Algoritmo para la integración/registro de nubes de puntos tridimensionales basado en técnicas de aprendizaje profundo

Juan Ricardo Albarracín Barbosa y Ángel Fabián Gómez Estupiñán

Trabajo de Grado para optar por el título de Ingeniero de Sistemas e Informática

Director

Andrés Leonardo González Gómez, PhD(c)

Candidato a Doctor en Ciencias de la Computación

Codirector

Jaime Enrique Meneses Fonseca, PhD

Doctor en Ciencias de la Ingeniería

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2022

Dedicatoria

A Dios por ser mi guía y mi protector, por bendecirme tanto, por darme la fortaleza, las capacidades y habilidades para afrontar todos los retos y adversidades; por demostrarme su poder y su gloria en cada paso que doy en mi vida. A mis padres, Jacqueline y Juan Paulo, por su amor incondicional, por su infaltable compañía, por sus consejos, regaños y su incansable dedicación y trabajo para hacerme un hombre íntegro y ayudarme a cumplir mis sueños. A mi hermana Daniela y mis sobrinos Juan Antonio y Juan Pablo, por ser parte de mi alegría y motivación para salir adelante. A mi tía Lilia por su ayuda, apoyo y cariño hacia nuestra familia. A mis demás familiares, mi novia, mis amigos y compañeros que me han apoyado y acompañado a lo largo de diferentes etapas en mi vida y que de una u otra manera han confiado en mí y en lo que puedo dar.

"Un emprendedor no debe tener miedo al fracaso. Si se fracasa, hay que intentar de nuevo, y si se fracasa de nuevo, hay que intentar de nuevo, hasta triunfar… Para tener éxito, a veces tienes que sufrir fracasos en el camino, eso hace que el éxito sea más satisfactorio"

Sir Richard Branson

Juan Ricardo Albarracín Barbosa

Dedicatoria

Al Señor, con quien cada día me maravillo y trato de entender lo que en sus designios y voluntad tienen para mi vida y obra.

A Juan Ricardo, quien, a pesar de las veces en que pensamos en claudicar, fue pieza fundamental con sus esfuerzos y trabajo incansable en la culminación de este proyecto.

A mi familia, especialmente a mi madre, María Rosalba Estupiñan, quien me ha acogido y apoyado siempre con su amor incondicional. A mi hermano Héctor Mario Gómez, quien como hermano mayor ha estado en muchos momentos difíciles y ha compartido el pan de su mesa para que yo hoy pueda estar en donde estoy. A mis demás hermanos, a mi padre, a toda mi familia y amigos que con su cariño y apoyo siempre han estado presentes.

A todos los docentes que han impactado en mi vida y que con su trabajo siguen haciendo mella en la vida de otros estudiantes.

“La vida de un hombre es lo que sus pensamientos hacen de ella”

Fragmento del libro Meditaciones, Marco Aurelio

Ángel Fabian Gómez Estupiñan

Agradecimientos

A los profesores Andrés González y Jaime Meneses, por compartir con nosotros su conocimiento, su tiempo y su experiencia; factores que fueron fundamentales en la elaboración y desarrollo del proyecto. A Dahlia Urbach, de Israel, por su solidaridad, abnegación y disposición a colaborarnos en la comprensión de temas puntuales. A Jorge Luis Pérez Gonzáles, de México, por su contribución y suministro de ideas, las cuales fueron de gran utilidad. Al grupo de Super Computación y Cálculo Científico (SC3) de la UIS por facilitarnos los mecanismos y herramientas necesarias para llevar a cabo tareas de alto nivel de procesamiento computacional.

A nuestras familias y amigos, que fueron el apoyo en los momentos más difíciles, a través de las exigencias que este trabajo de investigación requirió y en medio de una crisis sanitaria que exigió de nosotros la reinvención para sacar el proyecto adelante.

En general, a todas las personas que de una u otra manera hicieron posible la realización de este proyecto.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Resumen

Título: Algoritmo para la Integración/Registro de Nubes de Puntos Tridimensionales Basado en Técnicas de Aprendizaje Profundo *

Autores: Juan Ricardo Albarracín Barbosa – Ángel Fabian Gómez Estupiñán **

Palabras Clave: Inteligencia artificial, reconstrucción tridimensional, metrología óptica, visión por computador, digitalización de objetos.

Descripción: Con el continuo desarrollo de las técnicas de reconstrucción tridimensional (3D) y el aumento de las aplicaciones en diferentes ramas de la ciencia en las que esta tecnología puede tener cabida, han surgido multitud de metodologías que intentan solucionar los problemas de registrar y alinear nubes de puntos ubicadas en posiciones aleatorias o cuando no tienen una estimación de la transformación que los diferencia. En este proceso de reconstrucción 3D, después de escanear u obtener datos desde diferentes puntos de vista, uno de los principales desafíos es establecer los puntos correspondientes entre las diferentes capturas para alinear correctamente ese par de nubes de puntos (integración o registro) en base a un mismo sistema de referencia. Es necesario ajustar y alinear las nubes de puntos correspondientes a los escaneos del objeto o escena en cuestión para poder realizar la reconstrucción o digitalización para cual sea su aplicación. En este trabajo de investigación se propone un algoritmo que involucra técnicas de Aprendizaje Profundo para obtener un correcto registro de nubes de puntos capturadas en ambientes de oclusión, alta reflectancia y datos faltantes.

---

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática.
Director: Andrés Leonardo González Gómez, Candidato a Doctor en Ciencias de la Computación
Codirector: Jaime Enrique Meneses Fonseca, PhD en Ciencias de la Ingeniería.

Abstract

Title: Algorithm for the Integration/Registration of Three-Dimensional Point Clouds Based on Deep Learning Techniques *

Author: Juan Ricardo Albarracín Barbosa – Ángel Fabian Gómez Estupiñán **

Keywords: Artificial intelligence, three-dimensional reconstruction, optical metrology, computer vision, objects digitization.

Description: With the continuous development of three-dimensional (3D) reconstruction techniques and the increase in the applications in which this technology can have a place, many methodologies have emerged that try to solve the problem of registering and aligning point clouds located in random positions or when they do not have an estimate of the transformation that differentiates them. In this three-dimensional reconstruction process, after scanning or obtaining data from different points of view, one of the main challenges is establishing the corresponding points between the different captures to correctly align that pair of point clouds (integration or registration) based on the same reference system. Is necessary adjust and align point clouds corresponding to scans of the object or scene in question to perform the reconstruction or digitization for whatever application. In this research work, an algorithm that involves Deep Learning techniques is proposed to obtain a correct registration of point clouds captured in occlusion environments, high reflectance, and missing data.

---

* Bachelor Thesis

** Departament of Physical – Mechanical Engineering, School of Systems and Informatics Engineering.
Advisor: Andrés Leonardo González Gómez, PhD candidate in Computer Science.
Co-advisor: Jaime Enrique Meneses Fonseca, PhD in Engineering Sciences.

Introducción

En las últimas décadas, las técnicas de digitalización tridimensional (3D) han mejorado considerablemente, por consiguiente, han surgido algoritmos para llevar a cabo el proceso de digitalización o reconstrucción de objetos y escenas, para su posterior aplicación en ramas como la medicina, ingeniería, diseño gráfico, mecánica, arqueología, la industria del ocio o entretenimiento, entre otras. Esta tecnología permite a sus usuarios acercarse a una "realidad", disponiendo de modelos 3D digitales en un computador, con una enorme resolución y fidelidad, parámetros que cada vez tendrán mayor precisión. De este modo, la digitalización 3D se convierte en una herramienta muy útil, con la ventaja de que hoy en día los computadores estándar pueden visualizar perfectamente objetos 3D, lo que permite mayor accesibilidad a esta tecnología y la eclosión de nuevas aplicaciones.

La digitalización 3D es la generación de un modelo informático de un objeto o una escena, proceso en el cual, independientemente de la tecnología usada, se involucra una serie de etapas procedimentales en las que se incluye la toma de datos del objeto (Torres et al., 2010). Estos datos proveen una gran cantidad de información espacial, denominada *nube de puntos*, que debe ser interpretada y procesada para determinar ciertos detalles del objeto o escena (Salcedo et al., 2012). En este proceso de reconstrucción 3D, luego de realizar el escaneo de datos desde diferentes puntos de vista, uno de los principales desafíos radica en alinear adecuadamente ese par de nubes de puntos en base a un mismo sistema de referencia, a partir del hallazgo de los puntos correspondientes entre las diferentes capturas.

El propósito de este trabajo de investigación es proponer un algoritmo que efectúe la mejor alineación y registro de nubes de puntos captados en entornos de oclusión, reflectancia y datos faltantes entre los puntos a alinear. En primera instancia se determinarán los parámetros

más importantes que influyen en el proceso de integración. Una vez determinados dichos parámetros y sus valores, mediante el uso de herramientas de Inteligencia Artificial como las Redes Neuronales, se procederá a establecer coincidencias de puntos y sus correspondientes pares, con el fin de realizar el proceso de alineamiento, contando con el mínimo error, y la mayor precisión y eficiencia computacional posible durante el proceso.

Acerca del trabajo de investigación

1.1. Planteamiento del problema

Un sistema de reconstrucción 3D está formado por hardware y software. Con el hardware se realiza la etapa de registro, es decir, se adquieren los datos (nubes de puntos) usando técnicas de contacto como los brazos robotizados, o de no contacto como los métodos ópticos, magnéticos o acústicos. La eficiencia en las técnicas utilizadas infiere en el nivel de detalle o realismo de la representación digital final del objeto o escena, que es un factor realmente importante en el proceso.

Una vez adquiridas las nubes de puntos, se realiza la etapa de integración o registro. Los datos obtenidos deben ser alineados y procesados para transformarlos en información útil. Este procedimiento implica problemas como el ruido, oclusiones, distorsiones y datos faltantes o no válidos, ocasionados por el hardware o el ambiente en el cual se obtuvieron las imágenes (Malamas et al., 2013). Además, el campo de visión del dispositivo con el que se realizó el escaneo es un factor limitante, pues la información adquirida será únicamente la de las superficies que se encuentren dentro de dicho campo de visión, por tal motivo se requieren múltiples vistas para contar con la mayor información posible del objeto a reconstruir (Turk & Levoy, 1994; Curless & Levoy, 1996).

Existen procedimientos en los que se requiere calibrar más de un sistema de digitalización 3D, cada uno con transformaciones rígidas independientes, para luego llevar cada uno de estos sistemas a un mismo sistema coordenado para la realización de la integración de nubes basada en el registro rígido. Este proceso, además de contar con una complejidad considerable, requiere de tiempo de práctica y calibración.

A partir de lo anterior, surge la necesidad de ajustar y alinear nubes de puntos correspondientes a escaneos del objeto o escena en cuestión, tomados desde diferentes puntos de vista, donde cada vista tendrá su propio sistema de coordenadas (sistema de coordenadas con el que fueron escaneadas). En este proceso, el problema radica en encontrar la transformación rígida (rotación, translación) que alinee adecuadamente los puntos correspondientes expresados en un sistema coordenado de referencia sin afectar la morfología de la superficie a reconstruir (Maiseli et al., 2017). No obstante, las distorsiones, la alta reflectancia de los objetos y oclusiones, así como el ruido o datos faltantes o inconsistentes, hacen de este problema todo un reto.

Este proyecto de investigación busca realizar el cálculo de las transformaciones rígidas para cada uno de los sistemas coordenados de las vistas del objeto obtenidas para su digitalización, además de cambiar la representación de los puntos en otro tipo de representación, para que mediante técnicas de aprendizaje profundo se pueda determinar los puntos de "*matching"* entre los grupos de nubes que permitan su correcta integración o registro.

1.2. Justificación

Con el constante desarrollo de las técnicas de reconstrucción 3D y el aumento de las aplicaciones en las cuales esta tecnología puede tener cabida, han surgido metodologías que intentan solucionar el problema de alineamiento de nubes de puntos situadas en posiciones aleatorias y cuando no se tiene ninguna estimación de la transformación que las diferencia. La mayoría de estos algoritmos se basan en buscar en el espacio los parámetros de la transformación o en buscar en el espacio todas las correspondencias posibles entre los puntos de las nubes (Torre, 2011). Los principales inconvenientes de los algoritmos son la poca velocidad y la no convergencia en ciertos casos, lo que se traduce en un alto coste computacional.

Considerando que la correspondencia e integración de nubes de puntos de forma óptima y precisa es una etapa fundamental en el proceso de reconstrucción 3D para obtener resultados de calidad, aun cuando las nubes de puntos presentan valores atípicos, se plantea la siguiente pregunta de investigación: ¿Cómo optimizar y mejorar en términos de precisión la integración o registro de nubes de puntos 3D para la digitalización 3D de objetos?

Hipótesis: es factible que mediante el uso de técnicas de Inteligencia Artificial se pueda establecer la correspondencia de puntos entre varias nubes. Plantear un algoritmo que de forma geométrica realice el *matching* de puntos para su posterior alineado y reconstrucción 3D puede apoyar la optimización del proceso y aumentar la precisión para conseguir digitalizaciones con alto grado de calidad. Así, se pretende que, en entornos de ruido, oclusión o en objetos con un índice de reflectancia elevado, se facilite el proceso de digitalización, y en general que esta tecnología sea métricamente más precisa y computacionalmente de bajo costo para que pueda ser adoptada en muchas más áreas de aplicación.

Mediante el desarrollo de este trabajo de investigación se hace un aporte al campo de la óptica y visión por computador enfocado específicamente en la digitalización 3D de objetos, empleando metodologías basadas en Inteligencia Artificial.

1.3. Objetivos

1.3.1. Objetivo general

Mediante el uso de técnicas de Aprendizaje Profundo y arquitecturas existentes en el estado del arte, proponer un algoritmo que permita una óptima integración de nubes de puntos tridimensionales en el proceso de digitalización de objetos con alta reflectancia.

1.3.2. Objetivos específicos

- Determinar los parámetros más relevantes que influyen en el proceso de selección, correspondencia y alineación de nubes de puntos.
- Estudiar cómo funcionan los algoritmos actuales para realizar procesos de digitalización tridimensional, específicamente de integración de nubes de puntos.
- Usar un lenguaje de programación de código abierto (*open source)* que incorpore las librerías necesarias para modelar el algoritmo que permita la selección y correspondencia de puntos entre nubes.
- Validar, presentar y comparar los resultados obtenidos frente a otros algoritmos que existen en el estado del arte.

1.4. Estructura del documento

En el marco teórico y estado del arte (capítulo 2 y 3) se presentan algunos conceptos claves e instrumentos necesarios para el desarrollo y fundamentación del proyecto, los cuales están relacionados a la reconstrucción tridimensional, digitalización de objetos, aprendizaje automático y visión por computador.

Seguidamente, en el capítulo 4 se presentan las aproximaciones y avances realizados en temas de registro de nubes de puntos mediante toda la investigación realizada. Además, se expone el algoritmo propuesto para la realización de los objetivos planteados. Consecutivo a esta sección, se define el apartado de resultados (capítulo 5) donde se describe a detalle la precisión del algoritmo en base al tiempo de ejecución y a la cantidad de iteraciones, y además se compara con otras metodologías usadas para tareas de registro de nubes de puntos.

Por último, se plantean las conclusiones (capítulo 6) y recomendaciones o trabajo a futuro (capítulo 7) de acuerdo con la exhausta investigación y experimentación realizada.

Marco teórico

A continuación, se definen los términos más importantes para el desarrollo del proyecto, empezando desde lo fundamental, los modelos tridimensionales, hasta aspectos relacionados con las redes neuronales profundas. Una vez definido el cuerpo teórico de la investigación, se establecen las variables de interés y los instrumentos para la investigación.

2.1. Modelo tridimensional

Un modelo 3D es una representación digital de un objeto o escena real, es decir, una representación numérica. Para lograr tal representación, es necesario emplear estructuras que definan de manera discreta a los cuerpos estudiados. Esta discretización se logra mediante el uso en conjunto de hardware y software para la captura de datos geométricos, que serán la base de la digitalización 3D del objeto u escena, a este grupo de datos se les conoce como *nubes de puntos 3D*. Los modelos 3D son de gran utilidad, especialmente para dimensionar y analizar elementos de estudio, como es el caso de prótesis médicas (ver figura 1), que requieren de una gran exactitud respecto a sus medidas, o de localizaciones y estructuras, como es el caso de construcciones históricas o la topografía de un terreno.

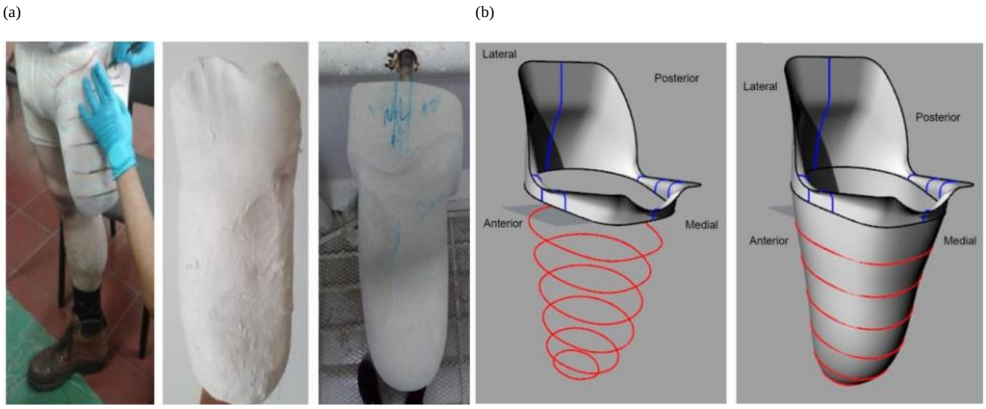

Figura 1. Modelo de prótesis para una pierna para su posterior digitalización, (a) Proceso de desarrollo de socket tipo cuadrilateral, (b) Modelado virtual 3D de socket. Adaptado de Gualdron et al., 2019.

La generación o edición de estos modelos 3D, basándose ya sea en un objeto digitalizado previamente o con medidas del objeto real, da lugar a aplicaciones como la creación de prototipos a bajo costo, como son los modelos de vehículos o aviones creados desde cero o que posteriormente sufren modificaciones respecto al original o variaciones creadas a partir del modelo tridimensional originalmente captado.

Otra de las aplicaciones del modelado 3D es la conducción autónoma, tecnología que hoy en día está teniendo gran auge y de la cual se han visto grandes avances. Un ejemplo es la tecnología LIDAR (Light Detection And Ranging) usada en vehículos autónomos como "Waymo" desarrollado por Google, el cual obtiene nubes de puntos de la carretera para detectar obstáculos y reconstruir la escena tridimensional, lanzando y recibiendo pulsos mediante un láser (Royo & Ballesta, 2019).

La fidelidad del modelo con respecto al objeto real varía de acuerdo con muchos aspectos, entre estos encontramos el objeto en sí mismo, el entorno en el que se realiza la obtención del modelo, los elementos y la técnica de obtención del modelo 3D, etc.

2.2. Nubes de puntos

Las nubes de puntos son un conjunto de datos que describen un objeto digitalizado en términos de coordenadas "x", "y" y "z" del espacio, estructurados en forma de matrices que proveen de información 3D acerca de la superficie del objeto (Bronstein et al., 2008) como se muestra en la ecuación (1):

$$P = \{p_1, p_2, \dots, p_n\}, \qquad | \, p_i \in R^3 \, , \tag{1}$$

donde P es el conjunto de puntos que compone la nube, p_i es un punto que pertenece a la nube y n es la cantidad de puntos en la nube. En su mayoría estos puntos no se encuentran

ordenados de forma consecutiva, por lo que hallar la correspondencia de puntos entre dos nubes con áreas similares no es tarea sencilla (Ben-Shabat et al., 2017).

El contenido de las nubes es producto de la interacción entre un sistema de obtención de datos superficiales de un objeto y una escena de interés. Al producirse esta interacción, el sistema de adquisición de datos puede traducir miles de datos obtenidos, establecidos como puntos, en el modelo digital de la superficie del objeto de estudio (ver figura 2).

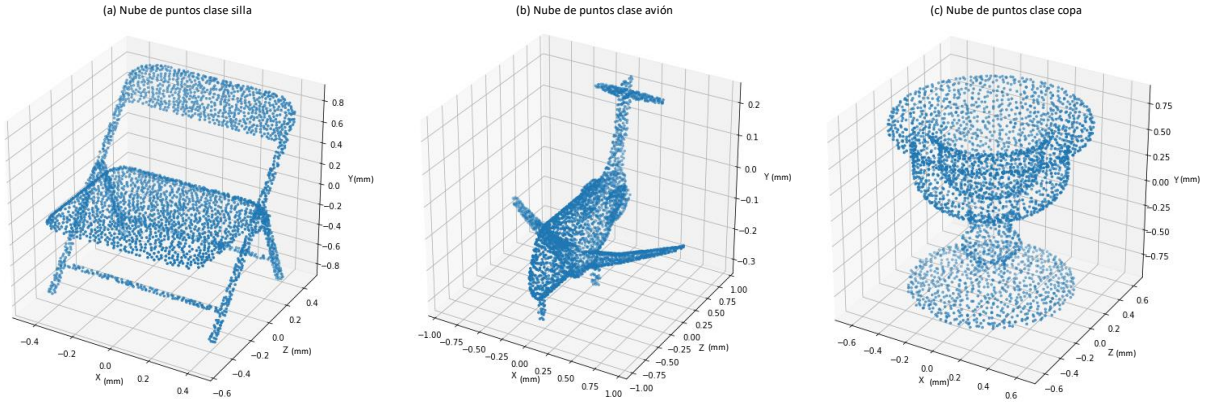

Figura 2. Ejemplos de nubes de puntos de objetos digitalizados, (a) modelo silla, (b) modelo avión, y (c) modelo copa.

2.3. Proceso de digitalización 3D

Este proceso parte del uso de un instrumento (hardware y software) con la capacidad de muestrear las coordenadas de la superficie del objeto a digitalizar. Este muestreo se puede realizar mediante diferentes técnicas basadas en mecanismos o fenómenos físicos que permita interactuar al sistema con dicha superficie.

Tras este muestreo se obtienen datos discretos de la superficie del objeto, la interpretación de estos datos que permitirá la obtención de un modelo digitalizado del objeto de estudio se compone de las siguientes fases:

2.3.1. Filtrado de datos

Tras poner en el proceso de muestreo de la superficie de la escena de interés se obtienen los primeros datos captados por el sistema de medición. Estos datos pueden contener ruido, distorsiones o datos no válidos, los cuales son ocasionados por el hardware usado en el procedimiento o por el ambiente. Por lo tanto, los datos adquiridos deben ser filtrados para corregir perturbaciones, datos no deseados o erróneos (Malamas et al., 2019).

Este proceso de filtrado puede bien ser realizado en la parte experimental de la captura de la nube de puntos, controlando parámetros como la oclusión, la reflectancia del objeto, una buena calibración del sistema de adquisición, o puede ser realizado posteriormente mediante técnicas computacionales.

2.3.2. Registro o Integración de Nubes de Puntos

Un dispositivo de reconstrucción 3D sólo puede adquirir información de la superficie del objeto que se encuentra dentro del campo de visión del captor. Por lo tanto, se necesitan múltiples vistas para adquirir datos de toda la superficie. El registro se usa para determinar la transformación de datos obtenidos a partir de vistas diferentes, de modo que éstos se puedan integrar bajo el mismo sistema de coordenadas. Usando la nube de puntos o los datos volumétricos adquiridos es posible construir una aproximación digital de la superficie de estudio (Turk & Levoy, 1994; Hope et al., 1992).

En el proceso de reconstrucción 3D, el registro de nubes de puntos es una tarea fundamental que nace de la necesidad de alinear muestras de puntos captadas de diferentes vistas y que se ha intentado sobrellevar con diferentes algoritmos propuestos.

Sean $A \in \mathbb{R}^3$ y $B \in \mathbb{R}^3$, dos nubes de puntos a registrar, donde A se considera un conjunto de puntos fijo y B un conjunto de puntos que ha sufrido una transformación

desconocida. El registro se basa básicamente en encontrar la transformación rígida T que alinee geométricamente el par de conjuntos de puntos A y B en un espacio de referencia común, de tal manera que A se pueda definir a partir de la ecuación (2):

$$A = \mathrm{T}\{R \cdot B + t\}, \quad (2)$$

donde $R \in \mathbb{R}^3$ y $t \in \mathbb{R}^3$ son los parámetros de rotación y traslación respectivamente.

El problema entonces se remonta a encontrar dichos parámetros R y t que permitan realizar tal transformación geométrica que alinee los pares de puntos. Entre los diferentes métodos que hacen registro de nubes de puntos, cuya idea principal y muy importante es determinar, en primera instancia, la correspondencia de puntos, para luego definir la relación que existe entre ellas (Zitová & Flusser, 2003), el Iterative Closest Point (ICP) es uno de los algoritmos más populares. Se basa en la cercanía de puntos para determinar dicha correspondencia y de una forma iterativa minimizar el error cuadrático medio entre sus distancias (Besl & Mckay, 1992).

El registro o integración de Nubes de Puntos se puede realizar en dos pasos: Correspondencia y Alineamiento de puntos (ver figura 3):

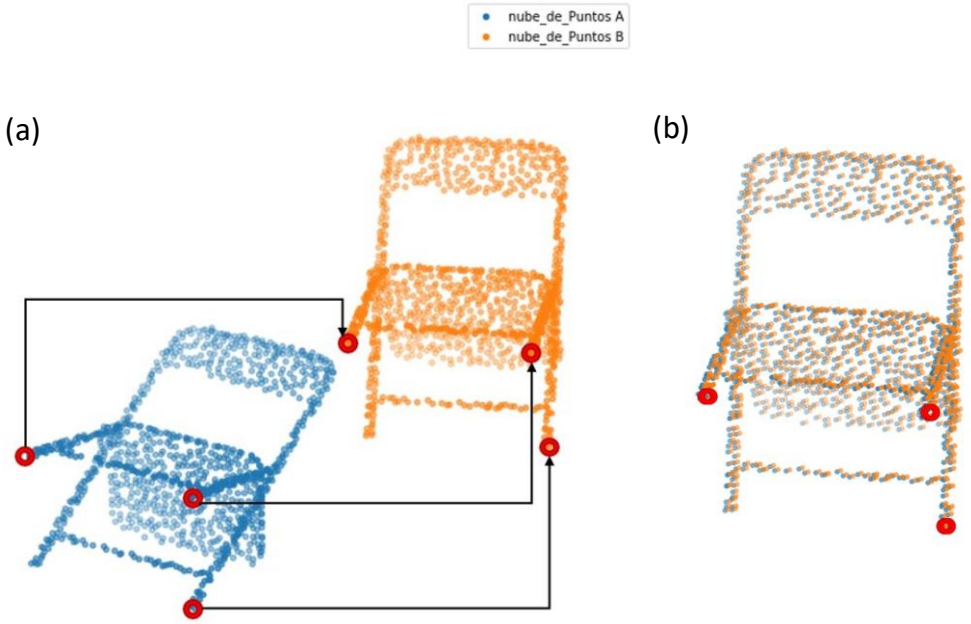

Figura 3. (a) Correspondencia y (b) Registro de dos nubes de puntos A y B.

2.3.2.1. Correspondencia de puntos. La correspondencia consiste en emparejar los puntos de las muestras de dos conjuntos de nubes que pretenden ser superpuestas, para ello se analizan y comparan características de los puntos de esta. Dependiendo del tipo de característica, podemos utilizar diferentes métodos para encontrarla correspondencia (Pérez y Gómez, 2016).

2.3.2.2. Alineamiento de nubes de puntos. Dado que la mayoría de los escáneres tienen un campo de visión limitado y que los objetos no son inmunes a las auto oclusiones, será necesario realizar varios escaneos del mismo objeto desde diferentes puntos de visión para poder obtener de él un conjunto de muestras suficiente con el que poder construir un modelo 3D que lo represente de manera adecuada (Curless & Levoy, 1996).

Cada vista del objeto estará referida al sistema de coordenadas del escáner, por lo que será necesario obtener la transformación que exprese sus puntos en el sistema de coordenadas de referencia. De esta manera será posible construir el modelo completo del objeto escaneado (Torre, 2011).

Posterior al registro, adicionalmente coexisten otras etapas para culminar con el proceso de reconstrucción, como son la simplificación, suavizado y mallado de datos.

2.4. Distancia Euclidiana

La distancia euclidiana $d(\vec{a}_1, \vec{a}_2)$ entre dos puntos $\vec{a}_1 = (x_1, y_1, z_1)$ y $\vec{a}_2 = (x_2, y_2, z_2)$ está definida por la ecuación (3):

$$d(\vec{a}_1, \vec{a}_2) = \|\vec{a}_1 - \vec{a}_2\| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}\,. \quad (3)$$

2.5. Norma de un vector

Sea $\vec{v} = (x, y, z)$ un vector en $\mathbb{R}^3$, la norma $\|\vec{v}\|$ de $\vec{v}$ está dada por la ecuación (4):

$$\|\vec{v}\| = \sqrt{x^2 + y^2 + z^2}\,, \quad (4)$$

$$\|\vec{v}\| = 0 \Leftrightarrow \vec{v} = 0\,.$$

2.6. Descomposición de Valores Singulares SVD

Teorema: Toda Matriz $A_{n\times m}$ puede expresarse a partir de la ecuación (5):

$$A = U_{n\times n} \cdot S_{n\times m} \cdot V^T_{m\times m}\,, \quad (5)$$

en donde $U_{n\times n}$, $V_{m\times m}$ son matrices ortogonales, si se cumple la igualdad descrita en las ecuaciones (6) y (7):

$$U^T U = I_{n\times n}\,, \quad (6)$$

$$V^T V = I_{m\times m}\,, \quad (7)$$

y además S es una matriz con r número de elementos positivos en la diagonal, siendo r el rango de la matriz, y el resto cero.

Los elementos de la diagonal de S se denominan valores singulares de A y las columnas de U y de V se denominar direcciones principales de salida y, de entrada, respectivamente. Representación gráfica de esta descomposición (ver figura 4).

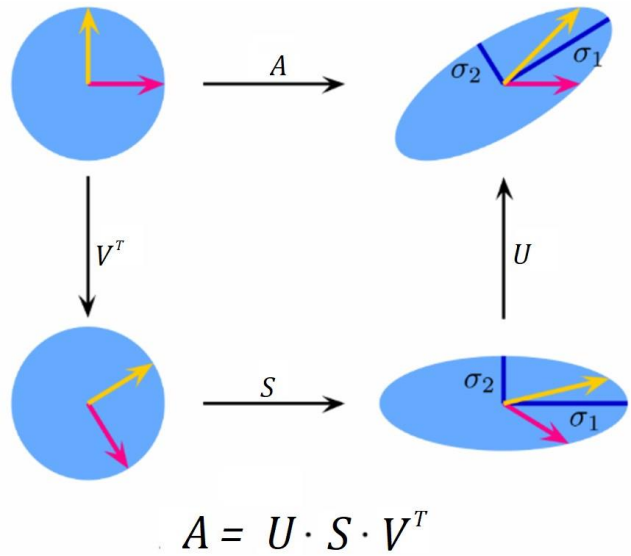

Figura 4. Descomposición de valores singulares. Adaptado de Johann, 2010.

