CAPÍTULO 6. MÓDULO GENERADOR DE TRAYECTORIAS.

6.1. Introducción.

Como se dijo en el capítulo anterior, existe un vacío entre el *Module Manager* y el módulo del seguidor de caminos a la hora del paso de *waypoints*. En efecto es necesario un módulo que genere una trayectoria a partir de los *waypoints* que reciba del *Module Manager* y se la envíe al seguidor a través de un *slot* tipo *DATOS*. Aunque la inclusión de este módulo no estaba prevista para el presente proyecto de final de carrera, se decidió su adición al mismo dada su utilidad y la necesidad de crearlo más tarde o más temprano.

6.2. Estructuración.

Al igual que con el resto de módulos, para el generador de trayectorias se ha seguido el modelo de programación de *CROMAT*. Como se explicó en el capítulo anterior éste consiste en separar las tareas de comunicaciones y cálculos en dos hilos diferentes, que se comunican a través de una sección crítica, donde las variables están protegidas por *mutex*.

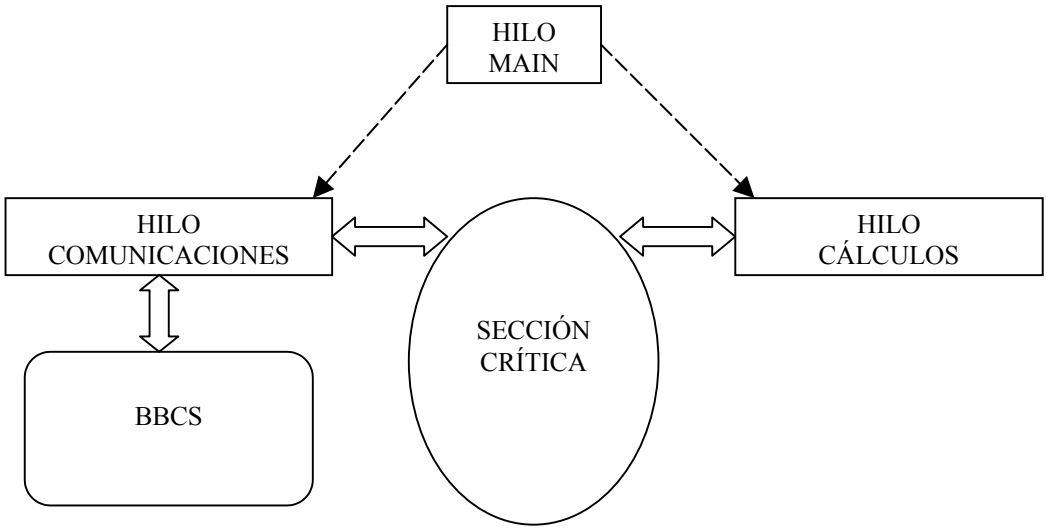


Figura 6-1. Esquema del módulo generador de trayectorias.

6.2.1. Interfaces.

Para este módulo se tendrán únicamente dos interfaces: una de entrada y otra de salida. Por la interfaz de entrada llega una estructura de tipo *DATO_GEN_TRAJ*, que se define a continuación:

```
typedef struct
{
     PUNTO_GEN_TRAJ waypoints[GOTOLIST_XYZ_MAXLENGTH];
     float speed;
     int length;
}DATO_GEN_TRAJ;

typedef struct
{
     float speed;

     CromatPositioningType posType;

     int ReferenceEllipsoid;
     char UtmZone[20];
     float UtmEasting;
     float UtmNorthing;

     double latitude;
     double longitude;
     float altitude;

     float heading;
     char forceHeading;

     char pad[3];

}PUNTO_GEN_TRAJ;
```

La interfaz de salida se hace por un *slot* tipo *DATOS* para el seguidor de caminos (ver capítulo 4).

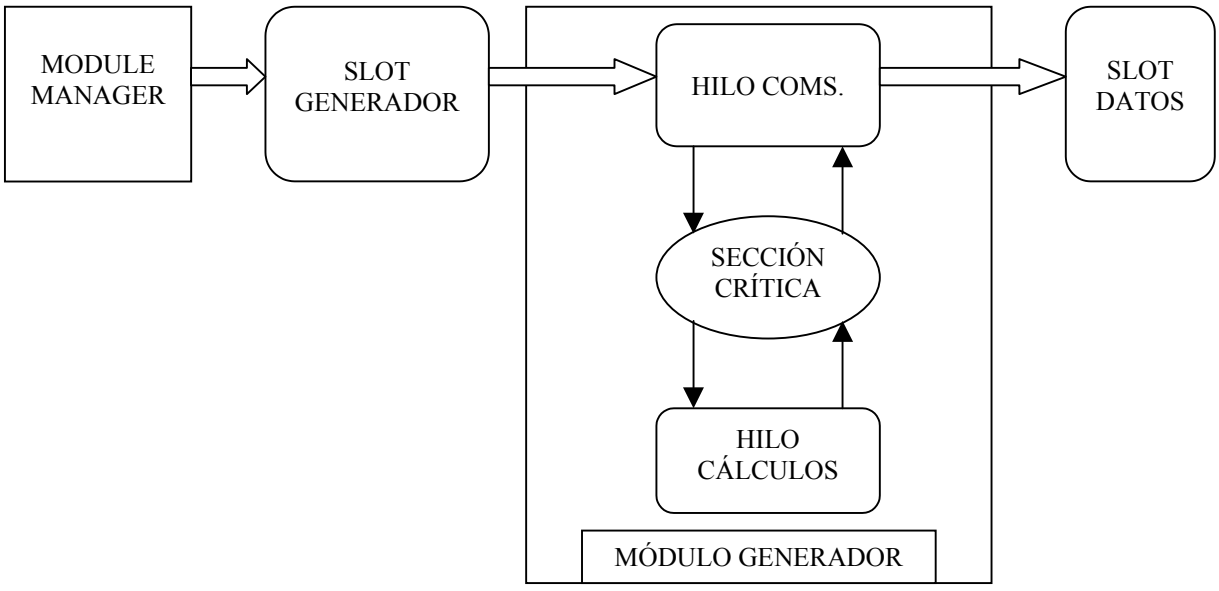


Figura 6-2. Esquema de las comunicaciones del módulo generador.

6.2.2. Hilos.

6.2.2.1. Hilo main.

Se encarga simplemente de lanzar el resto de hilos y esperar su terminación.

6.2.2.2. Hilo de cálculos (genalgorithmthread.cpp).

Se compone básicamente de un bucle, donde cada 50 ms se buscan datos nuevos en la sección crítica. En caso de haberlos se recogen, se genera una trayectoria a partir de ellos y se transforma la trayectoria al formato del *slot DATOS*. El resultado se guarda en la sección crítica.

