



TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DE COMPUTADORES



Sistema para la detección de árboles usando tecnología LIDAR

Estudiante: Julián Villamor Barreiro

Dirección: Carlos Vázquez Reguerio

Margarita Amor López

A Coruña, September de 2023.

Dedicatoria

Agradecimientos

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Resumen

El LiDAR (Light Detection and Ranging) es una tecnología de detección remota que emplea pulsos láser para medir distancias y generar modelos 3D de objetos y entornos. En el contexto de la detección de árboles, esta destaca por su capacidad para capturar detalles como la estructura y ubicación de la vegetación. En los últimos años, esta metodología de estudio de los bosques ha ganado popularidad. En este trabajo, se plantea y desarrolla la aplicación de algoritmos basados en características geométricas de la vegetación, junto con el análisis de los resultados obtenidos a partir de modelos de datos LIDAR de Luxemburgo.

Abstract

LiDAR (Light Detection and Ranging) is a remote sensing technology that employs laser pulses to measure distances and generate three-dimensional models of objects and environments. In the context of tree detection, LiDAR stands out for its ability to capture precise details about the structure and location of vegetation. In recent years, this approach to studying forests has gained popularity. In this work, the application of algorithms based on geometric characteristics of vegetation is proposed and developed, along with the analysis of the results obtained from LiDAR data models from Luxembourg.

Palabras clave:

- LiDAR
- Nube de Puntos
- Árboles
- Análisis de datos
- Características geométricas
- Dósdel arbóreo
- Gestión forestal

Keywords:

- LiDAR
- Point Cloud
- Trees
- Data Analysis
- Geometric Features
- Tree Canopy
- Forest Management

Índice general

1	Introducción	1
1.1	Motivación	1
1.2	Objetivos	2
1.3	Apartados de la memoria	3
2	Fundamentos Teóricos y Tecnológicos	4
2.1	LiDAR	4
2.1.1	Nubes de Puntos	4
2.2	Detección de Objetos	7
2.2.1	Segmentación basada en Clústeres	7
2.2.2	Extracción de Características	8
2.2.3	Aprendizaje Automático (Machine Learning)	8
2.2.4	Filtros Geométricos	8
2.2.5	Análisis de Vecinos Cercanos	8
2.3	Herramientas	9
2.3.1	QT Reader	9
2.3.2	PADL	9
2.3.3	Open3D	9
2.3.4	WhiteBox Tools	10
2.3.5	Python	10
2.3.6	Conda	10
3	Metodología	11
3.1	Introducción a la metodología	11
3.2	Desarrollo Incremental	11
3.2.1	Incrementos	12

4 Planificación	13
4.1 Iteración 1 : Investigación y Preparación Inicial	13
4.2 Iteración 2 : Preparación del entorno de Desarrollo	14
4.3 Iteración 3 : Desarrollo de una primera versión	14
4.4 Iteración 4 : Desarrollo de una usando Estratos	14
4.5 Iteración 5 : Mejora del Sistema de Centroides	15
4.6 Iteración 6 : Pruebas y análisis de los test	15
4.7 Iteración 7 : Realización de la Memoria	15
4.8 Costes	16
5 Detección de árboles basada en estratos	19
5.1 Estructura del Algoritmo	19
5.1.1 Obtención de las Alturas Normalizadas	20
5.1.2 Obtención del Last Return	20
5.1.3 Eliminación del suelo	20
5.1.4 Creación de un Mapa de Alturas	21
5.1.5 Detección usando Máximos	22
5.1.6 Detección de árboles mediante árboles	23
6 Resultados Experimentales	28
6.1 Entorno de pruebas	28
6.1.1 Obtencion del tile de un repositorio	29
6.1.2 Preparación del Tile	29
6.1.3 Recortar Tile en Tiles mas pequeños	29
6.1.4 Obtencion del Ground Truth	30
6.2 Equipo de Pruebas	31
6.3 Parámetros de Prueba	31
6.4 Resultados	32
7 Conclusións	41
7.1 Conclusiones	41
7.2 Trabajo Futuro	41
A Material adicional	43
Bibliografía	45

Índice de figuras

1.1	Gráfico con las principales causas de la perdida de bosques	2
2.1	Ejemplo de un sistema LiDAR en un UAV [1]	5
2.2	Ejemplo del funcionamiento del algoritmo de clustering <i>dbscan</i> [2]	7
2.3	Ejemplo de la red PointNet [3]	8
2.4	Interfaz de QR Reader	9
3.1	Ejemplo de Kanban para el desarrollo de la memoria	12
3.2	Ejemplo de un desarrollo incremental [4]	12
4.1	Primeras 4 iteraciones del Diagrama de Gantt	17
4.2	Últimas iteraciones del Diagrama de Gantt	18
5.1	Diagrama de Flujo del Algoritmo	19
5.4	Ejemplo de Obtencion de arboles mediante maximos	24
5.5	Ejemplo de un árbol de centroides	25
5.6	Ejemplo de puntos detectados como arboles, puntos negros son un árbol . . .	27
5.7	Ejemplo del proceso de obtención de los slices	27
6.1	Diagrama de flujo para obtención del dataset	28
6.3	Ejemplo de tiles recortados	30
6.4	Programa para marcar los arboles	31
6.5	Matriz de confusión con la manera de obtener las métricas usadas.[5] . . .	34
6.6	Ejemplo de Detección en Coníferas	35
6.7	Ejemplo de los centroides que son detectados como árboles	36
6.8	Ejemplo de los centroides que no son detectados como árboles	36
6.10	Ejemplo de Detección en Caducifolios	39
6.11	Ejemplo de los centroides que forman una linea para caducifolios	40

Índice de tablas

2.1	Códigos de clasificación LAS definidos por ASPRS para las versiones LAS 1.1 a 1.4	6
4.1	Tabla con el desglose de los costes humanos	16
4.2	Tabla con el desglose de los costes Materiales	16
4.3	Tabla con el coste total	17
6.1	Tabla con el desglose de los costes humanos	31
6.2	Tabla con el valor de las constantes	32
6.3	Valores de la matriz de confusión	34
6.4	Métricas obtenidas con los valores de la matriz	34
6.5	Tabla con el valor de las constantes para el test en caducifolios	38
6.6	Valores de la matriz de confusión en el nuevo dataset	38
6.7	Métricas obtenidas con los valores de la matriz en el nuevo dataset	38

Capítulo 1

Introducción

EN este capítulo, exploraremos la motivación detrás de este trabajo, proporcionando un análisis detallado de los propósitos establecidos al inicio del proyecto y, por último, se presentarán los apartados que compondrán la memoria.

