



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



ESCUELA TÉCNICA
SUPERIOR INGENIEROS
INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO DE UNA HERRAMIENTA PARA EL CÁLCULO DE TRAYECTORIAS LIBRES DE COLISIÓN BASADAS EN CURVAS DE BÉZIER PARA OBJETOS VOLADORES

AUTOR: CARLOS GIMENO RUIZ

TUTOR: ENRIQUE JORGE BERNABEU SOLER

Curso Académico: 2017-18

Agradecimientos

Quiero aprovechar la ocasión para agradecer el apoyo de mis padres, Alfredo y María del Carmen, y de mi hermana Jessica. Sin ellos jamás habría conseguido llegar hasta aquí y siempre me guiaron en la dirección correcta.

Tampoco puedo olvidarme de mis abuelos, que siempre han estado ahí cuando les he necesitado, así como de mis amigos y compañeros de clase, que me hicieron las cosas mucho más sencillas cuando lo veía todo negro.

Por último, quiero dar las gracias a mi tutor, Enrique Jorge Bernabeu, que supo darme las claves necesarias para avanzar paso a paso y con constancia durante estos meses.

Resumen

El presente Trabajo Final de Grado consiste en el diseño de una herramienta para el cálculo de trayectorias libres de colisión, la cual se aplicará a un objeto volador que detecte un posible choque con otro en su futuro inmediato, realizando así diversas simulaciones que verifiquen que dicha herramienta funciona de manera óptima.

Los dos objetos de interés siguen trayectorias tridimensionales en el aire basadas en un método conocido como curvas de Bézier. No obstante, éstos pueden sufrir una colisión entre sí durante su tiempo de vuelo, por lo que debe conseguir calcularse, sólo para uno de los objetos, una nueva curva lo más parecida posible a la original y con el mínimo desvío en el instante de choque tal que sirva para evitarlo.

También deben evitarse grandes diferencias de velocidad y aceleración entre el objeto con su trayectoria original y el mismo objeto con la nueva. Por tanto, se deben tener en cuenta una serie de condiciones durante la implementación, realizada en el programa MATLAB.

Palabras clave: diseño, cálculo de trayectorias, objeto volador, curvas de Bézier, MATLAB

Resum

El present Treball Final de Grau consistix en el disseny d'una ferramenta per al càlcul de trajectòries lliures de col·lisió, la qual s'aplicarà a un objecte volador que detecte un possible xoc amb un altre en el seu futur immediat, realitzant així diverses simulacions que verifiquen que la nomenada ferramenta funciona de manera òptima.

Els dos objectes d'interés segueixen trajectòries tridimensionals en l'aire basades en un mètode conegut com a corbes de Bézier. No obstant això, estos poden patir una col·lisió entre si durant el seu temps de vol, per la qual cosa ha d'aconseguir calcular-se, només per a un dels objectes, una nova corba el més pareguda possible a l'original i amb el mínim desviació en l'instant de xoc tal que servisca per a evitar-ho.

També han d'evitar-se grans diferències de velocitat i acceleració entre l'objecte amb la seua trajectòria original i el mateix objecte amb la nova. Per tant, s'han de tindre en compte una sèrie de condicions durant la implementació, realitzada en el programa MATLAB.

Paraules clau: disseny, càlcul de trajectòries, objecte volador, corbes de Bézier, MATLAB

Abstract

The aim of this Project is to design a tool for the calculation of collision-free trajectories, which will be applied to a flying object that detects a possible collision with another in its immediate future, running simulations to verify that the tool is working correctly.

The movement of the two objects of interest is based on 3-D aerial trajectories that consist of a method commonly known as Bézier curves. Nevertheless, that objects can fall into a collision between them during their flights, so must be created, only for one of that objects, a new curve which is very similar to the original and shows a minimum diversion at the moment of crash, so that it can be avoided.

Also must be avoided great differences of speed and acceleration between the object with its original trajectory and the same object with the new trajectory. Therefore, the tool has to consider several conditions during the implementation, and she is created with the software MATLAB.

Keywords: design, calculation of trajectories, flying objects, Bézier curves, MATLAB

Índice general

I	Memoria	1
1	Introducción	3
1.1	Antecedentes.....	3
1.1.1	Descripción del programa Matlab.....	3
1.1.2	Origen de las curvas de Bézier.....	5
1.2	Objetivos.....	6
1.3	Motivación y justificación.....	6
1.4	Estructura del trabajo.....	7
2	Curvas de Bézier	9
2.1	Curva de Bézier 3D de orden N.....	9
2.1.1	Expresión de una curva de Bézier de orden N: Control Points.....	9
2.1.2	Coefficient Points.....	10
2.1.3	Coefficient Points de velocidad.....	11
2.1.4	Coefficient Points de aceleración.....	11
2.2	Programación en MATLAB.....	12
2.2.1	Función “obtenerCoePointsPVA”.....	12
2.2.2	Función “obtenerCoeP”.....	13
2.2.3	Función “obtenerCoeV”.....	14
2.2.4	Función “obtenerCoeA”.....	15
2.2.5	Función “obtenerPVA”.....	16
2.2.6	Función “mostrarPVA”.....	17
3	Obtención de la distancia mínima entre curvas	19
3.1	Coefficient Points en diferencias.....	19
3.1.1	Influencia del orden de ambas curvas.....	19
3.1.2	Minimización de la ecuación de diferencias.....	20
3.2	Programación en MATLAB.....	21
3.2.1	Función “obtenerCoeDif”.....	21
3.2.2	Función “obtenerCoeDif_2”.....	24
4	Obtención de la nueva curva de Bézier	25
4.1	Condiciones necesarias en la nueva Bézier.....	25
4.2	Optimización por el método de mínimos cuadrados.....	27
4.3	Programación en MATLAB.....	32
4.3.1	Función “obtenerNuevaBezier”.....	32

4.3.2	Función “compararBezier”.....	35
4.3.3	Función “compararEsferas”.....	36
5	Ejemplos	39
5.1	Ejemplo 1: Curvas del mismo orden.....	40
5.2	Ejemplo 2: Curvas de diferente orden.....	49
6	Conclusiones	59
II	Bibliografía	61
	Bibliografía	62
III	Presupuesto	65
7	Presupuesto	66
7.1	Necesidad del presupuesto.....	66
7.2	Cuadro de precios básicos.....	66
7.3	Cuadro de precios unitarios descompuestos.....	67
7.4	Mediciones y cuadro de presupuestos parciales.....	68
7.5	Resumen del presupuesto.....	69

Índice de figuras

Figura 1.1: Ventana principal de MATLAB, donde se aprecia principalmente el Command Window	4
Figura 1.2: Apariencia de una función en MATLAB	4
Figura 1.3: Pierre Étienne Bézier [6]	5
Figura 1.4: Apariencia de una curva de Bézier de $N=3$ [4]	5
Figura 1.5: Barco y colgante representados con splines [7]	6
Figura 2.1: Aspecto de Workspace al introducir parámetros	12
Figura 2.2: Función “obtenerCoePointsPVA”	13
Figura 2.3: Función “obtenerCoeP”	13
Figura 2.4: Función “obtenerCoeV”	14
Figura 2.5: Función “obtenerCoeA”	15
Figura 2.6: Tamaño de las matrices de Coefficient Points	15
Figura 2.7: Función “obtenerPVA”	16
Figura 2.8: Función “mostrarPVA”	17
Figura 3.1: Función “obtenerCoeDif”	22
Figura 3.2: Función “obtenerCoeDif_2”	24
Figura 4.1: Ajuste de un conjunto de datos con mínimos cuadrados [9]	27
Figura 4.1: Función “obtenerNuevaBezier”	33
Figura 4.2: Función “compararBezier”	35
Figura 4.3: Función “compararEsferas”	36
Figura 5.1: Script “herramientaBezier”	40
Figura 5.2: Parámetros de entrada (ejemplo 1)	40
Figura 5.3: Trayectoria de la curva de Bézier 1 (ejemplo 1)	40
Figura 5.4: Velocidad de la curva de Bézier 1 (ejemplo 1)	41
Figura 5.5: Aceleración de la curva de Bézier 1 (ejemplo 1)	41
Figura 5.6: Magnitud de la velocidad en función del tiempo de la curva de Bézier 1 (ejemplo 1)	41
Figura 5.7: Magnitud de la aceleración en función del tiempo de la curva de Bézier 1 (ejemplo 1)	42
Figura 5.8: Trayectoria de la curva de Bézier 2 (ejemplo 1)	42
Figura 5.9: Velocidad de la curva de Bézier 2 (ejemplo 1)	42
Figura 5.10: Aceleración de la curva de Bézier 2 (ejemplo 1)	43
Figura 5.11: Magnitud de la velocidad en función del tiempo de la curva de Bézier 2 (ejemplo 1)	43

Figura 5.12: Magnitud de la aceleración en función del tiempo de la curva de Bézier 2 (ejemplo 1).....	43
Figura 5.13: Comparación entre “obtenerCoeDif” y “obtenerCoeDif_2” (ejemplo 1).....	44
Figura 5.14: Comparación entre trayectorias de la antigua y la nueva Bézier 1 (ejemplo 1).....	44
Figura 5.15: Comparación entre velocidades de la antigua y la nueva Bézier 1 (ejemplo 1)	45
Figura 5.16: Comparación entre aceleraciones de la antigua y la nueva Bézier 1 (ejemplo 1) ..	45
Figura 5.17: Comparación entre magnitudes de velocidad de la antigua y la nueva Bézier 1 (ejemplo 1)	45
Figura 5.18: Comparación entre magnitudes de aceleración de la antigua y la nueva Bézier 1 (ejemplo 1)	46
Figura 5.19: Esferas 1 y 2 originales y 1 modificando su posición (ejemplo 1)	46
Figura 5.20: Zoom a la Figura 5.20 (ejemplo 1)	47
Figura 5.21: Esferas 1 y 2 originales (ejemplo 1).....	47
Figura 5.22: Zoom 1 a la Figura 5.22 (ejemplo 1).....	47
Figura 5.23: Zoom 2 a la Figura 5.22 (ejemplo 1).....	48
Figura 5.24: Esfera 1 modificando su posición y 2 original (ejemplo 1).....	48
Figura 5.25: Zoom 1 a la Figura 5.25 (ejemplo 1).....	48
Figura 5.26: Zoom 2 a la Figura 5.25 (ejemplo 1).....	49
Figura 5.27: Parámetros de entrada (ejemplo 2).....	49
Figura 5.28: Trayectoria de la curva de Bézier 1 (ejemplo 2).....	50
Figura 5.29: Velocidad de la curva de Bézier 1 (ejemplo 2).....	50
Figura 5.30: Aceleración de la curva de Bézier 1 (ejemplo 2).....	50
Figura 5.31: Magnitud de la velocidad en función del tiempo de la curva de Bézier 1 (ejemplo 2).....	51
Figura 5.32: Magnitud de la aceleración en función del tiempo de la curva de Bézier 1 (ejemplo 1).....	51
Figura 5.33: Trayectoria de la curva de Bézier 2 (ejemplo 2).....	51
Figura 5.34: Velocidad de la curva de Bézier 2 (ejemplo 2).....	52
Figura 5.35: Aceleración de la curva de Bézier 2 (ejemplo 2).....	52
Figura 5.36: Magnitud de la velocidad en función del tiempo de la curva de Bézier 2 (ejemplo 2).....	52
Figura 5.37: Magnitud de la aceleración en función del tiempo de la curva de Bézier 2 (ejemplo 2).....	53
Figura 5.38: Comparación entre “obtenerCoeDif” y “obtenerCoeDif_2” (ejemplo 2).....	53
Figura 5.39: Comparación entre trayectorias de la antigua y la nueva Bézier 1 (ejemplo 2).....	54
Figura 5.40: Comparación entre velocidades de la antigua y la nueva Bézier 1 (ejemplo 2)	54
Figura 5.41: Comparación entre aceleraciones de la antigua y la nueva Bézier 1 (ejemplo 2) ..	54
Figura 5.42: Comparación entre magnitudes de velocidad de la antigua y la nueva Bézier 1 (ejemplo 2)	55

Figura 5.43: Comparación entre magnitudes de aceleración de la antigua y la nueva Bézier 1 (ejemplo 2)	55
Figura 5.44: Esferas 1 y 2 originales y 1 modificando su posición (ejemplo 2)	55
Figura 5.45: Zoom a la Figura 5.46 (ejemplo 2)	56
Figura 5.46: Esferas 1 y 2 originales (ejemplo 2).....	56
Figura 5.47: Zoom 1 a la Figura 5.48 (ejemplo 2).....	56
Figura 5.48: Zoom 2 a la Figura 5.48 (ejemplo 2).....	57
Figura 5.49: Esfera 1 modificando su posición y 2 original (ejemplo 2).....	57
Figura 5.50: Zoom 1 a la Figura 5.51 (ejemplo 2).....	57
Figura 5.51: Zoom 2 a la Figura 5.51 (ejemplo 2).....	58

Índice de tablas

Tabla 1: Cuadro de precios básicos	67
Tabla 2: Unidades de obra del capítulo 1	67
Tabla 3: Unidades de obra del capítulo 2	68
Tabla 4: Unidades de obra del capítulo 3	68
Tabla 5: Cuadro de presupuestos parciales	68
Tabla 6: Resumen del presupuesto total del proyecto	69

Parte I

Memoria

Capítulo 1

Introducción

1.1. Antecedentes

1.1.1. Descripción del programa MATLAB

Como bien es sabido, la programación es cada vez más necesaria y avanza a pasos agigantados dentro del mundo tecnológico e industrial actual. Es por ello que, para la realización de este TFG, se hará uso de una de las herramientas más destacables en el mundo de la programación y utilizados en universidades y centros de I+D: MATLAB.

Esta herramienta, acrónimo del nombre *Matrix Laboratory* (“laboratorio de matrices”), dispone de un lenguaje de programación propio y es fácilmente utilizable por cualquier persona con nociones básicas en el uso de vectores y matrices. Como puede presuponerse de lo dicho anteriormente, combina un entorno de escritorio basado en análisis iterativo y los procesos de diseño con un lenguaje que expresa las matemáticas de matrices y *arrays*, además de permitir ver cómo funcionan diferentes algoritmos con unos datos específicos [1]. Todas estas prestaciones serán aplicadas constantemente durante este proyecto.

No obstante, esto no es lo único que puede realizarse con MATLAB, y es que este programa dispone en su interior de otras herramientas tales como Simulink. Esta herramienta permite diseñar y simular el sistema requerido antes de convertirlo en hardware, es decir, crear rápidamente prototipos con los que poder apreciar si funcionaría en un objeto real [2]. De esta manera, pueden combinarse fácilmente MATLAB y Simulink incluyendo el código o algoritmo implementado en un diagrama de bloques, como los vistos en la asignatura Tecnología Electrónica de GITI, de Simulink [2].

Por otra parte, MATLAB dota a sus usuarios de funciones y herramientas para visualizar datos en 2D y 3D, y su lenguaje puede ser ejecutado tanto en *Command Window* como a través de Scripts y funciones de extensión *.m, haciendo uso de ambos fenómenos durante la elaboración del TFG. También pueden ser incluidos por el usuario comentarios que ayuden a la comprensión de ciertos fragmentos del código.

Diseño de una herramienta para el cálculo de trayectorias libres de colisión basadas en curvas de Bézier para objetos voladores

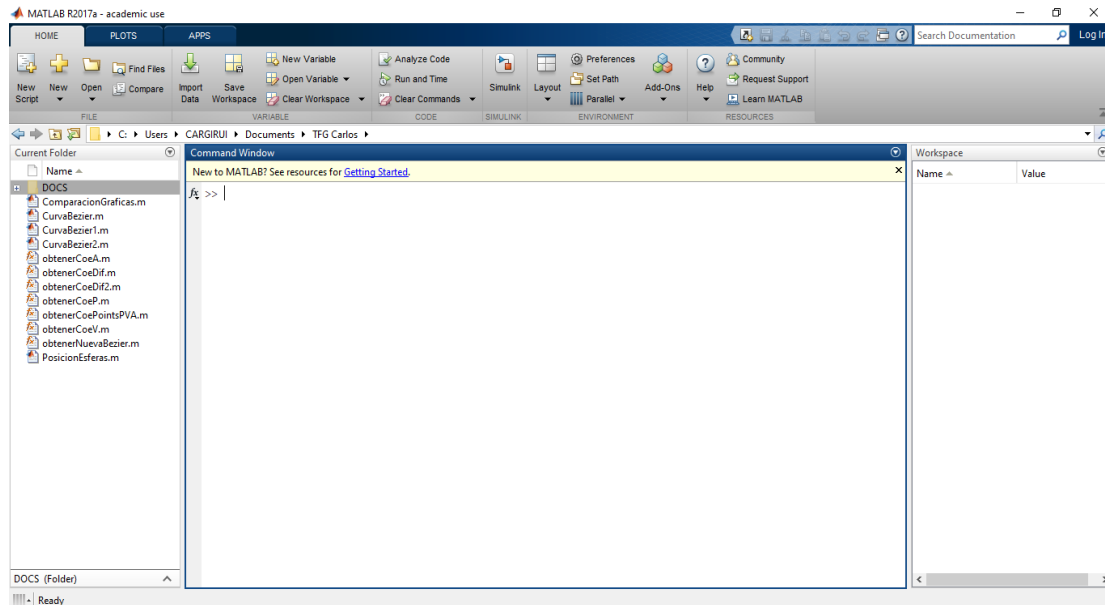


Figura 1.1: Ventana principal de MATLAB, donde se aprecia principalmente el Command Window

Respecto a la ventana principal mostrada inmediatamente después de abrir el programa [Figura 1.1], puede observarse que se dispone, además del *Command Window* nombrado anteriormente, de: un espacio de trabajo o *Workspace* donde se van almacenando las diferentes variables creadas, sean simples valores, vectores o matrices, mostrando además el valor mínimo y máximo de cada variable y teniendo la posibilidad de borrarlas todas mediante el comando “*clear all*”; y un archivo actual o *Current Folder* en el que se puede apreciar la carpeta con la que se está trabajando, además de mostrar todos los archivos pertenecientes a dicha carpeta.

Por otro lado, también existe la posibilidad de ejecutar Scripts o funciones, siendo el manejo de éstas ligeramente diferentes a la primera opción. En el caso de las funciones, se basan en el pase de parámetros de entrada con los que se trabaja para obtener como resultado unos parámetros de salida diferentes. La forma de crear una función sería la que se muestra en la imagen inferior [Figura 1.2].

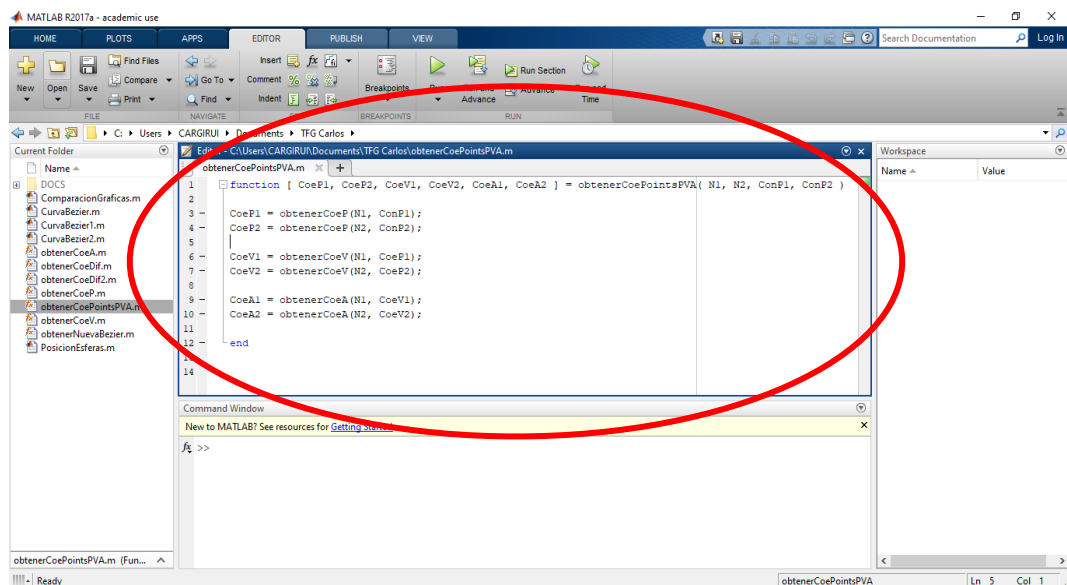


Figura 1.2: Apariencia de una función en MATLAB

1.1.2. Origen de las curvas de Bézier

Los orígenes de las Curvas de Bézier se remontan a mediados del siglo XX, concretamente durante los años 60, cuando Pierre Étienne Bézier (1910-1999) decidió elaborar un sistema para el trazado de dibujos técnicos. Este fantástico ingeniero francés buscaba una forma sencilla de realizar diseños aeronáuticos y principalmente automovilísticos, ya que trabajó durante 42 años



Figura 1.3: Pierre Étienne Bézier [6]

para el fabricante francés Renault, y de esta manera construyó un método de descripción matemática de las curvas que fue implementado en programas de CAD [3] [4] [5].

Pierre Bézier trabajó paralelamente a Paul de Casteljaou en prácticamente el mismo modelo, de manera que ambos llegaron esencialmente al mismo tipo de curvas, pero aplicando diferentes modelos matemáticos [7]. De hecho, de Casteljaou, que también trabajaba para otra gran empresa automovilística conocida como Citroën [3], elaboró un algoritmo numéricamente estable que lleva su mismo nombre para examinar el comportamiento de las curvas de Bézier [4].

Actualmente, encontramos este modelo matemático en multitud de trabajos de ingeniería, y el equivalente computacional se denomina *spline* [8].

Respecto a la forma de definir geoméricamente una curva de Bézier en tres dimensiones, ésta no muestra excesiva complicación: una curva de orden N está definida por N+1 puntos, siendo el primero y el último conocidos como “nodos” o “puntos de anclaje” por empezar y finalizar en ellos la curva, y el resto son denominados “puntos de control” o “manejadores” [4], puesto que son los encargados de otorgar forma a la curva, es decir, son puntos que determinan dónde se localiza cada uno de los puntos de inflexión de la curva. No obstante, generalmente todos los puntos suelen ser llamados “Control Points”, evitando así cualquier tipo de discusión o ambigüedad. La unión mediante rectas de estos Control Points forma lo que se conoce comúnmente como polígono de control.

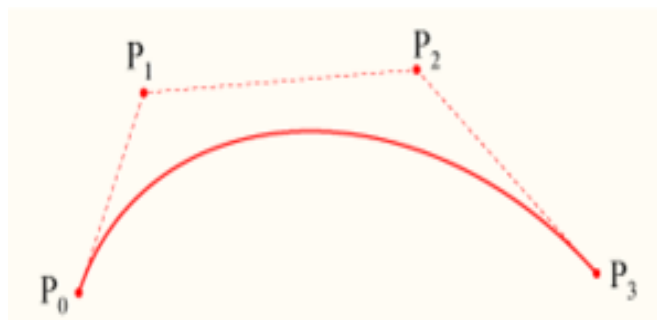


Figura 1.4: Apariencia de una curva de Bézier de N=3 [4]

Sumada a las destacadas anteriormente, otra de las propiedades fundamentales de las curvas de Bézier es el cumplimiento del principio de invariancia afín, consistente en que dados los puntos p_0, p_1, \dots, p_N del plano o del espacio afín, un método C para construir curvas a partir de dichos puntos y una transformación afín f se da la relación $f(C(p_0, p_1, \dots, p_N)) = C(f(p_0), f(p_1), \dots, f(p_N))$ [7].

Otras características de las curvas son la contención en la envolvente convexa de los Control Points o el cumplimiento de la propiedad de la interpolación de los extremos [7]. Sin embargo, no se entrará en mayor detalle.

1.2. Objetivos

Entre los objetivos principales de este TFG se encuentran:

- **Conocer la gran variedad de recursos que contiene la herramienta MATLAB**

El software matemático MATLAB ofrece multitud de comandos que facilitan en gran medida todo el cálculo matemático para la realización de este trabajo y que su máxima complicación reside en un conocimiento básico del uso de matrices y vectores, además de permitir la representación gráfica de trayectorias y esferas en tres dimensiones de una manera relativamente cómoda. Es por ello que se ha decidido realizar el diseño del algoritmo en esta plataforma multifuncional.

- **Definir los parámetros más influyentes en el uso de curvas de Bézier**

De lo dicho anteriormente se sabe que las curvas de Bézier juegan un papel fundamental en el desarrollo de este trabajo, debido a que son las encargadas de dotar a los objetos voladores de una trayectoria tridimensional a seguir. Por este motivo, se trata de dar un enfoque más detallado a los parámetros que las definen y así poder ser conscientes del gran abanico de posibilidades que ofrece este modelo matemático en el panorama ingenieril.

- **Diseñar un algoritmo capaz de calcular nuevas trayectorias para objetos voladores**

A partir de la información adquirida del tratamiento de curvas de Bézier, se ha diseñado un algoritmo en MATLAB basado principalmente en funciones capaz de calcular una nueva trayectoria para uno de los dos objetos voladores en caso de detectarse una futura colisión entre estos. Esta implementación se ha realizado con éxito gracias a la sencillez de trabajo que ofrece la herramienta en el uso de todo tipo de *arrays*.

Para simular el objeto volador en representaciones gráficas, se ha decidido utilizar esferas de cierto radio como símil.

1.3. Motivación y justificación

El uso del modelo diseñado por el ingeniero francés Pierre Bézier se expandió en la década de 1960 entre gran parte del sector automovilístico, y actualmente se emplean cuantiosamente en programas de diseño asistido por ordenador (CAD) siendo conocidas como *splines*. Por tanto, su utilización puede extenderse desde algo tan sencillo como un colgante hasta ejemplos complejos como puede ser un modelo naval.

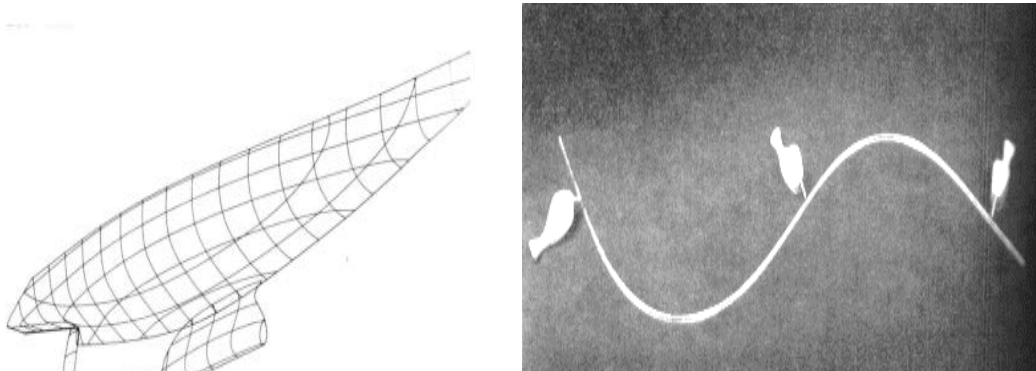


Figura 1.5: Barco y colgante representados con splines [7]

Sin embargo, no se había incidido en la posibilidad de emplear las famosas curvas de Bézier para crear trayectorias a seguir por cualquier tipo de objeto volador con elevada precisión, y menos aún en conseguir que un objeto varíe su movimiento en caso de detección de una colisión en un tiempo futuro a partir de éstas. Debido a ello, la motivación del presente TFG es la de diseñar una herramienta capaz de reunir estas funciones sin un elevado coste computacional y haciendo uso de algunos de los métodos numéricos más destacados durante estos cuatro años de grado.

1.4. Estructura del documento

El presente Trabajo Final de Grado está organizado en dos documentos principales: la memoria y el presupuesto.

En primer lugar, desea destacarse que la presente memoria está orientada a la programación en MATLAB, por lo que primeramente se introduce el concepto teórico para posteriormente mostrar cómo se implementa en código. La memoria del trabajo presenta 5 apartados:

- **Curvas de Bézier**, donde se pretende dar a conocer en mayor profundidad este concepto, primero introduciendo cómo sería la ecuación para una curva de orden $N=3$ y posteriormente para una curva de orden N cualquiera, donde entran en juego coeficientes binomiales, además de los ya nombrados “Control Points” o los “Coefficient Points” de posición. Por otro lado, se mostrará cómo se implementan estos conceptos en el lenguaje de MATLAB.
- **Obtención de la distancia mínima entre curvas**, donde se hará uso de los Coefficient Points para calcular la velocidad y aceleración tridimensionales. Asimismo, se introducirá el concepto de Coefficient Points en diferencias, necesarios para obtener la ecuación de la que surgirá la distancia mínima entre curvas. Al igual que en el anterior apartado, también se hablará de la programación en MATLAB.
- **Obtención de la nueva curva de Bézier**, desde las condiciones que debe cumplir esta nueva curva hasta la optimización realizada por el método de mínimos cuadrados, e incluyendo también la programación en MATLAB.
- **Ejemplos**, donde ya podrán apreciarse representaciones gráficas que ayudarán en la visualización del concepto.
- **Conclusiones**, donde se valorarán los resultados obtenidos, dificultades y trabajos futuros del algoritmo implementado.

