



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

**IDENTIFICACIÓN DE DAÑO EN PIEZAS METÁLICAS A PARTIR DE
ESCANEEO 3D PARA APLICACIONES DE REPARACIÓN CON
MANUFACTURA ADITIVA**

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MECÁNICO

CRISTOPHER ANDRÉS GÓMEZ GALLEGOS

PROFESOR GUÍA:
RUBÉN FERNÁNDEZ URRUTIA

MIEMBROS DE LA COMISIÓN:
MIGUEL CAMPUSANO ARAYA
BENJAMÍN HERRMANN PRIESNITZ

SANTIAGO DE CHILE
2024

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: CRISTOPHER ANDRÉS GÓMEZ GALLEGOS
FECHA: 2024
PROF. GUÍA: RUBÉN FERNÁNDEZ URRUTIA

IDENTIFICACIÓN DE DAÑO EN PIEZAS METÁLICAS A PARTIR DE ESCANEOS 3D PARA APLICACIONES DE REPARACIÓN CON MANUFACTURA ADITIVA

Este trabajo presenta el desarrollo de un sistema para la identificación de daño en piezas metálicas utilizando escaneo 3D y algoritmos de geometría computacional. El objetivo principal es desarrollar un software capaz de detectar y caracterizar daños en piezas metálicas para facilitar su reparación mediante técnicas de manufactura aditiva. Para ello, se utiliza el escaneo 3D para obtener modelos digitales de piezas dañadas, que luego se comparan con modelos 3D de referencia de las piezas en buen estado.

La metodología incluye la generación de una base de datos de evaluación con escaneos 3D existentes y modelos generados sintéticamente, implementación de algoritmos de alineación de mallas geométricas y técnicas de aprendizaje automático, y la realización de operaciones booleanas para la identificación de daño. Se utiliza Python y la librería open3D para el desarrollo del sistema.

Los resultados demuestran un sistema funcional con la capacidad para identificar y caracterizar daños en distintos tipos de piezas, como agujas de turbina Pelton, cilindros, espejos Pelton y placas. Se observa una variabilidad en el éxito de alineación y la identificación del daño dependiendo de las características de las piezas y la calidad de los escaneos 3D.

Este sistema representa un avance en el campo de la reparación de piezas metálicas, ofreciendo una herramienta que permite una rápida identificación de daño y facilitando la reparación mediante técnicas de manufactura aditiva. Sin embargo, se reconoce la necesidad de mejorar la alineación de las mallas y la precisión en la identificación del daño para diferentes geometrías y condiciones de las piezas. La investigación futura podría centrarse en optimizar estos aspectos y expandir la base de datos con más tipos de daños y piezas.

A mi madre.

Agradecimientos

En primer lugar, quiero agradecer a mi madre, Nancy Gallegos, por su apoyo y sacrificio en pos de mi bienestar y el de mis hermanos. También por enseñarme con el ejemplo la resiliencia y la constancia. Agradecer a mi hermana Constanza por su cercanía y las infinitas pequeñas conversaciones sobre temas no tan profundos.

Agradecer a Vanessa por su continuo apoyo y cariño.

Agradecer a mi amigo Mario, con el cual tantas aventuras hemos pasado. Gracias por tu apoyo y por creer más en mí que yo mismo. A mi amiga Natalia por su preocupación y bondad.

Agradecer a cada una de las personas que conocí gracias al laboratorio de robótica. Primero a Luz, Gonzalo, Matías M., Matías P., Rodrigo, Gustavo, que a través de conversaciones y compartir un espacio común, me enseñaron y motivaron cada año académico. Especialmente a Luz, por su dedicación y motivación por enseñarnos y hacernos parte del equipo de robótica. A Gabriel, por su carisma y comedia que sigue hasta el día de hoy en forma de memes. Agradecer a Francisco por las infinitas conversaciones y por su perspectiva sobre la vida y la ética.

Luego, al grupo con el que formamos la nueva generación. José, Lukas, Nico, Leo, Gio, Rodrigo, Ulises, gracias sobre todo por las risas y anécdotas que aliviaban el estrés de la época universitaria. José, gracias por tu amabilidad y cercanía.

Agradecer al profesor Javier por crear oportunidades de aprendizaje dentro de la universidad y abrir las puertas del laboratorio a quien quisiera participar.

Agradecer a la comunidad de robótica: a Joakin, Miguel, Pablo, Rodrigo, con quienes siempre era un gusto juntarse a compartir. Especialmente a Miguel, a quien conocí como sujeto de experimentación para su tesis de doctorado y que en el presente participa como profesor co-guía de este trabajo.

Agradecer al profesor Rubén por su excelente disposición en la docencia y su cercanía humana. Gracias por aceptar ser profesor guía de mi trabajo. También agradecer al profesor Benjamín por aceptar ser parte de la comisión de este trabajo.

Finalmente, agradecer a cada una de las personas que ha sido parte de mi vida. Somos un mosaico de las personas que hemos querido, incluso aquellas que fueron parte de nuestra vida por un breve instante.

Tabla de Contenido

1. Introducción	1
1.1. Antecedentes generales	1
1.1.1. El desafío del desgaste mecánico y su reparación	1
1.1.2. Reparación automatizada con manufactura aditiva	1
1.1.3. Tecnologías de digitalización 3D y procesamiento de datos	2
1.2. Objetivos	2
1.2.1. Objetivo general	2
1.2.2. Objetivos específicos	2
1.2.3. Alcances	2
2. Antecedentes	3
2.1. Desgaste mecánico	3
2.2. Reparación con manufactura aditiva	3
2.3. Reparación por manufactura aditiva automatizada	4
2.4. Tecnologías de escaneo 3D	4
2.5. Representación computacional de geometrías 3D	5
2.5.1. Nubes de puntos	5
2.5.2. Mallas triangulares	6
2.5.3. Grillas de voxels	7
2.6. Conversión entre representaciones de geometrías 3D	8
2.6.1. Coordenadas baricéntricas	8
2.6.2. Muestreo uniforme	9
2.6.3. Voxel Carving	11
2.6.4. Nube de puntos a voxels	13
2.7. Algoritmos de alineación de mallas geométricas	13
2.7.1. Iterative Closest Point	14
2.7.2. Iterative Closest Point aplicado en mallas geométricas	16
2.7.3. Algoritmos de aprendizaje de máquinas	16
2.7.3.1. Redes Neuronales	16
2.7.3.2. PointNetLK	16
2.8. Operaciones booleanas	18
2.9. Algoritmos de agrupamiento	19
2.10. Generación de datos sintéticos	20
3. Metodología	21
3.1. Arquitectura del sistema propuesto	21
3.2. Preparación de la base de datos de evaluación	23

3.3.	Elección del lenguaje de programación	27
3.4.	Implementación de Iterative Closest Point	28
3.5.	Implementación de PointNetLK	30
3.6.	Implementación de operaciones booleanas	30
3.6.1.	Transformación a representación de voxels	31
3.6.2.	Operación booleana de diferencia entre dos conjuntos de voxels	32
3.7.	Identificación del daño utilizando algoritmos de clusterización	33
4.	Resultados	35
4.1.	Resultados del proceso de alineación	35
4.1.1.	Aguja de turbina Pelton	37
4.1.2.	Cilindro	43
4.1.3.	Espejo Pelton	45
4.1.4.	Placas	46
4.1.5.	Modelos generados sintéticamente	48
4.2.	Resultados del proceso de identificación del daño	49
4.2.1.	Aguja de turbina Pelton	50
4.2.2.	Cilindro	54
4.2.3.	Espejo Pelton	57
4.2.4.	Placa	58
4.2.5.	Modelos generados sintéticamente	61
5.	Conclusiones	64
	Bibliografía	65
	Anexos	67
	Anexo A. Código desarrollado	67
A.1.	Manipulaciones geométricas	67
A.2.	Identificación de daño	70

Índice de Tablas

3.1.	Parte de la base de datos de evaluación. Modelos correspondientes a agujas Pelton.	24
3.2.	Parte de la base de datos de evaluación. Modelos correspondientes a cilindros.	25
3.3.	Parte de la base de datos de evaluación. Modelos correspondientes a espejos Pelton.	25
3.4.	Parte de la base de datos de evaluación. Modelos correspondientes a placas. . .	26
3.5.	Parte de la base de datos de evaluación. Modelos generados sintéticamente. . .	27

Índice de Ilustraciones

2.1.	Muestra de daño de erosión en la punta de un inyector de una turbina Pelton [2].	3
2.2.	Escáner 3D comercial. Utilizando un arreglo de cámaras y sensores de profundidad es posible generar un modelo 3D digital del objeto escaneado. Imagen de VIUScan.	5
2.3.	Nubes de puntos sencillas que representan un cilindro y una semi-esfera. . . .	5
2.4.	Representación mediante malla triangular.	6
2.5.	Grillas de voxels. A la izquierda una grilla completa de voxels. A la derecha, una esfera de baja resolución representada con voxels.	7
2.6.	Representación gráfica de las coordenadas baricéntricas. El punto de color verde claro se representa como una combinación lineal de los vértices del triángulo. Fuente [7]	9
2.7.	Muestreo uniforme de puntos sobre una malla de triángulos. Fuente [8]	10
2.8.	Disposición de cámaras virtuales entorno a la malla geométrica. Cada punto de color representa la posición de una cámara virtual con orientación hacia el centro de la malla geométrica.	11
2.9.	Ejemplo de algunas imágenes de profundidad obtenidas desde la proyección del modelo aguja Pelton.	12
2.10.	Diagrama simplificado del proceso de <i>voxel carving</i> . Fuente [9].	12
2.11.	Resultado de la conversión de nube de puntos a grilla de voxels. En (1) la nube de puntos contenía 100 puntos. En (2) la nube de puntos contenía 1000 puntos. En (3) la nube de puntos contenía 10.000 puntos. En (4) la nube de puntos contenía 100.000 puntos.	13
2.12.	El algoritmo ICP busca alinear dos nubes de puntos. En la izquierda, se tienen dos nubes de puntos no alineadas de color rojo y azul. En la derecha, las dos nubes de puntos se encuentran alineadas. Fuente [11].	14
2.13.	Arquitectura de la red PointNetLk. Fuente [12]	17
2.14.	Ejemplo gráfico de operaciones booleanas entre un cubo y una esfera. De izquierda a derecha: intersección, diferencia y unión. Imagen de Graphisoft.	18
2.15.	Ejemplo gráfico de operaciones booleanas entre dos grillas de voxels. Fuente [14].	19
3.1.	Arquitectura del sistema de identificación de daños. Fuente: Elaboración propia	22
3.2.	Proceso de alineación de nubes de puntos utilizando el algoritmo ICP. De izquierda a derecha: inicialmente, ambas nubes de puntos se sitúan en una posición arbitraria, determinada por la ubicación de la malla original. Seguidamente, se trasladan ambas nubes al origen del sistema de referencia. Finalmente, tras completar todas las iteraciones del algoritmo ICP, las nubes de puntos quedan alineadas.	29

3.3.	Resultados de la conversión a voxels de la pieza original y la pieza dañada. En (a) se visualiza la representación de un volumen denso de voxels correspondiente a la pieza original. En (b) se visualiza la superficie de voxels correspondientes al escaneo de la pieza dañada.	31
3.4.	Conjunto de voxels correspondientes a la pieza dañada y la pieza original. Las dos representaciones se encuentran alineadas. Los voxels de color amarillo corresponden a la pieza original. Los voxels de la pieza dañada han sido pintados con colores aleatorios para una mejor visualización.	32
3.5.	Corte longitudinal sobre el conjunto de voxels obtenidos desde la operación booleana de diferencia.	33
3.6.	Voxels coloreados según la etiqueta obtenida con el algoritmo DBSCAN. . . .	33
3.7.	Voxels correspondiente a la identificación del daño junto con el escaneo 3D de la pieza dañada. En (1) se muestra el escaneo original en formato malla. En (2) se muestra el escaneo original en formato malla y la identificación del daño en formato voxel. El coloreado de gradiente se usa solo por motivos de visualización.	34
4.1.	Raíz del error cuadrático medio obtenida utilizando el algoritmo ICP en la alineación de las mallas de la base de datos de evaluación.	36
4.2.	Raíz del error cuadrático medio obtenida utilizando el algoritmo ICP en la alineación de las mallas de la base de datos sintética.	36
4.3.	Raíz del error cuadrático medio obtenida utilizando ICP vs obtenida utilizando PointNetLk	37
4.4.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-E-S	38
4.5.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-BR-S	38
4.6.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-DCR-S	39
4.7.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-A-S	39
4.8.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-Aps-S	40
4.9.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-B-S	40
4.10.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-P-S	41
4.11.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-PR-S	41

4.12.	Visualización de la alineación del modelo con daño y el modelo original. Dependiendo de la distribución del daño, la alineación resultante se desplaza de la alineación esperada. En (a) el daño se encuentra distribuido a lo largo de la superficie y la alineación es buena. En (b) el daño se encuentra centrado en una zona y la alineación sufre un desplazamiento considerable de la posición esperada. En (c) el daño se encuentra concentrado hacia la punta de la aguja y sufre un desplazamiento grande de la posición esperada.	42
4.13.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos C-AP-RA	43
4.14.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos C-AP-RC	43
4.15.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos C-Aps-R	44
4.16.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos C-B-R	44
4.17.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos C-P-R	44
4.18.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos EIP-B-S	45
4.19.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos EIP-R1-S	45
4.20.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos EIP-R2-S	46
4.21.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos PL-Aps-R	47
4.22.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos PL-B-R	47
4.23.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos PL-E-R	48
4.24.	Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos PL-P-R	48

4.25.	Error porcentual en el volumen del daño identificado para cada modelo en la base de datos de evaluación. Cada barra representa el error acumulado por zona dañada, con distintos colores para cada zona en modelos como el A-E-S. Modelos de piezas no convexas y placas, así como algunos escaneos de cilindros, son omitidos debido a limitaciones en la obtención de la envoltura convexa y errores de malla.	49
4.26.	Error porcentual en el volumen del daño identificado para modelos de la base de datos sintética, utilizando la diferencia de volumen entre las mallas geométricas de las piezas original y dañada como referencia.	50
4.27.	Identificación de daño para A-Aps-S	51
4.28.	Identificación de daño para A-A-S	51
4.29.	Identificación de daño para A-B-S	52
4.30.	Identificación de daño para A-BR-S	52
4.31.	Identificación de daño para A-DCR-S	53
4.32.	Identificación de daño para A-E-S	53
4.33.	Identificación de daño para A-P-S	54
4.34.	Identificación de daño para A-PR-S	54
4.35.	Identificación de daño para C-AP-RA	55
4.36.	Identificación de daño para C-Aps-R	55
4.37.	Identificación de daño para C-P-R	56
4.38.	Identificación de daño para C-B-R	56
4.39.	Identificación de daño para C-AP-RC	57
4.40.	Identificación de daño para EIP-B-S	57
4.41.	Identificación de daño para EIP-R1-S	58
4.42.	Identificación de daño para EIP-R2-S	58
4.43.	Identificación de daño fallido PL-Aps-R	59
4.44.	Identificación de daño fallido PL-B-R	59
4.45.	Identificación de daño PL-E-R	60
4.46.	Identificación de daño fallido PL-P-R	60
4.47.	Identificación de daño en mallas correspondientes a eje con engranaje.	61
4.48.	Identificación de daños en mallas correspondientes a conos generados sintéticamente.	62
4.49.	Identificación de daño en malla BG-T.	63

Capítulo 1

Introducción

1.1. Antecedentes generales

En esta sección se presentan los antecedentes generales necesarios para comprender la motivación e importancia de este trabajo de título en el contexto de la ingeniería.

1.1.1. El desafío del desgaste mecánico y su reparación

Chile, siendo un país fuertemente ligado a la industria minera, ha experimentado durante los últimos años una creciente necesidad de mejorar los métodos de reparación y mantenimiento de maquinaria y equipos pesados. Esta demanda ha llevado al desarrollo de nuevas técnicas y tecnologías orientadas a extender la vida útil de dichos equipos, reduciendo así los tiempos de inactividad y los costos asociados a la sustitución de piezas defectuosas o desgastadas. En este contexto, la manufactura aditiva ha emergido como una solución prometedora, ya que permite la reparación de componentes metálicos de manera más rápida y económica que el reemplazo total de la pieza.

1.1.2. Reparación automatizada con manufactura aditiva

El Programa de Innovación en Manufactura Avanzada (IMA+) es un programa conformado por el gobierno de Chile, universidades y empresas de Chile [1]. Tiene como objetivos el desarrollo de soluciones tecnológicas de manufactura avanzada, promover estrategias de escalamiento y comercialización, fortalecer proveedores locales y formar capital humano especializado en manufactura avanzada. Este trabajo de título se enmarca en uno de los proyectos pertenecientes al programa es el llamado "Sistema Robotizado de Recuperación de Piezas Metálicas mediante Manufactura Aditiva". En este proyecto se busca la implementación de un sistema automatizado de reparación de piezas metálicas. Para lograr esto se requiere la utilización de sistemas digitales de detección y caracterización del daño y desgaste, procesos de soldadura metálica robotizada y técnicas de manufactura aditiva metálica.

En este trabajo de título se busca implementar un sistema computacional para la identificación de daño en piezas metálicas a partir de escaneos 3D de dicha pieza, para ser usado en el sistema de reparación automatizada con manufactura aditiva.

1.1.3. Tecnologías de digitalización 3D y procesamiento de datos

El rol de la digitalización 3D y los algoritmos de procesamiento de datos es fundamental en este proyecto. Las tecnologías de escaneo 3D permiten obtener un modelo tridimensional preciso de la pieza metálica a reparar, revelando detalles intrincados del desgaste y daño presentes. Una vez obtenido este modelo, se requieren algoritmos avanzados de procesamiento para analizar los datos 3D y distinguir las áreas dañadas de las sanas.

Existen algoritmos que permiten alinear y comparar la malla geométrica obtenida por el escaneo 3D con un modelo base de la pieza sin daño, revelando las discrepancias que se corresponden con el desgaste. Por otro lado, los algoritmos de aprendizaje automático, especialmente los que operan en dominios 3D, pueden ser entrenados para identificar patrones de daño comunes y categorizar el tipo y grado de desgaste presente. De esta forma, estas tecnologías conjuntas prometen una identificación precisa y confiable del daño en piezas metálicas.

1.2. Objetivos

1.2.1. Objetivo general

Desarrollar un software para la identificación de desgaste y daño en piezas metálicas a partir de un escaneo 3D para ser usado en un sistema de reparación robótica.

1.2.2. Objetivos específicos

1. Crear una base de datos de evaluación a partir de escaneos 3D existentes y modelos generados sintéticamente.
2. Implementar algoritmo ICP y algoritmos de aprendizaje de máquinas para la alineación de mallas geométricas.
3. Configurar los algoritmos seleccionados para trabajar con modelos de piezas metálicas.
4. Extraer una representación 3D del daño que pueda ser utilizada en un software de cálculo de trayectorias para un brazo soldador.
5. Evaluar el desempeño del sistema sobre la base de datos creada.

1.2.3. Alcances

1. Los inputs del sistema corresponden a: un modelo 3D (en formato STL o OBJ) correspondiente al escaneo 3D de la pieza y el modelo 3D base de la pieza sin daño.
2. El output del sistema corresponde a un modelo 3D correspondiente a la diferencia booleana entre los dos modelos.
3. Existe una base de datos de piezas escaneadas. No es parte del trabajo recolectar y escanear piezas dañadas.
4. Sí está en el alcance la generación de modelos 3D sintéticos para la evaluación de casos de mayor interés.

Capítulo 2

Antecedentes

2.1. Desgaste mecánico

El desgaste mecánico es un fenómeno que afecta a los componentes mecánicos de distintos sistemas y máquinas. La interacción física entre los componentes y en especial sus superficies, genera cambios estructurales que pueden comprometer la funcionalidad y rendimiento de los componentes. Este fenómeno puede ser causado por una variedad de mecanismos, incluyendo la abrasión, la adhesión, la fatiga superficial, y la corrosión. El desgaste mecánico puede llevar a la falla prematura de los componentes, lo que puede resultar en costos significativos en términos de reemplazo de piezas y tiempo de inactividad del sistema. En la Figura 2.1 se muestra un ejemplo de el daño de erosión en un inyector de una turbina Pelton. El daño corresponde a una leve erosión en la punta del inyector.

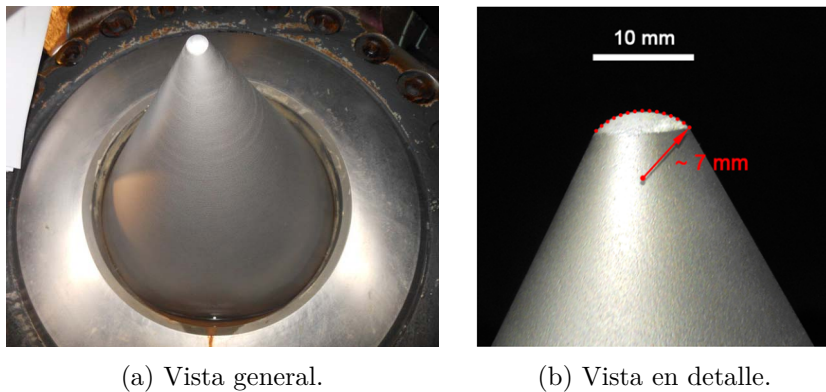


Figura 2.1: Muestra de daño de erosión en la punta de un inyector de una turbina Pelton [2].

2.2. Reparación con manufactura aditiva

La manufactura aditiva es una tecnología que permite la fabricación de objetos tridimensionales a partir de la adición de material capa por capa. Esta tecnología ha ganado popularidad en los últimos años debido a su capacidad para producir geometrías complejas, personalización a gran escala y reducción de residuos [3]. La manufactura aditiva de metales, en particular, ha demostrado ser útil en una variedad de aplicaciones, incluyendo la producción de componentes aeroespaciales, dispositivos médicos, y piezas de automóviles [4].

La reparación con manufactura aditiva es un método para la restauración de componentes metálicos dañados. Este proceso implica el uso de tecnologías de impresión 3D o soldadura para depositar material en las áreas dañadas de un componente, permitiendo la reparación precisa de defectos específicos [5]. Este enfoque puede ser particularmente útil para la reparación de componentes de alto valor que son costosos o difíciles de reemplazar.

2.3. Reparación por manufactura aditiva automatizada

La reparación por manufactura aditiva automatizada combina las ventajas de la manufactura aditiva y la automatización para proporcionar soluciones de reparación eficientes y precisas. Este enfoque puede ser particularmente útil para la reparación de componentes complejos o de alto valor, donde la precisión y la eficiencia son críticas. La reparación por manufactura aditiva automatizada implica el uso de sistemas robóticos para escanear y reparar componentes, utilizando datos de escaneo 3D para guiar el proceso de reparación.

La automatización del proceso de soldadura ha sido un área de investigación activa en los últimos años. La automatización puede mejorar la eficiencia y la calidad de la soldadura, reduciendo al mismo tiempo los riesgos asociados con la soldadura manual. Los sistemas de soldadura automatizados pueden utilizar una variedad de tecnologías, incluyendo la robótica, la visión computacional, y el control automático, para realizar tareas de soldadura de manera precisa y consistente [6].

2.4. Tecnologías de escaneo 3D

Las tecnologías de escaneo 3D (Figura 2.2) son herramientas esenciales para la captura de geometrías de objetos en el mundo real. Estas tecnologías utilizan una variedad de métodos, incluyendo la luz estructurada, la fotogrametría, y el láser, para capturar datos de superficie de alta resolución de objetos. Los datos de escaneo 3D pueden ser utilizados para una variedad de aplicaciones, incluyendo la inspección de calidad, la ingeniería inversa, y la fabricación aditiva. En la actualidad ha aumentado el acceso a estas tecnologías, por ejemplo un teléfono inteligente de la empresa Apple contiene un sensor auxiliar con el cual se pueden realizar escaneos 3D.

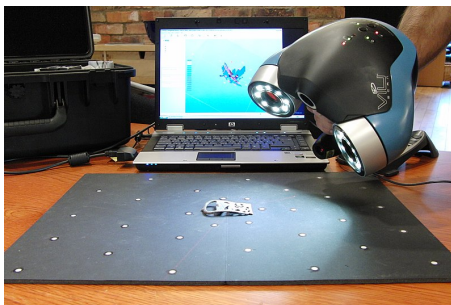


Figura 2.2: Escáner 3D comercial. Utilizando un arreglo de cámaras y sensores de profundidad es posible generar un modelo 3D digital del objeto escaneado. Imagen de VIUScan.

2.5. Representación computacional de geometrías 3D

La representación computacional de geometrías 3D es un aspecto crítico de muchas aplicaciones de ingeniería y diseño. Las representaciones de geometría 3D pueden ser utilizadas para modelar y analizar formas y estructuras complejas, y son esenciales para tecnologías como la fabricación aditiva y el diseño asistido por computador. Las representaciones de geometría 3D pueden ser basadas en mallas, que representan la geometría como un conjunto de polígonos interconectados, o pueden ser basadas en volúmenes, que representan la geometría como un conjunto de volúmenes discretos. Estos modelos 3D sirven como base para el análisis computacional de este trabajo de título, permitiendo aplicar algoritmos avanzados para detectar y cuantificar las áreas dañadas

2.5.1. Nubes de puntos

Las nubes de puntos son una forma común de representar geometrías 3D en aplicaciones de escaneo 3D. Una nube de puntos es simplemente un conjunto de puntos en un espacio tridimensional, cada uno de los cuales representa un punto en la superficie de un objeto (Figura 2.3). Las nubes de puntos pueden ser utilizadas para representar geometrías complejas y detalladas, pero pueden ser difíciles de manejar debido a su tamaño y complejidad.



Figura 2.3: Nubes de puntos sencillas que representan un cilindro y una semi-esfera.

