Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA

POLITECNICA DE INGENIERIAS DE GIJON

DESARROLLO DE LABORATORIOS VIRTUALES EN ENTORNO MATLAB/SIMULINK PARA DOCENCIA EN CONTROL

D. PARDO FERNANDEZ, ALEJANDRO
TUTOR/ES:
D. DIAZ BLANCO, IGNACIO
D. PEREZ, DIEGO GARCIA

FECHA: 14 de junio de 2023

Índice de contenido

Índice de figuras

1. Introducción

1.1.- Motivación y justificación del trabajo

La Ingeniería de Control es un campo en constante desarrollo, habiéndose introducido recientemente nuevos métodos y técnicas, además de mejorar los enfoques ya existentes. Ejemplos de estas nuevas incorporaciones pueden ser los gemelos digitales o los laboratorios virtuales. En este contexto los laboratorios virtuales están cobrando importancia siendo estos definidos como entornos virtuales que pretenden emular laboratorios físicos o como se menciona en el artículo [1] un laboratorio virtual es un instrumento simulado contenido en uno o más ordenadores, conectados o no entre sí, con capacidades de gestión y/o aprendizaje de contenido, además, este artículo proporciona la diferencia entre los diferentes entornos virtuales, estableciendo las diferencias entre un instrumento virtual (IV), un instrumento remoto (IR), un laboratorio remoto (LR), un laboratorio virtual (LV) y un laboratorio virtual y remoto (LVR). A su vez, como está expuesto en el artículo [2], publicado por la revista Elsevier, los laboratorios virtuales surgen básicamente por la necesidad de crear sistemas de apoyo al estudiante para sus prácticas de laboratorio con el objetivo de optimizar el tiempo que este emplea en la realización de dicha práctica. Estos laboratorios virtuales tienen como principales ventajas:

- Permiten investigar o probar nuevos modelos sin tener que realizarlos físicamente y, además, pudiendo cambiar o modificar algunos elementos del prototipo sin necesidad de realizar una inversión económica, lo cual puede resultar costoso.
- Actualmente los laboratorios en la docencia de control están compuestos en su mayoría por equipos físicos limitados, como puede ser el control de la velocidad o de la posición de un motor, teniendo los alumnos acceso a pocos ejemplos de control a lo largo de su educación. No solo en el área de la docencia, sino que en el ámbito industrial cada vez más empresas están invirtiendo en esta técnica, aunque en el contexto industrial la idea de laboratorio virtual está directamente relacionada con el concepto de “gemelo digital”, ilustrado en la figura 1.1 en la que el gemelo digital obtiene los datos de los sensores en tiempo real y, con un modelado teórico correcto, un software de *machine learning* y datos anteriores podría comunicarse con el sistema real mediante los actuadores además de poder proporcionarnos los valores de variables internas difíciles de medir directamente en el sistema real. Si también se dispone de un modelo 3D se podría conectar al gemelo digital, teniendo aparte una visualización 3D del proceso. Por lo que se puede llegar a desarrollar, y debido a ello, se ofrecen cada vez modelos más precisos y fiables que se asemejan y adaptan mejor al medio físico

que están emulando.

- Asimismo, los laboratorios virtuales pueden permitir tener el control del sistema real a través de la visualización por el medio virtual, si está conectado mediante el medio físico adecuado al servidor. Por lo tanto, si se dispone de la conexión a servidor y se tiene el software móvil necesario, se podría llevar a cabo un seguimiento a tiempo real del proceso con un modelo 3D con el que podamos interactuar.

- Los alumnos podrán analizar el comportamiento del sistema dependiendo del método de control que elijan o del controlador, aportándoles una visión más completa que la que obtendrían utilizando exclusivamente el Matlab como herramienta de diseño, facilitándolo con la visualización 3D del modelo. En este proyecto se desarrollará algún ejemplo de laboratorio virtual implementando el modelo en Simulink, haciendo uso en algunas ocasiones del lenguaje de programación de Matlab y el desarrollo de la representación 3D en lenguaje de programación X3D.

- Igualmente, los alumnos tendrán más material sobre el que trabajar y practicar, ya que dispondrán de varios laboratorios virtuales con distintos modelos de distinto tipo, ya sea mecánico, eléctrico, etc, con lo que el alumno podrá practicar con los tipos de modelo que crea más conveniente.

Estas razones hacen que los laboratorios virtuales sean una opción con muchas ventajas para su implementación didáctica en control.

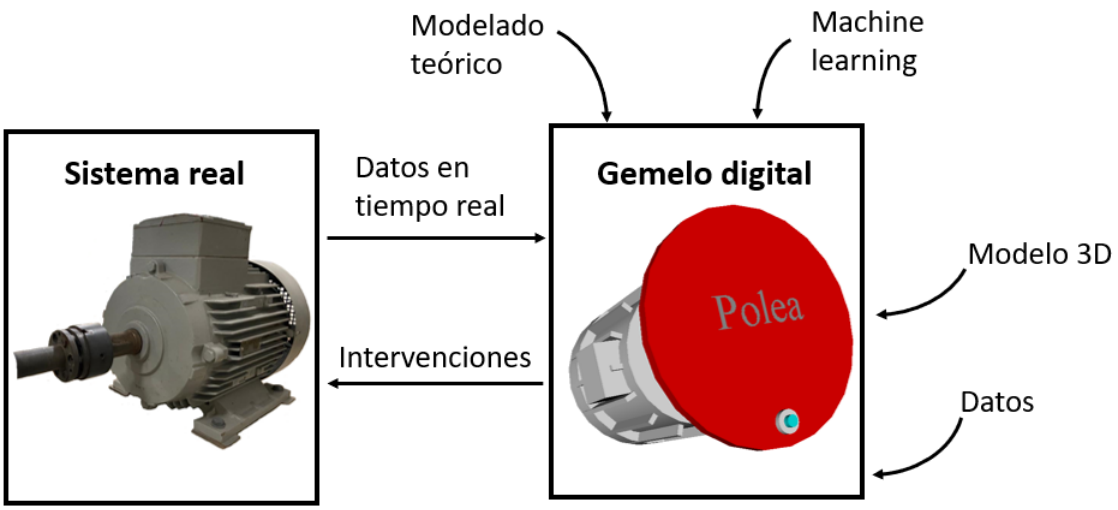


Figura 1.1: Gemelo digital

1.2.- Objetivos del trabajo

Este proyecto tiene como objetivo el diseño de laboratorios virtuales para su uso en el ámbito de la educación en Control, que emulen laboratorios físicos, para la prueba y diseño de sistemas de control. Los laboratorios virtuales consistirán en modelos visuales 3D alimentados por herramientas de simulación (Matlab/Simulink) que ejecutan modelos dinámicos (ecuaciones diferenciales) de los sistemas físicos a emular, permitiendo recrear su comportamiento. El propósito de los laboratorios virtuales es el de aunar la interpretabilidad y realimentación visual que ofrecen los laboratorios físicos, con la flexibilidad de las herramientas de simulación, suponiendo además un ahorro en costes y mantenimiento respecto a los laboratorios físicos, así como la ubicuidad, al permitir replicar el laboratorio en cualquier PC con el software de simulación. El proyecto implicará el desarrollo de dos laboratorios virtuales de sistemas de interés para la docencia en asignaturas de Control, capaces de utilizar un modelo oculto al usuario mediante s-functions codificadas con pcode para permitir su uso en trabajos o pruebas de evaluación. Como complemento a los laboratorios diseñados, se incluirá documentación útil para usos docentes y de investigación, con contenidos relativos al modelo matemático de los laboratorios implementados, posibilidades de configuración y ejemplos de sistemas de control. El proyecto tendrá también una componente de investigación en innovación docente. Se explorarán enfoques, ideas y tecnologías para mejorar las posibilidades docentes y enriquecer la experiencia del usuario (alumno, docente o investigador), tales como el uso de elementos de interacción con el laboratorio virtual (joysticks, acelerómetros, etc.), integración con otros sistemas (ej. comunicación con sistemas físicos para laboratorios remotos), técnicas de visualización 3D y valorar potencial de estos laboratorios en disciplinas relacionadas como la supervisión (ej. gemelo digital).

1.3.- Estructura del trabajo

El proyecto se estructura en tres bloques generales. El primero de ellos engloba los dos primeros capítulos (capítulos 1 y 2) en los cuales se introducen conceptos teóricos, como el concepto de laboratorios virtuales, su diferencia con los laboratorios tradicionales y por qué su implementación en la enseñanza de control puede redundar en una mejora educativa, además de mostrar otros conceptos teóricos necesarios como los softwares empleados.

El segundo bloque contiene los capítulos 3 y 4. En él se trata la implementación de los laboratorios virtuales, introduciéndolo con unas nociones básicas de control y seguido del modelado teórico y su implementación y diseño en el software escogido (tratando la parte en el software MATLAB/Simulink y también el diseño 3D).

Finalmente, en el tercer bloque (que contiene el capítulo 5) se debatirían los resultados obtenidos y posibles mejoras que se podrían implementar para trabajos o implementaciones posteriores.

2. Marco teórico y estado del arte en el desarrollo de laboratorios virtuales

2.1.- Laboratorios virtuales y su utilización en la docencia en control

Las diferencias principales entre los laboratorios virtuales y los laboratorios tradicionales, así como también se trata en el artículo de la Universidad Autónoma de Occidente Colombia [3] serían:

- La diferencia de coste entre un laboratorio físico y uno (o varios) virtuales ya que disponer de un laboratorio físico puede llegar a ser muy costoso y se tiene que llevar un mantenimiento de estos, pudiéndose estropear sensores, partes móviles, motores, etc. Además, el coste también impondría el número de equipos disponibles en las prácticas. Sin embargo, en el caso de los laboratorios virtuales, todos los alumnos con ordenador podrían llevarlo a cabo, ya que solo tendrían que descargar el programa y ejecutarlo con Matlab/Simulink.

- Otra diferencia importante es la variedad de ejemplos que se podrían enseñar con unos u otros, pudiéndose mostrar más ejemplos con la versión virtual. Además, los alumnos podrían practicar y ensayar con diferentes situaciones en casa, lo cual puede resultar bastante útil en la docencia de control, siendo esta una disciplina muy conceptual y con bastante grado de abstracción en algunos aspectos, en la que disponer de varios ejemplos y un mayor tiempo para implementarlo puede ayudar al alumno a entender y comprender más la asignatura.

- Los laboratorios virtuales tienen ventajas en cuanto a la evaluación del alumno, ya que gracias a la función p-code (mencionada en el apartado 5.3) se puede proteger el código de manera que el alumno no puede verlo, lo cual resulta muy útil ya que se puede proporcionar el modelo codificado de manera que el alumno no puede acceder a la programación del modelo, con lo que se puede aplicar en la realización de exámenes o trabajos teniendo los alumnos que llevar a cabo la tarea requerida. Se pueden proponer modelos variando las características de este con un generador de valores, por lo que cada alumno tendrá que realizar un diseño diferente.

- La principal ventaja de los laboratorios tradicionales es que el alumno lo puede tocar físicamente, viendo en un sistema real lo que pasa cuando cambia el controlador. Los efectos adversos que pueden experimentar el sistema al introducir distintas características de diseño pueden hacer que el sistema llegue a la saturación o a la inestabilidad. Aunque también se podrían introducir en el modelo virtual, esto significaría que el alumno tendría que fiarse del software implementado por el programador, de un mo-

do en el que esté bien programado, mientras que en el sistema real el alumno tiene confianza extrema.

Además, en dicho artículo de la Universidad Autónoma de Occidente Colombia [3] se explican las nociones básicas del concepto de simulación y su adhesión a la docencia y a la investigación y las circunstancias en las que sería conveniente hacer uso de simulaciones. En el se expone además la transferencia de conocimientos en dos grupos de estudiantes, uno utilizando laboratorios virtuales y otro usando los laboratorios tradicionales, en el que se ha visto por estadísticas que los alumnos adquieren con cierta mejoría los conocimientos en laboratorios experimentales, por lo que, aunque se disponga de laboratorios virtuales en la docencia, es también conveniente tener ejemplos de laboratorios tradicionales e incluso poder complementarlos, ya que el alumno puede tener más conocimiento del sistema controlado, en el que se debería poner la información recibida por el sensor analógico o digital (que a sí mismo se tendría que analizar, ya que la información leída es voltaje, pulsos de un encoder o alguna otra medida que posteriormente se tiene que tratar para leer la entrada correspondiente, ya sea velocidad, posición o incluso temperatura). Además, tendría que llevar la salida a un bloque físico (ya sea PWM, salida binaria...), proporcionándole un mayor conocimiento de la relación entre medio físico y medio virtual y cómo conectarlos. Esto también se muestra en el artículo [2], en el cuál se ha llevado a cabo un estudio entre los alumnos en un laboratorio de química que, aunque no es el mismo campo que el de este proyecto, puede dar una idea de lo expuesto anteriormente, generando la aplicación en la docencia de este laboratorio virtual un resultado positivo tanto en las aulas como en sus casas.

También hay que tener en cuenta, como se ha expuesto anteriormente, la gran expansión que se está llevando a cabo en la industria de el gemelo digital. En el artículo [4] publicado por la revista Elsevier se muestra un ejemplo de un laboratorio virtual de un ensayo de flexión haciendo énfasis en la necesidad de enseñar a los alumnos elementos relacionados con las nuevas necesidades de la industria, necesarias para las nuevas necesidades industriales. También se va a hacer uso del artículo publicado por el *Institute of Electrical and Electronics Engineers* [5] para explicar su importancia actual ya que, como está expuesto, es un concepto emergente en la industria e incluso académicamente, concepto con gran uso en la "industria 4.0" o cuarta revolución industrial sobre todo en la industria de la manufacturación. Por lo que el concepto de gemelo digital se puede definir generalmente como el esfuerzo de integración de datos entre una máquina física y otra virtual en ambas direcciones, lo cuál genera un entorno que nos permite un rápido análisis y toma de decisiones a tiempo real mediante un análisis preciso. Este concepto está relacionado con la evolución de las llamadas "ciudades inteligentes", en las que los servicios e infraestructuras están monitorizados con sensores haciendo uso de dispositivos del "Internet de las cosas", teniendo también beneficios en la

eficiencia energética y el ahorro de energía. También cobra importancia, como se ha expuesto antes, en la industria de fabricación, en las que se hace un seguimiento y monitorización de los productos, ya que el gemelo digital tiene el potencial de ofrecer una gran precisión mientras la máquina pueda soportar una gran cantidad de datos, necesario para el rendimiento y previsión del análisis, también introducido en la industria automovilística, con los coches inteligentes, usando estos modelos para multitud de aplicaciones. El último sector relevante sería el médico, teniendo como futura aplicación el gemelo digital emulando el sistema de una persona humana.

2.2.- Herramientas y tecnologías utilizadas en el desarrollo de laboratorios virtuales

El software MATLAB/Simulink, en el que se ha realizado este TFG, cuenta con una interfaz bastante intuitiva que consiste en unir bloques según lo que se necesite, incluyendo una opción de "add-ons" (complementos) en los que se pueden descargar otro tipo de bloques que no sean los simples requeridos, por ejemplo, si se quieren simular modelos térmicos o eléctricos. Además, en el caso de la representación 3D, permite importar modelos de varios tipos de lenguajes como VRML, X3D o de software como CAD (con AutoCad 3D o con SolidWorks).

2.2.1.- Matlab

A partir de la definición de Mathworks [6] MATLAB combina un entorno de escritorio perfeccionado para el análisis iterativo y los procesos de diseño con un lenguaje de programación que expresa las matemáticas de matrices y arrays directamente.

2.2.2.- Simulink

Tal como Mathworks[6] lo define, Simulink es un entorno de diagrama de bloques para simulación multidominio y diseño basado en modelos. Ofrece soporte para el diseño en el nivel de sistema, la simulación, la generación automática de código, y las pruebas y verificación continuas de sistemas embebidos. Simulink proporciona un editor gráfico, bibliotecas de bloques personalizables y solvers para modelar y simular sistemas dinámicos. Se integra con MATLAB, lo que permite incorporar algoritmos de MATLAB en modelos y exportar los resultados de simulación a MATLAB para seguir analizándolos.

2.2.3.- Modelado 3D con X3D

Para hacer el modelo 3D utilizaremos un archivo X3D, que es un formato de archivo de modelo 3D que utiliza una estructura de datos basada en XML (Lenguaje de Marcado Extensible). Es un estándar de la industria para la representación de gráficos 3D y es compatible con una amplia variedad de software de modelado y visualización. Por estos motivos resul-

ta idóneo para su uso en nuestro modelo de Simulink, ya que cumple todos los requisitos necesarios.

3. Fundamentos teóricos

3.1.- Conceptos básicos de control

Para llevar a cabo el desarrollo del proyecto, es necesario conocer algunos conceptos básicos de esta rama. Por lo que se va a llevar a cabo la explicación de los diferentes tipos de respuestas que podremos obtener de un sistema introduciendo en un instante temporal definido una referencia a escalón unitario, es decir, una referencia con valor unidad.

Teniendo esto en cuenta, una vez tenemos las ecuaciones físicas, pasaremos a realizar la función de transferencia del sistema que quedaría de la forma $G(s) = \frac{Y(s)}{X(s)}$, donde "Y" es la salida y "X" sería la entrada de nuestra función de transferencia. Para pasar de las ecuaciones físicas a una ecuación en "s", se realiza una transformación utilizando la transformada de Laplace. En esta transformación, se sustituyen las derivadas de la variable de interés por la variable compleja "s". Es importante tener en cuenta que el índice de derivación se convierte en el exponente correspondiente de "s" en la ecuación resultante. Esta función de tranferencia también se podría haber obtenido usando el método de modelado en espacio de estados y después haberse apoyado en el programa MATLAB con el código "$[num, den] = ss2tf(F,G,H,J)$", con el que, introduciendo los valores de las matrices resulta la función de transferencia del sistema, aunque el procedimiento para realizar este método esta explicado más adelante en este mismo apartado.

Una vez que se tiene la función de transferencia, ya se tendrá definido el sistema, por lo que se procede a diseñar el controlador. Para su diseño hace falta conocer las especificaciones que se requieren, ya sean especificaciones temporales, de frecuencia, de estabilidad o por limitaciones para no llegar a la saturación del sistema. Por ejemplo, en sistemas de segundo orden subamortiguados se tendrá que analizar la función de transferencia que se tiene con la de la forma:

$$G(s) = \frac{k \cdot {w_n}^2}{s^2 + 2 \cdot \xi \cdot s + {w_n}^2} \tag{3.1}$$

En la que:

- $k = \lim_{s \to 0} G(s)$
- $tiempo\ de\ pico = \frac{\pi}{w_d}$
- $tiempo\ de\ establecimiento\ (para\ un\ error\ en\ regimen\ permanente\ del\ 5\,\%) = \frac{3}{\sigma}$
- $sobreamortiguacion = e^{\frac{-\pi \cdot \sigma}{w_d}} = e^{-\pi \cdot \tan\theta}$

- $\xi = \sin\theta$
- $\sigma = \xi \cdot w_n$
- $y_\infty = k \cdot u_\infty$

Con esto se obtienen las ecuaciones necesarias para el diseño del sistema. En el caso de un sistema subamortiguado, la gráfica de la respuesta del sistema mostrará una amortiguación que depende del valor del coeficiente previamente calculado, así como de los demás parámetros. Esto resultará en una respuesta similar a la que se muestra en la figura 2, por ejemplo.