1.1 Motivación

Los bosques son una parte fundamental de nuestra vida cotidiana. Proporcionan hábitats para muchas especies, regulan el ciclo del agua, ayudan a moderar el cambio climático al absorber dióxido de carbono y proporcionan importantes recursos naturales como madera y alimentos.

Sin embargo, como podemos ver en la figura 1.1 obtenida en la web de *Our World in Data* [6] la deforestación los esta destruyendo. Entre los motivos para esto esta la creación de nuevos terrenos edificables o las substitución del cultivo como esta ocurriendo en indonesia con las palma aceitera. Si pudiéramos tener una lista de árboles en diferentes bosques, podríamos identificar estos eventos y buscar soluciones para mitigar las consecuencias de la pérdida de estos entornos.

En la actualidad, la realización de inventarios y de árboles se lleva a cabo de manera manual, lo que implica una inversión de mucho tiempo y recursos humanos considerables. Esta tarea se vuelve aún más desafiante en países donde la cobertura forestal puede representar hasta el 67% del territorio, como es el caso de Suecia [7].

En este contexto, entraría en juego la tecnología LiDAR, que aprovechando las nubes de puntos obtenidas mediante sensores equipados en vehículos aéreos no tripulados (UAV) y mediante la aplicación de algoritmos avanzados y técnicas , como el aprendizaje profundo, nos permitiría automatizar el proceso de inventariado de los bosques. Este enfoque ha ganado bastante popularidad en los últimos años, impulsado por avances en los sensores LiDAR y la creciente accesibilidad a UAVs.

La adopción de esta tecnología y la automatización en la detección de árboles en nubes de puntos prometen revolucionar la forma en que abordamos la gestión y conservación de los bosques. Al agilizar y perfeccionar el proceso de inventario, podemos detectar y responder de manera más efectiva a eventos como la deforestación.

Identifying the drivers of forest loss

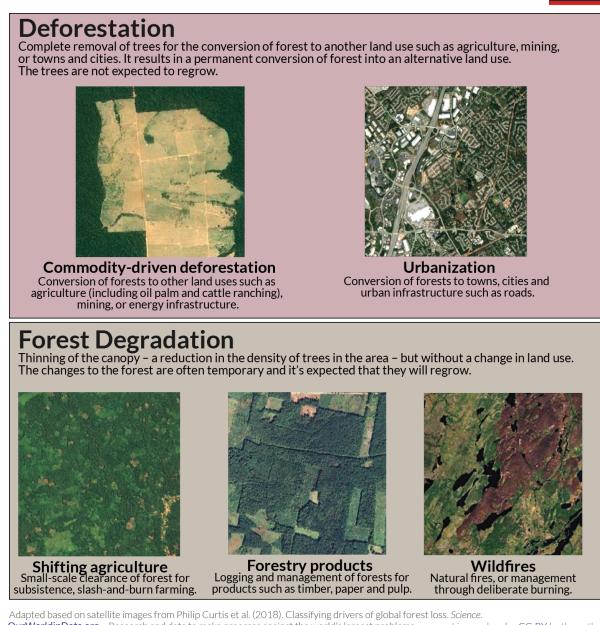


Figura 1.1: Gráfico con las principales causas de la perdida de bosques

1.2 Objetivos

El propósito central de este estudio es llevar a cabo la detección y el conteo de árboles en nubes de puntos LiDAR. La detección se realizará mediante la implementación de algoritmos basados en características geométricas. Estos objetivos principales los podemos desglosar en los siguientes :

- Comprender el funcionamiento de la tecnología LiDAR y los datos obtenidos con ella como las nubes de puntos para poder procesar la información en ellas.
- Buscar conjuntos de datos adecuados, con una cierta densidad de puntos, y aprender a utilizar las principales herramientas para el procesamiento y la visualización de nubes de puntos.
- Investigar las librerías para procesado de nubes de puntos en python.

- Probar diferentes algoritmos y aproximaciones para detectar árboles y estudiar su precisión.
- Analizar y Comparar los resultados obtenidos.

1.3 Apartados de la memoria

La memoria está conformada por los siguientes apartados:

1. **Introducción** : Se introduciremos el proyecto y las motivaciones para llevarlo a cabo. Además, se expondrán los objetivos que perseguiremos durante su desarrollo.
2. **Fundamentos Teóricos y Tecnológicos** : Se expondrá una breve introducción a la tecnología LiDAR y la forma en que se guardan los datos obtenidos de esta.
3. **Metodología** : Comentaremos la metodología elegida para llevar a cabo el trabajo y el por qué de su elección.
4. **Planificación** : Abordaremos la planificación llevada a cabo para el desarrollo de este proyecto, así como el costo que conllevó su ejecución.
5. **Detección de árboles basada en estratos** : Se explicará la implementación del algoritmo para la detección de los árboles.
6. **Análisis de Resultados** : Se probará el algoritmo en diferentes partes del conjunto de datos de Luxemburgo y se estudiarán y analizarán los resultados obtenidos.
7. **Conclusiones y Trabajo Futuro** : En el capítulo final se presentarán las conclusiones obtenidas en el análisis de los datos y se formularán posibles mejoras que podrían llevarse a cabo en el futuro.