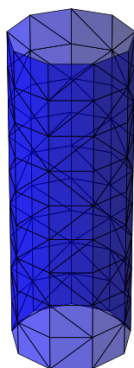
Matemáticamente una nube de puntos se denota como \mathcal{P} y se define como:

$$\mathcal{P} = \{p_i \mid p_i \in \mathbb{R}^3, i = 1, 2, \dots, N\} \quad (2.1)$$

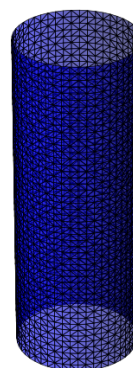
- Cada p_i es un vector en \mathbb{R}^3 , representando la posición de un punto en el espacio.
- N es el número total de puntos en la nube.
- La naturaleza *no ordenada* de \mathcal{P} implica que no hay un orden específico o secuencia en la que los puntos deben ser considerados.

2.5.2. Mallas triangulares

Las mallas triangulares son otra forma común de representar geometrías 3D. A diferencia de las nubes de puntos, que representan la geometría como un conjunto de puntos discretos, las mallas triangulares representan la geometría como un conjunto de triángulos. Cada triángulo está definido por tres vértices y sus correspondientes aristas. Si se cumplen ciertas restricciones entre los triángulos de la malla se puede lograr representar el objeto 3D como una superficie suave y continua. Esto puede ser útil para representar formas suaves y curvas, y puede facilitar ciertos tipos de análisis y operaciones de geometría.



(a) Malla triangular con forma cilíndrica.



(b) Malla triangular con forma cilíndrica. Al utilizar una cantidad mayor de triángulos pequeños se obtiene una mejor aproximación a la pieza real.

Figura 2.4: Representación mediante malla triangular.

Matemáticamente, una **malla triangular** es una colección de vértices, aristas y caras que definen la forma de un objeto tridimensional.

- Se puede representar como un conjunto M definido por:

$$M = (V, E, F) \quad (2.2)$$

donde V representa el conjunto de vértices, E el conjunto de aristas y F el conjunto de caras.

- Cada vértice $v \in V$ es un punto en \mathbb{R}^3 .

- Una arista $e \in E$ es un segmento de línea que conecta dos vértices.
- Una cara $f \in F$ es un polígono formado por aristas y vértices, típicamente triángulos o cuadriláteros.

La representación mediante mallas triangulares tiene varias ventajas. En primer lugar, permite capturar detalles finos de la geometría al dividir la superficie en triángulos más pequeños. Además, al ser una estructura discreta, se puede almacenar y procesar eficientemente en aplicaciones de modelado y renderizado en tiempo real.

Las mallas cerradas y *manifolds* son conceptos importantes en el contexto de las mallas triangulares para representar superficies en 3D de manera precisa y útil. Una malla cerrada es aquella donde cada arista es compartida por exactamente dos triángulos, lo que significa que no hay huecos o bordes libres; esta propiedad asegura que la malla encierra un volumen, lo cual es fundamental para representar sólidos en aplicaciones como la impresión 3D y la simulación física. Por otro lado, un *manifold* es una malla que, en cada vecindad de un punto, se asemeja a un espacio euclidiano. En términos más simples, esto significa que la malla debe comportarse localmente como una superficie continua y sin cruces, permitiendo que cada punto tenga un vecindario bien definido que se parezca a un disco plano. Las mallas que son *manifolds* aseguran la consistencia y la facilidad de aplicar algoritmos de geometría y física.

2.5.3. Grillas de voxels

Las grillas de voxels son una forma de representación tridimensional que divide el espacio en cubos regulares, conocidos como voxels. Cada voxel puede almacenar información, como color, densidad, o cualquier otro atributo relacionado con el volumen en el espacio que representa. Esta técnica es ampliamente utilizada en una variedad de campos, incluyendo computación gráfica, análisis médico (como la tomografía computarizada), y modelado geométrico, entre otros.

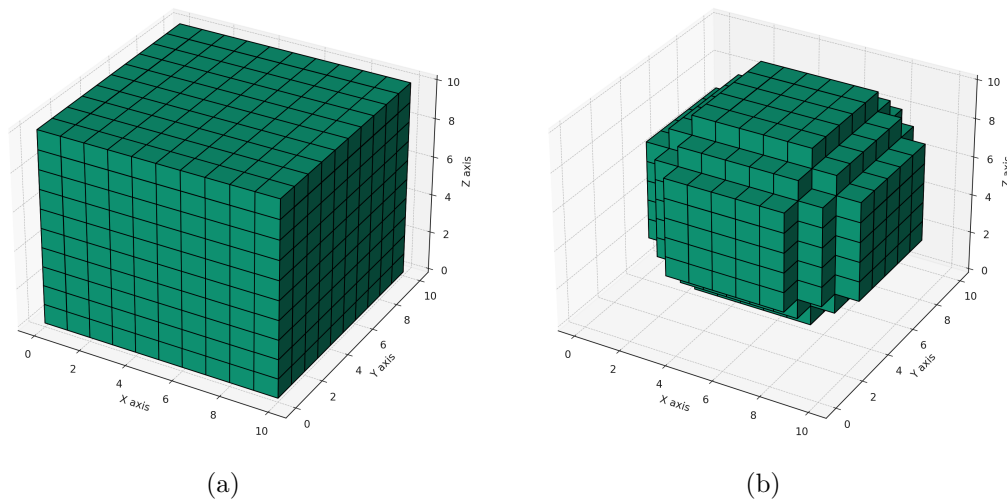


Figura 2.5: Grillas de voxels. A la izquierda una grilla completa de voxels. A la derecha, una esfera de baja resolución representada con voxels.

En el contexto del modelado 3D y el escaneo 3D, las grillas de voxels proporcionan una manera de convertir geometrías complejas en un formato discreto que es más fácil de manipular y analizar. Los voxels son particularmente útiles para modelar objetos internos y externos,

permitiendo la visualización y análisis de estructuras internas en 3D, lo cual es difícil con otras representaciones como mallas o nubes de puntos.

Matemáticamente, un conjunto de voxels V se puede representar como:

$$V = \{v_{ijk} \mid v_{ijk} \in \mathbb{R}^3, i = 1, \dots, N_x; j = 1, \dots, N_y; k = 1, \dots, N_z\} \quad (2.3)$$

- Donde v_{ijk} representa el vóxel en la posición (i, j, k) en una cuadrícula 3D.
- Las dimensiones N_x , N_y y N_z definen la resolución de la cuadrícula de voxels en las direcciones x , y y z , respectivamente.

Una desventaja de la representación con voxels corresponde a su uso intensivo de almacenamiento. La cantidad de memoria necesaria para almacenar una grilla de voxels aumenta al cubo con respecto a un aumento de la resolución.

2.6. Conversión entre representaciones de geometrías 3D

La conversión entre diferentes representaciones de geometrías 3D permite aprovechar las ventajas de una representación sobre otra en tareas específicas. Por ejemplo, con técnicas de fotogrametría es posible generar nubes de puntos. Sin embargo, para la visualización humana una nube de puntos no es tan útil como una representación de malla geométrica. Para solucionar esto, existen algoritmos para generar una malla geométrica a partir de una nube de puntos.

Existen algoritmos para convertir desde cada una de las representaciones a otras. Cada algoritmo tiene sus ventajas y desventajas, como el tiempo de procesamiento o la precisión de la conversión. Es importante destacar que al convertir un conjunto de datos desde una representación a otra es posible que ocurra una pérdida de información. Por esta razón es recomendable evaluar caso a caso si es necesaria la conversión.

En esta sección se describen los siguientes algoritmos que se utilizan en el desarrollo de este trabajo de título:

- Muestreo uniforme - Conversión de una malla geométrica a una nube de puntos.
- Voxel Carving - Conversión de una malla geométrica a un volumen denso de voxels.
- Nube de puntos a voxels.

Como se mencionó en el ejemplo del primer párrafo, existen algoritmos para convertir nubes de puntos a mallas geométricas. Considerando que en el desarrollo de este trabajo de título no se utilizan dichos algoritmos, no existe una sección dedicada a ellos.

2.6.1. Coordenadas baricéntricas

Las coordenadas baricéntricas (Figura 2.6) son un sistema de coordenadas utilizado para expresar la posición de un punto dentro de un triángulo en términos de los vértices del

triángulo. En un triángulo con vértices A , B , y C , cualquier punto P dentro del triángulo (o en su plano) puede representarse como una combinación lineal de los vértices:

$$P = \alpha A + \beta B + \gamma C \quad (2.4)$$

donde α , β , y γ son las coordenadas baricéntricas de P , que cumplen las condiciones:

$$\alpha + \beta + \gamma = 1 \quad \text{y} \quad \alpha, \beta, \gamma \geq 0 \quad (2.5)$$

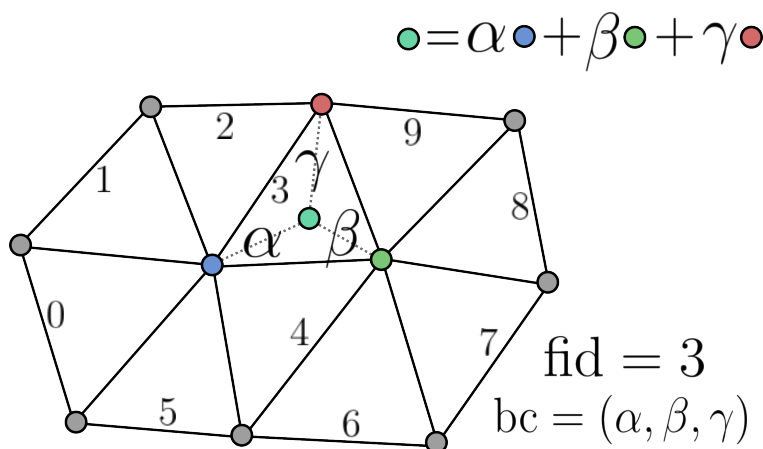


Figura 2.6: Representación gráfica de las coordenadas baricéntricas. El punto de color verde claro se representa como una combinación lineal de los vértices del triángulo. Fuente [7]

Para realizar el muestreo uniforme en el triángulo seleccionado definimos α , β , y γ en términos de dos números aleatorios $r1$ y $r2$:

$$\alpha = 1 - \sqrt{r1} \quad (2.6)$$

$$\beta = \sqrt{r1} \cdot (1 - r2) \quad (2.7)$$

$$\gamma = \sqrt{r1} \cdot r2 \quad (2.8)$$

Se pueden utilizar estas definiciones para calcular la posición de un punto P dentro del triángulo usando la ecuación 2.4. Sustituyendo α , β , y γ con las expresiones dadas:

$$P = (1 - \sqrt{r1}) \cdot A + (\sqrt{r1} \cdot (1 - r2)) \cdot B + (\sqrt{r1} \cdot r2) \cdot C \quad (2.9)$$

Este punto P representa una ubicación dentro del triángulo determinada de manera aleatoria y proporcional a las áreas formadas entre P y los vértices del triángulo, asegurando que la distribución de los puntos muestreados sea uniforme y acorde a las dimensiones del triángulo.

2.6.2. Muestreo uniforme

Para generar una nube de puntos a partir de una malla geométrica se busca muestrear puntos aleatorios que sean parte de los triángulos de la malla geométrica. En la Figura 2.7

se visualiza el resultado de muestrear puntos sobre una malla. El proceso de muestreo debe considerar una probabilidad proporcional al área de los triángulos para conservar la forma de la malla.

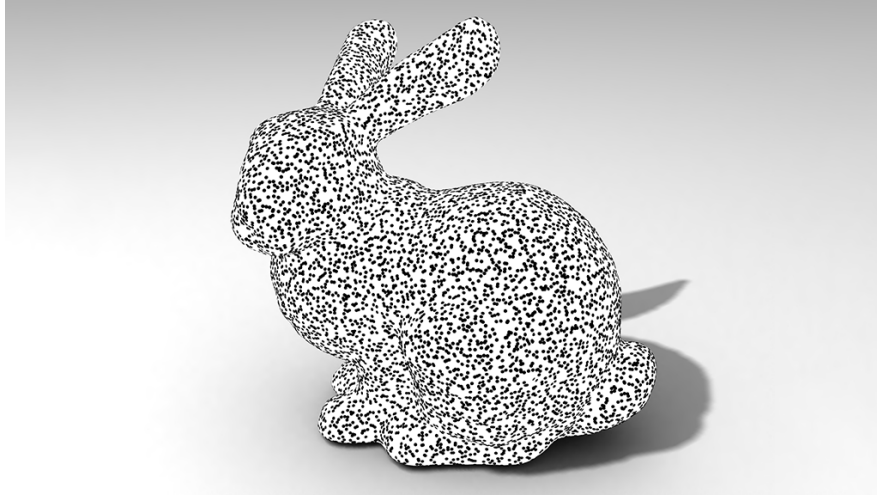


Figura 2.7: Muestreo uniforme de puntos sobre una malla de triángulos.
Fuente [8]

1. Construcción de la Función de Distribución Acumulada (FDA):

- Considerando una malla con N triángulos, y denotando el área del triángulo i como A_i .
- Se calcula el área total:

$$A_{\text{total}} = \sum_{i=1}^N A_i \quad (2.10)$$

- Se construye la FDA:

$$FDA(i) = \frac{\sum_{j=1}^i A_j}{A_{\text{total}}} \quad (2.11)$$

2. Selección de Triángulos:

- Se genera un número aleatorio r en el rango $[0, 1]$.
- Se selecciona el primer triángulo cuya FDA es mayor o igual a r .
- Sobre el triángulo correspondiente se va a muestrear un punto.

3. Muestreo dentro del Triángulo Seleccionado:

- Se generan dos números aleatorios r_1 y r_2 para las coordenadas baricéntricas dentro del triángulo.
- Se calcula la posición del punto utilizando las coordenadas baricéntricas y los vértices del triángulo.

2.6.3. Voxel Carving

Para lograr una representación tridimensional precisa de un objeto original intacto, se emplea un método conocido como tallado de voxels [9] (o *voxel carving* en inglés). Este proceso inicia con la creación de un cubo, repleto completamente de voxels.

La siguiente etapa implica la utilización de imágenes de profundidad. Estas imágenes se generan proyectando la malla del objeto original y capturándolas a través de una serie de cámaras virtuales. Estas cámaras se distribuyen alrededor del objeto, posicionadas en puntos estratégicos sobre una malla esférica que envuelve al objeto. La Figura 2.8 muestra la disposición de las cámaras entorno a la malla geométrica. El número de cámaras es ajustable, y cada una proporciona una imagen de profundidad única.

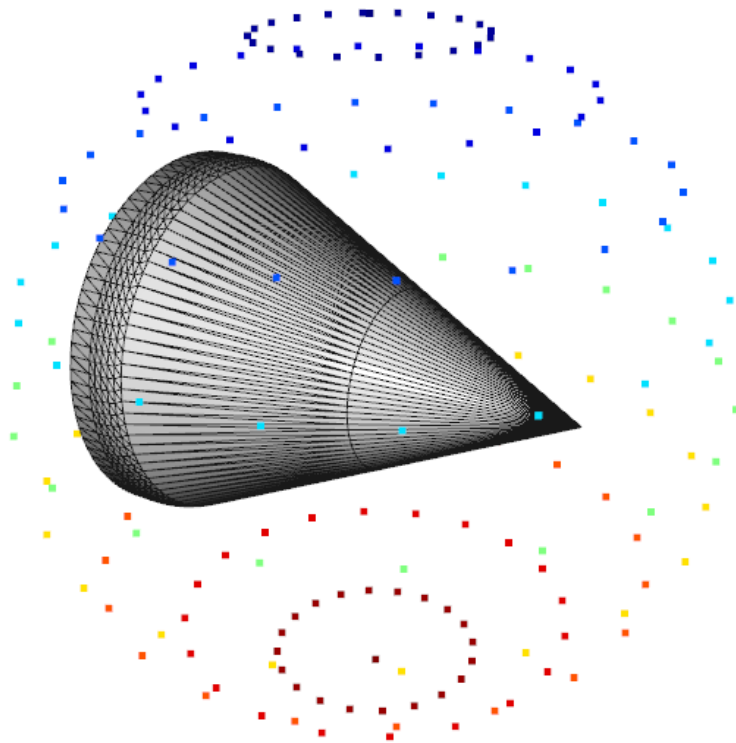


Figura 2.8: Disposición de cámaras virtuales entorno a la malla geométrica. Cada punto de color representa la posición de una cámara virtual con orientación hacia el centro de la malla geométrica.

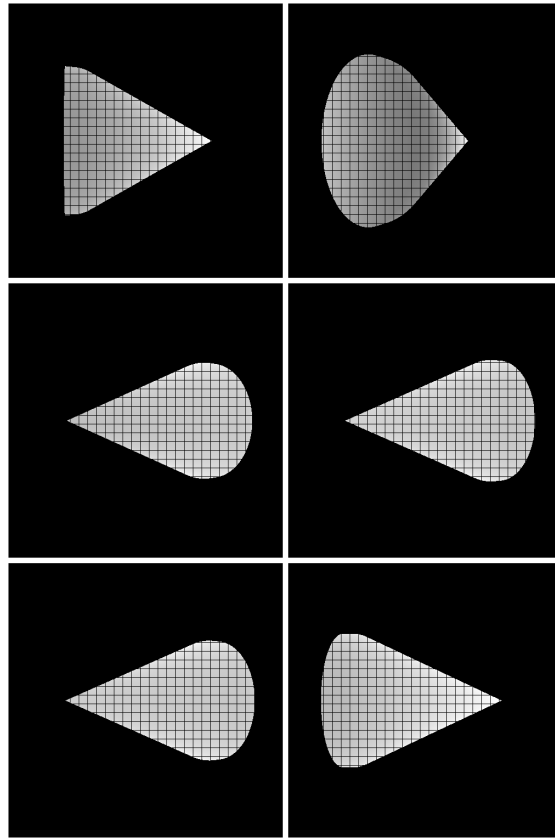


Figura 2.9: Ejemplo de algunas imágenes de profundidad obtenidas desde la proyección del modelo aguja Pelton.

Durante el proceso de tallado, estas imágenes de profundidad (Figura 2.9) se proyectan sobre el cubo lleno de voxels. En la Figura 2.10 se muestra un ejemplo simplificado en 2D. Los voxels que no coinciden con la proyección, es decir, aquellos que no forman parte de la imagen de profundidad, se eliminan. Esta eliminación se repite con cada imagen de profundidad obtenida por las diferentes cámaras virtuales.

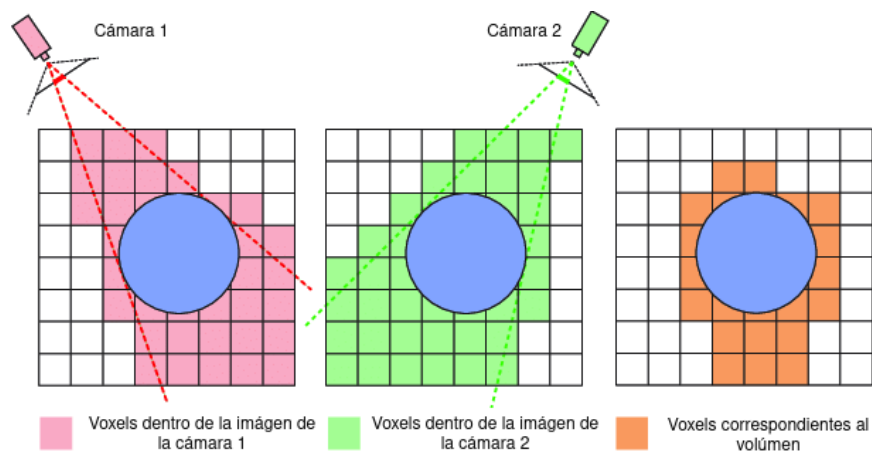


Figura 2.10: Diagrama simplificado del proceso de *voxel carving*. Fuente [9].

El resultado final es una representación volumétrica detallada del objeto original. Al elimi-

nar los voxels que no conforman el objeto, se logra una reconstrucción precisa y tridimensional, manteniendo la forma y el tamaño del modelo original. La resolución resultante depende del tamaño de los voxels y de la resolución de las imágenes de profundidad.

2.6.4. Nube de puntos a voxels

El proceso de conversión de una nube de puntos a grilla de voxels se realiza dentro de un espacio definido por límites mínimos y máximos. Esto es necesario ya que la grilla de voxels se inicializa con un tamaño determinado. La nube de puntos puede ser escalada para caber dentro de los límites definidos.

Durante la conversión, a cada punto de la nube se asigna a un voxel en la grilla, basándose en su ubicación espacial. Si la nube de puntos tiene información de color, esta se promedia dentro de cada voxel, permitiendo que la representación final no solo tenga forma, sino también color.

Este algoritmo puede generar grillas de voxels con voxels que no tienen ningún vecino directamente adyacente. Además, la densidad de los voxels depende de la densidad de la nube de puntos. En la Figura 2.11 se muestra el resultado de este algoritmo para nubes de puntos con diferentes cantidades de puntos. Lo que se aprecia en las imágenes corresponde a una superficie formada por voxels. A lo largo de las cuatro imágenes, la cantidad de puntos de las nubes de puntos originales va incrementando. Esto se traduce en que la cantidad de voxels obtenidos por el algoritmo de conversión también va incrementando.

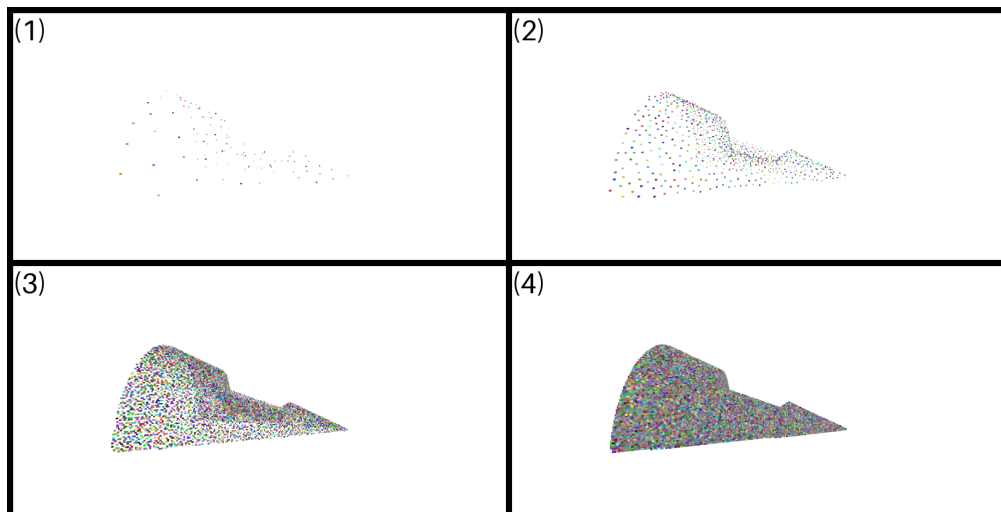


Figura 2.11: Resultado de la conversión de nube de puntos a grilla de voxels. En (1) la nube de puntos contenía 100 puntos. En (2) la nube de puntos contenía 1000 puntos. En (3) la nube de puntos contenía 10.000 puntos. En (4) la nube de puntos contenía 100.000 puntos.

2.7. Algoritmos de alineación de mallas geométricas

Los algoritmos de alineación de mallas geométricas (también llamados algoritmos de registro de mallas geométricas) son técnicas utilizadas para alinear y comparar diferentes mallas

geométricas. Estos algoritmos pueden ser utilizados para una variedad de aplicaciones, incluyendo la comparación de formas, la fusión de mallas, y la reparación de mallas.

2.7.1. Iterative Closest Point

El algoritmo Iterative Closest Point (ICP) se utiliza para alinear dos nubes de puntos (Figura 2.12). En cada iteración, el algoritmo selecciona un punto en un conjunto de datos y encuentra el punto más cercano en el otro conjunto [10]. Luego, aplica una transformación (una rotación y/o traslación) a uno de los conjuntos de datos para alinear mejor los dos conjuntos. Este proceso se repite hasta que la diferencia entre los conjuntos de datos alcanza un mínimo definido. Una de las principales limitaciones del algoritmo ICP es su tendencia a converger a soluciones subóptimas, conocidas como mínimos locales. La precisión de la alineación final pueden verse significativamente afectada si el punto de partida del algoritmo no está adecuadamente alineado o cercano a la posición objetivo.

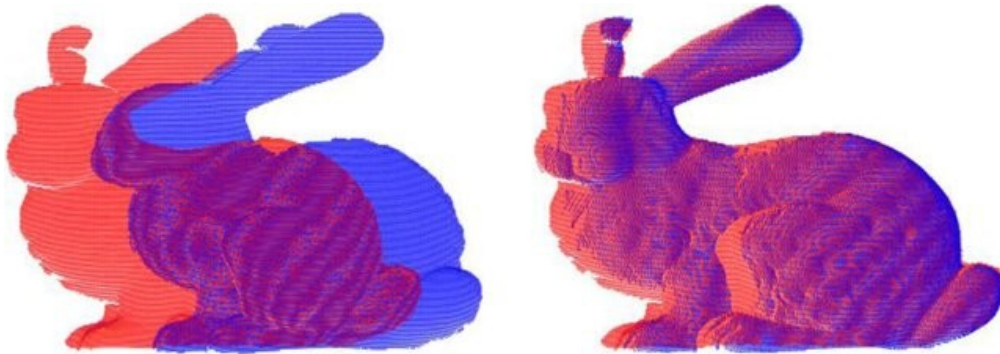


Figura 2.12: El algoritmo ICP busca alinear dos nubes de puntos. En la izquierda, se tienen dos nubes de puntos no alineadas de color rojo y azul. En la derecha, las dos nubes de puntos se encuentran alineadas. Fuente [11].

Matemáticamente, ICP trata de minimizar una función de error que está definida por la suma de las distancias cuadradas entre cada punto y su más cercano. Esto se puede expresar como:

$$E(R, t) = \sum_{i=1}^n \|p_i - (Rq_i + t)\|^2 \quad (2.12)$$

donde p_i y q_i son los puntos correspondientes en los dos conjuntos de datos, R es la matriz de rotación, y t es el vector de traslación. La actualización de R y t para minimizar $E(R, t)$ es un problema de mínimos cuadrados que puede ser resuelto mediante métodos estándar como DVS (Descomposición en Valores Singulares).

