

INFORME DE PRÁCTICAS



UNIVERSIDAD COMPLUTENSE MADRID

Control de un motor de corriente continua

Sistemas Dinámicos y Realimentación

Grado en Física

Javier Peña y Tristán Ortiz

15 de enero de 2026

Índice

1. Introducción de la práctica	3
1.1. Breve explicación del documento	3
2. Descripción del <i>hardware</i> y software empleado	4
2.1. Hardware empleado	4
2.1.1. Identificación componentes	4
2.1.2. Comprobación configuración del Hardware	5
2.2. Software empleado	5
3. Manejo del motor de corriente continua en lazo abierto	6
3.1. Modelo para manejo del motor en lazo abierto	6
3.2. Obtención de expresión para la posición angular de un motor en continua	9
3.3. Obtención de los parámetros	9
3.4. Comparación del modelo ideal con el sistema real	11
3.5. Simulación/ejecución del modelo completo del motor en lazo abierto	12
4. Control del motor por realimentación de estados estimados	14
4.1. Realimentación de estados estimados y acción integral	14
4.1.1. Modelo del motor	14
4.1.2. Efecto de diferentes variables sobre el modelo	16
4.2. Diseño de un sistema de control discretizado para el motor, basado en realimentación de estados	21
4.2.1. Discretización del sistema de control	21
4.2.2. Modelo del sistema de control discretizado	23
4.2.3. Efecto de diferentes variables sobre el sistema de control	23
4.3. Sistema de control completo, modelado y aplicado sobre el motor real	26
5. Conclusiones	28
6. Anexo	29
6.1. Código o funciones de MATLAB	29

1. Introducción de la práctica

1.1. Breve explicación del documento

A continuación se presenta el Informe de la práctica *Control de un motor de corriente continua*, de la asignatura *Sistemas Dinámicos y Realimentación*. Esta práctica se ha realizado siguiendo el guión de la misma [1] a lo largo de diferentes sesiones durante el cuatrimestre. No obstante, a pesar de la división en "prácticas" del guión, se tratará todas las prácticas como una sola, por una cuestión de coherencia y linealidad de contenidos durante el informe.

En cuanto a los modelos usados y otros archivos relevantes, como los livescripts/scripts (.mlx/.m) usados para estudiar los modelos están disponibles en un repositorio de Github [3] para su consulta y reproducibilidad.

2. Descripción del *hardware* y software empleado

2.1. Hardware empleado

2.1.1. Identificación componentes

- **Motor:** El motor usado es de corriente continua (pensado para trabajar a un voltaje nominal de 12 V) con una reductora de relación 30 : 1. También cuenta con encoders de efecto Hall en cuadratura que permiten medir la posición y velocidad angular del eje.
- **Unidad de Control (STM32 Nucleo):** Se utiliza una placa de desarrollo basada en el microcontrolador STM32 (Nucleo-F411RE). Su función es ejecutar el algoritmo de control generado desde Simulink, leer los pulsos de los encoders y generar la señal PWM de control.
- **Etapa de Potencia (Driver):** Placa de expansión conectada sobre la Nucleo que incorpora un puente en H. Se encarga de amplificar las señales lógicas (3,3 V) del microcontrolador y suministrar la potencia necesaria (12 V) al motor, permitiendo el cambio de sentido de giro.
- **Fuente de Alimentación:** Fuente externa regulada a 12 V que alimenta exclusivamente la etapa de potencia y el motor, separada de la alimentación lógica del microcontrolador (USB).

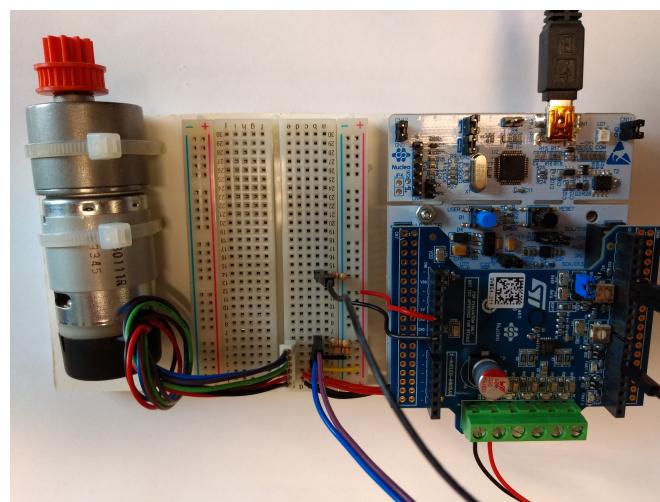


Figura 1: Vista del montaje completo [1]

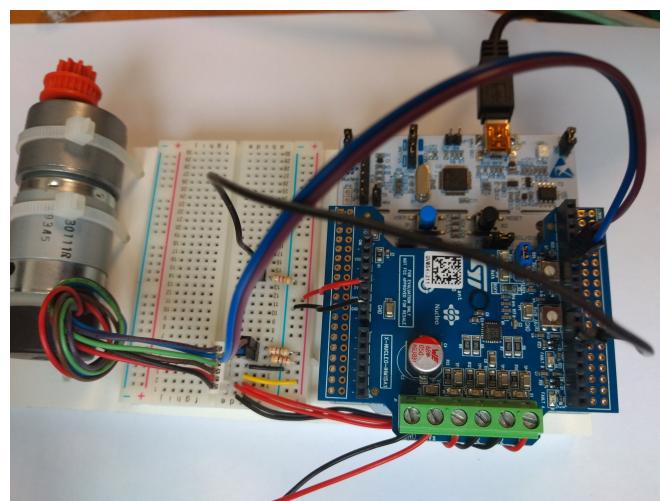


Figura 2: Segunda vista del montaje completo [1]

2.1.2. Comprobación configuración del Hardware

Antes de proceder con la toma de datos se ha realizado el montaje adecuado descrito en el guion: se ha comprobado el correcto acoplamiento de la placa de expansión sobre la Nucleo y la separación de las fuentes de alimentación (5 V para lógica vía USB y 12 V para potencia externa). Asimismo, se ha validado que el conexionado físico coincide con la asignación de pines del modelo de Simulink (PWM en D5, Dirección en D3 y Encoders en A0/A1).

2.2. Software empleado

Para la realización de la práctica, se ha usado Matlab, un software de cálculo y lenguaje de programación de alto nivel. Concretamente, Matlab ofrece un entorno llamado Simulink, en el que construir modelos de sistemas físicos e interactuar con la placa de desarrollo. Además, se ha instalado un *Add-on* específico para trabajar con placas de la familia NUCLEO [2]. Esto nos permitirá usar una librería con bloques con los que interactuar directamente con la placa de desarrollo, que usaremos para construir el modelo del bloque del motor, de la fig. 15. Asimismo, esta librería instalará software para "traducir"los modelos de Simulink a programas escritos en C/C++, así como un planificador de la ilbrería para ejecutar correctamente los modelos en la placa.

3. Manejo del motor de corriente continua en lazo abierto

3.1. Modelo para manejo del motor en lazo abierto

En las siguientes figuras se muestra el modelo de Simulink usado para simular el manejo del motor en lazo abierto. Los bloques de entrada en la figura 1 determinan la magnitud del voltaje y su signo, lo que determina la velocidad de giro del motor y su sentido. Los bloques de salida permiten observar la señal de la posición y de la velocidad del motor. El bloque motor, mostrado en la figura 2, se compone de varios bloques que envían las señales de control a los pines del driver del motor (puente en H). Además, contiene bloques de lectura que recogen los datos de los sensores y los envían al bloque 'Lector Encoder' (figura 3), el cual traduce las lecturas en posición y velocidad angular.

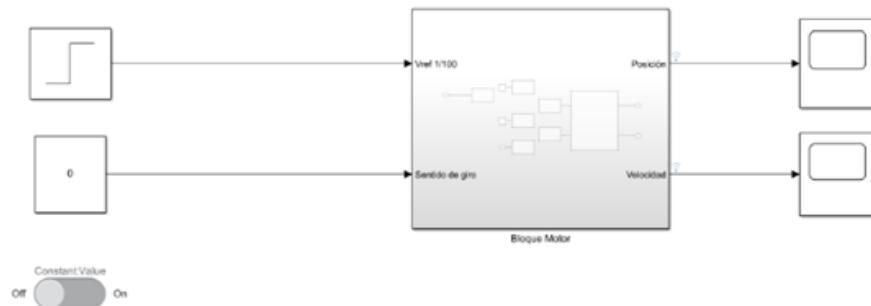


Figura 3: Modelo completo del motor ideal en lazo abierto

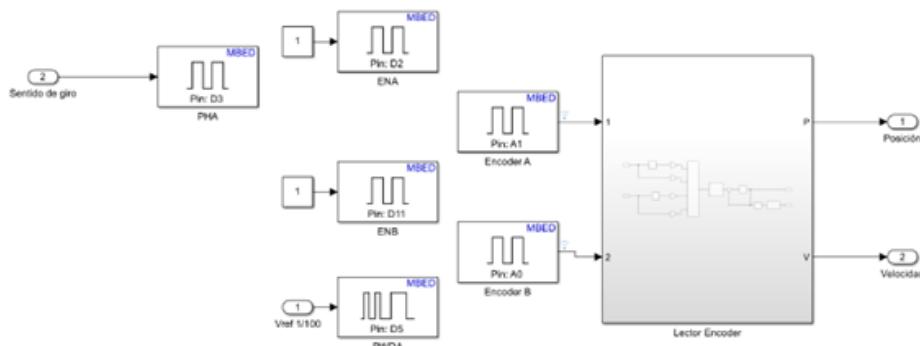


Figura 4: Modelo del bloque motor

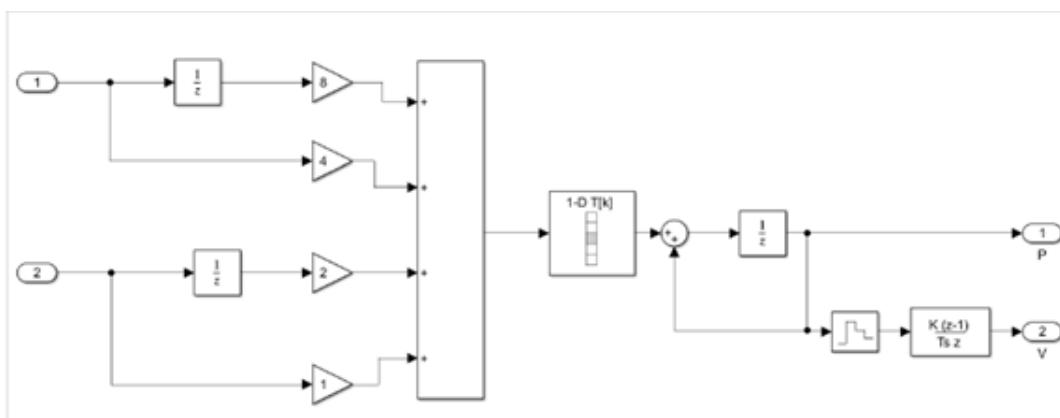


Figura 5: Modelo del Lector Encoder

Una vez conectados al motor, ejecutando el modelo mostrado se obtuvieron los siguientes valores de posición y velocidad para diferentes valores de voltaje suministrado. Aunque en el laboratorio tomamos valores desde 2 a 12 voltios (voltaje máximo que se puede suministrar al motor), por brevedad mostramos solo tres de las mediciones:

