

E.T.S. de Ingeniería Industrial,  
Informática y de Telecomunicación

# Método de cálculo de ecuaciones del movimiento para sistemas mecánicos holónomos

Librería en Matlab y aplicación a modelo de torre grúa.



Máster Universitario en  
Ingeniería Industrial

Trabajo Fin de Máster

Vadim Coselev Condrea

Irene Miquelez Madariaga

Jorge Elso Torralba

Pamplona, 21/06/2024

upna

Universidad Pública de Navarra  
Nafarroako Unibertsitate Publikoa

Este espacio se deja intencionalmente en blanco.

# Índice general

<b>1</b>	<b>Introducción</b>	<b>2</b>
1.1	Historia	2
1.2	Estado del arte	3
1.3	Justificación y objetivos	4
<b>2</b>	<b>Cálculo de ecuaciones del movimiento</b>	<b>6</b>
2.1	Marco teórico: Cinemática	6
2.1.1	Bases, sistemas de coordenadas y referencias	6
2.1.2	Posición, velocidad y aceleración	7
2.1.3	Derivada en base fija y móvil	8
2.1.4	Matriz de producto vectorial y vector velocidad angular	9
2.1.5	Ligaduras, sistemas holónomos y coordenadas generalizadas [1]	10
2.1.6	Descomposición en coordenadas generalizadas: Vector Velocidad Lineal	11
2.1.7	Descomposición en coordenadas generalizadas: Vector Velocidad Angular	11
2.1.8	Descomposición en coordenadas generalizadas: Derivada parcial respecto a primer orden	13
2.2	Marco práctico: Cinemática	13
2.2.1	Matriz de articulación: Euler 1-2-3	13
2.2.2	Matriz de articulación: Euler 2-3	14
2.2.3	Péndulo simple: Sistema de partículas	15
2.2.4	Péndulo simple: Sólido rígido	15
2.3	Marco teórico: Dinámica	16
2.3.1	Primer Teorema Vectorial: Teorema de la “Fuerza Resultante de Inercia” [2]	16
2.3.2	Segundo Teorema Vectorial: Teorema del “Momento Resultante de Inercia” [2]	17
2.3.3	Método de los trabajos virtuales. Principio de D’Alembert [1]	18
2.3.4	Método de los trabajos virtuales. Sólido Rígido	20
2.3.5	Torsores	20
2.3.6	Método de los trabajos virtuales. Método Matricial	20
2.4	Marco práctico: Dinámica	22
2.4.1	Acción de fuerza gravitatoria	22
2.4.2	Acción momento de entrada	23
2.4.3	Acción de rozamiento de Coulomb[2]	24
2.4.4	Modelo completo péndulo simple	26
<b>3</b>	<b>Librería Dinámica: Diseño</b>	<b>28</b>
3.0.1	Otras herramientas de cálculo	28
3.1	Marco teórico: Programación orientada a objetos (POO)	28
3.1.1	Clases, objetos, propiedades y métodos	28
3.2	Marco práctico: Espacio de nombres	29
3.3	Marco práctico: Implementación en Matlab de sub-clases	30
3.3.1	Clase: Base	31
3.3.2	Clase: Point	34
3.3.3	Clase: Rigid_Body	35
3.3.4	Clase: Accion	38

3.3.5	Clase: Sistema	39
<b>4</b>	<b>Librería Dinámica: Ejemplos de uso</b>	<b>44</b>
4.1	Péndulo simple	44
4.1.1	Definición del sistema	44
4.1.2	Cálculo de magnitudes del sistema	46
4.1.3	Unificación del sistema	47
4.2	Ejemplo: Torre grúa	49
4.2.1	Cinemática: Creación del sistema cinemático	49
4.2.2	Cinemática: Componentes parciales de velocidad	52
4.2.3	Dinámica: Parametros, acciones y entradas	52
4.2.4	Unión del sistema	54
<b>5</b>	<b>Conclusiones</b>	<b>56</b>
<b>6</b>	<b>Lineas futuras</b>	<b>58</b>

# Índice de figuras

2.1	Vector posición del punto P desde el sistema de referencia con origen en O. [3]	6
2.2	Representación de dos referencias R_1 y R_2.	7
2.3	Representación de los sistemas de coordenadas de dos referencias R_1 y R_2.	7
2.4	Representación de los sistemas de coordenadas y observadores de dos referencias R_1 y R_2.	7
2.5	Derivada temporal del vector $\mathbf{u}$ en diferentes referencias. [3]	8
2.6	Ilustración de una base fija y una base móvil. [3]	8
2.7	Esquema péndulo simple y péndulo doble. [4]	10
2.8	Esquema péndulo simple.	15
2.9	Péndulo simple como sólido rígido [5]	15
2.10	Rozamiento viscoso entre sólidos $Sol_i$ y $Sol_j$ .	24
2.11	Ilustración de fluido viscoso entre el eje de rotación y la barra del péndulo.	25
2.12	Momentos de inercia de casacara de cilindro [3].	25
2.13	Ilustración del péndulo simple completo.	26
3.1	Espacio de nombres “Dynamic_Library”	30
3.2	Clase “System” del espacio de nombres “Dynamic_Library”	30
3.3	Sub-espacio de nombres “Classes”	30
3.4	Clase “Rigid_Body” del Sub-Espacio de Nombres “Classes”	30
3.5	Espacio de trabajo péndulo simple.	40
3.6	short	40
3.7	Funciones Matlab de las componentes $\mathbf{b}^i$ , $\mathbf{b}_v^i$ , $\mathbf{c}^i$ y $\mathbf{c}_v^i$ del sólido “Jib” para las tres primeras coordenadas generalizadas del sistema.	43
4.1	Momentos de inercia de paralelepípedo rectangular macizo y homogéneo [3].	44
4.2	Workspace de péndulo simple antes de “unify” simbólico.	48
4.3	Workspace de péndulo simple después de “unify” simbólico.	48
4.4	Modelo torre grúa paper. [6]	49
4.5	Centros de gravedad de los sólidos del sistema.	51
4.6	Ventana de comandos de Matlab tras intentar visualizar la componente (1,1) de la matriz [4].	55

# Índice de cuadros

3.1	Propiedades de los objetos “Barra” del pendulo doble. . . . .	29
3.2	Constructor de la clase “Base”. . . . .	32
3.3	Método para obtener información de la clase “Base” . . . . .	33
3.4	Método estático “Rotation_Matrix” para obtener las matrices rotación. . . . .	34
3.5	Constructor de la clase “Point” . . . . .	35
3.6	Método para obtener información de la clase “Point” . . . . .	35
3.7	Constructor de la clase “Rigid_Body” . . . . .	36
3.8	Método para obtener información de la clase “Rigid_Body” . . . . .	37
3.9	Constructor de la clase “Action” . . . . .	38
3.10	Método para obtener información de la clase “Action” . . . . .	39
4.1	Velocidades parciales del sólido rígido “Jib” . . . . .	52
4.2	Velocidades parciales del sólido rígido “Trolley” . . . . .	52
4.3	Velocidades parciales del sólido rígido “Bar” . . . . .	52
4.4	Tensores de inercia sólidos rígidos . . . . .	53

# Índice de Códigos

3.1	Propiedades clase “Base” . . . . .	31
3.2	Programación de la clase “Base” . . . . .	32
3.3	Programación del método “Get_Info” de “Base” . . . . .	33
3.4	Programación del método estático de matriz de rotación. . . . .	34
3.5	Propiedades clase “Point” . . . . .	34
3.6	Propiedades clase “Rigid_Body” . . . . .	36
3.7	Propiedades clase “Action” . . . . .	38
3.8	Creación manual de sistema. . . . .	39
3.9	Creación del objeto de sistema “Pendulo_Simple” . . . . .	40
3.10	Creación de la coordenada generalizada $\theta(t)$ en el péndulo simple. . . . .	40
3.11	Creación de bases en sistema péndulo simple. . . . .	41
3.12	Función que genera un punto a través de un sistema de coordenadas. . . . .	41
3.13	Estructura de programación de los subsistemas de la clase “System”. . . . .	41
3.14	Generación de funciones Matlab para el cálculo de las componentes de las parciales $\mathbf{b}_i$ y $\mathbf{c}_i$ de las velocidades. . . . .	42
4.1	Inicialización parámetros péndulo simple. . . . .	44
4.2	Inicialización del sistema del péndulo simple. . . . .	45
4.3	Creación de base asociada al sólido Bar y el centro de masas $\mathbf{G}$ . . . . .	45
4.4	Creación del sólido rígido “Bar” asociado a la base “Bar”. . . . .	45
4.5	Creación de acción gravitatoria para el sólido “Bar”. . . . .	46
4.6	Creación de momento de entrada para el sólido “Bar”. . . . .	46
4.7	Creación de rozamiento de Coulomb para el sólido “Bar”. . . . .	46
4.8	Función de vector posición para el vector $\mathbf{NG}$ . . . . .	46
4.9	Función de vector velocidad para un punto $\mathbf{G}$ . . . . .	46
4.10	Función de vector aceleración para un punto $\mathbf{G}$ . . . . .	47
4.11	Función para el cálculo de matrices de cambio de base. . . . .	47
4.12	Función para el cálculo de la velocidad angular de una base respecto a otra. . . . .	47
4.13	Función que unifica el sistema simbólicamente. . . . .	48
4.14	Matriz $\mathbf{A}$ y vector $\mathbf{b}$ en péndulo simple. . . . .	48
4.15	Inicialización de sistema mecánico en Matlab. . . . .	49
4.16	Creación bases del modelo de torre grúa. . . . .	50
4.17	Introducción de coordenadas generalizadas $\theta, \beta_1, \beta_2, \gamma$ . . . . .	51
4.18	Introducción de sistemas de coordenadas en el sistema. . . . .	51
4.19	Introducción de coordenada generalizada $r(t)$ . . . . .	51
4.20	Masa y tensor de inercia simbólico de “Trolley”. . . . .	53
4.21	Creación del “Rigid_Body” “Trolley” en el sistema. . . . .	53
4.22	Código que añade acción gravitatoria al sólido “Jib”. . . . .	54
4.23	Código que añade acción $M_{in}$ . . . . .	54
4.24	Código que añade acción $M_{in}$ . . . . .	54
4.25	Código para invertir la matriz $[\mathbf{A}]$ . . . . .	54

## Abstract

This final project aims to address the dynamic modeling of rigid body systems. The context focuses on the need to obtain a model of a tower crane to later implement the controllers required by the user. This system, in a real environment, presents a large number of degrees of freedom and complex physical modeling forces. For this reason, a methodology is proposed to obtain the equations of motion for holonomic systems and the development of a tool in Matlab capable of performing the symbolic calculation of these equations. Finally, it is concluded that, although symbolic calculation is possible, the computational cost is very high for systems with many degrees of freedom, and it is necessary to rely on numerical linearizations to implement linear control methods or conduct studies of nonlinear control.

**Keyword**— partial velocity, , generalized force, inverse

## Resumen

Este trabajo final de estudios tiene como objetivo abordar el modelado dinámico de sistemas de sólidos rígidos. El contexto se centra en la necesidad de obtener un modelo de una torre grúa de construcción para, posteriormente, implementar los controladores requeridos por el usuario. Este sistema, en un entorno real, presenta un gran número de grados de libertad y fuerzas de modelado físico complejo. Por esta razón, se propone una metodología para obtener las ecuaciones del movimiento de sistemas holónomos y el desarrollo de una herramienta en Matlab capaz de realizar el cálculo simbólico de estas ecuaciones. Finalmente, se concluye que, aunque el cálculo simbólico es posible, el costo computacional es muy elevado para sistemas con muchos grados de libertad, y es necesario recurrir a linealizaciones numéricas para implementar métodos de control lineal o realizar estudios de control no lineal.

**Palabras clave**— velocidad parcial, , fuerza generalizada, inversa



# 1 Introducción

Muchas veces, a la hora de comenzar un proyecto, las características del sistema no están bien definidas en ese instante, es el tiempo el que hace que se vayan definiendo poco a poco las particularidades que hace falta estudiar. Mirando este concepto desde otro punto de vista, se puede plantear el caso de una empresa que se dedica a diseñar controladores para torres grúa de construcción, tanto para su movimiento como para la reducción de cargas. Debido a que cada fabricante diseña la torre grúa a su manera, las características son diferentes en cada grúa para la cual la empresa tiene que diseñar un controlador. En ambos casos, aunque parezcan diferentes, el desempeño de la empresa, radica en la capacidad que tiene esta para ajustarse lo más rápido posible a las propiedades particulares de un problema. En el primer caso, se deberá realizar un trabajo teniendo en cuenta que hay que facilitar la modificación de parámetros en momentos posteriores, mientras que en el segundo, cada grúa nueva es un nuevo diseño y, por lo tanto, unos nuevos parámetros.

El contexto de esta memoria nace en la necesidad de diseñar controladores para torres grúa de construcción, pero, teniendo en cuenta que los diseños de estas son variados o puede que incluso todavía no exista un diseño final de la grúa para la cual se estudia un controlador. Esta parte de la teoría de control que estudia el sistema físico se denomina modelado y, como se ha hecho ver con el ejemplo ilustrativo anterior, es importante encontrar una metodología que haga más fácil ajustarse a cambios en los modelos, o incluso facilitar crear modelos nuevos lo más eficientemente posible.

El modelado de sólidos mecánicos es una disciplina fundamental en la ingeniería y la física que ha evolucionado significativamente a lo largo de los siglos. Desde las formulaciones iniciales de las leyes del movimiento por Newton hasta las técnicas modernas de simulación y análisis, el campo ha visto avances que han mejorado enormemente la capacidad para predecir y optimizar el comportamiento de sistemas mecánicos complejos. Esta revisión explora tanto la historia del modelado de sólidos mecánicos como el estado del arte actual, proporcionando una visión comprensiva de los desarrollos clave y las metodologías empleadas hoy en día.

## 1.1 Historia

### Isaac Newton (1643-1727)

Isaac Newton sentó las bases de la mecánica clásica con la publicación de “*Philosophiae Naturalis Principia Mathematica*” en 1687. Sus leyes del movimiento y la ley de la gravitación universal proporcionaron un marco fundamental para el análisis de sistemas mecánicos. Las tres leyes del movimiento de Newton describen cómo las fuerzas afectan el movimiento de un objeto, estableciendo la relación entre la fuerza, la masa y la aceleración.

### Jean le Rond d’Alembert (1717-1783)

Jean le Rond d’Alembert, un matemático y físico francés, contribuyó significativamente a la mecánica clásica con el principio que lleva su nombre. El principio de d’Alembert reformula la segunda ley de Newton al considerar las fuerzas de inercia. Este principio se expresa como  $F - ma = 0$ , donde  $F$  es la fuerza aplicada,  $m$  es la masa, y  $a$  es la aceleración.

### Leonhard Euler (1707-1783)

Leonhard Euler hizo contribuciones fundamentales a la mecánica, incluida la formulación de las ecuaciones de movimiento para sistemas de partículas y cuerpos rígidos. Las ecuaciones de Euler describen

el movimiento rotacional de un cuerpo rígido en términos de sus momentos de inercia y momentos externos. Euler también introdujo la idea de coordenadas generalizadas, que son fundamentales en la mecánica analítica.

### Joseph-Louis Lagrange (1736-1813)

Joseph-Louis Lagrange desarrolló la mecánica lagrangiana, que reformula las ecuaciones de movimiento de Newton y Euler en términos de coordenadas generalizadas y energía. La ecuación de Lagrange se expresa como

$$\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = 0, \quad (1.1)$$

donde  $L$  es el lagrangiano (la diferencia entre la energía cinética y la energía potencial), y  $q_i$  son las coordenadas generalizadas. Esta formulación es extremadamente poderosa para sistemas complejos y con restricciones.

### Thomas R. Kane (1924-2019)

Thomas R. Kane introdujo un método alternativo para derivar las ecuaciones de movimiento de sistemas dinámicos, conocido como el método de Kane. A diferencia de la formulación de Lagrange, el método de Kane utiliza velocidades generalizadas y evita la necesidad de calcular derivadas de energía potencial y cinética. Las ecuaciones de movimiento se derivan a partir de fuerzas y momentos aplicados y de las aceleraciones generalizadas

$$\sum F_r \cdot \frac{\partial \mathbf{v}_i}{\partial \dot{q}_j} + \sum M_r \cdot \frac{\partial \boldsymbol{\omega}_i}{\partial \dot{q}_j} = Q_j, \quad (1.2)$$

donde  $F_r$  y  $M_r$  son las fuerzas y momentos de inercia aplicados,  $\mathbf{v}_i$  y  $\boldsymbol{\omega}_i$  son las velocidades lineales y angulares, y  $Q_j$  son las fuerzas generalizadas. Este enfoque es eficiente para sistemas multicuerpo complejos y facilita la implementación computacional.

## 1.2 Estado del arte

El modelado de sólidos mecánicos ha avanzado significativamente con el tiempo, incorporando una variedad de técnicas numéricas y computacionales. A partir de la década de 1950, el Método de los Elementos Finitos (MEF) se estableció como una técnica central para el análisis de estructuras complejas. Desde entonces, la combinación de modelos de sólidos rígidos y flexibles se ha vuelto esencial para capturar tanto deformaciones como vibraciones estructurales en sistemas mecánicos complejos. En las últimas décadas, técnicas de reducción de orden como la Descomposición en Valores Singulares (SVD) y el Análisis de Componentes Principales (PCA) han permitido simplificar modelos sin una pérdida significativa de precisión. Hoy en día, herramientas avanzadas como ANSYS, Abaqus, COMSOL Multiphysics, MATLAB/Simulink y OpenModelica son cruciales para el análisis y optimización de sistemas mecánicos en diversas aplicaciones industriales.

### Métodos de Elementos Finitos (MEF)

El Método de los Elementos Finitos (MEF) es una técnica numérica clave en el análisis de sólidos mecánicos. El MEF consiste en discretizar un sólido continuo en una malla de elementos finitos más pequeños, permitiendo la resolución de ecuaciones de movimiento y equilibrio para cada elemento. Este método facilita el análisis detallado de estructuras complejas bajo diversas condiciones de carga y geometría.

- Aplicaciones: MEF se utiliza ampliamente en la ingeniería civil para el diseño de edificios y puentes, en la automoción para el análisis de componentes estructurales, y en la aeroespacial para el diseño de aeronaves y cohetes.
- Ventajas: Permite un análisis preciso y detallado de la distribución de tensiones, deformaciones y otros parámetros mecánicos críticos.

- Desventajas: Requiere recursos computacionales significativos, especialmente para modelos con un alto número de elementos finitos.

## Modelado de Sólidos Rígidos y Flexibles

En la práctica moderna, la combinación de modelos de sólidos rígidos y flexibles es común para representar sistemas mecánicos complejos. Las componentes del sistema en las que no se vayan a producir deformaciones apreciables se aproximan a través de un modelo de sólido rígido. Por otro lado, los sólidos flexibles permiten capturar deformaciones y vibraciones lo cual es esencial en partes del sistema que cambian su forma significativamente ante determinadas cargas.

- Aplicaciones: Este enfoque híbrido se emplea en la dinámica de vehículos, robots industriales, y sistemas biomecánicos.
- Ventajas: Proporciona un equilibrio entre precisión y eficiencia computacional, permitiendo un análisis detallado donde sea necesario sin una carga computacional excesiva.
- Desventajas: La complejidad de combinar estos modelos puede incrementar la dificultad de la modelización y simulación.

## Reducción de Orden

La reducción de orden es una técnica que simplifica modelos complejos manteniendo su precisión. Dos métodos comunes son la Descomposición en Valores Singulares (SVD) y el Análisis de Componentes Principales (PCA).

### Descomposición en Valores Singulares (SVD)

- Proceso: Descompone una matriz en tres matrices  $(U, \Sigma, V)$  para identificar y truncar las componentes menos significativas.
- Aplicaciones: Utilizada en el análisis modal, simulaciones en tiempo real y optimización de control.
- Ventajas: Reduce el número de ecuaciones sin pérdida significativa de precisión, mejorando la eficiencia computacional.
- Desventajas: Puede requerir un conocimiento profundo de álgebra lineal avanzada.

### Análisis de Componentes Principales (PCA)

- Proceso: Reduce la dimensionalidad de un conjunto de datos transformándolo a un nuevo conjunto de variables (componentes principales) que capturan la mayor parte de la variabilidad del sistema.
- Aplicaciones: Utilizada en el análisis de datos, compresión de datos y modelos predictivos.
- Ventajas: Simplifica los modelos sin pérdida significativa de información crítica.
- Desventajas: Puede no capturar bien las no linealidades del sistema.

## 1.3 Justificación y objetivos

La metodología presentada en esta memoria nace de la necesidad de obtener las ecuaciones del movimiento para sistemas mecánicos de una forma sistemática. Además, dado que el contexto de la memoria es el diseño y la validación de controladores, es esencial que la implementación de métodos de control sea lo más ágil posible. En las técnicas lineales de control tradicional, se suelen utilizar modelos linealizados en torno a un punto de operación. Cuanto más se facilite esta linealización, existirá más tiempo para invertir en el diseño de controladores.

La era moderna permite desarrollar herramientas que faciliten el cálculo con esta metodología. Hoy en día, lenguajes de programación como Python o software como MATLAB implementan cálculos

simbólicos que simplifican la obtención de las ecuaciones del movimiento con un ordenador. Esto tiene la ventaja adicional de que la modificación del modelo o la corrección de errores pueden realizarse con relativa facilidad. Estas razones subrayan la necesidad de desarrollar metodologías y herramientas que cumplan con los requisitos mencionados anteriormente.

## Objetivos

- Obtener una metodología sistemática de cálculo de las ecuaciones del movimiento de sistemas mecánicos,
  - esta se presentará en forma de álgebra lineal, debido a que facilita implementar métodos de diseño de controladores y
  - la presentación de las ecuaciones debe ser clara y sencilla.
- Crear una herramienta capaz de calcular las ecuaciones del movimiento de sistemas mecánicos,
  - en la cual la modificación del modelo debe realizarse de forma rápida y sencilla y
  - los cálculos deben ser simbólicos para una posterior linealización.

Para conseguir los objetivos propuestos se realizará un estudio teórico de la metodología a aplicar y posteriormente utilizarán herramientas computacionales para agilizar los cálculos.

## 2 Cálculo de ecuaciones del movimiento

En este apartado se pretende dar una visión general del cálculo de las ecuaciones del movimiento de los sólidos rígidos. Debido a que la idea principal del trabajo expuesto es la programación de una librería que realice el cálculo de estas ecuaciones, se deben dar definiciones precisas de los elementos que componen la mecánica que hagan más sencilla la programación posterior.

### 2.1 Marco teórico: Cinemática

La cinemática es el estudio del movimiento de los cuerpos. En este no se tiene en cuenta el origen físico del movimiento sino el movimiento en sí.