Las etapas y las ecuaciones relacionadas del algoritmo ICP se explican a continuación:

1. Búsqueda de Correspondencias: Este paso tiene como objetivo establecer correspondencias punto a punto entre los puntos de de la nube de puntos fuente (denotada como S) y la nube de puntos objetivo (denotada como B). Supongamos que S tiene N_s puntos y B tiene N_t puntos. Las correspondencias se pueden encontrar minimizando la distancia entre

los pares de puntos:

$$\arg \min_{\{i=1 \text{ a } N_s\}} \sum_{i=1}^{N_s} \sum_{j=1}^{N_t} \|S_i - B_j\|^2 \quad (2.13)$$

2. Estimación de la Transformación: La transformación T , que alinea la nube de puntos fuente con la nube de puntos objetivo, se puede representar como una matriz de transformación homogénea de 4×4 . Dependiendo de la técnica de registro específica utilizada, las ecuaciones de estimación de la transformación pueden variar. Por ejemplo, en el registro rígido, la matriz de transformación se puede representar como:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \quad (2.14)$$

donde R es una matriz de rotación de 3×3 y t es un vector de traslación de 3×1 . **3. Actualización de la Transformación:** La estimación actual de la transformación T se actualiza en función de la transformación estimada obtenida en el paso anterior. Esto se puede hacer mediante una operación de composición, como la multiplicación de matrices, para actualizar la transformación actual:

$$T_{\text{nueva}} = T_{\text{antigua}} \cdot T_{\text{estimada}} \quad (2.15)$$

4. Cálculo del Error: El cálculo del error cuantifica la diferencia entre la nube de puntos fuente transformada y la nube de puntos objetivo. Este error es una métrica importante para la evaluación del desempeño del algoritmo. Dependiendo de la aplicación específica y la métrica de error utilizada, las ecuaciones pueden variar. Las métricas de error comunes incluyen la distancia punto a punto o la distancia punto a plano. Por ejemplo, el cálculo del error de distancia punto a punto se puede realizar de la siguiente manera:

$$\text{error} = \sum_{i=1}^{N_s} \|B_i - T(R \cdot S_i + t)\|^2 \quad (2.16)$$

donde B_i representa el i -ésimo punto de la nube de puntos objetivo, R y t representan los componentes de rotación y traslación de la transformación, y S_i representa el i -ésimo punto de la nube de puntos fuente.

Código 2.1: Pseudocódigo del algoritmo ICP

```

1 Entrada: SetA y SetB (conjuntos de puntos)
2 Salida: R (matriz de rotación), t (vector de traslación)
3
4 Inicializar R, t
5 For i = 1 Until max_iterations Do
6     correspondencia = []
7     For each puntoA in SetA Do
8         puntoB = encontrarPuntoMasCercano(puntoA, SetB)
9         Add (puntoA, puntoB) To correspondencia
10    End For
11    (R, t) = MinimizarError(Correspondencia)
12    If Error < umbral Then
13        Stop

```



```
14 End If  
15 End For  
16 return (R, t)
```

2.7.2. Iterative Closest Point aplicado en mallas geométricas

El algoritmo Iterative Closest Point (ICP) puede ser utilizado para alinear dos mallas geométricas. Para ambas mallas se genera una nube de puntos utilizando el muestreo uniforme. Esta nube de puntos generada comparte el mismo sistema de referencia que la malla geométrica. Cualquier transformación rígida aplicada sobre la nube de puntos puede ser aplicada sobre la malla para mantener la posición relativa entre la nube de puntos y la malla geométrica. Esto significa que si se encuentra una transformación para alinear la nube de puntos generada desde la malla fuente con la nube de puntos generada desde la malla objetivo, la misma transformación puede ser usada para alinear la malla fuente con la malla objetivo.

2.7.3. Algoritmos de aprendizaje de máquinas

Los algoritmos de aprendizaje de máquinas son una clase de algoritmos que pueden aprender a realizar tareas a partir de datos. En el contexto de la alineación de mallas geométricas, los algoritmos de aprendizaje de máquinas pueden ser utilizados para aprender a alinear mallas de manera eficiente y precisa. Estos algoritmos pueden ser particularmente útiles para alinear mallas de geometrías complejas o de alta dimensión, donde los métodos tradicionales pueden ser ineficientes o inexactos.

2.7.3.1. Redes Neuronales

Las redes neuronales son modelos computacionales inspirados en el sistema nervioso de los seres vivos, especialmente el cerebro humano, diseñados para simular la manera en que los humanos aprenden. Estos modelos consisten en un conjunto de nodos, o "neuronas", conectados en diversas capas que transmiten señales de una capa a la siguiente.

Una red neuronal típica está compuesta por una capa de entrada, una o varias capas ocultas y una capa de salida. Cada neurona en una capa está conectada a todas las neuronas en la siguiente capa mediante pesos que se ajustan durante el proceso de aprendizaje. El funcionamiento de una red neuronal implica la alimentación de datos en la capa de entrada, que luego se procesan a través de las capas ocultas utilizando funciones de activación, resultando en una salida que proporciona la predicción o clasificación.

Las redes neuronales se utilizan en varias áreas de las ciencias y la ingeniería. Su capacidad para aprender a partir de grandes cantidades de datos y reconocer patrones complejos las hace particularmente útiles en campos donde se trabaja con datos de alta dimensionalidad como imágenes o nubes de puntos.

2.7.3.2. PointNetLK

PointNetLK [12] corresponde a una arquitectura de redes neuronales profundas diseñada para la alineación de nubes de puntos. Se basa en la arquitectura de PointNet [13], una red neuronal profunda diseñada para la segmentación y clasificación de nubes de puntos. La idea fundamental de PointNet es utilizar un perceptrón multicapa aplicado a cada punto de la

nube de puntos. Luego, utiliza una función simétrica (como la función *max pooling*) para asegurar la invarianza a la permutación de los puntos de la nube de puntos. Esto significa que el orden en el que se presentan los puntos de la nube de puntos no afectan el resultado final.

PointNetLK (donde LK viene de Lucas-Kanade) introduce un método de alineación iterativo inspirado por el algoritmo de Lucas-Kanade. El algoritmo de Lucas-Kanade se usa en visión computacional para alinear imágenes. En la arquitectura PointNetLK se busca alinear una nube de puntos a otra estimando iterativamente una transformada que reduce la distancia punto a punto entre ambas nubes. PointNetLK utiliza las nubes de puntos sin realizar ningún pre-procesamiento y aprovecha la arquitectura de PointNet para aprender características útiles para la alineación.

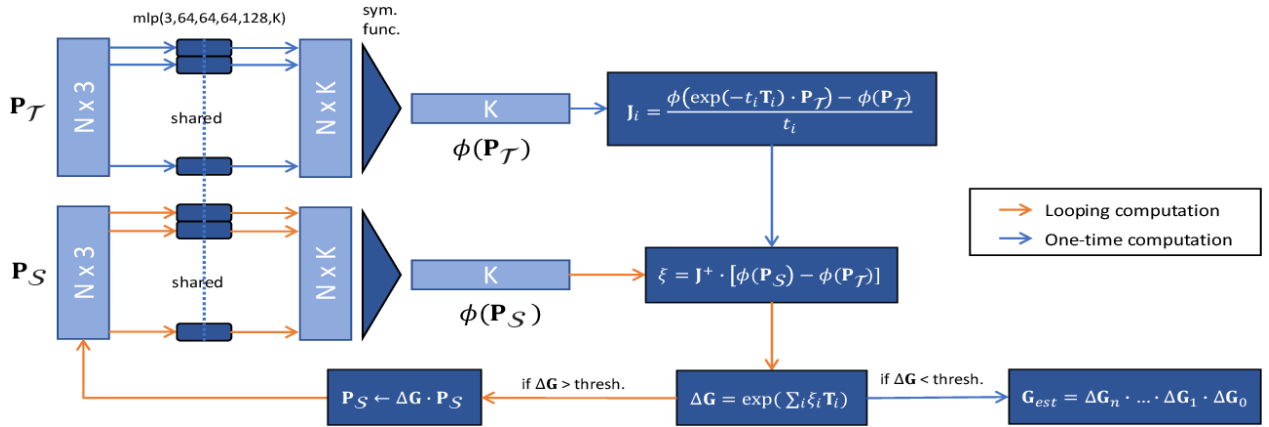


Figura 2.13: Arquitectura de la red PointNetLk. Fuente [12]

Los componentes y funcionalidades de la arquitectura PointNetLK (Figura 2.13) se explican a continuación:

1. **Nubes de puntos:** Las nubes de puntos origen P_S y objetivo P_T son las entradas del proceso. Estas son conjuntos de puntos en un sistema de coordenadas tridimensional que representan las superficies de objetos.
2. **MLP Compartido:** Ambas nubes de puntos se procesan a través de un Perceptrón Multicapa Compartido (MLP). El hecho de que sea compartido significa que se utilizan los mismos pesos para procesar ambas nubes de puntos.
3. **Función Simétrica:** Después del MLP, se aplica una función simétrica (como *max pooling*). Este paso es importante para garantizar que las características extraídas de las nubes de puntos sean invariantes a las permutaciones de los puntos en la nube.
4. **Vectores de Características Globales:** Las salidas de las funciones simétricas son los vectores de características globales $\phi(P_S)$ y $\phi(P_T)$. Estos vectores resumen toda la nube de puntos en una forma más compacta que aún captura la forma esencial y la estructura de los datos.
5. **Cálculo del Jacobiano J :** El Jacobiano J se calcula utilizando el vector de características de la nube de puntos origen $\phi(P_T)$.

6. **Transformación Óptima:** La transformación se encuentra a través de un proceso de optimización. Corresponde a la transformación para alinear la nube de puntos origen con la objetivo.
7. **Actualización Incremental de P_S :** Utilizando la transformación parcial, la pose de la nube de puntos fuente P_S se actualiza incrementalmente para alinearla con P_T .
8. **Cálculo de $\phi(P_S)$:** Después de actualizar la pose de P_S , su vector de características global se recalcula para reflejar la nueva pose.
9. **Función de Pérdida Durante el Entrenamiento:** En un escenario de entrenamiento, se utiliza una función de pérdida para medir la diferencia entre la transformación estimada y la óptima. La función de pérdida guía el entrenamiento del MLP para minimizar esta diferencia con el tiempo.

2.8. Operaciones booleanas

Las operaciones booleanas son una herramienta fundamental en la representación y manipulación de geometrías 3D. Estas operaciones, que incluyen la unión, la intersección y la diferencia (Figura 2.14), permiten la creación y modificación de formas complejas a partir de formas más simples. En el contexto de la reparación de componentes con manufactura aditiva, las operaciones booleanas pueden ser utilizadas para calcular la diferencia entre un componente dañado y su modelo original, proporcionando una representación precisa del daño.

Las primitivas geométricas son las formas más simples y fundamentales. Las operaciones booleanas sobre estas permiten crear formas más complejas a través de su combinación. Por ejemplo, al unir dos cilindros y un cubo se puede formar una forma más compleja.

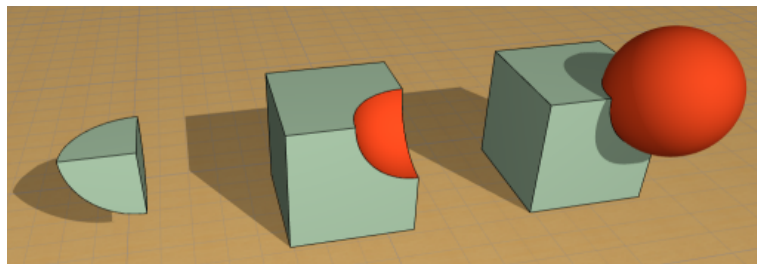


Figura 2.14: Ejemplo gráfico de operaciones booleanas entre un cubo y una esfera. De izquierda a derecha: intersección, diferencia y unión. Imagen de Graphisoft.

En el contexto de mallas (mesh), las operaciones booleanas se vuelven más complejas. La unión de dos mallas implica fusionar sus vértices y caras, mientras que la intersección o diferencia requiere calcular cómo se superponen y luego modificar la geometría de acuerdo. Esto a menudo requiere una reconstrucción de la topología de la malla, y puede ser computacionalmente intensivo. Adicionalmente, estas operaciones exigen ciertas características de las mallas de triángulos como que correspondan a un *manifold* y no tengan triángulos aislados.

Realizar operaciones booleanas sobre grillas de voxels es un proceso sencillo. Dada la estructura ordenada de las grillas de voxels, es posible comparar, una por una, las posiciones con existencia de voxels. Esto facilita la aplicación de reglas directas para operar intersecciones, uniones y diferencias (Figura 2.15).

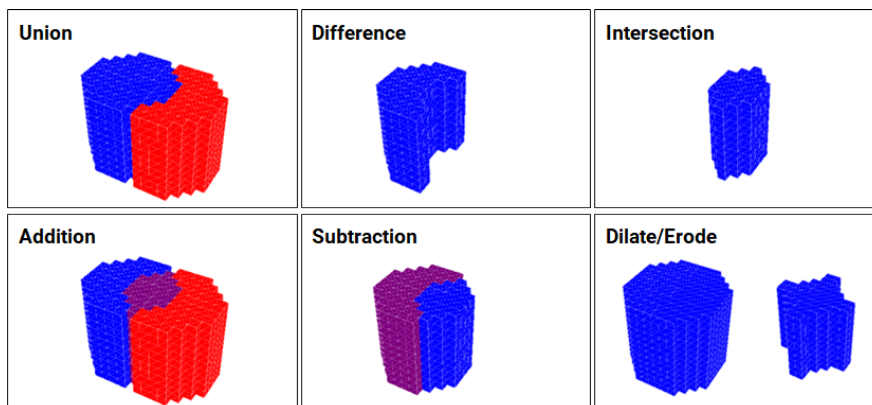


Figura 2.15: Ejemplo gráfico de operaciones booleanas entre dos grillas de voxels. Fuente [14].

2.9. Algoritmos de agrupamiento

Los algoritmos de agrupamiento (o algoritmos de *clustering*) son técnicas de aprendizaje automático que agrupan datos similares en conjuntos o clústeres. Estos algoritmos son particularmente útiles en la segmentación y clasificación en nubes de puntos y mallas. Algunos algoritmos de clusterización ampliamente utilizados corresponden a:

- K-Means [15]: Este algoritmo particiona las nubes de puntos o mallas en K clústeres, minimizando la varianza dentro de cada cluster. Aunque es simple y eficiente, requiere definir el número de clústeres de antemano y puede ser sensible a los valores iniciales.
- DBSCAN [16] (Density-Based Spatial Clustering of Applications with Noise): Este método agrupa puntos que están estrechamente juntos basándose en la densidad, siendo capaz de encontrar grupos de formas arbitrarias y manejar outliers. Es muy usado en datos espaciales como nubes de puntos. El algoritmo DBSCAN comienza inicializando un contador de clústeres a cero y procesa cada punto en la nube de puntos. Si un punto ya ha sido visitado, se salta; si no, se buscan sus vecinos dentro de un radio definido. Si un punto tiene menos vecinos que el mínimo requerido, se marca como ruido; de lo contrario, se asigna a un nuevo clúster o a uno existente. A continuación, se explora cada vecino del punto central, marcando los que estén etiquetados como ruido y asignándolos al clúster actual si no han sido visitados. Se agregan nuevos vecinos al conjunto de semillas si cumplen con el criterio de punto central. Este proceso se repite para todos los puntos en el conjunto de semillas y para cada nuevo punto no visitado en la base de datos hasta que todos los puntos han sido procesados. El algoritmo finaliza cuando cada punto ha sido etiquetado como parte de un clúster o como ruido, resultando en un conjunto de puntos agrupados según su proximidad y densidad.

2.10. Generación de datos sintéticos

La generación de datos sintéticos, conocida en inglés como *Data Augmentation* es un método eficaz para incrementar el volumen y la variabilidad de los datos disponibles para la evaluación de diferentes algoritmos. Este método puede ser utilizado con diversos tipos de datos, incluido modelos 3D [17]. Al crear nuevas versiones modificadas de los modelos 3D existentes, se genera un conjunto de datos más diverso que permite evaluar la eficacia y precisión de distintos algoritmos bajo diversas condiciones y contextos. Esto es especialmente útil para pruebas de robustez, rendimiento y precisión al manejar diferentes formas, tamaños y orientaciones de objetos. Algunas transformaciones que se pueden aplicar a los modelos 3D en formato *mesh* incluyen:

- **Rotación:** Girar el objeto alrededor de uno o más ejes.
- **Traslación:** Mover el objeto en el espacio 3D sin cambiar su orientación.
- **Escalado:** Aumentar o disminuir el tamaño del objeto manteniendo su forma.
- **Ruido Gaussiano:** Añadir una pequeña perturbación en las posiciones de los vértices del objeto para simular ruido o imprecisión.
- **Deformación Elástica:** Distorsionar el objeto de manera no uniforme para simular variaciones naturales o daños.

Estas transformaciones permiten generar versiones alteradas de los modelos originales, aumentando así la capacidad de los algoritmos para manejar y procesar datos 3D en diversas situaciones y contextos.

Capítulo 3

Metodología

En este capítulo se detallan las actividades realizadas para el desarrollo e implementación del sistema de identificación de daño. A grandes rasgos, las actividades desarrolladas se engloban de la siguiente manera:

1. **Generación de la base de datos de evaluación:** Esta etapa involucra la recolección de escaneos 3D existentes y la generación de modelos sintéticos. Estos datos son utilizados para probar y ajustar el software durante su desarrollo.
2. **Implementación de algoritmos:** En esta etapa se implementarán algoritmos basados en ICP (Iterative Closest Point) y algoritmos de aprendizaje de máquinas para la alineación de las mallas geométricas. Estos algoritmos permitirán comparar el modelo 3D de la pieza sin daño con el escaneo 3D de la pieza dañada.
3. **Configuración de algoritmos:** Los algoritmos implementados serán configurados y ajustados para trabajar específicamente con modelos de piezas metálicas. Esta etapa puede requerir pruebas repetitivas y ajustes iterativos para optimizar la precisión y eficacia de los algoritmos.
4. **Extracción de la representación 3D del daño:** Con los modelos alineados y comparados, el sistema procederá a extraer una representación 3D del daño. Este modelo 3D será generado a través de una diferencia booleana entre el modelo sin daño y el escaneo 3D.
5. **Evaluación del sistema:** Finalmente, se evaluará el desempeño del sistema utilizando la base de datos creada en la primera etapa. Esta evaluación permitirá determinar la precisión y eficiencia del sistema y hacer ajustes y mejoras si es necesario. Como principal métrica de desempeño cuantitativo, se utilizará el error de alineación entre las mallas geométricas.

3.1. Arquitectura del sistema propuesto

Para desarrollar un sistema robusto y sostenible, se implementa una arquitectura modular para el procesamiento de mallas de triángulos (Figura 3.1). Dividiendo el sistema en varios módulos independientes, se facilita la mejora incremental de todo el sistema al optimizar cada módulo individualmente.

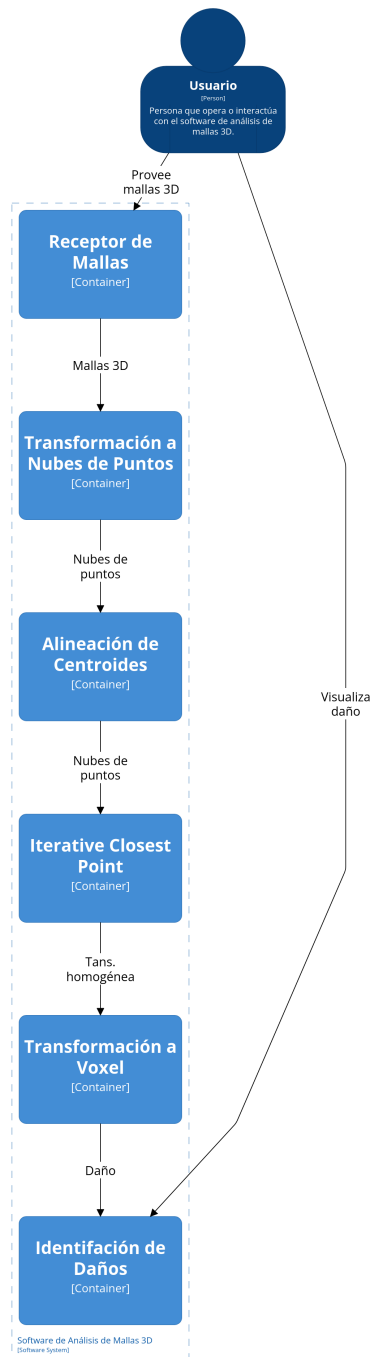


Figura 3.1: Arquitectura del sistema de identificación de daños. Fuente: Elaboración propia

Los componentes del sistema son los siguientes:

- **Usuario:** La persona que proporciona las mallas 3D de la pieza original y la pieza dañada.
- **Receptor de Mallas:** Módulo que recibe las mallas 3D.

- **Transformación a Nubes de Puntos:** Convierte mallas 3D en nubes de puntos.
- **Alineación de Centroides:** Alinea los centroides de las nubes de puntos.
- **Iterative Closest Point:** Algoritmo para alinear ambas nubes de puntos y por consiguiente las mallas de triángulos.
- **PointNetLK:** Red neuronal para procesar y alinear las nubes de puntos. Se propone como alternativa a ICP.
- **Transformación Homogénea:** La transformación resultante del algoritmo ICP o de la red neuronal PointNetLK.
- **Transformación a Voxel:** Convierte las mallas triangulares a una representación de voxels para realizar operaciones booleanas.
- **Identificación de Daños:** Realiza una operación booleana de diferencia sobre las representaciones de voxels de la pieza dañada y la pieza original. Luego aplica el algoritmo DBSCAN para etiquetar los voxels correspondientes a la identificación del daño.

3.2. Preparación de la base de datos de evaluación

El sistema de identificación de daños se diseña para identificar el daño en piezas mecánicas utilizadas en las industrias de generación de energía, extracción de recursos naturales y productivas. Es fundamental al momento de diseñar un sistema de software entender las características de los datos a procesar. En el caso de este trabajo de título, los datos a procesar corresponden a modelos tridimensionales de piezas metálicas obtenidos por escaneo 3D y por modelado por computadora (CAD).

La base de datos utilizada fue confeccionada por otros memoristas que trabajaron en el mismo proyecto. Las características de esta base de datos se muestran en las Tablas 3.1, 3.2, 3.3 y 3.4. El identificador usado sigue la siguiente estructura:

- Las primeras letras indican la pieza. **A** corresponde a aguja Pelton, **C** corresponde a cilindro, **EIP** corresponde a espejo inyector Pelton y **PL** corresponde a placa.
- La segunda letra corresponde a el tipo de daño. Dependiendo de la ubicación del daño se puede agregar el adjetivo “radial”. Este puede ser uno de los siguientes tipos:
 - Esferas.
 - Pitting.
 - Bolsillo
 - Circular.
 - Abrasión.
 - Abrasión punto silla.

Adicionalmente, pueden existir modelos con estos tipos de daños combinados.

- La última letra corresponde a si es un modelo simulado (en ese caso se usa **S**) o un modelo real (**R**). Una letra adicional significa si la malla de triángulos es abierta (**RA**) o cerrada (**RC**).

Tabla 3.1: Parte de la base de datos de evaluación. Modelos correspondientes a agujas Pelton.

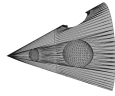
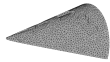
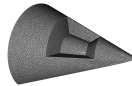
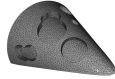
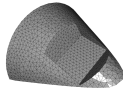
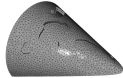


Identificador	Descripción	Imagen
A-E-S	Daño tipo esferas	
A-PR-S	Daño tipo pitting radial	
A-BR-S	Daño tipo bolsillo radial	
A-DCR-S	Daños tipo circulares radiales	
A-B-S	Daño tipo bolsillo	
A-P-S	Daño tipo pitting	
A-A-S	Daño tipo abrasión	
A-Aps-S	Daño tipo abrasión punto silla	

Tabla 3.2: Parte de la base de datos de evaluación. Modelos correspondientes a cilindros.






Identificador	Descripción	Imagen
C-AP-RC	Daño tipo abrasión y pitting	
C-Aps-R	Daño tipo abrasión punto silla	
C-P-R	Daño tipo pitting	
C-AP-RA	Daño tipo abrasión y pitting	
C-B-R	Daño tipo bolsillo	

Tabla 3.3: Parte de la base de datos de evaluación. Modelos correspondientes a espejos Pelton.

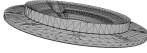
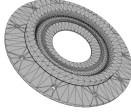
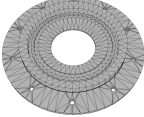
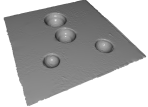
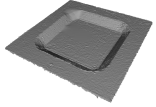
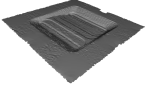
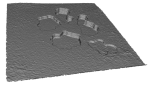
Identificador	Descripción	Imagen
EIP-B-S	Daño tipo bolsillo	
EIP-R1-S	Daño tipo radial	
EIP-R2-S	Daño tipo radial	

Tabla 3.4: Parte de la base de datos de evaluación. Modelos correspondientes a placas.

Identificador	Descripción	Imagen
PL-E-R	Daño tipo esferas	
PL-B-R	Daño tipo bolsillo	
PL-Aps-R	Daño tipo abrasión punto silla	
PL-P-R	Daño tipo pitting	



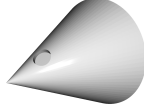
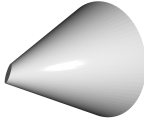


Es importante señalar que la base de datos incluye modelos con una variedad de características que influyen en el diseño del sistema de identificación de daños. Estos modelos son útiles debido a su relevancia y representatividad de las piezas mecánicas comúnmente utilizadas en el sector industrial.