Capítulo 2

Fundamentos Teóricos y Tecnológicos

En este capítulo, se expondrán los fundamentos teóricos y tecnológicos necesarios para comprender este trabajo. Además se comentarán las herramientas en las que nos apoyamos para llevar a cabo esta tarea.

2.1 LiDAR

El término LiDAR se proviene de “Light Detection and Ranging” en inglés. Esta tecnología utiliza un pulsoláser para medir con precisión las distancias y crear una representación tridimensionales de un entorno u objeto.

Su funcionamiento es el siguiente, en un emisor emite un pulso láser hacia una superficie u objeto, mientras que un receptor registra el tiempo que tarda en recibir los rebotes de ese pulso. Esto nos permite obtener con precisión la distancia entre el sensor y el objeto. En nuestro caso al trabajar con datos geoespaciales también es necesario que el sistema esté equipado con un GPS para obtener la ubicación geográfica del sensor.

En los últimos años esta tecnología ha experimentado un gran crecimiento debido a su versatilidad y capacidad para ser usado en multitud de escenarios. Entre estos están el uso en vehículos de conducción autónoma [8], gestión forestal, como este trabajo, [9], para arqueología [10], agricultura de precision [11] o para obtener modelos 3d de ciudades [12].

2.1.1 Nubes de Puntos

Las nubes de puntos son el resultado del escaneo de una superficie u objeto, esta nube de puntos no es mas que un conjunto de puntos tridimensionales. Cada uno de estos puntos guarda su coordenada (x, y, z). El par de coordenadas (x, y) representan la ubicación en el plano, y

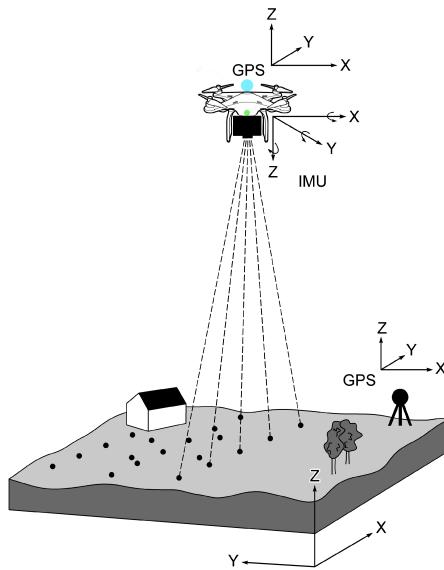


Figura 2.1: Ejemplo de un sistema LiDAR en un UAV [1]

la coordenada (z) la altura de ese punto. Junto con esto también se guardan atributos adicionales para tener mas información de los objetos y superficies. A continuación, explicaremos los más importantes para nuestro trabajo.

Numero de Retornos

Existen escenarios en los que un mismo pulso láser puede impactar en superficies con múltiples capas, como ocurre en el caso de un edificio o un árbol. Cada vez que un rayo regresa al sensor, lo denominamos "retorno".

Un símil que puede ayudarnos a comprender mejor esto sería pensar en cuando lanzamos una pelota contra una pared y esta rebota varias veces antes de ser recogida. Cada rebote representaría un "retorno" de la pelota.

En nuestra nube de puntos, registramos cada uno de estos "rebotes" como puntos individuales. Esta información resulta útil para comprender objetos con múltiples capas, como los árboles, que son el objeto de estudio en este trabajo. Específicamente, como veremos más adelante, utilizaremos el primer retorno, ya que en la mayoría de los casos representa la parte superior del árbol, conocida como "corona arbórea".

Clasificación del objeto

Cada punto en la nube de puntos puede ser clasificado según su origen. Por ejemplo, si los puntos pertenecen al suelo, edificios o vegetación. Para que esta clasificación sea consistente y homogénea entre todas las nubes de puntos la Sociedad Americana de Fotogrametría y

Sensores Remotos (ASPRS) [13] propone los siguiente códigos a la hora de clasificar los puntos 2.1.

Valor de clasificación	Significado
0	Nunca clasificado
1	No asignado
2	Terreno
3	Vegetación baja
4	Vegetación media
5	Vegetación alta
6	Edificio
7	Punto bajo
8	Reservado
9	Agua
10	Ferrocarril
11	Superficie de la carretera
12	Reservado
13	Protector de cable (señal)
14	Conductor de cable (fase)
15	Torre de transmisión
16	Conejero de la estructura de cables (aislante)
17	Plataforma del puente
18	Ruido alto
19-63	Reservado
64-255	Definido por el usuario

Tabla 2.1: Códigos de clasificación LAS definidos por ASPRS para las versiones LAS 1.1 a 1.4

Densidad de puntos

Este valor representa la cantidad de puntos en una región específica, lo que puede proporcionar detalles sobre la complejidad del objeto o la resolución del escaneo. Para este proyecto, es importante tener regiones con altas densidades de puntos para ser capaces de reconocer la

estructura del árbol, tanto la copa como el tronco.

Una vez comprendido qué conforma una nube de puntos, necesitamos entender cómo se guardan estos datos. El formato más ampliamente utilizado es el LAS (LiDAR Data Exchange Format), desarrollado por la ya mencionada Sociedad Americana de Fotogrametría y Sensores Remotos. Este formato es altamente eficiente y permite diferentes niveles de compresión. Al ser el formato más extendido, será el más utilizado por las herramientas de procesamiento y visualización.

