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

DETERMINACIÓN DE PATRONES PARA SOLDADURA CON CAMINOS DE FORMA LIBRE, PARA RELLENO DE DEFECTOS EN LA RECUPERACIÓN DE PIEZAS DE ACERO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MECÁNICO

FABIÁN ENRIQUE LORCA MONCADA

PROFESOR GUÍA:
RUBÉN MARCOS FERNÁNDEZ URRUTIA

MIEMBROS DE LA COMISIÓN:
JUAN CRISTÓBAL ZAGAL MONTEALEGRE
PATRICIO FERNANDO MÉNDEZ

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE: Ingeniero Civil Mecánico
POR: Fabián Enrique Lorca Moncada
FECHA: 17/05/2021
PROFESOR GUÍA: Rubén Marcos Fernández Urrutia

DETERMINACIÓN DE PATRONES PARA SOLDADURA CON CAMINOS DE FORMA LIBRE, PARA RELLENO DE DEFECTOS EN LA RECUPERACIÓN DE PIEZAS DE ACERO

Actualmente existe una necesidad desde la industria de recuperar piezas de acero, debido a que los defectos en ella pueden implicar el perder la pieza y todos los costos económicos y temporales asociados a su reemplazo. Esta recuperación puede llevarse a cabo de forma robotizada con soldadura GMAW, sin embargo, debido a la gran variedad de formas de piezas y defectos, no se ha llegado a una solución general ni adaptativa.

En el presente trabajo se tiene por objetivo el determinar patrones para soldadura en relleno de defectos utilizando caminos de forma libre, para recuperación de piezas de acero. Al mismo tiempo, se tendrán los objetivos de que la estrategia generada no dependa de la forma del defecto a rellenar; que se realice de forma semi automática y analizar las trayectorias generadas. Para ello se utiliza un modelo del defecto en un archivo tipo nube de puntos (PCD). Se utiliza el lenguaje de programación Python y las trayectorias son generadas para ser recorridas por un brazo robótico.

El proceso de generación de trayectorias consiste en un preprocesamiento, del archivo de la falla, donde se ingresan también los parámetros de soldadura, a partir de ello se generan capas no planares equiespaciadas sobre la superficie a recuperar hasta completar el volumen de la falla. Estas capas son divididas para facilitar su recorrido y en esas nuevas nubes de punto se aplica un algoritmo que genera la trayectoria. Estas trayectorias son post procesadas para añadir los levantamientos de herramienta. Finalmente se visualiza y se traduce al lenguaje del brazo robótico, para simularlo en el programa DTPS, un software especializado del robot. Adicionalmente se revisa el desempeño de la generación de trayectorias y se exploran nuevas utilidades, como la generación de capas que envuelvan la pieza que contiene la falla y el relleno de formas limitadas por otros modelos no planares.

La estrategia descrita se prueba en superficies que simulan defectos de interés, logrando generar patrones de relleno para ellas, con resultados satisfactorios. Por otro lado, existen problemas puntuales en la generación de espirales, donde ocurren levantamientos de herramienta no deseados. Además, se hacen notar otras limitantes propias del brazo robótico como lo son la cantidad de comandos que lee el programa y el alcance del brazo robótico. Al revisar el potencial en las nuevas utilidades mencionadas, se llega a resultados aceptables en ambos escenarios, pudiendo generar una capa que envuelve a un cilindro y generar caminos de soldadura limitados por una superficie curva.

El método propuesto logra cumplir con el objetivo de rellenar defectos con patrones no planares de manera semiautomática, por lo que puede ser una alternativa viable a la hora de recuperar piezas, reduciendo tiempos de programación y permitiendo recuperar piezas de geometrías más complejas.

Agradecimientos

El haber llegado a esta etapa no fue una tarea fácil, y decir que hubiese sido posible sin ayuda ni apoyo de mis cercanos sería lo más lejano a la realidad. En primer lugar, mis agradecimientos van dirigidos a mis padres Sandra y Enrique, y a mi hermana Camila, quienes siempre pusieron mi bienestar y educación como prioridad máxima. Me formaron de la mejor forma que podría imaginar

A los profesores de la comisión Rubén Fernández por proponer el tema de la memoria y dirigir su desarrollo, a Juan Cristóbal Zagal por reafirmar mi interés por la carrera escogida en mis primeros años de universidad. A los miembros del equipo del proyecto 03 del programa IMA+ de la universidad, Alejandra Flores, Francisco Cubillos y Erick Kracht, por su guía y ayuda en el área.

Agradezco a mis amigos de infancia, principalmente Giovanni Tello y Mauricio, con quienes tengo una relación de apoyo incondicional. A Javiera Farías, por hacer el mundo más interesante con su amor y compañía. A Marcela Orellana, José Luis Puebla y Claudio Labrín, porque sé que puedo confiar ciegamente en ellos y por ser mi principal apoyo en más de la mitad de esta etapa. A Gabriel Ceballos y Matías Abarzúa, quienes cumplieron un rol de formación en mis valores y personalidad y son hoy grandes amigos.

Me gustaría también agradecer a quienes de alguna u otra forma me acompañaron o apoyaron en mi carrera.

A Andrea Vergara, por ser un ejemplo a seguir como ingeniera mecánica y por incentivarme a seguir participando en el Grupo organizado Anime no Seishin Doukoukai, donde aprendí bastante y encontré un lugar acogedor dentro de la universidad. Agradezco también a los miembros de Seishin, de generaciones previas y consecutivas a la mía, por generar un espacio grato para distintos tipos de personas.

A mis compañeros de sección, Kevin, Katalina, Camila, Lucas, Onofre, Lukas, Joaquín, Nicolás, Exequiel, Ignacio, Alex, Ricardo, José Tomás y Rodrigo, con quienes compartimos intereses y apoyo en la facultad. A las funcionarias de la biblioteca Susana y Claudia, que me recibieron siempre de la mejor forma al abrir la biblioteca, a todos quienes me acompañaron alguna vez a estudiar y compartir en ella. A Aaron, Álvaro, Cristian y Cristián, por generar discusiones de todo tipo y saber que puedo contar con ellos. A la funcionaria Silvia del departamento de mecánica, que siempre me recibió con alegría. A Jorge, Matías y Nicolás, compañeros de prácticas y proyectos que terminaron siendo amigos. A Sofía, por su amistad y ayuda en las áreas de programación.

Hay mucha gente más que podría agregar, ya que en Beauchef siempre encontré personas apasionadas e interesantes, quienes estuvieron dispuestos a compartir sus motivaciones conmigo. Extrañaré los almuerzos, los encuentros de pasillos y conversaciones apuradas, que alegraron mi día a día en la universidad y me ayudaron a llegar hasta aquí.

Tabla de contenido

Índice de tablas

Índice de ilustraciones

1 Introducción

Actualmente en distintas industrias como por ejemplo la minera, energética y forestal, se deben desechar múltiples piezas debido a su desgaste. La recuperación de defectos por medio de soldadura es una solución para prolongar la vida útil de las piezas y comenzar nuevos ciclos. El proceso requiere personal altamente calificado y además el soldador debe estar expuesto a condiciones de riesgo como altas temperaturas o la exposición a gases nocivos, además de que la calidad del proceso de soldadura puede variar según la experticia del soldador. Buscando una solución que sea segura y además que asegure una calidad constante y admisible, es que la respuesta de robotizar el proceso aparece de forma casi natural.

La robotización de este proceso se ha hecho antes, sin embargo, es una tecnología aún en desarrollo, ya que los defectos en las piezas pueden tener múltiples formas, por lo que llegar a una solución general para su relleno es un problema complejo. Cabe mencionar que, tras robotizar la operación, se seguirá requiriendo de un operador experto, pero reducirá considerablemente los riesgos de la tarea. Finalmente, tener un proceso robotizado abre las puertas para manejar defectos de mayor tamaño que requieran una cantidad de tiempo considerable, lo que permitiría recuperar piezas que actualmente se descartan.

La soldadura robotizada no es un concepto nuevo, y existen distintas soluciones, desde CNC hasta brazos robóticos, no obstante, el desarrollo de estas tecnologías se mantiene en pleno desarrollo. Para los brazos robóticos se desea tener más alternativas en la metodología empleada para el relleno y es en esta tarea que se enmarca el trabajo desarrollado.

Por otro lado, la manufactura aditiva ha explorado la generación de volúmenes, existiendo múltiples alternativas, aun cuando el principal material explorado son polímeros, extender estas estrategias para rellenar con metal es una alternativa, pero estas soluciones se limitan en su mayoría a una estrategia 2.5D, es decir volúmenes generados por una superposición de capas planas. Bajo la intuición de que una deposición de material de forma completamente 3D será una mejor alternativa, es que se explora un camino de herramienta que utilice trayectorias no planares para el relleno, permitiendo seguir las irregulares superficies de los defectos.

El método se basa en que la geometría del cordón de soldadura y la del conjunto de estos puede ser predicha, y utiliza modelos de defectos simuladas por programas CAD o escaneadas de defectos reales. El desarrollo de este es por medio de programación en Python, y la estrategia se realiza considerando que la trayectoria será llevada a un brazo robótico modelo TM 1400 de la marca Panasonic que se caracteriza por especializarse en soldadura. Para generar el camino de soldadura para el relleno del defecto se generarán capas equiespaciadas entre si, que serán recorridas utilizando completamente el eje Z, además de agregar la orientación de la herramienta. La generación de las trayectorias no se realizará para una forma de defecto específico.

1.1 Motivación

Las principales razones de explorar nuevas estrategias para el proceso de recuperación de piezas de acero, es que sea una alternativa cada vez con menos restricciones y con mayor calidad.

La efectividad de la realización de esta tarea se traduce directamente en la disminución de los riesgos a los cuales se ven expuestos los soldadores.

Para la industria, la recuperación de piezas abre la posibilidad de ahorrar el costo de obtener una pieza completamente nueva, ya sean estos de carácter económico o de tiempo, ya que se reducirían los tiempos de traslado al cambiar el envío desde la fábrica de la pieza a el envío desde el lugar en que la pieza es reparada Este punto es clave considerando que a nivel nacional existen pocas alternativas de manufactura para piezas de grandes dimensiones o de geometría compleja.

Este trabajo de título se enmarca además en el programa de manufactura avanzada de la universidad de Chile IMA+, particularmente en el proyecto P03 “Desarrollo de un sistema automático de fabricación y recuperación de piezas metálicas mediante manufactura aditiva”, liderado por los profesores Juan Cristóbal Zagal y Rubén Fernández, que tiene por objetivo la recuperación de piezas de acero utilizando trayectorias generadas de forma automática para así reducir los tiempos de programación del robot por parte del operario y entregando adicionalmente un recorrido de herramienta que realice una recuperación satisfactoria.

1.2 Objetivos

1.2.1 Objetivo general

Generar patrones para soldadura en relleno de defectos utilizando caminos de forma libre, para recuperación de piezas de acero.

1.2.2 Objetivos específicos

Determinar una nueva estrategia de relleno de defectos, no limitado a capas planas.

Generar una estrategia de relleno que no dependa de la forma del defecto a rellenar.

Generar de forma semi automática las trayectorias para ser leídas por el software del brazo robótico.

Analizar el camino de soldadura generado en distintos defectos de interés.

1.3 Alcances

- El trabajo de título es parte del proyecto P03 del programa de manufactura avanzada de la universidad de Chile IMA+, el cual consiste en el desarrollo de un sistema automático de fabricación y recuperación de piezas metálicas mediante manufactura aditiva.
- El proyecto se enmarca en el contexto de recuperación de piezas de acero, y no otro material.
- Si bien el proceso podría desarrollarse en otro tipo de brazo robótico, el trabajo se llevará a cabo considerando un robot especializado para soldadura marca Panasonic modelo TM 1400
- El desarrollo será solo teórico, es decir, no se verá de forma práctica, debido a las limitantes del contexto mundial y restricciones provocadas por la pandemia.
- Los archivos PCD a utilizar pueden provenir de un escaneo de un defecto real, o bien pueden ser producto de simulaciones de estos defectos en un programa CAD.
- Se supondrá que los archivos de las piezas están orientadas según los ejes de coordenada del sistema.
- Se verá el proceso del relleno de un defecto a la vez, lo que significa que no será necesario identificar y separar defectos, en caso de existir múltiples de ellos.

2 Antecedentes y discusión bibliográfica

2.1 Manufactura aditiva

La manufactura aditiva es una forma de manufactura que a diferencia de los métodos tradicionales como maquinado en torno o fresado, consiste en adicionar material para generar una pieza, en vez de sustraerlo[1]. Este cambio en el paradigma de la sustracción de material trajo consigo un mejor uso del material en la fabricación de piezas, al reducir considerablemente el material desechado en la remoción de este. Además, estas estrategias permiten manufacturar piezas de geometrías complejas, personalizables[1], e incluso tener control en propiedades mecánicas de la pieza fabricada al poder decidir el tipo de estructura interna que tendrá esta. Por si lo anterior no fuese suficiente, además permite construir piezas dentro de otras, mecanismos internos.[1]

Esta tecnología ha tomado mayor importancia en la última década debido a la liberación de licencias para la fabricación de impresoras 3D de formato de modelado por deposición fundida (FDM), lo que ha masificado su presencia en el mercado[2]. Estas impresoras utilizan polímeros como su materia prima, principalmente PLA, el cual tiene un bajo costo. Su uso permite tener piezas de geometría compleja de forma considerablemente más rápidas que con los métodos tradicionales, por lo que son perfectas para la fabricación de prototipos. Sin embargo, las propiedades térmicas y mecánicas del PLA no son características fuertes del material.

Por lo anterior, el desarrollo de tecnologías de MA que permitan trabajar con metales es una respuesta natural al problema, para así poder fabricar piezas que soporten altas temperaturas, esfuerzos mecánicos y que aprovechen otras características propias de los metales. Algunas técnicas de MA que utilizan el metal son el depósito de metal basado en laser directo (LBDMD) depósito de metal directo (DMD) y la manufactura aditiva por alambre de soldadura (WAAM). En los últimos años esta última ha recibido especial atención debido a sus altas tazas de manufacturación [3].

Actualmente existen tecnologías que permiten utilizar MA en todo tipo de material, incluyendo cerámicos, como es el caso de impresoras que utilizan cemento, y materiales compuestos como lo son los biomateriales.

2.1.1 Conceptos básicos

Es útil revisar algunos conceptos básicos relacionados a la manufactura aditiva, ya que estos permitirán generar comparaciones y explicar algunas intuiciones propias de este tipo de manufactura.

Impresión 3D: Es la principal técnica de manufactura aditiva. Sus fundamentos son generar piezas a través de la adhesión de material hasta llegar a la geometría deseada, la cual suele ser determinada por un modelo CAD. Las maquinas utilizadas suelen consistir en sistemas de 3 o más ejes de libertad, que permiten adherir material en distintos puntos del espacio. Existen impresoras de distintas tecnologías, siendo las más comunes las tipo FDM que inyectan material a partir de un filamento de este, y las que funcionan por estereolitografía (SLA) las cuales son conocidas por ser el primer acercamiento a este método de manufactura.

Trayectoria o camino de herramienta (Toolpath): Es el conjunto de puntos que describe el desplazamiento de la herramienta.

Camino de forma libre: Se denomina camino de forma libre a las trayectorias que no estén limitadas por un plano.

Planificación de trayectoria (Path planning): La planificación de la trayectoria consiste en resolver ciertos aspectos del camino que seguirá la herramienta teniendo en consideración distintos factores para llegar a un resultado con la menor cantidad de problemas posibles. Estos factores son: la topología del camino; los parámetros del camino; el punto de contacto de la herramienta con la superficie. Es uno de los puntos más importantes en la MA.

Mallado: Se refiere a un método para representar la geometría de un objeto, a través de la división de la superficie en estructuras geométricas más simples, usualmente triángulos. El mallado tendrá información sobre la ubicación de los vértices del triángulo y la orientación de la normal de su cara, así como de los triángulos adyacentes a este.

Offset: El offset puede ser interpretado como una distancia desplazada de un punto en específico. Por ejemplo, en impresoras el offset en Z hace referencia una distancia entre la punta de la herramienta y la bandeja de impresión, la cual al ser modificada puede permitir el uso de una plataforma distinta como un acrílico. En otras aplicaciones, si un contorno es recorrido con un offset, se generará un nuevo contorno de forma similar al previo. Así es como es posible rellenar la superficie generando capas con un cierto offset.

2.1.2 Estrategias de relleno de capas planas

En la manufactura aditiva, la generación de la trayectoria de la herramienta es posiblemente el paso más importante, ya que tiene un impacto directo en la pieza final, determinando propiedades mecánicas, calidad y otros. Una pieza puede variar sus propiedades si el camino de herramienta es generado con una estrategia distinta, aun cuando se utilicen la misma máquina, parámetros y proceso de manufactura aditiva.[1]

A continuación, se mencionan algunas estrategias de relleno para el caso de MA con capas planas.

Raster: Consiste en recorrer la superficie por medio de líneas paralelas, con una dirección determinada.

Zigzag: Similar al raster, pero uniendo las líneas entre sí, saltando al extremo más próximo de la línea adyacente, para poder recorrer todas las líneas de manera continua, aprovechando el ir y venir de la herramienta. [4]. Idealmente se busca la orientación óptima que minimiza el número de capas generadas [5].

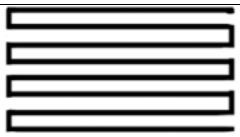
Contorno: Esta técnica es la evolución del recorrido con un offset equidistante del contorno para el caso con geometrías más curvas. Consiste en dividir la superficie utilizando una línea central que marcará el límite entre las capas generadas por el offset de los contornos. [6], [7]

Espiral: Se recorre la superficie siguiendo una espiral euclidiana, es decir que cuenta con una distancia constante entre los ciclos de la espiral [8], [9].

Continuo: Consiste en recorrer la superficie utilizando una línea lo más continua posible. Para ello se divide la superficie en polígonos convexos donde cada polígono será recorrido por una estrategia Zigzag combinada con una de contorno y uniéndose entre sí, generando un camino continuo y cerrado, posteriormente se unen los sub caminos con el camino del polígono adyacente [10], [11] .

Híbrido: Se encarga de combinar los métodos de forma de tener mejores tiempos de impresión, usando caminos de contorno en el exterior y caminos tipo zigzag en el interior del área a rellenar [12], [13] .

Tabla 2.1 Tipos de recorridos y sus ejemplos [14]

Referencias	Tipo de recorrido	Ejemplos
[1]	Ráster	
[4], [5]	Zigzag	
[6], [7]	Contorno	
[8], [9]	Espiral	
[10], [11]	Continuo	
[12], [13]	Híbrido	

2.2 Estrategias de relleno con capas no planares

Las estrategias para la generación de piezas por medio de capas planas en realidad no son puramente 3D, más bien son estrategias 2.5 D ya que se limitan a superponer capas de geometrías en 2D una tras otra, lo que termina por generar superficies escalonadas, un resultado totalmente opuesto a un acabado suave para los casos donde la superficie no es plana [15]. Utilizar capas no planares tiene ventajas tales como tener un resultado manufacturado que se adecúe más a la geometría deseada, tener mejores propiedades mecánicas al tener distintas orientaciones en la estructura, mantener propiedades aerodinámicas de la superficie del modelo, aumentar la adhesión entre capas al tener una mayor superficie de contacto entre ellas, la generación de piezas que requieran menos material de soporte, evitando perder calidad superficial al remover ese material.

Actualmente se han realizado distintas estrategias para lograr construir un modelo por capas no planares. En esta sección se revisarán algunas de ellas:

2.2.1 División multi direccional

Es útil cuando existen estructurar colgantes y reducir el uso de estructuras de soporte. El método fue diseñado para impresoras de 5 ejes, y consiste en generar caminos en múltiples direcciones, por lo que algunas capas serán no horizontales. Lo anterior se logra al utilizar dos módulos distintos, uno que se encarga de descomponer el volumen y reagruparlo. La descomposición se realiza buscando ciclos cóncavos y cerrados en la malla del volumen. Los ciclos contienen solo bordes que también son cóncavos. En bordes cóncavos las caras adyacentes formarán un ángulo menor a 180°. Además, los agujeros en la malla formados al descomponer el objeto se rellenan para ser recorridos después. La topología y posición de cada subvolumen se guarda para el posterior reagrupamiento. El otro módulo se encarga de la división para el recorrido de cada subvolumen. Se elige la mejor dirección para imprimir, y se genera el camino con algoritmos propios de capas planas. Los caminos se rotan y trasladan de vuelta a su posición original y se ordenan según su información topológica. En este proceso no se toma en cuenta las posibles colisiones y no se realizaron pruebas impresas [16], [17].

2.2.2 Impresión de eje Z activo

Con la finalidad de aumentar las propiedades mecánicas, se propone un método que utiliza capas no planares para la fabricación. El uso de capas 2D superpuestas genera una gran anisotropía en la pieza impresa, haciendo que sea mucho más débil cuando se cargan esfuerzos de forma perpendicular a la unión de las capas. El método consiste en dividir la pieza a imprimir en capas con una forma predeterminada por otro archivo STL, utilizando un software llamado *Bread Slicer*. Los parámetros de impresión tales como el espesor de la capa son indicados en un archivo de

configuración. Tras haber generado las capas sobre el objeto a imprimir, se recorren para generar los caminos para la herramienta. Las capas serán recorridas utilizando de manera simultánea los 3 ejes de impresión, razón que le da el nombre al método, siendo un perfecto ejemplo de camino de forma libre con más de 2 ejes. Debido al estado en desarrollo del software se producen problemas con el camino, por lo que se hace un post proceso en Matlab para dejarlo apto para imprimir. Se realizaron distintas impresiones y pruebas, donde se determinó que el uso de capas no planares resulta en piezas más resistentes [17], [18].

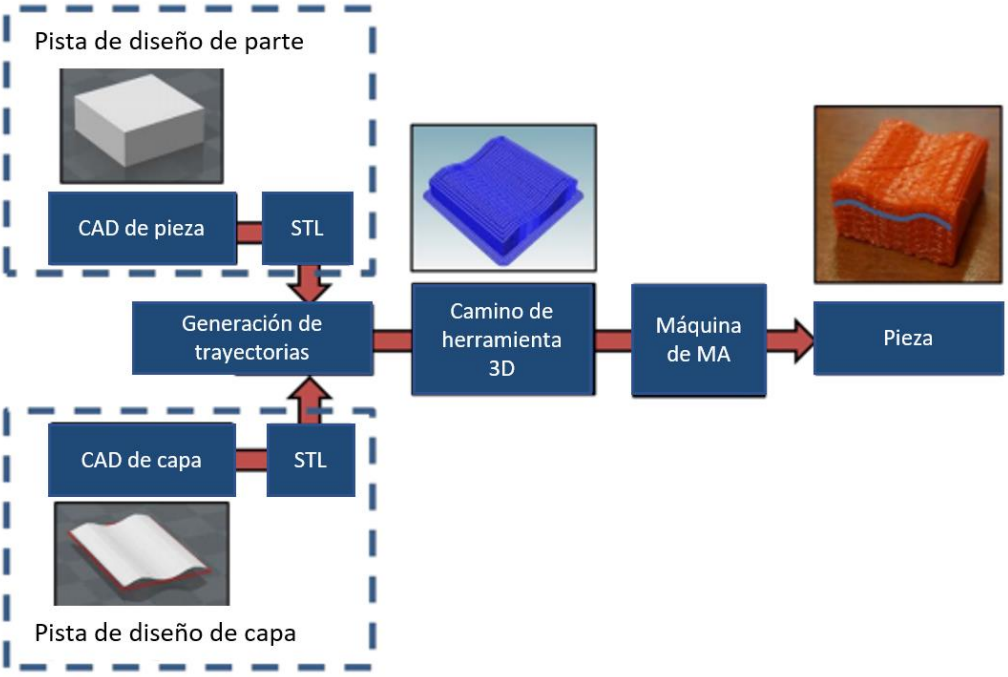


Figura 2.1 Flujo de la metodología de impresión con eje Z activo [18].

2.2.3 Extrusión de material multi eje

Este método fue creado para mejorar las propiedades mecánicas de la pieza. El proceso consiste en imprimir un cascarón reforzado sobre un núcleo impreso de forma regular, con el objetivo de disipar los esfuerzos ejercidos sobre la pieza en el cascarón que se imprime en una dirección distinta a la del núcleo. El camino se genera a con trayectorias que inician en la base a partir de un ángulo dado y se proyectan alrededor del objeto hasta que se alcanza la cima de este. El proceso se lleva a cabo con un brazo robótico de seis ejes. Para generar el código G se utilizan las coordenadas del objeto junto con las normales inversas de la superficie de la malla. El algoritmo solo puede procesar piezas que tengan un perímetro de un solo ciclo cerrado, además se necesita una altura Z definida sin máximos locales para que el algoritmo no se detenga antes de lo necesario. El objeto

no debe ser demasiado cóncavo, para así evitar posibles colisiones. Finalmente se logra tener mejores propiedades mecánicas, siendo el mejor resultado cuando la carcasa tiene direcciones de impresión colineal a la dirección de la fuerza esperada [17], [19].

2.2.4 Generación de caminos completamente tridimensional

La MA por superposición de capas no deja un acabado satisfactorio, debido a los escalones que quedan en la superficie de la pieza, es por ello que para lograr un mejor acabado, se propone colocar capa sobre los escalones de la superficie[15]. Para ello, se genera un camino 3D que puede seguir la superficie de forma libre. Está diseñado para métodos que usan boquilla en impresoras de 3 ejes comunes. El método está inspirado en la planificación de caminos para CNC.

La estrategia se caracteriza por considerar las geometrías de la máquina para evitar colisiones en el camino de la herramienta. Lo anterior se logra generando un camino con un offset invertido a la superficie. La boquilla del extrusor se "invierte" y se barre la superficie con la punta de la herramienta generando un "envoltorio" a la superficie. Con ello, el envoltorio es una superficie libre de colisiones Cuando la cobertura yace sobre la superficie a imprimir en cualquier punto, ese punto no es alcanzable con la boquilla, por lo que esa superficie no es posible de imprimir. Si la diferencia entre la superficie y el envoltorio no supera cierta tolerancia, puede utilizarse el envoltorio para generar el camino de herramienta. El proceso se repite con un nuevo punto de inicio cercano al término del anterior, hasta cubrir toda la superficie de la pieza.

Este método está diseñado para imprimir sobre una superficie dada y no para generar un objeto 3D [15], [17].

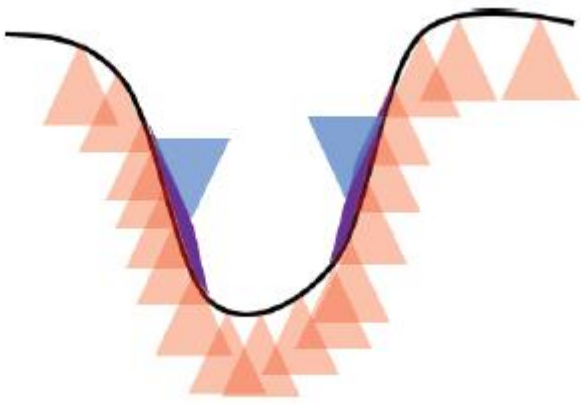


Figura 2.2 geometría de la boquilla de extrusor barriendo la superficie de forma invertida para verificar que el camino está libre de colisiones [15].

2.2.5 Modelado por deposición fundida de capas curvas

También nace con la intención de generar un mejor acabado superficial. El proceso consiste en imprimir planos curvos no planares en diferentes alturas-Z por lo que recibe el nombre modelado por deposición fundida de capas curvas (CLFDM). Se definen 3 factores clave para ver la posibilidad de impresión, un camino de soldadura apropiado, orientación de filamento apropiada y una adhesión de filamento apropiada. Si bien una impresora de 3 ejes cumple con lo necesario para generar estos caminos, una de 5 puede cumplir de mejor manera esta tarea. Se define el camino de herramienta de arriba hacia abajo, utilizando un script de Matlab se obtienen los caminos para la superficie paramétrica. El espesor del objeto se genera con un *offset* de la superficie superior hasta que se alcanza el número deseado de capas. Solo se realizaron visualizaciones de los caminos y no se llevaron a cabo impresiones [17], [20].

2.2.6 Combinación de capas planas y curvas

La unión de las estrategias de capas planas y capas curvas es una evolución esperada de las estrategias de impresión que deseen eliminar la superficie escalonada. Esta combinación fue explorada de dos formas distintas.

Método 1: Se clasifican las superficies según su orientación, entre superior, inferior y lateral, con el objetivo de identificar las áreas que serán impresas con capas curvas. Esta división se genera utilizando el ángulo entre la normal de la superficie y el eje Z, donde el usuario puede ingresar el ángulo límite para cada categoría. Luego se consideran las superficies conectadas como una única superficie continua. Se generan caminos con un offset de la superficie hacia abajo con respecto a las normales de sus caras hasta llegar a la cantidad de cascarones deseados. El método para los caminos es un recorrido tipo *raster*. Para tener capas planas, se restan las capas curvas generadas al modelo original obteniendo un nuevo STL, desde el cual se generarán las capas. El modelo final tarda más tiempo en ser impreso, sin embargo, se registra un mejor acabado superficial y una mejor meso-estructura. [17], [21]

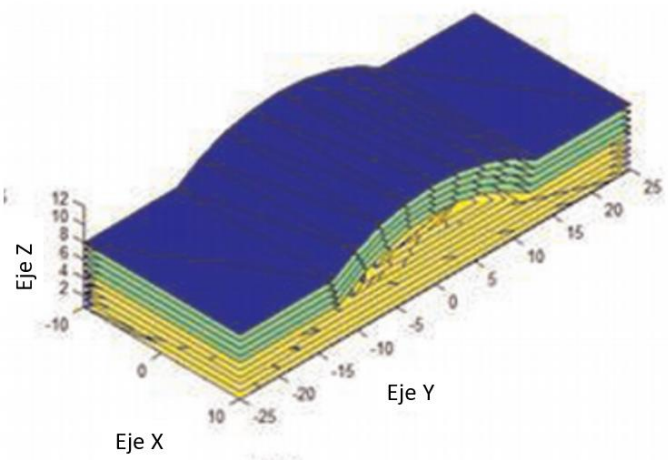


Figura 2.3 Método de combinación de capas planas (amarillo) y curvas (verde). [17], [21]

Método 2: El STL ingresado corresponde a una delgada capa de la superficie del modelo similar a un cascarón, debe ser más delgado que alto de la capa. Luego se repite esa geometría desplazada en el eje z hasta tener el espesor deseado. Esta alternativa está limitada a extrusiones en dirección del eje Z. El camino se genera con un *script* de Matlab, el cual genera una estructura de soporte sobre la cual los escalones pueden ser imprimidos. Para crear el camino del cascarón, primero se imprime el perímetro del borde del cascarón. La estructura de relleno se genera nuevamente rasterizando la superficie como en el método 1. La estructura de soporte se divide como capas planares y se imprime antes que el cascarón. [17], [22]

2.2.7 Planificación de trayectorias para CLFDM

Para tener mejores superficies en capas curvas, se analizó y optimizó la generación del camino. Las superficies de distintas capas son generadas haciendo un offset de los cascarones hacia abajo. El enfoque recae en como ubicar caminos en una superficie de una capa curva de forma apropiada. Para una calidad de superficie óptima, se debe tener una impresora de 5 ejes. El cabezal debe estar perpendicular a la superficie todo el tiempo. En el caso de tener 3 ejes, la altura del cabezal debe variar dependiendo de si se imprime un plano inclinado hacia arriba o hacia abajo para evitar la colisión del cabezal con el material recién extruido. La superficie del STL se transfiere a una B-spline que forma una superficie de referencia sobre la que se genera el camino de herramienta [23].

2.3 Soldadura GMAW/MIG

El nombre significa *Gas Metal Arc Welding* o alternativamente GMAW o MIG, este proceso de soldadura consiste en utilizar un alambre como electrodo, que se empuja a una velocidad controlada hacia la antorcha de soldar. La antorcha de soldar se encarga de derretir el alambre, el cual rellena la superficie a soldar, por lo que el proceso cae en la categoría de consumible. La soldadura es protegida por un gas de baja reactividad, el cual es suministrado de forma externa. El gas logra desplazar el aire en la zona de soldadura, haciendo que las condiciones de estas sean más controladas y evitando la exposición a contaminantes como vapor de agua o nitrógeno.[23]

El proceso de soldadura se compone de múltiples fenómenos físicos que ocurren de forma simultánea, de los cuales es complejo tener control en condiciones distintas a las de laboratorio. Es por ello que en la industria se suele recurrir a la experiencia adquirida con los años, a la hora de determinar el mejor tipo de soldadura según su aplicación, existen guías prácticas que ayudan en este proceso. Aun así, existen parámetros manejables en los cuales se posee control, como lo son el gas a utilizar, corriente, voltaje, diámetro del electrodo, superficie de la pieza a soldar. [23], [24]

El proceso de soldadura consiste en derretir el material base de las piezas e inyectar material de forma externa. Este material en el caso de la soldadura MIG es suministrado por el electrodo consumible. El material termina rellenando o cubriendo el área a soldar, formando antes piscinas de soldadura al mezclarse con el material base derretido.[23]

2.3.1 Tipos de soldadura GMAW/MIG según transferencia

Existen distintos tipos de transferencia de material según los parámetros seleccionados, dentro de los cuales destacan tres: Transferencia por corto circuito; transferencia globular; transferencia por aerosol.

Transferencia por cortocircuito: Ocurre cuando el electrodo entra en contacto con la piscina de soldadura o el metal de la pieza base, lo que produce un cortocircuito en el arco, lo que reduce el voltaje y aumenta la corriente y la fuerza electromagnética, lo que termina desprendiendo la gota de material. La transferencia se produce solo mientras se encuentran en contacto, y el diámetro del electrodo es clave para un resultado satisfactorio. El diámetro del electrodo adecuado variará según el tipo de gas utilizado, sin embargo, el espesor utilizado en el cortocircuito suele estar en el rango de 0.6 y 5mm de material. Las ventajas de este método son que permite múltiples orientaciones, produce una menor deformación por soldadura, no requiere de un operador altamente experimentado, tiene una alta eficiencia, del orden del 93% o mayor. Sus desventajas son que un bajo control en el proceso puede resultar en fusión incompleta, generando defectos, además de aumentar la salpicadura, aumentando los costos de post procesamiento [23].

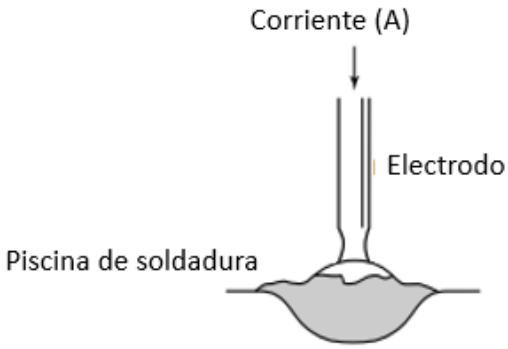


Figura 2.4 Diagrama de transferencia por cortocircuito [23].

Transferencia globular: Se alimenta de forma continua el electrodo y se deposita por largas gotas a través de una combinación de corto circuitos y el efeto de la gravedad. Existe una transición entre el final de la transferencia por cortocircuito y el inicio de la transferencia globular. Se caracteriza por tener una forma de gota irregular mayor al diámetro del electrodo. Estas gotas no siguen una trayectoria axial al electrodo, sino que pueden caer en el camino soldado o dirigirse a un punto de contacto. La forma irregular se debe a fuerzas de "jet" que se dirigen hacia arriba de la zona de trabajo, estas fuerzas son producidas por reacciones químicas y gases liberados en el proceso de soldadura. Sus ventajas es que se puede usar sin problemas con CO2, el cual es más barato que otros gases, es capaz de hacer soldadura a altas velocidades de desplazamiento, los electrodos y equipos de soldadura son de un costo no tan alto. Sus limitaciones se deben a un alto nivel de salpicadura, lo que encarece el proceso de limpieza, la eficiencia se reduce a un rango entre 87 y 93% [23].

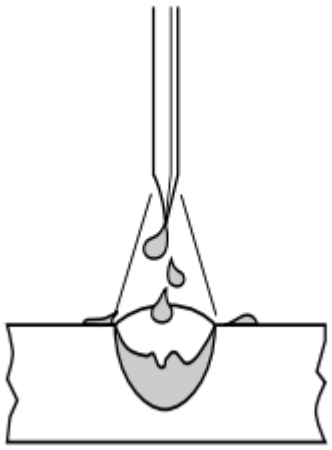


Figura 2.5 Diagrama transferencia globular [23].