Posteriormente se encuentra la bibliografía empleada en algunos subcapítulos y finalmente se muestra el presupuesto del proyecto.

Curvas de Bézier

Ya se ha dicho anteriormente que las curvas de Bézier constituyen la columna vertebral del presente TFG, y que su desarrollo se dio lugar entorno a los años 1960. Por tanto, todos los conceptos que se introducen en este capítulo no han sido creados por el alumno, sino que son simples adaptaciones de los mostrados en la referencia [4] al propósito de este trabajo.

2.1. Curva de Bézier 3D de orden N

2.1.1. Expresión general de una curva de Bézier de orden N: Control Points

Una curva Bézier de orden N está definida por N+1 Control Points: $P_0, P_1, P_2, \dots, P_{N-1}, P_N \in \mathbb{R}^3$, donde el inicio y el final de la curva vienen dados por P_0 y P_N respectivamente, y P_1, P_2, \dots, P_{N-1} son los encargados de darle forma sin estar contenidos en ella.

Dado que $P_0, P_1, P_2, \dots, P_{N-1}, P_N \in \mathbb{R}^3$ son puntos 3D, se deduce que cada $P_i = (P_{i_x}, P_{i_y}, P_{i_z})$.

Cada punto de la curva puede escribirse en función de un parámetro $\lambda \in [0,1]$ que está directamente relacionado con el instante de tiempo de vuelo en el que se encuentra el objeto. De esta manera, $P_{os}(\lambda) \in \mathbb{R}^3$ es cada punto de la trayectoria que describe el objeto volador mediante una curva de Bézier y puede ser expresada de la siguiente forma:

$$P_{os}(\lambda) = \binom{N}{0} \lambda^0 (1-\lambda)^{N-0} P_0 + \binom{N}{1} \lambda^1 (1-\lambda)^{N-1} P_1 + \binom{N}{2} \lambda^2 (1-\lambda)^{N-2} P_2 + \dots + \binom{N}{N-1} \lambda^{N-1} (1-\lambda)^{N-(N-1)} P_{N-1} + \binom{N}{N} \lambda^N (1-\lambda)^{N-N} P_N \quad (2.1)$$

Siendo la relación directa entre λ y el tiempo:

$$tiempo(\lambda) = t_{ini} + \lambda \cdot (t_{fin} - t_{ini}) \quad (2.2)$$

Por lo que el término general de la expresión (2.1) es:

$$P_{os}(\lambda) = \sum_{i=0}^N \binom{N}{i} \lambda^i (1-\lambda)^{N-i} P_i, \quad \forall \lambda \in [0,1] \quad (2.3)$$

Donde se determina el coeficiente binomial como:

$$\binom{N}{i} = \frac{N!}{i!(N-i)!} \quad (2.4)$$

Como puede extraerse de la expresión (2.1), cuando $\lambda = 0$ entonces $Pos(\lambda) = P_0$ equivalente al punto de origen, mientras que si $\lambda = 1$ se obtiene $Pos(\lambda) = P_N$ equivalente al punto final de la trayectoria.

2.1.2. Coefficient Points

No obstante, existe una forma de escribir la ecuación general de una curva de Bézier mucho más sencilla que la mostrada en (2.7), y que se construye a partir de los denominados Coefficient Points.

De la misma forma que con los Control Points, una curva de Bézier se expresa a partir de $N+1$ Coefficient Points $p_0, p_1, p_2, \dots, p_{N-1}, p_N \in \mathbb{R}^3$, por lo que $Pos(\lambda)$ puede escribirse de manera equivalente como:

$$Pos(\lambda) = p_0 + p_1\lambda + p_2\lambda^2 + \dots + p_{N-1}\lambda^{N-1} + p_N\lambda^N = \sum_{i=0}^N \binom{N}{i} \lambda^i (1-\lambda)^{N-i} P_i$$

$$Pos(\lambda) = \sum_{i=0}^N p_i \lambda^i \quad (2.5)$$

Y la relación entre Coefficient Points y Control Points es:

$$p_i = \binom{N}{i} \left(\sum_{j=0}^i \binom{i}{j} (-1)^j P_{i-j} \right) \quad (2.6)$$

$$p_i = (p_{ix}, p_{iy}, p_{iz})$$

$$i = 0, 1, 2, \dots, N$$

Aplicando la relación (2.10) los Coefficient Points quedarían tal que así:

$$p_0 = P_0, \quad p_1 = \binom{N}{1} (P_1 - P_0) = N(P_1 - P_0), \quad p_2 = \binom{N}{2} (P_2 - 2P_1 + P_0),$$

$$p_3 = \binom{N}{3} (P_3 - 3P_2 + 3P_1 - P_0), \dots,$$

$$p_N = \binom{N}{N} \left(\binom{N}{0} P_N - \binom{N}{1} P_{N-1} + \binom{N}{2} P_{N-2} + \dots + \binom{N}{N-1} (-1)^{N-1} P_1 + \binom{N}{N} (-1)^N P_0 \right)$$

Nótese que:

- **Posición inicial** ($\lambda = 0$): $Pos(0) = P_0 = p_0$
- **Posición final** ($\lambda = 1$): $Pos(1) = P_N = p_0 + p_1 + \dots + p_{N-1} + p_N$
- **Posición intermedia** ($\lambda = 0.5$): $Pos(0.5) = p_0 + 0.5p_1 + \dots + 0.5^{N-1}p_{N-1} + 0.5^N p_N$

2.1.3. Coefficient Points de velocidad

Además de la posición que sigue el objeto volador, también es interesante conocer cómo varía su velocidad durante el vuelo. Es por ello que se calculan unos nuevos Coefficient Points, siendo en esta ocasión los de la ecuación de velocidad.

Así, desarrollando la expresión general (2.5) de una curva de Bézier en función de los Coefficient Points:

$$Pos(\lambda) = p_0 + p_1\lambda + p_2\lambda^2 + \dots + p_N\lambda^N, \quad \forall \lambda \in [0,1]$$

Simplemente debe calcularse su derivada para obtener la ecuación de velocidad, y con ello los Coefficient Points de velocidad:

$$\begin{aligned} Vel(\lambda) &= Pos'(\lambda) = p_1 + 2p_2\lambda + 3p_3\lambda^2 + \dots + Np_N\lambda^{N-1} \\ &= v_0 + v_1\lambda + v_2\lambda^2 + \dots + v_{N-1}\lambda^{N-1} \\ Vel(\lambda) &= \sum_{i=0}^{N-1} v_i\lambda^i \end{aligned} \quad (2.7)$$

Siendo, por tanto, la relación entre Coefficient Points de posición y de velocidad la siguiente:

$$\begin{aligned} v_i &= (i + 1) \cdot p_{i+1} \\ i &= 0, 1, 2, \dots, N - 1 \end{aligned} \quad (2.8)$$

Con $v_0, v_1, v_2, \dots, v_{N-1} \in \mathbb{R}^3$, por lo que la representación gráfica de la velocidad será, al igual que con la posición, tridimensional.

2.1.4. Coefficient Points de aceleración

De igual manera que con la velocidad, también pueden obtenerse fácilmente los Coefficient Points para la ecuación de aceleración, sea aplicando la segunda derivada de la ecuación de posición o la primera derivada de la de velocidad.

Así, desarrollando la expresión general (2.7) de la velocidad del objeto volador:

$$Vel(\lambda) = v_0 + v_1\lambda + v_2\lambda^2 + \dots + v_{N-1}\lambda^{N-1}$$

Simplemente debe calcularse su derivada para alcanzar la ecuación de aceleración, y con ello los Coefficient Points de aceleración:

$$\begin{aligned} Ace(\lambda) &= Pos''(\lambda) = 2p_2 + 6p_3\lambda + 12p_4\lambda^2 + \dots + N(N-1)p_N\lambda^{N-2} = Vel'(\lambda) = \\ &= v_1 + 2v_2\lambda + 3v_3\lambda^2 + \dots + (N-1)v_{N-1}\lambda^{N-2} = \\ &= a_0 + a_1\lambda + a_2\lambda^2 + \dots + a_{N-2}\lambda^{N-2} \\ Ace(\lambda) &= \sum_{i=0}^{N-2} a_i\lambda^i \end{aligned} \quad (2.9)$$

Siendo, por tanto, la relación entre Coefficient Points de velocidad y de aceleración la siguiente:

$$\begin{aligned} a_i &= (i + 1)(i + 2) \cdot p_{i+2} = (i + 1) \cdot v_{i+1} \\ i &= 0, 1, 2, \dots, N - 2 \end{aligned} \quad (2.10)$$

Con $a_0, a_1, a_2, \dots, a_{N-2} \in \mathbb{R}^3$, por lo que la representación gráfica de la aceleración será, al igual que con la posición y la velocidad, tridimensional.

En el próximo subcapítulo, se verá que la implementación de las curvas de Bézier mediante Coefficient Points es mucho más eficaz que mediante Control Points, ya que requieren un coste computacional mucho menor que se traduce en menos tiempo de ejecución.

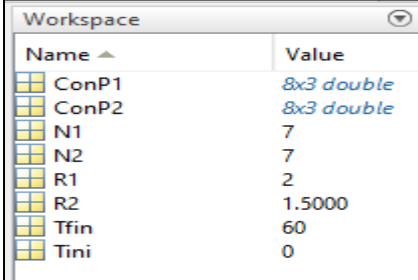
2.2. Programación en MATLAB

Como se ha comentado anteriormente, todo este desarrollo teórico será implementado en el software matemático MATLAB con un lenguaje muy sencillo y basado principalmente en el uso de matrices y vectores. De esta forma, tratará de explicarse cómo se transcribe mediante el uso de funciones y *scripts* todo lo explicado en párrafos atrás.

El primer paso a realizar es introducir una serie de parámetros de entrada a partir de los cuales se ejecutará el resto del código, siendo éstos:

- **Control Points** de las dos curvas de Bézier: nombrados en las funciones como “ConP1” y “ConP2”.
- **Orden** de cada una de las curvas: no existe necesidad de que sean iguales, y se denominan “N1” y “N2”.
- **Radio** de cada uno de los objetos voladores cuya trayectoria se basa en estas curvas: nombrados “R1” y “R2”.
- **Tiempo** durante el que los objetos se encuentran volando: al instante inicial se le llama “Tini” y al instante final “Tfin”.

Puede observarse en la Figura 2.1 que estos parámetros se almacenan en un espacio llamado *Workspace* donde se podrán tener a mano en todo momento, y que generalmente “ConP1” y “ConP2” serán matrices de dimensión (N+1) x 3, correspondiendo cada una de las columnas a las coordenadas X, Y o Z. “N1” y “N2” deben ser números naturales y el resto de variables pueden ser números decimales.



Name	Value
ConP1	8x3 double
ConP2	8x3 double
N1	7
N2	7
R1	2
R2	1.5000
Tfin	60
Tini	0

Figura 2.1: Aspecto de Workspace al introducir parámetros

2.2.1. Función “obtenerCoePointsPVA”

Una vez introducidos los parámetros de entrada en el espacio de trabajo, es el momento de presentar la primera función, llamada “obtenerCoePointsPVA”. Esta función se encarga de, a partir de los parámetros “ConP1”, “ConP2”, “N1” y “N2”, obtener los Coefficient Points necesarios para representar en tres dimensiones tanto la posición como la velocidad y la aceleración, así como la magnitud de la velocidad y de la aceleración en función del tiempo.

```
obtenerCoePointsPVA.m x obtenerCoeP.m x obtenerCoeV.m x obtenerCoeA.m x CurvaBezier.m x +
1 function [ CoeP1, CoeP2, CoeV1, CoeV2, CoeA1, CoeA2 ] = obtenerCoePointsPVA( N1, N2, ConP1, ConP2 )
2
3     CoeP1 = obtenerCoeP(N1, ConP1);
4     CoeP2 = obtenerCoeP(N2, ConP2);
5
6     CoeV1 = obtenerCoeV(N1, CoeP1);
7     CoeV2 = obtenerCoeV(N2, CoeP2);
8
9     CoeA1 = obtenerCoeA(N1, CoeV1);
10    CoeA2 = obtenerCoeA(N2, CoeV2);
11
12 end
13
14
```

Figura 2.2: Función “obtenerCoePointsPVA”

Como puede apreciarse en la Figura 2.2, dentro de esta función se hace una llamada a otras funciones, por lo que el método escogido consiste en una función global que contiene otras funciones secundarias encargadas de realizar las tareas más costosas. Por tanto, los Coefficient Points de posición “CoeP1” y “CoeP2” se obtienen mediante la función “obtenerCoeP”, los de velocidad “CoeV1” y “CoeV2” mediante la función “obtenerCoeV” y los de aceleración “CoeA1” y “CoeA2” mediante la función “obtenerCoeA”.

2.2.2. Función “obtenerCoeP”

```
obtenerCoePointsPVA.m x obtenerCoeP.m x obtenerCoeV.m x obtenerCoeA.m x +
1 function [ CoeP ] = obtenerCoeP(N, ConP)
2
3     %Mediante esta función obtenemos los Coefficient Points relativos a la
4     %posición del objeto volador a través de los Control Points
5     factN=factorial(N);
6
7     for i=0: 1: N
8         facti=factorial(i);
9         factNi=factorial(N-i);
10        prod=factN/(facti*factNi);
11        for j=0: 1: i
12            factj=factorial(j);
13            factij=factorial(i-j);
14            prod2=facti/(factj*factij);
15            prod3=prod*prod2;
16            if(j==0)
17                CoeP(i+1,:) = prod3*ConP(i-j+1,:);
18            else
19                CoeP(i+1,:) = CoeP(i+1,:) + prod3*((-1)^(j))*ConP(i-j+1,:);
20            end
21        end
22    end
23 end
```

Figura 2.3: Función “obtenerCoeP”

Se puede apreciar que la función “obtenerCoeP” recibe unos parámetros “N” y “ConP” genéricos y devuelve otro parámetro “CoeP” también genérico, de manera que puede ser llamada indefinidamente. Básicamente trata de simularse la ecuación mostrada en (2.6).

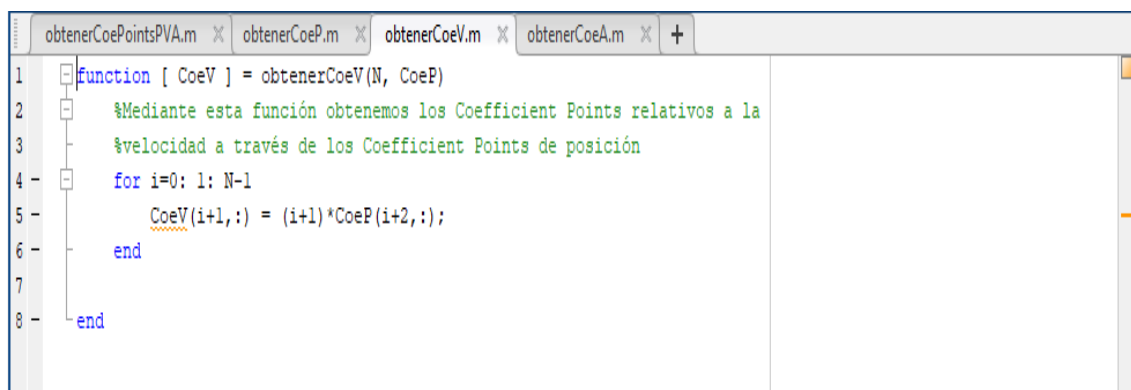
Respecto al código contenido en la función, debe destacarse que se hace uso de dos bucles ‘for’, que no son más que incrementos de cierto parámetro desde un valor inicial hasta otro valor final de la forma $for\ i = i_{ini} : \Delta i : i_{fin}$, así como de sentencias ‘if’ y ‘else’, consistentes en ejecutar una acción u otra según se cumplan o no las condiciones exigidas dentro de la sentencia.

En la línea 4 se calcula el factorial de “N”, que será constante para todas las iteraciones del bucle ‘for’ principal, y ya dentro de dicho bloque se calcula para cada iteración el factorial de “i”, siendo “i” el indicador del Coefficient Point que se está calculando, y de “N-i”, con los que obtener el primer coeficiente binomial, cuya forma se ve en (2.4), de la relación (2.6) entre Coefficient y Control Points.

Ya dentro del bucle ‘for’ contenido en el principal, se calcula el segundo coeficiente binomial, y lo que se realiza entre las líneas 15 y 18 es el sumatorio de todos los Control Points necesarios para obtener cada Coefficient Point, observándose que para la primera iteración de “j” siempre acudirá a la sentencia ‘if’ y en el resto de iteraciones posteriores irá a la sentencia ‘else’.

Por último, destacar que en las líneas 16 y 18 se empieza desde “CoeP (1, :)” y no desde “CoeP (0, :)” porque en MATLAB no existe la posición cero de un vector o una matriz, y que el símbolo “:” se utiliza para referirse a todas las columnas de una fila concreta.

2.2.3. Función “obtenerCoeV”



```
1 function [ CoeV ] = obtenerCoeV(N, CoeP)
2 %Mediante esta función obtenemos los Coefficient Points relativos a la
3 %velocidad a través de los Coefficient Points de posición
4 for i=0: 1: N-1
5     CoeV(i+1,:) = (i+1)*CoeP(i+2,:);
6 end
7
8 end
```

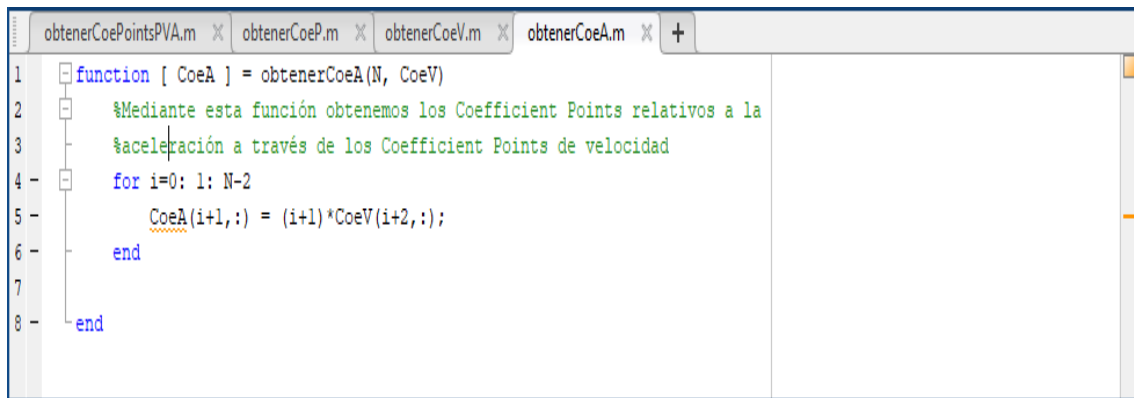
Figura 2.4: Función “obtenerCoeV”

Al igual que en el caso anterior, “obtenerCoeV” es una función que recibe unos parámetros genéricos “N” y “CoeP” y devuelve como parámetro de salida un genérico “CoeV”, pudiendo utilizarse todas las veces que se desee. Sin embargo, el código implementado aquí es mucho más sencillo pues ya se han obtenido los Coefficient Points de posición, en los cuales se sustenta el resto del trabajo.

De esta forma, lo que se realiza entre las líneas 4 y 6 es lo mostrado en la ecuación (2.8) para calcular los Coefficient Points de velocidad. Asimismo, recordar que en el bucle ‘for’ se parte de “CoeV (1, :)” porque no existe la posición cero en este programa, pero éste debe asociarse con v_0 .

Además, esta matriz “CoeV” dispondrá de una fila menos que “CoeP”, pues se corresponde con la primera derivada de la ecuación de posición y, por tanto, se pierde el primer término.

2.2.4. Función “obtenerCoeA”

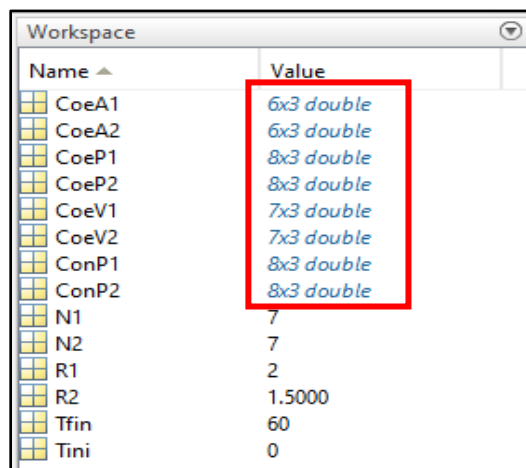


```
1 function [ CoeA ] = obtenerCoeA(N, CoeV)
2 %Mediante esta función obtenemos los Coefficient Points relativos a la
3 %aceleración a través de los Coefficient Points de velocidad
4 for i=0: 1: N-2
5     CoeA(i+1,:) = (i+1)*CoeV(i+2,:);
6 end
7
8 end
```

Figura 2.5: Función “obtenerCoeA”

De la misma forma que “obtenerCoeP” y “obtenerCoeV”, “obtenerCoeA” también es una función genérica que recibe un orden de curva “N” y unos Coefficient Points de velocidad “CoeV” y devuelve unos Coefficient Points de aceleración “CoeA”.

La programación implementada en esta función es muy sencilla, pues únicamente se está realizando una primera derivada de la ecuación de velocidad o una segunda derivada de la de posición para obtener los términos de la aceleración según la ecuación (2.10), siendo en este caso dos menos que para la posición, detalle que puede apreciarse en la Figura 2.6.



Name	Value
CoeA1	6x3 double
CoeA2	6x3 double
CoeP1	8x3 double
CoeP2	8x3 double
CoeV1	7x3 double
CoeV2	7x3 double
ConP1	8x3 double
ConP2	8x3 double
N1	7
N2	7
R1	2
R2	1.5000
Tfin	60
Tini	0

Figura 2.6: Tamaño de las matrices de Coefficient Points

2.2.5. Función “obtenerPVA”

```

1  function [Pos, Vel, Ace, Vel_mag, Ace_mag, tiempo]=obtenerPVA(CoeP, CoeV, CoeA, N, Tini, Tfin)
2  z=1;
3  diftiempo=Tfin-Tini;
4  for t=0.0: 0.01: 1.0
5      tiempo(z)=Tini+t*diftiempo;
6      for i=0: 1: N
7          if(i==0)
8              Pos(z, :)= CoeP(i+1, :);
9              Vel(z, :)= CoeV(i+1, :);
10             Ace(z, :)= CoeA(i+1, :);
11         else
12             Pos(z, :)= Pos(z, :) + (t^(i))*CoeP(i+1, :);
13             if(i<=N-1)
14                 Vel(z, :)= Vel(z, :) + (t^(i))*CoeV(i+1, :);
15             end
16             if(i<=N-2)
17                 Ace(z, :)= Ace(z, :) + (t^(i))*CoeA(i+1, :);
18             end
19         end
20     end
21     Vel_mag(z)=sqrt(Vel(z,1)^2+Vel(z,2)^2+Vel(z,3)^2)/diftiempo;
22     if(z==1)
23         dif=Vel_mag(z);
24     else
25         dif=Vel_mag(z)-Vel_mag(z-1);
26     end
27
28     if(dif>=0)
29         Ace_mag(z)=sqrt(Ace(z,1)^2+Ace(z,2)^2+Ace(z,3)^2)/(diftiempo)^2;
30     else
31         Ace_mag(z)=-sqrt(Ace(z,1)^2+Ace(z,2)^2+Ace(z,3)^2)/(diftiempo)^2;
32     end
33     z=z+1;
34 end
35 end

```

Figura 2.7: Función “obtenerPVA”

En esta función, básicamente se desarrolla entre las líneas 4 y 21 el sumatorio mostrado en las ecuaciones (2.5), (2.7) y (2.9) para obtener las coordenadas de posición, velocidad y aceleración para cada uno de los instantes λ , parámetro que en esta función se representa mediante la letra “t”.

Puede observarse que como parámetros de entrada se envían “CoeP”, “CoeV”, “CoeA”, “N”, “Tini” y “Tfin”, y como parámetros devueltos por la función están las matrices “Pos”, “Vel” y “Ace” y los vectores “Vel_mag”, “Ace_mag” y “tiempo”, que serán utilizados posteriormente para realizar representaciones gráficas en 3D y 2D.

Asimismo, entre las líneas 23 y 39 trata de calcularse el valor absoluto de la velocidad y de la aceleración, así como el instante de tiempo en el que se produce cada uno de estos valores, teniéndose en cuenta tanto aceleraciones como deceleraciones de un instante a otro (líneas 33-37). Para ello, se hace uso de la expresión del tiempo (2.2) y de las dos siguientes:

$$velocidad = \frac{\sqrt{v_x^2 + v_y^2 + v_z^2}}{(t_{fin} - t_{ini})} \quad (2.11)$$

$$aceleración = \frac{\pm \sqrt{a_x^2 + a_y^2 + a_z^2}}{(t_{fin} - t_{ini})^2} \quad (2.12)$$

Donde el signo de la aceleración depende de si la diferencia de velocidad entre un instante y el anterior es positiva o negativa.

2.2.6. Función “mostrarPVA”

```
mostrarPVA.m x +
1 function mostrarPVA(ConP,tiempo,Pos,Vel,Ace,Vel_mag,Ace_mag)
2     figure;
3     plot3(Pos(:,1),Pos(:,2),Pos(:,3),'blue',ConP(:,1),ConP(:,2),ConP(:,3),'green');
4     hold on;
5     scatter3(ConP(:,1),ConP(:,2),ConP(:,3),'green')
6     grid on;
7     title('POSICIÓN del punto que sigue la Bézier')
8
9     figure;
10    plot3(Vel(:,1),Vel(:,2),Vel(:,3),'blue');
11    grid on;
12    title('VELOCIDAD del punto que sigue la Bézier')
13
14    figure;
15    plot3(Ace(:,1),Ace(:,2),Ace(:,3),'blue');
16    grid on;
17    title('ACELERACIÓN del punto que sigue la Bézier')
18
19    figure;
20    plot(tiempo,Vel_mag);
21    grid on;
22    title('Magnitud de la Velocidad(tiempo) de Bézier')
23
24    figure;
25    plot(tiempo,Ace_mag);
26    grid on;
27    title('Magnitud de la Aceleración(tiempo) de Bézier ')
28 end
```

Figura 2.8: Función “mostrarPVA”

Tras haber ejecutado la función “obtenerPVA”, consiguiéndose así las matrices “Pos”, “Vel” y “Ace” y los vectores “Vel_mag”, “Ace_mag” y “tiempo” de cada una de las curvas Bézier, es el momento de realizar representaciones gráficas que aproximen un poco más al lector a comprender el concepto. Para ello, se hace uso de la función “mostrarPVA” que hace uso de estos parámetros así como de la matriz de Control Points “ConP”.

La función devuelve cinco imágenes diferentes, identificadas por cada vez que se hace uso del comando ‘figure’, y puede observarse que en todas ellas se utiliza el comando ‘grid on’ encargado de activar las rejillas de cada cuadrícula y el comando ‘title (‘)’ con el cual insertar un título a cada una de las imágenes.

Por otra parte, también se hace uso del comando ‘plot3(X, Y, Z)’ para realizar representaciones gráficas en 3D, ‘plot(X, Y)’ para conseguir lo mismo que en el comando anterior pero en 2D y ‘scatter3(X, Y, Z)’ que representa círculos en las coordenadas asignadas según X, Y y Z [10]. Además se hace uso del comando ‘hold on’, con el que se agregan diagramas o funciones a una gráfica ya existente, es decir, se produce una superposición [11].

De esta manera, lo que se implementa entre las líneas 2 y 17 son gráficas en tres dimensiones de la trayectoria seguida por los objetos voladores, su velocidad y su aceleración, que serán mostradas en color azul. Además, para la gráfica de la trayectoria o posición trata de superponerse los Control Points, dibujándolos mediante ‘scatter3’, uniéndolos mediante rectas con ‘plot3 para formar el ya nombrado “polígono de control” y mostrándolos en color verde, para demostrar así que el inicio de la trayectoria coincide con P_0 y el final con P_N .

Asimismo, entre las líneas 19 y 27 se implementan gráficas en dos dimensiones de la magnitud de la velocidad y de la aceleración en función del tiempo que está volando el objeto, pudiendo observarse, realizando una comparación de gráficas, que la aceleración se hace negativa en caso de una reducción de velocidad de un instante a otro.

Obtención de la distancia mínima entre curvas

3.1. Coefficient Points en diferencias

3.1.1. Influencia del orden de ambas curvas

Una vez que se ha entrado en un mayor grado de conocimiento de las curvas de Bézier, es el momento de trabajar en el verdadero objetivo de este TFG.

