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

MÉTODO PARA ESTABLECER EL MODELO TRIDIMENSIONAL ORIGINAL DE UNA PIEZA MECÁNICA DAÑADA.

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MECÁNICA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MECÁNICO

ERICK ALFRED KRACHT GAJARDO

PROFESOR GUÍA:
JUAN ZAGAL MONTEALEGRE
PROFESOR CO-GUÍA:
RUBÉN FERNÁNDEZ URRUTIA

COMISIÓN:
RODRIGO HERNÁNDEZ PELLICER

SANTIAGO DE CHILE
2022

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MECÁNICA
MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: **ERICK ALFRED KRACHT GAJARDO**
FECHA: 2022
PROF. GUÍA: JUAN ZAGAL MONTEALEGRE

MÉTODO PARA ESTABLECER EL MODELO TRIDIMENSIONAL ORIGINAL DE UNA PIEZA MECÁNICA DAÑADA.

Con la intención de un futuro donde la economía circular se vuelve el paradigma imperante y apuntando a que se pueda recuperar piezas con daño de manera fácil, económica y precisa, es que el objetivo de este trabajo es generar la representación de una pieza mecánica desde una nube de puntos o malla de triángulos, de una pieza que presente desgaste producto del trabajo en que se desempeñe. Esto surge del interés de poder automatizar el proceso de reparación para piezas mecánicas usadas en sectores como la minería o la industria manufacturera, ya que los recursos humanos y tiempos involucrados actualmente en este aspecto impactan en los costos de la industria. Para alcanzar este objetivo, se busca levantar nubes de puntos de piezas y mallas de triángulos con falla que puedan ser representadas en sus formas primitivas, y se pueda detectar la existencia de regiones de puntos distinguiendo secciones con y sin daño.

Para esto, se propone un método iterativo para cilindros y conos usando el algoritmo RANSAC donde en cada iteración se selecciona un conjunto de 3 puntos orientados en el espacio, en el caso de cilindros se define el eje central de éste desde la intersección de planos calculados a partir del conjunto de 3 puntos y el radio se calcula como la distancia de estos puntos al eje central; para el caso de conos, se obtiene el vértice del cono desde la intersección de 3 planos, la dirección del cono se obtiene de proyectar una línea desde el vértice a un punto que resulta de la intersección de las normales de estos planos, y el ángulo de apertura del cono se calcula mediante trigonometría conociendo la distancia de un punto al vértice y al eje del cono.

Se hace uso de Python para el desarrollo del trabajo así como librerías específicas para el entorno de trabajo que implican las nubes de puntos. Se escriben los códigos representando los métodos propuestos además de otros métodos para la detección de parámetros en figuras primitivas, para comparar la eficiencia y eficacia al determinar los parámetros de éstas. Para esto se evalúa un conjunto de 3 cilindros y 3 conos, a los cuales se le evalúa el desempeño de los métodos para cada pieza seleccionada y se toma registro de la confiabilidad de los valores obtenidos luego de usar cada método 2.000 veces. A partir de la estadística que se recoge se determina confiabilidad en la detección de dirección de las piezas, tiempo requerido para ejecutar el método, el radio calculado para el cilindro, el ángulo de apertura para el cono, y el error porcentual asociado al cálculo de estos dos últimos parámetros.

Los resultados obtenidos permiten afirmar que el método propuesto entrega los parámetros solicitados con una eficiencia mayor a lo actualmente disponible, con tiempos de ejecución menores al 15 % de lo ya existente en caso de cilindros y de un 50 % aproximadamente en figuras cónicas.

Agradecimientos

El primer agradecimiento que quiero hacer es a mi papá y a mi mamá, por apoyarme en el camino de mi formación como ingeniero y en el momento que opté por complementar un postgrado al mismo tiempo: A mi papá, por haber sido un apoyo y permitirme estudiar mi carrera costeando una parte importante de la misma, y a mi mamá por todas las veces que fue a verme a la pieza mientras yo estaba ocupado preparando el material de una prueba o trabajo de la universidad. También quiero agradecer a mi pareja, Verónica, que me ha acompañado desde antes de tomar la decisión de tomar el camino de postgrado, y me ha dado su apoyo y se llena de orgullo con cada logro que he alcanzado.

Agradecer también a mis profesores: el profesor guía Juan Cristobal Zagal y el profesor co-guía Rubén Fernández, por haberme recibido como estudiante y acogerme posteriormente como ingeniero de proyecto, por transmitirme el interés que finalmente motivó que mi área de investigación y trabajo se enfoque en la automatización y robótica, así como también agradecer la disposición que presentaron al momento de resolver las dudas que se fueron presentando y la guía que me dieron para encontrar la respuesta a éstas. Agradecer también al profesor Rodrigo Hernández por haber aceptado ser parte de la comisión evaluadora de mi trabajo, y un agradecimiento especial a los profesores Aquiles Sepúlveda y Ramón Frederick, de quienes fui ayudante durante varios semestres; les agradezco por haberme dado la oportunidad de asistirlos en sus asignaturas y adquirir confianza con ello.

Finalmente y no por ello menos importante, es que le quiero agradecer a los amigos que hice durante mi paso por la universidad, Diego, José y Mauricio, con quienes se hizo llevadero todos esos días que pasamos preparándonos para una prueba o simplemente divirtiéndonos en la sala del departamento de mecánica, con quienes las risas no faltaron. A Bastián, Sebastián y Daniel, que conocí más adelante en mi carrera y se volvieron importantes amigos que también tuvieron su aporte al ayudarme a resolver alguna que otra duda cuando no tenía claras las cosas. Y también a los amigos que tenía de antes de la universidad, Andrés que me ayudaba a mitigar el estrés de los estudios y el trabajo, sobre todo durante el contexto sanitario de pandemia que se vivió durante el desarrollo de este trabajo, y a David, que me distraía de mi estrés con las bromas que me daba y que en la etapa final de mi carrera me ayudó con la redacción de este escrito, la cual en más de una ocasión le hizo agarrarse el pelo consternado, pero pudo ayudarme a que redactara mejor el texto.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Antecedentes generales

En diversas industrias y servicios, tales como la minería, las centrales de energía, la industria manufacturera, entre otras actividades, se trabaja con piezas mecánicas que luego de su constante uso, se desgastan, ya sea por erosión, por abrasión o corrosión.[1] Indistinto de cuál sea el motivo del desgaste producido, éste puede aumentar en el tiempo si no llega a ser tratado, causando la aparición de fallas que posteriormente vuelven la pieza inutilizable. Esto repercute en costos por la adquisición de una nueva pieza y por el tiempo en que la máquina no funciona por no tener la pieza mecánica funcional. Para evitar llegar al descarte de la pieza, se hacen reparaciones a la pieza mediante soldadura, depositando material en las secciones de la pieza en la cual se encuentre una falla para luego hacer un mecanizado de la superficie recién reparada y así dar un acabado superficial; esto además de hacer una reducción en la concentración de esfuerzos que se produce al realizar la soldadura. Todo esto se hace con el fin de extender la vida útil de la pieza y no tener que recurrir al cambio de ésta por una nueva. Esta recuperación repercute en un mejor manejo de recursos, ya que por lo general la reparación de una pieza tiene costos asociados menores a lo que sería la adquisición de una nueva pieza.

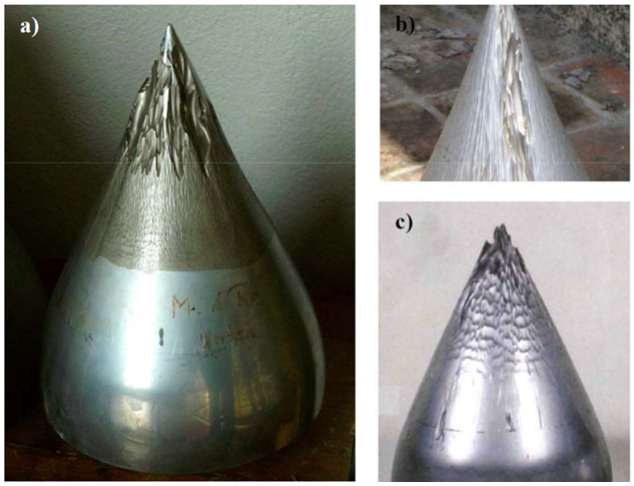


Figura 1.1: Puntas de tres agujas de turbina Pelton, mostrando el desgaste que se busca reparar. *Imagen tomada de [2]*

Bajo la premisa de hacer un mejor uso de los recursos de los que dispone la industria, es que la automatización y robótica ha tomado una fuerte participación en la definición de estas tareas, buscándose automatizar este proceso de recuperación de piezas mecánicas dado el alto uso de recursos humanos y tiempo (esto si se tiene en consideración las soluciones actuales que existen para abordar este problema), quienes son soldadores expertos por un lado, y robots manejados por un operador en el otro lado.[3]

A partir de esto surge la necesidad de generar sistemas de automatización de manera que puedan asistir a los operarios, reduciendo el costo asociado a largos tiempos de ejecución, aumentando la calidad de la reparación realizada y permitiendo una mayor flexibilidad a la hora de abordar distintos tipos de fallas. Sin embargo, al momento de hacer una recuperación de una pieza de manera automática, hay ciertos parámetros que se deben tomar en consideración: como la ubicación espacial de la pieza que se quiere escanear, entendiéndose su orientación en el espacio, como también la distinción de la forma que la pieza a reparar debería tener.

Sin embargo, el computador no necesariamente tiene esta capacidad y esto puede conllevar a que de hecho no sepa hasta que punto deba realizar la reparación de un eje o de una aguja, piezas que de hecho son aproximables sin mucha pérdida de información a estas figuras, cilindros y conos.

En este sentido y buscando la automatización de todo el proceso, es que surge la necesidad del desarrollo de una metodología que permita detectar parámetros de una figura de manera tal que se disponga de información pertinente para facilitar la recuperación de una pieza. Esto es una tarea que podría ser asistida mediante la participación de un operario, pero dado el enfoque de automatización que se quiere alcanzar, se requiere de una forma en la que el sistema sea capaz de determinar sin intervención humana cuando los datos que está observando corresponden a parte de la zona dañada de una pieza, o bien corresponden a secciones sin daño o sanas de la pieza, las cuales se usan como punto de referencia para saber como debería ser la pieza una vez esté recuperada. A partir de este enfoque de automatización, se busca que la interacción con el operario sea la mínima para que todo el trabajo sea lo más automatizado posible, por lo que el sistema computacional sea capaz de representar a partir de un escaneo hecho por el usuario cuales son las fallas presentes en la pieza. Con ello corresponde hacer una reconstrucción digital de la pieza para así poder entregar dicha información al usuario, con el fin de que este pueda hacer una recuperación de la pieza sin la necesidad de por ejemplo disponer de los planos de ésta.

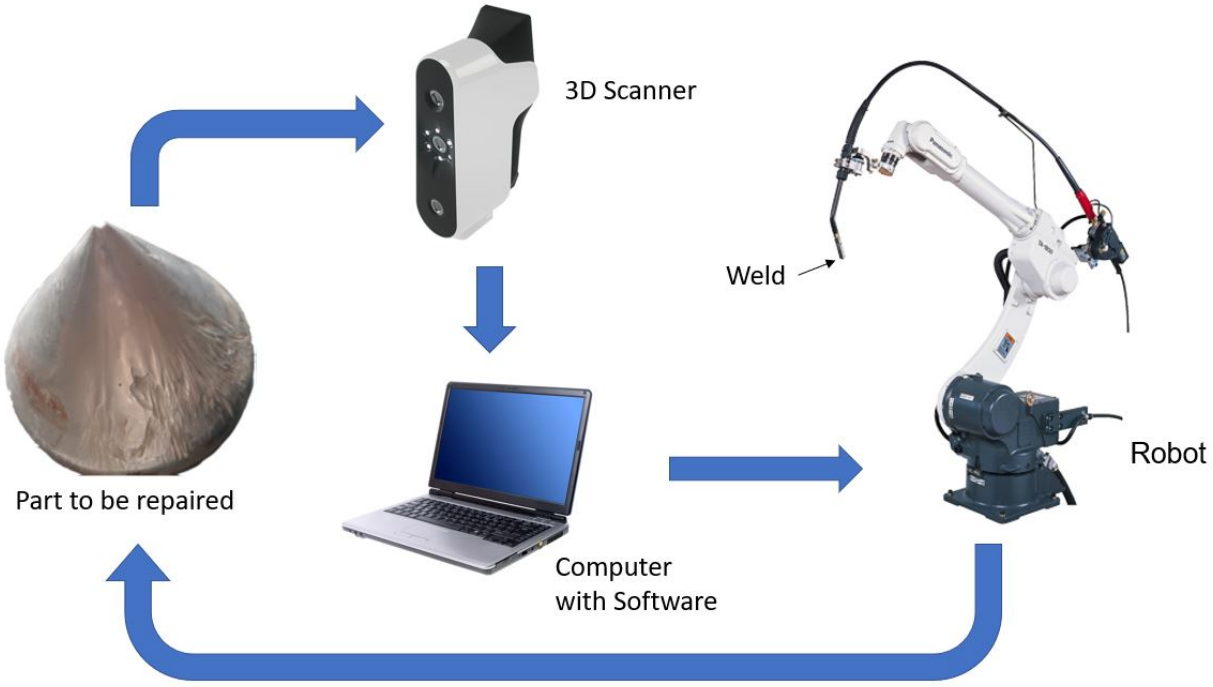


Figura 1.2: Esquema mostrando el flujo de la información, donde la pieza mecánica es escaneada, la información se transmite a un computador con el que se determinan parámetros e instrucciones para el robot, esto es transmitido al robot y se realiza reparación por soldadura de la pieza.

La recuperación planteada parte desde la premisa de generar un modelo 3D de la pieza mediante un escáner en que se conocen las regiones sanas y dañadas de la pieza. El escáner tiene un rol fundamental en el proceso de recuperación, ya que es el medio a través del cual se recoge la información de la pieza para determinar el modelo de la pieza sana y su región dañada. Es de gran importancia el poder definir correctamente cómo es el modelo de la pieza sana, como también la región dañada, esto con la finalidad de saber cuál es la región donde se debe hacer la reparación, y hasta que punto se debe realizar la reparación para que así la pieza restaurada coincida con el modelo de la pieza sana. Sin embargo, al momento de recoger esta información mediante un escáner para su posterior procesado en el computador y reconstrucción de la pieza 3D, no hay una forma clara de distinguir cual es una sección sana y una dañada, ya que al momento de visualizar la información se observa tanto puntos de interés o *inliers*, como también puntos que son parte de la señal capturada pero que no se desea expresar en el resultado final (*outliers*).[4]

Estos puntos outlier representan la región dañada de la pieza escaneada, mientras que los inliers representan las zonas sin daño, por lo que para hacer una distinción de estos puntos y así poder recuperar la información que corresponde a la pieza completa sin presencia de daños, es que se opta por un algoritmo iterativo denominado RANSAC (por su nombre en inglés, Random Sample Consensus).

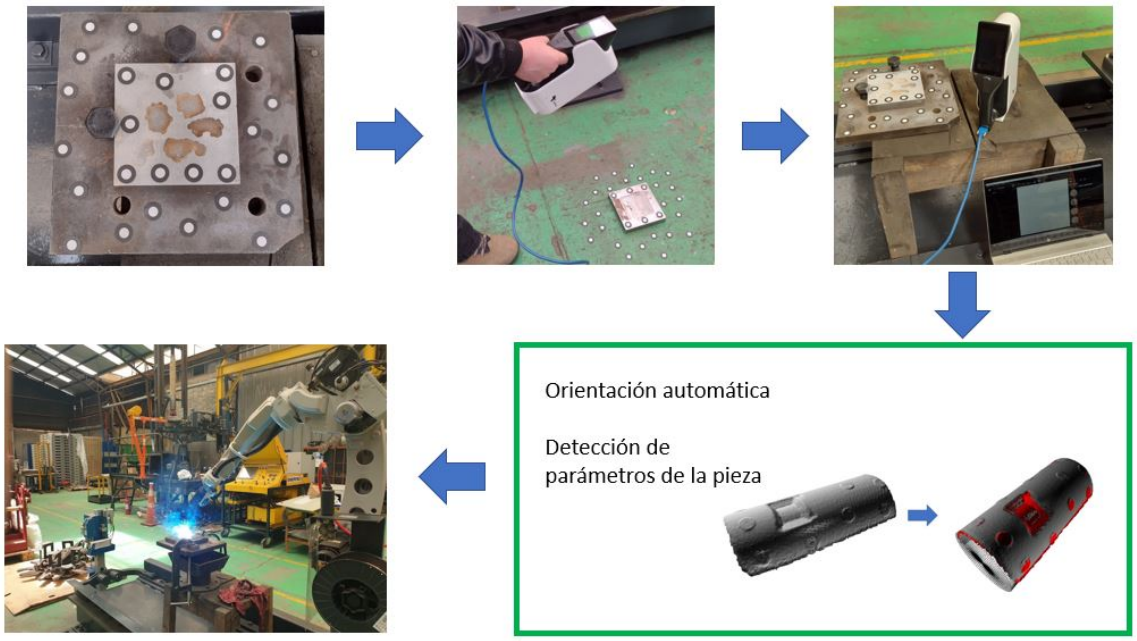


Figura 1.3: Esquema mostrando el flujo de la información, donde la pieza mecánica es escaneada, su información transferida al computador para orientar automáticamente la pieza y determinar sus parámetros, para luego ser traducido al lenguaje del robot para que éste repare la pieza mediante soldadura.

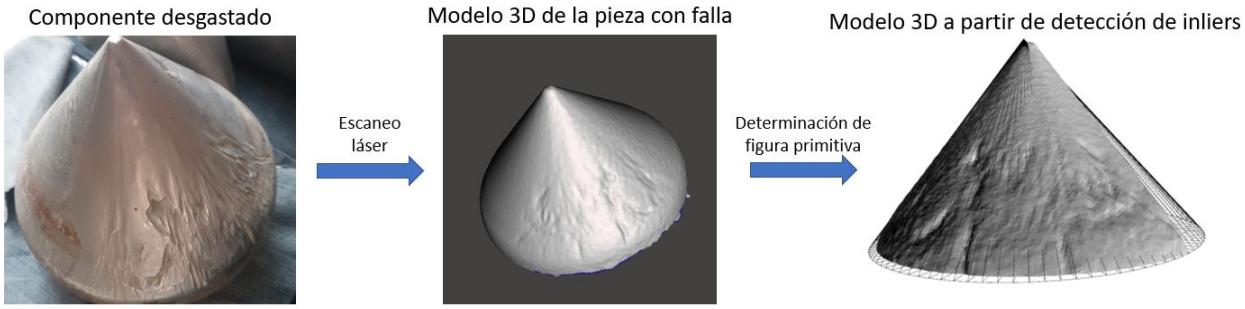


Figura 1.4: Esquema mostrando la determinación de los parámetros

RANSAC mediante iteraciones donde escoge un conjunto de puntos aleatorios, estima cual es el modelo que mejor ajusta a ese conjunto de puntos para luego compararlo con el resto de los puntos del modelo 3D.[5] Este algoritmo es robusto ante la detección de outliers, ya que la correcta detección del modelo que mejor ajusta va asociada a la probabilidad de encontrar un conjunto de puntos tal que todos los puntos sean puntos inlier, a diferencia de como puede ser con otros métodos donde la presencia de outliers causa conflicto.

RANSAC también permite distinguir el ruido que se pueda observar en la señal entregada para el modelo 3D, permitiendo así una detección de los parámetros del modelo 3D a partir de un conjunto de inliers donde no se toman en consideración ni los puntos outliers ni el

ruido que pudiese tener la señal. Esto también depende del tipo de modelo que se utilice para poder modelar una determinada figura, ya que se dispone de varios métodos que ajustan un modelo a un conjunto de datos. Se da la situación que algunos de estos métodos presentan problemas cuando se trata de un conjunto de datos muy grande, principalmente por cómo hacen el ajuste de los datos, por lo que incurren en un alto consumo de recursos computacionales, que terminan repercutiendo en un tiempo de ejecución que a veces resultaría poco práctico a la hora de utilizarlo en la reparación activa de componentes. Es por este motivo que surge la inquietud de desarrollar un método desde una perspectiva más geométrica que permita encontrar la figura primitiva en conjunto al algoritmo de tipo RANSAC, que permitiría distinguir la geometría que se busca, con lo que se haría uso de menor cantidad de recursos computacionales y con ello los tiempos de ejecución serían menores.

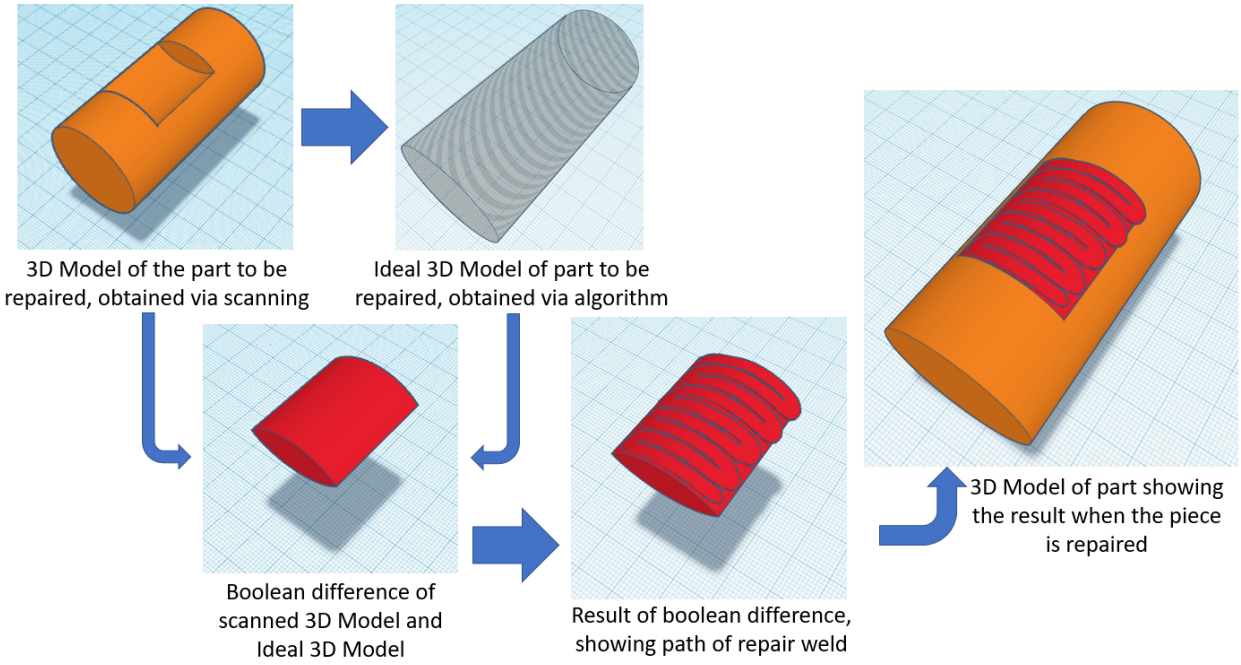


Figura 1.5: Esquema mostrando la diferencia lógica entre el modelo 3D escaneado de la pieza a reparar y el modelo 3D ideal, para luego determinar la ruta que seguirá el robot soldador en la pieza mecánica

Por otro lado, para la reparación de la pieza mecánica no solo es relevante obtener los datos de ésta mediante el escáner 3D y enviar los datos al computador, ya que también se debe tener en consideración la ruta de reparación que seguirá el robot una vez definida cuál es la región de la pieza dañada y cuál es el modelo de la pieza sana. Una vez se dispone del modelo ideal de la pieza, se realiza una diferencia lógica entre el modelo escaneado y el modelo ideal, con lo que se obtiene la región a rellenar con soldadura. Esto se ve ejemplificado en la figura 1.5, donde la sección que resulta de esta diferencia lógica es el segmento en rojo, sobre el cuál posteriormente se define la ruta de la soldadura de reparación, y finalmente se muestra la pieza ya reparada.

Un esquema de como se realiza esta ruta del robot soldador se puede ver a continuación:

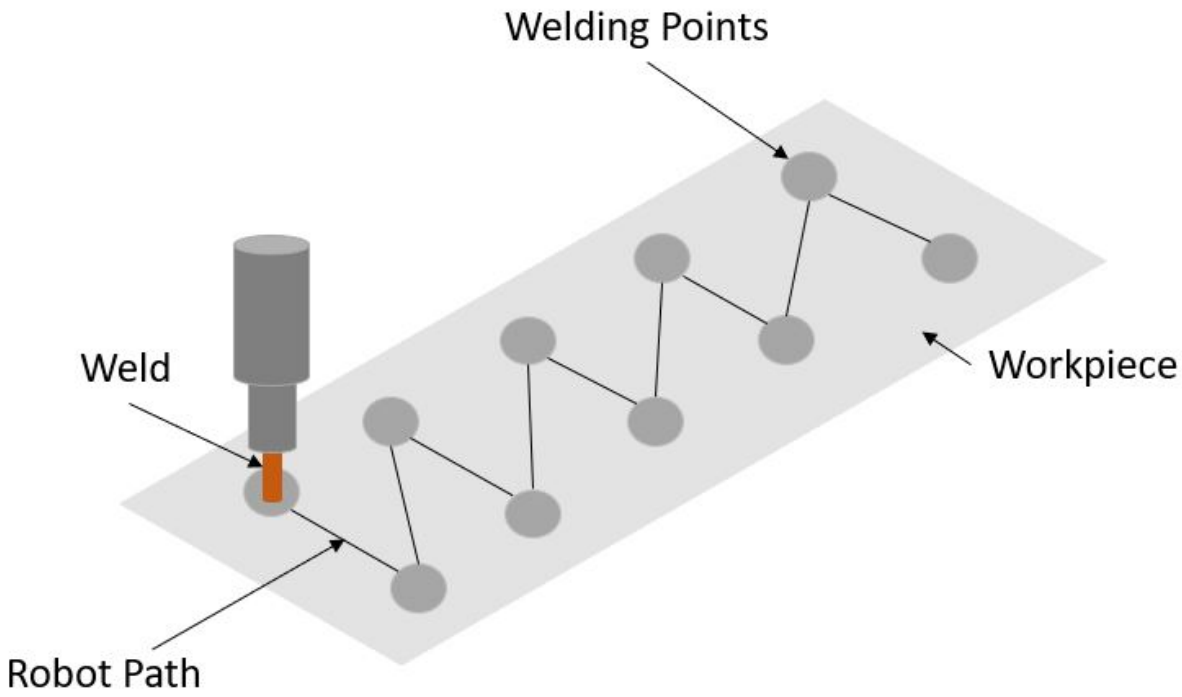


Figura 1.6: Esquema mostrando la ruta de reparación del robot en una pieza mecánica

En la imagen se aprecia un esquema mostrando como sería la reparación en una pieza plana, donde la soldadura va de un punto a otro siguiendo una determinada ruta, la cual es definida mediante software. La pieza mecánica es subdividida en capas, y la ruta del robot es definida para cada una de estas capas, permitiendo así un traslape de los cordones de soldadura y con ello la reparación de la pieza al depositar material y rellenar la totalidad del espacio correspondiente a la región dañada.

1.2. Motivación

La motivación de este trabajo surge desde la necesidad de mejorar la calidad y velocidad en los procesos de recuperación de piezas mecánicas, ya que al reducir el tiempo de ejecución requerido para conocer los parámetros específicos de una pieza a reparar, se logra contar con una reparación más expedita y de mejor calidad en comparación a la que puede brindar un operador realizando soldadura de manera manual, o la que podría conseguir un operador manejando un robot en una reparación supervisada. Si bien esto ya se consigue con metodología actual, el interés de este trabajo está centrado en brindar una propuesta que sea más eficiente y eficaz a la hora de la detección de parámetros principales en una pieza, con la finalidad de poder generar finalmente una representación 3D idealizada de la pieza mecánica a reparar, para así incorporar en el flujo de trabajo a la hora de considerar operaciones como el límite hasta el cuál se llevará a cabo la soldadura o hasta que punto debe ser mecanizada la pieza posterior a su proceso de soldadura.

Conociendo los parámetros de una pieza mecánica, así como una correcta determinación

de su región dañada, permite apuntar a la recuperación de ésta, con miras a una economía circular donde la recuperación de piezas toma un importante rol, al minimizar la producción necesaria de nuevas piezas y aprovechando de mejor forma piezas que de otra forma serían descartadas, como es el caso por ejemplo de herramientas que pierden una parte.

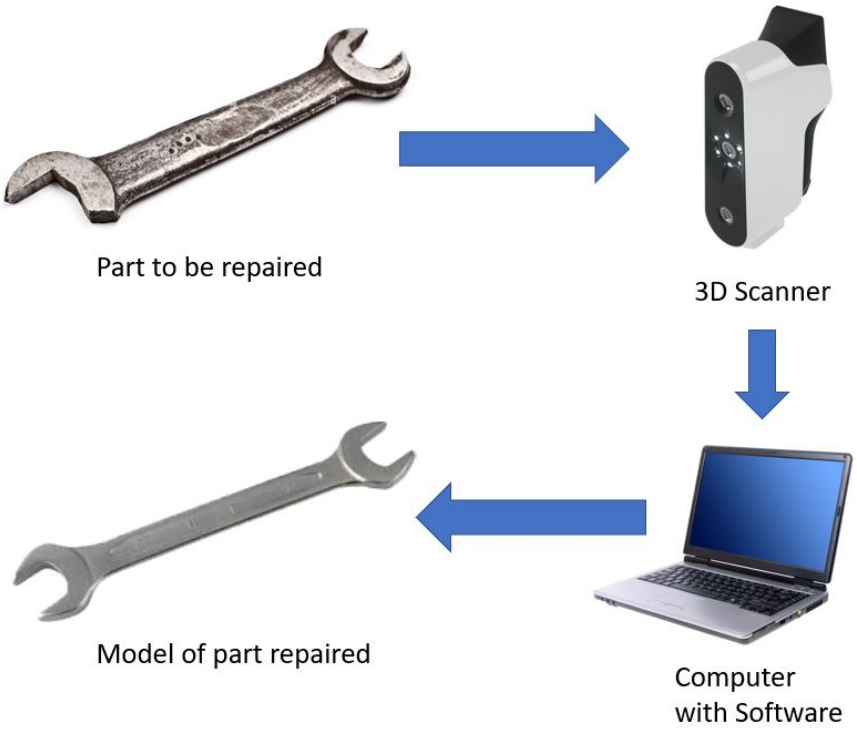


Figura 1.7: Una herramienta con un segmento faltante, es escaneada y su información transmitida al computador, mediante el que se determina cuál es su forma sana.