2.7. Regla de la cadena

La regla de la cadena es una fórmula que permite el cálculo de la derivada de una función compuesta $f(g(x))$ a partir de las derivadas de las funciones $f(x)$ y $g(x)$ (Weisstein, s.f.).

Sea $g(x)$ una función diferenciable en el punto x y $f(x)$ una función diferenciable en el punto $g(x)$, entonces, la función compuesta $f(g(x))$ es diferenciable en el punto x.

Formalmente, la regla de la cadena se define mediante la ecuación (8):

Si $F(x) = f(g(x))$, entonces la derivada $F'(x) = f'\big(g(x)\big) \cdot g'(x)$. (8)

2.8. Vector Gradiente

El vector gradiente de una función f evaluado en un punto x de su dominio, $\nabla f(x)$, indica la dirección en la que la función f cambia de forma más rápida y el módulo de este vector representa el ritmo de ese cambio en la dirección descrita por el gradiente (ver figura 5). También representa la pendiente de la recta tangente a la curva de nivel que se produce entre la superficie de f y un plano perpendicular a dicha superficie (Zill y Wright , 2011, p. 719).

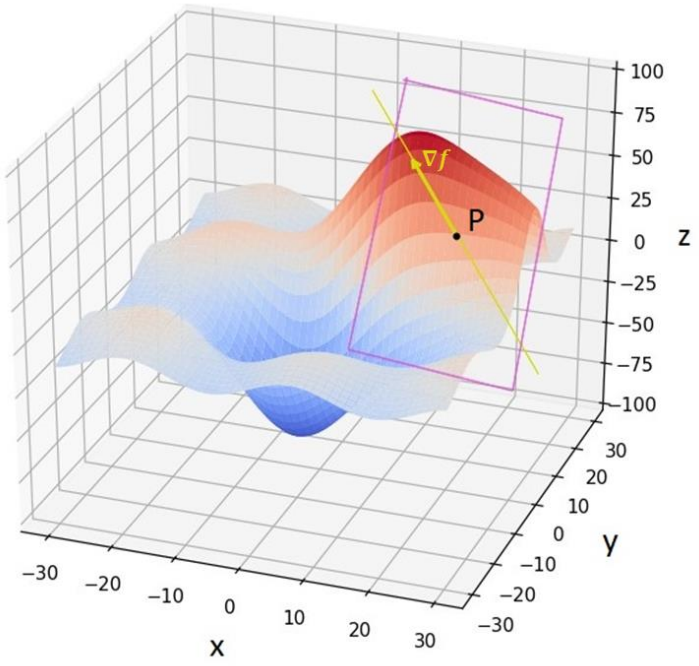

Figura 5. Vector gradiente de una superficie f evaluado en un punto P. Adaptado de Barrueco, s.f.

Formalmente, si la función $f: R^n \rightarrow R$ es parcialmente derivable en a (a pertenece al dominio de f), el gradiente de la función f en a es el vector $\nabla f(a) \in R^n$, sus componentes están dadas por la ecuación (9):

$$\nabla f(a) = \left(\frac{\partial f(a)}{\partial x_1}, \frac{\partial f(a)}{\partial x_2}, \ldots, \frac{\partial f(a)}{\partial x_n}\right) = \sum_{j=1}^{N} \frac{\partial f(a)}{\partial x_j} \cdot e_i \,, \tag{9}$$

de modo que la *j*-ésima coordenada del vector gradiente de f en a es la derivada parcial con respecto a la *j*-ésima variable de f en a, para todo $j = 1, \ldots, N$. Los vectores unitarios correspondientes a cada componente del gradiente están representados por e_i en la ecuación (10) (Payá, 2015).

Mediante el gradiente se puede expresar la derivada direccional en el sentido de un vector unitario u, pues representa la proyección del vector gradiente en el vector unitario u (Stewart, 2012, p. 936).

$$D_u f(x, y) = \nabla f(x, y) \cdot u \,. \tag{10}$$

Algunas propiedades asociadas al vector gradiente (∇f) son:

- El valor máximo de la derivada direccional $D_u f$ es $|\nabla f|$, es decir, la norma del vector gradiente (Stewart, 2012, p. 939).
- El valor máximo de la derivada direccional $D_u f$ se presenta cuando u tiene la misma dirección que el vector gradiente ∇f (Stewart, 2012, p. 939), mientras que en dirección del vector gradiente negativo $(-\nabla f)$ se presenta la derivada direccional mínima (Zill y Wright, 2011, p.722).
- En los puntos estacionarios de la función f (máximos, mínimos y puntos de inflexión) el gradiente se hace nulo debido a que, siempre y cuando las derivadas parciales de f existan, estas son cero (Stewart, 2012, p. 947).

Debido a esta última propiedad se hace útil el uso del gradiente en problemas de optimización, como es el caso de la función de pérdida para una red neuronal.

2.7. Modelo de Mezcla de Gaussianas (GMM)

En el trabajo de Reynolds (Reynolds, 2019), el autor menciona que un modelo GMM es una función de densidad de probabilidad paramétrica representada como una suma ponderada de las densidades de los componentes gaussianos. Comúnmente se usa como un modelo paramétrico de la distribución de probabilidad de mediciones continuas. Los parámetros de este modelo son estimados de los datos de entrenamiento usando el algoritmo iterativo *Expectation Maximization* (EM). La suma ponderada de las M densidades gaussianas está dada por la ecuación (11):

$$p(x|\lambda) = \sum_{i=1}^{M} w_i \, g(x|\mu_i, \Sigma_i) \,, \tag{11}$$

dónde x es un vector de datos de valor continuo D dimensional, w_i con $i = 1, \dots, M$ son los pesos de la mezcla de gaussianas, y $g(x|\mu_i, \Sigma_i)$ con $i = 1, \dots, M$ son las densidades gaussianas de los componentes. Cada densidad de componentes es una función gaussiana "*D*-variante" de la forma, expresada mediante la ecuación (12):

$$g(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu_i)' \, \Sigma_i^{-1} \, (x - \mu_i)\right\}. \tag{12}$$

Con un vector medio μ_i, y matriz de covarianza Σ_i, los pesos de la mezcla satisfacen la condición descrita por la ecuación (13):

$$\sum_{i=1}^{M} w_i = 1 \,. \tag{13}$$

El GMM completo está parametrizado por los vectores de media, matrices de covarianza y pesos de la mezcla de todas las densidades de los componentes. Estos parámetros están colectivamente representados por el conjunto dado por la ecuación (14):

$$\lambda = \{w_i,\ \mu_i,\ \Sigma_i\}\ \ con\ \ i = 1, \dots, M\ . \qquad (14)$$

Uno de los atributos poderosos de GMM es su capacidad de formar aproximaciones suavizadas a densidades formadas arbitrariamente. Actúa como un modelo híbrido entre el clásico modelo gaussiano unimodal y un modelo cuantificador vectorial (VQ) al usar un conjunto discreto de funciones gaussianas, cada una con su propia media y matriz de covarianza, para permitir una mejor capacidad de modelamiento. En la figura 6 se comparan las densidades obtenidas usando distintos modelos gaussianos.

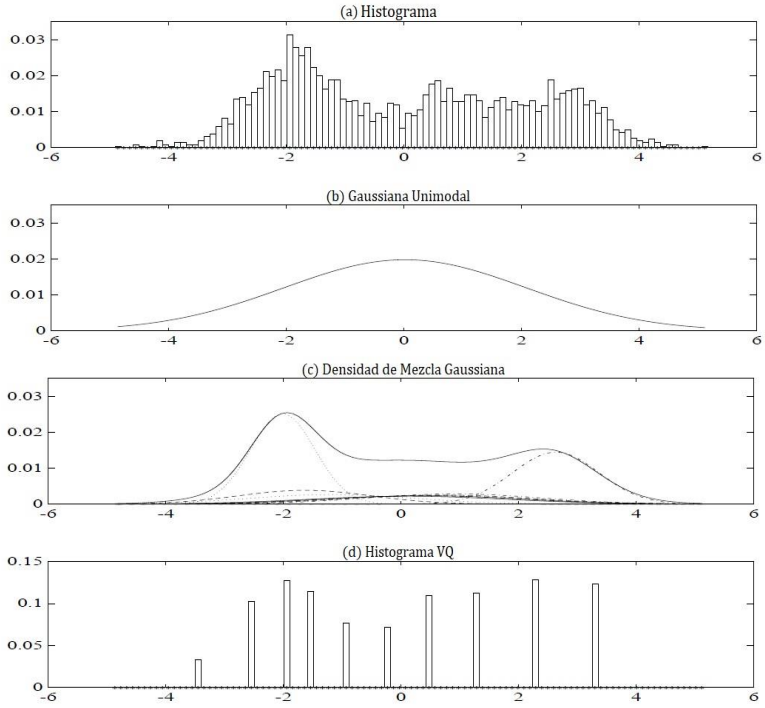

Figura 6. Comparaciones de densidades en distintos modelos, (a) a partir de un histograma, (b) unimodal, (c) un GMM y (d) un modelo VQ. Adaptado de Reynolds, 2009.

2.8. Aprendizaje Profundo

Deep Learning o Aprendizaje Profundo, es una estrategia de aprendizaje automático que permite a dispositivos electrónicos con la capacidad suficiente, aprender de la experiencia, generalizar y automatizar procesos. Debido a su naturaleza, no es necesario que un operador humano especifique formalmente todo el conocimiento que necesita la computadora (Goodfellow et al., 2016). La forma en que un algoritmo de esta índole logra aprender conceptos complicados parte de la construcción de "conocimiento" a partir de conceptos simples.

Actualmente, los modelos de aprendizaje profundo han obtenido grandes logros en diferentes áreas como la biomédica, la visión por computador y el entretenimiento, y con el aprovechamiento de los grandes conjuntos de datos generados constantemente, este tipo de algoritmos tienen la capacidad mejorada para realizar predicciones, reconocimientos y detecciones con un alto nivel de precisión, sin ambigüedad y confiabilidad.

2.8.1. Redes Neuronales

Inspiradas en sistemas biológicos como las conexiones neuronales en el cerebro, la concepción de este tipo de modelos está compuesta de unidades neuronales artificiales conectadas, capaces de crear relaciones entre datos y extraer de ellos representaciones de alto nivel para cual sea su aplicación (Clasificación o Regresión) (Hagan et al., 2014). A estas conexiones neuronales apiladas en múltiples capas comúnmente se les denomina Redes Neuronales Profundas (Deep Neural Networks - DNN) (Kim, 2017).

2.8.1.1. Arquitectura de una Red Neuronal. Las DNNs tienen una estructura compuesta por neuronas, capa de entrada, capas ocultas, funciones de activación y capa de salida. Cada capa representa un nivel de información o procesamiento (ver figura 7).

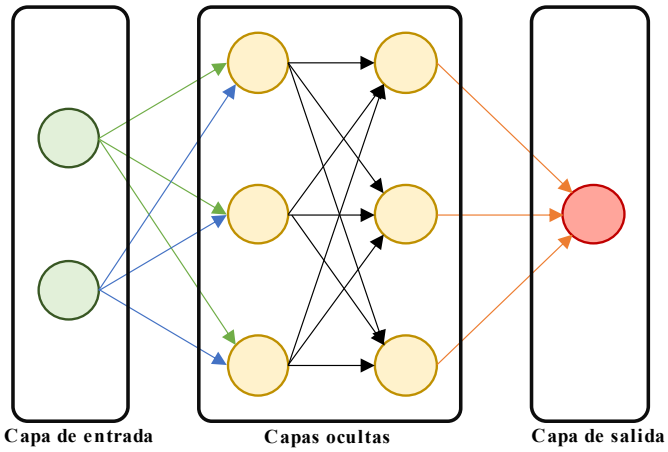

Figura 7. Estructura por capas de una red neuronal profunda. Adaptado de Nielsen, 2015.

Capa de entrada. Contiene información nueva y está compuesta por neuronas que reciben los datos de entrada. En general, son unidades pasivas ya que no modifican el valor de los datos o señal recibida. El objetivo principal de esta capa es la presentación y distribución de la información recibida a las siguientes capas (Aguilar, 1999).

Capas ocultas. Son las capas intermedias entre la capa de entrada y la capa de salida, y contienen las correlaciones internas entre las unidades de entrada y las capas anteriores (Nielsen, 2015).

Capa de Salida. Proporciona el resultado de la correlación de todas las neuronas a lo largo de las capas de la red. En otras palabras, es la respuesta de la red neuronal. La capa de salida define el problema a tratar. Para problemas de regresión, basta con una sola neurona en dicha capa, y para problemas de clasificación, usualmente se define una neurona por cada clase y una función de activación (Bishop, 1995).

Neuronas. Un perceptrón o neurona es la unidad básica de una red neuronal capaz de codificar una relación entre datos de capas anteriores (Kubat, 1999). Su forma estándar consiste en (ver figura 8):

- Un conjunto de entradas X_j.

- Un conjunto de pesos sinápticos w_j, y un bias o sesgo b asociados a las entradas.
- Un módulo de cómputo de la salida con respecto a las entradas, los pesos y el bias.
- Una función de activación σ que representa la salida de la neurona de forma no lineal.

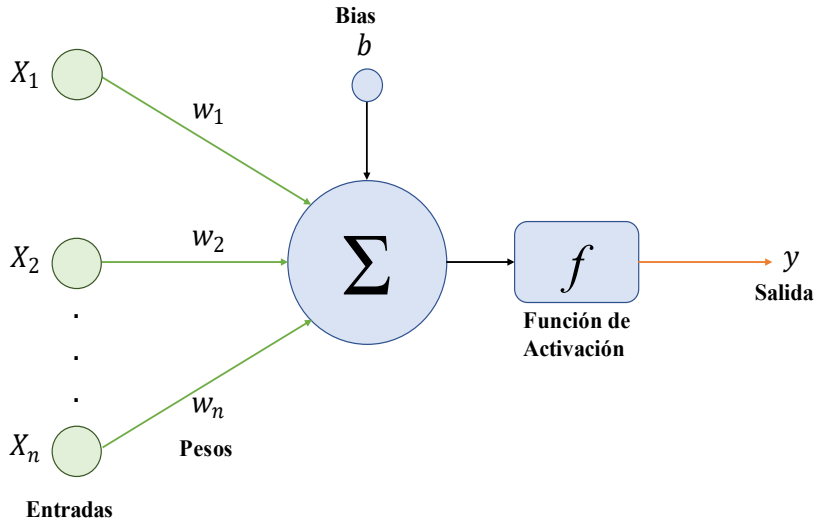

Figura 8. Arquitectura de una neurona. Adaptado de Braspenning et al., 1995.

2.8.1.2. Funcionamiento de una Red Neuronal. El funcionamiento de una DNN se puede dividir en dos pasos: Propagación hacia adelante (comúnmente conocido como Forward) y propagación hacia atrás (comúnmente conocido como Backward). El Forward es donde se computa la función de propagación hacia adelante para calcular la salida de la red y el Backward es la propagación hacia atrás para entrenar el modelo y ajustar los "mejores parámetros" (pesos) en base a una función de error.

Forward. En este paso, la salida de cada neurona a_j^L definida por la ecuación (15) se obtiene a partir de las salidas z_j^L de las neuronas de la capa anterior dadas a partir de la ecuación (16):

$$a_j^L = \sigma\left(z_j^L\right), \tag{15}$$

$$z_j^L = \sum_i w_{j,i}^L \cdot a_i^{L-1} + b_j^L = \sigma(W^L \cdot A^{L-1} + b^L), \tag{16}$$

donde L indica el número de la capa, j indica el número de la neurona de salida, W^L es el conjunto de pesos en la capa L, σ es la función de activación o no linealidad, A^{L-1} es el conjunto de salidas de las neuronas de la capa anterior y b es el bias o sesgo.

La rapidez del Aprendizaje Profundo se debe precisamente a que, para modelos ya entrenados, solo es necesario ejecutar el forward, compuesto únicamente por operaciones de adición y multiplicación, lo que lo hace computacionalmente veloz.

Backward. Este es el paso en el que "entrena" la red neuronal. En base a la reducción de una función de costo J descrita en la ecuación (17), se recalculan o actualizan los pesos sinápticos W hasta encontrar aquellos que en el forward obtengan un error mínimo. El descenso del gradiente es una de las técnicas más usadas para llevar a cabo este procedimiento y está descrito por la ecuación (18):

$$J(W) = J(a^L) = J\left(a^L(z^L)\right) = J(a^L(W^L \cdot A^{L-1} + b^L), \tag{17}$$

$$W := W - \alpha \frac{\partial J(W)}{\partial W}, \tag{18}$$

donde W es el conjunto de pesos aprendidos que se irán actualizando, α es la tasa de aprendizaje o la velocidad con la cual descenderá el gradiente, y $\frac{\partial J(W)}{\partial W}$ es la derivada de la función de costo J con respecto a los pesos W.

Para poder calcular $\frac{\partial J(W)}{\partial W}$ a lo largo de la red neuronal, es necesario *retropropagar* el error desde la última capa hasta la primera. Como J es una función compuesta, se usa la *regla de la cadena* de la ecuación (8) para definir el cambio del error con respecto a los pesos W y bias b en la capa L (ver figura 9).

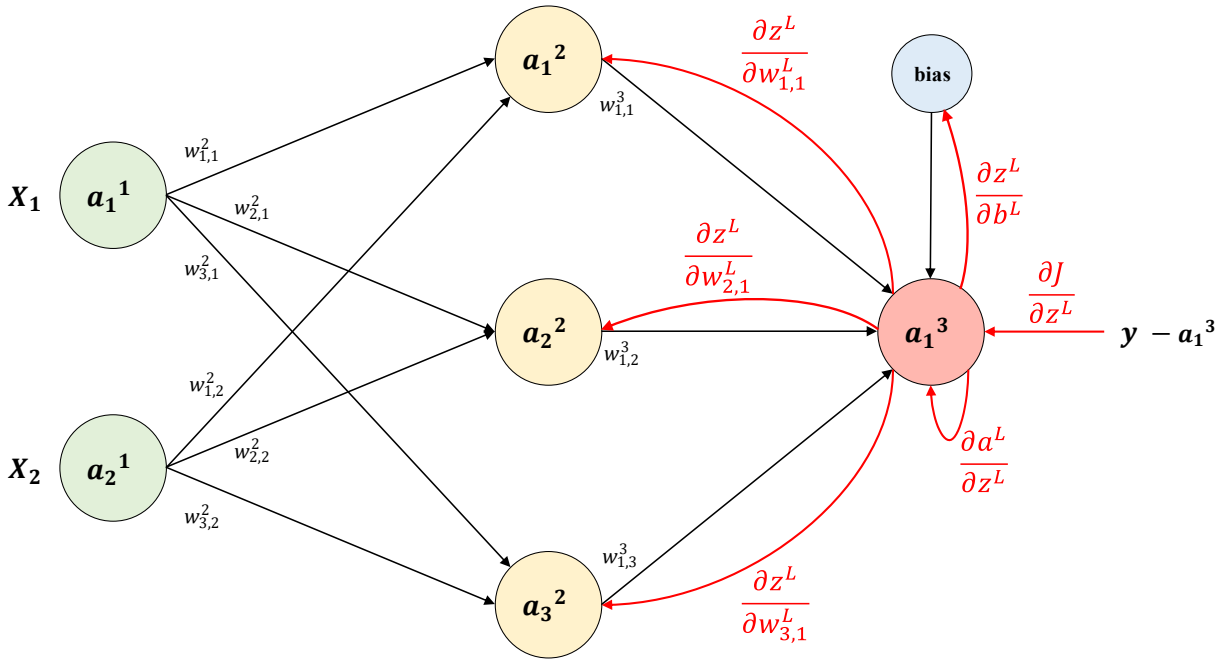

Figura 9. Retropropagación de la función de costo J en la capa L. Adaptado de Nielsen, 2015

Sea $J(W)$ la función compuesta definida mediante la ecuación (17), las derivadas de J respecto a W^L y a b^L respectivamente están dadas por las ecuaciones (19) y (20):

$$\frac{\partial J}{\partial W^L} = \frac{\partial J}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial W^L}, \tag{19}$$

$$\frac{\partial J}{\partial b^L} = \frac{\partial J}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial b^L}. \tag{20}$$

Si $J(a^L)$ está definida por la ecuación (21):

$$J(a^L) = \frac{1}{2n}\sum_{i}^{n}(y_i - a_i^L)^2 \,, \tag{21}$$

entonces, la derivada de J respecto a a^L está dada por la ecuación (22):

$$\frac{\partial J}{\partial a^L} = y_i - a_i^L \,. \tag{22}$$

Si la función de activación es una función sigmoide descrita por la ecuación (23):

$$a^L(z^L) = \frac{1}{1+e^z} \,, \tag{23}$$

entonces, la derivada de a^L con respecto a z^L está dada por la ecuación (24):

$$\frac{\partial a^L}{\partial z^L} = a^L(z^L)\big(1 - a^L(z^L)\big) \,. \tag{24}$$

Si z^L está definida por la ecuación (25):

$$z^L = \sum_{i}^{n} W_i^L \cdot A_i^{L-1} + b^L \,, \tag{25}$$

entonces, las derivadas de z^L con respecto a los pesos W^L y al bias b^L están representadas mediante las ecuaciones (26) y (27) respectivamente:

$$\frac{\partial z^L}{\partial W^L} = A_i^{L-1} \,, \tag{26}$$

$$\frac{\partial z^L}{\partial b^L} = 1 \,. \tag{27}$$

Si δ^L es el producto dado a partir de la ecuación (28):

$$\delta^L = \frac{\partial J}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \,, \tag{28}$$

reescribiendo y reemplazando δ^L en las ecuaciones (19) y (20), se obtienen las ecuaciones (29) y (30):

$$\frac{\partial J}{\partial W^L} = \delta^L \cdot A_i^{L-1}\,, \tag{29}$$

$$\frac{\partial J}{\partial b^L} = \delta^L \cdot 1\,. \tag{30}$$

Para propagar hacia atrás el error, es decir, a las capas anteriores a L, se expande la función compuesta $J(W)$ de la ecuación (17), obteniendo así la ecuación (31):

$$J(W) = J(a^L(W^L \cdot A^{L-1} + b^L))\,,$$

$$J(W) = J(a^L(W^L \cdot A^{L-1}(Z^{L-1}) + b^L)\,,$$

$$J(W) = J(a^L(W^L \cdot A^{L-1}(W^{L-1}A^{L-2} + b^{L-1}) + b^L))\,. \tag{31}$$

Posteriormente, se calculan recursivamente $\frac{\partial J}{\partial W^{L-1}}$ y $\frac{\partial J}{\partial b^{L-1}}$, mediante las ecuaciones (32) y (33):

$$\frac{\partial J}{\partial W^{L-1}} = \frac{\partial J}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial A^{L-1}} \cdot \frac{\partial A^{L-1}}{\partial Z^{L-1}} \cdot \frac{\partial Z^{L-1}}{\partial W^{L-1}}, \tag{32}$$

$$\frac{\partial J}{\partial b^{L-1}} = \frac{\partial J}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial A^{L-1}} \cdot \frac{\partial A^{L-1}}{\partial Z^{L-1}} \cdot \frac{\partial Z^{L-1}}{\partial b^{L-1}}\,. \tag{33}$$

Si z^L está definido a partir de la ecuación (34):

$$z^L = (W^L \cdot A^{L-1} + b^L)\,, \tag{34}$$

entonces, la derivada de z^L respecto a A^{L-1} está dada por la ecuación (35):

$$\frac{\partial z^L}{\partial A^{L-1}} = W^L \,. \tag{35}$$

Si $f(x)$ es una función de activación, y $\frac{\partial A^{L-1}}{\partial Z^{L-1}}$ es la derivada de dicha función en la capa anterior a L, descrita mediante la ecuación (36):

$$\frac{\partial A^{L-1}}{\partial Z^{L-1}} = d\big(f(x)\big) \,. \tag{36}$$

Si Z^{L-1} está definido por la ecuación (37):

$$Z^{L-1} = \sum_i^n W_i^{L-1} \cdot A_i^{L-2} + b^{L-1} \,, \tag{37}$$

entonces, las derivadas de Z^{L-1} respecto a W^{L-1} y a b^{L-1} respectivamente, están descritas a partir de las ecuaciones (38) y (39):

$$\frac{\partial Z^{L-1}}{\partial W^{L-1}} = A_i^{L-2} \,, \tag{38}$$

$$\frac{\partial Z^{L-1}}{\partial b^{L-1}} = 1 \,. \tag{39}$$

Reescribiendo las ecuaciones (32) y (33) se obtienen las ecuaciones (40) y (41):

$$\frac{\partial J}{\partial W^{L-1}} = \delta^L \cdot W^L \cdot d\big(f(x)\big) \cdot A_i^{L-2} \,, \tag{40}$$

$$\frac{\partial J}{\partial b^{L-1}} = \delta^L \cdot W^L \cdot d\big(f(x)\big) \cdot 1 \,. \tag{41}$$

En síntesis, el conjunto de pasos del algoritmo de retropropagación se define así:

Algoritmo 1 Algoritmo de Retropropagación para la actualización de parámetros en el aprendizaje de una red neuronal profunda.

1: Inicialización aleatoria de los pesos W.
2: Cálculo la salida a través del paso forward. ▷ver Ec. 16
3: Cálculo del error en la última capa δ^L. ▷ver Ec. 28
4: Propagar el error a las capas anteriores $\delta^{L-1} = \delta^L \cdot W^L \cdot d(f(x))$.
5: Calcular $\frac{\partial J(W)}{\partial W^L}$ y $\frac{\partial J(W)}{\partial b^L}$ mediante los cálculos anteriores.
6: Actualizar los pesos mediante Gradiente Descendiente. ▷ver Ec. 18

A cada ciclo de propagación hacia adelante y luego hacia atrás (forward y backward) se le denomina una época.

El proceso de funcionamiento de una red neuronal y retropropagación anteriormente descrito está basado en el trabajo de Nielsen (Nielsen, 2015).

2.8.1.3. Redes Neuronales Convolucionales. En la actualidad las redes neuronales cuentan con un número considerable de arquitecturas diferenciables por sus parámetros, su estructura propia y su tipo de aplicación. Dentro de esta cantidad de arquitecturas se encuentran las Redes Neuronales Convolucionales (Convolutional Neural Networks - CNN, en inglés) que son un tipo de arquitectura usada principalmente en el campo de la visión por computador para el reconocimiento de patrones dentro de imágenes.

Este tipo de red neuronal usa el principio matemático de la convolución discreta (ver figura 10), que es una transformación lineal que conserva la noción de la distribución misma y permite representar o extraer características de una entrada. Esto es aplicable a cualquier tipo de entrada, ya sea una imagen, un audio o una colección desordenada de características, sea cual sea su dimensionalidad (Dumoulin y Visin, 2016).

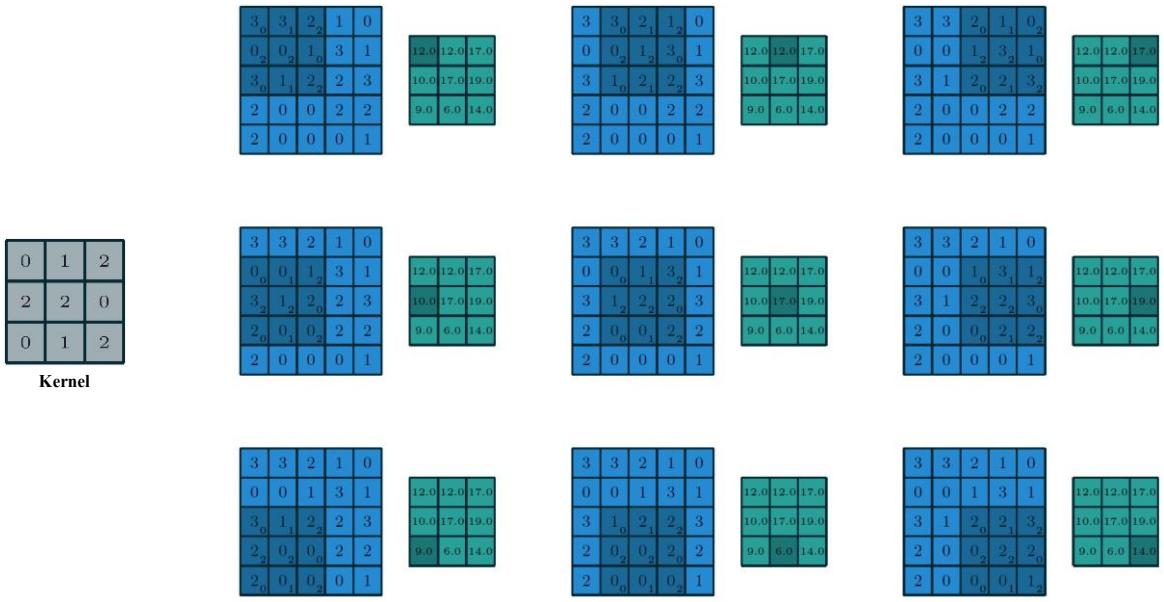

Figura 10. Cálculo de los valores de salida de una convolución discreta. Tomado de Dumoulin y Visin, 2016.

El kernel o un filtro móvil con el cual se realiza el proceso de convolución puede resaltar o suprimir, de forma selectiva, información contenida en una imagen o señal para destacar algunos elementos de ella, o también para ocultar valores anormales. Existen filtros especializados para detectar color, resaltar bordes, eliminar ruido, etc (Palomares et al, 2016).

Las CNNs son capaces de capturar con éxito las dependencias espaciales de una entrada, a través del aprendizaje de la mayoría de los kernels o filtros convolucionales. A partir de la señal de entrada x, la salida de capa subsiguiente z_j se calcula como en la ecuación (13), con la diferencia que, para el caso de las Redes Neuronales Convolucionales, los pesos W^l aprendidos son, en realidad, un conjunto de filtros aprendidos. Entonces, las capas son mapas de filtro y cada capa se puede escribir como una suma de convoluciones de la capa anterior (Royo y Ballesta, 2019).

La arquitectura de capas convolucionales (ver figura 12), permite capturar características de bajo nivel (primera capa) hasta características de alto nivel (última capa convolucional).

Debido a esto, es posible realizar un mejor aprendizaje al reducir el número de parámetros y procesar los datos localmente.