Transferencia por aerosol: Es un proceso donde la transferencia de metal se produce a altas energías, donde el electrodo en forma de alambre se transforma en gotitas fundidas que son expulsadas de forma axial a través del arco. Es la transferencia de mayor costo energético y requiere gas con bajo porcentaje de CO_2 y alto de Argón. Las direcciones son restringidas por la alta fluidez de la piscina de soldadura. Su aplicación dependerá del espesor del material base y la capacidad de ubicar la unión de soldadura de manera plana u horizontal. La terminación es excelente y se requiere de un operador altamente especializado. El resultado es aún mejor si la superficie está libre de contaminantes como aceites, polvo u óxido. Las ventajas son tener una alta tasa de depósito, una eficiencia de electrodo de 98% o mayor, poder ser utilizado en un amplio rango de metales y diámetros, terminación de soldadura excelente, no existe salpicadura. Sus limitaciones son que se restringe a superficies planas y horizontales, genera una mayor cantidad de gases de soldadura, irradia una mayor cantidad de calor y luz, lo que requiere una mayor protección [23].

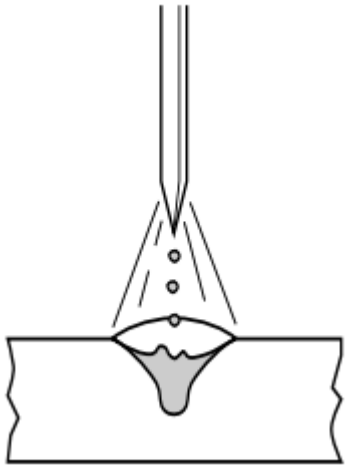


Figura 2.6 Diagrama de transferencia por aerosol [23].

2.3.2 Geometría del cordón de soldadura

El perfil del cordón de soldadura junto con la penetración en el sustrato son determinados por múltiples variables dentro de las que destacan: corriente de soldadura, voltaje del arco de soldadura, ángulo de la antorcha, distancia de contacto de la punta al sustrato (CTWD por sus siglas en inglés) y la velocidad de desplazamiento de la antorcha. La corriente de soldadura se relaciona a su vez de forma directa con la velocidad de alimentación del cable [24]. En la Figura 2.7 puede apreciarse un diagrama que muestra las principales influencias de cada parámetro en el proceso de soldadura.

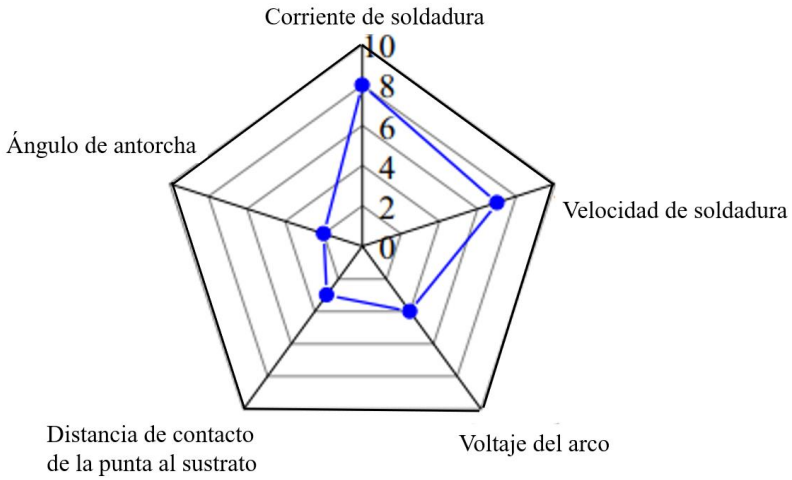


Figura 2.7 Influencias de los parámetros de soldadura en la geometría del cordón [24]

La geometría del cordón de soldadura ha sido descrita por distintos modelos, dentro de las cuales predominan el arco, la función coseno y la parábola. La parábola es la una aproximación aceptada y ha sido comprobada de forma empírica comparándola con secciones transversales del cordón de soldadura. Establecer que existe esta relación permite conocer la geometría del cordón solo conociendo la altura de esta, evitando el procedimiento antiguo que consistía en seccionar la pieza, pintar los puntos representativos de su geometría y escanear el cordón. Además, se establece una relación entre las velocidades de alimentación del cable de soldadura y la velocidad de la antorcha con el ancho con una regresión lineal entre los datos empíricos [25].

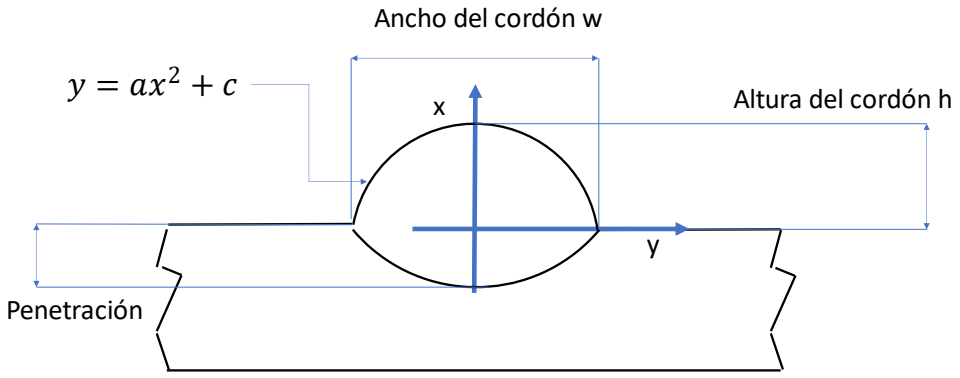


Figura 2.8 Geometría del cordón de soldadura con el modelo parabólico[26].

Al comparar las desviaciones del modelo con el resultado de la soldadura, se tiene que la parábola presenta una mejor coincidencia con sección transversal de la soldadura cuando la relación entre la velocidad de alimentación del electrodo y la velocidad de soldadura de la herramienta es menor a 12.5, por otro lado, para relaciones mayores, el arco logra un mejor ajuste [26].

Para describir la geometría de un conjunto de cordones de soldadura, se utiliza como base la geometría para el caso singular. En un principio no fue considerado el espacio de sobreposición de los cordones, por lo que el resultado distó de lo esperado, sin embargo, al incluir el volumen sobrepuesto de los cordones se llegó a una mejor aproximación. Con la geometría de un conjunto de cordones de soldadura ya definidos, se llega a que existen 3 formas predominantes que dependen de la distancia d entre los centros de los cordones, uno donde la superficie entre cordones es cóncava, otro convexo, y finalmente un caso donde esta es plana. Este último caso se da en el caso crítico donde el volumen de sobreposición es equivalente al volumen del valle que queda entre las 2 parábolas. Esta distancia que genera la superficie plana se denominará distancia crítica, representada por d*. Así se describirá que el caso convexo se da cuando el volumen sobrepuesto supera al del balle entre cordones, y d<d*, el caso cóncavo se da cuando el volumen sobrepuesto es menor al del balle entre cordones y se tiene d>d*.

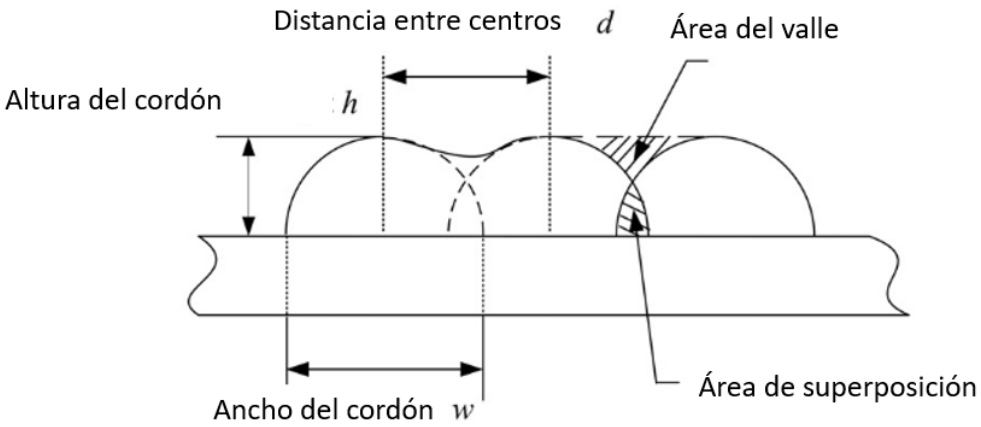


Figura 2.9 Superposición de cordones de soldadura.

La ventaja de tener modelos para la geometría del cordón es que esta puede ser predicha solo conociendo el alto del cordón de soldadura. Con ello y a partir de datos experimentales de anchos y altos de cordón para distintos valores de velocidades de antorcha y alimentación de electrodo, puede determinarse la geometría solo conociendo estos parámetros de soldadura [26].

Cabe mencionar, que los datos experimentales que relacionan las velocidades de antorcha y alimentación de electrodo con el alto del cordón serán propios de cada tipo de electrodo a utilizar, además que variará también con el tipo de gas a utilizar.

Debido a que se suele desear una taza de inyección de calor mínima y una distribución de calor más amplia, una mayor resolución, se recomienda utilizar la menor velocidad de cable de soldadura posible con la mayor velocidad de antorcha posible, combinada con un incremento de dos tercios del ancho del cordón de soldadura entre cada cordón [3].

Si bien la distancia entre cordones de soldadura ubicados de forma paralela ha sido determinada, también se ha estudiado el caso en que la superposición de cordones se realiza de forma perpendicular entre ellos, lo cual resulta bastante útil en estrategias que utilizan un recorrido tipo híbrido. Debido a que los cordones de los bordes tendrán además solo un cordón en su borde, tiene sentido que la distancia d entre ellos varíe. Con ello se definen d1 y d2 con los valores de 0.47w y 0.5 w respectivamente y l como 0.738 como indica la Figura 2.10 [12].

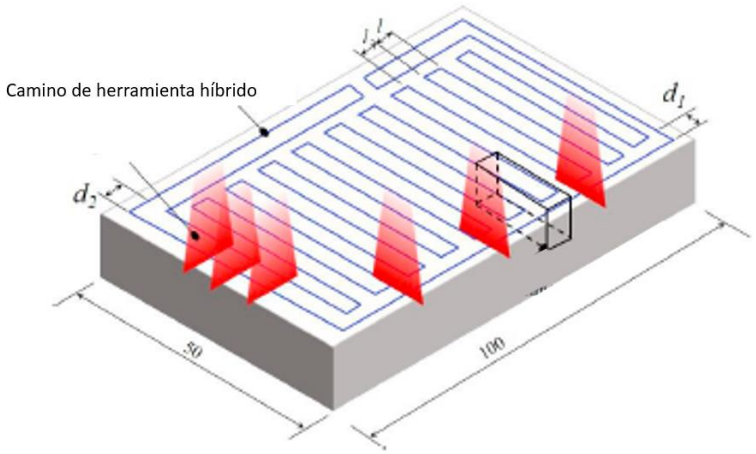


Figura 2.10 Diagrama de un recorrido híbrido con distintas distancias de cordones en los bordes [24].

2.4 Soldadura GMAW/MIG en manufactura aditiva

Las aplicaciones que puede tener este tipo de soldadura se enmarcan principalmente en 3, generación de piezas construidas completamente por medio de capas superpuestas de cordones de soldadura, el revestimiento de metales y finalmente la recuperación de piezas rellenando sus defectos.

La masificación de su uso para este tipo de aplicaciones se debe a su alta eficiencia de deposición, bajo costo, operación automática y excelentes propiedades mecánicas [24]. A lo

anterior también se suma que tiene una mejor homogeneidad, y una mayor densidad de metal depositado[27].

Cuando se utiliza la soldadura tipo GMAW en manufactura aditiva, se debe verificar que el camino de herramienta cubra completamente la superficie para poder tener un resultado óptimo. En segundo lugar se debe mantener la continuidad del camino para evitar una constante alternación entre encendido y apagado en el sistema de alimentación del alambre de soldadura. Finalmente, una precisión en los bordes de la geometría es deseada para minimizar o eliminar un maquinado posterior al proceso de soldadura. [28].

En la manufactura aditiva con GMAW, también es crucial tener un buen modelamiento de los cordones de soldadura, para así poder elegir de forma precisa la cantidad de capas necesarias para poder generar la pieza deseada [28].

Con lo anterior, se ha llegado a resultados satisfactorios en la generación de piezas, generando caminos de soldadura donde se varían los parámetros de soldadura para poder tener distintos espesores de cordón de soldadura, técnica llamada algoritmo de transformación de eje medio adaptativo (MAT por sus siglas en inglés). Este método logra tener una alta calidad de deposición de material, es decir, sin vacíos entre el material, y una precisión geométrica alta con un proceso automatizado [28] .

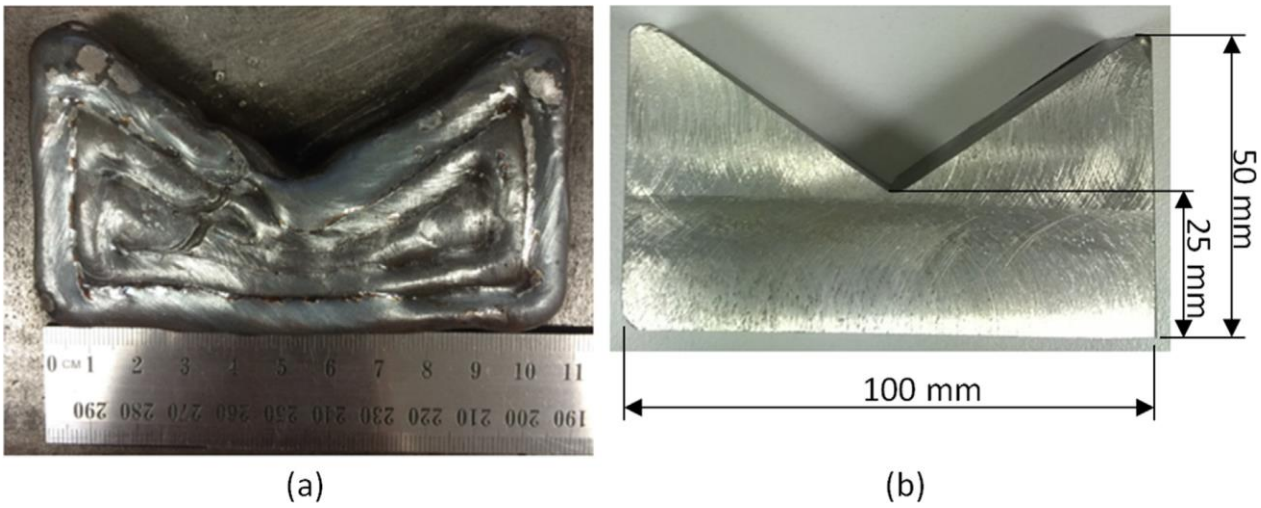


Figura 2.11 (a) Pieza generada con 15 capas. (b) Pieza terminada sin vacíos después de maquinar la superficie [28].

2.5 Tipos de piezas y defectos

En la industria existen todo tipo de piezas de metal que pueden presentar defectos por distintas razones, como por ejemplo corrosión, abrasión o condiciones de servicio de un estrés mecánico alto. Estos defectos pueden tener formas impredecibles e irregulares. [27].

3 Metodología

Debido a que el objetivo es proponer una nueva estrategia para el relleno de defectos, existe una etapa iterativa donde se desarrollan distintas ideas hasta llegar a una estrategia final para resolver el problema. Los problemas y desafíos que surgieron en el desarrollo de esta solución son descritos en la sección Definición de estrategia de relleno. Tras haber pasado por ese proceso, se llega que el método para resolver será:

1. Parámetros e información inicial
   1.1. Se cargan los archivos con la información de la falla
   1.2. Se ingresan los parámetros del cordón de soldadura
   1.3. Se ingresa la información de la ubicación de la pieza
   1.4. Se indica el tipo de estrategia a utilizar, espiral o ráster zigzag
2. Preprocesamiento
   2.1. Se reduce la nube de puntos según el tamaño de vóxel deseado
   2.2. Se aísla la geometría del defecto del resto de la pieza
3. Generación de capas no planares
   3.1. Se calcula la cantidad de capas necesarias para cubrir el defecto
   3.2. Se copian los puntos y se trasladan según su dirección normal asociada
   3.3. Se eliminan los puntos entre capas y se recalculan las normales
   3.4. Se repite el proceso hasta alcanzar el número de capas calculado
   3.5. Se eliminan los puntos de las capas generadas que se encuentren fuera del volumen a rellenar
4. Recorrido de capas
   4.1. Según la estrategia seleccionada, se interseca la superficie con un conjunto de nubes de puntos con la forma deseada del recorrido, resultando una división de la superficie original en múltiples nubes de puntos.
   4.2. Se identifica un punto inicial en la nube de puntos generada
   4.3. Se avanza en la dirección conveniente del recorrido hasta llegar a un extremo donde no se encuentran vecinos en un radio determinado
   4.4. Se eliminan los puntos cercanos a los puntos ya recorridos y se repite el paso 4.2 para la nube de puntos siguiente, si no se encuentran puntos, se avanza a la siguiente nube de puntos generada.
   4.5. Al terminar, se repite el proceso para la capa siguiente
5. Post procesamiento

5.1. Se identifican los puntos iniciales de cada recorrido, indicando si corresponde o no encender la antorcha
5.2. Se asocia un código de color a cada punto según la acción que corresponda
5.3. Se unen las nubes de puntos en el orden a recorrer

6. Visualización del camino de soldadura
6.1. Se recoge la información de los archivos PCD y se grafica con la librería matplotlib
6.2. Similar al paso anterior, se genera una animación del recorrido, para tener noción del orden de recorrido de los puntos
6.3. Se grafica con la librería open3D, para verificar la orientación de las normales de la trayectoria

7. Traducción al lenguaje del brazo robótico
7.1. Se toma la información de las coordenadas y normales de los puntos a recorrer de la unión de las nubes de puntos.
7.2. Se escribe utilizando el formato AV

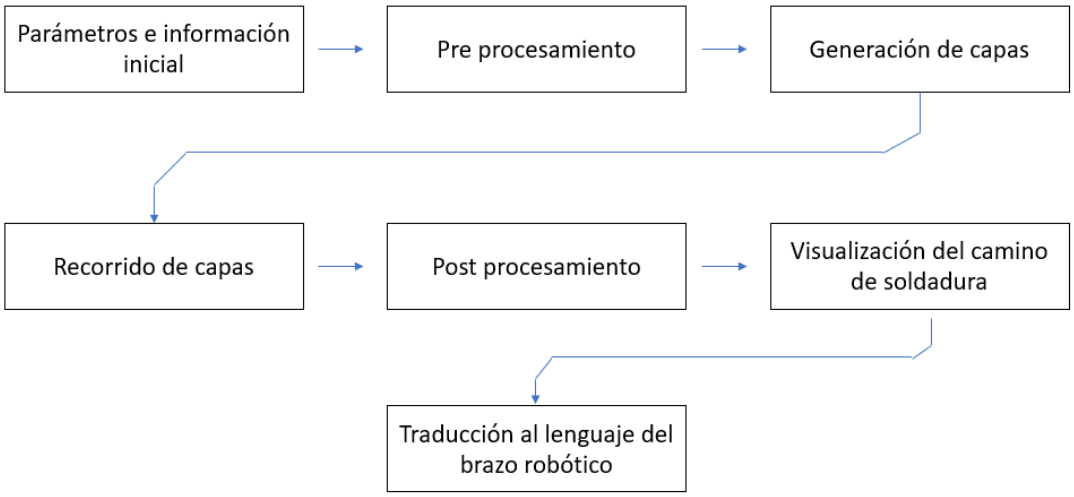


Figura 3.1 Diagrama de la estrategia final

3.1 Recursos

Para poder llevar a cabo el trabajo se necesitaron distintos recursos, los cuales van desde el lenguaje de programación utilizado hasta el software de manejo del brazo robótico. En esta sección se describirá cada uno de los recursos más relevantes.

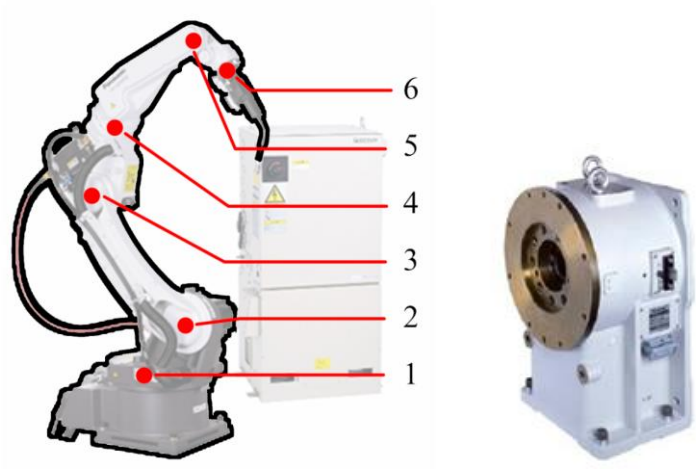
3.1.1 Brazo robótico

Existen distintas alternativas en el mercado de brazos robóticos, dentro de los cuales destacan Panasonic, Kuka, Yaskawa, entre otros. Los distintos modelos se diferencian entre si por su cantidad de ejes libres, dimensiones y además por poseer o no alguna especialización en alguna tarea.

Los robots Panasonic cuentan con una línea especializada para soldadura, los cuales pertenecen a la serie TM y cuentan con distintos largos dependiendo de su modelo, además de poseer seis ejes de movimiento que permiten movimientos con alta libertad y maniobrabilidad. En la Figura 3.2 pueden observarse el brazo robótico junto con la unidad que contiene los elementos propios del control de la soldadura.

Figura 3.2 Robot Panasonic de la serie TM [29].

Las articulaciones se indican en la Figura 3.3 ,en específico son: Rotación de la base (1); brazo (2); flexión antebrazo (3); giro del antebrazo (4); muñeca de flexión (5); giro de muñeca (6).



(a) (b)

Figura 3.3 (a) Ejes de rotación del brazo robótico y (b) Posicionador externo

El número de ejes además puede aumentarse utilizando posicionadores, en este caso se muestra el posicionador YA/1RJB32 de 1000 kg [30]

El robot cuenta con un software especializado para poder realizar su programación, que además permite hacer una simulación del recorrido de este.

3.1.2 Python

El lenguaje de programación Python ha ganado gran popularidad en el último tiempo debido a su gran versatilidad y ser un lenguaje pensado en ser amigable con el usuario al ser orientada a objetos, es decir que permite al usuario generar nuevos tipos de datos para resolver los problemas. Además, cuenta con una comunidad activa que constantemente actualiza paquetes y librerías que son utilizadas en distintas áreas, dentro de las que se destaca la ciencia y tecnología [31].

Dentro de las librerías destacan Matplotlib, que permite la visualización de datos de forma similar al programa Matlab; Open3D, que permite procesar y trabajar con archivos 3D.

3.1.3 Cloud Compare

Cloud Compare es un software de procesamiento de nubes de puntos 3D y de mallas triangulares. Fue creado originalmente para la comparación entre nubes de puntos densas, tales

como las que se obtienen con un escáner, o la comparación entre una nube de puntos y un modelo de mallas triangulares [32].

Este software permite convertir un modelo de la pieza desde STL a PCD, indicando la cantidad de puntos de la nube, o una densidad de puntos deseada.

3.1.4 Archivos PCD

Los archivos PCD (Point cloud data) contienen una nube de puntos, que permite representar formas en el espacio. Estos archivos serán utilizados para representar las fallas para luego ser procesadas y generar los caminos de soldadura. [10]. Se componen por atributos que indican sus coordenadas por medio de listas, las normales respectivas a cada punto, y además puede indicarse un color a cada punto.

3.1.5 Open3D

Esta librería permite operar con distintos objetos como nube de puntos, mallas, y otras entidades 3D [33]. Además, contiene distintas utilidades para el manejo de nubes de puntos, tales como la estimación de normales, la búsqueda de vecinos del punto, ya sea por medio de un radio o por la cantidad de puntos más cercanos. Sumado a lo anterior, también cuenta con funciones para trasladar, escalar, o rotar las geometrías en caso de ser necesario.

3.1.6 Software del brazo robótico

El programa del brazo robótico consiste en dos utilidades principales, “PC Tool” y “DTPS”. La primera se encarga de agregar las especificaciones del robot, tales como modelo utilizado, ejes adicionales, y distintas características del espacio de trabajo. Por otro lado, “DTPS” permite simular el proceso de soldadura [33].

El programa DTPS (Desk-Top Programming & Simulation System) es un software especializado para el controlador G2/G3 de los robots de marca Panasonic que permite la simulación del proceso de soldadura, entregando información vital para conocer los movimientos específicos que realizará el robot, la trayectoria seguida, la ubicación de las piezas, el tiempo que tardará el proceso, así como facilitar la edición de puntos o comandos. Este programa funciona leyendo archivos en formato CSR, los cuales traen información acerca del camino de soldadura y los comandos necesarios para llevarla a cabo [33].

En el programa además se indican los distintos elementos del lugar de trabajo, tales como la existencia o no de ejes externos, además se indica la ubicación de estos elementos. Para cada elemento se asocia también un eje de coordenadas, donde el brazo robótico tiene la particularidad de poseer dos, uno para su base y otro para la punta de la herramienta. Dependiendo del formato utilizado, puede ser posible describir trayectorias referenciando estos ejes de coordenadas. En la Figura 3.4 puede apreciarse los elementos y los ejes de coordenadas a asociados a ellos.

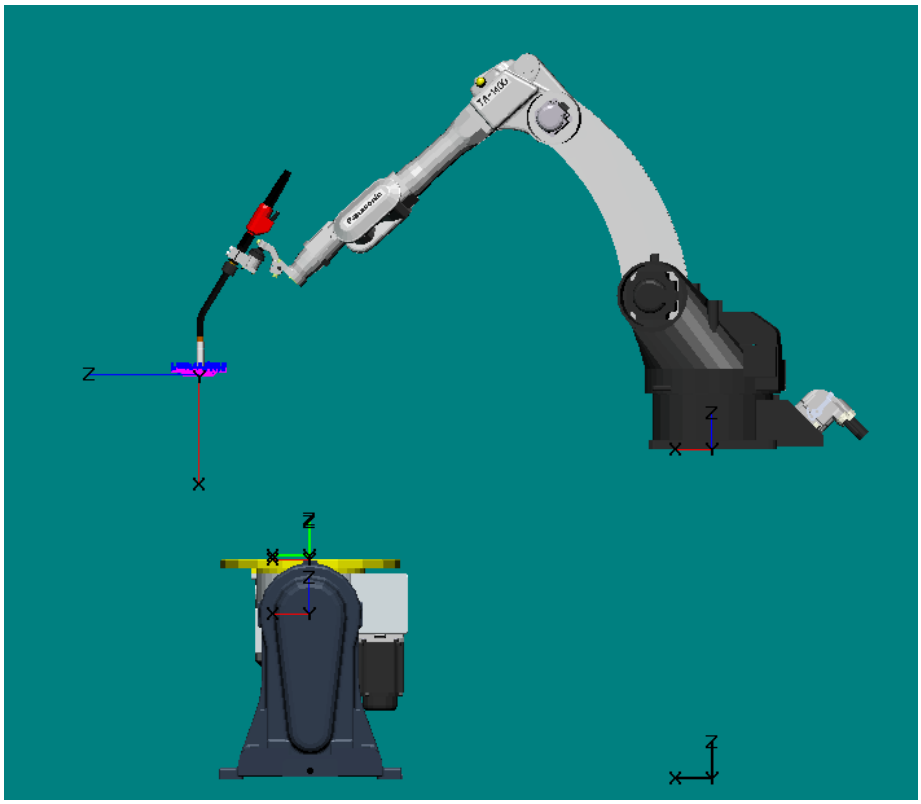


Figura 3.4 Disposición de elementos en el programa DTPS.

3.1.7 Archivos CSR

Los archivos CSR describen las especificaciones de los caminos de soldadura, así como los parámetros utilizados para recorrerlos. El archivo en cuestión recibe información del camino de soldadura por secciones, donde primero se indican la descripción de la parte, las coordenadas a recorrer, las variables locales de la pieza y operaciones que se tendrá en el recorrido y finalmente una serie de comandos que describe las acciones a realizar, como encender o apagar la antorcha,

reposicionar el robot, o indicar el método de desplazamiento que se tendrá de una coordenada a otra, así como la velocidad de la antorcha o del cable de soldadura.

La información sobre las coordenadas podrá darse en distintos formatos, los cuales permiten describirla según el sistema de referencia que sea más cómodo para el usuario.

Tabla 3.1 Distintos formatos de escritura para las coordenadas a recorrer [33]

Formato	Contenidos de los datos			Unidad
AJ (AJI)	Ángulos de articulación	1 a 6	Ángulo de la articulación de RT a TW	Grados
		7 a 27	Ángulos de las articulaciones de ejes externos G1 a G21	Grados o mm
AV (AVI)	Posición + Vector	1 a 3	Posición (X, Y ,Z)	mm
		4 a 6	Vector dirección de herramienta (X, Y, Z)	mm
		7 a 9	Vector orientación del giro herramienta (X, Y, Z)	mm
		10 a 30	Ángulos de las articulaciones de ejes externos G1 a G21	Grados o mm
AU (AUI)	Posición + UVW	1 a 3	Posición (X, Y, Z)	mm
		4 a 6	Orientación de la herramienta (U, V, W)	Grados
		7 a 27	Ángulos de las articulaciones de ejes externos G1 a G21	Grados o mm
AP (API)	Pulso	1 a 6	Valor del pulso de RT a TW	-
		7 a 27	Valor del pulso de los ejes externos G1 a G21	-

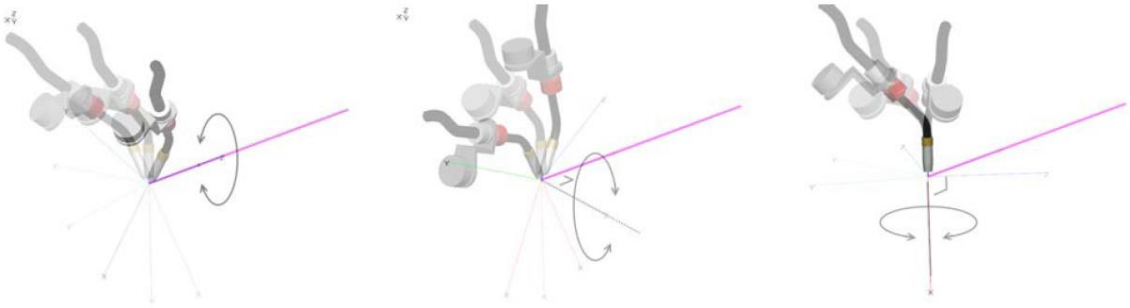
El formato AV permite indicar de forma directa la dirección de la herramienta, sin embargo, requiere indicar también la orientación de la muñeca del robot.

Dentro de la lista de comandos, indicados en la Tabla 3.2 destacan MOVEP, MOVEC y MOVEL, donde el primero describe que los puntos se recorrerán de un punto a otro sin describir el movimiento de la herramienta, dejando en libertad que el programa se traslade entre ellos como mejor lo estime conveniente; el segundo recorrerá los puntos con una trayectoria curva siguiendo un arco descrito por 3 puntos; finalmente MOVEL recorrerá desde un punto a otro con la herramienta siguiendo una línea recta entre ellos [33].

Tabla 3.2 Lista de comandos de movimiento [33].

Comando	Descripción
MOVEC	Movimiento en arco
MOVECW	Movimiento en arco, con oscilación
MOVEL	Movimiento en línea
MOVELW	Movimiento en línea, con oscilación
MOVEP	Movimiento punto a punto
WEAVEP	Movimiento punto a punto, con oscilación

Utilizando los formatos indicados en la Tabla 3.1 es posible configurar la antorcha para rotaciones en distintos ejes. En la Figura 3.5 la inclinación es la rotación en torno a la línea de soldadura, la flexión, es la rotación de la antorcha en dirección de la línea de soldadura y finalmente el giro es la rotación alrededor del cabezal de gas.



Inclinación Flexión Giro

Figura 3.5 Movimientos de la antorcha [15].

3.2 Descripción de las etapas

3.2.1 Parámetros e información inicial

Para proceder en la estrategia, se requiere partir de ciertos parámetros que serán dados por un usuario. Estos parámetros son:

Archivo PCD con la zona del defecto a rellenar

Amperaje a utilizar

Velocidad de desplazamiento de la antorcha

Ubicación de la pieza respecto al brazo robótico en coordenadas cartesianas

Punto de observación del defecto de la pieza

Tamaño de voxel

Con esta información es posible determinar la geometría del cordón de soldadura, así como la del conjunto de ellos gracias a los modelos de ella y los datos experimentales de anchos para distintas velocidades de alimentación de electrodo y antorcha.

En particular se utilizarán parámetros de geometría de cordón correspondientes a ensayos empíricos, asociados a un tipo de alambre electrodo específico, de características que se indican en la Tabla 3.3 y la Tabla 3.4.

Tabla 3.3 Características del alambre de soldadura

Alambre de soldadura	
Material	Acero AWS- ER70S-6
Diámetro	0,9 [mm]

Tabla 3.4 Características Gas de soldadura

Gas de soldadura	
Producto	INDURMIG-MAG
Composición	80% Ar , 20% CO_2

3.2.2 Preprocesamiento

El preprocesamiento tiene como objetivo preparar la nube de puntos para generar los caminos.

La primera operación será disminuir la cantidad de puntos de la nube dejando solo una cantidad suficiente para representar la geometría de la pieza. Tener una nube de puntos con demasiados puntos podría aumentar considerablemente el tiempo de procesamiento del programa. Lo anterior se logra fácilmente con la función de open3d *voxel_down_sample* , donde se deja una muestra de los puntos equiespaciada según el tamaño de los voxeles.

En el caso de recibir una nube de puntos que carece de normales, estas pueden ser estimadas con la función *estimate_normals* de open3d.

En esta etapa es posible aislar la zona con el defecto realizando una intersección entre la nube de puntos de la pieza sin defecto con la pieza con defecto.

3.2.3 Generación de capas no planares

La generación de capas se inspira en el relleno de superficies 2D con la técnica de contornos paralelos. Con ello se procede a replicar lo anterior, pero en dirección normal a la superficie, lo que genera capas que representarán la nueva superficie tras ser rellenada con soldadura la capa previa.

Se generarán tantas capas como sea necesario para rellenar el defecto. Esta cantidad n puede ser determinada dividiendo la diferencia entre el punto más profundo del defecto respecto a la superficie por la altura de la capa de un cordón de soldadura h para los parámetros indicados por el usuario.

La capa se produce al desplazar cada punto de la nube de puntos original una distancia h en la dirección de la normal estimada para el punto.

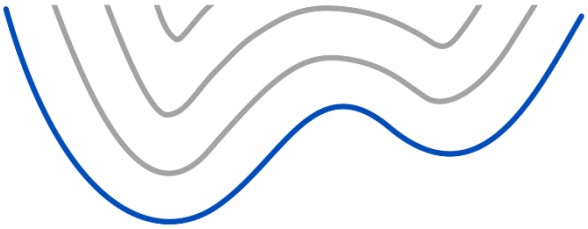


Figura 3.6 Diagrama de las trayectorias generadas al desplazar los puntos en su dirección normal.

Posteriormente se procesa esta capa para eliminar puntos que queden “ocultos” bajo la superficie respecto al punto de observación del defecto. Lo anterior genera que la capa no incluya puntos que estarían “insertos” en la capa de soldadura previa.

El proceso se repite sobre la capa generada para crear una nueva capa, hasta haber iterado n veces.

Finalmente, es posible eliminar los puntos de las capas que se encuentren sobre el límite del volumen de relleno. Esto se logra dejando solo los puntos que estén dentro del bounding box de la falla aislada. Análogamente, es posible cortar la nube de puntos utilizando una superficie no planar si el escenario así lo requiere. Lo anterior se logra generando un offset 3D y restando los puntos superpuestos, esta vez con la superficie de la pieza real, y luego restando la intersección entre las nubes de puntos.

3.2.4 Recorrido de capas

Para el recorrido de capas se utilizarán dos estrategias distintas la primera consiste en el método ráster zigzag, y el segundo en espiral. En ambos casos se realizará previamente una operación para simplificar la capa a recorrer, dividiéndola en nubes de puntos con la forma del camino deseada.

El camino de herramienta consistirá en vectores que contengan la información de las coordenadas de los puntos a recorrer, además de las normales de la superficie. Debido a que esta información es la que contienen las nubes de puntos, se decide almacenar esta información en una, lo que posteriormente facilitará el visualizar el recorrido y la orientación de ellos.

El proceso para llegar al recorrido deseado tiene dos etapas para cada tipo de estrategia, en la primera etapa se secciona la capa a recorrer en nubes de puntos que mantengan la forma de la trayectoria deseada. La función utilizada para esta acción es preslice01 para el ráster zigzag o preslice02 para espiral. Posteriormente se utiliza una función para extraer los puntos representativos de esas nubes, los cuales indicarán las coordenadas de las trayectorias. Si bien las funciones son en esencialmente similares, tienen pequeñas diferencias entre si.