Disponer de esta capacidad de recuperar piezas a partir de un escáner y un robot permite continuar operaciones en lugares remotos donde la pérdida de una pieza puede significar detenciones en la producción por no tener acceso a adquirir una pieza nueva por temas de lejanía o por la necesidad de restituir la pieza dañada con la mayor velocidad posible, como puede tratarse de las piezas de una instalación petrolífera en mar abierto donde llegar vía aérea o marítima repercute en grandes costos, o en estaciones espaciales donde el costo de enviar piezas de repuesto es aún mayor que para la instalación petrolífera, con lo que la recuperación de piezas es una condición imperativa.

Adicionalmente este trabajo de tesis se ve incorporado dentro de las labores hechas en el Programa de Innovación en Manufactura Avanzada IMA+, bajo el proyecto de "Desarrollo de un sistema automático de fabricación y recuperación de piezas metálicas mediante manufactura aditiva", donde se requiere de manera activa el ser capaz de detectar adecuadamente la orientación de la pieza a reparar, de manera tal que el robot disponga de toda la información pertinente para poder hacer la reparación de la manera más adecuada posible.

1.3. Objetivos

1.3.1. Objetivo general

- Generar una representación 3D idealizada de una pieza mecánica a partir de una nube de puntos de piezas mecánicas cilíndricas y cónicas que presenten desgaste.

1.3.2. Objetivos específicos

- Levantar nubes de punto de piezas mecánicas que presenten fallas en su superficie y puedan ser representados en sus formas primitivas como un cilindro o un cono.
- Detectar mediante algoritmos (RANSAC o sus variantes) la existencia de regiones de puntos sanos y regiones dañadas en el modelo 3D de la pieza a reparar.
- Programar métodos desde la literatura y métodos propios para utilizar en conjunto con RANSAC en la detección de las regiones de puntos sanos y regiones dañadas.
- Evaluar el desempeño del sistema mediante experimentos realizados sobre datos reales y aplicar el sistema en la recuperación de piezas dañadas.
- Contrastar el desempeño del método propuesto contra los métodos programados desde la literatura mediante estadística.

1.4. Alcances y resultados esperados

Se espera desarrollar un método tal que permita obtener el modelo tridimensional original de una pieza mecánica dañada, sin importar la geometría que esta pudiese presentar. Con esto se busca disminuir la cantidad de tiempo asignado a una tarea que en la actualidad está destinada a técnicos profesionales especializados en el área.

Se busca que el método sea robusto sin depender de mayores condicionantes por parte del usuario. Esto quiere decir que se espera que el método sea capaz de recuperar la pieza objetivo indistinto de las condiciones en que se le entregue la nube de puntos, ya sea una pieza sin una orientación determinada, una pieza con curvaturas, o la pieza que sea necesaria recuperar, también indistinto del tamaño.

La metodología desarrollada no busca resolver figuras de alta complejidad en esta etapa, proponiéndolo como un trabajo posterior la incorporación de nuevas geometrías más complejas como puede ser el caso de B-Splines y NURBS.

Capítulo 2

Antecedentes de estudio

2.1. Nubes de Puntos y Mallas de Triángulos

2.1.1. Nube de Puntos

Se entiende por nube de puntos como un conjunto de puntos ubicados en el espacio, que representan la superficie tridimensional de un objeto en computación gráfica.[6] Estas nubes de puntos por lo general implican una gran cantidad de datos, dependiendo de la resolución y densidad de puntos que tenga el escáner al momento de hacer la captura de la información. Uno de los principales atributos a destacar de las nubes de punto es que entrega información respecto de sus puntos no solo en una ubicación de coordenadas XYZ, si no que también es capaz de indicar una orientación del punto mediante la normal que éste posea. Para la determinación de la normal de un punto, uno de los métodos utilizados consiste en la determinación de la normal de un plano tangente a la superficie en cuestión.[7].

La determinación de este plano tangente a la superficie se hace siguiendo un algoritmo, en el que desde el punto en la nube que se desea calcular su normal, se toma una vecindad de puntos que están contenidos en una esfera de un determinado radio, con centro en el punto deseado. Esto corresponde con la distancia euclidiana en un espacio vectorial de p dimensiones, como se refleja en la siguiente ecuación:

$$d(x_i, x_j) = \sqrt{(\sum_{r=1}^{p}(x_{ri} - x_{rj})^2)} \tag{2.1}$$

Este método es usado en otras áreas y es planteado como método de los k vecinos más cercanos, o K-NN por sus siglas en inglés, K-Nearest Neighbours[8].

Ya teniendo una selección de estos puntos de la superficie que están contenidos dentro de la esfera, y por ende se considera que son cercanos, se puede hacer un ajuste de un plano mediante mínimos cuadrados a estos puntos[9], minimizando una función de error para esta función:

$$E(a_0, a_1, b) = \sum_{i=1}^{m}[(a_0 x_i + a_1 y_i + b) - h_i]^2 \tag{2.2}$$

Donde los parámetros a_0, a_1 y b, son los parámetros que se modifican en la función con tal de minimizar el error asociado al ajuste de mínimos cuadrados del plano; y m es la cantidad

de puntos que se consideró para los vecinos más próximos.

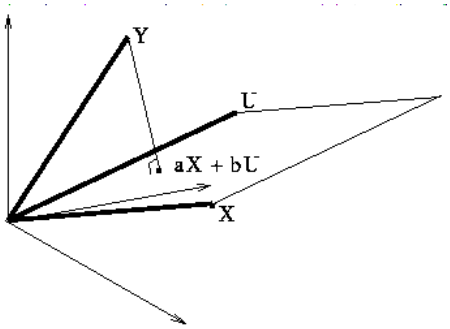


Figura 2.1: Ajuste de mínimos cuadrados a un plano. *Imagen tomada de [10]*

Una vez que ya se ha definido el plano que mejor ajusta al conjunto de puntos, mediante un análisis de componentes principales (PCA) se obtiene la normal del plano, a partir del vector propio asociado al menor de los valores propios de la matriz de covarianza.[11]

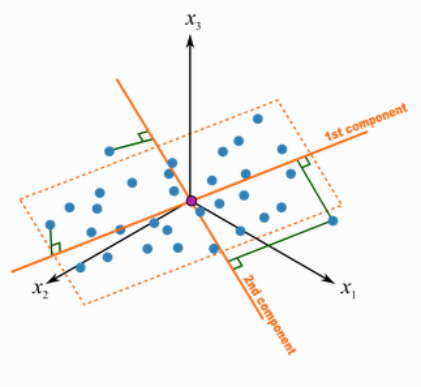


Figura 2.2: Representación de primer y segundo componente en un PCA hecho sobre un conjunto de datos. Geométricamente el tercer componente es el producto cruz de los primeros. *Imagen tomada de [11]*

2.1.2. Mallas de triángulos

Por su parte, también se puede dar el caso que la información sea expresada en forma de una malla de triángulos que define la superficie: Mientras más triángulos tenga, mayor es la definición de la superficie, pero también conlleva un mayor consumo de recursos computacionales al procesar la malla. De manera analógica con la nube de puntos, aquí los vértices de los triángulos pueden ser tratados como los puntos de la nube, mientras que los triángulos mismos ya cumplen con la condición de ser un plano tangente a la superficie, lo cual tiene una normal intrínseca que sirve para determinar otras propiedades en la medida que estas se requieran.

Las propiedades mencionadas como el cálculo de las normales para así definir puntos orientados en el caso de las nubes de puntos, o vértices y normales para una malla de triángulos,

son parámetros que resultan de obtención casi automática en librerías de Python, un lenguaje de programación computacional que será el utilizado durante el desarrollo de este trabajo, junto a las librerías para poder manipular archivos computacionales en que se almacenan estos datos (en el caso de la nube de puntos se guardan en archivos de tipo PCD, mientras que las mallas de triángulos se guardan en archivos tipo STL). Las librerías utilizadas para estos propósitos son Open3D[12] y Trimesh[13], que son específicas para ser utilizadas en conjunto con Python.

A partir de este punto en que las superficies ya pueden ser definidas, ya sea como puntos de una nube o como los vértices de una malla, y sus correspondientes normales, es que se puede buscar información de interés del cuerpo escaneado.

2.2. Detección de *outliers*

Tratándose de nubes de puntos y por ende de datos en general, hay presencia de dos tipos de puntos: Aquellos que son parte de la señal que se quiere representar (*inliers*) y aquellos que no quieren representarse (*outliers*). Hay una necesidad de detectar estos puntos *outlier* para poder distinguirlos de la señal que si se quiere representar, con lo que posteriormente se pueda determinar los parámetros principales de la pieza en el modelo 3D. A continuación se presentan brevemente algunos de estos métodos que pueden ser utilizados en la detección de puntos *outlier*[14]:

- DBSCAN (Density Based Spatial Clustering of Applications with Noise), es un algoritmo de agrupamiento basado en encontrar puntos cercanos por densidad. DBSCAN define tres tipos de puntos:
  - Core Point: En la figura, A es un core point si su vecindad, definida por un valor dado contiene al menos la misma cantidad o más puntos que un parámetro MinPts. Con ello, A forma un cluster con todos los puntos que son alcanzables desde él.
  - Border Point: En la figura, B y C son border points ya que su grupo o cluster contiene menos puntos que lo definido por MinPts, sin embargo son puntos 'alcanzables' por otros puntos en el cluster.
  - Outlier: En la figura, N es un outlier que no pertenece a ningún grupo o cluster, ni es 'alcanzable' por ningún otro punto, por lo que éste será 'su propio cluster'

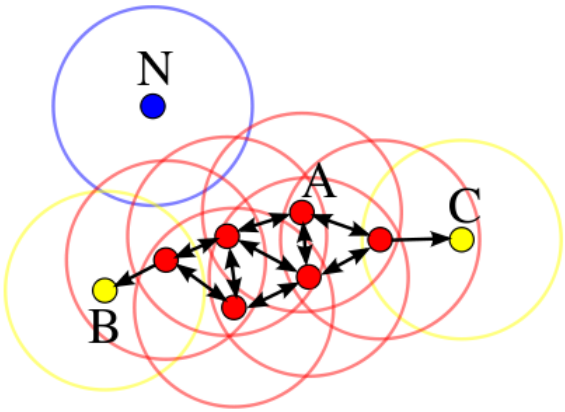


Figura 2.3: Conjunto de puntos explicando DBSCAN *Imagen tomada de [14]*

- Isolation Forest, es un algoritmo donde se aíslan los datos, bajo la premisa que es más fácil separar un punto outlier que uno que si corresponde a datos. Esto se hace de la siguiente forma:
  - Se escoge aleatoriamente un punto a aislar.
  - Para cada característica del punto, se define el rango entre el máximo y el mínimo.
  - Se escoge una característica aleatoriamente del punto.
  - Se escoge un valor aleatoriamente que esté en el rango definido para esa característica (y que difiera del valor del punto a aislar para esa característica).
  - Si el valor escogido hace que el punto quede arriba de la línea divisoria, se cambia el valor mínimo de la característica a ese valor. En caso contrario y quede bajo la línea, se cambia el valor máximo de la característica.
  - El proceso se repite hasta que el punto escogido inicialmente sea el único en el rango.
  - Este punto se considerará outlier si para cumplir lo anterior necesita un número de cortes inferior a un límite impuesto (este límite es un parámetro del algoritmo).

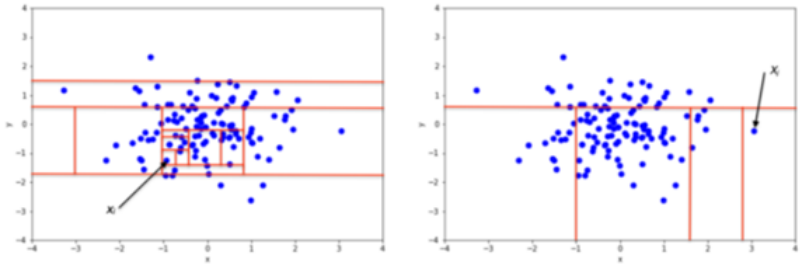


Figura 2.4: Cantidad de cortes para aislar dos puntos, a la izquierda un punto inlier que requiere muchos cortes, a la derecha un outlier que requiere pocos *Imagen tomada de [15]*

Sin embargo, estos son métodos utilizados en áreas como Machine Learning, por lo que no son necesariamente óptimos para el trabajo con nubes de puntos o mallas de triángulos, por lo que se toma la idea de trabajar con un algoritmo distinto, como es el caso del algoritmo RANSAC.

2.3. RANSAC

Dado el requerimiento de estimar los parámetros es que se opta por un algoritmo de RANSAC, que viene de Random Sample Consensus[5]. RANSAC es un algoritmo de tipo iterativo, en el que se toma una selección de puntos del total, y se ajusta un modelo a esta selección de puntos reducida. Luego, se compara el resto de los puntos para ver cuanto difieren del modelo, si la diferencia es menor a un cierto valor de umbral, se considera que este punto es parte del modelo y se considera un punto *inlier*, de lo contrario, se considerará un punto *outlier*. Esto es una metodología que resulta útil al momento de estimar parámetros donde hay presencia de datos que no se quiere representar, o como sería el caso de interés aquí, en una falla. Luego, la motivación al uso de RANSAC en este problema por sobre otras metodologías es por la robustez que tiene a la hora de detectar el modelo que mejor ajusta a los datos del modelo 3D, a pesar de la presencia de outliers, y dado que se está buscando obtener información de como sería el modelo 3D ideal desde los puntos inliers, entonces se sabe de antemano que hay puntos outlier que deben ser diferenciados. A modo de ejemplo, se tiene la siguiente figura:

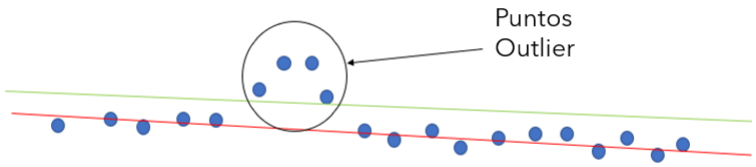


Figura 2.5: Esquema mostrando línea de tendencia con ruido (en verde) y sin ruido (en rojo).

En esta figura se observan dos rectas, una roja y una verde, además de un conjunto de puntos en un plano 2D. Estos puntos podrían representar por ejemplo puntos del suelo desde una vista lateral, con un grupo de puntos que pueden representar un levantamiento en el piso. Si se excluye a estos cuatro puntos que se comportan como *outlier*, al hacer la línea de tendencia de la totalidad de los puntos (verde), coincidiría con la línea roja, que está considerando solamente los puntos que antes se mencionó como *inlier*. Para poder hacer esto, se ejecuta el algoritmo de manera iterativa seleccionando un grupo de puntos de manera aleatoria en cada iteración, para así calcular modelos que ajusten al conjunto de datos. En una primera iteración de RANSAC, podría darse esta configuración:

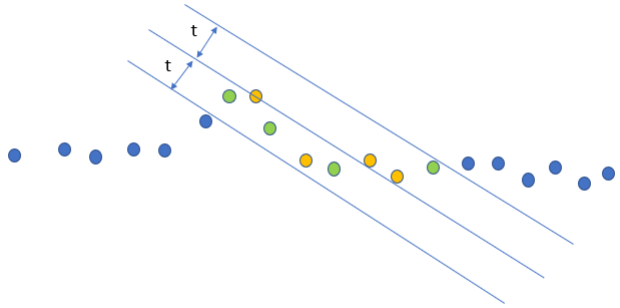


Figura 2.6: Esquema mostrando una primera iteración de RANSAC.

Aquí se escogió un total de cuatro puntos del total, y mediante una regresión lineal se determina la línea de tendencia que se corresponde con estos puntos. Una vez se dispone de esta línea de tendencia, se observa cuantos de los puntos del universo completo de puntos están a una distancia igual o menor a t con respecto a la línea de tendencia: Si los puntos cumplen con la condición indicada, son considerados *inlier*, y en el esquema son marcados como puntos verdes. Luego, el conteo total de puntos que califican como *inlier* será la suma de todos estos puntos, en este caso un total de 8, recordando que los puntos escogidos al inicio de la iteración también cumplen la condición de estar a una distancia igual o menor a t con respecto a la línea de tendencia, o sea son *inlier*. Se guarda la información de este modelo ya que al ser la primera iteración, no hay un mejor candidato, y se pasa a una segunda iteración:

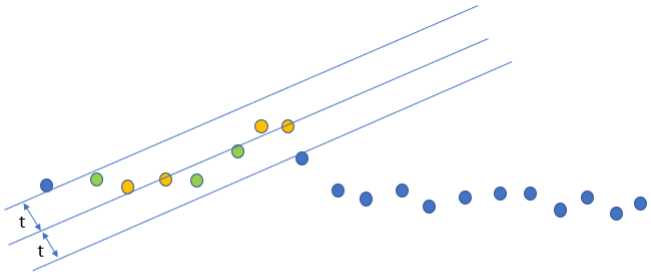


Figura 2.7: Esquema mostrando una segunda iteración de RANSAC.

En esta segunda iteración se escoge un grupo de cuatro puntos aleatorios y se repite el proceso, haciendo una regresión lineal y obteniendo una línea de tendencia nueva, que se usa como referencia para ver cuántos puntos están a una distancia t de la línea de tendencia, y se observa que son 7 los puntos que califican como *inlier*, por lo que se procede a comparar el modelo actual contra el modelo que está guardado. Como el modelo de la primera iteración calificó a ocho puntos como *inlier*, se define que el modelo que ya estaba guardado sigue siendo mejor, por lo que no se guarda el modelo de la segunda iteración y se conserva el de la primera iteración. En una siguiente iteración, que podría ser la tercera como una iteración número 100, se podría dar este escenario:

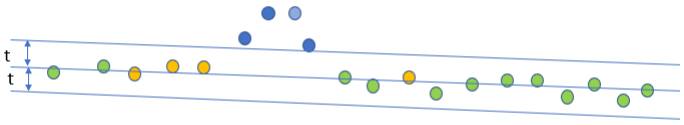


Figura 2.8: Esquema mostrando una iteración posterior de RANSAC.

Es en esta configuración donde se escogió aleatoriamente un conjunto de puntos tal, que al hacer la regresión lineal sobre estos puntos, se obtiene una línea de tendencia tal que 16 de los puntos son considerados como *inlier*. Dado que este modelo ajusta mejor al universo de puntos, se descarta el modelo de la primera iteración y se guarda este modelo como el válido. Si en iteraciones posteriores resulta no aparecer otro modelo que tenga mejor ajuste, entonces el obtenido en esta iteración representaría el modelo que ajusta a estos datos. Todo el algoritmo se representa en el siguiente diagrama de flujo para una mayor facilidad de seguimiento:

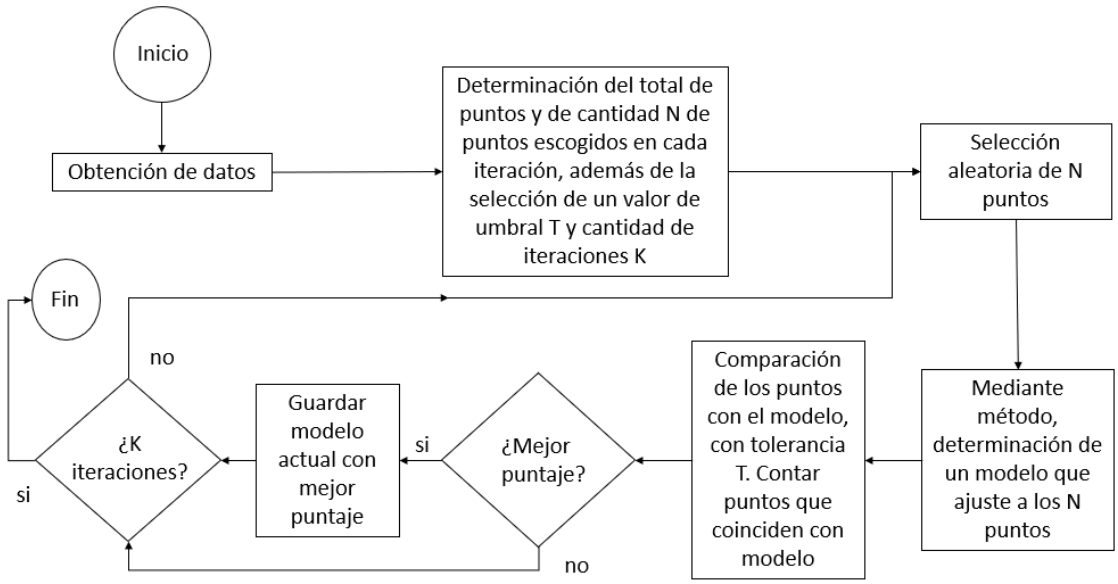


Figura 2.9: Diagrama de flujo representando el algoritmo RANSAC.

En el diagrama de flujo se observa que se determina un parámetro K, que es la cantidad de iteraciones que se realizarán antes de dar por finalizada la búsqueda del modelo que mejor ajuste. La elección de este parámetro no es arbitraria: De escogerse un valor bajo de iteraciones, es poco probable que se encuentre una combinación de candidatos tal que se detecte de manera adecuada el modelo que ajusta a la figura, mientras que un número elevado de iteraciones permitirá con mayor probabilidad de éxito el encontrar una combinación de

candidatos que detecten la forma, a costa de recurso computacional. Esto se debe a que hay una probabilidad asociada a la elección de puntos como podría ser en el caso de la detección de un cilindro o un cono: Si se considera una nube de puntos de un tamaño N y una figura consistente de n puntos, y se considera que r define la cantidad mínima de puntos para definir un candidato, entonces la probabilidad de detectar el modelo que ajusta correctamente en una iteración es:[20]

$$P(n) = \binom{n}{r} / \binom{N}{r} \approx (\frac{n}{N})^r \tag{2.3}$$

Desde aquí, la probabilidad de una detección exitosa $P(n, s)$ después de s conjuntos de puntos han sido seleccionado es igual al complemento de s fallos consecutivos:

$$P(n, s) = 1 - (1 - P(n))^s \tag{2.4}$$

Si se resuelve la ecuación anterior para el valor de s, se obtiene el número de intentos K requeridos para detectar una forma de tamaño n con una probabilidad $P(n, T) \geq$p$_t$:

$$K \geq \frac{ln(1 - p_t)}{ln(1 - P(n))} \tag{2.5}$$

Para valores pequeños de $P(n)$ el logaritmo en el denominador puede ser aproximado por su serie de Taylor $ln(1 - (P(n)) = -P(n) + O(P(n)^2)$, por lo que:

$$K \approx \frac{-ln(1 - p_t)}{P(n)} \tag{2.6}$$

Estas probabilidades permiten definir un valor adecuado para la cantidad de iteraciones, pero los otros valores, tanto la cantidad de puntos que se elige como el valor de umbral T, son parámetros que irán asociados más al modelo que se desea seguir para ajustar los datos, esto ya que el ajuste de un cono con el de un cilindro difieren en su forma de ajustar, sino que también dentro de los ajustes de un cilindro existen varias formas de modelar el problema como se ha apreciado en la literatura, de igual forma para el cono.

2.4. Modelos actuales que ajustan a geometrías cilíndricas y cónicas

RANSAC es el algoritmo con el que se busca obtener los parámetros, pero para poder definir si los datos corresponden o no a un punto *inlier*, hace falta primero un modelo que ajuste a los datos que se están estudiando. Dado lo transversal del problema de detectar cuerpos cuya forma se corresponde a una figura básica como un cilindro o un cono, es que en la literatura se puede encontrar diversas formas en que se ha abordado el problema, mediante distintos modelos dependiendo de si se trata de un cilindro o de un cono.

2.4.1. Modelos en cilindros

En uno de los métodos consultados en la literatura[16], se tiene el siguiente método de ajuste para los datos en cada iteración realizada:

1. Encontrar la orientación $\vec{a}$ del eje del cilindro mediante el vector propio correspondiente al menor de los valores propios de la matriz de covarianza de los vectores normales de los *inliers*. En este método la selección de puntos en cada iteración determina un punto inicial y una vecindad de puntos alrededor del inicial, y estos son los considerados como *inlier*.

2. Proyectar los puntos *inlier* sobre un plano con normal $\vec{a}$ y que pasa por el origen O.

3. Ajustar un círculo a los puntos proyectados para encontrar el radio y un punto p perteneciente al eje del cilindro.

4. Usar un filtro normal y de distancia para actualizar los puntos *inlier* para la siguiente iteración.

El primer paso es calcular la orientación del eje del cilindro, y eso es el vector más ortogonal a todos los vectores normales $\vec{n_i}$ de los *inliers*. Por lo que una matriz semi-positiva

$$C = \sum_{i=1}^{n_{inliers}} n_i^T n_i \tag{2.7}$$

es calculada y analizada, en donde cada n_i es un vector-fila. Luego el vector propio de C correspondiente al valor propio más pequeño es considerado como el eje de orientación $\vec{a}$, y C_x con C_y los otros vectores propios que crean un sistema de coordenadas en el plano.

En el segundo paso, $\tilde{p_i}$ es la proyección 2D de los puntos *inlier* en el plano con normal $\vec{a}$ que pasa por el origen, estos puntos se distribuyen en el plano como un círculo, por lo que el problema de determinar el radio y un punto en el eje se reducen a encontrar el radio y centro del círculo.

En otro de los métodos consultados en la literatura[17] se propone una forma de caracterizar un cilindro a partir de la ubicación del punto con respecto a regiones de Voronoi del cilindro.

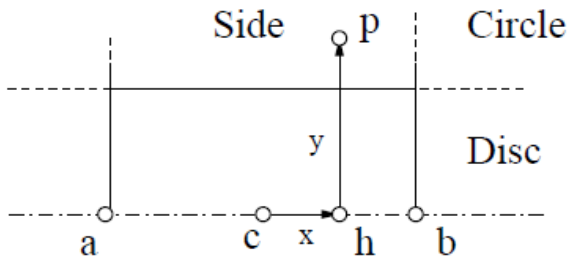


Figura 2.10: Regiones de Voronoi de un cilindro. *Imagen tomada de [17]*

Donde **a** y **b** denotan los vertices del final del eje, y r denota el radio del cilindro. El punto **c** es el centro del segmento [**a**,**b**], luego la distancia entre el punto **p** y el cilindro requiere clasificar en que región voronoi se encuentra para definir la fórmula que se aplicará, dado que la distancia del punto **p** sobre el eje x será:

$$x = (\mathbf{c} - \mathbf{p}) \cdot (\mathbf{b} - \mathbf{a}) / \parallel \mathbf{b} - \mathbf{a} \parallel \tag{2.8}$$

La distancia al cuadrado al vértice se denotará como $n^2 = (\mathbf{c} - \mathbf{p})^2$, y la distancia al cuadrado al eje desde el punto p se da por $y^2 = \| \mathbf{p} - \mathbf{h} \|^2 = n^2 - x^2$, donde **h** es la proyección ortogonal de p sobre el eje. Ya definidos estos valores, el punto p tiene una distancia al cilindro dependiendo de en qué región se encuentre:

- Si $|\mathbf{x}| < l/2$, con $l = \| \mathbf{b} - \mathbf{a} \|$, entonces **h** está contenido en el segmento de línea [**ab**]
  - Si $y^2 < r^2$, entonces p es un punto al interior del cilindro y la distancia es 0. Aquí se está asumiendo un cilindro macizo, no un manto de cilindro.
  - En caso contrario, la distancia al cuadrado al cilindro es $(y - r)^2$
- Por otro lado, si $|x| > l/2$, entonces lo que se calcula es la distancia de un punto a un disco.
  - Si $y^2 < r^2$ entonces **p** se proyecta sobre el disco y su distancia al cuadrado es simplemente $(|x| - l/s)^2$
  - Por el contrario, si $y^2 > r^2$, entonces **p** se proyecta sobre un círculo y la distancia al cuadrado es $(y - r)^2 + (|x| - l/2)^2$

Se observaron otras propuestas de ajuste de modelo para un cilindro en la literatura[18] mediante distintas configuraciones de puntos, siendo posibles soluciones mediante la selección desde 6 hasta 9 puntos, generando una variable cantidad de soluciones posibles, teniendo en particular que al escoger 7 o 9 puntos, la solución encontrada es única. Sin embargo estas soluciones en esencia también ocupan una metodología parecida a la del primer método planteado, al proyectar los puntos sobre un plano para obtener una representación del cilindro en forma de círculo proyectado en el plano.

2.4.2. Modelos en Conos

De igual forma, para el ajuste de conos se observó que en la literatura uno de los métodos en [17] también presenta un modelo para ajuste de conos. En este modelo, se tiene la siguiente región de voronoi:

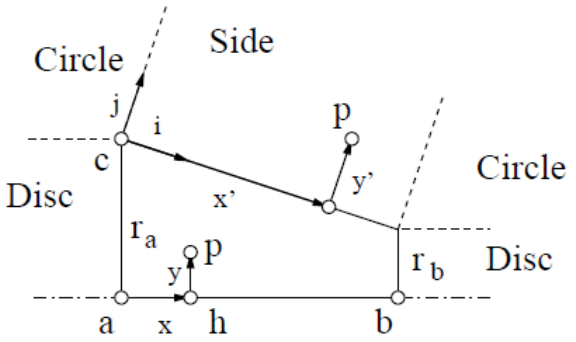


Figura 2.11: Regiones de Voronoi de un cono. *Imagen tomada de [17]*

Donde **a** y **b** denotan los vértices del eje, con r_a y r_b los radios correspondientes, y además $\delta = r_a$ - r_b. El largo de la generatriz del cono se denota por $s = (l^2 + \delta^2)^{1/2}$.

Dada la configuración del cono, se requiere un cambio de coordenadas que sea solidario al manto del cono, pero primero se debe verificar en que tramo se encuentra el punto **p** con respecto al eje x. La distancia del punto **p** sobre el eje será $x = (\mathbf{a} - \mathbf{p}) \cdot (\mathbf{b} - \mathbf{a}) / \parallel \mathbf{b} - \mathbf{a} \parallel$. La distancia al vértice **a** será $n^2 = (\mathbf{a} - \mathbf{p})^2$, y la distancia al eje al cuadrado se denota por $y^2 = \parallel \mathbf{p} - \mathbf{h} \parallel^2 = n^2 - x^2$. Definidos estos valores y de igual forma que para la región de voronoi de cilindros, el procedimiento varía dependiendo de en qué región se encuentre:

- Si x < 0, entonces:
  - Si $y^2 < r_a^2$ entonces **p** se proyecta en la base de mayor radio del cono y su distancia al cuadrado es de x^2
  - Si es el caso contrario, la distancia cuadrada al círculo será $(y - r_a)^2 + x^2$
- Si $y^2 < r_b^2$, entonces hay otros dos casos:
  - Si x $>$ l entonces **p** se proyecta sobre la base de menor radio del cono y su distancia cuadrada será $(x - l)^2$
  - En el caso contrario, **p** es un punto contenido dentro del cono y por tanto su distancia es 0.

