Universidad Carlos III de Madrid
Escuela Politécnica Superior
Área de Ingeniería Mecánica

Proyecto fin de carrera
Ingeniería Técnica Industrial (Esp. Mecánica)

CREACIÓN DE UNA LIBRERÍA DE ELEMENTOS ROTACIONALES MEDIANTE ECOSIMPRO

Autor:
Juan Alberto González Martín

Tutores:
Dra. Ester Olmeda Santamaría
Dra. Mª Jesús López Boada

CAPÍTULO 1

INTRODUCCIÓN

1.1 Introducción

Los problemas con los que se enfrenta la industria, el comercio o la sociedad en general aumentan en tamaño y complejidad día a día. El tamaño viene determinado por la gran cantidad de alternativas de diseño que se pretenden evaluar y comparar. La complejidad, sin embargo, está relacionada con el nivel de detalle que se quiere incluir en el análisis, la incertidumbre de los datos y el número de criterios que se utilizan en la comparación de alternativas.

Para el planteamiento y la resolución de estos problemas, es imprescindible la implementación de nuevas metodologías y herramientas cada vez más eficaces y eficientes, de manera que sea posible plantear y evaluar múltiples alternativas y las decisiones se puedan tomar en un corto plazo de tiempo. Afortunadamente, parte de la complejidad de este proceso de análisis se ha visto paliada, en los últimos tiempos, por la mejora tanto de las técnicas de resolución como de las innovaciones en el campo de la informática. Estas mejoras no van enfocadas a la experimentación con lo que se denomina sistema real, sino con un modelo, es decir, con una representación fiable de las características y propiedades de dicho sistema, que permite reproducir el comportamiento del mismo.

Una vez implementado el modelo, el siguiente paso que permite conocer el comportamiento del sistema, es la experimentación. La experimentación es un proceso en el cual se excitan las entradas de un sistema, con el objetivo de recopilar determinados datos de salida. Estos datos permiten conocer el comportamiento de dicho sistema frente a las condiciones dadas. Cuando la experimentación se realiza sobre un modelo, recibe el nombre de simulación.

Es importante tener presente las diferencias semánticas existentes entre modelo y simulación. También es importante señalar que estas definiciones no implican necesariamente que el modelo sea codificado en un ordenador, ni que la simulación se lleve a cabo mediante una computadora, ya que también pueden implementarse modelos físicos que sean simulados también en un entorno físico (ejemplos de esto son el túnel de viento o una maqueta de madera de un barco), aunque las únicas simulaciones que se tratarán en las siguientes líneas, serán aquellas que reciben el nombre de simulaciones informáticas.

Una simulación informática consiste en la implementación de un programa informático, cuyo fin es crear una simulación de un modelo de un determinado sistema. Las simulaciones por computadora se basan en modelos matemáticos, es decir, cualquier

formulismo matemático por el cual se establecen las relaciones entre variables, que sirve para estudiar comportamientos de sistemas complejos

Los ordenadores utilizan técnicas para imitar, o simular, el comportamiento de sistemas del mundo real. Para estudiar científicamente estos sistemas, a menudo se han de hacer una serie de suposiciones acerca de cómo trabaja éste. Estas suposiciones que usualmente toman la forma de relaciones matemáticas lógicas, constituyen un modelo que va a ser usado para intentar comprender el comportamiento del sistema correspondiente.

A medida que los ordenadores personales han ido mejorando sus prestaciones, el análisis de sistemas mediante simulación ha ido cobrando cada vez mayor importancia, ya que las posibilidades de representación de un sistema complejo de una herramienta como la simulación y la capacidad de cálculo de los ordenadores han permitido desarrollar un campo amplio y complicado como es la toma de decisiones en entornos multicriterio.

El tiempo de programación y de ejecución de un modelo muy realista disminuye cada vez más, lo que permite realizar una experimentación en la que es posible analizar y comparar muchas alternativas. La solución a implantar, decidida gracias a las técnicas de simulación por ordenador, habrá sido el resultado de realizar cierto número de experimentos, en un tiempo asumible, hasta dar con aquella que mejor satisfaga las necesidades requeridas.

Es evidente que hay límites en la habilidad humana para comprender la complejidad, pero gracias al modelado y a la simulación, es posible simplificar el problema que se está estudiando, creando una ilusión de simplicidad. De este modo, el modelador es capaz de centrarse cada vez en una parte del problema, reduciendo la complejidad del mismo y potenciando su capacidad de análisis. Por tanto la simulación se ha convertido en una potente herramienta que permite al modelador afrontar problemas cada vez más complicados, y resolverlos de una manera eficaz y eficiente.

1.2 Objetivos

La técnica de la simulación se usa constantemente dentro del ámbito de la ingeniería. Se simulan toda clase de sistemas, ya sean sistemas mecánicos, eléctricos, frigoríficos, de control, etc. El objetivo principal de este proyecto consiste en desarrollar una librería rotacional haciendo uso de una herramienta de modelado y simulación llamada EcosimPro. EcosimPro, es una herramienta de simulación con la que se pueden modelar desde los más simples a los más complejos procesos físicos, en términos de ecuaciones diferenciales algebraicas ó ecuaciones diferenciales ordinarias y eventos discretos. Dentro de los diferentes campos donde se puede aplicar la simulación, en el que se va a centrar el presente proyecto es en el campo de la simulación de sistemas dinámicos, más en concreto en sistemas mecánicos de dinámica rotacional.

Una de las características más importantes de las herramientas de simulación es la posibilidad de crear modelos de elementos independientes y poder agrupar varios de

estos modelos con propiedades comunes en conjuntos. Esto se conoce como crear una librería. Este será el objetivo principal del proyecto: la creación de una librería de elementos mecánicos rotacionales, a partir de la cual poder implementar distintos sistemas, que posteriormente serán simulados.

Además, los modelos que se implementarán mediante EcosimPro serán verificados, ya sea comparando los resultados obtenidos con los obtenidos mediante otras herramientas de simulación, o bien a través de expresiones analíticas equivalentes.

1.3 Estructura

El presente Proyecto de Fin de Carrera consta de seis capítulos, siendo el primero de ellos la presente introducción.

En el capítulo 2 se desarrolla un Estado del Arte acerca de los procesos de modelado y simulación de sistemas reales, además se analizan con profundidad todos los elementos y conceptos que en ellos intervienen. Al final de este capítulo se exponen las principales propiedades del Modelado Orientado a Objetos (MOO), que es la metodología empleada por EcosimPro para generar modelos.

En el capítulo 3 versa acerca de EcosimPro. Se definen las propiedades más relevantes en las que se basa y se exponen los conceptos fundamentas que definen este software, para que el lector pueda seguir el proyecto con la mayor fluidez posible y pueda tener una clara visión de lo que se pretende con la elaboración del mismo.

En el capítulo 4 se presenta la librería *Rotational Library* implementada mediante EcosimPro. Tras las primeras líneas, es las que se define la estructura de la librería, se analizan en detalle todos los aspectos de los componentes que la componen.

En el capítulo 5 se modelan y simulan varios sistemas dinámicos con el objetivo de validar los modelos implementados en la librería *Rotational Library.*

CAPÍTULO 2

ESTADO DEL ARTE

2.1. Modelado y simulación

2.1.1. Introducción

El modelado y la simulación son práctica común a todas las disciplinas de la ingeniería y de la ciencia. Son usados en el análisis de los sistemas, a fin de profundizar en su comprensión al permitir el estudio de su comportamiento aislando determinados efectos y eliminando o introduciendo perturbaciones. También se emplean en el diseño de nuevos sistemas con el fin de realizar predicciones del comportamiento de los mismos antes de que sean construidos.

Los orígenes del modelado y la simulación están en la teoría de muestreo estadístico y análisis de sistemas físicos probabilísticos complejos. El aspecto común de ambos es el uso de números y muestras aleatorias para aproximar soluciones.

Las primeras referencias sobre simulación se encuentran hacia el año 1940, cuando Von Neumann y Ullman trabajaron sobre la simulación del flujo de neutrones para la construcción de la bomba atómica en el proyecto "Montecarlo" [ROB04]. Desde entonces se conocían las técnicas de simulación como procesos Montecarlo, aunque en la actualidad se diferencian ambas cosas, siendo los segundos un tipo particular de simulación. También se realizó un proceso de simulación para el proyecto APOLLO dentro del plan espacial de la NASA, acerca del movimiento dentro de la atmósfera de la luna [LAW91].

Actualmente, la simulación es una poderosa técnica para la resolución de problemas por haberse producido un aumento muy significativo en el detalle y la precisión de los modelos de los procesos, en la velocidad y capacidad de memoria de los ordenadores y en la calidad del software de modelado y simulación. Los modelos matemáticos y la simulación han demostrado ser útiles tanto en la fase de investigación y desarrollo, como a la hora de realizar el diseño.

Si las relaciones que componen el modelo son suficientemente simples, es posible usar métodos matemáticos tales como el álgebra, el cálculo o la teoría de la probabilidad para obtener una información exacta de las cuestiones de interés, a esto se le llama solución analítica. Sin embargo, la mayoría de los sistemas del mundo real son demasiado complejos y normalmente los modelos realistas de los mismos no pueden evaluarse únicamente mediante el uso de técnicas analíticas. Lo que se suele hacer en estos casos es incluir la simulación en el estudio de dichos modelos. En una simulación se utiliza el ordenador para experimentar con un modelo numéricamente, de forma que con los resultados obtenidos se haga una estimación de las características del sistema.

2.1.2. Sistemas y experimentos

La simulación permite representar y analizar un sistema real mediante la implementación de un modelo y su posterior ejecución. No obstante, antes de poder comprender cómo simular un sistema y las características principales de la simulación, es necesario introducir y definir conceptos como sistema y experimento, ya que serán términos que aparecerán de manera recurrente durante prácticamente la totalidad del presente documento.

Un sistema puede definirse como un conjunto de objetos o entidades que interactúan entre sí para alcanzar un cierto objetivo. Por ejemplo, se podría considerar un supermercado como un sistema en el que se quiere estudiar el número de cajeros/as necesarios para ofrecer un buen servicio a sus clientes. En este ejemplo, los objetos del sistema podrían ser los clientes en espera de ser atendidos y los cajeros/as que realizan esta tarea. Nótese que los objetos de un sistema pueden variar en función de los objetivos del estudio. Considerando el ejemplo anterior, si lo que se quisiera estudiar fuese la atención a las necesidades de consumo de los clientes, el sistema debería tener en cuenta entidades como los productos disponibles en stock o la lista personal de la compra, entre otros. Así, se puede decir que el *estado de un sistema*, es el conjunto mínimo de variables necesarias para caracterizar o describir todos aquellos aspectos de interés del sistema en un cierto instante de tiempo. A estas variables las denominaremos *variables de estado.* De este modo, en el ejemplo anterior las variables de estado podrían ser cada uno de los cajeros y los clientes esperando a ser atendidos.

En definitiva, un sistema se caracteriza por el hecho de que se puede decir qué pertenece a él y qué no, y por el hecho de que se puede describir cómo interactúa con el entorno. Además, si se coge una parte de éste, se tiene un nuevo sistema. Otra característica de los sistemas es que pueden ser controlados y observados.
Las interacciones de un sistema con el entorno se pueden clasificar en dos categorías:

- Aquellas que son generadas por el entorno y tiene influencia sobre el comportamiento del sistema. Reciben el nombre de *"entradas"*.
- Aquellas que son generadas por el sistema y a su vez influyen en el comportamiento del entorno, *"salidas"*.

En general debe ser posible asignar valores a algunas de las entradas de un sistema y observar su comportamiento mediante el análisis de los datos de salida. Por lo tanto para definir un sistema de forma concisa y unívocamente, se puede definir sistema como toda fuente potencial de datos.

Además, esta última definición de sistema conduce directamente a la definición de *experimento*. Así, se habla de *experimento* para definir todo proceso que consiste en extraer datos de un sistema mediante la excitación del mismo a través de sus *entradas*. Es decir, realizar un *experimento* sobre un sistema significa aplicar un conjunto de condiciones externas a las *entradas* accesibles y observar las reacciones del mismo recogiendo los datos de las *salidas* accesibles.

Pero uno de los mayores inconvenientes de experimentar con sistemas reales radica en el hecho de que generalmente estos sistemas se encuentran sometidos a la influencia de

un gran número de variables de entrada inaccesibles, así como un gran número de datos de salida también inaccesibles. Es por este motivo que surge la *simulación,* cuya principal motivación consiste en que en el mundo de la simulación, todas las entradas y salidas del sistema objeto de estudio son accesibles. Esto permite realizar simulaciones fuera de los límites aplicables a los sistemas reales.

2.1.3. Modelos

En la vida cotidiana se emplean continuamente modelos para comprender y predecir el comportamiento de sistemas (figura 2.1). Por ejemplo, considerar que alguien es "amable" constituye un modelo de esta persona. Este modelo ayuda a responder a preguntas como: "¿cómo reaccionará si se le pide un favor?". Los *modelos mentales* que pueden ser construidos como resultado de la percepción, interacción social o experiencia interna permiten a los individuos hacer inferencias y predicciones, entender los fenómenos, decidir las acciones a tomar y controlar su ejecución[GRE98].

Por ejemplo, aprender a conducir un coche consiste parcialmente en desarrollar un *modelo mental* de las propiedades de la conducción del coche. Asimismo, un operario trabajando en determinado proceso industrial sabe cómo el proceso reacciona ante diferentes acciones: el operario, mediante el entrenamiento y la experiencia, ha desarrollado un *modelo mental* del proceso.

Inevitablemente los *modelos mentales* presentan un marcado carácter subjetivo, la figura 2.1 representa un claro ejemplo de la subjetividad de los *modelos mentales*.

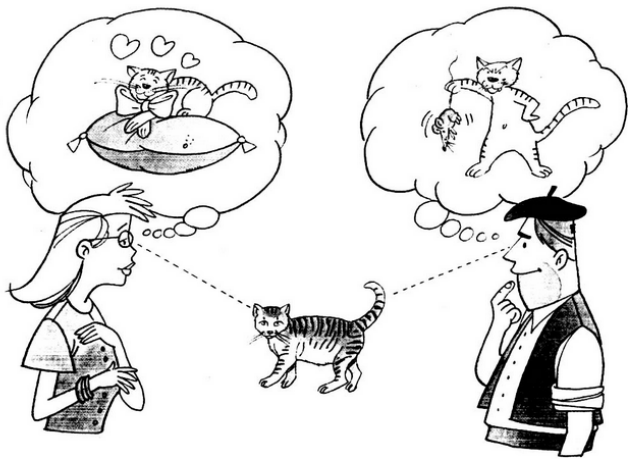


Fig. 2.1: Subjetividad presente en los *modelos mentales*.

Otro tipo de modelos son los *modelos verbales*, en los cuales el comportamiento del sistema es descrito mediante palabras: si se aprieta el freno, entonces la velocidad del coche se reduce.
Es importante diferenciar entre los *modelos mentales* y los *verbales*. Por ejemplo, se usa un modelo mental de la dinámica de la bicicleta cuando se conduce, sin embargo, no es sencillo convertirlo a un modelo verbal.

Además de los modelos mentales y verbales, existe otro tipo de modelos que tratan de imitar al sistema real. Son los *modelos físicos*, como las maquetas a escala que construyen los arquitectos, diseñadores de barcos o aeronaves para comprobar las propiedades estéticas, aerodinámicas, etc.
Finalmente, existe un cuarto tipo de modelos, los *modelos matemáticos*. En ellos, las relaciones entre las cantidades que pueden ser observadas del sistema (distancias, velocidades, flujos, etc.) están descritas mediante relaciones matemáticas. En este sentido, la mayoría de las leyes de la naturaleza son *modelos matemáticos*. Por ejemplo, para el sistema "masa puntual", la Ley de Newton del movimiento describe la relación entre la fuerza y la aceleración. Asimismo, para el sistema "resistencia eléctrica", la Ley de Ohm describe la relación entre la caída de tensión y el flujo de corriente. En algunos casos, las relaciones matemáticas que constituyen los modelos son sencillas y pueden resolverse analíticamente. Sin embargo, en la mayoría de los casos, los modelos no pueden resolverse exclusivamente mediante el uso de técnicas analíticas y deben estudiarse con ayuda del ordenador, aplicando métodos numéricos. Este experimento numérico realizado sobre el modelo matemático, recibe el nombre de *simulación,* concepto que se estudiará en detalle en la siguiente sección de este capítulo.
En la figura 2.2 se muestra la clasificación tradicional de modelos.

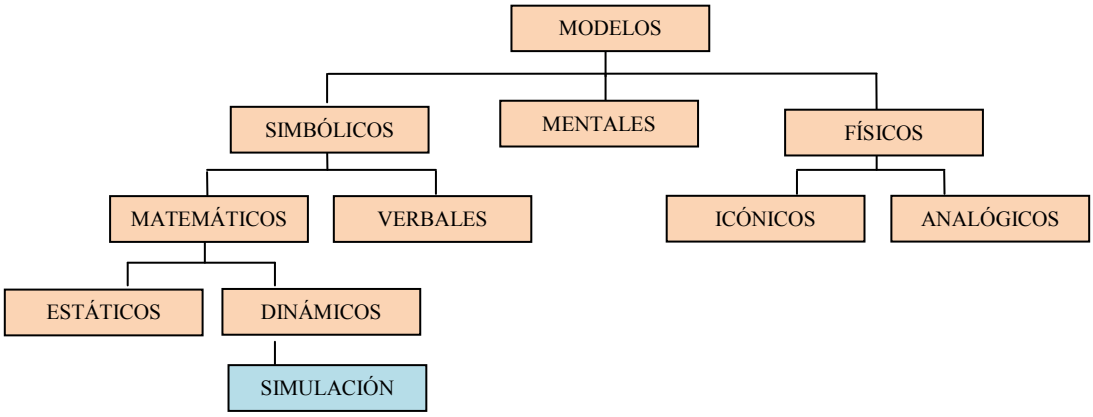


Fig. 2.2: Clasificación tradicional de modelos que representan la realidad.

2.1.4. Modelos matemáticos

Una de las herramientas para desarrollar modelos matemáticos que ha recibido mayor atención en los últimos treinta años es la simulación de sistemas por ordenador. Su auge en los últimos tiempos se debe a que se dispone de mejores prestaciones informáticas, en especial, en lo referente a la eficiencia, permitiendo el análisis de sistemas más complejos en un tiempo razonable, aún con la inclusión en el estudio de comportamientos aleatorios y de lógica a la representación matemática.

Por tanto, el concepto de modelo matemático de un sistema, se asocia normalmente al planteamiento de expresiones matemáticas basándose en las leyes de la física que gobiernan la dinámica de dicho sistema. Es por esto que la mayoría de teorías y herramientas de control que se han desarrollado se enfocan hacia este tipo de sistemas y, más en concreto, hacia sistemas cuya evolución se puede describir por funciones que

evolucionan de forma lineal o no-lineal *(sistemas continuos)*. Por otro lado, en otras aplicaciones los sistemas han de ser controlados por elementos discretos, como interruptores on/off, selectores de velocidad, válvulas, etc…*(sistemas discretos)* en los que la evolución del sistema depende de reglas tipo *"if_then_else"*.
Por último, se pueden encontrar también sistemas jerarquizados constituidos por componentes dinámicos continuos en un nivel inferior, gobernados a nivel superior mediante componentes discretos. Estos sistemas se conocen con el nombre de *sistemas híbridos*.

Se pueden establecer varios criterios a la hora de clasificar los diferentes tipos de modelos matemáticos. Sin embargo, el criterio de clasificación más útil, al menos para los fines que persigue este proyecto fin de carrera, consiste en clasificar los modelos en función de la manera en la que evolucionan las variables con el tiempo, así se tienen:

- **Modelos de tiempo continuo:** son sistemas continuos aquellos sistemas cuyas variables evolucionan continuamente en el tiempo. Los sistemas continuos se describen típicamente mediante ecuaciones diferenciales, ya sea ordinarias o en derivadas parciales.(Fig. 2.3)

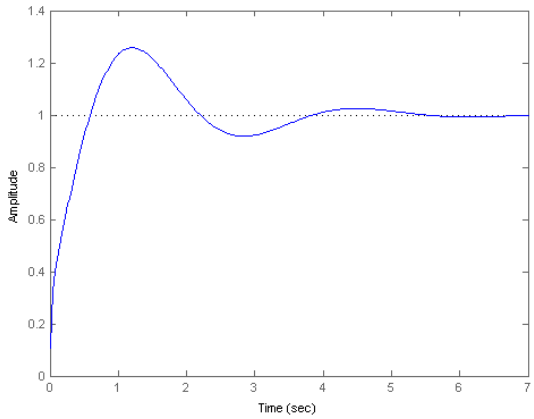


Fig. 2.3: Evolución de variable de un modelo matemático de tiempo continuo.

De esta forma, los modelos de este tipo están representados por conjuntos de ecuaciones diferenciales. Dentro de los modelos de tiempo continuo, se pueden distinguir dos categorías distintas:

I. *Modelos de parámetros concentrados*. Están descritos por ecuaciones diferenciales ordinarias (EDO u ODE, según acrónimo del castellano o del inglés), en general de la forma:

$$x' = f(x, u, t) \tag{2.1}$$

y con el caso especial para sistemas lineales de

$$x' = Ax + Bu \tag{2.2}$$

o también mediante ecuaciones diferenciales algebraicas (EDA o DAE)

$$f(x, x', u, t) = 0 \quad (2.3)$$
$$g(x, u, t) = 0 \quad (2.4)$$

II. *Modelos de parámetros distribuidos*. Sistemas formulados mediante ecuaciones diferenciales en derivadas parciales (PDEs, o en general, PDAEs).

$$\frac{\partial u}{\partial t} = \sigma \cdot \frac{\partial^2 u}{\partial x^2} \quad (2.5)$$

- **Modelos de tiempo discreto**. La Figura 2.4 muestra la trayectoria exhibida por un modelo de tiempo discreto.

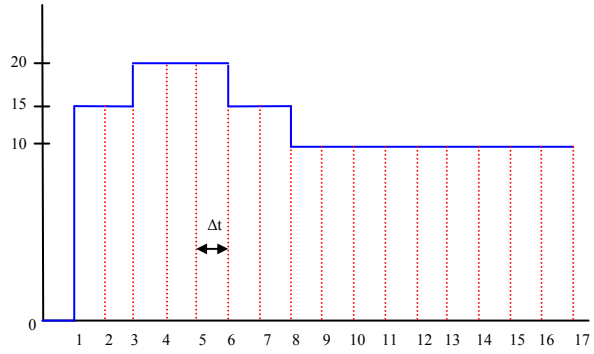


Fig. 2.4: Evolución de variable de un modelo matemático de tiempo discreto.

Los sistemas de tiempo discreto son aquellos cuyas magnitudes sólo pueden tomar un número finito de valores los cuales son funciones de la variable discreta tiempo.
En estos modelos, por tanto, el eje temporal se discretiza. Se presenta normalmente por ecuaciones diferenciales (EDif), al menos cuando la discretización está espaciada de forma equidistante, tales como la siguiente:

$$x_{k+1} = f(x_k, u_k, t_k) \quad (2.6)$$

Los modelos de tiempo continuo se utilizan frecuentemente en ingeniería, sobre todo en sistemas controlados por ordenador. Si se utiliza un ordenador en un lazo de control, este último no puede llevarse a cabo de forma continua, ya que la generación de la señal de control necesita un tiempo determinado. Es por este motivo por el que se discretiza el eje temporal, normalmente en intervalos equidistantes. Si este método se utiliza para controlar un sistema que es por sí mismo continuo (como ocurrirá normalmente) se habla de *sistemas de control muestrados*.
Los modelos de tiempo discreto pueden ser también versiones discretizadas de modelos contínuos. Por ejemplo, si se discretiza el eje temporal de un

modelo en el espacio de variables de estado de la ecuación (2.1) con un intervalo de discretización Δt , la derivada del estado será:

$$\frac{x_{k+1}-x_k}{t} \approx f(x_k, u_k, t_k) \tag{2.7}$$

o también

$$x_{k+1} = x_k + \Delta t \times f(x_k, u_k, t_k) \tag{2.8}$$

Lo que da lugar a un modelo discreto. De hecho, cuando se utiliza un ordenador para simular un modelo continuo, realmente lo que el ordenador hace, es discretizar el eje de los tiempos para evitar el problema de los infinitos cambios antes comentados.

- **Modelos cualitativos:** son, en realidad, modelos en tiempo discreto, aunque la discretización del eje temporal no tiene que ser necesariamente equidistante. La figura 2.5 muestra la trayectoria seguida por un modelo de esta categoría.

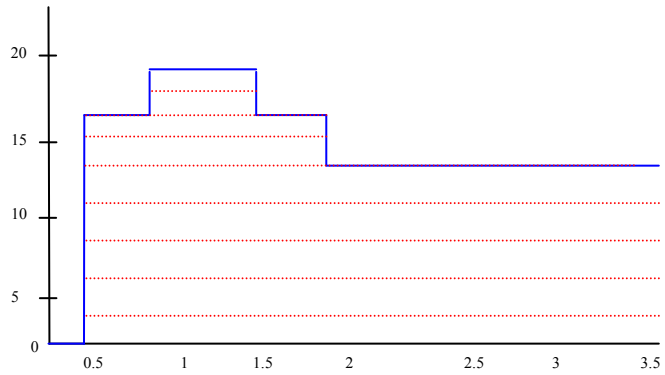


Fig. 2.5: respuesta de un modelo cualitativo.

- **Modelos de sucesos o de eventos discretos:** en éstos, tanto el eje temporal como el eje de estado son continuos, pero difieren de los modelos en tiempo continuo en qué sólo puede ocurrir un número finito de cambios. La figura 2.6 muestra la trayectoria de este tipo de modelos.

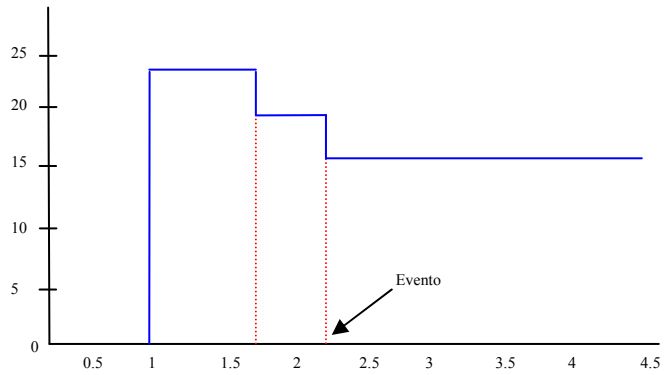


Fig. 2.6: Respuesta de un modelo de eventos discretos

- **Modelos híbridos:** son aquellos sistemas dinámicos que incluyen variables de estado continuas y discretas. Además, estos dos aspectos del comportamiento del sistema interactúan de manera tan importante que no pueden ser desacoplados, debiendo ser analizados a la vez. Los subsistemas continuo-discreto, sólo interactúan entre si en puntos inequívocos conocidos como *eventos*, además cuando ocurre un evento, los subsistemas discretos pueden modificar el estado de los subsistemas continuos de dos formas:

  - Alterando la estructura de las ecuaciones diferenciales que describen la evolución del estado continuo.
  - Provocando saltos en el estado continuo, dando lugar a discontinuidades.

  Por tanto, cuando ocurre un evento, las ecuaciones diferenciales que definen estos sistemas híbridos sufren alteraciones, que pueden ser tan sencillas como eliminar una ecuación y sustituirla por otra, o tan complejas como la eliminación y la inserción de incógnitas descritas por sus propios sistemas de ecuaciones diferenciales. Es por ello que existen múltiple técnicas y metodologías para modelar sistemas híbridos que, fundamentalmente, se basan en la discretización de los subsistemas continuos.

Pero, además de clasificar los modelos en función del tipo de variables, también se pueden encontrar otras clasificaciones de modelos matemáticos atendiendo a diferentes criterios, entre ellos:

- Según la aleatoriedad se pueden distinguir dos tipos de modelos: **los modelos deterministas y los estocásticos**. Un modelo determinista es aquel en el que se conoce de manera puntual la forma del resultado ya que no hay incertidumbre. Además, los datos utilizados para alimentar el modelo son completamente conocidos y determinados. Por otro lado, un modelo estocástico es aquel en el que no se conoce el resultado, sino su probabilidad y existe por tanto incertidumbre.

  Un buen ejemplo de modelo estocástico lo representan las simulaciones de tráfico, así, si se considera un modelo de parking, en el cual las entradas y salidas de coches se producen en instantes de tiempo aleatorios, la aleatoriedad de estas variables se propaga a través de la lógica del modelo, de modo que las variables dependientes de ellas también son aleatorias. Este sería el caso, por ejemplo, del tiempo que transcurre entre que un cliente deja aparcado su vehículo y lo recoge (tiempo de aparcamiento), el número de vehículos que hay aparcados en un determinado instante, etc. Es importante tener en cuenta que realizar una única réplica de una simulación estocástica es equivalente a realizar un experimento físico aleatorio una única vez.
  Por ejemplo, si se realiza una simulación del comportamiento del parking durante 24 horas, es equivalente a observar el funcionamiento del parking real durante 24 horas. Si se repite la observación al día siguiente, seguramente los resultados obtenidos serán diferentes, y lo mismo sucede con la simulación: si se realiza una segunda réplica independiente de la primera, seguramente los resultados serán diferentes. La consecuencia que debe extraerse de ello es que el diseño y el análisis de los experimentos de simulación estocásticos debe hacerse

teniendo en cuenta esta incertidumbre en los resultados, es decir, debe hacerse empleando técnicas estadísticas.

- **Modelos dinámicos y estáticos**. Si en el modelo existe una ligadura directa e instantánea entre las variables, se dice que es estático. Por el contrario, los sistemas cuyas variables pueden cambiar sin necesidad de que exista una influencia directa externa se denominan dinámicos.

En definitiva, un modelo es un objeto o concepto que se utiliza para representar cualquier otra entidad compleja mediante un proceso de abstracción y simplificación de dicha entidad, resultado del estudio y conocimiento de las leyes físicas y características que la definen. Dicho de otro modo, un modelo debe ser una representación simplificada del sistema a estudiar, que permita de una forma relativamente sencilla explicar, comprender, cambiar, prever y posiblemente controlar el comportamiento del mismo.

2.1.5. Simulación

La simulación, junto con la experimentación del sistema real, es el único proceso por el cual se obtiene el conocimiento del comportamiento de un sistema a partir de la observación de la conducta de un modelo que lo representa.

Por su parte, las técnicas analíticas de resolución son útiles, pero generalmente requieren de una serie de simplificaciones que no siempre pueden ser justificadas, pero incluso en el caso de que sí lo puedan ser, dicha justificación no puede ser verificada si no se experimenta o se simula el sistema. En otras palabras, la simulación es una potente herramienta, pero que se suele emplear junto con otras técnicas analíticas o semianalíticas.

El escenario típico de aplicación de técnicas analíticas junto con simulación podría ser el siguiente:

- Se realizan experimentos sobre el sistema real para extraes datos (adquirir conocimientos).
- Después se analizan los datos y se postulan una serie de hipótesis como consecuencia dicho análisis.
- Se realizan una serie de supuestos con el objetivo de simplificar los datos, para que así, puedan ser tratados con técnicas analíticas, con el fin de testar las hipótesis postuladas con anterioridad.
- Se realizan algunas simulaciones con distintos parámetros experimentales para verificar que las simplificaciones realizadas sobre el modelo están justificadas.
- Se repiten los cálculos analíticos, verificando (ó modificando) las hipótesis, y finalmente se extraen ciertas conclusiones.
- Por último se realizan las últimas simulaciones para verificar las conclusiones.

Como puede deducirse de la secuencia anterior, la simulación es aplicable allí donde las técnicas analíticas no lo son. Aunque hay que señalar que a pesar de que el dominio de aplicación de las técnicas analíticas es más reducido, éstas son más poderosas. La explicación a esta afirmación se debe a que mientras que las técnicas analíticas dan idea de cómo se comporta un sistema bajo unas condiciones experimentales arbitrarias, la

ejecución de una simulación sólo da idea de cómo se comporta el sistema bajo el conjunto de condiciones experimentales aplicadas a éste durante el transcurso de la simulación.

A pesar de esta ventaja de las soluciones analíticas frente a la simulación, existen otras muchas razones por las que resulta interesante simular sistemas:

- *El sistema físico no está disponible*: Hay ocasiones en las que el sistema físico no está disponible y la simulación es la única manera de validar algo.
- *El experimento puede ser peligroso*: cuando el experimento puede ser peligroso la simulación cobra protagonismo reduciendo así riesgos de pruebas. Un ejemplo de ello puede ser el empleo de simuladores de vuelo con pilotos inexpertos.
- *El coste del experimento es elevado*: Si el coste del experimento es muy alto, se pueden reducir considerablemente los costes haciendo uso de la simulación.
- *El tiempo del experimento no es compatible*: Si las constantes de tiempo del sistema son incompatibles con las del experimentador. Un ejemplo de ello puede ser el estudio del movimiento de las galaxias.
- *Las variables o los parámetros del sistema no son accesibles*: A veces las simulaciones se hacen imprescindibles ya que determinadas variables o parámetros del sistema estarían fuera del control de experimentador. Por ejemplo, si se decide cambiar la masa de un elemento del sistema de 40 kg a 400 kg, en un sistema real, una alteración de este tipo provocaría un incremento del coste y de tiempo considerables, sin embargo, haciendo uso de la simulación bastaría una pequeña modificación en el programa para poder repetir el experimento.
- *Eliminar perturbaciones*: En ocasiones las simulaciones se realizan porque permiten eliminar perturbaciones que son inevitables en el sistema real. Esto permite aislar determinados efectos, que conducen a una mejor comprensión del comportamiento del sistema.
- *Eliminación de efectos de segundo orden*: La simulación permite eliminar los efectos de segundo orden, como la no-linealidad de los elementos de un sistema. La idea que hay detrás de esta aproximación es que no todos los factores son igualmente importantes para determinar el comportamiento del sistema. Se trata de determinar qué factores son críticos (efectos de primer orden) y cuáles no (efectos de segundo orden). Omitir los detalles supone introducir deliberadamente *"imperfecciones"*, y por tanto podría considerarse que es una mala práctica. Sin embargo, nada más lejos de la realidad: solamente omitiendo los factores de segundo orden puede construirse un modelo útil [URQ01].

Por tanto, gracias a la simulación se pueden reducir costes, acortar tiempo de desarrollo de productos y minimizar materiales y riesgos en la realización de pruebas. Hacer varios diseños de un mismo producto y luego elegir la mejor variante ya no es tan complicado como lo podía ser en el pasado, incluso es posible realizar infinidad de pruebas virtuales para verificar la validez de estos diseños antes de fabricar el producto en sí. Debido a la gran versatilidad y a la elevada potencia que poseen estas herramientas de simulación es relativamente sencillo poder modelar sistemas con altas precisiones. En la figura 2.7 se representan algunas de las muchas ventajas del modelado y la simulación:

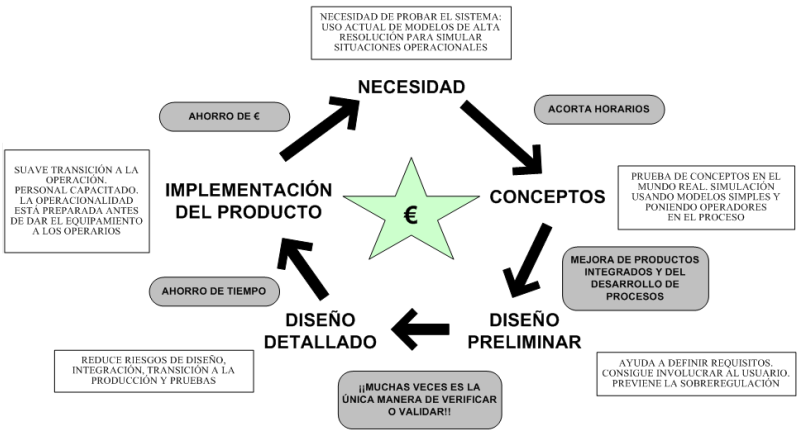


Fig. 2.7: Ventajas del modelado y la simulación.

Pero la simulación también tiene una serie de peligros que es interesante conocer. La facilidad de uso de la simulación es una desventaja muy importante, es sencillo para el usuario olvidar las limitaciones y condiciones bajo las que una simulación es valida, y por lo tanto sacar conclusiones erróneas de la simulación en este caso es muy probable. Para reducir estos peligros, se debería intentar siempre comparar al menos algunos de los resultados de la simulación del modelo con los resultados experimentales medidos del sistema real, esto es lo que se conoce como *validación del modelo*. Para no caer en estos peligros es necesario ser consciente de que el modelo no es el sistema real, este sólo representa al sistema real bajo ciertas condiciones. No hay que forzar a que la realidad encaje dentro de las restricciones del modelo y no hay que olvidar el nivel de precisión del modelo, todos los modelos tienen hipótesis simplificadoras que hay que tener en cuenta.

Es posible simular una gran variedad de sistemas y procesos, ya sean sistemas mecánicas, eléctricos, frigoríficos, de control, aeroespaciales, procesos químicos, geológicos, genéticos, etc. A continuación (Fig. 2.8) se presenta un ejemplo real de modelado y simulación de un sistema electrónico donde se muestran los componentes físicos y el esquema de la simulación:

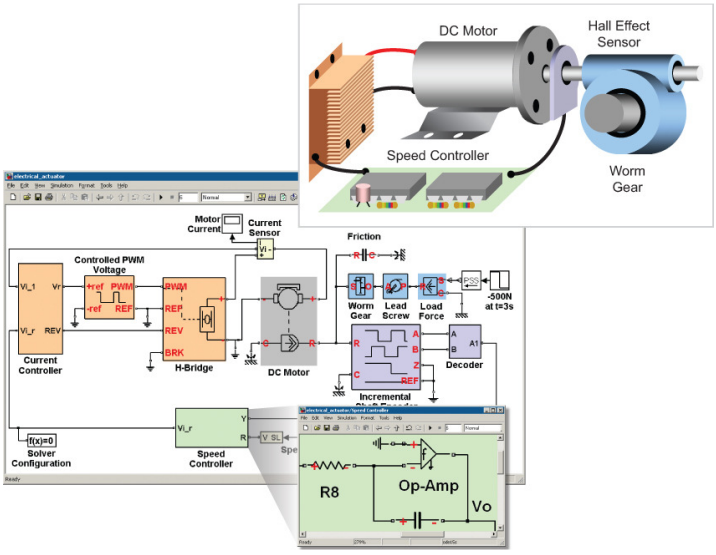


Fig. 2.8: Ejemplo real de modelado.

2.1.6. Lenguajes de programación

Hoy en día al modelador se le abren un amplio abanico de posibilidades para resolver todos los problemas que se le plantean y para programar las operaciones necesarias para realizar la simulación de su modelo. El abanico corresponde a los distintos lenguajes que se pueden utilizar para traducir los modelos en un ordenador y posteriormente resolverlos para obtener la simulación del comportamiento del sistema modelado. Así, se pueden utilizar lenguajes de programación general, lenguajes específicos para simulación o paquetes de software de simulación especialmente preparados para la misma.
A la hora de elegir una herramienta u otra hay que tener en cuenta en primer lugar la velocidad de ejecución de los programas y la utilización de recursos necesaria (memoria, coprocesadores, etc.).

Dentro de los lenguajes de programación existen distintos niveles, en el más bajo nivel se encuentra el lenguaje máquina cuyas instrucciones se escriben en la notación binaria que corresponden directamente con las funciones u operaciones elementales. Este lenguaje es sin duda el más tedioso y menos práctico de utilizar. En un nivel superior se encuentran el lenguaje ensamblador que utiliza símbolos (caracteres) nemónicos[1] para representar dichas funciones.
Los lenguajes de alto nivel o lenguajes de propósito general tales como C, Fortran, Basic, Cobol, Lisp , Algol, Pascal , etc… normalmente alejan al programador de las tareas de bajo nivel del computador y suelen ir apoyados en un conjunto de librerías que en el caso de la simulación facilitan mucho la tarea de modelar sistemas y reducen normalmente el tiempo de ejecución del programa.

Por otro lado, los lenguajes de simulación son similares a los lenguajes de programación de alto nivel, pero están especialmente preparados para determinadas aplicaciones de la simulación. Así, suelen venir acompañados de una metodología de programación apoyada por un sistema de símbolos propios para la descripción del modelo, por ejemplo mediante diagramas de flujo u otras herramientas que simplifican notablemente el modelado y facilitan la posterior depuración del modelo.
Entre estos lenguajes específicos se pueden nombrar los siguientes : MIDAS, DYSAC, DSL , GASP, MIMIC, DYNAMO, GPSS, SIMULA, CSSL(Continuous System Simulation Language) , CSMP, ACSL (Advanced Conrinuous Simulation Language), DARE-P and DARE-Interactive, C-Simscript, SLAM,SIMAN, SIMNON, SIMSCRIPT-II-5, ADA, GASP IV, SDL. Muchos de estos lenguajes dependen fuertemente de los lenguajes de propósito general como es el caso de Slam o Siman que dependen de Fortran para las subrutinas.

[1] es un dato simbólico que identifica a un comando generalmente numérico (binario, octal, hexadecimal) de una forma más sencilla que su numeración original, lo cuál facilita radicalmente la memorización de este comando para el programador.

Otro concepto importante es el de la Simulación Visual Interactiva , que puede definirse como aquella que posibilita la creación gráfica de modelos de simulación, permite mostrar por pantalla dinámicamente el sistema simulado, así como la interacción entre el usuario y el programa en ejecución.
Estos lenguajes permiten construir modelos complejos de manera incremental, a partir de la selección de componentes del sistema de entre un repertorio limitado a la extensión de las librerías que contienen unas entidades predefinidas, si bien las ultimas tendencias añaden a estos paquetes editores para crear nuevas plantillas con características a gusto del consumidor, introduciendo además utilidades de todo tipo incluidas las gestiones de configuración y control de las comunicaciones con un sistema de control real al que se puede conectar el equipo.

Resulta evidente, por tanto, que son muchas las ventajas de programar el modelo de simulación en un lenguaje de simulación en vez de hacerlo en un lenguaje general como FORTRAN, PASCAL, o C entre todas ellas destacan las siguientes:

- Los lenguajes de simulación proporcionan automáticamente las características necesarias para la programación de un modelo de simulación, lo que redunda en una reducción significativa del esfuerzo requerido para programar el modelo.
- Proporcionan un marco de trabajo natural para el uso de modelos de simulación. Los bloques básicos de construcción del lenguaje son mucho más afines a los propósitos de la simulación que los de un lenguaje de tipo general.
- Los modelos de simulación son mucho más fácilmente modificables.
- Proporcionan muchos de ellos una asignación dinámica de memoria durante la ejecución.
- Facilitan una mejor detección de los errores.
- Los paquetes de software especialmente diseñados para simulación contienen aplicaciones diversas que facilitan al simulador las tareas de comunicaciones, la depuración de errores sintácticos y de otro tipo de errores, la generación de escenarios, la manipulación “on-line” de los modelos, etc.

De todas formas, frente a la disyuntiva de tener que elegir uno u otro tipo de programación, lo primero a tener en cuenta es que la programación con un lenguaje de propósito general como C++, apoyado con librerías, permite afrontar problemas de la máxima complejidad y tamaño pudiendo ser el programa altamente portable. El mayor inconveniente de este tipo de programación se presenta al interpretar los mensajes de error del sistema de desarrollo, puesto que se requiere un alto grado de especialización en el desarrollo de software.
En el caso de utilizarse un lenguaje específico de simulación, la limitación está en que no permite desarrollar más allá de para lo que está pensado y diseñado el software, pero como contrapartida está que el usuario sólo precisa disponer de los conocimientos de programación relativos al producto.
Finalmente, los productos de modelización visual permiten realizar prototipos en tiempos récord siempre que los objetos a utilizar coincidan exactamente con los disponibles en el producto. En la medida que se requieran objetos específicos hay que recurrir a la programación.

2.2. Modelado Orientado a Objetos (MOO)

El modelado orientado a objetos es un método de la ingeniería del software, que se basa en analizar un sistema como un conjunto de elementos que interactúan entre si. Este método de diseño se fundamenta en tres conceptos básicos, como son la abstracción, la ocultación de la información y la modularidad.

La *modularidad*, que consiste es descomponer un sistema en sus partes, es muy recomendable a la hora de modelar sistemas complejos. De esta manera se puede modelar cada una de sus partes por separado como sub-modelos. Así, el análisis por reducción consiste en dividir el sistema en partes, suponiendo que las partes individuales serán más fáciles de entender que el conjunto y que, si se entiende la manera en la que actúan cada una de las partes y cómo interactúan entre sí las partes, se puede predecir, y por tanto comprender, el comportamiento del conjunto.

Las fases fundamentales del *modelado modular* son:

1. Definición de la estructura del sistema. Identificación de sus distintas partes.
2. Definición de la interacción entre las partes
3. Definición del comportamiento interno de cada parte independientemente de las demás.

Si se estructura el modelo de forma modular se facilita su diseño y realización, ya que resulta más sencillo desarrollar y validar sub-modelos más reducidos, que modelos de mayor tamaño. Además, la modularidad agiliza el desarrollo de los modelos, al permitir que varios especialistas trabajen de forma independiente en distintas partes del modelo. Es lógico pensar también, que un modelo implementado modularmente sea fácilmente actualizable si algún aspecto físico de la realidad que representa varía o si se modifica alguna de las hipótesis bajo las cuales se está trabajando. Asimismo la modularidad posibilita a reutilización de un mismo modelo en contextos diferentes, lo que origina una reducción de los costes de modelado.

Otro concepto importante dentro de los lenguajes MOO es la *abstracción*. Se llama abstracción a la posibilidad de usar cada parte del modelo sin conocer los detalles internos del mismo. Así, la abstracción es otra propiedad de los MOO que contribuye a afrontar el problema de la complejidad de los sistemas de gran tamaño, al posibilitar que varios especialistas trabajen independientemente en distintas partes del modelo del sistema sin tener que conocer todos los detalles de éste. Una manera de facilitar la abstracción consiste en identificar, en cada parte del modelo, la *interfaz* y la *descripción interna*.

La interfaz es la parte del modelo que describe la interacción del modelo con su entorno, al mismo tiempo que aísla a éste de la descripción interna del modelo, de forma que puedan ser considerados separadamente. La *descripción interna* del modelo contiene la información sobre su estructura y comportamiento.

La abstracción y la modularidad están estrechamente relacionadas mediante el encapsulamiento de la información (también denominada ocultación de la información), que consiste en que sólo las variables pertenecientes a la interfaz sean accesibles desde el exterior del módulo por otros módulos (aunque todas las variables pueden ser

observadas en la simulación de modelo). *La ocultación de información* como criterio de diseño de los sistemas modulares facilita las modificaciones, las pruebas y el mantenimiento de los modelos.

Con el fin de facilitar su diseño y entendimiento, el modelo puede, además de modularmente, implementarse de forma *jerárquica*, es decir, progresando de menor a mayor nivel de detalle en su descripción: el modelo se divide en sub-modelos, que a su vez pueden sud-dividirse en sub-submodelos, y así sucesivamente. Esta descripción de modelo en diferentes niveles de detalle facilita además su mantenimiento y reutilización.

El diseño orientado a objetos también se aplica al modelado de sistemas dinámicos. Los objetivos principales del modelado orientado a objetos para sistemas dinámico son:

- Incrementar la productividad del modelador al facilitar la modularidad y la reutilización del código y al permitir que varios programadores trabajen independientemente en distintas partes del modelo.
- Aumentar la calidad de código.
- Facilitar el refinamiento, modificación y mantenimiento de los modelos.

Del mismo modo que en el modelado modular y en el jerárquico, al realizar el diseño se define la estructura del sistema a modelar y la interacción entre sus partes antes de comenzar con la representación del comportamiento de sus componentes básicos.

Conceptos importantes en el *modelado orientado a objetos* son los de *clase* e *instanciación* de una clase. Una clase es una descripción de un grupo de objetos con propiedades parecidas. Los modelos se representan como clases más que como instancias, ya que un modelo normalmente es una descripción de un tipo de sistema, más que una representación de un sistema en particular. La simulación se realiza sobre una instanciación del modelo. Las dos formar principales de reutilización de las clases son la *composición* y la *especialización*. La *composición* es la capacidad de definir nuevos modelos formados por submodelos ya existentes y definidos, conectados entre sí. La *especialización* es la definición de nuevos modelos especializando otros ya existentes.

La *parametrización* es un concepto clave en la reutilización. Los *parámetros* de un modelo son todas aquellas propiedades que pueden cambiarse para adaptar el modelo a sus diferentes aplicaciones. Un parámetro puede ser, en sentido amplio, desde una simpe variable hasta una estructura completa de submodelos.

La *herencia* es un mecanismo de compartición de información entre clases mediante especialización. Una clase A puede definirse como una *subclase* de la clase B, diciéndose entonces que B es a *superclase* de A. Si una clase sólo puede tener una superclase se dice que existe *herencia simple* y en el caso de que pueda tener más de una superclase se dice que hay herencia múltiple. La subclase se atribuye todas las propiedades definidas en la superclase, con lo que puede considerarse como una especialización del concepto definido por ésta última, a la cual se añaden nuevos componentes o ecuaciones propios de la subclase. Es importante señalar que, en caso de que los *atributos locales* de una clase estén contrapuestos con los heredados de su superclase, prevalecerán en la clase los valores locales. Entendiendo por atributos

locales todas aquellos atributos que pertenecen a una subclase. Esta propiedad supone una forma de parametrización, ya que da lugar a la modificación de alguna de las propiedades para adaptar el modelo a sus diferentes aplicaciones.

El *polimorfismo* es un concepto íntimamente relacionado con la reutilización de los modelos. Son modelos polimórficos aquellos que poseen interfaces con las mismas estructuras y que además tienen el mismo grado de libertad (diferencia entre el número de variables internas al modelo y el número de ecuaciones del modelo). El polimorfismo es la condición de partida para que varios modelos puedan ser usados en el mismo contexto e intercambiados sin que sea necesario modificar el resto del sistema. Normalmente los modelos polimórficos poseen una superclase en común, que además es la que define la interfaz.

Debido a las ventajas que ofrecen todas estas propiedades, es común el uso de la metodología de MOO para modelar componentes y sistemas de componentes. El lenguaje de MOO usa los objetos y sus interacciones para diseñar aplicaciones informáticas y llevar a cabo simulaciones con ellas.

La forma de trabajar con lenguajes MOO es, normalmente, la que se presenta en la figura 2.9. En primer lugar hay que generar los modelos de los componentes (objetos) y puertos (uniones entre objetos) creando para ello sus códigos en el lenguaje de programación. Es conveniente organizar estos componentes en una librería para tener todo mucho más ordenado. Incluso es posible asignar iconos gráficos a estos componentes para tener mayor claridad a la hora de trabajar con ellos (la potencia de un entorno está en el número y en la calidad de sus librerías). Una vez creada la librería con la que se va a trabajar es imprescindible generar las ecuaciones que representan la mecánica del sistema. Creadas estas ecuaciones que definen el modelo se puede ordenar de forma secuencial, definir que incógnitas se van a despejar, decir cómo se resuelven los posibles lazos algebraicos, etc., pero esto es algo que algunos de los programas de modelado y simulación hacen automáticamente. Finalmente hay que ejecutar las simulaciones y los experimentos que se deseen.

