Universidad Cardenal Herrera-CEU
Departamento de Ingeniería de la Edificación y Producción Industrial.

TESIS DOCTORAL

TÉCNICAS COMPUTACIONALES Y ÁLGEBRA TENSORIAL PARA SU APLICACIÓN EN ROBÓTICA MÓVIL Y SIMULACIÓN NUMÉRICA

Presentada por: Lucía Hilario Pérez

Dirigida por:
Dr. Nicolás Montés Sánchez
Dr. Antonio Falcó Montesinos

Valencia, 2012

CEU Universidad Cardenal Herrera

TESIS DOCTORAL

TÉCNICAS COMPUTACIONALES Y ÁLGEBRA TENSORIAL PARA SU APLICACIÓN EN ROBÓTICA MÓVIL Y SIMULACIÓN NUMÉRICA

El Doctor Don Nicolás Montés Sánchez y el Doctor Don Antonio Falcó Montesinos, profesores de la Universidad Cardenal Herrera CEU, informan de que la Tesis, Técnicas Computacionales y Álgebra Tensorial para su Aplicación en Robótica Móvil y Simulación Numérica, de la que es autora Doña Lucía Hilario Pérez ha sido realizada bajo nuestra dirección y reúne todas las condiciones científicas y formales necesarias para su defensa.

V.º B.º de los directores:

DR. NICOLÁS MONTÉS SÁNCHEZ DR. ANTONIO FALCÓ MONTESINOS

Valencia, 22 de Octubre de 2012

Agradecimientos

Quisiera agradecer esta Tesis a mis padres, sin su esfuerzo durante años no hubiera llegado donde ahora mismo estoy. Su educación, constancia y disciplina han sido un ejemplo a seguir en momentos como éste, en la escritura de mi Tesis Doctoral. A mi hermana, a su marido y a mis sobrinas porque siempre me han ayudado. Sólo siento cuantos festivales y celebraciones me he perdido de Judith y Nuria por mi dedicación a esta Tesis.

A Pablo, sin él esta Tesis no la habría llevado nunca a cabo, su apoyo incondicional tanto físico como psíquico me han sido más que necesarios. Su saber estar a mi lado sin preguntar, tan sólo estar ahí, esa tranquilidad y serenidad que me aporta es la fuente de mi vida.

A las dos personas que son como mis hermanos: a Rebe y al "trasto" (Pedro). Sé que incondicionalmente los tengo ahí siempre. Por eso forman parte de mi familia también: Paz, Sergio, Mª Ángeles, Elisa y el/la que está en camino.

A mis dos directores, Nico y Antonio porque ambos han complementado un buen trabajo de dirección de la Tesis. Los dos hacen una unión perfecta. Gracias a su dedicación, apoyo y positividad he terminado esta Tesis.

A Malonda, por ser el mejor acompañante durante los años de Universidad y después uno de mis mejores amigos con el que he compartido los mejores momentos.

A Manu, porque sé que siempre estás ahí, porque viviendo juntas hemos compartido muchas cosas y espero que continuemos así.

A Pilar y a Pepe, porque para mí son como un modelo a seguir. Pilar gracias a tu psicología he terminado esta Tesis. Gracias por ser amiga y psicóloga.

A la familia de los "malacatones", porque desde el primer momento me adoptaron como una más y su apoyo ha sido crucial en todos los momentos.

A María, porque si ha habido alguien que me ha entendido ha sido ella, gracias por darme consejos que siempre he intentado llevar a cabo tanto en mi vida profesional como personal.

A Elena, porque es otra de las personas que sé que siempre está ahí y muchas veces he necesitado sus consejos maduros y sabios.

A mi familia política porque me hacen sentir como una de sus hijas.

A Marta Mora porque con ella he aprendido muchas cosas, su perfeccionismo y su saber hacer las cosas me han ayudado estos últimos años.

A Fernando Sánchez porque siempre me ha apoyado y ayudado para poder trabajar y terminar esta Tesis. Aunque sus críticas a veces han sido duras, las tomé como constructivas.

A Luis Domenech, Javi Muñoz y David Romero porque con sus conversaciones "frikis" siempre me han encendido una luz en el camino y siempre me han ayudado.

Al resto de mis compañeros del CEU porque siempre me he sentido apoyada por ellos y hacen que ir a trabajar sea un placer.

RESUMEN

Esta Tesis engloba dos aportaciones diferenciadas, que las podemos clasificar en transversales y verticales. Entendiendo aportaciones transversales como áreas de conocimiento que tienen una gran aplicación sobre otras áreas, por ejemplo, las matemáticas. Para esta Tesis las aportaciones transversales se subdividen en el ámbito CAGD (Computer Aided Geometric Design) y en el de los tensores. En el caso de las aportaciones verticales las entendemos como zonas del conocimiento más aisladas y que repercuten en menor medida sobre otras. En este caso, las aportaciones verticales son la robótica y el llenado de moldes en procesos tipo LCM (Liquid Composite Moulding).

En cuanto a las aportaciones transversales se ha mejorado algoritmos ya existentes que deforman curvas de Bézier y no sólo esto sino que además se ha introducido el uso de tensores en este tipo de técnicas que manipulan las curvas paramétricas. Tanto la deformación de curvas paramétricas como el uso de tensores son tópicos activos y actuales en cuanto a investigación se refiere.

Desde que se inició el uso de curvas paramétricas para el modelado de objetos resulta casi imprescindible poder manipularlas, por ello hay mucha investigación al respecto. El objetivo de estas publicaciones es desarrollar técnicas lo más simples posibles para cualquier tipo de usuario de estas curvas.

En los últimos años se ha ido introduciendo el uso de tensores en los algoritmos de grandes dimensiones ya que reducen el tiempo de cómputo. Puesto que las aportaciones transversales se van a intersectar con las verticales, la variable coste computacional es muy importante para las aplicaciones aquí utilizadas ya que en la mayoría de los casos las decisiones de los procesos son tomadas en tiempo real así que resulta relevante utilizar factores que nos ayuden en la reducción del tiempo de la CPU de los algoritmos. Esta es la razón más importante que nos ha llevado al uso de los tensores en algoritmos CAGD.

Por otra parte en las aportaciones verticales: robótica y LCM, se ha adaptado las técnicas antes citadas fusionándolas con otras ya existentes.

En el caso de la robótica hay dos problemas importantes para el diseño del camino de un robot móvil, la planificación de una trayectoria flexible y la detección-evitación de obstáculos. Utilizando las técnicas CAGD y algoritmos de detección-evitación de obstáculos (Campos Potenciales), se ha obtenido una trayectoria continua, suave y flexible capaz de evitar los obstáculos en tiempo real. La introducción de los tensores permitirá obtener la trayectoria más precisa posible.

Para los procesos LCM, identificar en el molde la frontera entre la zona seca y mojada por la resina (frente de avance) es muy importante para la mejora del proceso. Esta información se obtiene mediante técnicas de elementos finitos dando una solución discreta al problema. La introducción de técnicas CAGD fusionado con algoritmos de seguimiento de partículas mejorará la representación del frente ya que nos proporcionará un frente continuo y manipulable para actualizar su cálculo a medida que el molde se llena.

Gracias a la formulación llevada a cabo en las aportaciones transversales, abre la posibilidad de ampliar las verticales y obtener muchas más aplicaciones además de las citadas anteriormente.

RESUM

Aquesta Tesis agrupa dues aportacions diferenciades, que les podem classificar en transversals i verticals. S'entén aportacions transversals com àrees de coneixement que tenen una gran aplicació en altres diferents, per exemple, les matemàtiques. Per aquesta Tesis les aportacions transversals es subdivideixen en l'àmbit CAGD (Computer Aided Geometric Design) i en el dels tensors. En el cas de les aportacions verticals s'entén com a zones del coneixement més aïllades i que repercuteixen en menor mesura sobre altres. En aquest cas, les aportacions verticals són la robòtica i l'omplert de motlles de tipus LCM (Liquid Composite Moulding).

Pel que fa a les aportacions transversals s'ha millorat els algoritmes ja existents que deformen corbes de Bézier i no sols açò si no que a més a més s'ha introduït l'ús de tensors en aquest tipus de tècniques que manipulen les corbes paramètriques. Tant la deformació de corbes paramètriques como l'ús de tensors són tòpics actius i actuals en quant a investigació es refereix.

Des de que s'inicià l'ús de corbes paramètriques per al modelat d'objectes resulta quasi imprescindible poder manipular-les, per això existeix molta investigació al respecte. L'objectiu d 'aquestes publicacions és desenvolupar tècniques el més simple possible per a qualsevol tipus d'usuari d'aquestes corbes.

En els darrers anys s'ha introduït l'ús de tensors en els algoritmes de grans dimensions ja que redueixen el temps de còmput. Com que les aportacions transversals s'intersecten amb les verticals, la variable cost computacional és molt important per a les aplicacions que ací s'utilitzen ja que en la major part dels cassos les decisions dels processos s'han de prendre en temps real així que resulta rellevant utilitzar factors que ens ajuden a la reducció del temps de la CPU dels algoritmes. Aquesta és la raó més important que ens ha portat a l'ús dels tensors en algoritmes CAGD.

Per una altra banda en les aportacions verticals: robòtica i LCM, s'ha adaptat les tècniques abans esmentades fusionant amb les ja existents.

En el cas de la robòtica hi ha dues problemes importants per al disseny del camí d'un robot mòbil, la planificació d'una trajectòria flexible i la detecció-evitació d'obstacles. Utilitzant les tècniques CAGD i algoritmes de detecció-evitació d'obstacles (Camps Potencials), s'ha obtingut una trajectòria continua, suau i flexible capaç d'evitar els obstacles en temps real. La introducció dels tensors permetrà obtindre la trajectòria més precisa possible.

Per als processos LCM, identificar en el motlle la frontera entre la zona seca i mullada per la resina (front d'avançament) és molt important per a millorar el procés. Aquesta informació s'obté mitjançant tècniques d'elements finits donant una solució discreta al problema. La introducció de tècniques CAGD fusionades amb algoritmes de seguiment de partícules millorarà la representació del front ja que ens proporcionarà un front continu i manipulable per actualitzar el seu càlcul mentre s'ompli el motlle.

Gràcies a la formulació duta a terme en les aportacions transversals, s'obri la possibilitat d'ampliar les verticals i obtindre moltes més aplicacions a més a més de les esmentades anteriorment.

ABSTRACT

This Thesis covers two different contributions that it is possible to classify them in transverse and vertical. Transverse contributions mean knowledge areas with a huge application, for example, mathematics. In this Thesis the transverse contributions are divided into the area of CAGD (Computer Aided Geometric Design) and tensors. On the other hand, the vertical contributions mean more isolated areas of knowledge with fewer applications. In this case, the vertical contributions are related to robotics and LCM (Liquid Composite Moulding) mould filling simulation.

The transverse contributions have improved the algorithms that modified the Bézier curves and also it has been introduced the use of tensors in this techniques that modify the parametric curves. The use of tensors and the parametric curves deformation are important research issue.

The parametric curves are widely used for shape design, for that reason it is necessary to manipulate them, consequently there is a lot of research in this field. The objective of the articles is to develop techniques as simple as possible for all users.

Recently, there is an increased interest in algorithms of high dimensions that make use of tensors because they reduce the computational time. The transverse contributions intersect with the vertical ones, for that reason the computational cost is an important factor for the applications of this Thesis. The decisions must be taken in real-time and it is necessary to reduce the CPU time of the algorithms. This is the most important reason of using tensors in CAGD algorithms.

Otherwise, the vertical contributions: robotics and LCM, have been adapted to these techniques joining them with other ones in these fields.

In mobile robots there are two important research issues in the area of motion planning: flexible trajectory planning and avoiding the obstacles. It has been used the CAGD techniques and planning collision free algorithms (Potential Fields) to obtain a continuous, smooth and flexible trajectory free of collisions in real-time. The introduction of tensors allows obtaining the most accurate trajectory.

In LCM, an important issue to improve the process is the identification of the front between the wet and dry area of resin in the mould (flow front). This information is computed through finite element methods giving a discrete solution of the problem. The introduction of CAGD techniques combined with algorithms of particles tracking will improve the representation of the flow front, because it will be a continuous and deformable updating while it is filling the mould.

Thanks to the development carried out in the transverse contributions, it is opened the possibility of giving more vertical contributions to obtain as much applications as possible.

A mi futuro bebé, ella
ha sido sin duda el esfuerzo final
que necesitaba. Contigo empezaremos una
nueva vida, gracias por convertirme
en madre y doctora.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción.

No basta saber, se debe también aplicar. No es suficiente querer, se debe también hacer.

Johann Wolfgang Goethe (1749-1832)

1.1. Introducción.

La deformación de curvas paramétricas es un tema actual de investigación que suscita muchas publicaciones al respecto.

Por otra parte, los procesos de llenado de moldes con resina líquida para la construcción de piezas y la robótica móvil también son temas actuales y muy activos en cuanto a investigación se refiere.

En ambos casos las curvas paramétricas son un tema de interés que pueden mejorar las técnicas y algoritmos ya existentes. En particular, las curvas de Bézier calculan de forma rápida y sencilla sus puntos, además son flexibles, estables numéricamente, pasan por el primer y último punto de control, están contenidas en la envolvente convexa, son fácilmente manipulables, etc. Éstas y muchas otras propiedades marcan la ventaja del uso de las curvas paramétricas frente a otro tipo de curvas.

La presente Memoria agrupa el uso de las curvas paramétricas y su manipulación aplicado tanto en la robótica como en el llenado de moldes.

1.2. Motivación.

Los algoritmos que aquí se presentan vienen motivados inicialmente por la búsqueda de una solución para poder representar de forma continua el frente de avance de la resina en el llenado de un molde.

Hasta el momento el frente de avance se conocía de forma discreta mediante técnicas de elementos finitos. Dicha búsqueda tiene que ir ligada a tener la información del frente con una curva continua. Como las curvas paramétricas son las que más se utilizan en modelados geométricos, se ha utilizado una de ellas para calcular el frente de avance.

Como consecuencia de ello, hay una primera investigación que engloba el uso de las curvas paramétricas más utilizadas en CAGD (Computer Aided Geometric Design) y el estudio exhaustivo de las diferentes técnicas de manipulación de éstas. La deformación de curvas es uno de los tópicos más investigados en la actualidad ya que para los diseñadores resulta de vital importancia poder manipular las curvas cuando pretenden conseguir el modelo de cierto objeto.

Finalmente se desarrolla el primer algoritmo de la Memoria que deforma curvas Bézier mediante vectores.

Es una Tesis que se mueve en dos ámbitos diferenciados, por un parte el llenado de moldes con resina líquida y por la otra, la robótica móvil. Ambas líneas quedan integradas bajo el tronco común del uso de las curvas paramétricas y su deformación, fusionando el algoritmo con los ya existentes en ambas áreas.

Por otra parte, la robótica móvil es uno de los campos de investigación más activos donde se utilizan curvas paramétricas para la planificación de las trayectorias de los robotos autónomos. Hasta ahora el uso de las curvas paramétricas para las trayectorias de los robots no se había combinado con algoritmos de evitación de obstáculos.

Esto motiva la posibilidad de adaptar la técnica que se había desarrollado anteriormente, al ámbito de las trayectorias de los robots móviles. Así pues se consigue deformar en tiempo real, de forma que el robot no colisione con los obstáculos dinámicos del entorno.

Finalmente, el último punto que motiva esta Tesis es la evaluación del coste numérico de los algoritmos y como conseguir reducirlo al máximo sin pérdida de prestaciones.

El inconveniente que presenta el algoritmo utilizado en el ámbito de la robótica, es el aumento del coste computacional a medida que se quiere conseguir una trayectoria más precisa. Esto motiva la búsqueda de estrategias que reduzcan este coste numérico del algoritmo. Y ello nos llevará al uso de los tensores en los algoritmos desarrollados.

1.3. Objetivos de la Tesis.

La presente Tesis aborda la problemática de la deformación de curvas Bézier aplicándolo en dos líneas de investigación diferenciadas.

Estas dos líneas que ya hemos comentado antes serían:

- La robótica móvil y el diseño de trayectorias para una navegación segura mediante curvas paramétricas.
- El llenado de moldes de resina y la representación del frente de avance.

Así pues podríamos hablar de dos aportaciones diferenciadas.

I. Aportaciones transversales en el ámbito matemático. Mejoras en las técnicas desarrolladas en CAGD y uso de los tensores en la deformación de curvas paramétricas.

II. Aportaciones verticales en el ámbito de la ingeniería. La planificación de caminos en la robótica móvil y el estudio del frente de avance en el llenado de moldes.

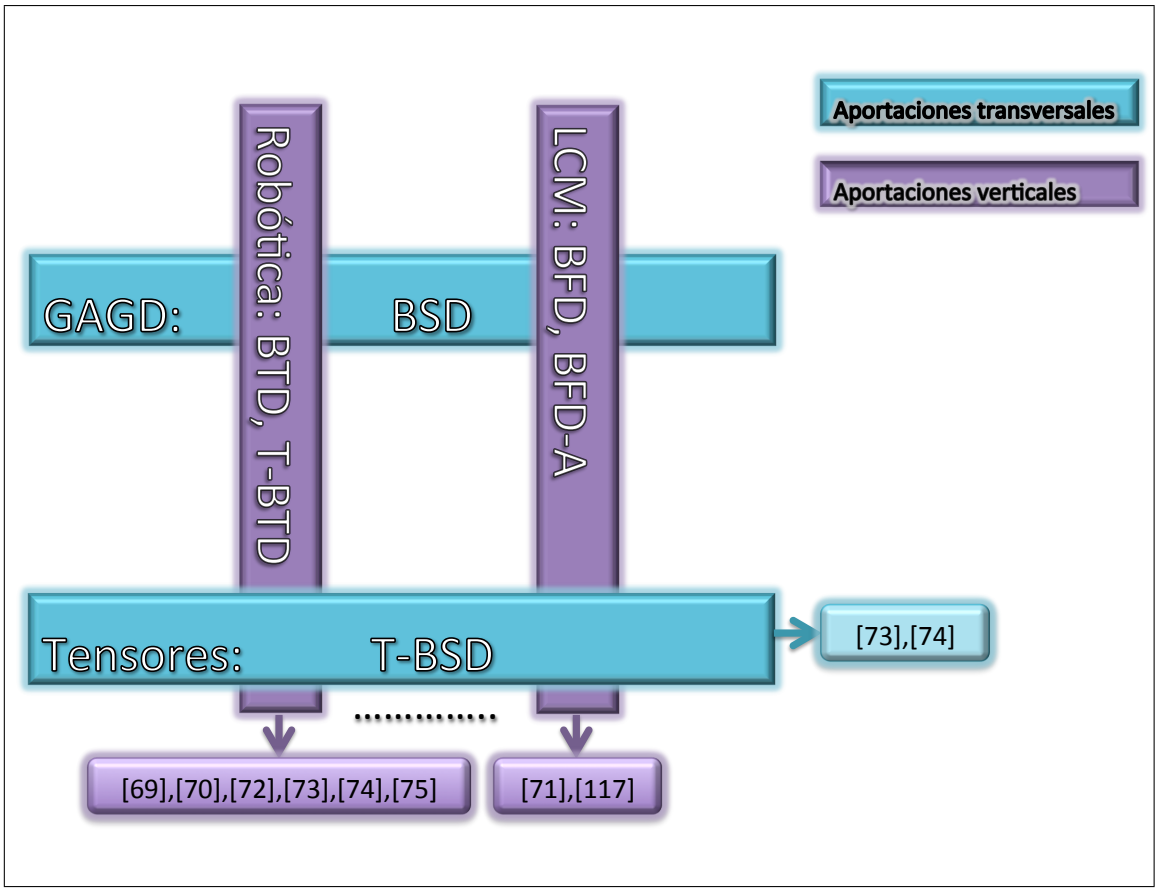


Figura 1.1: Aportaciones de la Tesis Doctoral.

En la figura 1.1 podemos ver un esquema gráfico de estas aportaciones con las respectivas publicaciones que ha suscitado la investigación llevada a cabo en esta Memoria.

En consecuencia se han abordado cuatro objetivos generales diferenciados:

- En un primer momento y realizando un estudio del estado del arte profundo en determinadas áreas de la ingeniería, se ha visto la necesidad de solucionar determinados problemas abiertos, como por ejemplo, la planificación de trayectorias para la robótica y el cálculo del frente de avance de la resina en el llenado de un molde. Ambos problemas están solucionados de diferentes formas, en la presente Memoria se ha mejorado la solución de la forma más eficiente y óptima.
- Con esa necesidad de buscar soluciones a problemas abiertos y apoyándonos en las diferentes publicaciones referentes a la deformación de curvas paramétricas, se ha desarrollo una técnica computacional para la deformación de curvas Bézier a través de un campo de vectores.

- En muchos de los casos en la ingeniería, las decisiones se toman en tiempo real. Por ello, las técnicas computacionales desarrolladas deben utilizar el menor tiempo posible de cómputo. Así pues se buscan estrategias matemáticas que permitan reducir el coste computacional en las técnicas desarrolladas. Una de esas estrategias es el uso de los tensores. Con ella se mejora el tiempo de cómputo, además de obtener una notación mucho más compacta y depurada.
- Por último, hay que adaptar los algoritmos aquí formulados en los diferentes ámbitos que se han citado anteriormente. Esto supone la fusión con técnicas ya existentes, consiguiendo de esta forma mejoras sustanciales.

1.4. Organización del documento.

La siguiente Memoria se estructura en cinco capítulos, el presente capítulo 1 que es donde se presenta la motivación y objetivos de la Tesis.

En el capítulo 2 detallaremos la deformación de una curva de Bézier a través de un campo de vectores. Está formulado de dos formas diferenciadas, una con una nomenclatura tradicional, que llamaremos *Bézier Shape Deformation* (**BSD**). Y la otra con una formulación basada en tensores, denominado como *Tensor-Bézier Shape Deformation* (**T-BSD**). Además de detallar la forma de realizar esta deformación, en el mismo capítulo se realizará un estado del arte en cuanto a la deformación de curvas paramétricas y del uso de tensores en diversos ámbitos para la mejora del tiempo de cómputo de los algoritmos.

Las aplicaciones del algoritmo **BSD** y **T-BSD** se describen en los siguientes dos capítulos: 3 y 4.

En el capítulo 3 se explicará la adaptación del algoritmo **BSD** a la simulación del llenado de un molde con resina líquida, los llamados procesos *Liquid Composite Moulding* (LCM). Con el algoritmo **BSD** se representa y actualiza el frente con una curva Bézier, recibiendo el nombre de *Bézier Flow Front Deformation*, **BFD**. En este caso se fusionará con el uso de las técnicas de elementos finitos (*Finite Element Method*), **FEM** y una técnica que hace evolucionar las partículas, **EP**, dentro del molde.

Gracias al planteamiento que tiene el algoritmo **BFD** se permite exigir más condiciones o restricciones al problema, en consecuencia se mejora la aproximación del frente de avance. Por ello, en este capítulo se introduce una restricción más al algoritmo para mejorar la aproximación obtenida del frente. Esta restricción está relacionada con el caudal de resina introducida en el molde en cada instante de tiempo, en este caso lo llamaremos *Bézier Flow Front Deformation-Area* (**BFD-A**).

La otra aplicación del algoritmo **BSD** y **T-BSD** se lleva a cabo en el capítulo 4. En este caso ambos algoritmos se adaptarán al ámbito de la robótica. La idea principal radica en definir la trayectoria de un robot móvil a través de curvas de Bézier y además el robot debe evitar colisionar con los obstáculos móviles del entorno. En este caso el algoritmo se denominará: *Bézier Trajectory Deformation* (**BTD**). Para poder deformar la trayectoria, el **BTD** se fusionará con una de las técnicas más novedosas y actuales para la evitación de obstáculos, que es el uso de los Campos Potenciales, los famosos *Potential Field Methods*,

PF. Este algoritmo necesita aplicarse en tiempo real y como el coste computacional del **BTD** aumenta con el número de curvas de Bézier que se necesitan para planificar la trayectoria, entonces se aplica por primera vez en la robótica un algoritmo basado en tensores (al menos para conocimiento del autor), obteniendo el llamado *Tensor-Bézier Trayectory Deformation*, **T-BTD**. De esa forma el coste computacional se reduce considerablemente.

Finalmente, en el capítulo 5 se obtiene un resumen de las conclusiones obtenidas tras la investigación llevada a cabo en la presente Memoria, además de detallar las líneas futuras de investigación que surgen tras ella.

El documento se cierra con toda la información de las referencias bibliográficas utilizadas a lo largo del texto y dos apéndices A y B que completan la información que se necesita a lo largo de los capítulos.

Capítulo 2

Bézier Shape Deformation y Tensor-Bézier Shape Deformation: BSD y T-BSD.

Las matemáticas poseen no sólo la verdad, sino cierta belleza suprema. Una belleza fría y austera, como la de una escultura.

Bertrand Rusell (1872-1970)

2.1. Introducción.

2.1.1. Las curvas paramétricas: Bézier.

Las curvas tanto en el espacio como en el plano, son una parte de la geometría necesarias para poder representar determinadas formas en diferentes áreas, ver por ejemplo la Figura 2.1. Las curvas surgen en muchas aplicaciones, como el arte, el diseño industrial, las matemáticas, arquitectura, ingeniería, etc. Las siguientes imágenes ilustran ejemplos en diferentes áreas donde podemos encontrar las curvas.

Podemos encontrar curvas en la construcción de edificios arquitectónicos, ver por ejemplo las figuras 2.2 y 2.3.

Incluso también podemos encontrar curvas en áreas como la biología, ver por ejemplo la figura 2.4 donde se observa una curva que modela la trayectoria de un insecto.

Además, las curvas son extensamente utilizadas en ámbitos como el de la ingeniería, ver la figura 2.5.

Una misma curva de puede definir de diferentes formas:

1. **Forma explícita.** Esta forma de representar una curva, despeja una de las variables en función de la otra. En el plano, las coordenadas (x,y) de los puntos de la curva plana definida de forma explícita satisfacen $y = f(x)$ o $x = g(y)$. En el caso de que

Figura 2.1: Curvas en diferentes áreas.

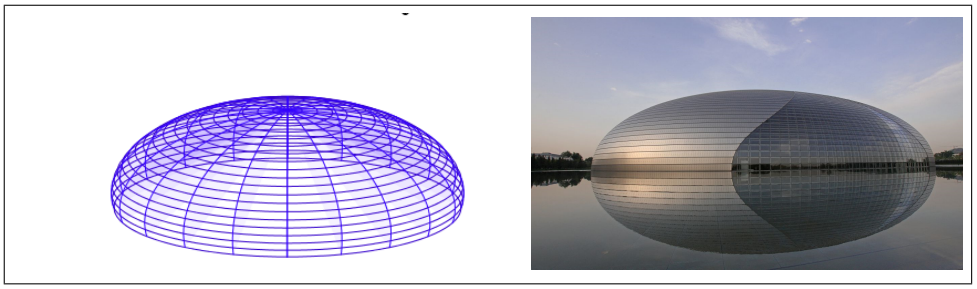


Figura 2.2: Curvas en la arquitectura.

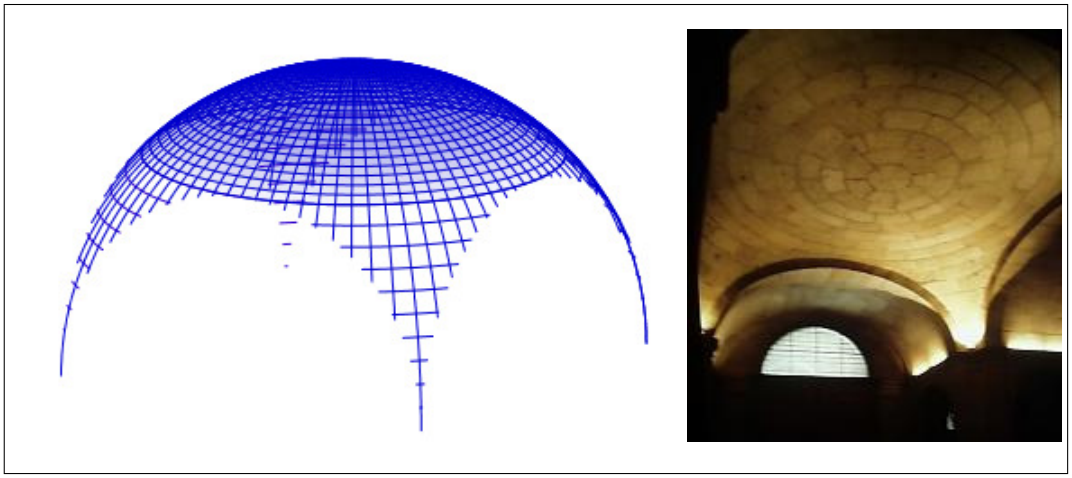


Figura 2.3: Curvas en la arquitectura.

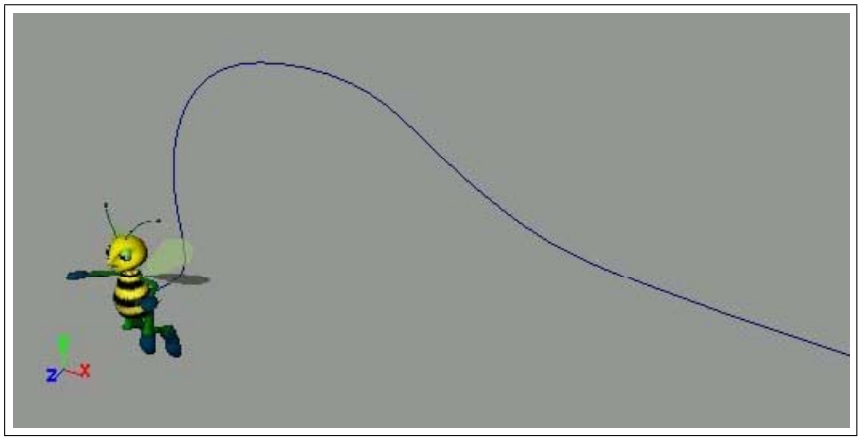


Figura 2.4: Curva para diseñar la trayectoria de una abeja.

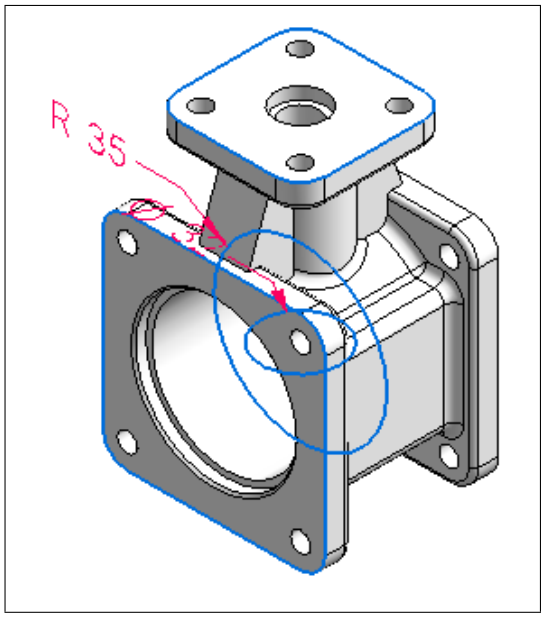


Figura 2.5: Curvas en piezas de ingeniería.

la curva esté en el espacio, su forma explícita se podría definir como: $x = f(z)$ y $y = g(z)$.

2. **Forma implícita.** Las coordenadas (x,y) de los puntos de una curva plana definida de forma implícita verifican que: $F(x,y) = 0$, para alguna función F. Si la curva está en R^3 entonces la curva debe satisfacer estas dos condiciones simultáneamente $F(x,y,z) = 0$ y $G(x,y,z) = 0$.

3. **Forma paramétrica.** Las coordenadas de una curva paramétrica están expresadas en función de un parámetro, por ejemplo u. La definición de una curva definida en R^n se podría hacer de la siguiente forma:

$$\boldsymbol{\alpha} : [a,b] \to R^n / \boldsymbol{\alpha}(u) = (\alpha_1(u), \cdots, \alpha_n(u)); u \in [a,b].$$

Cada α_i son las funciones coordenadas o funciones componentes. La imagen de $\boldsymbol{\alpha}(u)$ se denomina traza de $\boldsymbol{\alpha}$, y $\boldsymbol{\alpha}(u)$ es la parametrización de $\boldsymbol{\alpha}$.

El proceso de dibujar una curva se denomina renderizar. Las curvas paramétricas son las que más se utilizan en *computer graphics* y *geometric modelling* porque los puntos de la curva se calculan de forma sencilla. En cambio, el cálculo de los puntos a través de la expresión implícita es bastante más complejo.

Dentro de las curvas paramétricas podemos diferenciar entre curvas polinómicas y curvas racionales. La curvas polinómicas son aquellas cuyas funciones componentes son polinomios y las curvas racionales son aquellas que se expresan como cociente de polinomios, $\frac{p(u)}{q(u)}$, siendo $p(u)$ y $q(u)$ dos polinomios.

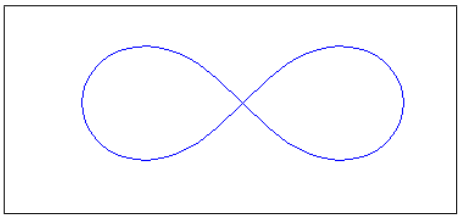
Ejemplo: En el cuadro 2.1 podemos ver un ejemplo de una misma curva representada de tres formas diferentes:

Explícita	$y = 2 + \sqrt{4-(x-1)^2}$
Paramétrica	$F(x,y) = (x-1)^2 + (y-2)^2 - 4 = 0$
Implícita	$\alpha(u) = (1+2\cdot\cos u, 2+2\cdot\sin u); u \in [0,\pi]$

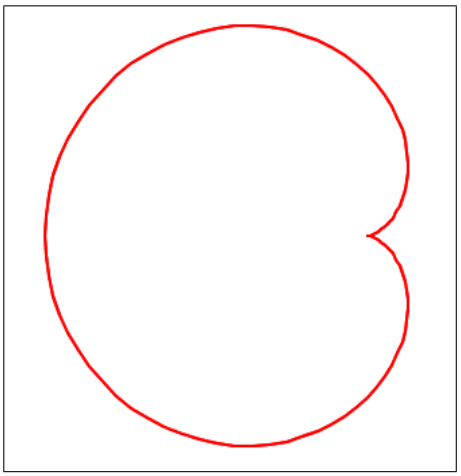
Cuadro 2.1: Diferentes formas de definir una misma curva.

Las ventajosas propiedades de las curvas paramétricas hacen que sean las más utilizadas, las citadas propiedades son:

- intuitivas
- flexibles
- afín-invariantes



(a) Lemniscata de Bernoulli, estudiada por Jacob Bernoulli en 1694.



(b) Cardiode, llamada así por su semejanza con el corazón.

Figura 2.6: Ejemplos de curvas paramétricas.

- rápidas de computar
- estables numéricamente

La representación de las curvas paramétricas posibilita una gran variedad de curvas, unas conocidas, otras extrañas, algunas complejas, otras sorprendentes por su simetría y belleza.

Algunos ejemplos de estas curvas pueden ser las que vemos en la figura 2.6.

Para poder modelar formas o superficies complicadas es necesario introducir una forma de representar curvas basándose en un polígono. A partir de esta idea surgen las curvas paramétricas más utilizas en CAGD (Computer Aided Geometric Design): Bézier, B-Splines, Rational Bézier (RBC) y las Non-Uniform Rational B-Splines (NURBS).

En la figura 2.7 se representa un esquema de las curvas más importantes en CAGD. En él se puede observar como las curvas NURBS son las más generales y las más particulares son las curvas de Bézier. Pero las Bézier son las más simples, poseyendo propiedades que hacen que sean de las más utilizadas.

En el apéndice A están definidas todas las curvas CAGD más relevantes. En esta Memoria el estudio se ha centrado en las curvas de Bézier.

Las curvas Bézier surgieron a consecuencia del modelado de automóviles tanto de Renault como Citroën por los ingenieros Pierre Bézier y Casteljau. La sencillez para poder manipular estas curvas hacen que su uso y aplicación sea mucho más amplio. Por ejemplo, en la arquitectura podemos encontrar tanto el uso de curvas como de superficies paramétricas. En la figura 2.8 podemos ver como una superficie de Bézier se utiliza para la construcción de la cubierta de un edificio, que en este caso es un Restaurante.

Otras áreas donde podemos encontrar curvas paramétricas también puede ser la ingeniería y dentro de ella en la simulación de trayectorias tanto de robots móviles como

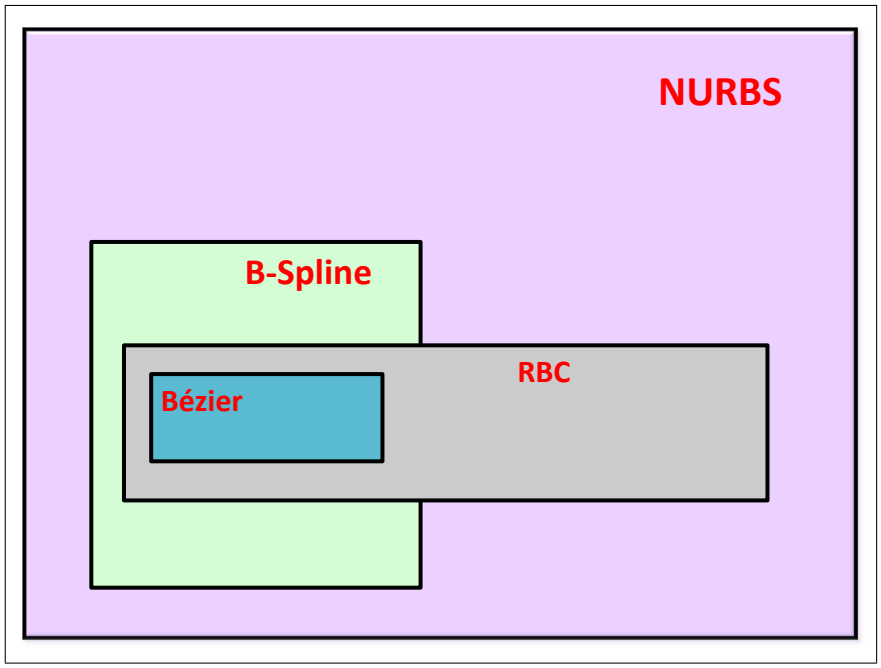
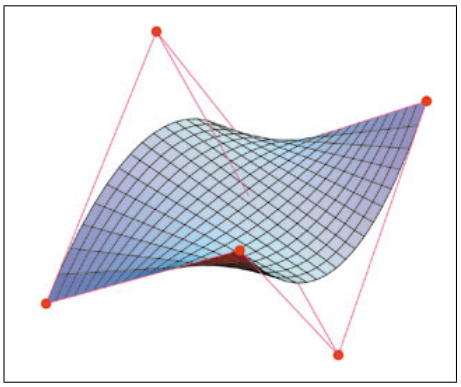


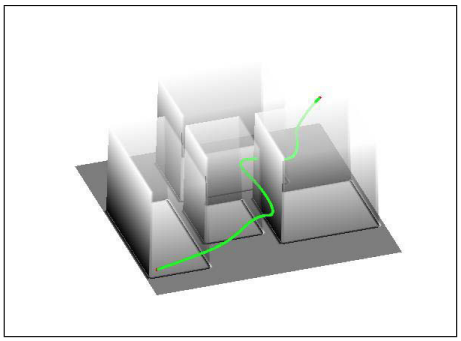
Figura 2.7: Esquema de las curvas más importante en CAGD.

(a) Restaurante de las Alquerias (Plana Baixa).

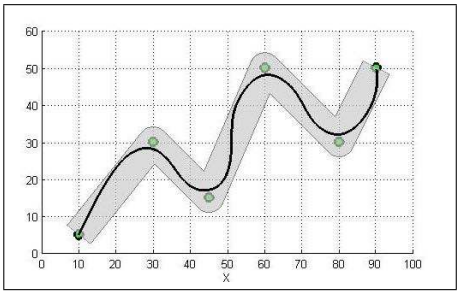


(b) Superficie de Bézier utilizada en la cubierta del restaurante.

Figura 2.8: Ejemplo de una superficie Bézier.



(a) Ejemplo de la simulación de una trayectoria de un UAV.



(b) Ejemplo de la simulación de la trayectoria de un robot móvil.

Figura 2.9: Ejemplo del uso de curvas paramétricas en la robótica móvil.

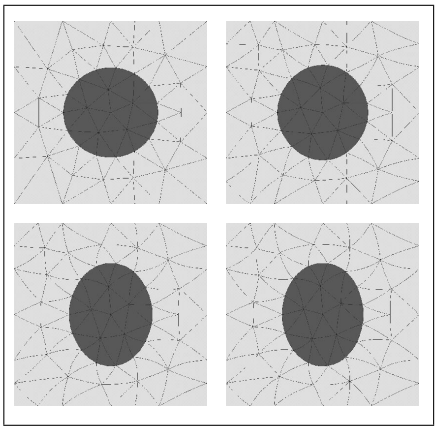


Figura 2.10: Utilización de curvas de Bézier en el mallado que se utiliza en elementos finitos.

de UAV's (Unmanned Aerial Vehicle), en la figura 2.9 podemos ver un par de ejemplos que ilustran el uso tanto de curvas paramétricas planas como curvas paramétricas en el espacio.

Además de la robótica, donde son muy utilizadas las curvas paramétricas, en menor medida encontramos el uso de ellas para mejorar técnicas de simulación de determinados procesos donde se aplican elementos finitos. Las curvas paramétricas aproximan mejor los dominios donde se resuelve el proceso. En la figura 2.10 se ilustra un ejemplo.

Curva de Bézier.

La popularidad de las curvas de Bézier se debe a sus numerosas propiedades matemáticas que facilitan su manipulación y análisis. Además no se requieren grandes conocimientos matemáticos para utilizarlas, algo que interesa a los diseñadores que dan forma a los objetos.

Definición 1. *Una curva de Bézier de grado n viene especificada por una secuencia de* $(n+1)$ *puntos de control,* $\boldsymbol{P}_i$*. El polígono que une los puntos de control es el llamado*

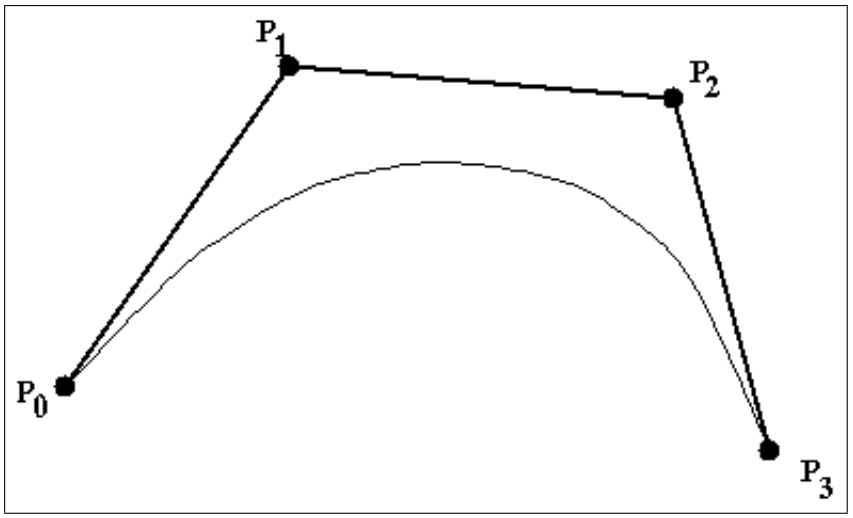


Figura 2.11: Curva de Bézier con cuatro puntos de control.

polígono de control. La definición la podemos ver en la ecuación 2.1, (ver [57], [129], [137] y [110]).

$$\alpha(u) = \sum_{i=0}^{n} \boldsymbol{P}_i \cdot B_{i,n}(u)\,, u \in [0,1] \tag{2.1}$$

Los puntos de control son los que dan forma al polígono de control. Y las funciones o bases utilizadas son los polinomios de Bernstein $B_{i,n}(u)$ que se definen de la siguiente forma.

$$B_{i,n}(u) = \binom{n}{i} u^i (1-u)^{n-i}\,, i = 0, \cdots, n \tag{2.2}$$

La dimensión del vector de los puntos de control estará relacionada con la dimensión del espacio donde se represente la curva. En la figura 2.11 vemos un ejemplo de una curva de Bézier en el plano de orden 3 con su respectivo polígono de control.

Cuando se trabaja con curvas, lo más importante no sólo es poder representarlas si no además es muy importante poder manipular su forma. El objeto que se quiere modelar condicionará el tipo de curva paramétrica que se escoja. Hay diferentes técnicas para obtener la deformación de una curva paramétrica, el objetivo es conseguir que sea lo más sencilla posible porque el usuario no siempre va a tener amplios conocimientos de matemáticas.

En la siguiente sección 2.1.2 vamos a destacar algunas de las publicaciones más relevantes de la deformación de curvas paramétricas.

2.1.2. Deformación de curvas paramétricas.

Uno de los tópicos más investigados en el ámbito de las curvas paramétricas es la deformación de éstas, como consecuencia podemos encontrar muchas publicaciones al

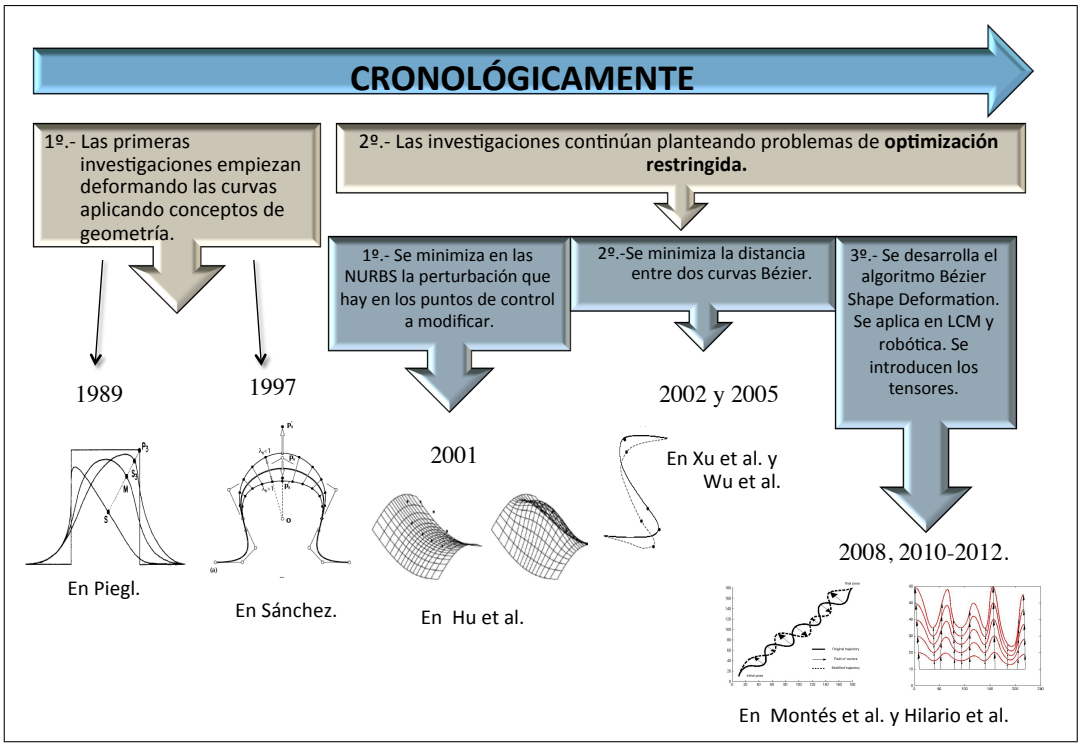


Figura 2.12: Esquema cronológico de la deformación de curvas.

respecto. Veamos un resumen de estos trabajos. En la figura 2.12 se presenta un breve esquema gráfico de la evolución que ha habido.

A partir de la figura se concluye como las primeras publicaciones estaban centradas en técnicas geométricas y posteriormente se desarrollan algoritmos donde se plantean problemas de optimización restringida. Veamos con detalle estas publicaciones.

Para obtener la deformación hay diferentes formas en función de si la curva o superficie es racional o no. Si no es racional las técnicas se centran en la localización de los puntos de control. En cambio si fuese racional, además de los puntos de control también se pueden modificar los pesos. En la figura 2.13 se observa la diferencia entre mover un punto de control o modificar el valor del peso asociado a un punto de control.

Los artículos donde podemos encontrar diferentes algoritmos son variados. En el caso de las curvas de Bézier podemos encontrar estas publicaciones [167, 168]. Para las B-Splines tenemos artículos como [59, 84, 128] y el artículo más reciente es el [135]. Por último, en el caso de las curvas o superficies más generales, las NURBS, encontramos estos trabajos: [13, 56, 78, 79, 83, 112, 130, 131, 141, 153]

Una de las primeras publicaciones en cuanto a la deformación de las curvas paramétricas es la desarrollada por Piegl, [130], en 1989. En ella se desarrolla la deformación de una B-Spline racional. Si observamos la definición de una curva NURBS en la ecuación A.4 del apéndice A, podemos concluir que para modificar una NURBS podemos o bien cambiar el knot, o bien mover los puntos de control, o por último cambiar los pesos asociados a cada punto de control. Piegl propuso dos algoritmos: uno basado en el cambio

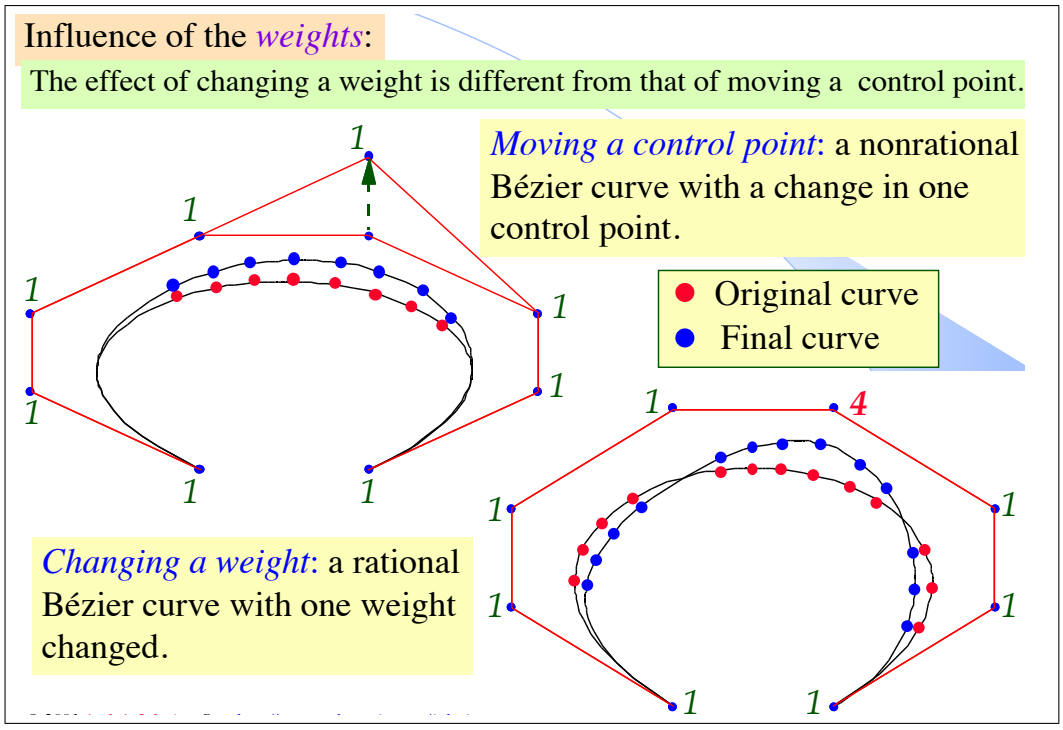


Figura 2.13: Modificación de los puntos de control de una curva de Bézier y los pesos de una curva racional de Bézier

de los puntos de control y el otro basado en el cambio de los pesos. Piegl destaca la importancia de desarrollar técnicas lo más sencillas posibles. La idea es obtener la modificación basándose en conceptos geométricos más que en cálculos. Más adelante, en 1995 se publicó el artículo [13] que modifica las curvas NURBS alterando simultáneamente los pesos asociados a los puntos de control y la localización de los puntos de control. Un par de años después, en 1997 se publicó el artículo [141] donde se desarrolla una herramienta simple para modificar NURBS. El usuario selecciona un punto **O** y desplaza el punto de control a través de una dirección radial que pasa por el punto **O**. La elección del punto **O** influye en la forma final de las curvas NURBS. En la figura 2.14 se puede ver un ejemplo de esta publicación.