Adicionalmente a la base de datos de modelos disponibles, se generan sintéticamente seis nuevos modelos para evaluar rasgos específicos del sistema de identificación de daño. Estos rasgos corresponden a:

- Identificación de daño en piezas no convexas.
- Identificación de daño en piezas que no siguen una geometría primitiva.
- Efecto de una vista completa de la pieza dañada.

Las características de la base de datos se presentan en la Tabla 3.5.

Tabla 3.5: Parte de la base de datos de evaluación. Modelos generados sintéticamente.

Identificador	Descripción	Imagen
AXG-B	Daño tipo bolsillo	
AXG-BT	Daño tipo bolsillo y diente	
CO-CP	Daño tipo circular	
CO-PU	Daño tipo punta quebrada	
CO-RAN	Daño tipo ranura	
BG-T	Daño tipo diente quebrado	

3.3. Elección del lenguaje de programación

Para la implementación del sistema de identificación de daños se utiliza el lenguaje de programación Python con la librería open3d especializada en manejo de datos tridimensionales. La elección del lenguaje de programación Python se realiza por los siguientes argumentos:

- Python es un lenguaje ampliamente utilizado en las ciencias y en la ingeniería, tanto en la academia como en la industria.
- Python es un lenguaje versátil con una curva de aprendizaje baja. Esto permite asegurar que el desarrollo se pueda mantener en el futuro por otras personas.

Sin embargo, existen contra-argumentos comunes en contra de utilizar Python para procesamiento de datos tridimensionales:

- Python es un lenguaje interpretado con menor rendimiento en comparación con lenguajes compilados como C++ o Fortran, especialmente en operaciones que requieren un alto grado de cálculo numérico y manejo intensivo de datos.

- Aunque las librerías como NumPy y SciPy mejoran significativamente el rendimiento para operaciones numéricas, pueden no ser tan eficientes como las soluciones específicamente optimizadas en lenguajes de bajo nivel.

Estos contra-argumentos se abordan con la utilización de la librería open3D. La librería open3D es verdaderamente una librería de C++ que provee *Python bindings*. Esto significa que la ejecución de las funciones realmente ocurre en código C++ compilado, aprovechando la eficiencia y desempeño de este lenguaje. Adicionalmente, la utilización de la librería open3D permite portar en un futuro el código desde Python a C++ si fuese necesario.

3.4. Implementación de Iterative Closest Point

Como parte del desarrollo del sistema de identificación de daño, se busca la alineación de dos mallas geométricas. Una proveniente de un escaneo 3D realizado de una pieza mecánica y la otra malla geométrica proveniente del modelo realizado utilizando herramientas CAD. Se supone que ambos modelos representan el mismo objeto a una alta resolución métrica. El algoritmo Iterative Closest Point (ICP) permite alinear dos nubes de puntos que correspondan a la representación del mismo objeto o escenario. Es posible utilizar este algoritmo para la alineación de dos mallas geométricas agregando un módulo de preprocesamiento de las mallas geométricas. Este módulo de preprocesamiento genera una nube de puntos a partir de una malla geométrica. Si se generan dos nubes de puntos correspondientes a las dos mallas geométricas que se desean alinear, se puede aplicar el algoritmo ICP sobre estas nubes de puntos. El resultado del algoritmo ICP corresponde a una transformación homogénea que al aplicarse a la nube de puntos origen, la traslada a la nube de puntos objetivo. Si se aplica esta transformación sobre la malla origen esta se traslada hacia la malla objetivo, obteniendo la alineación de ambas mallas.

Código 3.1: Pseudocódigo del pre-procesamiento de las mallas de triángulos.

```

1 // Paso 1: Normalizar ambas mallas
2 mallaOrigen = normalizar(mallaOrigen)
3 mallaObjetivo = normalizar(mallaObjetivo)
4
5 // Paso 2: Trasladar ambas mallas al origen según centroide
6 centrarMalla(mallaOrigen)
7 centrarMalla(mallaObjetivo)
8
9 // Paso 3: Generar nube de puntos
10 nubeOrigen = muestreoUniforme(mallaOrigen)
11 nubeObjetivo = muestreoUniforme(mallaObjetivo)
12
13 // Funciones auxiliares (pseudocódigo)
14 function centrarMalla(malla):
15     centroide = calcularCentroide(malla)
16     aplicarTraslacion(malla, -centroide)

```

Los resultados del algoritmo ICP son sensibles al estado inicial de las nubes de puntos y la estimación inicial de la transformación. Dependiendo de la configuración inicial, el resultado del algoritmo ICP puede corresponder a una alineación errónea, proveniente de un mínimo local de la función objetivo. Para abordar este fenómeno, en primer lugar ambas mallas y sus respectivas nubes de puntos generadas son trasladadas al origen de un sistema de coordenadas global fijo. Eso se realiza aplicando una traslación igual al inverso aditivo del vector posición del centroide de la malla. En segundo lugar, ambas mallas son normalizadas. Esto permite comparar los resultados entre distintos modelos con distintos tamaños. En tercer lugar, para cada par de mallas origen y objetivo, se realizan varias ejecuciones del algoritmo ICP. El número de ejecuciones es configurable. Antes de cada ejecución, se aplica una rotación aleatoria a la nube de puntos origen. Se guarda la transformación resultante junto con la distancia media cuadrática mínima. Al finalizar todas las ejecuciones, se almacena la transformación resultante asociada a la mínima distancia media cuadrática mínima. Esta heurística permite mejorar los resultados de alineación en comparación a la realización de una única ejecución del algoritmo ICP.

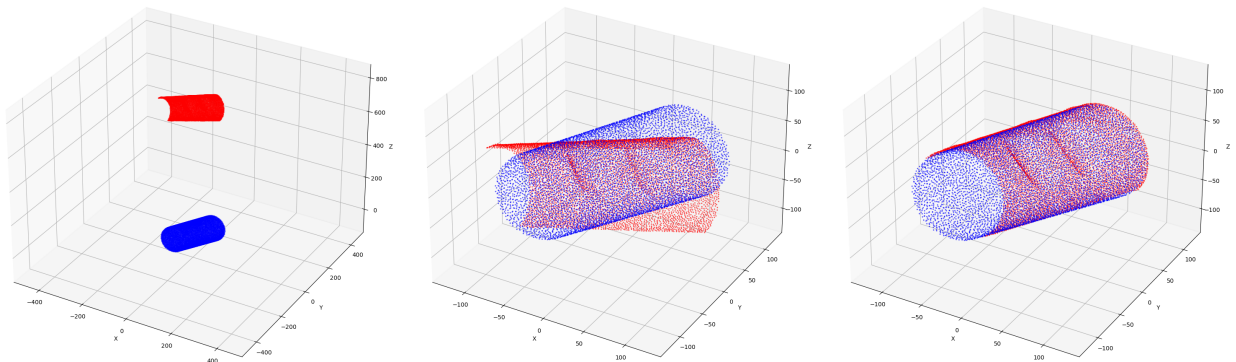


Figura 3.2: Proceso de alineación de nubes de puntos utilizando el algoritmo ICP. De izquierda a derecha: inicialmente, ambas nubes de puntos se sitúan en una posición arbitraria, determinada por la ubicación de la malla original. Seguidamente, se trasladan ambas nubes al origen del sistema de referencia. Finalmente, tras completar todas las iteraciones del algoritmo ICP, las nubes de puntos quedan alineadas.

Código 3.2: Pseudocódigo de la ejecución de múltiples ICP.

```

1 // Definir la función para mejorar la alineación con ICP
2 function alineacionDeNubes(nubeOrigen, nubeObjetivo, numeroDeEjecuciones):
3     // Inicializar la mejor transformación y la distancia mínima
4     mejorTransformacion = null
5     min_rmse = INFINITO
6
7     // Ejecutar varias veces el algoritmo ICP
8     For each i in 1..numeroDeEjecuciones Do
9         // Aplicar rotación aleatoria a la malla origen
10        rotacionInicial = generarRotacionAleatoria()
11
12        // Ejecutar ICP y obtener la transformación y la distancia
13        (transformacion, rmse) = ejecutarICP(nubeOrigen, nubeObjetivo, rotacionInicial)
14

```

```

15 // Comparar y actualizar la mejor transformación
16 If rmse < min_rmse Then
17     min_rmse = rmse
18     mejorTransformacion = transformacion
19 End If
20 End For
21 // Retornar la mejor transformación encontrada
22 return mejorTransformacion
23
24 function ejecutarICP(nubeOrigen, nubeObjetivo, transformacionInicial):
25     // Implementación específica del algoritmo ICP
26     // Retorna la transformación y la distancia media cuadrática mínima
27     // ...
28
29 // Llamada a la función principal
30 transformacionOptima = mejorarAlineacionICP(nubeOrigen, nubeObjetivo,
    ↪ numeroDeEjecuciones)

```

3.5. Implementación de PointNetLK

Para obtener resultados satisfactorios de alineación de mallas utilizando el algoritmo ICP, es necesario configurar una serie de parámetros. Estos parámetros pueden estar supeditados a las características de las mallas a alinear. Por ejemplo, para mallas con simetrías geométricas es necesario establecer un número alto de ejecuciones del algoritmo ICP con el fin de evitar un mínimo local en el resultado final. Adicionalmente, es complejo definir si los parámetros seleccionados son óptimos para la alineación de las mallas seleccionadas. Desde esta problemática aparece la necesidad de probar métodos alternativos para la alineación de mallas geométricas.

Considerando los excelentes resultados que las técnicas de aprendizaje de máquinas obtienen en diversas áreas de la ciencia y la ingeniería, se realiza una implementación de un modelo de aprendizaje de máquinas basado en redes neuronales llamado PointNetLK. Utilizando una implementación de código abierto con pesos pre-entrenados, se implementa la red neuronal y se evalúa su desempeño con la base de datos de evaluación. La evaluación resulta desfavorable para PointNetLK que obtiene un peor desempeño que ICP. Por esta razón, se utiliza ICP para realizar la alineación de los modelos.

3.6. Implementación de operaciones booleanas

Una vez que se han alineado satisfactoriamente las dos mallas, se busca identificar el daño a través del uso de operaciones booleanas sobre estas. Una alternativa es realizar una operación de diferencia entre la malla del modelo sin daño y la malla del escaneo del daño. Sin embargo, las operaciones booleanas sobre mallas exigen que ambas mallas sean *manifold*. Esta condición no se cumple para la mayoría de las mallas producidas por escaneo 3D. Es posible realizar un preprocesamiento a las mallas para modificarlas y obtener una malla *manifold*. Sin embargo, este proceso agrega una capa extra de complejidad al sistema por lo que se decide un método alternativo.

El método alternativo para realizar las operaciones booleanas y obtener una representación tridimensional del daño corresponde a utilizar voxels. En la siguiente sub-sección se presenta el proceso para transformar las mallas a voxels.

3.6.1. Transformación a representación de voxels

Con las dos mallas geométricas alineadas, se procede a transformar a una representación de voxel para cada malla. El objetivo del sistema de identificación de daño es detectar y entregar una representación tridimensional del daño. En el mundo físico, el daño de una pieza mecánica corresponde a la pérdida de un volumen de material. La representación de piezas tridimensionales en formato voxel permite representar de mejor manera el concepto de volumen.

Es posible obtener una representación tridimensional del daño sin trabajar con voxels. Sin embargo, las técnicas existentes exigen ciertas características de las mallas tridimensionales, rompiendo los requerimientos de un sistema de identificación de daño que funcione con una gran variedad de piezas mecánicas.

La representación de voxels ofrece simplicidad y versatilidad a cambio de la exigencia de recursos computacionales altos. Como se mencionó en el capítulo de antecedentes, la memoria necesaria para almacenar una grilla de voxels aumenta al cubo con respecto a la resolución de la grilla. Para el desarrollo de este trabajo se utiliza un tamaño de voxel igual a 0.0078125. Esto permite tener 256 voxels a lo largo de un segmento de largo 2. Esta cantidad de voxels permite resoluciones del rango de 1[mm] a 2[mm] en piezas de dimensiones en torno a los 200[mm]. Adicionalmente, mantiene los requerimientos computacionales a un nivel medio con el tiempo de procesamiento de cada pieza en el rango de 1 a 10 minutos.

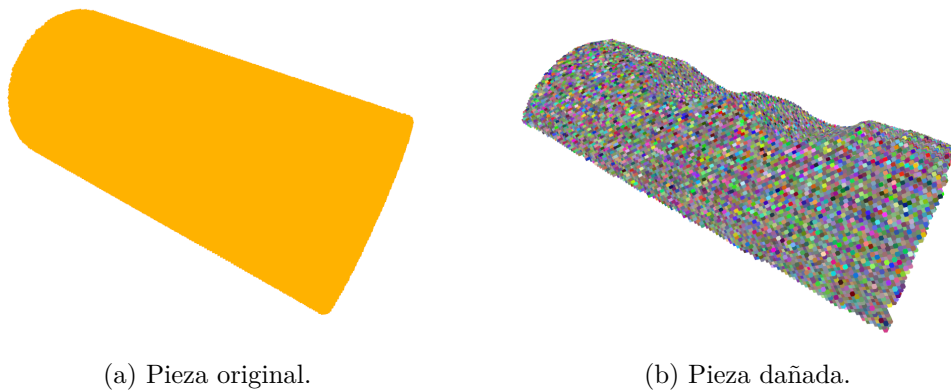


Figura 3.3: Resultados de la conversión a voxels de la pieza original y la pieza dañada. En (a) se visualiza la representación de un volumen denso de voxels correspondiente a la pieza original. En (b) se visualiza la superficie de voxels correspondientes al escaneo de la pieza dañada.

El algoritmo utilizado para transformar mallas tradicionales a voxels es distinto para la malla proveniente de el proceso de escaneo 3D y la malla proveniente del software CAD. La malla generada con un escáner 3D provee información sobre el daño en la pieza. Esta información existe en la superficie de la malla. Por esta razón, es suficiente con una transformación directa de la superficie de la malla a una superficie compuesta por voxels (Figura

3.3). En el caso de la malla proveniente del software CAD, es importante la generación de una representación volumétrica de la pieza. El daño identificado corresponde a un subconjunto del volumen de la pieza original. Para esto se utiliza *voxel carving* para la transformación desde malla a grilla de voxels (Figura 3.3).

3.6.2. Operación booleana de diferencia entre dos conjuntos de voxels

Tras alinear y convertir las mallas de la pieza dañada y la pieza original en representaciones de voxels (Figura 3.4), se inicia una operación booleana de diferencia. Este procedimiento comienza iterando sobre cada voxel de la pieza dañada. Para cada voxel, se examinan sus puntos de borde. En cada uno de estos puntos, se realiza una consulta en el conjunto de voxels de la pieza original. Dicha consulta implica identificar y obtener el voxel correspondiente que encierra al punto en cuestión. Una vez identificado, el voxel se elimina del conjunto. Este método se aplica sistemáticamente a todos los voxels de la pieza dañada.

Código 3.3: Pseudocódigo de la operación booleana de diferencia.

```
1 For each voxel in danada_voxels.obtenerVoxels() Do
2   punto_centro = voxel.punto_centro
3   puntos_borde = voxel.puntos_borde
4
5   voxel_a_remove = original_voxel.obtenerVoxel(punto_centro)
6   original_voxels.removeVoxel(voxel_a_remove)
7
8   For each punto in puntos_borde Do
9     voxel_a_remove = original_voxels.obtenerVoxel(punto)
10    original_voxels.removeVoxel(voxel_a_remove)
11  End For
12
13 End For
```

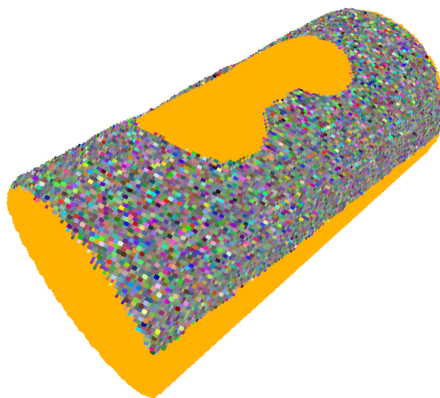


Figura 3.4: Conjunto de voxels correspondientes a la pieza dañada y la pieza original. Las dos representaciones se encuentran alineadas. Los voxeles de color amarillo corresponden a la pieza original. Los voxeles de la pieza dañada han sido pintados con colores aleatorios para una mejor visualización.

3.7. Identificación del daño utilizando algoritmos de clusterización

El resultado de realizar una operación booleana entre los conjuntos de voxels correspondientes a la pieza dañada y a la pieza original, es un conjunto de voxels en el que se distingue la representación del daño separada del volumen de la pieza original. Sin embargo, aunque una persona puede visualmente identificar la separación entre los subconjuntos de voxels, computacionalmente no se ha identificado el subconjunto correspondiente a la caracterización del daño.



Figura 3.5: Corte longitudinal sobre el conjunto de voxels obtenidos desde la operación booleana de diferencia.

Para lograr la identificación de los voxels correspondientes al daño en la pieza, se aplica un algoritmo de clusterización. El algoritmo de clusterización elegido es DBSCAN. DBSCAN solo necesita la especificación de dos parámetros para funcionar, lo que simplifica su implementación y configuración. El algoritmo DBSCAN realiza el agrupamiento considerando el punto central de cada voxel. El resultado corresponde a una etiqueta para cada voxel dependiendo del clúster en donde se encuentra. Para su visualización cada voxel es pintado de un determinado color dependiendo de su etiqueta, en la Figura 3.6 se muestra el cuerpo principal del objeto y el daño identificado con dos colores distintos. En color negro se encuentran voxels outliers que quedaron aislados durante la operación booleana de diferencia.



Figura 3.6: Voxels coloreados según la etiqueta obtenida con el algoritmo DBSCAN.

Una vez que se han etiquetados los voxels correspondientes al daño, estos pueden ser aislados del volumen principal de la pieza original. Como todos se han etiquetados con respecto al clúster al que pertenecen (o si corresponden a ruidos) es posible filtrar y obtener solos los voxels de determinado clúster. El clúster con mayor cantidad de voxels corresponde al volumen principal de la pieza original. Esto debido a que el daño es de mucho menor tamaño que la pieza en si misma. En la Figura 3.7 se presenta la identificación del daño junto a la malla de triángulos de la pieza dañada.

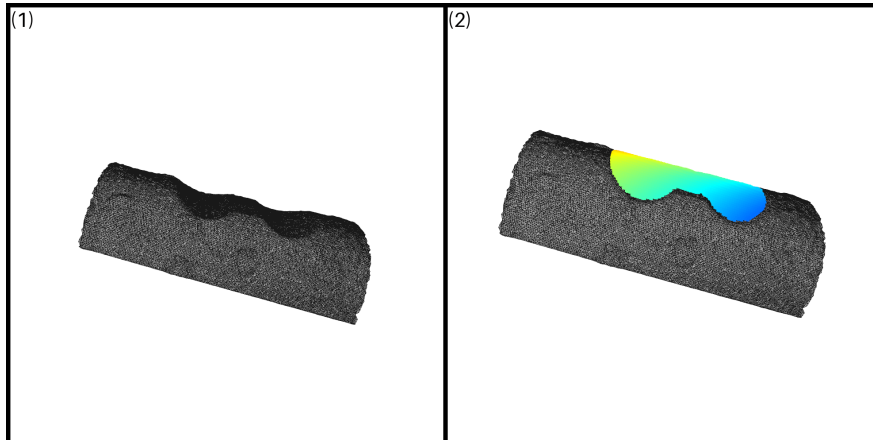


Figura 3.7: Voxels correspondiente a la identificación del daño junto con el escaneo 3D de la pieza dañada. En (1) se muestra el escaneo original en formato malla. En (2) se muestra el escaneo original en formato malla y la identificación del daño en formato voxel. El coloreado de gradiente se usa solo por motivos de visualización.

Capítulo 4

Resultados

En este capítulo se muestran y discuten los principales resultados del sistema de identificación de daño. En primer lugar se muestran y analizan los resultados del proceso de alineación de mallas triangulares. Estos resultados se presentan de manera cuantitativa y cualitativa. En segundo lugar se muestran y analizan los resultados del proceso de identificación del daño. Estos resultados se presentan de manera cualitativa.

4.1. Resultados del proceso de alineación

La métrica principal para evaluar el resultado del proceso de alineación corresponde a la raíz del error cuadrático medio. Esta corresponde a una medida de distancia entre las dos nubes de puntos alineadas.

En la Figura 4.1, se exhiben los valores correspondientes a la raíz del error cuadrático medio (Root Mean Square Error - RMSE) para diversas categorías de daños. Mediante una comparación entre los datos numéricos y la información cualitativa detallada en secciones subsiguientes, se establece que una alineación adecuada se caracteriza por un valor de RMSE igual o inferior a 0.02. Se observa que los modelos cilíndricos y los espejos Pelton logran una alineación satisfactoria, mientras que los modelos de aguja Pelton y las placas muestran alineaciones subóptimas. Las secciones siguientes profundizan en los resultados específicos del algoritmo de alineación para cada tipo de modelo.

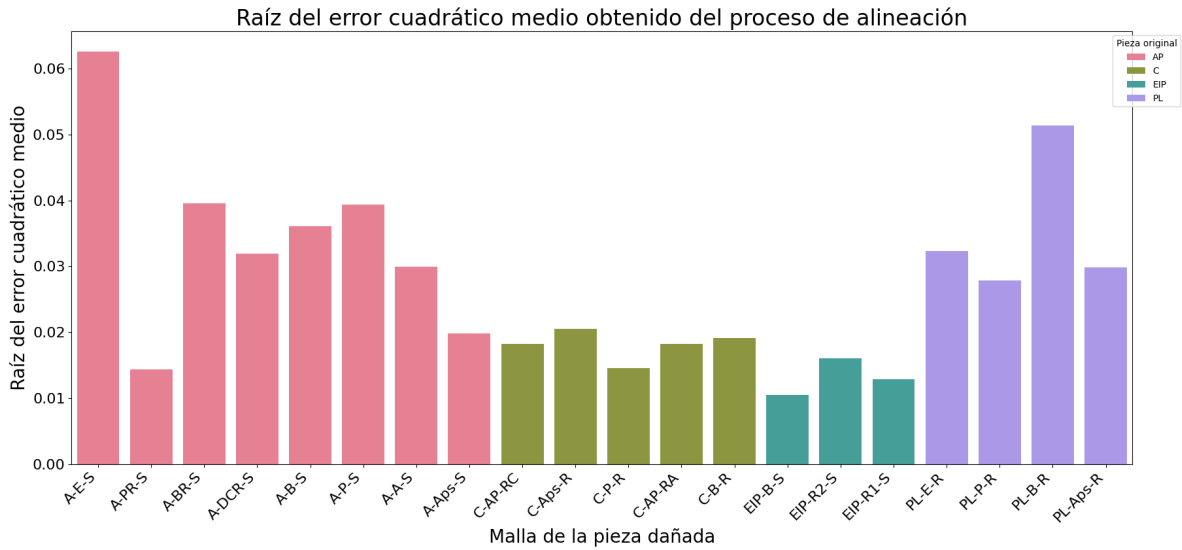


Figura 4.1: Raíz del error cuadrático medio obtenida utilizando el algoritmo ICP en la alineación de las mallas de la base de datos de evaluación.

En la Figura 4.2 se muestra la raíz del error cuadrático medio para los modelos de la base de datos sintética. Debido a que todas las mallas correspondientes a la pieza dañada corresponden a una vista total de la pieza, el algoritmo ICP logra resultados satisfactorios con un RMSE menor a 0.02.

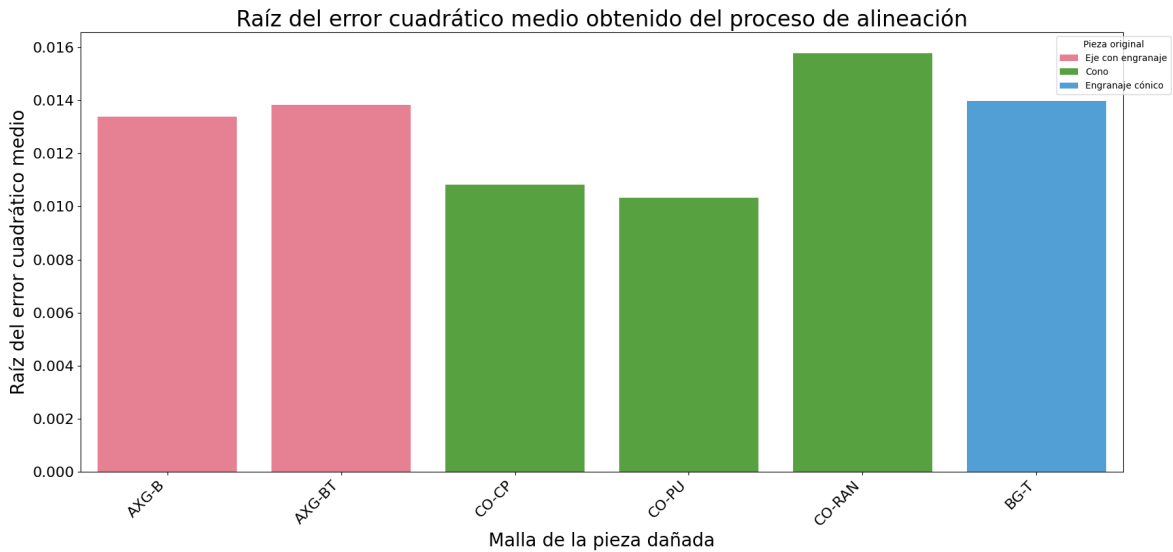


Figura 4.2: Raíz del error cuadrático medio obtenida utilizando el algoritmo ICP en la alineación de las mallas de la base de datos sintética.

