ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

ÁREA DE CONOCIMIENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

TRABAJO FIN DE GRADO

MODELADO E IDENTIFICACIÓN DEL SISTEMA DE SUSPENSIÓN PASIVA DEL ROBOT MÓVIL ANDÁBATA

AUTOR ANTONIO JAVIER GUERRERO ANGULO

TUTOR DR. JORGE LUIS MARTÍNEZ RODRÍGUEZ

TITULACIÓN GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

MÁLAGA, SEPTIEMBRE DE 2015

Dedicado a mi familia

y amigos.

Resumen

Este Trabajo Fin de Grado trata sobre la obtención y simulación de un modelo de la suspensión del robot móvil Andábata. El modelo elegido para ello es el conocido como de un cuarto de vehículo, completándolo con los choques producidos en los extremos del recorrido de la suspensión.

Se han registrado datos reales mediante un circuito de instrumentación basado en un potenciómetro lineal. Por su parte, para la obtención de los distintos parámetros del modelo se ha recurrido a la estimación mediante simulación con optimización *Simplex*.

También, se ha programado una aplicación en Matlab denominada *mcvAndábata* que permite simular y variar fácilmente los parámetros del modelo de un cuarto de vehículo. Además, la aplicación es capaz de exportar datos a SolidWorks para realizar animaciones tridimensionales del sistema de suspensión.

Palabras clave

Vehículo, suspensión, modelado, simulación, identificación.

Índice general

Capítulo 1.
Introducción

1.1 Antecedentes

El objetivo de la suspensión de un vehículo es doble; por un lado regula el movimiento vertical de la rueda asegurando el contacto entre el neumático y el terreno; por otro lado aísla el chasis de las irregularidades del suelo amortiguando vibraciones y oscilaciones.

La suspensión pasiva se caracteriza por tener parámetros constantes predeterminados y de no requerir de ningún actuador o sensor. Este tipo de suspensión es la más utilizada en automóviles de gama media y baja.

Para el estudio de la suspensión pasiva se suele emplear un modelo lineal de un cuarto de vehículo. Dicho modelo representa la suspensión como un sistema de doble masa – resorte – amortiguador de dos grados de libertad [4].

1.2 Motivación personal

Considero los proyectos en el ámbito de la ingeniería un reto y una motivación para estudiantes que, como en mi caso, sienten la necesidad de comprender la explicación de cualquier fenómeno físico. Por tanto, me es de especial interés el presente Trabajo Fin de Grado (TFG) para madurar mis conocimientos y determinar cuáles son las limitaciones de las aproximaciones matemáticas a dichos fenómenos.

Por otro lado, durante mis años de estudio he desarrollado interés por el área de conocimiento de Ingeniería de Sistemas y Automática (ISA). En concreto, este trabajo se centra en el modelado y simulación de sistemas, la primera etapa en un estudio de ISA, del que depende el éxito tanto económico como de prestaciones del sistema implementado.

1.3 Objetivos

El objeto principal del presente TFG es el de caracterizar y modelar el sistema de suspensión pasiva del robot móvil para exteriores Andábata.

Para alcanzar dicho fin se realizará un estudio teórico sobre la naturaleza de su sistema de suspensión, empleando un modelo lineal de un cuarto de vehículo que represente su dinámica de la forma más fiel posible y completándolo con los choques en los extremos del recorrido.

De una forma desglosada, se pueden definir los objetivos del presente trabajo como sigue:

1. Obtener un modelo teórico de un cuarto de vehículo, usando para ello ecuaciones y variables de estado.

2. Realizar un montaje de instrumentación electrónica que permita registrar el desplazamiento de las masas implicadas.

3. Determinar los parámetros del modelo matemático.

4. Crear un software, usando el entorno Matlab, que simule el comportamiento de la suspensión. Además, debe mostrar los resultados de interés de una forma clara y permitir la modificación de parámetros de manera cómoda para el usuario.

5. Usar los resultados calculados por el software para animar un modelo tridimensional en SolidWorks del robot.

1.4 Fases del trabajo

Las tareas llevadas a cabo durante la realización del presente trabajo han sido las siguientes:

1. Familiarización con el robot móvil Andábata.

2. Propuesta de un modelo lineal con recorrido limitado para un cuarto de vehículo.

3. Identificación experimental de los parámetros del modelo, registrando la respuesta ante condiciones iniciales mediante instrumentación electrónica.

4. Simulación en Matlab del comportamiento de la suspensión ante diversas situaciones de navegación.

5. Programación de una aplicación Matlab.

6. Exportar los datos generados por Matlab a un modelo tridimensional en SolidWorks para, posteriormente, animarlo.

7. Elaborar la presente memoria.

8. Preparar la presentación ante el tribunal calificador.

1.5 Estructura del Trabajo Fin de Grado

Este TFG consta de seis capítulos y dos anexos. La estructura de la memoria y su contenido es la siguiente:

Capítulo 1. Introducción

Supone un acercamiento al contenido del TFG.

Capítulo 2. El robot móvil Andábata y su sistema de suspensión

Describe las características del sistema de suspensión del robot móvil.

Capítulo 3. Modelado de un cuarto de vehículo

Detalla el modelo usado para el estudio de la suspensión del vehículo.

Capítulo 4. Identificación de parámetros

Trata sobre cómo se han determinado los distintos parámetros que son necesarios para el modelo.

Capítulo 5. La aplicación Matlab *mcvAndábata*

Introduce la aplicación *mcvAndábata* y su interfaz de usuario.

Capítulo 6. Conclusiones y trabajos futuros

Reflexiona sobre la labor realizada en este TFG.

Anexo A. Códigos Matlab

Presenta todos los códigos escritos en Matlab para la elaboración del presente TFG.

Anexo B. Plano del sistema de suspensión de Andábata.

Bibliografía

Fuentes de consulta a las que se ha recurrido.

Capítulo 2.
El robot móvil Andábata y su sistema de suspensión

2.1 Introducción

El robot móvil Andábata es fruto del *Proyecto de Investigación de Excelencia de la Junta de Andalucía P10 – TEP – 6101 – R* por el que se propone la navegación autónoma de un robot móvil 4x4 en entornos naturales mediante GPS y telémetro láser tridimensional (3D) [13]. Para ello cuenta con direccionamiento por deslizamiento (*skid – steer*) de cuatro ruedas, cada una con su propio motor. El movimiento es el resultado de combinar las velocidades de las ruedas del lado izquierdo con las del lado derecho. Esta característica dota a Andábata de una mayor estabilidad del chasis y de la capacidad de girar sobre sí mismo, aumentando así la maniobrabilidad en entornos más comprometidos [6]. El sistema de suspensión es independiente para cada una de las cuatro ruedas.

Las dimensiones de Andábata son reducidas, apenas 49 x 64 x 76 cm (ver *Figura 2.1*). Su chasis se compone de tres niveles:

- El nivel superior contiene los sistemas de percepción y comunicación.

- El nivel intermedio alberga el sistema de control de los motores de las ruedas y la computadora, así como una fuente de alimentación.

- El nivel inferior contiene una batería de iones de litio con una capacidad total de 7,2 Ah y 29,6 V de tensión continua máxima.

En este capítulo se describe el sistema de suspensión del robot Andábata y sus características principales. También se presenta cómo se han obtenido los valores de las masas que entran en juego en el modelo del vehículo y los límites del recorrido.

Figura 2.1. El robot móvil Andábata.

2.2 Características del sistema de suspensión

El diseño de un sistema de suspensión pasiva es de vital importancia para la navegación. Razón de ello es que dicho tipo de suspensión no está sometida a regulación alguna por parte de actuadores y sensores. Por lo tanto, los parámetros de la suspensión han de ser establecidos *a priori* y son invariantes. Es por ello que una mala elección previa puede poner en compromiso el buen funcionamiento durante el movimiento.

Una suspensión demasiado "dura" provoca un menor amortiguamiento de las irregularidades, lo que se traduce en un sufrimiento mayor de las partes internas del robot a causa de las vibraciones. Sin embargo, esta configuración aumenta considerablemente el comportamiento estable del chasis. Por otro lado, una suspensión demasiado "blanda" deriva en un mayor confort pero las oscilaciones del chasis son mayores [2].

Cada rueda del robot móvil Andábata posee una suspensión formada por una guía lineal (ver *Figura 2.2*). Esta guía solo permite un movimiento estrictamente vertical de las ruedas generando así una cierta fricción en su recorrido. Al no poseer un émbolo amortiguador como tal, este rozamiento es el único responsable de la disipación de energía cinética en forma de calor.

La masa suspendida (también se referirá a ella como chasis) se encuentra comunicada con la masa no suspendida mediante dos resortes de distintas longitudes. El papel de los resortes, en el sistema de suspensión, es el de soportar el peso propio del chasis e impedir que las vibraciones sufridas por las ruedas se transmitan al resto del vehículo. Para ello, cuando una rueda sufre una perturbación, los resortes son comprimidos o estirados, acumulando energía que deberán liberar oscilando varias veces hasta alcanzar el equilibrio. El responsable de reducir el número de oscilaciones, en una suspensión pasiva

convencional, es el amortiguador. En el caso de Andábata, la propia fricción de los patines sobre la guía juega el papel de amortiguador, como ya se ha expuesto con anterioridad.

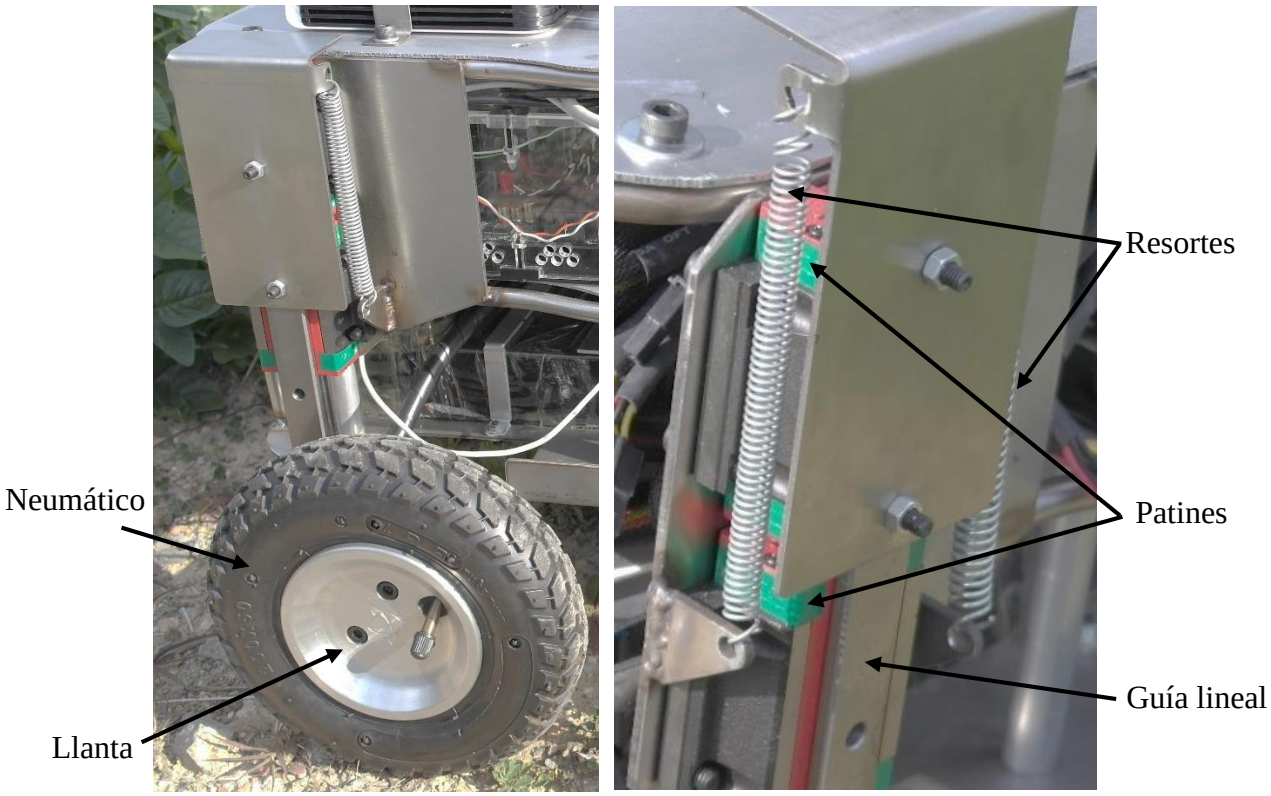


Figura 2.2. Elementos del sistema de suspensión de Andábata.

Cada uno de los cuatro sistemas de suspensión es independiente y, debido a las características mismas de su construcción, no poseen el mismo comportamiento en igualdad de condiciones. La masa no suspendida se compone de:

- Neumático.
- Llanta.
- Dos resortes.
- Guía lineal.
- Dos patines.
- Motor y reductora epicicloidal (ver *Figura 2.3*).

El contacto entre la rueda y el terreno se puede modelar como un resorte que representará la rigidez del neumático.

Las imperfecciones del terreno causan una compresión sobre el neumático, dando lugar a una fuerza que se transfiere a la masa no suspendida. Su respuesta como movimiento vertical es inversamente proporcional a la fuerza que se le transmite según la segunda ley de Newton [10, 11]. Por ello, es aconsejable diseñar esta masa de forma que su peso no sea demasiado alto. De esta forma, tendrá una respuesta rápida y absorberá de manera más eficaz las fuerzas provocadas por las irregularidades. En el Anexo B se pueden comprobar las dimensiones en milímetros asociadas al sistema de suspensión.

Figura 2.3. Conjunto motor y reductora acoplada a la rueda.

La *Figura 2.4* establece el criterio elegido de identificación numérica de las suspensiones. Se hará referencia a esta numeración a lo largo de la presente memoria.

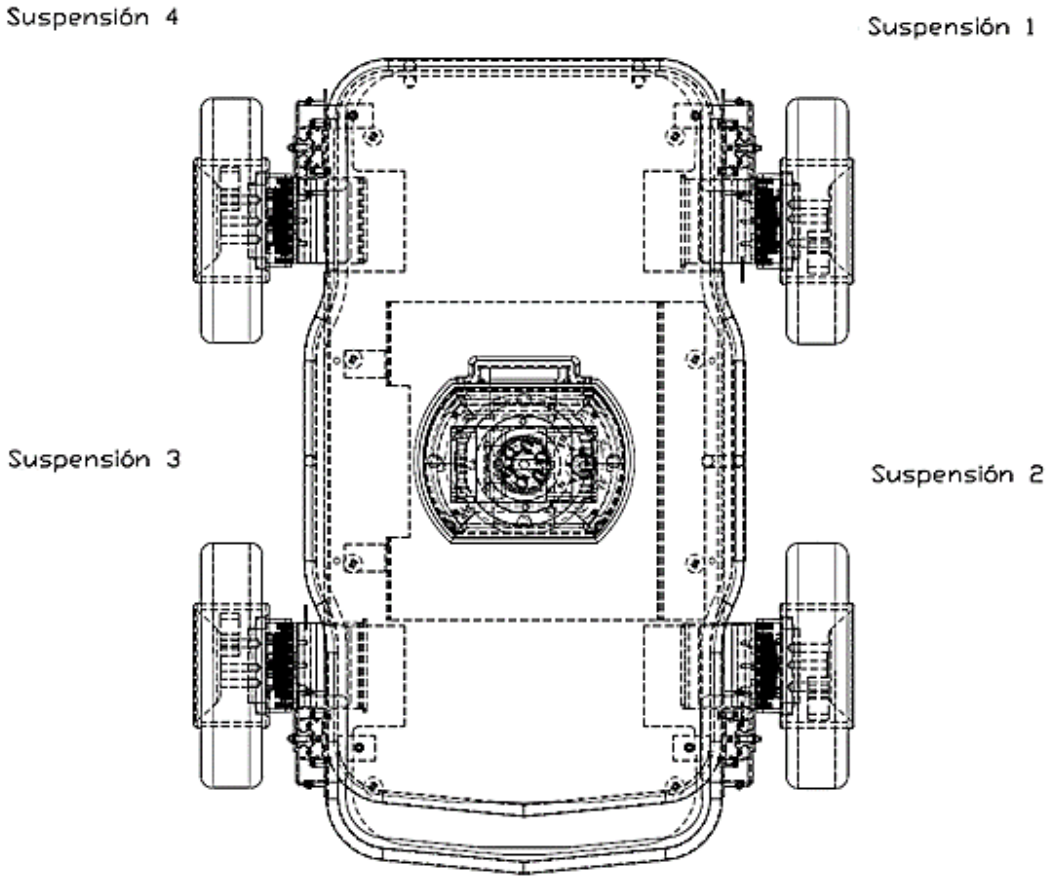


Figura 2.4. Vista superior esquemática de Andábata.

2.3 Medición de masas

Para la medida de las masas se ha utilizado una báscula industrial monocélula *gram precisión serie XK* (ver *Figura 2.5*). Esta báscula cuenta con una pantalla de dos decimales de resolución.

Una masa conocida de 80 *kg* es medida por la báscula como de 77 *kg*, por lo que se ha usado el factor de corrección:

$$F_{corrección} = \frac{80\ kg}{77\ kg} = 1{,}039 \tag{2.1}$$

Figura 2.5. Báscula industrial.

La medida de la masa total del vehículo es la siguiente:

$$M_{Total} = 41{,}92\ kg \tag{2.2}$$

Los resortes se retiraron para que no interfiriesen en la medición de la masa no suspendida. Atendiendo a la notación dada por la *Figura 2.4*, las mediciones de cada una de las cuatro masas no suspendidas se muestran en la *Tabla 2.1*.

Masa no suspendida	Masa (*kg*)
1	2,65
2	2,60
3	2,63
4	2,60
Valor medio	**2,62**

Tabla 2.1. Valores de las masas no suspendidas.

Como se puede apreciar, las variaciones entre unas masas y otras son muy pequeñas. Su valor promedio se calcula con una media aritmética de los datos de la *Tabla 2.1*:

$$M_r = 2{,}62\ kg \tag{2.3}$$

A partir del valor de las cuatro masas no suspendidas y la total del vehículo, se puede calcular la masa total del chasis ($M_{Total\ chasis}$).

$$M_{Total\ chasis} = M_{Total} - 4\,M_r = 31{,}44\ kg \tag{2.4}$$

2.4 Límites del recorrido

El recorrido de la suspensión está limitado por topes mecánicos que chocan contra el chasis tanto en el movimiento ascendente como descendente, tal y como se aprecia en la *Figura 2.6.*

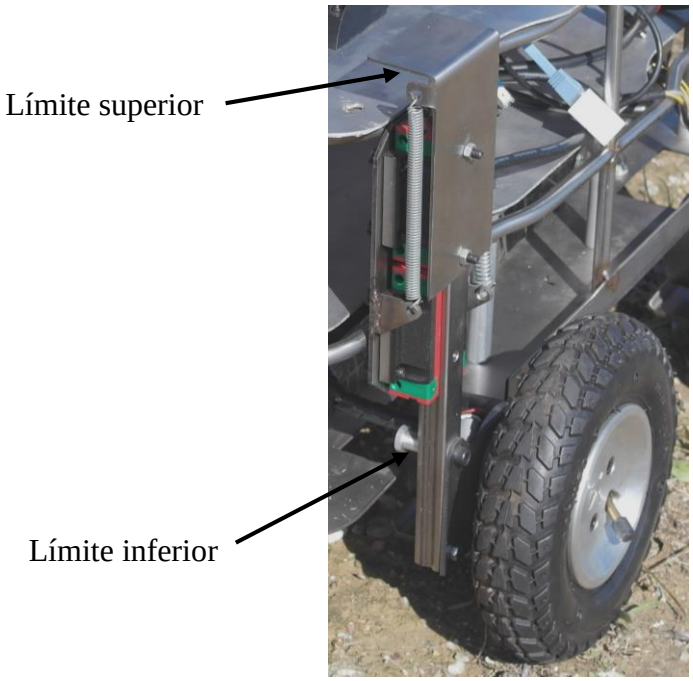


Figura 2.6. Límites de recorrido.

Por razones de construcción, los distintos recorridos de las suspensiones de Andábata pueden presentar pequeñas diferencias en cuanto a su longitud. En la *Tabla 2.2* se muestran los valores medidos para dichos recorridos.

Suspensión	Recorrido (*mm*)
1	68
2	68
3	68
4	66

Tabla 2.2. Recorrido de las suspensiones de Andábata.

Como se puede apreciar, la cuarta suspensión tiene un recorrido algo menor que las demás.

Capítulo 3.
Modelado de un cuarto de vehículo

3.1 Introducción

Para el estudio de la suspensión pasiva del robot Andábata se empleará el modelo de un cuarto de vehículo. Este modelo consta de un sistema de doble masa en el que su dinámica se ve condicionada por dos resortes y una fricción al movimiento vertical.

En la *Figura 3.1* se expone el modelo esquemático de un cuarto de vehículo. El valor de la masa no suspendida (M_1) viene dado por la *Tabla 2.1*. Suponiendo que el vehículo está nivelado y que el centro de gravedad está sobre el centro de los puntos de apoyo de las ruedas con el suelo, la masa M_2 representa un cuarto de la masa total del chasis del vehículo [3]:

$$M_2 = \frac{M_{Total\ chasis}}{4} = 7{,}86\ kg \tag{3.1}$$

Cuando el sistema se ve perturbado por las irregularidades del terreno (z_r) se producirá una evolución de la posición vertical, tanto de la masa no suspendida (z_1) como del chasis (z_2) de carácter oscilatorio.

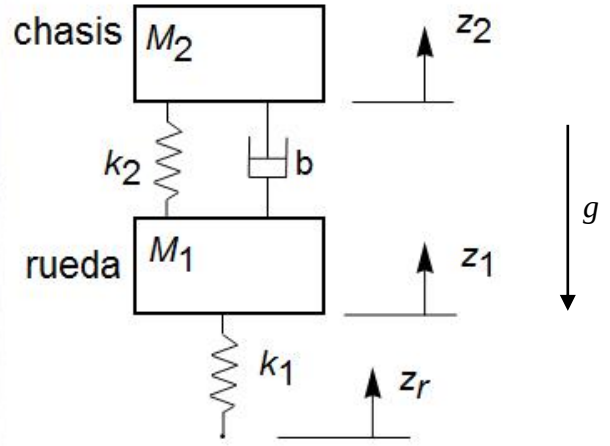


Figura 3.1. Modelo esquemático de un cuarto de vehículo.

3.2 Modelo matemático

En primer lugar, se plantea el diagrama de cuerpo libre del sistema en ausencia de perturbaciones externas. Las masas M_1 y M_2 de la suspensión solo se verán sometidas a fuerzas gravitacionales con aceleración g de 9,81 m/s^2 (ver *Figura 3.2*).

Empleando la tercera ley de Newton (principio de acción y reacción) se obtienen las fuerzas de reposo de la masa no suspendida (F_{01}) y del chasis (F_{02}):

$$F_{01} = (M_1 + M_2)g \tag{3.2}$$

$$F_{02} = M_2\, g \tag{3.3}$$

Según estas relaciones, el chasis está sometido a una fuerza proporcional a su masa (M_2) mientras que la fuerza de la masa no suspendida es proporcional al conjunto de masas rueda-chasis (M_1+M_2).