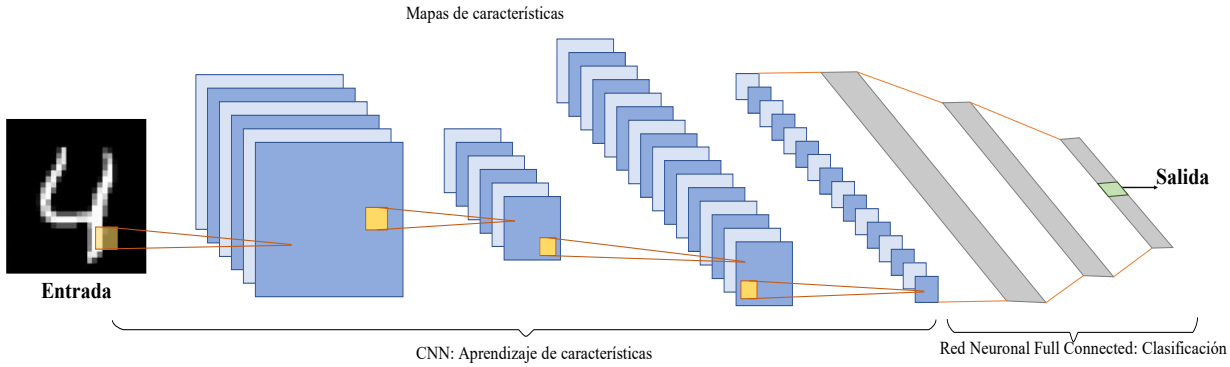

Figura 11. Arquitectura de una Red Neuronal Convolucional. Adaptado de O'Shea y Nash, 2015.

2.9. Variables de Interés

Son considerables las variables incidentes sobre el resultado del proyecto, entre ellas está la distribución de los datos (con sus respectivos valores estadísticos), la cantidad de puntos por nube, la distancia entre nubes, la rotación entre las dos nubes a registrar, la escala de las nubes.

La cantidad de puntos por nube es muy diciente para determinar la distribución de la nube. La distancia entre las nubes puede llegar a ser un problema según la implementación que se use (como veremos más adelante en el caso del ICP). La rotación de las nubes es definitivamente el factor más incidente sobre el problema a solucionar, dado que es la incógnita de ensamble entre nubes que se trata de hallar en este proyecto. Dependiendo de estas variables, se debe realizar tratamiento a los datos para su utilización, aunque cabe aclarar, este proyecto de investigación se centra en el uso de las nubes de puntos para su registro, no se ahonda en procesos de tratamiento o adecuación de los datos.

2.10. Instrumentos

Adicionalmente a las características intrínsecas de los datos, el resultado también es susceptible a los instrumentos o métodos utilizados para el desarrollo de la investigación, como por ejemplo el tratamiento de datos para hacer los cálculos computacionales menos complejos y el correcto uso de tecnologías.

2.10.1. Software

Para el desarrollo de la investigación utilizamos el lenguaje de programación Python, cuya relevancia hoy en día es amplia, especialmente por su facilidad y su calidad de código abierto, lo cual lo hace accesible a una mayor cantidad de personas. Dentro de este entorno de trabajo utilizamos librerías de código abierto como:

- **Tensorflow.** Es la principal librería utilizada en el proyecto para la implementación de modelos de aprendizaje automático. Ellos se definen como: "Plataforma de código abierto de extremo a extremo para el aprendizaje automático". Cabe aclarar que, por algunas implementaciones usadas para el desarrollo del proyecto, la versión de Tensorflow que se usa es la 1.14 [https://www.tensorflow.org/].
- **Keras.** Comparte cierta similitud con Tensorflow, esta librería se especializa en redes neuronales y deep learning. [https://keras.io/].
- **Numpy.** Esta librería es usada dentro del proyecto principalmente para el manejo de los datos (nubes de puntos) y algunos cálculos matemáticos sobre los mismos. Se definen como: "Paquete fundamental para la computación científica en Python. Es una biblioteca que proporciona un objeto de matriz multidimensional, varios

objetos derivados y una variedad de rutinas para operaciones rápidas en matrices". [https://numpy.org/].

- **Pandas.** Esta librería se usa en el proyecto para el manejo y tratamiento de los datos (especialmente los resultados de los cálculos hechos con las nubes), permitiéndonos hacer limpieza de los datos, encontrar relación entre datos y utilizar estructuras útiles para el entrenamiento de modelos de aprendizaje automático, como lo son los Dataframe. [https://pandas.pydata.org/].
- **Matplotlib.** Es la principal librería utilizada en el proyecto para graficar nubes de puntos y ver los resultados de los experimentos realizados sobre las mismas (traslaciones y rotaciones). Se definen como: "Una biblioteca integral para crear visualizaciones estáticas, animadas e interactivas en Python". [https://matplotlib.org/].

Las librerías anteriormente mencionadas fueron utilizadas mediante entornos de ejecución como Google Colaboratory, Spyder y Jupyter Notebook. Además, la mayoría de las implementaciones fueron codificadas en el entorno de trabajo Colaboratory, el cual permite el trabajo colaborativo mediante el uso de notebooks, una combinación de celdas de trabajo, en las cuales pueden ir bloques de código y celdas de texto, que permiten explicar los bloques de código relacionados a ellas.

2.10.2. Hardware

La mayoría del proyecto fue realizado en los computadores personales. Además, se usó la infraestructura de Supercomputación de la Universidad Industrial de Santander para ejecutar tareas de mayor necesidad de procesamiento y cuyas especificaciones son las siguientes:

2.10.2.1. GUANE (GpUs Advanced computiNg Environment). Es una infraestructura de referencia en cálculo de altas prestaciones de la Universidad Industrial de Santander. Desde su puesta en funcionamiento, concebida en coorganización y co-diseño por Hewlett-Packard (HPE), NVIDIA y el personal científico del grupo de Supercomputación y Cálculo Científico UIS (SC3UIS), ha sido considerada una de las infraestructuras de referencia en supercomputación a nivel internacional, no solo por su alta densidad y capacidades de procesamiento, sino por la innovación que representa **(**ver figura 12**).**

Características de Red:

- 1 Red Giga Ethernet de Administración
- 1 Red 10Gbps Infiniband

Especificaciones Técnicas:

GUANE – FRONTEND

- Fabricante: HPE
- Nombre: ProLiant DL380 G7
- Procesador: 2 Intel(R) Xeon(R) CPU E5640 @ 2.67GHz – 16 Cores
- Memoria Total: 98822288 kB – 94GB

GUANE - 5 NODOS Clase 1

- Fabricante: HPE
- Nombre: ProLiant SL390s G7
- Procesador: 2 Intel(R) Xeon(R) CPU E5645 @ 2.40GHz – 24 Cores
- Memoria Total: 107078768 kB – 102GB
- Dispositivos: 8 NVIDIA Tesla M2050 3GB

GUANE - 3 NODOS Clase 2

- Fabricante: HPE
- Nombre: ProLiant SL390s G7
- Procesador: 2 Intel(R) Xeon(R) CPU E5640 @ 2.67GHz – 16 Cores
- Memoria Total: 107078768 kB – 102GB
- Dispositivos: 8 NVIDIA Tesla M2050 3GB

GUANE - 8 NODOS Clase 3

- Fabricante: HPE
- Nombre: ProLiant SL390s G7
- Procesador: 2 Intel(R) Xeon(R) CPU E5645 @ 2.40GHz – 24 Cores
- Memoria Total: 107078768 kB – 102GB
- Dispositivos: 8 NVIDIA Tesla M2075 5G

Figura 12. GpUs Advanced computiNg Environment (GUANE). Tomado de "Cluster Guane" 2017.

2.10.2.2. FELIX (Framework to Enhance artificiaL Intelligence applications eXecution). Permite el alto rendimiento en aplicaciones de Inteligencia Artificial, principalmente aquellas que requieran ambientes de ejecución híbridos o basados en TPUs. (“Inicio”)

- Fabricante: Hewlett-Packard (HP)
- Nombre: ProLiant DL580 G7
- Procesador: 4 Intel(R) Xeon(R) CPU X7560 @ 2.27GHz – 64 Cores
- Memoria Total: 131844368 kB – 128GB
- Dispositivos: 2 NVIDIA GeForce GTX Titan X 12 GB

2.10.2.3. YAJE. Permite ofrecer una plataforma para explotar Visualización como Servicio (Viz as a Service) para usuarios científicos remotos. Este es un proyecto común, desarrollado con el apoyo de NVIDIA.

- Fabricante: Hewlett-Packard Enterprise (HPE)
- Nombre: ProLiant ML350 Gen9
- Procesador: 1 Intel(R) Xeon(R) CPU E5-2609 v3 @ 1.90GHz – 6 Cores
- Memoria Total: 49031292 kB – 48GB
- Dispositivos: 1 NVIDIA GeForce GTX Titan X 12 GB

Estado del Arte

3.1. Punto más Cercano Iterativo (ICP)

El Punto más Cercano Iterativo, reconocido comúnmente por su nombre en inglés Iterative Closets Point (ICP) es un algoritmo que estima la correspondencia de puntos mediante la métrica de distancia Euclidiana de la ecuación (3) y a partir de esto, usando la descomposición de valores singulares SVD con la ecuación (5), calcula la matriz de transformación rígida iterativamente hasta que la distancia entre los puntos correspondientes sea menor que un umbral o tolerancia dada, de modo que un conjunto de puntos se ajuste al otro bajo dicha transformación. Predeterminadamente usa la minimización de mínimos cuadrados, pero también se pueden usar otras funciones de criterio (Bergström & Edlund, 2014). El conjunto de pasos de ICP se describen a continuación:

Sean $A \in \mathbb{R}^3$ y $B \in \mathbb{R}^3$ dos conjuntos de N_A y N_B puntos a registrar, diferenciados mediante una transformación no conocida, tal que $N_A = N_B = N$. A se considera fija, y B desplazada y rotada.

1. Calcula los centroides de cada nube, mediante las ecuaciones (42) y (43)

$$centroide_A = \frac{1}{N}\sum_{i=1}^{N} A_i \,, \tag{42}$$

$$centroide_B = \frac{1}{N}\sum_{i=1}^{N} B_i \,, \tag{43}$$

   siendo A_i y B_i vectores de 3×1, por ejemplo $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$.

2. Mediante el cálculo de las ecuaciones (44) y (45), lleva las dos nubes de puntos al origen mediante la resta de su centroide,

$$A_{centrada} = A - centroide_A \,, \tag{44}$$

$$B_{centrada} = B - centroide_B \,. \tag{45}$$

3. Sea $\overrightarrow{p_A}$ un punto cualquiera de A. Calcula la correspondencia de puntos mediante la mínima Distancia Euclidiana calculada con la ecuación (3)

$$\min_{i \in \{1,\dots,N\}} d(\overrightarrow{p_A}, B_i) \,,$$

siendo d la Distancia Euclidiana y B_i un punto i del conjunto de puntos B.

4. Calcula la matriz de covarianza H entre las dos nubes de puntos resultantes asumiendo su correspondencia mediante la ecuación (46):

$$H = (A_{centrada}) \cdot (B_{centrada})^T \,. \tag{46}$$

5. Calcula la descomposición SVD de H representada mediante las ecuaciones (47) y (48), y con ello calcula los parámetros de rotación R y traslación t mediante las ecuaciones (49) y (51) respectivamente.

$$[U, S, V^T] = \text{SVD}\,(H) \,, \tag{47}$$

$$H = USV^T \,, \tag{48}$$

$$R = VU^T \,. \tag{49}$$

Despejando la ecuación (50), obtenemos el parámetro de traslación t de la ecuación (51):

$$R \cdot centroide_A + t = centroide_B \,, \tag{50}$$

$$t = centroide_B - R \cdot centroide_A \,. \tag{51}$$

Caso de reflexión especial: A veces, la descomposición SVD devolverá una matriz numéricamente correcta, pero, que en realidad no tiene sentido para la rotación R. Esto se comprueba con la negatividad del determinante de R, $\det(R) = |R| \leq 0$, y se soluciona multiplicando la tercera columna de V por -1.

6. Estructura de la matriz de transformación T a partir de los cálculos anteriores en la ecuación (52):

$$\mathrm{T} = \begin{pmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_1 \\ R_{2,1} & R_{2,2} & R_{2,3} & t_2 \\ R_{3,1} & R_{3,2} & R_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{52}$$

7. Transforma la nube de puntos A mediante la ecuación (53):

$$A_T = \mathrm{T}\{R\} \cdot A + \mathrm{T}\{t\} \,. \tag{53}$$

8. Mide el error cuadrático medio E entre la nube de puntos transformada A_T y la nube de puntos B, basado en el promedio de las distancias Euclidianas d, usando la ecuación (54):

$$E = \frac{1}{N}\big(d(A_T\,, B)\big) \,. \tag{54}$$

9. Itera, hasta que el error sea menor a la tolerancia dada. De este modo, se obtiene una matriz de transformación que permite alinear ambos conjuntos de puntos.

A partir de ICP, se han propuesto algoritmos variantes a este como GoICP (Yang et al.) que pretende mejorar los resultados mediante una buena inicialización de las correspondencias

de puntos y reducción de costo en los cálculos. También existe una modificación al ICP basada en la geometría y cromatografía que proporciona la nube de puntos propuesta por Alonso (Alonso et al., 2007) que se basa en la búsqueda automática para efectuar la correspondencia de puntos basados en su geometría o por su color, además de hacerlo mediante muestreos aleatorios, lo que traduce en la reducción de costos computacionales.

3.2. Consenso de Muestra Aleatoria (RANSAC)

El algoritmo Consenso de Muestra Aleatoria, reconocido por su nombre en inglés Random Sample Consensus (RANSAC), es altamente efectivo cuando los datos se ven afectados por puntos atípicos, como es el caso de las nubes de puntos obtenidas del mundo real mediante sensores. Es un estimador robusto que se ha convertido en estándar en el campo de visión por computador (Chum & Matas, 2008).

Utilizando una medición del error en la agrupación de los puntos de entrada (esta agrupación puede ser, por ejemplo, una recta o una figura sinusoidal), los datos se ven agrupados en dos subconjuntos, datos típicos *(inliers)*, que son los puntos que se ajustan a una determinada correspondencia, y datos atípicos *(outliers)*, que son los puntos que quedan fuera de la correspondencia.

Para la obtención del mejor modelo que haga esta distinción de los dos subconjuntos previamente explicados se realizan los siguientes 3 pasos:

1. Se muestrea un pequeño subconjunto de puntos de los datos de entrada y se considera a estos puntos como inliers (un conjunto de correspondencias tentativas).
2. Se determinan los parámetros que permiten llegar desde los datos de entrada a este subconjunto de puntos que se hallaron anteriormente.

3. Se evalúa el modelo determinando cuántos de estos puntos son inliers y cuantos outliers, así es la medición estándar para RANSAC.

Este proceso se itera hasta obtener el mejor modelo que, soportado por los datos de puntos, dé la mejor aproximación a la agrupación de inliers y outliers.

Con este modelo se puede definir la mejor distribución para los datos, la cual nos permitirá inferir la matriz de transformación a partir de correlación cruzada (Dennis et al., 2021).

La figura 13 muestra un ejemplo básico de agrupación de puntos dada una recta generada por dos puntos dentro de los puntos de entrada, A la izquierda se ve la selección de puntos (inliers y outliers) basados en la línea azul y un umbral determinado por las líneas punteadas, a la derecha se ve como esta nueva aproximación tiene un número mayor de puntos inliers que el anterior.

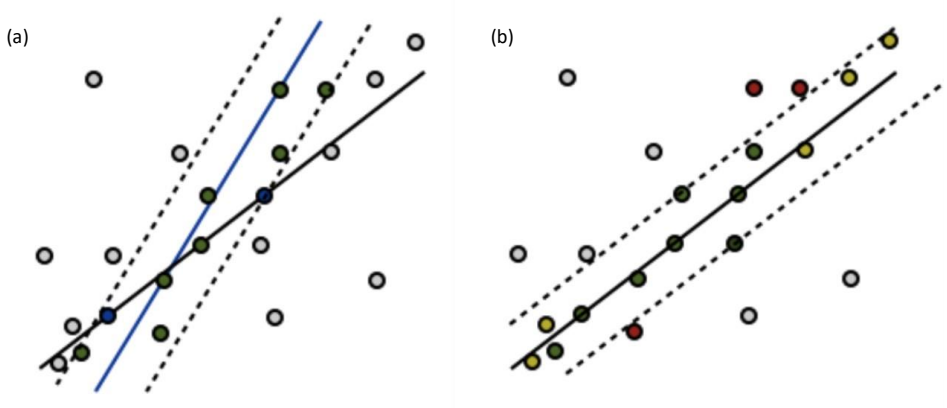

Figura 13. Ejemplo del proceso iterativo RANSAC, (a) Iteración con pocos inliers, (b) Iteración con mejor agrupación de inliers, Tomado de Skala, 2013, p.32.

3.3. Acumulación de Puntos Coherente (CPD)

Según Myronenko & Song, 2010. El algoritmo de Acumulación de Puntos Coherente, más conocido por su nombre en inglés Coherent Point Drift (CPD), es útil tanto para transformaciones rígidas como no rígidas, se basa en métodos probabilísticos al considerar la

alineación de dos conjuntos de puntos como un problema de estimación de densidad de probabilidad.

Para realizar la alineación de nubes de puntos se usa un Modelo de Gaussianas Mixtas (GMM) para representar uno de los dos conjuntos de puntos a alinear. Se ajustan los centroides del GMM al segundo conjunto de puntos para maximizar la probabilidad. Se fuerzan los centroides a moverse coherentemente como un grupo, esto para preservar la estructura topológica del conjunto de puntos.

En un punto óptimo, los conjuntos de puntos se alinean y la correspondencia se obtiene usando el máximo de probabilidad posterior del GMM para un punto de datos dado.

Según Myronenko (Myronenko et al., 2006) para el alineamiento no se hace ninguna suposición explícita del modelo de transformación, en lugar de esto, se considera el proceso de adaptar los centroides de las Gaussianas desde su posición inicial a su posición final como un proceso de movimiento temporal e imponer una restricción de coherencia de movimiento sobre un campo de velocidad (ver figura 14). La coherencia de velocidad es una forma particular de imponer suavidad en la trasformación subyacente.

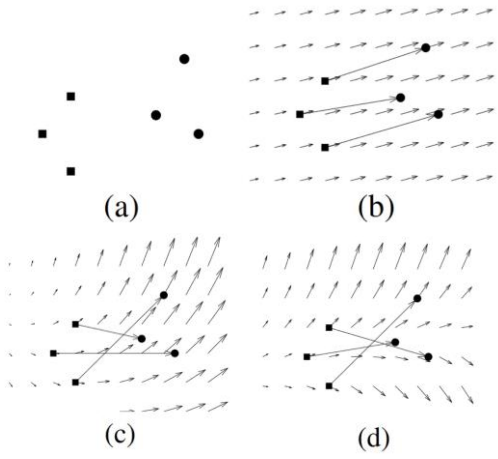

Figura 14. (a) Dos conjuntos de puntos. (b) Campo de velocidad coherente. (c,d) Campos de velocidad que son menos coherentes para las correspondencias dadas. Adaptado de Myronenko et al., 2006, p. 3

En la ejecución del CPD se efectúan los siguientes pasos:

Algoritmo 2. Algoritmo de Acumulación de Puntos Coherente (CPD) para el registro de nubes de puntos.

1: Submuestreo de las nubes de puntos que se desean alinear.
2: Inicialización de parámetros de ponderación de ruido, rotación y traslación.
3: Cálculo de la probabilidad posterior del conjunto de puntos según un GMM. ▷ver sec. 2.7
4: Actualización de los parámetros de rotación y traslación maximizando la probabilidad posterior mediante la ejecución del algoritmo Expectation Maximization (EM). ▷ver sec. 2.7
5: Evaluación de las condiciones de error para la alineación.
Si Se cumplen las condiciones **entonces**
Finaliza el algoritmo
No
Vuelve al paso 3.
6: **Fin Si**

3.4. Registro basado en Aprendizaje Profundo

Por el lado de la Inteligencia Artificial, se han propuesto algoritmos de Aprendizaje Profundo (Deep Learning) para tratar con nubes de puntos. Para problemas como los ya mencionados, las redes neuronales poseen gran ventaja frente a otros algoritmos comunes, ya que a partir de su arquitectura se puede realizar la extracción "automática" de características relevantes o *keypoints*, que contribuyen al proceso de establecer correspondencias entre pares de puntos (Silva et al., 2017).

Específicamente para el registro de nubes de puntos se ha propuesto la PCR-Net (Sarode et al., 2019) la cual aprende comparando nubes de puntos. El método tiene como objetivo encontrar la transformación que efectué correctamente la alineación de los puntos. Esta red, a su vez usa PointNet para obtener representaciones de puntos y se puede entrenar usando como función de pérdida la distancia de Chaflán o la Distancia del Movimiento de la Tierra (Earth Mover's Distance - EMD) entre las dos nubes de puntos.

La PointNet propuesta por Qi (Qi et al., 2017), es una arquitectura diseñada para recibir como entradas las nubes de puntos directamente, es decir, sin ningún tipo de transformación previo, que aprende características tanto locales como globales, proporcionando un enfoque simple, eficiente y efectivo para tareas de segmentación y reconocimiento 3D. A partir de esta red, se han propuesto arquitecturas como la PPFNet, que aprende descriptores de características 3D locales, al corriente del contexto global (Deng et al., 2018) y la VoxNet, que mediante su arquitectura supervisada de red neuronal convolucional 3D permite el reconocimiento de objetos dado un segmento de nube de puntos, aún en entornos que presentan ruido o desorden (Maturana y Scherer, 2015).

Otros aportes como la arquitectura PointNetLK (PointNet Lukas Kannade), que integra una red neuronal recurrente (RNN) modificada a la PointNet mencionada anteriormente, y permite realizar el alineamiento de nubes de puntos en entornos de ruido u oclusión. Además, la PointNetLK ofrece muy buen rendimiento computacional, lo que abre nuevos caminos para la profundización en el ámbito de reconstrucción 3D, y aplicación de técnicas de AI en la integración de nubes de puntos (Aoki et al., 2019).

En el trabajo de Pérez Gonzalez (Pérez Gonzalez et al., 2019) se propuso un método que a diferencia de los otros no asume previamente la cercanía entre nubes de puntos. Concretamente plantearon una red neuronal profunda capaz de aprender y estimar adecuadamente los parámetros de rotación y traslación necesarios para efectuar el alineamiento de puntos sin necesidad de requerir ajustes adicionales mediante algoritmos como el ICP.

Metodología

4.1. Primera Aproximación

En principio, se propuso realizar el registro de nubes de puntos usando una red neuronal profunda alimentada con las coordenadas de nubes de puntos y ángulos de rotación en los ejes x, y, z, basados en (Dennis et al., 2021). La red neuronal está configurada de la siguiente manera:

Conjunto de datos: Se usó un conjunto de datos de elaboración propia con 32768 nubes de puntos, cada una con tamaño de [2724, 3] a partir del modelo del conejo de Stanford (The Stanford 3D Scanning Repository, s.f.). Para realizarlo, se rotó el modelo del conejo en los 3 ejes coordenados cada 0.1 radianes y se registraron dichos valores de rotación para ser el *target* o etiquetas de los datos que alimentarían a la red. El total de datos fue dividido en el 80% para entrenamiento y el 20% para test. Así, se entrenó la red neuronal con 26214 modelos (x) y vectores de ángulos (y), y se testearon con 6554 modelos y ángulos.

Red Neuronal: Las entradas pasan a una capa de 1024 neuronas. En la capa de salida se tienen 3 neuronas, una para cada predicción del ángulo x, y, z, respectivamente.

El modelo se entrenó minimizando el error mediante el Gradiente Descendiente Estocástico, con una tasa de aprendizaje de 1×10^{-2} por 50 épocas. Por tratarse de un problema de regresión, se usó el Error Cuadrático Medio (ECM) como métrica de costo.

Resultados: En la tabla 1 se puede observar un ejemplo de los valores predichos por la red neuronal para dos nubes de puntos de la categoría “bunny” diferenciadas por una rotación inicial conocida. La salida de dicha red son 3 ángulos de rotación, uno por cada eje coordenado. También se presenta el error relativo (ecuación 84) de los ángulos predichos (valor teórico) con respecto a los iniciales (valor experimental) (ver figura 15):

Tabla 1. Resultados primera aproximación al registro de nubes de puntos.

	Eje X	**Eje Y**	**Eje Z**
Ángulo inicial [radianes]	1.450	1.509	0.368
Ángulo predicho [radianes]	1.688	1.435	0.668
Error Relativo [%]	16.413	4.903	81.521

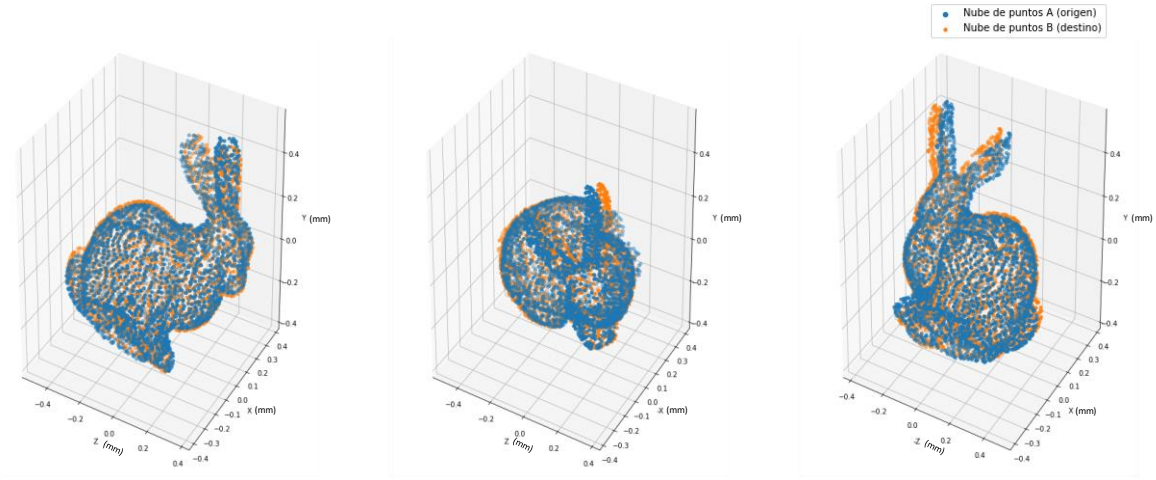

Figura 15. Ejemplo de registro de nubes de puntos usando una Red Neuronal Profunda. La nube de puntos A es la nube original, y la nube de puntos B es la nube destino.

Según los resultados obtenidos, y a pesar de que estos fueron buenos para ser la primera aproximación de registro, se determinaron los siguientes inconvenientes:

Variabilidad y dependencia de clases: Como la red neuronal se alimenta con coordenadas de los puntos, el problema se remonta a que el modelo propuesto solo aprenderá a registrar nubes de puntos del modelo “bunnny” (Conejo de Stanford), u otras cuya estructura, tamaño y dimensiones sean similares o iguales al modelo con el que se realizó el entrenamiento de la red. Es decir, si se quisiera registrar otra nube de puntos de otra clase, no se obtendrían óptimos resultados.

Para solucionar esto, se tendría que entrenar la red con la máxima variabilidad de clases posible, y, aun así, esto no garantizaría buenos resultados debido a que la red sigue procesando los datos en términos de coordenadas.

A partir de lo anterior, se establece la necesidad de encontrar una métrica de comparación entre dos nubes de puntos para generalizar el problema de registro de una clase a cualquier nube de puntos.

4.2. Distancia Profunda de Nubes de Puntos (DPDist)

El método de Distancia Profunda de Nubes de Puntos, conocido por su nombre en inglés Deep Point Cloud Distance (DPDist) propuesto por Urbach (Urbach et al., 2020), se usa para comparar dos nubes de puntos a partir de la estimación de una métrica cuantitativa entre ellas. Esta métrica se define en términos de "distancia" de los puntos desde una nube de puntos hasta la superficie continua subyacente correspondiente a la otra.

Sean $A, B \in \mathbb{R}^3$ dos nubes de puntos de tamaño N. Sea a_i un punto de A con coordenadas $[x, y, z]$. El propósito de DPDist es encontrar la "distancia" mínima entre el punto $a_i \in A$ y su punto coincidente y en la nube de puntos B, definida mediante la ecuación (55):

$$D(a, B) = \min_{y \in B} d(a, y)\,, \tag{55}$$

donde $D(.)$ es la distancia desde un punto a a la superficie B y $d(.)$ es una métrica de distancia entre los puntos a y $y \in \mathbb{R}^3$, por ejemplo, la distancia Euclidiana descrita en la ecuación (3).

DPDist plantea usar otra forma de representación de las nubes de puntos para solucionar la limitación de las coordenadas y generalizar la comparación para cualquier nube de puntos. Entonces, DPDist usa la representación del Vector de Fisher Modificado 3D (3DmFV) y con ello cambia la representación del sistema coordenado $[x, y, z]$ de los puntos a una representación

bidimensional mediante el cálculo de valores estadísticos que describen los puntos en el espacio respecto a una grilla de gaussianas. Esto permite dividir y describir la nube de puntos localmente y además aumentar la rapidez, precisión y disminuir la complejidad computacional de procesar los puntos directamente.

4.2.1. Cambio de representación de nubes de puntos

Usar nubes de puntos como entrada de una Red Neuronal Profunda no es una tarea sencilla, esto se debe a que este tipo de datos no tienen una estructura definida, es decir, son datos desordenados y no tienen un número fijo de puntos. Se han propuesto varios métodos para extender las Redes Neuronales a este tipo de datos 3D como la representación de nubes de puntos en voxeles (Maturana & Scherer, 2015; Qi et al., 2016; Wu et al., 2015), que discretiza el espacio 3D de manera similar a una imagen que discretiza el plano de proyección de una cámara, y que mediante la aplicación de redes como Inception (Szegedy et al., 2017) y ResNet (He et al., 2016) a datos voxelizados ha logrado una buena precisión, pero a un muy alto costo computacional y tiempo de entrenamiento.

La representación del Vector de Fisher modificado 3D (3DmFV), aunque está directamente relacionada a la representación del espacio coordenado de una nube de puntos, sí puede ser usada de manera eficaz como entrada a una red neuronal.

4.2.1.1. Vector de Fisher (FV). Antes de ver el 3DmFV, se debe tener en cuenta lo que es la representación del Vector de Fisher (FV) (Perronnin & Dance, 2007; Perronnin et al., 2010). Es un método de agregación de descriptores que caracteriza muestras de datos de

diferentes tamaños a partir de su desviación usando el modelo de mezcla gaussiana (GMM) (11).

Se basa en el principio del Kernel de Fisher (Ben-Shabat elt al., 2017), computando los gradientes del logaritmo de verosimilitud de la muestra con respecto a los parámetros del modelo (peso, media y covarianza). Sus características son funciones simétricas, lo que las hace independientes e invariantes del orden y de la estructura.

Sea, $X = \{p_t \in R^3, t = 1, \dots, T\}$ un conjunto de puntos 3D que pertenece a una nube de puntos dada, donde T denota el número de puntos en el conjunto.