En 1999 y 2001 se publicaron los artículos [78, 79] respectivamente. En ambos se desarrolla un algoritmo para modificar superficies NURBS mediante optimización restringida imponiendo que la nueva curva o superficie pase a través de unos puntos denominados *Target Points* (Puntos Finales). Este tipo de deformación se puede realizar imponiendo uno o varios *Target Points*, lo que denominaríamos *Multi-Target*. El problema se resolvió aplicando el Teorema de los Multiplicadores de Lagrange. La función a optimizar utilizada en este problema es:

$$\sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} \|\varepsilon_{ij}\|^2 \tag{2.3}$$

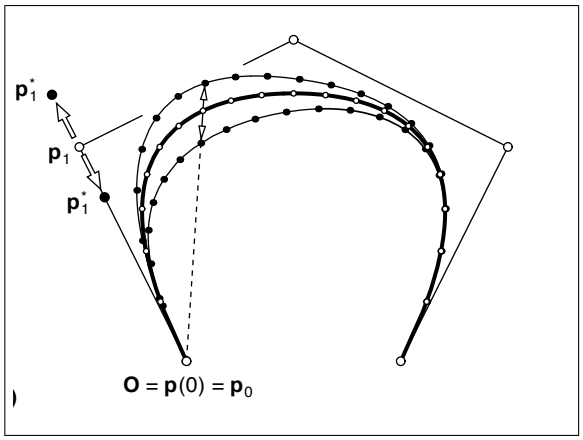


Figura 2.14: Modificación de una curva NURBS mediante la técnica publicada en el artículo [141].

Siendo ε_{ij} la perturbación de los puntos de control $\mathbf{P}_{ij}$ que se van a modificar. Podemos ver en la figura 2.15 la solución gráfica de un problema *Multi-Target*.

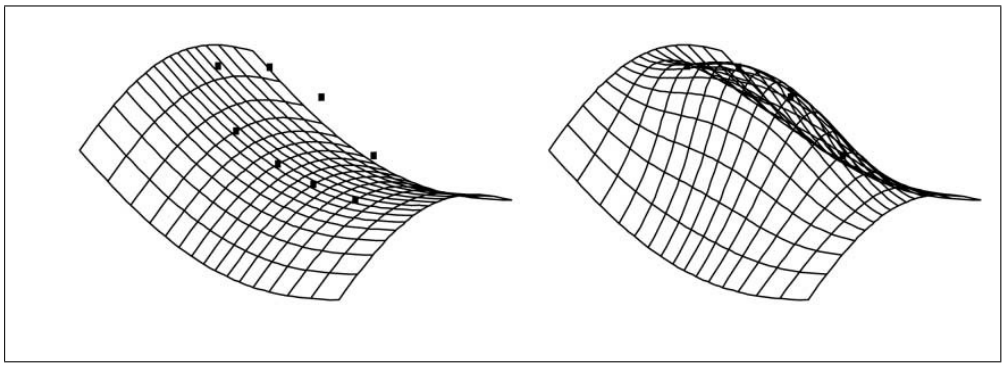


Figura 2.15: Deformación de una superficie NURBS mediante un problema Multi-Target.

Imre Juhász en esta misma línea publicó en 1999 el artículo [83]. El objetivo de dicha publicación es poder modificar los pesos de algunos puntos de control para que la curva NURBS atraviese un punto predeterminado (*target point*). Para poder obtener esta transformación se trabaja con otro espacio (*preimage space*). El problema se resuelve en dicho espacio y luego se proyecta la solución al espacio inicial. En la imagen 2.16 se puede ver un ejemplo gráfico de como se consigue la modificación.

En 2003, Meek publicaba [112], en este caso se propone la interpolación restringida con curvas cúbicas racionales, de forma que dado un conjunto de puntos ordenados situados a una parte de una polilínea, se construye una curva plana interpolando estos puntos. Esta forma de interpolar se realiza mediante curvas cúbicas racionales, de forma que muchas líneas se consideran como restricciones y los puntos no necesariamente tienen que estar a un lado de la polilínea.

Todas estas técnicas son las que podemos destacar en cuanto a la modificación de

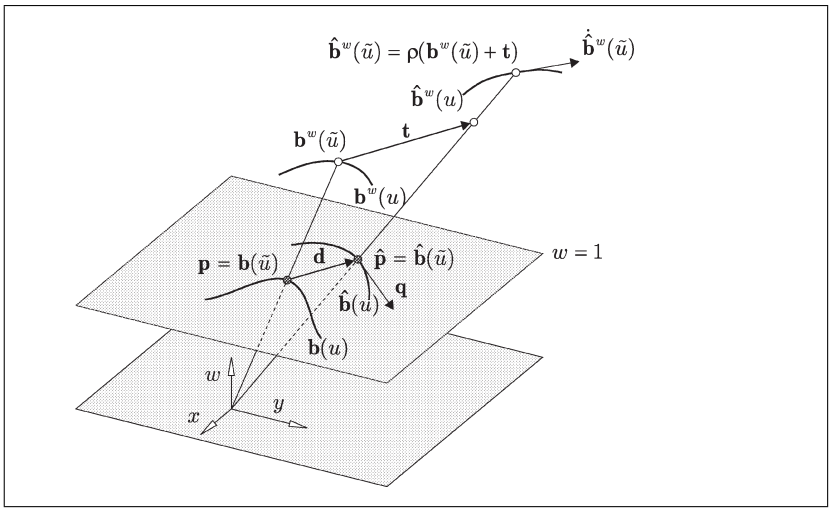


Figura 2.16: La curva $\mathbf{b}(u)$ es la proyección central de $\mathbf{b}^{\omega}$. La curva modificada es $\widehat{\mathbf{b}}(u)$ y su preimagen es $\widehat{\mathbf{b}^{\omega}}(u)$, q es la dirección tangente requerida.

curvas o superficies NURBS.

En cuanto a las curvas B-Splines, autores como Opfer y Oberle en 1988 publicaron [128]. Propusieron la derivación de una Spline cúbica con la presencia de obstáculos a través de métodos de optimización. Años después en 1993, Fowler y Barlets, [59], propusieron modificar una curva B-Spline mediante un problema de optimización restringida, minimizando la perturbación de los puntos de control con una norma o métrica determinada e imponiendo unos valores concretos para unos parámetros determinados. En 2004, encontramos el artículo [84], en él se desarrolla la modificación restringida de una B-Spline cúbica mediante los knots. La ventaja de modificar los knots es que la curva siempre quedará dentro de la envolvente convexa definida por la curva original. Así que de esa forma queda determinada la región donde seguro que va a estar definida la curva B-Spline.

Por último, una de las publicaciones más recientes relacionadas con la deformación de las curvas B-Splines fue publicada en 2009, [135]. En este caso se modifica de una curva uniforme B-Spline, minimizando los cambios de la deformación en el sentido de una aproximación por mínimos cuadrados. En este caso, se ha desarrollado un algoritmo de optimización restringida exigiendo que la curva B-Spline atraviese un determinado punto denominado *Target Point*.

De forma análoga a este último trabajo desarrollado para las curvas uniforme B-Spline, se publicaron previamente dos artículos relacionados con la deformación de las curvas Bézier, [167, 168], publicados en 2002 y 2005 respectivamente. En ambos, se aplica optimización restringida para modificar curvas de Bézier. Esta forma de deformar una curva paramétrica (en este caso una curva de Bézier) es similar a los artículos publicados [78, 79] (desarrollado para las NURBS). En ambos se plantea un problema de optimización restringida, pero difiere la función a optimizar que minimiza los cambios de la forma entre la curva o superficie original y la deformada. La función a minimizar está definida

utilizando normas diferentes. En los artículos referentes a las NURBS se minimiza la distancia entre un conjunto finito de puntos entre la original y la deformada, en cambio, en las publicaciones referentes a las curvas de Bézier se utiliza la norma $\|.\|_2$ donde queda definida la integral de la diferencia.

2.1.3. Tensores.

Uno de los factores más importantes en las técnicas o algoritmos desarrollados para aplicaciones en la ingeniería es el coste computacional. En la presente Memoria este punto es importante porque en las aplicaciones donde vamos a llevar las técnicas desarrolladas van a ser procesos cuyas decisiones deben ser tomadas on-line, por ello vamos a necesitar reducir el coste numérico tanto como se pueda.

Recientemente, el uso de tensores se ha incrementado en los algoritmos numéricos porque reducen el tiempo de cómputo de forma drástica. En particular en los espacios de grandes dimensiones donde el coste numérico es una variable importante, ya que en muchos de ellos se presentan dificultades numéricas de cómputo. El uso de tensores permite un almacenamiento de la información eficiente que permite operar de otro modo, y ello provoca una reducción en el número de operaciones. La consecuencia directa es la reducción del tiempo de cómputo, pasando de ser exponencial $O(n^f)$ a ser lineal $O(fn)$. La justificación de esta reducción la podemos encontrar en el libro de W. Hackbusch publicado en 2012, [64]. Hay otras publicaciones como [50] donde queda constancia de esta reducción del tiempo de cómputo en un algoritmo que resuelve ecuaciones en derivadas parciales de grandes dimensiones. Otras publicaciones que hacen referencia a esta reducción son: [18, 53, 54, 65, 66].

Un tensor es un vector multidimensional, es decir, un tensor de orden n es un elemento del producto tensorial de espacios vectoriales N, cada uno tiene su propio sistema de coordenadas. Un tensor de orden tres tiene tres índices, ver la figura 2.17. Un tensor de orden uno es un vector, un tensor de orden dos es una matriz y un tensor de orden tres o más se denomina tensores de orden superior. Vamos a dar un breve repaso a los tensores de orden superior y sus descomposiciones, siguiendo la publicación [93], porque pese a que en los últimos cuarenta años ha habido una investigación bastante activa referente a los tensores, no ha habido muchas publicaciones al respecto en revistas de matemática aplicada.

La representación con tensores se inicia con Hitchcock en 1927, ver las publicaciones [76, 77]. La idea del modelo "*multi-way*"se atribuye a Cattell en 1944, [29, 30]. Más tarde llega el trabajo de Tucker en 1960, [156–158], el artículo de Carroll y Chang [28] y la publicación de Harshman [67] en 1970, todas ellas aparecieron en psicometría . En 1981 Appellof y Davidson [12] fueron los primeros que utilizaron la descomposición con tensores para el ámbito de la quimiometría y fue a partir de entonces cuando empezaron a surgir publicaciones en esta área [8, 10, 21–24, 68, 88, 103, 148, 166]. Al mismo tiempo surgió un gran interés en las descomposiciones de las formas bilineales en el campo del álgebra compleja, por ejemplo tenemos el libro [89]. El ejemplo más interesante es la multiplicación Strassen de matrices, que es una aplicación de una descomposición

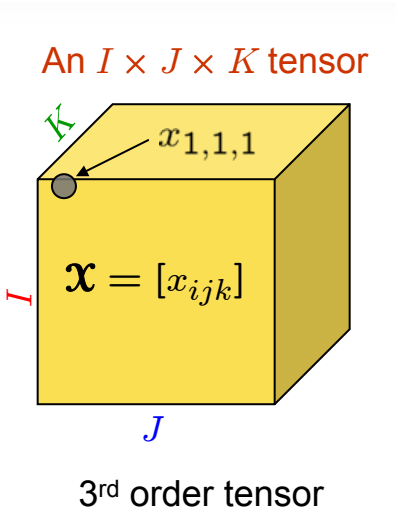


Figura 2.17: Tensor de tercer orden

$4 \times 4 \times 4$ de un tensor para describir multiplicaciones de matrices 2×2, ver por ejemplo [20, 98, 101, 150].

En los últimos diez años, el interés por la descomposición tensorial se ha expandido en otros campos. Por ejemplo, en el procesamiento de señales [41, 46, 48, 58, 145], álgebra lineal numérica [47, 63, 90, 102], visión artificial [6, 142, 161, 165], análisis numérico [18, 19, 66], minería de datos [1, 14, 92, 104, 142, 151], la psycometría [28, 158], la química [12], análisis de flujos turbulentos [17], análisis de imágenes y patrones de reconocimiento [161], análisis gráfico [14, 91, 92], neurociencia [111, 113, 122, 123] y muchos más.

Diferentes estudios han sido publicados en otros ámbitos, ver por ejemplo [21–23, 41, 45, 49, 97, 147] y recientemente podemos encontrar un libro referente a las múltiples formas de analizar datos [96]. Además podemos encontrar diferentes paquetes de software para poder trabajar con tensores, como por ejemplo [9, 100].

También hay un incremento en cuanto al uso del análisis numérico para la solución de problemas definidos en espacios tensoriales de dimensiones grandes, como por ejemplo las EDP que se utilizan en cálculos estocásticos [7, 25, 52] (por ejemplo, las ecuaciones de Fokker-Planck), EDP paramétricas estocásticas [50, 126, 127] y química [164].

2.1.4. Cálculo diferencial con Tensores.

Primero hay que introducir la notación utilizada en este capítulo relacionada con los tensores (ver [62] y [109] para más detalles). Denotamos el conjunto de $(n \times m)$ matrices para $\mathbb{R}^{n \times m}$, y la matriz transpuesta de una matriz A y se denota como A^T. Con $\langle \mathbf{x}, \mathbf{y} \rangle$ denotamos el producto escalar usual Euclídeo dado por: $\mathbf{x}^T \mathbf{y} = \mathbf{y}^T \mathbf{x}$ y se corresponde con con la 2-norma, $\|x\|_2 = \langle \mathbf{x}, \mathbf{x} \rangle^{1/2}$. La matriz I_n es la matriz identidad $(n \times n)$ y cuando

la dimensión queda clara con el contexto, la denotaremos como I. Vamos a definir el producto de Kronecker y algunas de las propiedades.

Definición 2. *El producto de Kronecker de $A \in \mathbb{R}^{n_1' \times n_1}$ y $B \in \mathbb{R}^{n_2' \times n_2}$, se denota como $A \otimes B$, es la operación tensor algebraica definida como*

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n_1'}B \\ a_{21}B & a_{22}B & \cdots & a_{2n_1'}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_1 1}B & a_{n_1 2}B & \cdots & a_{n_1 n_1'}B \end{bmatrix} \in \mathbb{R}^{n_1' n_2' \times n_1 n_2}.$$

También, el producto de Kronecker de dos matrices $A \in \mathbb{R}^{n_1' \times n_1}$ y $B \in \mathbb{R}^{n_2' \times n_2}$, puede definirse como $A \otimes B \in \mathbb{R}^{n_1' n_2' \times n_1 n_2}$, donde

$$(A \otimes B)_{(j_1-1)n_2'+j_2;(i_1-1)n_2+i_2} = A_{j_1;i_1} B_{j_2;i_2}. \tag{2.4}$$

Finalmente, algunas de las propiedades del producto de Kronecker (ver [62], [109] o [160]).

1. $A \otimes (B \otimes C) = (A \otimes B) \otimes C$.
2. $(A+B) \otimes (C+D) = (A \otimes C) + (B \otimes C) + (A \otimes D) + (B \otimes D)$.
3. Si $A+B$ y $C+D$ existe, $AB \otimes CD = (A \otimes C)(B \otimes D)$.
4. Si A y B son no-singulares, $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.
5. Si $(A \otimes B)^T = A^T \otimes B^T$.
6. Si A y B son diagonalmente por bandas, entonces $A \otimes B$ es diagonal por bandas.
7. Si A y B son simétricas, entonces $A \otimes B$ es simétrica.
8. Si A y B son definidas positivas, entonces $A \otimes B$ es definida positiva.

Definición 3. *Sea $A = [\mathbf{A}_1 \cdots \mathbf{A}_n]$ una matriz $m \times n$ donde $\mathbf{A}_j$ es la columna j-ésima. Entonces* $\mathrm{vec}A$ *es el vector $mn \times 1$*

$$\mathrm{vec}A = \begin{bmatrix} \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_n \end{bmatrix}.$$

*Por tanto, el operador **vec** transforma una matriz en un vector poniendo las columnas de la matriz una bajo de la otra. Observad que* $\mathrm{vec}A = \mathrm{vec}B$ *eso no implica $A = B$, a menos que A y B sean matrices del mismo orden.*

Las siguientes propiedades del operador vec$A$ y el producto de Kronecker nos serán útiles más adelante,

1. $\text{vec}\,\mathbf{u}^T = \text{vec}\,\mathbf{u} = \mathbf{u}$, para algún vector columna $\mathbf{u}$.
2. $\text{vec}\,\mathbf{u}\mathbf{v}^T = \mathbf{v} \otimes \mathbf{u}$, para cuales cualquiera dos vectores columna $\mathbf{u}$ y $\mathbf{v}$ (no necesariamente del mismo orden).
3. Sea A, B y C tres matrices tales que el producto ABC está definido. Entonces, $\text{vec}\,ABC = (C^T \otimes A)\text{vec}\,B$.

Definición 4. *Dada $F : \mathbb{R}^{n\times q} \longrightarrow \mathbb{R}^{m\times p}$ una función diferenciable. La matriz Jacobiana de F en X es la matriz de dimensión $mp \times nq$.*

$$DF(X) = \frac{\partial\,\text{vec}\,F(X)}{\partial\,(\text{vec}\,X)^T}.$$

Claramente, $DF(X)$ es una generalización de la tradicional definición de la matriz Jacobiana y todas las propiedades de las matrices Jacobianas se conservan. Por lo tanto, la siguiente definición reduce el estudio de las funciones de matrices para estudiar las funciones vectoriales de vectores, por lo tanto, eso permite $F(X)$ y X en su forma vectorizada vec$F$ y vec$X$.

1. Si $\mathbf{y} = f(X) = X\mathbf{u}$, tal que $\mathbf{u}$ es un vector de constantes, la matriz Jacobiana es $D(X\mathbf{u}) = \mathbf{u}^T \otimes I$.
2. $D(\mathbf{x}^T\mathbf{x}) = 2\mathbf{x}^T$

Teorema 1 (Regla de la Cadena)**.** *Sea S un subconjunto de $\mathbb{R}^{n\times q}$ y asumimos que $F : S \longrightarrow \mathbb{R}^{m\times p}$ es diferenciable en el punto interior C de S. Dado T un subcojunto de $\mathbb{R}^{m\times p}$ tal que $F(X) \in T$ para todo $X \in S$, y asumimos que $G : T \longrightarrow \mathbb{R}^{r\times s}$ es diferenciable en el punto interior $B = F(C) \in T$. Entonces la composición de funciones $H : S \longrightarrow \mathbb{R}^{r\times s}$ definida por $H(X) = G(F(X))$ es diferenciable en C, y*

$$DH(C) = (DG(B))(DF(C)).$$

2.1.5. Algoritmo BSD y T-BSD.

El algoritmo desarrollado en este capítulo, está basado en el planteamiento desarrollado en [167, 168]. Este algoritmo deforma una curva de Bézier a través de un campo de vectores. Esta técnica para deformar una curva de Bézier se ha formulado de dos formas diferentes, la primera se consigue utilizando una nomenclatura tradicional. En ese caso el algoritmo se ha denominado como: *Bézier Shape Deformation*, **BSD**. La segunda forma se ha logrado utilizando una formulación basada en tensores. El objetivo de utilizar los tensores no es más que la reducción del tiempo de cómputo al ejecutar el algoritmo. En este caso, el algoritmo recibe el nombre de: *Tensor-Bézier Shape Deformation*, **T-BSD**.

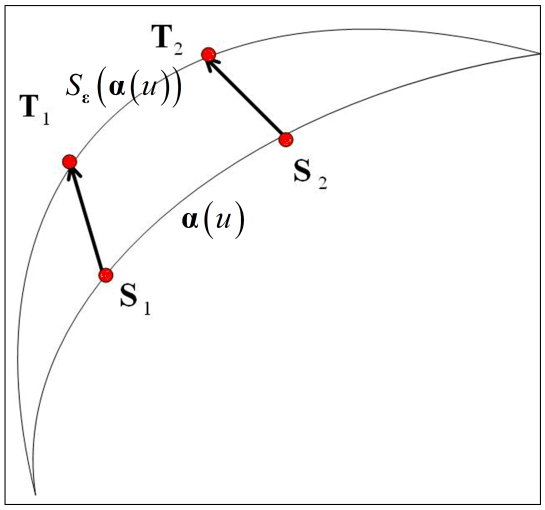


Figura 2.18: Deformación de una curva de Bézier mediante vectores.

Los resultados de esta investigación han sido publicados en [73, 74].

Este capítulo está organizado de la siguiente forma: en la sección 2.2 se desarrolla el algoritmo **BSD**; para luego en la sección 2.3 desarrollar el algoritmo **T-BSD**; en la sección 2.4 se realiza una comparativa entre el algoritmo **BSD** y **T-BSD**. En la sección 2.5 se destacan las propiedades más destacadas de las curvas de Bézier de forma que justifican su elección a la hora de desarrollar el algoritmo **BSD**; Finalmente, el capítulo se termina con la sección 2.6 donde se detallan las conclusiones.

2.2. Bézier Shape Deformation con formulación tradicional: BSD.

El algoritmo *Bézier Shape Deformation*, **BSD**, deforma una curva de Bézier $\boldsymbol{\alpha}(u)$ mediante un conjunto de vectores. Esta deformación exige que la curva modificada pase a través de un conjunto nuevo de puntos, los llamados Puntos Finales (*Target Points*), $\mathbf{T}_i$. Los vectores se forman uniendo puntos de la curva original, los Puntos Iniciales (*Start Points*), $\mathbf{S}_i$, con los Puntos Objetivos o Puntos Finales. Gráficamente podemos ver estos vectores en la figura 2.18.

En este caso, el algoritmo está diseñado con una curva de Bézier de orden n en el plano, es decir, que $\boldsymbol{\alpha}(u) \in \mathbb{R}^2$. Dado un conjunto de Puntos Iniciales de la curva de Bézier original ($\mathbf{S} = [\mathbf{S}_1 = \boldsymbol{\alpha}(u_1), \cdots, \mathbf{S}_r = \boldsymbol{\alpha}(u_r)]$), el objetivo es conseguir que la curva deformada atraviese un conjunto nuevo de puntos que son los llamados Puntos Finales ($\mathbf{T} = [\mathbf{T}_1 = S_\varepsilon(\boldsymbol{\alpha}(u_1)), \cdots, \mathbf{T}_r = S_\varepsilon(\boldsymbol{\alpha}(u_r))]$). De esta forma, se define un conjunto de vectores que se generan a partir de la unión de los Puntos Iniciales y los Puntos Finales.

Para esto se precisa de la modificación geométrica de los puntos de control. Con tan sólo modificar un punto de control ya se produce un cambio global de la curva de Bézier.

En consecuencia, para poder desarrollar el algoritmo **BSD** es necesario calcular la perturbación que se precisa en cada punto de control, es decir, el desplazamiento necesario. El conjunto de desplazamientos los denotaremos de la siguiente forma $\underline{\underline{\varepsilon}} = [\varepsilon_0, \cdots, \varepsilon_n]$. Con este conjunto de perturbaciones o desplazamientos, definimos, a partir de la curva de Bézier original, la curva de Bézier modificada o deformada como sigue, ver la ecuación 2.5, la denotaremos como $S_\varepsilon(\boldsymbol{\alpha}(u))$.

$$S_\varepsilon(\boldsymbol{\alpha}(u)) := \sum_{i=0}^{n} (\mathbf{P}_i + \varepsilon_i) \cdot B_{i,n}(u)\,; u \in [0,1] \tag{2.5}$$

El cálculo del vector desplazamiento, $\underline{\underline{\varepsilon}}$, se basa en las publicaciones, [167, 168], vistas en la sección 2.1.2. Para calcular $\underline{\underline{\varepsilon}}$ se plantea un problema de optimización restringida y se resuelve aplicando el Teorema de los Multiplicadores de Lagrange, ver apéndice B Teorema 2. La función a minimizar se define como la distancia entre las dos curvas, la original $\boldsymbol{\alpha}(u)$ y la curva modificada $S_\varepsilon(\boldsymbol{\alpha}(u))$. El objetivo es poder minimizar los cambios que sufra la curva de Bézier modificada, consiguiendo de esa forma minimizar la energía utilizada por la curva para poder moverla desde $\boldsymbol{\alpha}(u)$ a $S_\varepsilon(\boldsymbol{\alpha}(u))$. La definición de la función a minimizar la encontramos en la ecuación 2.6,

$$\min_{\underline{\underline{\varepsilon}}} \int_0^1 \|S_\varepsilon(\boldsymbol{\alpha}(u)) - \boldsymbol{\alpha}(u)\|_2^2 \, du, \tag{2.6}$$

siendo $\|.\|_2$ la norma euclídea.

Desarrollando 2.6, se obtiene,

$$\min_{\underline{\underline{\varepsilon}}} \int_0^1 \| \sum_{i=0}^{n} \varepsilon_i \cdot B_{i,n}(u)\|_2^2 \, du = \min_{\underline{\underline{\varepsilon}}} \int_0^1 \left(\sum_{i=0}^{n} \varepsilon_i \cdot B_{i,n}(u) \right)^2 du \tag{2.7}$$

Las curvas de Bézier presentan una desventaja y es su inestabilidad numérica a medida que aumenta el orden de la curva. A partir de $n = 10$ hay problemas de estabilidad, ver[57]. Por ello en el algoritmo **BSD** es necesario concatenar una familia de curvas de Bézier para poder obtener de forma completa la curva resultante. El algoritmo está definido para cualquier orden de forma que en función del tipo de aplicación podemos introducir un orden u otro.

La necesidad de concatenar curvas fuerza a una nueva definición de la función objetivo, ver 2.8.

Consideramos una familia de k curvas, siendo $\boldsymbol{\alpha}$ la curva resultante de concatenar esta familia de curvas $[\boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_k]$.

$$\min_{\underline{\underline{\varepsilon}}^{(1)} \cdots \underline{\underline{\varepsilon}}^{(k)}} \sum_{l=1}^{k} \int_0^1 \|S_\varepsilon(\boldsymbol{\alpha}_l(u)) - \boldsymbol{\alpha}_l(u)\|_2^2 \, du \tag{2.8}$$

De forma análoga a la expresión 2.7, se desarrolla la definición de la nueva función a optimizar 2.8.

$$\min_{\underline{\underline{\varepsilon}}^{(1)}\cdots\,\underline{\underline{\varepsilon}}^{(k)}} \sum_{l=1}^{k} \int_0^1 \| \sum_{i=0}^{n_l} \varepsilon_i^{(l)} \cdot B_{i,n_l}(u) \|_2^2 \, du = \min_{\underline{\underline{\varepsilon}}^{(1)}\cdots\,\underline{\underline{\varepsilon}}^{(k)}} \sum_{l=1}^{k} \int_0^1 \left(\sum_{i=0}^{n_l} \varepsilon_i^{(l)} \cdot B_{i,n_l(u)} \right)^2 du, \quad (2.9)$$

cada $\underline{\underline{\varepsilon}}^{(l)}$ corresponde al vector desplazamiento de cada curva $\boldsymbol{\alpha}_l$.
El conjunto de restricciones necesario para este problema se construye como sigue,

1. La curva modificada o deformada tiene que pasar por los Puntos Finales, eso implica que para un determinado valor del parámetro intrínseco la imagen de la curva modificada es el propio Punto Final. Considerando que en cada curva $\boldsymbol{\alpha}_l$, donde $1 \leq l \leq k$, se van a desplazar una cantidad de puntos r_l siendo $r_l \leq n_l - 1$. Esta condición se expresa en la ecuación 2.10,

$$S_\varepsilon \left(\boldsymbol{\alpha}_l \left(u_1^{(l)} \right) \right) = T_1^{(l)}, \cdots, S_\varepsilon \left(\boldsymbol{\alpha}_l \left(u_{r_l}^{(l)} \right) \right) = T_{r_l}^{(l)} \quad (2.10)$$

2. Para evitar tener oscilaciones en los extremos de las curvas concatenadas es necesario imponer una restricción que mantiene la tangencia en el punto inicial y final entre la curva original y la curva modificada. La ecuación que definen esta restricción en el punto inicial se expresa en 2.11,

$$S_\varepsilon \left(\boldsymbol{\alpha}_1' \left(0^+ \right) \right) = \boldsymbol{\alpha}_1' \left(0^+ \right) \quad (2.11)$$

De forma análoga se exige la tangencia entre la última curva original y la última deformada en la ecuación 2.12,

$$S_\varepsilon \left(\boldsymbol{\alpha}_k' \left(1^- \right) \right) = \boldsymbol{\alpha}_k' \left(1^- \right) \quad (2.12)$$

Se desarrolla cada término para poder obtener la expresión de la restricción en términos de la perturbación, ε.

Los cálculos para la primera curva son los siguientes:

$$\boldsymbol{\alpha}_1'(0^+) = n_1 \cdot \left[\mathbf{P}_1^{(1)} - \mathbf{P}_0^{(1)} \right] \quad (2.13)$$

$$S_\varepsilon \left(\boldsymbol{\alpha}_1'(0^+) \right) = n_1 \cdot \left[\left(\mathbf{P}_1^{(1)} + \varepsilon_1^{(1)} \right) - \left(\mathbf{P}_0^{(1)} + \varepsilon_0^{(1)} \right) \right] \quad (2.14)$$

Igualando ambas expresiones se obtiene la siguiente ecuación:

$$S_\varepsilon \left(\boldsymbol{\alpha}_1' \left(0^+ \right) \right) = \boldsymbol{\alpha}_1' \left(1^- \right) \Longleftrightarrow n_1 \cdot \left(\varepsilon_1^{(1)} - \varepsilon_0^{(1)} \right) = 0 \quad (2.15)$$

En el caso de la última curva, es decir, la curva k-ésima, los cálculos son los siguientes:

$$\boldsymbol{\alpha}_k'(1^-) = n_k \cdot \left[\mathbf{P}_{n_k}^{(k)} - \mathbf{P}_{n_{k-1}}^{(k)}\right] \tag{2.16}$$

$$S_\varepsilon\left(\boldsymbol{\alpha}_k'(1^-)\right) = n_k \cdot \left[\left(\mathbf{P}_{n_k}^{(k)} + \varepsilon_{n_k}^{(k)}\right) - \left(\mathbf{P}_{n_k-1}^{(k)} + \varepsilon_{n_k-1}^{(k)}\right)\right] \tag{2.17}$$

Igualando ambas expresiones se obtiene,

$$S_\varepsilon\left(\boldsymbol{\alpha}_k'\left(1^-\right)\right) = \boldsymbol{\alpha}_k'\left(0^+\right) \Longleftrightarrow n_k \cdot \left(\varepsilon_{n_k}^{(k)} - \varepsilon_{n_k-1}^{(k)}\right) = 0 \tag{2.18}$$

3. Como consecuencia de la concatenación de k curvas de Bézier es necesario garantizar continuidad de orden cero en los puntos donde se concatenan las curvas, además de continuidad de orden uno (continuidad en la primera derivada o continuidad tangencial). El objetivo es tener una curva suave pese a que se hayan concatenado una familia de curvas.

   Para poder garantizar continuidad de clase C^0 exigimos la ecuación 2.19,

$$S_\varepsilon\left(\boldsymbol{\alpha}_1\left(1^-\right)\right) = S_\varepsilon\left(\boldsymbol{\alpha}_2\left(0^+\right)\right), \cdots, S_\varepsilon\left(\boldsymbol{\alpha}_{k-2}\left(1^-\right)\right) = S_\varepsilon\left(\boldsymbol{\alpha}_{k-1}\left(0^+\right)\right) \tag{2.19}$$

   Para expresar cada término de la ecuación 2.19 en términos de la perturbación, ε, se precisan de los siguientes valores,

$$S_\varepsilon\left(\boldsymbol{\alpha}_l\left(1^-\right)\right) = \mathbf{P}_{n_l}^{(l)} + \varepsilon_{n_l}^{(l)} \tag{2.20}$$

   y

$$S_\varepsilon\left(\boldsymbol{\alpha}_{l+1}\left(0^+\right)\right) = \mathbf{P}_0^{(l+1)} + \varepsilon_0^{(l+1)} \tag{2.21}$$

   De forma análoga se impone la continuidad de clase C^1 y para ello exigimos en cada unión de curvas las ecuaciones en 2.22,

$$S_\varepsilon\left(\boldsymbol{\alpha}_1'\left(1^-\right)\right) = S_\varepsilon\left(\boldsymbol{\alpha}_2'\left(0^+\right)\right), \cdots, S_\varepsilon\left(\boldsymbol{\alpha}_{k-2}'\left(1^-\right)\right) = S_\varepsilon\left(\boldsymbol{\alpha}_{k-1}'\left(0^+\right)\right) \tag{2.22}$$

   Cada ecuación de la expresión 2.22 se expresa en términos de la perturbación, ε, de la siguiente forma,

$$S_\varepsilon\left(\boldsymbol{\alpha}_l'\left(1^-\right)\right) = n_l \cdot \left[\left(\mathbf{P}_{n_l}^{(l)} - \mathbf{P}_{n_l-1}^{(l)}\right) + \left(\varepsilon_{n_l}^{(l)} - \varepsilon_{n_l-1}^{(l)}\right)\right] \tag{2.23}$$

   y

$$S_\varepsilon\left(\boldsymbol{\alpha}_{l+1}'\left(0^+\right)\right) = n_{l+1} \cdot \left[\left(\mathbf{P}_1^{(l+1)} - \mathbf{P}_0^{(l+1)}\right) + \left(\varepsilon_1^{(l+1)} - \varepsilon_0^{(l+1)}\right)\right] \tag{2.24}$$

4. Un inconveniente que presentan las curvas de Bézier son los lazos (“loops”) que aparecen en curvas de Bézier de orden superior o igual a tres. En la figura 2.19 podemos ver un ejemplo de la representación de un lazo.

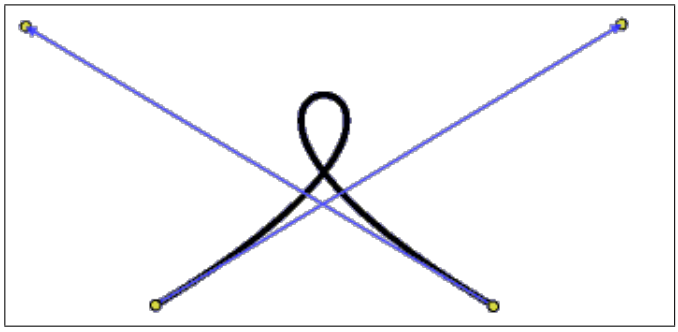


Figura 2.19: Representación de un lazo para un curva de Bézier de orden tres.

La formación de los lazos dependen del orden y posición de los puntos de control, [149]. El orden de los puntos de control puede generar arcos, puntos de inflexión, cúspides (“Cusp”), dos puntos de inflexión o lazos, observar figura 2.20. El cuarto punto de control condiciona cada uno de esos efectos.

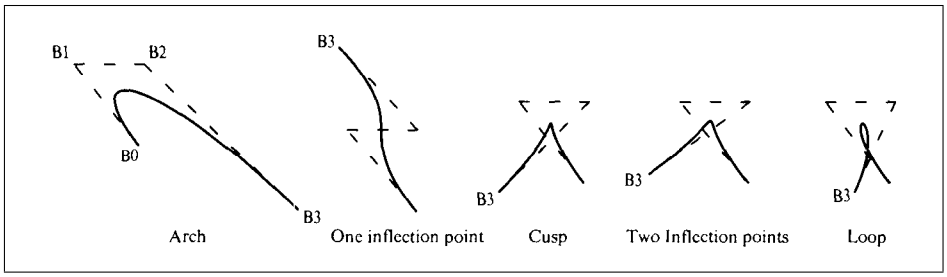


Figura 2.20: Curva de Bézier de orden tres mostrando un lazo, cúspide y puntos inflexiones.

Nuestro objetivo es evitar los lazos al deformar la unión de las curvas de Bézier. En la figura 2.21 el lazo está representado con un círculo azul.

Para poder solucionar este problema hay que utilizar curvas de Bézier que sean inyectivas, ver apéndice B definición 7. Tal y como podemos ver en la publicación [149] si se trabaja la posición del cuarto punto de control se pueden evitar estos efectos, incluso las cúspides y los dos puntos de inflexión que observamos en la figura 2.20. La posición de los puntos de control después de aplicar el **BSD** está condicionada a la dirección del vector que deformará la curva de Bézier.

Para conseguir la inyectividad de la curva de Bézier y evitar de esa forma este tipo de problemas, se han concatenado curvas de Bézier de orden dos. En la figura 2.22 se puede ver un ejemplo de una simulación del **BSD** con curvas de Bézier de orden

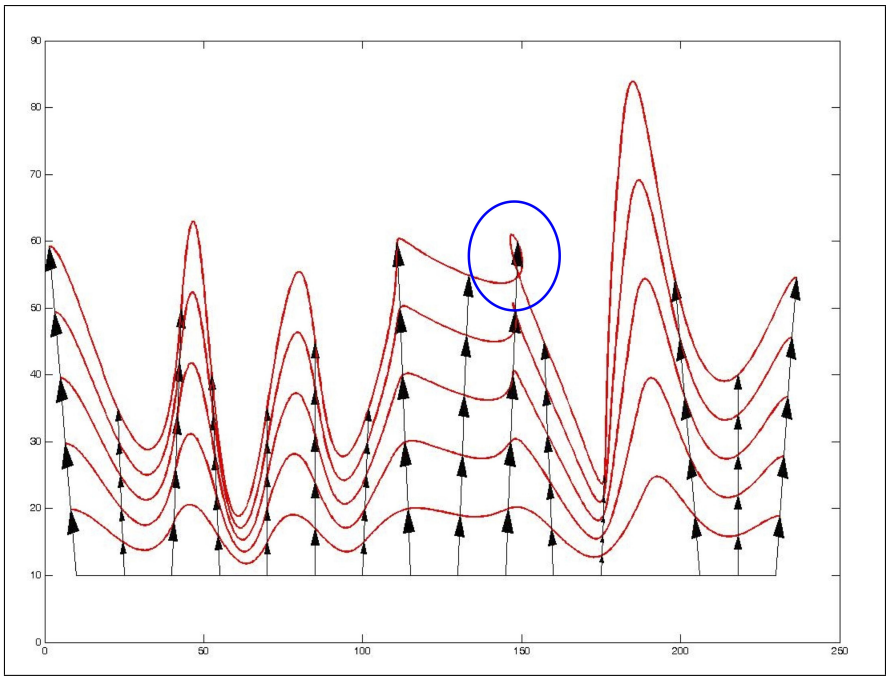


Figura 2.21: Simulación del **BSD** concatenando curvas de Bézier de tercer orden con lazos.

dos. Como el orden de la curva de Bézier es cuadrático y el número de los Puntos Finales es $r_l \leq n_l - 1$, hay tantas curvas como vectores representados en la figura.

El número total de restricciones del problema es:

$$\sum_{l=1}^{k} r_l + 1 + 1 + (k-1) + (k-1) = \sum_{l=1}^{k} r_l + 2k \tag{2.25}$$

Empleando el Teorema de los Multiplicadores de Lagrange, ver Teorema 2 en el apéndice B, se obtiene la solución del problema.

Para ello construimos la función Lagrangiana $L(X, \lambda_i)$ de la siguiente forma,

$$L(X, \lambda_i) = f(X) - \sum_{i=1}^{m} \lambda_i g_i(X) \tag{2.26}$$

Los extremos relativos (máximos o mínimos) se obtienen con el cálculo de los puntos estacionarios de la función Lagrangiana, ver 2.26.

En el caso del **BSD**, para poder obtener las perturbaciones necesarias en el movimiento de los puntos de control se define la función Lagrangiana en la ecuación 2.27 a partir de la función a optimizar definida en la ecuación 2.9 y el conjunto de restricciones descritas en las ecuaciones 2.10, 2.11, 2.12, 2.19 y 2.22.

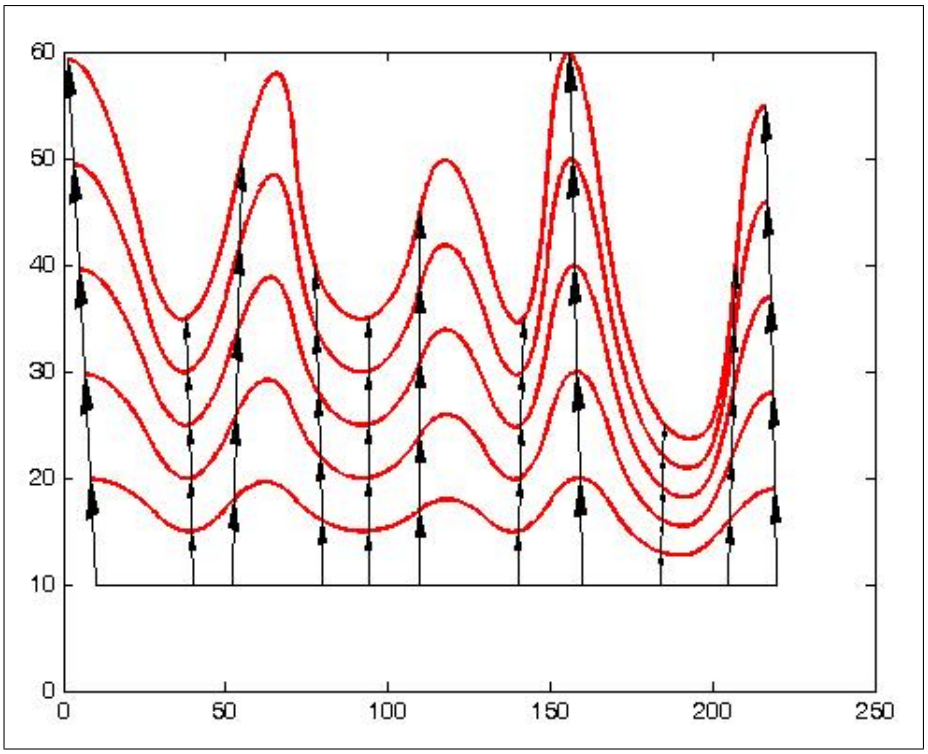


Figura 2.22: Simulación del **BSD** con curvas de Bézier concatenadas de segundo orden.

$$L(\varepsilon_i,\boldsymbol{\lambda}) = \sum_{l=1}^{k}\int_0^1 \| \sum_{i=0}^{n_l} \varepsilon_i^{(l)} \cdot B_{i,n_l}(u)\|_2^2 du + \sum_{l=1}^{k}\left[\sum_{j=1}^{r_l} < \boldsymbol{\lambda}, T_j^{(l)} - S_\varepsilon\left(\boldsymbol{\alpha}_l\left(u_j^{(l)}\right)\right) >\right] +$$
$$+ < \boldsymbol{\lambda}, \boldsymbol{\alpha}_1'\left(0^+\right) - S_\varepsilon\left(\boldsymbol{\alpha}_1'\left(0^+\right)\right) > + < \boldsymbol{\lambda}, \boldsymbol{\alpha}_k'\left(1^-\right) - S_\varepsilon\left(\boldsymbol{\alpha}_k'\left(1^-\right)\right) > +$$
$$\sum_{l=1}^{k-1} < \boldsymbol{\lambda}, S_\varepsilon\left(\boldsymbol{\alpha}_l\left(1^-\right)\right) - S_\varepsilon\left(\boldsymbol{\alpha}_{l+1}\left(0^+\right)\right) > + \sum_{l=1}^{k-1} < \boldsymbol{\lambda}, S_\varepsilon\left(\boldsymbol{\alpha}_l'\left(1^-\right)\right) - S_\varepsilon\left(\boldsymbol{\alpha}_{l+1}'\left(0^+\right)\right) >, \tag{2.27}$$

siendo $< .,. >$ el producto escalar euclídeo, ver apéndice B definición 8. Con el producto escalar se define la norma-2, ver apéndice B definición 9.

Además de la definición del producto escalar se va a utilizar también la del operador vec definido en 3 para reescribir la función a optimizar de la función Lagrangiana con el objetivo de facilitar el cálculo de las derivadas parciales,

Con las definiciones 8, 9 y 3, podemos realizar los siguientes desarrollos,

$$\| \sum_{i=0}^{n_l} \varepsilon_i^{(l)} \cdot B_{i,n_l}(u)\|_2^2 = \left(\sum_{i=0}^{n_l} \varepsilon_i^{(l)} \cdot B_{i,n_l}(u)\right)^T \cdot \left(\sum_{i=0}^{n_l} \varepsilon_i^{(l)} \cdot B_{i,n_l}(u)\right) =$$
$$= \sum_{i=0}^{n}\sum_{j=0}^{n} \left(\varepsilon_i^{(l)}\right)^T \varepsilon_j^{(l)} B_{i,n_l}(u) B_{j,n_l}(u) = \operatorname{vec}\left(\underline{\underline{\varepsilon}}^{(l)}\right)^T \mathbf{B}_{n_l}(u)\mathbf{B}_{n_l}(u)^T \operatorname{vec}\left(\underline{\underline{\varepsilon}}^{(l)}\right), \tag{2.28}$$

donde

$$\mathbf{B}_{n_l}(u)^T = \left[B_{0,n_l} \cdots B_{n_l,n_l}\right] \tag{2.29}$$

y

$$\underline{\underline{\varepsilon}}^{(l)} = \left[\varepsilon_0^{(l)} \cdots \varepsilon_{n_l}^{(l)}\right], \text{vec}\left(\underline{\underline{\varepsilon}}^{(l)}\right) = \begin{bmatrix} \varepsilon_0^{(l)} \\ \vdots \\ \varepsilon_{n_l}^{(l)} \end{bmatrix} \tag{2.30}$$

Consecuentemente, la función a optimizar se puede escribir de la siguiente forma,

$$\begin{aligned} L(\varepsilon_i, \boldsymbol{\lambda}) &= \sum_{l=1}^{k} \text{vec}\left(\underline{\underline{\varepsilon}}^{(l)}\right)^T \left(\int_0^1 \mathbf{B}_{n_l}\mathbf{B}_{n_l}^T du\right) \text{vec}\left(\underline{\underline{\varepsilon}}^{(l)}\right) + \\ &+ \sum_{l=1}^{k}\left[\sum_{j=1}^{r_l} < \lambda, T_j^{(l)} - S_\varepsilon\left(\boldsymbol{\alpha}_l\left(u_j^{(l)}\right)\right) >\right] + \\ &+ < \boldsymbol{\lambda}, \boldsymbol{\alpha}_1'(0) - S_\varepsilon\left(\boldsymbol{\alpha}_1'(0)\right) > + < \boldsymbol{\lambda}, \boldsymbol{\alpha}_k'(1) - S_\varepsilon\left(\boldsymbol{\alpha}_k'(1)\right) > + \\ &+ \sum_{l=1}^{k-1} < \boldsymbol{\lambda}, S_\varepsilon\left(\boldsymbol{\alpha}_l(1)\right) - S_\varepsilon\left(\boldsymbol{\alpha}_{l+1}(0)\right) > + \sum_{l=1}^{k-1} < \boldsymbol{\lambda}, S_\varepsilon\left(\boldsymbol{\alpha}_l'(1)\right) - S_\varepsilon\left(\boldsymbol{\alpha}_{l+1}'(0)\right) > \end{aligned} \tag{2.31}$$

La función 2.31 depende de las perturbaciones $\underline{\underline{\varepsilon}}$ y de los Multiplicadores de Lagrange λ asociados a cada restricción, es decir, que dicha función depende de,

$$L = L\left(\text{vec}\left(\underline{\underline{\varepsilon}}^{(1)}\right), \cdots, \text{vec}\left(\underline{\underline{\varepsilon}}^{(k)}\right), \boldsymbol{\lambda}\right) \tag{2.32}$$

Para proceder al cálculo de los puntos estacionarios se calculan las siguientes derivadas parciales,

$$\begin{cases} \dfrac{\partial L}{\partial \text{vec}\left(\underline{\underline{\varepsilon}}^{(l)}\right)} = 0 \\ \dfrac{\partial L}{\partial \boldsymbol{\lambda}} = 0 \end{cases} \tag{2.33}$$

Como resultado de calcular y anular las derivadas parciales de la función Lagrangiana se obtiene un sistema de ecuaciones lineal $\mathbf{A} \cdot \mathbf{X} = \mathbf{b}$, siendo:

1. **A** la matriz del sistema, siendo una matriz cuadrada cuya dimensión es:

$$\left[\left(\sum_{i=1}^{k}(n_i + r_i)\right) + 3k\right] \times \left[\left(\sum_{i=1}^{k}(n_i + r_i)\right) + 3k\right]$$

Esta matriz está definida a bloques de la siguiente forma:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} & \mathbf{A}_{14} \\ \mathbf{A}_{21} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{31} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{41} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{2.34}$$

La definición de cada bloque es la siguiente:

a) **El Bloque** A_{11}.

1) **Corresponde a:** las derivadas parciales de la función a optimizar.
2) **Dimensión del bloque:** $\sum_{i=0}^{k}(n_i+k) \times \sum_{i=0}^{k}(n_i+k)$.
3) **Definición del bloque:**

$$\mathbf{A}_{11} = \begin{bmatrix} \mathbf{A}_{11}^{(1)} & 0 & \cdots & 0 \\ 0 & \mathbf{A}_{11}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_{11}^{(k)} \end{bmatrix}, \tag{2.35}$$

donde

$$\mathbf{A}_{11}^{(l)} = \int_0^1 \mathbf{B}_{n_l}(u)\mathbf{B}_{n_l}(u)^T\,du = \begin{bmatrix} f(0,0) & \cdots & f(0,n_l) \\ \vdots & \ddots & \vdots \\ f(n_l,0) & \cdots & f(n_l,n_l) \end{bmatrix} \in \mathbf{M}_{(n_l+1)\times(n_l+1)} \tag{2.36}$$

Hay que obtener el desarrollo de cada $f(i,j)$, ver [136]. El resultado de la integral del producto de las Bases de Bernstein queda en función de la función Beta, ver apéndice B definición 10.

De esta forma el desarrollo de los términos $f(i,j)$ es el siguiente:

$$\begin{aligned} f(i,j) = \int_0^1 B_{i,n_l}(u)B_{j,n_u}(u)\,du = \int_0^1 \binom{n_l}{i} \cdot \binom{n_l}{j} u^{i+j}(1-u)^{2n_l-(i+j)}\,du = \\ = \tbinom{n_l}{i} \cdot \tbinom{n_l}{j} \cdot \beta(i+j+1, 2n_l+1-(i+j)) \end{aligned} \tag{2.37}$$

b) **El bloque** $\mathbf{A}_{21}$**.**

1) **Corresponde a:** el conjunto de restricciones correspondientes a los Puntos Finales.
2) **Dimensión del bloque:** $\sum_{i=1}^{k} r_i \times \sum_{i=1}^{k}(n_i+k)$.