Luego para hacer la rotación se tienen nuevos vectores ortonormales, denotados por (**i**,**j**), y estos son:

$$\mathbf{i} = (\frac{\delta}{s}, \frac{l}{s}); \mathbf{j} = (\frac{l}{s}, \frac{-\delta}{s}) \tag{2.9}$$

Sean las nuevas coordenadas del punto **p** los valores (y',x'), el algoritmo ahora sigue:

- Si x' < 0, entonces **p** se proyecta sobre el círculo de la base de radio mayor y su distancia al cuadrado es $(y - r_a)^2 + x^2$
- Si x' $>$ s, entonces **p** se proyecta sobre el círculo de la base de radio menor y su distancia al cuadrado es $y'^2 + (x' - s)^2$
- Si no es ninguno de estos dos escenarios, entonces **p** está por el lado del cono y su distancia es y'^2

Otro método revisado[19] propone utilizar una minimización de mínimos cuadrados sobre la función de distancias de un punto al cono, donde la distancia de un punto al cono se puede ver en la siguiente figura:

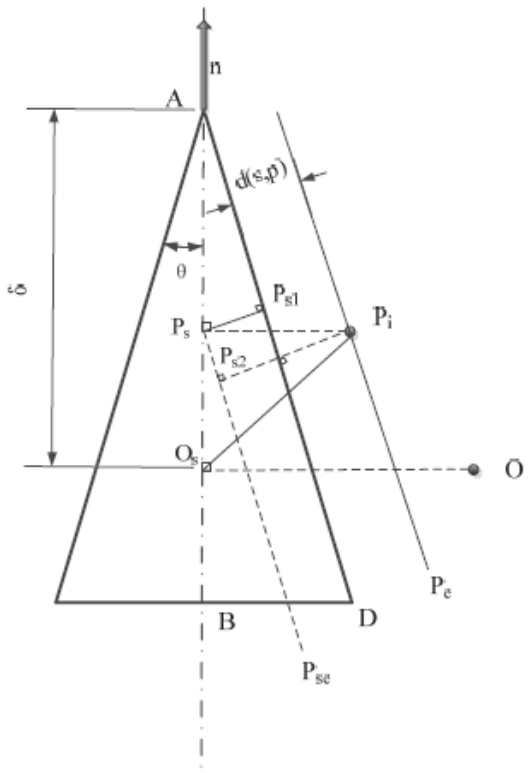


Figura 2.12: Modelo de la función de distancia de un punto a un cono. *Imagen tomada de [19]*

Donde $d(s, p_i)$ es la distancia entre un punto tridimensional y la superficie cónica; O es el origen de coordenadas, O_s es la proyección del origen en el eje del cono, δ es la distancia desde el O_s al vértice del cono A; P_s es la proyección espacial del punto P_i en el eje del cono; θ es la mitad del ángulo de apertura del cono; $\vec{n}$ es el vector axial unitario del cono; y las relaciones relevantes en el modelo son:

$$P_s P_{se} // P_i P_e // AD \tag{2.10}$$

$$P_s P_{sl} \perp AD \tag{2.11}$$

$$P_i P_{s2} \perp AD \tag{2.12}$$

Con lo que la función de distancia de un punto a la superficie del cono se expresa como:

$$d(s, p_i) = |P_i\vec{P}_{s2}| - |P_s\vec{P}_{s1}| = |P_i\vec{P}_s| \cdot cos\theta - |A_i\vec{P}_s| \cdot sin\theta \tag{2.13}$$

Finalmente, otro método utilizado para el ajuste de un conjunto de puntos a un cono es como el que se menciona[20] es:

- Se hace una selección de 3 puntos aleatorios orientados
- Se generan 3 planos donde cada uno pasa por un punto y con normal solidaria a la del punto.

- El vértice del cono se determina por la intersección de estos 3 planos.
- El eje del cono se determina desde la normal de un plano definido por tres puntos $c + (p_1 - c)/ \parallel p_1 - c \parallel, c + (p_2 - c)/ \parallel p_2 - c \parallel, c + (p_3 - c)/ \parallel p_3 - c \parallel$, con c el vértice del cono, y p_1, p_2 y p_3 los puntos elegidos aleatoriamente.
- El ángulo de apertura ω está dado por el promedio del arccoseno del producto punto entre (p_i-c) y a, donde a es el eje del cono, y p_i son los 3 puntos elegidos.

$$\omega = \sum_{i=1}^{3} arccos((p_i - c) \cdot a) \tag{2.14}$$

Capítulo 3

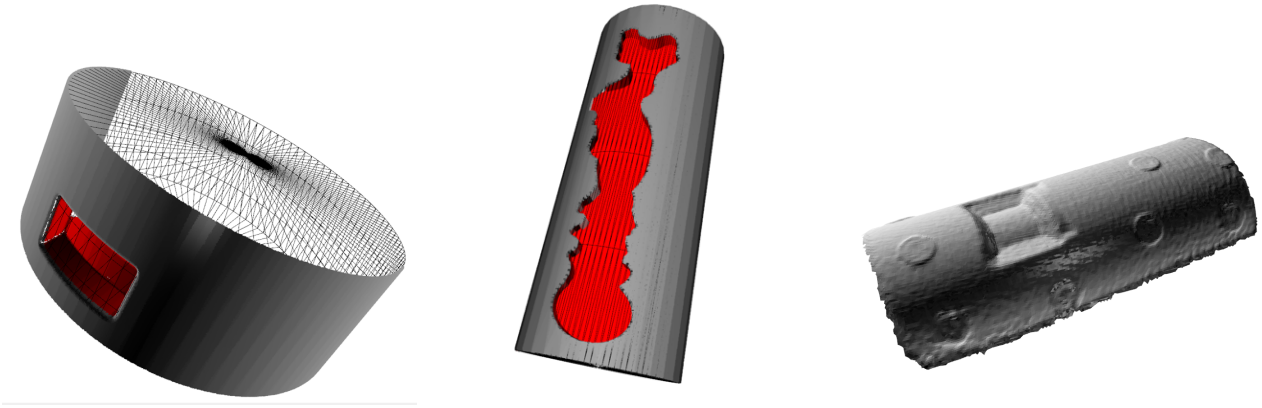
Metodología

3.1. Preparación de piezas dañadas

Para trabajar con los métodos planteados se hace uso de Fusion 360 (software de tipo CAD/CAM que permite el diseño de piezas según requerimiento) como base para poder generar piezas dañadas de cilindro y de cono, ya que al hacerlo de manera digital se dispone de una mayor variedad de potenciales piezas dañadas que si solo se dependiera de piezas físicas presencialmente. Luego de generarse mediante Fusion 360 como archivos STL, son procesadas de tal manera que se puede hacer un sampling de los datos a partir de la malla de triángulos guardada en el archivo STL, con lo que estos archivos STL pueden ser utilizados en formato PCD. Adicionalmente, se dispone de un archivo de manto de cilindro escaneado desde una pieza real y un archivo de la superficie de un cono, también escaneado desde una pieza real. Disponiendo ya de los archivos en el formato deseado, estos son cargados en el interprete de Python mediante librerías como Open3D o trimesh.

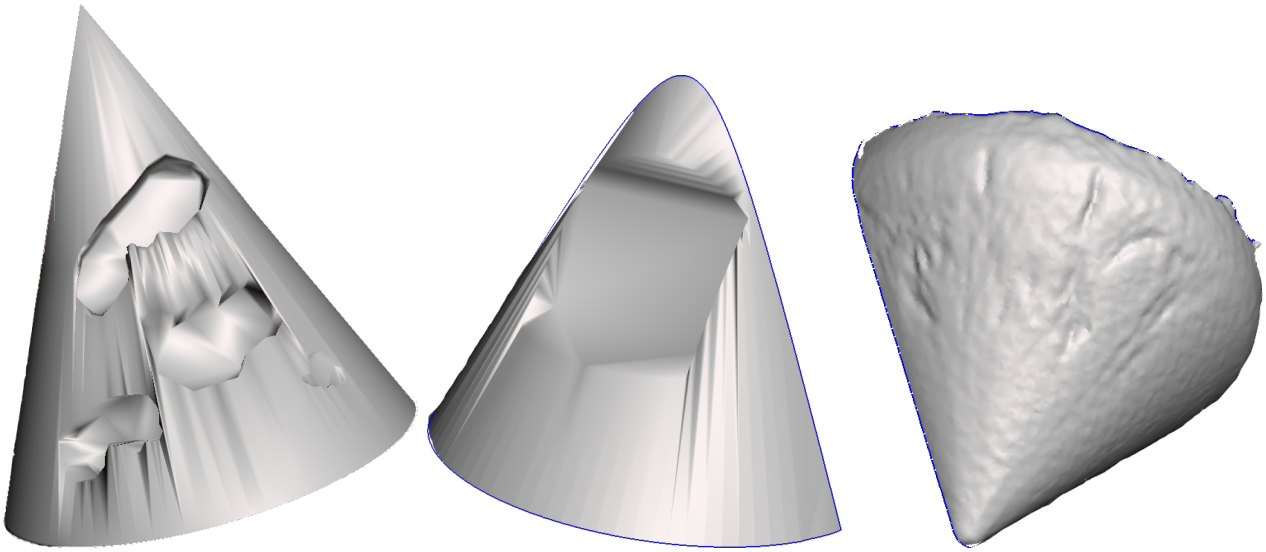
Se dispone del siguiente listado de configuraciones de cilindros y conos:

- Cilindro con falla tipo bolsillo, radio mayor que largo del cilindro. Mitad de manto. Archivo tipo PCD.
- Cilindro con falla tipo pitting, largo mayor que radio. Mitad de manto. Archivo tipo PCD.
- Cilindro con falla tipo bolsillo, largo mayor que radio. Mitad de manto. Archivo STL hecho a partir de un escaneo.
- Cono con falla tipo pitting, simulada mediante Fusion360. Archivo tipo STL.
- Cono con falla tipo bolsillo, simulada mediante Fusion360. Archivo tipo STL.
- Manto de una válvula con falla por despunte y con presencia de venas, escaneada desde una pieza real. Archivo STL.



(a) (b) (c)

Figura 3.1: (a) Cilindro con falla tipo bolsillo, PCD. (b) Cilindro con falla tipo pitting, PCD. (c) Cilindro con falla tipo bolsillo, STL



(a) (b) (c)

Figura 3.2: (a) Cono con falla tipo pitting, STL simulado. (b) Cono con falla tipo bolsillo, STL simulado. (c) Cono escaneado desde una válvula despuntada y con venas, STL

Figura 3.3: Válvula desde la que se hizo el escaneo para usar como archivo STL según lo que se muestra en la figura 3.2.c.

Cada una de las configuraciones se realiza con orientaciones arbitrarias, pero en particular buscando que no sean orientaciones paralelas a ninguno de los ejes principales, para evitar dependencia de los cálculos con respecto al eje.

3.2. Preprocesamiento y parámetros a utilizar durante iteraciones de RANSAC

Desde este punto, antes de hacer el ajuste mediante RANSAC se hace un pre procesado de los datos, el cual varía en sus métodos dependiendo si se trata de un archivo tipo PCD o de uno STL. Esta etapa de preprocesamiento es común a todos los métodos para determinar parámetros que se plantearon, tanto los de la literatura como el método propio, por lo que se presenta este esquema general:

- Se centra la nube de puntos o malla de triángulos.
- Se calculan las dimensiones generales del volumen que ocupa la pieza de manera general, a partir del valor máximo y mínimo que ocupan los puntos de la nube o los vértices en la malla de triángulos para cada uno de los 3 ejes. Este dato se usa como parámetro de referencia durante la iteración, de manera que si al hacer el ajuste del modelo se encuentra una figura con un tamaño que supere los calculados en este punto, sea descartado sin gastar tiempo de ejecución en verificar puntos *inlier*

Otro aspecto que tiene que ser definido antes de la ejecución del algoritmo RANSAC, es la definición de los parámetros de iteración, en específico se determina la cantidad de

iteraciones que se harán, el valor de umbral con el que se considerará si un punto pertenece o no al modelo (para considerarlo un punto inlier), y la cantidad de puntos que se selecciona en cada iteración. Los parámetros a utilizar para este trabajo son:

- Cantidad de Iteraciones: 1.000. Se define esta cantidad de iteraciones ya que ante un número elevado de iteraciones aumenta la probabilidad de escoger un conjunto de puntos que permitan definir el modelo que mejor ajusta al modelo 3D, y con ello la determinación de los parámetros buscados.
- Valor de Umbral: 0.1 % del valor de la máxima dimensión del modelo 3D, es decir si por ejemplo se define que el modelo en el espacio tiene una dimensión de 300 mm. en el eje X, 350 mm. en el eje Y, y 320 mm. en el eje Z, la dimensión más grande es el eje Y con 350 mm, por ende el valor de umbral será 0.1 % de estos 350 mm. Este es un valor dinámico ya que no se puede aplicar un mismo valor de umbral para un modelo que mide 20 mm. que para uno de 500 mm.
- Cantidad de puntos elegidos al azar en cada iteración: 3, se escoge esta cantidad dado que es el mínimo que permite determinar un plano completamente.

Por último, se necesita definir la forma en que se determinará si el punto de la nube de puntos califica como *inlier* para el modelo propuesto por el método. Para la metodología de este trabajo se determina la pertenencia de un punto al modelo mediante una función de distancia: en el caso de cilindros, se calcula la distancia del punto de la nube al eje calculado del cilindro, y si esta distancia es igual al radio calculado del cilindro (con una tolerancia definida por el valor de umbral), el punto se considera *inlier*; para conos, se calcula la distancia del punto en la nube a un cono con vértice y orientación definidos por método, y si esta distancia es cercana a cero (también con tolerancia definida por el valor de umbral), entonces el punto se considera *inlier*.

3.3. Programación de códigos para métodos de literatura

Para realizar este trabajo y dado que consiste en el desarrollo de un método, primero se implementan versiones preliminares de código en las que se usan métodos ya existentes en la literatura, los cuales se pueden ver en detalle en la sección de discusión bibliográfica.

- Método para la detección de parámetros de un cilindro a partir la minimización de una función de mínimos cuadrados para la distancia de los puntos a un cilindro. En este método se escoge el conjunto de puntos, se determina un punto inicial al promediar la posición de estos 3 puntos en el espacio, y se determina la orientación inicial como el vector asociado al mayor de los valores propios determinados luego de hacer la matriz de covarianza sobre la totalidad de los puntos (esta orientación inicial es única y no se calcula en cada iteración). El radio inicial se calcula como el promedio de la distancia de los 3 puntos seleccionados a un eje que pasa por el punto inicial con una orientación paralela a la orientación inicial. Luego con estos valores iniciales, se hace una minimización de la función de mínimos cuadrados para la distancia de todos los puntos de la nube usando estos valores iniciales, para luego definir si los puntos son inlier o no.

- Método para la detección de parámetros de un cilindro a partir de proyectar una selección de puntos sobre un plano con normal paralela al eje del cilindro. Se determina una orientación inicial $\vec{a}$ mediante el vector propio correspondiente al menor de los valores propios de la matriz de covarianza de los vectores normales de los 3 puntos seleccionados en la iteración, para luego proyectar los 3 puntos sobre un plano que tenga esta orientación y pase por el origen del sistema de coordenadas. Posterior a esto, se ajusta un círculo a los puntos proyectados en el plano desde el cual se determina el radio del cilindro.

- Método para la detección de parámetros en un cono a partir de la minimización de una función de mínimos cuadrados para la distancia de los puntos a un cono. La función de distancia de un punto a un cono se determina a partir de lo mostrado en la figura (2.12), sobre la cual se hace la minimización de la función de mínimos cuadrados.

- Método para la detección de parámetros en un cono a partir de seleccionar 3 puntos orientados, desde los cuales se generan 3 planos cuya intersección coincide con el vértice del cono, y el eje del cono corresponde a una línea que pasa por el vértice del cono, y es paralela a la normal de un plano generado a partir de los 3 puntos seleccionados.

Se presenta un pseudo-código para cada uno de los métodos recién mencionados, mientras que el código correspondiente a cada uno de estos métodos puede consultarse en el anexo de este documento.

Algoritmo 1 Determinación de parámetros de un cilindro mediante minimización de función de mínimos cuadrados.

Require: Puntos, vector desde matriz de covarianza, parámetros RANSAC.
Ensure: Los puntos son puntos orientados.

```
for i ∈ 1, ..., N do                                   ▷ N iteraciones de RANSAC
    p1, p2, p3 ← 3 puntos seleccionados al azar
    P0 ← Calcular punto medio de los 3 puntos orientados
    O0 ← Vector                                        ▷ Orientación inicial.
    R0 ← Distancia de los puntos a una línea que pasa por P0 con orientación O0
    P, O, R ← Minimizar mínimos cuadrados para distancia.
    for i ∈ 1, ..., T do                               ▷ Para cada punto del modelo 3D
        Ri ← Distancia del punto a línea que pasa por P con orientación O
        if Ri ∈ [R · (1 − t), R · (1 + t)] then        ▷ t valor de umbral
            S += 1                                     ▷ Suma 1 al contador de puntos inlier
        end if
    end for
    if S > PrevS then
        Guardar modelo actual, cambiar valor de PrevS por S.
    end if
end for
```

Algoritmo 2 Determinación de parámetros de un cilindro mediante la proyección de los puntos en un plano formando un círculo.

Require: Puntos, parámetros RANSAC.
Ensure: Los puntos son puntos orientados.

```
for i ∈ 1, ..., N do                                   ▷ N iteraciones de RANSAC
    p1, p2, p3 ← 3 puntos seleccionados al azar
    a⃗ ← Vector asociado a menor valor propio de matriz de covarianza de los 3 puntos.
    π ← Plano que pasa por el origen con normal paralela a a⃗
    p̂1, p̂2, p̂3 ← Proyección de p1, p2, p3 en π
    C ← Ajustar círculo a los puntos p̂1, p̂2, p̂3
    R, P ← Radio y centro de C
    O ← a⃗
    for i ∈ 1, ..., T do                               ▷ Para cada punto del modelo 3D
        R_i ← Distancia del punto a línea que pasa por P con orientación O
        if R_i ∈ [R · (1 − t), R · (1 + t)] then       ▷ t valor de umbral
            S += 1                                     ▷ Suma 1 al contador de puntos inlier
        end if
    end for
    if S > PrevS then
        Guardar modelo actual, cambiar valor de PrevS por S.
    end if
end for
```

Algoritmo 3 Determinación de parámetros de un cono mediante minimización de función de mínimos cuadrados.

Require: Puntos, parámetros RANSAC.
Ensure: Los puntos son puntos orientados.
for $i \in 1, ..., N$ **do** ▷ N iteraciones de RANSAC
  for $j \in 1, ..., 9$ **do**
    $p_j \leftarrow$ 9 puntos seleccionados al azar
  end for
  $P_0 \leftarrow$ Calcular punto medio de los 9 puntos orientados
  $\vec{a} \leftarrow$ Vector asociado a menor valor propio de matriz de covarianza de los 9 puntos.
  $O_0 \leftarrow \vec{a}$ ▷ Orientación inicial.
  $A_0 \leftarrow \pi/4$ ▷ Ángulo de apertura inicial en $\pi/4$
  $A, P, O \leftarrow$ Minimizar mínimos cuadrados para distancia. ▷ Ecuación 2.13
  for $i \in 1, ..., T$ **do** ▷ Para cada punto del modelo 3D
    $d_i \leftarrow$ Distancia del punto al cono con ángulo de apertura A, vértice en P, y orientación O ▷ Ecuación 2.13
    if $d_i \in [-t, t]$ **then** ▷ t valor de umbral
      S += 1 ▷ Suma 1 al contador de puntos inlier
    end if
  end for
  if S > PrevS **then**
    Guardar modelo actual, cambiar valor de PrevS por S.
  end if
end for

Algoritmo 4 Determinación de parámetros de un cono mediante la intersección de 3 planos y la generación de un plano a partir de 3 puntos.

Require: Puntos, parámetros RANSAC.
Ensure: Los puntos son puntos orientados.
for $i \in 1, ..., N$ **do** ▷ N iteraciones de RANSAC
$p_1, p_2, p_3 \leftarrow$ 3 puntos seleccionados al azar
$n_1, n_2, n_3 \leftarrow$ Normales de los 3 puntos seleccionados
$\pi_1, \pi_2, \pi_3 \leftarrow$ Planos que pasan por los puntos con normales n_1, n_2, n_3 respectivamente
$P \leftarrow$ Intersección de los planos π_1, π_2, π_3
$\pi_4 \leftarrow$ Plano que pasa por los 3 puntos seleccionados.
$O \leftarrow$ Normal del plano π_4
$A \leftarrow$ Ángulo de apertura ▷ Ecuación 2.14
for $i \in 1, ..., T$ **do** ▷ Para cada punto del modelo 3D
$d_i \leftarrow$ Distancia del punto al cono con ángulo de apertura A, vértice en P, y orientación O ▷ Ecuación 2.13
if $d_i \in [-t, t]$ **then** ▷ t valor de umbral
S += 1 ▷ Suma 1 al contador de puntos inlier
end if
end for
if S > PrevS **then**
Guardar modelo actual, cambiar valor de PrevS por S.
end if
end for

3.4. Programación y desarrollo de método propio

El método propuesto en este trabajo se enfoca en la obtención de un modelo que ajuste a la figura, ya sea un cilindro o un cono, mediante cálculos geométricos por sobre cálculos matemáticos. Si bien los métodos de cilindro y de cono son similares entre si, se describirán por separado ya que existen diferencias a considerar entre ambos métodos finalmente. A pesar de ello, se expone en resumen el procedimiento para cada método:

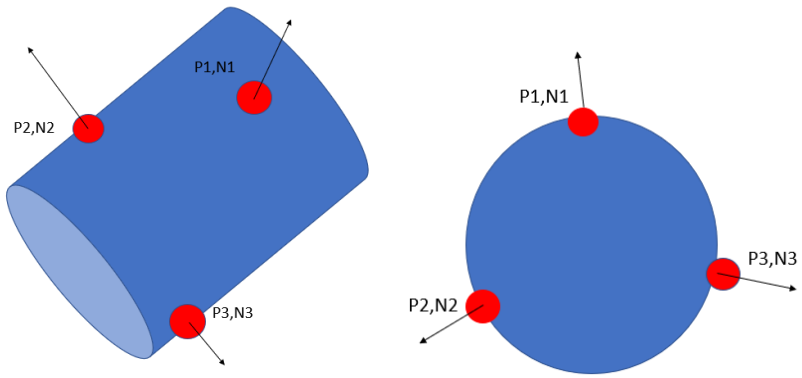
- Para el cilindro:
  - Se seleccionan 3 puntos orientados.
  - Se calcula el producto cruz entre las normales de los puntos.
  - Se intersectan planos creados por las normales y el producto cruz, que pasan por los puntos originales. Estos planos deben intersectarse en el centro del cilindro con lo que se obtiene el eje.
  - Se calcula el radio del cilindro como la distancia de los puntos seleccionados a este eje.
- Para el cono:
  - Se seleccionan 3 puntos orientados.

- Se intersectan planos que pasan por estos puntos con normales paralelas a las de los puntos orientados. La intersección de los puntos debe corresponder al vértice del cono.
- Se trazan rectas paralelas a las normales de los puntos seleccionados y se intersectan. La intersección de existir se ubica sobre el eje del cono.
- Se calcula el ángulo de apertura mediante trigonometría dado que se conoce la distancia de los puntos seleccionados al vértice y al eje del cono.

A continuación, se presenta el método propuesto para cilindro y para cono, acompañado de un pseudo código. El código completo para estos métodos puede encontrarse en el Anexo de este documento.

3.4.1. Método para cilindro

Sean 3 puntos orientados elegidos de manera aleatoria sobre la superficie del cilindro, como se muestra en la figura (3.4).



(a) (b)

Figura 3.4: (a) 3 puntos aleatorios en el manto de un cilindro, vista sin un eje específico. (b) 3 puntos aleatorios en el manto de un cilindro, vista axial.

A partir de las normales respectivas de estos puntos, N_1 N_2 y N_3 siguiendo la notación planteada en la figura, se calcula el producto cruz de estas normales formando pares, para luego calcular el promedio de estos:

$$v_1 = N_1 \times N_2, v_2 = N_1 \times N_3, v_3 = N_2 \times N_3 \tag{3.1}$$

$$N_0 = \frac{1}{3} \cdot (v_1 + v_2 + v_3) \tag{3.2}$$

Desde aquí se tiene una primera orientación, N_0, que será utilizada para calcular nuevamente productos cruz. No se conserva esta orientación N_0 como la definitiva del cilindro ya que durante el desarrollo del trabajo se pudo apreciar que se perdía precisión en la respuesta obtenida y que no era consistente, causando que no siempre entregara un resultado deseable.

$$NP_1 = N_1 \times N_0, NP_2 = N_2 \times N_0, NP_3 = N_3 \times N_0 \tag{3.3}$$

A partir de los valores recién calculados, se definen planos con normales NP_1, NP_2 y NP_3, que pasan por los puntos P_1, P_2 y P_3, respectivamente. Luego, las ecuaciones de estos planos serán:

$$\pi_1 =< NP_1 > \cdot < P_1 > \Rightarrow < a_1, b_1, c_1 > \cdot < x_1, y_1, z_1 > \Rightarrow a_1 \cdot x_1 + b_1 \cdot y_1 + c_1 \cdot z_1 = 0 \quad (3.4)$$

$$\pi_2 =< NP_2 > \cdot < P_2 > \Rightarrow < a_2, b_2, c_2 > \cdot < x_2, y_2, z_2 > \Rightarrow a_2 \cdot x_2 + b_2 \cdot y_2 + c_2 \cdot z_2 = 0 \quad (3.5)$$

$$\pi_3 =< NP_3 > \cdot < P_3 > \Rightarrow < a_3, b_3, c_3 > \cdot < x_3, y_3, z_3 > \Rightarrow a_3 \cdot x_3 + b_3 \cdot y_3 + c_3 \cdot z_3 = 0 \quad (3.6)$$

Teniendo estos planos, se calcula la intersección de ellos, nuevamente en pares y posteriormente se calcula su promedio:

$$L_1 = \pi_1 \cap \pi_2, L_2 = \pi_1 \cap \pi_3, L_3 = \pi_2 \cap \pi_3 \quad (3.7)$$

La intersección de dos planos resulta en una línea, esta línea coincide con el eje del cilindro, pero para dar más robustez al cálculo, se considera el promedio de las líneas L_1, L_2 y L_3 como la línea que representa al eje del cilindro.

$$L = \frac{1}{3} \cdot (L_1 + L_2 + L_3) \quad (3.8)$$

Esta línea en el espacio representa al eje, con su orientación en el espacio R^3. A partir de este punto ya se dispone del eje del cilindro y con ello la orientación que este tiene en el espacio. Para poder obtener el radio del cilindro, se calcula la distancia de cada uno de los puntos elegidos aleatoriamente (P_1, P_2 y P_3) al eje recién calculado, mediante la fórmula:

$$d(P_i, L) = \frac{\bar{P_0 P_i} \times \bar{s}}{\bar{s}}; i = 1, 2, 3 \quad (3.9)$$

Donde $\bar{s}$ es la orientación de la línea L, y P_0 es un punto en dicha línea.

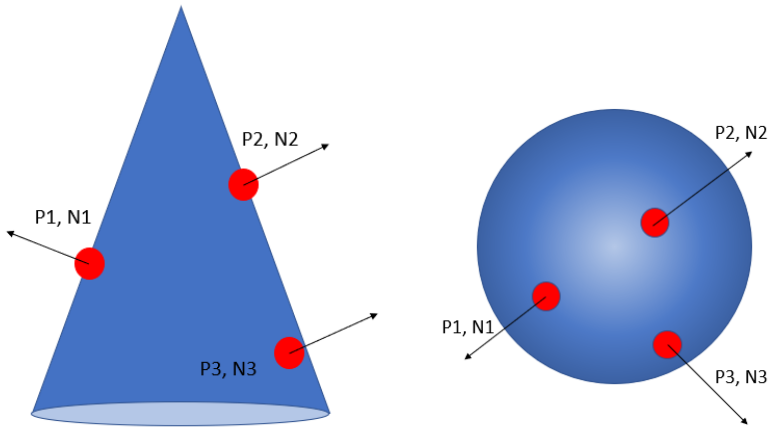
Teniendo los parámetros que definen el eje del cilindro y su radio, se puede comparar el resto de los puntos del PCD o STL, midiendo sus distancias al eje y si están a una distancia que sea igual al radio calculado (con una tolerancia definida al inicio del proceso), entonces se consideran como puntos *inlier*.

Algoritmo 5 Método propuesto para determinación de parámetros en un cilindro.

Require: Puntos, parámetros RANSAC.
Ensure: Los puntos son puntos orientados.
for $i \in 1, ..., N$ **do** ▷ N iteraciones de RANSAC
$p_1, p_2, p_3 \leftarrow$ 3 puntos seleccionados al azar
$n_1, n_2, n_3 \leftarrow$ Normales de los 3 puntos seleccionados
$n_0 \leftarrow$ Promedio de $n_1 \times n_2$, $n_1 \times n_3$ y $n_2 \times n_3$
$n_4, n_5, n_6 \leftarrow n_1 \times n_0$, $n_2 \times n_0$ y $n_3 \times n_0$
$\pi_1, \pi_2, \pi_3 \leftarrow$ Planos que pasan por los puntos p_1, p_2, p_3 con normales n_4, n_5, n_6 respectivamente
$L_1, L_2, L_3 \leftarrow \pi_1 \cap \pi_2, \pi_1 \cap \pi_3, \pi_2 \cap \pi_3$
$O \leftarrow$ Dirección Promedio de L_1, L_2 y L_3
$P \leftarrow$ Punto que está en la línea con orientación O
$R \leftarrow$ Distancia de los 3 puntos a O
for $i \in 1, ..., T$ **do** ▷ Para cada punto del modelo 3D
$R_i \leftarrow$ Distancia del punto a línea que pasa por P con orientación O
if $R_i \in [R \cdot (1-t), R \cdot (1+t)]$ **then** ▷ t valor de umbral
S += 1 ▷ Suma 1 al contador de puntos inlier
end if
end for
if S > PrevS **then**
Guardar modelo actual, cambiar valor de PrevS por S.
end if
end for

3.4.2. Método para cono

Sean 3 puntos orientados elegidos de manera aleatoria sobre la superficie del cono, como se muestra en la figura (4.2)



(a) (b)

Figura 3.5: (a) 3 puntos aleatorios en el manto de un cono, vista lateral. (b) 3 puntos aleatorios en el manto de un cono, vista axial.