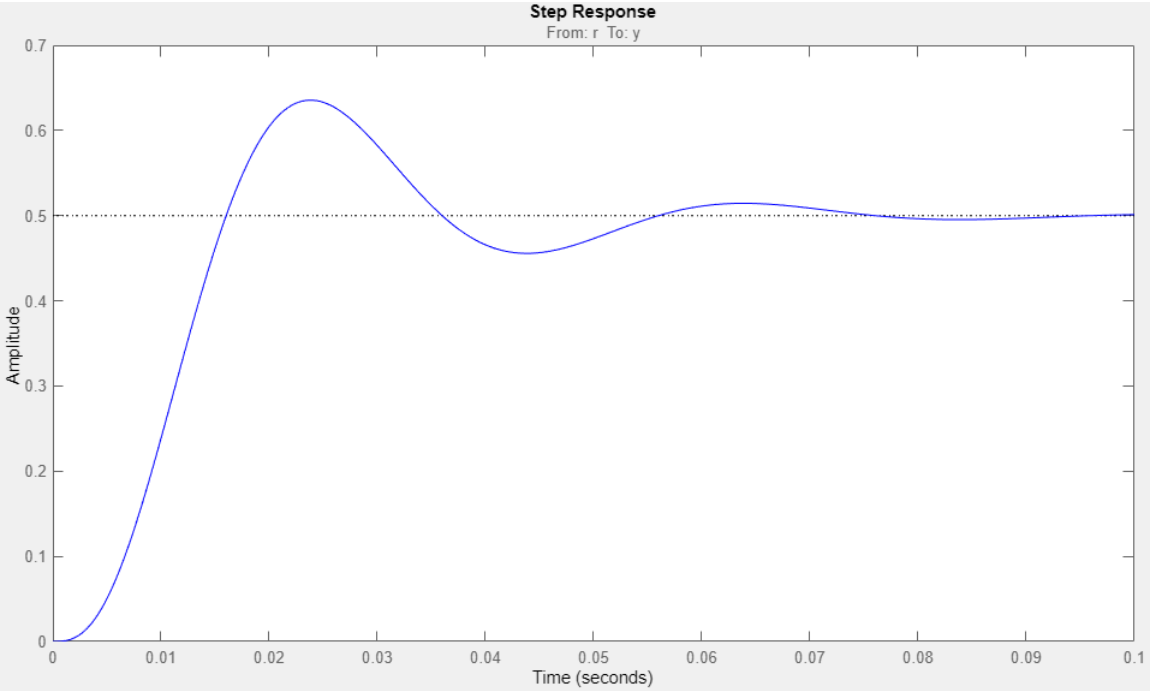


Figura 3.1: Ejemplo de respuesta de un sistema de segundo orden subamortiguado

Aunque los sistemas de segundo orden también podrían tener una respuesta sobreamortiguada, en la que no se tendría sobreoscilación, un ejemplo de este tipo de respuesta es el de la figura 3.2. Aunque para obtener las características fundamentales se usarían los polos dominantes, ya que estos son los que tienen una contribución más relevante en el comportamiento del sistema, sobre todo si los demás polos están alejados del eje.

También hay que tener en cuenta otros aspectos en el diseño, ya que hay ciertas situaciones en las que aunque se haya hecho un correcto diseño del controlador se puede llegar a la inestabilidad. Por ejemplo, si se tiene en cuenta el retardo que tendría un sensor real o si la acción de control es muy grande y el sistema llega a la saturación puede conllevar a que la acción de control tenga un límite, que influye en variaciones de la ganancia efectiva del

sistema (que tendría que ser igual a la ganancia teórica), efecto que cobra importancia si los polos están próximos al eje imaginario, ya que esta variación puede hacer que dichos polos pasen a la zona positiva del eje real volviendo al sistema inestable, situaciones que muchas veces no se contemplan en el diseño pero que después hace que el sistema no se comporte como debiera en la realidad.

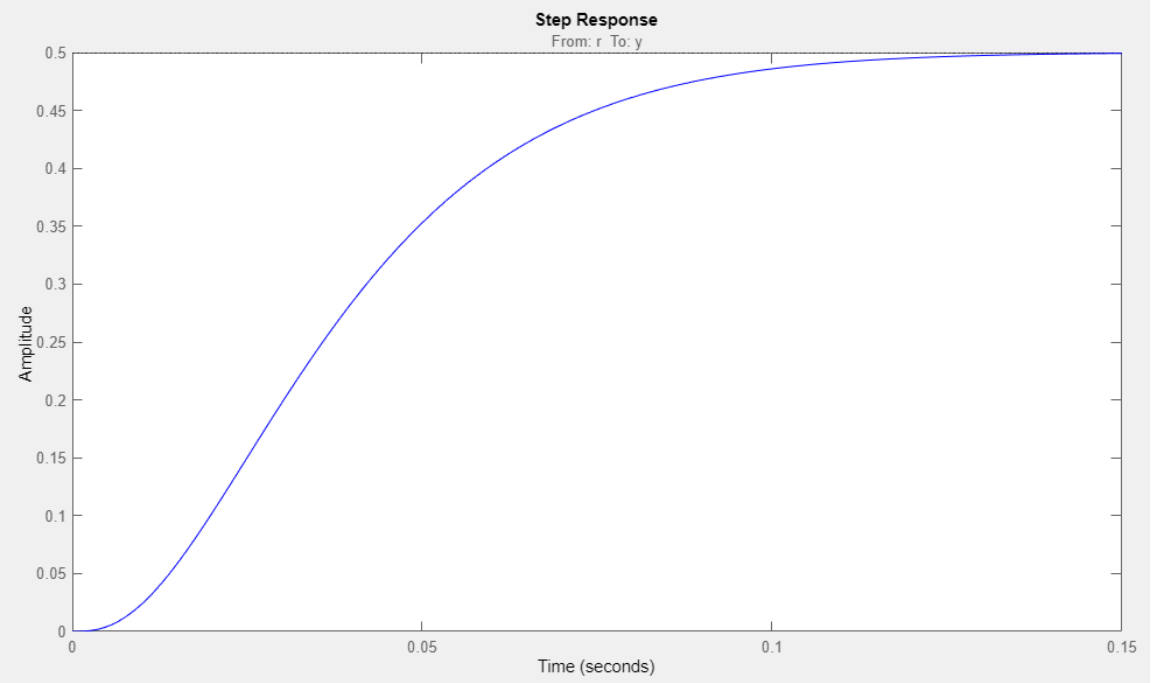


Figura 3.2: Ejemplo de respuesta de un sistema de segundo orden sobreamortiguado

En esta parte se introducirá el concepto de espacio de estados, utilizado en el diseño teórico de ambos modelos. Primero habrá que identificar las ecuaciones del sistema físico y también sus estados, determinando así su vector de estados correspondiente, después habrá que igualar las derivadas de cada estado para, finalmente, realizar la ecuación matricial siguiendo el siguiente esquema:

$$\dot{\mathbf{x}} = \mathbf{F} \cdot \mathbf{x} + \mathbf{G} \cdot u \tag{3.2}$$

$$y = \mathbf{H} \cdot \mathbf{x} + \mathbf{J} \cdot u \tag{3.3}$$

Donde $\dot{\mathbf{x}}$ es un vector columna que contiene el valor de las derivadas de los estados, $\mathbf{x}$ es un vector columna que contiene los valores de los estados y las matrices son resultado de interpretar los términos de las ecuaciones. Utilizando estas matrices, se llevará a cabo un análisis de la controlabilidad del sistema para lo que la matriz de controlabilidad desempeña un papel fundamental en este análisis, por lo que se procederá a su cálculo, que se obtiene a partir de las matrices de estado, $\mathbf{F}$ y entrada, $\mathbf{G}$, además de saber el número de estados n del sistema que coincide con el rango de la matriz $\mathbf{F}$, utilizándose la ecuación:

$$\mathbf{C} = [\mathbf{G};\ \mathbf{FG};\ \mathbf{F}^2\mathbf{G}\ ...\ \mathbf{F}^{n-1}\mathbf{G}] \tag{3.4}$$

Para que el sistema sea controlable, esta matriz ser de rango equivalente al del sistema y también revelará la capacidad del sistema para ser controlado y mediante su análisis se podrán tomar decisiones fundamentadas sobre la factibilidad y eficacia del diseño del controlador con el fin de lograr los objetivos de control establecidos.

3.2.- Ecuaciones en lazo cerrado

Cuando ya tengamos el sistema identificado, se procederá al diseño del controlador, para lo que, según las especificaciones requeridas (que puede ser diseño en el lugar de las raíces, por respuesta temporal, en frecuencia o bien teniendo en cuenta las propias limitaciones del sistema). En el control por espacio de estados (EESS) la ley de control (u) se diseña mediante la ecuación:

$$u = -\mathbf{K}\cdot\mathbf{x} + \bar{N}\cdot r \tag{3.5}$$

Donde **K** se usará para situar los polos en los lugares definidos, es decir, con su cálculo se establecerán las características técnicas del sistema, como la sobreoscilación, el tiempo de pico, el tiempo de establecimiento, etc, usando el comando de Matlab:

```
K=place(F,G,[polos_deseados]);
```

Se situarán los polos en el lugar resultante según los cálculos realizados de acuerdo con las especificaciones. Para conocer los valores propios del sistema se puede introducir el siguiente comando, el cuál devuelve un vector con tantos valores como variables de estado tenga el sistema.

```
eig(F)
```

$\bar{N}$ (lo denominaremos *Nbar* en el código de Matlab) nos permite obtener ganancia unitaria ante entrada escalón, necesaria para la ley de control. Teóricamente se realiza llevando a cabo el siguiente procedimiento.

$$\begin{bmatrix}\mathbf{N_x}\\ N_u\end{bmatrix} = \begin{bmatrix}F & G\\ H & J\end{bmatrix}^{-1}\cdot\begin{bmatrix}\mathbf{0}\\ 1\end{bmatrix} \tag{3.6}$$

$$\bar{N} = N_u + \mathbf{K}\cdot\mathbf{N_x} \tag{3.7}$$

Cabe destacar que $\mathbf{N_x}$ y el elemento **0** de dicha ecuación son vectores columna del mismo tamaño que el rango de la matriz **F**. Programando estas ecuaciones en Matlab, se tendrá que utilizar esta secuencia de comandos.

```
N_t=inv([F G;H J])*[zeros(size(G));1];
%Toma el ultimo valor del vector
N_u=N_t(end);
%Toma todos los valores del vector excepto el ultimo
N_x=N_t(1:end-1);
N_bar=N_u+K*N_x
```

3.3.- Modelado y simulación en Matlab/Simulink

La implementación del modelo en Simulink es una etapa crucial en el desarrollo de este trabajo de fin de grado. Mediante el entorno de Simulink, se puede diseñar y simular de forma visual el comportamiento del sistema de control propuesto.

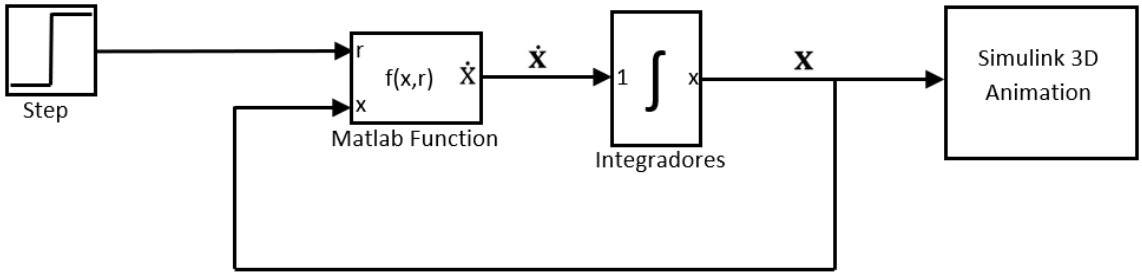


Figura 3.3: Diagrama de bloques genérico

El diagrama de bloques genérico de control consta de varios bloques funcionales que desempeñan diferentes roles en el sistema. Comenzando desde la izquierda, se encuentra un bloque de referencia, representado como un “step”, que indica el valor deseado para el sistema. En lugar de un bloque “step”, también se puede utilizar un bloque “Repeating Sequence Interpolated” para utilizar otras formas de referencia o incluso escalones temporales progresivos mediante el uso de varios bloques “step” y un bloque “sum”.

En lugar del bloque “step”, se puede integrar un joystick para permitir un control más interactivo y en tiempo real. Al mover el joystick en diferentes direcciones, se establece la referencia deseada para el sistema. La señal de salida del bloque de joystick varía entre “-1” y “1”. De esta manera, se escalan los valores dependiendo de los rangos positivos y negativos de la referencia, o si solo tiene rangos positivos, y el rango que pueden tener estos valores.

El siguiente bloque es un bloque de “Matlab Function” que recibe la referencia y el

valor de los estados, y devuelve las derivadas de los estados como salida. En este bloque, se implementaría el controlador desarrollado anteriormente, de la forma $u = -\mathbf{K} \cdot \mathbf{x} + \bar{N} \cdot r$, por lo que entonces las ecuaciones del sistema controlado quedarían de la forma:

$$\dot{\mathbf{x}} = (\mathbf{F} - \mathbf{G} \cdot \mathbf{k}) \cdot \mathbf{x} + \mathbf{G} \cdot \bar{N} \cdot r \tag{3.8}$$

$$y = (\mathbf{H} - \mathbf{J} \cdot \mathbf{k}) \cdot \mathbf{x} + \mathbf{J} \cdot \bar{N} \cdot r \tag{3.9}$$

En este caso, si en la función de MATLAB introducimos las ecuaciones de forma que las entradas al bloque sean la variable de entrada u y el valor actual de los estados $\mathbf{x}$ la estructura de las ecuaciones sería $\dot{\mathbf{x}} = f(\mathbf{x}, u$, la cual usamos en cadena abierta o cuando desacoplamos el controlador de la función de MATLAB.

También es posible utilizar el bloque “joystick input” (utilizado anteriormente para el joystick) para introducir los datos de los botones pulsados en el mando con lo que se puede modificar, por ejemplo, el controlador. La salida de los botones del joystick cambia a un estado lógico “1” cuando se pulsa y se mantiene en “0” cuando no se pulsa, lo que permite utilizar una instrucción “if-else” para seleccionar la acción deseada dependiendo del botón pulsado.

A continuación, se encuentra un bloque integrador que permite obtener el valor de los estados. Por último, mediante un “bus selector”, se seleccionan las variables que están introducidas en un bus de datos o se conectan directamente al modelo 3D de Simulink utilizando el bloque “VR Sink”, según los estados que sean relevantes para el sistema.

Estos bloques funcionales en el diagrama de bloques genérico de control se combinan de manera coherente para permitir la implementación y simulación de estrategias de control en un sistema específico. Al ajustar y personalizar los diferentes bloques, los ingenieros de control pueden analizar y optimizar el rendimiento del sistema bajo diferentes condiciones y requerimientos.

También se podría separar el controlador de la función de MATLAB, por lo que el resultado sería el de la figura 3.4, en el que se pueden observar la parte del controlador y la planta que, en este caso, contiene únicamente la ecuación de estado $\dot{\mathbf{x}} = \mathbf{F} \cdot \mathbf{x} + \mathbf{G} \cdot u$. Al separar el controlador de la función de MATLAB, se brinda flexibilidad en el diseño y la implementación del control, así como facilidad para el análisis y la corrección de fallos del sistema de control.

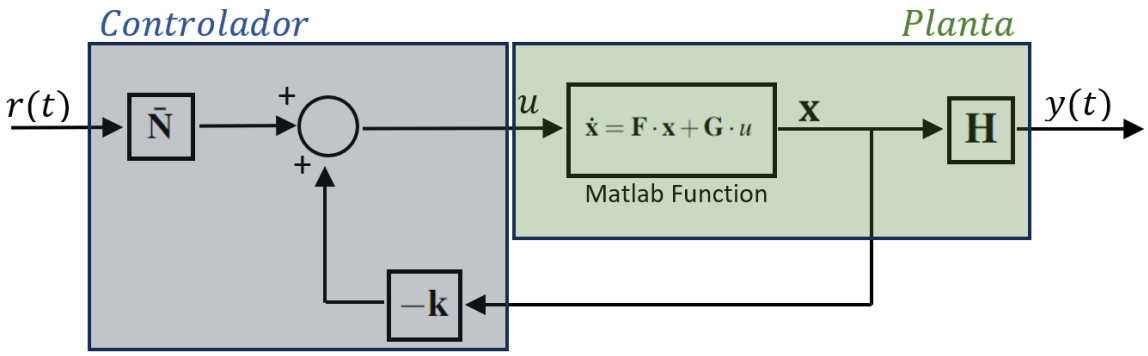
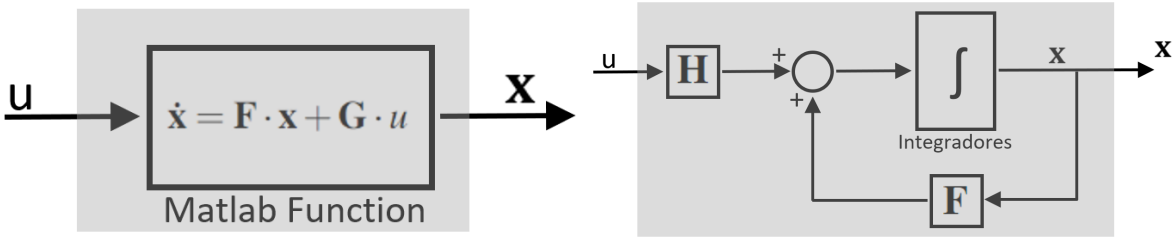


Figura 3.4: Diagrama de bloques genérico en EESS

También resultaría adecuado tener en cuenta que la función de MATLAB es una simplificación del diagrama de bloques mostrado en la figura 3.5, en la que la figura 3.5a de la función de MATLAB compacta da lugar a la figura 3.5b, lo cual proporciona una idea del diagrama de bloques de esta parte. Aunque esta implementación no resultaría útil ya que necesitamos una función con código del modelo para poder cifrarlo y que el alumno no pueda acceder a las características constructivas del modelo.



(a) Función de Matlab compacta (b) Función de Matlab ampliada

Figura 3.5: Diagramas de bloques de la función de MATLAB

La implementación del modelo en Simulink ha permitido visualizar de manera intuitiva el comportamiento del sistema de control propuesto, facilitando el análisis y la evaluación de su desempeño. Además, esta herramienta ha proporcionado un entorno flexible y accesible para realizar iteraciones y mejoras en el diseño del controlador, optimizando así el rendimiento del sistema. La capacidad de Simulink para integrar modelos y algoritmos de control de manera efectiva y visual ha sido fundamental para el correcto desarrollo de este trabajo de fin de grado, brindando una plataforma sólida para la investigación y el desarrollo de sistemas de control en tiempo real.

4. Diseño e implementación de los laboratorios virtuales

4.1.- Descripción de los laboratorios virtuales propuestos

4.1.1.- Ball and beam

El primer modelo propuesto es el del *ball and beam* (sistema barra-bola), partiendo del modelo propuesto por las universidades de Michigan, Carnegie Mellon y Detroit Mercy [7], como el primer paso en el estudio y análisis de conceptos básicos de control. Este modelo se centra en el control de la posición de una bola sobre una barra, mediante el control del ángulo de inclinación de la misma. El objetivo principal es lograr un control preciso de la posición de la bola. Este modelo es un ejemplo clásico en el campo de la ingeniería de control y proporciona una base sólida para el entendimiento de los fundamentos del control.