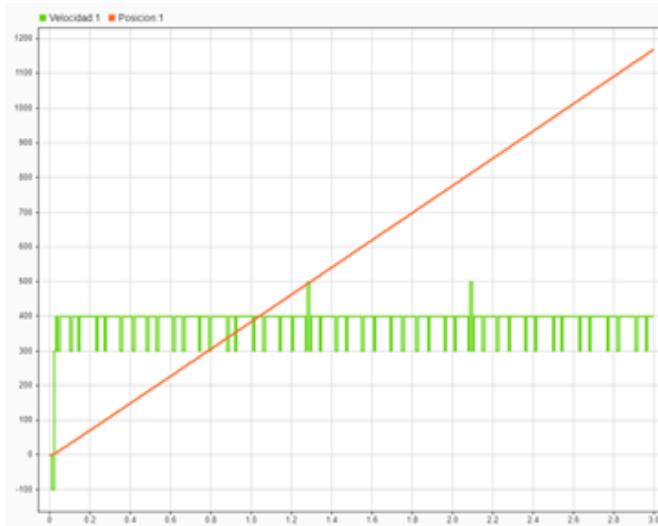


Figura 6: Registro posición y velocidad del motor ideal para 4V

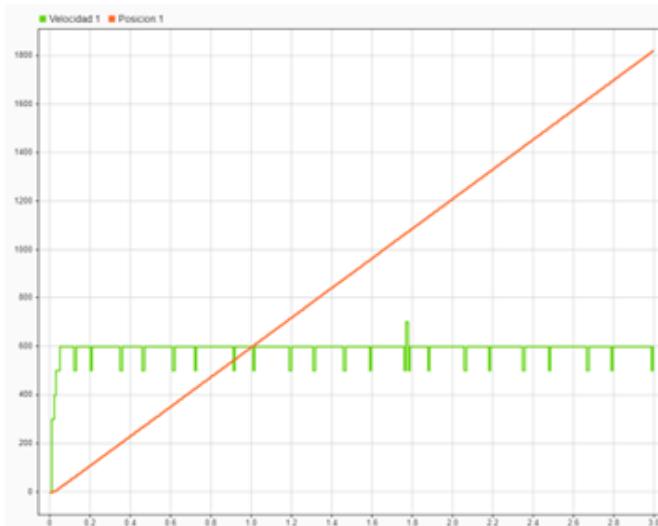


Figura 7: Registro posición y velocidad del motor ideal para 6V

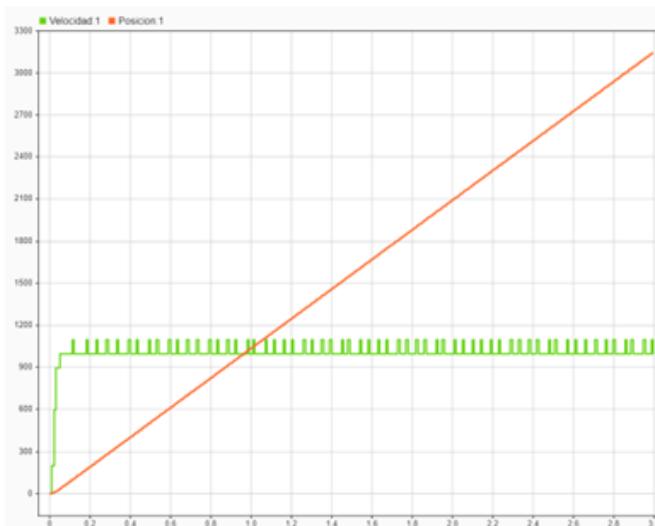


Figura 8: Registro posición y velocidad del motor ideal para 10V

En todos los casos se aprecia a simple vista una tendencia lineal, aunque más adelante se estudiará el comportamiento del motor con más detalle durante los primeros instantes de la medición. Una vez estabilizada, la velocidad angular se mantiene constante. También cabe destacar que el valor de la pendiente de la posición y el valor en el que se estabiliza la velocidad son mayores cuanto mayor es el voltaje suministrado.

Durante la práctica también se registró la lectura de los encoders, mostrando una señal que alterna entre 0 y 1 a intervalos constantes. Cabe destacar que la alternancia es más rápida cuanto mayor es el voltaje aportado, consecuencia de la mayor velocidad angular del motor. En las siguientes figuras se representan las mediciones para 4 y 12 voltios durante un mismo intervalo de tiempo, apreciándose esta diferencia:

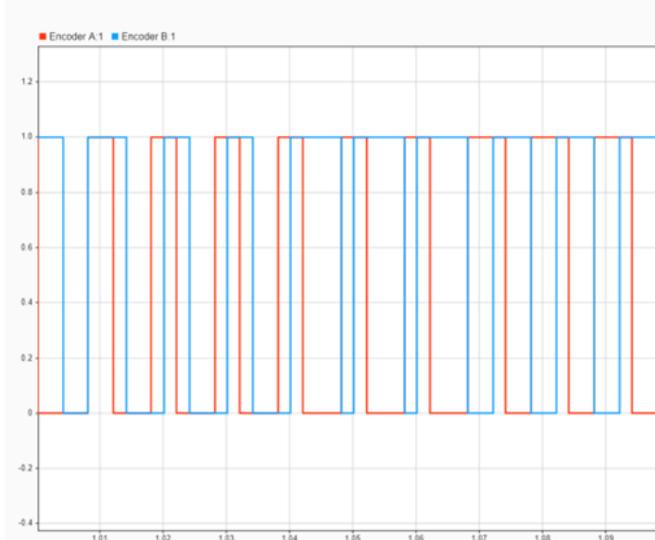


Figura 9: Lectura encoders para 4V

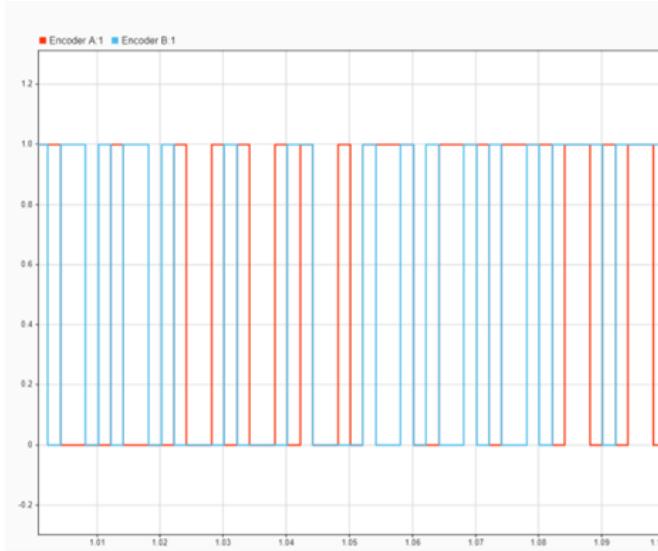


Figura 10: Lectura encoders para 12V

3.2. Obtención de expresión para la posición angular de un motor en continua

Tal y como se indica en el guion, el motor de corriente continua se puede modelar considerando una parte eléctrica y otra mecánica. La dinámica final del sistema se obtiene aplicando dos hipótesis simplificadoras: la dinámica eléctrica (autoinducción) es mucho más rápida que la mecánica y puede considerarse instantánea; y segundo, que el motor se enfrenta únicamente a su rozamiento interno. Bajo estas condiciones, el modelo en el espacio de estados queda definido por:

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -p \end{bmatrix} \begin{bmatrix} \theta \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ k_e \end{bmatrix} V$$

Donde $\theta(t)$ es la posición angular, $\omega(t)$ la velocidad angular, V el voltaje suministrado; k_e , p son los parámetros que caracterizan al motor.

Para obtener la expresión analítica de la posición angular $\theta(t)$, se ha utilizado el motor de cálculo simbólico de MATLAB. A continuación, se detalla el algoritmo implementado para dicha resolución, disponible en el archivo adjunto 'P2_AnalisisDatos mlx'.

```

1 % --- PARTE 1: Modelo en variables de estado ---
2 syms k_e p V w(t)
3 ec1 = diff(w,t) == -p*w + k_e*V;
4 w_0 = w(0) == 0;
5 omega(t) = simplify(dsolve(ec1, w_0))
6
7 syms theta(t)
8 ec2 = diff(theta,t) == omega(t);
9 theta_0 = theta(0) == 0;
10 theta(t) = simplify(dsolve(ec2, theta_0))

```

Así pues, la expresión obtenida de la posición angular es:

$$\theta(t) = V \frac{k_e}{p^2} e^{-pt} + V \frac{k_e}{p} t - V \frac{k_e}{p^2}$$

3.3. Obtención de los parámetros k_e y p

En la expresión obtenida de $\theta(t)$, el término exponencial representa la inercia mecánica del rotor y define un régimen transitorio en la respuesta. Como este término decae a cero para valores de t elevados, $\theta(t)$ acaba adoptando un comportamiento lineal, que representa la solución estacionaria donde el motor gira a velocidad constante, tal y como observamos durante el control del motor en lazo abierto. Aprovechando este estado estacionario, se pueden obtener los parámetros del motor k_e y p a partir de los parámetros del ajuste lineal (siendo m la pendiente y n la ordenada en el origen):

$$p = -\frac{m}{n} \quad k_e = \frac{mp}{V} = -\frac{m^2}{nV}$$

Durante la toma de medidas el motor siempre empezaba en reposo, por lo que el término inercial no es despreciable durante los primeros instantes. Por tanto, se deben retirar algunos datos iniciales para asegurar el comportamiento lineal. Dado que la constante de tiempo estimada de e^{-pt} es muy pequeña ($\tau \approx 1/p = 23$ ms), la influencia de la inercia desaparece en las primeras décimas de segundo. Así que quitando el 20% inicial de datos (equivalente a 0,6s) ya contaríamos con un ajuste lineal válido. Sin embargo, analizando los datos nos dimos cuenta que los

resultados de p_{medio} y de k_e medio varían ligeramente si realizamos el ajuste lineal para la primera parte de los datos o para los últimos. Cálculos disponibles en el archivo adjunto 'P2_AnálisisDatos'. Por ejemplo, en la siguiente tabla se muestra cómo varían los resultados en la medición de 8V:

% considerado	m	n	p	k_e
20-100	828.47	-14.85	55.8	5778
20-40	828.25	-14.50	57.0	5904
40-70	828.31	-14.72	56.3	5827
70-100	829.00	-16.10	51.3	5313

Tabla 1: Parámetros ajuste medición 8V

Con esta comparación se aprecia que durante toda la medición se tiene un ajuste lineal con parámetros muy similares (resultado apreciable a simple vista con ver la recta de la evolución de la posición). Pero pequeñas variaciones del valor de n afectan de forma apreciable a k_e y p debido a la dependencia entre las magnitudes: $p, k_e \propto \frac{1}{n}$

Finalmente se tomaron los datos a partir del 20% inicial hasta el final, para obtener el ajuste más preciso posible tras las consideraciones comentadas. Se calcularon los valores correspondientes de k_e y de p para la aplicación de voltajes promedios de 2, 4, 6, 8, 10 y 12 voltios durante 3 segundos. Las constantes obtenidas para 2V son considerablemente menores al resto, tal y como se muestra en la siguiente figura. Esto se debe a una mayor influencia de rozamientos internos e interferencias en el motor para un voltaje tan pequeño.

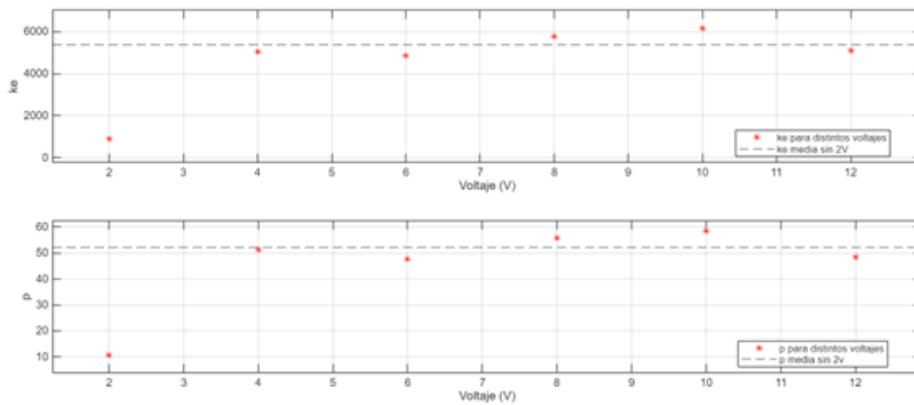


Figura 11: Representación parámetros del motor obtenidos para diferentes voltajes

Por tanto, se excluyó la medición de 2V para calcular la media de los parámetros, obteniéndose finalmente los parámetros del motor que se usarán durante el resto de la práctica:

$$p_{medio} = 46,3 \quad k_e \text{ medio} = 4729$$

3.4. Comparación del modelo ideal con el sistema real

Una vez determinados los parámetros k_e y p de nuestro motor, podemos simularlo con simulink. En la siguiente figura se muestra el modelo ideal del motor.

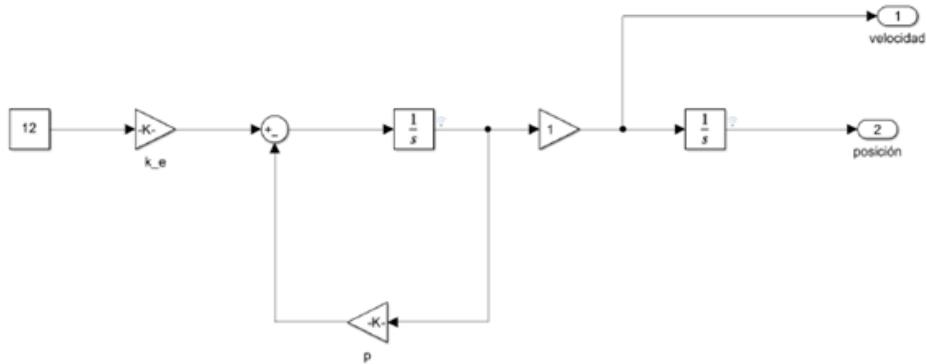


Figura 12: Modelo ideal de motor

En las siguientes figuras se muestra la comparación de los valores obtenidos frente a los simulados. Cabe destacar que la posición estimada crece ligeramente más rápido que la real para valores pequeños del voltaje (ejemplificado con 4V), mientras que para voltajes algo más elevados (ejemplificado con 10V) ocurre lo contrario.