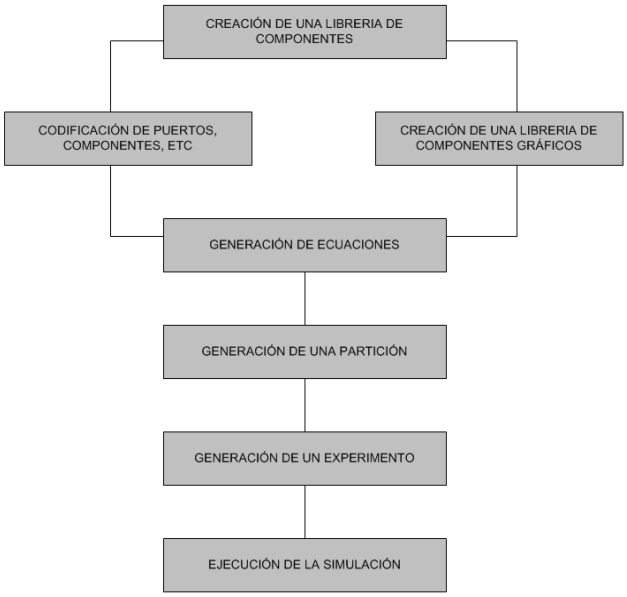


Fig. 2.9: Forma de trabajar en MOO.

Es importante señalar que los lenguajes de modelado orientado a objetos deben ser capaces de modelar los dos mundos: el *continuo* y el *discreto*. Mediante el modelado continuo el modelador expresa el medio físico con unas ecuaciones matemáticas concretas. Estas ecuaciones pueden ser algebraicas o deferenciales con respecto al tiempo. Estas ecuaciones serán procesadas, de forma que el modelo final calcule de manera óptima las variables desconocidas del sistema.

Pero además de permitir el modelado de sistemas continuos, estos lenguajes tienen que dar la posibilidad de implementar modelos discretos. Es habitual que un modelo continuo, en algún momento, tenga un comportamiento discreto. Así cada tipo de comportamiento continuo está asociado a una serie de eventos que se están comprobando continuamente. Cuando ocurre uno de ellos se ejecuta una serie de acciones asociada. Esto significa que los componentes se modelan utilizando ambos métodos de modelado: el continuo y el discreto. Debe ser tarea del entorno ocuparse de que ambas partes del modelo se ejecuten de forma coordinada.

Los lenguajes de MOO permiten tres tipos de sentencias: las *continuas*, las *discretas* y las *secuenciales*, es decir, permiten realizar modelos de todos los tipos explicados con anterioridad. Así, las sentencias *secuenciales* son aquellas que se ejecutan cada una a continuación de la precedente. Las *discretas* deberán estar en su propia sección y se ejecutan ante determinado eventos. Las *continuas* son las que se repiten en cada instante de tiempo y sufren la mayor parte del preprocesamiento, como la ordenación de las ecuaciones (en caso de lenguajes acausales), la detección de lazos, etc.

Por tanto, las ventajas de la metodología de MOO parecen ser claras, y pueden resumirse en los siguientes puntos:

- El modelador encapsula los datos y el comportamiento en componentes individuales.
- Un componente oculta información haciendo púbicas sólo parte de sus características, reduciendo de esta manera la complejidad aparente del sistema.
- Los componentes pueden reutilizarse.
- El formato de una ecuación es declarativo: los algoritmos las transforman para resolverlas de la mejor forma posible.

Todas estas ventajas además de la acausalidad de algunos programas de MOO, han convertido estos lenguajes en una herramienta fundamental, imprescindible para la simulación de cualquier sistema dinámico.

2.2.1. Causalidad en MOO

Dado un conjunto de relaciones matemáticas, la asignación de la causalidad computacional es la decisión de qué relación debe emplearse para calcular cada incógnita. El obstáculo principal en el camino hacia la modularidad y la componibilidad de los modelos matemáticos es la causalidad computacional, debido a que la causalidad computacional de cada parte en que se ha dividido el sistema depende de las demás

partes del mismo, de cómo se realice la conexión entre las partes y de las condiciones de contorno del sistema.

Las condiciones de contorno del sistema están ligadas con su propósito, esto es, con sus "causas" (variables de su interfaz que son manipuladas, son lo que se conoce como *entradas*) y con sus "efectos" (variables de su interfaz que cambian debido a esta manipulación, también conocidas como *salidas*). De este modo, la causalidad computacional no puede decidirse localmente en un subsistema sin tener en cuenta el sistema completo en conjunto, con lo cual, entra en conflicto con el concepto de modularidad.

Algunas de las herramientas comerciales de simulación de propósito general actuales, entre las que se encuentras ACSL y Simulink, imposibilitan el modelado modular al imponer que el modelo contenga explícitamente información acerca de su causalidad computacional. Son *lenguajes de simulación*, en los cuales el código es monolítico respecto a la estructura del proceso físico a modelar y estructurado en lo referente a la utilización de algoritmos numéricos.

Un modo de solucionar este problema consiste en no incluir información acerca de causalidad computacional en la implementación matemática del modelo. Éste es el enfoque que utilizan los programas que emplean *lenguajes de modelado acausales*, como son el Dymola, Modelica, Ecosimpro, etc., en los que se asigna automáticamente la causalidad computacional del modelo completo, partiendo de la descripción matemática de cada una de las partes que lo integran, del modo de conectarlas y de las condiciones de contorno.

Al contrario de lo que ocurre en los lenguajes de simulación con causalidad computacional explícita, en los que escribir una ecuación de la forma *variable=expresión* implica que en esa ecuación debe evaluarse la variable situada al lado izquierdo de la igualdad, en los *lenguajes de modelado acausales* el modo de escribir una ecuación no informa sobre la variable que debe calcularse primero. Los modelos son declarativos, al definir relaciones más que construir procedimientos para el cálculo de datos. Además del algoritmo para realizar automáticamente la asignación de causalidad computacional, estos lenguajes están dotados de potentes manipuladores simbólicos de fórmulas para despejar automáticamente la variable a calcular de cada ecuación, resolver lazos algebraicos lineales, diferenciar simbólicamente, etc.

Un ejemplo de lenguaje de modelado acausal a partir de la difusión térmica en un punto sería el que sigue:

$$T' = {}^{q}/_{mc} \tag{2.9}$$

Donde *T´[ºC/s]* es el gradiente térmico o a derivada de la temperatura con respecto al tiempo, *q[W]* es el flujo de calor en el punto y *mc[J/kg·K]* es la capacidad térmica del punto.

Por no causalidad se entiende que la ecuación 2.9 define una relación entre las tres variables, y el lenguaje de modelado acausal trata la ecuación 2.9 de la misma manera que si el usuario hubiese escrito cualquiera de las siguientes expresiones:

$$mc = {}^{q}/_{T'} \tag{2.10}$$

$$q = mcT' \tag{2.11}$$

Es decir, el usuario no tiene que hacer suposiciones acerca de cuáles son las variables conocidas y cuáles las desconocidas de la simulación. Las conocidas (condiciones de contorno) serán utilizadas por el algoritmo para calcular las desconocidas durante la ejecución de la simulación. Esta clara diferencia entre el modelo y los casos analizados permiten usar un mismo modelo para analizar diferentes casos (*rentabilidad*).

2.2.2. Ventajas del MOO

En resumen, se pueden citar como ventajas de la metodología MOO las características siguientes:

- El modelador puede encapsular los datos y el comportamiento en componentes individuales.
- Supone una simplificación en el diseño y realización del modelo al facilitar la modularidad.
- Un componente oculta la complejidad haciendo públicas sólo parte de sus características.
- Las características públicas comprende parámetros, puertos y datos. Mientras que las variables locales, eventos discretos y ecuaciones son privadas.
- Los componentes pueden reutilizarse.
- Un componente puede contener ecuaciones que pueden insertarse en tiempo de simulación o no, dependiendo de los parámetros que se pasen al componente.
- El formato de una ecuación es declarativo: los algoritmos las transforman para resolverlas de la mejor forma posible.

CAPÍTULO 3

ECOSIMPRO

Hasta ahora se ha hablado de los conceptos de modelado y simulación, y de cómo determinados lenguajes de programación (como los MOO) hacen uso de distintas herramientas como la abstracción, herencia, polimorfismo, etc… para poder modelar sistemas de la manera más sencilla posible. En las próximas líneas, se presentará un software denominado EcosimPro, que ha sido utilizado en el presente proyecto Fin de Carrera para modelar y simular determinados sistemas mecánicos.

Al finalizar este capítulo, el lector tendrá una visión global acerca de cómo trabaja EcosimPro y la opciones que esta herramienta ofrece para implementar y simular sistemas mecánicos.

3.1. Introducción

Ecosim comenzó a desarrollarse en 1989 producto de un contrato de Empresarios Agrupados Internacional (EAI) con la Agencia Espacial Europea (ESA) para la simulación de sistemas de control ambiental y soporte de vida en la Estación Espacial Internacional. En 1993 se completó la primera versión de Ecosim y en 1996 la segunda (ambas bajo UNIX). Con el desarrollo de la versión 3.0 (bajo Windows) se cambió el nombre de la herramienta de Ecosim a EcosimPro. La primera versión comercial apareció a finales de 1999. Hoy en día se puede encontrar en el mercado la versión 4.8, esta es la herramienta de modelado estándar de la ESA en las áreas de ECLSS (Environmental Control and Life Support System), Propulsión (Satélite y Cohete) y Sistemas Biológicos para misiones de larga duración. Para la realización del presente proyecto se ha utilizado la versión 4.6.

EcosimPro es una herramienta de modelado y simulación multidisciplinar de última generación, con un lenguaje de modelado (EL, EcosimPro Langauge) orientado a objetos que genera código C++ siguiendo un estándar desarrollado por la ESA. La descripción del modelo puede hacerse usando EL o usando el editor gráfico. Este dispone de un interfaz hombre-máquina que facilita la labor de creación de modelos de una manera intuitiva.

En este programa, el lenguaje de modelado EL impone el encapsulamiento de la información. Como ya se ha comentado anteriormente, los únicos elementos visibles de una componente en la fase de modelado son los de su interfaz pública: puertos, parámetros y datos. También se permite la composición de componentes y la especialización mediante herencia múltiple.

3.2. Comunicación con entornos externos

El uso de esta herramienta de simulación puede resultar más o menos complejo dependiendo del fin buscado. Se clasifican los niveles de usuario en base al grado de dificultad de uso de la siguiente manera: en un primer nivel aparecen modeladores de librerías, personas que requieren conocer a fondo la matemática de los componentes y el lenguaje de modelado. En un segundo nivel se encuentran los usuarios de librerías ya terminadas, lo que hacen es diseñar sistemas gráficamente. En el tercer nivel estarían las personas que crean múltiples experimentos sobre un modelo matemático ya cerrado. Y el cuarto y último nivel es para la gente que usa modelos de EcosimPro desde otros entornos externos, existe la posibilidad de trabajar con multitud de programas informáticos relacionados con esta herramienta de simulación como son Microsoft Excel, Matlab, clases C++, etc. La figura 3.1 muestra un gráfico donde se enumeran los diferentes programas que soportan la funcionalidad de EcosimPro:

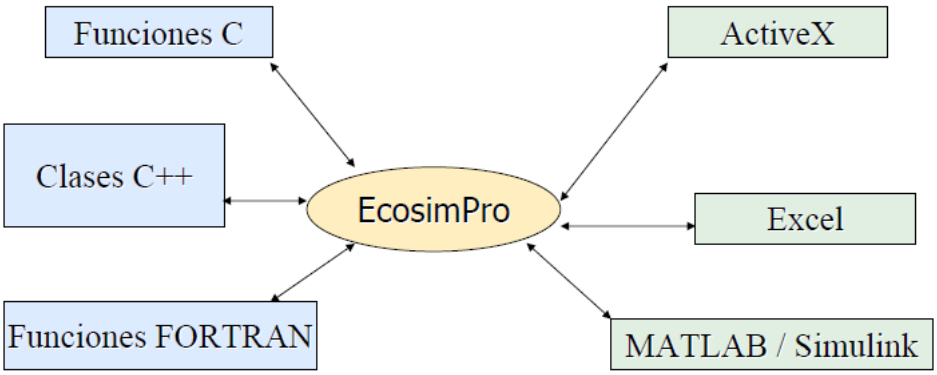


Fig. 3.1: Interfaz con programas externos.

Gracias a estas conexiones es posible usar toda la potencia que proporciona la herramienta sin necesidad de emplear el entorno de simulación y puede incluirse su funcionalidad en aplicaciones independientes desarrolladas por el usuario. Como ejemplo se puede citar la herramienta EcoExcel para exportar modelos de EcosimPro a Microsoft Excel a través de un *add-in* creado para conectarse a modelos generados desde la herramienta(figura 3.2).

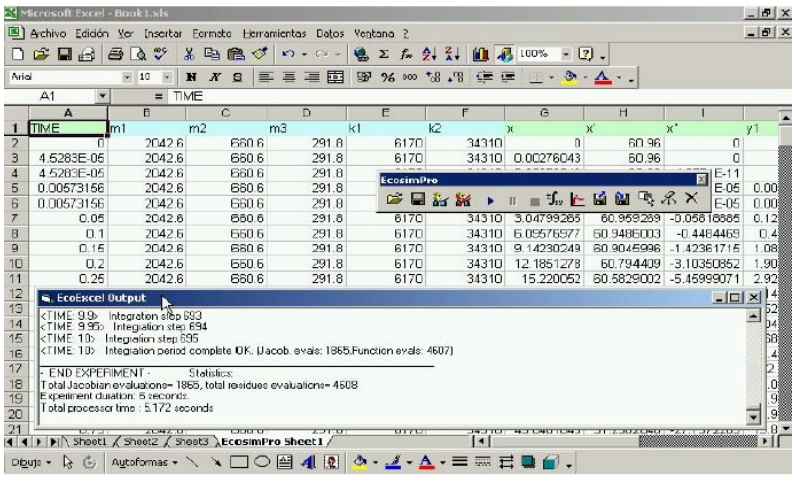


Fig. 3.2: *Add-in* de Excel generado desde EcosimPro.

3.3. "EL", el lenguaje de EcosimPro

El lenguaje que utiliza EcosimPro se denomina EL (EcosimPro Language). EL es un lenguaje desarrollado para modelar sistemas híbridos (continuos-discretos), que permite representar complejos modelos matemáticos basados en ecuaciones diferenciales. Uno de los objetivos de EL consiste en hacer esta tarea fácil y sencilla para todo usuario que pretenda hacer simulaciones de este tipo.
Los modelos en EL se representan de una forma natural e intuitiva. El EL constituye un simple pero poderoso lenguaje con el que realizar experimentos con los modelos implementados y calcular estados transitorios, estacionarios, ejecutar estudios paramétricos, etc.

El EL es un lenguaje para la resolución automática de sistemas de ecuaciones diferenciales. Pero como se comentó en el capítulo anterior, la complejidad de este tipo de lenguajes está oculta, esto es, todo lo que el modelador tiene que hacer consiste en definir ecuaciones de alto nivel, usando un lenguaje orientado a objetos, expresando las ecuaciones de la misma forma que lo haría el álgebra.

El EL es un lenguaje orientado a objetos, los componentes pueden *"heredar"* características de otros y pueden ser agregados, una vez probados, para crear componentes más complejos (modularidad).

Una vez que los modelos han sido creados, se pueden realizar experimentos con ellos. El EL posee un lenguaje para los experimentos muy similar al lenguaje de modelado que se utiliza para integrar el modelo, calcular estados estacionarios y, en definitiva, resolver el experimento.

En lo que se refiere a las habilidades matemáticas del EL, es importante señalar que este programa posee un manejo simbólico de las ecuaciones y potentes herramientas matemáticas para el cálculo de ecuaciones no lineales (Newton-Raphson [BRA05]) así como para ecuaciones diferenciales (DASSL [PET82][2] y Runge-Kutta [PIN98][3]). Estos métodos permiten simular problemas con cientos de variables.

EcosimPro dispone de asistentes matemáticos para:

- Definir el diseño de los problemas.
- Definir las condiciones de contorno.
- Reducir los problemas de índice superior de las DAE´s.

Además posee inteligentes algoritmos matemáticos basados en técnicas gráficas para minimizar el número de incógnitas y ecuaciones. También incorpora un potente gestor de eventos discretos para detectar cuándo ocurren dichos eventos [JOR08].

En los siguientes apartados se analizarán algunos de los aspectos más relevantes del EL así como aquellos que se han utilizado para implementar los distintos componentes y experimentos que se presentarán en los capítulos 4 y 5.

[2]es un código diseñado para el cálculo numérico de sistemas implícitos de ecuaciones diferenciales dadas de la forma F(t, y, y´)=0 donde F, y e y´ son vectores y los valores iniciales de y e y´ son conocidos.
[3]Los métodos de Runge-Kutta (RK) son un conjuntos de métodos iterativos (implícitos y explícitos) para la aproximación de soluciones de ecuaciones diferenciales ordinarias, concretamente, del problema de valor inicial.

3.4. Declaraciones en EL

El lenguaje EL permite usar declaraciones de tipo secuencial, continuas o discretas en función de las necesidades del modelo. Los lenguajes clásicos como FORTRAN o C++ sólo permiten declaraciones secuenciales. Los lenguajes orientados a eventos sólo permiten declaraciones de tipo discreto, mientras que otro tipo de lenguajes sólo admiten declaraciones de tipo continuo. El EL, sin embargo, acepta los tres tipos porque incluso aunque el principal propósito de EcosimPro es el modelado continuo, también se puede utilizar para simular componentes discretos o ejecutar códigos secuenciales. El uso de estos tres tipos de declaraciones se puede resumir de la siguiente manera:

- Las declaraciones ***secuenciales*** se pueden usar para las inicializaciones, funciones y para la programación de eventos discretos que requieren de un orden estricto en su ejecución.
  Estas declaraciones se ejecutan progresivamente y permiten al modelador controlar el flujo del programa, así, el flujo puede circular por una rama del código si ocurre un evento dado o bien transcurrir por otra distinta si ocurre un evento distinto, entrar en un bucle condicional, llamar a una función, etc.
  Las declaraciones secuenciales más simples son las "asignaciones", que básicamente consisten en asignar una expresión a una variable pre-declarada.

```
x= 6
x= y + 5.66
x= z / (maximum(z,y) * 0.45)
```

Fig. 3.3: Ejemplo de asignación secuencial.

La figura anterior representa un ejemplo de programación secuencial típico, en la que se establece una relación entre ambos miembros de la igualdad.

Por supuesto, dentro de las declaraciones de tipo secuencial del EL también se puede encontrar la clásica estructura *"IF-THEN-ELSE"*, cuyo propósito consiste en ejecutar una secuencia de declaraciones en función de valor de una ó más condiciones. La figura 3.4 es un ejemplo de estructura "IF-THEN-ELSE":

```
IF (signal == SIN) THEN
value = sin (2 * 3.14159 * x)
ELSEIF (signal == SQUARE) THEN
value = isqr
ELSE
value = 1.
END IF
```

Fig. 3.4: Ejemplo de estructura IF-THEN-ELSE.

Como se puede observar en el ejemplo este tipo de estructura asigna valores a una variable (en este caso a la variable *"value"*) en función del valor de otra (en este caso en función del valor de *"signal"*).

- Las declaraciones ***continuas*** se usan para declarar conjuntos de ecuaciones diferenciales, en las que carece de importancia el orden en que son declaradas, ya que serán ordenadas posteriormente por los algoritmos internos del programa. Estas declaraciones dan forma a la parte continua de los modelos.
  Cuando el programa ejecuta la parte continua de un modelo tiene en cuenta cualquier evento que haya sido programado en la parte discreta, y si alguno de ellos tiene lugar, la ejecución se detiene y se llevan a cabo las acciones asociadas a dicho evento.
  En el EL se usan ecuaciones continuas con cambios dinámicos de la forma de estas ecuaciones en función de ciertas condiciones y del momento exacto en que ocurren determinados eventos. La sentencia típica para este propósito, es la denominada "ZONE". En la figura 3.5 se representa un ejemplo de esta sentencia.

```
x= ZONE ( m > 0 ) y + 3*z
   OTHERS y + 2*z
```

Fig. 3.5: Ejemplo de sentencia ZONE.

Cuando el valor de *"m"* es mayor que cero, la ecuación válida es $x= y+3*z$, sin embargo, para cualquier otro valor la ecuación válida es $x= y+2*z$. La figura 3.6 muestra el comportamiento de la sentencia "ZONE" del ejemplo anterior, donde es importante fijarse en los instantes t_1, t_2 y t_3, en los cuales las ecuaciones cambian y se produce la re-inicialización de las variables.

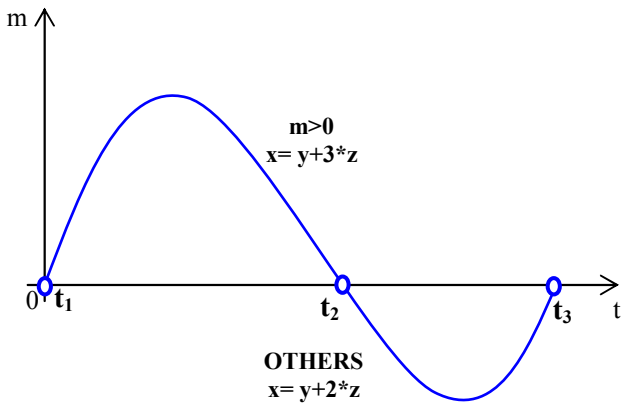


Fig. 3.6: Cambio en el valor de ecuación al detectar evento en sentencia ZONE.

Cuando sea necesario resolver sistemas de ecuaciones no lineales, se pueden programar ecuaciones condicionales mediante la sentencia *"IF"*. Esta forma de modelado es idónea cuándo las ecuaciones condicionales no varían durante la simulación (típico de estados estacionarios), como cuando el modelo se diseña para sistemas en los que la condición toma valores al comienzo de la simulación y permanecen constantes hasta el final, como en el ejemplo de la figura 3.6.

```
-- se define una variable de tipo enumerativo con vaarios modos
ENUM modes= { DESIGN, OFF_DESIGN, TRANSIENT }
--  se crea un componente llamado tif
COMPONENT tif
DECLS
REAL x= 3
ENUM modes switchMode= DESIGN
CONTINUOUS
x= IF ( switchMode == DESIGN ) 5.6
ELSEIF( switchMode== TRANSIENT ) sin(TIME)
ELSE sin(TIME) + cos(TIME)
END COMPONENT
```

Fig. 3.7: Ejemplo programa ecuaciones condicionales sin detección de eventos.

En el ejemplo de la figura anterior el valor de la *"x"* es función de la variable de tipo enumerativo *"switchMode"* que, como se comentó anteriormente tomará su valor al inicio de la simulación, y lo mantendrá hasta que ésta finalice. La ventaja de esta forma de programar radica en que el sistema no tiene que resolver todo el sistema de ecuaciones (al contrario de lo que ocurre con la sentencia *"ZONE"*) ya que el cambio es inmediato y no se introduce ningún retardo en la solución (sobre todo en el caso de sistemas complejos).

- Las declaraciones ***discretas*** se usan para declarar eventos. Estos eventos se controlan mediante declaraciones condicionales que indican cuándo ocurre el evento y qué acción se lleva a cabo cuando eso ocurre. La activación (ó desactivación) de los eventos tiene lugar cuando se hace *"verdadera"* (ó *"falsa"*) una condición lógica declarada a través del correspondiente operador lógico (AND, OR, NOT, ==, <=, >=, <, > y !=).

  Un ejemplo se muestra en la figura 3.8:

```
DISCRETE
   WHEN( velocidad > 3.4 ) THEN
      umbral_sobrepasado= TRUE
   END WHEN
```

Fig. 3.8: Ejemplo programación discreta con activación/desactivación de eventos.

Cuando la se cumple la condición *"velocidad > 3,4"*, se produce la activación del evento programado, es decir, *"umbral_sobrepasado = TRUE"* y por tanto esta variable sería verdadera en los puntos t_1 y t_2 tal y como se muestra en la figura 3.9.

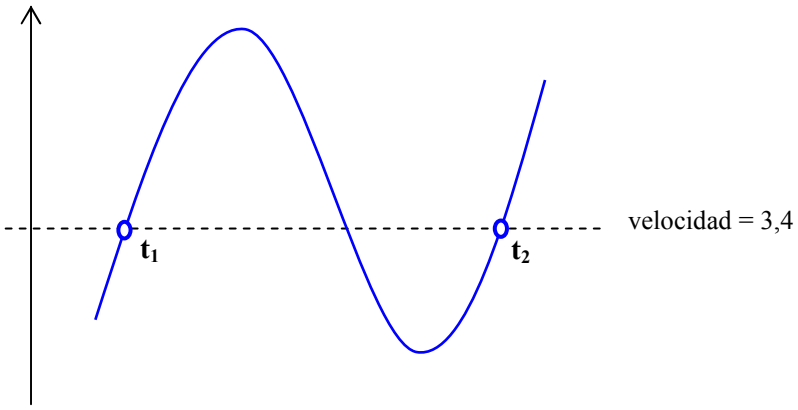


Fig. 3.9: Cambio en el valor de ecuación al detectar evento en sentencia ZONE.

Evidentemente, todo lo explicado hasta ahora tan sólo es una pequeña parte de las posibilidades del lenguaje EL. Para profundizar más en dicho lenguaje ó para resolver posibles dudas, el lector puede recurrir a la referencia bibliográfica [EAI09], dónde encontrará una descripción más detallada de todas las características y funcionalidades del mismo.

3.5. Modelado de componentes

Los *componentes* son los elementos más importantes del *EL*. Son el instrumento básico para representar desde los modelos más sencillos hasta los más complicados. Por tanto, mediante los componentes se describe el comportamiento, tanto continuo como discreto, de los modelos.

Normalmente, los componentes se crean para representar entidades físicas individuales como una resistencia, una válvula o un motor. Es decir, las componentes representan objetos físicos que actúan de manera independiente pero que pueden estar conectados con otros. Un componente, por tanto, puede ser algo tan simple como una ecuación continua, un pequeño código que gestione un evento discreto o puede ser la unión de otros componentes y no poseer ecuaciones propias. Resumiendo, todos los elementos de modelado en *EL* reciben el nombre de componentes.

En lo que se refiere a la sintaxis de la programación de los componentes, el nombre del *componente* se declara mediante un identificador que se escribe a continuación de la palabra "*component"*. Además un *componente* puede contener otros bloques (ó partes) que contengan información adicional sobre las características del componente.

En la figura 3.10 se observa un programa en el que se declara un *componente* denominado *R_Inertia*.

```
COMPONENT R_Inertia IS_A R_Rigid
"Rotating mass with inertia"
   DATA
      REAL I           UNITS "kg*m^2"
      REAL phi_0 = 0   UNITS "rad"

   DECLS
     REAL alpha        UNITS "rad/s^2"

   INIT
      phi = phi_0

   CONTINUOUS

      phi'' = alpha
      I * alpha = m_in.tau - m_out.tau

END COMPONENT
```

Fig. 3.10: Ejemplo declaración de componente en EL.

Los principales bloques y características de un *componente* se definen a continuación:

- ***Componentes abstractos***, usando la palabra clave *ABSTRACT*. Como ya se ha comentado en capítulos anteriores, los componentes pueden ser *abstractos* o *concretos*. Los componentes *abstractos* describen comportamientos que en sí mismos, no representan ningún componente físico, y sólo pueden usarse como base para otros componentes. Por ejemplo, en los componentes eléctricos es posible definir un componente interfaz común que defina la entrada y la salida de la mayoría de los componentes eléctricos (Fig. 3.11).

```
ABSTRACT COMPONENT twoPorts
   PORTS
      IN elec pi "input port"
      OUT elec po "output port"
END COMPONENT
```

Fig. 3.11: Ejemplo declaración de componente abstracto en *EL*.

Este tipo de componentes se denominan *componentes interfaz*, ya que sólo contienen puertos que conectan con el entorno exterior. Cualquier componente que herede de él, heredará también su misma interfaz.

Pero un *componente abstracto*, al igual que los *concretos*, también puede estar formado por ecuaciones. Es decir, un *componente abstracto* puede ser igual que cualquier otro componente, pero con la diferencia de que no puede ser usado para crear el modelo final.

La ventaja de estos elementos radica en el elevado número de líneas de código que ahorran a los modeladores, ya que reúnen ecuaciones e interfaces que pueden ser compartidas por una gran cantidad de componentes distintos, lo que supone una gran ventaja en cualquier área del modelado.

- El EL permite la ***herencia*** simple y múltiple entre sus componentes. Es decir, que una subclase admite como suyos todos los datos y ecuaciones de la superclase a la que pertenece. Este mecanismo permite unificar en componentes abstractos o concretos cualquier comportamiento que sea común a componentes del mismo tipo. Un ejemplo se muestra en la figura 3.12.

```
ABSTRACT COMPONENT R_Rigid IS_A R_Two_Ports
"Abstract class for the definition of a rigid connection of two rotational ports"
   DECLS
      REAL phi    UNITS "rad"      "Absolute angular position (rad)"

   CONTINUOUS
      m_in.phi = phi
      m_out.phi = phi
END COMPONENT
```

Fig. 3.12: Programa del componente *R_Rigid* con herencia de *R_Two_Ports*.

El componente *R_Rigid* hereda todas las características de *R_Two_Ports*. En este caso hereda los dos puertos *m_in* y *m_out* que se usan en la subclase como si estuvieran declarados en ella. En la figura 3.13 se muestra el resultado de programar el componente *R_Rigid* sin usar la herencia.

```
ABSTRACT COMPONENT R_Rigid
"Abstract class for the definition of a rigid connection of two rotational ports"
   PORTS
      IN  mech_rot m_in      "Left / driving mech_rot"
      OUT mech_rot m_out     "Right / driven mech_rot"


   DECLS
      REAL phi    UNITS "rad"      "Absolute angular position (rad)

   CONTINUOUS
      m_in.phi = phi
      m_out.phi = phi
END COMPONENT
```

Fig. 3.13: Programa del componente *R_Rigid* sin herencia de *R_Two_Ports*.

Ahora el componente *R_Rigid* podría usarse como base de otros componentes y así sucesivamente.

El usuario también puede beneficiarse de la herencia múltiple, en este caso el componente heredará parámetros, puertos, datos, variables y comportamientos de varios componentes padres ó superclases. Sólo hay una restricción importante, los componentes padre no podrán tener variables con el mismo nombre, de otro modo sería imposible identificar las variables en el componente hijo ó subclase. La figura 3.14 muestra un caso de herencia múltiple a partir de tres elementos simples:

```
COMPONENT base1
DECLS
REAL x
END COMPONENT

COMPONENT base2
DECLS
REAL y
END COMPONENT

COMPONENT base3
DECLS
REAL z
END COMPONENT

COMPONENT hijo IS_A base1, base2, base3
CONTINUOUS
x = y + z
END COMPONENT
```

Fig. 3.14: Ejemplo programación con herencia múltiple.

Este ejemplo resulta trivial porque los componentes *"base"* no poseen ecuaciones complicadas o comportamientos inteligentes, pero para casos más complicados el concepto es el mismo.

En el supuesto de que *"base3"* tuviera una variable llamada *"x"*, el compilador mostraría un mensaje de error en el componente *"hijo"* porque otra variable con el mismo nombre habría sido declarada en el componente "base1", y en la ecuación *"x=y+z"* no se podría deducir si la variable *"x"* pertenece a *"base1"* ó a *"base3"*.

- La comunicación de los componentes con el universo exterior se realiza por medio de los ***puertos*** de conexión. Los puertos sirven para conectar unos componentes con otros y sólo se pueden usar con esta finalidad. La sintaxis para declarar un puerto se muestra en la figura 3.15.

```
PORTS
[IN | OUT] port_type NOMBRE [,IDENTIFIER] [(parameters_init)] ( CARDINALITY range )
```

Fig. 3.15: Ejemplo sintaxis programación de un puerto.

Donde:

- Mediante *"IN"/ "OUT"* se indica si el puerto es de entra o salida.
- Con *"port_type"* se indica el tipo de puerto.
- En la zona reservada para el nombre del puerto se puede introducir entre corchetes otro identificador.
- Con *"parameters_init"* se puede introducir una lista con la estructura inicial de ciertos parámetros, si se da el caso de que el puerto lo requiere.

- Mediante *"CARDINALITY"* se puede especificar el número máximo y mínimo de conexiones en el puerto.

Las figuras 3.16 y 3.17 sirven para entender cómo funcionan los puertos, y la manera en que conectan componentes entre sí. Si se define el puerto tal y como aparece en la figura 3.16 sólo se consigue declarar dos variables de tipo *REAL*, *"v"* e *"i"*, en aquellos componentes que tengan como puerto *elec_port* , es decir, no se ha introducido ninguna ecuación en el código del puerto que se informe acerca de cómo se relacionan estas variables.

```
PORT elec_port
   REAL v UNITS "V" "Tensión"
   REAL i UNITS "A" "Intensidad"
END PORT
```

Fig. 3.16: *elec_port* sin información del comportamiento de variables.

Sin embargo EcosimPro permite afinar más. Se sabe que la tensión es igual entre todos los componentes conectados entre sí. Esto se puede indicar mediante el modificador *EQUAL*, así mismo la suma de las corrientes que salen y entran de un nudo es la misma (leyes de Kirchoff), indicándose esto por medio de la cláusula *SUM*, quedando la definición de *elec_port*:

```
PORT elec_port
   EQUAL REAL v UNITS "V" "Tensión"
   SUM   REAL i UNITS "A" "Intensidad"
END PORT
```

Fig. 3.17: *elec_port* con información del comportamiento de variables.

Sí se supone que tres componentes eléctricos cualesquiera *C1*, *C2* y *C3* se conectan entre sí por medio de una interfaz *elec_port p*, como en la figura 3.17.

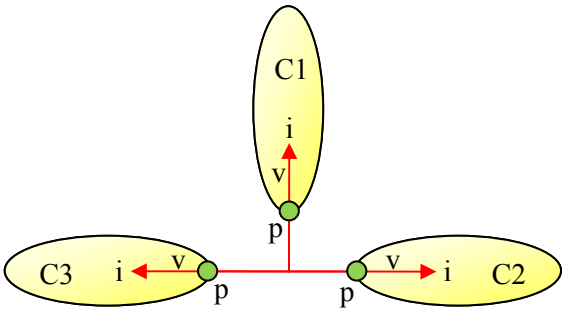


Fig. 3.18: Esquema de conexiones de C1, C2 y C3.

Las ecuaciones que Ecosimpro genera internamente para conectar los componentes C1, C2 y C3 según el esquema de la figura 3.16 serán:

- Para la variable v de elec_port que está modificada por EQUAL, se generan las ecuaciones:

$$C1.p.v = C2.p.v \quad (3.1)$$
$$C1.p.v = C3.p.v \quad (3.2)$$

- Para la variable I de elec_port, modificada por SUM, la ecuación es:

$$C1.p.i + C2.p.i + C3.p.i = 0 \quad (3.3)$$

- Los principales **tipos de variables** en los componentes se engloban en tres categorías:

  - El *bloque "Paraméter"* está formado por construcciones paramétricas que sólo se usan para instanciar el componente.
  - En el bloque *"DATA"* se encuentran todas las variables que son datos del problema y cuyo valor es conocido. Además su valor no permanece constante en el tiempo.
  - En el bloque *"DECLS"* se declaran las variables del componente. Estas variables no tienen conexión con el universo exterior y son calculadas en el interior del componente.

  Los dos primeros bloques se usan para definir el estado inicial del componente, y el bloque *"DECLS"* se usa para definir las variables locales para el componente.

- En ocasiones los componentes requieren un **bloque de inicialización**, denominado "INIT", en el cual se especifican una serie de instrucciones secuenciales que deben ser ejecutadas antes de la simulación. Es decir, se utiliza este bloque para inicializar aquellas variables que necesitan un valor inicial antes de lanzar la simulación.

- Todos los elementos discretos de la programación del componente deben ir en el **bloque "DISCRETE"**. La información discreta, como se comentó en el apartado 3.4, consiste en eventos que deben ser detectados durante la simulación. Una vez que el evento es detectado, el sistema ejecuta la secuencia de acciones asociada a dicho evento, las cuales pueden activar otros eventos. Cuando no ocurren más eventos, el sistema continúa ejecutando la parte continua de componente, si es que ésta existe.
  En este bloque sólo pueden usarse declaraciones de tipo discreto, entre las que destaca la declaración *"WHEN"*.

- En el **bloque "CONTINUOUS"** se programa la parte continua del componente. Al igual que en el bloque "DISCRETE", en este bloque sólo se pueden usar declaraciones de tipo continuo. En este bloque es donde se definen las ecuaciones diferenciales del modelo que el compilador de EcosimPro tendrá que resolver.

- La integración de un modelo continuo y discreto, se denomina **modelo híbrido**, y este tipo de modelos constituyen el núcleo principal de EcosimPro. En la figura 3.19 se representa un diagrama de flujo que ilustra la secuencia que sigue EcosimPro durante la simulación de modelos híbridos.

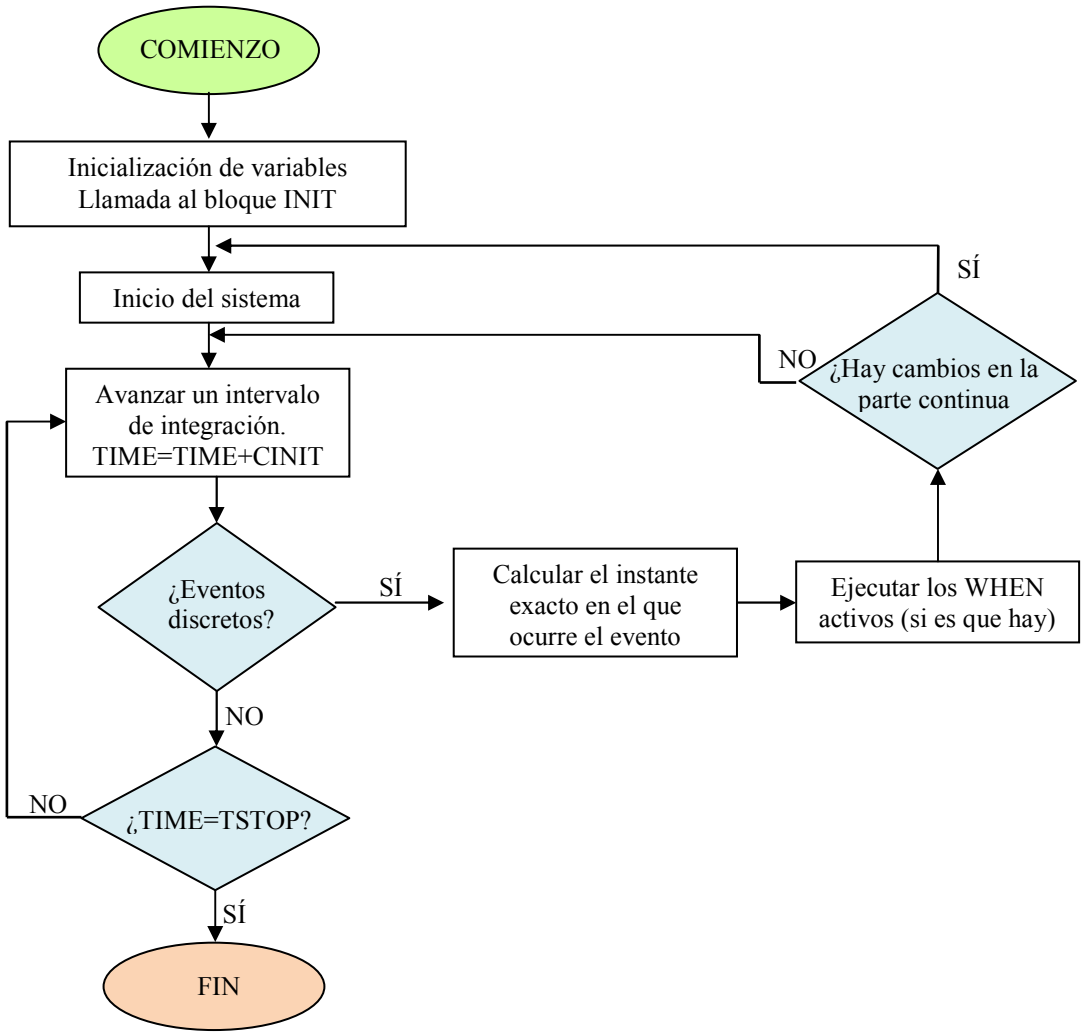


Fig. 3.19: Diagrama de flujo durante la simulación en sistemas híbridos.

Para detectar el momento exacto en el que ocurre un evento, se usa la declaración *"WHEN"* en el bloque discreto y la declaración *"ZONE"* en la parte continua del modelo. Si alguna de estas declaraciones involucra el uso de alguna variable continua, dará lugar a una condición de cambio de estado. El diagrama de la figura 3.19 ilustra cómo el proceso continuo es detenido por la parte discreta cada vez que se produce una condición de cambio de estado (=evento) en alguna de las declaraciones tipo *"WHEN"* o *"ZONE"*.

Por tanto, como se ha analizado en esta sección, los componentes constituyen los elementos básicos del EL, y mediante la agrupación e instanciación de los mismos se consigue dar forma a los modelos que serán objeto de estudio haciendo uso de la simulación.

Asimismo, una vez definidos los componentes hay que compilarlos, esto es, chequear la sintaxis, aunque es algo que EcosimPro hace automáticamente. Si el código es incorrecto, el programa informará mediante un mensaje de error y habrá que revisar el modelo para intentar arreglarlo y conseguir que funcione correctamente. Una vez que no haya errores se dará el visto bueno y se podrá utilizar el componente para crear modelos que serán simulados con posterioridad.

3.6 Creación de modelos

Como se explicó en el capítulo 2, un sistema puede definirse como un conjunto de objetos o entidades que interactúan entre sí para alcanzar un cierto objetivo, y estas entidades que dan forma a los sistemas son los componentes, cuyas características principales se analizaron en la sección anterior. Así, un sistema real puede ser modelado mediante un componente o estar formado por varios de ellos.

Aunque en realidad lo que permite mejorar el conocimiento acerca del funcionamiento de los sistemas reales es lo que se denomina *simulación* ó *experimentación*. No obstante, la correcta generación del modelo, es un aspecto que resulta fundamental si el usuario pretende obtener algún beneficio de dicha experimentación.

EcosimPro ofrece dos posibilidades para la generación de modelos. La primera consiste en introducir manualmente el código, mientras que la segunda permite construir el componente en un entrono gráfico y posteriormente el compilador de EcosimPro se encargará de traducir los símbolos gráficos y sus conexiones en el código adecuado.

Independientemente de la opción escogida, el resultado final será un componente parecido al de la figura 3.20.

```
USE ROTACIONAL

COMPONENT SISTEMA_ROT_2GDL
   TOPOLOGY
      PARED_ROT P1 (phi_0=0)
      MUELLE_ROT M1 (k=157913.4, phi_rel0=0)
      INERCIA_ROT_2PUERTOS I2 (I=4000, phi_0=0 ,omega_0=0)
      DAMPER_ROT D1 (b=12566.36, phi_rel0d=0)
      MOMENTO_ROT PAR1
      MUELLE_ROT M2 (k=157913.4, phi_rel0=0)
      INERCIA_ROT_2PUERTOS I1 (I=4000, phi_0=0 ,omega_0=0)

      CONNECT P1.p TO M2.q
      CONNECT M2.p TO I1.p
      CONNECT I1.q TO M1.q
      CONNECT I1.q TO D1.q
      CONNECT M1.p TO I2.p
      CONNECT D1.p TO I2.p
      CONNECT I2.q TO PAR1.p
 END COMPONENT
```

Fig. 3.20: Código del modelo de un sistema rotacional con dos grados de libertad.

Cada uno de los bloques que aparecen en el componente se explican a continuación:

- **USE:** en este bloque se hace mención a la, o las, librerías donde se encuentran los modelos de los elementos que van a formar el sistema.
- **COMPONENT:** en el bloque *component* contiene el nombre del componente a partir del cual generaremos nuestros experimentos.
- **TOPOLOGY:** en este apartado se introducen todos los componentes que van a formar el sistema, hay que asignarles un nombre para luego poder hacer referencia a ellos. También es posible crear o modificar el valor de las constantes de los componentes.
- **CONNECT_TO:** mediante la sentencia *connect_to* se realizan las conexiones entre los componentes del sistema, es decir, se define la estructura del sistema indicando cómo se conectan entre sí los distintos elementos que lo componen.

Una vez creado el componente, ya se puede crear la partición, para que el programa genere el modelo matemático del sistema, y después se crea un experimento para comenzar con la simulación.

A pesar de disponer de dos vías para la generación de modelos, la gran versatilidad de EcosimPro se basa en la existencia de un entorno gráfico que permite implementar modelos en un tiempo muy reducido y sin tener nociones de programación. Por estas razones el entorno gráfico de EcosimPro, desempeña un papel muy importante en la simulación de sistemas. A continuación se analizará en detalle cómo se pueden implementar sistemas mediante el esta herramienta.

3.6.1. Representación gráfica

Hasta ahora se ha explicado cómo se crean componentes en EcosimPro de forma manual, esto es, escribiendo el texto del código directamente. Una de las ventajas de EcosimPro, es que proporciona una herramienta para poder asignar iconos a los componentes de una librería. Así, pueden definirse componentes más complejos usando sólo los iconos de las librerías y las conexiones entre ellos, obteniéndose una mayor velocidad de desarrollo. El compilador de EcosimPro se encarga de traducir los símbolos gráficos y sus conexiones en el código adecuado, evitando todo este proceso al usuario.

Para asociar símbolos gráficos a componentes con EcosimPro hay que seguir los siguientes pasos:

1) Seleccionar la librería dentro de la cuál se encuentran los componentes a los que se quiere asignar los iconos (Fig. 3.21).

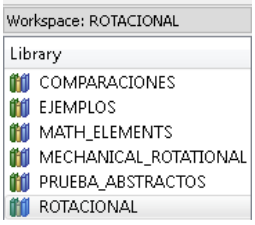


Fig. 3.21: Selección de a librería *ROTACIONAL* en un *Workspace*.

2) Seleccionar, dentro de la librería escogida, el componente al que se quiere asignar un icono. Una vez seleccionado el componente, construir el icono mediante las herramientas del entorno gráfico (Fig. 3.22).

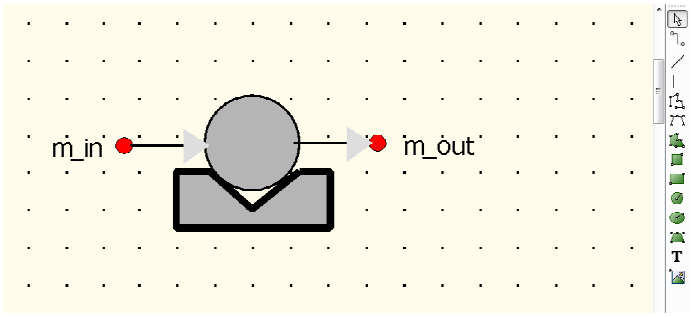


Fig. 3.22: Construcción icono mediante herramienta del entorno gráfico.

3) Cuando un icono se haya asignado a un elemento, aparecerá en una ventana situada en la parte inferior izquierda asociado al nombre del elemento (Fig. 3.23).

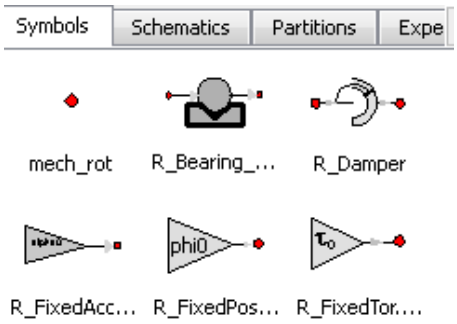


Fig. 3.23: Iconos asociados a los componentes.

Es recomendable comenzar con la creación de los puertos, ya que éstos forman parte de elementos más complejos. También el programa ofrece la posibilidad de asociar a un componente una imagen desde un archivo externo, lo que ayuda a reducir el tiempo de diseño.

3.6.2. Construcción de componentes y sistemas con el entorno gráfico

Una vez creados los iconos para todos los componentes de la librería, ya se puede construir y simular un sistema a partir de los mismos.

Del mismo modo que se asociaron los iconos a los componentes, para poder crear un sistema de forma gráfica hay que seleccionar la librería con la que se quiere trabajar y

seleccionar el entorno grafico para desarrollar el sistema, esto se hace eligiendo la vista esquemática, es decir, *Schematic View* (Fig. 3.24).

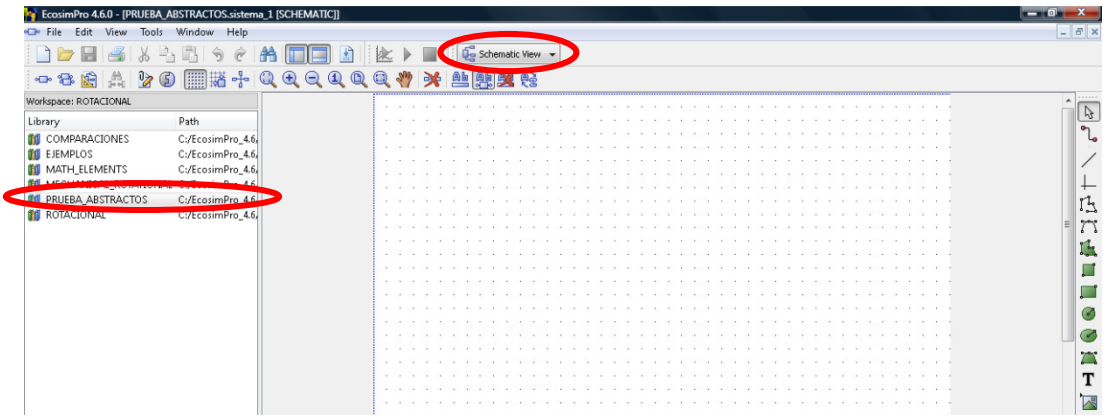


Fig. 3.24: Selección librería de trabajo y de entorno gráfico.

Dentro de esta librería se elige el icono del componente que se quiere usar para unirlo con otros componentes y formar así el sistema. Para incluir componentes en el sistema hay que arrastrarlos hasta el panel de dibujo como se muestra en la figura 3.25.

