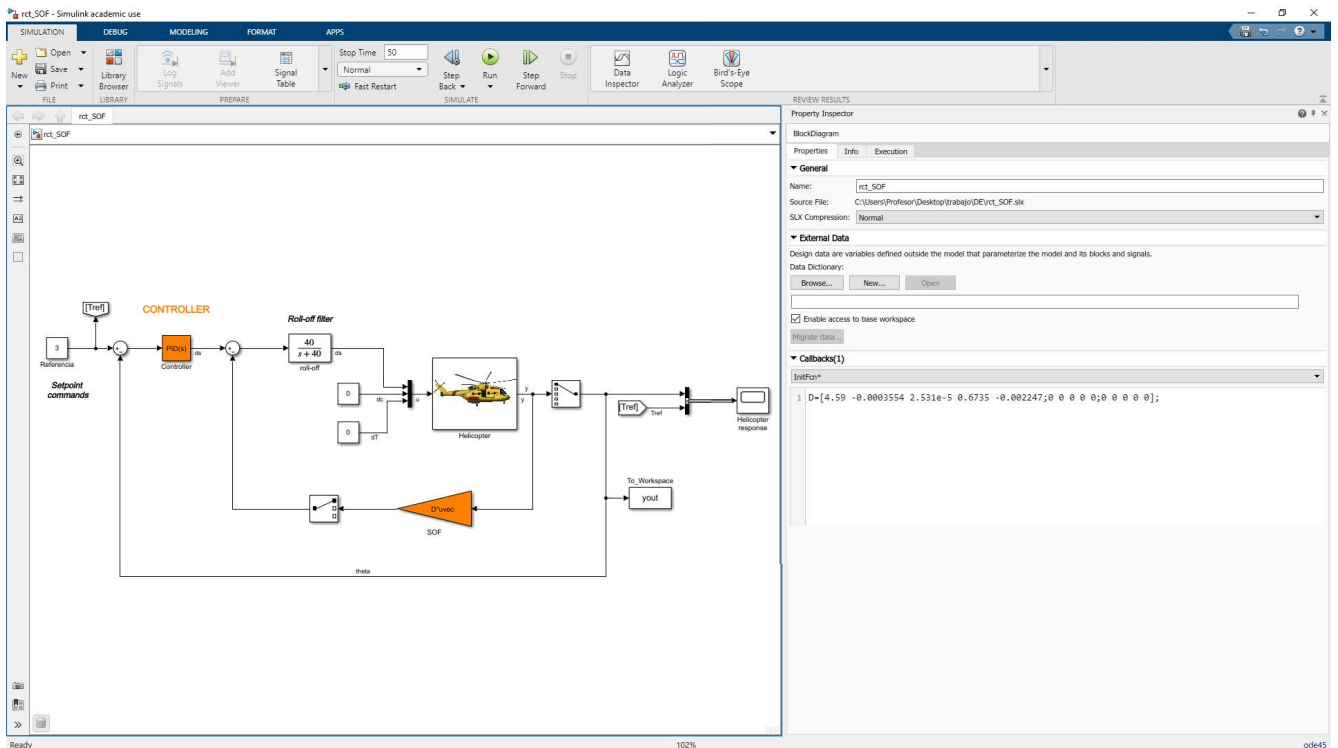


Figura 9: Ajuste del bloque SOF

Finalmente, igual que se hizo en el apartado anterior, se ejecuta el algoritmo de *Differential Evolution* varias veces para obtener diferentes resultados.

- **Ejecución 1:** $K_p = 1.1902$, $K_i = 2.5630$ y $K_d = 0.0359$.