Llegado este punto, es sabido que se va a trabajar con dos curvas de Bézier sin necesidad de que éstas tengan el mismo orden ni, en consecuencia, el mismo número de Coefficient Points (a mayor cantidad de Coefficient Points, mayor facilidad de trazado de la curva). Por tanto, el problema parte de:

Curva 1: $p(\lambda)$ de orden N , con $p_0, p_1, p_2, \dots, p_N \in \mathbb{R}^3$ Coefficient Points

Curva 2: $q(\lambda)$ de orden M , con $q_0, q_1, q_2, \dots, q_M \in \mathbb{R}^3$ Coefficient Points

$$\forall \lambda \in [0,1]$$

Con lo que se tiene:

$$p(\lambda) = p_0 + p_1\lambda + p_2\lambda^2 + \dots + p_N\lambda^N = (p(\lambda)_x, p(\lambda)_y, p(\lambda)_z) \quad (3.1)$$

$$q(\lambda) = q_0 + q_1\lambda + q_2\lambda^2 + \dots + q_M\lambda^M = (q(\lambda)_x, q(\lambda)_y, q(\lambda)_z) \quad (3.2)$$

La idea a desarrollar a partir de estos parámetros de entrada es la de obtener unos Coefficient Points en diferencias, que no son más que la resta entre los propios de una y otra curva, con los cuales poder completar una ecuación $m(\lambda)$ que devuelva la distancia absoluta entre ambas curvas para cada instante λ del intervalo $[0,1]$. Cabe recordar que, cuanto más reducido sea el valor de λ para cada iteración, mayor será la precisión a la hora de dibujar las curvas.

De esta forma, lo que se está buscando es una ecuación de la forma:

$$m(\lambda) = p(\lambda) - q(\lambda) = (p_0 + p_1\lambda + p_2\lambda^2 + \dots + p_N\lambda^N) - (q_0 + q_1\lambda + q_2\lambda^2 + \dots + q_M\lambda^M)$$

No obstante, y como se ha dicho anteriormente, el orden de ambas curvas no tiene necesidad de ser el mismo, ni por tanto el número de Coefficient Points, por lo que la influencia de ambos órdenes en la ecuación de diferencias será notable.

Por tanto, existirán dos casos: uno en el que el orden más elevado sea el de la primera curva, la cual se desea modificar posteriormente en caso de necesidad, u otro donde lo sea el de la segunda curva.

1º Caso: máx. = N

$$m(\lambda) = (p_0 - q_0) + (p_1 - q_1)\lambda + (p_2 - q_2)\lambda^2 + \dots + (p_M - q_M)\lambda^M + p_{M+1}\lambda^{M+1} + \dots + p_N\lambda^N \quad (3.3)$$

Donde se observa que todos los términos son positivos incluso después de sobrepasar el mínimo M.

2º Caso: máx. = M

$$m(\lambda) = (p_0 - q_0) + (p_1 - q_1)\lambda + (p_2 - q_2)\lambda^2 + \dots + (p_N - q_N)\lambda^N - q_{N+1}\lambda^{N+1} - \dots - q_M\lambda^M \quad (3.4)$$

Donde puede verse que tras alcanzar el mínimo N todos los términos de la ecuación se hacen negativos.

Sin embargo, ambos casos pueden reducirse a la expresión general:

$$m(\lambda) = m_0 + m_1\lambda + m_2\lambda^2 + \dots + m_w\lambda^w \quad (3.5)$$

$$\text{donde: } w = \max(N, M)$$

Siendo los Coefficient Points en diferencias:

$$m_i = (p_i - q_i) \in \mathbb{R}^3 \quad (3.6)$$

$$i = 0, 1, 2, \dots, w$$

3.1.2. Minimización de la ecuación de diferencias

Una vez obtenida la ecuación de diferencias (3.5), el siguiente obstáculo a resolver es el de obtener la raíz $\lambda_{min} \in [0,1]$, equivalente al instante de tiempo donde se producirá una distancia mínima (o máxima, pero no es de interés en esta implementación) entre ambas curvas.

Para ello, simplemente debe calcularse la primera derivada de la ecuación de diferencias e igualarla a cero, esto es:

$$\min \|m(\lambda)\| = \|p(\lambda) - q(\lambda)\| \quad (3.7)$$

Un error típico es sustituir esta minimización por una búsqueda exhaustiva del punto de máximo acercamiento, consistente en calcular la distancia entre curvas para cada incremento de λ y comprobar si ésta es menor que la anterior, ya que requiere un coste computacional que, por definición, no se acepta para saber si dos objetos voladores van a chocar en el futuro. Sin embargo, no está de más implementar este método para comprobar si el resultado formal se asemeja.

Tras aplicar la primera derivada, se obtendrá una ecuación de las raíces de grado $2w - 1$ cuya apariencia será la siguiente:

$$d_0 + d_1\lambda + d_2\lambda^2 + \dots + d_{2w-2}\lambda^{2w-2} + d_{2w-1}\lambda^{2w-1} = 0 \quad (3.8)$$

$$d_i \in \mathbb{R}$$

$$i = 0, 1, 2, \dots, 2w - 1$$

Por tanto, a partir de esta ecuación se obtendrán un conjunto de raíces que determinarán la distancia mínima entre ambas curvas.

No obstante, no todas ellas serán válidas para dicho cálculo, ya que deben cumplir una serie de condiciones: en primer lugar, deben ser raíces reales, descartando aquellas que sean complejas (naturalmente estas λ_{min} están directamente relacionadas con el tiempo que se mantiene el objeto en vuelo, por lo que sería un sinsentido disponer de un tiempo con término imaginario); y en segundo lugar, su valor debe estar contenido entre 0 y 1 (en este caso, sin considerar ambos límites), tal y como se viene insistiendo a lo largo de todo el documento.

En el caso de que no se cumpliera la segunda condición, sería tan sencillo como averiguar si la colisión se produce al inicio o al fin del vuelo. Todo lo narrado hasta ahora será trasladado al lenguaje propio de MATLAB, el cual dispone de una serie de comandos que simplifican drásticamente algunas de las operaciones.

3.2. Programación en MATLAB

En el presente subcapítulo se tratará de extender los conceptos de Coefficient Points en diferencias y minimización de la ecuación en diferencias al lenguaje empleado en MATLAB mediante el uso de una función donde se emplean nuevos comandos que facilitan el proceso.

3.2.1. Función “obtenerCoeDif”

```

1 function [raicesRealesPol,Distancias,distMin,distRadios,raizMin,mMin]=obtenerCoeDif(N1,N2,CoeP1,CoeP2,R1,R2)
2     MAX = max(N1,N2);
3     MIN = min(N1,N2);
4     for i = 0 : 1 : MAX
5         if(i>MIN)
6             if(MIN==N2)
7                 CoeDif(i+1,:) = CoeP1(i+1,:);
8             else
9                 CoeDif(i+1,:) = -CoeP2(i+1,:);
10            end
11        else
12            CoeDif(i+1,:) = (CoeP1(i+1,)-CoeP2(i+1,:));
13        end
14    end
15
16    CompX = CoeDif(:,1);
17    CompY = CoeDif(:,2);
18    CompZ = CoeDif(:,3);
19
20    for i=0:1:MAX-1
21        DerivCompX(i+1) = (i+1)*CompX(i+2);
22        DerivCompY(i+1) = (i+1)*CompY(i+2);

```

```
23 -     DerivCompZ(i+1) = (i+1)*CompZ(i+2);
24 - end
25
26 -     MultX = 2*conv(CompX, DerivCompX);
27 -     MultY = 2*conv(CompY, DerivCompY);
28 -     MultZ = 2*conv(CompZ, DerivCompZ);
29
30 -     %Se utiliza el comando flipud para invertir el orden de los vectores,
31 -     %necesario para utilizar el comando roots
32 -     MultX=flipud(MultX);
33 -     MultY=flipud(MultY);
34 -     MultZ=flipud(MultZ);
35
36 -     Polinomio=MultX+MultY+MultZ;
37
38 -     %Se aplica el comando roots para obtener los ceros, debiendo comprobar
39 -     %si son reales y si se encuentran entre 0 y 1
40 -     raicesPol=roots(Polinomio);
41
42 -     z=1;
43 -     distMin=100;
44
45 -     for w=0:1:(2*MAX-2)
46 -         tf=isreal(raicesPol(w+1));
47 -         if(tf==1 && raicesPol(w+1)>0 && raicesPol(w+1)<1)
48 -             raicesRealesPol(z)=raicesPol(w+1);
49 -             if(w<2*MAX-2)
50 -                 z=z+1;
51 -             end
52 -         end
53 -     end
54
55 -     %Una vez almacenada la raiz minima en raizMin, la utilizamos para calcular la distancia mínima
56 -     for w=1:1:z
57 -         for i=0:1:MAX
58 -             if(i==0)
59 -                 m(w,:)=CoeDif(i+1,:);
60 -             else
61 -                 m(w,:)=m(w,:)+CoeDif(i+1,:)*raicesRealesPol(w)^(i);
62 -             end
63 -         end
64 -         Distancias(w) = norm(m(w,:));
65 -         if(Distancias(w)<distMin)
66 -             distMin=Distancias(w);
67 -             raizMin=raicesRealesPol(w);
68 -             mMin=m(w,:);
69 -         end
70 -     end
71 -     distRadios=distMin-(R1+R2);
72 - end
```

Figura 3.1: Función “obtenerCoeDif”

En esta función se da lugar a algunos de los pasos más importantes en la elaboración de este trabajo, tal y como se ha nombrado en el subcapítulo anterior. La función “obtenerCoeDif” recibe

como parámetros de entrada los órdenes “N1” y “N2” de cada curva Bézier, sus respectivos Coefficient Points de posición “CoeP1” y “CoeP2” y los radios “R1” y “R2” de cada una de las esferas, y devuelve como salida:

- Un vector “**raicesRealesPol**” donde se almacenan las raíces reales (descartando las que disponen de parte imaginaria) obtenidas en la minimización de la ecuación de diferencias.
- Un parámetro “**raizMin**” en el que se guarda una de las raíces reales correspondiente a la distancia mínima entre curvas.
- Un vector “**Distancias**” donde se guardan las distancias absolutas para cada una de las raíces reales.
- Un parámetro “**distMin**” que almacena el menor valor contenido en el vector “Distancias” y que corresponde a la distancia mínima entre curvas.
- Un parámetro “**distRadios**” que almacena la distancia mínima entre ambas esferas, la cual es realmente de interés.
- Un vector “**mMin**” que almacena la distancia mínima en forma vectorial.

En las líneas 2 y 3 se obtienen respectivamente el orden mayor “MAX” y el menor “MIN” de entre los dos recibidos mediante los comandos ‘max (X, Y)’ y ‘min (X, Y)’ con la intención de reducir entre las líneas 4 y 14 los dos posibles casos introducidos en las ecuaciones (3.3) y (3.4) a un único bucle ‘for’ muy sencillo. Como puede apreciarse, el tamaño de la matriz “CoeDif”, que almacena los Coefficient Points en diferencias entre las dos curvas, será MAX x 3, siendo cada columna para una coordenada distinta.

Una vez obtenidos los Coefficient Points en diferencias con los que completar la respectiva ecuación, lo que se hace entre las líneas 16 y 18 es separar éstos en su componente X “CompX”, Y “CompY” y Z “CompZ”, para posteriormente obtener sus derivadas “DerivCompX”, “DerivCompY” y “DerivCompZ”.

Con motivo de haber almacenado todos estos coeficientes y sus derivadas en vectores, se ofrece la posibilidad de implementar la ecuación de las raíces de la siguiente forma:

En primer lugar, se almacenan en unos vectores “MultX”, “MultY” y “MultZ” el resultado de aplicar, para cada coordenada, la primera derivada a la ecuación (3.7). De esta manera, se utiliza el comando ‘conv (A, B)’ con el que operar con los vectores “CompX” y “DerivCompX” (en este caso, para la componente X) como si de un producto de polinomios se tratara. Posteriormente, se realiza una suma de estos vectores en la línea 36 y se almacena en un nuevo vector “Polinomio”, el cual contiene los coeficientes de la ecuación de las raíces (3.8).

Finalmente, en la línea 40 se aplica a “Polinomio” el comando ‘roots(C)’, equivalente a igualar a cero y calcular las raíces, almacenándolas en un vector “raicesPol”. Para aplicar dicho comando, anteriormente ha sido necesario invertir el orden de “MultX”, “MultY” y “MultZ” mediante otro comando ‘flipud(X)’, puesto que éste requiere que el orden del polinomio sea de la forma $C_N \cdot X^N + \dots + C_1 \cdot X + C_0$ y el orden inicial era el contrario.

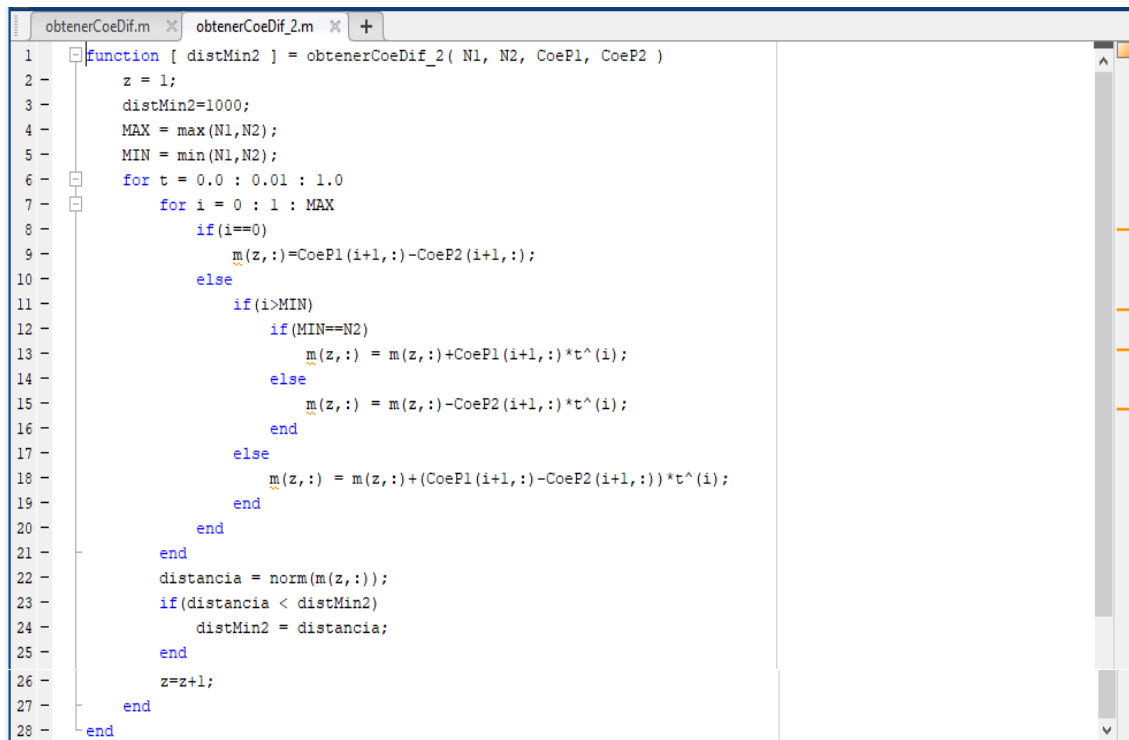
Sin embargo, en “raicesPol” se almacenan raíces del polinomio que pueden ser tanto reales como imaginarias, pero sólo son de interés las que se encuentran en el plano real. Por todo ello, entre las líneas 45 y 53 se comprueba para cada una de estas raíces las dos condiciones establecidas en el último párrafo del subcapítulo anterior, es decir, $\lambda_{min} \in \mathbb{R}$ y $\lambda_{min} \in [0,1]$, generándose un nuevo vector “raicesRealesPol” que almacena aquellas que las cumplen.

Posteriormente, entre las líneas 56 y 70 se implementa un bucle ‘for’ en el que se calcula para cada una de las raíces reales la distancia vectorial “m (w, :)” mediante la expresión (3.5), se pasa a distancia absoluta mediante el comando ‘norm(X)’ almacenándose en un vector “Distancias” y

se comprueba si es más pequeña que la obtenida anteriormente, en cuyo caso se guarda en la variable “distMin”, su raíz asociada en “raizMin” y el valor vectorial de la distancia en “mMin”.

Finalmente, a la distancia entre curvas “distMin” se le resta la suma de los radios de ambas esferas para obtener “distRadios”, equivalente a la distancia entre objetos, siendo ésta realmente de importancia para saber si el cálculo de una nueva curva Bézier es necesaria o no.

3.2.2. Función “obtenerCoeDif_2”



```
1 function [ distMin2 ] = obtenerCoeDif_2( N1, N2, CoeP1, CoeP2 )
2     z = 1;
3     distMin2=1000;
4     MAX = max(N1,N2);
5     MIN = min(N1,N2);
6     for t = 0.0 : 0.01 : 1.0
7         for i = 0 : 1 : MAX
8             if(i==0)
9                 m(z,:)=CoeP1(i+1,:)-CoeP2(i+1,:);
10            else
11                if(i>MIN)
12                    if(MIN==N2)
13                        m(z,:) = m(z,:)+CoeP1(i+1,:)*t^(i);
14                    else
15                        m(z,:) = m(z,:)-CoeP2(i+1,:)*t^(i);
16                    end
17                else
18                    m(z,:) = m(z,:)+(CoeP1(i+1,:)-CoeP2(i+1,:))*t^(i);
19                end
20            end
21        end
22        distancia = norm(m(z,:));
23        if(distancia < distMin2)
24            distMin2 = distancia;
25        end
26        z=z+1;
27    end
28 end
```

Figura 3.2: Función “obtenerCoeDif_2”

Lo que se muestra en la Figura 3.2 nombrado como “obtenerCoeDif_2” es la primera implementación que se realizó para la obtención de la distancia mínima entre curvas antes de alcanzar la función resultante “obtenerCoeDif”.

En esta función se empieza de igual forma que en la definitiva, es decir, estableciendo el máximo y el mínimo entre ambos órdenes “N1” y “N2” y creando una variable “distMin2” con un valor inicial elevado que sea sustituido fácilmente en posteriores líneas de código. Después de ello, se crea un bucle ‘for’ en el que se calcula, para todo $\lambda \in [0,1]$ con reducidos incrementos de 0.01 entre una iteración y otra, la distancia absoluta entre ambas curvas mediante la ecuación (3.5) y aplicando el comando ‘norm(X)’, y finalmente se comprueba si ésta es menor a la almacenada en “distMin2” para actualizarla en caso afirmativo.

Por tanto, este método consiste en la búsqueda exhaustiva del punto de máximo acercamiento para todo instante de tiempo, algo que a priori podría parecer correcto. Sin embargo, esta forma de obtención requiere un coste computacional muy elevado, lo que se traduce en una insuficiencia de tiempo por parte del objeto volador para rectificar su futura trayectoria.

No obstante, y como se demostrará en el capítulo dedicado a ejemplos, el resultado obtenido aplicando esta función da una aproximación de qué se debe obtener en “obtenerCoeDif”, por lo que será muy útil posteriormente.

Obtención de la nueva curva de Bézier

4.1. Condiciones necesarias en la nueva Bézier

Retomando la información proporcionada anteriormente, en el presente subcapítulo se parte de un conjunto de valores reales $\lambda_{min} \in [0,1]$ tales que indican el instante de tiempo en el que las respectivas curvas se encuentran a menor distancia, además de poder ser obtenida dicha distancia mínima. Sin embargo, es necesario tener en cuenta que en el espacio tridimensional lo que verdaderamente es de interés no es la diferencia entre trayectorias seguidas por los objetos, sino la distancia entre los propios objetos.

De esta forma, a la distancia obtenida aplicando un λ_{min} concreto se le ha de restar la suma de los radios de ambas esferas, es decir, considerando un R_{esf_P} y un R_{esf_Q} concretos:

$$dist_{min} = \|p(\lambda_{min}) - q(\lambda_{min})\| - (R_{esf_P} + R_{esf_Q}) \leq 0 \quad (4.1)$$

$$R_{esf_P} > 0, \quad R_{esf_Q} > 0$$

En el caso de que esta distancia sea mayor que 0, no es necesario realizar ninguna otra acción correctora puesto que los objetos (en el caso en el que se trabaja, pueden considerarse esferas de un radio determinado) no chocan entre sí, pero si ésta es igual a 0, equivalente a estar produciéndose un contacto, o menor que 0, equivalente a estar “atravesándose” (físicamente imposible), es necesario anticiparse a este suceso elaborando una nueva Bézier para el primer objeto. En resumen:

Si existe colisión $\rightarrow dist_{min} \leq 0 \rightarrow$ Construir nueva Bézier de P

$$Instante de tiempo: t_{min} = t_{ini} + \lambda_{min}(t_{fin} - t_{ini}) \quad (4.2)$$

$$t_{min} \in [t_{ini}, t_{fin}]$$

Esto puede traducirse de la siguiente forma: en el instante de tiempo t_{min} obtenido a partir del parámetro λ_{min} y situado entre los tiempos de inicio y de fin del vuelo, el centro de la esfera se encuentra en $p(\lambda_{min})$, donde existe colisión, y se desea que esté en otra posición sin colisión $\overline{p(\lambda_{min})}$ que podría calcularse como:

$$\overline{p(\lambda_{min})} = p(\lambda_{min}) - dist_{min} \cdot (1,05) \cdot \overline{v_{dist_{min}}} \quad (4.3)$$

Donde:

$$\overline{v_{dist_{min}}} = \frac{(p(\lambda_{min}) - q(\lambda_{min}))}{\|p(\lambda_{min}) - q(\lambda_{min})\|} \quad (4.4)$$

Es un vector unitario de módulo la unidad que contiene la dirección y el sentido del vector $(p(\lambda_{min}) - q(\lambda_{min}))$ y 1,05 es equivalente a un porcentaje de seguridad del 5% que no induce un gran error y garantiza con mayor fiabilidad que no existirá colisión entre la nueva curva de Bézier del primer objeto y la original del segundo.

Como es predecible, la ecuación resultante de (4.3) será una de las condiciones a cumplir por la nueva curva, y se combinará con otras dos para llegar a un resultado final satisfactorio. Por tanto, las condiciones a cumplir serán:

Dados $\overline{p_0}, \overline{p_1}, \overline{p_2}, \dots, \overline{p_N} \in \mathbb{R}^3$ *Coefficient Points*:

1. Posición inicial: $P_0 = p_0$ en el instante t_{ini}

Se desea que la nueva curva de Bézier surja del mismo origen que la antigua, es decir:

$$\overline{P_0} = P_0 \rightarrow \overline{p_0} = p_0 \quad (4.5)$$

Debe recordarse que el propósito no es el de construir una curva completamente diferente sino una que se asemeje lo máximo posible a la antigua produciendo el mínimo desvío posible.

2. Posición final: $P_N = \sum p_i$ en el instante t_{fin}

Por el mismo motivo que el comentado en la primera condición, es necesario que la nueva curva finalice en la misma posición que la antigua, es decir:

$$\sum \overline{p_i} = \sum p_i \quad (4.6)$$

$$\overline{p_1} + \overline{p_2} + \overline{p_3} + \dots + \overline{p_N} = p_1 + p_2 + p_3 + \dots + p_N$$

Siendo $\overline{p_i} = (\overline{p_{x_i}}, \overline{p_{y_i}}, \overline{p_{z_i}})$ y $p_i = (p_{x_i}, p_{y_i}, p_{z_i})$:

$$\overline{p_{x_1}} + \overline{p_{x_2}} + \dots + \overline{p_{x_N}} = p_{x_1} + p_{x_2} + \dots + p_{x_N} \quad (4.7)$$

$$\overline{p_{y_1}} + \overline{p_{y_2}} + \dots + \overline{p_{y_N}} = p_{y_1} + p_{y_2} + \dots + p_{y_N} \quad (4.8)$$

$$\overline{p_{z_1}} + \overline{p_{z_2}} + \dots + \overline{p_{z_N}} = p_{z_1} + p_{z_2} + \dots + p_{z_N} \quad (4.9)$$

3. Debe pasar por $\overline{p(\lambda_{min})}$ en el instante t_{min}

Finalmente, la nueva trayectoria debe pasar por $\overline{p(\lambda_{min})}$, lo cual puede expresarse de la siguiente forma:

$$\begin{aligned} \overline{p_1} \lambda_{min} + \overline{p_2} \lambda_{min}^2 + \dots + \overline{p_N} \lambda_{min}^N &= \\ &= p_1 \lambda_{min} + p_2 \lambda_{min}^2 + \dots + p_N \lambda_{min}^N - 1,05 dist_{min} \overline{v_{dist_{min}}} \end{aligned} \quad (4.10)$$

De igual manera que en la 2ª condición: (4.11)

$$\overline{p_{x_1}} \lambda_{min} + \overline{p_{x_2}} \lambda_{min}^2 + \dots + \overline{p_{x_N}} \lambda_{min}^N = p_{x_1} \lambda_{min} + p_{x_2} \lambda_{min}^2 + \dots + p_{x_N} \lambda_{min}^N - 1,05 dist_{min} \overline{v_{dist_{minx}}}$$

(4.12)

$$\overline{p_{y_1}}\lambda_{min} + \overline{p_{y_2}}\lambda_{min}^2 + \dots + \overline{p_{y_N}}\lambda_{min}^N = p_{y_1}\lambda_{min} + p_{y_2}\lambda_{min}^2 + \dots + p_{y_N}\lambda_{min}^N - 1,05 \overrightarrow{dist_{min} v_{dist_{miny}}}$$

(4.13)

$$\overline{p_{z_1}}\lambda_{min} + \overline{p_{z_2}}\lambda_{min}^2 + \dots + \overline{p_{z_N}}\lambda_{min}^N = p_{z_1}\lambda_{min} + p_{z_2}\lambda_{min}^2 + \dots + p_{z_N}\lambda_{min}^N - 1,05 \overrightarrow{dist_{min} v_{dist_{minz}}}$$

Tal y como puede apreciarse en las condiciones 2 y 3, no se han tenido en cuenta los términos $\overline{p_0}$ y p_0 en sus respectivas ecuaciones. Esto es debido a que el cumplimiento de la primera condición certifica que ambos términos son exactamente iguales y, por tanto, se anulan.

4.2. Optimización por el método de mínimos cuadrados

Todas las condiciones expuestas anteriormente deben verse sometidas a un proceso de optimización a partir del cual se obtendrá la nueva curva de Bézier sin colisión. Este proceso es llevado a cabo mediante el método de los mínimos cuadrados, una técnica de análisis numérico en la que, siendo conocidos una variable independiente, otra dependiente y un conjunto de funciones, se busca la función continua que mejor se aproxime a los datos de acuerdo con el criterio de mínimo error cuadrático, tal y como puede distinguirse en la Figura 4.1 [9].

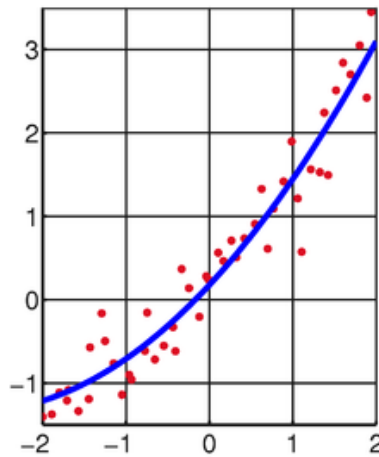


Figura 4.1: Ajuste de un conjunto de datos con mínimos cuadrados [9]

De esta manera, lo que desea es conseguir el mejor ajuste en función de los datos que se poseen induciendo el menor error posible. Para el caso de estudio, se dispone de un conjunto de Coefficient Points de la curva de Bézier original y del parámetro λ_{min} mediante los que se elaboran las matrices y vectores pertinentes para completar la optimización.

Todo los conceptos que van a introducirse a continuación en relación a sistemas matriciales, matrices y vectores son adaptaciones a las variables de este trabajo de la referencia [14]

La primera etapa del proceso consistirá en resolver el sistema matricial:

$$C \cdot x = d \tag{4.14}$$

Donde:

$$x = \begin{bmatrix} \overline{p_{x_1}} \\ \overline{p_{y_1}} \\ \overline{p_{z_1}} \\ \overline{p_{x_2}} \\ \overline{p_{y_2}} \\ \overline{p_{z_2}} \\ \vdots \\ \overline{p_{x_N}} \\ \overline{p_{y_N}} \\ \overline{p_{z_N}} \end{bmatrix} \quad (4.15)$$

“x” es una matriz columna de dimensión $3N \times 1$ que contiene las Coefficient Points de la nueva curva a desarrollar, es decir, las incógnitas;

$$C = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \lambda_{min} & 0 & 0 & \lambda_{min}^2 & 0 & 0 & \dots & \lambda_{min}^N & 0 \\ 0 & \lambda_{min} & 0 & 0 & \lambda_{min}^2 & 0 & 0 & 0 & \lambda_{min}^N \\ 0 & 0 & \lambda_{min} & 0 & 0 & \lambda_{min}^2 & 0 & 0 & \lambda_{min}^N \end{bmatrix} \quad (4.16)$$

“C” es una matriz no cuadrada de dimensión $6 \times 3N$ que contiene la parte izquierda de las condiciones 2 y 3 representadas en las ecuaciones (4.6) y (4.10);

$$d = \begin{bmatrix} p_{x_1} + p_{x_2} + \dots + p_{x_N} \\ p_{y_1} + p_{y_2} + \dots + p_{y_N} \\ p_{z_1} + p_{z_2} + \dots + p_{z_N} \\ p_{x_1} \lambda_{min} + p_{x_2} \lambda_{min}^2 + \dots + p_{x_N} \lambda_{min}^N - 1,05 \overline{dist_{min} v_{dist_{min_x}}} \\ p_{y_1} \lambda_{min} + p_{y_2} \lambda_{min}^2 + \dots + p_{y_N} \lambda_{min}^N - 1,05 \overline{dist_{min} v_{dist_{min_y}}} \\ p_{z_1} \lambda_{min} + p_{z_2} \lambda_{min}^2 + \dots + p_{z_N} \lambda_{min}^N - 1,05 \overline{dist_{min} v_{dist_{min_z}}} \end{bmatrix} \quad (4.17)$$

Y “d” es otra matriz columna de dimensión 6×1 que contiene la parte derecha de las ecuaciones (4.6) y (4.10) establecidas por las condiciones 2 y 3.

