UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN.

GUIADO GNSS DE TRACTORES; GENERACIÓN DE CURVAS PARALELAS CON TENDENCIAS A TRAYECTORIAS RECTAS.

SEPTIEMBRE 2025

AUTOR: ARTURO LABAJO MARTÍN

TUTOR: JAIME GÓMEZ GIL

CO-TUTOR: ÁNGEL ALONSO GARCÍA

TITULO: Guiado GNSS de tractores; generación de curvas paralelas con tendencias a trayectorias rectas.

AUTOR: Arturo Labajo Martín

TUTOR: Jaime Gómez Gil, Departamento de Teoría de la Señal, Comunicaciones e Ingeniería Telemática, E.T.S.I. Telecomunicación, Universidad de Valladolid.

DEPARTAMENTO: Departamento de Teoría de la Señal, Comunicaciones e Ingeniería Telemática.

Miembros del Tribunal

PRESIDENTE: Jaime Gómez Gil.

SECRETARIO: Javier Manuel Aguiar Pérez.

VOCAL: Juan Blas Prieto

SUPLENTE1: Alonso Alonso Alonso

SUPLENTE2: Juan Ignacio Arribas

FECHA DE LECTURA:

CALIFICACIÓN:

Resumen

En el ámbito de la agricultura, uno de los fundamentos se trata de planificar la trayectoria a realizar por un vehículo agrario en parcelas con formas irregulares. En muchos casos, los agricultores empiezan a tratar la parcela empezando por una linde que sea recta y trazan el resto de la misma forma. El problema surge cuando ninguna linde es recta, en este caso los agricultores realizan el trabajo mediante trayectorias curvas. En este trabajo se ha de buscar partir de una trayectoria curva y mediante la generación de trayectorias paralelas a la original concluir en una trayectoria que tienda a una recta, definiendo así el camino a seguir de los tractores facilitando el trabajo a los agricultores.

Este trabajo presenta un algoritmo que permite generar trayectorias paralelas a una curva base, controlando la distancia entre ellas y favoreciendo que cada nueva curva se enderece progresivamente. Para ello, se parte de una trayectoria inicial y se van generando nuevas curvas desplazadas mediante vectores normales. A cada paso, se reduce gradualmente el número de puntos de control utilizados para construir la curva, lo que suaviza los detalles y hace que la trayectoria tienda a una recta.

El sistema se ha implementado completamente en Python, utilizando librerías como *scipy* y *matplotlib*, y permite ajustar parámetros como el número de pasadas, la distancia entre trayectorias y la precisión deseada. Los resultados muestran que, tras varias iteraciones, se generan curvas suaves, precisas y cercanas a una línea recta, cumpliendo con los requisitos habituales para el guiado de tractores en agricultura de precisión.

PALABRAS CLAVE: Curvas paralelas, cúspides, enderezado, solape, sin tratar, splines, python, reducción de puntos.

Agradecimientos

Agradecer todo este trabajo a mis familiares y amigos que siempre me han apoyado durante toda mi vida ayudándome en todo lo que lo he necesitado, siendo el pilar fundamental de mi vida.

También agradecer al Dr. D. Jaime Gómez Gil y a Ángel Alonso García por apoyarme y ayudarme todos estos meses de trabajo, guiándome cuando más lo necesitaba.

Muchas gracias a todos.

Índice abreviado

Índice general

Índice de figuras

Capítulo 1:
Introducción y motivación

Capítulo 1: Introducción y motivación

La agricultura de precisión ha experimentado un gran avance gracias a la incorporación de sistemas de guiado automático basados en posicionamiento GNSS (*Global Navigation Satellite System*). Estos sistemas permiten que la maquinaria agrícola siga trayectorias predefinidas con gran exactitud, reduciendo solapes innecesarios entre pasadas y evitando que queden zonas sin tratar.

1.1 Contexto

En este contexto, la generación de trayectorias paralelas a partir de una línea base inicial es una herramienta fundamental:

- Permite cubrir de manera ordenada y completar la superficie agrícola.
- Optimiza el uso de los recursos (semillas, fertilizantes, pesticidas).
- Reduce costes y mejora la eficiencia de la explotación agrícola.

No obstante, la forma de las trayectorias tiene un impacto directo en la calidad del trabajo realizado:

- Los vehículos agrícolas (tractores, sembradoras, pulverizadores, arados,) funcionan de manera mucho más eficiente en tramos rectos, donde el avance es uniforme y las desviaciones son mínimas.
- En tramos curvos, el agricultor (o el sistema de guiado) debe realizar giros continuos que aumentan la fatiga, provocan desgaste y reducen la velocidad de trabajo.
- Los solapes y las zonas sin tratar tienden a incrementarse en las curvas, ya que la anchura efectiva de trabajo del vehículo en activo se ve afectada por el radio de giro.
- En términos energéticos, las trayectorias curvas también suponen un mayor consumo de combustible debido a las correcciones de dirección y al aumento de la resistencia al avance.

Por tanto, en agricultura mecanizada resulta de gran interés que, aunque la línea base inicial pueda contener curvas o irregularidades (por ejemplo, en cabeceras o bordes de parcelas), el proceso de generación de paralelas tienda a producir trayectorias rectilíneas en el interior del terreno cultivable.

Esto garantiza pasadas largas, eficientes y más fáciles de ejecutar tanto para el agricultor como para el sistema de guiado automático.

1.2 Objetivos y problemática

En agricultura, el rendimiento del tractor, y de los vehículos agrícolas en general, es mayor cuando trabajan en trayectorias rectilíneas largas y paralelas, ya que permiten mantener un avance uniforme, reducir consumos y evitar desviaciones, como hemos explicado anteriormente. Sin embargo, la trayectoria base de referencia a partir de la cual se generan las pasadas paralelas no siempre es recta: puede presentar curvas, irregularidades o contornos derivados de la forma de la parcela.

El objetivo inicial de este tipo de algoritmos es, por tanto, transformar progresivamente una trayectoria curva en trayectorias cada vez más rectas, de manera que las últimas paralelas generadas tiendan a una línea recta que optimice el trabajo agrícola.

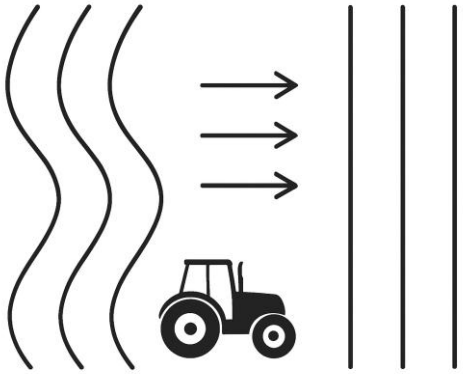

Figura 1.1. Ejemplo esquemático de cómo, a partir de una trayectoria curva inicial, el proceso de generación de paralelas y enderezado permite obtener trayectorias rectas, más adecuadas para el trabajo del tractor en campo. El paso de curvas a rectas no es trivial. Si se desplaza directamente una trayectoria curva mediante normales, aparecen problemas importantes.

- Solape máximo: en el interior de las curvas, las trayectorias se solapan demasiado, lo que implica duplicar tratamientos y malgastar recursos.
- Zona sin tratar máxima: en el exterior de las curvas, las trayectorias se separan demasiado, dejando franjas de terreno sin trabajar.

Los algoritmos basados en polilíneas intentan corregir este efecto aplicando un enderezado iterativo. Sin embargo, presentan limitaciones:

- Necesitan un elevado número de puntos para mantener precisión.
- No garantizan suavidad en las trayectorias, generando cambios bruscos de curvatura.
- El enderezado no siempre converge hacia una recta, por lo que las últimas trayectorias mantienen cierta curvatura residual.

- Trabajan con un número muy elevando de punto haciendo del algoritmo muy poco eficiente y con mucha carga computacional que dificultad su posterior implementación

Este trabajo tiene como objetivo desarrollar un algoritmo de generación de trayectorias paralelas basado en splines cúbicos, con los siguientes objetivos:

- Reducir el número de puntos de control necesarios gracias a la flexibilidad del spline.
- Garantizar suavidad y continuidad en la trayectoria generada.
- Aplicar un proceso de enderezado progresivo que transforme curvas en rectas de forma gradual.
- Mantener bajo control los criterios agronómicos de solape máximo (SM) y zona sin tratar máxima (STM).
- Implementar una reducción adaptativa de puntos según se va aplicando el enderezado

En definitiva, el objetivo es ofrecer un método más eficiente y realista que los algoritmos anteriores, logrando trayectorias rectilíneas finales que optimicen el guiado automático de tractores en agricultura de precisión.

1.3 Metodología general

La estrategia adoptada en este trabajo parte de la idea fundamental de transformar trayectorias curvas en trayectorias progresivamente más rectas mediante el uso de splines cúbicos y un proceso de enderezado iterativo como se refleja en la **Figura 3.2**.

En términos generales, la metodología propuesta se basa en los siguientes pasos:

1. Selección de puntos de control: a partir de la trayectoria base inicial $\boldsymbol{P}$, se extrae un conjunto reducido de puntos $\boldsymbol{Q}$, que describen su forma de manera representativa.

2. Generación de paralelas: se calculan normales en cada punto y se desplazan los puntos de control para construir trayectorias paralelas iniciales.

3. Enderezado local: mediante una función de coste que combina criterios geométricos (longitud, curvatura, alineación) y agronómicos (solape máximo SM, zona sin tratar máxima STM), los puntos de control se ajustan iterativamente buscando el menor valor de la función de coste.

4. Reducción adaptativa del número de puntos de control: Tras enderezar cada pasada se mide la suavidad lograda y, en función de ella, se decide automáticamente cuántos puntos de control usar para el spline de esa misma pasada. La reducción es progresiva y estable: nunca aumenta respecto a la pasada anterior y limita la caída por iteración para evitar saltos bruscos. En zonas con curvas cerradas se mantienen más puntos (decisión conservadora), mientras que en tramos casi rectos se reduce el número para ganar eficiencia y evitar ondulaciones innecesarias.

5. Remallado equiespaciado: tras el enderezado, los puntos de control se redistribuyen para evitar concentraciones locales y mantener homogeneidad a lo largo de la trayectoria.

6. Construcción del spline cúbico: con los puntos corregidos se genera un spline cúbico paramétrico que asegura continuidad en posición, tangentes y curvatura, obteniendo así trayectorias suaves y evaluables con pocos puntos.

7. Iteración progresiva: el proceso se repite para cada nueva paralela, de modo que las trayectorias tienden a ser rectilíneas de manera progresiva.

1.4 Estructura del documento

El presente Trabajo de Fin de Grado se organiza en 5 capítulos, siguiendo una secuencia lógica que va desde la contextualización inicial hasta las conclusiones:

Capítulo 1. Introducción y motivación del trabajo: presenta el contexto de la agricultura de precisión, los problemas asociados a la generación de trayectorias paralelas, los objetivos del trabajo y la metodología general propuesta

Capítulo 2. Antecedentes y trabajos previos: revisa el principal método existente para la generación de trayectorias en agricultura de precisión.

Capítulo 3. Metodología propuesta: escribe en detalle el nuevo algoritmo desarrollado y como se ha implementado prácticamente en Python.

Capítulo 4. Resultados y análisis: muestra los resultados obtenidos en diferentes escenarios de prueba con trayectorias reales.

Capítulo 5. Conclusiones y líneas futuras: resumen de las aportaciones más relevantes del trabajo y planteamiento de posibles líneas de mejora y continuidad.

Capítulo 2:
Antecedentes y trabajos previos

Capítulo 2: Antecedentes y trabajos previos

La agricultura de precisión ha impulsado el desarrollo de diferentes métodos para la generación de trayectorias que guíen el trabajo de máquinas agrícolas. Desde las primeras técnicas basadas en polilíneas simples hasta los algoritmos más recientes, el objetivo ha sido siempre el mismo: cubrir de manera uniforme y eficiente toda la parcela, evitando solapes excesivos y zonas sin tratar.

Este capítulo presenta una revisión de los antecedentes y trabajos previos relacionados con el problema de la generación de trayectorias paralelas.

En primer lugar, se analiza la evolución de los sistemas de guiado en agricultura de precisión y la necesidad de trayectorias rectilíneas como referencia de trabajo. A continuación, se revisan los métodos tradicionales empleados para generar trayectorias paralelas y sus limitaciones. Finalmente, se describe el Trabajo de Fin de Grado de Gabino, que constituye el punto de partida del presente estudio y cuya metodología será ampliada y mejorada mediante el principal uso de splines cúbicos.

2.1 Evolución del guiado en agricultura

El proceso de mecanización agrícola ha ido acompañado de un interés creciente por mejorar la exactitud y la eficiencia del trabajo en el campo. En un primer momento, el guiado de los tractores dependía exclusivamente del operario, que debía mantener la alineación del vehículo de manera visual, tomando como referencia los surcos, límites del terreno o puntos visibles en la parcela. Este enfoque manual resultaba limitado y generaba:

- Solapes innecesarios, debidos a correcciones excesivas de la trayectoria.
- Zonas sin tratar, como consecuencia de desviaciones acumuladas en pasadas largas.
- Pérdida de eficiencia en el uso de recursos, ya que los insumos (fertilizantes, semillas, pesticidas) no se aplicaban de manera uniforme.

Con la introducción de los sistemas GNSS y más tarde del guiado asistido y automático, se logró un avance significativo: ahora el tractor puede seguir con precisión una trayectoria digital predefinida. Sin embargo, este avance trasladó el problema hacia otro ámbito: ya no se trata solo de que el tractor siga una línea, sino de cómo generar un conjunto de trayectorias de referencia de alta calidad que permitan cubrir de forma homogénea toda la superficie cultivada.

En este contexto surge la importancia de las trayectorias paralelas:

- Permiten dividir la parcela en pasadas equidistantes, adaptadas a la anchura de trabajo.
- Garantizan que el terreno quede totalmente cubierto, minimizando solapes (SM) y zonas sin tratar (STM).

- Favorecen que el tractor realice la mayor parte de su labor en tramos rectos y largos, que son más eficientes desde el punto de vista mecánico, energético y agronómico.

Por tanto, la evolución del guiado en agricultura de precisión ha puesto de manifiesto la necesidad de diseñar algoritmos que no solo generen paralelas, sino que además tiendan de manera progresiva hacia trayectorias rectilíneas en el interior de la parcela, lo que constituye un elemento clave en la mejora de la eficiencia agrícola.

2.2 Métodos tradicionales de generación de curvas paralelas

La necesidad de cubrir una parcela agrícola mediante pasadas equidistantes ha motivado el desarrollo de diferentes procedimientos geométricos para generar trayectorias paralelas a una curva base. Entre los métodos más empleados en la bibliografía destacan los siguientes:

2.2.1 Desplazamiento mediante normales

Consiste en calcular en cada punto de la curva base $\boldsymbol{p}(s)$ el vector normal unitario asociado a la tangente de la trayectoria, y desplazar el punto una distancia fija d en esa dirección $\boldsymbol{p}_d(s) = \boldsymbol{p}(s) \pm d \cdot \vec{n}(s)$ donde $\vec{n}(s)$ es el vector normal $\vec{n}(s) = \left(-t_y(s), t_x(s)\right)$ y donde $\vec{t}(s)$ es el vector tangente unitario $\vec{t}(s) = \frac{\boldsymbol{p}'(s)}{\|\boldsymbol{p}'(s)\|}$ [1].

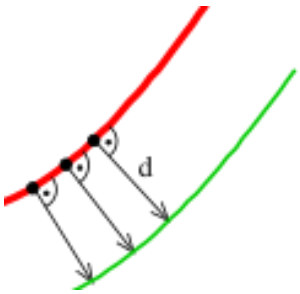

Figura 2.1. Generación de una paralela a través del método de desplazamiento mediante normales [1].

2.2.2 Envolvente de circunferencias

Una curva paralela a una trayectoria plana $\boldsymbol{p}(t) = (x(t), y(t))$ a una distancia fija d puede definirse como la envolvente de una familia de circunferencia congruentes (todas de radio d) cuyos centros recorren la curva original [1]. Formalmente si para cada parámetro β situamos un círculo de radio d centrado en $(x(\beta), y(\beta))$, la curva paralela es la envolvente de esa familia. Para una familia genérica $f(x, y, \beta) = 0$, la envolvente satisface el sistema necesario

$$\begin{cases} f(x, y, \beta) = 0 \\ \frac{\partial}{\partial \beta} \cdot f(x, y, \beta) = 0 \end{cases} \qquad (1)$$

1. Familia de círculos
   Definimos:

$$f(x, y, \beta) = (x - x(\beta))^2 + (y - y(\beta))^2 - d^2 = 0. \quad (2)$$

Esta es la ecuación implícita del círculo de radio *d* centrada en $\boldsymbol{p}(\beta)$. La envolvente de esta familia cumple $f = 0$ y $\frac{\partial}{\partial \beta} = 0$

2. Condición tangencial ($\frac{\partial}{\partial \beta} = 0$)

   Si desarrollamos esta expresión

$$\frac{\partial f}{\partial \beta} = -2(x - x(\beta))x'(\beta) - 2(y - y(\beta))y'(\beta) = 0. \quad (3)$$

   Entonces obtenemos que:

$$(x - x(\beta), y - y(\beta)) \cdot (x'(\beta), y'(\beta)) = 0. \quad (4)$$

   Dando entonces que, el radio del círculo hacia el punto de contacto es ortogonal a la tangente de la trayectoria β. Por tanto, ese radio apunta en dirección normal a la curva

3. Ecuación explicita de la envolvente

   Si $\vec{t}(\beta) = (x'(\beta), y'(\beta))$ y $\| \vec{t} \| = x'2(\beta) + y'2(\beta)$, una normal unitaria es $\vec{n}(\beta) = \frac{1}{\|\vec{t}\|}$ $(-y'(\beta), x'(\beta))$.

   Combinando $f = 0$ con la ortogonalidad, el punto de la envolvente queda tal que:

$$pd(\beta) = p(\beta) \pm dn(\beta) \quad (5)$$

   Esta es la misma expresión que se obtiene por el método del vector normal (apartado 2.2.1), mostrando la equivalencia entre ambas definiciones.

   En la Figura 2.1 se aprecia se aprecia la envolvente de una familia de circunferencia y como tras

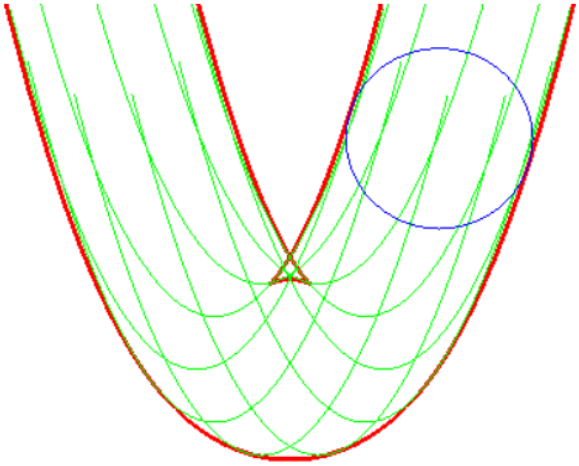

Figura 2.2. Generación de una paralela a traves del método de envolvente de circunferencias [2]

2.3 Algoritmo del TFG de Gabino

Previo a nuestro proyecto, Gabino Martínez García desarrollo su Trabajo Final de Grado el cual partía de una trayectoria medida y cumplía los siguientes puntos

1. Paralelización por vector normal para generar las paralelas consecutivas. Como resultado de esto se obtenía que, cuando se cerraban las curvas según se iban creando las paralelas se formaban cúspides que deformaban la trayectoria para resolver esto desarrollo el apartado siguiente.
2. Enderezado iterativo para ir reduciendo la curvatura hasta tender a una recta. El enderezado se formaliza como un problema de optimización con restricciones agronómicas: Solape Máximo (SM) y Sin Tratar Máximo (STM), que acotan cuánto puede desplazarse cada punto respecto a su posición original.

Para el apartado 1, Gabino aplico el método visto en el apartado 2.2.1 el cual trata de desplazar el vector normal de cada punto para conseguir la paralela como resultado.

Para el apartado 2, se desarrolló un algoritmo de enderezado plantado por el Dr. Cesar Palencia de Lara y que el propio Gabino llevó a cabo [3].

$$T = \sum M_i + R \tag{6}$$

2.4 Limitaciones detectadas en el método anterior

El enderezado se formula como la minimización de la ecuación (6) y se resuelve probando, para cada punto que forman la trayectoria, varios desplazamientos espaciados entre [STM, SM] a lo largo de la bisectriz del ángulo local eligiendo el que más reduzca T. Este procedimiento se repite mediante múltiples iteraciones por lo que el coste crece con el número de puntos el número de iteraciones), lo que el propio Gabino describió como “solución por fuerza bruta”

Como objetivo general vamos a buscar en nuestro proyecto mantener la intuición agronómica y la robustez geométrica, buscando reducir el coste drásticamente. Para ello buscaremos:

- Reducir el coste manteniendo SM y STM. Para ello buscaremos reducir el número de puntos con los que trabajaremos a través de splines cúbicos.
- Cambiar "fuerza bruta" por enderezado local eficiente. En vez de probar *K* candidatos por punto, plantearemos actualizaciones locales que reduzcan la longitud local, la energía de curvatura y la desalineación de rumbo, manteniendo la penalización asintótica M(t) para SM/STM.
- Evitar la acumulación de error entre pasadas, para ello validaremos cada nueva paralela contra la curva original desplazada y estabilizamos la convergencia hacia la recta

Capítulo 3:
Metodología

Capítulo 3: Metodología

Este capítulo describe, desde un punto de vista teórico, la estrategia que seguimos para transformar trayectorias agrícolas inicialmente curvas en pasadas progresivamente más rectas y suaves, respetando las restricciones agronómicas de solape máximo (SM) y zona sin tratar máxima (STM). El eje matemático del método es los splines, y en particular los splines cúbicos, por su continuidad en posición, dirección y curvatura (C^2), cualidades esenciales para un guiado estable.

3.1 Introducción a los splines y justificación de los splines cúbicos

Un spline es una función a trozos que interpola un conjunto de nodos (puntos), garantizando la suavidad en las uniones. Para trayectorias en el plano trabajamos con un spline paramétrico:

$$\boldsymbol{S}(t) = \big(x(t), y(t)\big) \qquad t \in [t_0, t_N] \tag{7}$$

Definido sobre un conjunto de nudos $t_0 < \cdots < t_N$. En cada intervalo $[t_k, t_{k+1}]$ la curva es un polinomio de grado fijo (el orden del spline) y en los nudos se imponen condiciones de continuidad de derivadas.

3.1.1 Splines cúbicos y su construcción

En un spline cúbico [4] el grado por tramo es de tres. Sea $h_k = t_{k+1} - t_k$ y nodos de datos $\{(t_k, X_k)\}$ (para x) y $\{(t_k, Y_k)\}$ (para y). En cada tramo:

$$x(t) = a_k + b_k(t - t_k) + c_k(t - t_k)^2 + d_k(t - t_k)^3 \qquad t \in [t_k, t_{k+1}] \tag{8}$$

y análogamente para $y(t)$. Los coeficientes del polinomio de grado tres que tenemos se fijan exigiendo:

1. Interpolación: $x(t_k) = X_k, x(t_{k+1}) = X_{k+1}$ (Igualmente para y)
2. Continuidad de derivadas en cada nudo interior t_k:

$$x, x', x'' \text{ continuas en } t_k \text{ (suavidad } C^2)$$

Estas condiciones generan un sistema tridiagonal para las segundas derivadas en los nudos $m_k = x''(t_k)$ (y análogo para y). Resuelto m_k, los coeficientes a_k, b_k, c_k, d_k salen de fórmulas cerradas. El resultado es una curva suave en posición, tangente y curvatura.