#### 2.1.1 Bases, sistemas de coordenadas y referencias

Las bases y sistemas de coordenadas son conceptos matemáticos, concretamente del álgebra lineal, que ayudan a localizar un cuerpo en el espacio tridimensional. Las referencias son nociones definidas en mecánica que contienen ideas físicas del movimiento. La relación entre las bases, los sistemas de coordenadas y las referencias es el primer paso para el estudio matemático de la mecánica de sólidos rígidos.

##### Bases

Una base es un conjunto de tres vectores en el espacio. En la mecánica de sólidos rígidos se hace uso de las bases ortonormales. En la Figura 2.1 se muestra la base formada por los vectores  $[e_1, e_2, e_3]$ .

##### Sistemas de Coordenadas

Se definirá un sistema de coordenadas como el conjunto de un punto y una base. La Figura 2.1 el conjunto del Punto “O” y la base formada por los vectores  $[e_1, e_2, e_3]$  es un sistema de coordenadas.

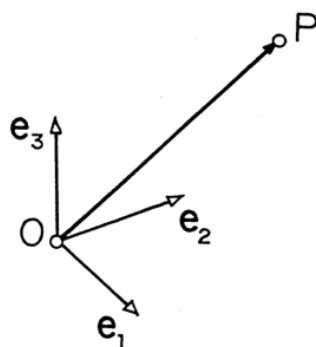


Figura 2.1: Vector posición del punto P desde el sistema de referencia con origen en O. [3]

##### Referencia

La existencia de sistemas de coordenadas con movimiento relativos entre ellos conduce a las referencias. Se define como referencia *el espacio de puntos fijos respecto a un sistema de coordenadas y por lo tanto*

fijos entre sí [3]. Se puede entender la referencia como el conjunto de sistemas de coordenadas entre los que no existe movimiento, por lo tanto, a efectos del estudio del movimiento de un punto o una partícula cualquiera de los sistemas de coordenadas serían válidos para el cálculo.

El sólido rígido es un buen ejemplo de referencia debido a que por definición sus puntos se mantienen fijos entre sí, sin importar en que punto se coloque el sistema de coordenadas, el movimiento estudiado será análogo.

La Figura 2.2 muestra dos referencias distintas definidas como el conjunto de puntos fijos entre cada una de ellas. Agulló [3] menciona que habitualmente estas referencias se representan a través de un sistema de coordenadas, es decir, que aunque la física del movimiento se estudia para las referencias, en la práctica se suele simplificar esta idea representando únicamente un sistema de coordenadas asociado a dicha referencia como muestra la Figura 2.3. Por último se añade la idea de observador que observa el movimiento relativo a su referencia representado por la Figura 2.4. Este observador en gran parte viene dado con un sistema de coordenadas.

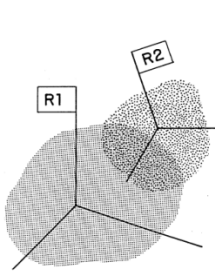


Figura 2.2: Representación de dos referencias  $R_1$  y  $R_2$ .

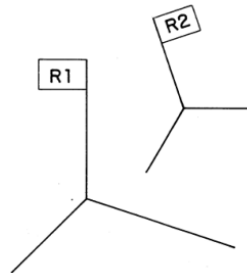


Figura 2.3: Representación de los sistemas de coordenadas de dos referencias  $R_1$  y  $R_2$ .

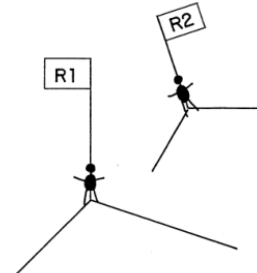


Figura 2.4: Representación de los sistemas de coordenadas y observadores de dos referencias  $R_1$  y  $R_2$ .

En la programación de la librería no se tendrán en cuenta las referencias sino los sistemas de coordenadas. El usuario debe conocer estos conceptos debido a que la implementación de una referencia por su definición sería muy laborioso.

### 2.1.2 Posición, velocidad y aceleración

En este apartado se realizará una breve introducción a los conceptos cinemáticos de posición, velocidad y aceleración.

#### Vector de posición

La posición de un punto o una partícula viene dada por un vector posición desde un sistema de coordenadas. Este vector  $\mathbf{r}$  se representa por las componentes en la base del sistema de coordenadas como

$$\mathbf{r} = \sum_{i=1}^3 a_i \mathbf{e}_i = \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix}_{123}, \quad (2.1)$$

donde  $a_i$  son las componentes del vector en la base que componen los vectores unitarios  $\mathbf{e}_i$ . Un vector de este tipo es un vector libre debido a que no se detalla su origen dentro de la definición. Para que sea un vector fijo el origen del vector se definirá como el origen del sistema de coordenadas. La figura 2.1 representa un esquema del vector posición  $\mathbf{OP}$ . Este vector tiene origen en O y se define a través de los vectores unitarios  $[\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3]$ .

#### Vector de velocidad

La velocidad de un punto o una partícula viene dada por la variación del vector posición en el tiempo según

$$\mathbf{v} = \frac{d\mathbf{r}}{dt}. \quad (2.2)$$

## Vector aceleración

La aceleración de un punto o una partícula viene dada por la variación del vector de velocidad en el tiempo como

$$\mathbf{a} = \frac{d\mathbf{v}}{dt} = \frac{d^2\mathbf{r}}{dt^2}. \quad (2.3)$$

### 2.1.3 Derivada en base fija y móvil

Dadas las definiciones de la posición, velocidad y aceleración, se debe resaltar que la derivada temporal de los vectores será diferente dependiendo de la referencia. Como ya se ha comentado, las referencias se pueden ver como un conjunto de sistemas de coordenadas por lo tanto la explicación se hará sobre estos mismos. La derivada temporal de un vector en un sistema de coordenadas puede ser diferente de otro. Este concepto se muestra en la Figura 2.5 en la que uno de los observadores calcula una derivada nula y el otro una no nula.

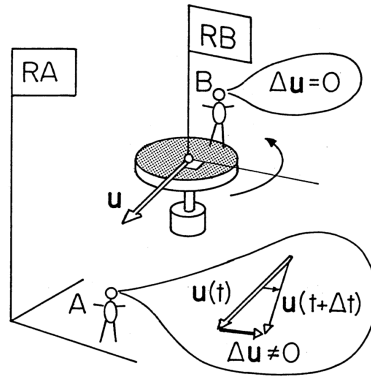


Figura 2.5: Derivada temporal del vector  $\mathbf{u}$  en diferentes referencias. [3]

A través de la definición de  $\mathbf{r}$  dada en la Ecuación 2.1 se pueden distinguir dos tipos de casos:

- Base del vector fija a la referencia.
- Base del vector móvil respecto a la referencia.

Estos dos casos se encuentran ilustrados en la Figura 2.6 en la que se entiende que la base “B” se mueve junto a la referencia “R2” y, por lo tanto, para esta referencia, la base se encuentra fija. Por otro lado para la referencia “R1” la base “B” se encuentra en movimiento y, por lo tanto, es una base móvil.

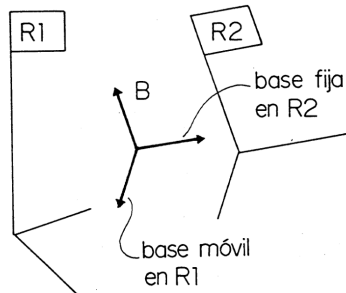


Figura 2.6: Ilustración de una base fija y una base móvil. [3]

Sea un vector  $\mathbf{u}$

$$\mathbf{u} = \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix}_{BF}, \quad (2.4)$$

se definen las derivadas en diferentes bases como:

## Base fija

En una base vectorial de orientación fija “BF” a la referencia “R”, las componentes del vector derivada temporal se obtienen simplemente por derivación temporal de las componentes del vector [3],

$$\left[ \left\{ \frac{d}{dt} \mathbf{u} \right\}_R \right]_{BF} = \left\{ \begin{matrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \end{matrix} \right\}_{BF} . \quad (2.5)$$

## Base móvil

En una base vectorial de orientación móvil “BM” respecto a la referencia “R”, en la cual existe una base fija “BF”, las componentes del vector derivada temporal se obtienen según

$$\left[ \left\{ \frac{d}{dt} \mathbf{u} \right\}_R \right]_{BM} = \frac{d}{dt} \{ \mathbf{u} \}_{BM} + [\mathbf{S}]^T [\dot{\mathbf{S}}] \{ \mathbf{u} \}_{BM} , \quad (2.6)$$

donde:

$[\mathbf{S}]$  = Cambio de base BM a BF.

= Las columnas de la matriz son los componentes en la base fija de los vectores de la base móvil.

En la librería todos los vectores serán calculados a través de una base fija a la referencia inercial. Esto es más sencillo de implementar y tiene la ventaja de no requerir dos métodos de diferenciación.

### 2.1.4 Matriz de producto vectorial y vector velocidad angular

Si bien la Ecuación 2.6 es correcta, generalmente en la literatura no se presenta de esta forma debido a que existe una manera más compacta de representar el producto  $[\mathbf{S}]^T [\dot{\mathbf{S}}]$ . Este producto tiene la peculiaridad de ser una matriz antisimétrica (*Skew-symmetric matrix* [7]) de la forma

$$[\mathbf{S}]^T [\dot{\mathbf{S}}] = \begin{bmatrix} 0 & -\Omega_3 & \Omega_2 \\ \Omega_3 & 0 & -\Omega_1 \\ -\Omega_2 & \Omega_1 & 0 \end{bmatrix} = [\boldsymbol{\Omega}]^\times . \quad (2.7)$$

Se puede ver que la matriz esta compuesta unicamente por 3 componentes :  $\Omega_1, \Omega_2$ , y  $\Omega_3$ . Estas tres componentes se definen comunmente como el “Vector de Velocidad Angular de la base móvil BM respecto a la referencia R” según

$$\{ \boldsymbol{\Omega}_R(BM) \}_{BM} = \{ \boldsymbol{\Omega}_{BF}(BM) \}_{BM} = \left\{ \begin{matrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \end{matrix} \right\}_{BM} . \quad (2.8)$$

La definición de este vector es interesante debido a que la matriz  $[\mathbf{S}]^T [\dot{\mathbf{S}}]$  es la operación de “Producto Vectorial de  $\boldsymbol{\Omega}_R(BM)$ ” y, por lo tanto, la Ecuación 2.6 se puede reescribir como

$$\left[ \left\{ \frac{d}{dt} \mathbf{u} \right\}_R \right]_{BM} = \frac{d}{dt} \{ \mathbf{u} \}_{BM} + \{ \boldsymbol{\Omega}_R(BM) \}_{BM} \times \{ \mathbf{u} \}_{BM} . \quad (2.9)$$

Una mención que se debe hacer es que el “Vector de Velocidad Angular” no se deriva de ningun otro vector como si lo hace la velocidad lineal. Esto es debido a que estas matrices y vectores pertenecen a un grupo específico denominado “Grupo Especial Rotación ” o “*Special Orthogonal Group*  $SO(3)$ ” [8]. Estos grupos pertenecen al estudio de los denominados “Grupos de Lie” de los cuales existen estructuras algebraicas definidas sobre un espacio vectorial denominadas “álgebra de Lie”.



Debido a que en la programación de la librería se hará uso de las bases y no tanto de las referencias se propone también la definición de este vector como “Vector de Velocidad Angular de la base móvil BM respecto a la base fija BF”. En la Ecuación 2.8 ya se ha introducido este concepto. Aunque, como se ha comentado anteriormente, no se va a hacer uso de la derivada en base móvil, el “Vector de Velocidad Angular” define el movimiento de rotación de los sólidos rígidos y por lo tanto se convierte en obligatorio su cálculo para obtener las ecuaciones del movimiento.

### 2.1.5 Ligaduras, sistemas holónomos y coordenadas generalizadas [1]

Hasta el momento se ha presentado el movimiento de la partícula en un sistema puramente cartesiano a través de 3 variables independientes una de otra  $\mathbf{r} = (x, y, z)$ . Con esta definición no se pueden estudiar muchos de los sistemas físicos que se pretende en esta memoria. A modo de ejemplo, considere dos partículas que pertenecen a un sólido rígido, si cada una de ellas tiene tres variables independientes  $\mathbf{r}_i = (x_i, y_i, z_i)$  y  $\mathbf{r}_j = (x_j, y_j, z_j)$ , no existe una relación que fuerce que ambas se mantengan a la misma distancia. Para representar de una forma real el sistema anterior se debe incluir la ecuación  $(\mathbf{r}_i - \mathbf{r}_j)^2 - c_{ij}^2 = 0$  [1] que introduce una relación entre las variables independientes  $x_i, y_i, z_i, x_j, y_j, z_j$ . Este tipo de ecuaciones que introducen relaciones cinemáticas entre partículas se denominan ligaduras o restricciones. En el péndulo simple de la figura 2.7 se puede observar que la posición de la partícula no es independiente debido a que su movimiento está restringido por la longitud de la barra ( $x^2 + y^2 - l_1^2 = 0$ ) y por el movimiento únicamente en el plano  $x - y$  ( $z = 0$ ) [4]. Estas dos restricciones hacen que el sistema que en un principio tenía 3 variables independientes acabe teniendo 1 variable independiente. Este concepto se puede pensar en términos de los grados de libertad que tiene el sistema, al principio se podía mover en las tres dimensiones mientras que con las ligaduras ya solo lo hace en el plano y alrededor de un eje. Para el caso del doble péndulo se puede realizar un estudio análogo en el que la posición de cada una de las partículas viene dada por  $\mathbf{r}_1 = (x_1, y_1, z_1)$  y  $\mathbf{r}_2 = (x_2, y_2, z_2)$ . En este caso el número de variables independientes es de 6 y el número de restricciones son 4 y, por lo tanto, el número de variables independientes es 2. De igual forma que el péndulo simple los grados de libertad del doble péndulo se ha reducido a dos que representa el giro de las barras alrededor de sus respectivos ejes.

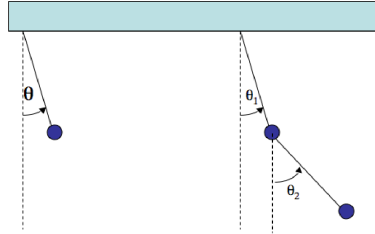


Figura 2.7: Esquema péndulo simple y péndulo doble. [4]

Este tipo de ligaduras que disminuyen el número de variables independientes del sistema se denominan ligaduras holónomas y tienen la forma

$$f(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \dots, t) = 0. \quad (2.10)$$

Para el caso general de un sistema de  $N$  partículas libres de ligaduras el número de variables independientes es  $3N$ . Si a este sistema se le introducen  $k$  ligaduras holónomas el número final de variables independientes o grados de libertad es de  $3N - k$ .

Aun hasta este punto, lo propuesto se basa en el uso de las coordenadas cartesianas y las variables independientes. Goldstein [1] muestra que otra forma de expresar esta eliminación de coordenadas es introduciendo nuevas  $3N - k$  variables independientes  $q_1, q_2, \dots, q_{3N-k}$  con las cuales expresar  $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N$ . Para el caso del péndulo simple  $3N - k = 1$  y por lo tanto se introduce una nueva variable independiente  $q_1$ . Esta variable se escogerá de tal forma que sea el ángulo de rotación del péndulo  $\theta$ . De esta forma la ligadura viene implícita dentro del propio vector de posición  $\mathbf{r} = (l \sin \theta, -l \cos \theta, 0)$ . El caso general de  $N$  partículas es

$$\begin{aligned}\mathbf{r} &= \mathbf{r}_1(q_1, q_2, \dots, q_{3N-k}, t), \\ \mathbf{r}_N &= \mathbf{r}_N(q_1, q_2, \dots, q_{3N-k}, t).\end{aligned}\tag{2.11}$$

Se debe tener en cuenta que se puede fijar cualquier tipo de manitud para la nueva variable independiente. Se podría haber escogido de tal forma que tuviese unidades de energía o de momento cinético, según convenga.

El estudio de ligaduras no holónomas es más complejo debido a que no existe una fórmula general de abordar este tipo de problemas. Existen casos en los que si la ligadura no es integrable se puede introducir las ecuaciones diferenciales de la ligadura junto con las del movimiento y, eliminar las ecuaciones dependientes a través de multiplicadores de Lagrange [1]. Pero, este tipo de ligaduras solo son un caso particular de los muchos que pueden existir a la hora de estudiar ligaduras no holónomas, siendo necesario un estudio individual en cada uno de ellos. En lo concierne a esta memoria, una de las simplificaciones que se realiza es únicamente considerar sistemas mecánicos holónomos, los cuales son más sencillos de estudiar e implementar.

### 2.1.6 Descomposición en coordenadas generalizadas: Vector Velocidad Lineal

Un resultado que hará falta más adelante es la descomposición del vector de velocidad lineal  $\mathbf{v}$  en las coordenadas generalizadas o el cálculo de las parciales de la velocidad.

Para un sistema de partículas definido por los vectores de posición de la ecuación 2.11 el vector velocidad puede ser obtenido a través de la regla de la cadena. La velocidad para la partícula  $i$

$$\mathbf{v}_i = \frac{d}{dt} \mathbf{r}_i = \left[ \frac{d\mathbf{r}_i}{dt} \right]_{\forall q_k = f_{ijo}} + \sum_k \frac{\partial \mathbf{r}_i}{\partial q_k} \frac{dq_k}{dt} = \frac{\partial \mathbf{r}_i}{\partial t} + \sum_k \frac{\partial \mathbf{r}_i}{\partial q_k} \dot{q}_k,\tag{2.12}$$

viene definida por [1]. Para sistemas en los que el vector de posición no depende explícitamente del tiempo esta relación se puede simplificar a

$$\mathbf{v}_i = \sum_k \frac{\partial \mathbf{r}_i}{\partial q_n} \dot{q}_n = \begin{bmatrix} \frac{\partial \mathbf{r}_i}{\partial q_1} & \frac{\partial \mathbf{r}_i}{\partial q_2} & \dots & \frac{\partial \mathbf{r}_i}{\partial q_n} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix} = [\mathbf{b}_1^i \quad \mathbf{b}_2^i \quad \dots \quad \mathbf{b}_n^i] \dot{\mathbf{q}} = [\mathbf{B}^i] \dot{\mathbf{q}}.\tag{2.13}$$

La Ecuación 2.13 muestra que la velocidad de cualquier partícula dentro del sistema viene dada por las coordenadas generalizadas debido a que la matriz  $[\mathbf{B}^i(\mathbf{q})]$  es dependiente de dichas coordenadas y existe un término diferencial de primer orden en la ecuación.

El cálculo de la aceleración de la partícula en cuestión es

$$\mathbf{a}_i = \sum_k \frac{\partial \mathbf{r}_i}{\partial q_n} \ddot{q}_n + \frac{d}{dt} \left( \sum_k \frac{\partial \mathbf{r}_i}{\partial q_n} \right) \dot{q}_n = \sum_k \frac{\partial \mathbf{r}_i}{\partial q_n} \ddot{q}_n + \sum_k \frac{\partial \mathbf{v}_i}{\partial q_n} \dot{q}_n = [\mathbf{B}^i] \ddot{\mathbf{q}} + [\mathbf{B}_v^i] \dot{\mathbf{q}},\tag{2.14}$$

donde la relación  $\frac{d}{dt} \left( \frac{\partial \mathbf{r}_i}{\partial q_j} \right) = \frac{\partial \mathbf{v}_i}{\partial q_j}$  viene dada por Goldstein en [1].

### 2.1.7 Descomposición en coordenadas generalizadas: Vector Velocidad Angular

La velocidad del sólido rígido se compone por una parte lineal  $\mathbf{v}$  y otra de rotación  $\boldsymbol{\Omega}$ . De la misma forma que se ha descompuesto la velocidad lineal en las coordenadas generalizadas, se hará el proceso

análogo para la velocidad angular. Debido a que el vector de velocidad angular aparece a partir de una matriz antisimétrica se estudiará desde este caso como

$$[\boldsymbol{\Omega}_i]^\times = [\mathbf{S}^i]^T \frac{d}{dt} [\mathbf{S}^i] = [\mathbf{S}^i]^T \left( \frac{\partial [\mathbf{S}^i]}{\partial t} + \sum_k \frac{\partial [\mathbf{S}^i]}{\partial q_k} \dot{q}_k \right). \quad (2.15)$$

Para sistemas en los que la matriz de transformación no depende explícitamente del tiempo la relación se simplifica a

$$[\boldsymbol{\Omega}_i]^\times = [\mathbf{S}^i]^T \frac{d}{dt} [\mathbf{S}^i] = [\mathbf{S}^i]^T \sum_k \frac{\partial [\mathbf{S}^i]}{\partial q_k} \dot{q}_k = \sum_k [\mathbf{S}^i]^T \frac{\partial [\mathbf{S}^i]}{\partial q_k} \dot{q}_k. \quad (2.16)$$

Obteniendo la descomposición de la matriz del producto vectorial en las coordenadas generalizadas esta se puede transformar nuevamente a vector como

$$\boldsymbol{\Omega}_i = \sum_k \left\{ [\mathbf{S}^i]^T \frac{\partial [\mathbf{S}^i]}{\partial q_k} \right\}^{Vect} \dot{q}_k, \quad (2.17)$$

donde el simbolo “Vect”  $\{\}$  simboliza el paso de matriz antisimétrica al vector del que se compone esa matriz. De la misma forma que en la velocidad lineal se denominaban los vectores  $\mathbf{b}_1^i, \mathbf{b}_2^i, \dots, \mathbf{b}_n^i$ , para el caso de la velocidad angular se definirán como  $\mathbf{c}_1^i, \mathbf{c}_2^i, \dots, \mathbf{c}_n^i$ . Esta definición es

$$\mathbf{c}_k^i = \left\{ [\mathbf{S}^i]^T \frac{\partial [\mathbf{S}^i]}{\partial q_k} \right\}^{Vect}, \quad (2.18)$$

y, de igual forma que para la velocidad lineal, el arreglo matricial para la velocidad angular viene dado como

$$\boldsymbol{\Omega}_i = \sum_k \mathbf{c}_k^i \dot{q}_k = [\mathbf{c}_1^i \quad \mathbf{c}_2^i \quad \dots \quad \mathbf{c}_n^i] \dot{\mathbf{q}} = [\mathbf{C}^i] \dot{\mathbf{q}}. \quad (2.19)$$

El vector de aceleración angular viene por

$$\boldsymbol{\alpha}_i = \frac{d}{dt} ([\mathbf{C}^i] \dot{\mathbf{q}}) = [\mathbf{C}^i] \ddot{\mathbf{q}} + [\mathbf{C}_v^i] \dot{\mathbf{q}}. \quad (2.20)$$

### 2.1.8 Descomposición en coordenadas generalizadas: Derivada parcial respecto a primer orden

La librería emplea métodos para el cálculo de las parciales de las velocidades presentados en las secciones 2.1.6 y 2.1.7, sin embargo existen métodos más sencillos y directos como

$$\mathbf{v}_i = \sum_k \frac{\partial \mathbf{v}_i}{\partial \dot{q}_k} \dot{q}_k \quad (2.21)$$

y

$$\boldsymbol{\Omega}_i = \sum_k \frac{\partial \boldsymbol{\Omega}_i}{\partial \dot{q}_k} \dot{q}_k. \quad (2.22)$$

Se puede observar que en vez de realizar derivadas parciales respecto al vector de posición  $\mathbf{r}$  y su análogo angular, se realizan sobre los mismos vectores de velocidad lineal  $\mathbf{v}$  y de velocidad angular  $\boldsymbol{\Omega}$ . Es fácil ver que en el caso de la velocidad angular, el método presentado anteriormente debe realizar la derivada parcial de una matriz seguido de un producto matricial, mientras que en este caso únicamente con el vector de velocidad angular y las derivadas parciales bastaría. Los vectores de velocidad se tienen que calcular de antemano pero, una vez se tienen, es más sencillo de implementar.