Seccionamiento de la nube: Este paso ayuda a solucionar el problema de no tener ordenadas las coordenadas de los puntos de la nube. Se realiza una intersección con una nube de puntos conocida. Para el caso zigzag la nube de puntos a intersecar será un conjunto de planos equiespaciados entre ellos. En el caso del recorrido espiral, se interseca con un conjunto de ciclos de una espiral euclidiana, ordenados desde dentro hacia afuera. Al final de este paso, las nubes de puntos tendrán una forma similar a una línea, ya sea recta o curva.

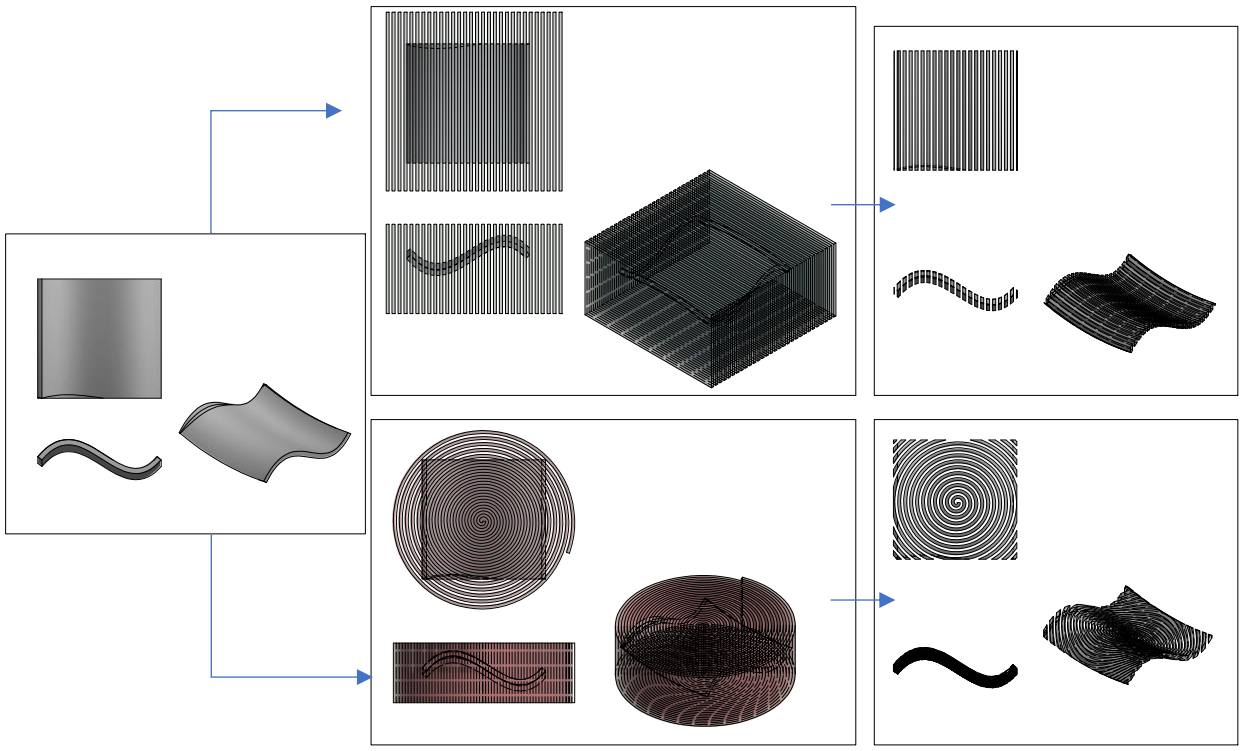


Figura 3.7 Representación del procedimiento de división de la nube de puntos previo a la generación de trayectorias.

Recorrido de la nube: El algoritmo para recorrer la nube consiste en seleccionar un punto inicial en un extremo de la nube generada en el seccionamiento, y avanzar a través de ella con una distancia específica, en este caso se utilizará el ancho del cordón de soldadura.

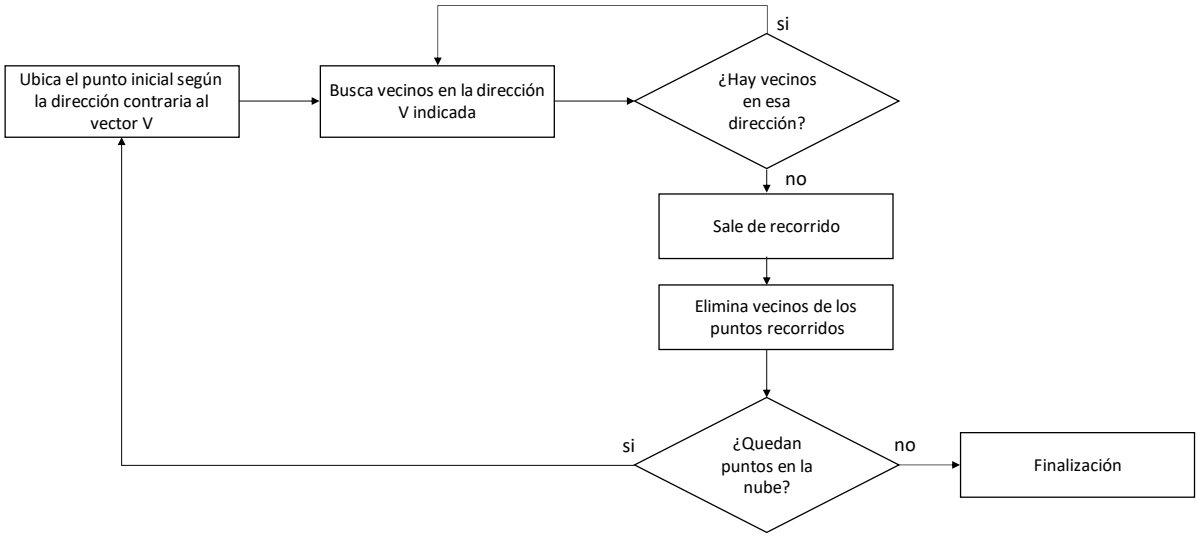


Figura 3.8 Diagrama del algoritmo utilizado para recorrer las sub nubes de puntos.

3.2.5 Post procesamiento

En este punto se opera el camino para poder tener posteriormente una mejor visualización y traducción al formato CSR.

En esta etapa se añaden puntos a la trayectoria que indiquen que la herramienta se levanta para desplazarse de un punto a otro cuando sea necesario. Los puntos en de inicio y término pueden ser fácilmente identificados ya que cada trayectoria se encuentra en una nube de puntos distinta.

Con ello, se sumarán puntos para levantar la herramienta donde la distancia entre el punto final de una nube de puntos y el punto inicial de la siguiente tengan una distancia superior a 2 veces la distancia entre cordones.

Se utiliza un código de color para representar la trayectoria, para así dar mayor información en la visualización y además utilizar esa información para convertir la nube de puntos al formato CSR.

Tabla 3.5 Descripción del código de colores utilizado en la nube de puntos.

Color	Descripción
Azul	Punto con soldadura
Rojo	Punto sin soldadura
Verde	Punto de inicio de soldadura

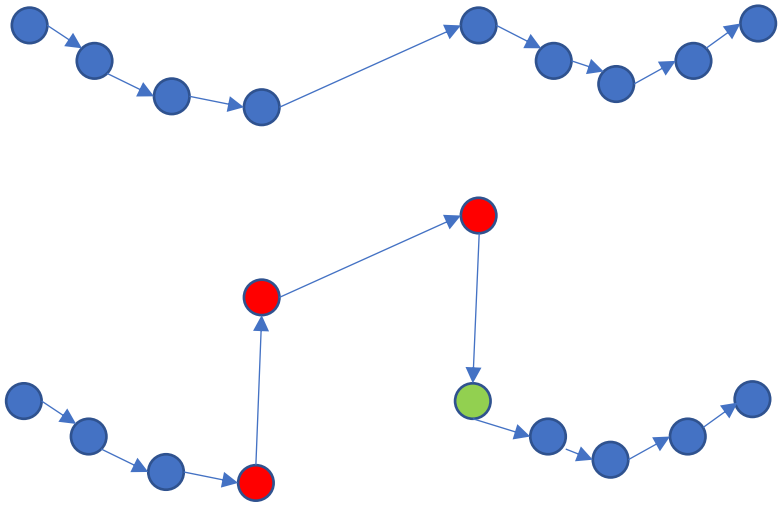


Figura 3.9 Comparación de los recorridos antes y después de añadir los puntos para levantamiento de la herramienta.

3.2.6 Visualización del camino de soldadura

La visualización del camino de soldadura tiene una gran relevancia ya que permite al usuario verificar si las trayectorias propuestas cumplen el objetivo deseado sin mayores complicaciones. Así mismo, la ubicación de los puntos de la trayectoria no es la única variable relevante, ya que también importan las normales de dichos puntos, la trayectoria que se sigue entre ellos y también el orden en que estos caminos son recorridos. Para verificar el orden en que son recorridos de forma rápida, es útil realizar una animación de la trayectoria, la cual dará una idea al usuario del comportamiento del brazo robótico antes de agregar el archivo generado al programa DTPS.

Las normales pueden ser visualizadas en el entorno de Open3D, mientras que la animación y el gráfico de los puntos se generan utilizando la librería matplotlib.

Para las visualizaciones en matplotlib, se extraen los puntos de la nube de puntos y se ingresan como listas.

3.2.7 Traducción al lenguaje del brazo robótico

Para realizar la traducción al lenguaje del brazo robótico, basta con generar un archivo CSR que incluya la información de las trayectorias deseadas.

Debido a que el método propuesto requiere que la herramienta se encuentre siempre perpendicular a la superficie, el formato de coordenadas más convenientes es el AV, ya que puede insertarse directamente el vector que indica la dirección de la herramienta.

Este vector X será igual al inverso del vector normal de la superficie de la capa, mientras que el vector Z que indica el giro de la muñeca del brazo robótico (eje 6 de la Figura 3.3) se mantendrá siempre orientado en la dirección del eje X global.

En esta sección se lee además el código de colores asociado, el cual incluirá los comandos de soldadura correspondientes al inicio o detención de la soldadura al identificar un salto en el camino de herramienta.

4 Desarrollo

4.1 Casos de estudio

La estrategia se diseña para poder solucionar 2 escenarios distintos, uno donde se cuenta con el modelo original de la pieza y el modelo con el defecto, y otro donde solo se tiene la información del defecto.

Además, se harán pruebas con 3 fallas distintas:

- Placa con defecto generado por una revolución de una curva
- Placa que presenta un defecto con superficie en forma de punto silla.
- Modelo generado por un escaneo de una pieza real, consistente en una placa.

Para cada falla, se recorre las capas generadas con 2 estrategias distintas, una con el método zigzag y la otra tipo espiral.

4.2 Definición de estrategia de relleno

El modo de relleno elegido consiste en generar capas sobre la superficie de la falla, que se caracterizan por ser desplazamientos de los puntos de la superficie en la dirección de su normal correspondiente. Las capas se generarán hasta poder rellenar el volumen deseado y llegar a la geometría original de la pieza.

Debido a que cada capa se genera siguiendo la forma de la falla, estas tendrán en la mayoría de los casos una forma no planar, por lo que para ser rellenadas deberán seguirse por medio de caminos de soldadura de forma libre.

Se considerarán los siguientes supuestos:

- La geometría del cordón de soldadura se mantiene de forma parabólica, aun cuando la normal de la superficie tiene una orientación distinta al eje Z+. Es decir, se considerará la gravedad como un factor no determinante.
- La orientación de la pieza y su ubicación en un lugar donde el robot pueda alcanzarla sin problemas es conocida.
- Se cuenta con un archivo PCD con la pieza con el defecto, así como un PCD con la pieza original sin defectos.
- No es necesario utilizar material de soporte, ya que el material se depositará siempre en una superficie de la pieza o del material depositado previamente.

Cada uno de los supuestos trae consigo consecuencias que facilitarán la implementación de la estrategia. El no necesitar material de soporte permite utilizar técnicas de trayectorias de impresión 3D sin las restricciones que tienen estas.

Sumado a lo anterior, se generará una estrategia que pueda seguir las trayectorias con la herramienta de forma perpendicular a la superficie, bajo el supuesto de que ello mejorará la deposición del material, similar a lo indicado en la sección 2.2.7

La estrategia determinada se inspira en una combinación de técnicas para la generación de capas no planares.

Para poner a prueba la estrategia, se utilizan archivos PCD con representaciones de una sección de la pieza que incluye el defecto. Estos archivos PCD son generados por un programa tipo CAD, que luego se procesa con el software CloudCompare. El origen de los modelos de las fallas también puede provenir del escaneo de una pieza real.

4.2.1 Traducción al lenguaje del brazo robótico

Si bien la traducción al lenguaje del brazo robótico es el último paso, se requirió estudiar los formatos y estructuras que tiene el lenguaje del robot, para así generar una solución al relleno de defectos que además sea intuitivo de convertir al lenguaje del robot.

Tras haber visto los distintos formatos de indicar los recorridos, se decidió utilizar el formato AV, por su conveniente relación con el formato de las nubes de puntos PCD, ya que el vector que indica la normal de la superficie del punto puede usarse como el de la dirección de la herramienta al invertir su valor.

4.2.2 Pre procesamiento

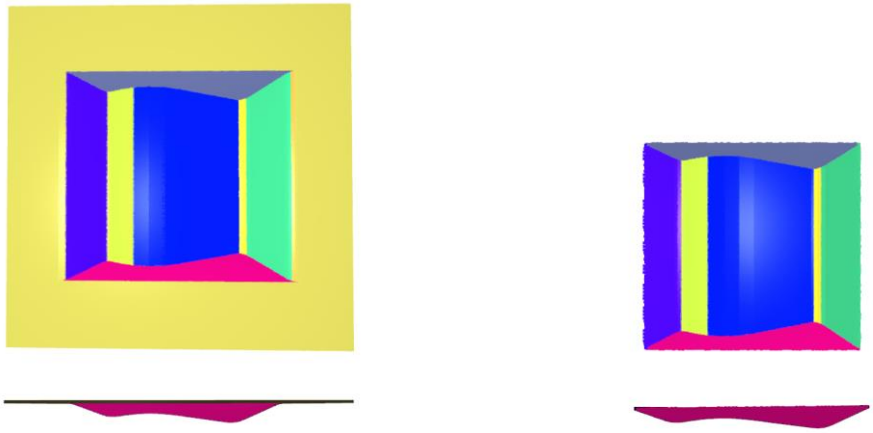
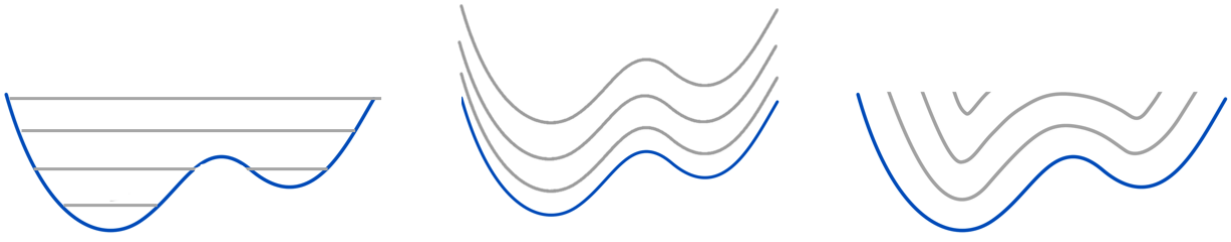


Figura 4.1 Vista superior y frontal de un modelo de defecto antes y después de aislar el defecto.

4.2.3 Generación de capas no planares

La generación de capas no planares sobre la superficie del defecto se inspira en las estrategias de contornos paralelos. La deposición de material se llevaría a cabo con una orientación normal a la superficie, por lo que se espera que se conserven las geometrías de los cordones de soldadura descritos. Con ello la generación de capas se vería como en la Figura 4.2 (c), donde se tiene una distancia constante entre las normales de las distintas capas, a diferencia del caso de la Figura 4.2 (b), que fue generado con un desplazamiento de la superficie en el eje Z, donde se terminan teniendo distintas alturas de cordones.

Por otro lado, en comparación con el caso de la Figura 4.2 (a), donde se rellena con capas planares, esta estrategia se acomoda bien para casos en que se tienen distintas profundidades en el defecto.



(a) (b) (c)

Figura 4.2 Diagramas de estrategias de relleno

Para llegar a la generación de capas del método c, se aprovecha que el formato de archivo PCD contiene información de las normal correspondiente a un punto que indica la dirección en la cual apunta la superficie. Con ello, es posible utilizar este vector para trasladar el punto. Si todos los puntos se trasladan en dicha dirección en una distancia determinada, se genera una capa de puntos sobre la superficie previa.

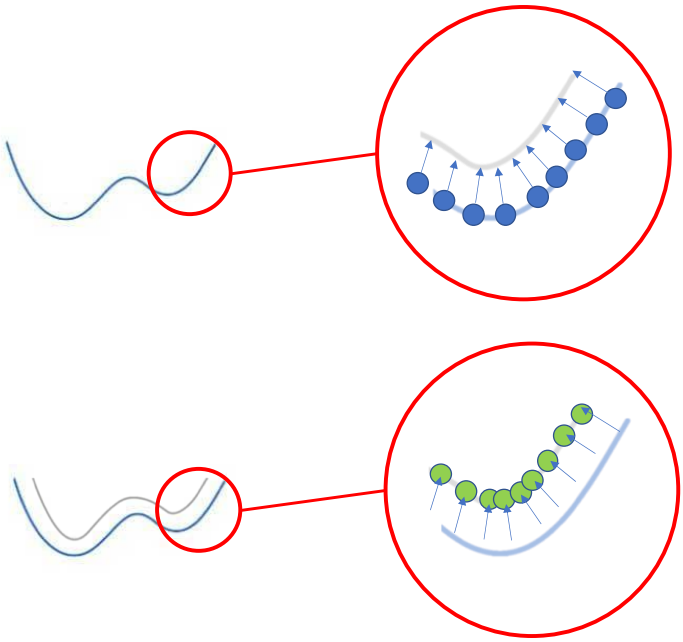


Figura 4.3 Representación de la traslación de puntos para generar nuevas capas.

Las principales dificultades de esta técnica radican en la elección de las normales orientadas hacia fuera de la pieza y además asegurarse de que la capa sea continua, por lo que el método requiere tener soluciones para las sobreposiciones de puntos y espacios volúmenes sin puntos. En la

Figura 4.4 (a) y (b) se ven representados estos problemas, la superposición de puntos y la ausencia de ellos respectivamente. Estos problemas se generan en los puntos donde la superficie cambia la orientación de sus normales de forma poco suave. La flecha indica desde donde se ve el defecto, ya que es un elemento clave para discernir qué puntos están sobre o bajo el relleno para el caso de la sobreposición de puntos.

(a) (b)

Figura 4.4 Problemas de la generación de capas por desplazamiento de normales.

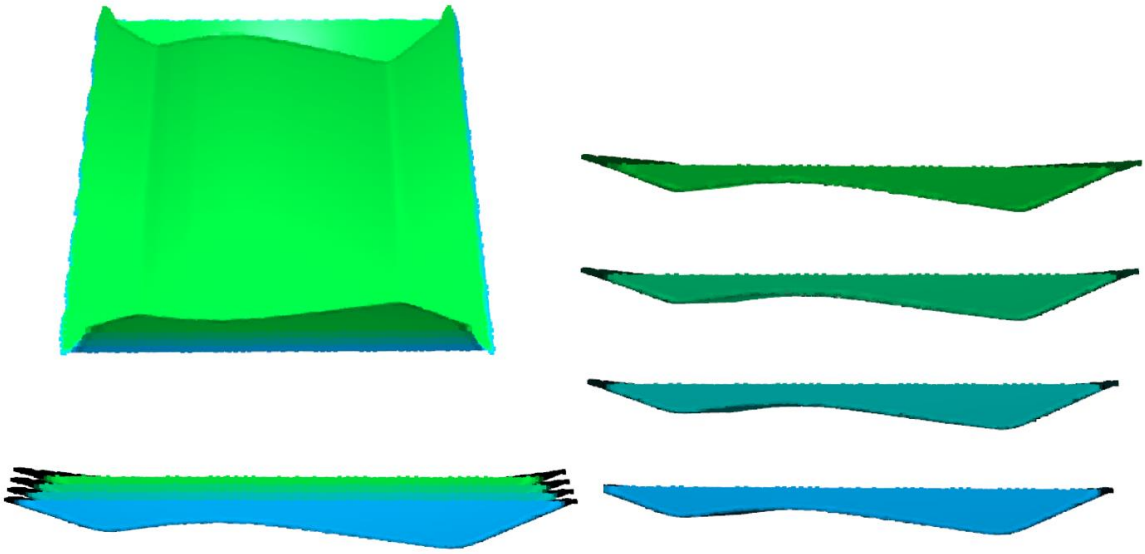


Figura 4.5 Visualización de las distintas capas generadas

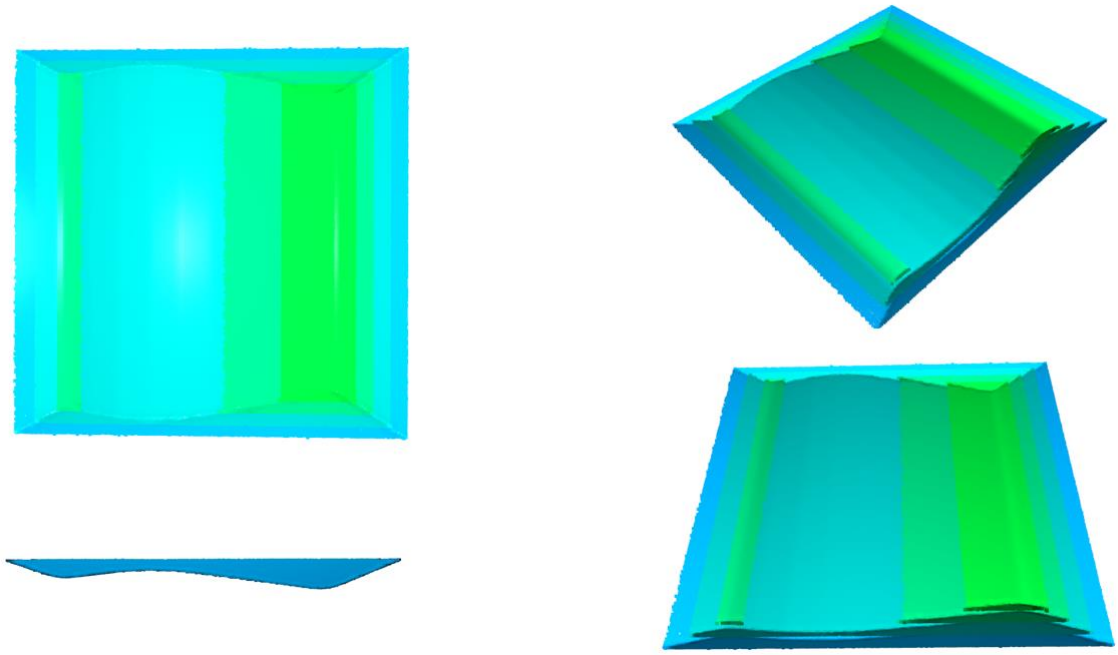


Figura 4.6 Visualización del conjunto de capas, tras haber eliminado los puntos fuera del volumen a rellenar.

4.2.4 Recorrido de capas

Una vez generadas las capas, estas deben ser recorridas, sin embargo, al trabajar con nubes de puntos, se tiene el inconveniente de que las posiciones se almacenan sin ningún orden en particular, por lo que para ser recorridas en un orden relativo a ejes cartesianos estas deben ser previamente ordenadas o encontrar otra solución que permita su recorrido.

A su vez, es posible identificar los puntos que se encuentran en la vecindad de un punto, utilizando un árbol que almacena sus posiciones. Con ello, esta herramienta permite conocer las posiciones cercanas a cierto punto, por lo que basta determinar un punto inicial para poder iniciar el recorrido en una dirección dada.

Debido a que las capas pueden tener distintas formas en el espacio, un recorrido en una dirección fija no será necesariamente la mejor solución.

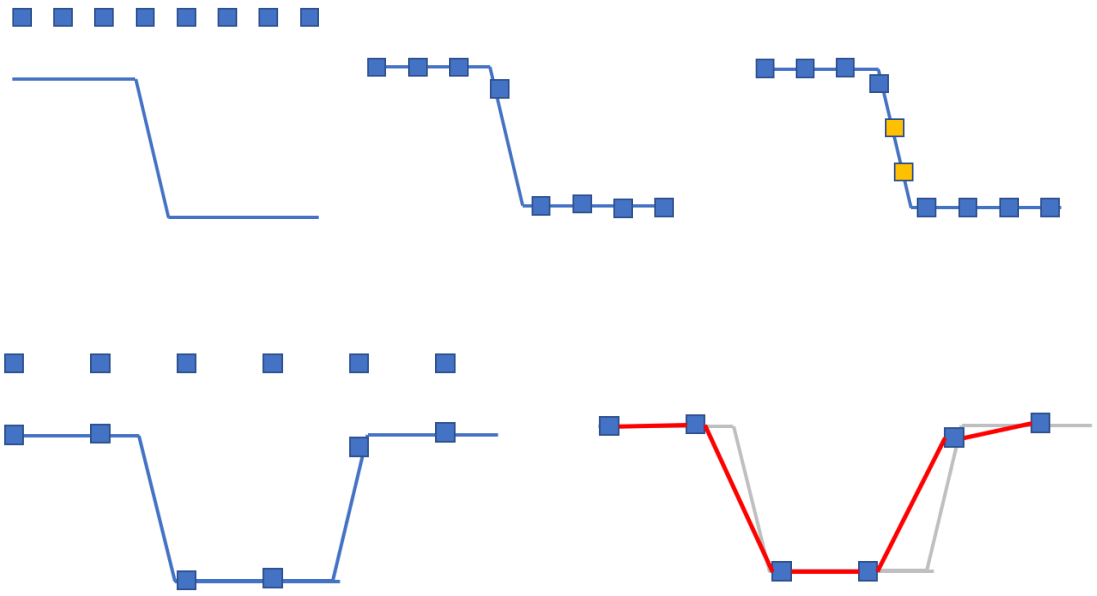


Figura 4.7 Problemas por elegir puntos espaciados en un eje fijo.

A la hora de recorrer la nube de puntos, es necesario que exista una distancia entre los puntos a intersecar lo suficientemente pequeña para que no se llegue a una geometría no representativa, tal como en el caso de la Figura 4.7 . En el caso de utilizar un método de estas características, debiese seguirse una intuición similar al teorema de muestreo de Nyquist, el cual señala que la frecuencia de muestreo debe ser al menos 2 veces mayor que la mayor frecuencia de la señal.

Teniendo en mente lo anterior, se propone un paso previo a la generación de trayectorias, que tiene como finalidad el facilitar el recorrido de la nube. Para lograr lo anterior, se buscará dividir la capa de la superficie a recorrer en nubes de puntos que tengan una forma lineal, es decir que tengan una forma estrecha y un inicio y fin determinado. Esta forma se generará como una nube de puntos, la cual será proyectada sobre la superficie a recorrer, almacenando en nubes de puntos las intersecciones entre la proyección y la superficie.

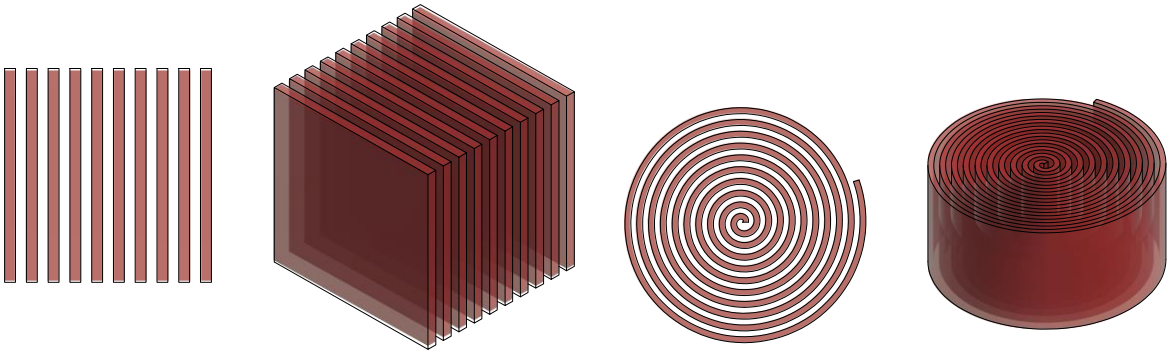


Figura 4.8 Representación de las proyecciones de distintas nubes de puntos.

Cabe mencionar que esta división podría tener distintas formas, solo bastaría proyectar una nube de puntos distintas y asegurarse de tener un método para poder recorrerla de manera efectiva.

Esta aproximación a la resolución del problema trae consigo una limitación, la cual se produce al utilizar una distancia de espaciado de líneas e equivalente a la distancia crítica d*.

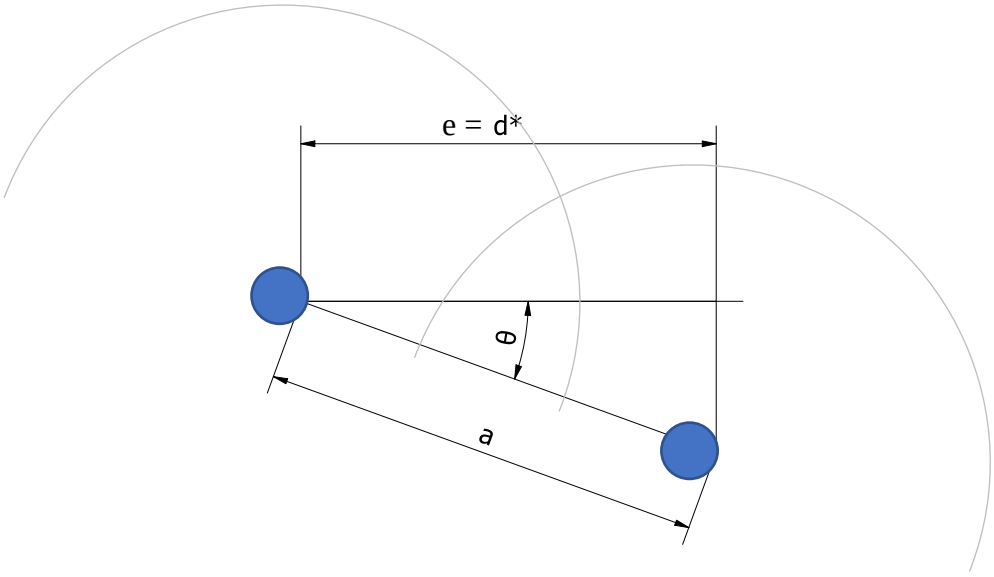


Figura 4.9 Representación de distancia entre centros (puntos azules) en el caso de una superficie inclinada.

Debido a que se desea mantener una distancia entre centros equivalente a d* para un mejor acabado superficial, el imponer esa distancia en el espaciado e de las nubes utilizadas para la intersección previa al recorrido, significa que no se puede mantener esa distancia d* cuando el plano es inclinado.

Con el fin de poder caracterizar a lo anterior, se calculan las inclinaciones de superficie para las cuales utilizar este espaciado entre líneas d* se mantiene en un rango aceptable. Para ello se considera la ecuación:

$$a = c \cdot w \tag{1}$$

Donde

a: es la distancia entre centros de cordones de soldadura

c: es la relación de la distancia entre centros con el ancho del cordón de soldadura

w: es al ancho del cordón de soldadura

Tabla 4.1 Relación entre centros de cordones de soldadura y el ángulo de la superficie al espaciar la división de la nube en d* = 0.738·w.

c	θ°
0,738	0
0,75	10,2
0,8	22,7
0,85	29,7
0,9	34,9

Con la tabla anterior, se escoge que un valor de c=0,8 es un límite aceptable, permitiendo tener una inclinación de 20° respecto a la horizontal. En caso de conocer la inclinación que tendrá la superficie, podría calcularse el valor de e adecuado para mantener una distancia entre cordones idónea.

Continuando con la intersección de nubes de puntos, para ayudar a la visualización de esta etapa, se colorean las distintas nubes de puntos con un color distinto, como se puede apreciar en la Figura 4.10 donde el más celeste indica que se encuentra al inicio y el verde al final. Además, se presentan en esta misma figura intersecciones con distintas nubes de puntos.

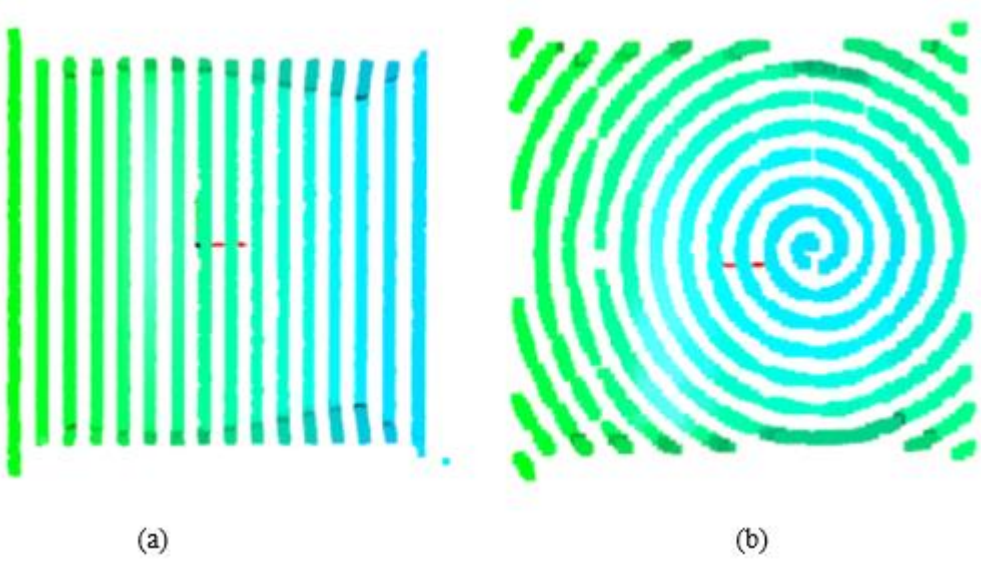


Figura 4.10 Seccionamiento de la nube, a) tipo raster zizag, b) tipo espiral

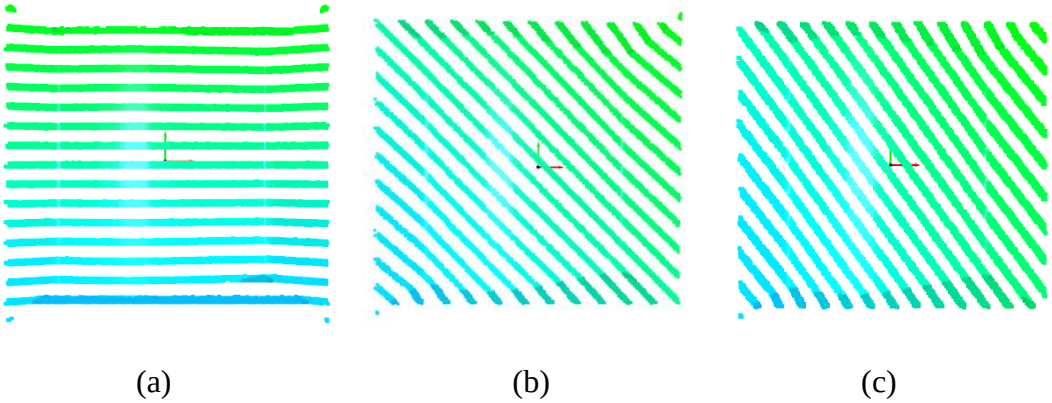


Figura 4.11 Seccionamiento tipo ráster zigzag con distintos ángulos.

Recorrer las sub nubes de puntos presenta un desafío ya que se debe procurar mantener las partes importantes de la geometría. Para ello se desarrolló el siguiente método

Para el caso del ráster zigzag, se ingresa en la función la orientación de las capas en forma de vector, por lo que el primer punto se buscará encontrando el punto más distante de la nube en la dirección contraria al vector de orientación. Posteriormente el recorrido se realiza buscando el vecino más lejano en la dirección de orientación de la nube, en un radio equivalente al ancho del cordón. El algoritmo se detiene al llegar a un punto donde el vecino más lejano en la dirección deseada esté a una distancia menor al ancho del cordón o que se elija el mismo punto.

