UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA CIVIL

GENERACIÓN DE ESTRUCTURAS A PARTIR DE DOMINIOS OPTIMIZADOS TOPÓLOGICAMENTE UTILIZANDO TÉCNICAS DE DISEÑO ASISTIDO POR ORDENADOR

POR

Felipe Ignacio Millar Medel

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil

Profesor Guía
Patricio Cendoya H.

Profesor Revisor
Rodrigo Silva M.

Junio, 2024
Concepción (Chile)

A mis queridos padres, Mario y Bernardita, cuyo apoyo me encamino en momentos difíciles. A mis hermanos Mario Millar Medel e Ignacia Millar Medel, cuya presencia y aliento han sido un regalo preciado en cada etapa de mi vida.

Igualmente, a mis apreciados amigos, Claudio Medina, Benjamín Vásquez, Mauricio Torres, Carlos Medina, Claudio Chacana, y muchos más, que la vida me brindó el valioso regalo de conocer. Su amistad ha iluminado mi camino con momentos de risa, aprendizaje y compañía inestimable.

Este logro está dedicado a ustedes

AGRADECIMIENTOS

Quiero extender mi más sincero agradecimiento a todos quienes contribuyeron de manera significativa en el desarrollo de esta memoria de título. En primer lugar, quiero expresar mi gratitud al equipo de trabajo que estuvo a mi lado durante todo este proceso formativo Rodrigo Silva y quienes brindaron su apoyo, retroalimentación y dedicación para hacer posible la realización de este proyecto.

Una mención especial va dirigida al profesor Patricio Cendoya cuya confianza, disposición y apoyo incondicional fueron fundamentales en cada etapa de este camino. Su orientación y motivación constante, incluso más allá del horario académico, han sido un pilar esencial.

También deseo expresar mi reconocimiento a mis amigos y compañeros de colegio, carrera y universidad, cuya compañía, distracciones y apoyo fueron un alivio durante este proceso desafiante. En particular, quiero agradecer especialmente a Claudio Chacana, Mauricio Torres, y Claudio Medina por su apoyo constante y entrega total en los momentos en que más los necesité.

Por último, mi agradecimiento se extiende a mi familia. Su amor apoyo constante y entrega de herramientas para mi crecimiento han sido un motor fundamental en mi camino.

Este logro es también de ustedes, y su contribución ha sido esencial en cada paso de esta travesía.
A todos ustedes ¡Muchísimas gracias!

RESUMEN

La optimización topológica (OT) se erige como un método esencial para definir distribuciones eficientes de material en dominios específicos. No obstante, las topologías resultantes suelen presentar bordes irregulares y poco aptos para aplicaciones industriales. En este contexto, esta investigación va más allá al no solo abordar el suavizado de bordes, sino también al aterrizaje de los resultados en la práctica nacional. Se logró la generación de geometrías optimizadas topológicamente con bordes suavizados y adaptados a la construcción real, incorporando bordes lineales, splines y circulares mediante la utilización de distintos tipos de materiales y una interfaz adaptada para la correcta puesta en marcha de innumerables tipologías estructurales.

El enfoque de esta investigación se amplía aún más al introducir una reconstrucción de imágenes ráster derivadas de los resultados topológicos dados por la estrategia de optimización soft kill BESO mediante un proceso de esqueletización y operaciones basadas en regresión y segmentación de ramas. Este tratamiento transforma las geometrías generadas en elementos finitos tipo frames, listos para ser utilizados en otros softwares comerciales de análisis estructural. Así, la metodología desarrollada permite una transición efectiva desde la optimización topológica hasta la aplicación práctica en el diseño y análisis de estructuras.

La validación de esta metodología se realizó a través de dos casos de análisis: una viga larga en voladizo con carga puntual en el extremo libre, una viga simplemente apoyada con carga puntual en el centro. Estos casos demostraron la viabilidad y eficacia de la rutina desarrollada, produciendo geometrías con bordes suavizados y adaptados, listas para ser utilizadas en aplicaciones estructurales.

De esta manera, se contribuye al campo del diseño y optimización estructural, proporcionando herramientas avanzadas para la generación de estructuras eficientes y prácticas en la industria actual.

SUMMARY

Topological optimization (TO) stands as an essential method for defining efficient material distributions within specific domains. However, the resulting topologies often exhibit irregular edges, making them unsuitable for industrial applications. In this context, this research goes beyond merely addressing edge smoothing by applying the results to practical, real-world scenarios. It achieved the generation of topologically optimized geometries with smooth edges, adapted to real construction by incorporating linear, spline, and circular edges using various materials and an interface tailored for the correct implementation of numerous structural typologies.

The focus of this research is further expanded by introducing a reconstruction of raster images derived from topological results given by the soft kill BESO optimization strategy. This involves a skeletonization process and operations based on branch regression and segmentation. This treatment transforms the generated geometries into frame-type finite elements, ready to be used in other commercial structural analysis software. Thus, the developed methodology enables an effective transition from topological optimization to practical application in the design and analysis of structures.

The validation of this methodology was carried out through two case analyses: a long cantilever beam with a point load at the free end, a simply supported beam with a central point load. These cases demonstrated the feasibility and effectiveness of the developed routine, producing geometries with smooth, adapted edges, ready for use in structural applications.

In this way, this research contributes to the field of structural design and optimization, providing advanced tools for generating efficient and practical structures in the current industry.

TABLA DE CONTENIDOS

LISTA DE TABLAS

ÍNDICE DE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN

1.1 Motivación

En el ámbito del diseño y optimización estructural, el proceso tradicional de diseño y cálculo estructural ha dependido en gran medida de la experiencia ingenieril y el análisis numérico. Este enfoque, aunque efectivo, a menudo implica un proceso iterativo de ajuste y refinamiento de diseños, lo cual puede ser propenso a la subjetividad y carecer de una base objetiva. En este contexto, la optimización topológica emerge como una alternativa innovadora que busca automatizar y objetivar este proceso, proporcionando soluciones más eficientes y efectivas.

La optimización topológica ha cobrado una importancia fundamental en el diseño asistido por computadora, permitiendo la generación automática de configuraciones estructurales óptimas a partir de dominios topológicos. Este enfoque facilita la identificación de la distribución óptima de material dentro de un dominio específico, considerando las solicitudes y restricciones definidas. Además de minimizar el consumo de material, se traduce en una reducción de costos y una menor huella ecológica durante la fabricación de elementos estructurales.

El método Soft Kill BESO (Bi-directional Evolutionary Structural Optimization) ajusta la distribución del material en función de la eficiencia estructural, generando imágenes ráster con densidades relativas que, debido a sus bordes irregulares, no son aptas para el cálculo estructural. Además, estas geometrías optimizadas pueden no ser utilizables directamente como estructuras de celosía debido a la naturaleza de los elementos finitos. Aquí, los algoritmos avanzados de predicción de celosías son cruciales para transformar estas geometrías en configuraciones prácticas y aptas para su el uso en programas de análisis de estructuras y métodos de construcción tradicionales.

1.2 Objetivos

1.2.1 Objetivo general

Desarrollar un algoritmo de suavizado de contornos y de definición de configuraciones estructurales a partir de la OT, aptas para el uso en programas de análisis de estructuras.

1.2.2 Objetivos específicos

Se contemplan los siguientes objetivos específicos:

- ·Desarrollar un algoritmo de suavizado de contornos utilizando técnicas de visión por ordenador.
- ·Implementar un algoritmo de predicción, segmentación, reconstrucción y alisamiento de esqueleto.
- ·Definir configuraciones estructurales aptas para el análisis en programas de elementos finitos (SAP2000®).
- ·Desarrollar ejemplos de validación de los algoritmos desarrollos.

1.3 Plan de trabajo

La etapa inicial consistió en una revisión bibliográfica sobre la optimización topológica en la ingeniería estructural, abarcando su evolución histórica, su rol en la superación de limitaciones del diseño tradicional y los métodos predominantes, con especial énfasis en el método Soft Kill BESO en dos dimensiones y sus fundamentos teóricos y matemáticos.

La etapa subsiguiente exploró aplicaciones de técnicas avanzadas de visión por ordenador para interpretar y optimizar resultados topológicos, integrándolas en el diseño asistido por ordenador para generar modelos CAD óptimos, desde la generación de un modelo optimizado topológicamente, seguido de la esqueletización, la extracción de marcos espaciales y la optimización de configuraciones resultantes.

Adicionalmente, se evaluaron las herramientas disponibles en MATLAB® para el suavizado de contornos, segmentación semántica, extracción de esqueletos, análisis estructural con librerías CALFEM® y FEATOOLS®, y la exportación a programas como SAP2000® y FUSION360®, así como el reconocimiento de patrones y la aplicación de modelos de ajuste de curvas paramétricas para datos afectados por ruido.

Específicamente, para el suavizado de contornos y la interpretación de celosías estructurales, se estableció una metodología avanzada para la transición hacia fases subsiguientes de optimización y modelado estructural. Empleando funciones de la Image Processing Toolbox de MATLAB®, se abarcó desde la adquisición y preprocesamiento de imágenes derivadas del algoritmo Soft Kill BESO, optimizando la calidad visual de las imágenes raster, hasta la implementación de métodos de esqueletización y segmentación de cuencas para analizar la topología de la estructura. Se aplicaron técnicas de filtrado, como la transformada de distancia y la umbralización, culminando en una reconstrucción detallada con métodos como el B-spline y por redondeo de vértices

Finalmente, se implementó una rutina en MATLAB® que sintetizó todos los procesos anteriores para generar y analizar ejemplos clásicos citados en la literatura, comparando y validando los resultados con investigaciones previas como las de Oviedo (2020) y Gedig (2010), asegurando la efectividad y aplicabilidad de la rutina propuesta.

1.4 Principales resultados

Se desarrollo una serie de rutinas en MATLAB® que refinan la generación y adaptación de configuraciones estructurales a partir de la Optimización Topológica (OT) utilizando el método *Soft Kill* BESO en dos dimensiones. Esto se logró mediante la integración de técnicas avanzadas de procesamiento de imágenes y análisis estructural, tales como el reconocimiento de patrones, filtrado, segmentación de cuencas, alisado y ordenación de bordes, vectorización de esqueletos binarios, y la aplicación de modelos de ajuste de curvas suaves.

El desarrollo encapsuló dos enfoques principales de reconstrucción: la interpretación de estructuras tipo celosía a partir de imágenes ráster derivadas de la OT y la reconstrucción mediante alisamiento de bordes. Además, se integraron rutinas para la exportación directa de los resultados a software de análisis estructural como SAP2000®, Fusion360®, y bibliotecas especializadas como CALFEM® y FEATOOLS® de MATLAB®.

La validación de este desarrollo se realizó mediante la aplicación en dos estudios de caso: una viga larga en voladizo y una viga simplemente apoyada. Los resultados demostraron cumplimiento con las restricciones de volumen y desplazamiento, evidenciando la eficacia del enfoque de reconstrucción.

1.5 Organización de la memoria

Este trabajo consta de seis capítulos. El primero es introductorio, presentando la motivación, objetivos, una breve metodología y los principales resultados obtenidos. Se introduce el contexto del diseño y optimización estructural, destacando las limitaciones del proceso tradicional basado en experiencia e iteración y la necesidad de objetivar y optimizar el diseño estructural mediante técnicas de optimización topológica y visión por ordenador.

En el Capítulo 2 se efectúa una revisión bibliográfica sobre la optimización topológica como una alternativa automatizada y objetiva para el diseño estructural eficiente. Se discuten las geometrías optimizadas y las limitaciones relacionadas con bordes irregulares y ubicación de nodos, destacando el enfoque propuesto para mejorar las configuraciones estructurales obtenidas mediante la OT.

El Capítulo 3 describe el desarrollo de los algoritmos propuestos, presentando la metodología de suavizado de contornos basada en técnicas de visión artificial y explicando la adaptación de configuraciones estructurales para su análisis en programas de elementos finitos.

En el Capítulo 4 se presenta el fundamento teórico/matemático del algoritmo de optimización topológica, el enfoque de reconstrucción de resultados a través del proceso de esqueletización y operaciones de regresión y segmentación, y cómo esta metodología transforma las geometrías en elementos finitos tipo barra listos para su análisis en otros softwares comerciales.

El Capítulo 5 aborda la validación de los algoritmos desarrollados, describiendo los casos de análisis utilizados y mostrando las geometrías resultantes con bordes suavizados y adaptados, así como topologías estructurales reconstruidas, analizando la eficacia de la metodología propuesta en aplicaciones estructurales.

El Capítulo 6 resume las conclusiones derivadas de la investigación, destacando los logros alcanzados en relación con los objetivos planteados y discutiendo las implicaciones y aplicaciones potenciales de los algoritmos desarrollados en la industria del diseño y análisis estructural. Finalmente, se proponen posibles direcciones para futuras investigaciones en este campo.

CAPÍTULO 2: ESTADO DEL ARTE

2.1 Introducción

En este capítulo se introduce el contexto del diseño y optimización estructural desde el aspecto topológico, destacando las limitaciones del proceso tradicional basado en experiencia e iteración. Se presentan técnicas de visión por ordenador aplicadas para interpretación de los resultados de la OT basadas en diseño asistido por ordenador cómo lo son los algoritmos de segmentación de imágenes y esqueletización para las distintas topologías estructurales resultantes y optimización de formas respectiva, todo esto con el fin de objetivar y optimizar el diseño estructural.

2.2 Optimización estructural

A lo largo de la historia, los seres humanos han buscado optimizar sus sistemas mediante el desarrollo de estructuras para mejorar el rendimiento. Inicialmente, las limitaciones tecnológicas impedían una real optimización. Tras la Segunda Guerra Mundial, el avance tecnológico permitió métodos más sofisticados, aunque no alcanzaron el mismo auge actual debido a las limitaciones de la época.

En este contexto, la optimización se define como el proceso mediante el cual se busca la mejor solución posible entre un conjunto de resultados; satisfaciendo simultáneamente las restricciones del sistema al que se aplica (Lozano et al., 2010). Dependiendo de las variables de diseño que se quieran optimizar y de la formulación inicial del problema de optimización, se pueden distinguir tres tipos principales de optimización estructural: optimización de dimensiones, optimización de forma y optimización topológica (Bendsøe y Sigmund, 2003).

2.2.1 Optimización de dimensiones

Este método busca optimizar el tamaño de los elementos previamente definidos por el diseñador. Su aplicación más usual es en las estructuras reticuladas, donde se definen las longitudes y conectividades, para luego optimizar la sección de las barras, que corresponde a la variable de diseño. La optimización de dimensiones es la forma más simple de optimización estructural, y por ende suele

ser la más común de encontrar en la literatura (Uarac et al., 2015). En la Figura 3.1 se ilustra un ejemplo de aplicación de optimización de dimensiones.

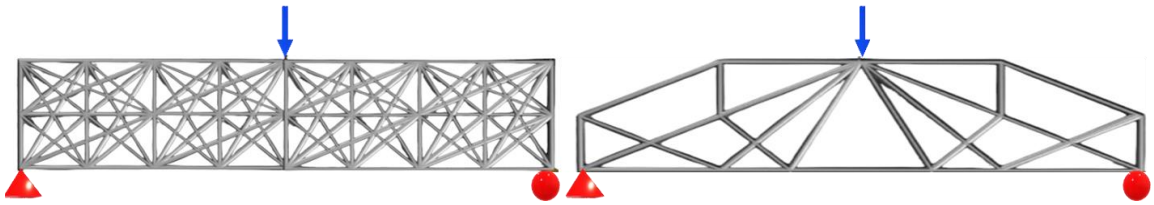

Figura 2.1 Ejemplo de aplicación de optimización de dimensiones. Izquierda dominio inicial con cargas y soporte. Derecha, diseño optimizado mediante algoritmos genéticos.

2.2.2 Optimización de forma

La optimización de forma se centra en modificar las características geométricas de una estructura, como la distribución de espesores a lo largo de los elementos estructurales, el diámetro de los agujeros, las coordenadas de los nodos de un reticulado, el contorno de una estructura continua, los radios de filetes u otras medidas. En la optimización de forma, las variables de diseño, como la distribución de espesores, afectan a varios elementos en lugar de a individuales, lo que la diferencia de la optimización de dimensiones (Olason & Tidman, 2011). En la Figura 3.2 se presenta un ejemplo de aplicación de optimización de forma.

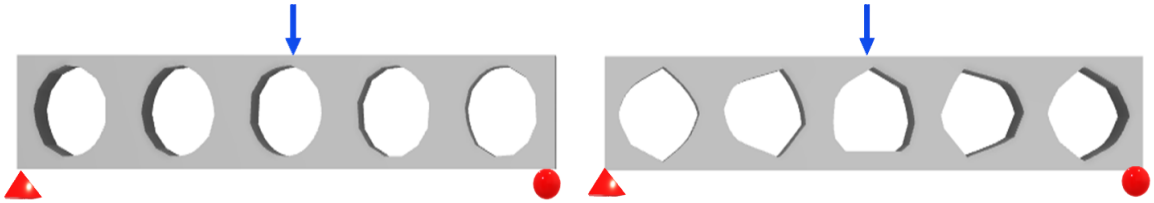

Figura 2.2 Ejemplo de aplicación de optimización de forma. Izquierda diseño inicial con cargas y soporte. Derecha, diseño optimizado

2.2.3 Optimización topológica

La optimización topológica en tanto se define como una técnica que permite hallar la mejor distribución de material dentro de un dominio inicial, para determinado estado de cargas y condiciones de apoyo, de forma que la estructura resultante maximice el desempeño mecánico (Tovar, 2005).

En la optimización topológica (OT) estructural se busca que la distribución de material en un componente sea óptima respecto a algún criterio como minimizar la energía interna de deformación lo que significa maximizar la rigidez, minimizar los esfuerzos dentro de la estructura, minimizar la deformación debido a vibraciones, entre otros (Leon-Medina & Cardenas-Flechas).

Desde el punto de vista técnico, la optimización topológica resulta ser mucho más desafiante que los otros dos métodos antes mencionados, y al mismo tiempo la que genera mayores beneficios económicos, junto con permitir diseños novedosos y altamente eficientes. Además, se diferencia de los otros dos modelos en que se apunta a la eliminación del criterio del diseñador en la toma de decisiones respecto al modelo inicial, siendo sólo necesario la definición de los límites del dominio o condiciones de contorno del problema y no requiere de un diseño de partida (Irribarra, 2011). En la Figura 2.3 se ilustra un ejemplo de aplicación de optimización topológica.

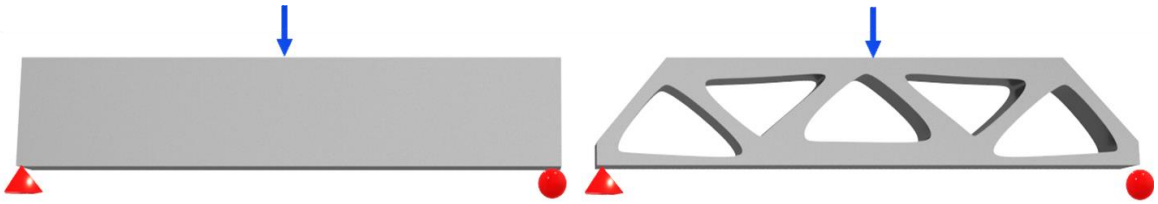

Figura 2.3 Ejemplo de aplicación de optimización topológica. Izquierda diseño inicial con cargas y soporte. Derecha, diseño optimizado.

2.2.4 Métodos de Optimización topológica

Los modelos propuestos para resolver este tipo de problemas se pueden clasificar en discretos, los cuales pueden tomar valores de 0 o 1, o continuos, con valores entre el intervalo de 0 y 1, dependiendo del tipo de variable de diseño (Lozano et al., 2010).

Existen distintas metodologías para la optimización topológica en el diseño de estructuras. El método SIMP (Solid Isotropic Material with Penalization) es el más utilizado para prever la disposición óptima de material en un diseño específico, considerando cargas, condiciones de contorno y criterios de rendimiento. En la formulación discreta, el método BESO (Bi-directional Evolutionary Structural Optimization) es preferido sobre ESO por su capacidad para eliminar y reincorporar elementos de la malla en iteraciones posteriores.

Este método avanza un paso más que el método ESO (Evolutionary Structural Optimization) original propuesto por Xie y Steven (Xie & Steven, 1993) y ya no sólo se eliminan elementos de la malla que no cumplan con un cierto estado tensional, sino que en iteraciones posteriores pueden volver a incorporarse a la malla y el índice de rechazo no necesita limitarse tanto como en ESO (Uarac, 2014). En respuesta a esta limitación, surge el método soft-kill BESO (híbrido), el cual fusiona las virtudes de ambos métodos. Este enfoque híbrido se implementa en la rutina MATLAB® empleada en el desarrollo de este trabajo.

2.3 Desafíos y Oportunidades en la Implementación CAD/CAM de Diseños Estructuralmente Optimizados

A pesar de que las técnicas de optimización estructural han producido diseños eficientes en materiales, la fabricación práctica e implementación de esos diseños se ven limitadas debido a las geometrías complejas de las salidas de diseño optimizadas. Los avances actuales en la fabricación aditiva sortean parcialmente este problema, predominantemente en el campo de la ingeniería mecánica. No obstante, en la ingeniería civil, el uso de estas técnicas sigue siendo limitado (Sarma et al., 2023).

Los enfoques computacionales modernos para optimizar diseños estructurales utilizan metaheurísticas como algoritmos genéticos y autómatas celulares. Estos métodos arrancan con una estructura base que sirve de población inicial. Esta técnica facilita la búsqueda de soluciones óptimas para la estructura.

La selección de tales estructuras base requiere suposiciones implícitas sobre la ubicación de esas estructuras en el paisaje de diseño. Como tal, hay una necesidad de métodos para generar tales estructuras automáticamente, sin suposiciones fuertes sobre el paisaje de diseño (Jootoo & Lattanzi, 2018).

2.4 Estrategias de Post-Procesamiento

En la práctica la mayoría de los paquetes comerciales de TO no tienen herramientas automatizadas para el postprocesamiento. Estas representaciones geométricas incluyen representación esquelética, modelo triangular simplificado, representación NURBS, descomposición de volumen, etc. Por lo

tanto, las estrategias de postprocesamiento pueden variar desde un simple remallado hasta una compleja extracción del esqueleto y ajuste de superficies analíticas.

La estrategia más común utilizada en los sistemas comerciales es la reconstrucción basada en superficies. PTC® utiliza la técnica de subdivisión, mientras que Evolve®, Rhino® y MeshMixer® emplean la reconstrucción basada en NURBS. Fusion 360 Generative Design® se basa en T-splines para generar múltiples modelos CAD estancos que satisfacen los requisitos del diseñador. Sin embargo, ninguna de estas herramientas genera eficientemente un modelo CAD paramétrico (Subedi et al., 2020).

2.5 Conclusiones

En este capítulo, se abordó la optimización estructural desde una perspectiva topológica, resaltando las deficiencias del enfoque tradicional, basado en la experiencia y la iteración, y cómo estas pudieron ser superadas mediante la implementación de técnicas avanzadas de visión computacional y diseño asistido por ordenador. Se exploraron las diversas facetas de la optimización estructural, incluyendo la optimización de dimensiones, formas y, de manera más prominente, la optimización topológica, que redefine la distribución de material en un diseño, orientándose hacia la maximización del rendimiento mecánico sin la necesidad de un diseño inicial.

Además, se abordaron los desafíos inherentes a la implementación de estos diseños optimizados en la práctica, particularmente en la ingeniería civil, donde las complejidades geométricas y los avances en fabricación aditiva presentaron tanto desafíos como oportunidades.

Finalmente, se examinaron las estrategias de postprocesamiento, cruciales para la transición de modelos optimizados a implementaciones prácticas, destacando la variedad de técnicas disponibles y la necesidad de una mayor automatización en la reconstrucción y adaptación de modelos CAD para facilitar la aplicación efectiva de diseños estructurales optimizados en el mundo real.

CAPÍTULO 3: FORMULACIÓN TEORICO/MATEMÁTICA

3.1 Introducción

Este capítulo se centra en la optimización topológica, formulando la función objetivo y sus restricciones. Se introduce el método Soft Kill BESO para reemplazar elementos ineficientes, logrando un diseño eficiente. Se discuten métodos de esqueletización y vectorización para analizar el diseño optimizado, mediante la construcción grafos relacionales para representar configuraciones estructurales. Además, se abordan estrategias para tratar inestabilidades numéricas, facilitar la fabricación y estabilidad estructural.

3.2 Optimización topológica

3.2.1 Función objetivo y restricciones asociadas

En la optimización topológica, el objetivo es encontrar la distribución óptima de materiales para maximizar la rigidez estructural o reducir la masa. Esto se logra minimizando la compliance promedio para un volumen de material dado. La compliance, que mide la flexibilidad o energía total de deformación de una estructura, es el inverso de la rigidez y se expresa según la Ecuación 3.1.

$$\min_x \mathrm{C}\,(x) = \frac{1}{2} U^T K U = \frac{1}{2} \sum_{i=1}^{n} {u_i}^T k_i u_i \tag{3.1}$$

Donde U y K son el vector de desplazamientos y la matriz de rigidez global de la estructura, n es la cantidad de elementos, u_i es el vector de desplazamiento nodal del elemento i y k_i es la matriz nodal de rigidez del elemento i. Dicho problema de minimización debe cumplir una serie de restricciones de masa o volumen objetivo mostradas en Ecuación 3.2 y Ecuación 3.3.

$$\mathrm{V}^* - \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{V_i x_i} = 0 \tag{3.2}$$

$$\mathrm{x_i} = \mathrm{x_{min}} = 0{,}001 \quad \text{o} \quad \mathrm{x_i} = 1 \tag{3.3}$$

Donde v_i es el volumen del elemento i, V^* es el volumen prescrito esperado de la optimización. Mientras que x_i corresponde a la variable binaria que denota la densidad relativa del elemento i-ésimo. con x_{min} típicamente 0,001, cuyo fin es no remover por completo los elementos innecesarios.

En efecto, la estrategia *Soft Kill* BESO plantea que en lugar de remover totalmente los elementos ineficientes estos sean reemplazados por elementos de baja densidad utilizando un esquema de interpolación de material con penalización (Huang y Xie 2009).

Para lograr un diseño de vacío casi sólido, el módulo de Young del material intermedio se interpola en función de la densidad del elemento como se evidencia en la Ecuación 3.4.

$$E(x_i) = E_0 x_i^p \tag{3.4}$$

Donde E_0 es el módulo de elasticidad del i-esimo elemento, y p es el factor de penalización que reduce la elasticidad de los elementos con densidad relativa intermedia. Mientras que K es la matriz de rigidez global, expresada mediante la matriz de rigidez elemental del sólido K_i^0 y las variables de diseño x_i como se lo expresa la Ecuación 3.5.

$$K = \sum_i x_i^p K_i^0 \tag{3.5}$$

Sujeto a restricciones de carácter estático lineal como lo son el equilibrio de fuerza-rigidez global y restricciones funcionales requeridas, que se presenta en Ecuación 3.6.

$$Ku = f \tag{3.6}$$

Donde f es el vector de fuerzas globales de la estructura, u_j^* y σ_j^* son el desplazamiento máximo admisible y la tensión máxima admisible en el i-ésimo elemento respectivamente.

3.2.2 Formulación del problema de optimización

A continuación, y haciendo el reemplazo respectivo de las ecuaciones antes presentadas, se procede a

resolver el problema de optimización el cual se encuentra dado por la expresión presentada en la Ecuación 3.7 sujeto a la restricción mostrada en Ecuación 3.6 y Ecuación 3.8.

$$\min_{x} C(x) = \frac{1}{2} \cdot f^{T} \cdot u = \frac{1}{2} \cdot u^{T} \cdot K \cdot u = \frac{1}{2} \cdot \sum_{e=1}^{n} x_e^{p} \cdot u_e^{T} \cdot K_e \cdot u_e \tag{3.7}$$

$$V^{*} = \sum_{e=1}^{n} V_e \cdot X_e \rightarrow V(x) - f \cdot V_0 = 0 \tag{3.8}$$

El problema de optimización se resuelve introduciendo un vector multiplicador lagrangiano λ utilizando la Ecuación 3.9, obteniéndose la función objetivo modificada (Huang y Xie, 2010), presentada en la Ecuación 3.9.

$$C = \frac{1}{2} f^{T} u + \lambda^{t}(f - Ku) \tag{3.9}$$

En la optimización estructural evolutiva, la restricción de volumen estructural dada en la Ecuación 3.8 se usa como un parámetro evolutivo, excluyéndose del problema de optimización lagrangiana. La sensibilidad de la función objetivo modificada se determina mediante la derivada de la función de compliance respecto de la densidad del elemento x_i expresada en la Ecuación 3.10.

$$\frac{\partial C}{\partial x_i} = \frac{1}{2}\frac{\partial f^{T}}{\partial x_i} u + \frac{1}{2}\frac{\partial u}{\partial x_i} f^{T} + \frac{\partial \lambda^{t}}{\partial x_i}(f - Ku) + \lambda^{t}\left(\frac{\partial f}{\partial x_i} - \frac{\partial K}{\partial x_i} u - \frac{\partial u}{\partial x_i} K\right) \tag{3.10}$$

Es importante señalar que el tercer término $\frac{\partial \lambda^{t}}{\partial x_i}(f - Ku)$ en la ecuación anterior se anula debido a que satisface la ecuación de equilibrio $Ku = f$. Además, se asume que la variación de un elemento no afecta al vector de carga aplicada y, por lo tanto $\frac{\partial f^{T}}{\partial x_i} = 0$, anulando el primer término. Así la sensibilidad de la función objetivo se simplifica a la expresión expuesta en Ecuación 3.11.

$$\frac{\partial C}{\partial x_i} = \left(\frac{1}{2} f^{T} - K\lambda^{t}\right)\frac{\partial u}{\partial x_i} - \lambda^{t}\frac{\partial K}{\partial x_i} u \tag{3.11}$$

Dado que $\left(\frac{1}{2}\mathrm{f}^{T} - \mathrm{K}\lambda^{\mathrm{t}}\right)$ es análogo a $(\mathrm{f} - \mathrm{Ku})$ el cual es igual a cero, el vector de multiplicador lagrangiano λ^{t} puede elegirse libremente. De está forma con el fin de eliminar $\frac{\partial \mathrm{u}}{\partial \mathrm{x_i}}$ se elige convenientemente $\lambda = \frac{1}{2}u$, de modo que $\frac{1}{2}\mathrm{f}^{T} - \mathrm{K}\lambda^{\mathrm{t}} = 0$. Al sustituir λ en el esquema de interpolación de material en la Ecuación 3.11, se puede encontrar la sensibilidad de la función objetivo con respecto al cambio en el i-ésimo elemento como muestra la Ecuación 3.12.

$$\frac{\partial \mathrm{C}}{\partial \mathrm{x_i}} = -\frac{1}{2}u^{\mathrm{t}}\frac{\partial \mathrm{K}}{\partial \mathrm{x_i}}u = -\frac{1}{2}\mathrm{p}\cdot \mathrm{x_i}^{\mathrm{p}-1}\cdot \mathrm{u_i}^{\mathrm{T}}\mathrm{K_i}^{0}\mathrm{u_i} \tag{3.12}$$

De esta forma y durante cada iteración es posible evaluar el impacto que la variación de las densidades del material tiene sobre la función objetivo para maximizar la rigidez. Luego a partir de la Ecuación 3.12 es posible obtener el número de sensibilidad elemental.

$$\alpha_{\mathrm{i}} = -\frac{1}{\mathrm{p}}\frac{\partial \mathrm{C}}{\partial \mathrm{x_i}} = \frac{1}{2}\mathrm{u_i}^{\mathrm{T}}\mathrm{K_i}^{0}\mathrm{u_i} \quad , \quad \text{si } \mathrm{x_i} = 1 \tag{3.13}$$

$$\alpha_{\mathrm{i}} = -\frac{1}{\mathrm{p}}\frac{\partial \mathrm{C}}{\partial \mathrm{x_i}} = \frac{\mathrm{x_{min}}^{\mathrm{p}-1}}{2}\mathrm{u_i}^{\mathrm{T}}\mathrm{K_i}^{0}\mathrm{u_i} \quad , \quad \text{si } \mathrm{x_i} = \mathrm{x_{min}} \tag{3.14}$$

Finalmente, los valores de la Ecuación 3.13 y Ecuación 3.14 son los que se utilizan como parámetros de comparación para la actualización de la densidad relativa $\mathrm{x_i}$ en cada iteración.

3.2.3 Filtro de sensibilidades

Durante el análisis de sensibilidad, los elementos ponderados de baja densidad de material terminan perdiendo su importancia estructural y se eliminan durante iteraciones posteriores, siendo habitual que aparezcan una serie de inestabilidades numéricas (Jorge Hernández, 2021).

Las inestabilidades se pueden clasificar en tres grandes grupos, como se presenta a continuación:

a) Checkerboards: Patrones alternantes de regiones sólidas y vacías, se caracterizan por la alternancia

en la configuración estructural entre elementos sólidos y vacíos, sobreestimando la rigidez estructural y alejándose de la solución óptima. Se sugiere el uso de elementos con funciones de interpolación de orden superior para evitarla (Bendsøe y Sigmund, 2003).

b) Dependencia de malla: Se refiere a obtener diferentes topologías al usar distintas mallas de elementos finitos. Una malla más fina produce más elementos pequeños, mejorando la modelización y definición de límites en la estructura óptima (Bendsøe and Sigmund 2003).

c) Mínimos locales: obtención de diferentes soluciones para una misma discretización a elegir diferentes parámetros de inicialización.

En la Figura 3.1 se presenta un ejemplo de una estructura afectada por inestabilidad numérica señalada.

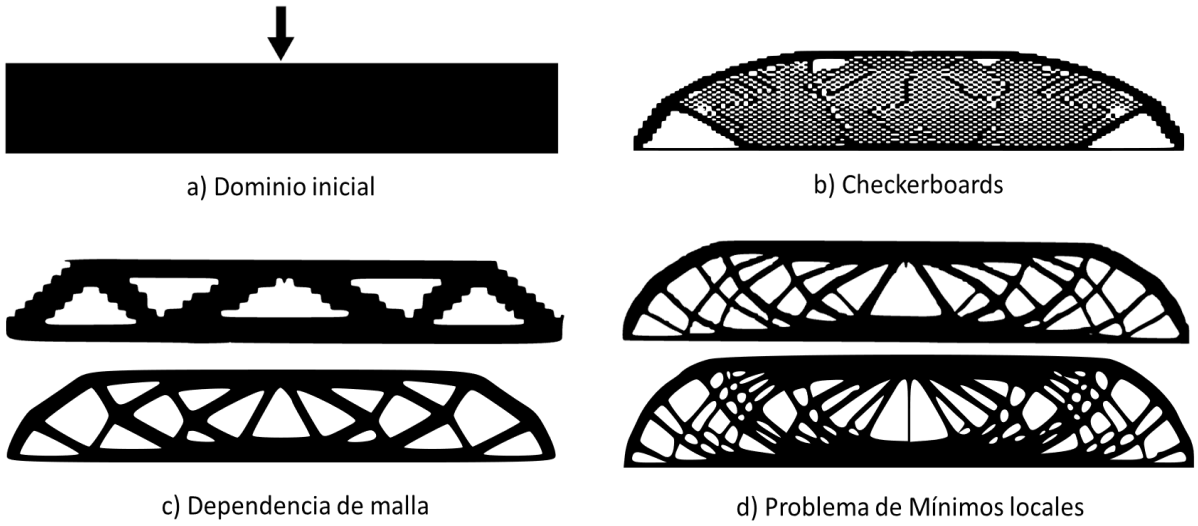

Figura 3.1 Ejemplo de estructura gravemente afectada por inestabilidades numéricas (Hernández & De, 2021).

La estrategia de filtrado sigue el enfoque de Huang y Xie (2010ª). Esta resuelve problemas de dependencia de malla y Checkerboards mediante el cálculo de un promedio ponderado de números de sensibilidad dentro de un radio r_{min}, utilizando un factor peso w_i, como especifica la Ecuación 3.15.

$$\alpha_j{}^n = \sum_{i=1}^{M} w_i \alpha_i{}^e \tag{3.15}$$

M corresponde a la cantidad de elementos conectados a través del j-ésimo nodo y w_i es el factor de peso correspondiente al i-ésimo elemento como se presenta en la Ecuación 3.16 y Ecuación 3.17.

$$w_i = \frac{1}{M-1}\left(1 - \frac{r_{ij}}{\sum_{i=1}^{M} r_{ij}}\right) \tag{3.16}$$

$$\sum_{i=1}^{M} w_i = 1 \tag{3.17}$$

El factor de peso anterior les asigna una mayor importancia a los números de sensibilidad de cada elemento según su cercanía al nodo en estudio. Donde r_{ij} es la distancia desde el centro del i-ésimo elemento y el j-ésimo nodo, tal cómo se muestra en Figura 3.2.

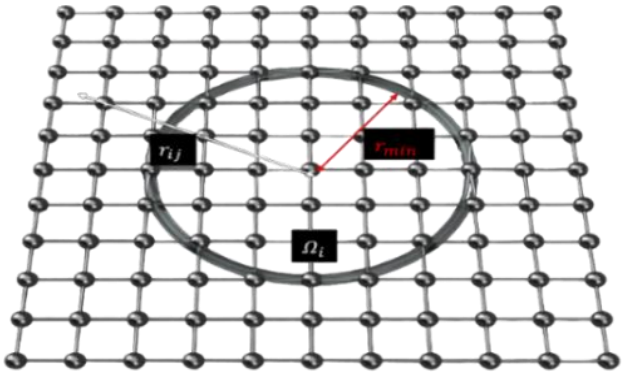

Figura 3.2 Filtro de sensibilidades.

A través de r_{min} se define un subdominio de diseño Ω_i. Se dibuja un círculo de radio r_{min} centrado en el centroide del i-ésimo elemento, asegurando que solo se consideren los números de sensibilidad de los nodos dentro de Ω_i. La zona de influencia Ω_i queda definida por un círculo de radio $r_i = r_{min}$ centrado en el i-ésimo elemento. El valor del r_{min} se elige de forma de cubrir más de un elemento. La Ecuación 3.18 permite obtener la nueva sensibilidad nodal.

$$\alpha_i = \frac{\sum_{j=1}^{K} w(r_{ij})\alpha_j{}^{n}}{\sum_{j=1}^{K} w(r_{ij})} \tag{3.18}$$

Donde K es el número total de nodos contenidos en el subdominio Ω_i y $w(r_{ij})$ es el factor de peso definido en la Ecuación 3.19.

$$w(r_{ij}) = r_{min} - r_{ij} \tag{3.19}$$

3.3 Postprocesado de imagen

Formulaciones como la de Bendsøe y Sigmund (2003) han ganado popularidad en diseño generativo. Sin embargo, la optimización topológica basada en densidad enfrenta limitaciones como baja precisión en los límites estructurales y el uso de elementos finitos de bajo orden. La falta de soluciones integrales ha llevado a depender del postprocesado manual, reduciendo la eficiencia. La subjetividad en las decisiones del diseñador también afecta la eficacia. Superar estas limitaciones es crucial para avanzar hacia enfoques más eficientes y precisos, alineados con normativas locales.

3.3.1 Esqueletización

La esqueletización es un proceso iterativo que reduce una imagen binaria a un grosor de un píxel, preservando las líneas y conexiones que caracterizan la geometría original, manteniendo su topología.

Los algoritmos de esqueletización pretenden extraer el eje medio de un objeto, que se define como el conjunto de puntos que tienen más de un punto cercano en el límite del objeto (Harry Blum,1967).

Las metodologías más comúnmente utilizadas son las basadas en adelgazamiento, Transformada de distancia, Métodos Geométricos, Diagramas de Voronoi y funciones de campo (Reniers, 2009).

3.3.2 Algoritmos de adelgazamiento

El adelgazamiento se puede definir como la reducción de una imagen al tamaño mínimo, o hasta el punto en que la imagen conserve los puntos necesarios para el procesamiento (Bansal & Kaur, 2016).

Las condiciones de adelgazamiento se aplican sucesivamente a cada píxel de una imagen por los algoritmos correspondientes, eliminando aquellos que cumplen con dichas condiciones y manteniendo los que no. De este modo, los píxeles que no son eliminados durante este proceso forman el esqueleto de la imagen (García Arellano, 2007).

T. Y. Zhang y C. Y. Suen (1984), proponen un algoritmo de esqueletización que busca obtener el esqueleto de objetos en imágenes binarias mediante un proceso de dos etapas. La fase uno consiste en eliminar los pixeles de la imagen a partir las direcciones que se muestran en la Figura 3.3.

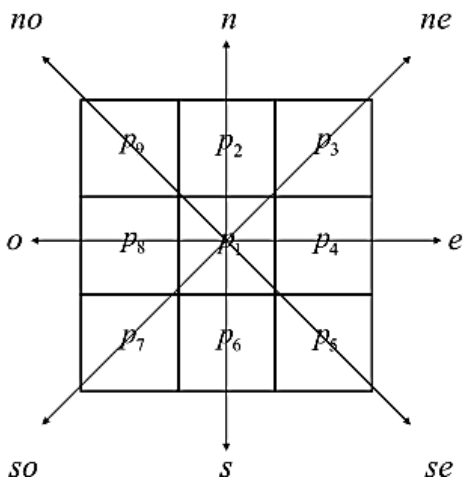

Figura 3.3 Direcciones utilizadas para el trazo de barrido algoritmo de Zhang y Suen. Fuente: (García Arellano, 2007)

Las condiciones prescritas para el píxel p1 dentro de la matriz de la imagen binaria son presentadas en la Ecuación 3.20.

$$\begin{aligned} &(a)\ 2 \leq \mathrm{B}(\mathrm{p}_1) \leq 6 \\ &(b)\ C(p_1) = 1 \\ &(c)\ p_2 \cdot p_4 \cdot p_6 = 0 \\ &(d)\ p_4 \cdot p_6 \cdot p_8 = 0 \end{aligned} \tag{3.20}$$

Donde, $C(p1)$ es el conteo de transiciones de 0 a 1 en la secuencia ordenada de píxeles *p2, p3, ..., p8, p9, p2*. Mientras que $\mathrm{B}(p_1)$ es la cantidad de píxeles circundantes a p1 que no son cero, mientras las condiciones prescritas para p_1 en la fase siguiente son presentadas en Ecuación 3.21.

$$\begin{aligned} &\text{(a)}\ 2 \leq B(p_1) \leq 6 \\ &\text{(b)}\ C(p_1) = 1 \\ &\text{(c')}\ p_2 \cdot p_4 \cdot p_8 = 0 \\ &\text{(d')}\ p_2 \cdot p_6 \cdot p_8 = 0 \end{aligned} \tag{3.21}$$

Inicialmente, se aplican las condiciones de la Ecuaciones 3.20 a todos los pixeles de la imagen, siendo marcados aquellos píxeles que deben ser eliminados. Luego, se eliminan los píxeles marcados y se aplican las condiciones de las Ecuaciones 3.21 a la imagen resultante, identificando nuevos píxeles a eliminar. Finalmente, se eliminan los píxeles identificados en el paso anterior. Este proceso se repite iterativamente hasta que no haya más píxeles por eliminar, como se muestra en la Figura 3.4.

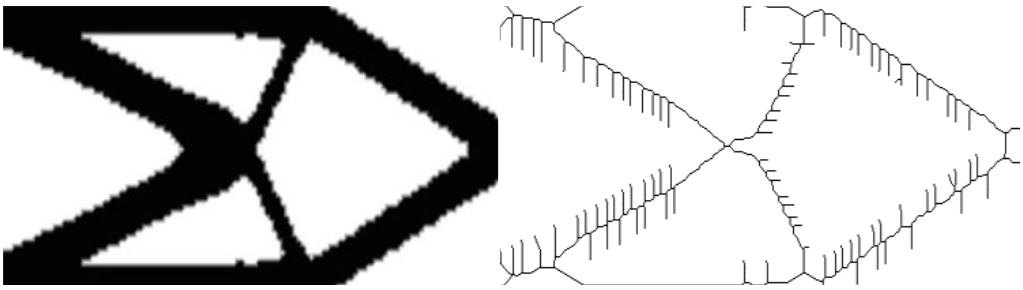

Figura 3.4 Esqueleto con excesivas ramificaciones.

Es importante señalar que este algoritmo exhibe ciertas limitaciones al ser aplicado en imágenes afectadas por ruido. Contrariamente a reducir el nivel de ruido, su aplicación puede dar lugar a un aumento en la presencia de ruido (García Arellano, 2007).

3.3.3 Poda de Esqueleto con Partición de Contorno

La poda de esqueletos es crucial en el procesamiento, proporcionando una representación simplificada y estructurada de la geometría de un objeto en una imagen. Su propósito es mejorar la calidad del esqueleto al eliminar información redundante o no deseada, especialmente en bordes ruidosos.

3.3.4 Contorno aproximado y parametrización de la curva de Bézier

Andrés Solís Montero y Jochen Lang (2012) proponen un método basado en la aproximación del contorno y la transformada del eje medial (IMA), conservando propiedades topológicas sin desplazar el eje medial ni acortar ramas. Este enfoque se demuestra robusto frente al ruido en el límite. El objetivo es minimizar diferencias cuadráticas en puntos de control utilizando curvas de Bézier cúbicas.

3.3.5 Poda Final

La tercera etapa del algoritmo de Solís Montero y Lang (2012) se enfoca en la poda final del esqueleto obtenido en la etapa anterior, utilizando el Eje Medial Entero (IMA). Se identifican puntos de bifurcación (b_p), con más de dos píxeles adyacentes y puntos de fin de rama (e_p) con un solo píxel adyacente. El criterio de poda final queda determinado por la expresión mostrada en Ecuación 3.22.

$$\left|e_p - b_p\right|_2 \leq s \cdot \left|f - b_p\right|_2 \tag{3.22}$$

La ecuación establece que una rama se elimina si la distancia entre el punto de fin de rama y el punto de bifurcación es menor o igual a (s) veces la distancia al contorno más cercano (f). El factor de escala (s) ajusta la amplitud de la regla, controlando cuánto puede alejarse el punto de fin de rama respecto al punto de bifurcación en relación con la distancia al contorno más cercano.

3.3.6 Localización de vértices

El proceso de esqueletización de una imagen, después de haber generado el esqueleto que representa las rutas o caminos a lo largo de la imagen, requiere identificar los vértices o intersecciones en ese esqueleto (Mendoza San Agustín et al., 2016).

Los vértices, donde convergen tres o más caminos, se identifican en una matriz binaria que representa el esqueleto, donde los píxeles blancos son 1 y los píxeles negros son 0. En la Figura 3.5.se presenta una matriz binaria que muestra la vecindad de los vértices de ramificación (Branch-Point) y extremo (End-Point).

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & V & V & 0 & 0 \\ 0 & 0 & 0 & V & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & V & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 3.5 Vecindad de vértices Branch-Point y End-Point. (Mendoza San Agustín et al., 2016)

La matriz se recorre mediante una submatriz, desplazándose de la esquina superior izquierda a la esquina inferior derecha con movimientos unitarios, avanzando un píxel a la vez hacia la derecha. En la Figura 3.6 se presenta la submatriz de orden tres utilizada en la identificación de vértices.

$$\begin{pmatrix} P11 & P12 & P13 \\ P21 & V & P23 \\ P31 & P32 & P33 \end{pmatrix}$$

Figura 3.6 Submatriz de orden 3x3 utilizada en identificación de vértices. (Mendoza San Agustín et al., 2016)

Cuando la posición central de esta submatriz (P_{22}) se coloca sobre un píxel blanco (valor uno), se suman los valores de las posiciones adyacentes a este píxel central. Si la suma de esos valores es igual o mayor a tres, se marca el píxel central como un vértice, según lo presentado en Ecuación 3.23.

$$P_{11} + P_{12} + P_{13} + P_{21} + P_{23} + P_{31} + P_{32} + P_{33} \geq 3 \tag{3.23}$$

Si en un mismo vértice concurren varios caminos, se genera una vecindad de vértices. En la Figura 3.7 se muestra la vecindad de vértices en una submatriz y un vértice único de la vecindad.

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & V & V \\ 0 & 0 & V \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & V \\ 0 & 0 & 1 \end{pmatrix}$$

a) Vecindad de vértices en submatriz b) Vértice único de la vecindad

Figura 3.7 Vecindad de vértices en submatriz y vértice único de la vecindad. (Mendoza San Agustín et al., 2016)

Finalmente, para obtener un solo vértice en una vecindad se calcula el centroide. del número de vértices a una distancia i en la dirección x y y respectivamente. De esta forma, la vecindad de vértices se sustituye por un nuevo vértice único resultando una poda como la mostrada en Figura 3.8.

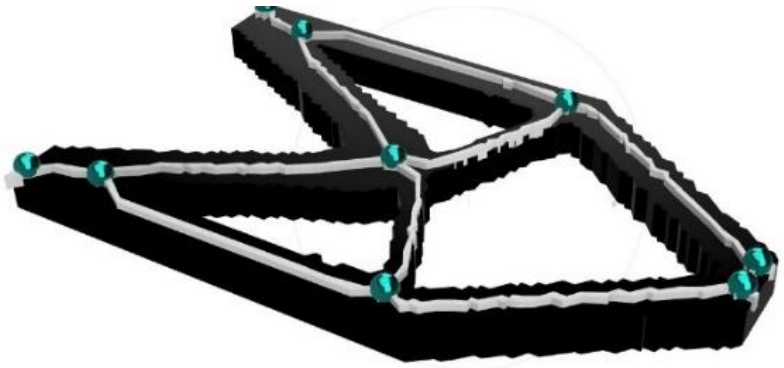

Figura 3.8 Geometría de bordes irregulares sometida a algoritmo de poda de esqueleto

3.3.7 Conectividad de vértices y vectorización de esqueleto

El siguiente paso es localizar los puntos de inicio y fin de los caminos en la cuadrícula para sustituirlos por líneas rectas. Se utiliza una técnica que coloca una submatriz de tres por tres sobre un punto de inicio y observa los puntos circundantes con valor uno, indicando la dirección de los caminos conectados al vértice (Mendoza San Agustín et al., 2016).

Se desplaza una submatriz de tres por tres centrada en puntos con valor uno alrededor del vértice inicial. Si se encuentra otro vértice en esta nueva posición, se marca el recorrido del camino. Este proceso se repite para cada punto con valor uno alrededor del vértice, identificando dónde comienzan y terminan los caminos para su representación con líneas segmentadas y definición de conectividad. Una vez hallados los vértices inicial y final, se define la conectividad del camino de la Figura 3.9.

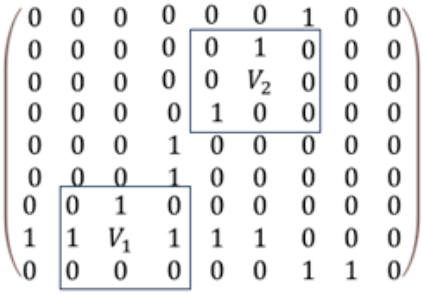

Figura 3.9 Matriz de conectividad vértices e identificación de caminos. (Mendoza San Agustín et al., 2016)

3.3.8 Construcción de Grafo a partir de Cadena de Píxeles

Gedig (2010) introduce el concepto de grafo relacional atribuido (ARG) como una representación para descripciones estructurales. Dicho grafo G se expresa mediante la notación $G = (V, E, AV, AE, \alpha V, \alpha E)$, donde V corresponde a los vértices del esqueleto y E representa el conjunto de los segmentos que conectan pares de vértices (Douglas Brent West et al, 2001), AV es el conjunto de atributos en vértices, mientras que AE se refiere a los atributos para las aristas. Las funciones $\alpha V, \alpha E$ asignan atributos específicos a cada vértice y arista, como fuerzas o condiciones de contorno. En la Figura 3.10 se presenta la representación del grafo propuesto por Gedig.

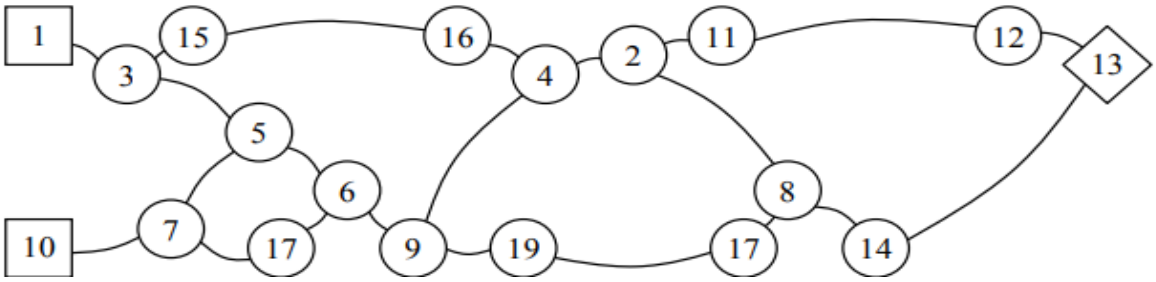

Figura 3.10 Grafo relacional de esqueleto vectorizado (Gedig, 2010).

Los nodos en forma de cuadrados indican restricciones de fijación, mientras que el nodo en diamante refleja la aplicación de una fuerza. Los vértices se presentan como círculos y se dividen en cuatro tipos, conteniendo datos sobre las coordenadas y los valores de condiciones límites.

Por otra parte, Sarma Lowhikana, Mallikarachchia y Heratha (2023) en su artículo "Design-Informed Generative Modelling using Structural Optimization", introducen un enfoque similar al de Gedig (2010), pero con ciertas mejoras. Estos autores señalan que la conversión directa de las aristas del grafo en miembros del marco no es práctica debido a la creación de miembros cortos e ineficaces.

Algoritmo de longitud mínima

La estructura-esqueleto puede incluir miembros muy cortos, lo cual complica la fabricación y estabilidad de esta. El algoritmo aborda esto contando la longitud de las curvas y eliminando aquellas que no cumplen con la longitud mínima requerida tras definir una relación para contraerlas.

Yin, Ge, Xiao y Cirak (2020) desarrollaron un enfoque basado en grafos ponderados no dirigidos para fusionar elementos en esqueletos de cadena, utilizando un proceso de colapso de aristas influenciado por la proximidad y la cantidad de elementos interconectados. Este método permite la fusión eficaz de nodos cercanos y la eliminación de ramas redundantes, enfocándose en mantener solo elementos estructurales significativos. Además, se eliminan ramas que conducen a elementos con tensión cero, vinculadas a límites de Neumann cero. Este proceso garantiza que solo se conserven miembros con una longitud adecuada. En la Figura 3.11 se muestra el proceso de eliminación de ramas y fusión de nodos en la estructura esqueleto.

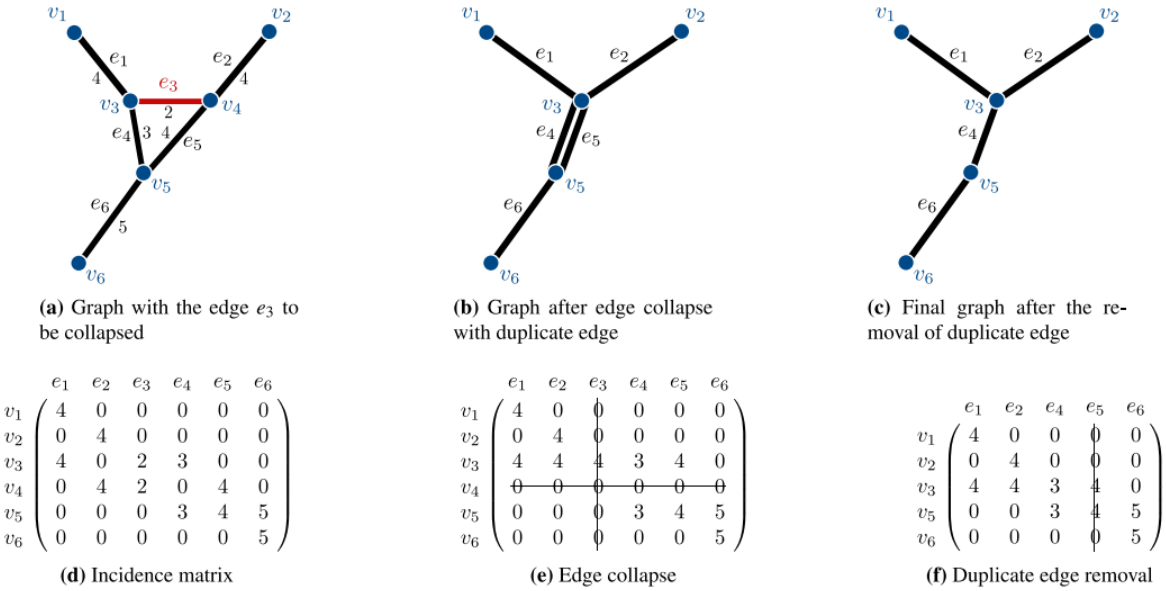

Figura 3.11 Colapso de aristas basado en proximidad y cantidad de pixeles interconectados. (Yin et al., 2020)

3.4 Conclusiones

En este capítulo se presentó la formulación matemática del método Soft Kill BESO, abordando la función objetivo y sus restricciones, y culminando con la resolución del problema de optimización mediante multiplicadores de Lagrange. Se enfatizo el papel del filtro de sensibilidades para superar inestabilidades numéricas, asegurando resultados que son estructuralmente factibles y optimizados.

Se exploro en profundidad los métodos de esqueletización y poda, técnicas esenciales para la interpretación y refinamiento geométrico post-optimización, permitiendo una transición fluida hacia representaciones estructurales más precisas y manejables. La implementación de un grafo relacional atribuido (ARG) se discute como un medio para capturar y representar la esencia estructural del modelo optimizado, proporcionando una plataforma robusta para análisis y aplicaciones prácticas.

CAPÍTULO 4: IMPLEMENTACIÓN COMPUTACIONAL

4.1 Introducción

En el capítulo actual se aborda la descripción detallada de la rutina implementada en MATLAB®, abarcando los procesos gobernantes, así como algunas aclaraciones de los comandos y rutinas implementados para llevar a cabo una reconstrucción morfológica basada en elementos finitos tipo barra y tetraédricos. Adicionalmente, se presenta un diagrama de flujo que esquematiza parte importante del conjunto de rutinas implementadas, junto con la especificación de las limitaciones inherentes del algoritmo.

4.2 Rutina implementada

La rutina implementada en MATLAB® utiliza un enfoque de reconocimiento, filtrado y segmentación de cuencas y esqueletos binarios para la reconstrucción de imágenes topológicas, aplicándolas a armaduras en estructuras optimizadas. La eficacia de este enfoque se pone de manifiesto mediante su comparación con métodos preexistentes en posteriores capítulos.

Estas variables abarcan aspectos como la discretización por cada eje, la imposición de restricciones asociadas, estados de cargas respectivos, la generación de agujeros y zonas no optimizables, el tipo de material, sus propiedades físicas y mecánicas (módulo de elasticidad, módulo de Poisson), así como variables avanzadas relativas al algoritmo de optimización topológica a utilizar, cómo lo son la fracción volumétrica objetiva, parámetros de penalización, criterios de convergencia, suavidad, entre otros La interfaz gráfica desarrollada, presentada en el Anexo 5.1, permite al usuario interactuar directamente con estos parámetros y variables, facilitando la definición y manipulación de condiciones de carga y restricciones en una retícula estructural.