La elección de los métodos anteriores radica en que en su momento se encontraron dificultades con Matlab para realizar de forma correcta una derivada parcial respecto a una derivada de la coordenadas generalizada de primer orden  $\frac{\partial \mathbf{v}}{\partial \dot{q}_k}$ . Al servir los dos métodos de igual forma, para evitar la reprogramación de la librería se han dejado los métodos anteriores.

## 2.2 Marco práctico: Cinemática

En este marco práctico se muestran dos ejemplos sobre la descomposición en las coordenadas generalizadas de los vectores velocidad lineal y angular. Para la velocidad angular se han propuesto los ejemplos de dos rotaciones a través de ángulos de Euler con las secuencias 1-2-3 y 2-3 respectivamente. Para la velocidad lineal se presenta el modelo de un péndulo simple como una partícula puntual. Ambas descomposiciones se exponen finalmente en el modelo del péndulo simple como un sólido rígido.

### 2.2.1 Matriz de articulación: Euler 1-2-3

La matriz de cambio de base  $[\mathbf{S}]$  que contiene las componentes de los vectores de la base  $1''2''3''$  en la base  $xyz$  es

$$[\mathbf{S}]_{xyz}^{1''2''3''} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_1 & -s_1 \\ 0 & s_1 & c_1 \end{bmatrix} \begin{bmatrix} c_2 & 0 & s_2 \\ 0 & 1 & 0 \\ -s_2 & 0 & c_2 \end{bmatrix} \begin{bmatrix} c_3 & -s_3 & 0 \\ s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_2 c_3 & -c_2 s_3 & s_2 \\ c_1 s_3 + c_3 s_1 s_2 & c_1 c_3 - s_1 s_2 s_3 & -c_2 s_1 \\ s_1 s_3 - c_1 c_3 s_2 & c_3 s_1 + c_1 s_2 s_3 & c_1 c_2 \end{bmatrix}. \quad (2.23)$$

La Ecuación 2.18 mostraba el cálculo de las componentes de la velocidad angular a través de la matriz de cambio de base. Debe recordarse que la Ecuación 2.22 es igualmente válida, e incluso más utilizada normalmente. Para el ángulo  $\theta_1$  la parcial es

$$[\mathbf{c}_1] = [\mathbf{S}]^T \frac{\partial [\mathbf{S}]}{\partial \theta_1} = \begin{bmatrix} 0 & -s_2 & -c_2 s_3 \\ s_2 & 0 & -c_2 c_3 \\ c_2 s_3 & c_2 c_3 & 0 \end{bmatrix} \rightarrow \mathbf{c}_1 = \begin{Bmatrix} c_2 c_3 \\ -c_2 s_3 \\ s_2 \end{Bmatrix}_{1''2''3''}, \quad (2.24)$$

para  $\theta_2$

$$[\mathbf{c}_2] = [\mathbf{S}]^T \frac{\partial [\mathbf{S}]}{\partial \theta_2} = \begin{bmatrix} 0 & 0 & c_3 \\ 0 & 0 & -s_3 \\ -c_3 & s_3 & 0 \end{bmatrix} \rightarrow \mathbf{c}_2 = \begin{Bmatrix} s_3 \\ c_3 \\ 0 \end{Bmatrix}_{1''2''3''}, \quad (2.25)$$

y finalmente, la parcial respecto a  $\theta_3$  se representa es

$$[\mathbf{c}_3] = [\mathbf{S}]^T \frac{\partial [\mathbf{S}]}{\partial \theta_3} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \mathbf{c}_3 = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}_{1''2''3''}. \quad (2.26)$$

Estas parciales se pueden presentar a través de la matriz  $[\mathbf{C}]$ , la cual define como “*Joint Partial*” o parciales de las articulaciones. Para este caso esta matriz

$$\{\boldsymbol{\Omega}_i\}_{1''2''3''} = [\mathbf{C}] \dot{\mathbf{q}} = \left\{ \begin{bmatrix} c_2 c_3 & s_3 & 0 \\ -c_2 s_3 & c_3 & 0 \\ s_2 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \right\}_{1''2''3''} \quad (2.27)$$

se muestra junto con el vector de las coordenadas generalizadas que intervienen en la rotación. Este resultado se corrobora con lo presentado por Stoneking en [9].

Para obtener otro tipo de matrices parciales se recomienda consultar [10].

### 2.2.2 Matriz de articulación: Euler 2-3

Para el caso de las parciales asociadas a la secuencia 2-3 de Euler se presenta la matriz de cambio de base como

$$[\mathbf{S}]_{xyz}^{1'2'3'} = \begin{bmatrix} c_1 & 0 & s_1 \\ 0 & 1 & 0 \\ -s_1 & 0 & c_1 \end{bmatrix} \begin{bmatrix} c_3 & -s_3 & 0 \\ s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_1 c_2 & -c_1 s_2 & s_1 \\ s_2 & c_2 & 0 \\ -c_2 s_1 & s_1 s_2 & c_1 \end{bmatrix}. \quad (2.28)$$

Las parciales vienen dadas por

$$\mathbf{c}_1 = \begin{Bmatrix} s_2 \\ c_2 \\ 0 \end{Bmatrix}_{1'2'3'} \quad (2.29)$$

y por

$$\mathbf{c}_2 = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}_{1'2'3'}. \quad (2.30)$$

La matriz parcial de la articulación es

$$\{\Omega_i\}_{1'2'3'} = [C] \dot{\mathbf{q}} = \left\{ \begin{bmatrix} s_2 & 0 \\ c_2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \right\}_{1'2'3'}, \quad (2.31)$$

la cual es idéntica a la mostrada por Stoneking en [11].

### 2.2.3 Péndulo simple: Sistema de partículas

En este apartado se realizará el cálculo de las componentes parciales de la velocidad lineal de un péndulo simple. La Figura 2.8 muestra el péndulo en cuestión y la dirección del vector parcial de la velocidad.

La ligadura entre la referencia inercial y la partícula es

$$\mathbf{O}_{R.I} \mathbf{P} = \begin{Bmatrix} 0 \\ l_1 s(\theta_1) \\ -l_1 c(\theta_1) \end{Bmatrix}_{xyz}, \quad (2.32)$$

y tiene la forma mostrada por la Ecuación 2.11. Para el cálculo de la componente parcial de la velocidad angular se hace uso de lo presentado anteriormente por la Ecuación 2.13. Siguiendo lo anterior la componente de la velocidad lineal de la partícula se representa como

$$\mathbf{b}_{\theta_1}^P = \frac{\partial}{\partial \theta_1} \mathbf{O}_{R.I} \mathbf{P} = \begin{Bmatrix} 0 \\ l_1 c(\theta_1) \\ l_1 s(\theta_1) \end{Bmatrix}_{xyz}. \quad (2.33)$$

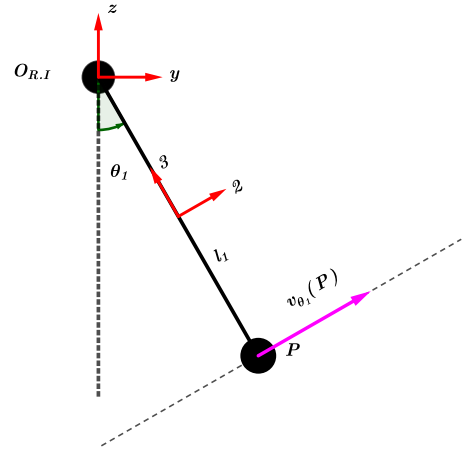


Figura 2.8: Esquema péndulo simple.

### 2.2.4 Péndulo simple: Sólido rígido

El modelo de péndulo físico como un sólido rígido se presenta en la Figura 2.9. La diferencia de este péndulo con el péndulo simple anterior radica en que se considera un sistema de partículas en el cual todo el movimiento se puede estudiar a través del centro de masas  $\mathbf{G}$ .

De forma análoga al péndulo simple anterior se presenta la ligadura del centro de masas  $\mathbf{G}$  con la referencia inercial como

$$\mathbf{OG} = \begin{Bmatrix} 0 \\ l_{CM} s(\theta) \\ -l_{CM} c(\theta) \end{Bmatrix}_{xyz}, \quad (2.34)$$

y

$$\mathbf{b}_{\theta}^{Bar} = \frac{\partial}{\partial \theta} \mathbf{OG} = \begin{Bmatrix} 0 \\ l_{CM} c(\theta) \\ l_{CM} s(\theta) \end{Bmatrix}_{xyz} \quad (2.35)$$

muestra la parcial de la velocidad lineal del sólido rígido respecto a  $\theta$ .

Para el cálculo de la velocidad angular es necesaria la matriz de cambio de base

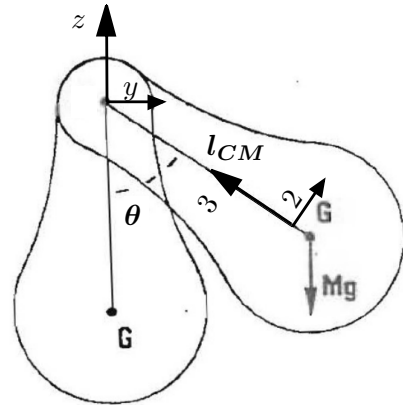


Figura 2.9: Péndulo simple como sólido rígido [5]

$$[\mathbf{S}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_{\theta} & -s_{\theta} \\ 0 & s_{\theta} & c_{\theta} \end{bmatrix}. \quad (2.36)$$

$$[c_\theta] = [S]^T \frac{\partial [S]}{\partial \theta} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow c_\theta^{Bar} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix}_{123} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix}_{xyz}, \quad (2.37)$$

La parcial de la velocidad angular del sólido rígido es  
y por lo tanto, la velocidad angular del sólido se calcula como

$$\{\Omega_i\}_{123} = [C] \dot{q} = \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \dot{\theta} \right\}_{123} = \left\{ \begin{bmatrix} \dot{\theta} \\ 0 \\ 0 \end{bmatrix} \right\}_{123} = \left\{ \begin{bmatrix} \dot{\theta} \\ 0 \\ 0 \end{bmatrix} \right\}_{xyz}. \quad (2.38)$$

## 2.3 Marco teórico: Dinámica

Tras haber introducido las nociones cinemáticas de los sistemas físicos en la sección 2.1, se deben definir las ideas que estudian el porqué del movimiento y como se genera este, es decir, la dinámica de los sistemas mecánicos.

### 2.3.1 Primer Teorema Vectorial: Teorema de la “Fuerza Resultante de Inercia” [2]

La segunda ley de Newton postula que la variación de la cantidad de movimiento (o momento lineal en algunos textos) de una partícula en una referencia inercial es igual a la suma de todas las fuerzas exteriores que interaccionan con la partícula. La Ecuación

$$\sum_i^{sist} \mathbf{F}_i(P) = \frac{d\mathbf{D}(P)}{dt}|_{R.I} = m \frac{d\mathbf{V}_{R.I}(P)}{dt}|_{R.I} = m\mathbf{A}_{R.I}(P) \quad (2.39)$$

expone la relación entre la cantidad de movimiento con la velocidad y la masa de la partícula, y por tanto, se observa de forma clara que la fuerza aplicada a un cuerpo produce una aceleración en este proporcional a su masa.

Otra forma de ver este postulado es que es necesaria una fuerza ficticia o de inercia  $\mathcal{F}_{R.I}(P) = -m\mathbf{A}(P)_{R.I}$  para mantener el sistema en equilibrio. La Ecuación

$$\mathcal{F}_{R.I}(P) + \sum_i \mathbf{F}(P) = \mathbf{0}, \quad (2.40)$$

donde

$\mathcal{F}_{R.I}(P) = -m(P)\mathbf{A}_{R.I}(P)$ . Fuerza de inercia de d'Alembert sobre P y  
 $\sum_i \mathbf{F}(P)$  = Fuerzas que actúan sobre P,

es otra forma de entender la segunda ley de Newton.

En un sistema de partículas, hace falta una fuerza de inercia  $\mathcal{F}_{R.I}(P_k)$  para mantener a esa partícula en equilibrio ante las fuerzas  $\sum_i \mathbf{F}(P_k)$ , si se aplica esto para el conjunto de partículas del sistema  $n_{sist}$  se obtiene

$$\sum_{k=1}^{n_{sist}} [\mathcal{F}_{R.I}(P_k)] + \sum_{k=1}^{n_{sist}} \left[ \sum_i \mathbf{F}(P_k) \right] = \mathbf{0} \quad (2.41)$$

donde:

$P_k$  = Partícula k del sistema,  
 $\mathcal{F}_{R.I}(P_k) = -m(P_k)\mathbf{A}_{R.I}(P_k)$ . Fuerza de inercia de d'Alembert sobre  $P_k$  y  
 $\sum_i \mathbf{F}(P_k)$  = Fuerzas que actúan sobre  $P_k$ .

Esta relación es conocida como “Primer Teorema Vectorial” o Teorema de la “Fuerza Resultante de Inercia”.

Sea  $\mathbf{G}$  el CDG del sistema de partículas  $Sist$  se puede demostrar que existe una fuerza de inercia total

$$\mathcal{F}(Sist) = \sum_{k=1}^{n_{sist}} [\mathcal{F}_{R.I}(P_k)] = -m_{Sist}\mathbf{A}_{R.I}(\mathbf{G}) \quad (2.42)$$

para todo el sistema.

La fuerza de inercia total del sistema se puede calcular a través de la masa total  $m_{Sist}$  y la aceleración del centro de masas  $\mathbf{A}_{R.I}(\mathbf{G})$ . Si se denomina  $\mathbf{F}(Sist)$  a todo el conjunto de fuerzas que se aplica sobre todo el sistema

$$\mathbf{F}(Sist) = \sum_{k=1}^{n_{sist}} \left[ \sum_i \mathbf{F}(P_k) \right], \quad (2.43)$$

el “Primer Teorema Vectorial” se puede escribir de la forma

$$\mathcal{F}(Sist) + \mathbf{F}(Sist) = \mathbf{0}. \quad (2.44)$$

Cabe destacar que la ecuación 2.44 sirve para cualquier sistema de partículas pero en el caso que atañe esta memoria se centrará principalmente en el sólido rígido.

### 2.3.2 Segundo Teorema Vectorial: Teorema del “Momento Resultante de Inercia” [2]

Para el estudio del sólido rígido se debe incluir la noción de momento o par de fuerzas. Este concepto añade a las leyes de Newton el caracter de rotación que tienen los sólidos rígidos y por lo tanto de forma análoga al “Primer Teorema Vectorial” se propone el “Segundo Teorema Vectorial”

$$\mathcal{M}_B(Sist) + \mathbf{M}_B(Sist) = \mathbf{0} \quad (2.45)$$

donde:

$\mathbf{B}$  = Punto de cálculo del “Segundo Teorema Vectorial”,  
 $\mathcal{M}_B(Sist) = -\frac{d\mathcal{H}_B}{dt}|_{R.I} - m_{Sol}\mathbf{B}\mathbf{G} \times \mathbf{A}_{R.I}(\mathbf{B})$ . Momento de Inercia de d'Alembert en  $\mathbf{B}$  y  
 $\mathbf{M}_B(Sist)$  = Momentos que actúan en  $\mathbf{B}$ .

Este teorema parte del momento cinético (o momento angular)  $\mathcal{H}$  y, a través de operaciones análogas a la cantidad de movimiento  $\mathcal{D}$  y su aplicación a sistemas de partículas, llega al caso general para sólidos rígidos. Si bien el teorema se ha definido para cualquier punto no perteneciente al sólido, este se vuelve más sencillo si es aplicado a un punto del mismo. En especial, si se selecciona el CGD  $\mathbf{G}$  del sólido rígido el momento cinético, puede escribirse como



$$\mathcal{H}_G(Sol) = \mathcal{I}_G(Sol)\Omega_{R.I}(Sol). \quad (2.46)$$

El momento cinético en este caso depende unicamente de una matriz que se denomina “tensor de inercia  $\mathcal{I}_G(Sol)$ ”, que contiene los parámetros del sólido rígido que describen su efecto ante una determinada rotación. Este momento cinético también depende de la velocidad angular del sólido rígido calculado en una referencia inercial  $\Omega_{R.I}(Sol)$ . Como se ha comentado, los cálculos se pueden realizar para puntos distintos de  $G$ , incluso si estos puntos no pertenecen al sólido. También es posible considerar referencias no inerciales pero de igual forma que realizar el cálculo en puntos diferentes, esto dificulta las operaciones y no es necesario.

De forma análoga a la fuerza de inercia se puede presentar el momento de inercia en  $G$  en función de las coordenadas generalizadas como

$$\mathcal{M}_{B_i} = \frac{d}{dt}\mathcal{H}_{G_i} = \frac{d}{dt}\left(\mathcal{I}_{G_i}\Omega_{R.I}^i\right) = \mathcal{I}_{G_i}\frac{d}{dt}\Omega_{R.I}^i = \mathcal{I}_{G_i}\alpha_{R.I}^i = \mathcal{I}_{G_i}\left(\left[C^i\right]\ddot{q} + \left[C_v^i\right]\dot{q}\right). \quad (2.47)$$

### 2.3.3 Método de los trabajos virtuales. Principio de D’Alembert [1]

Uno de los primeros saltos que acerca la mecánica newtoniana a la lagrangiana es el principio de D’Alembert. A continuación se van a introducir los conceptos de desplazamiento virtual y el estudio de los sistemas en los que el trabajo virtual de las fuerzas de ligaduras es nulo.

Desplazamiento virtual (infinitesimal) de un sistema es el cambio de configuración de este a consecuencia de una variación infinitesimal arbitraria de las coordenadas  $\delta\mathbf{r}_i$ , *compatible con las fuerzas y ligaduras impuestas al sistema en el instante dado  $t$* . Se llama virtual al desplazamiento para distinguirlo del desplazamiento real del sistema que tiene lugar en un intervalo de tiempo  $dt$ , durante el cual pueden variar las fuerzas y las ligaduras [1].

A través del “Primer Teorema Vectorial” de la ecuación 2.44 se puede calcular el trabajo virtual de las fuerzas de inercia y de las fuerzas que actúan sobre cada partícula como

$$\mathcal{F}(Sist)\delta\mathbf{r}_i + \mathbf{F}(Sist)\delta\mathbf{r}_i = \mathbf{0}. \quad (2.48)$$

Debido a que el sistema estudiado se encuentra en equilibrio, el trabajo virtual que realiza el sistema debe ser nulo.

A su vez, las fuerzas que actúan sobre cada una de las partículas en el sistema se pueden descomponer en fuerzas externas  $\mathbf{F}^e(Sist)$  y en fuerzas internas  $\mathbf{F}^i(Sist)$  como

$$\mathbf{F}(Sist)\delta\mathbf{r}_i = \mathbf{F}^e(Sist)\delta\mathbf{r}_i + \mathbf{F}^i(Sist)\delta\mathbf{r}_i. \quad (2.49)$$

Las fuerzas internas son aquellas que se relacionan con las ligaduras que existen en el sistema. Debido a que el vector de desplazamiento virtual puede ser arbitrario (dentro de los límites de la definición), se puede escoger de tal forma que el trabajo virtual de las fuerzas internas sea nulo

$$\mathbf{F}^i(Sist)\delta\mathbf{r}_i = \mathbf{0}. \quad (2.50)$$

De esta manera se puede obtener una forma simple del equilibrio del trabajo virtual realizado por el sistema de partículas como

Los sistemas con ligaduras holónomas son una aplicación interesante del concepto de desplazamiento virtual. Como ya se ha explicado, en este tipo de sistemas se pueden encontrar variables independientes

$$\mathcal{F}(Sist)\delta\mathbf{r}_i + \mathbf{F}^e(Sist)\delta\mathbf{r}_i = \mathbf{0}. \quad (2.51)$$

para definir las configuraciones del sistema (Ecuación 2.11). Al igual que para los desplazamientos reales, Goldstein [1] demuestra que los desplazamientos virtuales  $\delta\mathbf{r}_i$  se pueden relacionar mediante los desplazamientos virtuales de  $\delta q_j$  como

$$\delta\mathbf{r}_i = \sum_j \frac{\partial\mathbf{r}_i}{\partial q_j} \delta q_j. \quad (2.52)$$

Esta definición de desplazamientos virtuales se puede realizar de igual forma si se definen unas “velocidades virtuales”. Este concepto aparece en [3] como

$$\delta\mathbf{v}_i = \sum_j \frac{\partial\mathbf{r}_i}{\partial q_j} \delta\dot{q}_j = \sum_j \frac{\partial\mathbf{v}_i}{\partial \dot{q}_j} \delta\dot{q}_j = \sum_j \mathbf{b}_j^i \delta\dot{q}_j \quad (2.53)$$

En este caso, el principio pasa a ser “El Principio de las Potencias Virtuales” debido a que el producto final es el equilibrio del sistema de partículas por unidad de tiempo descrito como

$$\mathcal{F}(Sist)\delta\mathbf{v}_i + \mathbf{F}^e(Sist)\delta\mathbf{v}_i = \mathbf{0}. \quad (2.54)$$

El primer término de la Ecuación 2.51, que se calcula como

$$\mathcal{Q}(Sist) = \mathcal{F}(Sist)\delta\mathbf{r}_i = \sum_j \left[ \frac{d}{dt} \left( \frac{\partial}{\partial \dot{q}_j} \left( \sum_i \frac{1}{2} m_i v_i^2 \right) \right) - \frac{\partial}{\partial q_j} \left( \sum_i \frac{1}{2} m_i v_i^2 \right) \right], \quad (2.55)$$

se denomina “Fuerza de Inercia Generalizada”. El segundo término, asociado a las fuerzas externas, se calcula como

$$\mathcal{Q}(Sist) = \mathbf{F}^e(Sist)\delta\mathbf{r}_i = \sum_j \mathbf{F}^e(Sist) \frac{\partial\mathbf{r}_i}{\partial q_j}, \quad (2.56)$$

y se denomina “Fuerza Externa Generalizada”. Si la “Fuerza de Inercia Generalizada” se simplifica todavía más teniendo en cuenta que aparece el término de la energía cinética, se llega a las ecuaciones de Lagrange

$$\sum_j \left[ \frac{d}{dt} \left( \frac{\partial T}{\partial \dot{q}_j} \right) - \frac{\partial T}{\partial q_j} \right] \delta q_j - \sum_j \mathbf{F}_i^e(Sist) \frac{\partial\mathbf{r}_i}{\partial q_j} \delta q_j = \mathbf{0} \quad (2.57)$$

Para fuerzas externas conservativas, como puede ser la acción gravitatoria, la fuerza generalizada es igual al gradiente del potencial de dicha fuerza. Por eso mismo la mayor parte de la bibliografía científica muestra las ecuaciones de Lagrange a través de la diferencia de energía cinética y potencial que se denomina Lagrangiano  $\mathcal{L} = T - U$ . Esto reduce la Ecuación 2.57 a

$$\frac{d}{dt} \left( \frac{\partial \mathcal{L}}{\partial \dot{q}_j} \right) - \frac{\partial \mathcal{L}}{\partial q_j} = \mathbf{Q}(Sist)_j. \quad (2.58)$$

El paso de la Ecuación 2.57 a la 2.58 permite el desacoplo de las ecuaciones del movimiento a un sistema de  $n$  ecuaciones. Este resultado es muy importante debido a que las ecuaciones del movimiento se van a poder estudiar a través de álgebra lineal y por lo tanto la aplicación de métodos de control clásicos resulta más sencilla.