Como alternativa a este formato, y también muy utilizado, tenemos el LAZ, un formato que utiliza técnicas de compresión sin pérdida para reducir el tamaño del archivo sin perder precisión. Normalmente, los datos LAS se pueden convertir a LAZ sin ningún problema y posteriormente, para su procesamiento, se pueden volver a descomprimir.

2.2 Detección de Objetos

La detección de objetos en nubes de puntos es una tarea crucial en diversas aplicaciones como la conducción autónoma, la planificación urbana o la robótica. Existen varias aproximaciones para realizar esto, cada una con sus ventajas y limitaciones. Ahora expondremos algunas de las técnicas más comunes usadas en este ámbito:

2.2.1 Segmentación basada en Clústeres

La segmentación basada en clústeres agrupa puntos que están cerca unos de otros en clústeres, lo que permite identificar regiones que probablemente representen objetos o partes del entorno. Métodos como el crecimiento de regiones, K-means, Mean Shift y DBSCAN son ampliamente utilizados para esta técnica. La segmentación basada en clústeres es efectiva para detectar objetos con formas y tamaños variados.

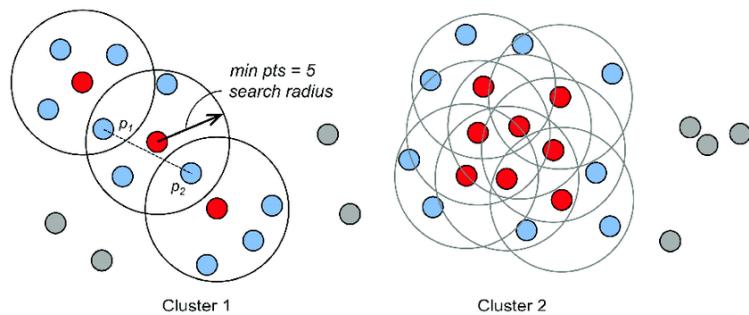


Figura 2.2: Ejemplo del funcionamiento del algoritmo de clustering *dbSCAN* [2]

2.2.2 Extracción de Características

La extracción de características implica la identificación de atributos significativos de los puntos en una nube, como la altura, la intensidad del retorno, la densidad de puntos y la distribución espacial. Estos atributos se utilizan para identificar regiones que podrían contener objetos. Esta técnica es muy versátil y puede combinarse con otras para mejorar los resultados.

2.2.3 Aprendizaje Automático (Machine Learning)

El aprendizaje automático, como el uso de redes neuronales convolucionales (CNN) y métodos de aprendizaje profundo, permite a los sistemas aprender patrones complejos directamente de los datos. Los modelos se entrena con datos etiquetados y luego se utilizan para detectar objetos en nuevas nubes de puntos. Esta técnica es eficaz para capturar relaciones no lineales y características sutiles.

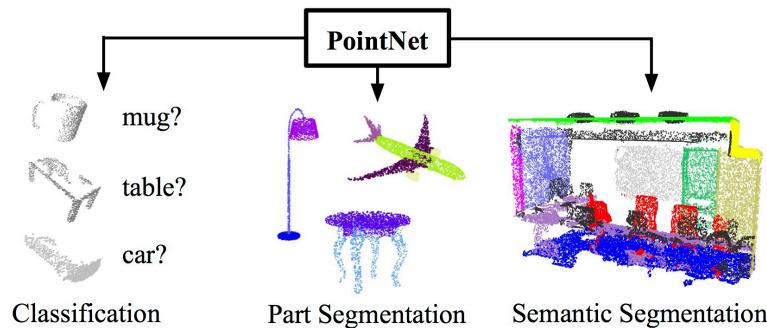


Figura 2.3: Ejemplo de la red PointNet [3]

2.2.4 Filtros Geométricos

Los filtros geométricos son técnicas que eliminan puntos que no pertenecen a objetos sólidos, como el suelo o elementos del entorno. Filtros como el de plano y el de distancia se emplean para mantener solo los puntos relevantes para la detección de objetos.

2.2.5 Análisis de Vecinos Cercanos

El análisis de vecinos cercanos implica evaluar la distribución y proximidad de los puntos en la nube. Los puntos con suficientes vecinos cercanos en un área se consideran parte de un objeto o una superficie. Esta técnica es efectiva para detectar regiones densas y agrupaciones de puntos.

2.3 Herramientas

Para el procesado de nubes de puntos es necesario utilizar herramientas para visualizar y tratar los datos. En la mayoría de los lenguajes de programación tenemos disponibles bibliotecas o librerías que nos lo permiten, como por ejemplo, PDAL, de la cual hablaremos más adelante. A continuación se presentaran las herramientas empleadas durante el desarrollo del trabajo.

2.3.1 QT Reader

Esta es una herramienta gráfica muy sencilla que nos permite visualizar nubes de puntos con una interfaz simple, concisa y veloz. Esta herramienta nos permite utilizar los formatos más comunes, como LAS o LAZ.

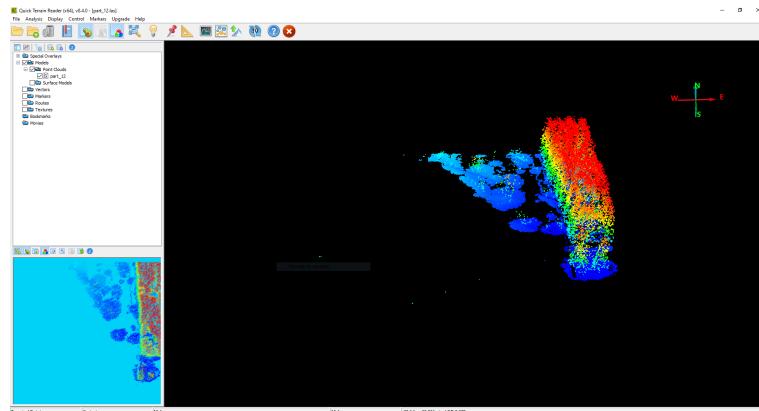


Figura 2.4: Interfaz de QR Reader

2.3.2 PADL

PDAL (Point Data Abstraction Library) es una biblioteca de código abierto desarrollada en C++ para procesar y analizar datos de nubes de puntos en 3D. Nos permite leer, escribir, filtrar y transformar datos en los principales formatos. Además, cuenta con bibliotecas que permiten su utilización en varios lenguajes diferentes [14].