En la Figura 4.3, se presenta una comparativa entre los valores de RMSE obtenidos mediante el uso del método de Iterative Closest Point (ICP) y los logrados con PointNetLK. Los

datos demuestran una clara superioridad de ICP frente a PointNetLK en todas las variantes de piezas y daños evaluados. Una explicación a este fenómeno radica en que PointNetLK fue diseñada para alinear pares de nubes de puntos correspondientes a vistas completas del modelo, mientras que la mayoría de los modelos en la base de datos de evaluación representan vistas parciales del objeto real. Los modelos que efectivamente presentan una vista completa logran resultados de alineación satisfactorios con PointNetLK.

Considerando las ventajas de ICP, se decide utilizar únicamente ICP para realizar la alineación de las mallas en el sistema de identificación de daño. En las próximas secciones se analiza cualitativamente los resultados de alineación del algoritmo ICP sobre los modelos de la base de datos.

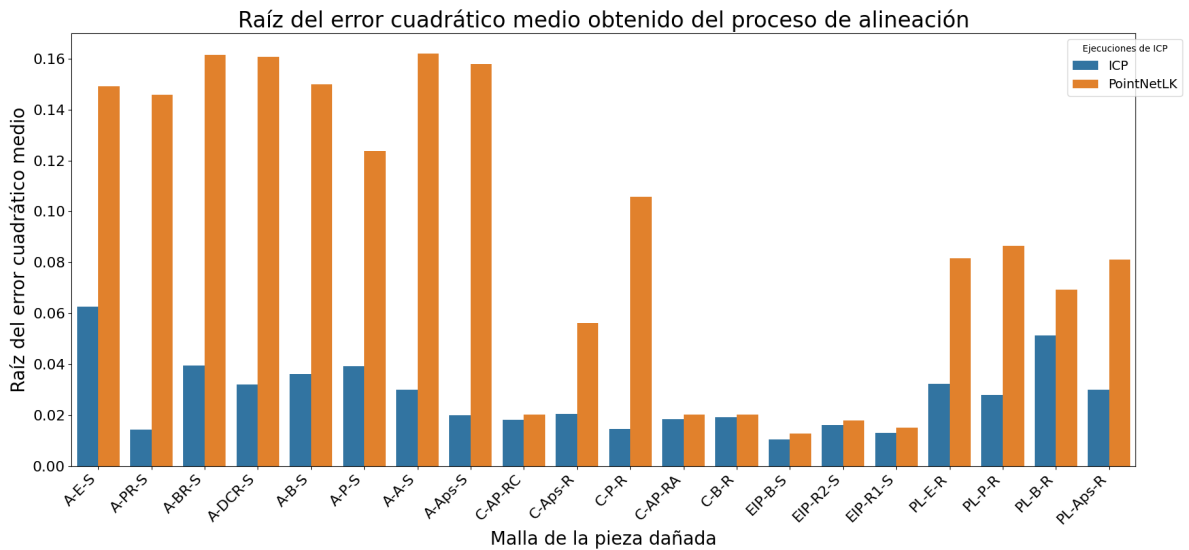


Figura 4.3: Raíz del error cuadrático medio obtenida utilizando ICP vs obtenida utilizando PointNetLk

4.1.1. Aguja de turbina Pelton

Los resultados de alineación del modelo A-E-S se muestran en la Figura 4.4. A simple vista se ve una buena alineación entre la nube de puntos correspondiente al daño y la nube de puntos de la pieza original. Sin embargo, existe una pequeña distancia entre ambas que afecta luego la identificación del daño. Este error de alineación puede ser detectado basándose en el gráfico de la Figura 4.1. El valor del error para A-E-S es considerablemente más alto que para los otros modelos.

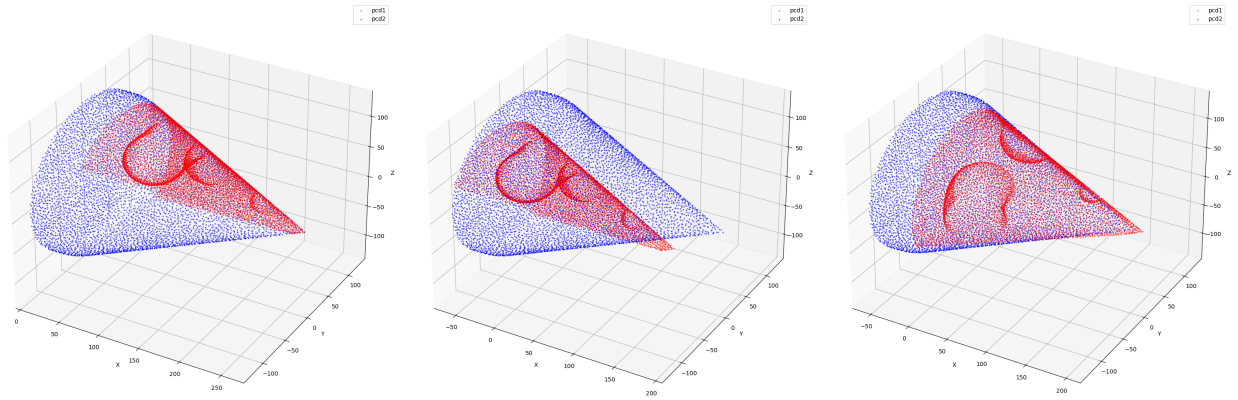


Figura 4.4: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-E-S

En la Figura 4.5 se visualiza el resultado del proceso de alineación sobre las mallas A-BR-S. Existe una pequeña distancia entre ambas nubes de puntos. Esto es congruente con los resultados del gráfico 4.1.

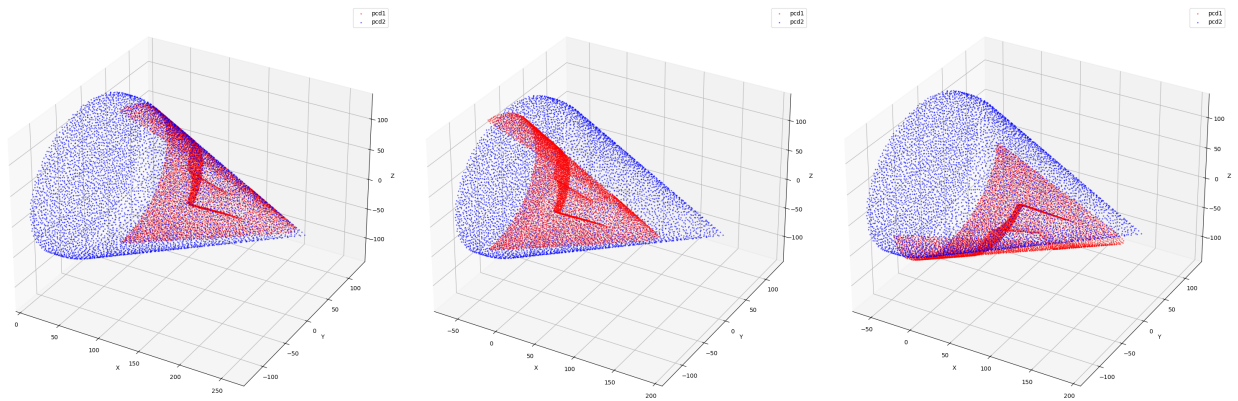


Figura 4.5: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-BR-S

La alineación de los modelos A-DCR-S presenta un pequeño desplazamiento con respecto a una alineación óptima (Figura 4.6). Esta tendencia se repite en los demás modelos de daño en agujas Pelton.

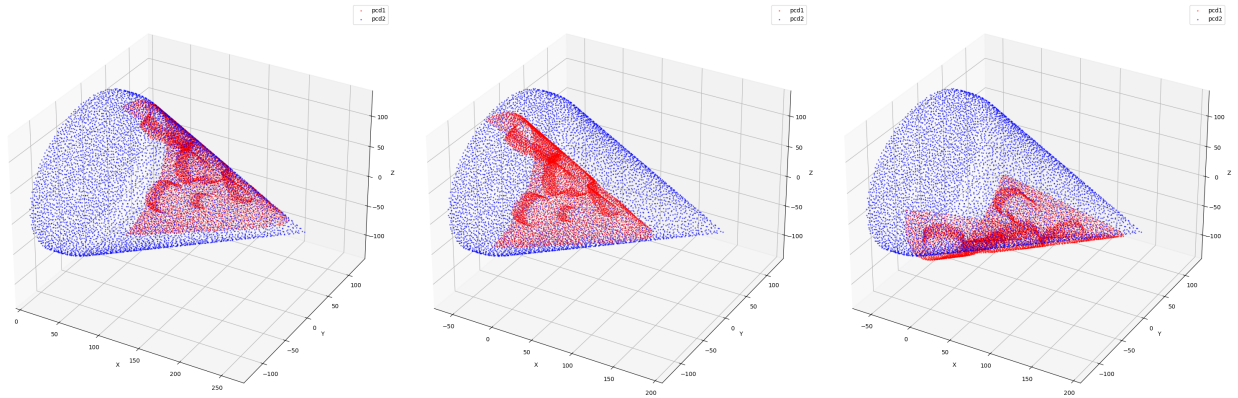


Figura 4.6: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-DCR-S

En los modelos A-A-S (Figura 4.7), la alineación es casi óptima. La distribución del daño se localiza cerca de la base de la aguja. Esto significa que en la zona de la punta de la aguja, no hay interrupción en la superficie. Dado que esta zona tiene menos puntos en comparación con la zona de la base, cualquier disrupción en ella puede influir en los resultados de la alineación.

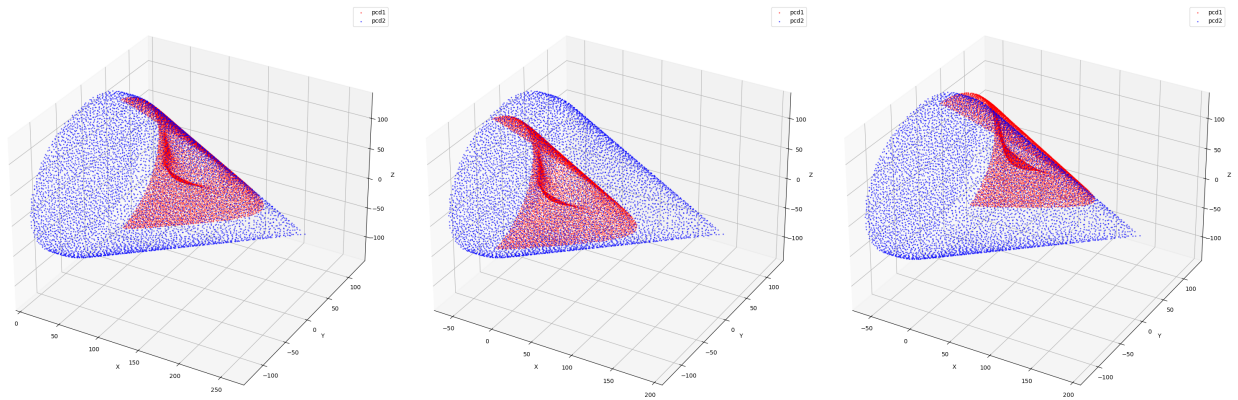


Figura 4.7: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-A-S

El proceso de alineación de las nubes de puntos A-Aps-S (Figura 4.8) obtiene un muy buen resultado. Según el gráfico de la Figura 4.1, es el segundo mejor resultado con respecto a los modelos de daño en aguja Pelton. Una razón para esto puede deberse a que el modelo de la pieza dañada cubre una vista amplia de la aguja, contribuyendo a una mayor cantidad de puntos a considerar en la alineación.

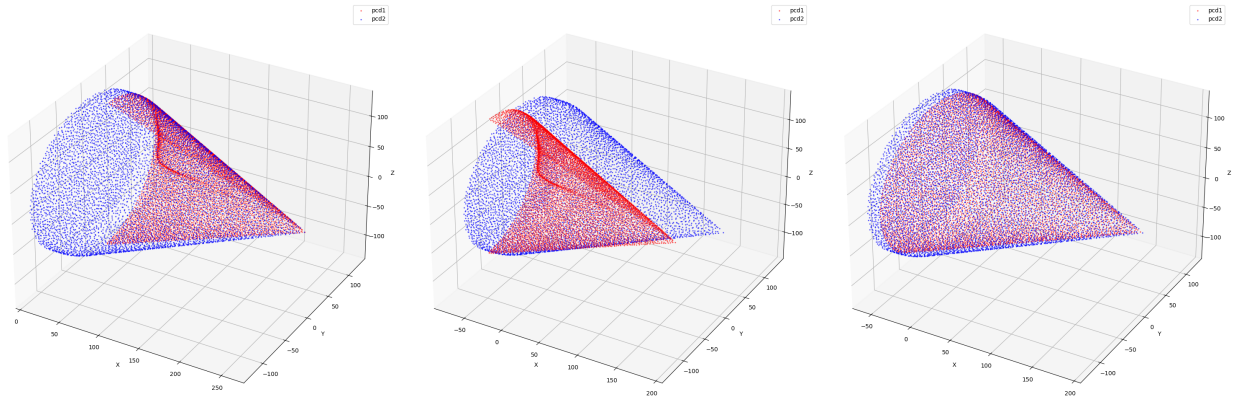


Figura 4.8: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-Aps-S

En la alineación del caso A-B-S (Figura 4.9), se observa una desalineación evidente entre la nube de puntos del daño y la nube de puntos de la pieza original. Teniendo en cuenta que el modelo de la pieza dañada representa solo una pequeña porción del modelo total, este resultado es previsible. Una mejora se podría lograr al escanear una mayor área de la pieza dañada, lo cual proporcionaría al algoritmo de alineación más puntos para considerar en el proceso de correspondencia entre las nubes de puntos.

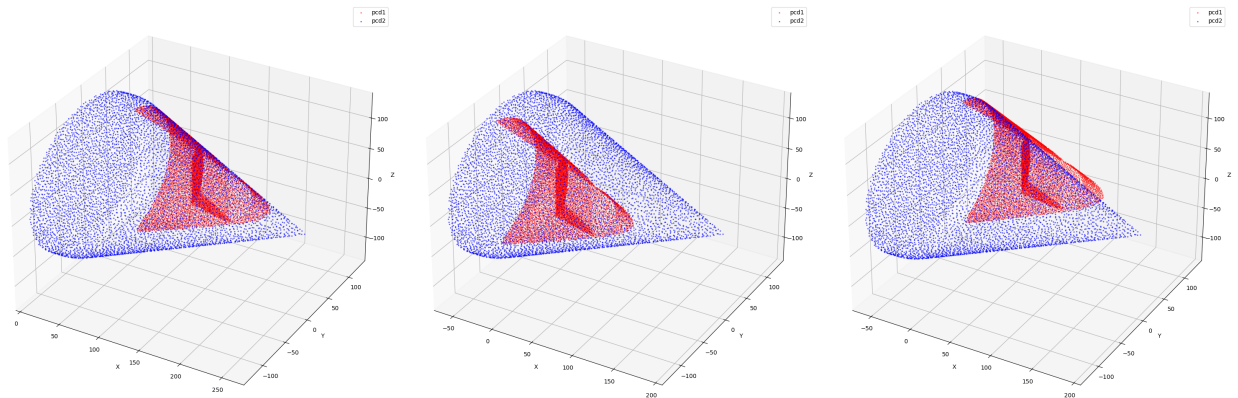


Figura 4.9: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-B-S

Es importante analizar la alineación en los casos A-P-S (Figura 4.10) y A-PR-S (Figura 4.11). Aunque ambos tipos de daños son similares (daño tipo pitting), su distribución varía. En A-P-S, el daño se concentra en la zona central al eje longitudinal, lo que recuerda al caso A-BR-S, donde el daño también está focalizado en una zona específica, resultando en una alineación desplazada de lo óptimo. En contraste, en A-PR-S, el daño está más disperso, lo que lleva a un resultado de alineación significativamente mejor.

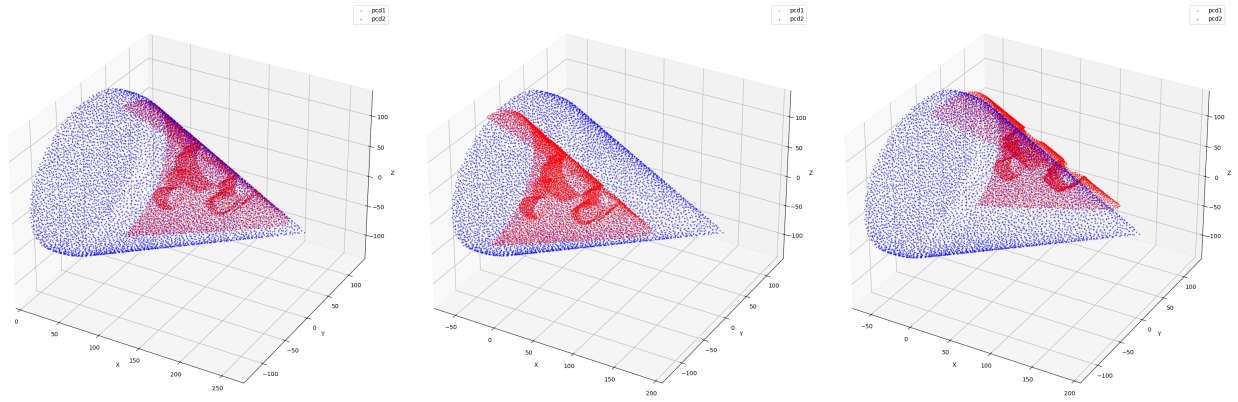


Figura 4.10: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-P-S

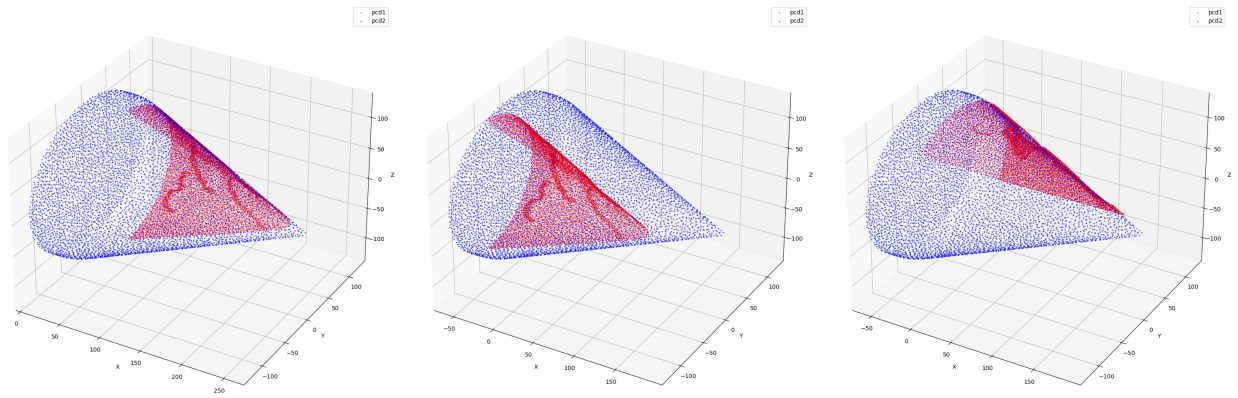


Figura 4.11: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos A-PR-S

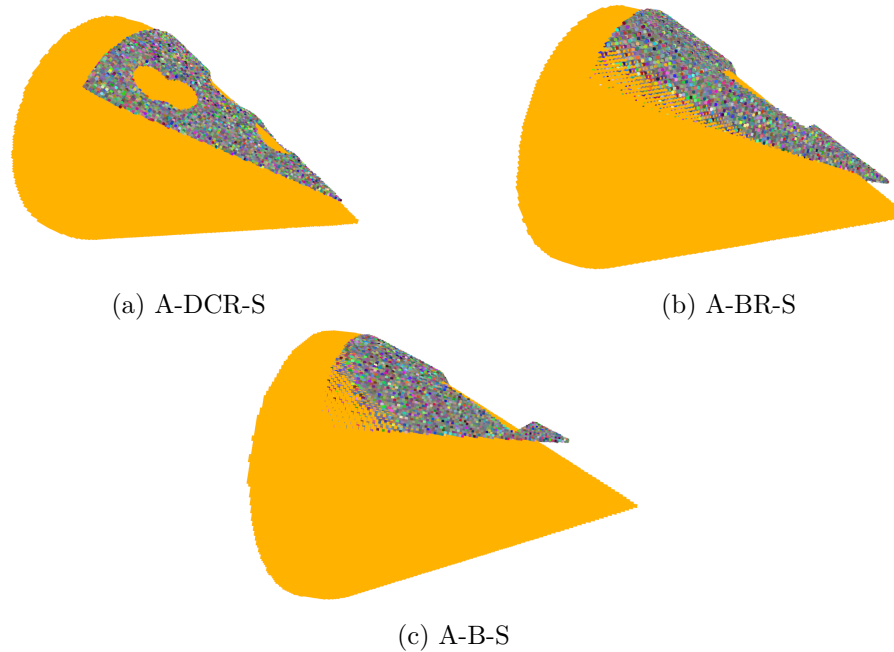


Figura 4.12: Visualización de la alineación del modelo con daño y el modelo original. Dependiendo de la distribución del daño, la alineación resultante se desplaza de la alineación esperada. En (a) el daño se encuentra distribuido a lo largo de la superficie y la alineación es buena. En (b) el daño se encuentra centrado en una zona y la alineación sufre un desplazamiento considerable de la posición esperada. En (c) el daño se encuentra concentrado hacia la punta de la aguja y sufre un desplazamiento grande de la posición esperada.

Los resultados de la alineación en el conjunto de mallas de la aguja Pelton son inferiores en comparación con los de la malla cilíndrica y el espejo Pelton. La principal causa es que en la aguja Pelton, la distribución del volumen varía a lo largo del eje longitudinal, siendo la base mucho más voluminosa que la punta. Esto resulta en una densidad de puntos más alta en la base que en la punta. Al aplicar el algoritmo ICP, este puede conformarse con alinear adecuadamente solo los puntos de la base, descuidando la alineación global de toda la geometría.

Adicionalmente, los escaneos 3D de la pieza dañada representan solo una parte de la pieza completa. Esto provoca que no haya una correspondencia directa entre los puntos de la pieza dañada y los de la pieza intacta. Por último, dado que la pieza dañada y la original difieren a causa del deterioro, pueden surgir correspondencias entre los puntos que definen la superficie dañada y aquellos de la pieza original. Esto podría resultar en un menor error cuadrático medio, pero sin relevancia práctica.

4.1.2. Cilindro

Los resultados de alineación de los cilindros se mantienen consistentes para todos los tipos de daños, como se muestra en las Figuras 4.13, 4.14, 4.15, 4.16 y 4.17. En cada caso, se logra una alineación satisfactoria con un error marginal. A diferencia de la aguja Pelton, el cilindro presenta una distribución uniforme tanto en los ejes longitudinales como radiales. Esto limita las posibles soluciones al problema de alineación, evitando así que el algoritmo se estanque en mínimos locales.