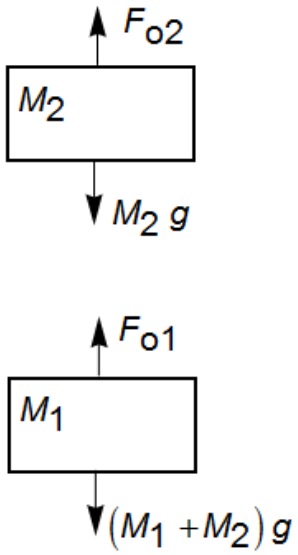


Figura 3.2. Diagrama de cuerpo libre en reposo.

Si ahora este sistema en reposo se ve perturbado por irregularidades del terreno, el diagrama de cuerpo libre resultante será el de la *Figura 3.3*.

Aplicándole la segunda ley de Newton se obtiene:

$$M_2 \frac{d^2 z_2}{dt^2} = F_{\mathrm{o2}} - F_{\mathrm{k2}} - M_2 g - F_b \tag{3.4}$$

$$M_1 \frac{d^2 z_1}{dt^2} = F_{\mathrm{o1}} + F_{\mathrm{k2}} + F_b - (M_1 + M_2)g - F_{\mathrm{k1}} \tag{3.5}$$

Donde F_{k1} y F_{k2} son las fuerzas asociadas a los resortes que, según la ley de Hook, son proporcionales a la constante elástica (k) y a la elongación de los mismos. La fuerza F_b modela la fricción, que es proporcional a la constante viscosa (b) y a la tasa de cambio de la posición vertical.

$$F_{k1} = k_1(z_1 - z_r) \tag{3.6}$$

$$F_{k2} = k_2(z_2 - z_1) \tag{3.7}$$

$$F_b = b(z_2' - z_1') \tag{3.8}$$

Desarrollando y empleando (3.2) y (3.3) además de (3.6), (3.7) y (3.8), las fuerzas gravitacionales se eliminan y las ecuaciones diferenciales quedan de la siguiente forma:

$$M_2 z_2'' = -k_2(z_2 - z_1) - b(z_2' - z_1') \tag{3.9}$$

$$M_1 z_1'' = k_2(z_2 - z_1) + b(z_2' - z_1') - k_1(z_1 - z_r) \tag{3.10}$$

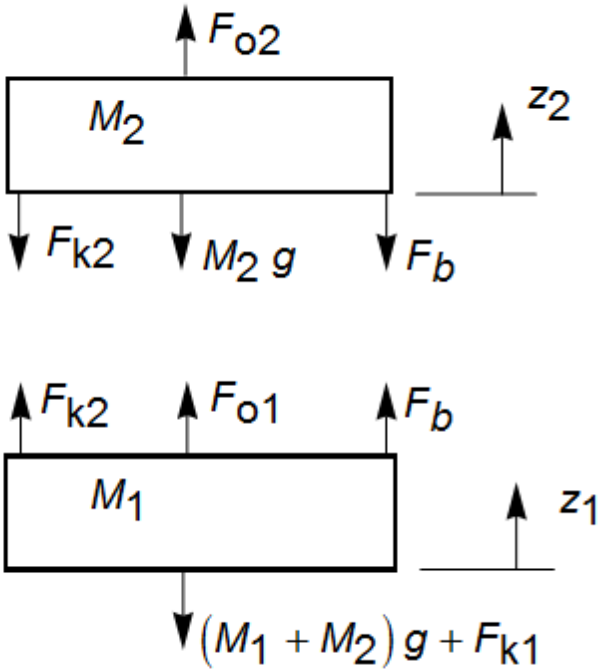


Figura 3.3. Diagrama de cuerpo libre ante perturbación del terreno.

3.3 Representación en el espacio de estados

Se considera como entrada al sistema:

$u = z_r$ (perfil del terreno),

y como variables de estado las siguientes:

$x_1 = z_1$ (posición vertical de la masa no suspendida),

$x_2 = z_1'$ (velocidad vertical de la masa no suspendida),

$x_3 = z_2$ (posición vertical del chasis),

$x_4 = z_2'$ (velocidad vertical del chasis).

Sustituyendo las anteriores variables de estado en las ecuaciones diferenciales lineales (3.9) y (3.10) se obtienen las ecuaciones de estado:

$$\begin{aligned} x_1{}' &= x_2 \\ x_2{}' &= \frac{b(x_4 - x_2) + k_2 x_3 - (k_1 + k_2)x_1 + k_1 u}{M_1} \\ x_3{}' &= x_4 \\ x_4{}' &= \frac{-(k_2(x_3 - x_1) + b(x_4 - x_2))}{M_2} \end{aligned} \tag{3.11}$$

Considerando la salida como la posición vertical del chasis se obtiene la representación interna:

$$\begin{pmatrix} x_1{}' \\ x_2{}' \\ x_3{}' \\ x_4{}' \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{-(k_1+k_2)}{M_1} & \frac{-b}{M_1} & \frac{k_2}{M_1} & \frac{b}{M_1} \\ 0 & 0 & 0 & 1 \\ \frac{k_2}{M_2} & \frac{b}{M_2} & \frac{-k_2}{M_2} & \frac{-b}{M_2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{k_1}{M_1} \\ 0 \\ 0 \end{pmatrix} u,$$

$$y = (0 \quad 0 \quad 1 \quad 0) \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + (0)\, u. \tag{3.12}$$

3.4 Estudio de choques

Puede ocurrir que debido a una gran perturbación del terreno se alcance uno de los extremos del recorrido, produciéndose así un choque frontal entre la masa no suspendida y el chasis. En ese instante se produce una no linealidad del comportamiento dejando de tener sentido físico las ecuaciones diferenciales lineales (3.9) y (3.10). En ese momento, entran en juego unas nuevas ecuaciones que describen la colisión y que determinan la velocidad que ambos cuerpos toman en el instante posterior al impacto.

Se define u_i y v_i como la velocidad antes y después del choque de la masa i = 1, 2, respectivamente. Además se introduce el coeficiente de restitución c_r que representa, en tanto por uno, la cantidad de energía cinética que se conserva en el impacto:

$$c_r = -\frac{v_1 - v_2}{u_1 - u_2} \tag{3.13}$$

Si el coeficiente de restitución es uno, la velocidad relativa al inicio del choque será la misma que después de haberse producido, esto es, no se ha producido pérdida alguna de energía cinética.

Por el contrario, si el coeficiente toma un valor nulo, en el choque se disipará toda la energía cinética en forma de calor y las velocidades posteriores al impacto de ambos cuerpos (v_1 y v_2) serán idénticas en dirección y sentido (permanecerán unidos).

Por lo tanto, se puede plantear lo siguiente:

$$M_1 u_1 + M_2 u_2 = M_1 v_1 + M_2 v_2 \tag{3.14}$$

Introduciendo la expresión (3.13) en (3.14) y despejando las velocidades posteriores al impacto se obtienen las expresiones:

$$\begin{aligned} v_1 &= \frac{(M_1 - M_2 c_r) u_1 + M_2 (1 + c_r) u_2}{M_1 + M_2} \\ v_2 &= \frac{M_1 (1 + c_r) u_1 + (M_2 - M_1 c_r) u_2}{M_1 + M_2} \end{aligned} \tag{3.15}$$

Estas expresiones se pueden simplificar definiendo la velocidad del centro de masas del sistema como:

$$V_{\text{cm}} = \frac{M_1 u_1 + M_2 u_2}{M_1 + M_2} \tag{3.16}$$

De esta forma, llevando (3.16) a (3.15) se obtienen relaciones simplificadas para las velocidades después del choque.

$$\begin{aligned} v_1 &= (1 + c_r) V_{\text{cm}} - c_r u_1 \\ v_2 &= (1 + c_r) V_{\text{cm}} - c_r u_2 \end{aligned} \tag{3.17}$$

Éstas serán las velocidades que tendrán ambos cuerpos después del impacto, momento en el que volverán a tener utilidad las ecuaciones diferenciales lineales (3.9) y (3.10), por consiguiente las ecuaciones de estado (3.11), con condiciones iniciales dadas por las velocidades obtenidas en (3.17).

Capítulo 4.
Identificación de parámetros

4.1 Introducción

En este capítulo se exponen los procedimientos seguidos para la obtención de los parámetros que componen el modelo de un cuarto de vehículo, expuesto en el Capítulo 3, para cada una de las cuatro suspensiones del robot Andábata.

Los parámetros que se determinarán son: la constante elástica de los neumáticos (k_1), la constante elástica de los resortes (k_2) y el coeficiente de fricción (b). Una vez determinados estos parámetros, se obtendrá el valor del coeficiente de restitución (c_r) y el tiempo en el que se produce el impacto ($t_{impacto}$). La obtención de estos parámetros se ha realizado mediante simulación.

4.2 Circuito de instrumentación

El objetivo de un sistema de medición es presentar a un observador un valor numérico correspondiente a la variable que se capta. En general, este valor numérico no es igual al valor verdadero de la variable. La *Figura 4.1* ilustra un sistema de medición básico.

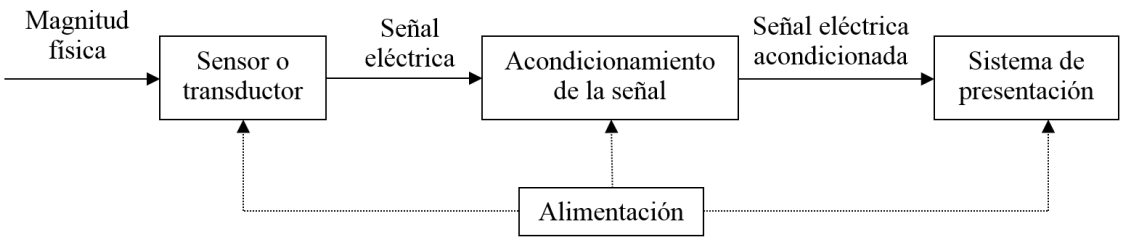


Figura 4.1. Estructura general de un sistema de medición básico

El sensor o transductor es el dispositivo que transforma la magnitud física en una magnitud eléctrica. Para realizar este cambio, en ocasiones se requiere un sensor

alimentado (sensor pasivo o modulador) y en otras ocasiones no es necesario (sensor activo o generador) [12].

En este apartado, se expone cómo se han realizado las conexiones para registrar el desplazamiento vertical de la masa suspendida (magnitud física a medir) mediante el uso de un transductor pasivo.

4.2.1 Componentes

Para realizar la toma de datos se han utilizado los siguientes componentes:

- Potenciómetro lineal de 6 *cm* de recorrido, resistencia máxima de 10 *kΩ* y 12 *W* de disipación máxima recomendada (ver *Figura 4.2*).

- Pila de 9 *V* de corriente continua (CC).

- Osciloscopio con salida USB modelo *Textronix TDS 220* cuya discretización temporal es de 1 *ms* (ver *Figura 4.3*).

- Cables.

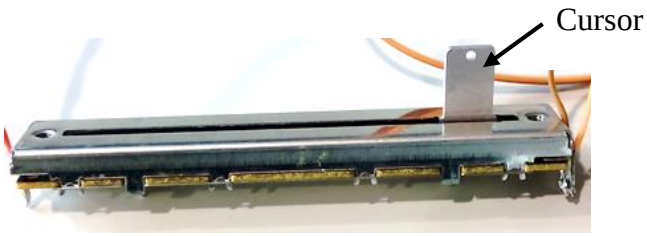


Figura 4.2. Potenciómetro lineal.

Figura 4.3. Osciloscopio Textronix TDS 220.

4.2.2 Montaje

Se trata de un montaje ex profeso para este TFG que no va a quedar instalado de forma permanente en el robot. En primer lugar, se fijó el potenciómetro al chasis mediante dos piezas metálicas atornilladas a la masa no suspendida. Al formar un triángulo se tiene

la certeza de que no existe holgura durante el movimiento (ver *Figura 4.4*). Se ha procurado que el cursor del potenciómetro llegue a uno de sus extremos cuando la suspensión también lo haga, aprovechando así el máximo de resolución en las medidas.

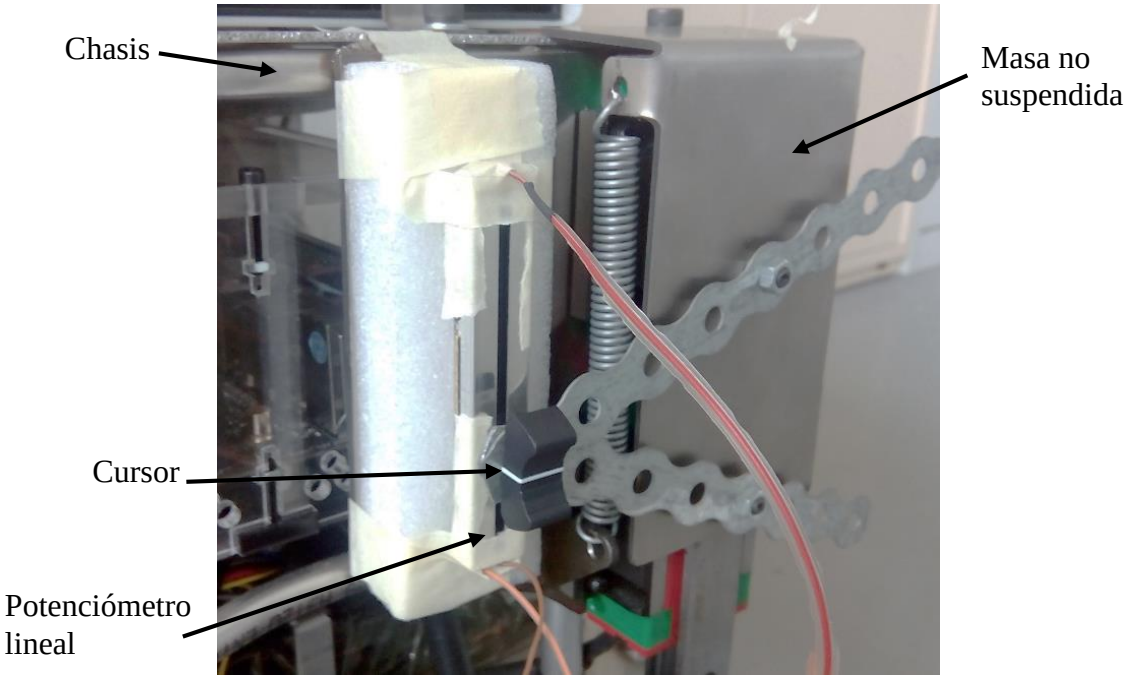


Figura 4.4. Fijación del potenciómetro al chasis.

En segundo lugar, se realizó el esquema de conexión de la *Figura 4.5* donde:

- V_e: Representa la tensión de 9 *V* de la pila de *CC*.
- *L*: Recorrido del potenciómetro lineal (6 *cm*).
- *x*: Posición del cursor.
- *R(x)*: Resistencia del potenciómetro cuando el cursor se encuentra en *x*.
- R_p: Resistencia total del potenciómetro (10 *kΩ*).
- $V_s\,(x)$: Tensión de salida leída por la sonda del osciloscopio.

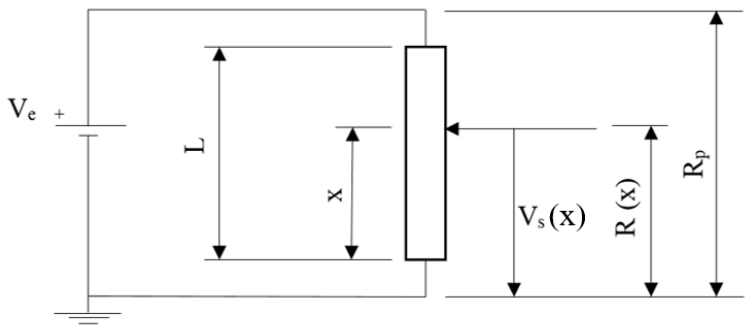


Figura 4.5. Esquema de conexión.

4.2.3 Conversión de tensión a desplazamiento

A partir de la *Figura 4.5* se deduce que la resistencia entre el contacto móvil y uno de los elementos de los terminales fijos tiene que verse afectada por un factor proporcional $\propto$:

$$R\,(x) = R_p \propto \tag{4.1}$$

$$\propto = \frac{x}{L} \qquad 0 \leq \propto \leq 1$$

Por tanto, conociendo el valor de la variación de la resistencia es posible determinar el valor del desplazamiento.

Aplicando la Ley de Ohm a la relación (4.1) y teniendo en cuenta que la intensidad circulante es común, se tiene la siguiente igualdad:

$$x = \frac{V_s(x)\,L}{V_e} \tag{4.2}$$

En la *Figura 4.6* se aprecia el aspecto final del circuito de instrumentación sobre una de las suspensiones de Andábata.

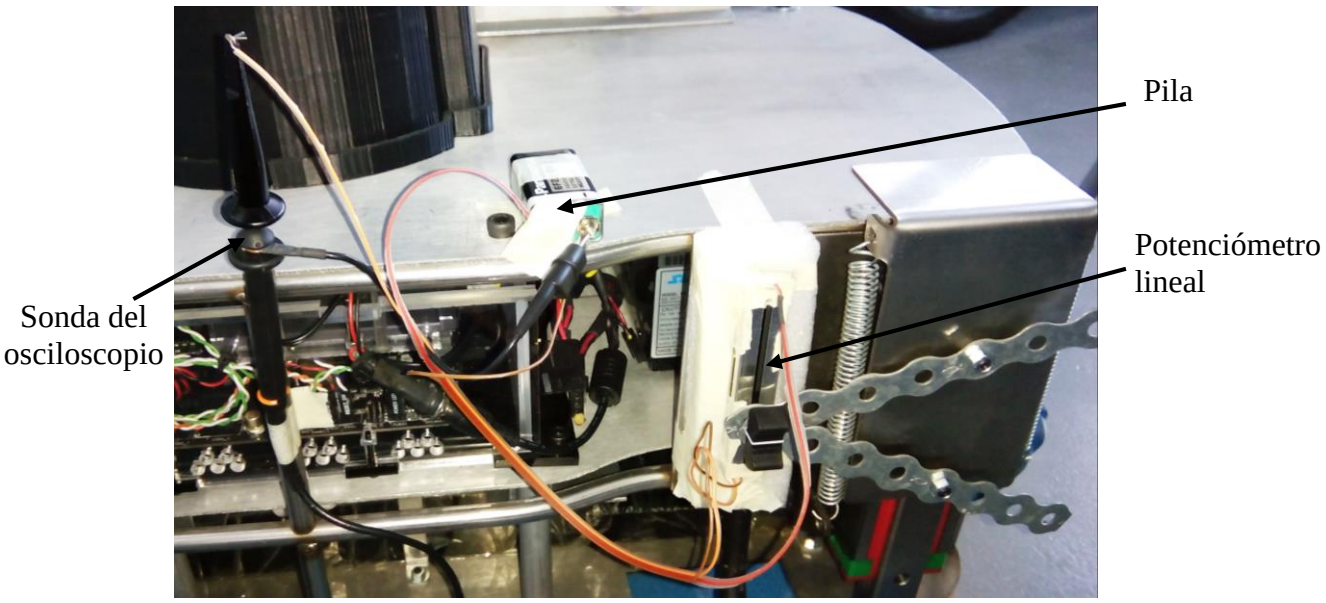


Figura 4.6. Circuito de instrumentación sobre Andábata.

Una vez obtenidos los datos de la suspensión de una rueda, el circuito se desmonta y se coloca sobre la siguiente rueda que se desea medir medir.

4.3 Estimación de parámetros mediante simulación

La idea básica consiste en determinar, mediante simulación, los errores de aplicación de los parámetros del modelo propuesto para posibles parámetros θ en relación a los experimentos previamente registrados sobre el proceso real (ver *Figura 4.7*).

Se pueden emplear diferentes criterios de evaluación de la función de coste:

$$J(\theta) = \begin{cases} \sum |e_i| \\ \sum e_i{}^2 \\ max|e_i| \end{cases} \tag{4.3}$$

Donde e_i representa el error en el instante i entre los datos registrados y simulados. Para la optimización de $J(\theta)$ se ha empleado el método *Simplex.*

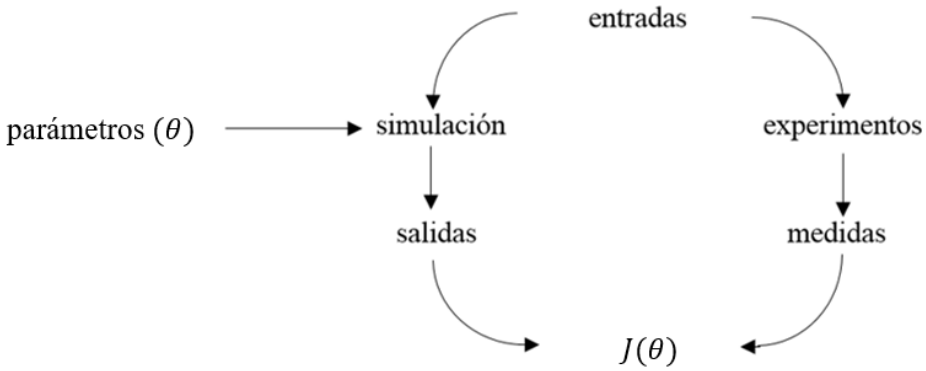


Figura 4.7. Comparación del modelo con el proceso [7].

4.3.1 El método *Simplex*

El método *Simplex* es una técnica de optimización por exploración directa. Se basa en el concepto geométrico del simplex [9], que es una colección de $a + 1$ puntos o vértices en un espacio de búsqueda de dimensión $a > 1$. Es un método iterativo que permite ir mejorando la solución en cada paso. Se dirá que el simplex no está degenerado si encierra un subespacio de dimensión a (ver *Figura 4.8*).

A grandes rasgos, el algoritmo busca los parámetros θ, los cuales son los vértices del simplex, que minimizan el valor de la función de coste $J(\theta)$. Para ello, parte de una estimación inicial θ^0 y transforma el simplex en cada iteración. Se dará por acabada la búsqueda cuando se supere un número máximo de iteraciones, la distancia entre los puntos del simplex esté por debajo de un determinado umbral o la máxima diferencia de la función objetivo entre todos los vértices sea menor que un cierto valor [9].

En principio, es adecuado para problemas de optimización multivariable y no requiere cálculo de derivadas.

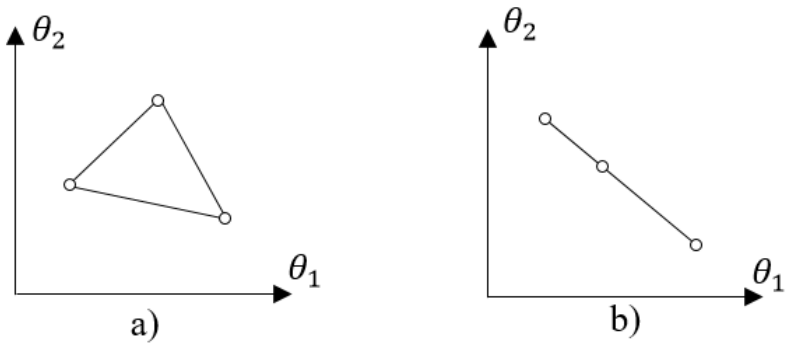


Figura 4.8. Simplex normal a) y degenerado b) en un plano de búsqueda (a = 2).