Sea $\lambda = \{(w_k, \mu_k, \Sigma_k), k = 1, \dots, K\}$ un conjunto de parámetros de un componente K del GMM, donde w_k, μ_k, Σ_k son el peso de la mezcla, el valor esperado y la matriz de la covarianza de la K-ésima Gaussiana respectivamente.

La probabilidad de un único punto asociado con la densidad del GMM está dada por la ecuación (56):

$$u_\lambda = \sum_{k=1}^{K} w_k u_k(p)\,. \tag{56}$$

Dado un GMM específico y bajo el supuesto de independencia común. El FV, denotado como $\mathcal{G}_\lambda^x$, describe la suma de los estadísticos, descritos por λ, del gradiente normalizados, calculado para cada punto p_t mediante la ecuación (57):

$$\mathcal{G}_\lambda^x = \sum_{t=1}^{T} L_\lambda \nabla_\lambda \log u_\lambda(p_t)\,, \tag{57}$$

dónde L_λ es la raíz cuadrada de la matriz inversa de la información de Fisher.

El FV puede ser visto como una generalización de la implementación del modelo Bag of Visual Words (BoV), este sufre menor perdida de características importantes a comparación de otros cambios de representación, como es el caso de la basada en voxeles, que usa desratización al momento de la extracción de características, mientras que en el FV se usan funciones continuas.

4.2.1.2. Vector de Fisher 3D Modificado (3DmFV). Según Ben-Shabat (Ben-Shabat elt al., 2017), la implementación del 3DmFV surgió del buen rendimiento de las redes convolucionales en imágenes, al querer tener un funcionamiento similar en nubes de puntos se requería una adaptación a la estructura de estos datos. Este cambio de representación para la nube de puntos es hibrido, pues, combina la estructura discreta de una cuadricula, con la generalización continua de los vectores de Fisher. En el caso de FV, a pesar de que la suma es una función simétrica y asintóticamente óptima, no provee toda la información acerca de los puntos de entrada para un conjunto finito de puntos.

Para el caso práctico de conjuntos grandes de puntos se propone adicionar información manteniendo la independencia al orden. Se escogieron adicionar funciones máximas y mínimas, por tanto, los componentes del propuesto 3DmFV están dados por la ecuación (58):

$$\mathcal{G}^{x}_{3DmFV_\lambda} = \begin{bmatrix} \sum_{t=1}^{T} L_\lambda \nabla_\lambda \log u_\lambda(p_t)|_{\lambda=\alpha,\mu,\sigma} \\ max_t \left(L_\lambda \nabla_\lambda \log u_\lambda (p_t)\right)|_{\lambda=\alpha,\mu,\sigma} \\ min_t\left(L_\lambda \nabla_\lambda \log u_\lambda (p_t)\right)|_{\lambda=\mu,\sigma} \end{bmatrix}. \quad (58)$$

Cada componente puede ser la función suma, máximo o mínimo, evaluadas en un conjunto de un componente gradiente. Este tipo de representaciones parciales y más compactas conducen a una mejora en la precisión. Se encontró que la derivada del peso mínimo es siempre constante y se omitió esa función específica.

Para K Gaussianas habrá un total de $20xK$ componentes en la representación del 3DmFV. Los 20 componentes surgen de calcular las funciones anteriormente descritas, para $\lambda = \alpha, \mu, \sigma$ (exceptuando α en la función min) y para cada una de ellas evaluarlas en los 3 ejes coordenados (x, y, z). A continuación, en la figura 16 se presenta el 3DmFV para la representación de un único punto dentro de una Gaussiana.

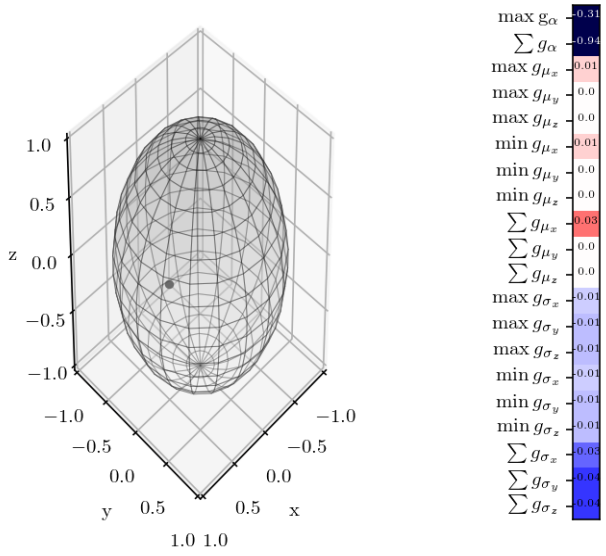

Figura 16. Representación del 3DmFV para una Gaussiana. Tomado de Ben-Shabat, 2018.

Cada columna dentro de la matriz de la representación 3DmFV hace referencia a una Gaussiana dentro de la cuadricula y cada fila describe un estadístico o componente del vector de Fisher.

A continuación, se presenta el 3DmFV para la nube de puntos del conejo de Stanford y una cuadricula de 8 Gaussianas (4x4) (ver figura 17). Los valores en 0 son las zonas blancas, los valores positivos y negativos corresponden respectivamente a las zonas rojas y azules respectivamente.

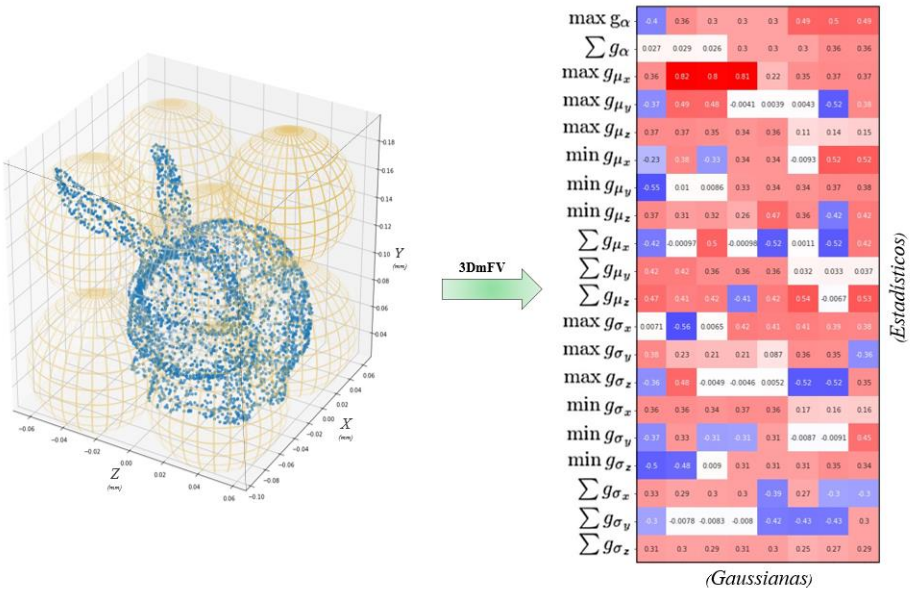

Figura 17. Representación del 3DmFV del modelo "bunny" para una cuadricula de 8 Gaussianas, Adaptado de Ben-Shabat, 2018.

El 3DmFV modifica y generaliza el FV de dos formas importantes:

- EL GMM propuesto es especificado usando un conjunto de Gaussianas uniformes continuos con centros en una cuadricula tridimensional, es decir, las gaussianas se distribuyen en la cuadricula propuesta.
- Los componentes que caracterizan el conjunto de puntos se generalizan a otras funciones de este conjunto, siendo en total 20 funciones que describen la distribución de puntos que contiene una gaussiana.

Algunas ventajas del 3DmFV son:

- Mantiene las propiedades continuas de una nube de puntos.
- La configuración de cuadrícula o grilla, a pesar de conservar las propiedades espaciales de las nubes de puntos, usa funciones simétricas que lo hacen independiente a su estructura. Esto hace que sea ideal para ser usada como entrada a una Red Neuronal Convolucional.
- Cada componente dentro de la representación describe una propiedad clara y significativa.

3DmFV representa la probabilidad de cada punto asociado con un modelo de mezcla gaussiana GMM y luego aplica las funciones simétricas para obtener una representación global generalizada del Vector de Fisher (ver figura 18).

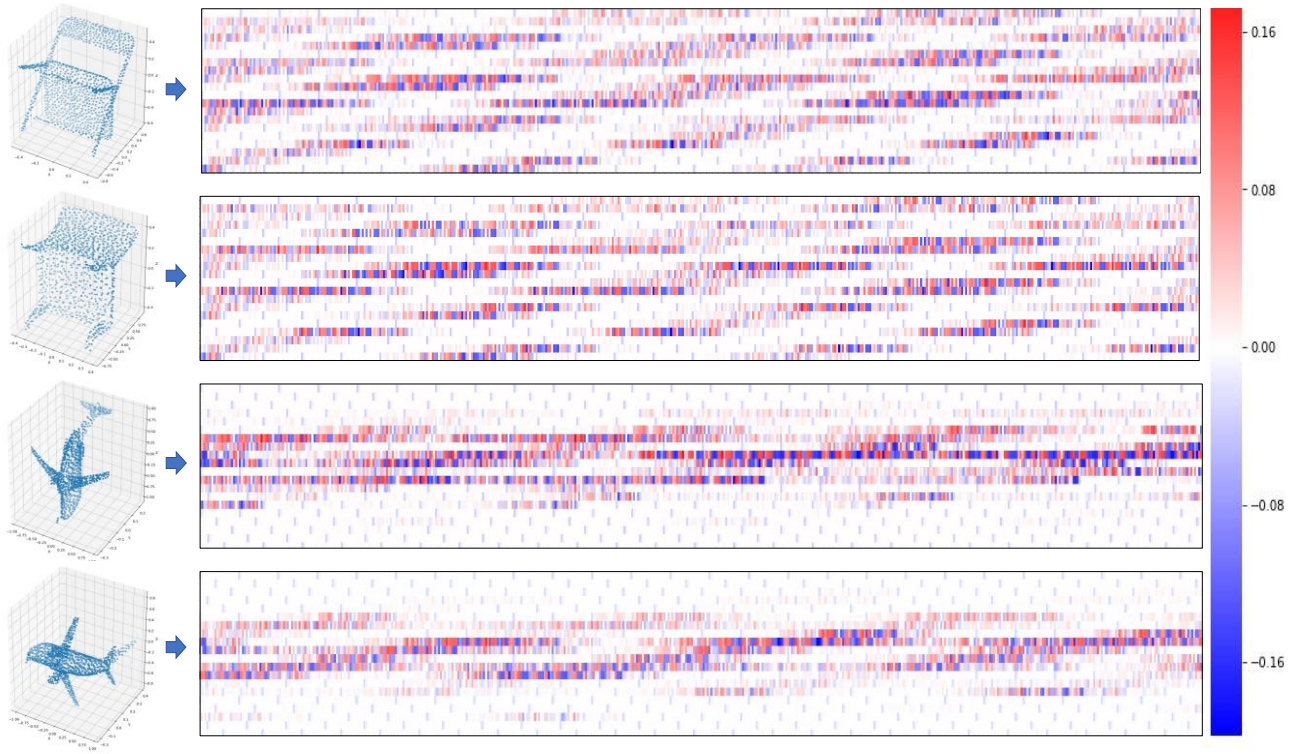

Figura 18. Representación del 3DmFV para nubes de puntos del ModelNet40 con parámetro $K=8^3$ Gaussianas

4.2.2. Superficie Subyacente

Una de las principales ventajas de la representación 3DmFV es la generación de una superficie subyacente a partir de la nube de puntos que modela, esto permite la comparación de puntos a superficie, el cual es el caso para el DPDist.

En la figura 19 se observan dos conjuntos de puntos $\boldsymbol{S_A}$ y $\boldsymbol{S_B}$ tomados de la misma nube de puntos, pero son puntos sin coincidencias exactas en el otro conjunto. Para este caso se genera una superficie subyacente a partir del conjunto $\boldsymbol{S_A}$, permitiendo encontrar la distancia de comparación de los puntos del conjunto $\boldsymbol{S_B}$ con la superficie generada.

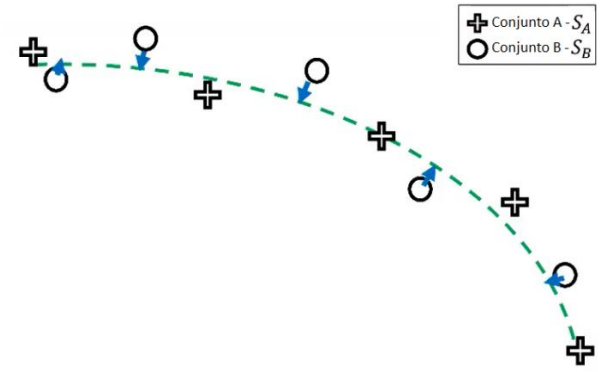

Figura 19. Superficie subyacente generada a partir de un conjunto de puntos. Tomado de Urbach, 2021**.**

Como se menciona a partir del ejemplo, el comparar puntos sin coincidencias aparentes es también otra gran ventaja del uso del 3DmFV como representación de la nube de puntos ya que esto permitirá realizar el proceso de comparación con nubes que, tras fenómenos de oclusión y reflectancia, pierdan datos de sus puntos o que naturalmente tengan pocas o nulas coincidencias exactas en sus nubes.

4.2.3. Estructura de DPDist

DPDist usa una red neuronal convolucional para estimar la distancia de un punto a la superficie asociada a la otra nube de puntos. Esta red neuronal es alimentada con las coordenadas del punto b_i concatenadas con la representación local del punto en términos del vector de Fisher modificado 3D $L^A(b_i)$. La red neuronal ψ aprende a estimar la distancia $\psi\left(b_j, L^A\left(b_j\right)\right)$ entre el punto y su correspondiente representación local, esta distancia se denota como Single Point Distance (SPD) y se define mediante la ecuación (59):

$$D_{SPD}\left(b_j, A\right) = \psi\left(b_j, L^A\left(b_j\right)\right). \quad (59)$$

Como se explica en el trabajo de Urbach (Urbach et al, 2020, p,19) dadas dos nubes A y B, se busca estimar la distancia entre cada punto consultado desde B hasta la superficie representada por A. El SPD usa el 3DmFV para extraer la representación de la superficie generada con A. Luego se aplica el proceso de localización para cada punto consultado b_j para encontrar su punto de grilla más cercano, dado el punto de grilla se obtiene la subcuadrícula de la localización $L^A\left(b_j\right)$.

La Distancia de un Punto a la Superficie (Single Point Distance - SPD) procesa cada punto consultado b_j y lo concatena con su respectiva representación local para producir la distancia desde la superficie (ver figura 20).

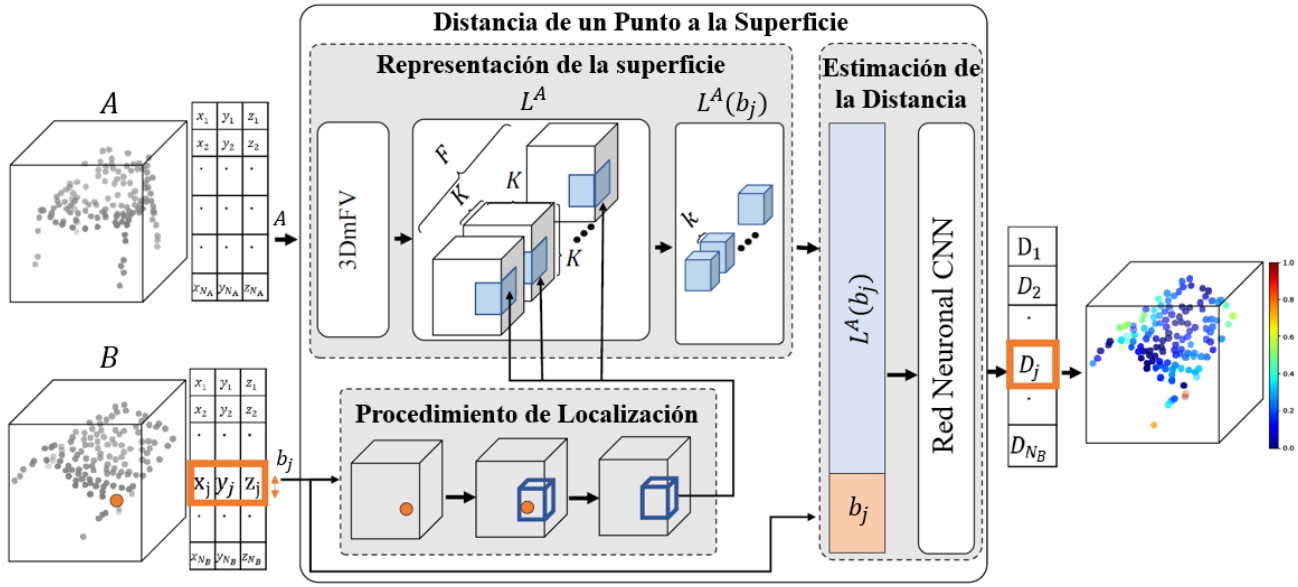

Figura 20. Estructura general del DPDist. Adaptado de Urbach et al., 2020.

4.2.3.1. Representación Local. Según el trabajo de Urbach (Urbach et al., 2020), debido a la dificultad de aprender una función de distancia se prefiere modelar la superficie subyacente A (S_A) por partes, así cada parte es geométricamente menos compleja.

Este modelamiento es posible en la representación del 3DmFV gracias a su estructura de cuadricula, pues, nos permite especificar una representación local y parcial escogiendo una sub cuadricula.

Para realizar este tipo de representación se realizan los siguientes pasos:

1. Se calcula la representación global de la nube *A* para ello, se define el tamaño de la cuadrícula de representación K, por lo que obtendremos una subcuadrícula de tamaño $KxKxK$, Esta grilla especifica la mezcla de Gaussianas (cantidad de Gausianas de la cuadricula para el 3DmFV), una por cada punto de grilla.
2. Se calculan los F estadísticos para cada Gaussiana y los concatenan a la representación global denominada $L^A \in R^{KxKxKxF}$, estos resultan de calcular la

derivada de la Gaussiana con respecto a sus parámetros y tomando estadísticas máximas, mínimas y promedio de esas derivadas. La representación global es calculada una vez por cada nube.

3. Para cada punto consultado b_j de la nube de puntos *B*, encontrar su punto de grilla más cercano (la gaussiana más cercana) y se especifica una subcuadrícula de tamaño k , con $k \leq K$, centrado en el punto de grilla más cercano hallado.
4. Para la obtención de la representación local, se corta la parte especifica de la representación global L^A que corresponda con la subcuadricula hallada, esta representación local de denota como $L^A(b_j) \in R^{kxkxkxF}$.

Note que la representación local es una representación de la nube de puntos A en una región determinada por b_j.

La red neuronal que procesa la subcuadrícula concatenada con las coordenadas del punto del cual fue hallada tiene la arquitectura de la figura 21.

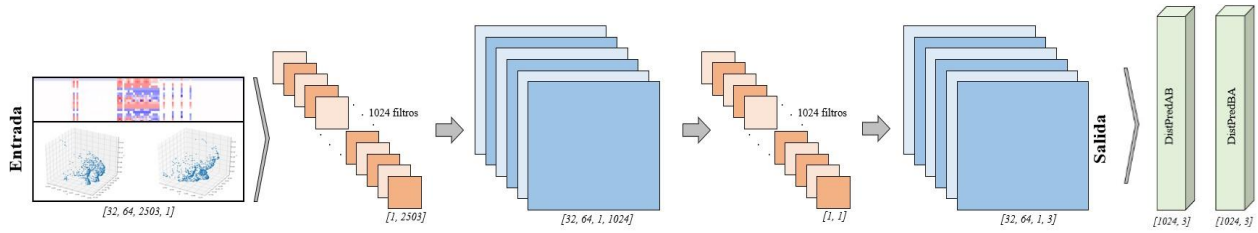

Figura 21. Arquitectura de la red convolucional que estima las distancias DPDist.

Además de la estimación de la distancia punto a superficie, se puede hallar la distancia entre nubes de puntos.

Siguiendo la estructura representada por la figura 21, la distancia promedio entre todos los puntos de la nube B a la superficie subyacente correspondiente a la otra nube, A, es una

buena alternativa para hallar esta distancia, la versión simétrica de la anterior idea se presenta con la ecuación (60):

$$DPDist(A,B) = \frac{1}{N_A}\sum_{i=1}^{N_A} D_{SPD}(a_i, B) + \frac{1}{N_B}\sum_{i=1}^{N_B} D_{SPD}(b_i, A)\,. \tag{60}$$

4.2.4. Entrenamiento DPDist

El código fuente de DPDist está bajo licencia MIT, la cual permite que cualquier persona que obtenga una copia de este software y los archivos de documentación asociados, pueda operar con este sin restricciones (Licenses, U.C.C.P, 2013). Además, está programado en Python y su módulo de Deep Learning está codificado usando la librería de Tensorflow. Por tal razón, este modelo fue ideal para ser usado en la resolución del problema de la comparación de nubes de puntos y a su vez, en la intención de solucionar el problema de registro usando técnicas de Aprendizaje Profundo, el principal motivo de este trabajo de investigación.

El entrenamiento del modelo fue realizado mediante el uso de los servidores FELIX y YAJE, recursos de computación y procesamiento del alto rendimiento de la Universidad Industrial de Santander (Recursos de computación de alto rendimiento, 2020).

El uso de estos servidores permitió el entrenamiento del modelo de DPDist de manera remota mediante conexión SSH por consola empleando el frontend de GUANE y el sistema de gestión SLURM.

Conjunto de datos. El entrenamiento del modelo fue realizado con los parámetros por defecto y la configuración original de DPDist (Urbach et al., 2020) para obtener un performance similar o igual al original. Se usó el conjunto de datos de ModelNet40 (Wu et al., 2015) que consta de 12311 modelos CAD, distribuidos en 40 categorías de objetos.

El conjunto de datos con el cual se entrenó está conformado únicamente por los modelos de la clase “chair” de ModelNet40. De este subconjunto se tomaron 889 modelos para entrenamiento y 100 modelos para test. Se eligió solo la categoría “chair” ya que se demostró en el trabajo de Urbach (Urbach et al., 2020) que, al entrenar el modelo sobre representaciones locales de una única categoría, hace que este se pueda generalizar a otras categorías de objetos. A partir de lo anterior, para obtener las distancias reales con las cuales entrenar, se realizó un muestreo de puntos de cada modelo y se registraron sus distancias a la superficie (Ground truth o Y).

En el trabajo de Urbach (Urbach et al., 2020 p. 9) ,la representación 3DmFV para los modelos de las sillas que servirán como input se realizó con los parámetros *K=8* y *k=5*, además de valor sigma para las Gaussianas de 0.125. Estos parámetros son los mismos que los desarrollados en el trabajo original del DPDist.

Entrenamiento. Para el módulo de 3DmFV de DPDist, se usaron los parámetros de $K = 8, k = 5$, sigma Gaussiana = 0.125. Durante el entrenamiento de la red neuronal (ver figura 21) se minimizó la función de costo J de la ecuación (61):

$$J = \frac{1}{|S_B|} \sum_{x_i \in S_B} \|D_{SPD}(x_i, S_A) - GT(x_i)\|, \tag{61}$$

donde S_A y S_B son las dos nubes de puntos, $D_{SPD}(x_i, S_A)$ es la salida de la red (predicción), es decir la distancia DPDist del punto x_i a la nube S_A, y $GT(x_i)$ es la distancia real asociada cada punto x_i en cuestión.

Se entrenaron dos versiones de modelos. El primero con *N*=512 puntos directamente y el segundo con *N*=1024 puntos divididos en 16 lotes o “batch” de 64 puntos cada uno. Ambas

versiones fueron entrenadas con optimizador ADAM, tasa de aprendizaje l_r = 0.0001, momentum m = 0.9 y 10000 épocas.

Los resultados del entrenamiento de DPDist fueron los siguientes, (ver tabla 2, figura 22):

Tabla 2. Resultados entrenamiento DPDist.

Modelo	N° de Épocas	Tiempo Total de Entrenamiento	Pérdida/Error Mínimo (entrenamiento)	Pérdida/Error Mínimo (test)
Versión 1	10000	≈ 60 horas, 10 minutos, 47 segundos	≈ 11.317 %	≈ 5.320 %
Versión 2		≈ 21 horas, 8 minutos, 8 segundos	≈ 18.428 %	≈ 12.985 %

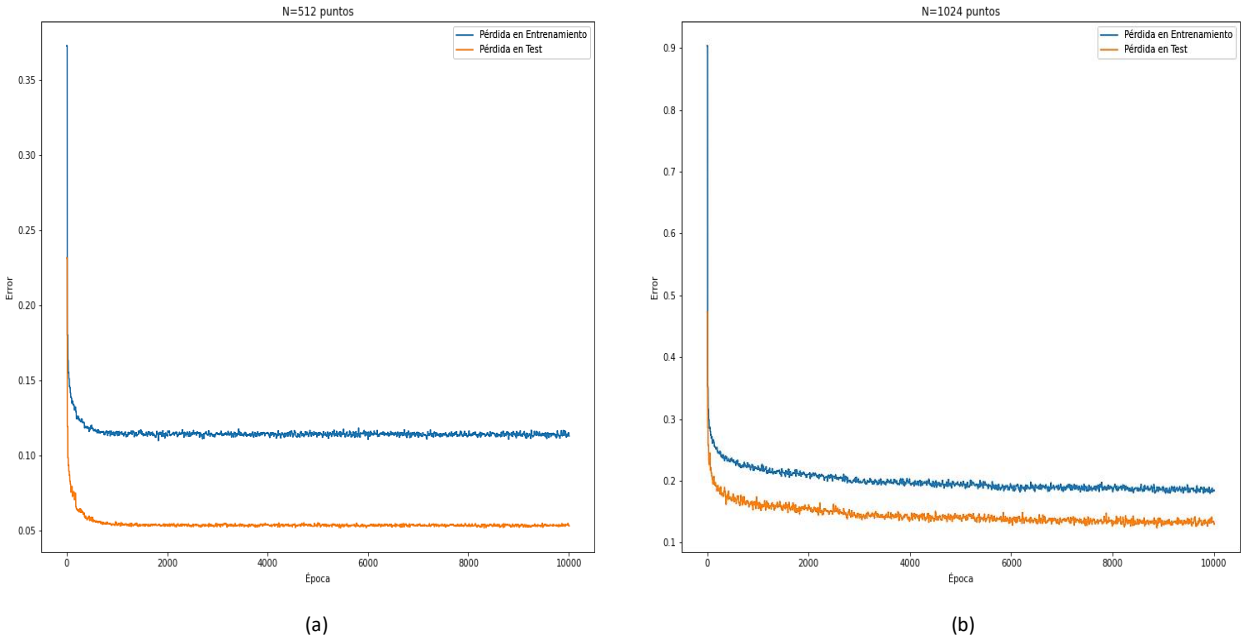

Figura 22. Histórico de error en entrenamiento y test a lo largo del entrenamiento. (a) Entrenamiento y test para 512 puntos y (b) entrenamiento y test para 1024 puntos.

Cada modelo entrenado, por defecto está configurado para guardarse en tres archivos de tipo checkpoint (.ckpt):

- Archivo meta: Aloja la estructura del grafo.

- Archivo índex: Aloja una tabla inmutable donde cada clave es un nombre de un tensor y su valor describe los metadatos de dicho tensor.
- Archivo data: Aloja una colección con los valores de todas las variables.

4.2.5. Pruebas al Modelo Entrenado

Posterior al entrenamiento, se cargó el modelo para probarlo con nubes de puntos diferentes a la categoría con la que fue entrenado, se obtuvieron resultados favorables de comparación de puntos. Se realizaron experimentos de rotación, traslación y discriminación entre clases. Sea O una nube de puntos cualquiera de una clase de ModelNet40 de $N = 5000$ puntos. Sean SA y SB dos muestras aleatorias de O, cada una de $N = 1024$ puntos (20.48% de la muestra original) (ver figura 23).

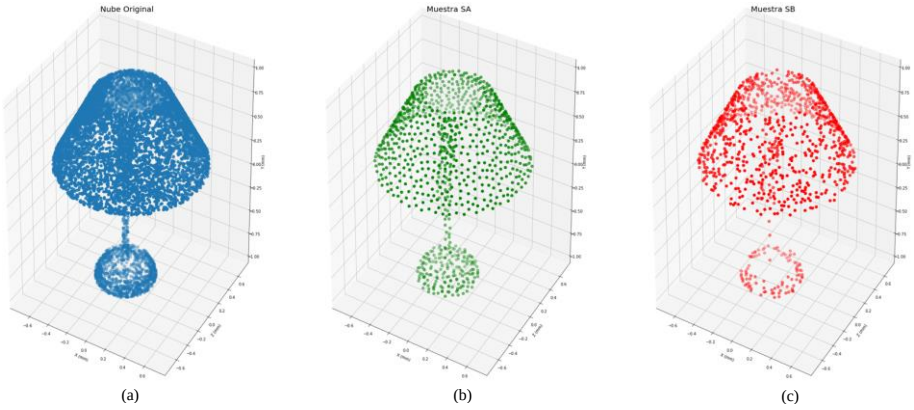

Figura 23. (a) Nube de puntos original O, (b) submuestreo SA, (c) submuestreo SB de la clase "lamp" de ModelNet40

Se calculó la métrica de comparación $D_{DPDist}(SA, SB)$ entre nubes de puntos mediante la ecuación (60), los resultados se representan mediante el mapa de calor de la figura 24:

$$D_{DPDist}(SA, SB) = 0.0957\,.$$

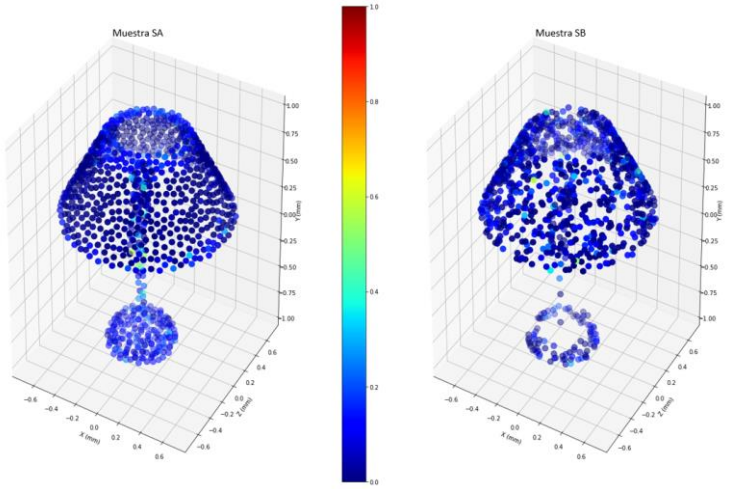

Figura 24. Comparación entre las muestras SA y SB en base al mapa de calor representado mediante la salida de DPDist.