4.3 Comando optimizador

Este comando implementa el algoritmo de optimización topológica *Soft Kill* BESO, desarrollado por Huang y Xie (2010), basado en el código abierto de 99 líneas *de* Sigmund (2001) y diseñado para estructuras en tensión plana con materiales isótropos. A continuación, se describe el proceso utilizado.

1) Utilizar una malla fija de elementos finitos cuadriláteros en un dominio rectangular.
2) Establecer los parámetros del esquema BESO, incluyendo r_min (parámetro del filtro de sensibilidades) y la tasa de evolución (ER).
3) Realizar un análisis de elementos finitos en el dominio definido.
4) Calcular el número de sensibilidad de cada elemento y se aplica dicho filtro.
5) Determinar el volumen objetivo para la siguiente iteración.
6) Actualizar la densidad de los elementos, siendo 1 para elementos sólidos y x_min vacío.
7) Iterar los pasos anteriores hasta alcanzar la convergencia

La función principal, *Softkillbeso*, proporciona la matriz de densidades relativas de los elementos y el vector de desplazamientos globales (vector U). Además, se utilizan funciones adicionales:

1) 'FE': Realiza el análisis de elementos finitos.
2) 'Check': Entrega la sensibilidad de cada elemento después de aplicar el filtro.
3) 'ADDDEL': Determina los elementos a remover/agregar basado en tolerancias establecidas.

Es importante destacar que la rutina antes mencionada se basa en el trabajo previo de Oviedo (2020), y para detalles adicionales, se remite al ANEXO 9.

4.4 Comando suavizador

4.4.1 Módulo de Preparación de Imágenes para Análisis de Optimización Topológica mediante MATLAB®

Este módulo encapsula procedimientos de procesamiento de imágenes diseñados específicamente para el análisis de resultados derivados del método de optimización topológica *Soft Kill* BESO. El objetivo es optimizar la preparación y evaluación de imágenes para extraer de manera precisa las características estructurales relevantes, facilitando una interpretación detallada y técnica del diseño optimizado, mayor detalle de las funciones implicadas se entrega en el Anexo 5.4.

La imagen proporcionada corresponde a una representación ráster de una viga en voladizo de 40 mm de alto y 160 mm de largo, sometida a una carga puntual de 1 Newton en su extremo. Esta estructura

se presenta durante una fase de optimización topológica, específicamente utilizando el método *Soft Kill* BESO. En la Figura 4.1 se muestra un ejemplo de topología obtenida, seleccionada por su diseño ramificado complejo, con el fin de evaluar la robustez del algoritmo empleado.

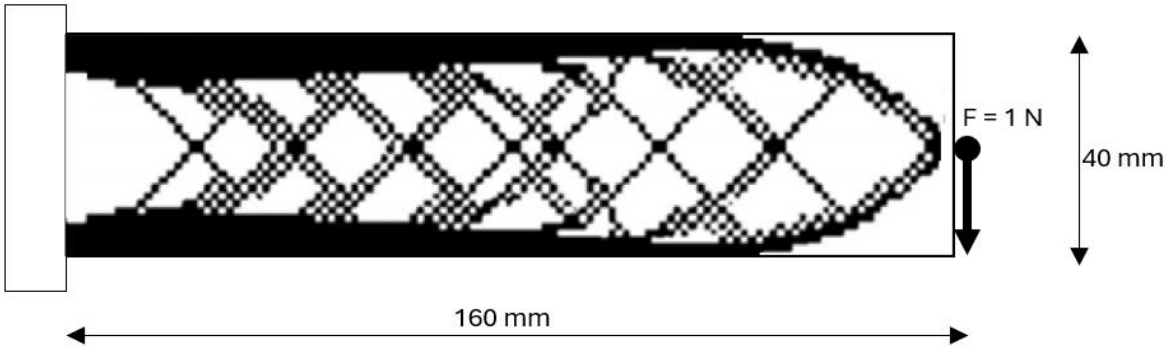

Figura 4.1 Ejemplo de topología extraída a partir de algoritmo Soft Kill BESO (Bendsøe & Sigmund, 2002).

El flujo de procesamiento desde la optimización topológica hasta la obtención de la imagen final umbralizada fue sintetizado mediante la Ecuación 4.1.

$$[\mathrm{BW}] = \mathrm{imbinarize}(\mathrm{imgaussfilt}(\mathrm{bwdist}(\mathrm{I}_{\mathrm{softkillbeso}}, '\mathrm{parameters}'))) \quad (4.1)$$

Donde $\mathrm{I}_{\mathrm{softkillbeso}}$ representa la imagen ráster inicial obtenida del algoritmo de optimización topológica *Soft Kill* BESO. (parameters) incluye los valores personalizados para las funciones aplicadas, como los parámetros del filtro gaussiano. La función $bwdist$ calcula la transformada de distancia de la imagen binaria invertida. La función i$mgaussfilt$ aplica un filtro gaussiano para suavizar la imagen resultante, mientras que la función $imbinarize$ umbraliza la imagen suavizada para destacar las estructuras de interés, resultando en una imagen binaria que destaca las áreas críticas de la estructura optimizada. En la Figura 4.2 se muestra el resultado de emplear estas funciones, detallando el proceso de transformación y optimización de la imagen.

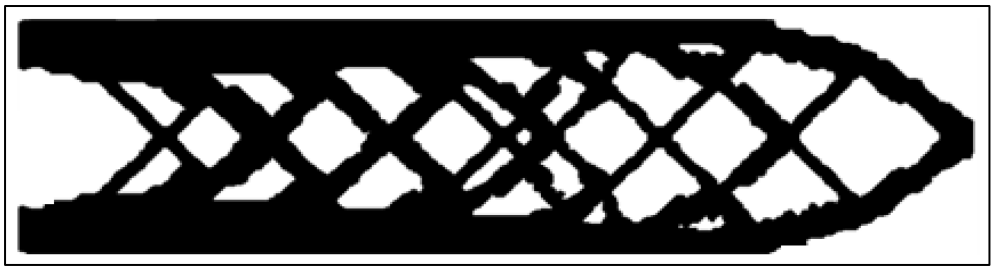

Figura 4.2 Ejemplo de umbralización.

4.4.2 Segmentación de Cuencas

Posterior a la fase de preprocesado, sigue la fase de la identificación del contornos y cavidades. Para ello se procede por medio del algoritmo de segmentación de cuencas Watershed, disponible entre las herramientas de procesamiento de imágenes propias de MATLAB®, como se indica en la Ecuación 4.2 y Ecuación 4.3.

$$[\mathrm{A}] = \mathrm{bwdist}(\mathrm{BW}, 'euclidean') \tag{4.2}$$

$$[\mathrm{L}] = \mathrm{Watershed}(\mathrm{A}, \mathrm{conn}) \tag{4.3}$$

Donde (A) es la transformada de distancia euclidiana de la imagen binaria (BW), en este caso la imagen posterior a la operación de umbralización, mientras (con) especifica la conectividad que se utilizará en el cálculo de Watershed (en nuestro caso de valor 8). De esta forma la máscara resultante queda como la mostrada en la Figura 4.3.

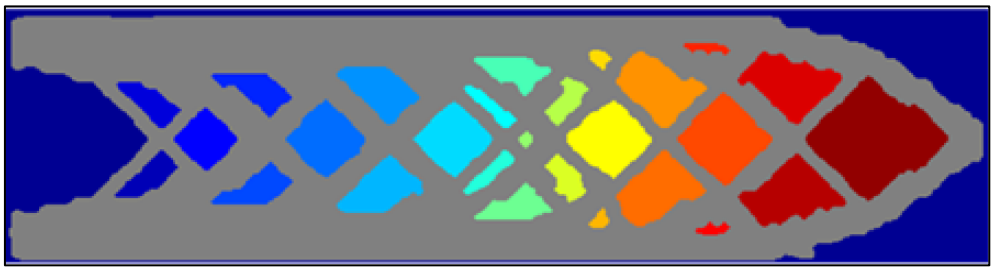

Figura 4.3 Ejemplo de segmentación de cuencas tras aplicación de algoritmo Watershed.

Relativo a la obtención del perímetro de las cavidades interiores y exteriores. Se extrae el perímetro de dicha imagen binaria utilizando la función presentada en la Ecuación 4.4.

$$[\mathrm{BW2}] = \mathrm{bwperim}(\mathrm{BW}, 'conn = 8') \tag{4.4}$$

Posteriormente, con el propósito de distinguir la geometría aditiva de contorno de la geometría de sustracción de cavidades, se emplea la función destinada al cálculo del límite bidimensional exterior mostrada en la Ecuación 4.5.

$$[\mathrm{k}] = \mathrm{boundary}(\mathrm{xi}, \mathrm{yi}, "s = 0.98") \tag{4.5}$$

En la Figura 4.4 se ilustra un ejemplo de identificación de contornos generado a partir de boundary.

Figura 4.4 Identificación de elementos de contorno (rojo) en ejemplo de implementación computacional.

Se utiliza un factor de encogimiento del 0.98 para capturar con mayor precisión el perímetro exterior de la figura. Posteriormente, se realiza una dilatación de este perímetro para sustraerlo de las regiones perimetrales obtenidas mediante el proceso de Watershed. Este método permite obtener cavidades internas diferenciadas del contorno perimetral de la figura. En la Figura 4.5 se muestra en detalle este proceso, ilustrando cómo el factor de encogimiento y la dilatación se aplican para distinguir claramente las cavidades internas del perímetro exterior, facilitando el análisis estructural.

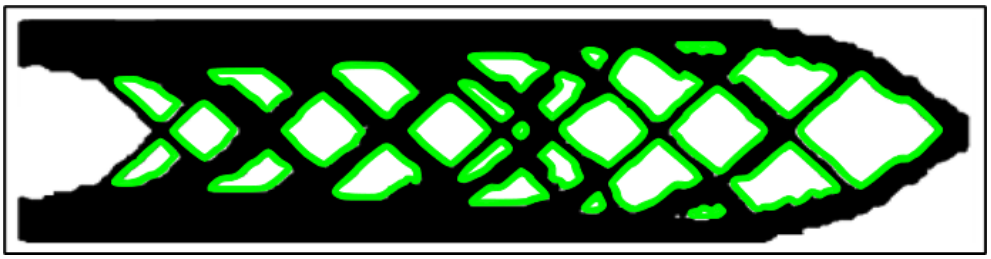

Figura 4.5 Identificación de cavidades interiores (verde) para ejemplo de implementación computacional.

En esta fase de cuencas perimetrales interiores existen cavidades internas que requieren ser eliminadas, este proceso se rige por una tolerancia establecida en proporción al área total definida por el usuario. La función de filtro umbral diseñada para este propósito es expresada en la Ecuación 4.6.

$$[\mathrm{BW}] = \mathrm{Basic\_filt\_umbral\_area}(\mathrm{BW2}, \text{"filtro} = 0.5\%\text{"}) \tag{4.6}$$

4.4.3 Ordenación y estructuración de datos capturados

La organización y estructuración de los datos capturados se realiza mediante una función, indicada en la Ecuación 4.7 diseñada para encontrar un camino vectorizado a lo largo del perímetro en un mapa dado. Se emplea una estrategia de búsqueda en profundidad (DFS), que calcula distancias y toma

decisiones basadas en ellas para formar caminos a lo largo del contorno de píxeles de cada cuenca, como se muestra en la Figura 4.6. El algoritmo simula los nodos de un grafo, expandiendo de manera recurrente cada píxel, avanzando desde el padre ($Point_{init}$) al hijo y regresando al predecesor cuando no hay más píxeles por visitar en dicho camino, proporcionando un camino coherente del contorno, tal como se presenta en la Ecuación 4.7.

$$[\text{Vectorfinal}] = \text{deep}_{\text{search}}(\text{BW2}, \text{Point}_{\text{init}}) \tag{4.7}$$

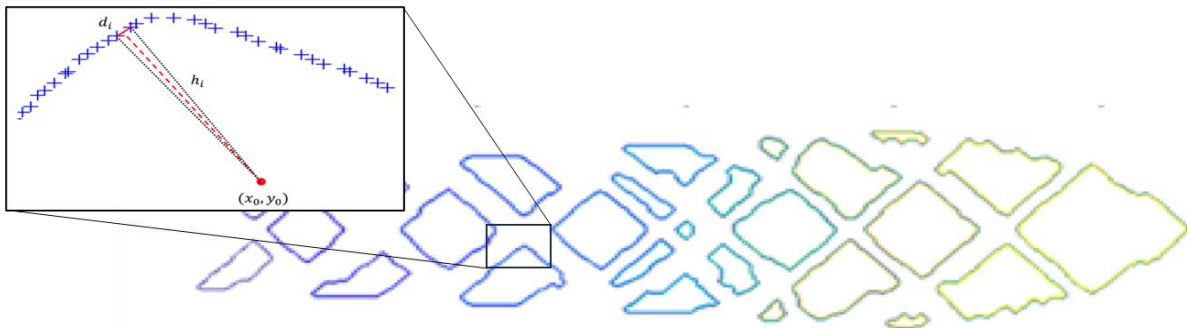

Figura 4.6 Segmentación y ordenación de cavidades interiores para ejemplo de implementación computacional.

De manera análoga, Matlab ofrece en su librería para procesamiento de imagen la herramienta de procesado de imágenes la función mostrada en la Ecuación 4.8.

$$\text{contour\{i\}} = \text{bwtraceboundary}(\text{BW}, [\text{P}], \text{fstep}, \text{conn}, \text{m}, \text{dir}) \tag{4.8}$$

El punto (P) se define por sus posiciones horizontal y vertical en el contorno del objeto, marcando el inicio del trazado. La variable (fstep) determina la dirección se comenzará a buscar el píxel contiguo.

4.4.4 Suavizado Automatizado de Contornos Pixelados

Finalizada esa etapa, se efectúa un suavizado mediante la función presentada en la Ecuación 4.9 obtenida a partir de la investigación de Le Tarnec y Garcia (2012) para suavizado autónomo.

$$\text{Vs\{i\}} = \text{smoothn}(\{\text{Vx}, \text{Vy}\}, '\text{robust}') \tag{4.9}$$

Donde (Vx) y (Vy) representan las coordenadas del contorno previamente ordenadas. Está función basada en un método de mínimos cuadrados penalizados permite mediante un suavizado automatizado para conjuntos de pixeles muestreados, suavizar rápidamente los datos en varias dimensiones mediante la transformada discreta del coseno (Le Tarnec & Garcia, 2012). En la Figura 4.7 se presenta un ejemplo detallado del suavizado robusto aplicado a los conjuntos de píxeles.

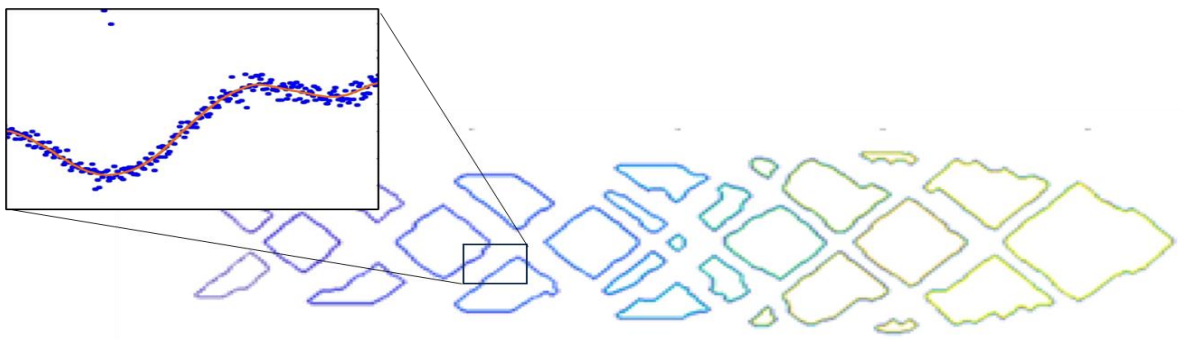

Figura 4.7 Suavizado robusto para conjuntos de pixeles muestreados estructurados pertenecientes a una cuenca.

La eliminación de la naturaleza ruidosa de los bordes pixelados y la mejora de la legibilidad de los contornos da como resultado una primera reconstrucción que, aunque imprecisa, presenta características distintivas y ha sido suavizada en nuestro dominio optimizado topológicamente.

4.4.5 Obtención de puntos característicos

Para la obtención de los puntos característicos de la estructura, se hace uso de la función dada en la Ecuación 4.10.

$$[\mathrm{L}, \mathrm{R}, \mathrm{k}] = \mathrm{curvature}(\mathrm{X}) \tag{4.10}$$

Esta función calcula la curvatura de una curva bidimensional representada por un conjunto de puntos. Proporciona tres salidas principales: la longitud acumulada de la curva (L), el radio de curvatura (R), y el vector de curvatura (k) que contiene las componentes de la curvatura en x, y, z.

Con los datos obtenidos desde la Ecuación 4.10 se aplica un filtro para detectar los puntos de menor curvatura, utilizando un umbral específico. La Figura 4.8 presenta el resultado del filtro de curvatura

de contornos basado en umbrales, mostrando cómo se identifican y destacan las características principales de los contornos procesados.

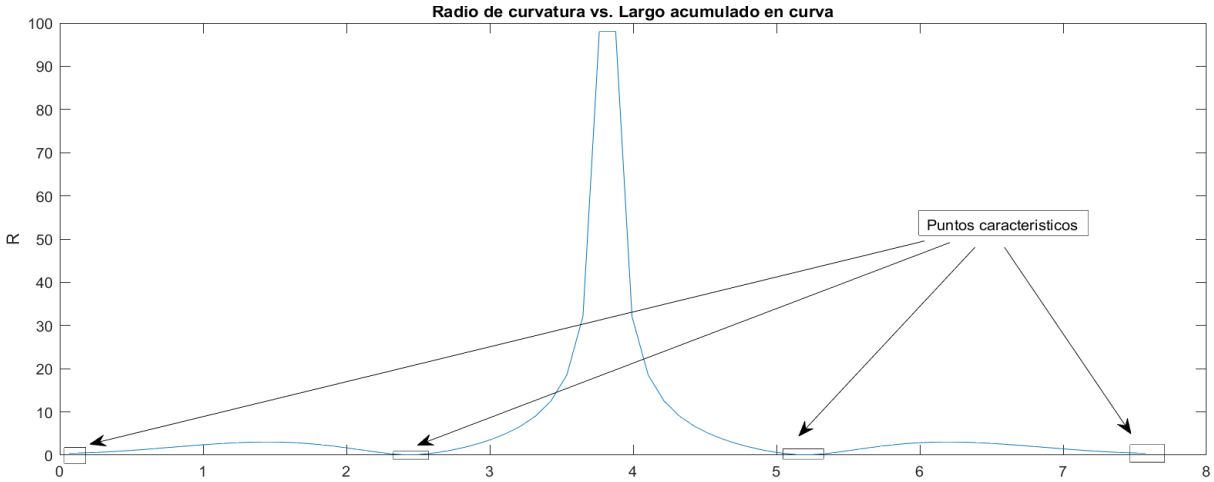

Figura 4.8 Filtro de curvatura de contornos basado en umbrales.

En esta etapa, se lleva a cabo un proceso de agrupación mediante la aplicación de clústeres a puntos cercanos. Primero, se recopilan todos los puntos de prueba identificados como vértices característicos de la cavidad objeto de estudio. Posteriormente, se calculan los centroides característicos de cada región de puntos vértices. Este procedimiento tiene como objetivo reunir puntos que pertenecen a la misma característica u objeto, contribuyendo a reducir la sensibilidad del algoritmo. Se repite este proceso para cada cavidad previamente segmentada, culminando en la identificación de vértices. La Figura 4.9 ilustra cómo se agrupan y calculan los centroides como vértices característicos.

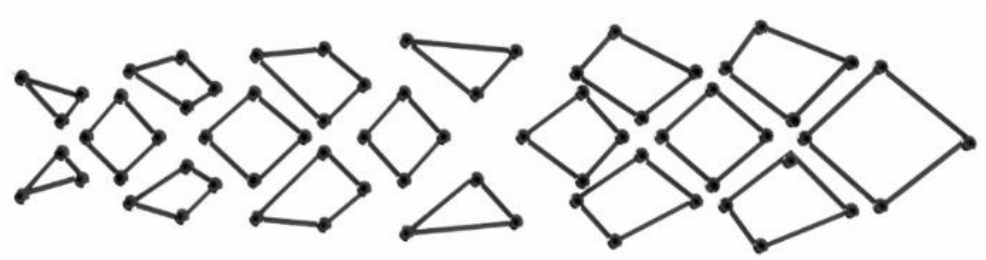

Figura 4.9 Identificación de vértices característicos del conjunto de cavidades.

4.4.6 Reconstrucción por B-spline y fillet

Para suavizar y lograr una interpolación precisa de los vértices de las cavidades generadas, se utiliza la función *spcrv* en MATLAB® con grado igual a tres, según se expresa en la Ecuación 4.11.

$$\text{values\_spline} = \text{spcrv}(c, 'k = 3', \text{maxpnt}) \tag{4.11}$$

Esta función utiliza splines cúbicos, conectando los puntos de entrada y proporcionando una representación continua de las cavidades. La expresión utiliza el vector de coeficientes de B-spline (c), asociado a los puntos de control de la cavidad, y el parámetro (k), que denota el orden de B-spline. En este caso, con k de tres, se emplea un polinomio cuadrático en cada segmento. La Figura 4.10 presenta un ejemplo de reconstrucción automatizada por interpolación B-spline.

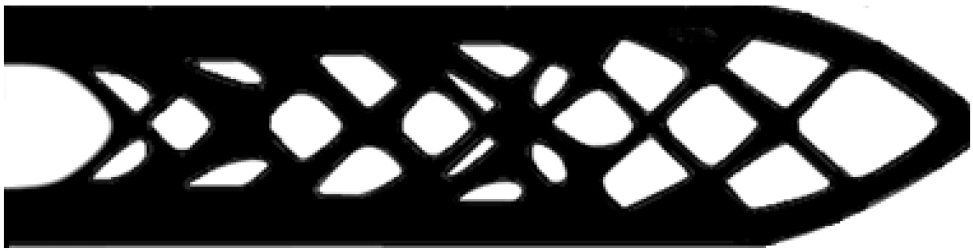

Figura 4.10 Ejemplo de reconstrucción automatizada por interpolación B-spline.

Las curvas y superficies B-spline son opciones factibles, pero su aplicación resulta costosa en el ámbito de la fabricación convencional. En este contexto, se desarrolló la función presentada en la Ecuación 4.12 para incorporar redondeos en las esquinas.

$$[\text{Arc}_x, \text{Arc}_y] = \text{RoundedCornerPolygon}(x_{ps}, y_{pts}, r, np_{arco}, db) \tag{4.12}$$

Esta función realiza el fileteado de una curva con un radio de entrada (r), utilizando (np) puntos por arco de filete. La Figura 4.11 presenta la implementación de dicha función, mostrando cómo se integran los redondeos en las esquinas de las geometrías para mejorar su fabricación y rendimiento.

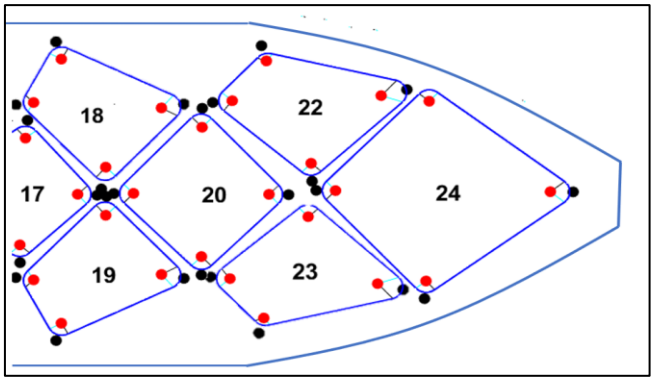

Figura 4.11 Ejemplo de interpolación B-splines de cavidades interiores y contorno.

La Figura 4.12 presenta una reconstrucción automatizada del dominio optimizado una vez que se han identificado los vértices característicos. Este proceso utiliza la función de redondeo de vértices para un radio especificado en las cavidades interiores y el contorno.

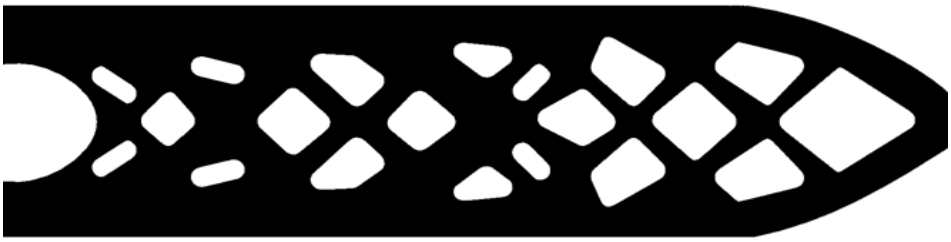

Figura 4.12 Ejemplo de reconstrucción automatizada por herramienta de redondeo de vértices para radio especificado en cavidades interiores y contorno

Con los resultados de la reconstrucción generada se procedió a mallar la estructura suavizada mediante la función *holes_generator*, encargada de generar una malla de elementos y nodos en 3D. Esta función es presentada en la Ecuación 4.13.

$$[\text{NODES}, \text{ELEMENTS}] = \text{holes\_generator}(\text{i}_{\text{holes}}, \text{grosor}) \tag{4.13}$$

Donde NODES corresponde a las coordenadas de los nodos de la malla generada, y ELEMENTS corresponde a la conectividad de los vértices representada a través de una matriz que define los elementos finitos de la malla, mientras que (i_{holes}) es un vector de celdas que almacena toda la información de las cavidades suavizadas que serán eliminadas para generar el respectivo mallado.

Para la fase final y mediante las herramientas de MATLAB®, es posible exportar la estructura final a un fichero STL utilizando la secuencia de Ecuaciones 4.14 hasta 4.16.

$$\text{fv.vertices} = \text{nodes}' \tag{4.14}$$

$$\text{fv.faces} = \text{elements}' \tag{4.15}$$

$$\text{patch2stl}('\text{ejemplovigavoladizo.stl}', \text{fv}) \tag{4.16}$$

Donde $fv.faces$ son las triangulaciones de las capas (inferior y superior) y la triangulación en el eje Z, mientras que $fv.vertices$ corresponde a las coordenadas de todos los nodos. La Ecuación 4.16

crea un archivo STL ('ejemplo.stl') que se almacena en el directorio de MATLAB®. La Figura 4.13 muestra la salida del algoritmo al interpretar la malla de elementos finitos con un grosor específico.

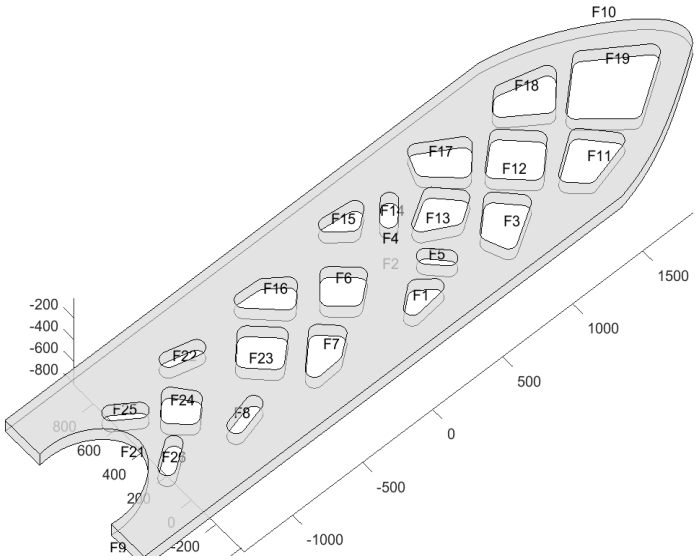

Figura 4.13 Ejemplo de reconstrucción automatizada por herramienta de redondeo de vértices para radio especificado en cavidades interiores y contorno.

La Figura 4.14 presenta el resultado del proceso de reconstrucción y exportación de una viga larga, demostrando su preparación para análisis mediante elementos finitos tetraédricos en la librería Featools de MATLAB®.

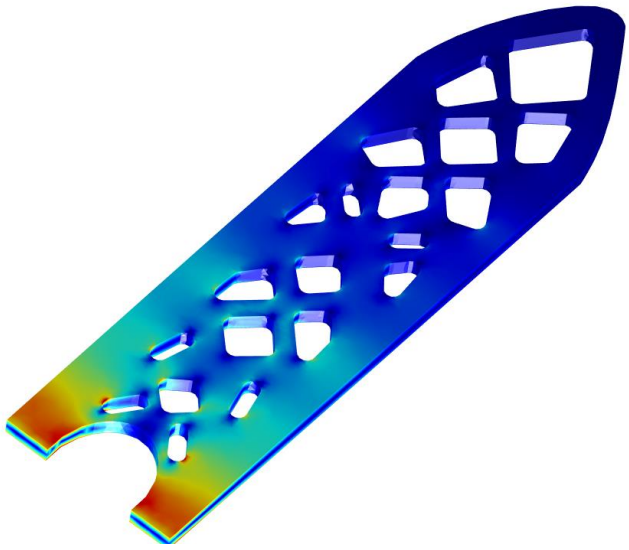

Figura 4.14 Ejemplo de viga larga reconstruida a partir de algoritmo de suavizado exportada como archivo STL para su análisis mediante elementos finitos tetraédricos.

4.5 Comando predictivo de celosía

4.5.1 Adquisición de imagen y preprocesado

Finalizada la etapa de segmentación, suavizado y reconstrucción por curvas de contorno, se obtuvo una imagen cómo la mostrada en Figura 4.12, las que pueden ser utilizadas como variables de entrada robustas para anticipar la configuración inicial de una celosía. No obstante, el algoritmo propuesto, por defecto incorpora una fase de adquisición, redimensionamiento y preprocesamiento de las imágenes correspondientes a la topología, análogo a las rutinas de suavizado y umbralización, implementadas en la fase de preprocesado del comando suavizador.

4.5.2 Extracción de esqueleto

Después del preprocesamiento, se lleva a cabo la extracción del esqueleto de la imagen. La función *bwdistgeodesic,* definida en la Ecuación 4.17, se aplica para proporcionar información sobre la distancia de cada píxel al píxel más cercano que pertenece al objeto en la imagen.

$$[D] = bwdistgeodesic(BW, seedpoint, method) \tag{4.17}$$

Donde (BW*)* representa la imagen binaria de entrada, ($Seed_{point}$*)* los puntos de semilla desde los cuales se calculan las distancias geodésicas de los pixeles y ($method$) especifica el método utilizado para el cálculo de distancias. La Figura 4.15 presenta el resultado del mapa de distancias geodésicas aplicado para la extracción precisa del esqueleto a partir de la imagen binaria.

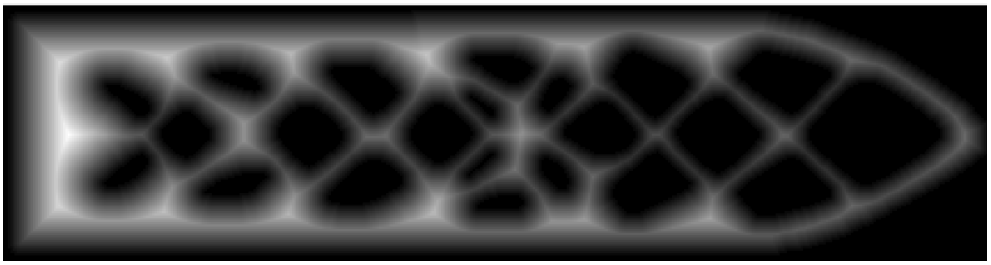

Figura 4.15 Ejemplo de mapa de distancias euclidianas aplicado a la imagen derivada de la optimización topológica.

Al aplicar el filtro-operador Laplaciano Gaussiano se realzan los bordes de las zonas de mayor intensidad o discontinuidad, mejorando las características de la imagen previa aplicación de la función *bwdistgeodesic* y revelando la representación esquelética. En la Figura 4.16 presenta el acentuado de características mediante el operador Laplaciano Gaussiano.

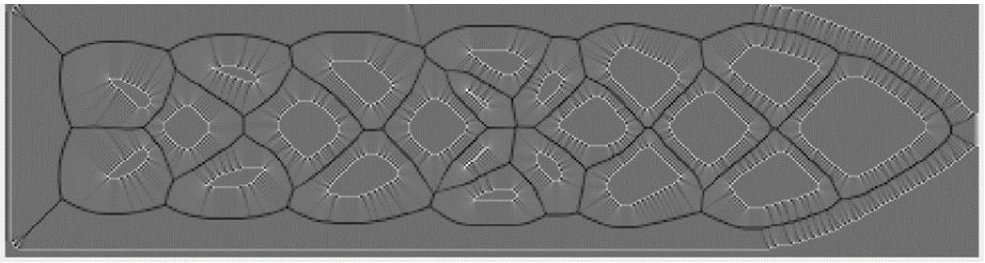

Figura 4.16 Acentuado de características mediante operador Laplaciano Gaussiano.

Posteriormente, se aplica una operación de esqueletización, eliminando detalles innecesarios e intentando mantener la topología general de la estructura, conservando las relaciones de conectividad y la forma general de la estructura. Sin embargo, en casos irregulares, la naturaleza discreta de los elementos finitos puede causar ramificaciones no deseadas, como espolones. La Figura 4.17 presenta un ejemplo de esqueletización en una imagen derivada de la optimización topológica.

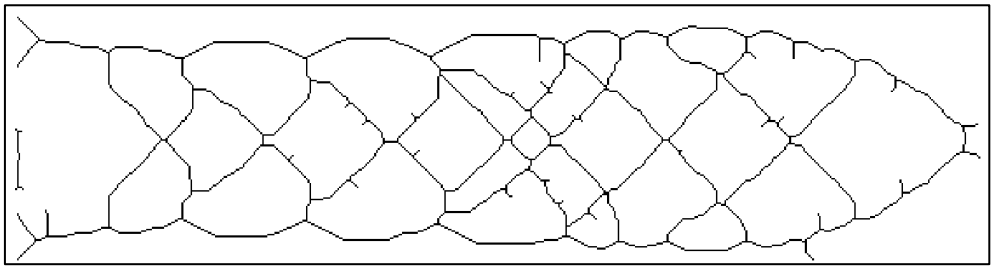

Figura 4.17 Ejemplo de esqueletización en imagen derivada de la optimización topológica con amplia variedad de ramificaciones no deseadas.

4.5.3 Algoritmo de adelgazamiento y poda de esqueleto

Una forma de simplificar y corregir dichas ramas redundantes consiste en la aplicación de la técnica conocida como skeleton prunning, la que se consigue por medios de diferentes técnicas como lo son umbrales de distancia, aproximaciones de forma, suavizado de límites, etc. No obstante, el resultado deseado no es universal y suele depender de su aplicación, quedando a criterio del examinador el

conjunto de técnicas que permiten preservar la topología original. En ese sentido para la erosión y eliminación de aquellas ramificaciones, se aplica la función señalada en Ecuación 4.18.

$$[prunedSkeleton, branchPoints, endPoints] = spur\_skeleton(D) \quad (4.18)$$

La función mencionada localiza y elimina gradualmente, píxel a píxel, la rama que contiene un punto endpoint. Este proceso se repite iterativamente hasta encontrar en su trayectoria un punto branchpoint. Posteriormente, se aplica una técnica de afinamiento esquelético mediante la operación *bwmorph* con el parámetro $(thin)$, con el objetivo de reducir aún más la estructura, y eliminar ramas finas e innecesarias. La Figura 4.18 muestra el resultado de la poda esquelética, resaltando cómo se simplifica la estructura y se eliminan las ramificaciones no deseadas.

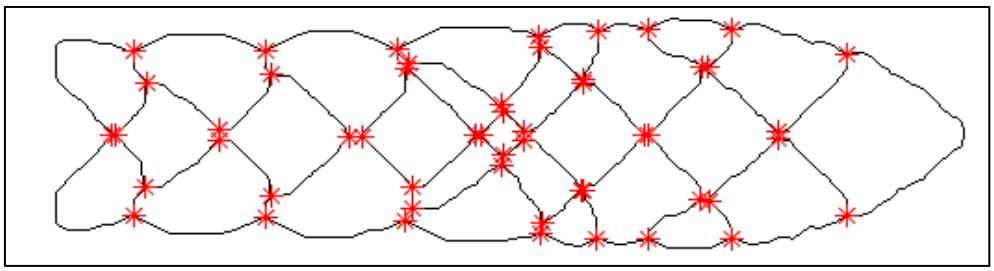

Figura 4.18 Ejemplo de poda esquelética en imagen derivada de la optimización topológica, en rojo se aprecian los Branchpoints.

4.5.4 Segmentación de ramificaciones

Se realiza una operación para eliminar los píxeles de bifurcación mediante la erosión con un elemento estructurante de disco, con un radio de tres píxeles para cada punto de bifurcación. Esta acción separa y aísla las partes de la estructura que corresponden a los puntos de bifurcación. Las funciones utilizadas se describen en las Ecuaciones 4.19 y 4.20.

$$[se] = strel('disk', "r = 3") \quad (4.19)$$

$$[\text{skel\_erode }] = strel(prunedSkeleton, se) \quad (4.20)$$

El siguiente paso es reconocer las diferentes ramas del esqueleto perforado dado por la variable de salida ($skel_{erode}$) mediante la función descrita en Ecuación 4.21 presente en la librería de procesamiento de imágenes de MATLAB®.

$$[\mathrm{CC}] = bwconncomp(skel\_erode) \tag{4.21}$$

La función busca y cuenta los componentes conectados (CC) de la imagen binaria ($skel_{erode}$), almacenando el número total de componentes conectados, regiones de interés, índices de la imagen y píxeles asignados a cada componente. La Figura 4.19 presenta un ejemplo de segmentación esquelética en una imagen derivada de la optimización topológica, destacando los componentes conectados identificados.

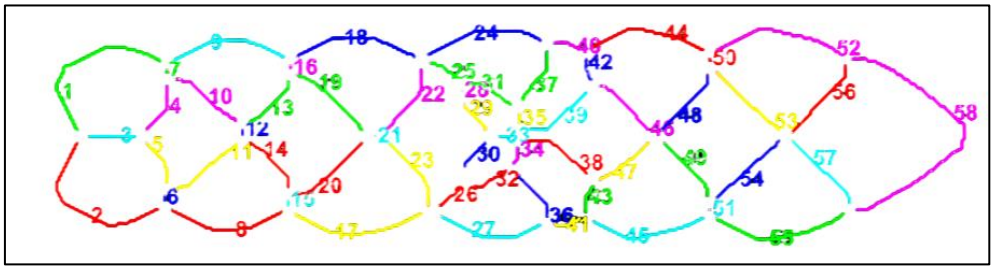

Figura 4.19 Ejemplo de segmentación esquelética en imagen derivada de la optimización topológica.

Obtenida la estructura de salida (CC) con las distintas ramas identificadas, se procede a la eliminación de los elementos que no cumplen con la longitud mínima especificada por usuario mediante la función de filtro umbral presentada en la Ecuación 4.22.

$$[skel\_erode] = \text{Basic\_filt\_umbral\_long}(\mathrm{CC}, \text{"filtro} = 0.5\%\text{"}) \tag{4.22}$$

4.5.5 Ordenación, estructuración y suavizado de ramas

Tras obtener la segmentación esquelética, se emplea la función de la Ecuación 4.8 para ordenar los puntos en sentido horario. Luego, se aplica la función de suavizado de ramas descrita en la Ecuación 4.9, permitiendo analizar curvaturas e incorporar nuevos cortes dentro de las ramificaciones con el fin de obtener una representación precisa de la estructura.

4.5.6 Obtención de puntos característicos

La obtención de puntos característicos se logra evaluando la curvatura de las ramificaciones suavizadas. Utilizando la función de la Ecuación 4.10, que analiza las derivadas de las coordenadas de los puntos a lo largo de las ramificaciones, se identifican áreas con alta curvatura, indicando posibles bifurcaciones o cambios abruptos en la dirección de la estructura, como se observa en la Figura 4.20.

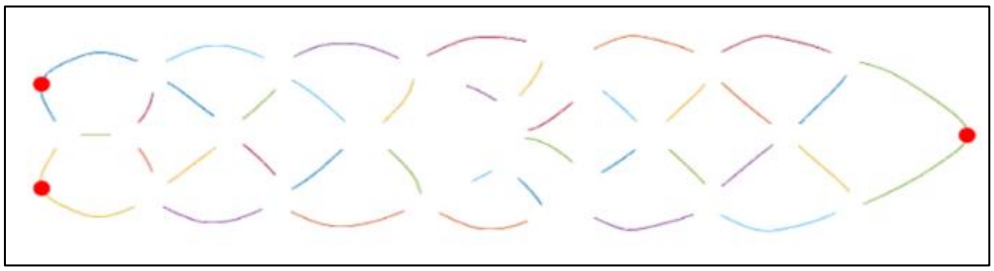

Figura 4.20 Reconocimiento de puntos de curvatura pronunciada en esqueleto suavizado

El siguiente paso es eliminar y erosionar dichas zonas con puntos característicos e incorporar las nuevas ramas mediante la herramienta de reconocimiento de ramas aisladas presentada en la Ecuación 4.20, de esta forma se logra una representación más detallada y precisa de la que será la estructura de celosía interpretada. La Figura 4.21 muestra esta representación mejorada de la estructura de celosía.

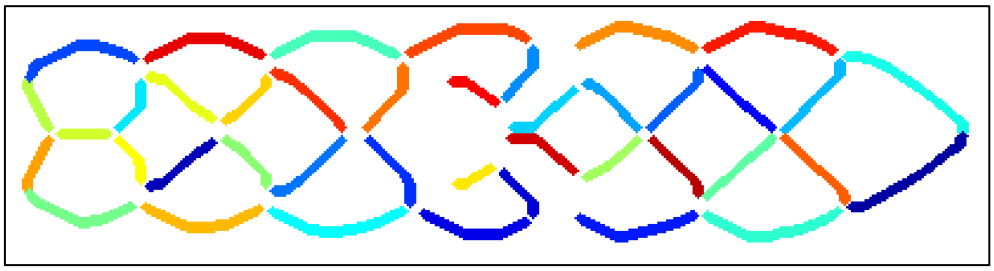

Figura 4.21 Ejemplo de segmentación esquelética robusta mediante filtrado, suavizado y reconocimiento de patrones de forma y topología.

4.5.7 Segmentación topológica

Este proceso de segmentación utiliza la segmentación esquelética de la imagen binaria original y la imagen umbralizada de la fase de preprocesamiento como entradas. El algoritmo identifica y etiqueta

las ramas aisladas del esqueleto, considerándolas como "mínimos locales" para el proceso de segmentación, generando así regiones de interés. Luego, aplica el algoritmo Watershed, creando capas donde cada región representa un área conectada de la imagen. En la Figura 4.22 se muestra un ejemplo de segmentación en regiones basada en partición de esqueleto e implementación de Watershed.

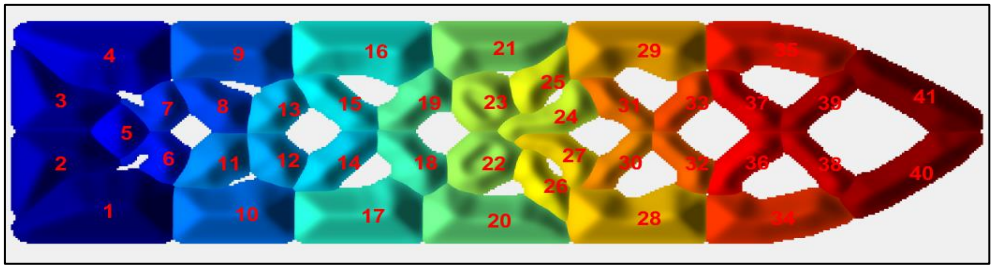

Figura 4.22 Ejemplo de segmentación topológica basado en segmentación por esqueletos y algoritmo watershed.

4.5.8 Grafo relacional

En esta fase, se obtienen las coordenadas ordenadas de la imagen segmentada por ramificaciones. Luego, se emplea un algoritmo de búsqueda basado en distancias para identificar nodos representativos al inicio y final de cada ramificación, estableciendo puntos comunes y comprendiendo la conexión de la estructura. Se calcula una matriz de incidencia para representar las relaciones entre nodos y regiones, indicando si un nodo está conectado a una región específica. Posteriormente, se re etiquetan las regiones de la segmentación topológica utilizando esta matriz para vincular los cambios en las ramificaciones esqueléticas con la segmentación de la forma. En la Figura 4.23 se ilustra este proceso de retiquetado y vinculación en la segmentación topológica.

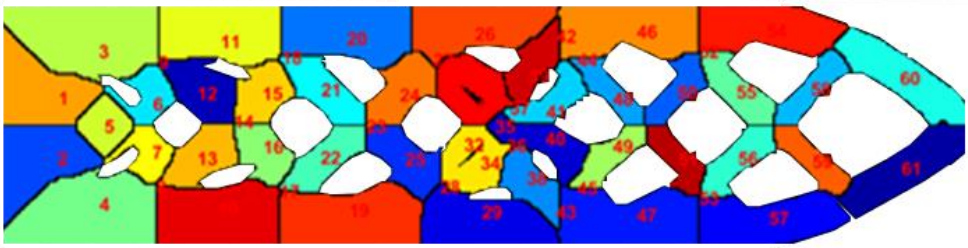

Figura 4.23 Re etiquetado de regiones para segmentación de forma

Se procede a la creación de un grafo de relaciones, previa transformación de la matriz de incidencia a una matriz de adyacencia. Las coordenadas de los nodos se establecen en función de las ubicaciones

de los puntos característicos o nodos en la imagen, y se asignan una serie de propiedades modificables tanto a nodos (v)., como aristas (e). La Figura 4.24 presenta un ejemplo de creación de grafo relacional.

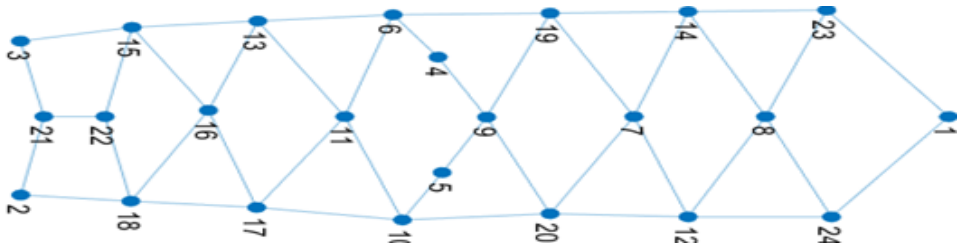

Figura 4.24 Ejemplo de grafo relacional entre segmentación de forma y segmentación por esqueleto

4.5.9 Ordenación y estructuración Bresenham

Se implementa un algoritmo para perfilar el ancho de banda, utilizando técnicas de procesamiento de imágenes, con el fin de analizar segmentos esqueletonizados de una imagen binarizada. Se emplea una regresión cúbica y un método de perfilado con un ancho de banda adaptativo, mediante el uso del algoritmo de Bresenham. Esto permite captar de manera ordenada puntos de interés, ajustados mediante curvas de regresión RANSAC de grado no mayor a tres, como se ilustra en la Figura 4.25.

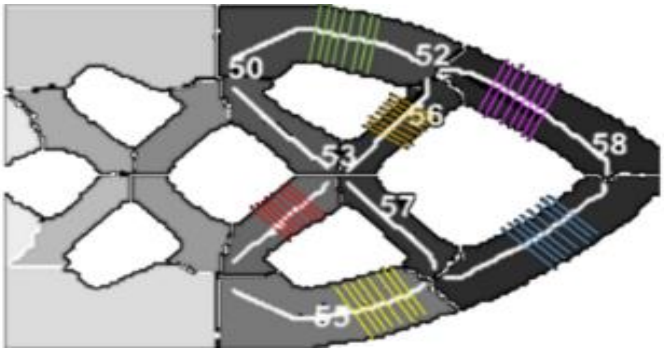

Figura 4.25 Ejemplo de perfilado de ancho de banda para regularización geométrica.

4.5.10 Reconstrucción basada en esqueletos

La etapa final implica la reconstrucción de la celosía utilizando las curvas de regresión RANSAC y la conectividad previamente establecida en el grafo relacional. Se calculan las intersecciones entre las ramas reconstruidas y se generan elementos de curvas de regresión con un grado máximo de tres. En la Figura 4.26 se presenta la reconstrucción final de la celosía a partir del algoritmo propuesto.

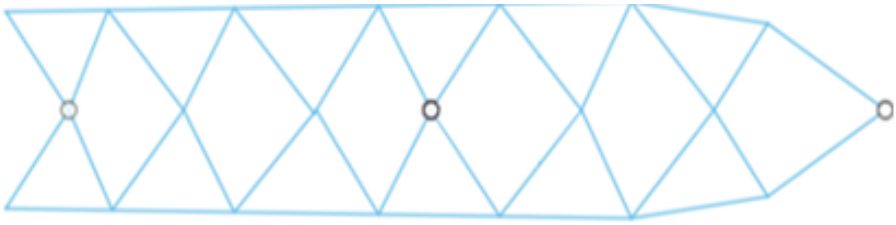

Figura 4.26 Reconstrucción final de la celosía mediante curvas de regresión y grafo relacional.

4.6 Limitaciones del algoritmo

El algoritmo, orientado a optimizar estructuras con materiales isótropos, presenta limitaciones en su interfaz gráfica rectangular y requiere adicionar un módulo CAD para importar geometrías complejas. La efectividad del suavizado y la reconstrucción depende de ajustes precisos de parámetros, siendo sensible a la geometría y calidad de la imagen. La función de redondeo automatizado tiene limitaciones en situaciones complejas. La segmentación Watershed puede generar resultados no deseados, y la eliminación de ramas cortas afecta la representación detallada. La complejidad computacional es alta, afectando el rendimiento según el tamaño y complejidad de la imagen.

4.7 Diagrama de Flujo

El diagrama de flujo de la rutina implementada se ilustra en la Figura 4.27. Este corresponde a una síntesis de los principales procesos.

4.8 Conclusiones

En este capítulo, se abordó la creación de una rutina en MATLAB® centrada en la reconstrucción morfológica mediante la optimización topológica y técnicas de suavizado de bordes y esqueletización. La metodología integró el procesamiento de imágenes y el análisis geométrico, lo que permitió transformar estructuras optimizadas en modelos detallados para aplicaciones en ingeniería. La optimización Soft Kill BESO generó una imagen ráster que sirvió como base para identificar las estructuras clave en las que aplicar segmentación de cuencas y suavizado de curvas. Finalmente, las estructuras resultantes se transformaron en grafos relacionales, utilizando la reconstrucción mediante curvas de regresión RANSAC para obtener una representación detallada y precisa de la estructura.

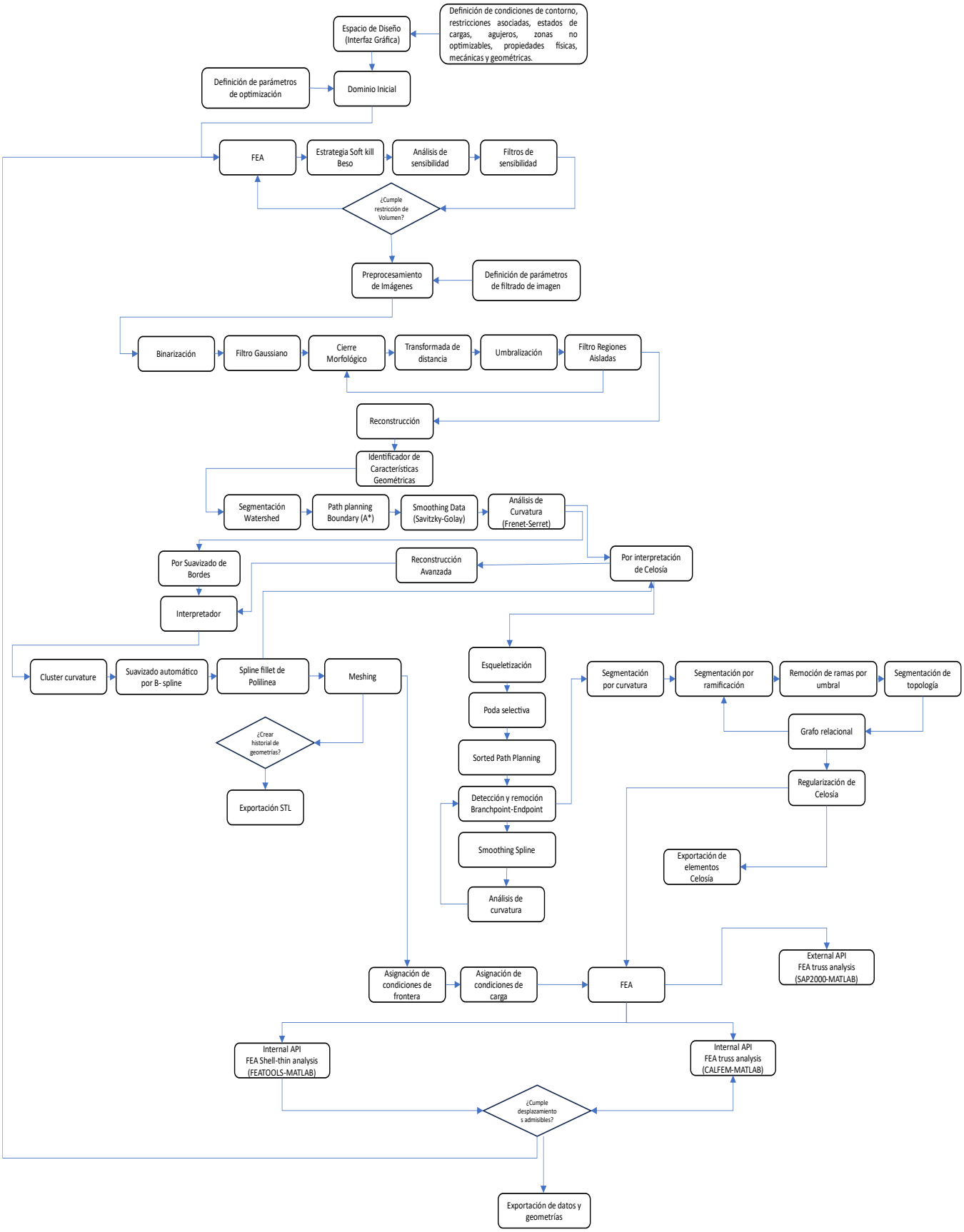

Figura 4.27 Diagrama de flujo de rutina implementada

CAPÍTULO 5: APLICACIONES NUMÉRICAS

5.1 Introducción

En este capítulo, se aborda la validación de la rutina implementada mediante la aplicación de ejemplos que abarcan diferentes configuraciones estructurales. Se busca demostrar la robustez del algoritmo propuesto en la interpretación y reconstrucción de características geométricas en el contexto de la optimización topológica. En particular, se destaca un ejemplo numérico basado en el trabajo previo de Oviedo (2020), el cual abarca una viga simplemente apoyada con dimensiones de 120x40 milímetros y carga puntual de 200 N. Se realiza un análisis comparativo del desempeño entre los resultados obtenidos mediante el algoritmo propuesto y aquellos derivados de Oviedo. Finalmente, se procede a comparar el algoritmo de predicción de celosía propuesto con los resultados del estudio previo de Gedig (2010), ampliando el enfoque hacia una viga larga en voladizo de dimensiones 8000x2000 milímetros y con carga puntual de 100 kN en el extremo.

5.2 Ejemplos de aplicación

Con el fin de demostrar la validez y la capacidad del algoritmo propuesto para la identificación, interpretación y reconstrucción de características geométricas, se prueban un ejemplo numérico basado en el trabajo de Oviedo (2020) y un ejemplo correspondiente al trabajo de Gedig (2010).

5.3 Viga simplemente apoyada

Se realizo un análisis comparativo del desempeño de una viga simplemente apoyada de dimensiones 120x40 milímetros con carga puntual de 200 N en el centro, reconstruida mediante el algoritmo propuesto, en contraste con los resultados obtenidos del trabajo previo de Oviedo (2020). Las propiedades del hormigón y parámetros de la optimización se detallan en la Tabla 5.1.

Tabla 5.1 Parámetros de estructura óptima de viga simplemente apoyada 120x40 milímetros que cumple restricción de desplazamientos

L [mm]	H [mm]	Malla	E [MPa]	v	P [N]	Tamaño malla triangular [mm]	Espesor [mm]	Δ admisible [mm]
120	40	120x40	23500	0,25	200	1,5	1	L/480

En relación con la configuración estructural basada en suavizado y redondeo de bordes se estableció un parámetro de filete de 3 mm. En contraste, para la configuración basada en la predicción e interpretación de la celosía, se ajustaron secciones rectangulares de 3 mm por 1 mm para cumplir con las deformaciones permitidas en la Tabla 5.1. Las topologías interpretadas resultantes y su comparación con la reconstrucción realizada por Oviedo (2020) son presentadas en la Figura 5.1.

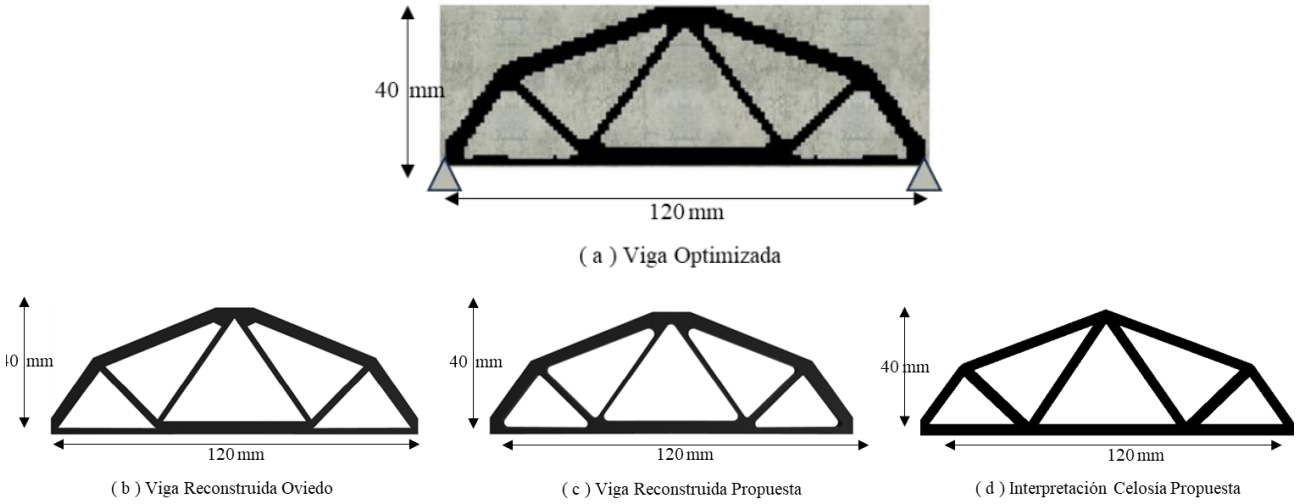

Figura 5.1 Optimización para viga simplemente apoyada de hormigón con carga puntual de 200 N en el centro: a) Configuración obtenida mediante algoritmo OT, b) Configuración obtenida a partir de algoritmo de Oviedo, c) Configuración obtenida a partir de algoritmo propuesto basado en suavizado y redondeo de bordes d) Configuración obtenida a partir de algoritmo propuesto basado en predicción de celosía estructural.