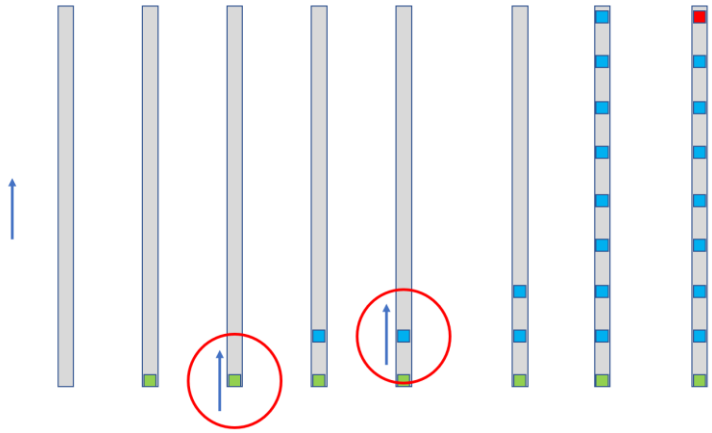


Figura 4.12 Diagrama del método de recorrido para el caso ráster zigzag

En el caso del espiral, el punto inicial se calcula según su distancia respecto al centro de la espiral en el plano X-Y, al ser una espiral euclidiana, este valor se relaciona con el ángulo. Tras haber seleccionado el punto inicial, se recorre la nube en dirección horario o antihorario según se haya indicado. El vector de dirección del recorrido se modifica según la ubicación del punto anterior, logrando que el algoritmo pueda recorrer distintas formas de nubes mientras se mantenga una forma de línea. Las condiciones de término del algoritmo son las mismas que en el caso de ráster zigzag.

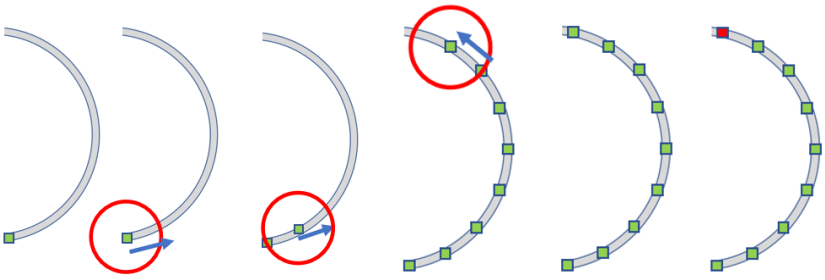


Figura 4.13 Diagrama del método de recorrido para el caso espiral.

En el caso de que la nube tenga una forma discontinúa, es posible obtener el recorrido de esos puntos eliminando las partes de las nubes que ya han sido recorridas y repitiendo el algoritmo de recorrido.

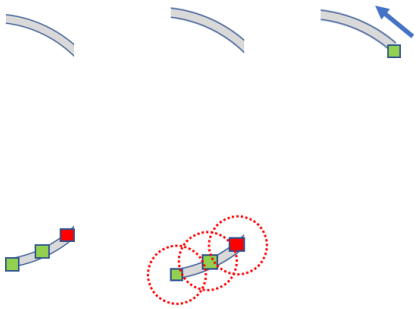


Figura 4.14 Diagrama del método para recorrer nubes de puntos de forma discontinua.

Debido a que el punto seleccionado es una representación de los puntos de su vecindad, es que la normal de este no debe ser heredada directamente, sino que se toma un promedio de las normales de sus vecinos.

Tras haber seccionado la capa en sub nubes de puntos, estas pueden ser recorridas, extrayendo sus puntos representativos. Para evitar tomar normales de puntos aislados, las normales de los puntos son el resultado del promedio de sus vecinos, teniendo así una normal característica de la superficie en esa ubicación.

(a) (b)

Figura 4.15 (a) sección de la nube de puntos y (b) recorrido generado a partir de ella.

Hacia el final de esta etapa, se tendrá una lista con nubes de puntos que representan el recorrido.

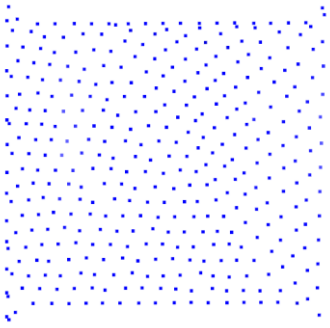


Figura 4.16 Nube de puntos que representa el camino de soldadura generado para una capa.

4.2.5 Post procesamiento

Tras tener todos los recorridos de las sub nubes de puntos, se requiere realizar las siguientes acciones: conectar las nubes de puntos para seguir un recorrido completo; agregar puntos para el levantamiento de herramienta donde sea necesario; integrar una forma de indicar la acción de soldadura o no en el punto recorrido para su traducción al lenguaje del robot.

Para conectar los puntos, se procede a unir las distintas trayectorias en una lista de nubes de puntos, las cuales se agregan según su orden de generación, ya que en la subdivisión misma se genera un orden espacial para estas trayectorias.

Por otro lado, los puntos de levantamiento de herramienta, se identifican comparando distancia entre el inicio y término de cada sub nube de puntos, agregando puntos cada vez que su distancia sea mayor a un umbral.

Finalmente, para poder almacenar la información correspondiente a la acción de encendido o apagado de la antorcha en el punto, se utiliza la información del color asociado al punto de la nube, característica que hasta el momento había sido utilizada solo para fines visuales.

4.2.6 Visualización del camino de soldadura

La visualización del camino de soldadura es una parte clave en el proceso, ya que permitirá la verificación por parte del usuario de la calidad de los caminos realizados. Con ello, una representación espacial en un gráfico 3D resulta insuficiente para tener la información necesaria, ya que la orientación de las normales junto con el orden del recorrido son características igual de críticas.

4.3 Revisión de la generación de trayectorias y otras aplicaciones

En esta etapa se procede a revisar el comportamiento de la estrategia en distintos defectos, correspondientes a los casos de estudio. Con lo anterior es posible identificar los principales problemas y limitaciones de la estrategia de relleno propuesta.

Posteriormente se revisan otras aplicaciones en que se pueda utilizar lo desarrollado, enfocado principalmente en las aplicaciones de recubrir piezas y el rellenar piezas de superficies no planas. Para ello se exploró la generación de capas que puedan envolver el volumen de forma total y el generar trayectorias delimitadas por la forma del modelo final.

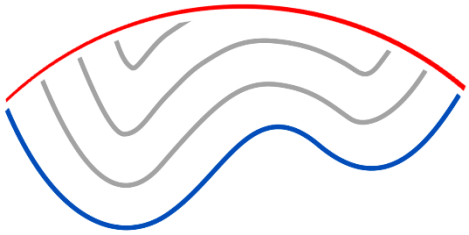


Figura 4.17 Diagrama del relleno de una pieza de superficie no plana.

4.4 Descripción de funciones utilizadas

En este apartado se describirán las funciones creadas y utilizadas, así como la lógica detrás de ellas.

Función Volt: Utiliza los datos empíricos para calcular el voltaje en función del amperaje.

Función beamgeometry: Retorna los parámetros que definen la geometría del cordón, altura y ancho, en función del amperaje. Utilizando los datos empíricos.

Función plane: Genera una nube de puntos distribuidos de forma uniforme en un plano.

Función spiral: Genera una nube de puntos distribuida en forma de espiral euclidiana, puede describir una forma horario o antihorario. Los puntos se generan siguiendo la ecuación del arco de la espiral euclidiana.

Función deleteintersect: La función elimina los puntos a la “nube de puntos 1” que se intersecan con la “nube de puntos 2”. La intersección se realiza midiendo la distancias entre los puntos de las nubes, considerando intersección aquellos puntos que se encuentran a una distancia “e” entre las nubes.

Función intersect: La función conserva los puntos de la "nube de puntos 1" que se intersecan con la "nube de puntos 2". Los criterios de intersección son los mismos que en deleteintersect.

Función addpoints: Mezcla los puntos de una lista de nubes de puntos en la misma nube PCD. Los puntos se agregan en forma consecutiva en el orden de la lista ingresada.

Función pathvis: Permite la visualización de la nube de puntos con el paquete matplotlib.

Función animate: Permite la animación de la nube de puntos con el paquete matplotlib.

Función projection: Función que proyecta una nube de puntos en una dirección determinada. Lo anterior se logra creando múltiples copias de la nube de puntos espaciadas una distancia determinada entre ellas.

Función normtrans: Crea una nube de puntos trasladando los puntos de la nube ingresada, desplazándolos una distancia d en la dirección de las normales asociadas a cada punto. Al final procesa la nube dejando solo los puntos que son visibles desde una "cámara" que se indica en coordenadas. El valor por defecto es ubicado en la parte superior de la nube de puntos.

Función offset3d: Genera tantas capas como se indiquen, utilizando nubes generadas con normtrans.

Función multioffset3d: Genera tantas capas como se indiquen, utilizando nubes generadas con normtrans, pero en múltiples direcciones, permitiendo cubrir una mayor superficie que offset3d.

Función preslice01: Esta función subdivide la nube de puntos ingresada en nubes de puntos en forma de líneas. Esto se logra intersecando planos consecutivos en la nube de puntos, distanciados en una distancia d entre ellos.

Función preslice02: Al igual que preslice01, esta función subdivide la nube de puntos en forma de líneas en espiral. Lo anterior se logra intersecando una espiral euclidiana proyectada en el eje Z.

Función slice01: Esta función recorre nubes de puntos que previamente han sido generadas en formas de líneas. Ubica un punto inicial en la dirección contraria a la ingresada, y luego recorre la nube buscando al vecino más lejano dentro de un radio determinado. Para determinar qué vecino se encuentra en la dirección deseada, se realiza un producto punto entre la dirección de recorrido y la coordenada del punto respecto al punto seleccionado. El recorrido se detiene al llegar a un borde.

Función slice02: Esta función recorre nubes de puntos que previamente han sido generadas en forma de líneas en espiral. Ubica al punto inicial ubicando el menor valor de la coordenada según su ángulo en el plano X-Y y respecto al centro de la espiral. El recorrido se realiza con un método similar a slice01, con la diferencia de que el vector dirección va variando para poder adaptarse a

las curvas de la espiral. El recorrido se detiene al llegar a un borde o a la vecindad de un punto contenido en el recorrido.

Función completepath01: Utiliza reiteradamente la función slice01 hasta recorrer todas las nubes de puntos ingresadas. Si la nube de puntos tiene una discontinuidad, se vuelve a utilizar la función slice01 en la sección que aún no se recorre.

Función completepath02: Utiliza reiteradamente la función slice02 hasta recorrer todas las nubes de puntos ingresadas. Al igual que completepath01, en caso de presentar una discontinuidad, se repetirá la función slice02 en la sección de la nube aún sin recorrer.

Función addsafepoints: Añade puntos para el levantamiento de la herramienta, además de asignarles un color según la acción que se realice. Los puntos donde se levanta la herramienta serán aquellos que se encuentren a una distancia 3 veces superior al ancho del cordón de soldadura. Se utiliza este valor para no levantar la herramienta entre puntos que se encuentren lo suficientemente cerca entre ellos.

Función dtpspoints: Es la función que se encarga de la traducción al lenguaje del brazo robótico. En ella se debe indicar los parámetros de soldadura utilizados y el nombre del archivo CSR que se generará.

5 Resultados y discusión

5.1 Resultados

El primer resultado de este trabajo es el programa mismo, que describe y ejecuta la estrategia de relleno propuesta, posteriormente el resto de los resultados corresponde a lo que el programa entrega como respuesta.

El programa entrega como resultado el camino representado de 4 formas distintas, estas corresponden a visualización con la librería matplotlib, que permite identificar los puntos del recorrido y la línea de trayectoria entre ellos; visualización con Open3D, que permite revisar que las normales estén orientadas de forma coherente; animación del recorrido con matplotlib, que permite al usuario revisar el orden de recorrido del camino antes de cargar los datos al robot; archivo CSR para ser abierto por el programa DTPS del robot, el cual permite al usuario poder simular el recorrido en el robot. Para cada uno se utilizó un amperaje de 100 [A].

Los casos revisados son:

- Placa punto silla – Primera capa
- Placa punto silla – Capa discontinua
- Placa punto silla – Recorrido completo zigzag
- Placa punto silla – Recorrido completo espiral
- Placa bolsillo – Recorrido completo zigzag
- Placa bolsillo – Recorrido completo espiral
- Placa sombrero invertido – Recorrido completo zigzag
- Escaneo placa bolsillo – Recorrido completo zigzag
- Escaneo placa bolsillo – Recorrido completo espiral

Además, se exploró otras utilidades del programa, además de agregar una revisión de los principales problemas en la generación de trayectorias. Lo cual añade los puntos:

- Problemas en generación de trayectorias
- Generación de capas en múltiples direcciones
- Relleno en volúmenes no planares

Se presentan los resultados de tiempo y largo de camino para los principales recorridos.

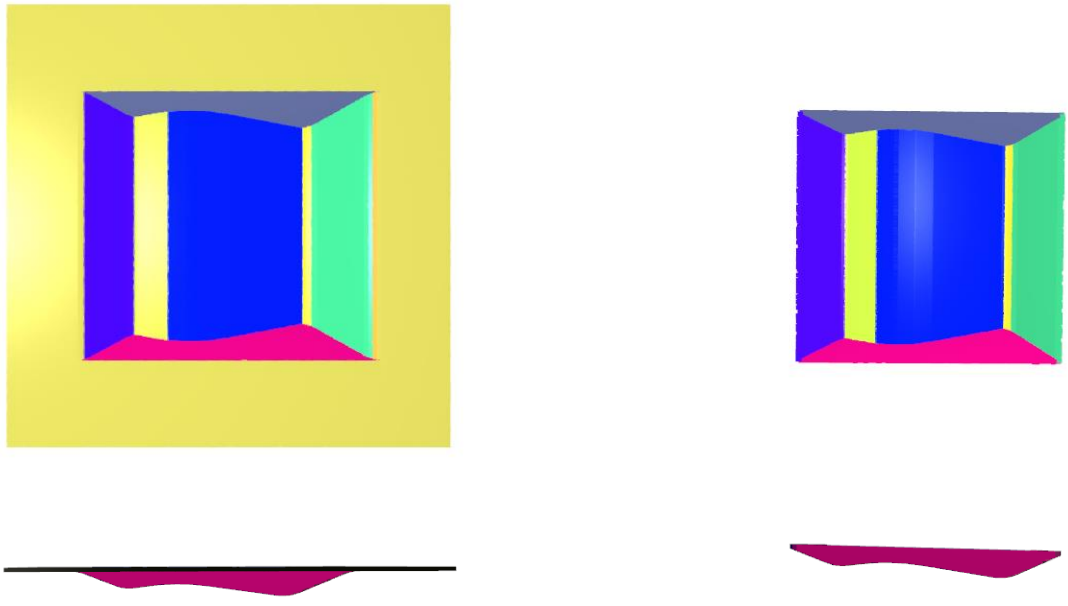
Tabla 5.1 Resultados de la simulación del recorrido en DTPS.

Recorrido	Tiempo total [s]	Largo de soldadura [m]	Pasos
Placa punto silla zigzag	707	3,616	920
Placa punto sillla espiral	832	3,477	1121
Escaneo real zigzag	1318	8,409	1661
Escaneo real espiral	1235	6,601	1520
Placa bolsillo zigzag	1112	6,37	1232
Placa bolsillo espiral	1328	6,192	1526

5.1.1 Placa punto silla – Primera capa

Primero se realizan pruebas en una capa en particular, para poder revisar de mejor forma como el algoritmo genera la trayectoria, ya que al tener muchas capas la visualización se sobrecarga.

Primero se revisa la aislación de la zona de la falla, lo cual se logra de manera satisfactoria, permitiendo seguir al siguiente paso.



(a) (b)

Figura 5.1 Nube de puntos antes (a) y después (b) de aislar la superficie con defecto.

La subdivisión de la capa en nubes de puntos que sean más fáciles de recorrer se logra ver en la Figura 5.2 , donde cada nube de puntos tiene un color distinto.

La Figura 5.2 muestra los puntos seleccionados de cada nube de puntos ubicados en el espacio correspondiente, además la Figura 5.4 muestra las normales de cada punto, mostrando que logran describir la forma de la superficie.

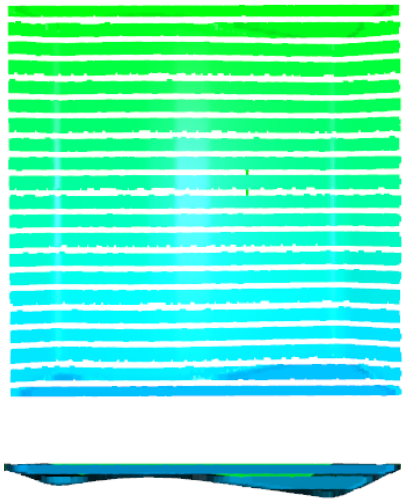


Figura 5.2 Visualización de la subdivisión de la nube de la primera capa.

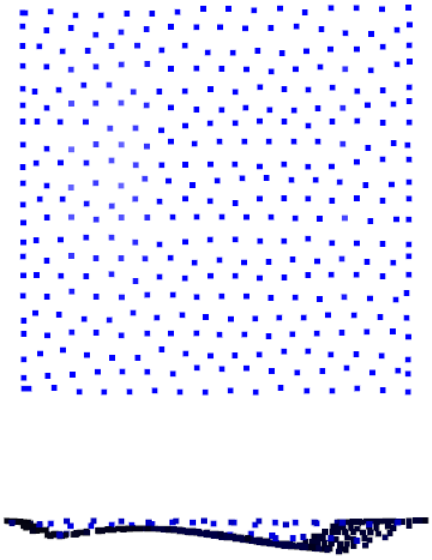


Figura 5.3 Representación de los puntos del camino generados para una capa.

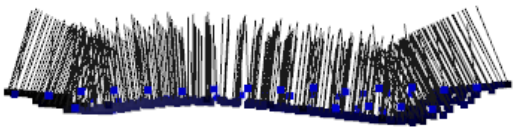


Figura 5.4 Representación de normales de los puntos generados para una capa.

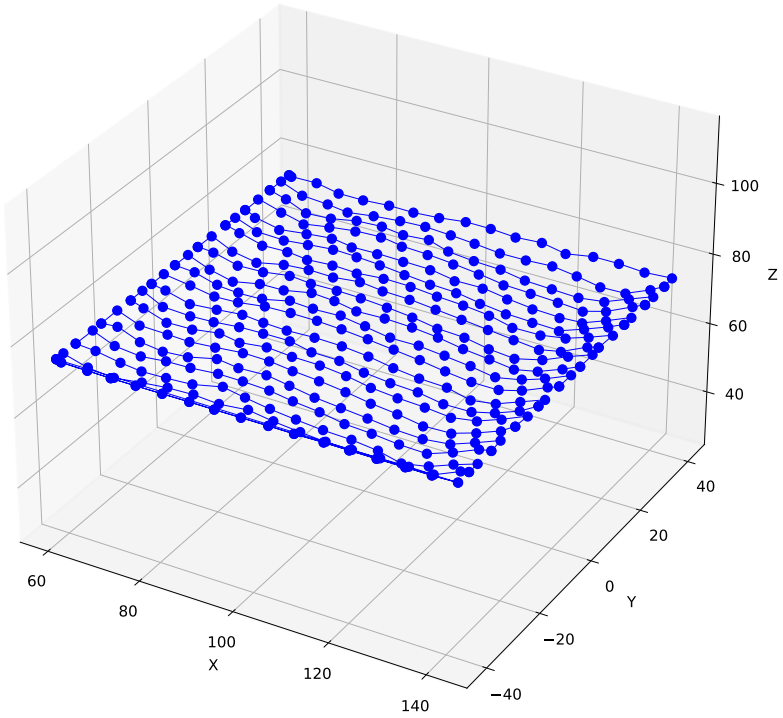


Figura 5.5 Representación de los puntos de la trayectoria generada para una capa.

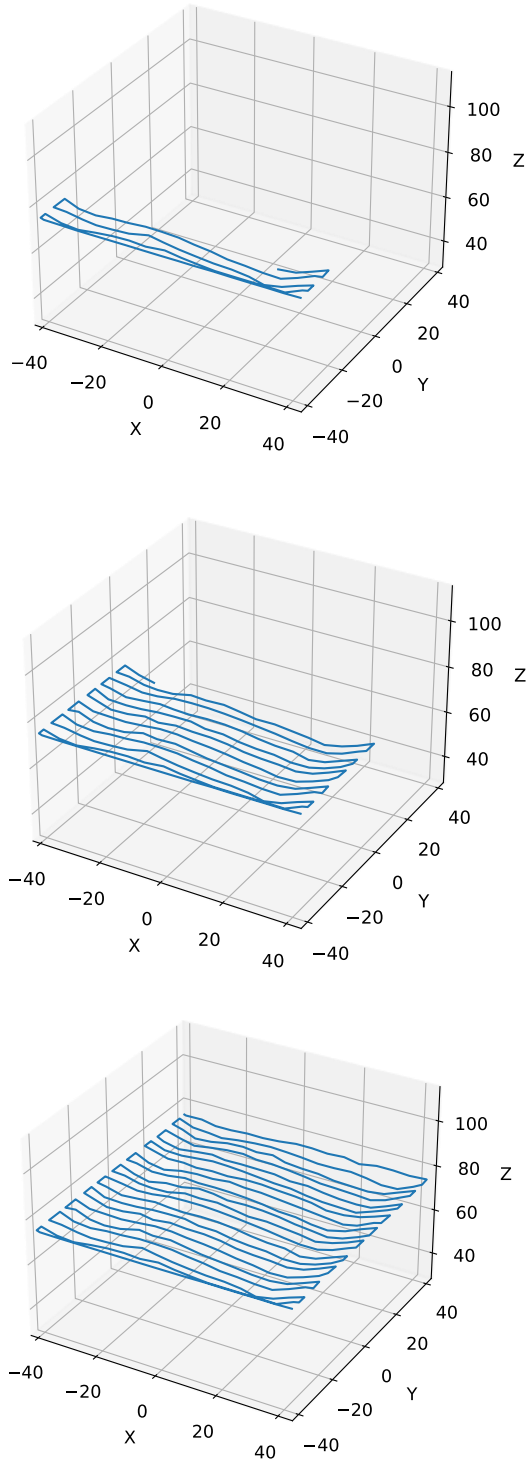


Figura 5.6 Visualización de la trayectoria recorrida en distintos tiempos.

5.1.2 Placa punto silla – Capa discontinua

Con la finalidad de revisar el comportamiento del algoritmo en capas que no son continuas, se realiza una prueba de una capa con estas características de forma aislada. La subdivisión en nubes de puntos logra llevarse a cabo sin inconvenientes como puede apreciarse en la Figura 5.7, así como la generación de la trayectoria y la inclusión de los puntos que describen el levantamiento de la herramienta, al mismo tiempo, las normales logran ser representadas de forma satisfactoria. Estos últimos puntos pueden revisarse en la Figura 5.8 y Figura 5.9 respectivamente.

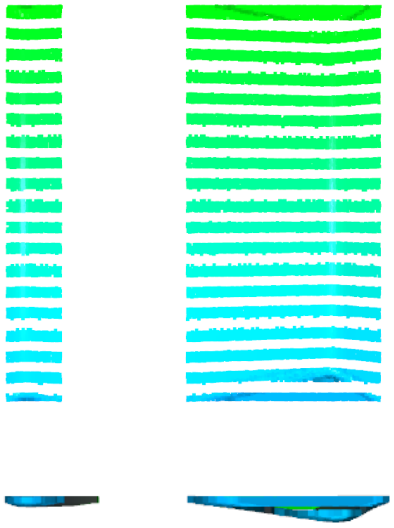


Figura 5.7 Representación de la subdivisión de una capa no continua.

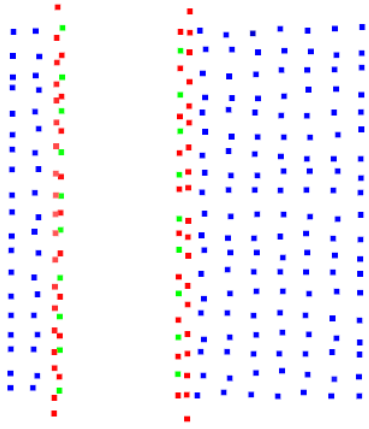


Figura 5.8 Representación de los puntos del camino generados para una capa no continua.

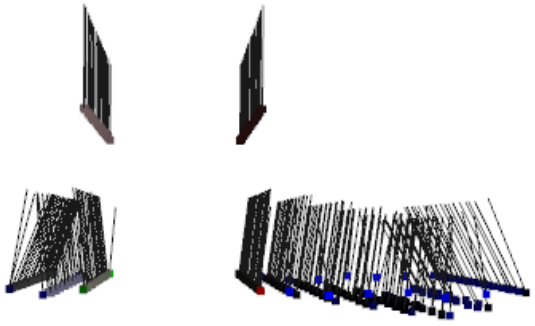


Figura 5.9 Visualización de las normales de cada punto de la trayectoria.

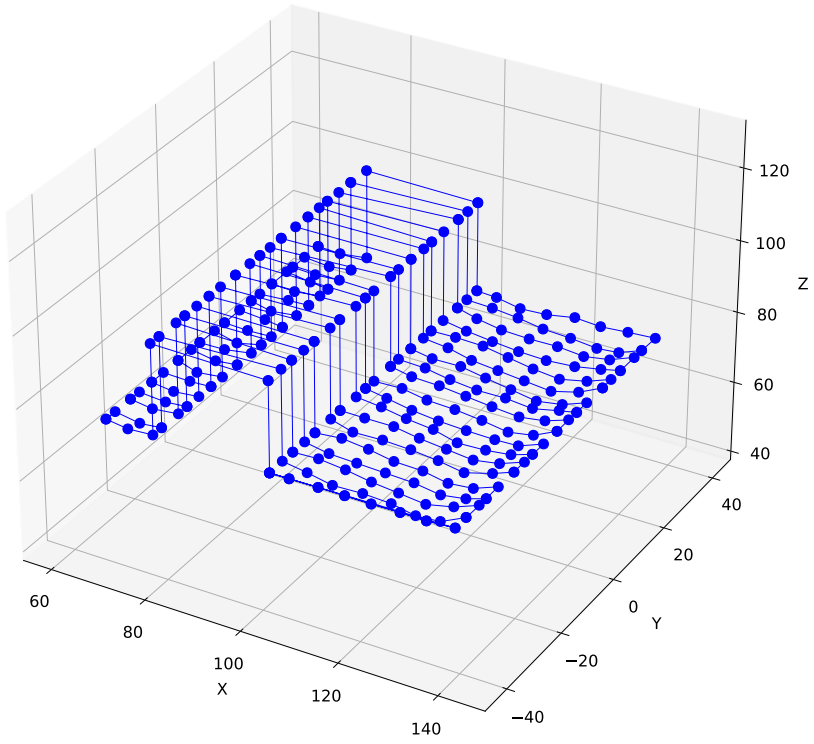


Figura 5.10 Representación de los puntos de trayectoria generada para una capa no continua.

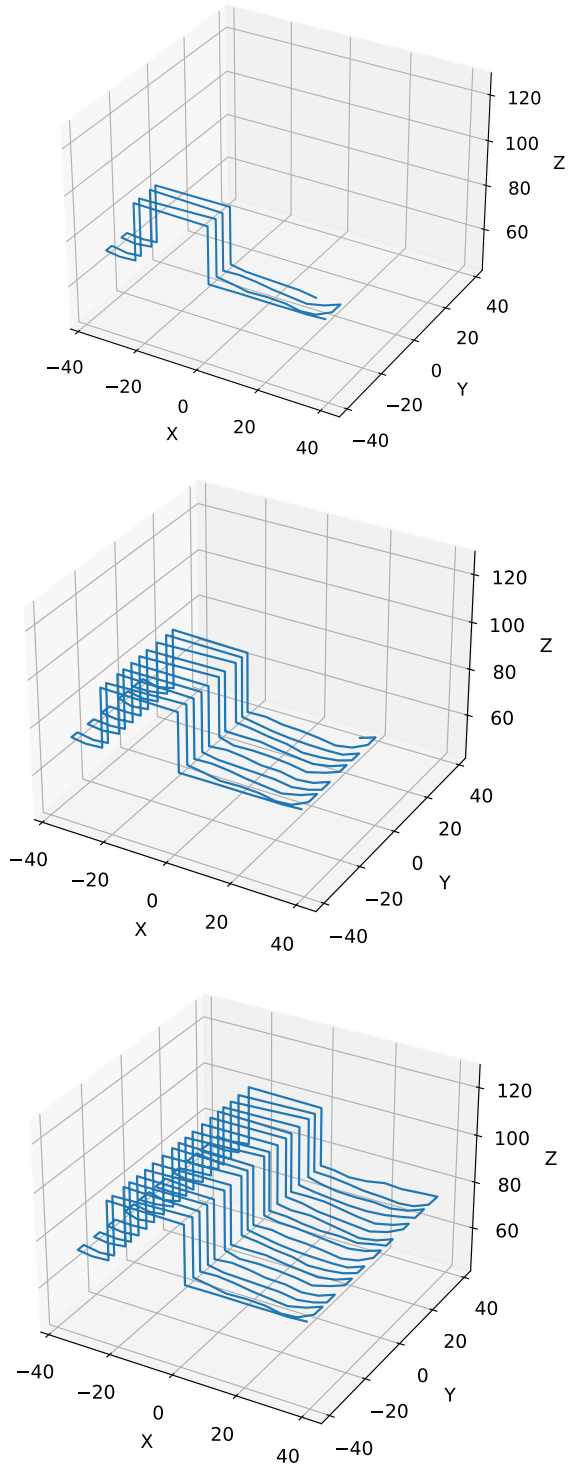


Figura 5.11 Visualización de la trayectoria para distintos tiempos.

5.1.3 Placa punto silla – Recorrido completo zigzag

El primer punto a revisar en el recorrido completo es la forma que tienen las capas generadas y su ubicación en el espacio respecto a las otras superficies, lo cual puede apreciarse en la Figura 5.12.

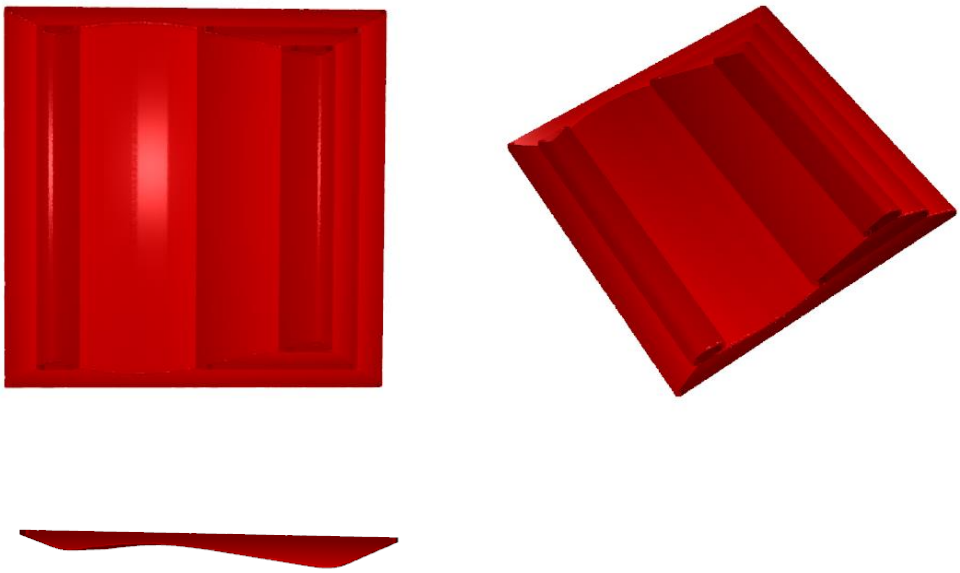


Figura 5.12 Vistas de las capas generadas.

A medida que se van añadiendo capas, los gráficos generados muestran cada vez menos información al superponer demasiados puntos, sin embargo, se revisan de todos modos.

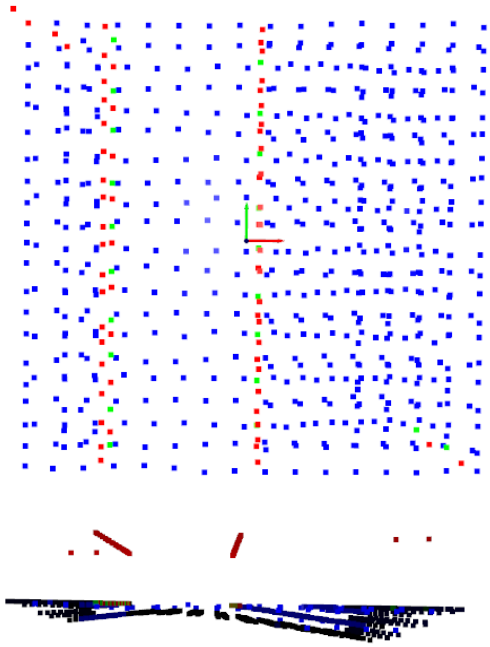


Figura 5.13 Visualización de la trayectoria generada en nubes de puntos.

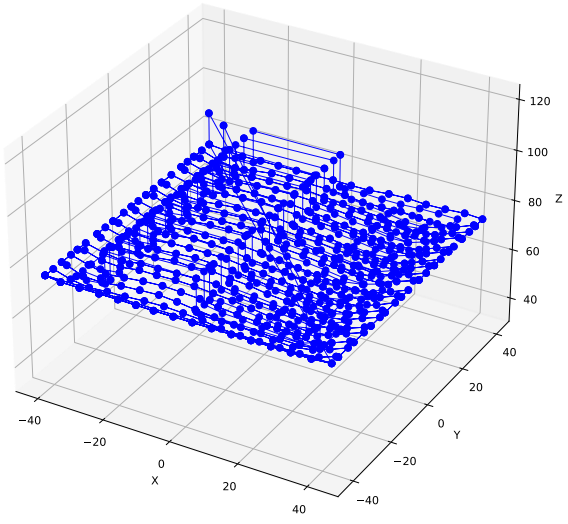


Figura 5.14 Visualización de los puntos de la trayectoria generada.

En las vistas del programa DTPS es posible ver las trayectorias en las que se aplica soldadura así como aquellas que se realizan en el aire. También es posible apreciar como la herramienta adopta la orientación de la normal descrita.

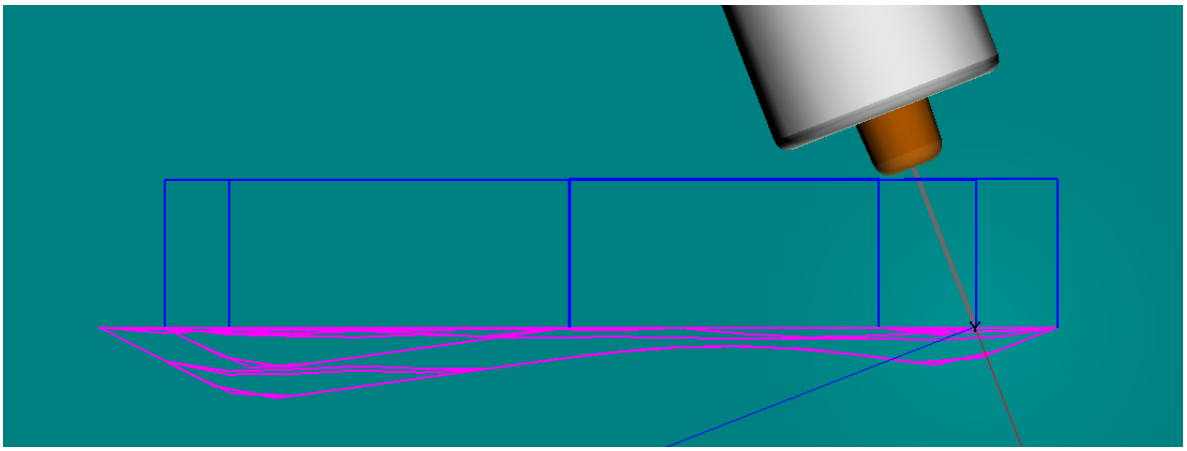


Figura 5.15 Vista lateral de la trayectoria visualizada en DTPS.

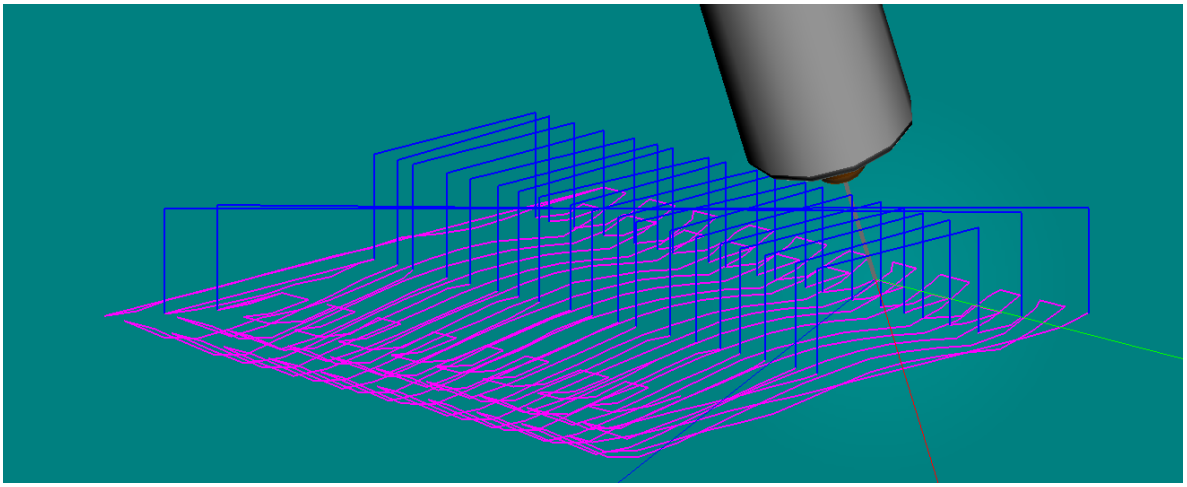


Figura 5.16 Vista de la trayectoria visualizada en DTPS.