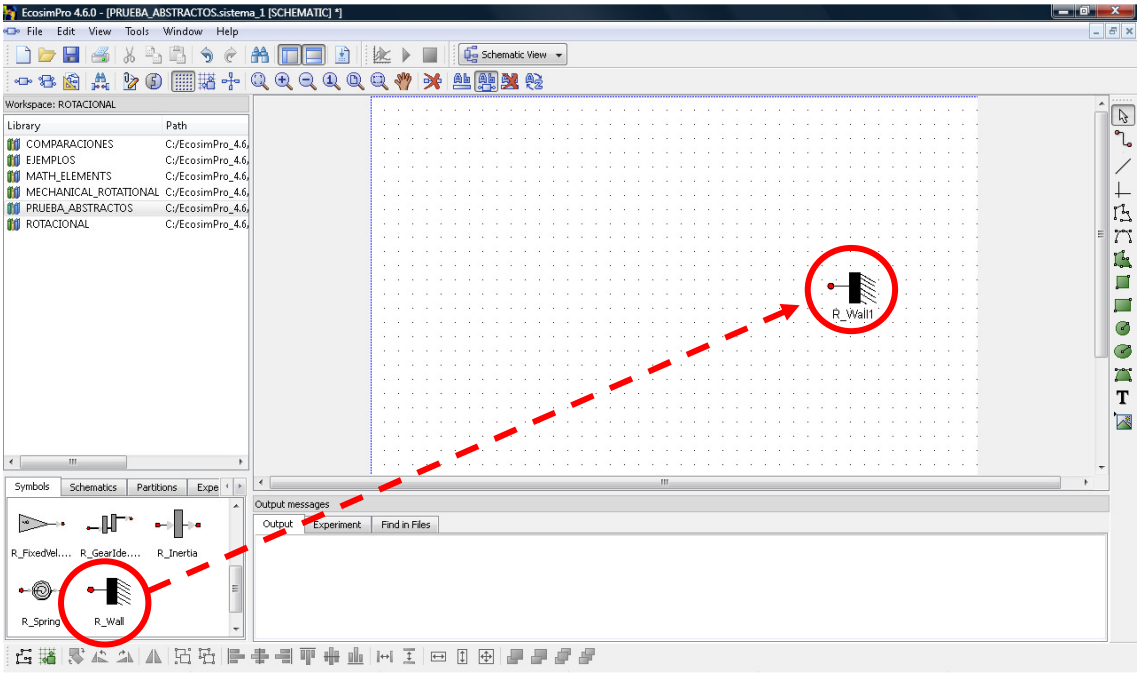


Fig. 3.25: Construcción del sistema en el entorno gráfico.

Tras arrastrar todos los componentes del sistema hasta el panel de dibujo, el siguiente paso será unir los puertos de los componentes con conectores (Fig. 3.26).

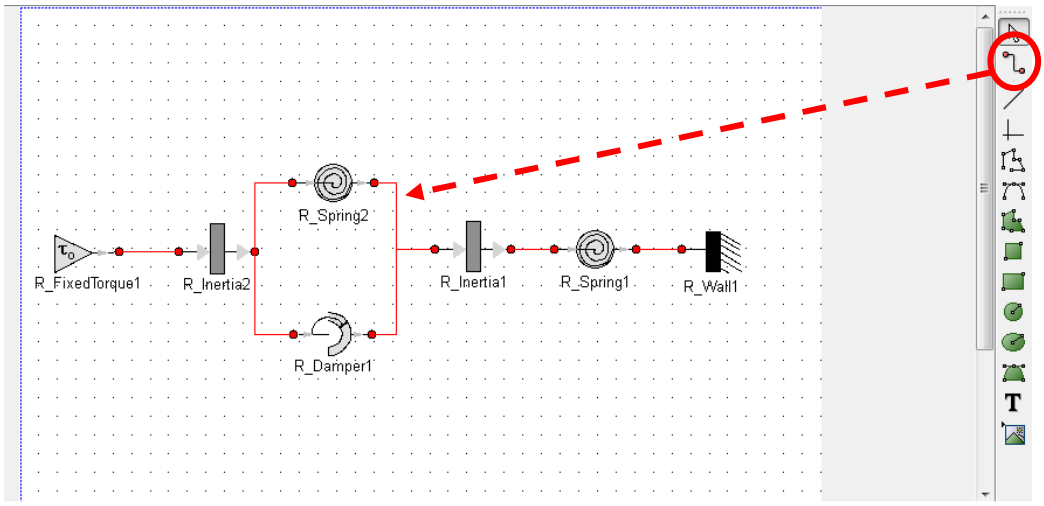


Fig. 3.26: Unión de los componentes del sistema con conectores.

También hay que asignar los valores de las constantes a cada uno de los componentes. Para ello basta con pinchar dos veces sobre el elemento, y aparecerá el cuadro de diálogo que se muestra en la figua 3.27. El campo *name* permite cambiar el nombre al elemento y, además, en la parte inferior se puede modificar el valor de los parámetros que lo definen.

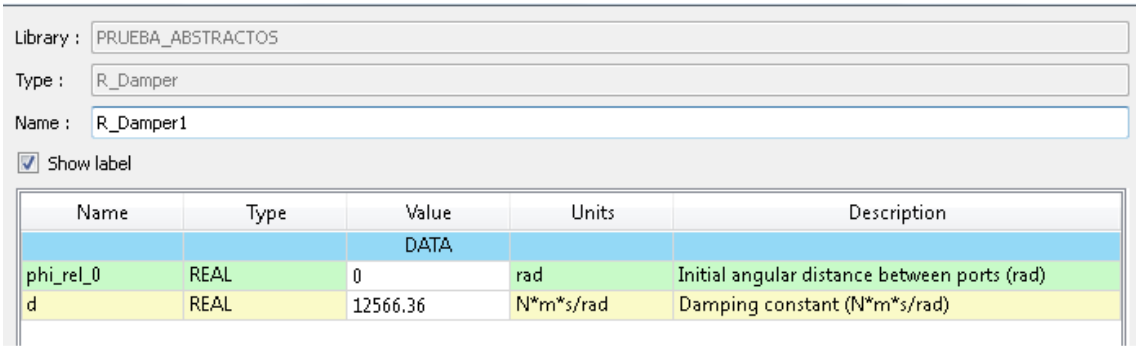


Fig. 3.27: Panel de instanciación del componente R_Damper1.

Una vez conectados todos los componentes y realizadas todas las instancias necesarias, es preciso convertir la información gráfica del nuevo componente a código EL y, posteriormente, compilarlo para, entre otras cosas, verificar la sintaxis y consistencia del modelo creado. Todo ello se consigue pinchando en el botón *Compile* de la barra de herramientas principal.

Finalizada la compilación, si se selecciona *code view* y después la pestaña *Files* del panel inferior izquierdo (ver figura 3.28), puede apreciarse que se ha creado un nuevo fichero, con el mismo nombre del sistema creado gráficamente, y extensión *.el*, que es el código de texto correspondiente al sistema gráfico. Éste código ha de ser equivalente al de la figura 3.20, en el que se creó un sistema equivalente, pero haciendo uso del lenguaje EL en lugar del entorno gráfico.

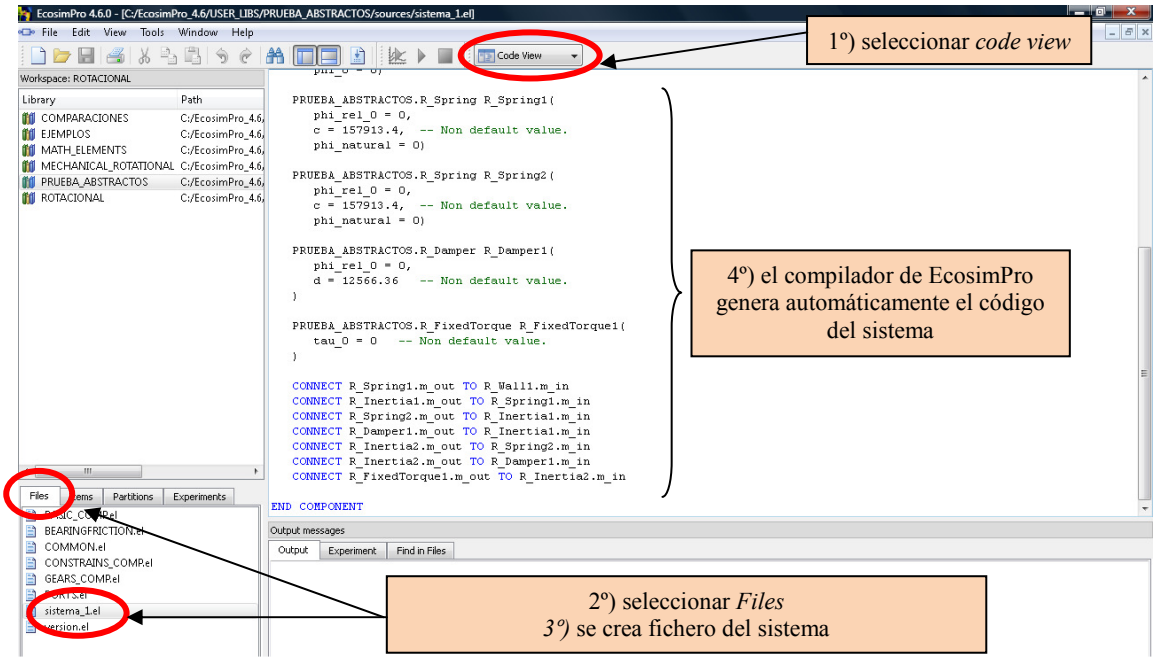


Fig. 3.28: Procedimiento a seguir para obtener el código EL del modelo diseñado en el entorno gráfico.

Finalmente, habría que crear una partición por defecto y un experimento para poder comenzar con la simulación.

Por tanto, mediante el entorno gráfico que ofrece EcosimPro y la posibilidad de utilizar librerías ya existentes, cualquier usuario puede crear modelos con gran facilidad y sin tener grandes nociones de programación. Esta circunstancia convierta a EcosimPro en una herramienta muy potente, ya que casi cualquier usuario podría utilizar este entorno gráfico para generar modelos y simularlos en un tiempo muy reducido.

3.7. Simulación de sistemas

Una vez que el modelo del sistema ha sido desarrollado y verificado, ya se pueden hacer simulaciones con él. EcosimPro dispone de un lenguaje para crear los experimentos muy similar al que se utilizó en los apartados anteriores para implementar los distintos componentes. La misión de este lenguaje es permitir al usuario inicializar los datos, calcular los estados estacionarios o integrar el modelo en un tiempo determinado mediante el uso de una secuencia de instrucciones.

Para declarar un nuevo experimento, es necesario identificarlo con un nombre, así como indicar la partición en la que el experimento se llevará a cabo. En la figura 3.29 aparece un ejemplo, que representa la forma en la que se declara un experimento en EcosimPro.

```
EXPERIMENT exp1 ON sistema_1.default
```

Fig. 3.29: Declaración de un experimento en EcosimPro.

Este texto indica que se ha declarado el *exp1* y que además se llevará a cabo en la partición *sistema_1.default*.

La sintaxis de un experimento consta de varios bloques claramente diferenciados y que se pueden ver en el ejemplo de la figura 3.30.

- **INIT:** es la parte donde se inicializan las variables que durante el proceso de creación de la partición EcosimPro ha detectado que son variables de estado algebraicas y, por lo tanto, tienen que ser inicializadas.
- **BOUNDS:** en esta parte se asignan valores a las variables que EcosimPro detecta como variables que no pueden despejarse de ninguna ecuación, que no están definidas en ningún lado y que no son variables de estado. Son las llamadas variables de contorno o de frontera.
- **BODY:** es el cuerpo de la simulación, aquí se asignan valores a parámetros como, por ejemplo, a la duración de la simulación.

En la figura siguiente se representa el código de un experimento generado por EcosimPro, el cual además puede ser modificado según las necesidades del modelador. Así se podrían inicializar las variables a gusto del modelador, cambiar el tiempo de finalización de experimento, añadir otras ecuaciones adicionales, etc.

```
EXPERIMENT exp1 ON pipeCircuit.math1
   DECLS
   INIT
   BOUNDS
      -- Set equations for boundaries: boundVar
      Pipe1.hp_in.p = 101325
      Pipe1.hp_in.w = 10 * (1 + sin(TIME))
      Pipe5.hp_out.w = 2 * (1 + sin(TIME))
      Pipe6.hp_out.w = 3 * (1 + sin(TIME))
      Pipe7.hp_out.w = 4 * (1 + sin(TIME))

   BODY
      -- report results in file reportAll.rpt
      REPORT_TABLE("reportAll.rpt", "*")
      -- for example integrate the model 15 seconds
      TIME = 0
      TSTOP = 15
      CINT = 0.1
      INTEG()
END EXPERIMENT
```

Fig. 3.30: Ejemplo del código de un experimento.

EcosimPro genera automáticamente las ecuaciones del modelo completo, detectando automáticamente las variables equivalentes y las ecuaciones triviales. Detecta los problemas que puedan surgir de índice superior y si estos problemas son debidos a ligaduras lineales entre las variables que aparecen derivadas, lo resuelve eliminando del modelo estas ligaduras y las variables que aparecen derivadas. En caso contrario aplica el algoritmo de *Pantelides* y muestra qué variables pueden ser seleccionadas como variables de estado, indica cuantas deben escogerse y propone una selección.

Una vez que el problema no tiene índice superior el programa compara el número de ecuaciones y de incógnitas del sistema. Si el número de ecuaciones es mayor que el de incógnitas muestra un mensaje de error indicando el conjunto de ecuaciones redundantes. Si hay más incógnitas que ecuaciones es preciso especificar condiciones de contorno adicionales. EcosimPro muestra una posible elección de variables sobre las que imponer las condiciones de contorno. Gracias a todas estas ayudas es relativamente sencillo poder simular el experimento deseado.

Además EcosimPro dispone de una herramienta denominada *GUI* (Interfaz Gráfica de Usuario) que permite monitorizar el experimento, así como conocer y cambiar el valor de las variables durante la simulación. En la figura 3.31 se puede observar la monitorización de diferentes tipos de experimentos mediante la interfaz gráfica.

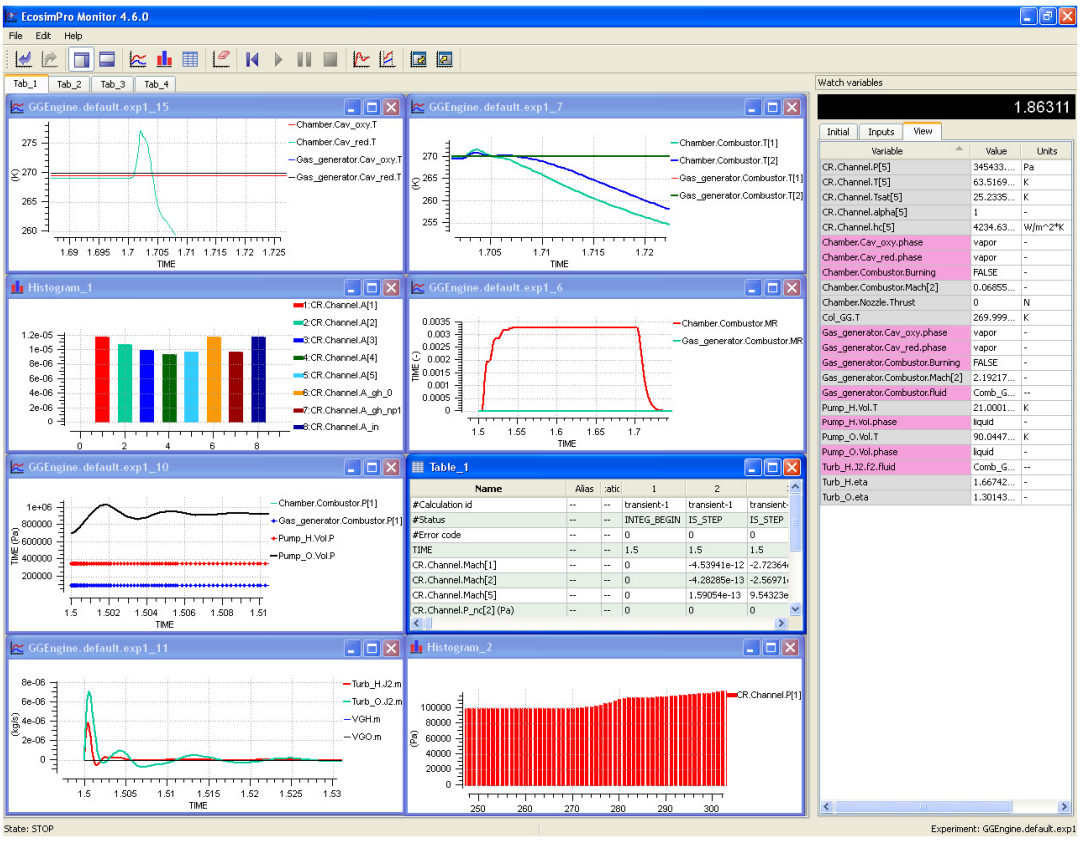


Fig. 3.31: Monitorización de distintos experimentos mediante el *GUI*.

CAPÍTULO 4

CREACIÓN LIBRERÍA "ROTATIONAL LIBRARY"

Una primera parte de esta sección se ocupa de la creación de puertos que se van a incluir en la librería. La segunda parte se dedica a la creación de clases abstractas de las cuales luego heredan otras clases o componentes. Y por último, en la tercera parte se definen todos los componentes de la librería con los cuales se pueden formar sistemas mecánicos rotacionales. En la presente sección del proyecto se va a describir la creación una librería con diferentes elementos mecánicos rotacionales.

Como ya se comentó en el capítulo 2, todos los sistemas mecánicos con dinámica rotacional, se pueden modelar, en parte, mediante el uso de tres elementos básicos de la mecánica rotacional. Estos elementos son la masa rotacional ó inercia, el muelle de torsión y el amortiguador de torsión.

4.1 Estructura de la librería "*Rotational library*"

Una de las ventajas que presenta EcosimPro consiste en poder organizar los componentes en librerías dependiendo de múltiples criterios, que serán escogidos por el modelador. Una clasificación lógica podría consistir en estructurar librerías según la disciplina a la que pertenecen los elementos que la componen, de esta manera se podría distinguir entre librerías de elementos mecánicos, eléctricos, de control, frigoríficos, etc. E incluso se puede crear, dentro de la parte mecánica, una librería de elementos mecánicos trasnlacionales y una librería de elementos mecánicos rotacionales. Es decir, las librerías permiten al modelador, clasificar y estructurar la información según los criterios más adecuados para cada modelo, pudiendo beneficiarse así de la modularidad de los lenguajes MOO.

Pero crear librerías no consiste simplemente en agrupar elementos que tienen características comunes. Para crear una librería hay que modelar una serie de puertos y de clases abstractas que luego ayudarán, mediante la abstracción y la herencia, a poder ir formando los componentes sin la necesidad de repetir constantes ni variables en sus modelos. Las librerías permiten crear sistemas de componentes de una manera mucho más sencilla que modelar directamente el sistema, e incluso como se explicó en el capítulo anterior, es posible hacerlo de una manera gráfica usando los símbolos previamente creados de los componentes, lo que es mucho más intuitivo y dinámico que la forma convencional.

La librería *Rotational Library* se estructura en subclases, en los que se han agrupado componentes con características dinámicas similares, Así se pueden encontrar las siguientes subclases; *ACTUATORS,* en el que se agrupan todos aquellos elementos diseñados con el propósito de excitar el sistema al que estén conectados; *CONSTRAINS*, en el que se encuentran aquellos elementos que imponen restricciones dinámicas a otros; *FRICTION*, donde se han ubicado todos los elementos necesarios para simular el fenómeno de la fricción así como aquellos que basan su funcionamiento en dicho fenómeno; *GEARS*, en el se encuentra la relación de transmisión mecánica; y *BEARING*, en el que se encuentra un modelo de un cojinete con lubricación hidrodinámica. Pero además de estos subgrupos existen otros tres que han sido creados en base a otros criterios, así el subgrupo *COMMON* agrupa todos los elementos abstractos que formarán la base de otros elementos; en el subgrupo *PORTS* se encuentran los puertos que se utilizan en el resto de componentes; y en el subgrupo *BASIC* se encuentran los elementos básicos del movimientos rotacional.

La figura 4.1 muestra un esquema con la estructura de la librería *Rotational Library*. Los elementos que aparecen rodeados por una línea roja, son elementos que pertenecen a librerías propias de EcosimPro y que se han incluido en la librería *Rotational Library*.

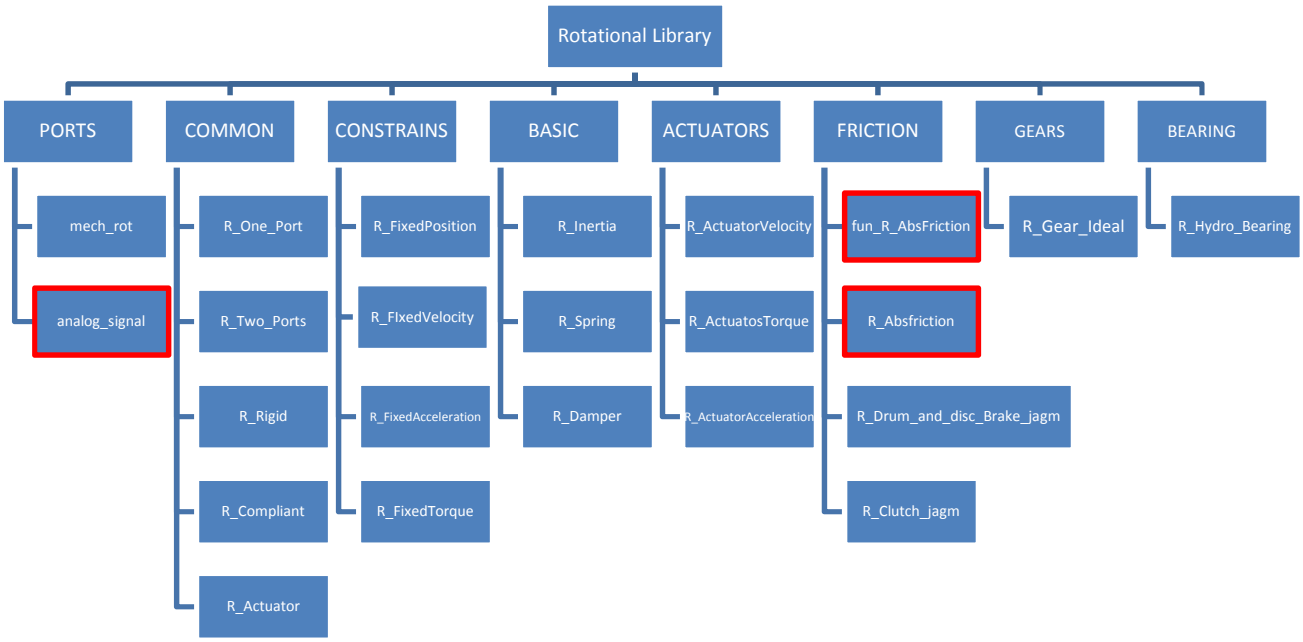


Fig. 4.1: Estructura librería *Rotational Library*.

En la tabla 4.1 se agrupan todos los componentes que forman la librería *Rotational Library*, además del tipo de componente y una breve descripción de las característica más importantes de cada uno de ellos.

Tabla 4.1: Estructura librería *Rotational Library*, incluyendo el tipo y características de los componentes que la integran.

Subgrupo	Nombre	Tipo	Descripción
PORTS	*mech_rot*	Puerto	Puerto mecánico rotacional.
	analog_signal		Puerto para señales analógicas.
COMMON	*R_One_Port*	Abstracto	Clase para elementos de 1 puerto.
	R_Two_Ports		Clase para elementos de 2 puertos.
	R_Rigid		Clase para elementos con dos puertos sin movimiento relativos entre ellos.
	R_Compliant		Clase para elementos con dos puertos con movimiento relativos entre ellos
	R_Actuator		Clase abstracta que define un actuador genérico rotacional.
CONSTRAINS	*R_FixedPosition*	Concreto	Elemento que impone una posición angular determinada.
	R_FixedVelocity		Elemento que impone una velocidad angular determinada.
	R_FixedAcceleration		Elemento que impone una aceleración angular determinada.
	R_FixedTorque		Elemento que impone un par determinado.
BASIC	*R_Inertia*	Concreto	Disco de inercia
	R_Spring		Muelle rotacional
	R_Damper		Amortiguador rotacional
ACTUATORS	*R_ActuatorVelocity*	Concreto	Actuador de velocidad
	R_ActuatorTorque		Actuador de par
	R_ActuatorAcceleration		Actuador de aceleración
FRICTION	*Fun_R_AbsFriction*	Función	Función residual para calcular implícitamente el parámetro de fricción *sa*[4].
	R_AbsFriction	Abstracto	Clase abstracta que representa la fricción de Coulomb.
	R_Drum_and_disc_Brake_jagm	Concreto	Elemento que modela frenos de disco y tambor
	R_Clutch_jagm		Elemento que modela embragues de fricción seca.
GEARS	*R_Gear_Ideal*	Concreto	Elemento que modela la relación de transmisión ideal.
BEARING	*R_Hydro_Bearing*	Concreto	Elemento que modela un cojinete con lubricación hidrodinámica.

[4] El parámetro *sa*, que se estudia en detalle en el sección 4.4.6, representa el parámetro de la curva parametrizada que representa un sistema dinámico con fricción de Coulomb, para el instante en el que la velocidad relativa de las superficies en contacto es nula.

4.2. Herencia en la librería "*Rotational library*"

En orientación a objetos la herencia es un mecanismo fundamental para lograr la reutilización del software y por tanto su extensibilidad. Como ya se explicó en apartados anteriores, el empleo de esta propiedad permite crear subclases a partir de superclases ya existentes, previamente comprobadas y verificadas.

La herencia es una de las propiedades de modelado orientado a objetos más utilizada por EcosimPro. Los componentes heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. Por tanto, la herencia permite a los componentes ser definidos y creados como tipos especializados de objetos preexistentes, y éstos pueden compartir y extender su comportamiento sin tener que volver a ser implementados.

En la construcción de la librería *Rotational Library* la herencia de clases abstractas ha sido una práctica habitual, debido a las ventajas que ello ofrece. En la figura 4.2 se representa un esquema que ilustra la herencia entre los componentes de la librería *Rotational Library*.

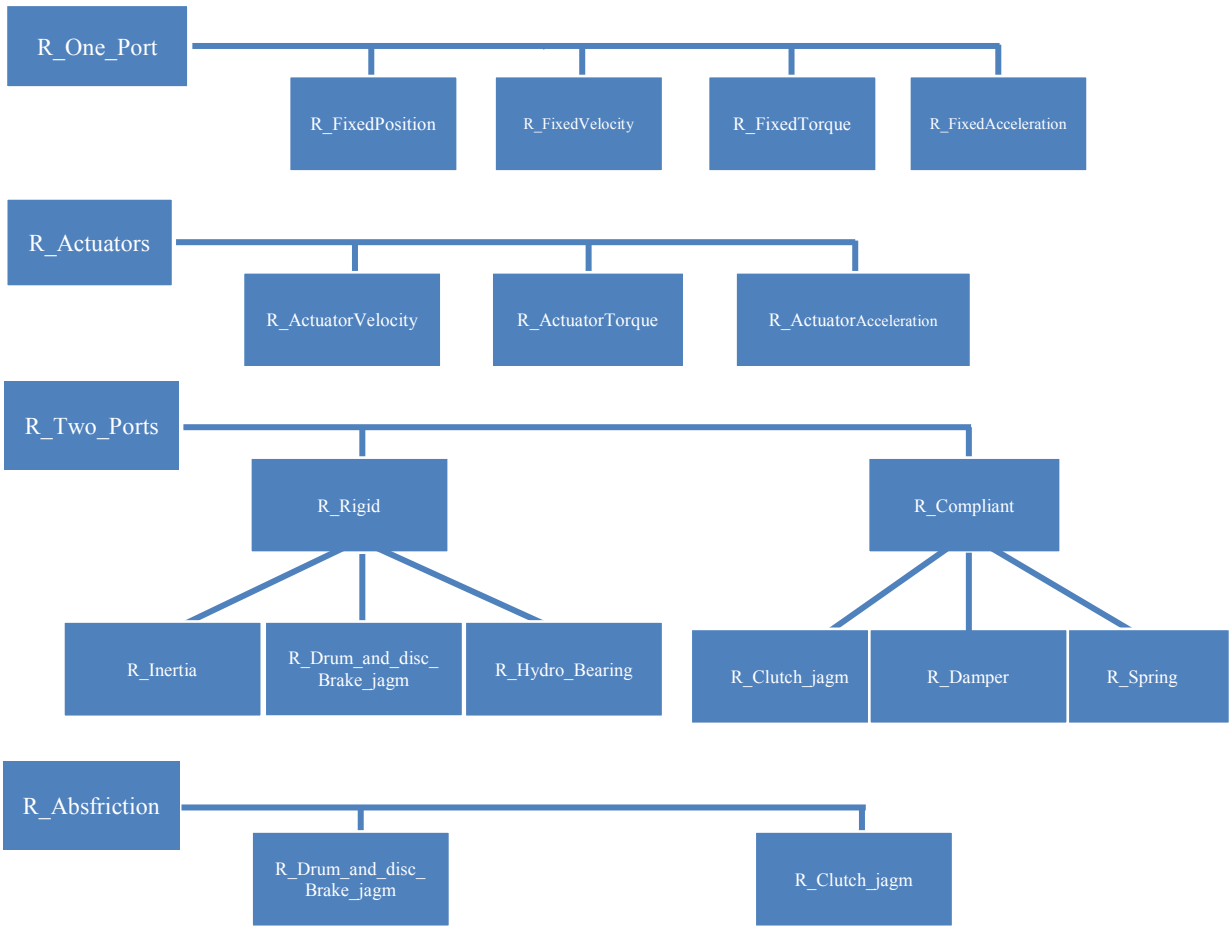


Fig. 4.2: Esquema ilustrativo de la herencia entre los componentes de *Rotational Library*.

El único elemento de la librería que no aparece en el esquema de la figura 4.2 es *R_Gear_Ideal*, ya es el único componente que no hace uso de la herencia.

En las siguientes líneas se analizarán en detalle las características de cada uno de los componentes, así como las motivaciones que han llevado a estructurar la herencia según se muestra en la figura 4.2.

4.3. Creación de puertos

El mecanismo básico para modelar un sistema de una forma modular consiste en conectar componentes usando sus puertos. Es decir, para poder unir elementos entre sí es necesario que éstos posean unos conectores llamados puertos, que fueron analizados en profundidad en el capítulo 3. La librería *Rotational Library*, consta de dos tipos de puertos, uno de naturaleza mecánica y otro que se utiliza para transmitir las señales emitidas por un componente generador de señales denominado *AnalogSource*.

4.3.1. Puerto *mech_rot*

La tabla 4.2 muestra la descripción y representación gráfica de este tipo de componentes.

Tabla 4.2: Descripción y representación gráfica de *mech_rot*.

Puerto	Descripción	Representación gráfica
mech_rot	El puerto de entrada o salida principal es un puerto mecánico rotacional	

Las variables que se definen en este puerto son se muestran en la tabla 4.3.

Tabla 4.3: Variables definidas en *mech_rot*.

Variable	Tipo	Descripción	Unidades
tau	REAL	Define el par o momento rotacional	N·m
phi	REAL	Define el ángulo de giro	rad

El código en lenguaje de programación EL que representa el modelo del puerto mecánico rotacional se muestra en la figura 4.3.

```
-------------------------------------------------------------------------------
-- Port mech_rot (Rotational port)
-------------------------------------------------------------------------------
PORT mech_rot "1D rotational flange"
   SUM   REAL tau   UNITS   "N*m"     "Torque (N*m)"
   EQUAL REAL phi   UNITS   "rad"     "Absolute angular position (rad)"


END PORT
```

Fig. 4.3: Código del puerto *mech_rot* en EL.

Se sabe que en los componentes que formarán la librería *Rotational Library*, la posición angular entre todos los componentes conectados entre sí será la misma. Esto se consigue mediante el modificador EQUAL. Asimismo, la suma de los momentos torsores que salen y entran de un nodo es la misma, indicándose esto por medio de la cláusula SUM. Entonces, si ahora se conectan tres componentes cualesquiera de la librería denominados *C1*, *C2* y *C3* mediante el puerto *mech_rot_p*, tal y como se muestra en la figura 4.4, se generan las siguientes ecuaciones:

$$C1.p.phi = C2.p.phi \quad (4.1)$$
$$C1.p.phi = C3.p.phi \quad (4.2)$$
$$C1.p.tau + C2.p.tau + C3.p.tau = 0 \quad (4.3)$$

Por tanto, mediante el puerto *mech_rot* y los operadores SUM y EQUAL se consigue el comportamiento deseado para conectar componentes de dinámica rotacional.

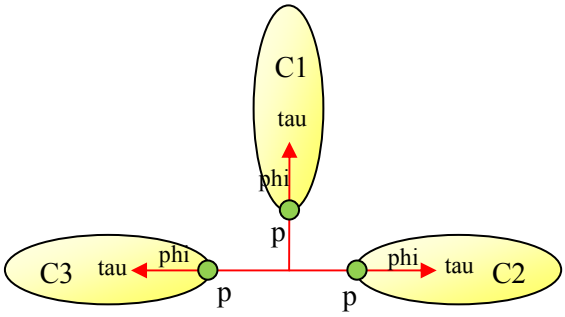


Fig. 4.4: Conexión de tres componentes cualesquiera de la librería denominados *C1*, *C2* y *C3* mediante el puerto *mech_rot_p*.

4.3.2. Puerto *analog_signal*

Para transmitir las señales emitidas por el componente generador de señales denominado *AnalogSource*, se ha utilizado el puerto *analog_signal*. La descripción y representación de este componente se muestra en la tabla 4.4. Además, su código, que ha sido tomado de las librerías que incluye EcosimPro, se presenta en la figura 4.5.

Tabla 4.4: Descripción y representación gráfica de *analog_signal*.

Puerto	Descripción	Representación gráfica
Analog_signal	Componente encargado de transmitir las señales analógicas emitidas por otros componentes.	●

```
-------------------------------------------------------------------------------
-- Ports analog_signal and bool_signal
-------------------------------------------------------------------------------

PORT analog_signal (INTEGER n = 1 "Number of outputs (-)")   SINGLE IN  "Analog signals 1D port"
   EQUAL OUT REAL signal[n]       "Analog signal values (-)"
END PORT
```

Fig. 4.5: Código del puerto *analog_signal* en EL.

4.4. Creación de componentes abstractos

Como se explicó en el apartado 3.5 del capítulo 3, los componentes abstractos describen un comportamiento que en sí mismo no representa ningún elemento físico, y que sólo puede ser usado como base para la creación de otros componentes. Es decir, para hacer más sencilla la forma de programar los componentes se crean una serie de clases abstractas las cuales no representan componentes físicos en sí, pero son usados para crear a partir de ellas, otros componentes concretos.

Estas clases abstractas no se crean de manera arbitraria, sino que surgen debido a que dos o más de los componentes que integrarán la librería poseen características comunes que pueden ser implementadas en estos elementos abstractos, de forma que ahorren numerosas líneas de programa.

Dentro de la librería Rotational Library, existen seis clases abstractas. Las clases abstractas ***R_One_Port*** *y* ***R_Two_Ports***, se utilizan para definir componentes que se construyen con uno y dos puertos respectivamente, la clase ***R_Rigid***, se emplea para reproducir las características de los componentes con comportamiento de sólido rígido, la clase ***R_Compliant***, que define la conexión entre dos puertos rotacionales cuando puede existir movimiento relativo entre ellos, la clase ***R_Actuator,*** reúne las características comunes de los actuadores y la clase ***R_AbsFriction***, que define la fricción de Coulomb para componentes de dinámica rotacional.

4.4.1. Componente *R_One_Port*

Todos los componentes que precisen únicamente de un puerto mecánico rotacional, podrán heredar las características de esta clase abstracta, formada por un solo puerto, cuya descripción se muestra en la tabla 4.5.

Tabla 4.5: Descripción de *R_One_Port.*

Nombre	Descripción
R_One_Port	Componente abstracto que define elementos con un sólo puerto mecánico rotacional

Los puertos de este elemento se presentan en la tabla 4.6.

Tabla 4.6: Descripción de los puertos de *R_One_Port.*

Puerto	Tipo	Dirección
m_out	*mech_rot*	OUT

El código en lenguaje de programación EL que representa el modelo del componente abstracto se muestra en la figura 4.6.

```
-------------------------------------------------------------------------------
-- Component R_One_Port
-------------------------------------------------------------------------------
ABSTRACT COMPONENT R_One_Port
"Abstract class for definition of components with one rotational port"
   PORTS
      OUT mech_rot m_out "Right / driven mech_rot"

END COMPONENT
```

Fig. 4.6: Código del componente *R_One_Port* en EL.

4.4.2. Componente *R_Two_Ports*

Esta clase abstracta será la base de todos aquellos elementos que se estructuren con dos conectores. Su descripción se muestra en la tabla 4.7.

Tabla 4.7: Descripción de *R_Two_Ports.*

Nombre	Descripción
R_Two_Port	Componente abstracto que define elementos con dos puertos mecánicos rotacionales

Este componente contiene dos puertos, uno de entrada (*m_in*) y otro de salida (*m_out*), cuya descripción se presenta en la tabla 4.8.

Tabla 4.8: Descripción de los puertos de *R_Two_Ports.*

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

El código en lenguaje de programación EL para este componente abstracto se representa en la figura 4.7.

```
-------------------------------------------------------------------------------
-- Component R_Two_Ports
-------------------------------------------------------------------------------
ABSTRACT COMPONENT R_Two_Ports
"Abstract class for definition of components with two rotational ports"
   PORTS
      IN  mech_rot m_in     "Left / driving mech_rot"
      OUT mech_rot m_out    "Right / driven mech_rot"

END COMPONENT
```

Fig. 4.7: Código del componente *R_Two_Ports* en EL.

4.4.3. Componente *R_Rigid*

Los componentes que presenten un comportamiento de sólido rígido, es decir, aquellos que estén formados por un conjunto de puntos que se mueven de tal manera que no se alteran las distancias entre ellos sea cual sea la fuerza actuante, deben heredar todas las propiedades del componente abstracto *R_Rigid*, cuya descripción se muestra en la tabla 4.9.

Tabla 4.9: Descripción de *R_Rigid.*

Nombre	Descripción
R_Rigid	Componente abstracto que define elementos con dos puertos mecánicos rotacionales sin movimiento relativo entre ellos

Ya que el componente *R_Rigid* es un componente que posee dos conectores, uno de entrada y otro de salida, puede heredar las propiedades de R_Two_Ports. Los puertos de *R_Rigid* se presentan en la tabla 4.10.

Tabla 4.10: Descripción de los puertos de *R_Rigid.*

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

El ángulo de giro absoluto *phi*, es la variable que se usa para definir el comportamiento de este tipo de componentes. Las variables que introduce el elemento *R_Rigid* se presentan en la tabla 4.11.

Tabla 4.11: Variables que se introducen con *R_Rigid.*

Variable	Tipo	Descripción	Unidades
phi	REAL	Define el ángulo de giro absoluto	rad

Como se debe modelar el comportamiento de un sólido rígido, el giro absoluto de los conectores del componente debe ser el mismo, es decir:

$$m\_in.phi = m\_out.phi \tag{4.4}$$

El código en lenguaje de programación EL del componente abstracto *R_Rigid* se representa en la figura 4.8. Es importante destacar la función de la cláusula *IS_A*, mediante la que se consigue que el componente herede las características de *R_Two_Ports*.

```
-------------------------------------------------------------------------------
-- Component R_Rigid
-------------------------------------------------------------------------------

ABSTRACT COMPONENT R_Rigid IS_A R_Two_Ports
"Abstract class for the definition of a rigid connection of two rotational ports"
   DECLS
      REAL phi   UNITS "rad"     "Absolute angular position (rad)"

   CONTINUOUS
      m_in.phi = phi
      m_out.phi = phi
END COMPONENT
```

Fig. 4.8: Código del componente *R_Rigid* en EL.

4.4.4. Componente *R_Compliant*

Este componente, cuya descripción se muestra en la tabla 4.12, se utiliza para modelar aquellos elementos que poseen dos conectores rotacionales con movimiento relativo. Además el valor del par en ambos conectores (ó puertos) debe ser el mismo.

Al igual que ocurría con *R_Rigid*, en este caso también se heredan las propiedades de *R_Two_Ports* vía *IS_A,* ya que esta clase abstracta también posee dos conectores.

Tabla 4.12: Descripción de *R_Compliant.*

Nombre	Descripción
R_Compliant	Componente abstracto que define elementos con dos puertos mecánicos rotacionales con movimiento relativo entre ellos

Los puertos que posee este elemento se presentan en la tabla 4.13.

Tabla 4.13: Descripción de los puertos de *R_Compliant.*

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Para la definición de R_Compliant, es necesario introducir las variables ángulo de giro relativo *phi_rel* y par *tau*, así como la constante giro relativo inicial *phi_rel_0* (Tablas 4.14 y 4.15).

Tabla 4.14: Constantes que se introducen con *R_Compliant.*

Dato	Tipo	Descripción	Valor inicial	Unidades
phi_rel_0	REAL	Define la distancia angular inicial entre los dos puertos	0	rad

Tabla 4.15: Variables que se introducen con *R_Compliant.*

Variable	Tipo	Descripción	Unidades
tau	REAL	Define el par o momento rotacional	N·m
phi_rel	REAL	Define el ángulo de giro relativo	rad

Hay ocasiones en las cuales para poder obtener el modelo de algún componente es necesario dar un valor inicial a alguna de sus variables. Esto ocurre cuando hay más incógnitas que ecuaciones y resulta imposible hallar todas estas incógnitas. Cuando esto sucede es necesario inicializar alguna de las variables con una ecuación de contorno o asignarle un valor inicial. Las ecuaciones de contorno definen la forma en la que actúa el elemento en un momento dado. Para el caso de R_Compliant se introduce la siguiente condición de contorno:

$$phi\_rel = phi\_rel\_0 \tag{4.5}$$

Esta ecuación expresa que en el estado inicial el ángulo de giro relativo coincide con el ángulo de giro absoluto.

Para definir el giro relativo entre los dos puertos se utiliza la ecuación 4.6, en la que se calcula el valor de la variable *phi_rel*. Mientras que la ecuación 4.7 indica que el valor del par *tau*, ha de ser el mismo en los dos extremos del componente.

$$phi\_rel = m\_out.phi - m\_in.phi \tag{4.6}$$

$$m\_in.tau = m\_out.tau \tag{4.7}$$

El código en lenguaje de programación EL que representa el componente abstracto *R_Compliant* se representa en la figura 4.9.

```
-------------------------------------------------------------------------------
-- Component R_Compliant
-------------------------------------------------------------------------------

ABSTRACT COMPONENT R_Compliant IS_A R_Two_Ports
"Abstract class for definition of a compliant connection of two rotational ports"
   DATA
      REAL phi_rel_0 = 0 UNITS "rad"          "Initial angular distance between ports (rad)"

   DECLS
      REAL tau           UNITS "N*m"           "Rotational transmited torque (N*m)"
      REAL phi_rel       UNITS "rad"           "Angular distance between ports (rad)"

   INIT
   phi_rel = phi_rel_0

   CONTINUOUS
      m_in.tau = tau
      m_out.tau = tau

      phi_rel = m_out.phi - m_in.phi

END COMPONENT
```

Fig. 4.9: Código del componente *R_Compliant* en EL.

4.4.5. Componente *R_Actuator*

Este componente, cuya descripción se muestra en la tabla 4.16, constituye una clase abstracta que se utiliza para definir actuadores rotacionales genéricos. Está formado por dos puertos, uno de entrada, de tipo *analog_signal*, y otro de salida de tipo *mech_rot* (Tabla 4.17).

Tabla 4.16: Descripción de *R_Actuator*.

Nombre	Descripción
R_Actuator	Clase abstracta que se utiliza para definir actuadores rotacionales genéricos.

Tabla 4.17: Descripción de los puertos de *R_Actuator*.

Puerto	Tipo	Dirección
s_in	*Analog_signal*	IN
m_out	*mech_rot*	OUT

El código en lenguaje de programación EL que representa el componente abstracto *R_Actuator* se muestra en la figura 4.10.

```
-------------------------------------------------------------------------------
-- Component R_Actuator
-------------------------------------------------------------------------------

ABSTRACT COMPONENT R_Actuator
"Abstract class for definition of generic rotational actuators"
   PORTS
      IN  analog_signal s_in     "Input signal port"
      OUT mech_rot m_out      "Outlet rotational mechanical port"

END COMPONENT
```

Fig. 4.10: Código del componente *R_Actuator* en EL.

4.4.6. Componente *R_AbsFriciton*

Mediante el componente *R_AbsFriction* se consigue modelar el fenómeno de la fricción. Éste fenómeno es vital para la simulación de sistemas mecánicos, ya que gracias a su existencia se puede explicar el comportamiento de aquellos dispositivos con movimiento relativo entre superficies que están en contacto. La fricción presenta un marcado comportamiento discontinuo, especialmente para bajas velocidades relativas entre las superficies en contacto, lo que complica bastante su modelado [OTT99].

Se han desarrollado multitud de modelos con el propósito de modelar la fricción, aunque se pueden clasificar en dos grandes grupos: los de tipo discontinuo y los de tipo continuo. En los modelos de tipo discontinuo, la fricción (par de fricción) es discontinua para velocidad nula, y cuando esto ocurre (modo estático), el par de fricción actúa equilibrando el resto de los pares aplicados al sistema para mantener la velocidad nula. Por otro lado los modelos continuos consideran pequeños desplazamientos elásticos (desplazamientos de pre-deslizamiento) en el modo estático.

Para modelar la fricción en la librería *Rotational Library* se ha elegido un modelo de tipo discontinuo, debido a la intuitiva simplicidad que éstos presentan. A pesar de esta ventaja, los modelos discontinuos siguen resultando incómodos ya que su definición para velocidad nula y no nula es totalmente distinta. Una forma de afrontar este inconveniente consiste en utilizar un umbral de velocidad que funcione como frontera entre las regiones de velocidad nula y no nula. Sin embargo, la elección de este umbral tiene una gran influencia en el comportamiento de los modelos, y una mala elección del mismo puede dar lugar a comportamientos no deseados de estos. Otra forma de afrontar el problema de la discontinuidad, y que es la que utiliza el componente *R_AbsFriction*, consiste en emplear una máquina de estados finitos basada en la detección de cambios de signo de la velocidad [KIK05].

En definitiva, el componente *R_AbsFriction* está basado un modelo de tipo híbrido, cuyas características fueron analizadas en el apartado 2.1.4 del capítulo 2, y dispone de una máquina de estados finitos que detecta los cambios de signo en la velocidad.

Fig. 4.11: Sistema con fricción de Coulomb.

La figura 4.11 representa un eje que gira a una velocidad angular *w(t)* debido a la acción del par *u(t).* El par de fricción *tau_friction,* actúa entre las superficies del eje y su

alojamiento, y es una función lineal de la velocidad angular relativa *w_relfric* existente entre ellas, cuando deslizan una sobre otra. Cuando la velocidad relativa se vuelve nula, las dos superficies se encuentran adheridas y el par de fricción ya no es una función de *w_relfric*. El elemento comienza a girar de nuevo si el par *u(t)* supera el valor del par de fricción estático *tau0*. El sistema de la figura 4.11, que representa el caso más simple de fricción en elementos con movimiento rotacional, se puede representar mediante una curva parametrizada[5] como la de la figura 4.12, de acuerdo con las ecuaciones (4.8) y (4.9), donde además de los términos *w_relfric, tau_friction* y *tau0,* que fueron introducidos con anterioridad aparece el término *s*, que representa el parámetro de la curva y *tau_pos*, que representa el par de fricción dinámico.

$$w\_relfric = \begin{cases} s-1 & si \quad s>1 \\ s+1 & si \quad s<-1 \\ 0 & resto \end{cases} \tag{4.8}$$

$$tau\_friction = \begin{cases} tau0 + tau\_pos*(s-1) & si \quad s>1 \\ -tau0 + tau\_pos*(s+1) & si \quad s<-1 \\ tau0*s & resto \end{cases} \tag{4.9}$$

Mediante el parámetro *s*, se consigue parametrizar el comportamiento del sistema de la figura 4.11 y así, las variables del sistema ya no dependen del tiempo sino de *s*.

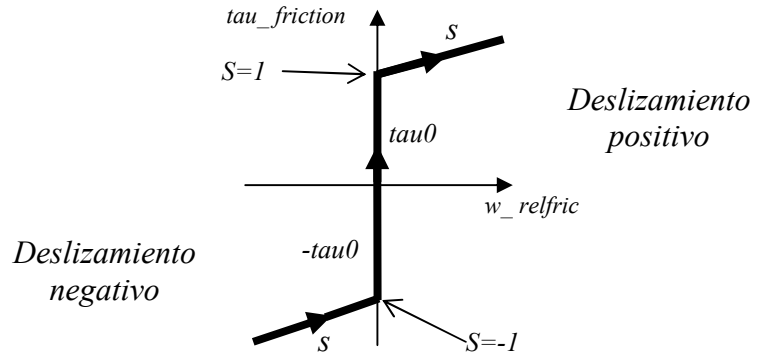


Fig. 4.12: Curva parametrizada[6] de un sistema con fricción de Coulomb.

[5]la parametrización consiste en la caracterización de un sistema (generalmente discontinuo) a través de una curva dependiente de un parámetro. Esta caracterización describe el sistema de una forma más general, lo que permite describir el sistema de un modo declarativo.

[6] en función del valor del parámetro *s* de la curva parametrizada de la figura 4.12, se pueden tener tres modos de operación distintos. El modo estático, para *-1≤s≤1*, el modo de deslizamiento positivo, para *s>1*, y el modo de deslizamiento negativo, para *s<-1*.

El modelo matemático implementado por las ecuaciones (4.8) y (4.9), describe por completo el fenómeno de la fricción de una forma declarativa, en un sistema muy simplificado. Por desgracia, debido a la dificultad que entraña definir con precisión la región de velocidad nula y sus proximidades, resulta complicado transformar el modelo de las ecuaciones (4.8) y (4.9) en otro que pueda ser simulado. Para comprender estas dificultades se analizará el sistema formado por un eje que gira en un alojamiento de la figura 4.11, cuyo comportamiento puede caracterizarse mediante la siguiente ecuación:

$$I \cdot \dot{w}\_relfric = u(t) - tau\_friction \tag{4.10}$$

Donde *I* es la inercia del eje y *u(t)* es el par conductor, que es un dato. Si el elemento se encuentra en el modo *deslizamiento positivo*, entonces $s \geq 1$, y por lo tanto el modelo queda descrito por las ecuaciones (4.10), (4.11) y (4.12), y se puede transformar fácilmente en un espacio de estados, con *w_relfric* como variable de estado.

$$tau\_friction = tau0 - tau\_pos \cdot (s - 1) \tag{4.11}$$

$$w\_relfric = s - 1 \tag{4.12}$$

Así, sí el eje se detiene, entonces, $-1 \leq s \leq 1$ y el valor *w_relfric* pasa a ser nulo, por lo tanto *w_relfric* no puede ser un estado, lo que origina un cambio de índice de las ecuaciones diferenciales. Pero junto con la dificultad de manejar los cambios en las variables de estado, existe otro problema más serio: si se supone que el eje se encuentra estático y el valor del parámetro *s* se vuelve mayor que la unidad. Justo antes de que esto ocurra se tienen las siguientes condiciones:

$$s \leq 1 \tag{4.13}$$

$$w\_relfric = 0 \tag{4.14}$$

Pero desde el momento en que se cumple que $s>1$(ver figura 4.12) el sistema pasa al modo de *deslizamiento positivo* donde *w_relfric* es una variable de estado, que es inicializada con su último valor, esto es, $w\_relfric=0$, y *s* es calculada a partir de ella según la ecuación (4.12), de modo que se obtiene un valor para este parámetro igual a la unidad, resultando así la relación $s>1$ falsa, lo que provoca que el sistema vuelva a pasar al *modo estático*. En otras palabras, nunca será posible pasar al modo de *deslizamiento positivo*.