4.2.5.1. Prueba de Traslación. Se le sumó un valor de traslación aleatorio a SB y se volvió a calcular $D_{DPDist}(SA, SB_t)$ mediante la ecuación (60), siendo SB_t la nube de puntos trasladada t unidades en las tres dimensiones). Los resultados se representan mediante el mapa de calor de la figura 25:

$$D_{DPDist}(SA, SB_t) = 0.4408\,.$$

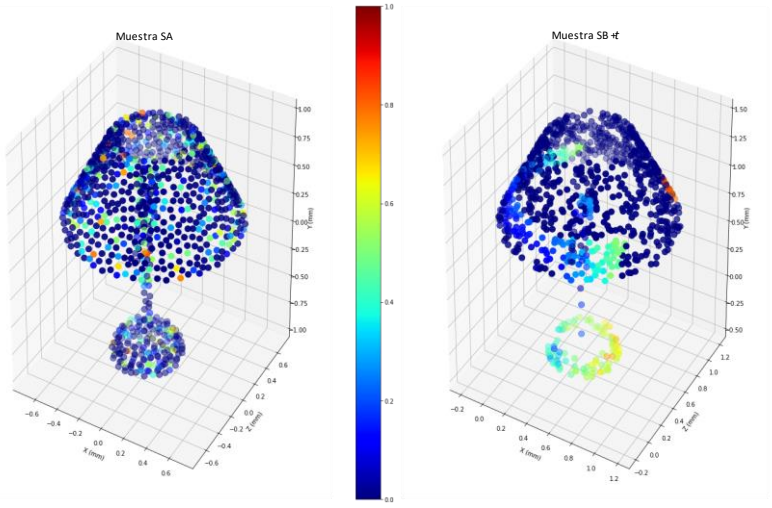

Figura 25. Comparación entre las muestras SA y SB_t en base al mapa de calor representado mediante la salida de DPDist.

4.2.5.2. Prueba de Rotación. Se le aplicó una rotación aleatoria a SB y se volvió a calcular $D_{DPDist}(SA, SB_r)$, siendo SB_r la nube de puntos rotada r radianes en un eje coordenado aleatorio). Los resultados se representan mediante el mapa de calor de la figura 26:

$$D_{DPDist}(SA, SB_r) = 0.2640\,.$$

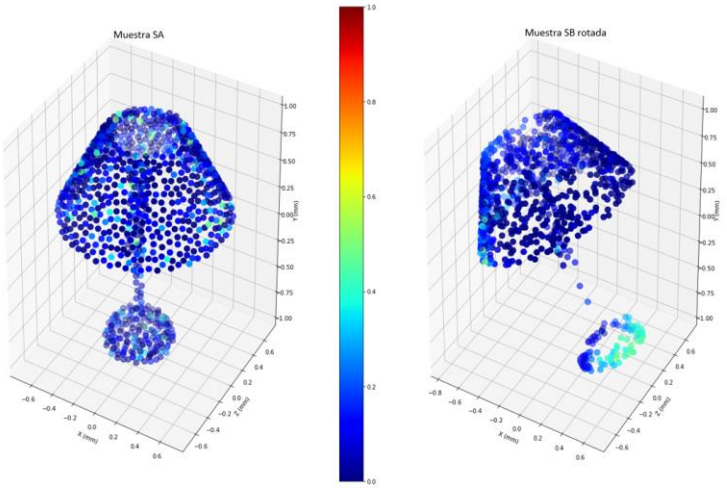

Figura 26. Comparación entre las muestras SA y SB_r en base al mapa de calor representado mediante la salida de DPDist.

A partir de lo anterior, se determinó que para poder comparar dos nubes de puntos mediante DPDist, se debe eludir el factor de traslación centrando las dos nubes en el origen. Esto se debe a que, en las pruebas realizadas, se encontró que la traslación influye drásticamente en el resultado del DPDist y en realidad no es un factor determinante al momento de comparar dos nubes de puntos por su estructura. El factor de rotación, aunque influye un poco en el resultado, no es un parámetro que se pueda evitar como con la traslación. Además, se logró corroborar la desigualdad descrita por la ecuación (62):

$$D_{DPDist}(SA, SB) < D_{DPDist}(SA, SB_t),\ D_{DPDist}(SA, SB_r)\,. \qquad (62)$$

4.2.5.3. Pruebas de discriminación entre clases. Mediante esta prueba se pretende comparar dos modelos de nubes de puntos distintos, a) de la misma categoría A_{c1}, B_{c1} (ver figura 27), y b) de categorías diferentes A_{c1}, B_{c2} (ver figura 28).

a) $D_{DPDist}(A_{c1}, B_{c1}) = 0.2975$

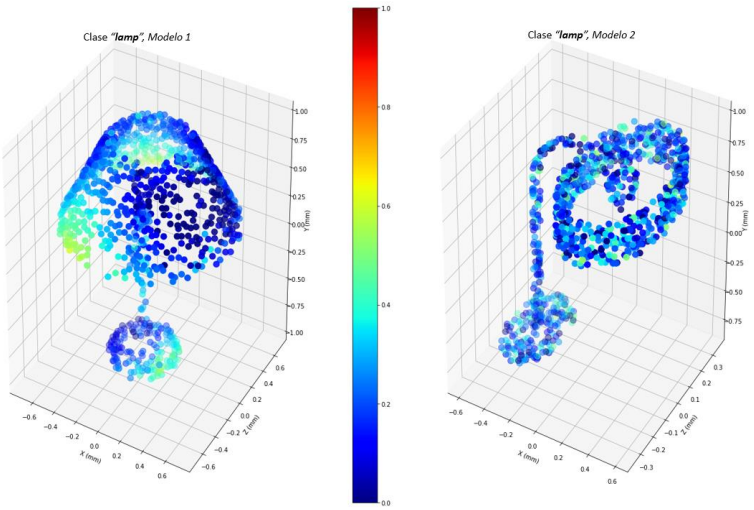

Figura 27. Comparación entre las nubes de puntos A_{c1} y B_{c1} de la clase "lamp" en base al mapa de calor representado mediante la salida de DPDist.

b) $D_{DPDist}(A_{c1}, B_{c2}) = 0.6978$

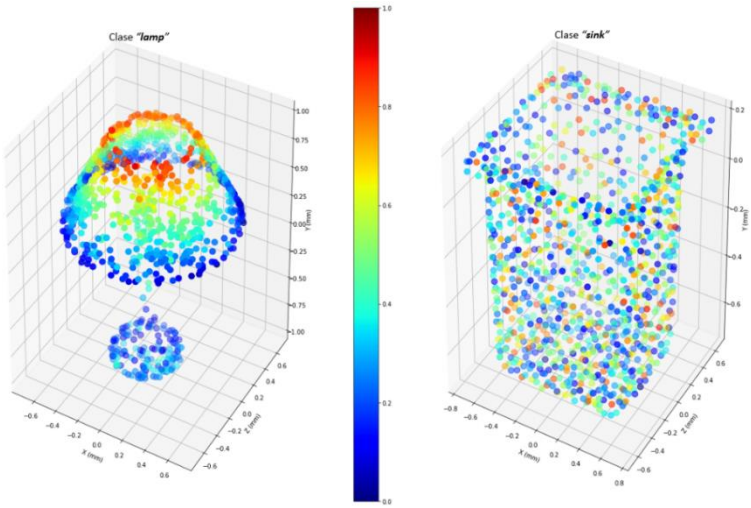

Figura 28. Comparación entre las nubes de puntos de la clase "lamp" A_{c1} y la clase "sink" B_{c2} en base al mapa de calor representado mediante la salida de DPDist.

Como era de esperarse, se obtuvo que, para modelos de la misma categoría, el DPDist es menor que para modelos de categorías diferentes, siguiendo la desigualdad descrita por la ecuación (63):

$$D_{DPDist}(A_{c1}, B_{c1}) < D_{DPDist}(A_{c1}, B_{c2})\,. \tag{63}$$

La figura 29 muestra las pruebas definidas anteriormente realizadas para otras categorías del conjunto de datos ModelNet40 (columnas). La primera fila corresponde a las pruebas de DPDist entre dos muestras de la misma nube de puntos. La segunda, corresponde a la prueba de DPDist entre una muestra fija y una trasladada aleatoriamente t unidades. La tercera fila es la prueba de DPDist entre una muestra fija y una muestra rotada aleatoriamente r radianes. La última fila corresponde a la prueba de DPDist con dos modelos distintos, pero de la misma categoría o clase. En todas las pruebas realizadas, se cumplen las ecuaciones (62) y (63).

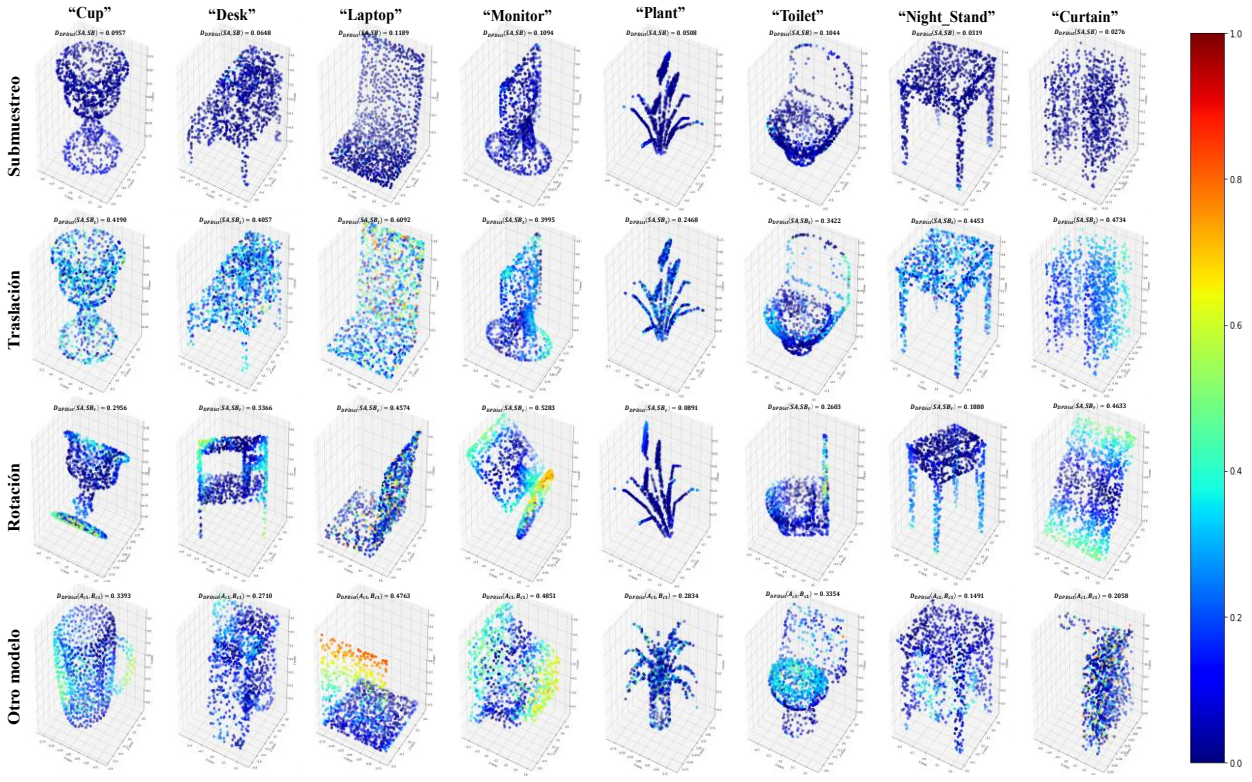

Figura 29. Pruebas de rotación, traslación y discriminación de clases realizadas a otros modelos de ModelNet40.

En base a lo anterior, se plantearon dos alternativas para resolver el problema de registro de nubes de puntos usando técnicas de Aprendizaje Profundo. La primera, mediante una Red Neuronal Profunda estimar la rotación implícita entre dos nubes de puntos comparadas DPDist. La segunda, encontrar la correspondencia, coincidencia o matching de puntos entre las dos nubes a registrar empleando la métrica de comparación DPDist y posteriormente usar la descomposición de valores singulares SVD para estimar la matriz de transformación implícita entre ellas.

4.3. Primera Alternativa

Como en la primera aproximación, se propuso hacer una red neuronal para predecir el ángulo de rotación implícito entre dos nubes de puntos, usando esta vez el resultado de comparación DPDist, usando la ecuación (60) entre las nubes de puntos, en lugar de las nubes de puntos directamente como entrada a la red. Esto con el fin de generalizar el problema de predicción a cualquier par de nubes que se deseen registrar.

4.3.1. Pruebas de rotación independientes para cada eje

Sea S_A y S_B dos nubes de puntos muestreadas aleatoriamente de un modelo de ModelNet40. En primera instancia se realizaron pruebas para observar el comportamiento del DPDist al rotar la nube de puntos S_B de 0 a 2π radianes cada $\frac{\pi}{15}$ radianes en cada eje coordenado con respecto a la nube de puntos fija S_A. Tales rotaciones fueron efectuadas usando los Ángulos de Euler (Diebel, 2006) mediante las ecuaciones (64), (65) y (66):

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}, \tag{64}$$

$$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix}, \quad (65)$$

$$R_z(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0) \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (66)$$

Además, se normalizó la métrica DPDist entre el rango de valores de [0, 1] con el fin de obtener resultados más congruentes independientemente de las nubes de puntos a comparar. Los resultados para el modelo “Guitar” fueron los siguientes (ver figura 30):

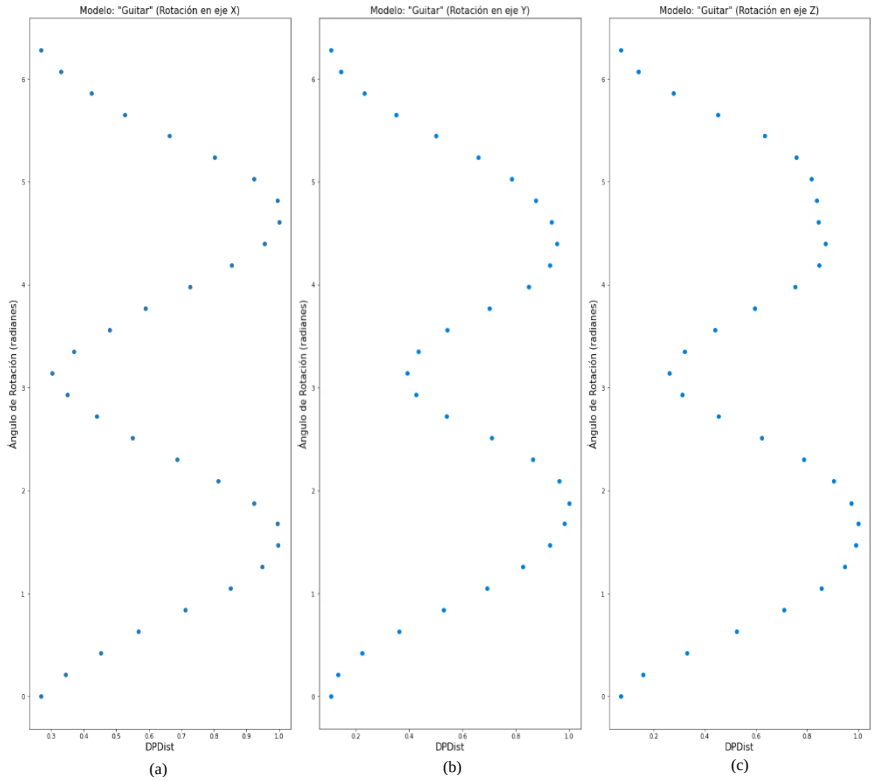

Figura 30. Prueba de DPDist vs Rotaciones para las nubes de puntos del modelo “Guitar”. S_A fija y S_B rotada independientemente cada $\pi/15$ radianes para (a) eje X, (b) eje Y, (c) eje Z.

Posteriormente se realizó la misma prueba de rotación con otras clases de objetos, los resultados fueron los siguientes (ver figura 31):

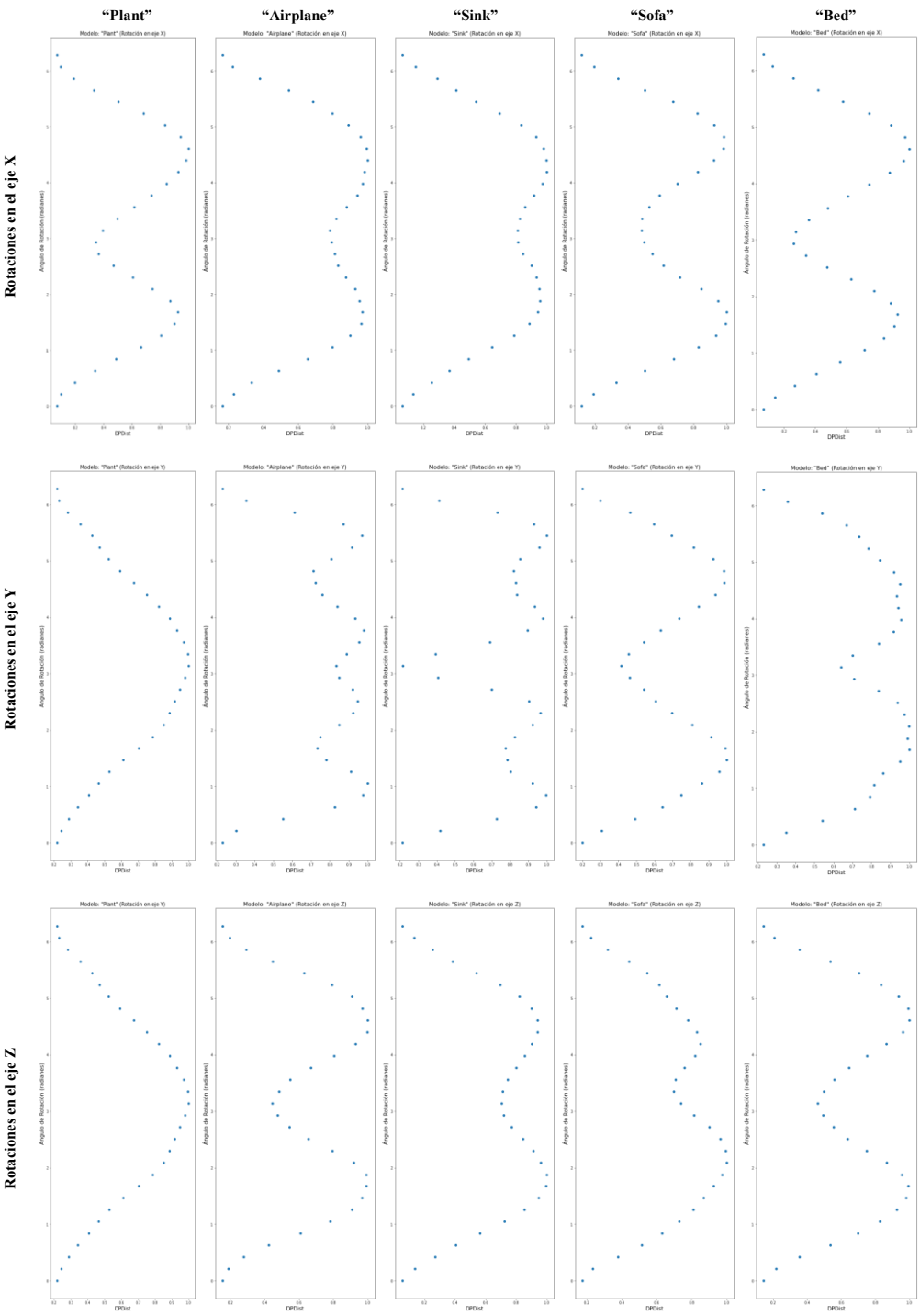

Figura 31. Pruebas DPDist vs Rotaciones nubes de puntos de otras categorías de ModelNet40 cada $\pi/15$ radianes para los ejes [x, y, z].

Posterior a las rotaciones independientes para cada ángulo, se procedió a generar rotaciones combinadas en los 3 ejes coordenados para observar el comportamiento de DPDist.

4.3.2 Pruebas de rotación combinadas en los tres ejes

Tal como en las pruebas anteriores, se tomó un modelo aleatorio para observar el comportamiento del DPDist vs la rotación entre las nubes a registrar, pero en esta ocasión, se realizaron pruebas de rotaciones combinadas en los tres ejes coordenados X, Y, Z a dos nubes rotadas previamente.

Se muestrearon dos conjuntos de puntos disjuntos, Nube A y Nube B, a partir del mismo modelo y se les aplicó una rotación inicial aleatoria en los tres ejes a ambas nubes. Para realizar las rotaciones combinadas se usaron las ecuaciones (67 a 72):

$$R_{xyz} = R_x \cdot R_y \cdot R_z \,, \tag{67}$$

$$R_{xzy} = R_x \cdot R_z \cdot R_y \,, \tag{68}$$

$$R_{yxz} = R_y \cdot R_x \cdot R_z \,, \tag{69}$$

$$R_{yzx} = R_y \cdot R_z \cdot R_x \,, \tag{70}$$

$$R_{zxy} = R_z \cdot R_x \cdot R_y \,, \tag{71}$$

$$R_{zyx} = R_z \cdot R_y \cdot R_x \,, \tag{72}$$

siendo R_x, R_y, R_z rotaciones independientes en cada eje calculadas mediante las ecuaciones (64), (65) y (66) respectivamente.

Las rotaciones combinadas definidas anteriores tienen como resultado nubes de puntos en posiciones diferentes, debido a que el producto punto, a pesar de ser conmutativo, no es asociativo, es decir, $(R_x \cdot R_y) \cdot R_z \neq R_x \cdot (R_y \cdot R_z)$, de aquí que rotaciones como R_{xyz} tenga resultados diferentes de R_{yzx} para valores iguales de R_x, R_y, R_z.

A partir de las dos nubes iniciales rotadas aleatoriamente con los ángulos de la tabla 3 (ver figura 32), se realizaron experimentos de DPDist vs Rotaciones en diferente orden.

Tabla 3. *Ángulos de rotación definidos para las nubes de puntos A y B.*

	Nube A [rad]	**Nube B [rad]**
Eje X	2.3337	-2.1061
Eje Y	-5.1679	1.2837
Eje Z	0.7299	5.2069

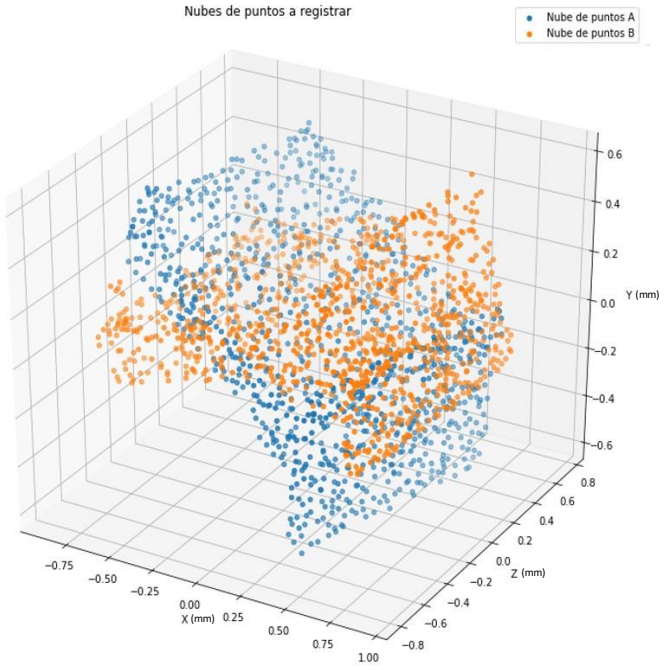

Figura 32. Nubes de puntos iniciales de la clase "Bed" con rotaciones aleatorias en el orden ZYX definidas en la tabla.

- **Rotaciones en el orden ZYX**

En este experimento, se realizan pruebas de DPDist vs Rotaciones cada $\frac{\pi}{15}$ radianes en el orden de rotación Z, Y, X. Se espera que el punto correspondiente al ángulo donde DPDist tome el menor valor, sea aquel cuya rotación debe aplicarse para obtener un registro temporal en ese eje. El resultado en la predicción del eje Z es el siguiente (ver figura 33):

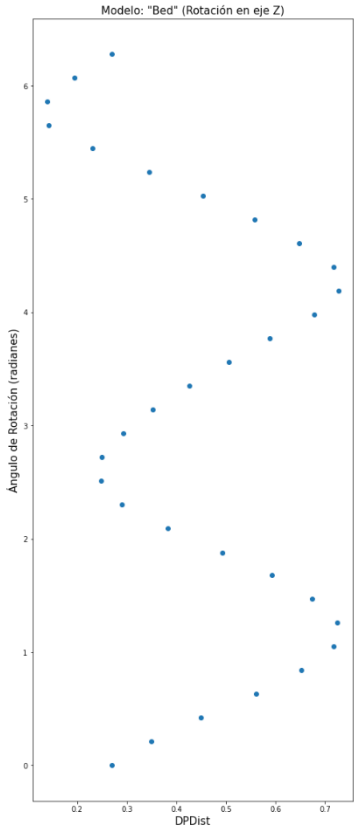

Figura 33. Resultado DPDist vs Rotaciones para el modelo “Bed” en el eje Z.

Posteriormente se le aplicó la rotación a la Nube *B* usando el ángulo hallado en Z y se procedió a comparar esta nueva nube rotada con la Nube *A* mediante la prueba DPDist vs rotaciones en el eje *Y* (ver figura 34 (a)). Una vez hallado el ángulo de rotación donde DPDist es menor, se le aplica la rotación en el eje *Y* a la nueva nube y se procede a hacer el mismo proceso, esta vez en el eje restante X (ver figura 34 (b)).

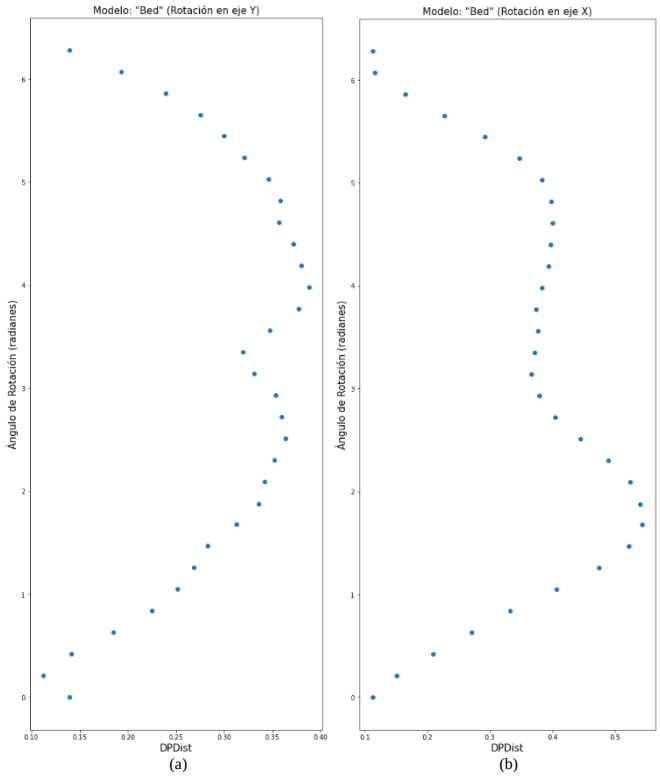

Figura 34. Resultado DPDist vs Rotaciones para el modelo “bed” en (a) eje Y, (b) eje X.

El resultado de las rotaciones con los 3 ángulos anteriormente predichos, y el consecuente registro entre las nubes de puntos se ve en la figura 35:

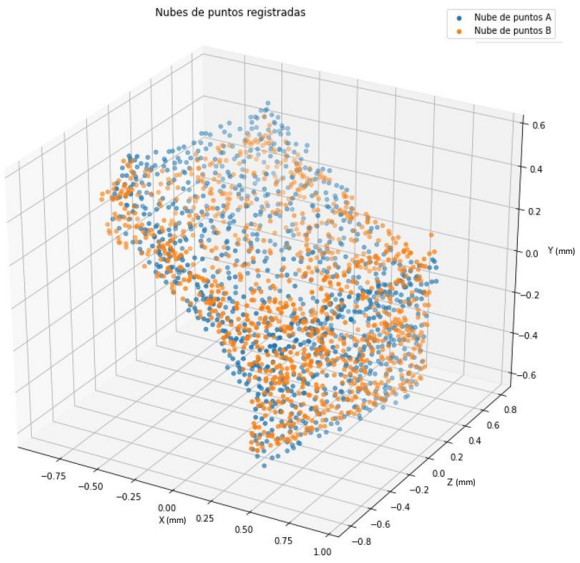

Figura 35. Nubes de puntos A y B registradas a partir de las rotaciones.

- **Rotaciones en diferente orden**

En base a la descripción anterior de la prueba para rotaciones en el orden ZYX, a continuación, se presentan las pruebas para el registro de las mismas nubes de la figura 32 usando diferente orden de rotación en los ejes (ver figuras 36, 37, 38, 39, 40).

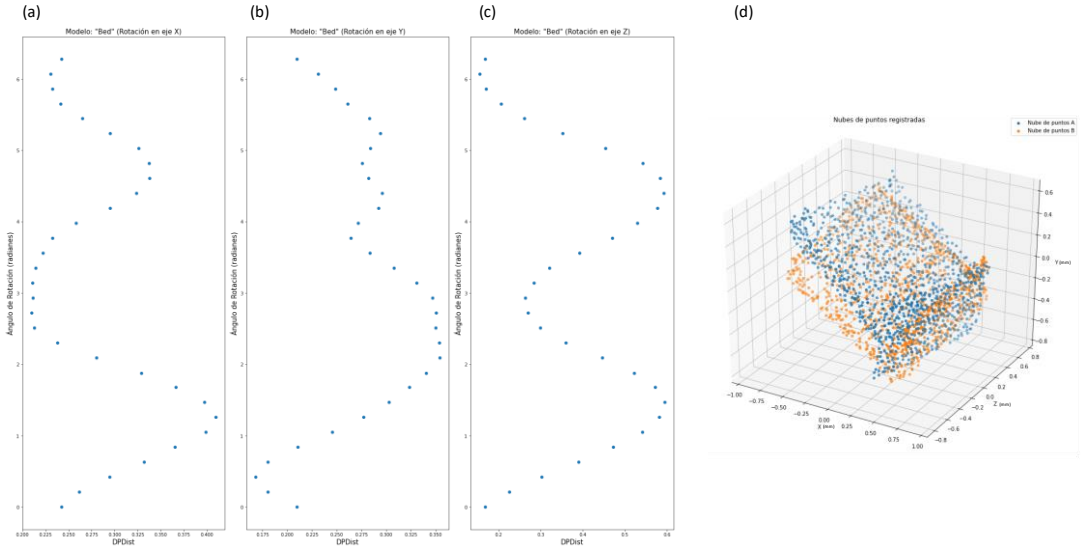

Figura 36. (a) DPDist vs Rotaciones en el eje X, (b) DPDist vs Rotaciones en el eje Y, (c) DPDist vs Rotaciones en el eje Z, (d) Registro de las nubes A y B para rotación en el orden XYZ.