Se definen los planos con normales N_1 N_2 y N_3, que pasan por los puntos P_1 P_2 y P_3, respectivamente.

$$\pi_1 =< N_1 > \cdot < P_1 > \Rightarrow < a_1, b_1, c_1 > \cdot < x_1, y_1, z_1 > \Rightarrow a_1 \cdot x_1 + b_1 \cdot y_1 + c_1 \cdot z_1 = 0 \quad (3.10)$$

$$\pi_2 =< N_2 > \cdot < P_2 > \Rightarrow < a_2, b_2, c_2 > \cdot < x_2, y_2, z_2 > \Rightarrow a_2 \cdot x_2 + b_2 \cdot y_2 + c_2 \cdot z_2 = 0 \quad (3.11)$$

$$\pi_3 =< N_3 > \cdot < P_3 > \Rightarrow < a_3, b_3, c_3 > \cdot < x_3, y_3, z_3 > \Rightarrow a_3 \cdot x_3 + b_3 \cdot y_3 + c_3 \cdot z_3 = 0 \quad (3.12)$$

Geométricamente, el vértice del cono corresponde a la intersección de éstos 3 planos, por lo que para encontrar las coordenadas de este vértice, se plantea resolver la ecuación matricial de la forma $Ax = b$, donde:

$$A = \begin{pmatrix} N_1 \\ N_2 \\ N_3 \end{pmatrix} = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix} \quad (3.13)$$

$$x = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \quad (3.14)$$

$$b = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{pmatrix} \quad (3.15)$$

$$Ax = b \Rightarrow \begin{pmatrix} N_1 \\ N_2 \\ N_3 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{pmatrix} \Rightarrow \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{pmatrix} \quad (3.16)$$

Luego, si los 3 puntos elegidos son puntos que corresponden a la superficie sana del cono, o dicho de otra forma, los 3 puntos elegidos aleatoriamente son puntos *inlier*, entonces la solución a la ecuación matricial corresponde al vértice del cono. En caso de tratarse de 3 puntos que no sean sanos, la solución a la ecuación podría no existir; en esos casos, esa combinación de puntos es descartada y se pasa a una siguiente iteración.

A continuación, se generan líneas que pasen por 2 de los puntos elegidos. Sin pérdida de generalidad, se escogen el primer y segundo puntos, P_1 y P_2. Desde estos puntos y con sus respectivas normales, se trazan rectas L_1 y L_2 con dirección igual a sus normales, y se busca el punto de intersección entre ambas: La intersección de ambas coincide en un punto del eje del cilindro, pero solo en el caso de que los puntos P_1 y P_2 pertenezcan a un mismo plano paralelo al plano basal del cono; si los puntos no cumplen esta condición, la intersección no existe y esta combinación de puntos es descartada, pasando a la siguiente iteración como ocurre para la resolución de la ecuación matricial.

$$L_1 \Rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} = P_1 + \alpha \cdot N_1 \Rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} P_1(x) \\ P_1(y) \\ P_1(z) \end{pmatrix} + \alpha \cdot \begin{pmatrix} N_1(x) \\ N_1(y) \\ N_1(z) \end{pmatrix} \quad (3.17)$$

$$L_2 \Rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} = P_2 + \alpha \cdot N_2 \Rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} P_2(x) \\ P_2(y) \\ P_2(z) \end{pmatrix} + \alpha \cdot \begin{pmatrix} N_2(x) \\ N_2(y) \\ N_2(z) \end{pmatrix} \tag{3.18}$$

$$P_4 = L_1 \cap L_2 \tag{3.19}$$

Al haber obtenido el vértice y un punto sobre el eje del cono, se puede definir la dirección del eje del cono al calcular el vector unitario a partir de estos puntos:

$$s = \frac{(P_0 - P_4)}{\| P_0 - P_4 \|} \tag{3.20}$$

Donde P_0 corresponde al vértice del cono calculado desde la ecuación matricial, y P_4 al punto de intersección de las líneas L_1 y L_2.

Ya definido el vértice del cono y su eje, se puede calcular el ángulo de apertura del cono para así tener su último parámetro. Como se observa en la figura (4.3), el ángulo β se calcula como el $arcsin(u/v)$, donde u es la distancia de uno de los 3 puntos elegidos aleatoriamente al inicio de la iteración al eje, y v es la distancia de este punto al vértice. Éste ángulo se calcula para los 3 puntos elegidos aleatoriamente y luego se promedian para así obtener el ángulo de apertura con una mayor precisión.

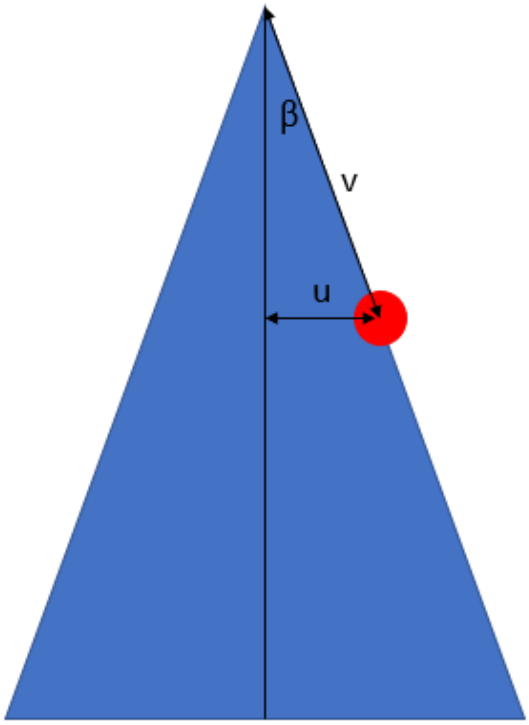


Figura 3.6: Esquema de valores para calcular el ángulo de apertura de un cono.

Definidos los parámetros del cono, se hace una comparación de todos los puntos a este cono. Para esto se usa la función de distancias de un punto al cono que se vio en la literatura [19], donde usaban la función para luego calcular los parámetros mediante minimización de mínimos cuadrados. Si la distancia calculada con esta función es cero o muy pequeña según

el valor de umbral escogido para la tolerancia, este punto se considera como *inlier*, de lo contrario califica como punto de zona dañada.

Algoritmo 6 Método propuesto para determinación de parámetros en un cono.

Require: Puntos, parámetros RANSAC.
Ensure: Los puntos son puntos orientados.
  for $i \in 1, ..., N$ **do** ▷ N iteraciones de RANSAC
    $p_1, p_2, p_3 \leftarrow$ 3 puntos seleccionados al azar
    $n_1, n_2, n_3 \leftarrow$ Normales de los 3 puntos seleccionados
    $\pi_1, \pi_2, \pi_3 \leftarrow$ Planos que pasan por los puntos con normales n_1, n_2, n_3 respectivamente
    $P \leftarrow$ Intersección de los planos π_1, π_2, π_3
    $p_4 \leftarrow$ Intersección de normales n_1 y n_2
    $O \leftarrow$ Dirección de vector que va desde P a p_4
    $A \leftarrow$ Ángulo de apertura ▷ Por trigonometría
    for $i \in 1, ..., T$ **do** ▷ Para cada punto del modelo 3D
      $d_i \leftarrow$ Distancia del punto al cono con ángulo de apertura A, vértice en P, y orientación O ▷ Ecuación 2.13
      if $d_i \in [-t, t]$ **then** ▷ t valor de umbral
        S += 1 ▷ Suma 1 al contador de puntos inlier
      end if
    end for
    if S > PrevS **then**
      Guardar modelo actual, cambiar valor de PrevS por S.
    end if
  end for

3.5. Evaluación

Cada configuración de cilindros y conos será evaluada con los métodos previamente programados y con el método propuesto en este trabajo. La evaluación consiste en ejecutar el método 2.000 veces en un mismo computador con procesador AMD Ryzen 7 5800X 3.8 GHz y 16 Gb. de memoria RAM, para que las capacidades del mismo no tengan influencia en que un método tenga una velocidad de ejecución mayor a la de otro método, y obtener un registro estadístico de los valores obtenidos para la orientación del eje y radio del cilindro, así como orientación y ángulo de apertura en cada una de estas 2.000 ejecuciones. Además, tanto para cilindro como para cono, se tomará registro del tiempo requerido para ejecutar el método en cada una de las veces.

Como se trata de piezas hechas mediante computadora y sus orientaciones son definidas de igual forma, se dispone de estos parámetros con antelación, por lo que todos estos parámetros obtenidos resultan comparables a los parámetros reales, y con ello se puede determinar el error asociado a cada método al calcular el radio del cilindro o el ángulo de apertura del cono. Este error porcentual se calcula siguiendo la siguiente ecuación:

$$ErrorPorcentual = \frac{|V_a - V_v|}{V_v} \cdot 100 \tag{3.21}$$

Con V_a el valor aproximado obtenido en cada ejecución del respectivo método, y V_v el valor verdadero que se conoce.

En cuanto a la diferencia en la orientación calculada para el modelo 3D en comparación a la orientación real que esta posee, se puede determinar despejando el valor de α para la siguiente ecuación:

$$cos(\alpha) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| \cdot |\vec{v}|} \tag{3.22}$$

Donde $\vec{u}$ es la orientación calculada mediante el método, y $\vec{v}$ es la orientación real conocida de antemano.

Los parámetros estadísticos se midieron sobre los datos con un valor de n igual a 2.000, esto es considerando todos los valores obtenidos. Dichos parámetros estadísticos son: la media muestral de cada método, con un valor

$$\mu = \frac{1}{n}\sum_{i=1}^{n} X_i = \frac{X_1 + X_2 + ... + X_n}{n} \tag{3.23}$$

La desviación estándar sobre todos los datos de la muestra, usando la fórmula:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{3.24}$$

Con la desviación estándar y la media muestral, y considerando un tamaño de muestra de 2.000 datos, se determinaron intervalos de confianza para los datos con una confiabilidad al 95 %m siguiendo una distribución normal con media μ y una desviación típica $\sigma/\sqrt{n}$, donde el con una confiabilidad del 95 % se puede afirmar que los valores calculados están contenidos en intervalos de la forma:

$$I_c = [\mu - z_{\alpha/2}\frac{\sigma}{\sqrt{n}}, \mu + z_{\alpha/2}\frac{\sigma}{\sqrt{n}}] \tag{3.25}$$

Donde I_c es el Intervalo de Confianza para una confiabilidad del 95 %, y el término $z_{\alpha/2}$ corresponde al valor crítico obtenido desde las tablas de distribución normal para un determinado valor de α, el nivel de significación. En específico, para un α de 0.05 se tiene que el valor crítico $z_{\alpha/2}$ es de 1.96. Estos intervalos de confianza fueron calculados para determinar con una confiabilidad del 95 % en que rango se encuentran los valores para los parámetros obtenidos, en el caso del cilindro su radio, su orientación, el tiempo de ejecución y el error porcentual con el que se calculó el radio. Para los conos por su parte, se calculan los intervalos de confianza para el ángulo de apertura, la orientación, el tiempo de ejecución y el error porcentual con el que se calculó el ángulo de apertura.

Capítulo 4

Resultados y discusiones

El primer resultado de este trabajo es el código para los métodos propuestos, que puede ser consultado en el Anexo de este documento. Seguido de esto, se presentan los resultados obtenidos de aplicar la metodología recién expuesta. También se exponen resultados en la detección de los puntos outlier y la generación de las figuras primitivas que se obtiene a partir de considerar solo los puntos inlier del modelo 3D.

El código programado al terminar su ejecución entrega dos resultados, el primero es en la forma de datos numéricos (los parámetros solicitados), mientras que el segundo resultado es visual, al mostrar como es la nube de puntos completa, marcando los puntos outlier con otro color y además generando una malla sobre la que se posiciona el modelo 3D, mostrando con ello como sería la figura a la que pertenecería este modelo 3D, asumiendo una figura cilíndrica o cónica según sea el caso.

4.1. Métodos para detección de parámetros en cilindros

A continuación, se presentan tablas haciendo comparativas de las estadísticas obtenidas con cada uno de los métodos, para cada tipo de cilindro utilizado. Por un tema de notación y para mayor facilidad a la hora del manejo de la información, en las tablas se hablará de Método Propuesto, Método Mínimo y Método Proyección, donde Método Mínimo corresponde al método de detección de parámetros en cilindro a partir de minimizar la función de mínimos cuadrados para la distancia, mientras que el Método Proyección es el método de detección de parámetros en cilindro que proyecta el conjunto de puntos sobre un plano para formar un círculo y determinar parámetros a partir del círculo. Hecho este alcance, los datos se exponen en tablas y gráficos para su atención:

4.1.1. Comparación cilindro con falla tipo bolsillo

- Modelo 3D: Cilindro con falla tipo bolsillo radial
- Radio: 500 [mm]
- Orientación: [-0.707 , 1 , -0.707]

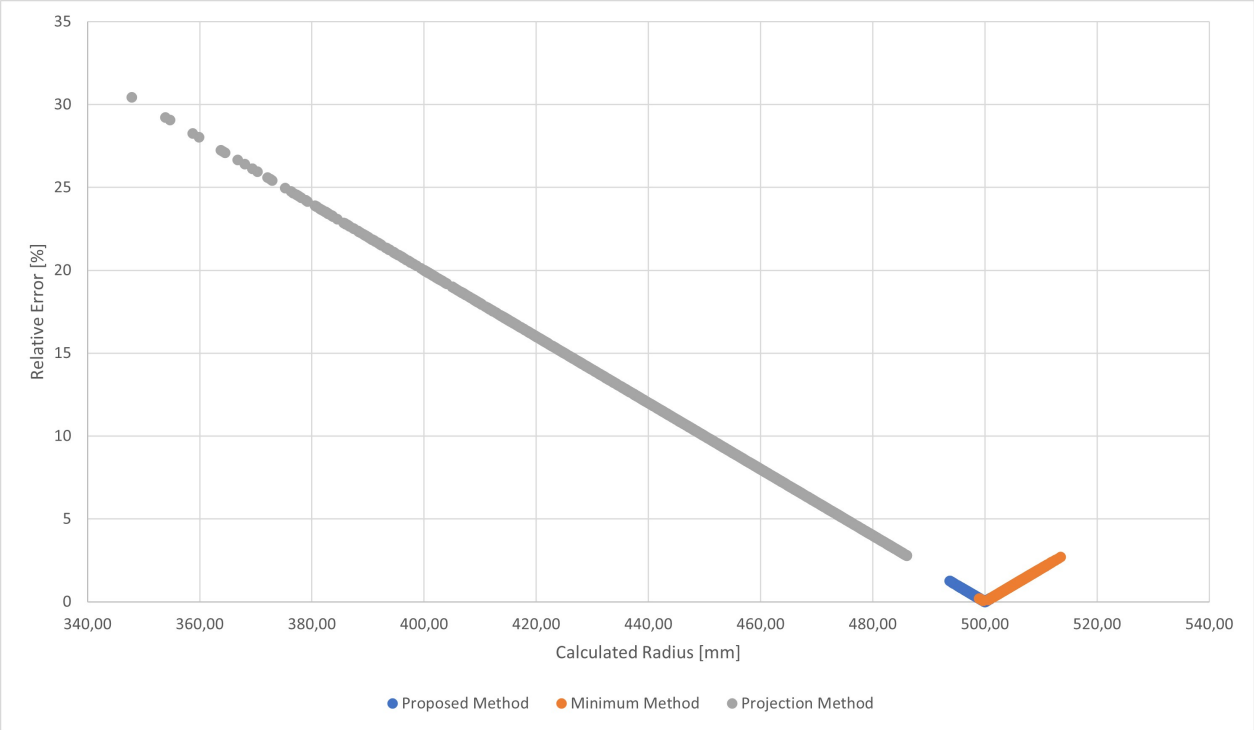


Figura 4.1: Gráfico mostrando error relativo porcentual para el radio calculado en cada método.

En la figura 4.1 se expone el error relativo que presentaron los métodos para las 2.000 veces que fueron ejecutados cada método. De este gráfico se observa que tanto el método propuesto como el método mínimo tienen un bajo error relativo a la hora de calcular el radio de la figura, además de estar concentrado en un rango pequeño, ya que en ninguno de los dos casos su error relativo alcanza el 2.5 %. Por otro lado, el método proyección tiene una gran dispersión en sus resultados, que van en su mejor resultado con 2.5 % hasta más de un 30 % de error relativo.

El ángulo entre la orientación calculada y la real conocida previamente se presenta en el siguiente gráfico de dispersión, mostrando en una escala semi logarítmica, donde el eje vertical el ángulo está en escala logarítmica (en grados), y en el eje horizontal se tiene el radio calculado para el mismo modelo en una escala lineal. En el gráfico se aprecia que a la vez que el método propuesto y el método minimización tienen un bajo error al momento de determinar la orientación del cilindro, el método proyección presenta una gran dispersión en cuanto a la orientación que detecta, la cual muestra una mayor variedad en la medida que se acerca al valor teórico, pero nunca en pocos casos logra determinar la orientación con menos de 10 grados de diferencia, en cambio el método propuesto con el método minimización tienen errores en la orientación que están en torno a un grado.

Tanto la información vista en el gráfico 4.1 como el 4.2 se resume en la tabla 4.1, donde se muestran intervalos de confianza al 95 % para el radio, el error relativo porcentual al calcular este radio y los intervalos de confianza al 95 % para el ángulo que hay entre la orientación calculada y la real. Como parámetro adicional y donde llega la atención es en el último parámetro de la tabla 4.1, que muestra el tiempo promedio por ejecución del método.

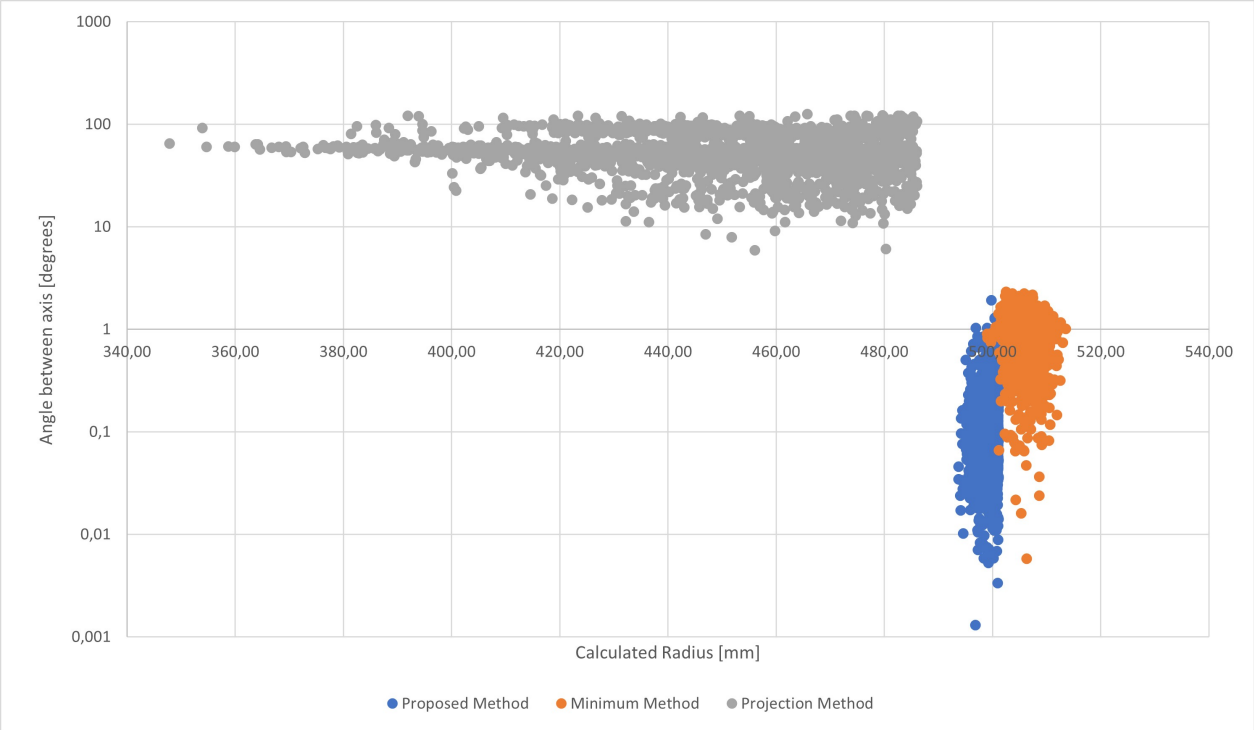


Figura 4.2: Gráfico mostrando ángulo formado entre el vector de orientación calculada y el vector de orientación real, para cada método.

Tabla 4.1: Tabla comparativa entre las estadísticas obtenidas para cilindro con falla tipo bolsillo para los 3 métodos utilizados.

	Método propuesto	Método Mínimo	Método Proyección
Radio [mm] (I. de C. al 95 %)	[499.224 , 499.351]	[505.83 , 506.029]	[447.151 , 449.41]
Error relativo promedio en cálculo del radio [%]	0.24	1.19	10.34
α [grados] (I. de C. al 95 %)	[0.134 , 0.146]	[0.906 , 0.945]	[56.351 , 58.411]
Tiempo promedio por ejecución del método [s]	6.9	302.7	1.5

Hasta este punto, si bien los parámetros obtenidos por el método propuesto presentan menor error que los observados por el método mínimo y el método proyección, es de notar que el tiempo requerido para ejecutar el método propuesto es de 6.9 segundos, muy por debajo de la solución del método mínimo que tarda 302.7 segundos, con lo que el método propuesto está obteniendo un resultado con menor error que ambos métodos, a una fracción del tiempo que le toma al método mínimo, tomando menos de un 3 % del tiempo en completar la misma tarea. Por su parte, si bien el método proyección tarda menos que el método propuesto, los errores al calcular el radio y la orientación hacen que el método resulte inviable, a pesar de su velocidad.

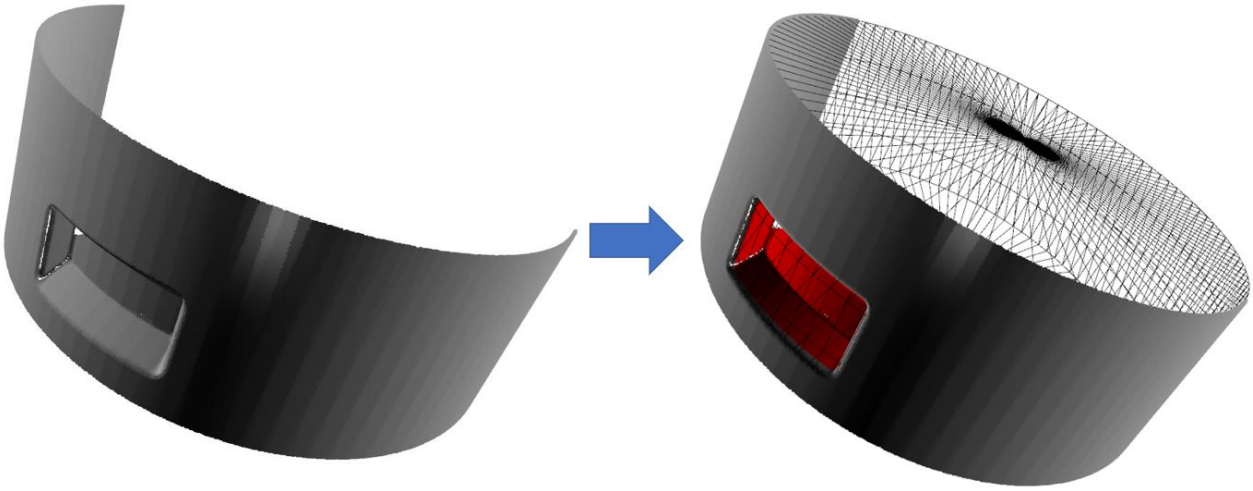


Figura 4.3: A la izquierda, la nube de puntos completa sin distinción de inlier y outliers. A la derecha, los puntos outliers son marcados en rojo y se muestra la malla de la primitiva, como resultado de aplicar el método propio.

4.1.2. Comparación cilindro con falla tipo pitting

- Modelo 3D: Cilindro con falla tipo pitting
- Radio: 20.6 [mm]
- Orientación: [0 , 1 , -1]

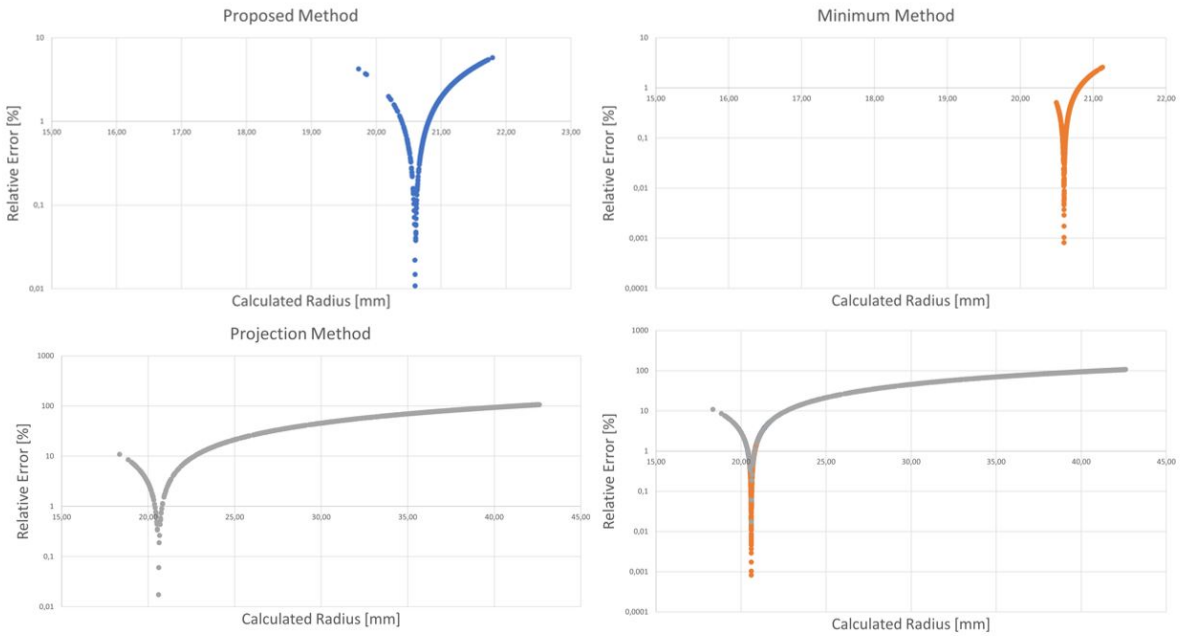


Figura 4.4: Gráficos mostrando el error relativo porcentual para el radio calculado en cada método.

En la figura 4.4 se exponen 3 gráficos en escala semi logarítmica, mostrando el error relativo para cada uno de los métodos, y una gráfica con los 3 métodos al mismo tiempo, también en escala semi logarítmica, esto ya que la dispersión de puntos no es tan marcada en este caso como lo fue para el cilindro con falla tipo bolsillo radial. Estos mismos gráficos se incorporan en el Anexo de manera individual en caso que se desee verlos más detenidamente. De estos gráficos se observa nuevamente que el método propuesto y el método mínimo presentan un bajo error relativo a la hora de determinar el radio del cilindro, siendo mejor el método mínimo por sobre los otros 2 métodos: Mientras el método propuesto tiene errores relativos que van desde el 0.01 % hasta el 5.8 %, el método mínimo tiene errores que van desde el 0.00 % hasta los 2.6 % de error. Por su parte el método proyección también presenta casos donde su error relativo es menor al 0.1 %, sin embargo un porcentaje mayor de los errores relativos son superiores al 10 %

También se presenta en la figura 4.5 el gráfico de dispersión para el ángulo entre el eje calculado y el teórico para cada método.

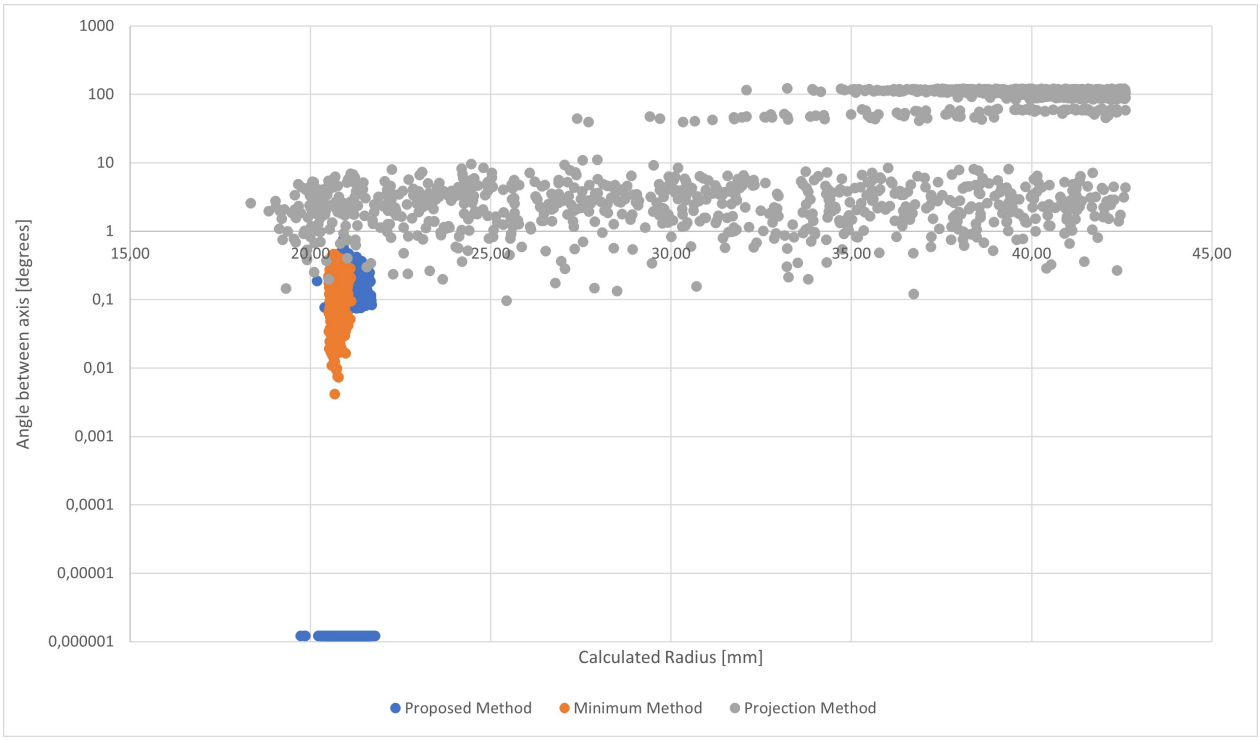


Figura 4.5: Gráfico mostrando ángulo formado entre el vector de orientación calculada y el vector de orientación real, para cada método.