Por otra parte, hay que realizar una segunda etapa consistente en minimizar el error cuadrático cometido al construir la nueva trayectoria, es decir:

$$\min_x \|A \cdot x - B\|^2 = \left\| (\overline{p_1} \lambda_c + \overline{p_2} \lambda_c^2 + \dots + \overline{p_N} \lambda_c^N) - (p_1 \lambda_c + p_2 \lambda_c^2 + \dots + p_N \lambda_c^N) \right\|^2 \quad (4.18)$$

Donde $\lambda_c \in (0,1)$ es un parámetro conocido que en todo momento debe ser diferente de λ_{min} y que no precisa de un valor constante, sino que parte de un valor ligeramente superior a 0 (en la programación que se verá en el próximo subcapítulo, se empieza con un valor $\lambda_{c_{ini}} = \lambda_{c_1} = 0.01$) y se van produciendo pequeños incrementos de su valor (en la programación, este incremento es $\Delta \lambda_c = 0.01$) hasta alcanzar un valor ligeramente inferior a 1 (por ejemplo, en la implementación en MATLAB este valor es $\lambda_{c_{fin}} = 0.99$).

Realizando esto se pretende que, exceptuando el instante de tiempo en el que la nueva curva se encuentra en λ_{min} , ésta se diferencie en la menor medida posible de la original, ya que lo único que se desea es un ligero desvío en el momento de la supuesta colisión.

Por tanto, se pueden definir las siguientes matrices:

$$A = \begin{bmatrix} \lambda_{c_1} & 0 & 0 & \lambda_{c_1}^2 & 0 & 0 & \dots & \lambda_{c_1}^N & 0 & 0 \\ 0 & \lambda_{c_1} & 0 & 0 & \lambda_{c_1}^2 & 0 & \dots & 0 & \lambda_{c_1}^N & 0 \\ 0 & 0 & \lambda_{c_1} & 0 & 0 & \lambda_{c_1}^2 & \dots & 0 & 0 & \lambda_{c_1}^N \\ \lambda_{c_2} & 0 & 0 & \lambda_{c_2}^2 & 0 & 0 & \dots & \lambda_{c_2}^N & 0 & 0 \\ 0 & \lambda_{c_2} & 0 & 0 & \lambda_{c_2}^2 & 0 & \dots & 0 & \lambda_{c_2}^N & 0 \\ 0 & 0 & \lambda_{c_2} & 0 & 0 & \lambda_{c_2}^2 & \dots & 0 & 0 & \lambda_{c_2}^N \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \lambda_{c_{fin}} & 0 & 0 & \lambda_{c_{fin}}^2 & 0 & 0 & \dots & \lambda_{c_{fin}}^N & 0 & 0 \\ 0 & \lambda_{c_{fin}} & 0 & 0 & \lambda_{c_{fin}}^2 & 0 & \dots & 0 & \lambda_{c_{fin}}^N & 0 \\ 0 & 0 & \lambda_{c_{fin}} & 0 & 0 & \lambda_{c_{fin}}^2 & \dots & 0 & 0 & \lambda_{c_{fin}}^N \end{bmatrix} \quad (4.19)$$

“A” es una matriz de una cantidad muy elevada e indeterminada de filas y de 3N columnas (en el ejemplo expuesto en párrafos anteriores, se trata de una matriz de dimensión 297x3N) que contiene muchos parámetros λ_c distintos con los que reducir al máximo esa diferencia de posición entre curvas para cada instante.

$$x = \begin{bmatrix} \overline{p_{x_1}} \\ \overline{p_{y_1}} \\ \overline{p_{z_1}} \\ \overline{p_{x_2}} \\ \overline{p_{y_2}} \\ \overline{p_{z_2}} \\ \vdots \\ \overline{p_{x_N}} \\ \overline{p_{y_N}} \\ \overline{p_{z_N}} \end{bmatrix}$$

“x” es la misma matriz de dimensión 3Nx1 utilizada en la primera etapa del proceso.

$$B = \begin{bmatrix} p_{x_1} \lambda_{c_1} + p_{x_2} \lambda_{c_1}^2 + \dots + p_{x_N} \lambda_{c_1}^N \\ p_{y_1} \lambda_{c_1} + p_{y_2} \lambda_{c_1}^2 + \dots + p_{y_N} \lambda_{c_1}^N \\ p_{z_1} \lambda_{c_1} + p_{z_2} \lambda_{c_1}^2 + \dots + p_{z_N} \lambda_{c_1}^N \\ p_{x_1} \lambda_{c_2} + p_{x_2} \lambda_{c_2}^2 + \dots + p_{x_N} \lambda_{c_2}^N \\ p_{y_1} \lambda_{c_2} + p_{y_2} \lambda_{c_2}^2 + \dots + p_{y_N} \lambda_{c_2}^N \\ p_{z_1} \lambda_{c_2} + p_{z_2} \lambda_{c_2}^2 + \dots + p_{z_N} \lambda_{c_2}^N \\ \vdots \\ p_{x_1} \lambda_{c_{fin}} + p_{x_2} \lambda_{c_{fin}}^2 + \dots + p_{x_N} \lambda_{c_{fin}}^N \\ p_{y_1} \lambda_{c_{fin}} + p_{y_2} \lambda_{c_{fin}}^2 + \dots + p_{y_N} \lambda_{c_{fin}}^N \\ p_{z_1} \lambda_{c_{fin}} + p_{z_2} \lambda_{c_{fin}}^2 + \dots + p_{z_N} \lambda_{c_{fin}}^N \end{bmatrix} \quad (4.20)$$

Y “B” es una matriz columna de una cantidad de filas elevada e igual al número de filas de la matriz “A” y de una columna (en el ejemplo expuesto anteriormente referente a lo aplicado en la implementación del algoritmo, es una matriz de dimensión 297x1) que contiene la ecuación de

posición, separada en sus respectivas coordenadas, de la antigua curva de Bézier para cada uno de los instantes de tiempo λ_c .

Una vez expuestas las dos etapas del proceso de optimización, se puede alcanzar una solución tal que combine ambas en un sistema matricial con una resolución bastante sencilla:

$$\begin{bmatrix} \hat{x} \\ \hat{p} \end{bmatrix} = \left[\begin{array}{c|c} 2A^T \cdot A & C^T \\ \hline C & 0 \end{array} \right]^{-1} \cdot \begin{bmatrix} 2A^T B \\ d \end{bmatrix} \quad (4.21)$$

Siendo \hat{x} el vector que contiene los nuevos Coefficient Points referentes a la nueva curva, y \hat{p} otro vector donde se almacenan unos multiplicadores lagrangianos que no serán tenidos en cuenta para ningún tipo de cálculo, por lo que se descartará su uso.

Como se puede observar:

$$\begin{aligned} 2A^T \cdot A : (3N \times \text{filasA}) \cdot (\text{filasA} \times 3N) &\rightarrow 3N \times 3N \\ C^T : 3N \times 6 \\ C : 6 \times 3N \\ 0 : 6 \times 6 \end{aligned}$$

Por tanto, la matriz:

$$M = \begin{bmatrix} 2A^T \cdot A & C^T \\ C & 0 \end{bmatrix} \quad (4.22)$$

Tiene una dimensión $(3N + 6) \times (3N + 6)$ con determinante no nulo, es decir, se trata de una matriz cuadrada no singular de la que puede obtenerse su inversa de mismas dimensiones, equivalente a calcular:

$$M^{-1} = \begin{bmatrix} 2A^T \cdot A & C^T \\ C & 0 \end{bmatrix}^{-1} = \frac{1}{|M|} \cdot \text{adj}(M) \quad (4.23)$$

Por otra parte, se tiene:

$$\begin{aligned} 2A^T \cdot B : (3N \times \text{filasA}) \cdot (\text{filasA} \times 1) &\rightarrow 3N \times 1 \\ d : 6 \times 1 \end{aligned}$$

Por lo que la matriz columna

$$S = \begin{bmatrix} 2A^T B \\ d \end{bmatrix} \quad (4.24)$$

Tiene dimensión $(3N + 6) \times 1$, y el producto

$$\begin{bmatrix} \hat{x} \\ \hat{p} \end{bmatrix} = M^{-1} \cdot S \quad (4.25)$$

Será de tamaño $(3N + 6) \times 1$, es decir, es una matriz columna donde las primeras 3N filas almacenan los nuevos Coefficient Points y las siguientes 6 filas guardan los multiplicadores lagrangianos, que carecen de uso en el presente trabajo.

Finalmente, sólo queda utilizar los nuevos Coefficient Points para calcular la nueva ecuación de posición $PosNueva(\lambda)$ que determina la trayectoria a seguir por la esfera, ya sin colisión con su homóloga:

$$PosNueva(\lambda) = \sum_{i=0}^N \bar{p}_i \lambda^i \quad (4.26)$$

En el caso de existir más de un instante de tiempo en el que las esferas chocasen entre sí, lo único que debería hacerse es volver a aplicar las tres condiciones que debe tener la nueva curva, puesto que, como se podrá ver en el siguiente subcapítulo, todos los instantes λ_{min} se almacenan en un mismo vector, siendo por tanto fácilmente localizables y todo el código rápidamente reutilizable.

4.3. Programación en MATLAB

En este subcapítulo se muestra una función encargada de calcular las matrices y vectores necesarios para aplicar la optimización por mínimos cuadrados, dando como resultado una nueva matriz de Coefficient Points o un mensaje indicador de que no ha sido necesario crearla.

4.3.1. Función “*obtenerNuevaBezier*”



```
1 function [ CoePlNueva, Error ] = obtenerNuevaBezier( distRadios, distMin, mMin, raizMin, CoePl, N1)
2     Vdist = mMin/distMin;
3
4     if(distRadios<=0)
5         CoePlNueva(1,:) = CoePl(1,:);
6
7         C=zeros(6,3*N1);
8         for i=1:1:3
9             for j=1:3:(3*N1)
10                if(i==1)
11                    C(i,j)=1;
12                else
13                    if(i==2)
14                        C(i,j+1)=1;
15                    else
16                        C(i,j+2)=1;
17                    end
18                end
19            end
20        end
21
22        for i=4:1:6
23            z=1;
24            for j=1:3:(3*N1)
25                if(i==4)
26                    C(i,j)=raizMin^(z);
27                else
28                    if(i==5)
29                        C(i,j+1)=raizMin^(z);
30                    else
31                        C(i,j+2)=raizMin^(z);
32                    end
33                end
34                z=z+1;
35            end
36        end
37
38        for i=1:1:3
39            for j=1:1:N1
40                if(j==1)
41                    d(i)=CoePl(j+1,i);
42                else
43                    d(i)=d(i)+CoePl(j+1,i);
44                end
45            end
46        end
47
48        for i=4:1:6
49            for j=1:1:N1
50                if(j==1)
```

Diseño de una herramienta para el cálculo de trayectorias libres de colisión basadas en curvas de Bézier para objetos voladores

```
51 -         d(i)=CoePl(j+1,i-3)*(raizMin)^(j)-distRadios*1.05*Vdist(i-3);
52 -     else
53 -         d(i)=d(i)+CoePl(j+1,i-3)*(raizMin)^(j);
54 -     end
55 - end
56 - end
57 -
58 - A=zeros(297,3*N1);
59 - s=1;
60 - for raizConocida=0.01:0.01:0.99
61 -     for i=s:1:s+2
62 -         z=1;
63 -         for j=1:3:(3*N1)
64 -             if(i==s)
65 -                 A(i,j)=raizConocida^z;
66 -             else
67 -                 if(i==s+1)
68 -                     A(i,j+1)=raizConocida^z;
69 -                 else
70 -                     A(i,j+2)=raizConocida^z;
71 -                 end
72 -             end
73 -             z=z+1;
74 -         end
75 -     end
76 -     s=s+3;
77 - end
78 -
79 -
80 - s=1;
81 - for raizConocida=0.01:0.01:0.99
82 -     y=1;
83 -     for i=s:1:s+2
84 -         for j=1:1:N1
85 -             if(j==1)
86 -                 B(i)=CoePl(j+1,y)*(raizConocida)^(j);
87 -             else
88 -                 B(i)=B(i)+CoePl(j+1,y)*(raizConocida)^(j);
89 -             end
90 -         end
91 -         y=y+1;
92 -     end
93 -     s=s+3;
94 - end
95 -
96 - M=[(2*A'*A) C'; C zeros(6,6)];
97 - S=[2*A'*B'; d'];
98 - X=M\S;
99 -
100 - h=1;
101 - for i=2:1:N1+1
102 -     CoePlNueva(i,:)=X(h) X(h+1) X(h+2);
103 -     h=h+3;
104 - end
105 -
106 - e=h;
107 - for i=1:1:6
108 -     Error(i)=X(e);
109 -     e=e+1;
110 - end
111 - fprintf("Se han calculado correctamente los nuevos Coefficient Points de la Bezier 1\n");
112 - else
113 -     fprintf("No existe colision entre las esferas con trayectorias basadas en sus Bezier originales\n");
114 -     CoePlNueva=CoePl;
115 -     Error=zeros(e-h,1);
116 - end
117 - end
```

Figura 4.1: Función “obtenerNuevaBezier”

La función “obtenerNuevaBezier” contiene toda la programación matricial y vectorial necesaria para calcular una nueva matriz de Coefficient Points tal que determine una trayectoria ligeramente

distinta para evitar la colisión detectada. Para su obtención, se le pasan como parámetros de entrada los valores “distMin”, ”distRadios”, “raizMin” y “N1”, el vector “mMin” y la matriz “CoeP1”, y devuelve los siguientes parámetros de salida:

- Una matriz “**CoeP1Nueva**” de las mismas dimensiones que “CoeP1” que almacena, efectivamente, los nuevos Coefficient Points de posición.
- Un vector “**Error**” que contiene unos multiplicadores lagrangianos en los que no se entrará en mayor detalle y que, por tanto, carecerá de utilidad en implementaciones posteriores.

En la primera línea de código se calcula el vector unitario mostrado en (4.4) con el que se trabaja en líneas posteriores, y en la línea 4 se aplica una sentencia ‘if’ donde se comprueba si la distancia entre objetos “distRadios” sería menor o igual que cero, en cuyo caso se ejecuta el resto del código. En caso contrario, el programa se iría directamente hasta la línea 112, donde hay una sentencia ‘else’ que devuelve un aviso de que no existe colisión mediante el comando ‘fprintf’, hace que la matriz a devolver “CoeP1Nueva” sea igual a “CoeP1” y que el vector “Error” se complete con ceros, puesto que la operación no se ha realizado y no existe posibilidad de fallo.

Una vez dentro del código interno del ‘if’, en la línea 5 trata de hacerse que la primera fila de la nueva matriz “CoeP1Nueva” sea igual a la primera fila de la antigua “CoeP1”, es decir, que la posición inicial nueva sea la misma que la antigua. Con ello, ya se está cumpliendo la primera condición expuesta en (4.5).

Lo que se hace en la línea 7 es crear una matriz “C”, la cual será finalmente la mostrada en (4.16), llena de ceros y de dimensiones $6 \times 3N_1$ a la que sólo se le modificarán ciertas posiciones mediante el uso de dos bucles ‘for’.

En el primer bucle ‘for’ (líneas 8-20) se trabaja con sus tres primeras filas, existiendo un segundo bucle ‘for’ en el que se aplican saltos de una columna a otra con incrementos de tres posiciones. Por ejemplo, para la primera fila ($i=1$) empieza por la primera columna y en la segunda iteración salta hasta la cuarta columna, y para la segunda fila ($i=2$) empieza por la segunda columna y salta hasta la quinta columna. De esta manera, se van añadiendo unos en las posiciones deseadas.

En el segundo bucle ‘for’ (líneas 22-36) se trabaja con sus tres siguientes y últimas filas, implementando fundamentalmente lo mismo que en el primero, pero cambiando unos por λ_{min}^z . De esta manera, se está reproduciendo la parte izquierda de las ecuaciones (4.6) y (4.10).

Posteriormente al desarrollo de la matriz C, se trata de obtener la matriz o vector columna “d” mostrada en (4.17). Para ello, se implementan dos bucles ‘for’ entre las líneas 38 y 56 donde el primero contiene la parte derecha de la segunda condición (4.6) y el segundo la de la tercera condición (4.10).

A partir de la línea 58 se trata el segundo proceso de la optimización por mínimos cuadrados, consistente en minimizar el error cuadrático, por lo que se calculan las matrices y vectores necesarios. En primer lugar, se define una matriz “A” de ceros, que finalmente se mostrará de la forma (4.19), de dimensiones $297 \times 3N_1$, y ya entre las líneas 60 y 77 se desarrolla otro bucle ‘for’ con el que se van completando filas de tres en tres cada vez con un valor distinto de λ_c , denominado como “raizConocida”. El número de filas de dicha matriz se determina según el incremento de “raizConocida” escogido para el bucle ‘for’ principal, luego en el caso de que éste fuera más pequeño habría mayor número de filas, y menor si el incremento fuera más elevado.

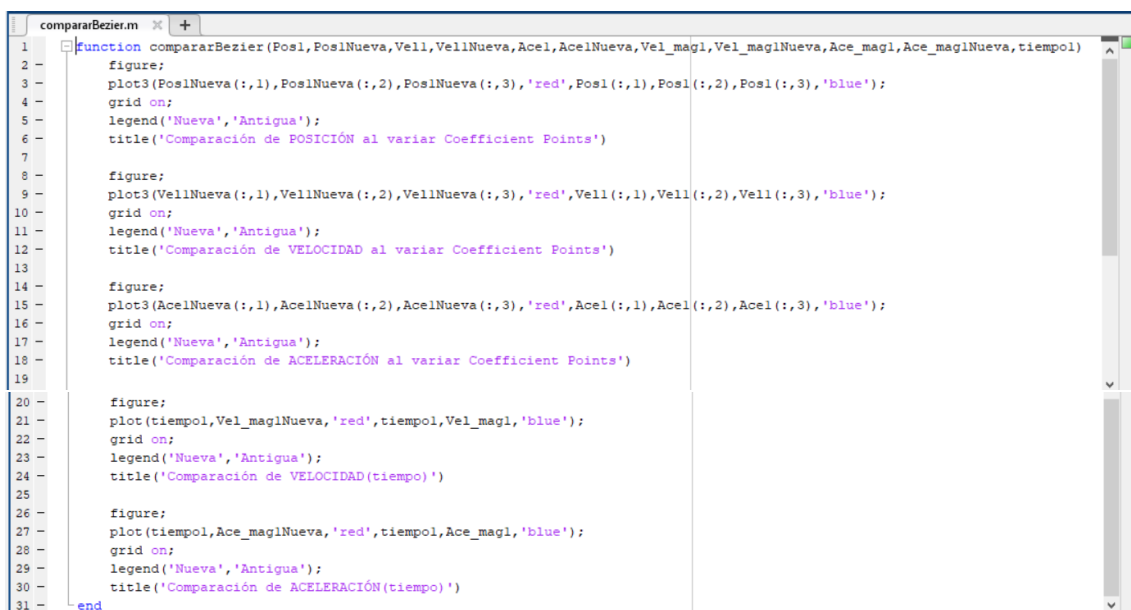
A continuación, entre las líneas 80 y 94 se implementa otro bucle ‘for’ exactamente igual al utilizado en la matriz “A” para obtener el vector columna “B” mostrado en (4.20). Por tanto, este

vector será de dimensiones 297 x 1, y de esta manera se puede implementar la ecuación (4.18) con la que minimizar el error cuadrático.

Después de esto se construye la matriz “M”, cuya forma es la mostrada en (4.22), y el vector columna “S” de la forma (4.24), para así resolver en la línea 98 el sistema matricial encargado de devolver el vector solución “X” y cuya apariencia es la mostrada en (4.21).

Finalmente, entre las líneas 100 y 110 se almacenan los valores deseados en la matriz “CoeP1Nueva” empezando por su segunda fila, puesto que la primera fila ya ha sido rellenada a comienzos de la función, y unos valores desechables, nombrados antes como multiplicadores de Lagrange, en el vector “Error”. Además, mediante el comando ‘fprintf’ se indica en *Command Window* que el cálculo de los nuevos parámetros ha sido exitoso.

4.3.2. Función “compararBezier”



```
1 function compararBezier(Pos1,Pos1Nueva,Vel1,Vel1Nueva,Acel1,Acel1Nueva,Vel_mag1,Vel_mag1Nueva,Ace_mag1,Ace_mag1Nueva,tiempol)
2     figure;
3     plot3(Pos1Nueva(:,1),Pos1Nueva(:,2),Pos1Nueva(:,3),'red',Pos1(:,1),Pos1(:,2),Pos1(:,3),'blue');
4     grid on;
5     legend('Nueva','Antigua');
6     title('Comparación de POSICIÓN al variar Coefficient Points')
7
8     figure;
9     plot3(Vel1Nueva(:,1),Vel1Nueva(:,2),Vel1Nueva(:,3),'red',Vel1(:,1),Vel1(:,2),Vel1(:,3),'blue');
10    grid on;
11    legend('Nueva','Antigua');
12    title('Comparación de VELOCIDAD al variar Coefficient Points')
13
14    figure;
15    plot3(Acel1Nueva(:,1),Acel1Nueva(:,2),Acel1Nueva(:,3),'red',Acel1(:,1),Acel1(:,2),Acel1(:,3),'blue');
16    grid on;
17    legend('Nueva','Antigua');
18    title('Comparación de ACELERACIÓN al variar Coefficient Points')
19
20    figure;
21    plot(tiempol,Vel_mag1Nueva,'red',tiempol,Vel_mag1,'blue');
22    grid on;
23    legend('Nueva','Antigua');
24    title('Comparación de VELOCIDAD(tiempo)')
25
26    figure;
27    plot(tiempol,Ace_mag1Nueva,'red',tiempol,Ace_mag1,'blue');
28    grid on;
29    legend('Nueva','Antigua');
30    title('Comparación de ACELERACIÓN(tiempo)')
31    end
```

Figura 4.2: Función “compararBezier”

Después de haber obtenido los Coefficient Points de la nueva Bézier “CoeP1Nueva”, es necesario realizar comparaciones gráficas con las que alcanzar una imagen visual de que el trabajo realizado ha sido correcto.

Para ello, se hace uso de la función “compararBezier”, que debe recibir como parámetros de entrada las matrices de posición “Pos1” y “Pos1Nueva”, de velocidad “Vel1” y “Vel1Nueva” y de aceleración “Acel1” y “Acel1Nueva”, así como los vectores de magnitud de velocidad “Vel_mag1” y “Vel_mag1Nueva”, de aceleración “Ace_mag1” y “Ace_mag1Nueva” y de tiempo “tiempol” de la antigua y la nueva curva de Bézier.

Respecto al uso de comandos, son exactamente los mismos que los mostrados en la función “mostrarPVA” a excepción de ‘legend’, cuya función es la de añadir en el extremo derecho de la gráfica un cuadro de texto que indique qué es cada función según el color, es decir, una leyenda.

De esta manera, entre las líneas 2 y 30 se producen cinco superposiciones de gráficas en dos y tres dimensiones para comparar el cambio de una a otra curva, mostrándose la original de color azul y la nueva de color rojo.

4.3.3. Función “compararEsferas”

```

1  function compararEsferas(N1, CoeP1, CoeP1Nueva, CoeP2, raizMin, Pos1, Pos1Nueva, Pos2, N2, R1, R2)
2  for i=0:1:N1
3      if(i==0)
4          PosMin1=CoeP1(i+1, :);
5          PosMin1Nueva=CoeP1Nueva(i+1, :);
6      else
7          PosMin1=PosMin1+(raizMin^(i))*CoeP1(i+1, :);
8          PosMin1Nueva=PosMin1Nueva+(raizMin^(i))*CoeP1Nueva(i+1, :);
9      end
10 end
11
12 for i=0:1:N2
13     if(i==0)
14         PosMin2=CoeP2(i+1, :);
15     else
16         PosMin2=PosMin2+(raizMin^(i))*CoeP2(i+1, :);
17     end
18 end
19
20 figure;
21 plot3(Pos1(:,1), Pos1(:,2), Pos1(:,3), 'blue', Pos2(:,1), Pos2(:,2), Pos2(:,3), 'green', Pos1Nueva(:,1), Pos1Nueva(:,2), Pos1Nueva(:,3), 'red');
22 grid on;
23 hold on;
24 [x y z] = sphere;
25 a=[PosMin1 R1; PosMin2 R2; PosMin1Nueva R1];
26 s1=surf(x*a(1,4)+a(1,1), y*a(1,4)+a(1,2), z*a(1,4)+a(1,3));
27 hold on
28 s2=surf(x*a(2,4)+a(2,1), y*a(2,4)+a(2,2), z*a(2,4)+a(2,3));
29 hold on
30 s3=surf(x*a(3,4)+a(3,1), y*a(3,4)+a(3,2), z*a(3,4)+a(3,3));
31 daspect([1 1 1])
32 legend('B1', 'B2', 'BN1', 'Esf1', 'Esf2', 'EsfN1');
33 title('Bézier 1 - Bézier 2 - Bézier 1 NUEVA')
34
35 figure;
36 plot3(Pos1(:,1), Pos1(:,2), Pos1(:,3), 'blue', Pos2(:,1), Pos2(:,2), Pos2(:,3), 'green');
37 grid on;
38 hold on;
39 [x y z] = sphere;
40 a=[PosMin1 R1; PosMin2 R2; PosMin1Nueva R1];
41 s1=surf(x*a(1,4)+a(1,1), y*a(1,4)+a(1,2), z*a(1,4)+a(1,3));
42 hold on
43 s2=surf(x*a(2,4)+a(2,1), y*a(2,4)+a(2,2), z*a(2,4)+a(2,3));
44 daspect([1 1 1])
45 legend('B1', 'B2', 'Esf1', 'Esf2');
46 title('Bézier 1 - Bézier 2')
47
48 figure;
49 plot3(Pos1Nueva(:,1), Pos1Nueva(:,2), Pos1Nueva(:,3), 'red', Pos2(:,1), Pos2(:,2), Pos2(:,3), 'green');
50 grid on;
51 hold on;
52 [x y z] = sphere;
53 a=[PosMin1 R1; PosMin2 R2; PosMin1Nueva R1];
54 s3=surf(x*a(3,4)+a(3,1), y*a(3,4)+a(3,2), z*a(3,4)+a(3,3));
55 hold on
56 s2=surf(x*a(2,4)+a(2,1), y*a(2,4)+a(2,2), z*a(2,4)+a(2,3));
57 daspect([1 1 1])
58 legend('BN1', 'B2', 'EsfN1', 'Esf2');
59 title('Bézier 1 NUEVA - Bézier 2')
60 end

```

Figura 4.3: Función “compararEsferas”

Pese a que es interesante apreciar las variaciones de una curva a otra tras haber calculado una nueva matriz de Coefficient Points, tal y como se hace en la función “compararBezier”, aquello que verdaderamente es de importancia en este trabajo es observar como varía la posición de la esfera en el instante de mínima distancia con la otra, pudiendo verse si se consigue evitar la colisión. Para ello, se hará uso de la función “compararEsferas”.

La función “compararEsferas” recibe como parámetros de entrada:

- Orden de cada curva, “N1” y “N2”.
- Coefficient Points “CoeP1”, “CoeP1Nueva” y “CoeP2”.
- El parámetro λ_{min} “raizMin”.
- Matrices de posición “Pos1”, “Pos1Nueva” y “Pos2”.
- Radio de cada esfera, “R1” y “R2”.

Y no devuelve ningún parámetro de salida, puesto que sólo se requiere obtener representaciones gráficas.

En referencia al uso de comandos, existen nuevas aportaciones respecto a las funciones “mostrarPVA” y “compararBezier”. Entre éstas se encuentran el comando ‘sphere’, que genera las coordenadas X, Y y Z de una esfera unitaria o de radio igual a uno [12], el comando ‘surf (X, Y, Z)’, que genera una superficie tridimensional [13], y ‘daspect ([1 1 1])’, que cambia la escala de cada eje a 1 [12].

Ya con el foco de atención puesto en el código de esta función, entre las líneas 2 y 18 se calcula, mediante dos bucles ‘for’, la posición exacta para cada curva en la que la distancia entre esferas sería mínima, es decir, es aquí donde se utiliza el parámetro “raizMin”. Estos vectores “PosMin1”, “PosMin1Nueva” y “PosMin2” de dimensión 1 x 3 se utilizan posteriormente para centrar cada esfera en su respectiva curva.

Posteriormente, entre las líneas 20 y 33 se generan, en primer lugar, las dos curvas originales y la nueva mediante el comando ‘plot3’. Después se aplica el fragmento ‘[x y z] = sphere’ para asociar los parámetros ‘x’, ‘y’ y ‘z’ con las coordenadas de una esfera.