La Ecuación 2.57 muestra que el elemento virtual  $\delta q_j$  es irrelevante y se puede simplificar para que no aparezca, como se ha hecho en la Ecuación 2.58. Este hecho hace que este desarrollo se pueda realizar tanto para desplazamientos virtuales como para velocidades virtuales como ya se ha mostrado en las ecuaciones 2.52 y 2.53, el único cambio radica en que en vez del cálculo del trabajo virtual se realiza el de la potencia virtual del sistema.

### 2.3.4 Método de los trabajos virtuales. Sólido Rígido

Debido a que en el sólido rígido intervienen los conceptos de velocidad angular y momentos, se deben incluir estos dentro del cálculo del trabajo o potencia virtual.

Análogamente a la Ecuación 2.53 se define un vector velocidad angular virtual

$$\delta \boldsymbol{\Omega}_i = \sum_j \left\{ [\mathbf{S}^i]^T \frac{\partial [\mathbf{S}^i]}{\partial q_k} \right\}^{Vect} \delta \dot{q}_j = \sum_j \frac{\partial \boldsymbol{\Omega}_i}{\partial \dot{q}_j} \delta \dot{q}_j = \sum_j \mathbf{c}_j^i \delta \dot{q}_j. \quad (2.59)$$

La potencia virtual total del sistema vendrá dada por

$$\mathcal{F}(Sist) \delta \mathbf{v}_i + \mathbf{F}^e(Sist) \delta \mathbf{v}_i + \mathcal{M}(Sist) \delta \boldsymbol{\Omega}_i + \mathbf{M}^e(Sist) \delta \boldsymbol{\Omega}_i = 0. \quad (2.60)$$

Este resultado es completamente análogo al conocido método de Kane [11].

### 2.3.5 Torsores

En esta memoria los torsores se definen como la combinación de la fuerza y el momento que actúa sobre un sólido rígido. Las fuerzas y los momentos se producen debido a las diferentes acciones: acción gravitatoria, rozamiento, viento, etc., pero como ya se ha definido, estas acciones generan tanto una fuerza como un momento. Estas dos magnitudes se introducen en un elemento denominado tursor, que define de forma compacta una acción sobre un sólido. Debido a que los momentos se definen respecto a un punto, el tursor también deberá estar definido en un punto. La Ecuación

$$(\mathbf{F}^{Vis}, \mathbf{M}_{\mathbf{G}_A}^{Vis}) \quad (2.61)$$

muestra el tursor de la acción de rozamiento viscoso que actúa sobre la centroides del área de contacto  $\mathbf{G}_A$  de un sólido rígido.

Para calcular el tursor en otro punto se debe aplicar la regla de traslación de torsos

$$(\mathbf{F}^{Vis}, \mathbf{M}_{\mathbf{G}_B}^{Vis}) = (\mathbf{F}^{Vis}, \mathbf{M}_{\mathbf{G}_A}^{Vis} + \mathbf{B}\mathbf{G}_A \times \mathbf{F}^{Vis}) \quad (2.62)$$

### 2.3.6 Método de los trabajos virtuales. Método Matricial

Si bien el cálculo de las fuerzas y momentos generalizados expuestos es correcto, la implementación en programación puede ser tediosa. En este apartado se presentarán los mismos conceptos ya presentados a través de objetos de álgebra lineal como lo son las matrices y los vectores. Esto, además de ser más sencillo computacionalmente, tiene la ventaja de aislar de forma sencilla las derivadas de segundo orden resultado en un sistema  $[\mathbf{A}(\mathbf{q})] \ddot{\mathbf{q}} = [\mathbf{RHS}(\mathbf{q}, \dot{\mathbf{q}})]$ . Con este sistema de ecuaciones final, es sencillo realizar una reducción de orden e incluso obtener un sistema de ecuaciones diferenciales ordinarias (ODE).

Una vez caracterizada la aceleración de cada partícula del sistema, es posible calcular la fuerza de inercia  $\mathcal{F}_{R.I}(\mathbf{P}_i)$  para cada una de las partículas. A través del método de las potencias virtuales se puede obtener la potencia virtual realizada por la fuerza de inercia como

$$\begin{aligned} W_{\mathcal{F}}^* &= \sum_i [-m_i \mathbf{a}_i] \delta \mathbf{v}_i = \sum_i \left[ -m_i [\mathbf{B}^i] \ddot{\mathbf{q}} \delta \mathbf{v}_i - m_i [\mathbf{B}_v^i] \dot{\mathbf{q}} \delta \mathbf{v}_i \right] = \\ &= \sum_i \left\{ -m_i [\mathbf{B}^i] \ddot{\mathbf{q}} [\mathbf{B}^i] - m_i [\mathbf{B}_v^i] \dot{\mathbf{q}} [\mathbf{B}^i] \right\} \delta v_i. \end{aligned} \quad (2.63)$$

De forma similar se puede realizar el cálculo de la potencia virtual realizado por el momento de inercia según

$$\begin{aligned} W_{\mathcal{M}}^* &= \sum_i \left[ -\mathcal{I}_{G_i} \alpha_{R.I}^i \right] \delta \Omega_i = \sum_i \left[ \mathcal{I}_{G_i} \left( -[\mathbf{C}^i] \ddot{\mathbf{q}} - [\mathbf{C}_v^i] \dot{\mathbf{q}} \right) \right] \delta \Omega_i = \\ &= \sum_i \left\{ -\mathcal{I}_{G_i} [\mathbf{C}^i] \ddot{\mathbf{q}} [\mathbf{C}^i] - \mathcal{I}_{G_i} [\mathbf{C}_v^i] \dot{\mathbf{q}} [\mathbf{C}^i] \right\} \delta \Omega_i. \end{aligned} \quad (2.64)$$

$$-m_i [\mathbf{B}^i] \ddot{\mathbf{q}} [\mathbf{B}^i] = -\sum_k^n \mathbf{b}_k^i \ddot{q}_k \sum_j^n \mathbf{b}_j^i = -m_i [\mathbf{A}_i^{\mathcal{F}}] \ddot{\mathbf{q}} \quad (2.65)$$

donde:

$$[\mathbf{A}_i^{\mathcal{F}}]_{kj} = m_i \mathbf{b}_k^i \cdot \mathbf{b}_j^i,$$

muestra el paso de sumatorios a operaciones matriciales de la componente que contiene las derivadas de segundo orden ( $\ddot{\mathbf{q}}$ ) de la fuerza de inercia  $\mathcal{F}$ . Este resultado se puede obtener de forma sencilla aplicando el “Convenio de suma de Einstein” [12] o desarrollando las expresiones hasta encontrar el patrón.

De igual forma

$$-\mathcal{I}_{G_i} [\mathbf{C}^i] \ddot{\mathbf{q}} [\mathbf{C}^i] = -\mathcal{I}_{G_i} \sum_k^n \mathbf{c}_k^i \ddot{q}_k \sum_j^n \mathbf{c}_j^i = -[\mathbf{A}_i^{\mathcal{M}}] \ddot{\mathbf{q}} \quad (2.66)$$

donde:

$$[\mathbf{A}_i^{\mathcal{M}}]_{kj} = [\mathcal{I}_{G_i} \mathbf{c}_k^i]^T \cdot \mathbf{c}_j^i,$$

realiza un procedimiento análogo para el cálculo del momento de inercia generalizado. Las componentes de la matriz  $[\mathbf{A}^i]$  completa de un sólido rígido es

$$[\mathbf{A}^i]_{kj} = m_i \mathbf{b}_k^i \cdot \mathbf{b}_j^i + [\mathcal{I}_{G_i} \mathbf{c}_k^i]^T \cdot \mathbf{c}_j^i. \quad (2.67)$$

Todo el desarrollo presentado hasta el momento se ha realizado para los términos diferenciales de segundo orden de la fuerza y momentos de inercia generalizados. Para el caso de los términos de primer orden se puede realizar un procedimiento completamente análogo en el que se definirá una matriz  $[\mathbf{A}_v^i]$  como

$$\left[ \mathbf{A}_v^i \right]_{kj} = m_i \mathbf{b}_{\mathbf{k}_v}^i \cdot \mathbf{b}_j^i + \left[ \mathcal{I}_{G_i} \mathbf{c}_{\mathbf{k}_v}^i \right]^T \cdot \mathbf{c}_j^i = [\mathbf{A}^i]_{kj}. \quad (2.68)$$

Esta matriz se puede calcular bien por sus vectores o sabiendo que es la derivada de primer orden de la matriz  $[\mathbf{A}^i]$ . De cualquiera de las dos formas, las ecuaciones que representan la parte de las fuerzas inerciales se pueden obtener como

$$\sum_i \left( -[\mathbf{A}^i] \ddot{\mathbf{q}} - [\mathbf{A}_v^i] \dot{\mathbf{q}} \right) = \sum_i \left( -[\mathbf{A}^i] \ddot{\mathbf{q}} - [\dot{\mathbf{A}}^i] \dot{\mathbf{q}} \right) = -[\mathbf{A}(\mathbf{Sist})] \ddot{\mathbf{q}} - [\mathbf{A}(\dot{\mathbf{Sist}})] \dot{\mathbf{q}}. \quad (2.69)$$

Este resultado es muy importante debido a que nos muestra varios conceptos que facilitan la programación de la librería. Si se tiene un sistema de partículas, las ecuaciones de la parte inercial de cada una de ellas puede representarse por: una matriz  $[\mathbf{A}_i]$  y las derivadas de primer y segundo orden de las coordenadas generalizadas  $\mathbf{q}$ . Este concepto se puede extrapolar a varios sistemas de partículas que interaccionan entre ellos en los que cada uno tiene una masa  $m_{Sist}^i$  y una matriz  $[\mathbf{A}_{Sist}^i]$ . Debido a que el sólido rígido nace de este concepto, cada uno de los sólidos rígidos tendrá su masa  $m(\mathbf{Sol})$  y su matriz  $[\mathbf{A}(\mathbf{Sol})]$ . De esta manera, se pueden estudiar las ecuaciones de inercia de cada uno de los sistemas de partículas y sólidos rígidos por separado, y posteriormente ensamblarlos todos dentro de una matriz  $[\mathbf{A}(\mathbf{Sist})]$  la cual define el sistema completo como

$$[\mathbf{A}(\mathbf{Sist})] = [\mathbf{A}(\mathbf{Sol}_1)] + [\mathbf{A}(\mathbf{Sol}_2)] + [\mathbf{A}(\mathbf{P}_1)] + \cdots + [\mathbf{A}(\mathbf{Sol}_N)] + [\mathbf{A}(\mathbf{P}_N)] = \sum_{i=1}^N [\mathbf{A}^i]. \quad (2.70)$$

Por lo tanto el sistema de ecuaciones del sistema completo se propone según

$$-[\mathbf{A}(\mathbf{Sist})] \ddot{\mathbf{q}} - [\mathbf{A}(\dot{\mathbf{Sist}})] \dot{\mathbf{q}} + \mathbf{Q}(\mathbf{Sist}) = \mathbf{0}. \quad (2.71)$$

Como se ha hecho mención al inicio de este apartado de esta forma es más sencillo tener una separación entre las derivadas de segundo orden y el resto de los términos. La expresión final es

$$[\mathbf{A}(\mathbf{Sist})] \ddot{\mathbf{q}} = \mathbf{Q}(\mathbf{Sist}) - [\mathbf{A}(\dot{\mathbf{Sist}})] \dot{\mathbf{q}}. \quad (2.72)$$

Stoneking en [11] define el sistema de la misma manera pero con etiquetas diferentes como

$$[\mathbf{COEF}] \ddot{\mathbf{q}} = [\mathbf{RHS}]. \quad (2.73)$$

## 2.4 Marco práctico: Dinámica

### 2.4.1 Acción de fuerza gravitatoria

En la Figura 2.9 se presentó el péndulo como un sólido rígido. Sobre este péndulo actúa una fuerza gravitatoria que resulta en el vector de valor  $Mg$  presentado en la figura. La fuerza de esta acción viene dada por

$$\mathbf{F}^g = \left\{ \begin{array}{c} 0 \\ 0 \\ -Mg \end{array} \right\}_{xyz}, \quad (2.74)$$

y el momento en el centro de masas  $\mathbf{G}$  viene dado por

$$\mathbf{M}_G^g = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}_{xyz} . \quad (2.75)$$

Por lo tanto la fuerza activa generalizada debido a la gravedad se calcula como

$$\mathbf{F}_g^* = \mathbf{F}^g \cdot \mathbf{b}_\theta^{Bar} + \mathbf{M}_G^g \cdot \mathbf{c}_\theta^{Bar} = \begin{Bmatrix} 0 \\ 0 \\ -Mg \end{Bmatrix} \cdot \begin{Bmatrix} 0 \\ l_{CM}c(\theta) \\ l_{CM}s(\theta) \end{Bmatrix} + \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \cdot \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} = -Mgl_{CM}s\theta. \quad (2.76)$$

### 2.4.2 Acción momento de entrada

Para introducir un momento de entrada al péndulo simple de la Figura 2.9 es necesario presentar los vectores de fuerzas y momentos de dicha acción. La fuerza de la acción de entrada es

$$\mathbf{F}^{in} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}_{xyz} , \quad (2.77)$$

y el momento en el centro de masas  $\mathbf{G}$  viene dado por

$$\mathbf{M}_G^{in} = \begin{Bmatrix} \tau_{in} \\ 0 \\ 0 \end{Bmatrix}_{xyz} . \quad (2.78)$$

Por lo tanto la fuerza activa generalizada debido a la acción de entrada se calcula según

$$\mathbf{F}_{in}^* = \mathbf{F}^{in} \cdot \mathbf{b}_\theta^{Bar} + \mathbf{M}_G^{in} \cdot \mathbf{c}_\theta^{Bar} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \cdot \begin{Bmatrix} 0 \\ l_{CM}c(\theta) \\ l_{CM}s(\theta) \end{Bmatrix} + \begin{Bmatrix} \tau_{in} \\ 0 \\ 0 \end{Bmatrix} \cdot \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} = \tau. \quad (2.79)$$

### 2.4.3 Acción de rozamiento de Coulomb[2]

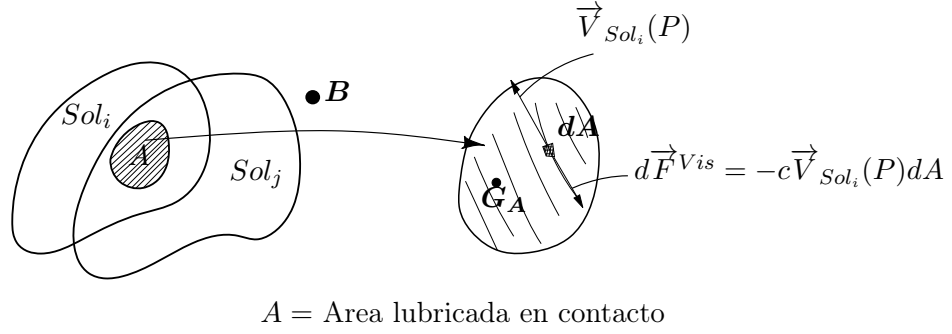


Figura 2.10: Rozamiento viscoso entre sólidos  $Sol_i$  y  $Sol_j$ .

En este apartado se va a caracterizar el torsor de rozamiento de Coulomb o rozamiento viscoso. Este torsor se va a presentar caracterizado en el centroide de la superficie  $G_A$ . El cálculo de fuerzas y momentos generalizados se realiza en el centro de gravedad. Por lo tanto, se deberá trasladar el momento resultante desde  $G_A$  hasta  $G$ . En la Figura 2.10 se muestran los sólidos  $Sol_i$  y  $Sol_j$  entre los cuales existe un líquido viscoso que produce rozamiento entre ellos.

Se puede probar que la fuerza de rozamiento viscoso actuante sobre  $Sol_j$  es

$$\mathbf{F}^{Vis} = -CAV_{Sol_i}(\mathbf{G}_A), \quad (2.80)$$

donde:

- $C = \mu/e$ ,
- $\mu$  = Viscosidad absoluta,
- $e$  = Espesor,
- $A$  = Superficie de contacto y
- $G_A$  = Centroide de la superficie de contacto,

y el momento en  $G_A$  viene dado por

$$\mathbf{M}_{G_A}^{Vis}(\mathbf{A}) = -C\mathcal{I}_{G_A}(\mathbf{A})\Omega_{Sol_i}(Sol_j), \quad (2.81)$$

donde:

- $C = \mu/e$ ,
- $\mu$  = Viscosidad absoluta,
- $e$  = Espesor ,
- $\mathcal{I}_{G_A}(\mathbf{A}) = -\int_A [\mathbf{G}_A \mathbf{P}_1]^\times [\mathbf{G}_A \mathbf{P}_1]^\times dA$  y
- $G_A$  = Centroide.

En esta ecuación  $\mathcal{I}_{G_A}(\mathbf{A})$  es una matriz que es análoga al tensor de inercia de un sólido rígido pero, en vez de realizarse sobre la masa del sólido se realiza sobre el area. El cálculo de esta matriz se puede realizar a través de la integración o con la relación

$$\mathcal{I}_{G_A}(\mathbf{A}) = -\frac{1}{e\rho} \int_A [\mathbf{G}_A \mathbf{P}_1]^\times [\mathbf{G}_A \mathbf{P}_1]^\times dm = \frac{\mathcal{I}_{G_A}(Sol_{vis})}{e\rho}. \quad (2.82)$$

En esta ecuación  $\mathcal{I}_{G_A}(Sol_{vis})$  hace referencia a la geometría del fluido viscoso. Si se supone este como un sólido que no va a cambiar su forma, pero que mantiene las propiedades del fluido en esa geometría su cálculo puede realizarse de forma análoga a los sólidos rígidos. Se debe tener en cuenta el espesor  $e$  y la densidad volumétrica  $\rho$  del fluido.

Para el péndulo simple de la Figura 2.9 se va a suponer un rozamiento viscoso en el eje de rotación. La película de fluido tiene un espesor  $e$  y se encuentra a una distancia  $R$  de la centroide  $G_A$ . La Figura 2.11 representa los parámetros y la geometría a estudiar.

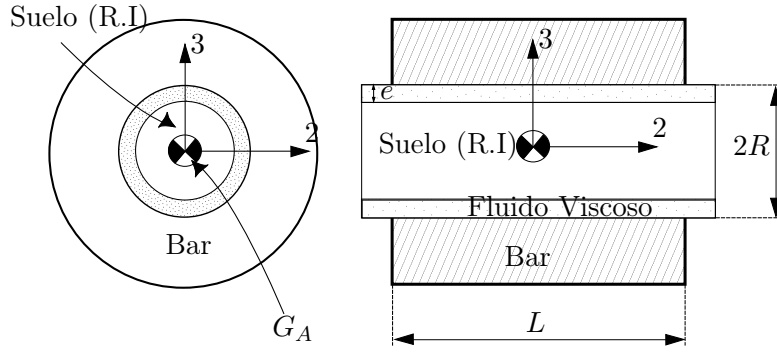


Figura 2.11: Ilustración de fluido viscoso entre el eje de rotación y la barra del péndulo.

### Fuerza Viscosa

Para el cálculo de la fuerza viscosa es necesaria la velocidad del punto  $G_A$  respecto a la referencia inercial  $\mathbf{V}_{Sue}(G_A)$ . Esta velocidad es 0 debido a que es el centro de rotación entre los dos sólidos. Por lo tanto la fuerza viscosa es el vector nulo

$$\mathbf{F}^{Vis} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}_{xyz} . \quad (2.83)$$

### Momento Viscoso en G

Para el cálculo del momento viscoso es necesaria la velocidad angular de la barra del péndulo respecto al suelo  $\Omega_{Sue}^{Bar}$ . Esta velocidad angular se ha calculado anteriormente en la Ecuación 2.38.

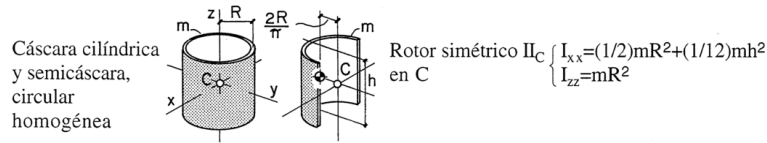


Figura 2.12: Momentos de inercia de casacara de cilindro [3].