De la figura 4.5 se desprende que el tanto el método propio como el método mínimo estiman la orientación de la figura con bastante precisión, ya que el ángulo que se genera entre la orientación calculada y la orientación real es menor a 1 grado en ambos casos, pero además el método propio tiene varios casos donde determina esta orientación con mejor precisión que el método mínimo, como se ve en el cúmulo de puntos azules en la parte baja del gráfico. Por su parte, el método proyección vuelve a presentar un ángulo entre ejes que sobrepasa en varios casos a 1 grado, teniendo un segmento no despreciable de puntos que genera un ángulo de 100 grados o más.

Toda la información expuesta en la figura 4.4 y 4.5 se resume en la tabla 4.2, donde se muestran intervalos de confianza al 95 % para el radio, el error relativo porcentual al calcular este radio y los intervalos de confianza al 95 % para el ángulo que hay entre la orientación calculada y la real. También se muestra el tiempo promedio de ejecución para cada método al resolver la misma tarea.

Si bien el error relativo del método propuesto al calcular el radio es casi 4 veces el error que tiene el método mínimo para la misma tarea, se debe notar que aquí se trata de una diferencia que llevado a medición es de menos de medio milímetro, por lo que este error relativo de 2.39 % es tolerable, más al tomar en consideración la ventaja que presenta el método propuesto por sobre el método mínimo al determinar la orientación del eje del cilindro y el tiempo que le toma en completar la tarea, ya que el ángulo generado para el método propuesto oscila entre los 0.02 y los 0.026 grados, mientras que en el método mínimo oscila entre los 0.16 y los 0.167. Esto puede ser una diferencia pequeña, ya que se sigue hablando de menos de medio grado de diferencia con la orientación original, pero cuando se trabaja con piezas de gran tamaño esa pequeña diferencia en la inclinación puede implicar que se deposite material varios milímetros

Tabla 4.2: Tabla comparativa entre las estadísticas obtenidas para cilindro con falla tipo bolsillo para los 3 métodos utilizados.

	Método propuesto	Método Mínimo	Método Proyección
Radio [mm] (I. de C. al 95 %)	[21.07 , 21.094]	[20.724 , 20.733]	[36.111 , 36.701]
Error relativo promedio en cálculo del radio [%]	2.39	0.65	76.96
α [grados] (I. de C. al 95 %)	[0.02 , 0.026]	[0.16 , 0.167]	[65.362 , 69.896]
Tiempo promedio por ejecución del método [s]	37.8	360.3	2.1

o hasta centímetros fuera de la zona que interesa reparar. Pero es en el tiempo de ejecución donde se nota el mayor impacto del método propuesto por sobre el método mínimo, ya que el propuesto tarda poco más del 10 % de lo que tarda el método mínimo; para el método proyección, si bien tarda una cantidad notoriamente menor en completar la tarea, no resulta confiable ya que de los datos recogidos presenta un alto error relativo al calcular el radio del cilindro, así como también tiene orientaciones que difieren demasiado de la orientación deseada.

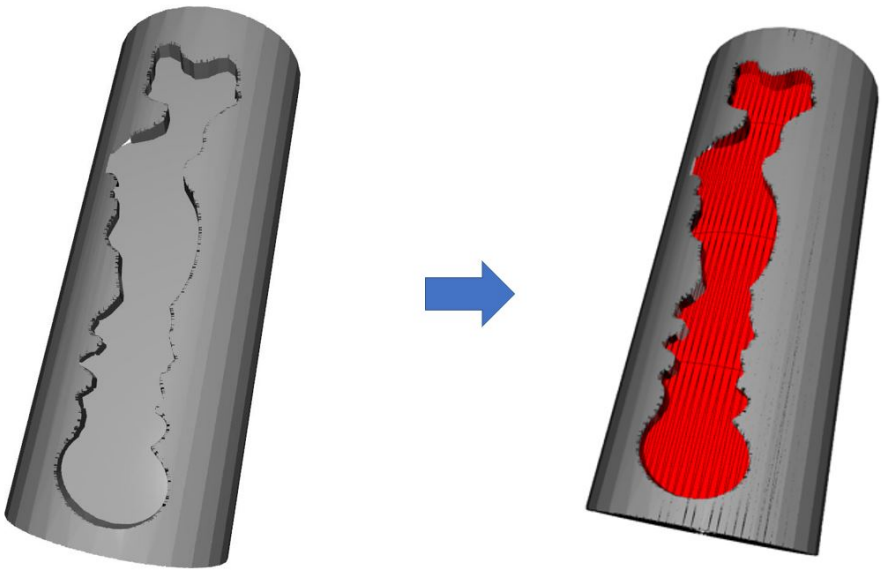


Figura 4.6: A la izquierda, la nube de puntos completa sin distinción de inlier y outliers. A la derecha, los puntos outliers son marcados en rojo y se muestra la malla de la primitiva, como resultado de aplicar el método propio.

4.1.3. Comparación cilindro escaneado con falla tipo bolsillo

- Modelo 3D: Cilindro escaneado con falla tipo bolsillo
- Radio: 63.5 [mm]
- Orientación: [0.09 , 1 , -0.09]

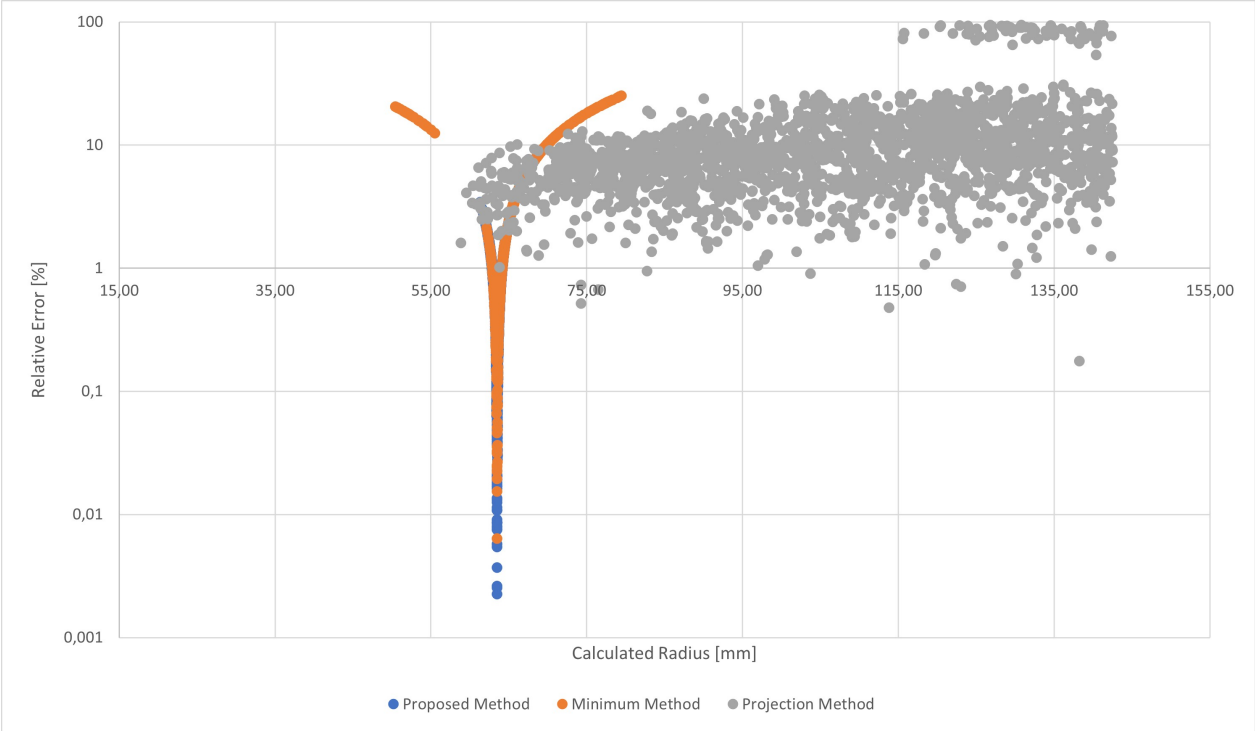


Figura 4.7: Gráfico mostrando error relativo porcentual para el radio calculado en cada método.

En la figura 4.7 se muestra el error relativo para el radio calculado para cada método con la última configuración de cilindro. Aquí el método proyección salvo casos puntuales, tiene errores porcentuales que sobrepasan el 1 %, estando mayoritariamente agrupados en torno al 10 %, mientras que el método propuesto y el método mínimo tienen errores que llegan a valores por debajo del 1 %, en especial el método propuesto que muestra casos con errores menores al 0.01 %. De este gráfico se expone nuevamente que el método proyección presenta resultados poco o nada precisos, lo cual se atribuye a que el método al escoger un conjunto de puntos aleatorios en cada iteración que no dependen de los que se escogieron en la iteración anterior, no puede propagar su respuesta, por lo que trata de ajustar círculos con 3 puntos en cada ocasión, lo cual tiene bajas probabilidades de resultar ya que la combinación específica de puntos que necesita es una tal que los 3 puntos pertenezcan a un mismo plano con normal paralela al eje del cilindro, esto mientras más grande sea la cantidad de puntos presente en la nube de puntos más baja se vuelve la probabilidad de encontrar esta combinación específica de puntos.

Se presenta en la figura 4.8 el gráfico de dispersión para el ángulo entre el eje calculado y el teórico para cada método.

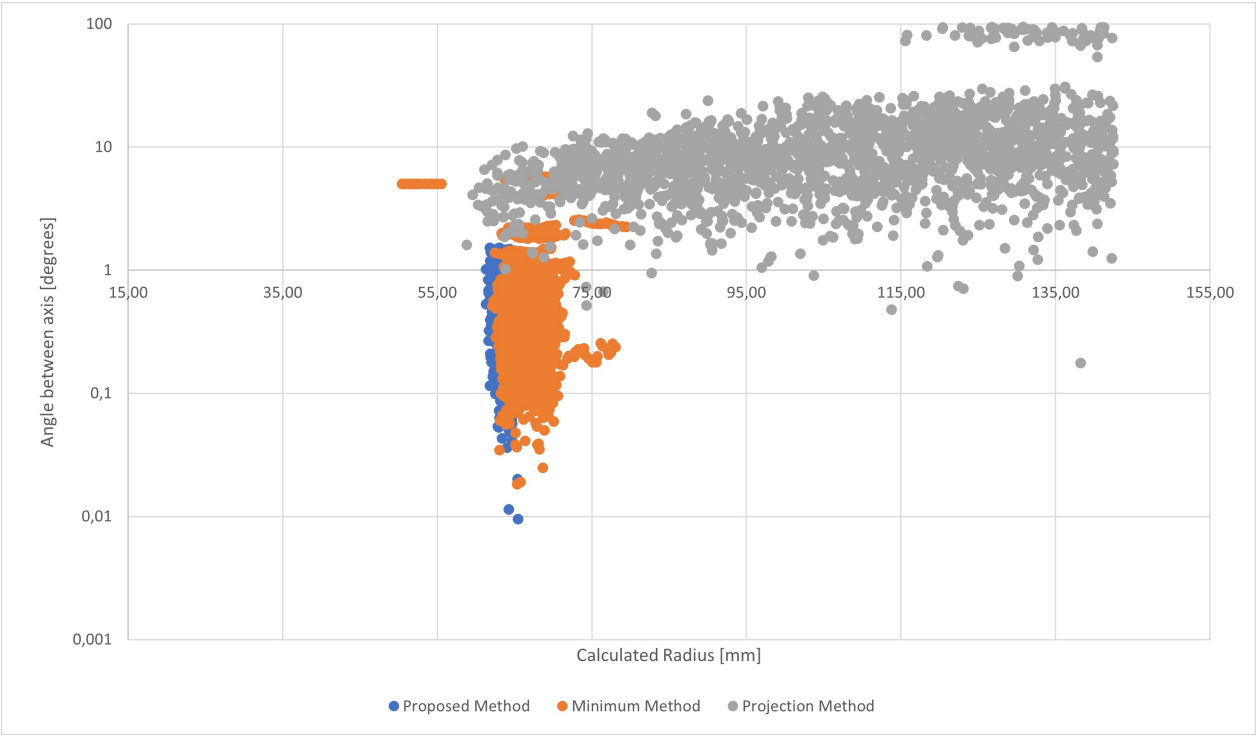


Figura 4.8: Gráfico mostrando ángulo formado entre el vector de orientación calculada y el vector de orientación real, para cada método.

Donde se observa que el método proyección también presenta imprecisiones al momento de determinar la orientación del eje del cilindro, esto como se analizó anteriormente es debido al funcionamiento del algoritmo que utiliza este método. Por su parte, el método mínimo y el método propuesto determinan la orientación con un ángulo que está mayoritariamente comprendido entre 0.1 y 1 grados, con algunos casos donde el ángulo entre el eje calculado y el real es menor a 0.1 grados.

En la tabla 4.3 se resume lo presentado en las figuras 4.7 y 4.8, mostrando los intervalos de confianza calculados para estos parámetros además del tiempo de ejecución para la tarea.

Tabla 4.3: Tabla comparativa entre las estadísticas obtenidas para cilindro con falla tipo bolsillo para los 3 métodos utilizados.

	Método propuesto	Método Mínimo	Método Proyección
Radio [mm] (I. de C. al 95 %)	[63.742 , 63.914]	[66.518 , 66.763]	[104.75 , 106.674]
Error relativo promedio en cálculo del radio [%]	1.08	5.35	66.55
α [grados] (I. de C. al 95 %)	[0.607 , 0.633]	[0.622 , 0.699]	[10.962 , 12.228]
Tiempo promedio por ejecución del método [s]	15.4	594.4	2.6

De la tabla se recoge que el método propuesto y el método mínimo determinan el radio del cilindro con un error relativo bajo, siendo del 1.08 % en el propuesto y 5.349 en el método mínimo, y que al determinar la orientación tienen valores para el ángulo bastante parecidos, pero la diferencia significativa es al comparar el tiempo de ejecución, ya que el tiempo que tarda el método propuesto es poco menos del 2.6 % del tiempo que tarda el método mínimo en completar la tarea. Este nivel de diferencias permite decir que dado que los niveles de error presentes para el método propuesto son similares o mejores a los del método mínimo, y mejores en precisión que el método proyección, la diferencia en los tiempos de ejecución conllevan que uno prefiera el uso del método propuesto al momento de hacer esta tarea, ya que menores tiempos de ejecución implican menores tiempos en la reparación de la pieza mecánica de la que se hizo el modelo 3D, y con ello es menor el tiempo en que se tiene la producción de la máquina detenida, por ende trae beneficios optar por la alternativa más rápida que sería el método propuesto.

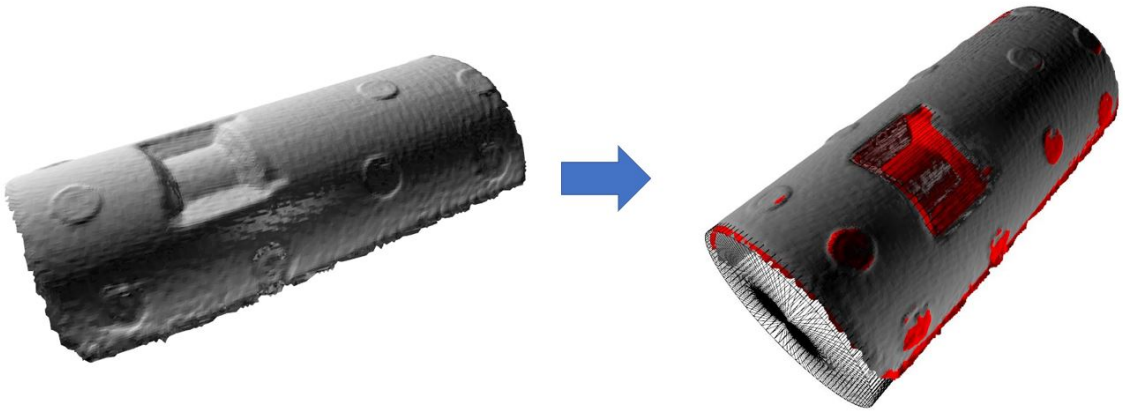


Figura 4.9: A la izquierda, la nube de puntos completa sin distinción de inlier y outliers. A la derecha, los puntos outliers son marcados en rojo y se muestra la malla de la primitiva, como resultado de aplicar el método propio.

4.1.4. Comparación del error relativo porcentual para cada método

La figura a continuación engloba la información expuesta durante los puntos anteriores, con el fin de evidenciar el nivel de error relativo que tuvieron los métodos para cada cilindro.

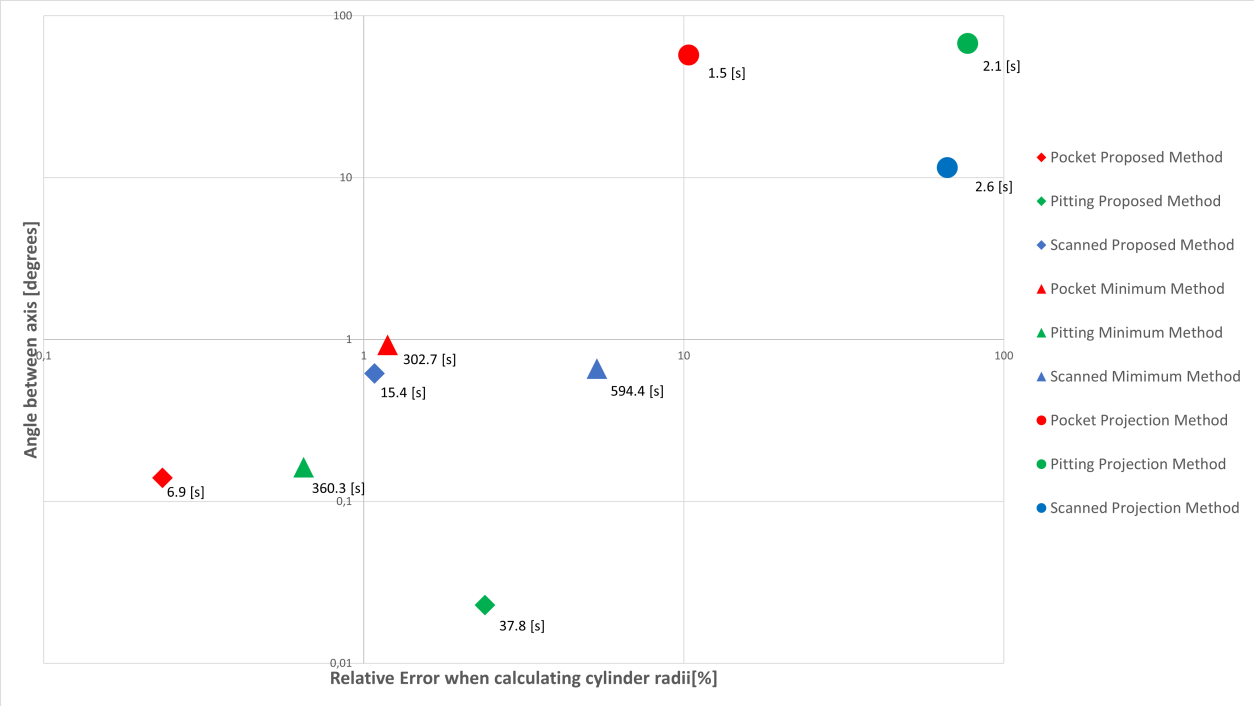


Figura 4.10: Comparación de los ángulos entre ejes y error relativo al calcular el radio del cilindro. En el gráfico, los rombos son resultados obtenidos con método propuesto, los triángulos son resultados obtenidos con método mínimo y los círculos son resultados obtenidos con método de proyección.

En la imagen se muestran 9 puntos, uno por cada combinación de cilindro y método. Para facilitar la distinción de qué punto pertenece a qué método, estos se marcan con distintas figuras, siendo rombos para los resultados del método propuesto, triángulos para los del método mínimo y círculos para los del método proyección; también se hace una distinción por color: en rojo están los resultados para cilindro con falla tipo bolsillo, en verde los resultados para cilindro con falla tipo pitting y en azul los resultados para cilindro escaneado con falla tipo bolsillo. Lo último que se puede rescatar de esta figura (en escala logarítmica) es que en general (2 de 3 casos) el método propuesto presenta un menor error relativo porcentual que el observado en el método mínimo, salvo en un caso, pero como se observó de los resultados reflejados en las tablas, a pesar de esta diferencia en el error relativo, el método propuesto en este trabajo es más eficiente que el método mínimo, esto ya que el algoritmo en ambos se diferencia en su enfoque: mientras el método propuesto tiene una respuesta geométrica a partir de planos y por ende más direccionada, el método mínimo cae en el problema de tener que encontrar un mínimo a una función, la cual a priori no se conocen parámetros y en la medida que la cantidad de puntos presentes en el modelo 3D sean mayores, también serán los tiempos requeridos para completar una minimización de la función de mínimos cuadrados para la distancia. Se recomienda preferir el método propuesto por sobre el método mínimo dada la velocidad con la que resuelve la tarea, además de la precisión con la que se determina la orientación del eje central del cilindro, que de las estadísticas planteadas, se desprende que la orientación se determinaba con menos de 1 grado de diferencia entre lo calculado y lo real.

4.2. Comparación de resultados en conos

De la misma forma en que se presentaron los resultados obtenidos para cilindros, se presentan los resultados para conos. Para el caso de conos dado que son otros métodos, se seguirá la siguiente notación: Hablando de Método Propuesto, Método Mínimo que será el de minimización de la función de mínimos cuadrados para la distancia de un punto a un cono, y el Método Planos corresponde al método donde se intersectan 3 planos y la orientación del eje del cono es paralela a un plano generado a partir de estos 3 puntos. Dada la similitud entre el método propuesto para cono y el método 2 es que se da especial énfasis en la comparación entre estos métodos.

4.2.1. Comparación cono con falla tipo pitting

- Modelo 3D: Cono con falla tipo pitting
- Ángulo de Apertura: 25 grados
- Orientación: [0.585 , -0.171 , 1]

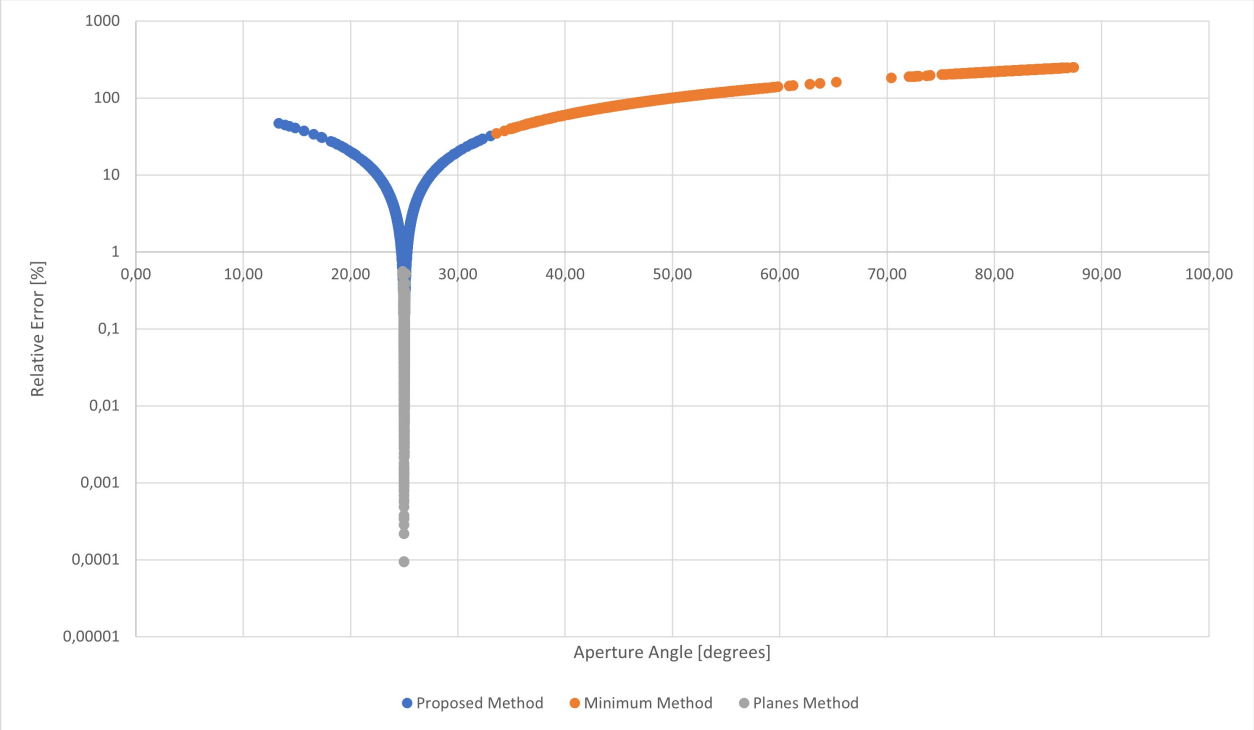


Figura 4.11: Gráfico mostrando error relativo porcentual para el ángulo de apertura en cada método.

En la figura 4.11 se presentan los errores relativos porcentuales para los ángulos de apertura en cada uno de los métodos, donde se ve que el método propuesto tiene casos donde el error va por sobre el 10 %, mientras que el método planos concentra su error en valores menores al 1 %. El error porcentual para el método mínimo alcanza valores que superan el 10 % y hasta del 100 %, esto se debe a que dado el algoritmo de minimización donde se escoge un conjunto

de 3 puntos, existe más de un cono que satisface la condición de pasar por esos 3 puntos, por lo que puede caer en un mínimo local que si bien satisface el modelo que determina a un cono, no necesariamente satisface el modelo para el cono específico que se está buscando, por lo que el método presenta una falla en ese sentido.

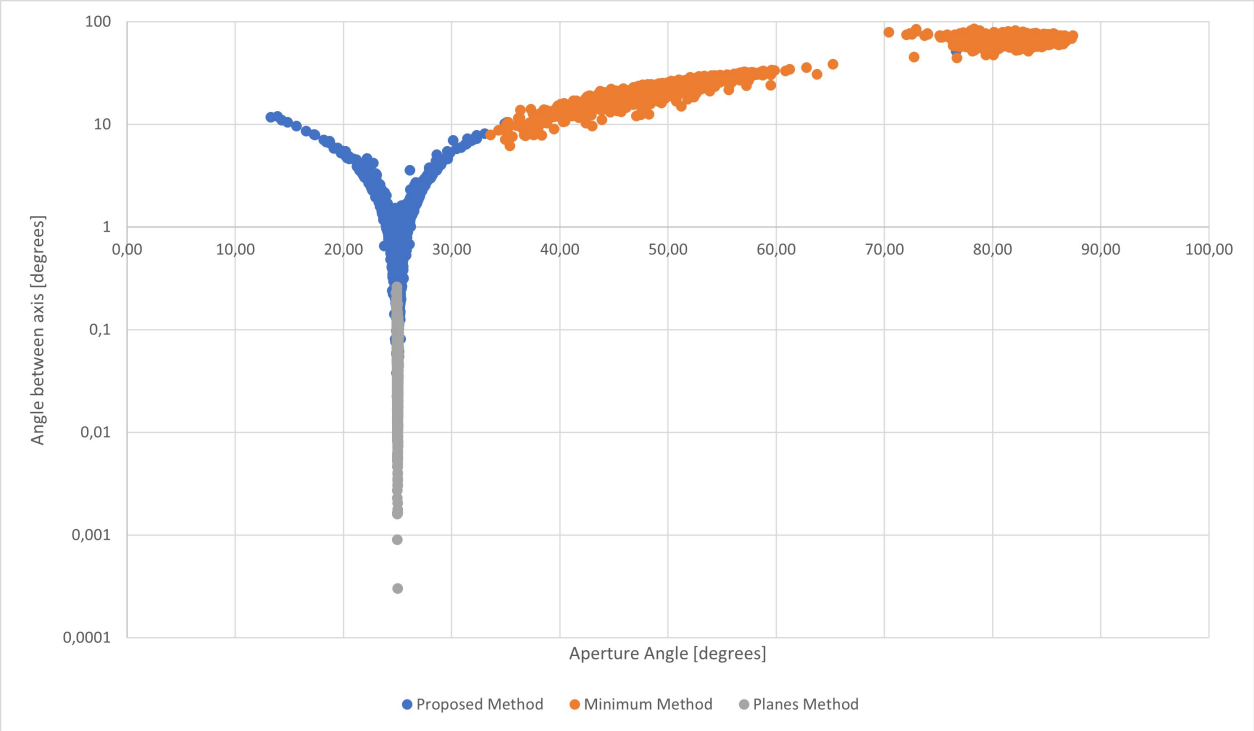


Figura 4.12: Gráfico mostrando ángulo formado entre el vector de orientación calculada y el vector de orientación real, para cada método.

De igual forma que para cilindros, también se determinó la orientación de los conos con respecto a su orientación real. Aquí el gráfico de la figura 4.12 es bastante similar al de la 4.11, teniendo que el método propuesto genera ángulos que van desde los 0.1 a los 10 grados, mientras el método planos determina la orientación de manera tal que su ángulo es de menos de 1 grado, llegando hasta a valores de menos de 0.001 grados. En concordancia con lo dicho anteriormente, el método mínimo también tiene dificultades al determinar la orientación del modelo, por la misma razón del mínimo local, ya que el conjunto de 3 puntos satisface el modelo para el cono que se está buscando, pero también satisface para el modelo de un cono con otra apertura y una orientación totalmente distintos a los buscados.