4.3.2 La función *fminsearch* de Matlab

Esta función implementa el método *Simplex* en Matlab. Su sintaxis es la siguiente:

[x,fval,exitflag,output] = fminsearch(fun,x0,options)

Los argumentos de la función son:

fun: función objetivo a optimizar. Se define normalmente en un archivo .m.

x0: valor inicial de θ.

options: opciones de optimización como, por ejemplo, el número máximo de evaluaciones de la función objetivo. Su valor por defecto es doscientas veces el número de variables a optimizar.

Los parámetros de la salida son:

x: mínimo local de *fun*.

fval: devuelve el valor de la función objetivo *fun* en la solución *x*.

exitflag: describe la condición de salida de *fminsearch*.

output: devuelve una estructura output que contiene información diversa sobre la optimización efectuada.

En esencia, *fminsearch* encuentra el valor de las variables *x* que minimizan la función descrita en *fun,* comenzando por el valor inicial especificado en *x0*.

Los códigos Matlab de optimización en los que se ha hecho uso de esta función se proporcionan en el Anexo A de la presente memoria.

4.4 Parámetros de la suspensión (k_1, k_2, b)

4.4.1 Experimento realizado

El experimento consiste en hacer evolucionar el sistema desde una posición inicial hasta el reposo. Para ello se desplazó el chasis verticalmente hacia abajo hasta una posición inicial tal que, después de ser liberado, no alcanzase los límites del recorrido durante las oscilaciones. Al no existir discontinuidades en la respuesta, se puede hacer uso de la propiedad de linealidad del sistema para normalizar la respuesta.

El ensayo se repitió cuatro veces, una por cada rueda, debido a que solo se usó un circuito de medida.

4.4.2 Valores iniciales

Es posible que el algoritmo *Simplex* llegue a errar en su búsqueda, quedándose estancado en un mínimo local. Por ello, se ha realizado un ensayo para encontrar un punto de partida aproximado para optimizar el parámetro k_2. Se trata de un simple experimento basado en la ley de Hook:

Se somete a Andábata a dos situaciones de carga: una con su peso propio ($M_{Total\ chasis}$) y otra con una carga adicional de 26,39 *kg* (M_c). Para cada uno de los casos se miden los desplazamientos de las suspensiones.

Los desplazamientos para el caso del peso propio se observan en la *Tabla 4.1*.

Suspensión	Desplazamiento (*mm*)
1	14
2	15
3	16
4	13
Media (δ_1)	**14,5**

Tabla 4.1. Desplazamiento de las suspensiones $M_{Total\ chasis}$.

Los desplazamientos de las suspensiones cuando Andábata soporta la carga M_c se exponen en la *Tabla 4.2*.

Suspensión	Desplazamiento (*mm*)
1	43
2	43
3	43
4	30
Media (δ_2)	**42**

Tabla 4.2. Desplazamiento de las suspensiones con M_c.

Aplicando la ley de Hook y teniendo en cuenta que la carga se reparte entre las cuatro ruedas se tiene lo siguiente:

$$k_2^0 = \frac{g(M_{Total\ chasis} + M_c) - gM_{Total\ chasis}}{4 \cdot (\delta_2 - \delta_1)} = \frac{M_c g}{4\ (\delta_2 - \delta_1)} = 2332{,}3\ N/m \qquad (4.4)$$

Por su parte, la rueda es muy rígida por lo que el coeficiente elástico (k_1^0) se ha elegido elevado. Por último, para la fricción viscosa (b^0) se ha tomado un valor pequeño.

Así pues, los valores iniciales usados para comenzar las iteraciones del *Simplex* son:

$$k_1^0 = 22800\ N/m.$$
$$k_2^0 = 2332{,}3\ N/m. \qquad (4.5)$$
$$b^0 = 20\ N{\cdot}s/m.$$

4.4.3 Resultados obtenidos

Partiendo de las lecturas tomadas por el circuito de instrumentación, se busca reducir una de las funciones de coste expuestas en (4.3). Siendo e_i el error existente, en el instante *i*, entre la respuesta real y la modelada. En este caso, $\theta = (k1, k2, b)$.

Los parámetros obtenidos a través del método de optimización *Simplex* se resumen en las *Tablas 4.3, 4.4, 4.5* para las cuatro suspensiones con distintas funciones de coste.

Las *Figuras 4.9, 4.10, 4.11* muestran las respuestas del modelo optimizado según las distintas funciones de coste y del sistema real para todas las suspensiones de Andábata.

Suspensión	k_1 (N/m)	k_2 (N/m)	b (N·s/m)
1	$8{,}9185 \times 10^{12}$	3780,3	93,5524
2	$2{,}8071 \times 10^{12}$	3875,3	72,2650
3	$1{,}1887 \times 10^{12}$	3649,7	76,9565
4	$9{,}4448 \times 10^{11}$	4018,7	76,4721
Valor medio	**$3{,}4647 \times 10^{12}$**	**3831**	**79,8115**

Tabla 4.3. Parámetros óptimos de las suspensiones según la función de coste $\sum |e_i|$.

Suspensión	k_1 (N/m)	k_2 (N/m)	b (N·s/m)
1	$5{,}8442 \times 10^{13}$	3755,3	75,2045
2	$5{,}8529 \times 10^{13}$	3900,9	73,5224
3	$4{,}9510 \times 10^{13}$	3830,1	77,5951
4	$4{,}0215 \times 10^{13}$	3744,0	69,3376
Valor medio	**$5{,}1674 \times 10^{13}$**	**3807,6**	**73,9149**

Tabla 4.4. Parámetros óptimos de las suspensiones según la función de coste $\sum e_i^2$.

Suspensión	k_1 (N/m)	k_2 (N/m)	b (N·s/m)
1	$1{,}4498 \times 10^{5}$	4186,8	82,9478
2	$2{,}3524 \times 10^{5}$	4322,8	67,5031
3	$9{,}5862 \times 10^{4}$	4578,9	82,7364
4	$4{,}2109 \times 10^{5}$	4076,1	73,0047
Valor medio	**$2{,}2429 \times 10^{5}$**	**4291,1**	**76,5480**

Tabla 4.5. Parámetros óptimos de las suspensiones según la función de coste $max|e_i|$.

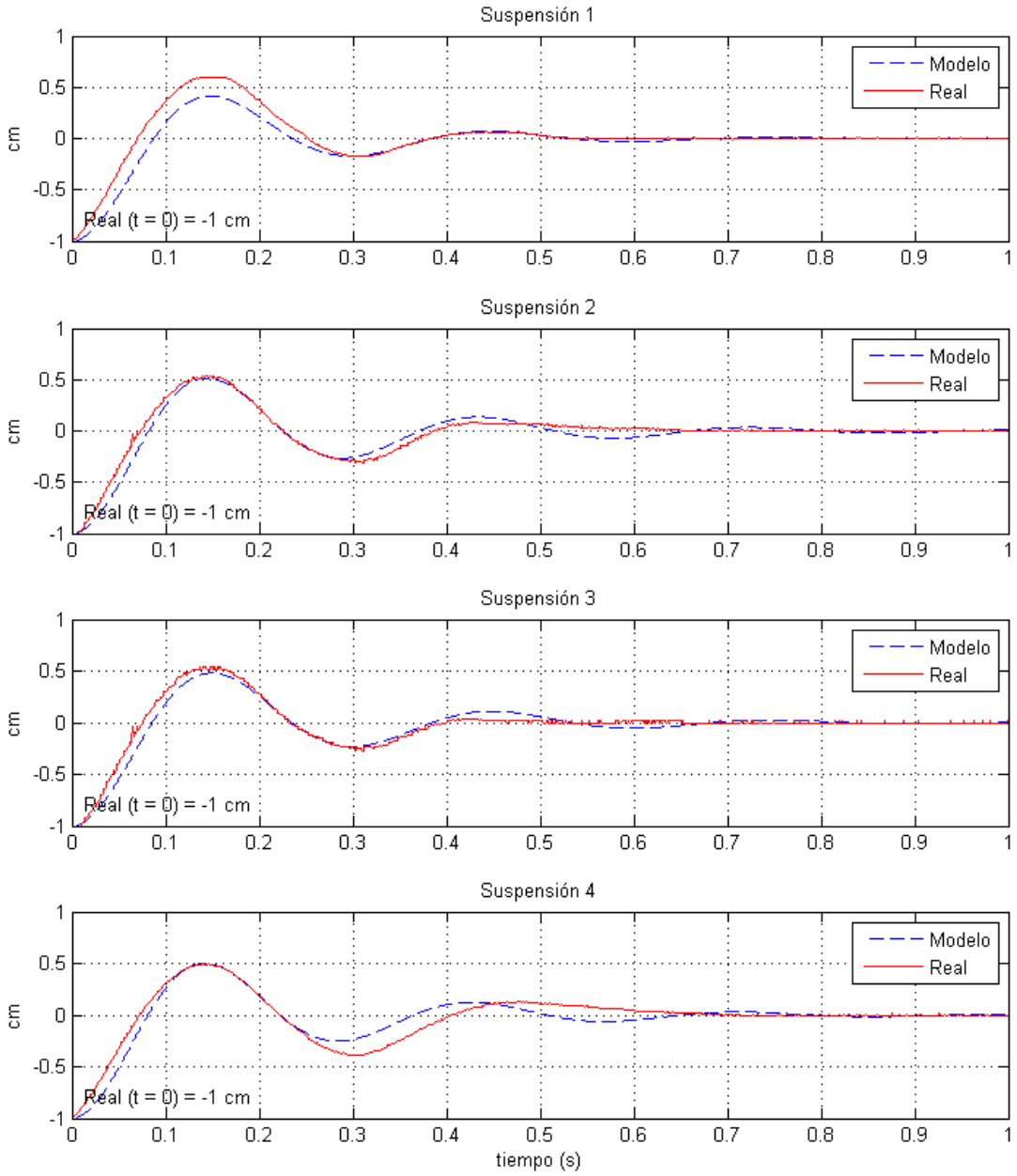


Figura 4.9. Comparación de respuestas modelo-real normalizadas con parámetros óptimos según función de coste $\sum |e_i|$.

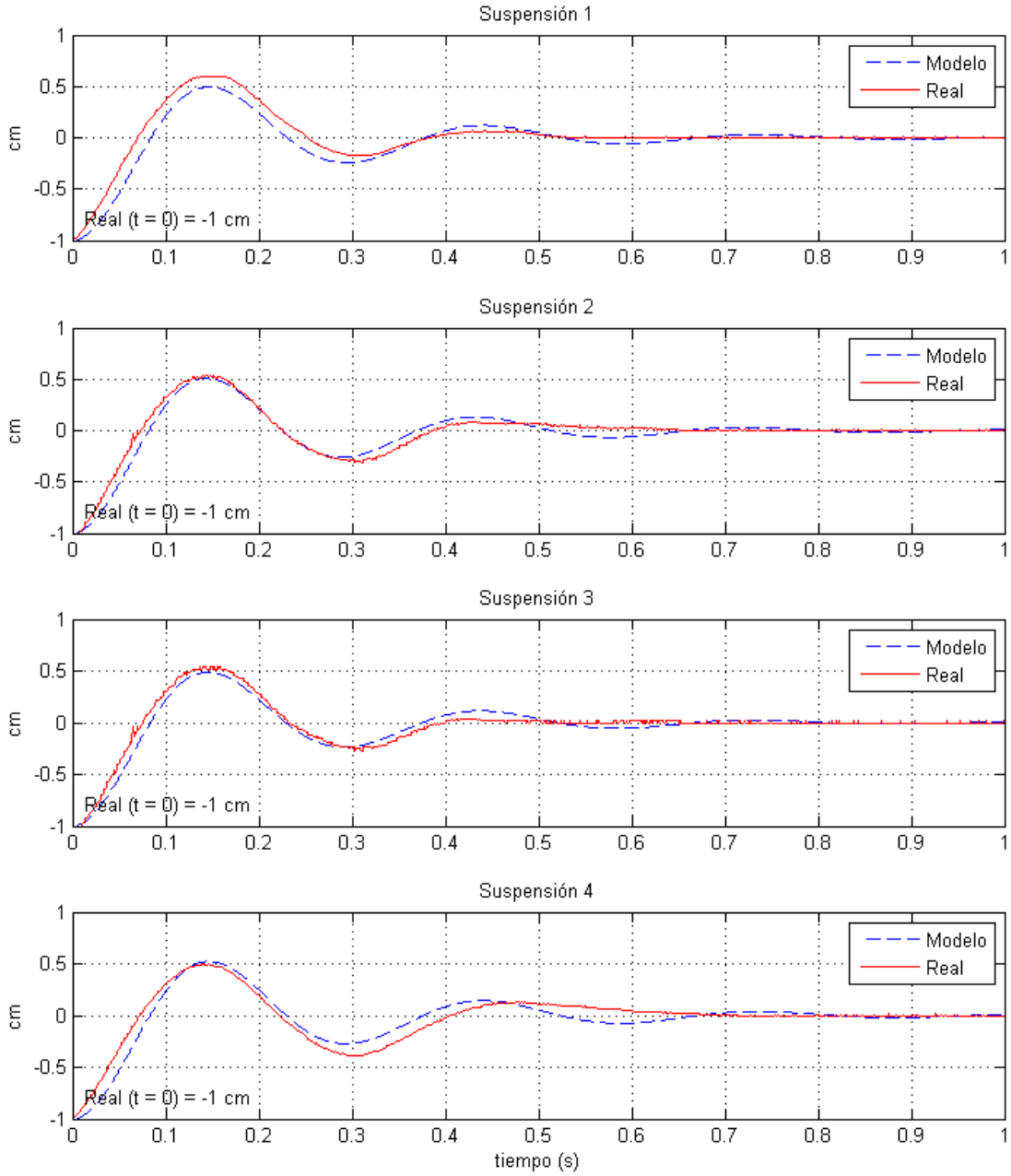


Figura 4.10. Comparación de respuestas modelo-real normalizadas con parámetros óptimos según función de coste $\sum e_i{}^2$.

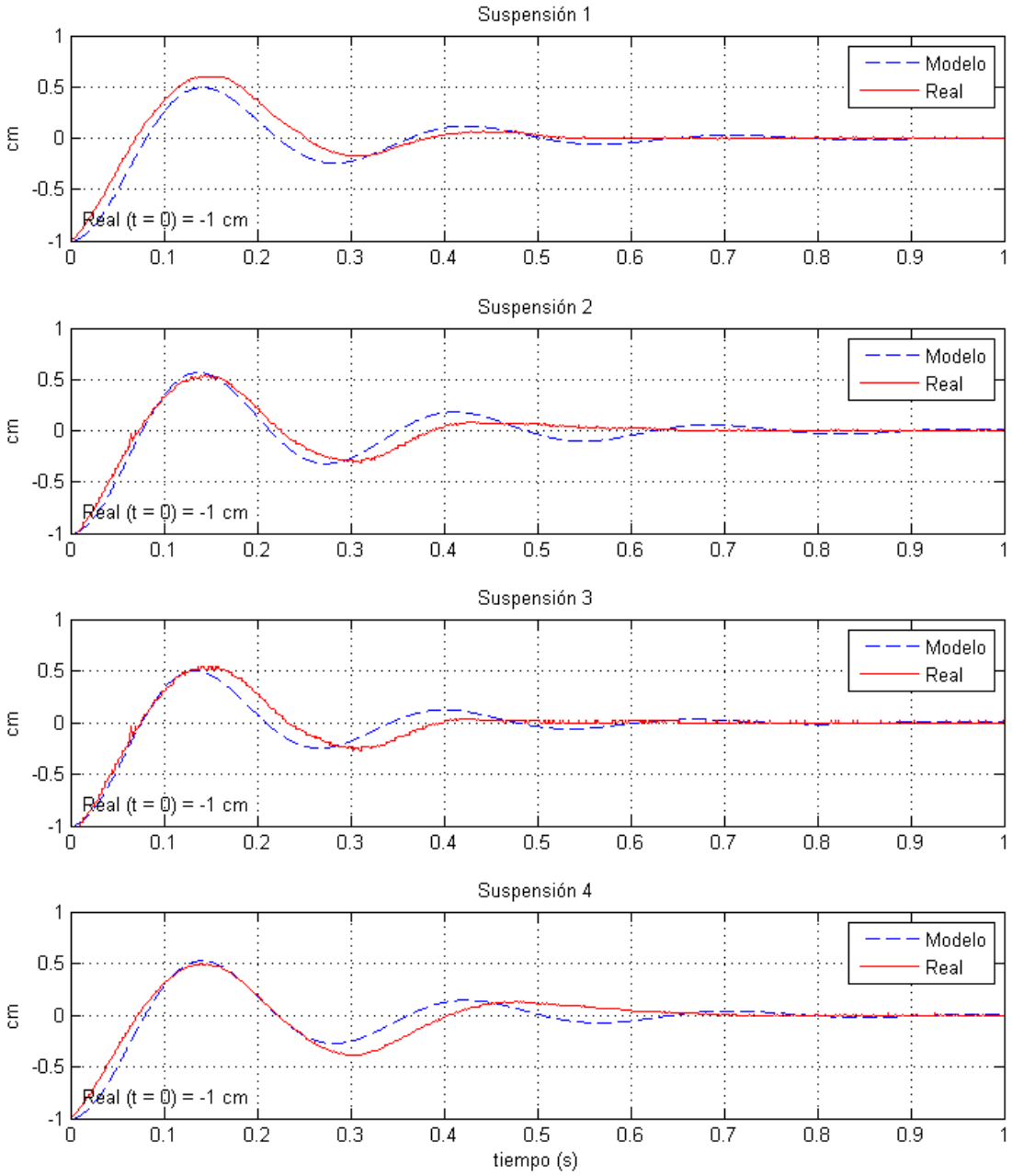


Figura 4.11. Comparación de respuestas modelo-real normalizadas con parámetros óptimos según función de coste $max|e_i|$.

4.4.4 Observaciones

En todos los casos, las suspensiones 2 y 3 son las que mejor se ajustan al comportamiento de un cuarto de vehículo. Por otro lado, las suspensiones 1 y 4 presentan diferencias notables en el tiempo de subida y establecimiento respectivamente.

Respecto de las distintas funciones de coste, se observa una mayor variación del parámetro k_1 pero siempre con valores muy altos.

Los valores medios entre los parámetros de las cuatro suspensiones dados por las distintas funciones de coste son:

$$k_1 = 1{,}8380\ x\ 10^{13}\ \mathrm{N/m}, \qquad k_2 = 3976{,}6\ \mathrm{N/m}, \qquad b = 75{,}7581\ \mathrm{N \cdot s/m}.$$

4.5 Coeficiente de restitución (c_r)

4.5.1 Experimento realizado

En este caso, el experimento es similar al ejecutado para determinar los parámetros k_1, k_2 y b pero la posición inicial cambia. Se ha seleccionado una posición inicial de recuperación lo suficientemente grande como para que se alcance un extremo del recorrido. De esta forma, se produce un choque que será registrado por el circuito de instrumentación. Una vez más el experimento se repitió cuatro veces, una por cada rueda.

4.5.2 Valores iniciales

Los parámetros que se han usado para optimizar el modelo de choque son el tiempo en el que éste se produce ($t_{impacto}$) y el coeficiente de restitución (c_r). El tiempo de impacto para comenzar las iteraciones de optimización *Simplex* se ha determinado visualmente de las respuestas registradas por el circuito de medida. Por su parte, el valor inicial para el coeficiente de restitución se ha tomado ligeramente elevado, ya que la masa suspendida, en la respuesta real, permanece poco tiempo en contacto con la masa no suspendida durante el impacto. Por lo tanto, los valores iniciales seleccionados son:

$$t_{impacto} = 0{,}1\ s \qquad c_r = 0{,}7 \tag{4.6}$$

4.5.3 Resultados obtenidos

Los parámetros óptimos de choque se han obtenido también mediante simulación, tomando como conocidos los valores $k1, k2$ y b obtenidos por las mismas funciones de coste en cada caso. Los resultados de la optimización para $\theta = (t_{impacto}, c_r)$ usando distintas funciones de coste se muestran en las *Tablas 4.6, 4.7, 4.8.*

Suspensión	$t_{impacto}$ (s)	c_r
1	0,118	0,455
2	0,114	0,756
3	0,120	0,460
4	0,119	0,781
Valor medio	**0,118**	**0,613**

Tabla 4.6. Parámetros óptimos de choque según funciones de coste $\sum |e_i|$.

Suspensión	$t_{impacto}$ (s)	c_r
1	0,115	0,508
2	0,117	0,604
3	0,108	0,630
4	0,115	0,760
Valor medio	**0,114**	**0,626**

Tabla 4.7. Parámetros óptimos de choque según funciones de coste $\sum e_i{}^2$.

Suspensión	$t_{impacto}$ (s)	c_r
1	0,124	0,525
2	0,117	0,538
3	0,116	0,555
4	0,117	0,563
Valor medio	**0,119**	**0,545**

Tabla 4.8. Parámetros óptimos de choque según funciones de coste $max|e_i|$.

Las *Figuras 4.12, 4.13, 4.14* ilustran la comparación entre la respuesta del modelo de choque optimizado a partir de las distintas funciones de coste.