3) **Definición del bloque:**

$$\mathbf{A}_{21} = \begin{bmatrix} \mathbf{A}_{21}^{(1)} & 0 & \cdots & 0 \\ 0 & \mathbf{A}_{21}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_{11}^{(k)} \end{bmatrix}, \tag{2.38}$$

donde

$$\mathbf{A}_{21}^{(l)} = \begin{bmatrix} B_{0,n_l}(u_1^{(l)}) & \cdots & B_{n_l,n_l}(u_1^{(l)}) \\ \vdots & \ddots & \vdots \\ B_{0,n_l}(u_{r_l}^{l}) & \cdots & B_{n_l,n_l}(u_{r_l}^{(l)}) \end{bmatrix} \in \mathbf{M}_{r_l \times (n_l+1)} \tag{2.39}$$

c) **El bloque $\mathbf{A}_{31}$.**

1) **Corresponde a:** las restricciones que mantienen la tangencia en el punto inicial y final de las curva de Bézier concatenadas.

2) **Dimensión del bloque:** $2 \times \left(\sum_{i=1}^{k} (n_i + k) \right)$.

3) **Definición del bloque:**

$$\mathbf{A}_{31} = \begin{bmatrix} -n_1 & n_1 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cdots & 0 & -n_k & n_k \end{bmatrix} \tag{2.40}$$

d) **El bloque $\mathbf{A}_{41}$.**

1) **Corresponde a:** las restricciones correspondientes a mantener continuidad C^0 y C^1 en los puntos donde se concatenan las curvas de Bézier.

2) **Dimensión del bloque:** $(2k-2) \times \left(\sum_{i=1}^{k} (n_i + k) \right)$

3) **Definición del bloque:**

$$\mathbf{A}_{41} = \begin{bmatrix} \mathbf{A}_{41}^{(1)} & 0 & \cdots & 0 \\ 0 & \mathbf{A}_{41}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{41}^{(k-1)} \end{bmatrix}, \tag{2.41}$$

donde

$$\mathbf{A}_{41}^{(l)} = \begin{bmatrix} 0 & \cdots & 0 & 1 & -1 & 0 & \cdots & 0 \\ 0 & \cdots & n_{l-1} & n_{l-1} & -n_l & -n_l & \cdots 0 & \end{bmatrix} \in \mathbf{M}_{2 \times (n_l + n_{l+1} + 2)} \tag{2.42}$$

e) **El bloque $\mathbf{A}_{12}$.**

1) **Corresponde a:** las derivadas parciales asociadas a las restricciones referentes a los Puntos Finales.
2) **Dimensión del bloque:** $\left(\sum_{i=1}^{k}(n_i+k)\right)\times\left(\sum_{i=1}^{k} r_i\right)$.
3) **Definición del bloque:**

$$\mathbf{A}_{12}=\begin{bmatrix} \mathbf{A}_{12}^{(1)} & 0 & \cdots & 0 \\ 0 & \mathbf{A}_{12}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{12}^{(k)} \end{bmatrix}, \tag{2.43}$$

donde

$$\mathbf{A}_{12}^{(l)}=\begin{bmatrix} \frac{-1}{2}B_{0,n_l}(u_1^{(l)}) & \cdots & \frac{-1}{2}B_{0,n_l}(u_{r_l}^{(l)}) \\ \vdots & \ddots & \vdots \\ \frac{-1}{2}B_{n_l,n_l}(u_1^{(l)}) & \cdots & \frac{-1}{2}B_{n_l,n_l}(u_{r_l}^{(l)}) \end{bmatrix} \in \mathbf{M}_{(n_l+1)\times r_l} \tag{2.44}$$

f) **El bloque $\mathbf{A}_{13}$.**

1) **Corresponde a:** las derivadas parciales asociadas a las restricciones que mantienen la tangencia en el punto inicial y final de la unión de todas las curvas.
2) **Dimensión del bloque:** $\left(\sum_{i=1}^{k}(n_i+k)\right)\times 2$.
3) **Definición del bloque:**

$$\mathbf{A}_{13}^{T}=\begin{bmatrix} 0 & 0 & 0 & \cdots & \cdots & 0 & \frac{-n_k}{2} & \frac{-n_k}{2} \\ \frac{-n_1}{2} & \frac{n_1}{2} & 0 & \cdots & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{2.45}$$

g) **El bloque $\mathbf{A}_{14}$.**

1) **Corresponde a:** las derivadas parciales asociadas a las restricciones que imponen continuidad C^0 y C^1 en los puntos donde se concatenan las curvas.
2) **Dimensión del bloque:** $\left(\sum_{i=1}^{k}(n_i+k)\right)\times(2k-2)$.
3) **Definición del bloque:**

$$\mathbf{A}_{14}=\begin{bmatrix} \mathbf{A}_{14}^{(1)} & 0 & \cdots & 0 \\ 0 & \mathbf{A}_{14}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{14}^{(k-1)} \end{bmatrix}, \tag{2.46}$$

donde

$$(\mathbf{A}_{14}^{(l)})^T = \begin{bmatrix} 0 & \cdots & \frac{-n_l}{2} & \frac{n_l}{2} & \frac{n_{l+1}}{2} & \frac{-n_{l+1}}{2} & \cdots & 0 \\ 0 & \cdots & 0 & \frac{1}{2} & \frac{-1}{2} & 0 & \cdots & 0 \end{bmatrix} \in \mathbf{M}_{(n_l+1)\times 2} \tag{2.47}$$

h) El resto de bloques son bloques con elementos nulos y que se podrían agrupar en un único bloque definido por una matriz nula $\mathbf{0}$ cuya dimensión es: $\left(\sum_{i=1}^{k}(r_i+2k)\right) \times \left(\sum_{i=1}^{k}(r_i+2k)\right)$.

2. $\mathbf{X}$ es el vector de las incógnitas cuya dimensión es

$$\left[\left(\sum_{i=1}^{k}(n_i+r_i)\right)+3k\right] \times 1.$$

El vector de las incógnitas se expresa como:

$$\mathbf{X} = \begin{bmatrix} \text{vec}\left(\underline{\underline{\varepsilon}}^{(l)}\right) \\ \boldsymbol{\lambda} \end{bmatrix} \tag{2.48}$$

3. $\mathbf{b}$ es el término independiente que tiene la misma dimensión que el vector de incógnitas y su expresión es la siguiente.

$$\mathbf{b} = \begin{bmatrix} \mathbf{0} \\ \mathbf{v}^{(1)} \\ \vdots \\ \mathbf{v}^{(k)} \\ \mathbf{C} \end{bmatrix} \tag{2.49}$$

El vector $\mathbf{v}^{(l)}$ se define como,

$$\mathbf{v}^{(l)} = \begin{bmatrix} \mathbf{T}_1^{(l)} - \mathbf{S}_1^{(l)} \\ \vdots \\ \mathbf{T}_{r_l}^{(l)} - \mathbf{S}_{r_l}^{(l)} \end{bmatrix} \tag{2.50}$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{0,(1,2)} \\ \mathbf{C}_{1,(1,2)} \\ \vdots \\ \mathbf{C}_{0,((k-1),k} \\ \mathbf{C}_{1,((k-1),k} \end{bmatrix} \tag{2.51}$$

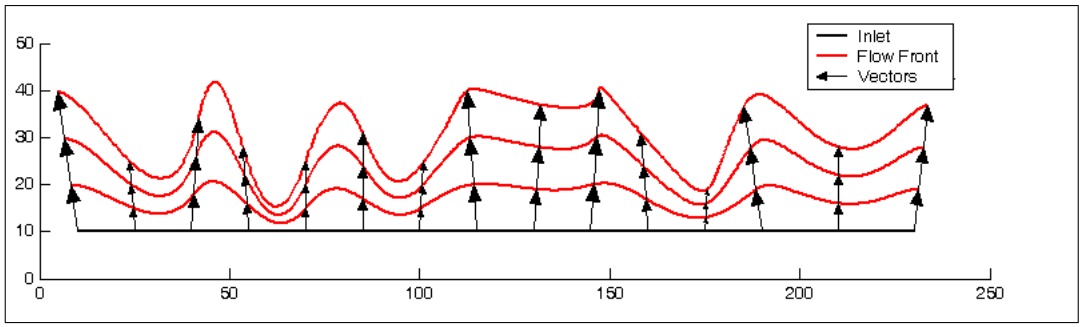
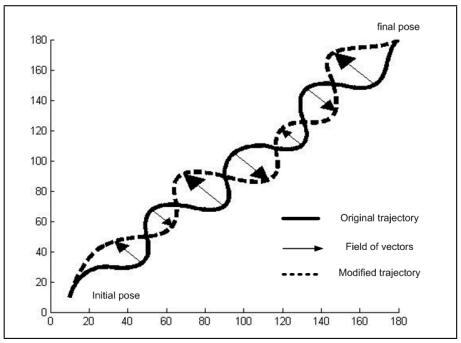
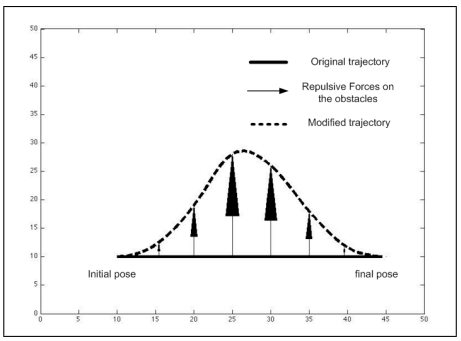


Figura 2.23: Ejemplo de la deformación de una curva de Bézier que puede representar el frente de avance de la resina en un llenado LCM.



(a) Ejemplo 1 **BSD** concatenando ocho curvas de Bézier.



(b) Ejemplo 2 **BSD** concatenando ocho curvas de Bézier.

Figura 2.24: Dos ejemplos diferentes al aplicar el **BSD** y utilizar en ambos ocho curvas de Bézier. En este caso, puede representar la trayectoria de un robot móvil.

$$\begin{aligned} \mathbf{C}_{0,((l-1),l} &= \mathbf{P}_0^{(l)} - \mathbf{P}_{n_l}^{(l-1)} \\ \mathbf{C}_{1,((l-1),l)} &= n_{l-1} \cdot \left(\mathbf{P}_{n_{l-1}-1}^{(l-1)} - \mathbf{P}_{n_l}^{(l-1)} \right) + n_l \cdot \left(\mathbf{P}_1^{(l)} - \mathbf{P}_0^{(l)} \right) \end{aligned} \tag{2.52}$$

Si la matriz es invertible, entonces la solución del sistema se obtiene como $\mathbf{X} = \mathbf{A}^{-1} \cdot \mathbf{b}$. Con la solución del sistema se obtiene la perturbación de cada punto de control y con ello la deformación de la curva de Bézier.

En las figuras 2.23, 2.24 podemos ver un ejemplo del algoritmo **BSD**. En estas figuras podemos ver la deformación del conjunto de curvas de Bézier concatenadas utilizando un conjunto de vectores. En cada caso la curva de Bézier y el vector tendrá un sentido físico diferente. En los capítulos 3 y 4 desarrollaremos las aplicaciones.

2.3. Bézier Shape Deformation basado en tensores: T-BSD.

El algoritmo desarrollado en la sección 2.2 se formula de nuevo, pero ahora se utiliza una notación basada en tensores. Uno de los objetivos principales de cambiar la formula-

ción es la reducción del tiempo de cómputo.

Con los conceptos vistos en 2.1.4 veamos como desarrollar el algoritmo **T-BSD**.

2.3.1. Formulación matemática.

Observación 1. *Basándonos en la definición vista en 1, redefinimos una curva de Bézier de la siguiente forma:*

$$\boldsymbol{\alpha}_t^n(u) = \sum_{i=0}^{n} \mathbf{P}_i(t) B_{i,n}(u); \quad u \in [0,1] \tag{2.53}$$

donde $\mathbf{P}_i(t)$ *son los puntos de control de la curva de Bézier* $\boldsymbol{\alpha}_t^n(u)$ *y* $B_{i,n}(u)$ *son los polinomios de Bernstein de grado n tal y como habíamos definido en 1.*

Aprovechando la reformulación del **BSD** con álgebra tensorial vamos a definir la curva de Bézier en cualquier espacio de dimensión d, es decir, $\mathbb{R}^d$. En principio el algoritmo está desarrollado para el caso particular $d = 2$ y la función a optimizar necesaria para resolver este problema está definida en ese caso concreto. Pero es cierto que en un futuro queda abierta la posibilidad de llevar el algoritmo a tres dimensiones. Así que en principio en este caso el valor de d siempre es dos.

La expresión de la curva de Bézier 2.53, la podemos escribir de forma equivalente en forma de matriz como sigue,

$$\boldsymbol{\alpha}_t^n(u) = P_n(t)\,\mathbf{B}_n(u); u \in [0,1], \tag{2.54}$$

donde

$$P_n(t) = \begin{bmatrix} \mathbf{P}_n^0(t) & \cdots & \mathbf{P}_n^n(t) \end{bmatrix} \in \mathbb{R}^{d\times(n+1)} \tag{2.55}$$

los puntos de control pueden estar en cualquier espacio de dimensión d.

$$\mathbf{B}_n(u) = \begin{bmatrix} B_{0,n}(u) & \cdots & B_{n,n}(u) \end{bmatrix}^T \in \mathbb{R}^{(n+1)\times 1}. \tag{2.56}$$

Utilizando la norma euclídea se escribe de la siguiente forma,

$$\|\boldsymbol{\alpha}_t^n(u)\|_p^p = \mathbf{B}_n(u)^T\, P_n(t)^T\, P_n(t)\, \mathbf{B}_n(u) \tag{2.57}$$

Para un valor fijo t, la energía de la curva u-parametrizada $\boldsymbol{\alpha}_t^n$ en $L^p([0,1],\mathbb{R}^d)$ viene dada por,

$$\|\boldsymbol{\alpha}_t^n\|_{\Delta_p} = \left(\int_0^1 \|\boldsymbol{\alpha}_t^n(u)\|_p^p\, du\right)^{1/p} = \left(\int_0^1 \mathbf{B}_n(u)^T\, P_n(t)^T\, P_n(t)\, \mathbf{B}_n(u)\, du\right)^{1/p}. \tag{2.58}$$

Ahora, consideramos un conjunto finito de Puntos Finales $\left\{\mathbf{T}_r^0, \ldots, \mathbf{T}_r^r\right\} \subset D$, siendo D un conjunto compacto y conexo en $\mathbb{R}^d$. Lo que pretendemos hacer es mover la curva inicial de Bézier, denotada por $\boldsymbol{\alpha}_t^n$ y caracterizado por el conjunto de los puntos de control

$P_n(t)$, a la curva, denotada por $\boldsymbol{\alpha}^n_{t+\Delta t}$ a través de un conjunto de perturbaciones para cada punto de control, es decir,

$$X_n = \left[\begin{array}{ccc} \mathbf{X}^0_n & \cdots & \mathbf{X}^n_n \end{array} \right] \in \mathbb{R}^{d\times(n+1)}. \tag{2.59}$$

La curva de Bézier resultante $\boldsymbol{\alpha}^n_{t+\Delta t}$ está dada por,

$$\boldsymbol{\alpha}^n_{t+\Delta t}(u) = (P_n(t) + X_n)\,\mathbf{B}_n(u); \quad u \in [0,1]. \tag{2.60}$$

Para calcular X_n se sigue el siguiente principio: queremos minimizar la energía utilizada por la curva para moverse desde $\boldsymbol{\alpha}^n_t$ a $\boldsymbol{\alpha}^n_{t+\Delta t}$. Además, hay que exigir que la curva transformada o modificada pase a través de los Puntos Finales para un conjunto de valores del parámetro intrínseco dado $0 = u^r_1 < u^r_2 < \cdots < u^r_{r-1} < u^r_r = 1$, eso equivale a,

$$\begin{aligned} &\text{mín } \|\boldsymbol{\alpha}^n_{t+\Delta t} - \boldsymbol{\alpha}^n_t\|^p_{\Delta_p} \\ &\text{s. t. } \boldsymbol{\alpha}^n_{t+\Delta t}(u^r_j) = \mathbf{T}^j_r; \text{ para} 1 \le j \le r, r \le n-1. \end{aligned} \tag{2.61}$$

Ahora, reescribimos (2.61) el problema en forma matricial de la siguiente forma. Dado el siguiente conjunto,

$$T_r = \left[\begin{array}{ccc} \mathbf{T}^1_r & \cdots & \mathbf{T}^r_r \end{array} \right] \in \mathbb{R}^{d\times r}. \tag{2.62}$$

y

$$B^r_n = \left[\begin{array}{ccc} \mathbf{B}_n(u^r_1) & \cdots & \mathbf{B}_n(u^r_r) \end{array} \right] \in \mathbb{R}^{(n+1)\times r} \tag{2.63}$$

Finalmente, consideramos la siguiente función matricial

$$\Phi_n(X_n) = \int_0^1 \mathbf{B}_n(u)^T X_n^T X_n \mathbf{B}_n(u)\,du, \tag{2.64}$$

entonces (2.61) podemos escribir en forma matricial tal y como sigue:

$$\begin{aligned} &\text{mín}_{X_n \in \mathbb{R}^{d\times(n+1)}}\, \Phi_n(X_n) \\ &\text{s. t. } (P_n(t) + X_n) B^r_n = T_r \end{aligned} \tag{2.65}$$

Si utilizamos el operador vec en (2.65) se obtiene el siguiente problema de minimización equivalente,

$$\begin{aligned} &\text{mín}_{(\text{vec}\, X_n) \in \mathbb{R}^{d\cdot(n+1)\times 1}}\, \Phi_n(\text{vec}\, X_n) \\ &\text{s. t. } ((B^r_n)^T \otimes I_d)\,\text{vec}\, X_n = \text{vec}\, T_r - \text{vec}\,(P_n(t) B^r_n) \end{aligned} \tag{2.66}$$

Observación 2. *Téngase en cuenta que el conjunto de restricciones de este problema 2.66 es lineal. En consecuencia, la aplicación Φ_n se define en un conjunto convexo. Por lo tanto, provando la convexidad de Φ_n, (ver definición 11 en el apéndice B) cada punto estacionario que se obtenga de (2.66) será un* ***mínimo absoluto****. La justificación la encontramos en el teorema 4 del apéndice B.*

Proposición 1. *Se cumplen los siguientes resultados:*

1.

$$D\Phi_n(X_n) = 2\int_0^1 (X_n \mathbf{B}_n(u))^T (\mathbf{B}_n(u)^T \otimes I_d)\, du, \in \mathbb{R}^{1\times d(n+1)}. \tag{2.67}$$

2.

$$D^2\Phi_n(X_n) = 2\int_0^1 (\mathbf{B}_n(u)\mathbf{B}_n(u)^T \otimes I_d)\, du = 2\left(\int_0^1 \mathbf{B}_n(u)\mathbf{B}_n(u)^T\, du\right) \otimes I_d. \tag{2.68}$$

Además, $D^2\Phi_n(X_n) \in \mathbb{R}^{d(n+1)\times d(n+1)}$ es una matriz simétrica definida positiva.

Demostración 1. *Primero, observamos que*

$$D\Phi_n(X_n) = \int_0^1 D\left(\mathbf{B}_n(u)^T X_n^T X_n \mathbf{B}_n(u)\right) du. \tag{2.69}$$

Consideramos $\mathbf{y}_n = F(X_n) = X_n\mathbf{B}_n(u)$ y $G(\mathbf{y_n}) = \mathbf{y}_n^T\mathbf{y}_n$. Entonces,
$DF(X_n) = \mathbf{B}_n(u)^T \otimes I_d$ and $DG(\mathbf{y_n}) = 2\mathbf{y}_n^T$. Por lo tanto, utilizando el Teorema 1 se obtiene que

$$D\left(\mathbf{B}_n(u)^T X_n^T X_n \mathbf{B}_n(u)\right) = 2\mathbf{y}_n^T\left(\mathbf{B}_n(u)^T \otimes I_d\right) = 2\left(X_n\mathbf{B}_n(u)\right)^T\left(\mathbf{B}_n(u)^T \otimes I_d\right), \tag{2.70}$$

y así queda probada la prosición 1. Para probar el enunciado 2 observamos que

$$D^2\left(\mathbf{B}_n(u)^T X_n^T X_n \mathbf{B}_n(u)\right) = 2\left(\mathbf{B}_n(u)^T \otimes I_d\right)^T\left(\mathbf{B}_n(u)^T \otimes I_d\right). \tag{2.71}$$

Como $(\mathbf{B}_n(u)\mathbf{B}_n(u)^T \otimes I_d)$ es definida positiva, obtenemos que $D^2\Phi_n(X_n)$ es también una definida positiva para todo $X_n \in \mathbb{R}^{d\times(n+1)}$.

Observación 3. *Con la proposición 1 y el teorema 3 (ver apéndice B) queda probada la convexidad de la función Φ_n.*

2.3.2. Bézier Shape Deformation basado en tensores concatenando Béziers.

Ahora, consideremos que la curva $\boldsymbol{\alpha}_t$ está descrita por un conjunto finito de curvas Bézier concatenadas $\boldsymbol{\alpha}_t^{n_1}, \ldots, \boldsymbol{\alpha}_t^{n_k}$ de grados $n_1, \ldots, n_k$, respectivamente. Utilizando esta notación cada curva se puede expresar como sigue

$$\boldsymbol{\alpha}_t^{n_i}(u) = P_{n_i}(t)\,\mathbf{B}_{n_i}(u); \quad u \in [0,1]; \quad 1 \le i \le k \tag{2.72}$$

donde,

$$P_{n_i}(t) = \left[\begin{array}{ccc} \mathbf{P}_{n_i}^0(t) & \cdots & \mathbf{P}_{n_i}^{n_i}(t) \end{array}\right] \in \mathbb{R}^{d\times(n_i+1)}. \tag{2.73}$$

y

$$\mathbf{B}_{n_i}(u) = \left[\begin{array}{ccc} B_{0,n_i}(u) & \cdots & B_{n_i,n_i}(u) \end{array}\right]^T \in \mathbb{R}^{(n_i+1)\times 1}. \tag{2.74}$$

La continuidad de $\boldsymbol{\alpha}_t$ implica que

$$\mathbf{P}_{n_i}^{n_i}(t) = \mathbf{P}_{n_{i+1}}^{0}(t) \tag{2.75}$$

para $1 \leq i \leq k-1$. Por tanto, si denotamos como

$$\boldsymbol{\alpha}_{t+\Delta t}^{n_i}(u) = (P_{n_i}(t) + X_{n_i})\,\mathbf{B}_{n_i}(u); \quad u \in [0,1] \tag{2.76}$$

donde,

$$X_{n_i} = \left[\begin{array}{ccc} \mathbf{X}_{n_i}^{0} & \cdots & \mathbf{X}_{n_i}^{n_i} \end{array}\right] \in \mathbb{R}^{d\times(n_i+1)}. \tag{2.77}$$

La continuidad de $\boldsymbol{\alpha}_{t+\Delta t}$ implica

$$\mathbf{X}_{n_i}^{n_i} = \mathbf{X}_{n_{i+1}}^{0}, \tag{2.78}$$

para $1 \leq i \leq k-1$. Asumimos que $\boldsymbol{\alpha}_t^{n_1}(0), \boldsymbol{\alpha}_t^{n_k}(1)$ pertenece a la frontera de Ω, denotado por $\partial\Omega$, y que

$$\frac{d}{du}\,\boldsymbol{\alpha}_t^{n_1}(u)\big|_{u=0^+} = \mathbf{V}_0(t), \quad \frac{d}{du}\,\boldsymbol{\alpha}_t^{n_k}(u)\big|_{u=1^-} = \mathbf{V}_k(t), \tag{2.79}$$

para todos los valores de t. La condición anterior implica que,

$$n_1(\mathbf{P}_{n_1}^{1}(t) - \mathbf{P}_{n_1}^{0}(t)) = \mathbf{V}_0(t), \quad n_k(\mathbf{P}_{n_k}^{n_k}(t) - \mathbf{P}_{n_k}^{n_{k-1}}(t)) = \mathbf{V}_k(t). \tag{2.80}$$

Por lo tanto, para $t+\Delta t$ este hecho implica que,

$$n_1(\mathbf{X}_{n_1}^{1} - \mathbf{X}_{n_1}^{0}) = \mathbf{V}_0(t+\Delta t) - \mathbf{V}_0(t), \tag{2.81}$$

$$n_k(\mathbf{X}_{n_k}^{n_k} - \mathbf{X}_{n_k}^{n_{k-1}}) = \mathbf{V}_k(t+\Delta t) - \mathbf{V}_k(t). \tag{2.82}$$

En este caso, asumimos

$$\mathbf{V}_0(t+\Delta t) - \mathbf{V}_0(t) = 0, \tag{2.83}$$

$$\mathbf{V}_k(t+\Delta t) - \mathbf{V}_k(t) = 0. \tag{2.84}$$

Ahora, introducimos la condición de diferenciabilidad para la curva global $\boldsymbol{\alpha}_t$. Puede escribirse como

$$\frac{d}{du}\,\boldsymbol{\alpha}_t^{n_i}(u)\big|_{u=1^-} = \frac{d}{du}\,\boldsymbol{\alpha}_t^{n_{i+1}}(u)\big|_{u=0^+}. \tag{2.85}$$

Por lo tanto,

$$n_i(\mathbf{P}_{n_i}^{n_i}(t) - \mathbf{P}_{n_i}^{n_i-1}(t)) = n_{i+1}(\mathbf{P}_{n_{i+1}}^{1}(t) - \mathbf{P}_{n_{i+1}}^{0}(t)), \tag{2.86}$$

ver ecuación 2.13 y 2.16 y para $t+\Delta t$ obtenemos,

$$n_i(\mathbf{X}_{n_i}^{n_i} - \mathbf{X}_{n_i}^{n_i-1}) = n_{i+1}(\mathbf{X}_{n_{i+1}}^{1} - \mathbf{X}_{n_{i+1}}^{0}) \tag{2.87}$$

para $1 \leq i \leq k-1$.
Dado,

$$T_{r_i} = \left[\begin{array}{ccc} \mathbf{T}_{r_i}^1 & \cdots & \mathbf{T}_{r_i}^{r_i} \end{array} \right] \in \mathbb{R}^{d \times r_i}, \tag{2.88}$$

y

$$B_{n_i}^{r_i} = \left[\begin{array}{ccc} \mathbf{B}_{n_i}(u_1^{r_i}) & \cdots & \mathbf{B}_{n_i}(u_{r_i}^{r_i}) \end{array} \right] \in \mathbb{R}^{(n_i+1)\times r_i}. \tag{2.89}$$

Hay que calcular $X_{n_i} \in \mathbb{R}^{d\times(n_i+1)}$ para $1 \leq i \leq k$ de forma que sastisfaga

$$\begin{array}{rl} \text{mín}_{(X_{n_1},\ldots,X_{n_k})} & \Phi(X_{n_1},\ldots,X_{n_k}) = \sum_{i=1}^k \Phi_{n_i}(X_{n_i}) \\ \text{s. t.} \quad (P_{n_i}(t)+X_{n_i})B_{n_i}^{r_i} & = T_{r_i}\, 1 \leq i \leq k, \\ \mathbf{X}_{n_i}^{n_i} & = \mathbf{X}_{n_{i+1}}^0,\, 1 \leq i \leq k-1, \\ n_1(\mathbf{X}_{n_1}^1 - \mathbf{X}_{n_1}^0) & = \mathbf{V}_0(t+\Delta t) - \mathbf{V}_0(t) = 0, \\ n_k(\mathbf{X}_{n_k}^{n_k} - \mathbf{X}_{n_k}^{n_k-1}) & = \mathbf{V}_k(t+\Delta t) - \mathbf{V}_k(t) = 0, \\ n_i(\mathbf{X}_{n_i}^{n_i} - \mathbf{X}_{n_i}^{n_i-1}) & = n_{i+1}(\mathbf{X}_{n_{i+1}}^1 - \mathbf{X}_{n_{i+1}}^0),\, 1 \leq i \leq k-1, \end{array} \tag{2.90}$$

Claramente,

$$D\Phi(X_{n_1},\ldots,X_{n_k}) = \left[\begin{array}{ccc} D\Phi_{n_1}(X_{n_1}) & \cdots & D\Phi_{n_k}(X_{n_k}) \end{array} \right] \in \mathbb{R}^{1\times d\sum_{i=1}^k (n_i+1)}. \tag{2.91}$$

Por tanto, $D^2\Phi(X_{n_1},\ldots,X_{n_k})$ es la matriz diagonal por bloques:

$$\left[\begin{array}{ccccc} D^2\Phi_{n_1}(X_{n_1}) & 0 & \cdots & 0 & 0 \\ 0 & D^2\Phi_{n_2}(X_{n_2}) & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & D^2\Phi_{n_{k-1}}(X_{n_{k-1}}) & 0 \\ 0 & 0 & \cdots & 0 & D^2\Phi_{n_k}(X_{n_k}) \end{array} \right]. \tag{2.92}$$

Utilizando la proposición 1, vemos que $D^2\Phi(X_{n_1},\ldots,X_{n_k})$ es una matriz definida positiva.
Ahora, se trata de reformular (2.90) con una notación más compacta. Para ello es necesario definir las siguientes matrices bloque. Para $1 \leq i \leq k$ definimos

$$R_{n_i} = \left[\begin{array}{ccccc} 0 & \cdots & 0 & I_d \end{array} \right] \in \mathbb{R}^{d\times d(n_i+1)}, \tag{2.93}$$

$$R_{n_i}^* = \left[\begin{array}{ccccc} 0 & \cdots & 0 & -I_d & I_d \end{array} \right] \in \mathbb{R}^{d\times d(n_i+1)}, \tag{2.94}$$

$$L_{n_i} = \left[\begin{array}{ccccc} I_d & 0 & \cdots & 0 \end{array} \right] \in \mathbb{R}^{d\times d(n_i+1)} \tag{2.95}$$

y

$$L_{n_i}^* = \left[\begin{array}{ccccc} -I_d & I_d & 0 & \cdots & 0 \end{array} \right] \in \mathbb{R}^{d\times d(n_i+1)}. \tag{2.96}$$

Finalmente, denotamos como

$$0_{n_i} = \left[\begin{array}{ccccc} 0 & 0 & 0 & \cdots & 0 \end{array} \right] \in \mathbb{R}^{d\times d(n_i+1)}, \tag{2.97}$$

en todo esta sección y para todas las matrices, cuando escribimos 0 denota la matriz cuadrada nula de tamaño $d \times d$.

$$\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \tag{2.98}$$

Gracias a la definición de estas matrices y al operador vec el conjunto de restricciones (2.90) se escribe como sigue,

$$\begin{array}{lcl} ((B_{n_i}^{r_i})^T \otimes I_d)\mathrm{vec}\, X_{n_i} & = & \mathrm{vec}\, T_{r_i} - \mathrm{vec}\,(P_{n_i}(t)B_{n_i}^{r_i}),\ 1 \leq i \leq k, \\ R_{n_i}\mathrm{vec}\, X_{n_i} & = & L_{n_{i+1}}\mathrm{vec}\, X_{n_{i+1}},\ 1 \leq i \leq k-1, \\ n_1 L_{n_1}^* \mathrm{vec}\, X_{n_1} & = & \mathbf{V}_0(t+\Delta t) - \mathbf{V}_0(t) = 0, \\ n_k R_{n_k}^* \mathrm{vec}\, X_{n_k} & = & \mathbf{V}_k(t+\Delta t) - \mathbf{V}_k(t) = 0, \\ n_i R_{n_i}^* \mathrm{vec} X_{n_i} & = & n_{i+1} L_{n_{i+1}}^* \mathrm{vec}\, X_{n_{i+1}},\ 1 \leq i \leq k-1, \end{array} \tag{2.99}$$

Entonces la función de Lagrange asociada a (2.90) se reescribe como,

$$\begin{aligned} \mathscr{L} \;=\; & \sum_{i=1}^{k} \Phi_{n_i}(\mathrm{vec}\, X_{n_i}) \\ & - \sum_{i=1}^{k} (\boldsymbol{\lambda}_i^{r_i})^T \left[((B_{n_i}^{r_i})^T \otimes I_d)\mathrm{vec}\, X_{n_i} - \mathrm{vec}\, T_{r_i} + \mathrm{vec}\,(P_{n_i}(t)B_{n_i}^{r_i}) \right] \\ & - \sum_{i=1}^{k-1} \boldsymbol{\mu}_i^T \left[R_{n_i}\mathrm{vec}\, X_{n_i} - L_{n_{i+1}}\mathrm{vec}\, X_{n_{i+1}} \right] \\ & - \boldsymbol{\mu}_k^T \left[n_1 L_{n_1}^* \mathrm{vec}\, X_{n_1} - \mathbf{V}_0(t+\Delta t) + \mathbf{V}_0(t) \right] \\ & - \boldsymbol{\mu}_{k+1}^T \left[n_k R_{n_k}^* \mathrm{vec}\, X_{n_k} - \mathbf{V}_k(t+\Delta t) + \mathbf{V}_k(t) \right] \\ & - \sum_{i=1}^{k-1} \boldsymbol{\mu}_{i+1+k}^T \left[n_i R_{n_i}^* \mathrm{vec} X_{n_i} - n_{i+1} L_{n_{i+1}}^* \mathrm{vec}\, X_{n_{i+1}} \right] \end{aligned} \tag{2.100}$$

Tal que,

$$\boldsymbol{\lambda}_i^{r_i} = \begin{bmatrix} \lambda_i^1 \\ \vdots \\ \lambda_i^{dr_i} \end{bmatrix} \in \mathbb{R}^{dr_i \times 1}. \tag{2.101}$$

y

$$\boldsymbol{\mu_i} = \begin{bmatrix} \mu_i^1 \\ \cdots \\ \mu_i^d \end{bmatrix} \in \mathbb{R}^{d \times 1}. \tag{2.102}$$

Entonces, la condición de primer orden de optimalidad son (2.99) y

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial (\operatorname{vec} X_{n_1})^T} &= D\Phi_{n_1}(\operatorname{vec} X_{n_1}) - (\boldsymbol{\lambda}_1^{r_1})^T((B_{n_1}^{r_1})^T \otimes I_d) \\ &\quad -\boldsymbol{\mu}_1^T R_{n_1} - \boldsymbol{\mu}_k^T n_1 L_{n_1}^* - \boldsymbol{\mu}_{k+2}^T n_1 R_{n_1}^* = 0,\end{aligned} \tag{2.103}$$

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial (\operatorname{vec} X_{n_i})^T} &= D\Phi_{n_i}(\operatorname{vec} X_{n_i}) - (\boldsymbol{\lambda}_i^{r_i})^T((B_{n_i}^{r_i})^T \otimes I_d) \\ &\quad -\boldsymbol{\mu}_i^T R_{n_i} + \boldsymbol{\mu}_{i-1}^T L_{n_i} - \boldsymbol{\mu}_{k+1+i}^T n_i R_{n_i}^* \\ &\quad +\boldsymbol{\mu}_{k+i}^T n_i L_{n_i}^* = 0,\end{aligned} \tag{2.104}$$

para $2 \leq i \leq k-1$, y

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial (\operatorname{vec} X_{n_k})^T} &= D\Phi_{n_k}(\operatorname{vec} X_{n_k}) - (\boldsymbol{\lambda}_k^{r_k})^T((B_{n_k}^{r_k})^T \otimes I_d) \\ &\quad +\boldsymbol{\mu}_{k-1}^T L_{n_k} - \boldsymbol{\mu}_{k+1}^T n_k R_{n_k}^* + \boldsymbol{\mu}_{2k}^T n_k L_{n_k}^* = 0,\end{aligned} \tag{2.105}$$

Utilizando el hecho de que,

$$(D\Phi_n(X_n))^T = 2\int_0^1 (\mathbf{B}_n(u) \otimes I_d)(X_n \mathbf{B}_n(u))\, du \in \mathbb{R}^{d(n+1)\times 1} \tag{2.106}$$

y cogiendo el operador vec se obtiene,

$$(D\Phi_n(\operatorname{vec} X_n))^T = 2\left(\int_0^1 (\mathbf{B}_n(u)^T \otimes \mathbf{B}_n(u) \otimes I_d)\, du\right) \operatorname{vec} X_n. \tag{2.107}$$

Por lo tanto, las condiciones de primer orden de optimalidad con respecto a $\operatorname{vec} X_{n_i}$-variables se pueden escribir como,

$$\begin{aligned}0 &= 2\left(\int_0^1 (\mathbf{B}_{n_1}(u)^T \otimes \mathbf{B}_{n_1}(u) \otimes I_d)\, du\right) \operatorname{vec} X_{n_1} - (B_{n_1}^{r_1} \otimes I_d)\boldsymbol{\lambda}_1^{r_1} \\ &\quad -R_{n_1}^T \boldsymbol{\mu}_1 - n_1 (L_{n_1}^*)^T \boldsymbol{\mu}_k - n_1 (R_{n_1}^*)^T \boldsymbol{\mu}_{k+2},\end{aligned} \tag{2.108}$$

$$\begin{aligned}0 &= 2\left(\int_0^1 (\mathbf{B}_{n_i}(u)^T \otimes \mathbf{B}_{n_i}(u) \otimes I_d)\, du\right) \operatorname{vec} X_{n_i} - (B_{n_i}^{r_i} \otimes I_d)\boldsymbol{\lambda}_i^{r_i} \\ &\quad +L_{n_i}^T \boldsymbol{\mu}_{i-1} - R_{n_i}^T \boldsymbol{\mu}_i + n_i (L_{n_i}^*)^T \boldsymbol{\mu}_{k+i} - n_i (R_{n_i}^*)^T \boldsymbol{\mu}_{k+i+1},\end{aligned} \tag{2.109}$$

para $2 \leq i \leq k-1$, y

$$\begin{aligned}0 &= 2\left(\int_0^1 (\mathbf{B}_{n_k}(u)^T \otimes \mathbf{B}_{n_k}(u) \otimes I_d)\, du\right) \operatorname{vec} X_{n_k} - (B_{n_k}^{r_k} \otimes I_d)\boldsymbol{\lambda}_k^{r_k} \\ &\quad +L_{n_k}^T \boldsymbol{\mu}_{k-1} - n_k (R_{n_k}^*)^T \boldsymbol{\mu}_{k+1} + n_k (L_{n_k}^*)^T \boldsymbol{\mu}_{2k},\end{aligned} \tag{2.110}$$

Finalmente, podemos concluir que para calcular la solución del problema de minimización (2.90) hay que resolver el siguiente sistema de ecuaciones lineales,

$$A\mathbf{z} = \mathbf{f} \tag{2.111}$$

La matriz A se define como,

$$\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} & A_{1,5} \\ A_{2,1} & 0 & 0 & 0 & 0 \\ A_{3,1} & 0 & 0 & 0 & 0 \\ A_{4,1} & 0 & 0 & 0 & 0 \\ A_{5,1} & 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{w\times w} \tag{2.112}$$

Observación 4. *La dimensión de los ceros que aparecen en esta matriz es coherente con la dimensión de cada bloque $A_{i,j}$.*

Donde,

$$w = d\sum_{i=1}^{k}(n_i+1) + d\sum_{i=1}^{k} r_i + d(k-1) + 2d + d(k-1) = d\sum_{i=1}^{k}(n_i+r_i) + 3dk. \tag{2.113}$$

La dimensión de cada bloque es,

$$A_{1,1} \in \mathbb{R}^{d\sum_{i=1}^{k}(n_i+1)\times d\sum_{i=1}^{k}(n_i+1)} \tag{2.114}$$

$$A_{1,2} \in \mathbb{R}^{d\sum_{i=1}^{k}(n_i+1)\times d\sum_{i=1}^{k} r_i} \tag{2.115}$$

$$A_{1,3} \in \mathbb{R}^{d\sum_{i=1}^{k}(n_i+1)\times d(k-1)} \tag{2.116}$$

$$A_{1,4} \in \mathbb{R}^{d\sum_{i=1}^{k}(n_i+1)\times 2d} \tag{2.117}$$

$$A_{1,5} \in \mathbb{R}^{d\sum_{i=1}^{k}(n_i+1)\times d(k-1)} \tag{2.118}$$

$$A_{2,1} \in \mathbb{R}^{d\sum_{i=1}^{k} r_i\times d\sum_{i=1}^{k}(n_i+1)} \tag{2.119}$$

$$A_{3,1} \in \mathbb{R}^{d(k-1)\times d\sum_{i=1}^{k}(n_i+1)} \tag{2.120}$$

$$A_{4,1} \in \mathbb{R}^{2d\times d\sum_{i=1}^{k}(n_i+1)} \tag{2.121}$$

y

$$A_{5,1} \in \mathbb{R}^{d(k-1)\times d\sum_{i=1}^{k}(n_i+1)}. \tag{2.122}$$

Aquí,

$$\mathbf{z} = \begin{bmatrix} \operatorname{vec} X_{n_1} \\ \vdots \\ \operatorname{vec} X_{n_k} \\ \boldsymbol{\lambda}_1^{r_1} \\ \vdots \\ \boldsymbol{\lambda}_k^{r_k} \\ \boldsymbol{\mu}_1 \\ \vdots \\ \boldsymbol{\mu}_{2k} \end{bmatrix} \in \mathbb{R}^{w\times 1}. \tag{2.123}$$

La definición de **f** es:

$$\mathbf{f} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \operatorname{vec} T_{r_1} - \operatorname{vec} P_{n_1}(t) B_{n_1}^{r_1} \\ \vdots \\ \operatorname{vec} T_{r_k} - \operatorname{vec} P_{n_k}(t) B_{n_k}^{r_k} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{w \times 1}. \tag{2.124}$$

$$A_{1,1} = \operatorname{diag}(Z_{n_1}, \ldots, Z_{n_k}) \tag{2.125}$$

donde,

$$Z_{n_i} = \int_0^1 (\mathbf{B}_{n_i}(u)^T \otimes \mathbf{B}_{n_i}(u) \otimes I_d) du \in \mathbb{R}^{d(n_i+1) \times d(n_i+1)} \tag{2.126}$$

para $i = 1, 2, \ldots, k$,

$$A_{1,2} = \operatorname{diag}((-1/2) \cdot (B_{n_1}^{r_1} \otimes I_d), \ldots, (-1/2) \cdot (B_{n_k}^{r_k} \otimes I_d)) \tag{2.127}$$

y

$$A_{1,3} = \begin{bmatrix} \frac{-1}{2} R_{n_1}^T & 0_{n_1}^T & 0_{n_1}^T & 0_{n_1}^T & \cdots & 0_{n_1}^T & 0_{n_1}^T \\ \frac{1}{2} L_{n_2}^T & \frac{-1}{2} R_{n_2}^T & 0_{n_2}^T & 0_{n_2}^T & \cdots & 0_{n_2}^T & 0_{n_2}^T \\ 0_{n_3}^T & \frac{1}{2} L_{n_3}^T & \frac{-1}{2} R_{n_3}^T & 0_{n_3}^T & \cdots & 0_{n_3}^T & 0_{n_3}^T \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0_{n_{k-1}}^T & 0_{n_{k-1}}^T & 0_{n_{k-1}}^T & 0_{n_{k-1}}^T & \cdots & \frac{1}{2} L_{n_{k-1}}^T & \frac{-1}{2} R_{n_{k-1}}^T \\ 0_{n_k}^T & 0_{n_k}^T & 0_{n_k}^T & 0_{n_k}^T & \cdots & 0_{n_k}^T & \frac{1}{2} L_{n_k}^T \end{bmatrix} \tag{2.128}$$

$$A_{1,4} = \begin{bmatrix} \frac{-1}{2} n_1 (L_{n_1}^*)^T & 0_{n_1}^T \\ 0_{n_2}^T & 0_{n_2}^T \\ \vdots & \vdots \\ 0_{n_{k-1}}^T & 0_{n_{k-1}}^T \\ 0_{n_k}^T & \frac{-1}{2} n_k (R_{n_k}^*)^T \end{bmatrix} \tag{2.129}$$

$$A_{1,5} = \begin{bmatrix} \frac{-1}{2} n_1 (R_{n_1}^*)^T & 0_{n_1}^T & 0_{n_1}^T & \cdots & 0_{n_1}^T & 0_{n_1}^T & 0_{n_1}^T \\ \frac{1}{2} n_2 (L_{n_2}^*)^T & \frac{-1}{2} n_2 (R_{n_2}^*)^T & 0_{n_2}^T & \cdots & 0_{n_2}^T & 0_{n_2}^T & 0_{n_2}^T \\ 0_{n_3}^T & \frac{1}{2} n_3 (L_{n_3}^*)^T & \frac{-1}{2} n_3 (R_{n_3}^*)^T & \cdots & 0_{n_3}^T & 0_{n_3}^T & 0_{n_3}^T \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0_{n_{k-1}}^T & 0_{n_{k-1}}^T & 0_{n_{k-1}}^T & \cdots & 0_{n_{k-1}}^T & \frac{1}{2} n_{k-1} (L_{n_{k-1}}^*)^T & \frac{-1}{2} n_{k-1} (R_{n_{k-1}}^*)^T \\ 0_{n_k}^T & 0_{n_k}^T & 0_{n_k}^T & \cdots & 0_{n_k}^T & 0_{n_k}^T & \frac{1}{2} n_k (L_{n_k}^*)^T \end{bmatrix} \tag{2.130}$$

Por otra parte,

$$A_{2,1} = \operatorname{diag}((B_{n_1}^{r_1})^T \otimes I_d, \ldots, (B_{n_k}^{r_k})^T \otimes I_d) \tag{2.131}$$

$$A_{3,1} = \begin{bmatrix} R_{n_1} & -L_{n_2} & 0_{n_3} & \cdots & 0_{n_{k-2}} & 0_{n_{k-1}} & 0_{n_k} \\ 0_{n_1} & R_{n_2} & -L_{n_3} & \cdots & 0_{n_{k-2}} & 0_{n_{k-1}} & 0_{n_k} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0_{n_1} & 0_{n_2} & 0_{n_3} & \cdots & R_{n_{k-2}} & R_{n_{k-1}} & 0_{n_k} \\ 0_{n_1} & 0_{n_2} & 0_{n_3} & \cdots & 0_{n_k}-2 & R_{n_{k-1}} & -L_{n_k} \end{bmatrix} \tag{2.132}$$

$$A_{4,1} = \begin{bmatrix} n_1 L^*_{n_1} & 0_{n_2} & 0_{n_3} & \cdots & 0_{n_{k-1}} & 0_{n_k} \\ 0_{n_1} & 0_{n_2} & 0_{n_3} & \cdots & 0_{n_{k-1}} & n_k R^*_{n_k} \end{bmatrix} \tag{2.133}$$

y

$$A_{5,1} = \begin{bmatrix} n_1 R^*_{n_1} & -n_2 L^*_{n_2} & 0_{n_3} & \cdots & 0_{n_{k-2}} & 0_{n_{k-1}} & 0_{n_k} \\ 0_{n_1} & n_2 R^*_{n_2} & -n_3 L^*_{n_3} & \cdots & 0_{n_{k-2}} & 0_{n_{k-1}} & 0_{n_k} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0_{n_1} & 0_{n_2} & 0_{n_3} & \cdots & n_{k-2} R^*_{n_{k-2}} & -n_{k-1} L^*_{n_{k-1}} & 0_{n_k} \\ 0_{n_1} & 0_{n_2} & 0_{n_3} & \cdots & 0_{n_{k-2}} & n_{k-1} R^*_{n_{k-1}} & -n_k L^*_{n_k} \end{bmatrix} \tag{2.134}$$

2.4. Comparativa del BSD y T-BSD: tiempos de cómputo.

Con el algoritmo **T-BSD** se consigue una reducción del tiempo de cómputo considerable. La figura 2.25 muestra la evolución del tiempo de cómputo que se necesita para obtener la deformación de una familia de curvas de Bézier concatenadas. En este gráfico 2.25 se muestra una comparativa del tiempo de cómputo de los métodos **BSD** y **T-BSD** frente al número de curvas de Bézier concatenadas en el algoritmo. El tiempo de cómputo está directamente relacionado con el aumento del número de curvas. En el caso del algoritmo **BSD**, su crecimiento es exponencial, es decir, $O(n^f)$. Por el contrario, en el caso del algoritmo **T-BSD** el coste numérico también aumenta a medida que aumentan el número de curvas de Bézier, pero en este caso el crecimiento es lineal, es decir, $O(fn)$.

Por lo tanto y tal y como habíamos indicado ya en la introducción de este capítulo, el uso del álgebra tensorial reduce considerablemente el tiempo de cómputo. En el libro [64] podemos encontrar más información al respecto.

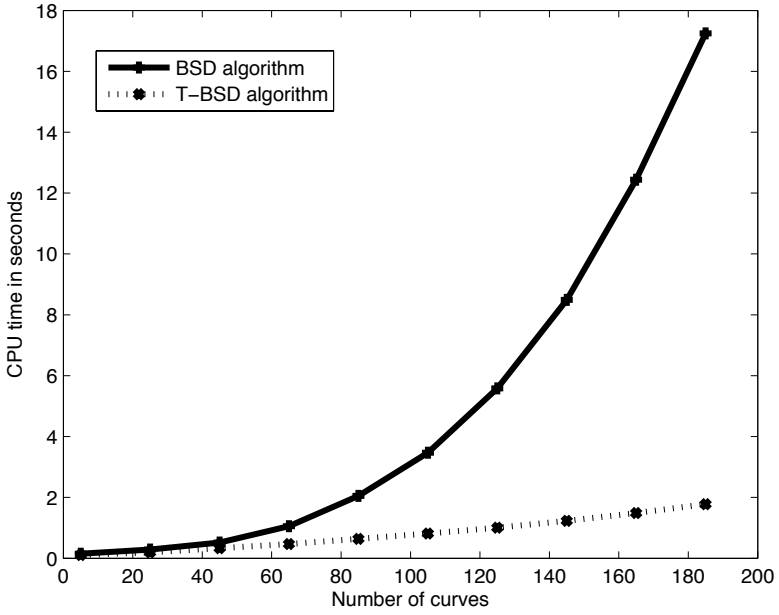


Figura 2.25: Comparativa del tiempo de cómputo entre el **BSD** y el **T-BSD**.

Esta comparación está calculada utilizando un PC 3,06 GHz Intel Core i3 y 4GB RAM.

La razón por la que se produce esta reducción tan drástica del tiempo de la CPU es la forma de almacenar la información de las matrices al utilizar el álgebra tensorial. Este almacenamiento provoca que se opere de forma diferente y eso hace que baje el número de operaciones necesarias para calcular la solución del sistema en este caso particular.

No es lo mismo resolver un sistema definido de esta forma $AX = b$ (donde A es una matriz cuadrada de tamaño $n^f \times n^f$ y X y b son vectores de tamaño $n^f \times 1$) que resolver un sistema cuya formulación está basada en tensores, de la siguiente forma:

$$(A^{(1)} \otimes \cdots \otimes A^{(f)})(x^{(1)} \otimes \cdots \otimes x^{(f)}) = A^{(1)}x^{(1)} \otimes \cdots \otimes A^{(f)}x^{(f)} = b^{(1)} \otimes \cdots \otimes b^{(f)}$$

Al formular el sistema utilizando el álgebra tensorial el número de operaciones se reduce porque se opera de forma distinta y con ello se reduce el coste numérico de los algoritmos.

2.5. Justificación de la elección de las curvas de Bézier para el algoritmo BSD.

Para desarrollar el **BSD** y el **T-BSD** se ha escogido una curva Bézier por la simplicidad de su expresión y sus numerosas propiedades. Es cierto que curvas como las B-Splines

y las NURBS proporcionan otro tipo de deformación, pero las bases que utilizan para definir la curva son recurrentes. Esto provoca un aumento del coste numérico además de la dificultad en cuanto al desarrollo de las operaciones. Veamos las propiedades más destacadas de las Bézier que justifican su elección para el **BSD** y el **T-BSD**.