La clave para solucionar este problema consiste en observar que *w_relfric* es nula en el *modo estático* y al inicio del modo de *deslizamiento positivo*, pero sin embargo, la aceleración relativa, $\dot{w}\_relfric$, es mayor que cero cuando comienza el *deslizamiento* y nula en el *modo estático*. De manera que para el instante en el que *w_relfric* es nula, se puede obtener una nueva curva parametrizada como la de la figura 4.13, que necesita de un nuevo parámetro s_a para satisfacer las ecuaciones siguientes:

$$a\_relfric = der(w\_relfric) \tag{4.15}$$

$$tau\_friction = \begin{cases} tau0 & si \quad s_a > 1 \\ -tau0 & si \quad s_a < -1 \\ tau0 \cdot s_a & resto \end{cases} \tag{4.16}$$

$$a\_relfric = \begin{cases} s_a - 1 & si \quad s_a > 1 \\ s_a + 1 & si \quad s_a < -1 \\ 0 & resto \end{cases} \tag{4.17}$$

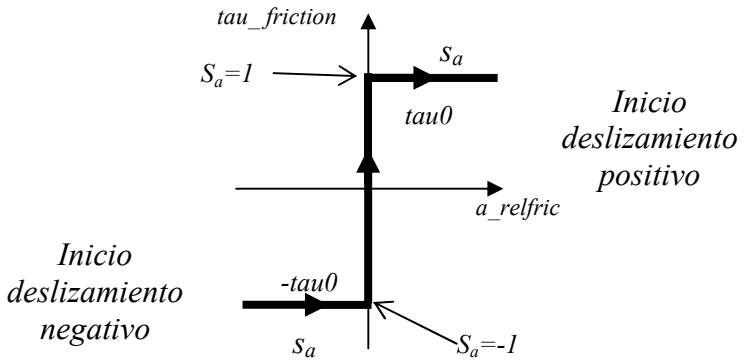


Fig. 4.13: Curva parametrizada de un sistema con fricción de Coulomb, para w_relfric=0.

Entonces para *w_relfric=0*, las ecuaciones (4.10), (4.15), (4.16) y (4.17) forman un conjunto de ecuaciones continuas/discretas que tienen que ser resueltas durante todos los instantes de tiempo, con lo que la velocidad *w_relfric* vuelve a ser una variable de estado incluso cuando es nula.

Consecuentemente, si se pasa del modo estático a cualquiera de los modos de deslizamiento, la velocidad angular relativa será muy pequeña, pero ha de tener algún signo, y será precisamente este signo el que permita conocer si el sistema desliza o no, y si lo hace en sentido positivo o negativo[7].

Por tanto, para modelar el fenómeno de la fricción, el componente *R_AbsFriction* se basa en la curva parametrizada de la figura 4.14, que es idéntica a la de la figura 4.13 salvo por la existencia de un salto en el par de fricción cuando se inicia el deslizamiento. Este salto, denominado *peak,* se declarará dentro de los elementos concretos que hereden de *R_AbsFriction,* cuya descripción se muestra en la tabla 4.18.

[7] La regla ó ley de la mano derecha o del sacacorchos será el criterio para establecer el signo de las magnitudes vectoriales

Tabla 4.18: Descripción de *R_AbsFriction.*

Nombre	Descripción
R_AbsFriction	Clase abstracta que se utiliza para fricción de Coulomb en componente con movimiento rotacional.

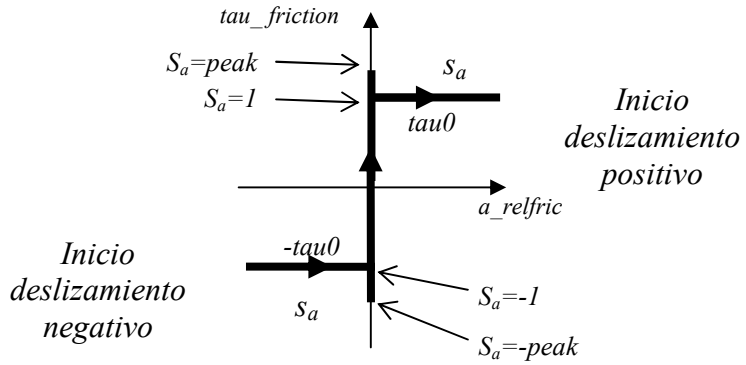


Figura 4.14: Curva parametrizada para *R_AbsFriction.*

En la tabla 4.19 se describen las variables declaradas en componente *R_Absfriction*.

Tabla 4.19: Descripción de las variables declaradas en *R_AbsFriction.*

Variable	Tipo	Descripción	Unidades
free	BOOLEAN	Verdadera, si el elemento fricción no está activado, falsa cuando existe fricción.	-
w_relfric	REAL	Define la velocidad angular relativa entre las superficies en contacto.	rad/s
a_relfric	REAL	Define la aceleración angular relativa entre las superficies en contacto.	rad/s^2
tau_friction	REAL	Par de fricción resultante. Se considera positivo si se opone a la velocidad angular relativa.	N·m
tau0	REAL	Par de fricción para *w_relfric=0* para modo de inicio del deslizamiento (*imode 1* ó *imode -1*).	N·m
tau0_max	REAL	Par de fricción estático para *w_relfric=0* (*imode 0*).Depende del valor de *peak*.	N·m
sa	REAL	Parámetro de la curva parametrizada para la fricción.	-
imode	INTEGER	Modos de operación. Valor: 0 — Tipo de deslizamiento: Estático (w_relfric=a_relfric=0) 1 — Inicio deslizamiento positivo (w_relfric=0 y a_relfric>0) -1 — Inicio deslizamiento negativo (w_relfric=0 y a_relfric<0) 2 — Deslizamiento positivo (w_relfric>0) -2 — Deslizamiento negativo (w_relfric<0)	-

El código en lenguaje de programación EL que representa el componente abstracto *R_ AbsFriction* se representa en la figura 4.15.

```
-------------------------------------------------------------------------------
-- Component R_AbsFriction
-------------------------------------------------------------------------------

ABSTRACT COMPONENT R_AbsFriction
"Abstract component of rotational coulomb friction components "
   DECLS

   BOOLEAN free = FALSE      "TRUE, if frictional element is not active (-)"
   REAL w_relfric            "Relative angular velocity between frictional surfaces (rad/s)"
   REAL a_relfric            "Relative angular acceleration between frictional surfaces (rad/s^2)"
   REAL tau_friction         "Friction torque: positive, if directed in opposite direction of w_rel (N*m)"
   REAL tau0                 "Friction torque for w=0 and forward sliding (N*m)"
   REAL tau0_max             "Maximum friction torque for w=0 and locked (N*m)"
   REAL sa                   "Path parameter of friction characteristic (-)"
   INTEGER imode = -2        "Operation mode (-)"

   DISCRETE

   WHEN(free) THEN
      imode = 3 AFTER 0.0
   END WHEN


   WHEN (imode == 1 AND w_relfric > 0) THEN
      imode = 2 AFTER 0.0
   END WHEN

   WHEN (imode == 2 AND w_relfric <= 0) THEN
      imode = 0 AFTER 0.0
   END WHEN

   WHEN (imode == -1 AND w_relfric < 0) THEN
      imode = -2 AFTER 0.0
   END WHEN
   WHEN (imode == 2 AND w_relfric <= 0) THEN
      imode = 0 AFTER 0.0
   END WHEN

   WHEN (imode == -1 AND w_relfric < 0) THEN
      imode = -2 AFTER 0.0
   END WHEN

   WHEN (imode == -2 AND w_relfric >= 0) THEN
      imode = 0 AFTER 0.0
   END WHEN

   WHEN (imode == 0 AND sa > tau0_max) THEN
      imode = 1 AFTER 0.0
   END WHEN

   WHEN (imode == 0 AND sa < -tau0_max) THEN
      imode = -1 AFTER 0.0
   END WHEN


   CONTINUOUS

   fun_R_AbsFriction(imode, a_relfric, sa, tau0) = 0

END COMPONENT
```

Figura 4.15: Código del componente *R_AbsFriction* en EL.

Como se puede observar, el código de la figura 4.15, además de declarar variables que ya se describen en la tabla 4.19, introduce una máquina de estados como la de la figura

4.16 para controlar el valor del par de fricción durante los distintos modos de deslizamiento.

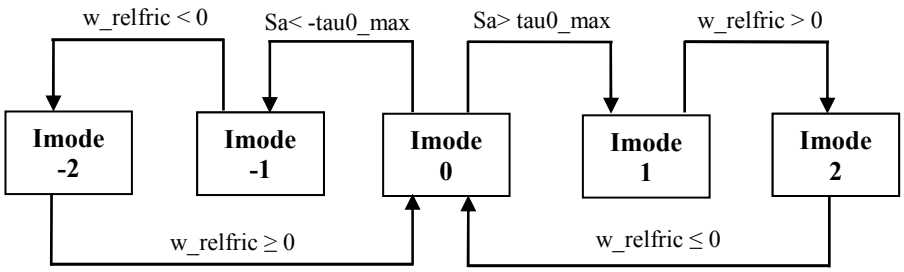


Figura 4.16: Máquina de estados de *R_AbsFriction*.

Además en el *R_AbsFrcition* aparece una función, denominada *fun_R_AbsFriction* cuyo código aparece en la figura 4.17. Se trata de una función residual creada para calcular implícitamente el valor del parámetro s_a.

```
-------------------------------------------------------------------------------
-- Function fun_R_AbsFriction
-------------------------------------------------------------------------------

FUNCTION REAL fun_R_AbsFriction(
   INTEGER imode  "Operation mode",
   REAL a_relfric "Relative angular acceleration (rad/s^2)",
   REAL sa      "Path parameter of friction characteristic (-)",
   REAL tau0      "Friction torque for w=0 (N*m)")
"Residue function to calculate implicitly the path parameter of friction \
characteristic of the components that inherit from the abstract component R_AbsFriction"

DECLS
   REAL resul
BODY
   IF(imode == 0) THEN
      resul = a_relfric
   ELSEIF(imode == 1) THEN
      resul = a_relfric - (sa - tau0)
   ELSEIF(imode == -1) THEN
      resul = a_relfric - (sa + tau0)
   ELSEIF(imode == 2) THEN
      resul = a_relfric - (sa - tau0)
   ELSEIF(imode == -2) THEN
      resul = a_relfric - (sa + tau0)
   ELSE
      resul = a_relfric - sa
   END IF

   RETURN resul

END FUNCTION
```

Figura 4.17: Código de la función *fun_R_AbsFriction* en EL.

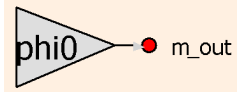
4.5. Creación de componentes de la subclase *constrains*

Dentro de esta subclase se encuentran aquellos elementos que imponen restricciones dinámicas a otros elementos o a una parte del sistema al que pertenezcan.

4.5.1. Componente *R_FixedPosition*

Este componente hereda de *R_One_port,* por lo tanto toma todas las propiedades de esta clase abstracta. La descripción de *R_FixedPosition* se muestra en la tabla 4.20.

Tabla 4.20: Descripción de *R_FixedPosition.*

Nombre	Descripción	Representación gráfica
R_FixedPosition	Define un elemento que fija la posición angular a la salida del puerto	

Los puertos y constantes del elemento *R_FixedPosition* se presentan en las tablas 4.21 y 4.22 respectivamente.

Tabla 4.21: Descripción de los puertos de *R_FixedPosition.*

Puerto	Tipo	Dirección
m_out	*mech_rot*	OUT

Tabla 4.22: Descripción de los datos declarados en *R_FixedPosition.*

Dato	Tipo	Descripción	Valor inicial	Unidades
phi_0	REAL	Define la posición angular fija del puerto	0	rad

Este elemento impone una posición angular fija e igual al valor del parámetro *phi_0*. Así la ecuación que caracteriza el comportamiento de *R_FixedPosition* es:

$$m\_out.phi = phi\_0 \tag{4.18}$$

El código en lenguaje de programación EL que representa el componente se muestra en la figura 4.18.

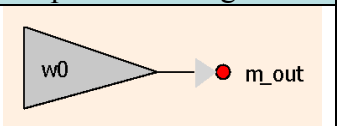
```
-------------------------------------------------------------------------------
-- Component R_FixedPosition
-------------------------------------------------------------------------------

COMPONENT R_FixedPosition IS_A R_One_Port
"Fixed angular position"

   DATA
      REAL phi_0 = 0 UNITS "rad"    "Fixed offset position (rad)"

   CONTINUOUS
      m_out.phi = phi_0

END COMPONENT
```

Figura 4.18: Código del componente *R_FixedPosition* en EL.

4.5.2. Componente *R_FixedVelocity*

Este componente hereda de *R_One_port,* por lo tanto toma todas las propiedades de esta clase abstracta. Su descripción se muestra en la tabla 4.23.

Tabla 4.23: Descripción de *R_FixedVelocity.*

Nombre	Descripción	Representación gráfica
R_FixedVelocity	Define un elemento que fija la velocidad angular a la salida del puerto	

Los puertos y contantes que introduce este elemento se presentan en las tablas 4.24 y 4.25 respectivamente.

Tabla 4.24: Descripción de los puertos de *R_FixedVelocity.*

Puerto	Tipo	Dirección
m_out	*mech_rot*	OUT

Tabla 4.25: Descripción de los datos declarados en *R_FixedVelocity.*

Dato	Tipo	Descripción	Valor inicial	Unidades
w0	REAL	Define la velocidad angular fija en el puerto	0	rad/s

Este elemento impone una velocidad angular constante e igual al valor del parámetro *w0*. La ecuación que caracteriza el comportamiento de *R_FixedVelocity* es:

$$m\_out.phi' = w0 \tag{4.19}$$

El código en lenguaje de programación EL que representa el componente se muestra en la figura 4.19.

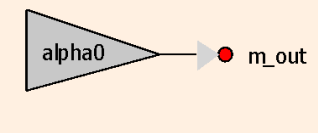
```
-------------------------------------------------------------------------------
-- Component R_FixedVelocity
-------------------------------------------------------------------------------

COMPONENT R_FixedVelocity IS_A R_One_Port
"Fixed angular velocity"

   DATA
      REAL w0 = 0.      "Fixed angular velocity (rad/s)"

   CONTINUOUS
      m_out.phi' = w0

END COMPONENT
```

Figura 4.19: Código del componente *R_FixedVelocity* en EL.

4.5.3. Componente *R_FixedAcceleration*

Este componente hereda de *R_One_port,* por lo tanto toma todas las propiedades de esta clase abstracta. Su descripción se muestra en la tabla 4.26.

Tabla 4.26: Descripción de *R_FixedAcceleration.*

Nombre	Descripción	Representación gráfica
R_FixedAcceleration	Define un elemento que fija la aceleración angular a la salida del puerto	

Los puertos y contantes que introduce este elemento se presentan en las tablas 4.27 y 4.28 respectivamente.

Tabla 4.27: Descripción de los puertos de *R_FixedAcceleration.*

Puerto	Tipo	Dirección
m_out	*mech_rot*	OUT

Tabla 4.28: Descripción de los datos declarados en *R_FixedAcceleration.*

Dato	Tipo	Descripción	Valor inicial	Unidades
alpha_0	REAL	Define la aceleración angular fija en el puerto	0	rad/s^2

Este elemento impone una aceleración angular constante e igual al valor del parámetro *alpha_0*. La ecuación que caracteriza el comportamiento de *R_FixedAcceleration* es:

$$m\_out.phi'' = alpha\_0 \qquad (4.20)$$

El código en lenguaje de programación EL que representa el componente se muestra en la figura 4.20.

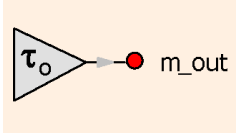
```
-------------------------------------------------------------------------------
-- Component R_FixedAcceleration
-------------------------------------------------------------------------------

COMPONENT R_FixedAcceleration IS_A R_One_Port
"Fixed angular acceleration"
   DATA
      REAL alpha_0 = 0. UNITS "rad/s^2"   "Fixed angular acceleration value (rad/s^2)"


   CONTINUOUS

      m_out.phi'' = alpha_0

END COMPONENT
```

Figura 4.20: Código del componente *R_FixedAcceleration* en EL.

4.5.4. Componente *R_FixedTorque*

Este componente también hereda de *R_One_port,* por lo tanto toma todas las propiedades de esta clase abstracta. Su descripción se muestra en la tabla 4.29.

Tabla 4.29: Descripción de *R_FixedTorque.*

Nombre	Descripción	Representación gráfica
R_FixedTorque	Define un elemento que fija el par rotacional a la salida del puerto	

Los puertos y contantes que introduce este elemento se presentan en las tablas 4.30 y 4.31 respectivamente.

Tabla 4.30: Descripción de los puertos de *R_FixedTorque.*

Puerto	Tipo	Dirección
m_out	*mech_rot*	OUT

Tabla 4.31: Descripción de los datos declarados en *R_FixedTorque.*

Dato	Tipo	Descripción	Valor inicial	Unidades
tau_0	REAL	Define el par fijo del puerto	0	N·m

Este elemento impone un par constante e igual al valor del parámetro *tau_0*. La ecuación que caracteriza el comportamiento de *R_FixedTorque* es:

$$m\_out.tau = tau\_0 \tag{4.21}$$

El código en lenguaje de programación EL que representa el componente se muestra en la figura 4.21.

```
-------------------------------------------------------------------------------
-- Component R_FixedTorque
-------------------------------------------------------------------------------

COMPONENT R_FixedTorque IS_A R_One_Port
"Fixed torque"

   DATA
      REAL tau_0 = 0  UNITS "N*m"              "Fixed torque value (N*m)"

   CONTINUOUS
      m_out.tau = tau_0

END COMPONENT
```

Figura 4.21: Código del componente *R_FixedTorque* en EL.

4.6. Creación de componentes de la subclase *basic*

Una gran variedad de sistemas mecánicos se basan en movimientos de rotación alrededor de un eje fijo o no acelerado, entendiendo este movimiento de rotación como aquel en el que todas las partículas que componen el cuerpo en rotación describen trayectorias circulares en torno a este eje.

Por suerte, todos estos sistemas mecánicos, desde una simple manivela, pasando por una bomba centrífuga y hasta el mecanismo más complejo que se pueda imaginar, se pueden modelar, en parte, mediante el uso de tres elementos básicos de la mecánica rotacional. Estos elementos, que integran la subclase *BASIC* de la librería *ROTATIONAL LIBRARY*, son la masa rotacional ó inercia, el muelle de torsión y el amortiguador de torsión, y sus características más importantes se analizan a continuación.

4.6.1. Componente *R_Inertia*

Mediante este elemento se modela la inercia de cualquier cuerpo en rotación, esto es, la resistencia que este opone a modificar su estado, ya sea de reposo o movimiento. La inercia de un cuerpo (en dinámica rotacional) se mide a través de su momento de inercia.

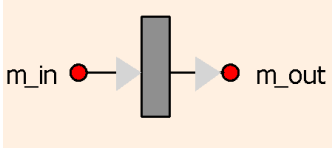
El momento de inercia es una magnitud que refleja la distribución de masa de un cuerpo o de un sistema de partículas en rotación, respecto a un eje de giro. El momento de inercia sólo depende de la geometría del cuerpo y de la posición del eje de giro; pero no depende de las fuerzas que intervienen en el movimiento.

Usando la *segunda ley de Newton* se puede establecer una relación entre la inercia del sólido en rotación, la aceleración angular y el sumatorio de los momentos externos que actúan sobre sólido.

$$I \cdot \alpha = \sum tau \tag{4.22}$$

La ecuación (4.22) será la que caracterice el comportamiento del componente *R_Inertia*, el cual además, heredará todas las propiedades y características del componente *R_Rigid*. La descripción de este elemento se muestra en la tabla 4.32.

Tabla 4.32: Descripción *R_Inertia.*

Nombre	Descripción	Representación gráfica
R_Inertia	Define la inercia de cualquier sólido rígido en rotación.	

Como *R_Inertia* hereda de *R_Rigid* todas sus características, también tendrá los mismos puertos, representados en la tabla 4.33.

Tabla 4.33: Descripción de los puertos de *R_Inertia.*

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Las constantes y variables introducidas en este elemento se muestra en las tablas 4.34 y 4.35 respectivamente.

Tabla 4.34: Descripción constantes introducidas en *R_Inertia.*

Dato	Tipo	Descripción	Valor inicial	Unidades
I	REAL	Valor del momento de inercia del elemento.	1	$Kg \cdot m^2$
phi_0	REAL	Define la posición angular inicial del elemento.	0	rad

Tabla 4.35: Descripción variables declaradas en *R_Inertia.*

Variable	Tipo	Descripción	Unidades
phi	REAL	Define el posición angular absoluta del elemento.	rad
alpha	REAL	Define la aceleración angular absoluta del elemento.	rad/s^2

Si ahora se reescribe la ecuación (4.22) en función de las variables del elemento se tiene:

$$I \cdot alpha = m\_in.tau - m\_out.tau \tag{4.23}$$

Mediante la ecuación (4.23) se dota a R_Inertia de la capacidad de conocer el comportamiento del sólido en rotación. Pero además de esta ecuación, en el modelo se introducen las ecuaciones (4.24), que relaciona la aceleración angular del sólido en rotación con su posición angular, y (4.25) que impone una condición de contorno que permite calcular todas las incógnitas del modelo.

$$phi'' = alpha \tag{4.24}$$

$$phi = phi\_0 \tag{4.25}$$

El código en lenguaje de programación EL que representa el componente se muestra en la figura 4.22.

```
-------------------------------------------------------------------------------
-- Component R_Inertia
-------------------------------------------------------------------------------
COMPONENT R_Inertia IS_A R_Rigid
"Rotating mass with inertia"
   DATA
      REAL I = 1       UNITS "kg*m^2"          "Moment of inertia of body (kg*m^2)"
      REAL phi_0 = 0   UNITS "rad"             "Initial angular position (rad)"

   DECLS
     REAL alpha        UNITS "rad/s^2"         "Absolute angular acceleration (rad/s^2)"

   INIT
      phi = phi_0

   CONTINUOUS

      phi'' = alpha
      I * alpha = m_in.tau - m_out.tau

END COMPONENT
```

Figura 4.22: Código del componente *R_Inertia* en EL.

4.6.2. Componente *R_Spring*

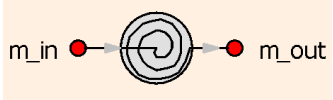
Mediante el componente *R_Spring* se consigue modelar el comportamiento de los resortes torsionales. La manera más sencilla de modelar un resorte de torsión ideal consiste en suponer que éste obedece la *Ley de Hooke*, la cual establece una relación entre el par en el resorte y la deformación angular que éste experimenta, a través de una constante *k_tor*, denominada constante de rigidez.

$$tau = - k\_tor \cdot (phi\_rel - phi\_natural) \tag{4.26}$$

La ecuación (4.26), que será la que caracterice el comportamiento de *R_Spring*, representa la *Ley de Hooke* para un resorte de torsión donde *tau* es el par en el resorte, *k_tor* es la constante de rigidez a torsión, *phi_rel* es la deformación que experimenta el resorte y *phi_natural* es la deformación natural del resorte. El signo negativo simboliza que el par en el resorte es opuesto al sentido de la deformación.

Además, este componente hereda todas las características de *R_Compliant*, que a su vez hereda de *R_two_Ports*, con lo que *R_Spring* adquiere las características de estos dos componentes abstractos. La descripción de *R_Spring* se muestra en la tabla 4.36.

Tabla 4.36: Descripción de *R_Spring*.

Nombre	Descripción	Representación gráfica
R_Spring	Define un resorte de torsión ideal con dos puertos mecánicos rotacionales y con movimiento relativo entre estos dos puertos.	

Los puertos que posee este elemento se presentan en la tabla 4.37.

Tabla 4.37: Descripción de los puertos de *R_Spring*.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Las constantes y variables introducidas en este elemento se muestran en las tablas 4.38 y 4.39 respectivamente.

Tabla 4.38: Descripción de las constantes introducidas por *R_Spring*.

Dato	Tipo	Descripción	Valor inicial	Unidades
k_tor	REAL	Define la constante de rigidez a torsión del resorte.	1	N·m/rad
phi_natural	REAL	Define la distancia angular natural entre los extremos del resote.	0	rad

Tabla 4.39: Descripción de las variables declaradas por *R_Spring*.

Variable	Tipo	Descripción	Unidades
phi_rel	REAL	Define el ángulo de giro relativo	rad

El código en lenguaje de programación EL que representa el componente se muestra en la figura 4.23.

```
-------------------------------------------------------------------------------
-- Component R_Spring
-------------------------------------------------------------------------------

COMPONENT R_Spring IS_A R_Compliant
"Ideal torsional spring"
   DATA
      REAL k_tor = 1        UNITS "N*m/rad"          "Spring constant (N*m/rad)"
      REAL phi_natural = 0  UNITS "rad"              "Angular distance between ports for unstretched spring (rad)"

   CONTINUOUS
      tau = - k_tor * (phi_rel - phi_natural)

END COMPONENT
```

Figura 4.23: Código del componente *R_Spring* en EL.

4.6.3. Componente *R_Damper*

El amortiguador es un dispositivo que absorbe energía, utilizado normalmente para disminuir las oscilaciones no deseadas de un movimiento periódico o para absorber energía proveniente de golpes o impactos.

Existen diversas formas de modelar el amortiguamiento, pero la más simple de todas ellas consta de una partícula o masa concentrada, que va perdiendo velocidad bajo la acción de una fuerza de amortiguamiento proporcional a su velocidad:

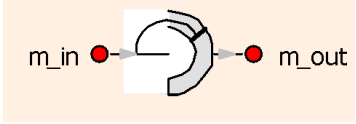
$$F = -c \cdot \frac{dx}{dt} \tag{4.27}$$

La ecuación (4.27) representa la fuerza de amortiguamiento para un movimiento de traslación. Si se traslada dicha ecuación a movimiento rotacional se obtiene la ecuación (4.28), que caracterizará el comportamiento del componente *R_Damper*:

$$tau = -c\_tor \cdot w\_rel \tag{4.28}$$

Donde *tau* representa el par que se opone al movimiento, *c_tor* es el amortiguamiento del sistema y *w_rel* es la velocidad relativa entre los extremos del amortiguador. La descripción de R_Damper se muestra en la tabla 4.40.

Tabla 4.40: Descripción de *R_Damper*.

Nombre	Descripción	Representación gráfica
R_Damper	Define un amortiguador rotacional ideal con dos puertos mecánicos rotacionales.	

Los puertos que posee este elemento se presentan en la tabla 4.41.

Tabla 4.41: Descripción de los puertos de *R_Damper*.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Las constantes y variables introducidas en este elemento se muestran en las tablas 4.42 y 4.43 respectivamente.

Tabla 4.42: Descripción de las constantes introducidas por *R_Damper*.

Dato	Tipo	Descripción	Valor inicial	Unidades
c_tor	REAL	Define la constante de amortiguamiento de *R_Damper*	1	N·m·s/rad

Tabla 4.43: Descripción de las variables declaradas en *R_Damper*.

Variable	Tipo	Descripción	Unidades
w_rel	REAL	Define la velocidad angular relativa entre los extremos del componente.	rad/s

El código en lenguaje de programación EL que representa el componente se muestra en la figura 4.24.

```
-------------------------------------------------------------------------------
-- Component R_Damper
-------------------------------------------------------------------------------

COMPONENT R_Damper IS_A R_Compliant
"Ideal rotational damper"
   DATA
      REAL c_tor = 1      UNITS  "N*m*s/rad"        "Damping constant (N*m*s/rad)"

   DECLS
      REAL w_rel       UNITS "rad/s"            "Relative angular velocity between ports (rad/s)"

   CONTINUOUS

   w_rel = m_out.phi' - m_in.phi'

      tau = - c_tor * w_rel

END COMPONENT
```

Figura 4.24: Código del componente *R_Damper* en EL.

4.7. Creación de componentes de la subclase *actuators*

En esta subclase se encuentran los tres actuadores genéricos de la librería. Estos actuadores están diseñados para recibir señales de una fuente analógica y transformarlas en señales que puedan ser interpretadas por los puertos mecánicos del resto de los componentes de la librería.

Todos los componentes de esta subclase heredan de la clase abstracta *R_Actuator*, por tanto, estarán formados por un puerto de entrada tipo *analog_signal*, y un puerto de salida tipo *mech_rot*, cuya descripción se muestra en la tabla 4.44.

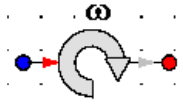
Tabla 4.44: Descripción de los puertos de los componentes de la subclase *R_Actuators*.

Puerto	Tipo	Dirección
s_in	*analog_signal*	IN
m_out	*mech_rot*	OUT

4.7.1. Componente *R_ ActuatorVelocity*

La descripción del componente R_ActuatorVelocity se muestra en la tabla 4.45.

Tabla 4.45: Descripción de *R_ActuatorVelocity*.

Nombre	Descripción	Representación gráfica
R_ActuatorVelocity	Define un actuador de velocidad angular.	

Este componente transforma la señal de entrada según la ecuación 4.29, que proviene de una fuente analógica *s_in.signal[1]*, en una señal de salida que los puertos de tipo *mech_rot*, identifican como velocidad angular (*m_out.phi'*).

$$m\_out.phi' = s\_in.signal[1] \tag{4.29}$$

El código en lenguaje de programación EL que representa el componente se muestra en la figura 4.25.

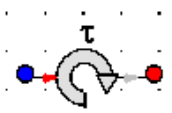
```
-------------------------------------------------------------------------------
-- Component R_ActuatorVelocity
-------------------------------------------------------------------------------
-- Purpose:
--    To represent an angular velocity actuator.
-------------------------------------------------------------------------------
COMPONENT R_ActuatorVelocity IS_A R_Actuator
"Angular velocity actuator"
   CONTINUOUS
      m_out.phi' = s_in.signal[1]

END COMPONENT
```

Figura 4.25: Código del componente *R_ActuatorVelocity* en EL.

4.7.2. Componente *R_ ActuatorTorque*

La descripción del componente R_ActuatorVelocity se muestra en la tabla 4.46.

Tabla 4.46: Descripción de *R_ActuatorTorque.*

Nombre	Descripción	Representación gráfica
R_ActuatorTorque	Define un actuador de par.	

Este componente transforma la señal de entrada según la ecuación 4.30, que proviene de una fuente analógica *s_in.signal[1]*, en una señal de salida que los puertos de tipo *mech_rot*, identifican como un par de torsión (*m_out.tau*).

$$m\_out.tau = s\_in.signal[1] \tag{4.30}$$

El código en lenguaje de programación EL que representa el componente se presenta en la figura 4.26.

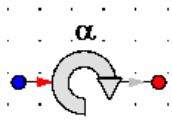
```
-------------------------------------------------------------------------------
-- Component R_ActuatorTorque
-------------------------------------------------------------------------------
-- Purpose:
--    To represent a torque actuator.
-------------------------------------------------------------------------------
COMPONENT R_ActuatorTorque IS_A R_Actuator
"Torque actuator"
   CONTINUOUS
   m_out.tau = s_in.signal[1]

END COMPONENT
```

Figura 4.26: Código del componente *R_ActuatorTorque* en EL.

4.7.3. Componente *R_ ActuatorAcceleration*

La descripción del componente R_ActuatorVelocity se muestra en la tabla 4.47.

Tabla 4.47: Descripción de *R_ActuatorAcceleration.*

Nombre	Descripción	Representación gráfica
R_ActuatorAcceleration	Define un actuador de aceleración angular.	

Este componente transforma la señal de entrada según la ecuación 4.31, que proviene de una fuente analógica *s_in.signal[1]*, en una señal de salida que los puertos de tipo *mech_rot*, identifican como una aceleración angular (*m_out. phi''*).

$$m\_out.phi'' = s\_in.signal[1] \tag{4.31}$$

El código en lenguaje de programación EL que representa el componente se muestra en la figura 4.27.

```
-------------------------------------------------------------------------------
-- Component R_ActuatorAcceleration
-------------------------------------------------------------------------------
-- Purpose:
--    To represent a rotational acceleration actuator.
-------------------------------------------------------------------------------
COMPONENT R_ActuatorAcceleration IS_A R_Actuator
"Rotational acceleration actuator"

   CONTINUOUS
      m_out.phi'' = s_in.signal[1]

END COMPONENT
```

Figura 4.27: Código del componente *R_ActuatorAcceleration* en EL.

4.8. Creación de componentes de la subclase *friction*

Dentro de esta subclase se encuentran los modelos de dos mecanismos, el embrague y el freno, que aunque comparten muchas características, persiguen objetivos distintos.

Cuando se conectan (ó desconectan) los elementos de una máquina que giran con velocidades angulares distintas, las cuales se quieren igualar, se habla de embrague. Si uno de estos elementos gira y el otro permanece fijo se habla de freno.

En las siguientes líneas se analizarán las características de los componentes *R_Clutch_jagm*, que modela el embregue mecánico, y *R_Brake_jagm*, que modela distintos tipos de frenos de fricción. Estos dos componentes heredan las características del elemento abstracto *R_AbsFriction*, que fue analizado en la sección 4.4.6 de este capítulo.

4.8.1. Componente *R_ Clutch_jagm*

El embrague es un dispositivo que permite conectar o desconectar dos elementos de una máquina que están girando, en general, a velocidades angulares distintas, consiguiendo que estas se igualen cuando el embrague esta embragado. La función principal del embrague consiste en conducir la potencia mecánica desde uno de los dispositivos que conecta, hacia el otro mediante el fenómeno de fricción [SAN05].

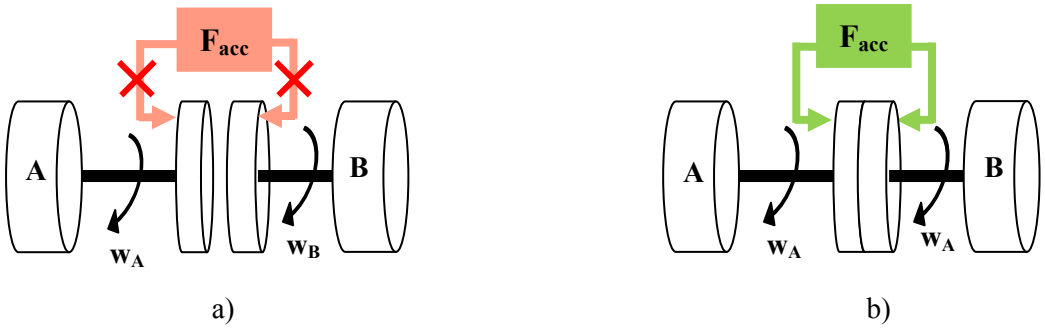


Figura 4.28: Croquis embrague. a) Desembragado. b) Embragado.

El funcionamiento del embrague de fricción queda descrito, de manera elemental, en la figura 4.28. Cuando la fuerza de accionamiento no actúa, figura 4.28.a, el embrague se encuentra desembragado y los elementos *A* y *B* no están en contacto, por lo que giran con velocidades angulares distintas, w_A y w_B respectivamente. Cuando la fuerza de accionamiento actúa sobre el embrague, figura 4.28.b, los elementos *A* y *B* quedan conectados a través de este y acaban girando a la velocidad del elemento conductor, que en este caso se ha supuesto que es el elemento *A*. Para conseguir transmitir la potencia desde *A* hacia *B*, los embragues emplean unos discos provistos de unos forros, denominados forros de fricción. Estos forros tienen forma de corona circular y están fabricados con materiales que poseen un elevado coeficiente de rozamiento para favorecer la transmisión de potencia mediante fricción.

Por tanto, en un embrague cuyos forros de fricción tienen un radio exterior R_e y un radio interior R_i, interesa determinar la fuerza normal (F_n) necesaria en el material de fricción para producir una presión p y poder transmitir así un cierto par N. Para realizar el cálculo de estas variables se pueden hacer dos hipótesis de partida diferentes:

- La presión p es constante a lo largo de toda la superficie de los forros de fricción. Este es el caso de que el embrague sea *nuevo*.

- Si por el contrario el embrague ha sufrido un determinado número de ciclos de vida, tendrá mayor desgaste en las zonas exteriores del mismo, ya que estos puntos están sometidos a mayor velocidad de deslizamiento, por lo tanto, la distribución de presiones ya no será constante. En estas circunstancias se dice que el embrague está *usado*.

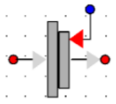
En la tabla 4.48, se resumen las expresiones que permiten calcular las variables de interés en los embragues, tanto para embragues nuevos como para embragues usados, donde p representa la presión en el material de fricción, p_a la presión máxima admisible en el material de fricción, R_e y R_i el radio exterior e interior del embrague respectivamente y μ el coeficiente de rozamiento entre las superficies en contacto.

Tabla 4.48: Expresiones que definen las principales variables de los embragues.

Variable	Estado del embrague	
	Nuevo	Usado
Presión en los forros de fricción	$p = cte$	$p(r) = \frac{p_a \cdot R_i}{r} \neq cte$[8]
Fuerza normal	$F_n = p \cdot \pi \cdot (R_e^2 - R_i^2)$	$F_n = p_a \cdot 2\pi \cdot R_i(R_e - R_i)$
Par generado	$N = \frac{4}{3} \cdot \mu \cdot \frac{(R_e^3 - R_i^3)}{(R_e^2 - R_i^2)} \cdot F_n$	$N = \mu \cdot (R_e + R_i) \cdot F_n$

Mediante el componente *R_Clutch_jagm*, se consigue modelar el comportamiento de un embrague de fricción seca, tanto para el caso de que este sea nuevo, como para el caso de que la distribución de presiones en los forros de fricción sea variable. La descripción de *R_Clutch_jagm* se muestra en la tabla 4.49.

Tabla 4.49: Descripción de *R_Clutch_jagm*.

Nombre	Descripción	Representación gráfica
R_Clutch_jagm	Define un embrague de fricción seca.	

[8] Si se parte de la hipótesis de que la distribución de presiones es variable, la presión máxima p_a tiene que darse para $r=R_i$, y, por lo tanto, la presión p a cualquier otra distancia depende del valor de r (distancia del centro al punto).

Además de los puertos declarados en *R_Two_Ports*, el componente R_Clutch_jagm dispone de un puerto analógico, cuya señal define el valor de la fuerza normal presente en los forros de fricción. La descripción de los puertos de *R_Clutch_jagm* se presenta en la tabla 4.50.

Tabla 4.50: Descripción de los puertos de *R_Clutch_jagm*

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT
inPort	*analog_signal*	IN

Las constantes y variables introducidas en este elemento se muestran en las tablas 4.51 y 4.52 respectivamente.

Tabla 4.51: Descripción de las constantes introducidas por *R_Clutch_jagm*.

Dato	Tipo	Descripción	Valor inicial	Unidades
data	ENUM	Constante de tipo enumerativo que se utiliza para seleccionar el dato de partida[9] (*pressure/force*).	*pressure*	-
clutch	ENUM	Constante de tipo enumerativo que se utiliza para indicar si el embrague es nuevo ó usado (*new/used*).	*new*	-
mue_pos	TABLE_1D	Tabla que define el valor del coeficiente de rozamiento de los forros de fricción en función de la velocidad w_relfric.	0,5 ∀ $w\_relfric$	-
peak	REAL	Define el valor máximo del coeficiente de rozamiento estático.	1,1	-
x	REAL	Constante auxiliar cuyo valor depende del dato de partida[9].	-	[N] si *data= force* [N/m^2] si *data= pressure*
n_disc	REAL	Define el número de discos de fricción del embrague (Cada disco posee dos superficies de contacto).	-	-
w_rel_i	REAL	Valor inicial para la velocidad angular relativa.	1	[rad/s]
Ri	REAL	Define el valor del radio interior de los forros de fricción del embrague.	-	[m]
Re	REAL	Define el valor del radio exterior de los forros de fricción del embrague.	-	[m]

[9] Mediante la constante enumerativa *data* se define el dato de partida que se utilizará para calcular el par de fricción en los forros del embrague. Cuando el dato conocido es la fuerza normal que el mecanismo de accionamiento ejerce sobre el material de fricción, habrá que seleccionar *data=force*. Si por el contrario, lo que se conoce es la presión máxima admisible en el material de fricción, habrá que seleccionar *data=pressure*.
La variable *x* es una variable necesaria para dotar al modelo la versatilidad en la selección del dato de partida que le ofrece la constante enumerativa *data*.

Tabla 4.52: Descripción de las variables declaradas en *R_Clutch_jagm*.

Variable	Tipo	Descripción	Unidades
p	REAL	Define la presión máxima admisible en el material de fricción.	[N/m^2]
fn_max	REAL	Define el valor de la fuerza normal máxima admisible en los forros de fricción.	[N]
mue0	REAL	Variable auxiliar que se usa para calcular el valor del coeficiente de fricción estático.	-
fn	REAL	Variable que define el valor de la fuerza normal real actuando en los forros de fricción.	[N]
cgeo	REAL	Variable que define la constante geométrica del embrague.	-

El código del elemento *R_Clutch_jagm* dispone de una parte discreta, en la que se establece cuándo y cómo se activa la fricción en el embrague (Figura 4.29), esto es, si se tiene fuerza normal el los forros entonces hay fricción, de lo contrario el embrague está desactivado.

```
 DISCRETE

--Discrete statements defining the activation of the clutch depending on the boolean data free.

   WHEN(fn<= 0.0) THEN
     free = TRUE
   END WHEN

--Discrete statements defining imode value when the the clutch is active.
   WHEN(fn > 0.0) THEN
     free = FALSE
     IF(w_relfric > 0) THEN
        imode = 2
     ELSE
        imode = -2
     END IF
   END WHEN
```

Figura 4.29: Fragmento del código de *R_Clutch_jagm* que define la activación de la fricción, y el valor inicial de la variable *imode*.

En el fragmento de *R_Clutch_jagm* de la figura 4.29, se establece el valor inicial de la variable *imode*. Es importante recordar que el valor de la variable *imode* cuando la variable booleana *free* es verdadera se establece en el elemento *R_AbsFriction* (ver sección 4.4.6).

En la figura 4.30 se representa la parte discreta del modelo donde se definen los valores de las variables *cgeo, fn_max y p*, los cuales dependen del dato de partida seleccionado vía *x* y *data*, y del estado del embrague.

```
--Discrete statements defining cgeo,p and fn_max, depending on the clutch status and  start_data.

   WHEN (Clutch == new) THEN
     cgeo =((n_disc*4)/3)*((Re**3-Ri**3)/(Re**2-Ri**2))
      IF(data==pressure)THEN
        fn_max =(x*PI)*(Re**2-Ri**2)
        P=x
      ELSE
        fn_max =x
        P=x/(PI*(Re**2-Ri**2))
      END IF
   END WHEN


   WHEN (Clutch == used) THEN
       cgeo = (n_disc*(Re+Ri))
        IF (data==pressure)THEN
          fn_max=2*x*PI*Ri*(Re-Ri)
          P=x
         ELSE
          fn_max=x
          P=x/(2*PI*Ri*(Re-Ri))
         END IF
     END WHEN

  CONTINUOUS
```

Figura 4.30: Fragmento del código de *R_Clutch_jagm en el que* se definen los valores de las variables *cgeo*, *fn_max* y *p*.

Las figuras 4.31 y 4.32, representan los diagramas de flujo de los fragmentos de código en los que se definen los valores de las variables *cgeo, fn_max* y *p*, y el valor inicial de la variable *imode,* respectivamente.

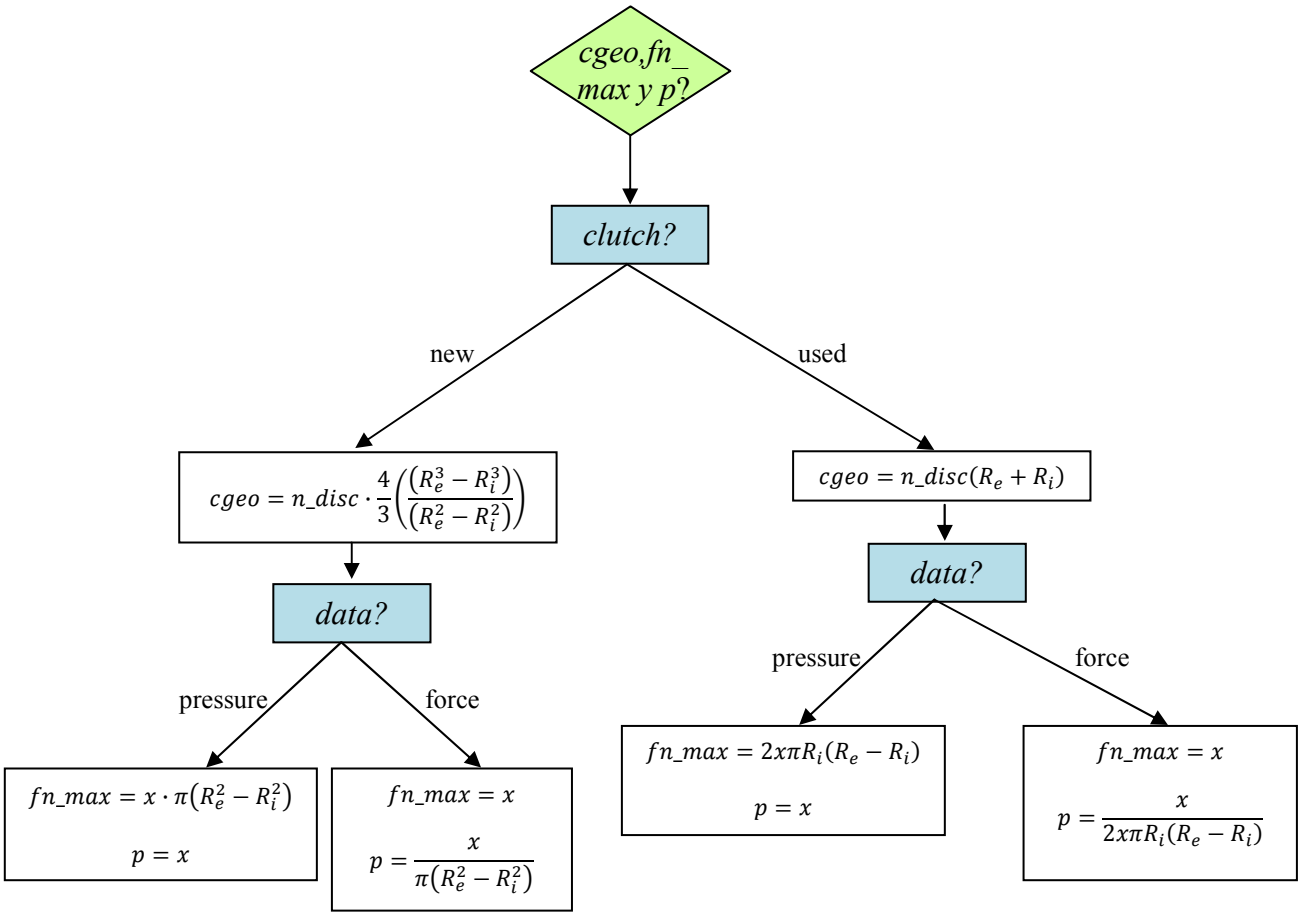


Figura 4.31: Diagrama de flujo del fragmento del código de *R_Clutch_jagm en el que* se definen los valores de las variables *cgeo, fn_max y p*.

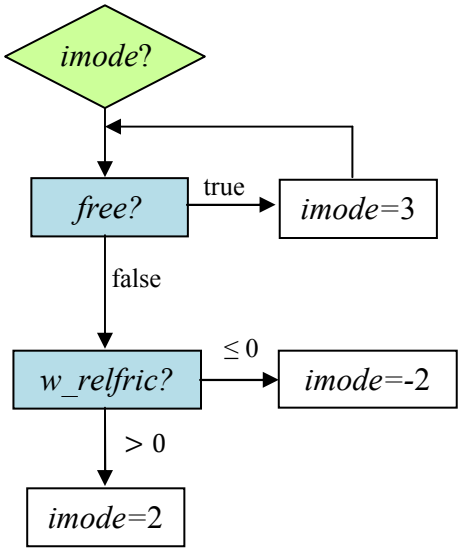


Figura 4.32: Diagrama de flujo del fragmento del código de *R_Clutch_jagm en el que* se define el valor inicial de la variable *imode*.

En la figura 4.33 se presenta la parte continua del componente *R_Clutch_jagm*.

```
   CONTINUOUS

   --auxiliary variable used to compute the friction coef. for w_relfric=0.
   mue0 = linearInterp1D(mue_pos, 0.0)

   --Relative quantities.
   w_relfric = phi_rel'
   a_relfric = phi_rel''

   --Normal force and friction torque for w_relfric=0.
   fn = fn_max*inPort.signal[1]
   tau0 = mue0*cgeo*fn
   tau0_max = peak*tau0

   --compute of Friction torque depending on the imode value.
   tau_friction = ZONE(imode == 0) sa
                  ZONE(imode == 1) cgeo*fn*linearInterp1D(mue_pos, phi_rel')
                  ZONE(imode == -1) -cgeo*fn*linearInterp1D(mue_pos, -phi_rel')
                  ZONE(imode == 2) cgeo*fn * linearInterp1D(mue_pos, phi_rel')
                  ZONE(imode == -2) - cgeo*fn*linearInterp1D(mue_pos, -phi_rel')
                  OTHERS 0.0

   tau_friction = -tau

END COMPONENT
```

Figura 4.33: Parte continua del código de *R_Clutch_jagm* .

En el fragmento de código de la figura anterior se llevan a cabo las siguientes acciones:

- Se calcula el coeficiente de rozamiento estático *mue0* según la ecuación 4.33.

$$mue0 = linearInterp1D\ (mue\_pos,\ 0.0) \tag{4.32}$$

  Este parámetro se calcula realizando una interpolación lineal de la tabla *mue_pos* para *w_relfric = 0*.

- Se asignan valores a las variables *w_relfric* y *a_relfric* según las ecuaciones 4.33 y 4.34 respectivamente.

$$w\_relfric = phi\_rel' \tag{4.33}$$

$$a\_relfric = phi\_rel'' \tag{4.34}$$

  Es importante recordar que la expresión para calcular la variable *phi_rel* se declara en el elemento *R_Compliant*, cuyas propiedades hereda *R_Cutch_jagm*.

- Se calcula el valor de la fuerza normal en el material de fricción según la ecuación 4.35.

$$fn = fn\_max \cdot inPort.signal[1] \tag{4.35}$$

  Esta ecuación define el valor de la fuerza normal presente en los forros *fn*, que depende del valor de la señal analógica del puerto *inPort*. Así, si *inPort.signal[1] = 1,* la fuerza normal en los forros será la máxima posible, esto es, *fn_max*, y, si *inPort.signal[1]* es nula no existirá par de fricción.

- Se calculan los valores de los pares de fricción estáticos *tau0* (aplicable para *imode=1/-1*) y *tau0_max* (aplicable para *imode=0*).

$$tau0 = mue0 \cdot cgeo \cdot fn \tag{4.36}$$

$$tau0\_max = peak \cdot tau0 \tag{4.37}$$

  Estos valores dan forma a la curva parametrizada de la figura 4.14, que define el comportamiento del elemento *R_AbsFriction*.