Figura 4.1: Diagrama del sistema de ball and beam

Dicho sistema es un buen ejemplo para mostrar a los alumnos como se controlaría un sistema físicamente inestable, teniendo que obtener las ecuaciones matemáticas que describen el modelo físico. Además, los alumnos pueden observar la diferencia entre controladores y los efectos que tiene sobre el sistema introducir un controlador más rápido o con referencias que implican una acción de control brusca que, en este caso, podría llevar a la inestabilidad del sistema.

4.1.2.- Modelado teórico del ball and beam

El sistema del *ball and beam* consta de dos grados de libertad: la inclinación de la barra α y la posición de la esfera r. Siendo este sistema inestable, lo que requiere la implementación de un lazo de control para asegurar su estabilidad.

Una de las partes físicas más importantes en este sistema es la medida de la posición de la bola con respecto a la barra. Uno de los sensores más fácilmente implementable sería como el de un potenciómetro, basado en un hilo resistivo que actuará como resistencia variable dependiendo de la posición de la bola (conductora eléctrica).

Para el desarrollo del primer modelo del *ball and beam* se realizará el sistema en cadena

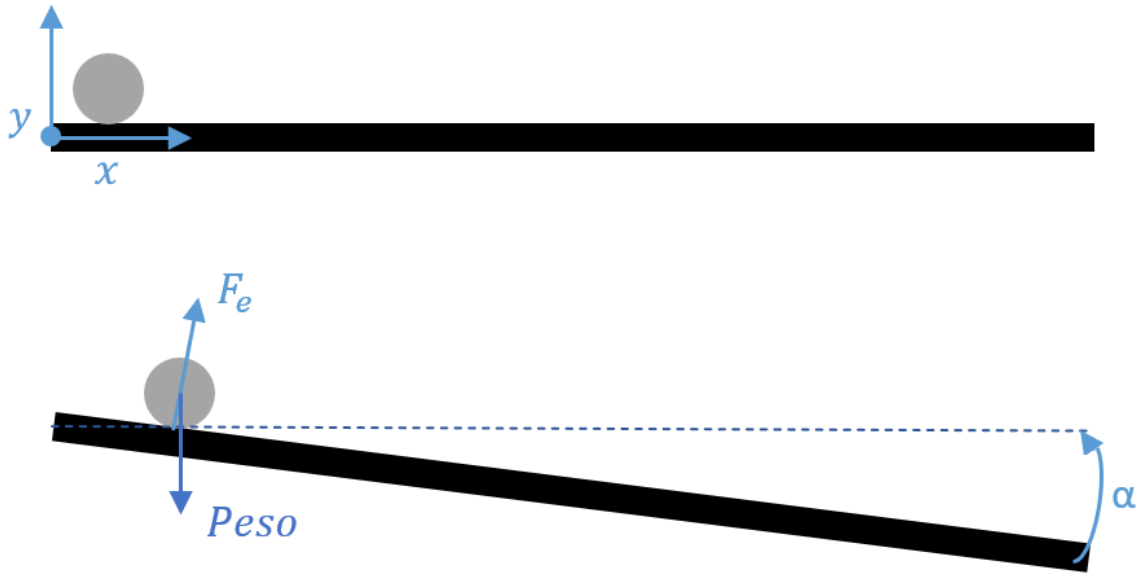


Figura 4.2: Diagrama de cuerpo libre del ball and beam

abierta sin control (para una primera prueba más simple), tomamndo como punto de partida las ecuaciones del modelo propuesto por las universidades de Michigan, Carnegie Mellon y Detroit Mercy [7], además de hacer uso de las ecuaciones del artículo [8]:

$$0 = \left(\frac{J}{R^2} + m\right)\ddot{r} - mg\sin\alpha - mr\dot{\alpha}^2 \tag{4.1}$$

Siendo R el radio de la bola, J su momento de inercia y m su masa. Además el último término de la ecuación hace referencia a la fuerza centrífuga, apreciable para movimientos bruscos (con una velocidad angular alta). Aparte de esto cabe destacar que en esta ecuación se ha supuesto que la masa de la bola es suficientemente pequeña para no provocar reacciones inerciales que afecten al movimiento de la barra. Además hay que tener en cuenta el subsistema de la barra, sacando sus ecuaciones aplicando la segunda ley de Newton, también explicadas en el artículo de la universidad de Malaysia [8], haciendo equilibrio de momentos:

$$J \cdot \ddot{\alpha} = \sum_{i=1}^{n} T_i \tag{4.2}$$

Para la que J es el momento de inercia, $\ddot{\alpha}$ es la aceleración angular y el término derecho de la ecuación 4.2 es el sumatorio de todos los pares que actúan sobre el pivote. Para lo que, la fórmula general de este subsistema sería $J \cdot \ddot{\alpha} = T$, siendo T el par aplicado. Pero teniendo en cuenta el rozamiento, para el que la ecuación del equilibrio de momentos sería $J \cdot \ddot{\alpha} = -b \cdot \dot{\alpha} + T$, por lo que la ecuación resultante teniendo en cuenta el rozamiento sería:

$$\ddot{\alpha} = -\frac{b}{J} \cdot \dot{\alpha} + \frac{1}{J} \cdot T \tag{4.3}$$

En este ejemplo se van a tomar, para las variables, los valores $m = 0.111,\ g = -9.8,\ L = 1.0,\ J = 9.99 \cdot 10^{-6}$ *y* $R = 0.015$. Se continuará con la linealización en torno al punto $\alpha = 0$, es decir, para pequeñas variaciones del ángulo el seno de este será aproximadamente el ángulo, $\sin\alpha \approx \alpha$, despreciándose también el último término de la ecuación 4.1 ($mr\dot{\alpha}^2$) resultando la ecuación:

$$\left(\frac{J}{R^2} + m\right)\ddot{r} = mg\alpha \tag{4.4}$$

Siguiendo la ecuación 4.1 se analizará cómo sería su implementación con Matlab en espacio de estados, cuyas entradas son la acción de control y el valor de los estados en cada instante y cuyas salidas son las derivadas de los estados, según lo establecido en el diagrama de bloques de la figura 3.3. Ahora se va a cambiar el nombre de las variables de estado para su próxima identificación en el código de MATLAB, por lo que la posición de la bola sería la primera variable x_1 y la velocidad de la bola, o derivada de la posición, la segunda variable x_2, seguido de la parte de la barra en la que el ángulo sería la variable x_3 y su derivada (velocidad angular) la variable x_4, quedando:

$$\begin{aligned} x_1 &= r \\ x_2 &= \dot{r} \\ x_3 &= \alpha \\ x_4 &= \dot{\alpha} \end{aligned}$$

El siguiente paso será identificar las ecuaciones que igualan las derivadas de los estados con éstos, para el cual hay que hacer uso de la ecuación 4.4 para la identificación de las variables x_1 y x_2 y la ecuación 4.3 para las variables x_3 y x_4, quedando las ecuaciones:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{mgx_3}{\frac{J}{R^2} + m} \\ \dot{x}_3 &= x_4 \\ \dot{x}_4 &= -\frac{b}{J} \cdot x_4 + \frac{1}{J} \cdot u \end{aligned}$$

Hay que tener en cuenta que las ecuaciones anteriores son para un modelo linealizado, haciendo uso de la ecuación 4.1 para obtener la ecuación no lineal, para el que variaría la ecuación de la variable $\dot{x}_2$, único término con ecuación no lineal, por lo que con los cambios

introducidos, el modelo no lineal resultaría de la siguiente forma:

$$\begin{aligned} \dot{x}_1 &= \dot{r} \\ \dot{x}_2 &= \frac{m \cdot g \cdot sinx_3 - m \cdot x1 \cdot x_4^2}{\frac{J}{R^2} + m} \\ \dot{x}_3 &= \alpha \\ \dot{x}_4 &= -\frac{b}{J} \cdot x_4 + \frac{1}{J} \cdot u \end{aligned}$$

Cabe destacar que si consideramos rozamiento nulo y que la bola posee un precio despreciable respecto al conjunto se puede asumir que el sistema tiene $b = 0$ y $J = 1$, con lo que la entrada al sistema u seria equivalente al par aplicado, caso específico considerado en este ejemplo.

4.1.3.- Desarrollo del modelo en Simulink del ball and beam

En esta sección van a estudiarse el desarrollo del modelo haciendo uso del programa "Matlab-Simulink", primero con los bloques del Simulink y después con el desarrollo del diseño en espacio de estados en la función de Matlab, para el diseño del controlador se empleará el software Matlab, analizando la ecuación resultante de linealizar en torno al punto de equilibrio $\alpha = 0$ (ecuación 4.4).

Para hacer el control en EESS se tiene que llegar a una ecuación genérica que relacione las derivadas de los estados con sus derivadas de la forma $\dot{\mathbf{x}} = \mathbf{F} \cdot \mathbf{x} + \mathbf{G} \cdot u$, donde $\dot{\mathbf{x}}$ es un vector columna 4×1 que contiene las derivadas de los estados, $\mathbf{F}$ es una matriz 4×4, $\mathbf{x}$ es un vector columna 4×1 que contiene los valores de los estados, $\mathbf{G}$ es una matriz 4×1 y u es la acción de control que será el momento que ejercemos a la tabla. Entonces las ecuaciones quedarán de la forma:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{mg}{\left(\frac{J}{R^2}+m\right)} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{b}{J} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{J} \end{bmatrix} \cdot u \tag{4.5}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + [0] \cdot u \tag{4.6}$$

Una vez obtenidas las matrices de estado se procederá al análisis de la controlabilidad, expuesto en el apartado 3.1, por lo que haciendo uso de la ecuación 3.4 sabiendo que el sistema es de rango 4, tendremos que adaptar dicha ecuación, por lo que resultaría $\mathbf{C} = [\mathbf{G};\ \mathbf{FG};\ \mathbf{F}^2\mathbf{G};\ \mathbf{F}^3\mathbf{G}]$, cálculo que siendo realizado en MATLAB resultaría muy sencillo, dando como resultado:

$$\begin{bmatrix} 0 & 0 & 0 & 7 \\ 0 & 0 & 7 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Esta matriz resultante es de rango 4, por lo que podemos concluir en que el sistema es controlable, condición necesaria para llevarlo a cabo. Como una primera prueba más sencilla y para comprobar que el sistema funciona correctamente se procederá al diseño del sistema en bucle abierto, es decir, proporcionando un ángulo constante, pero con el sistema no lineal, haciendo uso de la ecuación 4.1, con lo que cambiaría el término 2x3 de la matriz **F** (ecuación 4.5).

4.1.4.- Control del ángulo α

En la figura 4.3 se muestra el diagrama de bloques requerido para el diseño de nuestro modelo para que se pueda controlar el ángulo de la tabla como referencia del sistema, también ofreciendo ejemplo de un sistema controlado por espacio de estados en el que tenemos el controlador desacoplado en vez de integrar el controlador en la función de MATLAB, siendo el bloque de color azul el controlador, que englobaría la $\bar{N}$ y la **K** y el bloque de la planta con el fondo gris, englobando la función de MATLAB, los integradores y la matriz **H**. En la figura 4.3 se muestra la estructura del diagrama de bloques para hacer que el ángulo que queremos que se mantenga constante, ya que la acción de control del sistema diseñado sería la aceleración angular de la tabla como tenemos concretado en la ecuación 4.5. Esta implementación sería necesaria debido a que controlar directamente el ángulo de inclinación de la tabla sería un sistema no causal en el que no se puede controlar directamente el ángulo sino que se tiene que controlar modificando el par ejercido, esto es debido a que la posición de la bola no solo depende del ángulo de inclinación, sino que depende de otras variables, como la velocidad y la aceleración de la bola.

Trasladando este diagrama de bloques a Simulink quedaría tal como el mostrado en la figura 4.4 en la que se pueden observar las dos partes del sistema, controlador y planta, en el Simulink se tiene que realimentar la “s-function” con el valor actual de los estados, ya que la función no tiene variables que memoricen el valor anterior de dichas variables. Además,

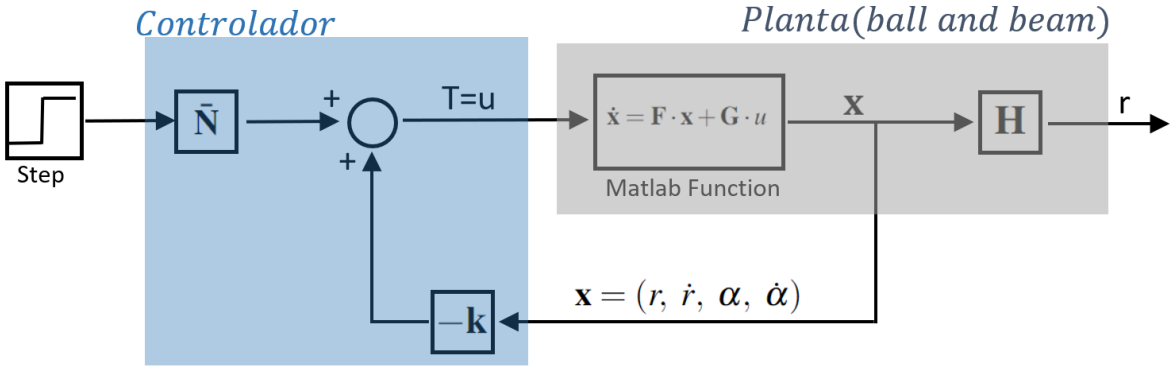


Figura 4.3: Diagrama de bloques del sistema de *ball and beam* en cadena abierta en espacio de estados

los integradores se han agrupado en un bloque al que entra un bus de datos con los valores de las derivadas de los estados para simplificar el diagrama de Simulink. Así como para las entradas al modelo 3D, ya que es una agrupación de varias variables tratadas en el subapartado 4.2.2.

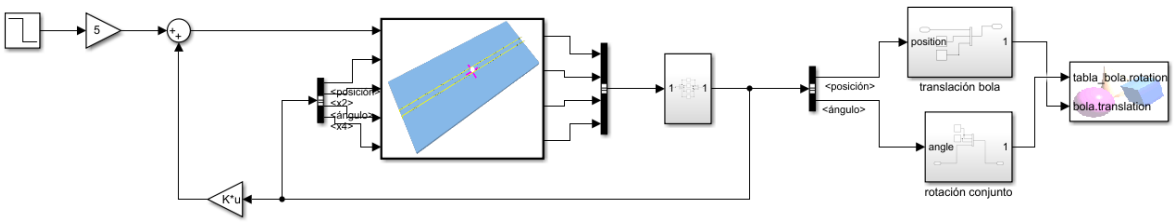


Figura 4.4: Sistema en cadena abierta del *ball and beam* en EESS en Simulink

Para esta implementación del controlador del ángulo en espacio de estados necesitaremos hallar el valor del vector **K** la que se diseñará teniendo en cuenta que se tiene que dividir el sistema de *ball and beam* en dos, para así controlar el subsistema de la barra, con el que podemos llevar a cabo el control del ángulo. Por lo tanto, nuestro subsistema estará formado por las variables $\alpha,\ \dot{\alpha}$, para el que ahora las ecuaciones serán:

$$\begin{bmatrix} \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{b}{J} \end{bmatrix} \cdot \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{J} \end{bmatrix} \cdot u \tag{4.7}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} + [0] \cdot u \tag{4.8}$$

Entonces en este ejemplo se van a cambiar de nombre a las matrices del sistema, pasando a denominarse F', G' y H' y también variará la estructura de la **K** que pasaría a ser $\mathbf{K} = [0,\ 0,\ k_3,\ k_4]$, con la que ya se tendría solo en cuenta el control de la posición de la bola. En este caso no se necesitarán unas especificaciones muy estrictas por lo que

se establecerá un tiempo de pico, t_p, de 1.57 segundos, un tiempo de establecimiento, t_s, (para un rango de error del 5 %) de 3 segundos y una sobreoscilación de $M_p = 21\,\%$. Para cumplir con estas especificaciones se pondrán los polos en las posiciones $-1 \pm 2 \cdot j$ para el subsistema de la tabla $\mathbf{K'} = [k'_3,\ k'_4]$, calculada mediante el comando de MATLAB "$K = place(F', G', polos\_deseados)$" y sacando $\bar{N}_r$ de la forma descrita en el apartado 3.2 resultando esta $K' = [5,\ 2]$ por lo que quedaría $\mathbf{K} = [0,\ 0,\ 5,\ 2]$ y con $\bar{N} = 5$, siguiendo las instrucciones de MATLAB explicadas en el apartado 3.1. Cabe destacar que, según la estructura del diagrama de bloques de la figura 4.3, podríamos cambiar del control de la posición y del control del ángulo dependiendo de los valores de $\bar{N}$ y de $\mathbf{K}$, ya que para controlar el ángulo introduciríamos los valores tal que $K = [0,\ 0,\ k'_3,\ k'_4]$ además de la $\bar{N}_r$ o, para controlar la posición de la bola, con $\mathbf{K} = [k_1,\ k_2,\ k_3,\ k_4]$ y $\bar{N}_x$.

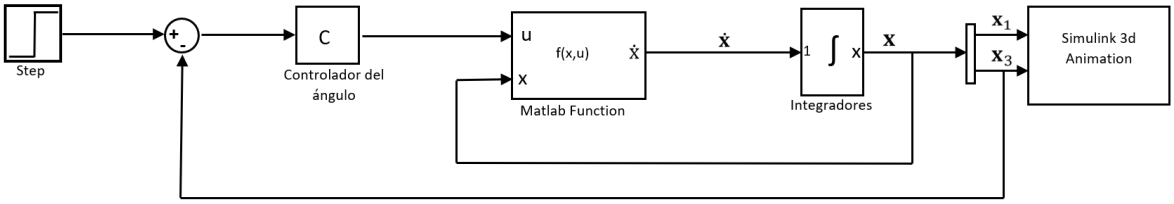


Figura 4.5: Diagrama de bloques del sistema en cadena abierta del *ball and beam*

Para este diseño 4.5 si se programa en Simulink, quedaría de la forma mostrada en la figura 4.6:

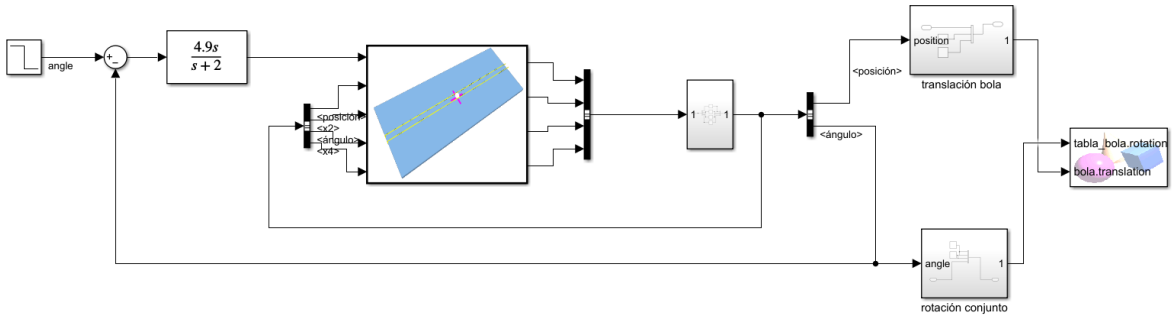


Figura 4.6: Sistema en cadena abierta del *ball and beam* en Simulink

El diagrama de la figura 4.5 proporciona una alternativa al diseño en espacio de estados, siguiendo un diseño por función de transferencia. Para su desarrollo obtendremos el ángulo actual de la variable x_3, se situarán los polos en el mismo lugar del diseño en EESS para obtener un control equivalente, ya que los resultados y gráficas resultantes son iguales. Para situar los polos en las situaciones requeridas se hará uso de la herramienta "sisotool" de Matlab, con el que quedará un controlador de la forma $4.9 \cdot \frac{s}{s+2}$. Dicho control se hará mediante una red de adelanto, teniendo la planta un doble integrador, con lo que el controlador

aportará un cero en el origen y moverá el otro polo hacia el lugar más conveniente dependiendo de las especificaciones requeridas. El código de Matlab empleado para la obtención de este controlador sería:

```
s=tf('s');
G=1/s^2;
sisotool(G);
```

En este caso, el subsistema que deseamos controlar se modela como un doble integrador, ya que la referencia de ángulo se traduce en el control de la aceleración angular. Utilizando la función "sisotool", se abre una ventana que permite establecer una red de adelanto. Esta red de adelanto se representa mediante la función de transferencia $C = k \cdot \frac{s}{s+p}$, donde "k" y "p" son los parámetros ajustables, mediante el uso de esta herramienta se proporciona la flexibilidad necesaria para diseñar y ajustar el controlador de manera interactiva.