3.1.2 Parametrización t

La calidad de un spline paramétrico depende de cómo asignamos t_k a los puntos de control. Tenemos tres opciones comunes

- Uniforme: $t_k = k$. Simple, pero puede concentrar curvatura donde los puntos estén desiguales.
- Longitud de cuerda: $t_k = \sum_{j \le k} \|\boldsymbol{q}_j - \boldsymbol{q}_{j-1}\|$. Adecúa el parámetro a la geometría
- Arco (normalizado): es igual que el método anterior pero escalado a [0, 1].

Usamos arco normalizado porque reparte el parámetro proporcionalmente a la longitud geométrica: evita acumulaciones en nodos eficaces y estabiliza la curvatura

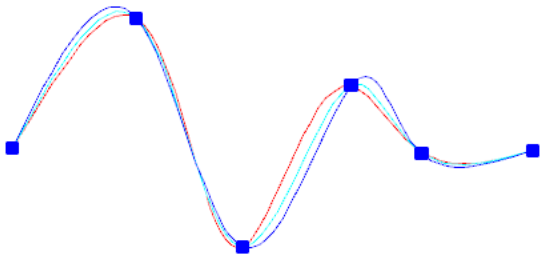

Figura 3.1. Construcción de un spline cúbico a partir de un mismo conjunto de puntos (cuadrados azules). Las curvas de colores muestran soluciones suaves obtenidas con distintas parametrizaciones/condiciones de contorno. Se aprecia la suavidad de la trayectoria y la influencia local de cada punto de control. [5]

3.1.3 Propiedades de los splines cúbicos

1. Suavidad. Continuidad de S, S', S''. En guiado esto significa movimientos más estables al tener curvatura sin sobresaltos.

2. Localidad. Cambiar un nodo solo afecta a tramos vecinos, lo que facilita ediciones como la reducción de puntos, y produce robustez.

3. Óptimo de "rigidez". El spline cúbico minimiza la energía de flexión $\int \|S''(t)\|^2 \, dt$ entre todas las curvas que interpolan el nodo, esto se traduce que es la curva más suave la que pasa por los puntos evitando ondulaciones de más

4. Exactitud en los nodos. La curva que se produce pasa exactamente por los puntos de control.

5. Curvatura bien definida. Para $S(t) = (x(t), y(t))$, la curvatura continua es [4]:

$$k(t) = \frac{|x'(t)y''(t) - y'(t)x''(t)|}{(x'(t)^2 + y'(t)^2)^{\frac{3}{2}}} \tag{9}$$

3.2 Herramientas software para trabajar con splines

En este apartado se van a exponen las herramientas que utilizamos para construir, evaluar y analizar splines desde un punto de vista teórico.

3.2.1 Entorno Python

NumPy: proporciona el cálculo vectorial necesario

- Distancias y longitudes de arco $s_j = \sum \|\boldsymbol{p}_j - \boldsymbol{p}_{j-1}\|$
- Tangentes $\vec{t}$ y normales $\vec{n} = (-\vec{t}_y, \vec{t}_x)$

SciPy: para aplicar la interpolación y splines

- *CubicSplines*: construye, para un conjunto de nodos t_k, polinomios de grado 3 en cada tramo con continuidad C^2 en los nudos. Matemáticamente resuelve un sistema tridiagonal para las segundas derivadas $S''(t_k)$
- *UnivariateSpline*: realiza un spline suavizado, pero no obliga a pasar exactamente por los datos, busca minimizar la función de coste (10)

$$\sum_k (y_k - s(t_k))^2 + \lambda \int (S''(t))^2 \, dt \qquad (10)$$

  Donde λ controla el compromiso ajuste-suavidad.

- *PchipInterpolator*: interpolante por tramos que evita sobrecorreciones "overshoot" y respeta la monotonía, pero sacrifica la suavidad.

Matplotlib: Soporte de figura tales como curvas o gráficos de métricas. Es esencial para documentar visualmente el verdadero efecto del algoritmo una vez aplicado y ver su correcto funcionamiento.

3.2.2 Herramientas equivalentes en MATLAB

- *Spline*: interpolación cúbica con continuidad C^2 (similar a CubicSpline).
- *Csape* (complete spline): similar a *spline* pero con la posibilidad de elección explicita de condiciones de contorno
- *Spapi*: ajuste de B-splines por mínimos cuadrado

Todas estas funciones trabajan internamente con bases B-splines y secuencias de nudos; la teoría de continuidad y compromisos ajuste/suavidad son los mismos que en SciPy.

3.3 Visión general del algoritmo desarrollado

El objetivo que buscamos es convertir una trayectoria medida P en una familia de pasadas paralelas suaves C^2 y regulares, respetando el solape máximo (SM) y zona sin

tratar máxima (STM). Trabajamos con una curva paramétrica $\boldsymbol{s}(t) = (x(t), y(t))$ descritra mediante splines cúbicos.

Notacion que usaremos:

- $\boldsymbol{P} = \{\boldsymbol{p}_i\}_{i=0}^{N-1}$: curva original
- $\boldsymbol{Q} = \{\boldsymbol{q}_i\}_{i=0}^{N-1}$: puntos de control equispaciados por arco
- $\boldsymbol{Q}^{desp} = \{\boldsymbol{q}_i^{desp}\}_{i=0}^{N-1}$: puntos tras el desplazamiento normal
- $\boldsymbol{Q}^{ender} = \{\boldsymbol{q}_i^{ender}\}_{i=0}^{N-1}$: puntos tras el enderezado
- $\boldsymbol{Q}^{red} = \{\boldsymbol{q}_i^{red}\}_{i=0}^{N-1}$: puntos tras aplicar la reducción de puntos adaptativas
- $\boldsymbol{R} = \{\boldsymbol{r}_j\}_{j=0}^{N-1}$: malla remallada
- $\vec{t}$ y $\vec{n}$: tangente y normal unitarias

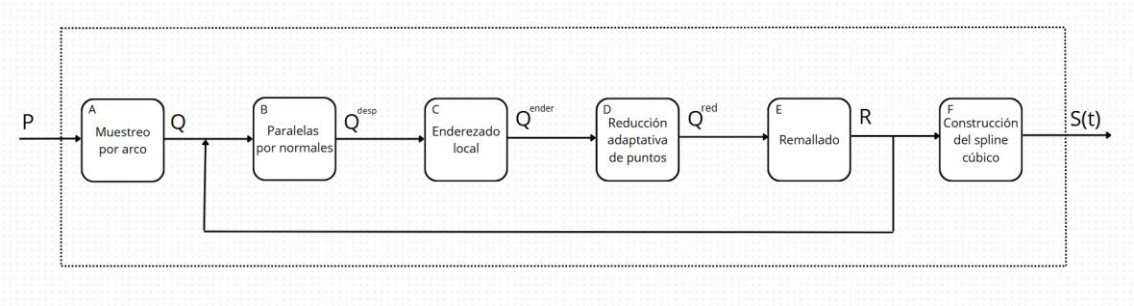

Figura 3.2. Vista de "caja negra". El sistema recibe la polilínea de entrada $\boldsymbol{P}$ *(puntos muestreados) y entrega un spline cúbico* $\boldsymbol{S}(t)$*. Internamente, los bloques A-F realizan muestreo por arco, paralelas por normales, enderezado con SM/STM, reducción adaptativa de puntos, remallado y construcción del spline; la realimentación implementa el modo encadenado para generar las pasadas sucesivas donde los puntos de entrada al Bloque B pueden venir de la trayectoria original o de la paralela anterior.*

El sistema recibe como entrada la polilínea $\boldsymbol{P}$ (puntos originales) y devuelve como salida un spline cúbico $\boldsymbol{S}(\boldsymbol{t})$ con suavidad $\boldsymbol{C^2}$, apto para el guiado. Internamente, el proceso recorre los bloques A–F de forma secuencial y, mediante la realimentación que se observa en la **Figura 3.2** se produce el encadenado entre todas las paralelas.

- Bloque A: Muestreo por arco ($\boldsymbol{P} \rightarrow \boldsymbol{Q}$)
  Compacta la trayectoria original seleccionando puntos de control equiespaciados por longitud de arco. Mantiene los extremos como anclajes y deja una malla uniforme para trabajar. Con esto conseguimos tener una base de puntos **Q** con la que trabajar mucho mejor que la cantidad de puntos originales

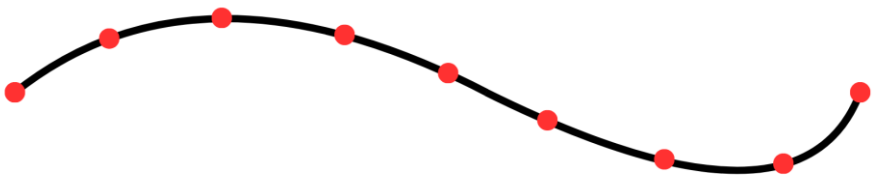

Figura 3.3. Trayectoria original $\boldsymbol{P}$ *(negro) y puntos de control* $\boldsymbol{q}$ *equispaciados por longitud de arco (rojo). Los extremos se mantiene anclados y los puntos de control resumen la forma y será la malla base para las siguientes etapa.*

- Bloque B: Paralelas por normales ($\boldsymbol{Q} \rightarrow \boldsymbol{Q^{desp}}$)
  Calcula tangentes y normales en cada punto y desplaza la malla una distancia fija (separación entre pasadas) para construir la paralela inicial. Esto se realiza mediante el desplazamiento por normales visto en el apartado 2.2.1

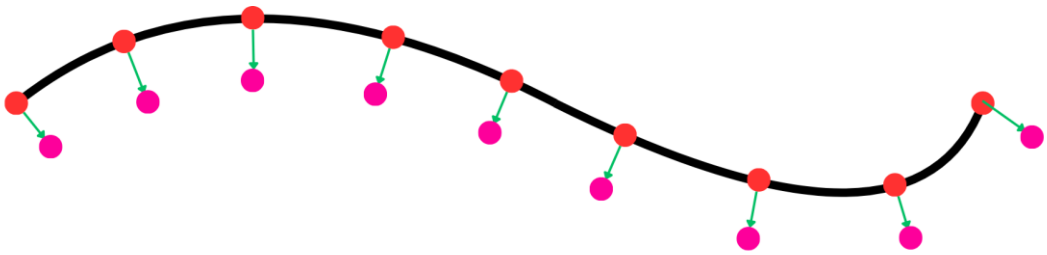

Figura 3.4. Puntos de control $\boldsymbol{q}$ (rojo) desplazados sobre su normal unitaria (flechas verdes) una distancia fija d y como resultado temenos los puntos desplazados (rosa).

- Bloque C: Enderezado local ($\boldsymbol{Q^{desp}} \rightarrow \boldsymbol{Q^{ender}}$)
  Ajusta cada punto solo sobre su normal minimizando una función de coste que combina longitud, suavidad/curvatura y restricciones agronómicas (SM/STM). Produce una pasada más recta y regular. Este bloque es el que se encarga de ir realizando el enderezado para obtener una trayectoria con tendencia a recta.

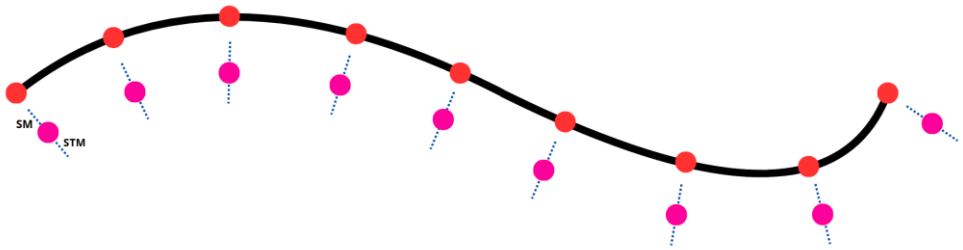

Figura 3.5. Cada punto de control $\boldsymbol{q}_i$ (rojo) solo puede moverse sobre su normal (trazos azules), acotado por los parámetros [SM, STM]

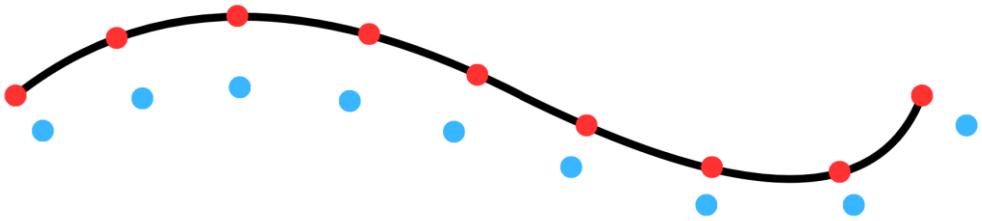

Figura 3.6. Posiciones óptimas $\boldsymbol{q}_i^{ender}$ (azul) obtenidas tras la minimización sobre la normal de cada $\boldsymbol{q}_i$ (rojo) obteniendo así una pasada más recta.

- Bloque D: Reducción adaptativa de puntos ($\boldsymbol{Q^{ender}} \rightarrow \boldsymbol{Q^{red}}$)

Mide la suavidad alcanzada y decide automáticamente cuántos puntos de control hacen falta. En zonas suaves reduce puntos (eficiencia); en curvas cerradas los mantiene (seguridad). Con este bloque conseguimos reducir la carga computacional reduciendo la cantidad de puntos con los que trabaja el sistema.

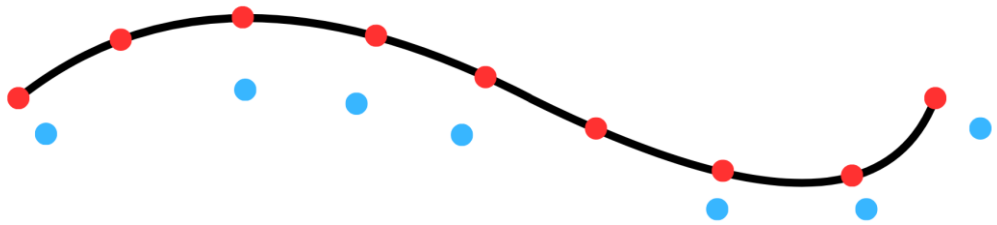

Figura 3.7. Reducción adaptativa de puntos de $\boldsymbol{Q}^{ender}$ tras medir su suavidad y decidir cual es el tamaño óptimo de la nueva malla obteniendo un número menor de puntos de control para realizar la pasada.

- Bloque E: Remallado equiespaciado ($\boldsymbol{Q^{ender} \rightarrow R}$)
  Este bloque redistribuye el número de puntos de control en puntos por arco, con extremos fijos, para evitar amontonamientos y dejar una malla homogénea que estabiliza el spline. Realiza un trabajo similar al Bloque A, pero mantiene el número de puntos solo para reorganizarlos de manera equidistante a la trayectoria.

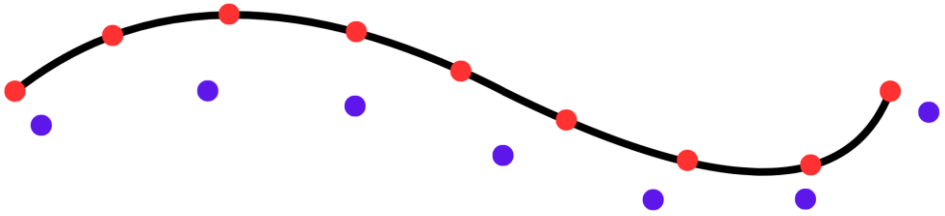

Figura 3.8. Desde la pasada enderezada $\boldsymbol{Q}^{ender}$ se reparametriza por longitud de arco y se distribuyen exactamente n_{ctrl} puntos uniformes para obtener la malla $\boldsymbol{R}$ (Morado), manteniendo los extremos fijos. El remallado elimina amontonamientos y deja una malla equidistante preparada para la construcción del spline.

- Bloque F: Construcción del spline cúbico ($\boldsymbol{R \rightarrow S(t)}$)
  Interpola la malla remallada con un spline cúbico paramétrico, garantizando continuidad en posición, rumbo y curvatura ($\boldsymbol{C^2}$) como hemos visto en el apartado 3.1. Este es el resultado final que usa el guiado.

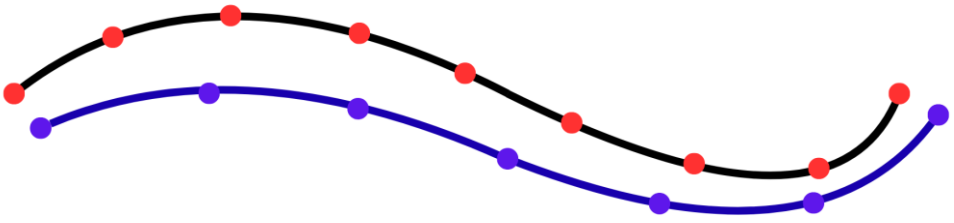

Figura 3.9. A partir de la malla $\boldsymbol{R}$ (puntos morados) se interpola el spline cúbico $\boldsymbol{S}(t)$ (curva azul) con suavidad C^2. Se muestra frente a la trayectoria base $\boldsymbol{P}$ (negro) formada por los puntos de control previos (rojo)

En las secciones siguientes detallaremos la matemática y la propia implementación del algoritmo en Python para su posterior visualización.

3.4 Bloque A – Muestreo por arco (P → Q)

3.4.1 Descripción

Este bloque parte de una trayectoria discreta $\boldsymbol{P} = \{\boldsymbol{p}_0, \dots, \boldsymbol{p}_{N-1}\} \subset \mathbb{R}^2$ formada con miles de puntos desigualmente organizados y constituye una malla base $\boldsymbol{Q} = \{\boldsymbol{q}_0, \dots, \boldsymbol{q}_{N-1}\}$ con un inferior número de puntos equiespaciados por longitud de arco, conservando los extremos. En este paso fijamos un número objetivo de puntos M, reducimos la cantidad de nodos. El proceso no cambia la forma de la curva y mantiene anclados los extremos, dejando una base regular y estable para el posterior uso por los siguientes bloques del algoritmo.

La **Figura 3.10** muestra el diagrama de flujo que sigue este bloque para conseguir su objetivo. Primero el bloque recibe los puntos de la curva y el número de nodos que queremos en la malla base. Si la entrada contiene menos de dos puntos o una longitud nula se devuelve los puntos recibidos sin cambios. En caso contrario se realiza el cálculo de la longitud de arco y se normaliza. Sobre este eje normalizado se construye una malla uniforme de n posiciones y se interpolan, generando una nueva malla de punto $\boldsymbol{Q} = \{\boldsymbol{q}_i\}$ equiespaciados.

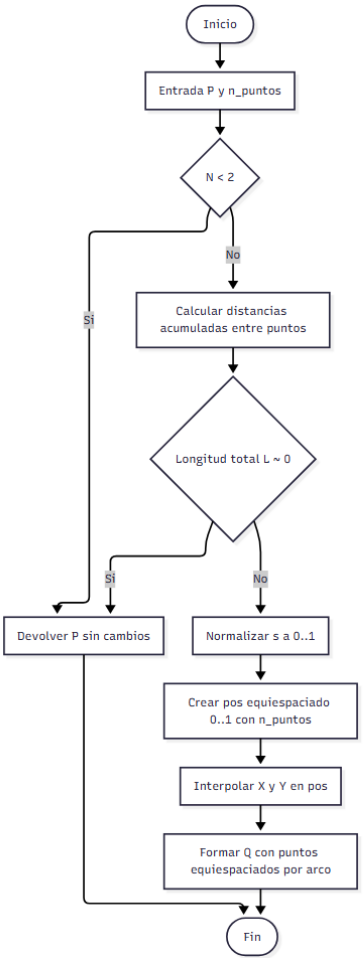

Figura 3.10. Diagrama de flujo del Bloque A. A partir de la polilinea original $\boldsymbol{P}$ *y un tamaño deseado* n_{ctr} *se calcula el arco acumulado, se normaliza a [0,1] y se interpola cada coordenada para obtener* $\boldsymbol{Q}$*, una malla de puntos equiespaciados por longitud de arco.*

3.4.2 Matemática asociada

En este bloque se plantea un muestreo equiespaciado por longitud de arco [6]. La idea es asignar un parámetro de arco acumulado a la trayectoria original y reinterpolar la curva para obtener un conjunto de puntos igualmente equiespaciados en términos de distancia recorrida. Con este procedimiento se obtiene una malla homogénea $\boldsymbol{Q}$, que estabiliza el spline cúbico del bloque final y permite que todos los cálculos intermedios se realicen sobre una base más uniforme.

Dada la trayectoria original discreta $\boldsymbol{P} = \{\boldsymbol{p}_0, \dots, \boldsymbol{p}_{N-1}\}$, $\boldsymbol{p}_i = (x_i, y_i) \subset \mathbb{R}^2$ donde cada $\boldsymbol{p}_i$ es un punto de posición en el plano.

1. Longitud acumulada
   Para cada tramo entre dos puntos consecutivos:

$$\Delta \boldsymbol{p}_i = \boldsymbol{p}_i - \boldsymbol{p}_{i-1}, \quad \|\Delta \boldsymbol{p}_i\| = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{11}$$

   Se define:

$$s_0 = 0, \quad s_i = \sum_{k=1}^{i} \|\Delta \boldsymbol{p}_k\|, \quad i = 1, \dots, N-1 \tag{12}$$

Donde s_i es un escalar que representa la longitud de arco acumulada hasta el punto i. La longitud total de la trayectoria es $L = s_{N-1}$

2. Parámetro normalizado
   Para eliminar unidades y facilitar la interpolación vamos a normalizar:
   $$u_i = \frac{s_i}{L}, \quad u_i \subset [0,1] \tag{13}$$

   Aquí u_i es un escalar adimensional

3. Malla uniforme de control
   Se definen *n* valores equiespaciados en el intervalo normalizado [0,1]:
   $$\hat{u}_j = \frac{j}{n-1} \qquad j = 0,1, \dots, n \tag{14}$$

   Que representan fracciones iguales de longitud con lo que se realizara la posterior interpolación de los nuevos puntos.

4. Interpolación por arco
   Finalmente, los nuevos puntos se calculan como:
   $$\boldsymbol{q}_j = \left(\hat{x}(\hat{u}_j), \hat{y}(\hat{u}_j)\right), \qquad j = 0,1, \dots, n \tag{15}$$

   Donde $\hat{x}(\hat{u}_j)$ y $\hat{y}(\hat{u}_j)$ son interpolaciones lineales de los datos originales. Cada $\boldsymbol{q}_j$ es un vector de posición de la nueva malla $\boldsymbol{Q}$ y están distribuidos de forma equidistante en nuestra trayectoria

3.4.3 Explicación del código

La **Figura 3.11** muestra la implementación de la función que transforma la polilínea $\boldsymbol{P} = \{\boldsymbol{p}_i\}$ en una malla $\boldsymbol{Q} = \{\boldsymbol{q}_i\}$ equiespaciada por longitud de arco.