2.3.3 Open3D

Es otra biblioteca de código abierto para el procesamiento y visualización de nubes de puntos. En este proyecto, se utilizó debido a las herramientas que ofrece para la visualización interactiva de las nubes de puntos [15].

2.3.4 WhiteBox Tools

Esta es una herramienta para el procesamiento de nubes de puntos que consta de diversos algoritmos diseñados para el procesamiento de datos LiDAR [16].

2.3.5 Python

Es un lenguaje de alto nivel muy flexible, con una amplia variedad de bibliotecas disponibles tanto para el procesamiento de datos LiDAR como para matemáticas. Este es el lenguaje sobre el cual se basa toda nuestra implementación [17].

2.3.6 Conda

Es un gestor de paquetes y un sistema de administración de entornos para lenguajes como Python. Permite instalar y gestionar paquetes; en nuestro caso, se usó para hacer uso de PDAL [18].

Capítulo 3

Metodología

En este capítulo se expondrá la metodología elegida. En nuestro caso, optamos por una metodología incremental ágil con una gestión del flujo de trabajo mediante Kanban.

3.1 Introducción a la metodología

Para este proyecto se terminó optando por la metodología ágil [19]. En concreto, se optó por un enfoque basado en un desarrollo incremental, donde la gestión del flujo de trabajo se realizó con Kanban.

Este enfoque incremental permite desarrollar pequeños incrementos que van mejorando y puliendo las funcionalidades de nuestro sistema de forma progresiva. Esta metodología se adapta muy a entornos donde los requisitos no están de todo claros en el principio y en el futuro son susceptibles a cambios. Este es el caso de un trabajo de fin de grado donde normalmente se empieza con una idea pero no se sabe muy bien como abordar el problema desde un principio, por lo que tener esta flexibilidad para hacer cambios conforme avance el proyecto y recibamos retroalimentación, es muy útil.

La Gestión de Flujo de Trabajo Kanban se implementó empleando un tablero visual en la plataforma Notion. Esta metodología facilitó una gestión eficiente del flujo de trabajo al proporcionar una representación visual de las tareas, organizadas en categorías de pendientes, en progreso y completadas. Además es muy fácil realizar ajustes en estas tareas según la retroalimentación recibida en las reuniones.

3.2 Desarrollo Incremental

Previamente se introdujo el desarrollo incremental; ahora lo comentaremos con más detalle. Este desarrollo se basa en la premisa de que es más efectivo y adaptable desarrollar algo en partes pequeñas y simples en vez de abordar el proyecto en su totalidad de una sola vez.

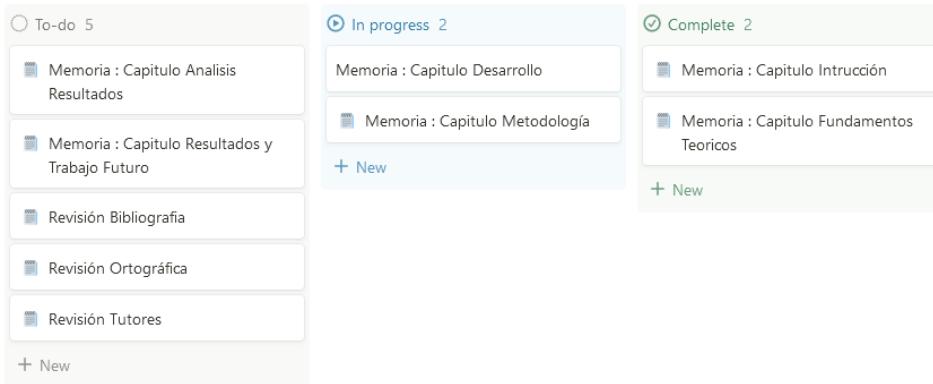


Figura 3.1: Ejemplo de Kanban para el desarrollo de la memoria

Cada incremento se desarrolla en un período de tiempo determinado con un grupo de funcionalidades. Esto nos permite obtener retroalimentación temprana para saber si nos estamos desviando y ajustarnos al resultado que buscamos. Además, al desarrollar en partes pequeñas se reduce el riesgo de desarrollar algo incorrecto y darnos cuenta al final.

3.2.1 Incrementos

Los incrementos son unidades de trabajo que representan un conjunto de funcionalidades entregables. Cada incremento agrega nuevas funcionalidades o corrige las anteriores, y está diseñado para ser funcional por sí mismo. Al final de cada incremento, en nuestro caso, era revisado con los tutores del trabajo, y se discutían las futuras líneas de trabajo sobre él, así como la forma en que podríamos mejorar lo hecho hasta ahora.