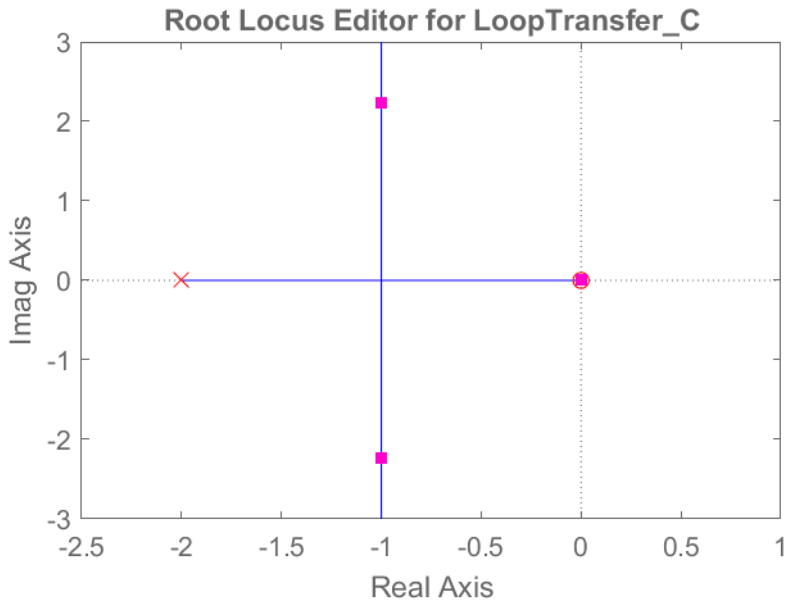


Figura 4.7: Polos del sistema controlado mediante "sisotool"

El resultado de introducir el controlador `C` deseado se puede ver en la figura 4.7, ya introducidos los valores que se habían calculado en el apartado anterior, teniendo ahora las raíces en las posiciones $-1 \pm 2 \cdot j$.

En cuanto a la figura 4.8, como habíamos dicho anteriormente, el comportamiento del sistema sigue las especificaciones que se establecieron anteriormente, siendo el instante inicial de la simulación 1 segundo (el bloque "step" fijará una salida de 0.031 radianes, es decir, 1.8 grados a partir del instante $t = 1$).

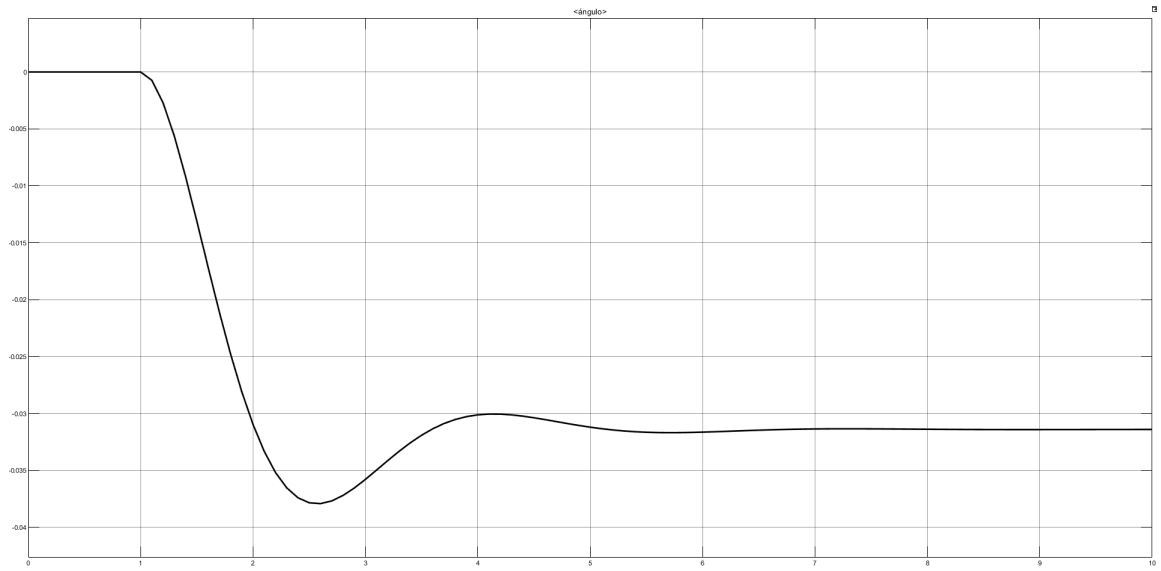


Figura 4.8: Simulación del ángulo controlado en bucle abierto

4.1.5.- Diseño del sistema controlado del ball and beam

En la siguiente figura 4.9 se mostrará el diagrama de bloques empleado para la realización del modelo básico del Simulink.

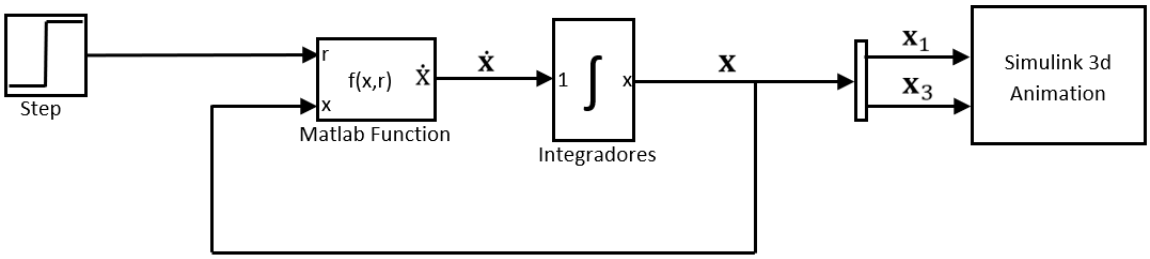


Figura 4.9: Diagrama de bloques simplificado del modelo controlado del *ball and beam*

Comentando sus elementos, el bloque más a la izquierda sería el "step", cuya función es la de establecer una entrada concreta en forma de escalón a partir de un momento específico de la simulación, estableciendo un ángulo fijo α concreto. Aunque para la implementación de este modelo se ha tenido que diseñar un controlador concreto para establecer el ángulo, ya que la variable de control del sistema es el par aplicado.

El siguiente bloque sería la función de Matlab, en la que serán introducidas las características del sistema y las ecuaciones correspondientes a las derivadas de cada estado, así como la ley de control, la cual tiene como entradas el valor de los estados y la referencia y como salida las derivadas de cada estado, por lo que el siguiente bloque sería un bloque integrador para así obtener los estados. Por último, como las líneas de datos están constituidas

como buses (para simplificar el diagrama), sacaremos las señales que sean necesarias para la simulación 3D (la posición de la bola y el ángulo del conjunto barra-bola), estando el diseño 3D más especificado en 2.2.3.

Para el diagrama de bloques del sistema controlado del *ball and beam* con el controlador introducido en la "s-function" correspondería un diseño en Simulink como el mostrado en la figura 4.10:

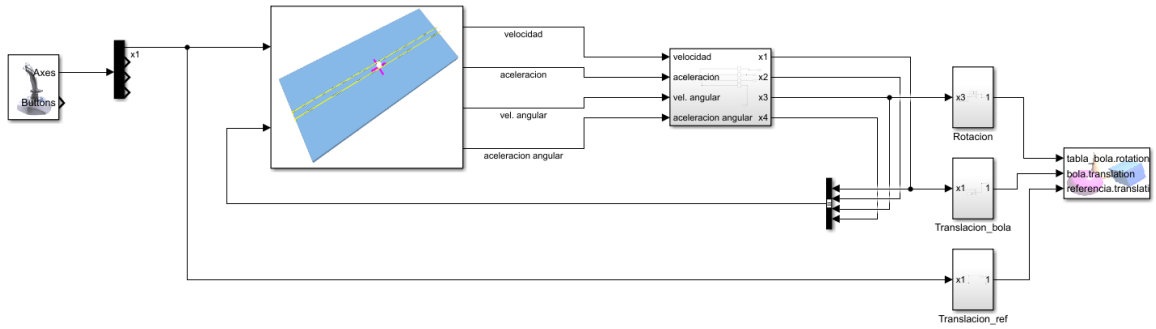


Figura 4.10: Sistema controlado del *ball and beam* en Simulink

También se podría realizar el diagrama de bloques separando el controlador de la función de MATLAB, para lo que se tendría que llevar a cabo siguiendo la estructura del diagrama de la figura 4.3, ya que como se había expuesto anteriormente, si se cambia la $\mathbf{K}$ y se introduce el valor de la $\bar{N}$ para el sistema controlado se obtendría un sistema equivalente a introducir el control dentro de la función de MATLAB.

4.1.6.- Diseño del controlador en Espacio de Estados para el ball and beam

En esta subsección vamos a diseñar el controlador y su implementación, así como el diseño según lo explicado en los apartados anteriores, ya se tendrán identificadas las matrices $\mathbf{F}$, $\mathbf{G}$, $\mathbf{H}$ y $\mathbf{J}$. El siguiente paso será identificar el lugar de las raíces del sistema, obtenidas con el siguiente código de Matlab (habiendo definido anteriormente las constantes del sistema y las matrices):

```
ball_ss=ss(F,G,H,J);
pzmap(ball_ss)
polos=pole(ball_ss)
```

Mediante este procedimiento se creará el modelo en espacio de estados a partir de sus matrices de estado, de entrada y de salida, al que llamaremos `ball_ss` (objeto de tipo modelo dinámico devuelto por la función `ss()`), en la que se obtendrá una representación gráfica del lugar de las raíces (instrucción `pzmap`) y además el lugar exacto de los polos (instrucción `pole`) que dará como resultado cuatro polos en el origen (sistema inestable). La representa-

ción gráfica de los polos en Matlab se puede ver en la figura 4.11.

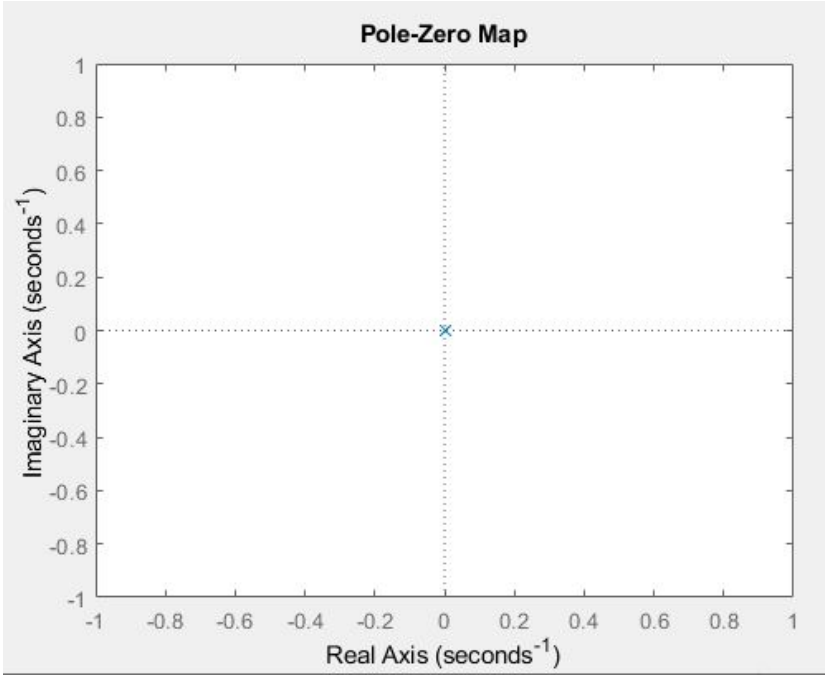


Figura 4.11: Mapa de polos y ceros del sistema

Con la información del lugar de los ceros y polos del sistema, en este caso en la Figura 4.11 se procederá al diseño del controlador con el siguiente código de Matlab:

```
K=place(F,G,[p1 p2 p3 p4])
Nt=inv([F G;H J]*[zeros(size(G));1]
Nu=Nt(end)
Nx=Nt(1:end-1)
Nbar=Nu+K*Nx
```

Donde $p_1,\ p_2,\ p_3$ *y* p_4 son las posiciones deseadas para los cuatro polos del sistema, las cuales se elegirán dependiendo las especificaciones requeridas, en este caso se propondrán dos ejemplos, el de uso de un controlador lento y el de uno rápido, así se tendrán dos tipos de controlador con respuestas diferentes. Para empezar dichos polos serán situados en las situaciones $[-2 \pm j,\ -20,\ -80]$ resultando la $\mathbf{K}$ $[-1828.6,\ -1028.6,\ 2008,\ 104]$ y la $\bar{N}$ -1828.6, por lo que el tiempo de establecimiento sería de 1.57 segundos (para un rango de error del 5 %), un tiempo de pico de 3.14 segundos y una sobreoscilación M_p de 0.18 %.

Mientras que si los polos fuesen situados en las posiciones $[-0.25 \pm j,\ -20,\ -80]$ se tendrá un controlador más lento y con más sobreoscilación, ya que los polos principales están más próximos al origen, en este caso resultaría $\mathbf{K}$ $[-242.9,\ -129.5,\ 1651,\ 100.5]$ y la $\bar{N}\ -242.85$. Para el que se tiene un tiempo de establecimiento de 12.56 segundos, un

tiempo de pico de 3.14 segundos y una sobreamortiguación del 45 %. Este controlador se diseñará utilizando ecuaciones lineales, pero la función de MATLAB contendrá el modelo no lineal para que se asemeje más a la realidad. Al diseñar el controlador para el modelo lineal y observar los resultados previstos en el sistema lineal, podremos identificar los posibles problemas que podrían surgir al aplicar el controlador al sistema no lineal. Por ejemplo, una acción de control brusca podría generar una fuerza centrípeta significativa. Para implementar el sistema no lineal en la función de MATLAB, se deben ingresar directamente las ecuaciones sin linealizar, como se muestra en el siguiente código:

```
x1p=x2;
x2p=(m*g*sin(x3)-m*x1*(x4)^2)/((J/R^2)+m);
x3p=x4;
x4p=(1/J)*(-b*x4+u);
```

Por lo que, una vez definidas en la función las constantes y el controlador faltaría el algoritmo de control, calculando la acción de control (u) mediante la ecuación genérica del control por espacio de estados $u = -\mathbf{K} \cdot \mathbf{x} + \bar{N} \cdot r$.

4.1.7.- Control de un motor

El siguiente modelo a implementar se enfoca en el control de posición de un motor. Este modelo permite establecer una referencia y garantizar que el motor se posicione en el punto requerido. Es una oportunidad para que los alumnos realicen pruebas y diseñen otro modelo típico de la Ingeniería de Control. El sistema abarca tanto un subsistema eléctrico como uno mecánico, lo cual es de gran relevancia en la docencia, ya que el alumno tendrá que llevar a cabo el análisis de este modelo que brinda una comprensión integral de los sistemas de control en aplicaciones que involucran componentes eléctricos y mecánicos.

4.1.8.- Modelado teórico del motor

En este sistema se controlará la posición del motor variando la tensión de entrada del motor.

Para el diseño del sistema se ha hecho uso del modelo de motor DC propuesto en el libro Franklin [9]. El sistema está compuesto de dos subsistemas, que se pueden observar en la figura 4.12, el subsistema eléctrico (sería la parte de la armadura del circuito) se puede simplificar como nuestra fuente de tensión como elemento activo y una resistencia (R) e inductancia (L) eléctrica del arrollamiento y la fuerza contra electromotriz (e), que llamaremos FCEM. Por otro lado, en la parte del subsistema mecánico se tendrán el par ejercido (T), la posición del eje del motor que denominaremos θ_m, la fuerza de fricción ($b \cdot \dot{\theta_m}$) y el momento de inercia calculado mediante la ecuación $J_m \cdot \ddot{\theta_m}$.

Además, la fuerza contra electromotriz se puede expresar mediante la ecuación $e =$

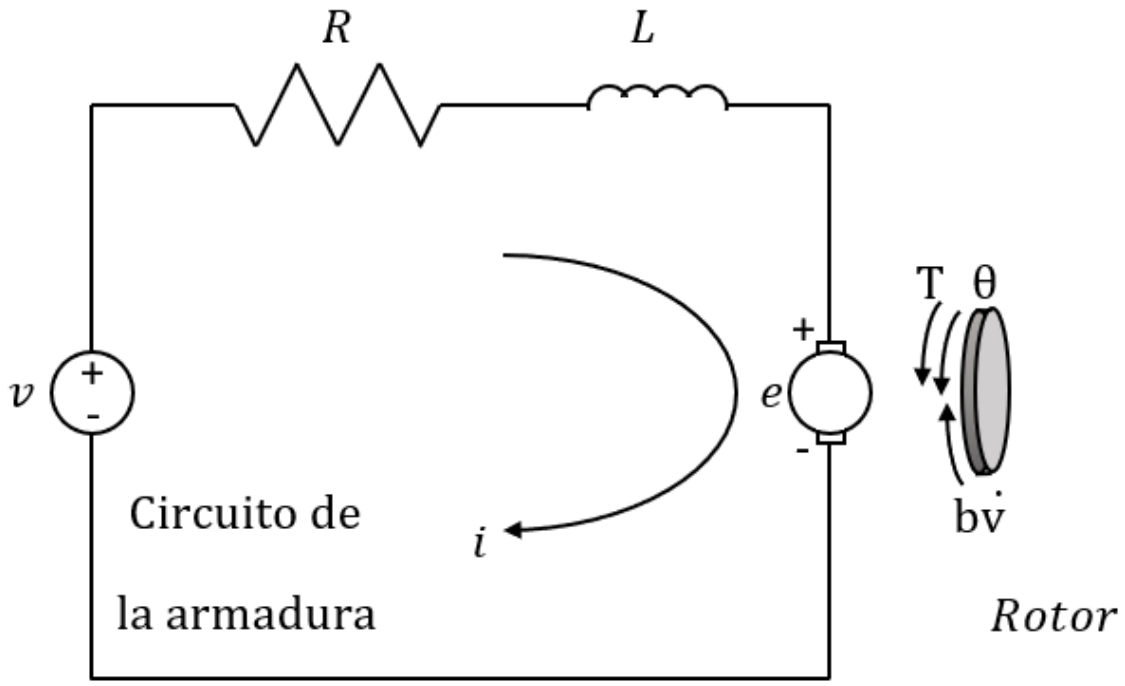


Figura 4.12: Diagrama de cuerpo libre de un motor

$K_e \cdot \dot{\theta}_m$ (directamente proporcional a la velocidad angular de motor), siendo K_e la constante de FCEM. Como ya tenemos la variable que relaciona las dos partes del sistema ahora despejaremos las ecuaciones:

$$J_m \cdot \ddot{\theta}_m + b \cdot \dot{\theta}_m = K_t \cdot i_a \tag{4.9}$$

$$L_a \frac{di_a}{dt} + R_a \cdot i_a = v_a - K_e \cdot \dot{\theta}_m \tag{4.10}$$

Estas ecuaciones son resultado de aplicar la ley de mallas al subsistema eléctrico (ecuación 4.9) y la segunda ley de Newton (ecuación 4.10). En este ejemplo se utilizarán los valores de las variables: $R_a = 1.1648$, $L_a = 0.0068$, $K_t = 0.55$, $K_e = 0.82$, $b = 0.00776$, $J_m = 0.0271$. Ahora, como en el modelo anterior, se cambiará el nombre de las variables de estado para su próxima identificación en el código de MATLAB:

$$\begin{aligned} x_1 &= \theta_m \\ x_2 &= \dot{\theta}_m \\ x_3 &= i_a \end{aligned}$$

Se identificarán las ecuaciones que igualan las derivadas de los estados con éstos, haciendo uso de las ecuaciones 4.9 y 4.10, quedando las ecuaciones:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{K_t \cdot x_3 - b \cdot x_2}{J_m} \\ \dot{x}_3 &= \frac{v_a - K_e \cdot x_2 - R_a \cdot x_3}{L_a} \end{aligned}$$

En la que se ha introducido el término correspondiente a K_t, referido a la constante de par.