5.1.4 Placa punto silla – Recorrido completo espiral

La estrategia espiral también logra llevarse a cabo con sus respectivos levantamientos de herramienta, los cuales pueden visualizarse en los colores de la nube de puntos generada.

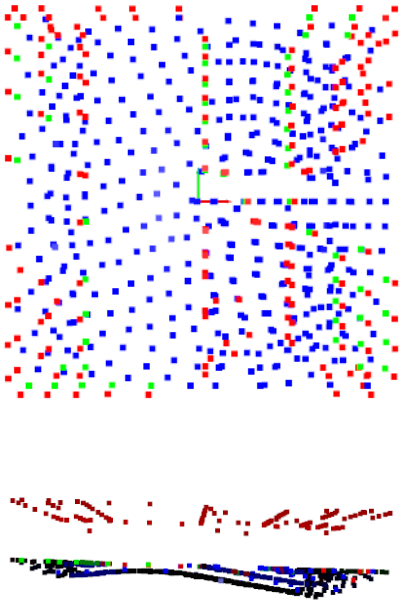


Figura 5.17 Visualización de la trayectoria generada.

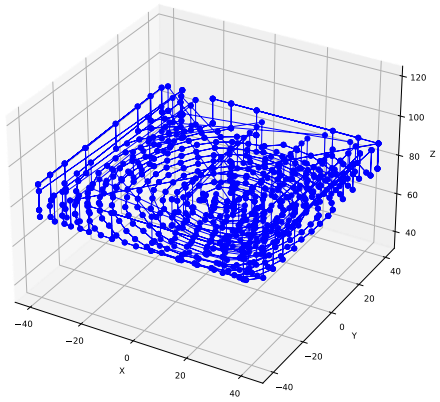


Figura 5.18 Visualización del recorrido en matplotlib.

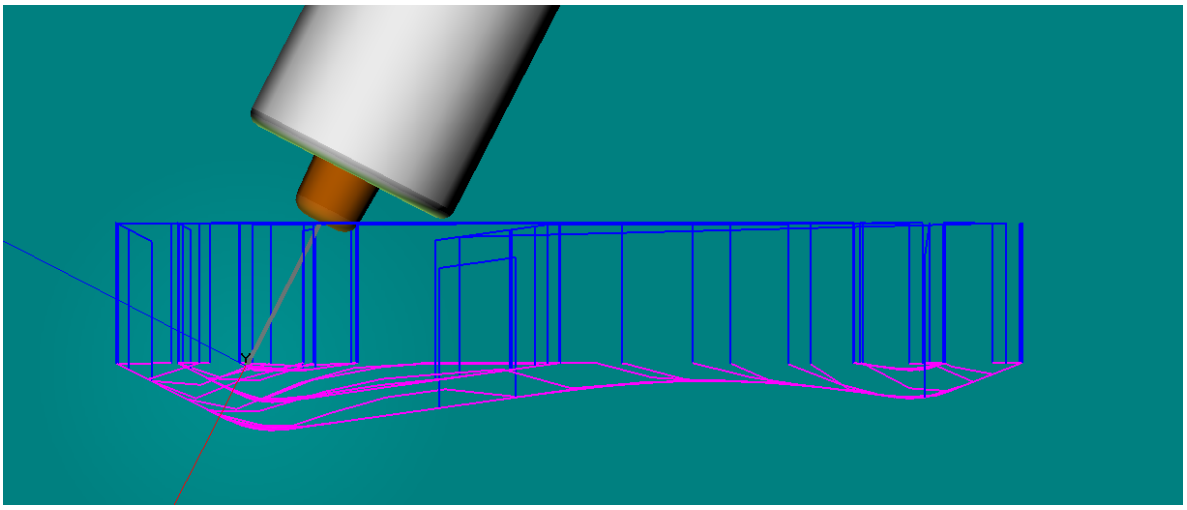


Figura 5.19 Vista lateral de la visualización de trayectoria en DTPS.

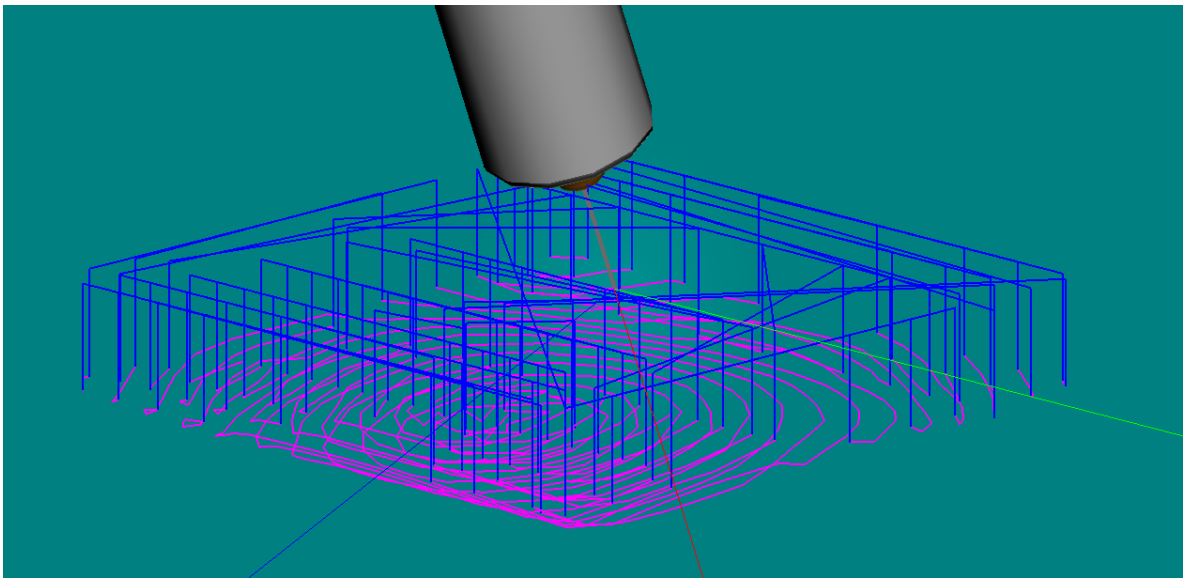


Figura 5.20 Vista de la trayectoria en la visualización en DTPS.

5.1.5 Placa bolsillo – Recorrido completo zigzag

Para generar estas trayectorias, fue necesario rotar levemente la pieza, debido a que se recorría de forma errática las superficies de las capas que eran planos totalmente lisos y alineados con el plano perpendicular a la vista de la cámara.

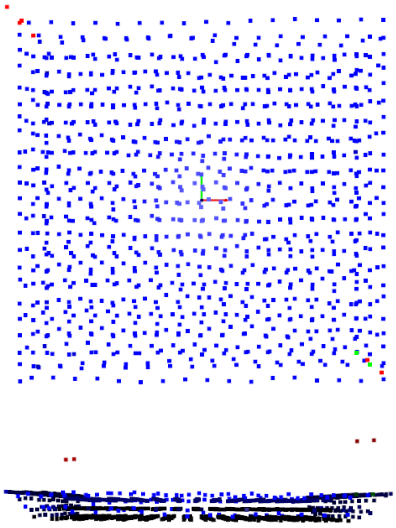


Figura 5.21 Visualización de los puntos generados en Open3D.

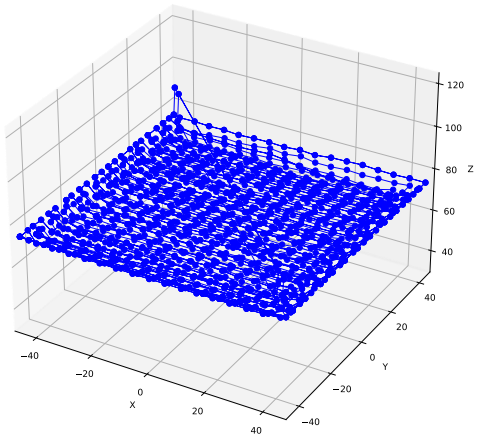


Figura 5.22 Visualización de los puntos de la trayectoria, en matplotlib.

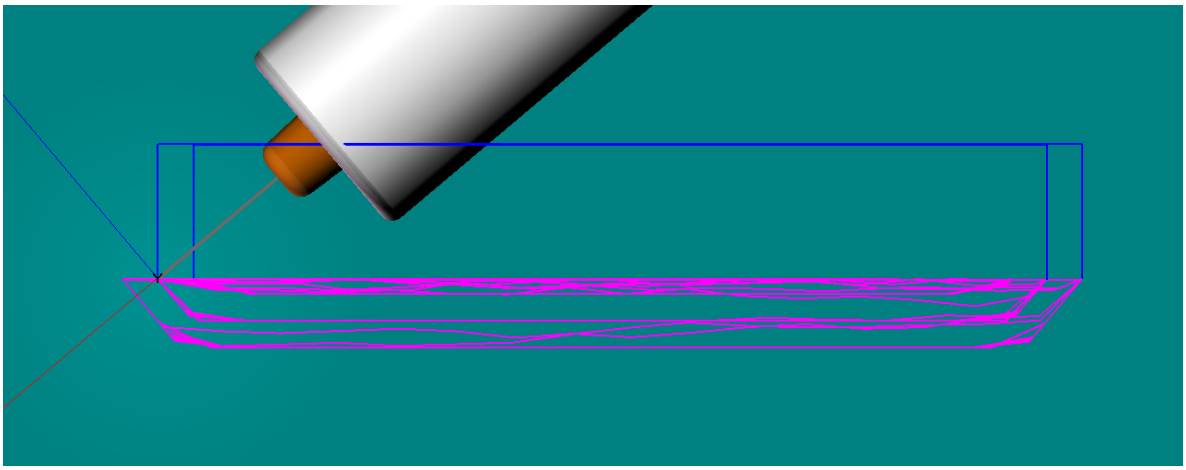


Figura 5.23 Vista lateral del recorrido en su visualización en DTPS.

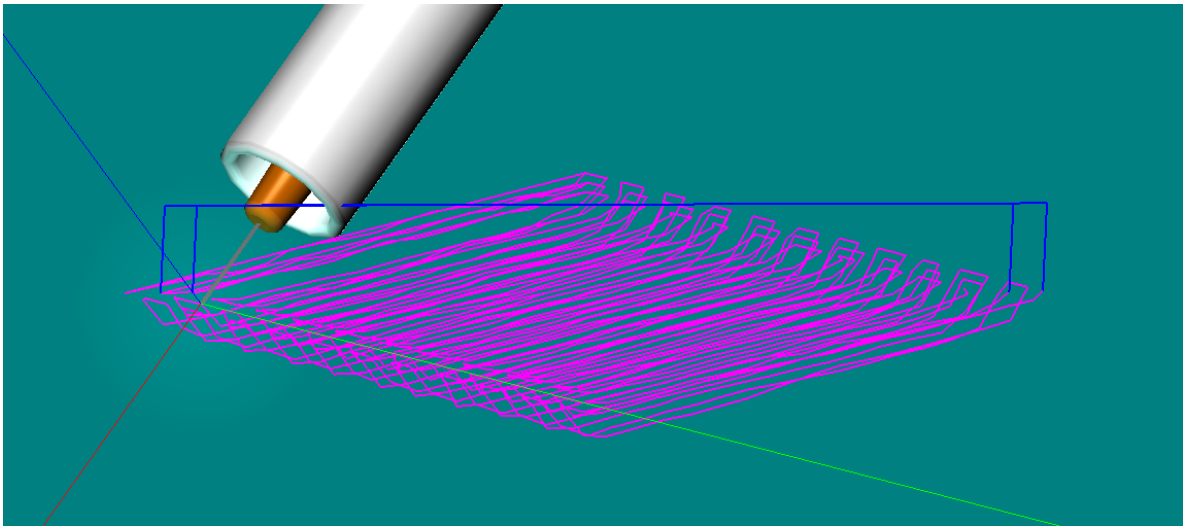


Figura 5.24 Vista del recorrido en su visualización en DTPS.

5.1.6 Placa bolsillo – Recorrido completo espiral

Al igual que en el caso del recorrido zigzag, para generar esta trayectoria fue necesario girar levemente la superficie para poder desarrollar la estrategia sin problemas. Se aprecia que existen puntos de levantamiento de herramienta en el interior de la espiral, evento que se escapa a lo esperado.

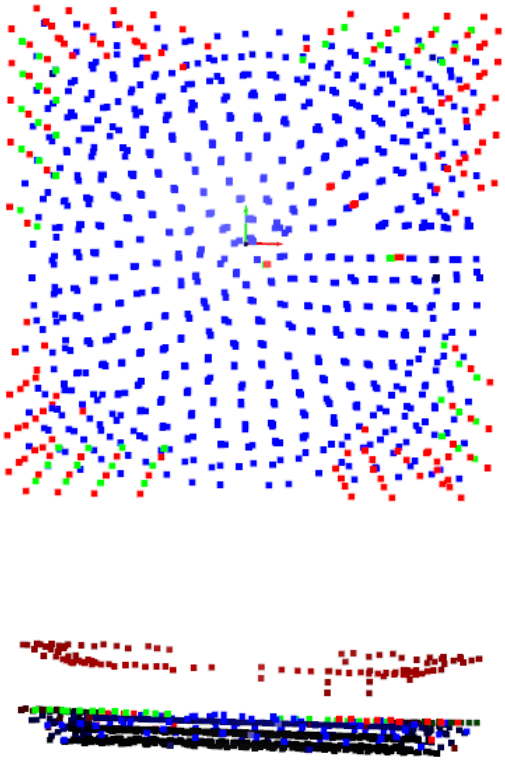


Figura 5.25 Puntos de la trayectoria visualizados en Open3d.

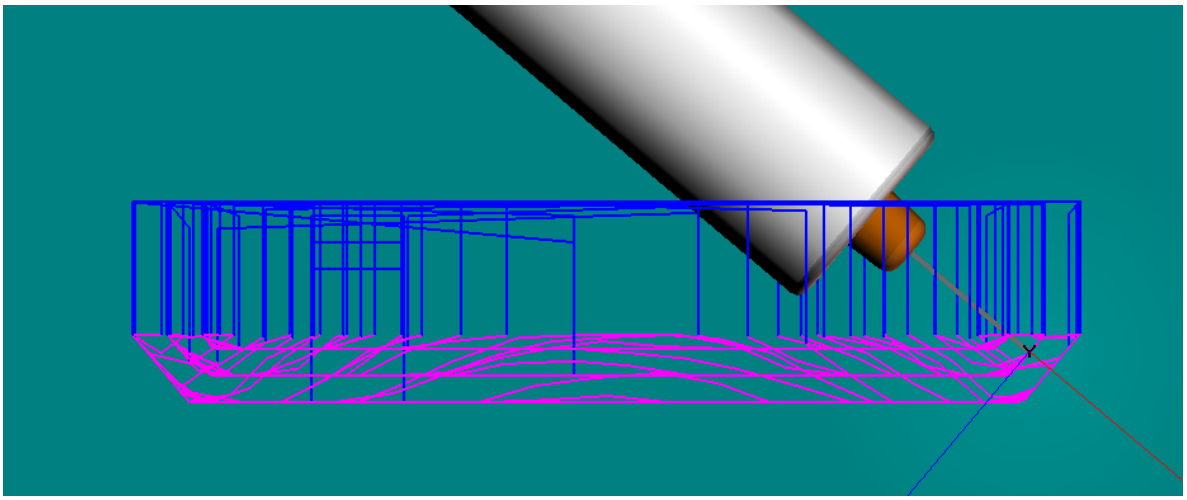


Figura 5.26 Vista lateral del recorrido visualizado en DTPS.

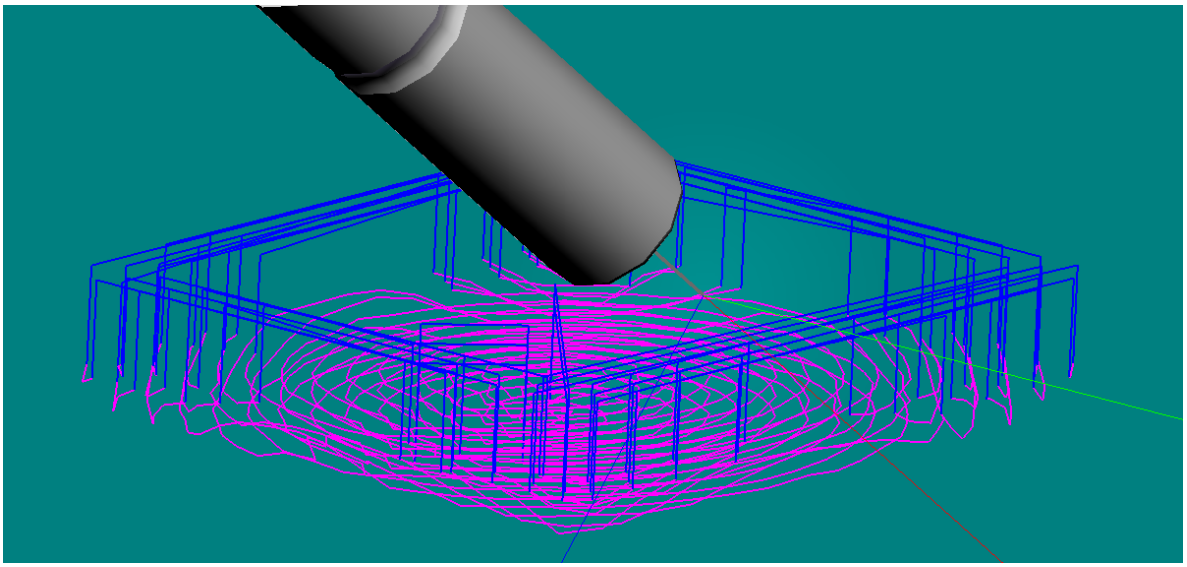


Figura 5.27 Vista del recorrido visualizado en DTPS.

5.1.7 Placa sombrero invertido– Recorrido completo zigzag

El recorrido de esta figura tiene como finalidad el revisar el comportamiento de la estrategia en formas que varíen más su altura, además las dimensiones de la pieza son del orden de 50x50 [cm], lo que lo hace considerablemente mayor a las otras piezas utilizadas. La dimensión de la nube de puntos aumentó el tiempo de procesamiento, sin embargo, el resultado de generación de capas fue satisfactorio como puede verse en la Figura 5.29, al igual que su recorrido con los respectivos levantamientos de herramienta como ilustra la Figura 5.30.

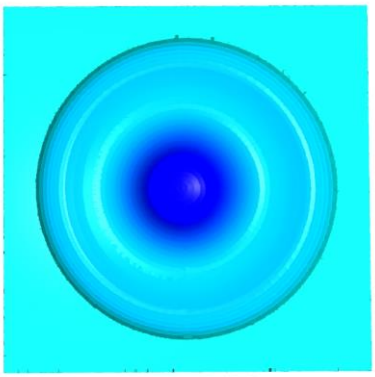


Figura 5.28 Vista superior del modelo del defecto de sombrero invertido.

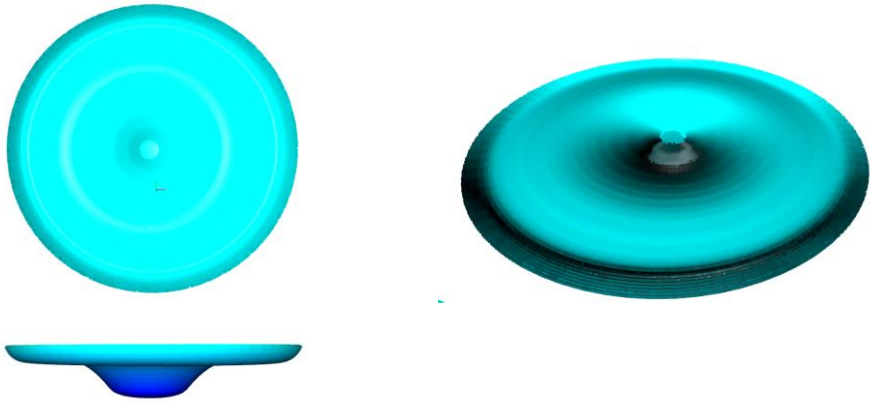


Figura 5.29 Distintas vistas de las capas generadas.

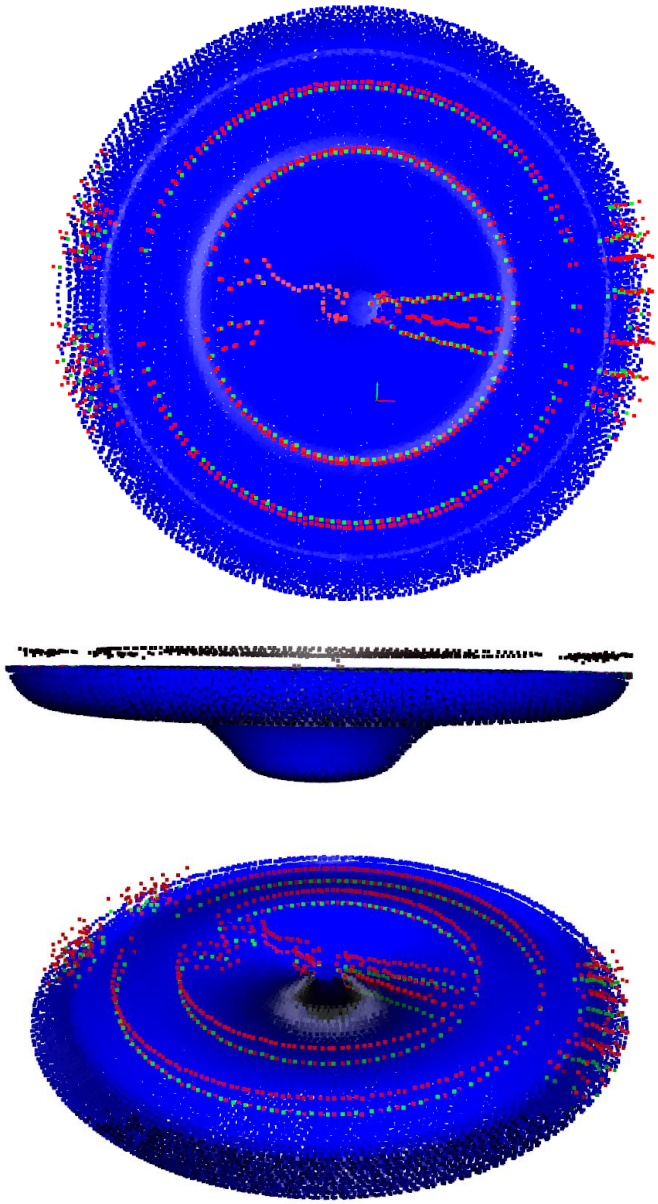


Figura 5.30 Visualización de los puntos de las trayectorias generadas para cada capa.

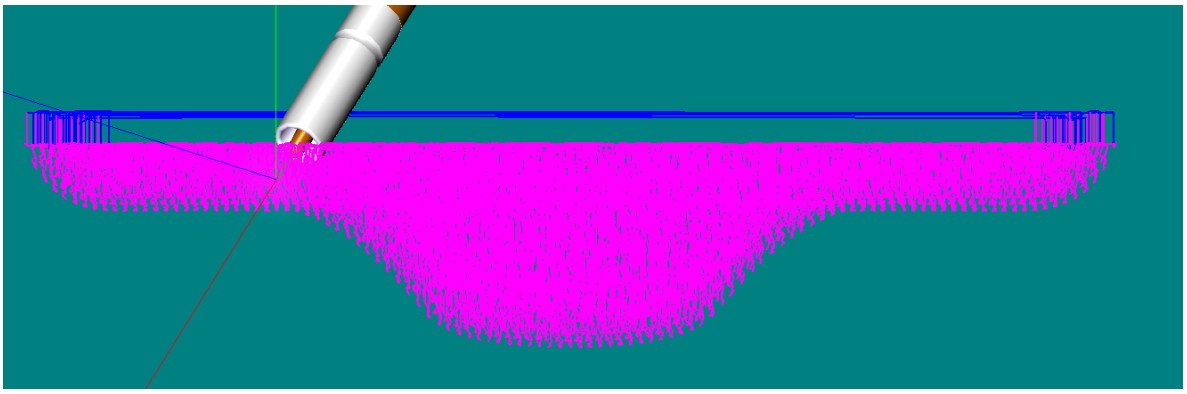


Figura 5.31 Vista lateral de la trayectoria en su visualización en DTPS.

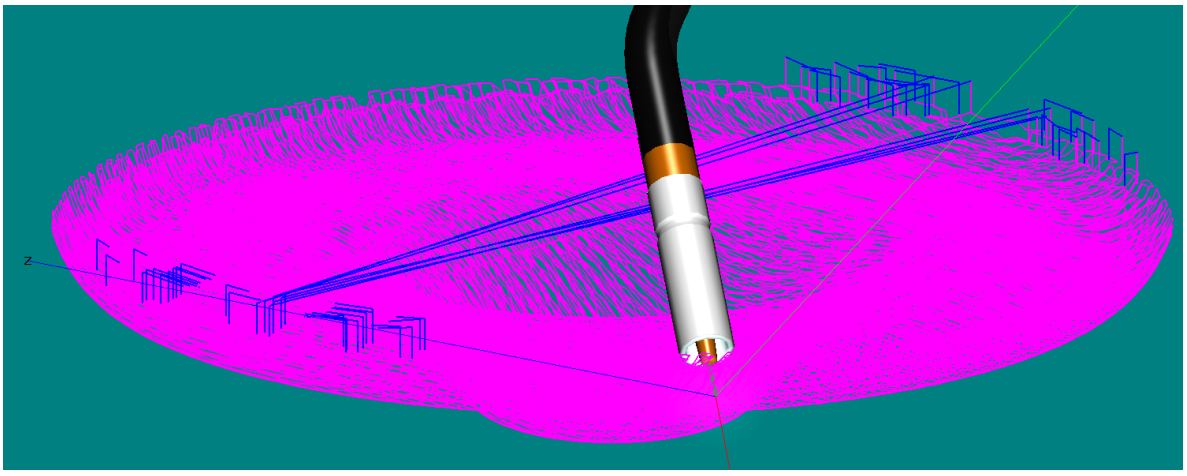
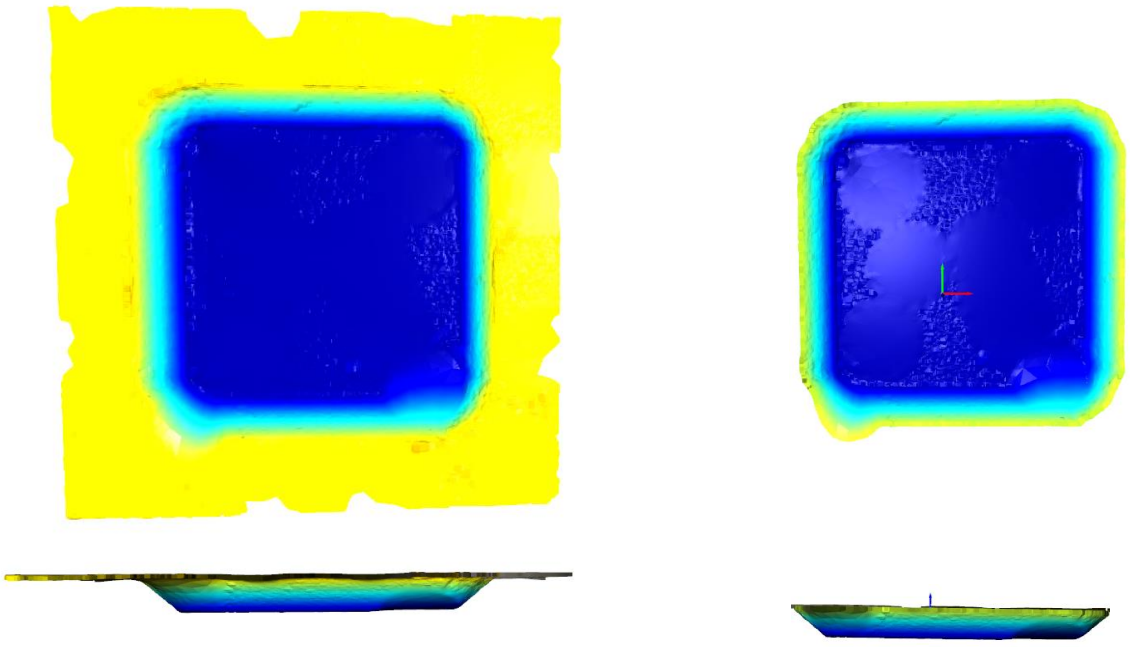


Figura 5.32 Vista de la trayectoria en su visualización en DTPS.

5.1.8 Escaneo placa bolsillo – Recorrido completo zigzag

A continuación, se ve el resultado de aplicar la estrategia a un escaneo de una placa real, la que describe la forma de un “bolsillo” de forma similar a un cuadrado. La aislación de la falla logra llevarse a cabo sin mayores problemas como indica la Figura 5.33. Posteriormente, la generación de capas para el relleno del defecto también se genera sin inconvenientes, lo cual se aprecia en la Figura 5.34.



(a) (b)

Figura 5.33 Nube de puntos del escaneo real, antes (a) y después (b) de aislar el defecto.

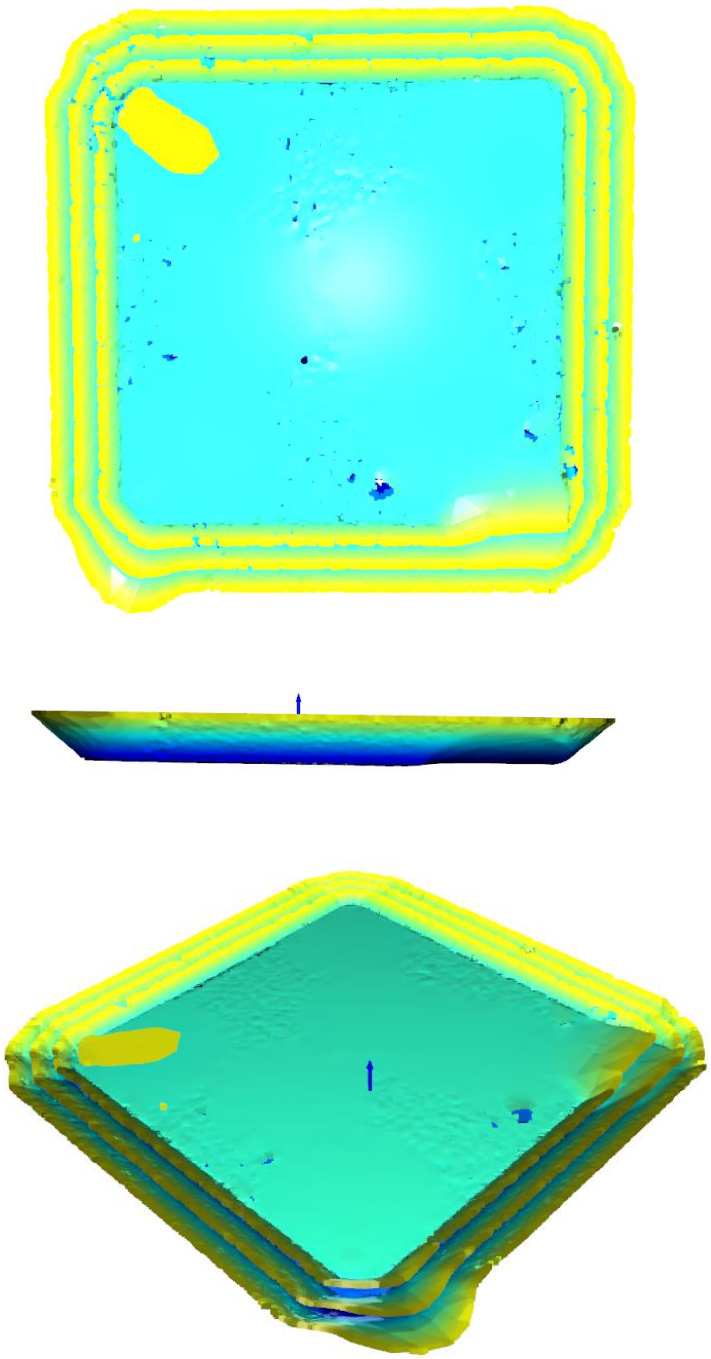


Figura 5.34 Vista de las capas generadas.

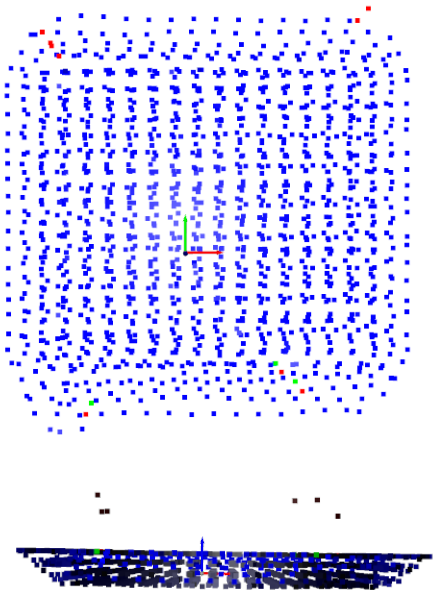


Figura 5.35 Visualización de los puntos de las trayectorias generadas en Open3d.

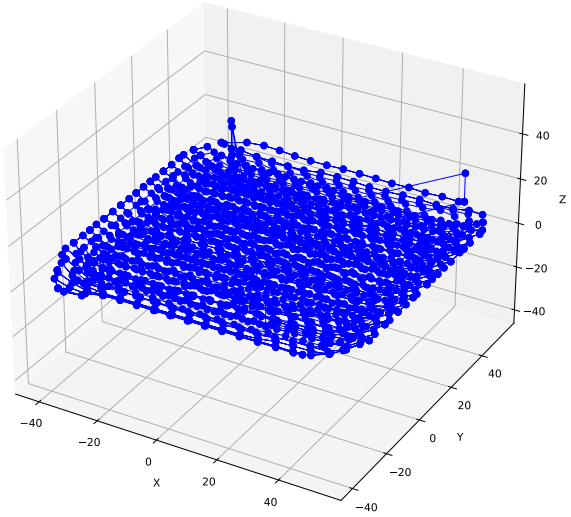


Figura 5.36 Visualización de los puntos de las trayectorias generadas en matplotlib.

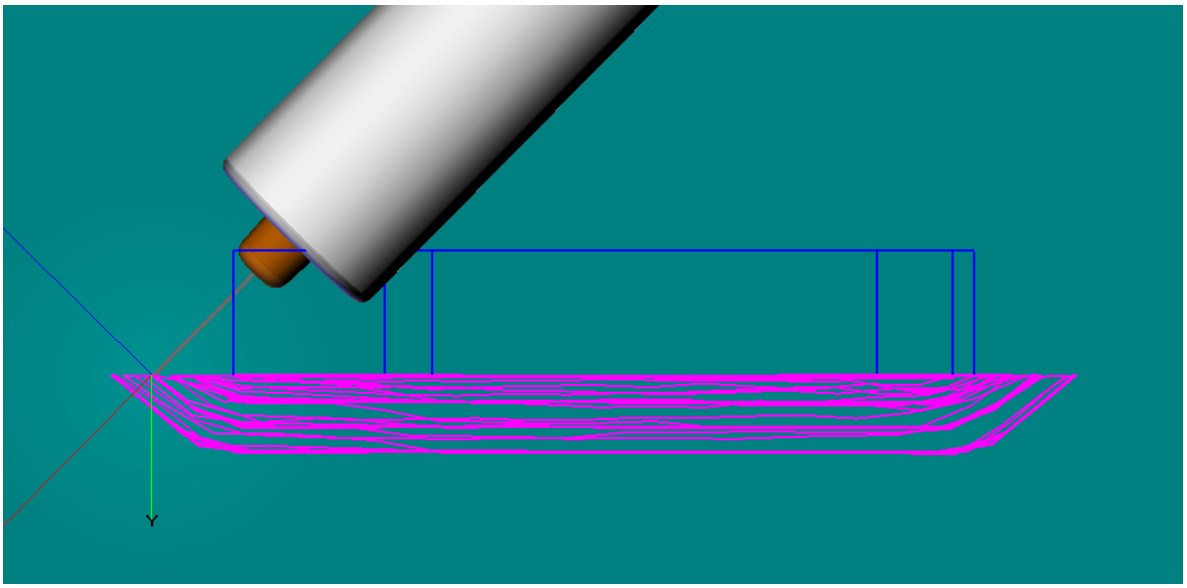


Figura 5.37 Vista lateral de la trayectoria en su visualización en DTPS.