A través de los valores tabulados de una cáscara de cilindro, que se muestran en la Figura 2.12, se calcula en tensor de inercia para el sólido viscoso como

$$\mathcal{I}_{G_A}(\mathbf{Sol}_{Vis}) = \rho 2\pi R e L \begin{bmatrix} R^2 & 0 & 0 \\ 0 & \frac{1}{2}R^2 + \frac{1}{12}L^2 & 0 \\ 0 & 0 & \frac{1}{2}R^2 + \frac{1}{12}L^2 \end{bmatrix}, \quad (2.84)$$

y por lo tanto, el tensor de area es

$$\mathcal{I}_{G_A}(\mathbf{A}) = \frac{\mathcal{I}_{G_A}(\mathbf{Sol}_{Vis})}{\rho e} = 2\pi R L \begin{bmatrix} R^2 & 0 & 0 \\ 0 & \frac{1}{2}R^2 + \frac{1}{12}L^2 & 0 \\ 0 & 0 & \frac{1}{2}R^2 + \frac{1}{12}L^2 \end{bmatrix}. \quad (2.85)$$

El momento viscoso en la centroide  $G_A$  viene dada por la Ecuación 2.81 y se calcula como



$$M_{G_A}^{Vis}(A) = M_G^{Vis}(A) = -C\mathcal{I}_{G_A}(A)\Omega_{Sue}(Bar) = -C2\pi RL \begin{bmatrix} R^2 & 0 & 0 \\ 0 & \frac{1}{2}R^2 + \frac{1}{12}L^2 & 0 \\ 0 & 0 & \frac{1}{2}R^2 + \frac{1}{12}L^2 \end{bmatrix} \begin{Bmatrix} \dot{\theta} \\ 0 \\ 0 \end{Bmatrix} = -C2\pi LR^3 \begin{Bmatrix} \dot{\theta} \\ 0 \\ 0 \end{Bmatrix}. \quad (2.86)$$

Debido a que la fuerza viscosa es nula, el momento viscoso es el mismo en el centroide  $G_A$  que en el centro de masas  $G$ .

### Fuerza activa generalizada

La fuerza activa para el torsor de rozamiento viscosos viene dada por

$$F_{Vis}^* = F^{Vis} \cdot b_{\theta}^{Bar} + M_G^{Vis} \cdot c_{\theta}^{Bar} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \cdot \begin{Bmatrix} 0 \\ l_{CM}c(\theta) \\ l_{CM}s(\theta) \end{Bmatrix} - C2\pi LR^3 \begin{Bmatrix} \dot{\theta} \\ 0 \\ 0 \end{Bmatrix} \cdot \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} = -C2\pi LR^3 \dot{\theta}. \quad (2.87)$$

### 2.4.4 Modelo completo péndulo simple

El modelo completo del péndulo simple que se muestra en la Figura 2.13 viene dado a través de la ecuaciones de Kane, que realizan la generalización de todas las fuerzas y momentos a la base de coordenadas generalizadas. Debido a la separación que se ha realizado en el desarrollo de dichas ecuaciones con las coordenadas de segundo y primer orden, las fuerzas de inercia se separarán en estos dos términos.

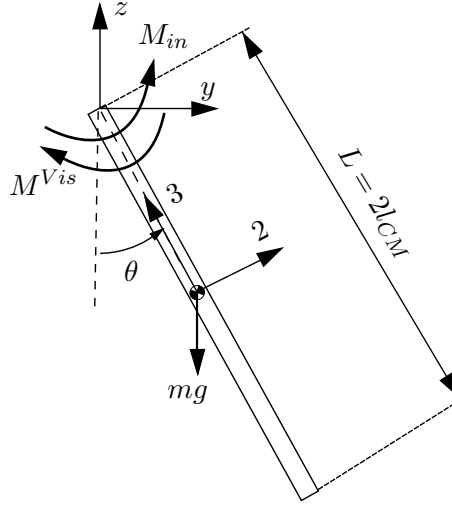


Figura 2.13: Ilustración del péndulo simple completo.

### Acción de inercia: Componente de segundo orden

La fuerza de inercia generalizada es

$$\mathcal{F}^* = -m_i [B^i] \ddot{q} [B^i] = -m \begin{Bmatrix} 0 \\ l_{CM}c(\theta) \\ l_{CM}s(\theta) \end{Bmatrix} \ddot{\theta} \begin{Bmatrix} 0 \\ l_{CM}c(\theta) \\ l_{CM}s(\theta) \end{Bmatrix} = -ml_{CM}^2 \ddot{\theta}, \quad (2.88)$$

y el momento de inercia generalizado es

$$\mathcal{M}^* = -\mathcal{I}_{G_i} [C^i] \ddot{q} [C^i] = - \begin{bmatrix} \frac{1}{12}m(2l_{CM})^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \ddot{\theta} \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} = -\frac{1}{3}ml_{CM}^2 \ddot{\theta}. \quad (2.89)$$

### Acción de inercia: Componente de primer orden

La fuerza de inercia generalizada es

$$\mathcal{F}^* = -m_i \left[ \mathbf{B}_v^i \right] \dot{\mathbf{q}} \left[ \mathbf{B}^i \right] = -m \begin{Bmatrix} 0 \\ -l_{CM}s(\theta) \\ l_{CM}c(\theta) \end{Bmatrix} \ddot{\theta} \begin{Bmatrix} 0 \\ l_{CM}c(\theta) \\ l_{CM}s(\theta) \end{Bmatrix} = 0, \quad (2.90)$$

y el momento de inercia generalizado es

$$\mathcal{M}^* = -\mathcal{I}_{G_i} \left[ \mathbf{C}_v^i \right] \dot{\mathbf{q}} \left[ \mathbf{C}^i \right] = - \begin{bmatrix} \frac{1}{12}m(2l_{CM})^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \ddot{\theta} \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} = 0. \quad (2.91)$$

### Ecuación diferencial del movimiento

Las fuerzas y los momentos aplicados al péndulo simple ya se han presentado en las secciones anteriores. Por lo tanto, la ecuación del movimiento que gobierna al péndulo simple es

$$\frac{4}{3}ml_{CM}^2\ddot{\theta} = \tau - mgl_{CM}s\theta - C2\pi LR^3\dot{\theta}. \quad (2.92)$$

## 3 Librería Dinámica: Diseño

Para el cálculo de las ecuaciones del movimiento se ha decidido programar en Matlab una librería denominada librería dinámica. Esta librería hace uso de las funciones simbólicas para realizar las operaciones vectoriales y matriciales así como la diferenciación de las expresiones obtenidas.

### 3.0.1 Otras herramientas de cálculo

Existen otras herramientas simbólicas de código abierto como la librería Sympy de Python o sageMath, que esta construida sobre otra amplia variedad de paquetes. La librería Sympy integra su propia librería para el cálculo de las ecuaciones del movimiento a través del método de Kane o del método de Lagrange. Por otro lado sageMath incluye herramientas para poder controlar los tiempos de procesado y mejorar la eficiencia del código y los algoritmos.

La elección de Matlab se debe al hecho de que la Universidad Pública de Navarra ofrece licencias de estudiante para este software y a que también es un programa contrastado y con mucha documentación. Pero se debe destacar que al realizar la librería dinámica desde sus bases, su implementación en cualquier otro lenguaje de programación que permita lenguaje simbólico es análoga.

## 3.1 Marco teórico: Programación orientada a objetos (POO)

Existen infinidad de formas de programar, de hecho, cada lenguaje de programación tiene sus características propias que lo hacen más versátil programado de una forma o de otra. Esto hace que elegir el lenguaje y método de programación sea un paso previo clave en el desarrollo de un código. En los últimos años, la programación orientada a objetos (POO) ha ido cogiendo cada vez más fama debido a la capacidad que se tiene para expresar conceptos reales en variables de programación. Esta característica es muy conveniente a la hora de programar una librería para el cálculo de sistemas mecánicos, debido a que se puede plasmar en código las ideas matemáticas y físicas necesarias para el cálculo de las ecuaciones del movimiento. Por este motivo, y debido a que Matlab permite la POO, esta será la metodología a usar.

### 3.1.1 Clases, objetos, propiedades y métodos

Las clases son tipos de datos abstractos que contienen propiedades y métodos, son plantillas a partir de las cuales se pueden formar objetos que pueden tener diferentes propiedades. Para el caso de un péndulo doble, donde cada una de las barras se considera un sólido rígido, los objetos serán los presentados en el Cuadro 3.1.

Propiedades	Nombre	$\vec{r}_G$	Base	$m$	$I_G$
Tipo	String	Sym(3x1)	Base	Sym	Sym(3x3)
Obj.1	Barra 1	$\overrightarrow{OG_1} = \begin{Bmatrix} 0 \\ \frac{1}{2}l_1s(\theta_1) \\ -\frac{1}{2}l_1c(\theta_1) \end{Bmatrix}_{xyz}$	123	$m_1$	$\begin{bmatrix} \frac{1}{12}m_1l_1^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_{xyz}$
Obj.2	Barra 2	$\overrightarrow{OG_2} = \begin{Bmatrix} 0 \\ l_1s(\theta_1) + \frac{1}{2}l_2s(\theta_2) \\ -l_1c(\theta_1) - \frac{1}{2}l_2c(\theta_2) \end{Bmatrix}_{xyz}$	1'2'3'	$m_2$	$\begin{bmatrix} \frac{1}{12}m_2l_2^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_{xyz}$

Cuadro 3.1: Propiedades de los objetos “Barra” del pendulo doble.

Este ejemplo se presenta para mostrar los diferentes tipos de datos que pueden contener las propiedades de un objeto. El nombre, en este caso, es un string y la masa, el vector posición y el tensor de inercia son clases simbólicas de Matlab. Esto pone de manifiesto que las propiedades de un objeto pueden ser otros objetos. La base cinemática de cada uno de los sólidos rígidos es una clase que se creará más adelante para representar el concepto matemático de base.

Cada uno de los objetos tiene a su vez métodos, es decir, funciones que pueden realizar los objetos. En el caso de la clase “Sólido Rígido” se presenta una lista de algunos de los métodos que se pueden integrar:

- obtener velocidad instantanea del sólido rígido,
- obtener aceleración instantanea del sólido rígido,
- obtener velocidad angular del sólido rígido,
- obtener componentes parciales de la velocidades del sólido rígido,
- obtener energía cinética del sólido rígido
- calcular matriz  $[A^i]$ .

Todos estos métodos utilizarán las propiedades del objeto y otras variables para dar una salida. En el caso del cálculo de las parciales de la velocidad del sistema hará falta obtener el vector posición del sólido rígido y posteriormente realizar las correspondientes derivadas parciales respecto a las coordenadas generalizadas  $q_i$ . Las entradas de este método son por lo tanto la propiedad  $\vec{r}_G$  y las variables externas (las coordenadas generalizadas  $\mathbf{q}$ ).

## 3.2 Marco práctico: Espacio de nombres

El espacio de nombres o “*namespace*” en Matlab son carpetas que contienen definiciones de clases, archivos de funciones y otros espacios de nombre. Estos espacios de nombre ayudan a organizar mejor el código, y en el caso de la librería dinámica, a que las definiciones internas no tengan problemas de compatibilidad con Matlab. Así, si creamos una clase llamada “System” fuera del espacio de nombres Matlab podría tener problemas debido a que ya tiene definidas funciones o clases que se denominan “System”.

En realidad, el uso de un espacio de nombres en esta librería trata de evitar el uso de variables globales que pueden crear interferencias dentro de los diferentes espacios de trabajo de Matlab.

En Matlab para definir un espacio de nombres se debe nombrar la carpeta con un signo “+” delante del nombre. La Figura 3.1 muestra el namespace utilizado para la librería dinámica presentada [13].

.git	22/04/2024 12:16	Carpeta de archivos	
+Dynamic_Library	14/05/2024 10:13	Carpeta de archivos	
fun_handles	22/04/2024 12:16	Carpeta de archivos	
resources	22/04/2024 12:16	Carpeta de archivos	

Figura 3.1: Espacio de nombres “Dynamic\_Library”

La librería se ha dividido en varias clases. La principal es la clase “System”, que se encarga de manejar todas las demás clases. Esta clase tiene la idea de ser un sistema físico y, por lo tanto, diferentes objetos de esta clase definen diferentes sistemas. En un mismo “*script*” se puede realizar el estudio tanto de un péndulo simple, un péndulo doble así como de la torre grúa. La clase “System” se presenta en la Figura 3.2.

+Classes	22/04/2024 12:16	Carpeta de archivos	
System.m	22/04/2024 11:17	MATLAB Code	55 KB

Figura 3.2: Clase “System” del espacio de nombres “Dynamic\_Library”

El resto de las clases se encuentran en otro sub-espacio de nombres denominado “Classes”. Este contiene las clases que definen las bases, los puntos, las acciones (torsesores) y los sólidos rígidos. En las Figuras 3.3 y 3.4 aparecen los directorios y nombres de las clases.

@Action	22/04/2024 12:16	Carpeta de archivos	
@Base	22/04/2024 12:16	Carpeta de archivos	
@Point	22/04/2024 12:16	Carpeta de archivos	
@Rigid_Body	22/04/2024 12:16	Carpeta de archivos	

Figura 3.3: Sub-espacio de nombres “Classes”

Rigid_Body.m	18/04/2024 9:05	MATLAB Code	6 KB
--------------	-----------------	-------------	------

Figura 3.4: Clase “Rigid\_Body” del Sub-Espacio de Nombres “Classes”

La clase “System” que pertenece al espacio de nombres principal se denominará como clase principal y el resto de clases que se encuentran en el espacio de nombres secundario se denominarán sub-clases.

### 3.3 Marco práctico: Implementación en Matlab de sub-clases

Las sub-clases se definen (cada una) por separado y se basan en los principios básicos ya definidos en la sección del cálculo de las ecuaciones del movimiento. Aunque todas son diferentes, tienen elementos en común que cabe destacar.

Cada una de las clases tiene un método denominado “Get\_Info” con el cual se obtiene información relevante sobre el objeto creado. Si por ejemplo se ha creado un punto P, el método “Get\_Info” da información sobre el nombre del punto o sus coordenadas desde la base canónica. Si bien en Matlab es posible acceder manualmente a estos parámetros es una buena práctica de programación negar el acceso directo del usuario a estas variables, debido a que puede ocasionar problemas posteriormente. Es más común generar funciones que modifiquen estos parámetros de una forma controlada por el programador. Esto también permite que en caso de fallo el propio código sea capaz de indicar el

motivo e incluso ofrecer sugerencias. Debido a la limitación de tiempo este tipo de mensajes de errores no se han introducido pero si que se ha realizado una estructura para su introducción posterior.

Todas las clases tienen un constructor y, en algunas de ellas, este constructor es posible inicializarlo de diferentes formas. Una base puede definirse por los vectores en su matriz de cambio de base manualmente, o que esta matriz la construya la propia librería. Las distintas formas de crear un objeto se especifica en cada una de las clases que se van a crear.

Debido a que el código de la librería es extenso, unicamente se presentarán ejemplos de códigos de la clase “Base”. Para más información sobre el resto de sub-clases se recomienda el estudio del código en cuestión [14].

### 3.3.1 Clase: Base

La clase “Base” crea el ente matemático de base vectorial ya definido en el marco teórico cinemático. Para su inicialización es necesario un nombre de base, nombre de los ejes de la base y la matriz de cambio de base desde la base canónica.

#### Propiedades

- Base\_Name
- Axis\_Labels
- Base\_Matrix\_From\_Canonical

```
1  properties(Access = public)
2      Base_Name
3      Axis_Labels
4      Base_Matrix_From_Canonical
5  end
```

Código 3.1: Propiedades clase “Base”

#### Métodos

#### Constructor

El constructor de “Base” puede inicializar el objeto con diferentes tipos de estructuras de entrada. Esto tiene la finalidad de que la base pueda ser creada a través de una matriz de cambio de base, que se introduce manualmente, o a través de la rotación respecto a una base ya creada. Aunque es posible introducir manualmente matrices de cambio de base, en esta librería estas deben ser ortonormales, debido a que la programación se ha realizado con la transpuesta de las matrices en vez de la inversa. El Cuadro 3.2 muestra las tres formas de inicializar el objeto “Base” y el Código 3.2 muestra la programación del constructor.

Base (constructor)			
Opción	Entradas	Salidas	Descripción
Default	Name	Obj.Base	El objeto devuelto tiene una base $\vec{e}_1 = [1, 0, 0]^T$ , $\vec{e}_2 = [0, 1, 0]^T$ y $\vec{e}_3 = [0, 0, 1]^T$ . Las etiquetas por defecto de los ejes son $[x, y, z]$ .
Only Labels	Name Labels	Obj.Base	El objeto devuelto tiene una base $\vec{e}_1 = [1, 0, 0]^T$ , $\vec{e}_2 = [0, 1, 0]^T$ y $\vec{e}_3 = [0, 0, 1]^T$ . Las etiquetas por defecto son las definidas por Labels.
Rotation	Name Labels Father Base Angle Of Rotation	Obj.Base	El objeto devuelto tiene una base que es la rotación respecto a una base padre definida por Father Base. El ángulo de rotación viene definido por Angle of Rotation.

Cuadro 3.2: Constructor de la clase “Base”.

```

1  function obj = Base(Name, varargin)
2  default_axis_labels = [str2sym("x");str2sym("y");str2sym("z")];
3  default_base = sym([1 0 0; 0 1 0; 0 0 1]);
4  default_axis = "1";
5  default_angle = 0;
6  %Function to test if variable is sym class.
7  issym = @(x) isequal(class(x),"sym");
8  issym_3_1 = @(x) isequal(class(x),"sym") && isequal(size(x),[3 1]);
9  issym_or_base = @(x) issym(x) || isequal(class(x),"Dynamic_Library.Classes.Base");
10 ;
11 %Function to test if variable is string "1", "2" or "3".
12 %Defined for axis.
13 isaxis = @(x) isequal(x,"1") || isequal(x,"2") || isequal(x,"3");
14
15 %Function to test if a variable is either a sym or a
16 %numeric type.
17 issym_or_numeric = @(x) issym(x) || isnumeric(x);
18
19 p = inputParser;
20 addRequired(p,'name',@isstring);
21 addParameter(p,'axis_labels',default_axis_labels,issym_3_1);
22 addParameter(p,'father_base',default_base,issym_or_base);
23 addParameter(p,'axis_rotation',default_axis,isaxis);
24 addParameter(p,'angle_rotation',default_angle,issym_or_numeric);
25
26 parse(p,Name,varargin{:});
27
28 obj.Base_Name = p.Results.name;
29 obj.Axis_Labels = p.Results.axis_labels;
30
31 switch(class(p.Results.father_base))
32     case "sym"
33         father_M_B_E = p.Results.father_base;
34         M_BF_BM = Dynamic_Library.Classes.Base.Rotation_Change_Base_Matrix(p.
Results.axis_rotation, p.Results.angle_rotation);
35         obj.M_B_E = (father_M_B_E.'*M_BF_BM).';
36     case "Dynamic_Library.Classes.Base"
37         father_M_B_E = p.Results.father_base.Get_Info("Base_Matrix_From_Canonical
");
38         M_BF_BM = Dynamic_Library.Classes.Base.Rotation_Change_Base_Matrix(p.
Results.axis_rotation, p.Results.angle_rotation);

```

```

38         obj.M_B_E = (father_M_B_E.*M_BF_BM).';
39     otherwise
40         error("Father base needs to be specified as a 3x3 'sym' matrix or an
object of the 'Dynamic_Library.Classes.Base' class");
41     end
42 end

```

Código 3.2: Programación de la clase “Base”

## Información

La información de la clase son sus propiedades por lo tanto el método “Get\_Info” se centra en poner a disposición del usuario estos valores. Este método se puede llamar de varias formas como señala el Cuadro 3.3 y, se pueden añadir en un futuro más opciones. El Código 3.3 muestra la programación de este método en Matlab.

Get_Info			
Opción	Entradas	Salidas	Descripción
Name	Obj.Base “Name”	String	Se obtiene el nombre de la base.
Axis_Labels	Obj.Base “Axis_Labels”	Sym $[3 \times 1]$	Se obtiene el array con los valores simbólicos de las etiquetas de los ejes.
Base_Matrix_From_Canonical	Obj.Base “Base_Matrix_From_Canonical”	Sym $[3 \times 3]$	Se obtiene la matriz simbólica de los vectores de la base.

Cuadro 3.3: Método para obtener información de la clase “Base”

```

1 function out = Get_Info(obj, Info)
2 %GET_INFO Get properties of an "Base" object.
3 switch(Info)
4     case "Name"
5         out = obj.Base_Name;
6     case "Axis_Labels"
7         out = obj.Axis_Labels;
8     case "Base_Matrix_From_Canonical"
9         out = obj.Base_Matrix_From_Canonical;
10 end
11 end

```

Código 3.3: Programación del método “Get\_Info” de “Base”

## Matriz de cambio de base

Para indicar las rotaciones entre diferentes bases se implementa un método estático que es capaz de calcular la matriz de cambio de base entre estas dos. Se denomina estático debido a que este método no se aplica a ningún objeto en particular, si no que, se puede utilizar indistintamente por cualquiera



de ellos sin que se involucren sus propiedades. El Cuadro 3.4 muestra las entradas de este método y las salidas. El Código 3.4 muestra como se definen las matrices de cambio de base para la rotación.

Rotation_Matrix (método estático)			
Opción	Entradas	Salidas	Descripción
Matriz de rotación	Eje Ángulo	Sym[3 × 3]	Se obtiene la matriz de rotación en el eje 1, 2 o 3 con el ángulo introducido.

Cuadro 3.4: Método estático “Rotation\_Matrix” para obtener las matrices rotación.

```

1  function M_BF_BM = Rotation_Change_Base_Matrix(axis, angle)
2      % FUNCTION TO GET THE ROTATION MATRIX OF CERTAIN ANGLE.
3      % ANGLE IN RADIANS
4      switch(axis)
5          case "1"
6              M_BF_BM = sym([1, 0, 0;
7                           0, cos(angle), -sin(angle);
8                           0, sin(angle), cos(angle)]);
9          case "2"
10             M_BF_BM = sym([cos(angle), 0, sin(angle);
11                           0, 1, 0;
12                           -sin(angle), 0, cos(angle)]);
13          case "3"
14             M_BF_BM = sym([cos(angle), -sin(angle), 0;
15                           sin(angle), cos(angle), 0;
16                           0, 0, 1]);
17          otherwise
18             error("Axis of rotation has not been specified correctly.
19             Please use string notation for the axis of rotation.");
20             end
21     end

```

Código 3.4: Programación del método estático de matriz de rotación.

### 3.3.2 Clase: Point

La clase “Point” es el siguiente ente cinemático básico para el estudio de los sistemas. De igual forma que la clase “Base” esta se conforma de un constructor y un método “Get\_Info” para obtener las propiedades del punto. El Código 3.5 y los Cuadros 3.5 y 3.6 muestran la configuración de esta clase.

#### Propiedades

- Point\_Name
- Point\_Coordinates\_From\_Canonical

```

1  properties(Access = public)
2      Point_Name
3      Point_Coordinates_From_Canonical
4  end

```

Código 3.5: Propiedades clase “Point”

## Métodos

### Constructor

Point (constructor)			
Opción	Entradas	Salidas	Descripción
Default	Name	Obj.Point	El punto devuelto tiene las coordenadas $[0, 0, 0]$ .
Coordenadas Relativas	Name	Obj.Point	El punto devuelto se posiciona en unas coordenadas relativas a la base padre y punto padre especificados.
	Father_Base		
	Father_Point		
	Relative_Coordinates		

Cuadro 3.5: Constructor de la clase “Point”

### Información

Se puede observar que el método ofrece la posibilidad de calcular la velocidad del punto. Aunque esto no es una propiedad del punto es posible programar el cálculo de diferentes magnitudes a través de las propiedades del objeto.