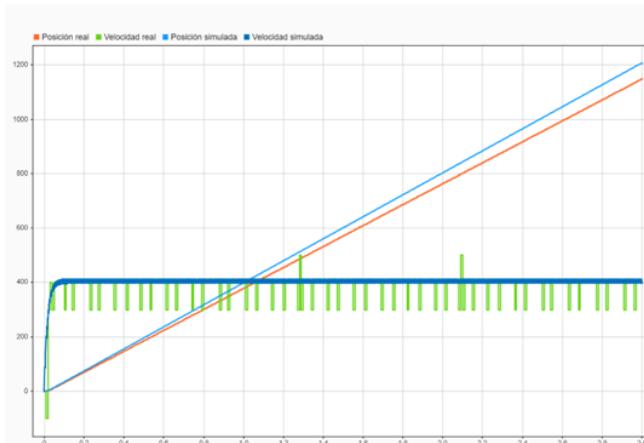


Figura 13: Comparación valores posición y velocidad angular reales frente a valores simulados por el motor ideal. 4V

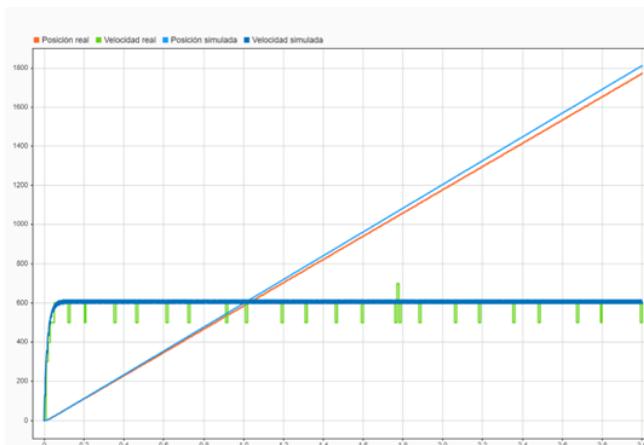


Figura 14: Comparación valores posición y velocidad angular reales frente a valores simulados por el motor ideal. 6V

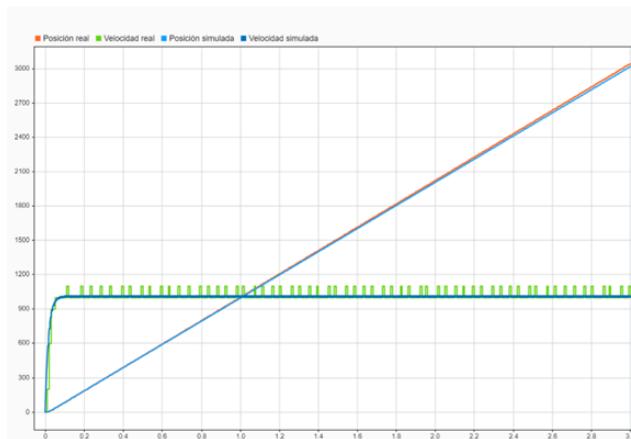


Figura 15: Comparación valores posición y velocidad angular reales frente a valores simulados por el motor ideal. 10V

3.5. Simulación/ejecución del modelo completo del motor en lazo abierto

A diferencia del motor real, en el modelo ideal empleado no se representa ni la alimentación del motor a través de una señal PWM, ni la obtención de las mediciones de posición y velocidad a partir de las lecturas de los encoders. En la siguiente figura se muestra la implementación de estas características, obteniendo un modelo de simulink completo del motor.

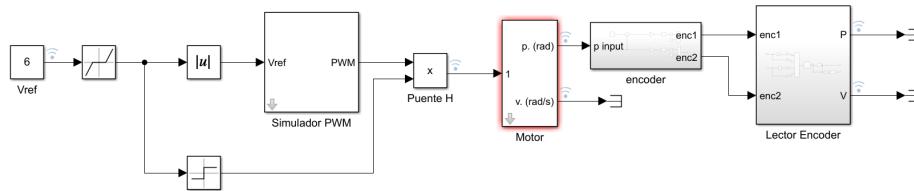


Figura 16: Modelo completo del motor en lazo abierto

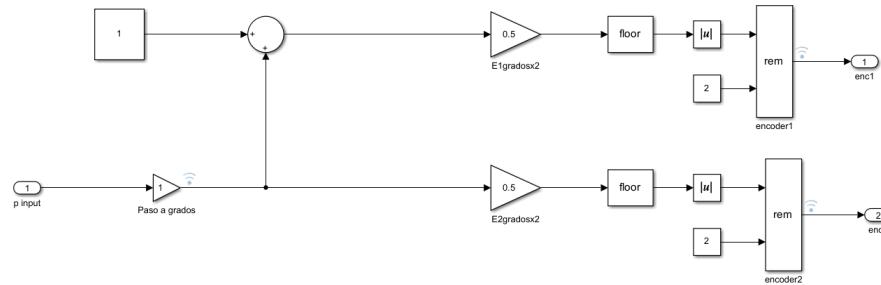


Figura 17: Encoder

En las siguientes figuras se muestran los resultados obtenidos al simular el modelo completo del motor frente a las medidas tomadas del motor real. Se adjunta además una imagen ampliada para que se pueda apreciar que el crecimiento de que la posición ocurre a través de pequeños saltos discretos, tal y como cabe esperar por las lecturas de los encoders.

Cabe destacar que las medidas del motor real tienen una evolución ligeramente más errática, pero de todas formas se aprecia la evolución a partir de pequeños saltos discretos, pudiéndose concluir que la modelización realizada ha sido exitosa.

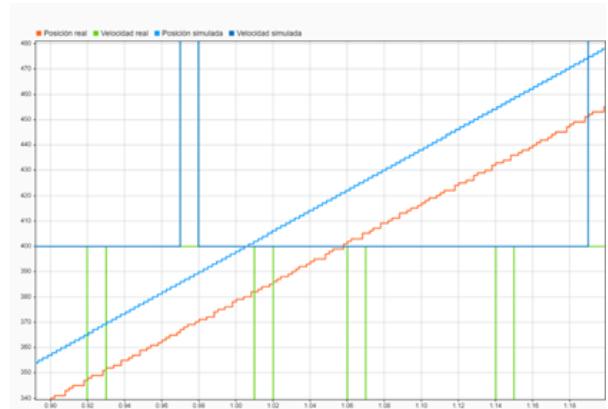
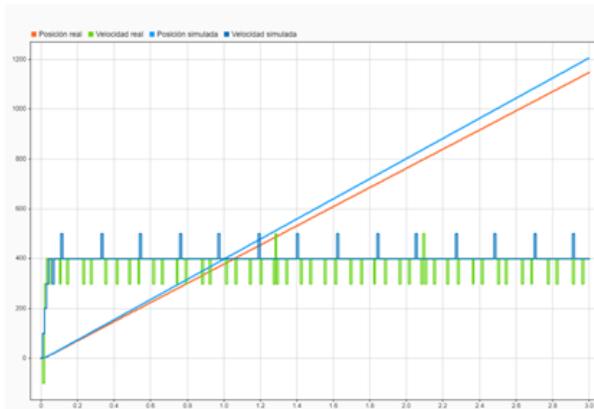


Figura 18: Lectura del encoder para una tensión de 4 V

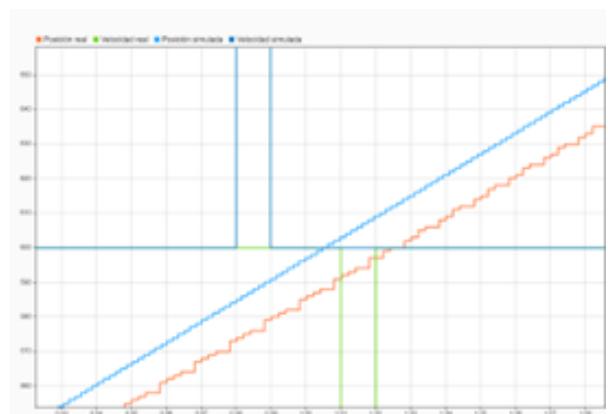
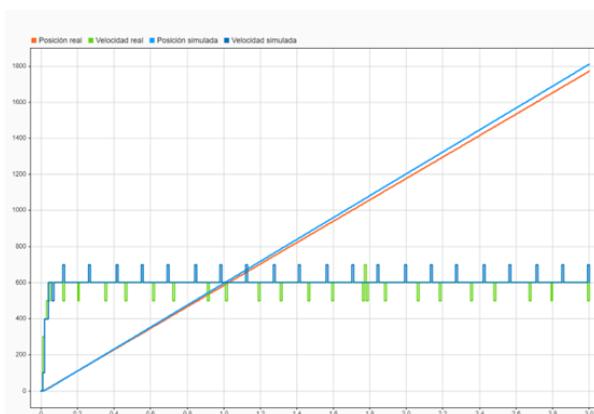


Figura 19: Lectura del encoder para una tensión de 6V

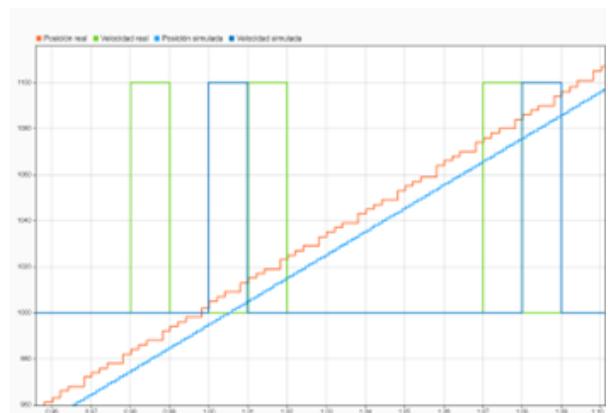
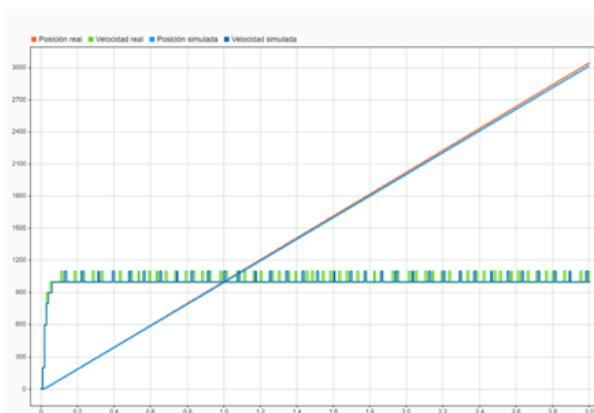


Figura 20: Lectura del encoder para una tensión de 10V

4. Control del motor por realimentación de estados estimados

4.1. Realimentación de estados estimados y acción integral

A continuación, a partir del bloque del motor ideal ?? ya construido, procederemos a construir un sistema de control basado en realimentación de estados estimados y acción integral.

4.1.1. Modelo del motor

Para construir dicho modelo, lo primero que haremos será crear un modelo de realimentación directa del sistema, cuyo esquema de Simulink se incluye en la fig. 21.

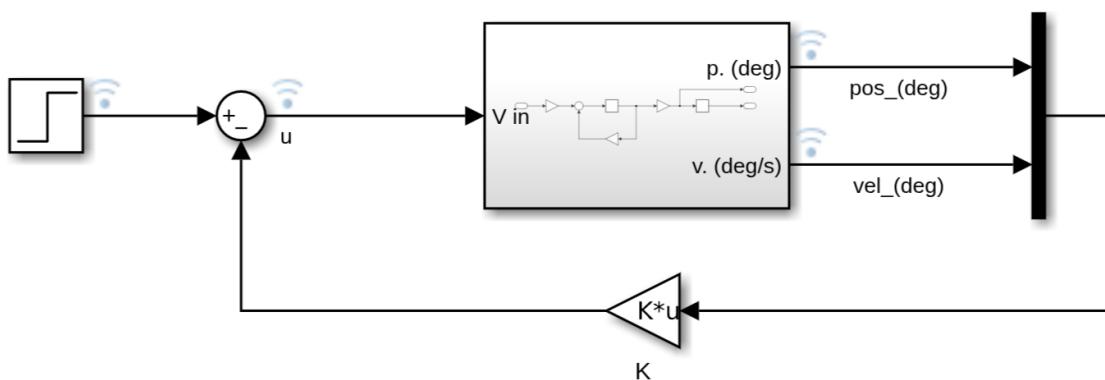


Figura 21: Modelo de realimentación directa del sistema

Antes de nada, los polos del sistema los situaremos en el semiplano negativo complejo para que el sistema sea estable. Debemos tener en cuenta que por un lado, dentro de dicho semiplano negativo complejo, situar los polos más hacia la izquierda implica que el motor va a requerir mayor V_{ref} durante el tiempo que tarde en estabilizarse. Y por el otro, que cuanto más cercanos al eje imaginario escogamos los polos, la respuesta del sistema podría ser tan lenta que las tensiones que demande sean tan bajas que el motor real entre en su zona muerta, no alcanzando el estado estacionario. Más adelante se analizará el efecto de la posición de los polos sobre el sistema realimentado.