A continuación se genera una matriz “a” de dimensiones 3 x 4 donde las tres primeras columnas de cada fila se asocian con las coordenadas X, Y y Z de las matrices “PosMin” que indican la posición exacta del centro de la esfera para el instante de mínima distancia, y la última columna contiene el radio de cada esfera.

Por último, se generan las esferas usando el comando ‘surf’ que diferencia las tres coordenadas, viéndose la aplicación de una ecuación de la forma ‘ $x \cdot Radio + X_{Esfera}$ ’ para cada una de ellas. Como puede apreciarse, “s1” se relaciona con la esfera de la curva 1 original, “s2” con la de la curva 2 original y “s3” con la de la curva 1 modificada.

Todo este fragmento de código se reproduce de forma análoga entre las líneas 35 y 59 para la creación de otras dos gráficas que representan esferas.

Capítulo 5

Ejemplos

Tras haber sido explicadas paso a paso todas las funciones necesarias para la obtención de una nueva curva de Bézier, es el momento de ejemplificar todo esto mediante representaciones gráficas.

No obstante, antes sería necesario recordar cuál es el orden a seguir a la hora de ejecutar cada una de las funciones, y esto es lo que hace el *script* “herramientaBezier” mostrado en la Figura 5.1.

```
1 %1° PASO: Introducir en Command Window los parámetros de entrada "ConP1", "ConP2", "N1",
2 %"N2", "R1", "R2", "Tini1", "Tini2", "Tfin1", "Tfin2".
3
4 %2° PASO: Obtener los Coefficient Points de las Bézier originales.
5 - [CoeP1,CoeP2,CoeV1,CoeV2,CoeA1,CoeA2]=obtenerCoePointsPVA(N1,N2,ConP1,ConP2)
6
7 %3° PASO: Calcular las matrices de posición, velocidad y aceleración así
8 %como las magnitudes de velocidad y aceleración en función del tiempo.
9 - [Pos1,Vel1,Acel,Vel_mag1,Ace_mag1,tiempo1]=obtenerPVA(CoeP1,CoeV1,CoeA1,N1,Tini1,Tfin1)
10 - [Pos2,Vel2,Ace2,Vel_mag2,Ace_mag2,tiempo2]=obtenerPVA(CoeP2,CoeV2,CoeA2,N2,Tini2,Tfin2)
11
12 %4° PASO: Representar la trayectoria, la velocidad y la aceleración de las
13 %Bézier originales.
14 - mostrarPVA(ConP1,tiempo1,Pos1,Vel1,Acel,Vel_mag1,Ace_mag1)
15 - mostrarPVA(ConP2,tiempo2,Pos2,Vel2,Ace2,Vel_mag2,Ace_mag2)
16
17 %5° PASO: Obtener la raíz para la que la distancia entre curvas es mínima.
18 - [raicesRealesPol,Distancias,distMin,distRadios,raizMin,mMin]=obtenerCoeDif(N1,N2,CoeP1,CoeP2,R1,R2)
19
20 %6° PASO: Calcular los nuevos Coefficient Points para obtener la nueva
21 %trayectoria.
22 - [CoeP1Nueva,Error]=obtenerNuevaBezier(distRadios,distMin,mMin,raizMin,CoeP1,N1)
23
24 %7° PASO: Obtener los Coefficient Points de velocidad y aceleración.
25 - [CoeV1Nueva]=obtenerCoeV(N1,CoeP1Nueva)
26 - [CoeA1Nueva]=obtenerCoeA(N1,CoeV1Nueva)
27
28 %8° PASO: Calcular las nuevas matrices de posición, velocidad y
29 %aceleración.
30 - [Pos1Nueva,Vel1Nueva,AcelNueva,Vel_mag1Nueva,Ace_mag1Nueva,tiempo1]=obtenerPVA(CoeP1Nueva,CoeV1Nueva,CoeA1Nueva,N1,Tini1,Tfin1)
31
```

```
32 %9º PASO: Realizar comparaciones entre la antigua y la nueva CURVA BÉZIER.  
33 - compararBezier(Pos1,Pos1Nueva,Vel1,Vel1Nueva,Acel,AcelNueva,Vel_mag1,Vel_mag1Nueva,Ace_mag1,Ace_mag1Nueva,tiempol)  
34  
35 %10º PASO: Realizar comparaciones entre la POSICIÓN de las ESFERAS.  
36 - compararEsferas(N1,CoeP1,CoeP1Nueva,CoeP2,raizMin,Pos1,Pos1Nueva,Pos2,N2,R1,R2)  
37
```

Figura 5.1: Script “herramientaBezier”

Por tanto, éste será el script utilizado en cada uno de los ejemplos que se mostrarán a continuación. Antes de comenzar, es preciso advertir de que, a pesar de que las funciones contemplan la posibilidad de que no exista colisión, se han escogido ejemplos en los que ésta se produzca para poder contemplar la potencia de esta herramienta.

5.1. Ejemplo 1: curvas del mismo orden

El primer paso a realizar es introducir todos los parámetros necesarios para empezar a calcular nuevas variables. Para este caso, se hace:

```
Command Window  
fx >> ConP1=[0 0 0; 1 -1 0; 3 3 -2; 5 -4 2; 6 4 8; 8 -5 2; 10 4 3; 11 5 14];  
ConP2=[0 8; 5; 1 5 -4; 3 -2 -3; 5 6 7; 6 -1 1; 8 4 -3; 10 0 -7; 11 -3 4];  
N1=7; N2=7; R1=2; R2=1.5; Tini1=0; Tfin1=60; Tini2=20; Tfin2=70;
```

Figura 5.2: Parámetros de entrada (ejemplo 1)

Quedando constancia de que no existe necesidad de que los objetos tengan el mismo tamaño ni que comiencen y finalicen su vuelo en el mismo instante.

Después de aplicar el segundo y tercer paso, ya es posible representar la posición, velocidad y aceleración de cada curva, quedando de la siguiente forma:

CURVA DE BÉZIER 1

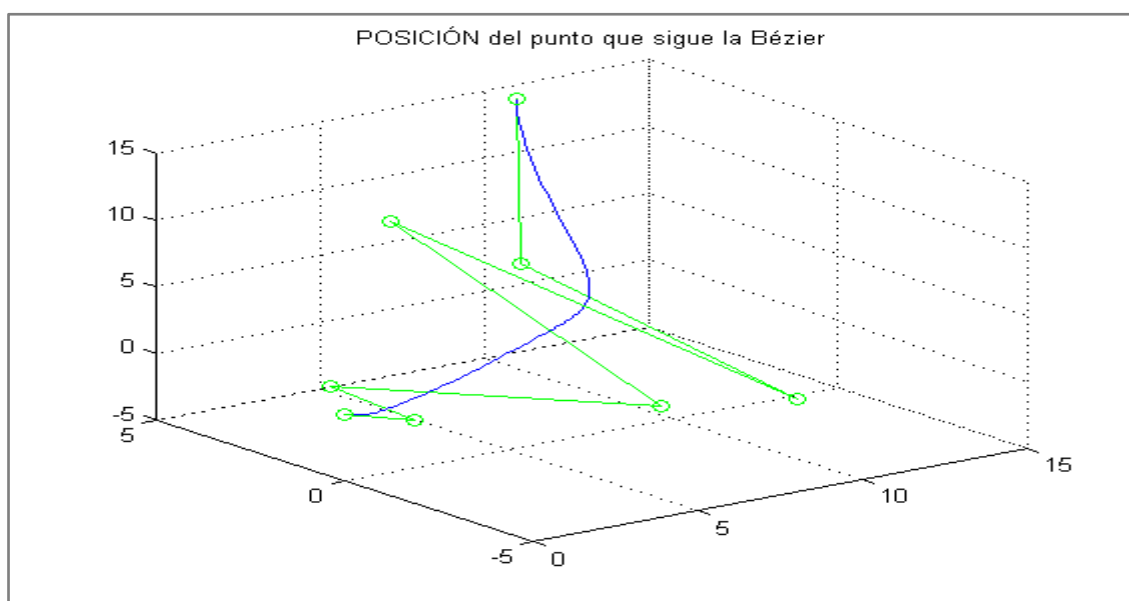


Figura 5.3: Trayectoria de la curva de Bézier 1 (ejemplo 1)

En la Figura 5.3 puede apreciarse cómo el inicio y el final de la trayectoria coinciden con los Control Points P_0 y P_N respectivamente.

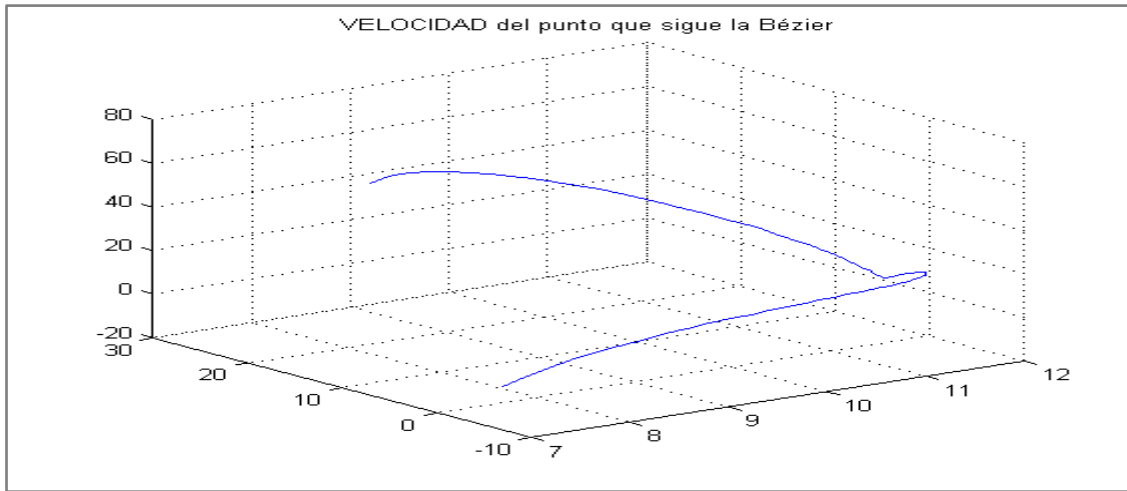


Figura 5.4: Velocidad de la curva de Bézier 1 (ejemplo 1)

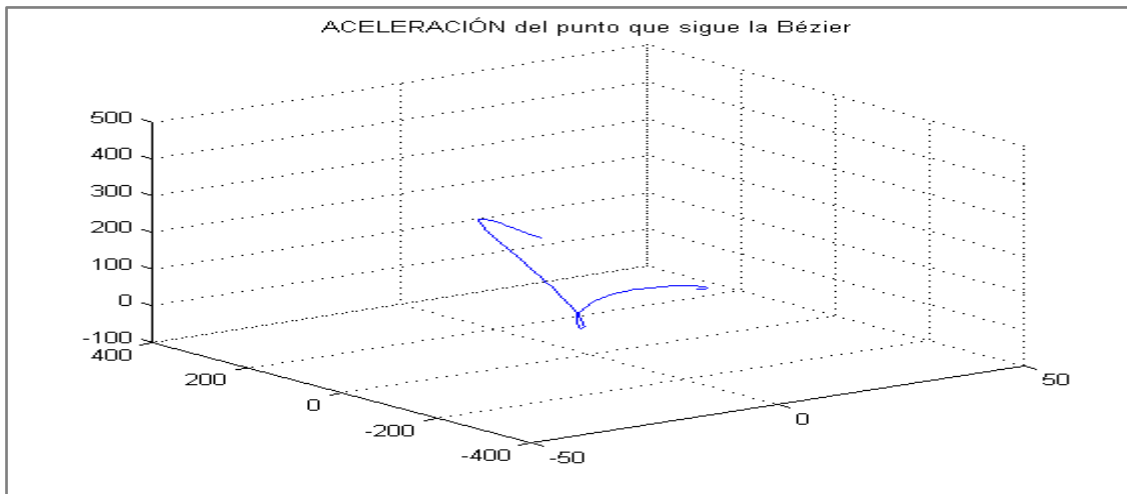


Figura 5.5: Aceleración de la curva de Bézier 1 (ejemplo 1)

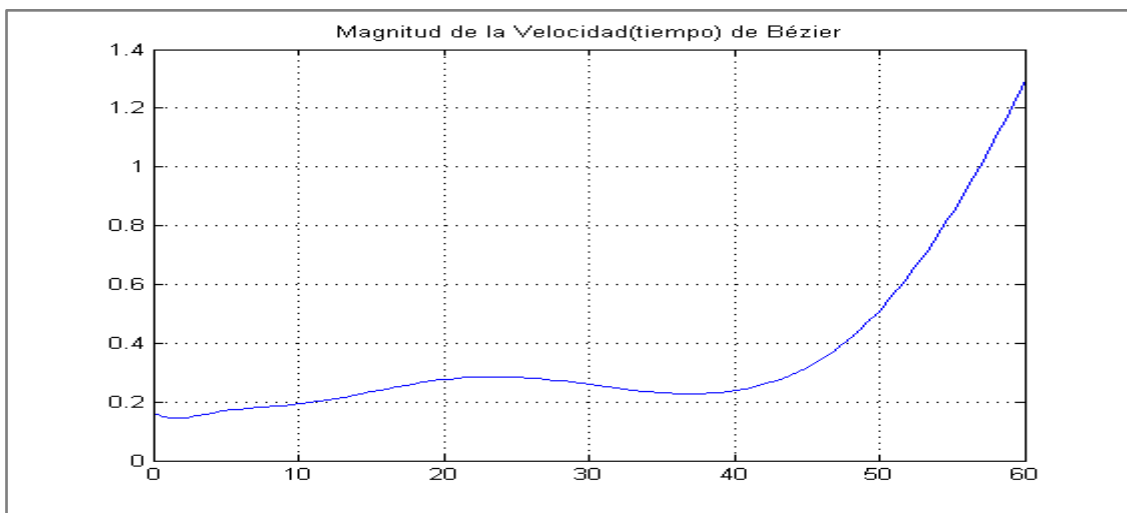


Figura 5.6: Magnitud de la velocidad en función del tiempo de la curva de Bézier 1 (ejemplo 1)

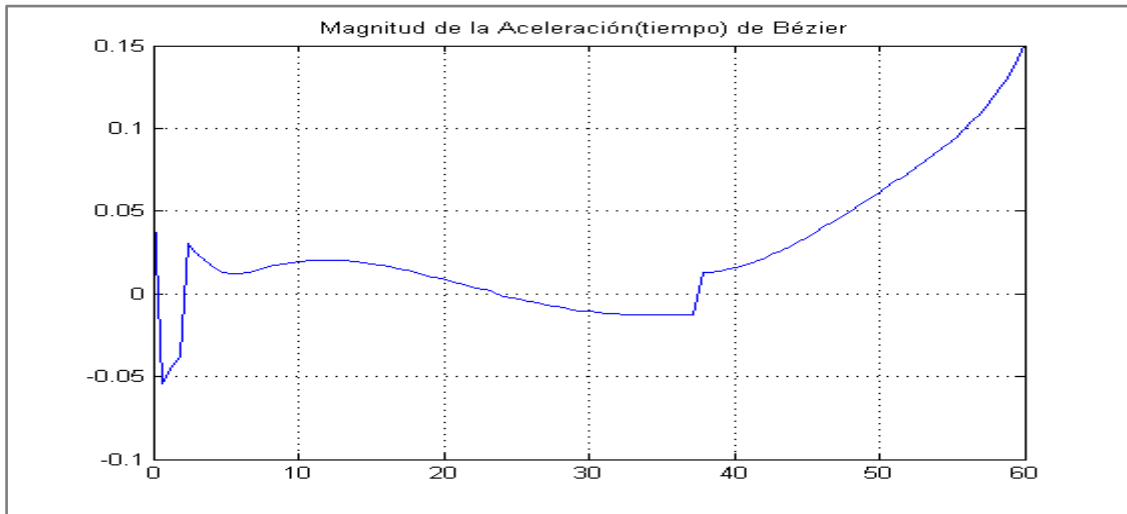


Figura 5.7: Magnitud de la aceleración en función del tiempo de la curva de Bézier 1 (ejemplo 1)

CURVA DE BÉZIER 2

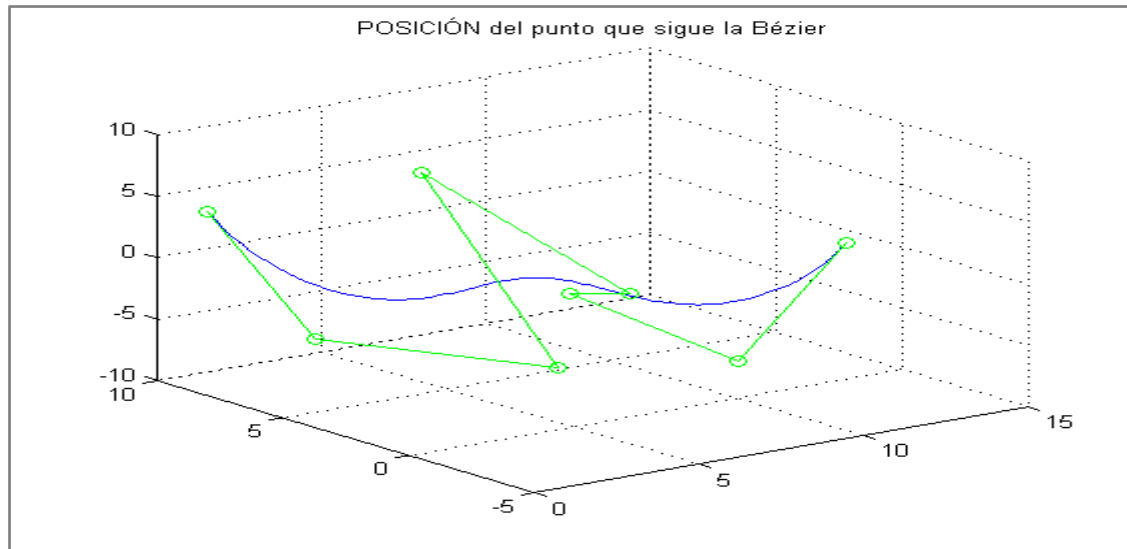


Figura 5.8: Trayectoria de la curva de Bézier 2 (ejemplo 1)

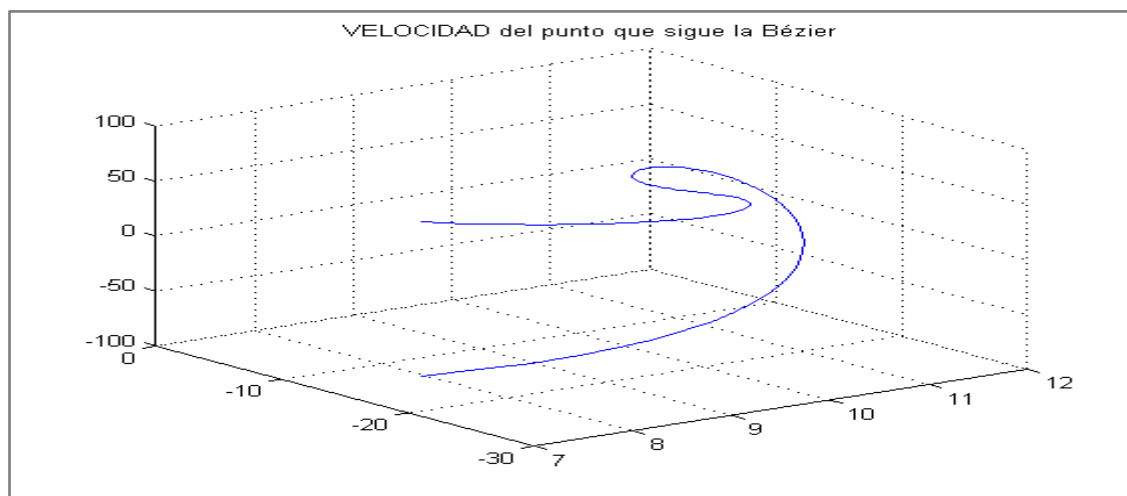


Figura 5.9: Velocidad de la curva de Bézier 2 (ejemplo 1)

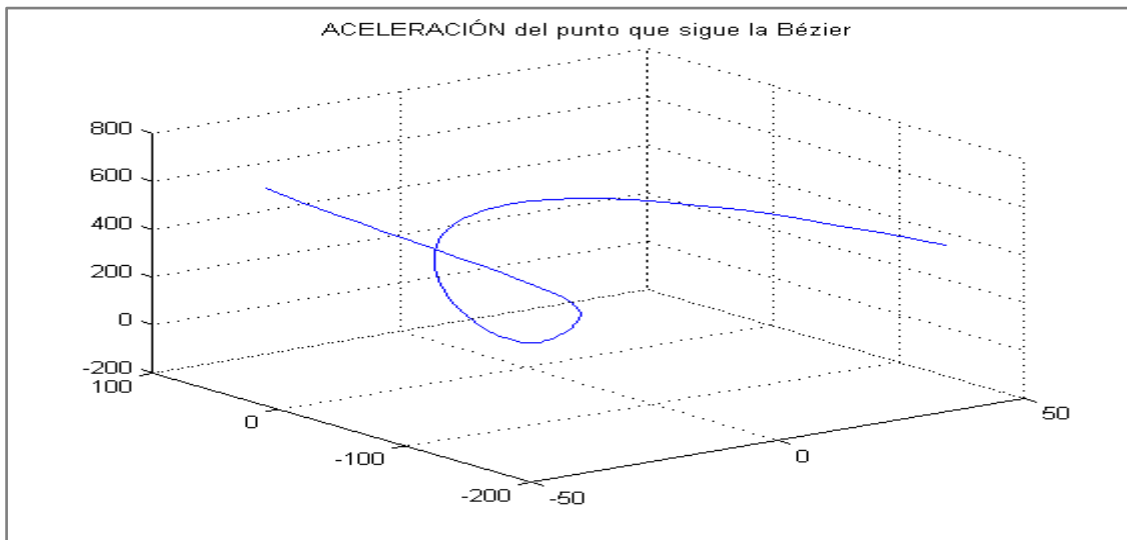


Figura 5.10: Aceleración de la curva de Bézier 2 (ejemplo 1)

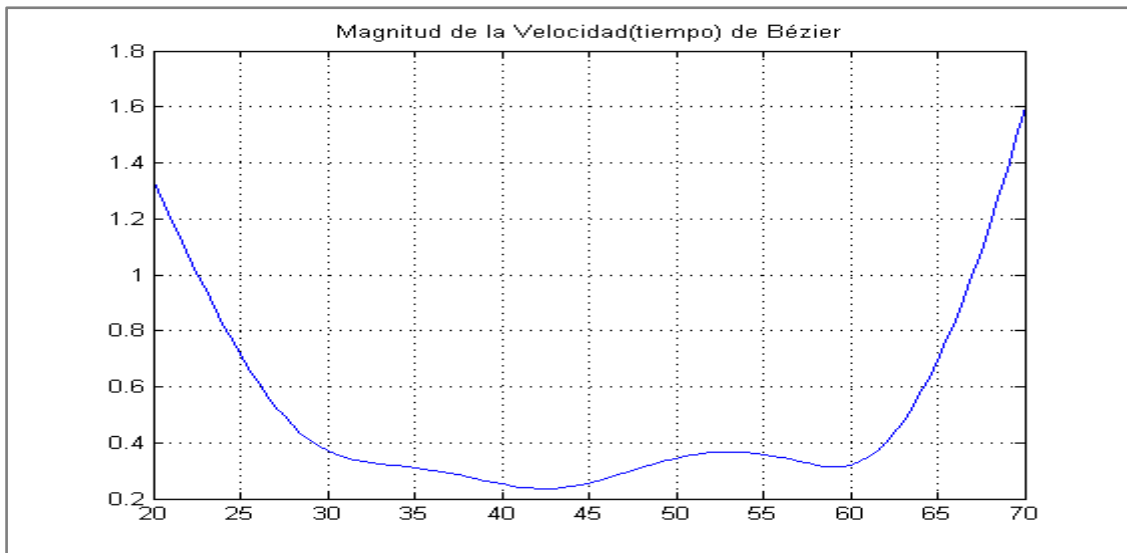


Figura 5.11: Magnitud de la velocidad en función del tiempo de la curva de Bézier 2 (ejemplo 1)

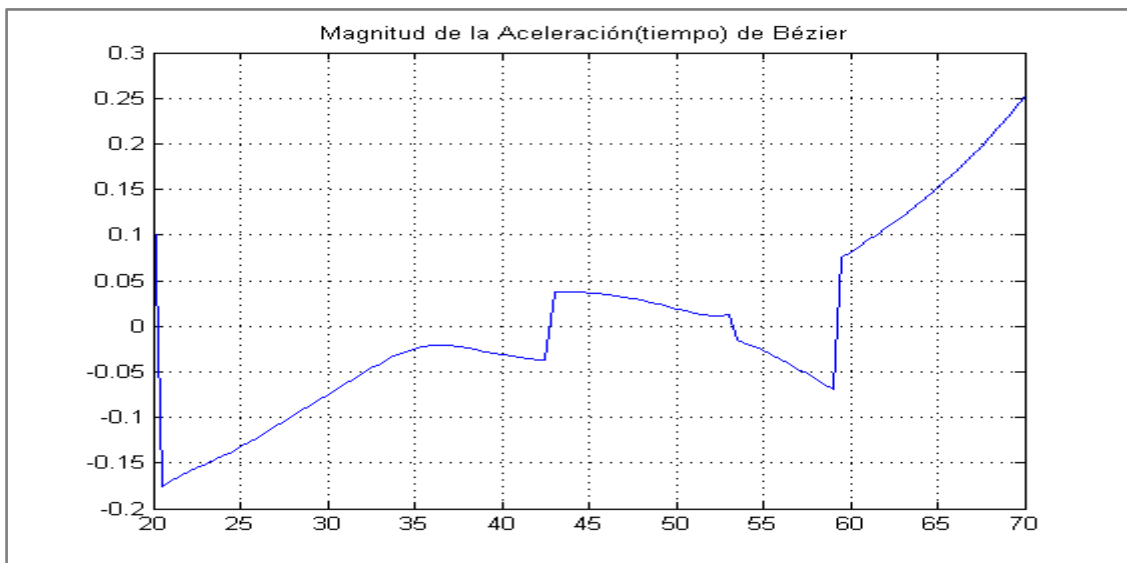


Figura 5.12: Magnitud de la aceleración en función del tiempo de la curva de Bézier 2 (ejemplo 1)

Todas estas representaciones gráficas no aportan una gran información extra más allá de ratificarse que las curvas se crean correctamente y que las gráficas de velocidad y de aceleración en función del tiempo tienen cierto sentido entre sí.

Sin embargo, tras haberse aplicado los pasos 5, 6, 7 y 8 indicados en “herramientaBezier”, ejecutando las funciones “compararBezier” y “compararEsferas” ya es posible obtener comparaciones útiles con las que asimilar realmente el trabajo realizado.