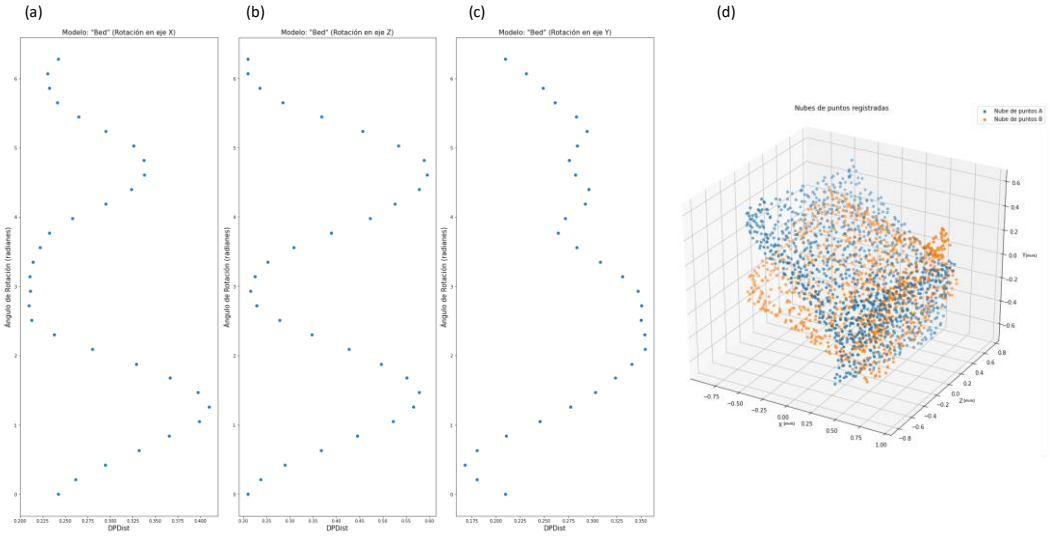

Figura 37. (a) DPDist vs Rotaciones en el eje X, (b) DPDist vs Rotaciones en el eje Z, (c) DPDist vs Rotaciones en el eje Y, (d) Registro de Nube A y Nube B para rotación en el orden XZY.

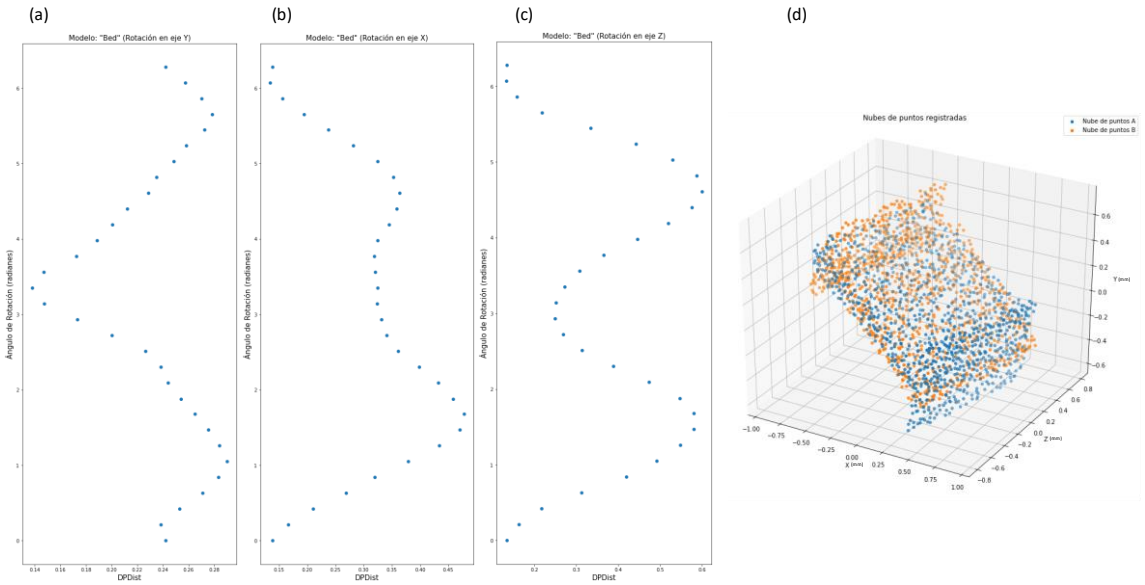

Figura 38. (a) DPDist vs Rotaciones en el eje Y, (b) DPDist vs Rotaciones en el eje X, (c) DPDist vs Rotaciones en el eje Z, (d) Registro de Nube A y Nube B para rotación en el orden YXZ.

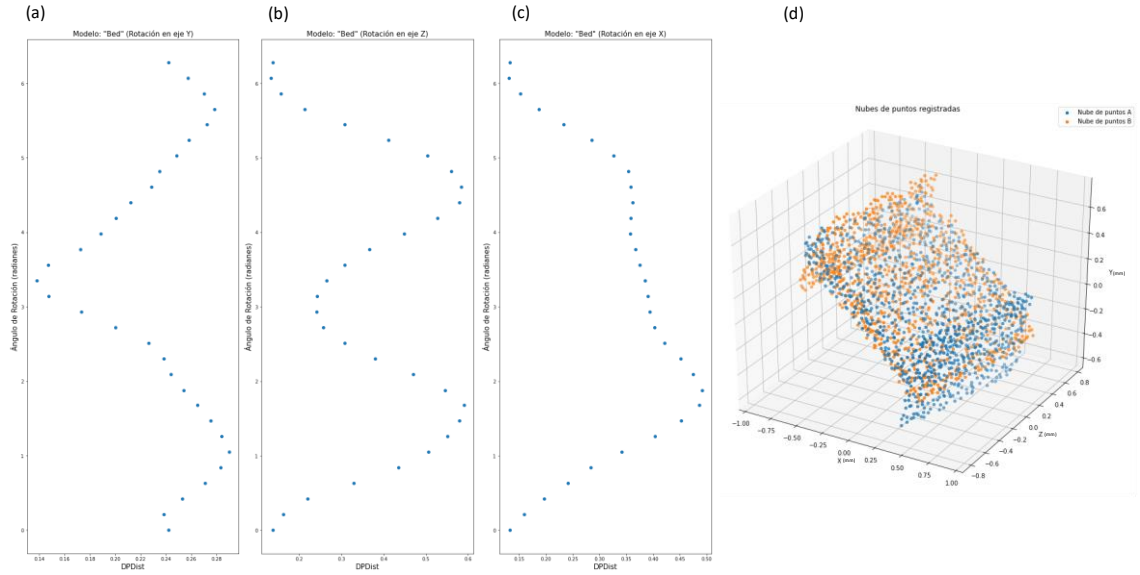

Figura 39. (a) DPDist vs Rotaciones en el eje Y, (b) DPDist vs Rotaciones en el eje Z, (c) DPDist vs Rotaciones en el eje X, (d) Registro de Nube A y Nube B para rotación en el orden YZX.

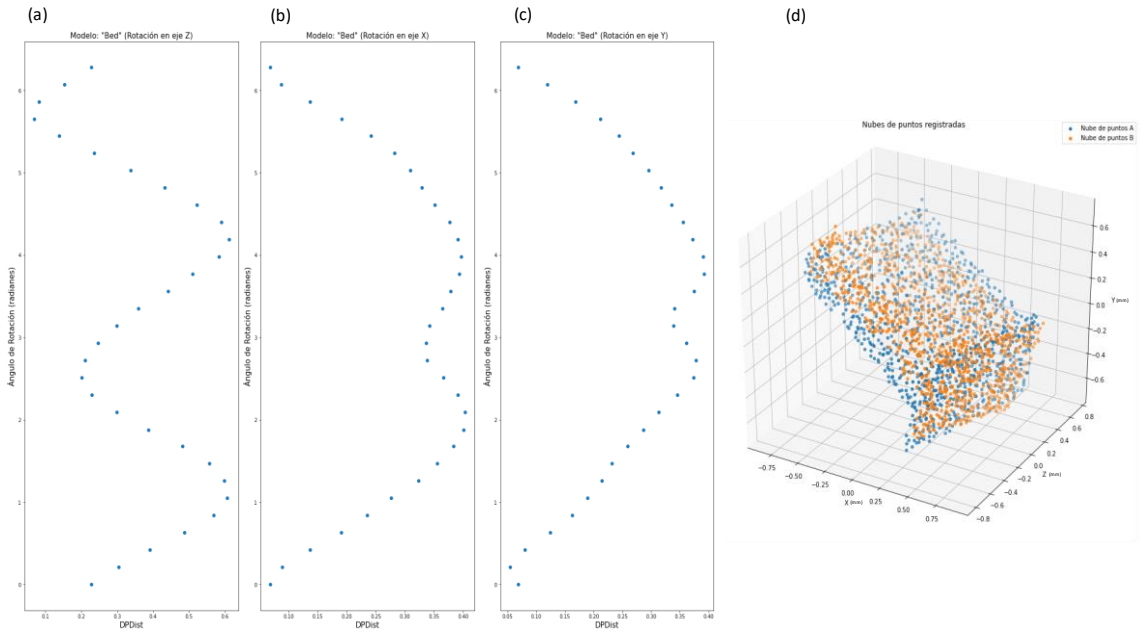

Figura 40. (a) DPDist vs Rotaciones en el eje Z, (b) DPDist vs Rotaciones en el eje X, (c) DPDist vs Rotaciones en el eje Y, (d) Registro de Nube A y Nube B para rotación en el orden ZXY.

Se realizaron las mismas pruebas para otras categorías de objetos y se obtuvieron resultados congruentes con los anteriormente mostrados. Se observa que, con la prueba de DPDist vs Rotaciones, es posible realizar un buen registro entre dos nubes rotadas en los tres ejes solo si se hace en el mismo orden en el que se les aplicaron las rotaciones a las nubes al principio *ZYX*, de lo contrario, los resultados no son aceptables. Esto se traduce en un problema, ya que, en el mundo real, se desconoce el orden de rotación implícito entre dos nubes.

A partir de las pruebas anteriores, se corroboró que:

1. DPDist es sensible a las rotaciones abruptas.
2. Entre menor sea DPDist, las nubes de puntos son más similares.
3. A pesar de que DPDist fue entrenado con modelos de la categoría “chair”, este funciona para otros modelos, es decir, es independientemente de la clase o categoría de los modelos a comparar.

4. Para rotaciones de π radianes o 180°, DPDist tiende a desacertar en algunos casos. Esto se debe a que para ese ángulo específico se da el caso de reflexión espacial que provoca equivocaciones en la métrica de comparación.

En base a lo anterior, se infiere que predecir la rotación implícita entre dos nubes de puntos comparadas con DPDist, mediante el uso de una Red Neuronal es complicado por las siguientes razones:

1. Se requiere una alta capacidad computacional para generar todas las combinaciones de rotaciones posibles en los tres ejes para producir un conjunto de datos lo suficientemente robusto para alimentar una Red Neuronal.
2. En caso de lograr superar el ítem anterior, como se pretende calcular un ángulo de rotación implícito para cada uno de los tres ejes coordenados, el orden de estas rotaciones es fundamental y no es un parámetro que se pueda predecir fácilmente mediante una Red Neuronal Profunda.
3. Los datos obtenidos del comportamiento de DPDist vs Rotaciones no son sólidos para alimentar una Red Neuronal que estime la rotación implícita entre dos nubes comparadas con DPDist porque esta métrica es variante a la rotación y se dan casos que para un valor DPDist corresponden varios ángulos de rotación. Para solucionar esto, se debería modificar en sí la métrica de comparación DPDist para hacerla independiente al grado de rotación entre las nubes de puntos y así tener datos únicos para cada ángulo.

Debido a lo anteriormente planteado, se procedió a trabajar en la segunda alternativa: Encontrar la correspondencia de puntos usando DPDist.

4.4. Segunda Alternativa: Correspondencia de puntos con gradientes

Se propuso usar la métrica de comparación DPDist entre las dos nubes de puntos a registrar para establecer el "matching" o correspondencia de puntos y posteriormente mediante el uso de SVD (como lo hace ICP), generar la matriz de rotación implícita entre los dos conjuntos de puntos cuyos puntos son coincidentes.

Tal como se describe en la sección de DPDist, este modelo usa la representación local para establecer la relación entre puntos y así calcular la "distancia" de comparación. Es decir, DPDist, implícitamente encuentra la correspondencia de un punto de la nube S_A con su propio o más afín en la superficie subyacente denotada de la nube de puntos S_B. En base a lo anterior, se plantea obtener el vector dirección a partir del gradiente en cada punto usando la retropropagación del DPDist, que es un modelo de tipo grafo codificado en TensorFlow, cuyos nodos representan cada una de las operaciones y cuyas conexiones entre nodos representan los conjuntos de datos multidimensionales o tensores.

Para calcular los gradientes a lo largo de un modelo basado en TensorFlow como DPDist, generalmente se usa la diferenciación automática, principio en el cual se basa el algoritmo de retro propagación de una Red Neuronal para su aprendizaje.

En el proceso de diferenciación automática, primero se recorre el grafo en forward para guardar los valores de cada variable y las dependencias entre nodos, y obtener la predicción o salida de la red. Posteriormente se hace un paso de backward o propagación hacia atrás, empezando por dicha salida, a partir de la función de costo y calculando las derivadas de cada variable respecto a las demás variables que le anteceden. De esta manera, el valor de la derivada de un nodo puede verse como la contribución de dicho nodo al cambio del siguiente. El proceso termina cuando se alcanzan los nodos que contienen los datos de entrada (nubes de puntos),

obteniendo el vector gradiente para hallar la dirección de las correspondencias de cada punto en la nube.

Este procedimiento es posible debido a que, como el modelo intrínsecamente localiza los puntos cuyo valor de distancia DPDist es mínimo, al realizar la diferenciación automática entre la predicción y los datos de entrada, se obtiene el valor del gradiente que apunta hacia la dirección en la que el DPDist se minimiza para cada punto, que en otras palabras, es la posible coincidencia los puntos de la nube sobre la superficie subyacente generada por la otra nube, ya que la función de gradiente en distancia (salida del DPDist) es la dirección al punto más cercano en la superficie.

Mediante el vector calculado con los gradientes, se debería poder establecer el *matching* entre puntos de las dos nubes., se usó la función tf.gradients() para obtener de forma automática el gradiente de DPDist.

Esta función de Tensorflow no solo calcula el gradiente en la retro propagación de una red, sino que, siempre que los datos estén almacenados en nodos, puede efectuar derivadas simbólicas para hallar la función que convierte los datos desde el nodo de salida al de entrada, teniendo en cuenta la función adscrita a los dos nodos. Esto se ve traducido en la capacidad de convertir valores de gradientes expresados en las dimensiones del 3DmFV a el vector gradiente que se necesita para determinar las correspondencias.

Sea $\vec{v}_{grad}$ el vector obtenido con la retropropagación de la red DPDist, S_A y S_B dos nubes de puntos muestreadas de manera diferente. Sean a_i y b_j dos puntos en la posición i y j en S_A y S_B respectivamente, con $i, j = 1, 2, \dots, N$, siendo N el número total de puntos en cada nube. Teniendo en cuenta que como S_A y S_B están desordenadas y su correspondencia de puntos se

desconoce, el punto i en la nube S_A no es el correspondiente al punto j en la nube S_B. Por ejemplo, a_1 no es el mismo punto que b_1.

4.4.1. Obtención del gradiente mediante tf.gradients()

La función $tf.gradients(Y, X)$ realiza el proceso de diferenciación automática construyendo derivadas simbólicas de la suma de $\frac{dY}{dX}$ para cada x en X. Esta función se usa en contextos de grafos y es capaz de agregar operaciones al grafo para generar las derivadas de Y con respecto a X (tf.gradients, 2022).

Para el cálculo del gradiente que permita encontrar la correspondencia de puntos, se usaron como parámetros de la función $tf.gradients$ los tensores de entrada y salida del modelo DPDist,

$$grads = tf.gradients(t_{salida}, t_{entrada}),$$

siendo t_{salida} el tensor conformado por las predicciones de distancias de comparación (DPDistAB y DPDistBA), y $t_{entrada}$ el tensor conformado por las dos nubes de puntos S_A y S_B a comparar mediante DPDist.

Como resultado del cálculo de $grads$, se obtiene un arreglo de tamaño [2, N, 3] que contiene tanto los gradientes de DPDistAB como los de DPDistBA con respecto a las dos nubes de puntos de entrada.

Cabe anotar que, para usar esta función, se debió hacer un estudio y análisis a profundidad del grafo del modelo DPDist para establecer las relaciones entre nodos y tensores, incluidos los de las capas de la red neuronal, ya que sin una conexión nodo a nodo constante y coherente desde la entrada hasta la salida, no es posible el cálculo del gradiente mediante este método.

4.4.2. Correspondencia o coincidencia de puntos

Se procedió a estimar la correspondencia entre puntos de dos nubes S_A y S_B pertenecientes a ModelNet40. Para realizar este proceso se siguieron los pasos descritos a continuación:

1. Se ingresaron las dos nubes de puntos (entrada) a DPDist y se obtuvieron las distancias de comparación (salida) y el error de predicción.
2. Se estimaron los gradientes en cada punto mediante la diferenciación automática desde la salida hasta la entrada de DPDist.
3. Se sumaron los gradientes hallados para cada punto tanto a S_A como a S_B. Como la salida D_{SPD} de DPDist es un arreglo que contiene las distancias de comparación tanto de S_A a S_B (DPDistAB), como de S_B a S_A (DPDistBA), el vector de dirección $\vec{v}_{grad}$ en realidad, también es un arreglo que estará compuesto por la concatenación de dos vectores $\vec{v}_1$ y $\vec{v}_2$ que serán las direcciones para llegar desde el punto a_i a su correspondiente más cercano b en la superficie de S_B y viceversa (ver figura 41).

   Como DPDist genera una superficie subyacente de cada nube, la suma del gradiente hallado para cada punto de S_A y S_B no conduce directamente al coincidente en la otra nube, sino a un espacio subyacente donde se encontrarán los pares de puntos correspondientes de S_A y S_B. Los pares de puntos más cercanos en este nuevo espacio se consideran coincidentes. Para determinar su cercanía se usó la Distancia Euclidiana.

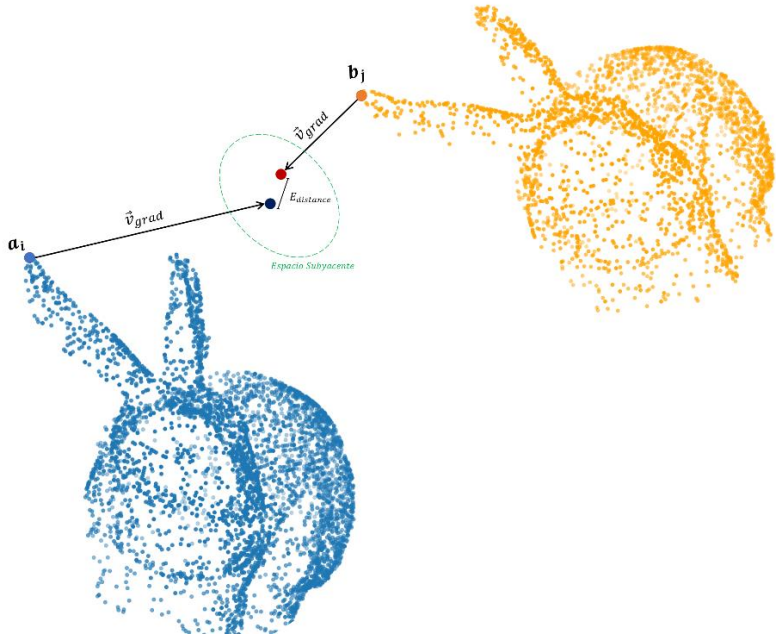

Figura 41. Correspondencia de pares de puntos con la suma de los vectores gradientes.

4. Determinados los puntos con menor distancia y con el fin de reducir el número de correspondencias erráticas o no válidas, se procede a definir un parámetro de correspondencia que cumpla lo anterior para S_A y S_B y para S_B y S_A. De modo que, si $(a_i + \vec{v}_{1_i}) \sim \left(b_j + \vec{v}_{1_j}\right)$ y $(a_i + \vec{v}_{2_i}) \sim \left(b_j + \vec{v}_{2_j}\right)$, a_i es un buen punto candidato a correspondiente con b_j.

Cabe aclarar que, como DPDist no funciona adecuadamente para nubes trasladadas, las nubes a registrar se centran previamente y se registra el valor del movimiento de traslación efectuado para ser computado posteriormente en la matriz de transformación hallada con la descomposición de valores singulares (SVD).

En las figuras 42, 43 y 44 se presentan algunos de los resultados de las pruebas de correspondencia realizadas siguiendo los pasos expuestos anteriormente:

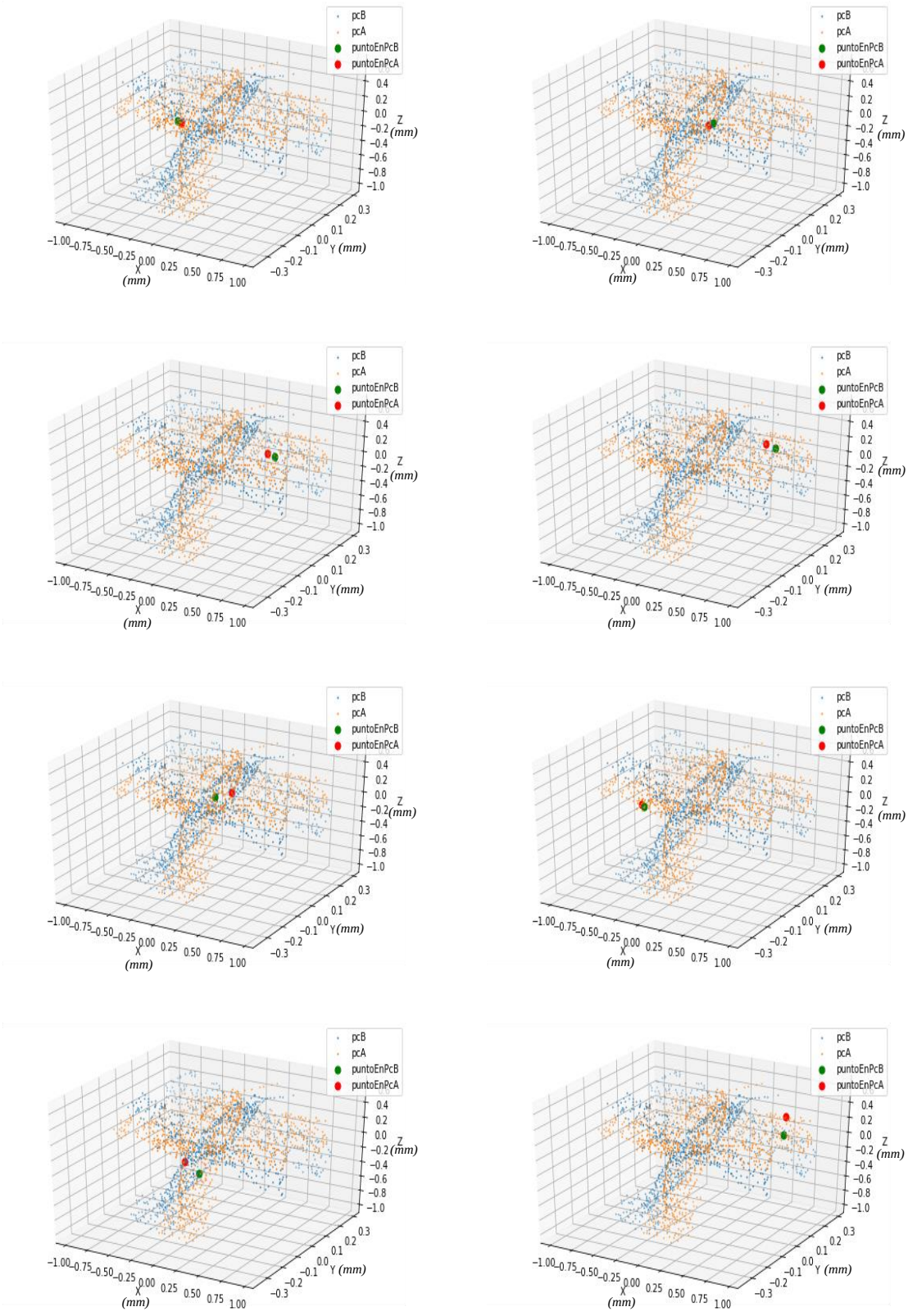

Figura 42. Resultados de correspondencia para un modelo de la categoría “airplane” de ModelNet40.

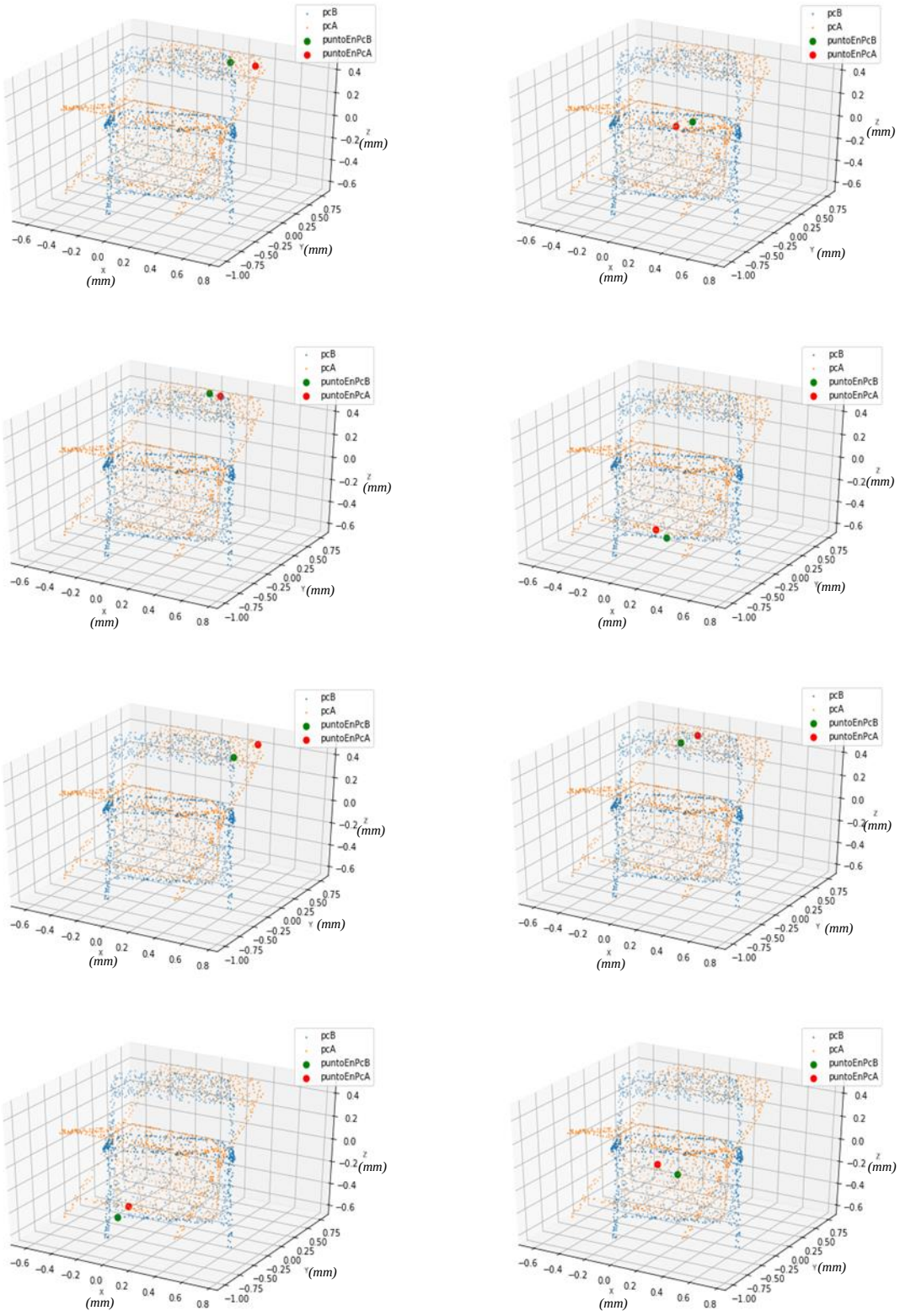

Figura 43. Resultados de correspondencia para un modelo de la categoría “chair” de ModelNet40

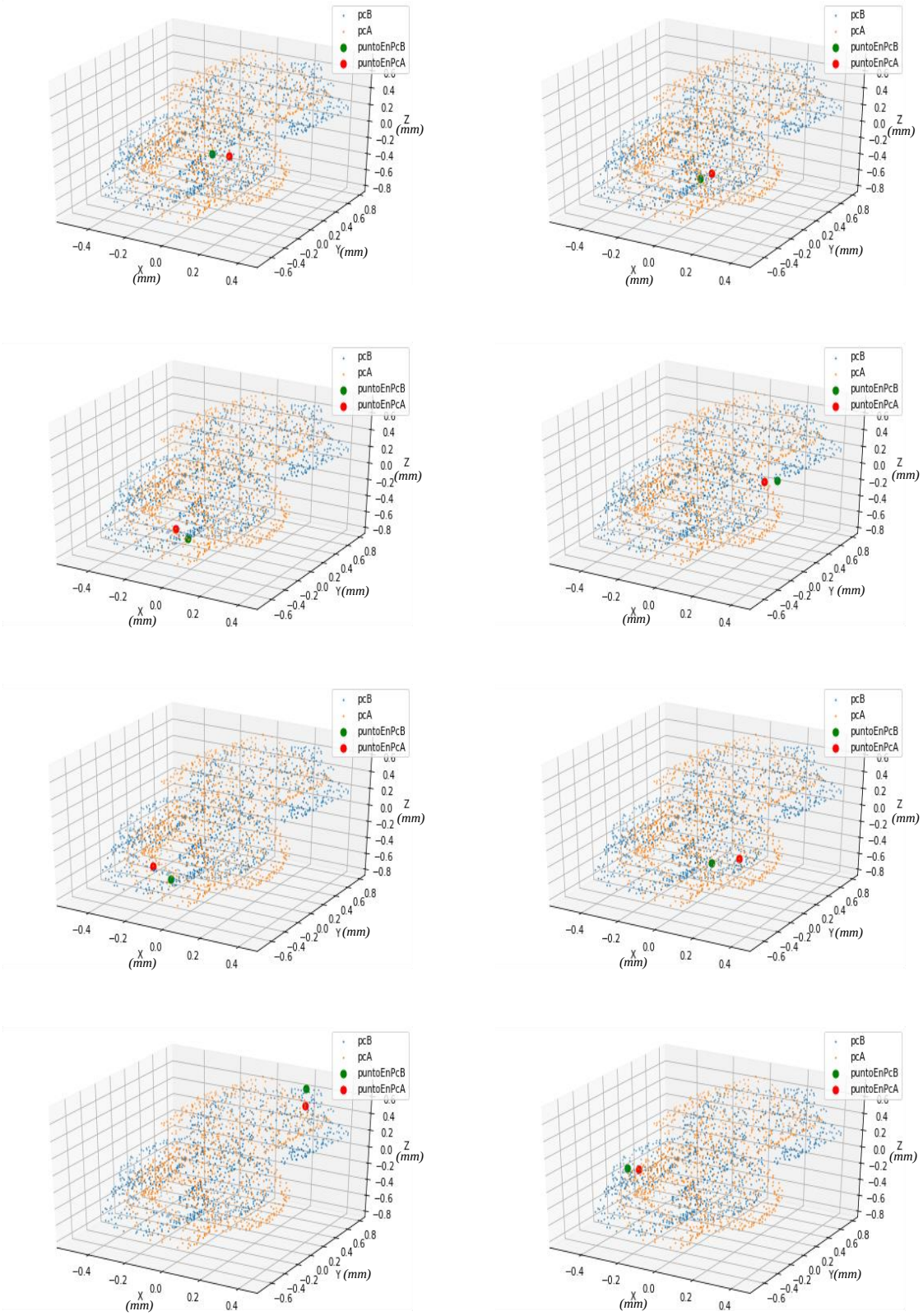

Figura 44. Resultados de correspondencia para un modelo de la categoría “bath” de ModelNet40.