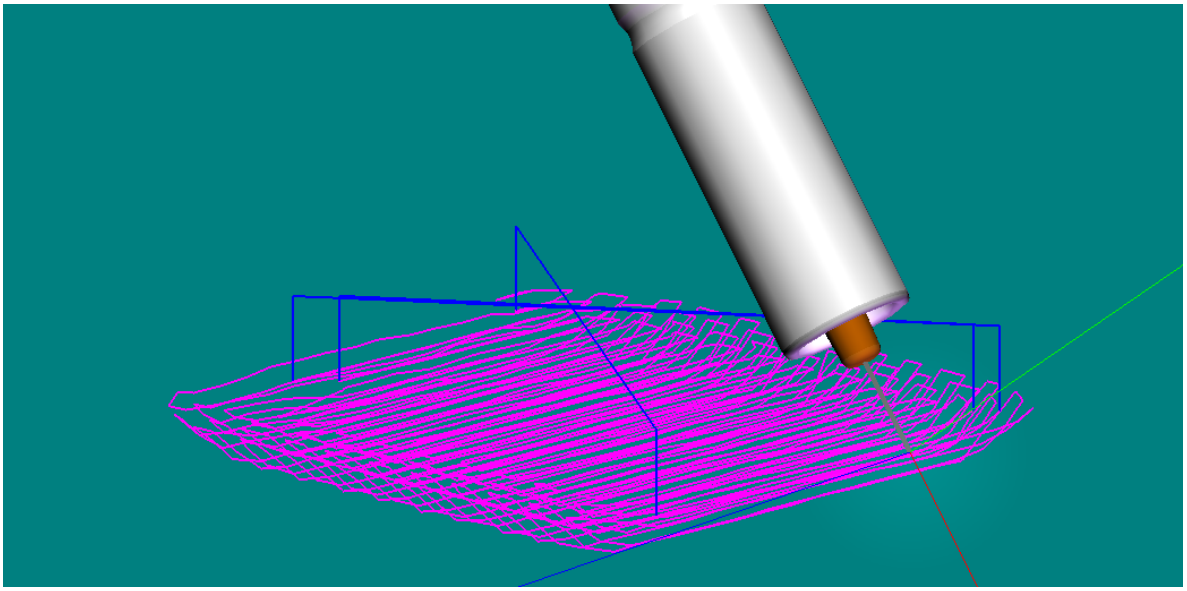


Figura 5.38 Vista de la trayectoria en su visualización en DTPS.

5.1.9 Escaneo placa bolsillo – Recorrido completo espiral

El recorrido con la técnica espiral se logra llevar sin problemas, pudiendo realizar todas las visualizaciones correspondientes. También se logra apreciar que los levantamientos de herramienta se incluyen sin inconvenientes.

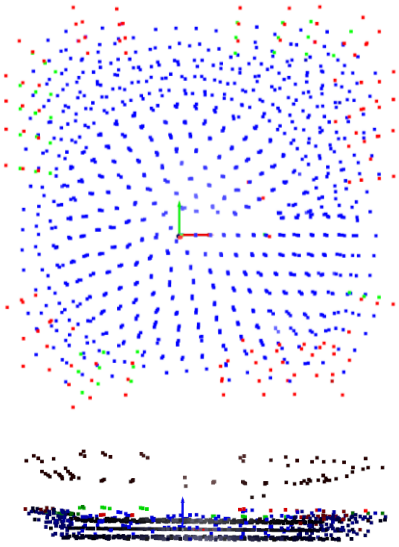


Figura 5.39 Visualización de los puntos de la trayectoria generada para las distintas capas.

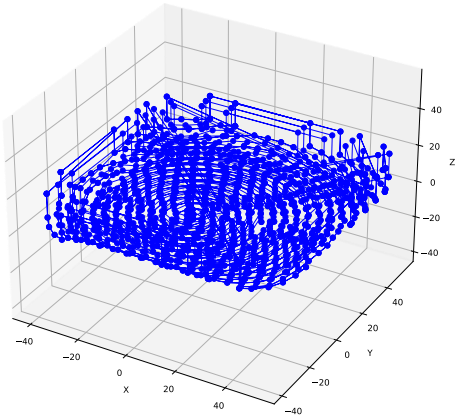


Figura 5.40 Visualización de puntos de trayectorias generadas para cada capa en matplotlib.

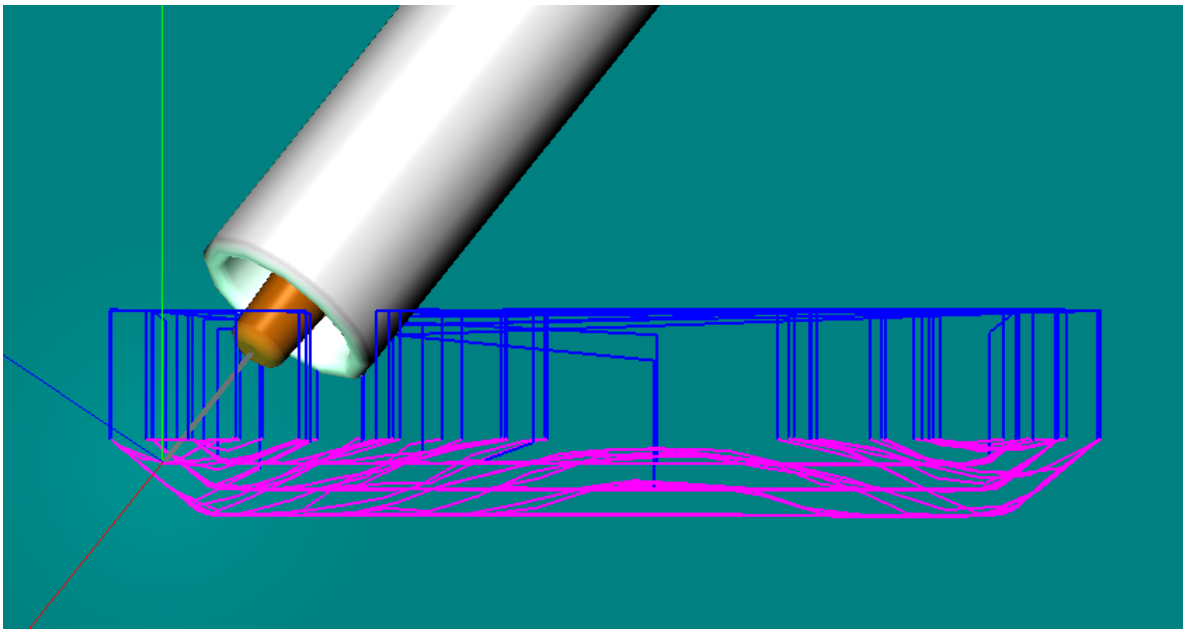


Figura 5.41 Vista lateral de la trayectoria visualizada en DTPS.

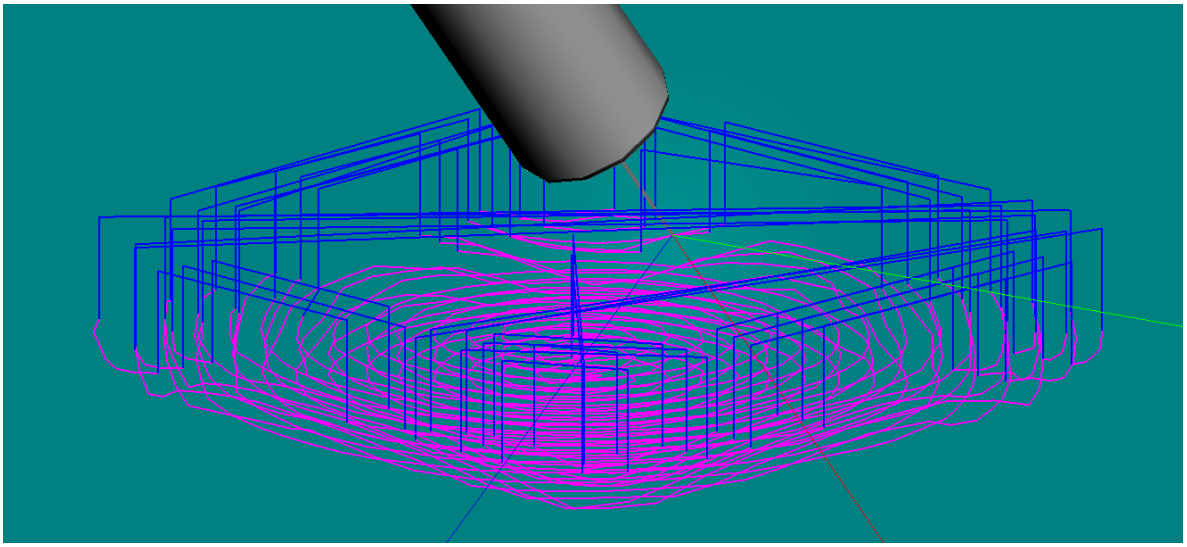


Figura 5.42 Vista de la trayectoria visualizada en DTPS.

5.1.10 Problemas en generación de trayectorias

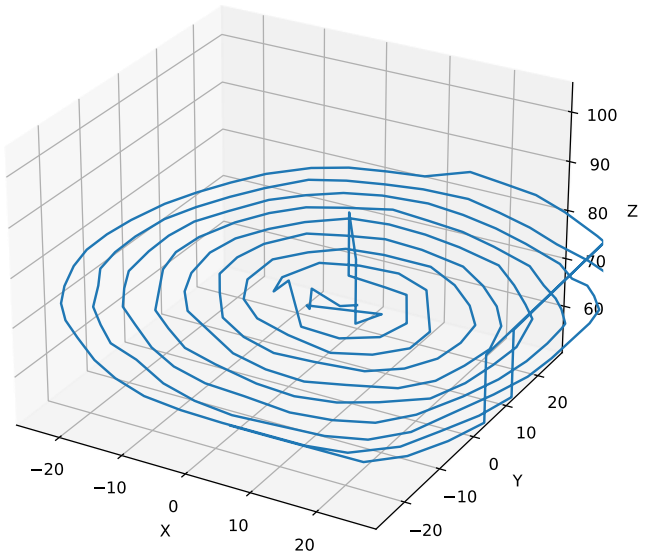


Figura 5.43 Error en la trayectoria del punto silla en recorrido espiral.

En el caso del recorrido en DTPS para la placa sombrero invertido, se produjo un error al leer el archivo generado, ya que solo se leyeron los comandos para describir 36208 puntos, de un total de 101744, evidenciando que existe una limitación en la cantidad de comandos que puede leer el programa.

Sumado a lo anterior, también se presenta un problema que ocurre muchas de las trayectorias generadas, el cual consiste en que el brazo robótico no es capaz de llegar a la posición y orientación descrita por los comandos. Cuando esto ocurre, se evidencia con un punto rojo con una cruz blanca al lado del comando, como puede apreciarse en la Figura 5.44 y con el color rojo en la articulación que tiene los problemas en su alcance, lo cual se ve en la Figura 5.44.

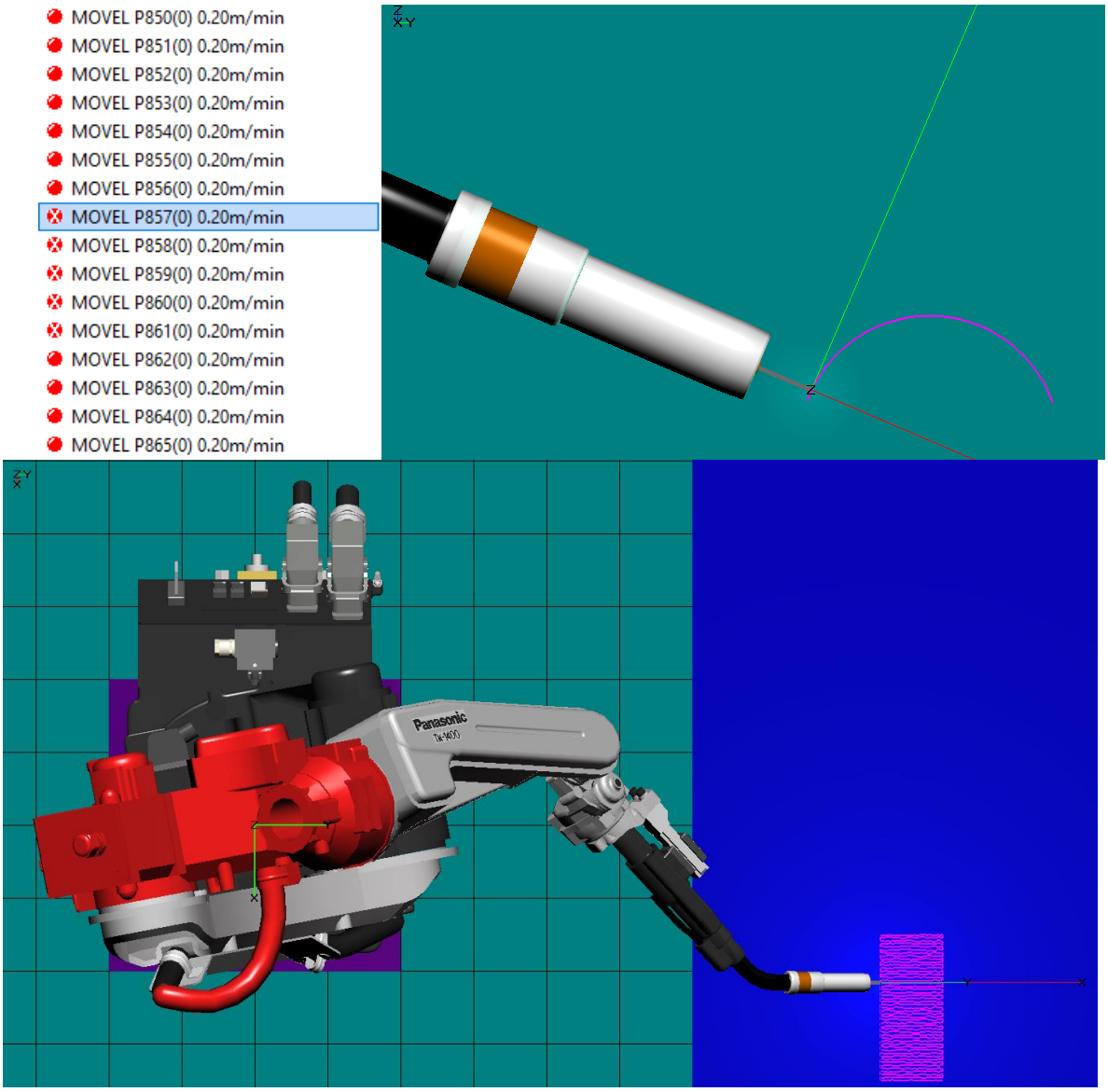


Figura 5.44 Visualización en DTPSde una trayectoria generada, donde presenta puntos fuera del alcance del robot.

Por otro lado, se comparó la simulación en DTPS de trayectorias con un ángulo de herramienta fijo respecto al robot, es decir, que no siguen la superficie de manera perpendicular. En dichos casos, se tuvo que todos los puntos de la trayectoria generada podían ser alcanzados por el robot.

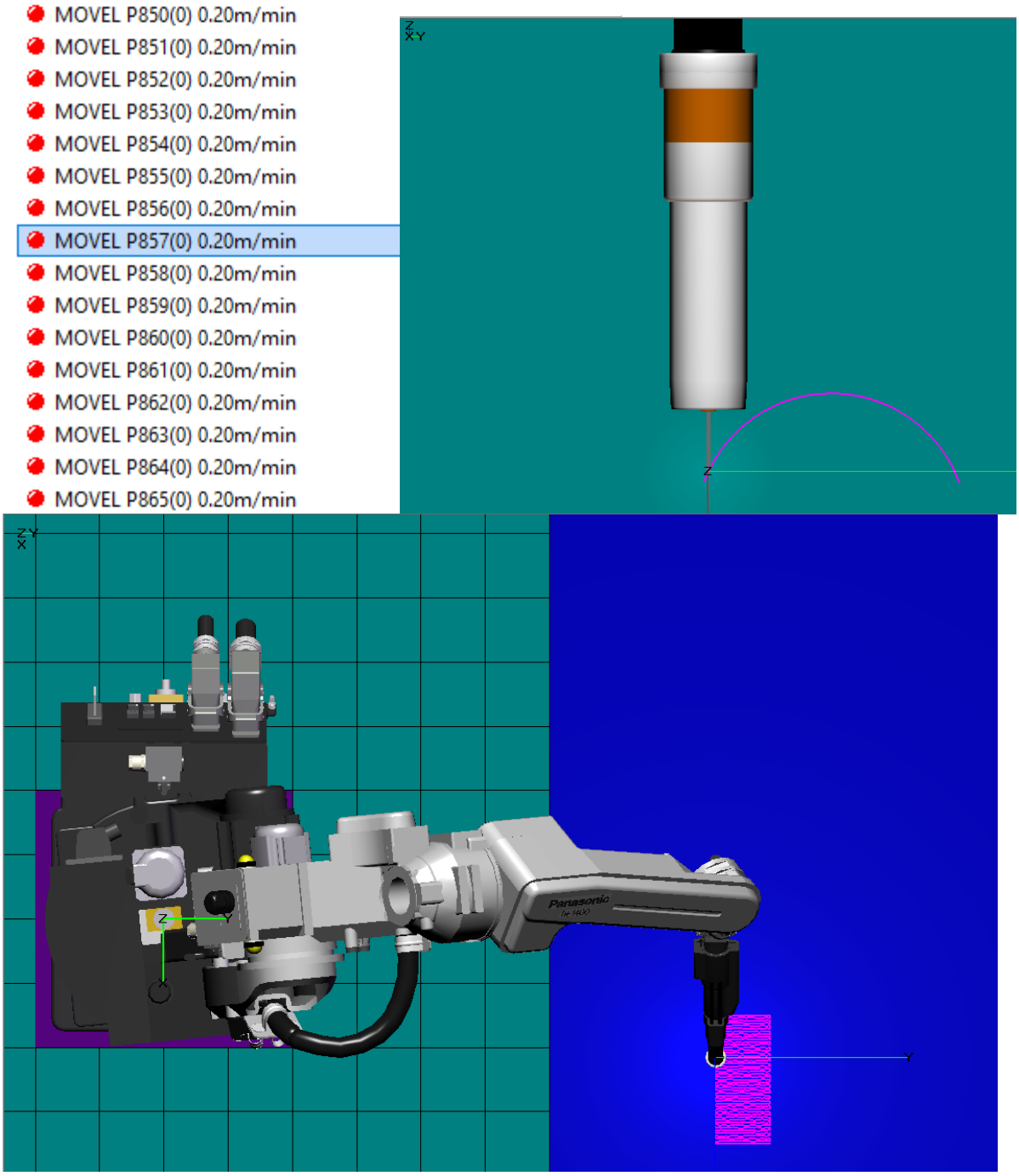


Figura 5.45 Visualización en DTPS de una trayectoria generada.

5.1.11 Generación de capas en múltiples direcciones

Como fue descrito en Generación de capas no planares, se utiliza una cámara para tomar solo los puntos que se encuentran frente a ella, permitiendo eliminar puntos internos en las nubes de puntos que no aportan a la generación de trayectorias. Para explorar otras utilidades del método, se realiza esta selección de puntos utilizando 4 cámaras distintas sobre un cilindro, ubicadas en la parte superior, inferior, lateral izquierda y lateral derecha del modelo. El objetivo de ello es sumar los puntos generados para así crear una capa que envuelve la superficie completamente. El resultado de la Figura 5.47 se produce al repetir 4 veces este proceso. Además, se incluye una cámara adicional para comparar los extremos del cilindro.

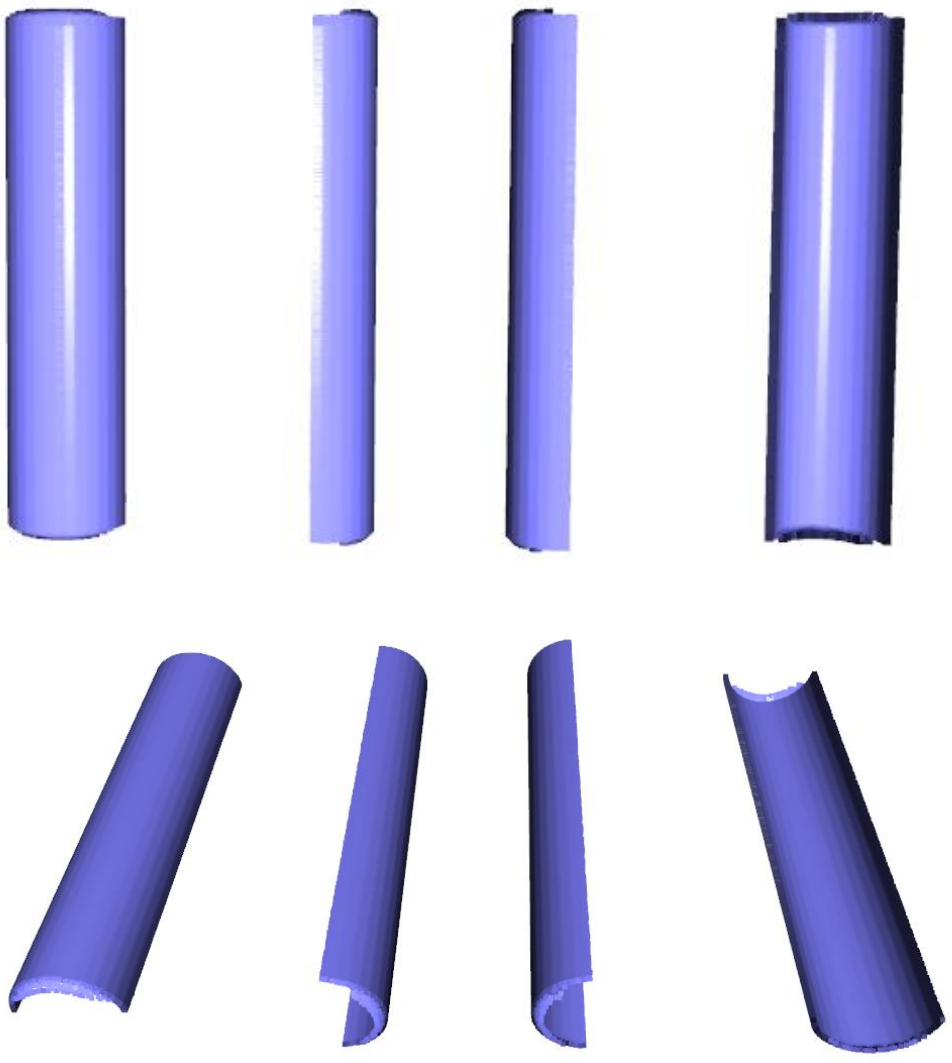


Figura 5.46 Visualización de las nubes de puntos generadas por las distintas cámaras.

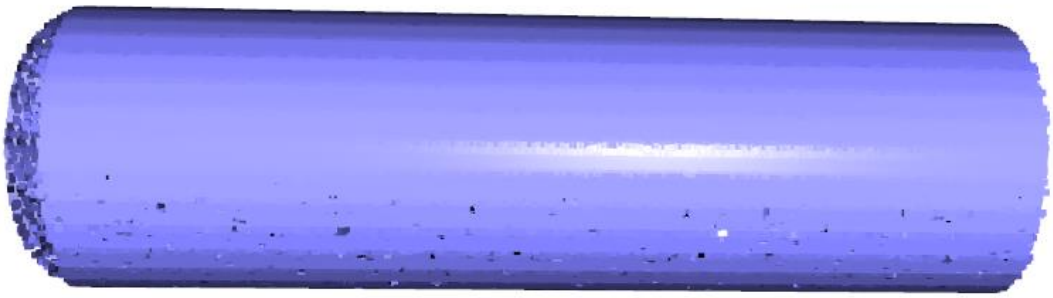
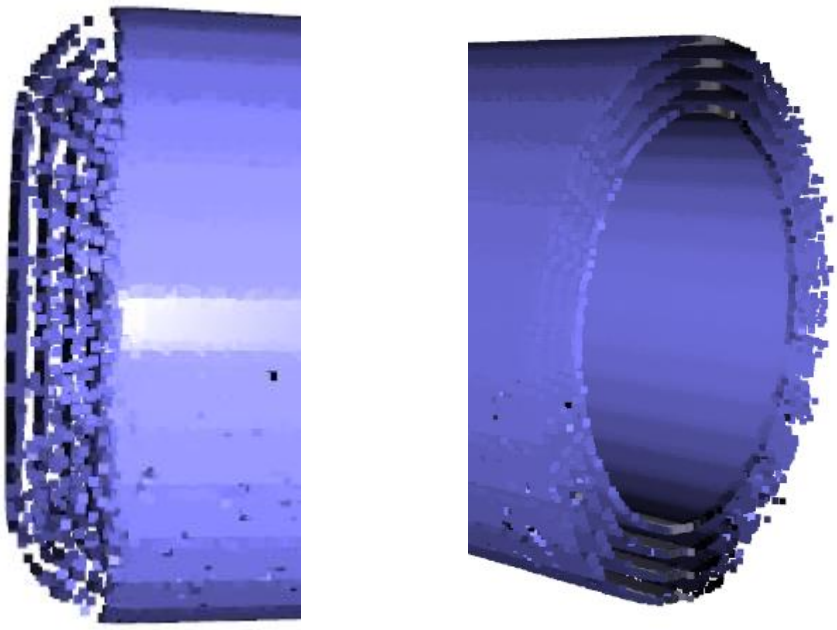


Figura 5.47 Visualización de múltiples capas generadas envolviendo el modelo.



(a) (b)

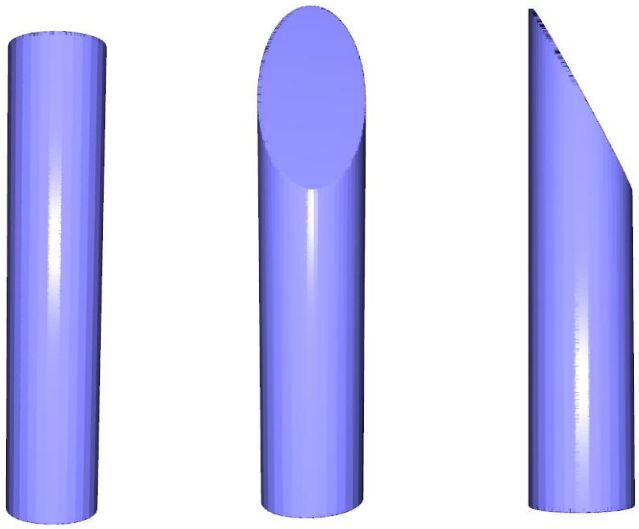
Figura 5.48 Comparación de los extremos, uno con la vista activada y el otro desactivada.

5.1.12 Relleno en volúmenes no planares

Para explorar la utilidad de la estrategia en el relleno de volúmenes no planares, se prueba en un cilindro del cual se tiene la nube de puntos de su forma original que se ve en Figura 5.49 (a), y el archivo que describe el defecto en Figura 5.49 (b) y (c).

El proceso de aislar la superficie del defecto se logra con éxito como indica la Figura 5.50. La cual se genera al restar la intersección entre la nube de puntos original y la del defecto.

Posteriormente se realiza una generación de capas en múltiples direcciones del modelo original, que se aprecia en la Figura 5.51 para así poder restar el volumen a las capas de relleno generadas, llegando las capas mostradas en la Figura 5.52.



(a) (b) (c)

Figura 5.49 Modelos de cilindro sin daño (a) y con defecto en vista frontal (b) y lateral (c).

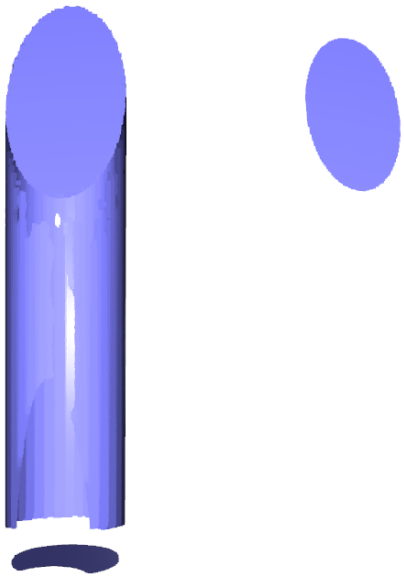


Figura 5.50 Nubes de puntos tras aislar la superficie a reparar.

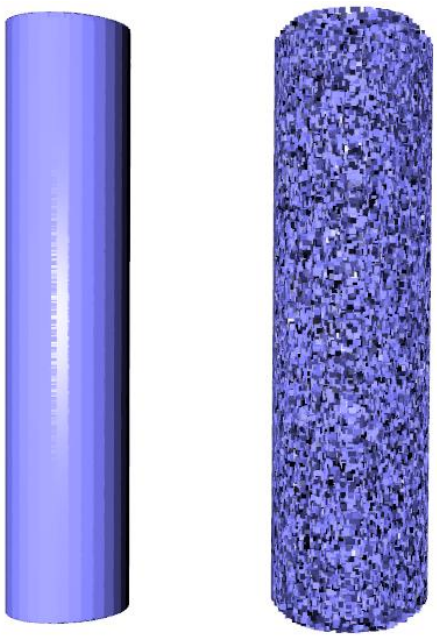
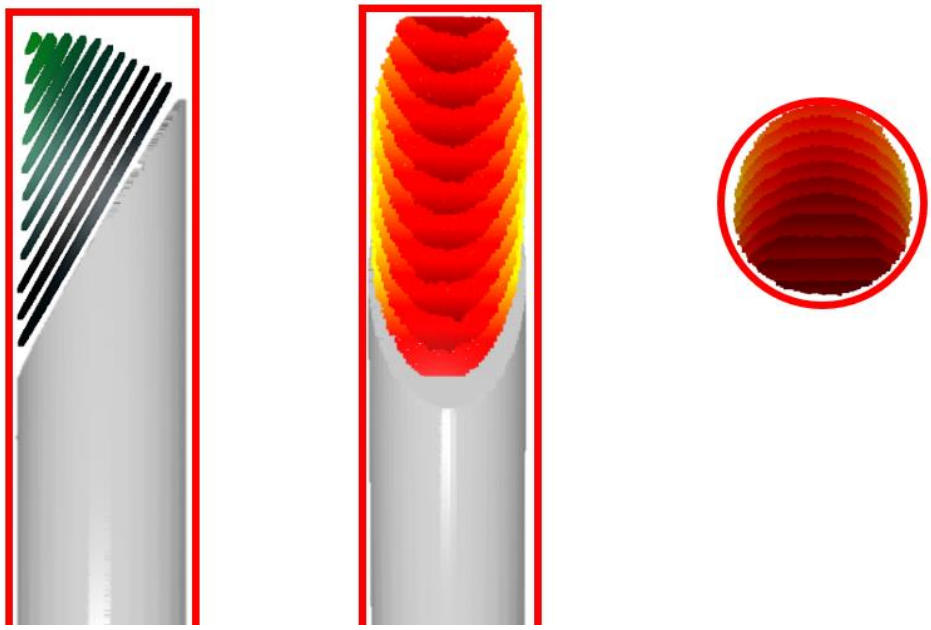


Figura 5.51 Nubes de puntos utilizadas en la resta



(a) (b) (c)

Figura 5.52 Visualización de las capas generadas sobre la superficie del defecto. Las vistas son lateral (a); frontal (b); y superior (c).

5.2 Discusión

La generación de trayectorias resulta ser satisfactoria para distintas superficies, independiente de la forma de su falla. Uno de los beneficios de trabajar con nubes de puntos es que no se requiere un volumen perfectamente cerrado para poder trabajar, a diferencia del caso de una malla.

Incluso en los escenarios donde la falla tiene una forma irregular o distintas profundidades, el algoritmo logra llegar a una solución para el relleno, sin embargo, esta no necesariamente será la mejor ruta. Por ejemplo, cuando se salta de una trayectoria a otra, el algoritmo no revisa si hay otra trayectoria cercana que pudiese ser recorrida, realizar dicho paso permitiría reducir el número de levantamientos de herramienta. No obstante, mantener esa estrategia asegura que se recorrerá toda la superficie de la capa.

Por otro lado, la generación de patrones de soldadura puede realizarse de manera semi automática, solo teniendo datos de los parámetros de soldadura e indicando el tipo de recorrido que se desea seguir.

Debido a que la estrategia se basa en la intersección entre nubes de puntos, existe una sensibilidad asociada a la densidad de puntos, por lo cual existen limitaciones al trabajar con nubes de puntos que no tengan una densidad suficiente. La densidad de la nube debe ser lo suficientemente alta como para permitir la intersección con las nubes de puntos en el paso 4.1 de dividir la capa en un conjunto de nubes. Así mismo, la densidad de la proyección de la nube también debe ser suficiente para generar dicha intersección.

Otras limitantes están asociadas al programa del robot, como lo son la cantidad de comandos aceptados por el programa, evitando que se puedan cargar directamente comandos para rellenar superficies de mayores dimensiones, como se demostró en el caso de la placa sombrero invertido. Lo anterior podría solucionarse procesando los caminos para quitar puntos que entreguen información redundante o generando los rellenos por etapas. De la misma forma, existe un problema asociado a los puntos donde el robot no logra ubicar la herramienta en el espacio y orientación indicados, sin embargo, la predicción de dichos puntos se escapa a los alcances de este trabajo, ya que son condiciones que dependen del robot mismo. La reubicación de la pieza en otro punto puede permitir que el robot logre llegar a puntos que antes no podía, así mismo la reorientación de la pieza provoca el mismo efecto, por último, el limitar la orientación de la herramienta para no seguir la superficie y mantenerse con un ángulo fijo, mostró tener una mayor cantidad de puntos alcanzables, tal como se vio en la comparación entre la Figura 5.44 y la Figura 5.45

En el caso de tener un modelo CAD de la forma final de la pieza, podría ser posible explorar otras estrategias, como por ejemplo realizando contornos paralelos tridimensionales sobre la superficie de la pieza y hacia el interior en vez de la superficie del defecto y hacia el exterior, de manera similar a lo realizado en Combinación de capas planas y curvas, con la diferencia de que las capas internas seguirían la superficie del defecto en vez de ser planas. Con ello queda en

evidencia que esta estrategia abre las puertas para otro tipo de aproximaciones en el relleno de volúmenes, rompiendo los paradigmas del 2.5 D.

Como puede observarse en la generación de capas en múltiples direcciones, es posible utilizar esta técnica en más de una dirección, lo cual permitiría cubrir la reparación de piezas que su superficie principal sea no planar. Además, la utilización de capas en múltiples direcciones se puede utilizar como una herramienta para eliminar puntos en dichas geometrías, complementando la eliminación de puntos utilizando una bounding box con la intersección con las capas generadas, tal como puede apreciarse en el relleno de volúmenes no planares.

Los tiempos que tarda el robot en reorientar la normal no fueron considerados en este trabajo, por lo que mantener la soldadura activada en estas operaciones podría generar una deposición de material superior a la deseada en ciertos puntos. El programa DTPS es el que termina definiendo los giros de cada articulación del robot, por lo que no es posible predecir en qué puntos sucederán estas reorientaciones de tiempo considerable, aunque se estima que estas ocurran entre puntos que tengan una diferencia significativa en la orientación de sus normales, no es posible asegurarlo del todo sin conocer los algoritmos que sigue el programa para llegar a los puntos.

El supuesto de que la geometría del cordón se mantendrá constante independiente de la orientación de la normal de la superficie es uno bastante fuerte, sin embargo, para resolver si es aceptable se hubiese requerido estudiarlo de forma específica y realizar pruebas experimentales para poder validarlo. Estas pruebas se ven dificultadas para ser realizadas en el contexto de pandemia que se vive mundialmente.

La colisión de las trayectorias con las partes ya soldadas no fue verificada, sin embargo, podrían realizarse siguiendo las indicaciones del método de generación de caminos completamente tridimensional [15], agregando también la orientación de la boquilla.

Esta estrategia destaca en defectos donde el desgaste es uniforme, ya que permite el relleno de este con un número menor de capas al que se tendría si se realizara una estrategia de 2.5D. Así mismo, si la superficie del defecto está alineada con la superficie de la pieza, podría significar tener una pieza final con un mejor acabado superficial.