Antes de seguir, es necesario hacer un inciso: al aplicar “obtenerCoeDif” y “obtenerCoeDif_2” puede verse para este caso que el resultado de la distancia mínima en una función, “distMin”, y en otra, “distMin2” es casi el mismo, lo que da una idea de que el trabajo realizado es satisfactorio:



Name	Value	Min	Max
distMin	2.4683	2.4683	2.4683
distMin2	2.4687	2.4687	2.4687

Figura 5.13: Comparación entre “obtenerCoeDif” y “obtenerCoeDif_2” (ejemplo 1)

Así, aplicando el paso 9 se obtienen las siguientes representaciones gráficas:

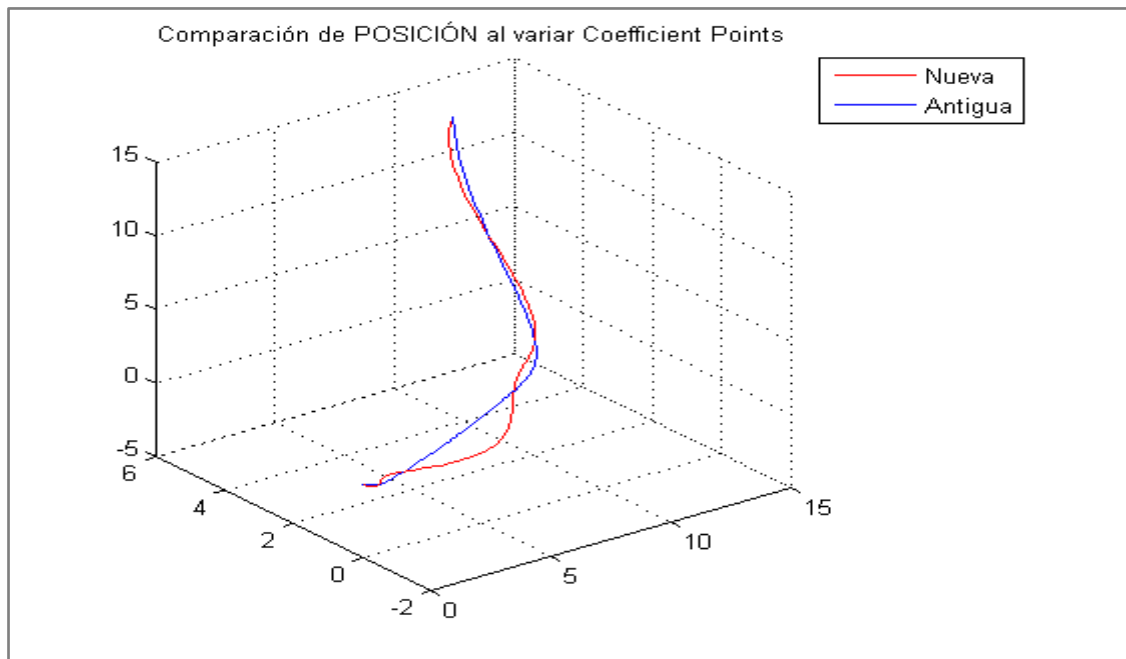


Figura 5.14: Comparación entre trayectorias de la antigua y la nueva Bézier 1 (ejemplo 1)

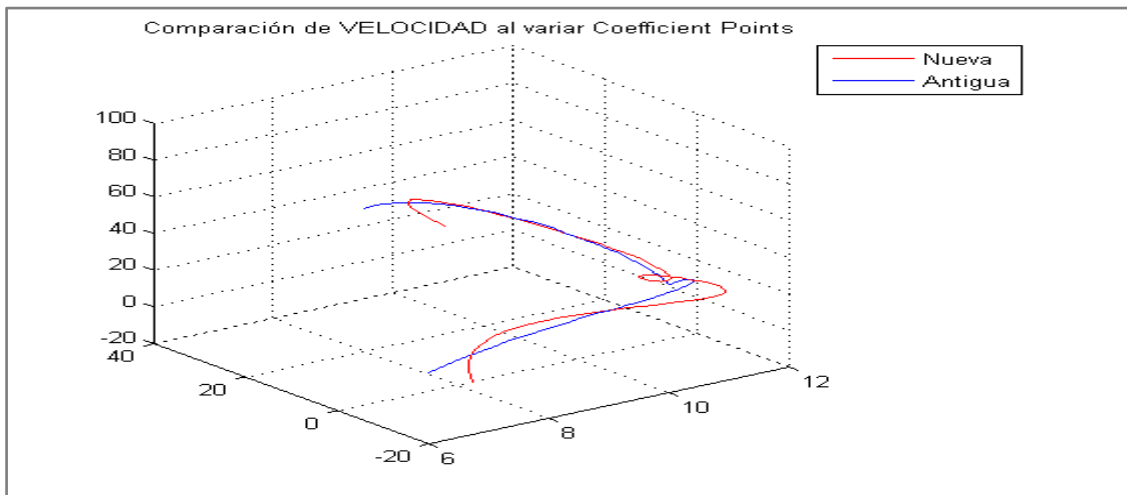


Figura 5.15: Comparación entre velocidades de la antigua y la nueva Bézier 1 (ejemplo 1)

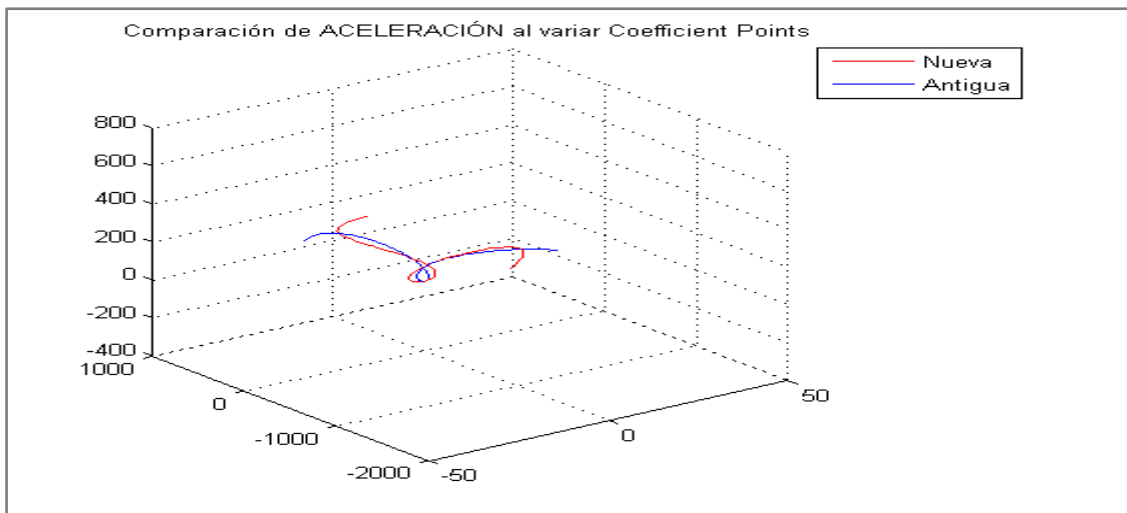


Figura 5.16: Comparación entre aceleraciones de la antigua y la nueva Bézier 1 (ejemplo 1)

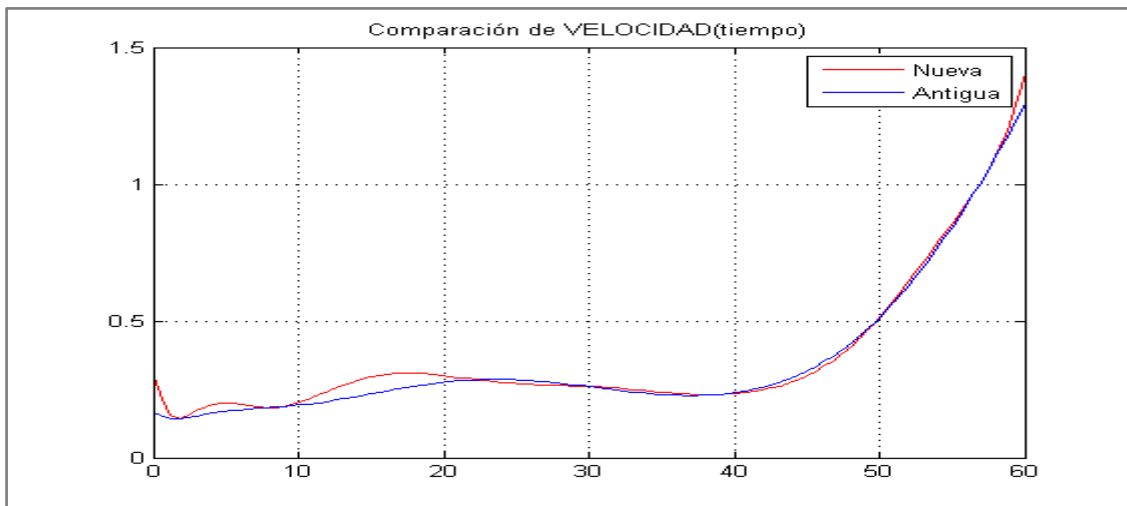


Figura 5.17: Comparación entre magnitudes de velocidad de la antigua y la nueva Bézier 1 (ejemplo 1)

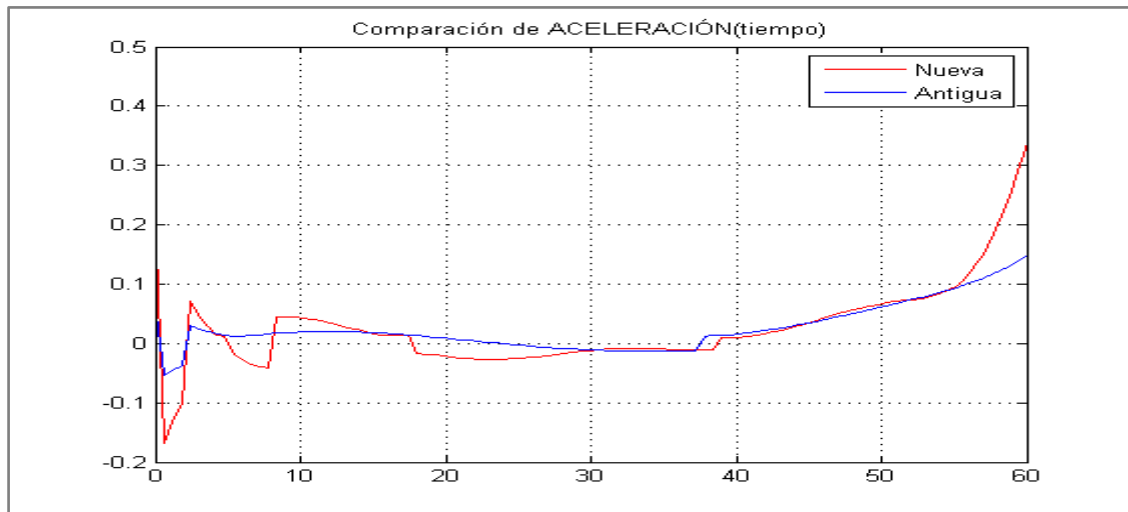


Figura 5.18: Comparación entre magnitudes de aceleración de la antigua y la nueva Bézier 1 (ejemplo 1)

Como puede apreciarse, se producen ligeros cambios en la trayectoria a seguir por el objeto volador de radio “R1”, los cuales traen como consecuencia algunas variaciones en la velocidad que éste debe adoptar y, por tanto, en la aceleración.

Por último, se aplica el paso 10 asociado a la función “compararEsferas”, con la que ya puede apreciarse un resultado final exitoso.

En las figuras 5.20 y 5.21 se representan las dos originales y la nueva curva de Bézier junto a sus respectivos objetos o esferas asociadas. No obstante, es complejo sacar alguna conclusión en claro mostrando todo en un espacio tan reducido, por lo que en las siguientes figuras se decide separar los dos posibles casos durante el vuelo.

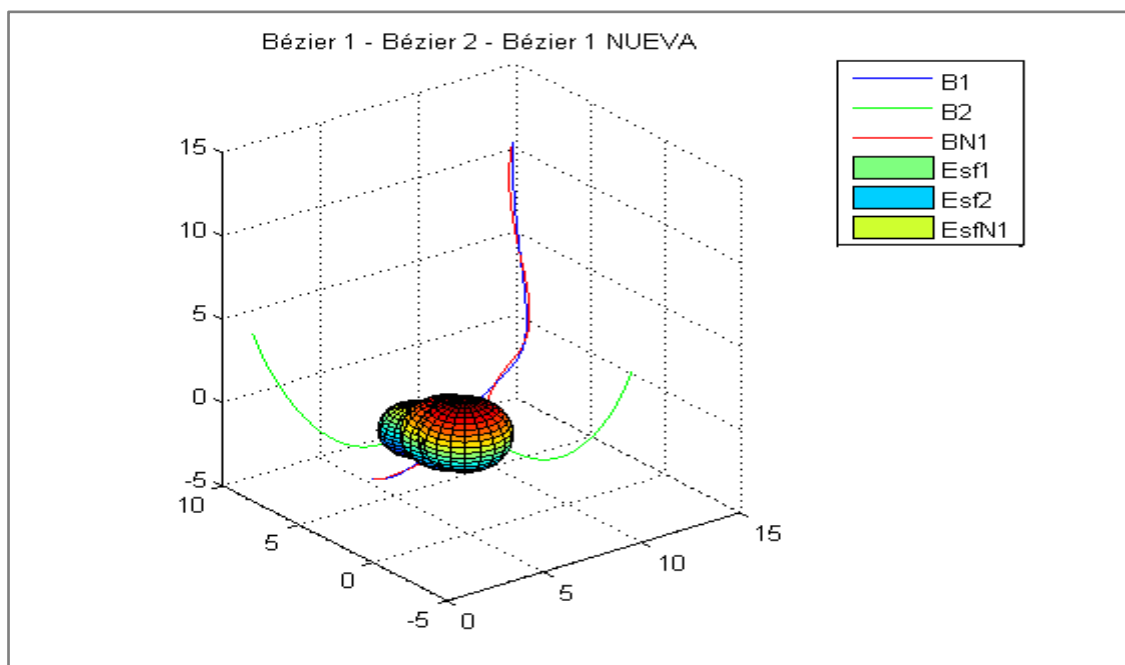


Figura 5.19: Esferas 1 y2 originales y 1 modificando su posición (ejemplo 1)

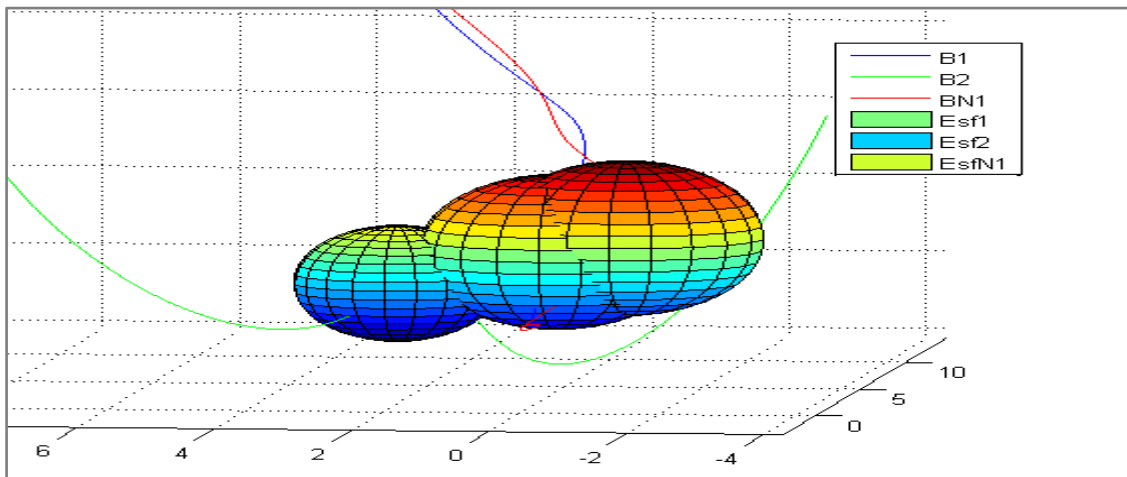


Figura 5.20: Zoom a la Figura 5.20 (ejemplo 1)

CASO 1: CURVAS 1 Y 2 ORIGINALES (CON COLISIÓN)

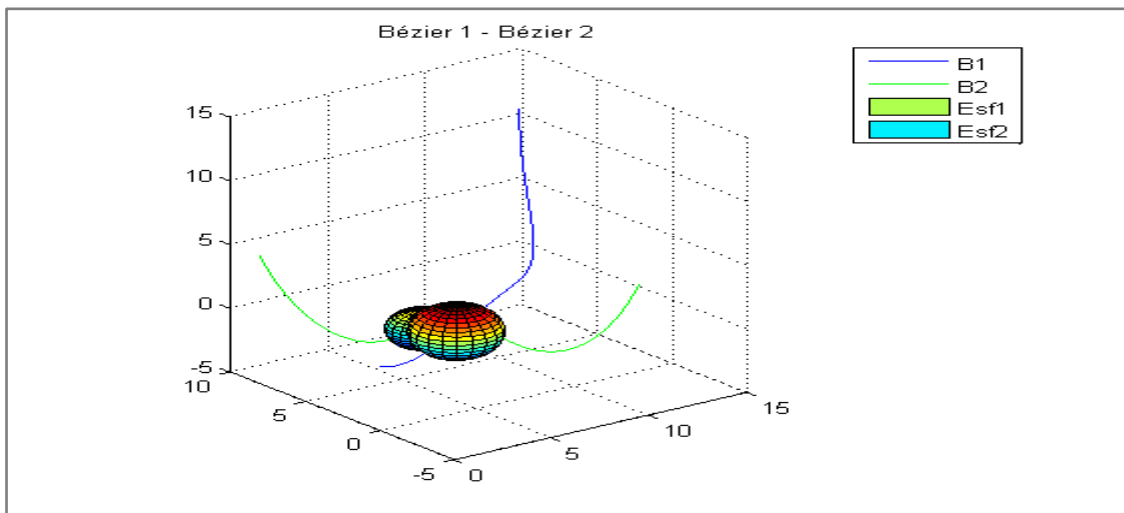


Figura 5.21: Esferas 1 y 2 originales (ejemplo 1)

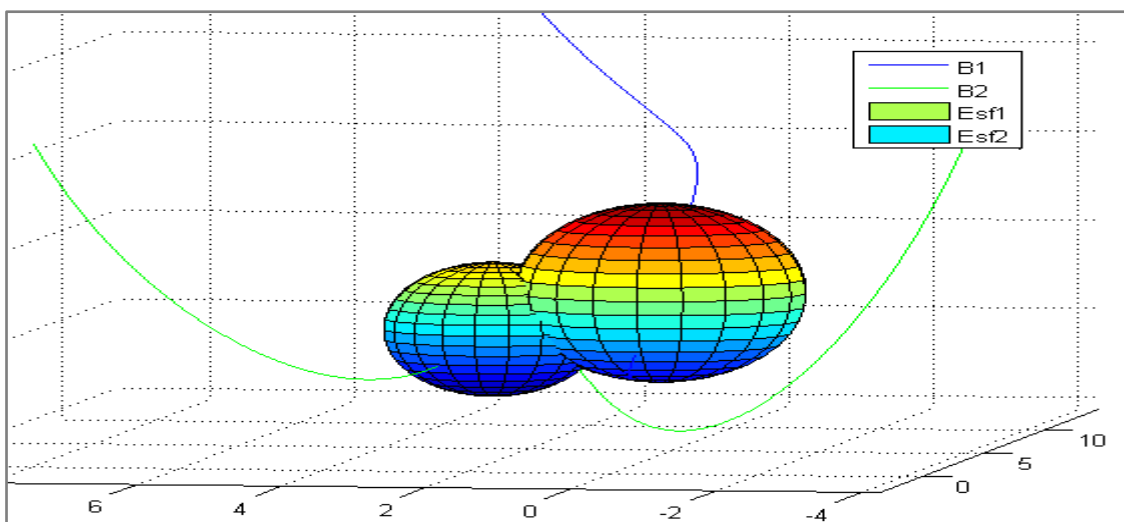


Figura 5.22: Zoom 1 a la Figura 5.22 (ejemplo 1)

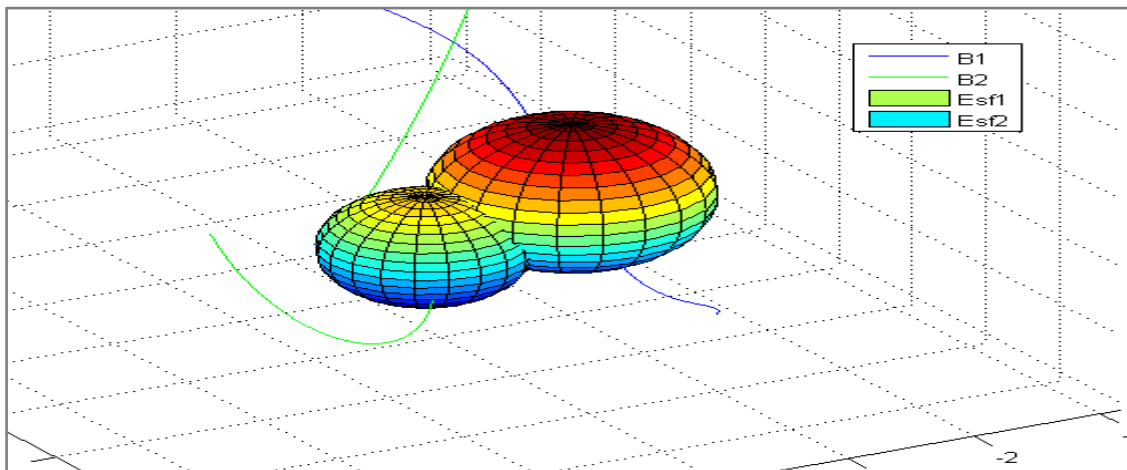


Figura 5.23: Zoom 2 a la Figura 5.22 (ejemplo 1)

Puede apreciarse con facilidad que las esferas con sus trayectorias iniciales colisionan o intersectan entre sí.

CASO 2: CURVA 1 MODIFICADA Y 2 ORIGINAL (SIN COLISIÓN)

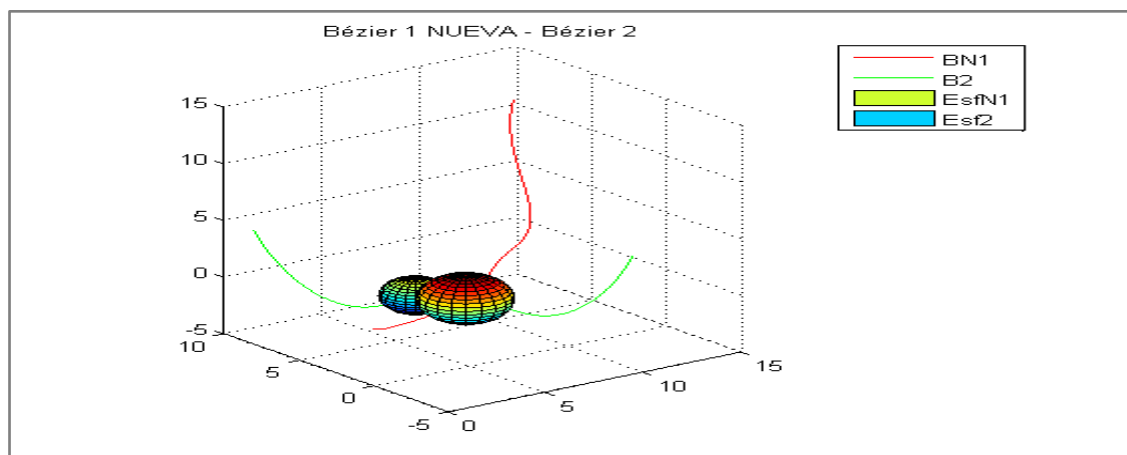


Figura 5.24: Esfera 1 modificando su posición y 2 original (ejemplo 1)

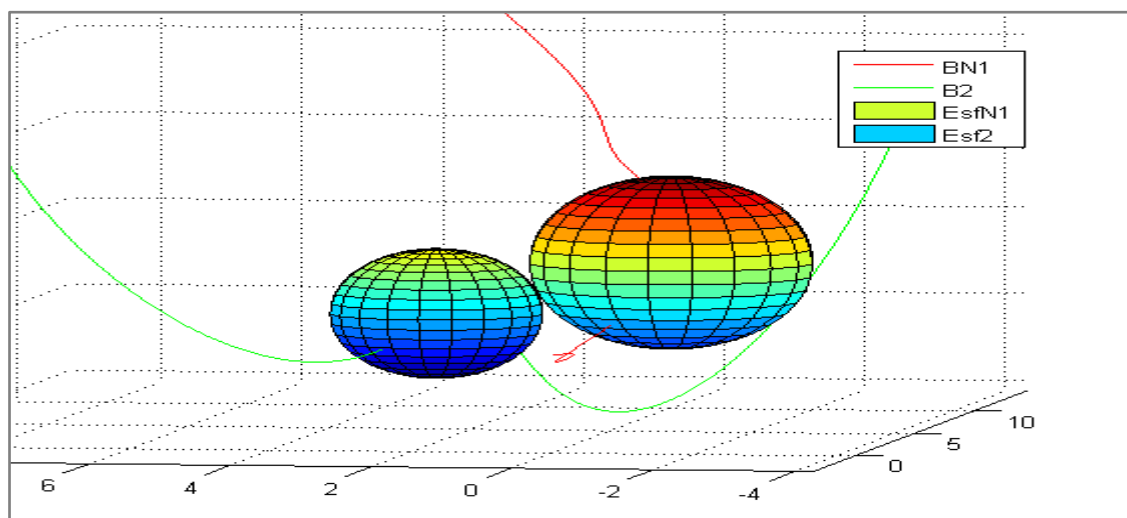


Figura 5.25: Zoom 1 a la Figura 5.25 (ejemplo 1)

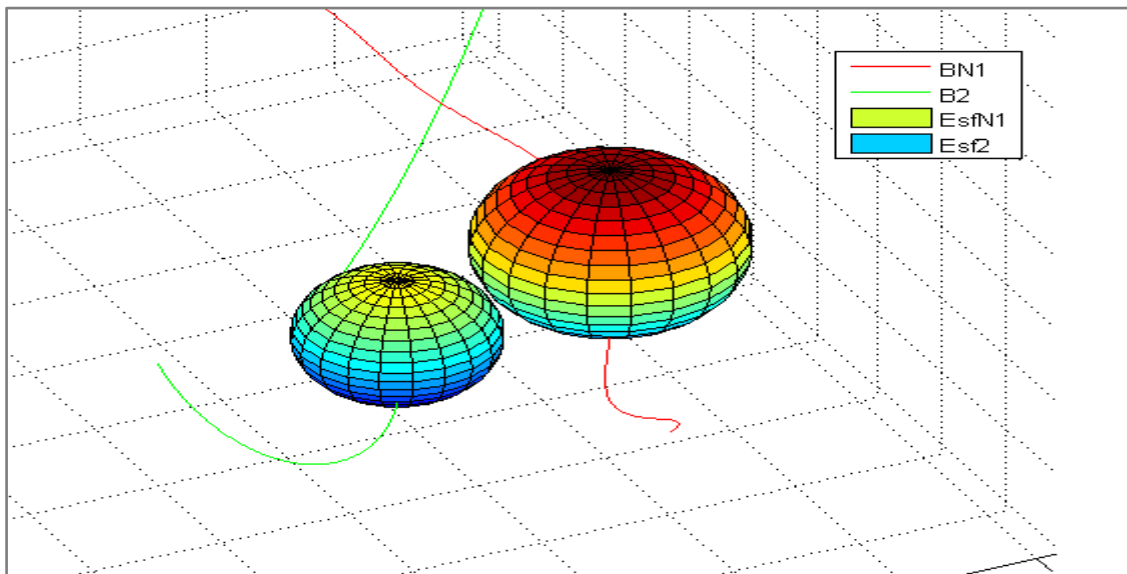


Figura 5.26: Zoom 2 a la Figura 5.25 (ejemplo 1)

A diferencia del primer caso, en el segundo sí que consigue evitarse la colisión entre esferas gracias a la optimización por mínimos cuadrados, apreciándose que el espacio entre una y otra es el mínimo necesario para ello. Por tanto, la implementación de la herramienta en este caso ha sido completamente exitosa.

5.2. Ejemplo 2: Curvas de diferente orden

Ya se ha comprobado en el ejemplo anterior que esta implementación funciona bastante bien con curvas definidas por un número elevado de Coefficient Points y con ecuaciones de orden N coincidente. Sin embargo, esta herramienta también debe obtener soluciones cuando el orden de una curva es reducido e inferior al de la otra curva. Es por ello que en este ejemplo se ha decidido demostrar el caso introducido.