Las estructuras resultantes de las optimizaciones de forma basadas en OT son exportadas, simuladas y comparadas mediante diversos softwares y librerías, los resultados son presentados a continuación.

5.3.1 Geometría de Oviedo

Los parámetros utilizados para obtener la estructura óptima se muestran en la Tabla 5.2. La fracción de volumen resultante se modificó desde el 30 % de la FV hasta el 24% de la misma.

Tabla 5.2 Resultados de estructura equivalente de Oviedo óptima para viga simplemente apoyada 120x40 milímetros que cumple restricción de desplazamiento. (MATLAB®)

FV inicial [%]	FV final [%]	Diferencia [%]	Δ máximo final [mm]	Δ admisible [mm]	Límite a Δ admisible [%]
30	24,0	6,0	0,232	0,250	L/480

La Tabla 5.3 presenta los esfuerzos principales máximos obtenidos del estado mostrado en Figura 5.2.

Tabla 5.3 Esfuerzos principales máximos y mínimos de estructura óptima para viga equivalente de Oviedo simplemente apoyada con carga puntual en el centro (MATLAB®).

Esfuerzos [MPa]	σ_1 [MPa]	σ_2 [MPa]	σ_{VM} [MPa]
Máximo	60,92	2,96	137,68
Mínimo	-7,97	-141,70	1,26

Respecto a la distribución de tensiones, se analizaron los esfuerzos principales y desplazamientos máximos mediante análisis estructural basado en elementos finitos Q4 a partir de la librería CALFEM® de MATLAB®. En la Figura 5.2 se muestra el estado tensional resultante de Oviedo.

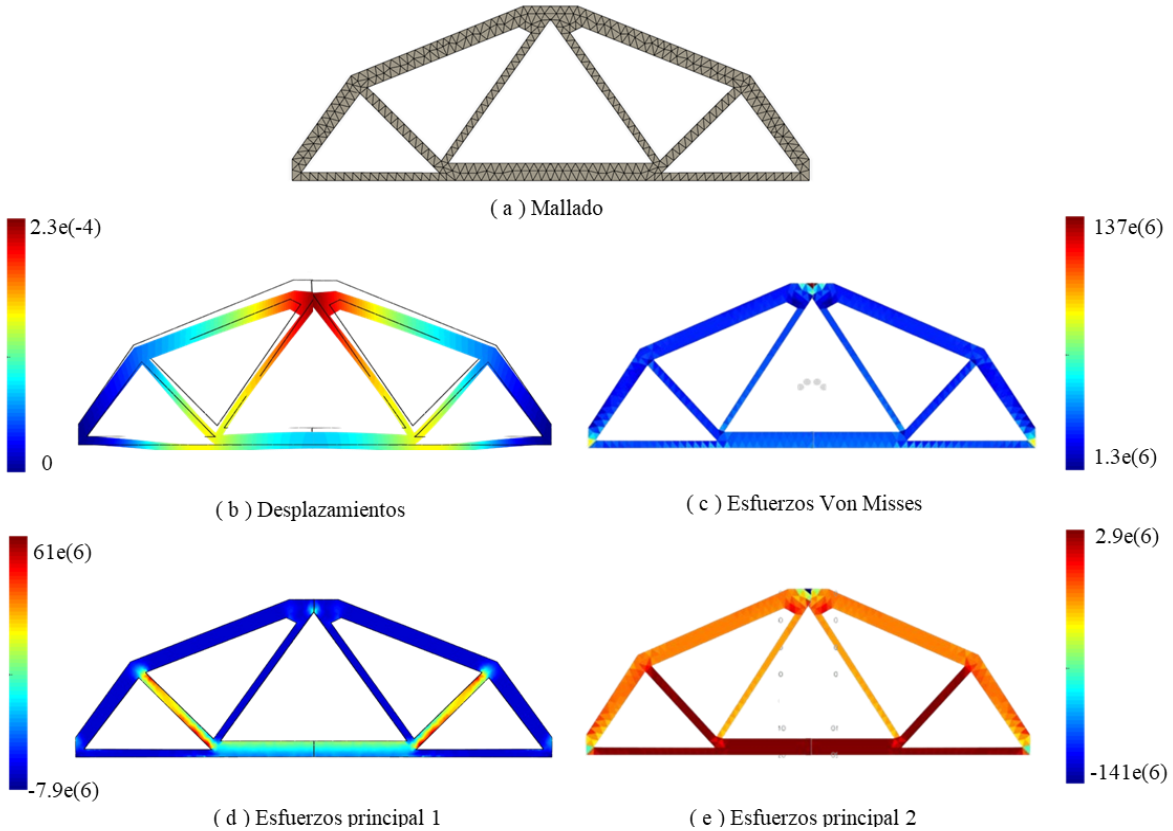

Figura 5.2 Estado tensional de estructura de Oviedo óptima para viga de hormigón simplemente apoyada equivalente de 60x40 milímetros con carga puntual en el centro simulada a partir de la librería CALFEM® de MATLAB®: a) Mallado, b) Desplazamientos, $c)$ σ_{VM}, $d)$ σ_1, $e)$ σ_2.

Para la estructura obtenida por Oviedo, se observa una fracción de volumen de 24,0 % después del suavizado, y una deformación máxima en el centro de 0,232 mm, inferior a la deformación admisible. Además, se detecta una diferencia del 6 % en la fracción volumétrica obtenida en el modelo alisado en comparación con la topología optimizada con bordes irregulares.

5.3.2 Geometría propuesta: Algoritmo suavizado de forma

Los parámetros utilizados para obtener la estructura óptima se presentan en la Tabla 5.4. La fracción volumétrica (FV) después de la reconstrucción se modifica desde un 30 % de FV hasta un 23,7 %.

Tabla 5.4 Resultados de la estructura equivalente optima reconstruida mediante algoritmo propuesto para viga simplemente apoyada de dimensiones 120x40 milímetros con restricción de desplazamiento. (MATLAB®)

FV inicial [%]	FV final [%]	Diferencia [%]	Δ máximo final [mm]	Δ admisible [mm]	Límite a Δ admisible [%]
30	23.70	6,3	0,221	0,250	L/480

En relación con la distribución de tensiones para el modelo suavizado propuesto, se llevó a cabo un análisis de esfuerzos principales máximos y los desplazamientos correspondientes mediante un enfoque de elementos finitos tetraédricos utilizando la librería de simulación FEATOOLS® presente en MATLAB®, la cual se presenta de manera detallada en la Figura 5.3.

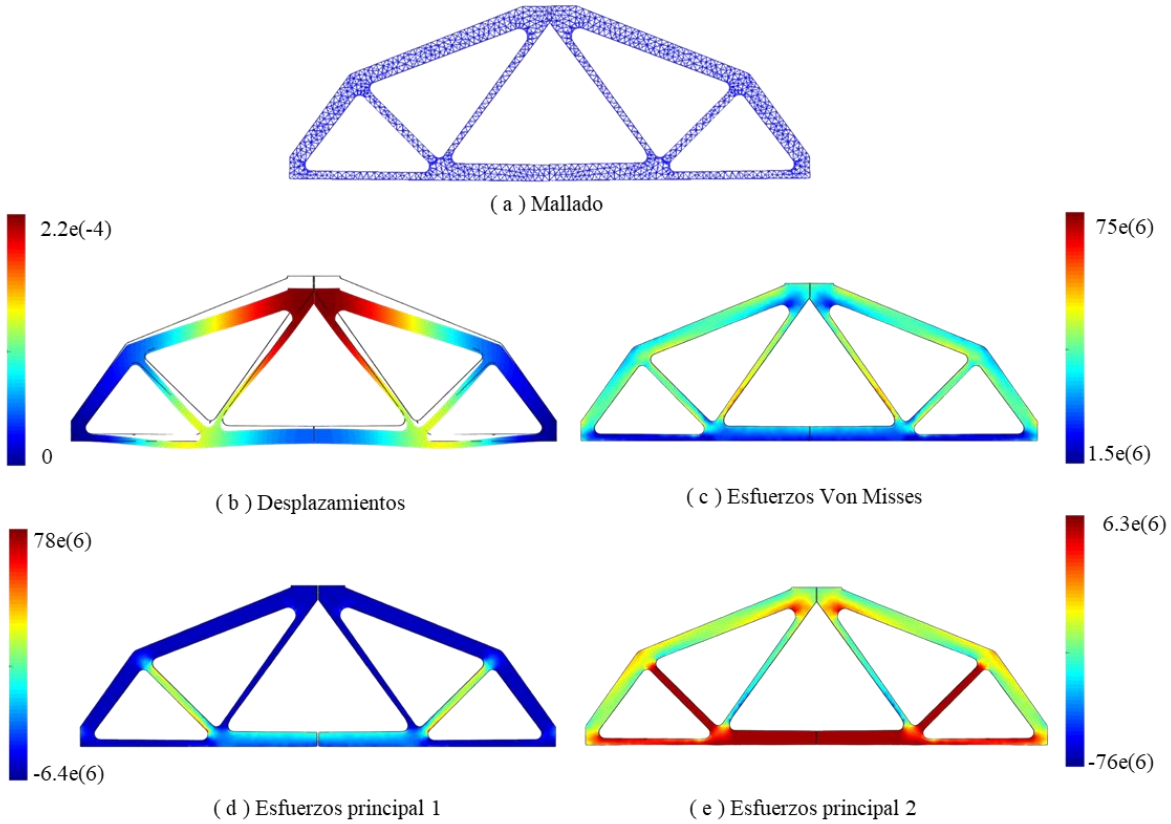

Figura 5.3 Estado tensional de estructura óptima mediante el algoritmo de suavizado propuesto para viga simplemente apoyada sometida a una carga puntual en el centro. La simulación se realiza utilizando la librería FEATOOLS® de MATLAB®: a) Representación de la discretización de la estructura, b) Evaluación de las deformaciones en diferentes puntos de la viga, c) σ_{VM}, d) σ_1, e) σ_2

La Tabla 5.5 presenta los esfuerzos principales máximos obtenidos para cada uno de los estados tensionales mostrados en la Figura 5.4.

Tabla 5.5 Esfuerzos principales máximos y mínimos de estructura óptima para viga equivalente simplemente apoyada con carga puntual en el centro reconstruida mediante algoritmo de suavizado propuesto (MATLAB®)

Esfuerzos [MPa]	σ_1 [MPa]	σ_2 [MPa]	σ_{VM} [MPa]
Máximo	78,102	6,352	74,861
Mínimo	-6,433	-75,675	1,518

Para la geometría reconstruida basada en el algoritmo de suavizado de bordes propuesto, se aprecia una fracción de volumen de 23,7% y una deformación máxima en el centro de 0,221 mm inferior a la deformación admisible y la desarrollada por Oviedo (2020). Respecto al estado tensional presentado en la Figura 5.3, se visualizan concentraciones de esfuerzos menores a las anteriores, pero significativamente por encima de los valores comunes de resistencia del hormigón. Los resultados obtenidos son comparados mediante el software de modelación FUSION360® de Autodesk® para la misma geometría, tal como se ilustra en la Figura 5.4.

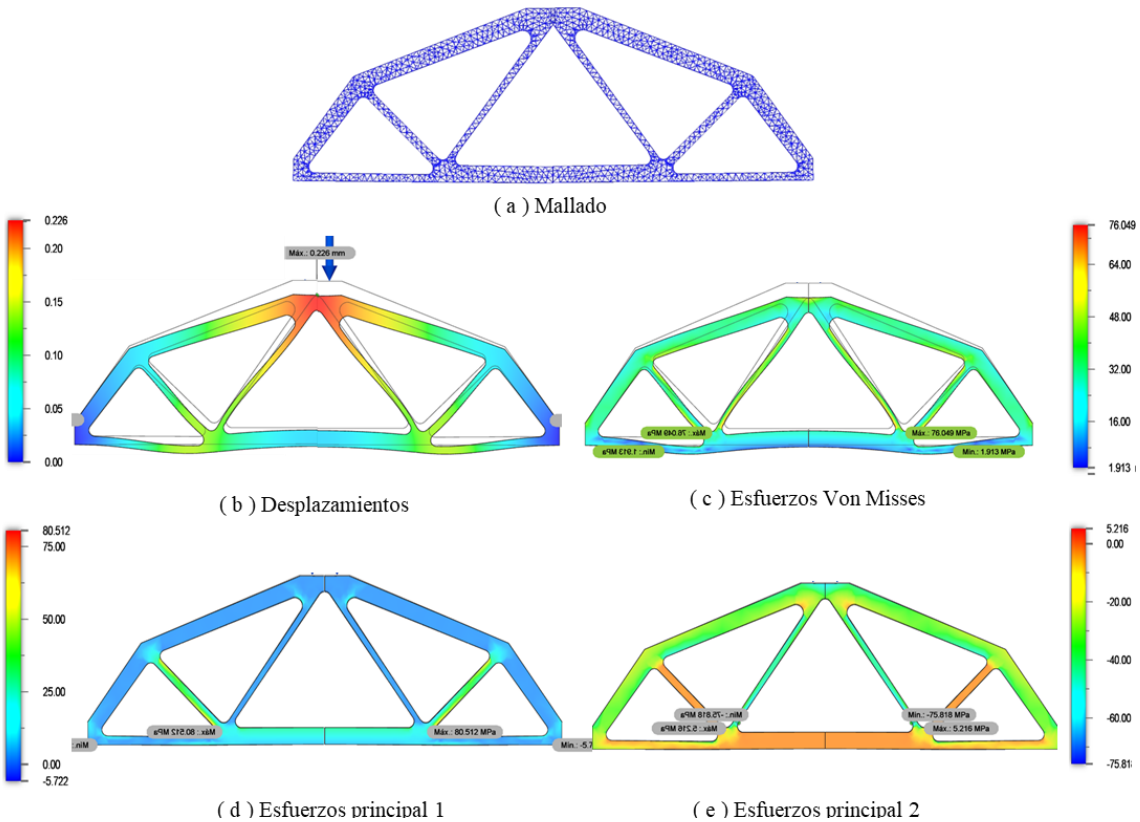

Figura 5.4 Estado tensional de estructura óptima mediante el algoritmo de suavizado propuesto para viga simplemente apoyada de hormigón con dimensiones de 120x40 milímetros, sometida a una carga puntual en el

centro. La simulación se realiza utilizando FUSION360® a) Representación gráfica de la discretización de la estructura, b) Evaluación de las deformaciones y movimientos en diferentes puntos de la viga, c) σ_{VM}, d) σ_1, e) σ_2

Los parámetros utilizados se muestran en la Tabla 5.6, mientras la geometría exportada en Figura 5.3.

Tabla 5.6 Resultados de la estructura equivalente optima suavizada mediante algoritmo propuesto para viga simplemente apoyada de dimensiones 120x40 milímetros que satisface la restricción de desplazamiento. (FUSION360®)

	FV inicial [%]	FV final [%]	Diferencia [%]	Δ máximo final [mm]	Δ admisible [mm]	Límite a Δ admisible [%]
Fusion360	30	23.70	6,3	0,226	0,25	L/480

La Tabla 5.7 presenta los esfuerzos principales máximos obtenidos para cada uno de los estados tensionales mostrados en la Figura 5.4.

Tabla 5.7 Esfuerzos principales máximos y mínimos de estructura óptima para viga equivalente simplemente apoyada con carga puntual en el centro reconstruida mediante algoritmo de suavizado propuesto (FUSION360®)

Esfuerzos [MPa]	σ_1 [MPa]	σ_2 [MPa]	σ_{VM} [MPa]
Máximo	80,51	5,21	76,05
Mínimo	-5,72	-75,81	1,91

En la Tabla 5.6 se comparan los resultados iniciales y finales de fracción de volumen (FV) obtenidos mediante FUSION360®. Se destaca una reducción del 6,3% en la FV, alcanzando un valor final del 23,7%. La deformación máxima final es de 0.226 mm, ligeramente inferior al límite admisible de 0.25 mm. En la Tabla 5.7 Se presentan los esfuerzos máximos y mínimos obtenidos en la estructura óptima reconstruida mediante el algoritmo de suavizado propuesto en FUSION360®. Se visualizan diferencias menores a los $\pm$ 2 MPa respecto de la misma geometría analizada en por medio de la librería FEATOOLS® de MATLAB® presentada en la Figura 5.3.

5.3.3 Geometría propuesta: Algoritmo de interpretación por celosía

Referente a la reconstrucción basada en interpretación de Celosía, se analizan las tensiones por tramos correspondientes a su configuración en base a elementos tetraédricos y lo obtenido en base a elementos de barra. En la Figura 5.5 se presenta la distribución de tensiones de Von Mises obtenida en cada caso.

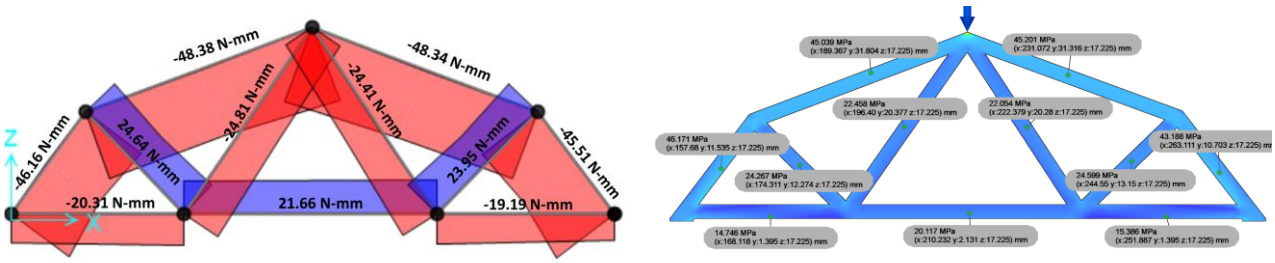

Figura 5.5 Distribución de tensiones de Von Misses en MPa. (a) Modelo de elementos finitos de barra en SAP2000® a partir de celosía interpretada por algoritmo propuesto. (b) Modelo de elementos finitos tetraédricos Fusion360® a partir de celosía interpretada por algoritmo propuesto.

Con el propósito de evaluar la integridad estructural de la celosía mediante el algoritmo propuesto, se llevó a cabo un análisis de tensiones de Von Mises por tramos los cuales se presentan en la Tabla 5.8.

Tabla 5.8 Resumen comparativo de tensiones de Von misses por tramos a partir de celosía interpretada (Algoritmo Propuesto), exportadas y simuladas mediante elementos finitos tetraédricos (FUSION360®) y elementos finitos de barra (SAP2000®).

SVM	I	II	III	IV	V	VI
	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]
SAP2000	48,09	51,45	52,46	48,35	20,03	21,87
Fusion360	46,17	45,04	45,20	43,19	15,39	20,12
Diferencia	4%	12%	14%	11%	23%	8%

SVM	VII	VIII	IX	X	XI
	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]
SAP2000	20,93	26,74	27,43	26,57	26,51
Fusion360	14,75	24,27	22,46	22,05	24,57
Diferencia	30%	9%	18%	17%	7%

La tabla se complementa con la Figura 5.6 y presenta las deformadas obtenidas a partir de la celosía interpretada bajo restricciones de desplazamiento en puntos característicos de la misma.

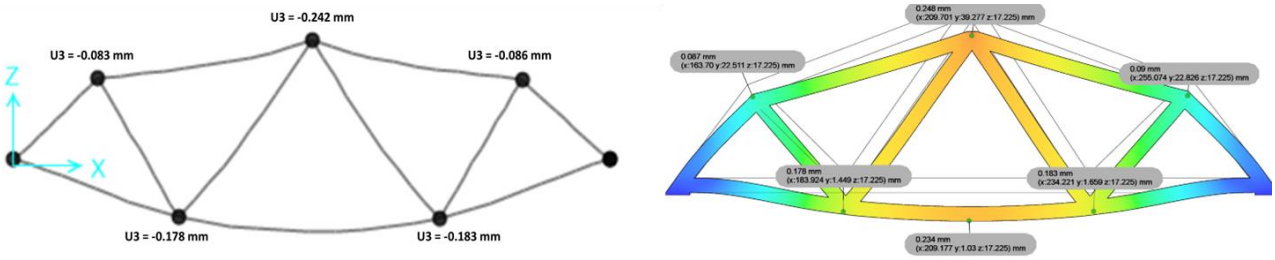

Figura 5.6 Deformada a partir de celosía interpretada bajo restricción específica de desplazamiento establecida en mm. (a) Modelo de elementos finitos de barra en SAP2000® a partir de celosía interpretada por algoritmo propuesto. (b) Modelo de elementos finitos tetraédricos Fusion360® a partir de celosía interpretada propuesta.

La Tabla 5.9 presenta una comparación detallada de las deflexiones nodales en puntos característicos derivados de la interpretación de una celosía bajo restricción específica de desplazamiento. Estas deflexiones fueron calculadas para dos tipos de modelado: utilizando elementos de barra en SAP2000® y elementos tetraédricos en FUSION360®.

Tabla 5.9 Resumen comparativo de las deflexiones nodales en mm a partir de celosía interpretada bajo restricción de desplazamiento establecida para elementos barra (SAP2000®) y elementos tetraédricos FUSION 360®.

Deflexión	P_1	P_2	P_3	P_4	P_5	P_6	P_7
	δ_d [mm]	δ_d [mm]	δ_d [mm]	δ_d [mm]	δ_d [mm]	δ_d [mm]	δ_d [mm]
SAP2000	0	0,183	0,178	0	0,086	0,083	0,242
Fusion360	0	0,183	0,178	0	0,09	0,087	0,248
Diferencia	0%	0%	0%	0%	5%	5%	2%

La comparación entre los modelos de elementos finitos de barra y tetraédricos presentadas en la Tabla 5.8, muestra variaciones en las tensiones de Von Mises de hasta 30%, destacando la influencia del elemento finito tipo barra que para todos los miembros sobrestima los esfuerzos desarrollados en la geometría obtenida. Las deformadas y deflexiones nodales presentadas en la Figura 5.6 muestran una consistencia razonable entre SAP2000® y FUSION360®. Estos valores se detallan en la Tabla 5.9 bajo las restricciones de carga y desplazamiento aplicadas. En la Figura 5.7 y Tabla 5.10 se presenta un análisis detallado de las fuerzas internas por miembro para la configuración estructural basada en celosía interpretada.

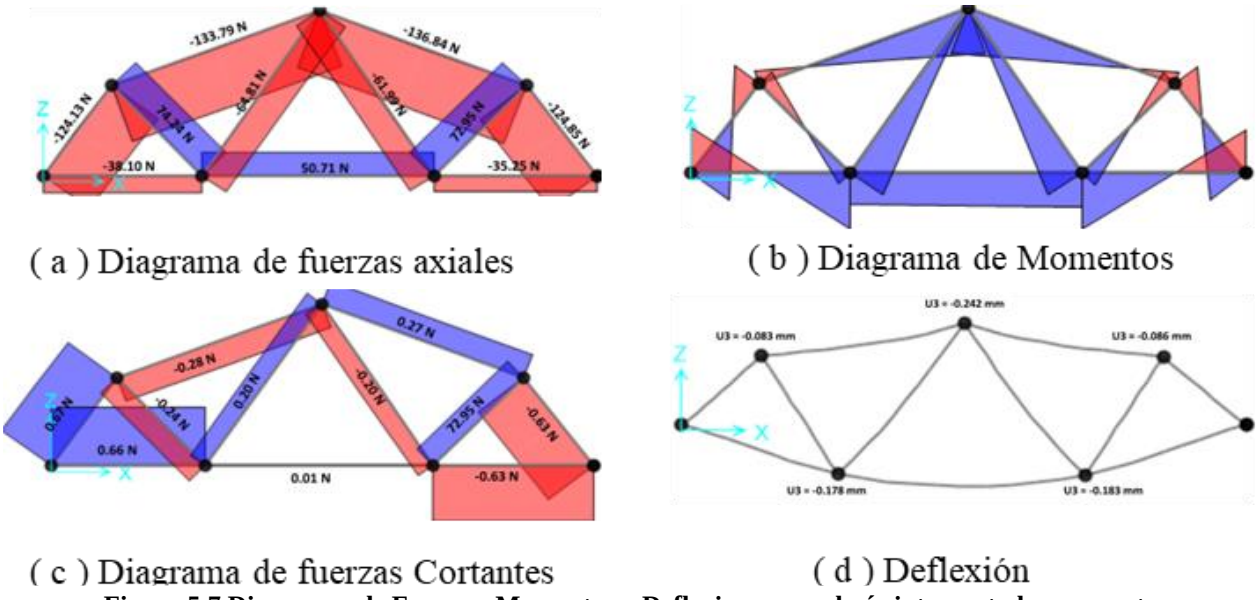

Figura 5.7 Diagramas de Fuerzas, Momentos y Deflexiones en celosía interpretada propuesta.

Tabla 5.10 Resumen de Extremos Globales para Diagrama de Fuerzas Internas por Miembro en Celosía Interpretada con Elementos Finitos de Barra en SAP2000® .

Miembro	**Axial [N]**	**Corte [N]**	**Momento [N-mm]**	**Deflexión [mm]**
I	-123,81	0,68	10,22	0,099
II	-133,18	-0,28	10,59	0,143
III	-136,18	0,28	10,60	0,140
IV	-125,54	-0,63	9,76	0,103
V	-34,93	-0,63	12,59	0,186
VI	50,25	0,01	7,68	0,005
VII	-37,74	0,66	12,53	0,180
VIII	73,01	-0,25	3,60	0,081
IX	-64,88	0,20	8,71	0,062
X	-62,11	-0,21	8,80	0,057
XI	71,77	0,26	3,89	0,083

5.4 Viga larga en voladizo

Para el tercer caso, se amplió el enfoque hacia una viga en voladizo de dimensiones 8000x2000 milímetros y con carga puntual de 100 kN en el extremo. Este escenario, a diferencia de los casos anteriores, proporciona una comparación detallada entre el algoritmo de predicción de celosía propuesto y los resultados derivados del estudio previo de Gedig (2010), con el fin de evidenciar su robustez y aplicabilidad en estructuras más extensas y complejas. La Figura 5.8 presenta la configuración del dominio de diseño para la viga en voladizo, así como la imagen binaria generada.

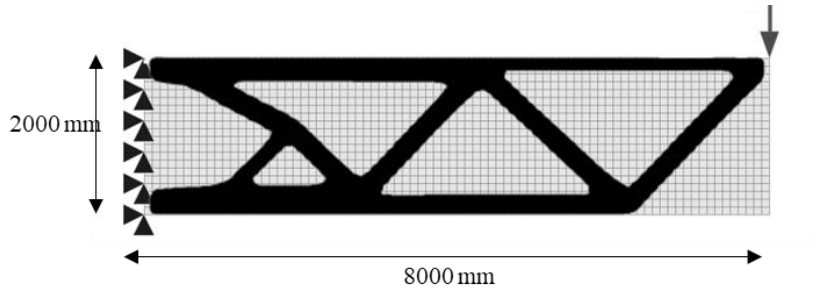

Figura 5.8 Viga larga en voladizo con carga puntual de 100 KN en el extremo: Dominio de diseño e imagen binaria generada para el modo optimizado de la topología.

La comparación de los resultados se realizó considerando diversas métricas, tales como el coeficiente de determinación R^2 para evaluar la similitud geométrica entre las representaciones estructurales y parámetros críticos relacionados con el comportamiento estructural. Estos parámetros incluyen la eficiencia estructural, los errores ponderados, así como los valores máximos y mínimos por miembro en categorías vinculadas al rendimiento de la estructura, tales como esfuerzos axiales, cortantes,

momentos, deflexiones y factores de utilización. En la Figura 5.9 y Figura 5.10 se revelan fases distintivas de cada algoritmo durante el proceso reconstructivo.

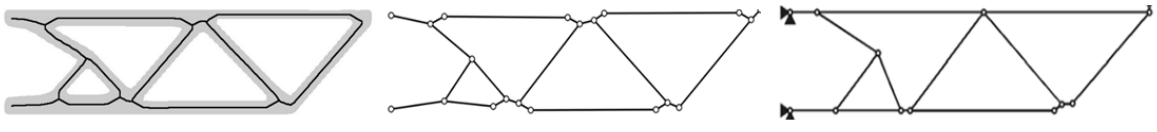

Figura 5.9 Proceso reconstructivo basado en algoritmo de reconocimiento de celosías. (Gedig, 2010).

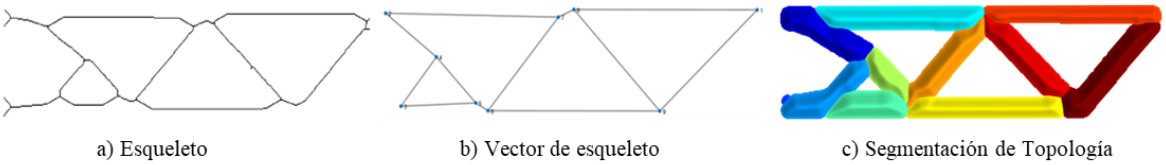

Figura 5.10 Proceso reconstructivo basado en algoritmo de reconocimiento de celosías propuesto.

Las Figura 5.11 y Figura 5.12 presentan las configuraciones estructurales optimizadas, siendo la primera predicha por el algoritmo de Gedig (2010) tras el proceso de estabilización, y la segunda predicha por nuestro algoritmo después de la regularización geométrica.

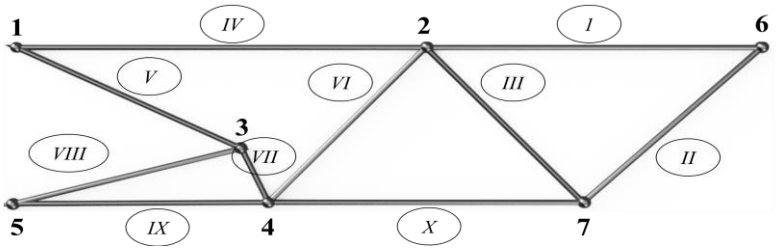

Figura 5.11 Configuración estructural optimizada predicha por algoritmo de predicción y reconstrucción Michael Gedig (2010) posterior a algoritmo de estabilización.

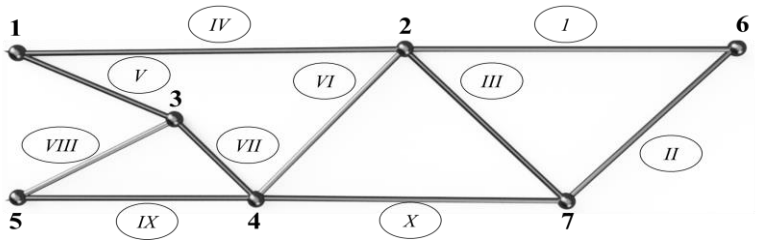

Figura 5.12 Configuración estructural optimizada predicha por el algoritmo propuesto, posterior a algoritmo de regularización geométrica.

Las Tabla 5.11 y Tabla 5.12 proporcionan los valores de coeficiente de determinación R^2 obtenidos al comparar cada miembro estructural. Estos valores reflejan la similitud geométrica en relación con la distribución y cercanía de los puntos entre el modelo propuesto y el modelo de Gedig (2010).

Tabla 5.11 Coeficiente de Determinación R^2 y Longitudes por miembros.

		I	II	III	IV	V	VI	VII	VIII	IX	X
L [mm]	**Gedig**	3560	2660	2520	4350	2680	2520	720	2510	2710	3350
	Prop.	3650	2590	2520	4260	1890	2410	1300	1950	2620	3410
R^2 [%]		99.42	99.86	99.13	99.37	68.74	98.50	-	61.48	98.79	99.55

Tabla 5.12 Resumen de Métricas de similitud y longitudes globales

Error Cuadrático Medio (ECM) global:	656.82 [u]
Coef. de Determinación R^2 global:	90.65 [%]
Diferencia longitud total entre modelos:	1092 [mm]

El Error Cuadrático Medio (ECM) global (medida de discrepancia entre los modelos), es de 656.82 unidades. El Coeficiente de Determinación R^2 global, que evalúa la similitud geométrica, alcanza el 90.65%. La longitud total para el modelo de Gedig exhibe de 27.6 m, mientras el modelo propuesto presenta una longitud de 26,6 m, indicando similitudes sustanciales entre los modelos.

5.4.1 Análisis Estructural mediante Uniones Articuladas

El análisis estructural se llevó a cabo mediante el software SAP2000®, mediante modelado estándar. Se modeló una viga larga en voladizo con uniones articuladas para representar la celosía propuesta y el modelo de Gedig (2010). Ambos modelos se sometieron a una carga puntual de 100 KN en el extremo de la viga. En la Figura 5.13 se presentan los resultados del análisis estructural para el modelo de Gedig, mostrando la respuesta de la viga en voladizo ante la carga puntual.

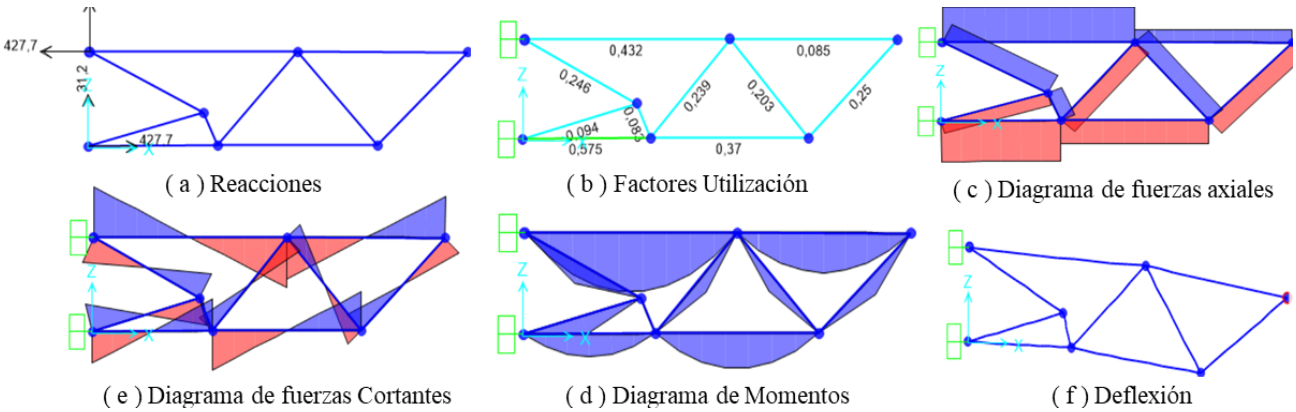

Figura 5.13 Resultados del análisis estructural para viga larga en voladizo con carga puntual de 100 KN en su extremo a partir de celosía Michael Gedig (2010) modelada mediante uniones articuladas en SAP2000®.

En la Figura 5.14 se exhiben los resultados del modelo desarrollado a partir de algoritmo propuesto.

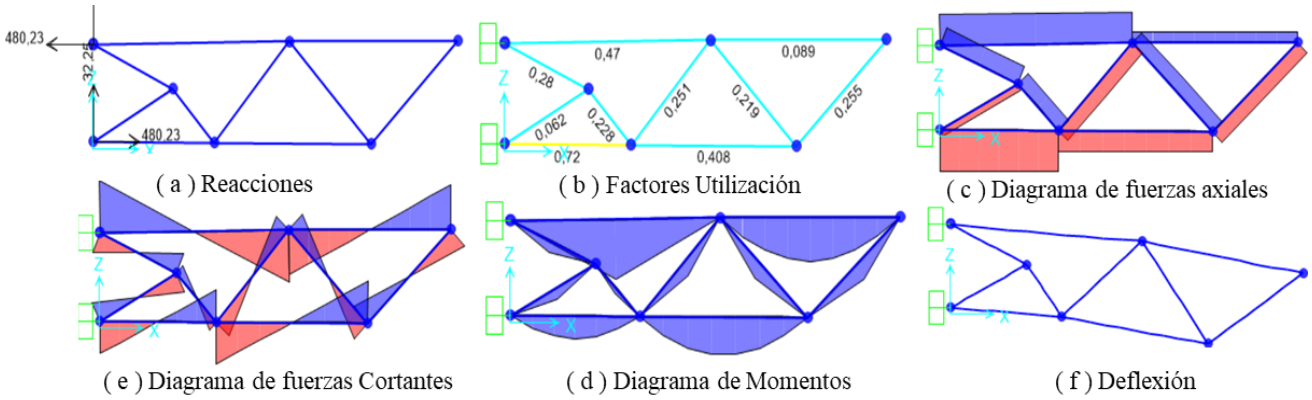

Figura 5.14 Resultados del análisis estructural para viga larga en voladizo con carga puntual de 100 KN en su extremo a partir de celosía propuesta modelada mediante de uniones articuladas en SAP2000®.

En la Tabla 5.13 se comparan los máximos y mínimos globales en el diagrama de fuerzas internas entre el modelo propuesto y el de Gedig (2010), utilizando el análisis de elementos finitos con uniones articuladas en SAP2000®. Se evalúan cinco parámetros clave (fuerza axial, fuerza cortante, momento, deflexión y factor de utilización) para diversos miembros estructurales, indicando en números romanos los miembros que controlaron dicho comportamiento en cada modelo.

Tabla 5.13 Comparación de máximos y mínimos globales del diagrama de fuerzas internas para el modelo de elementos finitos barra a unión articulada en SAP2000 a partir de celosía interpretada por algoritmo propuesto y algoritmo de predicción y reconstrucción Michael Gedig (2018).

	Axial [kN]	**Corte [kN]**	**Momento [kN -cm]**	**Deflexión [cm]**	**FU [%]**
Máximos Gedig	-327,46 **(IX)**	-0,51 **(IV)**	55,02 **(IV)**	1,04 **(I)**	57,5 **(IX)**
Propuesto	-415,27 **(IX)**	-0,50 **(IV)**	53,31 **(IV)**	1,30 **(I)**	72,0 **(IX)**
Mínimos Gedig	282,58 **(IV)**	-0,03 **(VII)**	0,61 **(VII)**	0,15 **(V)**	9,4 **(VIII)**
Propuesto	308,98 **(IV)**	-0,10 **(VII)**	3,43 **(VII)**	0,01 **(V)**	6,2 **(VIII)**

5.4.2 Análisis Estructural mediante Uniones Rígidas

Se compararon ambos modelos mediante análisis adicional con uniones rígidas para demostrar su estabilidad estática bajo condiciones de carga y ofreciendo una visión completa de su rendimiento estructural en ambas configuraciones de conexión.
En la Figura 5.15 y Figura 5.16 se presentan los resultados para los modelos de Gedig y propuesto, con el objetivo es destacar las diferencias y demostrar la estabilidad estática en ambas configuraciones.

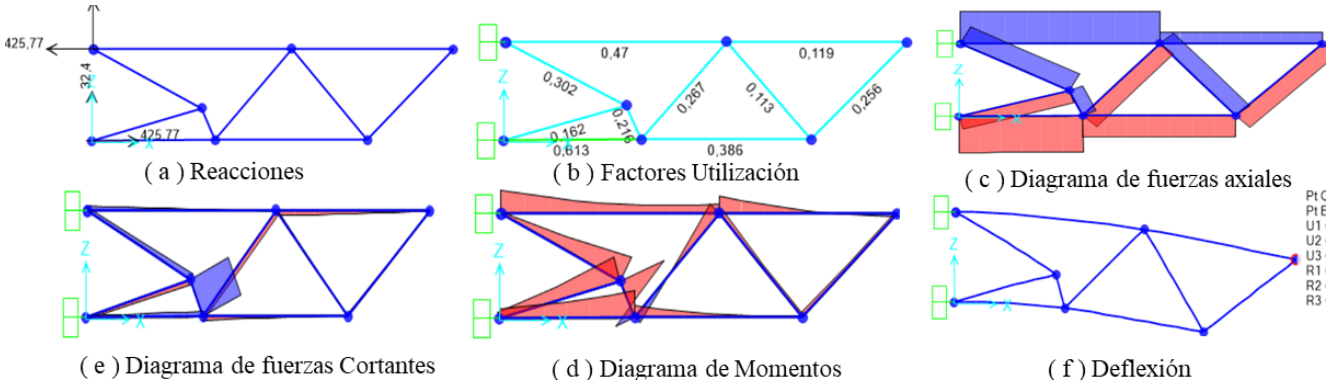

Figura 5.15 Resultados del análisis estructural para viga larga en voladizo con carga puntual de 100 kN en su extremo, a partir de la celosía de Michael Gedig (2018) mediante Uniones Rígidas en SAP2000®.

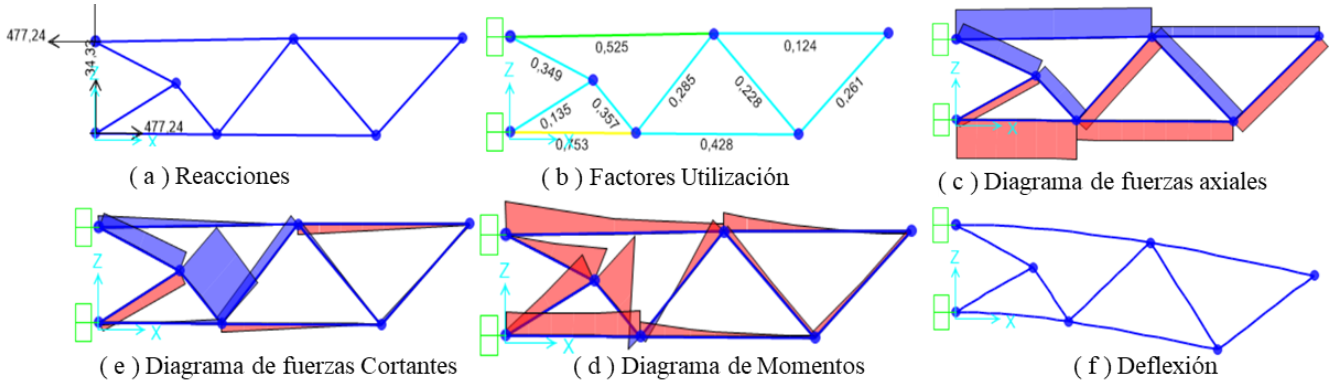

Figura 5.16 Resultados del análisis estructural para viga larga en voladizo con carga puntual de 100 kN en su extremo, a partir de a partir de celosía propuesta modelada mediante Uniones Rígidas en SAP2000®.

Los resultados, obtenidos a partir del modelo de elementos finitos de barras con uniones rígidas en SAP2000®, muestran los valores extremos de fuerzas axiales, cortantes, momentos, deflexiones y factores de utilización (FU) para el modelo de Gedig (2018) y el modelo propuesto. En la Tabla 5.14

se presenta una evaluación comparativa de los modelos para el modelo de elementos finitos de barras con uniones rígidas en SAP2000®.

Tabla 5.14 Comparación de máximos y mínimos globales del diagrama de fuerzas internas para el modelo de elementos finitos barra a unión rígida en SAP2000® a partir de celosía interpretada por algoritmo propuesto y algoritmo de predicción y reconstrucción Michael Gedig (2018).

	Axial [kN]	**Corte [kN]**	**Momento [kN -cm]**	**Deflexión [cm]**	**FU [%]**
Máximos Gedig	-316,36 **(IX)**	9,53 **(VII)**	475,19 **(VII)**	1,03 **(I)**	61,30 **(IX)**
Propuesto	-406,18 **(IX)**	5,47 **(VII)**	539,14 **(VII)**	1,29 **(I)**	75,30 **(IX)**
Mínimos Gedig	281,70 **(IV)**	-0,98 **(VIII)**	42,14 **(II)**	0,16 **(V)**	11,90 **(I)**
Propuesto	307,80 **(IV)**	-1,28 **(VIII)**	44,66 **(II)**	0,10 **(V)**	12,40 **(I)**

En la Tabla 5.15 se presenta un resumen del comportamiento global estructural entre el modelo propuesto y el trabajo de Gedig.

Tabla 5.15 Error ponderado a partir de máximos y mínimos locales del diagrama de fuerzas internas para el modelo de elementos finitos barra a unión articulada en SAP2000® a partir de celosía interpretada por algoritmo propuesto y algoritmo de predicción y reconstrucción Michael Gedig (2010).

Unión Articulada	**Axial [%]**	**Corte [%]**	**Momento [%]**	**Deflexión [%]**	**FU [%]**
Error Global	**14,9**	**28,7**	**59,2**	**36,2**	**29,7**

Unión Rígida	**Axial [%]**	**Corte [%]**	**Momento [%]**	**Deflexión [%]**	**FU [%]**
Error Global	15,0%	46,2%	15,7%	37,0%	25,8%

Las diferencias porcentuales entre los resultados del algoritmo propuesto y los obtenidos por el trabajo de Gedig se atribuyen a la inclusión de una capa de optimización de forma en el trabajo de Gedig, mientras que el algoritmo propuesto se enfoca solo en la interpretación y reconstrucción a partir de la optimización topológica. La optimización de forma complementa la topológica al ajustar la geometría de la estructura de barras para mejorar el rendimiento, lo que se refleja en las menores diferencias porcentuales en algunos parámetros como el corte y el momento para las uniones rígidas y articuladas.

En la Figura 5.17 se muestra el histograma comparativo del comportamiento estructural entre el modelo de celosía con el algoritmo propuesto y el modelo de Gedig.

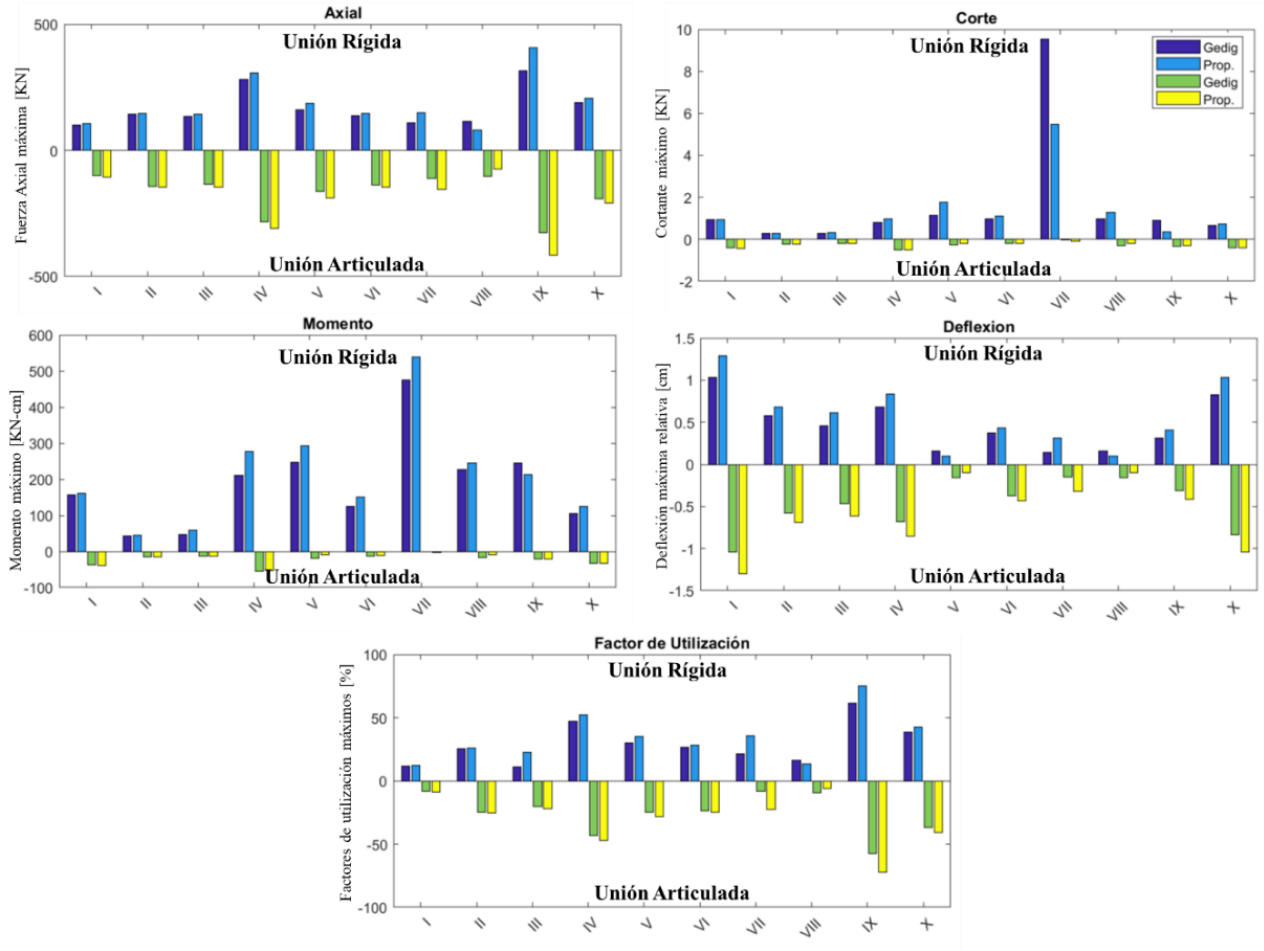

Figura 5.17 Histograma comparativo del comportamiento estructural entre el modelo de celosía con el algoritmo propuesto y el modelo de Gedig (2018) para cada miembro estructural, basado en modelación de elementos finitos de barra a unión Rígida y Articulada mediante software SAP2000®.

5.5 Conclusiones

En este capítulo se validaron ambos algoritmos de reconstrucción e interpretación propuestos a través de dos casos de estudio estructural: una viga larga en voladizo con carga en el extremo y una viga apoyada con carga central. Se observaron diferencias favorables en las proporciones de volumen antes y después del suavizado. Los desplazamientos observados en las estructuras optimizadas reconstruidas mostraron un desempeño adecuado, sugiriendo posible su uso práctico en estructuras reales.

CAPÍTULO 6: CONCLUSIONES

Se desarrollaron una serie de rutinas en MATLAB® capaces de mejorar la generación y adaptación de configuraciones estructurales derivadas de la Optimización Topológica (OT) mediante el método Soft Kill BESO en dos dimensiones. Estas rutinas integran técnicas avanzadas de procesamiento de imágenes basadas en reconocimiento de patrones, filtrado y segmentación de cuencas, alisado y ordenación de bordes, vectorización de esqueletos binarios, modelos de ajuste de curvas suaves y grafos relacionales.

El desarrollo abarcó dos ejes principales: un algoritmo de interpretación de imágenes ráster derivadas de la OT para estructuras tipo celosía, y un algoritmo de reconstrucción mediante alisamiento de bordes y posterior aplicación de chaflanes y filetes. Se integraron rutinas de exportación para análisis estructural en software SAP2000®, Fusion360® y librerías CALFEM® y FEATOOLS® de MATLAB®.

La fase de reconstrucción adoptó estrategias sofisticadas como curvas B-spline y técnicas de chaflán para refinar los contornos segmentados, transformándolos en estructuras tridimensionales aptas para métodos de construcción tradicionales y manufactura aditiva y sustractiva. El comando de celosía predictiva empleó técnicas como curvas de regresión RANSAC, algoritmos de segmentación topológica y construcción de grafos relacionales.

Este desarrollo se validó mediante la aplicación en dos casos de estudio: una viga simplemente apoyada y una viga larga en voladizo, comparando los resultados en términos de fracciones volumétricas, desplazamientos, aspectos geométricos y esfuerzos máximos con investigaciones previas. Las estructuras resultantes cumplieron con las restricciones de volumen y desplazamiento, demostrando la efectividad del enfoque de reconstrucción y reduciendo significativamente la concentración de tensiones y zonas de singularidad geométrica.

El enfoque mejorado presentado en este estudio amplía la aplicabilidad de la OT más allá de las limitaciones identificadas en investigaciones previas, extendiéndose a métodos de construcción tradicionales y manufactura sustractiva. Esto reduce los costos de producción y el material requerido, contribuyendo a la disminución de la huella de carbono sin sacrificar la resistencia o estabilidad estructural.

Aunque el algoritmo mejora significativamente la eficiencia en la generación de estructuras aligeradas, no aborda directamente la estabilidad estructural y puede no detectar inestabilidades inherentes.

Para poder generalizar el modelo obtenido y solucionar las limitaciones de este, se proponen las siguientes líneas futuras de investigación:

- Ampliar la interfaz gráfica existente para incorporar y afinar métodos alternativos de optimización topológica como ESO, BESO y SIMP.
- Extender el algoritmo propuesto hacia un paradigma tridimensional para la extracción de esqueletos y la generación de celosías a partir de datos voxelizados.
- Automatizar el proceso de definición de condiciones de borde iniciales y localización de fuerzas aplicadas.
- Generar rutinas que incorporen la capacidad de importar y manipular geometrías complejas extraídas desde softwares de diseño asistido como AutoCAD® y Revit®.
- Mejorar la integralidad y una comunicación más profunda y directa con SAP2000®, extendiéndose más allá de la simple exportación de geometría y propiedades de material.
- Robustecer la capa de regularización geométrica para integrar parametrización en las distintas configuraciones estructurales resultantes, posibilitando la creación de diseños que no solo sean óptimos en términos de material, sino que también sean ejecutables en la realidad constructiva.

Estas líneas de investigación propuestas son críticas para avanzar en la aplicación práctica de la OT en la ingeniería estructural y representan un enfoque estratégico hacia la solución de problemas complejos de diseño y análisis estructural en la industria contemporánea

NOMENCLATURA

BESO:	Bidirectional Evolutionary Structural Optimization
CAD:	Computer Aided Design
ER:	Evolution Ratio
ESO:	Evolutionary Structural Optimization
FDM:	Fused Deposition Modeling
FV final:	Fracción volumétrica final obtenida
FV inicial:	Fracción volumétrica inicial ingresada como variable de entrada
OT:	Optimización topológica
p:	Factor de penalización
PLA:	Poliácido Láctico
RMIN:	Tamaño de radio mínimo de círculo de filtro de sensibilidades
SIMP:	Solid Isotropic Material with Penalization
STL:	Standard Triangle Language
v:	Módulo de Poisson

REFERENCIAS

Le Tarnec, L., & Garcia, D. (2012). Erratum: Robust smoothing of gridded data in one and higher dimensions with missing values (Computational Statistics and Data Analysis (2010) 54 1167-1178). Computational Statistics and Data Analysis, 56(6), 2182. https://doi.org/10.1016/j.csda.2011.12.001

Lozano, D., Velázquez, F., & Zepeda, A. (2010). Optimización Estructural de Forma en el Diseño de Cavidades en Elementos Planos Mediante Algoritmos Evolutivos. Asociación Argentina de Mecánica Computacional, 24, 1143–1159.

Yin, G., Xiao, X., & Cirak, F. (2020). Topologically robust CAD model generation for structural optimisation. Computer Methods in Applied Mechanics and Engineering, 369, 113102. https://doi.org/10.1016/j.cma.2020.113102

Marconnet, B., Demoly, F., Monticolo, D., & Gomes, S. (2017). An assembly oriented design and optimization approach for mechatronic system engineering. International Journal for Simulation and Multidisciplinary Design Optimization, 8. https://doi.org/10.1051/smdo/2016016

García Arellano, A. (2007). Adelgazamiento y Detección de Bordes de Objetos en Imágenes Digitales Usando Conjuntos Difusos. https://inaoe.repositorioinstitucional.mx/jspui/bitstream/1009/597/1/GarciaAA.pdf

Jootoo, A., & Lattanzi, D. (2018). Extraction of structural system designs from topologies via morphological analysis and artificial intelligence. Designs, 2(1), 1–18. https://doi.org/10.3390/designs2010008

Araujo, M. V. O., Lages, E. N., & Cavalcante, M. A. A. (2020). Checkerboard-free topology optimization for compliance minimization of continuum elastic structures based on the generalized finite-volume theory. Latin American Journal of Solids and Structures, 17(8), 1–21. https://doi.org/10.1590/1679-78256053

Bansal, P., & Kaur, B. (2016). A Review on Thinning in Digital Image Processing. International Journal of Science and Research (IJSR), 5(4), 228–231. https://doi.org/10.21275/v5i4.nov162435

Mendoza San Agustín, A., Velázquez Villegas, F., & Zepeda Sánchez, A. (2016). Adaptation of Topologic Optimized Structures Based on Skeletonization. Ingeniería Mecanica Tecnologia Y Desarrollo, 5(4), 415–421.

Olason, A., & Tidman, D. (2011). Methodology for Topology and Shape Optimization in the Design Process Master's Thesis in the Master's programme Solid and Fluid Mechanics. 74.

Solís Montero, A., & Lang, J. (2012). Skeleton pruning by contour approximation and the integer medial axis transform. Computers and Graphics (Pergamon), 36(5), 477–487. https://doi.org/10.1016/j.cag.2012.03.029

Subedi, S. C., Verma, C. S., & Suresh, K. (2020). A review of methods for the geometric post-processing of topology optimized models. Journal of Computing and Information Science in Engineering, 20(6). https://doi.org/10.1115/1.4047429

Zhang, T. Y., & Suen, C. Y. (1984). A fast parallel algorithm for thinning digital patterns. Communications of the ACM, 27(3), 236–239. https://doi.org/10.1145/357994.358023

Hernández, J. E., & De, E. Y. D. (2021). Estudio y desarrollo de herramientas de optimización topológica para el diseño de mediasuelas de zapatillas.

Uarac, P. (2014). Optimización Topológica de Estructuras Planas considerando Múltiples Restricciones de Desplazamiento. Universidad de Concepción.

Gedig, M. (2010). A framework for form-based conceptual design in structural engineering (Issue April).

Sarma, L. S., Mallikarachchi, C., & Herath, S. (2023). Design-Informed Generative Modelling using Structural Optimization. http://arxiv.org/abs/2308.06734

Leon-medina, J. X. (n.d.). Optimización topológica estructural aplicada en ingeniería mecánica.

Reniers, D. (2009). Skeletonization and Segmentation of Binary Voxel Shapes (Vol. 1, Issue 2009). https://doi.org/10.6100/IR639872

Oviedo, I. (2020) Desarrollo de un modelo de optimización topológica para procesos de manufactura aditiva. Memoria de Título Ingeniero Civil. Departamento de Ingeniería Civil. Universidad de Concepción. Concepción.

ANEXO 1: Diseño de interfaz gráfica para pre y post procesado de algoritmos basados en OT

En esta sección se presenta la interfaz gráfica desarrollada como parte de la herramienta de reconstrucción automatizada, que permite al usuario interactuar directamente con los parámetros y variables involucradas en el proceso de optimización topológica y reconstrucción de estructuras.

En la Figura A.1.1 se muestra un entorno para la definición de condiciones de carga y restricciones en una retícula estructural. Permite ingresar propiedades materiales, seleccionar estrategias de optimización y visualizar en tiempo real los cambios sobre la estructura. Las herramientas integradas para la simulación y visualización de deformaciones realzan su funcionalidad práctica.