6.2.2.3. Hilo de comunicaciones(gencommunications.cpp).

Inicializa y pone a punto el *BBCS*. Prueba a recoger *waypoints* del *slot* de salida del *Module Manager* para guardarlo en la sección crítica, y saca datos de la misma procedentes del hilo de cálculos para enviarlos al *slot DATOS*.

6.3 Algoritmos.

Para la generación de trayectorias se han implementado dos algoritmos. Se escogerá uno u otro según las necesidades del usuario. La elección se hace a nivel de compilación. Basta cambiar una única línea en el código del hilo de cálculos para que se utilice uno u otro. De esta manera queda muy sencillo el cambio o la ampliación, ya que si se desea implementar un nuevo algoritmo generador basta con programarlo y cambiar la citada línea. El único requisito que ha de cumplir la nueva función es que respete el siguiente prototipo:

char *NombreAlgoritmo(DATO_GEN_TRAJ WPlist, char *buffer, int *p_size)

Argumentos de entrada:

DATO_GEN_TRAJ WPlist:
Lista de *waypoints* a partir de los cuales se debe generar la trayectoria.

*char *buffer:*
Puntero a un buffer donde almacenar la trayectoria.

*int *p_size:*
Puntero a un entero donde se guarda el tamaño del *buffer*.

Argumento de salida:

Puntero a *char* que llevará la dirección del *buffer*.

6.3.1. Algoritmo simple.

El primer algoritmo que se programó es el más sencillo de todos los posibles: devolver los mismos puntos que se recogen del *Module Manager*, solo que cambiando su formato convenientemente.

Aunque la implementación de este algoritmo puede parecer algo absurda, no lo es tanto teniendo en cuenta que de esta manera se conoce a priori la trayectoria exacta del *UAV*. El helicóptero volará en línea recta de un punto a otro.

El problema que tienen este tipo de trayectorias es que son curvas de clase C^0 , es decir continuas, pero con derivadas discontinuas. Esto implica velocidades discontinuas, por lo que el aparato se tendrá que detener en cada *waypoint*, cambiar la dirección de la velocidad y dirigirse al objetivo siguiente.

6.3.2. Algoritmo *spline*.

6.3.2.1. Especificaciones.

Debe pasar por todos los *waypoints* que se le hayan indicado.

Por pura lógica. No obstante podrían hacerse trayectorias que simplemente se acercaran a dichos puntos, suavizando de esta manera la curva. Se ha estimado oportuno que, por ser éste el primer algoritmo que se implementa, se pase por todos los puntos.

Curva continua al menos para la posición y la primera derivada (clase C^1).

De esta manera la velocidad será siempre continua, evitando los problemas que tenía el algoritmo simple.

Para cada par de *waypoints* consecutivos las alturas máxima y mínima en el intervalo han de estar en los mencionados puntos.

Al crear una trayectoria a partir de una serie de puntos no se sabe a priori con exactitud cómo va a ser la misma. Un buen diseño ha de procurar no llevar la trayectoria por lugares peligrosos. Se entiende que no hay problema en sobrevolar todo el área que se quiera, por lo que no se imponen restricciones en cuanto la superficie. No sucede lo mismo con la altitud. Una altitud excesiva puede llevar problemas de vuelo. Mucho más grave es el problema de una altura demasiado baja, que puede llevar a colisiones o directamente a un choque contra el suelo. Resulta por tanto indispensable mantener un control estricto de la altura.

6.3.2.2. Solución elegida.

Tras estudiar las especificaciones se decidió que la trayectoria a crear sería de tipo *spline* cúbico natural para la latitud y la longitud, y de tipo cúbica para la altura.

6.3.2.2.1. Splines cúbicos.

De entre la infinidad de curvas que cumplen las especificaciones se ha escogido este tipo de trayectorias por ser suaves y relativamente sencillas de programar. Los *splines* cúbicos se han venido utilizando en robótica desde hace mucho tiempo, tanto en robots móviles como anclados; por lo que a las ventajas citadas antes hay que añadirle la seguridad que otorga el uso de un algoritmo altamente contrastado a lo largo de los años.

Un *spline* cúbico no es más que una curva de interpolación entre varios puntos, de manera que entre cada dos puntos consecutivos haya una curva definida por una función cúbica.

En los puntos de intersección de dichas funciones se han de cumplir condiciones de continuidad hasta la segunda derivada. Aun con esas restricciones todavía quedan dos libertades. Tales libertades podrían cubrirse con dos especificaciones (por ejemplo, la pendiente en los puntos inicial y final). Sin embargo haría falta aportar dos datos más al generador, y esto complica la interfaz de entrada innecesariamente. Resulta más lógico imponer dos restricciones más. Según de qué restricciones se trate el *spline* será de un tipo o de otro.

Las dos tipos de *splines* más conocidos son el *spline 'Not-a-knot'* y el natural. En el primero las restricciones que se añaden son la continuidad de la tercera derivada en el primer y el último punto de intersección. Por su parte el *spline* natural impone una segunda derivada nula en los puntos inicial y final de la trayectoria.

Tras realizar diversas pruebas se comprobó que los dos *splines* se parecían mucho en los tramos intermedios, pero el *spline* natural proporciona una curva mucho más suave para los tramos inicial y final. Consecuentemente se decidió utilizar este último.

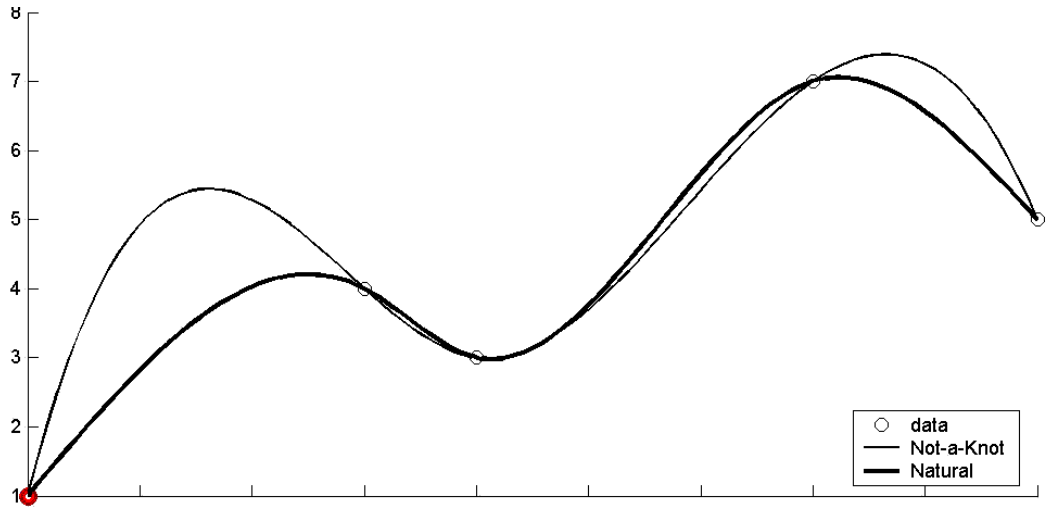


Figura 6-3. Comparación entre el *spline* natural y el *'not-a-knot'*.