Get_Info			
Opción	Entradas	Salidas	Descripción
Name	Obj.Point “Name”	String	Se obtiene el nombre de la base.
Coordinates	Obj.Point “Point_Coordinates_From_Canonical”	Sym $[3 \times 1]$	Se obtiene el array con los valores simbólicos de las etiquetas de los ejes.
Velocity	Obj.Point “Point_Velocity_Coordinates_From_Canonical”	Sym $[3 \times 1]$	Se obtiene la matriz simbólica de los vectores de la base.

Cuadro 3.6: Método para obtener información de la clase “Point”

### 3.3.3 Clase: Rigid\_Body

La clase “Rigid\_Body” se crea para definir la parte dinámica del sistema. Esta clase se compone de un objeto “Base” y otro objeto “Point” además de las propiedades necesarias para el funcionamiento.

### Propiedades

- Rigid\_Body\_Name
- G\_Point\_From\_Canonical
- Rigid\_Body\_Base

- Mass
- Inertial\_Tensor

```

1  properties(Access = public)
2      Rigid_Body_Name
3      G_Point_From_Canonical
4      Rigid_Body_Base
5      Mass
6      Inertial_Tensor
7  end

```

Código 3.6: Propiedades clase “Rigid\_Body”

## Métodos

### Constructor

Por el momento el constructor del sólido rígido tiene únicamente dos formas de inicializarse: el modo por defecto y el sólido rígido con propiedades como muestra el Cuadro 3.7. Una futura incorporacion puede ser añadir una partícula, debido a que esta se puede considerar como un sólido rígido con tensor de inercia nulo.

Rigid_Body (Constructor)			
Opción	Entradas	Salidas	Descripción
Default	Name	Obj.Rigid_Body	El sólido rígido devuelto tiene una base $\vec{e}_1 = [1, 0, 0]^T$ , $\vec{e}_2 = [0, 1, 0]^T$ y $\vec{e}_3 = [0, 0, 1]^T$ . Las etiquetas por defecto de los ejes son $[x, y, z]$ . La masa por defecto es $m_{Name}$ y el tensor de inercia es $\begin{bmatrix} I_{Name_{11}} & I_{Name_{12}} & I_{Name_{13}} \\ I_{Name_{12}} & I_{Name_{22}} & I_{Name_{23}} \\ I_{Name_{13}} & I_{Name_{23}} & I_{Name_{33}} \end{bmatrix}$ .
	Name CDG(Obj.Point)		
Default	Base(Obj.Base) Masa Tensor de Inercia	Obj.Rigid_Body	Se obtiene el objeto “Rigid_Body” con los parámetros incluidos.

Cuadro 3.7: Constructor de la clase “Rigid\_Body”

### Información

El método “Get\_Info” de la clase “Rigid\_Body” es el que más opciones tiene hasta la fecha. Esto es debido a que es necesario un amplio arsenal de herramientas para construir las matrices que representan las ecuaciones del sistema. El Cuadro 3.8 muestra algunas de las salidas que se puede obtener a través de esta función, siendo cada una de ellas relevante para la obtención final de las ecuaciones del movimiento.

Get_Info			
Opción	Entradas	Salidas	Descripción
Name	Obj.Rigid_Body "Name"	String	Se obtiene el nombre del sólido rígido.
G_Point_From_Canonical	Obj.Rigid_Body "G_Point_From_Canonical"	Sym $[3 \times 1]$	Se obtiene el vector de posición del CGD del sólido rígido.
Rigid_Body_Base	Obj.Rigid_Body "Rigid_Body_Base"	Obj.Base	Se obtiene el objeto Base del sólido rígido.
Mass	Obj.Rigid_Body "Mass"	Sym or double	Se obtiene el valor numérico o simbólico de la masa del sólido rígido.
Inertial_Tensor	Obj.Rigid_Body "Inertial_Tensor"	Sym $[3 \times 3]$	Se obtiene el tensor de inercia del sólido rígido.
Absolute_Velocity	Obj.Rigid_Body "Absolute_Velocity"	Sym $[3 \times 1]$	Se obtiene el vector velocidad del CGD del sólido rígido.
Absolute_Acceleration	Obj.Rigid_Body "Absolute_Acceleration"	Sym $[3 \times 1]$	Se obtiene el vector aceleración del CGD del sólido rígido.
Absolute_Angular_Velocity	Obj.Rigid_Body "Absolute_Angular_Velocity"	Sym $[3 \times 1]$	Se obtiene el vector velocidad angular del sólido rígido.
Virtual_Velocities	Obj.Rigid_Body "Virtual_Velocities" $\mathbf{q}$	Sym	Se obtienen los coeficientes $b_i$ y $c_i$ de las velocidades virtuales.
Kinetic_Energy	Obj.Rigid_Body "Kinetic_Energy"	Sym	Se obtiene la expresión simbólica de la energía cinética del sólido rígido.

Cuadro 3.8: Método para obtener información de la clase "Rigid\_Body"

### 3.3.4 Clase: Accion

Las acciones son conjuntos de fuerzas y momentos (torsores) que actúan en un punto del sólido. Esta clase representa los torsores que afectan a los sólidos rígidos. El Código 3.7 muestra las propiedades que se definen para esta clase.

#### Propiedades

- Action\_Name
- Action\_Point
- Action\_Base
- Action\_Vector

```
1 properties(Access = public)
2     Action_Name
3     Action_Point
4     Action_Base
5     Action_Vector
6 end
```

Código 3.7: Propiedades clase “Action”

#### Métodos

##### Constructor

El constructor se puede inicializar con una acción nula en el punto  $[0, 0, 0]^T$  o a través de la introducción manual de parámetros.

Point (constructor)			
Opción	Entradas	Salidas	Descripción
Default	Name	Obj.Action	La acción devuelta esta aplicada en el punto $[0, 0, 0]^T$ y tiene un torsor $[0, 0, 0, 0, 0, 0]^T$ .
Definición relativa	Name Point Base Vector	Obj.Action	La acción devuelta se posiciona en el punto y la base descritos con el torsor introducido en vector.

Cuadro 3.9: Constructor de la clase “Action”

#### Información

Get_Info			
Opción	Entradas	Salidas	Descripción
Name	Obj.Action "Name"	String	Se obtiene el nombre de la acción.
Point	Obj.Action "Action_Point"	Sym $[3 \times 1]$	Se obtiene el objeto "Point" donde se aplica la acción.
Base	Obj.Action "Action_Base"	Sym $[3 \times 1]$	Se obtiene la base en la que se presenta el tursor de la acción.
Vector	Obj.Action "Action_Vector"	Sym $[3 \times 1]$	Se obtienen los componentes del vector que representa el tursor.

Cuadro 3.10: Método para obtener información de la clase "Action"

### 3.3.5 Clase: Sistema

#### Motivo

Como se ha mostrado en la Figura 3.2, la clase "System" se mantiene por encima de todas las demás, de hecho se crea otro espacio de nombres denominado "Classes" para diferenciar muy bien entre esta clase y el resto.

La importancia de la clase "System" radica en que es la encargada de manejar el resto de clases. A través de las clases presentadas hasta el momento el usuario es capaz de realizar uno o varios scripts para generar un sistema por su cuenta. El Código 3.8 muestra un ejemplo para la creación de una base "B\_0" y un punto "P\_0" canónicos. A partir de esta base y estos puntos se puede ir generando manualmente diferentes bases y diferentes puntos hasta formar un sistema.

```

1 B_0 = Dynamic_Library.Classes.Base( ...
2 "Canonical", ...
3 P_0 = "axis_labels",[str2sym("x");str2sym("y");str2sym("z")]);
4 Dynamic_Library.Classes.Point("Canonical");

```

Código 3.8: Creación manual de sistema.

Los inconvenientes de este método son claros:

- Existirán problemas si se quieren estudiar dos o más sistemas en el mismo espacio de trabajo en Matlab. Si, por ejemplo, se desea estudiar el mismo sistema con o sin rozamiento, para que no existan problemas, las bases, puntos y demás objetos de cada uno de los sistemas deberán tener diferentes nombres.
- Se permite al usuario acceder a las clases más fundamentales de la librería. Esto no es una buena práctica a realizar en proyectos debido a que, en caso de fallo, resulta más complicado encontrar el motivo.

Por otro lado, el uso de librería como "System", que se encuentra en el *namespace* principal, tiene las siguientes ventajas:

- El código se encuentra más estructurado debido a que la clase final utiliza clases más fundamentales para el trabajo.
- Debido a que las clases deben trabajar unas con otras, es la clase "System" la que maneja esas compatibilidades, no es el usuario el que las debe tener en cuenta. Esto permite una adaptación más rápida del técnico a las herramientas.

Si bien este tipo de programación ayuda con todo lo mencionado anteriormente también es más laborioso el diseño de la estructura. Aunque, como ya se ha comentado, debido a las ventajas que presenta este método es el escogido.

## Estructura

La clase “System” es la encargada de crear los objetos “Base”, “Point”, “Rigid\_Body” y “Action”. Además de crearlos lleva un registro de cada uno de ellos y los guarda dentro del objeto de la clase “System” creado. El Código 3.9 muestra la creación de un sistema para un péndulo simple. La propia clase “System” genera un punto y una base canónicos, pero como práctica habitual se recomienda crear las propias a partir de estas.

```
1 sys_pendolo = Dynamic_Library.System("Pendulo_Simple");
2 sys_pendolo.Create_New_Point("N", "Canonical", sym([0;0;0]));
3 sys_pendolo.Create_New_Base("B_0", 'axis_labels', [str2sym("x_0"); str2sym("y_0");
4 str2sym("z_0")]);
5 sys_pendolo.Create_New_Coordinate_System("C_0", "N", "B_0");
```

Código 3.9: Creación del objeto de sistema “Pendulo\_Simple”

La Figura 3.5 muestra el espacio de trabajo tras ejecutar dicho código. Las nueva base “B\_0” y el punto “N” se encuentran dentro del objeto “sys\_pendolo”.

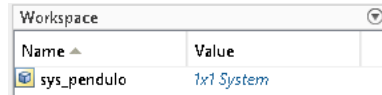


Figura 3.5: Espacio de trabajo péndulo simple.

La estructura interna de la clase “System” se muestra en el diagrama de la Figura 3.6. Como se puede ver, no involucra unicamente a las sub-clases si no que también tiene otros parámetros como por ejemplo las coordenadas generalizadas. Cada uno de estos subsistemas se gestiona a través de una tabla generada en Matlab y esta, es única para cada subsistema.

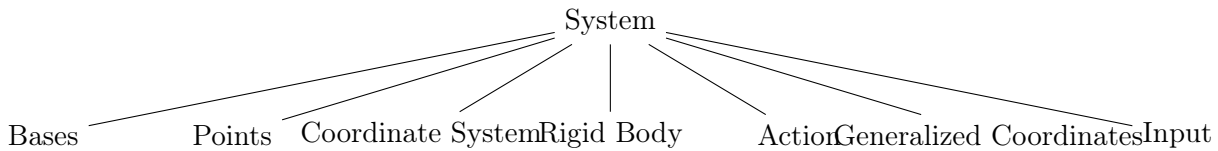


Figura 3.6: Diagrama de la estructura interna de la clase “System”.

Cada una de las tablas contiene la información necesaria para ejecutar los comandos del sistema. El subsistema de coordenadas generalizadas llevará un registro de cada coordenada generalizada que el usuario añada al sistema. El Código 3.10 muestra la tabla “System\_Generalized\_Coordinates” antes y después de añadir la coordenada generalizada  $\theta(t)$  al sistema del péndulo simple.

```
1 %%%%%%%%%%%%%SCRIPT%%%%%%%%%%%%%
2 sys_pendolo.System_Generalized_Coordinates
3 %%%%%%%%%%%%%TERMINAL%%%%%%%%%%%%%
4 ans =
5     {"Canonical"}    {[NaN]}
6 %%%%%%%%%%%%%SCRIPT%%%%%%%%%%%%%
7 sys_pendolo.Create_New_Generalized_Coordinate("theta", str2sym("theta(t)"));
8 sys_pendolo.System_Generalized_Coordinates
9 %%%%%%%%%%%%%TERMINAL%%%%%%%%%%%%%
10 ans =
11     {"Canonical"}    {[NaN]}
12     {"theta"}        {[theta(t)]}
```

Código 3.10: Creación de la coordenada generalizada  $\theta(t)$  en el péndulo simple.

De forma análoga, las tablas “System\_Bases” y “System\_Points” contienen los objetos de las bases y los puntos creados por el usuario. El Código 3.11 muestra estas tablas en el sistema creado para el péndulo simple.

```

1  %%%%%%%%%%%%%SCRIPT%%%%%%%%%%%%%
2  sys_pendolo.Create_New_Point("N","Canonical",sym([0;0;0]));
3  sys_pendolo.Create_New_Base("B_0",'axis_labels',[str2sym("x_0");str2sym("y_0");
4  str2sym("z_0")]);
5  sys_pendolo.Create_New_Base("Bar", ...
6  'axis_labels',[str2sym("B_1");str2sym("B_2");str2sym("B_3
7  ")]), ...
8  'father_base',"B_0", ...
9  'angle',str2sym("theta(t)", ...
10 'axis_rotation',"1");
11 sys_pendolo.Create_New_Coordinate_System("Bar_CS_N","N","Bar");
12 sys_pendolo.Create_New_Point("G","Bar_CS_N",[0;0;-str2sym("l")/2]);
13 sys_pendolo.System_Points
14 sys_pendolo.System_Bases
15 %%%%%%%%%%%%%TERMINAL%%%%%%%%%%%%%
16 ans =
17 3x2 cell array
18 {"Canonical"}    {1x1 Dynamic_Library.Classes.Base}
19 {"B_0"}          {1x1 Dynamic_Library.Classes.Base}
20 {"Bar"}          {1x1 Dynamic_Library.Classes.Base}
21
22 ans =
23 3x2 cell array
24 {"Canonical"}    {1x1 Dynamic_Library.Classes.Point}
25 {"N"}            {1x1 Dynamic_Library.Classes.Point}
26 {"G"}            {1x1 Dynamic_Library.Classes.Point}

```

Código 3.11: Creación de bases en sistema péndulo simple.

Cada uno de estos subsistemas contiene funciones que hacen que unos interactúen con otros. De esta forma un sistema de coordenadas se genera a través de un punto y una base y, este mismo sistema de coordenadas sirve para generar nuevos puntos. El Código 3.12 muestra la función “Create\_New\_Point” la cual crea un punto a partir de un sistema de coordenadas. Cabe destacar que aunque no exista una clase específica para los sistemas de coordenadas la clase “System” integra este concepto en su interior.

```

1  function Create_New_Point(obj, Name, Coordinate_System, Coordinates)
2  ...
3  Coordinate_System_Info = obj.Get_Coordinate_System_Info("
4  System_Coordinate_Systems",'coordinate_system',Coordinate_System);
5  Point_Name = Coordinate_System_Info{2};
6  Base_Name = Coordinate_System_Info{3};
7  Point_Object = obj.Get_Point_Info("System_Points",'point',Point_Name)
8  ;
9  Point_Object = Point_Object{2};
10 Base_Object = obj.Get_Base_Info("System_Bases",'base',Base_Name);
11 Base_Object = Base_Object{2};

```

Código 3.12: Función que genera un punto a través de un sistema de coordenadas.

Las clases en Matlab tienen propiedades y métodos que, a su vez, se utilizan para definir las propiedades y los métodos de los subsistemas. Cada uno de ellos tiene la estructura de programación que se muestra en el Código 3.13. Aunque hay algunos que no utilicen propiedades protegidas o métodos privados, de esta forma todos pueden tratarse de la misma manera.

```

1  %% BASES
2  % PROPERTIES
3  % PRIVATE
4  properties(Access = private)
5
6  end
7  % PROTECTED
8  properties(Access = protected)

```



```

9
10     end
11
12     % PUBLIC
13     properties(Access = public)
14
15     end
16 % METHODS
17
18     methods(Access = private)
19
20     end
21     % PROTECTED
22     methods(Access = protected)
23
24     end
25
26     % PUBLIC
27     methods(Access = public)
28
29     end
30     methods(Static, Access = public)
31
32     end

```

Código 3.13: Estructura de programación de los subsistemas de la clase “System”.

## Generación de funciones

Para cada una de las componentes parciales de la velocidad se puede realizar una función en Matlab que devuelva el valor de esas componentes. Hasta el momento, la librería estaba pensada únicamente para la resolución simbólica pero la inclusión de este tipo de funciones hace que se de el paso hacia la resolución numérica.

La idea general es sustituir los valores simbólicos por unos numéricos, algo que la clase “sym” puede realizar a través de la función “subs [15]”. Este método, si bien hace directo el paso de simbólico a numérico, se ha comprobado que tiene un costo computacional muy alto. Este problema se ha solventado con la creación de funciones de Matlab generadas automáticamente, las cuales ya han realizado de antemano la sustitución de los valores simbólicos por numéricos.

En Matlab existen dos alternativas para generar funciones automáticamente:

- matlabFunction [16]
- odeFunction [17]

Tras comprobar el funcionamiento de ambas, se ha visto que la función “matlabFunction” tiene problemas con el uso de variables dependientes de otras. En caso de funciones que representan la dinámica de diversos sistemas, como es el caso de los sistemas mecánicos, muchas de las variables dependen del tiempo. Así, si se tiene un péndulo simple, la coordenada generalizada del ángulo de giro  $\theta(t)$  depende del tiempo  $t$  y esta función genera problemas a la hora de sustituir los valores numéricos. Finalmente, se ha optado por usar “odeFunction”.

Como ambas funciones pueden crear archivos “.m” de funciones de Matlab o punteros a funciones, para el uso en la librería no existe ningún inconveniente al usar “odeFunction”. El Código 3.14 muestra el ejemplo de como se generan las funciones que calculan las componentes parciales de las velocidades. Estas funciones posteriormente se guardan en la carpeta “fun\_handles” con sus respectivos nombres para cada sólido y cada coordenada generalizada.

```

1     virtual_velocity_uv{i,3} = odeFunction(vv_uv,[v;u], ...
2     "File", "fun_handles/Virtual_Velocity_Components_uv_"+Name+"_"+string(i), ...
3     "Comments","Virtual Velocity Components: Rigid Body: "+Name+" "+ string(q(i)));
4     virtual_velocity_uv{i,4} = dt_vv;
5     virtual_velocity_uv{i,5} = odeFunction(d_vv_uv,[v;u], ...
6     "File", "fun_handles/d_Virtual_Velocity_Components_uv_"+Name+"_"+string(i), ...

```

```
"Comments","Virtual Velocity Components: Rigid Body: "+Name+" "+ string(q(i)));
```

Código 3.14: Generación de funciones Matlab para el cálculo de las componentes de las parciales  $b_i$  y  $c_i$  de las velocidades.

La Figura 3.7 muestra algunas componentes para el sólido rígido “Jib” generadas a través del Código 3.14.







	d_Virtual_Velocity_Components_uv_Jib_1....	20/05/2024 18:52	MATLAB Code	1 KB
	d_Virtual_Velocity_Components_uv_Jib_2....	20/05/2024 18:52	MATLAB Code	1 KB
	d_Virtual_Velocity_Components_uv_Jib_3....	20/05/2024 18:52	MATLAB Code	1 KB
	Virtual_Velocity_Components_uv_Jib_1.m	20/05/2024 18:52	MATLAB Code	1 KB
	Virtual_Velocity_Components_uv_Jib_2.m	20/05/2024 18:52	MATLAB Code	1 KB
	Virtual_Velocity_Components_uv_Jib_3.m	20/05/2024 18:52	MATLAB Code	1 KB

Figura 3.7: Funciones Matlab de las componentes  $b^i$ ,  $b_v^i$ ,  $c^i$  y  $c_v^i$  del sólido “Jib” para las tres primeras coordenadas generalizadas del sistema.

Esta generación de funciones es el pilar fundamental de la función “unify\_system”, la cual, realiza la unión numérica de sistemas mecánicos. Esta unión se basa en ensamblar la matriz  $[A^i]$  de cada uno de los sólidos para calcular finalmente la del sistema completo, así como realizar el cálculo de todas las fuerzas generalizadas  $Q$ .

### Diferencia entre Action e Input

Llegados a este punto se debe aclarar diferenciación entre una acción y una señal de entrada. A fin de cuentas, en la resolución simbólica, ambas se consideran como una acción externa que debe ser generalizada. Esta idea surge de la necesidad de realizar simulaciones numéricas del sistema y, por lo tanto, la necesidad del usuario de especificar la entrada en cada instante de tiempo. Dentro de la clase “System” ambos conceptos son completamente análogos y se construyen encima de la clase “Action”

## 4 Librería Dinámica: Ejemplos de uso

Esta sección hará uso de la librería “Dynamic\_Library” para obtener las ecuaciones del movimiento de: un péndulo simple y una torre grúa. La finalidad de estos ejemplos es proporcionar una guía con la que el usuario se familiarice con las funciones y comandos de la librería a la vez que aprende la estructura de programación necesaria para obtener las ecuaciones del movimiento.

### 4.1 Péndulo simple

El péndulo simple es un sistema ya estudiado en la Sección 2.4.4. Este ejemplo ayuda a validar el modelo ya presentado mostrando los pasos necesarios para formar el sistema en Matlab.

#### 4.1.1 Definición del sistema

En esta sección se hará uso de la librería dinámica para el cálculo del modelo de sólido rígido del péndulo simple presentado en la Figura 2.13.

##### Parámetros

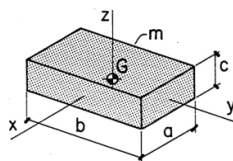
El Código 4.1 muestra la inicialización de algunos de los parámetros que se van a ir utilizando durante la creación del sistema. Se inicializa la gravedad  $g$ , la distancia del eje al centro de masas de la barra  $l_{CM}$  y la masa del péndulo  $m$ . Con estos parámetros se construye el tensor de inercia  $\mathcal{I}_{Bar}$  en el centro de gravedad  $G$  a través del valor tabulado mostrado en la Figura 4.1. Finalmente se inicializa la variable  $\theta$  que representa la única coordenada generalizada del sistema.

```
1  % PARAMETROS
2  g = str2sym("g");
3  l_CM = str2sym("l_CM");
4  m = str2sym("m");
5  I_11_Bar = 1/12*m*(2*l_CM)^2;
6  I_Bar = [I_11_Bar 0 0;
7           0 0 0;
8           0 0 0];
9  theta = str2sym("theta(t)");
```

Código 4.1: Inicialización parámetros péndulo simple.