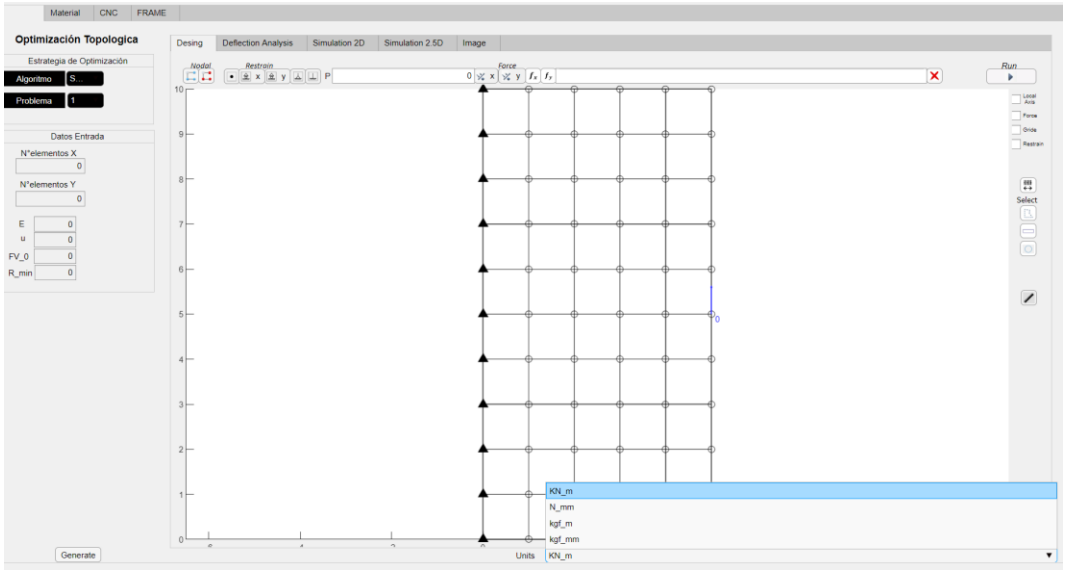

Figura A.1.1 Entorno para la definición de condiciones de carga y restricciones.

La interfaz está diseñada para la entrada y manipulación de datos en un contexto de optimización topológica, ofreciendo al usuario una representación visual de una estructura reticular con puntos nodales y opciones para aplicar fuerzas y restricciones. La claridad de la visualización ayuda a evaluar cómo se distribuyen las fuerzas a lo largo de la estructura, lo cual es vital para ajustes precisos durante el proceso de diseño.

El panel izquierdo de la interfaz permite ingresar datos fundamentales como número de elementos en los ejes X e Y, las propiedades del material (como módulo de elasticidad y coeficiente de Poisson) y parámetros de optimización como el radio mínimo. La capacidad de generar la estructura directamente desde esta interfaz agiliza el flujo de trabajo, optimizando el tiempo de diseño y análisis.

Los controles en la parte superior permiten cambiar entre diferentes análisis y vistas, proporcionando una interacción dinámica con la estructura modelada. Se incluye una funcionalidad para ajustar las unidades de medida, lo cual es crucial para asegurar la precisión en los cálculos y la interpretación de los resultados, adaptándose a diferentes estándares o preferencias regionales.

Las formas geométricas y polilíneas se utilizan para especificar zonas de influencia, aplicar fuerzas, momentos o restringir desplazamientos en patrones que no se limitan a selecciones individuales o rectangulares simples. En la Figura A.1.2 se ilustra cómo esta herramienta permite una personalización avanzada en la definición de condiciones de carga y restricciones en la estructura modelada.

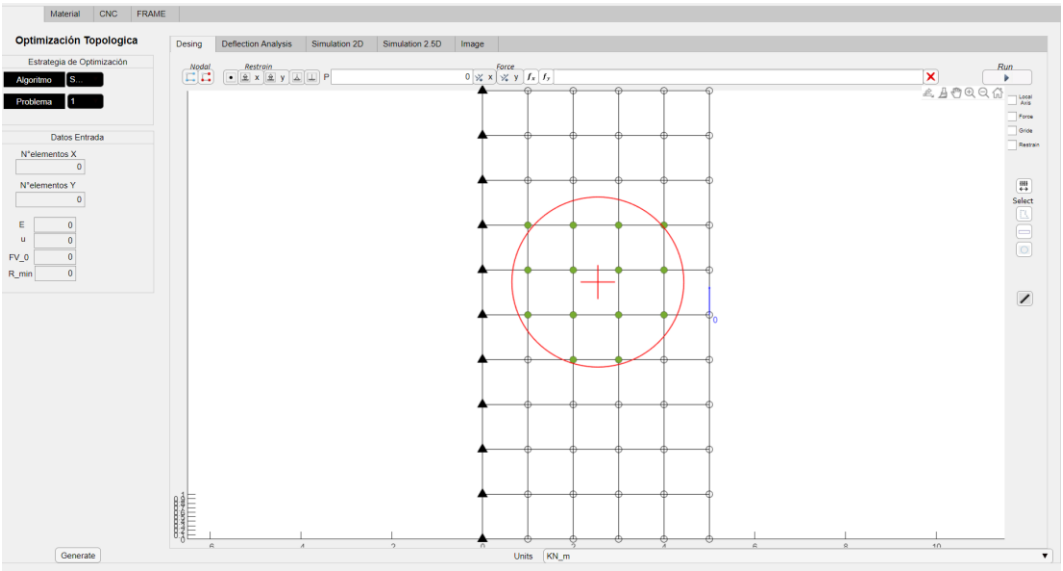

Figura A.1.2 Herramienta de selección nodal a partir de formas geométricas y polilíneas

Las polilíneas, por otro lado, ofrecen una personalización aún mayor. Se pueden trazar líneas que conectan una serie de nodos siguiendo trayectorias específicas, lo que es particularmente útil para aplicar cargas a lo largo de caminos definidos o para simular condiciones de contorno que siguen un perfil irregular.

Una vez seleccionados los nodos mediante la interfaz gráfica, el siguiente paso es la asignación de cargas y la aplicación de restricciones de desplazamiento, entre otras condiciones de contorno. En la Figura A.1.3 se presenta el módulo que permite a los usuarios especificar condiciones directamente sobre la geometría del modelo. Esto incluye la aplicación de cargas puntuales, distribuidas o momentos, y la imposición de condiciones de fijación o desplazamiento en los nodos seleccionados.

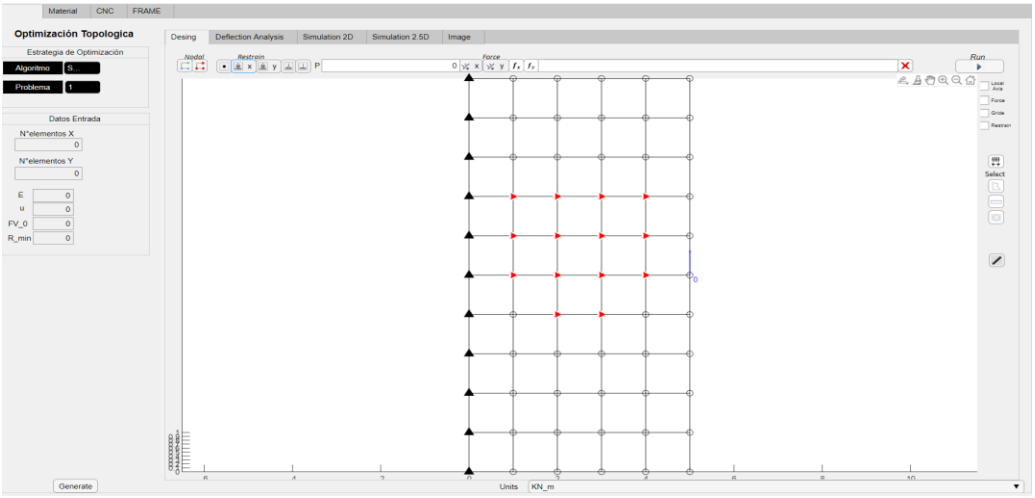

Figura A.1.3 Asignación de cargas y la aplicación de restricciones de desplazamiento

La interfaz permite realizar y almacenar múltiples configuraciones sin pérdida de datos previos proporcionando una ventaja significativa. Esto no solo incrementa la eficiencia y la productividad del diseñador, sino que también permite una exploración más amplia de posibles soluciones estructurales sin el temor de perder configuraciones valiosas por errores o cambios no deseados. En la Figura A.1.4 se muestra la visualización de múltiples condiciones de carga y desplazamiento en la interfaz.

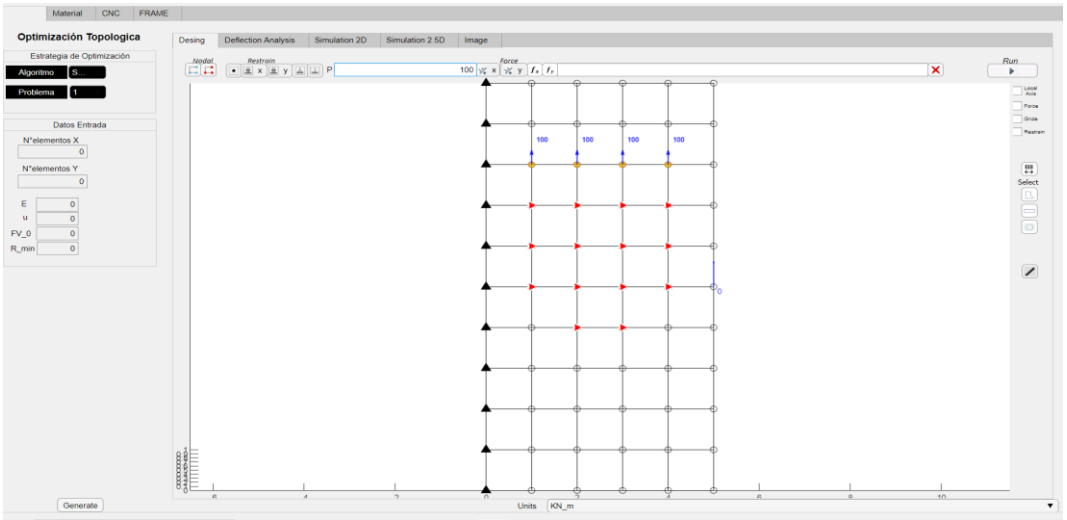

Figura A.1.4 Visualización de múltiples condiciones de carga y desplazamiento en la interfaz

La sección "Material" mostrada en la Figura A.1.5 incluye un menú desplegable que permite al usuario seleccionar "Madera" como el material de construcción, con el objetivo de manejar múltiples tipos de materiales y ajustes específicos para cada uno de ellos.

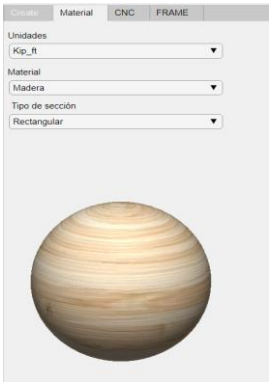

Figura A.1.5 Selección de material y configuración de sección en la interfaz de diseño estructural.

Esta sección permite evaluar el rendimiento estructural antes de la optimización, facilitando decisiones informadas sobre la viabilidad y seguridad del diseño.

A su vez permite evaluar el rendimiento estructural antes de la optimización, mediante un análisis de elementos finitos cómo se ilustra en Figura A.1.6. La gama de colores, desde el azul hasta el rojo, sugiere un gradiente que representa una magnitud física, la deflexión o el estrés, con el rojo indicando valores altos y el azul valores bajos.

El menú desplegable en la parte inferior derecha parece permitir la selección del tipo de análisis, como deformación, tensión de Von Mises, momentos, entre otros.

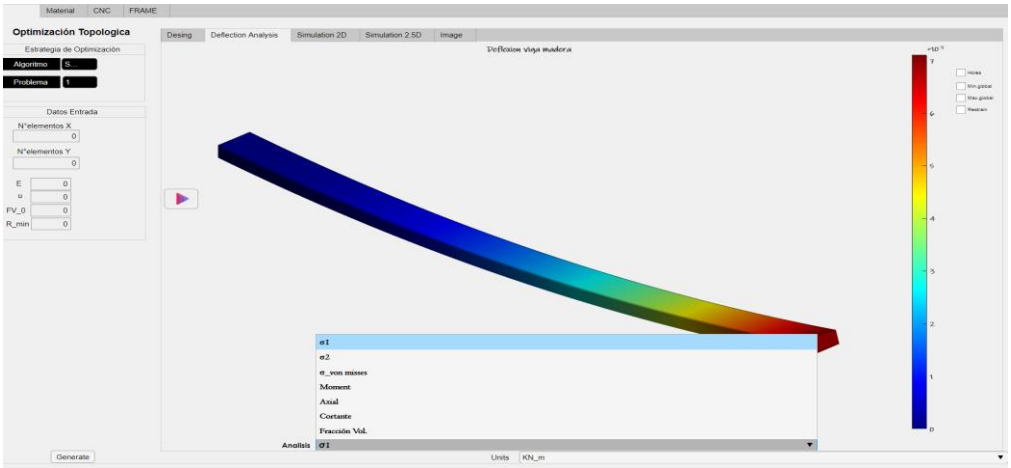

Figura A.1.6 Análisis de elementos finitos pre-optimización con gradiente de colores para la evaluación del estrés y deformada.

La interfaz muestra una sección que facilita la reconstrucción y visualización de modelos estructurales. Se destacan opciones para redondeo, filtrado, renderizado y formatos de exportación, como STL y OBJ, que son comunes en la impresión 3D y el análisis estructural. En la Figura A.1.7 se presenta la cinta de opciones de redondeo, filtrado y renderizado, así como los comandos de exportación en formatos STL y OBJ.

Se permite al usuario optar por utilizar configuraciones preestablecidas para una reconstrucción rápida o ajustar parámetros específicos para obtener resultados personalizados. Esto permite una flexibilidad en el modelado y puede ser particularmente útil para iterar rápidamente entre diseños conceptuales y soluciones detalladas. La capacidad de influir en la interpretación y los resultados del modelo mediante la manipulación de parámetros ayuda al usuario a adaptar el proceso de diseño a requisitos específicos o a explorar diversas soluciones estructurales.

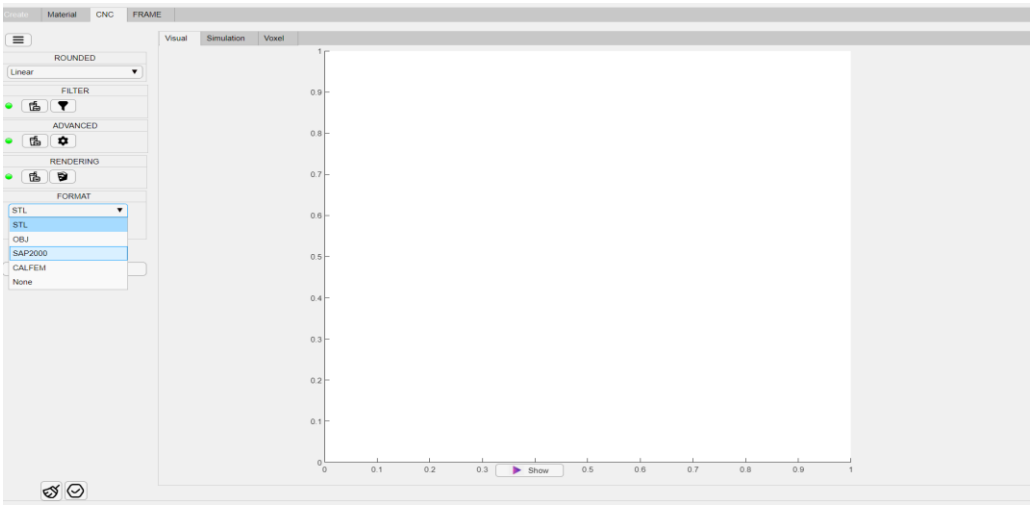

Figura A.1.7 Interfaz de opciones avanzadas para personalización y exportación de modelos estructurales

La interfaz gráfica integra un algoritmo que permite la identificación precisa de elementos como caras y vértices. Este nivel de detalle es crucial para aplicar nuevas cargas o restricciones y refinar modelos a través de iteraciones de optimización. Las etiquetas para cada nodo y la representación en 3D permiten una mejor interpretación y manipulación del modelo en el proceso de optimización estructural. En la Figura A.1.8 se presenta un ejemplo de cómo se identifican estos elementos estructurales en la interfaz gráfica, mostrando claramente las etiquetas de los nodos y las caras en un modelo tridimensional

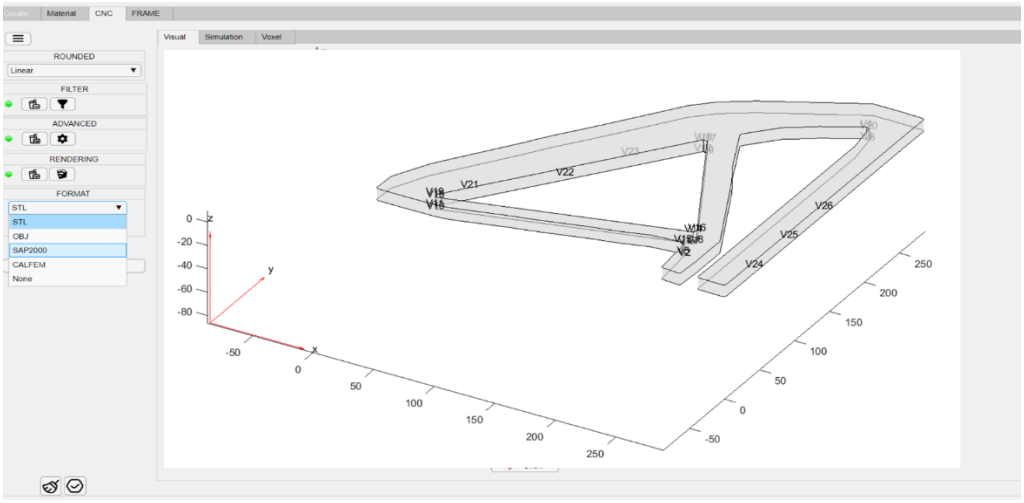

Figura A.1.8 Creación de un modelo 3D basado en la reconstrucción de resultados derivados de la optimización topológica.

El modelo 3D es el resultado de un proceso de suavizado y refinamiento de la estructura originalmente optimizada, lo cual es crucial para aplicaciones prácticas de ingeniería. Esto asegura que el diseño no solo sea funcional, sino también fabricable, y cumpla con los estándares de seguridad y funcionalidad. En la Figura A.1.9 se muestra un ejemplo de la estructura optimizada, reconstruida y simulada mediante la librería FEATOOLS® de MATLAB®, representada en un modelo tridimensional exportable para manufactura aditiva y sustractiva.

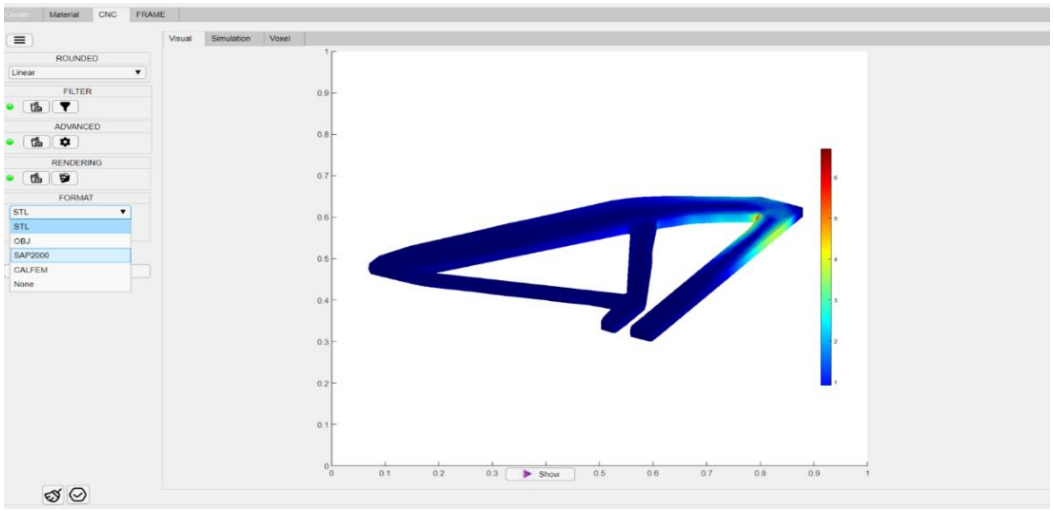

Figura A.1.9 Evaluación de deformación y estrés en el modelo 3D optimizado y reconstruido.

ANEXO 2: VIGA CORTA EN VOLADIZO

Para este apartado se realizó un análisis comparativo del desempeño de una viga en voladizo con dimensiones de 80x50 milímetros con carga puntual de 100 [N] en su extremo, reconstruida e interpretada mediante el algoritmo propuesto, en contraste con los resultados obtenidos a partir del trabajo previo de Oviedo (2020). Las propiedades del hormigón y parámetros que gobiernan el proceso de optimización se muestran detallados en la Tabla A.2.1.

Tabla A.2.1 Parámetros de estructura óptima de viga simplemente apoyada 120x40 milímetros que cumple restricción de desplazamientos

L [mm]	H [mm]	Malla	E [MPa]	V	P [N]	Tamaño máx. malla triang. [mm]	Decimales	Δ admisible [mm]
60	40	60x40	23500	0,25	100	1,5	1	L/480

En la Figura A.2.1 se presentan las representaciones resultantes entre el presente estudio y la reconstrucción efectuada por Oviedo (2020). Respecto a la disposición estructural que se fundamenta en el suavizado y redondeo de bordes, detallada en la misma figura, se introduce un parámetro de filete de 4 mm para cada punto característico identificado.

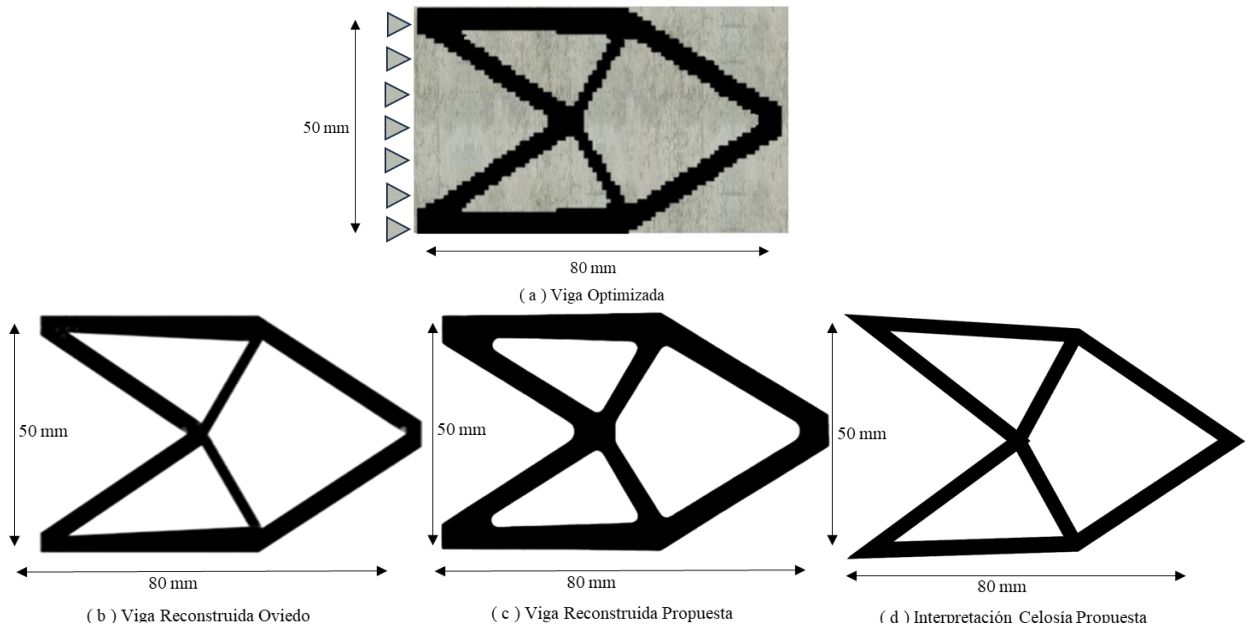

Figura A.2.1 Optimización para viga corta en cantiléver de hormigón con carga puntual de 100 N en el extremo: a) Configuración obtenida mediante algoritmo OT, b) Configuración obtenida a partir de algoritmo de Oviedo, c) Configuración obtenida a partir de algoritmo propuesto basado en suavizado y redondeo de bordes, d) Configuración obtenida a partir de algoritmo propuesto basado en predicción de celosía estructural.

En contraposición, para la configuración derivada de la predicción e interpretación de la celosía, se lleva a cabo un redimensionamiento de los elementos de barras mediante secciones rectangulares de 5 mm de altura y 1 mm de ancho. Este ajuste se realiza con el propósito de cumplir con las deformaciones admisibles especificadas en la Tabla A.2.1.

A.2.1 Geometría de Oviedo

En relación con la distribución de tensiones, Oviedo (2020) efectuó un análisis de los esfuerzos principales máximos y sus correspondientes desplazamientos, utilizando un enfoque estructural basado en elementos finitos Q4 a través de la librería CALFEM® de MATLAB®. En la Figura A.2.2 se muestra el estado tensional resultante de Oviedo.

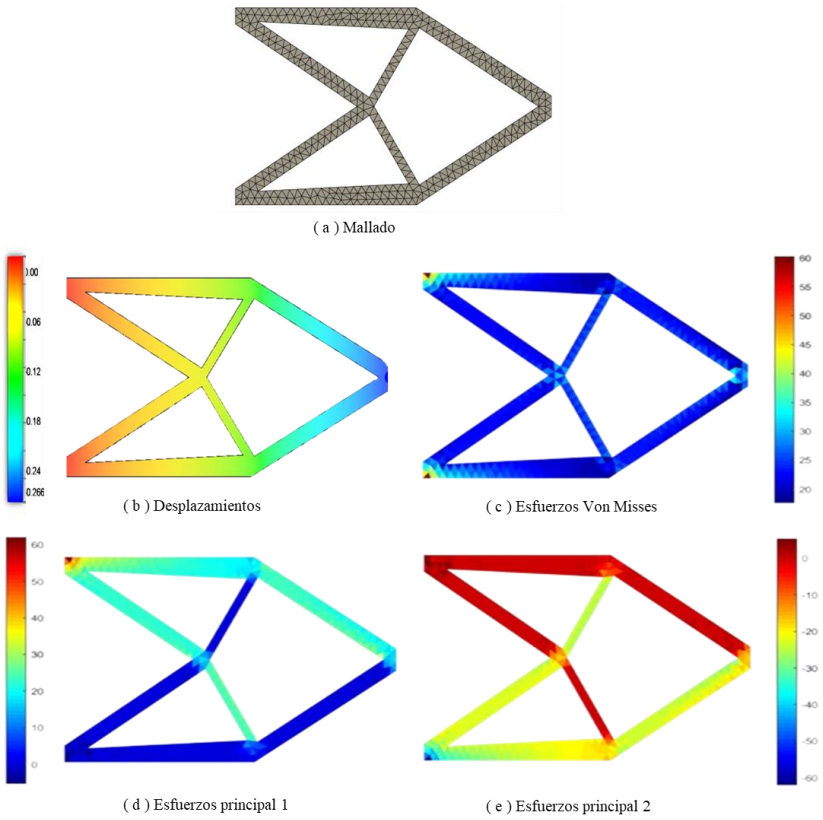

Figura A.2.2 Estado **tensional de estructura de Oviedo óptima para viga corta en cantiléver de 80x50 milímetros con carga puntual de 100 N en el extremo, mediante librería CALFEM® de MATLAB®: a) Mallado, b) Desplazamientos,** $c) \sigma_{VM}, d) \sigma_1, e) \sigma_2$

Los parámetros para obtener la estructura óptima se indican en la Tabla A.2.2 y el desplazamiento admisible se alcanza entre el 20 % y 30 % de la FV final.

Tabla A.2.2 Resultados de estructura de Oviedo óptima para viga corta en cantiléver de 80x50milímetros que cumple restricción de desplazamiento. (MATLAB®)

FV inicial [%]	FV final [%]	Diferencia [%]	Δ máximo final [mm]	Δ admisible [mm]	Límite a Δ admisible [%]
32	27,1	4,9	0,265	0,266	99,4

La Tabla A.2.3 exhibe los esfuerzos principales máximos obtenidos para cada uno de los estados tensionales presentados en la Figura A.2.2.

Tabla A.2.3 Esfuerzos principales máximos y mínimos de estructura de Oviedo óptima para viga corta en cantiléver con carga puntual en el extremo (MATLAB®)

Esfuerzos [MPa]	σ_1 [MPa]	σ_2 [MPa]	σ_{VM} [MPa]
Máximo	62,07	5,41	60,27
Mínimo	-5,41	-62,07	17,56

Para la estructura obtenida por Oviedo se observa una fracción de volumen final del 27.1%, con una diferencia del 5% en comparación con la topología optimizada con bordes irregulares. La deformación máxima en el centro es de 0.265 mm, idéntica a la deformación admisible de 0.266 mm.

La Tabla A.2.3 presenta los esfuerzos principales máximos y mínimos de la estructura, expresados en MPa. Se destaca un esfuerzo máximo de 62.07 MPa en σ_1, mientras que el esfuerzo mínimo es -62.07 MPa en σ_2. El esfuerzo de Von Mises alcanza un máximo de 60.27 MPa, con mayores concentraciones de esfuerzos en áreas de singularidad geométrica.

A.2.2 Geometría propuesta: Algoritmo suavizado de forma

En referencia a la distribución de tensiones para el modelo suavizado propuesto, se ejecutó un análisis de los esfuerzos principales máximos y los desplazamientos correspondientes mediante la aplicación de un enfoque de elementos finitos tetraédricos. Este análisis se llevó a cabo utilizando la librería de simulación FEATOOLS® incorporada en MATLAB®.

Los parámetros utilizados para obtener la estructura óptima se muestran en la Tabla A.2.4, mientras la fracción de volumen FV posreconstrucción se modifica desde el 32 % de la FV hasta el 33.9 %.

Tabla A.2.4 Resultados de la estructura optima reconstruida mediante algoritmo de suavizado propuesto para viga corta en cantiléver de 80x50 milímetros que satisface la restricción de desplazamiento. (MATLAB)

	FV inicial [%]	FV final [%]	Diferencia [%]	Δ máximo final [mm]	Δ admisible [mm]	Límite a Δ admisible [%]
MATLAB	32	33,9	-1,9	0,221	0,266	82,9

La elevación del 1.9% en la Fracción Volumétrica (FV) se debe a la implementación de agujeros basados en tamaños nominales reales durante la reconstrucción, lo cual introdujo restricciones geométricas al proceso. Técnicamente, las limitaciones geométricas del redondeo se relacionan con la necesidad de mantener tangencias entre los círculos de redondeo, lo cual, al aplicar tamaños nominales reales, limita la flexibilidad en la manipulación de las dimensiones y contribuye al aumento en la FV final. En la Figura A.2.3 se presentan los resultados del estado tensional y deformaciones efectuado.

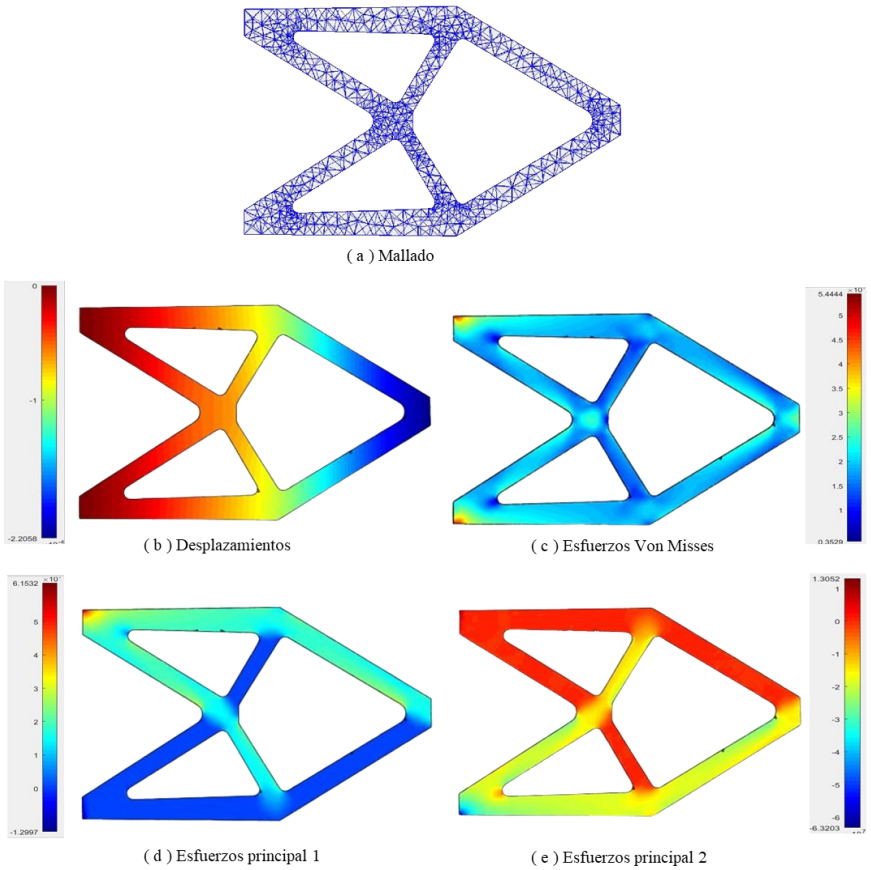

Figura A.2.3 Estado tensional de estructura óptima mediante el algoritmo de suavizado propuesto para viga corta en cantiléver de hormigón con dimensiones de 80x50 milímetros, sometida a una carga puntual en el extremo. La simulación se realiza utilizando el motor de físicas FEATOOLS® de MATLAB®: a) Representación gráfica de la discretización de la estructura, b) Evaluación de las deformaciones y movimientos en diferentes puntos de la viga, c) σ_{VM}, d) σ_1, e) σ_2

La Tabla A.2.5 presenta los esfuerzos principales máximos obtenidos para cada uno de los estados tensionales mostrados en la Figura A.2.3.

Tabla A.2.5 Esfuerzos principales máximos y mínimos para viga corta cantiléver con carga puntual en el extremo reconstruida mediante algoritmo de suavizado propuesto (MATLAB®)

Esfuerzos [MPa]	σ_1 [MPa]	σ_2 [MPa]	σ_{VM} [MPa]
Máximo	61,53	13,05	46,96
Mínimo	-12,99	-63,20	3,52

La deformación máxima en el centro de 0.221 mm, inferior a la deformación admisible, indica una respuesta estructural satisfactoria ante la carga aplicada, satisfaciendo los requerimientos de servicialidad.

Para la Tabla A.2.5 se observan concentraciones de esfuerzos menores que en la geometría propuesta por Oviedo, alcanzando máximos y mínimos de 61.53 MPa y -63.20 MPa, superior a los valores típicos de resistencia del hormigón, lo que indica áreas de preocupación.

Los resultados obtenidos son comparados mediante el software de modelación FUSION360® de Autodesk® para la misma geometría, haciendo uso de elementos finitos tetraédricos. En la Figura A.2.4 se presentan los resultados del estado tensional y deformaciones efectuado en FUSION360® para elementos finitos tetraédricos.

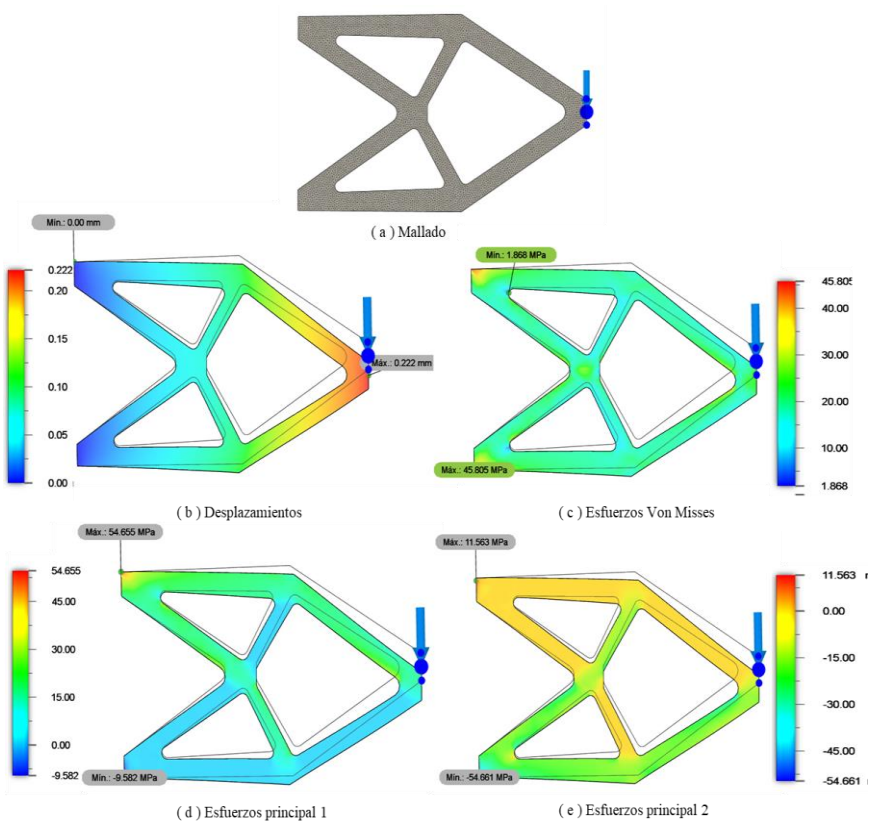

Figura A.2.4 Estado tensional de estructura óptima mediante el algoritmo de suavizado propuesto para viga corta en cantiléver de hormigón sometida a una carga puntual en el extremo. La simulación se realiza utilizando FUSION360® a) Representación gráfica de la discretización de la estructura, b) Evaluación de las deformaciones y movimientos en diferentes puntos de la viga, c) σ_{VM}, d) σ_1, e) σ_2

En la Tabla A.2.6 se comparan los resultados iniciales y finales de fracción de volumen (FV) para la misma geometría mediante FUSION360®.

Tabla A.2.6 Resultados de la estructura optima reconstruida mediante algoritmo de suavizado propuesto para viga corta en cantiléver de 80x50 milímetros que satisface la restricción de desplazamiento. (FUSION360®)

FV inicial [%]	FV final [%]	Diferencia [%]	Δ máximo final [mm]	Δ admisible [mm]	Límite a Δ admisible [%]
32	33,9	-1,9	0,222	0,2666	83,3

Se destaca una deformación máxima final es de 0.222 mm, inferior al límite admisible de 0.266 mm e idéntica a la obtenida en MATLAB®. En la Tabla A.2.7 se presentan los esfuerzos máximos y mínimos obtenidos.

Tabla A.2.7 Esfuerzos principales máximos y mínimos de estructura óptima para viga corta cantiléver con carga puntual en el extremo (FUSION360®)

Esfuerzos [MPa]	σ_1 [MPa]	σ_2 [MPa]	σ_{VM} [MPa]
Máximo	54,66	11,56	45,80
Mínimo	-9,58	-54,66	1,86

Se visualizan diferencias menores a los ± 1 MPa respecto a las tensiones de Von Mises en comparación con las presentadas en Tabla A.2.5.

A.2.3 Geometría propuesta: Algoritmo de interpretación por celosía

La reconstrucción generada, fue analizada por tramos y comparada utilizando elementos finitos tetraédricos y tipo barra, mediante 2 softwares aceptados por la comunidad. En la Figura A.2.5 se presenta una representación visual comparativa de las tensiones de Von Mises obtenidas.

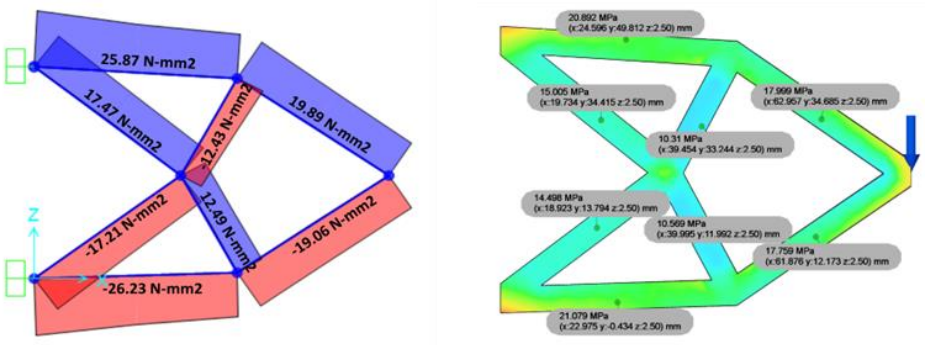

Figura A.2.5 Distribución de tensiones de Von Misses en MPa. (a) Modelo de elementos finitos de barra en SAP2000® a partir de celosía interpretada por algoritmo propuesto. (b) Modelo de elementos finitos tetraédricos Fusion360® a partir de celosía interpretada por algoritmo propuesto.

La Tabla A.2.8 presenta un resumen comparativo de las tensiones de Von Mises obtenidas para cada tramo de la celosía mediante el algoritmo propuesto. Las diferencias porcentuales destacan la variación entre las simulaciones realizadas con ambos softwares.

Tabla A.2.8 Resumen comparativo de tensiones de Von misses por tramos a partir de celosía interpretada (Algoritmo Propuesto), exportadas y simuladas mediante elementos finitos tetraédricos (FUSION360®) y elementos finitos de barra (SAP2000®).

SVM	I	II	III	IV	V	VI	VII	VIII
	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]	σ_{VM} [MPa]
SAP2000	17,86	26,2	20,25	19,18	26,7	17,74	12,63	13,05
FUSION360	15,01	20,89	18	17,76	21,03	14,5	10,57	10,31
Diferencia	16%	20%	11%	7%	21%	18%	16%	21%

La comparación revela variaciones importantes en las tensiones de Von Mises, alcanzando diferencias de magnitud de hasta un 25%, destacando la influencia del elemento finito tipo barra, que tiende nuevamente a sobrestimar los esfuerzos desarrollados en la geometría analizada, como se detalla en la Tabla A.2.8. Se visualiza además que todos los miembros se encuentran por debajo de los 30 MPa, valor típico de resistencia a compresión alcanzado por una amplia gama de hormigones. Sin embargo, es relevante señalar que todos estos valores superan significativamente la resistencia típica a tracción, que generalmente constituye alrededor del 10% de la resistencia a compresión.

En la Figura A.2.6 se presentan las deformaciones generadas por la celosía interpretada bajo restricciones de desplazamiento en puntos clave de la estructura.

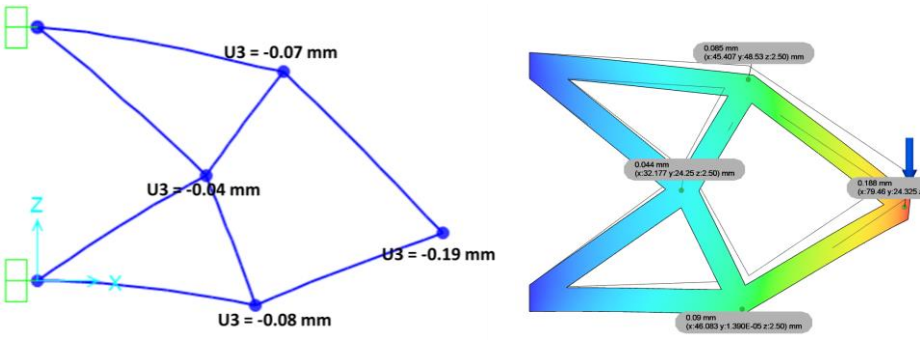

Figura A.2.6 Deformada a partir de celosía interpretada bajo restricción específica de desplazamiento establecida en mm. (a) Modelo de elementos finitos de barra en SAP2000® a partir de celosía interpretada por algoritmo propuesto. (b) Modelo de elementos finitos tetraédricos Fusion360® a partir de celosía interpretada por algoritmo propuesto.

La Tabla A.2.9 presenta una comparación detallada de las deflexiones nodales en puntos característicos derivados de la interpretación de una celosía bajo restricción específica de desplazamiento. Estas deflexiones fueron calculadas para dos tipos de modelado: utilizando elementos de barra en SAP2000® y elementos tetraédricos en FUSION360®. Las unidades de medida de las deflexiones se expresan en milímetros.

Tabla A.2.9 Resumen comparativo de las deflexiones nodales en mm a partir de celosía interpretada bajo restricción específica de desplazamiento establecida para celosía tipo elementos barra (SAP2000®) y elementos tetraédricos FUSION 360®.

Deflexión	P_1	P_2	P_3	P_4	P_5	P_6
	δ_d [mm]	δ_d [mm]	δ_d [mm]	δ_d [mm]	δ_d [mm]	δ_d [mm]
SAP2000	0	0,075	0,184	0,076	0	0,043
FUSION360	0	0,085	0,188	0,09	0	0,044
Diferencia	-	13%	2%	18%	-	2%

La consistencia razonable entre las deformadas y deflexiones nodales presentadas en la Figura A.2.6 indica respuestas similares entre SAP2000® y Fusion360®. No obstante, las diferencias en las deflexiones, especialmente en el nodo P_4, alcanzan hasta un 18%, como se muestra en la Tabla A.2.9. Sin embargo, en deflexiones máximas esperadas se tienen respuestas casi idénticas con una diferencia solo de 2% y ampliamente por debajo de la deformación admisible. Finalmente, en la Tabla A.2.10 se presenta un análisis detallado de las fuerzas internas por miembro para la configuración estructural basada en la celosía interpretada obtenida.

Tabla A.2.10 Resumen por miembro del diagrama de fuerzas internas para modelo de elementos finitos de barra en SAP2000® a partir de celosía interpretada por algoritmo propuesto

Miembro	Axial [N]	Corte [N]	Momento [N-mm]	Deflexión [mm]
I	75,28	0,16	11,68	0,043
II	98,96	-0,65	26,72	0,089
III	89,57	-0,28	9,71	0,095
IV	-89,45	0,09	5,39	0,092
V	-99,76	0,69	28,10	0,092
VI	-72,76	-0,23	13,29	0,043
VII	52,56	0,06	8,84	0,049
VIII	-49,93	0,42	12,77	0,046