6.3.2.2.2. Función cúbica para la altura.

En la figura 6-3 anterior se muestra con claridad la ventaja del uso del *spline* natural sobre el *'not-a-knot'*. También se ve que para la altura las curvas han de ser de otra clase, ya que se incumple claramente el requisito de que en cada intervalo los valores máximo y mínimo estén en los extremos.

Una primera solución al problema sería utilizar funciones lineales. Sin embargo habría que renunciar a la continuidad de las primeras derivadas. Algo parecido sucedería con funciones de segundo grado. Sin embargo sí es posible el uso de funciones de tercer grado que cumplan la restricción del máximo y mínimo teniendo continuidad en la velocidad.

En efecto, una función de tercer grado puede tener hasta dos puntos críticos. Por lo tanto basta forzar que esos dos puntos estén en los *waypoints* que definen el intervalo. Esto se consigue haciendo nula la primera derivada en los extremos de cada tramo.

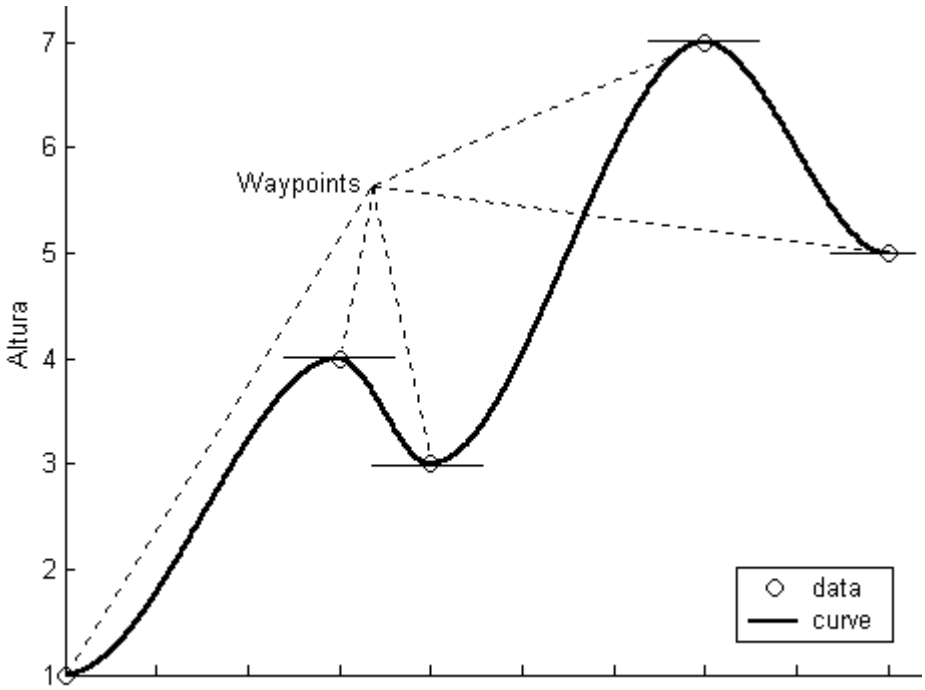


Figura 6-4. Curvas cúbicas para la altura.

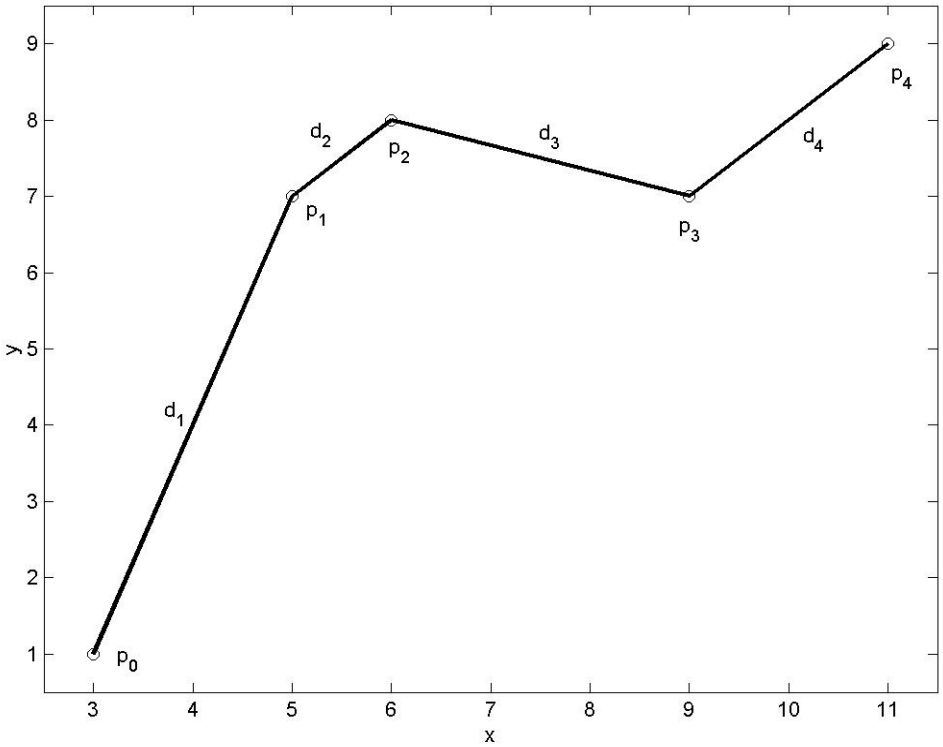
En la figura 6-4 se ve cómo al coincidir todos los puntos críticos con los *waypoints* la altura en cada intervalo siempre se mantiene entre los valores de los extremos, tal y como se deseaba.

6.3.2.3. Splines en el plano.

6.3.2.3.1. Parámetro común.

Una vez determinados los métodos para generar la trayectoria se pasa al diseño de la misma. Hasta ahora se ha hablado de funciones cúbicas de una magnitud (longitud, latitud, altura) respecto a una variable de la que aún no se ha comentado nada. No es posible hallar directamente un *spline* entre la latitud y la longitud, ya que para ello al menos una de esas dos magnitudes debería ser monoevaluada respecto a la otra. Es evidente que en una trayectoria real no se tiene el porqué dar tal caso. Es necesario crear un parámetro común cuyo valor crezca según vaya siendo mayor el número orden del *waypoint* en la lista. Es decir, para el punto uno ese parámetro ha de ser menor que para el punto dos, y a su vez éste menor que para el tercero.

El parámetro escogido (a partir de ahora parámetro 't') ha sido la distancia horizontal mínima desde el punto al punto origen, pasando por los puntos anteriores. Esta distancia resulta ser la suma de las longitudes de los segmentos que unen los pares de puntos consecutivos, desde el primero hasta el punto para el que se esté haciendo el cálculo. El concepto queda mucho más clara en la figura 6-5.