4.1.9.- Desarrollo del modelo en Simulink del motor

En esta sección, como en el modelo del ball and beam, se estudiará el desarrollo del modelo en el programa "Matlab-Simulink", primero con los bloques del Simulink y después con el desarrollo del diseño en Espacio de Estados en la función de Matlab.

Para hacer el control en EE habrá que llegar a una ecuación genérica que relacione las derivadas de los estados con sus derivadas de la forma $\dot{\mathbf{x}} = \mathbf{F} \cdot \mathbf{x} + \mathbf{G} \cdot u$, donde $\dot{\mathbf{x}}$ es un vector columna 3×1 que contiene las derivadas de los estados, $\mathbf{F}$ es una matriz 3×3, $\mathbf{x}$ es un vector columna 3×1 que contiene los valores de los estados, $\mathbf{G}$ es una matriz 3×1 y u es la acción de control que será el voltaje aplicado mediante la fuente de tensión.

Entonces las ecuaciones quedarán de la forma:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{b}{J_m} & -\frac{K_t}{J_m} \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L_a} \end{bmatrix} \cdot u \tag{4.11}$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + [0] \cdot u \tag{4.12}$$

Como en el ejemplo anterior, se tiene que realizar el estudio de la observabilidad, para el que se adaptará la ecuación 3.4, siendo nuestro sistema de rango 3, quedando la ecuación $\mathbf{C} = [\mathbf{G};\ \mathbf{FG};\ \mathbf{F}^2 G]$, que calculado en MATLAB resulta la matriz:

$$10^6 \cdot \begin{bmatrix} 0 & 0 & 0.0030 \\ 0 & 0.0030 & -0.512 \\ 0.0001 & -0.0252 & 3.9550 \end{bmatrix}$$

Dicha matriz es de rango 3, mismo rango que el de nuestro sistema, por lo que podemos concluir que el sistema es controlable.

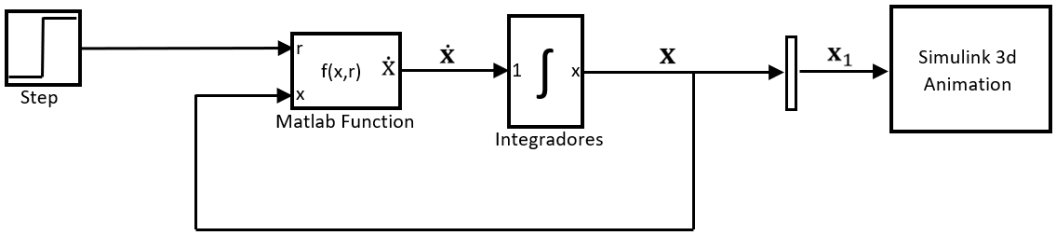


Figura 4.13: Diagrama de bloques del sistema del motor en cadena abierta

El diseño en bucle abierto consiste en aplicar una tensión que modificaremos como variable de entrada, por lo que el motor se moverá con una velocidad dependiente de dicha tensión, en este caso al diagrama 3D solo la variable x_1. Este diseño en Simulink correspondería a la figura 4.14:

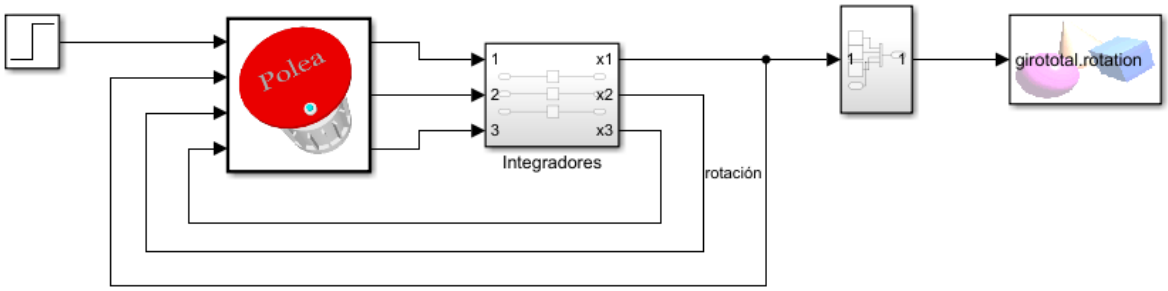


Figura 4.14: Sistema del motor en cadena abierta en Simulink

4.1.10.- Diseño del sistema controlado del motor

Ahora se diseñará el sistema en cadena cerrada, introduciendo el código específico para implementar el controlador, por lo que ahora se controlará la acción de control estableciendo una referencia (en radianes) de la posición deseada.

El diagrama de bloques para el sistema de motor introduciendo el controlador en la "s-function" tiene la implementación de la figura 4.16, en la que, como se introduce una referencia entre 0 y 1 con el bloque "joystick input" se multiplica con el bloque "multiply" con

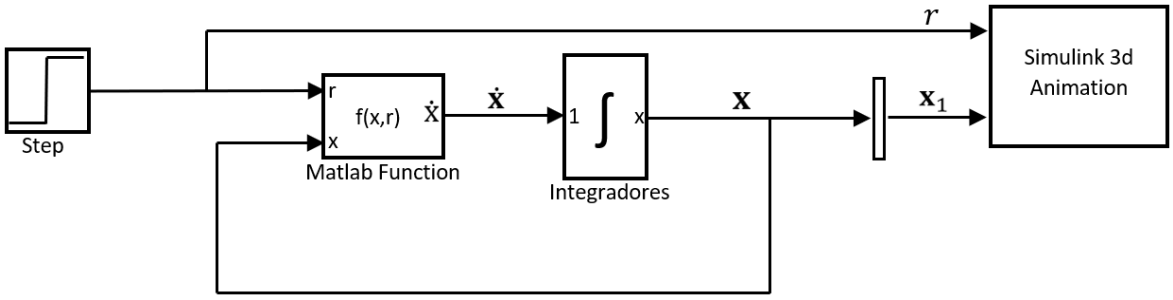


Figura 4.15: Diagrama de bloques simplificado del motor

valor $2 \cdot \pi$ para así tener en radianes el total del giro del motor.

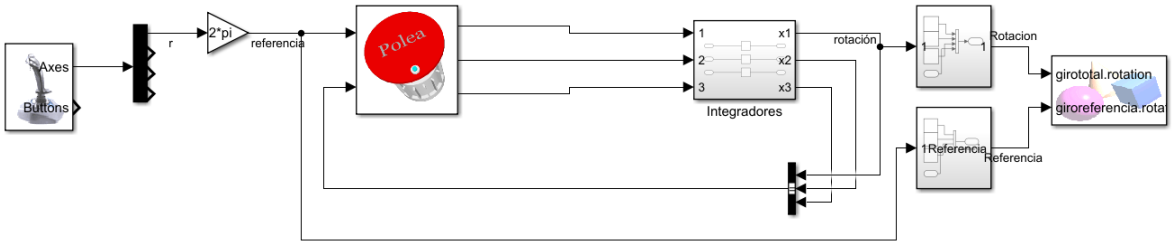


Figura 4.16: Sistema simplificado del motor en Simulink

Este diagrama muestra el típico diseño básico, en la que la referencia entra a la `Matlab Function` junto con los valores de los estados, obteniendo el valor de los estados. Para este ejemplo, como se necesita el valor de la rotación de la referencia y del eje, se establecerá como entrada al bloque 3D, el valor de la referencia establecida y el primer estado x_1 respectivamente.

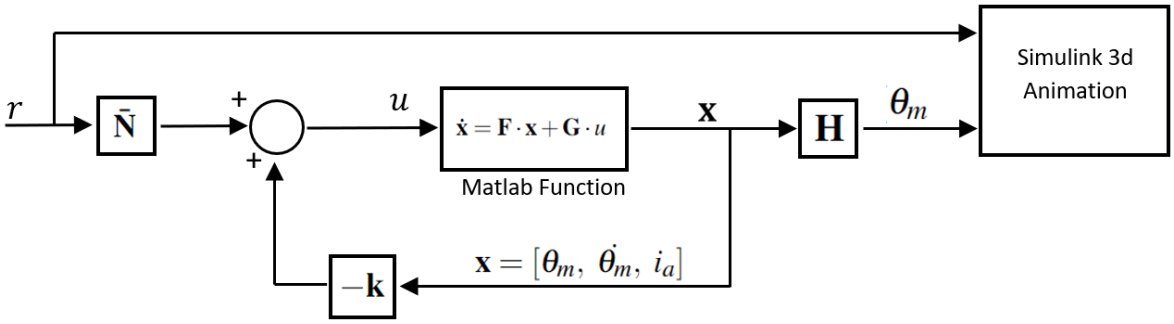


Figura 4.17: Diagrama de bloques separando planta y controlador del sistema del motor

También resultaría interesante separar controlador de la función de MATLAB en el contexto mostrado en la figura 4.17. En este enfoque, el contenido de la función se refiere al propio sistema, el cual se proporcionaría a los alumnos mediante la instrucción “pcode”, permitiéndoles diseñar el controlador de acuerdo con el método que elijan.

El diseño separando la planta y el controlador del sistema del motor de la figura 4.17 tendría como implementación en Simulink la mostrada en la figura 4.18 en la que también se multiplica el valor obtenido por el joystick por $2 \cdot \pi$ para poder situar la referencia en todo el recorrido del eje.

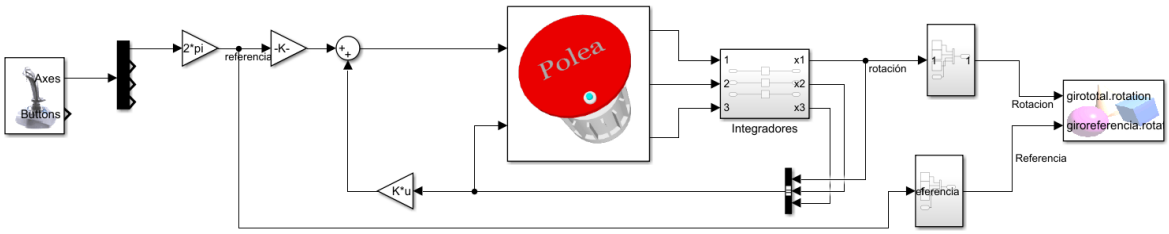


Figura 4.18: Sistema del motor separando planta y controlador en Simulink

4.1.11.- Diseño del controlador en Espacio de Estados para el motor

A continuación se diseñará el controlador y su implementación para programar el sistema expuesto anteriormente. Una vez identificadas las matrices **F**, **G**, **H** *y* **J**. El siguiente paso será identificar el lugar de las raíces del sistema, que serán obtenidas con el siguiente código de Matlab (en el que anteriormente habremos definido las constantes del sistema y las matrices):

```
motor_ss=ss(F,G,H,J);
pzmap(motor_ss)
polos=pole(motor_ss)
```

Con este código se obtendrá un diagrama gráfico del lugar de las raíces (figura 4.19 además del lugar exacto de la situación de los polos que aparece por consola, estando estos en las posiciones 0, -16.0511 y -155.5294, por lo que es un sistema limitadamente estable, también podemos observar que se tiene un cero en la posición -20.6.

Figura 4.19 procederemos al diseño del controlador con el mismo código de Matlab que empleamos anteriormente en el modelo *ball and beam*, pero en este caso, como el sistema tiene tres polos, se tendrá que cambiar la primera línea.

```
K=place(F,G,[p1 p2 p3])
Nt=inv([F G;H J]*[zeros(size(G));1]
Nu=Nt(end)
Nx=Nt(1:end-1)
Nbar=Nu+K*Nx
```

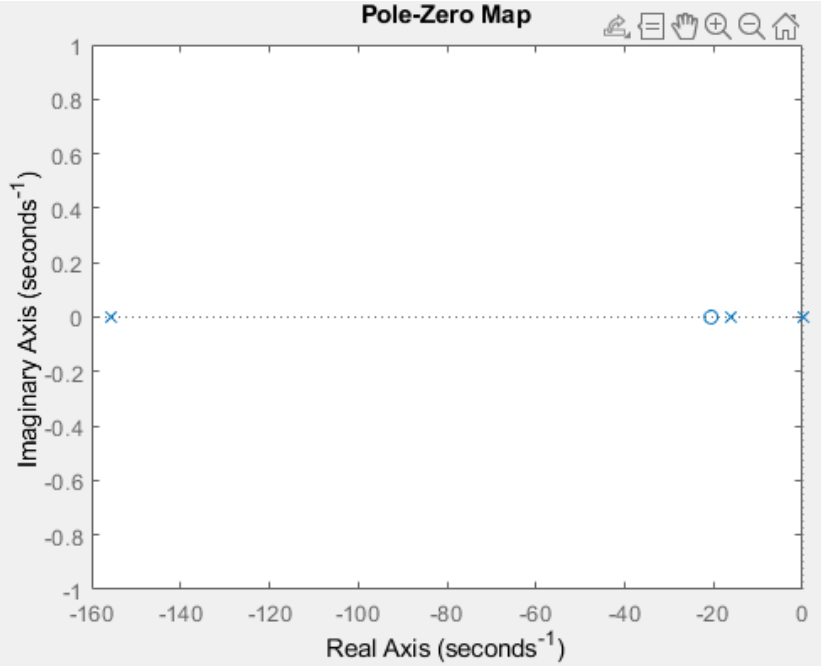


Figura 4.19: Mapa de polos y ceros del sistema del motor

Donde $p_1,\ p_2$ *y* p_3 son las posiciones deseadas para los tres polos del sistema, las cuales se elegirán dependiendo de las especificaciones requeridas. Para empezar situaremos dichos polos en las situaciones $[-1,\ -2,\ -5]$ resultando la $\mathbf{K}$ $[0.0033,\ -0.8150,\ -1.1123]$ y la $\bar{N}$ 0.0033, por lo que el t_s sería de 4.88 segundos (para un rango de error del 5 %), no teniendo sobreamortiguación

También se puede hacer que el sistema tenga una respuesta más rápida y subamortiguada, por ejemplo, estableciendo los polos en las situaciones $[-50 \pm 50j,\ -100]$ con una $\mathbf{K}$ de $[20.9409,\ 0.4269,\ -0.4867]$ y la $\bar{N}$ 20.9409. Para lo que tendremos un t_s de 0.1256, un t_p de 0.12 y una sobreoscilación 4.32 %.

Por lo que, una vez definidas en la función las constantes y el controlador faltaría el algoritmo de control, calculando la acción de control (u) mediante la ecuación genérica del control por espacio de estados $u = -\mathbf{K} \cdot \mathbf{x} + \bar{N} \cdot r$.

4.2.- Desarrollo de la interfaz gráfica y funcionalidades

4.2.1.- Aspectos básicos del diseño 3D

Como ya hemos descrito en el subapartado 2.2.3, para el diseño del modelo 3D se empleará un archivo X3D, creando y editando el texto con el editor de texto sublime. A con-

tinuación se explicará la programación de un diseño básico en formato X3D, para el conocimiento de los comandos para llevarlo a cabo se ha hecho uso de la página web [10], en la que se exponen varios ejemplos de creación de objetos 3D y se muestran también otras funcionalidades que se pueden añadir al 3D.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.2//EN"
"http://www.web3d.org/specifications/x3d-3.2.dtd">
<X3D profile="Interchange" version="3.2" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema-instance" xsd:
noNamespaceSchemaLocation="http://www.web3d.org/
specifications/x3d-3.2.xsd">

 <Scene>
  <Background skyColor='255 255 0'/>
  <Transform DEF="Tabla" translation='0.5 0.5 0.5'
  rotation="0 0 1 0.3">
   <Shape>
    <Appearance><Material diffuseColor='0 0 0'></
    Material></Appearance>
    <Box size='1 0.02 0.4'></Box>
   </Shape>
  </Transform>
 </Scene>
</X3D>
```

Figura 4.20: Ejemplo de implementación de un diseño 3D

La figura 4.20 muestra como se realizaría el modelo de una tabla de color negro con un fondo amarillo. Para que el archivo tenga un formato reconocible por Simulink, la cabecera de dicho archivo tiene que contener el mismo código que en nuestro ejemplo.

Cada comando que figure entre símbolos de comparación como por ejemplo `<Scene>` tiene que cerrarse con una barra inclinada después del primer símbolo tal como `</Scene>` cuando hayamos incluido todo el código dentro de la función.

Podremos modificar el color del fondo con la instrucción “Background skyColor” en el que, como con todos los comandos de modificación de colores, lo introduciremos en código RGB entre comillas simples, en nuestro caso siendo el color amarillo.

4.2.2.- Modificación de características y propiedades

Si ese objeto está dedicado a la posterior modificación de alguna de sus características dentro del modelo de Simulink, será necesario meter el objeto dentro de un bloque “Transform” definiendo el nombre del objeto creado y los atributos que se van a modificar, en este caso la translación y la rotación, si se incluye dentro de esta transformación un grupo de objetos en vez de un único objeto se aplicará el cambio a todo el grupo. A continuación se expone un ejemplo de la creación de una transformación con la que se podría modificar la rotación y la translación.