De igual manera que en el ejemplo anterior, se introducen los parámetros de entrada necesarios:

```
Command Window
fx >> ConP1=[ 1 7 -4; 5 -5 8; 8 12 4; 13 -7 1];
ConP2=[0 -4 6; 1 6 -10; 3 -4 9; 5 0 -2; 6 -5 -4; 8 5 6; 10 10 -1; 11 2 -4];
N1=3; N2=7; R1=3; R2=2; Tini1=0; Tfin1=80; Tini2=10; Tfin2=75;
```

Figura 5.27: Parámetros de entrada (ejemplo 2)

Obteniéndose así las respectivas gráficas de posición, velocidad y aceleración de cada curva de Bézier original:

CURVA DE BÉZIER 1

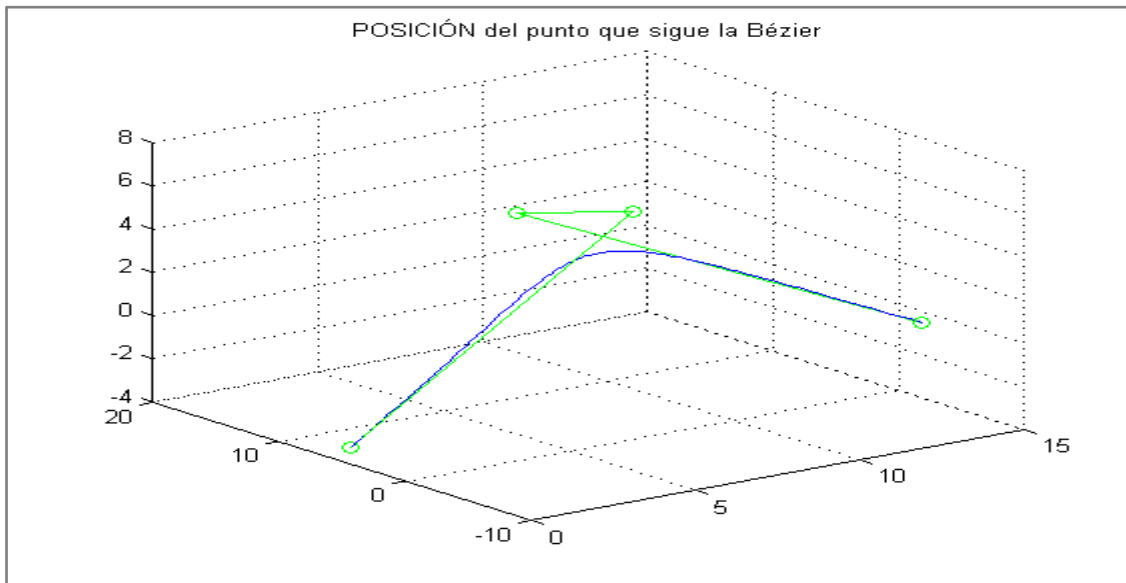


Figura 5.28: Trayectoria de la curva de Bézier 1 (ejemplo 2)

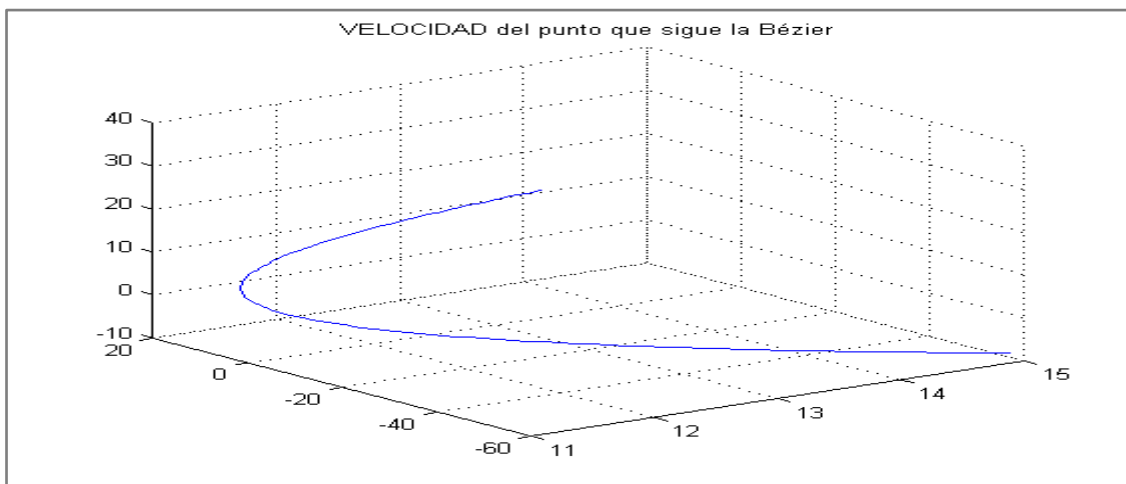


Figura 5.29: Velocidad de la curva de Bézier 1 (ejemplo 2)

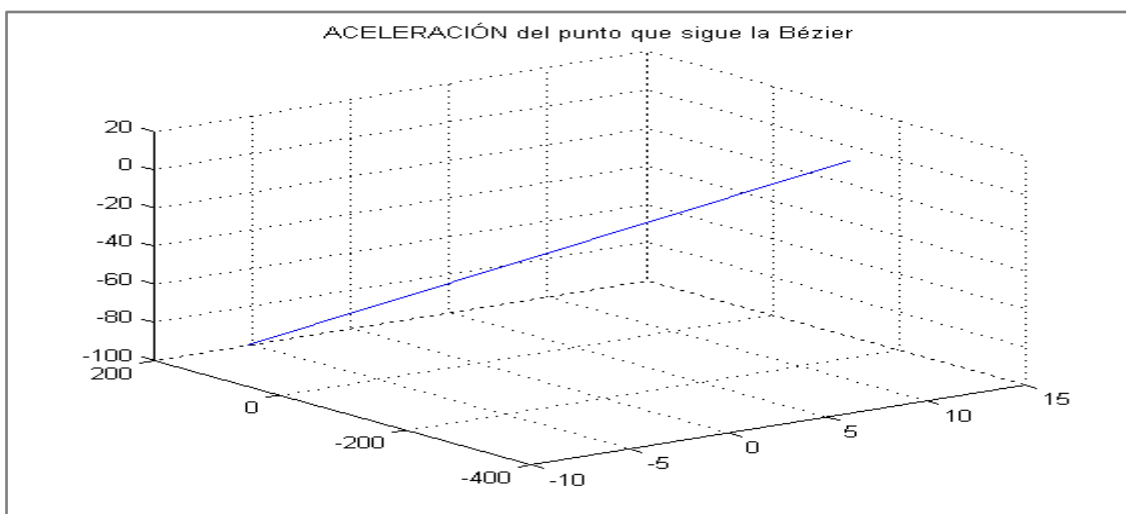


Figura 5.30: Aceleración de la curva de Bézier 1 (ejemplo 2)

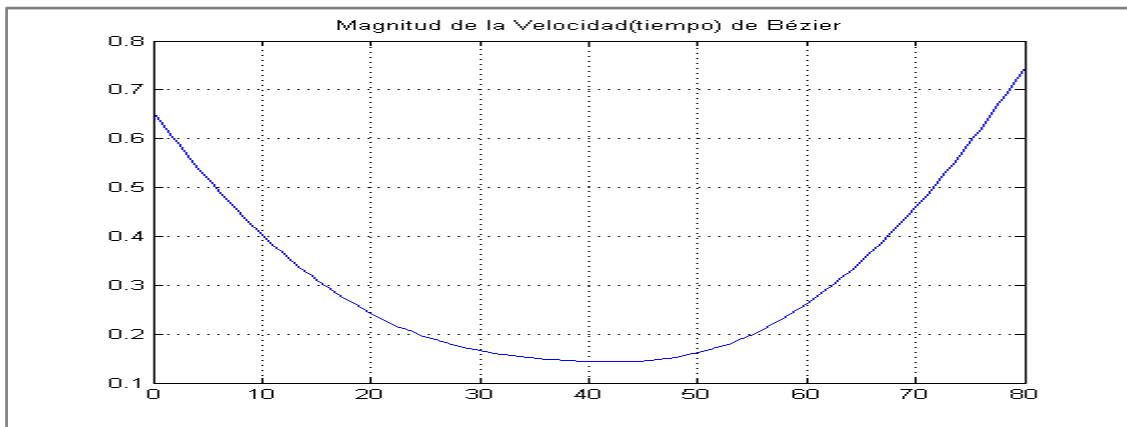


Figura 5.31: Magnitud de la velocidad en función del tiempo de la curva de Bézier 1 (ejemplo 2)

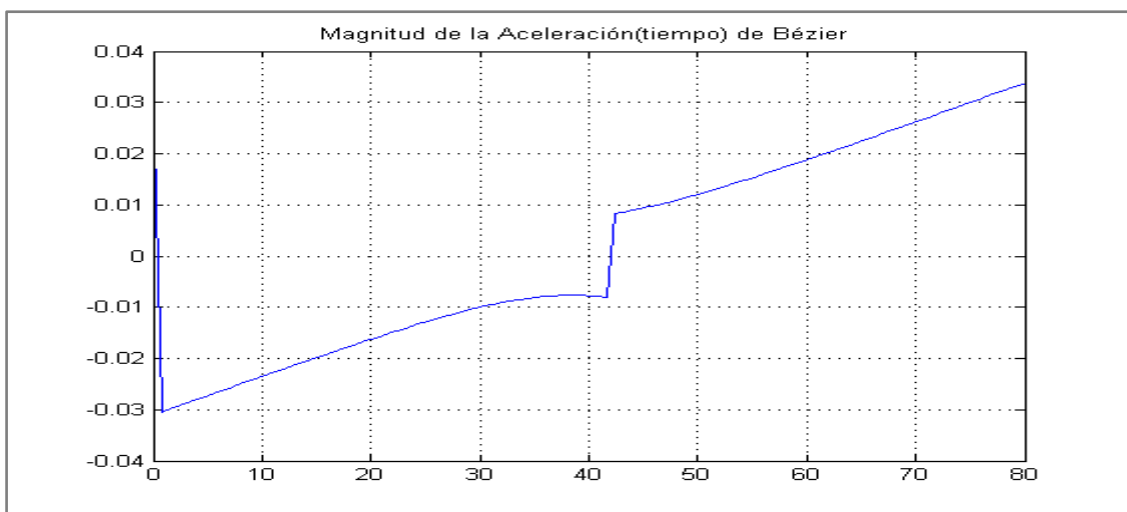


Figura 5.32: Magnitud de la aceleración en función del tiempo de la curva de Bézier 1 (ejemplo 1)

CURVA DE BÉZIER 2

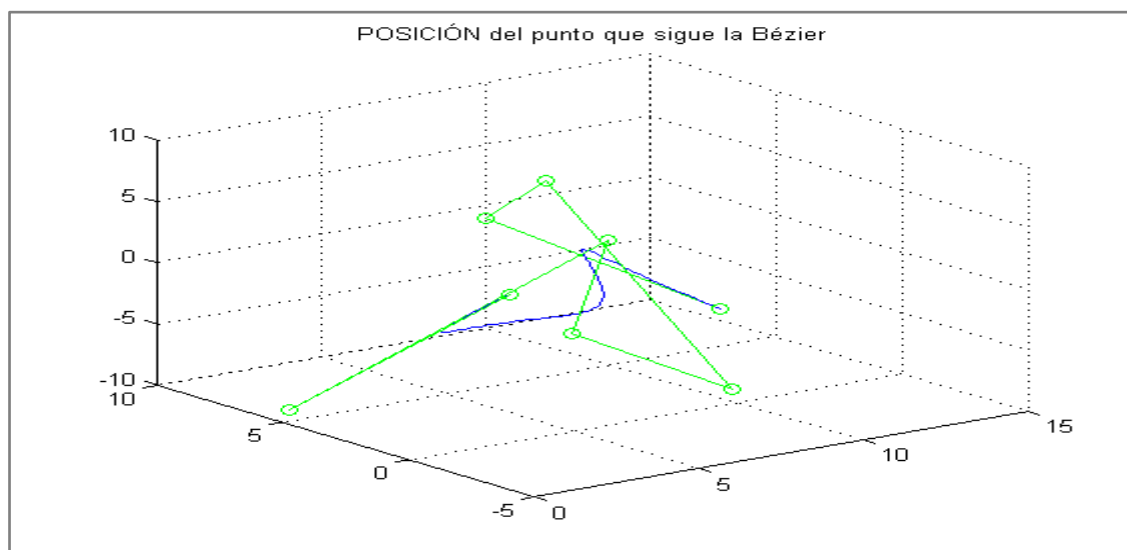


Figura 5.33: Trayectoria de la curva de Bézier 2 (ejemplo 2)

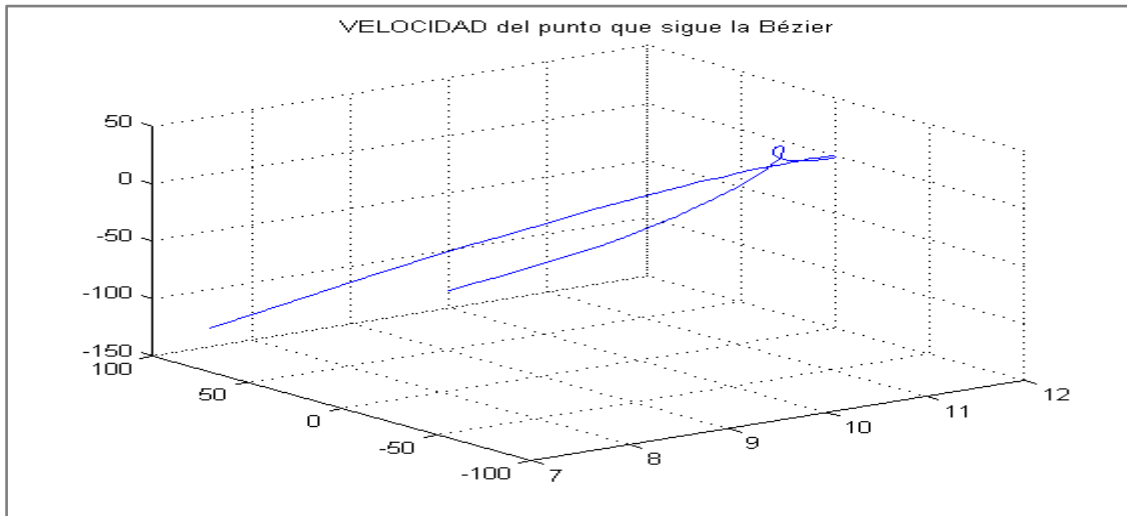


Figura 5.34: Velocidad de la curva de Bézier 2 (ejemplo 2)

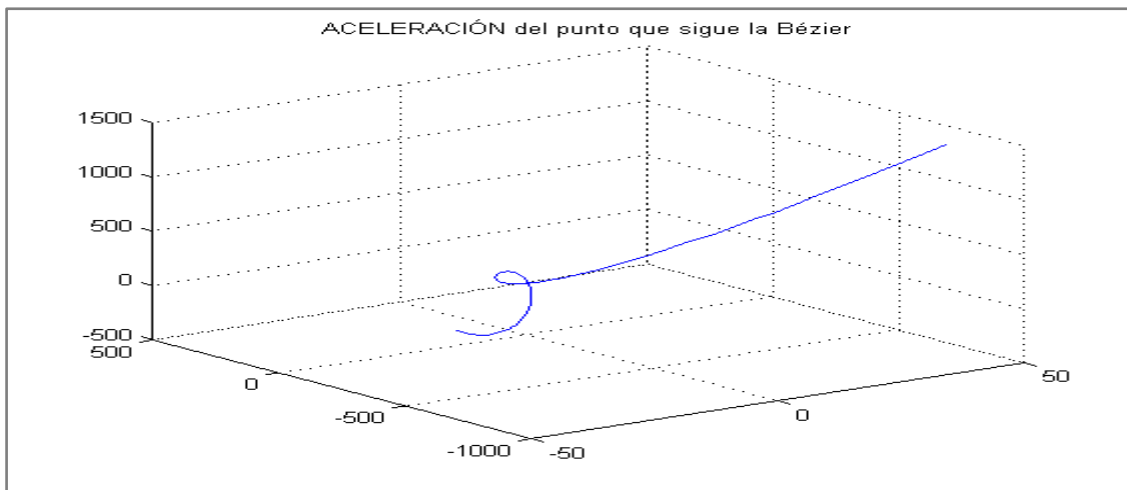


Figura 5.35: Aceleración de la curva de Bézier 2 (ejemplo 2)

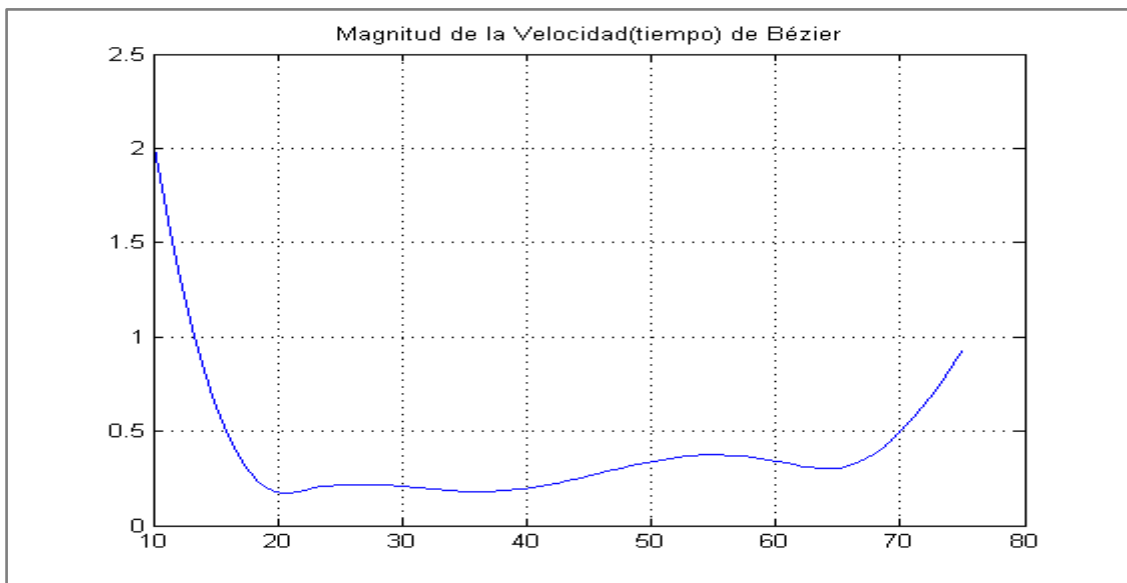


Figura 5.36: Magnitud de la velocidad en función del tiempo de la curva de Bézier 2 (ejemplo 2)

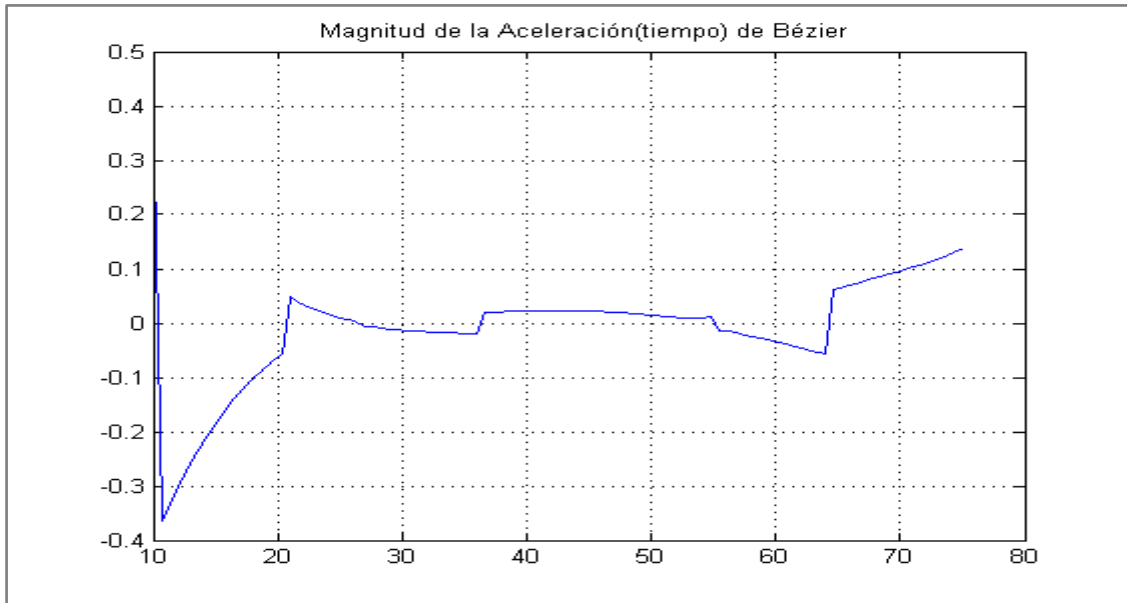


Figura 5.37: Magnitud de la aceleración en función del tiempo de la curva de Bézier 2 (ejemplo 2)

Y de la misma manera que se ha hecho en el anterior ejemplo, ejecutando las funciones “obtenerCoeDif” y “obtenerNuevaBezier” se detecta si existe colisión futura y se obtienen los Coefficient Points de la nueva trayectoria en caso afirmativo.

Realizando una comparación entre la distancia “distMin” obtenida en “obtenerCoeDif” y “distMin2” obtenida en “obtenerCoeDif_2” se observa una semejanza entre resultados similar a la del ejemplo 1:

Name	Value	Min	Max
distMin	3.3396	3.3396	3.3396
distMin2	3.3396	3.3396	3.3396

Figura 5.38: Comparación entre “obtenerCoeDif” y “obtenerCoeDif_2” (ejemplo 2)

Posteriormente a ello, se obtienen gráficas comparativas entre la nueva y la antigua Bézier 1 y entre la posición original y nueva de la esfera de radio “R1”:

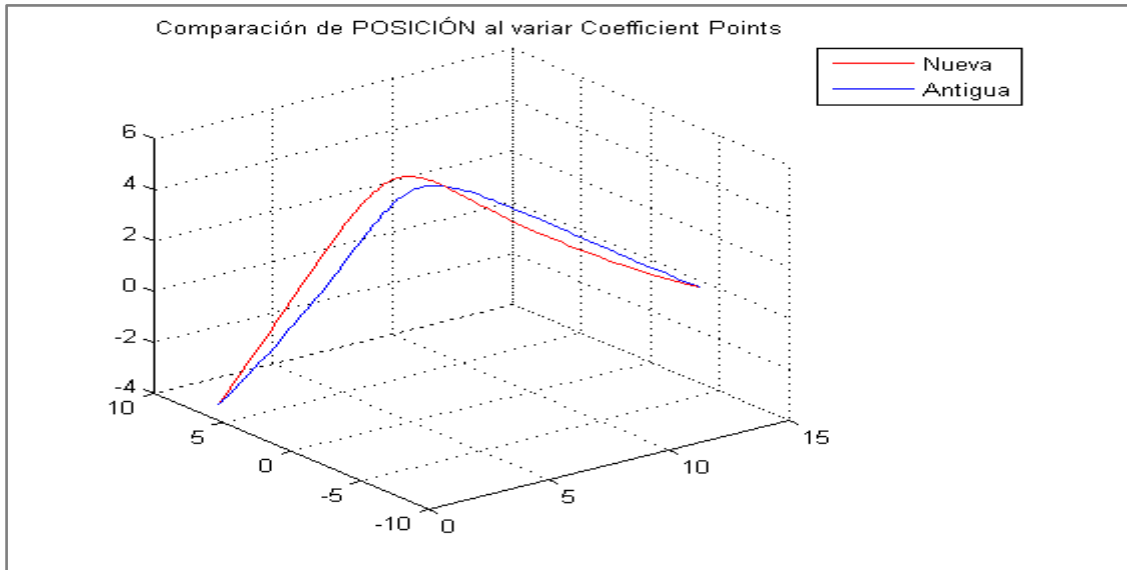


Figura 5.39: Comparación entre trayectorias de la antigua y la nueva Bézier 1 (ejemplo 2)

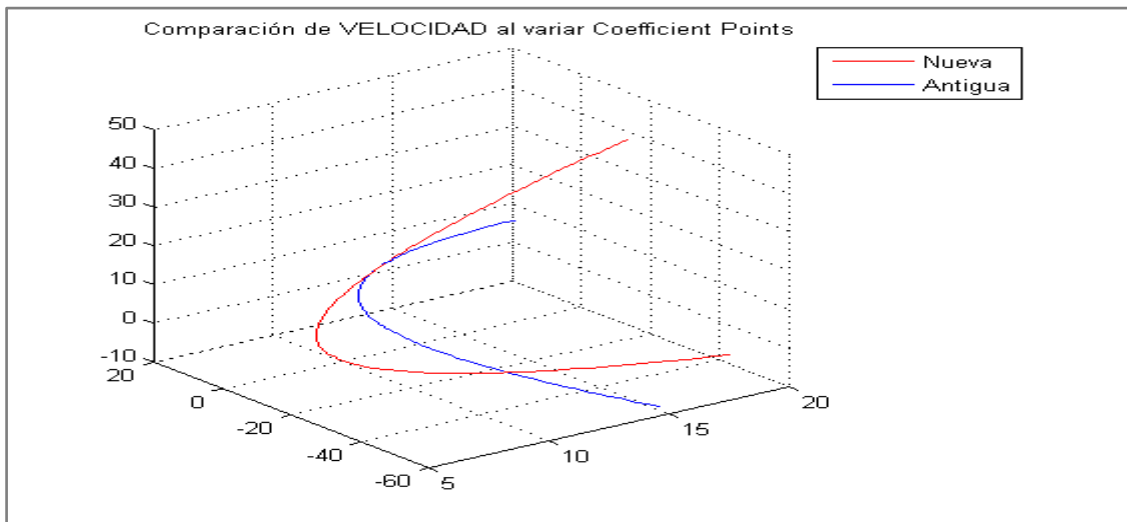


Figura 5.40: Comparación entre velocidades de la antigua y la nueva Bézier 1 (ejemplo 2)

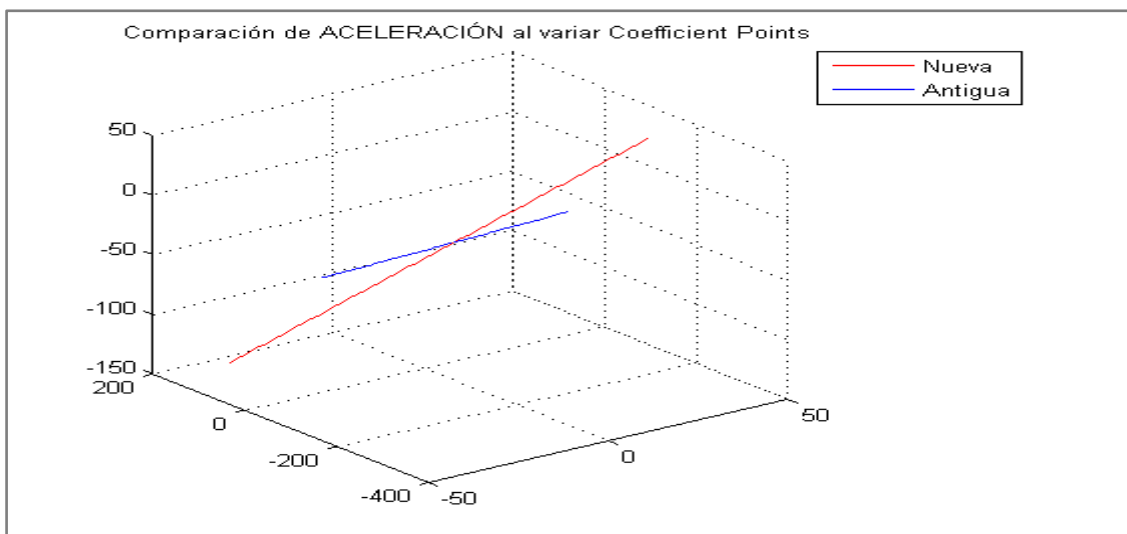


Figura 5.41: Comparación entre aceleraciones de la antigua y la nueva Bézier 1 (ejemplo 2)

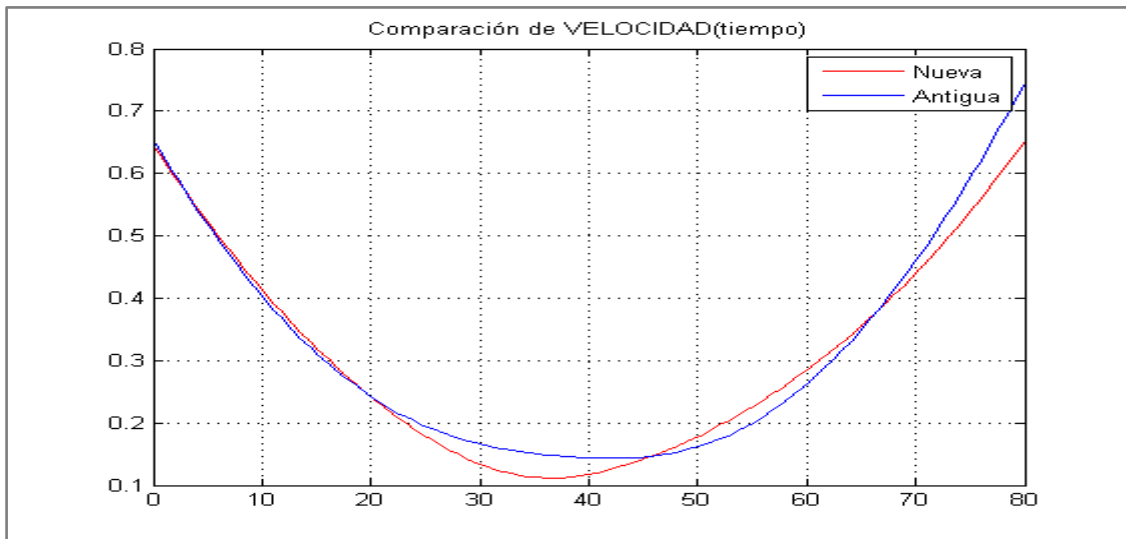


Figura 5.42: Comparación entre magnitudes de velocidad de la antigua y la nueva Bézier 1 (ejemplo 2)

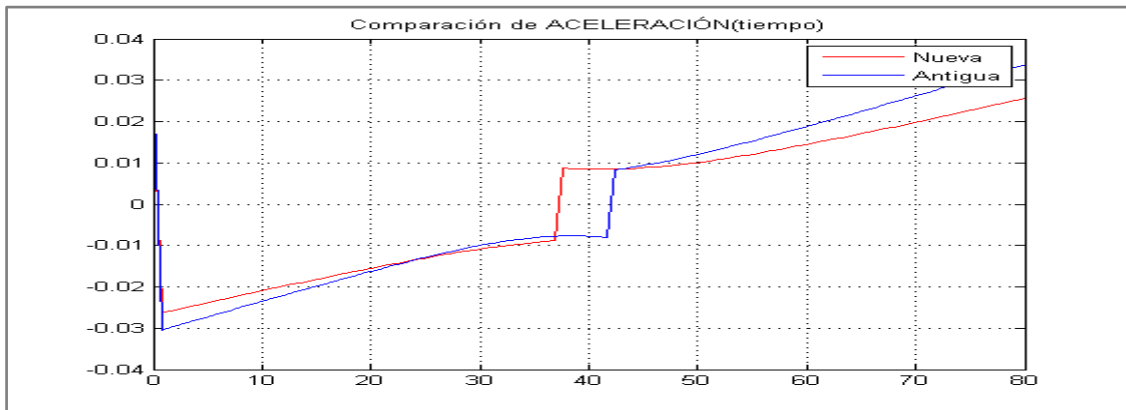


Figura 5.43: Comparación entre magnitudes de aceleración de la antigua y la nueva Bézier 1 (ejemplo 2)