Punto	t
p_0	0
p_1	d_1
p_2	d_1+d_2
...	...
p_n	$d_1+d_{2+...}$ $...+d_n$

Figura 6-5. Distancias horizontales.

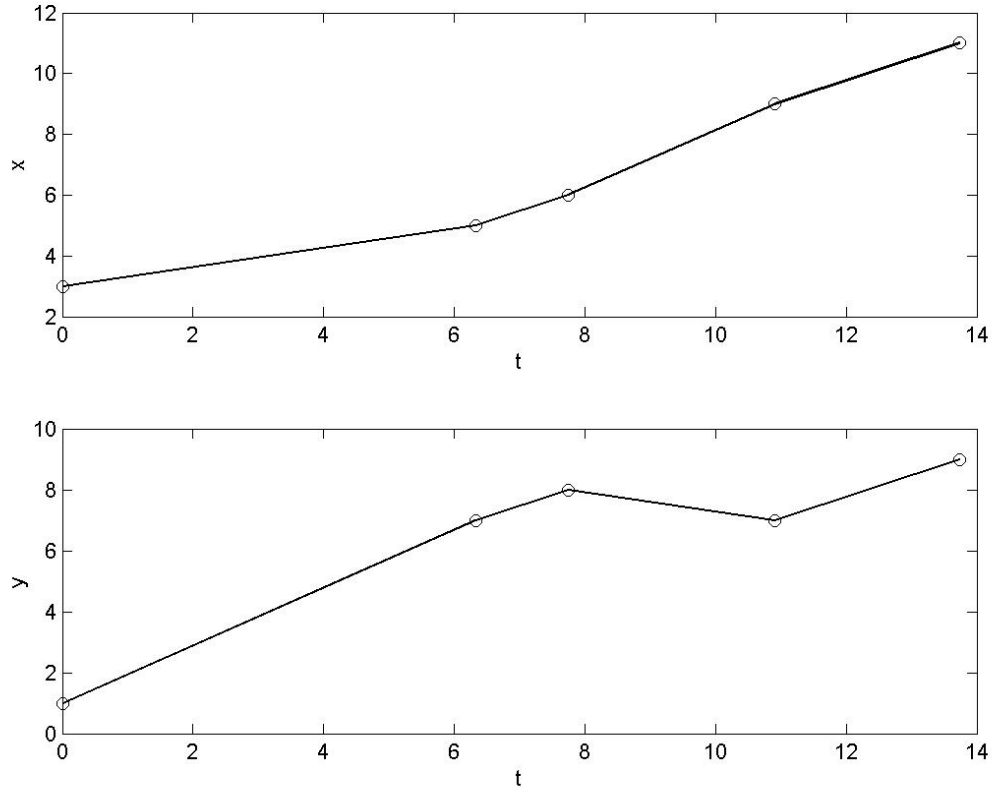


Figura 6-6. Variables x e y frente al parámetro t.

6.3.2.3.2. Cálculo de las distancias.

Hasta ahora sólo se ha expuesto la parte matemática del diseño. Llevando el mismo a la realidad aparecen algunos problemas. A la hora de calcular las distancias de los puntos, resulta que la latitud y la longitud se miden en grados, mientras que la altura se mide en metros. Podrían convertirse las coordenadas de latitud-longitud a *UTM*, pero se necesitarían más datos, y se complicaría el algoritmo. Como por regla general las distancias horizontales serán mucho mayores que las verticales, se puede prescindir de últimas para el cálculo de las distancias, y por tanto del parámetro *t*.

Sin embargo el problema aún no queda del todo resuelto. Aunque la latitud y la longitud se midan en grados, el arco que barre un grado de latitud no es el mismo que el que barre uno de longitud. Los meridianos tienen su centro en el centro de la tierra, mientras que los paralelos lo tienen en el punto más cercano del eje de rotación.

Para equiparar las dos medidas basta con multiplicar la latitud por el coseno de sí misma. Esta solución no aporta resultados exactos, ya que la tierra no es perfectamente esférica. Sin embargo no se necesita tanta precisión, ya que se trata del parámetro que seguirán las curvas, por lo que no afecta de manera directa a la trayectoria final.

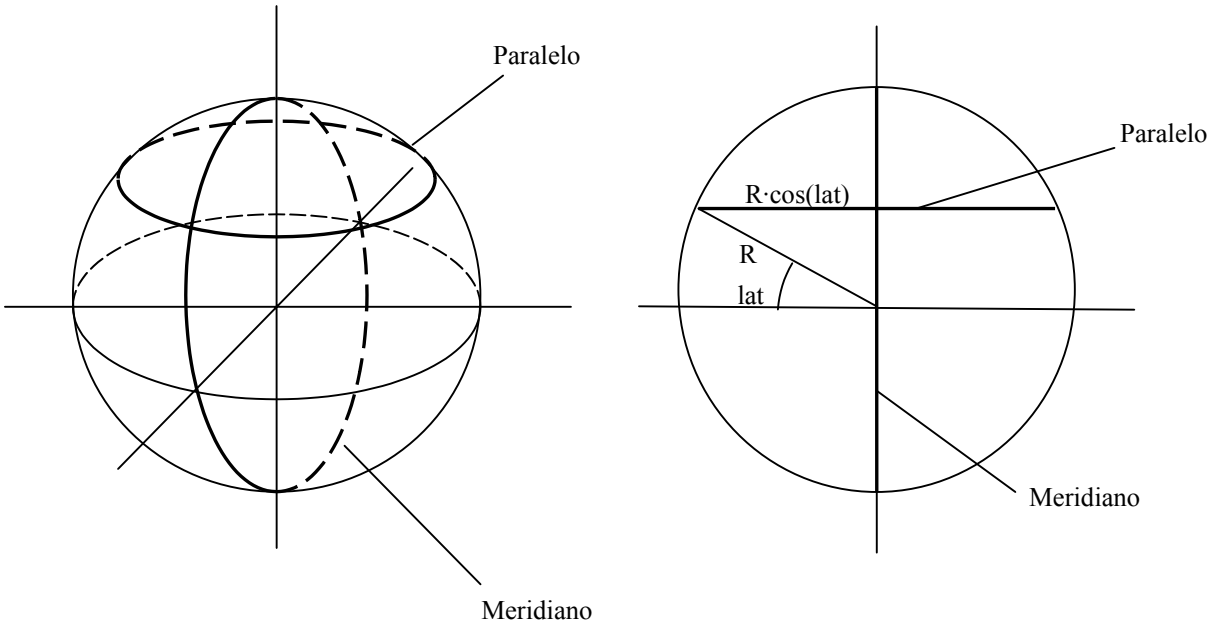


Figura 6-7. La tierra. Vista general y de perfil.
Meridianos y paralelos.