- Se calcula el valor del par de fricción *tau_friction,* esto es, el par que se opone al movimiento relativo del embrague*,* según la ecuación 4.38, y el par transmitido por el embrague, es decir, el par que transmite el embrague desde el elemento conductor al conducido, según la ecuación 4.39.

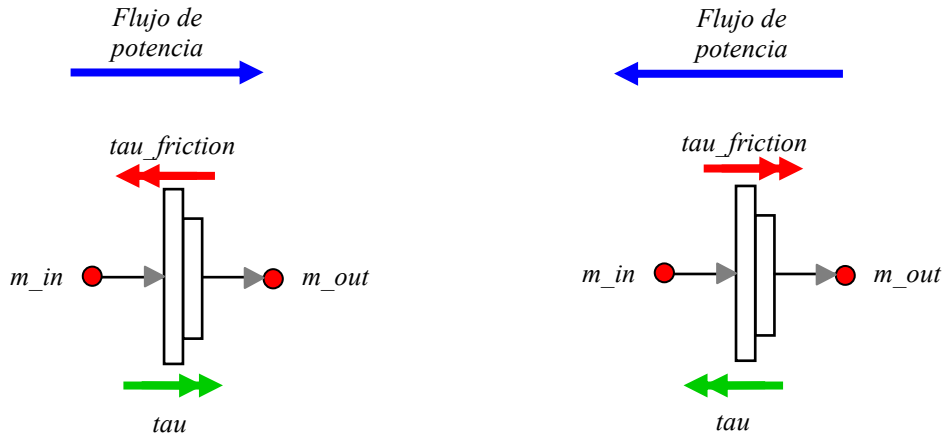
$$tau\_friction \begin{cases} sa & para\ imode = 0 \\ cgeo \cdot fn \cdot linearInterp1D\ (mue\_pos, phi\_rel') & para\ imode = 1\ ó\ 2 \\ -cgeo \cdot fn \cdot linearInterp1D\ (mue_{pos}, -phi\_rel') & para\ imode = -1\ ó\ -2 \end{cases} \tag{4.38}$$

$$tau\_friction = -\ tau \tag{4.39}$$

La ecuación (4.38), que se utiliza para calcular el valor del par de fricción, se define por tramos, ya que el valor de la fricción varía en función del modo de deslizamiento del

embrague (ver tabla 4.19). Además, el par transmitido por el embrague *tau*, debe ser opuesto al par de fricción, ya que debe fluir en el sentido de la transmisión de potencia.

La figura 4.34 representa los sentidos de *tau_friction* y *tau*, en función del modo de deslizamiento, que a su vez depende del signo de la velocidad relativa *w_relfric* calculada según la ecuación (4.33).



a) m_in.phi'>m_out.phi'
w_relfric<0 → imode = -2

b) *m_in.phi'<m_out.phi'*
w_relfric>0 → imode = 2

Figura 4.34: Sentidos de *tau_friction* y *tau* en función de *w_relfric* durante la fase de acoplamiento del embrague (embrague *patinando*); *a)* Los elementos a la izquierda del embrague actúan como conductores; *b)* los elementos a la derecha del embrague actúan como conductores.

La figura 4.35 muestra el código completo del componente *R_Clutch_jagm*.

```
COMPONENT R_Clutch_jagm  IS_A R_AbsFriction, R_Compliant
"Dry clutch"

  PORTS
   IN analog_signal(n = 1) inPort         "Normalized force signal"

  DATA
   ENUM Start_data data=pressure           "Enumerative data type for select the starting data"
   ENUM Clutch_status Clutch=new           "Clutch status"
   TABLE_1D mue_pos ={(0,1), (0.5, 0.5)}   "Sliding friction coefficient"
   REAL peak = 1.1                         "Maximum value of mue for w_rel=0"
   REAL x                                  "Strat_data value [N] if drive_force and [N/m**2] if maximun pressure"
   REAL n_disc                             "number of friction discs [-]"
   REAL w_rel_i = 1                        "Initial relative angular velocity [rad/s]"
   REAL Ri                                 "disc brake internal radius [m]"
   REAL Re                                 "disc brake external radius [m]"

  DECLS

   REAL P          "maximum pressure on the lining [N/m**2]"
   REAL fn_max     "Maximum normal force (N)"
   REAL mue0       "Friction coefficient for w=0 and forward sliding"
   REAL fn         "Normal force (N)"
   REAL cgeo       "Clutch geometry"

  INIT

   w_relfric = w_rel_i
  DISCRETE

 --Discrete statements defining the activation of the clutch depending on the boolean data free.

    WHEN(fn<= 0.0) THEN
      free = TRUE
    END WHEN

 --Discrete statements defining imode value when the the clutch is active.
    WHEN(fn > 0.0) THEN
      free = FALSE
      IF(w_relfric > 0) THEN
         imode = 2
      ELSE
         imode = -2
      END IF
    END WHEN
```

```
--Discrete statements defining cgeo,p and fn_max, depending on the clutch status and  start_data.

   WHEN (Clutch == new) THEN
     cgeo =((n_disc*4)/3)*((Re**3-Ri**3)/(Re**2-Ri**2))
      IF(data==pressure)THEN
        fn_max =(x*PI)*(Re**2-Ri**2)
        P=x
      ELSE
        fn_max =x
        P=x/(PI*(Re**2-Ri**2))
      END IF
   END WHEN


   WHEN (Clutch == used) THEN
       cgeo = (n_disc*(Re+Ri))
        IF (data==pressure)THEN
          fn_max=2*x*PI*Ri*(Re-Ri)
          P=x
        ELSE
          fn_max=x
          P=x/(2*PI*Ri*(Re-Ri))
        END IF
     END WHEN

           CONTINUOUS

           --auxiliary variable used to compute the friction coef. for w_relfric=0.
           mue0 = linearInterp1D(mue_pos, 0.0)

           --Relative quantities.
           w_relfric = phi_rel'
           a_relfric = phi_rel''

           --Normal force and friction torque for w_relfric=0.
           fn = fn_max*inPort.signal[1]
           tau0 = mue0*cgeo*fn
           tau0_max = peak*tau0

           --compute of Friction torque depending on the imode value.
           tau_friction = ZONE(imode == 0) sa
                          ZONE(imode == 1) cgeo*fn*linearInterp1D(mue_pos, phi_rel')
                          ZONE(imode == -1) -cgeo*fn*linearInterp1D(mue_pos, -phi_rel')
                          ZONE(imode == 2) cgeo*fn * linearInterp1D(mue_pos, phi_rel')
                          ZONE(imode == -2) - cgeo*fn*linearInterp1D(mue_pos, -phi_rel')
                          OTHERS 0.0

           tau_friction = -tau

        END COMPONENT
```

Figura 4.35: Código del componente *R_Clutch_jagm* en EL.

4.8.2. Componente *R_Drum_and_disc_Brake_jagm*

Mediante el componente *R_Drum_and_disc_Brake_jagm*, se pueden simular distintos tipos de frenos de fricción. Estos frenos están diseñados para desacelerar cuerpos en movimiento transformando en calor la energía cinética mediante fenómenos de fricción. Siempre constan de un cuerpo con una superficie corrugada (superficie fija), que es presionado sobre un disco u objeto a detener (superficie en rotación).

En general, independientemente del origen de la fuerza de accionamiento, se pueden distinguir dos tipos de frenos de fricción:

- ***Frenos de tambor***. Un freno de tambor es un freno de fricción en el que mediante la aplicación de una fuerza de accionamiento, las zapatas, que son elementos con velocidad nula provistos de un material de fricción, se ponen en contacto con la superficie (interna, externa o por ambas caras) de un cilindro denominado tambor, que gira solidario con el cuerpo a desacelerar. Este contacto produce unas fuerzas de rozamiento que originan un par de frenada que tiende a reducir la velocidad angular del tambor. Cuando la fuerza de

accionamiento cesa, las zapatas retornan a su posición inicial liberando así al tambor de la acción de las fuerzas de rozamiento.

En la figura 4.36 se representan los elementos principales de un freno de tambor.

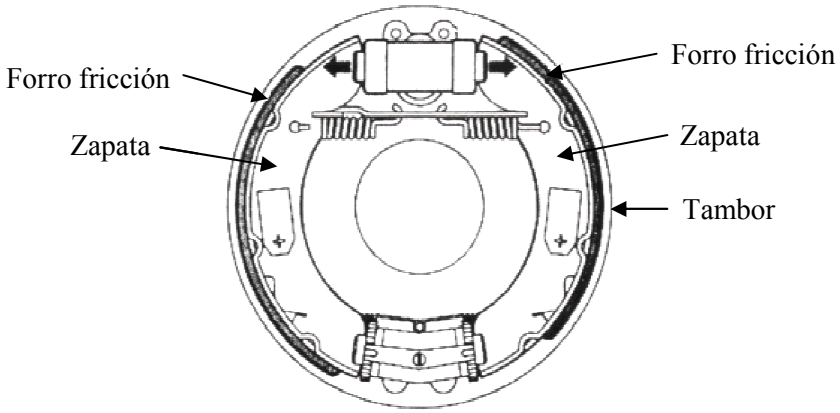


Figura 4.36: Croquis freno de tambor.

En los frenos de tambor con dos zapatas iguales aparece un efecto denominado de arrastre o auto frenado (ver figura 4.37), por el cual las reacciones que aparecen en las zapatas T_1 y T_2 , tienden a empujar una de las zapatas contra el tambor (zapata primaria) y a separar la otra (zapata secundaria). Si se invierte el sentido de giro del tambor también lo hace el efecto de arrastre.

Por otra parte, en el caso de zapatas articuladas (Figura 4.37), que son las que modela el componente *R_Drum_and_disc_Brake_jagm,* se hace la hipótesis de que la distribución de presiones que se genera en el contacto zapata-tambor es tal que la presión en un punto es proporcional a la distancia vertical al punto de articulación y ésta es, a su vez, proporcional al ángulo θ comprendido entre la horizontal que pasa por el punto de la articulación y el radio que une el centro del tambor con el punto de contacto en cuestión (ver figura 4.39 y tabla 4.54).

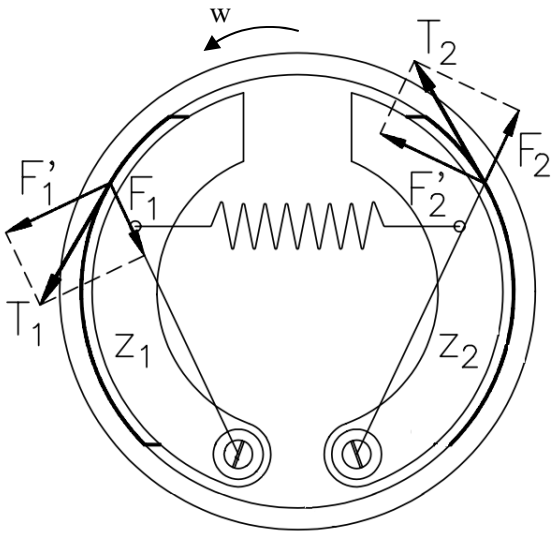


Figura 4.37: Efecto de arrastre o auto frenado en zapatas articuladas.

- ***Frenos de disco***. En este tipo de frenos, cuyo esquema general se presenta en la figura 4.38.a, la fuerza de frenado se obtiene por la aplicación de fuerzas axiales sobre un disco (1) solidario a un eje (2). Además este tipo de freno dispone de una mordaza o pinza de freno (3), que puede alojar en su interior uno o más bombines (4), que pueden ser hidráulicos, neumáticos, etc. Entre los bombines y el disco se interpone una pastilla o placa de freno (5) de alto coeficiente de fricción.

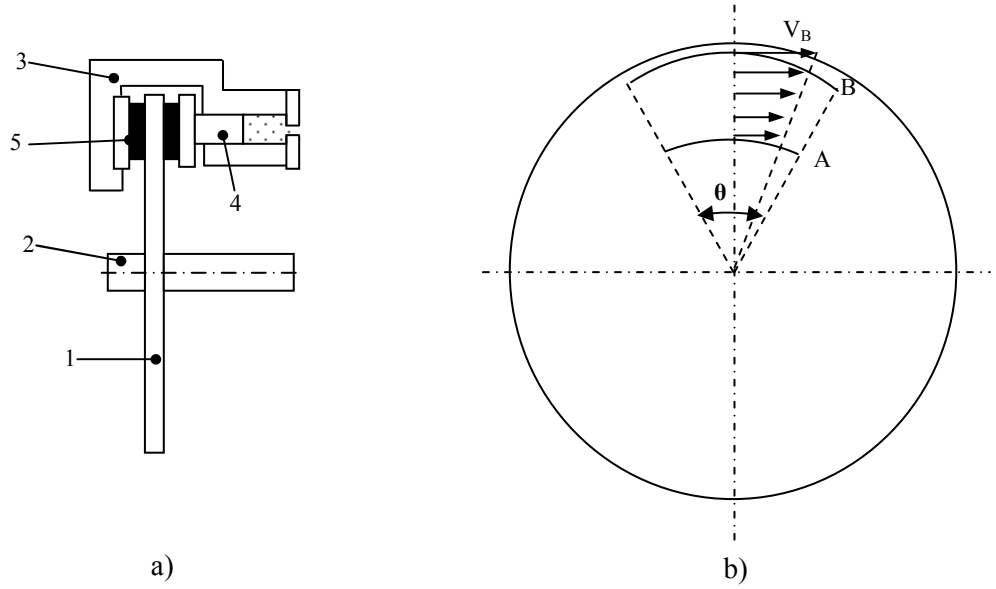


Figura 4.38: a) Croquis freno de disco; b) Distribución de velocidades en freno de disco.

La velocidad de deslizamiento entre disco y pastillas aumenta del centro a la periferia (Figura 4.38.b) y puede variar hasta duplicar su valor de A hasta B. Esto significa que si la presión es la misma en todos los puntos del forro, por ejemplo, en el caso de un freno nuevo, el desgaste no será uniforme. Por el contrario, en el caso de que el freno esté muy usado, el desgaste pasará a ser uniforme en toda la superficie y, por lo tanto, la presión no será la misma en todos los puntos de la pastilla [SAN05].

En la tabla 4.53, se resumen las expresiones que permiten calcular las variables de interés en frenos de disco (con una única superficie rozante), donde p representa la presión en el material de fricción, p_a la presión máxima admisible en el material de fricción, R_e y R_i el radio exterior e interior de la pastilla de freno, μ el coeficiente de rozamiento entre pastilla y disco y α el ángulo abarcado por la pastilla de freno.

Tabla 4.53: Expresiones que definen las principales variables de los frenos de disco.

Variable	Estado del freno	
	Nuevo	Usado
Presión en el material de fricción[10]	$p = cte$	$p(r) = \frac{p_a \cdot R_i}{r} \neq cte$
Fuerza normal	$Fn = \frac{1}{2} \cdot p \cdot \alpha \cdot (R_e^2 - R_i^2)$	$F_n = p_a \cdot \alpha \cdot R_i(R_e - R_i)$
Par de frenada	$N = \frac{2}{3} \cdot \mu \cdot \frac{(R_e^3 - R_i^3)}{(R_e^2 - R_i^2)} \cdot F_n$	$N = \frac{1}{2} \cdot \mu \cdot (R_e + R_i) \cdot F_n$

Si en lugar de un freno de disco, se tiene un freno de tambor con zapatas como la de la figura 4.39, donde *T* representa la fuerza de accionamiento, dF_n y dF_{roz} los diferenciales de fuerza normal y rozamiento sobre la zapata para el sentido de giro dado por *w*, θ la posición angular con θ_1 y θ_2 los ángulos de inicio y fin del material de fricción (tomando como origen la articulación), *a* la distancia de la articulación al centro del tambor, *c* la distancia entre la línea de acción de *T* y la articulación, *r* el radio del tambor y *b* el ancho de la zapata, las expresiones que permiten calcular las variables de interés se muestran en la tabla 4.54.

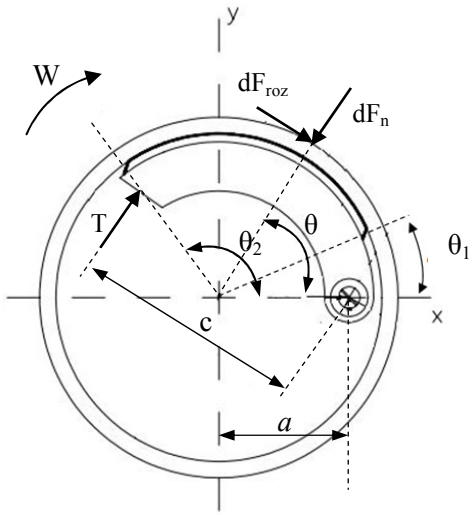


Figura 4.39: Croquis zapata articulada.

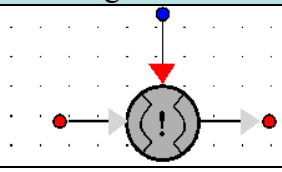
[10] En el caso de que el freno esté usado y la presión en todos los puntos de la pastilla no sea la misma, se hace la hipótesis de que la presión máxima p_a se produce para $r=R_i$, y por lo tanto la presión P en cualquier otro punto dependerá de la distancia *r* al centro del freno [SAN05].

Tabla 4.54: Expresiones que definen las principales variables de los frenos de tambor.

Variable	Expresión
Presión en el material de fricción[11]	$p(\theta) = p_a \cdot \frac{sen(\theta)}{sen(\theta_a)}$
Fuerza normal	$F_n = \frac{p_a \cdot b \cdot r}{sen\theta_a} \cdot [\cos\theta_1 - \cos\theta_2]$
Par de frenada	$N = r \cdot \mu \cdot F_n$
Par producido en la articulación por las fuerzas de rozamiento	$N_{roz} = \frac{\mu \cdot p_a \cdot b \cdot r}{sen\theta_a} \cdot \{(-r \cdot \cos\theta)\Big\|_{\theta_1}^{\theta_2} - a \cdot \left(\frac{1}{2} \cdot sen^2\theta\right)\Big\|_{\theta_1}^{\theta_2}\}$
Par producido en la articulación por las fuerzas de normales	$N_n = \frac{p_a \cdot b \cdot r \cdot a}{sen\theta_a} \cdot \left(\frac{\theta}{2} - \frac{1}{4} \cdot sen(2\theta)\right)\Big\|_{\theta_1}^{\theta_2}$

Una vez descritos los principales tipos de frenos, se analizará el componente *R_Drum_and_disc_Brake_jagm,* con el que se podrán simular discos de frenos con múltiples disposiciones geométricas, tanto nuevos como usados, así como frenos de zapatas articuladas con distintas disposiciones de las zapatas. La descripción de R_Drum_and_disc_Brake_jagm se muestra en la tabla 4.55.

Tabla 4.55: Descripción de *R_Drum_and_disc_Brake_jagm*.

Nombre	Descripción	Representación gráfica
R_Drum_and_disc_Brake_jagm	Define distintos tipos de frenos basados en el fenómeno de fricción.	

Además de los puertos declarados en *R_Two_Ports*, el componente *R_Drum_and_disc_Brake_jagm* dispone de un puerto analógico cuya señal define el valor de la fuerza normal presente en los forros de fricción (ver tabla 4.56).

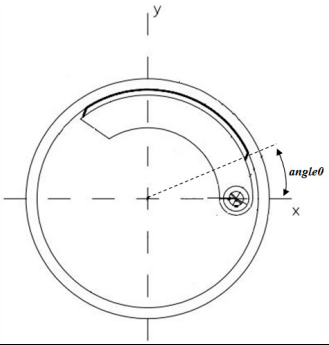
Tabla 4.56: Descripción de los puertos de *R_Drum_and_disc_Brake_jagm*.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT
inPort	*analog_signal*	IN

[11] Dada la geometría del freno, existirá un punto de máxima presión p_a, que forme un ángulo θ_a con la horizontal que pasa por el punto de articulación de la zapata En esta expresión se observa que la presión se hace máxima cuando θ=90º y mínima cuando θ=0º. No obstante en general las zapatas abarcan un ángulo mayor de 90º y la presión máxima se produce para θ_a=90º, con lo que en estos casos *sen* θ_a=1 [SAN05].

Las constantes y variables que se introducen en este elemento se describen en las tablas 4.57 y 4.58 respectivamente.

Tabla 4.57: Descripción de las contantes introducidas por *R_Drum_and_disc_Brake_jagm.*

Dato	Tipo	Descripción	Valor inicial	Unidades
type	ENUM	Constante de tipo enumerativo que se utiliza para seleccionar el tipo de freno.	*new_brake_disc*	-
mue_pos	TABLE_1D	Tabla que define el valor del coeficiente de rozamiento de los forros de fricción en función de la velocidad w_relfric.	$0{,}5 \; \forall \; w\_relfric$	-
peak	REAL	Define el valor máximo del coeficiente de rozamiento estático.	1,1	-
p_max	REAL	Presión máxima admisible en el material de fricción.	-	[N/m^2]
angle	REAL	Ángulo abarcado por las pastillas de freno, o por las zapatas.	-	[º]
angle0	REAL	Desfase angular del forro de la zapata con respecto a la articulación. 	0	[º]
angle_Pmax	REAL	Ángulo de la zapata para el que se tiene mayor presión.		
w_rel_i	REAL	Valor inicial para la velocidad angular relativa.	1	[rad/s]
Ri	REAL	Define el valor del radio interior del disco de freno.	-	[m]
Re	REAL	Define el valor del radio exterior en caso de freno de disco, o el radio del tambor en caso de seleccionar frenos de tambor.	-	[m]
n_pad	REAL	Número de pastillas de freno para discos de freno.	-	-

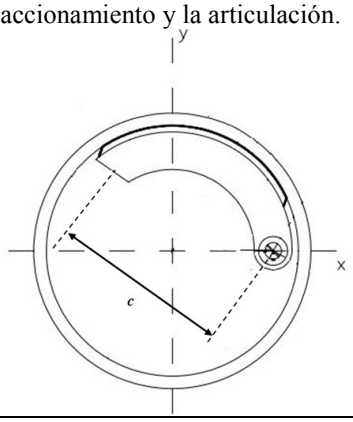
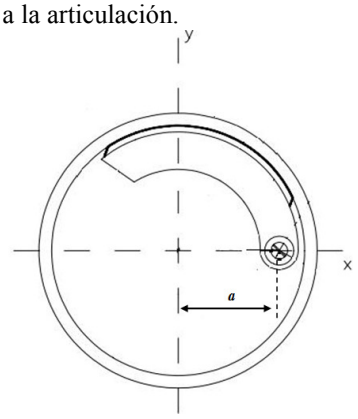
c	REAL	Distancia entre el punto de aplicación de la fuerza de accionamiento y la articulación. 	-	[m]
b	REAL	Ancho de la zapata.	-	[m]
a	REAL	Distancia del centro del tambor a la articulación. 	-	[m]

Tabla 4.58: Descripción de las variables declaradas en *R_Drum_and_disc_Brake_jagm*.

Variable	Tipo	Descripción	Unidades
fn_max	REAL	Define el valor de la fuerza normal máxima admisible en el material de fricción.	[N]
cgeo	REAL	Variable que define la constante geométrica del freno.	-
alpha	REAL	Define la aceleración angular absoluta de componente.	[rad/s^2]
mue0	REAL	Variable auxiliar que se usa para calcular el valor del coeficiente de fricción estático.	-
fn	REAL	Variable que define el valor de la fuerza normal real actuando en el material de fricción.	[N]
p_2	REAL	Para frenos de tambor, define la presión en las zapatas secundarias.	[N/m^2]

pa	REAL	Variable auxiliar para determinar, en función del tipo de freno, la presión máxima utilizada para calcular el par de frenada.	[N/m^2]
F	REAL	Define el valor de la fuerza de accionamiento.	[N]
N_n	REAL	Para frenos de tambor, par en la articulación producido por las fuerzas normales.	[N·m]
N_f	REAL	Para frenos de tambor, par en la articulación producido por las fuerzas de rozamiento.	[N·m]

Al igual que ocurría con el componente *R_Clutch_jagm,* el código del elemento *R_Drum_and_disc_Brake_jagm* dispone de una parte discreta en la que se establece cuándo y cómo se activa la fricción (ver figura 4.40), esto es, si se tiene fuerza normal en el material de fricción entonces hay fricción, de lo contrario el freno está desactivado.

```
DISCRETE
 --Discrete statements defining the activation of the brake depending on the boolean data free.
 WHEN(fn<= 0.0) THEN
   free = TRUE
 END WHEN

--Discrete statements defining imode value when the the brake is active.
 WHEN(fn > 0.0) THEN
   free = FALSE
   IF(w_relfric > 0) THEN
      imode = 2 AFTER 0.0
   ELSE
      imode = -2 AFTER 0.0
   END IF
 END WHEN
```

Figura 4.40: Fragmento del código de *R_Drum_and_disc_Brake_jagm* que define la activación de la fricción, y el valor inicial de la variable *imode*.

En el fragmento de *R_Drum_and_disc_Brake_jagm* de la figura 4.40, se establece el valor inicial de la variable *imode*.

La variable de tipo enumerativo *type* permite seleccionar el tipo de freno que simulará el componente. La elección del tipo de freno implica una serie hipótesis de cálculo que se reflejan en la tabla 4.59. Se considera que la velocidad de giro es positiva *(w>0)* si su sentido es horario, y negativa *(w<0)* si es antihorario.

Figura 4.59: Hipótesis de cálculo en función del tipo de freno seleccionado.

Tipo de freno	Hipótesis de cálculo
twin_shoes_for	Dos zapatas gemelas (primarias) con presión $p_1 \neq cte$ cuando la

	velocidad de giro es positiva, y dos zapatas gemelas (secundarias) con presión $p_2 \neq cte$ cuando la velocidad de giro es negativa. $(p_1 > p_2)$ (Figura 4.41.e y 4.41.f)
twin_shoes_back	Dos zapatas gemelas (primarias) con presión $p_1 \neq cte$ cuando la velocidad de giro es negativa, y dos zapatas gemelas (secundarias) con presión $p_2 \neq cte$ cuando la velocidad de giro es positiva. $(p_1 > p_2)$ (Figura 4.41.c y 4.41.d)
primary_secondary_shoes	Una zapata primaria con presión $p_1 \neq cte$ y otra zapata secundaria con presión $p_2 \neq cte$ $(p_1 > p_2)$ (Figura 4.41.a y 4.41.b)
new_brake_disc	$p=cte$
used_brake_disc	$p(r) = \frac{p_a \cdot R_i}{r} \neq cte$[12]

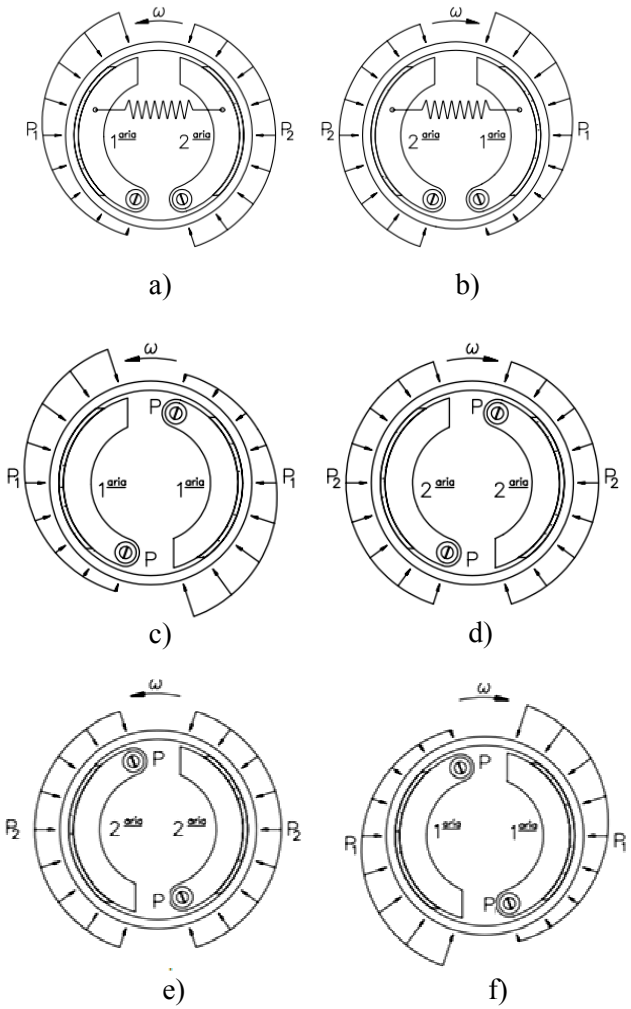


Figura 4.41: Configuración de las zapatas según tipo de freno de tambor y el sentido de la velocidad de giro: a) *primary_secondary_shoe*s y $w<0$; b) *primary_secondary_shoes* y $w>0$; c)*twin_shoes_back* y $w<0$; d)*twin_shoes_back* y $w>0$; e)*twin_shoes_for* y $w<0$; f)*twin_shoes_for* y $w>0$.

[12] Si se parte de la hipótesis de que la distribución de presiones es variable, la presión máxima p_a tiene que darse para $r=R_i$, y por lo tanto la presión p a cualquier otra distancia depende del valor de r (distancia del centro al punto).

En la figura 4.42 se representa la parte discreta del modelo donde se asigna un valor a la variable *pa* (presión empleada para calcular el par de frenada) en función del tipo de freno seleccionado vía *type,* y del signo de la velocidad relativa entre las superficies de fricción *w_relfric*.

```
--Discrete statements defining the pressure on the shoes depending on the brake type and w_relfric.
 WHEN(w_relfric > 0) THEN
   IF (type == twin_shoes_back) THEN
      pa = p_2    AFTER 0.0
   ELSE
      pa = p_max AFTER 0.0
   END IF
 END WHEN

 WHEN(w_relfric < 0) THEN
   IF(type == twin_shoes_for) THEN
      pa = p_2   AFTER 0.0
   ELSE
      pa = p_max   AFTER 0.0
   END IF
 END WHEN

CONTINUOUS
```

Figura 4.42: Fragmento discreto de *R_Drum_and_disc_Brake_jagm* en el que se asigna un valor a la variable *pa*.

Así, si *w_relfric>0,* el único tipo freno en el que la presión en los forros no alcanzará el valor máximo admisible *p_max* será *twin_shoes_back* (ver figura 4.41). Por ello, para este tipo de freno, la variable *pa* toma el valor de *p_2*, que será calculada en la parte continua del modelo. Lo mismo ocurre para *twin_shoes_for* cuando *w_relfric<0.*

En la parte continua del modelo se calculan múltiples variables con el único objetivo de determinar el par de frenada *tau_friction*. Independientemente del freno seleccionado, el par de frenada se calculará, al igual que en el componente *R_Clutch_jagm*, según la ecuación (4.41) implementada en EL en la figura 4.43.

```
 --Friction torque representing the braking torque
 tau_friction =    ZONE(imode == 0) sa
                   ZONE(imode == 1) cgeo*fn*linearInterp1D(mue_pos, phi')
                   ZONE(imode == -1) -cgeo*fn*linearInterp1D(mue_pos, -phi')
                   ZONE(imode == 2) cgeo*fn * linearInterp1D(mue_pos, phi')
                   ZONE(imode == -2) - cgeo*fn*linearInterp1D(mue_pos, -phi')
                   OTHERS 0.0
```

Figura 4.43: Fragmento de *R_Drum_and_disc_Brake_jagm* en el que se implementa la función que calcula el par de frenada.

Como se observa en la figura 4.43, cuando el modelo se encuentra en el modo de deslizamiento estático (para consultar modos de deslizamiento ir a tabla 4.19) el par se calcula vía *sa* según se describe en el componente *R_AbsFriction*. Sin embargo para el resto de modos de deslizamiento, el par de frenada depende de las variables *cgeo*, *fn* y del valor de *mue_pos* para la velocidad de giro en el instante dado.

Debido a que las expresiones que determinan el valor de las variables *cgeo* y *fn* varían en función del freno seleccionado vía *type*, es necesario que el código del componente sea capaz de utilizar las expresiones adecuadas en cada momento.

El fragmento de *R_Drum_and_disc_Brake_jagm* de la figura 4.44 determina las expresiones que se emplearán para calcular las variables *fn_max* y *cgeo* en función del tipo de freno seleccionado. La primera ecuación de esta figura define el valor de la fuerza normal presente en los forros *fn*, que depende de *fn_max* y del valor de la señal analógica del puerto *inPort*. Así si *inPort.signal[1] = 1* la fuerza normal en los forros será la máxima posible, esto es, *fn_max*, y si *inPort.signal[1]* es nula no existirá par de fricción.

```
fn = fn_max*inPort.signal[1]

--Maximun normal force depending on the brake type.

 fn_max = IF (type == new_brake_disc)
           (0.5*pa*((PI*angle)/180)*(Re**2-Ri**2))
          ELSEIF (type == used_brake_disc)
           pa*((PI*angle)/180)*Ri*(Re-Ri)
          ELSE (Re*pa*b*(cos((angle0*PI)/180)-cos((angle*PI)/180)))/sin((angle_Pmax*PI)/180)

  --Geometric constant depending on the brake type.

  cgeo = IF (type == new_brake_disc)
          ((n_pad*2)/3)*((Re**3-Ri**3)/(Re**2-Ri**2))
         ELSEIF (type == used_brake_disc)
          (n_pad*(Re+Ri))/2
         ELSEIF (type == primary_secondary_shoes)
          (Re*(p_max+p_2))/p_max
         ELSE  2*Re
```

Figura 4.44: Fragmento de la parte continua de *R_Drum_and_disc_Brake_jagm* en el que se calcula el valor de las variables *fn_max* y *cgeo*.

Además, las ecuaciones que definen *fn_max* son una transcripción a EL de las expresiones que caracterizan la fuerza normal en frenos de disco y tambor (ver tabla 4.53 y 4.54).

Sin embargo, para comprender cómo se construyen las ecuaciones que definen la variable *cgeo* son necesarias algunas aclaraciones. En el caso de frenos de disco, la estructura de *cgeo* se explica fácilmente mediante la tabla 4.60.

Tabla 4.60: Origen de la variable *cgeo* para frenos de disco.

Valor de *type*	Par de frenada (para una pastilla)	Expresión para *cgeo*
new_brake_disc	$N = \frac{2}{3} \cdot \mu \cdot \frac{(R_e^3 - R_i^3)}{(R_e^2 - R_i^2)} \cdot F_n$	$cgeo = n\_pad \cdot \frac{2}{3} \cdot \frac{(R_e^3 - R_i^3)}{(R_e^2 - R_i^2)}$
used_brake_disc	$N = \frac{1}{2} \cdot \mu \cdot (R_e + R_i) \cdot F_n$	$cgeo = n\_pad \cdot \frac{1}{2} \cdot (R_e + R_i)$

Pero, en el caso de tener un freno de tambor con zapatas como la de la figura 4.39, la expresión que caracteriza el par de frenada en una zapata según la tabla 4.53 es:

$$N_{1\ zapata} = R_e \cdot \mu \cdot F_n \tag{4.40}$$

puesto, que:

$$F_n = \frac{p_a \cdot b \cdot R_e}{sen\theta_a} \cdot [\cos\theta_1 - \cos\theta_2] \tag{4.41}$$

entonces:

$$N_{1\,zapata} = \frac{\mu \cdot p_a \cdot b \cdot R_e^2}{sen\theta_a} \cdot [\cos\theta_1 - \cos\theta_2] \tag{4.42}$$

Si se particulariza la ecuación (4.42) para un freno de tambor tipo *primary_secondary_shoes* se tiene:

$$N_{tot} = \underbrace{\frac{\mu \cdot p\_max \cdot b \cdot R_e^2}{sen\theta_a} \cdot [\cos\theta_1 - \cos\theta_2]}_{\text{Par de frenada zapata primaria } (p_a = p\_max)} + \underbrace{\frac{\mu \cdot p\_2 \cdot b \cdot R_e^2}{sen\theta_a} \cdot [\cos\theta_1 - \cos\theta_2]}_{\text{Par de frenada zapata secundaria } (p_a = p\_2)} \tag{4.43}$$

Operando la expresión (4.43) y multiplicando y dividiendo por p_a se llega a la siguiente expresión:

$$N_{tot} = \mu \cdot \underbrace{\frac{R_e \cdot (p\_max + p\_2)}{p_a}}_{cgeo} \cdot \underbrace{\frac{p_a \cdot b \cdot R_e}{sen\theta_a} \cdot [\cos\theta_1 - \cos\theta_2]}_{F_n} \tag{4.44}$$

y como el valor de la variable *pa* cuando *type=primary_secondary_shoes* es *p_max* (ver figura 4.41), el valor de *cgeo* para este tipo de frenos es:

$$cgeo = \frac{R_e \cdot (p\_max + p\_2)}{p\_max} \tag{4.45}$$

Para el resto de frenos de tambor, es decir, para *twin_shoes_for* y *twin_shoes_back* la presión en las zapatas es la misma cada vez, por tanto el par de frenada total será:

$$N_{tot} = 2 \cdot R_e \cdot \mu \cdot F_n \tag{4.46}$$

con lo que en ambos casos la constante geométrica resulta ser:

$$cgeo = 2 \cdot R_e \tag{4.47}$$

Las expresiones que definen la constante geométrica para los distintos tipos de frenos de tambor queda resumida en la tabla 4.61.

Tabla 4.61: Expresiones que definen *cgeo* para frenos de tambor.

Valor de *type*	Expresión para *cgeo*
primary_secondary_shoes	$cgeo = \frac{R_e \cdot (p\_max + p\_2)}{p\_max}$
twin_shoes_for twin_shoes_back	$cgeo = 2 \cdot R_e$

Otro fragmento de la parte continua del modelo, se encarga de calcular la presión en las zapatas secundarias, es decir, calcular el valor de *p_2*. Para obtener este valor, el componente *R_Drum_and_disc_Brake_jagm* realiza un equilibrio de momentos en la articulación de una de las zapatas del freno simulado, suponiéndola primaria. De nuevo, si se supone una zapata como la de la figura 4.39 se tiene:

$$F = \frac{(N_n - N_{roz})}{c} \tag{4.48}$$

donde F representa la fuerza ejercida por el dispositivo de accionamiento sobre las zapatas (supuestas idénticas las zapatas primaria y secundaria), c es la distancia entre el punto de aplicación de la fuerza de accionamiento y la articulación y N_n y N_{roz} son el par producido por las fuerzas normales a la zapata y las de rozamiento respectivamente, y se calculan según las ecuaciones (4.49) y (4.50).

$$N_n = \frac{p\_max \cdot b \cdot R_e \cdot a}{sen\theta_a} \cdot \left(\frac{\theta}{2} - \frac{1}{4} \cdot sen\, 2\theta\right)_{\theta 1}^{\theta 2} \tag{4.49}$$

$$N_{roz} = \mu \cdot \frac{p\_max \cdot b \cdot R_e}{sen\theta_a} \cdot \left\{(-R_e \cdot \cos\theta)_{\theta 1}^{\theta 2} - a \cdot \left(\frac{1}{2} \cdot sen^2\theta\right)_{\theta 1}^{\theta 2}\right\} \tag{4.50}$$

Señalar que la presión utilizada en las ecuaciones anteriores es *p_max*, ya que se ha supuesto que la zapata es primaria.

El fragmento del componente que se encarga de calcular el valor de F se representa en la figura 4.45.

```
--Torque balance for a primary shoe defining the force F exerted to the shoes by the driving device.

F = ((N_n-N_f)/c)
N_n = ((a*p_max*b*Re)/sin((angle_Pmax*PI)/180))*((PI/(2*180))*(angle-angle0)- 0.25*(sin((angle*PI)/90)-sin((angle0*PI)/90)))
N_f=  ((linearInterp1D(mue_pos, phi')*p_max*b*Re)/sin((angle_Pmax*PI)/180))*((-Re)*(cos((angle*PI)/180)-cos((angle0*PI)/180))-
    -(a/2)*(sin((angle*PI)/180)**2-sin((angle0*PI)/180)**2))
```

Figura 4.45: Fragmento de la parte continua de *R_Drum_and_disc_Brake_jagm* en el que se calcula el valor de fuerza ejercida por el dispositivo de accionamiento de las zapatas.

Conocido el valor de *F*, calculado según la ecuación (4.48), se realiza de nuevo el equilibrio de momentos, pero esta vez en la zapata secundaria.

$$F = \frac{(N_n + N_{roz})}{c} \tag{4.51}$$

con N_n y N_{roz} :

$$N_n = \frac{p\_2 \cdot b \cdot R_e \cdot a}{sen\theta_a} \cdot \left(\frac{\theta}{2} - \frac{1}{4} \cdot sen\, 2\theta\right)_{\theta 1}^{\theta 2} \tag{4.52}$$

$$N_{roz} = \mu \cdot \frac{p\_2 \cdot b \cdot R_e}{sen\theta_a} \cdot \left\{(-R_e \cdot \cos\theta)_{\theta 1}^{\theta 2} - a \cdot \left(\frac{1}{2} \cdot sen^2\theta\right)_{\theta 1}^{\theta 2}\right\} \tag{4.53}$$

Finalmente el valor de *p_2* se obtiene de la ecuación (4.51) para las expresiones de N_n y N_{roz} dadas por las ecuaciones (4.52) y (4.53). En la figura 4.46 se representa el fragmento de *R_Drum_and_disc_Brake_jagm* en el que se obtiene *p_2*.

```
--Torque balance for a secondary shoe defining the pressure on the lining p_2.
F*c = ((a*p_2*b*Re)/sin((angle_Pmax*PI)/180))*((PI/(2*180))*(angle-angle0)- 0.25*(sin((angle*PI)/90)-sin((angle0*PI)/90))) +
      +((linearInterp1D(mue_pos, phi')*p_2*b*Re)/sin((angle_Pmax*PI)/180))*((-Re)*(cos((angle*PI)/180)-cos((angle0*PI)/180))-
      -(a/2)*(sin((angle*PI)/180)**2-sin((angle0*PI)/180)**2))
```

Figura 4.46: Fragmento de la parte continua de *R_Drum_and_disc_Brake_jagm* en el que se calcula el valor de la presión en zapatas secundarias *p_2*.

La ecuación que define la relación entre el par de frenada y el par de entrada y salida del componente se representa en la figura 4.47.

```
--Torque balance for the brake
0 = m_in.tau - m_out.tau - tau_friction
```

Figura 4.47: Equilibrio de momentos para el componente *R_Drum_and_disc_Brake_jagm*.

Además, el componente dispone de un fragmento continuo donde se calculan los valores de los pares de fricción estáticos *tau0* (aplicable para *imode=1/-1*) y *tau0_max* (aplicable para *imode=0*), y asignan valores a las variables *w_relfric* y *a_relfric* (Figura 4.48).

```
--Angular velocity and angular acceleration
w_relfric = phi'
a_relfric = phi''

--Friction torque for w_relfric=0.
tau0 = mue0*cgeo*fn
tau0_max = peak*tau0
```

Figura 4.48: Fragmento continuo que define las expresiones para *w_relfric, a_relfric, tau0* y *tau0_max.*

El código completo del elemento *R_Drum_and_disc_Brake_jagm* se presenta en la figura 4.49:

```
-------------------------------------------------------------------------------
-- Component R_Drum_and_disc_Brake_jagm
-------------------------------------------------------------------------------
-- Purpose:
--    To represent a frictional brake.
-------------------------------------------------------------------------------
COMPONENT R_Drum_and_disc_Brake_jagm IS_A R_AbsFriction, R_Rigid
"Frictional brake"

   PORTS
    IN analog_signal(n = 1) inPort  "Normal force signal"

   DATA
    ENUM Brake_type type = new_brake_disc          "Brake type"
    TABLE_1D mue_pos =((0,1), (0.5, 0.5))          "[w,mue] positive sliding friction coefficient: w_rel>=0 [-]"
    REAL peak = 1.1                                "Maximum value of mue for w_rel=0 [-]"
    REAL Ri                        UNITS "m"       "Disc brake internal radius [m]"
    REAL Re                        UNITS "m"       "Disc brake external radius or radius shoe [m]"
    REAL p_max                     UNITS "N/m^2"   "Maximum pressure on the lining [N/m^2]"
    REAL angle                     UNITS "°"       "Angle covered by the brake pad or by the shoe [°]"
    REAL angle0=0                  UNITS "°"       "Start angle of the shoe [°]"
    REAL angle_Pmax=90             UNITS "°"       "Maximun pressure angle in the shoe [°]"
    REAL w_i = 1                   UNITS "rad/s"   "Initial relative angular velocity [rad/s]"
    REAL n_pad                     UNITS "-"       "Number of brake pads [-]"
    REAL c                         UNITS "m"       "Distance between the force application point and the joint [m]"
    REAL b                         UNITS "m"       "Shoe width [m]"
    REAL a                         UNITS "m"       "Distance between de drum centre and the joint [m]"

  DECLS
   REAL fn_max                     UNITS "N"       "Maximum normal force [N]"
   REAL cgeo                       UNITS "-"       "Brake geometry constant [-]"
   REAL mue0                       UNITS "-"       "Friction coefficient for w=0 and forward sliding [-]"
   REAL fn                         UNITS "N"       "Normal force [N]"
   REAL p_2                        UNITS "N/m^2"   "Pressure on secondary shoes [N/m^2]"
   REAL pa                         UNITS "N/m^2"   "Auxiliary variable to determine the pressure depending on the brake type [N/m^2]"
   REAL F                          UNITS "N"       "Operating force [N]"
   REAL N_n                        UNITS "Nm"      "Normal forces torque [Nm]"
   REAL N_f                        UNITS "Nm"      "Friction forces torque [Nm]"

  INIT
   phi' = w_i

  DISCRETE
   --Discrete statements defining the activation of the brake depending on the boolean data free.
   WHEN(fn<= 0.0) THEN
     free = TRUE
   END WHEN

  --Discrete statements defining imode value when the the brake is active.
   WHEN(fn > 0.0) THEN
     free = FALSE
     IF(w_relfric > 0) THEN
        imode = 2 AFTER 0.0
     ELSE
        imode = -2 AFTER 0.0
     END IF
   END WHEN

  --Discrete statements defining the pressure on the shoes depending on the brake type and w_relfric.
   WHEN(w_relfric > 0) THEN
     IF (type == twin_shoes_back) THEN
        pa = p_2   AFTER 0.0
     ELSE
        pa = p_max AFTER 0.0
     END IF
   END WHEN

    WHEN(w_relfric < 0) THEN
      IF(type == twin_shoes_for) THEN
        pa = p_2  AFTER 0.0
      ELSE
        pa = p_max  AFTER 0.0
      END IF
    END WHEN

  CONTINUOUS

  --Geometric constant depending on the brake type.

  cgeo = IF (type == new_brake_disc)
          ((n_pad*2)/3)*((Re**3-Ri**3)/(Re**2-Ri**2))
         ELSEIF (type == used_brake_disc)
          (n_pad*(Re+Ri))/2
         ELSEIF (type == primary_secondary_shoes)
          (Re*(p_max+p_2))/p_max
         ELSE  2*Re

  --Constant auxiliary variable
  mue0 = linearInterp1D(mue_pos, 0.0)

  --Angular velocity and angular acceleration

  w_relfric = phi'
  a_relfric = phi''
```

```
  --Torque balance for the brake

  0 = m_in.tau - m_out.tau - tau_friction

  --Maximun normal force depending on the brake type.

  fn_max = IF (type == new_brake_disc)
            (0.5*pa*((PI*angle)/180)*(Re**2-Ri**2))
           ELSEIF (type == used_brake_disc)
            pa*((PI*angle)/180)*Ri*(Re-Ri)
           ELSE (Re*pa*b*(cos((angle0*PI)/180)-cos((angle*PI)/180)))/sin((angle_Pmax*PI)/180)

  --Torque balance for a primary shoe defining the force F exerted to the shoes by the driving device.

  F = ((N_n-N_f)/c)
  N_n = ((a*p_max*b*Re)/sin((angle_Pmax*PI)/180))*((PI/(2*180))*(angle-angle0)- 0.25*(sin((angle*PI)/90)-sin((angle0*PI)/90)))
  N_f=  ((linearInterp1D(mue_pos, phi')*p_max*b*Re)/sin((angle_Pmax*PI)/180))*((-Re)*(cos((angle*PI)/180)-cos((angle0*PI)/180))-
        -(a/2)*(sin((angle*PI)/180)**2-sin((angle0*PI)/180)**2))

 --Torque balance for a secondary shoe defining the pressure on the lining p_2.

 F*c = ((a*p_2*b*Re)/sin((angle_Pmax*PI)/180))*((PI/(2*180))*(angle-angle0)- 0.25*(sin((angle*PI)/90)-sin((angle0*PI)/90))) +
     +((linearInterp1D(mue_pos, phi')*p_2*b*Re)/sin((angle_Pmax*PI)/180))*((-Re)*(cos((angle*PI)/180)-cos((angle0*PI)/180))-
     -(a/2)*(sin((angle*PI)/180)**2-sin((angle0*PI)/180)**2))


  --Normal force

   fn = fn_max*inPort.signal[1]

  --Friction torque for w_relfric=0.

   tau0 = mue0*cgeo*fn
   tau0_max = peak*tau0

  --Friction torque representing the braking torque.

   tau_friction =   ZONE(imode == 0) sa
                    ZONE(imode == 1) cgeo*fn*linearInterp1D(mue_pos, phi')
                    ZONE(imode == -1) -cgeo*fn*linearInterp1D(mue_pos, -phi')
                    ZONE(imode == 2) cgeo*fn * linearInterp1D(mue_pos, phi')
                    ZONE(imode == -2) - cgeo*fn*linearInterp1D(mue_pos, -phi')
                    OTHERS 0.0

END COMPONENT
```

Figura 4.49: Código del componente *R_Drum_and_disc_Brake_jagm* en EL.

4.9. Creación de *R_Hydro_Bearing*

En las máquinas es frecuente encontrar interacciones entre piezas con deslizamiento relativo en las que es necesario reducir la fricción y minimizar el desgaste. Esto se consigue mediante unos elementos denominados cojinetes.

Un cojinete es un dispositivo que permite el movimiento relativo entre superficies, minimizando la pérdida de energía y el desgaste de las mismas.

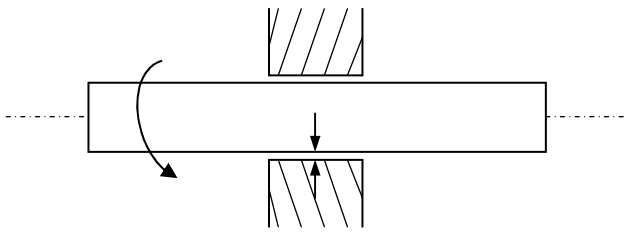


Figura 4.50: Esquema básico de cojinete cilíndrico.

Existen diferentes disposiciones constructivas y múltiples regímenes de lubricación para satisfacer los requerimientos cinemáticos y de carga de un cojinete pero, de entre todas

ellas, el componente *R_Hydro_Bearing* representará un cojinete cilíndrico, como el de la figura 4.50, con lubricación hidrodinámica.