Como se puede comprobar todos los parámetros y variables se inicializan a través de la clase simbólica de Matlab. Aunque la librería acepta valores numéricos, se realiza de esta forma para obtener las ecuaciones del movimiento simbólicas.

Paralelepípedo  
rectangular macizo  
y homogéneo



$$\Pi_G \{I_{xx} = (1/12)m(b^2 + c^2)\}$$

Figura 4.1: Momentos de inercia de paralelepípedo rectangular macizo y homogéneo [3].

## Inicialización del sistema “sys\_péndulo”

Para la inicialización del sistema se crea el objeto “sys\_péndulo” de la clase System. La clase system (por defecto) crea una base y un punto canónicos, pero una buena práctica es crear los propios que simbolicen el sistema a estudiar. Para comenzar a definir puntos se crea también un sistema de coordenadas. El Código 4.2 muestra la creación del punto “N” canónico, la base “B\_0” canónica y el sistema de coordenadas “C\_0” asociado a al punto y la base “N” y “B\_0”.

```
1 % Inicializamos el sistema
2 sys_pendolo = Dynamic_Library.System("Pendulo_Simple");
3 %Creamos el origen de nuestro sistema.
4 sys_pendolo.Create_New_Point("N", "Canonical", sym([0;0;0]));
5 sys_pendolo.Create_New_Base("B_0", 'axis_labels', [str2sym("x_0"); str2sym("y_0");
6 str2sym("z_0")]);
7 sys_pendolo.Create_New_Coordinate_System("C_0", "N", "B_0");
```

Código 4.2: Inicialización del sistema del péndulo simple.

## Base móvil y centro de masas $G$

La rotación del péndulo se definirá a través de una base móvil respecto a la referencia inercial. El giro se realiza en el eje x (o en el primer eje según la nomenclatura de la librería) un ángulo  $\theta$  y se le da el nombre de “Bar”. Esta base, junto con el punto  $N$  creado anteriormente formarán el sistema de coordenadas “Bar\_CS\_N”. El Código 4.3 muestra lo explicado anteriormente y añade la creación del punto  $G$  a una distancia  $l_{CM}$  de  $N$ .

```
1 sys_pendolo.Create_New_Generalized_Coordinate("theta", theta);
2 sys_pendolo.Create_New_Base("Bar", ...
3 'axis_labels', [str2sym("B_1"); str2sym("B_2"); str2sym(
4 "B_3")], ...
5 'father_base', "B_0", ...
6 'angle', theta, ...
7 'axis_rotation', "1");
8 sys_pendolo.Create_New_Coordinate_System("Bar_CS_N", "N", "Bar");
9 sys_pendolo.Create_New_Point("G", "Bar_CS_N", [0;0;-l_CM]);
```

Código 4.3: Creación de base asociada al sólido Bar y el centro de masas  $G$

## Sólido rígido “Bar”

Con los parámetros cinemáticos ya formados, es hora de crear el sólido rígido referente a la barra del péndulo. Este sólido rígido se compone de un centro de masas en  $G$ , una base “Bar” asociada a el, una masa  $m$  y un tensor de inercia  $\mathcal{I}_{Bar}$ . El tensor de inercia debe estar calculado en el punto  $G$ , como ya se ha mencionado anteriormente en la parte teórica y la formación de la librería. El Código 4.4 muestra la creación de un objeto “Rigid\_Body” con el nombre “Bar”. Este objeto se forma a incluyendo el punto “G”, la base “Bar”, la masa “m” y el tensor de inercia “I\_Bar”.

```
1 sys_pendolo.Create_New_Rigid_Body("Bar", ...
2 'G_Point', "G", ...
3 'Base', "Bar", ...
4 'Mass', m, ...
5 'Intertial_Tensor', I_Bar);
```

Código 4.4: Creación del sólido rígido “Bar” asociado a la base “Bar”.

## Acciones externas

Las acciones externas se crean introduciendo el punto de aplicación, la base en la que se encuentra el tursor, el sólido rígido al que aplica y el tursor en si mismo.

Las acciones que actúan sobre el sistema son:

- la acción gravitatoria que se crea con el Código 4.5,

- el momento de entrada creado con el Código 4.6 y
- el rozamiento de Coulomb creado a través del Código 4.7.

```

1 sys_pendolo.Create_New_Action("Gravity",...
2                               'Point','G', ...
3                               'Base','B_0', ...
4                               'Rigid_Body','Bar', ...
5                               'Vector',sym([0;0;0;0;0;-m*g]));

```

Código 4.5: Creación de acción gravitatoria para el sólido “Bar”.

```

1 M_in = [str2sym("M_in");0;0;0;0;0];
2 sys_pendolo.Create_New_Action("M_in",...
3                               'Point','G', ...
4                               'Base','B_0', ...
5                               'Rigid_Body','Bar', ...
6                               'Vector',M_in);

```

Código 4.6: Creación de momento de entrada para el sólido “Bar”.

```

1 omega_b0_bar = sys_pendolo.Angular_Velocity("B_0","Bar","B_0");
2 C = str2sym("C");
3 L = str2sym("L");
4 R = str2sym("R");
5
6 M_Vis = [-C*2*pi*L*R^3*omega_b0_bar;0;0;0];
7
8 sys_pendolo.Create_New_Action("M_Vis",...
9                               'Point','G', ...
10                              'Base','B_0', ...
11                              'Rigid_Body','Bar', ...
12                              'Vector',M_Vis);

```

Código 4.7: Creación de rozamiento de Coulomb para el sólido “Bar”.

### 4.1.2 Cálculo de magnitudes del sistema

Antes de unifirar el sistema, es posible realizar cálculos manuales de varias magnitudes del sistema. En esta sección se presentarán algunas de las más importantes para el cálculo de las ecuaciones del movimiento.

#### Vector posición, velocidad y aceleración

Los vectores de posición, velocidad y aceleración son la parte fundamental de la cinemática y por eso mismo se han introducido funciones que facilitan el cálculo de estas magnitudes. Los códigos 4.8, 4.9 y 4.10 muestran el uso de estas funciones para el cálculo de la posición, velocidad y aceleración del punto  $G$  del sólido.

```

1 %function out = Get_Two_Points_Vector_Base(obj,Start_Point, End_Point, Base)
2 %##### CODIGO #####
3 sys_pendolo.Get_Two_Points_Vector_Base("N","G","B_0")
4 %##### SALIDA #####
5 0; 1_CM*sin(theta(t)); -1_CM*cos(theta(t))
6 %##### CODIGO #####
7 sys_pendolo.Get_Two_Points_Vector_Base("N","G","Bar")
8 %##### SALIDA #####
9 0; 0; -1_CM

```

Código 4.8: Función de vector posición para el vector  $NG$

```

1 %function out = Get_RI_Point_Velocity(obj, Point, Components_Base)
2 %##### CODIGO #####
3 sys_pendolo.Get_RI_Point_Velocity("G","B_0")
4 %##### SALIDA #####

```

```

5      0; l_CM*cos(theta(t))*diff(theta(t), t); l_CM*sin(theta(t))*diff(theta(t), t)
6      %##### CODIGO #####
7      sys_pendulo.Get_RI_Point_Velocity("G","Bar")
8      %##### SALIDA #####
9      0; l_CM*diff(theta(t), t); 0

```

Código 4.9: Función de vector velocidad para un punto  $G$

```

1      %function out = Get_RI_Point_Acceleration(obj, Point, Components_Base)
2      %##### CODIGO #####
3      sys_pendulo.Get_RI_Point_Acceleration("G","B_0")
4      %##### SALIDA #####
5      0; l_CM*cos(theta(t))*diff(theta(t), t, t) - l_CM*sin(theta(t))*diff(theta(t), t)
~2; l_CM*sin(theta(t))*diff(theta(t), t, t) + l_CM*cos(theta(t))*diff(theta(t), t
)~2
6      %##### CODIGO #####
7      sys_pendulo.Get_RI_Point_Acceleration("G","Bar")
8      %##### SALIDA #####
9      0; l_CM*diff(theta(t), t, t); l_CM*diff(theta(t), t)^2

```

Código 4.10: Función de vector aceleración para un punto  $G$

## Matriz de cambio de base

Las matrices de cambio de base, además de usarse como método de cálculo de la velocidad angular, pueden servir para confirmar que no existen errores en el sistema introducido. El Código 4.11 muestra las matrices de cambio desde la base “B\_0” a la base “Bar” y viceversa.

```

1      %function R_Base_1_Base_2 = Change_Basis_Matrix(obj,Base_1, Base_2)
2      %##### CODIGO #####
3      sys_pendulo.Change_Basis_Matrix("B_0","Bar")
4      %##### SALIDA #####
5      [1,      0,      0]
6      [0,  cos(theta(t)), sin(theta(t))]
7      [0, -sin(theta(t)), cos(theta(t))]
8      %##### CODIGO #####
9      sys_pendulo.Change_Basis_Matrix("Bar","B_0")
10     %##### SALIDA #####
11     [1,      0,      0]
12     [0,  cos(theta(t)), -sin(theta(t))]
13     [0,  sin(theta(t)),  cos(theta(t))]

```

Código 4.11: Función para el cálculo de matrices de cambio de base.

## Vector de velocidad angular

Como ya se ha hecho mención en la sección teórica, el vector de velocidad angular es importante para la cinemática de los sólidos rígidos. El Código 4.12 muestra un ejemplo de cálculo del vector de velocidad angular  $\Omega_{B_0}^{Bar}$  a través de la función “Angular\_Velocity”.

```

1      %function out = Angular_Velocity(obj, Base_1, Base_2, Base_Components)
2      %##### CODIGO #####
3      sys_pendulo.Angular_Velocity("B_0","Bar","B_0")
4      %##### SALIDA #####
5      diff(theta(t), t); 0; 0

```

Código 4.12: Función para el cálculo de la velocidad angular de una base respecto a otra.

### 4.1.3 Unificación del sistema

Las funciones individuales tienen la capacidad de dar toda la información del modelo, incluso es posible obtener las ecuaciones del movimiento a través de un buen uso de ellas. Todos estos cálculos vienen programados en la función “unify\_symbolic\_system”, la cual se encarga de, una vez el usuario ha introducido todos los parámetros e inicializado los objetos, realizar el cálculo de las velocidades

parciales, generalizar las acciones y ensamblar las componentes de las ecuaciones del movimiento. Las Figuras 4.2 y 4.3 muestran el antes y el después de utilizar la función en el sistema del péndulo simple. Se observa que se ha ensamblado la matriz  $A$  y el vector  $b$  que definen las ecuaciones del sistema.

```
1 %function unify_symbolic_system(obj)
2 %##### CODIGO #####
3 sys_pendulo.unify_symbolic_system
```

Código 4.13: Función que unifica el sistema simbólicamente.

System_Name	"Pendulo_Simple"
q_variables	[]
Virtual_Velocity_Components_fun_handles	[]
A_Matrix_Fun_Handles	[]
u	[]
v	[]
Transformation_Matrix_Handles	[]
A	[]
A_uv	[]
d_A	[]
d_A_uv	[]
q	[]
uv	[]
vu	[]
b	[]
b_uv	[]
System_Bases	3x2 cell
System_Points	3x2 cell
System_Coordinate_Systems	3x3 cell
System_Rigid_Bodies	2x4 cell
System_Actions	4x3 cell
System_Generalized_Coordinates	2x2 cell
System_Inputs	1x4 cell

Figura 4.2: Workspace de péndulo simple antes de “unify” simbólico.

System_Name	"Pendulo_Simple"
q_variables	[]
Virtual_Velocity_Components_fun_handles	[]
A_Matrix_Fun_Handles	[]
u	[]
v	[]
Transformation_Matrix_Handles	[]
A	1x1 sym
A_uv	2x2 sym
d_A	1x1 sym
d_A_uv	[]
q	1x1 sym
uv	2x1 sym
vu	2x1 sym
b	1x1 sym
b_uv	2x1 sym
System_Bases	3x2 cell
System_Points	3x2 cell
System_Coordinate_Systems	3x3 cell
System_Rigid_Bodies	2x4 cell
System_Actions	4x3 cell
System_Generalized_Coordinates	2x2 cell
System_Inputs	1x4 cell

Figura 4.3: Workspace de péndulo simple después de “unify” simbólico.

El Código 4.13 muestra la obtención de la matriz  $[A]$  y el vector  $[RHS]$  para el péndulo simple. También, es posible obtener la matriz vector  $b$  con la reducción de orden del sistema con las variables  $u$  y  $v$  sustituidas directamente.

```
1 %##### CODIGO #####
2 sys_pendulo.A
3 %##### SALIDA #####
4 (4*l_CM^2*m)/3
5 %##### CODIGO #####
6 sys_pendulo.b
7 %##### SALIDA #####
8 - 2*C*L*pi*diff(theta(t), t)*R^3 + M_in - g*l_CM*m*sin(theta(t))
```

Código 4.14: Matriz  $A$  y vector  $b$  en péndulo simple.

## 4.2 Ejemplo: Torre grúa

La primera aproximación al modelo de la torre grúa viene dado por [6] y es el mostrado en la Figura 4.4. En este modelo se proponen 5 coordenadas generalizadas  $\mathbf{q} = [\theta(t), r(t), \beta_2(t), \beta_1(t), \gamma(t)]^T$ . Debido a que los autores del artículo utilizan una nomenclatura que difiere de la presentada en esta memoria se presentará aquí la forma de resolución matricial explicada en la Sección 2.

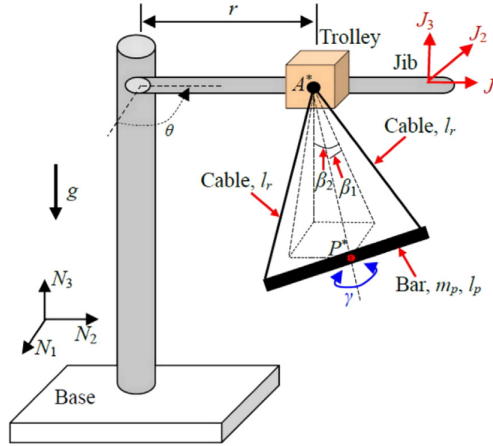


Figura 4.4: Modelo torre grúa paper. [6]

Los sólidos rígidos de los que se compone la torre grúa son:

- la base (*base*),
- la pluma (*Jib*),
- el carro (*trolley*) y
- la barra (*bar*).

Como se hará mención más adelante, estos sólidos rígidos, a excepción de la base, tienen propiedades que son necesarias para el estudio de las ecuaciones del movimiento, como por ejemplo la masa. Al estudiar esta torre grúa como un sistema de sólidos rígidos la base se mantiene anclada al suelo, es decir a la referencia inercial. Es debido a esto mismo que en el estudio de las ecuaciones del movimiento este sólido se puede considerar directamente como la referencia inercial y no es necesario caracterizar sus parámetros.

### 4.2.1 Cinemática: Creación del sistema cinemático

#### Inicialización de objetos “System”

En el Código 4.15 se muestran los comandos para inicializar el sistema mecánico. Esto creará el objeto “Crane\_System\_Paper” sobre el que se va a comenzar a trabajar. La librería ya contiene bases y puntos canónicos por lo que a partir de ellos se va a crear el sistema de coordenadas asociado a la referencia inercial. Este sistema de coordenadas “C\_0” se compone del Punto “N” y la base “B\_0”.

```

1 Crane_System_Paper = Dynamic_Library.System("Crane_Paper");
2 Crane_System_Paper.Create_New_Point("N", "Canonical", sym([0;0;0]));
3 Crane_System_Paper.Create_New_Base("B_0", 'axis_labels', [str2sym("N_1"); str2sym("N_2"); str2sym("N_3")]);
4 Crane_System_Paper.Create_New_Coordinate_System("C_0", "N", "B_0");

```

Código 4.15: Inicialización de sistema mecánico en Matlab.



## Creación de objetos “Base”

Cada uno de los sólidos que interviene en el movimiento tiene una base asociada al mismo de forma que:

- Base(R.I) :  $[\mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3]^T$
- Jib(Pluma)/Trolley(Carro) :  $[\mathbf{J}_1, \mathbf{J}_2, \mathbf{J}_3]^T$
- Bar(Barra) :  $[\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3]^T$

La matriz que define la rotación entre la base y la pluma es

$$\begin{Bmatrix} \mathbf{N}_1 \\ \mathbf{N}_2 \\ \mathbf{N}_3 \end{Bmatrix} = \begin{bmatrix} c_\theta & s_\theta & 0 \\ -s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \mathbf{J}_1 \\ \mathbf{J}_2 \\ \mathbf{J}_3 \end{Bmatrix}. \quad (4.1)$$

La matriz que define la rotación entre la pluma y la barra se calcula como

$$\begin{Bmatrix} \mathbf{J}_1 \\ \mathbf{J}_2 \\ \mathbf{J}_3 \end{Bmatrix} = \begin{bmatrix} c_{\beta_2} & 0 & -s_{\beta_2} \\ 0 & 1 & c_\theta \\ s_{\beta_2} & 0 & c_{\beta_2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_{\beta_1} & s_{\beta_1} \\ 0 & -s_{\beta_1} & c_{\beta_1} \end{bmatrix} \begin{bmatrix} c_\gamma & s_\gamma & 0 \\ -s_\gamma & c_\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{Bmatrix}, \quad (4.2)$$

y es

$$\begin{Bmatrix} \mathbf{J}_1 \\ \mathbf{J}_2 \\ \mathbf{J}_3 \end{Bmatrix} = \begin{bmatrix} c_\gamma c_{\beta_2} + s_\gamma s_{\beta_1} s_{\beta_2} & c_\gamma s_{\beta_1} s_{\beta_2} - s_\gamma c_{\beta_2} & c_{\beta_1} s_{\beta_2} \\ s_\gamma c_{\beta_1} & c_\gamma c_{\beta_1} & -s_{\beta_1} \\ s_\gamma c_{\beta_2} s_{\beta_1} - c_\gamma s_{\beta_2} & s_\gamma s_{\beta_2} + c_\gamma c_{\beta_2} s_{\beta_1} & c_{\beta_1} c_{\beta_2} \end{bmatrix} \begin{Bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{Bmatrix}. \quad (4.3)$$

En el Código 4.16 se muestra la programación para introducir las bases mencionadas. Debido a que la librería únicamente tiene implementada rotaciones alrededor de un eje se introducen las bases auxiliares “Bar\_beta\_2” y “Bar\_beta\_1”.

```

1 Crane_System_Paper.Create_New_Base("Jib", ...
2     'axis_labels',[str2sym("J_1");str2sym("J_2");str2sym("J_3
3     ")]), ...
4     'father_base',"B_0", ...
5     'angle',-str2sym("theta(t)"), ...
6     'axis_rotation',"3");
7 Crane_System_Paper.Create_New_Base("Bar_beta_2", ...
8     'axis_labels',[str2sym("P_1_2");str2sym("P_2_2");str2sym("
9     "P_3_2")]), ...
10    'father_base',"Jib", ...
11    'angle',str2sym("beta_2(t)"), ...
12    'axis_rotation',"2");
13 Crane_System_Paper.Create_New_Base("Bar_beta_1", ...
14    'axis_labels',[str2sym("P_1_1");str2sym("P_2_1");str2sym("
15    "P_3_1")]), ...
16    'father_base',"Bar_beta_2", ...
17    'angle',str2sym("beta_1(t)"), ...
18    'axis_rotation',"1");
19 Crane_System_Paper.Create_New_Base("Bar", ...
20    'axis_labels',[str2sym("P_1");str2sym("P_2");str2sym("P_3
21    ")]), ...
22    'father_base',"Bar_beta_1", ...
23    'angle',str2sym("Gamma(t)"), ...
24    'axis_rotation',"3");

```

Código 4.16: Creación bases del modelo de torre grúa.

## Inserción de “Coordenadas Generalizadas”

Conforme se han ido añadiendo las bases hay que tener en cuenta que se han definido nuevas coordenadas generalizadas en el sistema y por lo tanto, hay que añadirlas. Por el momento se deben añadir  $\theta, \beta_1, \beta_2, \gamma$  y esto se muestra en el Código 4.17.

```

1 Crane_System_Paper.Create_New_Generalized_Coordinate("theta",str2sym("theta(t)"))
;
2 Crane_System_Paper.Create_New_Generalized_Coordinate("beta_2",str2sym("beta_2(t)"))
;
3 Crane_System_Paper.Create_New_Generalized_Coordinate("beta_1",str2sym("beta_1(t)"))
;
4 Crane_System_Paper.Create_New_Generalized_Coordinate("Gamma",str2sym("Gamma(t)"))
;

```

Código 4.17: Introducción de coordenadas generalizadas  $\theta, \beta_1, \beta_2, \gamma$ .

## Creación de “Sistemas de Coordenadas” y objetos “Point”

Los tres sólidos rígidos vienen caracterizados por la posición de su centro de gravedad mostrados en la Figura 4.5.

Sólido	Punto	$\mathbf{r}$
Pluma	$G_{Jib}$	$\mathbf{G}_{Jib} = \begin{Bmatrix} L_{G_{Jib}} \\ 0 \\ 0 \end{Bmatrix}_J$
Carro	$A^*$	$\mathbf{OA}^* = \begin{Bmatrix} r(t) \\ 0 \\ 0 \end{Bmatrix}_J$
Barra	$P^*$	$\mathbf{OP}^* = \mathbf{OA}^* + \begin{Bmatrix} 0 \\ 0 \\ -l_y \end{Bmatrix}_P$

Figura 4.5: Centros de gravedad de los sólidos del sistema.

Para introducir estos puntos dentro del sistema se requiere la creación de sistemas de coordenadas. Debido a que ya se han introducido las bases asociadas a los sólidos rígidos el Código 4.18 muestra la creación de los sistemas de coordenadas que se requieren para crear los puntos necesarios. De igual forma que en las rotaciones, se debe introducir una coordenada generalizada más al sistema debido al desplazamiento del carro  $r(t)$ . El código para añadir la nueva coordenada generalizada viene dado en 4.19.

```

1 Crane_System_Paper.Create_New_Coordinate_System("Referencial_Jib","N","Jib");
2 Crane_System_Paper.Create_New_Point("A_ast","Referencial_Jib",sym([str2sym("r(t)"))
;0;0]));
3 Crane_System_Paper.Create_New_Coordinate_System("Referencial_Trolley","A_ast","
Jib");
4 Crane_System_Paper.Create_New_Coordinate_System("Referencial_Trolley_Bar","A_ast"
,"Bar");
5 Crane_System_Paper.Create_New_Point("P_ast","Referencial_Trolley_Bar",sym([0;0;-
str2sym("ly")]));

```

Código 4.18: Introducción de sistemas de coordenadas en el sistema.

```

1 Crane_System_Paper.Create_New_Generalized_Coordinate("r",str2sym("r(t)"));

```

Código 4.19: Introducción de coordenada generalizada  $r(t)$ .