El procedimiento comienza calculando la longitud acumulada de la curva original (11) (12). Esta longitud acumulada se normaliza para obtener un parámetro adimensional $t \subset [0,1]$ (13). A continuación, se aplica una interpolación lineal sobre las coordenadas *x* e *y* en función de dicho parámetro t (15), en base a una malla uniforme de tamaño n (14), generando un nuevo conjunto de puntos distribuidos de manera uniforme

El código utiliza librerías *NumPy* y *SciPy* para optimizar estas operaciones. En particular, la función *np.sqrt* [7]se emplea para calcular las distancias euclídeas entre pares consecutivos de puntos, mientras que *np.interp* [8] permite reconstruir las coordenadas de los nuevos puntos a partir de la parametrización.

De esta forma el resultado es una curva remallada que mantiene los extremos de la trayectoria original, pero con una densidad de puntos uniforme.

```python
def muestrear_equiespaciado_arco(curva, n_puntos):
    curva = np.array(curva, float)
    if len(curva) < 2: return curva
    dist = np.zeros(len(curva))
    for i in range(1, len(curva)):
        dist[i] = dist[i-1] + np.linalg.norm(curva[i] - curva[i-1])
    if dist[-1] == 0: return curva
    s = dist/dist[-1]
    pos = np.linspace(0, 1, n_puntos)
    x_eq = np.interp(pos, s, curva[:,0]); y_eq = np.interp(pos, s, curva[:,1])
    return np.column_stack([x_eq, y_eq])
```

Figura 3.11. Implementación de la función muestrear_equispaciado_arco. Esta función corresponde al Bloque A del algoritmo.

3.4.4 Resultados del bloque

La **Figura 3.12** muestra el resultado tras la aplicación del bloque A, primero partimos de una trayectoria discreta de puntos ***P*** (negro) que forman la trayectoria base, tras realizar el muestreo por arco obtenemos un conjunto de puntos reducidos ***Q*** (rojo) equiespaciados, con los que trabajaremos en los posteriores bloques consiguiendo una reducción de puntos para trabajar posteriormente reduciendo la carga computacional y que gracias a la posterior aplicaciones de los splines obtendremos una trayectoria suave con menos puntos.

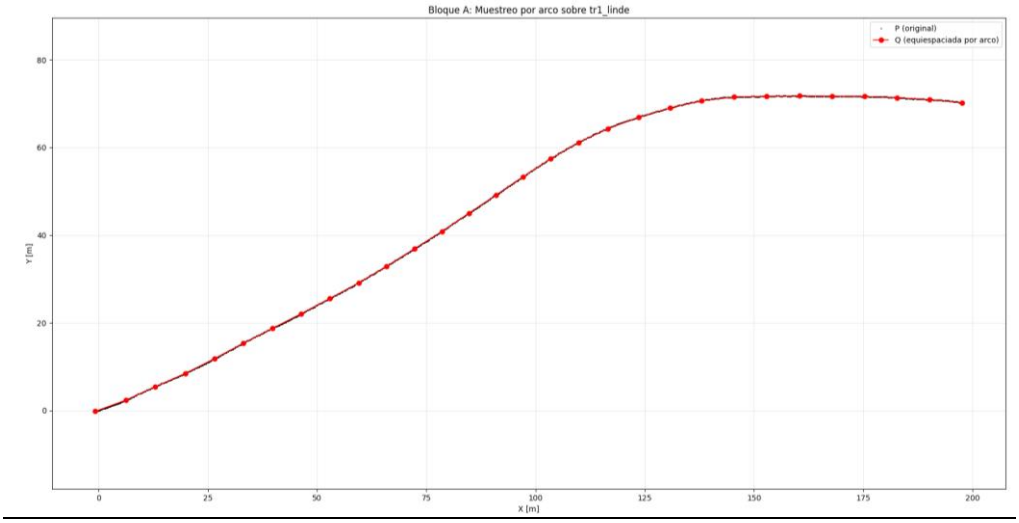

*Figura 3.12. Resultado de la implementación del Bloque A. La curva original **P** (negro) presenta una trayectoria formada por puntos no uniformes. Tras aplicar la función muestrear_equispaciado_arco, se obtiene la curva **Q** (rojo), cuyos puntos estan equispaciados por longitud de arco, proporcionando una malla homogénea y formada por puntos equidistantes unos de otros.*

3.5 Bloque B – Paralelas por normales ($\boldsymbol{Q} \rightarrow \boldsymbol{Q^{desp}}$)

3.5.1 Descripción del algoritmo

A partir de la malla base $\boldsymbol{Q}$ del Bloque A, este bloque genera la paralela $\boldsymbol{Q^{desp}}$. Para ello, cada punto $\boldsymbol{q_i}$ se desplaza una distancia fija *d* en la dirección de la normal unitaria de la curva. El resultado es un nuevo conjunto de puntos $\boldsymbol{Q^{desp}} = \{\boldsymbol{q_i^{desp}}\}$, que representa la curva paralela.

En la **Figura 3.13** vemos el diagrama de flujo que sigue este bloque donde recibimos una serie de puntos equiespaciados que pueden venir de la trayectoria original si es la primera paralela o pueden venir de la paralela anterior como vemos en la **Figura 3.2**. Tras recibir los puntos y comprobando que se han recibido más de dos puntos se calculan las tangentes normalizadas y las normales unitarias rotando la tangente 90º. Tras esto se calcula el desplazamiento y con esto ya podemos desplazar cada punto $\boldsymbol{q_i}$ obteniendo en su conjunto una malla de puntos desplazados por su normal $\boldsymbol{Q^{desp}}$ y con este conjunto ya obtendríamos la paralela, pero de momento solo nos interesa quedarnos con los puntos para seguir trabajando con ellos posteriormente.

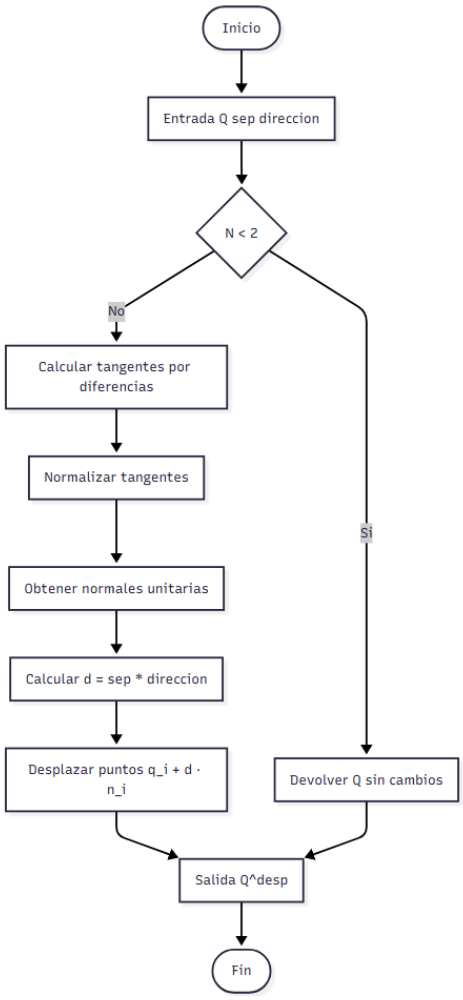

Figura 3.13. Diagrama de flujo del Bloque B. A partir de la malla $\boldsymbol{Q}$, la separación y el signo de dirección, se calculan tangente y normales unitarias y se desplaza cada nodo una distancia para obtener $\boldsymbol{Q^{desp}}$.

3.5.2 Matemática asociada

Partimos de la secuencia de puntos muestreados en el bloque A:

$$\boldsymbol{Q} = \{\boldsymbol{q}_0, \boldsymbol{q}_1, \dots, \boldsymbol{q}_N\}, \quad \boldsymbol{q}_i = (x_i, y_i) \subset \mathbb{R}^2$$

Cada $\boldsymbol{q}_i$ es un vector de posición en el plano.

1. Vector tangente [9]
   Para conocer la dirección de la curva en $\boldsymbol{q}_i$, necesitamos su tangente. Como tenemos una curva discreta, la aproximamos con diferencias entre puntos vecinos:

$$\vec{t}_i = \begin{cases} \boldsymbol{q}_1 - \boldsymbol{q}_0 & i = 0 \\ \boldsymbol{q}_{i+1} - \boldsymbol{q}_{i-1} & 0 < i < n-1 \\ \boldsymbol{q}_{n-1} - \boldsymbol{q}_{n-2} & i = n-1 \end{cases} \tag{16}$$

   Donde $\vec{t}_i$ es un vector dirreción de la curva en el punto *i*, es decir este paso nos dice "hacía donde va la curva" en cada punto.

2. Vector normal unitario
   La normal es un vector perpendicular a la tangente. Para obtenerla basta con rotar la tangente 90º:

$$\vec{n}_i = \frac{1}{\|\vec{t}_i\|}(-t_{iy}, t_{ix}) \tag{17}$$

   Donde $(-t_{iy}, t_{ix})$ es la rotación de $\vec{t}_i = (t_{iy}, t_{ix})$ y al dividir por $\|\vec{t}_i\|$ asegura que la normal tenga longitud uno, es decir este paso nos da "hacía que lado desplazamos".

3. Desplazamiento por la normal
   Finalmente construimos el punto desplazado [9]:

$$\boldsymbol{q}_i^{desp} = \boldsymbol{q}_i + d\vec{n}_i \tag{18}$$

   Donde $\boldsymbol{q}_i^{desp}$ es un vector posición de la curva paralela, d es un escalar que fija la separación entre la curva original y la paralela. Si $d > 0$ la paralela queda hacia la izquierda y si $d < 0$, hacia la izquierda

4. Curva resultante
   Repitiendo el proceso para todos los puntos:

$$\boldsymbol{Q}^{desp} = \{\boldsymbol{q}_0^{desp}, \boldsymbol{q}_1^{desp}, \dots, \boldsymbol{q}_{n-1}^{desp}\} \tag{19}$$

   Obtenemos la trayectoria paralela.

3.5.3 Explicación del código

La **Figura 3.14** es la función que recibe como entrada una curva discreta definida en un conjunto de puntos $\boldsymbol{p}_i = (x_i, y_i)$. Devuelve un array *n* con los vectores unitarios en cada uno de los puntos. Si la trayectoria tiene menos de dos puntos, devuelve directamente *n* ya que no se puede calcular direcciones en ese caso. A continuación, se recorre la curva con un bucle.

Para cada índice *i*, se calcula un vector de diferencia *d* que representa la dirección local de la curva (16): en el primer punto se usa la diferencia con el siguiente, en el último con el anterior, y en los intermedios con el punto anterior y el siguiente. Este vector *d* se normaliza dividiéndolo entre su norma *np.linalg.norm(d)* [10], añadiendo un pequeño valor $1e-12$ para evitar divisiones por cero en casos degenerados. Una vez normalizado este, se construye el vector normal mediante la operación *[-t[1], t[0]]* (17), que corresponde a rotar el vector tangente 90 grados. Finalmente, la función devuelve el array *n* con todas las normales unitarias de la curva.

Por último, el desplazamiento se introduce en la función de la **Figura 3.36** en la línea *puntos_desplazados=puntos_control_base+(sep*direccion)*normales* donde estamos introduciendo la ecuación (18), y el conjunto de los puntos desplazados componen $\boldsymbol{Q}^{desp}$ (19).

En resumen, el código implementa de manera práctica el cálculo de las normales: crea un array vacío, recorre los puntos, calcula la dirección local con diferencias, normaliza, rota el vector y guarda la normal.

```
def calcular_normales_puntos(p):
    N = len(p); n = np.zeros_like(p)
    if N < 2: return n
    for i in range(N):
        if i == 0: d = p[1]-p[0]
        elif i == N-1: d = p[-1]-p[-2]
        else: d = p[i+1]-p[i-1]
        t = d/(np.linalg.norm(d)+1e-12); n[i] = np.array([-t[1], t[0]])
    return n
```

Figura 3.14. Implementación en Python de la function calcular_normales_puntos. Este procedimiento calcula el vector normal unitario en cada punto de una curva discreto $\boldsymbol{Q}$*, utilizando direfencias entre puntos adyacentes para estimar la tangente y una rotación de 90º para obtener la normal.*

3.5.4 Resultado del bloque

La **Figura 3.15** muestra el resultado de aplicar el bloque B al conjunto de puntos muestreados obtenidos del bloque A. A partir de la curva base $\boldsymbol{Q}$, representada en color negro, se calculan las normales unitarias en cada punto y se utiliza un desplazamiento fijo *d* sobre dichas direcciones para generar la curva paralela $\boldsymbol{Q}^{desp}$ representada en rojo.

Este resultado confirma que el algoritmo implementado en el bloque B es capaz de construir correctamente las trayectorias paralelas a la curva original.

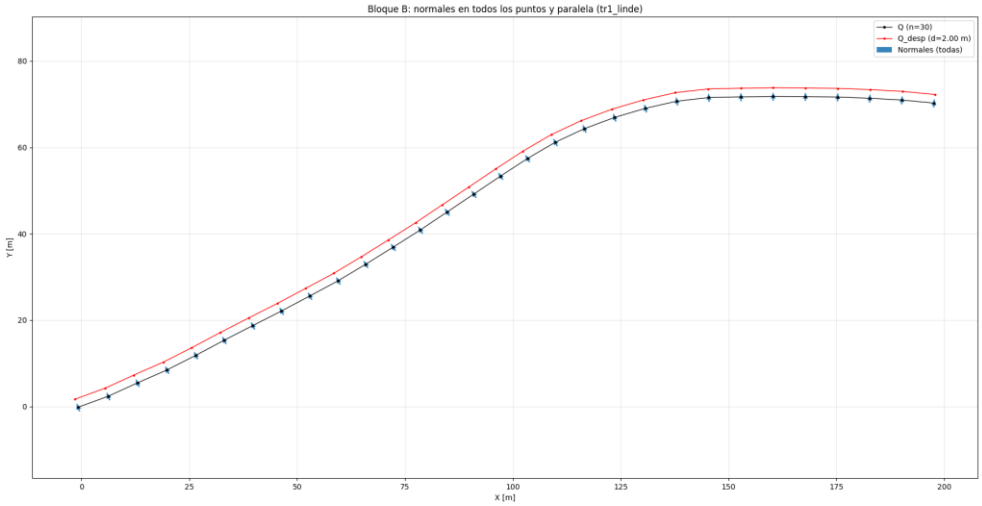

Figura 3.15. Representación de la trayectoria base $\boldsymbol{Q}$ *(negro) y de su paralela* $\boldsymbol{Q}^{desp}$ *(rojo), obtenida mediante el desplazamiento de cada punto en la dirección de la normal unitaria a una distancia d.*

3.6 Enderezado local – Bloque C ($\boldsymbol{Q}^{desp} \rightarrow \boldsymbol{Q}^{ender}$)

3.6.1 Descripción del algoritmo

Este bloque toma la paralela del Bloque B $\boldsymbol{Q}^{desp}$ y realiza un enderezado de forma iterativa. Recorre solo los puntos interiores y. para cada uno. Prueba pequeños desplazamientos sobre su normal, siempre dentro de recorrido permitido [-STM, +SM] [3]. Si el cambio mejora la forma, se acepta; si no, se deja como está. Los extremos permanecen fijos para conservar la dirección general. El proceso repite varias pasadas hasta que los cambios son apenas apreciables o se agota el máximo de iteraciones. El resultado es $\boldsymbol{Q}^{ender}$, una trayectoria más suave y recta.

La **Figura 3.16** muestra el diagrama de flujo del Bloque C, donde este bloque toma la paralela del Bloque B y mediante iteraciones realiza un enderezado progresivo. Para cada punto interior mira en su normal y prueba pequeños desplazamientos permitidos por SM/STM. Elige el que mejor suaviza la curva y pasa al siguiente punto. Cuando termina una pasada comprueba si ya casi no cambia nada; si aún cambia, hace otra pasada. Los extremos se mantienen fijos para conservar el rumbo. El resultado es la misma trayectoria, pero más suave y alineada.

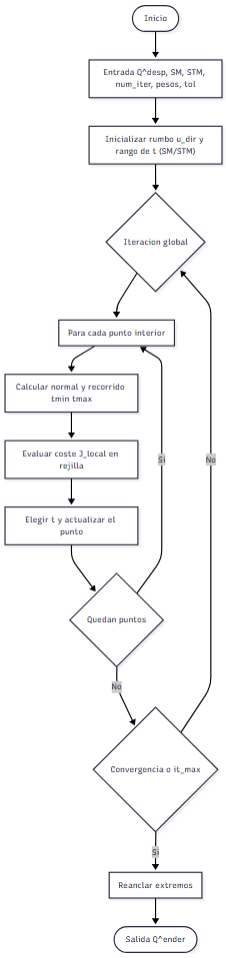

Figura 3.16. Diagrama de flujo del Bloque C. A partir de la paralela obtenida del Bloque B el algoritmo itera globalmente; en cada iteración recorre los puntos interiores y evalúa la función de coste y acepta el mejor desplazamiento y continua hasta converger.

3.6.2 Matemática asociada

Tras el bloque B, para la curva paralela obtenida $\boldsymbol{Q}^{desp} = \{\boldsymbol{q}_i^{desp}\}_{i=0}^{n-1}$ es necesario aplicar un proceso de enderezado que reduzca la curvatura, suavice irregularidades y acerque progresivamente la trayectoria a una línea recta. El enderezado no puede realizarse de manera arbitraria, debe respetar condiciones agronómicas, como el solape máximo (SM) y de zona sin tratar máximo (STM), que fijan un rango de desplazamientos laterales admisibles

Para formalizar este proceso, se define una función de coste global J (20)

El enderezado se plantea, por tanto, como un problema de optimización discreta, encontrar la nueva secuencia de puntos $\boldsymbol{q}_i^{ender}$ que mínima J y obtener $\boldsymbol{Q}^{ender} = \{\boldsymbol{q}_i^{ender}\}_{i=0}^{n-1}$. De esta forma, se logra que, tras varias paralelas, el trazado tienda a ser recto.

El enderezado se formula como la minimización de:

$$J_i(t) = w_L \Delta L_i + w_B \Delta K_i + \frac{w_A}{M-2} \Delta A_i + \frac{w_R}{M-2} \Delta R_i + M(t) \qquad (20)$$

La función de coste propuesta en este trabajo ha sido desarrollada específicamente en el marco del presente TFG, combinando criterios geométricos y agronómicos. Por tanto, la función de coste procede una combinación original diseñada para este trabajo.

Donde M es el número de puntos de control, los términos Δ (·) son cambios locales (antes/después de mover el punto en evaluación $\boldsymbol{q}_i$) y t un parámetro acotado tal que $t \in (-STM, SM)$:

Los parámetros que componen la función de coste son:

- ΔL (longitud local): evita encogimientos o estiramientos bruscos de la trayectoria.
- ΔK (curvatura local): suaviza posibles oscilaciones.
- ΔA (alineación con la recta global): alinea tangentes locales con la dirección de los extremos para que las paralelas no se desvíen.
- ΔR (rectitud global): fuerza a que la curva tienda a la recta "tirando" de cada punto hacia una recta imaginaria que une los extremos.
- M (t): garantiza que el desplazamiento no rompa las condiciones de solape o sin tratar (SM, STM

Ahora se va a explicar al detalle la función de coste y todos los parámetros que la engloban:

1. Longitud local $\Delta L_i(t)$
   Este parámetro se encarga de medir como cambia la longitud de los segmentos que inciden en $\mathbf{q}_i$ al desplazarlo, con el objetivo de mantener la longitud local razonable evita "serruchos" y estabiliza el ajuste [11].

$$L_i(\boldsymbol{q}) = \left|\left|\boldsymbol{q}_i - \boldsymbol{q}_{i-1}\right|\right| + \left|\left|\boldsymbol{q}_{i+1} - \boldsymbol{q}_i\right|\right| \tag{21}$$

$$\Delta L_i(t) = L_i\left(\boldsymbol{q}^{(t)}\right) - L_i(\boldsymbol{q}) \tag{22}$$

   El termino ΔL_i corresponde a la variación de longitud local de los segmentos incidentes en $\boldsymbol{q}_i$.

2. Curvatura discreta $\Delta K_i(t)$
   Se basa en la segunda diferencial discreta para evaluar el cambio en la suma de curvaturas de los vecinos de i para suavizar las posibles oscilaciones y evitar picos de curvatura:

$$K(\boldsymbol{q}) = \sum_{j=0}^{M-3} ||\Delta^2 \boldsymbol{q}_j||^2 \tag{23}$$

$$\Delta^2 \boldsymbol{q}_j = \boldsymbol{q}_j - 2\boldsymbol{q}_{j+1} + \boldsymbol{q}_{j+2} \tag{24}$$

$$\Delta K(\boldsymbol{q}) = \sum_{j\in\{i-2,i-1,i\}}^{M-3} (||\Delta^2 \boldsymbol{q}_j(t)||^2 - ||\Delta^2 \boldsymbol{q}_j||^2) \tag{25}$$

   La medida de curvatura discreta se construye a partir de la segunda diferencia finita $\Delta^2 \boldsymbol{q}_j$ que actúa como aproximación discreta de la segunda derivada de la curva [11].

3. Alineación con el rumbo $\Delta A_i(t)$
   Calcula cómo varía la alineación de la tangente local con la dirección general de la curva. Con esto evitamos que la trayectoria se tuerza respecto a la dirección de trabajo. En la función de coste normalizamos por $M-2$ para que la magnitud no dependa del número de puntos [12]

   Definimos el rumbo global como un vector unitario entre los extremos de la curva, este vector marcara la dirección "objetivo":

$$\vec{u} = \frac{\boldsymbol{q}_{M-1}-\boldsymbol{q}_0}{||\boldsymbol{q}_{M-1}-\boldsymbol{q}_0||} \tag{26}$$

Medimos la orientación local de la curva mediante tangentes de dos saltos para que sea mejor la estimación d:

$$\vec{\tau}_k = \boldsymbol{q}_{k+2} - \boldsymbol{q}_k \tag{27}$$

$$\widehat{\tau_k} = \frac{\vec{\tau}_k}{||\vec{\tau}_k||} \tag{28}$$

Ahora penalizamos la diferencia entre 1 y el producto escalar de τ_ky u. Esto nos dirá si están alineados o no:

$$A(Q;u) = \sum_k (1 - (\hat{\tau}_k \cdot \vec{u}) \tag{29}$$

Si τ_k está alineada con u, el producto escalar ≈ 1 ⇒ penalización ≈ 0.

Si está perpendicular, el producto escalar ≈ 0 ⇒ penalización ≈ 1 (máxima).

4. Rectitud (Distancia a $\overline{\boldsymbol{q}_0\boldsymbol{q}_{M-1}}$) $\Delta R_i(t)$
   Este parámetro evalúa el cambio en la distancia del punto $\boldsymbol{q}_i$ a una recta imaginaria que une los extremos de la curva ($\boldsymbol{q}_0$ y $\boldsymbol{q}_{N-1}$). Igual que el parámetro anterior también se normaliza por $M-2$.

   Con $A = \boldsymbol{q}_0, B = \boldsymbol{q}_{N-1}$ y $n_{AB} = ||B-A||$, la distancia cuadrática de $\boldsymbol{q}_i$ a la recta $\overline{AB}$

$$R_i(\boldsymbol{q}) = \frac{((B-A)*(\boldsymbol{q}_i-A))^2}{||B-A||^2,} \tag{30}$$

$$\Delta R_i(t) = R_i\left(\boldsymbol{q}^{(t)}\right) - R_i(\boldsymbol{q}) \tag{31}$$

5. Penalización agronómica $M(t)$
   Con esta función nos aseguramos de que el desplazamiento lateral t no supere el SM ni la STM [3]:

$$M(t) = \begin{cases} c\frac{t^2}{1-\frac{t}{SM}} & t > 0 \\ d\frac{t^2}{1+\frac{t}{STM}} & t \leq 0 \end{cases} \tag{32}$$

c y d son constantes de ajuste de exactitud que determinan la dureza de la barrera. Con valores como $c = d = 1$ penaliza mucho si nos acercamos al límite, por el otro lado si $c = d = 10^{-4}$ permite que los puntos se acerquen más al límite antes de penalizar.