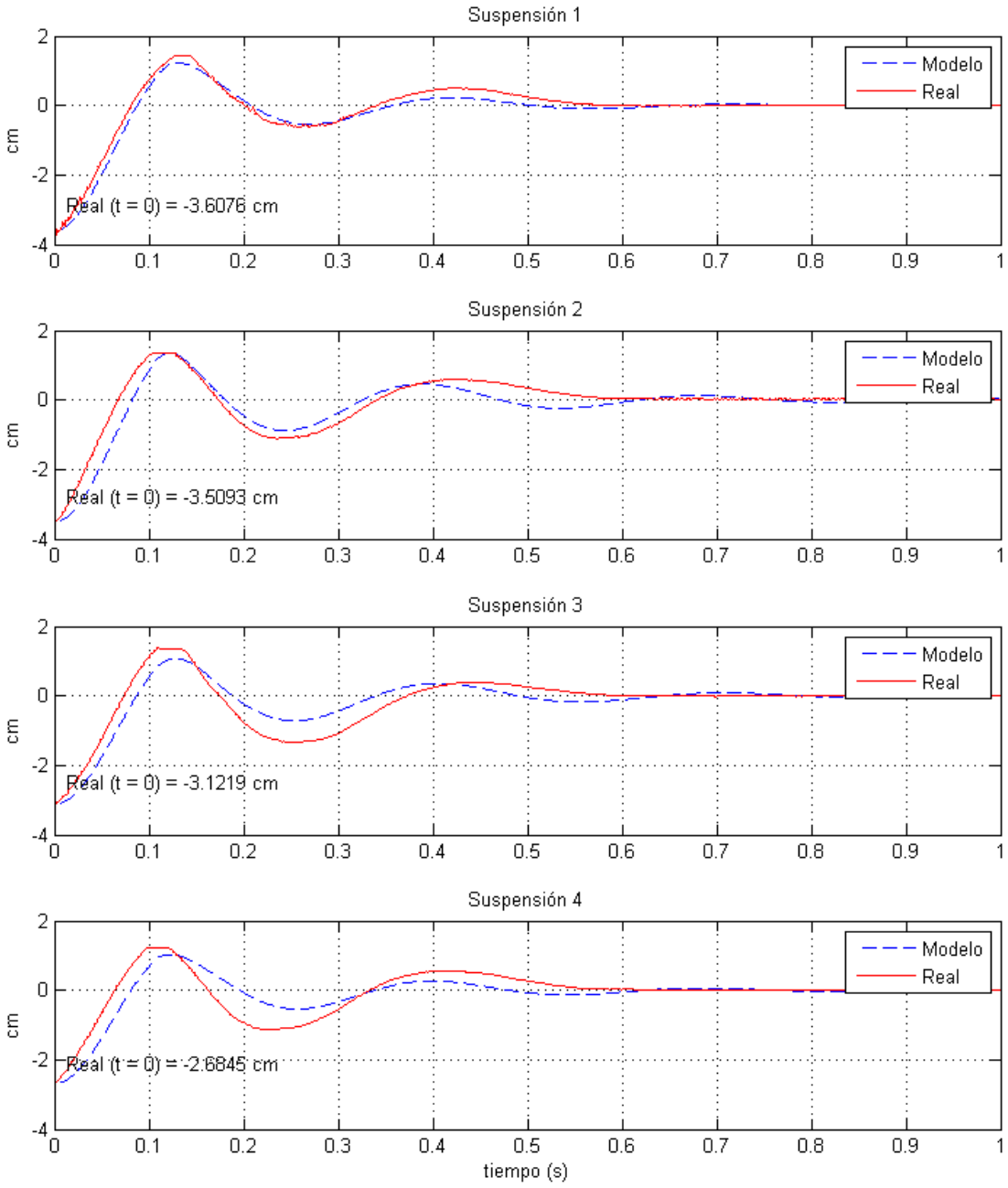


Figura 4.12.Comparación de respuestas modelo-real con parámetros de choque óptimos según función de coste $\sum |e_i|$.

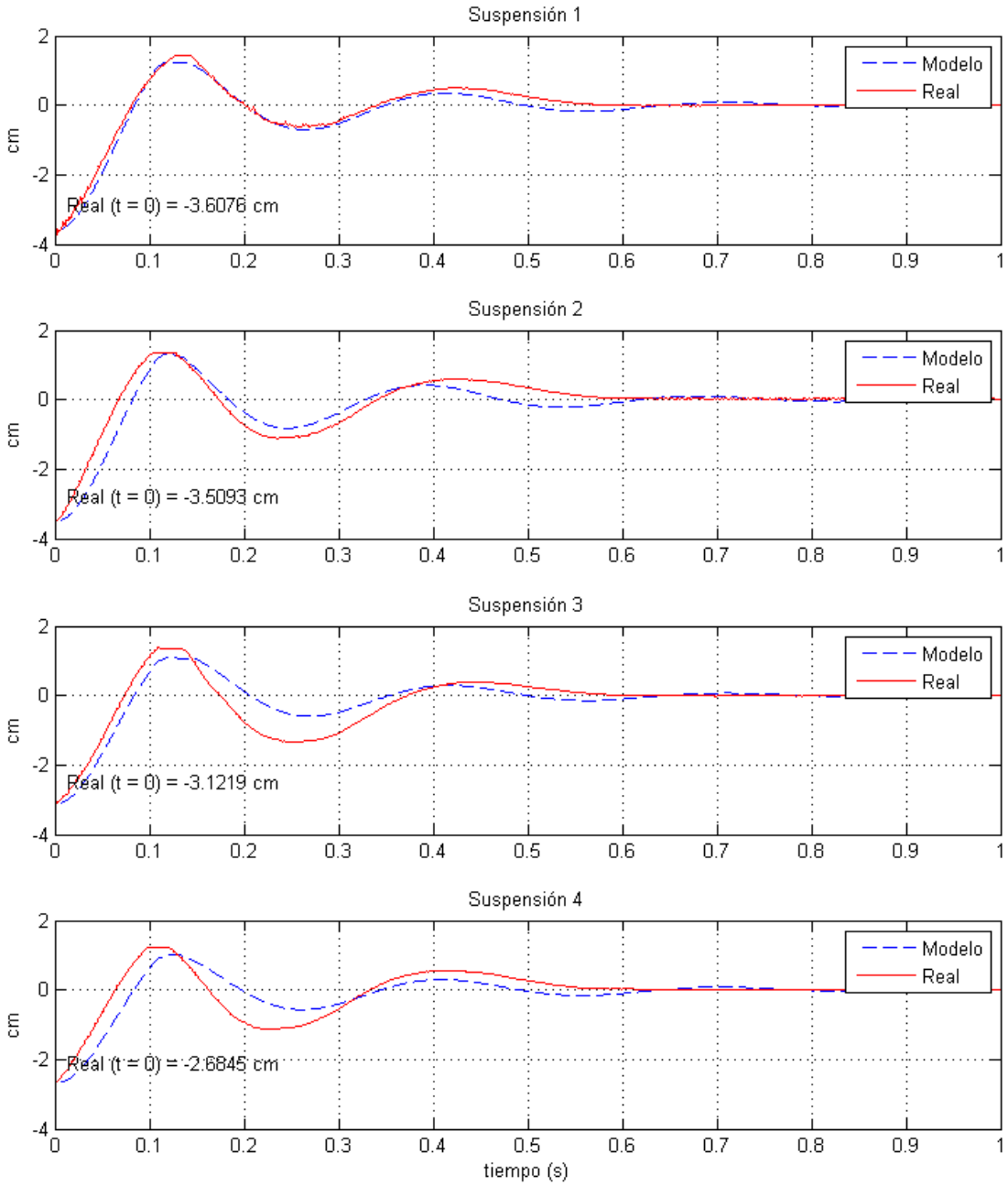


Figura 4.13. Comparación de respuestas modelo-real con parámetros de choque óptimos según función de coste $\sum e_i^2$.

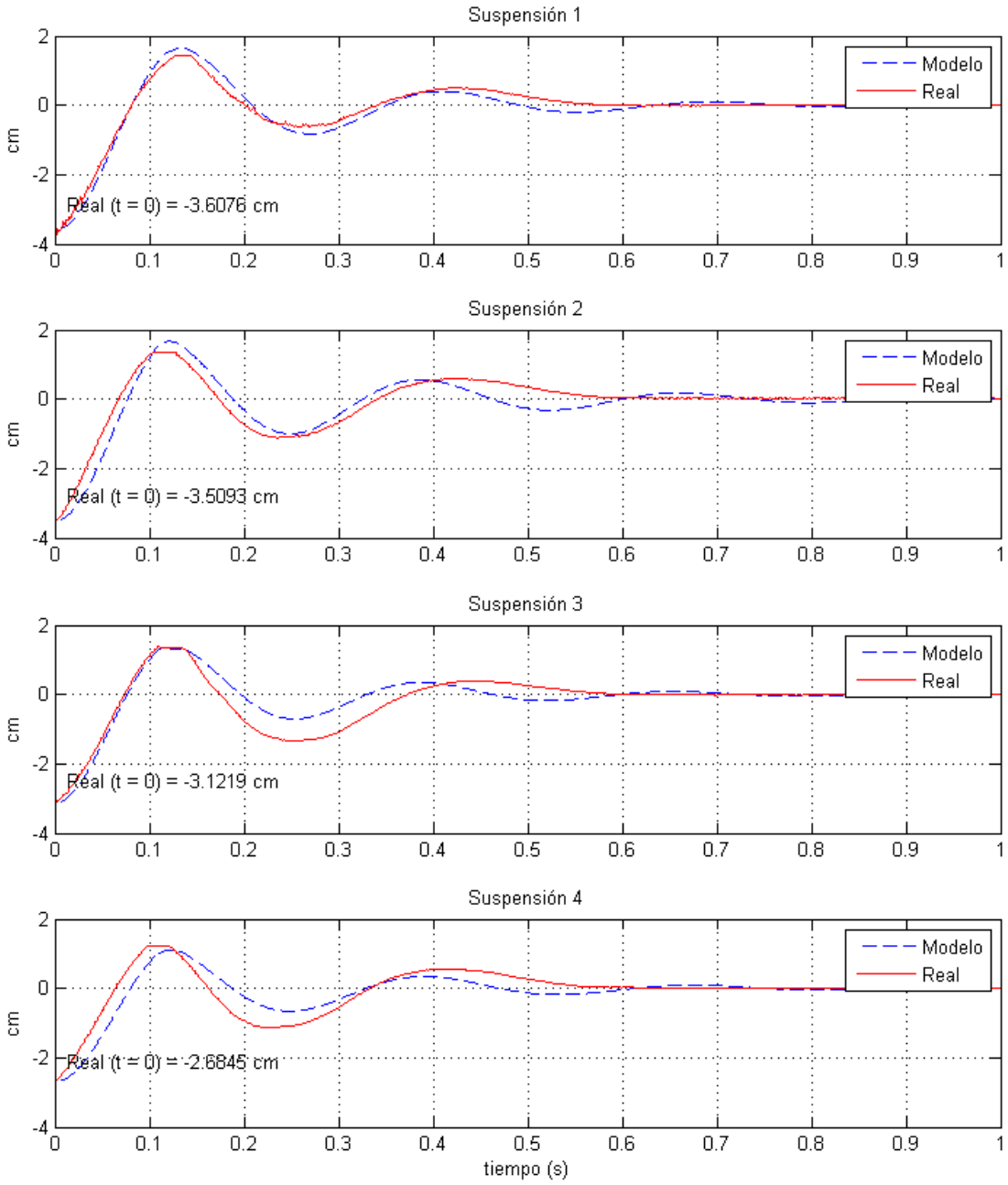


Figura 4.14. Comparación de respuestas modelo-real con parámetros de choque óptimos según función de coste $max|e_i|$.

4.5.4 Observaciones

Una vez que el choque se produce, las discrepancias entre el modelo y la respuesta real son más notorias. La explicación para este fenómeno es que el modelo empleado considera masas aisladas. Se entiende por esto que el modelado que se ha usado no tiene en cuenta la influencia que ejerce el movimiento de una suspensión sobre las demás.

Por otro lado, la masa suspendida no choca a la misma altura en todos los casos. Según la *Tabla 2.2*, la cuarta suspensión tiene un recorrido menor y, por ello, es la primera en impactar. Este primer impacto influye sobre el comportamiento de las demás suspensiones, desequilibrando el chasis. Este fenómeno no está considerado en el modelo. Aún así se obtienen resultados bastante aceptables, con un coeficiente de restitución medio c_r = 0,595.

Capítulo 5.
La aplicación Matlab *mcvAndábata*

5.1 Introducción

La finalidad de la aplicación *mcvAndábata* es ofrecer una interfaz de uso sencillo para el usuario, permitiendo modificar de manera cómoda los distintos parámetros del modelo de un cuarto de vehículo y visualizar rápidamente su respuesta ante distintas entradas.

Para ello, *mcvAndábata* cuenta con varias interfaces de usuario. Estas interfaces muestran los resultados de las variables de estado y además, si se desea, permiten visualizar una animación mientras se está ejecutando. También es capaz de mostrar los resultados en frecuencia de las masas del chasis y de la rueda mediante diagramas de Bode.

5.2 Interfaces de usuario

La aplicación *mcvAndábata* está formada por siete interfaces: interfaz principal, escalón, rampa, seno, condiciones iniciales, resultados avanzados y exportar. Todas han sido programadas mediante el asistente de *Graphical User Interface* (GUI) de Matlab [8]. A continuación se comenta el cometido de cada una de ellas.

5.2.1 Interfaz principal (GUI.m)

Es la primera que se lanza y la responsable de mostrar al usuario información generalista sobre el modelo de un cuarto de vehículo y notación relacionada. Además, en ella se pueden seleccionar los datos de partida de la simulación. Para ello cuenta con tres submenús (ver *Figura 5.1*) que se explican en los siguientes subapartados. Una vez realizada la simulación, la interfaz muestra una animación y los resultados obtenidos (ver *Figura 5.2*).

Figura 1. *Modelo esquemático de un cuarto de vehículo.*

Figura 5.1. Interfaz gráfica principal (GUI.m).

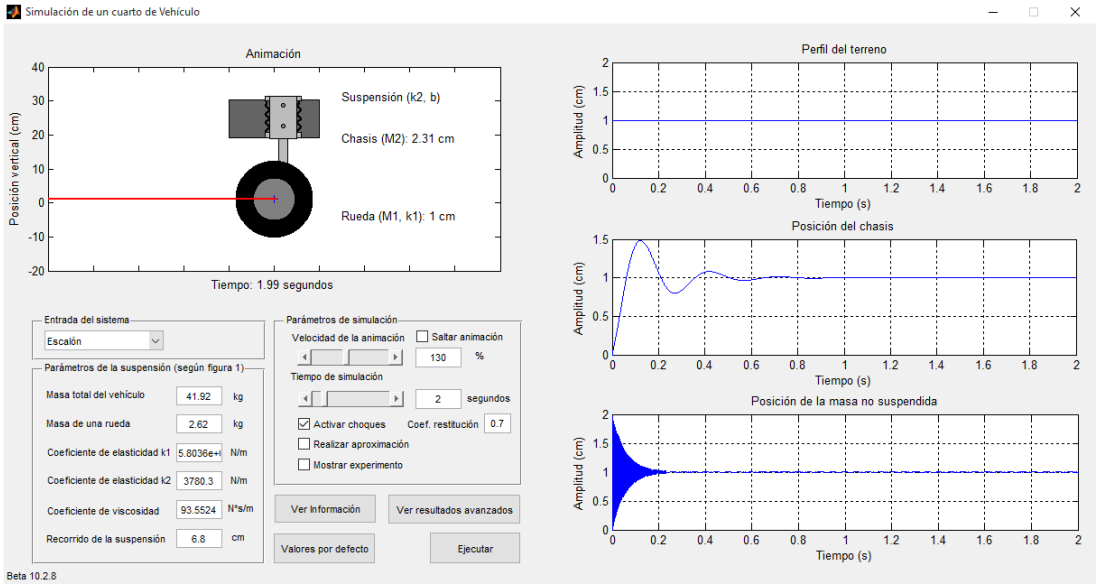


Figura 5.2. Resultados en la interfaz gráfica principal (GUI.m).

5.2.1.1 Entrada del sistema

Mediante un menú desplegable se pueden seleccionar tres posibles entradas (z_r) para el sistema o bien establecer las condiciones iniciales (x_0) de la simulación. Incluso si se desea se pueden combinar ambas opciones, de manera que es posible establecer unas condiciones iniciales y una entrada tipo escalón al mismo tiempo, por ejemplo. Por otro lado, no es posible combinar varias entradas tipo (z_r). Las entradas (z_r) disponibles son: escalón, rampa y sinusoide.

El aspecto de sus interfaces se detalla en el apartado 5.2.2.

5.2.1.2 Parámetros de la suspensión

En este submenú se introducen todos los parámetros necesarios para definir el modelo de un cuarto de vehículo. Estos parámetros son:

- Masa total del vehículo: Internamente la aplicación calcula el valor de la masa suspendida en *kg* (M_2).
- Masa de una rueda: se corresponde con la masa no suspendida en *kg* (M_1).
- Coeficiente de elasticidad k_1: representa la rigidez del neumático en *N/m*.
- Coeficiente de elasticidad k_2: representa la rigidez de los resortes en *N/m*.
- Coeficiente de viscosidad *b*: modela la fricción del patín en *N·s/m*.
- Recorrido de la suspensión: se introduce en *cm*.

5.2.1.3 Parámetros de simulación

En este apartado se seleccionan aspectos relacionados con la simulación tales como velocidad de la animación o tiempo final de cálculo.

Si se desea, se pueden mostrar directamente los resultados finales sin realizar la animación marcando la casilla “Saltar animación”.

Al marcar la casilla “Activar choques”, se tendrán en cuenta los límites del recorrido de la suspensión y será necesario establecer un coeficiente de restitución.

Si se activa “Realizar aproximación”, la entrada al sistema comenzará en el instante dado por el 10% del tiempo final de simulación en lugar de empezar en el instante 0.

Si se selecciona la casilla “Mostrar experimento” se le dará al usuario la opción de elegir una de las cuatro suspensiones de Andábata. Al hacer esto, aparecerá en los resultados finales la respuesta real registrada correspondiente a la suspensión seleccionada en el menú desplegable. Si la casilla “Activar choques” está seleccionada, los experimentos que se mostrarán serán los de impacto. En el caso de que la casilla esté desmarcada, las respuestas reales sin choque serán las que se visualizarán.

5.2.2 Interfaces de entrada al sistema (Escalon.m, Rampa.m, Seno.m, Condiciones.m)

A continuación se comentan las diferentes interfaces de entrada al sistema. Sus aspectos se muestran en la *Figura 5.3*.

5.2.2.1 Escalón

Mediante un *slider* (o barra deslizante), se puede seleccionar la amplitud del escalón en *cm*.

5.2.2.2 Rampa

Haciendo uso de una barra deslizante, se selecciona la pendiente de la rampa en tanto por ciento.

5.2.2.3 Seno

La interfaz de la entrada senoidal permite introducir, mediante dos *sliders*, la amplitud de la onda en *cm* así como su frecuencia en *Hz*.

5.2.2.4 Condiciones iniciales

Las condiciones iniciales pueden ser de velocidad o posición. También se puede imponer, como condición inicial, una fuerza inicial sobre el chasis que se mantenga durante la simulación o que deje de actuar justo en el instante inicial de la misma. Esto se realiza marcando o desmarcando la casilla "Mantener" respectivamente.

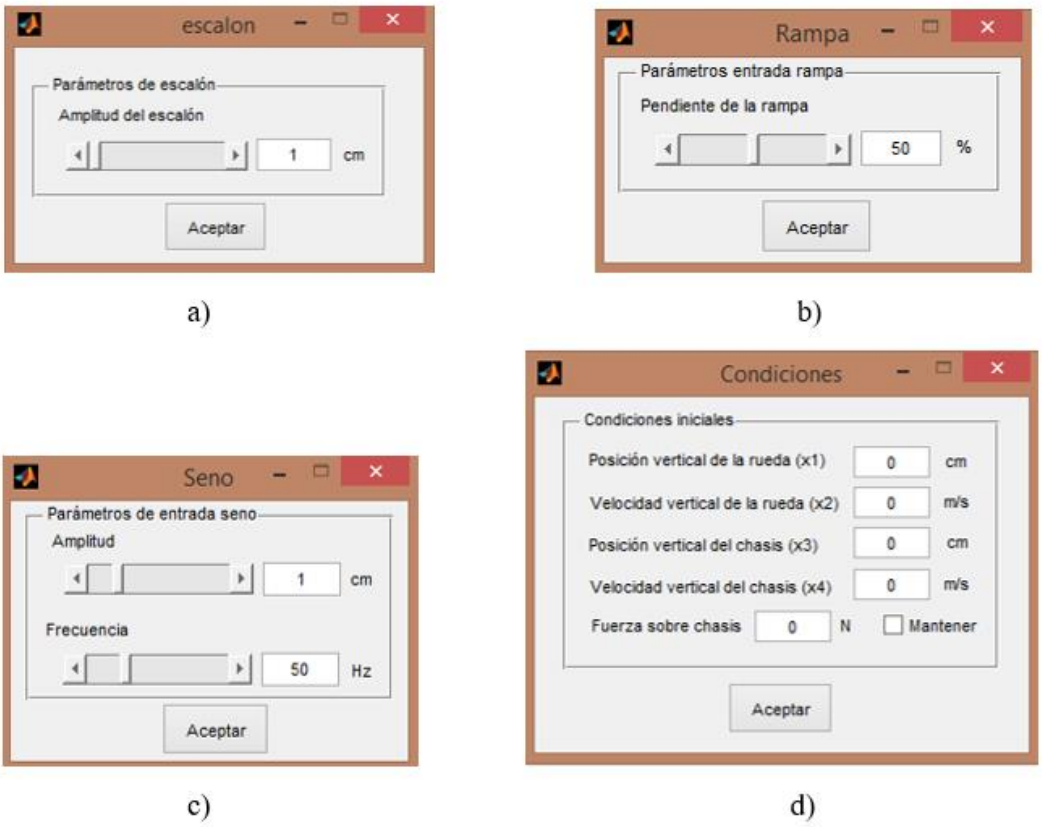


Figura 5.3. Interfaces para la entrada entrada escalón a), *rampa* b), *senoidal* c) *y condiciones iniciales* d).

5.2.3 Interfaz de resultados avanzados (Avanzados.m)

Una vez realizada la simulación y al pulsar el botón "Ver resultados avanzados", se visualiza esta interfaz de usuario en la que se dibujan resultados en frecuencia, así como velocidades y aceleraciones tanto del chasis como de la rueda (ver *Figura 5.4*).

Haciendo uso del botón “Mostrar rueda” o “Mostrar chasis”, dependiendo del caso, se puede alternar entre representar los resultados de la masa no suspendida o del chasis respectivamente.

Por último, se expone la utilidad de los siguientes botones de la interfaz:

- Ver estudio teórico: muestra el contenido de la presente memoria en formato *.pdf*.

- Modelo 3D: muestra un modelo tridimensional de Andábata en formato *.pdf*. Dicho modelo es interactivo, por lo que se puede rotar y ampliar haciendo uso del ratón.

- Exportar: abre la interfaz de exportación (Exportar.m) que permite generar archivos *.csv* con los datos deseados que pueden ser interpretados por SolidWorks para crear animaciones tridimensionales. Este procedimiento se explica en el siguiente apartado 5.3.

- Video simulación: reproduce un vídeo de demostración en formato *.mp4* de una animación realizada en SolidWorks mediante los datos generados por *mcvAndábata* haciendo uso de la función “Exportar”.