ANEXO 3: Rutina MATLAB® para reconstrucción de imágenes ráster por suavizado de bordes

```
clc
clear all
close all;


% Cargar la imagen y convertirla en binaria
img = imread('Raster OT.jpg');
img = imresize(img, 1);

figure(109)
% Convertir la imagen a escala de grises
img_gray = rgb2gray(img);
% Realizar una operación de suavizado
%img_smooth = imgaussfilt(img_gray,0.30);%(opcional)
img_smooth = imgaussfilt(img_gray,2.39);
% Aplicar la transformada de distancia
img_dist = bwdist(~imbinarize(img_smooth));
% Aplicar la umbralización
%img_bw = imbinarize(img_dist, 0.001*max(img_dist(:)));
img_bw = imbinarize(img_dist, 0.049*max(img_dist(:)));
montage({img_gray,img_smooth, img_dist,img_bw}, 'Size', [4, 1]);

% Dimensiones de la imagen 2D
[filas, columnas] = size(img_gray);
 % Escala de valores de la imagen 2D a valores de opacidad
(transparencia)
% Puedes ajustar esta escala según tus necesidades
opacity_scale = imbinarize(img_gray); % Escala entre 0 y 1

% Definir la matriz 3D para los voxels
% En este ejemplo, se crea una matriz de 3D con la misma resolución
que la imagen 2D
% Puedes ajustar las dimensiones según tus necesidades
depth = 15; % Profundidad deseada en voxels
voxels = zeros(filas, columnas, depth);

% Asignar los valores de opacidad a los voxels en función de la
escala
for d = 1:depth
    voxels(:,:,d) = opacity_scale;
end
% Identificar y establecer como NaN los valores cercanos a 1
umbral = 0.90; % Umbral para identificar valores cercanos a 1
voxels(abs(voxels - 1) < umbral) = NaN;
```

```
figure(107)
% Visualizar los voxels semitransparentes
%vol3d('CData', voxels);
%daspect([1 1 1])
%colormap bone;
axis tight;
xlabel('X');
ylabel('Y');
zlabel('Profundidad');



% Obtener las coordenadas (x, y) de los píxeles blancos
[xi, yi] = find(((img_bw-1)*-1));

% Realizar el cálculo del perímetro exterior usando boundary
k = boundary(xi, yi, 0.999);
%perimeterpixel=k;
% Obtener las coordenadas (x, y) del perímetro exterior
perimeter_x = xi(k);
perimeter_y = yi(k);
perimeterpixel = sub2ind(size(img_bw), perimeter_x, perimeter_y);
T=zeros(size(img_bw));
T(perimeterpixel)=1;
% Elemento estructurado para la dilatación (radio 2)
radio_dilatacion = 2;
se = strel('disk', radio_dilatacion);

% Aplicar dilatación
T = imdilate(T, se);
% Visualizar el resultado
figure(88)
imshow(T);
% Visualizar el resultado
% Realizar una operación de erosión para eliminar las áreas
pequeñas
se = strel('disk', 1);
img_eroded = imerode(img_bw, se);

% Realizar una operación de dilatación para unir las áreas que
quedaron separadas
se = strel('disk', 1);
img_dilated = imdilate(img_eroded, se);

% Realizar la segmentación
img_segmentada = img_gray .* uint8(img_dilated);

% Mostrar la imagen segmentada
```

```
figure(2)
imshow(img_dilated);



%Inicializamos la imagen
%I = rgb2gray(imread('prueba17.jpg'))
%procedo a binarización de esta
BW=img_dilated
%ELIMINACION DE RUIDO
BW = bwareaopen(BW, 35);

%Llamamos al mapa de distancias euclideanas
D_euclidean = bwdist(BW,'euclidean');
BWJ = imcomplement(BW);

%_____________________________________________________________________
im=BW
figure(12)
imd=bwdist(BW);
subplot(2,2,1)
imshow(imd,[]);
title('Euclidean Distance Transform');

%imw=watershed(imd);
%imw=imw==0;
log=del2(imd);
subplot(2,2,2)
imshow(log,[]);
title('Laplacian of Gaussian');
imp=bwperim(im);
se=strel('disk',1);
imp=imdilate(imp,se);

bin=log < -0.2;
mask=bin-imp;
mask=mask>0;
subplot(2,2,3)
imshow(mask,[]);
title('Binazired mask');

rp = regionprops(mask, 'PixelIdxList', 'Area');
rp = rp(vertcat(rp.Area) > 30);

centerline=zeros(size(im));
centerline(vertcat(rp.PixelIdxList))=1;
centerline=bwmorph(centerline,'thin');
subplot(2,2,3)
```

```
imshow(centerline,[]);
title('Final mask');


L = watershed(D_euclidean);


L(~BW) = 0;
rgb = label2rgb(L,'jet',[.5 .5 .5]);
figure(9)
imshow(rgb)
title('Watershed Transform')


figure(11)
imcontour(L,20)


% aplicar watershed a la imagen binaria
imagen_etiquetada = watershed(~img_dilated);

% extraer el perímetro de las formas capturadas
perimetro = bwperim(L);

figure(3)
% visualizar el resultado
imshow(perimetro);
capa_perimetro=perimetro;
% Crear el marco
% Espesor del marco en píxeles
espesor_marco = 1;
imagen_con_marco = zeros(size(perimetro) - 2*espesor_marco);
imagen_con_marco = padarray(imagen_con_marco, [espesor_marco,
espesor_marco], 1, 'both')
perimetro=perimetro-T;
perimetro(perimetro < 0) = 0;



[Bi,L,N] = bwboundaries(BW,'holes')


% Recorrer todos los objetos de la imagen etiquetada
%for i=1:max(imagen_etiquetada(:))
    % Obtener el perímetro del objeto actual
    %perimetro_objeto = (imagen_etiquetada==i) & perimetro;
    % Obtener los índices de los píxeles del perímetro del objeto
actual
```

```
   % indices_perimetro = find(perimetro_objeto);
    % Si se encontraron píxeles en el perímetro del objeto actual
    %if ~isempty(indices_perimetro)
        % Hacer cero el primer elemento del vector
      %  perimetro(indices_perimetro(1)) = 0;
    %end
%end
nn=perimetro;
% Obtener el tamaño de la imagen
[filas, columnas] = size(nn);
r_agujero=5
% Crear una imagen con un agujero en la esquina
agujero = zeros(filas, columnas);
agujero(1:r_agujero, 1:r_agujero) = 1; % Aquí puedes ajustar el
tamaño del agujero

% Aplicar la operación 'AND' para crear el agujero en la esquina de
la imagen
nn = nn & ~agujero;


nn = bwmorph(nn, 'spur', 'Inf');
nn = cleanSkel(bwmorph(nn,'skel',Inf),[]);
% Definir el elemento estructurante
%elemento_estructurante = strel('square', 3);
figure(20)
imshow(nn)
% Aplicar la erosión
%imagen_binaria_erodada = imerode(nn, elemento_estructurante);

% Convertir la imagen binaria de vuelta a una matriz de valores 0 y
1
%nn = double(nn-imagen_binaria_erodada);
capa_total_holes = padarray(BW, [espesor_marco, espesor_marco]*5,
1, 'both')
capa_total_holes=bwperim(capa_total_holes)
capa_holes_int = padarray(nn, [espesor_marco, espesor_marco]*5, 0,
'both')
capa_exterior_hole=capa_total_holes-capa_holes_int;
imagen_recortada = capa_exterior_hole(espesor_marco*5:end-
espesor_marco*4, espesor_marco*5:end-espesor_marco*4);
imagen_recortada(imagen_recortada < 0) = 0;
figure(4)

imshow(imagen_recortada)
% Obtener la nueva imagen etiquetada usando bwconncomp
%nn = padarray(nn, [espesor_marco, espesor_marco], 0, 'both')
NNn = bwconncomp(nn, 8);
```

```
% Obtener las regiones conectadas en la imagen
Contourpixel = bwconncomp(imagen_recortada, 8);
% Obtener las propiedades de cada región conectada
propiedades = regionprops(Contourpixel, 'Area');
% Encontrar el índice de la región con el área más grande
areas = [propiedades.Area];
[~, indice_max_area] = max(areas);

[indcx,indcy]=ind2sub(size(imagen_recortada),
Contourpixel.PixelIdxList{indice_max_area})
contour
=bwtraceboundary(imagen_recortada,[indcx(1),indcy(1)],'SE',8,Inf,'c
lockwise');;
%perimeterpixel = sub2ind(size(imagen_recortada), contour(:,1),
contour(:,2));
%NNn.PixelIdxList = Bi';
figure(1)

plot(contour(:,1),contour(:,2),'r','LineWidth',2)

figure(6)
imshow(capa_holes_int)

capa_perimetro = padarray(capa_perimetro, [espesor_marco,
espesor_marco]*5, 0, 'both')
holes_exterior=capa_perimetro-capa_holes_int
holes_exterior = holes_exterior(espesor_marco*6:end-
espesor_marco*5, espesor_marco*6:end-espesor_marco*5);

Contourholes_exterior = bwconncomp(holes_exterior, 8);
indc = cellfun(@(idx) ind2sub(size(holes_exterior), idx),
Contourholes_exterior.PixelIdxList, 'UniformOutput', false);

%contourholesexterior = cell(size(indcx));

for i = 1:numel(indc)
[indcx,indcy]=ind2sub(size(imagen_recortada),
Contourholes_exterior.PixelIdxList{i})
contourholesexterior{i}
=bwtraceboundary(imagen_recortada,[indcx(10),indcy(10)],'SE',8,Inf,
'clockwise')
contourholesextpixel{i} =
sub2ind(size(imagen_recortada),contourholesexterior{i}(:,1)
,contourholesexterior{i}(:,2))
end

%figure(8)
```

```
%plot(contourholesexterior(:,1),contourholesexterior(:,2),'r','Line
Width',2)


figure(7)
imshow(holes_exterior)
ppo=3
cont=0

yyy=1
%ALGORITMO DE BUSQUEDA PROFUNDA
if yyy==0
for k=1:length(NNn.PixelIdxList')
% Obtener los índices de los píxeles del perímetro
cont=cont+1;
indices_perimetro{k} = NNn.PixelIdxList{k};

% Obtener las coordenadas (x,y) de los píxeles del perímetro
[coord_x, coord_y] = ind2sub(size(imagen_etiquetada),
indices_perimetro{k});

% Obtener el centroide de los píxeles del perímetro
centroide_x = mean(coord_x);
centroide_y = mean(coord_y);

% Obtener los ángulos de los píxeles con respecto al centroide
angulos = atan2(coord_y - centroide_y, coord_x - centroide_x);

% Ordenar los ángulos en sentido antihorario
[~, orden] = sort(angulos, 'descend');

% Convertir los ángulos a coordenadas cartesianas y obtener los
índices lineales
coord_x_ordenado = round(coord_x(orden));
coord_y_ordenado = round(coord_y(orden));
indices_perimetro_ordenado = sub2ind(size(imagen_etiquetada),
coord_x_ordenado, coord_y_ordenado);
NNn.PixelIdxList{k}=indices_perimetro_ordenado;
% Sobreescribir los datos del vector de perímetro en el orden
antihorario
%perimetro_ordenado = perimetro;
%perimetro_ordenado(indices_perimetro{k}) = 0;
%perimetro_ordenado(indices_perimetro_ordenado) = 1;

end
end
%FIN ALGORITMO BUSQUEDA PROFUNDA
```

```
%nn=perimetro;

%NNn = bwconncomp(nn,8);

%grain = false(size(nn));
%grain(NNn.PixelIdxList{2}) = true;


%ALGORITMO DE BUSQUEDA PROFUNDA Rapida
figure(100)
hold on
imagen_binaria = zeros(size(imagen_etiquetada));
imagen_binaria(perimeterpixel) = 1;
imshow(imagen_binaria);
figure(10)

if yyy==1
for k=1:length(NNn.PixelIdxList')
% Obtener los índices de los píxeles del perímetro
clear mapa end_point
mapa=zeros(size(BW));
cont=cont+1;
indices_perimetro{k} = NNn.PixelIdxList{k};
mapa(indices_perimetro{k})=1;
% Obtener las coordenadas (x,y) de los píxeles del perímetro
[coord_x, coord_y] = ind2sub(size(BW), indices_perimetro{k});

%ALGORITMO DE BUSQUEDA DE PUNTOS MAS PROXIMOS
%Este algoritmo de ordenacion por distancia presenta fallas para
elementos
%de curvatura acentuada
[start_point, end_point] = find_start_and_end_points(coord_x',
coord_y');
reference_point=[coord_x(1) coord_y(1)];
[Vectorfinal] = deep_search_perimeter(mapa,reference_point);
%Vectorfinal = deep_search(caminos_new,[end_point]);

% Definir la matriz del mapa y el punto de inicio
%mapa = [0 0 0 0 0 0;
  %      0 1 1 1 1 0;
  %      0 1 0 0 0 0;
  %      0 1 0 0 1 0;
  %      0 0 1 5 1 0;
  %      0 0 0 0 0 0];
%inicio = [2, 2];

%sorted_points=sort_points_by_distance(coord_x', coord_y',
[end_point])
```

```
Vectorfinal(end,:)=[];
coord_x_ordenado=Vectorfinal(:,1);
coord_y_ordenado=Vectorfinal(:,2);

%plot(coord_x_ordenado,coord_y_ordenado)
%text(mean(coord_x_ordenado),mean(coord_y_ordenado),['\leftarrow k=
' num2str(k)])
indices_perimetro_ordenado = sub2ind(size(BW), coord_x_ordenado,
coord_y_ordenado);
NNn.PixelIdxList{k}=indices_perimetro_ordenado;
ZZZZ{k}=indices_perimetro_ordenado;
% Sobreescribir los datos del vector de perímetro en el orden
antihorario
%perimetro_ordenado = perimetro;
%perimetro_ordenado(indices_perimetro{k}) = 0;
%perimetro_ordenado(indices_perimetro_ordenado) = 1;

end
end
%hold off

%figure(29)
NNn.PixelIdxList=[{perimeterpixel}, NNn.PixelIdxList,
contourholesextpixel]


%labeled = labelmatrix(NNn);
%whos labeled
%RGB_label = label2rgb(labeled,'jet','k','shuffle');
%imshow(RGB_label)
%imshow(grain)
%NNn=bwmorph(nn, 'thin', Inf);
ppo=50
%C{2,2} = []
%C(2,:) = []
cor=[]
cont=0;
Data_XYpoint_curv=[];
figure(80)
hold on
for k=1:length(NNn.PixelIdxList')
    if length(NNn.PixelIdxList{k}')>ppo
        cont=cont+1
        segm_vector{cont} = NNn.PixelIdxList{k}';
        %cix = mod(k,length(colors))+1;
        %plot(boundary(:,2), boundary(:,1),...
        %colors(cix),'LineWidth',2);
        Orden_length(cont)=length(NNn.PixelIdxList{k}');
```

```
       %AGREGO COMANDO POLIFIT
ind = [segm_vector{cont}];
zzz = NNn.ImageSize;
[row,col] = ind2sub(zzz,ind);

x = double(row);
y = double(col);


 %AGREGO COMANDO CONVEXHULL
        ind = [segm_vector{cont}];
        zzz = NNn.ImageSize;
        [row,col] = ind2sub(zzz,ind);
        %K = convhull(row,col);
        %K = boundary(row',col',0.002);
        %CALCULO SPLINE

        %xs = row(K)
        %ys = col(K)

        %PLOT SPLINE
        %kh=plot(xs,ys,'k','linewidth',2.5);

i_holes=[length(NNn.PixelIdxList')-
numel(indc)+1:length(NNn.PixelIdxList')]


     %xy_patch{cont} = [xs' ys']

 if ismember(k,[1,[i_holes]])
     w=370*3^3
     threshold = 0.005;
    num_puntos_arco=10;
    f_escala=1.0;
    r=7*3 %Fillet radii for each point
 else
    w=200*3^3*1.5
    threshold = 0.018;
    num_puntos_arco=10;
    f_escala=1.005;
    r=5*3 %Fillet radii for each point
end
%yy2 = smooth(xs,ys,0.9,'rloess')
%kh=plot(xs,yy2)
%regresión lineal
%w = 90;
%x_smooth = smoothdata(x, 'movmean', w);
```

```
%y_smooth = smoothdata(y, 'movmean', w);

%z = smoothn({x,y},w,'robust');
%kh=plot(x,y,'k','linewidth',1.5)


x_corner=[x x(1:round(length(x)*0.11))]
y_corner=[y y(1:round(length(y)*0.11))]
z = smoothn({x_corner,y_corner},w,'robust');
x=z{1};
y=z{2};
xxx=[]
yyy=[]
xxx=z{1};
yyy=z{2};
%kh=plot(x,y,'k','linewidth',1.5)

%plot(x,f,'-','LineWidth',4)
%text(x,f,{R_ajust})
%legend('data','linear fit')

% Cálculo de la curvatura en cada punto
dx = diff(x);
dy = diff(y);
d2x = diff(dx);
d2y = diff(dy);
curvature = abs(d2x.*dy(1:end-1) - dx(1:end-1).*d2y) ./ ((dx(1:end-
1).^2 + dy(1:end-1).^2).^(3/2));

% Filtrado para detectar los puntos de mayor curvatura
%threshold = 0.035; % Umbral para considerar un punto como de mayor
curvatura
[~, locs] = findpeaks(curvature,'MinPeakHeight', threshold);

% Gráfica de los resultados
%plot(x, y)
cornerpoints{cont}=[x(locs)', y(locs)']
%cornerpoints_x{cont}=[x(locs)']
%cornerpoints_y{cont}=[y(locs)']
 % Verificar si el vector en cornerpoints{cont} está vacío
    if isempty(cornerpoints{cont})
        % Si está vacío, realizar la acción apropiada (en este
caso, omitirlo)
        continue;
    end

ratio=7
y_points=y(locs)';
```

```
x_points=x(locs)';

[centroids] = point_sorted_ratio(x_points,y_points,ratio)
x_points=centroids(:,1);
y_points=centroids(:,2);



%ENCUENTRO LOS PUNTOS ORIGINALES MAS
CERCANOS::::::::::::::::::::::::::::
Perimetro=[];
perimetro=[double(row') double(col')]
puntosCurvatura = [x_points y_points];

puntosCercanos = zeros(size(puntosCurvatura));

% Bucle para cada punto de curvatura
for i = 1:size(puntosCurvatura, 1)
    % Extraer las coordenadas del punto de curvatura actual
    puntoCurvaturaActual = puntosCurvatura(i, :);

    % Calcular la distancia a todos los puntos en el perímetro
    distancias = sqrt(sum((perimetro -
puntoCurvaturaActual*1.02).^2, 2));

    % Encontrar el índice del punto en el perímetro con la
distancia mínima
    indicePuntoCercano = find(distancias == min(distancias), 1);

    % Almacenar el punto más cercano en el vector
    puntosCercanos(i, :) = perimetro(indicePuntoCercano, :);
end
%x_points=puntosCercanos(:,1);
%y_points=puntosCercanos(:,2);
%%%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::::
puntos = [puntosCurvatura];

% Definir el punto de referencia
punto_referencia = mean(puntosCercanos);  % Puedes ajustar las
coordenadas según sea necesario

% Factor de escala
factor_escala = f_escala;

% Calcular la diferencia entre cada punto y el punto de referencia
diferencia = puntos - punto_referencia;
```

```
% Escalar los puntos con respecto al punto de referencia
puntos_escalados = punto_referencia + factor_escala * diferencia;


%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
x_points=puntos_escalados(:,1);
y_points=puntos_escalados(:,2);



search{cont}=centroids
center=mean(centroids)
          num = cont; % Número que deseas mostrar junto al punto
(cambiar según necesidad)
offset = 0.2; % Ajuste para la posición del texto cerca del punto
%text(center(:, 1) + offset, center(:, 2) + offset, num2str(num),
'FontSize', 12, 'FontWeight', 'bold');

%radio=12;
%[clustersCentroids,clustersGeoMedians,clustersXY]=clusterXYpoints(
[x_points y_points],radio)
%x_points=[clustersGeoMedians(:,1) ];
%y_points=[clustersGeoMedians(:,2) ];
densidadbordes=0.7
if k==1
    num_puntos_arco=10;
else
    num_puntos_arco=10;
end
if numel(x_points)>2
%num_puntos_arco=10;
radius = 5;
%[ARCOENX,ARCOENY] =
RoundedCornerPolygon(x_points,y_points,radius,num_puntos_arco,densi
dadbordes)
V_pointsXY=[x_points y_points]; %no va cerrada la curva, son
%todos los puntos sin repetirse,un cuadrado serian 4 puntos, no 5
%r=5;
np=10; %Number of points used to construct each fillet edge
f_poblacion=1;
[T2]=Reconstruccion_por_chaflan(V_pointsXY,r,np,f_poblacion)
ARCOENX=T2(:,1)
ARCOENY=T2(:,2)
else
    ARCOENX=[];
    ARCOENY=[];
end
hold on
```

```
plot(ARCOENX,ARCOENY, 'b','LineWidth',1);
plot(x,y, 'k','LineWidth',1);

if k==1
  x_hole{cont}=xxx
   y_hole{cont}=yyy
else
x_hole{cont}=ARCOENX';
y_hole{cont}=ARCOENY';
end

cor=[cor;[x(locs)', y(locs)']]
%scatter(x(locs), y(locs), 'r', 'filled')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
scatter(x_points,y_points, 'r', 'filled')
%plot(x(locs), y(locs), 'k')
VectorXY=[double(row)' double(col)']
%XYpoint_curv=[x(locs); y(locs)]'

rd = 2; % valor de r
ind = locs; % vector de índices
N = length(ind); % longitud de ind
ind_buscados = reshape(repelem(ind,3),N,3) %+ repelem([-rd 0
rd],N,1); % generar los índices buscados
ind_buscados = ind_buscados(:)'+repmat([-rd 0 rd], 1, N);
ind_buscados([2:3:end])=[];

%_____________________________________
ind_buscados(ind_buscados<rd)=rd
%_____________________________________

XYpoint_curv=[x(ind_buscados); y(ind_buscados)]'
%CLUSTER PUNTOS CURVATURE
radio=12;
[clustersCentroids,clustersGeoMedians,clustersXY]=clusterXYpoints([
XYpoint_curv(:,1) XYpoint_curv(:,2)],radio)


%CREACION DE HOLES BSPLINE
x=[XYpoint_curv(:,1) ];
y=[XYpoint_curv(:,2) ];


%x=[clustersGeoMedians(:,1) ];
%y=[clustersGeoMedians(:,2) ];

xf=x(1:2)
```

```
yf=y(1:2)
x=[x;xf]
y=[y;yf]

%y=[p.Vertices(:,2) ;[p.Vertices(1:2,2)]'];
points = [x y].';
if k==0
values = spcrv(points,3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%x_hole{cont}=[]
%y_hole{cont}=[]
%x_hole{cont}=values(1,:)';
%y_hole{cont}=values(2,:)';  (opcional)
else
end
%plot(points(1,:),points(2,:),'k');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%plot(values(1,:),values(2,:),'r','LineWidth',0.5);
legend({'Control Polygon' 'Quadratic Spline Curve'},
'location','SE');
daspect([1 1 1])

%ALMACENO LOS PUNTOS DE CURVATURA
Data_XYpoint_curv=[XYpoint_curv; Data_XYpoint_curv];
%AGREGO LOS HOLES RESPECTIVOS
[nn] = post_detect_curvature(XYpoint_curv,VectorXY,nn)



%ITEM OPCIONAL
% Suavizado de curva con csaps
%pp = csaps(x,y);

% Graficar curva original y suavizada
%xx = linspace(x(1),x(end-1),100);
%yy = fnval(pp,xx);
%plot(x,y,'o',xx,yy);
%kh=plot(xx,yy,'k','linewidth',0.5)


%fg=contourinfoplot(xs, ys, 0, 1)


%coeficienteslineales = polyfit(x, y, 1);          % Coeficientes
%yfit = polyval(coeficienteslineales, x);          % línea de
regresión estimada
```

```
%SStot = sum((y-mean(y)).^2);                    % Suma total de
cuadrados
%SSres = sum((y-yfit).^2);                       % Residual Suma de
Cuadrados
%Rsq = 1-SSres/SStot;                            %R^2



%f = polyval(coeficienteslineales,x);
%R_ajust=double(Rsq);

%plot(x,f,'-','LineWidth',4)
%text(x,f,{R_ajust})
legend('data','linear fit')

    else
    end
end
hold off
daspect([1 1 1])
camroll(-90)

% Eliminar vectores vacíos
x_hole = x_hole(~cellfun(@isempty, x_hole));
y_hole = y_hole(~cellfun(@isempty, y_hole));
% Iterar a través de cada celda en NNn.PixelIdxList


%x2 e y2 =agujeros
% Coordenadas del polígono
y = y_hole{1}'
%y_2= [y_hole{2:6}]
x = x_hole{1}'
%x_2= [x_hole{2:6}]
% Número de puntos en la grilla
num_points_x =65;
num_points_y = 200;

% Generar grilla de puntos dentro del rango del polígono
[x_grid, y_grid] = meshgrid(linspace(min(x), max(x), num_points_x),
...
                            linspace(min(y), max(y),
num_points_y));

% Convertir la grilla en una lista de puntos
points_to_check = [x_grid(:), y_grid(:)];

% Lista para almacenar los puntos dentro del polígono
```

```
points_inside_polygon = [];

% Verificar si los puntos de la grilla están dentro del polígono
for i = 1:size(points_to_check, 1)
    if inpolygon(points_to_check(i, 1), points_to_check(i, 2), x,
y)
        points_inside_polygon = [points_inside_polygon;
points_to_check(i, :)];
    end

end
figure(81)
xg=points_inside_polygon(:,1);
xg
xg=[xg;x]
yg=points_inside_polygon(:,2);
yg=[yg;y]

for i=2:length(x_hole)
    x2=[];
    y2=[];
    x2=double(x_hole{i}');
    y2=double(y_hole{i}');
[in] = inpolygon(xg,yg,x2,y2);
xg=[xg(~in);x2]
yg=[yg(~in);y2]
end

zg = ones(numel(xg),1)*15;
xg = repmat(xg,5,1);
yg = repmat(yg,5,1);
zg = zg*(0:.25:1);
zg = zg(:);
shp = alphaShape(xg,yg,zg);
% Crear alphaShape
shp.Alpha = 3.5*3;



[elements,nodes] = boundaryFacets(shp);
%Put the data in the correct shape for geometryFromMesh.

nodes = nodes';
elements = elements';

%nodes(1,:) = nodes(1,:).*[1/1000]; nodes(2,:) =
nodes(2,:).*[1/1000]; nodes(3,:) = nodes(3,:).*[1/1000]
model = createpde();
```

```
g=geometryFromMesh(model,nodes,elements);
%View the geometry and face numbers.
msh = generateMesh(model);
pdemesh(model)
pdegplot(model,"FaceLabels","on","FaceAlpha",0.95)
foco1 = light('Position', [50, 50, 50], 'Style', 'local', 'Color',
'white');
foco2 = light('Position', [-50, -50, -50], 'Style', 'local',
'Color', 'white');
foco3 = light('Position', [0, 0, 50], 'Style', 'local', 'Color',
'white'); % Nuevo foco

% Ajustar propiedades de la luz
lighting phong; % Cambiar a Gouraud para suavizar la iluminación

% Configurar propiedades de la esfera grande

axis off;

% Configurar vista y posición de la cámara
view(3);
campos([50, 50, 50]);

%Obtain a surface mesh of the alphaShape object.
F = ([90 218 10]);
VertexIDs_force = addVertex(g,"Coordinates",F)

%Agrego puntos , lados o caras de reestricciones
R = ([13.43 8.843 10;22.84 8.843 10;145.2 8.843 10;154.6 8.843
10;13.43 8.843 0;22.84 8.843 0;145.2 8.843 0;154.6 8.843 0]);
VertexIDs_restraint = addVertex(g,"Coordinates",R)

pdegplot(g, "VertexLabels", "on","FaceAlpha",0.5)
% Graficar el polígono y los puntos agregados
figure(2);
plot(x, y, 'b-', points_inside_polygon(:, 1),
points_inside_polygon(:, 2), 'ro');
axis equal;
title('Polígono y Puntos de la Grilla Agregados');
legend('Polígono', 'Puntos Agregados');


structuralModel = createpde("structural","static-solid");

%Create and plot the geometry.
gm = g;
structuralModel.Geometry = gm;
pdegplot(structuralModel,"FaceAlpha",1)
```

```
%Definición de propiedades mecanicas
structuralProperties(structuralModel,"YoungsModulus",2E3, ...
                                     "PoissonsRatio",0.3, ...
                                     "MassDensity",2.7E-6);
%Agrego peso propio?
structuralBodyLoad(structuralModel, ...
    "GravitationalAcceleration",[0;0;0])

% Obtener los nodos de la malla
%nodes = structuralModel.Geometry.Nodes;

% Definir los vértices en los que deseas aplicar fuerzas y
restricciones
force_vertex_ids = VertexIDs_force;  % IDs de los vértices donde
aplicarás fuerzas

restraint_vertex_ids = VertexIDs_restraint;  % IDs de los vértices
con restricciones

% Aplicar fuerzas en los vértices seleccionados
for i = 1:length(force_vertex_ids)

structuralBoundaryLoad(structuralModel,"Vertex",force_vertex_ids(i)
, ...
                                        "Force",[100; 100; 100]);
   % Fuerza en dirección z
end

% Aplicar restricciones en los vértices seleccionados
for i = 1:length(restraint_vertex_ids)
    %structuralBC(structuralmodel,"Face",[5
10],"Constraint","fixed");
    structuralBC(structuralModel, "Vertex",
restraint_vertex_ids(i), ...
        "Constraint", "fixed");
     % Restricción de movimiento en todos los grados de libertad
end
% Generar la malla de elementos finitos para el modelo estructural
structuralMesh = generateMesh(structuralModel);
% Resolver el modelo
results = solve(structuralModel);

% Visualizar la deformación resultante
figure;
pdeplot3D(results.Mesh, ...
          'ColorMapData',results.VonMisesStress, ...
          'Deformation', 'on',...
```

```
          'DeformationScaleFactor',10000)
%pdeplot3D(structuralModel, results.Displacement, ...
%          'Deformation', 'on', 'DeformationScaleFactor', 100);
title('Deformación resultante');


%yt=xy_patch{1,:}(:,2)
%figure, clf, hold on
%patch(xy_patch{1,:}(1,:),xy_patch{1,:}(2,:), 'r', 'linestyle',
'none')

% Crear el structure de malla
fv.vertices = nodes';
fv.faces = elements';
%cd 'C:\Users\felip\Documents\MEMORIA FELIPE'
patch2stl('testhole5circle.stl',fv)        % Save to binary .stl
```

ANEXO 4: Rutina MATLAB® para interpretación de imagen ráster por celosía estructural.

```
clc
clear all
close all
%Estos parametros modelan la trasformación que deseamos obtener
%_____________________________________________________________
ppo=5; %este debe ser mayor a 5 para que no falle el findpeak de
curvature
alt=8.5
%Inicializamos la imagen
I = rgb2gray(imread('Raster OT.jpg'))
I = imresize(I, 1);
%I=imrotate(I,90)
% Cargar la imagen original (reemplaza 'imagen_original.jpg' con la
ruta de tu imagen)
imagen_original = I;

% Definir las dimensiones deseadas en milímetros
tamx_mm = 48.4; % Ancho en milímetros
tamy_mm = 50.0; % Alto en milímetros

% Definir el número de elementos en X e Y (nelx y nely)
nelx = 1240;
nely = 150;

% Calcular las dimensiones finales en píxeles
ancho_final_px = 1200; % Convertir de mm a píxeles
alto_final_px = 150; % Convertir de mm a píxeles

% Redimensionar la imagen a las dimensiones deseadas
%I = imresize(imagen_original, [alto_final_px, ancho_final_px]);


% Convertir la imagen a escala de grises


% Realizar una operación de suavizado
img_smooth = imgaussfilt(I,1.69);

% Aplicar la transformada de distancia
img_dist = bwdist(~imbinarize(img_smooth));

% Aplicar la umbralización
I = imbinarize(img_dist, 0.043*max(img_dist(:)));
```

```
%procedo a binarización de esta
BW=(I)
%esta propiedad me permite invertir el color
BW_false=(BW-1)/-1;

%********ESTA SECCIOM DECODIGO NO LA USO*******
%think_bw=bwmorph(BW_false,'thin', inf);
%figure(99)
%imshow(think_bw)

%Llamamos al mapa de distancias euclideanas
D_euclidean = bwdist(BW,'euclidean');
BWJ = imcomplement(BW);

%Usamos el detector de esquinas , el cual no es util en la
esquelitización
corners = detectHarrisFeatures(I);
figure(2)
imshow(I); hold on;
Strongest=corners.selectStrongest(400)
%plot(imclearborder(I));
ppp=Strongest.Location

%_____________________________________________________________________
im=BW
figure(12)
imd=bwdist(BW);
subplot(2,1,1)
imshow(imd,[]);
title('Euclidean Distance Transform');

%imw=watershed(imd);
%imw=imw==0;
log=del2(imd);
subplot(2,1,1)
imshow(log,[]);
title('Laplacian of Gaussian');
imp=bwperim(im);
se=strel('disk',1);
imp=imdilate(imp,se);

bin=log < -0.2;
mask=bin-imp;
mask=mask>0;
subplot(2,2,3)
imshow(mask,[]);
title('Binazired mask');
```

```
rp = regionprops(mask, 'PixelIdxList', 'Area');
rp = rp(vertcat(rp.Area) > 30);

centerline=zeros(size(im));
centerline(vertcat(rp.PixelIdxList))=1;
centerline=bwmorph(centerline,'thin');
%subplot(2,2,3)
%imshow(centerline,[]);
%title('Final mask');


ultimateErosion = bwulterode(imcomplement(BW));
figure(2), imshow(ultimateErosion)



L = watershed(D_euclidean);
L(~BW) = 0;
rgb = label2rgb(L,'jet',[.5 .5 .5]);
figure(9)
imshow(rgb)
title('Watershed Transform')

% Obtener las coordenadas de los pixeles negros
[y_negro, x_negro] = find(BW == 0);

% Crear la matriz de nube de puntos
nube_de_puntos = [x_negro, y_negro];
%ELIMINAR:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:

figure(11)
imcontour(L,20)

figure(4)
%out2 = bwskel(BW,'MinBranchLength',15);
%BMS=imadjust(I,[0 1],[1 0])
out2 = bwmorph(D_euclidean,'skel',Inf);
imshow(imcomplement(out2))

%algoritmo de poda

% Imagen leída
img = out2;

b_img_skel = bwmorph (img, 'skel', Inf);
b_img_spur1 = bwmorph(b_img_skel, 'spur', 50);
%b_img_spur = bwmorph(b_img_spur1, 'bridge');
```

```
b_img_spur1   = bwmorph(b_img_spur1, 'thin', 'inf');
BWedge = edge(b_img_spur1,'Canny');
figure(15)
%figure('Name', 'Pruning');
subplot(1,2,1);
imshow(b_img_skel);
title(sprintf('Image Skeleton'));
subplot(1,2,2);
imshow(b_img_spur1);
title(sprintf('Skeleton Image Pruned'));
%out2=b_img_spur1;
out2=b_img_spur1;

figure(16)
imshow(BWedge)

%detectar puntos de bifurcacion
skelImg   = bwmorph(out2, 'thin', 'inf');
branchImg = bwmorph(skelImg, 'branchpoints');
endImg    = bwmorph(skelImg, 'endpoints');

[row, column] = find(endImg);
endPts        = [row column];
[row, column] = find(branchImg);
branchPts     = [row column];
branchPts     = [branchPts;endPts];%%%%%%%%%%%%%%%%%%%%%remover


figure(18); imshow(imcomplement(skelImg)); hold on;
plot(branchPts(:,2),branchPts(:,1),'r*');
hold on; plot(endPts(:,2),endPts(:,1),'*');


% Leer la imagen esqueletizada
img = skelImg;

% Binarizar la imagen si no está binarizada
if numel(size(img)) > 2
    img = rgb2gray(img);
end
img = img > 0;

% Etiquetar los objetos en la imagen
[L, num] = bwlabel(img);

% Buscar los puntos de bifurcación en la imagen etiquetada
stats = regionprops(L, 'PixelList');
bifurcations = [];
```

```
for i = 1:num
    if length(stats(i).PixelList) == 3
        bifurcations = [bifurcations; stats(i).PixelList(2,:)];
    end
end

% Erosionar las ramas que salen de los puntos de bifurcación y que
son menores a 10 pixeles
se = strel('disk', 1);
for i = 1:size(bifurcations, 1)
    rama = bwselect(L, bifurcations(i,1), bifurcations(i,2), 8);
    if sum(rama(:)) < 10
        img(rama) = imerode(img(rama), se);
    end
end


%while eliminar_ramificaciones
    % Aplicar la función bwmorph con el parámetro 'spur' para
eliminar las ramificaciones de un solo píxel
    %imagen_sin_ramificaciones = bwmorph(img, 'spur', 100);
     imagen_sin_ramificaciones = out2;
figure(77)
imshow(imcomplement(imagen_sin_ramificaciones))



% Encontrar los píxeles de bifurcación en la imagen esqueletizada
im_bif = bwmorph(imagen_sin_ramificaciones, 'branchpoints');

% Mostrar la imagen con los puntos de bifurcación resaltados en
rojo

hold on
[Bx, By] = find(im_bif);
figure(82)
plot(By, Bx, 'r.', 'MarkerSize', 10)
hold off

clear branchImg branchPts nn
branchImg = bwmorph(imagen_sin_ramificaciones, 'branchpoints');
[row, column] = find(branchImg);
branchPts     = [row column];


figure(45)
nn=imagen_sin_ramificaciones;
```

```
linearInd1 = sub2ind(size(nn),branchPts(:,1)',branchPts(:,2)')
nn(linearInd1) = 0;
linearInd2 = sub2ind(size(nn),branchPts(:,1)',branchPts(:,2)'+2)
nn(linearInd2) = 0;
linearInd3 = sub2ind(size(nn),branchPts(:,1)',branchPts(:,2)'-2)
nn(linearInd3) = 0;
linearInd4 = sub2ind(size(nn),branchPts(:,1)',branchPts(:,2)'+1)
nn(linearInd4) = 0;
linearInd5 = sub2ind(size(nn),branchPts(:,1)',branchPts(:,2)'-1)
nn(linearInd5) = 0;

linearInd6 = sub2ind(size(nn),branchPts(:,1)'+1,branchPts(:,2)')
nn(linearInd6) = 0;
linearInd7 = sub2ind(size(nn),branchPts(:,1)'+1,branchPts(:,2)'+2)
nn(linearInd7) = 0;
linearInd8 = sub2ind(size(nn),branchPts(:,1)'+1,branchPts(:,2)'-2)
nn(linearInd8) = 0;
linearInd9 = sub2ind(size(nn),branchPts(:,1)'+1,branchPts(:,2)'+1)
nn(linearInd9) = 0;
linearInd10 = sub2ind(size(nn),branchPts(:,1)'+1,branchPts(:,2)'-1)
nn(linearInd10) = 0

linearInd11 = sub2ind(size(nn),branchPts(:,1)'-1,branchPts(:,2)')
nn(linearInd11) = 0;
linearInd12 = sub2ind(size(nn),branchPts(:,1)'-1,branchPts(:,2)'+2)
nn(linearInd12) = 0;
linearInd13 = sub2ind(size(nn),branchPts(:,1)'-1,branchPts(:,2)'-2)
nn(linearInd13)= 0;
linearInd14 = sub2ind(size(nn),branchPts(:,1)'-1,branchPts(:,2)'+1)
nn(linearInd14) = 0;
linearInd15 = sub2ind(size(nn),branchPts(:,1)'-1,branchPts(:,2)'-1)
nn(linearInd15) = 0

linearInd16 = sub2ind(size(nn),branchPts(:,1)'-2,branchPts(:,2)')
nn(linearInd16) = 0;
linearInd17 = sub2ind(size(nn),branchPts(:,1)'-2,branchPts(:,2)'+2)
nn(linearInd17) = 0;
linearInd18 = sub2ind(size(nn),branchPts(:,1)'-2,branchPts(:,2)'-2)
nn(linearInd18) = 0;
linearInd19 = sub2ind(size(nn),branchPts(:,1)'-2,branchPts(:,2)'+1)
nn(linearInd19) = 0;
linearInd20 = sub2ind(size(nn),branchPts(:,1)'-2,branchPts(:,2)'-1)
nn(linearInd20) = 0

linearInd21 = sub2ind(size(nn),branchPts(:,1)'+2,branchPts(:,2)')
nn(linearInd21) = 0;
linearInd22 = sub2ind(size(nn),branchPts(:,1)'+2,branchPts(:,2)'+2)
nn(linearInd22) = 0;
```

```
linearInd23 = sub2ind(size(nn),branchPts(:,1)'+2,branchPts(:,2)'-2)
nn(linearInd23) = 0;
linearInd24 = sub2ind(size(nn),branchPts(:,1)'+2,branchPts(:,2)'+1)
nn(linearInd24) = 0;
linearInd25 = sub2ind(size(nn),branchPts(:,1)'+2,branchPts(:,2)'-1)
nn(linearInd25) = 0


%ahora eliminamos radio influencia de endpoints
linearInd1 = sub2ind(size(nn),endPts(:,1)',endPts(:,2)')


NNn = bwconncomp(nn,8)

%grain = false(size(nn));
%grain(NNn.PixelIdxList{2}) = true;
figure(29)

labeled = labelmatrix(NNn);
whos labeled
RGB_label = label2rgb(labeled,'jet','k','shuffle');
imshow(RGB_label)
%imshow(grain)
%NNn=bwmorph(nn, 'thin', Inf);
figure(78)
cont=0;
hold on

%ALGORITMO DE BUSQUEDA PROFUNDA
for k=1:length(NNn.PixelIdxList')
% Obtener los índices de los píxeles del perímetro
clear mapa end_point
mapa=zeros(size(BW));
cont=cont+1
indices_perimetro{k} = NNn.PixelIdxList{k};
mapa(indices_perimetro{k})=1;
% Obtener las coordenadas (x,y) de los píxeles del perímetro
[coord_x, coord_y] = ind2sub(size(BW), indices_perimetro{k});


%ALGORITMO DE BUSQUEDA DE PUNTOS MAS PROXIMOS
%Este algoritmo de ordenacion por distancia presenta fallas para
elementos
%de curvatura acentuada
[start_point, end_point] = find_start_and_end_points(coord_x',
coord_y');
reference_point=[start_point];
Vectorfinal = deep_search(mapa,[end_point])
```

```
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here


% Definir la matriz del mapa y el punto de inicio
%mapa = [0 0 0 0 0 0;
  %      0 1 1 1 1 0;
  %      0 1 0 0 0 0;
  %      0 1 0 0 1 0;
  %      0 0 1 5 1 0;
  %      0 0 0 0 0 0];
%inicio = [2, 2];

%sorted_points=sort_points_by_distance(coord_x', coord_y',
[end_point])
coord_x_ordenado=Vectorfinal(:,1);
coord_y_ordenado=Vectorfinal(:,2);



plot(coord_x_ordenado,coord_y_ordenado)
text(mean(coord_x_ordenado),mean(coord_y_ordenado),['\leftarrow k=
' num2str(k)])
indices_perimetro_ordenado = sub2ind(size(BW), coord_x_ordenado,
coord_y_ordenado);
NNn.PixelIdxList{k}=indices_perimetro_ordenado;
% Sobreescribir los datos del vector de perímetro en el orden
antihorario
%perimetro_ordenado = perimetro;
%perimetro_ordenado(indices_perimetro{k}) = 0;
%perimetro_ordenado(indices_perimetro_ordenado) = 1;

end
hold off

figure(101)
cont=0
Data_XYpoint_curv=[];
for k=1:length(NNn.PixelIdxList')
    if length(NNn.PixelIdxList{k}')>ppo
        cont=cont+1
  segm_vector{cont} = NNn.PixelIdxList{k}';
  %cix = mod(k,length(colors))+1;
  %plot(boundary(:,2), boundary(:,1),...
```

```
        %colors(cix),'LineWidth',2);
        Orden_length(cont)=length(NNn.PixelIdxList{k}');

        %AGREGO COMANDO POLIFIT
        ind = [segm_vector{cont}];
zzz = NNn.ImageSize;
[row,col] = ind2sub(zzz,ind)


xr = double(row);
yr = double(col);

x = double(row);
y = double(col);


%FILTRO DE DISTANCIAS PONDERADAS
% Definir el vector de puntos donde se evaluará la curva suavizada
%X_eval = linspace(x(1), x(numel(x)), numel(x));

% Ajustar una curva suave a los puntos dados mediante splines
%Y_eval = csaps(x, y, 0.05, X_eval);

%x=X_eval ;
%y=Y_eval ;


w=199
z = smoothn({x,y},w,'robust');

%plot(x,f,'-','LineWidth',4)
%text(x,f,{R_ajust})
%legend('data','linear fit')

x=z{1};
y=z{2};

% Cálculo de la curvatura en cada punto
dx = diff(x);
dy = diff(y);
d2x = diff(dx);
d2y = diff(dy);
curvature = abs(d2x.*dy(1:end-1) - dx(1:end-1).*d2y) ./ ((dx(1:end-
1).^2 + dy(1:end-1).^2).^(3/2));

% Filtrado para detectar los puntos de mayor curvatura
threshold = 0.085; % Umbral para considerar un punto como de mayor
curvatura
```

```
[~, locs] = findpeaks(curvature, 'MinPeakHeight', threshold);

% Gráfica de los resultados
plot(x, y)
hold on
scatter(x(locs), y(locs), 'r', 'filled')

VectorXY=[double(row)' double(col)']
%XYpoint_curv=[x(locs), y(locs)]
XYpoint_curv=[x(locs); y(locs)]'

%ALMACENO LOS PUNTOS DE CURVATURA
Data_XYpoint_curv=[XYpoint_curv; Data_XYpoint_curv];
%AGREGO LOS HOLES RESPECTIVOS
[nn] = post_detect_curvature(XYpoint_curv,VectorXY,nn)

    end
end

%AGREGO LOS POINT CURVATURE DENTRO DE LOS BranchPoints
branchPts=[branchPts;Data_XYpoint_curv];

ppo=ppo;
branchPts=[branchPts;Data_XYpoint_curv];
radio = ppo+alt;
[clustersCentroids,clustersGeoMedians,clustersXY]=clusterXYpoints([
branchPts(:,1) branchPts(:,2)],radio)
branchPts=clustersCentroids;

%ALGORITMO DE CLUSTERIZACION Y SEGMENTACION DE PUNTOS CERCANOS
% Definir el radio de segmentación
NNn = bwconncomp(nn,8)
figure(84)

hold on
for k=1:length(NNn.PixelIdxList')

     %if length(NNn.PixelIdxList{k}')>ppo

clear mapa end_point
mapa=zeros(size(BW));
cont=cont+1
indices_perimetro{k} = NNn.PixelIdxList{k};
mapa(indices_perimetro{k})=1;
% Obtener las coordenadas (x,y) de los píxeles del perímetro
[coord_x, coord_y] = ind2sub(size(BW), indices_perimetro{k});
```

```
[start_point, end_point] = find_start_and_end_points(coord_x',
coord_y');
reference_point=[start_point];
Vectorfinal = deep_search(mapa,[end_point])
%ALGORITMO DE BUSQUEDA DE PUNTOS MAS PROXIMOS
%Este algoritmo de ordenacion por distancia presenta fallas para
elementos
%de curvatura acentuada
[start_point, end_point] = find_start_and_end_points(coord_x',
coord_y');
reference_point=[start_point];
Vectorfinal = deep_search(mapa,[end_point])
coord_x_ordenado=Vectorfinal(:,1);
coord_y_ordenado=Vectorfinal(:,2);



plot(coord_x_ordenado,coord_y_ordenado)
text(mean(coord_x_ordenado),mean(coord_y_ordenado),['\leftarrow k=
' num2str(k)])

eskel_renodes{k}=[coord_x_ordenado(1)
coord_y_ordenado(1);coord_x_ordenado(end) coord_y_ordenado(end)];
% Calcular la distancia euclidiana entre XpYp y todos los
branchpoints
distancias1 = sqrt(sum((branchPts - [coord_x_ordenado(1)
coord_y_ordenado(1)]).^2, 2));
distancias2 = sqrt(sum((branchPts - [coord_x_ordenado(end)
coord_y_ordenado(end)]).^2, 2));
% Encontrar el índice del branchpoint más cercano
indice_del_mas_cercano1 = find(distancias1 == min(distancias1));
indice_del_mas_cercano2 = find(distancias2 == min(distancias2));
% Obtener las coordenadas del branchpoint más cercano
branchpoint_mas_cercano1 = branchPts(indice_del_mas_cercano1, :);

branchpoint_mas_cercano2 = branchPts(indice_del_mas_cercano2, :);
eskel_Branchnodes{k}=[branchpoint_mas_cercano1(1)
branchpoint_mas_cercano1(2);branchpoint_mas_cercano2(1)
branchpoint_mas_cercano2(2)];

v_incidenc_zeros=zeros(1, length(branchPts(:,1)));
v_incidenc_zeros(indice_del_mas_cercano1)=1;
v_incidenc_zeros(indice_del_mas_cercano2)=1;
incidencia_vector{k}=v_incidenc_zeros';

  %if length(NNn.PixelIdxList{k}')>ppo
     cont=cont+1
ind_clear_graph{cont}=k; %aislo la arista problematica
```

```
indices_perimetro_ordenado = sub2ind(size(BW), coord_x_ordenado,
coord_y_ordenado);
NNn.PixelIdxList{k}=indices_perimetro_ordenado;


segm_vector{cont} = NNn.PixelIdxList{k}';
ind = [segm_vector{cont}];
zzz = NNn.ImageSize;
[row,col] = ind2sub(zzz,ind)

x_b = coord_x_ordenado;
y_b = coord_y_ordenado;
Start_End_point=[x_b(1) y_b(1);x_b(end) y_b(end)] %este debe
capturar dentro de un for los elementos de cada vector ordenado



%StartEndMinDistanceBranchPts =
MinDistanceBranchPts(Start_End_point,branchPts)
XY1 = Start_End_point; % Vector de puntos XY1
XY2 = branchPts; % Vector de puntos XY2

% Encontrar el índice de los puntos más cercanos en XY2 a cada
punto de XY1
if numel(XY1)>0
idx = zeros(size(XY1, 1), 1); % Inicializar vector de índices
for i = 1:2
    d = sqrt(sum((XY2-XY1(i,:)).^2, 2)); % Distancias entre
XY1(i,:) y cada punto de XY2

    %d= sqrt((XY2(:,1)-XY1(i,1)).^2 + (XY2(:,2)-XY1(i,2)).^2);
    [~, idx(i)] = min(d); % Índice del punto más cercano
end

% Hacer los puntos más cercanos igual a cero
StartEndMinDistanceBranchPts=XY2(idx, :) ;
end

x_branchpts=[StartEndMinDistanceBranchPts(1,1);x_b;StartEndMinDista
nceBranchPts(end,1)]
y_branchpts=[StartEndMinDistanceBranchPts(1,2);y_b;StartEndMinDista
nceBranchPts(end,2)]
indices_perimetro_ordenado = sub2ind(size(BW), x_branchpts,
y_branchpts);
NNn.PixelIdxList{k}=indices_perimetro_ordenado;
```

```
x=x_branchpts;
y=y_branchpts;
%regresión lineal
w=100
z = smoothn({x,y},w,'robust');
kh=plot(x,y,'k','linewidth',2.5)
     N = 1;            % second-degree polynomial
maxDistance = 5; % maximum allowed distance for a point to be
inlier
almacen_x=z{2};
almacen_y=z{1};
%[P, inlierIdx] =
fitPolynomialRANSAC([almacen_x,almacen_y],N,maxDistance);

%yRecoveredCurve = polyval(P,almacen_x);

%plot(almacen_x,yRecoveredCurve,'-g','LineWidth',3)

[Regresionlineal{k},Sl] = polyfit(almacen_x,almacen_y,1)
regresionalmacen(k,:)=double(Regresionlineal{k});
   rsquarelineal = 1 - Sl.normr^2 / norm(almacen_y -
mean(almacen_y))^2

   y_pred_lineal = polyval(Regresionlineal{k},almacen_x); % valores
predichos


     plot(almacen_x,y_pred_lineal,'-g','LineWidth',3)

%Caracterización del tipo de regresion para el vector
[Id_vector,punto_max_curvatura] = CaractericeVector_LoNL(VectorXY)
%end
end
hold off
NNn_skel=NNn;


NNn = bwconncomp(nn,8)
% Supongamos que NNn.PixelIdxList es una celda de vectores

% Crear un vector lógico que indica qué elementos deben eliminarse
should_remove = cellfun(@(x) numel(x) < ppo, NNn.PixelIdxList);

% Eliminar los elementos que cumplen la condición
NNn.PixelIdxList(should_remove) = [];
% Actualizar NNn.NumObjects
NNn.NumObjects = numel(NNn.PixelIdxList);
```

```
% También puedes eliminar los elementos correspondientes de otras
propiedades, si es necesario
% Por ejemplo, si NNn tiene un campo 'Area', puedes eliminar
elementos correspondientes
% a través de NNn.Area de manera similar
% NNn.Area(should_remove) = [];

% Ahora, NNn.PixelIdxList contendrá solo los vectores con un tamaño
mayor o igual a 50

%grain = false(size(nn));
%grain(NNn.PixelIdxList{2}) = true;
figure(129)

labeled = labelmatrix(NNn);
whos labeled
% Crear el elemento estructurante tipo disco de 2 píxeles de radio
%radio = 2;
%se = strel('disk', radio);

% Aplicar la dilatación a la imagen etiquetada
%labeled = imdilate(labeled, se);
RGB_label = label2rgb(labeled,'jet','k','shuffle');
%imshow(RGB_label)
% Encontrar píxeles negros en la imagen RGB_Label
% Cargar la imagen RGB
imagen = RGB_label; % Reemplaza 'tu_imagen.jpg' con la ruta de tu
imagen

% Definir un umbral para considerar un píxel como completamente
negro
umbral = 3; % Ajusta este valor según tus necesidades

% Crear una máscara para los píxeles negros
mascara_negros = all(imagen <= umbral, 3);

% Reemplazar los píxeles negros con blanco (255, 255, 255)
imagen(repmat(mascara_negros, [1, 1, 3])) = 255;

% Mostrar la imagen resultante
imshow(imagen);
% Mostrar la imagen resultante
imshow(imagen);


figure(50)
I = (labeled);
SE = strel("disk",0)
```

```
dilatedI = imdilate(I,SE);
D = -bwdist(~dilatedI);

Ld = watershed(D);
figure(201)
imshow(label2rgb(Ld))
Ld(BW == 1) = 0;
bw2 =dilatedI ;
bw2(Ld == 0) = 0;
figure(56)
imshow(Ld)

NNn = bwconncomp(Ld,8)
mask = imextendedmin(dilatedI,2);
%imshowpair(bw,mask,'blend')

D2 = imimposemin(D,dilatedI);
Ld2 = watershed(D2);

bw3 = I;
bw3(Ld2 == 0) = 0;
%imshow(bw3)

gmag = imgradient(dilatedI);

%imshow(gmag,[])
title("Gradient Magnitude")
L = watershed(dilatedI);
Lrgb = label2rgb(L);
%imshow(Lrgb)
figure(55)
imshowpair(label2rgb(Ld),bw2,'montage')
title("Watershed Transform of Gradient Magnitude")

NNn_mask=NNn;

IblurX1 = imgaussfilt(label2rgb(Ld),[4 4]);


figure(51)
imshow(IblurX1)
title('Smoothed image, \sigma_x = 4, \sigma_y = 1')


mapa_colores = colormap('jet');
% Generar el vector de celdas de colores RGB
vector_colores = cell(length(NNn.PixelIdxList), 1);
```

```
D_smooth=double(rgb2gray(IblurX1));
D_smooth(D_smooth <= 0.053) = NaN;
% Definición de los ejes X, Y y Z
[ny, nx] = size(D_smooth);
[X, Y] = meshgrid(1:nx, 1:ny);

figure(7)
     hold on
% Iterar sobre cada segmento y dibujar la curva correspondiente
for i = 1:length(NNn.PixelIdxList)

    S=ones(size(D_smooth))
    S(NNn.PixelIdxList{i})=0;
    D_euclidean = bwdist(S,'euclidean');
% Aplicación del filtro gaussiano
sigma = 2.3; % Desviación estándar del filtro gaussiano
D_smooth = imgaussfilt(D_euclidean, sigma);
D_smooth(D_smooth <= 0.15) = NaN;

    Z = D_smooth;
    fs = surf(X, Y, Z);
    centroid = regionprops(NNn, 'Centroid');
    x = double(centroid(i).Centroid(1));
    y = double(centroid(i).Centroid(2));
    z = double(max(Z(:))); % Altura para colocar el texto

    % Agrega el número de componente como texto en la posición del
centroide
    text(x, y, z, num2str(i), 'FontSize', 12, 'FontWeight', 'bold',
'Color', 'r');



    fs.EdgeAlpha=0
    % Pintar la curva del segmento actual
    % Obtener el mapa de colores 'turbo'

    % Obtener los valores RGB del color actual
    R = mapa_colores(round(i*255/length(NNn.PixelIdxList)), 1);
    G = mapa_colores(round(i*255/length(NNn.PixelIdxList)), 2);
    B = mapa_colores(round(i*255/length(NNn.PixelIdxList)), 3);

    % Guardar los valores RGB en una celda del vector
    vector_colores{i} = [R, G, B];
    fs.FaceColor=vector_colores{i};
```

```
    % Agregar una pausa para apreciar cada segmento de la curva
    %pause(1);
end

% Establecer el mapa de colores deseado
hold off
daspect([1 1 1]);
view(3);
axis tight equal

light                % create a light
lighting gouraud     % preferred method for lighting curved surfaces
material dull
daspect([1 1 1])
grid off
axis off


% Supongamos que tienes las coordenadas de los vértices en
eskel_Branchnodes y
% los números de los elementos (barras) en k.

matriz_incidencia = horzcat(incidencia_vector{:});
% Supongamos que tienes una matriz de incidencia llamada
matriz_incidencia


%for  i = 1:length(ind_clear_graph)

%    k=ind_clear_graph{i};
   % incidenciaaaaaa=matriz_incidencia(:,k);
%   ind_clear_branchPts=find(matriz_incidencia(:,k)==1) % encuentro
los vertices que generaran un problema
%centroid_branchPts=mean(branchPts(ind_clear_branchPts)) %calculo
el centroide de los vertices problematicos
%New_vertice_vector=sum(matriz_incidencia(ind_clear_branchPts,:)) %
uno ambos vectores vertices
% New_vertice_vector(New_vertice_vector>0)=1; % las conexxiones
quedan como 1 de ese nuevo branchpoints
%branchPts(ind_clear_branchPts)=[]; %elimino ambos vertices
problematicos
%branchPts(ind_clear_branchPts)=[];
%branchPts(min(ind_clear_branchPts))=[];
%matriz_incidencia(ind_clear_branchPts,:)=[];
%matriz_incidencia(min(ind_clear_branchPts),:)=New_vertice_vector'

%matriz_incidencia(:,k)=[]
%branchPts(ind_clear_branchPts)= cut_brancPts;
```

```
%end



% Calcula la matriz de adyacencia
% Dada una matriz de incidencia llamada 'matriz_incidencia'
% Donde las filas son nodos y las columnas son aristas (0 y 1)

% Obtener el número de nodos y aristas
num_nodos = size(matriz_incidencia, 1);
num_aristas = size(matriz_incidencia, 2);



% Recorrer las aristas y llenar la matriz de adyacencia
for i = 1:num_nodos
    clear k
    X_incidencia{i}=matriz_incidencia(i, :);
    k = find(X_incidencia{i}>0);
    X_adyacencia=sum(matriz_incidencia(:,k),2);
    X_adyacencia(X_adyacencia>0)=1;
    adyacencia_vector{i}=X_adyacencia;

    %[Regresionlineal{k},Sl] = polyfit(almacen_x,almacen_y,1)
  % rsquarelineal = 1 - Sl.normr^2 / norm(almacen_y -
mean(almacen_y))^2

   % Tu arreglo de 5 elementos
     arreglo{i} = k;
   % Encuentra todas las combinaciones duales (pares) utilizando
nchoosek
     combinaciones_duales{i} = nchoosek(arreglo{i}, 2);
     % Elimina duplicados
     combinaciones_duales{i}= unique(sort(combinaciones_duales{i},
2), 'rows');

   % coeficientes1 y coeficientes2 son los coeficientes de los
polinomios
    % en orden descendente (del término de mayor grado al término
constante).
    ii=[];

for ii = 1:length(combinaciones_duales{i}(:,1))
    combinacion_actual = combinaciones_duales{i};

    % Polinomios dados por los coeficientes
```

```
    polinomio1 =
poly2sym(double(regresionalmacen(combinacion_actual(ii,1),:)),sym('
x'));
    polinomio2 =
poly2sym(double(regresionalmacen(combinacion_actual(ii,2),:)),sym('
x'));

    %polinomio1 =
poly2sym(Regresionlineal{combinacion_actual(ii,1)});
    %polinomio2 =
poly2sym(Regresionlineal{combinacion_actual(ii,2)});
    % Encontrar las raíces (intersecciones)
    ecuacion_total = polinomio1 - polinomio2;
    raices = solve(ecuacion_total,sym('x'));
      if isempty(raices)
        disp('No hay soluciones para la ecuación');
          x_interseccion(ii) = [];
          y_interseccion(ii) = [];
        % Puedes agregar aquí cualquier código que desees ejecutar
en caso de no haber soluciones
      else
      x_interseccion=[];y_interseccion=[]
    % Tomar la primera raíz (puedes ajustar esto según tus
necesidades)
    x_interseccion(ii) = double(raices);
    % Evaluar en las ecuaciones originales para encontrar y
    y_interseccion(ii) =
polyval(double(regresionalmacen(combinacion_actual(ii,1),:)),
x_interseccion(ii));
      end
end
   x_interseccion_final{i} = [x_interseccion';branchPts(i, 2)];
%nuevos nodos en x
   y_interseccion_final{i} = [y_interseccion';branchPts(i, 1)];
%nuevos nodos en y


   % Punto de referencia
   punto_referencia = [];
punto_referencia = [branchPts(i, 2), branchPts(i, 1)];


% Calcular la distancia euclidiana
distancias{i} = sqrt((x_interseccion_final{i} -
punto_referencia(1)).^2 + (y_interseccion_final{i} -
punto_referencia(2)).^2);

% Definir umbral de distancia
```

```
umbral = 25;
indices_mantenidoss=[];
% Filtrar los puntos que están dentro del umbral de distancia
indices_mantenidos{i} = find(distancias{i} <= umbral);
indices_mantenidoss= indices_mantenidos{i};
% Mantener solo los puntos que cumplen con el criterio
x_interseccion_final_filtrado{i} =
x_interseccion_final{i}(indices_mantenidos{i},1);
y_interseccion_final_filtrado{i} =
y_interseccion_final{i}(indices_mantenidos{i},1);

% Convertir celdas a matrices numéricas
%puntos_filtrados = cell2mat([x_interseccion_final_filtrado{i},
y_interseccion_final_filtrado{i}]);

   radio = umbral;

%clustersCentroids=mean([x_interseccion_final_filtrado{i},y_interse
ccion_final_filtrado{i}]);
  % branchPts(:, 1)
  % Aplicar clusterXYpoints a las matrices numéricas
%[clustersCentroids, clustersGeoMedians, clustersXY] =
clusterXYpoints(puntos_filtrados, radio);

[clustersCentroids,clustersGeoMedians,clustersXY]=clusterXYpoints([
x_interseccion_final_filtrado{i}
y_interseccion_final_filtrado{i}],radio)
branchPts_interseccion{i}=clustersCentroids;
[row_find,col_find] = find(clustersCentroids(:,1)>0 &
clustersCentroids(:,2)>0)
branchPts_corregido(i,:)=clustersCentroids(row_find,:);


   %y_pred_lineal = polyval(Regresionlineal,almacen_x); % valores
predichos


end
matriz_adyacencia = horzcat(adyacencia_vector{:});
matriz_adyacencia = matriz_adyacencia -
diag(diag(matriz_adyacencia));

% Ahora, matriz_adyacencia es una matriz cuadrada que representa
las conexiones entre nodos
figure(34)
      N = 1;           % second-degree polynomial
maxDistance = 0.1; % maximum allowed distance for a point to be
inlier
```

```
[P, inlierIdx] =
fitPolynomialRANSAC([branchPts_corregido(:,1),branchPts_corregido(:
,2)],N,maxDistance);

yRecoveredCurve = polyval(P,branchPts_corregido(:,2));

plot(branchPts_corregido(:,2),yRecoveredCurve,'-g','LineWidth',3)

figure(97)
hold on
for i = 1:num_nodos


scatter(branchPts_interseccion{i}(:,1),branchPts_interseccion{i}(:,
2),'filled')
%scatter(x_interseccion_final{i}(:,1),y_interseccion_final{i}(:,1),
'filled')
end
hold off
% Supongamos que tienes una matriz de adyacencia llamada
matriz_adyacencia

figure(98)
% Crear un objeto de gráfico a partir de la matriz de adyacencia y
las coordenadas de los nodos
G = graph(matriz_adyacencia);

% Establecer las coordenadas de los nodos en el objeto del grafo
%branchPts_corregido(3,1)=[10]
%branchPts_corregido(2,1)=[10]
G.Nodes.Coordinates = branchPts_corregido;
%j = boundary(branchPts_corregido(:,1),branchPts_corregido(:,2));
%hold on;
%plot(branchPts_corregido(:,1),branchPts_corregido(:,2));

% Plotear el grafo con las coordenadas especificadas
h = plot(G, 'XData', branchPts_corregido(:, 1), 'YData', -
branchPts_corregido(:, 2));
daspect([1 1 1])
% Personalizar la apariencia del grafo (opcional)
title('Grafo a partir de la Matriz de Adyacencia');
xlabel('Eje X');
ylabel('Eje Y');



for i=1:length(NNn_skel.PixelIdxList)
```

```
    %Ld(NNn_skel.PixelIdxList{i})=0;
    list_NNn_skel=NNn_skel.PixelIdxList{i};
    indice_NNn_skel = round(numel(NNn_skel.PixelIdxList{i})/2);
    indice_NNn_skel = list_NNn_skel(indice_NNn_skel);
    k_falso(i)=Ld(indice_NNn_skel);
    if k_falso(i)==0
        k_falso(i)=[];
    else
         NNn_mask_real{i}=NNn_mask.PixelIdxList{k_falso(i)};
    end

end

%RECTIFICACION GRAFO::::::::::::::::::::::::::::::::::::
% Supongamos que tienes un objeto de gráfico G con las coordenadas
de los nodos
% G = graph(matriz_adyacencia);
% G.Nodes.Coordinates = [x1 y1; x2 y2; ...];

% Obtén las coordenadas de los nodos y las aristas del grafo
nodos = G.Nodes.Coordinates;
aristas = table2array(G.Edges);

% Parámetro de diferencia en y permitido
diferencia_y_permitida = 5;

% Inicializa una nueva matriz de nodos corregidos
nodos_corregidos = nodos;

% Itera sobre las aristas
for i = 1:size(aristas, 1)
    % Obtiene los nodos conectados por la arista
    nodo1 = aristas(i, 1);
    nodo2 = aristas(i, 2);

    % Verifica si los nodos están alineados y tienen una diferencia
en y permitida
    if abs(nodos(nodo1, 2) - nodos(nodo2, 2)) <=
diferencia_y_permitida
        % Calcula el promedio de las coordenadas y corrige ambos
nodos
        promedio_y = mean([nodos(nodo1, 2); nodos(nodo1, 2)]);
        nodos_corregidos([nodo1, nodo2], 2) = promedio_y;
    end
end

% Crea un nuevo objeto de gráfico con los nodos corregidos
G_corregido = graph(aristas(:, 1), aristas(:, 2));
```

```
G_corregido.Nodes.Coordinates = nodos_corregidos;

% Visualiza el grafo original y el grafo corregido
figure(33);
subplot(1, 2, 1);
plot(G, 'Layout', 'force', 'NodeLabel', {});
title('Grafo Original');

subplot(1, 2, 2);
plot(G_corregido, 'Layout', 'force', 'NodeLabel', {});
title('Grafo Corregido');
%::::::::::::::::::::::::::::::::::::::::::::::::::::::::
NNn_mask.PixelIdxList=NNn_mask_real;
figure(130)

labeled = labelmatrix(NNn);
whos labeled
RGB_label = label2rgb(labeled,'jet','k','shuffle');
imshow(RGB_label)
for i=1:length(NNn_skel.PixelIdxList)
     centroid = regionprops(NNn_skel, 'Centroid');
    x1 = double(centroid(i).Centroid(1));
    y1 = double(centroid(i).Centroid(2));
    z1 = double(length(NNn_skel.PixelIdxList)); % Altura para
colocar el texto

    % Agrega el número de componente como texto en la posición del
centroide
    text(x1, y1, z1, num2str(i), 'FontSize', 12, 'FontWeight',
'bold', 'Color', 'r');
end

figure(88)
hold on
for k=1:length(NNn_skel.PixelIdxList)
NNn_mask.PixelIdxList

segm_vector{k} = NNn_mask.PixelIdxList{k};
ind = [segm_vector{k}];
zzz = NNn_mask.ImageSize;
[almacen_x,almacen_y] = ind2sub(zzz,ind)


[Regresionlineal,Sl] = polyfit(almacen_x,almacen_y,1)
   rsquarelineal = 1 - Sl.normr^2 / norm(almacen_y -
mean(almacen_y))^2
```

```
   y_pred_lineal = polyval(Regresionlineal,almacen_x); % valores
predichos

   plot(almacen_x,almacen_y)
     plot(almacen_x,y_pred_lineal,'r')
        N = 1;           % second-degree polynomial
maxDistance = 0.1; % maximum allowed distance for a point to be
inlier

[P, inlierIdx] =
fitPolynomialRANSAC([almacen_x,almacen_y],N,maxDistance);

yRecoveredCurve = polyval(P,almacen_x);

plot(almacen_x,yRecoveredCurve,'-g','LineWidth',3)

end
hold off
% Crear una imagen binaria de ejemplo (reemplaza esto con tu
esqueleto)
esqueleto_binario = logical(labeled);

% Etiquetar regiones conectadas en la imagen
etiquetas = bwlabel(esqueleto_binario, 4);

% Calcular el número de regiones etiquetadas
num_regiones = max(etiquetas(:));

% Inicializar una matriz de conectividad
matriz_conectividad = zeros(num_regiones, num_regiones);

% Llenar la matriz de conectividad
for i = 1:num_regiones
    for j = 1:num_regiones
        % Comprobar si las regiones i y j están conectadas
        if any(etiquetas(:) == i) && any(etiquetas(:) == j)
            % En este ejemplo simplificado, se consideran
conectadas si comparten al menos un píxel vecino
            conexion = any(etiquetas == i & imdilate(etiquetas ==
j, strel('square', 3)));
            matriz_conectividad(i, j) = conexion;
        end
    end
end

% Visualizar la matriz de conectividad
disp(matriz_conectividad);
```

ANEXO 5.5: Funciones Utilizadas en el Procesamiento de Imágenes

En esta sección, se describen las funciones desarrolladas para el procesamiento de imágenes, las cuales se utilizaron en diversos procedimientos de segmentación y análisis de contornos. Las funciones están implementadas en MATLAB y permiten realizar operaciones complejas de una manera eficiente y modular.