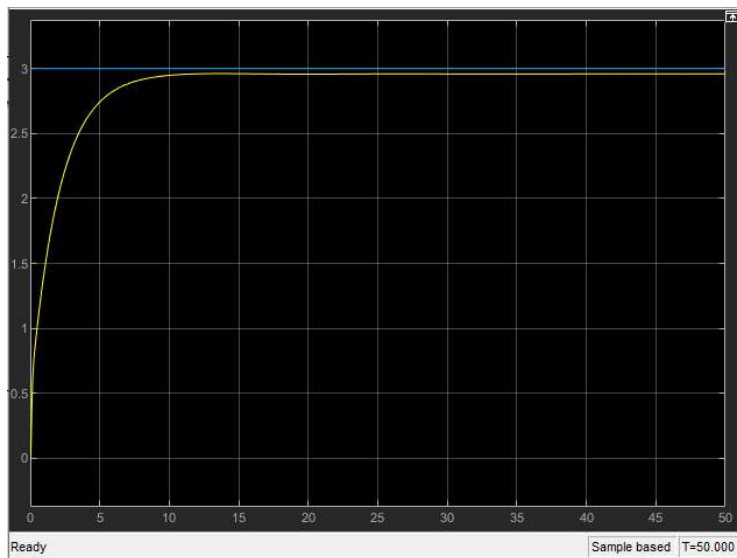


Figura 10: Ejecución 1 algoritmo DE con bloque SOF

- **Ejecución 2:** $K_p = 2.8822$, $K_i = 2.8902$ y $K_d = 0.0801$.

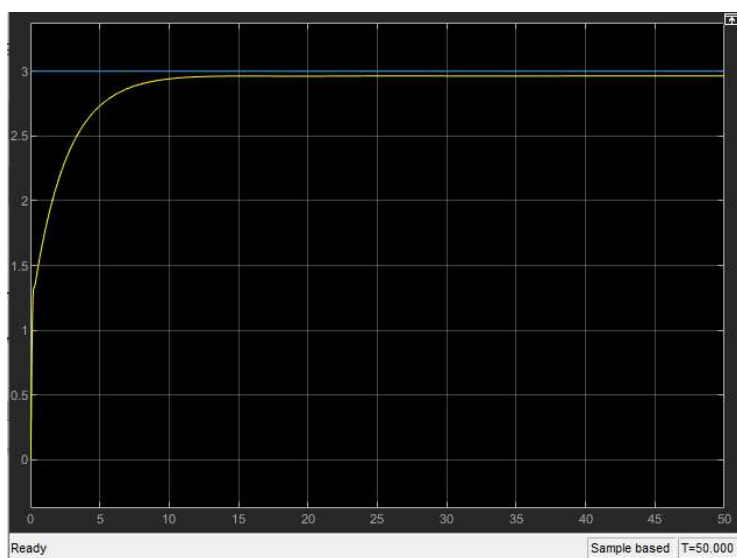


Figura 11: Ejecución 2 algoritmo DE con bloque SOF

- **Ejecución 3:** $K_p = 1.3056$, $K_i = 3.3404$ y $K_d = 0.0229$.

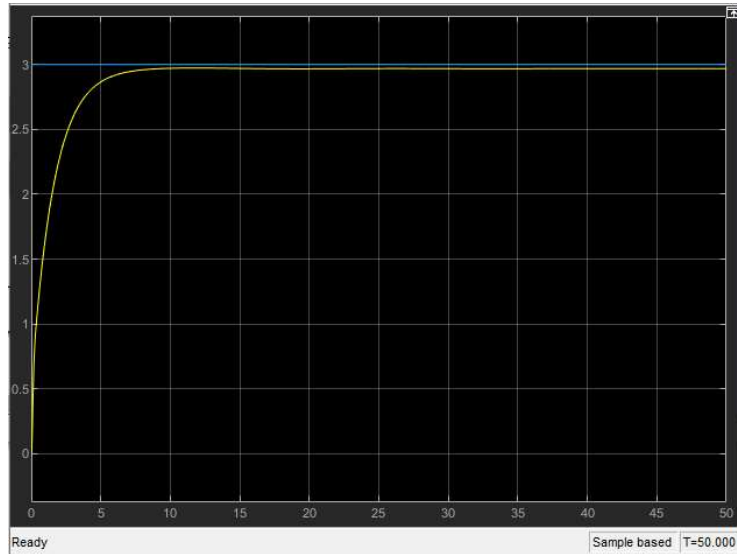


Figura 12: Ejecución 3 algoritmo DE con bloque SOF

Con estos ejemplos se puede ver cómo ahora, al haber incluido el control estático con el bloque SOF, la respuesta del sistema es más lenta y, por tanto, realizable. Sin embargo, se observa que ahora el tiempo de establecimiento es mayor y que 50s no es suficiente para que el sistema alcance el régimen permanente. Se comprobó, por ejemplo con los valores del último PID, que ahora el tiempo de establecimiento está alrededor de 3000s (figura 13).