Una alternativa al proceso de recorrer las capas pudo haber sido transformar la capa a un espacio 2D para luego ser recorrida con métodos para capas planes típicos, y posteriormente transformar esa geometría devuelta al espacio 3D. Esta opción no fue escogida debido a que se abordó el problema sin la utilización de formatos *mesh*, aun así, es una alternativa que queda por explorar, la cual tiene la ventaja de heredar los conocimientos de los recorridos 2D.

La implementación de un sistema que permita rellenar las capas con estrategias distintas para volúmenes que tengan una geometría similar podría ser de ayuda en casos donde la falla tiene una forma más regular, sin embargo, se consideró que las fallas tendrían formas irregulares en la mayoría de las veces, por lo cual esta opción no fue explorada.

Finalmente, la estrategia al consistir en la generación de caminos, si bien está orientada al relleno de defectos, podría utilizarse en otros tipos de aplicaciones, debido a su capacidad de seguir la superficie de la pieza. Algunas de estas aplicaciones pueden ser el recubrimiento de piezas con aleaciones de metal, por distintos métodos, como por ejemplo por proyección térmica. Por otro lado, podría utilizarse en otro tipo de manufactura aditiva, pudiendo extruir material en superficies no planas, lo que sería útil para reducir la utilización de material de soporte o depositar material sobre otras piezas.

6 Propuesta de trabajo y conclusión

6.1 Conclusión

Como fue visto en antecedentes, la industria se vería fuertemente beneficiada con la recuperación de piezas de metal, por lo que se investigan nuevas formas que permitan realizar el proceso de forma segura, versátil, veloz y efectiva. Es con el afán de aportar en estas direcciones en que el trabajo explora un nuevo método de recuperación de piezas, utilizando trayectorias de forma libre, generadas de forma semi automática.

Se logra implementar la generación de patrones de soldadura utilizando caminos de forma libre con una estrategia nueva. Esto se llevó a cabo al rellenar el volumen del defecto con capas generadas sobre la superficie de este, y posteriormente recorriendo dichas capas con el método elegido por el usuario, zigzag o espiral.

Al generar distintas geometrías fue posible el análisis de la generación de patrones de soldadura en distintos tipos de defectos. Debido a lo anterior se puede concluir que esta estrategia logra tener robustez ante las posibles distintas formas de defectos en piezas de acero, destacando en piezas con profundidades variables. Por otro lado, existen limitaciones en escenarios donde la cantidad de comandos supera la memoria que puede recibir el programa DTPS. Además de ello, se tienen las restricciones propias del alcance del brazo robótico, para llegar al punto indicado por la trayectoria y con la orientación indicada por la normal asociada al punto.

Por otro lado, la utilización del formato PCD para el camino de soldadura permite una fácil traducción al lenguaje de robot, al reunir las principales características para definir su trayectoria, por lo que su generación puede realizarse de forma semiautomática, indicando los parámetros iniciales correspondientes. Inclusive el usuario podría modificar la orientación o ubicación de la trayectoria fácilmente utilizando las funciones para archivos PCD.

Finalmente, y aún con las limitaciones mencionadas, el trabajo realizado se presenta como una alternativa viable a la hora de reparar piezas de metal, debido a que puede adaptarse a distintas formas, reduciendo los tiempos de programación de la herramienta de soldadura, lo que abre las puertas a reparar una mayor variedad de piezas, con geometrías más complejas. Por otra parte, se posiciona como una alternativa de manufactura aditiva completamente en 3D y no en 2.5D.

6.2 Propuestas de trabajo

Se propone investigar la geometría de los cordones de soldadura para los casos en que la orientación del plano a soldar no coincide con el plano horizontal, para verificar la efectividad del supuesto de mantener la geometría de parábola para estos casos.

La colisión de la herramienta con los caminos generados no fue tomada en cuenta, por lo que realizar una verificación utilizando el método mencionado en la sección 3 podría ser una alternativa que permita descartar caminos antes de cargar la trayectoria en el programa DTPS.

La estrategia propuesta se basó en tener un archivo en formato PCD con la información de la pieza, sin embargo, podría desarrollarse de manera similar una estrategia utilizando el formato tipo *Mesh* o mallado, además de tener posibles ventajas en el proceso al utilizar métodos especializados para este tipo de formato.

Por último, explorar la utilización de la estrategia en otros tipos de aplicaciones, como lo son el recubrimiento de piezas y la manufactura aditiva sobre superficies no planares.

7 Glosario

2D	2 dimensiones
3D	3 dimensiones
AM	Manufactura aditiva (Additive manufacturing)
CAD	Diseño asistido por computadora (Computer aided design)
CAM	Manufactura asistida por computadora (Computer aided manufacturing)
FDM	Modelado por deposición fundida (Fused deposition modeling)
CNC	Control numérico computarizado (Computer numerical control)
Voxel	Unidad elemental de una representación en el espacio 3D, similar al pixel para el caso 2D
CLFDM	Modelado por deposición fundida con capas curvas (Curved layered fused deposition modeling)
WAAM	Manufactura aditiva por alambre de soldadura (Wire arc additive manufacturing)
CTWD	Distancia de contacto entre punta y pieza de trabajo (Contact tip to work distance)

8 Bibliografía

[1] J. Jiang and Y. Ma, "Path planning strategies to optimize accuracy, quality, build time and material use in additive manufacturing: A review," *Micromachines*, vol. 11, no. 7, 2020, doi: 10.3390/MI11070633.

[2] T. Llewellyn-Jones, R. Allen, and R. Trask, "Curved Layer Fused Filament Fabrication Using Automated Toolpath Generation," *3D Printing and Additive Manufacturing*, vol. 3, no. 4, pp. 236–243, 2016, doi: 10.1089/3dp.2016.0033.

[3] L. Nguyen, J. Buhl, and M. Bambach, "Multi-bead overlapping models for tool path generation in wire-arc additive manufacturing processes," *Procedia Manufacturing*, vol. 47, no. 2019, pp. 1123–1128, 2020, doi: 10.1016/j.promfg.2020.04.129.

[4] S.C. Park and B.K. Choi, "Tool-path planning for direction-parallel area milling," *Computer-Aided Design*, vol. 32, pp. 17–25, 2000, doi: 10.1007/s00170-010-2909-7.

[5] V. T. Rajan, V. Srinivasan, and K. A. Tarabanis, "The optimal zigzag direction for filling a two-dimensional region," *Rapid Prototyping Journal*, vol. 7, no. 5, pp. 231–241, 2001, doi: 10.1108/13552540110410431.

[6] R. T. Farouki, T. Koenig, K. A. Tarabanis, J. U. Korein, and J. S. Batchelder, "Path planning with offset curves for layered fabrication processes," *Journal of Manufacturing Systems*, vol. 14, no. 5, pp. 355–368, 1995, doi: 10.1016/0278-6125(95)98872-4.

[7] H. Li, Z. Dong, and G. W. Vickers, "Optimal toolpath pattern identification for single island, sculptured part rough machining using fuzzy pattern analysis," *Computer-Aided Design*, vol. 26, no. 11, pp. 787–795, 1994, doi: 10.1016/0010-4485(94)90092-2.

[8] H. Wang, P. Jang, and J. A. Stori, "A metric-based approach to two-dimensional (2D) tool-path optimization for high-speed machining," *Journal of Manufacturing Science and Engineering, Transactions of the ASME*, vol. 127, no. 1, pp. 33–48, 2005, doi: 10.1115/1.1830492.

[9] F. Ren, Y. Sun, and D. Guo, "Combined reparameterization-based spiral toolpath generation for five-axis sculptured surface machining," *International Journal of Advanced Manufacturing Technology*, vol. 40, no. 7–8, pp. 760–768, 2009, doi: 10.1007/s00170-008-1385-9.

[10] D. Ding, Z. Pan, D. Cuiuri, and H. Li, “A tool-path generation strategy for wire and arc additive manufacturing,” *International Journal of Advanced Manufacturing Technology*, vol. 73, no. 1–4, pp. 173–183, 2014, doi: 10.1007/s00170-014-5808-5.

[11] M. Bertoldi, M. a Yardimci, C. M. Pistor, and S. I. Giiveri, “Domain Decomposition and Space Filling Curves in Toolpath Planning and Generation,” *Proceeeding of the 1998 Solid Fabrication Symposium*, pp. 267–274, 1998.

[12] Y. M. Zhang, Y. Chen, P. Li, and A. T. Male, “Weld deposition-based rapid prototyping: A preliminary study,” *Journal of Materials Processing Technology*, vol. 135, no. 2-3 SPEC., pp. 347–357, 2003, doi: 10.1016/S0924-0136(02)00867-1.

[13] G. Q. Jin, W. D. Li, and L. Gao, “An adaptive process planning approach of rapid prototyping and manufacturing,” *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 1, pp. 23–38, 2013, doi: 10.1016/j.rcim.2012.07.001.

[14] D. Ding, Z. Pan, D. Cuiuri, and H. Li, “A practical path planning methodology for wire and arc additive manufacturing of thin-walled structures,” *Robotics and Computer-Integrated Manufacturing*, vol. 34, pp. 8–19, 2015, doi: 10.1016/j.rcim.2015.01.003.

[15] M. K. Micali and D. Dornfeld, “Fully three-dimensional toolpath generation for point-based additive manufacturing systems,” *Solid Freeform Fabrication 2016: Proceedings of the 27th Annual International Solid Freeform Fabrication Symposium - An Additive Manufacturing Conference, SFF 2016*, pp. 36–52, 2016.

[16] D. Ding, Z. Pan, D. Cuiuri, H. Li, N. Larkin, and S. van Duin, “Multi-direction slicing of STL models for robotic wire-feed additive manufacturing,” *Proceedings - 26th Annual International Solid Freeform Fabrication Symposium - An Additive Manufacturing Conference, SFF 2015*, no. August, pp. 1059–1069, 2015.

[17] D. Ahlers, F. Wasserfall, N. Hendrich, and J. Zhang, “3D printing of nonplanar layers for smooth surface generation,” *IEEE International Conference on Automation Science and Engineering*, vol. 2019-Augus, pp. 1737–1743, 2019, doi: 10.1109/COASE.2019.8843116.

[18] J. B. Khurana, S. Dinda, and T. W. Simpson, “Active - Z printing: A new approach to increasing3D printed part strength,” *Solid Freeform Fabrication 2017: Proceedings of the 28th Annual International Solid Freeform Fabrication Symposium - An Additive Manufacturing Conference, SFF 2017*, pp. 1627–1644, 2017.

[19] J. R. Kubalak, A. L. Wicks, and C. B. Williams, “Using multi-axis material extrusion to improve mechanical properties through surface reinforcement,” *Virtual and Physical Prototyping*, vol. 13, no. 1, pp. 32–38, 2017, doi: 10.1080/17452759.2017.1392686.

[20] D. Chakraborty, B. Aneesh Reddy, and A. Roy Choudhury, “Extruder path generation for Curved Layer Fused Deposition Modeling,” *CAD Computer Aided Design*, vol. 40, no. 2, pp. 235–243, 2008, doi: 10.1016/j.cad.2007.10.014.

[21] B. Huang and S. Singamneni, “A mixed-layer approach combining both flat and curved layer slicing for fused deposition modelling,” *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 229, no. 12, pp. 2238–2249, 2015, doi: 10.1177/0954405414551076.

[22] Y. Jin, J. Du, Y. He, and G. Fu, “Modeling and process planning for curved layer fused deposition,” *International Journal of Advanced Manufacturing Technology*, vol. 91, no. 1–4, pp. 273–285, 2017, doi: 10.1007/s00170-016-9743-5.

[23] J. Nassan, “Gas Metal Arc Welding: Process Overview,” p. 96, 2003, [Online]. Available: https://m.lincolnelectric.com/assets/global/Products/Consumable_MIGGMAWWires-SuperArc-SuperArcLA-90/c4200.pdf

[24] Z. Hu, X. Qin, T. Shao, and H. Liu, “Understanding and overcoming of abnormity at start and end of the weld bead in additive manufacturing with GMAW,” *International Journal of Advanced Manufacturing Technology*, vol. 95, no. 5–8, pp. 2357–2368, 2018, doi: 10.1007/s00170-017-1392-9.

[25] J. Xiong, G. Zhang, H. Gao, and L. Wu, “Modeling of bead section profile and overlapping beads with experimental validation for robotic GMAW-based rapid manufacturing,” *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 2, pp. 417–423, 2013, doi: 10.1016/j.rcim.2012.09.011.

[26] S. Suryakumar, K. P. Karunakaran, A. Bernard, U. Chandrasekhar, N. Raghavender, and D. Sharma, “Weld bead modeling and process optimization in Hybrid Layered Manufacturing,” *CAD Computer Aided Design*, vol. 43, no. 4, pp. 331–344, 2011, doi: 10.1016/j.cad.2011.01.006.

[27] X. Li, Q. Han, and G. Zhang, “Large-size sprocket repairing based on robotic GMAW additive manufacturing,” *Welding in the World*, pp. 793–805, 2021, doi: 10.1007/s40194-021-01080-9.

[28] D. Ding, Z. Pan, D. Cuiuri, H. Li, S. van Duin, and N. Larkin, “Bead modelling and implementation of adaptive MAT path in wire and arc additive manufacturing,” *Robotics and Computer-Integrated Manufacturing*, vol. 39, pp. 32–42, 2016, doi: 10.1016/j.rcim.2015.12.004.

[29] Sandiman, “Posicionador,” 2020. https://sandiman.cl/producto/posicionador-ya-1rjb32/ (accessed Jul. 25, 2020).

[30] Python, “Python,” 2020. https://www.python.org/about/ (accessed Mar. 10, 2021).

[31] D. Girardeau-Montaut, “CloudCompare - User manual,” *Webpage: http://www.cloudcompare.org*, 2015.

[32] Open3d, “Open3D,” 2021. http://www.open3d.org/docs/release/introduction.html (accessed Mar. 10, 2021).

[33] Panasonic, “Operating Instructions Integrated PC Tool Software YA-1NPCD1 Series,” p. 244, 2002.

9 Anexos

9.1 Anexo A – Datos empíricos de la geometría del cordón de soldadura

Tabla 9.1 Parámetros de soldadura según amperaje y velocidad de herramienta.

Velocidad [m/min]		0,2		0,3		0,4		0,5		0,6	
Amperaje	Voltaje	Alto	Ancho	Alto	Ancho	Alto	Ancho	Alto	Ancho	Alto	Ancho
90	16,4	2,4	5,2	2,3	4,6	2,2	4,0	2,1	3,4	2,0	2,8
100	16,6	2,7	5,6	2,5	5	2,3	4,4	2,2	3,8	2,0	3,2
110	16,7	2,8	6,2	2,6	5,5	2,4	4,8	2,2	4,1	2,1	3,4
120	17	2,7	6,8	2,6	6	2,5	5,2	2,3	4,5	2,2	3,7
130	17,4	3,0	7,3	2,8	6,5	2,6	5,7	2,4	4,8	2,2	4,0
140	17,9	3,3	7,6	3	6,8	2,8	6,0	2,5	5,2	2,3	4,4
150	18,3	3,4	8,9	3,1	7,7	2,8	6,6	2,5	5,4	2,2	4,3
160	18,8	3,5	9,0	3,2	8,0	3,0	7,0	2,7	6	2,4	5,0
170	19,1	3,6	9,6	3,3	8,5	3,1	7,4	2,8	6,3	2,5	5,2
180	19,3	3,8	10,2	3,5	9,0	3,2	7,8	2,9	6,6	2,6	5,4
190	19,4	4,0	10,8	3,6	9,5	3,3	8,1	2,9	6,8	2,6	5,5
200	19,6	4,0	11,4	3,7	10,0	3,4	8,6	3,1	7,2	2,8	5,8
210	19,8	4,3	11,9	3,9	10,5	3,4	9,1	3	7,7	2,6	6,3

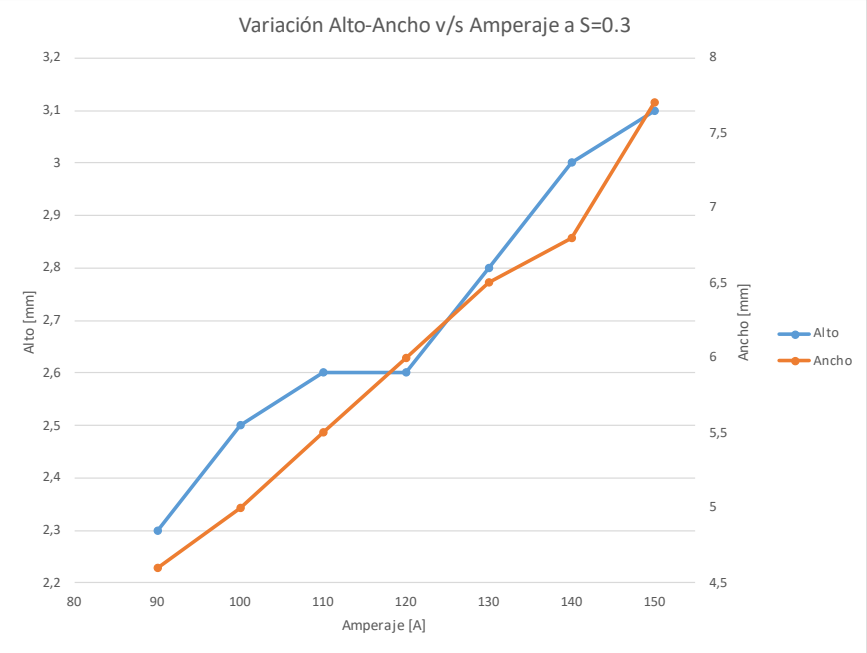


Figura 9.1 Variación alto-ancho vs amperaje a S=0.3.

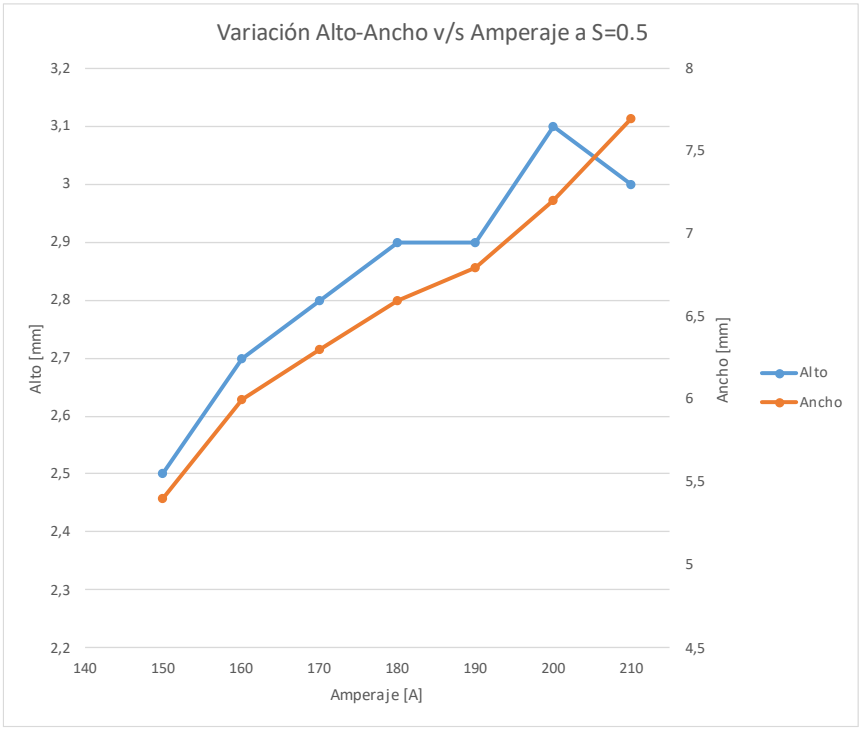


Figura 9.2 Variación alto-ancho vs amperaje a S=0.5.

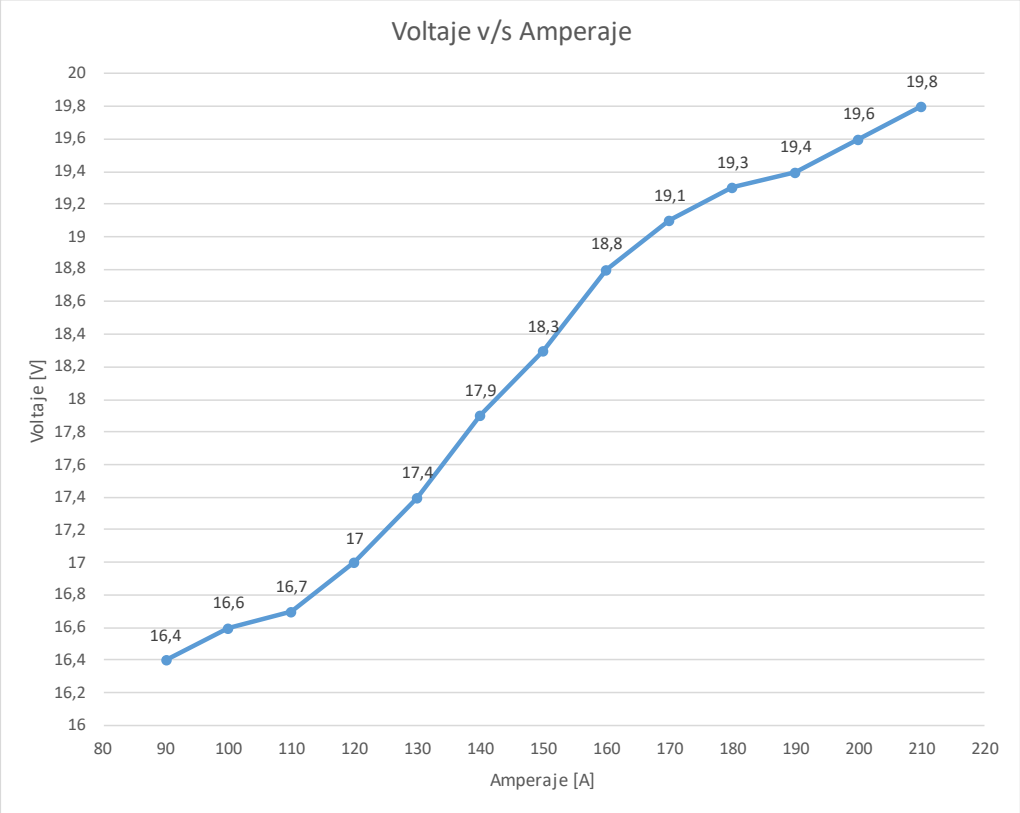


Figura 9.3 Voltaje vs amperaje.

Tabla 9.2 Estimaciones lineales de parámetros del cordón de soldadura.

Estimaciones Lineales [y=mx+b]			
Voltaje	m/b	0,03153846	13,44615385
	Error m/b	0,00151582	0,23433968
	S=0.3		
Alto	m/b	0,01285714	1,15714286
	Error m/b	0,00110657	0,13461965
Ancho	m/b	0,04964286	0,05714286
	Error m/b	0,00251018	0,30537698
	S=0.5		
Alto	m/b	0,00857143	1,30000000
	Error m/b	0,00149830	0,27135337
Ancho	m/b	0,03500000	0,27142857
	Error m/b	0,00214286	0,38808793
	S=0.4		
Alto	m/b	0,01071429	1,228571429
	Error m/b	0,00023999	0,037101639
Ancho	m/b	0,04232143	0,164285714
	Error m/b	0,00036175	0,0559254
	S=0.2		
Alto	m/b	0,0150	1,085714286
	Error m/b	0,000491	0,075907458
Ancho	m/b	0,05696429	-0,0500
	Error m/b	0,00111822	0,172872049
	S=0.6		
Alto	m/b	0,00642857	1,371428571
	Error m/b	0,00061952	0,095774824
Ancho	m/b	0,02767857	0,378571429
	Error m/b	0,00100566	0,155471959

9.2 Anexo B - Código

9.2.1 Código principal

```
import open3d as o3d
import numpy as np
import math
from scipy.spatial.transform import Rotation as R
import matplotlib.pyplot as plt
import copy
import freeformpathplanning as fp

# Parámetros iniciales-----------------------------------------------------------
----------------------------------------
# -------------------------------------------------------------------------------
----------------------------------------

ampere= 100 #Valor del ampearaje a utilizar
s=0.2 #Valor de la velocidad de la herramienta
[h,w]=fp.beamgeoemetry(ampere,s) #Calcula la geometría del cordón
c =0.738  # Cuociente entre distancia entre cordones y ancho de cordón de
soldadura
v=0.05 #tamaño del voxel
d=w*c #Distancia entre cordones
theta= np.pi/(2) #Ángulo de la dirección del ráster
vector=(np.cos(theta),np.sin(theta),0) #Vector que indica la dirección del
raster
vector2=(np.cos(theta+np.pi/2),np.sin(theta+np.pi/2),0) #Vector que indica la
dirección sugerida para recorrer con zigzag
clockwise=False #Indica si se desea el recorrido espiral en sentido horario o
antihorario

# Lectura de archivo-------------------------------------------------------------
----------------------------------------
#-------------------------------------------------------------------------------
----------------------------------------
#En esta sección se cargan los archivos, puede o no contener el archivo
original
defectpcd = o3d.io.read_point_cloud("") #dirección del archivo PCD que
contiene el defecto
#originalpcd = o3d.io.read_point_cloud("dirección aquí") #dirección del
archivo PCD original

axis = o3d.geometry.TriangleMesh.create_coordinate_frame()  # se crea eje
coordenado para referencia
```

```
axis.scale(15, (0, 0, 0)) #se escala el eje para archivos creados en
CloudCompare

# Pre procesamiento de la nube----------------------------------------------
-----------------------------------------
# ---------------------------------------------------------------------------
-----------------------------------------
pcd1 = defectpcd.voxel_down_sample(voxel_size=v) #Reducción de puntos para la
nube del defecto
#pcd2 = originalpcd.voxel_down_sample(voxel_size=v) #Reducción de puntos para
la nube original
#R = originalpcd.get_rotation_matrix_from_xyz((np.pi/2, 0, 0)) #Matriz de
rotación para orientar la nube de puntos en caso de ser necesario
#pcd1.rotate(R) #Rotación
#pcd2.rotate(R) #Rotación
pcd1.scale(10, (0, 0, 0)) #Escalamiento para nubes creadas en CloudCompare
#pcd2.scale(10, (0, 0, 0)) #Escalamiento para nubes creadas en CloudCompare
pcd1.translate((0,0,50)) #Traslación para ubicar la nube en el lugar deseado
#pcd2.translate((0,0,50)) #Traslación para ubicar la nube en el lugar deseado

#Aislación de la falla caso con PCD original y cálculo de n número de capas
#-----------------------------------------
# defect=fp.deleteintersect(pcd2,pcd1,2) #Resta entre las nubes
# defectminz= np.min(np.asarray(pcd1.points)[:,2]) #Obtención del punto mínimo
en Z
# originalmaxz=np.max(np.asarray(pcd2.points)[:,2]) #Obtención del punto
máximo en Z
# n=int(np.ceil((originalmaxz-defectminz)/h)  #Determinación del número de
capas

#Aislación de la falla caso sin PCD original y cálculo de n número de capas
#-----------------------------------------
defectminz= np.min(np.asarray(pcd1.points)[:,2]) #Obtención del punto mínimo
en Z
originalmaxz=np.max(np.asarray(pcd1.points)[:,2]) #Obtención del punto máximo
en Z
n=int(np.ceil((originalmaxz-defectminz)/h)) #Determinación del número de capas
pl = fp.plane((0,0, originalmaxz), 150, 1) #Generación de plano alineado con
la capa superior
defect = fp.deleteintersect(pcd1, pl, 2) #Eliminación e puntos intersecados
con el plano generado

#Generación de capas-------------------------------------------------------
-----------------------------------------
#---------------------------------------------------------------------------
-----------------------------------------
layers=fp.offset3d(defect,h,n) #Generación de capas
layersvis=fp.addpoints(layers) #Unión d ecapas para permitir su visualización
```

```
bbox = defect.get_axis_aligned_bounding_box() #Bounding box que encierra el
defecto
croplayer=layersvis.crop(bbox) #Corte de capas según la bounding del defecto

#Visualización de capas y defecto-----------------------------------------------
----------------------------------------
#-------------------------------------------------------------------------------
----------------------------------------
o3d.visualization.draw_geometries([axis,defect]) #Visualización del defecto
aislado
# o3d.visualization.draw_geometries([axis,croplayer]) #Visualización de las
capas generadas

#Generación de trayectorias-----------------------------------------------------
----------------------------------------
#-------------------------------------------------------------------------------
----------------------------------------
#En esta sección se debe elegir el tipo de recorrido
allpath=[]

#Generación de trayectoria para caso zigzag
#-----------------------------------------
for i in layers:
    i=i.crop(bbox) #Elimina los puntos excedentes de la capa
    if i.has_points()==True:
        slice=fp.preslice01(i,d,vector,voxel_size=v) #Genera la subdivisión de
nubes para facilitar el recorrido
        if slice!=[]:
            path=fp.completepath01(slice,vector2,w) #Genera el recorrido de
las subnubes
            allpath.append(fp.addpoints(fp.addsafepoints(path,h,d))) #Une los
caminos y añade puntos de levantamiento de herramienta para las subnubes y el
código de color
finalpath=fp.addpoints(fp.addsafepoints(allpath,h,d))  #Une los caminos y
añade puntos de levantamiento de herramienta para las capas y el código de
color


#Generación de trayectoria para caso espiral
#-----------------------------------------
# for i in layers:
#     i=i.crop(bbox) #Elimina los puntos excedentes de la capa
#     if i.has_points()==True:
#         slice=fp.preslice02(i,d,voxel_size=v,clockwise=clockwise) #Genera la
subdivisión de nubes para facilitar el recorrido
#         if slice!=[]:
#             path=fp.completepath02(slice,w,clockwise=clockwise) #Genera el
recorrido de las subnubes
```

```
#             allpath.append(fp.addpoints(fp.addsafepoints(path,h,d))) #Une
los caminos y añade puntos de levantamiento de herramienta para las subnubes y
el código de color
# finalpath=fp.addpoints(fp.addsafepoints(allpath,h,d)) #Une los caminos y
añade puntos de levantamiento de herramienta para las capas y el código de
color


#Visualización y generación de código-----------------------------------------
----------------------------------------
#-----------------------------------------------------------------------------
----------------------------------------
o3d.visualization.draw_geometries([axis,finalpath]) #Visualización en Open3D
fp.animate([finalpath],t=100) #Animación con matplotlib
fp.pathvis([finalpath]) #Gráfico en matplotlib
finalpath.translate((250,900,250)) #Traslación hasta el punto de trabajo en
DTPS
fp.dtpspoints([finalpath],name='testing23',s=s,ampere=ampere) #Generación de
archivo CSR
```