Esta función convierte una imagen en escala de grises, la suaviza utilizando un filtro gaussiano, aplica la transformada de distancia y finalmente binariza la imagen resultante.

$$[\mathrm{BW}] = \mathrm{imbinarize}(\mathrm{imgaussfilt}(\mathrm{bwdist}(\mathrm{I}_{\mathrm{softkillbeso}}, \mathrm{'parameters'})))$$

```
function BW = procesarImagen(I_softkillbeso)
    % Convertir a escala de grises, suavizar, aplicar transformada
de distancia y binarizar
    BW =
imbinarize(bwdist(~imbinarize(imgaussfilt(rgb2gray(I_softkillbeso),
2.39))), 0.049 *
max(bwdist(~imbinarize(imgaussfilt(rgb2gray(I_softkillbeso),
2.39)))(:)));

    % Mostrar las imágenes en un montaje (opcional)
    img_gray = rgb2gray(I_softkillbeso);
    img_smooth = imgaussfilt(img_gray, 2.39);
    img_dist = bwdist(~imbinarize(img_smooth));
    figure;
    montage({img_gray, img_smooth, img_dist, BW}, 'Size', [4, 1]);
end
```

Esta función elimina componentes conectados en una imagen binarizada que son menores a un área mínima especificada, calculada en proporción al área total .

$$[\mathrm{BW}] = \mathrm{Basic\_filt\_umbral\_area}(\mathrm{BW2}, \mathrm{"filtro} = 0.5\%\mathrm{"})$$

```
function BW = Basic_filt_umbral_area(BW2, filtro)
    % Calcular el área mínima tolerable
    area_total = sum(BW2(:)); % Calcular el área total
    area_min = filtro * area_total; % Área mínima en proporción al
área total
```

```
    % Identificar componentes conectados
    CC = bwconncomp(BW2);
    stats = regionprops(CC, 'Area');

    % Eliminar componentes con área menor al umbral
    idx = find([stats.Area] < area_min);
    BW = BW2;
    for i = 1:length(idx)
        BW(CC.PixelIdxList{idx(i)}) = 0;
    end
end
```

Esta función aplica un suavizado robusto a las coordenadas de un contorno, utilizando métodos de mínimos cuadrados penalizados.

$$Vs\{i\} = \text{smoothn}(\{Vx, Vy\}, 'robust')$$

```
function Vs = smooth_contour(Vx, Vy)
    % Aplicar suavizado a las coordenadas del contorno

    % Calcular la curvatura en cada punto
    dx = diff(Vx);
    dy = diff(Vy);
    d2x = diff(dx);
    d2y = diff(dy);
    curvature = abs(d2x .* dy(1:end-1) - dx(1:end-1) .* d2y) ./
((dx(1:end-1).^2 + dy(1:end-1).^2).^(3/2));

    % Filtrar para detectar los puntos de mayor curvatura
    threshold = 0.018; % Umbral para considerar un punto como de
mayor curvatura
    [~, locs] = findpeaks(curvature, 'MinPeakHeight', threshold);

    % Obtener los puntos de curvatura
    x_points = Vx(locs);
    y_points = Vy(locs);

    % Escalar los puntos de curvatura respecto a su centroide
    puntos_curvatura = [x_points, y_points];
    punto_referencia = mean(puntos_curvatura);
    factor_escala = 1.005;
    diferencia = puntos_curvatura - punto_referencia;
    puntos_escalados = punto_referencia + factor_escala *
diferencia;
```

```
    % Obtener las coordenadas escaladas
    x_points = puntos_escalados(:, 1);
    y_points = puntos_escalados(:, 2);

    % Preparar los datos para el suavizado
    Vx = [Vx; Vx(1:round(length(Vx)*0.11))];
    Vy = [Vy; Vy(1:round(length(Vy)*0.11))];

    % Aplicar el suavizado
    Vs = smoothn({Vx, Vy}, 'robust');

    % Devolver las coordenadas suavizadas
    Vs = cellfun(@(x) x(1:end-round(length(x)*0.11)), Vs,
'UniformOutput', false);
end
```

Esta función calcula la curvatura de un conjunto de puntos, además de la longitud del arco y el radio de curvatura.

$$[L, R, k] = \text{curvature}(X)$$

```
function [L, R, k] = curvature(X)
    % Calcular la curvatura de un conjunto de puntos
    % X es una matriz de Nx2, donde N es el número de puntos (x, y)

    % Extraer las coordenadas
    x = X(:, 1);
    y = X(:, 2);

    % Calcular las derivadas primeras y segundas
    dx = gradient(x);
    dy = gradient(y);
    d2x = gradient(dx);
    d2y = gradient(dy);

    % Calcular la curvatura
    k = abs(d2x .* dy - dx .* d2y) ./ ((dx .^ 2 + dy .^ 2) .^ (3 /
2));

    % Calcular la longitud del arco
    L = cumsum([0; sqrt(diff(x) .^ 2 + diff(y) .^ 2)]);

    % Calcular el radio de curvatura
    R = 1 ./ k;
end
```

Esta función implementa un algoritmo de búsqueda en profundidad para obtener el camino vectorizado a lo largo de un perímetro en una imagen binarizada.

$$[\text{Vectorfinal}] = \text{deep}_{\text{search}}(\text{BW2}, \text{Point}_{\text{init}})$$

```
function Vectorfinal = deep_search(BW2, Point_init)
    % Implementar la búsqueda en profundidad para obtener el camino
vectorizado
    % ejemplo simplificado

    % Crear una pila para almacenar los puntos a visitar
    stack = Point_init;
    visited = false(size(BW2));
    Vectorfinal = [];

    % Mientras haya puntos en la pila
    while ~isempty(stack)
        % Sacar el punto actual de la pila
        current_poi
nt = stack(end, :);
        stack(end, :) = [];

        % Marcar el punto como visitado
        visited(current_point(1), current_point(2)) = true;

        % Añadir el punto al camino final
        Vectorfinal = [Vectorfinal; current_point];

        % Obtener los vecinos del punto actual
        neighbors = get_neighbors(current_point, size(BW2));

        % Añadir los vecinos no visitados a la pila
        for i = 1:size(neighbors, 1)
            if ~visited(neighbors(i, 1), neighbors(i, 2)) &&
BW2(neighbors(i, 1), neighbors(i, 2))
                stack = [stack; neighbors(i, :)];
            end
        end
    end
end

function neighbors = get_neighbors(point, img_size)
    % Obtener los vecinos de un punto
    x = point(1);
    y = point(2);
```

```
    neighbors = [x-1, y; x+1, y; x, y-1; x, y+1];

    % Eliminar vecinos fuera de los límites de la imagen
    neighbors(neighbors(:, 1) < 1 | neighbors(:, 1) > img_size(1) |
neighbors(:, 2) < 1 | neighbors(:, 2) > img_size(2), :) = [];
end
```

Esta función realiza el fileteado de una curva con un radio de entrada, utilizando un número específico de puntos por arco de filete.

$$[\mathrm{Arc_x}, \mathrm{Arc_y}] = \mathrm{RoundedCornerPolygon}(\mathrm{x_{ps}}, \mathrm{y_{pts}}, \mathrm{r}, \mathrm{np_{arco}}, \mathrm{db})$$

```
function [Arc_x, Arc_y] = RoundedCornerPolygon(x_ps, y_pts, r,
np_arco, db)
    % Realiza el fileteado de una curva
    % x_ps, y_pts: Coordenadas de los puntos
    % r: Radio del filete
    % np_arco: Número de puntos por arco de filete
    % db: Densidad de bordes (opcional)

    % Inicializar las listas de puntos del arco
    Arc_x = [];
    Arc_y = [];

    % Calcular los puntos del arco para cada esquina
    for i = 1:length(x_ps)
        % Puntos actuales
        x1 = x_ps(mod(i-1-1, length(x_ps))+1);
        y1 = y_pts(mod(i-1-1, length(y_pts))+1);
        x2 = x_ps(i);
        y2 = y_pts(i);
        x3 = x_ps(mod(i+1-1, length(x_ps))+1);
        y3 = y_pts(mod(i+1-1, length(y_pts))+1);

        % Calcular el ángulo entre los puntos
        ang = atan2(y3 - y2, x3 - x2) - atan2(y1 - y2, x1 - x2);

        % Calcular los puntos del arco
        theta = linspace(0, abs(ang), np_arco);
        if ang > 0
            theta = -theta;
        end
        arc_x = x2 + r * cos(theta);
        arc_y = y2 + r * sin(theta);

        % Añadir los puntos del arco a las listas
        Arc_x = [Arc_x, arc_x];
```

```
        Arc_y = [Arc_y, arc_y];
    end

    % Suavizar las curvas del arco (opcional)
    if exist('db', 'var') && db > 0
        Arc_x = smooth(Arc_x, db);
        Arc_y = smooth(Arc_y, db);
    end
end
```

Esta función utiliza splines cúbicos para conectar puntos de control y proporcionar una representación continua de una cavidad.

$$\text{values\_spline} = \text{spcrv}(c, 'k = 3', \text{maxpnt})$$

```
function values_spline = reconstruirSpline(c, k, maxpnt)
    % Reconstrucción por B-spline y fillet utilizando la función
spcrv
    % c: Vector de coeficientes de B-spline (puntos de control)
    % k: Orden de B-spline (3 para splines cúbicos)
    % maxpnt: Número de puntos a generar a lo largo de la cavidad

    % Calcular los valores de la spline
    values_spline = spcrv(c, k, maxpnt);
end
```

Esta función genera una malla de nodos y elementos en 3D con cavidades, utilizando una grilla de puntos y ajustando los puntos según su pertenencia a las cavidades.

$$[\text{NODES}, \text{ELEMENTS}] = \text{holes\_generator}(\text{i\_holes}, \text{grosor})$$

```
function [nodes, elements, xg, yg] = holes_generator(i_holes,
grosor, alpha_xy, alpha_xyz, x_hole, y_hole, x_a, x_b, y_a, y_b,
Number_spacing_x, Number_spacing_y)
    % Generar malla de nodos y elementos 3D con cavidades
    % i_holes: Índices de cavidades
    % grosor: Grosor de la malla
    % alpha_xy: Parámetro alpha para xy
    % alpha_xyz: Parámetro alpha para xyz
    % x_hole, y_hole: Coordenadas de cavidades
    % x_a, x_b, y_a, y_b: Límites del área de interés
    % Number_spacing_x, Number_spacing_y: Número de puntos en x y y

    % Generar grilla de puntos en 2D
```

```
    [xg, yg] = meshgrid(linspace(x_a, x_b, Number_spacing_x),
linspace(y_a, y_b, Number_spacing_y));
    xg = xg(:);
    yg = yg(:);

    % Inicializar celdas para coordenadas de cavidades
    N = numel(x_hole);
    x_cells = cell(N, 1);
    y_cells = cell(N, 1);

    % Procesar cada cavidad
    for i = 1:N
        % Guardar coordenadas de cavidades en celdas
        x_cells{i} = x_hole{i};
        y_cells{i} = y_hole{i};

        % Crear alphaShape para la cavidad actual
        circShp = alphaShape([x_cells{i}], [y_cells{i}], alpha_xy);
        in = inShape(circShp, xg, yg); % Puntos dentro de la
cavidad

        % Actualizar puntos según su pertenencia a cavidades
        if i == 1
            xg = [xg(in); x_cells{i}];
            yg = [yg(in); y_cells{i}];
            inside = inpolygon(x_hole{1}, y_hole{1}, xg, yg);
            xg = xg(inside);
            yg = yg(inside);
        elseif ismember(i, i_holes)
            xg = [xg(~in)];
            yg = [yg(~in)];
            if i == N
                xg = [xg; x_cells{1}];
                yg = [yg; y_cells{1}];
            end
        else
            xg = [xg(~in); x_cells{i}];
            yg = [yg(~in); y_cells{i}];
        end
    end

    % Crear copias 3D de los puntos restantes
    zg = ones(numel(xg), 1) * grosor;
    xg = repmat(xg, 3, 1);
    yg = repmat(yg, 3, 1);
    zg = zg * (0:0.5:1);
    zg = zg(:);
    shp = alphaShape(xg, yg, zg, alpha_xyz);
```

```
    % Obtener malla de la figura
    [elements, nodes] = boundaryFacets(shp);
    nodes = nodes';
    elements = elements';
end
```

Esta función realiza la erosión y eliminación de ramificaciones no deseadas en un esqueleto, preservando la topología original.

$$[prunedSkeleton, branchPoints, endPoints] = spur\_skeleton(D) \quad (0.1)$$

```
function [prunedSkeleton, branchPoints, endPoints] =
spur_skeleton(D)
    % Binarizar la imagen de distancia euclidiana
    BW = imbinarize(D);

    % Obtener el esqueleto de la imagen binarizada
    skelImg = bwmorph(BW, 'skel', Inf);

    % Realizar la poda del esqueleto
    prunedSkeleton = bwmorph(skelImg, 'spur', Inf);

    % Detectar puntos de bifurcación y extremos
    branchImg = bwmorph(prunedSkeleton, 'branchpoints');
    endImg = bwmorph(prunedSkeleton, 'endpoints');

    % Obtener las coordenadas de los puntos de bifurcación y
extremos
    [branchRows, branchCols] = find(branchImg);
    [endRows, endCols] = find(endImg);

    branchPoints = [branchRows, branchCols];
    endPoints = [endRows, endCols];
end
```

Las funciones presentadas en este anexo forman parte de los procesos de segmentación y análisis de contornos en imágenes. Su implementación modular permite una manipulación eficiente y precisa de las imágenes, facilitando el desarrollo de técnicas avanzadas de procesamiento y análisis.

ANEXO 6: Conexión con SAP2000 v.23

Esta función recibe como entrada la matriz *bars* que contiene la información de las barras (columnas) y crea cada una de ellas en SAP2000 utilizando las coordenadas y el material definido. La función también configura las unidades y define las propiedades de los materiales de concreto y acero de refuerzo.

```
function [FrameNames] = generarEstructuraConSAP2000(bars)
    % Inicialización de la conexión con SAP2000
    SM.App('sap');
    SM.Ver('23');
    APIDLLPath ='C:\Program Files\Computers and Structures\SAP2000
23\SAP2000v1.dll';
    ProgramPath ='C:\Program Files\Computers and Structures\SAP2000
23\SAP2000.exe';
    [Sobj] = SM.Helper.CreateObject(ProgramPath, APIDLLPath);
    [Smdl] = SM.SapModel();
    SM.ApplicationStart;
    Smdl.File.NewBlank;

    % Definición de unidades
    Smdl.SetPresentUnits(SM.eUnits.N_cm_C);

    % Definición de material de concreto
    Smdl.PropMaterial.SetMaterial('C210', SM.eMatType.Concrete);
    Smdl.PropMaterial.SetOConcrete_1('C210', 210, false, 0, 2, 2,
0.002, 0.005, -0.1);
    Smdl.PropMaterial.SetMPIsotropic('C210', 2173706, 0.20, 9.9E-
06);

    % Definición de material de acero de refuerzo
    [ret, Name] = Smdl.PropMaterial.AddQuick(SM.eMatType.Rebar,
'RebarType', SM.eMatTypeRebar.ASTM_A615Gr60);
    Smdl.PropMaterial.SetORebar_1('A615Gr60', 42000, 63000, 46200,
69300, 1, 1, 0.01, 0.09, -0.1, false);
    Smdl.PropMaterial.SetMPIsotropic('A615Gr60', 200000000, 0.30,
1.170E-05);

    % Definición de la sección transversal para columnas
    Smdl.PropFrame.SetRectangle('Columna', 'C210', 0.5, 0.5);

    % Creación de columnas en SAP2000 basado en las barras
    FrameNames = cell(length(bars), 1);
    for j = 1:length(bars)
        FrameName = sprintf('Columna%d', j);
```

```
        [ret, ~] = Smdl.FrameObj.AddByCoord(bars(j,1), bars(j,2),
bars(j,3), bars(j,4), bars(j,5), bars(j,6), 'Columna', FrameName);
        FrameNames{j} = FrameName;
    end

    % Aquí se pueden añadir más elementos como vigas o losas si se
requiere.

    % Cerrar la conexión con SAP2000
    Sobj.ApplicationExit(False);
    Sobj.release;
end


function [bars, bars_simet_der, bars_simet_izq] =
generarCurvasInterpolacion(Point_inicio, Point_final,
clustersGeoMedians)
    % Inicialización de matrices para almacenar barras y sus
simetrías
    bars = [];
    bars_simet_der = [];
    bars_simet_izq = [];

    % Generación de puntos iniciales y finales
    x = double([Point_inicio(1), Point_final(1)]);
    y = double([Point_inicio(2), Point_final(2)]);

    % Generación de curvas de interpolación
    XY = [x', y'];
    BS = BSpline(XY, 5);

    % Dibujo de las curvas
    figure(101);
    hold on;
    plot(BS(:,2), BS(:,1), '-', 'LineWidth', 4, 'Color', [0, 0,
0]);
    camroll(-180);
    hold off;

    % Cálculo de puntos de simetría y generación de barras
simétricas
    x_simetria = mean(clustersGeoMedians(:, 2));
    y_simetria = mean(clustersGeoMedians(:, 1));
    for kk = 1:length(clustersGeoMedians(:, 1))
        if x(kk) - (x_simetria + 2) >= 0
            bars_simet_der = [bars_simet_der;
generarBarraSimetriaDerecha(x(kk, :), y(kk, :), x_simetria)];
        else
```

```
            bars_simet_izq = [bars_simet_izq;
generarBarraSimetriaIzquierda(x(kk, :), y(kk, :), y_simetria)];
        end
    end

    % Combinación de barras para obtener la estructura final
    bars = [bars; bars_simet_der; bars_simet_izq];
end

% Funciones auxiliares para generar barras simétricas
function barra = generarBarraSimetriaDerecha(x, y, x_simetria)
    barra = [x(1), y(1), 0, x(2) - 2 * (x(2) - x_simetria), y(2),
0];
end

function barra = generarBarraSimetriaIzquierda(x, y, y_simetria)
    barra = [x(1), y(1) - 2 * (y(1) - y_simetria), 0, x(2), y(2) -
2 * (y(2) - y_simetria), 0];
end
```

ANEXO 7: Generación de tablas basado en comportamiento estructural desde SAP2000

La siguiente función en MATLAB procesa y tabula datos de fuerzas en las articulaciones y deformaciones obtenidas de SAP2000, convirtiendo los datos de celdas a un formato adecuado y organizando la información en tablas para su análisis y visualización:

```
function tabulateSAP2000JointData(elementsforces_joints,
elementsdeformate_joints)
    % Función para convertir datos de celdas a un formato adecuado
    function result = convertirCeldaAutomatico(r)
        result = cell(size(r));
        for i = 1:numel(r)
            if isnumeric(r{i})
                result{i} = r{i};
            elseif ischar(r{i})
                if isempty(strfind(r{i}, ' ')) &&
~isempty(strfind(r{i}, 'e'))
                    result{i} = [''''' r{i} '''''];
                else
                    result{i} = r{i};
                end
            else
                result{i} = r{i};
            end
        end
    end

    % Aplicar la conversión a los datos de entrada
    elementsforces_joints =
convertirCeldaAutomatico(elementsforces_joints);
    elementsdeformate_joints =
convertirCeldaAutomatico(elementsdeformate_joints);

    % Procesamiento de fuerzas en las articulaciones
    jointForcesData = cell2mat(elementsforces_joints(:, [1, 5, 6,
10]));
    uniqueJoints = unique(jointForcesData(:, 1));

    P_tabulated = zeros(size(uniqueJoints));
    v2_tabulated = zeros(size(uniqueJoints));
    M3_tabulated = zeros(size(uniqueJoints));

    for i = 1:length(uniqueJoints)
        jointIndices = jointForcesData(:, 1) == uniqueJoints(i);
```

```
        P_tabulated(i) = max(abs(jointForcesData(jointIndices,
2)));
        v2_tabulated(i) = max(abs(jointForcesData(jointIndices,
3)));
        M3_tabulated(i) = max(abs(jointForcesData(jointIndices,
4)));
    end

    % Procesamiento de deformaciones en las articulaciones
    jointDeformationsData = cell2mat(elementsdeformate_joints(:,
[1, 4, 5, 6]));
    U123_tabulated = zeros(size(uniqueJoints));

    for i = 1:length(uniqueJoints)
        deformationIndices = jointDeformationsData(:, 1) ==
uniqueJoints(i);
        U123_tabulated(i) =
max(sqrt(sum(jointDeformationsData(deformationIndices, 2:4).^2,
2)));
    end

    % Mostrar las tablas de resultados
    fprintf('Nro.Punto   Axial [KN]   Corte [KN]   Momento [KN-m]
Deflexión [cm]\n');
    fprintf('----------------------------------------------------
----------------\n');
    for i = 1:length(uniqueJoints)
        fprintf('%d           %.2f           %.2f           %.2f
%.2f\n', ...
                uniqueJoints(i), P_tabulated(i), v2_tabulated(i),
M3_tabulated(i), U123_tabulated(i) * 100);
    end
end
```

Esta función tabulateSAP2000FrameData toma una matriz de celdas con los datos de los marcos y los procesa para generar una tabla que muestra el número de miembro junto con los valores máximos de fuerza axial, corte, momento y deflexión para cada elemento. Los datos son convertidos a un formato adecuado para su manipulación y visualización, permitiendo una interpretación clara de los resultados de análisis estructural obtenidos de SAP2000.

```
function tabulateSAP2000FrameData(elementsforces_frames)
    % Conversión de datos de la matriz de celdas a formato adecuado
    function result = convertirCeldaAutomatico(r)
        result = cell(size(r));
        for i = 1:numel(r)
            if isnumeric(r{i})
```

```
                result{i} = r{i};
            elseif ischar(r{i})
                if isempty(strfind(r{i}, ' ')) &&
~isempty(strfind(r{i}, 'e'))
                    result{i} = [''''' r{i} '''''];
                else
                    result{i} = r{i};
                end
            else
                result{i} = r{i};
            end
        end
    end

    % Aplicar la conversión a los datos de entrada
    elementsforces_frames =
convertirCeldaAutomatico(elementsforces_frames);

    % Extraer y procesar los datos de los elementos
    frameData = cell2mat(elementsforces_frames(:, [1, 5, 6, 10,
12]));
    uniqueFrames = unique(frameData(:, 1));

    % Inicialización de matrices para almacenar los datos tabulados
    P_tabulated = zeros(size(uniqueFrames));
    v2_tabulated = zeros(size(uniqueFrames));
    M3_tabulated = zeros(size(uniqueFrames));
    flecha_tabulated = zeros(size(uniqueFrames));

    % Tabulación de los datos
    for i = 1:length(uniqueFrames)
        frameIndices = frameData(:, 1) == uniqueFrames(i);
        P_tabulated(i) = max(abs(frameData(frameIndices, 2)));
        v2_tabulated(i) = max(abs(frameData(frameIndices, 3)));
        M3_tabulated(i) = max(abs(frameData(frameIndices, 4)));
        flecha_tabulated(i) = max(abs(frameData(frameIndices, 5)));
    end

    % Mostrar la tabla de resultados
    fprintf('Nro.Miembro   Axial [KN]   Corte [KN]   Momento [KN-
cm]   Deflexión [cm]\n');
    fprintf('-----------------------------------------------------
----------------\n');
    for i = 1:length(uniqueFrames)
        fprintf('%d           %.2f           %.2f           %.2f
%.2f\n', ...
                uniqueFrames(i), P_tabulated(i), v2_tabulated(i),
M3_tabulated(i), flecha_tabulated(i));
```

```
    end
end
```

La siguiente función en MATLAB procesa los datos de deformación en las articulaciones de una estructura analizada en SAP2000, comparando las deformaciones en los puntos de conexión de los elementos para identificar y calcular la diferencia máxima de deformación (deltaU) entre puntos conectados:

```
function calculateDeformationDifferences(elementsdeformate_joints,
conectivity)
    % Convertir celda a formato numérico adecuado
    function result = convertirCeldaAutomatico(r)
        result = cell(size(r));
        for i = 1:numel(r)
            if isnumeric(r{i})
                result{i} = r{i};
            elseif ischar(r{i})
                if isempty(strfind(r{i}, ' ')) &&
~isempty(strfind(r{i}, 'e'))
                    result{i} = [''''' r{i} '''''];
                else
                    result{i} = r{i};
                end
            else
                result{i} = r{i};
            end
        end
    end

    elementsdeformate_joints =
convertirCeldaAutomatico(elementsdeformate_joints);

    % Extraer datos de deformación
    U1 = cell2mat(elementsdeformate_joints(:,4));
    U2 = cell2mat(elementsdeformate_joints(:,5));
    U3 = cell2mat(elementsdeformate_joints(:,6));

    % Calcular la deformación total en cada punto
    U123_dist = sqrt(U1.^2 + U2.^2 + U3.^2);

    % Identificar la máxima deformación en cada punto
    uniquePoints = unique(cell2mat(elementsdeformate_joints(:,1)));
    U123_tabulated = zeros(length(uniquePoints), 1);

    for i = 1:length(uniquePoints)
```

```
        indices = cell2mat(elementsdeformate_joints(:,1)) ==
uniquePoints(i);
        U123_tabulated(i) = max(U123_dist(indices));
    end

    % Calcular la diferencia de deformación entre puntos conectados
    deltaU = zeros(size(conectivity, 1), 1);

    for i = 1:size(conectivity, 1)
        point1 = conectivity(i, 2);
        point2 = conectivity(i, 3);
        deltaU(i) = abs(U123_tabulated(point1) -
U123_tabulated(point2));
    end

    % Mostrar resultados
    disp('Elemento    Delta Deformación (m)');
    disp('------------------------------');
    for i = 1:length(deltaU)
        fprintf('%d           %.6f\n', i, deltaU(i));
    end
end
```