Como se puede observar, a pesar de que hay buenos puntos coincidentes, hay otros que son desacertados y esto se debe precisamente a que como el método de correspondencia se basa en la retropropagación del DPDist, el error de coincidencia de puntos será la misma pérdida de dicho modelo.

4.4.3. Descomposición de Valores singulares

Posterior al cálculo de puntos correspondientes, se propuso usar la SVD para obtener la matriz rígida de transformación, tal como lo hace ICP en la ecuación (47). Para este caso, solo se hallará el parámetro de rotación que se complementará con el valor de traslación hallado al centrar las dos nubes de puntos.

Un factor importante es que mediante experimentaciones realizadas se concluyó que, para el correcto cálculo de la matriz de transformación mediante SVD, se deben emplear al menos cuatro puntos coincidentes ordenados, que cumplan con la condición:

$$a_i \text{ es correspondiente a } b_i, \qquad \forall\ i = \{1, \dots, 4\ o\ N\},$$

siendo a_i un punto de la nube S_A correspondiente al punto b_i de la nube S_B,de lo contrario, la SVD no funcionará adecuadamente y la matriz de rotación hallada será desacertada.

4.4.4. Criterio de parada

El criterio de parada es un aspecto importante del algoritmo, pues, al ser iterativo, fácilmente puede caer en un bucle infinito. Como criterio de parada se estableció un valor de tolerancia para la distancia euclidiana entre los coincidentes hallados, si al sacar un promedio de las distancias halladas de los puntos determinados como coincidentes, es menor que el valor de tolerancia establecido se puede determinar que la nube ya se encontraría registrada o muy cerca de estarlo.

Matemáticamente se definiría este criterio mediante la ecuación (73):

$$distanciaPromedio = \frac{1}{N}\sum_{i=1}^{N} distanciaCoincidentes_i \,, \tag{73}$$

donde N es la cantidad de puntos coincidentes hallados y $distanciaCoincidentes_i$ es la distancia euclidiana entre los *i-pésimos* par de coincidentes hallados.

Teniendo la distancia promedio, comparamos con la *tolerancia* establecida para que, al momento de ser menor la distancia promedio que la tolerancia, se detenga el algoritmo y retorne la matriz de rotación para el registro entre las nubes, es decir,

$$si,\ distanciaPromedio < tolerancia \ \rightarrow\ \ Detener\ el\ algoritmo\ .$$

4.4.4. Resumen del Algoritmo propuesto

Para realizar el registro rígido de dos nubes de puntos tridimensionales muestreadas de forma diferente, con puntos faltantes, reflectados u ocluidos se establecieron los siguientes pasos (ver figura 45) que describen el algoritmo propuesto en base a toda la investigación realizada:

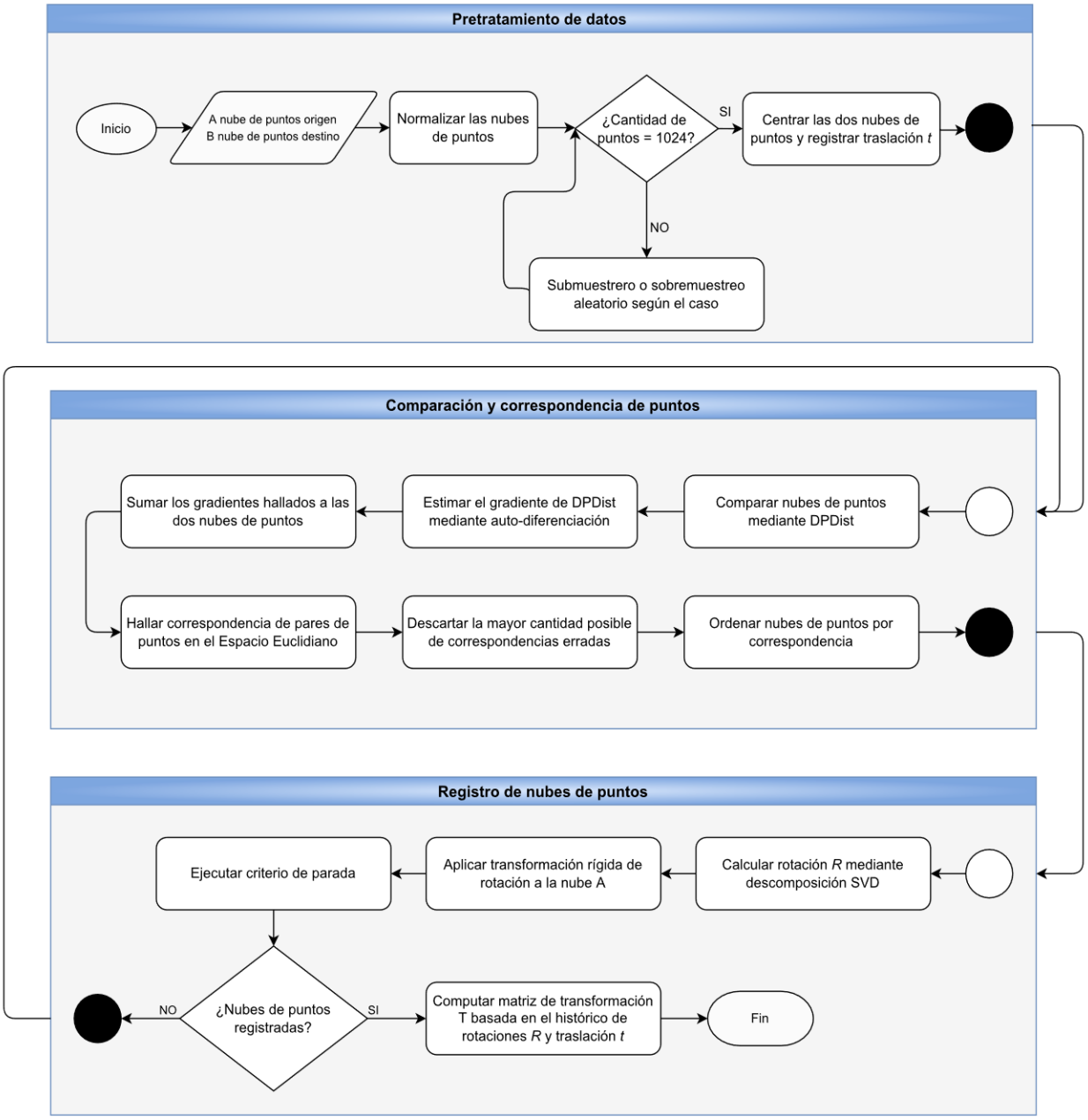

Figura 45. Diagrama de flujo del Algoritmo Propuesto.

A continuación, se muestran algunos experimentos efectuados para probar el algoritmo propuesto en tareas de registro de puntos. Sean pcA y pcB dos muestras diferentes de nubes de puntos del mismo objeto sin correspondencias ni pose inicial conocida.

4.4.4.1. Prueba de rotación. Se pretende calcular una matriz rígida de transformación T que contenga el parámetro de rotación $R \in \mathbb{R}^3$ que cumpla con la ecuación (74) para las nubes de la figura 46, las cuales están diferenciadas por una rotación aleatoria de 0.1959 radianes.

$$pcB = \mathrm{T}\{R \cdot pcA\}. \tag{74}$$

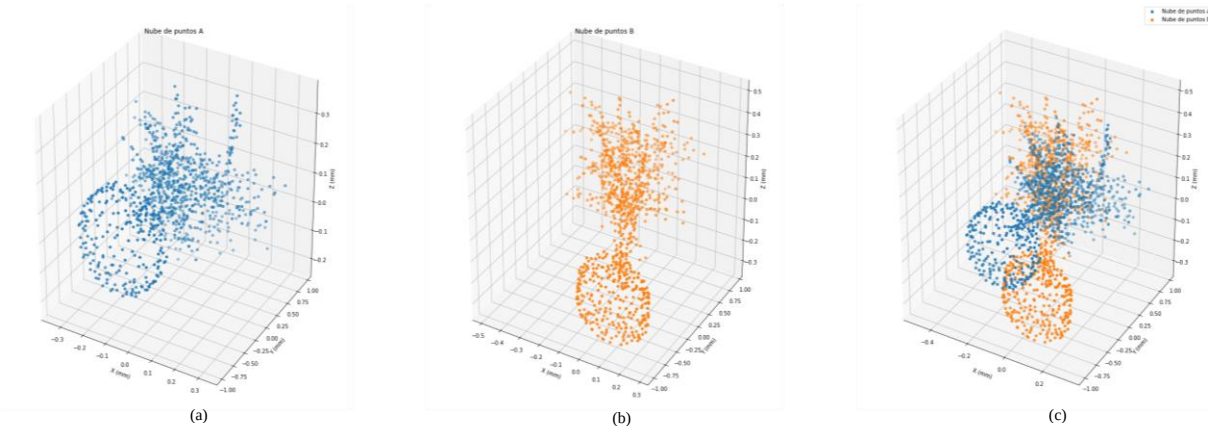

Figura 46. (a) Nube de puntos A (origen), (b) Nube de puntos B (destino), (c) Nube de puntos A y B (diferenciadas por rotación) a registrar.

Ejecutando el algoritmo propuesto, se obtiene la aproximación de matriz de transformación T de la ecuación (75) para un total de 13 iteraciones (ver figura 47):

$$\mathrm{T} = \begin{pmatrix} 0.8368 & -0.3053 & 0.4543 & 0.0207 \\ 0.2979 & 0.9503 & 0.0899 & -0.0115 \\ -0.4592 & 0.0601 & 0.8862 & 0.0033 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{75}$$

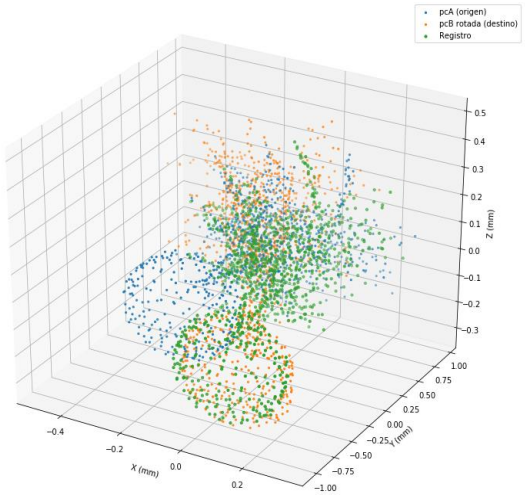

Figura 47. Resultado de registro para prueba de rotación.

4.4.4.2. Prueba de traslación. Se pretende calcular una matriz rígida de transformación T que contenga el parámetro de traslación $t \in \mathbb{R}^3$ que cumpla con la ecuación (76) para las nubes de la figura 48, las cuales están diferenciadas por una traslación aleatoria de 0.4603 milímetros.

$$pcB = \mathrm{T}\{pcA + t\}. \quad (76)$$

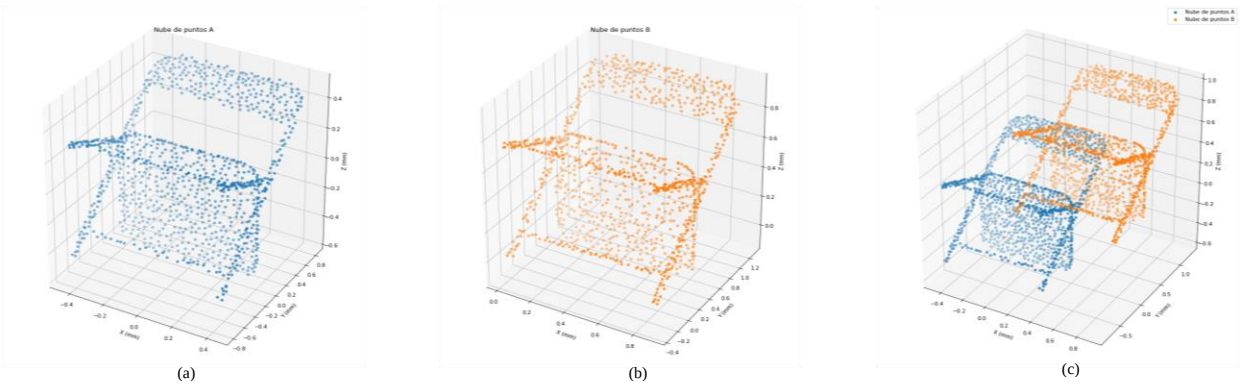

Figura 48. (a) Nube de puntos A (origen), (b) Nube de puntos B (destino), (c) Nube de puntos A y B (diferenciadas por traslación) a registrar.

Ejecutando el algoritmo propuesto, se obtiene la aproximación de matriz de transformación T de la ecuación (77) para un total de 5 iteraciones (ver figura 49):

$$\mathrm{T} = \begin{pmatrix} 9.99 \times 10^{-1} & 2.96 \times 10^{-2} & 7.65 \times 10^{-4} & 4.74 \times 10^{-1} \\ -2.96 \times 10^{-2} & 9.99 \times 10^{-1} & 1.32 \times 10^{-3} & 4.37 \times 10^{-1} \\ -7.26 \times 10^{-4} & -1.34 \times 10^{-3} & 9.99 \times 10^{-1} & 4.63 \times 10^{-1} \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (77)$$

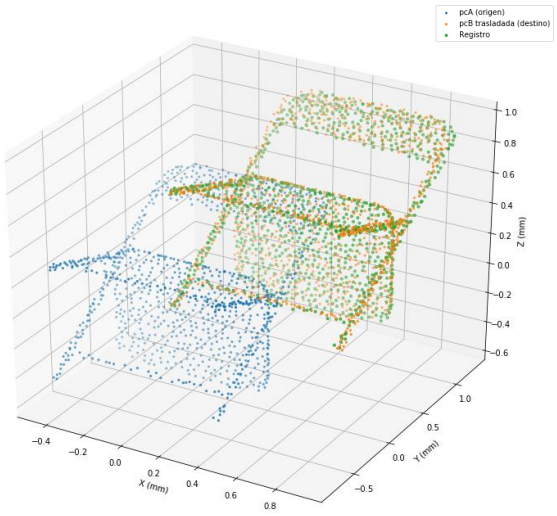

Figura 49. Resultado de registro para prueba de traslación.

4.4.4.3. Prueba de rotación y traslación. Se pretende calcular una matriz rígida de transformación T que contenga los parámetros de rotación y traslación $R, t \in \mathbb{R}^3$ que cumplan con la ecuación (78) para las nubes de la figura 50, las cuales están diferenciadas por una rotación y traslación aleatoria de 0.3134 radianes y 0.4989 milímetros respectivamente.

$$pcB = \mathrm{T}\{R \cdot pcA + t\}. \tag{78}$$

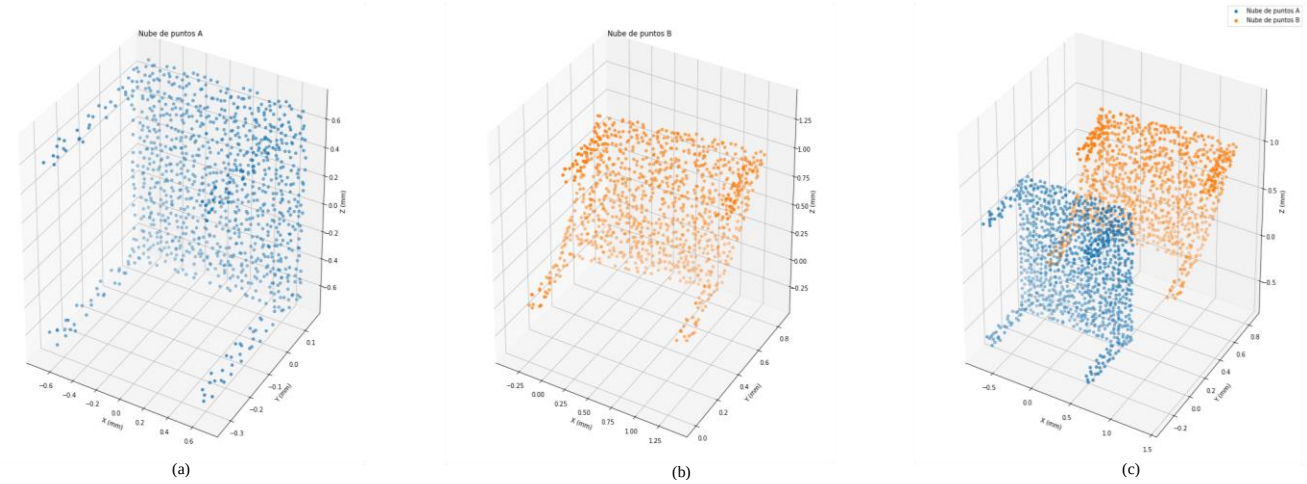

Figura 50. (a) Nube de puntos A (origen), (b) Nube de puntos B (destino), (c) Nube de puntos A y B (diferenciadas por rotación y traslación) a registrar.

Ejecutando el algoritmo propuesto, la aproximación de matriz de transformación T de la ecuación (79) para un total de 12 iteraciones:

$$\mathrm{T} = \begin{pmatrix} 0.9139 & -0.2795 & 0.2942 & 0.4575 \\ 0.3267 & 0.9368 & -0.1248 & 0.4887 \\ -0.2407 & 0.2102 & 0.9475 & 0.5196 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{79}$$

En la figura 51 se observa el resultado de aplicar la matriz de transformación T de la ecuación (79) a la nube de puntos origen pcA para llegar a la nube de puntos destino pcB:

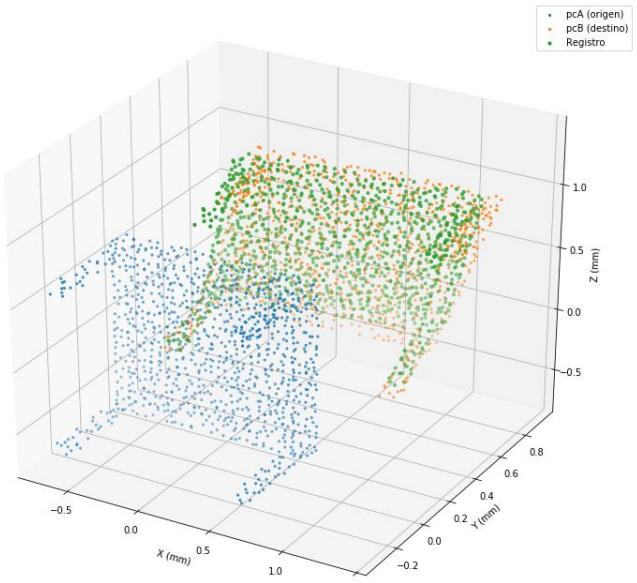

Figura 51. Resultado de registro para prueba de rotación y traslación.

4.4.4.4. Prueba de Integración. El registro de nubes de puntos es el fundamento base para llevar a cabo tareas de integración. Para probar el algoritmo propuesto en dichas tareas, se tomaron dos muestras del modelo Armadillo de Standford, pcA ocluida o incompleta y pcB completa y transformada mediante una rotación y traslación de 0.3126 radianes y 0.0152 milímetros respectivamente (ver figura 52).

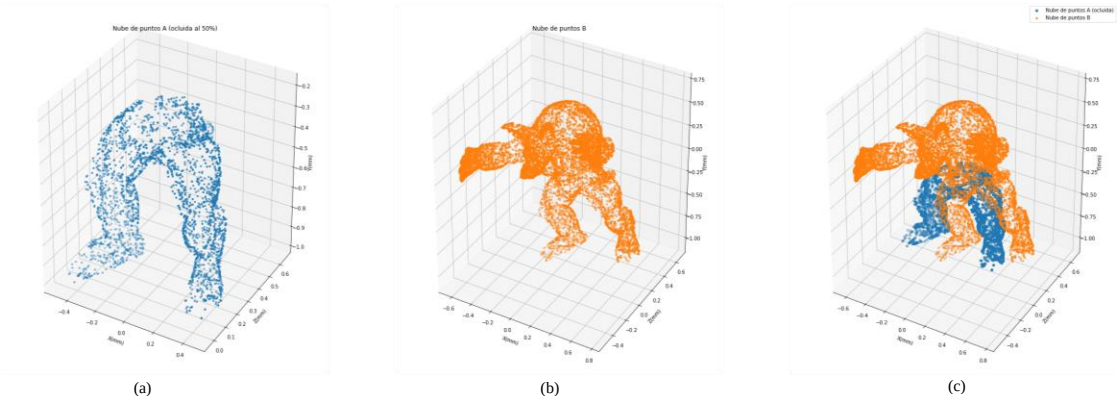

Figura 52. (a) Nube de puntos pcA (ocluida), (b) Nube de puntos B (transformada), (c) Nubes de puntos pcA y pcB para integrar.

Posteriormente, se procedió a segmentar manualmente las nubes de puntos (Seg_pcA, Seg_pcB) (ver figura 53) con el fin de hallar la transformación rígida T para tales segmentos que hacen que se cumpla la condición descrita en la ecuación (80):

$$\text{Si, } Seg_{pcB} = \mathrm{T}\left\{R \cdot Seg_{pcA} + t\right\}, \quad \text{entonces} \quad pcB = \mathrm{T}\left\{R \cdot Seg\_pcA + t\right\}, \tag{80}$$

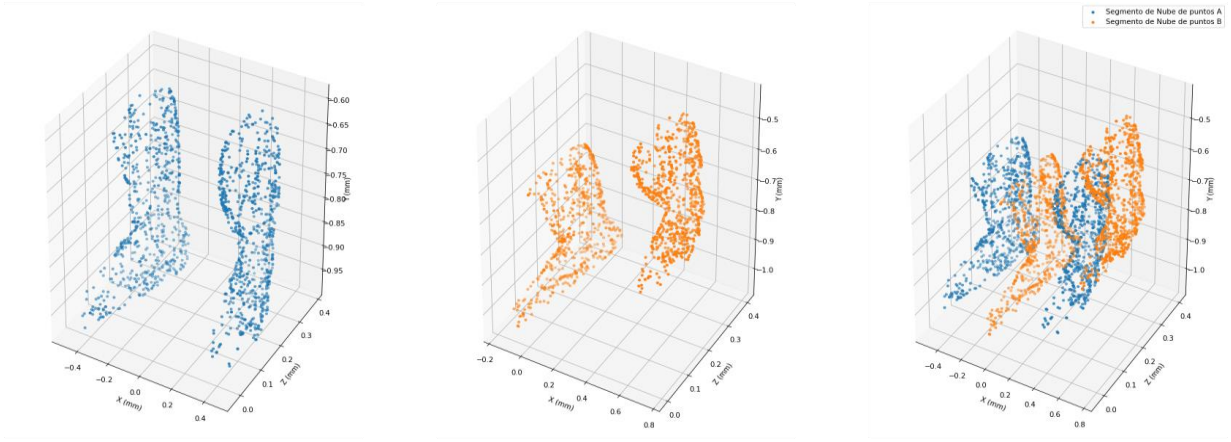

Figura 53. Segmentos de las nubes de puntos a registrar.

Una vez obtenida la aproximación de la transformación rígida T de la ecuación (81), para segmentos en común entre dos nubes de puntos diferentes, dicha transformación puede ser aplicada a toda la nube de puntos para obtener la integración de los dos conjuntos (ver figura 54).

$$\mathrm{T} = \begin{pmatrix} 0.9731 & -0.2115 & -0.0911 & 0.2748 \\ 0.2232 & 0.9637 & 0.1464 & 0.0618 \\ 0.0568 & -0.1628 & 0.9850 & 0.0150 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{81}$$

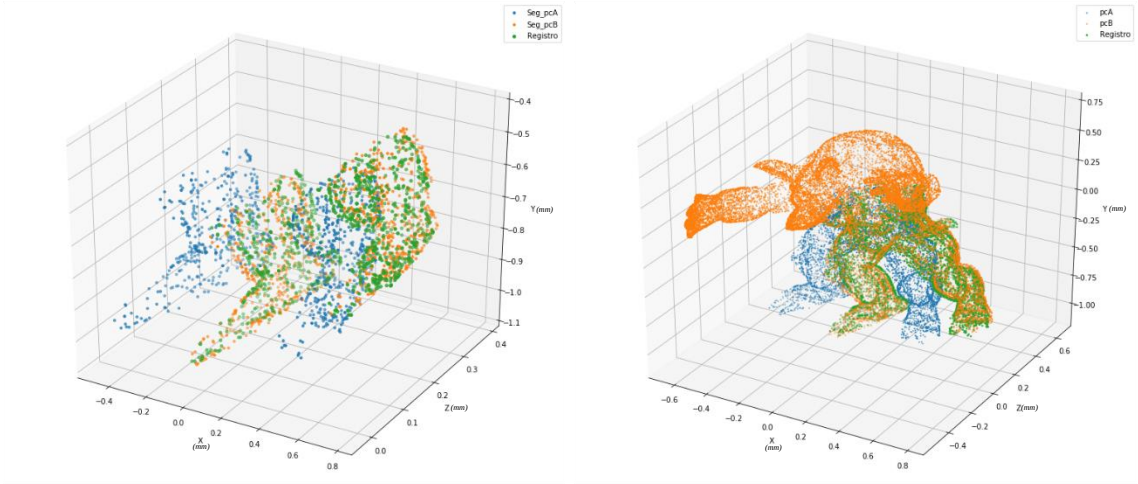

Figura 54. Resultado de registro de segmentos e Integración de nubes de puntos.

En base a lo anterior, es posible inferir que, para tareas de integración de nubes, solo basta con realizar el registro de alguna porción en común entre las dos nubes de puntos y aplicar dicha transformación al resto de puntos para obtener la integración o el registro completo. El desafío en este proceso radica en encontrar el segmento en común de forma automática.

4.4.4.5. Prueba de Reflectancia. Para probar el algoritmo propuesto en labores de registro cuando hay reflectancia de datos, se tomaron dos muestras de puntos diferentes del modelo Conejo de Standford pcA y pcB (ver figura 55), y a pcB se le adicionó un factor de ruido que simularía la reflectancia. También se le aplicó una transformación de rotación y traslación aleatoria (para el ejemplo fue de $R = 0.3671$ y $t = 0.2854$). El parámetro de ruido que simula la reflectancia se describe mediante la ecuación (82):

$$pcB = pcB + \sigma r \,, \tag{82}$$

onde r es un parámetro de ruido aleatorio a partir de una distribución normal y σ es un factor de amplitud del ruido que determina su magnitud.

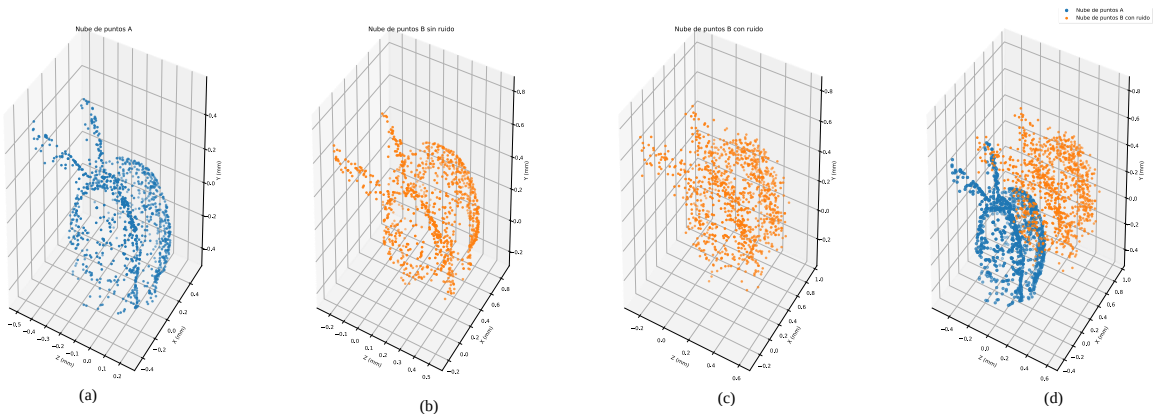

Figura 55. (a) Nubes de puntos A (origen), (b) Nube de puntos B sin ruido, (c) Nube de puntos B con ruido (destino), (d) Nube de puntos A y nube de puntos B con ruido.

A pesar del ruido que tiene la nube de puntos pcB, el algoritmo propuesto logra hacer una buena aproximación de la matriz de transformación T (83) para realizar el registro en este tipo de casos (ver figura 56):

$$\mathrm{T} = \begin{pmatrix} 0.9664 & -0.2562 & -0.0176 & 0.3173 \\ 0.2566 & 0.9661 & 0.0262 & 0.2725 \\ 0.0102 & -0.0298 & 0.9995 & 0.2681 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{83}$$

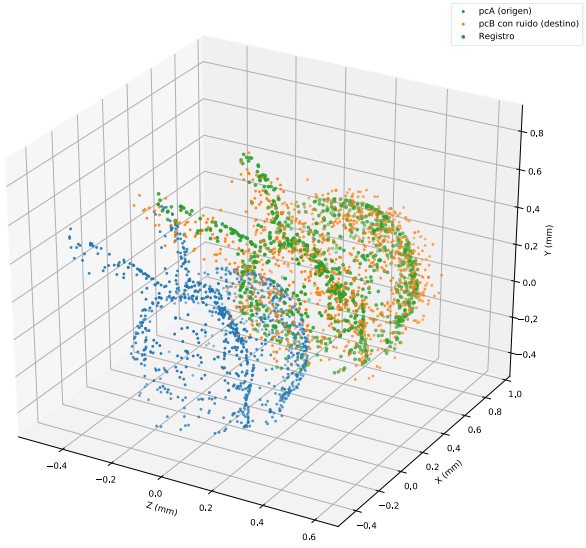

Figura 56. Resultado de registro de nubes con parámetro de reflectancia.