3.6.3 Explicación del código

La **Figura 3.17** muestra la función principal que realiza el enderezado. La función toma la curva desplazada por normales del Bloque B y la endereza iterativamente. Lo primero que hace es copiar los puntos a P y fijar los extremos A y B como fijos, porque la recta que une ambos define el rumbo global frente al que se mide la alineación. También inicializa un desplazamiento acumulado por punto *t_acum* y establece, para cada punto, el recorrido de movimiento permitido por los criterios agronómicos: de STM y SM.

A continuación, recorre un número finito de iteraciones. En cada iteración calcula un parámetro α entre 0 y 1 con el que interpola linealmente los pesos del coste: hace crecer el peso de la longitud local (para estabilizar al final), decrecer el de la curvatura discreta de segunda diferencia (para aplanar dientes al principio sin sobre–suavizar al final) y aumentar el de la alineación de rumbo, de manera que, cuando la curva ya está suave, se priorice orientar sus tangentes hacia la dirección global. El peso hacia la recta de extremos (*wR*) puede venir elevado desde fuera cuando generamos pasadas tardías, porque en ese momento el objetivo práctico es "tirar" de la trayectoria hacia la rectitud.

Con los pesos fijados para la iteración, vuelve a calcular el rumbo global como el vector unitario u dirigido de A a B. Para cada punto interior $\boldsymbol{q}_i$ comprueba primero si ya está prácticamente alineado con curvatura local despreciable; si se cumple, no lo toca en esta vuelta, lo que ahorra cómputo. Si el punto merece tratamiento, estima su dirección local con la bisectriz de las tangentes a izquierda y derecha y construye la normal unitaria $\hat{n}_i$ girando noventa grados esa bisectriz. El enderezado se hace precisamente moviendo el punto sobre esa normal (**Figura 3.5**), porque desplazar en normal reduce el ángulo entre segmentos adyacentes sin introducir artificialmente avances o retrocesos a lo largo de la curva, lo que se traduce en una caída natural de curvatura.

Antes de probar movimientos, la función evalúa las magnitudes "de partida" en la vecindad de i: la longitud local como suma de los dos tramos incidentes (22), la curvatura discreta mediante la segunda diferencia (25), la desalineación con el rumbo (29), y la distancia perpendicular del punto a la recta $\overline{AB}$ (31). Con esos valores base define el coste local $J_i(t)$ como la suma ponderada de los incrementos de esas magnitudes al mover $\boldsymbol{q}_i$ a $\boldsymbol{q}_i^{(t)} = \boldsymbol{q}_i + (t - t_o)\vec{n}_i$, más la penalización

agronómica $M(t)$ (32). Esta última es la barrera asintótica que hemos descrito previamente: crece con rapidez al aproximarse a +SM o a −STM, de modo que el algoritmo "sienta" los límites de solape y sin tratar sin necesidad de imponer cortes duros; con constantes pequeñas la barrera permite acercarse algo al límite y acelerar la convergencia, y con constantes mayores endurece el borde para escenarios más conservadores.

La minimización de $J_i(t)$ se resuelve como una búsqueda: primero muestrea siete candidatos uniformes en el recorrido permitido del punto, elige el mejor y, alrededor de ese mínimo provisional, vuelve a muestrear siete valores en un intervalo más estrecho. Con el mínimo refinado actualiza el punto y anota el desplazamiento efectivo. Cuando la media de desplazamientos de una iteración cae por debajo de la tolerancia, el proceso se detiene; en caso contrario, continúa hasta agotar el máximo de iteraciones. Al terminar, reafirma los anclajes en los extremos para garantizar que el resultado preserve el punto de partida y el de llegada.

```python
def aplicar_enderezado_local_gs( puntos_desplazados, SM, STM, num_iter=60, wL_ini=2.0,  wL_fin=12.0,wB_ini=0.6,  wB_fin=0.03,  wA_ini=1.2,  wA_fin=16.0, wR=0.0, exact_c=1e-8, exact_d=1e-8, tol_mean=1e-4):
    P = puntos_desplazados.copy(); N = len(P)
    if N < 3: return P
    A, B = P[0].copy(), P[-1].copy()
    AB = B - A; nAB = np.linalg.norm(AB)
    t_acum = np.zeros(N)
    rng_ini = (-STM, SM)
    shrink = 0.6
    for it in range(num_iter):
        alpha = it/max(1, num_iter-1)
        wL = wL_ini + (wL_fin - wL_ini)*alpha
        wB = wB_ini + (wB_fin - wB_ini)*alpha
        wA = wA_ini + (wA_fin - wA_ini)*alpha

        u_dir = (P[-1]-P[0]); u_dir = u_dir/(np.linalg.norm(u_dir)+1e-12)
        if alpha < 0.8: rng_min, rng_max = rng_ini
        else: rng_min, rng_max = rng_ini[0]*shrink, rng_ini[1]*shrink

        mejoras = []

        for i in range(1, N-1):
            tau_i = P[i+1] - P[i-1]
            tn = np.linalg.norm(tau_i)
            ali = abs(np.dot(tau_i/tn, u_dir)) if tn > 1e-12 else 0.0
            curv_loc = _curv_local(P, i)
            if ali > 0.995 and curv_loc < 1e-4: mejoras.append(0.0); continue
            d_prev = P[i]-P[i-1]; d_next = P[i+1]-P[i]
            v1 = d_prev/(np.linalg.norm(d_prev)+1e-12)
            v2 = d_next/(np.linalg.norm(d_next)+1e-12)
            tng = v1 + v2
            if np.linalg.norm(tng) < 1e-12: tng = P[i+1]-P[i-1]
            tng = tng/(np.linalg.norm(tng)+1e-12)
            nrm = np.array([-tng[1], tng[0]])

            t0 = t_acum[i]
            tmin = max(-STM, t0 + rng_min)
            tmax = min( SM, t0 + rng_max)

            L0   = _len_local(P, i)
            K0   = _curv_local(P, i)
            A0,_ = _alig_local(P, i, u_dir)
            R0   = _line_local(P, i, A, B, nAB)

            J_local_at = make_J_local_at( P, i, t0, nrm, u_dir, A, B, nAB, L0, K0, A0, R0, wL, wB, wA, wR, N,SM, STM, exact_c, exact_d)

            grid  = np.linspace(tmin, tmax, 7)
            vals  = [J_local_at(t) for t in grid]
            t_star = grid[int(np.argmin(vals))]
            width = (tmax - tmin)*0.35
            rmin, rmax = max(-STM, t_star - width), min(SM, t_star + width)
            grid2 = np.linspace(rmin, rmax, 7)
            vals2 = [J_local_at(t) for t in grid2]
            t_best = float(grid2[int(np.argmin(vals2))])

            P[i] = P[i] + (t_best - t0)*nrm
            mejoras.append(abs(t_best - t0)); t_acum[i] = t_best
        if np.mean(mejoras) < tol_mean: break
    P[0], P[-1] = A, B
    return P
```

Figura 3.17. Función aplicar_enderezado_local_gs, núcleo del Bloque C que recibe $\boldsymbol{Q}^{desp}$ y devuelve $\boldsymbol{Q}^{ender}$.

La **Figura 3.18** muestra todo el contexto local necesario para evaluar, para un punto interior $\boldsymbol{q}_i$, el coste incremental de moverlo una cantidad escalar t sobre su dirección de enderezado (20).

La función *make_J_local_at* prepara y devuelve una función evaluadora de coste para un punto interior concreto. Su cometido es muy preciso: dado el estado actual de la trayectoria, el índice del punto que estamos tratando, la dirección de enderezado y los valores de referencia antes de mover el punto, construye una función *J_local_at* que responde a la pregunta: *si desplazo este punto una cantidad escalar t sobre su normal, ¿mejora o empeora la calidad geométrica de la curva dentro de*

los límites agronómicos? Para poder contestar sin tener que pasar decenas de parámetros en cada llamada, *make_J_local_at* captura por cierre todo el contexto necesario: el array de puntos, el índice, la normal, el rumbo global de la pasada, los extremos anclados, las medidas locales "antes" de mover (longitud, curvatura, alineación y rectitud), los pesos de cada término, el tamaño de la curva para las normalizaciones y los límites SM/STM junto con la dureza de su penalización.

La función resultante, *J_local_at*, no altera el estado global cuando se usa: hace una copia del punto, lo mueve temporalmente en la dirección indicada la cantidad pedida, vuelve a medir solo en la vecindad de ese punto las cuatro magnitudes que nos interesan (longitud en los dos tramos adyacentes, curvatura discreta por segunda diferencia, alineación de las tangentes locales con el rumbo y distancia perpendicular del punto a la recta definida por los extremos), y a continuación restaura el punto a su posición original. Con esas cuatro diferencias "después–antes" calcula un balance ponderado con los pesos vigentes en esa iteración, aplica las normalizaciones por número de puntos donde corresponde para que el valor no dependa del tamaño de la curva, y le suma la penalización agronómica asociada al propio t, que está diseñada para crecer al acercarse a los límites de solape y de zona sin tratar. El resultado es un único número real: cuanto menor es, mejor es ese desplazamiento desde el punto de vista del enderezado y del respeto a SM/STM.

```
def make_J_local_at(
        P, i, t0, nrm, u_dir,
        A, B, nAB,
        L0, K0, A0, R0,
        wL, wB, wA, wR, N,
        SM, STM,
        exact_c=1e-8, exact_d=1e-8
):
    def J_local_at(t: float) -> float:
        old = P[i].copy()
        P[i] = old + (t - t0) * nrm

        L1   = _len_local(P, i)
        K1   = _curv_local(P, i)
        A1,_ = _alig_local(P, i, u_dir)
        R1   = _line_local(P, i, A, B, nAB)

        P[i] = old

        dJ_geo = (
                wL * (L1 - L0) +
                wB * (K1 - K0) +
                wA * ((A1 - A0) / max(1, (N - 2))) +
                wR * ((R1 - R0) / max(1, (N - 2)))
        )

        barr = penalizacion_M(t, SM, STM, c=exact_c, d=exact_d)
        return float(dJ_geo + barr)

    return J_local_at
```

Figura 3.18 Función make_J_local_at que hace de evaluador de la function de coste para comprobar si el punto desplazado mejora o empeora el enderezado. Para ello de apoya en la subfunción J_local_at

La **Figura 3.19** muesta la función *_len_*local. Esta función calcula la longitud "que ve" el punto en su vecindad inmediata: suma las longitudes de los dos tramos que comparten ese punto, el que lo

une con su vecino anterior y el que lo une con el siguiente (22). En los extremos solo habría un tramo, aunque en el enderezado no se tocan los extremos y, por tanto, la función se emplea efectivamente en puntos interiores. Su papel en el coste es muy claro: cuando se prueba un desplazamiento provisional del punto, se vuelve a medir esa longitud local y se compara con la de partida; si el movimiento provoca un acortamiento o un estiramiento brusco en esa pequeña ventana, el término de longitud crece y desincentiva ese cambio. Con ello se evita el "serrucho" típico de los ajustes punto a punto y se estabiliza la geometría a escala local cuando ya se ha reducido la curvatura y conviene consolidar el trazado sin introducir micro–oscilaciones.

```
def _len_local(P, i):
    L = 0.0
    if i-1 >= 0: L += np.linalg.norm(P[i]-P[i-1])
    if i+1 < len(P): L += np.linalg.norm(P[i+1]-P[i])
    return L
```

Figura 3.19. Función _len_local la cual devuelve la lonfitus de los dos segmentos que inciden en el punto $\boldsymbol{q}_i$.

La función *_curv_local* de la **Figura 3.20** mide cuánta "curvatura" hay alrededor del punto de forma puramente local y muy barata de calcular (25). Lo hace evaluando en tres ventanas que son las únicas que se ven afectadas cuando movemos el punto, las que empiezan en $i-2, i-1$ e i. En cada ventna se observa cómo cambia la dirección de la curva entre tres puntos consecutivos; si la trayectoria hace un "quiebro" o un "diente", el valor crece, y si la zona es suave o casi recta, el valor es pequeño.

```
def _curv_local(P, i):
    N = len(P); J = 0.0
    for j in (i-2, i-1, i):
        if 0 <= j <= N-3:
            seg = P[j] - 2*P[j+1] + P[j+2]
            J += float(np.dot(seg, seg))
    return J
```

Figura 3.20. Función _curv_local la cual devuelve un valor que refleja la curvatura entre tres puntos vecinos.

La función *_alig_local* de la **Figura 3.21** cuantifica cuánto se desvía la orientación local de la curva respecto al rumbo global u (26) que marca la recta entre los extremos. Para evitar medidas ruidosas basadas en un solo segmento, no mira las aristas inmediatas sino tangentes de dos saltos: compara la dirección que une P_k con P_{k+2}. Con eso obtiene una estimación más estable de "hacia dónde va" la curva en esa zona (29). En la práctica evalúa dos pequeñas ventanas que rodean al punto, las que empiezan en $i-2$ y en i (si existen), y en cada una calcula cuánto se aparta esa tangente local del rumbo u. Usa siempre el valor absoluto del coseno para que la métrica sea independiente del sentido (avanzar o retroceder no cambia el grado de alineación) y eleva la desviación al cuadrado para que la penalización sea suave cuando ya se está muy alineado y crezca si la dirección se tuerce. Si en alguna ventana la longitud es prácticamente nula, la función la descarta para evitar divisiones peligrosas.

```python
def _alig_local(P, i, u):
    N = len(P); S = 0.0; cnt = 0
    for k in (i-2, i):
        if 0 <= k <= N-3:
            tau = P[k+2] - P[k]
            n = np.linalg.norm(tau)
            if n > 1e-12:
                c = abs(np.dot(tau/n, u))
                S += (1.0 - c*c); cnt += 1
    return S, cnt
```

Figura 3.21. Función _alig_local mide cuanto se orientan las tangente locales en la dirección objetivo u definida por los extremos.

La función *_line_local* de la **Figura 3.22** mide, para el punto interior $\boldsymbol{q_i}$, cuánto se separa en perpendicular respecto a la recta de extremos definida por $A = P[0]$ y $B = P[-1]$. Este término empuja a que, a medida que se avanaza el enderezado y disminuye la curvatura, el trazo se acerque a la recta base (31). No actúa en los extremos

```python
def _line_local(P, i, A, B, nAB):
    if i == 0 or i == len(P)-1: return 0.0
    AP = P[i] - A
    cross = (B[0]-A[0])*AP[1] - (B[1]-A[1])*AP[0]
    return (cross*cross)/(nAB*nAB + 1e-12)
```

Figura 3.22. Función _line_local calcula la distancia de un punto interno a una recta que une los extremos de la trayectoria, empujando este punto hacia esa recta según disminuye la curvatura.

La función *penalización_M* de la **Figura 3.23** actúa como un guardarraíl agronómico durante el enderezado. Su cometido es encarecer los desplazamientos laterales que se acercan a los límites operativos del equipo: SM (solape máximo, lado positivo) y STM (zona sin tratar máxima, lado negativo). En lugar de cortar en seco cuando se rebasa el límite, la penalización crece suavemente al principio y muy rápido al aproximarse al borde, de modo que el algoritmo "sienta" el límite y evite escoger movimientos que lo rocen.

La función es asimétrica a propósito: trata de forma distinta el desplazamiento hacia el solape (valores positivos de t, controlado por SM y el coeficiente c) y el desplazamiento hacia la zona sin tratar (valores negativos de t, controlado por STM y el coeficiente d). Esto encaja con la práctica agrícola, donde no siempre es igual de tolerable solapar que dejar sin cubrir; por eso se puede ajustar c y d de forma independiente (32). Con c y d muy pequeños la barrera es blanda: permite acercarse bastante al límite antes de penalizar con fuerza, lo que da más libertad al algoritmo para enderezar rápido en las primeras iteraciones. Con c y d más grandes la barrera se vuelve dura: desaconseja enseguida cualquier movimiento que intente aproximarse a los márgenes, útil cuando priorizas seguridad o cuando el enderezado ya está hecho y solo quieres retoques finos.

```python
def penalizacion_M(t, SM, STM, c=1e-8, d=1e-8):

    if t > 0:
        return c*(t*t)/(1 - t/(SM+1e-12) + 1e-12)
    else:
        return d*(t*t)/(1 + t/(STM+1e-12) + 1e-12)
```

Figura 3.23. Función penalización_M devuelve una penalización suave pero creciente según se llega a los límites SM/STM. Los coeficientes c y d controlan cuanto penaliza esta función.

3.6.4 Resultado del bloque

La **Figura 3.24** ilustra el efecto aislado del Bloque C. Partimos de la paralela "cruda" (naranja), que reproduce fielmente la geometría de la original (Bloque B). El enderezado local (puntos azules) desplaza cada nodo únicamente en su normal, dentro del corredor permitido por SM/STM, buscando reducir a la vez los dientes de curvatura, estabilizar la longitud de los tramos adyacentes y orientar las tangentes hacia el rumbo global que marcan los extremos. El resultado es una pasada más suave y con trazado más estable, sin invadir los márgenes agronómicos. La línea azul clara corresponde al spline reconstruido tras el enderezado: aporta continuidad y consolida la suavidad alcanzada, preparándola para ser base de la siguiente pasada en el encadenado. Se aprecia que, especialmente en la zona de mayor curvatura, el azul se aproxima a un perfil más regular que el naranja, y que el spline sigue esa tendencia sin picos ni sobre oscilaciones; dicho de otro modo. Se construyo el spline para la visualización ya que esto es un paso que se implementa posteriormente en el Bloque F y que no se implementa en este bloque

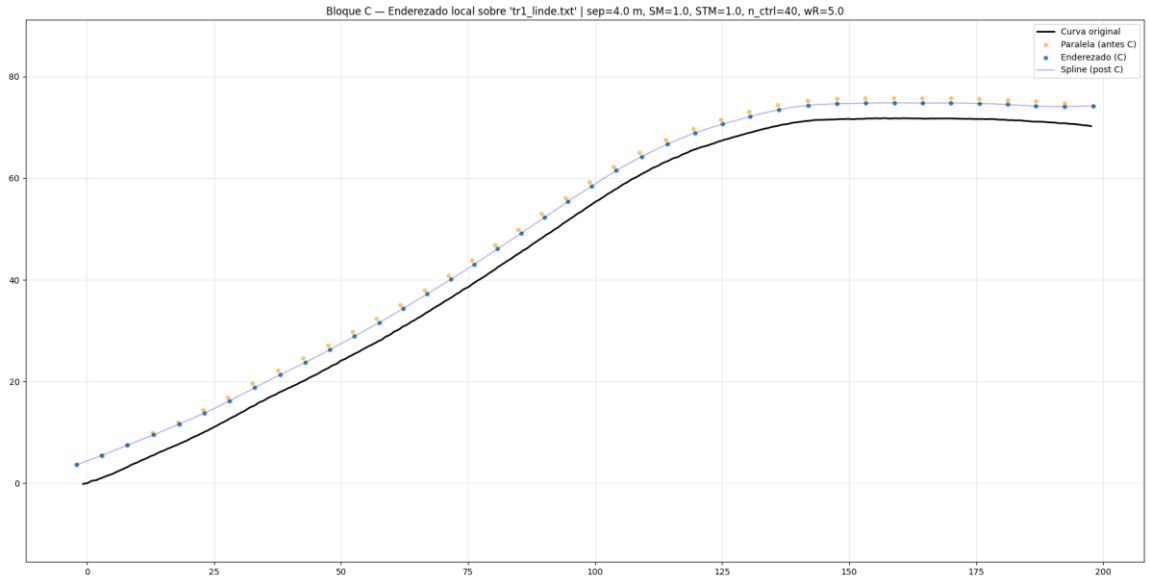

Figura 3.24. Resultado del Bloque C (enderezado local) sobre la pasada generada a partir de tr1_linde.txt. En negro se muestra la curva original; en naranja, los puntos de la paralela obtenida tras el desplazamiento por normales (Bloque B); en azul, los puntos después del enderezado local (Bloque C); y en azul claro, el spline cúbico que se reconstruye tras C. Parámetros de la demo: separación 4,0 m, límites SM=1,0 m y STM=1,0 m, 40 puntos de control y peso de recta wR=5,0.

3.7 Reducción adaptativa – Bloque D ($\boldsymbol{Q^{ender} \rightarrow Q^{red}}$)

3.7.1 Descripción del algoritmo.

Tras el enderezado del Bloque C, este bloque decide cuántos puntos de control son realmente necesarios y eliminan los redundantes. Para ello mide la curvatura de la pasada y según el nivel de "giro", fija un tamaño de malla entre unos límites $[n_{min}, n_{max}]$. Más puntos si aún hay curvas, menos puntos si la trayectoria ya está lista. La reducción se hace de tal que nunca se sube el número de puntos y limita la caída por iteración. Como resultado obtenemos $\boldsymbol{Q^{red}}$, una malla más ligera de puntos, pero fiel a la forma.

El diagrama de la **Figura 3.35** resume el Boque D donde recibe la trayectoria del Bloque C $\boldsymbol{Q^{ender}}$ y calcula la curvatura media y máxima. Con esto calcula un indicador unidimensional s y lo transforma en un tamaño de malla n_{ctrl} dentro de $[n_{min}, n_{max}]$. La salida es n_{ctrl} que se usará en el remallado (Bloque E) y el spline (Bloque F)

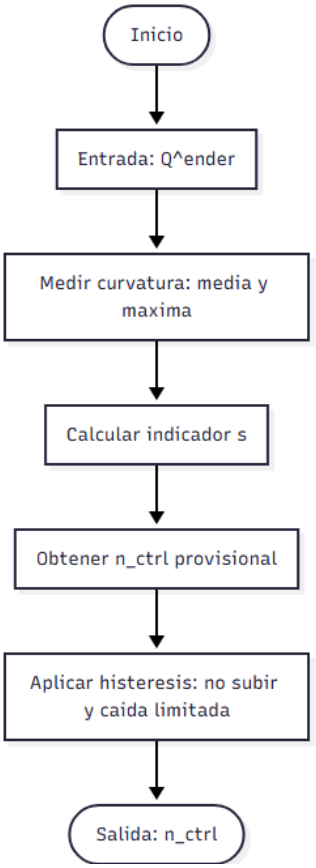

Figura 3.25. Diagrama de flujo del Bloque D, a partir de la pasada enderezada obtenemos un nuevo número de puntos de control.

3.7.2 Matemática asociada

Para cada punto interior, sea $\vec{v}_{i-} = \boldsymbol{q}_i^{ender} - \boldsymbol{q}_{i-1}^{ender}$ $\vec{v}_{i+} = \boldsymbol{q}_{i+1}^{ender} - \boldsymbol{q}_i^{ender}$ los vectores de los tramos que "entran" y "salen" y que tratan de capturar la dirección y longitud de la curva justo antes y después de i.