El siguiente paso que podemos seguir es implementar un estimador de estados. Para ello, debemos de comprobar que el sistema es observable:

```

1 clear all; clc;
2 %valores de ke y p obtenidos antes
3 p = 46.3
4 ke = 4.729e+3
5
6 %defino las matrices del sistema
7 A = [0 1; 0 -p];
8 B = [0; ke];
9 C = [1 0];
10
11 Oo = obsv(A,C);
12 rank(Oo)

```

Como el rango de la matriz de observabilidad es 2, podemos ver que es observable. Por lo que sí podemos implementar un estimador en nuestro modelo, consiguiendo así una realimentación de estados estimados, representada en la fig. 22.

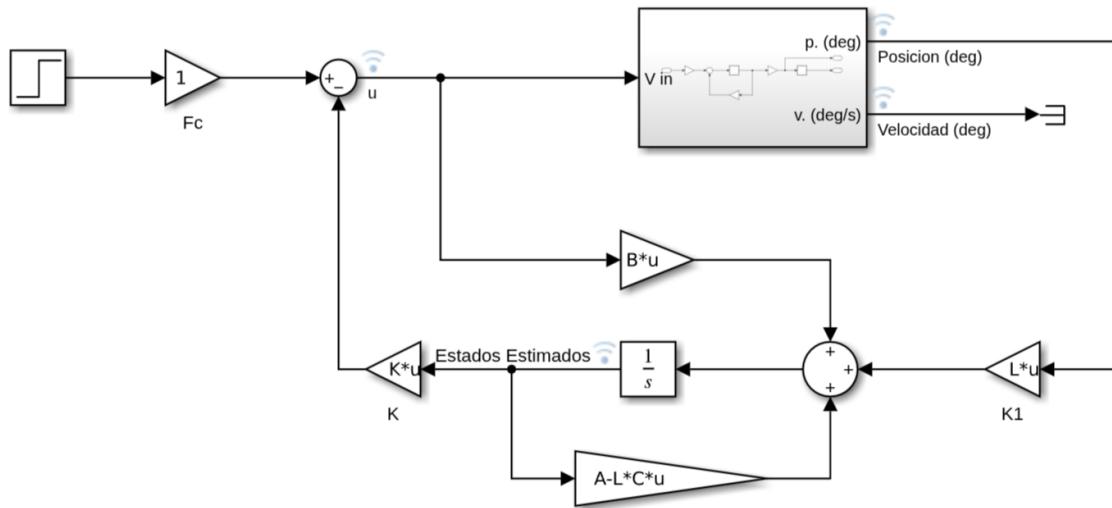


Figura 22: Modelo del sistema realimentado por estados estimados

Finalmente, ya estamos listos para añadir el control integral y completar así el sistema de control que queríamos implementar.

Construimos entonces las matrices ampliadas y comprobamos que sea un sistema controlable con el siguiente código de MATLAB. Como el rango sale de 2, el sistema ampliado es controlable.

```

1 %definimos sistema ampliado
2 Amp = [A zeros(2,1);C 0];
3 Bamp = [B;0];
4
5 control = ctrb(Amp ,Bamp);
6 rank(control)

```

A continuación, se incluye en la fig. 23 un modelo del motor ideal con un sistema de control por realimentación de estados estimados y acción integral.

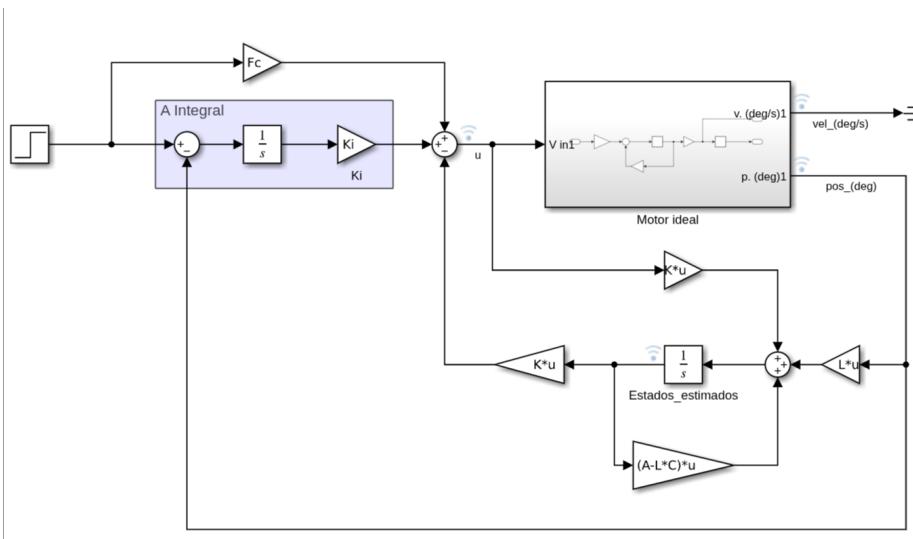


Figura 23: Modelo del motor con sistema de control por realimentación de estados estimados y acción integral

4.1.2. Efecto de diferentes variables sobre el modelo

A continuación, se estudiará el efecto de la posición de los polos de los estados y de los estimadores para el caso más simple de realimentación directa y para el modelo 23 con el estimador e integrador. Después se analizará el efecto de la ganancia de alimentación directa F_c sobre el modelo 23 y finalmente se observará el efecto del error de posición inicial sobre la señal de entrada.

Efecto de la posición de los polos sobre el modelo con realimentación directa:

Para comprobar el efecto de la posición de los polos sobre la velocidad de respuesta y valor de la señal de control u del sistema, podemos recoger los valores de los estados de nuestro sistema realimentado (posición y velocidad) durante la simulación, así como el valor de la señal de entrada al motor u para su posterior análisis. Modificaremos el valor de los polos 3 veces, variándolos en potencias de 10.

Para ello, se ha usado el siguiente código de MATLAB, del archivo 'P4_EstudioPosicionPolos_RealimDir.slx' del repositorio de github [3]:

```

1 clear all; clc;
2 %valores de ke y p obtenidos antes
3 p = 46.3
4 ke = 4.729e+3
5 %defino mis matrices del sistema
6 A = [0 1; 0 -p]
7 B = [0; ke];
8 %defino valores iniciales de posicion y velocidad
9 theta0 = 5;
10 w0 = 2;
11
12 %llamada a funcion cuyo prototipo es:
13 %make_sims_realimdir(A, B, ke, p, theta0, w0, num_sims)
14 make_sims_realimdir(A, B, ke, p, theta0, w0, 3)
```

Listing 1: Simulación del modelo 21 para diferentes posiciones de polos

La función llamada en la linea 16 de 1 está recogida en el Anexo en 7 y hace lo siguiente. Llama al modelo de realimentación directa 21 y lo ejecuta, guarda los datos de posición, velocidad y señal de control, y obtiene una gráfica con la evolución de los estados y otra con la evolución de u . Esto lo hace para el número de veces indicado, modificando las posiciones de los polos en potencias de 10 en cada simulación.

Las posiciones de los polos para cada simulación fueron las siguientes:

Nº Sim	Modificador	Polos	
1	0,01	-0,2315	-0,2315
2	0,1	-2,315	-2,315
3	1	-23,15	-23,15

Tabla 2: Posiciones de los polos del sistema en cada simulación de realimentación directa

Las gráficas obtenidas se recogen a continuación. Los datos de las simulaciones se pueden encontrar en el repositorio de github [3], en Practica4/"P4_RealimDir_MOD.mat"(MOD representa el factor de modificación de los polos: 0,01; 0,10; 1,00; 10,00).

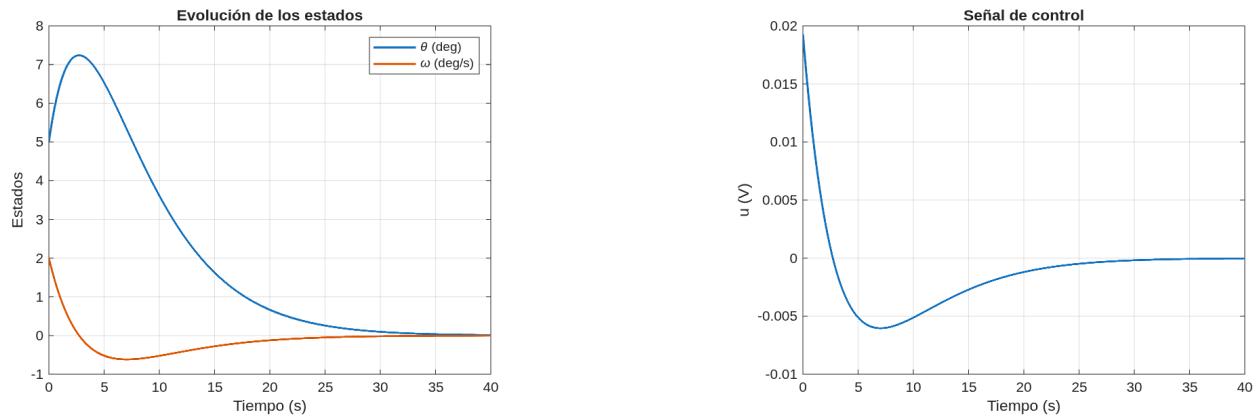


Figura 24: Evolución de posición y velocidad y de la señal de control u en sistema con realimentación directa para polos en $-0,2315$.

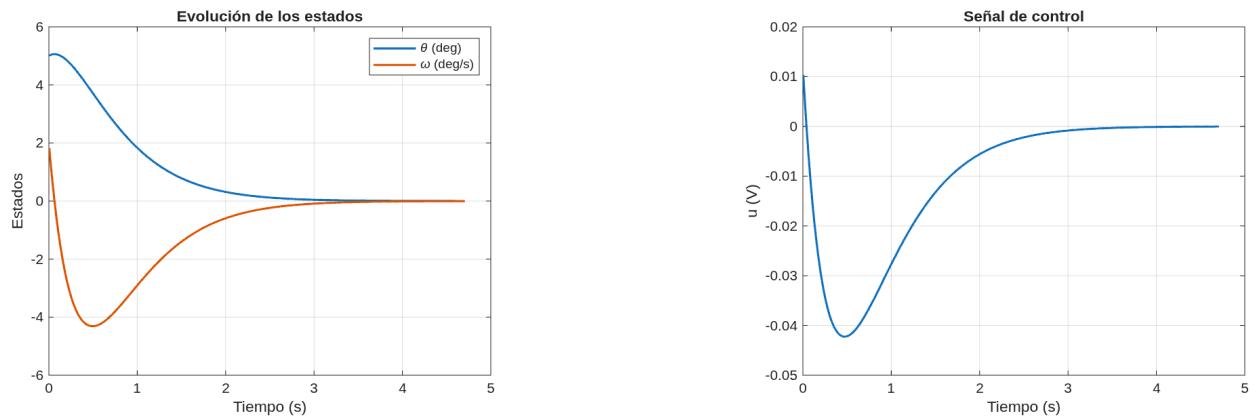


Figura 25: Evolución de posición y velocidad y de la señal de control u en sistema con realimentación directa para polos en $-2,315$.

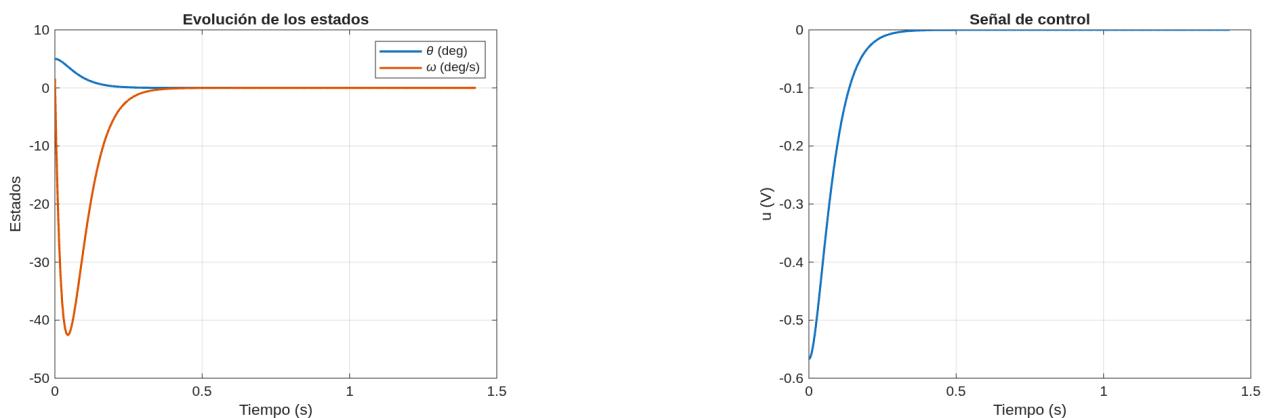


Figura 26: Evolución de posición y velocidad y de la señal de control u en sistema con realimentación directa para polos en $-23,15$.