9.2.2 Funciones

```
import open3d as o3d
import numpy as np
import math
from scipy.spatial.transform import Rotation as R
import matplotlib.pyplot as plt
import copy
import matplotlib.animation as animation

# ------------------------------------------------------------------------
-------------------------------------
#Parámetros de soldadura
def volt(ampere):
    '''
    volt(ampere):
    Función que retorna el valor del voltaje en función del amperaje, para el
material acero AWS-ER70S-6, de diámetro
    0.99 [mm], con gas de soldadura de composición 80% Ar y 20% CO2.
    :param ampere: Valor del amperaje utilizado
    :return Volt: Valor del voltaje asociado a los ampere utilizados
    '''
    V=0.03153846*ampere+13.44615385
    return V
def beamgeoemetry(ampere,torchspeed):
    '''
    beamgeoemetry(ampere,torchspeed):
    Función que retorna la geometría del cordón de soldadura en función del
amperaje, para el material acero
    AWS-ER70S-6, de diámetro 0.99 [mm], con gas de soldadura de composición
80% Ar y 20% CO2.
    :param ampere: Valor del amperaje utilizado
    :param torchspeed: Valor de la velocidad de herramienta utilizada
    :return [h,w]: h la altura del cordón de soldadura y w el ancho del
cordón.
    '''
    if (torchspeed>=0.15) and (torchspeed<0.25):
        h=0.015*ampere + 1.085714286
        w=0.056964286*ampere -0.05
    elif (torchspeed >= 0.25) and (torchspeed < 0.35):
        h = 0.01285714 * ampere + 1.15714286
        w = 0.04964286 * ampere + 0.05714286
    elif (torchspeed>=0.35) and (torchspeed<0.45):
        h=0.010714286*ampere + 1.228571429
        w=0.042321429*ampere +0.164285714
    elif (torchspeed>=0.45) and (torchspeed<0.55):
```

```
        h=0.00857143*ampere + 1.3
        w=0.035*ampere -0.27142857
    elif (torchspeed>=0.55) and (torchspeed<0.65):
        h=0.006428571*ampere + 1.371428571
        w=0.027678571*ampere +0.378571429
    else:
        print('out of data')
    return [h,w]


# ------------------------------------------------------------------------
-------------------------------------
#Funciones generadoras de nubes de puntos
def plane(p, b,d=1):
    '''
    plane(p,b,d=1)
    generación de nube de puntos planar, a partir de un punto p perteneciente
al plano, una normal n, equidistanciados entre si en distancia d.
    :param p: Punto perteneciente al plano
    :param b: Ancho del plano
    :param d: Distancia entre los puntos del plano
    :return: Nube de puntos que contiene puntos ubicados en un mismo plano
    '''
    a = np.arange(-b / 2, b / 2, 1 / d)
    l = np.ones_like(a)
    X = []
    Y = []
    for i in np.arange(0, len(a), 1):
        x = l * a[i]
        y = a
        X = np.concatenate((X, x), axis=0)
        Y = np.concatenate((Y, y), axis=0)
    z = np.zeros_like(X)
    c = np.c_[X, Y, z]
    plane = o3d.geometry.PointCloud()
    plane.points = o3d.utility.Vector3dVector(c)
    plane.translate((p), False)

plane.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHybrid(radiu
s=0.1, max_nn=30))
    return plane


def spiral(d, D,c=[0,0,0],clockwise=True,t=1):
    '''
    Genera una nube de puntos que siguen una espiral de euclides, con c=
centro de la espiral, d el espacio entre ciclos
```

```
    :param d: Espaciado entre cordones de soldadura
    :param D: Diámetro de la espiral
    :param c: Coordenadas del centro de la espiral
    :param clockwise: Sentido de las agujas del reloj si es verdadero
    :return: Nube de puntos de la espiral con todos sus ciclos
    '''

    n = int((D * 0.5)/ d) + 1
    pcd_list = []
    if clockwise==False:
        h=1
    else:
        h=-1

    for i in np.arange(1, n, 1):
        o1 = (2 * np.pi * (i - 1))
        o2 = (2 * np.pi * (i))
        s1 = 0.5 * (d / (2 * np.pi)) * ((o1) * math.sqrt(1 + (o1 ** 2)) +
np.log(1 + (o1 ** 2)))  # arco de la espiral
        s2 = 0.5 * (d / (2 * np.pi)) * ((o2) * math.sqrt(1 + (o2 ** 2)) +
np.log(1 + (o2 ** 2)))
        s = s2 - s1
        n2 = s / t
        a = np.arange(o1, o2-(2 * np.pi / n2), 2 * np.pi / n2)
        r = np.arange((i - 1) * d, i * d-(d / n2), d / n2)
        X = (r * np.cos(a*h))
        Y = (r * np.sin(a*h))
        z = np.ones_like(X)
        v = np.c_[X+c[0], Y+c[1], z*c[2]]
        sp = o3d.geometry.PointCloud()
        sp.points = o3d.utility.Vector3dVector(v)
        R = sp.get_rotation_matrix_from_xyz((0,0, 0))
        sp.rotate(R)
        pcd_list.append(sp)
    add = addpoints(pcd_list)

    return [pcd_list, add]
# ------------------------------------------------------------------------
-------------------------------------
# Funciones extra para visualización y manipulación de nubes de puntos
def deleteintersect(pcd1, pcd2, e=0.01):
    '''
    deleteintersect(pcd1, pcd2, e=0.01)
    Elimina los puntos de la nube pcd1 que se intersecten con la nube pcd2, la
intersección serán los puntos que tengan una distancia menor a e entre las
nubes

    parámetros
```

```
    ----------
    pcd1: PCD
        Nube de puntos base
    pcd2: PCD
        Nube de puntos que se intersecará
    e: int
        Distancia entre puntos que se considerará como intersección

    Retorno
    ----------
    pcd: PCD
        Nube de puntos que contiene los puntos de pcd1 no intersecados.

    '''
    dist = np.asarray(pcd1.compute_point_cloud_distance(pcd2))
    ind = np.where(dist > e)[0]
    pcd = pcd1.select_by_index(ind)
    return pcd

def intersect(pcd1, pcd2, e=1):
    '''
    deleteintersect(pcd1, pcd2, e=0.01)

    Selecciona los puntos de la nube pcd1 que se intersecten con la nube pcd2,
la intersección serán los puntos que tengan una distancia menor a e entre las
nubes

    Parámetros
    ----------
    pcd1: PCD
        Nube de puntos base

    pcd2: PCD
        Nube de puntos que se intersecará

    e: int
        Distancia entre puntos que se considerará como intersección

    Retorno
    ----------
    pcd: PCD
        Nube de puntos que contiene los puntos de pcd1 intersecados.

    '''
    dist = np.asarray(pcd1.compute_point_cloud_distance(pcd2))
    ind = np.where(dist <= e)[0]
    pcd = pcd1.select_by_index(ind)
    return pcd
```

```
def cleanlayerpcd(originalpcd, pcd_list, r):
    '''
    cleanlayerpcd(originalpcd, pcd_list, r)

    Quita puntos dispersos que se encuentren dentro de una distancia r
respecto a la nube de puntos anterior en la lista de nubes de puntos.

    Parámetros
    ----------
    original pcd: PCD
        Contiene la nube de puntos original.

    pcd_list: Lista de PCD
        Contiene las capas generadas a partir de la nube de puntos original.

    Retorno
    ----------
    clean_list: Lista de PCD
        Lista con las capas tras ser procesadas.

    '''
    clean_list = [deleteintersect(pcd_list[0], originalpcd, r)]
    i = 1
    while i < len(pcd_list):
        clean_list.append(deleteintersect(pcd_list[i], pcd_list[i - 1], r))
        i += 1
    return clean_list

def pcdlistvis(pcd_list, d=100, n=(0, 0,1)):
    '''
    vispcdlist(pcd_list, d=100, n=(0, 0,1))

    Visualización de la lista de PCD, desplaza cada elemento una distancia d
en dirección n para visualizarlo mejor

    Parámetros
    ----------
    pcd_list : Lista de PCD

        Lista de PCD a visualizar
    d : int
        Distancia entre cada elemento de la lista

    n : array (x,y,z)
        Dirección en la que se desplazarán los elementos de la lista
```

```
    Retorno
    ----------


    '''
    c = 0
    n = np.asarray(n)
    v = n * d
    pcd_list2 = []
    for i in pcd_list:
        pcd_list2.append(i.translate(v * c))
        c += 1
    return pcd_list2


def color(pcd_list):
    '''
    color(pcd_list)
    Cambia los colores de los PCD de la lista para poder diferenciarlos entre
si

    parámetros
    ----------
    pcd_list : Lista de PCD
        Lista de PCD a cambiar de color.

    Retorno
    ----------
    pcd_list : Lista de PCD
        Lista con los colores ya modificados.

    '''
    if len(pcd_list) == 0:
        return pcd_list
    n = 1 / len(pcd_list)
    j = 0
    for i in pcd_list:
        s = j / 4
        i.paint_uniform_color([0, 0.7 + s, 1 - j])
        j += n
    return pcd_list


def addpoints(pcd_list):
    '''
    addpoints(pcd_list)
```

```
    Combina los puntos de las nubes de punto de la lista para poder
visualizarla como una misma nube.

    Parámetros
    ----------
    pcd_list : Lista de PCD
        Lista con los PCD para combinar

    Retorno
    ----------
    PCD : PCD
        PCD que contiene todos los puntos de las nubes en un mismo elemento

    '''
    pcd_points = np.asarray(pcd_list[0].points)
    for i in pcd_list[1:]:
        pcd_load = np.asarray(i.points)
        pcd_points = np.concatenate((pcd_points, pcd_load), axis=0)
    pcd_colors = np.asarray(pcd_list[0].colors)
    for i in pcd_list[1:]:
        pcd_loadc = np.asarray(i.colors)
        pcd_colors = np.concatenate((pcd_colors, pcd_loadc), axis=0)
    pcd_normals = np.asarray(pcd_list[0].normals)
    for i in pcd_list[1:]:
        pcd_loadn = np.asarray(i.normals)
        pcd_normals = np.concatenate((pcd_normals, pcd_loadn), axis=0)
    PCD = o3d.geometry.PointCloud()
    PCD.points = o3d.utility.Vector3dVector(pcd_points)
    PCD.colors = o3d.utility.Vector3dVector(pcd_colors)
    PCD.normals = o3d.utility.Vector3dVector(pcd_normals)
    return PCD


def colorindex(pcd, L):
    '''
    colorindex(pcd, L)

    Colorea los puntos contenidos en la lista L según su indice

    Parámetros
    ----------

    pcd : PCD
        Nube de puntos a colorear
    L : array
        Lista que contiene los índices de los puntos a modificar su color.

    Retorno
```

```
    ----------

    pcd : PCD
        Retorna la nube de puntos con sus elementos ya coloreados

    '''
    n = 0
    print('L', L)
    for i in L:
        n += 1
        pcd.colors[i] = [1, 0, 0]
    return pcd

def pathvis(completepath):
    '''
    pathvis(completepath)

    Visualiza el camino completepath con la librería matplotlib

    Parámetros
    ----------
    completepath: array
        Arreglo con las coordenadas a recorrer
    x : List[int]
        Límite inferior y superior del eje X
    y : List[int]
        Límite inferior y superior del eje Y
    z : List[int]
        Límite inferior y superior del eje Z
    Retorno
    ----------

    '''
    ax = plt.axes(projection='3d')
    X = []
    Y = []
    Z = []
    for i in completepath:
        points=np.asarray(i.points)
        X.append(points[:, 0])
        Y.append(points[:, 1])
        Z.append(points[:, 2])
    X = np.hstack(X)
    Y = np.hstack(Y)
    Z = np.hstack(Z)
    ax.plot3D(X, Y, Z, 'blue',linewidth=0.5, alpha=1)
    # parámetros gráfico
    all=addpoints(completepath)
```

```
    max=all.get_max_bound()
    min=all.get_min_bound()
    xl=max[0]-min[0]
    yl = max[1] - min[1]
    zl = max[2] - min[2]
    maxlimit=np.max([xl,yl,zl])
    ax.set_xlim3d([((max[0]+min[0])/2)-
(maxlimit/2),((max[0]+min[0])/2)+(maxlimit/2)])
    ax.set_xlabel('X')
    ax.set_ylim3d([((max[1]+min[1])/2)-
(maxlimit/2),((max[1]+min[1])/2)+(maxlimit/2)])
    ax.set_ylabel('Y')
    ax.set_zlim3d([((max[2]+min[2])/2)-
(maxlimit/2),((max[2]+min[2])/2)+(maxlimit/2)])
    ax.set_zlabel('Z')
    # ax.set_aspect(auto)
    # ax.autoscale()

    plt.show()
def update_lines(num, dataLines, lines):
    for line, data in zip(lines, dataLines):
        # NOTE: there is no .set_data() for 3 dim data...
        line.set_data(data[0:2, :num])
        line.set_3d_properties(data[2, :num])
    return lines

def animate(completepath,t=1000):
    '''
    animate(completepath,x=[-100,100],y=[-100,100],z=[-100,100],t=1000)

    Visualiza una animación del camino completepath con la librería matplotlib

    Parámetros
    ----------
    completepath: array
        Arreglo con las coordenadas a recorrer
    x : List[int]
        Límite inferior y superior del eje X
    y : List[int]
        Límite inferior y superior del eje Y
    z : List[int]
        Límite inferior y superior del eje Z
    t : int
        Intervalo de tiempo entre cada cuadro de animación, en ms

    Retorno
    ----------
    '''
```

```
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    X = []
    Y = []
    Z = []
    for i in completepath:
        points=np.asarray(i.points)
        X.append(points[:, 0])
        Y.append(points[:, 1])
        Z.append(points[:, 2])
    X = np.hstack(X)
    Y = np.hstack(Y)
    Z = np.hstack(Z)
    data = [np.array([X, Y, Z])]
    lines = [ax.plot(dat[0, 0:1], dat[1, 0:1], dat[2, 0:1])[0] for dat in
data]
    #ax.set_title('3D Test')
    all=addpoints(completepath)
    max=all.get_max_bound()
    min=all.get_min_bound()
    xl=max[0]-min[0]
    yl = max[1] - min[1]
    zl = max[2] - min[2]
    maxlimit=np.max([xl,yl,zl])
    ax.set_xlim3d([((max[0]+min[0])/2)-
(maxlimit/2),((max[0]+min[0])/2)+(maxlimit/2)])
    ax.set_xlabel('X')
    ax.set_ylim3d([((max[1]+min[1])/2)-
(maxlimit/2),((max[1]+min[1])/2)+(maxlimit/2)])
    ax.set_ylabel('Y')
    ax.set_zlim3d([((max[2]+min[2])/2)-
(maxlimit/2),((max[2]+min[2])/2)+(maxlimit/2)])
    ax.set_zlabel('Z')
    # corre animación

    line_ani = animation.FuncAnimation(fig, update_lines, frames=2000,
fargs=(data, lines), interval=t, blit=False)
    plt.show()
# ------------------------------------------------------------------------
-------------------------------------
# Generación de capas
def offset3d(pcd, h, n):
    '''
    offset3dproc(pcd, d, n)
    Generación de capas no planares desplazadas en direcciones normales a la
superficie

    parámetros
```

```
    ----------
    pcd: PCD
        Nube de puntos base
    h: int
        Altura del cordón de soldadura
    n: int
        Número de capas a realizar

    Retorno
    ----------
    pcd_list: Lista de PCD
        Lista que contiene las capas generadas en formato PCD

    '''
    pcd_list = []
    pcd_list.append(normtrans(pcd, h))
    for i in range(n):
        pcd_list.append(normtrans(pcd_list[-1], h))
    pcd_list=cleanlayerpcd(pcd,pcd_list,h - h*0.2)
    return pcd_list

def multioffset3d(pcd, h,
n,px=False,py=False,pz=True,nx=False,ny=False,nz=False):
    '''
    offset3dproc(pcd, d, n)
    Generación de capas no planares desplazadas en direcciones normales a la
superficie

    parámetros
    ----------
    pcd: PCD
        Nube de puntos base
    h: int
        Altura del cordón de soldadura
    n: int
        Número de capas a realizar

    Retorno
    ----------
    pcd_list: Lista de PCD
        Lista que contiene las capas generadas en formato PCD

    '''
    pcd_list = []
    if pz == True:
        layer = normtrans(pcd, h, cam=[0, 0, 1])
    if py == True:
        layerf = normtrans(pcd, h, cam=[0, 1, 0])
```

```
            layer = addpoints([layer, layerf])
        if px == True:
            layerbk = normtrans(pcd, h, cam=[1, 0, 0])
            layer = addpoints([layer, layerbk])
        if nx == True:
            layerl = normtrans(pcd, h, cam=[-1, 0, 0])
            layer = addpoints([layer, layerl])
        if ny == True:
            layerr = normtrans(pcd, h, cam=[0, -1, 0])
            layer = addpoints([layer, layerr])
        if nz == True:
            layerb = normtrans(pcd, h, cam=[0, 0, -1])
            layer = addpoints([layer, layerb])

        pcd_list.append(layer)

        for i in range(n):
            if pz==True:
                layer=normtrans(pcd_list[-1], h,cam=[0,0,1])
            if py==True:
                layerf = normtrans(pcd_list[-1], h,cam=[0,1,0])
                layer=addpoints([layer,layerf])
            if px==True:
                layerbk = normtrans(pcd_list[-1], h, cam=[1, 0, 0])
                layer = addpoints([layer, layerbk])
            if nx==True:
                layerl = normtrans(pcd_list[-1], h, cam=[-1, 0, 0])
                layer = addpoints([layer, layerl])
            if ny==True:
                layerr = normtrans(pcd_list[-1], h, cam=[0, -1, 0])
                layer = addpoints([layer, layerr])
            if nz==True:
                layerb = normtrans(pcd_list[-1], h, cam=[0, 0, -1])
                layer = addpoints([layer, layerb])

            pcd_list.append(layer)

        return pcd_list

def normtrans(pcd,h,cam=[0,0,1]):
    '''
    normtransproc(pcd,h)
    Crea una capa sobre la superficie de la nube de puntos, trasladando cada
punto de la nube PCD una distancia h en la dirección de su normal

    parámetros
    ----------
    pcd: PCD
```

```
        Nube de puntos base
    h: int
        Altura del cordón de soldadura

    Retorno
    ----------
    d: PCD
        Nube de puntos de la capa generada sobre la superficie de la nube de
puntos base

    '''
    c = o3d.geometry.PointCloud()
    p1 = np.asarray(pcd.points)  # puntos de la nube
    v1 = np.asarray(pcd.normals)  # vectores normales de la nube
    p2 = p1 + v1 * h
    c.points = o3d.utility.Vector3dVector(p2)
    c.colors = o3d.utility.Vector3dVector(np.asarray(pcd.colors))
    c.estimate_normals()
    c.orient_normals_to_align_with_direction(np.asarray(cam))
    c.remove_statistical_outlier(5, h)
    diameter = np.linalg.norm(np.asarray(c.get_max_bound()) -
np.asarray(c.get_min_bound()))
    camera = np.asarray(cam)*diameter
    radius = diameter * 100
    _, pt_map = c.hidden_point_removal(camera, radius)
    d = c.select_by_index(pt_map)
    return d

# Separación en secciones
# ------------------------------------------------------------------------------
------------------------
def preslice01(pcd, d=3, v=(0.85, 0.85, 0 ),voxel_size=0.05) :
    '''
    slice01(pcd, w=1, v=(0, 1, 0))
    secciona la nube de puntos intersecando planos equiespaciados en distancia
d , con normal en v

    parámetros
    ----------
    pcd: PCD
        Nube de puntos base a intersecar
    d : int
        Distancia que separa a los planos
    v : array (x,y,z)
        Vector que indica la normal de los planos

    Retorno
    ----------
```

```
    [pcd_list, pcd_tot]
    pcd_list: Lista de PCD
        Contiene las intersecciones de los planos con la superficie de la nube
base.
    pcd_tot:
        Nube de puntos que contiene todas las intersecciones.

    '''
    pcd=copy.deepcopy(pcd)

    points = np.asarray(pcd.points)
    center=pcd.get_center()
    newpoints=points-center
    A=np.asarray(v)*(-1)
    ind = np.argmax(np.dot(newpoints,A))
    p=points[ind]
    minbound=pcd.get_min_bound()
    maxbound=pcd.get_max_bound()
    b=np.linalg.norm(maxbound-minbound)*2
    pl = plane(p,b, d)
    R = pl.get_rotation_matrix_from_xyz((np.pi / 2, 0, 0))
    pl.rotate(R)
    R2 = pl.get_rotation_matrix_from_xyz((0,0,np.arccos(v[0])-np.pi/2))  # -
np.pi/4
    pl.rotate(R2)
    pcd_list = []
    pcd_list.append(intersect(pcd, pl,d/4))
    while len(np.asarray(pcd_list[-1].points)) != 0:
        pl = pl.translate(np.array(v) * d)
        pcd_list.append(intersect(pcd, pl, d/4))
    pcd_list.pop(-1)
    pcd_list = color(pcd_list)
    return pcd_list

def preslice02(pcd, d=1,voxel_size=0.05,center=[0,0,0],clockwise=False ) :
    '''
    slice02(pcd, d=1, v=(0, 1, 0))
    secciona la nube de puntos intersecando espirales de euclides con
distancia entre ciclos d, con normal en v

    parámetros
    ----------
    pcd: PCD
        Nube de puntos base a intersecar
    d : int
        Distancia que separa a los planos
    v : array (x,y,z)
        Vector que indica la normal de los planos
```

```
    Retorno
    ----------
    [pcd_list, pcd_tot]
    pcd_list: Lista de PCD
        Contiene las intersecciones de los planos con la superficie de la nube
base.
    pcd_tot:
        Nube de puntos que contiene todas las intersecciones.

    '''
    pcdp = copy.deepcopy(pcd)  # Nube de puntos a proyectar, crea una copia
para no modificar la original
    points = np.asarray(pcdp.points)
    c= o3d.geometry.PointCloud()
    c.points=o3d.utility.Vector3dVector(np.c_[center[0],center[1],center[2]])
    dist = np.asarray(pcdp.compute_point_cloud_distance(c))
    ind = np.where(dist==np.min(dist))[0][0]
    minbound=pcdp.get_min_bound()
    maxbound=pcdp.get_max_bound()
    p=points[ind]
    q=copy.deepcopy(p)
    q[2]=minbound[2]
    D=np.linalg.norm(maxbound-minbound)*2
    sp = spiral(d, D,c=q,clockwise=clockwise,t=int(d/4))  # crea espiral
    pcd_list = []
    i=1
    n=int(maxbound[2]-minbound[2])+1
    pr = projection(sp[0], n, (0, 0, -1), 1)  # proyecta la espiral
    pcd_list.append(intersect(pcdp, pr[0],d/6)) #intersecta la espiral
    while len(np.asarray(pcd_list[-1].points)) != 0 and i < len(pr):
            pcd_list.append(intersect(pcdp, pr[i],d/6))
            i += 1

    pcd_list.pop(-1)
    pcd_list = color(pcd_list)

    return pcd_list

def projection(pcd_list, n=10, v=(0, 0, 1), d=0.5):  # repite n veces la nube
PCD en la dirección v
    '''
    projection(pcd_list, n=10, v=(0, 0, 1), d=1)

    Repite n veces la nube PCD en la dirección v, con una separación d

    Parámetros
    ----------
```

```
    pcd_list: Lista de PCD
        Contiene una lista con archivos PCD
    n: int
        Número de repeticiones
    v : array (x,y,z)
        Vector en que se repetirán las nubes de puntos
    d : int
       Distancia de separación entre repeticiones

    Retorno
    ----------
    pcd_list2 : Lista de PCD
        Contiene las proyecciones de cada nube de puntos de la lista de PCD
original.

    '''

    pcd_list2 = []
    for i in pcd_list:
        i.translate((np.array(v) * (-d)*(n+1)))
        pr = o3d.geometry.PointCloud()
        pcd_points = np.asarray(i.points)
        for j in np.arange(1, n*2, 1):
            pcd_load = np.asarray(i.points)
            pcd_points = np.concatenate((pcd_points, pcd_load), axis=0)
            i.translate((np.array(v) * d))
            pr.points = o3d.utility.Vector3dVector(pcd_points)
        pcd_list2.append(pr)
    return pcd_list2
# ------------------------------------------------------------------------------
-------------------------------------
#Recorrido
def slice01(pcd, v=(1, 0, 0), d=5):
    '''
    slice01(pcd, v=(1, 0, 0), d=5):
    Función que se encarga de generar una trayectoria con puntos distanciados
en distacncia d, que recorren la línea de
    puntos pcd, en la dirección del vector v
    :param pcd: Nube de puntos a recorrer
    :param v: Vector que indica la dirección en que se recorrerá
    :param d: Distancia entre los puntos a recorrer
    :return pathpcd: Nube de puntos con los puntos de la trayectoria
    '''
    pcd = copy.deepcopy(pcd)
    pcd.normalize_normals()
    pcd_tree = o3d.geometry.KDTreeFlann(pcd)
    path = []
    points = np.asarray(pcd.points)
```

```
    points2 = [arr.tolist() for arr in points]
    normals= np.asarray(pcd.normals)
    center=pcd.get_center()
    newpoints=points-center
    A=np.asarray(v)*(-1)
    ind = np.argmax(np.dot(newpoints,A))
    ind2 = ind + 1
    i = 0
    c1=[]
    c2=[]
    c3=[]
    n1=[]
    n2=[]
    n3=[]
    c1.append(points[ind][0])
    c2.append(points[ind][1])
    c3.append(points[ind][2])
    n1.append(normals[ind][0])
    n2.append(normals[ind][1])
    n3.append(normals[ind][2])
    while ind not in path: #Se detiene al elegir un punto ya contenido en la
trayectoria (lo cual puede indicar un borde)
        path.append(ind)
        [k, idx, _] = pcd_tree.search_radius_vector_3d(pcd.points[ind], d)
        # np.asarray(pcd.colors)[idx[1:], :] = [0, 1, 0]
        np.asarray(pcd.colors)[ind] = [1, 0, 0]
        [k, idx2, _] = pcd_tree.search_knn_vector_3d(pcd.points[ind], 5)  #
selección de los vecinos de ind
        ns = np.asarray(pcd.normals)[idx2]  # array para trabajar con ellos
        [x, y, z] = [np.mean(ns[:, 0]), np.mean(ns[:, 1]),np.mean(ns[:, 2])]
# promedio de las normales de los vecinos de ind
        c = np.c_[x, y, z]  # nueva normal
        np.asarray(pcd.normals)[ind] = o3d.utility.Vector3dVector(c)  #
reescribe la normal del punto seleccionado con el promedio de sus vecinos
        h = points[idx]
        h2 = h - (np.ones_like(h) * points[ind])  # cambia el 0 de los puntos
        dot = np.dot(h2, v)  # producto punto para determinar los vecinos
ubicados en la dirección deseada
        max = h[np.argmax(dot)]  # selección del vecino más lejano en la
dirección deseada
        if path[0] in idx and i>2:
            path.pop(-1)
            c1.pop(-1)
            c2.pop(-1)
            c3.pop(-1)
            n1.pop(-1)
            n2.pop(-1)
            n3.pop(-1)
```

```
            print('break',i)
            break
        ind2 = ind
        max2=np.asarray(max)
        max2=[arr.tolist() for arr in max2]
        ind= points2.index(max2)
        i += 1
        c1.append(points[ind][0])
        c2.append(points[ind][1])
        c3.append(points[ind][2])
        n1.append(normals[ind][0])
        n2.append(normals[ind][1])
        n3.append(normals[ind][2])

    pathpoints=np.c_[c1, c2, c3]
    pathnormals = np.c_[n1, n2, n3]
    pathpcd = o3d.geometry.PointCloud()
    pathpcd.points = o3d.utility.Vector3dVector(pathpoints)
    pathpcd.normals = o3d.utility.Vector3dVector(pathnormals)
    pathpcd.paint_uniform_color((0,0,1))
    return pathpcd

def arctanfull(y,x):
    '''
    arctanfull(y,x):
    Función que devuelve la arcotangente de x/y, se diferencia de arctan2 ya
que su recorrido va de 0 a 2pi
    :param y: Valor en y
    :param x: Valor en x
    :return angle: Valor del ángulo en el plano
    '''
    angle=np.arctan2(y,x)
    ind=np.where(angle<0)[0]
    negativeangle=2*np.pi+angle
    angle[ind]=negativeangle[ind]
    return angle

def slice02(pcd, d=5,clockwise=False,center=[0,0,0]):
    '''
    slice02(pcd, d=5,clockwise=False):
    Función que se encarga de generar una trayectoria con puntos distanciados
en distacncia d, que recorren la línea de
    puntos en espiral pcd , en sentido horario o antihorario
    :param pcd: Nube de puntos
    :param d: Ancho del cordón de soldadura
    :param clockwise: Indica el sentido de la espiral recorrida
    :return:
    '''
```

```
    pcd = copy.deepcopy(pcd)
    #pcd.normalize_normals()
    pcd_tree = o3d.geometry.KDTreeFlann(pcd)
    path = []
    points = np.asarray(pcd.points)
    points2 = [arr.tolist() for arr in points]
    normals= np.asarray(pcd.normals)
    c = o3d.geometry.PointCloud()
    c.points = o3d.utility.Vector3dVector(np.c_[center[0], center[1],
center[2]])
    angle = arctanfull(points[:,1]-center[1],points[:,0]-center[0])
    if clockwise==True:
        cw=-1
        ind = np.where(angle == np.max(angle))[0][0]
    else:
        cw=1
        ind = np.where(angle == np.min(angle))[0][0]
    p = points[ind]
    theta=angle[ind]
    v=(np.cos(theta+(np.pi/2)*cw),np.sin(theta+(np.pi/2)*cw),0)
    ind2 = ind + 1
    # dirección sugerida para buscar el máximo
    i = 0
    c1=[]
    c2=[]
    c3=[]
    n1=[]
    n2=[]
    n3=[]
    c1.append(points[ind][0])
    c2.append(points[ind][1])
    c3.append(points[ind][2])
    n1.append(normals[ind][0])
    n2.append(normals[ind][1])
    n3.append(normals[ind][2])
    #while theta<=(np.pi*2):
    while ind not in path:  # Se detiene al quedar en el mismo punto
        path.append(ind)
        [k, idx, _] = pcd_tree.search_radius_vector_3d(pcd.points[ind], d)
        np.asarray(pcd.colors)[ind] = [1, 0, 0]
        [k, idx2, _] = pcd_tree.search_knn_vector_3d(pcd.points[ind], 5)  #
selección de los vecinos de ind
        ns = np.asarray(pcd.normals)[idx2]  # array para trabajar con ellos
        [x, y, z] = [np.mean(ns[:, 0]), np.mean(ns[:, 1]),
                     np.mean(ns[:, 2])]  # promedio de las normales de los
vecinos de ind
        c = np.c_[x, y, z]  # nueva normal
```

```
        np.asarray(pcd.normals)[ind] = o3d.utility.Vector3dVector(c)  #
reescribe la normal del punto seleccionado con el promedio de sus vecinos
        h = points[idx]
        h2 = h - (np.ones_like(h) * points[ind])  # cambia el 0 de los puntos
        dot = np.dot(h2, v)  # producto punto para determinar los vecinos
ubicados en la dirección deseada
        max = h[np.argmax(dot)]  # selección del vecino más lejano en la
dirección deseada
        if path[0] in idx and i > 2:
            path.pop(-1)
            c1.pop(-1)
            c2.pop(-1)
            c3.pop(-1)
            n1.pop(-1)
            n2.pop(-1)
            n3.pop(-1)
            break
        if  i > 3:
            if path[-1] in idx:
                path.pop(-1)
                c1.pop(-1)
                c2.pop(-1)
                c3.pop(-1)
                n1.pop(-1)
                n2.pop(-1)
                n3.pop(-1)
                break
        ind2 = ind
        max2=np.asarray(max)
        max2=[arr.tolist() for arr in max2]
        ind= points2.index(max2)

        i += 1
        c1.append(points[ind][0])
        c2.append(points[ind][1])
        c3.append(points[ind][2])
        n1.append(normals[ind][0])
        n2.append(normals[ind][1])
        n3.append(normals[ind][2])
        norm = np.linalg.norm(
            points[ind2] - points[ind])  # norma del vector, para poder
convertirlo en vector unitario
        if norm == 0:
            break
        v = np.transpose((points[ind] - points[ind2]) / norm)
        p=points[ind]
        theta=arctanfull([p[1]-center[1]],[p[0]-center[0]])
```

```
        v2 = (np.cos(theta + (np.pi / 2) * cw), np.sin(theta + (np.pi / 2) *
cw), 0)
        turn=np.dot(v, v2)
        v=v2
        #print(turn)
        if turn < 0:
            break
        # ind = np.where(dist == np.min(dist))[0][0]
        # p = points[ind]
        # theta = np.arccos(p[0] / np.sqrt(((p[0] - center[0]) ** 2 + (p[1] -
center[1]) ** 2)))
    pathpoints=np.c_[c1, c2, c3]
    pathnormals = np.c_[n1, n2, n3]
    pathpcd = o3d.geometry.PointCloud()
    pathpcd.points = o3d.utility.Vector3dVector(pathpoints)
    pathpcd.normals = o3d.utility.Vector3dVector(pathnormals)
    pathpcd.paint_uniform_color((0,0,1))
    return pathpcd

def completepath01(pcd_list,v,d):
    '''
    completepath(pcd_list)

    Recorre la lista de nubes de puntos con la función slice01 para raster

    Parámetros
    ----------

    pcd_list : Lista de PCD
        Lista que contiene los PCD a recorrer

    Retorno
    ----------
    [completepath, completepathpcd]

    completepath : Lista
        Lista con las coordenadas de cada recorrido

    completepathpcd : Lista de PCD
        Lista con las nubes de puntos de los recorridos
    '''
    completepathpcd = []
    n=0
    for i in pcd_list:
        p = slice01(i,v,d)
        completepathpcd.append(p)
        nointer=deleteintersect(i,p,d)
        while len(np.asarray(nointer.points))!=0:
```

```
            n+=1
            p=slice01(nointer,v,d)
            nointer=deleteintersect(nointer,p,d)
            completepathpcd.append(p)
        v=(np.ones_like(v)*(-1))*v

    return completepathpcd

def completepath02(pcd_list,d,clockwise=False,center=[0,0,0]):
    '''
    completepath(pcd_list)

    Recorre la lista de nubes de puntos con la función rec1 para raster

    Parámetros
    ----------

    pcd_list : Lista de PCD
        Lista que contiene los PCD a recorrer

    Retorno
    ----------
    [completepath, completepathpcd]

    completepath : Lista
        Lista con las coordenadas de cada recorrido

    completepathpcd : Lista de PCD
        Lista con las nubes de puntos de los recorridos
    '''
    completepathpcd = []
    n = 0
    for i in pcd_list:
        D=np.linalg.norm(i.points[0]-center)
        s = 0
        if D < 3*d:
            ini=0.5
        else:
            ini=1
        p = slice02(i,d*ini,clockwise)
        completepathpcd.append(p)
        nointer=deleteintersect(i,p,d*1.2*ini)
        n+=1
        while len(np.asarray(nointer.points))!=0:
            p = slice02(nointer, d*ini, clockwise)
            nointer = deleteintersect(nointer, p, d*1.2*ini)
            completepathpcd.append(p)
            #print(n,s)
```

```
            s+=1
    return completepathpcd

def addsafepoints(completepathpcd,h,d):
    '''
    addsafepoints(completepathpcd,h,d):
    Añade puntos a la trayectoria permitiendo que la herramienta se levante 5
veces h, cuando la distancia entre 2 puntos supera
    en 2.5 veces la distancia d
    :param completepathpcd:
    :param h: Altura del cordón
    :param d: Distancia entre cordones
    :return: Nube de puntos con la trayectoria con puntos donde se levanta la
herramienta
    '''
    completepathpcdsp=copy.deepcopy(completepathpcd)
    for i in np.arange(0,len(np.asarray(completepathpcdsp))-1,1):
        dist= np.linalg.norm( completepathpcd[i+1].points[0]-
completepathpcd[i].points[-1])
        if dist > 2.5*d:
            fin=edgepoint(completepathpcdsp[i],5*h,end=True)
            ini=edgepoint(completepathpcdsp[i+1],5*h,end=False)
            completepathpcdsp[i].colors[-1]= ((1,0,0))
            completepathpcdsp[i+1].colors[0] = ((0, 1, 0))
            completepathpcdsp[i+1]=addpoints([fin,ini,completepathpcdsp[i+1]])
    return completepathpcdsp

def uniquepoints(pcd_list):
    completepath=addpoints(pcd_list)
    ind=np.unique(np.asarray(completepath.points),return_index=True)[1]
    print(ind)
    unique=o3d.geometry.PointCloud()
    points= np.asarray(completepath.points)[ind]
    colors=np.asarray(completepath.colors)[ind]
    normals= np.asarray(completepath.normals)[ind]
    unique.points = o3d.utility.Vector3dVector(np.c_[points[:,0], points[:,1],
points[:,2]])
    unique.colors = o3d.utility.Vector3dVector(np.c_[colors[:,0], colors[:,1],
colors[:,2]])
    unique.normals = o3d.utility.Vector3dVector(np.c_[normals[:,0],
normals[:,1], normals[:,2]])
    return unique

def edgepoint(pcd,L,end=False):
    '''
        endpoint(pcd,L,end=False)):
        Función utilizada para agregar puntos finales e iniciales en la
trayectoria
```

```
    :param pcd:
    :param L: Indica la distancia de elevación para el nuevo punto
    :param end: Indica si es un punto terminal o no, si es Falso, significa
que es un punto inicial

    :return:  Retorna una nube de puntos de un solo punto
    '''
    pcd=copy.deepcopy(pcd)
    p = o3d.geometry.PointCloud()
    if end==True:
        i=-1
    else:
        i = 0
    c = np.asarray(pcd.points)[i]
    c[2]=c[2]+L
    p.points = o3d.utility.Vector3dVector(np.c_[c[0], c[1], c[2]])
    p.paint_uniform_color((1,0,0))
    p.normals = o3d.utility.Vector3dVector(np.c_[0, 0, 1])
    #print(np.asarray(p.points),np.asarray(pcd.points)[-
1],np.asarray(pcd.points)[0])
    return p

# ------------------------------------------------------------------------------
-------------------------------------
#Escritura programa
def wristvector(normal):
    '''

    :param normal:
    :return wv: Vector que indica la dirección de la muñeca del robot
    '''
    wv=copy.deepcopy(normal)
    wv[2]=0
    wv[0] = 1
    wv[1] = 0
    #print('wv',wv)
    return wv

def dtpspoints(path_list,name='outfile',s=0.2,ampere=100):
    '''
    dtpspoints(path_list)
    Crea un archivo CSR para ser leído por el programa DTPS

    parámetros
    ----------
    path_list : array
        Lista que contiene las coordenadas
```

```
    Retorno
    ----------
    none

    '''
    outfilename_default = name+'.csr'
    outfilepath_default = r'..\Programas
Escritos\{}'.format(outfilename_default)
    #Strings de parámetros
    description= '''[Description]
Robot, TM1400(G3)
Comment,
SubComment1,
SubComment2,
Mechanism, 1(0001)
Tool, 1:TOOL01
Creator, IMA+
User coordinates, None
Create, 2021, 03, 17, 11, 05, 26
Update, 2021, 03, 17, 11, 05, 26
Original,
Edit, 0

[Work]
Position, 5, 0, 0, 10, -0, 0, 0

[Pose]
/Name, Type, X, Y, Z, X1, X2, X3, Z1, Z2, Z3, G4'''+'\n'
    variable='''[Variable]
LB, LB001, , 0
LB, LB002, , 0
LB, LB003, , 0
LB, LB004, , 0
LB, LB005, , 0
LI, LI001, , 0
LI, LI002, , 0
LI, LI003, , 0
LI, LI004, , 0
LI, LI005, , 0
LL, LL001, , 0
LL, LL002, , 0
LL, LL003, , 0
LL, LL004, , 0
LL, LL005, , 0
LR, LR001, , 0.000000000000000
LR, LR002, , 0.000000000000000
LR, LR003, , 0.000000000000000
LR, LR004, , 0.000000000000000
```

```
LR, LR005, , 0.000000000000000

[Command]
TOOL, 1:TOOL01'''+'\n'
    l=1
    command=[]
    try:
        fileID = open(outfilename_default, mode='x', encoding='UTF-8')
        fileID.write(description)
        for i in np.arange(0,len(path_list),1):
            points=path_list[i].points
            normals=path_list[i].normals
            colors=path_list[i].colors
            for j in np.arange(0,len(points),1):
                if colors[j][0]==1:
                    if colors[j-1][2]==1:
                        command.append('ARC-OFF, ArcEnd1.rpg, 0'+'\n'+'DELAY,
3'+ '\n')
                    command.append('MOVEL, P' + str(l) + ','+str(50) +',
m/min, 0, N, -1' + '\n')
                if colors[j][1]==1:
                    command.append('ARC-SET,'
+str(ampere)+','+str(volt(ampere))+ ',0.5'+'\n'+'ARC-ON, ArcStart1.rpg,
0'+'\n')
                    command.append('MOVEL, P' + str(l) + ','+str(s) +', m/min,
0, W, -1' + '\n')
                if colors[j][2]==1:
                    print(normals[j])
                    command.append('MOVEL, P' + str(l) + ','+str(s) +', m/min,
0, W, -1' + '\n')
                #Código con herramienta que sigue la orientaicón normal a la
superficie--------------------------------
                fileID.write('P' + str(l) + ', AV,' +
np.array2string(points[j], separator=',', prefix='',

suffix='').translate(
                    {ord(h): None for h in '[]'}) + ',' +
np.array2string(normals[j] * (-1), separator=',', prefix='',

suffix='').translate(
                    {ord(h): None for h in '[]'})+','+
np.array2string(wristvector(normals[j]), separator=',', prefix='',

suffix='').translate(
                    {ord(h): None for h in '[]'}) +'\n')
                #Código con orientación de la herramienta fija----------------
----------------------------------------
                #
```

```
                # fileID.write('P' + str(l) + ', AV,' +
np.array2string(points[j], separator=',', prefix='',
                #
suffix='').translate(
                #     {ord(h): None for h in '[]'}) + ',' + "0,0,-1,1,0,0"
+'\n')
                #-------------------------------------------------------------
----------------------------
                l+=1
        fileID.write(variable)
        fileID.write('MOVEL, P' + str(1) + ','+str(s) +', m/min, 0, N, -1' +
'\n')
        #fileID.write('MOVEL, P' + str(1) + ',' + str(s) + ', m/min, 0, N, -1'
+ '\n')
        fileID.write('''ARC-SET, '''+str(ampere)+','+str(volt(ampere)) +''',
0.50
ARC-ON, ArcStart1.rpg, 0'''+'\n')
        for n in command:
            fileID.write(n)
        fileID.write('''CRATER, '''+str(ampere)+','+str(volt(ampere)) +''',
0.00
ARC-OFF, ArcEnd1.rpg, 0
DELAY, 3''')
        fileID.close()
    except FileExistsError:
        print('EXISTE')
```