Definimos el ángulo de giro:

$$\theta_i = \tan^{-1} 2\,((\vec{v}_{i-} \times \vec{v}_{i+}), dot(\vec{v}_{i-} \cdot \vec{v}_{i+})) \tag{33}$$

Que es el ángulo en radianes que hay que girar para pasar de $\vec{v}_{i-}$ a $\vec{v}_{i+}$ [13].

Calculamos la longitud efectiva local $L_i = \frac{1}{2}(\|\vec{v}_{i-}\| + \|\vec{v}_{i+}\|)$, que es la medida de "cuanta distancia" hay alrededor de i. Es la media de las longitudes de los dos tramos que confluyen en i.

Tenemos la curvatura discreta $k_i = \frac{|\theta_i|}{L_i}$ que es "cuanta distancia" hay alrededor de $\boldsymbol{q}_i$ [14]. Teniendo que:

- Si la curva es casi recta $|\theta_i| \approx 0$, entonces k_i pequeña
- Si hay un "codo" fuerte y corto $|\theta_i|$ grande, L_i pequeña y entonces k_i grande

Ahora con estos cálculos, para decidir el número de puntos de control n_{ctrl} seguimos una puntuación que llamamos s.

Tal que $n_{ctrl} = n_{min} + (n_{max} - n_{min}) * s^{\beta}$. Donde s, es un número adimensional entre 0 y 1 que resume "cuanta curvatura queda" en la pasada i comparada con la curvatura de referencia del inicio. Si $s \approx 1$ la pasada está tan rizada como al principio (no conviene recortar puntos), pero si $s \approx 0$ la pasada está mucho más suave (sí conviene recortar puntos).

$$\mathrm{r} = \left(\frac{\mathrm{w}*\mathrm{k_{med}}+(1-\mathrm{w})\mathrm{k_{max}}}{\mathrm{k_{ref}}}, 0, 1\right), \quad \mathrm{w}\in[0,1] \tag{31}$$

$$s(r)=\begin{cases}0, & r<0\\ r, & 0\le r\le 1\\ 1, & r>1\end{cases} \tag{32}$$

Donde:

- k_{med}: curvatura media de la polilínea enderezada $\boldsymbol{Q}_i^{ender}$
- k_{max}: curvatura máxima de la misma
- k_{ref}: valor de referencia que compara la curvatura de la pasada frente al inicio.

$$k_{ref} = \frac{1}{2}(k_{med}^{(0)} + k_{max}^{(0)}) \tag{33}$$

- w: pondera la importancia de "lo global" frente a "los picos"

Explicación de la formula:

1. Adimensional

Dividir por k_{ref} normaliza las curvaturas, trabajamos con curvatura actual / curvatura inicial.
Así mismo da igual si trabajamos en metros o en kilómetros s no depende de unidades ni del tamaño del campo

2. Equilibrio entre global y picos
   k_{med} capta el estado global de la curva mientras que k_{max} nos indica los puntos "críticos" con radios muy pequeños y cerrados.
   La combinación lineal $w * k_{med} + (1 - w)k_{max}$ es la forma más simple de mezclarlos.
   Si nos quedamos solo con la media no estamos teniendo en cuenta los picos críticos que puedan tener las trayectorias y si solo nos quedamos con la máxima tenemos mayor efecto en los picos, pero una reacción tardía en los tramos suaves.

   Con w alto damos prioridad a las curvas generales despreocupándonos más de los picos y con w bajo tenemos un resultado más sensible a los picos y más conservador.

3. Casos límites
   - Si las curvaturas coinciden con las de referencia $s = 1$
   - Si la pasada es recta $s \approx 0$.

4. Estabilidad
   En la fórmula anterior $n_{ctrl} = n_{min} + (n_{max} - n_{min}) * s^{\beta}$ [15]:
   - Con el parámetro s aseguramos que n_{ctrl} se mueva entre el mínimo y el máximo permitido
   - Con $\beta > 1$ se hace el recorte prudente: hasta que s no es pequeño, el descenso de puntos es moderado

3.7.3 Explicación del código

La función de la **Figura 3.26** recibe la curva como un array **P** y, si tiene menos de tres puntos, devuelve ceros porque no hay giro que medir. A continuación, construye para cada punto interior, los dos vectores que llegan y salen de él, sus longitudes sirven para detectar casos degenerados (segmentos prácticamente nulos). Con una máscara se ignoran esos casos y se normalizan solo los vectores válidos, evitando inestabilidades numéricas. Sobre cada pareja de vectores unitarios se calcula el giro local mediante el seno y el coseno implícitos (producto cruzado y producto escalar) y se obtiene el ángulo con arctan2 (33). Ese ángulo se pone en escala dividiéndolo por la longitud efectiva de los dos tramos adyacentes, lo que hace comparable la medida, aunque los puntos estén más o menos espaciados. El resultado por punto es una curvatura discreta sin signo: vale más cuanto más brusco es el cambio de dirección y tiende a cero cuando la trayectoria es recta. Finalmente, la rutina toma la media y el máximo de esa serie y los devuelve. Estas dos magnitudes resumen el

"tono" general de la pasada y la presencia de picos locales, y son las que el Bloque D usa justo después para decidir cuántos puntos de control conservar y cómo remallar por arco.

```python
def metricas_curvatura(P):
    P = np.asarray(P, float)
    N = len(P)
    if N < 3:
        return 0.0, 0.0

    v_prev = P[1:-1] - P[:-2]
    v_next = P[2:]   - P[1:-1]
    n_prev = np.linalg.norm(v_prev, axis=1)
    n_next = np.linalg.norm(v_next, axis=1)

    mask = (n_prev > 1e-12) & (n_next > 1e-12)
    if not np.any(mask):
        return 0.0, 0.0

    v_prev_u = np.zeros_like(v_prev); v_prev_u[mask] = v_prev[mask] / n_prev[mask, None]
    v_next_u = np.zeros_like(v_next); v_next_u[mask] = v_next[mask] / n_next[mask, None]

    cross = v_prev_u[:,0]*v_next_u[:,1] - v_prev_u[:,1]*v_next_u[:,0]
    dot   = np.einsum('ij,ij->i', v_prev_u, v_next_u)
    theta = np.arctan2(cross, dot)

    Leff  = 0.5*(n_prev + n_next)
    kappa = np.zeros_like(Leff)
    m2 = mask.copy()
    kappa[m2] = np.abs(theta[m2]) / (Leff[m2] + 1e-12)

    k_mean = float(np.mean(kappa[m2])) if np.any(m2) else 0.0
    k_max  = float(np.max(kappa[m2]))  if np.any(m2) else 0.0
    return k_mean, k_max
```

Figura 3.26. Función metricas_curvatura. Calcula, para una polilínea, la curvatura discreta basada en el ángulo de giro entre segmentos consecutivos y devuelve dos indicadores globales de la pasada: curvatura media y curvatura máxima.

La función de la **Figura 3.27** recibe la curvatura media y la máxima de la pasada junto con una referencia k_{ref}. Si la referencia es prácticamente nula, devuelve directamente el mínimo de puntos para evitar divisiones inestables. En el caso general, primero calcula un indicador adimensional s (32) comparando la curvatura actual con la de referencia: combina media y máximo con un peso w (por defecto 0'6 que favorece el comportamiento global frente a picos) y divide por *k_ref*. Ese s se recorta a [0,1]: valores cercanos a 0 representan curvas casi rectas y valores cercanos a 1 curvas exigentes. Después interpola entre *n_min* y *n_max* usando *s**beta*; con *beta>1* el mapeo es más agresivo cuando la curvatura ya es baja, empujando antes hacia *n_min* y evitando mantener puntos de control innecesarios.

Por último, redondea a entero y vuelve a acotar el resultado en [*n_min*, *n_max*]. Esta propuesta se estabiliza justo después con la histéresis: no permitir aumentos de puntos y limitar la caída por pasada, de modo que el tamaño de la malla descienda de forma suave y predecible.

```python
def nctrl_por_curvatura(k_mean, k_max, k_ref, n_min, n_max, w=0.6, beta=1.6):
    if k_ref < 1e-12:
        return n_min
    s = (w*k_mean + (1.0 - w)*k_max) / (k_ref + 1e-12)
    s = max(0.0, min(1.0, s))
    val = n_min + (n_max - n_min)*(s**beta)
    return int(max(n_min, min(n_max, round(val))))
```

Figura 3.27. Función nctrl_por_curvatura. Mapea la curvatura medida en la pasada a un número de puntos de control dentro de los límites [n_min,n_max], una referencia k_ref, una mezcla entre curvatura media y máxima, y un sesgo controlado por β.

3.7.4 Resultados del código

La imagen de la **Figura 3.28** muestra, para cada pasada, los puntos de control resultantes después del enderezado local y de la reducción adaptativa con remallado por arco. El degradado de color indica el orden de generación: a medida que avanzan las pasadas, se observa cómo el algoritmo reduce el número de puntos y los redistribuye de forma uniforme a lo largo de la trayectoria, manteniendo siempre los extremos anclados. La mayor densidad relativa se concentra de manera natural en la zona con más giro de la finca; en los tramos suaves, la malla se hace más escasa sin perder forma.

La imagen de la **Figura 3.29** recoge los splines finales de todas las paralelas. Este spline se ha construido solo para la visualización del resultado tras reducir los puntos, pero el spline final se construye en el último bloque tras reorganizar lo puntos en el bloque anterior y que vamos a ver más adelante

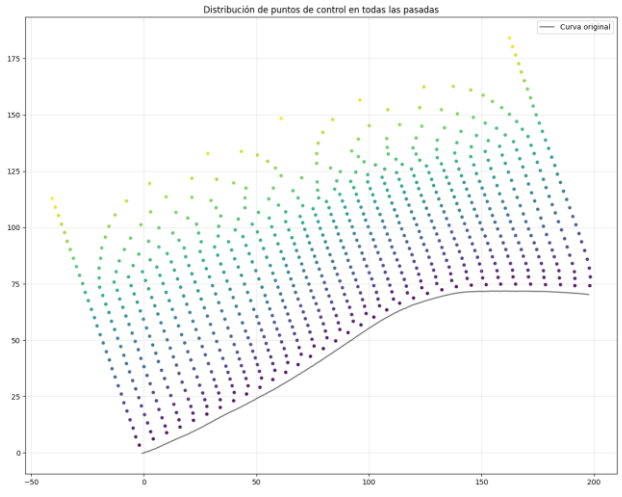

Figura 3.28. Distribución de los puntos de control tras los bloques C–D–E, coloreados por índice de pasada, sobre la curva original en negro.

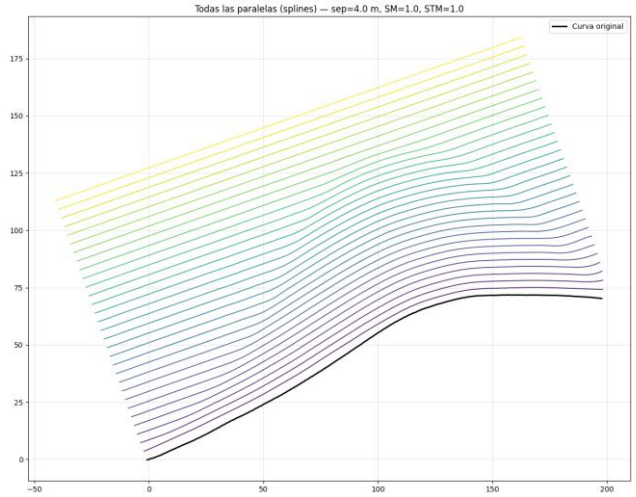

Figura 3.29. Splines finales de todas las paralelas (bloques C–D–E–F) con el mismo gradiente de color. Parámetros: sep = 4,0 m, SM = 1,0 m, STM = 1,0 m.

El gráfico de la **Figura 3.30** muestra cómo el sistema mantiene constante el número de puntos de control mientras la pasada sigue teniendo curvatura apreciable, y solo cuando la geometría se vuelve suficientemente dócil comienza una caída escalonada. Ese primer escalón corresponde a pasadas en las que la curvatura media y, sobre todo, la máxima sigue indicando que hacen falta muchos grados de libertad. A partir de cierta pasada, el enderezado ya ha eliminado gran parte de las ondulaciones y la métrica de curvatura desciende por debajo de la referencia; entonces *nctrl_por_curvatura* propone tamaños menores y la histéresis (32) impone dos reglas: no crecer nunca y reducir como mucho un salto limitado por porcentaje y por un mínimo absoluto. Por eso la gráfica baja en escalones regulares, sin rebotes, hasta alcanzar la cota mínima operativa. En términos prácticos, esta figura confirma que el Bloque D no recorta a ciegas: espera a que la pasada esté limpia, reduce de

forma controlada y entrega una malla de puntos cada vez más simple pero suficiente para conservar la forma.

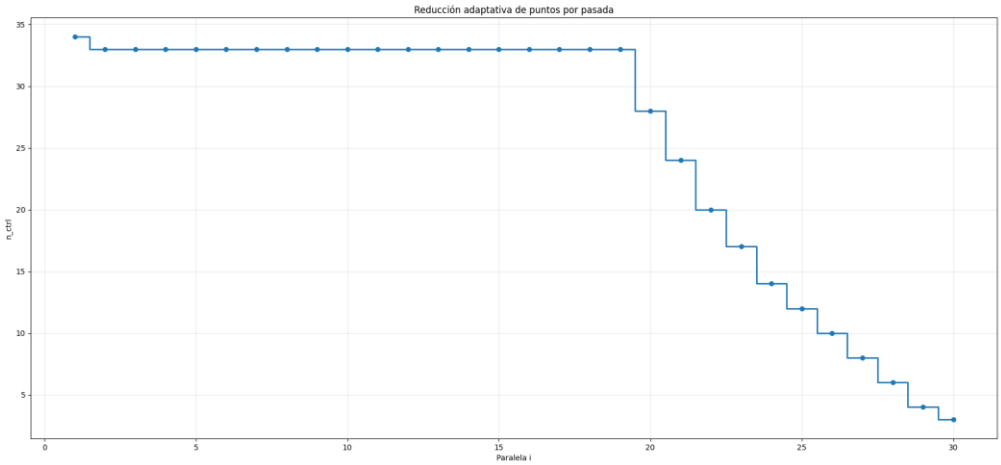

Figura 3.30. Reducción adaptativa de puntos por pasada. Evolución del número de puntos de control n_ctrl a lo largo de todas las paralelas (eje X). La curva en escalones refleja la combinación de la política curvatura/tamaño.

3.8 Remallado equiespaciado – Bloque E ($Q^{red} \rightarrow R$)

3.8.1 Descripción del algoritmo

Al terminar los bloques C (enderezado) y D (reducción adaptativa), con la malla reducida Q^{red} la distribución de puntos puede quedar descompensada: muchos puntos apretados en zonas con curvatura cerrada y otros muy separados en tramos suaves. Esa distribución irregular es mala para el spline porque puede producir solape o incluso cruces entre puntos. El Bloque E corrige esto sin cambiar la forma global.

Vuelve a muestrear por longitud de arco para obtener una malla de N puntos equiespaciados a lo largo de la trayectoria, fijando los extremos prácticamente igual que en el Bloque A, pero sin reducir el número de puntos. El resultado es una base homogénea y estable para la construcción del spline del Bloque F.

En la **Figura 3.31** muestra el diagrama de flujo del Bloque E. El proceso comienza recibiendo la polilínea y, si no se especifica tamaño, adopta el número actual de nodos. Se comprueban dos casos límite para devolver la curva sin cambios: que haya menos de tres puntos (no hay interior que redistribuir) o que la longitud total sea prácticamente nula (evita normalizar sobre cero). En el caso general se calcula la longitud de arco acumulada y con ella se crea un nuevo eje de distancias equiespaciadas desde 0 hasta la longitud total. Las coordenadas x e y se reconstruyen por interpolación lineal a lo largo de ese eje, y finalmente se reafirman los anclajes copiando exactamente los extremos originales.

El resultado conserva la misma forma, pero con nodos homogéneos por arco, lo que estabiliza el spline del Bloque F al evitar sobre-muestreos locales y mejorar la regularidad de tangentes y curvaturas. El coste es lineal en el número de puntos y la implementación es robusta al basarse únicamente en distancias euclídeas e interpolación por tramos.

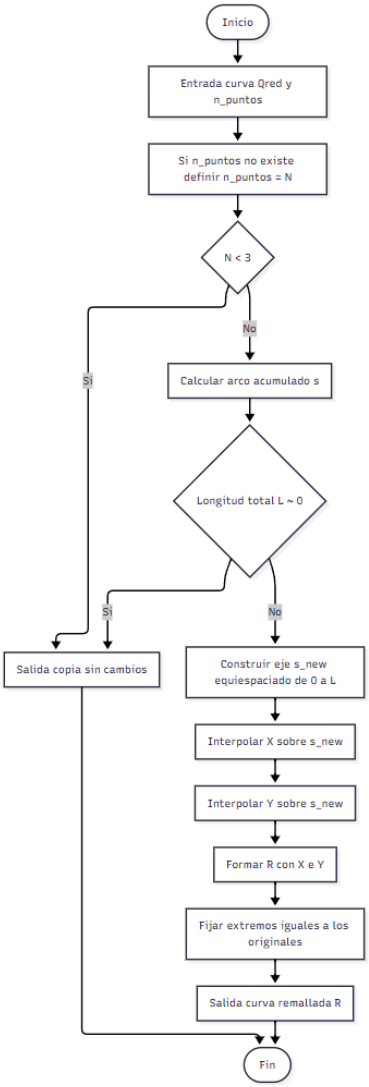

Figura 3.31. Diagrama de flujo del remallado por arco. A partir de la curva enderezada Q^{red} y del tamaño deseado, el algoritmo reparametriza por longitud de arco, genera un eje equiespaciado, interpola las coordenadas sobre ese eje y fija los extremos, devolviendo una curva con los puntos uniformemente distribuidos.

3.8.2 Matemática asociada

Tras el enderezado y la reducción, partimos de la polilínea tras el Bloque D tal que:

$$\boldsymbol{Q}^{red} = \{\boldsymbol{q}_i^{red}\}_{i=0}^{N-1} \quad N = n_{ctrl} \tag{34}$$

Construimos la longitud de arco acumula tal como hacemos en el Bloque A (11):

$$s_0 = 0, \quad s_i = \sum_{k=1}^{i} \|\boldsymbol{q}_k - \boldsymbol{q}_{k-1}\|, \quad i = 1, \dots, N-1 \tag{35}$$

Donde s_i mide cuanta distancia real llevamos recorrida sobre la polilínea al llegar a $\boldsymbol{q}_i$ y obtenemos la longitud total $L = s_{N-1}$

Tras esto normalizamos la distancia recorrida a [0, 1]:

$$t_i = \frac{s_i}{L},\ i = 0, \dots, N-1 \quad (t_0 = 0, t_{N-1} = 1) \tag{36}$$

Ahora elegimos N parámetros equiespaciados en el intervalo para que cada nuevo nodo este a la misma distancia que el anterior:

$$s_j^{new} = \frac{j}{N-1}, \quad j = 0, \dots, N-1 \tag{37}$$

Por último, solo nos queda definir la nueva malla $\boldsymbol{R} = \{\boldsymbol{r}_j\}_{j=0}^{N-1}$ evaluando la polilínea $\boldsymbol{Q}^{red}$ en s_j^{new} mediante la interpolación lineal a trozos sobre:

$$\boldsymbol{r}_j = \left(interp\left(t,\ \boldsymbol{q}_x,\ s_j^{new}\right),\ interp\left(t,\ \boldsymbol{q}_y,\ s_j^{new}\right)\right)\ \ \boldsymbol{R}_0 = \boldsymbol{q}_0^{red},\ \ R_{N-1} = \boldsymbol{q}_{N-1}^{red} \tag{38}$$

Como resultado obtenemos $\boldsymbol{R}$ que tiene la misma cantidad de puntos que $\boldsymbol{Q}^{red}$, pero distribuidos homogéneamente por arco, estabilizando el spline que vamos a construir a continuación

3.8.3 Explicación del código

En la **Figura 3.32** muestra la función principal *remallar_equispaciado* aplica el remallado propiamente dicho. Lo primero que hace es copiar la polilínea y tratar los casos degenerados: si hay menos de tres puntos, devuelve una copia sin tocar porque no hay interior que redistribuir; si *n_puntos* no se especifica, conserva el tamaño actual para "solo reordenar" por arco; y si la longitud total L que devuelve *_acum_arco* es prácticamente nula, también devuelve la curva sin cambios para evitar una normalización sobre cero. Con la curva válida, toma el s acumulado, extrae la longitud total y construye un nuevo eje de arco equiespaciado *s_new* (37) con tantos nodos como *n_puntos*. A continuación, interpola independientemente las coordenadas x e y sobre ese eje: *np.interp* [16] se usa aquí como interpolación lineal a trozos sobre la polilínea, que equivale a "caminar" por la curva y colocar un punto cada Δs constante.

Finalmente, ancla los extremos asignando *Q[0]=P[0] y Q[-1]=P[-1]*: esto asegura que el inicio y el final, que actúan como referencias geométricas y de rumbo, se conserven exactamente. El resultado **R** (38) tiene el mismo número de puntos que se pidió, pero ahora equiespaciados por arco, lo que estabiliza la construcción del spline del Bloque F

```python
def remallar_equispaciado(P, n_puntos=None):

    P = np.asarray(P, float)
    if len(P) < 3:
        return P.copy()
    if n_puntos is None:
        n_puntos = len(P)

    s = _acum_arco(P)
    L = s[-1]
    if L < 1e-12:
        return P.copy()

    s_new = np.linspace(0.0, L, n_puntos)
    x = np.interp(s_new, s, P[:,0])
    y = np.interp(s_new, s, P[:,1])
    Q = np.column_stack([x, y])

    Q[0]  = P[0]
    Q[-1] = P[-1]
    return Q
```

Figura 3.32. Implementación de remallar_equiespaciado. La rutina reparametriza la polilínea por longitud de arco: calcula el arco acumulado (_acum_arco), genera un eje equiespaciado s_new, interpola por tramos las coordenadas x e y sobre ese eje y fija los extremos para conservar los anclajes. Si la curva es demasiado corta o tiene menos de tres puntos, devuelve una copia sin cambios.

La función auxiliar *_acum_arco* en la **Figura 3.33** calcula la longitud de arco acumulada de la polilínea **P**. Recorre la lista de puntos y, para cada índice, suma la distancia euclídea al punto anterior. El resultado es un vector *s* que empieza en cero y termina en la longitud total; esto es la base para volver a parametrizar por arco sin cambiar la forma.

```python
def _acum_arco(P):
    s = np.zeros(len(P))
    for i in range(1, len(P)):
        s[i] = s[i-1] + np.linalg.norm(P[i]-P[i-1])
    return s
```

Figura 3.33. *Función auxiliar _acum_arco. Recorre la polilínea sumando la distancia entre puntos consecutivos para construir el vector de longitud de arco acumulada, base del remallado por arco del Bloque E.*

3.8.4 Resultado del bloque

La **Figura 3.34** ilustra que el Bloque E no cambia la forma, solo reparametriza la pasada. Tras el enderezado y la reducción, los puntos pueden estar concentrados en algunas zonas y escasos en otras. El remallado vuelve a distribuir el mismo número de puntos a lo largo de la trayectoria. Por eso las dos curvas se superponen casi por completo, mientras que los marcadores azules aparecen

regularmente espaciados a lo largo de toda la trayectoria. Este reparto estabiliza el spline del Bloque F, evita que haya tramos donde se solapen o se crucen puntos vecinos tras varias paralelas.