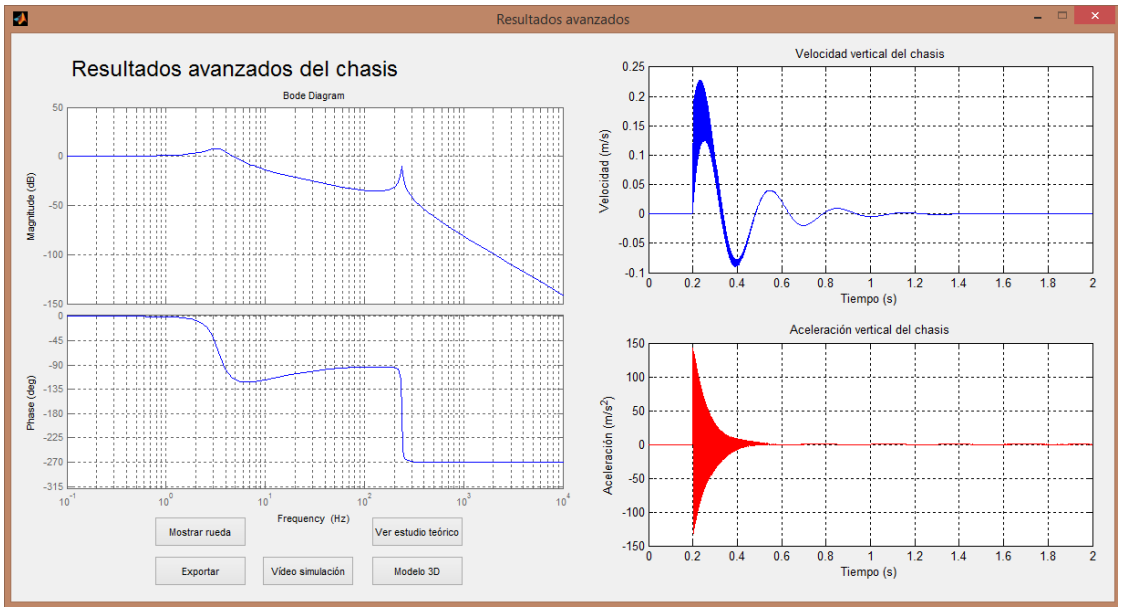


Figura 5.4. Interfaz de resultados avanzados (Avanzados.m).

5.2.4 Interfaz de exportación de datos (Exportar.m)

Mediante esta interfaz el usuario puede seleccionar qué datos desea guardar en archivos *.csv*. El vector de tiempo será siempre exportado junto con los otros datos seleccionados por el usuario. Al pulsar el botón “Aceptar” tras haber marcado los vectores que se quieren exportar, se generarán dos archivos *.csv*, uno para los datos de la masa no suspendida y otro que alberga los datos del chasis.

5.3 Exportación de datos entre *mcvAndábata* y SolidWorks

En el presente apartado se explican los pasos a seguir para transferir los datos calculados por la aplicación *mcvAndábata* a SolidWorks. Esto permitirá al usuario crear una animación usando el modelo tridimensional del robot (ver *Figura 5.5*).

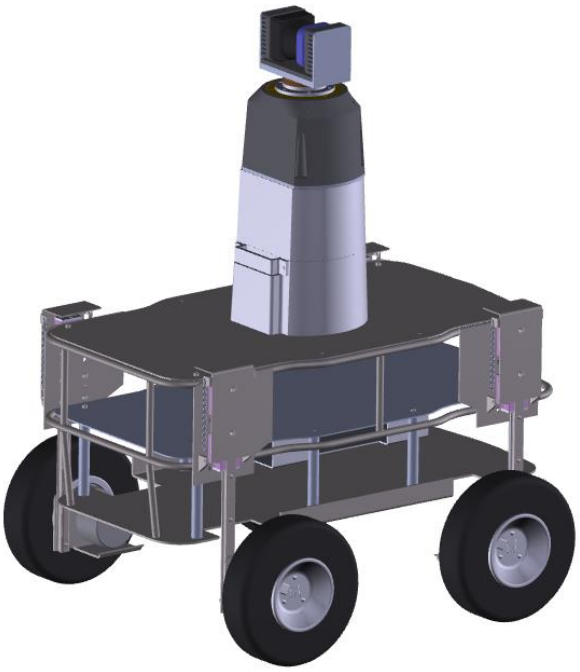


Figura 5.5. Modelo tridimensional de Andábata en SolidWorks.

5.3.1 Procedimiento

El procedimiento para realizar la exportación es el descrito a continuación:

1. Abrir la aplicación *mcvAndábata*.

2. Simular una situación de navegación.

3. Pulsar en el botón “Ver resultados avanzados” (ver *Figura 5.6*). Esto abrirá una nueva ventana y puede tardar unos segundos.

4. Abrir la ventana de exportación usando para ello la opción “Exportar” (ver *Figura 5.7*).

5. Una vez abierto el menú de exportación, seleccionar los datos que se quieren registrar. El vector de tiempo siempre se guardará junto a los demás datos deseados. Se generarán dos archivos *.csv*; uno para los datos referentes a la masa no suspendida (*ruedaData.csv*) y otro para los referentes al chasis (*chasisData.csv*). Aparecerá una ventana emergente informando que debe tener permisos de administrador para llevar a cabo esta tarea ya que, de lo contrario, Matlab no podrá crear estos nuevos archivos.

6. Abrir SolidWorks y el modelo 3D de Andábata. Crear un nuevo estudio de movimiento.

7. Seleccionar el modo de estudio como *"Animación"* y abrir el menú de opciones de la utilidad *"motor rotatorio"* (ver *Figura 5.8*).

8. En dicho menú, seleccionar *"motor lineal (actuador)"* y elegir el ensamblaje que se quiere animar. Por último, en este menú, seleccionar dentro del apartado *"Movimiento"* la opción *"Puntos de datos"* que se encuentra en el panel desplegable.

9. Esta nueva ventana permite importar los datos contenidos en los archivos *.csv* que se crearon en el *paso 5*. Se debe especificar que los vectores que se están leyendo son de desplazamiento mediante el panel desplegable (ver *Figura 5.9*).

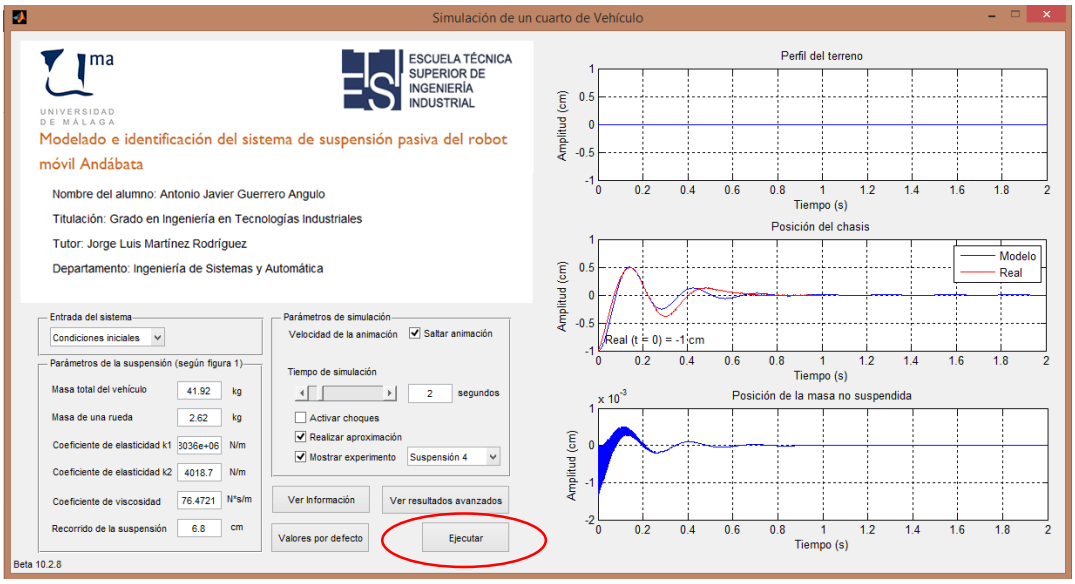


Figura 5.6. Paso 3, Ver resultados avanzados.

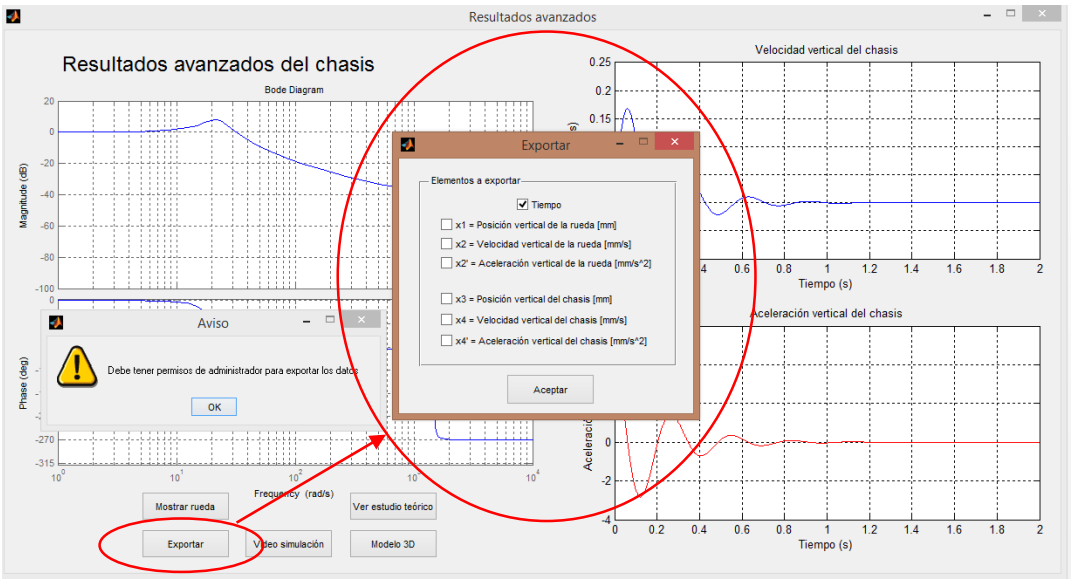


Figura 5.7. Paso 4, menú exportar.

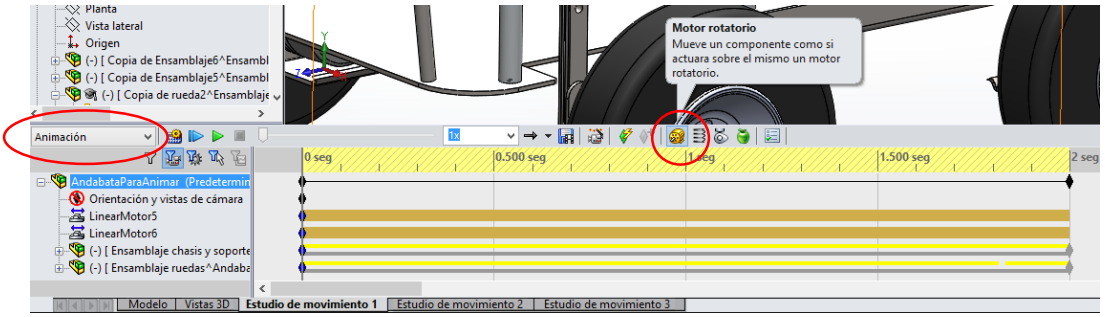


Figura 5.8. Paso 7, estudio de movimiento.

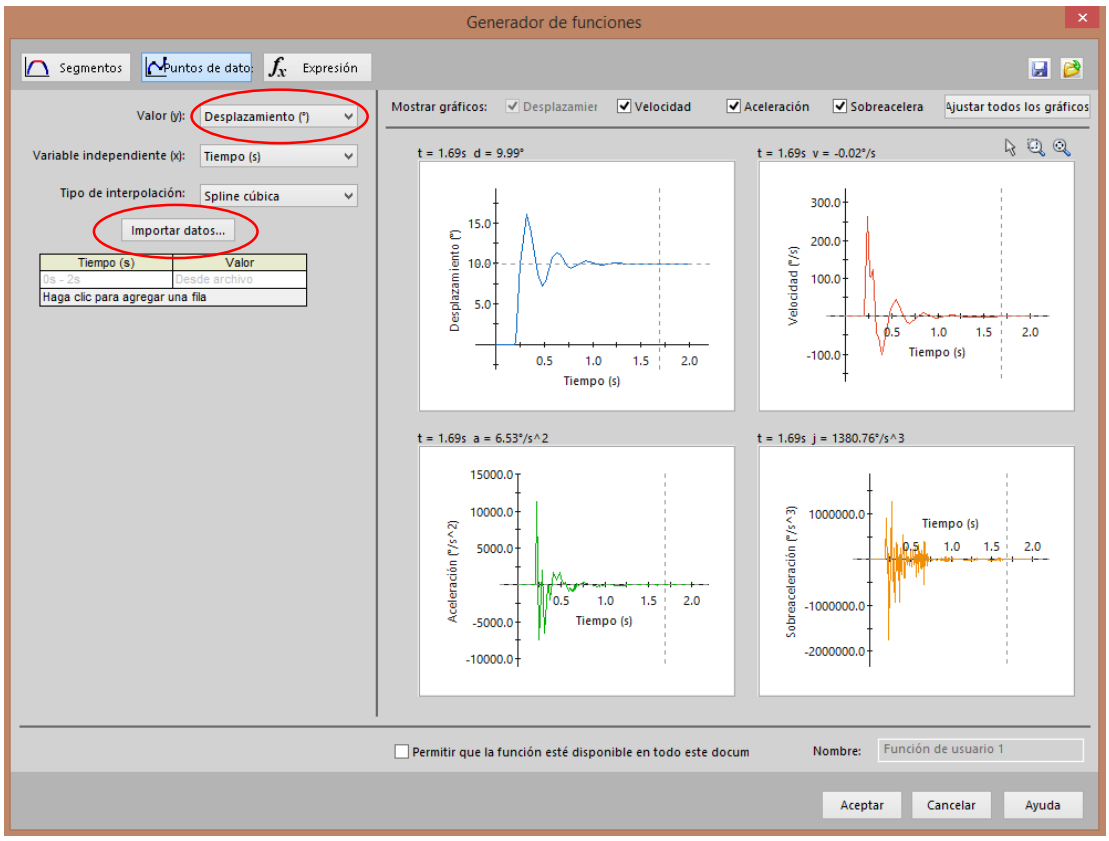


Figura 5.9. Paso 9, importación de datos.

Capítulo 6.
Conclusiones y trabajos futuros

6.1 Conclusiones

Se ha cumplido el objetivo principal del presente Trabajo Fin de Grado al haber caracterizado y modelado el sistema de suspensión pasiva del robot móvil Andábata. Gracias a ello, se conoce mejor el comportamiento de este vehículo ante irregularidades del terreno.

Más concretamente, las conclusiones obtenidas en este TFG son las siguientes:

- El modelo de un cuarto de vehículo proporciona una buena aproximación al comportamiento real de la suspensión cuando no se presentan colisiones (discontinuidades en la resolución de las ecuaciones de estado).

- La simplificación de considerar que el movimiento de las masas de las distintas suspensiones no influye sobre la dinámica de las demás hace que la fiabilidad de los resultados en los experimentos con choque sea algo más pobre.

- Las suspensiones de Andábata tienen comportamientos considerablemente distintos por razones de construcción. Estas discrepancias se ven ampliadas más aún cuando se producen choques entre la masa suspendida y no suspendida.

- Para la realización de este TFG, el único presupuesto empleado ha sido la adquisición de dos potenciómetros lineales por 15,28 euros y una pila de 9 V por 1,95 euros. El resto de componentes (osciloscopio, báscula, etc.) han sido proporcionados por el departamento ISA y el robot por el proyecto de investigación [13].

6.2 Conclusiones a título personal

Personalmente, considero que este Trabajo Fin de Grado me ha ayudado a incrementar notablemente mi capacidad de análisis y síntesis. Así, la programación de una aplicación independiente a partir de Matlab ha supuesto un reto que nunca antes había afrontado y, gracias a ello, he aprendido mucho sobre la elaboración de GUIs.

Además, me he familiarizado con el método de optimización multivariable *Simplex*, que es de amplia aplicación en Ingeniería. La elaboración de esta memoria técnica, con todas las correcciones recibidas, también ha contribuido a mi apredizaje.

Por último, he tenido mi primer contacto con el campo de los robots móviles terrestres, que ha logrado despertar un gran interés en mí hacia la Ingeniería Mecatrónica en general. Por estas razones, pienso que he obtenido unos conocimientos muy valiosos.

6.3 Trabajos futuros

Como continuación de este estudio se podrían plantear los siguientes objetivos como trabajos futuros:

- Realizar una simulación del comportamiento completo de las suspensiones del robot Andábata, considerando las influencias del movimiento de unas suspensiones sobre las demás. Para ello, se puede considerar variable la masa del chasis que soporta cada rueda dependiendo de la altura de cada suspensión y de la inclinación del terreno que se transita.

- Añadir nuevas utilidades a la aplicación *mcvAndabata* como puede ser, por ejemplo, una animación tridimensional de Andábata sobre distintos tipos de terreno.

Anexo A.
Códigos Matlab

A.1. Códigos para optimización

A continuación, se exponen los códigos que han sido escritos en Matlab para determinar los parámetros buscados. Es importante tener en cuenta que los siguientes códigos llaman a funciones ya programadas en el apartado A.2. La función con la que se comienza la optimización es "optimizarParametros".

v = coste(x)

```
% Función v = coste(x)
%
% Evalúa la función de coste dada por fcoste.
% Atributos x = [k1, k2, b]
% Es llamada por fminsearch.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function v = coste(x)
global M1 M2 tiempo r x0 y_osciloscopio fcoste Y T

% Parámetros a optimizar
b = x(1);
k1 = x(2);
k2 = x(3);

% Espacio de estados
A=[0 1 0 0; -(k1+k2)/M1 -b/M1 k2/M1 b/M1; 0 0 0 1; k2/M2 b/M2 -k2/M2 -
b/M2];
B=[0; k1/M1; 0; 0];
C_chasis=[0 0 1 0];  %La salida es la posición vertical del chasis
D=0;
stateSpace=ss(A,B,C_chasis,D);

%Simulación mediante las ecuaciones de estado
[Y,T,X] = lsim(stateSpace,r,tiempo,x0);

% Cálculo de la función de coste
switch fcoste
    case 1
        v = sum(abs(Y-y_osciloscopio));
    case 2
        v = sum((Y-y_osciloscopio).^2);
    case 3
        v = max(abs(Y-y_osciloscopio));
end
end
```

v = costeChoque(x)

```
% Función v = costeChoque(x)
%
% Evalúa la función de coste dada por fcoste.
% Atributos: x = [cr, tImpacto].
%
% Es llamada por fminsearch.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function v = costeChoque(x)

global tiempo x0 y_osciloscopio fcoste T Y tImpacto

%Simulación mediante las ecuaciones de estado
cr = x(1);

% El tiempo de impacto tiene una precisión de milisegundos
tImpacto = fix(x(2)*100)/100;

% Resolución considerando choques
[T,X] = resolverChoquesMod(cr,tiempo,x0/100);
Y = X(:,3)*100;
Y = Y(1:length(y_osciloscopio));

% Funciones de coste
switch fcoste
    case 1
        v = sum(abs(Y-y_osciloscopio));
    case 2
        v = sum((Y-y_osciloscopio).^2);
    case 3
        v = max(abs(Y-y_osciloscopio));
end

end
```

[detect, stopin, direction] = eventoChoqueTiempo(t,x)

```
% Función [detect, stopin, direction] = eventoChoqueTiempo(t,x)
%
% Detecta los eventos de interrupción por colisiones. La función que
% controla el evento está basada en el tiempo de impacto para el
% primer choque. Para los choques posteriores se calcula la distancia
% existente entre las masas.
%
%   detect: Devuelve 1 cuando la función pasa definida por cero
%   stopin: Si se establece como uno se interrumpirá la resolución
%   direction: Establece el criterio de paso por 0 de la función.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function [detect, stopin, direction] = eventoChoqueTiempo(t,x)

global tImpacto nChoques distChoque

    % Función que se quiere evaluar su paso por cero
    if nChoques == 0
        detect = tImpacto-t;
    else
        detect = x(1)-x(3)+distChoque;
    end

    % Se parará la ejecución al detectar el evento
    stopin = 1;

    % Se tomará como evento un paso por cero de positivo a negativo.
    direction = -1;

end
```

parametros = optimizarParametros(rueda, choque)

```
% Función parametros = optimizarParametros(rueda, choque, fun_coste)
%
% Función que hace uso del método de optimización Simplex para
% encontrar los parámetros k1, k2 y b de la suspensión numerada por el
% atributo "rueda".
%
% "choque" toma el valor 1 si se desea evaluar los experimentos con
% choques y el valor 0 en caso contrario.
%
% "fun_coste" determina la función de coste empleada para el cálculo.
%   fun_coste = 1    ->    J(k1,k2,b) = sum(abs(error))
%   fun_coste = 2    ->    J(k1,k2,b) = sum(error^2)
%   fun_coste = 3    ->    J(k1,k2,b) = sum(max(abs(error))
%
% "parametros" es un vector de salida de tres componentes de la forma
% [b, k1, k2] si choque = 0.
%
% Si choque = 1, "parametros" es de la forma [cr, tImpacto].
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function parametros = optimizarParametros(rueda, choque, fun_coste)

global M1 M2 y_osciloscopio tiempo r x0 fcoste k1 k2 b T Y
fcoste = fun_coste;

% Respuesta real
[tiempo, y_osciloscopio] = getRespuesta(rueda, choque);

% Parámetros conocidos
M1 = 2.62;
M2 = (41.92-4*M1)/4;

% Entrada nula en todo el intervalo evaluado
r = zeros(1, length(tiempo));

% Condiciones iniciales
x0 = [0 0 y_osciloscopio(1) 0];

if choque == 1
    % Establecer base de datos de las distintas suspensiones y
    % funciones de coste. Las filas vienen dadas por 'rueda', las
    % columnas son los parámetros k1, k2 y b respectivamente.
    datos1 = [8.9185*10^5 3780.3 93.5524;
        2.8071*10^5 3875.3 72.265;
        1.1887*10^5 3649.7 76.9565;
        9.448*10^5 4018.7 76.4721];

    datos2 = [5.8442*10^5 3755.3 75.2045;
        5.8529*10^5 3900.9 73.5224;
        4.9510*10^5 3830.1 77.5951;
        4.0215*10^5 3744 69.3376];

    datos3 = [1.4498*10^5 4186.8 82.9478;
        2.3524*10^5 4322.8 67.5031;
        9.5862*10^4 4578.9 82.7364;
```

```
        4.2109*10^5 4076.1 73.0047];

    tablaDatos(:,:,1) = datos1;
    tablaDatos(:,:,2) = datos2;
    tablaDatos(:,:,3) = datos3;

    % Definir los parámetros b, k1, k2
    k1 = tablaDatos(rueda,1,fcoste);
    k2 = tablaDatos(rueda,2,fcoste);
    b = tablaDatos(rueda,3,fcoste);

    % Búsqueda de parámetro óptimo 'cr' (coeficiente de restitución)
    % y de "t_in" (tiempo inelástico).
    parametros = fminsearch('costeChoque', [0.7 0.1]);
    plot(tiempo,Y,'--')
    hold on
    plot(tiempo,y_osciloscopio,'r')
    axis([0,1,-4,2])
    grid
    text(tiempo(1), y_osciloscopio(1)+abs(0.2*y_osciloscopio(1)), ['
        Real (t = 0) = ' num2str(y_osciloscopio(1)) ' cm'])
    legend('Modelo','Real')
    %xlabel('tiempo (s)')
    ylabel('cm')
    hold off

else
    % Búsqueda de parámetros óptimos (b, k1, k2)
    parametros = fminsearch('coste', [20 22800 2332.3]);
    plot(T,Y,'--')
    hold on
    plot(T,y_osciloscopio,'r')
    grid
    text(T(1), y_osciloscopio(1)+abs(0.2*y_osciloscopio(1)), ['  Real
       (t = 0) = ' num2str(y_osciloscopio(1)) ' cm'])
    legend('Modelo','Real')
    xlabel('tiempo (s)')
    ylabel('cm')
    hold off
end
end
```