```
<Transform translation='0,0,0' rotation='0,0,0,0'>
<Shape>
<Appearance><Material diffuseColor='1 1 1'></Material></Appearance>
<Box size='1,1,1'></Box>
</Shape>
</Transform>
```

Con el que se crearía una caja de lado unidad la cual se puede rotar y trasladar (aunque en este ejemplo no hay ninguna variación la figura inicial puede estar con estos valores cambiados) y si se acopla al Simulink mediante el bloque “VR Sink” se podrá seleccionar la característica o características que se van a modificar.

Cabe destacar que en el caso de la translación definiremos 3 valores, el primero será el “eje x”, el segundo el “eje y” y el último, el “eje z”. Asimismo, en la rotación se necesitan 4 entradas, las tres primeras para especificar el eje sobre el que se va a girar y la última serviría para especificar el ángulo girado sobre dicho eje, en radianes. Dependiendo de las entradas que componen las características a modificar en Simulink se crearán buses de datos con el bloque “Bus Creator” y se enlazarán a su entrada correspondiente en el modelo 3D. Ahora se expondrá un ejemplo en Simulink.

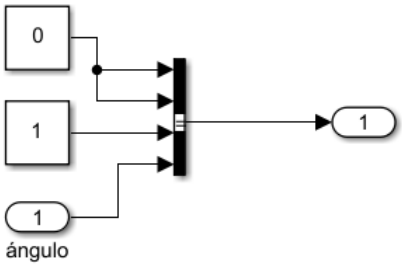


Figura 4.21: Ejemplo en Simulink de un grupo de rotación

La figura 4.21 muestra un grupo de rotación típico necesario para poder modificar la rotación del modelo 3D, en ella se crea un bus de datos siendo las tres primeras entradas el eje sobre el que va a girar, por lo que esta combinación de valores (vector $[0,\ 0,\ 1]$) corresponderá a hacer un giro sobre el “eje z” con el valor en radianes del ángulo correspondiente a la entrada del cuarto valor del bus de datos.

En la figura 4.22 se expone el ejemplo de un grupo de traslación. En esta configuración, el objeto cambiaría su posición en la dirección del “eje x” según el valor de la variable de entrada “posición”, además de estar trasladado permanentemente una unidad en el “eje z”.

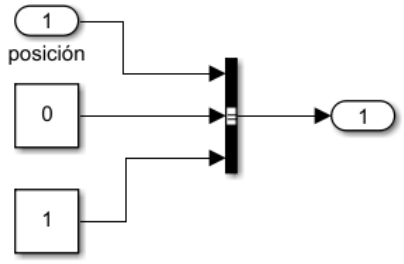


Figura 4.22: Ejemplo en Simulink de un grupo de translación

Una vez creado el objeto, que puede ser un rectángulo (código “Box”), una esfera (“Sphere”) o un cilindro (“Cylinder”) se deben establecer cada uno de sus atributos, como el radio, el lado o la altura, entre otros. Además, es posible modificar la apariencia del material utilizando el comando “Appearance”, para ajustar el color, la textura, entre otros aspectos.

En nuestro caso, solo introduciremos cambios en la rotación y translación del modelo, pero también se podrían modificar otras características, con otros grados de libertad... Por ejemplo, en el caso de diseños hidráulicos se podría modificar el color para mostrar la temperatura o presión de un fluido. Otro caso sería la fuerza ejercida por un motor, pudiendo cambiar el tamaño de un vector 3D dependiendo de la fuerza ejercida por este.

También se podría hacer el modelo con softwares gráficos, como programas CAD, los cuales podremos importar y nos proporcionan un gran número de ventajas, ya que tendremos un mayor control sobre el diseño y las posibles transformaciones de cada objeto que diseñemos, pero es conveniente haber empezado con modelos sencillos como los expuestos en este TFG, haciendo uso del lenguaje X3D. Estos programas serían más convenientes para modelos complejos en los que el lenguaje de programación X3D crearía complicaciones a la hora el diseño y también al importarlos en Simulink.

4.2.3.- Modelos 3D del proyecto

En la figura 4.23 se puede observar el diseño del motor (imagen izquierda) y el del *ball and beam* (imagen derecha).

El modelo del motor consta de los elementos decorativos de la parte posterior que intenta asemejarse a un motor real y los elementos móviles, que son la polea (que rota con la marca de posición, el eje del motor y el texto) y la referencia de posición. Por lo que las transformaciones del modelo 3D será el conjunto de giro total y la de la rotación de la referencia.

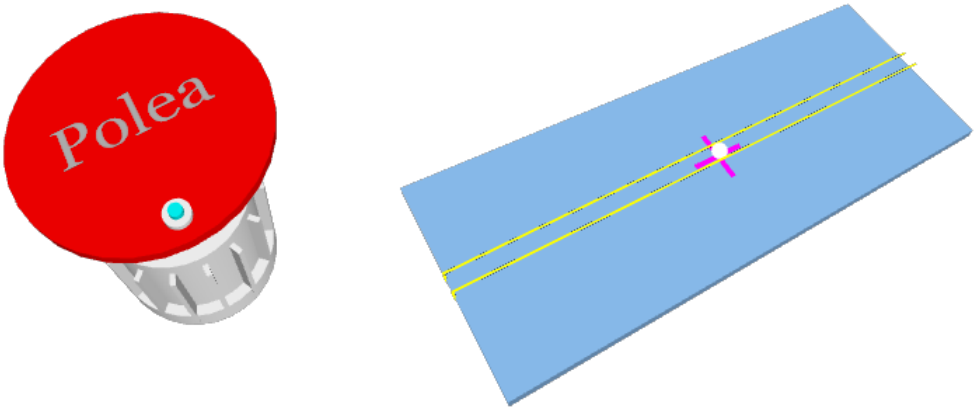


Figura 4.23: Modelos del motor y *ball and beam*

El modelo del *ball and beam* también consta de algunos elementos decorativos compuestos por cuatro barras amarillas, que indican que la bola solo se moverá en dos coordenadas. Las partes móviles que componen las transformaciones serían el conjunto tabla-bola en la que también se engloba la referencia, puesto que estos elementos rotarán conjuntamente. Dentro de esta transformación hay otras dos transformaciones de translación, la de la bola y la de la referencia.

Ahora se van a exponer las partes más importantes de los comandos usados en ambos modelos, empezando por aspectos generales y después profundizando en cada modelo específico. Lo primero sería la cabecera del archivo el cuál va a cambiar dependiendo si se está editando o si por el contrario, ya se va a introducir en el Simulink. La diferencia está en que para hacer más sencilla la edición se ha creado un archivo con extensión “html”, por lo que si abrimos este archivo podremos visualizar el diseño 3D en el navegador web, y si hacemos un cambio solo habría que guardarlo y después refrescar la página. El código de cabecera en este caso sería:

```
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge"/>
<title>My first X3DOM page</title>
<script type='text/javascript' src='https://www.x3dom.org/download/x3d
om.js'> </script>
<link rel='stylesheet' type='text/css' href='https://www.x3dom.org/dow
nload/x3dom.css'></link>
</head>
```

Para crear el archivo con extensión “x3d” habría que hacer un cambio en la cabecera, quedando el código de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.3//EN" "http://www.web3d.or
g/specifications/x3d-3.3.dtd">
<X3D profile='Immersive' version='3.3' xmlns:xsd='http://www.w3.org/200
1/XMLSchema-instance' xsd:noNamespaceSchemaLocation='http://www.web3d.o
rg/specifications/x3d-3.3.xsd'>
```

Cabe destacar, que como se ha expuesto anteriormente, es necesario cerrar todos los comandos abiertos en el modelo por lo que, al final del documento hay que agregar el código “$</html>$” para el primero y “$</X3D>$” para el segundo. También se puede cambiar el color del fondo del modelo, para ello seguiremos con el código:

```
<Scene>
<Background skyColor='0.2 0.2 0.8'/>
<NavigationInfo type="EXAMINE" headlight="true"/>
```

Y ahora se procederá a añadir los elementos específicos de cada modelo, las transformaciones y los objetos. Si se requiere que varios objetos roten o se trasladen solidariamente habrá que agruparlos, para esto se creamos transformaciones que contengan todos los objetos del grupo, ejemplo del código necesario podría ser el de la bola y la tabla:

```
<Transform DEF="tabla_bola" rotation='0 0 1 0'>
<Group>
<!-- tabla -->
<Group>
<Transform translation='0,-0.275,0'>
<Shape>
<Appearance><Material diffuseColor='0 152 204'></Material></Appearance>
<Box size='10,0.2,4'></Box>
</Shape>
</Transform>
</Group>
<!-- bola -->
<Transform DEF="bola" translation='0,0,0'>
<Group>
<Shape>
```

```
<Appearance>
<Material diffuseColor="0.7 0.7 0.7"></Material>
</Appearance>
<Sphere DEF='bola' radius='0.15'></Sphere>
</Shape>
</Group>
</Transform>
<!-- marca referencia -->
<Transform DEF="referencia" translation='0,-0.271,0'>
<Group>
<Shape>
<Appearance><Material diffuseColor='51 0 102'></Material></Appearance>
<Box size='1,0.2,0.1'></Box>
</Shape>
<Shape>
<Appearance><Material diffuseColor='51 0 102'></Material></Appearance>
<Box size='0.1,0.2,1'></Box>
</Shape>
</Group>
</Transform>
</Group>
</Transform>
```

Aunque en esta transformación también estarían incluidos los elementos decorativos, que forma parte del conjunto de rotación. Con el código implementado del objeto 3D, además de la rotación, se podrá cambiar la translación de la marca de referencia y la translación de la bola.

En el caso del motor, el estator junto con las decoraciones traseras y laterales serán los elementos fijos, mientras que el eje del motor, la polea, la marca de posición y el texto estarán dentro de un grupo de rotación y la marca de referencia a parte.

```
<!-- stator-->
<Transform translation='0,-18,0'>
<Shape>
<Appearance><Material diffuseColor='1 1 1'></Material></Appearance>
<Cylinder height='15' radius='7'></Cylinder>
</Shape>
```

```
</Transform>
<!-- parte trasera stator -->
<Transform translation='0,-23,0'>
<Shape>
<Appearance><Material diffuseColor='1 1 1'></Material></Appearance>
<Cylinder height='3' radius='8'></Cylinder>
</Shape>
</Transform>
<Transform DEF="girototal" rotation='0,1,0,0'>
<Group>
<!-- polea -->
<Transform translation='0,0,0'>
<Shape>
<Appearance><Material diffuseColor='1 0 0'></Material></Appearance>
<Cylinder height='1' radius='10'></Cylinder>
</Shape>
</Transform>
<!-- texto de la polea -->
<Transform rotation='1,0,0,-1.57' translation='0,1,0'>
<Shape>
<Text DEF='textoPolea' string='"Polea"'>
<FontStyle DEF='testFontStyle' justify='"MIDDLE" "MIDDLE"' size='5.0'/>
</Text>
<Appearance>
<Material diffuseColor='0.5 0.5 0.5'/>
</Appearance>
</Shape>
</Transform>
<!-- masa excéntrica -->
<Transform translation='0,0,8'>
<Shape>
<Appearance><Material diffuseColor='1 1 1'></Material></Appearance>
<Cylinder height='2' radius='1'></Cylinder>
</Shape>
</Transform>
<!-- eje -->
<Transform translation='0,-11,0'>
<Shape>
```

```
<Appearance><Material diffuseColor='1 1 1'></Material></Appearance>
<Cylinder height='22' radius='1'></Cylinder>
</Shape>
</Transform>
</Group>
</Transform>
<Transform DEF="giroreferencia" rotation='0,1,0,0'>
<Group>
<!-- referencia de posición -->
<Transform translation='0,0,8'>
<Shape>
<Appearance><Material diffuseColor='0 1 1'></Material></Appearance>
<Cylinder height='3' radius='0.5'></Cylinder>
</Shape>
</Transform>
</Group>
</Transform>
```

Con esto tendremos los elementos más importantes del motor sin las decoraciones, estas están compuestas por un cilindro en la parte trasera del estator, unas placas cúbicas iguales pero rotadas 45 grados unas de otras y una caja para parecerse a la caja de conexiones de un motor real. El código de la primera placa (no rotada) será:

```
<Transform translation='0,-18,0' rotation='0,1,0,0'>
<Shape>
<Appearance><Material diffuseColor='1 1 1'></Material></Appearance>
<Box size='16,12,0.5'></Box>
</Shape>
</Transform>
```

4.3.- Integración con plataformas de e-learning

Actualmente, la herramienta principal de los alumnos en el aprendizaje son las plataformas de e-learning, como el campus virtual de las universidades, la plataforma moodle, etc. Estas plataformas ofrecen acceso al material docente necesario para el aprendizaje, que incluye recursos como apuntes, libros digitales, vídeos explicativos y actividades interactivas. Gracias a estas plataformas, los alumnos tienen la posibilidad de acceder de manera adecuada y autónoma a todos los recursos educativos, facilitando así su proceso de aprendizaje.

Integrar laboratorios virtuales en estas plataformas supondría que los alumnos tendrían acceso en todo momento a practicar sus intentos de configuración, por lo que, dependiendo de su progreso en entender los conceptos, podrían intentar diseñar el control de un modelo u otro. Esta flexibilidad permitiría a los estudiantes experimentar con diferentes enfoques y estrategias de control, quienes tendrían la oportunidad de experimentar de manera activa y práctica.. Además, al poder repetir y ajustar sus intentos, los alumnos podrían consolidar su comprensión y mejorar sus habilidades de diseño de control de manera iterativa.

Otro aspecto beneficioso sería la posibilidad de incorporar un programa de evaluación para medir la precisión de los resultados obtenidos por los estudiantes. Esta evaluación se basaría en una función de coste que compararía los resultados generados con los resultados esperados, permitiendo a los alumnos monitorizar su progreso y a los profesores observar los resultados generales. La función de coste desempeñaría un papel clave al cuantificar la discrepancia entre los resultados obtenidos y los esperados, teniendo en cuenta factores como el error absoluto o relativo, y la precisión en la representación gráfica de los datos. Además de proporcionar una evaluación inmediata, la función de coste permitiría a los estudiantes identificar áreas de mejora y ajustar sus enfoques. Para los profesores, esta función sería útil al identificar patrones de errores comunes y áreas problemáticas, guiando así la adaptación del contenido y la metodología de enseñanza para una mejor eficacia del aprendizaje..

Los laboratorios virtuales podrían ser proporcionados a los alumnos como la función de MATLAB codificada (una “s-function” a la que se le ha aplicado el comando “pcode”), o directamente como el modelo de Simulink en cadena abierta. Esta incorporación permitiría a los estudiantes realizar prácticas y exámenes utilizando el programa Simulink, interactuando con los modelos de laboratorios virtuales. Al codificar la función, se les proporcionaría una herramienta interactiva y personalizable puesto que, por ejemplo, en un examen cada alumno podría tener un laboratorio virtual distinto con una dificultad similar o, aunque sea el mismo laboratorio virtual, se podrían cambiar los parámetros internos de cada modelo.

El artículo [1], publicado por la revista Elsevier muestra un ejemplo práctico de implementación de un laboratorio virtual y su introducción en el ámbito de la docencia. También expone las diferentes fases y maneras de introducir los laboratorios virtuales como herramienta en la docencia para que los alumnos dispongan de ella y su integración en plataformas de e-learning (en este caso en el Moodle de la universidad), cabe resaltar que el experimento también incluyó un laboratorio remoto, con el que los alumnos pueden probar sus diseños si están conectados al sistema y probar cómo funcionaría para un modelo real, pudiendo ver todo el proceso mediante una cámara. Todo ello permite al alumno asentar los conocimientos

adquiridos ya que podría diseñar el sistema y probarlo en el laboratorio virtual y, cuando ya tenga un diseño definitivo con los resultados esperados, probarlo en el sistema real haciendo uso del laboratorio remoto, completando así la fase de aprendizaje.

5. Evaluación y resultado

5.1.- Diseño experimental y protocolo de evaluación

5.1.1.- Modelo del ball and beam

Para comprobar el correcto funcionamiento del sistema diseñado se procederá a su simulación en el modelo en bucle abierto con un ángulo constante de -1.8º ($-0.01 \cdot \pi$ radianes). Se hará uso de un tiempo de simulación más elevado debido a que el tiempo de establecimiento del control del ángulo es de 3 segundos, por lo que para que haya tiempo suficiente para percibir el comportamiento del sistema cuando este ha alcanzado su régimen permanente se deberá establecer un tiempo de simulación de 15 segundos. La gráfica resultante de la posición de la bola respecto a la barra sería:

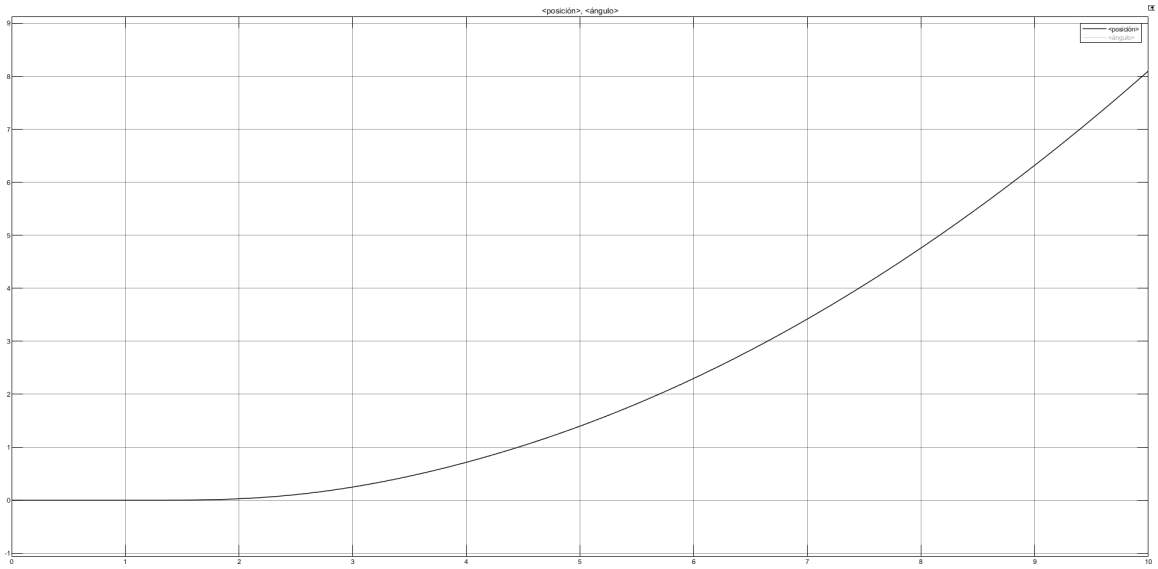


Figura 5.1: Simulación de la translación de la bola en cadena abierta

En dicha figura 5.1 se puede ver que la trayectoria de posición sigue la gráfica de un movimiento rectilíneo uniformemente acelerado, condición que concuerda con lo esperado, puesto que la bola está cayendo con aceleración constante sobre un plano inclinado con un ángulo constante.

Figura 5.2: Vídeo del sistema *ball and beam* en bucle abierto

El vídeo del enlace de la imagen 5.2, también accesible mediante el enlace `https://youtu.be/9KGuoGgFrCA`, muestra el resultado de la simulación del modelo del motor en el

que establecemos un ángulo variable mediante un `joystick`. Se muestran simultáneamente el resultado del 3D y el resultado de la medida de un bloque `scope`, que muestra el ángulo aplicado y la posición de la bola.