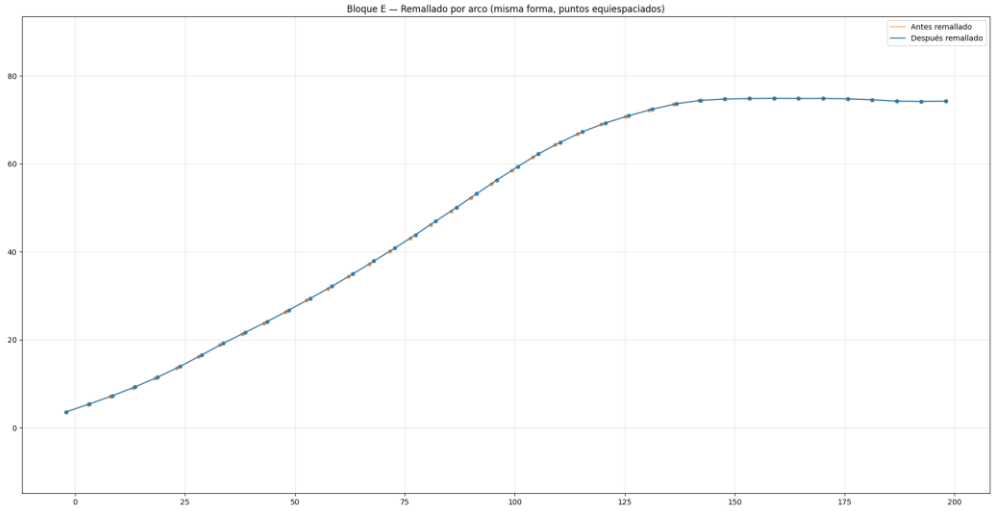

Figura 3.34. Efecto del remallado por arco. En naranja se muestran los puntos y la polilínea antes del remallado; en azul, después. La geometría es la misma, pero los nodos quedan equiespaciados por longitud de arco y se mantienen los extremos anclados.

3.9 Construcción del spline cúbico – Bloque F ($\boldsymbol{R \rightarrow S(t)}$)

3.9.1 Descripción del algoritmo

A partir de la malla $\boldsymbol{R}$ obtenida en el Bloque E, este bloque construye la trayectoria suave de guiado como un spline cúbico paramétrico que interpola exactamente por esos puntos. El resultado es una curva paramétrica suave C^2 estable y fácil de muestrear, que respeta los puntos de control dejando una ruta suave y continua lista para el guiado.

El diagrama de flujo **Figura 3.35** muestra como el proceso parte de los puntos remallados **R**. Si hay menos de dos, no puede construirse un spline y se devuelve una copia sin cambios. En caso contrario, se calcula la longitud de arco acumulada y se normaliza a un parámetro $t \in [0,1]$. Con ese parámetro se interpolan por separado x e y mediante splines cúbicos obteniendo la curva paramétrica $\boldsymbol{S}(t) = (Sx(t), Sy(t))$. Finalmente se crea una malla uniforme de evaluación y se evalúa $\boldsymbol{S}$ en esos nodos para producir la trayectoria densa, que es la que se entrega al guiado.

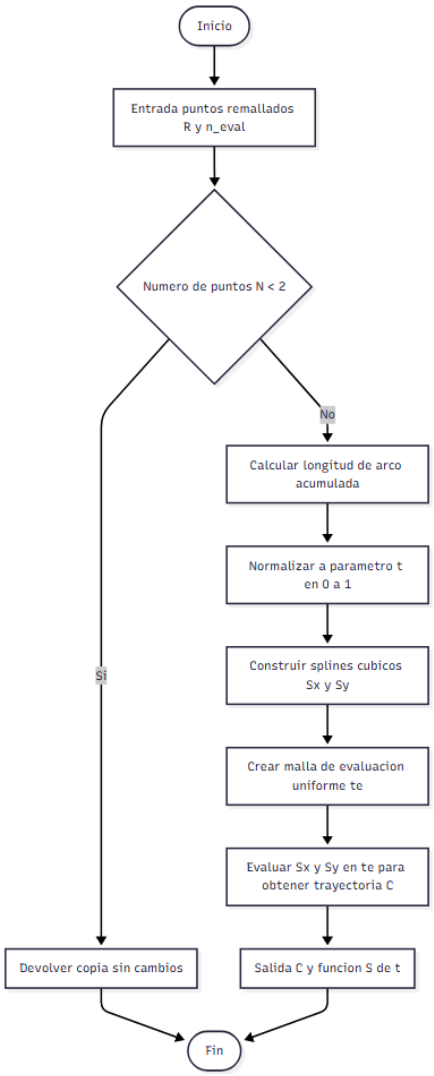

Figura 3.35. Diagrama de flujo de la construcción del spline cúbico. A partir de los puntos remallados $\boldsymbol{R}$ *y del tamaño de evaluación, se parametriza por arco, se construyen los splines y se obtiene la trayectoria densa* C.

3.9.2 Matemática asociada

Primero antes de construir el spline debemos calcular la longitud de arco como hicimos anteriormente para los puntos nuevos $\boldsymbol{r}_i = (x_i, y_i)$:

$$s_0 = 0, \quad s_i = \sum_{k=1}^{i} \|\boldsymbol{r}_k - \boldsymbol{r}_{k-1}\|, \quad i = 1, \dots, N-1 \tag{39}$$

Y como antes calculamos la longitud total $L = s_{N-1}$ y normalizamos:

$$t_i = \frac{s_i}{L}, \; i = 0, \dots, N-1 \quad (t_0 = 0, t_{N-1} = 1) \tag{40}$$

Ahora ya podemos construir los splines cúbicos que necesitamos para obtener la trayectoria:

$$x(t) = S_x(t), \; y(t) = S_x(t) \tag{41}$$

Ya con los splines podemos definir la curva paramétrica:

$$\boldsymbol{S}(t) = \left(S_x(t), S_y(t)\right) \quad t \in [0,1] \tag{42}$$

Por último, solo nos queda representar la nueva trayectoria. Para ello primero vamos a definir los puntos en los que evaluaremos el spline según un número de puntos de evaluación:

$$t_j = \frac{j}{n_{eval}-1}, \qquad j = 0, \dots, n_{eval} - 1 \tag{43}$$

Con esto obtenemos la trayectoria final:

$$C = \{\boldsymbol{S}(t)\}_{j=0}^{n_{eval}-1} \tag{44}$$

3.9.3 Explicación del código

La función de la **Figura 3.36** recibe la malla remallada $\boldsymbol{R}$ y devuelve una trayectoria densa y suave lista para el guiado. Lo primero que hace es proteger casos degenerados: si entran menos de dos puntos, no hay nada que interpolar y devuelve una copia. A continuación, construye el parámetro por arco. Recorre los puntos acumulando distancias entre consecutivos y normaliza por la longitud total; el resultado es un vector t monótono en [0,1], lo que evita concentraciones de curvatura por una parametrización mala.

Con ese t crea dos splines cúbicos interpolantes, uno para x y otro para y (41), usando *scipy.interpolate.CubicSpline* [17]. El spline es C^2en los nudos interiores y ajusta suavemente las dos primeras celdas en los extremos, lo que en la práctica reduce oscilaciones spurious sin obligarnos a estimar tangentes.

Por último, fija una malla de evaluación (43) uniforme y evalúa ambos splines ahí. Esa cuadrícula uniforme en *[0,1]*, al estar el parámetro ligado al arco, produce una nube de puntos regular a lo largo de la curva. La función devuelve la matriz $C = \{(S_x(t_j), S_y(t_j)\}$ (42) (44) que es la trayectoria final.

```
def spline_desde_puntos(puntos, n_eval=200):

    p = np.asarray(puntos, float)
    if len(p) < 2: return p.copy()
    t = np.zeros(len(p))
    for i in range(1, len(p)):
        t[i] = t[i-1] + np.linalg.norm(p[i]-p[i-1])
    t = t/(t[-1]+1e-12)
    csx = CubicSpline(t, p[:,0]); csy = CubicSpline(t, p[:,1])
    te = np.linspace(0, 1, n_eval)
    return np.column_stack([csx(te), csy(te)])
```

Figura 3.36. Implementación de spline_desde_puntos. La función calcula el parámetro por longitud de arco normalizada, construye dos CubicSpline (en x e y) y

evalúa el spline en una malla uniforme de t para devolver la trayectoria densa de guiado.

La función de la **Figura 3.37** es la función principal del proceso completo y, en particular, es donde se ejecuta el Bloque D dentro del bucle de pasadas. Comienza creando una base de control equiespaciada por arco con *n_ctrl_max* de puntos y calculando sus normales; sobre esa base fija una referencia de curvatura (33) (*K_REF*) que se usará para comparar la complejidad de cada pasada. A continuación, itera tantas veces como paralelas se quieran generar. En cada iteración desplaza la base por sus normales la distancia fijada (Bloque B) y, salvo que se desactive para la primera, endereza localmente con *aplicar_enderezado_local_gs* (Bloque C); el número de iteraciones del enderezado y el peso hacia la recta (wR) se hacen crecer con el progreso de pasadas para ir forzando la rectitud cuando la geometría ya está limpia.

Con la polilínea enderezada de esa iteración, calcula curvatura media y máxima y, con ellas, decide el número objetivo de puntos de control mediante *nctrl_por_curvatura*: si la curva aún gira, conserva más puntos; si ya es dócil, reduce. Acto seguido aplica la histéresis: impide que el número crezca respecto a la pasada anterior y limita la caída máxima permitida (por porcentaje y por mínimo absoluto), de modo que el tamaño de la malla descienda de forma estable y predecible. Con el tamaño final, remalla por arco a exactamente esos puntos (Bloque E), anclando extremos para estabilidad, y luego reconstruye un spline cúbico de evaluación para visualizar y registrar la pasada (Bloque F). nodos de control de cada pasada y las métricas registradas, dejando listo el informe y las figuras del capítulo.

```python
def generar_paralelas_adaptativas(curva_original: np.ndarray, cantidad: int, direccion: int, SM: float, STM: float, sep: float,
        n_ctrl_max: int,n_ctrl_min: int = 3,w_curv: float = 0.7, beta_curv: float = 1.6,frac_drop_max: float = 0.15, drop_min_abs: int = 2,
        encadenado: bool = True, endereza_primera: bool = True
):
    t0 = time.time()

    puntos_control_base = muestrear_equiespaciado_arco(curva_original, n_ctrl_max)
    puntos_control_original = puntos_control_base.copy()
    normales_base = calcular_normales_puntos(puntos_control_base)
    paralelas, ctrl_paralelas = [], []
    puntos_desplazados_lista, puntos_enderezados_lista = [], []

    tiempos_por_paralela = []
    n_ctrl_hist = []

    kmean_ref, kmax_ref = metricas_curvatura(puntos_control_base)
    K_REF = max(1e-6, 0.5*(kmean_ref + kmax_ref))

    n_ctrl_prev = n_ctrl_max
    for i in range(1, cantidad+1):
        t_loop_ini = time.perf_counter()
        ratio = (i-1)/(cantidad-1) if cantidad > 1 else 0.0
        distancia = sep*direccion

        puntos_desplazados = puntos_control_base + distancia*normales_base

        peso_recta = 5000.0*ratio
        num_iter = int(25 + (60 - 25)*(ratio**1.2))

        if not endereza_primera and i == 1:
            puntos_enderezados = puntos_desplazados.copy()
        else:
            puntos_enderezados = aplicar_enderezado_local_gs(puntos_desplazados, SM, STM,num_iter=num_iter,
                wL_ini=2.0,  wL_fin=12.0,wB_ini=0.6,  wB_fin=0.03,wA_ini=1.2,  wA_fin=16.0,wR=peso_recta,tol_mean=1e-4)
            puntos_enderezados[0]  = puntos_desplazados[0]
            puntos_enderezados[-1] = puntos_desplazados[-1]
        kmean_i, kmax_i = metricas_curvatura(puntos_enderezados)

        n_ctrl_i = nctrl_por_curvatura(kmean_i, kmax_i, K_REF, n_ctrl_min, n_ctrl_max, w=w_curv, beta=beta_curv)
        n_ctrl_i = min(n_ctrl_i, n_ctrl_prev)
        drop_allowed = max(drop_min_abs, int(round(frac_drop_max * n_ctrl_prev)))
        n_ctrl_i = max(n_ctrl_i, n_ctrl_prev - drop_allowed)
        n_ctrl_prev = n_ctrl_i

        puntos_enderezados = remallar_equispaciado(puntos_enderezados, n_puntos=n_ctrl_i)
        curva_paralela = spline_desde_puntos(puntos_enderezados, 200)
        paralelas.append(curva_paralela)
        ctrl_paralelas.append(puntos_enderezados)
        puntos_desplazados_lista.append(puntos_desplazados)
        puntos_enderezados_lista.append(puntos_enderezados)
        n_ctrl_hist.append(int(n_ctrl_i))
        t_loop = time.perf_counter() - t_loop_ini
        tiempos_por_paralela.append(t_loop)
        print(f"Paralela {i}/{cantidad}: n_ctrl={n_ctrl_i} | " f"k_mean={kmean_i:.4e} | k_max={kmax_i:.4e} | tiempo={t_loop:.3f} s")
        if encadenado:
            puntos_control_base = puntos_enderezados.copy()
            normales_base = calcular_normales_puntos(puntos_control_base)
    tiempo_total = time.time() - t0
    return {
        "paralelas": paralelas, "ctrl_paralelas": ctrl_paralelas, "puntos_desplazados_lista": puntos_desplazados_lista,
        "puntos_enderezados_lista": puntos_enderezados_lista, "puntos_control_original": puntos_control_original, "tiempo_total": tiempo_total,
        "tiempos_por_paralela": np.array(tiempos_por_paralela, float),"n_ctrl_hist": np.array(n_ctrl_hist, int),
    }
```

Figura 3.37. Función generar_paralelas_adaptativas. Orquesta el flujo de los Bloques B→C→D→E→F: desplazamiento por normales, enderezado local, medición de curvatura y asignación adaptativa de puntos con histéresis, remallado

por arco y reconstrucción spline; todo ello repetido pasada a pasada y con opción de encadenado.

3.9.4 Resultados del bloque

La imagen de la **Figura 3.38** compara la linde original con la primera pasada paralela. Los marcadores morados son los nodos obtenidos tras el enderezado (C), la reducción adaptativa (D) y el remallado por arco (E). El trazo azul es el resultado del Bloque F: el spline cúbico paramétrico construido sobre parámetro de arco normalizado. Se aprecia que el spline pasa exactamente por cada punto de control y "planchan". En los tramos casi rectos ambas trayectorias prácticamente coinciden; en las zonas de mayor giro, el spline suaviza la transición manteniendo el rumbo y la separación respecto a la original, dejando una trayectoria apta para el guiado.

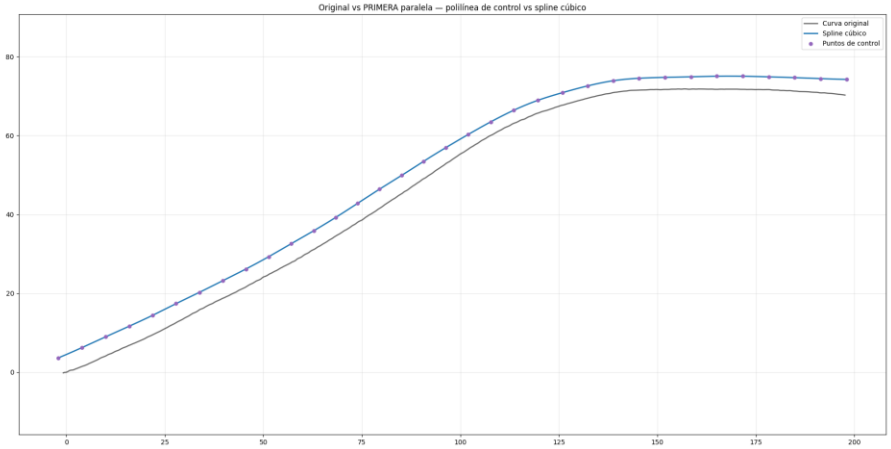

Figura 3.38. Curva original (negro) frente a la primera paralela generada tras C–D–E: se muestran los puntos de control (morado) y el spline cúbico que los interpola (azul).

3.10 Funciones auxiliares

En este apartado reunimos las funciones auxiliares que sostienen el flujo de trabajo, pero no constituyen, por sí mismas, un bloque del algoritmo (A-B-C–D–E–F).

3.10.1 Pedir parametros

Esta función de la **Figura 3.39** crea una pequeña interfaz modal con *Tkinter*. Primero oculta la ventana raíz *(root.withdraw)*, y a continuación abre seis cuadros de diálogo con *simpledialog* para leer, con validación mínima, los parámetros que alimentan al pipeline:

- cantidad (entero ≥ 1): cuántas paralelas generar.
- dirección (entero, típico +1 derecha, −1 izquierda): signo del desplazamiento.
- SM y STM (reales ≥ 0.1): solape y zona máximos sin tratar, que acotan los desplazamientos en el enderezado.
- sep (real ≥ 0.1): separación entre pasadas
- n_ctrl (entero ≥ 5): tamaño inicial de la malla de control.

Cada cuadro fija un valor inicial (*initialvalue*) cómodo para pruebas y un mínimo (*minvalue*) para evitar entradas vacías o no físicas. Al terminar, destruye la ventana (*root.destroy*) y devuelve una

tupla con los seis valores. Es una utilidad de entrada/salida: no altera la geometría ni la lógica del algoritmo.

```python
def pedir_parametros_usuario():
    root = tk.Tk(); root.withdraw()
    cantidad = simpledialog.askinteger("Parametro", "Numero de paralelas a generar:", minvalue=1, initialvalue=30)
    direccion = simpledialog.askinteger("Parametro", "Direccion (+1 = derecha, -1 = izquierda):", initialvalue=1)
    SM = simpledialog.askfloat("Parametro", "Solape maximo (metros):", minvalue=0.1, initialvalue=1.0)
    STM = simpledialog.askfloat("Parametro", "Zona maxima sin tratar (metros):", minvalue=0.1, initialvalue=1.0)
    sep = simpledialog.askfloat("Parametro", "Distancia entre paralelas (metros):", minvalue=0.1, initialvalue=4.0)
    n_ctrl = simpledialog.askinteger("Parametro", "Puntos de control del spline:", minvalue=5, initialvalue=30)
    root.destroy()
    return cantidad, direccion, SM, STM, sep, n_ctrl
```

Figura 3.39. Función pedir_parametros_usuario. Diálogo ligero en Tkinter para recoger los parámetros de ejecución: nº de paralelas, dirección, límites agronómicos (SM/STM), separación entre pasadas y nº inicial de puntos de control.

3.10.2 Cargar trayectoria

En la **Figura 3.40** se muestra como la rutina abre el archivo con codificación *UTF-8* y *errors="ignore"* para tolerar caracteres ajenos a ASCII. Descarta la primera línea ya que es una cabecera y recorre el resto; para cada línea elimina espacios (*strip()*), separa por el punto y coma (*split(';')*) y convierte cada campo a *float* tras sustituir la coma decimal por punto (*replace(',', '.')*).

Con esa comprensión se construye una lista de pares (x, y) que finalmente se convierte a *numpy* con tipo float. El resultado es una polilínea tal cual viene del fichero, lista para pasar al muestreo por arco y al resto de bloques.

```python
def cargar_curva(path_txt: str):
    with open(path_txt, "r", encoding="utf-8", errors="ignore") as f:
        L = f.readlines()[1:]
        puntos = [(float(a.replace(',', '.')), float(b.replace(',', '.')))
                  for a, b in (l.strip().split(';') for l in L)]
    return np.array(puntos, float)
```

Figura 3.40. Función cargar_curva. Lectura robusta de un fichero .txt con coordenadas en formato x;y (cabecera en la primera línea). Normaliza comas decimales a punto y devuelve un np.array de tamaño N×2N en coma flotante.

3.10.3 Plot de los resultados

La **Figura 3.41** es la función que va a graficar los resultados obtenidos de nuestro algoritmo. Dibuja la linde original en negro y superpone los puntos de control iniciales en rojo. Para cada pasada, toma de resultados los puntos desplazados por la normal y los enderezados y, según los flags, los pinta como nubes (naranja y azul, respectivamente). Además, traza la trayectoria final de esa pasada (el spline) como una línea azul clara. Fija la misma escala en ambos ejes, añade cuadrícula suave y ajusta márgenes. Con una sola figura se ve, de un vistazo, el antes (original), el proceso (desplazamiento y enderezado) y el resultado (spline) de todas las pasadas.

```python
def plot_resultados(curva_original, resultados, titulo_base: str,
                    pintar_despl=True, pintar_ender=True, s_pts=12):
    plt.figure(figsize=(18, 9))
    plt.plot(curva_original[:,0], curva_original[:,1], 'k-', lw=2, label="Curva original")
    pco = resultados["puntos_control_original"]
    plt.scatter(pco[:,0], pco[:,1], c='red', s=45, label="Puntos control orig.", zorder=5)

    paralelas = resultados["paralelas"]
    PDL = resultados["puntos_desplazados_lista"]
    PEL = resultados["puntos_enderezados_lista"]

    for i, par in enumerate(paralelas):
        if pintar_ender:
            plt.scatter(PEL[i][:,0], PEL[i][:,1], color='blue', s=s_pts, alpha=0.75,
                        label="Puntos enderezados" if i == 0 else "")
        plt.plot(par[:,0], par[:,1], color='royalblue', alpha=0.25, lw=1.0,
                 label="Paralelas" if i == 0 else "")

    plt.title(titulo_base)
    plt.axis('equal'); plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()
```

Figura 3.41. Función plot_resultados: visualización comparada de la curva original, los puntos de control, los puntos desplazados y enderezados de cada passada y los splines finales que construyen las paralelas.

3.10.4 Plot de las métricas

La **Figura 3.42** es la función que va a graficar el número de puntos usado en cada pasada y lo dibuja frente al índice de paralela. Los ejes se rotulan con la paralela i y n_{ctrl}, se activa cuadrícula suave y *tight_layout* ajusta márgenes. El resultado permite ver de un vistazo la reducción escalonada de puntos a medida que la trayectoria se va enderezando.

```python
def plot_metricas(resultados):
    tiempos = resultados["tiempos_por_paralela"]
    nctrl = resultados["n_ctrl_hist"]
    idx = np.arange(1, len(nctrl)+1)

    plt.figure(figsize=(12, 4.5))
    plt.step(idx, nctrl, where='mid', lw=2.0)
    plt.scatter(idx, nctrl, s=35, zorder=3)
    plt.xlabel("Paralela (i)")
    plt.ylabel("n puntos de control (n_ctrl)")
    plt.title("Reduccion adaptativa de puntos de control por paralela")
    plt.grid(True, alpha=0.3); plt.tight_layout(); plt.show()
```

Figura 3.42. Función plot_metricas: evolución del número de puntos de control por pasada.

3.10.5 Main

La **Figura 3.43** es la función *main*. Este main actúa como "lanzador" de la demostración. Primero carga la linde desde el archivo introducido y pide por GUI los parámetros operativos (número de pasadas, dirección, SM, STM, separación y tamaño inicial de malla).