Resultados

Siguiendo con la metodología planteada, el algoritmo propuesto es validado en términos de exactitud y comparado con otros algoritmos y métodos como ICP, RANSAC y CPD para tareas de registro de nubes de puntos. El proceso consta de los siguientes pasos:

1. Tomar dos nubes de puntos S_A y S_B desordenadas y muestreadas aleatoriamente de un modelo de ModelNet40 y aplicar una rotación r y traslación t aleatoria a S_B para ser registradas posteriormente.

2. Conociendo los parámetros r y t con los que se transformó a S_B, aplicarlos para transformar a S_A y así obtener el registro esperado o *ground truth*.

3. Concatenar o unir las dos nubes registradas como una sola nube de puntos S_c y hallar su centroide.

4. Calcular el promedio de las distancias euclidianas entre las todos los puntos y el centroide de S_c para obtener el Valor Teórico o de referencia V_{teo}.

5. Finalizada la última iteración del algoritmo de registro, realizar el paso 3 con la nube de puntos registrada y la nube de puntos S_B, para posteriormente calcular el promedio de las distancias euclidianas entre todos los puntos y el centroide, y así obtener el Valor Experimental V_{exp}.

6. Comparar los valores de referencia respecto a los obtenidos experimentalmente mediante el cálculo del error relativo porcentual a partir de la ecuación (84),

$$Error = \frac{|V_{teo} - V_{exp}|}{V_{teo}} \times 100\,. \tag{84}$$

Se realizaron los siguientes experimentos de la misma manera para el algoritmo propuesto y los otros métodos de registro con el fin de compararlos entre sí:

Experimento 1: Rotación y traslación aleatoria en los tres ejes de rotación.

Experimento 2: Rotación aleatoria en el eje x y traslación de 0.5 mm en los 3 ejes.

Experimento 3: Rotación aleatoria en el eje y y traslación de 0.5 mm en los tres ejes.

Experimento 4: Rotación aleatoria en el eje z y traslación de 0.5 mm en los tres ejes.

Cada uno de los cuatro experimentos anteriormente mencionados se hicieron adicionándole ruido a la nube de puntos S_B mediante la ecuación (82) con $\sigma = 0.02, 0.03, 0.04$ y 0.05 unidades para simular diferentes parámetros de reflectancia. A partir de la ecuación (84) se calculan los errores E_1, E_2, E_3, E_4 para 5, 10 y 15 iteraciones (ver tabla 3):

- E_1: Error relativo del algoritmo propuesto.
- E_2: Error relativo de ICP.
- E_3: Error relativo de RANSAC.
- E_4: Error relativo de CPD.

Además, en cada uno de los experimentos anteriormente mencionados se tomaron los tiempos de ejecución que demora cada algoritmo de registro en registrar dos nubes de puntos, es decir, el tiempo que tarda desde la lectura de las nubes hasta que retorna la matriz de transformación rígida que las registra (ver tabla 4):

- T_1: Tiempo de ejecución del algoritmo propuesto.
- T_2: Tiempo de ejecución de ICP.
- T_3: Tiempo de ejecución de RANSAC.
- T_4: Tiempo de ejecución de CPD.

Tabla 3. Tabla de comparación de errores de algoritmos para diferentes experimentos.

		Experimento 1				Experimento 2				Experimento 3				Experimento 4			
Ruido[u]	**iter**	E_1[%]	E_2[%]	E_3[%]	E_4[%]	E_1[%]	E_2[%]	E_3[%]	E_4[%]	E_1[%]	E_2[%]	E_3[%]	E_4[%]	E_1[%]	E_2[%]	E_3[%]	E_4[%]
0.02	5	0.073	0.047	0.277	7.331	0.012	0.012	0.292	8.807	0.025	0.070	0.319	6.931	0.105	0.040	0.320	6.903
	10	0.051	0.001	0.324	12.73	0.103	0.093	0.325	18.19	0.162	0.018	0.320	15.28	0.075	0.062	0.343	13.70
	15	0.016	0.044	0.323	11.54	0.081	0.025	0.383	10.53	0.130	0.170	0.376	16.08	0.110	0.033	0.409	12.40
0.03	5	0.070	0.174	0.281	6.564	0.228	0.177	0.297	9.261	0.244	0.213	0.315	7.276	0.286	0.195	0.323	6.871
	10	0.213	0.085	0.329	12.16	0.233	0.176	0.353	15.58	0.146	0.415	0.539	14.01	0.251	0.066	0.342	12.15
	15	0.050	0.166	0.470	11.43	0.198	0.109	0.284	15.67	0.190	0.187	0.374	14.95	0.174	0.009	0.404	13.91
0.04	5	0.524	0.443	0.278	5.778	0.371	0.382	0.296	8.400	0.303	0.517	0.320	6.788	0.330	0.363	0.325	6.774
	10	0.255	0.387	0.320	12.01	0.417	0.245	0.353	15.91	0.286	0.374	0.351	13.25	0.462	0.596	0.352	10.89
	15	0.362	0.438	0.363	11.26	0.271	0.318	0.378	17.44	0.330	0.382	0.380	10.46	0.383	0.529	0.387	14.09
0.05	5	0.682	0.554	0.288	5.204	0.670	0.542	0.309	6.781	0.498	0.560	0.317	7.199	0.543	0.708	0.323	6.400
	10	0.352	0.569	0.326	18.72	0.490	0.700	0.357	13.75	0.435	0.454	0.329	14.94	0.796	0.589	0.356	11.19
	15	0.660	0.567	0.541	11.56	0.255	0.464	0.372	15.82	0.403	0.333	0.396	14.18	0.609	0.475	0.416	10.63
Promedio		0.275	0.289	0.343	10.52	0.277	0.270	0.333	13.01	0.262	0.307	0.360	11.77	0.343	0.305	0.358	10.49

A partir de lo anterior, se promediaron los errores obtenidos para cada algoritmo para 5, 10 y 15 iteraciones, independientemente del experimento. Los resultados obtenidos se pueden visualizar en la figura 57.

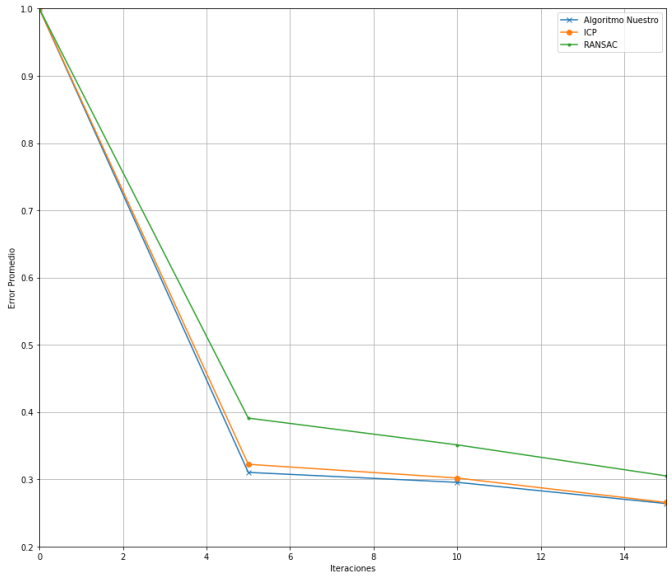

Figura 57. Error promedio obtenido por iteraciones para diferentes algoritmos.

Tabla 4. Comparación de tiempos de ejecución de algoritmos para diferentes experimentos.

		Experimento 1				**Experimento 2**				**Experimento 3**				**Experimento 4**			
Ruido[u]	**iter**	T_1[s]	T_2[s]	T_3[s]	T_4[s]	T_1[s]	T_2[s]	T_3[s]	T_4[s]	T_1[s]	T_2[s]	T_3[s]	T_4[s]	T_1[s]	T_2[s]	T_3[s]	T_4[s]
0.02	5	32.63	0.229	0.029	55.24	30.28	0.129	0.001	58.13	32.24	0.134	0.002	58.03	30.78	0.123	0.001	57.92
	10	63.68	0.224	0.002	52.16	60.78	0.219	0.004	51.74	64.95	0.208	0.005	51.70	57.57	0.224	0.007	52.26
	15	99.98	0.371	0.018	59.83	96.89	0.294	0.020	52.88	116.2	0.289	0.014	61.05	95.88	0.323	0.005	60.28
0.03	5	35.21	0.138	0.001	59.26	34.23	0.128	0.001	58.16	36.98	0.122	0.001	57.82	34.67	0.117	0.004	57.78
	10	70.56	0.217	0.010	51.80	71.53	0.217	0.005	51.81	74.86	0.205	0.002	52.05	74.85	0.212	0.023	52.30
	15	107.1	0.277	0.010	59.91	98.77	0.319	0.040	60.81	106.1	0.298	0.015	60.66	110.3	0.273	0.032	60.45
0.04	5	34.48	0.149	0.004	58.82	34.45	0.130	0.001	57.01	37.63	0.131	0.001	57.65	37.46	0.125	0.003	37.75
	10	77.56	0.264	0.002	51.76	71.79	0.218	0.006	51.69	75.81	0.247	0.015	52.77	79.36	0.257	0.016	52.50
	15	122.1	0.330	0.013	59.51	118.0	0.303	0.031	60.34	114.7	0.321	0.008	54.27	129.8	0.354	0.007	59.64
0.05	5	38.75	0.163	0.001	59.60	38.17	0.122	0.015	58.45	39.84	0.130	0.001	57.59	40.79	0.125	0.002	38.02
	10	84.39	0.203	0.004	58.07	75.76	0.240	0.021	51.53	82.48	0.257	0.005	52.39	84.65	0.252	0.004	52.46
	15	128.9	0.295	0.015	59.84	121.1	0.318	0.012	68.13	133.6	0.291	0.011	60.26	135.1	0.379	0.023	59.97
Promedio		78.43	0.238	0.009	57.150	70.99	0.220	0.013	56.72	76.30	0.219	0.007	56.35	75.95	0.230	0.011	53.44

Adicionalmente, se realizaron tres pruebas o experimentos para valorar la cantidad de puntos correspondientes que es capaz de hallar el algoritmo propuesto. En estas, se presenta el número de pares puntos coincidentes predichos por el algoritmo para diferentes nubes de puntos en cada iteración, hasta 10 iteraciones. Además, se muestran los respectivos errores asociados al registro efectuado con los puntos coincidentes hallados. Cabe aclarar que, para la obtención de estos datos experimentales se deshabilitó el criterio de parada del algoritmo para obtener los resultados en las 10 iteraciones.

- Experimento 1: Dadas dos muestras A y B de una nube de puntos de la categoría "bunny", ambas de tamaño $N = 1024$ puntos. Inicialmente se transformó la nube B con parámetros de rotación $R = 0.3671$ radianes en el eje Z, y traslación $t = 0.2855$ mm. Además, también se le adicionó un factor de ruido de $\sigma = 0.04$ mediante la ecuación (82).
- Experimento 2: Dadas dos muestras A y B de una nube de puntos de la categoría "airplane", ambas de tamaño $N = 1024$ puntos. Inicialmente se transformó la nube B con parámetros de rotación $R = 0.4231$ radianes en los tres ejes, y traslación $t = 0.1544$ mm.
- Experimento 3: Dadas dos muestras A y B de una nube de puntos de la categoría "bed", ambas de tamaño $N = 1024$ puntos. Inicialmente se transformó la nube B con parámetros de 0.6092 radianes de rotación en el eje Z y 0.6112 mm de traslación. Además, se le adición un factor de ruido de $\sigma = 0.1$ mediante la ecuación (82).

Para cada uno de los experimentos anteriores, se procedió a ejecutar el algoritmo propuesto con el fin de encontrar la matriz de transformación que haga el registro rígido de las

nubes *A* y *B* (transformada), pero imprimiendo en cada iteración, la cantidad de pares coincidentes, valores que se pueden observar en la tabla 5.

Tabla 5. Cálculo de pares de puntos coincidentes para tres tipos de experimentos.

	Experimento 1		**Experimento 2**		**Experimento 3**	
Iteración	**N° de pares de puntos coincidentes**	**Error de Registro**	**N° de pares de puntos coincidentes**	**Error de Registro**	**N° de pares de puntos coincidentes**	**Error de Registro**
1	26	0.029	29	0.034	20	0.064
2	24	0.033	26	0.037	14	0.066
3	26	0.033	20	0.034	18	0.067
4	28	0.033	30	0.033	13	0.064
5	32	0.032	30	0.032	25	0.067
6	35	0.031	31	0.053	21	0.063
7	31	0.032	31	0.042	23	0.066
8	35	0.031	41	0.043	21	0.065
9	36	0.030	35	0.040	25	0.067
10	35	0.030	33	0.043	22	0.065
Promedio	30.4	0.031	30.6	0.040	20.6	0.065

Conclusiones

En el proceso de registro de nubes de puntos, un factor fundamental es la búsqueda de la correspondencia o coincidencia de pares de puntos en las nubes. Esto garantiza que, aunque existan elementos externos como la oclusión o la reflectancia, se pueda generar una buena aproximación al registro mediante la estimación de la matriz de transformación rígida. Usualmente en los algoritmos de registro del estado del arte, estos pares de puntos deben ser suministrados mediante intervención humana por el usuario, quien apoyado por su capacidad visual proporciona al algoritmo la información aproximada de cuáles son los pares de puntos coincidentes. El uso de técnicas de Aprendizaje Automático para determinar dichas correspondencias de puntos hace que se pueda automatizar el proceso de registro rígido de nubes de puntos sin que haya la necesidad de intervención humana.

En base a los resultados experimentales, mediante la aplicación del algoritmo propuesto, el cual no necesita de la estimación inicial de correspondencia de puntos, sino que automáticamente estima los pares de puntos coincidentes en las dos nubes de puntos a registrar, se logra obtener un porcentaje de error promedio del 0.275% en el experimento 1, 0.277% en el experimento 2, 0.262% en el experimento 3 y 0.343% en el experimento 4; similar o incluso mejor al de algoritmos como ICP, RANSAC o CPD aun cuando hay datos faltantes o ruidosos por oclusión o reflectancia (ver tabla 3). El rendimiento del algoritmo propuesto en términos de estas comparaciones es el siguiente:

- Diferencia del error relativo en el registro con respecto a ICP: favorable del 0.014% en el experimento 1, desfavorable del 0.003% en el experimento 2, favorable del 0.045% en el experimento 3 y desfavorable del 0.038% en el experimento 4.

- Diferencia del error relativo en el registro con respecto a RANSAC: favorable del 0.068% en el experimento 1, favorable del 0.056% en el experimento 2, favorable del 0.098% en el experimento 3 y favorable del 0.015% en el experimento 4.
- Diferencia del error relativo en el registro con respecto a CPD: favorable del 10.24% en el experimento 1, favorable del 12.73% en el experimento 2, favorable del 11.50% en el experimento 3 y favorable del 10.14% en el experimento 4.

Por otra parte, el algoritmo propuesto sacrifica tiempo de ejecución, pues basado en los resultados, este es muy lento en comparación con los otros algoritmos, consumiendo tiempos de más de 70 segundos en promedio; mientras que algoritmos como ICP o RANSAC no exceden 1 segundo y CPD supera los 50 segundos de ejecución en los cuatro experimentos (ver tabla 4). Los largos tiempos obtenidos en el algoritmo propuesto se deben al coste computacional por el cálculo o estimación de los gradientes mediante la diferenciación automática en cada iteración del algoritmo.

En cuanto a los resultados de correspondencias del algoritmo expuestos en la tabla 5, la cantidad de pares de puntos coincidentes calculados en los experimentos planteados superan ampliamente el número mínimo de puntos coincidentes requerido por el método de descomposición de valores singulares SVD para la obtención de la matriz de transformación, el cual es de 4 puntos. Así, en el experimento 1 y 2, en promedio la cantidad de puntos hallados es aproximadamente 7.5 veces mayor, y en el experimento 3, la cantidad de puntos hallados es aproximadamente 5 veces mayor que dicho valor mínimo.

En base a lo anterior, la cantidad de pares de puntos coincidentes calculados por el algoritmo para efectuar el proceso de registro es variable y depende en gran medida de factores como la similitud ambas nubes, es decir, que tan cerca están de ser registradas, de la estructura

formada por los puntos y de la densidad de estos. Además, también depende del error de predicción del DPDist, el cual es del 12.985% (ver figura 22). Sin embargo, la virtud del algoritmo propuesto es que siempre va a establecer una cantidad mayor a 4 puntos coincidentes, por tanto, siempre se garantizará que se pueda usar de manera favorable el SVD (o incluso otro método) para estimar la matriz de transformación rígida que permita el registro de dos nubes de puntos.

A nivel de software, el algoritmo propuesto cumple con los requerimientos establecidos pues permite un óptimo registro de nubes de puntos tridimensionales (cuantificado por el error relativo de resultados teóricos y experimentales) mediante el uso de técnicas de Aprendizaje Profundo.

Finalmente, se concluye que, el algoritmo propuesto y diseñado a partir del uso y puesta en práctica de conceptos y técnicas de Aprendizaje Profundo, y además, basados en los resultados obtenidos, es capaz de determinar una muy buena aproximación de la matriz de transformación rígida entre dos nubes de puntos desordenadas y sin estimación de correspondencia inicial. Además, satisface medidas de exactitud a partir del error relativo respecto a registros hechos manualmente y es capaz de competir con otros algoritmos de registro vigentes en el estado del arte.

El presente trabajo de investigación fue aceptado para ser presentado en el evento internacional *Latin America Optics and Photonics Conference*, organizado por la Sociedad de Óptica Estadounidense (OSA) y llevado a cabo en la ciudad de Recife, Pernambuco, Brasil, del día 07 al 11 de agosto del año en curso.

Recomendaciones y Trabajo Futuro

- Automatizar el proceso de localización de las regiones comunes en dos nubes de puntos que requieren integración es fundamental para dicha tarea pues una vez determinadas las regiones se puede aplicar el algoritmo de registro propuesto y obtener buenos resultados.
- Mejorar la estimación de correspondencia o coincidencia de puntos es clave para mejorar el rendimiento del algoritmo propuesto, esto se podría lograr mediante:
  - Un entrenamiento más robusto del DPDist: Aumentar a la mayor cantidad posible, variar y mejorar el conjunto de datos con el que se entrena el modelo, ya que el error de la correspondencia de puntos mediante el gradiente depende del rendimiento obtenido en el entrenamiento del DPDist.
  - La modificación del modelo DPDist para que además de generar una representación por localidades, no pierda la generalidad de la nube de puntos. Esto para hacer que el DPDist sea menos susceptible a las grandes rotaciones.
- DPDist como métrica de comparación puede resultar útil para tareas como la clasificación de nubes. También es posible usar DPDist como función de pérdida para entrenar una red neuronal especializada en registro de nubes de puntos, aprovechando su capacidad de cambiar la representación de los puntos hace que puedan surgir nuevos métodos de registro mediante una Aprendizaje Automático directamente (Urbach et al., 2020).
- Basados en el comportamiento secuencial de los resultados de DPDist obtenidos a partir de las pruebas realizadas con rotaciones uniformes de 0° a 360°, la posibilidad de usar una arquitectura de Red Neuronal Recurrente (Karpathy et al., 2015) podría

ser de gran utilidad para tratar este tipo de datos y poder predecir los ángulos de rotación implícitos entre dos nubes de puntos comparadas mediante DPDist.

- El método propuesto para el hallazgo de correspondencias de puntos mediante el gradiente del modelo DPDist, también puede ser usado en combinación con algoritmos de registro como RANSAC, CPD y otros, ya que proporcionaría una buena estimación inicial de pares de puntos y así sería posible mejorar las estimaciones de registro de cada algoritmo de manera individual.

Referencias Bibliográficas

Aguilar, R. (1999) "*Red Neuronal de Topología Flexible*". En: VI Congreso Nacional de Ciencias de la Computación, La Paz, Bolivia. Septiembre de 1999.

Alonso, R. F., García-Bermejo, J. G., & Casanova, E. Z. (2007). *Alineamiento automático de nubes densas de puntos a partir de información geométrica y cromática*.

Aoki, Y., Goforth, H., Srivatsan, R. A., & Lucey, S. (2019*). Pointnetlk: Robust & efficient point cloud registration using pointnet.* In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 7163-7172).

Barrueco,D. *"Gradientes"*. Recurso Web Interactive. https://interactivechaos.com/es/manual/tutorial-de-deep-learning/gradientes

Ben-Shabat, Y. (2018) *What is 3D modified Fisher Vector (3DmFV) representation for 3D point clouds.* Yizhak Ben-Shabat (Itzik), Phd Research Fellow Web Resource: https://www.itzikbs.com/what-is-3d-modified-fisher-vector-3dmfv-representation-for-3d-point-clouds?doing_wp_cron=1633035226.9158279895782470703125

Ben-Shabat, Y., Lindenbaum, M., & Fischer, A. (2017). 3d point cloud classification and segmentation using 3d modified fisher vector representation for convolutional neural networks. arXiv preprint arXiv:1711.08241.

Bergström, P., & Edlund, O. (2014). Robust registration of point sets using iteratively reweighted least squares. Computational optimization and applications, 58(3), 543-561.

Besl, P. J. & McKay, N. D. (1992). "*A method for registration of 3-D shapes*," IEEE Trans. Pattern Anal. Mach. Intell., vol. 14, no. 2, pp. 239–256, Feb. 1992.

Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford university press.

Braspenning, P. J., Thuijsman, F., & Weijters, A. J. M. M. (1995). Artificial neural networks: an introduction to ANN theory and practice (Vol. 931). Springer Science & Business Media.

Bronstein, A., Bronstein, M. & Kimmel, R. (2008). *Numerical Geometry of Non-Rigid Shapes*. 1.a Ed. Springer PublishingCompany,Incorporated.

Chum, O. and Matas, J. (2008) *Optimal Randomized RANSAC*. IEEE transactions on pattern analysis and machine intelligence. Vol 30, August 2008.

(2017). *"Cluster Guane*". Recurso Web Super Computación y Cálculo Científico UIS http://wiki.sc3.uis.edu.co/index.php/Cluster_Guane

Curless, B., and Levoy, M. (1996) *A volumetric method for building complex models from range images.* In SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pages 303{312, New York, NY, USA, ACM}.

Deng, H., Birdal, T., & Ilic, S. (2018). *Ppfnet: Global context aware local features for robust 3d point matching*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 195-205).

Dennis, B.S., Solís, M. y Aranda, B.S.H. (2021). *Registro de Nubes de Puntos 2D: Comparación entre Métodos Iterativos y una Red Neuronal Artificial*. Rev. Aristas 2021, 8, 301–305.

Diebel, J. (2006). *Representing attitude: Euler angles, unit quaternions, and rotation vectors. Matrix*, 58(15-16), 1-35.

Dumoulin, V., & Visin, F. (2016). *A guide to convolution arithmetic for deep learning*.

Fu, K., Liu, S., Luo, X., & Wang, M. (2021). Robust point cloud registration framework based on deep graph matching. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 8893-8902).

Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning. Introduction.* The MIT Press.

Gualdrón, C. Bautista, L. Machuca, J. (2019). *Reconstrucción 3D para el desarrollo de prótesis de miembro inferior*. Revista UIS Ingenierías, Vol. 19, n.°1, pp 73-86.

Hagan, M.T., Demuth, H.B., Beale M.H. Y De Junio, O. (2014) *Neural Network Design* (2nd Edition). USA: Oklahoma State University, 2014.

He, K., Zhang X., Ren, S. and Sun, J. (2016). *Deep residual learning for image recognition.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzkle, W. (1992). Surface reconstruction for unorganized points (Vol. 26, No. 2, pp. 71-78). ACM.

Johann, G. (2010). "Valor Singular de descomposición", Recurso Web Software Broderbund: https://es.frwiki.wiki/wiki/D%C3%A9composition_en_valeurs_singuli%C3%A8res

Karpathy, A., Johnson, J., & Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. arXiv preprint arXiv:1506.02078.

Kim P. (2017) *Deep Learning. In: MATLAB Deep Learning*. Apress, Berkeley, CA.

Kubat, M. (1999). *Neural networks: A comprehensive foundation by Simon Haykin, Macmillan,* 1994, ISBN 0-02-352781-7. The Knowledge Engineering Review, 13(4), 409-412.

Licenses, U. C. C. P. (2013). The MIT License (MIT).

Maiseli, B., Gu, Y., & Gao, H. (2017). *Recent developments and trends in point set registration methods,* Journal of Visual Communication and Image Representation, vol. 46, pp. 95 – 106.

Malamas, E. N., Petrakis, E. G., Zervakis, M., Petit, L., y Legat, J. D. (2003). *A survey on industrial vision systems, applications, and tools*. Image and vision computing, 21(2), 171-188.

Maturana, D., & Scherer, S. (2015). *Voxnet: A 3d convolutional neural network for real-time object recognition*. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 922-928). IEEE.

Myronenko, A., & Song, X. (2010). *Point set registration: Coherent point drift*. IEEE transactions on pattern analysis and machine intelligence, 32(12), 2262-2275.

Myronenko, A., Song, X., & Carreira-Perpiñán, M. (2006). *Non-rigid point set registration: Coherent Point Drift*. NIPS. 19. 1009-1016.

Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25). San Francisco, CA, USA: Determination press.

O'Shea, K., & Nash, R. (2015). *An introduction to convolutional neural networks*. arXiv preprint arXiv:1511.08458.

Palomares, F. G., Monsoriu, J. A., & Alemany, E. (2016). *Aplicación de la convolución de matrices al filtrado de imágenes*. Modelling in Science Education and Learning, 9(1), 97-108.

Payá, R.(2015). *Tema 8 Vector Gradiente*. Recurso Web de la asignatura Análisis Matemático. Universidad de Granada (España): https://www.ugr.es/~rpaya/documentos/AnalisisI/2015-16/Gradiente.pdf

Pérez, F., Gómez, J. (2016). *Registro Automático de nubes de puntos*. Correspondencias (pp. 31-32).

Perez-Gonzalez, J., Luna-Madrigal, F., & Piña-Ramirez, O. (2019). *Deep learning point cloud registration based on distance features.* IEEE Latin America Transactions, 17(12), 2053-2060.

Perronnin, F. & Dance, C. (2007). *Fisher kernel on visual vocabularies for image categorization*. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1–8. IEEE, 2007.

Perronnin, F., Sánchez, J. and Mensink, T. (2010). *Improving the fisher kernel for large-scale*

Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). *Pointnet: Deep learning on point sets for 3d classification and segmentation*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 652-660).

Qi, C.R., Su, H., Nießner, M., Dai, A., Yan, M. and Guibas, L.J. (2016) *Volumetric and multi-view cnns for object classification on 3d data.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5648–5656.

(2020). *Recursos de computacion de alto rendimiento (HPC)*. Recurso Web Super Computación y Cálculo Científico UIS https://www.sc3.uis.edu.co/recursoshpc/

Reynolds, D. A. (2009). Gaussian mixture models. Encyclopedia of biometrics, 741(659-663).

Royo, S., Ballesta, M. (2019). *An Overview of Lidar Imaging Systems for Autonomous Vehicles*. Applied Sciences 9, no. 19: 4093. https://doi.org/10.3390/app9194093.

Salcedo, A. F. C., Martínez, A. B., y Salazar, E. A. Q. (2012). *Procesamiento de nubes de puntos por medio de la librería PCL*. Scientia et technica, 2(52), 136-142.

Sarode, V., Li, X., Goforth, H., Aoki, Y., Srivatsan, R. A., Lucey, S., & Choset, H. (2019). Pcrnet: Point cloud registration network using pointnet encoding. arXiv preprint arXiv:1908.07906.

Silva, C., Welfer, D., Gioda, F. P., and Dornelles, C. (2017). "*Cattle brand recognition using convolutional neural network and support vector machines*," IEEE Latin America

Transactions, vol. 15, no. 2, pp. 310–316, Feb. 2017. [Online]. Available: https://doi.org/10.1109/tla.2017.7854627

Skala, V.(2013). *Journal of WSCG*. University of West Bohemia. Vol 21, Number 1. Pp. 32.

Stewart, J. (2012) *Cálculo de varias variables, Trascendentes tempranas.* Cengage Learning. 7ma Edición.

Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A.A. (2017). *Inception-v4, inception-resnet and the impact of residual connections on learning.* In AAAI, pages 4278–4284.

(2022). *tf.gradients*. Recurso Web Libreria Tensorflow: https://www.tensorflow.org/api_docs/python/tf/gradients

The Stanford 3D Scanning Repository. [Online]. Available: http://www.graphics.stanford.edu/data/ 3Dscanrep/

Tian, B., Jiang, P., Zhang, X., Zhang, Y., Wang, F. (2016) *A Novel Feature Point Detection Algorithm of Unstructured 3D Point Cloud*. In: Huang DS., Han K., Hussain A. (eds) Intelligent Computing Methodologies. ICIC 2016. Lecture Notes in Computer Science, vol 9773. Springer, Cham, pag 737.

Torre, C. (2011). *Contribuciones al alineamiento de nubes de puntos 3d para su uso en aplicaciones de captura robotizada de objetos*. Alineamiento 3D (pp. 18). Universidad de Cantabria.

Torres, J. C., Cano, P., Melero, J., España, M., y Moreno, J. (2010). *Aplicaciones de la digitalización 3D del patrimonio*. Virtual Archaeology Review, 1(1), 51-54.

Turk, G., & Levoy, M. (1994, July). *Zippered polygon meshes from range images.* In Proceedings of the 21st annual conference on Computer graphics and interactive techniques (pp. 311-318).

Urbach, D. (10 de marzo de 2021). *DPDist: Comparing Point Clouds Using Deep Point Cloud Distance - ECCV2020* [Video]. https://www.youtube.com/watch?v=6G73JEut-UQ&t=120s

Urbach, D., Ben-Shabat, Y., & Lindenbaum, M. (2020). *DPDist: Comparing point clouds using deep point cloud distance*. In European Conference on Computer Vision (pp. 545-560). Springer, Cham.

Weisstein, E.W. "*Chain Rule.*" From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/ChainRule.html

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J (2015) *3d shapenets: A deep representation for volumetric shapes*. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2015)

Yang, J., Li, H., Campbell, D., & Jia, Y. (2015). Go-ICP: A globally optimal solution to 3D ICP point-set registration. IEEE transactions on pattern analysis and machine intelligence, 38(11), 2241-2254.

Zill, D., Wright, W. (2011) *Cálculo de varias variables.* The McGraw Hill Companies. 4ta Edición.

Zitová, B. & Flusser, J. (2003). *Image Registration: A Survey*, Image Vision Computing, vol. 21, no. 11, pp. 977– 1000, Oct. 2003.