De las gráficas presentadas (figs. 24, 25 y 26), se puede comentar lo siguiente. Hemos de notar que los valores iniciales de posición y velocidad fueron 5° y $2^\circ/\text{s}$ para todas las simulaciones. La entrada del bloque escalón ha sido 0 para todas estas simulaciones, que es el valor en el que debía estabilizarse la posición. En estas simulaciones las condiciones iniciales se han mantenido constantes intencionadamente, para poder analizar mejor el efecto de los polos en la evolución del sistema.

Lo más evidente es la diferencia de tiempos en los que cada sistema se estabiliza. Cuanto más a la izquierda situamos los polos (fig. 26), menos tarda en estabilizarse. Esto además se corrobora en las gráficas de la señal de control correspondiente, puesto que los valores de las señales de control con polos más a la izquierda son bastante mayores (en valor absoluto) que los de las señales de control con polos más a la derecha, como la u representada en la fig. 24.

Efecto de la posición de los polos sobre el modelo con realimentación de estados estimados y acción integral:

Se ha hecho el mismo estudio con el sistema de realimentación de estados estimados y control integral, fig. 23. También se han realizado 3 simulaciones para distintas posiciones de polos del sistema y los de los estimadores, recogidos en la siguiente tabla 3.

Los polos del estimador siempre han sido colocados a la izquierda de los polos de los estados para que el estimador converja al valor del estado real antes de que el estado real se estabilice. Asimismo, el polo del integrador se ha colocado siempre a la izquierda de todos ellos.

Nº Sim	Modificador	Polos		Polos estimador		Polo Integrador
1	0,01	-0,2315	-0,2315	-0,3241	-0,3241	-0,6019
2	0,1	-2,315	-2,315	-3,241	-3,241	-6,019
3	1	-23,15	-23,15	-32,41	-32,41	-60,19

Tabla 3: Posiciones de los polos del sistema y estimador en simulaciones de realimentación de estados estimados y control integral

Para ello, se ha usado el siguiente código de MATLAB, del archivo 'P4_EstudioPosicionPolos_EstimInt.slx' del repositorio de github [3]:

```

1 clear all; clc;
2 %valores de ke y p obtenidos antes
3 p = 46.3
4 ke = 4.729e+3
5 %defino mis matrices del sistema
6 A = [0 1; 0 -p]
7 B = [0; ke];
8 C =[1 0];
9 %defino valores iniciales de posicion y velocidad
10 theta0 = 5;
11 w0 = 2;
12 thetaref = 0; %bloque escalon
13 %llamada a funcion cuyo prototipo es:
14 %make_sims_realimdir(A, B, ke, p, theta0, w0, num_sims)
15 make_sims_realimdir(A, B, ke, p, theta0, w0, 4)
```

Listing 2: Simulación del modelo 23 para diferentes posiciones de polos de estados y estimador

La función llamada en la línea 15 del código 2 está recogida en el Anexo en 8. Es análoga a la que se usó en el análisis para realimentación directa. En este caso realizamos 3 simulaciones con los polos descritos en la table 3. Cabe destacar que los valores iniciales siguen siendo los mismos que en las simulaciones de realimentación directa y que el valor de posición de referencia se fijó a cero para estas simulaciones.

Además, cabe señalar que en estas simulaciones se incluye también una ganancia de acción directa o *feedforward* F_c . Dicha ganancia, que lleva el motor a la posición indicada por el bloque escalón de entrada, viene dada por la expresión:

$$F_c = (C(BK - A)^{-1}B)^{-1} \quad (1)$$

En nuestro caso, para las posiciones de los polos de la tercera fila de la tabla 3 (el modificador de p es 1), el *feedforward* tiene un valor de $F_c = 7,0232e - 03$.

Incluimos a continuación las gráficas de las simulaciones. Los datos de dichas simulaciones se pueden encontrar en el repositorio de github [3], en Practica4/'P4_EstimInt_MOD.mat' (MOD representa el factor de modificación de los polos: 0,01; 0,10; 1,00; 10,00).

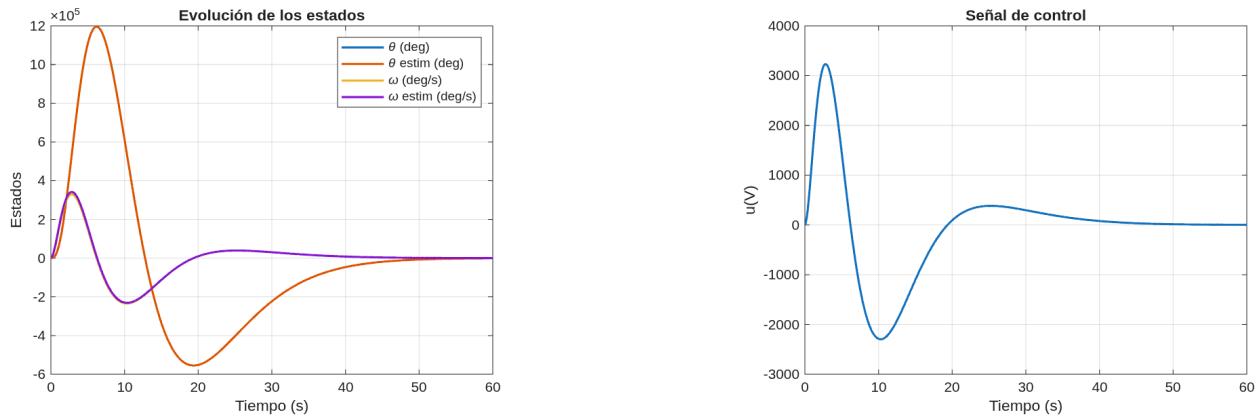


Figura 27: Evolución de posición y velocidad y de la señal de control u para realimentación de estados estimados e integrador con factor de modificación 0,01.

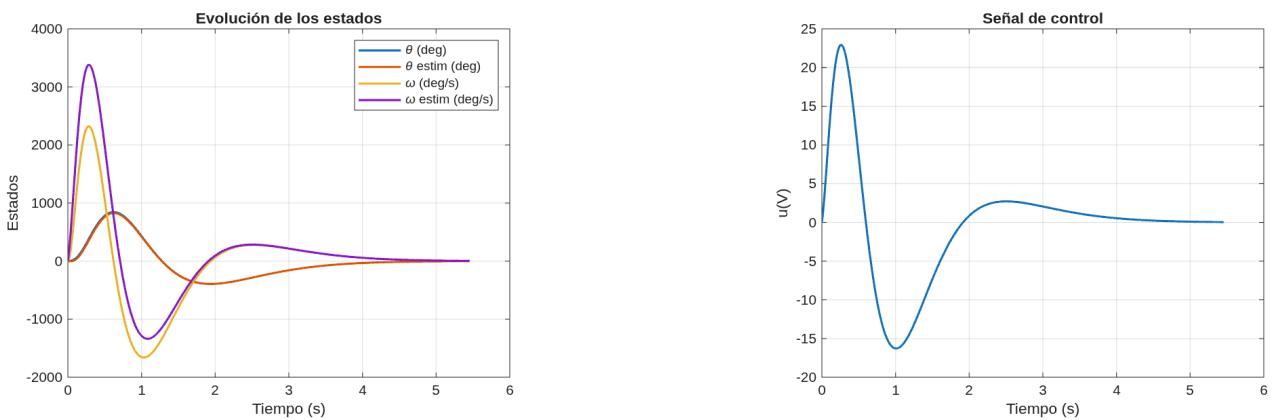


Figura 28: Evolución de posición y velocidad y de la señal de control u para realimentación de estados estimados e integrador con factor de modificación 0,10.

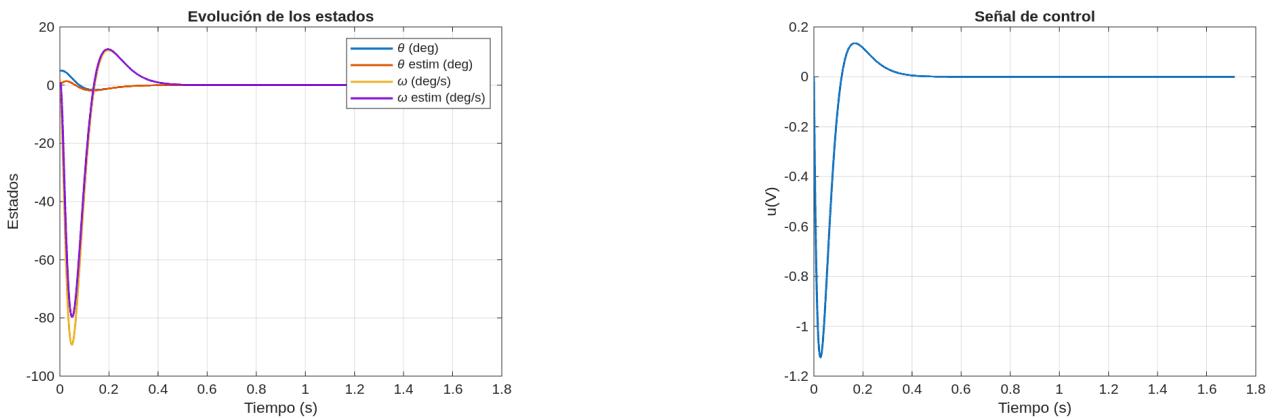


Figura 29: Evolución de posición y velocidad y de la señal de control u para realimentación de estados estimados e integrador con factor de modificación 1,00.

De nuevo, en con el sistema de control de estados estimados y acción integral seguimos apreciando el efecto que tiene la posición de los polos de los estados y los estimadores en la velocidad de respuesta del sistema. En la fig. 29 se aprecia claramente que la señal de control evoluciona muy bruscamente, que se traduce en que la pasa de $2^\circ/s$ a $\approx -60^\circ/s$ en menos de 0,2s, es decir, muy bruscamente. En contraposición, podemos ver el caso de la fig. 27, en la que la posición sigue aumentando hasta pasados los 5s, muy por encima del valor donde debía estabilizarse. Cuando la realimentación y acción consigue hacer efecto para disminuir esa posición, se vuelve a pasar y la posición acaba por debajo del valor objetivo, por lo que la señal de control debe cambiar de sentido de nuevo para hacer que la velocidad sea positiva y desplazarse al valor objetivo: cero. Para cuando se estabilizan, han pasado 60s.

Por otro lado, cabe preguntarse sobre el efecto que tiene el error de la posición inicial sobre la señal de entrada u , que llamaremos $z = \theta_{ref} - \theta(t)$. Este error en la posición inicial se refiere a la diferencia entre la posición actual y la posición de referencia a la que se desea llegar. A priori, podemos pensar que un error de posición muy elevado hará que la señal de control comience siendo elevada. No obstante, debemos tener también en cuenta el error del estimador con respecto al valor real, puesto que la discrepancia entre la posición real y la posición estimada que va a afectar al valor de la señal de entrada u . A medida que la simulación evoluciona y el estimador se acerque al valor real de la posición, esa discrepancia disminuirá. Esto se puede observar en la 2^a y 3^a columna de las gráficas siguientes.

En la gráfica, los valores iniciales y de referencia para la simulación de cada columna han sido, correspondientemente:

<pre> 1 theta0 = 0; 2 w0 = 2; 3 thetaref = 360; 4 %inicialmente 5 %z = 360 </pre>	<pre> 1 theta0 = 350; 2 w0 = 2; 3 thetaref = 360; 4 %inicialmente 5 %z = 10 </pre>	<pre> 1 theta0 = 0; 2 w0 = 2; 3 thetaref = 10; 4 %inicialmente 5 %z=10 </pre>
---	--	---

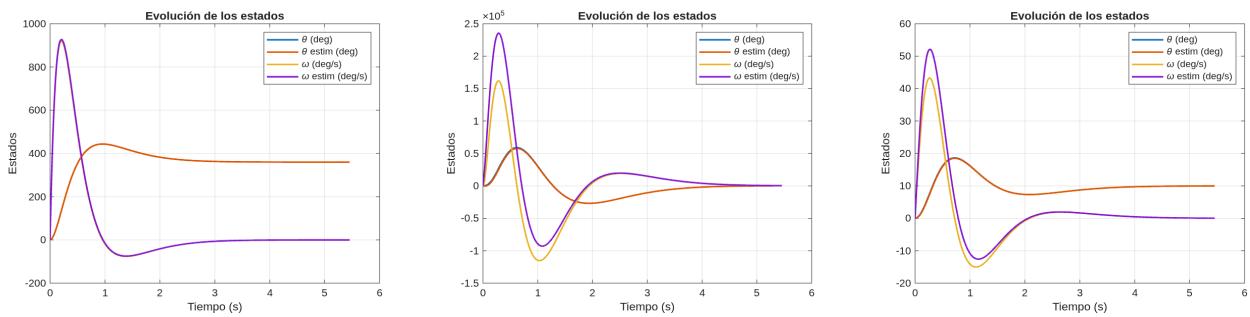


Figura 30: Evolución de estados y estimadores para realimentación de estados estimados e integrador. ($\text{mod_p} = 0.10$).