2.5.1. Sentido físico de los puntos de control de las curvas de Bézier.

Un polinomio cualquiera de grado n se define de la siguiente forma:

$$f(u) = a_0 + a_1 u + a_2 u^2 + \cdots + a_n u^n; a_i \in \mathbb{R} \tag{2.135}$$

Con esta definición un polinomio de grado n viene expresado como una combinación de los elementos de la base canónica $< 1, u, \cdots, u^n >$. Los coeficientes a_i son las coordenadas del polinomio $f(u)$ en la base canónica y no tienen en principio una interpretación física como la que pueden tener los puntos de control de una curva de Bézier. Partiendo que los polinomios de grado n son un espacio vectorial de dimensión n y que los espacios vectoriales vienen generados por infinitas y diversas bases. En el caso de las curvas de Bézier también representan un polinomio de grado n, pero en este caso viene expresado como combinación lineal de los polinomios de Bernstein. Los "coeficientes" en este caso que posibilitan esa combinación lineal son los puntos de control. Los polinomios de las Bases de Bernstein surgen de la expansión del número 1. La expansión se desarrolla con la fórmula del Binomio de Newton,

$$\begin{aligned} &1^n = (1-u+u)^n = \\ &= \tbinom{n}{0}(1-u)^n u^0 + \tbinom{n}{1}(1-u)^{n-1}u + \cdots + \tbinom{n}{n-1}(1-u)^1 u^{n-1} + \tbinom{n}{n}(1-u)^0 u^n \end{aligned} \tag{2.136}$$

Cada elemento del sumatorio es un elemento de la Base de Bernstein que se define en la expresión 2.137

$$\{\binom{n}{0}(1-u)^n u^0, \binom{n}{1}(1-u)^{n-1}u^1, \cdots, \binom{n}{n}(1-u)^0 u^n\} \tag{2.137}$$

Cada elemento de la base de Bernstein de puede expresar como sigue:

$$B_{i,n}(u) = \binom{n}{i} u^i (1-u)^{n-i}; i = 0, \cdots, n \tag{2.138}$$

Con esta base Bernstein del espacio vectorial de los polinomios, se puede definir una curva como:

$$\boldsymbol{\alpha}(u) = \mathbf{P}_0 \cdot B_{o,n}(u) + \mathbf{P}_1 \cdot B_{1,n}(u) + \cdots + \mathbf{P}_n \cdot B_{n,n}(u) \tag{2.139}$$

Esta curva paramétrica que se acaba de definir, es la curva de Bézier. Los coeficientes del polinomio en la base de Bernstein coincide con los puntos de control necesarios para representar la curva de Bézier.

2.5.2. Interpolación del primer y último punto de control.

Una curva de Bézier interpola siempre el primer y último punto, es decir, que los dos únicos puntos que atraviesa son P_0 y P_n. A partir de su definición podemos comprobar que se cumple: $\boldsymbol{\alpha}(0) = \mathbf{P}_0$ y $\boldsymbol{\alpha}(1) = \mathbf{P}_n$. De esta forma, al aplicar el **BSD** tenemos controlado el primer y último punto de la curva de Bézier.

2.5.3. Afín invariante.

Una curva de Bézier es afín invariante. Eso significa que si aplicamos una transformación afín a una curva de Bézier, el resultado puede construirse desde las imágenes afines de los puntos de control. La transformación afín es:

$$\varphi(x) = A \cdot x + b; A \in \mathbb{R}^{n \times n}; b \in \mathbb{R}^n \tag{2.140}$$

Se cumple:

$$\begin{aligned} &\sum_{i=0}^{n} \varphi(\mathbf{P}_i) \cdot B_{i,n}(u) = \sum_{i=0}^{n} (A \cdot \mathbf{P}_i + b) \cdot B_{i,n}(u) = \\ &A \cdot \sum_{i=0}^{n} \mathbf{P}_i \cdot B_{i,n}(u) + b \cdot \sum_{i=0}^{n} B_{i,n}(u) = A \cdot \sum_{i=0}^{n} \mathbf{P}_i \cdot B_{i,n}(u) + b \cdot 1 = \varphi(\sum_{i=0}^{n} \mathbf{P}_i \cdot B_{i,n}(u)) \end{aligned} \tag{2.141}$$

La consecuencia de esta propiedad es importante para el algoritmo **BSD** porque nos indica que si queremos transformar una curva de Bézier debemos de transformar para ello los puntos de control.

2.5.4. Vectores tangentes a la curva de Bézier.

Dada una curva de Bézier de orden n hay dos vectores tangentes a la curva en el primer punto de control, $\mathbf{P}_0$, y en el último punto de control, $\mathbf{P}_n$. Los vectores tangentes a la curva son $\overrightarrow{\mathbf{P}_0\mathbf{P}_1}$ y $\overrightarrow{\mathbf{P}_{n-1}\mathbf{P}_n}$. En la figura 2.26 tenemos una curva de Bézier en la que se trazan los vectores tangentes a la curva.

Esta propiedad es relevante para el algoritmo **BSD** porque estudia si la unión entre dos curvas de Bézier se realiza con continuidad de clase C^1. Para que dos curvas de Bézier se unan de forma suave, entonces los vectores tangentes a la primera curva en el último punto y a la segunda curva en el primer punto son paralelos o tiene que estar alineados. En la figura 2.27 se pueden ver como dos curvas de Bézier de tercer orden se unen con tan sólo la condición de continuidad. En cambio, en la figura 2.28 al concatenar las dos curvas de Bézier su unión se realiza de forma suave porque es de clase C^1.

Esta propiedad también resulta interesante para el algoritmo **BSD** porque en él se necesita concatenar curvas de Bézier y así su unión genera una curva suave.

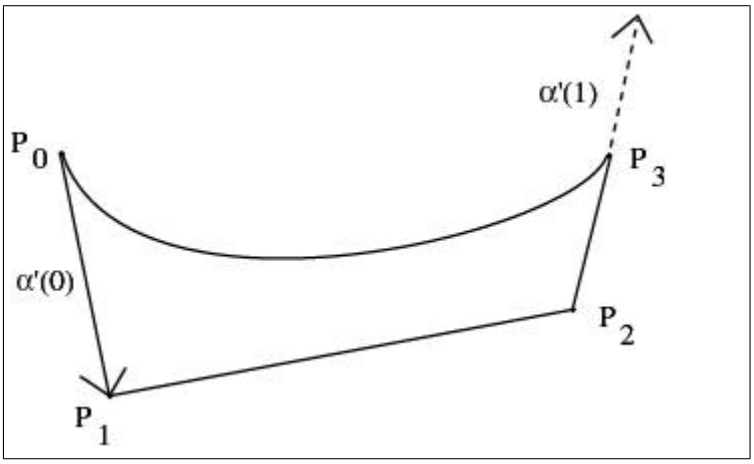


Figura 2.26: Vectores tangentes a una curva de Bézier de tercer orden.

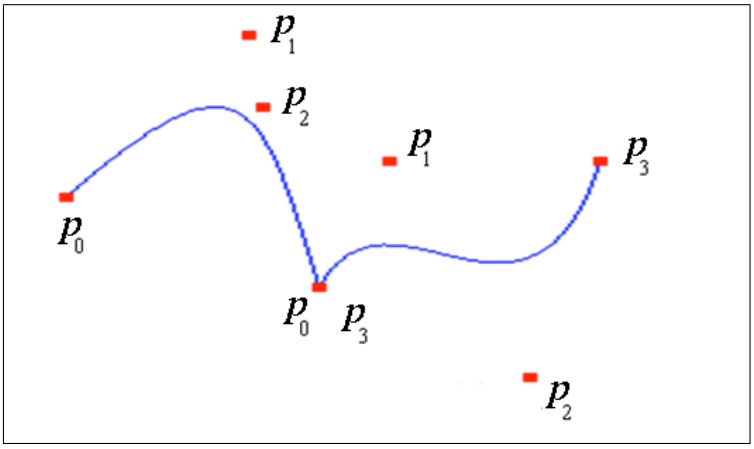


Figura 2.27: Unión de dos curvas de Bézier de clase C^0.

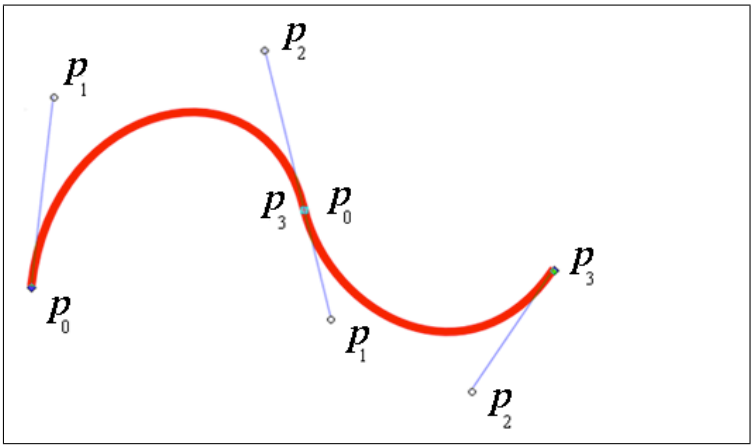


Figura 2.28: Unión de dos curvas de Bézier de clase C^1.

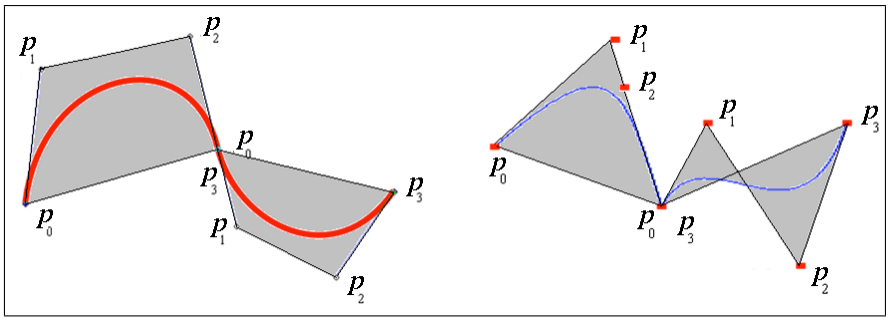


Figura 2.29: Envolvente convexa.

2.5.5. Envolvente convexa.

Una curva de Bézier está contenida en la envolvente convexa, ver apéndice B definición 6, definida por los puntos de control. Esta propiedad se cumple porque los polinomios de Bernstein son siempre positivos y la suma de todos ellos es igual a uno, ver ecuación 2.136.

Con esta propiedad tenemos controlada la región donde estará contenida la curva de Bézier. En la figura 2.29 podemos ver la envolvente convexa que encierra a las curvas de Bézier, en este caso está representada por la zona gris delimitada por las curvas de Bézier.

2.6. Conclusiones.

En este capítulo se ha realizado una introducción de las curvas haciendo hincapié en las curvas de Bézier que son aquellas en las hemos basado nuestro algoritmo. Además se ha dejado constancia de lo importante que es tener la capacidad de poder modificar la forma de las curvas paramétricas, siendo uno de los tópicos más investigados en este ámbito.

Son distintas y variadas las aplicaciones donde es necesario obtener la deformación de curvas o superficies paramétricas. En esta Tesis el objetivo de desarrollar una técnica o algoritmo que permita esta deformación es poderlo llevar al mundo de la ingeniería.

Además se ha dado una introducción a los tensores y sus aplicaciones más destacadas para luego poder utilizar una formulación en el algoritmo **BSD** basada en tensores.

Finalmente se ha desarrollado el algoritmo **BSD** que deforma las curvas de Bézier mediante vectores formulando un problema de optimización restringida. El problema se resuelve aplicando los Multiplicadores de Lagrange y como consecuencia se obtiene un sistema de ecuaciones lineal $AX = b$. El algoritmo se ha definido con dos formulaciones diferenciadas.

I. Con una notación tradicional para construir el sistema lineal: **BSD**.

II. Con una formulación basada en tensores: **T-BSD**.

La solución de ambos sistemas obviamente es la misma, pero al utilizar los tensores conseguimos ciertas ventajas que podemos destacar en los siguientes puntos:

1. Una reducción del tiempo de cómputo considerable.

   Con el uso de los tensores queríamos demostrar la reducción del tiempo de cómputo, algo que ya está publicado en [53, 64, 66]. En la figura 2.25 se observa en el **T-BSD** un comportamiento lineal del tiempo de cómputo frente a un comportamiento exponencial en el caso del **BSD**. Esta reducción justifica la eficiencia de la construcción de la matriz A del sistema de ecuaciones al basarse su construcción con tensores. Tanto en el **BSD** como en el **T-BSD** la matriz del sistema A tiene la misma dimensión, es por ello que el tiempo de cómputo se invierte en la construcción de la matriz y en la forma de operar luego para resolver el sistema lineal de ecuaciones. Al formular el algoritmo con los tensores se logra una construcción mucho más eficiente que conlleva una forma de operar en la que se invierte mucho menos tiempo provocando un menor coste numérico. En [62, 109] podemos encontrar más información acerca de los tensores.

2. Demostración de que la solución obtenida es un mínimo.

   Otra de las ventajas a destacar con el algoritmo **T-BSD** frente al **BSD** es la demostración de que el punto estacionario de la función Lagrangiana es un mínimo, es decir, que la única solución que se obtiene al resolver el sistema lineal de ecuaciones corresponde al mínimo buscado. Hay que recordar que se busca la mínima distancia entre las dos curvas de Bézier, la original y la modificada. La justificación de que el punto estacionario es un mínimo corresponde a la convexidad de la función a optimizar, ver 2.

3. Posibilidad de aumentar la dimensión de la curva de Bézier.

   Con la formulación basada en tensores se consigue una notación mucho más compacta que permite incluso aumentar la dimensión como trabajo futuro para poder deformar una curva de Bézier que esté en $\mathbb{R}^3$.

Capítulo 3

Aplicación del BSD en Procesos LCM: BFD y BFD-A.

Como es posible que la matemática, un producto del pensamiento humano independiente de la experiencia, se adapte tan admirablemente a los objetos de la realidad.

Albert Einstein (1879-1955)

3.1. Introducción.

El algoritmo **BSD** desarrollado en el capítulo 2 tiene una primera aplicación: la representación del frente de avance de la resina en los llenados de moldes con resina líquida. El algoritmo **BSD** se adapta para calcular el frente de avance (flow front) con una curva de Bézier. En este caso, el algoritmo se denomina *Bézier Flow Front Deformation*, **BFD**. Además de obtener el frente de avance mediante una curva continua, se actualiza en cada instante de tiempo. Los vectores velocidad del flujo de la resina son los que se proporcionan al **BFD** para deformar el frente representado con Béziers. Los resultados de esta investigación se han publicado en: [117].

Además en este capítulo veremos la opción de introducir una restricción nueva en el algoritmo para mejorar el cálculo del frente. Se trata de introducir el área que hay encerrada entre las dos curvas de Bézier como una restricción más. El valor del área debe ser igual al valor de la cantidad de resina introducida en un intervalo de tiempo determinado. En este caso, el algoritmo se denomina **BFD-A**. El resultado de esta investigación se ha publicado en [71].

El presente capítulo está organizado en diferentes secciones: en la sección 3.2 se hará una breve introducción a los procesos Liquid Composite Moulding (LCM); en la sección 3.3 también se realizará un breve resumen de las técnicas Finite Element Method (FEM) que más se utilizan en estos procesos para la simulación del llenado; en la sección 3.4 se va a detallar las ventajas del uso de las curvas paramétricas en esta aplicación con-

creta. En la sección 3.5 se detalla una técnica que hace evolucionar las partículas que es la que se fusionará con el **BFD**. Posteriormente, en la sección 3.6 se explicará como se adapta el algoritmo **BSD** al **BFD** y como se fusiona con las técnicas de elementos finitos y la técnica de evolución de partículas vista en 3.5. En la sección 3.7 se introduce una restricción nueva en el modelo **BFD**, generando el algoritmo **BFD-A**. Por último, en 3.8 se detallarán las conclusiones referentes a este capítulo.

3.2. Introducción a los procesos LCM.

La fabricación de piezas mediante el llenado de moldes se engloba en dos grandes grupos:

- *Procesos de fabricación con moldes abiertos*: Dentro de los procesos con moldes abiertos podemos diferenciar dos grandes técnicas, "Hand lay-up" y "Prepreg". En ambas, la resina se impregna en la preforma o en el refuerzo de forma manual. La diferencia está en que en la técnica del Hand lay-up, la impregnación se hace con rodillo y en el caso de la técnica Prepreg se hace con pistola. Son procesos utilizados en la industria caracterizados por ser sencillos de llevar a cabo puesto que no se precisa de una mano de obra especializada. Pero, es cierto que el gran incoveniente del proceso manual es la emisión continua de compuestos orgánicos volátices que en su mayoría son tóxicos.

- *Procesos de fabricación con moldes cerrados*: En este grupo entran todos aquellos procesos que cuentan con un molde y un contramolde para llevar a cabo la fabricación de la pieza. El molde siempre estará fabricado con materiales rígidos, en cambio el contramolde puede fabricarse con flexibles, semirígidos o rígidos. Hay varios métodos dentro de los procesos con moldes cerrados. Los métodos quizás más nuevos en cuanto a la fabricación reciben el nombre de LCM (Liquid Composite Molding). Dentro de los procesos LCM podemos estudiar diferentes técnicas separándolas en dos grupos:

  - aquellas técnicas que requieren presión positiva, como por ejemplo RTM (Resin Transfer Molding).
  - o derivados de éstas que serían técnicas que requieren presión negativa como por ejemplo VARTM (Vacuum-Assisted RTM) o VARI (Vacuum-Assisted Resin Infusion).

En la figura 3.1 se ilustran los pasos que se siguen en los procesos que requieren presión positiva, como el caso de RTM. El venteo está abierto a la atmósfera y la resina es inyectada con presión positiva, esto hace que tanto la abrazadera como el molde y el contramolde sean los adecuados ya que tienen que soportar la presión. En el caso de tener grandes piezas es un problema obtener una abrazadera adecuada, por ello hay técnicas alternativas en las que no se ejerce presión, sino que es un proceso de infusión por vacío.

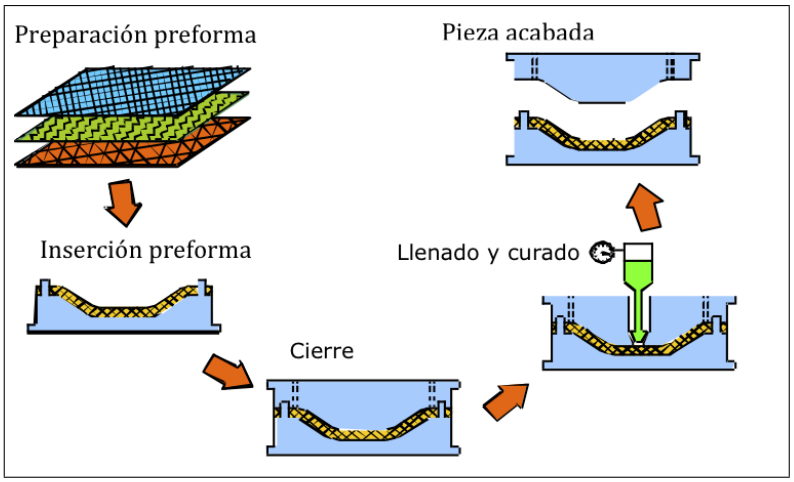


Figura 3.1: Fases del Proceso RTM

En la figura 3.2 se ilustran los procesos que requieren presión negativa. La diferencia entre esta técnica y la anterior es que la resina no es inyectada a presión, al hacer vacío se conduce la resina dentro de la lámina. Como se observa en la figura 3.2, la preforma se introduce en seco en el molde y el vacío se aplica antes de que se introduzca la resina. Cuando se consigue el vacío, la resina es aspirada en el laminado a través de ciertos tubos que se habrán colocado previamente. La presión negativa permite que la mitad superior del molde pueda estar hecha de un material más flexible y de esta forma se reducen los costes de fabricación.

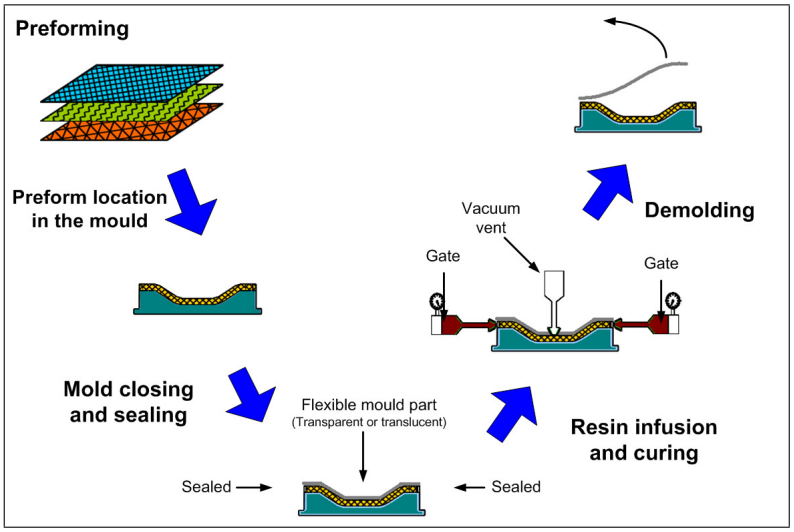


Figura 3.2: Fases del Proceso con presión negativa

El frente de avance de la resina se define como la frontera entre la zona seca del molde y la zona ya impregnada por la resina. En la figura 3.3 podemos ver un ejemplo real del frente de avance.

Figura 3.3: Ejemplo del frente de avance.

Es la herramienta más común que permite implementar una mejora desde diferentes aspectos: desde la optimización del llenado hasta la búsqueda de un diseño adecuado del molde para poder obtener la pieza deseada. Las aplicaciones de estos procesos son utilizadas en la industria aeronáutica, naval, automovilística, etc.

Un aspecto importante en la simulación del llenado de moldes en los procesos LCM es el tratamiento del frente de avance de la resina. Puesto que el tratamiento analítico de la evolución del flujo de la resina es difícil, a menos que la geometría sea sencilla, hay que recurrir al estudio numérico. De esta forma se estudia con un enfoque discreto, se obtiene una gran nube de puntos en forma de diente de sierra, ver figura 3.5.

Con el proceso de la simulación se pueden destacar dos inconvenientes:

I. El tipo de curva que se obtiene con forma de diente de sierra dista del frente de avance real.

II. El alto coste computacional que este proceso precisa.

Una de las técnicas utilizadas para la simulación numérica del frente de avance es la de elementos finitos, más conocida como FEM (Finite Element Method). Como consecuencia de la simulación se obtiene la posición del frente de avance en cada instante de tiempo y esto nos ayudará a hacer un estudio de la optimización tanto del llenado como del diseño del molde y del curado de la resina.

En [114, 138] podemos encontrar más información acerca de estos procesos.

3.3. Simulación por elementos finitos: Vectores velocidad.

En general, la impregnación de la resina en la fibra se describe utilizando el frente de avance a través de la teoría del medio poroso. El fenómeno de la presión impulsada por el flujo se describe con la Ley de Darcy:

$$\mathbf{v} = -\frac{\underline{\underline{k}}}{\eta} \cdot \nabla p \tag{3.1}$$

Siendo $\underline{\underline{k}}$ el tensor de la permeabilidad de las preformas, η es la viscosidad de la resina y p es la presión del fluido.

La Ley de Darcy determina los vectores velocidad que se calculan a partir del gradiente de presiones, la relación que hay entre la velocidad del flujo y el gradiente de presiones es lineal, tal y como se puede ver en la ecuación (3.1). El conocimiento adecuado de la permeabilidad de las telas y de la viscosidad de la resina permitirá una modelización correcta del flujo por esta ley. La gran mayoría de procesos de llenado de moldes en medios porosos se pueden considerar para cavidades de moldes de pequeño espesor y por tanto, el flujo de la resina puede ser simplificado como un problema bidimensional, es decir, sin considerar el trasiego del flujo que atraviesa el grosor de la preforma. El problema de trasiego de flujo se define en un volumen $\Omega = \Omega_f(t) \cup \Omega_e(t)$, donde el fluido ocupa el volumen $\Omega_f(t)$ para un determinado tiempo t y Ω_e indica el volumen de la parte vacía en el instante t. En la figura 3.4 podemos ver un modelo de dos dimensiones en un molde RTM.

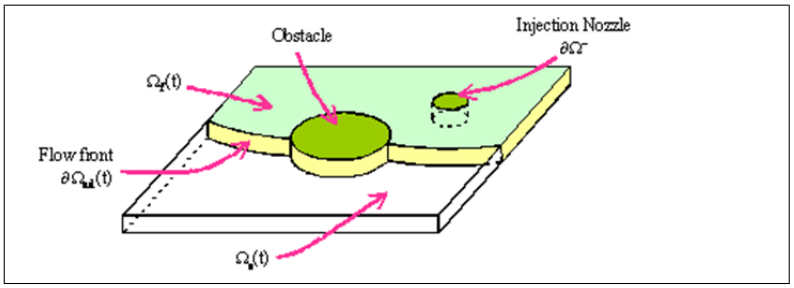


Figura 3.4: Modelo del flujo bidimensional

Asumiendo constante y ortotrópica la permeabilidad de la preforma y constante la viscosidad de la resina, entonces la resolución de las presiones en el dominio ocupado por el fluido se obtiene con la formulación variacional de:

$$\int_{\Omega_f(t)} \nabla p^* \cdot \frac{\underline{\underline{k}}}{\eta} \nabla p d\Omega = 0 \tag{3.2}$$

La localización del fluido dentro de todo el dominio Ω se define mediante la función característica I, definida por:

$$I(\underline{x},t) = \begin{cases} 1 & \text{si } \underline{x} \in \Omega_f(t) \\ 0 & \text{si } \underline{x} \notin \Omega_f(t) \end{cases} \tag{3.3}$$

La evolución de la fracción volumétrica, I, viene dada por la ecuación lineal de advección:

$$\frac{dI}{dt} = \frac{\partial I}{\partial t} + \underline{v} \cdot \nabla I = 0 \tag{3.4}$$

con $I = 1$ en la frontera interior del flujo.

La resolución numérica de las ecuaciones que rigen la cinemática del flujo puede ser tratada mediante la técnica de Galerkin basada en elementos finitos, más conocida como FEM (Finite Element Method). Para la actualización del dominio del flujo se utiliza la técnica de los volúmenes de control. El esquema que conlleva su resolución está formado por tres pasos:

I. Obtener el campo de presiones discretizando por elementos finitos, dada la ecuación variacional (3.2). Se impone presión nula en los nodos no contenidos por el elemento lleno: en la figura 3.5 los nodos 7, 8 y 9.

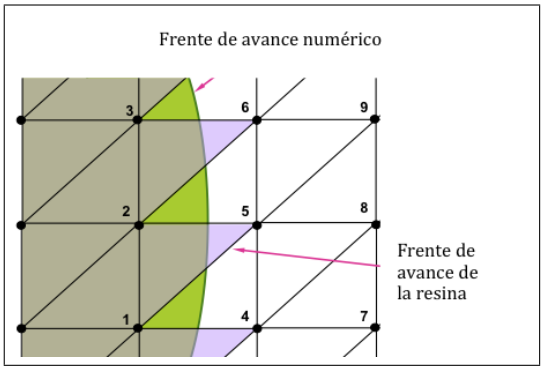


Figura 3.5: Mallado triangular con los volúmenes de control en los elementos.

II. Calcular el campo de velocidades con la ley de Darcy.

III. Actualizar la fracción volumétrica, I, integrando la ecuación (3.4).

Las condiciones en la frontera vienen dadas por las siguientes condiciones:

- El gradiente de presiones en la dirección normal a las paredes del molde es cero, de esta forma el fluido no puede salir de la cavidad del molde.
- La presión o el caudal se definen en la parte llena del molde, $\partial\Omega^-$ como sigue: $p(\underline{x} \in \partial\Omega^-) = P_i$ o $\underline{v}(\underline{x} \in \partial\Omega^-) = \underline{v}_i$ tal que:

$$\partial\Omega^- = \{\underline{x} : \underline{v} \cdot \underline{n} < 0\}$$

  y $\underline{n}(\underline{x})$ es el vector exterior unitario definido en la frontera en el punto $\underline{x}$.
- Presión nula en el frente de avance, es decir, $p(\underline{x} \in \partial\Omega_{ad}(t)) = 0$.

Y si asumimos que en el instante de tiempo $t = 0$, el molde está vacío, la condición inicial de la función I resulta:

$$I(\underline{x}, t = 0) = \begin{cases} 0 & \text{si } \underline{x} \in \Omega \\ 1 & \text{si } \underline{x} \in \partial\Omega^- \end{cases} \tag{3.5}$$

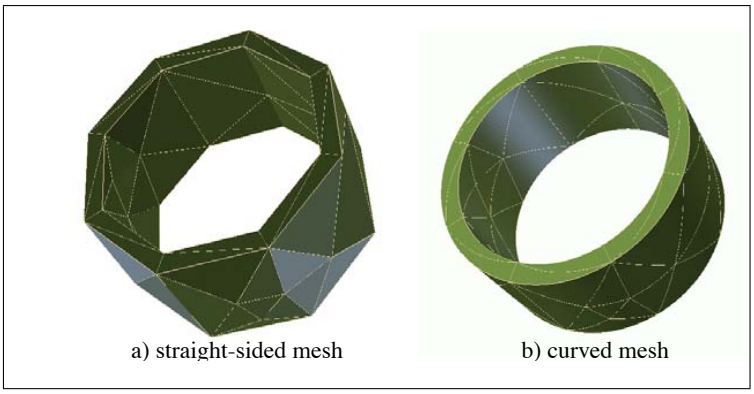


Figura 3.6: Ejemplo de un mallado curvo.

El dominio ocupado por el fluido tiene que estar definido para cada instante de tiempo durante la simulación y los cambios deben estar integrados continuamente en las ecuaciones que rigen. Considerando el estudio del frente de avance como un factor importante en la mejora de la producción, resulta necesario implementar nuevas técnicas que aproximen mejor la representación de éste. La técnica innovadora en este capítulo es darle al frente de avance un trato de curva continua parametrizándolo mediante una curva que precisará de pocos puntos de control, de esta forma se evitan las técnicas de aproximación que son las que hasta ahora se han venido utilizando.

3.4. Ventajas del uso de curvas paramétricas en procesos LCM.

Las técnicas CAGD de parametrización de curvas no están muy extendidas en el caso de los procesos LCM. El uso de las curvas paramétricas está extendido en las técnicas FEM pero como una mejora de los elementos finitos. El objetivo sobre todo está en la minimización del error que se va acumulando al aplicar elementos finitos, debido a la discretización del problema. Muchos de los trabajos publicados al respecto se preocupan sobre todo por utilizar técnicas CAGD para representar la frontera (frente de avance y contorno del molde) de forma más precisa. En publicaciones como [15] ya destacan la importancia del trato de las condiciones de la frontera. En este artículo se destaca que la solución de las ecuaciones de Euler es más precisa mejorando la geometría del dominio de la frontera (elementos finitos isoparamétricos). Si la representación geométrica tiene en cuenta la curvatura de la frontera (por ejemplo, una representación cuadrática) se obtienen soluciones numéricas más significativas. En la publicaciones [107, 108] se muestra la importancia de utilizar mallas adecuadas para discretizaciones de alto orden cuando se resuelven ecuaciones en derivadas parciales.

En el artículo [107] se hace un estudio de la mejora de la precisión de la solución con tan sólo introducir mallados curvos, podemos obervar un ejemplo en la figura 3.6.

Por ejemplo, en [108] generan “The p-version finite element method” donde se desa-

rrolla un método eficaz para aplicarlo en tecnologías de simulación para la ingeniería del diseño. En el artículo se destaca como la aplicación de este método requiere la correcta construcción del mallado del objeto, además de una buena aproximación geométrica para dominios curvos . En la figura 3.7 podemos ver las diferencias al mallar de otra forma, en la figura de arriba el mallado utilizado son líneas rectas. En cambio, en la figura que hay justo bajo se han curvado las líneas rectas utilizando polinomios de grado máximo tres.

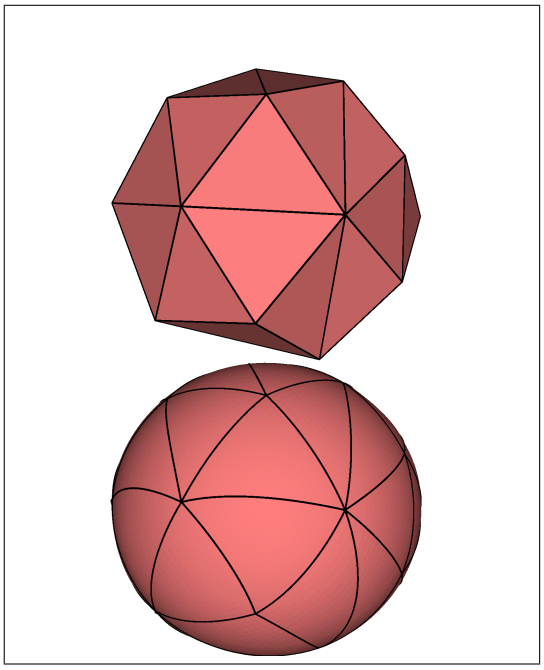


Figura 3.7: El resultado de curvar una malla con lados rectos utilizando un polinomio de grado 3 como máximo.

Además en la figura 3.8 vemos otro ejemplo también de esa misma publicación donde se utilizan las curvas de Bézier para mallar el modelo sin utilizar las líneas rectas. Se utilizan curvas de Bézier cuadráticas y cúbicas, el orden de la curva de Bézier afecta a la forma del borde de la malla.

En esta publicación utilizan curvas de Bézier para el mallado destacando las siguientes propiedades:

- Envolvente convexa: una curva o superficie de Bézier siempre está contenida en la envolvente convexa formada por los puntos de control.
- Disminución de la variación: un plano infinito no puede cortar una curva de Bézier más veces de las que pueda intersectar con el polígono de control, eso permite cálculos de la intersección más eficientes.
- Las derivadas y los productos de las curvas de Bézier se calculan de forma muy sencilla.

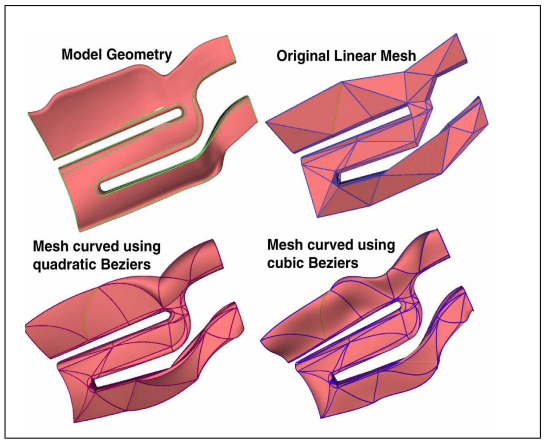


Figura 3.8: Ejemplo de mallado curvo más complejo mostrando como afecta el grado del polinomio de la curva de Bézier.

- Son posibles algoritmos computacionalmente eficientes al elevar el grado o subdividir. Esta propiedad se puede utilizar para refinar la forma de la envolvente convexa tanto como sea necesario para refinar la forma del mallado.

Estas mejoras en cuanto a la representación del dominio introduciendo geometrías curvilíneas las podemos encontrar en trabajos como [169]. Esta publicación se centra en modelar de forma más exacta las geometrías en el contexto de dos tipos de ecuaciones en derivadas parciales: problemas elípticos y problemas de Maxwell.

Además podemos encontrar publicaciones donde introducen el uso de las NURBS en las técnicas de los elementos finitos. En el artículo [80] se propone una nueva metodología: el análisis isogeométrico. El objetivo principal es obtener una exacta representación de la geometría, sin depender de la discretización espacial. En el análisis isogeométrico la solución del problema de la frontera o contorno también se aproxima utilizando las NURBS para la descripción de la geometría. Esta idea ya fue introducida en la publicación [40] en el contexto del análisis de láminas delgadas, pero en vez de utilizar NURBS utilizaban subdivisión de superficies.

Otras publicaciones posteriores donde también se mejora el método clásico de los elementos finitos con una representación más exacta de la frontera es el trabajo [143]. En este caso se utilizan NURBS, una de las curvas más utilizadas en las técnicas CAGD, para representar la frontera del dominio computacional. La técnica que desarrollan la denominan NEFEM (Nurbs-enhanced finite element method). En la figura 3.9 podemos ver un ejemplo gráfico de un mallado utilizando la técnica NEFEM.

Además del uso de las curvas y superficies NURBS también se encuentran trabajos donde se utilizan curvas de Bézier para generar el mallado del dominio de otra forma. Encontramos la publicación [26], donde podemos ver como introducen el uso de triángulos curvos. Justifican que el uso de elementos curvos es la idea clave para no tener que utilizar un mallado muy refinado. De esa forma se utilizan un bajo número de elementos finitos. En este caso utilizan las curvas B-Splines para modelar la frontera o el contorno

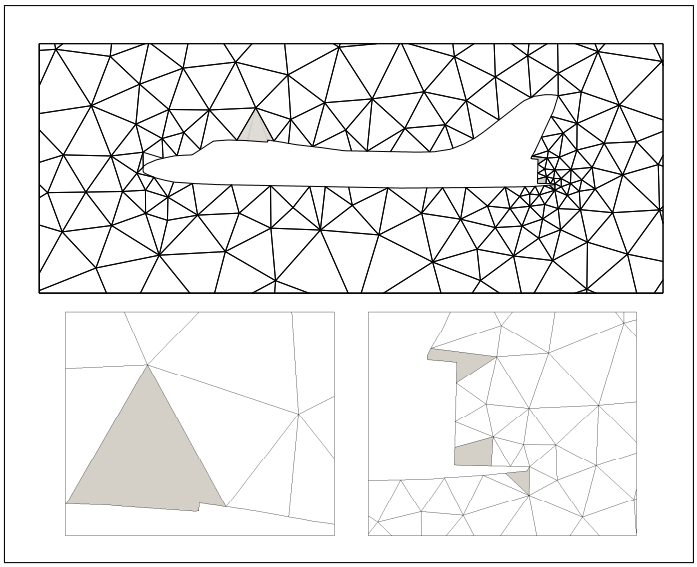


Figura 3.9: Perfil de la aeronave: detalles del mallado computacional del tipo NEFEM.

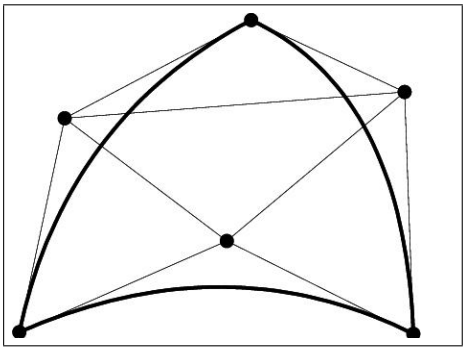


Figura 3.10: Un triángulo de Bézier cuadrático.

del objeto y luego el mallado del objeto se realiza con triángulos Bézier de segundo orden. En la figura 3.10 podemos ver un ejemplo de un triángulo cuadrático de Bézier.

En la 3.11 podemos observar en la figura de la derecha un ejemplo de un mallado utilizando los triángulos de Bézier.

Esta forma de hacer los mallados con los triángulos de Bézier es aplicada en las ecuaciones incompresibles Navier-Stokes [27].

3.5. Evolución de las partículas: EP.

En la publicación [139] se propone una técnica para hacer evolucionar partículas sobre la simulación de un llenado, que denotamos como **EP** (evolución partículas).

El objetivo de este método es determinar la edad del flujo, a través del tiempo que lleva cada partícula dentro del molde. En cada instante de tiempo se introduce partículas de edad cero que se van a desplazar por la geometría discretizada. Para que estas partículas

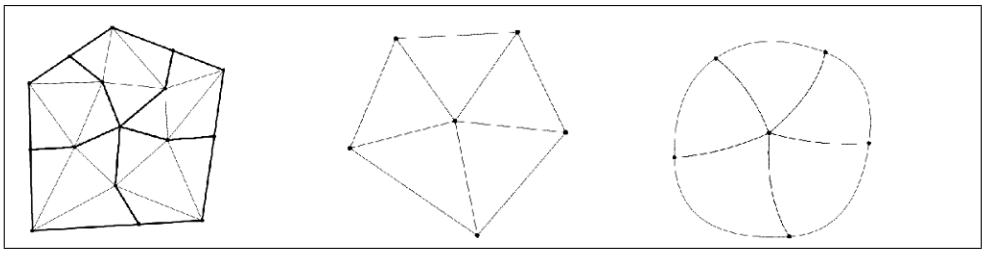


Figura 3.11: En la figura de la izquierda está la malla de control, en el centro está la malla lógica y en la derecha está el mallado con Bézier.

se muevan toman el vector velocidad del elemento finito donde está situada la partícula. Con ello, la posición de la partícula se computa como,

$$\underline{\mathbf{x}}_{k+1} = \underline{\mathbf{x}}_k + \overrightarrow{v} \cdot \Delta t \tag{3.6}$$

Siendo, $\underline{\mathbf{x}}_k$ la posición en el instante actual de la partícula, $\underline{\mathbf{x}}_{k+1}$ la posición de la partícula en el instante siguiente, $\overrightarrow{v}$ es la velocidad obtenida del elemento finito donde está situada la partícula y Δt es el incremento de tiempo que se utiliza en la simulación.

La edad de la partícula $\underline{\mathbf{x}}$ para el instante t se denota como $E(t)$, la nueva edad para el instante siguiente $t + \Delta t$ se expresa como:

$$E(t + \Delta t) = E(t) + \Delta t \tag{3.7}$$

En la figura 3.12 podemos ver un ejemplo de este método que estudia la evolución de las partículas en el llenado del molde.

3.6. Representación y actualización del Frente de avance mediante una curva de Bézier: BFD.

En los procesos LCM ha quedado constancia de tres problemas importantes:

I. La importancia de la información del frente da avance, es decir, de su correcta representación para la mejora del proceso del llenado.

II. La información del frente se tiene que actualizar en cada instante de tiempo, es decir, que a medida que el molde se llena su geometría cambiará en función del tipo de molde, de la viscosidad de la resina, de la cantidad inyectada en ese instante de tiempo, de la estructura de la preforma, etc...La velocidad del flujo de la resina calculada mediante la cinemática del flujo tras aplicar elementos finitos, es la que proporcionará esta actualización. Tener la información de la velocidad equivale a tener un campo de vectores asociados a las partículas del frente de avance.

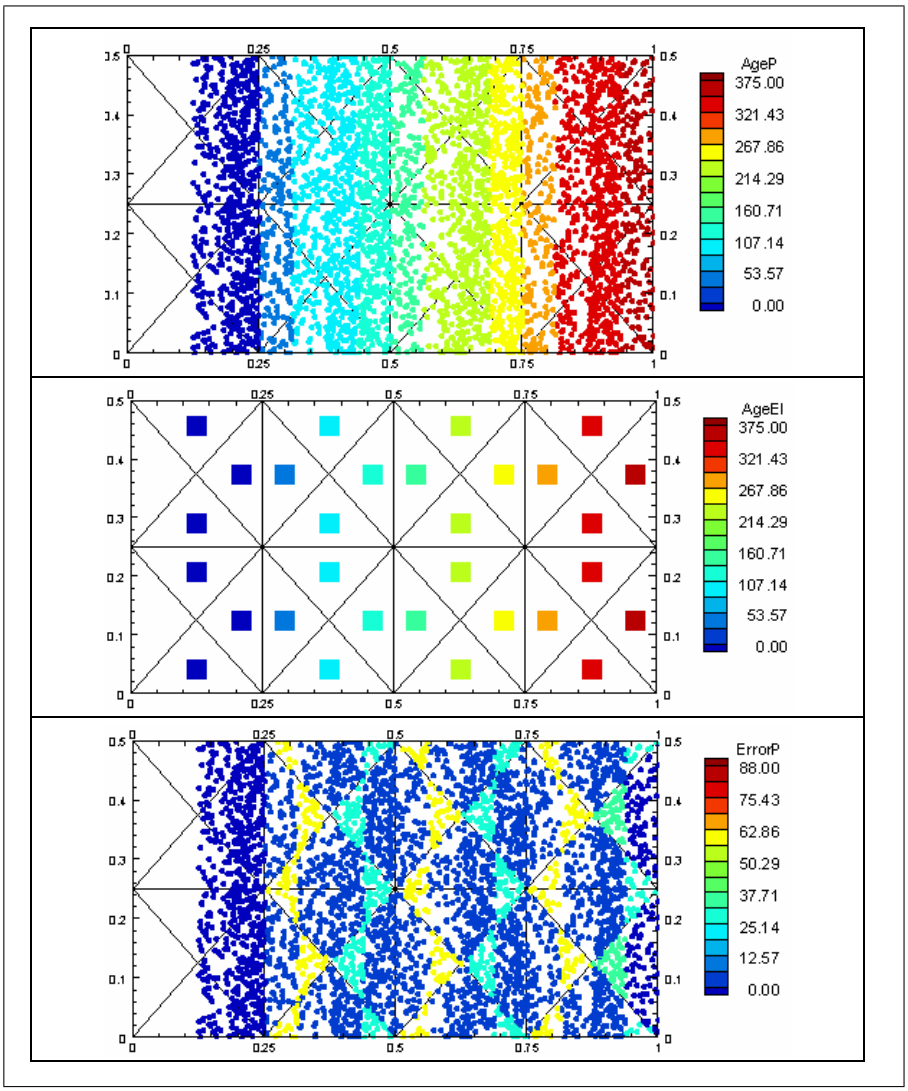


Figura 3.12: De arriba a abajo: edad de las partículas, solución por elementos finitos y la diferencia en cuanto a la posición de las partículas entre ambas. La malla tiene 32 elementos.

III. Puesto que las técnicas más utilizadas para obtener la información del frente de avance son los elementos finitos, esto supone un mallado del dominio. En consecuencia, se pierde la precisión en la frontera del dominio, ya que en muchos de los casos se malla mediante triángulos. Ha quedado constancia de lo relevante que es poder aproximar esos dominios correctamente, por eso en muchas de las técnicas que se utilizan elementos finitos se está introduciendo el uso de curvas paramétricas para aproximarse de forma más precisa en las fronteras de los dominios.

La gran ventaja del algoritmo **BSD** es que nos permite calcular el frente de avance con una curva continua, es decir, con una Bézier. De entre todas las curvas paramétricas, en el capítulo 2 se ha destacado las propiedades ventajosas de las curvas de Bézier. Además, con el **BSD** se tiene la opción de actualizar su información con los vectores velocidad que se obtienen con las técnicas FEM y el método que estudia la evolución de las partículas, **EP**, visto en la sección 3.5.

En consecuencia se define el algoritmo *Bézier Flow Front Deformation*, **BFD**.

De esta forma, a través del método **EP** se obtiene un conjunto discreto de partículas con una edad correspondiente. Estas partículas son las que se necesitan para generar la curva inicial de Bézier, es decir, los Puntos Iniciales necesarios en el **BFD**.

Con la información del vector velocidad, dichas partículas cogen los correspondientes vectores del elemento finito donde están situados y se calcula la nueva localización de éstas. El campo de vectores velocidad es el que es proporcionado al algoritmo **BFD** para actualizar la información del frente de avance, reparametrizando el frente con la curva de Bézier modificada. Como consecuencia de utilizar el **BFD** conseguimos: una representación más real y precisa del frente de avance de la resina y conseguimos la información de los frentes de avance de la misma edad, es decir, no sólo queda delimitada la frontera entre la parte seca y llena del molde, si no que además se tiene información de lo que ha ocurrido en los instantes anteriores.

3.6.1. Adaptación del BSD al BFD.

La adaptación del **BFD** al **BSD** se va a detallar como sigue ya que la función a optimizar mantiene su definición y todas las restricciones incluidas en el **BSD** son útiles para el **BFD**.

La función a optimizar en el **BSD**, ecuación 2.8, minimiza la energía utilizada entre la curva de Bézier original y la curva de Bézier modificada. En el caso del **BFD** esto significa que el frente de avance se deforma lo mínimo posible respecto del original cumpliendo todas las restricciones incluidas en el problema.

En cuanto a las restricciones, hay que pensar que el frente de avance se tiene que actualizar, por ello el **BFD** necesita de un campo de vectores, en este caso velocidades. La forma de obtener estas velocidades la va a proporcionar el método **EP**, por ello vamos a fusionar dos algoritmos el **BFD** y el **EP**: **BFD+EP**. En la figura 3.13 podemos ver el esquema de como se fusionan el algoritmo de evolución de las partículas y el algoritmo que representa y actualiza la información del frente mediante una Bézier.

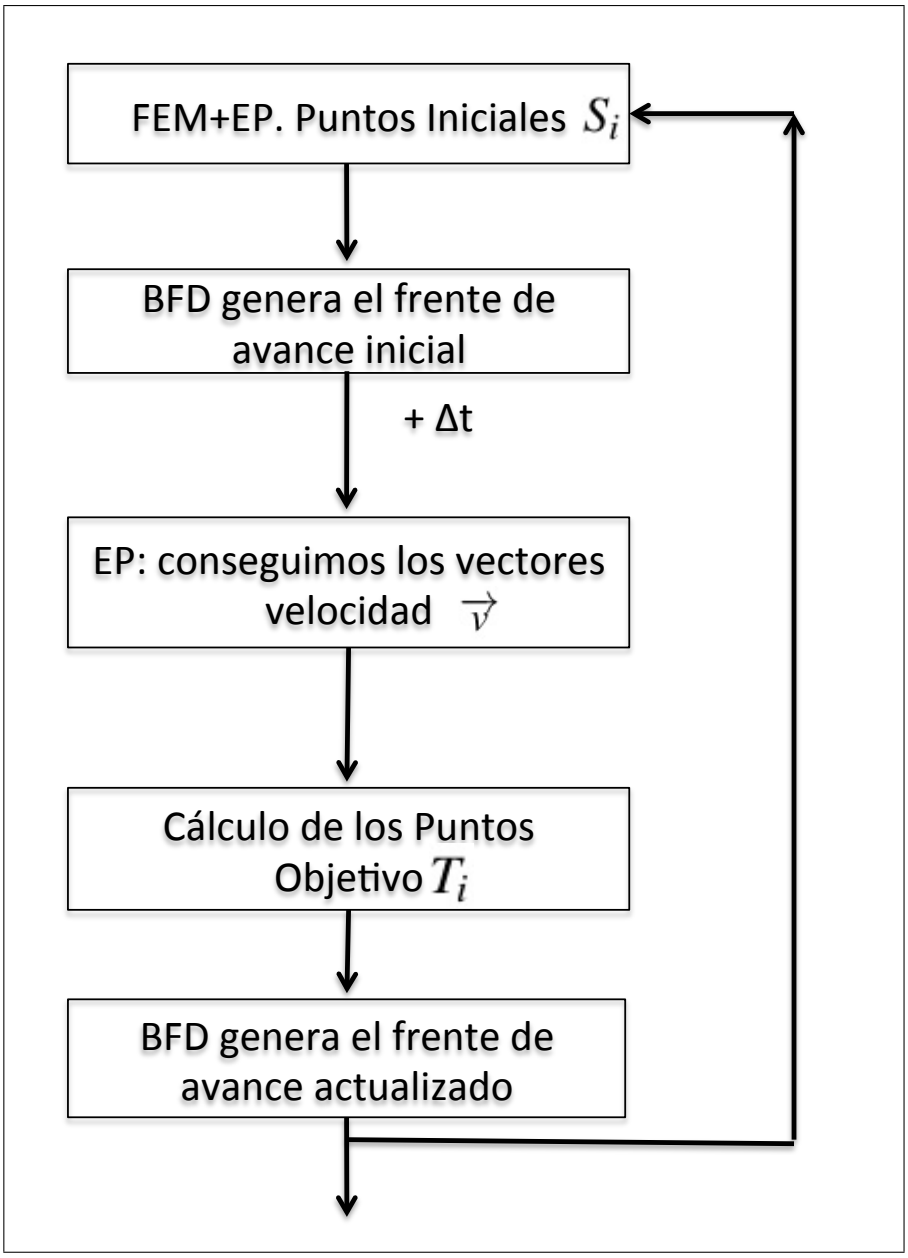


Figura 3.13: Esquema del algoritmo **BFD+EP**.