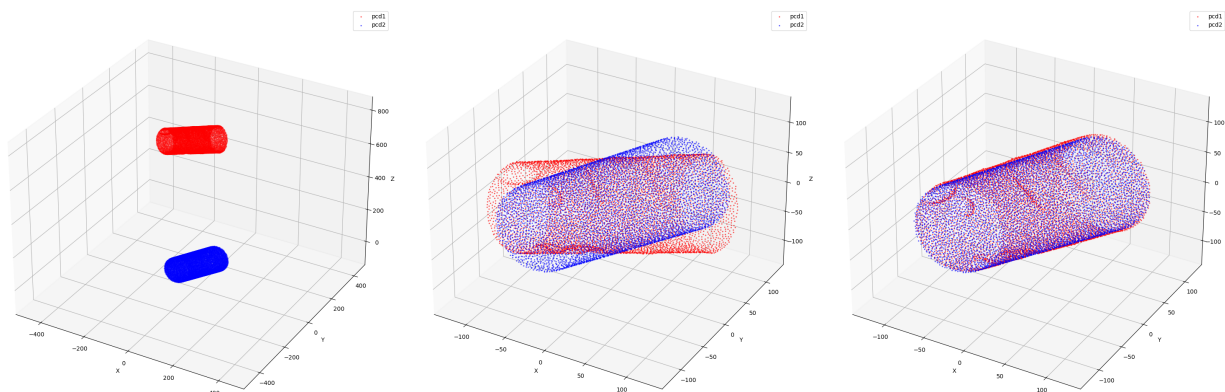


Figura 4.13: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos C-AP-RA

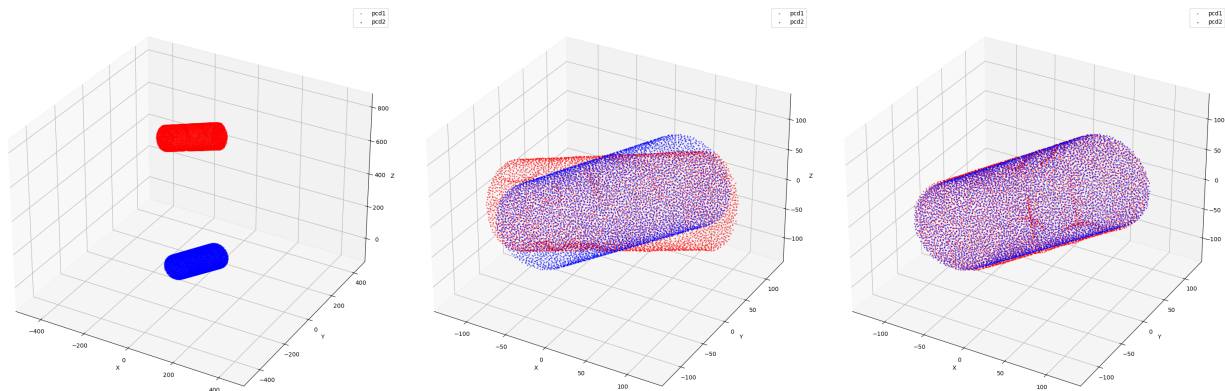


Figura 4.14: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos C-AP-RC

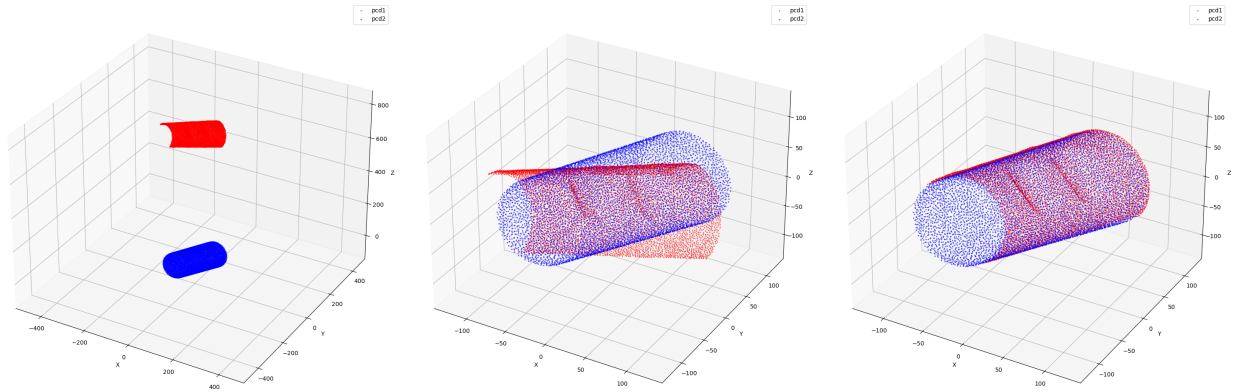


Figura 4.15: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos C-Aps-R

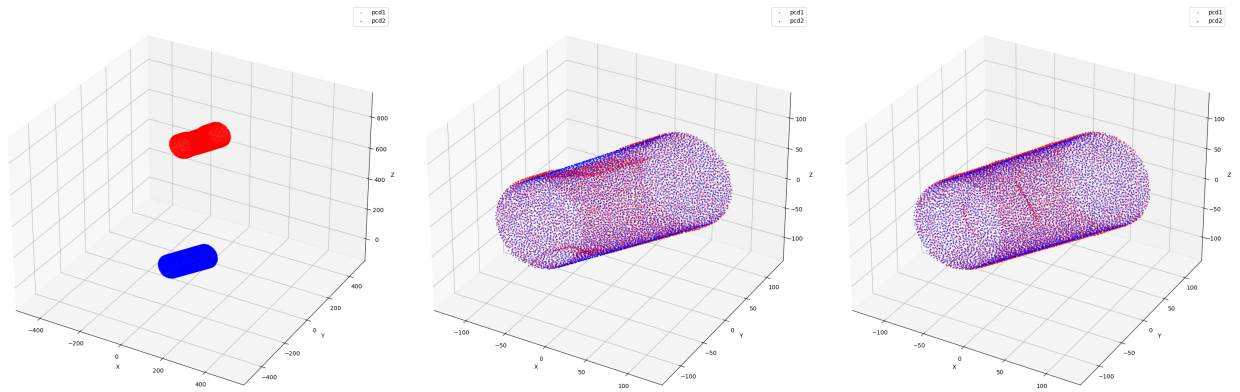


Figura 4.16: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos C-B-R

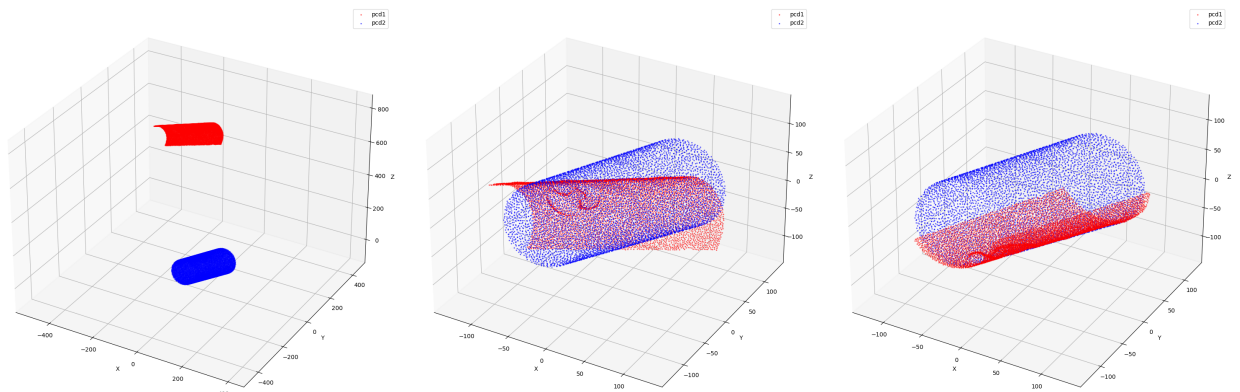


Figura 4.17: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos C-P-R

4.1.3. Espejo Pelton

En los modelos del espejo inyector Pelton, la pieza dañada es idéntica a la original salvo por un daño artificial. Esto implica que al alinear los centroides, los modelos quedan automáticamente alineados. Evaluar el sistema sobre estos modelos ya alineado sirve para detectar errores en el software. Si el sistema no logra mantener las dos nubes de puntos alineadas, indicaría un error grave en la implementación. Los resultados en las Figuras 4.18, 4.19 y 4.20 confirman que la alineación de las nubes de puntos es consistente en todos los casos examinados.

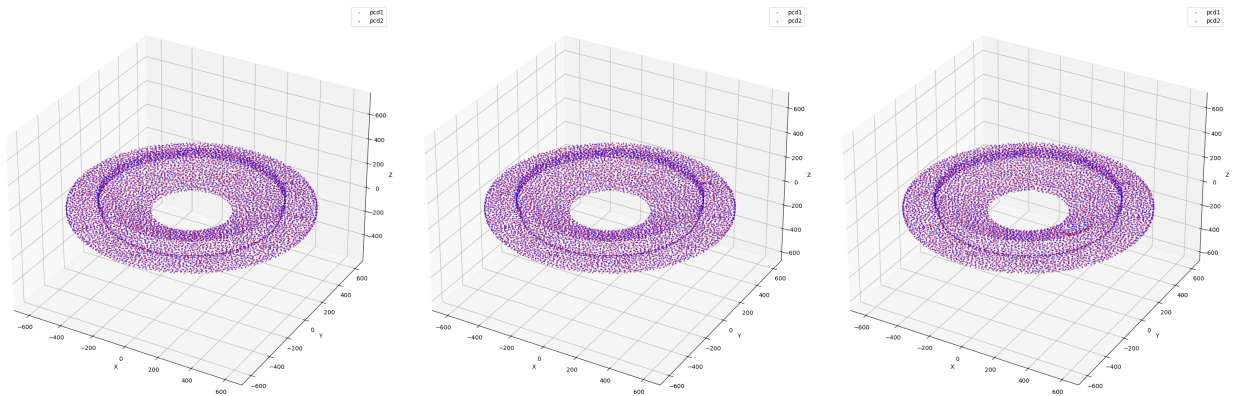


Figura 4.18: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos EIP-B-S

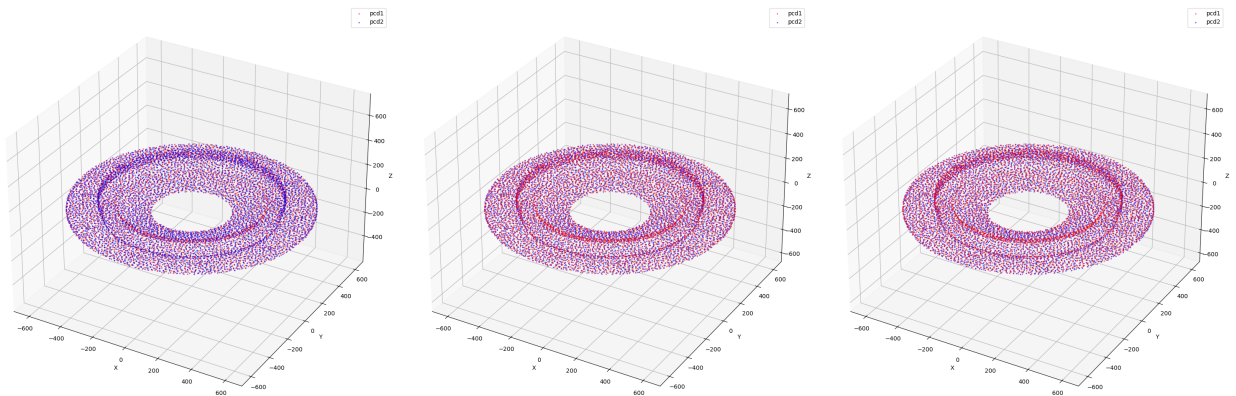


Figura 4.19: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos EIP-R1-S

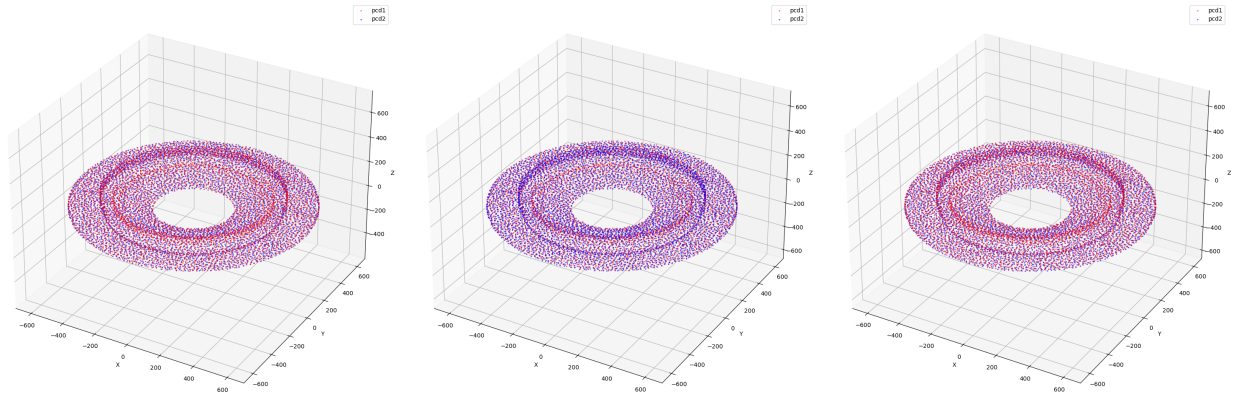


Figura 4.20: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos EIP-R2-S

4.1.4. Placas

En la Figura 4.21 se presenta el resultado de la alineación sobre los modelos PL-Aps-R. La alineación obtenida es incorrecta. Este resultado erróneo se debe a las particularidades geométricas del modelo de placas. El modelo de la pieza original corresponde a una malla cerrada con forma rectangular. Al generar una nube de puntos a partir de esta malla, se aprecian dos planos del mismo tamaño y la misma cantidad de puntos. La forma de los modelos de la pieza dañada corresponde globalmente a un plano. Intuitivamente, existen cuatro soluciones posibles para la alineación:

- Alineación con el plano superior con la pieza dañada orientada correctamente.
- Alineación con el plano superior con la pieza dañada orientada incorrectamente.
- Alineación con el plano inferior con la pieza dañada orientada correctamente.
- Alineación con el plano inferior con la pieza dañada orientada incorrectamente.

De estas cuatro soluciones solo dos se pueden clasificar de exitosas. Los resultados experimentales corroboran esto obteniendo solo una alineación exitosa entre los cuatro tipos de daños.

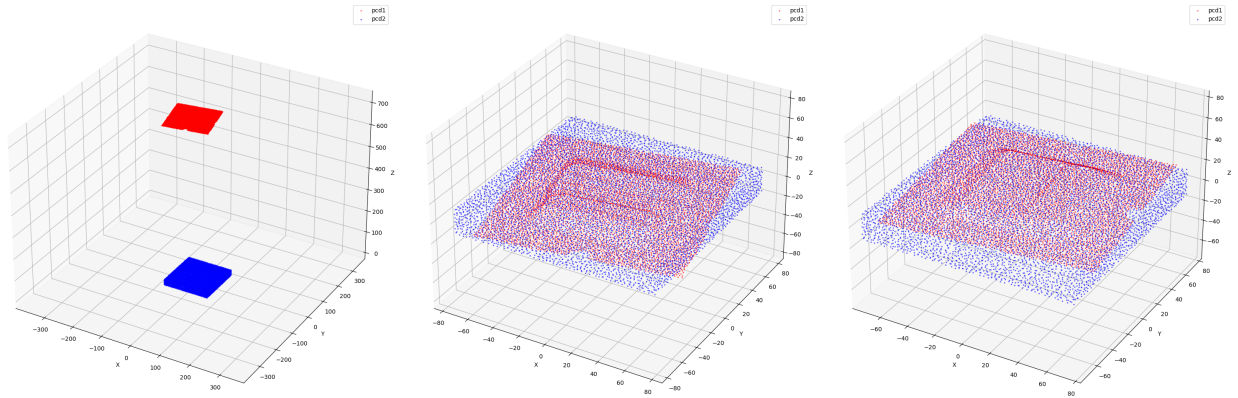


Figura 4.21: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos PL-Aps-R

En la Figura 4.22 se muestra el resultado de alineación de los modelos PL-B-R. La alineación es incorrecta, el modelo de la pieza dañada se alineó con la diagonal de la placa original. Este resultado es congruente con la medida del error mostrada en el gráfico de la Figura ??.

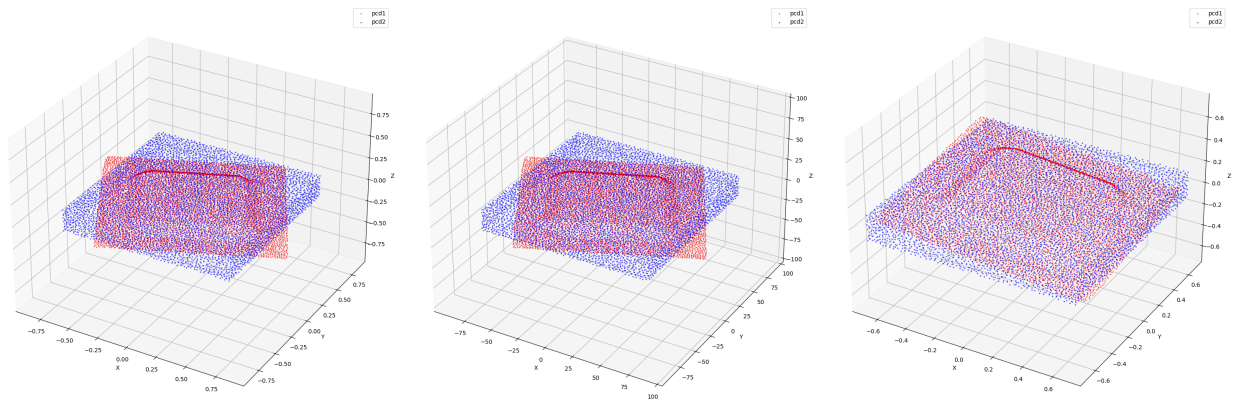


Figura 4.22: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos PL-B-R

En la Figura 4.23 se muestra el resultado de alineación de los modelos PL-E-R. Estrictamente, esta corresponde a una alineación incorrecta ya que lo esperado es una alineación con la parte superior del modelo de la placa original. Sin embargo, la alineación obtenida es una simetría rotacional de la esperada y es útil para el proceso de identificación de daño.

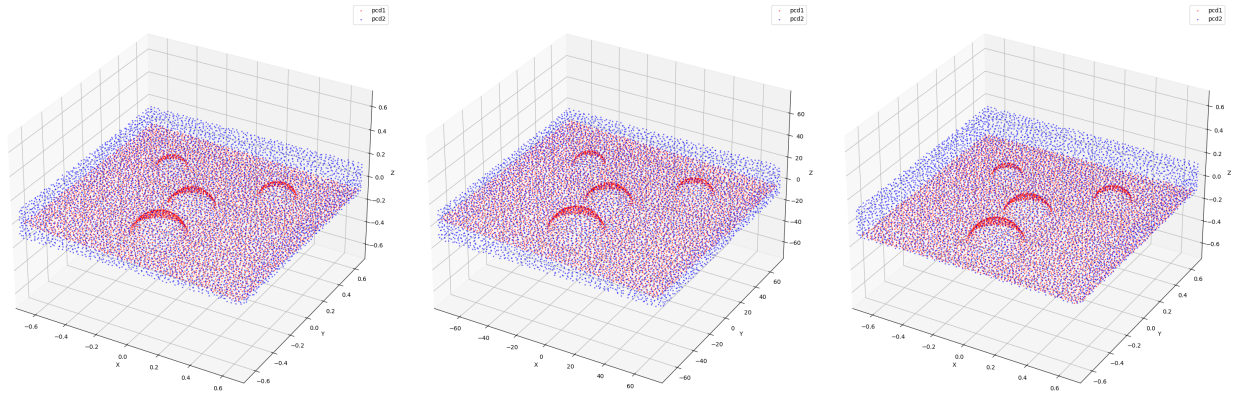


Figura 4.23: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos PL-E-R

En la Figura 4.24 se muestra un resultado similar al del caso PL-Aps-R. El plano se alinea correctamente con uno de los planos de la placa pero la orientación es incorrecta. Esta alineación no permite una correcta identificación del daño.

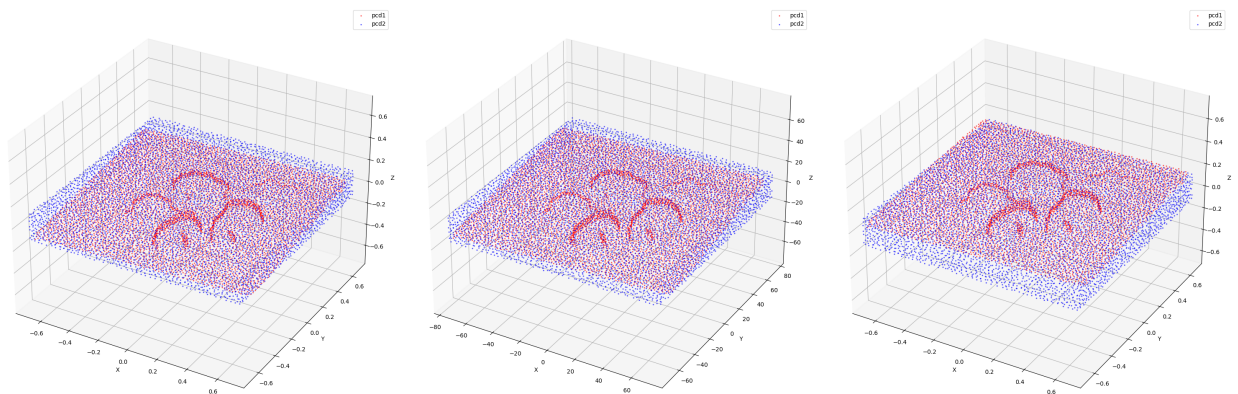


Figura 4.24: Proceso de alineación de las nubes de puntos. De izquierda a derecha: posición inicial de las nubes de puntos, desplazamiento de los centroides al origen y resultado final de la alineación. Modelos PL-P-R

4.1.5. Modelos generados sintéticamente

Dado que se han obtenido resultados positivos en el proceso de alineación para todos los modelos generados sintéticamente, no se identifican errores de alineación que requieran un análisis mediante la visualización de las nubes de puntos.

4.2. Resultados del proceso de identificación del daño

En esta sección, se presentan los resultados obtenidos del proceso de identificación del daño. Inicialmente, el gráfico de la Figura 4.25 ilustra el error porcentual en la identificación del daño en comparación con el volumen real del daño de la pieza. Se calcula el volumen real del daño convirtiendo las mallas a un formato de voxel y ejecutando una operación booleana con la envoltura convexa de la malla dañada. Este enfoque, equivalente a lograr una alineación perfecta con la pieza original, solo es aplicable a mallas de piezas originales convexas. Las mallas de espejos Pelton, que no son convexas, y algunos escaneos de cilindros con errores de mallado que impiden la obtención de la envoltura convexa adecuada, no se incluyen en el análisis. Asimismo, los modelos de placas se excluyen del gráfico debido a que el proceso de identificación de daño no fue exitoso.

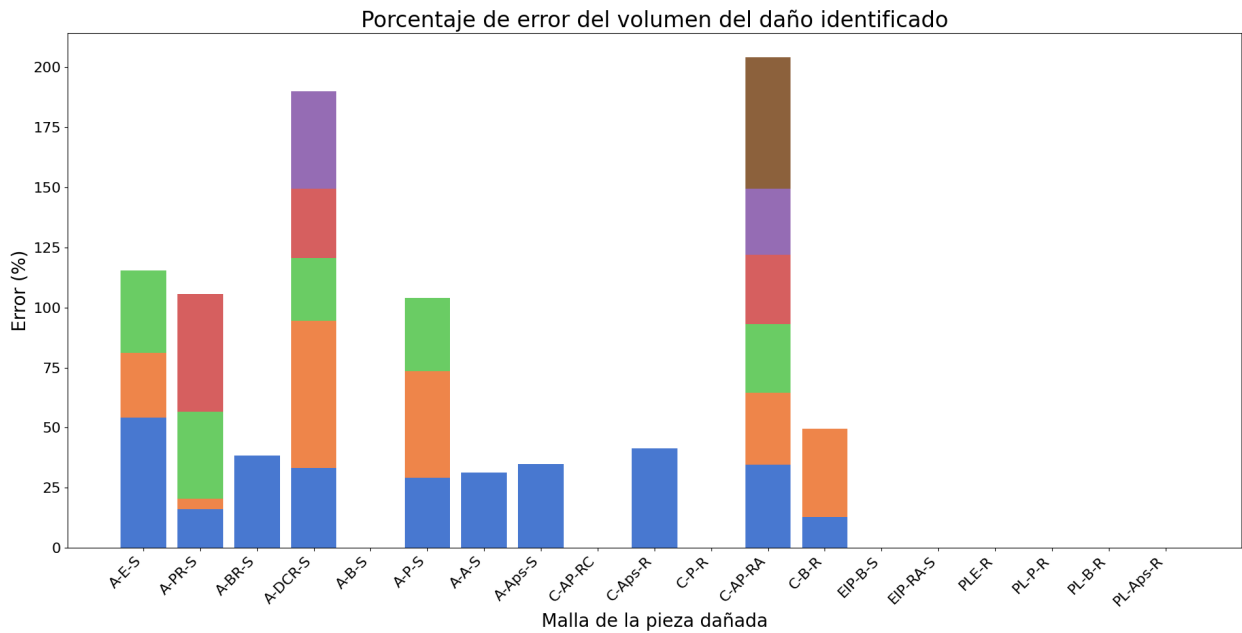


Figura 4.25: Error porcentual en el volumen del daño identificado para cada modelo en la base de datos de evaluación. Cada barra representa el error acumulado por zona dañada, con distintos colores para cada zona en modelos como el A-E-S. Modelos de piezas no convexas y placas, así como algunos escaneos de cilindros, son omitidos debido a limitaciones en la obtención de la envoltura convexa y errores de malla.

El gráfico de la Figura 4.25 revela que el error porcentual por zona de daño varía entre el 25 % y el 50 %, con ciertos valores extremos. Este error, definido en la Ecuación 4.1, tiende a ser mayor en modelos con alineaciones deficientes. Factores como la alineación, la resolución de la grilla de voxels, las características geométricas del daño, y su posición relativa en la pieza influyen significativamente en el tamaño del error.

$$Error = 100 \times \left| \frac{V_{id} - V_{real}}{V_{real}} \right| \quad (4.1)$$

- V_{id} : Volumen identificado.
- V_{real} : Volumen real.

Los errores observados no cumplen con las expectativas para aplicaciones de alta precisión, como la reparación mediante manufactura robótica. No obstante, la claridad sobre las fuentes de error abre vías para futuras mejoras del sistema y la reducción de estos márgenes.

Adicionalmente, el gráfico de la Figura 4.26 expone el error porcentual en la identificación del daño en la base de datos sintética, la cual incluye únicamente modelos de vistas completas. El volumen real se obtiene directamente de la malla geométrica, calculando el volumen de la pieza original y el dañado, siendo la diferencia entre estos el volumen de la zona dañada.

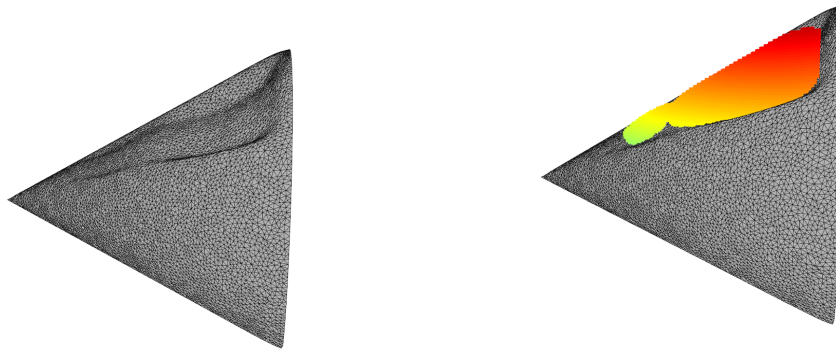


Figura 4.26: Error porcentual en el volumen del daño identificado para modelos de la base de datos sintética, utilizando la diferencia de volumen entre las mallas geométricas de las piezas original y dañada como referencia.

La Figura 4.26 indica errores porcentuales similares a los observados en la Figura 4.25. Notablemente, los daños donde una pieza se ha separado completamente (como un diente dañado en un engranaje o una punta quebrada de un cono) presentan un error significativamente menor. Esto se debe probablemente a que una menor superficie de contacto entre el volumen dañado y la pieza original reduce la cantidad de voxels erróneamente eliminados durante la operación booleana, lo que a su vez minimiza el error en la identificación del daño.