### 4.2.2 Cinemática: Componentes parciales de velocidad

Las funciones descritas en el cálculo de las ecuaciones del péndulo permiten el cálculo de las componentes parciales de la velocidad. Se presentan todas las componentes parciales para cada uno de los sólidos rígidos del sistema en los Cuadros 4.1, 4.2 y 4.3.

La librería calcula las velocidades parciales en la base “Canonical” la cual por definición de la librería es la referencia inercial.

<i>Jib</i>		
$q_k$	$\mathbf{b}^{Jib}$	$\mathbf{c}^{Jib}$
$\theta(t)$	$\begin{Bmatrix} -L_{Jib_G} s_\theta \\ -L_{Jib_G} c_\theta \\ 0 \end{Bmatrix}_N$	$\begin{Bmatrix} 0 \\ 0 \\ -1 \end{Bmatrix}_N$
$r(t)$	$\mathbf{0}$	$\mathbf{0}$
$\beta_2(t)$	$\mathbf{0}$	$\mathbf{0}$
$\beta_1(t)$	$\mathbf{0}$	$\mathbf{0}$
$\gamma(t)$	$\mathbf{0}$	$\mathbf{0}$

Cuadro 4.1: Velocidades parciales del sólido rígido “Jib”

<i>Trolley</i>		
$q_k$	$\mathbf{b}^{Jib}$	$\mathbf{c}^{Jib}$
$\theta(t)$	$\begin{Bmatrix} -rs_\theta \\ -rc_\theta \\ 0 \end{Bmatrix}_N$	$\begin{Bmatrix} 0 \\ 0 \\ -1 \end{Bmatrix}_N$
$r(t)$	$\begin{Bmatrix} c_\theta \\ -s_\theta \\ 0 \end{Bmatrix}_N$	$\mathbf{0}$
$\beta_2(t)$	$\mathbf{0}$	$\mathbf{0}$
$\beta_1(t)$	$\mathbf{0}$	$\mathbf{0}$
$\gamma(t)$	$\mathbf{0}$	$\mathbf{0}$

Cuadro 4.2: Velocidades parciales del sólido rígido “Trolley”

<i>Bar</i>		
$q_k$	$\mathbf{b}^{Jib}$	$\mathbf{c}^{Jib}$
$\theta(t)$	$\begin{Bmatrix} -l_y s_{\beta_1} c_\theta + c_{\beta_1} s_{\beta_2} s_\theta - r s_\theta \\ l_y s_{\beta_1} s_\theta - c_{\beta_1} s_{\beta_2} c_\theta - r c_\theta \\ 0 \end{Bmatrix}_N$	$\begin{Bmatrix} 0 \\ 0 \\ -1 \end{Bmatrix}_N$
$r(t)$	$\begin{Bmatrix} c_\theta \\ -s_\theta \\ 0 \end{Bmatrix}_N$	$\begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}_N$
$\beta_2(t)$	$\begin{Bmatrix} -l_y c_{\beta_1} c_{\beta_2} c_\theta \\ l_y c_{\beta_1} c_{\beta_2} s_\theta \\ -l_y c_{\beta_1} s_{\beta_2} \end{Bmatrix}_N$	$\begin{Bmatrix} -s_\theta \\ -c_\theta \\ 0 \end{Bmatrix}_N$
$\beta_1(t)$	$\begin{Bmatrix} l_y c_{\beta_1} s_\theta + l_y s_{\beta_1} s_{\beta_2} c_\theta \\ l_y c_{\beta_1} c_\theta l_y s_{\beta_1} s_{\beta_2} s_\theta \\ -l_y c_{\beta_2} s_{\beta_1} \end{Bmatrix}_N$	$\begin{Bmatrix} -c_{\beta_2} c_\theta \\ c_{\beta_2} s_\theta \\ -s_{\beta_2} \end{Bmatrix}_N$
$\gamma(t)$	$\begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}_N$	$\begin{Bmatrix} c_{\beta_1} s_{\beta_2} c_\theta - s_{\beta_1} s_\theta \\ -s_{\beta_1} c_\theta - c_{\beta_1} s_{\beta_2} s_\theta \\ -c_{\beta_1} c_{\beta_2} \end{Bmatrix}_N$

Cuadro 4.3: Velocidades parciales del sólido rígido “Bar”

### 4.2.3 Dinámica: Parametros, acciones y entradas

#### Masa y tensores de inercia

Previo al cálculo dinámico, se deben introducir las propiedades dinámicas de cada uno de los sólidos. Estas propiedades vienen recogidas en la masa y el tensor de inercia. Se debe tener especial cuidado a la hora de introducir los tensores de inercia debido a que estos se pueden calcular en diferentes

puntos y en diferentes bases cinemáticas. Debido a que la librería realiza todos los cálculos en la base “Canónica” esta será la base en la que se deberán introducir los tensores de inercia. Por otro lado, como ya se comentó a través de la Ecuación 2.46 el tensor debe estar calculado en el centro de gravedad del sólido ya que ha sido una de las simplificaciones tomadas para la programación de la librería.

Teniendo lo anterior en cuenta el Cuadro 4.4 presenta simbólicamente las masas y los tensores de inercia utilizados para los diferentes sólidos rígidos del sistema.

Sólido	Masa	Tensor de Inercia
Jib	$m_{Jib}$	$\mathcal{I}_G(Jib) = \begin{bmatrix} I_{Jib_{11}} & I_{Jib_{12}} & I_{Jib_{13}} \\ I_{Jib_{12}} & I_{Jib_{22}} & I_{Jib_{23}} \\ I_{Jib_{13}} & I_{Jib_{23}} & I_{Jib_{33}} \end{bmatrix}_N$
Trolley	$m_{Trolley}$	$\mathcal{I}_G(Trolley) = \begin{bmatrix} I_{Trolley_{11}} & I_{Trolley_{12}} & I_{Trolley_{13}} \\ I_{Trolley_{12}} & I_{Trolley_{22}} & I_{Trolley_{23}} \\ I_{Trolley_{13}} & I_{Trolley_{23}} & I_{Trolley_{33}} \end{bmatrix}_N$
Bar	$m_{Bar}$	$\mathcal{I}_G(Bar) = \begin{bmatrix} I_{Bar_{11}} & I_{Bar_{12}} & I_{Bar_{13}} \\ I_{Bar_{12}} & I_{Bar_{22}} & I_{Bar_{23}} \\ I_{Bar_{13}} & I_{Bar_{23}} & I_{Bar_{33}} \end{bmatrix}_N$

Cuadro 4.4: Tensores de inercia sólidos rígidos

El Código 4.20 muestra un script para formar el tensor de inercia simbólico del sólido “Trolley”.

```

1  Mass_Trolley = str2sym("m_Trolley");
2  I_Trolley = sym([]);
3  for i = 1:3
4      for j = 1:3
5          if i <= j
6              str = "I_Trolley_"+string(i)+string(j);
7              I_Trolley(i,j)=str2sym(str);
8              Crane_System_Paper.Create_New_Parameter(str,str2sym(str));
9
10             else
11                 str = "I_Trolley_"+string(j)+string(i);
12                 I_Trolley(i,j)=str2sym(str);
13             end
14         end
15     end
16
17     Mass_Trolley =
18     m_Trolley
19
20     I_Trolley =
21     [I_Trolley_11, I_Trolley_12, I_Trolley_13]
22     [I_Trolley_12, I_Trolley_22, I_Trolley_23]
23     [I_Trolley_13, I_Trolley_23, I_Trolley_33]

```

Código 4.20: Masa y tensor de inercia simbólico de “Trolley”.

Con estas variables ya creadas, es posible generar los objetos que representan a los sólidos rígidos como muestra el Código 4.21.

```

1  Crane_System_Paper.Create_New_Rigid_Body("Trolley", ...
2      'G_Point','A_ast',...
3      'Base','Jib',...
4      'Mass',Mass_Trolley, ...
5      'Intertial_Tensor',I_Trolley);

```

Código 4.21: Creación del “Rigid\_Body” “Trolley” en el sistema.

La función “Create\_New\_Rigid\_Body” crea un objeto “Rigid\_Body” y lo añade al sistema. Es importante escoger el centro de gravedad correctamente debido a que en caso de incluir acciones que no actúen en el centro de gravedad estas generarán momentos.

### Acción gravitatoria

La acción gravitatoria para cada sólido se introduce de la misma forma, el Código 4.22 muestra la manera de implementar la acción gravitatoria al sólido “Jib”. Esta acción viene caracterizada por el tórsor  $[0, 0, 0, 0, 0, -gMass_{Jib}]^T$  y se aplica en el centro de gravedad  $G_{Jib}$ .

```
1 Crane_System_Paper.Create_New_Action("Gravity_Jib",...
2                                     'Point',"G_Jib", ...
3                                     'Base',"B_0", ...
4                                     'Rigid_Body',"Jib", ...
5                                     'Vector',sym([0;0;0;0;0;-Mass_Jib*g]));
```

Código 4.22: Código que añade acción gravitatoria al sólido “Jib”.

### Acción de momento de entrada $M_{in}$

El momento de entrada en el “Jib” se introduce según el Código 4.23.

```
1 Crane_System_Paper.Create_New_Action("Moment_In_Jib",...
2                                     'Point',"N", ...
3                                     'Base',"B_0", ...
4                                     'Rigid_Body',"Jib", ...
5                                     'Vector',sym([str2sym("tau_in");0;0;0;0;0]));
```

Código 4.23: Código que añade acción  $M_{in}$ .

## 4.2.4 Unión del sistema

Habiendo introducido todas las propiedades y variables del sistema, es hora de realizar la unión. El proceso de ensamblaje consiste en calcular cada una de las matrices  $[A^i]$  para cada sólido y sumarlos en una matriz  $[A]$  del sistema. También, generaliza todas las acciones a las coordenadas generalizadas introducidas por el usuario. El Código 4.24 muestra la función “unify\_symbolic\_system” que realiza la unión.

```
1 tic
2 Crane_System_Paper.unify_symbolic_system
3 toc
4
5 Elapsed time is 25.620687 seconds.
```

Código 4.24: Código que añade acción  $M_{in}$ .

En este código en concreto, se ha medido el tiempo necesario para el cálculo de la matriz  $[A]$  y el vector  $RHS$  que ya se presentó en el cálculo de las ecuaciones del movimiento. Para este sistema con 5 grados de libertad se han necesitado 110 segundos, lo cual para un modelo complejo puede ser un tiempo manejable.

Sumado al tiempo de unión del sistema, si se desea convertir la ecuación diferencial algebraica (DAE) en ODE, es necesario realizar la inversa de la matriz  $[A]$  como se veía en la Ecuación 2.73. El Código 4.25 muestra los comandos para realizar esto.

```
1 tic
2 A_inv = inv(Crane_System_Paper.A);
3 toc
4
5 Elapsed time is 3241.897960 seconds.
```

Código 4.25: Código para invertir la matriz  $[A]$ .

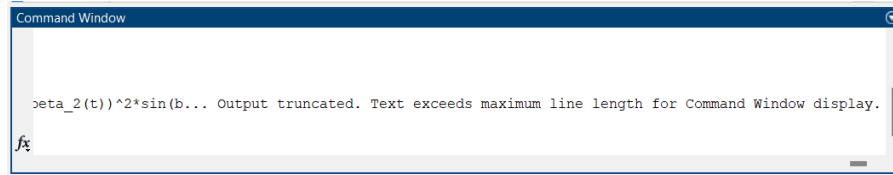


Figura 4.6: Ventana de comandos de Matlab tras intentar visualizar la componente (1,1) de la matriz [4].

El tiempo que se tarda en invertir esta matriz es de 3.241 segundos (aproximadamente 54 minutos). En el supuesto caso de que el modelo sea el definitivo y se necesite pasar al diseño del controlador, este tiempo es asequible, pero el problema va mas allá. Las expresiones resultantes de la inversa no son manejables, tal es el punto, que la propia ventana de comandos de Matlab no permite visualizar la expresión completa de la matriz. La Figura 4.6 muestra el intento de visualizar una de las componentes de la matriz a través de la ventana de comandos, en la que el propio Matlab trunca la expresión.

Aún siendo la matriz no singular (por definición), se encuentran dificultades en encontrar la inversa simbólica debido a que:

- conforme aumentan los grados de libertad del sistema el tamaño de esta matriz aumenta,
- el costo computacional de las operaciones simbólicas es mucho mayor que con datos numéricos y,
- las expresiones que se pueden obtener no son manejables.

Dentro de la librería se ha dado el primer paso para solventar esta dificultad a través de la función “unify\_system” la cual únicamente unifica sistemas numéricos. Esta función genera funciones de cada uno de los parámetros necesarios para la composición del sistema, como se ha mostrado en la Sección 3.3.5, y permite calcular el sistema final con datos numéricos. Esto ayuda a que la inversa de la matriz (en este caso) sea numérica y no simbólica. Además de la ventaja de convertir la ecuación diferencial algebraica (DAE) en ODE, permite acercar el modelado a la realidad con la introducción de valores numéricos, de esta forma la resolución numérica o simulación es el siguiente paso natural de estudio.

## 5 Conclusiones

Teniendo en mente los objetivos que se han presentado al principio de esta memoria se puede concluir que se han cumplido una gran parte de los objetivos propuestos.

Se ha encontrado una forma sistemática de modelar sistemas mecánicos holónomos. El objetivo principal se basaba en poder modelar también sistemas que incorporan ligaduras no holónomas, pero debido a que el estudio de estos es más complicado, como se ha hecho mención en la Sección 2.1.5, se ha reducido el alcance del proyecto a sistemas holónomos. Estos sistemas incluyen a la mayor parte de torres grúas de construcción, por lo tanto, el estudio de sistemas holónomos da solución al contexto presentado en el resumen inicial.

Las ecuaciones del movimiento se pueden presentar de una forma clara y sencilla debido a que el sistema puede descomponerse en diferentes subsistemas. Este concepto se ha presentado en la Sección 2.3.6, a través de la Ecuación 2.70, la cual muestra que la aportación que cada sólido hace al sistema se representa mediante una matriz  $[A^i]$  propia. Esta matriz se construye a partir de las componentes parciales de la velocidad  $b_i$  y  $c_i$  y, a partir de la masa  $m$  y el tensor de inercia  $\mathcal{I}$  de cada sólido rígido. En concreto, se ha estudiado el modelo de una torre grúa que se descompone en cada uno de los sólidos rígidos mencionados en la Sección 4.2. Para este caso los vectores  $b_i$  y  $c_i$  se han presentado en los Cuadros 4.1, 4.2 y 4.3.

Estas mismas componentes parciales de las velocidades son utilizadas para introducir fuerzas externas al sistema y, en concreto, a cada uno de los sólidos. En las Secciones 2.4.1, 2.4.2 y 2.4.3 se han presentado ejemplos de tres tipos de fuerzas comunes en sistemas mecánicos y la forma en la que se introducen en la metodología. Estas fuerzas, denominadas “generalizadas”, facilitan tener en cuenta fenómenos físicos de una forma más sencilla y el estudio de fuerzas y momentos de entrada, los cuales son importantes en la teoría de control.

Todo el desarrollo de la metodología se puede llevar a cabo con la teoría del álgebra lineal, lo cual hace el método más implementable a nivel computacional, como se ha expuesto en la Sección 2.3.6. Esta facilidad para la implementación computacional se ha puesto de manifiesto en el diseño de la librería dinámica “Dynamic\_Library” (Sección 3) que ayuda en el cálculo de las ecuaciones del movimiento. Dado que el uso de una librería sin documentación es complejo para un usuario, las Secciones 4.1 y 4.2 se han presentado como guía para calcular las ecuaciones del movimiento de dos sistemas mecánicos.

Como ya se ha mencionado en la Sección 4.2.4 uno de los mayores inconvenientes que intervienen en este método es la necesidad de invertir la matriz  $[A]$  del sistema para obtener una ecuación diferencial ordinaria (ODE). Esto hace que se dificulte en gran medida el objetivo de la linealización simbólica de este tipo de sistemas, que si bien no es imposible, resultan en una implementación poco práctica. Por lo tanto, para conseguir resultados que se puedan utilizar en contextos reales, se deben estudiar métodos de linealización numéricos o métodos de control de sistemas no lineales.

Como resumen, se ha conseguido presentar una metodología sistemática para el cálculo de las ecuaciones del movimiento de sistemas holónomos, los cuales representan una gran parte de los modelos de torre grúa. Se ha mostrado que la implementación para sistemas no holónomos tiene que ser particular para cada caso, haciendo que no exista un método general que se pueda aplicar. Para facilitar el cálculo de las ecuaciones del movimiento se ha diseñado la librería “Dynamic\_Library” en Matlab, la cual, ha sido programado a través de matrices y vectores (álgebra lineal) que hacen más cercano el diseño de controladores. Finalmente, se ha visto que conseguir ODEs simbólicas para modelos complejos no re-

sulta práctico debido al coste computacional que esto supone. Esto hace que se deban utilizar métodos de linealización numéricos o estudiar el sistema con métodos de control no lineal.



## 6 Lineas futuras

Si bien el proyecto presenta una librería totalmente operativa y una metodología clara existen estudios futuros a tener en cuenta para mejorar tanto los tiempos de computación como los desarrollos.

### Algoritmos recursivos para el cálculo de las ecuaciones del movimiento

La metodología se ha presentado de forma que todos los calculos se realizan desde una base canónica asociada a la referencia inercial. Existen casos en los que este convenio no es el más adecuado. En caso de que los sólidos rígidos esten acoplados en forma de “arbol” una buena práctica es referenciar las velocidades de un sólido con el siguiente. Existen metodologías que implementan esto añadiendo una identificación clara para cada uno de los sólidos como indica Roy Featherstone [18]. Este autor desarrolla una teoría cinemática y dinámica a través de lo que se define como vectores de espacio, y el movimiento de los sólidos los referencia unos respecto de otros. Esto permite que los algoritmos de resolución sean recursivos de  $\mathcal{O}(n)$  disminuyendo considerablemente el coste computacional. El autor también tiene en cuenta ligaduras no holónomas a la hora de realizar el estudio del sistema, por lo que el alcance de los modelos aumenta cumpliendo el objetivo que se tenía en un principio en el desarrollo de este proyecto. El método de Kane original [10] también tiene en cuenta estas ideas aunque no presenta diferencias entre cadena abierta o cerrada y no presenta explícitamente los algoritmos a implementar.

### Linealización numérica o estudio de métodos de control no lineales

El uso del álgebra lineal ayuda a estrechar la brecha entre el modelo y el control, pero sigue sin haberlo hecho del todo. El modelo presentado por la Ecuación 2.73 es una DAE, que si bien puede convertirse en ODE, se ha visto que en la práctica el tiempo de cómputo puede ser elevado al trabajar con objetos simbólicos. Esto hace que exista cierta complicación en estudiar este tipo de sistemas con métodos de control lineales tradicionales ya que se debe linealizar el sistema simbólicamente, y por lo tanto, primero hay que invertir la matriz  $[\mathbf{A}^i]$ . En última instancia, otra linea futura propone el uso de métodos numéricos de linealización o la introducción al control de sistemas no lineales (*feedback linearization*).

### Simulación de sistemas

La simulación de modelos es una parte fundamental para el diseño de controladores, por lo tanto, es esencial implementar métodos numéricos que resuelvan ecuaciones diferenciales para obtener la respuesta de los sistemas en el tiempo. La función “unify\_system” (Sección 3.3.5) da un primer paso hacia esta resolución numérica ya que genera las funciones necesarias que calculan las matrices y vectores para caracterizar un sistema.



# Bibliografía

- [1] H. Goldstein, “Mecánica clásica,” 2006.
- [2] J. M. A. Aguinagalde, “Apuntes de la asignatura Mecánica,” 2019.
- [3] J. A. B. Agulló, “Mecanica de la particula y del solido rigido.” <https://www.agullobatlle.cat/activitat-docent>, 2000.
- [4] G. González, “Single and Double plane pendulum.” <https://www.phys.lsu.edu/faculty/gonzalez/Teaching/Phys7221/DoublePendulum.pdf>, 2006.
- [5] U. C. I. de Madrid, “Rigid Body: Physical Pendulum.” [https://laboratoriofisica.uc3m.es/guiones\\_ing/mr/Physical-Pendulum.pdf](https://laboratoriofisica.uc3m.es/guiones_ing/mr/Physical-Pendulum.pdf).
- [6] J. H. Jiahui Ye, “Analytical analysis and oscillation control of payload twisting dynamics in a tower crane carrying a slender payload.” <https://www.sciencedirect.com/science/article/pii/S0888327021001588>, 2021.
- [7] Wikipedia, “Skew-symmetric matrix.” [https://en.wikipedia.org/wiki/Skew-symmetric\\_matrix](https://en.wikipedia.org/wiki/Skew-symmetric_matrix).
- [8] T. D. Barfoot, “State estimation for robotics.” [http://asrl.utias.utoronto.ca/~tdb/bib/barfoot\\_ser24.pdf](http://asrl.utias.utoronto.ca/~tdb/bib/barfoot_ser24.pdf), 2017.
- [9] E. T. Stoneking, “Newton-Euler Dynamic Equations of Motion for a Multi-body Spacecraft.” <https://ntrs.nasa.gov/api/citations/20080044854/downloads/20080044854.pdf>.
- [10] T. R. Kane, “Spacecraft Dynamics,” 2005.
- [11] E. T. Stoneking, “Implementation of Kane’s Method for a Spacecraft Composed of Multiple Rigid Bodies.” <https://ntrs.nasa.gov/api/citations/20160000805/downloads/20160000805.pdf>.
- [12] Wikipedia, “Einstein notation .” [https://en.wikipedia.org/wiki/Einstein\\_notation](https://en.wikipedia.org/wiki/Einstein_notation).
- [13] T. MathWorks, “Create Namespaces.” [https://es.mathworks.com/help/matlab/matlab\\_oop/namespaces.html](https://es.mathworks.com/help/matlab/matlab_oop/namespaces.html), 2024.
- [14] V. Coselev, “Dynamic Library.” [https://github.com/vcoselev/Dynamic\\_Library](https://github.com/vcoselev/Dynamic_Library), 2024.
- [15] T. MathWorks, “subs: Symbolic Substitution.” <https://es.mathworks.com/help/symbolic/subs.html>, 2024.
- [16] T. MathWorks, “matlabFunction: Convert symbolic expression to function handle or file.” <https://es.mathworks.com/help/symbolic/sym.matlabfunction.html>, 2024.
- [17] T. MathWorks, “odeFunction: Convert symbolic expressions to function handle for ODE solvers.” <https://es.mathworks.com/help/symbolic/sym.odefunction.html>, 2024.
- [18] R. Featherstone, “Rigid Body Dynamics Algorithms.” <https://link.springer.com/book/10.1007/978-1-4899-7560-7>, 2008.