Todas las restricciones utilizadas para el **BSD** son necesarias en el **BFD**. En el caso del **BFD** cada restricción tiene un significado concreto que se detalla como sigue:

1. **Fusión del BFD y el método EP.**

   La ecuación 2.10 es necesaria incluirla porque es la restricción básica del algoritmo. En ella se exige que la curva de Bézier modificada (que en este caso representa el frente de avance) pase a través del Punto Final. En el caso del **BFD**, el vector velocidad del flujo de la resina es el vector que une la posición inicial de la partícula

que corresponde a un punto de la curva de Bézier original (el Punto Inicial) con la posición final de la partícula que tiene que corresponder a un punto de la curva de Bézier modificada (el Punto Final). En la figura 3.14 vemos un ejemplo del vector velocidad que une los Puntos Iniciales, S_i, con los Puntos Finales, T_i. El número de vectores que podemos colocar en una curva de Bézier de orden n es $(n-1)$.

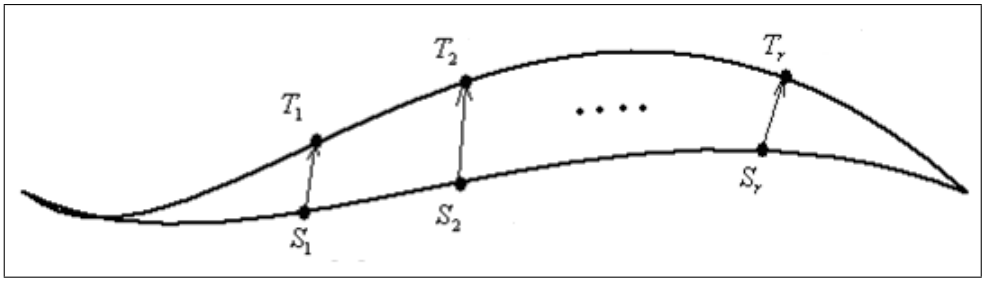


Figura 3.14: Representación del **BFD**, los vectores velocidad unen los Puntos Iniciales S_i con los Puntos Finales T_i.

Estos vectores velocidad (situados en el centroide de cada elemento) asociados a cada elemento finito se obtienen aplicando técnicas FEM y con el método **EP** tal y como se ha detallado en la secciones 3.3 y 3.5 o en las publicaciones [138, 139]. Fusionando **BFD** con **EP**, obteniendo **BFD+EP**, se calculan los Puntos Finales, T_i a partir de los Puntos Iniciales, S_i y los vectores velocidad,

$$\begin{aligned} T_1 &= S_1 + v_1 \Delta t \\ T_2 &= S_2 + v_2 \Delta t \\ &\vdots \\ T_r &= S_r + v_r \Delta t \end{aligned} \tag{3.8}$$

2. **Mantener la tangencia entre el punto inicial de la primera curva de Bézier y el punto final de la última curva de Bézier**.

   Las ecuaciones 2.11 y 2.12 también son necesarias incluirlas. Con ellas mantenemos la tangencia en el punto inicial y final. Esta restricción es necesaria incluirla en el **BFD** porque en el caso de la resina, el frente de avance que se obtiene deformado debe mantener las propiedades del frente de avance original. De esa forma se evitan oscilaciones en el Punto Inicial y Final de las curvas concatenadas tal y como podemos ver también en la figura 3.14.

3. **Continuidad y derivabilidad en los puntos donde se concatenan las curvas**.

   A lo largo de este capítulo ya hemos visto que el frente de avance en realidad es una curva suave, esa es la razón principal por la que el frente de avance se representa con una curva de Bézier. Como consecuencia de concatenar una familia de k curvas de Bézier es necesario incluir las condiciones de continuidad y derivabilidad en los puntos donde se concatenan los puntos de las curvas de Bézier. Así que hay que incluir las ecuaciones 2.19 y 2.22.

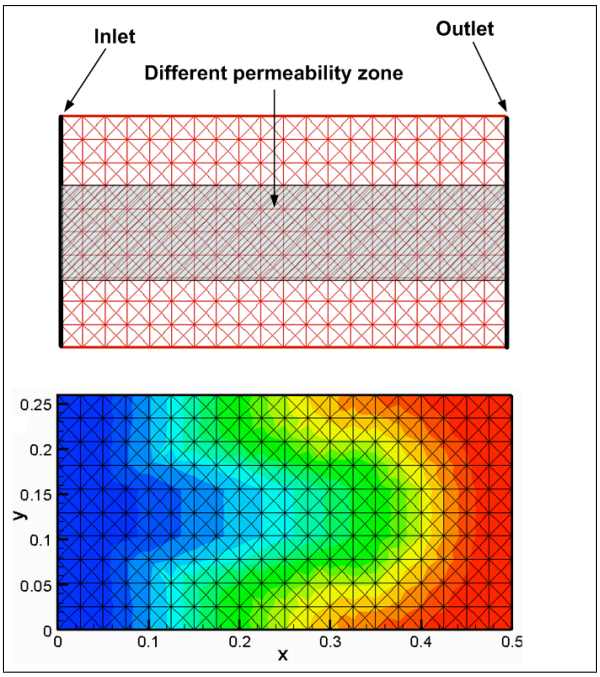


Figura 3.15: Simulación de un llenado.

3.6.2. Resultados de simulaciones.

La Figura 3.15 representa la simulación de un llenado, en ese caso el molde es rectangular y el inyector está situado a la izquierda mientras que el vacío se coloca a la derecha. En la parte central del rectángulo se ha colocado una zona con permeabilidad alta, como consecuencia de ello se producen cambios en el frente de avance. En este caso, la simulación del frente de avance se obtiene mediante elementos finitos.

Con el algoritmo **BFD** queremos calcular el frente de avance de forma continua mediante la unión de curvas de Bézier y actualizarlo con el campo de vectores velocidad obtenidos mediante la técnica **EP**.

En la parte superior de la figura 3.16 tenemos el frente de avance obtenido al aplicar **BFD+EP** superpuesto con el resultado de la simulación numérica. En la parte inferior de la figura 3.16 tenemos la edad del frente de avance y la forma de cada frente para esa edad, representado a través del **BFD+EP**. La zona gris es la simulación del frente de avance. Las curvas de Bézier están superpuestas y representan el frente de avance parametrizado de forma continua. Para evitar los posibles lazos el **BFD+EP** se ha computado con curvas de Bézier de segundo orden.

Las ventajas que se obtienen al utilizar el **BFD** son dos principalmente: la primera es que el frente se computa mediante una curva suave, flexible y continua como es la curva de Bézier, la segunda es que también se obtiene la información del frente de avance en función de la edad de las partículas de la resina. Si se conoce la edad del frente se pueden diseñar técnicas para mejorar el curado de la resina en el llenado.

Cuando se inyecta resina en el molde se inyectan dos tipo de componente: la resina y

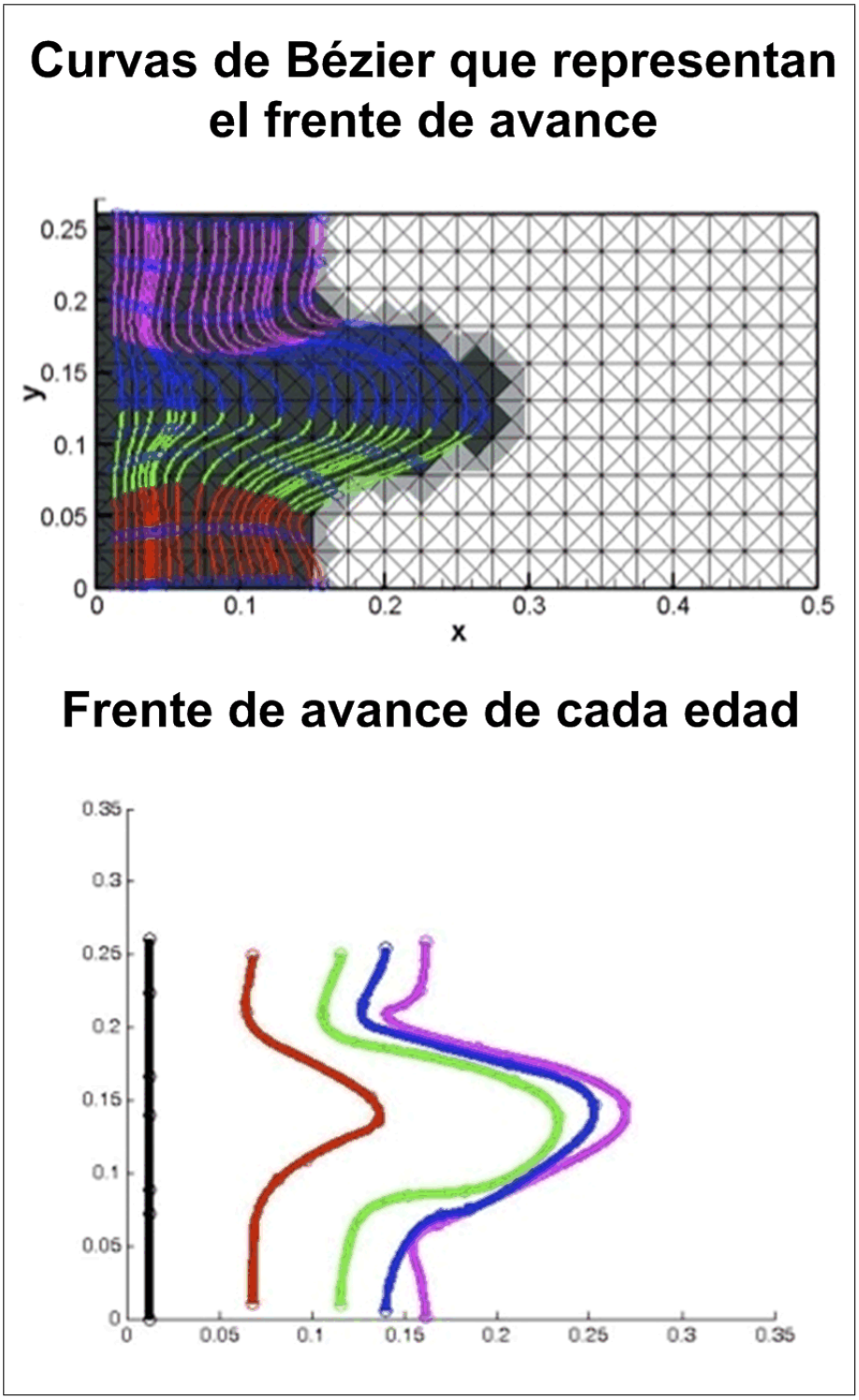


Figura 3.16: Frente de avance para la edad de las partículas representado mediante el **BFD** con curvas de Bézier de segundo orden.

un agente que actúa para el curado de la resina. Al principio del llenado se suele inyectar más resina y menos agente para que no cure tan rápido, en cambio a medida que el molde se va llenando hay que inyectar más agente para que al final la resina cure de la misma forma.

El objetivo es que el curado de la resina sea homogéneo, si se conoce previamente la forma del curado es posible aplicar más o menos calor en determinadas zonas en función de la forma del frente. Por ejemplo, en geometrías complejas el llenado de un molde no resulta tan sencillo porque se encuentran flujos de resina de diferentes edades y eso provoca diferencias en cuanto al curado.

Para el estudio de la edad se introducen en cada instante de tiempo la familia de curvas de Bézier concatenadas que evolucionan. La edad de cada una de ellas evoluciona con la ecuación 3.7.

3.7. Introducción de otra restricción: BFD-A.

La ventaja de la formulación del **BFD** es la posibilidad de introducir nuevas restricciones al problema. Con el objetivo de mejorar el **BFD** se introdujo una nueva restricción que relaciona el área encerrada entre dos curvas de Bézier en diferentes instantes de tiempo y el caudal o cantidad de resina introducida en este intervalo de tiempo.

De esa forma la deformación de las curvas de Bézier a través de los vectores velocidad del flujo, verificarían la Ley de la Conservación de la Masa. En consecuencia, la curva de Bézier deformada no sólo incluye la información de los vectores velocidad si no que además introduce la información del caudal de resina utilizado en ese intervalo de tiempo concreto.

El algoritmo correspondiente a esta técnica se denomina:
Bézier Flow Front Deformation-Area, **BFD-A**.

3.7.1. Desarrollo matemático del algortimo BFD-A.

Para poder aplicar la restricción hay que calcular el área encerrada entre dos curvas de Bézier, eso implica que hay que calcular el área entre dos curvas definidas de forma paramétrica.

Para calcular el valor del área se va a utilizar el Corolario del Teorema de Green-Riemann, ver apéndice B Teorema 5 y Corolario 6.

Este resultado, el corolario 6, se utiliza para calcular el área encerrada entre dos curvas de Bézier, ver figura 3.17. En el algoritmo **BFD-A** estos resultados se utilizan para calcular el área encerrada entre dos frentes de avance en diferentes instantes de tiempo. Este valor se compara con el volumen o caudal de resina inyectado en este instante de tiempo concreto.

Considerando el caso que se representa en la figura 3.17 y empleando el Corolario del Teorema de Green-Riemmann junto con la definición de integral curvilínea, (ver apéndice B Corolario 6 y definición 12) planteamos las siguientes parametrizaciones.

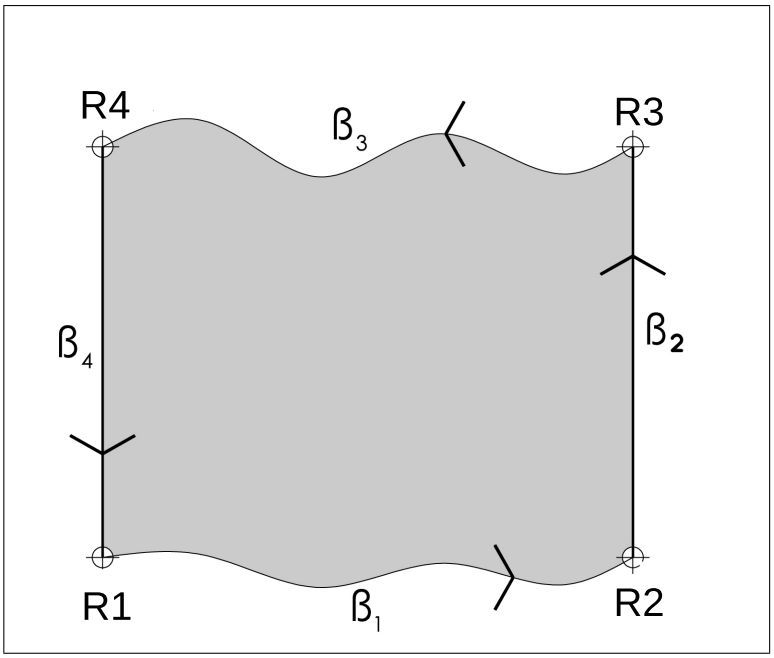


Figura 3.17: Área entre dos curvas de Bézier.

1. β_1 y β_3 son curvas de Bézier concatenadas. En principio se han realizado los cálculos y las simulaciones utilizando dos curvas de Bézier de segundo orden.

2. β_2 y β_4 son segmentos que unen los puntos R_2 con R_3 y R_4 con R_1 respectivamente.

Considerando que $\beta = \beta_1 \cup \beta_2 \cup \beta_3 \cup \beta_4$, se desarrolla como sigue:

$$area(D) = \int\int_D dA = \int_\beta -y\,dx = \int_0^1 F(\beta(u)) \cdot \beta'(u)\,du =$$

$$= \int_{\beta_1} -y\,dx + \int_{\beta_2} -y\,dx + \int_{\beta_3} -y\,dx + \int_{\beta_4} -y\,dx =$$

$$\int_0^1 -y_1(u) \cdot x_1'(u)\,du + \int_0^1 -y_2(u) \cdot x_2'(u)\,du + \int_0^1 -y_3(u) \cdot x_3'(u)\,du + \int_0^1 -y_4(u) \cdot x_4'(u)\,du \tag{3.9}$$

Siendo las parametrizaciones de cada camino las siguientes:

1. β_1 será la concatenación de varias curvas de Bézier de orden n_1 y n_2 respectivamente.

   Para que las curvas sean inyectivas y no se produzcan lazos, se trabajará con curvas de orden 2. Se van a concatenar dos curvas de Bézier, si se necesita concatenar más, tan solo habría que extender el resultado.

Puesto que β_1 es la concatenación de dos curvas de Bézier, entonces lo podemos denotar de la siguiente forma: β_1=$\alpha_1 \cup \alpha_2$.

$$\begin{aligned}
&\alpha_1(u) = (x_1(u), y_1(u)) = \sum_{i=0}^{n_1} \mathbf{P}_i \binom{n_1}{i} u^i (1-u)^{(n_1-i)} = \\
&= \mathbf{P}_0(1-u)^2 + \mathbf{P}_1 2u(1-u) + \mathbf{P}_2 u^2 = \\
&= \mathbf{P}_0(1-2u+u^2) + \mathbf{P}_1(2u-2u^2) + \mathbf{P}_2 u^2 = \\
&= u^2(\mathbf{P}_0 - 2\mathbf{P}_1 + \mathbf{P}_2) + u(-2\mathbf{P}_0 + 2\mathbf{P}_1) + \mathbf{P}_0; u \in [0,1]
\end{aligned} \tag{3.10}$$

Siendo $\mathbf{P}_i$ los puntos de control de la primera curva de Bézier.

De la forma en la que se ha construido el problema, $\mathbf{P}_0 = \mathbf{R}_1$. La segunda curva, concatenada con la primera tiene la siguiente expresión:

$$\begin{aligned}
&\alpha_2(u) = (x_1(u), y_1(u)) = \sum_{i=0}^{n_2} \mathbf{Q}_i \binom{n_2}{i} u^i (1-u)^{(n_2-i)} = \\
&= \mathbf{Q}_0(1-u)^2 + \mathbf{Q}_1 2u(1-u) + \mathbf{Q}_2 u^2 = \\
&= \mathbf{Q}_0(1-2u+u^2) + \mathbf{Q}_1(2u-2u^2) + \mathbf{Q}_2 u^2 = \\
&= u^2(\mathbf{Q}_0 - 2\mathbf{Q}_1 + \mathbf{Q}_2) + u(-2\mathbf{Q}_0 + 2\mathbf{Q}_1) + \mathbf{Q}_0; u \in [0,1]
\end{aligned} \tag{3.11}$$

Siendo $\mathbf{Q}_i$ los puntos de control de la segunda curva de Bézier.

Tal y como está construido el problema se cumple que: $\mathbf{Q}_2 = \mathbf{R}_2$ y puesto que α_1 está concatenada con α_2, entonces: $\mathbf{P}_2 = \mathbf{Q}_0$.

2. β_2 es el segmento que une R_2 con R_3.

$$\beta_2 = (x_2(u), y_2(u)) = \mathbf{R}_2 + u\overrightarrow{\mathbf{R}_2\mathbf{R}_3} = \mathbf{R}_2 + u(\mathbf{R}_3 - \mathbf{R}_2); u \in [0,1] \tag{3.12}$$

3. β_3 es la unión de dos curvas de Bézier concatenadas. Además es la curva de Bézier modificada después de introducir los vectores en el algoritmo, es decir, que la ecuación de estas curvas de Bézier se obtienen a partir de β_1 perturbando los puntos de control $\mathbf{P}_i$ y $\mathbf{Q}_i$ con un vector perturbación $\varepsilon_i^{(1)}$ y $\varepsilon_i^{(2)}$ respectivamente.

   Así pues $\beta_3 = S_\varepsilon(-\alpha_1(u)) \cup S_\varepsilon(-\alpha_2(u))$

$$\begin{aligned}
&S_\varepsilon(-\alpha_1(u)) = (x_3(u), y_3(u)) = \sum_{i=0}^{n_1} (\mathbf{P}_i + \varepsilon_i^{(1)}) \binom{n_1}{i} u^i (1-u)^{(n_1-i)} = \\
&= (\mathbf{P}_0 + \varepsilon_0^{(1)})(1-u)^2 + (\mathbf{P}_1 + \varepsilon_1^{(1)}) 2u(1-u) + (\mathbf{P}_2 + \varepsilon_2^{(1)}) u^2, u \in [0,1]
\end{aligned} \tag{3.13}$$

$$S_\varepsilon(-\boldsymbol{\alpha}_2(u)) = (x_3(u), y_3(u)) = \sum_{i=0}^{n_2} (\mathbf{Q}_i + \varepsilon_i^{(2)}) \binom{n_2}{i} u^i (1-u)^{(n_2-i)} =$$

$$= (\mathbf{Q}_0 + \varepsilon_0^{(2)})(1-u)^2 + (\mathbf{Q}_1 + \varepsilon_1^{(2)}) 2u(1-u) + (\mathbf{Q}_2 + \varepsilon_2^{(2)}) u^2, u \in [0,1] \tag{3.14}$$

4. β_4 es el segmento que une R_1 con R_4.

$$\beta_4 = (x_4(u), y_4(u)) = \mathbf{R}_4 + u\overrightarrow{\mathbf{R}_4\mathbf{R}_1} = \mathbf{R}_4 + u(\mathbf{R}_1 - \mathbf{R}_4); u \in [0,1] \tag{3.15}$$

Los cálculos que tenemos que hacer para las integrales son los siguientes:

- $$\int_{\beta_1} -y\,dx = \int_{\alpha_1} -y\,dx + \int_{\alpha_2} -y\,dx = \int_0^1 -y_1 x_1'(u)\,du + \int_0^1 -y_1 x_1'(u)\,du =$$

$$= \begin{pmatrix} P_0^x & P_1^x & P_2^x \end{pmatrix} \begin{pmatrix} 1/2 & 1/3 & 1/6 \\ -1/3 & 0 & 1/3 \\ -1/6 & -1/3 & -1/2 \end{pmatrix} \begin{pmatrix} P_0^y \\ P_1^y \\ P_2^y \end{pmatrix} +$$

$$+ \begin{pmatrix} Q_0^x & Q_1^x & Q_2^x \end{pmatrix} \begin{pmatrix} 1/2 & 1/3 & 1/6 \\ -1/3 & 0 & 1/3 \\ -1/6 & -1/3 & -1/2 \end{pmatrix} \begin{pmatrix} Q_0^y \\ Q_1^y \\ Q_2^y \end{pmatrix} =$$

$$= \mathbf{P}_X^T \cdot B \cdot \mathbf{P}_Y + \mathbf{Q}_X^T \cdot B \cdot \mathbf{Q}_Y \tag{3.16}$$

  Siendo:

  - $\mathbf{P}_X$ el vector de las componentes x de los puntos de control de la primera curva $\boldsymbol{\alpha}_1$.
  - $\mathbf{P}_Y$ el vector de las componentes y de los puntos de control de la primera curva $\boldsymbol{\alpha}_1$.
  - $\mathbf{Q}_X$ el vector de las componentes x de los puntos de control de la segunda curva $\boldsymbol{\alpha}_2$.
  - $\mathbf{Q}_Y$ el vector de las componentes y de los puntos de control de la segunda curva $\boldsymbol{\alpha}_2$.
  - $B = \begin{pmatrix} 1/2 & 1/3 & 1/6 \\ -1/3 & 0 & 1/3 \\ -1/6 & -1/3 & -1/2 \end{pmatrix}$. La dimensión de esta matriz depende del orden de las curvas de Bézier, y va ligada directamente al número de puntos de control. En este caso, puesto que las curvas de Bézier tienen orden 2, entonces hay tres puntos de control y la matriz es de tamaño $3 * 3$.

- $$\int_{\beta_2} -y\,dx = \int_0^1 -y_2(u)x_2'(u)\,du = \frac{-1}{2}(R_3^x - R_2^x)\cdot(R_2^y + R_3^y) \tag{3.17}$$

  Siendo $\mathbf{R}_2 = \mathbf{Q}_2 = (Q_2^x, Q_2^y)$ y $\mathbf{R}_3 = Q_2 + \varepsilon_2^{(2)} = (Q_2^x, Q_2^y) + (\varepsilon_2^{(2),(x)}, \varepsilon_2^{(2),(y)})$.

  Entonces:
  $\int_{\beta_2} -y\,dx = -Q_2^y\varepsilon_2^{(2),(x)} - \frac{1}{2}\varepsilon_2^{(2),(x)}\varepsilon_2^{(2),(y)}$

- $\int_{\beta_3} -y\,dx = \int_{S_\varepsilon(-\alpha_1)} -y\,dx + \int_{S_\varepsilon(-\alpha_2)} -y\,dx = -\int_{S_\varepsilon(\alpha_1)} -y\,dx - \int_{S_\varepsilon(\alpha_2)} -y\,dx =$

  $$= -\begin{pmatrix} P_0^x + \varepsilon_0^{(1),x} & P_1^x + \varepsilon_1^{(1),x} & P_2^x + \varepsilon_2^{(1),x} \end{pmatrix} \begin{pmatrix} 1/2 & 1/3 & 1/6 \\ -1/3 & 0 & 1/3 \\ -1/6 & -1/3 & -1/2 \end{pmatrix} \begin{pmatrix} P_0^y + \varepsilon_0^{(1),y} \\ P_1^y + \varepsilon_1^{(1),y} \\ P_2^y + \varepsilon_2^{(1),y} \end{pmatrix} -$$

  $$- \begin{pmatrix} Q_0^x + \varepsilon_0^{(2),x} & Q_1^x + \varepsilon_1^{(2),x} & Q_2^x + \varepsilon_2^{(2),x} \end{pmatrix} \begin{pmatrix} 1/2 & 1/3 & 1/6 \\ -1/3 & 0 & 1/3 \\ -1/6 & -1/3 & -1/2 \end{pmatrix} \begin{pmatrix} Q_0^y + \varepsilon_0^{(2),y} \\ Q_1^y + \varepsilon_1^{(2),y} \\ Q_2^y + \varepsilon_2^{(2),y} \end{pmatrix} =$$

  $$= (\mathbf{P}_X + \varepsilon^{(1),X})^T \cdot (-B) \cdot (\mathbf{P}_Y + \varepsilon^{(1),Y}) + (\mathbf{Q}_X + \varepsilon^{(2),X})^T \cdot (-B) \cdot (\mathbf{Q}_Y + \varepsilon^{(2),Y}) \tag{3.18}$$

- $$\int_{\beta_4} -y\,dx = \int_0^1 -y_4(u)x_4'(u)\,du = \frac{-1}{2}(R_1^x - R_4^x)\cdot(R_1^y + R_4^y) \tag{3.19}$$

  Siendo: $\mathbf{R}_1 = \mathbf{P}_0 = (P_0^x, P_0^y)$ y $\mathbf{R}_4 = \mathbf{P}_0 + \varepsilon^{(1)} = (P_0^x, P_0^y) + (\varepsilon_0^{(1),(x)}, \varepsilon_0^{(1),(y)})$.
  Entonces: $\int_{\beta_4} -y\,dx = P_0^y\varepsilon_0^{(1),(x)} + \frac{1}{2}\varepsilon_0^{(1),(x)}\varepsilon_0^{(1),(y)}$

El valor del área, es la suma de las cuatro integrales anteriores, quedando de la siguiente forma:

$$\begin{aligned} \text{area}(D) &= \int\int_D dA = \int_\beta -y\,dx = \\ &= (-1)\mathbf{P}_X^T B(\varepsilon^{(1),Y}) + (-1)(\varepsilon^{(1),X})^T B\mathbf{P}_Y + (-1)(\varepsilon^{(1),X})^T B(\varepsilon^{(1),Y}) + \\ &+ (-1)\mathbf{Q}_X^T B(\varepsilon^{(2),Y}) + (-1)(\varepsilon^{(2),X})^T B\mathbf{Q}_Y + (-1)(\varepsilon^{(2),X})^T B(\varepsilon^{(2),Y}) + \\ &+ (P_0^y\varepsilon_0^{(1),(x)} + \tfrac{1}{2}\varepsilon_0^{(1),(x)}\varepsilon_0^{(1),(y)}) + (-Q_2^y\varepsilon_2^{(2),(x)} - \tfrac{1}{2}\varepsilon_2^{(2),(x)}\varepsilon_2^{(2),(y)}) \end{aligned} \tag{3.20}$$

Para terminar el planteamiento del problema tan sólo hay que añadir una restricción más al planteamiento original del **BFD** de forma que esa nueva restricción sea:

$$area - caudal * \frac{\Delta t}{grosor} = 0. \tag{3.21}$$

Si se construye de esta forma la restricción, dimensionalmente será correcta. Hay que tener en cuenta que el caudal es volumen dividido por intervalo de tiempo, es decir, $\frac{V}{t}$, considerando que el grosor se considera constante, entonces no variará a medida que transcurre el tiempo. Puesto que las unidades del área son, $[u^2]$, entonces el caudal hay que multiplicarlo por el intervalo de tiempo transcurrido entre la curva original de Bézier y la modificada y luego dividirlo por el grosor del molde.

Con todo esto se construye una nueva función Lagrangiana $\overline{L(\varepsilon,\boldsymbol{\lambda})}$ a partir de la que se ha construido en el **BFD**.

$$\overline{L(\varepsilon,\boldsymbol{\lambda})} = L(\varepsilon,\boldsymbol{\lambda}) + \boldsymbol{\lambda}(area(D) - caudal) \tag{3.22}$$

Siendo $L(\varepsilon,\boldsymbol{\lambda})$ la ecuación 2.31.

El sistema se resuelve de la misma forma que en el capítulo 2, haciendo cero las parciales como en la ecuación 2.33. Al añadir una restricción más como consecuencia se añade una fila y columna más a la matriz A que definíamos en la ecuación 2.34.

La desventaja del **BFD-A** es el tipo de sistema que se obtiene. Al añadir la restricción del área el sistema que se obtiene no es lineal y eso implica que para poder resolverlo tenemos que aplicar métodos iterativos de resolución de sistemas no lineales. El método utilizado para resolver este sistema es: "trust region-dogleg", ver [85]. El inconveniente de estos métodos es el coste computacional puesto que es bastante elevado.

3.7.2. Resultados de simulación del BFD-A.

Algunas de las simulaciones de este algoritmo las podemos ver en las siguientes figuras: en la figura 3.18 tenemos la prueba de que el algoritmo funciona utilizando vectores paralelos y con la misma longitud, de forma que así conocemos el valor real del área encerrada entre esas curvas de Bézier porque son rectángulos.

En la siguiente figura 3.19 tenemos un ejemplo de la deformación de dos curvas de Bézier concatenadas cambiando el ángulo y la longitud de los vectores.

Por último en la figura 3.20 tenemos la simulación de un llenado en un molde rectangular donde tenemos representada la evolución de la edad del frente de la resina utilizando seis curvas de Bézier de segundo orden en el algoritmo BFD-A.

3.8. Conclusiones.

El algoritmo **BFD+EP** simula el frente de avance de la resina mediante una curva paramétrica continua, en este caso Bézier, y lo actualiza en cada instante de tiempo me-

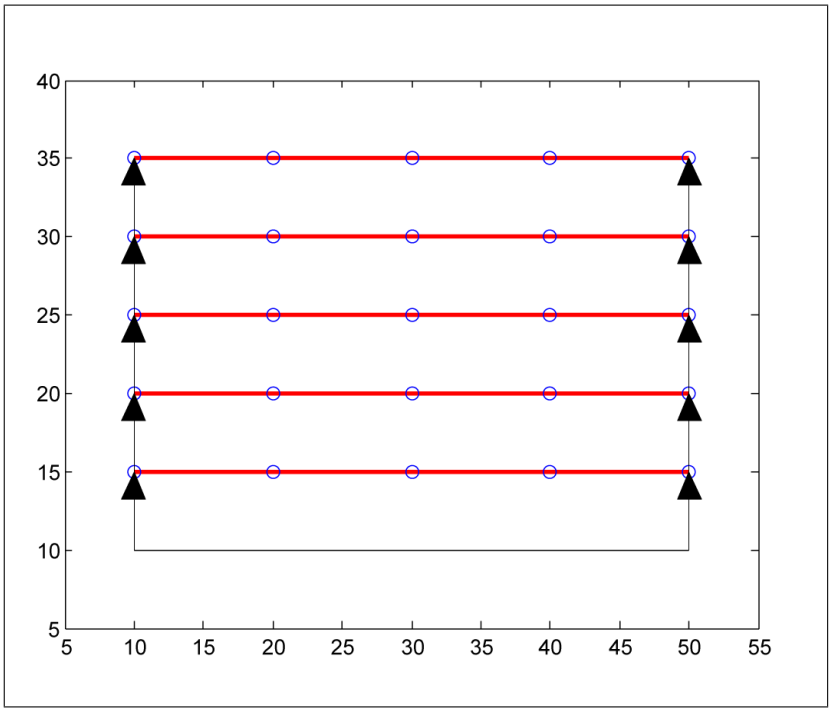


Figura 3.18: Ejemplo 1 de la simulación del **BFD-A**.

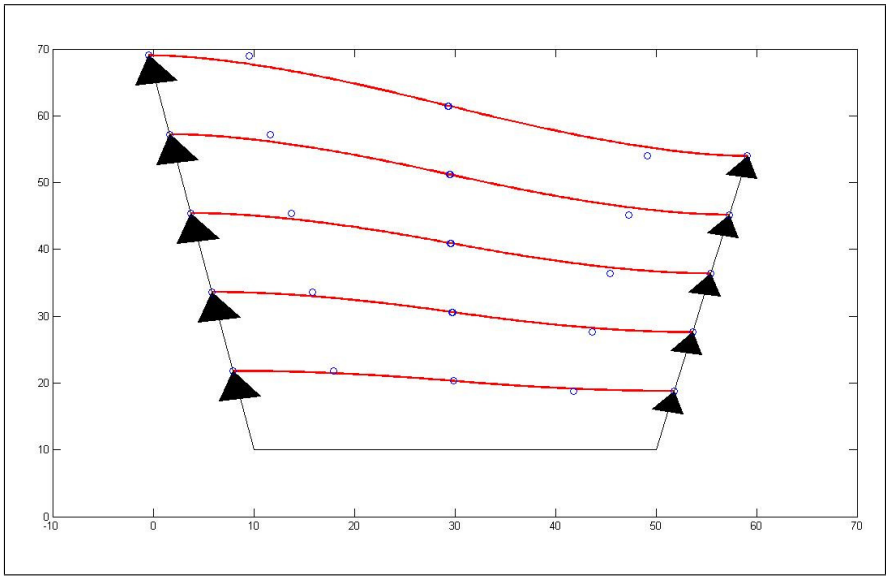


Figura 3.19: Ejemplo 2 de la simulación del **BFD-A**.

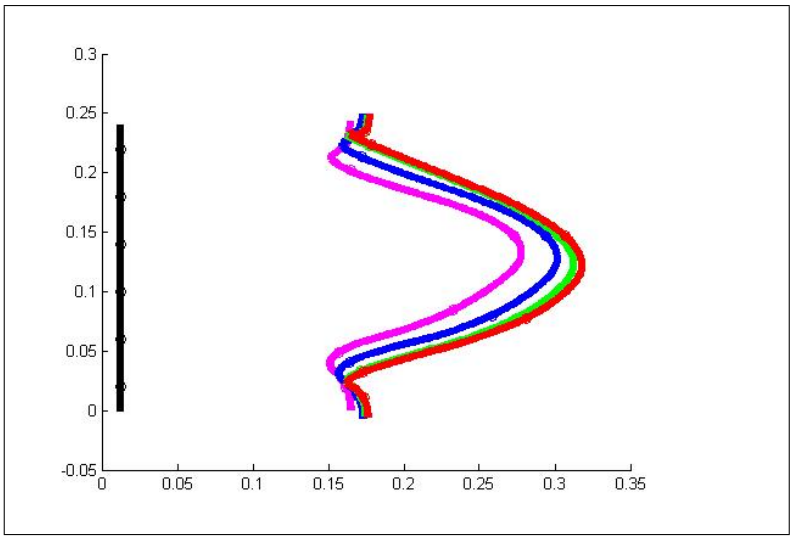


Figura 3.20: Evolución de la edad del frente de avance de la resina con el algoritmo **BFD-A** utilizando seis curvas de Bézier de segundo orden.

diante los vectores velocidad obtenidos a través de las técnicas de simulación que nos proporcionan los elementos finitos y la técnica **EP**.

Las ventajas al utilizar el algoritmo **BFD+EP** en los procesos LCM son:

I. Puesto que las curvas de Bézier son curvas continuas que además son derivables e integrables, permiten un análisis más profundo del frente de avance.

II. Es una alternativa continua al frente de avance calculado como un diente de sierra, que es la representación que se obtiene como consecuencia del uso de las técnicas FEM, ver la figura 3.5.

III. Como el frente de avance computado con el **BFD** se actualiza en cada instante de tiempo mediante los vectores velocidad, ello permite conocer la edad de la evolución de la resina. Con esta información se permite analizar el comportamiento de la resina dentro del fluido, ya que es importante poder desarrollar técnicas que nos ofrezcan información del curado de la resina. Uno de los objetivos principales es poder conseguir un curado de la resina en el molde de forma homogénea.

IV. Reducción drástica de los nodos involucrados en la descripción del frente de avance cuando se aplican técnicas clásicas de simulación por elementos finitos.

La formulación con la que se desarrolla el algoritmo **BFD** permite cambiar, añadir o quitar más restricciones al algoritmo y con ello mejorar el resultado obtenido exigiendo otras condiciones al problema.

Esa ventaja se plasma en el algoritmo **BFD-A** donde se incluye la información del caudal de la resina que se inyecta en cada instante de tiempo. El objetivo del algoritmo **BFD-A** es mejorar la solución del frente de avance obtenida mediante el **BFD**.

El gran inconveniente es el sistema que se obtiene como consecuencia del cálculo del área de la región encerrada entre dos curvas de Bézier. El sistema obtenido es un sistema no lineal y eso implica que no se puede resolver con la inversa de la matriz asociada al sistema de ecuaciones. Como consecuencia de ello hay que aplicar un método iterativo "trust region-dodleg" que se puede resolver con un software como el Matlab. Al recurrir a un método iterativo provoca un aumento del coste de la CPU, así que para futuras líneas de investigación hay que incluir la reducción de este coste. En este caso por la forma en la que se calcula el área de la región encerrada entre dos curvas de Bézier, no es posible aplicar la formulación basada en tensores que utilizamos en el algoritmo **T-BSD**.

Capítulo 4

Aplicación del BSD en Robótica Móvil: BTD y T-BTD.

Tu tiempo es limitado, de modo que no lo malgastes viviendo la vida de alguien distinto. No quedes atrapado en el dogma, que es vivir como otros piensan que deberías vivir. No dejes que los ruidos de las opiniones de los demás acallen tu propia voz interior. Y, lo que es más importante, ten el coraje para hacer lo que te dice tu corazón y tu intuición. Ellos ya saben de algún modo en qué quieres convertirte realmente. Todo lo demás es secundario.

Steve Paul Jobs (1955-2011)

4.1. Introducción.

En este capítulo se adapta de nuevo el **BSD**, pero en este caso el objetivo es poder diseñar una trayectoria flexible para un robot móvil exigiendo que esté libre obstáculos. Este algoritmo recibe el nombre de *Bézier Trajectory Deformation*, **BTD**. La citada trayectoria debe tener en cuenta los cambios del entorno, fusionando el **BTD** con técnicas de planificación de caminos que utilizan métodos reactivos para que el robot móvil tenga una navegación segura.

El inconveniente del **BTD** está en el número de curvas de Bézier utilizadas para obtener una trayectoria lo más precisa posible. A medida que el número Béziers aumenta, también lo hace el tiempo de cómputo y de forma exponencial. Puesto que en robótica móvil las decisiones han de tomarse lo más rápido posible puesto que se toman en tiempo-real, los algoritmos deben ser tan eficientes, computacionalmente hablando, como se pueda. Por ello se ha utilizado una formulación basada en tensores y de esa forma se ha reducido el coste de la CPU a lineal. Adaptando el **T-BSD**, visto en el capítulo

2, a la robótica móvil. En este caso se denomina el algoritmo *Tensor-Bézier Trajectory Deformation*, **T-BTD**.

Los resultados que aquí se han investigado se han publicado en: [70, 72–74] y en revisión están [69, 75].

Las publicaciones [69, 73–75] son las primeras publicaciones en robótica donde aparece el uso de tensores.

El presente capítulo está organizado en diferentes secciones: en la sección 4.2 se ha realizado una pequeña introducción a la robótica para luego pasar a la sección 4.3 donde se realiza un estado del arte del uso de las curvas paramétricas en la robótica móvil; en la sección 4.4 se detallará con un breve resumen una de las técnicas más utilizadas en la evitación de los obstáculos: los campos potenciales (Potential Field, **PF**). En la sección 4.5 se desarrolla la adaptación del **BSD** al **BTD** y la introducción de los tensores obteniendo el **T-BTD**, fusionando ambos con el **PF**; posteriormente en la sección 4.6 se mostraran algunas simulaciones del algoritmo **BTD**, terminando con la sección 4.7 donde se detallarán las conclusiones referentes a este capítulo.

4.2. Introducción a la robótica.

El concepto de robot se remonta casi al principio de la civilización, donde los mitos hablan de seres mecánicos dotados de vida. Durante varios siglos, se han intentado construir máquinas que imitan las tareas humanas asemejando estas máquinas a determinadas partes del cuerpo humano. Por ejemplo, los antiguos egipcios unieron brazos mecánicos a las estatuas de sus dioses. En el caso de la civilización griega aparecen figuras que se mueven mediante poleas y bombas hidráulicas que se usan para propósitos estéticos y artísticos.

La civilización árabe cubre el primero de estos puntos al concebir el robot como un elemento para el confort del ser humano. Un ejemplo es la figura 4.1, es la Fuente del Pavo Real. Un sencillo dispositivo que medía el nivel del agua vertida en un recipiente de forma que, al alcanzarse un cierto umbral, aparecía un autómata portando una pastilla de jabón y, transcurrido un cierto tiempo, una toalla.

Durante el Renacimiento, los estudios de Leonardo da Vinci sobre anatomía del cuerpo humano aportaron un valioso conocimiento a la hora de desarrollar la mecánica del robot. El inicio de la robótica actual puede fijarse en la industria textil del siglo *XVIII*, cuando Joseph Jacquard inventa en 1801 una máquina textil programable mediante tarjetas perforadas. A finales del 1800 se empieza a contemplar de forma científica el concepto de autonomía e inteligencia artificial, cuando Nikola Tesla se propone crear una máquina capaz de tomar sus propias decisiones sin necesidad de un telecontrol. Luego con la Revolución Industrial se impulsaron el desarrollo de muchos agentes mecánicos.

La palabra robot se utilizó por primera vez en 1920 por el escritor checo Karel Capek, en una obra llamada *"Rossum's Universal Robots"*. Su trama trataba sobre un hombre que fabricó un robot y luego este último mata al hombre. El término deriva de la palabra checa *robota*, significa servidumbre o trabajo forzado. Cuando se tradujo al inglés se

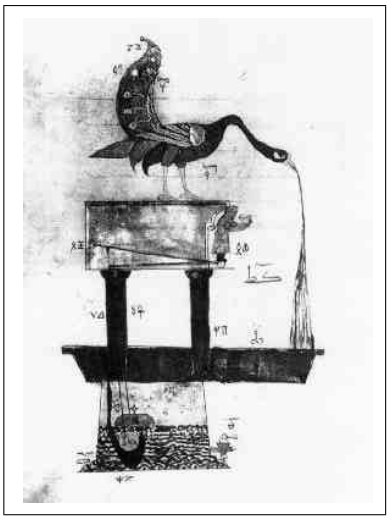


Figura 4.1: Robot en la antigua civilización árabe.

convirtió en el término **robot**.

La palabra robótica fue utilizada por primera vez por el científico y escritor de ciencia ficción de origen ruso Isaac Asimov en 1942. Con él surgen las denominadas ”Leyes de la Robótica”que son las siguientes:

I. *Ley 0:* Un robot no puede realizar ninguna acción, ni por inacción permitir que nadie la realice, que resulte perjudicial para la humanidad, aun cuando ello entre en conflicto con las otras leyes.

II. *Ley 1:* Un robot no puede dañar a un ser humano ni, por inacción, permitir que éste sea dañado.

III. *Ley 2:* Un robot debe obedecer las órdenes dadas por los seres humanos excepto cuando estas órdenes entren en conflicto con las leyes anteriores.

IV. *Ley 3:* Un robot debe proteger su propia existencia hasta donde esta protección no entre en conflicto con las leyes anteriores.

Son varios los factores que intervienen para que se desarrollen los primeros robots en la década de los $50's$. La investigación en inteligencia artificial desarrolló maneras de emular el procesamiento de información humana con computadoras electrónicas e inventó una variedad de mecanismos para probar sus teorías.

Los primeros robots industriales modernos se denominaron ‘Unimates” y se desarrollaron a finales de la década de los $50's$ y principios de los $60's$ ‘por George Devol y Joe Engelberger. Estos robots “Unimate” se basan en la transferencia de artículos programados utilizando principios de control numérico para el control del manipulador. Además fueron robots manejados hidráulicamente. En la figura 4.2 tenemos una imagen de ese tipo de robot.

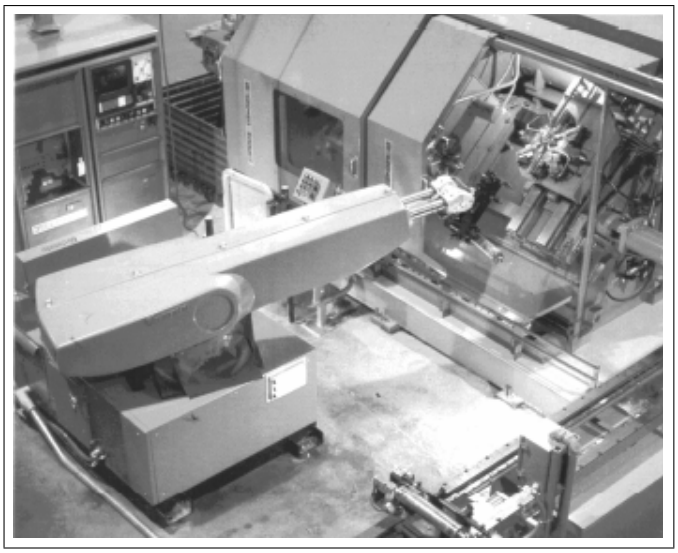


Figura 4.2: Primer robot industrial 1956 (Devol-Engelberger).

En 1961 un robot Unimate se instaló en la Ford Motors Company para atender una máquina de fundición de troquel.

En 1966 Trallfa, una firma noruega, construyó e instaló un robot de pintura por pulverización.

A finales de la década de los $70's$ y principios de los $80's$ el desarrollo de los robots industriales tuvo un rápido desarrollo debido principalmente a grandes inversiones desarrolladas por la industria automotriz.

En 1971 el "Standford Arm", un pequeño brazo de robot de accionamiento eléctrico, se desarrolló en la Standford University.

En 1978 se introdujo el robot PUMA para tareas de montaje por Unimation, basándose en diseños obtenidos en un estudio de la General Motors.

En la actualidad, la robótica se debate entre modelos sumamente ambiciosos, como es el caso del IT, diseñado para expresar emociones, el COG, también conocido como el robot de cuatro sentidos, el famoso SOUJOURNER o el LUNAR ROVER, vehículo de turismo con control remotos, y otros mucho más específicos como el CYPHER, un helicóptero robot de uso militar, el guardia de tráfico japonés ANZEN TARO o los robots mascotas de Sony.

En general la historia de la robótica la podemos clasificar en cinco generaciones: las dos primeras, ya alcanzadas en los ochenta, incluían la gestión de tareas repetitivas con autonomía muy limitada. La tercera generación incluiría visión artificial, en lo cual se ha avanzado mucho en los ochenta y noventa. La cuarta incluye movilidad avanzada en exteriores e interiores y la quinta entraría en el dominio de la inteligencia artificial en lo que se está trabajando actualmente.

En la actualidad, el concepto de robótica ha evolucionado hacia dos tipos de robots claramente diferenciados dentro de la clasificación holonómica. La holonomicidad es una característica que depende de la movilidad del robot.

Para entender la citada clasificación previamente necesitamos definir los grados de libertad de un robot.

Grados de libertad (**DOF**): se dice que el número de grados de libertad de un robot es el número de magnitudes que pueden variarse independientemente. El control está relacionado con los grados de libertad. En general cada grado de libertad requiere de un actuador (mecanismo real que genera el movimiento de los elementos del robot). Si disponemos de un actuador por cada DOF, diremos que todos los DOF son controlables. Habitualmente existe un actuador por grado de libertad, pero no siempre es así.

Con ello podemos clasificar los sistemas holonómicos y no holonómicos:

- **Robots holonómicos**: Diremos que un robot es holonómico si tiene los mismos grados de libertad efectivos que controlables. Por ello son robots formados por ligaduras, como los brazos y manipuladores (robots de brazo o robots de ejes): encuentran su aplicación en la industria, para entornos, en general estructurados.

- **Robots no-holonómicos**: Un robot es no holonómico si tiene menos grados de libertad controlables que número total de grados de libertad. La aplicación de estos robots está en la industria, para el trasporte de materiales, en terrenos abruptos o peligrosos para el ser humano.

En la figura 4.3 podemos ver la clasificación gráficamente.

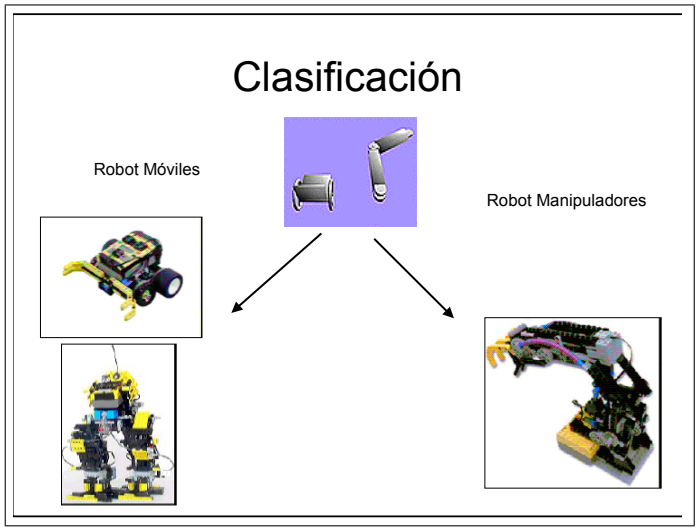


Figura 4.3: Clasificación holonómica de los robots.

En [106, 140, 159] se puede encontrar más información acerca de la robótica.

Viendo esta pequeña introducción sobre la robótica y su historia, destacamos que un robot móvil autónomo es aquel que es capaz de desenvolverse por sí mismo en entornos desconocidos y parcialmente cambiantes. Estos sistemas autónomos son capaces de extraer información del medio en que desarrollan su actividad mediante sensores y pueden alterar su comportamiento de forma dinámica de acuerdo a la información obtenida.