4.2.1. Aguja de turbina Pelton

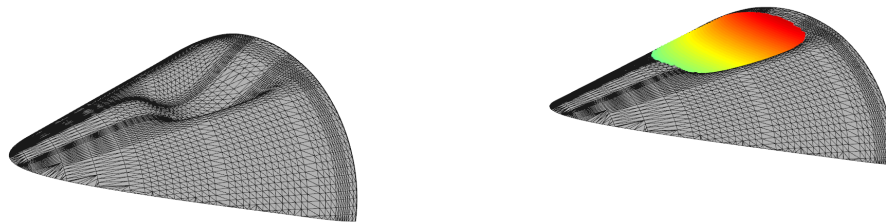
Considerando que la alineación obtenida para las mallas correspondientes a la aguja de turbina Pelton fue deficiente, el proceso de identificación del daño también se ve afectado. Sin embargo, como se ve en las Figuras 4.27 y 4.28 el volumen de daño no identificado es considerablemente menor al volumen identificado. Esto muestra que el algoritmo de identificación de daño es robusto a pequeños errores de alineación.



(a) Pieza dañada.

(b) Pieza dañada con el daño identificado.

Figura 4.27: Identificación de daño para A-Aps-S

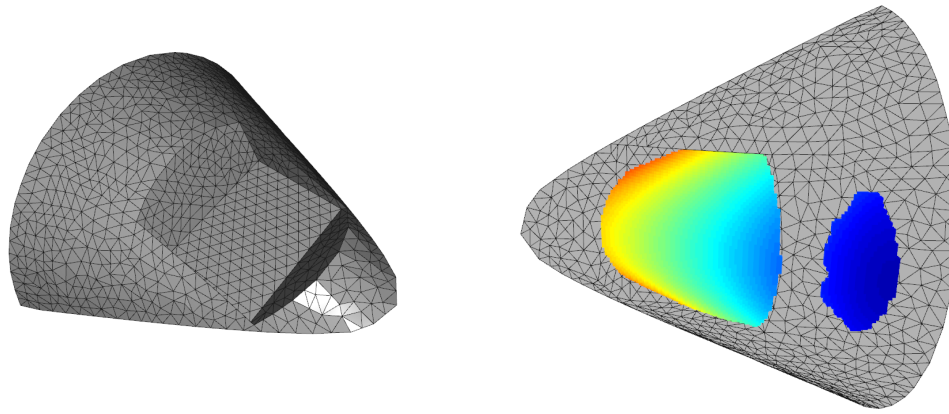


(a) Pieza dañada.

(b) Pieza dañada con el daño identificado.

Figura 4.28: Identificación de daño para A-A-S

En la Figura 4.29 se muestra la identificación del daño para el modelo A-B-S. Producto de una alineación incorrecta, se identifica un daño no existente (en color azul). No obstante, igualmente el sistema logra identificar el daño real en la pieza.

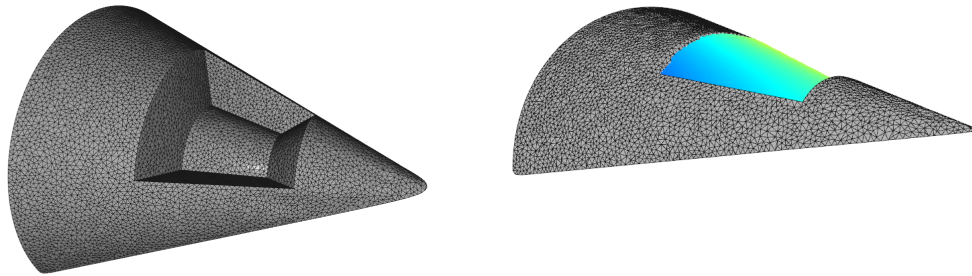


(a) Pieza dañada.

(b) Pieza dañada con el daño identificado.

Figura 4.29: Identificación de daño para A-B-S

En la Figura 4.30 se muestra la identificación del daño para el modelo A-BR-S. Una correcta alineación de ambos modelos da como resultado una correcta identificación del daño. Todo el volumen correspondiente a la zona con daño es identificado.



(a) Pieza dañada.

(b) Pieza dañada con el daño identificado.

Figura 4.30: Identificación de daño para A-BR-S

La Figura 4.31 ilustra la identificación del daño para el modelo A-BR-S, mostrando que todas las zonas dañadas han sido identificadas. No obstante, debido a pequeños errores de alineación, el volumen del daño identificado no engloba por completo la zona afectada. Mejorar el sistema de alineación de modelos podría optimizar estos resultados, logrando una cobertura más precisa del daño identificado.

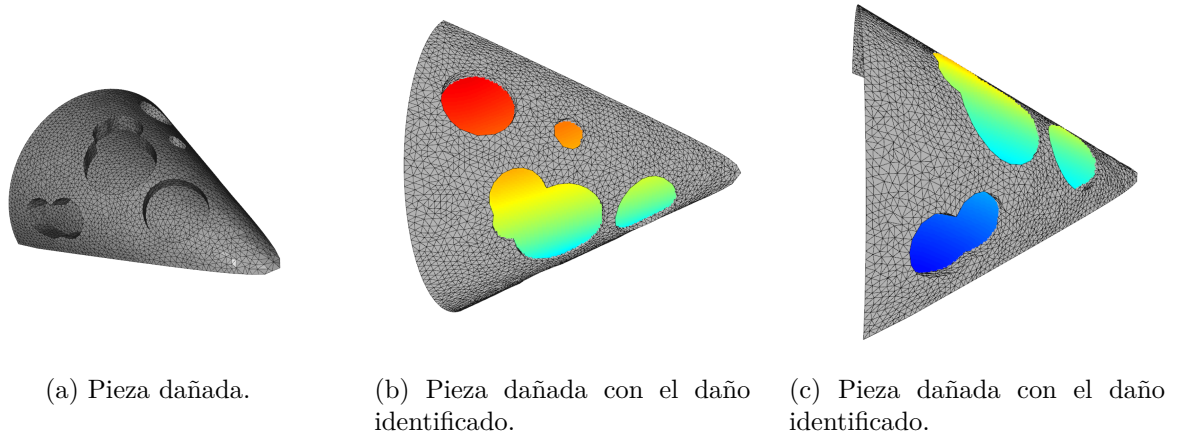


Figura 4.31: Identificación de daño para A-DCR-S

La Figura 4.32 presenta la identificación del daño para el modelo A-E-S, mostrando que, en las áreas reconocidas como dañadas, el volumen del daño identificado cubre completamente estas zonas. Sin embargo, hay una pequeña área donde el daño no fue detectado, posiblemente debido a que el algoritmo de agrupamiento clasificó esa sección como ruido a causa de la baja cantidad de voxels presentes. Ajustar los parámetros del sistema de identificación de daño podría remediar este problema y asegurar una detección más precisa y completa.

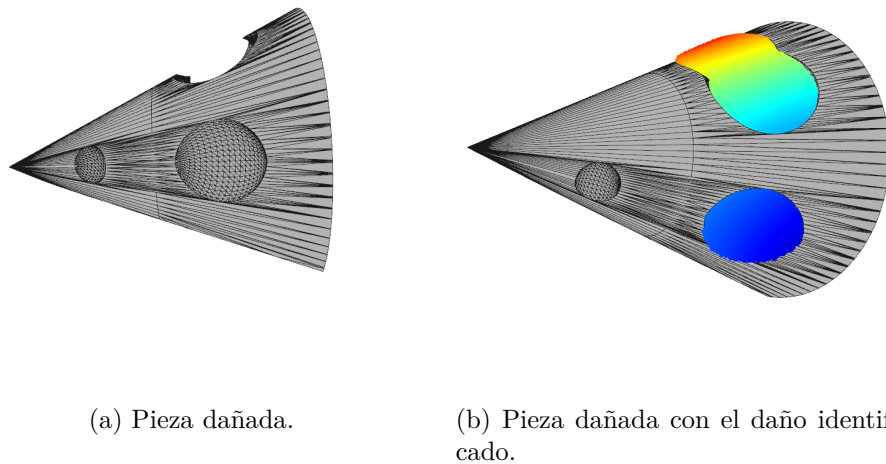


Figura 4.32: Identificación de daño para A-E-S

La Figura 4.33 muestra la identificación del daño para el modelo A-P-S, donde se logra identificar todas las zonas dañadas. No obstante, a causa de errores de alineación, el volumen del daño identificado no abarca completamente la zona afectada. Similarmente al caso A-BR-S, perfeccionar el sistema de alineación podría conducir a una identificación más exhaustiva y precisa del daño.

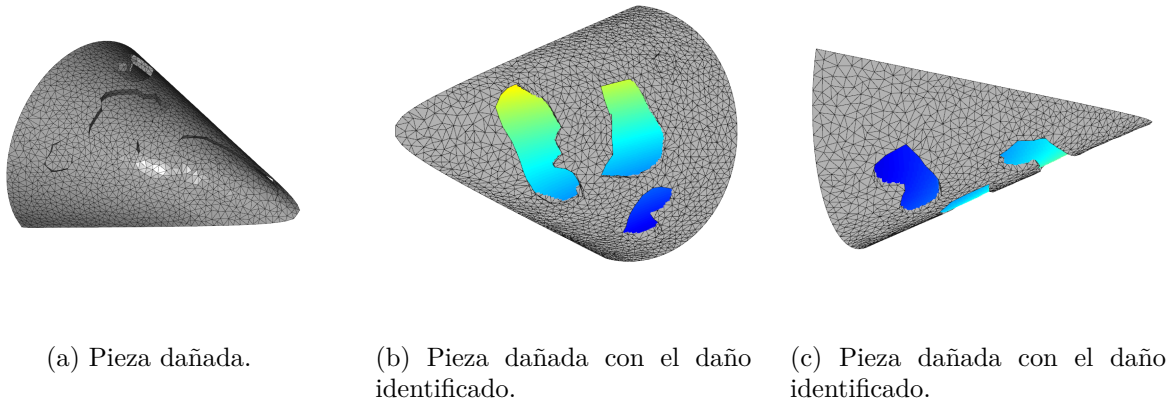


Figura 4.33: Identificación de daño para A-P-S

La Figura 4.34 muestra la correcta y completa identificación del daño para el modelo A-PR-S. Dado el resultado correcto de alineación de modelos, el sistema de identificación de daños logra identificar correctamente todo el volumen correspondiente al daño de la pieza.

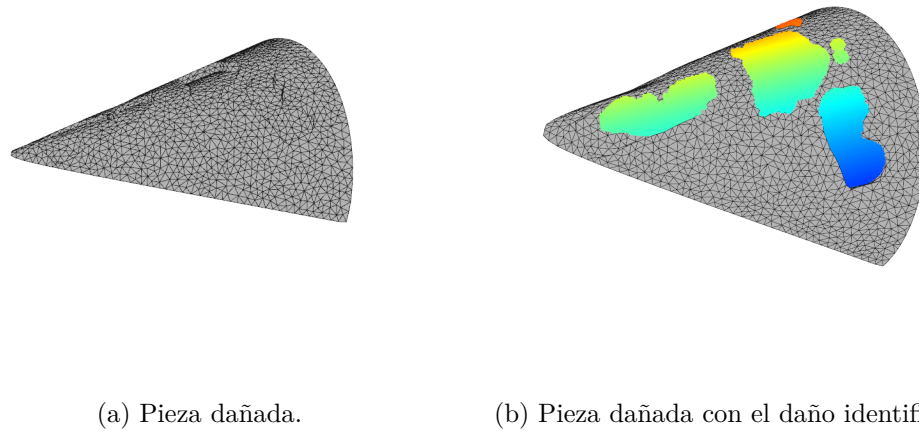


Figura 4.34: Identificación de daño para A-PR-S

4.2.2. Cilindro

Considerando los buenos resultados obtenidos en el proceso de alineación para los modelos cilíndricos, se tienen buenos resultados en el proceso de identificación de daño.

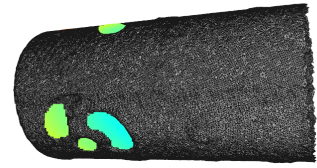
Las Figuras 4.35, 4.36, 4.37, 4.38 y 4.39 demuestran que el sistema de identificación de daños identifica correctamente y de forma completa todo el volumen del daño en la pieza. Este resultado se mantiene constante tanto en visualizaciones parciales y detalladas de la zona afectada como en escaneos completos de la pieza. La eficacia en la identificación de daños se atribuye al buen resultado del sistema de alineación.



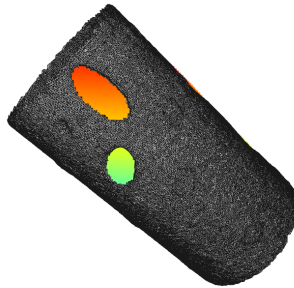
(a) Pieza dañada.



(b) Pieza dañada con el daño identificado.

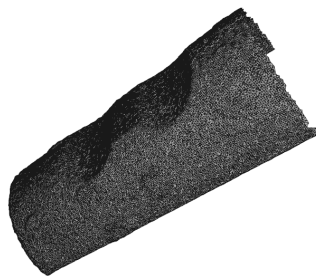


(c) Pieza dañada con el daño identificado.

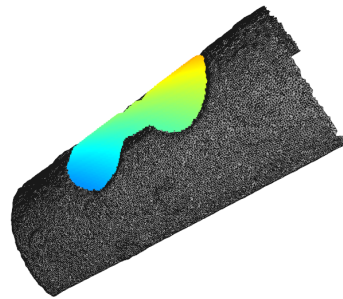


(d) Pieza dañada con el daño identificado.

Figura 4.35: Identificación de daño para C-AP-RA

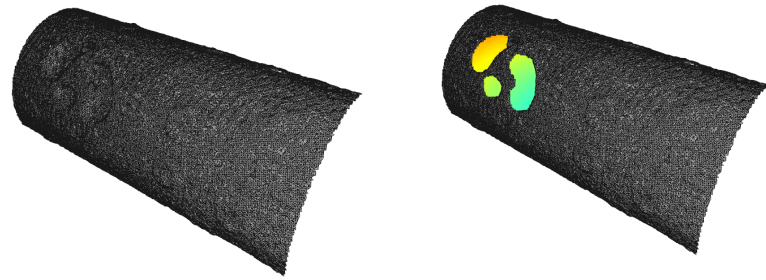


(a) Pieza dañada.



(b) Pieza dañada con el daño identificado.

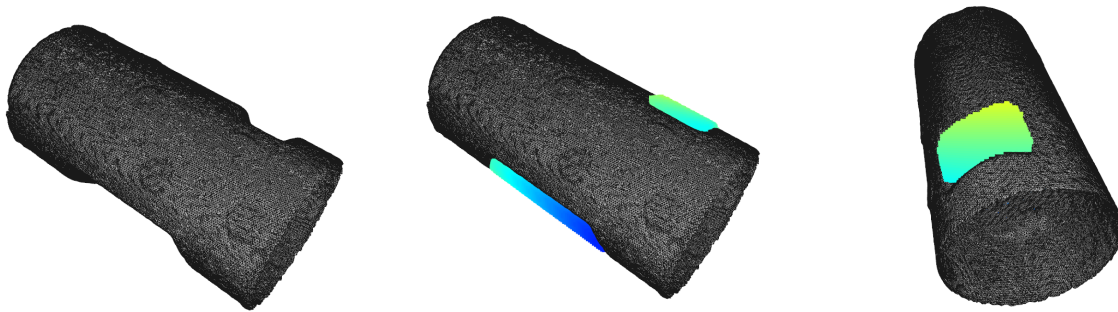
Figura 4.36: Identificación de daño para C-Aps-R



(a) Pieza dañada.

(b) Pieza dañada con el daño identificado.

Figura 4.37: Identificación de daño para C-P-R



(a) Pieza dañada.

(b) Pieza dañada con el daño identificado.

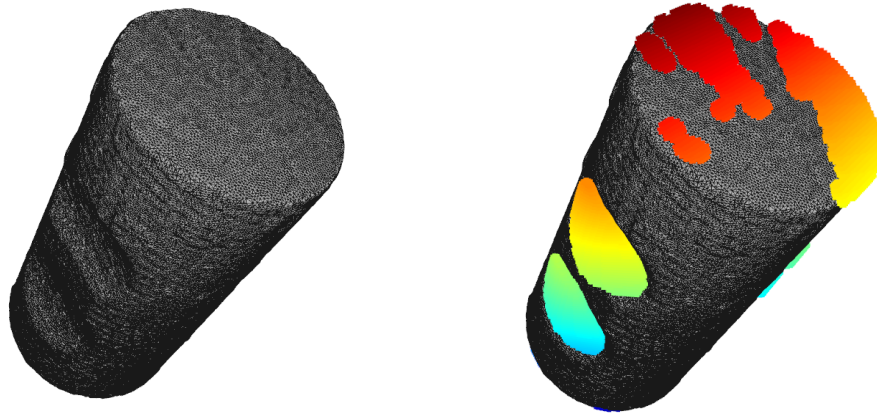
(c) Pieza dañada con el daño identificado.

Figura 4.38: Identificación de daño para C-B-R

La Figura 4.39 exhibe un falso positivo en la identificación del daño, específicamente en la tapa superior del cilindro. Este error puede atribuirse principalmente a que el modelo C-AP-RC es un modelo cerrado, a diferencia de otros modelos cilíndricos, lo que introduce variaciones en la nube de puntos y longitud del modelo. Los posibles motivos para esta identificación errónea incluyen:

- Desalineación por adición de tapas: La incorporación de tapas en el cilindro introduce nuevos puntos en la nube de puntos, lo que puede provocar desalineaciones menores durante la correspondencia con el modelo original.
- Error dimensional por longitud adicional: La adición de tapas también aumenta la longitud del modelo del cilindro. Si este fue generado a partir de un escaneo 3D, el error inherente a este método puede resultar en dimensiones incorrectas.
- Interpretación de superficie rugosa como daño: Los escaneos 3D pueden crear superficies con texturas rugosas que el sistema de identificación interpreta erróneamente como daños.

Una solución práctica para mitigar este problema es utilizar un modelo que represente solo una vista parcial y enfocada de la zona dañada. Esto limitaría la inclusión de áreas innecesarias que podrían conducir a errores en la identificación del daño.



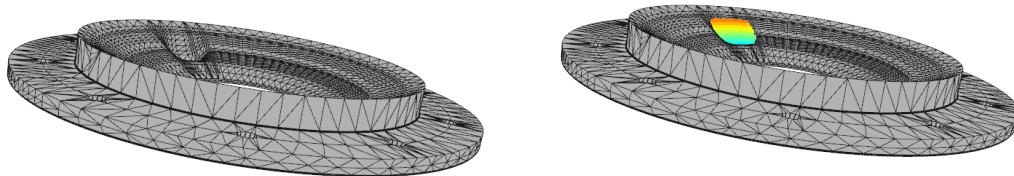
(a) Pieza dañada.

(b) Pieza dañada con el daño identificado.

Figura 4.39: Identificación de daño para C-AP-RC

4.2.3. Espejo Pelton

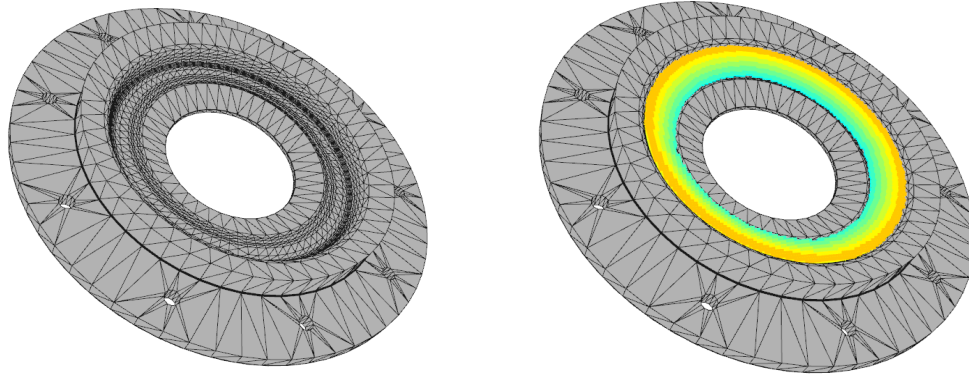
Al igual que con los modelos cilíndricos, el proceso de alineación para los modelos de espejos Pelton obtiene resultados satisfactorios. Esto significa que el sistema de identificación de daños puede funcionar correctamente e identificar completamente el volumen correspondiente al daño. Esto se comprueba en las Figuras 4.40, 4.41 y 4.42. El sistema identifica correctamente cada tipo de daño presente en el espejo Pelton.



(a) Pieza dañada.

(b) Pieza dañada con el daño identificado.

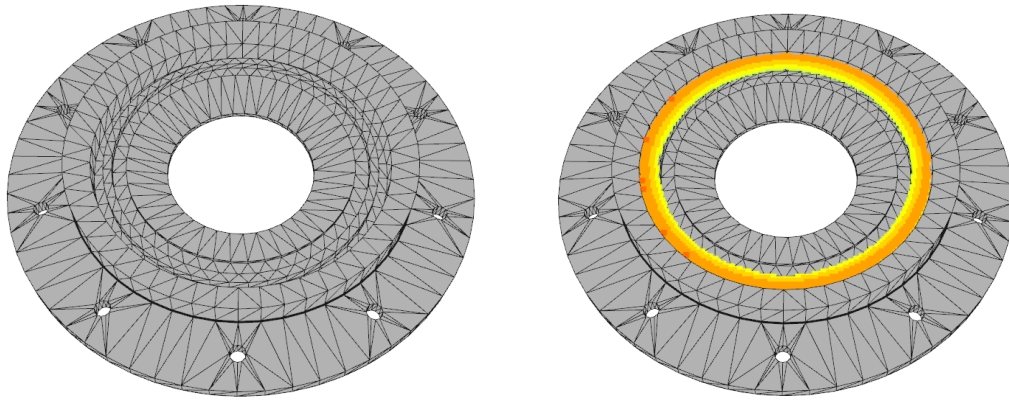
Figura 4.40: Identificación de daño para EIP-B-S



(a) Pieza dañada.

(b) Pieza dañada con el daño identificado.

Figura 4.41: Identificación de daño para EIP-R1-S



(a) Pieza dañada.

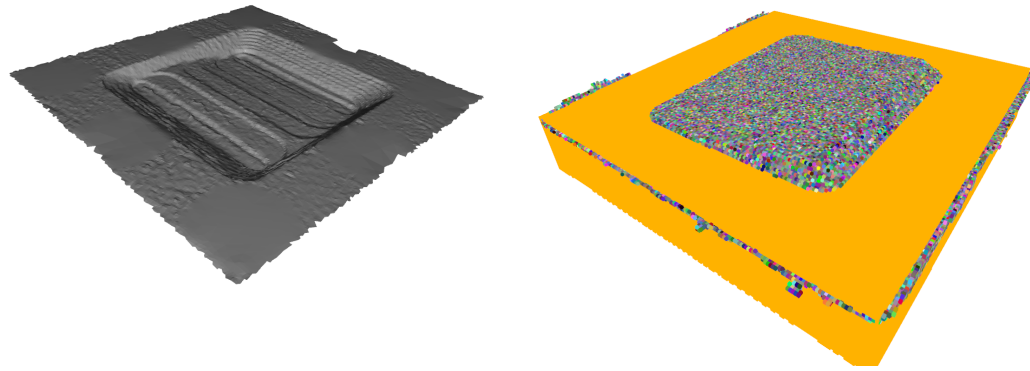
(b) Pieza dañada con el daño identificado.

Figura 4.42: Identificación de daño para EIP-R2-S

4.2.4. Placa

Considerando los problemas de alineación de los modelos correspondiente a placas, el sistema de identificación de daño falla y no logra identificar el volumen correspondiente al daño de la pieza.

En la Figura 4.43 se muestra la pieza dañada y un detalle de la alineación incorrecta entre las representaciones de grilla de voxels de la pieza dañada y la pieza original. Producto de la mala alineación, la operación booleana de diferencia solo genera un volumen principal de la pieza y agrupaciones de ruido.



(a) Pieza dañada.

(b) Detalle de la alineación errónea.

Figura 4.43: Identificación de daño fallido PL-Aps-R

La Figura 4.44 ilustra cómo una alineación errónea puede resultar en la identificación de daños inexistentes. Debido a que la alineación posiciona la pieza dañada junto a la diagonal interior de la pieza original, la operación booleana de diferencia resulta en un volumen equivalente a la pieza original dividida por su diagonal. Este volumen es incorrectamente identificado como el daño de la pieza.



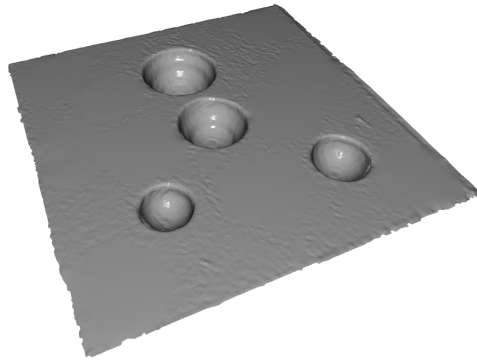
(a) Pieza dañada.

(b) Pieza dañada.

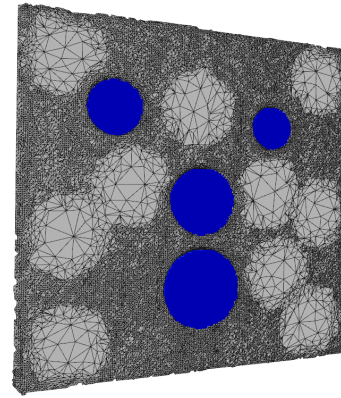
(c) Detalle de la identificación de daño fallido.

Figura 4.44: Identificación de daño fallido PL-B-R

En la Figura 4.45, se exhibe la correcta identificación del daño para el modelo PL-E-R. Esto se debe a que el módulo de alineación de mallas logró una alineación adecuada para el modelo en cuestión, permitiendo así una identificación exitosa del daño. Sin embargo, cabe destacar que para los modelos tipo placas los resultados del módulo de alineación no son consistentes y pueden cambiar de una ejecución a otra.