La tabla 4.4 muestra un resumen de los datos asociados al cono con falla tipo pitting, donde se ve que el tiempo de ejecución del método propuesto es de 2.2 segundos contra los 3.4 segundos del método planos. Esta diferencia en el tiempo se produce ya que el método propuesto tiene menos combinaciones posibles para poder trabajar, dado que la dirección del eje se determina a partir del vértice del cono y la intersección de las normales en un punto del eje del cono, pero esta intersección sólo existe en determinados casos, por lo que en los casos donde esta intersección no existe, el algoritmo pasa inmediatamente a la siguiente iteración, evitando el uso de recursos en la etapa donde se comparan todos los puntos del modelo 3D al modelo calculado por RANSAC. Esto no ocurre en el método planos, ya que dado un conjunto de 3 puntos siempre se va a poder crear un plano a partir de ellos, por lo

Tabla 4.4: Tabla comparativa entre las estadísticas obtenidas para cono con falla tipo pitting para los 3 métodos utilizados.

	Método propuesto	Método Mínimo	Método Planos
Ángulo Apertura [grados] (I. de C. al 95 %)	[24.904 , 25.069]	[64.93 , 66.459]	[24.993 , 24.996]
Error relativo promedio en cálculo de apertura [%]	3.61	162.78	0.08
α [grados] (I. de C. al 95 %)	[1.051 , 1.2]	[43.636 , 45.675]	[0.052 , 0.055]
Tiempo promedio por ejecución del método [s]	2.2	13.1	3.4

que no hay combinaciones de puntos que resulten inviables, por lo tanto se ejecutan todas las iteraciones, repercutiendo en un tiempo adicional de más del 50 % con respecto al método propuesto. El método mínimo toma un tiempo mayor por el carácter de minimizar la función y con ello tener que pasar por todos los puntos del modelo 3D en su resolución antes de dar con los parámetros, para luego comparar todos los puntos al modelo calculado.

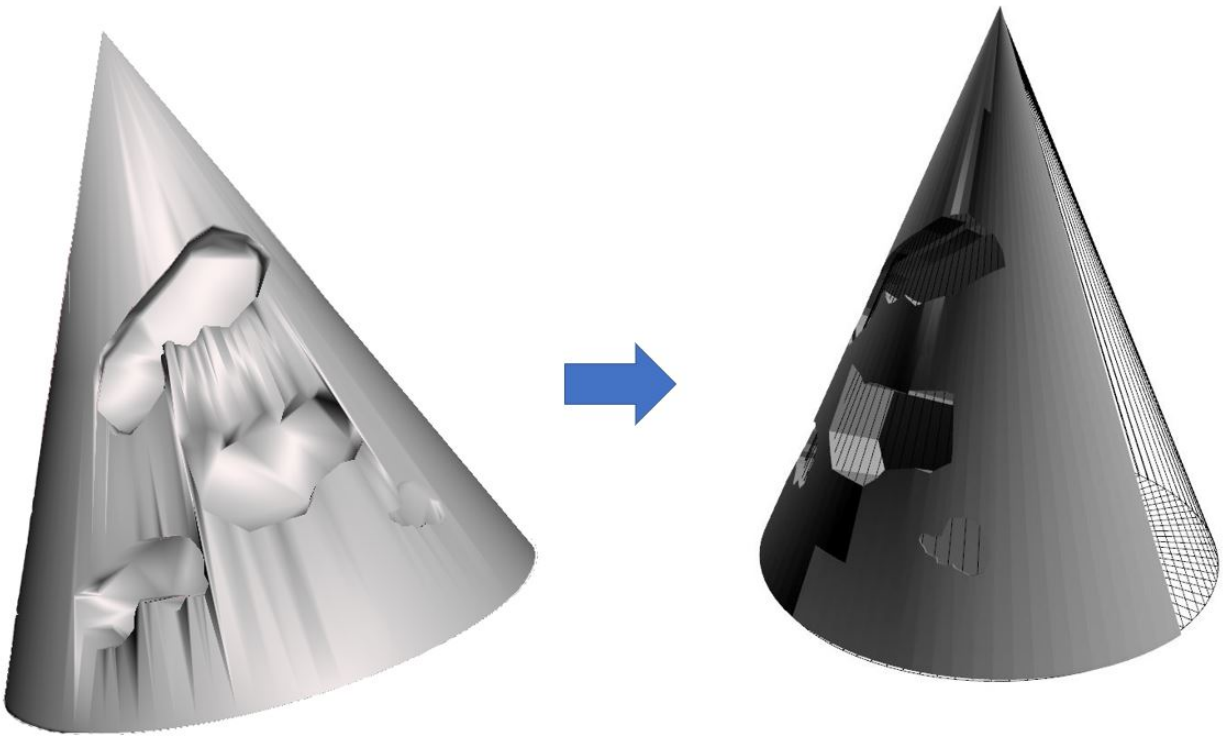


Figura 4.13: A la izquierda, la nube de puntos completa sin distinción de inlier y outliers. A la derecha, se muestra la malla de la primitiva, como resultado de aplicar el método propio.

4.2.2. Comparación cono con falla tipo bolsillo

- Modelo 3D: Cono con falla tipo bolsillo
- Ángulo de Apertura: 25 grados
- Orientación: [0.585 , -0.171 , 1]

De la figura 4.13 y 4.14 se ve un comportamiento similar a lo visto en las figuras 4.11 y 4.12, respectivamente. Esto se debe a que el modelo 3D es muy similar en ambos casos, teniendo como diferencia solamente el tipo de falla, pero el mismo ángulo de apertura y orientación en ambos casos. Aquí en las figuras 4.13 y 4.14 se aprecian casos donde el método propuesto alcanza errores relativos de menos del 0.01 %, y ángulos para la orientación menos a 0.01 grados; no así el método mínimo, cuyo comportamiento al determinar la orientación cambia, pero sigue una dispersión parecida a los gráficos anteriores.

Esto se reafirma al ver la información resumida en la tabla 4.5, donde el error relativo promedio para el método propuesto es de 1.85 %, contra el 0.46 % del método planos. A pesar de que el error del método propuesto es mayor, hay que notar que el ángulo de apertura se detecta entre 24.874 y 24.943, o sea es menos de 0.2 grados de diferencia, lo cual dependiendo el tamaño de la pieza a reparar puede ser despreciable. Lo mismo ocurre para la orientación, que a pesar de definir un ángulo casi el triple del que define el método planos, sigue siendo una diferencia menor a 1 grado, con lo que si la pieza a reparar no es significativamente grande, esta diferencia es efectivamente despreciable, y el interés de despreciar estas diferencias apunta a la diferencia de tiempo que hay al ejecutar los métodos, ya que el método propuesto tarda 0.7 segundos, menos de la mitad de lo que tarda el método planos.

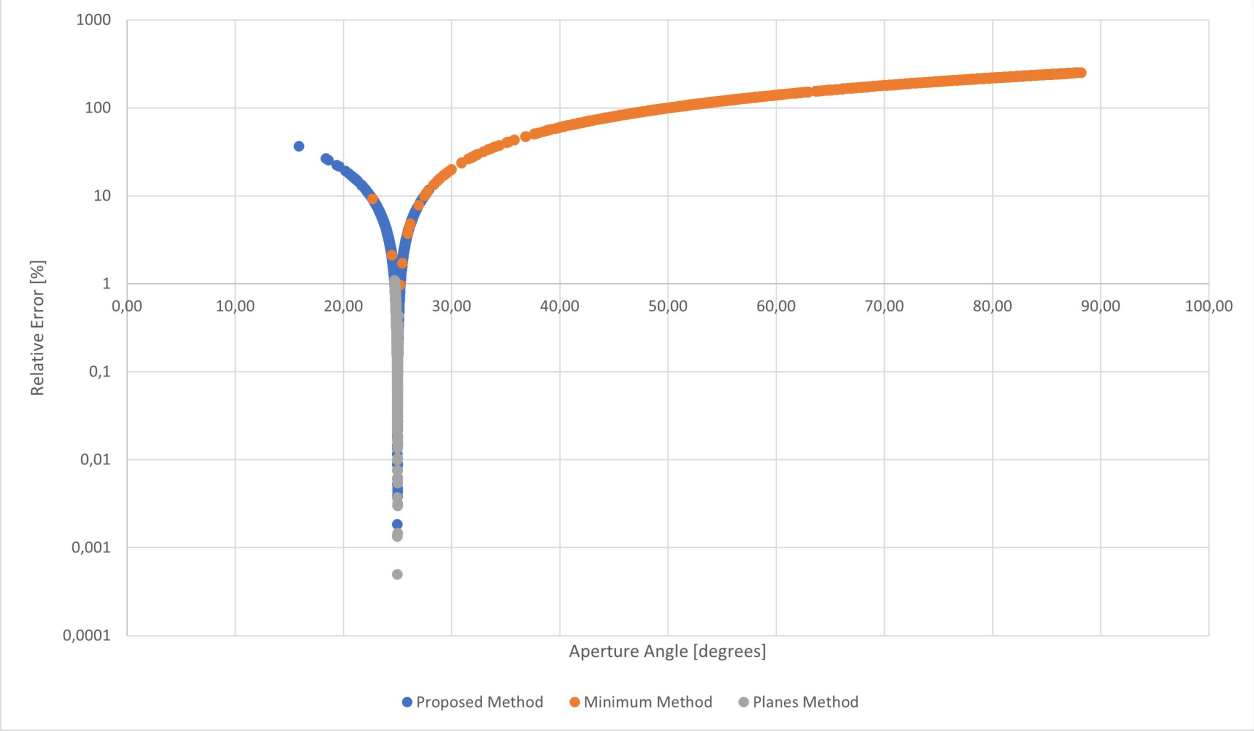


Figura 4.14: Gráfico mostrando error relativo porcentual para el ángulo de apertura en cada método.

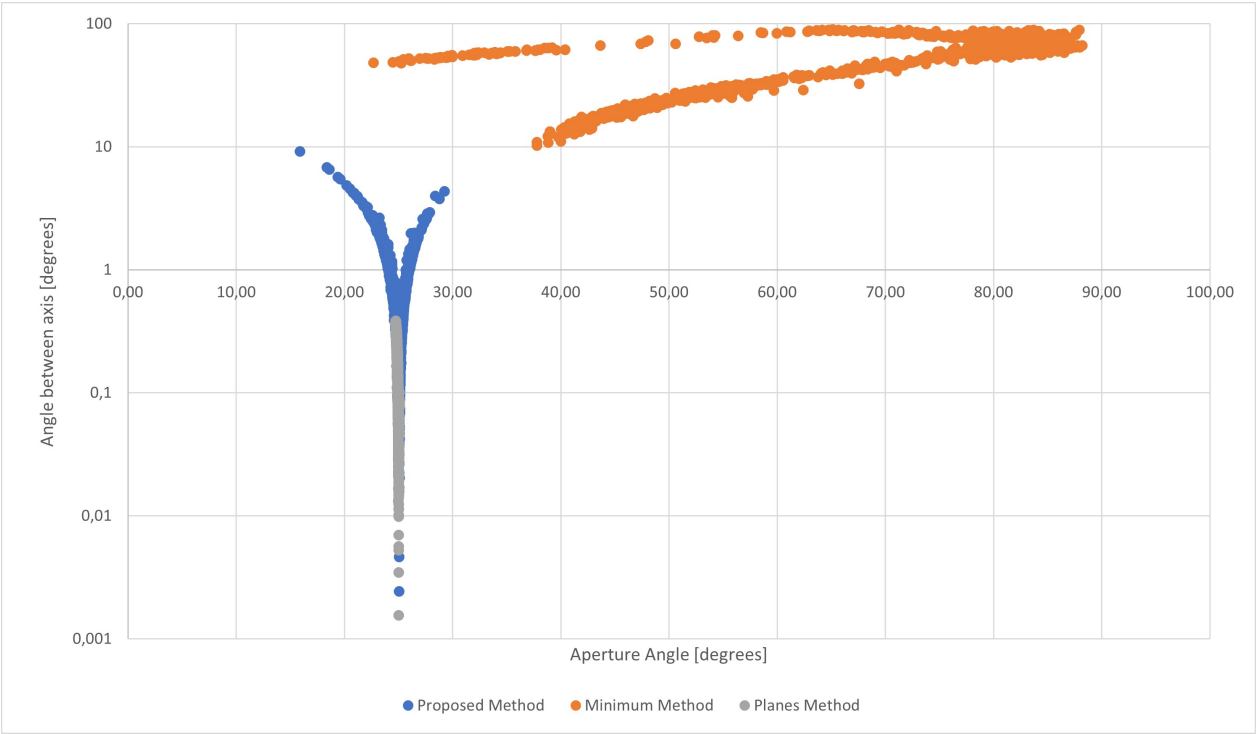


Figura 4.15: Gráfico mostrando ángulo formado entre el vector de orientación calculada y el vector de orientación real, para cada método.

Tabla 4.5: Tabla comparativa entre las estadísticas obtenidas para cono con falla tipo bolsillo para los 3 métodos utilizados.

	Método propuesto	Método Mínimo	Método Planos
Ángulo Apertura [grados] (I. de C. al 95 %)	[24.874 , 24.943]	[67.899 , 69.332]	[24.883 , 24.889]
Error relativo promedio en cálculo de apertura [%]	1.85	174.47	0.46
α [grados] (I. de C. al 95 %)	[0.536 , 0.595]	[53.066 , 55.03]	[0.187 , 0.195]
Tiempo promedio por ejecución del método [s]	0.7	17.4	1.6

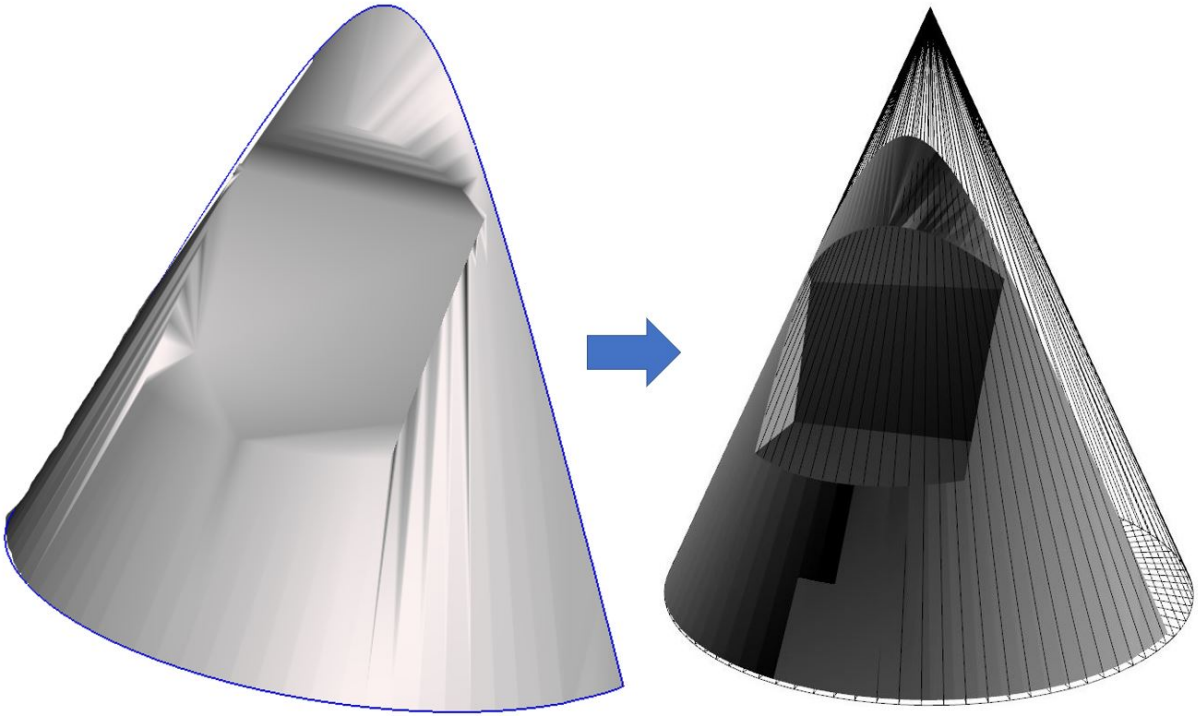


Figura 4.16: A la izquierda, la nube de puntos completa sin distinción de inlier y outliers. A la derecha, se muestra la malla de la primitiva, como resultado de aplicar el método propio.

4.2.3. Comparación cono escaneado con falla por despunte

- Modelo 3D: Cono escaneado con falla por despunte
- Ángulo de Apertura: 38 grados
- Orientación: [0.318 , 0.091 1]

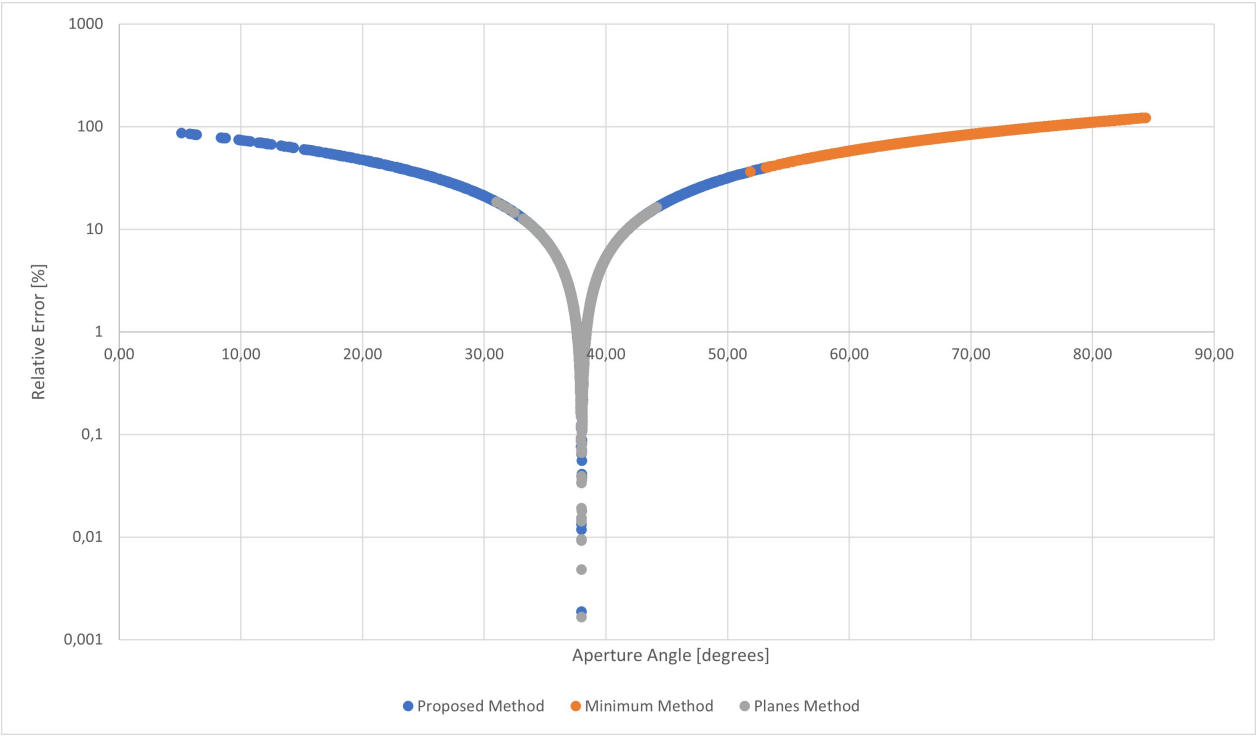


Figura 4.17: Gráfico mostrando error relativo porcentual para el ángulo de apertura en cada método.

Del gráfico en la figura 4.15 se ve que el método propuesto tiene algunos casos con un error relativo cercano al 100 %, pero al revisar los datos resumidos en la tabla 4.6 se observa que son casos aislados, ya que con una confiabilidad del 95 % las soluciones entregadas por el método están entre los 37.244 y 37.891 grados, esto considerando que el ángulo de apertura buscado es de 38 grados resulta en que el error en realidad es bastante bajo y el error relativo promedio se ve tan alto producto los casos aislados que llegan cerca del 100 %. No así como pasa para el método mínimo donde la mayor acumulación se ve en un tramo desde 50 % de error a más de 100 %, dando un promedio de 92 %, mientras que el método planos tiene su mayor acumulación en errores menores al 5 %.

Sin embargo, la dispersión observada para el método propuesto con respecto al ángulo de los ejes difiere de lo observado en los gráficos de las figuras anteriores, donde se veía un comportamiento similar en todo momento para el método. La razón de esto es que al determinar el eje del modelo a partir de la intersección de las normales de los puntos, no solo depende de que estos puntos estén en un mismo plano paralelo a la base del cono, si no que sus normales estén debidamente orientadas, por lo que una ligera desviación en como están definidas estas normales puede impactar en que la dirección del eje no sea estimada correctamente; este tipo de desviaciones en las normales se justifica ya que al hacer un escaneo de una pieza, el nivel de definición de la malla de triángulos determina cuántos triángulos efectivamente hay, que es el paso intermedio a que se pase a una nube de puntos. Si se tiene una baja cantidad de triángulos, el modelo 3D pierde definición y con ello las normales de estos puntos no necesariamente quedan orientadas de manera correcta. Esto no es algo que impacte para el método planos ya que utiliza las normales para poder determinar la posición del vértice, sin embargo la orientación del modelo la determina haciendo un plano con los 3

puntos escogidos, donde difiere del método propuesto. Este tipo de problema con la definición del modelo 3D se puede abordar mediante un remallado utilizando algún software externo, con lo que se redefine la cantidad de triángulos de la malla y con ello aumentando el nivel de definición del modelo.

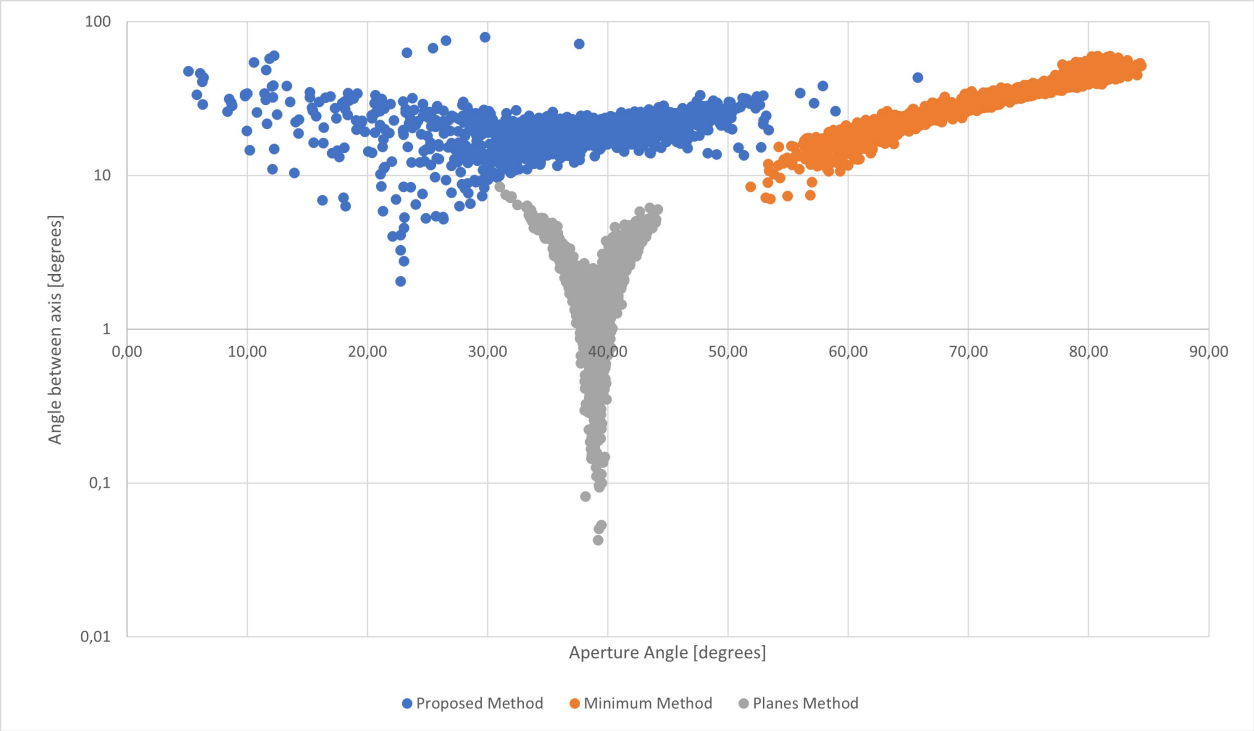


Figura 4.18: Gráfico mostrando ángulo formado entre el vector de orientación calculada y el vector de orientación real, para cada método.

Tabla 4.6: Tabla comparativa entre las estadísticas obtenidas para cono escaneado con falla por despunte para los 3 métodos utilizados.

	Método propuesto	Método Mínimo	Método Planos
Ángulo Apertura [grados] (I. de C. al 95 %)	[37.244 , 37.891]	[72.806 , 73.515]	[39.269 , 39.418]
Error relativo promedio en cálculo de apertura [%]	13.64	92.52	4.77
α [grados] (I. de C. al 95 %)	[19.662 , 20.134]	[34.902 , 36.01]	[1.89 , 1.99]
Tiempo promedio por ejecución del método [s]	3.4	20.9	4.6

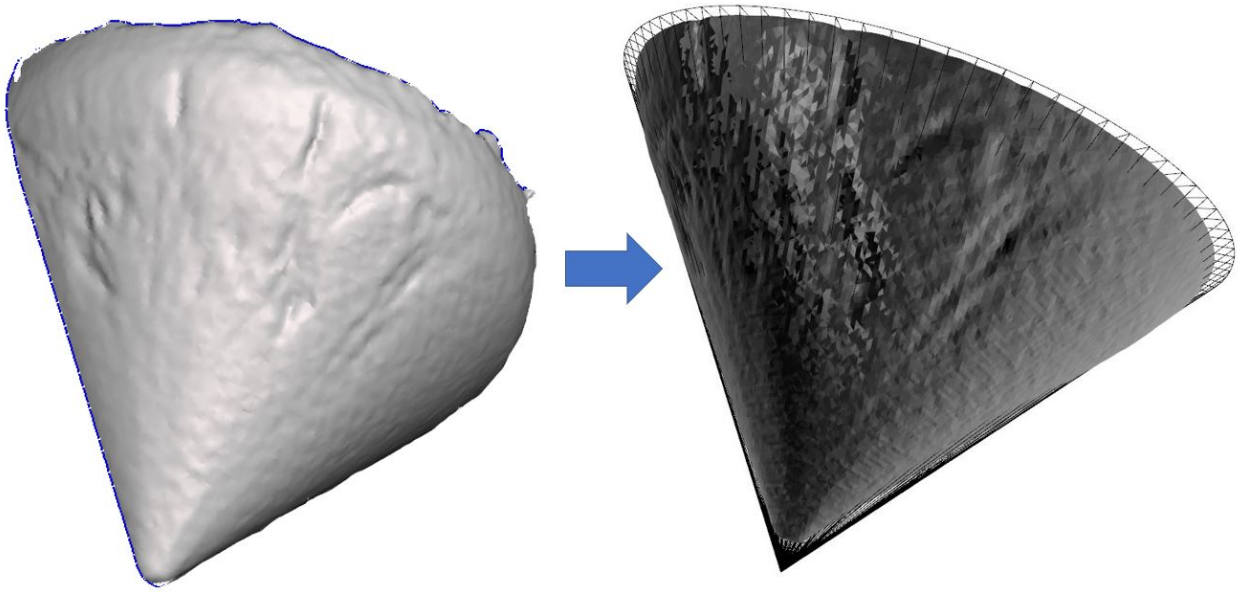


Figura 4.19: A la izquierda, la nube de puntos completa sin distinción de inlier y outliers. A la derecha, se muestra la malla de la primitiva, como resultado de aplicar el método propio.

4.2.4. Comparación del ángulo entre ejes para cada método

En la figura se ve un cuadro comparativo entre los resultados de todos los métodos, marcados con un rombo están los resultados del método propuesto, con un triángulo los del método mínimo y con círculo los resultados del método planos. El color denota qué nube de puntos fue trabajada en cada caso: con rojo se marcaron los resultados obtenidos para cono con falla tipo pitting, con verde los resultados para cono con falla tipo bolsillo, y con azul los resultados para cono escaneado con falla por despunte. En el eje vertical está el ángulo que forma la orientación calculada con la real, en la horizontal el ángulo de apertura del modelo 3D. Dada la configuración que se estudió, se esperan valores para el eje horizontal en torno a los 25 grados y los 38, mientras que en la vertical se esperan valores lo más cercanos al 0 que sean posibles. Dicho esto y con la excepción del caso de despunte para el método propuesto y los 3 resultados para el método mínimo, los resultados se agrupan en el sector deseado, con un bajo ángulo entre ejes. Esto como se mencionaba, lleva a que el desempeño del método propuesto sacrifica un poco de precisión a cambio de mejorar la velocidad con la que se entrega la respuesta.