El simple hecho de moverse desde una posición inicial a una posición final u objetivo involucra multitud de tareas y cada una de ellas en sí misma un campo de investigación dentro de la robótica: evitación de colisiones, fusión sensorial, generación del mapa y autolocalización, generación de trayectorias, sistemas de control, etc. Uno de los tópicos anteriores más investigados es la definición de trayectorias o caminos y la evitación de los obstáculos.

En robótica diferenciamos entre *Trayectorias* y *Caminos*.

I. Una *trayectoria* en el plano es una curva temporal para cada una de las coordenadas sobre las que se debe realizar un control del robot $(x(t), y(t), \theta(t))$.

II. En cambio, un *camino* es una curva definida en el espacio cartesiano o de configuración que debe seguir el robot sobre la que también se realiza control del mismo. Sin embargo, en este caso no se tiene en cuenta el factor tiempo, $C = (x, y, \theta)$.

La diferencia entre el seguimiento de una trayectoria y de un camino está en que en el camino no se tiene en cuenta la variable tiempo y por tanto se busca en todo momento un punto objetivo del camino, según la posición real del robot. El inconveniente del robot móvil está en la no colisión con los obstáculos y más teniendo en cuenta que podemos tener un entorno dinámico. Para el control de la trayectoria se debe replanificar en el caso de que aparezca un obstáculo, y eso puede ser costoso. En cambio, si se utiliza un seguimiento de caminos, se puede aumentar y disminuir la velocidad del robot móvil para poder adelantar o dejar pasar el obstáculo. La principal dificultad reside en definir caminos de forma paramétrica y calcular un punto objetivo.

Aprovechando el desarrollo del algoritmo **BSD** nuestro objetivo es poder utilizarlo y adaptarlo para poder planificar una trayectoria (determinando dónde y cómo iremos de un punto a otro). Para poder planificarlo se distinguen tres grandes tipos de tareas:

- Planificación de trayectorias.
- Detección de colisiones.
- Evitación de colisiones.

Teniendo en cuenta estas tres tareas principales, adaptaremos el algoritmo **BSD** para que pueda ser utilizado en robótica.

4.3. Generación de trayectorias para la robótica móvil mediante curvas paramétricas.

Ya hemos visto en la sección anterior 4.2 que predecir el movimiento de un robot es muy importante. La idea radica en como conseguir una trayectoria adecuada. El simple hecho de mover un robot móvil desde una posición inicial (x_i, y_i, θ_i) a una posición final

(x_g, y_g, θ_g) de forma segura involucra muchos campos de investigación. Todos estos campos de investigación se deben combinar para poder generar una algoritmo que planifique el movimiento del robot móvil de forma eficiente.

Muchos investigadores consideran que las curvas paramétricas son muy útiles para la construcción de las trayectorias de los robots con ruedas. Esto es debido a las propiedades ventajosas que tienen las curvas paramétricas para mejorar las trayectorias producidas por la planificación del camino. En el caso de las curvas de Bézier, las propiedades que podemos destacar son:

I. La segunda derivada es continua: por lo tanto se genera una trayectoria suave.

II. Una curva de Bézier siempre pasa por el primer y último punto de control: de esta forma tenemos controlada la posición inicial y final del robot.

III. Una curva de Bézier siempre está contenida dentro de la envolvente convexa: así se puede planificar la trayectoria de forma más eficiente y sin colisiones con los obstáculos puesto que tenemos controlada la curva en una determinada región que es la envolvente convexa.

IV. El cálculo de la derivada es sencillo: con esto se puede controlar la orientación del robot móvil cuando está controlado en la trayectoria planificada.

V. El comienzo y final de la curva es tangente al polígono de control correspondiente: es tangente al vector de la diferencia $\mathbf{P}_1 - \mathbf{P}_0$ en el primer punto de control y el vector de la diferencia $\mathbf{P}_n - \mathbf{P}_{n-1}$ en el último punto de control.

VI. Alta flexibilidad: la curva puede ser modificada libremente cambiando los puntos de control. Nos permite manipular la trayectoria.

VII. La curva es un segmento recto si, y sólo si, todos los puntos de control están alineados: la trayectoria óptima en principio es una línea recta, resulta sencillo obtenerla.

Las curvas paramétricas más utilizadas en el campo de la robótica móvil son las B-Splines, las NURBS, las Bézier y las Rational Bézier. En el apéndice A están las definiciones correspondientes a estas curvas paramétricas más utilizadas en el ámbito CAGD y en este caso la robótica móvil. La diferencia entre ellas está en la complejidad de su definición. Las curvas de Bézier son las más simples y poseen numerosas propiedades matemáticas que facilitan su manipulación y análisis.

Veamos las publicaciones más destacadas relacionadas con las curvas paramétricas y la robótica móvil.

4.3.1. Publicaciones de robótica relacionadas con el uso de las B-Spline.

En 1989, en [94] incorporaron las curvas B-Splines en el diseño de la trayectoria. En dicha publicación añadían segmentos para generar la trayectoria entera cerca de la deseada. Esta trayectoria no pasaba a través de los puntos exactos.

Más tarde, en 1994, encontramos la publicación [162] donde vuelven a utilizar las curvas B-Splines para la planificación del camino pero añadiendo una variable temporal. En este caso, la velocidad del robot está controlada por la misma B-Spline. El mismo año, encontramos otro artículo [171] en el que simulaban una curva B-Spline con una trayectoria *fuzzy* de control. En 1999, en el trabajo [51] también utilizaban la curva B-Spline para calcular la trayectoria del robot móvil. En este caso se generaban muchos puntos con una spline y de esa forma permitían que el robot siguiese los puntos en forma de sucesión. El mismo año en [170] consideraron las restricciones dinámicas y cinemáticas basadas en la planificación de un camino utilizando curvas B-Spline para encontrar la trayectoria temporal óptima en un entorno estático.

Posteriormente, en el año 2007, aparecen publicados los siguientes artículos: [42–44]. En estos trabajos se desarrolla un método para resolver los problemas de la planificación de caminos utilizando Splines cúbicas para evitar los obstáculos. Este método refina el camino a seguir de forma iterativa para calcular un camino factible y por lo tanto poder construir un camino libre de obstáculos en tiempo real cuando el entorno no es estructurado.

En [44] se detalla como se implementa la planificación del camino basada con B-Splines. El uso de las Splines nos permite restringir los polinomios ya que la primera derivada de $P_1, \cdots, P_{n-1}$ es continua a través de toda la frontera. Además, algunas restricciones pueden forzarse en el primer y último punto para forzar un valor particular de la derivada. Estas características de las Splines ofrecen muchas propiedades ventajosas para planificar un camino adecuado. Si se impone un valor de la derivada, se puede generar un camino empezando por una posición y teniendo una dirección impuesta por el valor de la derivada. Por lo tanto, se pueden generar e inicializar desde la posición actual y dirección del vehículo. La primera derivada es proporcional a la dirección del vehículo, entonces se podría obtener una derivada no continua y como consecuencia un camino no factible para ese tipo de vehículo. Pero la segunda derivada es proporcional a la dirección del ángulo y algunas discontinuidades podrían forzar que el vehículo se pare en cada punto de control para ajustar su dirección.

Las curvas B-Splines permiten una fácil construcción de caminos suaves a través de los puntos de control. Para poder evitar los obstaćulos, introducen puntos de control cerca de ellos y se desarrollan métodos para mover estos puntos de control lejos de los obstáculos y en zonas libres de ellos. Moviendo los puntos de control, podemos deformar la curva para atravesar el entorno de forma segura.

Métodos anteriores a éste, también trabajan con Splines para generar un camino suave y además que evite los obstáculos [16, 144]. Pero estos métodos anteriores necesitan un alto coste computacional para evaluar el camino entero. Ver la figura 4.4 como ejemplo gráfico del algoritmo generado en [44]. En [42] se analiza el tiempo computacional y la viabilidad del algoritmo dado que se ejecuta con un método iterativo. Con las simulaciones del método Monte Carlo indican un grado elevado de éxito para entornos complejos. Además miden el *running time* que se incrementa con la complejidad del entorno. Por último, en [43] se estudian los resultados experimentales del algoritmo.

El inconveniente de este algoritmo es su forma de computar la solución porque para

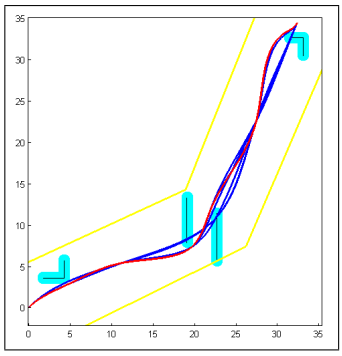


Figura 4.4: Iterativamente generan curvas suaves (azul) curvándolas alrededor de los obstáculos hasta que se encuentra un camino libre de obstáculos (rojo).

encontrar el camino libre de obstáculos recurre a un método iterativo que siempre aumentará su tiempo de cómputo respecto de otros algoritmos que no sean iterativos.

4.3.2. Publicaciones de robótica relacionadas con el uso de las NURBS.

Este tipo de curvas paramétricas se utilizan en la reconstrucción de trayectorias para generar caminos suaves que aproximen el movimiento real del robot. En el capítulo 2 ya se vieron publicaciones como la de [132] donde se destacan las ventajas y desventajas de las curvas NURBS, haciendo un estudio detallado de sus propiedades. En artículos relacionados dentro del ámbito de la robótica, en [152] se destacan las propiedades ventajosas de las NURBS para la planificación de caminos tanto en el plano como en el espacio.

En otras publicaciones como [3–5] han estado trabajando con curvas NURBS para aproximar o describir la trayectoria descrita por el brazo robot PUMA 560. Para obtener el comportamiento del robot, utilizan el *Programming by Demonstration* (PbD), una buena solución para transferir de forma automática el conocimiento de un humano a un robot. La desventaja de este algoritmo radica en que esta trayectoria NURBS no garantiza la evitación de los obstáculos.

4.3.3. Publicaciones de robótica relacionadas con el uso de las Rational Bézier (RBC).

En el artículo [116] se presenta una metodología off-line para aproximar una Clotoide (Integrales de Fresnel) a una RBC. Posteriormente y como continuación de esta publicación encontramos el artículo [115] donde se presenta un método para obtener trayectorias en tiempo real con Clotoides. Para ello hay dos pasos: definir de forma off-line aproximaciones de las Clotoides con curvas Rational Bézier (RBC) y generar caminos on-line escalando, rotando y trasladando la formulación previa que se ha realizado off-line. Una de las ventajas de este método es la realización del cálculo off-line, ya que eso reduce considerablemente el tiempo de cómputo. En todo el proceso, los coeficientes de peso

y puntos de control se mantienen invariantes. En este trabajo se garantiza que una RBC tiene el mismo comportamiento que una Clotoide utilizando un bajo orden para la curva.

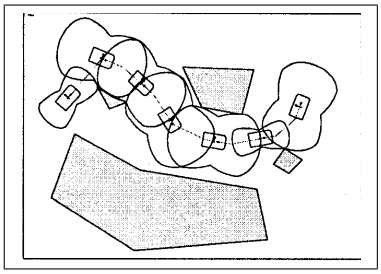
4.3.4. Publicaciones de robótica relacionadas con el uso de las Bézier.

Las primeras publicaciones relevantes en la robótica utilizando curvas de Bézier se publican en 1997 y 1998. Estos artículos, [82, 86], combinan la planificación del camino y el control reactivo para un robot móvil no-holonómico. En él, se propone el concepto de "Bubble Band" (camino de burbujas). Con una métrica apropiada se conectan las "bubles" con curvas de Bézier generando una trayectoria. Estas "bubbles" son el máximo espacio libre que puede ser alcanzado en cualquier dirección sin riesgo de colisión. Esto es debido a la propiedad de la envolvente convexa e implica que si los puntos de control están dentro de la "bubble", entonces la aproximación del camino está dentro de la "bubble". El algoritmo descrito en esta publicación está implementado con un sistema llamado "Kinan" (Kinematics Integrated in nonholonomic Autonomous Navigation). Un planificador, utilizando un modelo del entorno, genera una trayectoria conectando las posiciones iniciales y finales, y dicha trayectoria no necesariamente tiene que ser la adecuada. De esta forma se pasa a un módulo de "bubble band". En un primer paso, el algoritmo generado para la "bubble band" genera una sucesión de bubbles conectando los dos extremos, reemplazando la trayectoria original, ver figura 4.5(a). La banda generada se expone a las fuerzas que hay en el entorno, figura 4.5(b). Como consecuencia, la banda es modificada, figura 4.5(c).

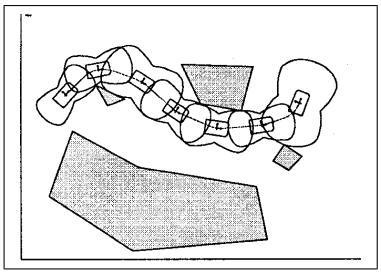
Posteriormente a esta publicación, en 2001 encontramos otra publicación [61] relacionada con este concepto de las "bubble band". En este caso se parte de un mapa del entorno estático y luego también el algoritmo se lleva a un entorno con obstáculos dinámicos. Otra publicación [124] de ese mismo año utiliza también las curvas de Bézier para el algoritmo que planifica el camino del robot de forma local. Inicialmente se genera un algoritmo que planifique el camino utilizando la teoría GVG (Generalized Voronoi Graph) y luego se deforma suavemente maximizando la evaluación de una función definida en el artículo. Los candidatos que se obtienen como un camino suave, se expresan con las curvas de Bézier.

En el año 2003 se publica [81] donde se presenta una nueva interface utilizando una pantalla táctil para controlar el robot móvil, evitando los obstáculos en tiempo real. En esta publicación se desarrollan dos algoritmos: uno en el que se extraen una sucesión de puntos importantes y un segundo algoritmo que genera una trayectoria utilizando curvas de Bézier cúbicas. En la figura 4.6 podemos ver un esquema de este algoritmo.

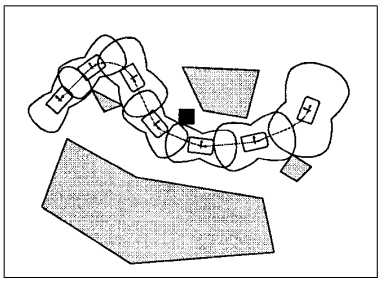
En 2007 encontramos el trabajo [146], en él se desarrolla un algoritmo que contempla la evitación de colisiones cooperativa cuando se tienen varios robots móviles no-holonómicos. Relacionados con esta publicación anteriormente tenemos estos dos artículos [60, 99]. En [146] la evitación cooperativa de colisiones se desarrolla considerando que los robots están cambiando su camino cooperativo para conseguir la posición final. Se desarrollan dos tareas: la primera es la planificación del camino basada en Bézier para cada robot de forma individual consiguiendo su posición final, y la segunda es controlar



(a) Primera creación de la "bubble band".



(b) "Bubble band" bajo las fuerzas.



(c) La deformación de la "bubble band" para evitar un obstáculo móvil.

Figura 4.5: Imágenes de la publicaciones [82, 86]

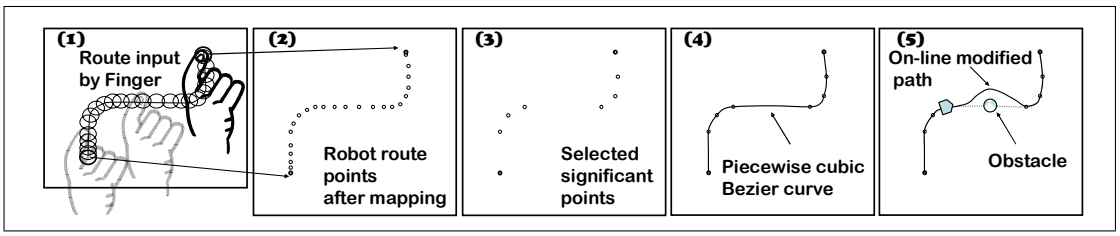


Figura 4.6: Esquema del algoritmo publicado en [81].

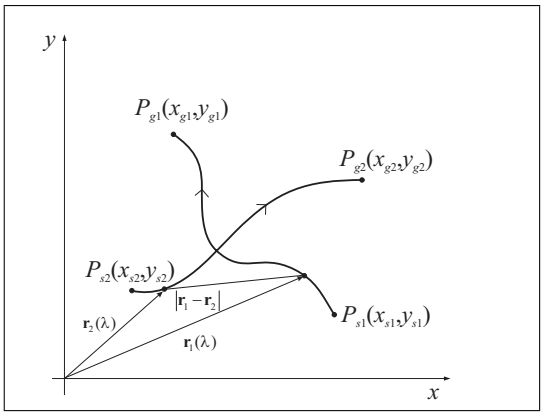


Figura 4.7: Evitación de los obstáculos basada en las curvas de Bézier.

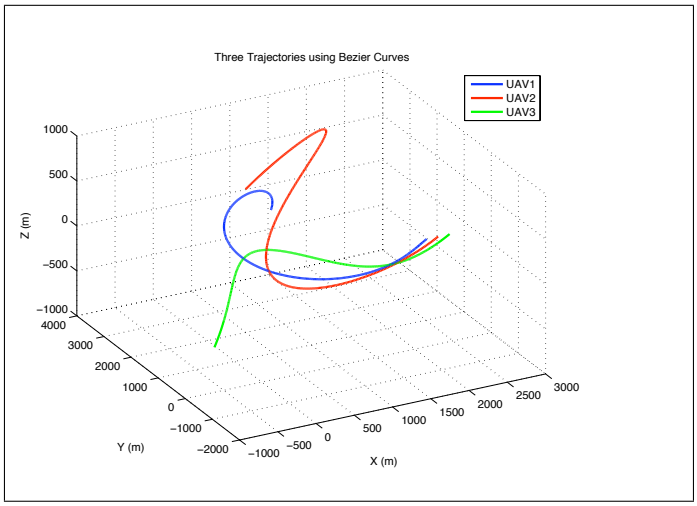


Figura 4.8: Simulación de la trayectoria de vuelo.

la trayectoria para obtener un camino óptimo minimizando la función "penalty" (teniendo en cuenta la suma de los tiempos máximos sujetos a las distancias entre los robots). En la figura 4.7 tenemos un ejemplo de este algoritmo considerando dos robots móviles.

Relacionado con los UAVs (Unmanned Aerial Vehicle) y las curvas de Bézier $3D$ está el artículo [105] publicado en 2008. En esta publicación se presenta un marco preliminar que genera trayectorias en el espacio para múltiples UAVs utilizando curvas de Bézier. El algoritmo para poder generar las trayectorias se resuelve como un problema de optimización restringida. En este caso, la función a optimizar penaliza una longitud excesiva. El UAV debe utilizar el camino más corto posible. Las restricciones son las distancias entre los múltiples UAVs. El sistema no es lineal y se aplican métodos numéricos para resolverlo. En la figura 4.8 tenemos un ejemplo de las trayectorias generadas con tres UAVs.

Uno de los autores que recientemente tiene diferentes publicaciones relacionadas con las trayectorias de robots móviles diseñadas a partir de curvas de Bézier es *Choi et al.*

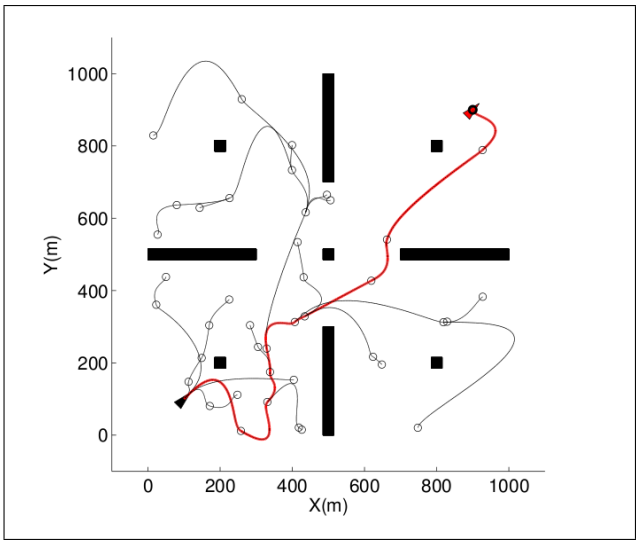


Figura 4.9: *Rapidly-exploring random trees* utilizando curvas de Bézier de orden siete. La curva roja representa el camino final del árbol.

Algunas de las publicaciones que podemos encontrar incluida su Tesis Doctoral son: [31–39]. En muchas de ellas, plantea un problema de optimización restringida donde la función a optimizar es la curvatura de la curva de Bézier.

Finalmente, también podemos encontrar este artículo [125] donde se presenta una metodología basada en la variación del *Rapidly-exploring Random Trees (RRTs)* que genera trayectorias adecuadas para vehículos autónomos con restricciones holonómicas en entornos con obstáculos. Este algoritmo está basado en el uso de curvas de Bézier de séptimo orden que conectan los vértices del árbol. De esa forma, se generan caminos que no violan la restricción principal cinemática del vehículo. La suavidad de la aceleración del camino entero está garantizada al controlar los valores de la curvatura de los puntos extremos de cada curva de Bézier que compone el árbol. El algoritmo propuesto proporciona una rápida convergencia al resultado final. Además se reduce el número de vértices del árbol porque el método permite las conexiones entre los vértices del árbol con un rango ilimitado. Las propiedades de las curvas de Bézier de séptimo orden también se utilizan para evitar los obstáculos estáticos en el entorno. Está técnica se simuló con un pequeño UAV: aqVS, desarrollado en la Universidad Federal de Minas Gerais (Brasil). En la figura 4.9 tenemos un ejemplo de una simulación de este artículo.

Realizado este estudio exhaustivo del uso de las curvas paramétricas, queda constancia de su importancia en el diseño de trayectorias de un robot móvil. Hay que destacar las curvas de Bézier por la simplicidad de su definición y su fácil manejo y manipulación.

No sólo es importante definir la trayectoria del robot sino que además tiene que evitar los obstáculos del entorno. En consecuencia, la trayectoria inicial debe ser modificada en tiempo real para que el robot móvil evite los posibles obstáculos dinámicos con los que se encuentre.

Con el algoritmo **BSD** definido en el capítulo 2 se tiene la posibilidad de deformar

una curva de Bézier a través de un campo de vectores. Gracias a la construcción del **BSD** permite emplearlo en la robótica móvil. Tan sólo hay que introducir el parámetro temporal en la Bézier para transformarlo en trayectoria y conseguir un campo de vectores que modifique la trayectoria inicial. Es en este momento cuando entran en juego los métodos reactivos o campos potenciales que generan caminos para el robot libres de obstáculos. En estos métodos el movimiento del robot viene determinado por fuerzas repulsivas asociadas a los obstáculos y fuerzas atractivas hacia la posición final del robot móvil.

En la siguente sección 4.4 de este capítulo se van a detallar el uso de los Campos Potenciales o métodos reactivos, *Potential Field* (**PF**).

4.4. Campos Potenciales: PF y PFP.

El algoritmo **BSD** necesita una técnica que genere una trayectoria inicial y un campo de vectores que la modifiquen cuando se detecte un obstáculo en el entorno. El objetivo es generar una nueva trayectoria libre de obstáculos. Para ello se puede utilizar cualquiera de las técnicas de Campos Potenciales Artificiales (**PF**) que se utilizan para la planificación de caminos libre de obstáculos.

Estos métodos **PF** son muy populares en el diseño de movimientos (*path planning*) por su sencillez y por la rapidez de los cálculos implicados, lo que hace que sean adecuados para aplicaciones de navegación autónoma en tiempo real. La idea imaginaria de fuerzas actuando sobre un robot fue sugerida por Andrews and Hogan [11] en 1983 y Khatib [87] en 1985, [95]. Estos métodos reactivos generan una respuesta rápida ante los posibles cambios que nos podemos encontrar en el entorno. La primera técnica que podemos encontrar es el método de Campos Potenciales Artificiales, desarrollada en la publicación [87].

Esta técnica de planificación de movimientos, representa al robot como una partícula bajo la influencia de un campo artificial cuyas variaciones locales reflejan la estructura del espacio libre de obstáculos. La función potencial habitualmente se define como la suma de dos campos potenciales: uno atractivo, que atrae al robot hacia el destino final y uno repulsivo que empuja al robot lejos de los obstáculos. Se realiza de forma iterativa, [118]. En la figura 4.10 podemos ver los posibles campos o fuerzas que se generan en los objetos que encontramos en una superficie por la que se moverá el robot móvil.

Para utilizar el algoritmo **BSD** se ha aplicado una de las últimas y novedosas técnicas de los campos potenciales descrito en [118–121] denominada la Propagación del Campo Potencial (Potential Field Projection, **PFP**).

Este método está basado en la combinación de los métodos clásico **PF** [87] y los filtros de Kalman multifrecuencia (*Multi-rate Kalman filters*,**MKF**) [134, 155].

El **PFP** modifica la técnica de Campos Potenciales Artificiales en su definición del potencial para incluir las trayectorias futuras tanto del robot como de los obstáculos, mediante la incorporación de filtros de Kalman multifrecuencia y la utilización de modelos esféricos para modelar los objetos así como la incertidumbre asociada a su movimiento, que deriva de la utilización de la estimación antes mencionada. La predicción de trayec-

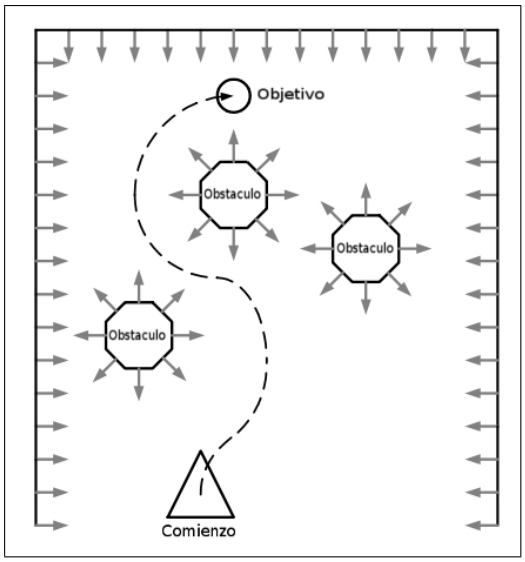


Figura 4.10: Posibles campos potenciales generados por los diferentes objetos de una superficie.

torias mediante filtros de Kalman multifrecuencia supone considerar la cinemática en la evitación de obstáculos así como la información multifrecuencia proporcionada por los sensores (que suelen trabajar a distintos períodos).

El resultado es un campo potencial que guía al robot hacia el destino evitando la colisión con obstáculos fijos y móviles. Las ventajas de esta técnica frente a otras existentes radican en que se considera el movimiento futuro de los objetos (si son móviles) y se utiliza la incertidumbre derivada de la predicción de las trayectorias de los mismos para obtener un volumen donde es probable que se encuentre el objeto en cada momento. Este volumen de incertidumbre se emplea, asimismo, en el cálculo del campo potencial dando como resultado unos vectores de desplazamiento donde debería de posicionarse el robot en cada instante de tiempo para conseguir una posición segura.

Con el **PFP** se genera la predicción de unos puntos discretos en un horizonte de predicción T_h para el robot teniendo en cuenta su modelo cinemático y las fuerzas atractivas generadas por el campo potencial en el momento presente ($j = 0$) y los instantes futuros ($j > 0$). Además se predicen las trayectorias de los obstáculos, en ellos se generan fuerzas repulsivas que se aplican sobre el robot móvil para evitar que colisione con los obstáculos.

El modelo cinemático de segundo orden se ha utilizado para modelar el movimiento del robot y de los obstáculos, ecuación 4.1 y 4.2. Donde T es el período de control o estimación, el vector $\mathbf{x} \in \mathbb{R}^4$ es el estado a estimar en el instante j-th ($j \in [0, \cdots, T_h/T]$) y está compuesto por las posiciones (x, y) y la velocidades (v_x, v_y) en el plano de coordenadas XY y la entrada de control es el objeto de la aceleración (a_x, a_y). Las ecuaciones 4.1 y 4.2 corresponden a la representación del espacio lineal del estado 4.3, donde $\mathbf{u} \in \mathbb{R}^2$ son las entradas de control, $\mathbf{z} \in \mathbb{R}^2$ es la medida de los sensores, $\mathbf{w} \in \mathbb{R}^4$ es el ruido del proceso y $\mathbf{v} \in \mathbb{R}^2$ es la medida del ruido del proceso. Por otra parte, $\mathbf{D} \in \mathbb{R}^{4\times4}$, $\mathbf{E} \in \mathbb{R}^{4\times2}$ y $\mathbf{F} \in \mathbb{R}^{2\times4}$ son las matrices de representación del sistema lineal en el espacio de estados.

$$\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_{j+1} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_j + \begin{bmatrix} T^2/2 & 0 \\ 0 & T^2/2 \\ T & 0 \\ 0 & T \end{bmatrix} \cdot \begin{bmatrix} a_x \\ a_y \end{bmatrix} \tag{4.1}$$

$$\begin{bmatrix} x \\ y \end{bmatrix}_j = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_j \tag{4.2}$$

La predicción de las posiciones futuras y sus incertidumbres son obtenidas de las ecuaciones de predicción del filtro de Kalman multifrecuencia 4.4 para cada objeto en el entorno, donde $\widehat{\mathbf{x}} \in \mathbb{R}^4$ es el vector del estado estimado, $\mathbf{S} \in \mathbb{R}^{4\times4}$ es la varianza del error de estimación, $\mathbf{K} \in \mathbb{R}^{4\times2}$ es la ganancia de Kalman y $\mathbf{Q} \in \mathbb{R}^{4\times4}$ y $\mathbf{R} \in \mathbb{R}^{2\times4}$ son, respectivamente, las varianzas de los ruidos de proceso y de medida.

$$\begin{aligned} \mathbf{x}_{j+1} &= \mathbf{D}\cdot\mathbf{x}_j + \mathbf{E}\cdot\mathbf{u}_j + \mathbf{w}_j \\ \mathbf{z}_j &= \mathbf{F}\cdot\mathbf{x}_j + \mathbf{v}_j \end{aligned} \tag{4.3}$$

La función delta Δ modifica la expresión de la ganancia de Kalman indicando la presencia (matriz Δ unidad) o la ausencia (matriz Δ nula) de medidas en un instante particular de estimación j. Las predicciones de las posiciones futuras se obtienen al imponer las matrices Δ nulas para instantes futuros, no se pueden obtener como medidas.

$$\begin{aligned} \widehat{\mathbf{x}}_{j+1/j} &= \mathbf{D}\cdot\widehat{\mathbf{x}}_{j/j} + \mathbf{E}\cdot\mathbf{u}_j \\ \mathbf{S}_{j+1/j} &= \mathbf{D}\cdot\mathbf{S}_{j/j}\cdot\mathbf{D}^T + \mathbf{Q} \\ \mathbf{K}_j &= \mathbf{S}_{j/j-1}\cdot\mathbf{C}^T\cdot[\mathbf{C}\cdot\mathbf{S}_{j/j-1}\cdot\mathbf{C}^T + \mathbf{R}]^{-1}\cdot\Delta_j \\ \widehat{\mathbf{x}}_{j/j} &= \widehat{\mathbf{x}}_{j/j-1} + \mathbf{K}_j\cdot[\mathbf{z}_j - \mathbf{C}\cdot\widehat{\mathbf{x}}_{j/j-1}] \\ \mathbf{S}_{j/j} &= \mathbf{S}_{j/j-1} - \mathbf{K}_j\cdot\mathbf{C}\cdot\mathbf{S}_{j/j-1} \end{aligned} \tag{4.4}$$

La predicción de las posiciones y sus incertidumbres se utilizan en el cálculo del campo potencial $U_j(\mathbf{x}) = U_{att,j}(\mathbf{x}) + U_{rep,j}(\mathbf{x})$ que guiará al robot a la posición final evitando los obstáculos del entorno. Este campo potencial está compuesto por una componente atractiva $U_{att,j}(\mathbf{x})$ que va ligada a la posición final y una componente repulsiva $U_{rep,j}(\mathbf{x})$ que se genera al detectar los obstáculos y sus incertidumbres, que están consideradas como áreas restringidas para la planificación del camino. Ambas componentes están definidas en [121] incluso en los instantes de tiempo sin medida del entorno ($j > 0$). Estos campos potenciales generan fuerzas en cada instante de tiempo de predicción j. Por otra parte, el campo de fuerzas atractivas $\mathbf{F}_{att,j}(\mathbf{x}) = -\nabla U_{att,j}(\mathbf{x})$ se transforman en aceleraciones por el modelo dinámico de partícula y, de esta forma, están considerados entradas de control en el ciclo de predicción del filtro de Kalman multifrecuencia 4.4, porque las fuerzas atractivas tiene la estructura de un controlador PD. Por lo tanto, se obtiene un conjunto de posiciones de predicción que conducirán al robot hacia la posición final si no hubiese ningún obstáculo.

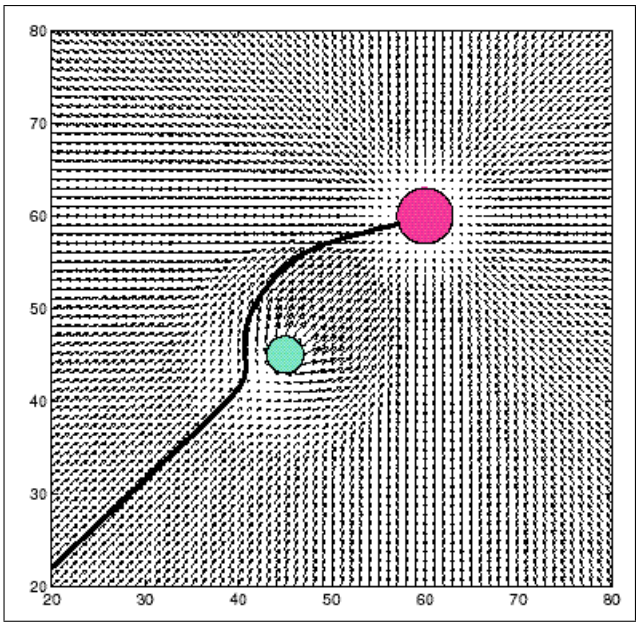


Figura 4.11: *Trayectoria del robot atraída por la posición final y con las fuerzas repulsivas evitando los obstáculos.*

Por otra parte, esta predicción se modifica teniendo en cuenta las fuerzas repulsivas $\mathbf{F}_{rep,j}(\mathbf{x}) = -\nabla U_{rep,j}(\mathbf{x})$. En la figura 4.11 tenemos un ejemplo del campo de fuerzas que se genera en el obstáculo y el punto final u objetivo del robot móvil. Podemos observar como el robot tiene que modificar su trayectoria original para no colisionar con el obstáculo del entorno.

4.5. Deformación de las trayectorias mediante vectores de repulsión: BTD y T-BTD.

En la robótica móvil se han observado dos problemas o necesidades principales a la hora de planificar el camino de un robot móvil, estas necesidades son:

- En primer lugar, la necesidad de definir inicialmente la trayectoria de un robot móvil a través una curva continua, suave y fácil de manipular. En los capítulos anteriores se ha visto que las propiedades de las curvas paramétricas reúnen estas condiciones para poder representar la trayectoria a través de alguna de las curvas más utilizadas en CAGD.

- En segundo lugar, la posibilidad de modificar esa trayectoria inicial puesto que las condiciones del entorno para el robot móvil varían y se puede encontrar con obstáculos dinámicos.

El algoritmo **BSD** proporciona la posibilidad de definir la trayectoria del robot móvil a través de una curva de Bézier y luego poder manipularla mediante las fuerzas repulsivas obtenidas con uno de los métodos de los **PF**.

De esta forma definiremos el llamado *Bézier Trajectory Deformation*, **BTD**.

El conjunto de los puntos discretos, obtenidos a través de cualquiera de las técnicas **PF**, que forman parte de la predicción de posiciones, son considerados como los Puntos Iniciales, S_i, que necesitamos en la curva de Bézier original, ver sección 2.2.

Los Puntos Iniciales formaran parte de la trayectoria de referencia para el algoritmo **BTD**. Posteriormente, el conjunto de fuerzas repulsivas obtenidas mediante el **PF** se transformaran en desplazamientos mediante el modelo dinámico de partícula, y de esa forma el **BTD** modificará la trayectoria Bézier original obteniendo la trayectoria Bézier modificada libre de obstáculos. Con ello obtendremos el algoritmo fusionado **BTD+PF**. En la figura 4.12 podemos ver un esquema del algoritmo fusionado.

La reducción del tiempo de cómputo del algoritmo **BTD** se consigue introduciendo el uso de tensores en el algoritmo, desarrollando en este caso el *Tensor-Bézier Trajectory Deformation*, **T-BTD**. La solución que se obtiene con el **T-BTD** es la misma que con el **BTD**, pero utilizando un coste computacional considerablemente inferior, visto ya en la sección 2.4. Esto proporciona una gran ventaja y es poder aumentar la precisión de la trayectoria obtenida ya que podemos aumentar el uso de curvas de Bézier concatenadas y el coste computacional tan sólo crecerá de forma lineal, evitando el crecimiento exponencial que se consigue con el **BTD**.

En la siguiente sección vamos a adaptar el **BSD** al **BTD** para posteriormente detallar la fusión **BTD+PF**.

4.5.1. Adaptación del BSD al BTD.

1. **Definición de la trayectoria Bézier y su orden.**

   Una curva de Bézier tiene un parámetro intrínseco, u, que es no dimensional, ver definición 2.1. Uno de los objetivos de este capítulo es que la curva Bézier represente la trayectoria de un robot. Para ello el parámetro intrínseco debe ser definido como una variable temporal. De esa forma se asocia la posición de cada curva (posición del robot) con un instante de tiempo $u \in [u_0, u_f]$, donde u_0 y u_f representan los tiempos iniciales y finales de la trayectoria. La definición de la trayectoria Bézier es:

   $$\boldsymbol{\alpha}(u) = \sum_{i=0}^{n} \mathbf{P}_i \cdot B_{i,n}(u) \,; u \in [u_0, u_f] \tag{4.5}$$

   Siendo n el orden, $\mathbf{P}_i$ los puntos de control y $B_{i,n}(u)$ las Bases de Bernstein que se definen como sigue,

   $$B_{i,n}(u) = \binom{n}{i} \left(\frac{u_f - u}{u_f - u_0}\right)^{n-i} \cdot \left(\frac{u - u_0}{u_f - u_0}\right)^{i} ; i = 0, 1, \cdots, n \tag{4.6}$$

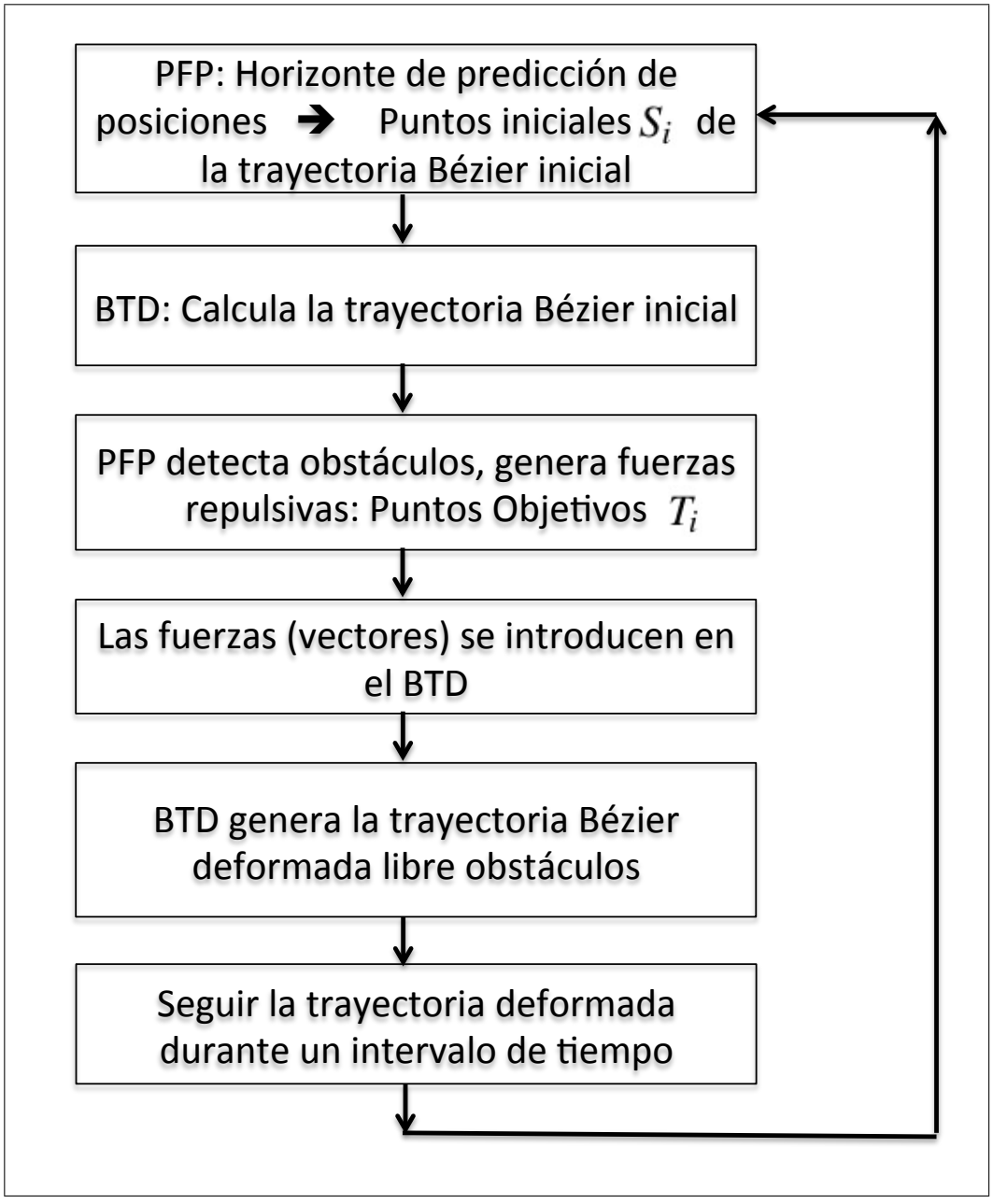


Figura 4.12: Esquema del algoritmo **BTD+PF**.

Como la trayectoria Bézier no debe tener lazos, figura 2.19, el orden de las Béziers debe ser cuadrático como ya justificamos en la sección 2.2.

Esa trayectoria Bézier inicial se modificará evitando los obstáculos móviles del entorno: el algoritmo **BSD** se adapta para ser **BTD**, modificando la posición de los puntos de control desde una posición inicial a la nueva, que es la que impondrá el algoritmo de evitación de obstáculos (en este caso el **PFP**).

El desplazamiento de cada punto de control $\mathbf{P}_i$ se denota como ε_i, de forma que el vector $\underline{\underline{\varepsilon}} = [\varepsilon_0, \cdots, \varepsilon_n]$ es el desplazamiento de todos los puntos de control que definen la trayectoria Bézier. Como vimos en el capítulo 2 en la ecuación 2.5 se definía la curva de Bézier modificada, en el algoritmo **BTD** se define la nueva trayectoria Bézier modificada, $S_{\varepsilon}(\boldsymbol{\alpha}(u))$, como sigue:

$$S_{\varepsilon}(\boldsymbol{\alpha}(u)) = \sum_{i=0}^{n} (\mathbf{P}_i + \varepsilon_i) \cdot B_{i,n}(u) \, ; u \in [u_0, u_f] \tag{4.7}$$

En consecuencia, la función a optimizar que se utilizará para resolver el problema se define de forma análoga a la vista en 2.6, tan sólo hay que cambiar el intervalo de la integral,

$$\min_{\underline{\underline{\varepsilon}}} \int_{u_0}^{u_f} \| S_{\varepsilon}(\boldsymbol{\alpha}(u)) - \boldsymbol{\alpha}(u) \|_2^2 \, du \tag{4.8}$$

Esta función objetivo minimizará los cambios de la forma de la trayectoria Bézier inicial: minimiza la distancia entre la trayectoria Bézier original y la trayectoria Bézier modificada. Esta definición es adecuada para robots móviles holónomos ya que la trayectoria original ha sido generada mediante un planificador de trayectorias y partimos de que la trayectoria original ya es óptima.

2. **Número de curvas de Bézier.**

   Hay que recordar que en el capítulo 2 se estudió la necesidad de concatenar un número de curvas de Bézier porque dichas curvas presentan un inconveniente al aumentar el orden. Puesto que es necesario obtener la trayectoria Bézier de forma completa debemos concatenar un conjunto k de curvas de Bézier. Consecuentemente se cambia la función a optimizar definida previamente en 4.8 tal y como sigue,

$$\min_{\underline{\underline{\varepsilon}}^{(1)} \cdots \underline{\underline{\varepsilon}}^{(k)}} \sum_{l=1}^{k} \int_{u_f^{(l)}}^{u_0^{(l)}} \| S_{\varepsilon}(\boldsymbol{\alpha}_l(u)) - \boldsymbol{\alpha}_l(u) \|_2^2 \, du \tag{4.9}$$

   Siendo:

   - $\boldsymbol{\alpha}_l$ cada curva de Bézier (l).
   - $S_{\varepsilon}(\boldsymbol{\alpha}_l)$ la curva (l) deformada de Bézier.

- $\left[u_0^{(l)}, u_f^{(l)}\right]$ instante inicial y final de la trayectoria Bézier correspondiente a la curva (l).
- El vector $\underline{\underline{\varepsilon}}^{(l)}$ representa el conjunto de perturbaciones de los puntos de control de la curva modificada (l).

Para $1 \leq l \leq k$.

Observación 5. *El número de curvas de Bézier que se van a utilizar para calcular la trayectoria Bézier tiene que ser igual al número de fuerzas repulsivas que se generan con las técnicas **PF**. Se coloca un vector por curva de Bézier puesto que su orden es cuadrático.*

4.5.2. Fusión del algoritmo BTD y las técnicas PF, en particular PFP: BTD+PFP.

1. **Cálculo de los puntos de control a partir del horizonte de predicción que se genera con el PFP**.

   Los puntos de control están distribuidos de forma uniforme a lo largo de todo el horizonte de predicción que se genera al aplicar el método **PFP**. El modelo desarrollado es para robots holonómicos. Por ello, la predicción de las posiciones futuras está situada en una línea recta. Los puntos de control se van a calcular a través de la siguiente tabla 4.1.

2. **Localización de la fuerza de repulsión sobre la curva de Bézier.**

   Como los puntos de control están distribuidos de forma uniforme, había que decidir un criterio para colocar las fuerzas de repulsión, obtenidas mediante el **PFP**, en la curva de Bézier. El criterio elegido es colocar cada vector en la mitad de la curva de Bézier excepto en la primera y última curva donde el vector se colocará en el primer y último punto respectivamente. En la figura 4.13 tenemos un ejemplo donde podemos observar una línea recta que representaría la predicción de la trayectoria óptima del robot móvil obtenida con el algoritmo **PFP**. Además, los puntos de control necesarios para obtener las curvas de Bézier se visualizan con círculos rojos. Las fuerzas repulsivas se colocan en las posiciones adecuadas de la predicción de la trayectoria. En este ejemplo gráfico hay ocho puntos del horizonte de predicción, con lo que hay ocho curvas de Bézier.

3. **Transformación de la unión de Béziers en una trayectoria del robot móvil.**

   Para transformar una curva de Bézier en una trayectoria hay que tener en cuenta el instante de tiempo en el que el robot llega a su posición. Para ello se cambia el parámetro intrínseco y se introduce el parámetro temporal, esta conversión está desarrollada en la fórmula 4.5. Si seguimos con el ejemplo de la figura 4.13 generamos la siguiente tabla 4.2 con los instantes de tiempo inicial y final en segundos.

Puntos de Control de la primera curva	Puntos de Control de la segunda Curva
$\mathbf{P}_0^{(1)} = \widehat{\mathbf{x}}(1)$	$\mathbf{P}_0^{(2)} = \mathbf{P}_2^{(1)}$
$\mathbf{P}_1^{(1)} = \widehat{\mathbf{x}}(1) + \frac{1}{3} \cdot (\widehat{\mathbf{x}}(2) - \widehat{\mathbf{x}}(1))$	$\mathbf{P}_2^{(2)} = \widehat{\mathbf{x}}(2) + \frac{1}{2} \cdot (\widehat{\mathbf{x}}(3) - \widehat{\mathbf{x}}(2))$
$\mathbf{P}_2^{(1)} = \widehat{\mathbf{x}}(1) + \frac{2}{3} \cdot (\widehat{\mathbf{x}}(2) - \widehat{\mathbf{x}}(1))$	$\mathbf{P}_1^{(2)} = \mathbf{P}_0^{(2)} + \frac{1}{2} \cdot \left(\mathbf{P}_2^{(2)} - \mathbf{P}_0^{(2)}\right)$
Puntos de Control para la j-curva, $3 \leq j \leq k-2$	
$\mathbf{P}_0^{(j)} = \widehat{\mathbf{x}}(j-1) + \frac{1}{2} \cdot (\widehat{\mathbf{x}}(j) - \widehat{\mathbf{x}}(j-1))$	
$\mathbf{P}_1^{(j)} = \mathbf{P}_0^{j}$	
$\mathbf{P}_2^{(1)} = \widehat{\mathbf{x}}(j)$	
Puntos de Control de la penúltima curva	Puntos de Control de la última curva
$\mathbf{P}_0^{(k-1)} = \widehat{\mathbf{x}}(k-2) + \frac{1}{2} \cdot (\widehat{\mathbf{x}}(k-1) - \widehat{\mathbf{x}}(k-2))$	$\mathbf{P}_0^{(k)} = \mathbf{P}_2^{(k-1)}$
$\mathbf{P}_2^{(k-1)} = \widehat{\mathbf{x}}(k-1) + \frac{1}{3} \cdot (\widehat{\mathbf{x}}(k) - \widehat{\mathbf{x}}(k-1))$	$\mathbf{P}_1^{(k)} = \widehat{\mathbf{x}}(k-1) + \frac{2}{3} \cdot (\widehat{\mathbf{x}}(k) - \widehat{\mathbf{x}}(k-1))$
$\mathbf{P}_1^{(k-1)} = \mathbf{P}_0^{(k-1)} + \frac{1}{2} \cdot \left(\mathbf{P}_2^{(k-1)} - \mathbf{P}_0^{(k-1)}\right)$	$\mathbf{P}_2^{(k)} = \widehat{\mathbf{x}}(k)$

Cuadro 4.1: Cálculo de los puntos de control a partir del horizonte de predicción, $\widehat{\mathbf{x}}$ es el vector del horizonte de predicción que genera la predicción de la trayectoria futura y $\mathbf{P}_i^{(j)}$ es el punto de control i-ésimo de la curva j-ésima.

Curva	Tiempo inicial (seg.)	Tiempo final (seg.)
Primera curva $k = 1$	$u_0 = 0$	$u_f = 1{,}33$
Segunda curva $k = 2$	$u_0 = 1{,}33$	$u_f = 3$
Tercera curva $k = 3$	$u_0 = 3$	$u_f = 5$
Cuarta curva $k = 4$	$u_0 = 5$	$u_f = 7$
Quinta curva $k = 5$	$u_0 = 7$	$u_f = 9$
Sexta curva $k = 6$	$u_0 = 9$	$u_f = 11$
Séptima curva $k = 7$	$u_0 = 11$	$u_f = 12{,}66$
Octava curva $k = 8$	$u_0 = 12{,}66$	$u_f = 14$

Cuadro 4.2: Los tiempos iniciales y finales de cada curva.