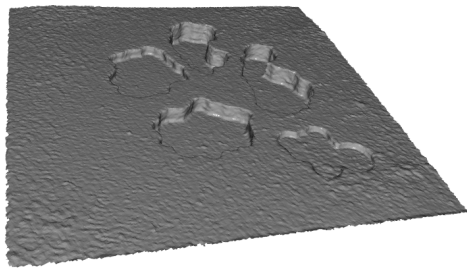
(a) Pieza dañada.



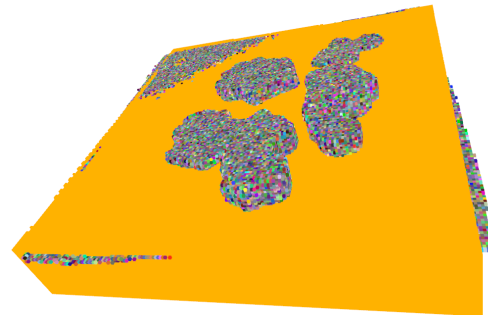
(b) Pieza dañada con el daño identificado.

Figura 4.45: Identificación de daño PL-E-R

En la Figura 4.46 se presenta un caso similar al del modelo PL-Aps-R. Una alineación errónea en donde el plano de la pieza dañada tiene una orientación incorrecta. Esto causa que el sistema de identificación de daño no identifique ningún tipo de daño en la pieza.



(a) Pieza dañada.

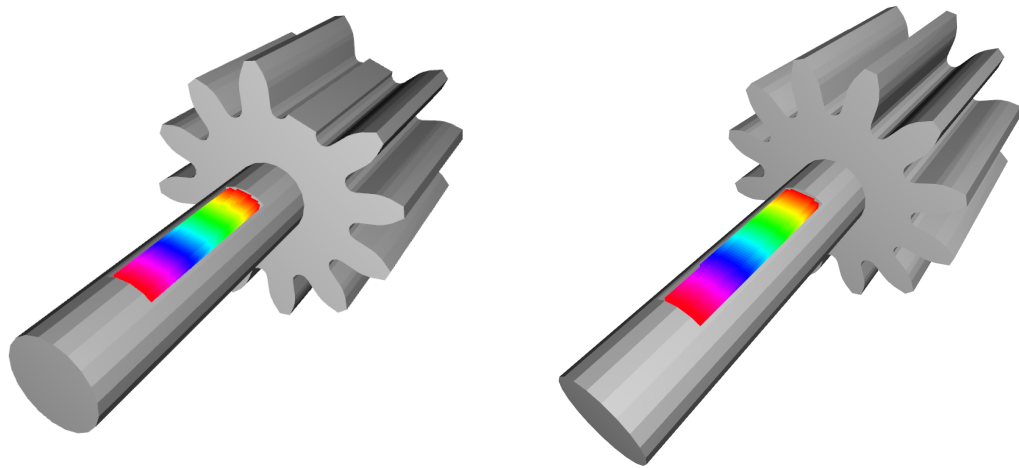


(b) Detalle de la alineación errónea.

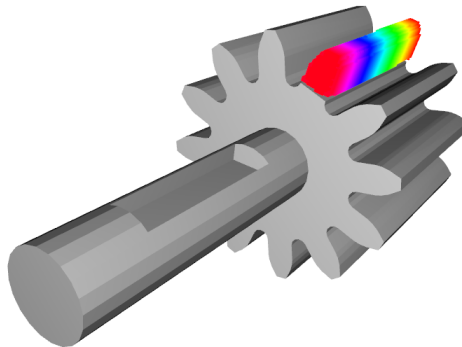
Figura 4.46: Identificación de daño fallido PL-P-R

4.2.5. Modelos generados sintéticamente

Se han creado modelos generados sintéticamente con el propósito de evaluar el rendimiento del sistema de identificación de daños bajo diversas condiciones. Estas condiciones incluyen la evaluación del sistema en geometrías no primitivas, con geometrías no convexas, y el rendimiento del sistema cuando se dispone de una vista completa, en contraposición a una vista parcial, de la pieza dañada. La Figura 4.47 ilustra la correcta identificación de daños en una pieza correspondiente a un eje con engranaje. El sistema de identificación de daños logra identificar de manera precisa el daño de bolsillo tanto cuando este es el único daño presente en la pieza (AXG-B) como cuando coexiste con otros daños (AXG-BT). Esto demuestra la capacidad de adaptación del sistema de identificación de daños. Además, el sistema puede identificar y modelar daños significativos en la pieza, como es el caso de un diente quebrado en un engranaje (AXG-BT). Es importante destacar que el acceso a una vista completa de la pieza dañada mejora sustancialmente el proceso de alineación de mallas y, por ende, el proceso de identificación de daños.



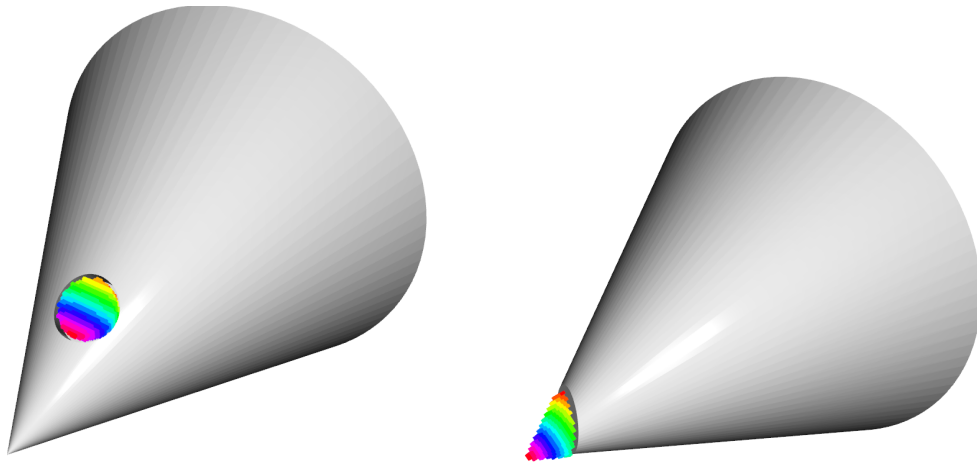
(a) Identificación de daño en malla AXG-B. (b) Identificación de daño en malla AXG-BT. Identificación de bolsillo en el eje.



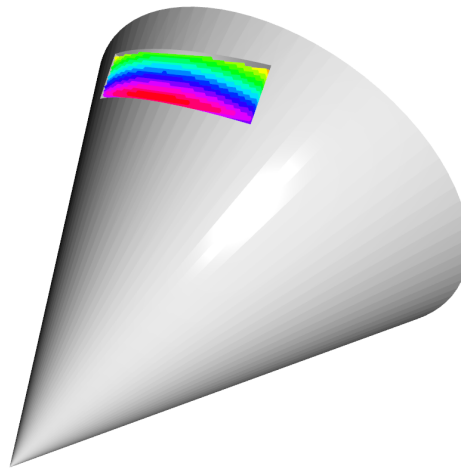
(c) Identificación de daño en malla AXG-BT. Identificación de daño en diente de engranaje.

Figura 4.47: Identificación de daño en mallas correspondientes a eje con engranaje.

En la Figura 4.48, se presenta la identificación de distintos tipos de daño en un cono. A diferencia de los modelos aguja Pelton, disponer de una vista completa de la pieza dañada en este caso permite lograr una alineación de mallas precisa y, como resultado, una identificación exitosa del daño. Similar al caso del diente dañado en un engranaje, el sistema es capaz de identificar partes faltantes en la pieza original (CO-PU).



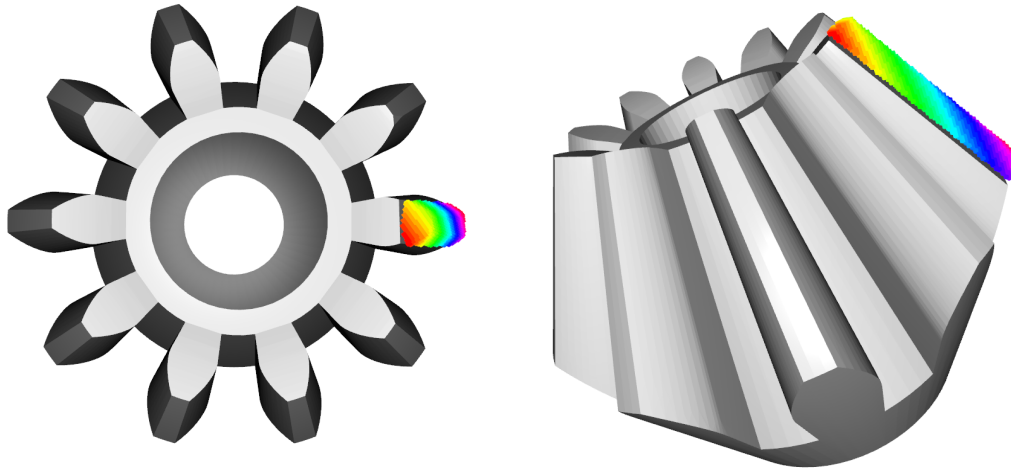
(a) Identificación de daño en malla CO-CP. (b) Identificación de daño en malla CO-PU.



(c) Identificación de daño en malla CO-RAN.

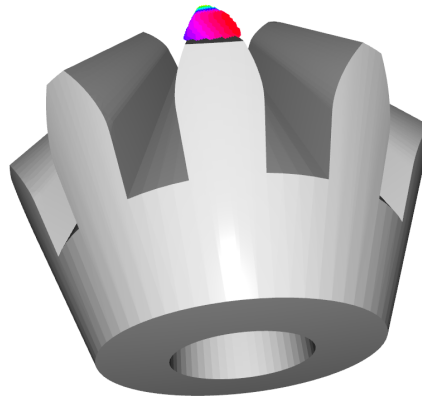
Figura 4.48: Identificación de daños en mallas correspondientes a conos generados sintéticamente.

La Figura 4.49 muestra la correcta identificación de daño en una pieza con geometría compleja. El sistema de identificación de daños logra identificar con precisión el diente faltante del engranaje sin generar identificaciones erróneas.



(a) Vista 1.

(b) Vista 2.



(c) Vista 3.

Figura 4.49: Identificación de daño en malla BG-T.

Capítulo 5

Conclusiones

Este trabajo de título enfrenta el desafío de identificar daños en piezas metálicas mediante el uso de tecnologías de escaneo 3D y algoritmos de geometría computacional. Se ha diseñado e implementado un sistema integral que integra algoritmos de alineación de mallas, técnicas de procesamiento de datos tridimensionales y operaciones booleanas, para detectar y caracterizar con precisión los daños en piezas metálicas. La implementación, llevada a cabo con Python y la librería open3D, ha demostrado ser efectiva para el manejo de datos tridimensionales, ofreciendo una plataforma sólida para el desarrollo y la escalabilidad futura del proyecto.

El sistema desarrollado demuestra una capacidad satisfactoria para identificar daños en diversas piezas metálicas. Se ha constatado que la eficacia en la detección de daños está directamente relacionada con la calidad de la alineación inicial de las mallas de triángulos y la resolución de los modelos 3D empleados. Las pruebas efectuadas muestran que, cuanto mejor es la alineación inicial entre el modelo de la pieza dañada y el modelo CAD original, más precisa y completa es la identificación del daño. Esta dependencia subraya la importancia crucial de una sólida alineación inicial y un meticuloso preprocesamiento de las mallas para el éxito del sistema de identificación de daños.

Sin embargo, la investigación también pone de manifiesto desafíos significativos, especialmente en lo referente a la alineación de mallas en piezas de geometrías complejas o simétricas, lo que en ocasiones conduce a soluciones subóptimas por la convergencia en mínimos locales. Además, el error cuantitativo en la identificación del volumen dañado es considerable (entre un 25 % y un 50 % de error), lo que indica un área importante para mejoras futuras.

Para investigaciones futuras, se sugiere continuar con la optimización del algoritmo de alineación y explorar técnicas avanzadas de aprendizaje automático que puedan incrementar la precisión y eficiencia del sistema. Ampliar la base de datos con más variedades de daños y geometrías de piezas podría contribuir a fortalecer la robustez del sistema ante una gama más amplia de situaciones.

En conclusión, este trabajo de título representa un avance significativo en la identificación de daños en piezas metálicas a través de tecnologías avanzadas. Aunque aún existen desafíos por resolver, las bases establecidas aquí marcan un camino prometedor hacia mejoras futuras y aplicaciones prácticas del sistema.

Bibliografía

- [1] Programa Nacional de Innovación en Manufactura Avanzada, “Proyecto 3: Sistema Robotizado de Recuperación de Piezas Metálicas mediante Manufactura Aditiva.” <https://www.programaima.cl/proyecto3/>, 2019. Accedido el 29 de junio de 2023.
- [2] Felix, D., “Experimental investigation on suspended sediment, hydro-abrasive erosion and efficiency reductions of coated pelton turbines,” *VAW-Mitteilungen*, vol. 238, 2017.
- [3] Frazier, W. E., “Metal additive manufacturing: a review,” *Journal of Materials Engineering and Performance*, vol. 23, no. 6, pp. 1917–1928, 2014, [doi:10.1007/s11665-014-0958-z](https://doi.org/10.1007/s11665-014-0958-z).
- [4] DebRoy, T., Wei, H. L., Zuback, J. S., Mukherjee, T., Elmer, J. W., Milewski, J. O., Beese, A. M., Wilson-Heid, A., De, A., y Zhang, W., “Additive manufacturing of metallic components—process, structure and properties,” *Progress in Materials Science*, vol. 92, pp. 112–224, 2018, [doi:10.1016/J.PMATSCI.2017.10.001](https://doi.org/10.1016/J.PMATSCI.2017.10.001).
- [5] Saboori, A., Aversa, A., Marchese, G., Biamino, S., Lombardi, M., y Fino, P., “Application of directed energy deposition-based additive manufacturing in repair,” *Applied Sciences*, vol. 9, no. 16, p. 3316, 2019, [doi:10.3390/APP9163316](https://doi.org/10.3390/APP9163316).
- [6] Tavares, P., Costa, C. M., Rocha, L., Malaca, P., Costa, P., Moreira, A., Sousa, A., y Veiga, G., “Collaborative welding system using bim for robotic reprogramming and spatial augmented reality,” *Automation in Construction*, vol. 104, pp. 102–115, 2019, [doi:10.1016/J.AUTCON.2019.04.020](https://doi.org/10.1016/J.AUTCON.2019.04.020).
- [7] Williams, F., “Point cloud utils,” 2022. <https://www.github.com/fwilliams/point-cloud-utils>.
- [8] Yuksel, C., “Sample elimination for generating poisson disk sample sets,” *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2015)*, vol. 34, no. 2, pp. 25–32, 2015, [doi:10.1111/cgf.12538](https://doi.org/10.1111/cgf.12538).
- [9] Hasenfratz, J.-M., Lapierre, M., Gascuel, J.-D., y Boyer, E., “Real-time capture, reconstruction and insertion into virtual world of human actors,” pp. 49–56, 2003.
- [10] Rusinkiewicz, S. y Levoy, M., “Efficient variants of the icp algorithm,” en *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pp. 145–152, 2001, [doi:10.1109/IM.2001.924423](https://doi.org/10.1109/IM.2001.924423).
- [11] Wan, T., Du, S., Xu, Y., Xu, G., Li, Z., Chen, B., y Gao, Y., “Rgb-d point cloud registration via infrared and color camera,” *Multimedia Tools and Applications*, vol. 78, 2019, [doi:10.1007/s11042-019-7159-6](https://doi.org/10.1007/s11042-019-7159-6).
- [12] Aoki, Y., Goforth, H., Srivatsan, R. A., y Lucey, S., “Pointnetlk: Robust efficient point cloud registration using pointnet,” 2019.

- [13] Qi, C. R., Su, H., Mo, K., y Guibas, L. J., “Pointnet: Deep learning on point sets for 3d classification and segmentation,” 2017.
- [14] Brauer, C. y Aukes, D., “Voxel-based cad framework for planning functionally graded and multi-step rapid fabrication processes,” 2019, [doi:10.1115/DETC2019-98103](https://doi.org/10.1115/DETC2019-98103).
- [15] MacQueen, J. *et al.*, “Some methods for classification and analysis of multivariate observations,” en Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.
- [16] Ester, M., Kriegel, H.-P., Sander, J., y Xu, X., “A density-based algorithm for discovering clusters in large spatial databases with noise,” en Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD’96, p. 226–231, AAAI Press, 1996.
- [17] Chen, Y., Hu, V. T., Gavves, E., Mensink, T., Mettes, P., Yang, P., y Snoek, C. G. M., “Pointmixup: Augmentation for point clouds,” 2020.

Anexo A

Código desarrollado

A.1. Manipulaciones geométricas

Código A.1: Funciones para manipulaciones geométricas

```
1 import numpy as np
2 import open3d as o3d
3
4 def translate_to_origin(pcd):
5     centroid = pcd.get_center()
6     translation_matrix = np.eye(4)
7     translation_matrix[0:3, 3] = -centroid
8     return pcd.transform(translation_matrix)
9
10
11 def get_transformation_to_origin(pcd):
12     centroid = np.asarray(pcd.points).mean(axis=0)
13     translation_matrix = np.eye(4)
14     translation_matrix[0:3, 3] = -centroid
15     return translation_matrix
16
17
18 def to_homogeneous(rotation_matrix, translation=np.array([0, 0, 0])):
19     transformation = np.eye(4)
20     transformation[:3, :3] = rotation_matrix
21     transformation[:3, 3] = translation
22     return transformation
23
24
25 def random_rotation():
26     axis = np.random.rand(3)
27     axis /= np.linalg.norm(axis)
28     angle = np.random.uniform(0, 2 * np.pi)
29     return o3d.geometry.get_rotation_matrix_from_axis_angle(axis * angle)
30
31
32 def xyz_spherical(xyz):
33     x = xyz[0]
```

```

34 y = xyz[1]
35 z = xyz[2]
36 r = np.sqrt(x * x + y * y + z * z)
37 r_x = np.arccos(y / r)
38 r_y = np.arctan2(z, x)
39 return [r, r_x, r_y]
40
41
42 def get_rotation_matrix(r_x, r_y):
43     rot_x = np.asarray(
44         [[1, 0, 0], [0, np.cos(r_x), -np.sin(r_x)], [0, np.sin(r_x), np.cos(r_x)]]
45     )
46     rot_y = np.asarray(
47         [[np.cos(r_y), 0, np.sin(r_y)], [0, 1, 0], [-np.sin(r_y), 0, np.cos(r_y)]]
48     )
49     return rot_y.dot(rot_x)
50
51
52 def get_extrinsic(xyz):
53     rvec = xyz_spherical(xyz)
54     r = get_rotation_matrix(rvec[1], rvec[2])
55     t = np.asarray([0, 0, 2]).transpose()
56     trans = np.eye(4)
57     trans[:3, :3] = r
58     trans[:3, 3] = t
59     return trans
60
61
62 def get_scale_and_center(model):
63     min_bound = model.get_min_bound()
64     max_bound = model.get_max_bound()
65     center = min_bound + (max_bound - min_bound) / 2.0
66     scale = np.linalg.norm(max_bound - min_bound) / 2.0
67     return scale, center
68
69
70 def preprocess_with_scale(model, scale):
71     min_bound = model.get_min_bound()
72     max_bound = model.get_max_bound()
73     center = min_bound + (max_bound - min_bound) / 2.0
74     vertices = np.asarray(model.vertices)
75     vertices -= center
76     model.vertices = o3d.utility.Vector3dVector(vertices / scale)
77     return model
78
79
80 def preprocess(model):
81     min_bound = model.get_min_bound()
82     max_bound = model.get_max_bound()
83     center = min_bound + (max_bound - min_bound) / 2.0
84     scale = np.linalg.norm(max_bound - min_bound) / 2.0
85     vertices = np.asarray(model.vertices)

```

```

86     vertices -= center
87     model.vertices = o3d.utility.Vector3dVector(vertices / scale)
88     return model
89
90 def add_cube_corners_to_pointcloud(point_cloud, cube_side=0.0078125):
91     half_side = cube_side / 2
92
93     offsets = np.array([
94         [half_side, half_side, half_side],
95         [-half_side, half_side, half_side],
96         [half_side, -half_side, half_side],
97         [-half_side, -half_side, half_side],
98         [half_side, half_side, -half_side],
99         [-half_side, half_side, -half_side],
100        [half_side, -half_side, -half_side],
101        [-half_side, -half_side, -half_side]
102    ])
103
104     original_points = np.asarray(point_cloud.points)
105
106     new_points = []
107
108     for point in original_points:
109         corners = point + offsets
110         new_points.extend(corners)
111
112     upsampled_point_cloud = o3d.geometry.PointCloud()
113     upsampled_point_cloud.points = o3d.utility.Vector3dVector(np.vstack((original_points,
114         ↪ new_points)))
114
115     return upsampled_point_cloud
116

```

A.2. Identificación de daño

Código A.2: Funciones para realizar la alineación de nubes de puntos y la conversión a voxels.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import os
4 import geometric_manipulations as gm
5 import open3d as o3d
6
7 def plot_and_save(pcd1, pcd2, filename=None, save_fig=False, show_fig=True):
8     fig = plt.figure(figsize=(10, 10))
9     ax = fig.add_subplot(111, projection="3d")
10
11     points1 = np.asarray(pcd1.points)
12     points2 = np.asarray(pcd2.points)
13
14     ax.scatter(
15         points1[:, 0],
16         points1[:, 1],
17         points1[:, 2],
18         c="r",
19         marker="o",
20         s=2,
21         alpha=0.5,
22         label="pcd1",
23     )
24     ax.scatter(
25         points2[:, 0],
26         points2[:, 1],
27         points2[:, 2],
28         c="b",
29         marker="o",
30         s=2,
31         alpha=0.5,
32         label="pcd2",
33     )
34
35     ax.set_xlabel("X")
36     ax.set_ylabel("Y")
37     ax.set_zlabel("Z")
38     ax.legend()
39
40     all_points = np.vstack([points1, points2])
41     max_limits = np.max(all_points, axis=0)
42     min_limits = np.min(all_points, axis=0)
43     max_range = np.max(max_limits - min_limits)
44
45     mid = 0.5 * (max_limits + min_limits)
46     ax.set_xlim(mid[0] - 0.5 * max_range, mid[0] + 0.5 * max_range)
```

```

47 ax.set_ylim(mid[1] - 0.5 * max_range, mid[1] + 0.5 * max_range)
48 ax.set_zlim(mid[2] - 0.5 * max_range, mid[2] + 0.5 * max_range)
49
50 plt.tight_layout()
51 if save_fig:
52     plt.savefig(filename)
53 if show_fig:
54     plt.show()
55 plt.close()
56
57
58 def visualize_icp_steps_save_image(
59     pcd1, pcd2, image_dir, num_trials=100, max_correspondence_distance=15000, show_fig
    ↪ =False, prefix=""
60 ):
61     lower_rmse = 1e1000
62     final_results = None
63
64     for _ in range(num_trials):
65         initial_transformation = gm.to_homogeneous(gm.random_rotation())
66         results = o3d.pipelines.registration.registration_icp(
67             pcd1,
68             pcd2,
69             max_correspondence_distance=max_correspondence_distance,
70             init=initial_transformation,
71             estimation_method=o3d.pipelines.registration.
    ↪ TransformationEstimationPointToPoint(),
72             criteria=o3d.pipelines.registration.ICPConvergenceCriteria(
73                 relative_fitness=0.00001, relative_rmse=0.000001, max_iteration=400
74             ),
75         )
76
77         if results.inlier_rmse < lower_rmse:
78             lower_rmse = results.inlier_rmse
79             transformation_matrix = results.transformation
80             final_results = results
81
82     transformation = transformation_matrix
83     pcd1_transformed = pcd1.transform(transformation)
84
85     image_path = None
86     save_fig = False
87     if image_dir is not None:
88         save_fig = True
89         image_path = os.path.join(image_dir, f"{prefix}icp_step_1.png")
90     print(lower_rmse)
91
92     plot_and_save(pcd1_transformed, pcd2, image_path, save_fig, show_fig=show_fig)
93
94     return final_results
95
96

```



```

97 def voxel_carving(mesh, cubic_size, voxel_resolution, w=500, h=500, base_name="gato"):
98     """
99     funcion basada en ejemplo de open3d
100     """
101     mesh.compute_vertex_normals()
102     camera_sphere = o3d.geometry.TriangleMesh.create_sphere(radius=1.0, resolution=10)
103
104     voxel_carving = o3d.geometry.VoxelGrid.create_dense(
105         width=cubic_size,
106         height=cubic_size,
107         depth=cubic_size,
108         voxel_size=cubic_size / voxel_resolution,
109         origin=[-cubic_size / 2.0, -cubic_size / 2.0, -cubic_size / 2.0],
110         color=[1.0, 0.7, 0.0],
111     )
112
113     camera_sphere = gm.preprocess(camera_sphere)
114
115     vis = o3d.visualization.Visualizer()
116     vis.create_window(width=w, height=h, visible=False)
117     vis.add_geometry(mesh)
118     vis.get_render_option().mesh_show_back_face = True
119     ctr = vis.get_view_control()
120     param = ctr.convert_to_pinhole_camera_parameters()
121     print("Start carving")
122     centers_pts = np.zeros((len(camera_sphere.vertices), 3))
123     for cid, xyz in enumerate(camera_sphere.vertices):
124         trans = gm.get_extrinsic(xyz)
125         param.extrinsic = trans
126         c = np.linalg.inv(trans).dot(np.asarray([0, 0, 0, 1]).transpose())
127         centers_pts[cid, :] = c[:3]
128         ctr.convert_from_pinhole_camera_parameters(param)
129
130     vis.poll_events()
131     vis.update_renderer()
132     depth = vis.capture_depth_float_buffer(False)
133
134     voxel_carving.carve_depth_map(o3d.geometry.Image(depth), param)
135     print("Carve view %03d/%03d" % (cid + 1, len(camera_sphere.vertices)))
136     vis.destroy_window()
137     print("End carving")
138
139     return voxel_carving
140

```