`[T, X, nChoques] = resolverChoquesMod(cr, tiempo x0)`

```
% Función [T,X,nChoques] = resolverChoquesMod(cr,tiempo,x0)
%
% Versión modificada de la función resolverChoques que usa un vector
% de tiempo completo especificado en vez del tiempo final.
%
%   cr     -> coeficiente de restitución.
%   tiempo -> vector tiempo de simulación completo.
%   x0     -> Condiciones iniciales.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function [T,X] = resolverChoquesMod(cr,tiempo,x0)

global M1 M2 nChoques distChoque

    t_fin = tiempo(end);

    % Contador de número de choques producidos
    nChoques = 0;

    % Opción para detener la resolución numérica posterior
    opts = odeset('events', @eventoChoqueTiempo, 'AbsTol', 0.0001);

    % Inicialización del tiempo trancurrido
    T = 0;

    % Inicialización del tiempo en el que se produce el evento
    te = 0;

    % Comienza la resolución buscando eventos hasta t_fin
    while T(end) < t_fin

        % Simula la dinámica. Detiene la integración cuando detecta un
        % choque.
        [T,X,te,xe,ie] = ode23(@simulacionChoque,
            (fix(te*100)/100):0.001:t_fin, x0, opts);

        % Si se ha producido un choque se procesa
        if T(end) < t_fin
            % Cuenta los choques producidos
            nChoques=nChoques+1;

            % Se calculan las velocidades después de la colisión a
            partir de las velocidades inmediatamente anteriores
            v = choque([xe(4) xe(2)], [M2 M1], cr);

            % Las nuevas condiciones iniciales para retomar el cálculo
            de las ecuaciones diferenciales son las velocidades después
            del choque y las posiciones en el instante del evento
            x0 = [xe(1) v(2) xe(3) v(1)];

            % Primer choque que se produce
            if nChoques == 1
                % Guarda los resultados de tiempo
                Ttotal = T;
```

```
                distChoque = xe(3);

                % Guarda los resultados de las variables de estado
                Xtotal = X;

            % Choques sucesivos al primero
            else
                % Concatena los resultados con los anteriores
                Ttotal = vertcat(Ttotal, T);
                Xtotal = vertcat(Xtotal, X);
            end

        end

    end

    % Si se han producido choques completa el intervalo final restante
    if nChoques > 0
        % Concatena los resultados con los anteriores para completar
        el último intervalo
        T = vertcat(Ttotal, T);
        X = vertcat(Xtotal, X);
    end

end
```

dx = simulacionChoque(t,x)

```
% Función dx = simulacionChoque(t,x)
%
% Simula el sistema y devuelve los resultados de las variables de estado.
% Es llamada por ode23.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga


function dx = simulacionChoque(t,x)

global k1 k2 M1 M2 b

dx=zeros(4,1);

% Ecuaciones de estado
dx(1) = x(2);
dx(2) = (b*(x(4)-x(2))+k2*x(3)-(k1+k2)*x(1))/M1;
dx(3) = x(4);
dx(4) = -(k2*(x(3)-x(1))+b*(x(4)-x(2)))/M2;

end
```

A.2. Códigos de *mcvAndábata*

En este apartado se muestran todos los códigos que componen la aplicación *mcvAndábata*. La aplicación se lanza a partir de la función "main".

```
yOK = acondicionar(y)
```

```
% Función yOK = acondicionar(y)
%
% Trata los datos de "y" para eliminar tiempo muerto antes de comenzar
% la respuesta y además eliminar el offset final de la señal haciendo
% que ésta se estabilice en 0.
%
% "t" es el vector de tiempo del experimento original.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga



function [tOK, yOK] = acondicionar(y, t)

% Búsqueda de la posición "k" donde empieza la respuesta empleando
para ello un umbral de +/-0.3
for k = 1:length(t)
    if y(k) > y(1)+0.3 || y(k) < y(1)-0.3
        break
    end
end

% Un vector de tiempo de 1 segundo es suficiente para visualizar la
% respuesta
tOK = 0:0.001:1;

% Nuevo vector "y" al que se le ha eliminado el desplazamiento durante
el
% tiempo muerto
y = y(k:length(tOK)+(k-1));

% Elimina el offset final
yOK = y-abs(y(end));

end
```

animacionMCV()

```
% Este código realiza toda la animación, cálculo y representaciones en
% la interfaz gráfica principal.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga


global m_vehiculo m_rueda M2 M1 k2 k1 b N boolMantener boolAnimacion
t_fin u F completado terminar ejecutando x1 x2 x3 x4 X G_chasis
G_rueda T lonReposo seleccion boolApro experimento boolExperimento
boolChoque boolChoqueExperimento cr h

% CÁLCULO DE LOS PARAMETROS MECÁNICOS DEL VEHÍCULO

% Masa del chasis que le corresponde a cada una de la suspensiones.
M2 = (m_vehiculo - 4*m_rueda)/4;

% Masa de la rueda.
M1 = m_rueda;

% Puesta a 0 de variable que permite interrumpir la animación
terminar = 0;

% Variable que indica el estado de la ejecución del código. Toma el
valor 1 si se está ejecutando y 0 si no lo está.
ejecutando = 1;

% Calcula la elongación inicial de los muelles debido a la masa del
chasis y al valor de la constante elástica del conjunto de resortes en
cm.
lonReposo = 100*(M2*9.81)/k2-0.728;

% Durante la ejecución el botón de ejecutar toma la funcionalidad de
% detener la animación.
set(handles.ejecutar,'String','Parar')

% Las variables de estado definidas son:
% x1: posición vertical de la rueda
% x2: velocidad vertical de la rueda
% x3: posición vertical del chasis
% x4: velocidad vertical del chasis

% DEFINICIÓN DEL ESPACIO DE ESTADOS DEL SISTEMA DE ESTUDIO
A=[0 1 0 0; -(k1+k2)/M1 -b/M1 k2/M1 b/M1; 0 0 0 1; k2/M2 b/M2 -k2/M2 -
b/M2];
B=[0; k1/M1; 0; 0];

%La salida es la posición vertical del chasis
C_chasis=[0 0 1 0];

%La salida es la posición vertical de la rueda
C_rueda=[1 0 0 0];

D=0;
```

```
% Espacio de estados
stateSpace=ss(A,B,C_chasis,D);

% La barra de ejecución avanza al 25%
waitbar(0.25)

% FUNCIONES DE TRANSFERENCIA
% Estas funciones de transferencia tienen utilidad en Avanzados.m para
la representación de diagramas de Bode.
[NUM_chasis,DEN_chasis] = ss2tf(A,B,C_chasis,D,1);
G_chasis = tf(NUM_chasis,DEN_chasis); % función de transferencia
                              perfil del terreno -> posición del chasis

[NUM_rueda,DEN_rueda] = ss2tf(A,B,C_rueda,D,1);
G_rueda = tf(NUM_rueda,DEN_rueda);    % función de transferencia
                              perfil del terreno -> posición de la rueda

% La barra de ejecución avanza al 50%
waitbar(0.5)

% SIMULACIÓN, DETERMINAR LAS CONDICIONES INICIALES
% El usuario quiere aplicar una fuerza como condición inicial.

% Esta variable se usará para no sobreescribir la fuerza F
f = 0;

% Si el usuario quiere aplicar una carga se evalúan los posibles
casos, si no se leen las condicones iniciales de velocidad y posición.
F es una variable global que será utilizada por simulacion.m
if F~=0
    % El usuario quiere mantener la carga aplicada
    if boolMantener == 1
        % Condiciones iniciales en cm
        x0 = [x1/100 x2 x3/100 x4];
        f = F;

    % Calcula las condiciones iniciales generadas por dejar de aplicar
    % una fuerza sobre el chasis
    else
        [ci1, ci2, ci3, ci4] = determinarCI;
        x0 = [ci1 ci2 ci3 ci4];
        f = F;

        % La fueza deja de ser aplicada al comienzo de la simulación.
        F = 0;
    end

% Condiciones iniciales simples definidas por el usuario sin fuerza
aplicada
else
    x0 = [x1/100 x2 x3/100 x4];
end

% SIMULACIÓN MEDIANTE ECUACIONES DE ESTADO
% Si la simulación de colisiones está activada se procede a
detectarlas, si no simplemente se resuelve las ecuaciones de estado
con ode45 ya que no habrá interrupciones (eventos).
```

```
if boolChoqueExperimento == 1
    [T,X] = resolverChoquesExperimento(cr,t_fin,x0);

% Caso en el que no se quieren detectar los choques
elseif boolChoque == 1
    [T,X] = resolverChoques(cr,t_fin,x0);
else
    [T,X] = ode23(@simulacion, [0 t_fin], x0);
end

% La barra de ejecución avanza al 75%
waitbar(0.75)

% Calula el perfil del terreno
u = groundInput(T, seleccion, boolApro, 1);

% posición relativa del chasis en cm
y = X(:,3)*100;

% posición relativa de la rueda en cm
x = X(:,1)*100;

% La barra de ejecución avanza al 100%
waitbar(1)
close(h)

% Comienza la animación si está activada
if boolAnimacion == 1

    % ANIMACIÓN A PARTIR DE LA SIMULACIÓN REALIZADA
    axes(handles.figura1)
    imshow('grey.png')
    axes(handles.animacion1)

    for k = 1:N/10:length(T)
        % Detiene la animación en este caso
        if terminar == 1
            break

        else
            % Dibuja el chasis
            plotMasa(0, y(k)-lonReposo+18+12.5-11, 0.2, 11, [100 100
              100])

            hold on

            % Muestra la fuerza aplicada
            if F ~=0
                text(-0.15, y(k)-lonReposo+18+15, ['F = ' num2str(F) '
                       N'])
            end

            % Dibuja el resorte izquierdo
            plotResorte(-0.015, y(k)-lonReposo+18+12.5-11, 0.01,
                        ((x(k)+10+lonReposo)-y(k))/9)

            % Dibuja el resorte derecho
```

```
plotResorte(0.055, y(k)-lonReposo+18+12.5-11, 0.01,
            ((x(k)+10+lonReposo)-y(k))/9)

% Dibuja la guía lineal
plotMasa(0.02, x(k), 0.02, 18, [188 188 188])

% Dibuja el soporte superior de los resortes
plotMasa(0.02, x(k)+29, 0.08, 1.5, [188 188 188])

% Apoyo inferior resorte izquierdo
plotMasa(-0.015, y(k)-lonReposo+18+12.5-11, 0.01, 1.5,
            [188 188 188])

% Apoyo inferior resorte derecho
plotMasa(0.055, y(k)-lonReposo+18+12.5-11, 0.01, 1.5, [188
            188 188])

% Dibuja el soporte de la guía lineal
plotMasa(0.02, x(k)+18, 0.06, 12.5, [188 188 188])

% Dibuja los tornillos de la suspensión
plot(0.02, x(k)+21.6, 'o','markersize', 2,
       'markeredgecolor', 'k', 'markerfacecolor', [128 128
       128]/255)
plot(0.02, x(k)+27.6, 'o','markersize', 2,
       'markeredgecolor', 'k', 'markerfacecolor', [128 128
       128]/255)

% Texto "suspensión"
text(0.15, x(k)+30, 'Suspensión (k2, b)')

% Texto "chasis"
text(0.15, y(k)+18, ['Chasis (M2): '
       num2str(round((y(k)+lonReposo)*100)/100) ' cm'])

% Dibuja la rueda
plot(0, x(k), 'o','markersize', 50, 'markeredgecolor',
       'k', 'linewidth', 15,'markerfacecolor', [128 128
       128]/255)

% Texto "rueda"
text(0.15, x(k)-5,['Rueda (M1, k1): '
       num2str(round(x(k)*100)/100) ' cm'])

% Especifica el centro de masas de la rueda
plot(0, x(k), '+')

% Dibuja el perfil del terreno
plot(T-T(k),u,'r','linewidth',2)

title('Animación')
xlabel(['Tiempo: ' num2str(round(T(k)*100)/100) '
       segundos'])
ylabel('Posición vertical (cm)')
ylim([-20 40])
xlim([-0.5 0.5])
set(gca,'XTickLabel',[])
```

```
            hold off

            % Fuerza el refresco de la animación en cada iteración
            drawnow

        end
    end

% Se muestra el panel de información en lugar de la animación
else
    axes(handles.figura1);
    imshow('informacion.png')
end

% Retoma el valor anterior de F para que sea recordado por
Condiciones.m
F = f;

% REPRESENTACIÓN DE LA EVOLUCIÓN TEMPORAL

% Si la animación se ha detenido se muestra el panel de esquema
if terminar == 1
    axes(handles.figura2)
    imshow('esquema.png')

% Si la animación no se ha detenido se dibujan los resultados
else
    axes(handles.figura2)
    imshow('grey.png')

    % Dibuja la entrada del sistema
    axes(handles.figura3)
    plot(T, u)
    grid
    title('Perfil del terreno')
    xlabel('Tiempo (s)')
    ylabel('Amplitud (cm)')

    % Dibuja la posición de la rueda
    axes(handles.figura4)
    plot(T, X(:,1)*100)
    grid
    title('Posición de la masa no suspendida')
    xlabel('Tiempo (s)')
    ylabel('Amplitud (cm)')

    % Dibuja la posición del chasis
    axes(handles.figura5)
        plot(T, X(:,3)*100)

    % Muestra los experimentos deseados
    if boolExperimento == 1
        hold on
        plotRespuesta(experimento, boolChoque, 'r')
        legend('Modelo', 'Real')
        hold off
    end
```

```
        grid
        title('Posición del chasis')
        xlabel('Tiempo (s)')
        ylabel('Amplitud (cm)')

        %title('Suspensión 4')
        %xlabel('tiempo (s)')
        %ylabel('cm')

    end

    % Reestablece la habilidad de los botones
    set(handles.botonInformacion,'Visible','on')
    set(handles.botonInformacion,'String','Ver Información')
    set(handles.botonAvanzados,'Visible','on')

    set(handles.botonInformacion,'Enable','on')
    set(handles.botonDefecto,'Enable','on')
    set(handles.botonAvanzados,'Enable','on')
    set(handles.tipoTerreno,'Enable','on')
    set(handles.ejecutar,'String','Ejecutar')

    ejecutando = 0;
    completado = 1;
```

Avanzados()

```
function varargout = Avanzados(varargin)
% Función varargout = Seno(varargin)
%
% Interfaz gráfica de usuario para la toma de parámetros de entrada
% tipo senoidal.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

% Código de instalación inicial, generado automáticamente por la GUI
toolbox de Matlab.
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Avanzados_OpeningFcn, ...
                   'gui_OutputFcn',  @Avanzados_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin de código de instalación inicial


% --- Se ejecuta sólo cuando Avanzados() es visible, función generada
automáticamente por la GUI toolbox de matlab.
function Avanzados_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

guidata(hObject, handles);

global T X G_chasis k1 k2 b M1 M2 ac_chasis ac_rueda nPantalla u h

% Establece que los siguientes resultados a mostrar si se pulsa el
% botonAlternar sean los de la masa no suspendida
nPantalla = 2;

% Cálculo de aceleraciones verticales
ac_chasis = -(k2*(X(:,3)-X(:,1))+b*(X(:,4)-X(:,2)))/M2;
ac_rueda = (k2*(X(:,3)-X(:,1))+b*(X(:,4)-X(:,2))-k1*(X(:,1)-u'))/M1;

% Barra de carga al 50%
waitbar(0.5)

% Dibuja el diagrama de Bode del chasis
axes(handles.bodeFig)
bode(G_chasis)
set(cstprefs.tbxprefs,'FrequencyUnits','Hz')
grid
```

```
% Dibuja la velocidad del chasis
axes(handles.velocidadFig)
plot(T,X(:,4))
grid
title('Velocidad vertical del chasis')
xlabel('Tiempo (s)')
ylabel('Velocidad (m/s)')

% Dibuja la aceleración del chasis
axes(handles.aceleracionFig)
plot(T,ac_chasis,'r')
grid
title('Aceleración vertical del chasis')
xlabel('Tiempo (s)')
ylabel('Aceleración (m/s^2)')

% Barra de carga al 100%
waitbar(1)
close(h)


% --- Las salidas de esta función son devueltas a la línea de comado,
función generada automáticamente por la GUI toolbox de matlab.
function varargout = Avanzados_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


% --- Se ejecuta al presionar el botonAlternar.
function botonAlternar_Callback(hObject, eventdata, handles)

global ac_rueda ac_chasis X G_chasis G_rueda nPantalla T

% Abre la barra de carga
h = waitbar(0, 'Cargando, por favor espere...');

% Elige los resultados mostrar
switch nPantalla
    % Resultados de velocidad o aceleración del chasis
    case 1
        set(handles.botonAlternar,'String','Mostrar suspensión')

        set(handles.textInfo,'String','Resultados avanzados del
              chasis')

        % Dibuja el diagrama de Bode de la masa no suspendida
        axes(handles.bodeFig)
        bode(G_chasis)
        set(cstprefs.tbxprefs,'FrequencyUnits','Hz')
        grid

        % Dibuja la velocidad de la masa suspendida
        axes(handles.velocidadFig)
        plot(T,X(:,4))
        grid
        title('Velocidad vertical del chasis')
        xlabel('Tiempo (s)')
        ylabel('Velocidad (m/s)')
```

```
        % Dibuja la aceleración rueda
        axes(handles.aceleracionFig)
        plot(T,ac_chasis,'r')
        grid
        title('Aceleración vertical del chasis')
        xlabel('Tiempo (s)')
        ylabel('Aceleración (m/s^2)')
        nPantalla = 2;

    % Resultados de velocidad o aceleración de la masa no suspendida
    case 2
        set(handles.botonAlternar,'String','Mostrar chasis')

        set(handles.textInfo,'String','Resultados avanzados de la masa
              no suspendida')

        % Dibuja el diagrama de Bode de la masa no suspendida
        axes(handles.bodeFig)
        bode(G_rueda)
        set(cstprefs.tbxprefs,'FrequencyUnits','Hz')
        grid

        % Dibuja la velocidad de la masa no suspendida
        axes(handles.velocidadFig)
        plot(T,X(:,2))
        grid
        title('Velocidad vertical de la masa no suspendida')
        xlabel('Tiempo (s)')
        ylabel('Velocidad (m/s)')

        % Dibuja la aceleración rueda
        axes(handles.aceleracionFig)
        plot(T,ac_rueda,'r')
        grid
        title('Aceleración vertical de la masa no suspendida')
        xlabel('Tiempo (s)')
        ylabel('Aceleración (m/s^2)')

        nPantalla = 1;
end

% Barra de carga al 100%
waitbar(1)
close(h)


% --- Se ejecuta al presionar botonEstudio.
function botonEstudio_Callback(hObject, eventdata, handles)

h = waitbar(0, 'Cargando, porfavor espere...');

% Abre un archivo .pdf que muestra la obtención del modelo de un
cuarto de vehículo
winopen('modeloCuartoVehiculo.pdf')
waitbar(1)
close(h)
```

```
% --- Se ejecuta al presionar botonExportar.
function botonExportar_Callback(hObject, eventdata, handles)

% Llama a la función Exportar.m
Exportar
msgbox('Debe tener permisos de administrador para exportar los datos',
'Aviso', 'warn')


% --- Se ejecuta al presionar botonVideo.
function botonVideo_Callback(hObject, eventdata, handles)

h = waitbar(0, 'Cargando, porfavor espere...');

% Carga un vídeo en el que se muestra una animación en SolidWorks
winopen('videoTFG.mp4')

% Barra de carga al 100%
waitbar(1)
close(h)


% --- Se ejecuta al presionar boton3D.
function boton3D_Callback(hObject, eventdata, handles)

h = waitbar(0, 'Cargando, porfavor espere...');

% Abre un modelo 3D de Andábata
winopen('andabata3D.pdf')
waitbar(1)
close(h)
```

v = choque(u, M, cr)

```
% Función v = choque(u, M, cr)
%
% Resuelve el choque entre dos partículas de masas 'M=[M1, M2]',
% velocidades antes del choque 'u=[u1, u2]' y coeficiente de
% restitución 'cr'. Devuelve como resultado las velocidades de ambos
% elementos después del mismo 'v=[v1, v2]'.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function v = choque(u, M, cr)

u1=u(1);
u2=u(2);

M1=M(1);
M2=M(2);

Vcm=(M1*u1+M2*u2)/(M1+M2);      % Velocidad del centro de masas

% Velocidades después del choque
v1=(1+cr)*Vcm-cr*u1;
v2=(1+cr)*Vcm-cr*u2;

v=[v1 v2];

end
```

Condiciones()

```
function varargout = Condiciones(varargin)
% Función varargout = Condiciones(varargin)
%
% Interfaz gráfica de usuario para la toma de condiciones inciales.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

% Código de instalación inicial, generado automáticamente por la GUI
toolbox de Matlab.
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Condiciones_OpeningFcn, ...
                   'gui_OutputFcn',  @Condiciones_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin de código de instalación inicial


% --- Se ejecuta sólo una vez que Condiciones() se hace visible.
function Condiciones_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

guidata(hObject, handles);

global x1 x2 x3 x4 F boolMantener

% Reconoce el valor anterior de los parámetros
set(handles.textX1,'String',x1);
set(handles.textX2,'String',x2);
set(handles.textX3,'String',x3);
set(handles.textX4,'String',x4);
set(handles.textF,'String',F);
set(handles.mantener,'Value',boolMantener);


% --- Las salidas de esta función son devueltas a la línea de
commandos, código generado automáticamente por la GUI toolbox de
matlab.
function varargout = Condiciones_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;
```

```
function textX1_Callback(hObject, eventdata, handles)


% --- Creación de objeto de texto.
function textX1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function textX2_Callback(hObject, eventdata, handles)
% hObject    handle to textX2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textX2 as text
%        str2double(get(hObject,'String')) returns contents of textX2
as a double


% --- Executes during object creation, after setting all properties.
function textX2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textX2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function textX3_Callback(hObject, eventdata, handles)
% hObject    handle to textX3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textX3 as text
%        str2double(get(hObject,'String')) returns contents of textX3
as a double


% --- Executes during object creation, after setting all properties.
function textX3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textX3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function textX4_Callback(hObject, eventdata, handles)
% hObject    handle to textX4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textX4 as text
%        str2double(get(hObject,'String')) returns contents of textX4
as a double


% --- Executes during object creation, after setting all properties.
function textX4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textX4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in aceptarCondiciones.
function aceptarCondiciones_Callback(hObject, eventdata, handles)
% hObject    handle to aceptarCondiciones (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global x1 x2 x3 x4 F boolMantener

% Lee los valores ingresados
x1 = str2double(get(handles.textX1,'String'));
x2 = str2double(get(handles.textX2,'String'));
x3 = str2double(get(handles.textX3,'String'));
x4 = str2double(get(handles.textX4,'String'));
F = str2double(get(handles.textF,'String'));

if get(handles.mantener,'Value') == 1
    boolMantener = 1;
else
    boolMantener = 0;
end

close Condiciones
```

```
function textF_Callback(hObject, eventdata, handles)
% hObject    handle to textF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textF as text
%        str2double(get(hObject,'String')) returns contents of textF
as a double


% --- Executes during object creation, after setting all properties.
function textF_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in mantener.
function mantener_Callback(hObject, eventdata, handles)
% hObject    handle to mantener (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mantener
```