En este ejemplo, tampoco se puede extraer una gran información visual al representar todas las esferas con sus respectivas curvas al mismo tiempo, por lo que volverán a diferenciarse los dos casos nombrados anteriormente:

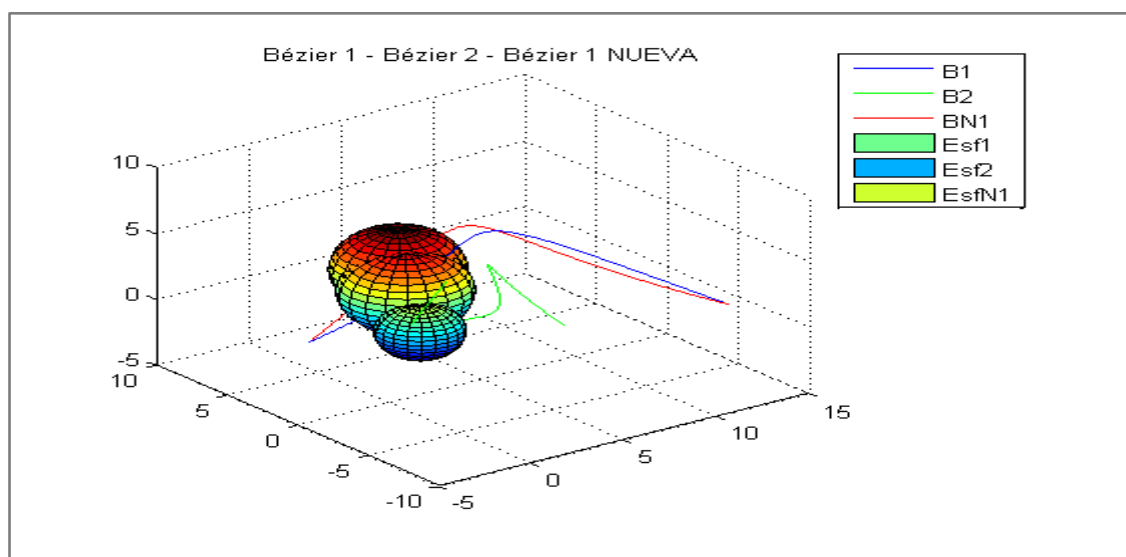


Figura 5.44: Esferas 1 y2 originales y 1 modificando su posición (ejemplo 2)

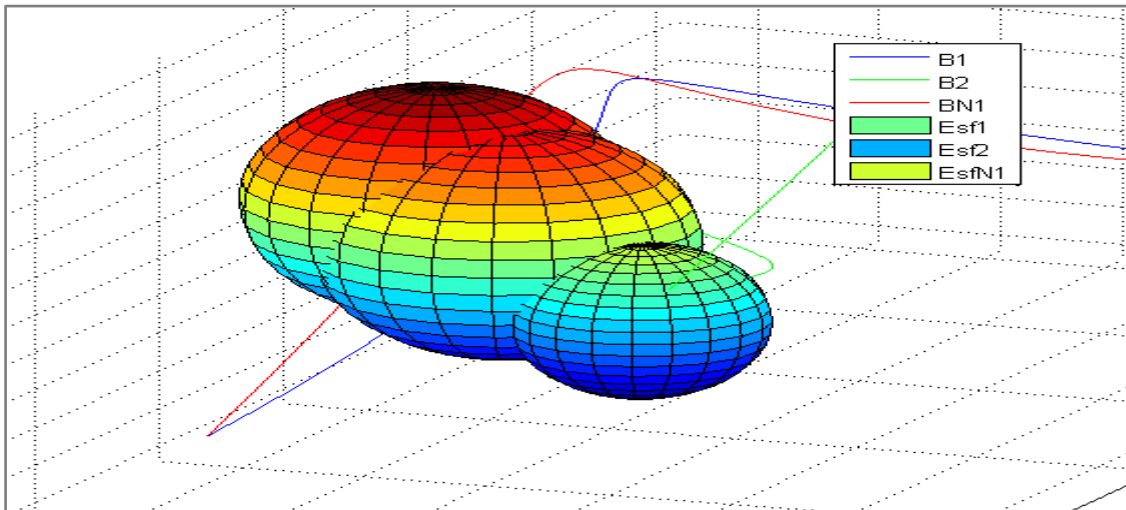


Figura 5.45: Zoom a la Figura 5.46 (ejemplo 2)

CASO 1: CURVAS 1 Y 2 ORIGINALES (CON COLISIÓN)

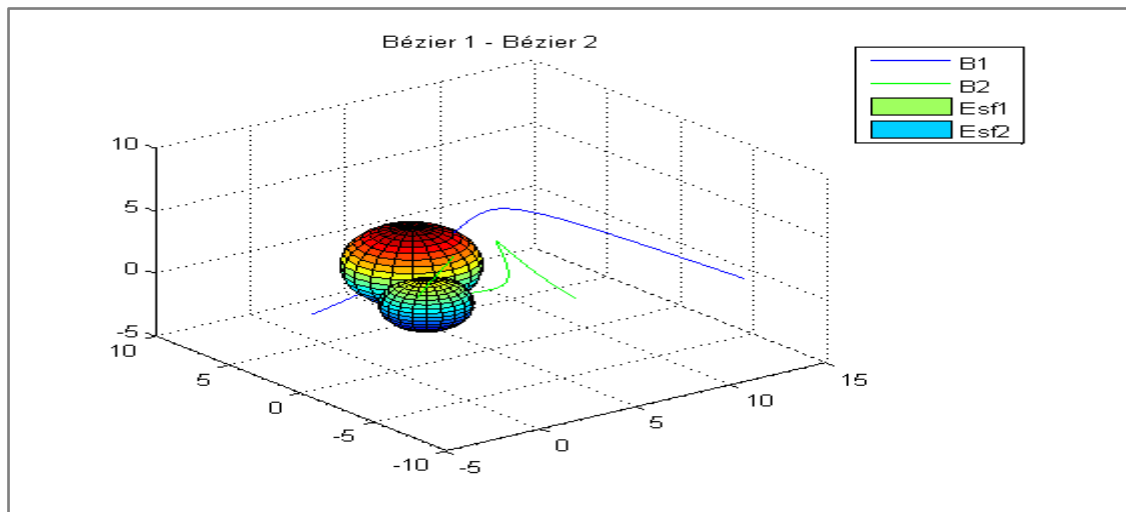


Figura 5.46: Esferas 1 y 2 originales (ejemplo 2)

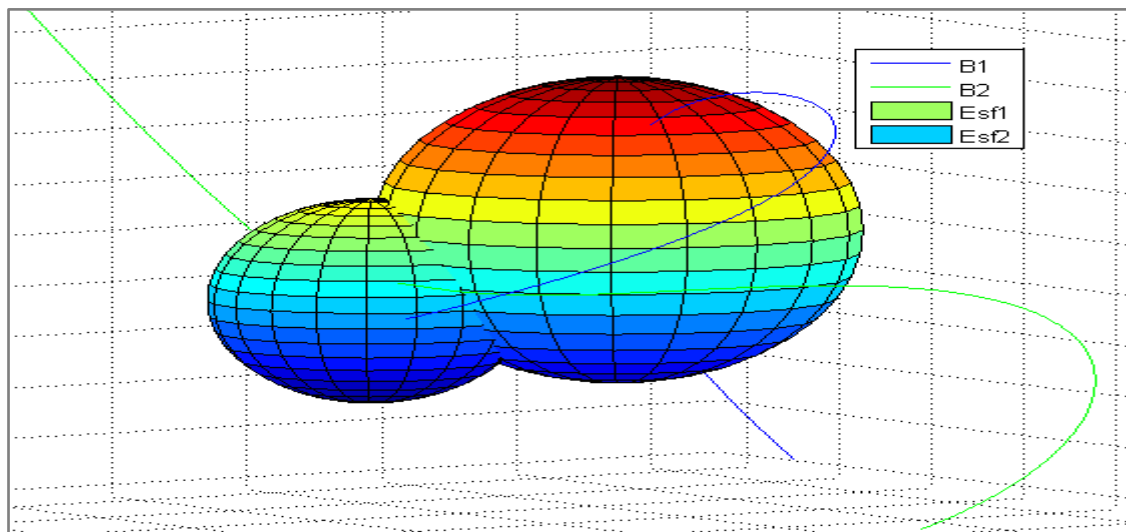


Figura 5.47: Zoom 1 a la Figura 5.48 (ejemplo 2)

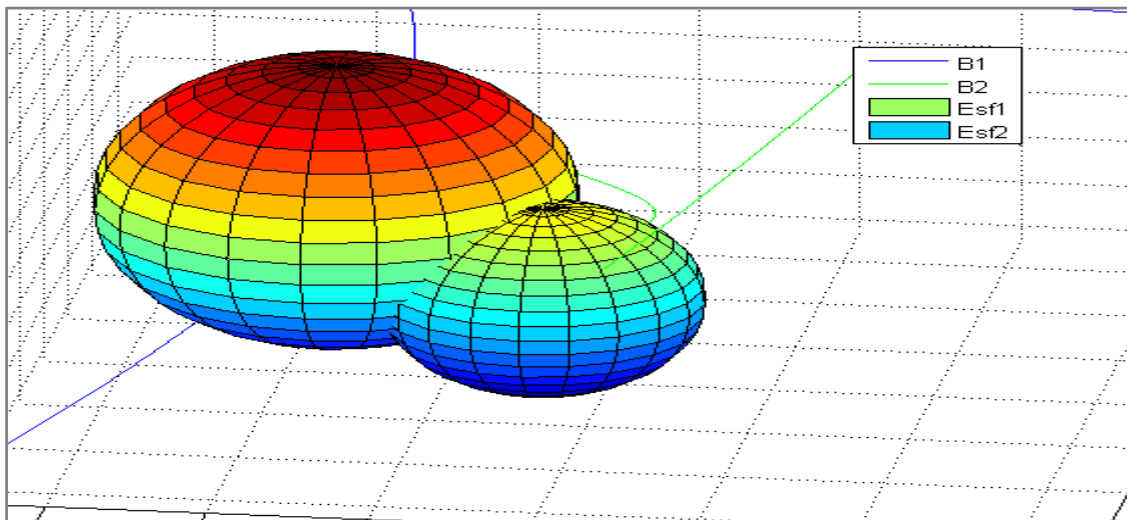


Figura 5.48: Zoom 2 a la Figura 5.48 (ejemplo 2)

CASO 2: CURVA 1 MODIFICADA Y 2 ORIGINAL (SIN COLISIÓN)

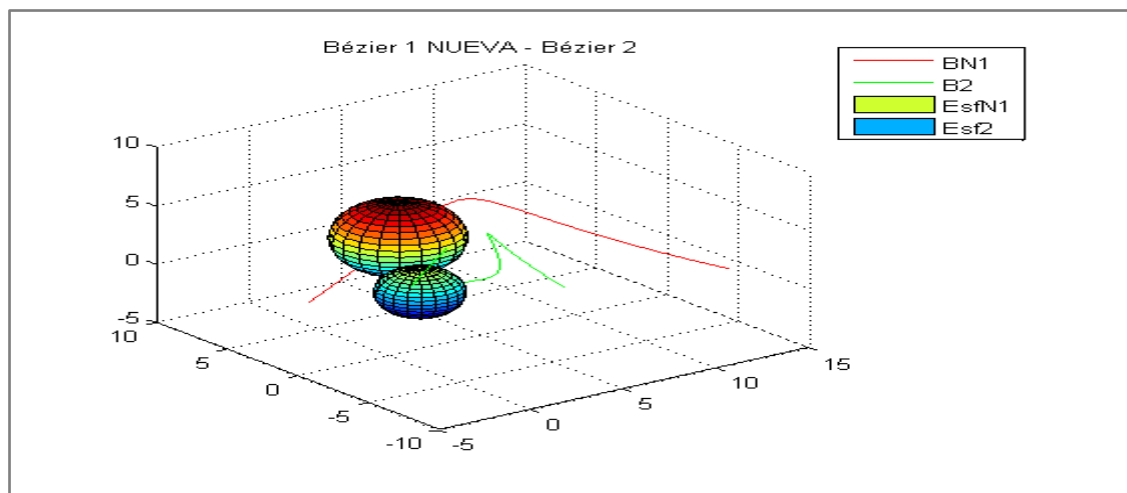


Figura 5.49: Esfera 1 modificando su posición y 2 original (ejemplo 2)

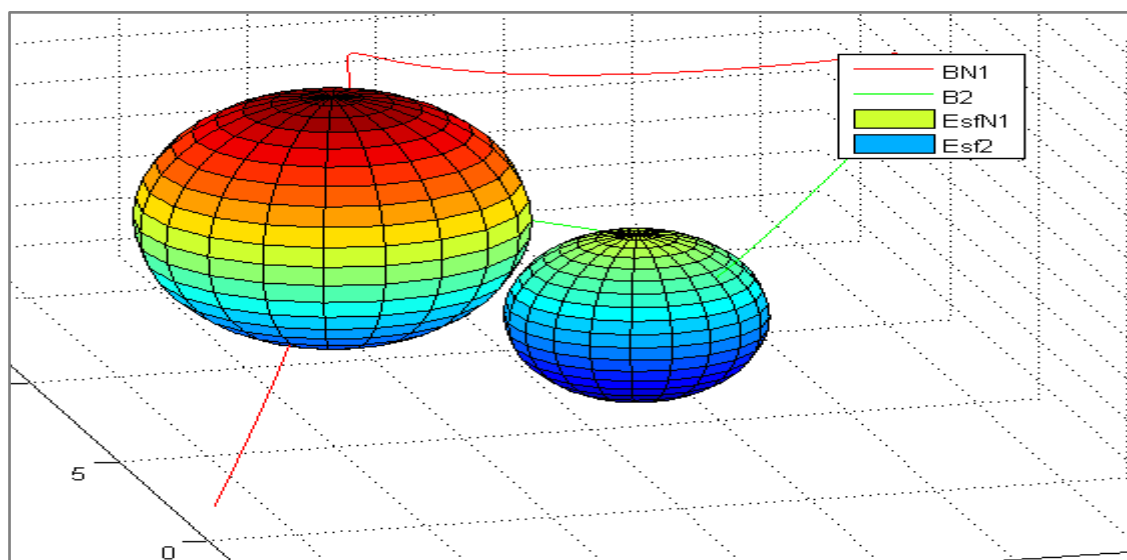


Figura 5.50: Zoom 1 a la Figura 5.51 (ejemplo 2)

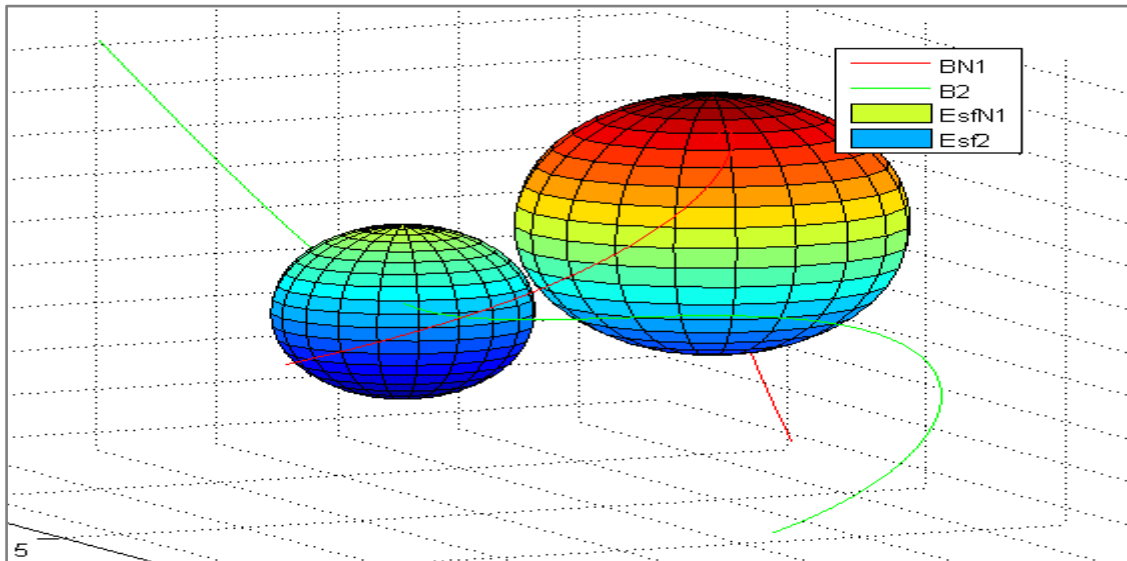


Figura 5.51: Zoom 2 a la Figura 5.51 (ejemplo 2)

Como puede apreciarse, entre un caso y otro se detecta la desaparición de la colisión en el instante de distancia mínima entre esferas. Por tanto, mediante estos dos ejemplos se ha podido demostrar que el resultado que ofrece este conjunto de funciones es correcto.

Capítulo 6

Conclusiones

La implementación en MATLAB de las diferentes funciones, todas ellas unificadas como una sola herramienta, presentadas a lo largo de la memoria ha traído como consecuencia unos resultados muy favorecedores, y es que la evasión de una futura colisión durante el vuelo de los objetos voladores consigue cumplirse con creces.

También es destacable que se generen nuevas trayectorias muy similares a las que debía seguir el objeto inicialmente sin producir grandes cambios en su velocidad ni en su aceleración. Sin embargo, el proceso de programación basada en matrices y vectores también ha ofrecido una serie de dificultades, ya que no siempre es sencillo el trasvase de algunos conceptos teóricos a la programación.

Entre algunos de los problemas se han encontrado: la dificultad para generar y completar de una forma concreta algunas matrices cuyo tamaño dependiese de los parámetros de entrada que se dan de inicio; saber cómo representar los objetos voladores, puesto que todos los ejemplos mostrados son simulaciones que quizás no se acercan totalmente a la forma real de éstos; y conseguir generar las esferas en las representaciones gráficas, cuyos comandos necesarios eran desconocidos hasta la elaboración de este trabajo.

Finalmente, como trabajos futuros podría considerarse la aplicación de la herramienta un objeto volador real, puesto que el funcionamiento de ésta se ha demostrado constantemente con simulaciones.

Parte II

Bibliografía

Bibliografía

- [1] Página Web. *MathWorks: MATLAB, 2018* [fecha de consulta: 2 de julio de 2018]. Disponible en <<https://es.mathworks.com/products/matlab.html>>.
- [2] Página Web. *MathWorks: Simulink, 2018* [fecha consulta: 2 de julio de 2018]. Disponible en <https://es.mathworks.com/products/simulink.html?s_tid=hp_ff_p_simulink>.
- [3] Página Web. *Pierre Bézier [en línea]. Wikipedia, La enciclopedia libre, 2017* [fecha de consulta: 28 de mayo de 2018]. Disponible en <https://es.wikipedia.org/wiki/Pierre_B%C3%A9zier>.
- [4] Página Web. *Curva de Bézier [en línea]. Wikipedia, La enciclopedia libre, 2018* [fecha de consulta: 29 de mayo de 2018]. Disponible en <https://es.wikipedia.org/wiki/Curva_de_B%C3%A9zier>.
- [5] Página Web. *EcuRed, Pierre Étienne Bézier, 2018* [fecha de consulta: 28 de mayo de 2018]. Disponible en <http://www.ecured.cu/Pierre_%C3%89tienne_B%C3%A9zier>.
- [6] Página Web. *Solid Modeling Association, 2018* [fecha de consulta: 28 de mayo de 2018]. Disponible en <<http://solidmodeling.org/awards/bezier-award/>>.
- [7] Página Web. *Temas para la Educación, 2011* [fecha de consulta: 29 de mayo de 2018]. Disponible en <<https://www.feandalucia.ccoo.es/docu/p5sd8625.pdf>>.
- [8] Página Web. *Curvas de Bézier, 2010* [fecha de consulta: 29 de mayo de 2018]. Disponible en <<http://www.geometriadinamica.cl/2010/12/curvas-de-bezier/>>.
- [9] Página Web. *Mínimos cuadrados [en línea]. Wikipedia, La enciclopedia libre, 2017* [fecha de consulta: 24 de junio de 2018]. Disponible en <https://es.wikipedia.org/wiki/M%C3%ADnimos_cuadrados>.
- [10] Página Web. *MathWorks: scatter3, 2018* [fecha de consulta: 18 de mayo de 2018]. Disponible en <<https://es.mathworks.com/help/matlab/ref/scatter3.html>>.
- [11] Página Web. *MathWorks: Funciones de diagramación básicas, 2018* [fecha de consulta: 18 de mayo de 2018]. Disponible en <https://es.mathworks.com/help/matlab/learn_matlab/basic-plotting-functions.html>.

- [12] Página Web. *An Example of Plotting Spheres in Matlab, 2018* [fecha de consulta: 25 de mayo de 2018]. Disponible en <<https://www.uwyo.edu/ceas/resources/current-students/classes/esig%20help/windows%20help%20files/matlab/matlab-plotting%203d%20spheres.pdf>>.
- [13] Página Web. *MathWorks: surf, 2018* [fecha de consulta: 15 de mayo de 2018]. Disponible en <<https://es.mathworks.com/help/matlab/ref/surf.html>>.
- [14] Página Web. *Constrained Least Squares, 2017* [fecha de consulta: 24 de junio de 2014]. Disponible en <https://stanford.edu/class/ee103/lectures/constrained-least-squares_slides.pdf>.

Parte III

Presupuesto

Presupuesto

7.1. Necesidad del presupuesto

Cualquier trabajo o proyecto de ingeniería requiere de un gran esfuerzo en lo referente a la obtención de cálculos y resultados con el mínimo error. Sin embargo, también es necesario hacer uso de diferentes tipos de mano de obra, maquinaria o software, los cuales conllevan unos costes económicos.

Es por ello que se elabora un presupuesto para este TFG incluyendo los recursos empleados, entre los que se encuentran los honorarios del autor del mismo, graduado en GITI, y del tutor, la amortización del ordenador y de las licencias de software y los costes de impresión y encuadernación.

7.2. Cuadro de precios básicos

En el presente subcapítulo se definen los precios básicos de los diferentes recursos físicos o inmateriales utilizados durante el trabajo.

En primer lugar, se ha establecido que el coste del autor del TFG, siendo éste un Graduado en Ingeniería en Tecnologías Industriales, puede ser de 22 €/h por la ejecución de un proyecto de estas características, mientras que el tutor, siendo éste Ingeniero Industrial, podría cobrar 30 €/h.

Por otra parte, el trabajo se ha desarrollado fundamentalmente mediante el uso de un ordenador portátil de la casa Lenovo, el cual se adquirió por 800 € y se ha considerado que su periodo de amortización es de 5 años. Teniendo en cuenta que este año 2018 cuenta con 251 días laborables, sin considerar días festivos ni fines de semana, y que la jornada laboral es de 8 horas, se obtiene un coste de amortización del ordenador de 0.08 €/h.

De igual manera que para el ordenador, se ha supuesto que las licencias de MATLAB y Microsoft Office disponen de un periodo de amortización de 5 años, por lo que aplicando los mismos datos que antes y considerando que los precios por obtener sus licencias perpetuas son 2000 € y 149 € respectivamente, se obtienen unos costes de amortización de 0.20 €/h y 0.01 €/h.

Finalmente, para la impresión y encuadernación en color y a doble cara de los diferentes documentos que componen el TFG, se ha establecido un precio de 20 € obtenido como un valor medio de los que pueden encontrarse en papelerías y copisterías.

Código	Unidad	Descripción	Precio (€)
O1	h	Graduado en GITI	22,00
O2	h	Ingeniero Industrial	30,00
H1	h	Ordenador personal	0,08
S1	h	Licencia de MATLAB	0,20
S2	h	Licencia de Microsoft Office	0,01
M1	ud	Impresión y encuadernación	20,00

Tabla 1: Cuadro de precios básicos

7.3. Cuadro de precios unitarios descompuestos

A continuación, se muestran, distinguidas por capítulos, las diferentes unidades de obra que componen la realización del proyecto. Asimismo, se ha considerado un 2% en concepto de costes directos complementarios para cada una de estas unidades, de manera que se tengan en cuenta gastos secundarios como el de la luz, la conexión a internet o material fungible.

Código	Ud.	Descripción	Rdto.	Precio	Importe (€)
CAPÍTULO 1: Trabajos previos					
1.1	h	Búsqueda de información			
		Estudio del programa MATLAB y búsqueda de información adicional			
O1	h	Graduado en GITI	1,00	22,00	22,00
H1	h	Ordenador personal	1,00	0,08	0,08
S1	h	Licencia de MATLAB	1,00	0,20	0,20
CD Comp	%	Costes Directos Complementarios	0,02	22,28	0,45
Coste Total					22,72

Tabla 2: Unidades de obra del capítulo 1

Código	Ud.	Descripción	Rdto.	Precio	Importe (€)
CAPÍTULO 2: Elaboración del proyecto					
2.1	h	Diseño de la herramienta			
		Diseño de las funciones para el cálculo de la nueva trayectoria			
O1	h	Graduado en GITI	1,00	22,00	22,00
H1	h	Ordenador personal	1,00	0,08	0,08
S1	h	Licencia de MATLAB	1,00	0,20	0,20
CD Comp	%	Costes Directos Complementarios	0,02	22,28	0,45
Coste Total					22,72
2.2	h	Redacción de la memoria			
		Redacción de distintos documentos del TFG			
O1	h	Graduado en GITI	1,00	22,00	22,00
H1	h	Ordenador personal	1,00	0,08	0,08
S2	h	Licencia de Microsoft Office	1,00	0,01	0,01
CD Comp	%	Costes Directos Complementarios	0,02	22,09	0,44
Coste Total					22,54

2.3	ud	Impresión de documentos			
		Impresión y encuadernación de los documentos del TFG			
M1	ud	Impresión y encuadernación	1,00	20,00	20,00
CD Comp	%	Costes Directos Complementarios	0,02	20,00	0,40
			Coste Total		20,40

Tabla 3: Unidades de obra del capítulo 2

Código	Ud.	Descripción	Rdto.	Precio	Importe (€)
CAPÍTULO 3: Reuniones con el tutor					
3.1	h	Tutorías privadas e individuales			
		Reuniones con el tutor para explicar conceptos teóricos y revisar la realización del trabajo			
O1	h	Graduado en GITI	1,00	22,00	22,00
O2	h	Ingeniero Industrial	1,00	30,00	30,00
CD Comp	%	Costes Directos Complementarios	0,02	52,00	1,04
			Coste Total		53,04

Tabla 4: Unidades de obra del capítulo 3

7.4. Mediciones y cuadro de presupuestos parciales

En el caso de este proyecto, se ha dedicado un total de 260 horas a su elaboración, las cuales están distribuidas en las diferentes unidades de obra.

De esta manera, para los trabajos previos se empleó un 15.83% del tiempo total, cantidad considerable ya que se debe ampliar el conocimiento de MATLAB adquirido inicialmente, además de obtener información sobre el concepto principal del proyecto. La segunda tarea que más tiempo requiere es la redacción de la memoria con un 30.77% del tiempo total, y la que más horas ha requerido por su complejidad es el diseño de la herramienta en MATLAB, ocupando la mitad de la duración del proyecto.

Código	Ud.	Descripción	Medición	Precio	Importe (€)
CAPÍTULO 1: Trabajos previos					
1.1	h	Búsqueda de información	40,00	22,72	908,98
			Coste Total		908,98
CAPÍTULO 2: Elaboración del proyecto					
2.1	h	Diseño de la herramienta	130,00	22,72	2954,18
2.2	h	Redacción de la memoria	80,00	22,54	1802,91
2.3	ud	Impresión de documentos	1,00	20,40	20,4
			Coste Total		4777,49
CAPÍTULO 3: Reuniones con el tutor					
3.1	h	Tutorías privadas e individuales	10,00	53,04	530,40
			Coste Total		530,40

Tabla 5: Cuadro de presupuestos parciales

7.5. Resumen del presupuesto

Finalmente, en este subcapítulo se muestra un resumen del presupuesto, en el que ya se consideran ciertos porcentajes en concepto de gastos generales, beneficio industrial e IVA.

RESUMEN	
Capítulo 1: Trabajos previos	908,98 €
Capítulo 2: Elaboración del proyecto	4.777,49 €
Capítulo 3: Reuniones con el tutor	530,40 €
PRESUPUESTO DE EJECUCIÓN MATERIAL (PEM)	6.216,87 €
GASTOS GENERALES (13%)	808,19 €
BENEFICIO INDUSTRIAL (6%)	373,01 €
PRESUPUESTO DE EJECUCIÓN POR CONTRATA (PEC)	7.398,08 €
IVA (21%)	1.553,60 €
PRESUPUESTO BASE DE LICITACIÓN (PBL)	8.951,67 €

Tabla 6: Resumen del presupuesto total del proyecto

El presupuesto asciende a:

OCHO MIL NOVECIENTOS CINCUENTA Y UN EUROS CON SESENTA Y SIETE CÉNTIMOS