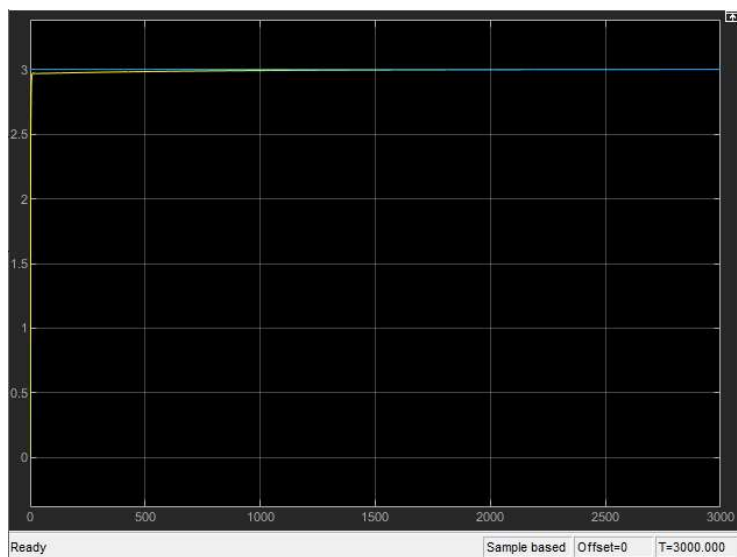


Figura 13: Tiempo de establecimiento con bloque SOF

Por último se quiso también probar la respuesta del sistema ante entrada escalón, con los valores del último PID, obteniéndose el siguiente resultado (figura 14).

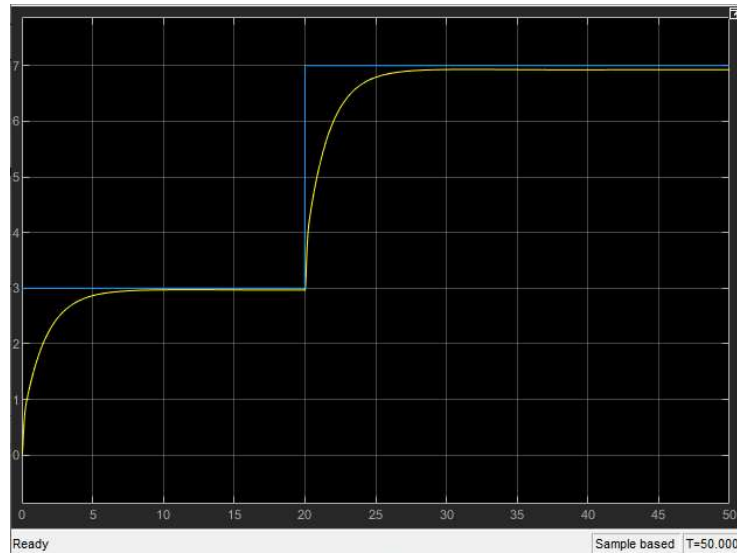


Figura 14: Respuesta ante escalón con bloque SOF

3. Conclusiones

Con el desarrollo de este trabajo se ha comprobado la necesidad de incluir un bucle de realimentación estática para el correcto control del helicóptero. También se ha visto cómo el algoritmo de *Differential Evolution* ha sido capaz de encontrar unos valores de PID óptimos. Además, con los valores de PID obtenidos en estas últimas pruebas, se puede confirmar que un controlador PI junto con el bloque SOF sería suficiente para controlar el sistema, ya que la constante K_d del regulador PID siempre ha resultado en valores muy próximos a 0.

Referencias

- [1] Multiloop Control of a Helicopter, Recuperado de <https://es.mathworks.com/help/control/examples/multi-loop-control-of-a-helicopter.html>