`[ci1, ci2, ci3, ci4] = determinarCI()`

```
% Función [ci1, ci2, ci3, ci4] = determinarCI()
%
% Calcula las condiciones iniciales de las que partiría el sistema en
% el caso de haberle aplicado una fuerza para posteriormente dejar de
% aplicarla.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function [ci1, ci2, ci3, ci4] = determinarCI

global t_fin x1 x2 x3 x4

[T,X] = ode45(@simulacion, [0 t_fin], [x1 x2 x3 x4]);

ci1 = X(end,1);
ci2 = 0;
ci3 = X(end,3);
ci4 = 0;

end
```

escalon()

```
function varargout = escalon(varargin)
% Función varargout = escalon(varargin)
%
% Interfaz gráfica de usuario para la toma parámetros de entrada
% escalón.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

% Código de instalación inicial
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @escalon_OpeningFcn, ...
                   'gui_OutputFcn',  @escalon_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin de código de instalación inicial


% --- Executes just before escalon is made visible.
function escalon_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to escalon (see VARARGIN)

% Choose default command line output for escalon
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes escalon wait for user response (see UIRESUME)
% uiwait(handles.figure1);
global es

% Reconoce el valor anterior de los parámetros
set(handles.amplitud,'Value',es);
set(handles.textAmplitud,'String',es);


% --- Outputs from this function are returned to the command line.
function varargout = escalon_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in aceptar.
function aceptar_Callback(hObject, eventdata, handles)
% hObject    handle to aceptar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global es

% Lee los valores ingresados
es = str2double(get(handles.textAmplitud,'String'));
close escalon


% --- Executes on slider movement.
function amplitud_Callback(hObject, eventdata, handles)
% hObject    handle to amplitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider

% Se toman dos decimales de resolución
v = get(handles.amplitud,'Value');
set(handles.textAmplitud,'String',fix(v*100)/100)

% --- Executes during object creation, after setting all properties.
function amplitud_CreateFcn(hObject, eventdata, handles)
% hObject    handle to amplitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function textAmplitud_Callback(hObject, eventdata, handles)
% hObject    handle to textAmplitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textAmplitud as
text
%        str2double(get(hObject,'String')) returns contents of
textAmplitud as a double
```

```
% --- Executes during object creation, after setting all properties.
function textAmplitud_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textAmplitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

[detect, stopin, direction] = eventoChoque(t,x)

```
% Función [detect, stopin, direction] = eventoChoque(t,x)
%
% Detecta los eventos de interrupción por colisiones.
%
%   detect: Devuelve 1 cuando la función pasa definida por cero
%   stopin: Si se establece como uno se interrumpirá la ejecución de
%           ode23
%   direction: Establece el criterio de paso por 0 de la función.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function [detect, stopin, direction] = eventoChoque(t,x)
    global lonReposo lon

    % Funciones que se quieren evaluar su paso por cero
    detect = [100*(x(1)-x(3))+lonReposo 100*(x(3)-x(1))+lon-
              lonReposo];

    % Se parará la ejecución al detectar el evento
    stopin = [1 1];

    % Se tomará como evento un paso por cero de positivo a negativo.
    direction = [-1 -1];

end
```

```
[detect, stopin, direction] = eventoChoqueTiempo(t,x)
```

```
% Función [detect, stopin, direction] = eventoChoqueTiempo(t,x)
%
% Detecta los eventos de interrupción por colisiones. La función que
% controla el evento está basada en el tiempo de impacto para el
% primer choque. Para los choques posteriores se calcula la distancia
% existente entre las masas.
%
%   detect: Devuelve 1 cuando la función pasa definida por cero
%   stopin: Si se establece como uno se interrumpirá la resolución
%   direction: Establece el criterio de paso por 0 de la función.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function [detect, stopin, direction] = eventoChoqueTiempo(t,x)

global tImpacto nChoques distChoque

    % Función que se quiere evaluar su paso por cero
    if nChoques == 0
        detect = tImpacto-t;
    else
        detect = x(1)-x(3)+distChoque;
    end

    % Se parará la ejecución al detectar el evento
    stopin = 1;

    % Se tomará como evento un paso por cero de positivo a negativo.
    direction = -1;

end
```

Exportar()

```
function varargout = Exportar(varargin)
% Función varargout = Exportar(varargin)
%
% Interfaz gráfica de usuario para la exportación de datos a archivos
% .CSV.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

% Código de instalación inicial
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Exportar_OpeningFcn, ...
                   'gui_OutputFcn',  @Exportar_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin de código de instalación inicial


% --- Executes just before Exportar is made visible.
function Exportar_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Exportar (see VARARGIN)

% Choose default command line output for Exportar
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Exportar wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Exportar_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on button press in t.
function t_Callback(hObject, eventdata, handles)
% hObject    handle to t (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of t


% --- Executes on button press in x1.
function x1_Callback(hObject, eventdata, handles)
% hObject    handle to x1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of x1


% --- Executes on button press in x2.
function x2_Callback(hObject, eventdata, handles)
% hObject    handle to x2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of x2


% --- Executes on button press in x3.
function x3_Callback(hObject, eventdata, handles)
% hObject    handle to x3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of x3


% --- Executes on button press in x4.
function x4_Callback(hObject, eventdata, handles)
% hObject    handle to x4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of x4


% --- Executes on button press in botonAceptar.
function botonAceptar_Callback(hObject, eventdata, handles)
% hObject    handle to botonAceptar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global T X ac_rueda ac_chasis

Mr = T;
```

```
% Datos relativos a la masa no suspendida
% Guarda la posición vertical
if get(handles.x1, 'Value') == 1
    Mr = horzcat(Mr, X(:,1)*1000);
end

% Guarda la velocidad vertical
if get(handles.x2, 'Value') == 1
    Mr = horzcat(Mr, X(:,2)*1000);
end

% Guarda la aceleración vertical
if get(handles.x22, 'Value') == 1
    Mr = horzcat(Mr, ac_rueda*1000);
end

% El vector de tiempo se guarda en la primera columna, los demás datos
se concatenan.
Mc = T;

% Datos relativos al chasis
% Guarda la posición vertical
if get(handles.x3, 'Value') == 1
    Mc = horzcat(Mc, X(:,3)*1000);
end

% Guarda la velocidad vertical
if get(handles.x4, 'Value') == 1
    Mc = horzcat(Mc, X(:,4)*1000);
end

% Guarda la aceleración vertical
if get(handles.x44, 'Value') == 1
    Mc = horzcat(Mc, ac_chasis*1000);
end

% Se crean los archivos .CSV
csvwrite('chasisData.csv', Mc)
csvwrite('ruedaData.csv', Mr)

msgbox(['Los datos se han exportado a la ruta: ' pwd], 'Exportar',
'help')

% --- Executes on button press in x22.
function x22_Callback(hObject, eventdata, handles)
% hObject    handle to x22 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of x22


% --- Executes on button press in x44.
function x44_Callback(hObject, eventdata, handles)
% hObject    handle to x44 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of x44
```

[t, y] = getRespuesta(rueda, choque)

```
% Función [t, y] = getRespuesta(rueda, choque)
%
% Lee los valores de los archivos .CSV en los que se han registrado
% las respuestas reales y los devuelve en dos vectores.
%   t: vector de tiempo.
%   y: vector desplazamiento vertical normalizado.
%
% El parámetro "rueda" indica cuál de las cuatro suspensiones se
% quieren obtener, si su valor es 5 devuelve la respuesta media de
% todas.
%
% Si el parámetro choque es igual a 1 se devolverán los experimentos
% realizados para choques. Si es 0 se muestran los experimentos sin
% colisión.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function [t, y] = getRespuesta(rueda, choque)


if choque == 0

    % Experimentos sin colisiones
    switch rueda
        case 1
            m = csvread('rueda1.CSV',0,3);

            t = m(:,1);
            yvol = m(:,2);
        case 2
            m = csvread('rueda2.CSV',0,3);

            t = m(:,1);
            yvol = m(:,2);
        case 3
            m = csvread('rueda3.CSV',0,3);

            t = m(:,1);
            yvol = m(:,2);
        case 4
            m = csvread('rueda4.CSV',0,3);

            t = m(:,1);
            yvol = m(:,2);
        case 5
            % Se calculan las respuestas por separado
            [t1, y1] = getRespuesta(1,0);
            [t2, y2] = getRespuesta(2,0);
            [t3, y3] = getRespuesta(3,0);
            [t4, y4] = getRespuesta(4,0);

            % Se calcula la media de todas las respuestas
            t = t1;
            y = (y1 + y2 + y3 + y4)/4;
    end
else
```

```
    % Experimentos con colisiones
    switch rueda
        case 1
            m = csvread('choque1.CSV',0,3);

            t = m(:,1);
            yvol = m(:,2);
        case 2
            m = csvread('choque2.CSV',0,3);

            t = m(:,1);
            yvol = m(:,2);
        case 3
            m = csvread('choque3.CSV',0,3);

            t = m(:,1);
            yvol = m(:,2);
        case 4
            m = csvread('choque4.CSV',0,3);

            t = m(:,1);
            yvol = m(:,2);
        case 5
            % Se calculan las respuestas por separado
            [t1, y1] = getRespuesta(1,0);
            [t2, y2] = getRespuesta(2,0);
            [t3, y3] = getRespuesta(3,0);
            [t4, y4] = getRespuesta(4,0);

            % Se calcula la media de todas las respuestas
            t = t1;
            y = (y1 + y2 + y3 + y4)/4;
    end
end

% Si no se desea mostrar la respuesta media
if rueda ~= 5
    % Conversión de voltios a centímetros corrigiendo la lógica
    inversa de la medición
    y = vol2des(yvol, 6, 9);

    [t, y] = acondicionar(y, t);

    % Si la respuesta es sin choque se normaliza la respuesta y se
     corrige la lógica inversa de la medición.
    % Si la respuesta es con colisión únicamente se se tiene en cuenta
     la lógica inversa. La normalización carece de utilidad en este
     caso.
    if choque == 0
        y = -1*y/y(1);
    else
        y = -1*y;
    end
end
end
```

u = groundInput(t, selección, boolApro)

```
% Función u = groundInput(t, seleccion, boolApro)
%
% Calcula la entrada deseada al sistema.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function u = groundInput(t, seleccion, boolApro, vectCompleto)

global m a w es t_fin

% Se quiere obtener el vector completo (para realizar un plot)
if vectCompleto == 1
    for k = 1:length(t)
        switch seleccion
            % Escalón unitario
            case 1
                if boolApro == 1 && t(k) < t_fin*0.1
                    u(k) = 0;
                else
                    u(k) = 1*es;
                end

            % Entrada rampa
            case 2
                if boolApro == 1 && t(k) < t_fin*0.1
                    u(k) = 0;
                else
                    u(k) = m*t(k);
                end

            % Entrada senoidal
            case 3
                if boolApro == 1 && t(k) < t_fin*0.1
                    u(k) = 0;
                else
                    u(k) = a*sin(2*pi*w*t(k));
                end

            % Condiciones iniciales
            case 4
                u(k) = 0;
        end
    end

% Se quiere obtener el vector por partes (para simulacion.m)
else
    % Establece el tipo de entrada
    switch seleccion
        % Escalón unitario
        case 1
            % Se quiere representar un tiempo muerto correspondiente
             al 10% del tiempo final
            if boolApro == 1 && t < t_fin*0.1
                u = 0;
            else
                u = 1*es;
```

```
            end

        % Entrada rampa
        case 2
            % Se quiere representar un tiempo muerto correspondiente
             al 10% del tiempo final
            if boolApro == 1 && t < t_fin*0.1
                u = 0;
            else
                u = m*t;
            end

        % Entrada senoidal
        case 3
            % Se quiere representar un tiempo muerto correspondiente
             al 10% del tiempo final
            if boolApro == 1 && t < t_fin*0.1
                u = 0;
            else
                u = a*sin(2*pi*w*t);
            end

        % Condiciones iniciales
        case 4
            u = 0;
    end

end

end
```

GUI()

```
function varargout = GUI(varargin)
% Función varargout = Rampa(varargin)
%
% Interfaz gráfica de usuario principal.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

% Código de instalación inicial
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin de código de instalación inicial


% --- Executes just before GUI is made visible.
function GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI (see VARARGIN)
% Choose default command line output for GUI
global seleccion completado terminar x1 x2 x3 x4 es m a w
boolAnimacion boolChoque boolMantener F
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI wait for user response (see UIRESUME)
% uiwait(handles.panelPrincipal);
axes(handles.figura1);
imshow('informacion.png')
axes(handles.figura2);
imshow('esquema.png')

% Inicialización de variables
seleccion = 1;
completado = 0;
terminar = 0;

x1=0;
x2=0;
```

```
x3=0;
x4=0;

F = 0;

es = 1;
m = 0.5;
a = 1;
w = 50;

boolAnimacion = 1;
boolChoque = 1;
boolMantener = 0;


% --- Outputs from this function are returned to the command line.
function varargout = GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in ejecutar.
function ejecutar_Callback(hObject, eventdata, handles)
% hObject    handle to ejecutar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global m_vehiculo m_rueda k2 k1 b N lon t_fin ejecutando terminar
boolApro boolExperimento cr h experimento tImpacto
boolChoqueExperimento

% Lee todos los datos introducidos
m_vehiculo = str2double(get(handles.mVehiculo,'String'));
m_rueda = str2double(get(handles.mRueda,'String'));
k1 = str2double(get(handles.coefk1,'String'));
k2 = str2double(get(handles.coefk2,'String'));
b = str2double(get(handles.coefb,'String'));
lon = str2double(get(handles.longitud,'String'));
t_fin = str2double(get(handles.textSimulacion,'String'));
N = str2double(get(handles.textIntegracion,'String'));
cr = str2double(get(handles.coefRestitucion,'String'));
tImpacto = fix(str2double(get(handles.tChoque,'String'))*100)/100;

if get(handles.choques,'Value') && get(handles.experimentos,'Value') == 1
    boolChoqueExperimento = 1;
else
    boolChoqueExperimento = 0;
end

% Lee si se mostrarán o no los experimetos
if get(handles.experimentos,'Value') == 1
    boolExperimento = 1;

    % Establece el experimento que se mostrará
    switch get(handles.popExperimentos, 'Value')
```

```
            case 1
                experimento = 1;
            case 2
                experimento = 2;
            case 3
                experimento = 3;
            case 4
                experimento = 4;
            case 5
                experimento = 5;
        end
    else
        boolExperimento = 0;
    end

    % Lee si se desea introducir un tiempo muerto
    if get(handles.aproximacion,'Value') == 1
        boolApro = 1;
    else
        boolApro = 0;
    end

    % Si se pulsa el botón mientras se está ejecutando la animación, la
    % animación se detendrá
    if ejecutando == 1
        terminar = 1;
        ejecutando = 0;
    else
        % Comprobacion en busca de fallos
        if m_vehiculo<=0 || m_rueda<=0 || k1<=0 || k2<=0 || b<=0 || lon<=0 || cr<0 || cr>1
            beep
            errordlg('Uno o varios campos son inválidos')
            uiwait
            return
        else
            % Deshabilita ciertos botones durante la animación
            set(handles.botonInformacion,'Enable','Inactive')
            set(handles.botonDefecto,'Enable','Inactive')
            set(handles.botonAvanzados,'Enable','Inactive')
            set(handles.tipoTerreno,'Enable','Inactive')
            h = waitbar(0,'Cargando, por favor espere...');

            % Ejecuta la animación
            animacionMCV
        end
    end


    % --- Executes on selection change in tipoTerreno.
    function tipoTerreno_Callback(hObject, eventdata, handles)
    % hObject    handle to tipoTerreno (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: contents = cellstr(get(hObject,'String')) returns tipoTerreno
    contents as cell array
    %        contents{get(hObject,'Value')} returns selected item from
    tipoTerreno
    global seleccion
    v = get(handles.tipoTerreno,'Value');
```

```
% Abre las opciones específicas de cada tipo de entrada
switch v
    % Seleccion escalón unitario
    case 1
        escalon
        uiwait
        seleccion = 1;

    % Selección rampa
    case 2
        Rampa
        uiwait
        seleccion = 2;

    % Selección senoidal
    case 3
        Seno
        uiwait
        seleccion = 3;

    % Selección con diciones iniciales
    case 4
        Condiciones
        uiwait
        seleccion = 4;
end

% --- Executes during object creation, after setting all properties.
function tipoTerreno_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tipoTerreno (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function integracion_Callback(hObject, eventdata, handles)
% hObject    handle to integracion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
v = get(handles.integracion,'Value');
set(handles.textIntegracion,'String',fix(v/10)*10)

% --- Executes during object creation, after setting all properties.
function integracion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to integracion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function tSimulacion_Callback(hObject, eventdata, handles)
% hObject    handle to tSimulacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
v = get(handles.tSimulacion,'Value');
set(handles.textSimulacion,'String',round(v*10)/10)

% --- Executes during object creation, after setting all properties.
function tSimulacion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tSimulacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function textIntegracion_Callback(hObject, eventdata, handles)
% hObject    handle to textIntegracion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textIntegracion as
text
%        str2double(get(hObject,'String')) returns contents of
textIntegracion as a double


% --- Executes during object creation, after setting all properties.
function textIntegracion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textIntegracion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function textSimulacion_Callback(hObject, eventdata, handles)
% hObject    handle to textSimulacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textSimulacion as
text
%        str2double(get(hObject,'String')) returns contents of
textSimulacion as a double


% --- Executes during object creation, after setting all properties.
function textSimulacion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textSimulacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function mVehiculo_Callback(hObject, eventdata, handles)
% hObject    handle to mVehiculo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of mVehiculo as text
%        str2double(get(hObject,'String')) returns contents of
mVehiculo as a double


% --- Executes during object creation, after setting all properties.
function mVehiculo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mVehiculo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function mRueda_Callback(hObject, eventdata, handles)
% hObject    handle to mRueda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of mRueda as text
%        str2double(get(hObject,'String')) returns contents of mRueda
as a double


% --- Executes during object creation, after setting all properties.
function mRueda_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mRueda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function coefk1_Callback(hObject, eventdata, handles)
% hObject    handle to coefk1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of coefk1 as text
%        str2double(get(hObject,'String')) returns contents of coefk1
as a double


% --- Executes during object creation, after setting all properties.
function coefk1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to coefk1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function coefk2_Callback(hObject, eventdata, handles)
% hObject    handle to coefk2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of coefk2 as text
%        str2double(get(hObject,'String')) returns contents of coefk2
as a double


% --- Executes during object creation, after setting all properties.
function coefk2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to coefk2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function coefb_Callback(hObject, eventdata, handles)
% hObject    handle to coefb (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of coefb as text
%        str2double(get(hObject,'String')) returns contents of coefb
as a double


% --- Executes during object creation, after setting all properties.
```

```
function coefb_CreateFcn(hObject, eventdata, handles)
% hObject    handle to coefb (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function longitud_Callback(hObject, eventdata, handles)
% hObject    handle to longitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of longitud as text
%        str2double(get(hObject,'String')) returns contents of
longitud as a double


% --- Executes during object creation, after setting all properties.
function longitud_CreateFcn(hObject, eventdata, handles)
% hObject    handle to longitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in botonInformacion.
function botonInformacion_Callback(hObject, eventdata, handles)
% hObject    handle to botonInformacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global completado

% Alterna entre mostrar los resultados o mostrar paneles de
información
if completado == 1
    axes(handles.figura2);
    imshow('esquema.png')

    axes(handles.figura1);
    imshow('informacion.png')
    set(hObject,'String','Ver resultados')
    completado = 0;
else
```

```
    axes(handles.figura1);
    imshow('grey.png')
    axes(handles.animacion1);

    axes(handles.figura2);
    imshow('grey.png')
    axes(handles.figura3);
    axes(handles.figura4);
    axes(handles.figura5);
    set(hObject,'String','Ver información')
    completado = 1;
end
% --- Executes on button press in botonAvanzados.
function botonAvanzados_Callback(hObject, eventdata, handles)
% hObject    handle to botonAvanzados (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%cargando
global h

% Abre la ventana de resultados avanzados
h = waitbar(0, 'Cargando, por favor espere...');
Avanzados
uiwait

% --- Executes on button press in botonDefecto.
function botonDefecto_Callback(hObject, eventdata, handles)
% hObject    handle to botonDefecto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Establece los valores por defecto
global seleccionExperimento

if get(handles.experimentos, 'Value') == 1

    switch seleccionExperimento
        case 1
            set(handles.coefk1, 'String', 5803600);
            set(handles.coefk2, 'String', 3780.3);
            set(handles.coefb, 'String', 93.5524);

        case 2
            set(handles.coefk1, 'String', 5803600);
            set(handles.coefk2, 'String', 3875.3);
            set(handles.coefb, 'String', 72.2650);

        case 3
            set(handles.coefk1, 'String', 5803600);
            set(handles.coefk2, 'String', 3649.7);
            set(handles.coefb, 'String', 76.9565);

        case 4
            set(handles.coefk1, 'String', 5803600);
            set(handles.coefk2, 'String', 4018.7);
            set(handles.coefb, 'String', 76.4721);

        case 5
            set(handles.coefk1, 'String', 5803600);
```

```
            set(handles.coefk2, 'String', 3670.2);
            set(handles.coefb, 'String', 72.7311);
    end
else
    set(handles.coefk1, 'String', 5803600);
    set(handles.coefk2, 'String', 4018.7);
    set(handles.coefb, 'String', 76.4721);
end

set(handles.mVehiculo,'String',41.92);
set(handles.mRueda,'String',2.62);
set(handles.longitud,'String',6.8);
set(handles.textSimulacion,'String',2);
set(handles.textIntegracion,'String',50);
set(handles.integracion,'Value',40);
set(handles.tSimulacion,'Value',2);



% --- Executes on slider movement.
function aproximacion_Callback(hObject, eventdata, handles)
% hObject    handle to aproximacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.
function aproximacion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to aproximacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function posReposo_Callback(hObject, eventdata, handles)
% hObject    handle to posReposo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of posReposo as text
%        str2double(get(hObject,'String')) returns contents of
posReposo as a double


% --- Executes during object creation, after setting all properties.
function posReposo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to posReposo (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in experimentos.
function experimentos_Callback(hObject, eventdata, handles)
% hObject    handle to experimentos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of experimentos