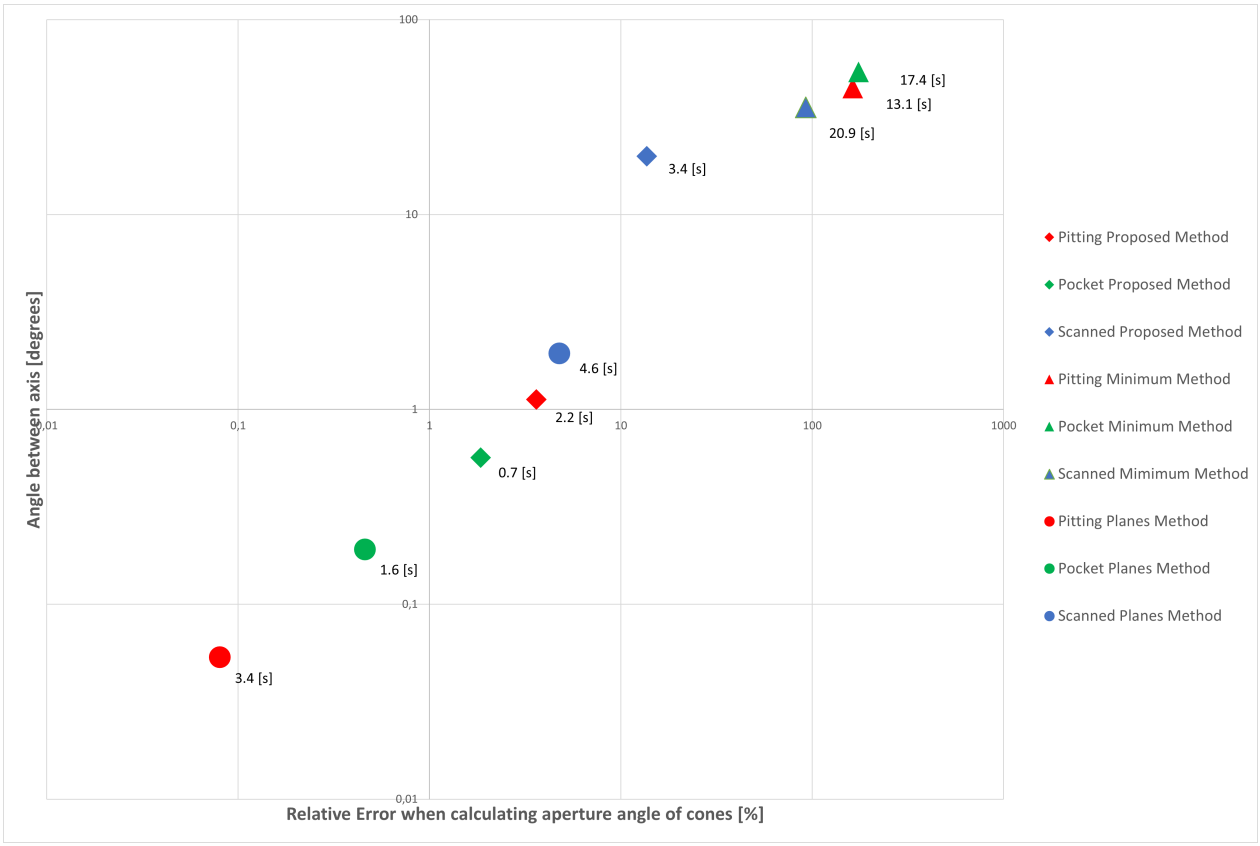


Figura 4.20: Comparación de los ángulos entre ejes y error relativo al calcular el ángulo de apertura. En el gráfico, los rombos son resultados obtenidos con método propuesto, los triángulos son resultados obtenidos con método mínimo y los círculos son resultados obtenidos con método de planos.

Capítulo 5

Conclusiones

Se consigue generar una representación 3D idealizada a partir de una nube de puntos de piezas mecánicas con forma cilíndrica o cónica, además de la programación de métodos vistos en la literatura con los que realizar comparación en los desempeños.

Con la metodología propuesta, es posible la detección de puntos outlier tanto en modelos cilíndricos como cónicos, con lo que se puede generar correctamente la pieza idealizada solo con los inliers, así como también la detección de los parámetros del modelo. Esto último contribuye a ser incorporado en flujos de trabajo donde se haga reparación de componentes mecánicos como es el caso comentado en la motivación de este trabajo, en que se usa como una alternativa para dar orientación automática de las piezas al momento de determinar las rutas de reparación para la pieza. La solución propuesta en este trabajo es mejor en su velocidad de ejecución comparado contra otras soluciones vistas durante este estudio. Esto permite a la hora de aplicarlo en la industria tener menores tiempos asignados a la reparación de componentes, con ello los tiempos de producción se verán interrumpidos en menor medida, en particular para el caso de reparaciones de eje donde la solución propuesta alcanza tiempos menores al 15 % de lo que actualmente se dispone, con lo que la disminución de tiempo a utilizar es significativa, como también para la reparación de agujas donde sus figuras son cónicas, en esos ambientes el tiempo de ejecución es de alrededor del 50 % comparado a la solución computacional actual, y menor tiempo de ejecución que el que toma para un operador hacer la orientación manual del modelo 3D.

Bibliografía

[1] Análisis de fallas en metales por fractura, desgaste y corrosión – Dictuc. (s. f.). Dictuc. https://www.dictuc.cl/servicios/analisis-de-fallas-en-metales-por-fractura-desgaste-y-corrosion/

[2] Felix, D. (2017). Experimental investigation on suspended sediment, hydro-abrasive erosion and efficiency reductions of coated Pelton turbines. Zürich: ETH Zürich. Obtenido de https://www.ethz.ch/content/dam/ethz/special-interest/baug/vaw/vaw-dam/documents/das-institut/mitteilungen/2010-2019/238.pdf

[3] Programa Nacional de Innovación en Manufactura Avanzada. (2019, 23 agosto). Proyecto 3: Sistema Robotizado de Recuperación de Piezas Metálicas mediante Manufactura Aditiva. Programa IMA. https://www.programaima.cl/proyecto3/

[4] Flynn, A. (2020, 27 noviembre). What do you understand by outliers and inliers? Greedhead. https://greedhead.net/what-do-you-understand-by-outliers-and-inliers/

[5] Fischler, M. A., Bolles, R. C. (1981). Random sample consensus. Communications of the ACM, 24(6), 381–395. https://doi.org/10.1145/358669.358692

[6] Linsen, L. (2001). Point cloud representation. Technical Report, Faculty of Computer Science, University of Karlsruhe: Univ., Fak. für Informatik, Bibliothek.

[7] Estimating Surface Normals in a PointCloud — Point Cloud Library 1.12.0-dev documentation. (s. f.). Pointclouds. https://pointclouds.org/documentation/tutorials/normal_estimation.html

[8] El algoritmo K-NN y su importancia en el modelado de datos | Blog. (s. f.). Merkle. https://www.merkleinc.com/es/es/blog/algoritmo-knn-modelado-datos

[9] 3D Least Squares Plane. (2009, 9 septiembre). Stack Overflow. https://stackoverflow.com/questions/1400213/3d-least-squares-plane

[10] Least-Squares Fit. (s. f.). Math.Brown.Edu. http://www.math.brown.edu/tbanchof/gc/linalg/linalg.html

[11] RPubs - Análisis de componentes principales (PCA). (2019, 27 septiembre). RPubs. https://rpubs.com/Cristina_Gil/PCA

[12] Open3D – A Modern Library for 3D Data Processing. (s. f.). Open3D. http://www.open3d.org/

[13] trimesh — trimesh 3.9.36 documentation. (s. f.). Trimesh. https://trimsh.org/trimesh.html

[14] Santoyo, S. (2018, 21 junio). A Brief Overview of Outlier Detection Techniques - To-

wards Data Science. Medium. https://towardsdatascience.com/a-brief-overview-of-outlier-detection-techniques-1e0b2c19e561

[15] M. (2020, 28 septiembre). What are Isolation Forests? How to use them for Anomaly Detection? Machine Learning Interviews. https://machinelearninginterview.com/topics/machine-learning/explain-isolation-forests-for-anomaly-detection/

[16] Tran, T. T., Cao, V. T., Laurendeau, D. (2015). Extraction of cylinders and estimation of their parameters from point clouds. Computers Graphics, 46, 345–357. https://doi.org/10.1016/j.cag.2014.09.027

[17] Barbier, A., Galin, E. (2004). Fast Distance Computation Between a Point and Cylinders, Cones, Line-Swept Spheres and Cone-Spheres. Journal of Graphics Tools, 9(2), 11–19. https://doi.org/10.1080/10867651.2004.10504892

[18] Beder, C., Förstner, W. (2006). Direct Solutions for Computing Cylinders from Minimal Sets of 3D Points. Computer Vision – ECCV 2006, 135–146. https://doi.org/10.1007/11744023_11

[19] Zhang, L., Guo, C. (2018). A method for cone fitting based on certain sampling strategy in CMM metrology. AIP Conference Proceedings. Published. https://doi.org/10.1063/1.5033838

[20] Schnabel, R., Wahl, R., Klein, R. (2007). Efficient RANSAC for Point-Cloud Shape Detection. Computer Graphics Forum, 26(2), 214–226. https://doi.org/10.1111/j.1467-8659.2007.01016.x

Anexos

Anexo A

Código utilizado

A.1. Método propuesto cilindros

Código A.1: Código desarrollado para determinar parámetros de un cilindro

```
"""@author: Erick Kracht G."""
#%% RANSAC-3D
"""Import Libraries and Local Functions"""

import open3d as o3d
import numpy as np
import random
import timeit
import trimesh
from scipy.spatial.transform import Rotation
from skspatial.objects import Plane
from skspatial.objects import Line

def calc_dist_to_line(point, position, orientation):
    """ Calculate the distance of a 3D point to a 3D line that passes through (xc, yc, zc) with
    ↪ direction
    (sx, sy, sz) """
    point = np.asarray(point)
    position = np.asarray(position)
    orientation = np.asarray(orientation)
    Dist = position - point
    D = np.cross(Dist,orientation)
    return (np.linalg.norm(D, axis=1))/(np.linalg.norm(orientation))

def calc_singlepointdist_to_line(point, position, orientation):
    """ Calculate the distance of a 3D point to a 3D line that passes through (xc, yc, zc) with
    ↪ direction
    (sx, sy, sz) """
    point = np.asarray(point)
    position = np.asarray(position)
    orientation = np.asarray(orientation)
    Dist = position - point
```

```
    D = np.cross(Dist,orientation)
    return (np.linalg.norm(D))/(np.linalg.norm(orientation))

def in_sphere(radio,center,points):
    radios = dist_pts3d(points,center)
    posiciones = radios < radio
    contenidos = points[np.where(posiciones == 1)[0]]
    return contenidos

def dist_pts3d(x,y):
    """ Calculate distance between two 3D points """
    x = np.asarray(x)
    x = x.T
    return np.sqrt((x[0]-y[0])**2 + (x[1]-y[1])**2 + (x[2]-y[2])**2)

def dist_cyl(cyl_args,points):
    """ Calculate the total distance of multiple points to a cylinder """
    radius, positionX, positionY, positionZ, orientationX, orientationY, orientationZ =
     ↪ cyl_args
    position = positionX, positionY, positionZ
    orientation = orientationX, orientationY, orientationZ
    comp_R = calc_dist_to_line(vertices,position, orientation)
    loss = sum(abs(comp_R - radius))**2
    return loss

#%% Data Input and Pre-Processing
start_time = timeit.default_timer()
pcd = []
mesh = []

"""Archivos elegidos para estadística de la Tesis"""
# pcd = o3d.io.read_point_cloud("STLs/Mantos/cilindro_1m_abrasion
    ↪ v1_halfmantle_X45Y45.pcd") #Bolsillo rectangular, radio más que largo
# pcd = o3d.io.read_point_cloud("STLs/Mantos/cilindro-pitting2_10x.pcd") #Pitting
mesh = o3d.io.read_triangle_mesh("STLs/escaneados/cilindro_bolsillo_radial.stl") #Bolsillo
    ↪  rectangular más largo que radio, escaneado
pcd = mesh.sample_points_uniformly(number_of_points=500000)

if pcd:
    """Centrado de PCD en origen"""
    pcd = pcd.translate((0,0,0), relative=False)

    """Carga de puntos de la pcd como un array."""
    pcd_array = np.asarray(pcd.points)
    pcd_normals = np.asarray(pcd.normals)
    axis_1 = max(pcd_array[:,0])-min(pcd_array[:,0])
    axis_2 = max(pcd_array[:,1])-min(pcd_array[:,1])
    axis_3 = max(pcd_array[:,2])-min(pcd_array[:,2])
    max_axis = axis_1, axis_2, axis_3
    max_dim = max(max_axis)

    """Selección de los puntos con sus respectivas normales"""
```

```
80      Points_With_Normals = np.array(pcd.points)[:,0],np.array(pcd.points)[:,1],np.array(pcd.
        ↪ points)[:,2],np.array(pcd.normals)[:,0],np.array(pcd.normals)[:,1],np.array(pcd.normals
        ↪ )[:,2]
81      Points_With_Normals = np.array(Points_With_Normals).T
82      List_PointsWithNormals_FullSize = list(Points_With_Normals)
83      if len(List_PointsWithNormals_FullSize) > 1000000:
84          List_PointsWithNormals = random.sample(List_PointsWithNormals_FullSize
        ↪ ,1000000)
85          PointsWithNormals = np.asarray(List_PointsWithNormals)
86          vertices = PointsWithNormals[:,0], PointsWithNormals[:,1], PointsWithNormals[:,2]
87          vertices = np.asarray(vertices)
88          vertices = vertices.T
89      else:
90          List_PointsWithNormals = List_PointsWithNormals_FullSize
91          vertices = pcd_array
92
93  elif mesh:
94      """Centrado de Mesh en origen"""
95      if type(mesh) == trimesh.base.Trimesh:
96          mesh = mesh.apply_translation((0,0,0))
97          vertices_full = mesh.vertices
98          normales = mesh.vertex_normals
99      elif type(mesh) == o3d.cpu.pybind.geometry.TriangleMesh:
100         mesh = mesh.translate((0,0,0), relative=False)
101
102         """Carga de vertices del mesh como un array."""
103         vertices_full = np.asarray(mesh.vertices)
104         mesh.compute_vertex_normals()
105         normals = np.asarray(mesh.triangle_normals)
106         normales = list(range(len(vertices_full)))
107         for i in range(len(normales)):
108             normales[i] = normals[int(np.floor(i/3))]
109         normales = np.asarray(normales)
110
111     axis_1 = max(vertices_full[:,0])-min(vertices_full[:,0])
112     axis_2 = max(vertices_full[:,1])-min(vertices_full[:,1])
113     axis_3 = max(vertices_full[:,2])-min(vertices_full[:,2])
114     max_axis = axis_1, axis_2, axis_3
115     min_dim = min(max_axis)
116     max_dim = max(max_axis)
117
118     """Selección de los puntos con sus respectivas normales"""
119     Points_With_Normals = vertices_full[:,0],vertices_full[:,1],vertices_full[:,2],normales[:,0],
        ↪ normales[:,1],normales[:,2]
120
121     Points_With_Normals = np.array(Points_With_Normals).T
122     List_PointsWithNormals_FullSize = list(Points_With_Normals)
123     if len(List_PointsWithNormals_FullSize) > 1000000:
124         List_PointsWithNormals = random.sample(List_PointsWithNormals_FullSize,500000)
125         PointsWithNormals = np.asarray(List_PointsWithNormals)
126         vertices = PointsWithNormals[:,0], PointsWithNormals[:,1], PointsWithNormals[:,2]
127         vertices = np.asarray(vertices)
```

```
128         vertices = vertices.T
129     else:
130         List_PointsWithNormals = List_PointsWithNormals_FullSize
131         vertices = vertices_full
132     list_vertices = list(vertices)
133 
134 #%% Section for iteration of RANSAC
135 
136 Big_Iterations = 1
137 for j in range(Big_Iterations):
138     start_time_internal = timeit.default_timer()
139     NumIter = 750
140     Thresh = 0.02 # Percentage, use with radius
141     Score = 0
142     Best_Sample = [0, 0, 0, 0, 0, 0, 0, 0]
143     Best_Sample_Position = 0
144     Data_Iterations = list(range(NumIter))
145     Best_Comparison = 0
146     Loss = 10**20
147 
148     Max_Possible_Score = len(vertices)
149 
150     for iteration in range(NumIter):
151         if Loss < 1:
152             break
153         """Se eligen 3 puntos al azar, se extraen sus posiciones XYZ y sus normales"""
154         points = np.asarray(random.sample(List_PointsWithNormals,3))
155 
156         Point1 = np.array([points[0][0],points[0][1],points[0][2]])
157         Point2 = np.array([points[1][0],points[1][1],points[1][2]])
158         Point3 = np.array([points[2][0],points[2][1],points[2][2]])
159 
160         N1 = np.array([points[0][3],points[0][4],points[0][5]])
161         N2 = np.array([points[1][3],points[1][4],points[1][5]])
162         N3 = np.array([points[2][3],points[2][4],points[2][5]])
163 
164         """Se calcula el producto cruz de cada par y se promedian entre si"""
165         Orientation = (np.cross(N1,N2)+np.cross(N1,N3)+np.cross(N2,N3))/3
166 
167         """Producto cruz entre la normal calculada anteriormente y las 3 normales iniciales"""
168         N4 = np.cross(N1,Orientation)
169         N5 = np.cross(N2,Orientation)
170         N6 = np.cross(N3,Orientation)
171 
172         try:
173             """Generacion de planos que pasan por los puntos elegidos con la normal calculada
    ↪ """
174             Plane1 = Plane(point=Point1, normal=N4)
175             Plane2 = Plane(point=Point2, normal=N5)
176             Plane3 = Plane(point=Point3, normal=N6)
177 
178             """A partir de los planos, se intersectan de a pares para generar 3 intersecciones"""
```

```
        Line1 = Plane1.intersect_plane(Plane2)
        Line2 = Plane1.intersect_plane(Plane3)
        Line3 = Plane2.intersect_plane(Plane3)

        """Linea que define al eje del cilindro"""
        Pnt = (Line1.point + Line2.point + Line3.point)/3
        Direction = (Line1.direction + Line2.direction + Line3.direction)/3
        line = Line(point=Pnt, direction=Direction)


        """Cálculo del radio"""
        R1 = calc_singlepointdist_to_line(Point1,line.point,line.direction)
        R2 = calc_singlepointdist_to_line(Point2,line.point,line.direction)
        R3 = calc_singlepointdist_to_line(Point3,line.point,line.direction)
        R = np.asarray([R1, R2, R3])
        maxR = max(R)
        R = R/maxR
        for i in range(len(R)):
            if R[i] < 0.75:
                R[i] = max(R)
        R = R*maxR
        R = R.mean()

        """Se descartan soluciones que generen un radio más grande que toda la PCD"""
        if R > max_dim*0.5:
            continue

        radius = R
        position_optimized = line.point
        orientation_optimized = line.direction

        """Threshold for comparison"""
        UmDist = radius*Thresh
        lowerRadius = radius-UmDist
        upperRadius = radius+UmDist
        comp_R = calc_dist_to_line(vertices,position_optimized, orientation_optimized)
        comparison = (comp_R > lowerRadius) & (comp_R < upperRadius)
        loss = sum(abs(comp_R - radius))**2
        Local_Score = comparison.sum()
        RANSAC_Score = Local_Score
        Data_Iterations[iteration] = [radius,line.point,line.direction], RANSAC_Score

        if RANSAC_Score > Score and RANSAC_Score < Max_Possible_Score:# and loss
    ↪ < Loss:
            Score = RANSAC_Score
            Best_Sample = radius, position_optimized[0], position_optimized[1],
    ↪ position_optimized[2], orientation_optimized[0], orientation_optimized[1],
    ↪ orientation_optimized[2]
            Best_Sample_Position = iteration+1
            Best_Comparison = comparison
            Loss = loss
            print("Iteration: {} (Run {} of {})".format(iteration+1,j+1,Big_Iterations))
```

```
228                 print('Current Radius: {:.5} mm'.format(Best_Sample[0]))
229                 print("Current Position: X: {:.5}, Y:{:.5}, Z:{:.5}".format(Best_Sample[1],
    ↪ Best_Sample[2],Best_Sample[3]))
230                 print("Current Orientation: X: {:.5}, Y:{:.5}, Z:{:.5}".format(Best_Sample[4],
    ↪ Best_Sample[5],Best_Sample[6]))
231                 print("RANSAC Score: {}, Score: {} (Found in Iteration {})".format(
    ↪ RANSAC_Score,Score,Best_Sample_Position))
232                 print("----------")
233             else:
234                 print("Iteration: {} (Run {} of {})".format(iteration+1,j+1,Big_Iterations))
235                 print('Current Radius: {:.5} mm'.format(Best_Sample[0]))
236                 print("RANSAC Score: {}, Score: {} (Found in Iteration {})".format(
    ↪ RANSAC_Score,Score,Best_Sample_Position))
237                 print("----------")
238         except:
239             continue
240
241     stop_time_internal = timeit.default_timer()
242     tiempo = stop_time_internal - start_time_internal
243
244 stop_time = timeit.default_timer()
245
246 #%%
247 """Parámetros para construcción de primitiva de cilindro"""
248 Cylinder_Height = Best_Sample[0]/10
249 Cylinder_Radius = Best_Sample[0]
250 Centroid = [Best_Sample[1],Best_Sample[2],Best_Sample[3]]
251
252 """Pintado de todos los puntos como rojos, y posterior pintado de los inliers como plomos"""
253 if pcd:
254     Best_Comparison_Positions = np.asarray(np.where(Best_Comparison == True)[0])
255     pcd.paint_uniform_color([1,0,0])
256     Inliers = list(Best_Comparison_Positions)
257     np.asarray(pcd.colors)[Inliers] = [0.5,0.5,0.5]
258
259 """Traslación de la PCD y generación de matrices de rotación"""
260 if pcd:
261     pcd = pcd.translate((-Best_Sample[1],-Best_Sample[2],-Best_Sample[3]), relative=False)
262 if mesh:
263     if type(mesh) == trimesh.base.Trimesh:
264         mesh.apply_translation((-Best_Sample[1],-Best_Sample[2],-Best_Sample[3]))
265     elif type(mesh) == o3d.cpu.pybind.geometry.TriangleMesh:
266         mesh = mesh.translate((-Best_Sample[1],-Best_Sample[2],-Best_Sample[3]), relative=
    ↪ False)
267
268 Vector_Orientacion = [Best_Sample[4],Best_Sample[5],Best_Sample[6]]
269 normalizado_orientacion = max(abs(np.asarray(Vector_Orientacion)))
270 Vector_Orientacion = Vector_Orientacion/normalizado_orientacion
271
272 Rotate = Rotation.align_vectors([Vector_Orientacion], [[0,0,1]])
273 Inv_Rotate = Rotation.align_vectors([[0,0,1]],[Vector_Orientacion])
274 New_Orientation = Rotate[0].apply(Vector_Orientacion)
```

```
normalizado_nuevo = max(abs(np.asarray(New_Orientation)))
New_Orientation = New_Orientation/normalizado_nuevo

coord_frame = o3d.geometry.TriangleMesh.create_coordinate_frame(size=60, origin=[0, 0, 0])

"""For rotation of pcd and alignment with Z axis for calculation of whole PCD size"""
if pcd:
    pcd_points = np.asarray(pcd.points)
    pcd_points = Inv_Rotate[0].apply(pcd_points)
    Centroid = Inv_Rotate[0].apply(Centroid)
    Cylinder_Height = max(pcd_points[:,2])-min(pcd_points[:,2])
    pcd = pcd.translate((0,0,-Centroid[2]), relative=False)
    pcd_points = Rotate[0].apply(pcd_points)
    pcd.points = o3d.utility.Vector3dVector(pcd_points)

if mesh:
    mesh_points = np.asarray(mesh.vertices)
    mesh_points = Inv_Rotate[0].apply(mesh_points)
    Centroid = Inv_Rotate[0].apply(Centroid)
    Cylinder_Height = 2*(max(mesh_points[:,2])-min(mesh_points[:,2]))
    if type(mesh) == trimesh.base.Trimesh:
        mesh.apply_translation((0,0,-Centroid[2]))
    elif type(mesh) == o3d.cpu.pybind.geometry.TriangleMesh:
        mesh.translate((0,0,-Centroid[2]), relative=False)
    mesh_points = Rotate[0].apply(mesh_points)
    mesh.vertices = o3d.utility.Vector3dVector(mesh_points)

RANSAC_Cylinder = o3d.geometry.TriangleMesh.create_cylinder(radius=Cylinder_Radius, height=Cylinder_Height,resolution=100)
RANSAC_Cylinder.paint_uniform_color((0.75,0.75,0.75))

"""Rotation of primitive and alignment with PCD"""
RANSAC_Cylinder_Vertex = np.asarray(RANSAC_Cylinder.vertices)
RANSAC_Cylinder_Vertex = Rotate[0].apply(RANSAC_Cylinder_Vertex)
RANSAC_Cylinder.vertices = o3d.utility.Vector3dVector(RANSAC_Cylinder_Vertex)
WireFrame = o3d.geometry.LineSet.create_from_triangle_mesh(RANSAC_Cylinder)
if pcd:
    o3d.visualization.draw_geometries([WireFrame, pcd])
elif mesh:
    if type(mesh) == trimesh.base.Trimesh:
        mesh_o3d = mesh.as_open3d
        # mesh_o3d = mesh_o3d.translate((0,0,0), relative=False)
        o3d.visualization.draw_geometries([WireFrame,mesh_o3d])
    elif type(mesh) == o3d.cpu.pybind.geometry.TriangleMesh:
        o3d.visualization.draw_geometries([coord_frame, mesh])

print('---------------------------------')
print('Time: {:.5} seconds'.format(stop_time - start_time))
print('Radius: {:.5} mm'.format(Best_Sample[0]))
print("Original Orientation: X: {:.5f}, Y:{:.5f}, Z:{:.5f}".format(Vector_Orientacion[0], Vector_Orientacion[1],Vector_Orientacion[2]))
```