Un posible punto peligroso de esta ponderación es aquél en el que el coseno se acerca a cero. El coseno de un ángulo está cercano a cero cuando el ángulo se aproxima a noventa grados, positivos o negativos. Los puntos de la tierra con tales latitudes son los polos. Allí siempre va a haber dificultad a la hora de usar la longitud, ya que el avance de pocos metros puede implicar barrer un ángulo muy grande. De todas maneras ni el helicóptero ni la electrónica están mecánicamente preparados para las condiciones climatológicas de los polos, por lo que carece de importancia que el generador de trayectorias pueda fallar en las circunstancias descritas.

6.3.2.3.3. Cálculo de los coeficientes.

Una vez se tienen los parámetros asociados a los puntos ya se puede comenzar a calcular la trayectoria en sí. Lo primero de todo será hallar los coeficientes de las ecuaciones de las curvas cúbicas en cada intervalo. Esto se puede hacer a partir del método que se describe a continuación.

Ecuación para cada tramo :

$$S_i(t) = D_i + (t - t_i)\cdot\left(C_i + (t - t_i)\cdot\left(B_i + (t - t_i)\cdot A_i\right)\right)$$

Donde :

$$A_i = \frac{1}{6h_i}\left(z_{i+1} - z_i\right)$$

$$B_i = \frac{z_i}{2}$$

$$C_i = \frac{-h_i}{6} z_{i+1} - \frac{h_i}{3}\cdot z_i + \frac{1}{h_i}\left(x_{i+1} - x_i\right)$$

$$D_i = x_i$$

Variables intermedias :

$$h_i = t_{i+1} - t_i$$

$$u_i = 2\left(h_i - h_{i-1}\right) - \frac{h^2{}_{i-1}}{u_{i-1}}$$

$$b_i = \frac{6}{h_i}\left(x_{i+1} - x_i\right)$$

$$v_i = b_i - b_{i-1} - \frac{h_{i-1} v_{i-1}}{u_{i-1}}$$

$z_0 = z_n = 0$ Condiciones del spline natural.

$$z_i = \frac{\left(v_i - h_i z_{i+1}\right)}{u_i}$$

Estas variables se hallan con facilidad utilizando varios bucles (ver anexo 7).

6.3.2.3.4. Cálculo de los puntos intermedios.

Quizás una de las partes más complicadas del diseño de un generador de trayectorias sea decidir la manera de elegir la cantidad y ubicación de los puntos intermedios en cada tramo. Existen básicamente dos tipos de método: distancias fijas y distancias variables.

El primer método es muy sencillo, y consiste en dividir cada intervalo en partes iguales. Esta manera de operar suele proporcionar buenos resultados. Sin embargo no es una forma óptima de describir la curva.

Resulta mucho más provechoso aplicar un algoritmo de distancias variables. De esta manera las partes más lineales de las curvas tendrán pocos puntos intermedios, al contrario que las no lineales.

Para este generador se ha diseñado un método que calcula la derivada de los *spline*s, obteniendo así la pendiente. Tras esta operación se pasa la pendiente a ángulo y se eligen los puntos intermedios de manera que entre ellos siempre haya la misma diferencia de pendiente. Concretando algo más, para cada tramo se siguen los siguientes pasos:

1) Se calcula la arcotangente de la derivada de la función cúbica (pendiente en radianes) tanto para la latitud como para la longitud.

2) Las ecuaciones resultantes (arcotangentes de una ecuación de segundo grado) tienen un único punto crítico cada una. Se calculan las pendientes máximas y mínimas para el intervalo tanto en latitud como en longitud, y se escoge aquella medida que tenga mayor diferencia, y se trabajará sólo con ella.

3) Si el punto crítico está en el intervalo, el tramo se divide en dos, cortando por el punto crítico.

4) Se divide la diferencia de las pendientes de los extremos entre un ángulo determinado (este ángulo se puede cambiar en el código). El resultado se redondea al alza, y se divide el tramo en ese número de partes, de manera que cada parte tenga la misma diferencia de pendientes.

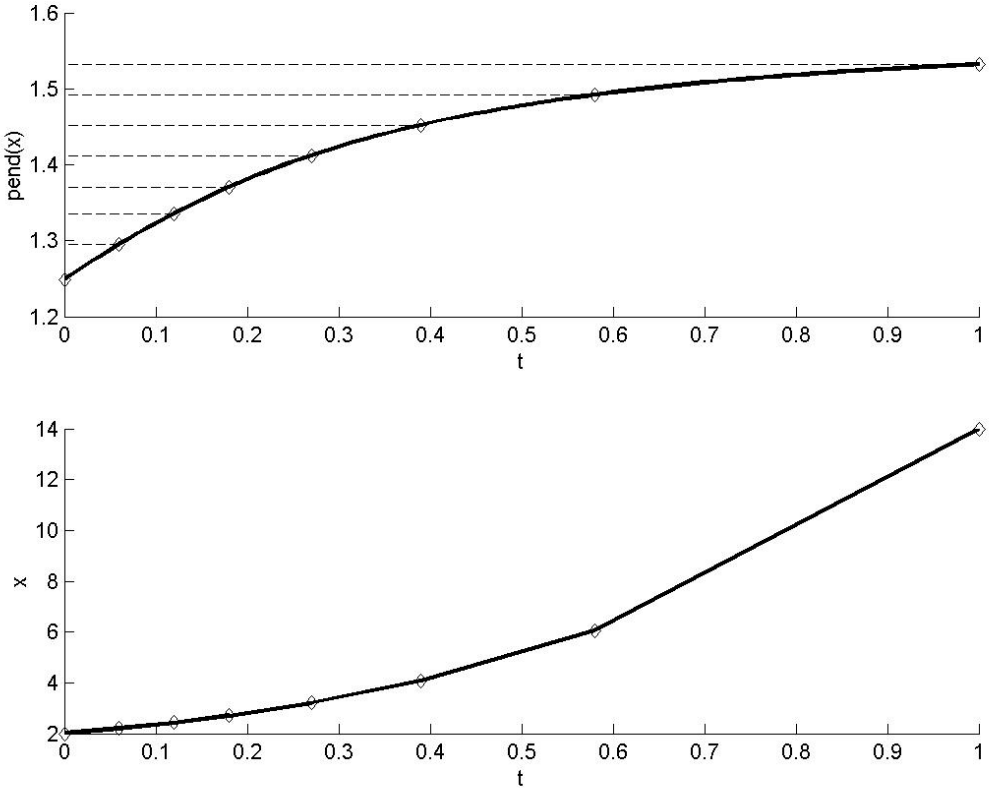


Figura 6-8. División del intervalo en trozos con la misma diferencia de pendientes

En la figura 6-8 se observa cómo se obtienen más puntos en las zonas donde más varía la pendiente, tal y como se pretendía.