ANEXO 8: Rutina MATLAB® para viga diseño de interfaz gráfica

```
classdef MemoriaHomerespaldo < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        TabGroup                    matlab.ui.container.TabGroup
        CreateTab                   matlab.ui.container.Tab
        Panel2                      matlab.ui.container.Panel
        NelementosYEditFieldLabel   matlab.ui.control.Label
        NelementosYEditField
matlab.ui.control.NumericEditField
        EEditFieldLabel             matlab.ui.control.Label
        EEditField
matlab.ui.control.NumericEditField
        uEditFieldLabel             matlab.ui.control.Label
        uEditField
matlab.ui.control.NumericEditField
        FV_0EditFieldLabel          matlab.ui.control.Label
        FV_0EditField
matlab.ui.control.NumericEditField
        R_minEditFieldLabel         matlab.ui.control.Label
        R_minEditField
matlab.ui.control.NumericEditField
        NelementosXEditFieldLabel   matlab.ui.control.Label
        NelementosXEditField
matlab.ui.control.NumericEditField
        Panel2_2                    matlab.ui.container.Panel
        AlgoritmoDropDownLabel      matlab.ui.control.Label
        AlgoritmoDropDown           matlab.ui.control.DropDown
        ProblemaDropDownLabel       matlab.ui.control.Label
        ProblemaDropDown            matlab.ui.control.DropDown
        OptimizacinTopologicaLabel  matlab.ui.control.Label
        TabGroup2_2                 matlab.ui.container.TabGroup
        DesingTab                   matlab.ui.container.Tab
        UIAxes_5                    matlab.ui.control.UIAxes
        Button_11                   matlab.ui.control.Button
        ForceCheckBox               matlab.ui.control.CheckBox
        LocalAxisCheckBox           matlab.ui.control.CheckBox
        GrideCheckBox               matlab.ui.control.CheckBox
        RestrainCheckBox            matlab.ui.control.CheckBox
        Button                      matlab.ui.control.Button
        remove                      matlab.ui.control.Button
        NoRestrain                  matlab.ui.control.Button
        xRestrainDeslizante         matlab.ui.control.Button
        RestrainFixed               matlab.ui.control.Button
```

```
        RestrainEmpotred            matlab.ui.control.Button
        xButton_2                   matlab.ui.control.Button
        Button_9                    matlab.ui.control.Button
        RestrainLabel               matlab.ui.control.Label
        NodalLabel                  matlab.ui.control.Label
        ForceLabel                  matlab.ui.control.Label
        Button_10                   matlab.ui.control.Button
        RunLabel                    matlab.ui.control.Label
        SelectPolyLine              matlab.ui.control.Button
        SelecLine                   matlab.ui.control.Button
        SelectLabel                 matlab.ui.control.Label
        yRestrainDeslizante         matlab.ui.control.Button
        Button_14                   matlab.ui.control.Button
        yButton_2                   matlab.ui.control.Button
        PEditFieldLabel             matlab.ui.control.Label
        PEditField
matlab.ui.control.NumericEditField
        EditField                   matlab.ui.control.EditField
        SelectPolyLine_2            matlab.ui.control.Button
        circle                      matlab.ui.control.Button
        SelectPolyLine_4            matlab.ui.control.Button
        DeflectionAnalysisTab       matlab.ui.container.Tab
        UIAxes_11                   matlab.ui.control.UIAxes
        Button_15                   matlab.ui.control.Button
        AnalisisDropDownLabel_2     matlab.ui.control.Label
        AnalisisDropDown_3          matlab.ui.control.DropDown
        AnalisisDropDownLabel_3     matlab.ui.control.Label
        MinglobalCheckBox           matlab.ui.control.CheckBox
        HolesCheckBox_2             matlab.ui.control.CheckBox
        MaxglobalCheckBox           matlab.ui.control.CheckBox
        RestrainCheckBox_2          matlab.ui.control.CheckBox
        Simulation2DTab             matlab.ui.container.Tab
        UIAxes_9                    matlab.ui.control.UIAxes
Button_16 matlab.ui.control.Button
Simulation25DTab matlab.ui.container.Tab
UIAxes_10 matlab.ui.control.UIAxes
Button_17 matlab.ui.control.Button
ImageTab matlab.ui.container.Tab
image matlab.ui.control.Button
UIAxes3 matlab.ui.control.UIAxes
GenerateButton matlab.ui.control.Button
UnitsDropDownLabel matlab.ui.control.Label
UnitsDropDown matlab.ui.control.DropDown
MaterialTab matlab.ui.container.Tab
UIAxes2 matlab.ui.control.UIAxes
MaterialDropDownLabel matlab.ui.control.Label
MaterialDropDown matlab.ui.control.DropDown
TipodeseccinDropDownLabel matlab.ui.control.Label
```

```
TipodeseccinDropDown matlab.ui.control.DropDown
UnidadesDropDownLabel matlab.ui.control.Label
UnidadesDropDown matlab.ui.control.DropDown
SectionButton matlab.ui.control.Button
UIAxes2_2 matlab.ui.control.UIAxes
OnButton matlab.ui.control.Button
CNCTab matlab.ui.container.Tab
ROUNDEDButtonGroup matlab.ui.container.ButtonGroup
DropDown matlab.ui.control.DropDown
FORMATButtonGroup_2 matlab.ui.container.ButtonGroup
Button_24 matlab.ui.control.Button
DropDown_2 matlab.ui.control.DropDown
ExportButton_2 matlab.ui.control.Button
TabGroup2 matlab.ui.container.TabGroup
VisualTab matlab.ui.container.Tab
UIAxes matlab.ui.control.UIAxes
ShowButton_5 matlab.ui.control.Button
SimulationTab matlab.ui.container.Tab
UIAxes_3 matlab.ui.control.UIAxes
ShowButton_6 matlab.ui.control.Button
AnalisisDropDown_4Label matlab.ui.control.Label
AnalisisDropDown_4 matlab.ui.control.DropDown
EjeDropDownLabel matlab.ui.control.Label
EjeDropDown matlab.ui.control.DropDown
VoxelTab matlab.ui.container.Tab
UIAxes_4 matlab.ui.control.UIAxes
ShowButton_7 matlab.ui.control.Button
FILTERButtonGroup matlab.ui.container.ButtonGroup
ButtonFilter matlab.ui.control.Button
Button_29 matlab.ui.control.Button
Lamp matlab.ui.control.Lamp
Button_18 matlab.ui.control.Button
Button_23 matlab.ui.control.Button
ADVANCEDButtonGroup_2 matlab.ui.container.ButtonGroup
Button_25 matlab.ui.control.Button
Button_27 matlab.ui.control.Button
Lamp_2 matlab.ui.control.Lamp
RENDERINGButtonGroup matlab.ui.container.ButtonGroup
Button_26 matlab.ui.control.Button
Button_28 matlab.ui.control.Button
Lamp_3 matlab.ui.control.Lamp
Button_30 matlab.ui.control.Button
FRAMETab matlab.ui.container.Tab
end

properties (Access = private)
filter % Description
end
```

```
methods (Access = public)
function results = importdata(app,value)
end
end

% Callbacks that handle component events
methods (Access = private)

% Button pushed function: SectionButton
function SectionButtonPushed(app, event)
perfil = {'CHS 20x2.0', 'CHS 25x2.0', 'CHS 30x2.0', 'CHS 40x2.0',
'CHS 50x2.0', 'CHS 60x2.0', 'CHS 80x2.0', 'CHS 100x2.0', 'CHS
120x2.0', 'CHS 150x2.0', 'CHS 200x2.0', 'CHS 250x2.0', 'CHS
300x2.0', 'CHS 350x2.0', 'CHS 400x2.0'};
diametro_total = [20 25 30 40 50 60 80 100 120 150 200 250 300 350
400];
grosor_nominal = [2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0
2.0 2.0 2.0];

% Imprimir la lista
%for i = 1:length(perfil)
% fprintf('%s: Diámetro total = %d mm, Grosor nominal = %.1f mm\n',
perfil{i}, diametro_total(i), grosor_nominal(i));
%end

e=5
gm = multicylinder([diametro_total(e)-grosor_nominal(e)
diametro_total(e)],400,"Void",[true,false]) % en mm

%model = createpde
%model.Geometry = gm

%mesh_default = generateMesh(model)
%v = pdeviz(mesh_default,'Parent',app.UIAxes2)
%pdemesh(mesh_default)


%model = createpde;
%importGeometry(model,'Block.stl');
%applyBoundaryCondition(model,'dirichlet','Face',[1:4],'u',0);
%specifyCoefficients(model,'m',0,'d',0,'c',1,'a',0,'f',2);
%generateMesh(model);
%results = solvepde(model);
%u = results.NodalSolution;
%h=pdeplot3D(model);
%h(1).Parent = app.UIAxes2;
%view(app.UIAxes2,45,45)
%colorbar(app.UIAxes2)
```

```
%close(h(1).Parent)
% axis(app.UIAxes2,h,'on')
model = createpde;
importGeometry(model,'Block.stl');
applyBoundaryCondition(model,'dirichlet','Face',[1],'u',1.5);
specifyCoefficients(model,'m',0,'d',0,'c',1,'a',0,'f',2);
generateMesh(model);
results = solvepde(model);
u = results.NodalSolution;
h = pdeplot3D(model,'ColorMapData',u);

%h = pdeplot3D(model);

% Assign the parent of the parent of the handle
%h.Parent = app.UIAxes2;

% Assign the parent of the parent of the handle
h(2).Parent.Parent = app.UIAxes2;
%figure=h
view(app.UIAxes2,45,45)
colorbar(app.UIAxes2)
colormap(app.UIAxes2,'jet')
axis(app.UIAxes2,'off')
light('Parent', app.UIAxes2);

% Configurar el tipo de iluminación
lighting(app.UIAxes2, 'gouraud');
shading (app.UIAxes2,'interp')
% Configurar el material
material(app.UIAxes2, 'dull');
%nodes = mesh_default.Nodes';
%deltaPos = [1000 0 50];
%nodesTranslated = nodes + deltaPos;
%elements = mesh_default.Elements';


% Convertir la matriz de coordenadas nodales en un objeto de nube
de puntos
%ptCloud = pointCloud(nodesTranslated);

% Generar una triangulación de Delaunay a partir de los puntos
%triangulation = delaunayTriangulation(ptCloud.Location);

% Crear una malla a partir de la triangulación
%mesh = generateMesh(triangulation.Points,
triangulation.ConnectivityList, 'GeometricOrder', 'linear');

% Visualizar la malla generada a partir de los vértices
```

```
%hold on
%patch('Faces', triangulation.ConnectivityList, 'Vertices',
triangulation.Points, 'FaceColor', 'cyan', 'EdgeColor', 'k');
%trimesh(triangulation.ConnectivityList, triangulation.Points(:,1),
triangulation.Points(:,2), triangulation.Points(:,3), 'EdgeColor',
'none', 'FaceColor', 'interp');
%trimesh(elements, nodes(:,1), nodes(:,2), nodes(:,3), 'EdgeColor',
'k', 'FaceColor', 'cyan','EdgeAlpha',0.5,'Parent',app.UIAxes2);
daspect(app.UIAxes2,[1 1 1])
end
% Callback function
function ShowButton_13Pushed(app, event)
end

% Button pushed function: OnButton
function OnButtonPushed(app, event)
earth = imread('wood3.jpg');
clouds = imread('wood3.jpg');
[px,py,pz] = sphere(50);

% Crear las superficies en el UIAxes de App Designer
sEarth = surface(app.UIAxes2_2, py, px ,flip(pz));
sEarth.FaceColor = 'texturemap';
sEarth.EdgeColor = 'none';
sEarth.CData = earth;
hold(app.UIAxes2_2, 'on')
sCloud = surface(app.UIAxes2_2, px*1.02, py*1.02, flip(pz)*1.02);
sCloud.FaceColor = 'texturemap';
sCloud.EdgeColor = 'none';
sCloud.CData = clouds;

sCloud.FaceAlpha = 'texturemap';
sCloud.AlphaData = max(clouds,[],3);
hold(app.UIAxes2_2, 'off')

% Configurar las propiedades de visualización del UIAxes
view(app.UIAxes2_2, [80 2])
daspect(app.UIAxes2_2, [1 1 1])
axis(app.UIAxes2_2, 'off', 'tight')
axis(app.UIAxes2_2, 'equal')
% Crear una luz
light('Parent', app.UIAxes2_2);

% Configurar el tipo de iluminación
lighting(app.UIAxes2_2, 'gouraud');
% Configurar el material
material(app.UIAxes2_2, 'dull');
```

```
end

% Button pushed function: Button_15
function Button_15Pushed(app, event)
structuralmodel = createpde("structural","static-solid");
L = 0.1; % m
W = 5e-3; % m
H = 1e-3; % m
gm = multicuboid(L,W,[H,H],"Zoffset",[0,H]);
structuralmodel.Geometry = gm;

%figure
%pdegplot(structuralmodel)
Ec = 137e9; % N/m^2
nuc = 0.28;
CTEc = 20.00e-6; % m/m-C
%Assign the material properties of copper to the bottom cell.
structuralProperties(structuralmodel,"Cell",1, ...
"YoungsModulus",Ec, ...
"PoissonsRatio",nuc, ...
"CTE",CTEc);
%Assign the material properties of invar to the top cell.
Ei = 130e9; % N/m^2
nui = 0.354;
CTEi = 1.2e-6; % m/m-C
structuralProperties(structuralmodel,"Cell",2, ...
"YoungsModulus",Ei, ...
"PoissonsRatio",nui, ...
"CTE",CTEi);
%Apply a fixed boundary condition on faces 5 and 10
structuralBC(structuralmodel,"Face",[5,10],"Constraint","fixed");
structuralBodyLoad(structuralmodel,"Temperature",125);
structuralmodel.ReferenceTemperature = 25;
generateMesh(structuralmodel,"Hmax",H/2);
R = solve(structuralmodel);
h=pdeplot3D(structuralmodel,"ColorMapData",R.Displacement.Magnitude
, ...
"Deformation",R.Displacement, ...
"DeformationScaleFactor",2)
title("Deflexion viga madera")

%h = pdeplot3D(model);

% Assign the parent of the parent of the handle
%h.Parent = app.UIAxes_11;

% Assign the parent of the parent of the handle
h(2).Parent.Parent = app.UIAxes_11;
```

```
%figure=h
view(app.UIAxes_11,45,45)
title(app.UIAxes_11,"Deflexion viga madera")
colorbar(app.UIAxes_11)
colormap(app.UIAxes_11,'jet')
axis(app.UIAxes_11,'off')
light('Parent', app.UIAxes_11);

% Configurar el tipo de iluminación
lighting(app.UIAxes_11, 'gouraud');
shading (app.UIAxes_11,'interp')
% Configurar el material
material(app.UIAxes_11, 'dull');
daspect(app.UIAxes_11,[1 1 1])
end

% Button pushed function: GenerateButton
function GenerateButtonPushed(app, event)
nelx = app.NelementosXEditField.Value;
nely = app.NelementosYEditField.Value;
P=app.PEditField.Value

%P=100

%aca estan al reves
nelx = 5; % número de elementos en la dirección x
nely =10; % número de elementos en la dirección y
nn = (nelx + 1) * (nely + 1); % número total de nodos
ne = nelx * nely; % número total de elementos
%edof = zeros(ne, 4); % matriz de orden de los edof

%matriz_nodos =
GL_bynodo=2
%_____________________________________________________________________
_____
node_inds = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);; %indice
nodal %este es el unico cambio realizado de momento
%_____________________________________________________________________
_____

elem_inds = reshape(1:ne, [nelx, nely])'; %indice de cada elemento
num_local =reshape(1:GL_bynodo*nn, [GL_bynodo, nn])'

%_____________________________________________________________________
_____
num_local=[num_local(:,2) num_local(:,1)]
%_____________________________________________________________________
____
```

```
x_nod = 0:nelx;
y_nod = 0:nely;

%_____________________________________________________________________
____
[Y_nod,X_nod] = meshgrid(y_nod ,x_nod)
%_____________________________________________________________________
____
%[X_nod,Y_nod] = meshgrid(x_nod ,y_nod)
%ind_coord = reshape(node_inds, 1, numel(node_inds));

%matrix_nodos = [X_nod(ind_coord);Y_nod(ind_coord)]

% Recorrer todos los elementos en la malla
for elx = 1:nelx
for ely = 1:nely
% Obtener los índices de los nodos que conforman el elemento actual
%n1 = (nely + 1) * (ex - 1) + 1;
%n2 = n1 + 1;
%n3 = n1 - (nely + 1);
%n4 = n3+1;
%n1 = node_inds(ely, elx);
%n2 = node_inds(ely, elx+1);
%n3 = node_inds(ely+1, elx+1);
%n4 = node_inds(ely+1, elx);
n1 = node_inds(ely, elx);
n2 = node_inds(ely, elx+1);
n3 = node_inds(ely+1, elx+1);
n4 = node_inds(ely+1, elx);
e = (ely - 1) * nelx + elx;
connectivity(e, :)=[num_local(n1,:) num_local(n2,:) num_local(n3,:)
num_local(n4,:)]
% connectivity(e, :)=[n1 n2 n3 n4]
% efof(e, :)
edof(e, :) =[num_local(n1,:) num_local(n2,:) num_local(n3,:)
num_local(n4,:)]
% EDof((ely-1)*nelx+elx,:) = [x(elx) y(ely) n1 x(elx+1) y(ely) n2
...
% x(elx+1) y(ely+1) n3 x(elx) y(ely+1) n4];
% Calcular el orden de los edof para el elemento actual y
almacenarlo en la matriz de orden de los edof
%e = (ely - 1) * nelx + elx;
%edof(e, :) = [n1, n2, n3, n4];
end
end
%edof
```

```
%ok_____________________________________________________________________
____

ind_coord = reshape(connectivity', 1, []);
vector=reshape(connectivity,4,[])
matrix_nodos=[]
matrix_nodos_n=[]
FF=[]

%matrix_nodos_n=reshape(matrix_nodos_n,[],2)
% Inicializar la lista vacía
coords = {};
coords_m=[]
X_nod=X_nod'
Y_nod=Y_nod'
%sucede que lo recorre de arriba hacia abajo estos comandos numel y
%reshape por lo que debemos trasponerla, ademas estamos usando
notación indicial lineal en la matriz para
%recorrerla, por eso solo usamos un solo indice i

% Iterar sobre cada par de coordenadas y agregarlo a la lista
for i = 1:numel(X_nod)
coords{i} = [X_nod(i) Y_nod(i)];
Koords{i} = [X_nod(i) Y_nod(i)];
%coords_m = [coords_m;X_nod(i), Y_nod(i)];
end

% Duplicar los elementos de la matriz
%_____________________________________________________________
coords = repelem(coords, 1, GL_bynodo)';%aca hay que trasponer % de
aca nace el erro
coords_nodales=cell2mat(coords);
m_nodal=coords(vector);

Connect_element=coords(connectivity);


%Reshap_Connect_element=cell2mat(reshape(Connect_element,[],1)')
Reshap_Connect_element=cell2mat(Connect_element)
Lineal_connect_element = reshape(Connect_element',[],1)
Double_Lineal_connect_element=cell2mat(Lineal_connect_element);

AAAA = reshape(coords,[],1)'
% Vectorizar en una sola fila
C = reshape(m_nodal',1,[])'
coords_m = cell2mat(C)

% Definir las coordenadas x e y de los nodos
```

```
coords_patch = cell2mat(coords)%acá esta el error
%x = coords_patch(:,1);
%y = coords_patch(:,2);


% Graficar
%_____________________________________________________________________
_____________
%figure(2);
%hold on
%patch('Faces', connectivity, 'Vertices', coords_patch,
'FaceColor', 'none', 'EdgeColor', 'k');
%_____________________________________________________________________
_____________
% Iterar sobre cada figura y mostrar el número de cada una
for i = 1:ne
%n_elem(i)=coords_patch(i,1)+1
n_elem(i)=i;
X_mean(i)=mean([Reshap_Connect_element(i,1)
Reshap_Connect_element(i,3) Reshap_Connect_element(i,5)
Reshap_Connect_element(i,7)]);
Y_mean(i)=mean([Reshap_Connect_element(i,2)
Reshap_Connect_element(i,4) Reshap_Connect_element(i,6)
Reshap_Connect_element(i,8)]);
%x = coords_patch(i, :, 1);
%%y = coords_patch(i, :, 2);
%_____________________________________________________________________
_____________________________
%text(X_mean(i), Y_mean(i), num2str(n_elem(i)),
'HorizontalAlignment', 'center', 'VerticalAlignment', 'middle');
%---------------------------------------------------------------------
---------------------------------
end

x = Double_Lineal_connect_element(:,1)
y = Double_Lineal_connect_element(:,2)
DOF = ind_coord';
cont=0;
c_par=0;
c_impar=0;
%matriz_conectividad = connectivity;
k_arrow=0.6 %tamaño vectores
pos=0.15; %desface esquinas para ejes locales

if GL_bynodo==1
```

```
pos=0.01
end
for i=1:numel(ind_coord')
cont=cont+1;
%si el contador es par le corresponde posición horizontal
if rem(cont,2)==0
c_par=c_par+1;
if c_par==1
quiver(x(i)+pos, y(i)+pos, ones(size(DOF(i)))*k_arrow,
zeros(size(DOF(i))), 0.5, 'LineWidth', 1, 'Color', 'r'); %ok
text(x(i) +2*pos, y(i) +pos/2, num2str(DOF(i)), 'Color', 'r'); %ok
%esq inf izq horizontal
elseif c_par==2
quiver(x(i)-pos, y(i) +pos, -1*ones(size(DOF(i)))*k_arrow,
zeros(size(DOF(i))), 0.5, 'LineWidth', 1, 'Color', 'r');
text(x(i) -2*pos, y(i) +pos/2, num2str(DOF(i)), 'Color', 'r');
%esq ind der horizontal
elseif c_par==3
quiver(x(i)-pos, y(i)-pos, -1*ones(size(DOF(i)))*k_arrow,
zeros(size(DOF(i))), 0.5, 'LineWidth', 1, 'Color', 'r');
text(x(i) -2*pos, y(i) -pos/2, num2str(DOF(i)), 'Color', 'r');
%esq sup der horizontal
elseif c_par==4
quiver(x(i)+pos, y(i)-pos, ones(size(DOF(i)))*k_arrow,
zeros(size(DOF(i))), 0.5, 'LineWidth', 1, 'Color', 'r');
text(x(i) +2*pos, y(i) -pos/2, num2str(DOF(i)), 'Color', 'r');
%esq sup izq horizontal
end
if c_par==GL_bynodo*2
c_par=0;
end
end
%si el contador es impar le corresponde posición vertical
if rem(cont,2)==1
c_impar=c_impar+1;
if c_impar==1
quiver(x(i)+pos, y(i)+pos, zeros(size(DOF(i))),
ones(size(DOF(i)))*k_arrow, 0.5, 'LineWidth', 1, 'Color', 'b');
text(x(i) +pos, y(i) +2*pos, num2str(DOF(i)), 'Color', 'b');
%esq inf izq vertical
elseif c_impar==2
quiver(x(i)-pos, y(i) +pos, zeros(size(DOF(i))),
ones(size(DOF(i)))*k_arrow, 0.5, 'LineWidth', 1, 'Color', 'b');
text(x(i) -pos, y(i) +2*pos, num2str(DOF(i)), 'Color', 'b');
%esq ind der vertical
elseif c_impar==3
quiver(x(i)-pos, y(i)-pos, zeros(size(DOF(i))), -
ones(size(DOF(i)))*k_arrow, 0.5, 'LineWidth', 1, 'Color', 'b');
```

```
text(x(i) -pos, y(i) -2*pos, num2str(DOF(i)), 'Color', 'b');
%esq sup der vertical
elseif c_impar==4
quiver(x(i)+pos, y(i)-pos, zeros(size(DOF(i))), -
ones(size(DOF(i)))*k_arrow, 0.5, 'LineWidth', 1, 'Color', 'b');
text(x(i) +pos, y(i) -2*pos, num2str(DOF(i)), 'Color', 'b');
%esq sup izq vertical
end
if c_impar==GL_bynodo*2
c_impar=0;
end
end
% text(x(i) + 0.2, y(i) + 0.2, num2str(DOF_x(i)), 'Color', 'r');
% text(x(i) - 0.2, y(i) + 0.2, num2str(DOF_y(i)), 'Color', 'b');
end
%quiver(x, y, ones(size(DOF)), zeros(size(DOF)), 0.5, 'LineWidth',
1, 'Color', 'r');
%quiver(x, y, ones(size(DOF)), zeros(size(DOF)), 0.5, 'LineWidth',
1, 'Color', 'r');
hold off
%ind_coord' %numero locl de la conectividad
%Double_Lineal_connect_element %coordenadda de la conectividad



% DEFINE LOADS AND SUPPORTS (Cantilever)
if rem(nely,2)==0
F((2*(nelx+1)*(nely+1)-nely),1)=-P;
end
if rem(nely,2)==1
F((2*(nelx+1)*(nely+1)-nely-1),1)=-P/2;
F((2*(nelx+1)*(nely+1)-nely+1),1)=-P/2;
end
% Definir las coordenadas de los nodos
x = linspace(0,(nelx+1),nelx+2);
y = linspace(0,(nely+1),nely+2);

% Definir el vector de conectividad de las fuerzas
F_connect = [121]; %ingresar como vector fila

F_Connect=coords(F_connect);
F_Lineal_connect_element = reshape(F_Connect',[],1)
F_Double_Lineal_connect_element=cell2mat(F_Lineal_connect_element);
Fx = F_Double_Lineal_connect_element(:,1)
Fy = F_Double_Lineal_connect_element(:,2)

%F_ind_coord = reshape(connectivity', 1, []);
%_____________________________
```

```
F_ind_coord =ind_coord
%_____________________________


F_DOF = F_ind_coord';
%DOF = ind_coord';
%escribirlo como funcion
global x_nodal y_nodal

x_nodal=coords_nodales(:,1);
y_nodal=coords_nodales(:,2);

fixeddofs=[1:2*(nely+1)]; %aca cambie nely por nelx
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);

hold
axes(app.UIAxes_5);
hold on;
hold(app.UIAxes_5,'on');


patch('Faces', connectivity, 'Vertices', coords_patch, 'FaceColor',
'none', 'EdgeColor', 'k','Parent',app.UIAxes_5);
%quiver(x_nodal(F), y_nodal(F), zeros(size(F)), -ones(size(F)),
0.5, 'LineWidth', 1, 'Color', 'b');
% text(x_nodal(F), y_nodal(F) , num2str(F), 'Color', 'b');
scatter(x_nodal(fixeddofs),y_nodal(fixeddofs),100,"^",'filled','bla
ck','Parent',app.UIAxes_5);
scatter(x_nodal(freedofs),y_nodal(freedofs),50,'o','black','Parent'
,app.UIAxes_5);


%P=100%es un vector de fuerzas?
%if P(i)>=0
%k_force=1
%else
k_force=1
%end %crear vector de orientacion de -1 y 1 de la misma longitud
que P
cont=0
for i=1:numel(F_connect)
cont=cont+1;
%si el contador es par le corresponde posición horizontal
if rem(F_connect(i),2)==0
%c_par=c_par+1;
```

```
quiver(Fx(i), Fy(i), ones(size(F_DOF(i)))*0.25*k_force,
zeros(size(F_DOF(i))), 2.5, 'LineWidth', 1, 'Color',
'r','Parent',app.UIAxes_5); %ok
text(Fx(i) +0.08, Fy(i) +0.1, num2str(P), 'Color',
'r','Parent',app.UIAxes_5); %ok

elseif rem(F_connect(i),2)==1
%c_impar=c_impar+1;
quiver(Fx(i), Fy(i), zeros(size(F_DOF(i))),
ones(size(F_DOF(i)))*0.25*k_force, 2.5, 'LineWidth', 1, 'Color',
'b','Parent',app.UIAxes_5);
text(Fx(i) +0.08, Fy(i) -0.1, num2str(P), 'Color',
'b','Parent',app.UIAxes_5);
else
end
end
%en Rem debe ir la asignacion local, no el contador ,si el
conectivity
OOO=rem(67,2)

axis equal;

hold off
daspect(app.UIAxes_5,[1 1 1])

%ALMACENAMIENTO DE CAMBIOS EFECTUADOS
global cambio cont_cambios;
cont_cambios=0;
cambio = {};
end

% Button pushed function: SelecLine
function SelecLineButtonPushed(app, event)
global x_nodal y_nodal XY_selected
ax=app.UIAxes_5; %plot(ax,rand(1,100))
[x,y] = ginputuiax(ax,2)

% Encontrar los nodos que están dentro del rectángulo de selección
selected_nodes = [];
for i = 1:length(x_nodal)
if x_nodal(i) >= min([x(1), x(2)]) && x_nodal(i) <= max([x(1),
x(2)]) && ...
y_nodal(i) >= min([y(1), y(2)]) && y_nodal(i) <= max([y(1), y(2)])
selected_nodes = [selected_nodes i];
end
end
XY_selected=[x_nodal(selected_nodes), y_nodal(selected_nodes)];
```

```
scatter(x_nodal(selected_nodes),
y_nodal(selected_nodes),50,'o','filled','Parent',app.UIAxes_5);

end

% Button pushed function: NoRestrain
function NoRestrainButtonPushed(app, event)
global XY_selected
scatter(XY_selected(:,1),XY_selected(:,2),70,'o','MarkerEdgeColor',
'w','MarkerFaceColor','w','Parent',app.UIAxes_5);
scatter(XY_selected(:,1),XY_selected(:,2),50,'o','MarkerEdgeColor',
'black','MarkerFaceColor','w','Parent',app.UIAxes_5);
registrarCambio('NoRestrainButtonPushed', XY_selected,[]);

app.PEditField.Value = 0;
end

% Button pushed function: xRestrainDeslizante
function xRestrainDeslizanteButtonPushed(app, event)
global XY_selected
scatter(XY_selected(:,1),XY_selected(:,2),70,'o','MarkerEdgeColor',
'w','MarkerFaceColor','w','Parent',app.UIAxes_5);
scatter(XY_selected(:,1),XY_selected(:,2),50,">",'filled','red','Pa
rent',app.UIAxes_5);
registrarCambio('xRestrainDeslizanteButtonPushed', XY_selected,[]);

app.PEditField.Value = 0;
end

% Button pushed function: yRestrainDeslizante
function yRestrainDeslizanteButtonPushed(app, event)
global XY_selected
scatter(XY_selected(:,1),XY_selected(:,2),70,'o','MarkerEdgeColor',
'w','MarkerFaceColor','w','Parent',app.UIAxes_5);
scatter(XY_selected(:,1),XY_selected(:,2),50,"^",'filled','b','Pare
nt',app.UIAxes_5);
registrarCambio('yRestrainDeslizanteButtonPushed', XY_selected,[]);

app.PEditField.Value = 0;
end

% Button pushed function: RestrainFixed
function RestrainFixedButtonPushed(app, event)
global XY_selected
scatter(XY_selected(:,1),XY_selected(:,2),70,'o','MarkerEdgeColor',
'w','MarkerFaceColor','w','Parent',app.UIAxes_5);
scatter(XY_selected(:,1),XY_selected(:,2),60,"^",'filled','k','Pare
nt',app.UIAxes_5);
```

```
registrarCambio('RestrainFixedButtonPushed', XY_selected,[]);
app.PEditField.Value = 0;
end

% Button pushed function: RestrainEmpotred
function RestrainEmpotredButtonPushed(app, event)
global XY_selected
scatter(XY_selected(:,1),XY_selected(:,2),70,'o','MarkerEdgeColor',
'w','MarkerFaceColor','w','Parent',app.UIAxes_5);
scatter(XY_selected(:,1),XY_selected(:,2),70,"square",'filled','k',
'Parent',app.UIAxes_5);
registrarCambio('RestrainEmpotredButtonPushed', XY_selected,[]);

app.PEditField.Value = 0;
end

% Button pushed function: xButton_2
function xButton_2Pushed(app, event)
global XY_selected value_MagnitudeP vectorF
value_MagnitudeP = app.PEditField.Value;
vectorF = ones(length(XY_selected(:,2)), 1)*value_MagnitudeP
signo=(abs(value_MagnitudeP)/value_MagnitudeP)
quiver(XY_selected(:,1), XY_selected(:,2),
signo*ones(size(XY_selected(:,2)))*0.6,
zeros(size(XY_selected(:,2))), 0.5/1.5, 'LineWidth', 1.0, 'Color',
'r','Parent',app.UIAxes_5);

% Obtén las coordenadas para colocar el texto al lado de la flecha
xText = XY_selected(:,1) + 0.62*signo; % Ajusta la posición en x
según tus necesidades
yText = XY_selected(:,2) + 0.1; % Ajusta la posición en y según tus
necesidades

% Crea un texto con el valor_MagnitudeP
text(xText, yText, num2str(value_MagnitudeP), 'Color', 'r',
'FontSize', 10, 'Parent', app.UIAxes_5);
registrarCambio('xButton_2Pushed', XY_selected, vectorF);

app.PEditField.Value = 0;
end

% Button pushed function: yButton_2
function yButton_2Pushed(app, event)
global XY_selected value_MagnitudeP vectorF
value_MagnitudeP = app.PEditField.Value;
vectorF = ones(length(XY_selected(:,2)), 1)*value_MagnitudeP
signo=(abs(value_MagnitudeP)/value_MagnitudeP)
```

```
quiver(XY_selected(:,1), XY_selected(:,2),
zeros(size(XY_selected(:,2))),signo*ones(size(XY_selected(:,2)))*0.
6, 0.5/1.5, 'LineWidth', 1.0, 'Color', 'b','Parent',app.UIAxes_5);

% Obtén las coordenadas para colocar el texto al lado de la flecha
xText = XY_selected(:,1)+0.1; % Ajusta la posición en x según tus
necesidades
yText = XY_selected(:,2)+ 0.62*signo; % Ajusta la posición en y
según tus necesidades

% Crea un texto con el valor_MagnitudeP
text(xText, yText, num2str(value_MagnitudeP), 'Color', 'b',
'FontSize', 10, 'Parent', app.UIAxes_5);
registrarCambio('yButton_2Pushed', XY_selected,vectorF);

app.PEditField.Value = 0;
end

% Button pushed function: circle
function circleButtonPushed(app, event)
global x_nodal y_nodal XY_selected
ax=app.UIAxes_5; %plot(ax,rand(1,100))
[x,y] = ginputcircleuiax(ax,2)

% Encontrar los nodos que están dentro del rectángulo de selección
[x_center, y_center, radius] = deal(x(1), y(1), norm([x(2)-x(1),
y(2)-y(1)]));

selected_nodes = [];
selected_nodes = inpolygon(x_nodal, y_nodal, x_center +
radius*cos(linspace(0, 2*pi)), y_center + radius*sin(linspace(0,
2*pi)));

for i = 1:length(x_nodal)
%if x_nodal(i) >= min([x(1), x(2)]) && x_nodal(i) <= max([x(1),
x(2)]) && ...
% y_nodal(i) >= min([y(1), y(2)]) && y_nodal(i) <= max([y(1),
y(2)])

% Encontrar los nodos que están dentro del círculo de selección
%selected_nodes = inpolygon(x_nodal, y_nodal, x_center +
radius*cos(linspace(0, 2*pi)), y_center + radius*sin(linspace(0,
2*pi)));

%selected_nodes = [selected_nodes i];
%end
end
XY_selected=[x_nodal(selected_nodes), y_nodal(selected_nodes)];
```

```
scatter(x_nodal(selected_nodes),
y_nodal(selected_nodes),50,'o','filled','Parent',app.UIAxes_5);

app.PEditField.Value = 0;
end

% Button pushed function: image
function imagePushed(app, event)
% Definición del radio R y el ángulo central ?

% Definir las dimensiones del canal circular
R = 50; % radio del canal circular
L1 = 20; % longitud de la base del triángulo inscrito en el círculo

% Calcular el ángulo de apertura del canal circular
theta1 = atan(L1 / (2 * R));

ntableros=6
S=8%
%R = 10; %[mts]
t=2.5; %[mts]
theta = linspace(-theta1*pi, theta1*pi, 500);
%theta = linspace(0, pi, 500);
ancho=0.8;
nCS=3
nBarras=8
% Cálculo de las coordenadas cartesianas x e y
x1 = R * cos(theta);
y1 = R * (sin(theta));
x2 = (R-t) * cos(theta);
y2 = (R-t) * (sin(theta));

%ALGORITMO Posicion BARRAS
x3 = 0.5*(2*R-t) * cos(theta);
y3 = 0.5*(2*R-t) * (sin(theta));
ind_pos = round(linspace(1, length(x3), (nBarras+2)))
ind_pos(1)=[]
ind_pos(length(ind_pos))=[]
x_barras=x3(ind_pos)
y_barras=y3(ind_pos)
%ALGORITMO QUIVER FUERZAS
%theta_force = linspace(0, pi, 20);
theta_force = linspace(-theta1*pi, theta1*pi, 20);
x_force = R * cos(theta_force);
y_force = R * (sin(theta_force));
nNodes_force=length(x_force(1,:));
z_force=linspace(-ancho/2,ancho/2,nCS)
X_force=repmat(x_force,nCS,1);
```

```
Y_force=repmat(y_force,nCS,1);
Z_force=repmat(z_force',1,nNodes_force);
%_____________________________________________________

%Algoritmo Barras cilindricas
Rb=0.2
L=S*(ntableros-1)
nCS=10
nNodes=30

r=Rb*ones(1,nNodes);
th= linspace(0,2*pi,nNodes);
z=linspace(0,L,nCS)';
[x,y]=pol2cart(th,r);
X_c=repmat(x,nCS,1);
Y_c=repmat(y,nCS,1);
Z_c=repmat(z,1,nNodes);

x_lid=zeros(2,nNodes)
y_lid=zeros(2,nNodes)
z_lid=repmat([0;L],1,nNodes);

X_c=[x_lid(1,:);X_c;x_lid(2,:)];
Y_c=[y_lid(1,:);Y_c;y_lid(2,:)];
Z_c=[z_lid(1,:);Z_c;z_lid(2,:)];
%_____________________________________________________________

x=[x2 fliplr(x1) x2(1)]'
y=[y2 fliplr(y1) y2(1)]
nNodes=length(x(:,1));
z=linspace(-ancho/2,ancho/2,nCS)
X=repmat(x',nCS,1);
Y=repmat(y',nCS,1);
Z=repmat(z',1,nNodes);
u=ones(1,nNodes)
%X=[ones(1,nNodes); X; ones(1,nNodes)]
%Y=[zeros(1,nNodes); Y; ones(1,nNodes)]
%Z=[ones(1,nNodes)*NaN; Z; ones(1,nNodes)*ancho/2]
% Definir los valores de x, y y z
[X_sup,Y_sup] = meshgrid(linspace(min(x1), max(x1), length(x1)),
linspace(min(y1), max(y1), length(y1)));
%[X_sup,Y_sup] = meshgrid(linspace(min(x1), max(x1), length(x1)),
linspace(0, 10, length(x1)));
Z_sup = ones(size(X_sup))*ancho/2;

% Definir las coordenadas del polígono
x_poly = [x2 fliplr(x1) x2(1)];
y_poly = [y2 fliplr(y1) y2(1)];
```

```
% Encontrar los puntos dentro del polígono
points_inside = inpolygon(X_sup, Y_sup, x_poly, y_poly);
X_sup(~points_inside) = NaN;
Y_sup(~points_inside) = NaN;
Z_sup(~points_inside) = NaN;
% Hacer cero los puntos por encima de la función
%Y(Y > y1) = NaN;
%X(X > x1) = NaN;
%Y(Y < y2) = NaN;
%X(X > x2) = NaN;
%Y(Y < y2) = NaN;
%X(X < x2) = NaN;

% Graficar el surf
%surf(X, Y, Z);
hold
axes(app.UIAxes3);
hold on;
hold(app.UIAxes3,'on');app.UIAxes

for i=1:ntableros

plate_sup=surf(X_sup,Y_sup,Z_sup+(i-1)*S,'Parent',app.UIAxes3)
textura3 = imread('wood3.jpg');
set(plate_sup, 'FaceColor', 'texturemap', 'CData', textura3);
plate_sup.FaceColor=[168, 121, 79]/255
plate_sup.EdgeAlpha=0
plate_sup=surf(X_sup,Y_sup,(Z_sup-ancho)+(i-
1)*S,'Parent',app.UIAxes3)
%textura3 = imread('wood6.jpg');
set(plate_sup, 'FaceColor', 'texturemap', 'CData', textura3);
plate_sup.FaceColor=[168, 121, 79]/255
plate_sup.EdgeAlpha=0

plate3=surf(X,Y,Z+(i-1)*S,'Parent',app.UIAxes3)
%textura3 = imread('wood6.jpg');
%set(plate3, 'FaceColor', 'texturemap', 'CData', textura3);
plate3.EdgeAlpha=0.3
plate3.FaceColor=[168, 121, 79]/255

%[U,V,W] = surfnorm(X,Y,Z);
%V(:,1)=-1;
%V(:,3)=-1;
%q=quiver3(X,Y,Z+(i-1)*S,U*0,-(V+1),W*0)

[U,V,W] = surfnorm(X_force,Y_force,Z_force,'Parent',app.UIAxes3);
V(1,:)=-1;
```

```
V(2,:)=0;
V(3,:)=-1;
q=quiver3(X_force,Y_force+1.15,Z_force+(i-1)*S,U*0,-
(V+1)*0.1,W*0,'Parent',app.UIAxes_7)
%(X2,Y2,Z2,U.*0,V.*0,-
W,zeros(size(X2)),zeros(size(X2)),ones(size(X2)))
q.MaxHeadSize=0.6
q.ShowArrowHead='on'
q.AutoScaleFactor=0.3
q.Color = 'blue';
q.LineWidth=1.15
end


for k=1:numel(x_barras)
T = [x_barras(k) y_barras(k) 0];
cilindro=surf(X_c+ T(1),Y_c+ T(2),Z_c+ T(3),'Parent',app.UIAxes3)
textura = imread('steel.jpg');
set(cilindro, 'FaceColor', 'texturemap', 'CData', textura);
cilindro.EdgeAlpha=0
cilindro.FaceColor= [0.56 0.57 0.58];
end

view(app.UIAxes3,3)
colorbar(app.UIAxes3)
colormap(app.UIAxes3,'jet')
axis(app.UIAxes3,'off')
light('Parent', app.UIAxes3);
%view(3)
% Configurar el tipo de iluminación
lighting(app.UIAxes3, 'gouraud');
%shading (app.UIAxes3,'interp')
% Configurar el material
material(app.UIAxes3, 'dull');
hold(app.UIAxes3, 'off')

end

% Button pushed function: Button_16
function Button_16Pushed(app, event)
% Definición del radio R y el ángulo central ?

% Definir las dimensiones del canal circular
R = 50; % radio del canal circular
L1 = 20; % longitud de la base del triángulo inscrito en el círculo

% Calcular el ángulo de apertura del canal circular
theta1 = atan(L1 / (2 * R));
```

```
ntableros=6
S=8%
%R = 10; %[mts]
t=2.5; %[mts]
theta = linspace(-theta1*pi, theta1*pi, 500);
%theta = linspace(0, pi, 500);
ancho=0.8;
nCS=3
nBarras=8
% Cálculo de las coordenadas cartesianas x e y
x1 = R * cos(theta);
y1 = R * (sin(theta));
x2 = (R-t) * cos(theta);
y2 = (R-t) * (sin(theta));

%ALGORITMO Posicion BARRAS
x3 = 0.5*(2*R-t) * cos(theta);
y3 = 0.5*(2*R-t) * (sin(theta));
ind_pos = round(linspace(1, length(x3), (nBarras+2)))
ind_pos(1)=[]
ind_pos(end)=[]
x_barras=x3(ind_pos)
y_barras=y3(ind_pos)
%ALGORITMO QUIVER FUERZAS
%theta_force = linspace(0, pi, 20);
theta_force = linspace(-theta1*pi, theta1*pi, 20);
x_force = R * cos(theta_force);
y_force = R * (sin(theta_force));
nNodes_force=length(x_force(1,:));
z_force=linspace(-ancho/2,ancho/2,nCS)
X_force=repmat(x_force,nCS,1);
Y_force=repmat(y_force,nCS,1);
Z_force=repmat(z_force',1,nNodes_force);
%_____________________________________________________

%Algoritmo Barras cilindricas
Rb=0.2
L=S*(ntableros-1)
nCS=10
nNodes=30

r=Rb*ones(1,nNodes);
th= linspace(0,2*pi,nNodes);
z=linspace(0,L,nCS)';
[x,y]=pol2cart(th,r);
X_c=repmat(x,nCS,1);
Y_c=repmat(y,nCS,1);
```

```
Z_c=repmat(z,1,nNodes);

x_lid=zeros(2,nNodes)
y_lid=zeros(2,nNodes)
z_lid=repmat([0;L],1,nNodes);

X_c=[x_lid(1,:);X_c;x_lid(2,:)];
Y_c=[y_lid(1,:);Y_c;y_lid(2,:)];
Z_c=[z_lid(1,:);Z_c;z_lid(2,:)];


%_____________________________________________________________________




x=[x2 fliplr(x1) x2(1)]'
y=[y2 fliplr(y1) y2(1)]'



nNodes=length(x(:,1));
z=linspace(-ancho/2,ancho/2,nCS)

X=repmat(x',nCS,1);
Y=repmat(y',nCS,1);
Z=repmat(z',1,nNodes);
u=ones(1,nNodes)

%X=[ones(1,nNodes); X; ones(1,nNodes)]
%Y=[zeros(1,nNodes); Y; ones(1,nNodes)]
%Z=[ones(1,nNodes)*NaN; Z; ones(1,nNodes)*ancho/2]




% Definir los valores de x, y y z
[X_sup,Y_sup] = meshgrid(linspace(min(x1), max(x1), length(x1)),
linspace(min(y1), max(y1), length(y1)));
%[X_sup,Y_sup] = meshgrid(linspace(min(x1), max(x1), length(x1)),
linspace(0, 10, length(x1)));
Z_sup = ones(size(X_sup))*ancho/2;

% Definir las coordenadas del polígono
x_poly = [x2 fliplr(x1) x2(1)];
y_poly = [y2 fliplr(y1) y2(1)];
```

```
% Encontrar los puntos dentro del polígono
points_inside = inpolygon(X_sup, Y_sup, x_poly, y_poly);

X_sup(~points_inside) = NaN;
Y_sup(~points_inside) = NaN;
Z_sup(~points_inside) = NaN;

% Hacer cero los puntos por encima de la función
%Y(Y > y1) = NaN;
%X(X > x1) = NaN;
%Y(Y < y2) = NaN;
%X(X > x2) = NaN;
%Y(Y < y2) = NaN;
%X(X < x2) = NaN;
% Graficar el surf
%surf(X, Y, Z);

hold
axes(app.UIAxes_9);
hold on;
hold(app.UIAxes_9,'on');


for i=1:ntableros

%plate_sup=surf(X_sup,Y_sup,Z_sup+(i-1)*S,'Parent',app.UIAxes_9)
plate_sup=surf(Z_sup+(i-1)*S,X_sup,Y_sup,'Parent',app.UIAxes_9)
textura3 = imread('wood4.jpg');
plate_sup.FaceColor = 'texturemap';
plate_sup.EdgeColor = 'none';
plate_sup.CData = textura3;
%set(plate_sup, 'FaceColor', 'texturemap', 'CData', textura3);
%plate_sup.FaceColor=[168, 121, 79]/255
%plate_sup.EdgeAlpha=0
plate_sup=surf((Z_sup-ancho)+(i-
1)*S,X_sup,Y_sup,'Parent',app.UIAxes_9)
%textura3 = imread('wood6.jpg');
%set(plate_sup, 'FaceColor', 'texturemap', 'CData', textura3);
plate_sup.FaceColor = 'texturemap';
plate_sup.EdgeColor = 'none';
plate_sup.CData = textura3;
%plate_sup.FaceColor=[168, 121, 79]/255
%plate_sup.EdgeAlpha=0

plate3=surf(Z+(i-1)*S,X,Y,'Parent',app.UIAxes_9)
%textura3 = imread('wood6.jpg');
%set(plate3, 'FaceColor', 'texturemap', 'CData', textura3);
```

```
plate3.FaceColor = 'texturemap';
plate3.EdgeColor = 'none';
plate3.CData = textura3;
%plate3.EdgeAlpha=0.3
%plate3.FaceColor=[168, 121, 79]/255

%[U,V,W] = surfnorm(X,Y,Z);
%V(:,1)=-1;
%V(:,3)=-1;
%q=quiver3(X,Y,Z+(i-1)*S,U*0,-(V+1),W*0)

%[U,V,W] = surfnorm(X_force,Y_force,Z_force,'Parent',app.UIAxes_9);
%V(1,:)=-1;
%V(2,:)=0;
%V(3,:)=-1;
%q=quiver3(X_force,Y_force+1.15,Z_force+(i-1)*S,U*0,-
(V+1)*0.1,W*0,'Parent',app.UIAxes_9)
%(X2,Y2,Z2,U.*0,V.*0,-
W,zeros(size(X2)),zeros(size(X2)),ones(size(X2)))
%q.MaxHeadSize=0.6
%q.ShowArrowHead='on'
%q.AutoScaleFactor=0.3
%q.Color = 'blue';
%q.LineWidth=1.15
end


for k=1:numel(x_barras)
T = [x_barras(k) y_barras(k) 0];
cilindro=surf(Z_c+ T(3),X_c+ T(1),Y_c+ T(2),'Parent',app.UIAxes_9)
textura = imread('steel.jpg');
%set(cilindro, 'FaceColor', 'texturemap', 'CData', textura);
%cilindro.EdgeAlpha=0
cilindro.FaceColor = 'texturemap';
cilindro.EdgeColor = 'none';
cilindro.CData = textura;
%cilindro.FaceColor= [0.56 0.57 0.58];
end

view(app.UIAxes_9,3)
%colorbar(app.UIAxes_9)
%colormap(app.UIAxes_9,'jet')
%axis(app.UIAxes_9,'off')
light('Parent', app.UIAxes_9);
%view(3)
% Configurar el tipo de iluminación
%lighting(app.UIAxes_9, 'phong');
lighting(app.UIAxes_9, 'gouraud');
```

```
%lightangle(app.UIAxes_9,0, 55)
%shading (app.UIAxes_9,'interp')
% Configurar el material
material(app.UIAxes_9, 'dull');
hold(app.UIAxes_9, 'off')

end

% Button pushed function: SelectPolyLine
function SelectPolyLineButtonPushed(app, event)
global x_nodal y_nodal XY_selected
ax=app.UIAxes_5; %plot(ax,rand(1,100))
N = 5; % Número de lados del polígono deseado
[x, y] = ginputpolygonuiax(ax, N);

% Encontrar los nodos que están dentro del poligono de selección

selected_nodes = [];
selected_nodes = inpolygon(x_nodal, y_nodal, x, y);

for i = 1:length(x_nodal)
%if x_nodal(i) >= min([x(1), x(2)]) && x_nodal(i) <= max([x(1),
x(2)]) && ...
% y_nodal(i) >= min([y(1), y(2)]) && y_nodal(i) <= max([y(1),
y(2)])

% Encontrar los nodos que están dentro del círculo de selección
%selected_nodes = inpolygon(x_nodal, y_nodal, x_center +
radius*cos(linspace(0, 2*pi)), y_center + radius*sin(linspace(0,
2*pi)));

%selected_nodes = [selected_nodes i];
%end
end
XY_selected=[x_nodal(selected_nodes), y_nodal(selected_nodes)];
scatter(x_nodal(selected_nodes),
y_nodal(selected_nodes),50,'o','filled','Parent',app.UIAxes_5);

app.PEditField.Value = 0;
end

% Button pushed function: SelectPolyLine_4
function SelectPolyLine_4ButtonPushed(app, event)
global x_nodal y_nodal XY_selected
global x_nodal y_nodal XY_selected
ax=app.UIAxes_5; %plot(ax,rand(1,100))
%[x,y] = ginputuiax(ax,2)
[x,y] = ginputpolycotasuiax(ax,5)
```

```
end

% Button pushed function: ButtonFilter
function ButtonFilterPushed(app, event)
Filterr
% Crear una nueva ventana
%app.SecondWindow = SecondWindowApp;
% Hacer que la nueva ventana sea modal (bloquear ventana principal)
%waitfor(app.filter);
% filter
%RatioClearRuido
%SuavizadoGaussiano
%Umbralizador
%WeightsmoothPerimeter
%WeightsmoothHoles
%ThresholdPerimeter
%ThresholdHoles
%RadioArcoPerimeter
%RadioArcoHoles
%AlphaShape
%Num_points_x
%Num_points_y
end

% Value changed function: PEditField
function PEditFieldValueChanged(app, event)
global value_MagnitudeP
value_MagnitudeP = app.PEditField.Value;
end

% Button pushed function: remove
function removeButtonPushed(app, event)
global XY_selected
scatter(XY_selected(:,1),XY_selected(:,2),70,'o','MarkerEdgeColor',
'w','MarkerFaceColor','w','Parent',app.UIAxes_5);
scatter(XY_selected(:,1),XY_selected(:,2),60,"hexagram",'filled','k
','Parent',app.UIAxes_5);
registrarCambio('removeButtonPushed', XY_selected,[]);
app.PEditField.Value = 0;
end
end

% Component initialization
methods (Access = private)

% Create UIFigure and components
function createComponents(app)
```

```
% Create UIFigure and hide until all components are created
app.UIFigure = uifigure('Visible', 'off');
app.UIFigure.Position = [100 100 699 508];
app.UIFigure.Name = 'MATLAB App';

% Create TabGroup
app.TabGroup = uitabgroup(app.UIFigure);
app.TabGroup.Position = [1 15 699 484];

% Create CreateTab
app.CreateTab = uitab(app.TabGroup);
app.CreateTab.Title = 'Create';
app.CreateTab.BackgroundColor = [0.9412 0.9412 0.9412];
app.CreateTab.ForegroundColor = [0.9412 0.9412 0.9412];

% Create Panel2
app.Panel2 = uipanel(app.CreateTab);
app.Panel2.AutoResizeChildren = 'off';
app.Panel2.TitlePosition = 'centertop';
app.Panel2.Title = 'Datos Entrada ';
app.Panel2.BackgroundColor = [0.9412 0.9412 0.9412];
app.Panel2.Position = [3 64 101 238];

% Create NelementosYEditFieldLabel
app.NelementosYEditFieldLabel = uilabel(app.Panel2);
app.NelementosYEditFieldLabel.BackgroundColor = [0.9412 0.9412
0.9412];
app.NelementosYEditFieldLabel.HorizontalAlignment = 'right';
app.NelementosYEditFieldLabel.Position = [22 147 86 22];
app.NelementosYEditFieldLabel.Text = 'N°elementos Y';

% Create NelementosYEditField
app.NelementosYEditField = uieditfield(app.Panel2, 'numeric');
app.NelementosYEditField.BackgroundColor = [0.9412 0.9412 0.9412];
app.NelementosYEditField.Position = [21 126 100 22];

% Create EEditFieldLabel
app.EEditFieldLabel = uilabel(app.Panel2);
app.EEditFieldLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.EEditFieldLabel.HorizontalAlignment = 'right';
app.EEditFieldLabel.Position = [9 90 25 22];
app.EEditFieldLabel.Text = 'E';

% Create EEditField
app.EEditField = uieditfield(app.Panel2, 'numeric');
app.EEditField.BackgroundColor = [0.9412 0.9412 0.9412];
app.EEditField.Position = [47 89 61 22];
```

```
% Create uEditFieldLabel
app.uEditFieldLabel = uilabel(app.Panel2);
app.uEditFieldLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.uEditFieldLabel.HorizontalAlignment = 'right';
app.uEditFieldLabel.Position = [10 69 25 22];
app.uEditFieldLabel.Text = 'u';

% Create uEditField
app.uEditField = uieditfield(app.Panel2, 'numeric');
app.uEditField.BackgroundColor = [0.9412 0.9412 0.9412];
app.uEditField.Position = [47 66 61 22];

% Create FV_0EditFieldLabel
app.FV_0EditFieldLabel = uilabel(app.Panel2);
app.FV_0EditFieldLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.FV_0EditFieldLabel.HorizontalAlignment = 'right';
app.FV_0EditFieldLabel.Position = [6 42 34 22];
app.FV_0EditFieldLabel.Text = 'FV_0';

% Create FV_0EditField
app.FV_0EditField = uieditfield(app.Panel2, 'numeric');
app.FV_0EditField.BackgroundColor = [0.9412 0.9412 0.9412];
app.FV_0EditField.Position = [47 42 61 22];

% Create R_minEditFieldLabel
app.R_minEditFieldLabel = uilabel(app.Panel2);
app.R_minEditFieldLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.R_minEditFieldLabel.HorizontalAlignment = 'right';
app.R_minEditFieldLabel.Position = [5 18 40 22];
app.R_minEditFieldLabel.Text = 'R_min';

% Create R_minEditField
app.R_minEditField = uieditfield(app.Panel2, 'numeric');
app.R_minEditField.BackgroundColor = [0.9412 0.9412 0.9412];
app.R_minEditField.Position = [48 18 60 22];

% Create NelementosXEditFieldLabel
app.NelementosXEditFieldLabel = uilabel(app.Panel2);
app.NelementosXEditFieldLabel.BackgroundColor = [0.9412 0.9412
0.9412];
app.NelementosXEditFieldLabel.HorizontalAlignment = 'right';
app.NelementosXEditFieldLabel.Position = [23 193 86 22];
app.NelementosXEditFieldLabel.Text = 'N°elementos X';

% Create NelementosXEditField
app.NelementosXEditField = uieditfield(app.Panel2, 'numeric');
app.NelementosXEditField.BackgroundColor = [0.9412 0.9412 0.9412];
app.NelementosXEditField.Position = [21 174 100 22];
```

```
% Create Panel2_2
app.Panel2_2 = uipanel(app.CreateTab);
app.Panel2_2.AutoResizeChildren = 'off';
app.Panel2_2.TitlePosition = 'centertop';
app.Panel2_2.Title = 'Estrategia de Optimización';
app.Panel2_2.BackgroundColor = [0.9412 0.9412 0.9412];
app.Panel2_2.Position = [2 310 101 103];

% Create AlgoritmoDropDownLabel
app.AlgoritmoDropDownLabel = uilabel(app.Panel2_2);
app.AlgoritmoDropDownLabel.BackgroundColor = [0 0 0];
app.AlgoritmoDropDownLabel.HorizontalAlignment = 'center';
app.AlgoritmoDropDownLabel.FontColor = [1 1 1];
app.AlgoritmoDropDownLabel.Position = [4 56 89 22];
app.AlgoritmoDropDownLabel.Text = 'Algoritmo';

% Create AlgoritmoDropDown
app.AlgoritmoDropDown = uidropdown(app.Panel2_2);
app.AlgoritmoDropDown.Items = {'SIMP', 'TOP99', 'TOP88', 'Softkill
Beso'};
app.AlgoritmoDropDown.FontColor = [1 1 1];
app.AlgoritmoDropDown.BackgroundColor = [0 0 0];
app.AlgoritmoDropDown.Position = [95 56 54 22];
app.AlgoritmoDropDown.Value = 'Softkill Beso';

% Create ProblemaDropDownLabel
app.ProblemaDropDownLabel = uilabel(app.Panel2_2);
app.ProblemaDropDownLabel.BackgroundColor = [0 0 0];
app.ProblemaDropDownLabel.HorizontalAlignment = 'center';
app.ProblemaDropDownLabel.FontColor = [1 1 1];
app.ProblemaDropDownLabel.Position = [4 24 89 22];
app.ProblemaDropDownLabel.Text = 'Problema';

% Create ProblemaDropDown
app.ProblemaDropDown = uidropdown(app.Panel2_2);
app.ProblemaDropDown.Items = {'1', '2', '3', '4', 'Otro'};
app.ProblemaDropDown.FontColor = [1 1 1];
app.ProblemaDropDown.BackgroundColor = [0 0 0];
app.ProblemaDropDown.Position = [96 24 53 22];
app.ProblemaDropDown.Value = '1';

% Create OptimizacinTopologicaLabel
app.OptimizacinTopologicaLabel = uilabel(app.CreateTab);
app.OptimizacinTopologicaLabel.HorizontalAlignment = 'center';
app.OptimizacinTopologicaLabel.FontName = 'OCR A Extended';
app.OptimizacinTopologicaLabel.FontSize = 15;
app.OptimizacinTopologicaLabel.FontWeight = 'bold';
```

```
app.OptimizacinTopologicaLabel.Position = [3 423 98 22];
app.OptimizacinTopologicaLabel.Text = 'Optimización Topologica';

% Create TabGroup2_2
app.TabGroup2_2 = uitabgroup(app.CreateTab);
app.TabGroup2_2.Position = [126 22 572 419];

% Create DesingTab
app.DesingTab = uitab(app.TabGroup2_2);
app.DesingTab.Title = 'Desing';

% Create UIAxes_5
app.UIAxes_5 = uiaxes(app.DesingTab);
title(app.UIAxes_5, '')
xlabel(app.UIAxes_5, '')
ylabel(app.UIAxes_5, '')
app.UIAxes_5.PlotBoxAspectRatio = [1.8 1 1];
app.UIAxes_5.YTick = [0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1];
app.UIAxes_5.Position = [6 -4 538 365];

% Create Button_11
app.Button_11 = uibutton(app.DesingTab, 'push');
app.Button_11.Icon = 'run.jpg';
app.Button_11.BackgroundColor = [0.9686 0.9608 0.9686];
app.Button_11.Position = [507 362 72 22];
app.Button_11.Text = '';

% Create ForceCheckBox
app.ForceCheckBox = uicheckbox(app.DesingTab);
app.ForceCheckBox.Text = 'Force';
app.ForceCheckBox.FontSize = 8;
app.ForceCheckBox.Position = [543 305 42 22];

% Create LocalAxisCheckBox
app.LocalAxisCheckBox = uicheckbox(app.DesingTab);
app.LocalAxisCheckBox.Text = {'Local'; ' Axis'};
app.LocalAxisCheckBox.FontSize = 8;
app.LocalAxisCheckBox.Position = [543 326 42 28];

% Create GrideCheckBox
app.GrideCheckBox = uicheckbox(app.DesingTab);
app.GrideCheckBox.Text = 'Gride';
app.GrideCheckBox.FontSize = 8;
app.GrideCheckBox.Position = [543 284 39 22];

% Create RestrainCheckBox
app.RestrainCheckBox = uicheckbox(app.DesingTab);
app.RestrainCheckBox.Text = 'Restrain';
```

```
app.RestrainCheckBox.FontSize = 8;
app.RestrainCheckBox.Position = [543 263 49 22];

% Create Button
app.Button = uibutton(app.DesingTab, 'push');
app.Button.Icon = 'button_additive.jpg';
app.Button.Position = [19 363 23 22];
app.Button.Text = '';

% Create remove
app.remove = uibutton(app.DesingTab, 'push');
app.remove.ButtonPushedFcn = createCallbackFcn(app,
@removeButtonPushed, true);
app.remove.Icon = 'button_remove.jpg';
app.remove.Position = [41 363 23 22];
app.remove.Text = '';

% Create NoRestrain
app.NoRestrain = uibutton(app.DesingTab, 'push');
app.NoRestrain.ButtonPushedFcn = createCallbackFcn(app,
@NoRestrainButtonPushed, true);
app.NoRestrain.Icon = 'no restrain.jpg';
app.NoRestrain.BackgroundColor = [0.9412 0.9412 0.9412];
app.NoRestrain.Position = [78 362 23 22];
app.NoRestrain.Text = '';

% Create xRestrainDeslizante
app.xRestrainDeslizante = uibutton(app.DesingTab, 'push');
app.xRestrainDeslizante.ButtonPushedFcn = createCallbackFcn(app,
@xRestrainDeslizanteButtonPushed, true);
app.xRestrainDeslizante.Icon = 'deslizante.jpg';
app.xRestrainDeslizante.IconAlignment = 'center';
app.xRestrainDeslizante.HorizontalAlignment = 'left';
app.xRestrainDeslizante.BackgroundColor = [0.9412 0.9412 0.9412];
app.xRestrainDeslizante.Position = [99 362 38 22];
app.xRestrainDeslizante.Text = ' x';

% Create RestrainFixed
app.RestrainFixed = uibutton(app.DesingTab, 'push');
app.RestrainFixed.ButtonPushedFcn = createCallbackFcn(app,
@RestrainFixedButtonPushed, true);
app.RestrainFixed.Icon = 'fijo.jpg';
app.RestrainFixed.BackgroundColor = [0.9412 0.9412 0.9412];
app.RestrainFixed.Position = [173 362 23 22];
app.RestrainFixed.Text = '';

% Create RestrainEmpotred
app.RestrainEmpotred = uibutton(app.DesingTab, 'push');
```

```
app.RestrainEmpotred.ButtonPushedFcn = createCallbackFcn(app,
@RestrainEmpotredButtonPushed, true);
app.RestrainEmpotred.Icon = 'empotrado.jpg';
app.RestrainEmpotred.BackgroundColor = [0.9412 0.9412 0.9412];
app.RestrainEmpotred.Position = [195 362 23 22];
app.RestrainEmpotred.Text = '';

% Create xButton_2
app.xButton_2 = uibutton(app.DesingTab, 'push');
app.xButton_2.ButtonPushedFcn = createCallbackFcn(app,
@xButton_2Pushed, true);
app.xButton_2.Icon = 'fuerza local.jpg';
app.xButton_2.BackgroundColor = [0.9412 0.9412 0.9412];
app.xButton_2.Position = [252 363 35 22];
app.xButton_2.Text = 'x';

% Create Button_9
app.Button_9 = uibutton(app.DesingTab, 'push');
app.Button_9.Icon = 'fx.jpg';
app.Button_9.IconAlignment = 'center';
app.Button_9.BackgroundColor = [1 1 1];
app.Button_9.Position = [326 363 23 22];
app.Button_9.Text = '';

% Create RestrainLabel
app.RestrainLabel = uilabel(app.DesingTab);
app.RestrainLabel.FontName = 'Calibri';
app.RestrainLabel.FontAngle = 'italic';
app.RestrainLabel.Position = [109 378 45 22];
app.RestrainLabel.Text = 'Restrain';

% Create NodalLabel
app.NodalLabel = uilabel(app.DesingTab);
app.NodalLabel.FontName = 'Calibri';
app.NodalLabel.FontAngle = 'italic';
app.NodalLabel.Position = [30 378 34 22];
app.NodalLabel.Text = 'Nodal';

% Create ForceLabel
app.ForceLabel = uilabel(app.DesingTab);
app.ForceLabel.FontName = 'Calibri';
app.ForceLabel.FontAngle = 'italic';
app.ForceLabel.Position = [287 378 32 22];
app.ForceLabel.Text = 'Force';

% Create Button_10
app.Button_10 = uibutton(app.DesingTab, 'push');
app.Button_10.Icon = 'x.jpg';
```

```
app.Button_10.Position = [421 363 23 22];
app.Button_10.Text = '';

% Create RunLabel
app.RunLabel = uilabel(app.DesingTab);
app.RunLabel.FontAngle = 'italic';
app.RunLabel.Position = [529 378 28 22];
app.RunLabel.Text = 'Run';

% Create SelectPolyLine
app.SelectPolyLine = uibutton(app.DesingTab, 'push');
app.SelectPolyLine.ButtonPushedFcn = createCallbackFcn(app,
@SelectPolyLineButtonPushed, true);
app.SelectPolyLine.Icon = 'polyselect.jpg';
app.SelectPolyLine.BackgroundColor = [1 1 1];
app.SelectPolyLine.Position = [556 162 23 22];
app.SelectPolyLine.Text = '';

% Create SelecLine
app.SelecLine = uibutton(app.DesingTab, 'push');
app.SelecLine.ButtonPushedFcn = createCallbackFcn(app,
@SelecLineButtonPushed, true);
app.SelecLine.Icon = 'lineselection.jpg';
app.SelecLine.BackgroundColor = [1 1 1];
app.SelecLine.Position = [556 203 23 22];
app.SelecLine.Text = '';

% Create SelectLabel
app.SelectLabel = uilabel(app.DesingTab);
app.SelectLabel.FontName = 'Dubai';
app.SelectLabel.Position = [551 181 35 22];
app.SelectLabel.Text = 'Select';

% Create yRestrainDeslizante
app.yRestrainDeslizante = uibutton(app.DesingTab, 'push');
app.yRestrainDeslizante.ButtonPushedFcn = createCallbackFcn(app,
@yRestrainDeslizanteButtonPushed, true);
app.yRestrainDeslizante.Icon = 'deslizante.jpg';
app.yRestrainDeslizante.IconAlignment = 'center';
app.yRestrainDeslizante.HorizontalAlignment = 'left';
app.yRestrainDeslizante.BackgroundColor = [0.9412 0.9412 0.9412];
app.yRestrainDeslizante.Position = [136 362 38 22];
app.yRestrainDeslizante.Text = ' y';

% Create Button_14
app.Button_14 = uibutton(app.DesingTab, 'push');
app.Button_14.Icon = 'fy.jpg';
app.Button_14.IconAlignment = 'center';
```

```
app.Button_14.BackgroundColor = [1 1 1];
app.Button_14.Position = [348 363 23 22];
app.Button_14.Text = '';

% Create yButton_2
app.yButton_2 = uibutton(app.DesingTab, 'push');
app.yButton_2.ButtonPushedFcn = createCallbackFcn(app,
@yButton_2Pushed, true);
app.yButton_2.Icon = 'fuerza local.jpg';
app.yButton_2.BackgroundColor = [0.9412 0.9412 0.9412];
app.yButton_2.Position = [287 363 40 22];
app.yButton_2.Text = 'y';

% Create PEditFieldLabel
app.PEditFieldLabel = uilabel(app.DesingTab);
app.PEditFieldLabel.HorizontalAlignment = 'center';
app.PEditFieldLabel.Position = [219 363 17 22];
app.PEditFieldLabel.Text = 'P';

% Create PEditField
app.PEditField = uieditfield(app.DesingTab, 'numeric');
app.PEditField.ValueChangedFcn = createCallbackFcn(app,
@PEditFieldValueChanged, true);
app.PEditField.Position = [234 363 20 22];

% Create EditField
app.EditField = uieditfield(app.DesingTab, 'text');
app.EditField.Position = [370 363 52 22];

% Create SelectPolyLine_2
app.SelectPolyLine_2 = uibutton(app.DesingTab, 'push');
app.SelectPolyLine_2.Icon = 'rectangle.jpg';
app.SelectPolyLine_2.BackgroundColor = [1 1 1];
app.SelectPolyLine_2.Position = [556 136 23 22];
app.SelectPolyLine_2.Text = '';

% Create circle
app.circle = uibutton(app.DesingTab, 'push');
app.circle.ButtonPushedFcn = createCallbackFcn(app,
@circleButtonPushed, true);
app.circle.Icon = 'circle.jpg';
app.circle.BackgroundColor = [1 1 1];
app.circle.Position = [556 110 23 22];
app.circle.Text = '';

% Create SelectPolyLine_4
app.SelectPolyLine_4 = uibutton(app.DesingTab, 'push');
```

```
app.SelectPolyLine_4.ButtonPushedFcn = createCallbackFcn(app,
@SelectPolyLine_4ButtonPushed, true);
app.SelectPolyLine_4.Icon = 'cota.jpg';
app.SelectPolyLine_4.BackgroundColor = [0.9412 0.9412 0.9412];
app.SelectPolyLine_4.Position = [557 38 23 22];
app.SelectPolyLine_4.Text = '';

% Create DeflectionAnalysisTab
app.DeflectionAnalysisTab = uitab(app.TabGroup2_2);
app.DeflectionAnalysisTab.Title = 'Deflection Analysis';
app.DeflectionAnalysisTab.BackgroundColor = [0.9412 0.9412 0.9412];

% Create UIAxes_11
app.UIAxes_11 = uiaxes(app.DeflectionAnalysisTab);
title(app.UIAxes_11, '')
xlabel(app.UIAxes_11, '')
ylabel(app.UIAxes_11, '')
app.UIAxes_11.PlotBoxAspectRatio = [1.5 1 1];
app.UIAxes_11.FontName = 'Ink Free';
app.UIAxes_11.FontWeight = 'bold';
app.UIAxes_11.GridColor = [1 1 1];
app.UIAxes_11.MinorGridColor = [0 0 0];
app.UIAxes_11.XTick = [0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1];
app.UIAxes_11.BackgroundColor = [0.9412 0.9412 0.9412];
app.UIAxes_11.Position = [10 25 520 367];

% Create Button_15
app.Button_15 = uibutton(app.DeflectionAnalysisTab, 'push');
app.Button_15.ButtonPushedFcn = createCallbackFcn(app,
@Button_15Pushed, true);
app.Button_15.Icon = 'playapp.jpg';
app.Button_15.Position = [6 98 51 35];
app.Button_15.Text = '';

% Create AnalisisDropDownLabel_2
app.AnalisisDropDownLabel_2 = uilabel(app.DeflectionAnalysisTab);
app.AnalisisDropDownLabel_2.HorizontalAlignment = 'right';
app.AnalisisDropDownLabel_2.Position = [229 -1 47 22];
app.AnalisisDropDownLabel_2.Text = 'Analisis';

% Create AnalisisDropDown_3
app.AnalisisDropDown_3 = uidropdown(app.DeflectionAnalysisTab);
app.AnalisisDropDown_3.Items = {'?1', '?2', '?_von misses',
'Moment', 'Axial', 'Cortante', 'Fracción Vol.'};
app.AnalisisDropDown_3.FontName = 'Centaur';
app.AnalisisDropDown_3.FontSize = 14;
app.AnalisisDropDown_3.FontWeight = 'bold';
app.AnalisisDropDown_3.BackgroundColor = [0.9412 0.9412 0.9412];
```

```
app.AnalisisDropDown_3.Position = [234 -1 100 23];
app.AnalisisDropDown_3.Value = '?1';

% Create AnalisisDropDownLabel_3
app.AnalisisDropDownLabel_3 = uilabel(app.DeflectionAnalysisTab);
app.AnalisisDropDownLabel_3.HorizontalAlignment = 'right';
app.AnalisisDropDownLabel_3.FontName = 'Century Gothic';
app.AnalisisDropDownLabel_3.FontWeight = 'bold';
app.AnalisisDropDownLabel_3.Position = [178 0 48 22];
app.AnalisisDropDownLabel_3.Text = 'Analisis';

% Create MinglobalCheckBox
app.MinglobalCheckBox = uicheckbox(app.DeflectionAnalysisTab);
app.MinglobalCheckBox.Text = 'Min.global';
app.MinglobalCheckBox.FontSize = 8;
app.MinglobalCheckBox.Position = [543 305 58 22];

% Create HolesCheckBox_2
app.HolesCheckBox_2 = uicheckbox(app.DeflectionAnalysisTab);
app.HolesCheckBox_2.Text = 'Holes';
app.HolesCheckBox_2.FontSize = 8;
app.HolesCheckBox_2.Position = [543 326 42 28];

% Create MaxglobalCheckBox
app.MaxglobalCheckBox = uicheckbox(app.DeflectionAnalysisTab);
app.MaxglobalCheckBox.Text = 'Max.global';
app.MaxglobalCheckBox.FontSize = 8;
app.MaxglobalCheckBox.Position = [543 284 61 22];

% Create RestrainCheckBox_2
app.RestrainCheckBox_2 = uicheckbox(app.DeflectionAnalysisTab);
app.RestrainCheckBox_2.Text = 'Restrain';
app.RestrainCheckBox_2.FontSize = 8;
app.RestrainCheckBox_2.Position = [543 263 49 22];

% Create Simulation2DTab
app.Simulation2DTab = uitab(app.TabGroup2_2);
app.Simulation2DTab.Title = 'Simulation 2D';

% Create UIAxes_9
app.UIAxes_9 = uiaxes(app.Simulation2DTab);
title(app.UIAxes_9, '')
xlabel(app.UIAxes_9, '')
ylabel(app.UIAxes_9, '')
app.UIAxes_9.PlotBoxAspectRatio = [1.65 1 1];
app.UIAxes_9.FontWeight = 'bold';
app.UIAxes_9.Position = [6 -4 579 398];
```

```
% Create Button_16
app.Button_16 = uibutton(app.Simulation2DTab, 'push');
app.Button_16.ButtonPushedFcn = createCallbackFcn(app,
@Button_16Pushed, true);
app.Button_16.Icon = 'playapp2.jpg';
app.Button_16.IconAlignment = 'top';
app.Button_16.Position = [6 98 51 35];
app.Button_16.Text = '';

% Create Simulation25DTab
app.Simulation25DTab = uitab(app.TabGroup2_2);
app.Simulation25DTab.Title = 'Simulation 2.5D';
app.Simulation25DTab.BackgroundColor = [0.9412 0.9412 0.9412];

% Create UIAxes_10
app.UIAxes_10 = uiaxes(app.Simulation25DTab);
title(app.UIAxes_10, '')
xlabel(app.UIAxes_10, '')
ylabel(app.UIAxes_10, '')
app.UIAxes_10.PlotBoxAspectRatio = [1.65 1 1];
app.UIAxes_10.FontWeight = 'bold';
app.UIAxes_10.XTick = [0 0.2 0.4 0.6 0.8 1];
app.UIAxes_10.BackgroundColor = [0.9412 0.9412 0.9412];
app.UIAxes_10.Position = [1 -2 582 394];

% Create Button_17
app.Button_17 = uibutton(app.Simulation25DTab, 'push');
app.Button_17.Icon = 'playapp3.jpg';
app.Button_17.Position = [6 98 51 35];
app.Button_17.Text = '';

% Create ImageTab
app.ImageTab = uitab(app.TabGroup2_2);
app.ImageTab.Title = 'Image';

% Create image
app.image = uibutton(app.ImageTab, 'push');
app.image.ButtonPushedFcn = createCallbackFcn(app, @imagePushed,
true);
app.image.Icon = 'playapp4.jpg';
app.image.Position = [6 98 51 35];
app.image.Text = '';

% Create UIAxes3
app.UIAxes3 = uiaxes(app.ImageTab);
title(app.UIAxes3, 'Title')
xlabel(app.UIAxes3, 'X')
ylabel(app.UIAxes3, 'Y')
```

```
app.UIAxes3.PlotBoxAspectRatio = [1.27350427350427 1 1];
app.UIAxes3.Position = [63 12 445 368];

% Create GenerateButton
app.GenerateButton = uibutton(app.CreateTab, 'push');
app.GenerateButton.ButtonPushedFcn = createCallbackFcn(app,
@GenerateButtonPushed, true);
app.GenerateButton.Position = [21 1 66 22];
app.GenerateButton.Text = 'Generate';

% Create UnitsDropDownLabel
app.UnitsDropDownLabel = uilabel(app.CreateTab);
app.UnitsDropDownLabel.HorizontalAlignment = 'right';
app.UnitsDropDownLabel.Position = [619 0 33 22];
app.UnitsDropDownLabel.Text = 'Units';

% Create UnitsDropDown
app.UnitsDropDown = uidropdown(app.CreateTab);
app.UnitsDropDown.Items = {'KN_m', 'N_mm', 'kgf_m', 'kgf_mm'};
app.UnitsDropDown.Position = [667 0 39 22];
app.UnitsDropDown.Value = 'KN_m';

% Create MaterialTab
app.MaterialTab = uitab(app.TabGroup);
app.MaterialTab.Title = 'Material';

% Create UIAxes2
app.UIAxes2 = uiaxes(app.MaterialTab);
title(app.UIAxes2, '')
xlabel(app.UIAxes2, '')
ylabel(app.UIAxes2, '')
app.UIAxes2.DataAspectRatio = [1 1 1];
app.UIAxes2.PlotBoxAspectRatio = [1.6 1.02272727272727 1];
app.UIAxes2.Position = [150 38 400 403];

% Create MaterialDropDownLabel
app.MaterialDropDownLabel = uilabel(app.MaterialTab);
app.MaterialDropDownLabel.HorizontalAlignment = 'right';
app.MaterialDropDownLabel.Position = [3 374 46 22];
app.MaterialDropDownLabel.Text = 'Material';

% Create MaterialDropDown
app.MaterialDropDown = uidropdown(app.MaterialTab);
app.MaterialDropDown.Items = {'Concreto', 'Acero', 'Madera'};
app.MaterialDropDown.Position = [8 351 95 22];
app.MaterialDropDown.Value = 'Concreto';

% Create TipodeseccinDropDownLabel
```

```
app.TipodeseccinDropDownLabel = uilabel(app.MaterialTab);
app.TipodeseccinDropDownLabel.HorizontalAlignment = 'right';
app.TipodeseccinDropDownLabel.Position = [9 325 86 22];
app.TipodeseccinDropDownLabel.Text = 'Tipo de sección';

% Create TipodeseccinDropDown
app.TipodeseccinDropDown = uidropdown(app.MaterialTab);
app.TipodeseccinDropDown.Items = {'Rectangular', 'Circular',
'Tubular', 'HN', 'XL'};
app.TipodeseccinDropDown.Position = [8 301 95 22];
app.TipodeseccinDropDown.Value = 'Rectangular';

% Create UnidadesDropDownLabel
app.UnidadesDropDownLabel = uilabel(app.MaterialTab);
app.UnidadesDropDownLabel.HorizontalAlignment = 'right';
app.UnidadesDropDownLabel.Position = [4 428 53 22];
app.UnidadesDropDownLabel.Text = 'Unidades';

% Create UnidadesDropDown
app.UnidadesDropDown = uidropdown(app.MaterialTab);
app.UnidadesDropDown.Items = {'Kip_ft', 'N_m', 'Tonf_m'};
app.UnidadesDropDown.Position = [8 405 95 22];
app.UnidadesDropDown.Value = 'Kip_ft';

% Create SectionButton
app.SectionButton = uibutton(app.MaterialTab, 'push');
app.SectionButton.ButtonPushedFcn = createCallbackFcn(app,
@SectionButtonPushed, true);
app.SectionButton.Position = [11 43 100 22];
app.SectionButton.Text = 'Section';

% Create UIAxes2_2
app.UIAxes2_2 = uiaxes(app.MaterialTab);
title(app.UIAxes2_2, '')
xlabel(app.UIAxes2_2, '')
ylabel(app.UIAxes2_2, '')
app.UIAxes2_2.PlotBoxAspectRatio = [1.54545454545455 1 1];
app.UIAxes2_2.Position = [1 200 112 87];

% Create OnButton
app.OnButton = uibutton(app.MaterialTab, 'push');
app.OnButton.ButtonPushedFcn = createCallbackFcn(app,
@OnButtonPushed, true);
app.OnButton.Position = [9 160 100 22];
app.OnButton.Text = 'On';

% Create CNCTab
app.CNCTab = uitab(app.TabGroup);
```

```
app.CNCTab.Title = 'CNC';
app.CNCTab.BackgroundColor = [0.9412 0.9412 0.9412];

% Create ROUNDEDButtonGroup
app.ROUNDEDButtonGroup = uibuttongroup(app.CNCTab);
app.ROUNDEDButtonGroup.TitlePosition = 'centertop';
app.ROUNDEDButtonGroup.Title = 'ROUNDED';
app.ROUNDEDButtonGroup.Position = [7 361 101 49];

% Create DropDown
app.DropDown = uidropdown(app.ROUNDEDButtonGroup);
app.DropDown.Items = {'Linear', 'Circular', 'Spline', 'None'};
app.DropDown.Position = [8 5 87 22];
app.DropDown.Value = 'Linear';

% Create FORMATButtonGroup_2
app.FORMATButtonGroup_2 = uibuttongroup(app.CNCTab);
app.FORMATButtonGroup_2.TitlePosition = 'centertop';
app.FORMATButtonGroup_2.Title = 'FORMAT';
app.FORMATButtonGroup_2.Position = [4 100 104 81];

% Create Button_24
app.Button_24 = uibutton(app.FORMATButtonGroup_2, 'push');
app.Button_24.Icon = 'cld-cloud-computer-network-svgrepo-com.svg';
app.Button_24.Position = [70 2 46 25];
app.Button_24.Text = '';

% Create DropDown_2
app.DropDown_2 = uidropdown(app.FORMATButtonGroup_2);
app.DropDown_2.Items = {'STL', 'OBJ', 'SAP2000', 'CALFEM', 'None'};
app.DropDown_2.Position = [13 37 87 22];
app.DropDown_2.Value = 'STL';

% Create ExportButton_2
app.ExportButton_2 = uibutton(app.CNCTab, 'push');
app.ExportButton_2.Position = [8 40 100 22];
app.ExportButton_2.Text = ' Export';

% Create TabGroup2
app.TabGroup2 = uitabgroup(app.CNCTab);
app.TabGroup2.Position = [125 26 569 419];

% Create VisualTab
app.VisualTab = uitab(app.TabGroup2);
app.VisualTab.Title = 'Visual';

% Create UIAxes
app.UIAxes = uiaxes(app.VisualTab);
```

```
title(app.UIAxes, '')
xlabel(app.UIAxes, '')
ylabel(app.UIAxes, '')
app.UIAxes.PlotBoxAspectRatio = [1.13253012048193 1 1];
app.UIAxes.Position = [1 19 567 375];

% Create ShowButton_5
app.ShowButton_5 = uibutton(app.VisualTab, 'push');
app.ShowButton_5.Icon = 'playapp2.jpg';
app.ShowButton_5.Position = [494 14 100 22];
app.ShowButton_5.Text = 'Show';

% Create SimulationTab
app.SimulationTab = uitab(app.TabGroup2);
app.SimulationTab.Title = 'Simulation';

% Create UIAxes_3
app.UIAxes_3 = uiaxes(app.SimulationTab);
title(app.UIAxes_3, '')
xlabel(app.UIAxes_3, '')
ylabel(app.UIAxes_3, '')
app.UIAxes_3.PlotBoxAspectRatio = [1.13654618473896 1 1];
app.UIAxes_3.Position = [0 19 425 375];

% Create ShowButton_6
app.ShowButton_6 = uibutton(app.SimulationTab, 'push');
app.ShowButton_6.Position = [163 14 100 22];
app.ShowButton_6.Text = 'Show';

% Create AnalisisDropDown_4Label
app.AnalisisDropDown_4Label = uilabel(app.SimulationTab);
app.AnalisisDropDown_4Label.HorizontalAlignment = 'right';
app.AnalisisDropDown_4Label.Position = [426 369 47 22];
app.AnalisisDropDown_4Label.Text = 'Analisis';

% Create AnalisisDropDown_4
app.AnalisisDropDown_4 = uidropdown(app.SimulationTab);
app.AnalisisDropDown_4.Items = {'Displacement ', 'Stress',
'Strain', 'Von Misses'};
app.AnalisisDropDown_4.Position = [412 369 13 22];
app.AnalisisDropDown_4.Value = 'Displacement ';

% Create EjeDropDownLabel
app.EjeDropDownLabel = uilabel(app.SimulationTab);
app.EjeDropDownLabel.HorizontalAlignment = 'right';
app.EjeDropDownLabel.Position = [426 342 25 22];
app.EjeDropDownLabel.Text = 'Eje';
```

```
% Create EjeDropDown
app.EjeDropDown = uidropdown(app.SimulationTab);
app.EjeDropDown.Items = {'x', 'y', 'z', 'xx', 'yy', 'zz', 'xz',
'xy', 'yz', 'Magnitud'};
app.EjeDropDown.Position = [412 342 14 22];
app.EjeDropDown.Value = 'x';

% Create VoxelTab
app.VoxelTab = uitab(app.TabGroup2);
app.VoxelTab.Title = 'Voxel';

% Create UIAxes_4
app.UIAxes_4 = uiaxes(app.VoxelTab);
title(app.UIAxes_4, '')
xlabel(app.UIAxes_4, '')
ylabel(app.UIAxes_4, '')
app.UIAxes_4.PlotBoxAspectRatio = [1.13654618473896 1 1];
app.UIAxes_4.FontWeight = 'bold';
app.UIAxes_4.Position = [0 19 425 375];

% Create ShowButton_7
app.ShowButton_7 = uibutton(app.VoxelTab, 'push');
app.ShowButton_7.Position = [163 14 100 22];
app.ShowButton_7.Text = 'Show';

% Create FILTERButtonGroup
app.FILTERButtonGroup = uibuttongroup(app.CNCTab);
app.FILTERButtonGroup.TitlePosition = 'centertop';
app.FILTERButtonGroup.Title = 'FILTER';
app.FILTERButtonGroup.Position = [6 301 104 55];

% Create ButtonFilter
app.ButtonFilter = uibutton(app.FILTERButtonGroup, 'push');
app.ButtonFilter.ButtonPushedFcn = createCallbackFcn(app,
@ButtonFilterPushed, true);
app.ButtonFilter.Icon = 'filter.svg';
app.ButtonFilter.Position = [72 9 38 22];
app.ButtonFilter.Text = '';

% Create Button_29
app.Button_29 = uibutton(app.FILTERButtonGroup, 'push');
app.Button_29.Icon = 'hamburger-soda.svg';
app.Button_29.Position = [30 9 38 22];
app.Button_29.Text = '';

% Create Lamp
app.Lamp = uilamp(app.FILTERButtonGroup);
app.Lamp.Position = [6 15 10 10];
```

```
% Create Button_18
app.Button_18 = uibutton(app.CNCTab, 'push');
app.Button_18.Icon = 'menu.svg';
app.Button_18.Position = [12 419 38 22];
app.Button_18.Text = '';

% Create Button_23
app.Button_23 = uibutton(app.CNCTab, 'push');
app.Button_23.Icon = 'clean.svg';
app.Button_23.Position = [64 0 31 32];
app.Button_23.Text = '';

% Create ADVANCEDButtonGroup_2
app.ADVANCEDButtonGroup_2 = uibuttongroup(app.CNCTab);
app.ADVANCEDButtonGroup_2.TitlePosition = 'centertop';
app.ADVANCEDButtonGroup_2.Title = 'ADVANCED';
app.ADVANCEDButtonGroup_2.Position = [6 243 104 55];

% Create Button_25
app.Button_25 = uibutton(app.ADVANCEDButtonGroup_2, 'push');
app.Button_25.Icon = 'settings.svg';
app.Button_25.Position = [72 9 38 22];
app.Button_25.Text = '';

% Create Button_27
app.Button_27 = uibutton(app.ADVANCEDButtonGroup_2, 'push');
app.Button_27.Icon = 'hamburger-soda.svg';
app.Button_27.Position = [31 9 38 22];
app.Button_27.Text = '';

% Create Lamp_2
app.Lamp_2 = uilamp(app.ADVANCEDButtonGroup_2);
app.Lamp_2.Position = [6 15 10 10];

% Create RENDERINGButtonGroup
app.RENDERINGButtonGroup = uibuttongroup(app.CNCTab);
app.RENDERINGButtonGroup.TitlePosition = 'centertop';
app.RENDERINGButtonGroup.Title = 'RENDERING';
app.RENDERINGButtonGroup.Position = [5 184 104 55];

% Create Button_26
app.Button_26 = uibutton(app.RENDERINGButtonGroup, 'push');
app.Button_26.Icon = 'sketchup-02-svgrepo-com.svg';
app.Button_26.Position = [72 9 38 22];
app.Button_26.Text = '';

% Create Button_28
```

```
app.Button_28 = uibutton(app.RENDERINGButtonGroup, 'push');
app.Button_28.Icon = 'hamburger-soda.svg';
app.Button_28.Position = [31 9 38 22];
app.Button_28.Text = '';

% Create Lamp_3
app.Lamp_3 = uilamp(app.RENDERINGButtonGroup);
app.Lamp_3.Position = [7 15 10 10];

% Create Button_30
app.Button_30 = uibutton(app.CNCTab, 'push');
app.Button_30.Icon = 'hexagon-check.svg';
app.Button_30.Position = [99 1 33 31];
app.Button_30.Text = '';

% Create FRAMETab
app.FRAMETab = uitab(app.TabGroup);
app.FRAMETab.Title = 'FRAME';

% Show the figure after all components are created
app.UIFigure.Visible = 'on';
end
end

% App creation and deletion
methods (Access = public)

% Construct app
function app = MemoriaHomerespaldo

% Create UIFigure and components
createComponents(app)

% Register the app with App Designer
registerApp(app, app.UIFigure)

if nargout == 0
clear app
end
end

% Code that executes before app deletion
function delete(app)

% Delete UIFigure when app is deleted
delete(app.UIFigure)
end
end;end
```