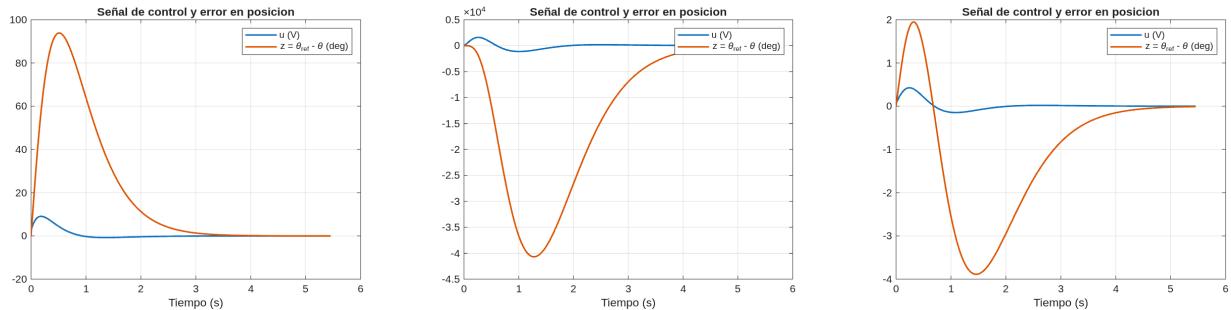


Figura 31: Evolución de la señal de control u y el error en posición z para realimentación de estados estimados e integrador. ($\text{mod_p} = 0.10$).

Por un lado podemos observar que, así como la señal de control se estabiliza en cero (lo cual es esperable), si analizamos de cerca las imágenes de la fig. 31, vemos que el error z no siempre se estabiliza en cero, sino que se converge a un valor fijo cerca del 0. Esto podría ser porque el valor de la posición sea cercano al de referencia pero la diferencia entre los dos (el error z) es tan pequeño que la señal que demanda para corregir la posición es un voltaje de la zona muerta del motor; es decir un voltaje al que el motor no responde debido a la fricción estática del mismo.

Por último, es interesante observar que para errores de posición iniciales muy elevados, como es el caso de la simulación recogida en la 1^a columna de las figs. 30 y 31, hay solamente una oscilación de la señal de control, error de posición e incluso de los propios estados y estimadores. Sin embargo, para un error de posición inicial pequeño (como la 2^a y 3^a columna) aparece más de una oscilación. Es decir, la respuesta que da el sistema es un poco desproporcionada. Probablemente, si se colocasen los polos más a la derecha (pero siempre negativos, ojo), la respuesta del sistema sería menos impetuosa de primeras y no se sobrepondrían.

4.2. Diseño de un sistema de control discretizado para el motor, basado en realimentación de estados

4.2.1. Discretización del sistema de control

En esta sección tratamos de seguir construyendo el sistema de control para el motor real, no para el modelo del mismo. Para ello, se debe tener en cuenta que, dado que en el mundo real el motor es controlado y observado por un sistema digital, la señal de voltaje que recibe el motor, así como la lectura de encoders (la señal de salida) no se realiza de forma continua, sino discretizada.

Esta discretización de las señales está definida por el periodo de muestreo, que llamamos T , al que la placa *ST Nucleo* envía o recibe datos por sus pines.

Por lo tanto, al desarrollar un sistema de control en Simulink para nuestro motor, debemos tener en mente que se trata de un sistema discreto. A continuación, en la eq. (2), definimos un modelo en variables de estado discreto.

$$\begin{aligned} \mathbf{x}(n+1) &= F\mathbf{x}(n) + G\mathbf{u}(n) \\ \mathbf{y}(n) &= C\mathbf{x}(n) + D\mathbf{u}(n) \end{aligned} \quad (2)$$

Donde las matrices F, G, C, D son análogas a las matrices del sistema continuo A, B, C, D . Ha de notarse que las variables de estado han pasado de depender del tiempo a depender del número de muestra n .

$$x(t) \rightarrow x(n)$$

El periodo de muestreo que tomaremos será de

$$T = 0,0001s.$$

Para aplicar una realimentación de estados estimados al modelo real, debemos transformar el estimador continuo 22 por uno discreto. Se debe calcular entonces las matrices F y G a partir de sus recíprocas en el sistema continuo, A y B , y del periodo de muestreo T .

Podemos discretizar nuestro sistema con el siguiente código de MATLAB, con funciones proporcionadas por MATLAB para discretizar un sistema continuo. Este código se puede encontrar en el archivo 'P5_SistControlDiscret.mlx' del repositorio de github [3] o adjuntado con el informe.

```

1 clear all; clc;
2 %valores de ke y p obtenidos antes
3 p = 46.29
4 ke = 4.729e+3
5
6 %defino las matrices del sistema continuo
7 A = [0 1; 0 -p]
8 B = [0; ke];
9 C = [1 0];
10
11 %definimos valores de polos y feedforward:
12 p1 = -1*p;
13 p_int = -1.5*p;
14 Fc = 0.00702;
15
16 %definimos valores iniciales
17 theta0 = 5;
18 w0 = 2;
19 thetaref = 0; % esto es el valor de escalon (que es cte)
20 %% Discretizacion del sistema: %%
21 % tiempo de muestreo T
22 T = 0.0001;
23 sys = ss(A, B, C, zeros(1, 1));
24 sysd = c2d(sys, T);
25 F = sysd.A;
26 G = sysd.B;
27 C = sysd.C;
28 D = sysd.D;
```

Listing 3: Discretización del sistema de control

Pero no nos es suficiente con eso; también debemos discretizar los polos, la acción directa, así como la acción integral [1].

Discretización de los polos:

Para ello, como se indica en el guión de la práctica [1], sabemos que un polo continuo λ se transforma en uno discreto Λ como:

$$\lambda \rightarrow \Lambda = e^{\lambda T} \quad (3)$$

También sabemos que las matrices de ganancia se pueden obtener como:

$$\begin{aligned} K &= \text{acker}(F, G, [\Lambda_{p1}, \Lambda_{p2}, \dots]) \\ K &= \text{acker}(F^T, G^T, [\Lambda_{est1}, \Lambda_{est2}, \dots])^T \end{aligned} \quad (4)$$

Siendo $[\Lambda_{p1}, \Lambda_{p2}, \dots]$ los polos discretizados de los estados del sistema y $[\Lambda_{est1}, \Lambda_{est2}, \dots]$ los polos discretizados de los estimadores.

Podemos discretizar los polos en MATLAB con el siguiente código, de nuevo en el archivo 'P5_SistContrDiscret.mlx':

```

1 polos = [-0.5*p -0.5*p -1.3*p]; %El ultimo corresponde a la accion integral
2 polos_estim = [-0.7*p -0.7*p];
3
4 %Ahora, discretizamos
5 polos_discr = exp(polos*T);
6 polos_estim_discr = exp(polos_estim*T);
```

Listing 4: Discretización de los polos de los estados y estimadores del sistema

Discretización de la acción directa:

La discretización de la acción directa se puede realizar a partir de F_c como se indica en el guión de la práctica [1], o también mediante la ecuación 5, a continuación.

$$f = [C[I - (F - GK)]^{-1}G]^{-1} \quad (5)$$

La implementación de la discretización de la acción directa en MATLAB (en el archivo 'P5_SistContrDiscret.mlx') es trivial:

```

1 % discretizamos la accion directa
2 f = inv(C*inv(eye(2) - (F-G*K))*G);
```

Listing 5: Discretización de la acción directa sobre el sistema

Discretización de la acción integral:

Como tercer y último ingrediente a discretizar para discretizar nuestro sistema de control construido hasta ahora, debemos discretizar la acción integral.

Para ello, se puede partir de la ecuación de estado para la acción integral y discretizarla, como se indica a continuación:

$$\begin{aligned} \dot{z}(t) &= C\mathbf{x}(t) - x_{ref} \Rightarrow \frac{z(n+1) - z(n)}{T} = C\mathbf{x}(n) - x_{ref} \Rightarrow \\ &\Rightarrow z(n+1) = z(n) + TC\mathbf{x}(n) + Tx_{ref} \end{aligned} \quad (6)$$

Con la última expresión de la eq. (6), se construye el sistema ampliado, que exponemos a continuación.

$$\begin{bmatrix} \mathbf{x}(n+1) \\ z(n+1) \end{bmatrix} = \begin{bmatrix} F & 0 \\ TC & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}(n) \\ z(n) \end{bmatrix} + \begin{bmatrix} G \\ 0 \end{bmatrix} u(n) + \begin{bmatrix} 0 \\ -T \end{bmatrix} x_{ref} \quad (7)$$

Así, podemos obtener las matrices ampliadas del sistema discretizado: F_{amp} y G_{amp} y recalcular los valores de las ganancias $[K_{dis}, Ki_{dis}]$. Esto lo hacemos en el siguiente código de MATLAB, recogido en el archivo 'P5_SistContrDiscr.mlx' del repositorio de github [3].

```

1 %matrices ampliadas del sist discretizado
2 Famp = [F zeros(2,1); T*C 1];
3 Gamp = [G;0];
4
5 %matrices de ganancia:
6 Kamp_dis = acker(Famp,Gamp,polos_discr);
7 Kdis = Kamp_dis(1,1:2);
8 Kidis = Kamp_dis(3);
9 L = acker(F',C',polos_estim_discr)';

```

Listing 6: Discretización del sistema ampliado con control integral y obtención de matrices de ganancia

4.2.2. Modelo del sistema de control discretizado

Ya tenemos pues, lo necesario para construir el sistema de control discretizado en Simulink. Para ello, solamente tenemos que coger el modelo desarrollado antes 23 y sustituir las matrices del sistema continuo por las del discretizado, así como las nuevas ganancias K_{dis} . Es importante también cambiar los bloques *integradores* por bloques *UnitDelay* con el tiempo de muestreo que mencionamos anteriormente $T = 0,0001s$.

El modelo discretizado se presenta a continuación, en la fig. 32, el archivo de Simulink correspondiente se llama 'P5_Modelo_SistContrInt.slx' y se puede encontrar adjuntado al informe o en el repositorio de github [3].

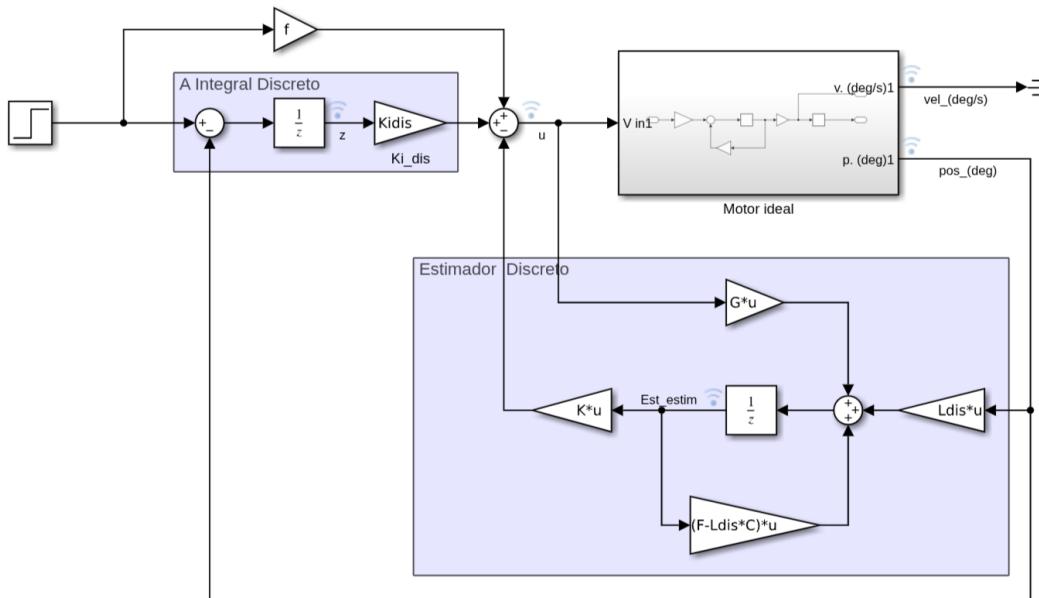


Figura 32: Modelo discretizado con realimentación de estados estimados y control integral.