A.2. Método propuesto conos

Código A.2: Código desarrollado para determinar parámetros de un cono

```
"""@author: Erick Kracht G."""
#%% RANSAC-3D
"""Import Libraries and Local Functions"""

import open3d as o3d
import numpy as np
import random
import timeit
import sys
from scipy.spatial.transform import Rotation

def calc_dist_to_line(point, position, orientation):
    """ Calculate the distance of a 3D point to a 3D line that passes through (xc, yc, zc) with
    ↪ direction
    (sx, sy, sz) """
    point = np.asarray(point)
    position = np.asarray(position)
    orientation = np.asarray(orientation)
    Dist = position - point
    D = np.cross(Dist,orientation)
    return (np.linalg.norm(D, axis=1))/(np.linalg.norm(orientation))

def calc_singlepointdist_to_line(point, position, orientation):
    """ Calculate the distance of a 3D point to a 3D line that passes through (xc, yc, zc) with
    ↪ direction
    (sx, sy, sz) """
    point = np.asarray(point)
    position = np.asarray(position)
    orientation = np.asarray(orientation)
    Dist = position - point
    D = np.cross(Dist,orientation)
    return (np.linalg.norm(D))/(np.linalg.norm(orientation))

def dist_pts3d(x,y):
    """ Calculate distance between two 3D points """
    x = np.asarray(x)
    x = x.T
    return np.sqrt((x[0]-y[0])**2 + (x[1]-y[1])**2 + (x[2]-y[2])**2)

def direction_vector(x,y):
    """ Calculate direction of vector """
    return (x[0]-y[0])/dist_pts3d(x,y) , (x[1]-y[1])/dist_pts3d(x,y) , (x[2]-y[2])/dist_pts3d(x,y
    ↪ )

def dist_point_to_cone(cone_args,point):
    """ Calculate the distance of single point to a cone """
    phi, posX, posY, posZ, oriX, oriY, oriZ = cone_args
```

```
45      position = [posX, posY, posZ]
46      orientation = [oriX, oriY, oriZ]
47
48      Xr = calc_dist_to_line(point,position,orientation)
49      Xh = np.sqrt((dist_pts3d(point,position))**2-Xr**2)
50
51      dist = Xr*np.cos(phi) - (Xh)*np.sin(phi)
52      return abs(dist)
53
54  def dist_points_to_cone(cone_args,points):
55      """ Calculate the sum of distance of multiple points to a cone """
56      Distancias = dist_point_to_cone(cone_args,points)
57      return sum(Distancias)**2
58
59  def project_point_to_line(point,orientation):
60      """ Projects a point to a line with certain orientation """
61      p = np.asarray(point)
62      p = np.reshape(p,(-1,3))
63      s = np.asarray(orientation)
64      s = np.reshape(s,(-1,3))
65      s = s.T
66      projection = p@s/np.linalg.norm(s)*s.T
67      return projection
68
69  def pick_points(pcd):
70      print("")
71      print(
72          "1) Please pick at least three correspondences using [shift + left click]"
73      )
74      print("   Press [shift + right click] to undo point picking")
75      print("2) After picking points, press 'Q' to close the window")
76      vis = o3d.visualization.VisualizerWithEditing()
77      vis.create_window()
78      vis.add_geometry(pcd)
79      vis.run()  # user picks points
80      vis.destroy_window()
81      print("")
82      return vis.get_picked_points()
83
84  def in_sphere(radio,center,points):
85      radios = dist_pts3d(points,center)
86      posiciones = radios < radio
87      contenidos = points[np.where(posiciones == 1)[0]]
88      return contenidos
89
90  #%% Data Input and Pre-Processing
91  start_time = timeit.default_timer()
92  pcd = []
93  Check = 0
94  if Check == 0:
95
96      """TESIS"""
```

```
97  # mesh = o3d.io.read_triangle_mesh("STLs/Conos/aguja_bolsillo_rotadaTesis.stl")
98  # mesh = o3d.io.read_triangle_mesh("STLs/Conos/aguja_pitting_rotadaTesis.stl")
99  mesh = o3d.io.read_triangle_mesh("STLs/Conos/aguja_escaneada_venas.stl")
100 pcd = mesh.sample_points_uniformly(number_of_points=500001)
101 mesh = []
102 if pcd and mesh:
103     print('Hay una pcd y un mesh cargados, escoger uno solo')
104     sys.exit()
105 elif pcd == [] and mesh == []:
106     print('No hay pcd ni mesh seleccionados, escoger uno de los dos')
107     sys.exit()
108
109 if pcd:
110     """Centrado de PCD en origen"""
111     pcd = pcd.translate((0,0,0), relative=False)
112
113     """Carga de puntos de la pcd como un array."""
114     pcd_array = np.asarray(pcd.points)
115     pcd_normals = np.asarray(pcd.normals)
116     # pcd_array = np.load('pcd_array.npy')
117     axis_1 = max(pcd_array[:,0])-min(pcd_array[:,0])
118     axis_2 = max(pcd_array[:,1])-min(pcd_array[:,1])
119     axis_3 = max(pcd_array[:,2])-min(pcd_array[:,2])
120     max_axis = axis_1, axis_2, axis_3
121     max_dim = max(max_axis)
122
123     """Selección de los puntos con sus respectivas normales"""
124     Points_With_Normals = np.array(pcd.points)[:,0],np.array(pcd.points)[:,1],np.array(
    ↪ pcd.points)[:,2],np.array(pcd.normals)[:,0],np.array(pcd.normals)[:,1],np.array(pcd.
    ↪ normals)[:,2]
125
126 elif mesh:
127     """Centrado de Mesh en origen"""
128     mesh = mesh.translate((0,0,0), relative=False)
129
130     """Carga de vertices del mesh como un array."""
131     vertices_full = np.asarray(mesh.vertices)
132     mesh.compute_vertex_normals()
133     normals = np.asarray(mesh.triangle_normals)
134     normales = list(range(len(vertices_full)))
135     for i in range(len(normales)):
136         normales[i] = normals[int(np.floor(i/3))]
137     normales = np.asarray(normales)
138     triangles = np.asarray(mesh.triangles)
139     axis_1 = max(vertices_full[:,0])-min(vertices_full[:,0])
140     axis_2 = max(vertices_full[:,1])-min(vertices_full[:,1])
141     axis_3 = max(vertices_full[:,2])-min(vertices_full[:,2])
142     max_axis = axis_1, axis_2, axis_3
143     max_dim = max(max_axis)
144
145     """Selección de los puntos con sus respectivas normales"""
146     Points_With_Normals = vertices_full[:,0],vertices_full[:,1],vertices_full[:,2],normales
```

```
    ↪ [:,0],normales[:,1],normales[:,2]

Points_With_Normals = np.array(Points_With_Normals).T
List_PointsWithNormals_FullSize = list(Points_With_Normals)
if len(List_PointsWithNormals_FullSize) > 500000:
    List_PointsWithNormals = random.sample(List_PointsWithNormals_FullSize,500000)
    PointsWithNormals = np.asarray(List_PointsWithNormals)
    vertices = PointsWithNormals[:,0], PointsWithNormals[:,1], PointsWithNormals[:,2]
    vertices = np.asarray(vertices)
    vertices = vertices.T
else:
    List_PointsWithNormals = List_PointsWithNormals_FullSize
    vertices = vertices_full

#%% Section for iteration of RANSAC

Big_Iterations = 1
for j in range(Big_Iterations):
    start_time_internal = timeit.default_timer()
    if pcd:
        NumIter = 500
    elif mesh:
        NumIter = 1000
    Thresh = 0.001*max_dim
    Score = 0
    Best_Sample = [0, 0, 0, 0, 0, 0, 0]
    Best_Sample_Position = 0
    Data_Iterations = list(range(NumIter))
    Best_Comparison = 0

    for iteration in range(NumIter):
        Local_Score = 0
        points = np.asarray(random.sample(List_PointsWithNormals,3))

        Point1 = np.array([points[0][0],points[0][1],points[0][2]])
        Point2 = np.array([points[1][0],points[1][1],points[1][2]])
        Point3 = np.array([points[2][0],points[2][1],points[2][2]])
        N1 = np.array([points[0][3],points[0][4],points[0][5]])
        N2 = np.array([points[1][3],points[1][4],points[1][5]])
        N3 = np.array([points[2][3],points[2][4],points[2][5]])

        try:

            a = np.array([list(N1),list(N2),list(N3)])
            b = np.array([np.dot(Point1,N1),np.dot(Point2,N2),np.dot(Point3,N3)])
            x1 = np.linalg.solve(a,b)
            q = (N1[0]*(Point1[1]-Point2[1])+N1[1]*(Point2[0]-Point1[0]))/(N1[0]*N2[1]-N1
    ↪ [1]*N2[0])
            x2 = np.array([Point2[0]+q*N2[0], Point2[1]+q*N2[1], Point2[2]+q*N2[2]])
            x = direction_vector(x1,x2)
            Angles = np.array([np.arcsin(calc_singlepointdist_to_line(Point1,x1,x)/
    ↪ dist_pts3d(x1,Point1)),np.arcsin(calc_singlepointdist_to_line(Point2,x1,x)/
```

```
    ↪ dist_pts3d(x1,Point2)),np.arcsin(calc_singlepointdist_to_line(Point3,x1,x)/
    ↪ dist_pts3d(x1,Point3))])
196             Apex = x1
197             Angle = Angles.mean()
198             if Angle < np.radians(5) or Angle > np.radians(90):
199                 continue
200             Direction = x
201             cone_args = [Angle, Apex[0], Apex[1], Apex[2], Direction[0], Direction[1],
    ↪ Direction[2]]
202 
203             """Comparison and scoring of points"""
204             if pcd:
205                 comparison = dist_point_to_cone(cone_args,pcd_array) < Thresh
206             elif mesh:
207                 comparison = dist_point_to_cone(cone_args,vertices) < Thresh
208             Local_Score = comparison.sum()
209             RANSAC_Score = Local_Score
210             Data_Iterations[iteration] = RANSAC_Score, cone_args
211 
212             if RANSAC_Score > Score:
213                 Score = RANSAC_Score
214                 Best_Sample = cone_args
215                 Best_Sample_Position = iteration+1
216                 Best_Comparison = comparison
217 
218                 print("Iteration: {} (Run {} of {})".format(iteration+1,j+1,Big_Iterations))
219                 print("Current Aperture Angle (in degrees):", abs((Best_Sample[0]*180/np.pi)
    ↪  %180))
220                 print("Current Apex Position: X: {}, Y:{}, Z:{}".format(Best_Sample[1],
    ↪ Best_Sample[2],Best_Sample[3]))
221                 print("Current Orientation: X: {}, Y:{}, Z:{}".format(Best_Sample[4],
    ↪ Best_Sample[5],Best_Sample[6]))
222                 print("RANSAC Score: {}, Score: {} (Found in Iteration {})".format(
    ↪ RANSAC_Score,Score,Best_Sample_Position))
223                 print("----------")
224 
225             else:
226                 print("Iteration: {} (Run {} of {})".format(iteration+1,j+1,Big_Iterations))
227                 print("Current Aperture Angle (in degrees):", abs((Best_Sample[0]*180/np.pi)
    ↪  %180))
228                 print("RANSAC Score: {}, Score: {} (Found in Iteration {})".format(
    ↪ RANSAC_Score,Score,Best_Sample_Position))
229                 print("----------")
230           except:
231             continue
232 
233       stop_time_internal = timeit.default_timer()
234       tiempo = stop_time_internal - start_time_internal
235 
236 #%%
237 """Matrices de rotacion"""
238 Vector_Orientacion = [Best_Sample[4],Best_Sample[5],Best_Sample[6]]
```

```
Rotate = Rotation.align_vectors([Vector_Orientacion], [[0,0,1]]) # Para rotar primitiva a orientacion de pcd
Inv_Rotate = Rotation.align_vectors([[0,0,1]],[Vector_Orientacion]) # Para rotar pcd a orientación [0,0,1]

"""Parametros para la generación de la primitiva del cono desde PCD"""
Apex_Position = [Best_Sample[1],Best_Sample[2],Best_Sample[3]]
if pcd:
    Max_Generatriz = max(dist_pts3d(Apex_Position,pcd_array.T))
elif mesh:
    Max_Generatriz = max(dist_pts3d(Apex_Position,vertices.T))

"""Pintado de PCD o Mesh. Si es PCD se pintan inliers y outliers"""
Best_Comparison_Positions = np.asarray(np.where(Best_Comparison == True)[0])
Inliers = list(Best_Comparison_Positions)
if pcd:
    pcd.paint_uniform_color([1,0,0])
    np.asarray(pcd.colors)[Inliers] = [0.5,0.5,0.5]
elif mesh:
    mesh.paint_uniform_color([0.5,0.5,0.5])

coord_frame = o3d.geometry.TriangleMesh.create_coordinate_frame(size=60, origin=[0, 0, 0])

"""Definicion Cono Primitivo"""
Cone_Height = Max_Generatriz*np.cos(Best_Sample[0])
Cone_Radius = abs(np.sin(Best_Sample[0])*Max_Generatriz)
RANSAC_Cone = RANSAC_Cone = o3d.geometry.TriangleMesh.create_cone(radius=Cone_Radius,height=Cone_Height,resolution=100)
RANSAC_Cone.paint_uniform_color((0.75,0.75,0.75))

"""Rotation of primitive and alignment with PCD/Mesh"""
if pcd:
    pcd_array = Inv_Rotate[0].apply(pcd_array) #PCD alineada con [0,0,1]
    Desfase_Altura = abs(min(pcd_array[:,2]))
    pcd_array[:,2] = pcd_array[:,2]+Desfase_Altura #Se corrige de manera que PCD tenga su base en plano XY
    pcd.points = o3d.utility.Vector3dVector(pcd_array)
elif mesh:
    """For rotation of mesh and alignment with Z axis for calculation of whole STL size"""
    mesh_points = np.asarray(mesh.vertices)
    mesh_points = Inv_Rotate[0].apply(mesh_points)
    Desfase_Altura = abs(min(mesh_points[:,2]))
    mesh_points[:,2] = mesh_points[:,2]+Desfase_Altura #Se corrige de manera que PCD tenga su base en plano XY
    mesh.vertices = o3d.utility.Vector3dVector(mesh_points)

"""Traslation of PCD for alignment with Primitive figure"""
Apex_Position = np.asarray(Apex_Position)
Apex_Position = Inv_Rotate[0].apply(Apex_Position)
if pcd:
    pcd_center = pcd.get_center()
```

```
285     pcd = pcd.translate((-Apex_Position[0],-Apex_Position[1],pcd_center[2]),relative=False)
286 elif mesh:
287     mesh_center = mesh.get_center()
288     mesh = mesh.translate((-Apex_Position[0],-Apex_Position[1],mesh_center[2]),relative=
        ↪ False)
289
290 stop_time = timeit.default_timer()
291
292 WireFrame = o3d.geometry.LineSet.create_from_triangle_mesh(RANSAC_Cone)
293 if pcd:
294     o3d.visualization.draw_geometries([coord_frame, pcd])
295     o3d.visualization.draw_geometries([coord_frame, WireFrame, pcd])
296 elif mesh:
297     o3d.visualization.draw_geometries([coord_frame, mesh])
298     o3d.visualization.draw_geometries([coord_frame, WireFrame, mesh])
299
300 print('-------------------------')
301 print('Cone Angle : {:.5f}'.format(abs((Best_Sample[0]*180/np.pi) %180)))
302 print("Apex Position: X: {:.5f}, Y:{:.5f}, Z:{:.5f}".format(Best_Sample[1],Best_Sample[2],
        ↪ Best_Sample[3]))
303 print("Orientation: X: {:.5f}, Y:{:.5f}, Z:{:.5f}".format(Best_Sample[4],Best_Sample[5],
        ↪ Best_Sample[6]))
304 print('Time: {:.5} seconds'.format(stop_time - start_time))
```

A.3. Segmentos de código utilizados durante el trabajo

Código A.3: Segmento de código del método de mínimos cuadrados para detectar parámetros de cilindro

```
1  for j in range(Big_Iterations):
2      start_time_internal = timeit.default_timer()
3      NumIter = 750
4      Thresh = 0.02 # Percentage, use with radius
5      Sample_Points_RANSAC = 10
6      Percentage_Reduction = 1 #Percentage for reducing the size of points, use values
       ↪ between 0 and 1.
7      Score = 0
8      Iteration = 0
9      Best_Sample = [0, 0, 0, 0, 0, 0, 0, 0]
10
11     # A sample of a  % of the total points is used for comparison of inliers in order to reduce
       ↪ computing time
12     Big_Sample = random.sample(List_PointsWithNormals,int(len(List_PointsWithNormals)
       ↪ *Percentage_Reduction))
13     Big_Sample = np.asarray(Big_Sample)
14     Big_Sample = Big_Sample.T
15
16     Big_Sample_Score = len(Big_Sample.T)
17
18     for iteration in range(NumIter):
```

```
        Local_Score = 0
        points = np.asarray(random.sample(List_PointsWithNormals,
    ↪ Sample_Points_RANSAC))

        # Estimated coordinates of cylinder center
        position_0 = points[:,0].mean(), points[:,1].mean(), points[:,2].mean()
        positionX_0 = position_0[0]
        positionY_0 = position_0[1]
        positionZ_0 = position_0[2]

        # First aproximation of cylinder orientation
        orientation_0 = vectors[0]
        orientationX_0 = orientation_0[0]
        orientationY_0 = orientation_0[1]
        orientationZ_0 = orientation_0[2]

        # Random starting value for length of cylinder
        length_0 = random.random()

        # First aproximation of radius of cylinder
        radius_0 = calc_dist_to_line(points, position_0, orientation_0).mean()

        # First Cylinder parameters for fitting
        x0 = [radius_0, positionX_0, positionY_0, positionZ_0, orientationX_0,
    ↪ orientationY_0, orientationZ_0, length_0]

        # params = opt.minimize(dist_cyl,x0,args=(points.T,))
        params = opt.least_squares(dist_cyl,x0,args=(points.T,))

        # Optimized parameters
        radius, posX, posY, posZ, orientationX, orientationY, orientationZ, length = params.x
        position_optimized = posX, posY, posZ
        orientation_optimized = orientationX, orientationY, orientationZ
```

Código A.4: Segmento de código del método de círculo proyectado para detectar parámetros de cilindro

```
for j in range(Big_Iterations):
    start_time_internal = timeit.default_timer()
    NumIter = 750
    Thresh = 0.05 # Percentage, use with radius
    Score = 0
    Best_Sample = [0, 0, 0, 0, 0, 0, 0, 0]
    Best_Sample_Position = 0
    Data_Iterations = list(range(NumIter))
    Best_Comparison = 0
    Loss = 10**20

    Max_Possible_Score = len(vertices)

    for iteration in range(NumIter):
        # start_time = timeit.default_timer()
```

```
        if Loss < 1:
            break
        """Se eligen 3 puntos al azar, se extraen sus posiciones XYZ y sus normales"""
        pts = np.asarray(random.sample(list_vertices,30))

        values, vectors = eig(cov((pts - mean(pts.T, axis=1)).T))

        Direction = vectors[np.where(values == values.min())].T[:,0]
        Direction3x1 = vectors[np.where(values == values.min())].T

        proyectados = projected_points_to_plane(pts,Direction.T[0])

        # Coordenadas del baricentro (estimación inicial del centro del círculo)
        x_m = pts[0].mean()
        y_m = pts[1].mean()
        z_m = pts[2].mean()
        barycenter = x_m, y_m, z_m

        circle_center, ier = opt.leastsq(f_2, barycenter, args=(pts[0],pts[1],pts[2]))

        try:

            line = Line(point=circle_center, direction=Direction)

            """Cálculo del radio"""
            R = calc_dist_to_line(pts,circle_center,Direction)
            maxR = max(R)
            R = R/maxR
            for i in range(len(R)):
                if R[i] < 0.75:
                    R[i] = max(R)
            R = R*maxR
            R = R.mean()

            """Se descartan soluciones que generen un radio más grande que toda la PCD"""
            if R > max_dim*0.5:
                continue

            radius = R
            position_optimized = line.point
            orientation_optimized = line.direction
```

Código A.5: Segmento de código del método de mínimos cuadrados para detectar parámetros de cono

```
for j in range(Big_Iterations):
    start_time_internal = timeit.default_timer()
    if pcd:
        NumIter = 50
    elif mesh:
        NumIter = 1000
    Thresh = 0.05*max_dim
```

```
    Score = 0
    Best_Sample = [0, 0, 0, 0, 0, 0, 0]
    Best_Sample_Position = 0
    Data_Iterations = list(range(NumIter))
    Best_Comparison = 0

    for iteration in range(NumIter):
        Local_Score = 0
        points = np.asarray(random.sample(List_PointsWithNormals,3))

        pt1 = np.asarray([points[0][0],points[0][1],points[0][2]])
        pt2 = np.asarray([points[1][0],points[1][1],points[1][2]])
        pt3 = np.asarray([points[2][0],points[2][1],points[2][2]])

        pts = pt1, pt2, pt3
        pts = np.asarray(pts)

        values, vectors = eig(cov((pts - mean(pts.T, axis=1)).T))

        # Estimated coordinates of cone center
        position_0 = points[:,0].mean(), points[:,1].mean(), points[:,2].mean()
        positionX_0 = position_0[0]
        positionY_0 = position_0[1]
        positionZ_0 = position_0[2]

        # First aproximation of cone orientation
        orientation_0 = vectors[np.where(values == values.min())].T
        orientationX_0 = orientation_0[0][0]
        orientationY_0 = orientation_0[1][0]
        orientationZ_0 = orientation_0[2][0]

        # First angle for cone
        angle_0 = np.radians(45)

        x0 = [angle_0, positionX_0, positionY_0, positionZ_0, orientationX_0,
    ↪ orientationY_0, orientationZ_0]


        try:


            params = opt.least_squares(dist_points_to_cone,x0,args=(points.T,))
            Angle, PosX, PosY, PosZ, DirectionX, DirectionY, DirectionZ = params.x

            cone_args = [Angle, PosX, PosY, PosZ, DirectionX, DirectionY, DirectionZ]
```

Código A.6: Segmento de código del método de intersección de planos para detectar parámetros de cono

```
from skspatial.objects import Line
from skspatial.objects import Plane
for j in range(Big_Iterations):
```

```
    start_time_internal = timeit.default_timer()
    if pcd:
        NumIter = 50
    elif mesh:
        NumIter = 1000
    Thresh = 0.001*max_dim

    Score = 0
    Best_Sample = [0, 0, 0, 0, 0, 0, 0]
    Best_Sample_Position = 0
    Data_Iterations = list(range(NumIter))
    Best_Comparison = 0

    for iteration in range(NumIter):
        Local_Score = 0
        points = np.asarray(random.sample(List_PointsWithNormals,3))

        Point1 = np.array([points[0][0],points[0][1],points[0][2]])
        Point2 = np.array([points[1][0],points[1][1],points[1][2]])
        Point3 = np.array([points[2][0],points[2][1],points[2][2]])
        N1 = np.array([points[0][3],points[0][4],points[0][5]])
        N2 = np.array([points[1][3],points[1][4],points[1][5]])
        N3 = np.array([points[2][3],points[2][4],points[2][5]])

        try:

            plane_A = Plane(Point1, N1)
            plane_B = Plane(Point2, N2)
            plane_C = Plane(Point3, N3)
            line_AxB = plane_A.intersect_plane(plane_B)
            Apex = plane_C.intersect_line(line_AxB)
            plano_aux = Plane.from_points(Apex + (Point1 - Apex)/(np.linalg.norm(Point1
↪  - Apex)),Apex + (Point2 - Apex)/(np.linalg.norm(Point2 - Apex)),Apex + (Point3 -
↪ Apex)/(np.linalg.norm(Point3 - Apex)))
            Direction = plano_aux.normal
            x = Direction
            x1 = Apex
            Angles = np.array([np.arcsin(calc_singlepointdist_to_line(Point1,x1,x)/
↪ dist_pts3d(x1,Point1)),np.arcsin(calc_singlepointdist_to_line(Point2,x1,x)/
↪ dist_pts3d(x1,Point2)),np.arcsin(calc_singlepointdist_to_line(Point3,x1,x)/
↪ dist_pts3d(x1,Point3))])
            Angle = Angles.mean()

            cone_args = [Angle, Apex[0], Apex[1], Apex[2], Direction[0], Direction[1],
↪ Direction[2]]
```

Anexo B

Gráficos adicionales

B.1. Gráficos utilizados para mostrar el error relativo en caso de cilindro con falla tipo pitting.

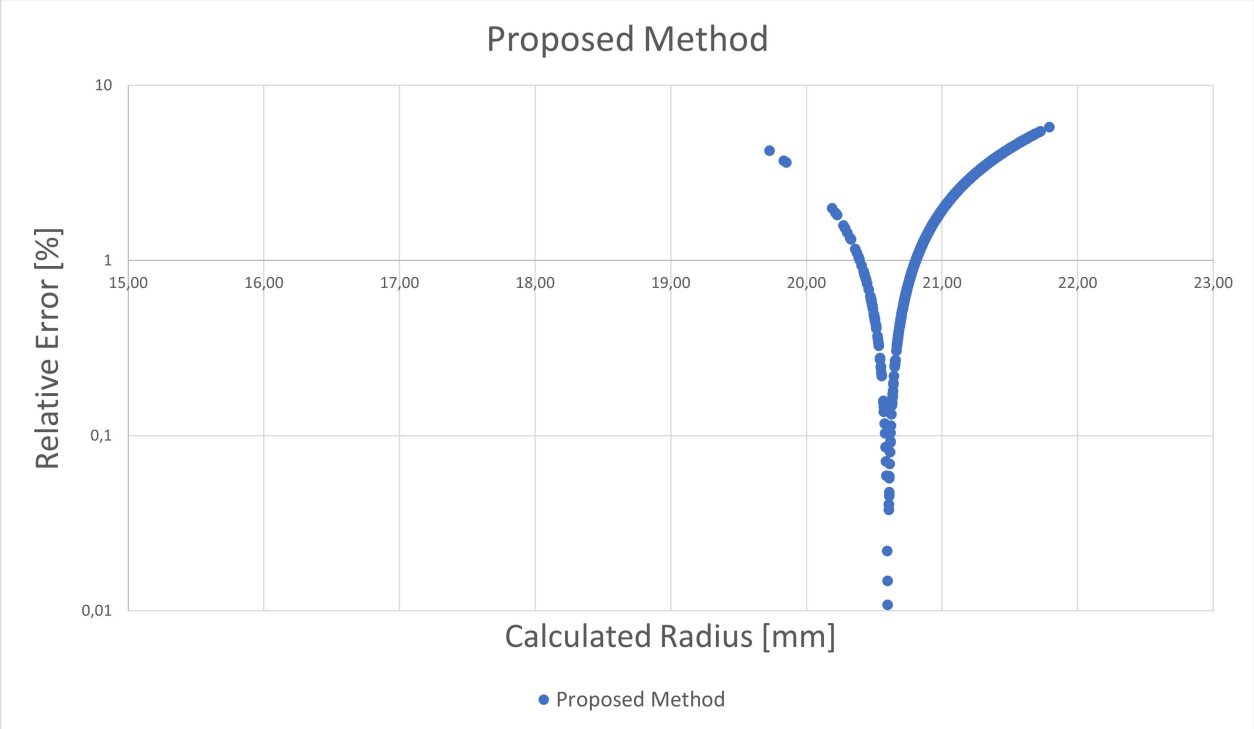


Figura B.1: Gráfico mostrando el error relativo para los radios calculados por método propio.

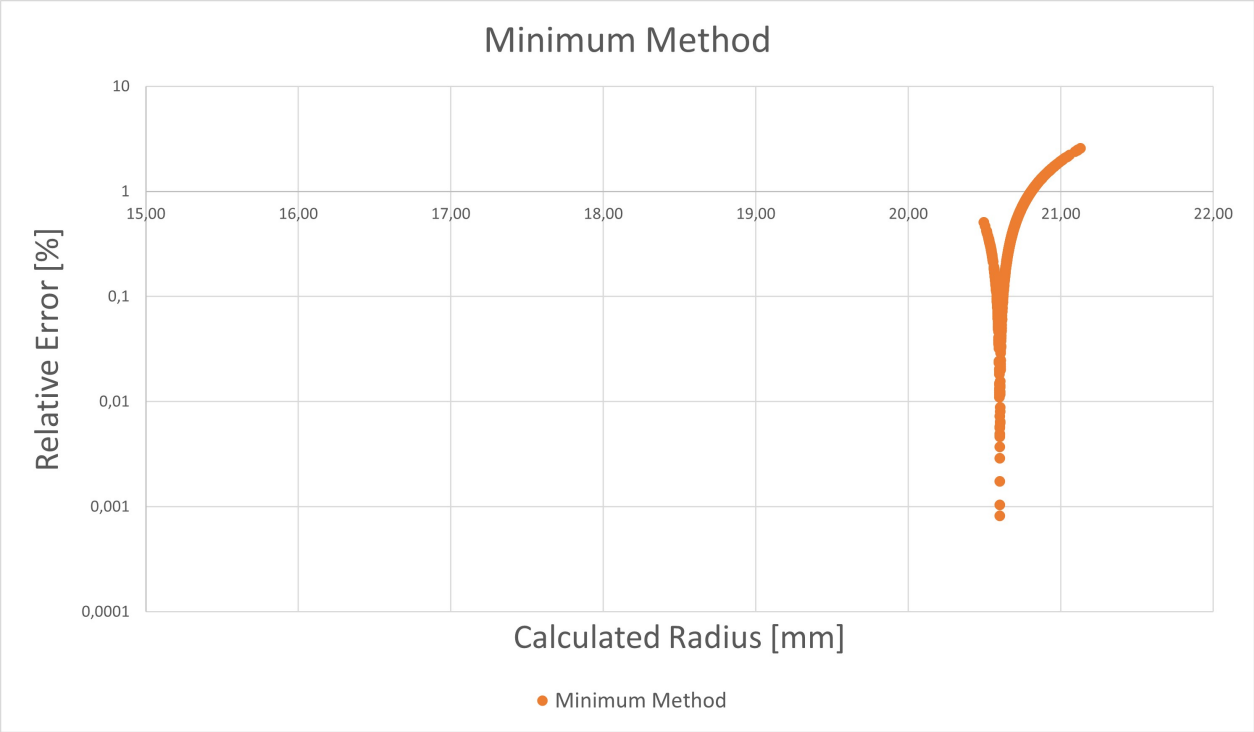


Figura B.2: Gráfico mostrando el error relativo para los radios calculados por método 1.

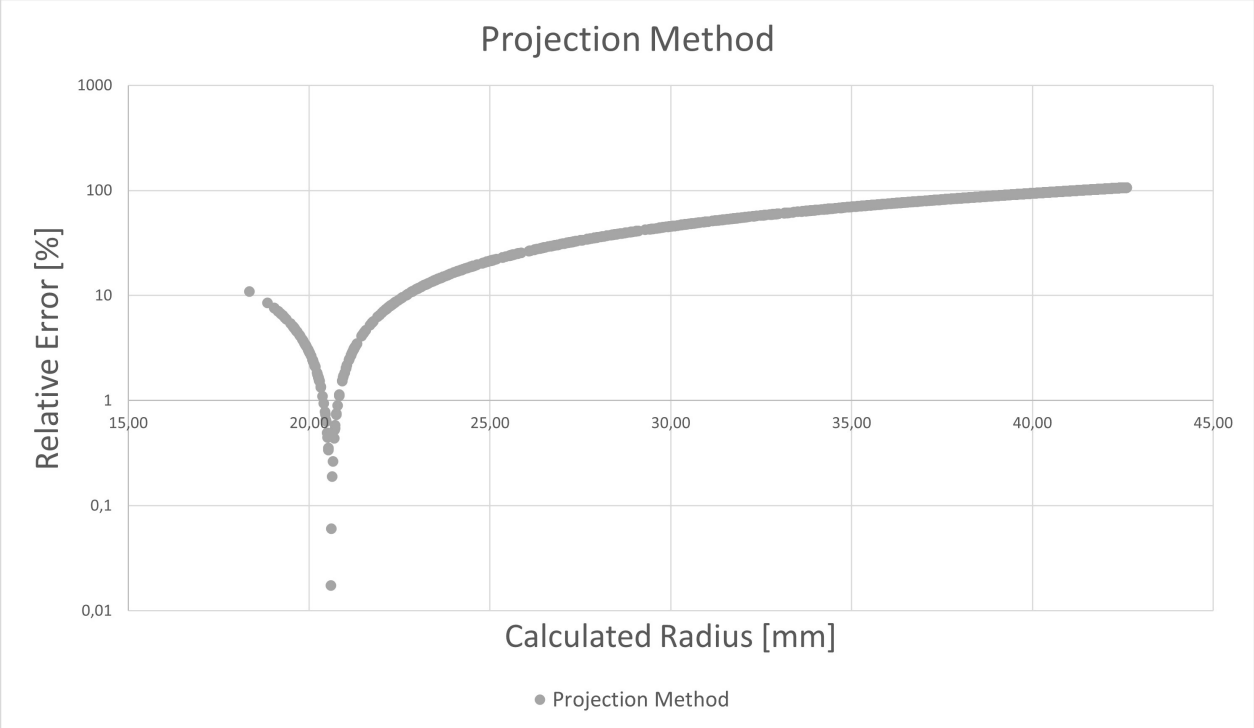


Figura B.3: Gráfico mostrando el error relativo para los radios calculados por método 2.

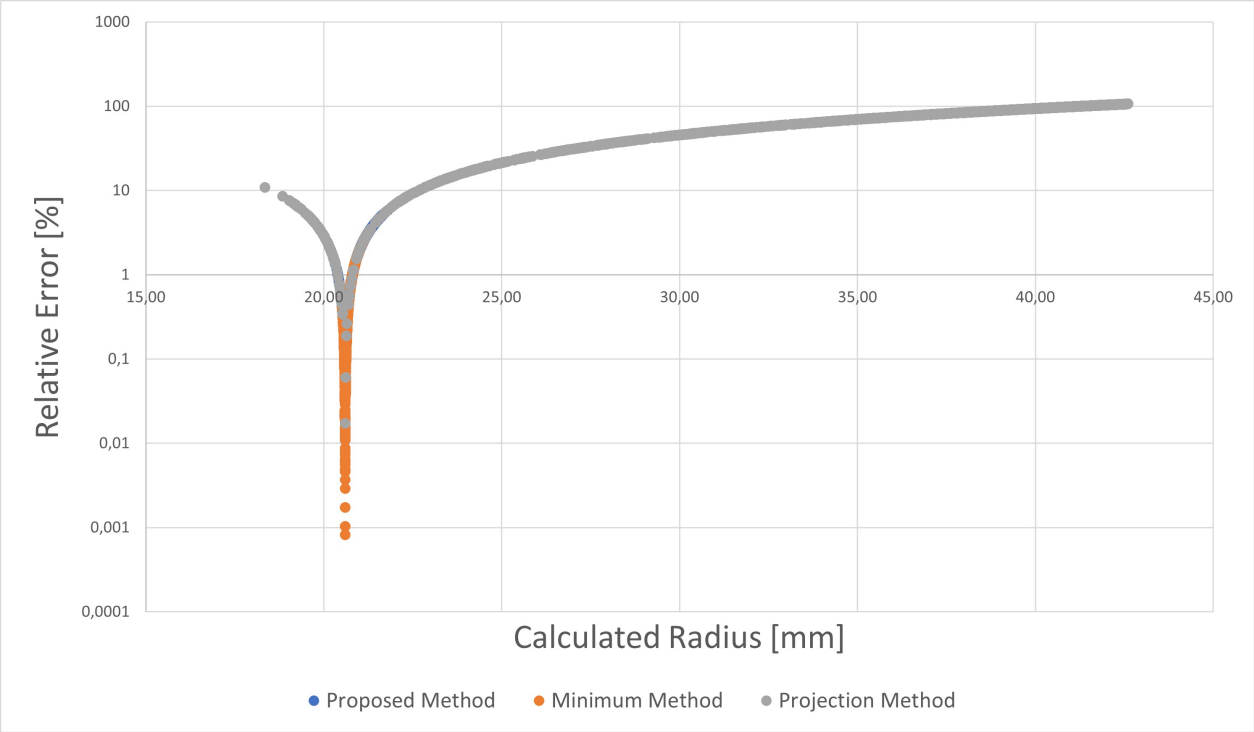


Figura B.4: Gráfico mostrando el error relativo para los radios calculados por método propio, método 1 y método 2.