A continuación, fija la configuración de la reducción adaptativa por curvatura: cotas de puntos (N_CTRL_MAX, N_CTRL_MIN), los parámetros del planificador (W_CURV=0.6, BETA_CURV=1.6) y la histéresis de caída para límitar la reducción máxima por pasda (FRAC_DROP_MAX, DROP_MIN_ABS). Con todo ello llama a *generar_paralelas_adaptativas*, que orquesta los bloques A-B–C–D–E–F sobre cada pasada; después imprime el tiempo total y dibuja, con *plot_resultados* y *plot_metricas*, la geometría final y la evolución de n_{ctrl}. En suma, este

punto concentra la ejecución de extremo a extremo y deja listos los gráficos que documentan el comportamiento del algoritmo con los parámetros seleccionados.

```python
def main():
    curva_original = cargar_curva("tr1_linde.txt")
    print(f"Cargados {len(curva_original)} puntos de la curva original.")

    cantidad, direccion, SM, STM, sep, n_ctrl_gui = pedir_parametros_usuario()
    print(f"Cantidad={cantidad} | sep={sep} | dir={direccion} | SM={SM} | STM={STM} | n_ctrl={n_ctrl_gui}")

    N_CTRL_MAX = n_ctrl_gui
    N_CTRL_MIN = 2
    W_CURV     = 0.6
    BETA_CURV  = 1.6
    FRAC_DROP_MAX = 0.15
    DROP_MIN_ABS  = 2
    ENCADENADO = True
    ENDEREZA_PRIMERA = True

    resultados = generar_paralelas_adaptativas(curva_original=curva_original,cantidad=cantidad, direccion=direccion, SM=SM, STM=STM, sep=sep,
        n_ctrl_max=N_CTRL_MAX,n_ctrl_min=N_CTRL_MIN,w_curv=W_CURV, beta_curv=BETA_CURV,
        frac_drop_max=FRAC_DROP_MAX, drop_min_abs=DROP_MIN_ABS, encadenado=ENCADENADO, endereza_primera=ENDEREZA_PRIMERA
    )

    print(f"\nTiempo total: {resultados['tiempo_total']:.2f} s")

    titulo = (f"Paralelas con enderezado encadenado (GS local + remallado adaptativo por CURVATURA, sin EMA) | "
              f"{cantidad} paralelas | sep={sep} m | SM={SM} m | STM={STM} m | "
              f"Nmin={N_CTRL_MIN}, Nmax={N_CTRL_MAX}, "
              f"w={W_CURV}, beta={BETA_CURV}, drop≤max({DROP_MIN_ABS}, {FRAC_DROP_MAX:.0%}·n_prev)")
    plot_resultados(curva_original, resultados, titulo_base=titulo)
    plot_metricas(resultados)
```

Figura 3.43. Función main(): lectura de la trayectoria, captura de parámetros, configuración de la reducción por curvatura y ejecución del flujo de trabajo completo, con generación de gráficos e informe de tiempos.

Capítulo 4:
Resultados obtenidos tras la aplicación del algoritmo

Capítulo 4: Resultados obtenidos tras la aplicación del algoritmo

Dentro de este capítulo se van a mostrar los resultados obtenidos tras la aplicación del algoritmo presentado y desarrollado en el capítulo anterior. Estos resultados son los obtenidos tras aplicar el código que se ha explicado a cuatro trayectorias reales recogidas de una finca.

Para estas simulaciones hemos configurado unos parámetros por defecto:

- N.º paralelas: 30
- Distancia entre paralelas: 4
- Zona sin tratar máximo: 1
- Solape máximo: 1
- Iteraciones: 60
- Puntos de control: 30

4.1 Curva 1

Como podemos observar en la figura 4.1 disponemos de una trayectoria simple con una única curva suave.

Aplicando el algoritmo a esta trayectoria con los parámetros por defecto obtenemos la **Figura 4.2** donde podemos observar que primero pasamos de una trayectoria que se define por miles de puntos a únicamente 30 puntos con los que el algoritmo trabaja. Se puede apreciar como poco a poco la curvatura va desapareciendo y cuando llegamos al último tercio de la trayectoria vemos como se realiza la reducción de puntos y esto consigue que consigamos el enderezado total en la paralela nº 23. En la **Figura 4.3** se puede observar una gráfica que indica como se van reduciendo los puntos en cada trayectoria y se aprecia al principio un pequeño escalón para la primera paralela y luego tenemos una zona plana donde principalmente se está aplicando el enderezado, a partir de la paralela nº 20 vemos como se vuelve a reducir el número de puntos hasta alcanzar el número mínimo cuando ya hemos alcanzado la recta por completo.

También podemos ver el resultado de aplicar el mismo algoritmo, pero en la dirección contraria en la **Figura 4.4** donde tenemos un resultado casi idéntico donde se aprecia poco a poco como va desapareciendo la curvatura y en el mismo caso que anteriormente obtenemos el enderezado total en la paralela nº 23.

En la **Figura 4.4** se han modificado los parámetros para la curva indicando un solape y una zona sin tratar de 0.25. Se puede observar cómo tenemos una pequeña desviación que produce un solape total entre las paralelas nº32 y nº33, debido a la reducción de puntos debido a que el algoritmo está detectando una curva muy suave en la que es posible eliminar algún punto y al ser un punto crítico tras solo quedar cinco únicos puntos siendo dos de ellos los extremos. Al eliminar este punto crítico obtenemos un desvío y como solución vamos a limitar a cinco el número mínimo de puntos que puede reducir el bloque de reducción de puntos. Podemos ver en la **Figura 4.5** como ya tenemos un buen resultado y aunque hemos necesitado más paralelas hemos resuelto este problema.

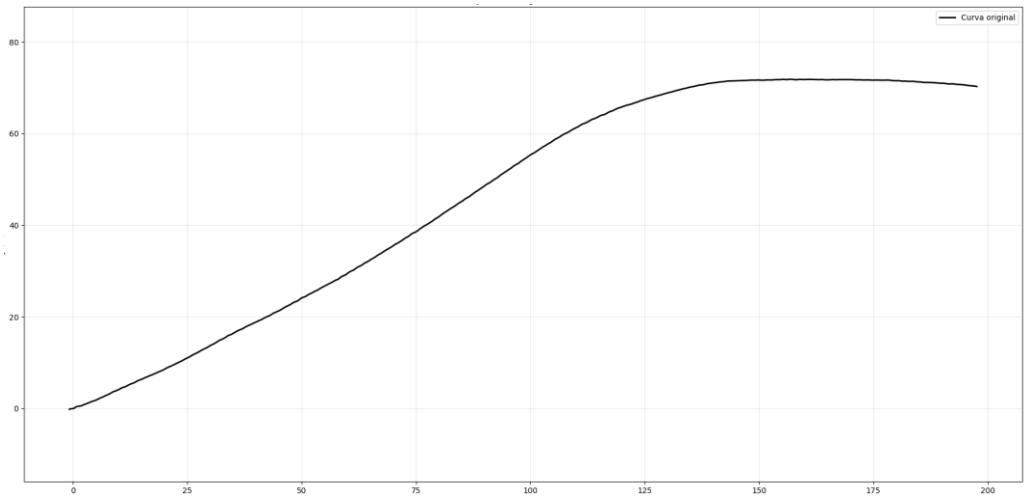

Figura 4.1. Representación gráfica de la curva 1.

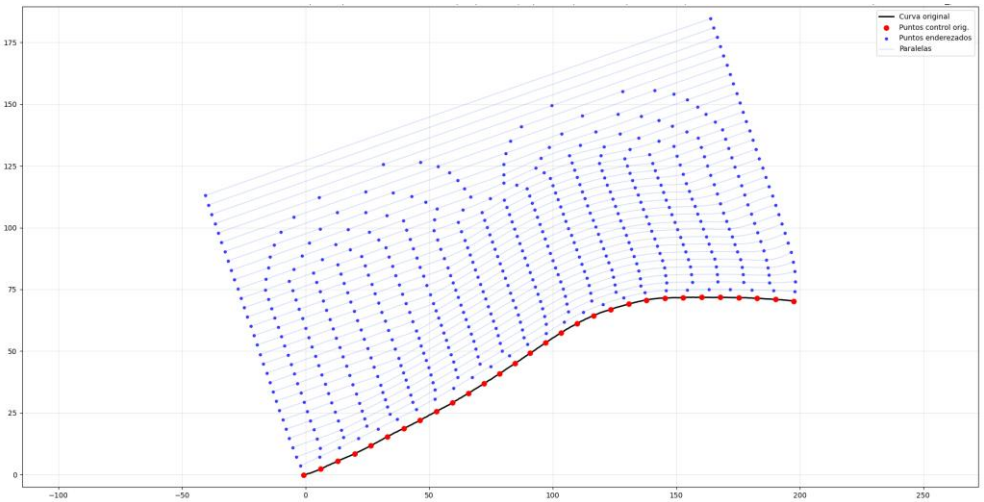

Figura 4.2. Resultado de la curva 1 en dirección positva tras aplcar el algoritmo con los parámetros por defecto.

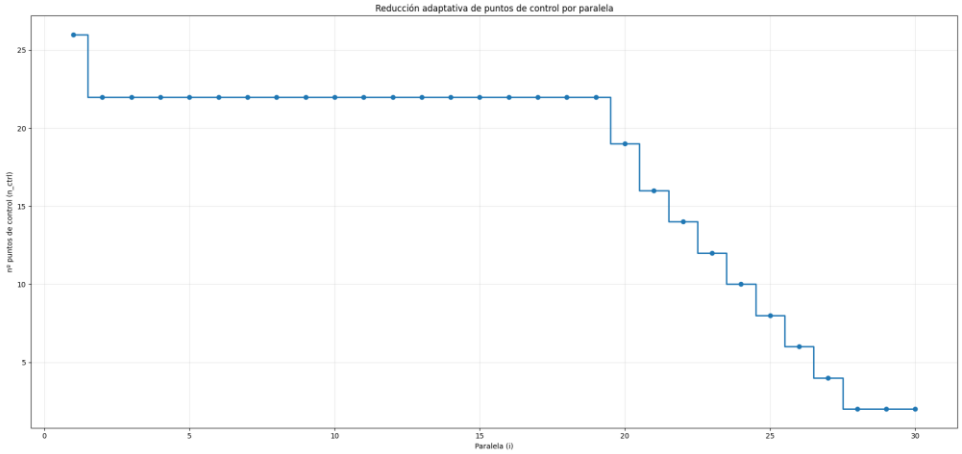

Figura 4.3. Gráfica que muestra la reducción de puntos que se va realizando a lo largo de las paralelas en la curva 1.

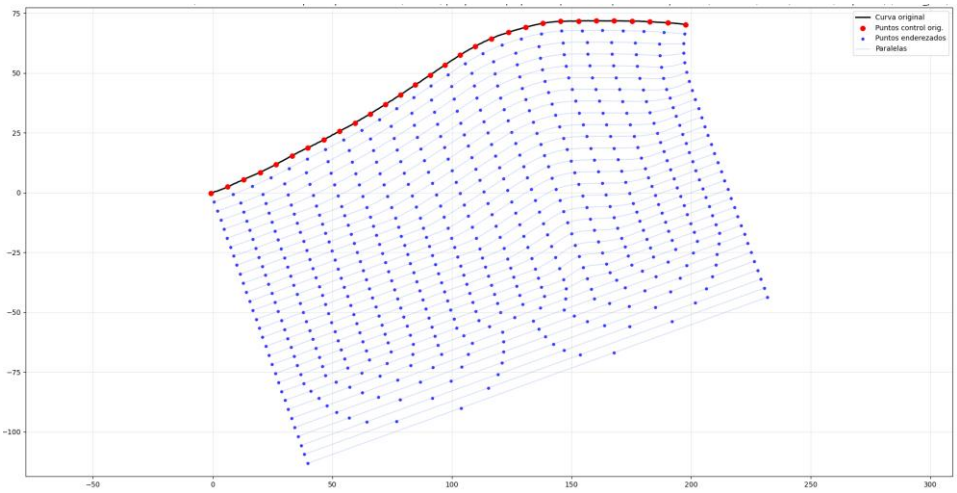

Figura 4.4. Resultado de la curva 1 en dirección negativa tras aplicar el algoritmo con los parámetros por defecto.

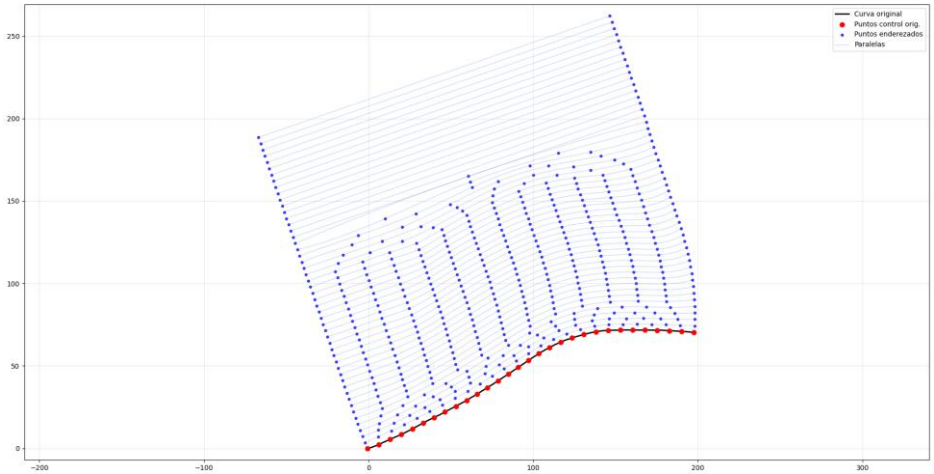

Figura 4.5. Representación de la Curva 1 con SM=0.50, STM=0.50 y 50 paralelas. Se puede observar un solape total.

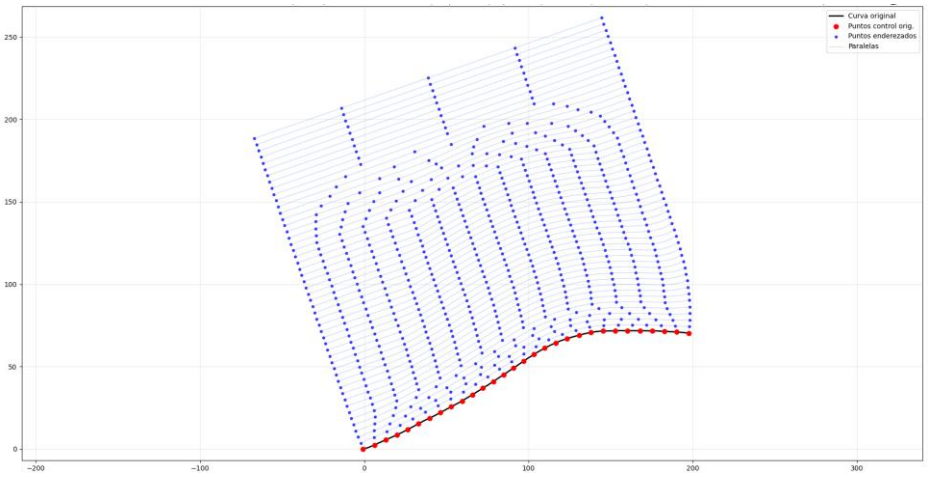

Figura 4.6. Representación de la Curva 1 con SM=0.50, STM=0.50 y 50 paralelas y con 5 puntos mínimo tras la reducción de puntos.

4.2 Curva 2

En la **Figura 4.7** se aprecia la forma de esta curva donde ya vemos una trayectoria con más curvaturas y cambios de sentido. En la **Figura 4.8** vemos el resultado con los parámetros por defecto donde se pueden apreciar varias zonas donde se produce mucho solape o zonas sin tratar. Esto es efecto se produce por dos motivos, el primero es que al ser una trayectoria con tantas curvas y pronunciadas necesitamos un número mayor de puntos para representarla, el segundo punto que afecto a este resultado es el de la reducción de puntos por el mismo caso que hemos visto en la curva anterior. Para solucionar este problema en esta trayectoria con tantas curvas hemos necesitado un número mínimo de ocho puntos para la reducción de puntos y un número de puntos de control inicial de cuarenta para representar la trayectoria. En la **Figura 4.9** vemos el resultado tras modifica el número mínimo de puntos para el enderezado y vemos que conseguimos el enderezado total en la paralela nº17. La **Figura 4.10** indica como se van reduciendo los puntos en esta trayectoria teniendo una zona plana y luego tras tener un buen enderezado se produce la reducción de puntos.

En la **Figura 4.11** tenemos el resultado de la curva en la otra dirección donde obtenemos el enderezado total en la paralela nº17

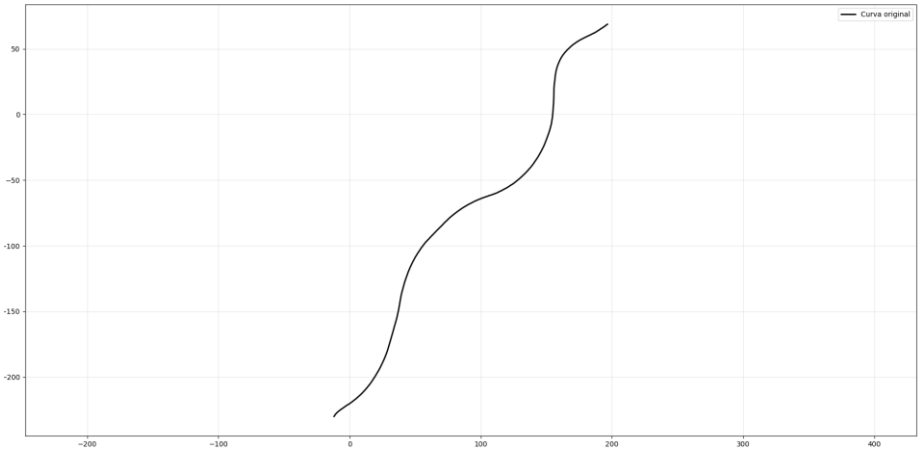

Figura 4.7. Representación gráfica de la curva 2.

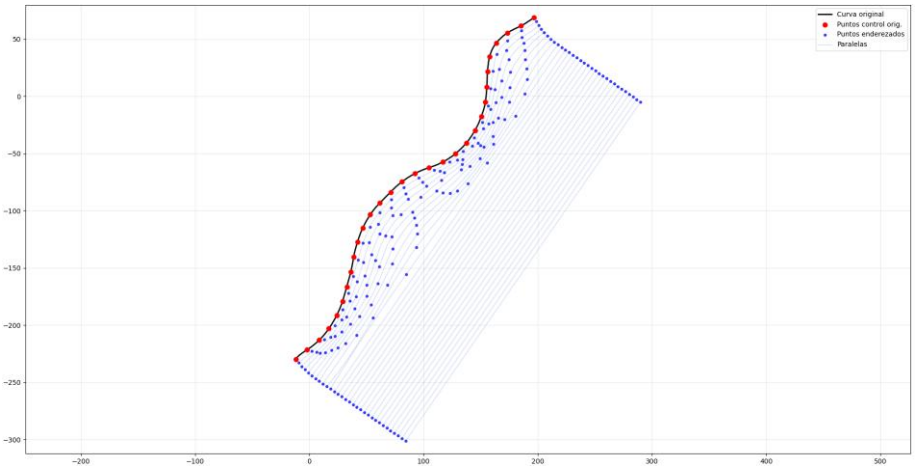

Figura 4.8. Resultado de la curva 2 en dirección positiva tras aplcar el algoritmo con los parámetros por defecto.

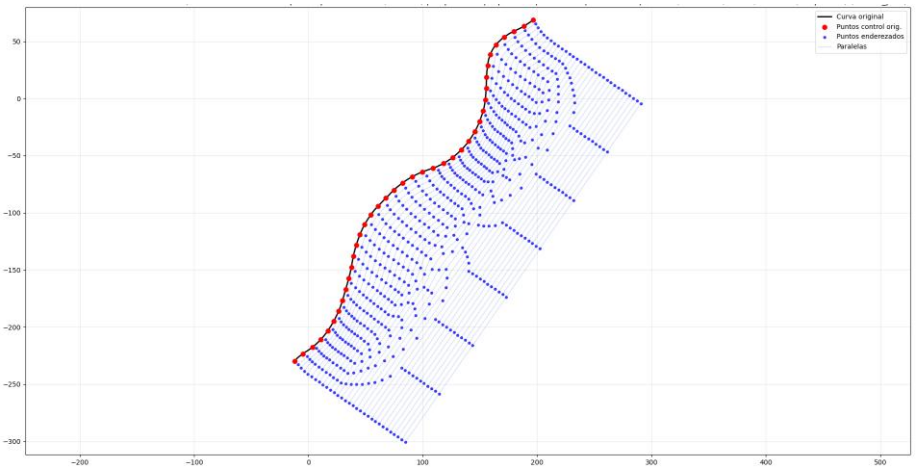

Figura 4.9. Resultado de la curva 2 en dirección positva con un número mínimo de ocho puntos para la reducción de puntos y un número puntos de control de cuarenta.

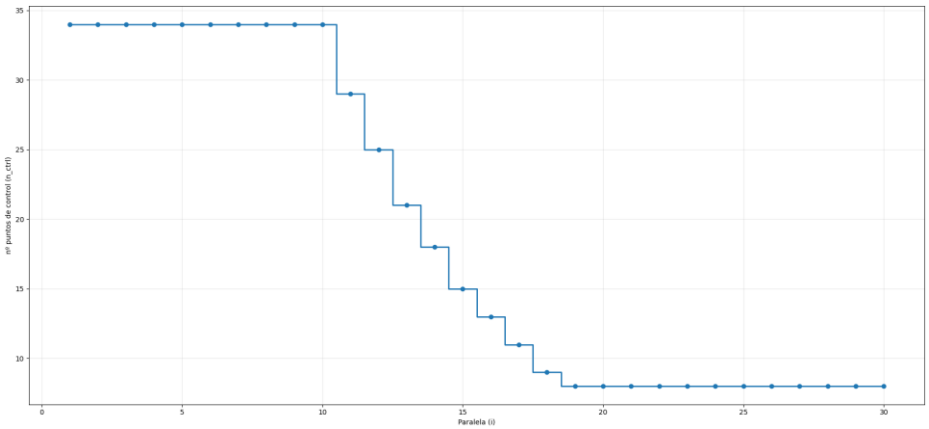

Figura 4.10. Gráfica que muestra la reducción de puntos que se va realizando a lo largo de las paralelas en la curva 2.

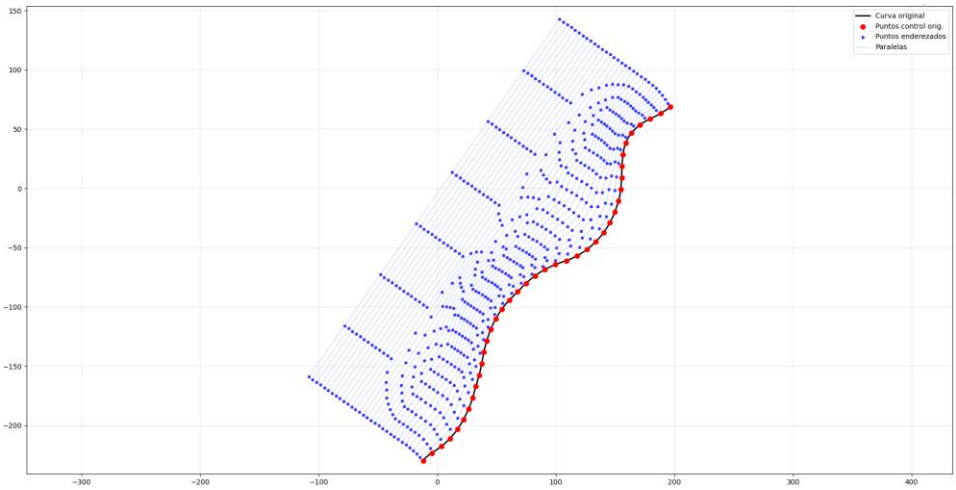

Figura 4.11. Resultado de la curva 2 en dirección negativa con un número mínimo de ocho puntos para la reducción de puntos y un número puntos de control de cuarenta.