5.1.2.- Modelo del motor

En este caso, se podrá hacer la simulación también en código de Matlab y comparar gráficas, para ello usaremos el siguiente código, una vez definidas las variables:

```
%% En esta subseccion se definiran las matrices
s=tf('s');
F=[0 1 0;0 -b/Jm Kt/Jm; 0 -Ke/La -Ra/La]
G=[0 0 1/La]'
H=[1 0 0]
J=[0]
%% Situamos los polos en los lugares deseados
eig(F);
K=place(F,G,[-1,-2,-5]);
eig(F-G*K) % Comprobacion de la situacion de los polos
%% Calculamos el valor de Nbar
Nt=inv([F G;H J]) * [zeros(size(G));1];
Nu=Nt(end);
Nx=Nt(1:end-1);
Nbar=Nu+K*Nx
%% Recalculamos las matrices para dicha situacion de los polos
Fprim=F-G*k
Gprim=G*Nbar
Hprim=H-J*k
Jprim=J*Nbar
sistema_controlado=ss(Fprim,Gprim,Hprim,Jprim)
t=linspace(0,10,10000); % Crea un vector para simular t entre 0 y 10
    segundos
r=1*(t>1);
ylsim=lsim(sistema_controlado,r,t); % Calculamos la respuesta del
    sistema
figure; % Representamos la grafica
plot(t,r,t,ylsim,'LineWidth',2);
legend('escalon', 'respuesta');
```

En dicho código se establecerá el sistema original y se situarán los polos en los luga-

res deseados para después calcular las matrices del sistema controlado para dichos polos, teniendo que calcular también la $\bar{N}$ y el vector **K** siguiendo el código genérico empleado anteriormente. Finalmente se representará gráficamente, pudiendo modificar el tiempo de simulación y el paso de simulación cambiando los parámetros de la función `linspace` del código. Si se requiere podemos cambiar la amplitud de escalón y el instante de inicio, con la línea de código `r=1*(t>1)`.

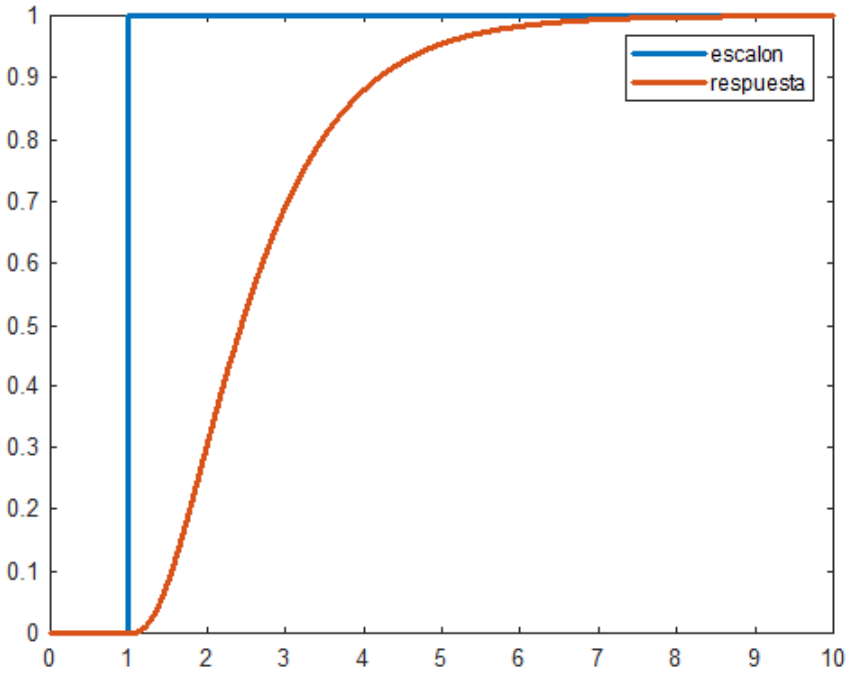


Figura 5.3: Respuesta a escalón del motor en Matlab

Si se compara la gráfica de la figura 5.3 con la gráfica resultante en el Simulink son idénticas (subsección 5.2.2).

Figura 5.4: Vídeo del sistema del motor en bucle abierto

En este caso 5.4 (hipervínculo `https://youtu.be/BgbYGh68dIU`) sería equivalente al otro vídeo solo que el bloque `scope` muestra la acción de control en voltaje aplicado y la rotación del eje.

5.2.- Resultados y análisis de la evaluación

5.2.1.- Modelo del ball and beam

Se precederá al análisis de las gráficas de los dos controladores para un mismo tiempo de simulación de 20 segundos y una referencia de 0.5 (para situar la bola sobre la mitad de la tabla).

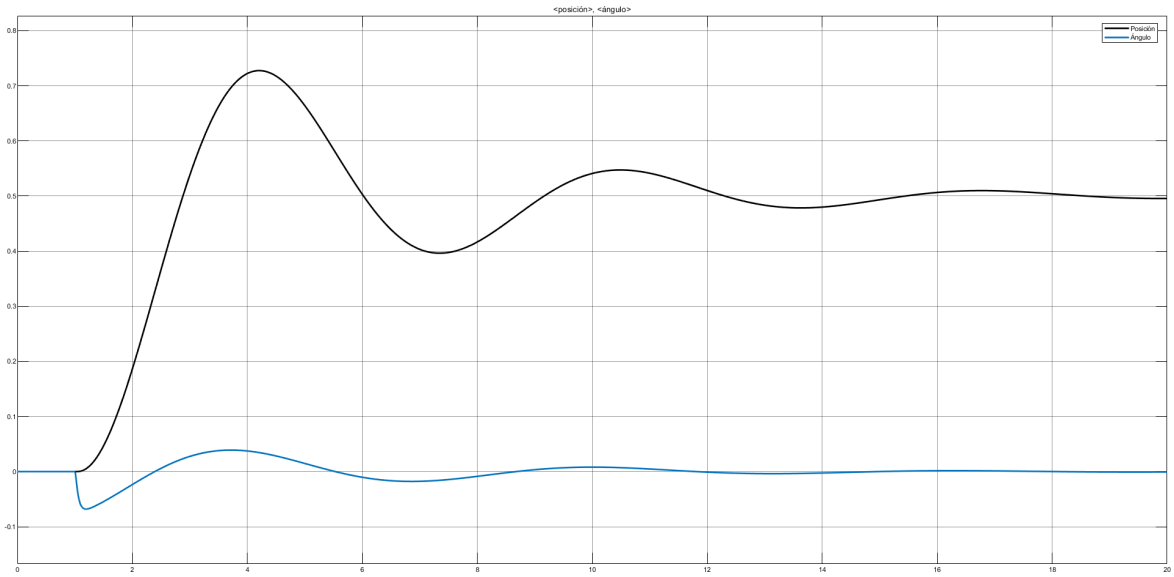


Figura 5.5: Simulación del ángulo y posición usando un controlador lento en el ball and beam

En este caso 5.5 se puede observar que la respuesta conlleva una acción de control menor, siendo esta menos brusca, pero implica que el sistema se comporte de manera más oscilatoria con tiempos de establecimiento mayores, cumpliendo con las características calculadas en el subapartado 4.1.11. Cabe destacar que el sobreamortiguamiento es considerable (del 45 %), por lo que para ciertas referencias la posición de la bola será mayor que la longitud de la propia barra.

Ahora, en la figura 5.6, como se ha cambiado a un controlador más rápido para cumplir con las especificaciones temporales (ya que el tiempo de establecimiento es mucho menor), el sistema requiere una acción de control mayor, aunque finalmente la sobreoscilación es menor.

Mediante el hipervínculo `https://youtu.be/dBni9hBOFNk`, o escaneando la imagen 5.7 se redirecciona al vídeo en el que se muestra el sistema ball and beam en funcionamiento. La referencia impuesta se modifica con un `joystick` y también se muestra que, para esta implementación de la función de Matlab si se pulsa el botón en la parte superior derecha se cambia de controlador (mostrado a la mitad de vídeo). En este caso, en el bloque `scope` se muestra la referencia establecida y la posición de la bola.

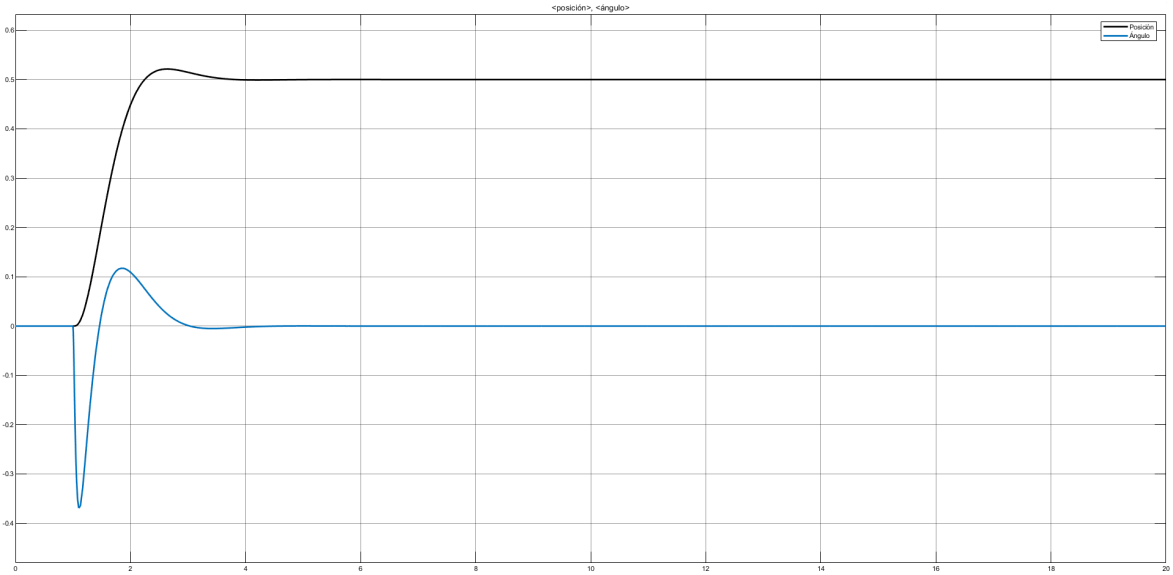


Figura 5.6: Simulación del ángulo y posición usando un controlador rápido en el ball and beam

Figura 5.7: Vídeo del modelo *ball and beam* controlado

Como se mencionó previamente, en el diseño del controlador se han utilizado ecuaciones lineales para el sistema, mientras que la función de MATLAB está programada con el modelo no lineal. Esto significa que en ciertos casos, como cuando se aplica una acción de control muy intensa, el sistema puede volverse inestable. En la figura 5.8 se puede observar cómo, al introducir una referencia de 3 unidades, el sistema requiere una gran acción de control, lo cual genera una fuerza centrípeta considerable y provoca la inestabilidad del sistema.

5.2.2.- Modelo del motor

Ahora vamos a observar el cambio en las gráficas introduciendo dos tipos de controladores con diferentes características: uno lento, sobreamortiguado y con un tiempo de establecimiento mayor, y otro rápido, subamortiguado y con un tiempo de establecimiento menor..

La respuesta de este controlador 5.9 es muy lenta, siendo el t_s alto, pero el voltaje requerido para establecer el motor en la posición requerida no es muy grande, por lo que dependiendo la aplicación que necesitemos será mejor emplear este tipo de controlador. Por ejemplo, si queremos diseñar un control de posición en un motor de gran tamaño (con un momento de inercia y una carga significante), aunque la posición deseada tarde en alcanzar-

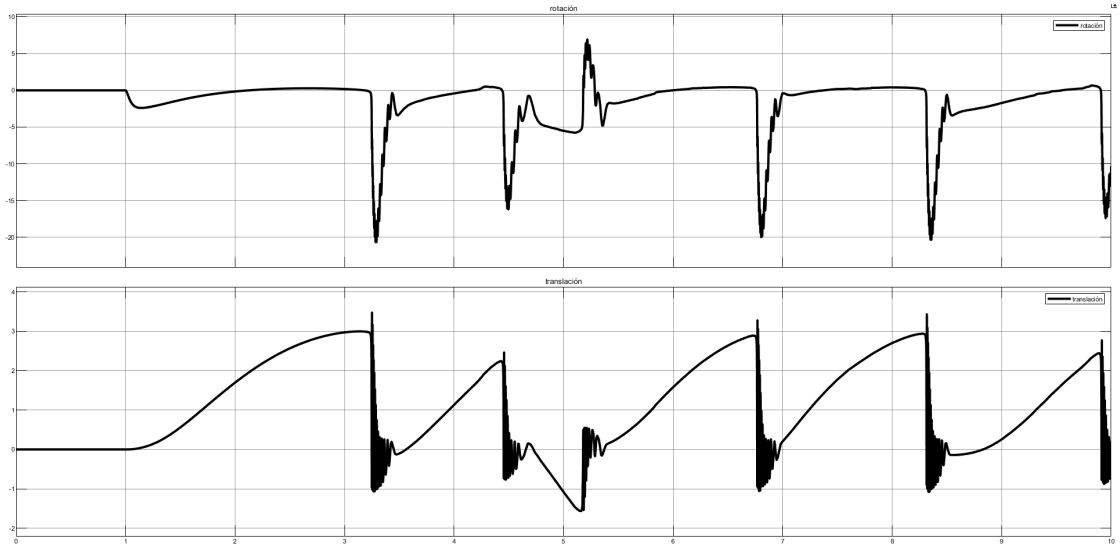


Figura 5.8: Gráfica del sistema ball and beam inestable

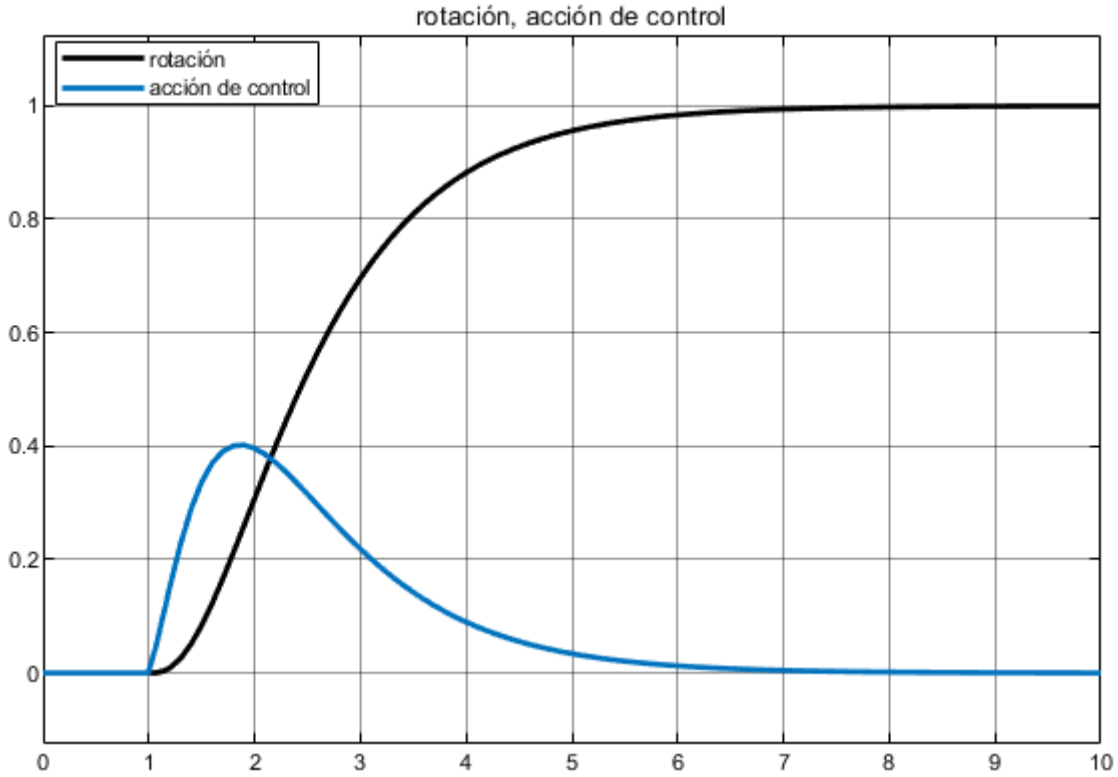


Figura 5.9: Control de la posición de un motor usando un controlador lento

se, se necesitará aplicar un voltaje menor que significará una sobrecarga menor de la red e incluso hacer el sistema eléctrico más seguro.

El controlador lento, figura 5.9, con un tiempo de establecimiento mayor, proporciona una respuesta del sistema muy lenta. Esto significa que la posición deseada del motor puede tardar más tiempo en alcanzarse. Sin embargo, una ventaja de este controlador es que el

voltaje requerido para establecer el motor en la posición deseada no es muy alto. Esto tiene implicaciones importantes en términos de la carga impuesta a la red eléctrica. Al requerir un voltaje menor, se reduce la sobrecarga en la red y se evita el riesgo de sobrecalentamiento o fallas eléctricas.

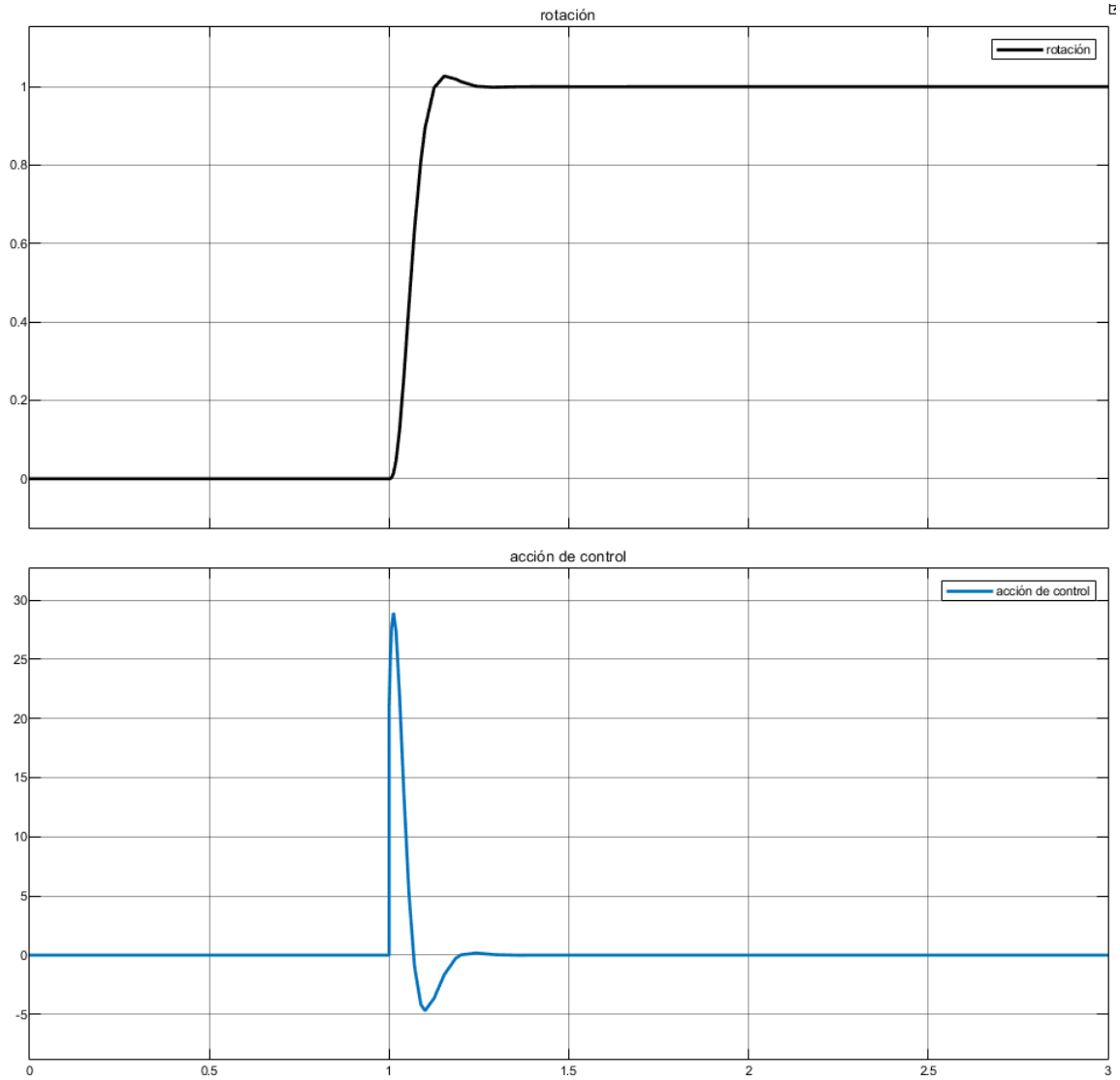


Figura 5.10: Control de la posición de un motor usando un controlador rápido

Por otro lado, el controlador rápido, cuya gráfica de la respuesta se puede apreciar en la figura 5.10, subamortiguado y con un tiempo de establecimiento menor permite que el sistema alcance el régimen permanente más rápidamente. La posición deseada del motor se alcanza en un tiempo más corto en comparación con el controlador lento. Sin embargo, debido a la respuesta más rápida, se requiere un voltaje más alto para lograr el mismo resultado en términos de posición. Esto puede generar una mayor carga en la red eléctrica y, en ciertos casos, puede suponer un riesgo de sobrecarga o inestabilidad en el sistema eléctrico.