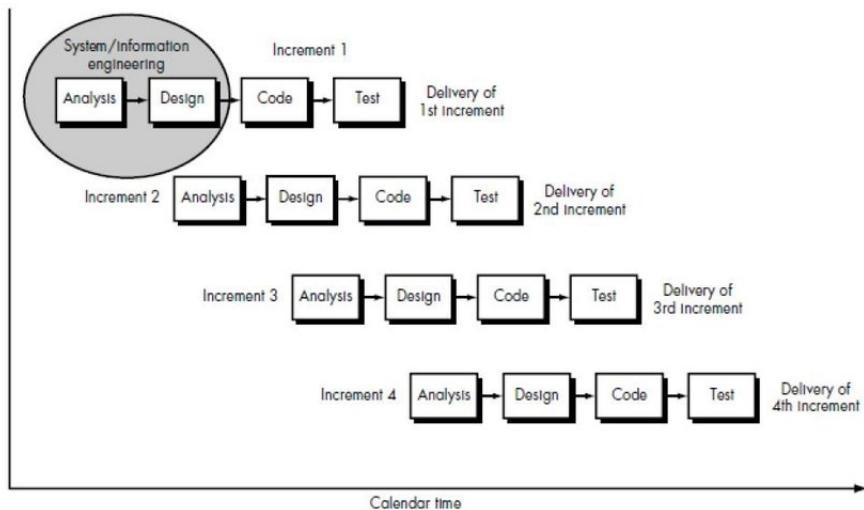


Figura 3.2: Ejemplo de un desarrollo incremental [4]

Capítulo 4

Planificación

Una vez comentada la metodología a usar, discutiremos cómo se llevó a cabo la planificación de nuestro proyecto. A continuación, presentaremos y analizaremos las diferentes iteraciones que conformaron dicha planificación.

4.1 Iteración 1 : Investigación y Preparación Inicial

En esta primera iteración partimos desde cero, aún no tenemos nada construido. Esta etapa tuvo una duración de una semana, del 9 al 16 de mayo de 2023.

Primero se realizaron búsquedas de publicaciones sobre la detección de árboles en nubes de puntos. Durante esta fase, nos encontramos con la publicación [20], de la cual obtuvimos la idea de utilizar máximos como un enfoque inicial. También destacamos la publicación de *Forest Ecosystems* [21], de la cual tomamos la idea de utilizar las secciones del tronco. Además, en paralelo, comenzamos la búsqueda de repositorios de nubes de puntos. En esta búsqueda, tuvimos en cuenta la densidad de puntos, ya que era fundamental poder identificar el tronco en ella, y no lo encontrábamos en bajas densidades. Finalmente, después de considerar varias opciones, optamos por utilizar el repositorio de Luxemburgo.

Después de informarnos y revisar las aproximaciones ya existentes, procedimos a realizar pruebas con la herramienta *WhiteBox Tools* para evaluar diferentes algoritmos destinados a la modificación de nubes de puntos, tales como la eliminación del suelo o la reducción de ruido. Una vez completada esta iteración, concluyó con una reunión con los tutores, durante la cual se discutió la información obtenida y se propusieron nuevos objetivos para la próxima interacción en el repositorio de Luxemburgo.

4.2 Iteración 2 : Preparación del entorno de Desarrollo

En esta segunda fase, se comenzó a buscar el entorno y los lenguajes que se utilizarían para desarrollar el trabajo. Durante esta etapa, se comenzó barajando la opción de usar el lenguaje *Rust* debido a su gran eficiencia. El problema radica en que aprender este lenguaje en poco tiempo es algo complejo, y las bibliotecas disponibles para el procesamiento de nubes de puntos son escasas y tienen poca documentación, por lo que esta opción se descartó.

La otra opción que se barajó fue *Python*. Dado que es un lenguaje más extendido en el mundo del procesamiento de datos y lleva más tiempo en el mercado, ofrece una amplia variedad de herramientas tanto para modificar como para visualizar nubes de puntos. La mayor desventaja de este enfoque es que es un lenguaje de muy alto nivel que, en términos de rendimiento, puede ser el doble de lento. Para solucionarlo inconveniente, existen bibliotecas que permiten llamar a funciones escritas en otro lenguaje, como *C++*, desde nuestro código de *Python*. Un ejemplo de esto es PDAL, que nos permite utilizar el pipeline de procesamiento de nubes de puntos escrito en *C++* desde *Python* o *Conda*. Finalmente, esta fue la opción por la que se optó. Informamos a los tutores de esta decisión y procedimos a la instalación de *Conda* para utilizar PDAL. Esta tarea abarcó desde el 16 de mayo al 23 de mayo de 2023.

4.3 Iteración 3 : Desarrollo de una primera versión

En esta Iteración se procedió a desarrollar una primera versión capaz de convertir la nube de puntos 3d en un matriz 2d que represente la altura de cada punto y a partir de esta matriz obtener los máximos que nos servirán como un punto de partida en la detección. La primera tarea sera convertir ese conjunto de datos 3D en una matriz 2D. Una vez con esta matriz procederemos a buscar los máximos locales dentro de un umbral de vecindad. Esta tarea tomo desde el 23 de mayo hasta el 6 de junio, para terminarla se reunió con los tutores se les presento las nuevas funcionalidades y se propusieron mejoras para una próxima Iteración.

4.4 Iteración 4 : Desarrollo de una usando Estratos

En esta fase se desarrollara el algoritmo que comentamos en la sección 5. Partiremos de los máximos locales obtenidos previamente para buscar en ellos características que puedan confirmar que es un árbol. En nuestro caso se opto por coger los puntos dentro de una sección cilíndrica con centro en el máximo y radio un parámetro que configuraremos en el código, y partiendo de esos puntos usar técnicas como RANSAC o obtener los centroides de diferentes secciones para determinar si es un árbol o no. A mayores de esto para esta parte de análisis de características se le pasara la nube de puntos con alturas normalizadas y sin suelo, esto lo obtendremos gracias al pipeline de PDAL.

Lo ultimo que se implemento esta iteración se seleccionaron unos tiles de prueba y se les asigno un clasificación con la localización de los puntos, para hacer esta tarea se desarrollo una pequeña utilidad que guarde los puntos que nosotros marcamos manualmente como arboles en un CSV. A partir de estos tiles podremos ir obteniendo unas primeras métricas para ver que tan preciso es nuestro sistema.