4.3 Curva 3

En la **Figura 4.12** vemos la forma de la curva donde aquí tenemos una curva bastante cerrada seguida deuna curva bastante pronunciada. En la **Figura 4.13** vemos el resultado con los parámetros por defecto donde se pueden apreciar que a partir de cierta paralela se produce un descontrol debido, como hemos visto anteriormente, al número mínimo de puntos tras la reducción de puntos. En la **Figura 4.14** vemos el resultado donde hemos tenido que limitar el número mínimo de puntos de la reducción a diez puntos, hemos tenido que aumentar el número de paralelas a generar ya que se aprecia que se consigue el enderezado en la paralela nº46. En la **Figura 4.15** indica como se van reduciendo los puntos en esta trayectoria teniendo una zona plana y luego tras tener un buen enderezado se produce la reducción de puntos siendo esta como se aprecia en los escalones bastante brusca.

En la **Figura 4.16** vemos el resultado en la otra dirección donde podemos percibir en las primeras paralelas donde se encuentra la curva más cerrada se sobrepasa el solape máximo introducido, esto se produce debido a que estamos en una zona con un tramo de radio muy reducido, esto es una consecuencia geométrica de desplazar una curva muy cerrada ya que, aunque el algoritmo no permite desplazar el punto más del límite la propia geometría de las paralelas favorece el solape, con este detalle conseguimos el enderezado total en la paralela nº39.

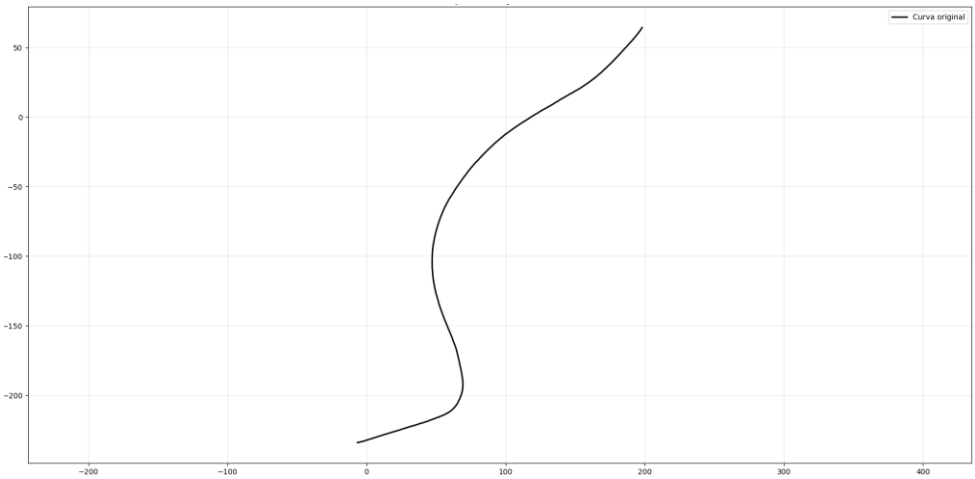

Figura 4.12 Representación gráfica de la curva 3.

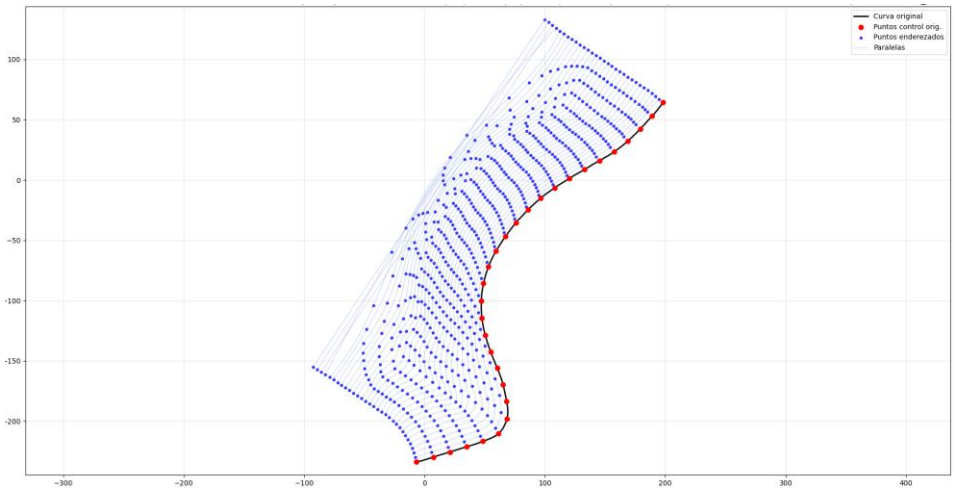

Figura 4.13. Resultado de la curva 3 en dirección positva tras aplcar el algoritmo con los parámetros por defecto.

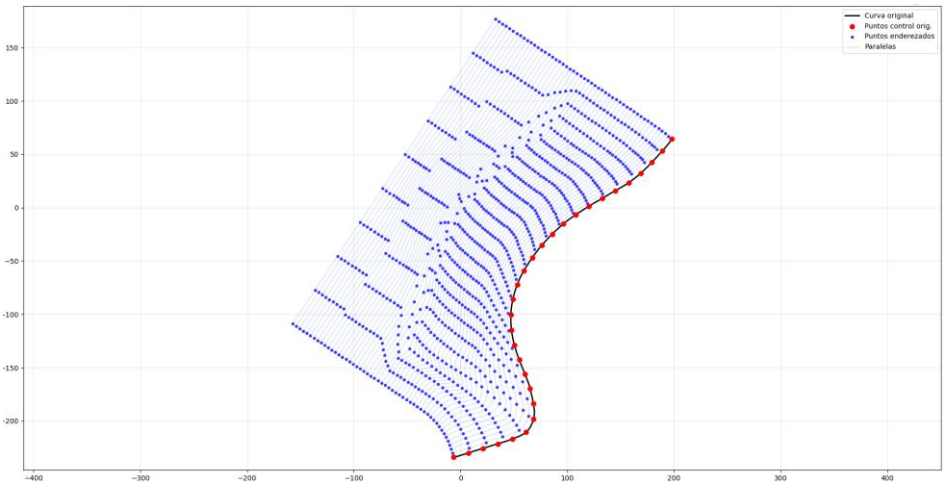

Figura 4.14. *Resultado de la curva 3 en dirección positiva con un número mínimo de diez puntos para la reducción de puntos.*

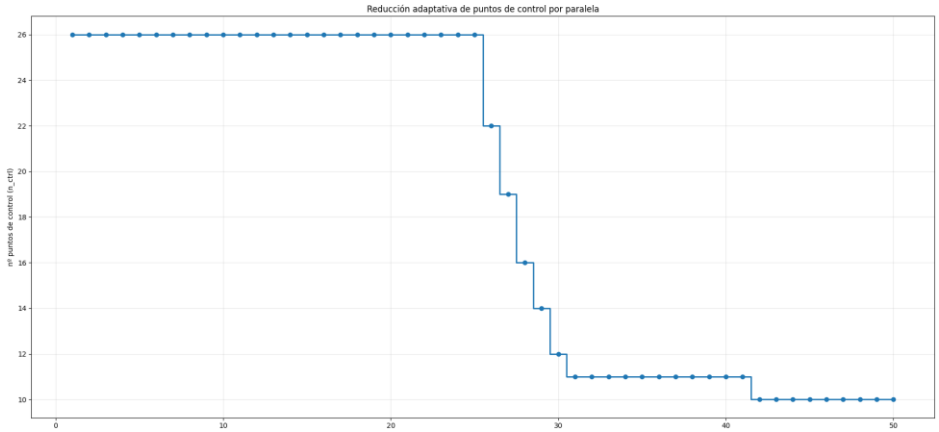

Figura 4.15 Gráfica que muestra la reducción de puntos que se va realizando a lo largo de las paralelas para la curva 3.

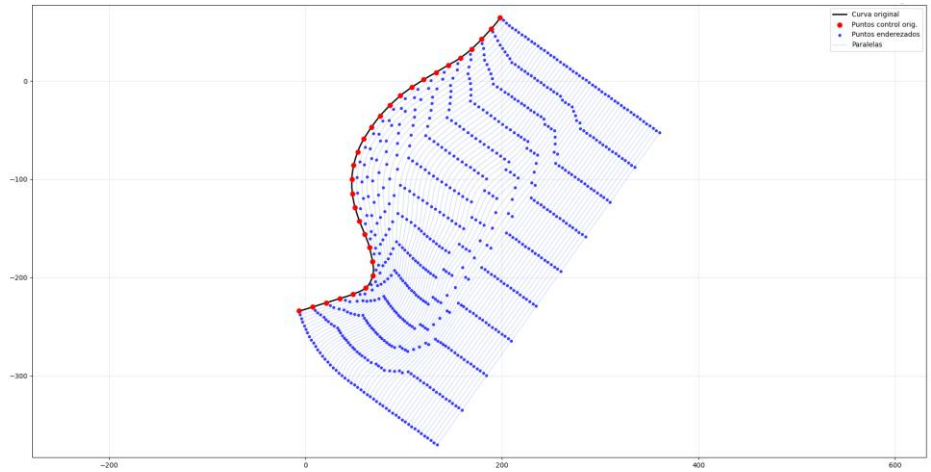

Figura 4.16 Resultado de la curva 3 en dirección negativa con un número mínimo de diez puntos para la reducción de puntos.

4.4 Curva 4

En la **Figura 4.17** vemos la última trayectoria que vamos a ver, dónde se aprecia que está compuesta por dos curvas pronunciadas. La **Figura 4.18** muestra el resultado con los parámetros por defecto cambiando a cincuenta el número de paralelas generadas ya que conseguimos el enderezado total en la parelala nº47, al tener curvas bastante abiertas esta trayectoria no produce problemas igual que la curva 1. La Figura 4.19 muestra el resultado del proceso de reducción de puntos, donde se puede apreciar que la reducción de puntos se produce tarde ya que el proceso de enderezado dura bastantes paralelas.

En la **Figura 4.20** tenemos el resultado en la otra dirección donde obtenemos el enderezado total en la paralela nº47

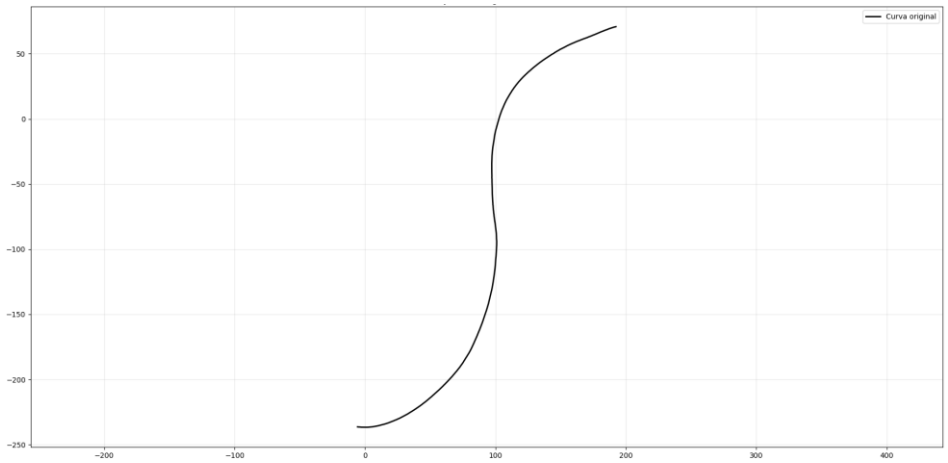

Figura 4.17. *Representación gráfica de la curva 4*

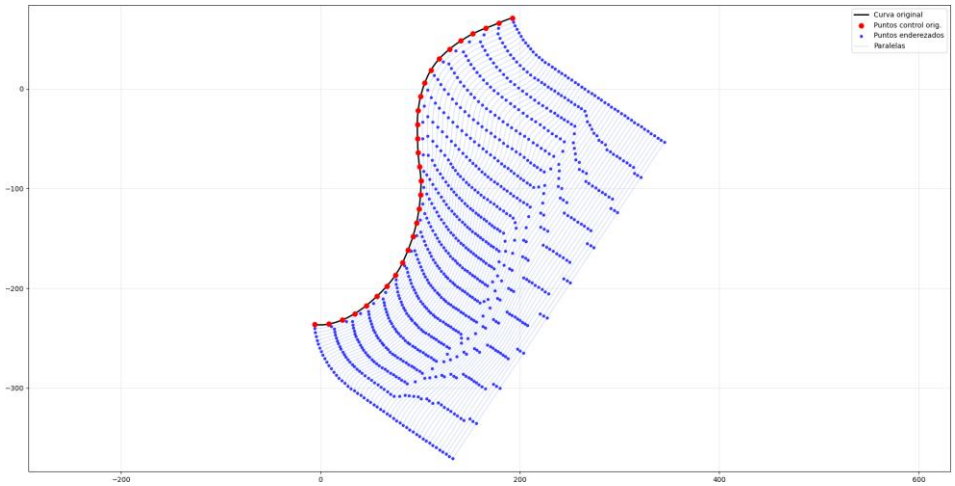

Figura 4.18. *Resultado de la curva 3 en dirección positva tras aplcar el algoritmo con los parámetros por defecto.*

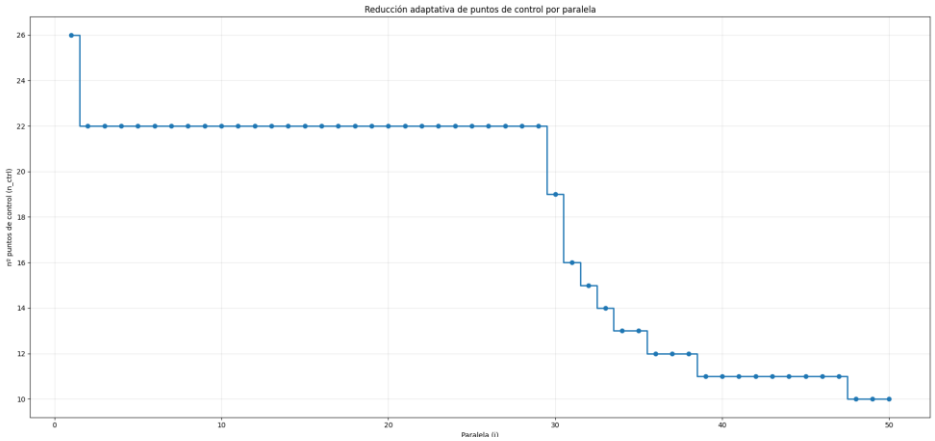

Figura 4.19 *Gráfica que muestra la reducción de puntos que se va realizando a lo largo de las paralelas en la curva 4*

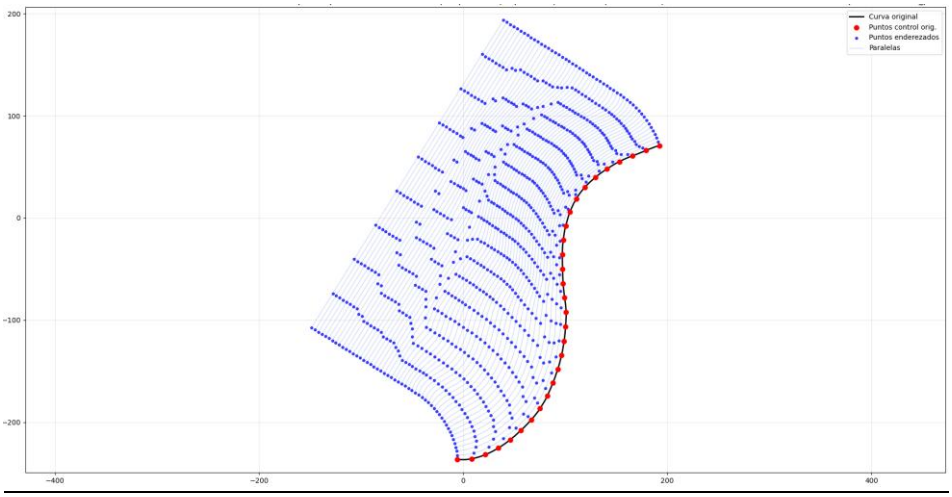

Figura 4.20 *Resultado de la curva 4 en dirección negativa con un número mínimo de diez puntos para la reducción de puntos.*

Capítulo 5
Conclusiones y líneas futuras

Capítulo 5: Conclusiones y líneas futuras

Este trabajo ha presentado un flujo de trabajo completo y coherente para transformar una trayectoria agrícola medida en campo en una familia de pasadas paralelas que tienden a la recta, respetando de forma explícita los límites agronómicos de solape máximo (SM) y zona sin tratar máxima (STM).

El proceso se articula en seis bloques (A–F): muestreo por arco, paralelas por normales, enderezado local, reducción adaptativa de puntos, remallado equiespaciado y construcción del spline cúbico para guiado. La elección de splines cúbicos paramétricos con parámetro de arco ha sido clave para garantizar suavidad, robustez y evaluaciones densas con un coste moderado.

En los resultados experimentales sobre trayectorias reales, el algoritmo genera pasadas con separación constante, suaviza ondulaciones locales y converge progresivamente hacia trayectorias rectas. Además, la combinación C–D–E–F (enderezado → reducción adaptativa → remallado → spline) conseguimos el objetivo de obtener trayectorias rectas de forma rápida y eficiente.

Aportaciones principales del trabajo:

1. Enderezado local eficiente (Bloque C):
   Se reemplaza la "fuerza bruta" por un ajuste local en la normal de cada punto, que minimiza una función de coste con términos de longitud, curvatura, alineación y rectitud, acotada por una barrera agronómica para SM/STM. Este diseño mantiene la intuición agronómica, pero reduce claramente el coste computacional y mejora la estabilidad de la convergencia.

2. Reducción adaptativa del número de puntos (Bloque D):
   Tras medir curvatura media y máxima, el sistema decide automáticamente cuántos puntos de control son necesarios en cada pasada. Con ello se simplifica la malla sin perder forma, lo que favorece el objetivo de tender a la recta.

3. Remallado equiespaciado por arco (Bloque E):
   Redistribuye los puntos supervivientes sin alterar la forma, mejora la condición del spline y previene ondulaciones espurias por sobre-muestreo local. Los extremos se mantienen anclados para fijar referencias geométricas y de rumbo.

4. Spline cúbico paramétrico (Bloque F):
   Con parámetro de arco normalizado, el spline interpola exactamente los nodos y garantiza

suavidad. El número de puntos de evaluación controla solo la densidad de muestreo, no la forma, lo que simplifica la explotación posterior para guiado.

La metodología propuesta reduce el número de puntos necesarios, mantiene los criterios agronómicos dentro de límites realistas y produce trayectorias suaves y evaluables con bajo coste. Estas propiedades la hacen apta para su integración en sistemas GNSS de guiado de tractores, donde se necesita calcular, almacenar y seguir pasadas largas y estables con recursos limitados.

Líneas futuras:

1. Ajuste automático de parámetros
    - Sintonía automática de SM/STM en función de radio local estimado.
    - Aprendizaje del peso w y de n_min a partir de métricas históricas (auto–calibración por parcela).

2. Control de calidad geométrica en línea
   Detección temprana de ondulaciones residuales o “dientes” mediante filtros de curvatura y reajuste local antes del spline final.

3. Robustez a datos GNSS reales:
   Inyección de ruido y outliers (GNSS)

En resumen, el trabajo consolida una versión mejorada y más práctica del método de generación de paralelas: mantiene el control agronómico, reduce la complejidad y entrega trayectorias sólidas para guiado. La experiencia acumulada sugiere que una sintonía adaptativa plenamente automática y la robustez a datos reales son las dos direcciones más prometedoras para cerrar el ciclo y llevar el algoritmo a una explotación operativa en campo.

Referencias

[1] «Parallel curve», *Wikipedia*. 4 de septiembre de 2025. Accedido: 10 de septiembre de 2025. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Parallel_curve&oldid=1309481884

[2] «Parallel curves». Accedido: 17 de septiembre de 2025. [En línea]. Disponible en: https://mathcurve.com/courbes2d.gb/parallele/parallele.shtml

[3] G. Martínez García, «Guiado GNSS de tractores; generación de curvas paralelas», 2024, Accedido: 10 de septiembre de 2025. [En línea]. Disponible en: https://uvadoc.uva.es/handle/10324/71263

[4] C. R de Boor, *A Practical Guide to Splines.*

[5] «AutoCAD 2015 Ayuda: No se encontró la página». Accedido: 10 de septiembre de 2025. [En línea]. Disponible en: https://help.autodesk.com/view/ACD/2015/ESP/?guid=GUID-58316136-30EB-499C-ACAD-31D0C653B2B

[6] J. W. Peterson, «Arc Length Parameterization of Spline Curves».

[7] «numpy.sqrt — NumPy v2.2 Manual». Accedido: 10 de septiembre de 2025. [En línea]. Disponible en: https://numpy.org/doc/2.2/reference/generated/numpy.sqrt.html

[8] «numpy.interp — NumPy v2.2 Manual». Accedido: 10 de septiembre de 2025. [En línea]. Disponible en: https://numpy.org/doc/2.2/reference/generated/numpy.interp.html

[9] Andrew Pressley, *Andrew Pressley Elementary Differential Geometry (2010, Springer)*. 2010. Accedido: 11 de septiembre de 2025. [En línea]. Disponible en: http://archive.org/details/andrew-pressley-elementary-differential-geometry-2010-springer

[10] «numpy.linalg.norm — NumPy v2.3 Manual». Accedido: 11 de septiembre de 2025. [En línea]. Disponible en: https://numpy.org/doc/2.3/reference/generated/numpy.linalg.norm.html

[11] Andrew Pressley, *Andrew Pressley Elementary Differential Geometry (2010, Springer)*. 2010. Accedido: 18 de septiembre de 2025. [En línea]. Disponible en: http://archive.org/details/andrew-pressley-elementary-differential-geometry-2010-springer

[12] «quaternions-cs520.ppt». Accedido: 18 de septiembre de 2025. [En línea]. Disponible en: https://viterbi-web.usc.edu/~jbarbic/cs520-s15/quaternions/quaternions-cs520-6up.pdf

[13] «atan2», *Wikipedia*. 10 de septiembre de 2025. Accedido: 14 de septiembre de 2025. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Atan2&oldid=1310660488

[14] «Curvature», *Wikipedia*. 9 de agosto de 2025. Accedido: 14 de septiembre de 2025. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Curvature&oldid=1304952285

[15] L. Pagani y P. J. Scott, «Curvature based sampling of curves and surfaces», *Comput. Aided Geom. Des.*, vol. 59, pp. 32-48, ene. 2018, doi: 10.1016/j.cagd.2017.11.004.

[16] «numpy.interp — NumPy v2.3 Manual». Accedido: 15 de septiembre de 2025. [En línea]. Disponible en: https://numpy.org/doc/stable/reference/generated/numpy.interp.html

[17] «CubicSpline — SciPy v1.16.2 Manual». Accedido: 15 de septiembre de 2025. [En línea]. Disponible en: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html