6.3.2.3.5. Composición de la curva en el plano.

En este punto se tienen dos *spline*s: una para la latitud y otra para la longitud. Ambas curvas dependen del mismo parámetro, y tienen los puntos situados en los mismos valores del mismo. Para obtener la trayectoria en el plano sólo queda hacer corresponder cada uno de los puntos con los de la otra magnitud.

En las figuras siguientes se muestra un ejemplo ilustrativo, en coordenadas cartesianas. En particular se ha hallado una curva para los siguientes puntos:

x=[0 -500 -500 500 500 -500 -500 500 500 0] m
y=[0 0 200 200 400 400 600 600 0 0] m

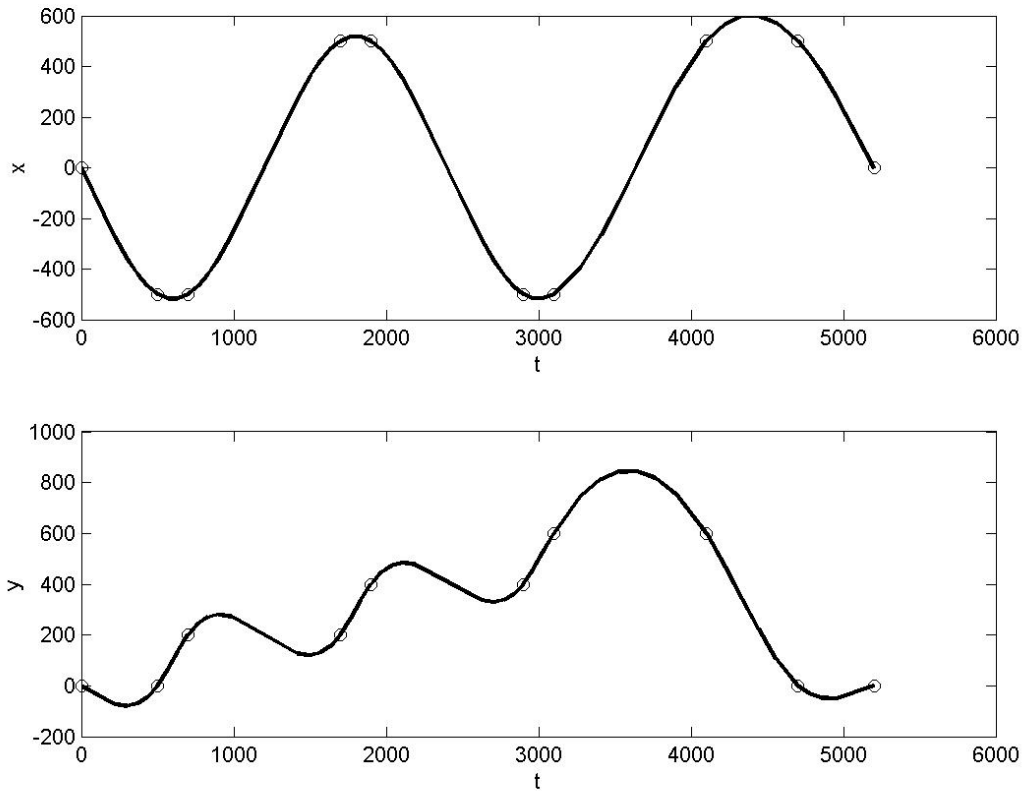


Figura 6-9. Coordenadas frente al parámetro t.

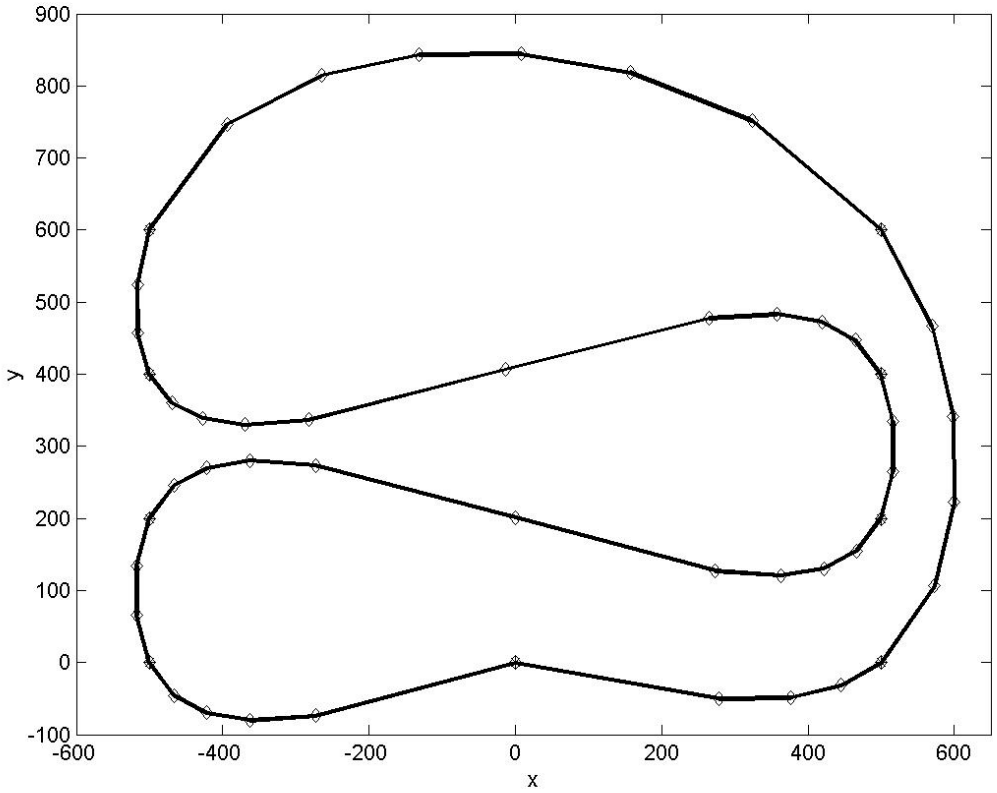


Figura 6-10. Coordenadas x e y desparametrizadas.

En la figura 6-10 se ve cómo la curva queda perfectamente descrita. Se puede obtener una mejor o peor resolución cambiando el ángulo entre el que se divide la pendiente en el intervalo.

6.3.2.4. Curvas para la altura.

El algoritmo para la altura resulta mucho más sencillo que el de la latitud-longitud. Las funciones están desacopladas unas de otras y además, dado que ya se ha discretizado la curva, no será necesario volver a hacer divisiones. Lo único que se necesita es calcular los coeficientes de las funciones cúbicas y los valores que éstas toman en los puntos del intervalo hallados antes.

Los coeficientes se hallan a partir de las condiciones de contorno, según el sistema de ecuaciones que se muestra a continuación.