4.5 Iteración 5 : Mejora del Sistema de Centroides

Anteriormente se creo un sistema para obtener el centroide de diferentes secciones del Estrato y comprobar la linealidad, si el valor obtenido era mayor de cierto umbral se clasificaba ese punto como un árbol. En esta iteracion se planteo y desarollo un sistema para que en cada seccion se aplique un algoritmo de clustering y obtener el centroide de cada cluster, esto se decidió hacer así por que previamente la aparición de ramas podaría dar resultado a un falso negativo. Con este nuevo sistema tendremos un árbol con todos los centroides y al recorrer todas las secciones buscaremos en esa estructura de datos el mejor conjunto de centroides, esto lo veremos mejor en el capitulo [5](#).

A mayores de esto se comenzó a realizar la memoria, en concreto la parte relacionada con el algoritmo comentado previamente.

4.6 Iteración 6 : Pruebas y análisis de los test

Una vez con el sistema desarrollado se paso a la obtención de los tiles y su preparación para realizar pruebas. Se eligieron solo tiles de Luxemburgo de diferentes zonas forestales con diferentes tipos de arboles. Sobre estos se realizaron pruebas y se ejecutaron los algoritmos, con los resultados obtenidos se guardaron y se analizaron posteriormente para incluir los resultados obtenidos en la memoria.

4.7 Iteración 7 : Realización de la Memoria

En esta ultima fase con toda la información obtenida a lo largo del desarrollo del proyecto se realizo el desarrollo de la memoria. Se crearon grafismos y diagramas con el fin de que los lectores entiendan mejor el funcionamiento. Para la realización del diagrama de gantt para mostrar la planificación de la memoria se opto por usar la herramienta de código abierto *Mermaid.js* que nos permite la creación de estos mediante código y css.

4.8 Costes

Una vez con la planificación hecha realizaremos una estimación de los costes de este proyecto. Primero establecemos el coste de los recursos humanos. El coste por hora de un desarrollador junior son aproximadamente 9€/h [22] y el coste promedio de dos profesores doctorados por hora es de 18€/h [23]. El tiempo total lo obtendremos de la planificación y suponemos que se trabajan 3 horas al día y las reuniones con los tutores son de 2h. El coste total lo vemos en la tabla 6.1

Nombre del Recurso	Coste por hora	Horas Trabajadas	Total
Programador Junior	9 €/h	420 h	3780 €
Tutor 1	18 €/h	18 h	324 €
Tutor 2	18 €/h	18 h	324 €
Total			4428 €

Tabla 4.1: Tabla con el desglose de los costes humanos

Ahora nombraremos los costes materiales, en esta parte contaremos tanto los recursos físicos como el ordenador y el coste de las licencias del software usado. Primero empezaremos por el Ordenador, estos tiene un ciclo de vida de entre 6 a 10 años y su costo fue de 900 €, si el proyecto duró 4 meses y la vida media la definimos en 7 años el coste por mes sería de sobre 11 € al mes. Para el desarrollo se uso el IDE Pycharm de la empresa JetBrains que tiene un coste de 24.99 € al mes [24].

Nombre del Recurso	Coste Mensual	Meses de Uso	Total
Ordenador	11 €/mes	4 meses	44 €
Licencia Pycharm	24.99 €/mes	4 meses	99.96 €
Total			143.96 €

Tabla 4.2: Tabla con el desglose de los costes Materiales

Con todo esto en la figura 4.3 vemos el coste total del proyecto

Nombre del Recurso	Coste
Costes Humanos	4428 €
Costes Materiales	143.96 €
Total	4571.96 €

Tabla 4.3: Tabla con el coste total

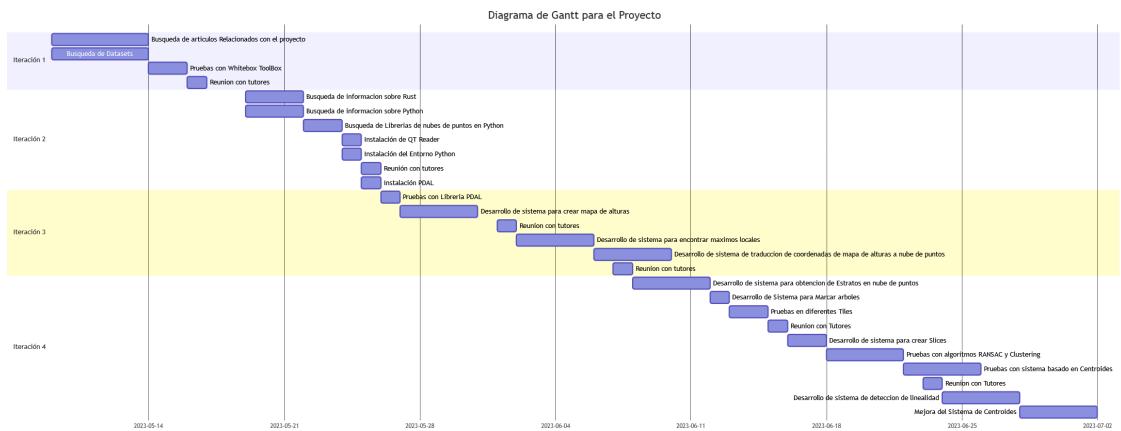


Figura 4.1: Primeras 4 iteraciones del Diagrama de Gantt

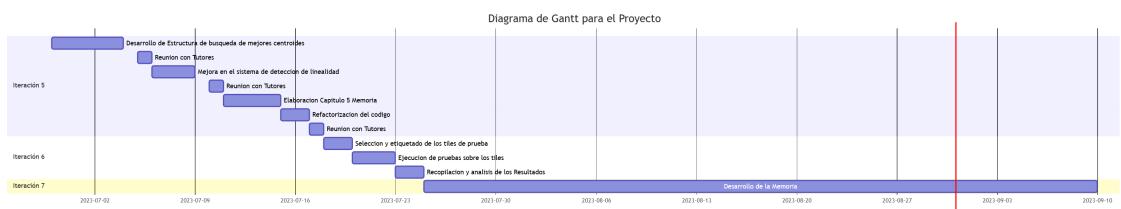


Figura 4.2: Últimas iteraciones del Diagrama de Gantt

Capítulo 5

Detección de árboles basada en estratos

Este capítulo se dedicará a explicar el algoritmo planteado para la detección de árboles en Nubes de Puntos. La figura 5.1 representa el diagrama de flujo del algoritmo empleado para la detección de árboles, cada fase de este tendrá una sección dedicada a él explicando a fondo su funcionamiento.