La lubricación consiste en interponer una película de lubricante entre dos superficies con movimiento relativo, de forma que las pérdidas de energía no sean consecuencia del rozamiento entre superficies sólidas, sino de efectos de viscosidad en el fluido lubricante debido a tensiones de cortadura. En el caso de lubricación hidrodinámica, la película de lubricante es presurizada mediante efectos hidrodinámicos producidos por la combinación de una disposición geométrica y un movimiento relativo entre superficies adecuados y por el carácter viscoso del fluido.

Los orígenes de la teoría de lubricación hidrodinámica se remontan a finales del siglo XIX, como consecuencia de los estudios del investigador Beauchamp Tower acerca de la lubricación en los cojinetes de ejes de ferrocarriles. Tower abrió un orificio en la parte superior de unos de los cojinetes que ensayaba y descubrió que las presiones que el lubricante alcanzaba dentro del mismo eran muy elevadas [SAN12].

Los resultados obtenidos por Tower, llevaron a Osborne Reynolds a pensar que debía existir una ley que relacionara la presión del fluido, la velocidad relativa y el coeficiente de fricción, obteniendo una ecuación diferencial que, aunque no tiene una solución general, sigue siendo el punto de partida para los actuales estudios de lubricación [LOP12].

Sin embargo, A.A. Raimondi y John Boyd, de los Whestinhouse Research Laboratories, emplearon técnicas de iteración por ordenador para resolver la ecuación de Reynolds. Los resultados fueron representados en forma de gráficos, como el de la figura 4.51, que utilizan en abcisas el número de Sommerfeld o número característico del cojinete [SAN12]:

$$S = \left(\frac{r}{c}\right)^2 \frac{\eta N}{P} \tag{4.54}$$

Donde r es el radio del muñón, c la holgura radial, η la viscosidad absoluta, N la velocidad relativa entre el muñón y el cojinete y P la carga por unidad de área proyectada.

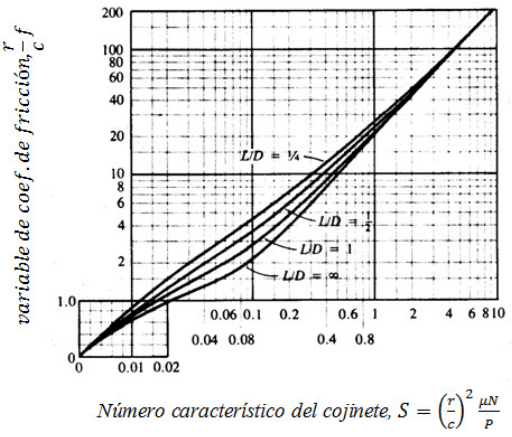
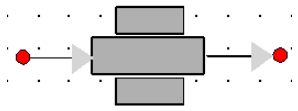


Figura 4.51: Diagrama para determinar la variable de fricción $\frac{r}{c}f$.

La descripción de *R_Hydro_Bearing* se muestra en la tabla 4.62.

Tabla 4.62: Descripción de *R_Hydro_Bearing*.

Nombre	Descripción	Representación gráfica
R_Hydro_Bearing	Define un cojinete cilíndrico con lubricación hidrodinámica .	

Puesto que *R_Hydro_Bearing* hereda las características de *R_Rigid*, está formado por dos puertos de tipo *mech_rot,* que se describen en la tabla 4.63.

Tabla 4.63: Descripción de los puertos de *R_Hydro_Bearing*.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Las constantes y variables que se introducen en este elemento se muestran en la tabla 4.64 y 4.65 respectivamente.

Tabla 4.64: Descripción de las constantes introducidas por *R_Hydro_Bearing*.

Dato	Tipo	Descripción	Valor inicial	Unidades
c	REAL	Define la holgura radial.	-	[mm]
v	REAL	Define la viscosidad del lubricante.	-	[Pa·s]
l	REAL	Define la longitud del cojinete.	-	[mm]
d	REAL	Define el diámetro del muñón del cojinete.	-	[mm]
W	REAL	Define la carga que soporta el cojinete.	-	[N]

Tabla 4.65: Descripción de las variables que introduce *R_Hydro_Bearing*.

Variable	Tipo	Descripción	Unidades
P	REAL	Define la carga que soporta el cojinete por unidad de área proyectada.	[N/m^2]
S	REAL	Define el número de Sommerfeld.	[-]
H	REAL	Define la potencia perdida en el cojinete.	[W]

f	REAL	Define el coeficiente de fricción equivalente (estos datos se hallaran mediante tablas)	[-]
Table1D	REAL	Tabla que incluye los valores del término $\frac{r}{c}f$ en función del número de Sommerfeld y de la relación de aspecto l/d.	[]

El componente *R_Hydro_Bearing* comienza con el fragmento de código de la figura 4.52, mediante el cual se asignan los valores a la variable *Table1D* en función de la relación $\frac{l}{d}$ seleccionada.

```
--Assign values to Table1d  depending on the relationship l/d
INIT
   IF(l/d == 1) THEN
      readTable1D("f_1.txt",table1, 1)
  ELSEIF(l/d == 1/4) THEN
      readTable1D("f_1-4.txt",table1, 1)
  ELSEIF(l/d == 1/4) THEN
      readTable1D("f_1-2.txt",table1, 1)
  ELSEIF(l/d == 1/4) THEN
      readTable1D("f_inf.txt",table1, 1)
  END IF
```

Figura 4.52: Inicialización de la variable *Table1D*.

Así, si la relación de aspecto equivale a la unidad, se asigna a *Table1D* los valores de la tabla *f_1.txt*, para relaciones de aspecto de 1/4, se asigna a *Table1D* los valores de la tabla *f_1-4.txt,* para relaciones de aspecto de 1/2, se asigna a *Table1D* los valores de la tabla *f_1-2.txt* y por último, cuando la longitud es mucho mayor que el diámetro, se asigna a *Table1D* los valores de la tabla *f_inf.txt*. De esta forma se consigue que el componente sea capaz de leer el diagrama de la figura 4.51, en el que se obtiene el valor de la variable de fricción a partir del número de *Sommerfeld* y a relación de aspecto del cojinete.

Las tablas *f_1.txt, f_1-4.txt, f_1-2.txt y f_inf.txt,* se presentan en la tabla 4.66:

Tabla 4.66: Valores para la variable de fricción; a) *l/d* = 1 ; b) l/d = 1/4; c)*l/d=1/2*; d)*l/d*=∞.

a)

l/d = 1	
S	$\frac{r}{c}f$
0.000	0.000
0.002	0.250
0.004	0.450
0.006	0.600
0.008	0.750
0.010	0.900

b)

l/d = 1/4	
S	$\frac{r}{c}f$
0.000	0.000
0.002	0.250
0.004	0.450
0.006	0.550
0.008	0.650
0.010	0.760

0.012	1.000
0.020	1.400
0.030	1.800
0.040	2.150
0.045	2.800
0.060	3.200
0.080	4.250
0.100	4.500
0.200	7.250
0.400	12.000
0.600	17.500
0.800	20.000
1.000	24.000
1.200	28.000
1.400	32.000
1.600	38.000
2.000	45.000
2.400	57.500
4.000	90.000
6.000	120.000
8.000	160.000
10.000	200.000

c)

l/d = 1/2	
S	$\frac{r}{c}f$
0.000	0.000
0.002	0.250
0.004	0.450
0.006	0.600
0.009	0.800
0.010	0.800
0.012	0.950
0.014	1.000
0.016	1.150
0.020	1.300
0.030	1.600
0.040	1.900
0.050	2.190
0.060	2.400
0.080	3.000
0.100	3.600

0.012	0.850
0.020	1.160
0.030	1.380
0.040	1.580
0.044	1.670
0.048	1.780
0.052	1.900
0.056	1.990
0.060	2.010
0.080	2.400
0.100	2.800
0.120	3.100
0.140	3.500
0.160	3.900
0.200	4.600
0.240	5.400
0.300	6.500
0.400	8.250
0.488	10.000
0.600	12.000
0.700	14.000
0.800	16.000
1.000	20.000
1.400	28.000
1.600	32.000
2.000	40.000
10.000	200.000

d)

l/d = ∞	
S	$\frac{r}{c}f$
0.000	0.000
0.002	0.150
0.004	0.300
0.006	0.450
0.008	0.550
0.010	0.650
0.012	0.720
0.014	0.800
0.016	0.825
0.018	0.875
0.020	0.900
0.022	0.950
0.024	1.000
0.026	1.000
0.028	1.050
0.030	1.100

0.200	5.200
0.300	7.500
0.400	9.500
0.600	14.000
0.800	17.500
1.000	22.000
1.200	26.000
1.400	32.000
1.600	38.000
2.000	45.000
2.400	57.500
4.000	90.000
6.000	120.000
8.000	160.000
10.000	200.000

0.032	1.120
0.034	1.150
0.036	1.190
0.038	1.200
0.040	1.210
0.045	1.300
0.050	1.390
0.055	1.400
0.060	1.480
0.065	1.550
0.070	1.600
0.075	1.650
0.080	1.750
0.085	1.800
0.090	1.850
0.095	1.950
0.100	2.000
0.120	2.350
0.140	2.700
0.160	3.000
0.180	3.420
0.200	3.800
0.220	4.300
0.240	4.500
0.260	4.800
0.280	5.250
0.300	5.500
0.320	5.900
0.340	6.400
0.360	6.750
0.380	7.100
0.400	7.500
0.450	8.500
0.500	9.500
0.550	10.000
0.600	14.000
0.800	17.500
1.000	22.000
1.200	26.000
1.400	32.000
1.600	38.000
2.000	45.000
2.400	57.500
4.000	90.000
6.000	120.000
8.000	160.000
10.000	200.000

Ahora que el componente *R_Hydro_Bearing* puede interpretar qué curva de la figura 4.51 tiene que utilizar, es posible determinar el valor del coeficiente de fricción *f*. Para ello se realiza una interpolación lineal en la tabla *Tabla_1D* para el valor del número característico del cojinete *S* obtenido según la ecuación (4.54), donde *P* se calcula como sigue:

$$P = W/(l \cdot d) \tag{4.55}$$

El resultado obtenido en la interpolación permite obtener el valor del coeficiente de fricción para el cojinete simulado.

$$f = \frac{r}{c} \cdot Interpolación(Table\_1D, S) \tag{4.56}$$

Una vez conocido el valor de *f,* la potencia perdida en el cojinete se calcula mediante la ecuación (4.57), donde *r* representa el radio del cojinete, *W* la carga que soporta el cojinete, *phi'* la velocidad angular del cojinete y *f* representa el coeficiente de rozamiento equivalente.

$$H = f \cdot W \cdot phi' \cdot r \tag{4.57}$$

A partir de la pérdida de potencia *H*, y mediante un equilibrio de momentos en el cojinete, se determina el par a la salida del cojinete.

El código en lenguaje de programación EL que representa el componente se muestra en la figura 4.53:

```
COMPONENT R_Hydro_Bearing IS_A R_Rigid
"Cylindrical bearing with hydrodinamic lubrication"

    DATA
      REAL c   UNITS "mm"        "Radial play [mm]"
      REAL v   UNITS "Pa*s"      "Lubricant viscosity [Pa*s]"
      REAL l   UNITS "mm"        "Bearing length [mm]"
      REAL d   UNITS "mm"        "Bearing diameter [mm]"
      REAL W   UNITS "N"         "Bearing load [N]"

    DECLS
       REAL S   UNITS "[]"       "Sommerfeld number[]"
       REAL P   UNITS "N/m**2"    "Bearing load on proyected area [N/m**2]"
       REAL H   UNITS "W"        "Power loss [W]"
       REAL f   UNITS "[]"       "Equivalent friction coeffincient []"
       TABLE_1D table1

    --Assign values to Table1d  depending on the relationship l/d
    INIT
       IF(l/d == 1) THEN
          readTable1D("f_1.txt",table1, 1)
      ELSEIF(l/d == 1/4) THEN
          readTable1D("f_1-4.txt",table1, 1)
      ELSEIF(l/d == 1/4) THEN
          readTable1D("f_1-2.txt",table1, 1)
      ELSEIF(l/d == 1/4) THEN
          readTable1D("f_inf.txt",table1, 1)
      END IF


    CONTINUOUS

    --Load on projected area
    P=(W/(l*d*(10**-6)))

    --Sommerfeld number
    S=((d/(2*c))**2)*((v*m_in.phi')/(2*P*MATH_ELEMENTS.PI))

    --Friction coef. depending on Sommerfeld number
    f=linearInterp1D(table1,S)*((2*c)/d)

    --Power losses
    H=f*W*m_in.phi'*(d/2000)

    --Torque balance for the bearing
    m_out.tau+(H/m_in.phi')=m_in.tau

END COMPONENT
```

Figura 4.53: Código del componente *R_Hydro_Bearing* en EL.

4.10. Creación de *R_GearIdeal*

El componente R_GearIdeal, pretende modelar el comportamiento de un engranaje, entendido éste como aquel elemento que se utiliza para transmitir potencia de un eje a otro de una máquina. Por tanto, estos elementos permiten modificar la velocidad de giro y el par transmitido en virtud de un parámetro denominado relación de transmisión, que se obtiene a partir del cociente entre la velocidad de entrada y salida del engranaje.

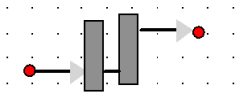
$$i = \frac{w_{entrada}}{w_{salida}} [-] \tag{4.58}$$

Además y dado que la potencia a un lado y a otro del engranaje debe permanecer constante, se tiene la siguiente relación entre los pares de entrada y salida:

$$T_{salida} = i \cdot T_{entrada} \tag{4.59}$$

La descripción de *R_GearIdeal* se muestra en la tabla 4.67.

Tabla 4.67: Descripción de *R_GearIdeal*.

Nombre	Descripción	Representación gráfica
R_GearIdeal	Define un engranaje con una relación de transmisión dada.	

El componente *R_GearIdeal* hereda de *R_Two_Ports*, por tanto tendrá dos puertos tipo *mech_rot ,* los cuales se describen en la tabla 4.68.

Tabla 4.68: Descripción de los puertos de *R_GearIdeal*.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

La única constante que se introduce en este componente es la relación de transmisión:

Tabla 4.69: Constante introducida por *R_GearIdeal*.

Dato	Tipo	Descripción	Valor inicial	Unidades
ratio	REAL	Define la relación de transmisión.	1	[mm]

El código del componente se define en la figura 4.54.

```
-------------------------------------------------------------------------------
-- Component R_GearIdeal
-------------------------------------------------------------------------------

COMPONENT R_GearIdeal IS_A R_Two_Ports
"Ideal Gear without inertia"
   DATA
     REAL ratio = 1    "Transmission ratio (-)"

   CONTINUOUS
   m_in.phi' = ratio * m_out.phi'
   ratio * m_in.tau = m_out.tau

END COMPONENT
```

Figura 4.54: Código del componente *R_GearIdeal* en EL.

CAPÍTULO 5

SIMULACIÓN DE SISTEMAS DINÁMICOS

El objetivo fundamental de este proyecto consiste en desarrollar una librería de elementos rotacionales con el fin de poder simular sistemas dinámicos rotacionales.

Una vez implementados los modelos de los elementos que componen la librería rotacional, el siguiente paso consiste en comprobar que reproducen fielmente el comportamiento de los elementos físicos que representan, esto es validar los elementos creados. Para ello se modelarán y simularán diferentes sistemas (utilizando los elementos que forman la librería rotacional) mediante EcosimPro, y los resultados se compararán con los que se obtengan al simular los mismos modelos con Simulink o mediante las expresiones analíticas que los caracterizan.

5.1. Sistema dinámico con dos grados de libertad

En esta sección se modelará un sistema dinámico con dos grados de libertad como el de la figura 5.1 mediante los componentes de la librería *Rotational_Library* implementada en EcosimPro.

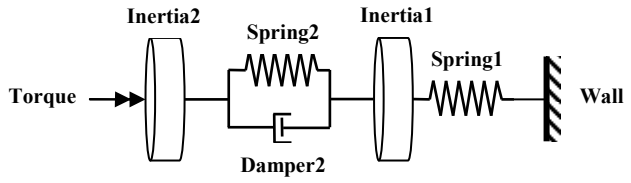


Fig. 5.1: Croquis sistema dinámico con dos grados de libertad.

Posteriormente se simulará dicho modelo y los resultados serán comparados con los resultados obtenidos de la simulación del mismo sistema mediante la herramienta *Simulink*.

Simulink es una aplicación que permite construir y simular modelos de sistemas físicos y sistemas de control mediante diagramas de bloques. El comportamiento de dichos

sistemas se define mediante funciones de transferencia, operaciones matemáticas, elementos de *Matlab* y señales predefinidas de todo tipo.

En la figura 5.2 se puede observar el modelo del sistema con dos grados de libertad, en el entorno gráfico de EcosimPro.

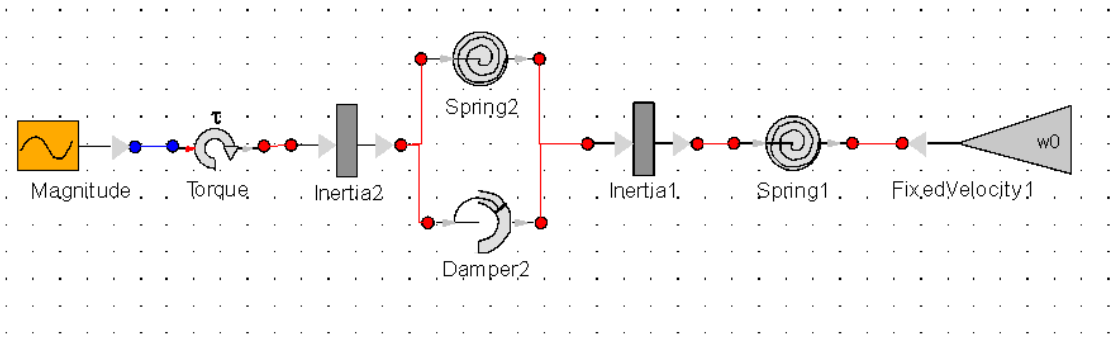


Fig. 5.2: Croquis sistema dinámico *Dos grados de libertad* en Ecosimpro.

El sistema está formado por una fuente analógica de señales (*Magnitude*), un actuador de momento (*Torque*), dos inercias (*Inertia1 e Inertia2*), dos muelles torsionales (*Spring1 y Spring2*), un amortiguador torsional (*Damper2*) y una restricción de velocidad (*FixedVelocity1*).

La fuente analógica emite una señal, que el actuador de momento transforma en una par de torsión y éste es transmitido al resto del sistema. En el otro extremo, el elemento *FixedVelocity1*, permite imponer una velocidad de giro al extremo derecho del elemento *Spring1*.

Tras crear el modelo del sistema a simular, el siguiente paso consiste en instanciar todos los elementos que lo componen.

En primer lugar se instancia la fuente analógica. En este caso se quiere una función impulso de amplitud *500000 [-]*[13], con un ancho de escalón de medio segundo y con un periodo de *100 [s]*. En la figura 5.3 se muestran los atributos modificados de la fuente analógica.

[13] La amplitud es adimensional, ya que las unidades de la señal emitida por la fuente analógica se determinan mediante el actuador que la acompaña (en este caso el actuador *Torque*).

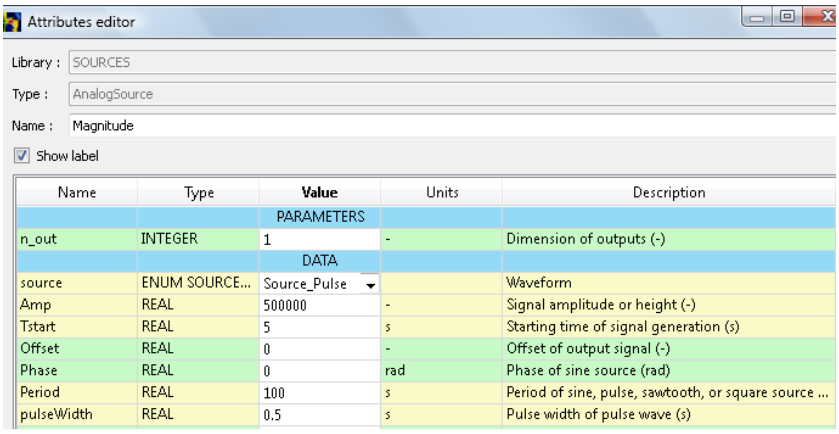


Fig. 5.3: Atributos modificados de la fuente analógica.

Del mismo modo se procede a instanciar el resto de los elementos del sistema. En la tabla 5.1, se representan las instancias realizadas sobre cada uno de los elementos del sistema.

Tabla. 5.1: Instanciación de los elementos del sistema "dos grados de libertad".

Elemento	Parámetro o dato	Tipo	Valor	Unidades
Inertia1/Inertia2	*I*	REAL	4000	[kg·m^2]
	phi_0	REAL	0	[rad]
Spring2/ Spring1	*phi_rel_0*	REAL	0	[rad]
	k_tor	REAL	157913.4	[N·m/rad]
	phi_natural	REAL	0	[rad]
Damper2	*phi_rel_0*	REAL	0	[rad]
	d_tor	REAL	12566.36	[N·m·s/rad]
FixedVelocity1	*w0*	REAL	0	[rad/s]

Como se puede observar en la tabla 5.1 los elementos *Inertia1* e *Inertia2*, así como *Spring1* y *Spring2* son equivalentes entre sí. Además, todos los elementos parten de un estado inicial, en el que la velocidad angular y la posición angular son iguales a cero.
Por otro lado, la instancia del parámetro *w0* del elemento *FixedVelocity* es nula, así se garantiza que durante todo el experimento, tanto la posición como la velocidad del extremo derecho del elemento *Spring1* serán nulos, es decir, ese punto no se desplaza, está fijo.

Los resultados obtenidos al simular el sistema "dos grados de libertad" mediante Ecosimpro serán comparados con los obtenidos al simular el sistema de la figura 5.4 mediante Simulink. Es evidente que para obtener resultados comparables, las instancias de los elementos del modelo de Simulink deben de ser las mismas que las de la figura 5.2 de EcosimPro.

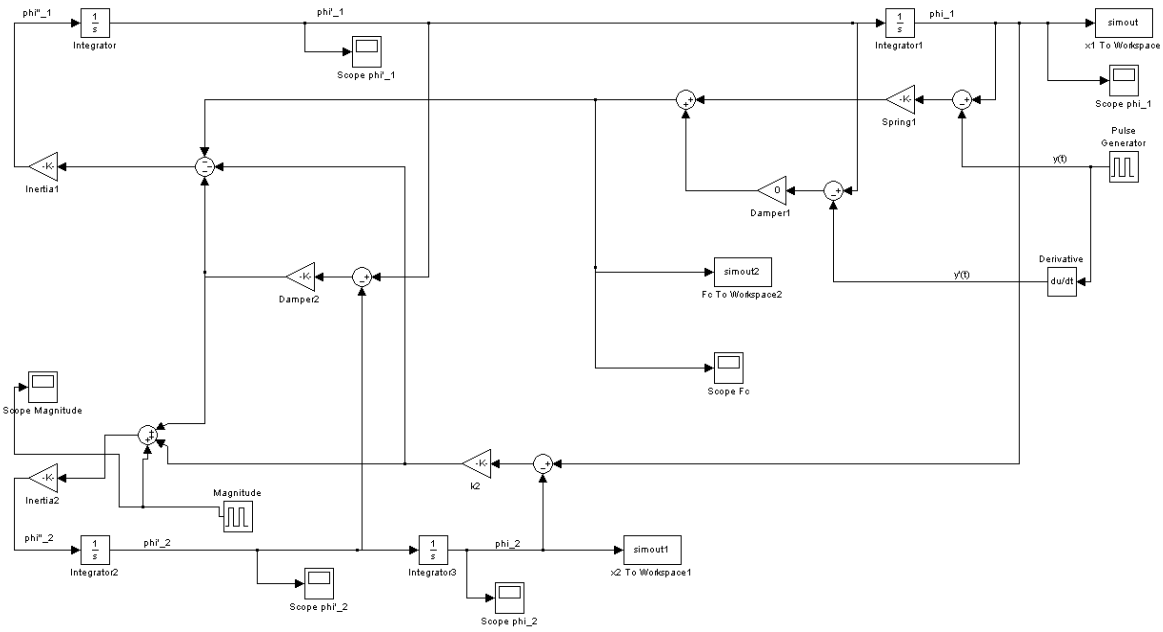


Fig. 5.4: Croquis sistema dinámico "dos grados de libertad" en Simulink.

El primer dato a comparar será el par de torsión que excita el sistema. Si las señales que excitan ambos sistemas no coinciden será difícil poder extraer conclusiones comparando ambos modelos.
En la figura 5.5 se representa el valor de la variable *Torque.m_out.tau* frente al tiempo. Como se indicó en la figura 5.3, se trata de un pulso de amplitud *500000 [N·m]* que actúa transcurridos *5[s]* con una duración de *0,5[s]*. Como el periodo de esta señal es mayor que la duración del experimento, el pulso sólo actúa en una ocasión.

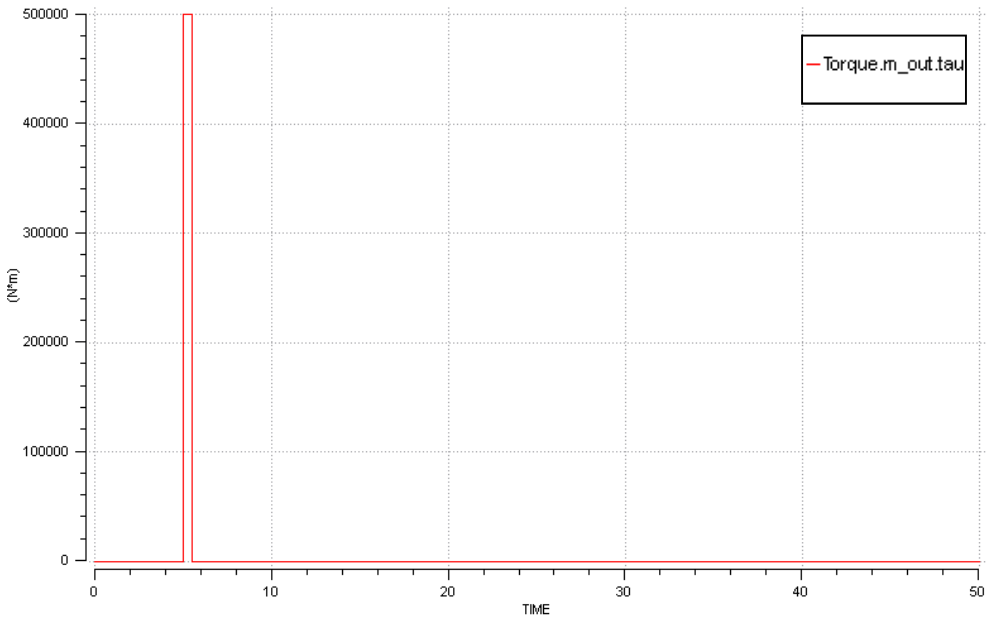


Fig. 5.5: Representación de la variable *Torque.m_out.tau* frente al tiempo.

Como se observa en la figura 5.6 el pulso que actúa en el modelo de Simulink es idéntico al de la figura 5.5, con lo que se puede proceder a comparar los resultados de ambos modelos.

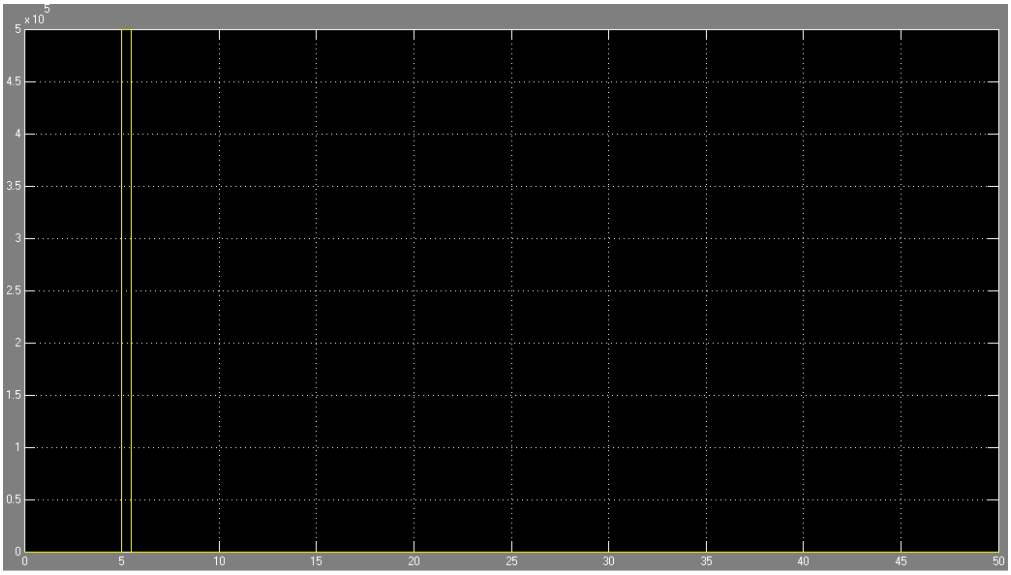


Fig. 5.6: Representación temporal de la señal que excita el modelo de Simulink.

En primer lugar se compararán las posiciones de los dos elementos inerciales (*Inertia1* e *Inertia2*). Debido a que estos elementos pertenecen a la clase abstracta *R_Rigid*, la posición angular de los puertos de salida debe ser la misma que la de sus puertos de entrada, es decir:

$$Inertia1.m\_in.phi = Inertia1.m\_out.phi = Inertia1.phi \quad (5.1)$$

$$Inertia2.m\_in.phi = Inertia2.m\_out.phi = Inertia2.phi \quad (5.2)$$

Por tanto, y de acuerdo con las ecuaciones 5.1 y 5.2, al representar las variables *Inertia1.m_in.phi*, *Inertia1.m_out.phi*, *Inertia1.phi, Inertia2.m_in.phi, Inertia2.m_out.phi* e *Inertia2.phi*. en un mismo gráfico (ver figura 5.7), sólo deben aparecer dos curvas, una de las cuales representa en ángulo girado por el elemento *Inertia1* y otra para el elemento *Inertia2.*

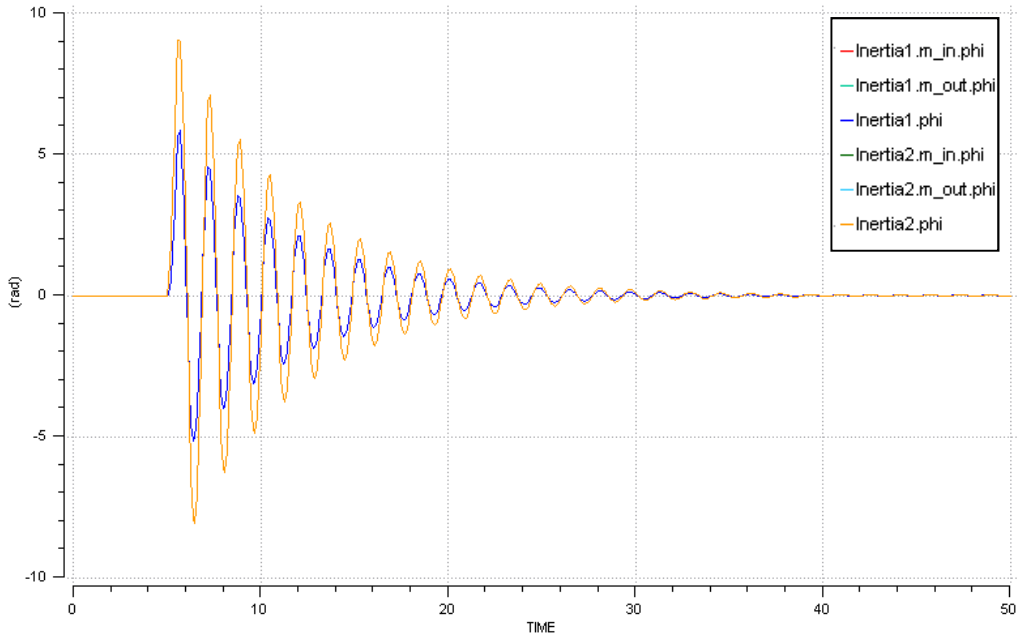


Fig. 5.7. Evolución temporal de las variable Inertia1_phi e Inertia2_phi en Ecosimpro.

Los resultados obtenidos en la simulación muestran que la amplitud de las oscilaciones del elemento *Inertia2* son mayores que las del elemento *Inertia1*, lo que era de esperar al encontrarse el elemento *Inertia2* más próximo a la fuente. Además, la presencia del elemento *Damper2* da lugar una disipación de energía que reduce en el tiempo la amplitud de las oscilaciones de los elementos *Inertia1* e *Inertia2*. En la figura 5.8 se muestran los resultados que se obtienen al simular el modelo con Simulink.

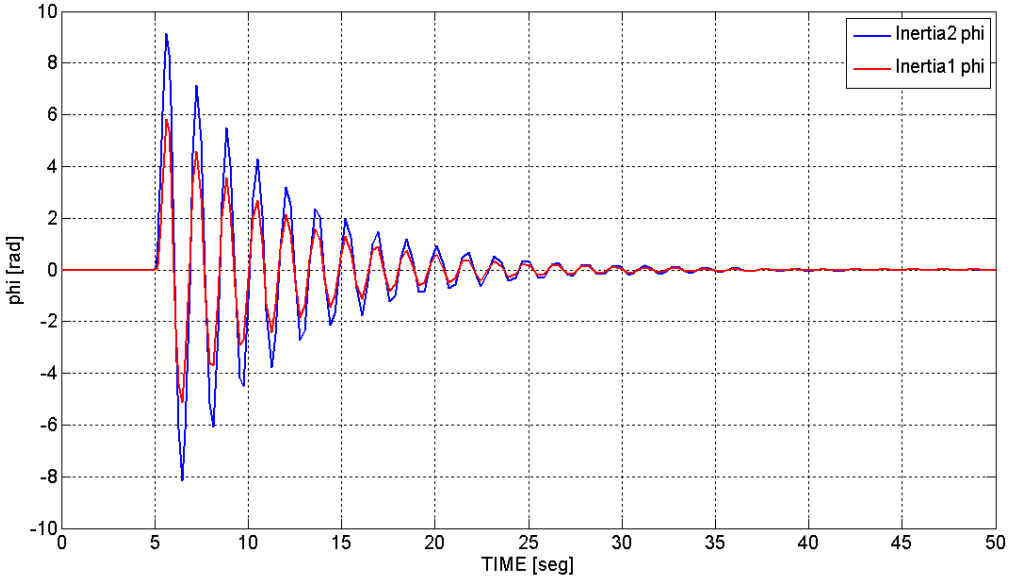
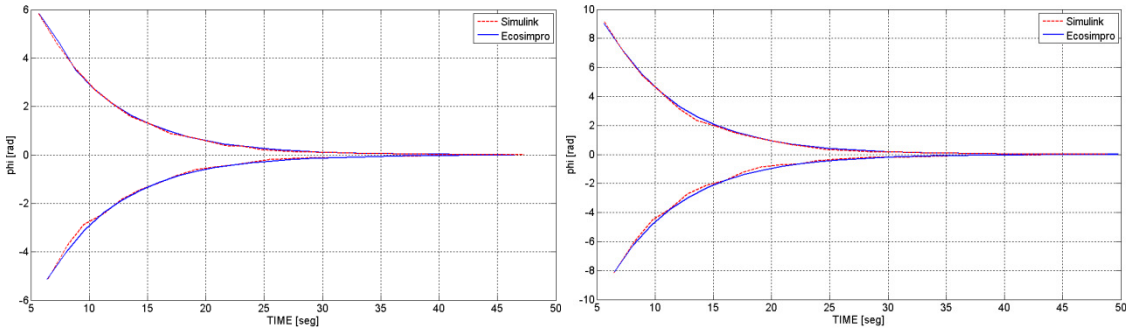


Fig. 5.8: Evolución temporal de las variables *Inertia1_phi* e *Inertia2_phi* en Simulink.

Aunque en las figuras 5.7 y 5.8 ya se observa un comportamiento muy similar de ambos modelos, dicha similitud se hace más evidente en las figuras 5.9.a y 5.9.b, en las que se

representan las curvas que unen los puntos dónde la velocidad angular de los elementos *Inertia1* e *Inertia2* cambia de signo. Un solapamiento casi total durante el experimento, confirma la validez de los modelos implementados en Ecosimpro.



a) b)

Fig. 5.9: Evolución temporal de las posiciones angulares de cambio de signo para Ecosimpro y Simulink para: a) elemento *Inertia1*; b) elemento *Inertia2*.

El siguiente paso para comprobar la validez de los modelos implementados en Ecosimpro, será modificar el valor de las instancias de algunos elementos y deducir, antes de lanzar el experimento, cuál debe ser el comportamiento del modelo.

La primera instancia a modificar será la constante de amortiguamiento *d_tor,* que aparece en el elemento *Damper2* (*R_Damper* en la librería *Rotational_Library*). La constante de amortiguamiento es una magnitud adimensional que mide cómo decaen las oscilaciones de un sistema después de ser perturbado, de manera que cuanto mayor es la constante de amortiguamiento, más rápido debe estabilizarse el sistema. En la tabla 5.2 se muestran los valores de la posición angular de los elementos *Inertia1* e *Inertia2* en un instante cercano a *t=30 [seg]*.

Tabla. 5.2: Amplitud de las oscilaciones para dos valores distintos de *d_tor*.

		phi[rad] (*d_tor*=12566,36)	**phi[rad]** (*d_tor*=25132,27)
Inertia1	Ecosimpro	0,1292	0,0102
	Simuink	0,1215	0,0128
Inertia2	Ecosimpro	0,2041	0,0145
	Simulink	0,1738	0,0170

Los valores de la tabla anterior, junto con la figura 5.10 en la que se representa la evolución temporal de las posiciones angulares de *Inertia1* e *Inertia2* para un valor de *d_tor* de *25132,27[-]*, confirman que un aumento de la constante *d_tor* provoca la respuesta esperada a priori, esto es, la amplitud de las oscilaciones decaen a medida que aumenta la constante de amortiguamiento. Además, la similitud entre los resultados que se obtienen con ambas herramientas, corroboran que el comportamiento del elemento *R_Damper* es adecuado.

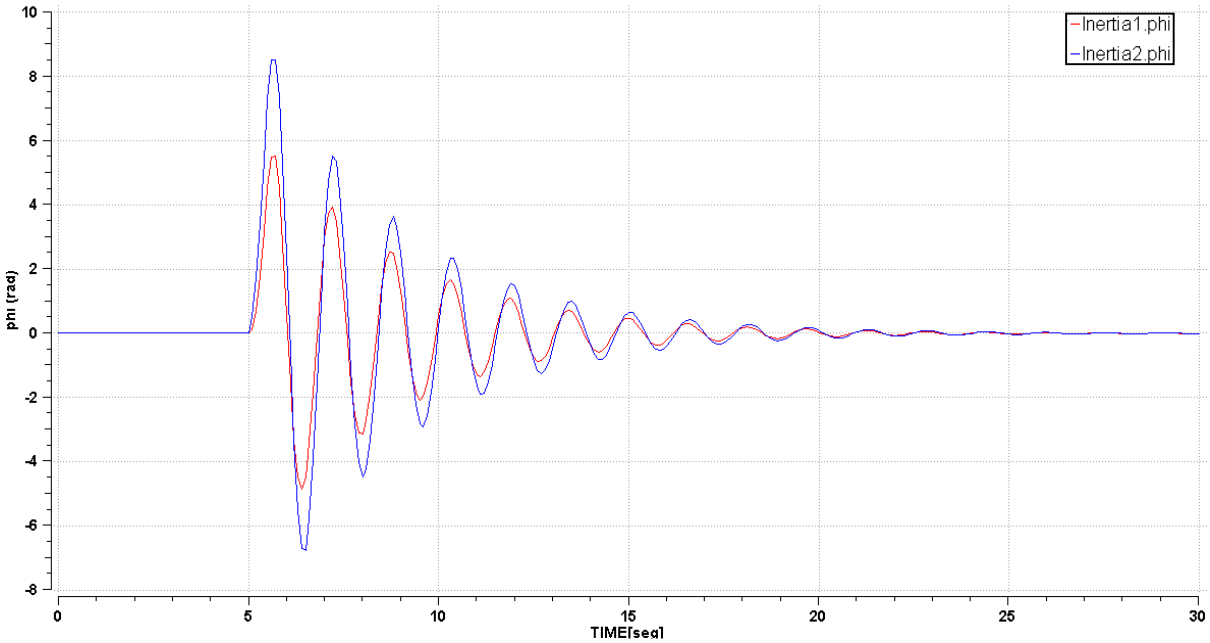


Fig. 5.10: Evolución temporal de las variables *Inertia1_phi* e *Inertia2_phi* en Ecosimpro para *d*=25132,27[-]

Si un aumento de la constante de amortiguamiento disminuye el tiempo que emplea el sistema en alcanzar el equilibrio, un decremento de la misma debe provocar el efecto contrario. La figura 5.11 muestra cómo responde el sistema *dos grados de libertad* para un valor unitario de *d_tor*.

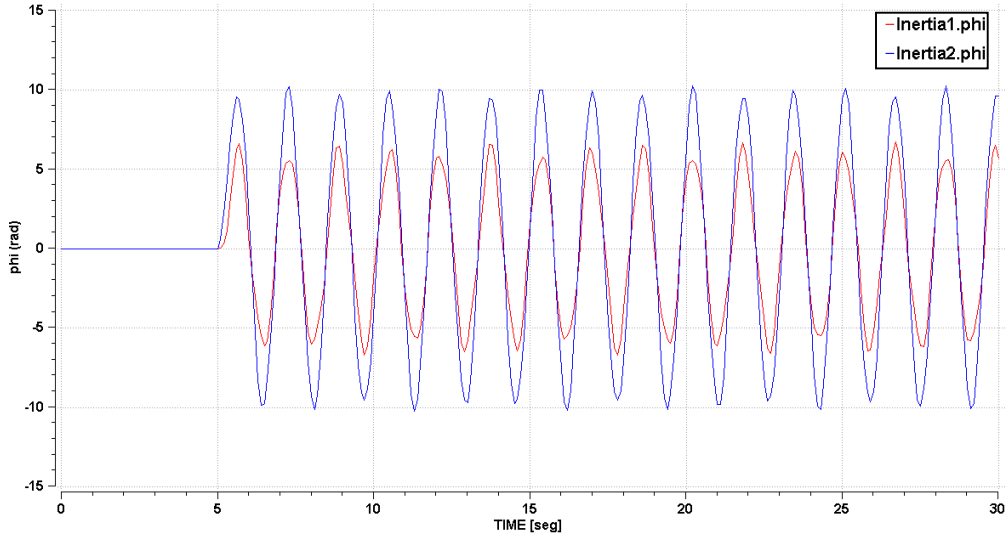


Fig. 5.11: Evolución temporal de las variables Inertia1_phi e Inertia2_phi en Ecosimpro para un valor unitario de *d_tor*.

En el caso de la figura 5.11 se observa que al no existir ningún elemento que disipe la energía almacenada en los resortes, el sistema oscila contínuamente en virtud de las constantes elásticas de los elementos *R_Spring1* y *R_Spring2*.

Además, cuanto mayor sea el valor de la constante elástica de los resortes torsionales, mayor será la rigidez que ofrezcan éstos y por tanto, para una misma excitación, el sistema debería disminuir la amplitud de sus oscilaciones si *k_tor* aumenta y aumentar dicha amplitud en caso de que *k_tor* disminuya. En la figura 5.12 se muestran distintos experimentos de los modelos *dos grados de libertad* de Ecosimpro y Simulink en los que se pueden apreciar los efectos de modificar la constante elástica del *Spring_2* (ver figura 5.1), para un valor unitario de la constante de amortigamiento de *Damper_2* (ver figura 5.1).

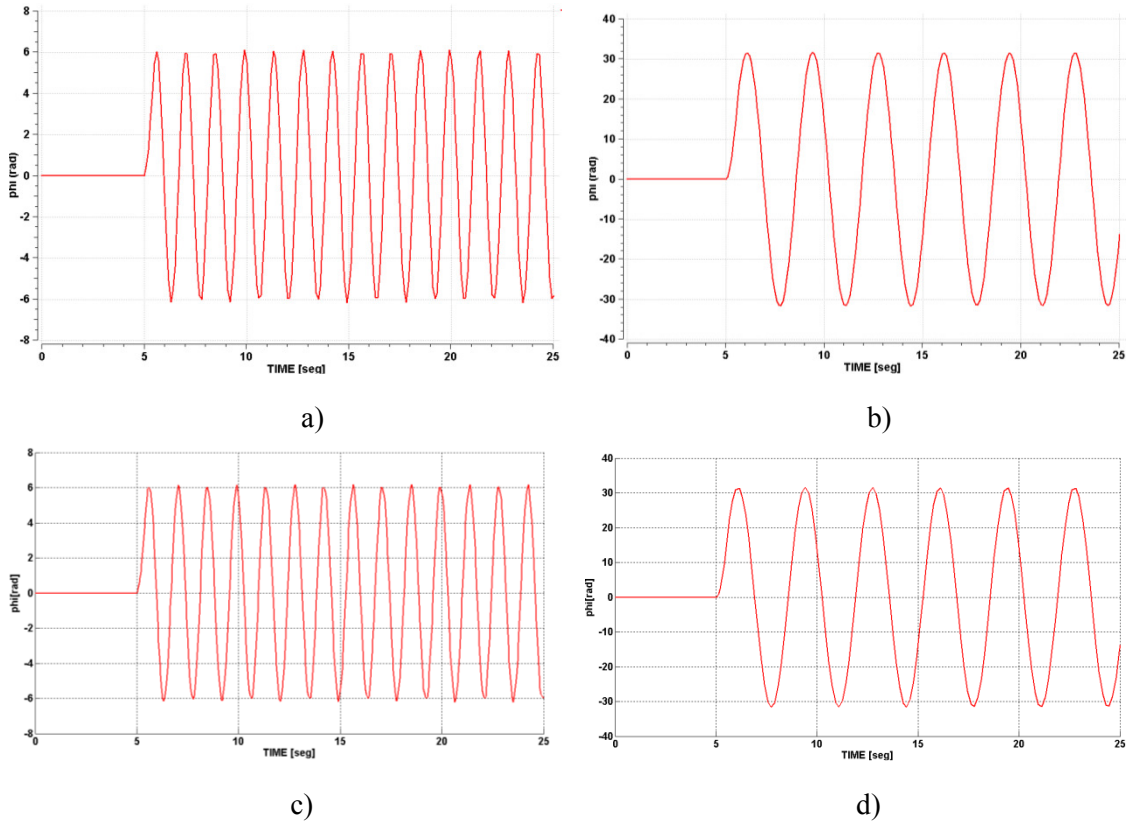


Fig. 5.12: Evolución temporal de las posiciones angulares del elemento *Inertia2* para *d_tor=1* y distintos valores de la constante *k_tor* del elemento *Spring2*: a) *k_tor =1579134 [Nm/rad]* modelo de Ecosimpro; b) *k_tor =15791,34[Nm/rad]* modelo de Ecosimpro; c) *k_tor =1579134 [Nm/rad]* modelo de Simulink; d) *k_tor =15791,34 [Nm/rad]* modelo de Simulink

A la vista de los resultados obtenidos en las figuras anteriores se puede afirmar que, los elementos *R_Spring* y R_*Damper* de la librería *Rotational_Library* son válidos, ya que además de responder como se esperaba frente a variaciones de sus constantes características *(k_tor y d_tor)*, los resultados que se obtienen tras su simulación son prácticamente idénticos a los obtenidos con el modelo de Simulink.

5.2. Simulación de cojinetes hidrodinámicos

Como se explicó en la sección 4.9 del capítulo anterior, un cojinete es un dispositivo mecánico que permite el movimiento relativo entre piezas en contacto, minimizando la fricción y el desgaste entre ellas. Estas ventajas implican un coste mecánico que se traduce en pérdidas energéticas, debido a efectos de cortadura en el seno del lubricante, que el componente *R_Hydro_Bearing* es capaz de determinar.

Si se supone un cojinete hidrodinámico con las características que se representan en la tabla 5.3.

Tabla. 5.3: Características cojinete hidrodinámico.

Dato	Valor	Unidades
d (diámetro del muñón)	25	[mm]
l/d (relación de aspecto)	1	[-]
W (carga soportada)	1,25	[kN]
n (velocidad de giro del muñón)	1200	[rpm]
	125,66	[rad/s]
c (holgura radial)	0,02	[mm]
v (viscosidad media del lubricante)	50	[mPa·s]

Para calcular la pérdida de potencia en el cojinete de la tabla 5.3 de forma analítica hay que seguir los siguientes pasos:

1. Calcular la carga por unidad de área proyectada según la ecuación (4.55):

$$P = \frac{W}{l \cdot d} = \frac{1{,}25 \cdot 10^3}{(25 \cdot 10^{-3})^2} = 2 \cdot 10^6 \left[\frac{N}{m^2}\right] \tag{5.3}$$

2. Obtener el número de *Sommerfeld* según la ecuación (4.54):

$$S = \left(\frac{r}{c}\right)^2 \frac{\eta N}{P} = \left(\frac{12{,}5}{0{,}02}\right)^2 \frac{50 \cdot 10^{-3} \cdot \frac{1200}{60}}{2 \cdot 10^6} = 0{,}195\ [-] \tag{5.4}$$

3. Con el número de *Sommerfeld* obtenido en (5.4) y la relación *l/d* dada, entrar en el gráfico de la figura 4.51 para obtener la variable de fricción del cojinete (figura 5.13).

$$\frac{r}{c} \cdot f = 4{,}5\ [-] \tag{5.5}$$

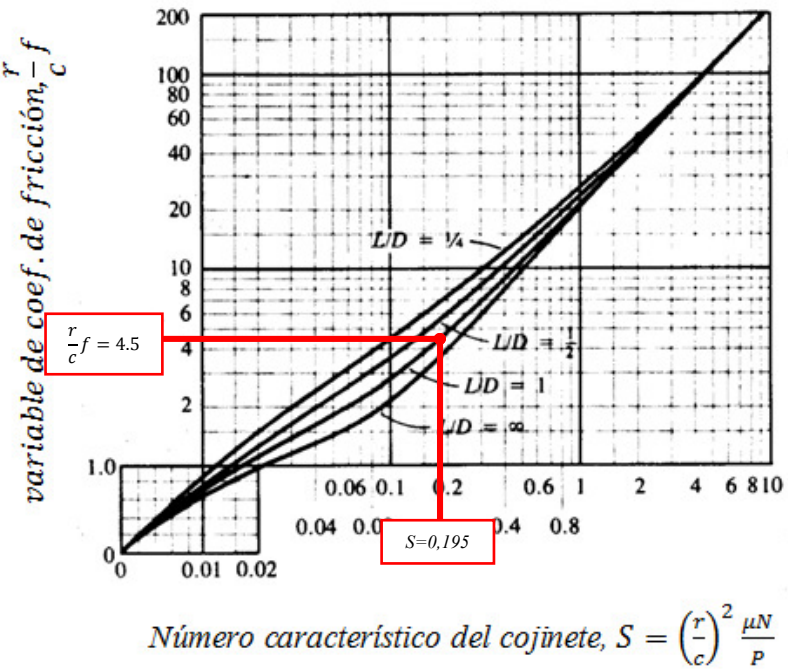


Figura: 5.13: Variable de fricción para el cojinete del ejemplo.