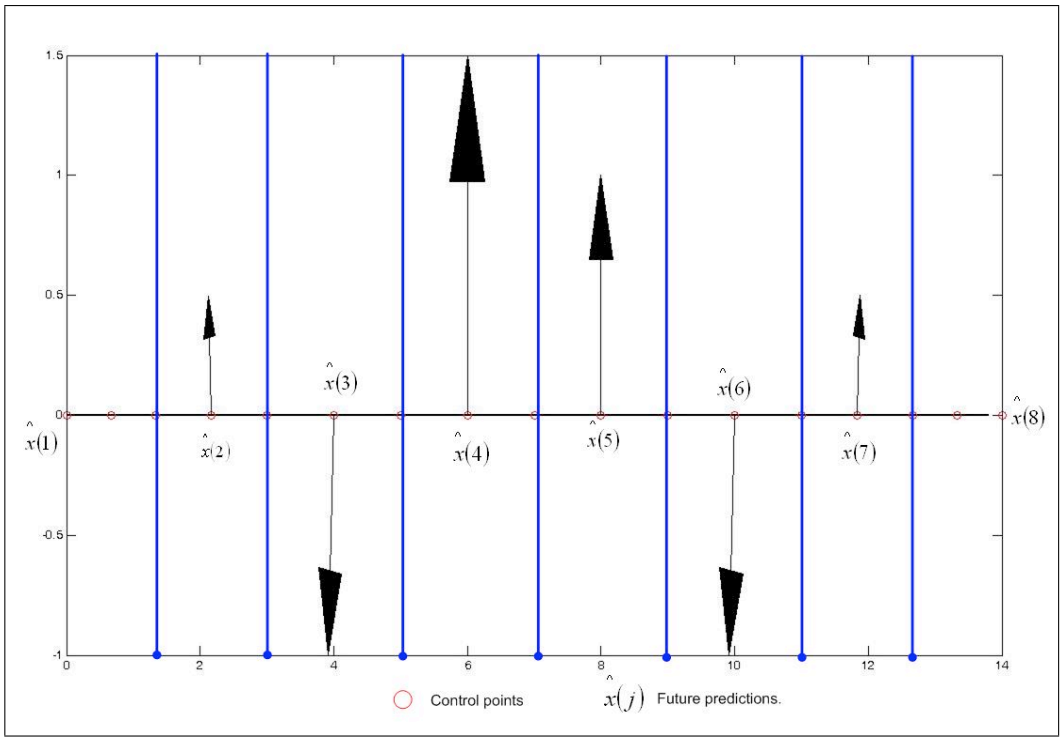


Figura 4.13: Puntos de control y predicciones futuras de la trayectoria Bézier.

4. **Remuestreo de la trayectoria deformada.**

   Utilizando las definiciones vistas anteriormente, la unión de las curvas de Bézier se transforma en una trayectoria continua. Sin embargo, el robot y en particular el bucle de control necesita posiciones discretas en cada instante de tiempo. Para poder generar el proceso del remuestreo se genera una polilínea. En el eje X de la polilínea se representa el tiempo acumulado y en el eje Y se representa el parámetro intrínseco acumulado. Utilizando la polilínea, el proceso de remuestreo es sencillo. Tomando valores en el eje X homogéneamente distribuidos (con un período de remuestreo T) se calcula su imagen a través de la polilínea y se obtiene un número real. La parte entera de este número real corresponde al número de la curva. La parte decimal de este número real pertenece al parámetro intrínseco de esta curva. En la figura 4.14 está representada la polilínea correspondiente al remuestreo para la tabla 4.2 y el ejemplo visto en la figura 4.13.

   La representación del remuestreo para la concatenación de ocho curvas de Bézier se representa en la figura 4.15. Siendo:

   - El asterisco azul que hay sobre la línea recta, representa el remuestreo del parámetro intrínseco en el horizonte de predicción de la trayectoria Bézier inicial.
   - La curva roja representa la trayectoria Bézier modificada mediante las fuerzas repulsivas asociadas a cada predicción de la posición inicial.

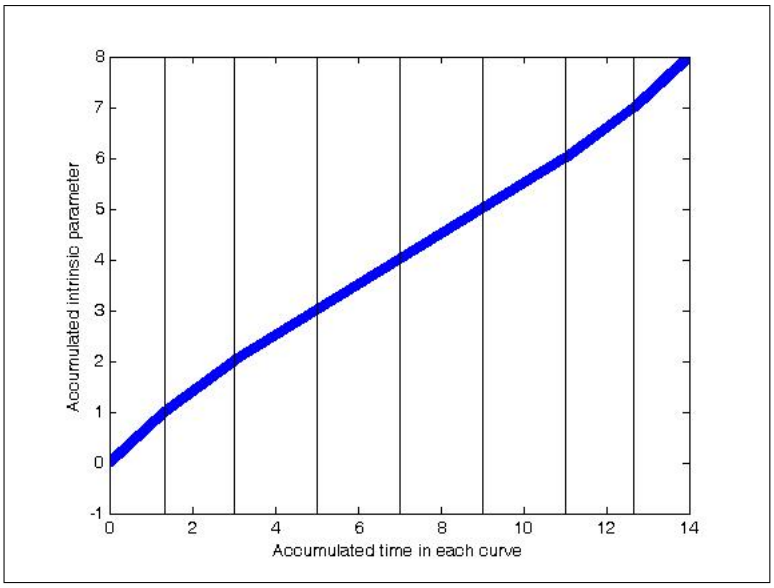


Figura 4.14: La polilínea utilizada para el remuestreo utilizando ocho curvas de Bézier.

- El cuadrado azul que hay sobre la curva de Bézier deformada es el remuestreo del parámetro intrínseco en la trayectoria Bézier deformada.

Una vez hemos visto la transformación del algoritmo **BSD** en **BTD** y su fusión con la técnica **PFP**, generando el **BTD+PFP**, pasemos a estudiar ahora el significado de todas las restricciones vistas en el problema de optimización restringida que se desarrolló en el capítulo 2.

El significado de cada restricción para una trayectoria Bézier en el algoritmo **BTD**:

1. **El robot móvil debe ser guiado por una trayectoria libre de colisiones.**
   Esta restricción se incluye ya que en la primera condición que veíamos en el capítulo 2 en la ecuación 2.10, se imponía que la curva modificada de Bézier pasase a través de los Puntos Finales. En este caso, la trayectoria Bézier modificada debe pasar por los citados Puntos. En consecuencia, el robot no colisionará con los obstáculos detectados en el entorno. Los vectores que unen los Puntos Iniciales y Finales son los campos de fuerzas repulsivas obtenidas mediante el **PFP**. En la ecuación 4.10 tenemos como se expresa esa restricción cuando tenemos definida la trayectoria Bézier.

$$\sum_{l=1}^{k}\sum_{j=1}^{r_l}\langle \boldsymbol{\lambda}, T_j^{(l)} - S_{\varepsilon}(\alpha_l(u_j^{(l)}))\rangle \tag{4.10}$$

2. **La trayectoria del robot debe ser una trayectoria suave.**

   Esta condición también se cumple ya que en el algoritmo **BTD** hay una restricción que impone continuidad y derivabilidad en los puntos donde se concatenan las curvas. Dicha restricción se estudió en el capítulo 2 en las ecuaciones 2.19 y 2.22. Para

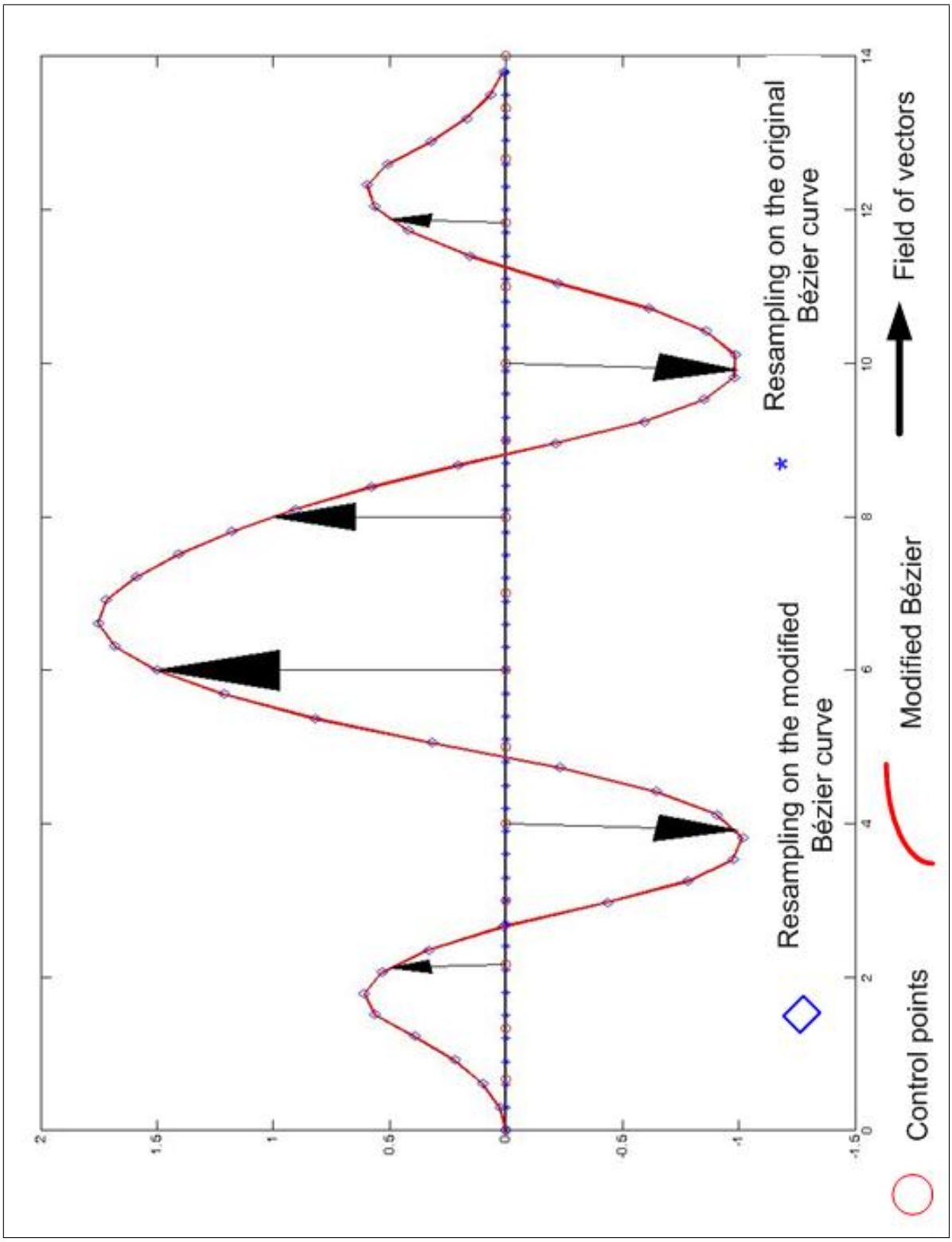


Figura 4.15: La deformación de ocho curvas de Bézier concatenadas.

la trayectoria Bézier estas restricciones se expresan con las siguientes ecuaciones 4.11 y 4.12.

$$\sum_{l=1}^{k-1} \langle \boldsymbol{\lambda}, S_{\varepsilon}(\boldsymbol{\alpha}_l(u_f^{(l)})) - S_{\varepsilon}(\boldsymbol{\alpha}_{l+1}(u_0^{(l+1)}))\rangle \tag{4.11}$$

$$\sum_{l=1}^{k-1} \langle \boldsymbol{\lambda}, S_{\varepsilon}(\boldsymbol{\alpha}'_l(u_f^{(l)})) - S_{\varepsilon}(\boldsymbol{\alpha}'_{l+1}(u_0^{(l+1)}))\rangle \tag{4.12}$$

3. **Como consecuencia de aplicar el algoritmo PFP, se debe asegurar la continuidad entre las posiciones presentes y la predicción de las posiciones futuras.**

   Esta restricción se incluye en la definición del **BTD** ya que se mantiene la tangencia entre la trayectoria Bézier original y la trayectoria Bézier deformada en el instante inicial y final de la trayectoria completa. Esta restricción en el capítulo 2 se expresa con las siguientes ecuaciones 2.11 y 2.12. En el caso de la trayectoria Bézier se definen en 4.13.

$$\langle \boldsymbol{\lambda}, \boldsymbol{\alpha}'_1(u_0^{(1)}) - S_{\varepsilon}(\boldsymbol{\alpha}'_1(u_0^{(1)}))\rangle + \langle \boldsymbol{\lambda}, \boldsymbol{\alpha}'_k(u_f^{(k)}) - S_{\varepsilon}(\boldsymbol{\alpha}'_k(u_f^{(k)}))\rangle \tag{4.13}$$

4.5.3. Introducción de los tensores en el BTD: T-BTD.

El algoritmo **BTD+PFP** tiene un coste computacional bajo, pero es cierto que crece de forma exponencial a medida que se concatenan más curvas de Bézier para poder generar la trayectoria. El hecho de concatenar más o menos curvas es directamente proporcional a la precisión requerida para la trayectoria Bézier. Cuanto más precisa sea la trayectoria, más curvas de Bézier vamos a requerir.

Puesto que el coste computacional en robótica es un punto crítico, una forma de reducirlo es adaptar el **T-BSD**, definido en el capítulo 2, al **T-BTD** (*Tensor-Bézier Trajectory Deformation*), de esa forma el algoritmo se reformula con una notación basada en tensores y con ello se reduce de forma considerable el tiempo de cómputo. Hay que tener en cuenta la definición de la trayectoria Bézier, que conlleva un cambio en el parámetro intrínseco. Este cambio debe reflejarse en la ecuación 2.53 del algoritmo **T-BSD** para poder adaptarse al algoritmo **T-BTD**.

En el capítulo 2 hemos visto la figura 2.25. En esa figura se observa como al aumentar el número de curvas de Bézier que se concatenan el tiempo de cómputo se incrementa de forma exponencial, es decir, que tenemos un orden de operaciones $O(n^f)$. Al formular el algoritmo con tensores el comportamiento se reduce a lineal, $O(fn)$.

La solución que se obtiene con el **BTD** y el **T-BTD** es la misma. En el caso del **T-BTD** la matriz del sistema lineal obtenido para calcular la perturbación de los puntos de control que harán que se deforme la trayectoria original, se construye de forma más eficiente al basar su notación en tensores. La diferencia sustancial del tiempo de cómputo entre el **BTD** y el **T-BTD** está en la construcción de la matriz y en consecuencia en su forma de operar.

4.6. Resultados de Simulación.

El algoritmo **BTD** se valida mediante una simulación implementada con el Matlab para poder visualizar el comportamiento del robot cuando se fusiona el algoritmo **BTD** con el **PFP**, es decir, se emplea el **BTD+PFP**. En la parte izquierda de la figura 4.16 se representa un escenario 2D con cinco obstáculos móviles (en amarillo) y un robot móvil (en azul). Los obstáculos siguen una trayectoria lineal (dada por su modelo cinemático) yendo de lado a lado del entorno, simulando un rebote en una mesa de billar. Los círculos representados en el robot y los obstáculos, son las incertidumbres de la predicción de la trayectoria. Cuando los obstáculos se acercan al robot, empieza a evitarlos con una maniobra suave basada en el **BTD**, que modifica su trayectoria inicial (dada por el **PFP**) y que guía al robot a su posición final sin colisionar con los obstáculos. Si no se detectan obstáculos en el entorno, entonces el **BTD** transforma la predicción de la trayectoria en una trayectoria Bézier que será la más apropiada para la navegación del robot dadas las propiedades de las curvas de Bézier.

Además, en esta figura 4.16 podemos ver como se representa la trayectoria que debe seguir el robot para llegar a las posiciones finales (*goal*) 1 y 2 evitando los obstáculos en el entorno en diferentes instantes de tiempo de simulación. En la secuencia de figuras de la derecha que se muestran en la figura 4.16 podemos observar los detalles de la trayectoria del robot en determinados instantes específicos de tiempo en la imagen correspondiente de la figura de la izquierda. En la figura 4.17 tenemos otro ejemplo, pero en este caso encontramos quince obstáculos dinámicos en el entorno.

4.7. Conclusiones.

En este capítulo se ha diseñado una trayectoria con una curva paramétrica, lo que hemos llamado trayectoria Bézier. Se ha generado a partir de un planificador de caminos mediante una de las técnicas de los Campos Potenciales (**PF**). Los **PF** planifican el camino del robot libre de obstáculos. Al fusionar el algoritmo **BTD** con **PF**, obteniendo **BTD+PF**, se produce una mejora de los Campos Potenciales puesto que el camino que se describe mediante una cantidad de puntos discretos que son las predicciones futuras del robot, con el **BTD** se representará mediante una Bézier suave y continua. Además de representar la trayectoria inicial del robot, el algoritmo **BTD+PFP** permite la deformación de la trayectoria Bézier inicial.

Hemos visto a lo largo del presente capítulo que existen muchas publicaciones para generar trayectorias de forma continua utilizando curvas paramétricas. También existen muchas publicaciones que planifican el camino del robot mediante Campos Potenciales y es uno de los métodos más utilizados para la evitación de los obstáculos de los robots móviles. Para conocimiento del autor, es la primera vez que se fusionan ambas metodologías porque hasta ahora cuando se representaba una trayectoria mediante una curva paramétrica no se había desarrollado el algoritmo que la deformase mediante vectores.

Como consecuencia de ello se obtiene la trayectoria del robot móvil de forma con-

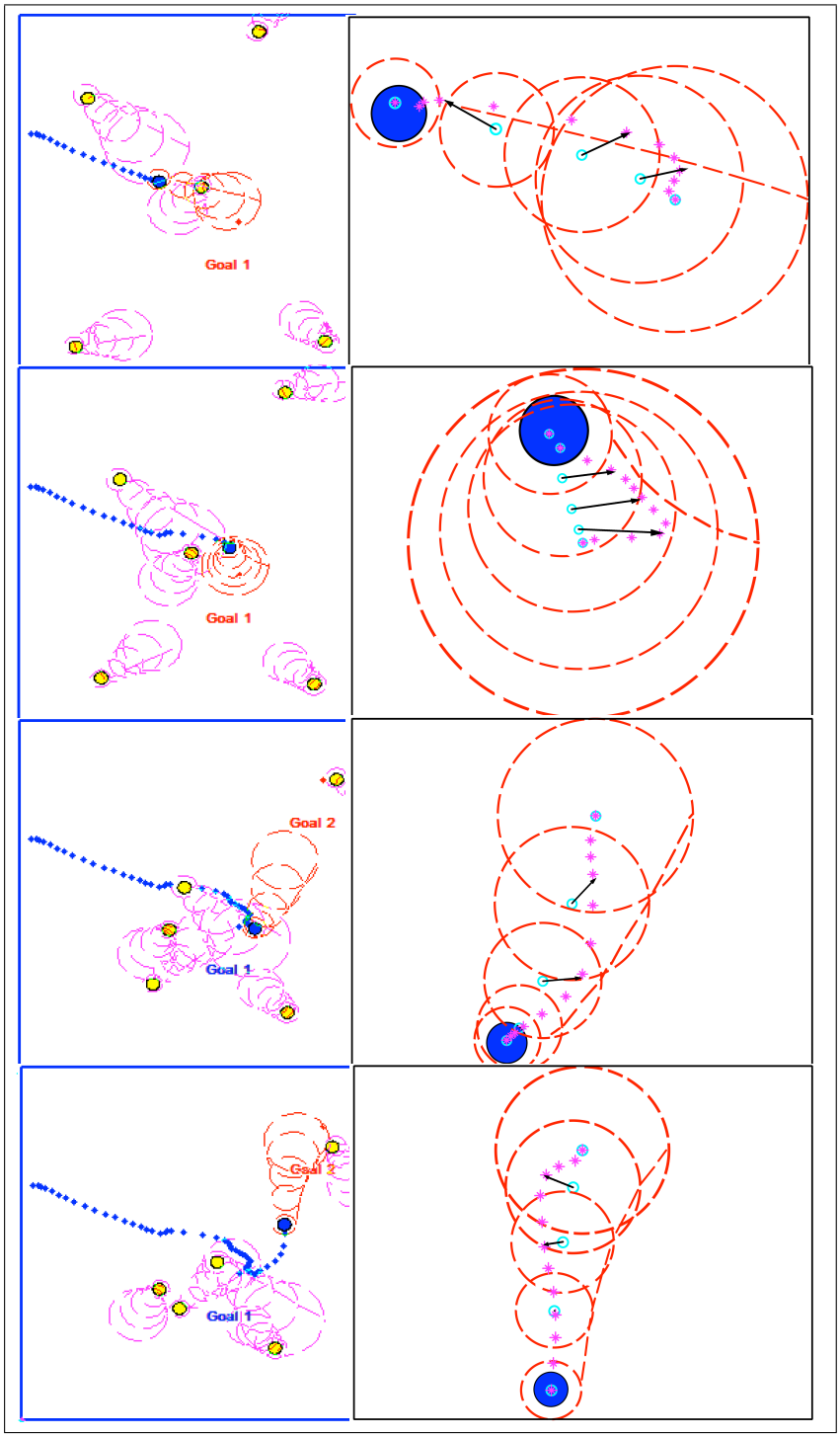


Figura 4.16: Instantáneas de la trayectoria del robot (imágenes de la izquierda) obtenidas al aplicar el algoritmo **BTD+PFP** en un entorno con cinco obstáculos móviles. En la parte de la derecha se muestran un zoom de la trayectoria del robot de su respectiva imagen de la izquierda.

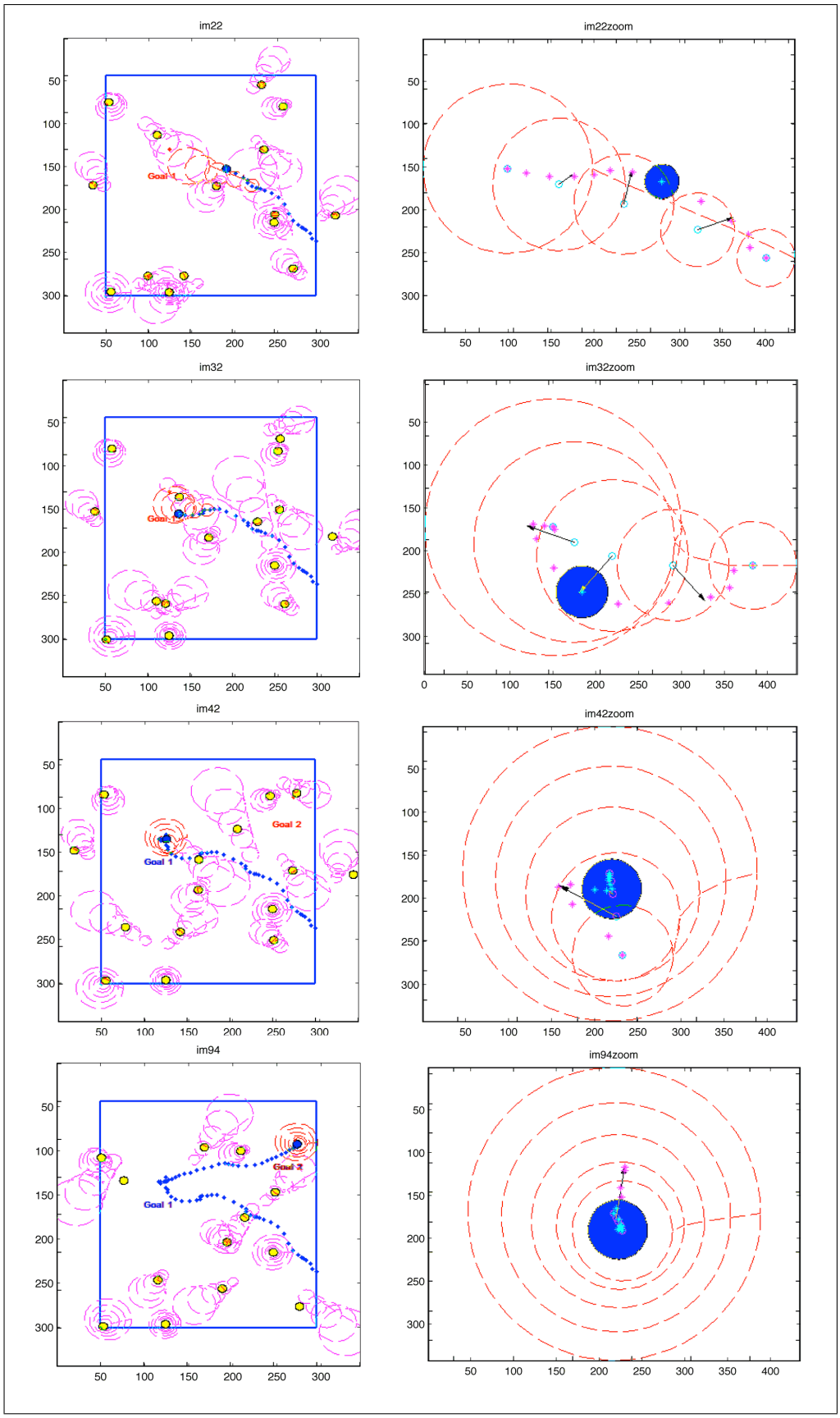


Figura 4.17: Imagen análoga a la figura 4.16 pero con quince obstáculos.

tinua con una curva Bézier, que por sus propiedades permite una sencilla manipulación. Por lo tanto, la trayectoria es modificada en tiempo real para que el robot móvil pueda navegar en un entorno dinámico donde se pueden encontrar muchos obstáculos móviles y no colisione con ellos.

Además de generar una trayectoria Bézier y modificarla a través de vectores con el **BTD**, se ha introducido el uso de tensores en el algoritmo. En el capítulo 2 ya vimos la importancia de los tensores para reducir el tiempo de cómputo en los algoritmos y además en los últimos años se ha utilizado en diferentes ámbitos. Pero todavía no se ha utilizado el álgebra tensorial para algoritmos relacionados con la robótica. Al introducir los tensores en el algoritmo **BTD** se ha generado el algoritmo **T-BTD**. El coste computacional del **BTD** era exponencial a medida que aumentaba el número de curvas de Bézier que se concatenaban para generar la trayectoria Bézier. Con el **T-BTD** el comportamiento del tiempo de cómputo es lineal. Así pues se ha mejorado considerablemente el algoritmo al tener la posibilidad de concatenar muchas curvas de Bézier, ya que eso implica obtener una trayectoria mucho más precisa.

Capítulo 5

Conclusiones y Trabajos Futuros.

La posibilidad de realizar un sueño es lo que
hace que la vida sea interesante.

Paulo Coelho (1947-?)

Este es el último capítulo de esta Memoria y ahora es el momento de agrupar las principales conclusiones y de enumerar las principales aportaciones que se han ido comentando a lo largo de todo el documento. Este capítulo estará organizado de la siguiente forma, un primer punto donde se resuman las conclusiones y se indiquen las principales aportaciones, y un segundo punto donde se resuman las líneas futuras de investigación que quedan abiertas a partir de este primer trabajo.

5.1. Conclusiones y principales aportaciones de la Tesis Doctoral.

El objetivo principal de esta Tesis estaba en la búsqueda de soluciones a problemas abiertos en dos ámbitos de la ingeniería: la robótica móvil y los procesos LCM. Se han desarrollado soluciones para estos problemas mediante técnicas o algoritmos matemáticos. En un principio se ha desarrollado un algoritmo que deforma curvas de Bézier mediante vectores: el *Bézier Shape Deformation*, **BSD**. Además de desarrollar el **BSD** se ha fusionado con las técnicas de elementos finitos y seguimiento de partículas que se aplican en los procesos LCM y en la parte de la robótica se ha fusionado con los métodos reactivos que se aplican para la planificación de caminos libres de obstáculos.

En cuanto a los procesos LCM, el objetivo es poder representar el frente de avance de la resina con una curva continua a medida que se llena el molde. Además de ser capaz de poder calcular dicho frente, hay que poder actualizar su información en cada instante de tiempo, es decir, que el frente de avance se deforma o se modifica a medida que el molde se llena. Por la forma en la que está definido y desarrollado el **BSD** permite esta actualización. Tener esa información es vital en los llenados de moldes para la mejora del proceso.

Por otra parte en la robótica, la planificación de la trayectoria de un robot móvil es uno de los problemas más investigados en este campo. Hay diversas formas de abordar esta planificación porque hay que tener en cuenta que hay que definir una trayectoria lo suficientemente flexible como para permitirle al robot libertad de movimiento, entendiendo con ello que se puedan producir cambios en su trayectoria, como por ejemplo que el robot tenga que girar. Hay que tener varios factores en cuenta, entre ellos uno de los más importantes es que el entorno de un robot móvil en principio puede ser cambiante, eso implica que pueda encontrarse con obstáculos dinámicos que le impidan continuar con su trayectoria inicial. El algoritmo **BSD** contempla dos factores importantes: el diseño de una trayectoria flexible, suave, continua y fácil de manipular; y la evitación de los obstáculos, es decir, que la trayectoria inicial se cambiará cuando el robot se cruce o se vaya a encontrar con uno o varios obstáculos.

En la presente Memoria se ha podido comprobar como ambos ámbitos son muy activos en cuanto a investigación se refiere, así que con estos algoritmos se ha intentado mejorar los ya existentes introduciendo técnicas no utilizadas anteriormente. En el caso de los procesos LCM, son los primeros trabajos que representan el frente de avance mediante una curva paramétrica fácilmente manipulable. Por otra parte, en el caso de la robótica móvil por primera vez se combina el uso de los Campos Potenciales con el uso de las curvas paramétricas para la simulación de la trayectoria del robot móvil. Es cierto que existen muchas publicaciones referentes a los Campos Potenciales y al diseño de trayectorias con curvas paramétricas, pero en ningún caso (para conocimiento del autor) se había fusionado hasta el momento.

Los tensores vienen siendo utilizados en diferentes ámbitos de la investigación para la reducción del coste numérico de muchos algoritmos. En la industria poder tomar decisiones lo más rápidas posibles supone mejorar el rendimiento y eso conlleva beneficios económicos. Para mejorar el tiempo de cómputo del algoritmo desarrollado, se ha utilizado una formulación basada en tensores mejora de esta forma la respuesta on-line de la deformación de la curva paramétrica. Además con los tensores se ha logrado demostrar que la solución obtenida es un mínimo tal y como se pretendía.

En el capítulo 1 veíamos la figura 1.1. Esta figura nos ayuda a resumir las principales aportaciones de esta Memoria diferénciandolas en:

- Aportaciones transversales al ámbito CAGD y al del álgebra tensorial.
- Aportaciones verticales en la robótica móvil y el llenado de moldes LCM.

5.1.1. Aportaciones Transversales: CAGD y Tensores.

- Desarrollo de una técnica que deforma una familia de curvas concatenadas de Bézier mediante vectores, el algoritmo **BSD**. Este algoritmo está formulado y construido con una notación tradicional.
- Utilizar una formulación basada en tensores para el algoritmo **BSD**. De esta forma es la primera vez que se desarrolla un algoritmo que computa la deformación de una

curva de Bézier mediante vectores basando su notación en álgebra tensorial. Dicho algoritmo se denomina el **T-BSD**. Con él se consigue: reducción del tiempo de cómputo, clasificación del punto estacionario de la función Lagrangiana mediante la minimización convexa y una notación más compacta que permite aumentar la dimensión de las curvas de Bézier para un futuro.

5.1.2. Aportaciones verticales: Robótica y LCM.

- Fusionar el algoritmo **BSD** con las técnicas de elementos finitos y de seguimiento de partículas para poder calcular y actualizar el frente de avance en el llenado de moldes de resina mediante una curva paramétrica continua. El llamado algoritmo **BFD**.
- Desarrollar un algoritmo para mejorar la representación y actualización del frente de avance de la resina introduciendo la información del caudal de resina inyectada en el molde en cada instante de tiempo. El llamado algoritmo **BFD-A**.
- Fusionar el algoritmo **BSD** con los algoritmos de los Campos Potenciales para la evitación de obstáculos, de esa forma se genera la trayectoria de un robot móvil mediante una curva paramétrica y además es modificada para que no colisione con los obstáculos móviles del entorno dinámico. El denominado algoritmo **BTD**.
- La introducción de los tensores en el algoritmo **BTD**, definiendo así el **T-BTD**. Con ello se consigue una trayectoria más precisa.

Las publicaciones relacionadas con esta Tesis Doctoral son: [70–74, 117] y en revisión están las publicaciones [69, 75].

La investigación que se ha llevado a cabo ha sido financiada por los siguientes proyectos de investigación:

- Proyecto “Frabricación avanzada de composites moldeados con resinas líquidas mediante técnicas de visión artificial”, financiado por el Gobierno Español.
- Proyecto “Caracterización y aplicación de nanopartículas magnéticas en el control de llenado de moldes de composites con resinas líquidas”, financiado por la Universidad Cardenal Herrera CEU.
- Proyecto “Técnicas CAD/CAM avanzadas para la fabriación y producción automatizada (TRAYELÁSTICA)”, financiado por la Generalitat Valenciana.

5.2. Líneas futuras de investigación.

Terminando esta Tesis Doctoral es momento de poner un punto y seguido a la investigación ahora mismo iniciada. Ahora es cuando se empiezan abrir nuevas puertas para

continuar con los problemas ya iniciados y resueltos. Así pues se va a seguir en esta actividad investigadora ya iniciada con esta Memoria.

Las líneas futuras de investigación que surgen al finalizar esta Tesis las podemos diferenciar en cuanto a mejoras en sí del algoritmo **BSD** y **T-BSD**, o en cuanto a mejoras de sus respectivas aplicaciones: **BFD**, **BFD-A**, **BTD** y **T-BTD**. Por ello, lo podemos diferenciar entre:

- Líneas futuras transversales en CAGD y tensores: mejoras del **BSD** y **T-BSD**.
- Líneas futuras verticales en LCM: mejoras del **BFD** y **BFD-A**.
- Líneas futuras verticales en Robótica: mejoras del **BTD** y **T-BTD**.

5.2.1. Líneas futuras transversales.

Los trabajos futuros que surgen a partir del algoritmo **BSD** y **T-BSD** se pueden subdividir en los siguientes grupos:

- La introducción de nuevas restricciones en el **BSD**.

  Introducir nuevas restricciones o cambiar la función que se optimiza en el problema va a ir ligado al tipo de aplicación que se le dé para conseguir una mejora sustancial de la solución obtenida en la aplicación. En el capítulo 3 se ha visto un caso concreto de como introducir una restricción más al problema, viendo sus ventajas e inconvenientes. En cuanto a la posibilidad de cambiar la función a optimizar se puede ver desde varios enfoques, o bien porque el problema pretende minimizar otro factor, como por ejemplo la curvatura, o bien porque se quiere calcular la distancia entre las dos curvas paramétricas Bézier de forma diferente. En la publicación [2] podemos ver otra forma de calcular la distancia entre dos curvas de Bézier, en este caso utilizando la métrica Hausdorff.

- La deformación de otro tipo de curvas paramétricas.

  Con la curva de Bézier se ha obtenido el objetivo deseado, que es la deformación de una curva paramétrica a través de vectores y con Bézier se ha conseguido obtener la solución adecuada para las correspondientes aplicaciones. Sin embargo, queda abierto la posibilidad de desarrollar el algoritmo de forma análoga al **BSD** pero con otro tipo de curva paramétrica. Hay que estudiar si la forma de definir las otras curvas paramétricas permiten deformar la curva de otra forma, por ejemplo, deformando de forma local en vez de forma global. O también la posibilidad de introducir una curva racional que permitiría deformar la curva no sólo modificando los puntos de control si no modificando también los pesos. Por ejemplo, en la figura 2.13 se observa la diferencia entre modificar los pesos y modificar los puntos de control.

Para ello habrá que tener en cuenta dos factores principalmente: la expresión de la curva (debe ser sencilla para poder desarrollar todos los cálculos) y el tiempo de cómputo (debe ser el mínimo posible).

Por ejemplo, en la publicación [135] se computa la deformación de un caso particular de una B-Spline de forma análoga. Se trabaja con una uniforme B-Spline porque en su definición no se utilizan bases recurrentes, que lo más probable es que genere inconvenientes en los cálculos necesarios para obtener la perturbación de los puntos de control.

- Trabajar la inyectividad de las curvas de Bézier.

  El problema que surge en las curvas de Bézier e incluso en otras curvas paramétricas son las geometrías no deseadas: lazos, cúspides o puntos de inflexión. Ya se vio en el capítulo 2 que estas geometrías se deben a la posición de los puntos de control en el plano. Como consecuencia, el algoritmo **BSD** está desarrollado con curvas de Bézier cuadráticas. Subir el orden de las Béziers es otro objetivo para obtener más libertad en la deformación e incluso reducir el número de curvas a concatenar, pero para ello hay que estudiar la forma de conseguir que la curva de Bézier sea inyectiva.

5.2.2. Líneas futuras verticales: LCM.

En cuanto a las líneas futuras en el ámbito de los procesos LCM podemos dividirlos en los siguientes puntos:

- Generalización del algoritmo **BFD** y **BFD-A**.

  El objetivo de formular un algoritmo más generalizado radica en poder simular el frente de avance mediante curvas de Bézier para cualquier tipo de molde, por ejemplo, teniendo un obstáculo en su interior, ver la figura 5.1. En este tipo de casos hay que tener en cuenta que el frente de avance Bézier tiene que intersectar con los obstáculos, así que habría que contemplar la intersección para poder representar el frente cuando el molde consta de obstáculos. En definitiva, hay que diseñar el algoritmo para que sea más general y abarque todos los casos posibles.

- Mejora del coste computacional del algoritmo **BFD-A**.

  El algoritmo **BFD-A** resuelve un sistema de ecuaciones no lineal, esto provoca un aumento del coste de la CPU, por ello hay que buscar soluciones para reducirlo. O bien se mejora la forma de resolverlo, o bien se cambia la forma de calcular el área de la zona encerrada entre las dos curvas de Bézier. Tal y como está calculada el área provoca una no linealidad del sistema de ecuaciones. En consecuencia, no se puede aplicar una formulación basada en tensores y reducir de esta forma el tiempo de cómputo.

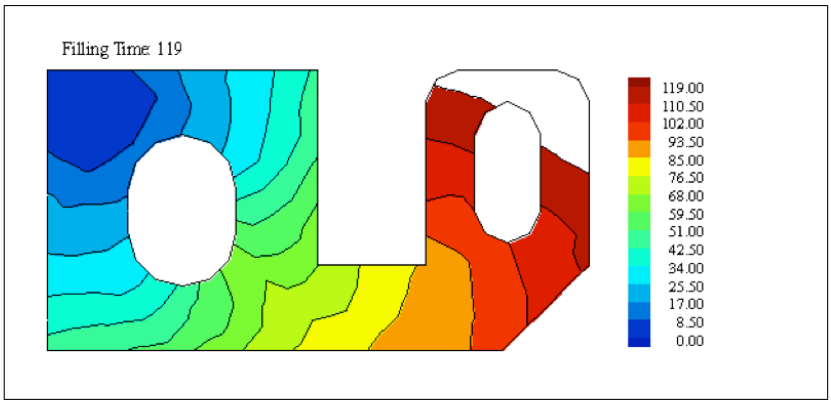


Figura 5.1: Simulación del llenado de un molde con obstáculos.

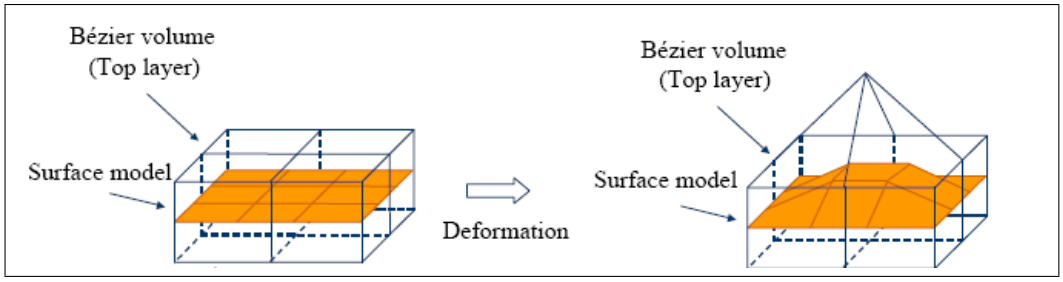


Figura 5.2: Simulación del llenado 2.5D.

- Desarrollar el **BFD** para superficies de Bézier.

  Si el **BFD** estuviese desarrollado para superficies, en el caso de los procesos LCM ganaríamos una ventaja porque se podría aplicar el algoritmo con modelos 2.5D. De esa forma se considera que el frente de avance de la resina no es una curva plana, si no que es una superficie porque tiene un grosor determinado, mirar la figura 3.4, en ese caso, el frente de avance no es una curva, es una superficie.

  Al deformar una superficie de Bézier conseguiríamos actualizar el frente de avance en los modelos 2.5D, ver figura 5.2.

  En el capítulo 2 ya se ha visto que existen trabajos que deforman también las superficies paramétricas. Generalizar el algoritmo **BSD** a superficies no resulta complicado porque tan sólo se trata de desarrollar el algoritmo de forma análoga pero añadiendo un parámetro intrínseco más. En la ecuación 5.1 tenemos la definición de una superficie de Bézier y en la imagen 5.3 podemos ver un ejemplo de una superficie de Bézier. Para conseguir el **BSD** para superficies habría que calcular las perturbaciones de los puntos de control que vemos en esta imagen.

$$r(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{P}_{i,j} B_{i,m}(u) B_{j,n}(v) \, (u,v) \in [0,1] \times [0,1] \tag{5.1}$$

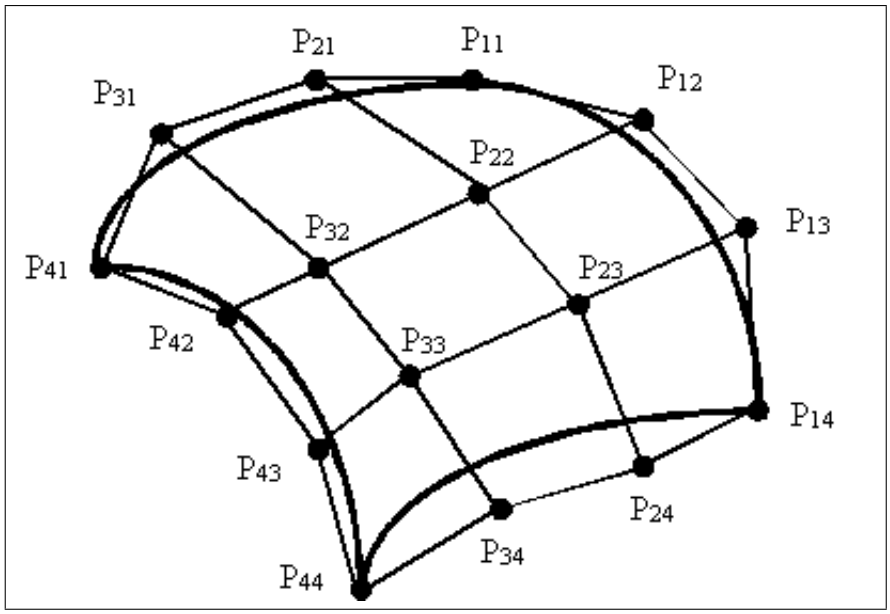


Figura 5.3: Ejemplo de una Superficie de Bézier.

- Mejora de los elementos finitos a través del **BFD**.

  Cuando se aplican las técnicas de elementos finitos es necesario mallar el dominio. Representar el frente de avance y actualizarlo a través del **BFD** mejora la información y precisión del frente. Es por ello que se permitiría una mallado menos fino y consecuentemente una mejora de las técnicas de elementos finitos.

- Introducción de los tensores en el **BFD**.

  Al igual que los tensores se han introducido para el ámbito de la robótica queda pendiente el uso de los tensores para los procesos LCM. De esta forma, se generará el algoritmo **T-BFD**.

5.2.3. Líneas futuras verticales: Robótica.

Si nos vamos al área de la robótica podemos sugerir los siguientes trabajos futuros a raíz de esta primera investigación:

- Mejora del algoritmo **BTD** y **T-BTD**.

  Cuando se habla de mejora se entiende como la posibilidad de poder introducir otros factores importantes para el robot que no se han tenido en cuenta en el caso del **BTD** y **T-BTD**. Como por ejemplo, el estudio de la curvatura de la curva que ello permite un giro más seguro del robot móvil.

  El problema de optimización tal y como está planteado nos permite la opción de poder introducir o eliminar restricciones al problema. Además también se podría

estudiar poder cambiar la función a optimizar en el problema porque de esa forma se cambiaría deformación de la trayectoria Bézier, siempre buscando la mayor precisión posible y la trayectoria óptima para el robot.

- Definir el algoritmo **BTD** y **T-BTD** para tres dimensiones.

  Otra de las líneas futuras de investigación relacionadas con este capítulo es poder llevar este algoritmo a trayectorias en tres dimensiones de forma que se diseñe una trayectoria en el espacio que evite obstáculos también. Este tipo de trayectorias en 3D se pueden aplicar en los UAV (Unmanned Aerial Vehicle), figura 5.4 , o un brazo robot, figura 5.5.

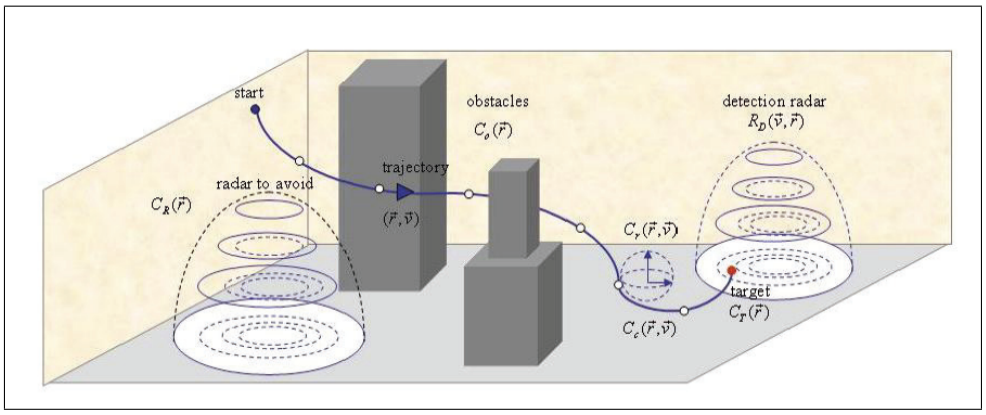


Figura 5.4: Ejemplo de la posible trayectoria de un UAV.

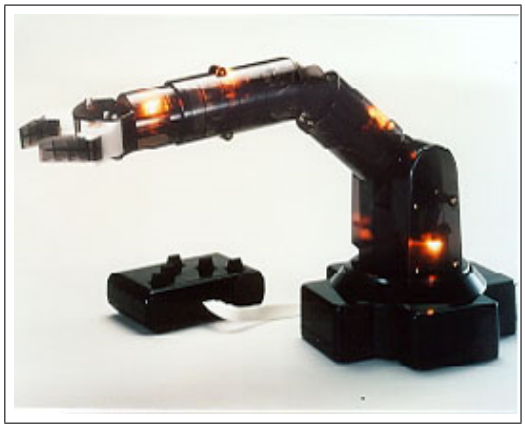


Figura 5.5: Ejemplo de la posible trayectoria de un brazo robot.

Una de las ventajas de la formulación del **BTD** basada en tensores es que permite el aumento de dimensión de forma sencilla. De echo la formulación del **T-BTD** está preparada para utilizar o bien dimensión dos o tres. El inconveniente está en la definición de la distancia entre dos curvas que no están contenidas en un plano.

Como trabajo futuro se plantea el definir de forma adecuada la función objetivo correspondiente.

- Introducción de los tensores en los algoritmos del ámbito de la robótica.

  El uso de los tensores que en las últimas decádas están siendo introducidos en muchos de los algoritmos con problemas numéricos porque se trabaja con espacios de grandes dimensiones, no está siendo utilizado todavía en la robótica móvil (para conocimiento del autor). Las publicaciones [73, 74] son las primeras al respecto que formulan un algoritmo basado en tensores. Además están en proceso de revisión estas otras publicaciones [69, 75].

  Esto abre las puertas a poder tensorizar muchos de los algoritmos ya existentes con el objetivo principal de poder reducir su tiempo de cómputo al igual que se ha realizado en el **BTD**. Nuestro primer objetivo es introducir los tensores en alguna de las técnicas de Campos Potenciales que se pueden fusionar con el **BTD**. Al combinar ambos algoritmos si están los dos con una formulación basada en tensores lograremos una reducción considerable del coste computacional. En consecuencia estas mejoras abrirían las puertas a la posibilidad de aplicar el algoritmo en escenarios realistas saturados a obstáculos.

Apéndice A

Curvas Paramétricas.

A.1. Definición de las curvas paramétricas más utilizadas en CAGD.

I **Curvas de Bézier racionales (RBC).**

Hay tres tipos de cónicas irreducibles, hipérbola, parábola y elipse, ver A.1.

Las parábolas se pueden parametrizar mediante funciones polinomiales, en cambio para poder parametrizar una hipérbola o una elipse se necesitan funciones racionales. Por ello, es necesario introducir las curvas de Bézier racionales. Su definición es la siguiente:

$$\alpha(u) = \frac{\sum_{i=0}^{n} \omega_i \mathbf{P}_i B_{i,n}(u)}{\sum_{i=0}^{n} \omega_i B_{i,n}(u)}, u \in [0,1] \tag{A.1}$$

Donde $\mathbf{P}_i$ son los puntos de control, $B_{i,n}(u)$ son las Bases de Bernstein y ω_i pesos

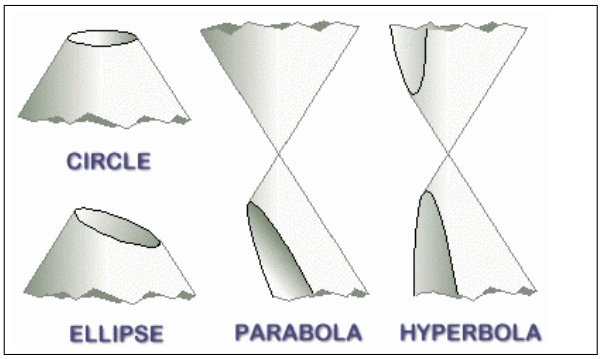


Figura A.1: Secciones cónicas.

asociados a cada punto de control. Al introducir el peso se introduce otra forma de poder obtener la modificación de la curva.

II **Curvas B-Splines.**

Las B-Splines son curvas polinómicas por trozos continuamente diferenciables hasta un orden prescrito. El nombre *Spline* es una palabra que en inglés significa "listón elástico". Estos listones eran usados por artesanos para crear curvas, que describen superficies a construir, como cascos de barcos y fuselajes de aviones. Constreñidos por pesos, estos listones elásticos o splines asumen una forma que minimiza su energía elástica.

Las curvas B-Splines se desarrollan para poder solucionar las limitaciones de las curvas de Bézier: necesidad de un control local de la curva, laboriosidad requerida para poder imponer una continuidad del tipo C^2 y de hecho de que el número de puntos de control de una curva de Bézier impone su grado.