4.2.3. Efecto de diferentes variables sobre el sistema de control

A continuación, estudiaremos el efecto de la posición de los polos y de la realimentación directa discretizada (*feedforward*) f en la respuesta obtenida para entradas escalón, estudiando también el efecto sobre la entrada del sistema u .

Para ello, ejecutaremos el archivo 'P5_SistContrIntDiscr.mlx', adjuntado con el informe o en el repositorio de github [3] para ejecutar el modelo a continuación.

Esto lo realizaremos para posiciones de los polos más a la izquierda y menos a la izquierda, con realimentación directa discretizada f y sin.

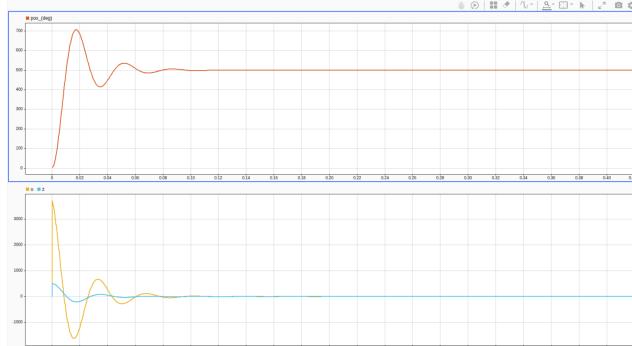
Las simulaciones se han realizado para las siguientes condiciones iniciales:

```

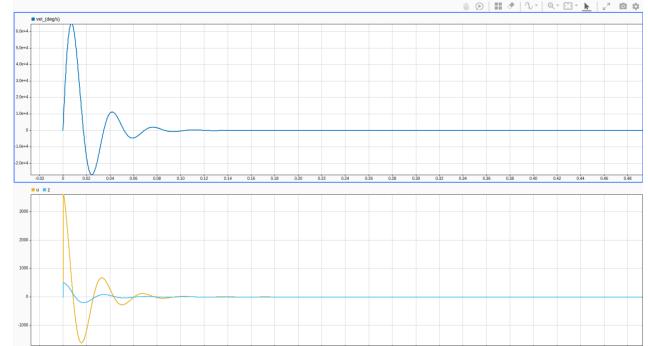
1 %definimos valores
2 theta0 = 5;
3 w0 = 2;
4 thetaref = 500;
```

y los polos colocados han sido en $-0,5p$ y en $-p$.

Las figuras a continuación muestran en cada una la evolución de la posición o velocidad en la mitad superior y la evolución de la señal de control del sistema, u , así como z .



(a) Evolución de posición



(b) Evolución de velocidad

Figura 33: Evolución de posición y velocidad, además de u y z para: $-0,5p$, feedforward = yes

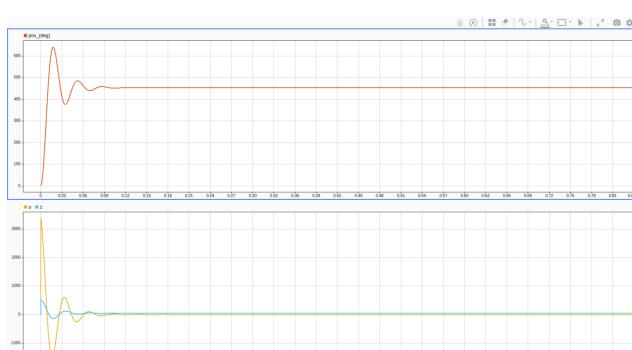


(a) Evolución de posición

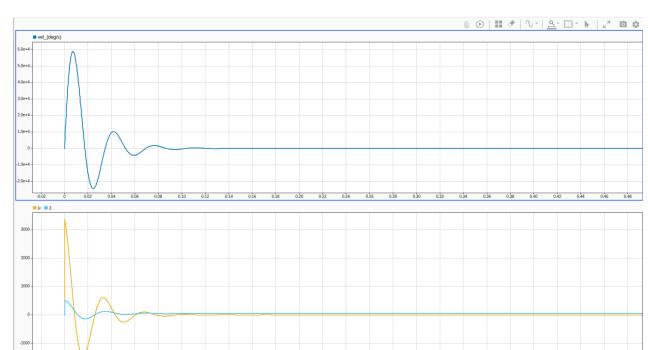


(b) Evolución de velocidad

Figura 34: Evolución de posición y velocidad, además de u y z para: $-p$, feedforward = yes



(a) Evolución de posición



(b) Evolución de velocidad

Figura 35: Evolución de posición y velocidad, además de u y z para: $-0,5p$, feedforward = yes

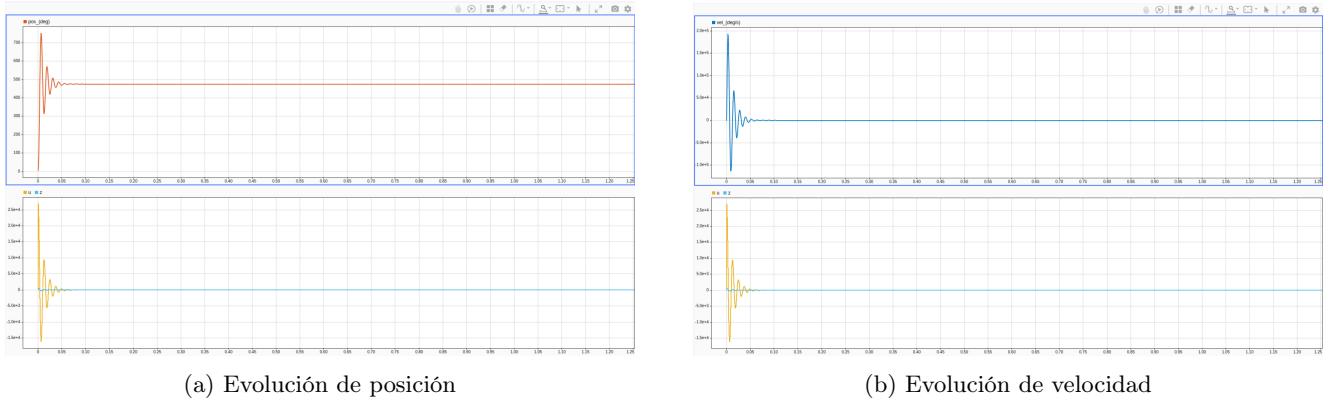


Figura 36: Evolución de posición y velocidad, además de $, u$ y z para: $-p$, feedforward = no

De nuevo, al igual que pasaba con el sistema continuo, que los polos del sistema y estimadores estén más a la izquierda va a suponer que los estados converjan al valor de referencia antes y por tanto van a requerir de valores más grandes de u para poder realizarlo. Esto se ve claramente contrastando el par de figs. 35 y 36 entre sí, o el par de figs. 33 y 34.

Ahora es muy interesante estudiar el papel que juega la realimentación directa discretizada (*feedforward*), f sobre si el sistema llega a converger al valor de referencia. En el caso de las simulaciones realizadas, el valor de referencia siempre ha sido el mismo, para facilitar el posterior estudio de las mismas. Si comparamos la posición en la fig. 33 con la fig. 35, podemos apreciar que la primera sí llega a estabilizarse en $\theta = 500^\circ$, mientras que la segunda de estas no; converge a un valor por debajo del valor de referencia, en torno a $\theta = 460^\circ$. Esto se corrobora estudiando la señal z correspondiente en cada gráfica; en la fig. 33, en la que la posición sí llega a converger al valor de referencia, el valor de z converge a 0. Esto no ocurre para la fig. 35, en la que se ve que el valor de la señal z no converge en 0.

Si no se apreciasen bien las imágenes que se han comentado, se pueden consultar y descargar, puesto que están recogidas en el repositorio de github [3], en la carpeta 'P5_Estudio'.

4.3. Sistema de control completo, modelado y aplicado sobre el motor real

Al haber simulado satisfactoriamente el control discreto para un motor ideal, podremos controlar el motor real sustituyéndolo por el bloque del motor ideal. Sin embargo, cuando el motor alcanza su límite físico de voltaje (12V en nuestro caso) se acumula un error integral que provoca que el sistema tarde en frenar al llegar a la consigna establecida. Este efecto indeseado es el *wind-up*. Para reducir sus consecuencias, se añade al modelo un sistema *anti wind-up* que atenúa la señal integral cuando el potencial está saturado. En la siguiente figura se muestra el sistema de control y realimentación implementado para el motor real con el *anti wind-up* incorporado.

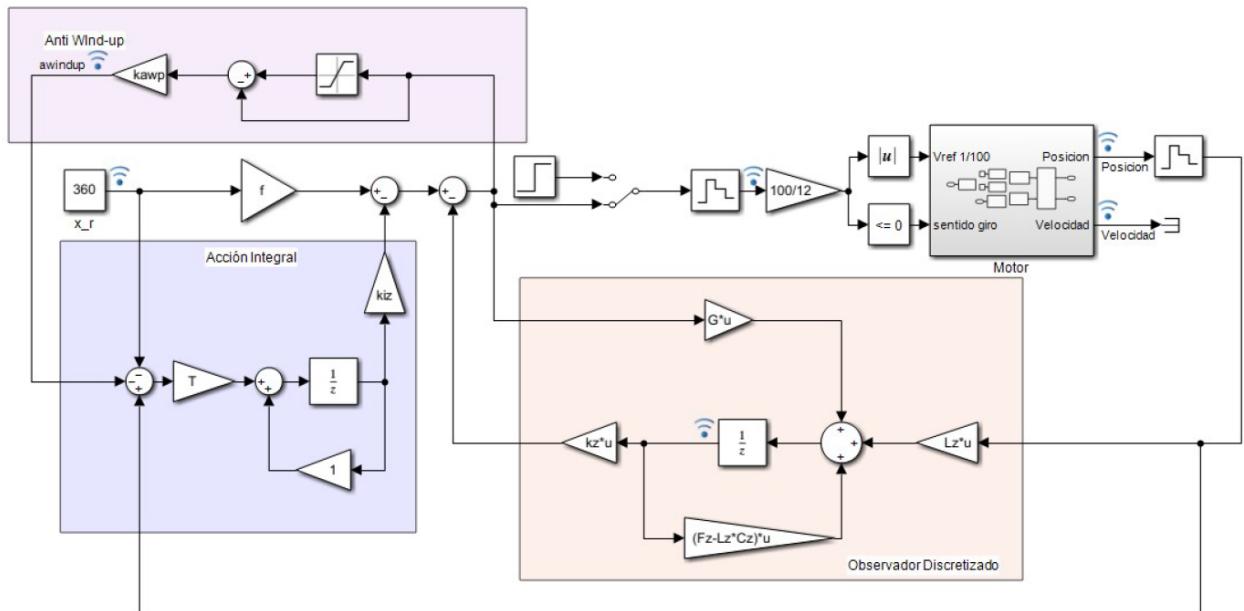
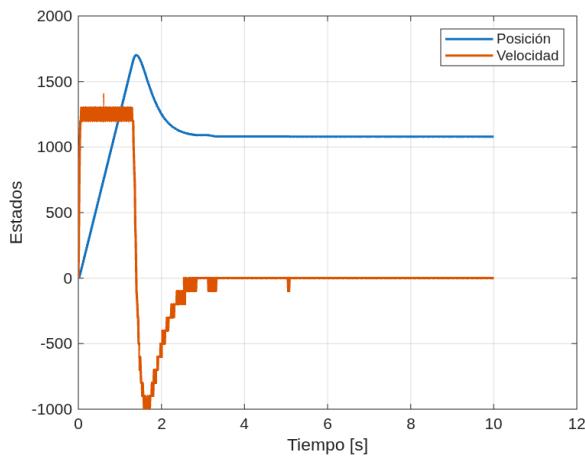
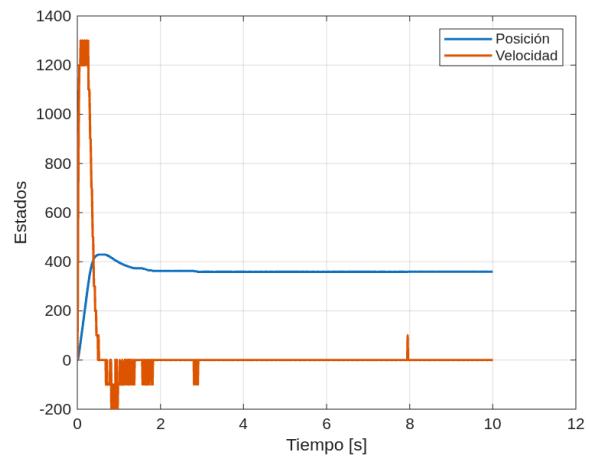


Figura 37: Modelo real del motor, con sistema de control por estados estimados, acción integral y acción directa ya incorporado

En las siguientes figuras se muestran los resultados obtenidos al hacer que el motor real alcance de forma estable una posición final de 1080° .