4. Obtener, a partir del resultado obtenido en la ecuación (5.5) el coeficiente de fricción:

$$f = \frac{c}{r} \cdot 4{,}5 = 0{,}0072 \ [-] \tag{5.6}$$

5. Por último, calcular la pérdida de potencia en el cojinete según la ecuación (4.57):

$$H = f \cdot W \cdot \left(n \cdot \frac{2\pi}{60}\right) \cdot r = 0{,}0072 \cdot 1{,}25 \cdot 10^3 \cdot \left(\frac{1200 \cdot 2\pi}{60}\right) \cdot \left(\frac{25 \cdot 10^{-3}}{2}\right) = 14{,}13 \ [W] \tag{5.7}$$

Para calcular la pérdida de potencia en un cojinete como el de la tabla 5.3 mediante el componente *R_Hydro_Bearing*, basta con simular el sistema de la figura 5.14.

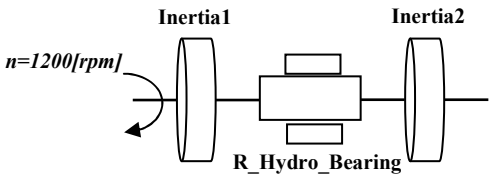


Figura: 5.14: Croquis sistema para simular cojinete hidrodinámico.

El sistema dinámico de la figura 5.14, que bien podría representar un eje que gira apoyado en un cojinete, consta de dos inercias (*Inertia1* e *Inertia2*) que giran a una velocidad angular de *1200 [rpm]* unidas mediante un cojinete cilíndrico con lubricación hidrodinámica (*R_Hydro_Bearing*).

El modelo en EcosimPro que representa el sistema de la figura 5.14 se representa en la figura 5.15.

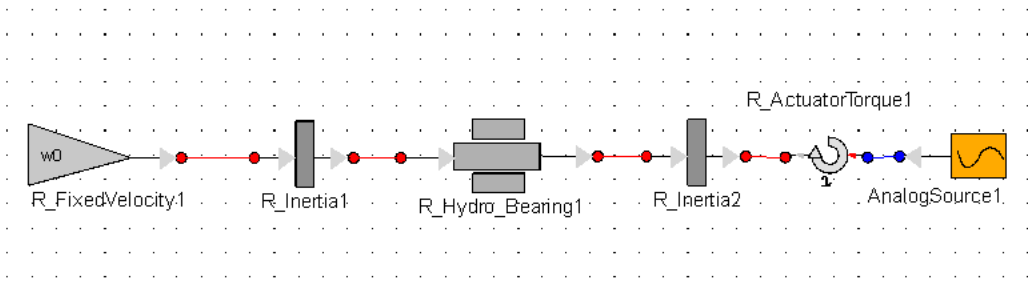


Figura: 5.15: Modelo en EcosimPro del sistema que simula un cojinete hidrodinámico.

El elemento *R_FixedVelocity* impone una velocidad constante al sistema, en este caso se quiere que el cojinete gire a *1200 [rpm]*, es decir *125,66 [rad/s]*. La inercia de los elementos *R_Inertia1* y *R_Inertia2* no es relevante en este caso, por defecto se asigna un valor unitario. El cojinete se instancia según los datos de la tabla 5.3, y siguiendo los criterios establecidos en la tabla 4.64 para las unidades de las constantes introducidas por el componente *R_Hydro_Bearing*. Por último, para instanciar la fuente *AnalogSource1* basta con asignar un valor nulo a su amplitud, ya que en este caso no es necesario que esté operativa.

La tabla 5.4 resume las instancias realizadas para los elementos que simulan el cojinete hidrodinámico.

Tabla: 5.4: Instancias para los elementos del sistema que simula un cojinete hidrodinámico.

Elemento	Parámetro o dato	Tipo	Valor
Inertia1/Inertia2	*I*	REAL	1
	phi_0	REAL	0
R_Hydro_Bearing	*c*	REAL	0,02
	v	REAL	0,05
	l	REAL	25
	d	REAL	25
	W	REAL	1250
R_FixedVelocity1	*w0*	REAL	125,66
AnalogSource1	*Amp*	REAL	0

Una vez realizadas todas las instancias se compila el modelo, se crean la partición por defecto y el experimento (Figura 5.16). En este caso, no es necesario modificar el código que EcosimPro genera de forma automática para el experimento.

```
EXPERIMENT exp1 ON R_Hydro_Bearing.default
   DECLS
   INIT
      -- initial values for state variables
      R_Inertia1.phi = 0

   BOUNDS
   BODY
      -- report results in file reportAll.rpt
      REPORT_TABLE("reportAll.rpt", "*")
      -- for example integrate the model 15 seconds (obtain results every 0.1 seconds)
      TIME = 0
      TSTOP = 15
      CINT = 0.1
      INTEG()
END EXPERIMENT
```

Figura: 5.16: Código del experimento que simula el cojinete hidrodinámico.

En la figura 5.17 se muestra la velocidad de giro del cojinete hidrodinámico durante el experimento.

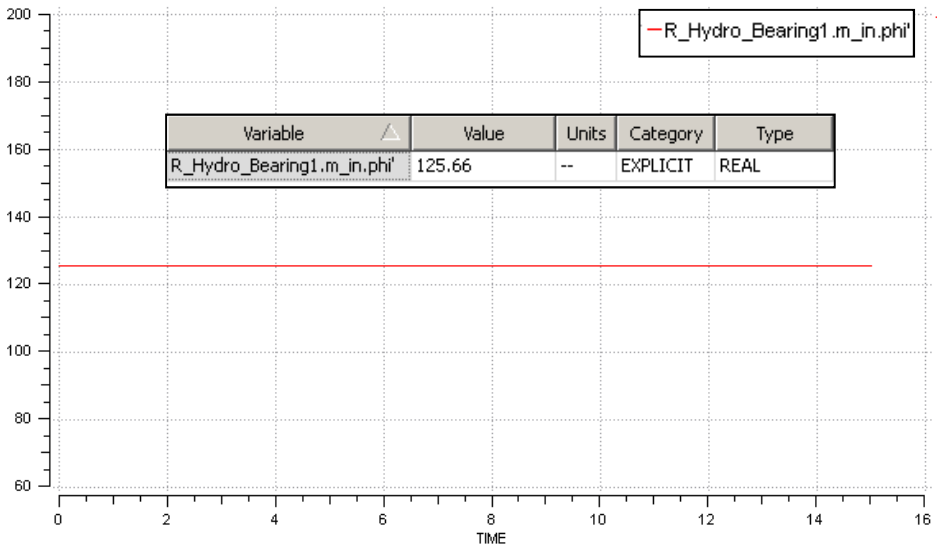


Figura: 5.17: Velocidad de giro del cojinete hidrodinámico en [rad/s].

La pérdida de potencia *H* que se produce en el cojinete simulado con EcosimPro, se representa en la figura 5.18. Dado que la pérdida de potencia en el cojinete obtenida en la simulación del componente *R_Hydro_Bearing* (*H=14.19 [W]*) es muy similar a la obtenida de forma analítica según la ecuación (5.7) (*H=14.13 [W]*), se puede afirmar que el modelo de cojinete hidrodinámico implementado en la librería *Rotational library* es válido.

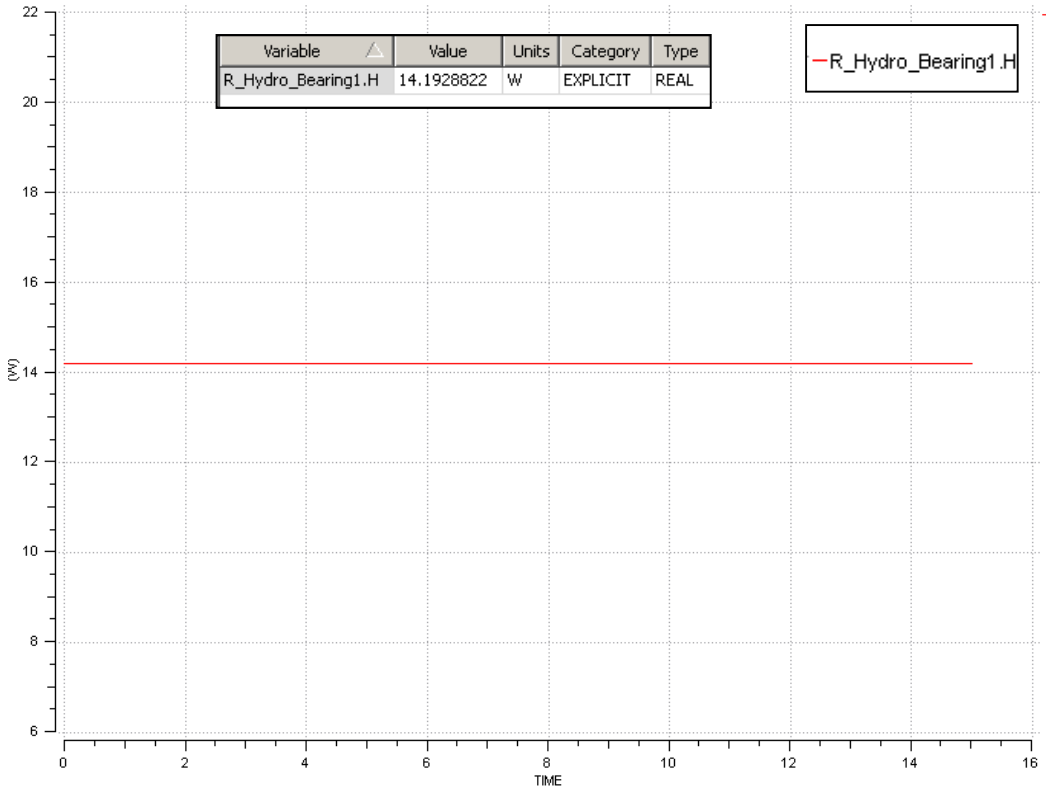


Figura: 5.18: Pérdida de potencia en el cojinete hidrodinámico en [W].

5.3. Simulación de sistemas de frenado

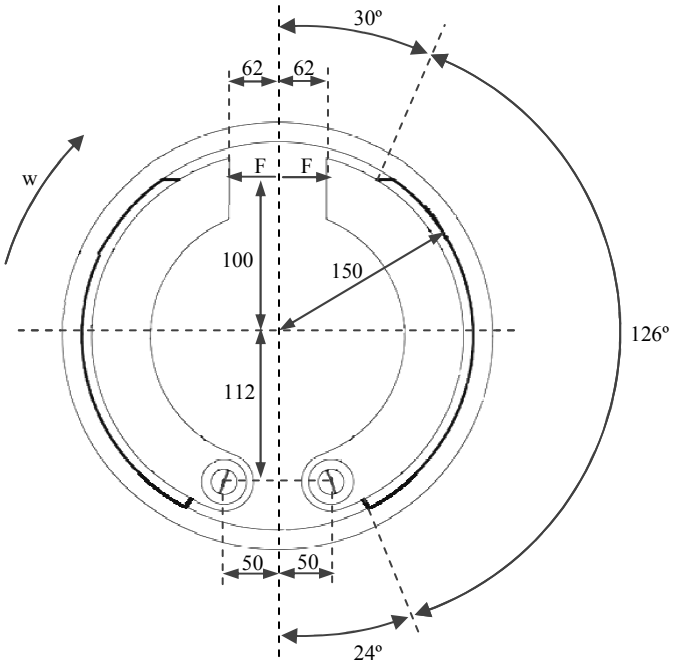
En el presente apartado se modelarán varios sistemas de frenado haciendo uso del componente *R_Drum_and_disc_Brake_jagm.* Los resultados obtenidos serán comparados con los obtenidos de forma analítica con el fin de verificar el funcionamiento de dicho componente.

Pero, además de verificar el funcionamiento del componente, se analizarán en detalle todos los parámetros que lo definen y cómo influyen éstos en el comportamiento de los sistemas simulados.

Por último, se realizará una comparación entre los dos sistemas de frenado que se pueden simular con *R_Drum_and_disc_Brake_jagm*.

5.3.1. Simulación de frenos de tambor

La figura 5.19 representa un freno de tambor de *300 [mm]* de diámetro, que es accionado por un mecanismo que aplica una fuerza F sobre cada zapata. Éstas son idénticas y tienen un ancho de cara de *32 [mm]*. El material de fricción presenta un coeficiente de rozamiento de 0,32 y un límite de presión de *1000 [kPa]*



Para calcular de forma analítica el par de frenada generado por el freno de la figura 5.19, primero es necesario determinar, mediante un equilibrio de momentos en la zapata primaria, la fuerza de accionamiento *F*, así:

(5.8)

donde:

} (5.9)

(5.10)

Con los valores de los parámetros de la tabla 5.5, que se obtienen según las figuras 5.19 y 5.20 , y operando las ecuaciones (5.8), (5.9) y (5.10) se obtiene un valor para la fuerza de accionamiento de:

(5.11)

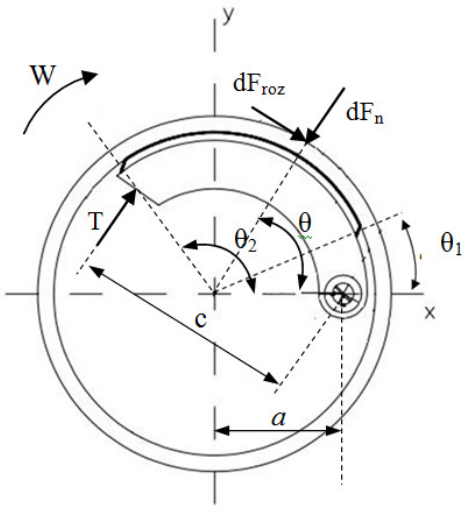


Figura: 5.20: Croquis parámetros de diseño para un freno de tambor.

Tabla: 5.5: Parámetros de cálculo freno de tambor figura 5.18.

Parámetro	Valor	Unidades
μ	0,32	[-]
p_a	1000	[kPa]
b	0,032	[m]
r	0,15	[m]
θ_a	90	[º]
θ_1	0	[º]
θ_2	126	[º]
c	0,212	[m]

Con el valor de la fuerza de accionamiento F de la ecuación (5.11) es posible calcular el valor del par de frenada en la zapata primaria:

$$N = r \cdot \mu \cdot F_n \tag{5.12}$$

y como:

$$F_n = \frac{p_a \cdot b \cdot r}{sen\theta_a} \cdot [\cos\theta_1 - \cos\theta_2] \tag{5.13}$$

entonces el par de frenada en la zapata primaria es:

$$N_{frenada}^{1aria} = \frac{p_a \cdot b \cdot \mu \cdot r^2}{sen\theta_a} \cdot [\cos\theta_1 - \cos\theta_2] = 365{,}82\ [Nm] \tag{5.14}$$

Como la fuerza de accionamiento es ahora conocida y común en ambas zapatas, es posible obtener el valor de la presión en la zapata secundaria realizando un equilibrio de momentos en dicha zapata, así:

$$F \cdot c = N_{rozamiento}^{2^{aria}} + N_{normales}^{2^{aria}} \tag{5.15}$$

donde:

$$N_{rozamiento}^{2^{aria}} = \frac{\mu \cdot p \cdot b \cdot r}{sen\theta_a} \cdot \{(-r \cdot \cos\theta)\,{}_{\theta_1}^{\theta_2} - a \cdot \left(\frac{1}{2} \cdot sen^2\theta\right)_{\theta_1}^{\theta_2}\} \tag{5.16}$$

$$N_{normales}^{2^{aria}} = \frac{p \cdot b \cdot r \cdot a}{sen\theta_a} \cdot \left(\frac{\theta}{2} - \frac{1}{4} \cdot sen(2\theta)\right)_{\theta_1}^{\theta_2} \tag{5.17}$$

Operando las ecuaciones (5.15), (5.16) y (5.17) y despejando la presión p, se obtiene un valor para esta presión de:

$$p = 442\ [kPa] \tag{5.18}$$

Entonces el par de frenada en la zapata secundaria es:

$$N_{frenada}^{2^{aria}} = \frac{p \cdot b \cdot \mu \cdot r^2}{sen\theta_a} \cdot [\cos\theta_1 - \cos\theta_2] = 162\ [Nm] \tag{5.19}$$

Finalmente el par de frenada total será:

$$N_{frenada}^{total} = N_{frenada}^{1^{aria}} + N_{frenada}^{2^{aria}} = 365{,}82 + 162 = 527{,}82[Nm] \tag{5.20}$$

Hasta ahora se ha calculado el par de frenada para el freno de tambor de la figura 5.19 de forma analítica. Para obtener dicho par de frenada mediante el componente *R_Drum_and_disc_Brake_jagm* basta con simular el sistema de la figura 5.21.

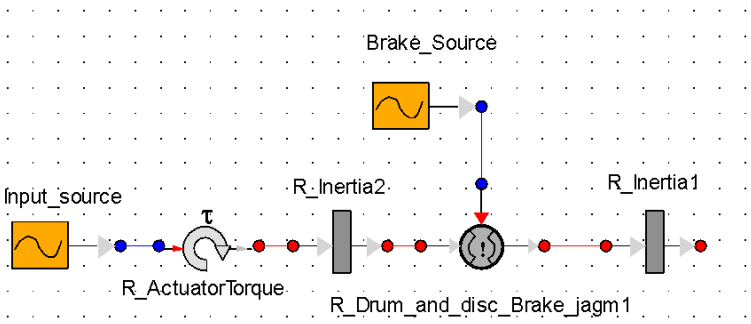


Figura: 5.21: Sistema que simula un freno de tambor.

El sistema de la figura 5.21 representa un freno que actúa entre dos inercias, *R_Inertia2* y *R_Inertia1*. El sistema es excitado por un actuador de par que, en virtud de la fuente analógica *Input_Source,* genera un pulso con una amplitud de *500 [N·m]* desde el inicio del experimento, y hasta el instante *t=4.9 [s]* (Figura 5.22).

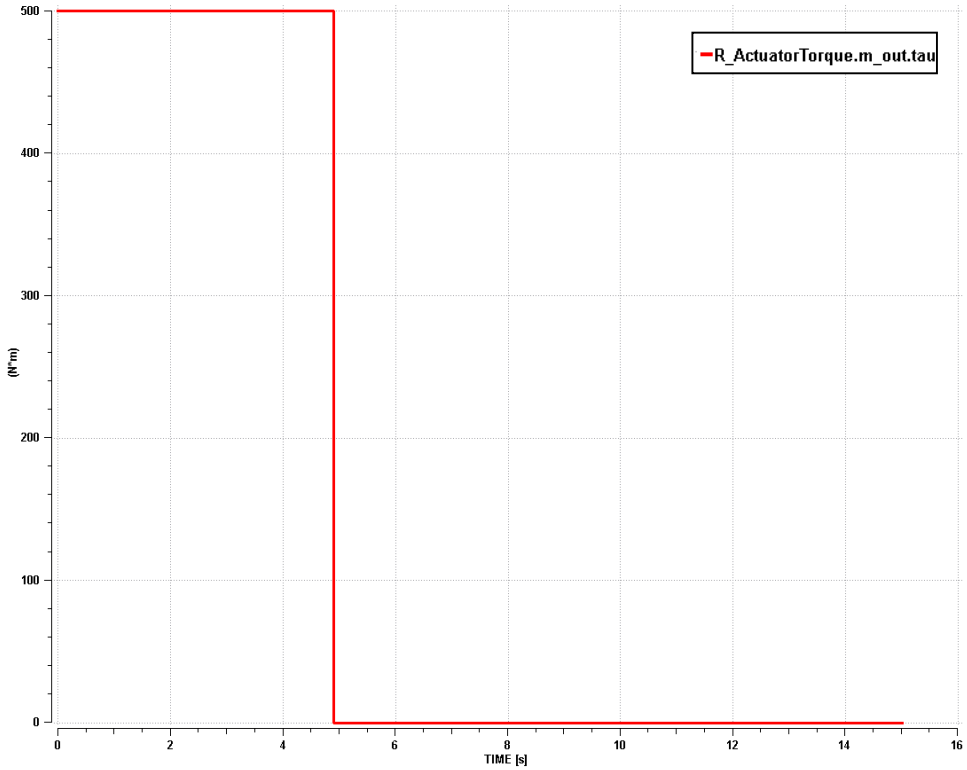


Figura: 5.22: Par generado por *R_ActuatorTorque*.

La instanciación de la fuente analógica *Input_Source* se muestra en la figura 5.23.

Name	Type	Value	Units	Description
		PARAMETERS		
n_out	INTEGER	1	-	Dimension of outputs (-)
		DATA		
source	ENUM SOURCES.SourceOption	Source_Pulse		Waveform
Amp	REAL	500	-	Signal amplitude or height (-)
Tstart	REAL	0	s	Starting time of signal genera...
Offset	REAL	0	-	Offset of output signal (-)
Phase	REAL	0	rad	Phase of sine source (rad)
Period	REAL	100	s	Period of sine, pulse, sawtoot...
pulseWidth	REAL	4.9	s	Pulse width of pulse wave (s)
rampDuration	REAL	10	s	Duration of the ramp (s)
tabmethod	ENUM SOURCES.tableMethod	LinearInterpWithEvents		Method to interpolate or conn...
timeTable	TABLE 1D	Edit	-	Table for table source (-)

Figura: 5.23: Instanciación de la fuente analógica *Input_Source*.

En el sistema de frenado de un vehículo convencional, el recorrido del pedal de freno, caracteriza la presión del sistema hidráulico que acciona dicho sistema. De esta forma, en una frenada, cuanto más se pisa el freno, mayor presión tendrá el circuito hidráulico, y, por tanto, mayor será el par de frenada desarrollado.

En el caso del freno simulado en la figura 5.21, la señal de salida de la fuente analógica *Brake_Source*, cuyo valor puede variar entre cero y uno, equivale al pedal de freno de un sistema de frenos de un vehículo convencional. Así, si dicha señal alcanza un valor unitario, el par de frenada desarrollado por el componente *R_Drum_and_disc_Brake* será el máximo posible, y por el contrario, si dicha señal es igual a cero, el par de frenada también será igual a cero.

Como lo que se pretende en este caso es calcular el par máximo de frenada, se ha instanciado la fuente *Brake_Source* de manera que el valor de su señal de salida (figura 5.24) sea unitario.

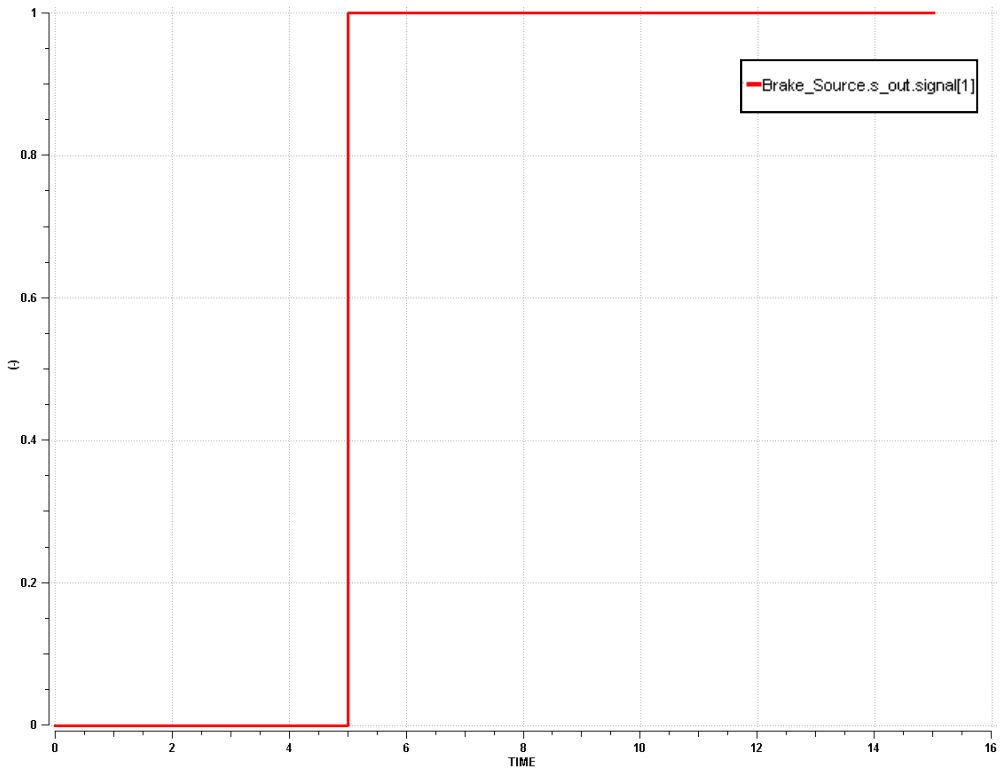


Figura: 5.24: Señal de salida de la fuente *Brake_Source*.

Name	Type	Value	Units	Description
		PARAMETERS		
n_out	INTEGER	1	-	Dimension of outputs (-)
		DATA		
source	ENUM SOURCES....	Source_Step		Waveform
Amp	REAL	1	-	Signal amplitude or height (-)
Tstart	REAL	5	s	Starting time of signal generation (s)
Offset	REAL	0	-	Offset of output signal (-)
Phase	REAL	0	rad	Phase of sine source (rad)
Period	REAL	100	s	Period of sine, pulse, sawtooth, or square source (s)
pulseWidth	REAL	1	s	Pulse width of pulse wave (s)
rampDuration	REAL	10	s	Duration of the ramp (s)
tabmethod	ENUM SOURCES....	nterpWithEvents		Method to interpolate or connect the table points
timeTable	TABLE 1D	Edit	-	Table for table source (-)

Figura: 5.25: Instanciación de la fuente analógica *Brake_Source*.

Las instancias de las inercias se muestran en la tabla 5.6.

Tabla: 5.6: Instanciación los componentes *R_Inertia1* y *R_Inertia2*.

Elemento	Parámetro o dato	Tipo	Valor
Inertia1/Inertia2	*I*	REAL	1
	phi_0	REAL	0

Por último, hay que instanciar el componente *R_Drum_and_disc_Brake_jagm* (Figura 5.26) según las características especificadas para el freno de tambor de la figura 5.19.

Name	Type	Value	Units	Description
		DATA		
type	ENUM PRUEBA_A...	primary_secondary_shoes		Brake type
mue_pos	TABLE 1D	Edit		[w,mue] positive sliding friction coefficient: w_rel>=0
peak	REAL	1.1		Maximum value of mue for w_rel=0
Ri	REAL	0.025	m	Disc brake internal radius
Re	REAL	0.15	m	Disc brake external radius or radius shoe(m)
p_max	REAL	1e6	N/m^2	Maximum pressure on the lining
angle	REAL	126	º	Angle covered by the brake pad or by the shoe
angle0	REAL	0	º	Start angle of the shoe
angle_Pmax	REAL	90	º	Maximun pressure angle in the shoe
w_i	REAL	1	rad/s	Initial relative angular velocity
n_pad	REAL	1	-	Number of brake pads
c	REAL	0.212	m	Distance between the force application point and th...
b	REAL	0.032	m	Shoe width
a	REAL	0.123	m	Distance between de drum centre and the joint

Figura: 5.26: Instanciación del componente *R_Drum_and_disc_Brake*.

Para instanciar el componente según el freno de tambor de la figura 5.19, es necesario especificar que se trata de un freno de tambor con una zapata primaria y otra secundaria, para ello se selecciona *primary_secondary_shoes* en la variable *type.* También se completan los campos correspondientes a las constantes geométricas (*Re, angle, angle0*, *angle_Pmax, c, b* y *a*) así como el correspondiente a la presión máxima admisible en el material de fricción (*p_max*). Como el valor de *mue_pos* se toma de una tabla, su instanciación es distinta a la del resto de constantes. Para instanciarla hay desplegar el editor de la tabla de *mue_pos* (Figura 5.27), y rellenar los campos correspondientes. En este caso el coeficiente de fricción no varía con la velocidad, por ello la recta que representa la tabla *mue_pos* tiene pendiente nula.

Es importante señalar que, como se observa en la figura 5.26, en el editor de atributos del componente aparecen parámetros que no son relevantes para el tipo de freno seleccionado vía *type,* como por ejemplo *Ri* ó *n_pad*, aún así, es necesario asignarles algún valor, de lo contrario el compilador de EcosimPro mostraría un mensaje de error al compilar el modelo.

Una vez realizadas todas las instancias se compila el modelo se crea la partición por defecto y se crea el experimento.

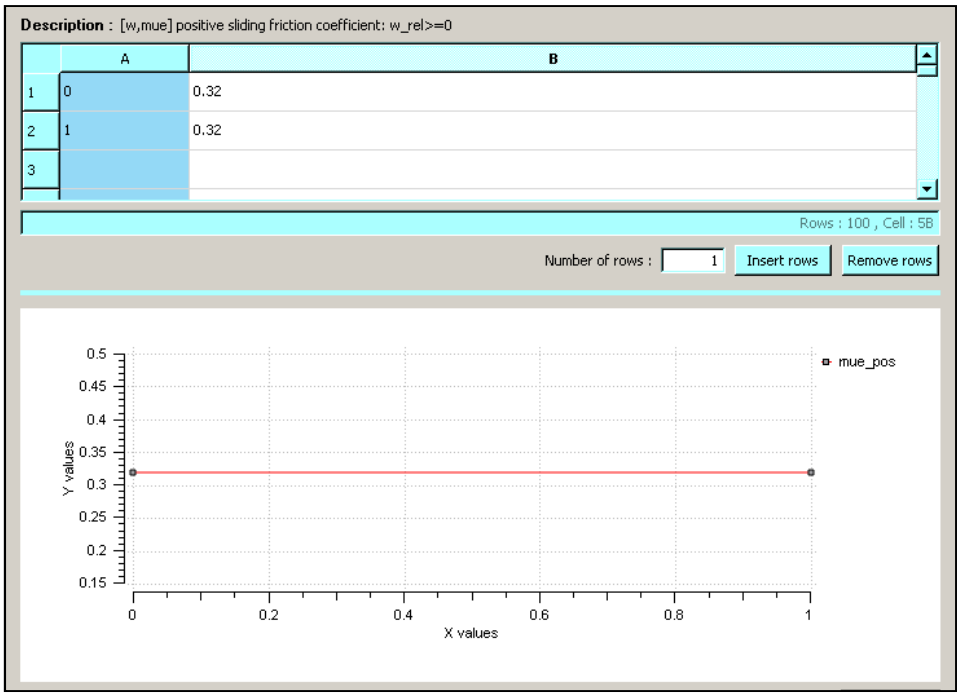


Figura: 5.27: Instanciación de la tabla 1D *mue_pos*.

El valor de la variable *tau_friction* representa el par de frenada del componente *R_Drum_and_disc_Brake_jagm*. La evolución de esta variable durante del experimento se representa en la figura 5.28.

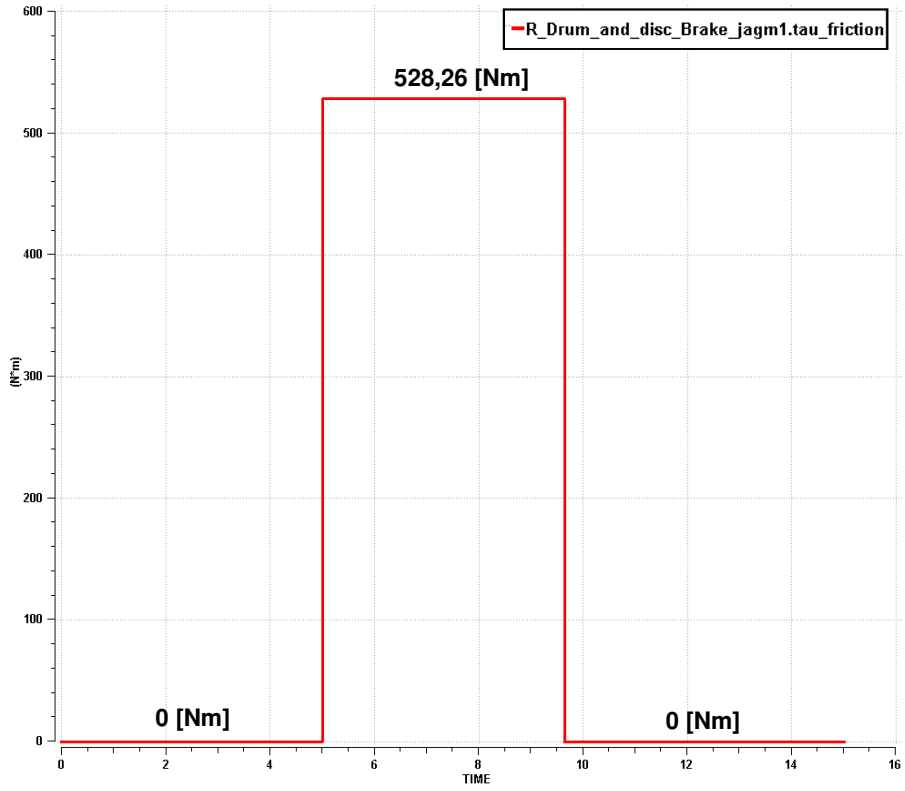


Figura: 5.28: Evolución de la variable *tau_friction*.

En la figura 5.28 se observa que el valor de *tau_friction* es nulo hasta el instante *t=5 [s]*, momento en el que la señal de salida de la fuente *Brake_Source* alcanza un valor unitario. Desde este momento y hasta el instante *t=9,64 [s]* el valor de *tau_friction* permanece constante e igual a *528,26 [N·m]*. A partir del instante *t=9,64 [s]* y hasta el final del experimento el valor del par de frenada vuelve a ser nulo.

El gráfico representado en la figura 5.29 ayuda a entender los saltos en el par de frenada. En el instante *t=5 [s]* cuando la señal de salida de la fuente *Brake_Source* es unitaria, el modo de deslizamiento del freno (*imode*) pasa de *3* a *2*, con lo que comienza a actuar el freno, y la velocidad relativa entre el freno y los elementos giratorios conectados a él (*w_relfric*), que alcanza su máximo valor en este instante, comienza a disminuir. Cuando el freno detiene el sistema, *w_relfric* es nula, y el modo de deslizamiento del freno pasa de *2* a *0*, en este punto el freno sigue actuando, pero como no existe movimiento, el valor del par de frenada es nulo.

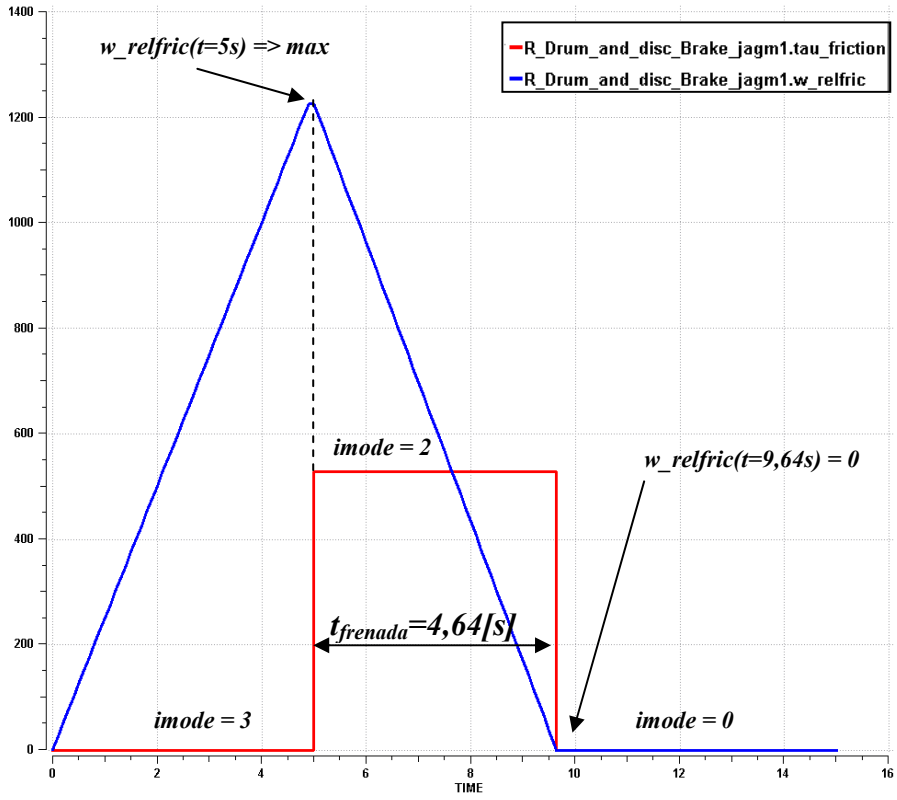


Figura: 5.29: Evolución de la variable *tau_friction* y *w_relfric* para freno tipo *primary_secondary_shoes*.

El valor obtenido para el par de frenada mediante el modelo simulado de la figura 5.21,que resultó ser *528,26 [N·m]*, es muy similar al obtenido de forma analítica según la ecuación (5.18) (*527,26 [N·m]*), resultado que verifica el correcto funcionamiento del componente *R_Drum_and_disc_Brake_jagm*.

Por otro lado, en la figura 5.29, se observa que el tiempo empleado por el freno, que es de tipo *primary_secondary_shoes*, en detener el sistema es de *4,64 [s]*. Si ahora se cambia la disposición de las zapatas, pasando a un freno de tambor de tipo

twin_shoes_for, es decir, que las dos zapatas sean primarias mientras *w_relfric* sea mayor que cero, se consigue reducir el tiempo de frenada en un 28% (Figura 5.30).

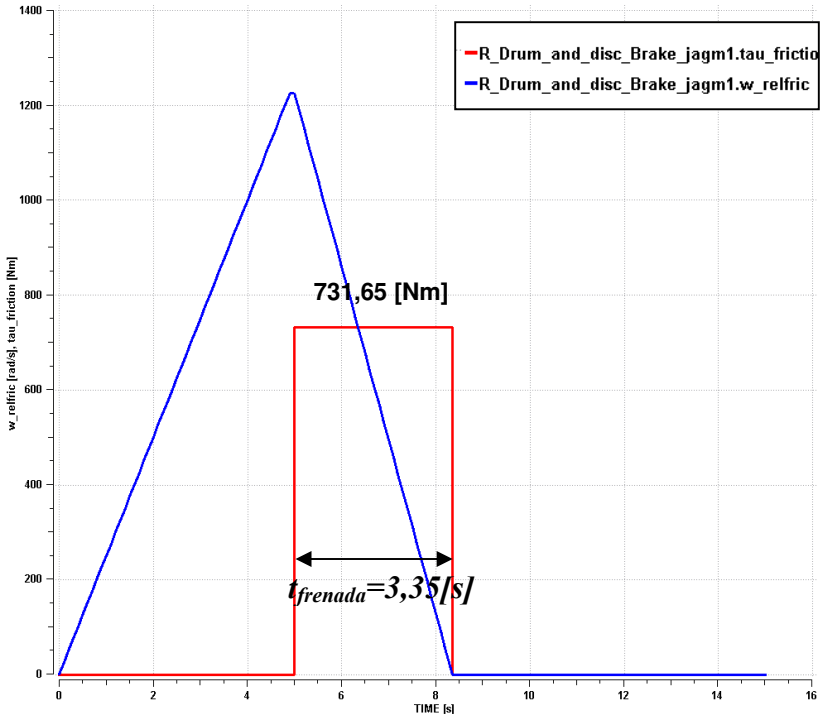
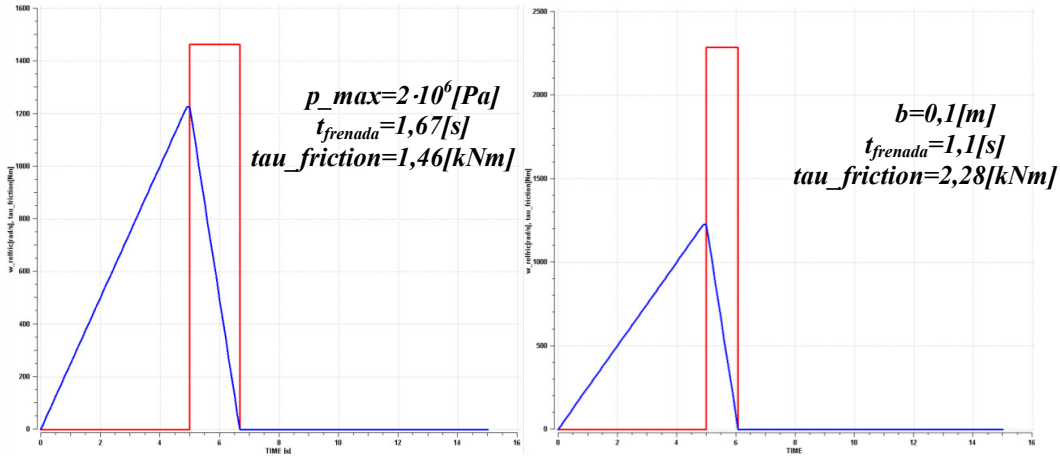


Figura: 5.30: Evolución de la variable *tau_friction* y *w_relfric* para freno tipo *twin_shoes_for*.

Además, se produce un aumento del par de frenada de aproximadamente *204 [N·m]*, lo que supone casi un 39% del par de frenada generado por el freno de tambor de tipo *primary_secondary_shoes*.

Por último, como cabe esperar, cualquier modificación en los parámetros del freno que implique mejoras en sus características mecánicas, como un incremento de la presión admisible en el material de fricción, ó geométricas, como un incremento del ancho de zapata, dará como resultado una reducción del tiempo de frenada (Figura 5.31).



a) b)

Figura: 5.31: Evolución de la variable *tau_friction* y *w_relfric* para freno tipo *twin_shoes_for;* a) *p_max= 2 10^6[Pa];* b) *b=0,1[m]*

5.3.2. Simulación de frenos de disco

Mediante las instancias *new_brake_disc* y *old_brake_disc* de la constante enumerativa *type,* es posible simular frenos de disco como el representado en la figura 5.32. Dicho freno está constituido por un disco en la parte central, que gira solidario con el mecanismo cuya velocidad se pretende disminuir, y dos superficies de fricción, que se encuentran una a cada lado del disco.

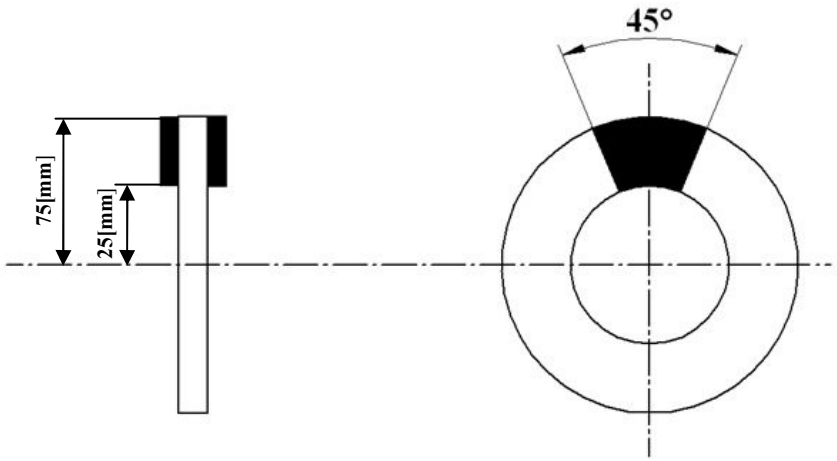


Figura 5.32: Freno de disco.

El par de frenada generado por el freno de disco de la figura anterior, suponiendo que la distribución de presiones en el disco es variable (disco nuevo), se calcula según la ecuación (5.21).

$$N = \mu \cdot (R_e + R_i) \cdot F_n \tag{5.21}$$

donde:

$$F_n = p_a \cdot \alpha \cdot R_i(R_e - R_i) \tag{5.22}$$

Si se supone que el material de fricción soporta una presión máxima de *1.5 [MPa]* y que el coeficiente de fricción es de *0,2[-]*, el par total de frenada resulta ser:

$$N = 29{,}45\ [N \cdot m] \tag{5.23}$$

En la figura 5.33 se representa el par de frenada obtenido al simular el sistema de la figura 5.21, pero instanciando el componente *R_Drum_and_disc_Brake_jagm* de manera que represente el freno de la figura 5.32.

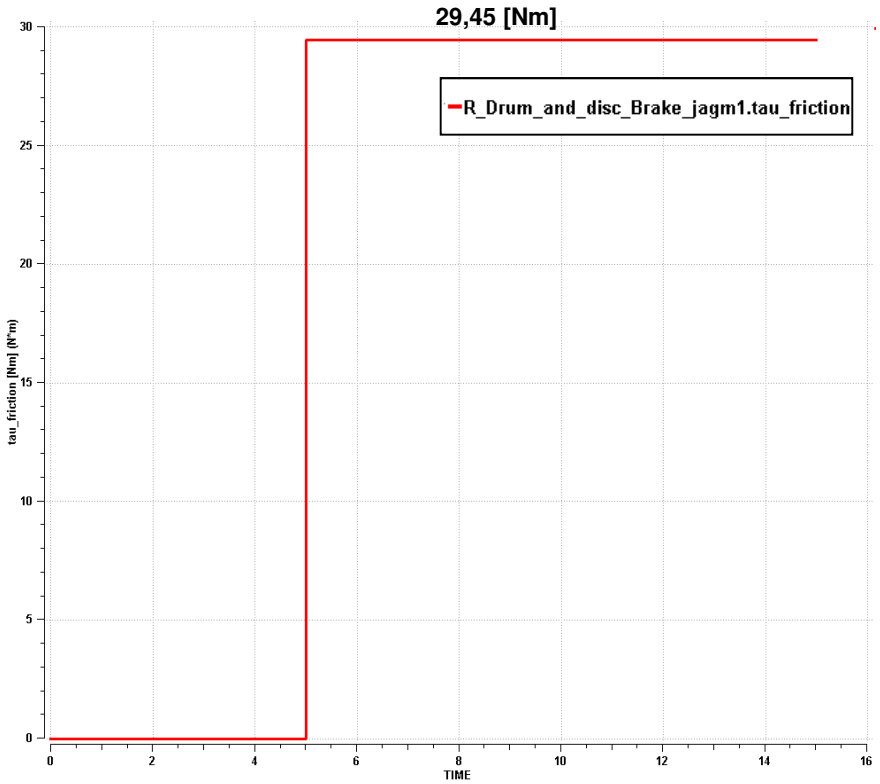


Figura 5.33: Par de frenada generado por el freno de disco de la figura 5.31.

Las instancias realizadas sobre el componente *R_Drum_and_disc_Brake_jagm* para simular el freno de disco de la figura 5.32 se muestran en la figura 5.34:

Name	Type	Value	Units	Description
		DATA		
type	ENUM PRUEBA_A...	used_brake_disc		Brake type
mue_pos	TABLE 1D	Edit		[w,mue] positive sliding friction coefficient: w_rel>=0
peak	REAL	1.1		Maximum value of mue for w_rel=0
Ri	REAL	0.025	m	Disc brake internal radius
Re	REAL	0.075	m	Disc brake external radius or radius shoe(m)
p_max	REAL	1.5e6	N/m^2	Maximum pressure on the lining
angle	REAL	45	°	Angle covered by the brake pad or by the shoe
angle0	REAL	0	°	Start angle of the shoe
angle_Pmax	REAL	90	°	Maximun pressure angle in the shoe
w_i	REAL	1	rad/s	Initial relative angular velocity
n_pad	REAL	2	-	Number of brake pads
c	REAL	0.212	m	Distance between the force application point and the ...
b	REAL	0.1	m	Shoe width
a	REAL	0.123	m	Distance between de drum centre and the joint

Figura 5.34: Instancias de *R_Drum_and_disc_Brake_jagm* para el freno de la figura 5.32.

El valor del coeficiente de fricción queda definido mediante la instanciación de la tabla *mue_pos* según la figura 5.35.

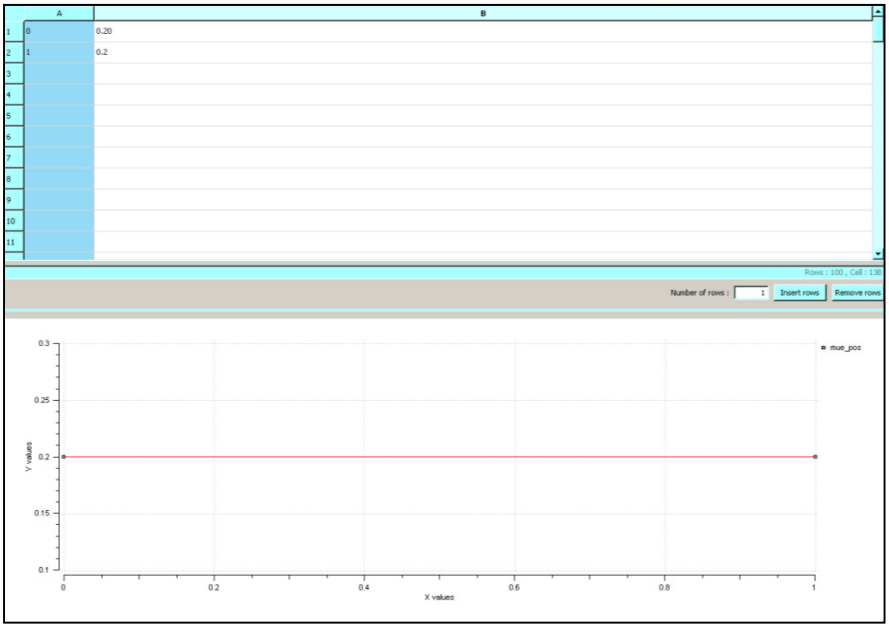


Figura 5.35: Instancias de la tabla *mue_pos* para definir el coeficiente de fricción.

Debido al reducido par de frenada que ofrece el freno simulado, la duración del experimento es insuficiente para detener el sistema (Figura 5.36), al contrario de lo que ocurría en el apartado anterior.

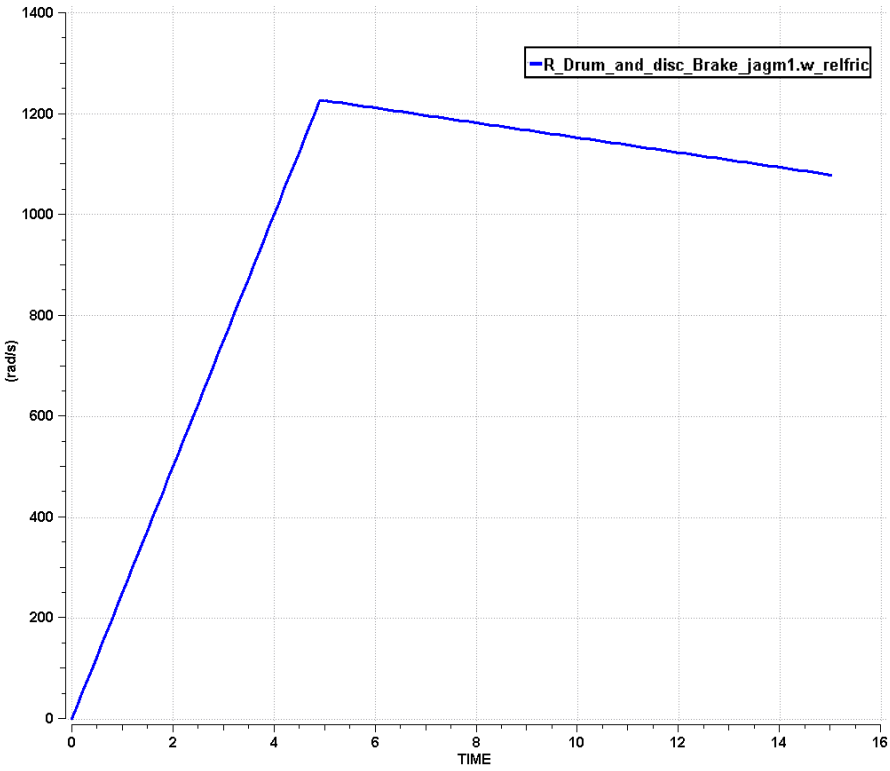


Figura 5.36: Evolución temporal de la velocidad del sistama.