Posición :

$$At^3_0 + Bt^2_0 + Ct_0 + D = z_0$$

$$At^3_f + Bt^2_f + Ct_f + D = z_f$$

Velocidad :

$$3At^2_0 + 2Bt_0 + C = 0$$

$$3At^2_f + 2Bt_f + C = 0$$

Matricialmente :

$$\begin{bmatrix} t^3_0 & t^2_0 & t_0 & 1 \\ t^3_f & t^2_f & t_f & 1 \\ 3t^2_0 & 2t_f & 1 & 0 \\ 3t^2_f & 2t_f & 1 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} z_0 \\ z_f \\ 0 \\ 0 \end{bmatrix}$$

Resolviendo el sistema para cada intervalo y evaluando las funciones para los puntos intermedios hallados para los splines ya se tiene la curva para las alturas.

6.4 Resultados.

La mejor manera de comprobar los resultados es mostrar un ejemplo. Se creará una trayectoria según los siguientes *waypoints*:

Magnitud \ nº WP	**1**	**2**	**3**	**4**
Latitud (grados)	37.412312	37.412312	37.411385	37.411385
Longitud (grados)	-6.000942	-5.999814	-5.999814	-6.002106
Altura (m)	0.0	5.0	10.0	7.0

Ángulo para las pendientes: 2π/50.

Aplicando el algoritmo:

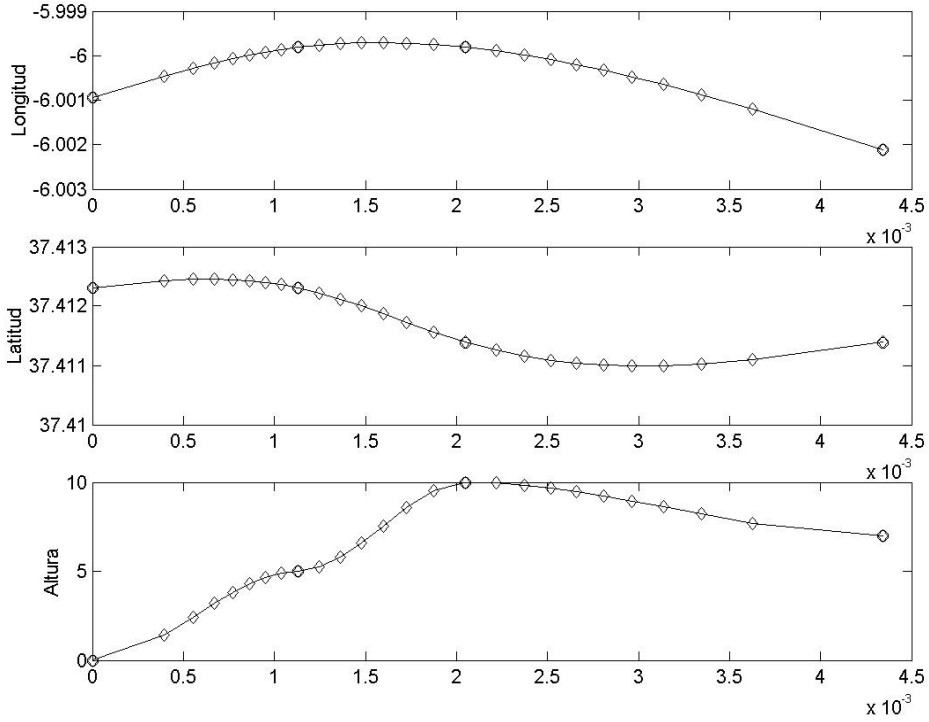


Figura 6-11. Funciones cúbicas de las magnitudes respecto al parámetro t.

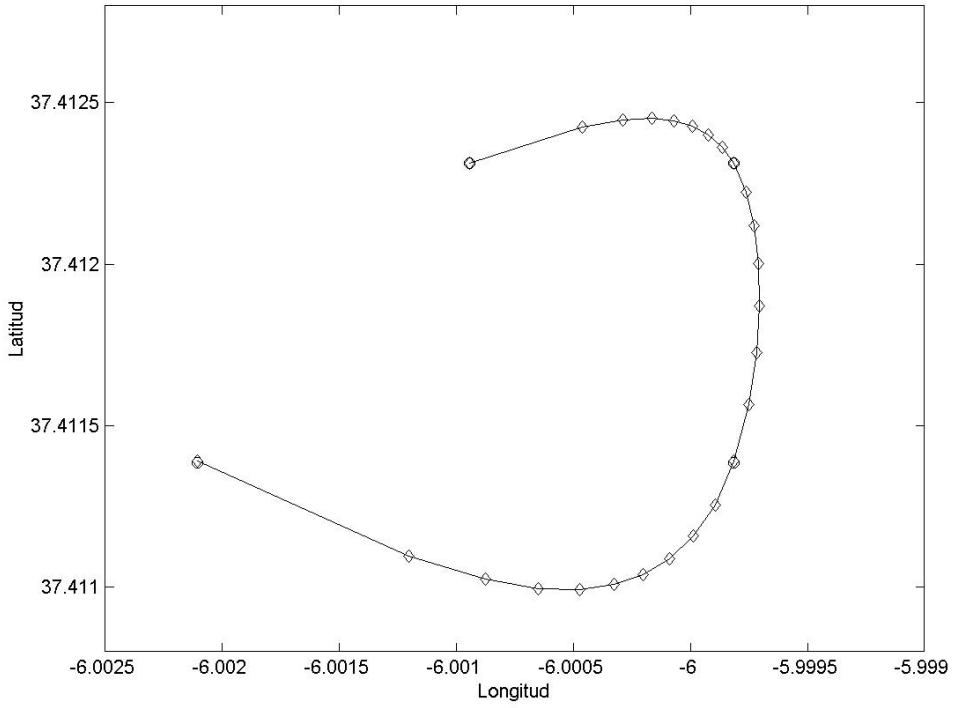


Figura 6-12. Curva en el plano horizontal.