% Si se marca la casilla de mostrar experimentos, se hace visible el
menú
% desplegable de selección de experiemtos, el tiempo de choque se hace
% visible si se desean visualizar los experimentos con choque.
if get(hObject, 'Value') == 1
        set(handles.popExperimentos, 'Visible', 'on');

        if get(handles.choques, 'Value') == 1
            set(handles.text_tChoque, 'Visible', 'on');
            set(handles.tChoque, 'Visible', 'on');
        end
else
        set(handles.popExperimentos, 'Visible', 'off');

        set(handles.text_tChoque, 'Visible', 'off');
        set(handles.tChoque, 'Visible', 'off');

end

% --- Executes on selection change in popExperimentos.
function popExperimentos_Callback(hObject, eventdata, handles)
% hObject    handle to popExperimentos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
popExperimentos contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popExperimentos
global seleccionExperimento

seleccionExperimento = get(handles.popExperimentos,'Value');

switch seleccionExperimento
    case 1
        set(handles.coefk1, 'String', 8.9185e+05);
        set(handles.coefk2, 'String', 3780.3);
        set(handles.coefb, 'String', 93.5524);
```

```
    case 2
        set(handles.coefk1, 'String', 2.8071e+05);
        set(handles.coefk2, 'String', 3875.3);
        set(handles.coefb, 'String', 72.2650);

    case 3
        set(handles.coefk1, 'String', 1.1887e+05);
        set(handles.coefk2, 'String', 3649.7);
        set(handles.coefb, 'String', 76.9565);

    case 4
        set(handles.coefk1, 'String', 9.448e+05);
        set(handles.coefk2, 'String', 4018.7);
        set(handles.coefb, 'String', 76.4721);

    case 5
        set(handles.coefk1, 'String', 5803600);
        set(handles.coefk2, 'String', 3670.2);
        set(handles.coefb, 'String', 72.7311);
end


% --- Executes during object creation, after setting all properties.
function popExperimentos_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popExperimentos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function coefRestitucion_Callback(hObject, eventdata, handles)
% hObject    handle to coefRestitucion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of coefRestitucion as
text
%        str2double(get(hObject,'String')) returns contents of
coefRestitucion as a double


% --- Executes during object creation, after setting all properties.
function coefRestitucion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to coefRestitucion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in choques.
function choques_Callback(hObject, eventdata, handles)
% hObject    handle to choques (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of choques
global boolChoque

% Si se desea procesar los choques se muestra el coeficiente de
restitución
% para poder modificarlo, el tiempo de impacto solo se muestra si se
va a
% comparar la respuesta con los experimentos reales.
if get(hObject, 'Value') == 1
        set(handles.textChoques, 'Visible', 'on');
        set(handles.coefRestitucion, 'Visible', 'on');
        boolChoque = 1;

        if get(handles.experimentos, 'Value') == 1
            set(handles.text_tChoque, 'Visible', 'on');
            set(handles.tChoque, 'Visible', 'on');
        end

else
        set(handles.textChoques, 'Visible', 'off');
        set(handles.coefRestitucion, 'Visible', 'off');
        boolChoque = 0;

        set(handles.text_tChoque, 'Visible', 'off');
        set(handles.tChoque, 'Visible', 'off');
end


% --- Executes on button press in saltarAnimacion.
function saltarAnimacion_Callback(hObject, eventdata, handles)
% hObject    handle to saltarAnimacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of saltarAnimacion
global boolAnimacion

% Establece si se realizará o no la animación
if get(hObject, 'Value') == 1
    set(handles.integracion, 'Visible', 'off');
    set(handles.textIntegracion, 'Visible', 'off');
    set(handles.text37, 'Visible', 'off');
    boolAnimacion = 0;
else
    set(handles.integracion, 'Visible', 'on');
    set(handles.textIntegracion, 'Visible', 'on');
```

```
    set(handles.text37, 'Visible', 'on');
    boolAnimacion = 1;
end



function tChoque_Callback(hObject, eventdata, handles)
% hObject    handle to tChoque (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of tChoque as text
%        str2double(get(hObject,'String')) returns contents of tChoque
as a double


% --- Executes during object creation, after setting all properties.
function tChoque_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tChoque (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

main()

```
% Función main()
%
% Función de apertura de la aplicación mcvAndabata.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function main()
    GUI
end
```

plotMasa(posX, posY, ancho, alto)

```
% Función plotMasa(posX, posY, ancho, alto)
%
% Dibuja un rectángulo cuya base está centrada en la coordenada (posX,
% posY). Las dimensiones están dadas por 'ancho' y 'alto'. El campo
% 'color' es un vector de dimension 3 correspondiente a la designación
% de color [R G B].
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function plotMasa(posX, posY, ancho, alto, color)

x0t = [posX;posY];
x1t = x0t + [-ancho/2;0];
x2t = x0t + [-ancho/2;alto];
x3t = x0t + [ancho/2;alto];
x4t = x0t + [ancho/2;0];
fill([x1t(1) x2t(1) x3t(1) x4t(1)],[x1t(2) x2t(2) x3t(2) x4t(2)],
color/255);

end
```

plotResorte(posX, posY, ancho, longitud)

```
% Función plotResorte(posX, posY, ancho, longitud)
%
% Dibuja un resorte cuya base se encuentra en (posX, posY) y cuyas
% dimensiones geométricas son de 'ancho' y 'longitud' por cada vuelta.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga


function plotResorte(posX, posY, ancho, longitud)

x0s = [posX;posY];
x1s = x0s + [0;longitud];
x2s = x0s + [-ancho/2;3/2*longitud];
x3s = x2s + [ancho;longitud];
x4s = x3s + [-ancho;longitud];
x5s = x4s + [ancho;longitud];
x6s = x5s + [-ancho;longitud];
x7s = x6s + [ancho;longitud];
x8s = x7s + [-ancho;longitud];
x9s = x8s + [ancho/2;longitud/2];
x10s = x9s + [0;longitud];
plot([x0s(1) x1s(1) x2s(1) x3s(1) x4s(1) x5s(1) x6s(1) x7s(1) x8s(1)
     x9s(1) x10s(1)], [x0s(2) x1s(2) x2s(2) x3s(2) x4s(2) x5s(2) x6s(2)
     x7s(2) x8s(2) x9s(2) x10s(2)], 'k-','LineWidth',1.5)

end
```

plotRespuesta(rueda, choque, string)

```
% Función plotRespuesta(rueda, choque, string)
%
% Esta función permite dibujar las respuestas reales de los distintos
% experimentos.
%
% En el atributo "rueda" se especifica cuál de los experimentos se
% desea mostrar. Si toma el valor 5 se dibujará la respuesta media
% siempre y cuando no ocurra choque.
%
% El parámetro "choque" vale 1 si se desea mostrar los experimentos de
% colisiones.
%
% El parámetro string especifica el color de la representación.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function plotRespuesta(rueda, choque, string)

if nargin == 2
    string = 'b';
end

[t, y] = getRespuesta(rueda, choque);

plot(t, y, string)

text(t(1), y(1)+abs(0.2*y(1)), ['  Real (t = 0) = ' num2str(y(1)) '
     cm'])

end
```

Rampa()

```
function varargout = Rampa(varargin)
% Función varargout = Rampa(varargin)
%
% Interfaz gráfica de usuario para la toma de parámetros de entrada
% tipo rampa.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

% Código de instalación inicial
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Rampa_OpeningFcn, ...
                   'gui_OutputFcn',  @Rampa_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin de código de instalación inicial


% --- Executes just before Rampa is made visible.
function Rampa_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Rampa (see VARARGIN)

% Choose default command line output for Rampa
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Rampa wait for user response (see UIRESUME)
% uiwait(handles.panelRampa);
global m

% Reconoce el valor anterior de la pendiente "m" de la rampa
% Establece la posición del slider
set(handles.pendiente,'Value', m*100);

% Establece el valor numérico
set(handles.textPendiente,'String', m*100);

% --- Outputs from this function are returned to the command line.
function varargout = Rampa_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in botonAceptar.
function botonAceptar_Callback(hObject, eventdata, handles)
% hObject    handle to botonAceptar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global m

% Toma el nuevo valor de la pendiente "m" de la rampa
m = str2double(get(handles.textPendiente,'String'));

% Conversión a tanto por 1
m = m/100;
close Rampa

% --- Executes on slider movement.
function pendiente_Callback(hObject, eventdata, handles)
% hObject    handle to pendiente (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider

% Lee el valor del slider
v = get(handles.pendiente,'Value');

% Únicamente toma la parte entera del valor leído
set(handles.textPendiente,'String',fix(v))

% --- Executes during object creation, after setting all properties.
function pendiente_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pendiente (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function textPendiente_Callback(hObject, eventdata, handles)
% hObject    handle to textPendiente (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of textPendiente as
text
%        str2double(get(hObject,'String')) returns contents of
textPendiente as a double


% --- Executes during object creation, after setting all properties.
function textPendiente_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textPendiente (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

[T,X] = resolverChoques(cr,t_fin,x0)

```
% Función [T,X] = resolverChoques(cr,t_fin,x0)
%
% Realiza la simulación de la respuesta usando ode23 teniendo en
% cuenta los eventos que se producirán en la resolución de las
% ecuaciones diferenciales (choques).
%
%   cr    -> coeficiente de restitución.
%   t_fin -> tiempo final de simulación
%   x0    -> Condiciones iniciales
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function [T,X,nChoques] = resolverChoques(cr,t_fin,x0)

global M1 M2

% Contador de número de choques producidos
    nChoques = 0;

    % Opción para detener la resolución numérica posterior
    opts = odeset('events', @eventoChoque, 'AbsTol', 0.000001,
          'MaxStep', 0.001);

    % Inicialización del tiempo trancurrido
    T = 0;

    % Inicialización del tiempo en el que se produce el evento
    te = 0;

    % Comienza la resolución buscando eventos hasta t_fin
    while T(end) < t_fin

        % Simula la dinámica. Detiene la integración cuando detecta un
        % choque.
        [T,X,te,xe,ie] = ode23(@simulacion, [te t_fin], x0, opts);


        % Si se ha producido un choque se procesa
        if T(end) < t_fin
            % Se calculan las velocidades después de la colisión a
            partir de las velocidades inmediatamente anteriores
            v = choque([xe(4) xe(2)], [M2 M1], cr);

            % Las nuevas condiciones iniciales para retomar el cálculo
            de las ecuaciones diferenciales son las velocidades
            después del choque y las posiciones en el instante del
            evento
            x0 = [xe(1) v(2) xe(3) v(1)];

            % Primer choque que se produce
            if nChoques == 0
                % Guarda los resultados de tiempo
                Ttotal = T;
                mantener = 1;
```

```
                % Guarda los resultados de las variables de estado
                Xtotal = X;

            % Choques sucesivos al primero
            else
                % Concatena los resultados con los anteriores
                Ttotal = vertcat(Ttotal, T);
                Xtotal = vertcat(Xtotal, X);
            end


            % Cuenta los choques producidos
            nChoques=nChoques+1;
        end

    end

    % Si se han producido choques completa el intervalo final restante
    if nChoques > 0
        % Concatena los resultados con los anteriores para completar
        el último intervalo
        T = vertcat(Ttotal, T);
        X = vertcat(Xtotal, X);
    end

end
```

```
[T,X] = resolverChoquesExperimento(cr,t_fin,x0)
```

```
% Función [T,X] = resolverChoquesExperimento(cr,t_fin,x0)
%
% Realiza la simulación de la respuesta usando ode23 teniendo en
% cuenta los eventos que se producirán en la resolución de las
% ecuaciones diferenciales (choques) para el caso experimental real.
%
%   cr       -> coeficiente de restitución.
%   t_choque -> instante en el que se produce el choque.
%   t_fin    -> tiempo final de simulación.
%   x0       -> Condiciones iniciales.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function [T,X] = resolverChoquesExperimento(cr,t_fin,x0)

global M1 M2 nChoques distChoque

% Contador de número de choques producidos
    nChoques = 0;

    % Opción para detener la resolución numérica posterior
    opts = odeset('events', @eventoChoqueTiempo, 'AbsTol', 0.0001);

    % Inicialización del tiempo trancurrido
    T = 0;

    % Inicialización del tiempo en el que se produce el evento
    te = 0;

    % Comienza la resolución buscando eventos hasta t_fin
    while T(end) < t_fin

        % Simula la dinámica. Detiene la integración cuando detecta un
        % choque.
        [T,X,te,xe,ie] = ode23(@simulacion, [te t_fin], x0, opts);

        % Si se ha producido un choque se procesa
        if T(end) < t_fin
            % Cuenta los choques producidos
            nChoques=nChoques+1;

            % Se calculan las velocidades después de la colisión a
            partir de las velocidades inmediatamente anteriores
            v = choque([xe(4) xe(2)], [M2 M1], cr);

            % Las nuevas condiciones iniciales para retomar el cálculo
            de las ecuaciones diferenciales son las velocidades
            después del choque y las posiciones en el instante del
            evento
            x0 = [xe(1) v(2) xe(3) v(1)];

            % Primer choque que se produce
            if nChoques == 1
                % Guarda los resultados de tiempo
                Ttotal = T;
```

```
                % Calcula la altura a la que se ha producido el choque
                distChoque = xe(3);
                % Guarda los resultados de las variables de estado
                Xtotal = X;

            % Choques sucesivos al primero
            else
                % Concatena los resultados con los anteriores
                Ttotal = vertcat(Ttotal, T);
                Xtotal = vertcat(Xtotal, X);
            end

        end

    end

    % Si se han producido choques completa el intervalo final restante
    if nChoques > 0
        % Concatena los resultados con los anteriores para completar
        el último intervalo
        T = vertcat(Ttotal, T);
        X = vertcat(Xtotal, X);
    end

end
```

Seno()

```
function varargout = Seno(varargin)
% Función varargout = Seno(varargin)
%
% Interfaz gráfica de usuario para la toma de parámetros de entrada
% tipo senoidal.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

% Código de instalación inicial
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Seno_OpeningFcn, ...
                   'gui_OutputFcn',  @Seno_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin de código de instalación inicial


% --- Executes just before Seno is made visible.
function Seno_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Seno (see VARARGIN)

% Choose default command line output for Seno
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Seno wait for user response (see UIRESUME)
% uiwait(handles.panelSeno);
global a w

% Reconoce el valor anterior de los parámetros
% Establece la posición de los sliders
set(handles.amplitud,'Value', a);
set(handles.frecuencia,'Value', w);

% Escribe los valores numéricos
set(handles.textAmplitud,'String', a);
set(handles.textFrecuencia,'String', w);

% --- Outputs from this function are returned to the command line.
```

```
function varargout = Seno_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in botonAceptar.
function botonAceptar_Callback(hObject, eventdata, handles)
% hObject    handle to botonAceptar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a w

% Guarda los datos ingresados
a = str2double(get(handles.textAmplitud,'String'));
w = str2double(get(handles.textFrecuencia,'String'));

close Seno

% --- Executes on slider movement.
function amplitud_Callback(hObject, eventdata, handles)
% hObject    handle to amplitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider

% El valor de amplitud se toma con resolución de dos decimales
v = get(handles.amplitud,'Value');
set(handles.textAmplitud,'String',fix(v*100)/100)

% --- Executes during object creation, after setting all properties.
function amplitud_CreateFcn(hObject, eventdata, handles)
% hObject    handle to amplitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function textAmplitud_Callback(hObject, eventdata, handles)
% hObject    handle to textAmplitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of textAmplitud as
text
%        str2double(get(hObject,'String')) returns contents of
textAmplitud as a double


% --- Executes during object creation, after setting all properties.
function textAmplitud_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textAmplitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function frecuencia_Callback(hObject, eventdata, handles)
% hObject    handle to frecuencia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider

% El valor de frecuencia es entero
v = get(handles.frecuencia,'Value');
set(handles.textFrecuencia,'String',fix(v))

% --- Executes during object creation, after setting all properties.
function frecuencia_CreateFcn(hObject, eventdata, handles)
% hObject    handle to frecuencia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function textFrecuencia_Callback(hObject, eventdata, handles)
% hObject    handle to textFrecuencia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textFrecuencia as
text
```

```
%        str2double(get(hObject,'String')) returns contents of
textFrecuencia as a double


% --- Executes during object creation, after setting all properties.
function textFrecuencia_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textFrecuencia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

dx = simulación(t,x)

```
% Función dx = simulacion(t,x)
%
% Simula el sistema y devuelve los resultados de las variables de
% estado. Es llamada por ode23.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function dx = simulacion(t,x)

global a w m k1 k2 M1 M2 b seleccion t_fin boolApro es F

dx=zeros(4,1);

u = groundInput(t, seleccion, boolApro, 0);

% Conversión 'u' en metros
u=u/100;

% Ecuaciones de estado
dx(1) = x(2);
dx(2) = (b*(x(4)-x(2))+k2*x(3)-(k1+k2)*x(1)+k1*u)/M1;
dx(3) = x(4);
dx(4) = -(k2*(x(3)-x(1))+b*(x(4)-x(2))+F)/M2;

end
```

x = vol2des(vs, L, ve)

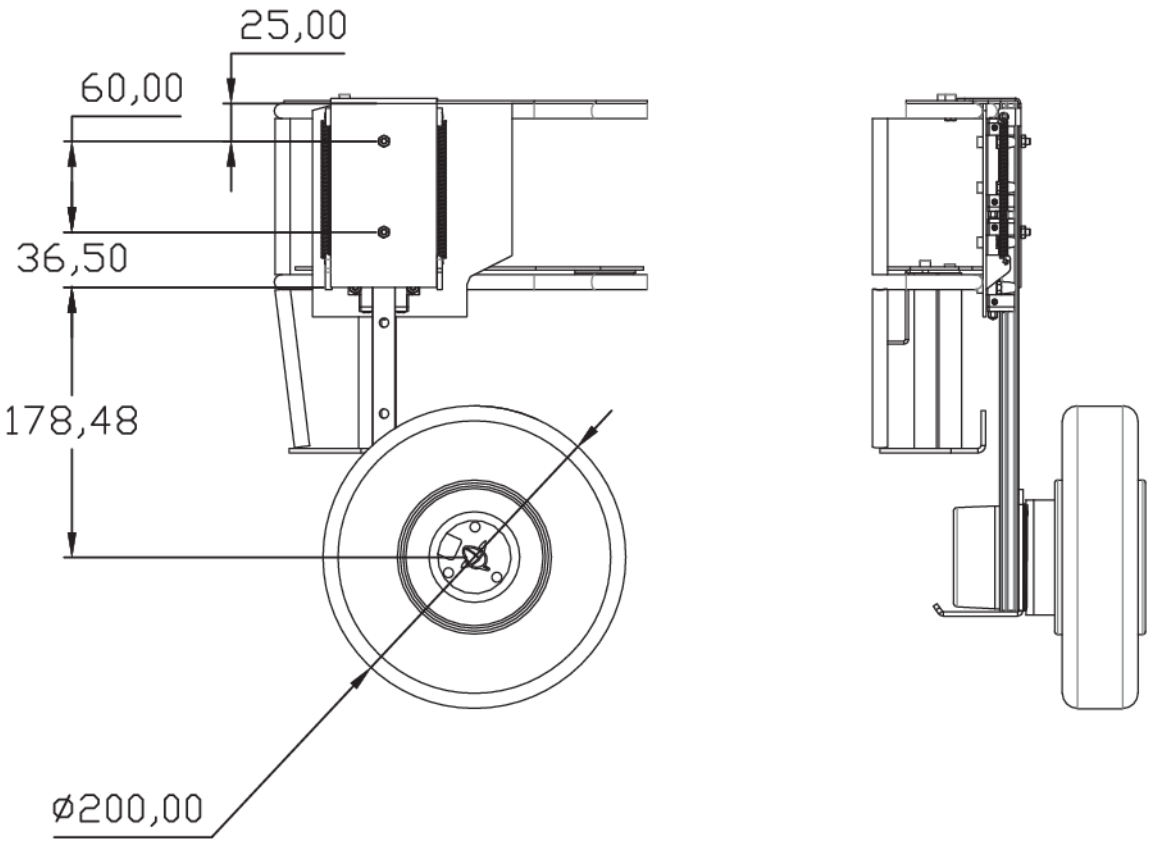
```
% Función x = vol2des(vs, L, ve)
%
% Convierte la medida del cursor en voltios 'vs' realizada sobre un
% potenciómetro de longitud 'L' alimentado con una fuente de CC cuya
% tensión es de 've'.
%
% Antonio Javier Guerrero Angulo
% E.T.S Ingeniería Industrial, Málaga

function x = vol2des(vs, L, ve)

x = (vs*L)/ve;

end
```

Anexo B. Plano del sistema de suspensión de Andábata.



Autor: Antonio Javier Guerrero Angulo	TFG de: Modelado y simulación del sistema de suspensión pasiva del robot móvil Andábata
Delineante: Ingeniería UNO	
Plano de: Suspensión de Andábata	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL — Plano nº: 1
Tutor: Jorge Luis Martínez Rodríguez (E.T.S.I.I.)	Fecha: SEPTIEMBRE 2.015
	Escala: 1:5

Índice de figuras

Índice de tablas

Bibliografía

[1] Alexandru, C. y Alexandru, P. (2011). *A comparative analysis between the vehicles' passive and active suspensions*. International Journal of Mechanics, 5 (4), pag. 371-378.

[2] Alonso, M. (2006). *Estudio del comportamiento térmico y dinámico de los amortiguadores para vehículos automóviles tipo turismo*. Tesis doctoral. Universidad Politécnica de Terrassa.

[3] Arbeláez J. J. y Marín J. P. (2007). *Modelado multicuerpo de un cuarto de vehículo liviano, utilizando el software visualnastran para ser evaluado, bajo los criterios de la norma eusama en cuanto a la adhesión*. Scientia et Technica Año XIII, No. 35 (pag. 237 - 241).

[4] Hurel, J. L. (2013). *Modelado analítico y control inteligente de un sistema de suspensión activa para un cuarto de vehículo*. Tesis doctoral. Universidad de Málaga.

[5] Karnopp, D. (2009). *How significant are transfer function relations and invariant points for a quarter car suspension model?* Vehicle System Dynamics, 47 (4), pag. 457-464.

[6] Mandow A., Martínez J.L., Morales J., Blanco J.L., García A., y González J. (2007). *Experimental Kinematics for Wheeled Skid-Steer Mobile Robots*. Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (pag. 1222-1227).

[7] Martínez, J.L. y Morales, J. (2010). *Control Aplicado con Variables de Estado*. Paraninfo.

[8] Moroe H. (2008). *Matlab para ingenieros*. Prentice Hall.

[9] Nelder J. A. y Mead R. (1965). *A simplex method for function minimization*. The Computer Journal. Vol. 7, pag 308 - 313.

[10] Ogata, K. (1980). *Ingeniería de Control Moderna*, Prentice Hall, 1ª Edición.

[11] Ogata, K. (1987). *Dinámica de Sistemas*, Prentice Hall, 1ª Edición.

[12] Pozo, A. (2014) *Instrumentación e Informática Industrial*. Cuaderno de trabajo. Universidad de Málaga.

[13] Proyecto P10-TEP-6101-R, *Navegación autónoma de un robot móvil 4x4 en entornos naturales mediante GPS diferencial y telémetro láser tridimensional*. http://www.uma.es/cms/base/ver/section/document/73618/navegacion-autonoma-de-un-robot-movil-4x4/