Por otro lado, dado que el par de frenada obtenido al simular el sistema de la figura 5.21, que resultó ser de *29.45 [N·m]*, es idéntico al obtenido de forma analítica según la ecuación (5.21), queda confirmado el correcto funcionamiento de los diferentes componentes implicados en dicha simulación.

5.3.3. Frenos de tambor vs frenos disco

En el presente apartado se simulará el sistema de la figura 5.37, realizando distintas instanciaciones del componente *R_Drum_and_disc_Brake_jagm* con el fin de determinar cuál es el tipo de freno que ofrece un mejor par de frenada.

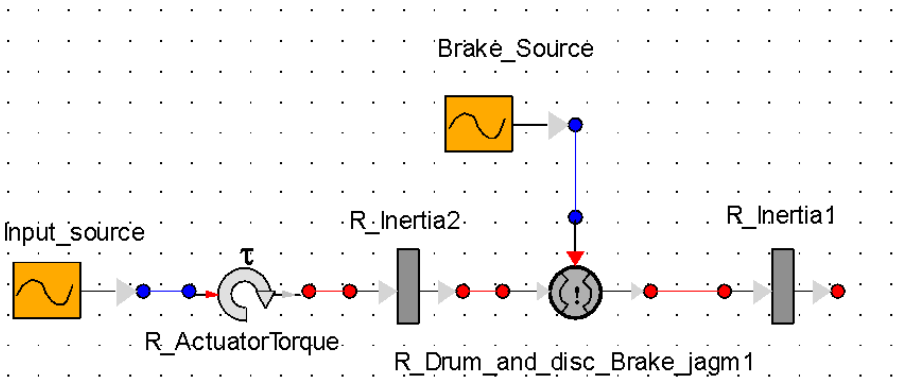


Figura 5.37: Modelo de sistema de frenado.

Pero, para poder extraer resultados comparables, es necesario que ciertos parámetros de los frenos simulados sean equivalentes. Estos parámetros son tres: la presión en el material de fricción, la superficie total de fricción[14] y el coeficiente de rozamiento.

Tabla 5.7: Instancias comunes del componente *R_Drum_and_disc_Brake_jagm*.

Parámetro	Valor	Unidades
Presión en material fricción	10^6	[Pa]
Superficie total de fricción	0,01	[m^2]
Coeficiente de rozamiento	0,2	[-]

El resto de los elementos del sistema de la figura 5.37 se mantienen según las instancias realizadas para esos componentes en el sistema del apartado 5.3.1.

El la figura 5.38, se muestra la evolución temporal del par de frenada en función de la instancia *type* del componente *R_Drum_and_disc_Brake_jagm*. En ella se puede observar que el freno de tambor con dos zapatas primarias es el que ofrece un mayor par de frenada, aunque su resultado es muy similar al del freno de disco nuevo. Resulta sorprendente el empeoramiento de las prestaciones de los frenos de disco cuando la presión en el material de fricción no es constante (freno usado), siendo este el único caso en el que la duración del experimento es insuficiente para que el freno detenga el sistema.

[14] Se llama *superficie total de fricción* al área de contacto existente entre el material de fricción de las pastillas y disco (en el caso de los frenos de disco) y al existente entre el material de fricción de las zapatas y tambor (en frenos de tambor) , la cual está relacionada con la magnitud del par de frenada.

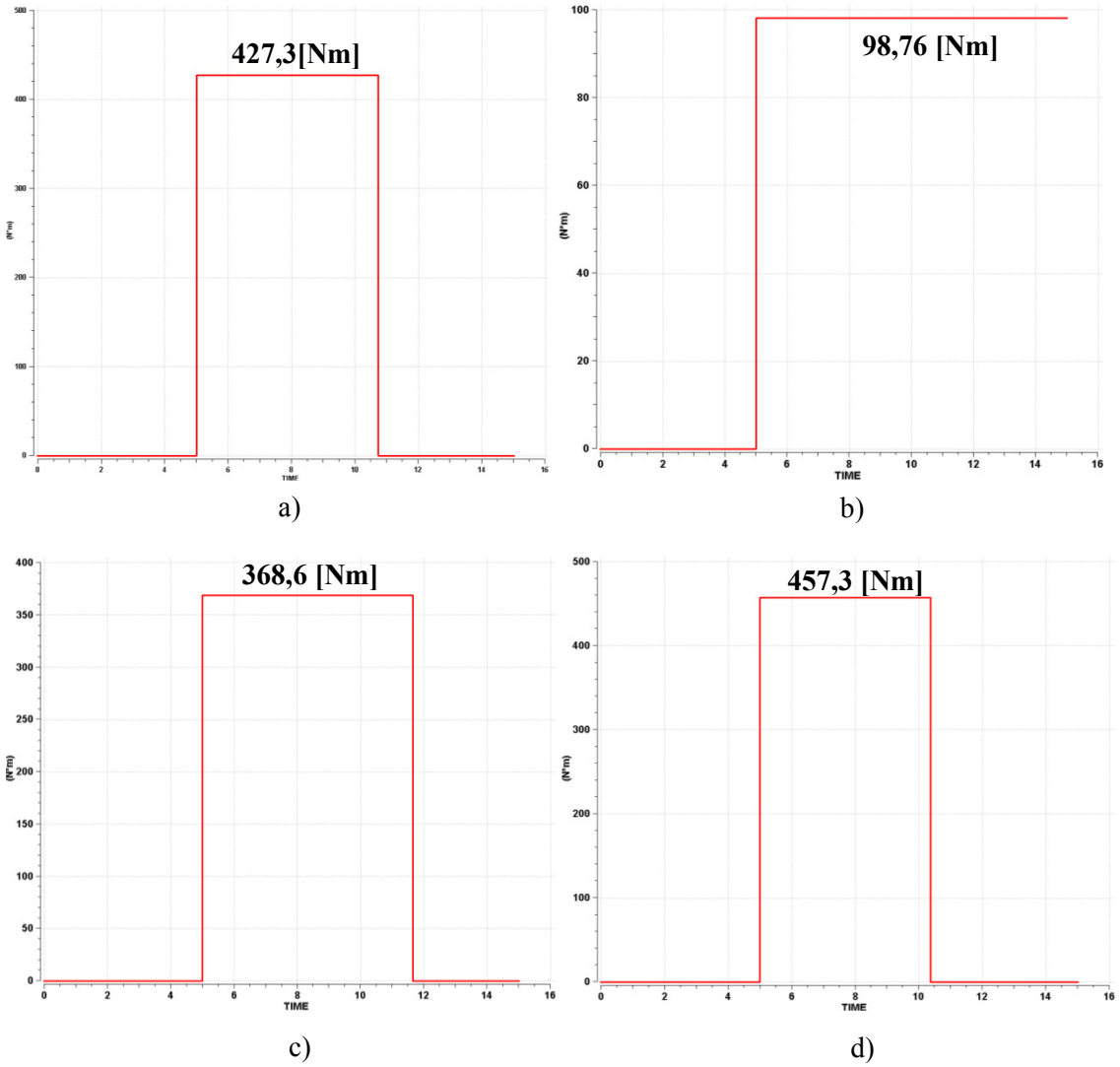


Figura 5.38: Evolución temporal del par de frenada para distintas instancias de la variable *type*: a)*type=new_brake_disc;* b) *type=used_brake_disc* ;c) *type=primary_secondary_shoes* ; d) *type=twin_shoes*

Los resultados de la figura anterior podrían llevar a pensar que los frenos de tambor con zapatas primarias gemelas son más eficaces que los frenos de disco, aunque en realidad esto no es cierto. Los efectos térmicos, no contemplados en estos modelos, tienen una influencia decisiva en la eficiencia de los sistemas de frenado. Por este motivo los frenos de disco, que son capaces de evacuar el calor generado durante la frenada con mayor facilidad, resultan más eficaces que los frenos de tambor. Por tanto, lo que la figura 5.38 confirma, es que los frenos de tambor con zapatas primarias gemelas presentan, en condiciones similares, una capacidad de frenado mayor que la del resto de los sistemas de frenado simulados en la presente sección.

5.4. Simulación de sistemas de transmisión

5.4.1. Verificación del componente *R_Clutch_jagm*

Para verificar de una forma sencilla el embrague modelado mediante el componente *R_Clutch_jagm,* se simulará el sistema representado en la figura 5.39 que recibe el nombre de *Transmisión_simple*, ya que representa un sistema de transmisión muy simplificado. Dicho sistema está compuesto por un embrague que conecta el motor con la transmisión del vehículo. El par motor T_e actúa sobre la inercia del motor J_e, que está directamente conectada al embrague. El embrague, formado por un disco cuyos diámetros exterior e interior son *240[mm]* y *160[mm]* respectivamente, transmite el par generado por el motor a la transmisión representada mediante la inercia J_t.

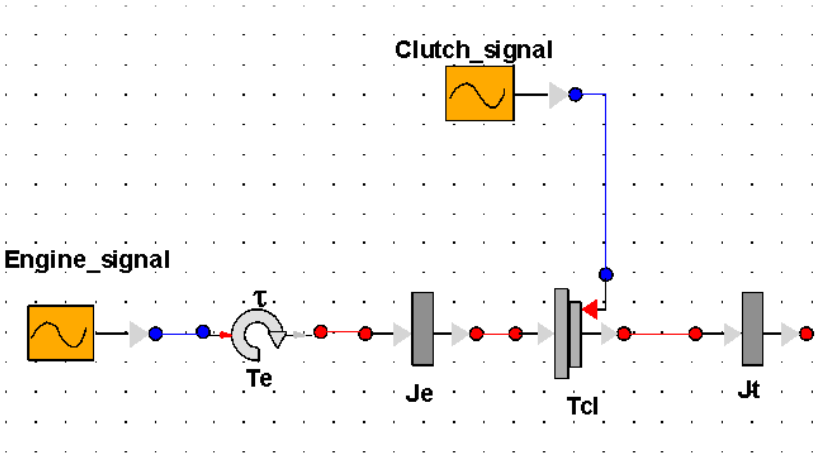


Figura 5.39: Esquema del sistema *Transmisión_simple*.

El motor, cuya inercia es de *0,5 [kg·m²]*, se instancia de forma que desarrolle un par constante de *280[N·m]* durante todo el experimento. El embrague, que se ha supuesto nuevo, es accionado a partir del instante *t=2[s]* por un mecanismo que aplica una fuerza máxima de *8100[N]*. La inercia de la transmisión se ha supuesto igual a *1,7[kg·m²]*. Las instancias del sistema *Transmisión_simple* se resumen en la tabla 5.8.

Tabla 5.8: Resumen instanciación sistema *Transmisión_simple*.

Elemento	Parámetro o dato	Tipo	Valor	Unidades
Engine_signal	*Source*	ENUMERATIVO	*Source_constant*	[-]
	Amp	REAL	*280*	[-]
	Tstart	REAL	*0*	[s]
Clutch_signal	*Source*	ENUMERATIVO	*Source_constant*	[-]
	Amp	REAL	*1*	[-]
	Tstart	REAL	*2*	[s]
J_e	*I*	REAL	*0,5*	[kg·m²]
	phi_0	REAL	*0*	[rad]

J_t	*I*	REAL	*1,7*	[kg·m^2]
	phi_0	REAL	*0*	[rad]
T_{cl}	*phi_rel_0*	REAL	*0*	[rad]
	data	ENUMERATIVO	*force*	[-]
	clutch	ENUMERATIVO	*new*	[-]
	mue_pos	TABLA_1D	0,4 ∀ *w*	[-]
	peak	REAL	*1,1*	[-]
	x	REAL	*8100*	[N]
	n_disc	REAL	*1*	[-]
	w_rel_i	REAL	*0*	[rad/s]
	Ri	REAL	*0,08*	[m]
	Re	REAL	*0,12*	[m]

La figura 5.40 muestra la evolución temporal de las velocidades de los elementos inerciales J_e y J_t. Como se observa en dicha figura, la diferencia entre las velocidades angulares de motor y embrague disminuye desde su valor máximo, que se alcanza justo antes de que el embrague se active en *t=2[s]*, hasta cero en *t=2,98[s]*. El periodo que transcurre desde la activación del embrague hasta que se igualan las velocidades angulares del sistema se denomina *fase de acoplamiento*. A partir de este momento se dice que el embrague está acoplado. Como el par transmitido por el motor es constante, y dado que la aceleración angular de un sistema en rotación es proporcional a dicho par (análogo rotacional de la *2ª Ley de Newton*), ésta también debe ser constante, y por tanto la velocidad angular aumentará con el tiempo (lo que explica la pendiente positiva en la velocidad angular del sistema de la Fig.5.40).

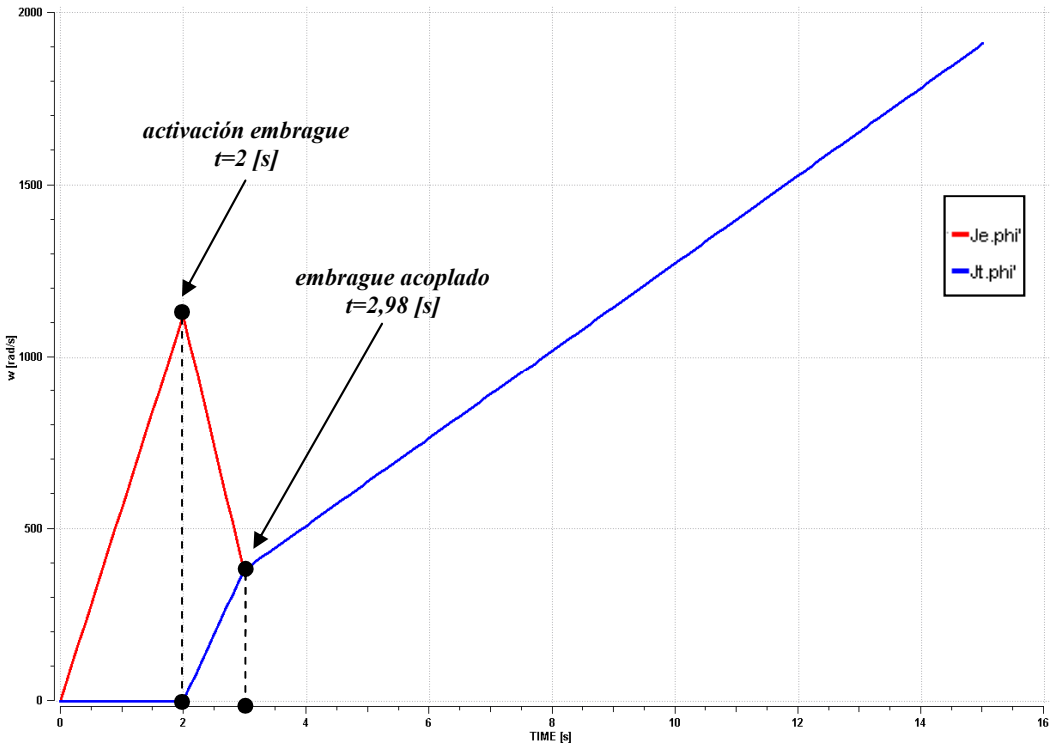


Figura 5.40: Modelo simplificado de una transmisión.

El par presente en el disco de fricción del embrague se representa en la figura 5.41. En ella se puede observar con claridad los saltos que se producen en el par transmitido por el embrague, los cuales son debidos a los distintos modos de operación del embrague, que ya fueron estudiados en el apartado 4.8.1 del capítulo anterior. Así, entre los instantes *t=0[s]* y *t=2[s]* el par transmitido es nulo ya que el embrague no está activado, entre *t=2[s]* y *t=2,98[s],* el par transmitido es máximo mientras exista deslizamiento, y por último, a partir de *t=2,98[s]* el embrague está acoplado, y por tanto, el par transmitido durante esta fase debe ser menor que el transmitido durante la fase de acoplamiento, de otro modo, el embrague no podría acoplarse ni permanecer acoplado. Además es evidente que una vez el sistema está acoplado, el par transmitido por el embrague debe ser menor que el par motor, ya que parte de éste es almacenado en forma de energía cinética en la inercia del motor.

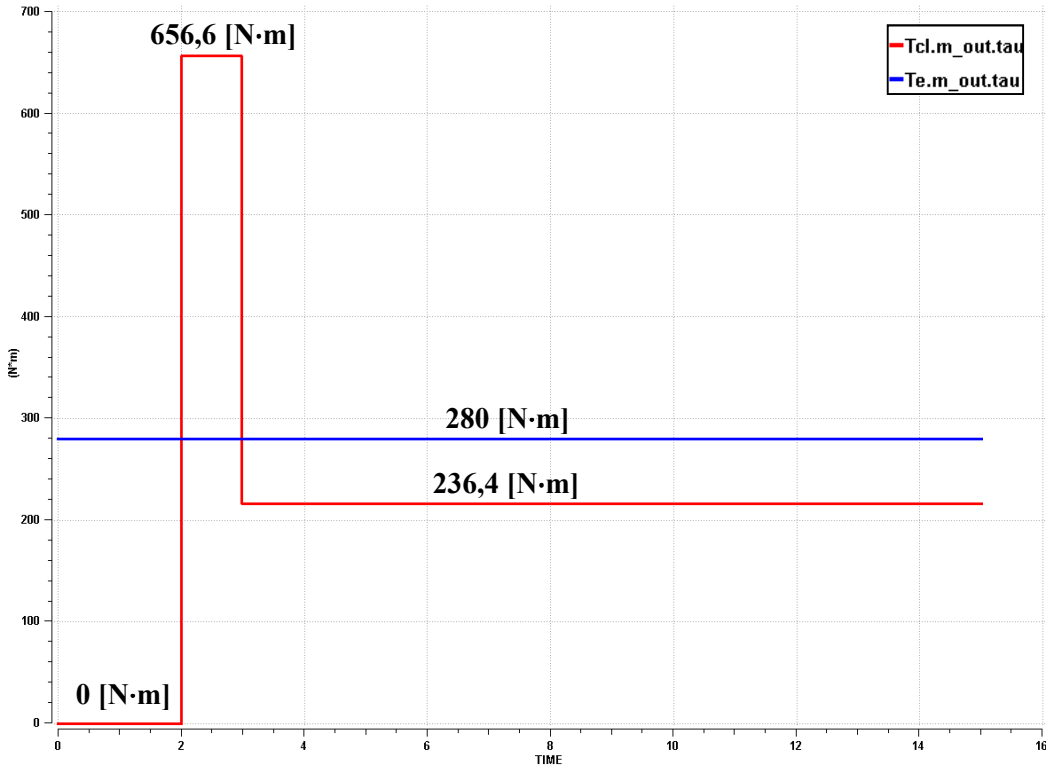


Figura 5.41: Evolución temporal del par motor y el par en el embrague.

Por último, para verificar el funcionamiento del componente *R_Clutch_jagm* se comparará el par obtenido en el embrague mediante el sistema *Transmisión_simple* durante la fase de acoplamiento, con el obtenido de forma analítica vía (5.24)

$$N = \frac{4}{3} \cdot \mu \cdot \frac{(R_e^3 - R_i^3)}{(R_e^2 - R_i^2)} \cdot F_n \tag{5.24}$$

donde:

$$F_n = p \cdot \pi \cdot (R_e^2 - R_i^2) \tag{5.25}$$

Despejando de (5.25) la presión p ya se puede calcular el par en el embrague

$$N = \frac{4 \cdot \pi \cdot p \cdot \mu}{3} \cdot (R_e^3 - R_i^3) = 656{,}5[Nm] \qquad (5.26)$$

Por tanto, si se comparan los resultados para el par en el embrague durante la fase de acoplamiento, obtenidos, por un lado mediante la simulación en Ecosimpro (*656,6 [N·m]*,Figura 5.41), y por otro, mediante expresiones analíticas (*656,5 [N·m]*,ecuación (5.26)), se puede afirmar que el funcionamiento del componente *R_Clutch_jagm* es correcto.

5.4.2. Simulación de un sistema de transmisión

Como se explicó en el capítulo 2, la simulación por ordenador es un proceso por el cual se obtiene el conocimiento del comportamiento de un sistema a partir de la observación de la conducta del modelo informático que lo representa. Por tanto, para que la simulación por ordenador de un sistema real sea satisfactoria, el modelo informático debe representar de la manera más fiel posible el comportamiento del sistema real, así, cuanto mayor fidelidad tenga el modelo, más se aproximará su comportamiento al del sistema real.

Combinando varios de los elementos incluidos en la librería *Rotational_Library*, ha sido posible modificar el sistema de la figura 5.39, hasta conseguir un modelo más detallado de una transmisión, que será simulado en la presente sección. Así, en el modelo representado en la figura 5.42, la señal de la fuente analógica *Engine* define el par motor *Te.* La inercia del motor *Je,* se conecta con el embrague a través de un eje supuesto rígido. Junto al embrague se han añadido los elementos *Jcl* y *k1*, que representan la inercia del plato de fricción y la rigidez del sistema elástico torsional del embrague respectivamente. Los elementos *J2* e *i*, representan la inercia y la relación de transmisión de la caja de cambios. El par transmitido por la caja de cambios se transmite a las ruedas conductoras, representadas mediante la inercia *J3*, a través de un eje con una rigidez *k2*. La inercia del resto del vehículo, incluidas las ruedas conducidas, se representa mediante *Jv.* La tracción de las ruedas con el asfalto ha sido modelada mediante un constante amortiguadora *b,* que actúa entre la inercia de las ruedas y la inercia del vehículo [DAS03]. Además, se han incluido dos actuadores de par *Tdriven* y *Tdriving*, que representan las resistencias externas que experimenta tanto el vehículo y como los neumáticos (rodadura, gravitacionales y aerodinámicas).

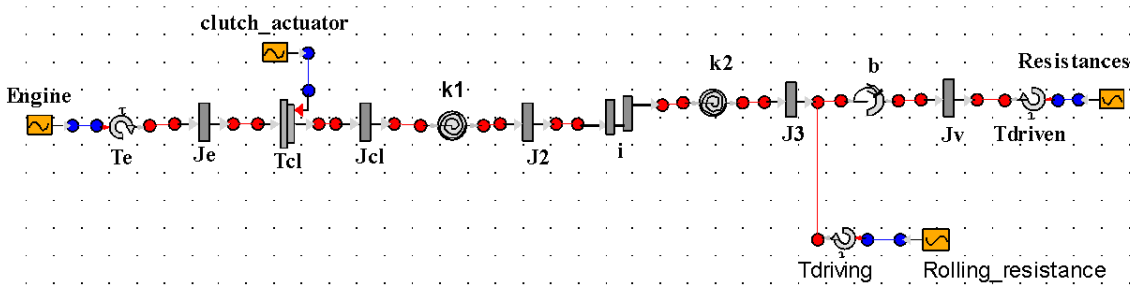


Figura 5.42: Esquema del sistema *Transmisión*.

Las instancias que se representan en la tabla 5.9, sirven para simular el sistema *Transmisión*, suponiendo que éste forma parte de un vehículo con una velocidad inicial

nula, cuyo caja de cambios, que es conectada con el motor vía *Tcl* a partir del instante *t=2[s],* tiene una relación de transmisión de valor *i=3[-]* . En la siguiente tabla no se han incluido los elementos *Rolling_resistance* y *Resistances*, ya que por simplicidad, se ha supuesto que no existen resistencias externas.

Tabla 5.9: Resumen instanciación sistema *Transmisión*.

Elemento	Parámetro o dato	Tipo	Valor	Unidades
Engine	*Source*	ENUMERATIVO	*Source_constant*	[-]
	Amp	REAL	*280*	[-]
	Tstart	REAL	*0*	[s]
Clutch_actuator	*Source*	ENUMERATIVO	*Source_constant*	[-]
	Amp	REAL	*1*	[-]
	Tstart	REAL	*2*	[s]
J_e	*I*	REAL	*0.156*	[kg·m^2]
	phi_0	REAL	*0*	[rad]
J_{cl}	*I*	REAL	*0.5*	[kg·m^2]
	phi_0	REAL	*0*	[rad]
J_2	*I*	REAL	*0.2*	[kg·m^2]
	phi_0	REAL	*0*	[rad]
J_3	*I*	REAL	*1.7*	[kg·m^2]
	phi_0	REAL	*0*	[rad]
J_v	*I*	REAL	*115*	[kg·m^2]
	phi_0	REAL	*0*	[rad]
k_1	*phi_rel_0*	REAL	*0*	[rad]
	k_tor	REAL	*1182*	[N·m/rad]
	phi_natural	REAL	*0*	[rad]
k_2	*phi_rel_0*	REAL	*0*	[rad]
	k_tor	REAL	*6000*	[N·m/rad]
	phi_natural	REAL	*0*	[rad]
b	*phi_rel_0*	REAL	*0*	[rad]
	d	REAL	*500*	[N·m·s/rad]
i	*ratio*	REAL	*3*	[-]
T_{cl}	*phi_rel_0*	REAL	*0*	[rad]
	data	ENUMERATIVO	*force*	[-]
	clutch	ENUMERATIVO	*new*	[-]
	mue_pos	TABLA_1D	0,4 ∀ *w*	[-]
	peak	REAL	*1,1*	[-]
	x	REAL	*8100*	[N]
	n_disc	REAL	*1*	[-]
	w_rel_i	REAL	*0*	[rad/s]
	Ri	REAL	*0,08*	[m]
	Re	REAL	*0,12*	[m]

Una vez realizadas todas las instancias se procede a la simulación del modelo. El objetivo principal de cualquier sistema de transmisión consiste en transmitir potencia de

un elemento a otro, en este caso del motor a las ruedas. Pero, para que esta transmisión de potencia sea posible, es necesario que el embrague esté *acoplado*. El acoplamiento del embrague puede ser detectado a partir de la diferencia en las velocidades de giro del motor y el embrague (Figura 5.43).

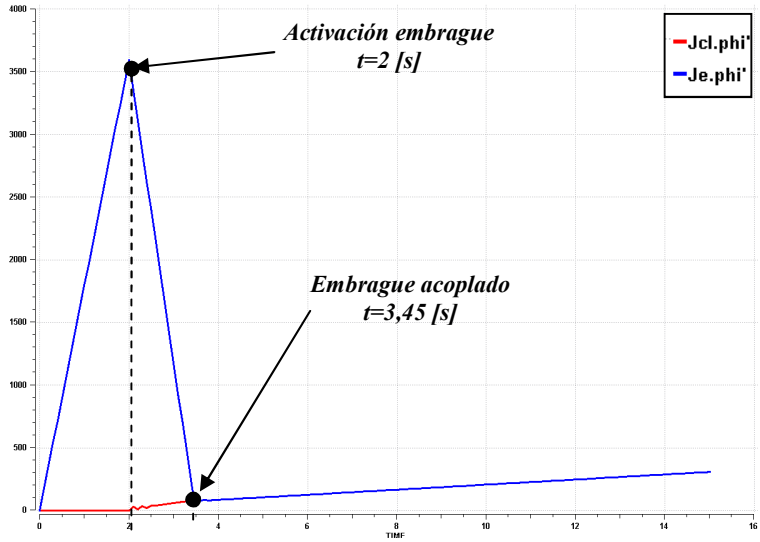


Figura 5.43: Diferencia velocidades de giro de motor y embrague.

Como se observa en el gráfico de la figura anterior, la diferencia entre las velocidades de giro de motor y embrague disminuye hasta hacerse nula y, cuando esto ocurre, el embregue está acoplado.

El par generado en la superficie de fricción del embrague se representa en la figura 5.44, y al igual que sucedía en la transmisión simulada en la sección anterior (ver figura 5.41), el par transmitido cuando el embrague está acoplado debe ser menor que el transmitido durante la fase de acoplamiento, y además el par transmitido por el embrague debe ser menor que el par motor, debido a la energía cinética almacenada en las inercias. Por otro lado, las oscilaciones presentes en el par del embrague (Figura 5.45) se deben, en parte a los muelles con rigidez *k1* del sistema elástico torsional del embrague, y en parte a la rigidez *k2* del eje de la transmisión.

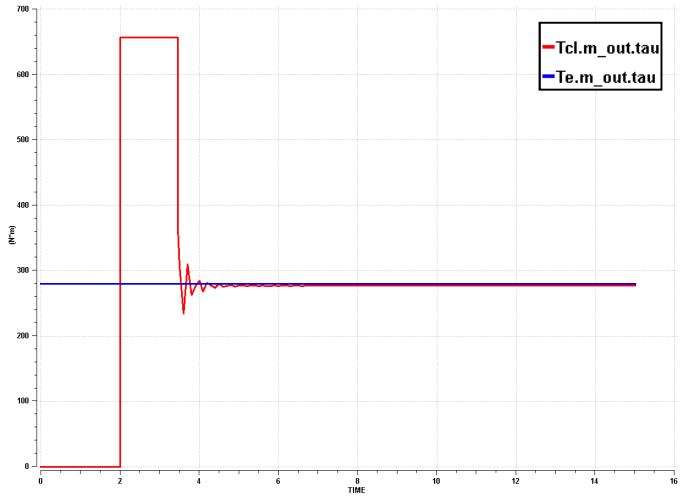


Figura 5.44: Par generado en embrague y motor.

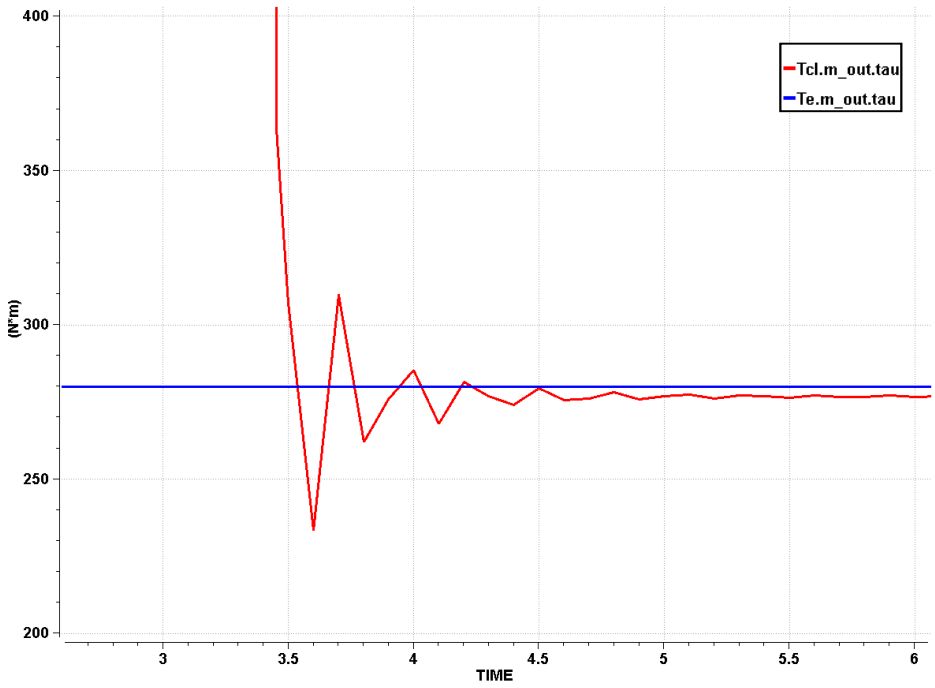


Figura 5.45: Detalle vibraciones generadas por sistema elástico torsional y rigidez ejes.

En una transmisión real estas vibraciones se disiparían más rápido, ya que los ejes reales presentarían un cierto efecto amortiguador. El par entregado por la transmisión a las ruedas conductoras se muestra en la figura 5.46. Este par, alcanza un valor máximo durante la fase de acoplamiento, ya que durante esta fase el par transmitido por el embrague es máximo. Una vez el embrague está acoplado, el par transmitido por las ruedas conductoras disminuye hasta un valor menor y permanece constante. De nuevo, las vibraciones causadas por los muelles del sistema elástico torsional disminuyen lentamente.

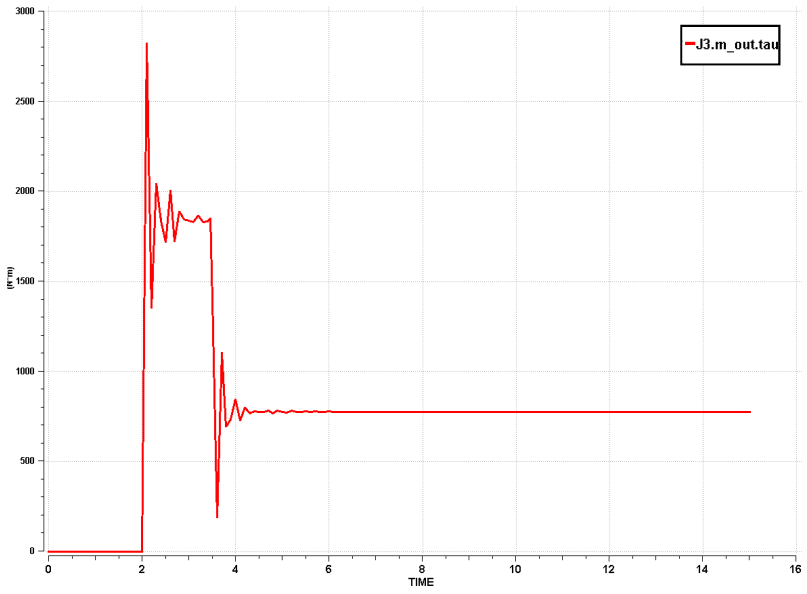


Figura 5.46: Par entregado a las ruedas conductoras.

Con el fin de mejorar el sistema de transmisión, es posible introducir en el modelo una nueva constante amortiguadora, que simule un cierto efecto amortiguador en el eje de la transmisión tal y como se detalla en la figura 5.47 [BAT10].

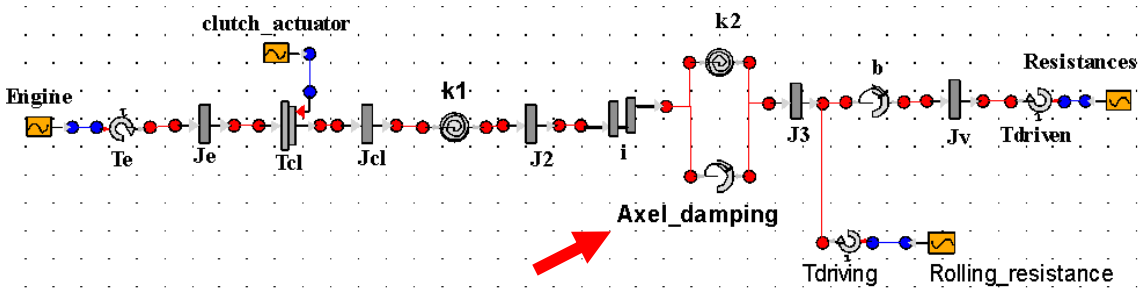


Figura 5.47: Modelo del sistema de transmisión con efecto amortiguador en los ejes.

La instanciación del nuevo componente se presenta en la tabla tabla 5.10:

Tabla 5.10: Instanciación componente *Shaft_damping*.

Elemento	Parámetro o dato	Tipo	Valor	Unidades
Shaft_damping	*phi_rel_0*	REAL	0	[rad]
	d	REAL	100	[N·m·s/rad]

El par generado en las superficies de fricción del modelo con efectos amortiguadores en los ejes, se representa figura 5.48. En ella se puede observar que las oscilaciones causadas por los resortes del embrague y la rigidez de los ejes se amortiguan mucho antes que en el modelo anterior.

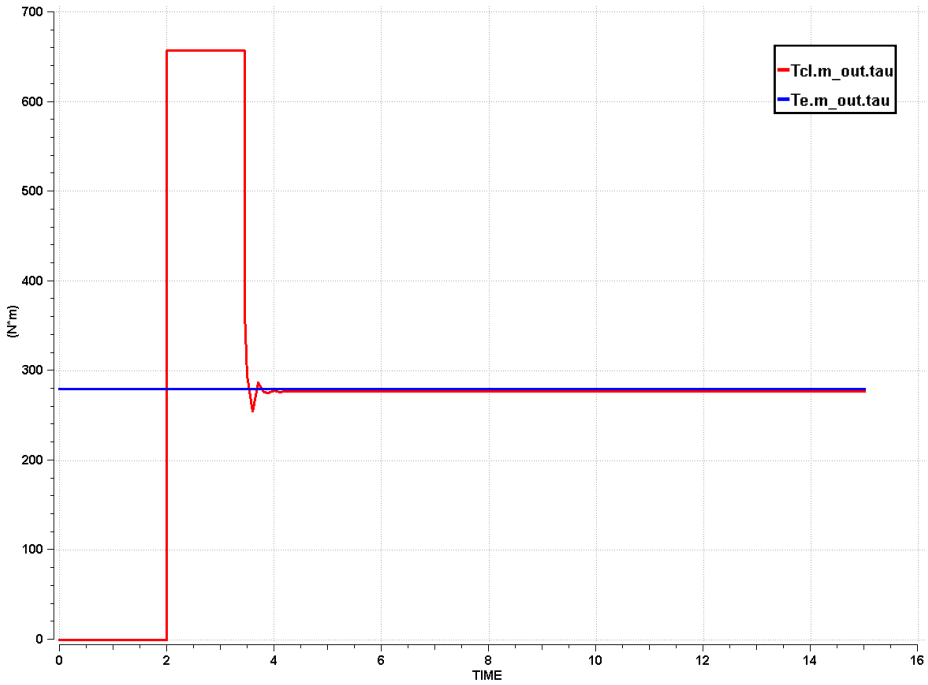


Figura 5.48: Par generado en las superficies de fricción en modelo con ejes amortiguados.

Mientras que en este último modelo las oscilaciones se hacen imperceptibles a partir del instante *t=4,5[s]* (figura 5.49), en el modelo sin amortiguación en los ejes estas vibraciones siguen existiendo pasado este instante.

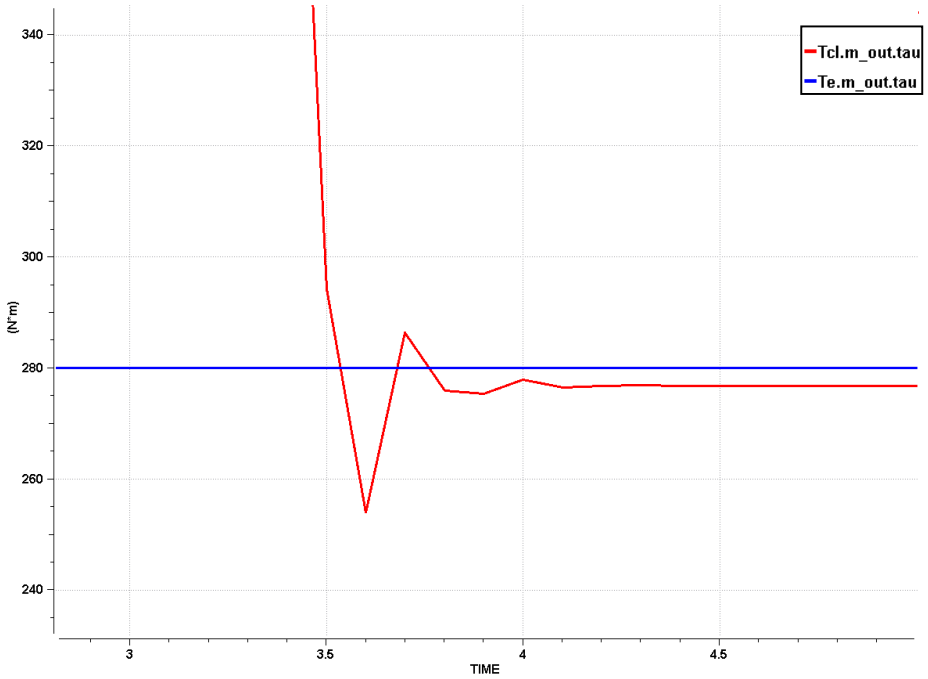


Figura 5.49: Detalle vibraciones generadas por sistema elástico torsional y rigidez ejes en modelo con componente *Shaft_damping*.

En el par entregado a las ruedas motrices, también se observa una reducción de las oscilaciones tras la incorporación del componente *Shaft_damping* (Figura 5.50).

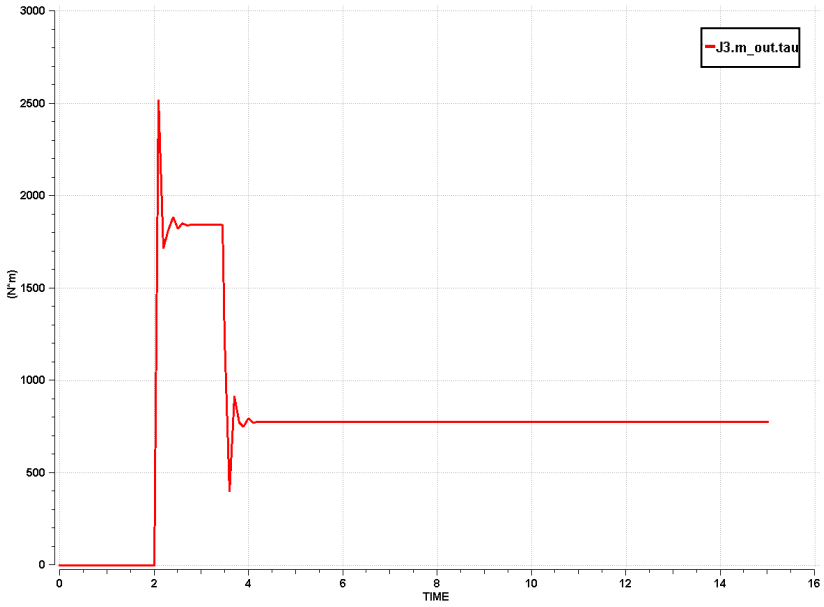


Figura 5.50: Detalle vibraciones generadas por sistema elástico torsional y rigidez ejes en modelo con componente *Shaft_damping*.

Por tanto, a partir de la simulación de un modelo inicial (figura 5.42), ha sido posible corregir ciertos resultados no deseados mediante la incorporación de un único componente, dando lugar a un nuevo modelo (figura 5.47).

En definitiva, se puede concluir afirmando que la dinámica utilizada en esta sección, representa fielmente la esencia del modelado y la simulación por computadora de sistemas, cuyas numerosas ventajas han sido analizadas a lo largo del presente documento. Por tanto, los componentes modelados quedan validados.

CAPÍTULO 6

CONCLUSIONES Y DESARROLLOS FUTUROS

6.1 CONCLUSIONES

En este Proyecto Fin de Carrera se ha desarrollado una librería que, posteriormente, ha sido utilizada para modelar y validar sistemas de dinámica rotacional.

La herramienta utilizada para tal fin ha sido EcosimPro, que constituye un software donde se recogen todas las ventajas de la programación orientada a objetos (MOO), como son la *herencia* y la *abstracción*, junto con la capacidad de reutilización de los componentes generados en otros contextos de simulación, ya sea variando las condiciones de operación, ó utilizando los componentes en la construcción de modelos más complejos.

Además el uso de la metodología MOO tiene una influencia directa en la reducción del elevado tiempo que tradicionalmente supone la obtención de nuevos modelos, como se demuestra en la sección 5.4.2 del capítulo 5, en la que se crea un nuevo modelo para eliminar un comportamiento no deseado de forma rápida y sencilla.

Otra ventaja del MOO es el elevado grado de confianza en la validez de los modelos construidos. Volviendo al ejemplo utilizado en el párrafo anterior, el nuevo modelo se construye mediante una pequeña modificación de un modelo ya validado, por lo que éste último goza a priori de una validez constatada.

En definitiva, EcosimPro reúne todas las ventajas que presentan los lenguajes de programación orientados a objetos y, además, añade las del potente entorno gráfico del que dispone, que permite modelar multitud de sistemas en un tiempo muy reducido y de un modo sencillo. Todas estas ventajas hacen de EcosimPro una potente herramienta, tanto a nivel docente como en el ámbito profesional.

6.2 DESARROLLOS FUTUROS

No cabe duda que los componentes cuya implementación ha sido más complicada, y por tanto han requerido un mayor tiempo de desarrollo, son *R_Drum_and_disc_Brake_jagm* y *R_Clutch_jagm*. No obstante, y aunque estos componentes presentan una gran versatilidad en lo que a características físicas y geométricas se refiere, todavía disponen de un amplio margen de desarrollo del elemento *R_Absfiction,* que supone la base del funcionamiento de los elementos antes mencionados. Así, en lugar de modelar la fricción a partir del modelo clásico de Coulomb, se podrían implementar nuevos

modelos de fricción basados en otros modelos, como por ejemplo el modelo de *LuGre* [CAN95], el de *Karnopp* [KAR85], el de *Dahl* [DAH75] ó cualquier otro.

En lo que referente a la simulación de sistemas, parece evidente que existe un amplio horizonte por explorar. En el caso del sistema de transmisión simulado en el capítulo 5, introducir condiciones de operación más realistas como las propuestas en [GAR02], contemplar efectos térmicos, mejorar el comportamiento no lineal de resortes ó imponer condiciones más realistas al par generado por el motor son objeto de estudio, en la actualidad, por parte de numerosos autores.

BIBLIOGRAFÍA

[BAT10] BATAUS, M.(2010). Automotive clutch models for real time simulation. The publishing House of the Romanian Academy, University of Bucarest, Automotive Engineering Department.

[BRA05] BRAVO, J. E. & BOTERO, A .J. & BOTERO, M. (2005). El método de Newton-Raphson: la alternativa del ingeniero para resolver sistemas de ecuaciones no lineales. *Scientia et Technica* Año XI, No 27, Abril 2005. UTP. ISSN 0122-1701.

[CAN95] CANUDAS, C. & OLSSON, H. & ASTROM, K.J. & LISCHINSKY, P. (1995). *A new model for control of systems with friction*. IEEE Control Systems Society, 10.1109/9.376053.

[DAH75] DAHL, P. (1975). *Solid friction damping of spacecraft oscillations*.

[DAS03] DASSEN, M.H. (2003) *Modelling and control of automotive clutch systems*. Einthoven, 22nd July 2003. Department of mechanical engineering.

[EAI09] EA International (2009). *EcosimPro modelling language version 4.6 user manual*.

[EAI32] EA International. EcosimPro 3.2 documentation.

[FRI03] FRITZSON, P. (2003). *Introducción al Modelado y Simulación de Sistemas Técnicos y Físicos*. ISBN 84-611-2094-9.

[GAR02] GAROFALO, F. & GLIEMO, L. & IANNELLI, L. & VASCA, F. (2002). Tracking control of dry clutch engagement for Vehicle launch. Universita di Napoli Federico II - Dipartimento di Informatica e Sistemistica.

[GRE98] GRECA, I. M. & MOREIRA,M. A.(1998). Modelos mentales y aprendizaje de física y electricidad y magnetismo. *Enseñanza de las ciencias*, 16(2), 298-303.

[GUA02] GUASCH A. & PIERA, M.A. & CASANOVAS, J. & FIGUERAS, J. (2002). Modelado *y simulación: Aplicación a procesos logísticos de fabricación y servicios*.Ed. UPC, ISBN:84-8301-704-0.

[JOR08] JORRÍN, A. & PRADA, C. & COBAS, P. (2008). *EcosimPro and its Object-Oriented Modeling Language*. Department of Systems Engineering and Automatic Control, University of Valladolid, Spain.

[KAR85] KARNOPP, D. (1985) *Computer simulation of slip-stick friction in mechanical dynamic systems*. Journal of Dynamic Systems, Measurement, and Control volume 107.

[KIK05] KIKUUWE, R. & TAKESUE, N. & SANO, A. & MOCHIYAMA, H. & FUJIMOTO, H. (2005). *Fixed-Step friction simulation: from classical Coulomb model to modern continuous models.* Touch Tech Lab Funded By Toyota, Nagoya Institute of Technology, Nagoya, Aichi 466-8555, Japan.

[MOD02] MODELICA ASSOCIATION. (2002). *Modelica language specification, version 2.0*.

[LAW91] LAWRENCE, J.A. (1991). *The role of JSC Engineering Simulation in the Apollo Program.* Simulation July 1991,vol.57, nº 1, 9-16.

[LOP12] LOPEZ, A. *Lubricación* [en línea]. Departamento de Ingeniería Industrial, Universidad Antonio Nebrija, Madrid. [ref. de 29 de febrero de 2012]. Disponible en web: http://www.nebrija.es/~alopezro/Lubricacion.pdf

[OTT99] OTTER, M. & ELMQUIST, H. & MATTSSON, S. E. (1999). *Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle*. CACSD ´99, August 22-26, Hawaii, USA 1999.

[PET82] PETZOLD, L. R. (1982). *A description of DASSL: a differential-algebraic system solver*. Sandia National Laboratories, Livermore.

[PIN98] PINILLA, M. C. (1998). *Los métodos de Runge-Kutta en la resolución numérica de ecuaciones diferenciales*. Academia de ciencias exactas, físicas, químicas y naturales de Zaragoza.

[ROB04] ROBERT, C.P. (2004). Monte Carlo Statistical Methods. Ed. Springer.ISBN 0-387-21239-6.

[SAN05] SAN ROMÁN, J.L. & MUÑOS, B. & BOADA, B.L. & QUESADA, A. (2005). *Frenos y embragues*. Apuntes asignatura Diseño de Máquinas, departamento de Ing. Mecánica, Universidad Carlos III de Madrid.

[SAN12] SAN ROMAN, J. L. Lubricación [en línea]. Material de estudio asignatura diseño de Máquinas, Open Course Ware, Universidad Carlos III de Madrid [ref. de 29 de febrero de 2012]. Disponible en web: http://ocw.uc3m.es/ingenieria-mecanica/diseño-de-maquinas/material-de-estudio

[STR97] STROUSTRUP, B.(1997). The C++ programming language, third edition.Ed. Addison-Wesley ISBN 0-201-88954-4.

[TIL01] TILLER, M.M. (2001). *Introduction to physical modelling with Modelica*. Kluwer Academic Publishers.

[URQ00] URQUÍA, Alonso (2000). *Modelado Orientado a Objetos y Simulación de Sistemas Híbridos en el ámbito del Control de Procesos Químicos.* PhD thesis, Facultad de Ciencias. Universidad Nacional de Educación a Distancia.

[URQ01] URQUÍA, A. (2001) Simulación, texto base de teoría. Departamento de Automática e Informática, Escuela Técnica Superior de Ing. Informática, UNED, Madrid, España.

[YEB03] YEBRA, L.J & VARA, R.P & DORMIDO, S. & BERENGUEL, M. (2003). *Comparación entre Modélica 2.0 y EcosimPro 3.2*. 2ª Reunión de usuarios de EcosimPro, UNED, Madrid Febrero 2003.