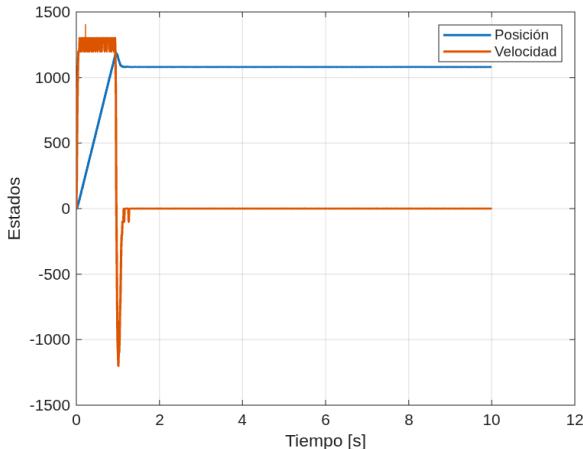
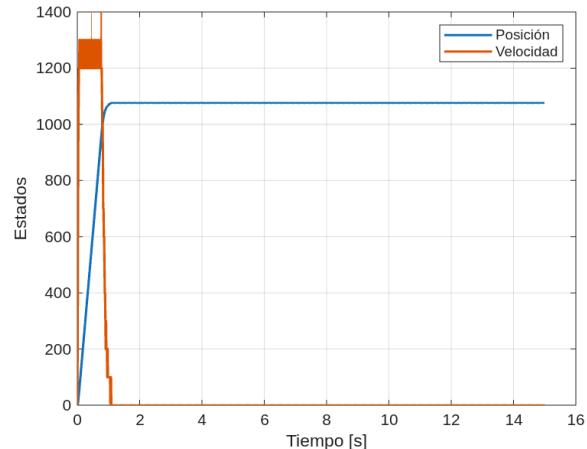
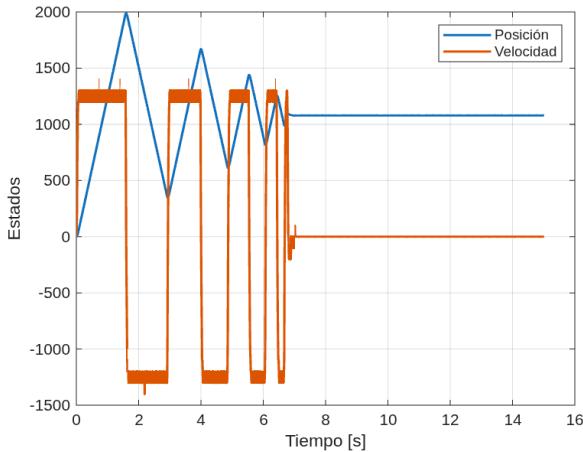
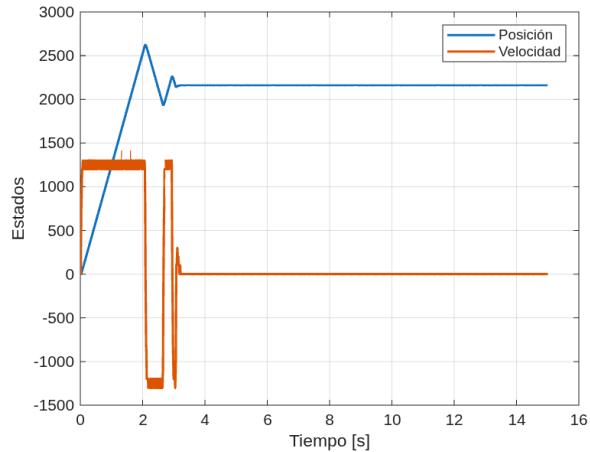


(a) $k_{amp} = 0,1$



(b) $k_{amp} = 2$

Figura 38: Datos de posición y velocidad registrados en el motor real. $p1 = 4,35$

(a) $k_{amp} = 1$ (b) $k_{amp} = 2$ Figura 39: Datos de posición y velocidad registrados en el motor real. $p1 = 86$ (a) $k_{amp} = 0,01$ (b) $k_{amp} = 0,1$ Figura 40: Datos de posición y velocidad registrados en el motor real. $p1 = 217$

En los datos registrados se aprecia que inicialmente la posición crece de forma constante incluso superando el valor de la consigna que se quiere alcanzar. Posteriormente se invierte el sentido de la velocidad y la posición decae pasando la consigna. Este ciclo del wind-up se va atenuando hasta que la consigna se queda estable.

Comparando diferentes mediciones se comprueba que un aumento de la constante del bloque anti wind-up (k_{amp}) hace que se tengan menos oscilaciones y con una amplitud menor.

También se puede observar el efecto de los polos de los estados afecta a la suavidad de la trayectoria: para polos más próximo a cero la trayectoria es mucho más suave porque no se satura tanto el voltaje, de forma una modificación de k_{amp} no afecta significativamente al tiempo necesario para alcanzar la posición final de forma estable. En cambio, como para valores más negativos de los polos el voltaje se encuentra más saturado, el aumento de k_{amp} sí que reduce notablemente la cantidad de oscilaciones y se alcanza antes la posición final.

5. Conclusiones

A lo largo de la práctica se ha logrado caracterizar y controlar con éxito la posición del motor de corriente continua mediante el uso de modelos en Simulink y la placa STM32. Se ha logrado determinar los parámetros k_e y p del motor, a partir de los cuales se pudo establecer un modelo matemático lineal que aproxima fielmente el comportamiento real del motor, salvo en rangos de voltaje muy bajos ($2V$ o menos) donde la fricción no es despreciable.

En cuanto al diseño del controlador, se ha logrado implementar empleando realimentación de estados estimados junto con acción integral. Asimismo, el estudio de la ubicación de los polos ha evidenciado que una respuesta temporal más rápida exige una mayor de la señal de control u , la cual está acotada físicamente por la saturación del actuador ($12 V$), haciendo necesario el uso de mecanismos *anti wind-up*.

Este logro de diseño e implementación se ha podido corroborar con las ejecuciones del modelo real con el sistema de control completo discretizado [37](#) sobre el motor real, conectándonos a la placa STM32. Dado que se ha aprendido y conseguido un sistema de control discretizado que permite llevar un motor a un ángulo de referencia prefijado θ_{ref} y parar el motor en dicha posición, podemos concluir que la práctica de la asignatura se ha completado con éxito.

6. Anexo

6.1. Código o funciones de MATLAB

```

1 function make_sims_realimdir(A, B, ke, p, theta0, w0, num_sims)
2 % Loadeamos el modelo que quiero ejecutar:
3 load_system('P4_Modelo_4_9_SistRealim')
4 %Exportar variables al workspace para que las encuentre el modelo
5 assignin('base','ke',ke);
6 assignin('base','theta0',theta0);
7 assignin('base','w0',w0);
8
9 for k = 1:num_sims
10 %Coloco polos:
11 exp = k-3;
12 mod_p = 1*10^exp;
13 polos = [-mod_p*0.5*p, -mod_p*0.5*p];
14 K = acker(A, B, polos);
15 assignin('base','K',K); % exporto K al workspace
16 % Configuramos simulacion:
17 tsim = 40/(k^3+0.5*(k-1)); %polos mas a la izq, menos tarda en estabilizarse
18 set_param('P4_Modelo_4_9_SistRealim', ...
19 'StopTime', num2str(tsim), 'MaxStep', '0.005', 'Solver', 'ode45');
20 %Ejecuto simulacion
21 simOut = sim('P4_Modelo_4_9_SistRealim');
22 %Me guardo los datos relevantes
23 t = simOut.tout;
24 theta = simOut.logsout.getElement('pos_(deg)').Values.Data;
25 w = simOut.logsout.getElement('vel_(deg)').Values.Data;
26 u = simOut.logsout.getElement('u').Values.Data;
27 resultados.t = t;
28 resultados.theta = theta;
29 resultados.w = w;
30 resultados.u = u;
31 save(sprintf('P4_RealimDir_%0.2f.mat', mod_p), 'resultados');
32 % Obtengo fig con estados
33 fig1 = figure;
34 plot(t, theta, 'LineWidth', 1.5)
35 hold on
36 plot(t, w, 'LineWidth', 1.5)
37 grid on
38 xlabel('Tiempo(s)')
39 ylabel('Estados')
40 legend('\theta(deg)', '\omega(deg/s)')
41 title('Evolucion de los estados')
42 saveas(fig1, sprintf('P4_Fig_RealimDir_Est_%0.2f.png', mod_p));
43 % Obtengo una fig con u
44 fig2 = figure;
45 plot(t, u, 'LineWidth', 1.5)
46 grid on
47 xlabel('Tiempo(s)')
48 ylabel('u(V)')
49 title('Senal de control')
50 saveas(fig2, sprintf('P4_Fig_RealimDir_SenalContr_%0.2f.png', mod_p));
51 %Cierro las dos figuras
52 close(fig1);
53 close(fig2);
54 end
55 end

```

Listing 7: Función para ejecutar simulaciones del modelo 21, guardar datos relevantes y gráficas de cada simulación

```

1 function make_sims_realimestim_int(A, B, C, ke, p, theta0, w0, num_sims)
2 % Loadeamos modelo que quiero ejecutar:
3 load_system('P4_Modelo_4_11_RealimEstim_ContrInt')
4 assignin('base','ke',ke); %exporto variable al workspace
5 assignin('base','theta0',theta0);
6 assignin('base','w0', w0);
7 for k = 1:num_sims
8     %Coloco polos:
9     exp = k-3;
10    mod_p = 1*10^exp
11    polos = [-mod_p*0.5*p, -mod_p*0.5*p];
12    polos_estim = [-mod_p*0.7*p, -mod_p*0.7*p];
13    % Configuramos simulacion:
14    tsim = 60/(k^3 + (k^2-1) ); % polos mas a la izq , menos tarda en estabilizarse
15    set_param('P4_Modelo_4_11_RealimEstim_ContrInt', ...
16              'StopTime', num2str(tsim), 'MaxStep', '0.0001', 'Solver', 'ode45');
17    % Obtengo ganancias:
18    L = (acker(A',C',polos_estim))';
19    assignin('base','L',L); % exporto L al workspace
20    Amp = [A zeros(2,1);C 0];
21    Bamp = [B;0];
22    Kamp = acker(Amp,Bamp,[polos -1.3*mod_p*p]); %polo integrador mas izq
23    Ki = Kamp(3);
24    K = Kamp(1,1:2);
25    assignin('base','Ki',Ki); % exporto Ki al workspace
26    assignin('base','K',K); % exporto K al workspace
27    Fc = inv(C*inv(B*K - A)*B);
28    assignin('base','Fc',Fc); % exporto Fc al workspace
29    %Ejecuto simulacion
30    simOut = sim('P4_Modelo_4_11_RealimEstim_ContrInt');
31    %Me guardo los datos relevantes
32    t = simOut.tout;
33    theta = simOut.logsout.getElement('pos_(deg)').Values.Data;
34    w = simOut.logsout.getElement('vel_(deg/s)').Values.Data;
35    u = simOut.logsout.getElement('u').Values.Data;
36    resultados.t = t;
37    resultados.theta = theta;
38    resultados.w = w;
39    resultados.u = u;
40    save(sprintf('P4_EstimInt_%2f.mat', mod_p), 'resultados');
41    % Obtengo una fig con pos y vel y otra con u:
42    fig1 = figure;
43    plot(t, theta, 'LineWidth', 1.5)
44    hold on
45    plot(t, w, 'LineWidth', 1.5)
46    grid on
47    xlabel('Tiempo(s)')
48    ylabel('Estados')
49    legend('\theta(deg)', '\omega(deg/s)')
50    title('Evolucion de los estados')
51    saveas(fig1, sprintf('P4_Fig_EstimInt_Est_%0.2f.png', mod_p));
52    fig2 = figure;
53    plot(t, u, 'LineWidth', 1.5)
54    grid on
55    xlabel('Tiempo(s)')
56    ylabel('u(V)')
57    title('Senal de control')
58    saveas(fig2, sprintf('P4_Fig_EstimInt_SenalContr_%0.2f.png', mod_p));
59    close(fig1); %cierro fig1
60    close(fig2); %cierro fig2
61 end
62 end

```

Listing 8: Función para ejecutar simulaciones del modelo 23, guardar datos relevantes y gráficas de cada simulación

Referencias

- [1] Jiménez, Juan. *guion.pdf — Práctica control de un motor CC*. Licencia Creative Commons Attribution–ShareAlike 4.0. 2025.
- [2] MathWorks. *Simulink® Coder™ Support Package for STMicroelectronics Nucleo Boards*. Ver. R2025b.
- [3] Tristan Ortiz, Javier Peña. *Repositorio de Github de la realización de la práctica*. Contiene los archivos empleados para la obtención de resultados.