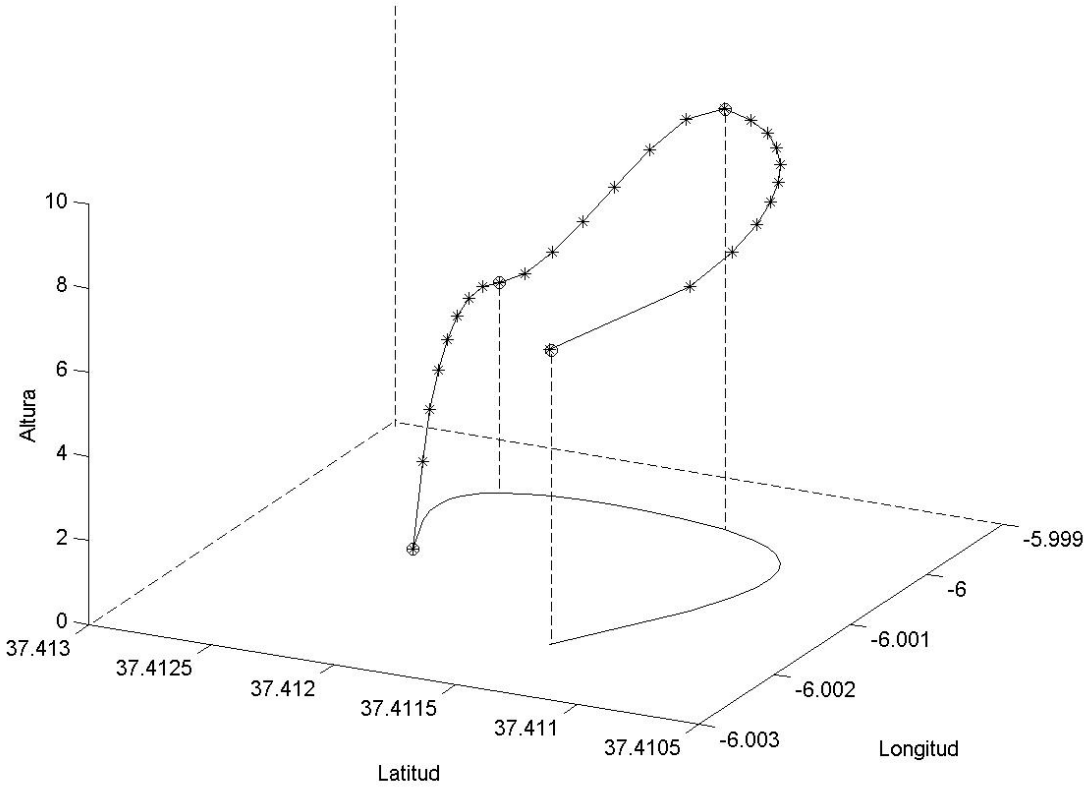


Figura 6-13. Trayectoria completa en el espacio.

6.5. Conclusiones.

6.5.1. Posibles situaciones peligrosas.

Antes de aplicar este generador a una trayectoria conviene conocer una serie de problemas que podrían surgir con determinadas listas de puntos.

6.5.1.1. Puntos consecutivos con idénticas longitud y latitud.

En caso de recibir una lista de puntos con dos puntos consecutivos iguales en el plano, la distancia entre ambos sería nula y con ella el parámetro t. Quedaría imposibilitado el cálculo de la curva.

El algoritmo tiene un mecanismo que detecta estas situaciones, y varía el incremento de t de cero a un valor pequeño positivo. De esta manera se salva el algoritmo, si bien no deja de ser una situación poco recomendable.

6.5.1.2. Problemas propios de los *splines*.

Aunque los *splines* se han venido utilizando desde hace años y se tienen como curvas suaves y fiables, hay que tener un poco de precaución a la hora de indicar los puntos de paso. En particular puede haber problemas si las distancias entre ellos son de órdenes de magnitud diferentes, ya que la curva podría desviarse excesivamente.

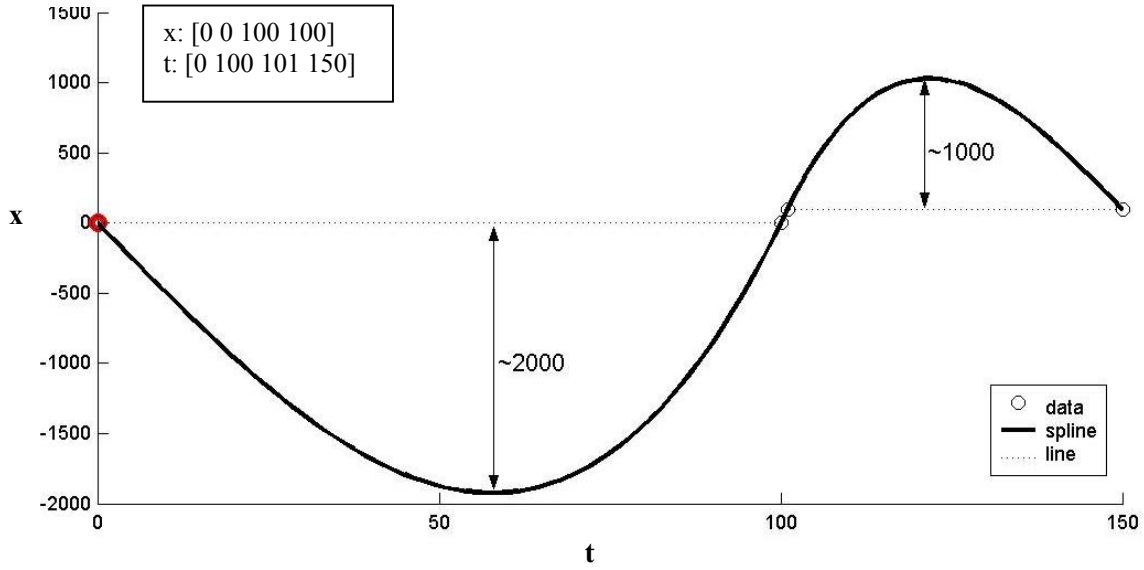


Figura 6-14. Problemas propios de los splines.

En la figura 6-14 se muestra un caso en el que ocurre el mencionado fenómeno en los *splines*. Mientras que los datos de partida para la variable x varían entre 0 y 100, la curva oscila aproximadamente entre -2000 y 1000. Esto supone que la desviación es un orden de magnitud superior a los datos de entrada, lo cual resulta claramente inconveniente.

En el ejemplo de la figura 6-4 esta situación se da debido a que los puntos 2 y 3 están muy cercanos entre sí respecto a los otros dos (entre uno y dos órdenes de magnitud). En el algoritmo del generador de trayectorias este caso se podría dar únicamente si hay grupos de dos o más puntos cercanos entre sí y a su vez alejados del resto, con una diferencia entre las distancias de al menos un orden de magnitud.

6.5.1.3. Límites latitud.

Esta situación ya se comentó en el apartado 6.3.2.3.2, donde se llegó a la conclusión de que no suponía ninguna clase de problema.

6.5.1.4. Límites longitud:

Al llegar a los límites de la latitud existe una no linealidad. Se cambia la coordenada de 180º a -180º o viceversa. El algoritmo no está preparado para tal situación, y trazaría la trayectoria literalmente dando la vuelta al mundo, lo cual es absolutamente indeseable. No obstante se considera poco probable que este problema aparezca debido a la lejanía del meridiano 180º.

6.5.2. Conclusiones generales.

Como se mostró en el ejemplo del apartado anterior, el generador proporciona una trayectoria de funciones cúbicas suave, segura, bien definida y válida para la práctica totalidad de las aplicaciones del *UAV* en las que se requiera un recorrido.

En el caso en el que por circunstancias especiales la lista de *waypoints* presente alguno de los problemas descritos en el apartado 6.5.1, bastará con pasar del algoritmo *spline* al simple, que resulta del todo seguro.

Además resulta sencillo de modificar, ya que utiliza el modelo de programación de *CROMAT*, y se puede ampliar con más algoritmos de un modo sencillo.