De forma análoga a la definición de las curvas de Bézier, las curvas B-Splines se pueden expresar como una combinación afín de ciertos puntos de control $\mathbf{P}_i$. La definición de una curva B-Spline de grado k (u orden $k+1$) la podemos ver en A.2.

$$\alpha(u) = \sum_{i=0}^{n} \mathbf{P}_i \cdot N_{i,k}(u) \tag{A.2}$$

donde los $N_{i,k}$ son funciones polinómicas por trozos con soporte finito de orden k (grado $k-1$, se anulan fuera de un intervalo finito) y satisfacen ciertas condiciones de continuidad. Cada una de estas funciones se puede calcular empleando las fórmulas recursivas de Cox-de Boor. Para introducir una relación de recurrencia para definir los B-Splines, se considera por simplicidad una secuencia (a_i) doblemente infinita de nodos simples tales que $a_i < a_{i+1}$ para todo i. Entonces los B-Splines $N_{i,n}$ se definen a través de la siguiente relación de recurrencia.

$$N_{i,0}(u) = \begin{cases} 1 & \text{si } u \in [a_i, a_{i+1}) \\ 0 & \text{en caso contrario} \end{cases} \tag{A.3}$$

donde $N_{i,k}(u) = \dfrac{t-a_i}{a_{i+k-1}-a_i} N_{i,k-1}(u) + \dfrac{a_{i+k}-t}{a_{i+k}-a_{i+1}} N_{i+1,k-1}(u)$.
La figura A.2 muestra algunos B-Splines de grado 0, 1 y 2.

III **Curvas NURBS (Non uniform Rational B-Splines).** Al igual que se han obtenido las curvas racionales de Bézier a partir de las curvas de Bézier, análogamente se obtienen las curvas racionales B-Splines. Las curvas NURBS se definen en A.4,

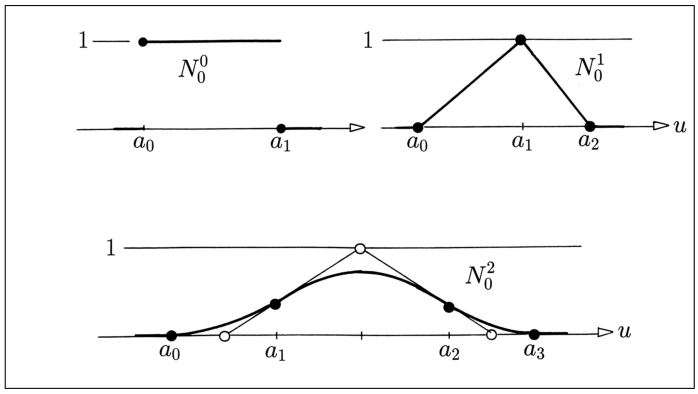


Figura A.2: Algunos B-Splines de grado 0,1 y 2

$$\alpha(u) = \frac{\sum_{i=0}^{n} \mathbf{P}_i \omega_i N_{i,k}(u)}{\sum_{i=0}^{n} \omega_i N_{i,k}(u)} \tag{A.4}$$

Información más detallada de las curvas y superficies paramétricas las podemos encontrar en los diferentes libros, como por ejemplo: [57, 110, 129, 137].

A.2. Estado del arte de las curvas paramétricas.

El año pasado 2011 se celebró el 30 aniversario del uso de las curvas y superficies NURBS para el modelado en 3D de su uso en la industria. En agosto de 1981 un estadounidense de aeronaves Boeing propuso la utilización de las NURBS. Hoy casi 30 años más tarde, es casi imposible encontrar un diseño asistido por ordenador, CAD (*Computer Aided Design*) que no sea compatible con las NURBS.

Los estudios de modelado geométrico en 3D comenzaron como parte de la manufactura asistida por ordenador, CAM (*Computer-Aided Manufacturing*), en vez de como parte de un CAD. Había diversos grupos de investigación que estudiaban los principios de los objetos de modelado en 3D. Los principales clientes de este tipo de estudios eran empresas aeroespaciales y empresas de la automoción.

El modelo de Citroën DS que se muestra en la figura A.3 (de 1955 a 1975) se convirtió en un icono de todos los tiempos. Su fabricación requiere de diferentes y complejas técnicas matemáticas. Uno de los primeros investigadores que inicia los estudios en este campo es el francés Paul de Casteljau, un ingeniero que trabajaba para Citroën. Los resultados de la investigación de Casteljau fueron publicados en 1974, pero en realidad el estudio estaba terminado en el año 1959.

Pierre Bézier al principio de los años 1960 estaba desarrollando el sistema UNISURF para diseñar superficies de los coches de Renault, él fue el que entendió lo importante que

Figura A.3: Citroën DS.

es controlar la forma de las curvas o de las superficies desde dentro. Pierre Bézier que era uno de los representantes de la tradición matemática francesa, era muy consciente de las obras de Charles Hermite (matemático francés del siglo XIX). En particular, las curvas cúbicas que llevan su nombre. La curva de Hermite es una representación geométrica de una curva cúbica utilizando puntos finales y vectores tangentes. La forma de una curva de Hermite puede ser controlada mediante la variación de las direcciones del vector y tamaños, ver la figura A.4.

Bézier no contento con las curvas de Hermite porque sólo podía modificar su conducta desde los puntos inicial o final, diseñó las curvas de Bézier. Estas curvas se pueden modificar, cambiando puntos intermedios, los llamados puntos de control, ver la definición 2.1. Una curva de Bézier independientemente de su orden siempre pasar por el primer y último punto de control. Además la curva es tangente al primer polígono de control que une el primer punto de control con el segundo, y esa misma tangencia ocurre en el último punto de control, ver la figura A.5. En dicha figura se muestran distintas curvas de Bézier donde se pueden observar estas propiedades. Además,las curvas de Bézier siempre quedan incluídas dentro de la envolvente convexa definida con los polígonos de control que unen los puntos de control que la definen.

Bézier publicó ese trabajo en 1962, pero cuando Citroën vio sus estudios doce años después se dieron cuenta que Paul Casteljau tenía conocimientos de estas curvas al menos tres años antes. Casteljau calculaba las curvas de Bézier de forma constructiva y dicho algoritmo fue nombrado como el algoritmo De Casteljau.

Forrest en 1972 encontró conexión entre las curvas de Bézier y los polinomios de Bernstein. Él fue el que demostró que las curvas de Bézier se representan como una com-

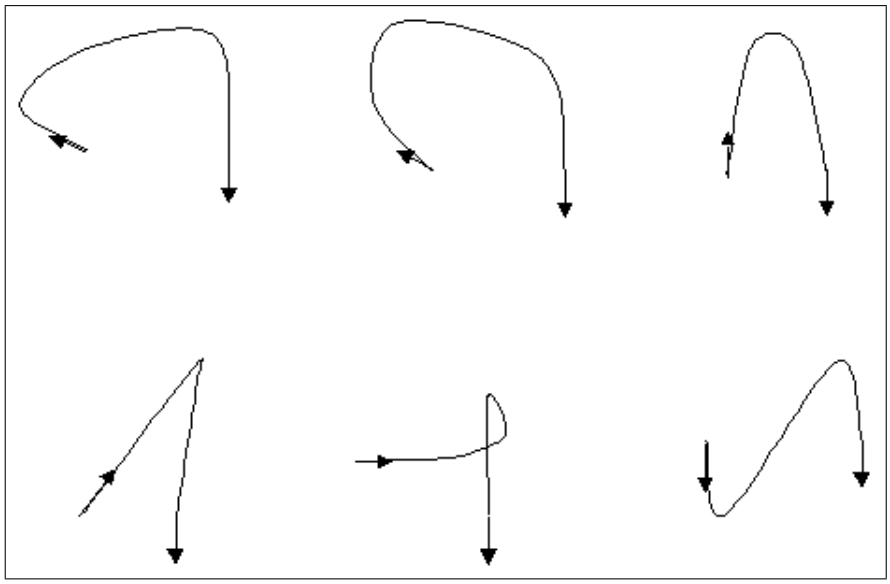


Figura A.4: Una familia de curvas Hermite

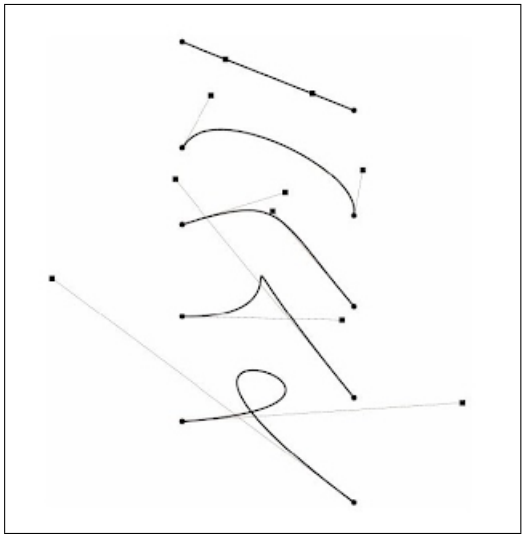


Figura A.5: Diferentes curvas de Bézier.

binación de polinomios de Bernstein. Esto permitió estudiar las propiedades de las curvas de Bézier con las propiedades de bases de Bernstein.

A pesar de lo ventajosas que resultan ser las curvas de Bézier, hay un par de propiedades que restringen su aplicación. La primera es que las curvas de Bézier no pueden representar de forma exacta las secciones cónicas, por ejemplo, un arco circular. La segunda es el grado algebraico de una curva de Bézier, ya que dicho grado se incrementa en función de los puntos de control. Subir el grado complica más los cálculos. Para combatir el problema del grado algebraico, matemáticamente se soluciona construyendo una curva a partir de segmentos, es decir, a partir de trozos, de forma que limitamos el grado. Este tipo de curvas se denominan Splines. La primera referencia de este tipo de curvas la hizo Isaac Schoenberg en 1946, un matemático americano de origen rumano. Más tarde, matemáticos como Carl de Boor y Cox en 1972 establecieron conexiones entre la forma geométrica de una curva compuesta por trozos y un método algebraico para su definición.

De esta forma quedaron definidas las curvas B-Splines que son una generalización de las curvas de Bézier porque se definen utilizando también unos puntos de referencia o puntos de control, pero la ventaja es que su grado algebraico no depende del número de puntos de control.

La definición de la curva B-Spline es similar a la curva Bézier, pero en vez de ser combinación lineal de las bases de Bernstein es combinación de unas funciones que se definen de forma recursiva y que reciben el nombre de funciones mezcla.

En 1975 Versprille, un estudiante de doctorado de la Universidad de Syracusa [163] propuso las curvas Non-Uniform Rational B-Splines o NURBS. Este tipo de curvas generalizaba las curvas B-Splines y las curvas Bézier. Las curvas NURBS ganaron rápidamente popularidad y fueron incorporadas dentro de muchos programas comerciales de modelado [132]. Las curvas NURBS tienen diferentes propiedades atractivas. Dichas propiedades ofrecen una formulación unificada no sólo para representar curvas o superficies con forma libre sino también para representar cónicas, cuádricas y superficies de revolución. Ajustando los puntos de control o manipulando los pesos asociados a cada punto de control, se pueden diseñar diferentes formas utilizando las curvas o superficies NURBS [55, 56, 130, 130, 131, 133, 154].

En los artículos [13, 153] podemos observar como se destaca la importancia de las curvas NURBS. Uno de los primeros factores que remarcan su importancia es que ofrecen una representación matemática general para objetos con una geometría analítica y para aquellos con una forma libre. Manipular las curvas o superficies NURBS modificando tanto los puntos de control como los pesos permite obtener una diferente variedad de formas geométricas. Los cálculos con las curvas o superficies NURBS se pueden hacer con bastante rapidez y son numéricamente estables. Las NURBS tienen una sencilla interpretación geométrica que es muy útil para los diseñadores con buenos conocimientos de la geometría. Además constan de muchas herramientas (la inserción de knots, aumento del grado, etc.) que se pueden utilizar para crear y analizar estos objetos, [132].

Al mismo tiempo, se pueden encontrar algunas desventajas, como por ejemplo, se requiere más memoria. Cuando se representa un círculo con una curva NURBS se necesitan siete puntos de referencia y diez knots, lo que significa el ahorro de treinta y ocho núme-

ros punto flotante en lugar de siete (centro, superficie normal y radio). Otra desventaja es la elección de la función peso, si dicha elección es incorrecta puede producir una parametrización muy pobre. Además hay algoritmos fundamentales, como por ejemplo el *inverse mapping*, que son numéricamente inestables con las NURBS.

Apéndice B

Teoremas y Definiciones.

Definición 5. *Un conjunto A es* ***convexo*** *si dados dos puntos $X,Y \in A$ el segmento que los une a ambos queda dentro del conjunto A, es decir, $\overline{XY} \subset A$.*

Definición 6. *La* ***envolvente convexa*** *de un conjunto de puntos es el conjunto convexo, ver definición 5, más pequeño que contiene a los puntos de control.*

Teorema 2 (Teorema de los Multiplicadores de Lagrange)**.** *Sea U un abierto de $\mathbb{R}^n$, $f : U \to \mathbb{R}$ y $g_i : U \to \mathbb{R}$ tal que $1 \leq i \leq m$ siendo $m < n$ funciones de clase C^1. Sea $S = \{X \in U : g_i(X) = 0, 1 \leq i \leq m\}$ y $X_0 \in S$. Supongamos que rango $(\nabla g_i(X_0))$ es máximo.*

Si $f|_S$ tiene un extremo relativo en X_0, entonces existen $\lambda_1, \cdots, \lambda_m$ números reales tales que

$$\nabla f(X_0) = \sum_{i=1}^{m} \lambda_i \nabla g_i(X_0)$$

Los números reales $\lambda_1, \cdots, \lambda_m$ se denominan Multiplicadores de Lagrange.

Definición 7. *Una curva paramétrica definida como $\alpha : [a,b] \longrightarrow \mathbb{R}^n$ se dice que es* ***inyectiva*** *si se cumple que: $\alpha(t_1) = \alpha(t_2) \Longrightarrow t_1 = t_2$ siendo $t_1, t_2 \in [a,b]$. En la figura B.1 podemos ver un ejemplo de una curva paramétrica que no es inyectiva.*

Definición 8. *Dados dos elementos $\boldsymbol{X} \in \mathbb{R}^n$ y $\boldsymbol{Y} \in \mathbb{R}^n$, el producto escalar se define como*

$$< \boldsymbol{X}, \boldsymbol{Y} >= \sum_{i=1}^{n} x_i \cdot y_i = \boldsymbol{Y}^T \boldsymbol{X} \tag{B.1}$$

Definición 9. *Dado un elemento $\boldsymbol{X} \in \mathbb{R}^n$, la norma$-2$ de dicho elemento de define como,*

$$\|\boldsymbol{X}\|_2^2 =< \boldsymbol{X}, \boldsymbol{X} >= \boldsymbol{X}^T \boldsymbol{X} \tag{B.2}$$

Definición 10. *Para cada valor s y t positivos se define la función Beta como sigue,*

$$\beta(s,t) = \int_0^1 x^s (1-x)^{t-1} \, dx \; ; s > 0, t > 0 \tag{B.3}$$

Propiedades de la función Beta.

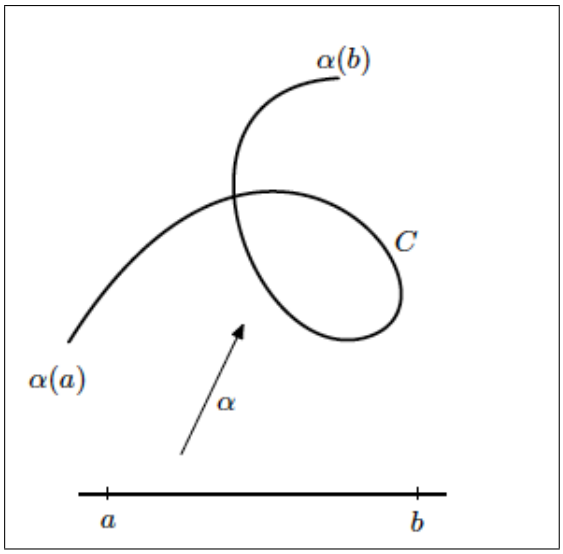


Figura B.1: Ejemplo de curva paramétrica no inyectiva.

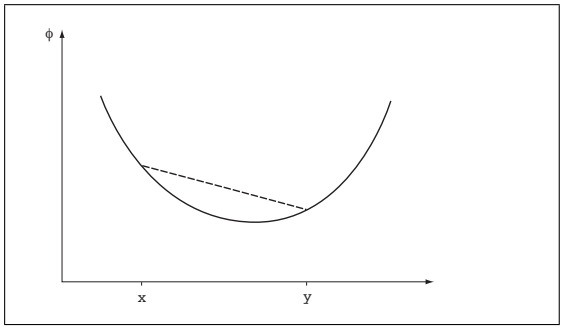


Figura B.2: Función convexa.

1. *Si* $s > 0$, $t > 0$, *entonces* $\beta(s,t) = \beta(t,s)$

2. *Si* $s > 0$, $t > 0$ *y* s *y* $t \in \mathbb{N}$, *entonces:* $\beta(s,t) = \dfrac{(s-1)!(t-1)!}{(s+t-1)!}$

Definición 11. *Dada una función* $\phi : S \to \mathbb{R}$ *definida en un conjunto* S *convexo en* $\mathbb{R}^n$. *Entonces:*

1. ϕ *es* ***convexa*** *en* S, *si*

$$\phi(\theta x + (1-\theta)y) \leq \theta\phi(x) + (1-\theta)\phi(y) \tag{B.4}$$

para cada par de puntos x, y *en* S *y para cada* $\theta \in (0,1)$, *ver figura B.2.*

2. ϕ *es* ***estrictamente convexa*** *en* S, *si*

$$\phi(\theta x + (1-\theta)y) < \theta\phi(x) + (1-\theta)\phi(y) \tag{B.5}$$

Teorema 3. *Dada una función $\phi : S \to \mathbb{R}$ dos veces derivable en un conjunto abierto y convexo S en $\mathbb{R}^n$. Entonces ϕ es convexa (ver definición 11)en S si y sólo si,*

$$d^2\phi(x;u) \geq 0, \forall x \in Syu \in \mathbb{R}^n. \tag{B.6}$$

Además, si la desigualdad en B.6 es estricta $\forall x \in S$ y $u \neq 0$ en $\mathbb{R}^n$, entonces ϕ es estrictamente convexa en S.

Observación 6. *El teorema 3 nos afirma que si ϕ es convexa (estrictamente convexa, cóncava, estrictamente cóncava) en S si la matriz Hessiana de ϕ es semidefinida positiva (definida positiva, semidefinida negativa, definida negativa) $\forall x \in S$.*

Teorema 4. *Dada una función $\phi : S \to \mathbb{R}$ diferenciable en un conjunto abierto y convexo S en $\mathbb{R}^n$, y dado c un punto de S donde*

$$d\phi(c;u) = 0 \tag{B.7}$$

*para cada $u \in \mathbb{R}^n$. Si ϕ es (estrictamente) convexa en S, entonces ϕ tiene un (estricto) **mínimo absoluto** en c.*

Teorema 5. ***Teorema de Green-Riemann*** *Dada una curva de Jordan derivable a trozos $\gamma : [a,b] \longrightarrow \mathbb{R}^n$. Dado D un conjunto convexo acotado por γ. γ está orientado positivamente. Dado $F = (M,N)$ un campo vectorial, $F : A \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^2$, $F \in C^1(A)$ tal que $D \subset A$. Entonces,*

$$\int_\gamma M\,dx + N\,dy = \int\int_D \left(\frac{\partial N}{\partial x} - \frac{\partial M}{\partial y}\right) dA \tag{B.8}$$

Corolario 6. *Una consecuencia del Teorema de Green-Riemann se utiliza para el cálculo del área de D a través de una integral curvilínea, siendo D y γ.*

$$\text{area}(D) = \int\int_D dA = \int_\gamma -y\,dx \tag{B.9}$$

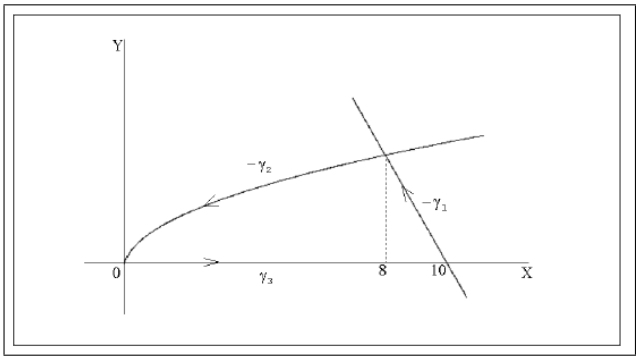


Figura B.3: Región D y su frontera γ.

Definición 12. *Dado $F : A \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$ un campo vectorial continuo y dado $\gamma : [a,b] \longrightarrow \mathbb{R}^n$ una curva derivable a trozos. La integral curvilínea a lo largo del camino γ se define como:*

$$\int_\gamma F = \int_a^b F(\gamma(u)) \cdot \gamma'(u)\, du \tag{B.10}$$

Bibliografía

[1] E. Acar, S. A. Çamtepe, M.S. Krishnamoorthy, and B. Yener. Modeling and multiway analysis of chatroom tensors. In *IEEE International Conference on Intelligence and Security Informatics*, volume 3495 of *ISI'05*, pages 256–268, Berlin, Heidelberg, 2005. Springer-Verlag.

[2] Young Joon Ahn. Hausdorff distance between the offset curve of quadratic bézier curve and its quadratic approximation. *Communications of the Korean Mathematical Society*, 22(4):641–648, 2007.

[3] J. Aleotti and S. Caselli. Trajectory clustering and stochastic approximation for robot programming by demonstration. In *Intelligent Robots and Systems,IROS.*, pages 1029–1034, 2005.

[4] J. Aleotti and S. Caselli. Trajectory reconstruction with nurbs curves for robot programming by demonstration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 73–78, 2005.

[5] J. Aleotti and S. Caselli. Grasp recognition in virtual reality for robot pregrasp planning by demonstration. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 2801–2806, 2006.

[6] M. Alex, O. Vasilescu, and Demetri Terzopoulos. Multilinear image analysis for facial recognition. In *International Conference on Pattern Recognition ICPR'02*, volume 3, pages 511–514, 2002.

[7] A. Ammar, B. Mokdad, F. Chinesta, and R. Keunings. A new family of solvers for some classes of multidimensional partial differential equations encountered in kinetic theory modelling of complex fluids. *Non-Newtonian Fluid Mechanics*, 139(3):153–176, 2006.

[8] C.M. Andersen and R. Bro. Practical aspects of parafac modeling of fluorescence excitation–emission data. *Journal of Chemometrics*, 17:200–215, 2003.

[9] C. A. Andersson and R. Bro. The n–way toolbox for matlab. *Chemometrics and Intelligent Laboratory Systems*, 52:1–4, 2000.

[10] C.A. Andersson and R. Henrion. A general algorithm for obtaining simple structure of core arrays in n-way pca with applications fo fluometric data. *Computational Statistics Data Analysis*, 31:255–278, 1999.

[11] J.R. Andrews and N. Hogan. Impedance control as a framework for implementing obstacle avoidance in a manipulator. *Control of Manufacturing Processes and Robotic Systems*, Eds.Hart, D.E. and Book, W., ASME Boston:243–251, 1983.

[12] C.J. Appellof and E.R. Davidson. Strategies for analyzing data from video fluorometric monitoring of liquid–chromatographic effluents. *Analytical Chemistry*, 53(13):2053–2056, 1981.

[13] C.K. Au and M.M.F. Yuen. Unified approach to nurbs curve shape modification. *Computer-Aided Design*, 27(2):85–93, 1995.

[14] B. W. Bader, R. A. Harshman, and T. G. Kolda. Temporal analysis of semantic graphs using asalsan. In 7^{th} *IEEE International Conference on Data Mining*, volume 0, pages 33–42, Lo Alamitos, CA, USA, 2007. IEEE Computer Society.

[15] F. Bassi and S. Rebay. High-order accurate discontinuous finite element solution of the 2d euler equations-order accurate discontinuous finite element solution of the 2d euler equations. *Journal of Computational Physiscs*, 138(2):251–285, December 1997.

[16] T. Berglund, H. Jonsson, and I. Sderkvist. An obstacle-avoiding minimum variation b-spline problem. In *International Conference on Geometric Modeling and Graphics*, 2003.

[17] G. Berkooz, P. Holmes, and J.L. Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 25:539–575, 1993.

[18] G. Beylkin and M.J. Mohlenkamp. Numerical operator calculus in higher dimensions. In *National Academy of Sciences*, volume 99, pages 10246–10251, 2002.

[19] G. Beylkin and M.J. Mohlenkamp. Algorithms for numerical analysis in high dimensions. *SIAM Journal on Scientific Computation*, 23:2133–2159, 2005.

[20] D. Bini. The role of tensor rank in the complexity analysis of bilinear forms. In *Presentation at ICIAM07*, 2007.

[21] R. Bro. Parafac. tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 38:149–171, 1997.

[22] R. Bro. *Multi–way Analysis in the Food Industry: Models, Algorithms, and Applications*. PhD thesis, University of Amsterdam, 1998.

[23] R. Bro. Review on multiway analysis in chemistry. *Critical Reviews in Analytical Chemistry*, 36:279–293, 2006.

[24] R. Bro, C.A. Andersson, and H.A.L. Kiers. Parafac2–part ii.modelling chromatographic data with retention time shifts. *Journal of Chemometrics*, 13:295–309, 1999.

[25] E. Cances, V. Ehrlacher, and T. Lelievre. Convergence of a greedy algorithm for high–dimensional convex nonlinear problems. *Mathematical Models and Methods for Applied Sciences*, February 2011.

[26] D. Cardoze, A. Cunha, G.L. Miller, T. Phillips, and Noel Walkington. A bézier-based approach to unstructured moving meshes. In *Symposium on Computational Geometry*, SCG '04, pages 310–319, New York, NY, USA, 2004. ACM.

[27] D. Cardoze, Gary L. Miller, M. Olah, and T. Phillips. A bézier-based moving mesh framework for simulation with elastic membranes. In 13^{th} *International Meshing Roundtable*, pages 71–80, Williamburg, VA, September 2004. Sandia.

[28] J.D. Carroll and J.J. Chang. Analysis of individual differences in multidimensional scaling via an *n*–way generalization of eckart-young decomposition. *Psychometrika*, 35(3):283–319, 1970.

[29] R.B. Cattell. Parallel proportional profiles and other principles for determining the choice of factors by rotation. *Psychometrika*, 9:267–283, 1944.

[30] R.B. Cattell. The three basic factor–analytic research designs–their interrelations and derivatives. *Psychological Bulletin*, 49(5):499–520, 1952.

[31] J. Choi. *Real–Time Obstacle Avoiding Motion Planning for Autonomous Ground Vehicles*. PhD thesis, University of California, California, December 2010.

[32] J. Choi, R. Curry, and G. Elkaim. *Path Planning based on Bézier Curve for Autonomous Ground Vehicles*, volume 1, pages 158–166. IEEE Computer Society, 2008.

[33] J. Choi, R. Curry, and G. Elkaim. Collision free real–time motion planning for omnidirectional vehicles. In *European Control Conference, ECC*, Budapest, Hungary, August 2009.

[34] J. Choi, R. Curry, and G. Elkaim. Obstacle avoiding real–time trajectory generation and control of omnidirectional vehicles. In *American Control Conference, ACC*, pages 5510–5515, St. Louis, Missouri, June 2009.

[35] J. Choi, R. Curry, and G. Elkaim. Smooth path generation based on bézier curves for autonomous vehicles. In *World Congress on Engineering and Computer Sciences, WCECS*, pages 668–673, San Francisco, CA, October 2009.

[36] J. Choi, R. Curry, and G. Elkaim. Continuous curvature path generation based on bézier curves for autonomous vehicles. *International Journal of Applied Mathematics*, 40(2), 2010.

[37] J. Choi, R. Curry, and G. Elkaim. Curvature–continuous trajectory generation with corridor constraint for autonomous ground vehicles. In 49^{th} *IEEE Conference on Decision and Control, CDC*, Atlanta, Georgia, December 2010.

[38] J. Choi, R. Curry, and G. Elkaim. Real–time obstacle–avoiding path planning for mobile robots. In *AIAA Guidance, Navigation and Control, AIAA GNC*, Toronto, August 2010.

[39] J. Choi and G. Elkaim. Bézier curves for trajectory guidance. In *World Congress on Engineering and Computer Sciences, WCECS*, pages 625–630, San Francisco, CA, October 2008.

[40] F. Cirak, M. Ortiz, and Schröder. Subdivision surfaces: A new paradigm for thin-shell finite-element analysis. *International Journal for Numerical Methods in engineering*, 47:2039–2072, 2000.

[41] P. Comon. Tensor decompositions: State of the art and applications. pages 1–24. in Mathematics in Signal Processing V, J.G. McWhirter and I.K. Proudler, 2001.

[42] J. Connors and G. Elkaim. Analysis of a spline based, obstacle avoding path planning algorithm. *IEEE Vehicle Technology Conference*, April 2007.

[43] J. Connors and G. Elkaim. Experimental results fo spline based obstacle avoidance of an off–road ground vehicle. *ION Global Navigation Satellite Systems Conference, ION GNSS.*, September 2007.

[44] J. Connors and G. Elkaim. Manipulating b–spline based paths for obstacle avoidance in autonomous ground vehicles. In *ION National Technical Meeting, ION NTM*, pages 1081–1088, 2007.

[45] R. Coppi and S. Bolasco. *Multiway Data Analysis*. Elsevier Science, North-Holland, Amsterdam, 1989.

[46] L. De Lathauwer and B. De Moor. From matrix to tensor: Multinear algebra and signal processing. *Mathematics in Signal Processing IV, J. McWhirter and I. Proudler*, pages 1–15, 1988.

[47] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21:1253–1278, 2000.

[48] L. De Lathauwer and J. Vandewalle. Dimensionality reduction in higher–order signal processing and rank $(r_1, r_2, \cdots, r_n)$ reduction in multilinear algebra. *Linear Algebra and its Applications*, 391:31–55, 2004.

[49] Vin de Silva and Lek-Heng Lim. Tensor rank and the iil–posedness of the best low-rank approximation problem. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1084–1127, September 2008.

[50] A. Doostan and G. Laccarino. A least–squares approximation of partial differential equations with high–dimensional random inputs. *Journal of Computational Physiscs*, 228(12):4332–4345, 2009.

[51] H. Eren, C.C. Fung, and J. Evans. Implementation of the spline method for mobile robot path control. In *IEEE Instrumentation and Measurement Technology Conference*, volume 2, pages 739–744, 1999.

[52] A. Falcó. Algorithms and numerical methods for high dimensional financial market models. *Revista de Economia Financiera*, 20:51–68, 2010.

[53] A. Falcó and W. Hackbusch. On minimal subspaces in tensor representations. *Foundations of Computational Mathematics*, 2011.

[54] A. Falcó, L. Hilario, N. Montés, and M.C. Mora. Numerical strategies for the galerkin-proper generalized decomposition method. *Mathematical and Computer Modelling*, 2011.

[55] G. Farin. Trends in curve and surface design. *Computer-Aided Design*, 21(5):293–296, 1989.

[56] G. Farin. From conics to nurbs: A tutorial and survey. *IEEE Computer Graphics and Applications*, 12(5):78–86, September 1992.

[57] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design (Fifth Edition)*. Morgan Kaufmann Publishers, 2002.

[58] D. FitzGerald, M. Cranitch, and E. Coyle. Non–negative tensor factorisation for sound source separation. In *Irish Signals and Systems Conference, ISSC*, Dublin, 2005.

[59] B. Fowler and R. Bartels. Constrained–based curve manipulation. In *IEEE Computer Graphics and Application*, volume 13(5), pages 43–49, 1993.

[60] A. Fujimori, M. Teramoto, P.N. Nikiforunk, and M.M. Gupta. Cooperative collision avoidance between multiple robots. *Journal of Robotic Systems*, 17(7):347–363, 2000.

[61] B. Graf, J.M. Hostalet, and C. Schaeffer. Flexible path planning for nonholonomic mobile robots. In 4^{th} *European workshop on advanced mobile robots (EUROBOT 01)*, pages 199–206, Lund, Sweden, September 19–21 2001.

[62] A. Graham. *Kronecker Products and Matrix Calculus with Applications*. Ellis Horwood series in mathematics and its applications, 1981.

[63] V.S. Grigorascu and P.A. Regalia. Tensor displacement structures and polyspectral matching. In T. Kailath and A.H. Sayed, editors, *Fast Reliable Algorithms for Matrices with Structure*, pages 245–276. Society for Industrial and Applied Mathematics, SIAM, Philadelphia, 1999.

[64] W. Hackbusch. *Tensor spaces and numerical tensor calculus*, volume 42 of *Springer Series in Computational Mathematics*. Computational Mathematics, March 2012.

[65] W. Hackbusch, B. N. Khoromskij, S. Sauter, and E.E. Tyrtyshnikov. Use of tensor formats in elliptic eigenvalue problems. *Numerical Linear Algebra with Applications*, 19:133–151, October 2012.

[66] W. Hackbusch, B. N. Khoromskij, and E.E. Tyrtyshnikov. Hierarchical kronecker tensor–product approximations. *Journal of Numerical Mathematics*, 13:119–156, 2005.

[67] R.A. Harshman. Foundations of the parafac procedure: Models and conditions for an explanatory factor analysis. *UCLA Working Papers in Phonetics*, 16(1):84, 1970.

[68] R. Henrion. Body diagonalization of core matrices in three–way principal components analysis: Theoretical bounds and simulation. *Journal of Chemometrics*, 7:477–494, 1993.

[69] L. Hilario, A. Falcó, N. Montés, and M.C. Mora. A tensor calculus approach for bézier trajectory deformation for real-time applications in mobile robots. *International Journal for Numerical Methods in Engineering*, Under review, 2012.

[70] L. Hilario, N. Montés, A. Falcó, and M.C. Mora. Real–time trajectory deformation for potential fields planning methods. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 1567–1572, San Francisco, CA, September 2011.

[71] L. Hilario, N. Montés, A. Falcó, and F. Sánchez. An improvement of flow front computation through bézier shape deformation guaranteeing mass conservation law. In Springer-Verlag, editor, *International Journal of Material Forming*, volume 3, pages 923–926, 2010.

[72] L. Hilario, N. Montés, M.C. Mora, and A. Falcó. Real–time trajectory modificacion based on bézier shape deformation. In *International Conference on Evolutionary Computation*, pages 243–248, Valencia (Spain), October 2010.

[73] L. Hilario, N. Montés, M.C. Mora, and A. Falcó. Tensorial representation of the bézier shape deformation on robotic applications. In *Conference on Geometry Theory and Applications*, Vorau (Austria), June 2011.

[74] L. Hilario, N. Montés, M.C. Mora, and A. Falcó. A tensor calculus approach for bézier shape deformation. Valencia (Spain), June 2012. SIAM Conference on Applied Linear Algebra.

[75] L. Hilario, M.C. Mora, N. Montés, and A. Falcó. Tensor bézier trajectory deformation for potential fields planning in cluttered environments. *IEEE Transactions on Robotics*, Under review, 2012.

[76] F.L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics Physics*, 6:164–189, 1927.

[77] F.L. Hitchcock. Multiple invariants and generalized rank of a p–way matrix or tensor. *J. Math. Phys.*, 7:39–79, 1927.

[78] S.M. Hu, D.W. Zhou, and J.G. Sun. Shape modification of nurbs curves via constrained optimization. *Proceedings of the CAD/Graphics*, pages 958–962, 1999.

[79] S.M. Hu, D.W. Zhou, and J.G. Sun. Modifying the shape of nurbs surfaces with geometric constraints. *Computer-Aided Design*, 33(2):903–912, 2001.

[80] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39–41):4135–4195, October 2005.

[81] J.H. Hwang, R.C. Arkin, and D.S. Know. Mobile robots at your fingertrip: Bézier curve on–line trajectory generation for supervisory control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, volume 2, pages 1444–1449, 2003.

[82] H. Jaouni, M. Khatib, and J.P. Laumond. Elastic bands for nonholonomic car-like robots: Algorithms and combinatorial issues. In *3rd International Workshop on the Algorithmic Foundantions of Robotics (WAFR'98)*, Houston, 1998.

[83] I. Juhász. Weight–based shape modification of nurbs curves. *Computer Aided Geometric Design*, pages 377–383, 1999.

[84] I. Juhász and M. Hoffmann. Constrained shape modification of cubic b–spline curves by means of knots. *Computer-Aided Design*, 36:437–445, 2004.

[85] C.T. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics, SIAM, Raleigh, North Carolina, 1999.

[86] M. Khatib, H. Jaouni, R. Chatila, and J.P. Laumond. Dynamic path modification for car–like nonholonomic mobile robots. In *IEEE International Conference on Robotics and Automation, ICRA*, volume 4, pages 2920–2925, 1997.

[87] O. Khatib. Real–time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[88] H.A.L. Kiers. A three–step algorithm for candecomp/parafac analysis of large data sets with multicollinearity. *Journal of Chemometrics*, 12:171–255, 1998.

[89] D.E. Knuth. *The Art of Computer Programming: Seminumerical algorithms*. The Art of Computer Programming. Addison-Wesley Pub. Co., 1981.

[90] T. G. Kolda. Orthogonal tensor decompositions. *SIAM Journal on Matrix Analysis and Applications*, 23(1):243–255, 2001.

[91] T. G. Kolda and B. W. Bader. The tophits model for higher–order web link analysis. In *Workshop on Link Analysis*. Counterterrorism and Security, 2006.

[92] T. G. Kolda, B. W. Bader, and J. P. Kenny. Higher–order web link analysis using multi-linear algebra. In 5^{th} *IEEE International Conference on Data Mining*, pages 242–249. IEEE Computer Society, 2005.

[93] T.G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[94] K. Komoriya and K. Tanie. Trajectory design and control of a wheel–type mobile robot using b–spline curve. In *International Workshop on Intelligence Robots and Systems*, pages 389–405, 1989.

[95] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *IEEE Conference on Robotics and Automation*, pages 1398–1404, Sacramento, California, 1991.

[96] P.M. Kronenberg. *Applied Multiway Data Analysis*, volume 74. Springer New York, 2008.

[97] P. M. Kroonenberg. *Three–Mode Principal Component Analysis: Theory and Applications*. PhD thesis, DWO Press, Leiden, The Netherlands, 1983.

[98] J. B. Kruskal. Three–way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear Algebra and its Applications*, 18(2):95–138, 1977.

[99] F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for nonholonomic mobile robots. In *IEEE Transactions on Robotics and Automation*, volume 20, pages 967–977, 2004.

[100] W. Landry. Implementing a high performance tensor library. *Scientific Programming*, 11(4):273–290, 2003.

[101] J.M. Landsberg. The border rank of the multiplication of 2*2 matrices is seven. *Journal of the American Mathematical Society*, 19(2):447–459, September 2005.

[102] L. De Lathauwer, B. De Moor, and J. Vandewalle. On the best rank–1 and rank–$(r_1, r_2, \cdots, r_n)$ approximation of higher order tensors. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1324–1342, 2000.

[103] S. Leurgans and R.T. Ross. Multinear models: Applications in spectroscopy. *Statistical Science*, 7:289–310, 1992.

[104] N. Liu, B. Zhang, J. Yang, Z. Chen, W. Liu, F. Bai, and L. Chien. Text representation: From vector to tensor. In 5^{th} *IEEE International Conference on Data Mining*, ICDM'05, pages 725–728, Washington, DC, USA, 2005. IEEE Computer Society.

[105] M. Lizarraga and G. Elklaim. Spatially deconflicted path generation for multiple uavs in a bounded airspace. In *IEEE/ION Position, Location and Navigation Symposium*, pages 633–640, 2008.

[106] J.R. Lucena, Coronel Paúl Coronel, and Y. Orellana. Historia, evolución, estado actual y futuro de la cirugía robótica. *Revista de la Facultad de Medicina*, 30(2):109–114, 2007.

[107] X. J. Luo, M.S. Shephard, and J.F. Remacle. The influence of geometric approximation on the accuracy of high order methods. (1), 2001.

[108] X. J. Luo, M.S. Shephard, J.F. Remacle, R.M. O'Bara, M.W. Beall, B. Szabo, and R. Actis. p-version mesh generation issues. (1):343–354, 2002.

[109] J.R. Magnus and H. Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics (Third Edition)*. John Wiley, 2007.

[110] D. Marsh. *Applied Geometry for Computer Graphics and CAD (Second Edition)*. Springer, 2005.

[111] E. Martínez-Montes, P.A. Valdés-Sosa, F. Miwakeichi, R. I. Goldman, and M.S. Cohen. Concurrent eeg/fmri analysis by multiway partial least squares. *NeuroImage*, 22:1023–1034, 2004.

[112] D.S. Meek, B.H. Ong, and D.J. Walton. Constrained interpolation with rational cubics. *Computer Aided Geometric Design*, 20(5):253–275, August 2003.

[113] F. Miwakeichi, E. Martínez-Montes, P.A. Valds-Sosa, N. Nishiyama, H. Mizuhara, and Y. Yamaguchi. Decomposing eeg data into space–time–frequency components using parallel factor analysis. *NeuroImage*, 22:1035–1045, 2004.

[114] N. Montés. *Marco Computacional para el Diseño, optimización y control de procesos de moldeo con resinas líquidas (LCM)*. PhD thesis, Universidad Cardenal Herrera CEU, 2009.

[115] N. Montés, A. Herráez, L Armesto, and J. Tornero. Real-time clothoid approximation by rational bézier curves. In *International Conference on Robotics and Automation, ICRA*, pages 2246–2251, May 2008.

[116] N. Montés, M.C. Mora, and J. Tornero. Trajectory generation based on rational bézier curves as clothoids. In *Intelligent Vehicles Symposium, IEEE*, volume 505, pages 505–510, June 2007.

[117] N. Montés, F. Sánchez, L. Hilario, and A. Falcó. Flow numerical computation through bézier shape deformation for lcm process simulation methods. *International Journal of Material Forming. Lyon(France)*, 1:919–922, 2008.

[118] M.C. Mora. *Planificación de Movimientos Predictiva y Multifrecuencial con Campos Potenciales Artificiales*. PhD thesis, Universidad Politécnica de Valencia, Valencia (Spain), Diciembre 2009.

[119] M.C. Mora, R. Pizá, and J. Tornero. Multirate obstacle tracking and path planning for intelligent vehicles. In *IEEE Intelligent Vehicles Symposium*, pages 172–177, 2007.

[120] M.C. Mora and J. Tornero. Planificación de movimientos mediante la propagación de campos potenciales artificiales. In *Congreso Iberoamericano de Ingeniería mecánica, Octava Edición.*, October 2007.

[121] M.C. Mora and J. Tornero. Path planning and trajectory generation using multi–rate predictive artificial potential-fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 2990–2995, 2008.

[122] M. Mørup, L. K. Hansen, and S. M. Arnfred. Algorithms for sparse non–negative TUCKER. 20(8):2112–2131, August 2008.

[123] M. Mørup, L. K. Hansen, J. Parnas, and S. M. Arnfred. Decomposing the time–frequency representation of EEG using non-negative matrix and multi-way factorization. Technical report, 2006.

[124] K. Nagatani, Y. Iwai, and Y. Tanaka. Sensor based navigation for car–like mobile robots using generalized voroni graph. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, volume 2, pages 1017–1022, 2001.

[125] A. Neto, D.G. Macharet, and M.F.M. Campos. Feasible rrt-based path planning using seventh order bézier curves. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 1445–1450, Taipei (Taiwan), October 2010.

[126] A. Nouy. A generalized spectral decomposition technique to solve a class of linear stochastic partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 196(45–48):4521–4537, 2007.

[127] A. Nouy. Proper generalized decompositions and separated representations for the numerical solution of high dimensional stochastic problems. *Archives of Computational Methods in Engineering*, 17(4):403–434, 2010.

[128] Gerhard Opfer and Hans Joachim Oberle. The derivation of cubic splines with obstacles by methods of optimization and optimal control. *Numerische Mathematik*, 52(1):17–31, 1987-01-01.

[129] Paluszny-Prautzch-Boehm. *Métodos de Bézier y B-Splines*. Springer-Verlag, 2002.

[130] L. Piegl. Modifying of the shape of rational b-spline part 1: Curves. *Computer-Aided Design*, 21(10):509–518, 1989.

[131] L. Piegl. Modifying of the shape of rational b-spline part 2: Surfaces. *Computer-Aided Design*, 21(9):538–546, 1989.

[132] L. Piegl. On nurbs: A survey. In *IEEE Computer Graphics and Applications*, volume 11(1), pages 55–71, 1991.

[133] L. Piegl and W. Tiller. Curve and surface constructions using rational b–splines. *Computer-Aided Design*, 19(9):485–498, 1987.

[134] R. Pizá. *PhD Thesis Modelado del entorno y localización de robots móviles autónomos mediante técnicas de muestreo no convencional*. PhD thesis, Universidad Politécnica de Valencia, Valencia (Spain), 2003.

[135] Wu Qingbiao and Tao Shuyi. Shape modification of b–splines curves via constrained optimization for multitarget points. In *Computer Science and Information Engineering, WRI World Congress*, volume 1, pages 733–737, 2009.

[136] Abedallah Rababah and Mohammad Al-Natour. The weighted dual functions for the univariate bernstein basis. *Applied Mathematics an Computation.*, 186(2):1581–1590, 2007.

[137] D. Salomon. *Curves and Surfaces for Computer Graphics*. Springer, 2006.

[138] F. Sánchez. *Propuesta de un esquema numérico eficiente para el tratamiento de los problemas de transporte en la simulacióon del moldeo por transferencia de resina*. PhD thesis, Universidad Politécnica de Valencia, 2004.

[139] F. Sánchez, L.I Gascon, F.A. Garcia, and F. Chinesta. On the incubation time computation in resin transfer molding process simulation. *International Journal of Forming Processes*, Special Issue:51–67, 2005.

[140] F.M. Sánchez, P. Jiménez, F. Millán, J. Salvador, V. Monllau, J. Palou, and H. Villavicencio. Historia de la robótica: de arquitas de tarento al robot da vinci (parte ii). *Actas urológicas españolas*, March 2007.

[141] R.J. Sánchez. A simple technique for nurbs shape modification. In *IEEE Computer Graphics and Applications*, volume 17(1), pages 52–59, 1997.

[142] B. Savas. Analyses and tests of handwritten digit recognition algorithms. Master's thesis, Linköping University, Sweden, 2003.

[143] R. Sevilla, S. Fernandez-Mendez, and A. Huerta. Nurbs-Enhanced Finite Element Method (NEFEM). *Archives of Computational Methods in Engineering*, 18(4):441–484, 2007.

[144] Z. Shiller and Y.R. Gwo. Dynamic motion planning of autonomous vehicles. In *IEEE Transactions on Robotics and Automation*, volume 7, page 241, 1991.

[145] N. Sidiropoulos, R. Bro, and G. Giannakis. Parallel factor analysis in sensor array processing. *IEEE Transactions on Signal Processing*, 48(8):2377–2388, August 2000.

[146] I. Skrjanc and G. Klancar. Cooperative collision avoidance between multiple robots based on bézier curves. In *International Conference on Information Technology Interfaces*, pages 451–456, 2007.

[147] A. Smilde, R. Bro, and P. Geladi. *Multi–Way Analysis: Applications in the Chemical Sciences*. John Wiley '—&' Sons, 2004.

[148] A.K. Smilde, Y. Wang, and B.R. Kowalski. Theory of medium–rank second-order calibration with restricted–tucker models. *Journal of Chemometrics*, 8:21–36, 1994.

[149] Maureen C. Stone, Xerox Parc, and Seattle) DeRose, Tony D. (University of Washington. A geometric characterization of parametric cubic curves. *ACM Transactions on Graphics, TOG*, 8(Issue 3), 1989.

[150] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969-08-29.

[151] J.-T. Sun, H.-J. Zeng, H. Liu, Y. Lu, and Z. Chen. Cubesvd: A novel approach to personalized web search. In 14^{th} *International Conference on World Wide Web*, WWW'05, pages 382–390, New York, NY, USA, 2005. ACM.

[152] N. Tatematsu and K. Ohnishi. Tracking motion of mobile robot for moving target using nurbs curve. In *International Conference on Industrial Technology. San Francisco (USA)*, volume 1, pages 245–249, 2003.

[153] D. Terzopolous and H. Qin. Dynamic nurbs with geometric constraints for interactive sculpting. *ACM Transactions on Graphics*, 13(2):103–136, 1994.

[154] W. Tiller. Rational b–splines for curve and surface representation. *IEEE Computer Graphics and Applications*, 3(6):61–19, September 1983.

[155] J. Tornero, J. Salt, and P. Albertos. *lq* optimal control for multirate sampled data systems. In *IFAC World Congress,* 14^{th} *Edition*, pages 211–216, 1999.

[156] L.R. Tucker. Implications of factor analysis of three-way matrices for measurement of change. In C.W. Harris, editor, *Problems in measuring change*, pages 122–137. University of Wisconsin Press, Madison WI, 1963.

[157] L.R. Tucker. The extension of factor analysis to three-dimensional matrices. In H. Gulliksen and N. Frederiksen, editors, *Contributions to mathematical psychology*, pages 110–127. Holt, Rinehart and Winston, New York, 1964.

[158] L.R. Tucker. Some mathematical notes on three–mode factor analysis. *Psychometrika*, 31(3):279–311, 1996.

[159] Cristina Urdianes García. *Introducción a la robótica*. Departamento de Tecnología Electrónica, Universidad de Málaga, Málaga.

[160] C. F. Van Loan. The ubiquitous kronecker product. *Journal Computation and Applied Mathematics*, 123(1-2):85–100, 2000.

[161] M.A.O. Vasilescu and D. Terzopoulos. Multinear analysis of image ensembles: tensorfaces. In Lecture Notes in Computer Science, editor, 7^{th} *European Conference on Computer Vision, ECCV*, volume 2350 of *Springer*, pages 447–460, May 2002.

[162] G.B. Vázquez, A.H. Sossa, and Diaz de Leon. S. Auto guided vehicle control using expanded time b–spline. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2786–2791, 1994.

[163] K.J. Versprille. *Computer–Aided Design Applications of the Rational B-Spline Approximation form*. Phd thesis, Syracuse University, 1975.

[164] G. Vidal. Efficient classical simulation of slightly entangled quantum computations. *Physical Review Letters*, 91(14):147902–147906, October 2003.

[165] H. Wang and N. Ahuja. Compact representation of compact representation of multidimensional data using tensor rank–one decomposition. In 17^{th} *International Conference on Pattern Recognition, ICPR*, volume 1, pages 44–47, 2004.

[166] H.L. Wu, M. Shibukawa, and K. Oguma. An alternating trilinear descompostion algorithm with application to calibration of hplc-dad for simultaneos determination of overlapped chlorinated aromatic hydrocarbons. *Journal of Chemometrics*, 12:1–26, 1998.

[167] O.B. Wu and F.H. Xia. Shape modification of bézier curves by constrained optimization. *Journal of Zhejiang University SCIENCE*, pages 124–127, 2005.

[168] L. Xu, Y.J Chen, and N. Hu. Shape modification of bézier curves by constrained optimization. *Journal of Software*, 13(6):1069–1074, 2002.

[169] D. Xue and L. Demkowicz. Control of geometry induced error in hp finite element (fe) simulations i. evaluation of fe error for curvilinear geometries. *International Journal of Numerical Analysis and Modelling*, 2(3):283–300, 2005.

[170] M. Yamamoto, M. Iwamura, and Mohri. Quasi time-optimal motion planning of mobile platforms in the presence of obstacles. In *International Conference on Robotics and Automation, ICRA*, pages 2958–2963, 1999.

[171] J. Zhang, J. Raczkowsky, and A. Herp. Emulation of spline curve and its applications in robot motion control. In *IEEE International Conference on Fuzzy Systems*, volume 2, pages 831–836, 1994.