ANEXO 9: Rutina principal en MATLAB® desarrollada por Oviedo

Se adjunta rutina principal elaborada por Oviedo (2020) en el desarrollo de un modelo de optimización topológica para procesos de manufactura aditiva en MATLAB® para viga simplemente apoyada con carga puntual en el centro. Se integran las funciones 'Softkillbeso', 'FE', 'lk', 'check' y 'ADDDEL' necesarias para el funcionamiento del programa en este caso.

```
clear all
close all
clc

%%
%%%%% INICIO INPUT %%%%%

prom={'Ingrese N° de elementos en X:','Ingrese N° de elementos en
Y:','Ingrese fracción de volumen deseada FV inicial [%]:',...
    'Ingrese ER (defecto 0.02):', 'Ingrese RMIN [mm] (defecto
3):','Ingrese módulo de elasticidad E [MPa]):',...
    'Ingrese módulo de Poisson v:','Ingrese fuerza aplicada P
[N]:','Ingrese tamaño máximo de elementos de malla triangular
[mm]:',...
    'Ingrese N° de decimales a considerar:'};
def={'60','40','30','0.02','3','23500','0.25','100','2','1'};
INPUT=inputdlg(prom,'Ingreso datos de entrada',1,def,'on');
tic
nelx=str2double(INPUT{1})+1;
nely=str2double(INPUT{2})+1;
volfrac=(str2double(INPUT{3}))/100;
er=str2double(INPUT{4});
rmin=str2double(INPUT{5});
E=str2double(INPUT{6});
nu=str2double(INPUT{7});
P=str2double(INPUT{8});
sizelement=str2double(INPUT{9});
tolr=str2double(INPUT{10});

%%%%% TÉRMINO INPUT %%%%%

%%
%%%%% INICIO COMANDO OPTIMIZADOR %%%%%

[x,U]=Softkillbeso(nelx,nely,volfrac,er,rmin,E,nu,P);

%%%%% TÉRMINO COMANDO OPTIMIZADOR %%%%%
```

```
%%
%%%%% INICIO COMANDO SUAVIZADOR %%%%%

x=fliplr(rot90(rot90(x)));
[b,a]=size(x);
M=zeros(b,a);
R=zeros(b,a);
for i=1:b
    for j=1:a
        if x(i,j)==1
            M(i,j)=1;
        elseif x(i,j)==0.001
            M(i,j)=0;
        end
    end
end
figure(2)
surf(M)
view(0,90)
axis([-5 a+5 -5 b+5])
ppf=sum(M);
puntos=sum(sum(M));
CX=[];
CY=[];
for i=1:a
    CX=[CX,i*ones(1,ppf(i))];
end
for i=1:a
    CY=[CY,find(M(:,i))'];
end
figure(3)
D=scatter(CX',CY',400,'b','.');
view(0,90)
axis([-5 a+5 -5 b+5])
for i=2:b-1
    for j=2:a-1
        if M(i,j)==1 && M(i,j+1)==0
            R(i,j)=1;
        elseif M(i,j)==1 && M(i,j-1)==0
            R(i,j)=1;
        elseif M(i,j)==1 && M(i+1,j)==0
            R(i,j)=1;
        elseif M(i,j)==1 && M(i-1,j)==0
            R(i,j)=1;
        end
    end
end
for i=1:b
```

```
    if M(i,1)==1
        R(i,1)=1;
    end
end
for i=1:b
    if M(i,a)==1
        R(i,a)=1;
    end
end
for j=1:a
    if M(1,j)==1
        R(1,j)=1;
    end
end
for j=1:a
    if M(b,j)==1
        R(b,j)=1;
    end
end
figure(4)
surf(R)
view(0,90)
axis([-5 a+5 -5 b+5])
ppf1=sum(R);
puntos1=sum(sum(R));
CX1=[];
CY1=[];
for i=1:a
    CX1=[CX1,i*ones(1,ppf1(i))];
end
for i=1:a
    CY1=[CY1,find(R(:,i))'];
end
figure(5)
D1=scatter(CX1',CY1',400,'r','.');
view(0,90)
axis([-5 a+5 -5 b+5])

%%%%% INICIO IDENTIFICACIÓN DE CONTORNO Y CAVIDADES INTERNAS %%%%%

[B,L,N,A]=bwboundaries(M);
figure(6)
imshow(M)
hold on
colors=['b','g','r','c','m','y'];
for k=1:length(B)
  boundary=B{k};
  cidx=mod(k,length(colors))+1;
```

```
  plot(boundary(:,2), boundary(:,1),colors(cidx),'LineWidth',2);
  rndRow=ceil(length(boundary)/(mod(rand*k,7)+1));
  col=boundary(rndRow,2); row = boundary(rndRow,1);
  h=text(col+1, row-1, num2str(L(row,col)));
  set(h,'Color',colors(cidx),'FontSize',14,'FontWeight','bold');
end
figure(7)
imshow(M)
hold on
for k=1:N
    if (nnz(A(:,k))>0)
        boundary=B{k};
        plot(boundary(:,2),boundary(:,1),'r','LineWidth',2);
        for l=find(A(:,k))'
            boundary=B{l};
            plot(boundary(:,2),boundary(:,1),'g','LineWidth',2);
        end
    end
end
pcont(1,:)=B{1}(1,:);
contour{1}=bwtraceboundary(R,pcont(1,:),'E',8,Inf,'clockwise');
figure(8)
plot(contour{1}(:,2),contour{1}(:,1),'g','LineWidth',2);
hold on
eliminar=[];
refill=[];
for i=1:length(B)-1
    if length(B{i})<=9
        eliminar=[eliminar,1];
        refill=[refill,i];
    end
end
if length(eliminar)>=1
    B1=length(B);
    B2=B;
    while length(B2)>B1-sum(eliminar)
        auxelim=[];
        for i=1:length(refill)
            for j=1:length(B2{refill(i)})-1
                auxelim=B2{refill(i)}(j,:);
                M(auxelim(1),auxelim(2))=1;
            end
        end
        [B2,L,N,A]=bwboundaries(M);
    end
    B=B2;
end
for i=1:length(B)-1
```

```
    pvacio(i,:)=B{i+1}(1,:);
end
for k=1:length(B)-1
    for i=2:b-1
        for j=2:a-1
            if R(pvacio(k,1),pvacio(k,2))==0 &&
R(pvacio(k,1),pvacio(k,2)+1)==1
                pborde(k,:)=[pvacio(k,1),pvacio(k,2)+1];
                break
                break
            elseif R(pvacio(k,1),pvacio(k,2))==0 &&
R(pvacio(k,1)+1,pvacio(k,2)+1)==1
                pborde(k,:)=[pvacio(k,1)+1,pvacio(k,2)+1];
                break
                break
            elseif R(pvacio(k,1),pvacio(k,2))==0 &&
R(pvacio(k,1)+1,pvacio(k,2))==1
                pborde(k,:)=[pvacio(k,1)+1,pvacio(k,2)];
                break
                break
            elseif R(pvacio(k,1),pvacio(k,2))==0 &&
R(pvacio(k,1)+1,pvacio(k,2)-1)==1
                pborde(k,:)=[pvacio(k,1)+1,pvacio(k,2)-1];
                break
                break
            elseif R(pvacio(k,1),pvacio(k,2))==0 &&
R(pvacio(k,1),pvacio(k,2)-1)==1
                pborde(k,:)=[pvacio(k,1),pvacio(k,2)-1];
                break
                break
            elseif R(pvacio(k,1),pvacio(k,2))==0 && R(pvacio(k,1)-
1,pvacio(k,2)-1)==1
                pborde(k,:)=[pvacio(k,1)-1,pvacio(k,2)-1];
                break
                break
            elseif R(pvacio(k,1),pvacio(k,2))==0 && R(pvacio(k,1)-
1,pvacio(k,2))==1
                pborde(k,:)=[pvacio(k,1)-1,pvacio(k,2)];
                break
                break
            elseif R(pvacio(k,1),pvacio(k,2))==0 && R(pvacio(k,1)-
1,pvacio(k,2)+1)==1
                pborde(k,:)=[pvacio(k,1)-1,pvacio(k,2)+1];
                break
                break
            end
        end
    end
```

```
end
for k=1:length(B)-1
    for i=2:b-1
        for j=2:a-1
            if R(pborde(k,1),pborde(k,2))==1 &&
R(pborde(k,1),pborde(k,2)+1)==1

contour{k+1}=bwtraceboundary(M,pborde(k,:),'N',8,Inf,'clockwise');
                figure(8)

plot(contour{k+1}(:,2),contour{k+1}(:,1),'g','LineWidth',2);
                view(0,90)
                axis([-5 a+5 -5 b+5])
                hold on
                break
                break
            elseif R(pborde(k,1),pborde(k,2))==1 &&
R(pborde(k,1)+1,pborde(k,2)+1)==1

contour{k+1}=bwtraceboundary(M,pborde(k,:),'NE',8,Inf,'clockwise');
                figure(8)

plot(contour{k+1}(:,2),contour{k+1}(:,1),'g','LineWidth',2);
                view(0,90)
                axis([-5 a+5 -5 b+5])
                hold on
                break
                break
            elseif R(pborde(k,1),pborde(k,2))==1 &&
R(pborde(k,1)+1,pborde(k,2))==1

contour{k+1}=bwtraceboundary(M,pborde(k,:),'E',8,Inf,'clockwise');
                figure(8)

plot(contour{k+1}(:,2),contour{k+1}(:,1),'g','LineWidth',2);
                view(0,90)
                axis([-5 a+5 -5 b+5])
                hold on
                break
                break
            elseif R(pborde(k,1),pborde(k,2))==1 &&
R(pborde(k,1)+1,pborde(k,2)-1)==1

contour{k+1}=bwtraceboundary(M,pborde(k,:),'SE',8,Inf,'clockwise');
                figure(8)

plot(contour{k+1}(:,2),contour{k+1}(:,1),'g','LineWidth',2);
                view(0,90)
```

```
                axis([-5 a+5 -5 b+5])
                hold on
                break
                break
            elseif R(pborde(k,1),pborde(k,2))==1 &&
R(pborde(k,1),pborde(k,2)-1)==1

contour{k+1}=bwtraceboundary(M,pborde(k,:),'S',8,Inf,'clockwise');
                figure(8)

plot(contour{k+1}(:,2),contour{k+1}(:,1),'g','LineWidth',2);
                view(0,90)
                axis([-5 a+5 -5 b+5])
                hold on
                break
                break
            elseif R(pborde(k,1),pborde(k,2))==1 && R(pborde(k,1)-
1,pborde(k,2)-1)==1

contour{k+1}=bwtraceboundary(M,pborde(k,:),'SW',8,Inf,'clockwise');
                figure(8)

plot(contour{k+1}(:,2),contour{k+1}(:,1),'g','LineWidth',2);
                view(0,90)
                axis([-5 a+5 -5 b+5])
                hold on
                break
                break
            elseif R(pborde(k,1),pborde(k,2))==1 && R(pborde(k,1)-
1,pborde(k,2))==1

contour{k+1}=bwtraceboundary(M,pborde(k,:),'W',8,Inf,'clockwise');
                figure(8)

plot(contour{k+1}(:,2),contour{k+1}(:,1),'g','LineWidth',2);
                view(0,90)
                axis([-5 a+5 -5 b+5])
                hold on
                break
                break
            elseif R(pborde(k,1),pborde(k,2))==1 && R(pborde(k,1)-
1,pborde(k,2)+1)==1

contour{k+1}=bwtraceboundary(M,pborde(k,:),'NW',8,Inf,'clockwise');
                figure(8)

plot(contour{k+1}(:,2),contour{k+1}(:,1),'g','LineWidth',2);
                view(0,90)
```

```
                axis([-5 a+5 -5 b+5])
                hold on
                break
                break
            end
        end
    end
end

%%%%% TÉRMINO IDENTIFICACIÓN DE CONTORNO Y CAVIDADES INTERNAS %%%%%

%%%%% INICIO BÚSQUEDA PUNTOS INTERPOLACIÓN LINEAL CONTORNO %%%

pos=[1];
dex=[contour{1}(:,2),contour{1}(:,1)];
npt=length(dex);
for i=1:length(dex)-1
    if dex(i,:)==[a,1]
        pos=[pos,i];
    end
    if dex(i,1)==a && dex(i+1,1)==a-1
        pos=[pos,i];
    end
    if dex(i,2)==b-1 && dex(i+1,2)==b
        pos=[pos,i+1];
    end
    if dex(i,:)==[1,b]
        pos=[pos,i];
    end
end
cont=pos(end);
dexf=[contour{1}(cont:npt,2),contour{1}(cont:npt,1)];
for i=1:length(dexf)-1
    if dexf(i,1)==1 && dexf(i+1,1)~=1
        pos=[pos,i+cont-1];
    end
    if dexf(i,1)~=1 && dexf(i+1,1)==1
        pos=[pos,i+1+cont-1];
    end
end
for i=1:length(dexf)-1
    if dexf(i,2)>contour{1}(pos(end)) &&
dexf(i+1,2)==contour{1}(pos(end))
        pos=[pos,i+cont-1];
    end
end
pos=sort(pos);
ptosbreak=flip(fliplr(contour{1}(pos(3):pos(4),:)));
```

```
nodobreak=[];
apb=[zeros(6,2);ptosbreak;zeros(6,2)];
for i=7:length(apb)-6
    if apb(i-1,1)==apb(i,1)-1 && apb(i,1)==apb(i+1,1)...
            && apb(i-1,2)==apb(i,2) && apb(i,2)==apb(i+1,2)+1
        nodobreak=[nodobreak,i-6];
    end % CASO 1
    if apb(i-1,1)==apb(i,1)-1 && apb(i,1)==apb(i+1,1)-1 &&
apb(i+1,1)==apb(i+2,1)...
            && apb(i-1,2)==apb(i,2) && apb(i,2)==apb(i+1,2)+1 &&
apb(i+1,2)==apb(i+2,2)+1
        nodobreak=[nodobreak,i-6,i+1-6];
    end % CASO 2
    if apb(i-2,1)==apb(i-1,1)-1 && apb(i-1,1)==apb(i,1)-1 &&
apb(i,1)==apb(i+1,1)-1 && apb(i+1,1)==apb(i+2,1)...
            && apb(i-2,2)==apb(i-1,2) && apb(i-1,2)==apb(i,2)+1 &&
apb(i,2)==apb(i+1,2)+1 && apb(i+1,2)==apb(i+2,2)+1
        nodobreak=[nodobreak,i-6];
    end % CASO 3
    if apb(i-2,1)==apb(i-1,1)-1 && apb(i-1,1)==apb(i,1)-1 &&
apb(i,1)==apb(i+1,1)-1 && apb(i+1,1)==apb(i+2,1)-1 &&
apb(i+2,1)==apb(i+3,1)...
            && apb(i-2,2)==apb(i-1,2) && apb(i-1,2)==apb(i,2)+1 &&
apb(i,2)==apb(i+1,2)+1 && apb(i+1,2)==apb(i+2,2)+1 &&
apb(i+2,2)==apb(i+3,2)+1
        nodobreak=[nodobreak,i-6,i+1-6];
    end % CASO 4
    if apb(i-3,1)==apb(i-2,1)-1 && apb(i-2,1)==apb(i-1,1)-1 &&
apb(i-1,1)==apb(i,1)-1 && apb(i,1)==apb(i+1,1)-1 &&
apb(i+1,1)==apb(i+2,1)-1 && apb(i+2,1)==apb(i+3,1)...
            && apb(i-3,2)==apb(i-2,2) && apb(i-2,2)==apb(i-1,2)+1
&& apb(i-1,2)==apb(i,2)+1 && apb(i,2)==apb(i+1,2)+1 &&
apb(i+1,2)==apb(i+2,2)+1 && apb(i+2,2)==apb(i+3,2)+1
        nodobreak=[nodobreak,i-6];
    end % CASO 5
    if apb(i-3,1)==apb(i-2,1)-1 && apb(i-2,1)==apb(i-1,1)-1 &&
apb(i-1,1)==apb(i,1)-1 && apb(i,1)==apb(i+1,1)-1 &&
apb(i+1,1)==apb(i+2,1)-1 && apb(i+2,1)==apb(i+3,1)-1 &&
apb(i+3,1)==apb(i+4,1)...
            && apb(i-3,2)==apb(i-2,2) && apb(i-2,2)==apb(i-1,2)+1
&& apb(i-1,2)==apb(i,2)+1 && apb(i,2)==apb(i+1,2)+1 &&
apb(i+1,2)==apb(i+2,2)+1 && apb(i+2,2)==apb(i+3,2)+1 &&
apb(i+3,2)==apb(i+4,2)+1
        nodobreak=[nodobreak,i-6,i+1-6];
    end % CASO 6
    if apb(i-4,1)==apb(i-3,1)-1 && apb(i-3,1)==apb(i-2,1)-1 &&
apb(i-2,1)==apb(i-1,1)-1 && apb(i-1,1)==apb(i,1)-1 &&
```

```
apb(i,1)==apb(i+1,1)-1 && apb(i+1,1)==apb(i+2,1)-1 &&
apb(i+2,1)==apb(i+3,1)-1 && apb(i+3,1)==apb(i+4,1)...
            && apb(i-4,2)==apb(i-3,2) && apb(i-3,2)==apb(i-2,2)+1
&& apb(i-2,2)==apb(i-1,2)+1 && apb(i-1,2)==apb(i,2)+1 &&
apb(i,2)==apb(i+1,2)+1 && apb(i+1,2)==apb(i+2,2)+1 &&
apb(i+2,2)==apb(i+3,2)+1 && apb(i+3,2)==apb(i+4,2)+1
        nodobreak=[nodobreak,i-6];
    end % CASO 7
    if apb(i-4,1)==apb(i-3,1)-1 && apb(i-3,1)==apb(i-2,1)-1 &&
apb(i-2,1)==apb(i-1,1)-1 && apb(i-1,1)==apb(i,1)-1 &&
apb(i,1)==apb(i+1,1)-1 && apb(i+1,1)==apb(i+2,1)-1 &&
apb(i+2,1)==apb(i+3,1)-1 && apb(i+3,1)==apb(i+4,1)-1 &&
apb(i+4,1)==apb(i+5,1)...
            && apb(i-4,2)==apb(i-3,2) && apb(i-3,2)==apb(i-2,2)+1
&& apb(i-2,2)==apb(i-1,2)+1 && apb(i-1,2)==apb(i,2)+1 &&
apb(i,2)==apb(i+1,2)+1 && apb(i+1,2)==apb(i+2,2)+1 &&
apb(i+2,2)==apb(i+3,2)+1 && apb(i+3,2)==apb(i+4,2)+1 &&
apb(i+4,2)==apb(i+5,2)+1
        nodobreak=[nodobreak,i-6,i+1-6];
    end % CASO 8
    if apb(i-5,1)==apb(i-4,1)-1 && apb(i-4,1)==apb(i-3,1)-1 &&
apb(i-3,1)==apb(i-2,1)-1 && apb(i-2,1)==apb(i-1,1)-1 && apb(i-
1,1)==apb(i,1)-1 && apb(i,1)==apb(i+1,1)-1 &&
apb(i+1,1)==apb(i+2,1)-1 && apb(i+2,1)==apb(i+3,1)-1 &&
apb(i+3,1)==apb(i+4,1)-1 && apb(i+4,1)==apb(i+5,1)...
            && apb(i-5,2)==apb(i-4,2) && apb(i-4,2)==apb(i-3,2)+1
&& apb(i-3,2)==apb(i-2,2)+1 && apb(i-2,2)==apb(i-1,2)+1 && apb(i-
1,2)==apb(i,2)+1 && apb(i,2)==apb(i+1,2)+1 &&
apb(i+1,2)==apb(i+2,2)+1 && apb(i+2,2)==apb(i+3,2)+1 &&
apb(i+3,2)==apb(i+4,2)+1 && apb(i+4,2)==apb(i+5,2)+1
        nodobreak=[nodobreak,i-6];
    end % CASO 9
    if apb(i-5,1)==apb(i-4,1)-1 && apb(i-4,1)==apb(i-3,1)-1 &&
apb(i-3,1)==apb(i-2,1)-1 && apb(i-2,1)==apb(i-1,1)-1 && apb(i-
1,1)==apb(i,1)-1 && apb(i,1)==apb(i+1,1)-1 &&
apb(i+1,1)==apb(i+2,1)-1 && apb(i+2,1)==apb(i+3,1)-1 &&
apb(i+3,1)==apb(i+4,1)-1 && apb(i+4,1)==apb(i+5,1)-1 &&
apb(i+5,1)==apb(i+6,1)...
            && apb(i-5,2)==apb(i-4,2) && apb(i-4,2)==apb(i-3,2)+1
&& apb(i-3,2)==apb(i-2,2)+1 && apb(i-2,2)==apb(i-1,2)+1 && apb(i-
1,2)==apb(i,2)+1 && apb(i,2)==apb(i+1,2)+1 &&
apb(i+1,2)==apb(i+2,2)+1 && apb(i+2,2)==apb(i+3,2)+1 &&
apb(i+3,2)==apb(i+4,2)+1 && apb(i+4,2)==apb(i+5,2)+1 &&
apb(i+5,2)==apb(i+6,2)+1
        nodobreak=[nodobreak,i-6,i+1-6];
    end % CASO 10
    if apb(i-6,1)==apb(i-5,1)-1 && apb(i-5,1)==apb(i-4,1)-1 &&
apb(i-4,1)==apb(i-3,1)-1 && apb(i-3,1)==apb(i-2,1)-1 && apb(i-
```

```
2,1)==apb(i-1,1)-1 && apb(i-1,1)==apb(i,1)-1 &&
apb(i,1)==apb(i+1,1)-1 && apb(i+1,1)==apb(i+2,1)-1 &&
apb(i+2,1)==apb(i+3,1)-1 && apb(i+3,1)==apb(i+4,1)-1 &&
apb(i+4,1)==apb(i+5,1)-1 && apb(i+5,1)==apb(i+6,1)...
            && apb(i-6,2)==apb(i-5,2) && apb(i-5,2)==apb(i-4,2)+1
&& apb(i-4,2)==apb(i-3,2)+1 && apb(i-3,2)==apb(i-2,2)+1 && apb(i-
2,2)==apb(i-1,2)+1 && apb(i-1,2)==apb(i,2)+1 &&
apb(i,2)==apb(i+1,2)+1 && apb(i+1,2)==apb(i+2,2)+1 &&
apb(i+2,2)==apb(i+3,2)+1 && apb(i+3,2)==apb(i+4,2)+1 &&
apb(i+4,2)==apb(i+5,2)+1 && apb(i+5,2)==apb(i+6,2)+1
        nodobreak=[nodobreak,i-6,i+1-6];
    end % CASO 11
end
pos=sort([pos,pos(4)+1-nodobreak]);

%%%%% TÉRMINO BÚSQUEDA PUNTOS INTERPOLACIÓN LINEAL CONTORNO %%%%%

%%%%% INICIO INTERPOLACIÓN LINEAL CONTORNO %%%%%

if length(pos)==9 || length(pos)==10
    c1=[1,1];
    c2=[a,1];
    if pos(4)<pos(5)-1
        f3=fit(dex(pos(3):pos(4),1),dex(pos(3):pos(4),2),'poly1');
        cf3=coeffvalues(f3);
        f4=fit(dex(pos(4):pos(5),1),dex(pos(4):pos(5),2),'poly1');
        cf4=coeffvalues(f4);
        c3=[a,round(cf3(1)*a+cf3(2),tolr)];
        c4=[round(abs((cf4(2)-cf3(2))/(cf3(1)-cf4(1))),tolr),...
        round(abs((cf4(1)*cf3(2)-cf3(1)*cf4(2))/(cf4(1)-
cf3(1))),tolr)];
        c5=[round((b-cf4(2))/cf4(1),tolr),b];
        c6=[1,b];
        f7=fit(dex(pos(7):pos(8),1),dex(pos(7):pos(8),2),'poly1');
        cf7=coeffvalues(f7);
        c7=[1,round(cf7(1)*1+cf7(2),tolr)];
        c8=[round((dex(pos(end),2)-
cf7(2))/cf7(1),tolr),dex(pos(end),2)];
        c9=[1,dex(pos(end),2)];
        Ccontorno=[c1;c2;c3;c4;c5;c6;c7;c8;c9];
    end
    if pos(4)==pos(5)-1
        f3=fit(dex(pos(3):pos(4),1),dex(pos(3):pos(4),2),'poly1');
        cf3=coeffvalues(f3);
        f4=fit(dex(pos(5):pos(6),1),dex(pos(5):pos(6),2),'poly1');
        cf4=coeffvalues(f4);
        c3=[a,round(cf3(1)*a+cf3(2),tolr)];
        c4=[round(abs((cf4(2)-cf3(2))/(cf3(1)-cf4(1))),tolr),...
```

```
        round(abs((cf4(1)*cf3(2)-cf3(1)*cf4(2))/(cf4(1)-
cf3(1))),tolr)];
        c5=[round((b-cf4(2))/cf4(1),tolr),b];
        c6=[1,b];
        f7=fit(dex(pos(8):pos(9),1),dex(pos(8):pos(9),2),'poly1');
        cf7=coeffvalues(f7);
        c7=[1,round(cf7(1)*1+cf7(2),tolr)];
        c8=[round((dex(pos(end),2)-
cf7(2))/cf7(1),tolr),dex(pos(end),2)];
        c9=[1,dex(pos(end),2)];
        Ccontorno=[c1;c2;c3;c4;c5;c6;c7;c8;c9];
    end
end
if length(pos)==6 || length(pos)==7
    c1=[1,1];
    c2=[a,1];
    if pos(4)<pos(5)-1
        f3=fit(dex(pos(3):pos(4),1),dex(pos(3):pos(4),2),'poly1');
        cf3=coeffvalues(f3);
        f4=fit(dex(pos(4):pos(5),1),dex(pos(4):pos(5),2),'poly1');
        cf4=coeffvalues(f4);
        c3=[a,round(cf3(1)*a+cf3(2),tolr)];
        c4=[round(abs((cf4(2)-cf3(2))/(cf3(1)-cf4(1))),tolr),...
        round(abs((cf4(1)*cf3(2)-cf3(1)*cf4(2))/(cf4(1)-
cf3(1))),tolr)];
        c5=[round((b-cf4(2))/cf4(1),tolr),b];
        c6=[1,b];
        Ccontorno=[c1;c2;c3;c4;c5;c6];
    end
    if pos(4)==pos(5)-1
        f3=fit(dex(pos(3):pos(4),1),dex(pos(3):pos(4),2),'poly1');
        cf3=coeffvalues(f3);
        f4=fit(dex(pos(5):pos(6),1),dex(pos(5):pos(6),2),'poly1');
        cf4=coeffvalues(f4);
        c3=[a,round(cf3(1)*a+cf3(2),tolr)];
        c4=[round(abs((cf4(2)-cf3(2))/(cf3(1)-cf4(1))),tolr),...
        round(abs((cf4(1)*cf3(2)-cf3(1)*cf4(2))/(cf4(1)-
cf3(1))),tolr)];
        c5=[round((b-cf4(2))/cf4(1),tolr),b];
        c6=[1,b];
        Ccontorno=[c1;c2;c3;c4;c5;c6];
    end
end
figure(8)
plot([Ccontorno(:,1);1],[Ccontorno(:,2);1],'b')
view(0,90)
axis([-5 a+5 -5 b+5])
hold on
```

```
figure(9)
plot([Ccontorno(:,1);1],[Ccontorno(:,2);1],'k')
view(0,90)
axis([-5 a+5 -5 b+5])
hold on

%%%%% TÉRMINO INTERPOLACIÓN LINEAL CONTORNO %%%%%

%%%%% INICIO BÚSQUEDA PUNTOS INTERPOLACIÓN LINEAL CAVIDADES
INTERNAS %%%%%

for i=2:length(contour)
    cav{i-1}=[contour{i}(:,2),contour{i}(:,1)];
end
totalcav=length(cav);
aux=cell(1,totalcav);
for i=1:totalcav
    auxl{i}=sort(Suavizador([cav{i}(1:end-1,:);cav{i}]));
    T=[];
    for j=1:length(auxl{i})
        if auxl{i}(j)-length(cav{i})==-1
            T=[T,auxl{i}(j)-length(cav{i})+length(cav{i})];
        end
        if auxl{i}(j)-length(cav{i})==0
            T=[T,auxl{i}(j)-length(cav{i})+length(cav{i})];
        end
        if auxl{i}(j)-length(cav{i})>0 && auxl{i}(j)-
length(cav{i})<length(cav{i})-2
            T=[T,auxl{i}(j)-length(cav{i})+1];
        end
    end
    aux{i}=unique(sort(T));
end
if length(aux)>2
    if volfrac>=0.25 && volfrac<0.4 || volfrac>0.5 && volfrac<=0.9
        if aux{end}(1)==aux{end}(2)-1 && aux{end}(3)==aux{end}(4)-1
&& aux{end}(end-1)==aux{end}(end)-1
            aux{end}=[aux{end}(1:4),aux{end}(end-1:end)];
        end
        if aux{end}(1)<aux{end}(2)-1 && aux{end}(2)<aux{end}(3)-1
&& aux{end}(end-1)<aux{end}(end)-1
            aux{end}=[aux{end}(1:2),aux{end}(end)];
        end
        if aux{end}(1)==aux{end}(2)-1 && aux{end}(3)<aux{end}(4)-1
&& aux{end}(end-1)<aux{end}(end)-1
            aux{end}=[aux{end}(1:3),aux{end}(end)];
        end
```

```
        if aux{end}(1)<aux{end}(2)-1 && aux{end}(2)==aux{end}(3)-1
&& aux{end}(end-1)<aux{end}(end)-1
            aux{end}=[aux{end}(1:3),aux{end}(end)];
        end
        if aux{end}(1)<aux{end}(2)-1 && aux{end}(2)<aux{end}(3)-1
&& aux{end}(end-1)==aux{end}(end)-1
            aux{end}=[aux{end}(1:2),aux{end}(end-1:end)];
        end
        if aux{end}(1)==aux{end}(2)-1 && aux{end}(3)==aux{end}(4)-1
&& aux{end}(end-1)<aux{end}(end)-1
            aux{end}=[aux{end}(1:4),aux{end}(end)];
        end
        if aux{end}(1)<aux{end}(2)-1 && aux{end}(2)==aux{end}(3)-1
&& aux{end}(end-1)==aux{end}(end)-1
            aux{end}=[aux{end}(1:3),aux{end}(end-1:end)];
        end
        if aux{end}(1)==aux{end}(2)-1 && aux{end}(3)<aux{end}(4)-1
&& aux{end}(end-1)==aux{end}(end)-1
            aux{end}=[aux{end}(1:3),aux{end}(end-1:end)];
        end
    end
    if volfrac>=0.4 && volfrac<=0.5
        if aux{3}(1)==aux{3}(2)-1 && aux{3}(3)==aux{3}(4)-1 &&
aux{3}(end-1)==aux{3}(end)-1
            aux{3}=[aux{3}(1:4),aux{3}(end-1:end)];
        end
        if aux{3}(1)<aux{3}(2)-1 && aux{3}(2)<aux{3}(3)-1 &&
aux{3}(end-1)<aux{3}(end)-1
            aux{3}=[aux{3}(1:2),aux{3}(end)];
        end
        if aux{3}(1)==aux{3}(2)-1 && aux{3}(3)<aux{3}(4)-1 &&
aux{3}(end-1)<aux{3}(end)-1
            aux{3}=[aux{3}(1:3),aux{3}(end)];
        end
        if aux{3}(1)<aux{3}(2)-1 && aux{3}(2)==aux{3}(3)-1 &&
aux{3}(end-1)<aux{3}(end)-1
            aux{3}=[aux{3}(1:3),aux{3}(end)];
        end
        if aux{3}(1)<aux{3}(2)-1 && aux{3}(2)<aux{3}(3)-1 &&
aux{3}(end-1)==aux{3}(end)-1
            aux{3}=[aux{3}(1:2),aux{3}(end-1:end)];
        end
        if aux{3}(1)==aux{3}(2)-1 && aux{3}(3)==aux{3}(4)-1 &&
aux{3}(end-1)<aux{3}(end)-1
            aux{3}=[aux{3}(1:4),aux{3}(end)];
        end
        if aux{3}(1)<aux{3}(2)-1 && aux{3}(2)==aux{3}(3)-1 &&
aux{3}(end-1)==aux{3}(end)-1
```

```
            aux{3}=[aux{3}(1:3),aux{3}(end-1:end)];
        end
        if aux{3}(1)==aux{3}(2)-1 && aux{3}(3)<aux{3}(4)-1 &&
aux{3}(end-1)==aux{3}(end)-1
            aux{3}=[aux{3}(1:3),aux{3}(end-1:end)];
        end
    end
end
if length(aux)<=2
    if volfrac>=0.25 &&  volfrac<=0.9
        if aux{end}(1)==aux{end}(2)-1 && aux{end}(3)==aux{end}(4)-1
&& aux{end}(end-1)==aux{end}(end)-1
            aux{end}=[aux{end}(1:4),aux{end}(end-1:end)];
        end
        if aux{end}(1)<aux{end}(2)-1 && aux{end}(2)<aux{end}(3)-1
&& aux{end}(end-1)<aux{end}(end)-1
            aux{end}=[aux{end}(1:2),aux{end}(end)];
        end
        if aux{end}(1)==aux{end}(2)-1 && aux{end}(3)<aux{end}(4)-1
&& aux{end}(end-1)<aux{end}(end)-1
            aux{end}=[aux{end}(1:3),aux{end}(end)];
        end
        if aux{end}(1)<aux{end}(2)-1 && aux{end}(2)==aux{end}(3)-1
&& aux{end}(end-1)<aux{end}(end)-1
            aux{end}=[aux{end}(1:3),aux{end}(end)];
        end
        if aux{end}(1)<aux{end}(2)-1 && aux{end}(2)<aux{end}(3)-1
&& aux{end}(end-1)==aux{end}(end)-1
            aux{end}=[aux{end}(1:2),aux{end}(end-1:end)];
        end
        if aux{end}(1)==aux{end}(2)-1 && aux{end}(3)==aux{end}(4)-1
&& aux{end}(end-1)<aux{end}(end)-1
            aux{end}=[aux{end}(1:4),aux{end}(end)];
        end
        if aux{end}(1)<aux{end}(2)-1 && aux{end}(2)==aux{end}(3)-1
&& aux{end}(end-1)==aux{end}(end)-1
            aux{end}=[aux{end}(1:3),aux{end}(end-1:end)];
        end
        if aux{end}(1)==aux{end}(2)-1 && aux{end}(3)<aux{end}(4)-1
&& aux{end}(end-1)==aux{end}(end)-1
            aux{end}=[aux{end}(1:3),aux{end}(end-1:end)];
        end
    end
end
celda=aux;
pares=cell(1,totalcav);
for j=1:totalcav
    q=length(celda{j});
```

```
    if aux{j}(1)==2 && aux{j}(end)==length(cav{j}) &&
(aux{j}(1)<aux{j}(2)-1 || aux{j}(end-1)<aux{j}(end)-1)
        pares{j}=[pares{j},1];
    end
    for i=1:q
        if celda{j}(1)<celda{j}(2)-1
            celda{j}(1)=[];
            q=q-1;
            if q==1 || q==0
                break
                break
            end
        elseif celda{j}(1)==celda{j}(2)-1
            pares{j}=[pares{j},1];
            celda{j}(1)=[];
            celda{j}(1)=[];
            q=q-2;
            if q==1 || q==0
                break
                break
            end
        end
    end
end
vertices=[];
for i=1:totalcav
    vertices=[vertices,(length(aux{i})-sum(pares{i}))];
end
for j=1:totalcav
        if aux{j}(1)==aux{j}(2)-1 && aux{j}(end-1)==aux{j}(end)-1
            orden{j}=zeros(1,2*vertices(j)-2);
            orden{j}(1)=aux{j}(2);
            orden{j}(end)=aux{j}(end-1);
            vp=[];
            par=[];
            zr=[zeros(1),aux{j}(3:length(aux{j})-2),zeros(1)];
            for k=2:length(zr)-1
                if zr(k)==zr(k+1)-1
                    vp=[vp,k,k+1];
                    par=[par,zr(k),zr(k+1)];
                end
            end
            for l=1:length(vp)
                zr(vp(l))=[];
                vp=vp-1;
            end
            rep=[zr(2:end-1),zr(2:end-1)];
            orden{j}(2:end-1)=sort([par,rep]);
```

```
        elseif aux{j}(1)==aux{j}(2)-1 && aux{j}(end-1)<aux{j}(end)-
1
            orden{j}=zeros(1,2*vertices(j)-2);
            orden{j}(1)=aux{j}(2);
            orden{j}(end)=aux{j}(end);
            vp=[];
            par=[];
            zr=[zeros(1),aux{j}(3:length(aux{j})-1),zeros(1)];
            for k=2:length(zr)-1
                if zr(k)==zr(k+1)-1
                    vp=[vp,k,k+1];
                    par=[par,zr(k),zr(k+1)];
                end
            end
            for l=1:length(vp)
                zr(vp(l))=[];
                vp=vp-1;
            end
            rep=[zr(2:end-1),zr(2:end-1)];
            orden{j}(2:end-1)=sort([par,rep]);
        elseif aux{j}(1)<aux{j}(2)-1 && aux{j}(end-1)==aux{j}(end)-
1
            orden{j}=zeros(1,2*vertices(j)-2);
            orden{j}(1)=aux{j}(1);
            orden{j}(end)=aux{j}(end-1);
            vp=[];
            par=[];
            zr=[zeros(1),aux{j}(2:length(aux{j})-2),zeros(1)];
            for k=2:length(zr)-1
                if zr(k)==zr(k+1)-1
                    vp=[vp,k,k+1];
                    par=[par,zr(k),zr(k+1)];
                end
            end
            for l=1:length(vp)
                zr(vp(l))=[];
                vp=vp-1;
            end
            rep=[zr(2:end-1),zr(2:end-1)];
            orden{j}(2:end-1)=sort([par,rep]);
        elseif aux{j}(1)<aux{j}(2)-1 && aux{j}(end-1)<aux{j}(end)-1
&& aux{j}(1)==2 && aux{j}(end)~=length(cav{j})
            orden{j}=zeros(1,2*vertices(j)-2);
            orden{j}(1)=aux{j}(1);
            orden{j}(end)=aux{j}(end);
            vp=[];
            par=[];
            zr=[zeros(1),aux{j}(2:length(aux{j})-1),zeros(1)];
```

```
            for k=2:length(zr)-1
                if zr(k)==zr(k+1)-1
                    vp=[vp,k,k+1];
                    par=[par,zr(k),zr(k+1)];
                end
            end
            for l=1:length(vp)
                zr(vp(l))=[];
                vp=vp-1;
            end
            rep=[zr(2:end-1),zr(2:end-1)];
            orden{j}(2:end-1)=sort([par,rep]);
        elseif aux{j}(1)<aux{j}(2)-1 && aux{j}(end-1)<aux{j}(end)-1
&& aux{j}(1)~=2 && aux{j}(end)==length(cav{j})
            orden{j}=zeros(1,2*vertices(j)-2);
            orden{j}(1)=aux{j}(1);
            orden{j}(end)=aux{j}(end);
            vp=[];
            par=[];
            zr=[zeros(1),aux{j}(2:length(aux{j})-1),zeros(1)];
            for k=2:length(zr)-1
                if zr(k)==zr(k+1)-1
                    vp=[vp,k,k+1];
                    par=[par,zr(k),zr(k+1)];
                end
            end
            for l=1:length(vp)
                zr(vp(l))=[];
                vp=vp-1;
            end
            rep=[zr(2:end-1),zr(2:end-1)];
            orden{j}(2:end-1)=sort([par,rep]);
        elseif aux{j}(1)<aux{j}(2)-1 && aux{j}(end-1)<aux{j}(end)-1
&& aux{j}(1)==2 && aux{j}(end)==length(cav{j})
            orden{j}=zeros(1,2*vertices(j));
            orden{j}(1)=aux{j}(1);
            orden{j}(end)=aux{j}(end);
            vp=[];
            par=[];
            zr=[zeros(1),aux{j}(2:length(aux{j})-1),zeros(1)];
            for k=2:length(zr)-1
                if zr(k)==zr(k+1)-1
                    vp=[vp,k,k+1];
                    par=[par,zr(k),zr(k+1)];
                end
            end
            for l=1:length(vp)
                zr(vp(l))=[];
```

```
                vp=vp-1;
            end
            rep=[zr(2:end-1),zr(2:end-1)];
            orden{j}(2:end-1)=sort([par,rep]);
        elseif aux{j}(1)<aux{j}(2)-1 && aux{j}(end-1)<aux{j}(end)-1
&& aux{j}(1)~=2 && aux{j}(end)~=length(cav{j})
            orden{j}=zeros(1,2*vertices(j)-2);
            orden{j}(1)=aux{j}(1);
            orden{j}(end)=aux{j}(end);
            vp=[];
            par=[];
            zr=[zeros(1),aux{j}(2:length(aux{j})-1),zeros(1)];
            for k=2:length(zr)-1
                if zr(k)==zr(k+1)-1
                    vp=[vp,k,k+1];
                    par=[par,zr(k),zr(k+1)];
                end
            end
            for l=1:length(vp)
                zr(vp(l))=[];
                vp=vp-1;
            end
            rep=[zr(2:end-1),zr(2:end-1)];
            orden{j}(2:end-1)=sort([par,rep]);
        end
end
paq=cell(1,totalcav);
for i=1:totalcav
    if aux{i}(1)==2 && aux{i}(end)==length(cav{i})
        if orden{i}(1)~=3 && orden{i}(end)~=length(cav{i})-1
            paq{i}=orden{i};
        end
        if orden{i}(1)==3 && orden{i}(end)==length(cav{i})-1
        paq{i}=[orden{i},aux{i}(end),aux{i}(1)];
        end
    else paq{i}=[orden{i},aux{i}(end),aux{i}(1)];
    end
end

%%%%% TÉRMINO BÚSQUEDA PUNTOS INTERPOLACIÓN LINEAL CAVIDADES
INTERNAS %%%%%

%%%%% INICIO INTERPOLACIÓN LINEAL CAVIDADES INTERNAS %%%%%

tempo={};
for i=1:length(vertices)
    temp=cell(1,vertices(i));
    if paq{i}(1)==2 && paq{i}(end)==length(cav{i})
```

```
        w=1;
        for k=1:vertices(i)
            temp{k}=cav{i}(paq{i}(w):paq{i}(w+1),:);
            w=w+2;
        end
        tempo=[tempo,temp];
    else temp{vertices(i)}=[cav{i}(paq{i}(2*vertices(i)-
1):end,:);cav{i}(2:paq{i}(2*vertices(i)),:)];
        w=1;
        for k=1:vertices(i)-1
            temp{k}=cav{i}(paq{i}(w):paq{i}(w+1),:);
            w=w+2;
        end
        tempo=[tempo,temp];
    end
end
coefs={};
for i=1:sum(vertices)
    r=fit(tempo{i}(:,1),tempo{i}(:,2),'poly1');
    p=coeffvalues(r);
    coefs=[coefs,p'];
end
for i=1:totalcav
    cfcav{i}=[coefs{(sum(vertices)+1-
sum(vertices(i:end))):sum(vertices(1:i))}];
end
Coordcav={};
for k=1:totalcav
    for i=1:length(cfcav{k})-1
        Ccav{k}(i,:)=[round(abs((cfcav{k}(2,i+1)-
cfcav{k}(2,i))./(cfcav{k}(1,i)-cfcav{k}(1,i+1))),tolr),...
            round(abs(((cfcav{k}(1,i+1).*cfcav{k}(2,i))-
(cfcav{k}(1,i).*cfcav{k}(2,i+1)))./(cfcav{k}(1,i+1)-
cfcav{k}(1,i))),tolr)];
    end
    Ccav{k}(length(cfcav{k}),:)=[round(abs((cfcav{k}(2,end)-
cfcav{k}(2,1))./(cfcav{k}(1,1)-cfcav{k}(1,end))),tolr),...
            round(abs(((cfcav{k}(1,end).*cfcav{k}(2,1))-
(cfcav{k}(1,1).*cfcav{k}(2,end)))./(cfcav{k}(1,end)-
cfcav{k}(1,1))),tolr)];
    Coordcav{k}=Ccav{k};
end
for i=1:totalcav
    figure(8)

plot([Coordcav{i}(:,1);Coordcav{i}(1,1)],[Coordcav{i}(:,2);Coordcav
{i}(1,2)],'b')
    view(0,90)
```

```
    axis([-5 a+5 -5 b+5])
    hold on
    figure(9)

plot([Coordcav{i}(:,1);Coordcav{i}(1,1)],[Coordcav{i}(:,2);Coordcav
{i}(1,2)],'k')
    view(0,90)
    axis([-5 a+5 -5 b+5])
    hold on
end

%%%%% TÉRMINO INTERPOLACIÓN LINEAL CAVIDADES INTERNAS %%%%%

%%%%% TÉRMINO COMANDO SUAVIZADOR %%%%%

%%
%%%%% INICIO COMANDO PROCESADOR %%%%%

Dom=[a-1,b-1];
NODE=[];
auxNODE=[];
for i=1:length(Coordcav)
    auxNODE=[auxNODE;flip(Coordcav{i})];
end
NODE=[Ccontorno;auxNODE];
acumulador={};
t={};
for k=1:length(vertices)
    for j=2:vertices(k)
        t{k}(j-1,:)=[j-1,j];
        t{k}(vertices(k),:)=[vertices(k),1];
    end
    acumulador{k}=t{k};
end
g={};
u=[length(Ccontorno),vertices(1:end-1)];
for i=1:length(u)
    g{i}=[acumulador{i}+sum(u(1:i))];
end
e=[];
for i=1:length(u)
    e=[e;g{i}];
end
for i=1:length(u)
    EDGE=[(1:length(Ccontorno)-
1)',(2:length(Ccontorno))';length(Ccontorno),1;e];
end
[VERT,ETRI,TRIA,TNUM]=refine2(NODE,EDGE,[],[],sizelement);
```

```
figure(10)
patch('faces',TRIA,'vertices',VERT,'facecolor','w','edgecolor','b')
;
hold on;
axis image on;
patch('faces',EDGE,'vertices',NODE,'facecolor','w','edgecolor','g')
;
COORDENADA=[VERT(TRIA(:,1),:),VERT(TRIA(:,2),:),VERT(TRIA(:,3),:)];
CBORDE=[VERT(ETRI(:,1),:),VERT(ETRI(:,2),:)];
Area=0;
for i=1:length(TNUM)

MA=[VERT(TRIA(i,1),:),1;VERT(TRIA(i,2),:),1;VERT(TRIA(i,3),:),1];
    Area=Area+0.5.*abs(det(MA));
end
Atotal=Dom(1,1)*Dom(1,2);
FV=(Area/Atotal)*100;
ptype=1;
D=hooke(ptype,E,nu);
Coord=VERT;
Dof=[1:2:(length(VERT)*2);2:2:(length(VERT)*2)+1]';
nen=3;
ned=6;
for i=1:length(TNUM)

Edofin(i,:)=[Dof(TRIA(i,1),:),Dof(TRIA(i,2),:),Dof(TRIA(i,3),:)];
end
Edofel=zeros(length(TNUM),ned+1);
Edofel(1,1)=1;
for i=1:length(TNUM)-1
    Edofel(1,1)=1;
    Edofel(i+1,1)=i+1;
end
Edof=zeros(length(TNUM),ned+1);
for i=1:length(TNUM)
    Edof(i,:)=[Edofel(i,1),Edofin(i,:)];
end
[Ex,Ey]=coordxtr(Edof,Coord,Dof,nen);
plotpar1=[1,1,1];
figure(11)
eldraw2(Ex,Ey,plotpar1)
t=1;
ep=[ptype,t];
Ke=zeros(ned);
K=zeros(max(max(Dof)));
f=zeros(max(max(Dof)),1);
for i=1:length(TNUM)
    Ke=plante(Ex(i,:),Ey(i,:),ep,D);
```

```
    K=assem(Edof(i,:),K,Ke);
end
f=zeros(max(max(Dof)),1);
funit=-P;
for i=1:length(Coord)
    if Coord(i,:)==[1,Dom(2)+1]
       f(Dof(i,2))=funit;
    end
end
aux1=[];
for i=1:length(Coord)
    if Coord(i,1)==1
        aux1=[aux1 i];
    end
end
aux2=[];
for i=1:length(aux1)
    aux2=[aux2 Dof(aux1(i),1)];
end
bc=sort([aux2',zeros(length(aux2),1);4,0]);
a=solveq(K,f,bc);
Ed=extract(Edof,a);
[Es,Et]=plants(Ex,Ey,ep,D,Ed);
Ef=plantf(Ex,Ey,ep,Es);
plotpar2=[1,2,1];
[sfac]=eldisp2(Ex,Ey,Ed,plotpar2);
figure(12)
eldraw2(Ex,Ey,plotpar1);
plotpar3=[1,4];
elprinc2(Ex,Ey,Es,plotpar3);
for i=1:length(TNUM)
    sigma1(i)=((Es(i,1)+Es(i,2))/2)+sqrt(((Es(i,1)-
Es(i,2))/2)^2)+(Es(i,3)^2));
    sigma2(i)=((Es(i,1)+Es(i,2))/2)-sqrt(((Es(i,1)-
Es(i,2))/2)^2)+(Es(i,3)^2));
    sigmaVM(i)=sqrt((sigma1(i)^2)+(sigma2(i)^2)-
(sigma1(i)*sigma2(i)));
end
figure(13)
colormap(jet)
fill(Ex',Ey',[sigma1',sigma1',sigma1']')
shading interp
colorbar
axis([-5,Dom(1)+5,-5,Dom(2)+5])
axis off
title('\sigma1')
figure(14)
colormap(jet)
```

```
fill(Ex',Ey',[sigma2',sigma2',sigma2']')
shading interp
colorbar
axis([-5,Dom(1)+5,-5,Dom(2)+5])
axis off
title('\sigma2')
figure(15)
colormap(jet)
fill(Ex',Ey',[sigmaVM',sigmaVM',sigmaVM']')
shading interp
colorbar
axis([-5,Dom(1)+5,-5,Dom(2)+5])
axis off
title('\sigma Von Mises')

%%%%% FIN COMANDO PROCESADOR %%%%%

%%
%%%%% INICIO COMANDO EXPORTADOR %%%%%

NODEb=[NODE(2:5,1)+Dom(1),NODE(2:5,2)];
NODEa=flip([(NODE(2:5,1)+Dom(1)-((NODE(2:5,1)+Dom(1)-
Dom(1))*2))+2,NODE(2:5,2)]);
NODEc=[NODEb;NODEa];
auxNODEb=[auxNODE(:,1)+Dom(1),auxNODE(:,2)];
auxNODEa=[(auxNODE(:,1)+Dom(1)-((auxNODE(:,1)+Dom(1)-
Dom(1))*2))+2,auxNODE(:,2)];
auxNODEc=[auxNODEb;auxNODEa];
EDGEb=[[1:7;2:8]';[8,1]];
EDGEa=[e-1;e-1+length(e)];
EDGEc=[EDGEb;EDGEa];
for i=1:length(NODE)-1
    if length(Ccontorno)>6 && NODE(i,2)==Ccontorno(6,2) &
NODE(i+1,2)~=Ccontorno(end)
        auxNODEd=[NODE(i+1,:);NODE(i+2,:)];
        auxNODEe=[auxNODEd(:,1)+Dom(1),auxNODEd(:,2)];
        auxNODEf=[(auxNODEd(2,1)+Dom(1)-((auxNODEd(2,1)+Dom(1)-
Dom(1))*2))+2,auxNODEd(2,2)];
        EDGEb=[[1:7;2:8]';[8,1]];
        EDGEa=[e-1;e-1+length(e)];
        EDGEc=[EDGEb;EDGEa];

EDGEd=[max(max(EDGEc))+1,max(max(EDGEc))+2;max(max(EDGEc))+2,max(ma
x(EDGEc))+3;max(max(EDGEc))+3,max(max(EDGEc))+1];
        NODEprint=[NODEc;auxNODEc;auxNODEe;auxNODEf];
        EDGEprint=[EDGEc;EDGEd];
    end
    if length(Ccontorno)==6
```

```
        EDGEb=[[1:7;2:8]';[8,1]];
        EDGEa=[e+2;e+2+length(e)];
        EDGEc=[EDGEb;EDGEa];
        NODEprint=[NODEc;auxNODEc];
        EDGEprint=EDGEc;
    end
end
[VERTprint,ETRIprint,TRIAprint,TNUMprint]=refine2(NODEprint,EDGEpri
nt,[],[],sizelement);
V0=[VERTprint,zeros(length(VERTprint),1)];
V1=[VERTprint,ones(length(VERTprint),1)];
V=[V0;V1];
T0=TRIAprint;
T1=TRIAprint+max(max(TRIAprint));
TZ=[T1(:,1),T0(:,1),T0(:,2)
    T0(:,2),T1(:,2),T1(:,1)
    T1(:,2),T0(:,2),T0(:,3)
    T0(:,3),T1(:,3),T1(:,2)
    T1(:,3),T0(:,3),T0(:,1)
    T0(:,1),T1(:,1),T1(:,3)];
T=[T0;T1;TZ];
TR=triangulation(T,V);
figure(16)
trisurf(TR)
PT=TR.Points;
CL=TR.ConnectivityList;
stlwrite(TR,'3DPRINT.stl','text')

%%%%% FIN COMANDO EXPORTADOR %%%%%

%%
%%%%% INICIO ENTREGA DE RESULTADOS %%%%%

clc
fprintf('LA FRACCIÓN VOLUMÉTRICA DE LA ESTRUCTURA FINAL ES: %4.4f
[%%]\n',FV)
fprintf('EL DESPLAZAMIENTO MÁXIMO DE LA ESTRUCTURA FINAL ES: %4.4f
[mm]\n',min(a))
fprintf('EL ESFUERZO MÁXIMO AL QUE ESTÁ SOMETIDO LA ESTRUCTURA
FINAL ES DE: %4.4f [MPa]\n',max(sigmaVM))
toc

%%%%% TÉRMINO ENTREGA DE RESULTADOS %%%%%

function [x,U]=Softkillbeso(nelx,nely,volfrac,er,rmin,E,nu,P)
% INITIALIZE
x(1:nely,1:nelx) = 1.; vol=1.; i = 0; change = 1.; penal = 3.;
% START iTH ITERATION
```

```
while change > 0.001
  i = i + 1;  vol = max(vol*(1-er),volfrac);
  if i >1; olddc = dc; end
% FE-ANALYSIS
  [U]=FE(nelx,nely,x,penal,E,nu,P);
% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  [KE] = lk(E,nu);
  c(i) = 0.;
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx   +ely;
      Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;2*n2+2;
2*n1+1;2*n1+2],1);
      c(i) = c(i) + 0.5*x(ely,elx)^penal*Ue'*KE*Ue;
      dc(ely,elx) = 0.5*x(ely,elx)^(penal-1)*Ue'*KE*Ue;
    end
 end
% FILTERING OF SENSITIVITIES
[dc]   = check(nelx,nely,rmin,x,dc);
% STABLIZATION OF EVOLUTIONARY PROCESS
if i > 1; dc = (dc+olddc)/2.; end
% BESO DESIGN UPDATE
[x]    = ADDDEL(nelx,nely,vol,dc,x);
% PRINT RESULTS
if i>10;
 change=abs(sum(c(i-9:i-5))-sum(c(i-4:i)))/sum(c(i-4:i));
end
disp([' It.: ' sprintf('%4i',i) ' Obj.: ' sprintf('%10.4f',c(i))
...
       ' Vol.: ' sprintf('%6.3f',sum(sum(x))/(nelx*nely)) ...
        ' ch.: ' sprintf('%6.3f',change )])
% PLOT DENSITIES
  colormap(gray); imagesc(-x); axis equal; axis tight; axis
off;pause(1e-6);
end
end


%%%%%%%%%% FE-ANALYSIS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [U]=FE(nelx,nely,x,penal,E,nu,P)
[KE] = lk(E,nu);
K = sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*(nely+1));
F = sparse(2*(nely+1)*(nelx+1),1); U =
zeros(2*(nely+1)*(nelx+1),1);
for elx = 1:nelx
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
```

```
    n2 = (nely+1)* elx   +ely;
    edof = [2*n1-1; 2*n1; 2*n2-1; 2*n2; 2*n2+1; 2*n2+2; 2*n1+1;
2*n1+2];
    K(edof,edof) = K(edof,edof) + x(ely,elx)^penal*KE;
  end
end

% DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)
F(2,1) = -P;
fixeddofs   = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
alldofs     = [1:2*(nely+1)*(nelx+1)];
freedofs    = setdiff(alldofs,fixeddofs);

% SOLVING
U(freedofs,:) = K(freedofs,freedofs) \ F(freedofs,:);
U(fixeddofs,:)= 0;
end


%%%%%%%%%% ELEMENT STIFFNESS MATRIX
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [KE]=lk(E,nu)
k=[ 1/2-nu/6   1/8+nu/8 -1/4-nu/12 -1/8+3*nu/8 ...
   -1/4+nu/12 -1/8-nu/8  nu/6       1/8-3*nu/8];
KE = E/(1-nu^2)*[ k(1) k(2) k(3) k(4) k(5) k(6) k(7) k(8)
                  k(2) k(1) k(8) k(7) k(6) k(5) k(4) k(3)
                  k(3) k(8) k(1) k(6) k(7) k(4) k(5) k(2)
                  k(4) k(7) k(6) k(1) k(8) k(3) k(2) k(5)
                  k(5) k(6) k(7) k(8) k(1) k(2) k(3) k(4)
                  k(6) k(5) k(4) k(3) k(2) k(1) k(8) k(7)
                  k(7) k(4) k(5) k(2) k(3) k(8) k(1) k(6)
                  k(8) k(3) k(2) k(5) k(4) k(7) k(6) k(1)];
end


%%%%%%%%%% MESH-INDEPENDENCY FILTER
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dcf]=check(nelx,nely,rmin,x,dc)
dcf=zeros(nely,nelx);
for i = 1:nelx
  for j = 1:nely
    sum=0.0;
    for k = max(i-floor(rmin),1):min(i+floor(rmin),nelx)
      for l = max(j-floor(rmin),1):min(j+floor(rmin),nely)
       fac = rmin-sqrt((i-k)^2+(j-l)^2);
       sum = sum+max(0,fac);
       dcf(j,i) = dcf(j,i) + max(0,fac)*dc(l,k);
      end
    end
```

```
    dcf(j,i) = dcf(j,i)/sum;
  end
end

%%%%%%%%%% OPTIMALITY CRITERIA UPDATE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x]=ADDDEL(nelx,nely,vol,dc,x)
l1 = min(min(dc)); l2 = max(max(dc));
while ((l2-l1)/l2 > 1.0e-5)
   th = (l1+l2)/2.0;
   x = max(0.001,sign(dc-th));
   if sum(sum(x))-vol*(nelx*nely) > 0;
      l1 = th;
   else
      l2 = th;
   end
end
```

UNIVERSIDAD DE CONCEPCIÓN – FACULTAD DE INGENIERÍA
RESUMEN DE MEMORIA DE TÍTULO

Departamento : Departamento de Ingeniería Civil
Carrera : Ingeniería Civil
Nombre del memorista : Felipe Ignacio Millar Medel
Título de la memoria : Generación de estructuras a partir de dominios optimizados topológicamente utilizando técnicas de diseño asistido por ordenador
Fecha de la presentación oral :
Profesor(es) Guía : Dr. Patricio Cendoya H.
Profesor(es) Revisor(es) : Dr. Rodrigo Silva M.
Concepto :
Calificación :

Resumen

La optimización topológica (OT) es fundamental para definir distribuciones eficientes de material en dominios específicos. Sin embargo, las topologías resultantes suelen tener bordes irregulares, inapropiados para aplicaciones industriales. Esta investigación no solo aborda el suavizado de bordes, sino que también aterriza los resultados en la práctica nacional. Se generaron geometrías optimizadas topológicamente con bordes suavizados y adaptados a la construcción real, utilizando bordes lineales, splines y circulares, con diversos materiales y una interfaz adaptada para implementar múltiples tipologías estructurales.

Además, se introdujo una reconstrucción de imágenes ráster derivadas de los resultados topológicos obtenidos con la estrategia de optimización soft kill BESO, utilizando esqueletización y operaciones de regresión y segmentación de ramas. Este tratamiento convierte las geometrías generadas en elementos finitos tipo frames, listos para ser utilizados en otros softwares comerciales de análisis estructural. Así, la metodología desarrollada permite una transición efectiva desde la optimización topológica hasta la aplicación práctica en el diseño y análisis de estructuras.

La validación de esta metodología se realizó mediante dos casos de análisis: una viga larga en voladizo con carga puntual en el extremo libre y una viga simplemente apoyada con carga puntual en el centro, además de un muro sometido a cargas puntuales laterales. Estos casos demostraron la viabilidad y eficacia de la rutina desarrollada, produciendo geometrías con bordes suavizados y adaptados, listas para ser utilizadas en aplicaciones estructurales.