La elección entre el controlador lento y el controlador rápido depende de la aplicación específica y sus requisitos. En el caso de un motor de gran tamaño con un momento de inercia y una carga significante, donde la posición deseada puede tardar en alcanzarse, el controlador lento puede ser más adecuado. Aunque la respuesta es más lenta, se necesita aplicar un voltaje menor, lo cual implica una sobrecarga menor en la red eléctrica y contribuye a un sistema eléctrico más seguro y estable.

El vídeo del enlace de la imagen 5.11, también pudiendo acceder con el enlace `https://youtu.be/uI0b3mBlRR4`, muestra el resultado de la simulación del modelo del motor controlado por un `joystick` variando la referencia. Se muestran simultáneamente el resultado del 3D y el resultado de la medida de un bloque `scope`, que muestra el cambio de referencia y la posición.

Figura 5.11: Vídeo del modelo del motor controlado

5.3.- Descripción de la plataforma de desarrollo

Se ha empleado la versión R2021a de Matlab, pero durante la realización del proyecto se han tenido que incorporar bibliotecas específicas para implementar distintas funcionalidades. Teniéndose que añadir la librería `Simulink 3D Animation` para poder importar el modelo 3D para su correcta visualización, esta librería soporta extensiones del tipo CAD, URDF, así como X3D y estándares ISO. De esta librería vamos a usar el bloque `VR Sink` desde el cuál se importa el modelo con las anteriores extensiones y se elijen las entradas (rotación, translación, tamaño...) que se quieren modificar. Cabe destacar que para modificar la translación se necesita crear un bus de datos con un bloque `bus creator` de Simulink, ya que se necesitan 3 entradas, así como en la rotación se necesitan 4 entradas en el bus, todo esto explicado más detalladamente en el subapartado 4.2.

Para poder conectar un joystick se necesitaría añadir la librería `Aerospace BlockSet` y la `Aerospace ToolBox`, que proporciona un bloque denominado `Pilot Joystick all` el cual proporciona salidas analógicas del joystick y también el valor digital de los pulsadores. Puede ser que el joystick no sea reconocido por el ordenador, en este caso se han usado el joystick y los botones de la consola “Play Station 3” en el sistema operativo “windows 10”.

Teniendo estas herramientas hace falta instalar otro software para permitir la conexión con el ordenador, siendo utilizado el programa gratuito "SCPServer" además de la necesidad de tener todos los *drivers* del ordenador actualizados así como las actualizaciones de su sistema operativo ya que si no están actualizados podría llevar a una posible inhabilitación de este.

La herramienta "s-function" que será utilizada será de nivel 2, lo que quiere decir que su contenido será en el lenguaje de programación propio de Matlab, ya que también se tendría la opción de utilizar una "s-function" de nivel 1, lo que quiere decir que el lenguaje de programación que usa es C++, esta "s-function" permite al usuario crear y personalizar bloques de funciones específicas para su aplicación. Para saber como realizar la programación de dicha "s-function" se ha hecho uso de un archivo predefinido de MATLAB usando el comando `edit msfuntmpl_basic` en la consola, el cuál abre una pestaña con el código de una "s-function" básica que contiene las funciones principales y ejemplos de cómo se crearían nuevas variables de entrada, salida o en la parte del archivo en la que se introduciría el código.

Ahora vamos a analizar el código fundamental que se usará de esta función de ejemplo proporcionada por MATLAB, no será comentado el código que no tiene utilidad práctica para estos ejemplo, por lo que se puede observar que en la función "setup(block)" se tendrán que establecer las características de cada entrada y salida a la función, teniendo que especificar en las dos primeras líneas de código el número total de entradas y salidas que tendremos, con el código de MATLAB:

```
% Register number of ports
block.NumInputPorts = 1;
block.NumOutputPorts = 1;
```

En la primera línea se establecerá el número de entradas y en la segunda el número de salidas. Seguidamente tendremos que configurar las características de cada entrada individualmente con el código:

```
% Override input port properties
block.InputPort(1).Dimensions = 1;
block.InputPort(1).DatatypeID = 0;
block.InputPort(1).Complexity = 'Real';
block.InputPort(1).DirectFeedthrough = true;
```

Habrá tantas veces esta secuencia de código como entradas tenga la función, en la primera línea estableceremos las dimensiones del vector de entrada, después habrá que definir el tipo de dato que contiene dicho vector, en este caso los datos numéricos tipo "double"

corresponderán al número “0” y los datos booleanos corresponderán al número “8”, que se puede emplear para cuando se usen los botones del mando. También se puede definir si es un número real o imaginario. Para configurar las salidas sería lo mismo que lo anterior, con lo que en la función quedaría:

```
% Override output port properties
block.OutputPort(1).Dimensions = 1;
block.OutputPort(1).DatatypeID = 0; % double
block.OutputPort(1).Complexity = 'Real';
```

Finalmente, para la implementación del modelo en espacio de estados se tiene que escribir el código en la función “Outputs(block)”, con el siguiente esquema:

```
% Parametros del modelo

% Asociacion de entradas a las variables
variable_de_entrada=block.InputPort(1).Data;
% Codigo de la implementacion en espacio de estados

%Traslado de las variables de salida
block.OutputPort(1).Data = variable_de_salida;
```

Aquí se muestra un ejemplo de cómo habría que definir las entradas y salidas. Cabe destacar que, para que la “s-function” esté bien implementada el nombre del fichero y de la función principal deben ser el mismo.

5.3.1.- Codificación de las funciones de MATLAB

También se hará uso de la instrucción `pcode(`nombre_del_archivo')` introducida en la ventana de comandos de Matlab, donde el nombre_del_archivo es el nombre del archivo en el que se encuentra la “s-function”, archivo que será cifrado, así se transforma en un archivo protegido con lo que no se podrá acceder al contenido explícito de código. Con esto se consigue proteger el contenido y ocultar toda la implementación que lleva consigo, a partir de ahora tendrá una extensión `.p` en vez de la extensión `.m` propias de las funciones de MATLAB. El archivo resultante podrá seguir siendo introducido en el bloque “Level-2 MATLAB S-Function” de Simulink, ya que sigue permitiendo acceder a sus funcionalidades pero no permite acceder al código interno, resultando casi imposible acceder al contenido. Estas características proporcionan un gran interés didáctico ya que nadie puede tener acceso al código, aportando valor añadido en la realización de exámenes, ya que el alumno no tiene la capacidad de acceder al modelo teórico programado en la “s-function”.

6. Conclusiones y trabajo futuro

6.1.- Conclusiones generales

El trabajo incluye dos ejemplos de laboratorios virtuales, que aunque sean ejemplos básicos, sirven para ilustrar cómo estos pueden ser utilizados en la docencia en control. Describen con bastante precisión lo que sería cada modelo físico y crean la posibilidad de mejorar sustancialmente la forma de entender los conceptos, pudiendo hacer uso de modelos que en la realidad sería muy costoso disponer en un laboratorio tradicional.

A continuación se van a enumerar las aportaciones al proyecto:

- Se ha realizado el modelado de los sistemas estudiados, estableciendo las ecuaciones que describen su comportamiento dinámico. A partir de estas ecuaciones, se ha procedido al desarrollo en espacio de estados, expresando las relaciones entre las variables de estado, entradas y salidas en forma matricial. Luego, se ha analizado la estabilidad y controlabilidad del sistema mediante la verificación de la matriz de controlabilidad. Esta aportación facilita el análisis y diseño de los sistemas, así como la identificación de las variables clave.

- Se ha realizado el desarrollo de los controladores para cada modelo, teniendo en cuenta las especificaciones requeridas. Se analizan y seleccionan las técnicas de control adecuadas para lograr los objetivos establecidos. Se definen los criterios de desempeño y las especificaciones relevantes para cada sistema. Se han empleado dos técnicas de control, como el controlador PD utilizado en el control del ángulo del modelo de bucle abierto del *ball and beam* y controladores de realimentación del estado. Esto ha permitido asegurar que los controladores desarrollados sean capaces de cumplir con los requisitos establecidos y garantizar un desempeño óptimo en los sistemas controlados así como las diferencias entre controladores.

- Se ha llevado a cabo el desarrollo de los modelos en Simulink, utilizando el programa Simulink para representar y simular el comportamiento de los sistemas. Se construyen los modelos mediante la interconexión de bloques funcionales, donde cada bloque representa una operación o función específica del sistema. Mediante la comparación con el diagrama de bloques general, permitiendo una comprensión visual clara de la dinámica y las interacciones entre los componentes. Estos modelos en Simulink proporcionan una herramienta poderosa para analizar y optimizar el rendimiento de los sistemas, así como para evaluar y verificar el diseño de los controladores desarrollados previamente.

- Se ha abordado el modelado del espacio tridimensional (3D) para representar de manera precisa y detallada el entorno físico en el que se desarrollan los sistemas. Se han utilizado archivos de extensión ".X3D" para crear modelos virtuales del espacio en el que se realizarán las simulaciones y pruebas. El modelado del espacio 3D permite una mejor comprensión del comportamiento de los sistemas en un contexto realista y proporciona una representación visual y espacial que facilita la toma de decisiones y el análisis de los resultados obtenidos, ayudando en la interacción del mundo físico con su entorno tridimensional.

Por lo que podemos considerar que este proyecto proporciona una idea general de cómo se elaboraría un entorno de laboratorio virtual para aplicarlo a la docencia de control y el gran beneficio que tendría para los estudiantes de ingeniería, proporcionándoles una visión más amplia de la materia y más ejemplos con los que poder trabajar. Además de introducir a los alumnos en el campo del control de sistemas complejos por computadora, cada vez más importante en la actividad industrial actual, con el nuevo concepto de "industria 4.0" que hace uso de conceptos como el *big data*, *digital twin*, *Internet of Things* (IoT) o Inteligencia artificial (IA), en el que se van a emplear grandes cantidades de datos (que habrá que saber interpretar y usarlos de manera correcta) para optimizar el proceso, en el que también hace énfasis el artículo de la revista Elsevier [4]. También entra en juego la visualización y obtención de datos a tiempo real de múltiples puntos de nuestro sistema, como el ejemplo del ensayo de tracción mostrado en el artículo [4] mencionado anteriormente, en el que el laboratorio virtual ofrece una visualización 3D en la que se pueden observar también las zonas de esfuerzo de la pieza cambiando de color esas áreas. Otro ejemplo de esto puede ser el del sistema eléctrico, el cuál supone un gran problema en la actualidad, ya que para integrar todas las energías renovables necesarias para descarbonizar la sociedad y garantizar el suministro eléctrico se necesitan las llamadas "redes inteligentes" en las que se tiene que tener información de cada parte del sistema, desde la producción, el transporte y almacenamiento hasta el consumidor final. En este contexto, el uso de laboratorios virtuales en la docencia de control proporciona a los estudiantes las habilidades y conocimientos necesarios para enfrentarse a los desafíos actuales y futuros en el ámbito de la ingeniería y la industria.

6.2.- Limitaciones y posibles mejoras

- La limitación más relevante es el diseño 3D, ya que este es un diseño sencillo que no se asemeja del todo a la realidad, pudiéndose añadir una serie de complementos visuales que pueden mejorar y hacer más realista la simulación. Ejemplo de estos sería realizar los modelos con texturas más realistas y minimalistas. Adicionalmente se podría cambiar el color de ciertos elementos para indicar, por ejemplo, una sobrecarga, que esté encendido o apagado... O también se podría añadir un vector que cambiara de

tamaño dependiendo de la acción de control.

- Además, se podría introducir un bloque "slider" de Simulink que entrará en la función para variar las características del modelo, permitiendo al alumno observar cómo cambia el controlador y las variables del sistema. Estos cambios podrían incluir variaciones en el tamaño, peso o forma de los elementos del modelo. Esta flexibilidad brindaría una comprensión más completa y realista de cómo las modificaciones en el sistema afectan al control y a su dinámica. Al abordar el concepto de sensibilidad, se exploraría cómo pequeñas variaciones en las variables del sistema pueden tener un impacto significativo en la respuesta del controlador. El ajuste de los parámetros del modelo mediante el bloque "slider" también permitiría al alumno analizar la sensibilidad del sistema a diferentes configuraciones y comprender la importancia de calibrar adecuadamente los valores para lograr un control óptimo. Estas funcionalidades ampliadas proporcionarían ejemplos prácticos y variados, permitiendo a los alumnos adquirir una comprensión más profunda de los sistemas de control y su relación con los parámetros del modelo, así como una visión sobre la influencia de cada variable en la respuesta global del sistema, fomentando el análisis detallado en el desarrollo del diseño.

- Otra mejora sería introduciendo perturbaciones o entradas externas ya sean continuas en el tiempo con el bloque "constant" o variaciones temporales con un bloque "repeating sequence" o con un bloque "waveform generator" con los que se pueden introducir una gran cantidad de tipos de perturbaciones , para que en su caso el alumno diseñe un controlador adecuado para el tratamiento de perturbaciones.

- Una mejora interesante podría ser incorporar en los modelos restricciones físicas, por ejemplo en el modelo de *ball and beam* al incorporar una restricción física para la bola en la tabla. Esta restricción, que podría ser la caída de la bola al salirse de la tabla o la inclusión de un tope que limite su movimiento, permitiría simular de manera más realista las limitaciones físicas que se pueden encontrar en los sistemas. Esta implementación proporcionaría a los estudiantes una experiencia de laboratorio virtual más completa y aproximada a la realidad, donde podrían explorar los efectos de las restricciones físicas en el comportamiento y control del sistema.

- Para una mejor aproximación a la realidad, se puede simular de manera discreta en vez de manera continua, ya que en esta opción se está despreciando el tiempo de adquisición de datos de los sensores, que puede llevar al sistema a la inestabilidad, para ello se puede hacer uso del bloque "transport delay", el cual retrasa la señal a la que está unida el tiempo que se haya definido. Con esto el alumno puede llevar a cabo el diseño del controlador discreto y estudiar el sensor escogido según el tiempo de muestreo mínimo para no llevar al sistema a la inestabilidad. Otra mejora para hacer los

modelos más reales es introducir una saturación a la acción de control, con lo que ya no se dispondría de una acción de control ilimitada.

6.3.- Trabajo futuro

En la realización de este proyecto se ha intentado la implementación física del modelo de motor, haciendo uso de un motor DC equipado con un encoder con el que medir la posición. Para ello se ha utilizado un microcontrolador Arduino, pero finalmente no se ha podido llevar a cabo por el fallo en la lectura de pulsos del encoder.

Además se podría llevar a cabo una "función de coste" que orientase a los alumnos para saber si están haciendo un correcto diseño. Esto conllevaría diseñar un programa con el que los alumnos, al introducir el resultado de controlador que ellos creyesen oportuno dependiendo de las especificaciones requeridas, evaluará el rendimiento del controlador dependiendo de la aproximación del diseño a lo requerido.

Por último, para completar la ayuda docente se necesitarían implementar más modelos y en campos más variados, como por ejemplo, modelos de control del nivel en depósitos de agua, con los que se pueden abordar aspectos como la dinámica de llenado y vaciado del depósito, la influencia de las entradas y salidas de agua, así como las acciones de control necesarias para mantener el nivel deseado y añadir perturbaciones como la fugas en el depósito. Algunos ejemplos a parte pueden ser los expuestos en el artículo de la universidad politécnica de Madrid [11] en los que se expone el programa "EducaControLaboV" con ejemplos como "respuesta de sistemas mecánicos clásicos ante señales de entrada típicas", o el ejemplo "robots industriales. Grados de libertad" además de otros ejemplos interesantes de laboratorios virtuales con los que los alumnos podrán diseñar múltiples sistemas de control de gran valor didáctico. Otro ejemplo relevante es el del trabajo de fin de grado de la universidad de Alcalá [12] en el que se hace un estudio de la aplicación de un gemelo virtual en una placa solar fotovoltaica, pudiendo los alumnos tener ejemplo del desarrollo y utilización de estos en la industria actual.

Bibliografía

[1] J. A. Márquez and T. M. Sanguino, "Diseño de laboratorios virtuales y/o remotos. un caso práctico," *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 7, no. 1, pp. 64–72, 2010.

[2] J. M. M. Jordá, "Herramientas virtuales: laboratorios virtuales para ciencias experimentales–una experiencia con la herramienta vcl," in *Recuperado de: http://aprendeenlinea. udea. edu. co/lms/moodle/mod/forum/discuss. php*, 2012.

[3] G. A. Franky, "Laboratorios reales versus laboratorios virtuales, en la enseñanza de la física," *El hombre y la Máquina*, no. 33, pp. 82–95, 2009.

[4] J. Grodotzki, T. R. Ortelt, and A. E. Tekkaya, "Remote and virtual labs for engineering education 4.0: achievements of the elli project at the tu dortmund university," *Procedia manufacturing*, vol. 26, pp. 1349–1360, 2018.

[5] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital twin: Enabling technologies, challenges and open research," *IEEE access*, vol. 8, pp. 108952–108971, 2020.

[6] "Mathworks webpage." `https://es.mathworks.com/?s_tid=gn_logo`. Accedido 20-03-2023.

[7] "Página web de control tutorials for matlab and simulink." `https://ctms.engin.umich.edu/CTMS/index.php?example=BallBeam&section=ControlStateSpace`. Accedido 20-03-2023.

[8] M. F. Rahmat, H. Wahid, and N. A. Wahab, "Application of intelligent controller in a ball and beam control system," *International journal on smart sensing and intelligent systems*, vol. 3, no. 1, pp. 45–60, 2010.

[9] J. D. P. G. F. Franklin and A. Emami-Naeini, *Feedback Control of Dynamic Systems 5th eedition*. International series of monographs on physics, 2006.

[10] "Web 3d." `https://doc.x3dom.org/tutorials/index.html`. Accedido 02-06-2023.

[11] F. Cerezo and F. Sastrón, "Laboratorios virtuales y docencia de la automática en la formación tecnológica de base de alumnos preuniversitarios," *Revista Iberoamericana de Automática e Informática industrial*, vol. 12, no. 4, pp. 419–431, 2015.

[12] G. Arribas Fernández *et al.*, "Gemelo digital de una planta solar fotovoltaica," 2022.