5.1 Estructura del Algoritmo

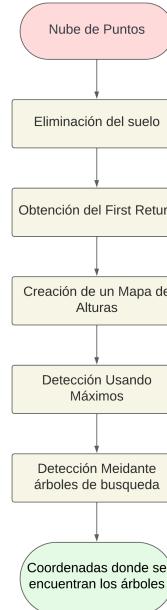


Figura 5.1: Diagrama de Flujo del Algoritmo

5.1.1 Obtención de las Alturas Normalizadas

La obtención de las alturas normalizadas, también conocidas como alturas sobre el suelo (HAG), es un paso esencial para la detección y análisis de objetos elevados, como árboles, en nubes de puntos. Las alturas normalizadas eliminan la elevación del terreno y se centran en la altura relativa de los objetos con respecto a la superficie del suelo. Esto nos ayuda a reducir errores provocados por los diferentes desniveles del terreno y nos ayuda a la comparación de los resultados entre diferentes regiones ya que nos concentraremos únicamente en la verticalidad de los puntos que conforman los árboles.



(a) Nube de puntos sin alturas normalizadas



(b) Nube de puntos con alturas normalizadas

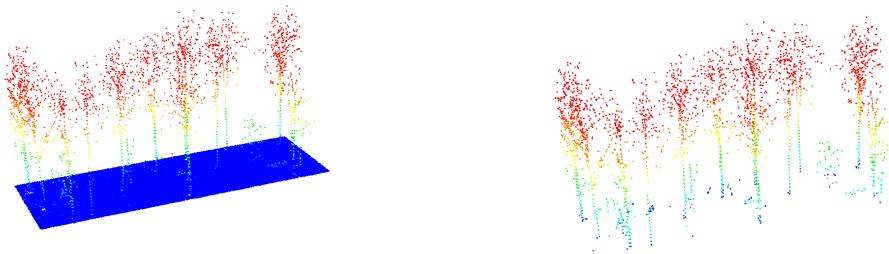
5.1.2 Obtención del Last Return

Esta parte es fundamental en este algoritmo. El uso del último retorno en la detección de árboles mejora la visualización y la caracterización del tronco, debido a que este retorno sufre menos interferencia de la vegetación y ruido. Al alcanzar la cima del árbol, proporciona una visión directa del tronco sin obstrucciones de las hojas y ramas superiores. Esto permite una identificación más precisa de la altura, forma y detalles del tronco, siendo esencial en aplicaciones como análisis de vegetación y modelado del entorno.

5.1.3 Eliminación del suelo

En esta primera fase, el objetivo es eliminar los puntos que corresponden al suelo, ya que no son necesarios para la detección y su inclusión solo requeriría un mayor consumo de recursos. Para este paso, se utilizó el algoritmo basado en simulación de ropa de Zhang, el cual está disponible en PDAL.

El método CSF crea una malla virtual que abarca toda la nube de puntos y luego ajusta dicha malla a los puntos de la nube. A cada punto se le estima la altura de la malla mediante la interpolación de los puntos dentro de cada triángulo. Luego, se identifican los puntos que constituyen ruido al comparar la altitud de los puntos de la nube con la altitud estimada en la malla.



(a) Nube de puntos con los últimos retornos (b) Nube de puntos con últimos retornos sin suelo

Esta aproximación nos permite filtrar los datos de manera sencilla, sin requerir una gran cantidad de parámetros, y proporciona resultados precisos.

5.1.4 Creación de un Mapa de Alturas

El mapa de alturas es una matriz 2D que representa la altura del dosel de los árboles en función de las coordenadas espaciales X e Y.

En primer lugar, se determinan las coordenadas mínimas y máximas del modelo de nube de puntos. Esto proporciona el rango espacial en el que se encuentra la nube de puntos.

A continuación, se crea una cuadrícula regular con una resolución especificada. Esta cuadrícula se forma mediante una serie de puntos espaciados de manera uniforme en el rango determinado por las coordenadas mínimas y máximas.

Luego, se construye un árbol KD (KD-tree) a partir del modelo de nube de puntos. Un árbol KD es una estructura de datos que permite realizar búsquedas eficientes de vecinos más cercanos.

Después de tener la cuadrícula y el árbol KD, se procede a realizar una consulta a los vecinos más cercanos para cada punto de la cuadrícula. Esto implica encontrar el punto más cercano en el modelo de nube de puntos para cada punto de la cuadrícula.

Una vez obtenidos los vecinos más cercanos, se recupera la altura del dosel correspondiente a cada punto de la cuadrícula en relación con el suelo. Esto se logra obteniendo las alturas asociadas a los puntos encontrados en la consulta previa.

Finalmente, los valores de altura del dosel se organizan en un mapa de alturas, donde cada punto de la cuadrícula tiene asignada una altura del dosel correspondiente. Este mapa se usará en la próxima etapa para obtener los puntos donde potencialmente se encuentra un árbol.

Se podría pensar que una forma más rápida de obtener este mapa de alturas sería coger simplemente la altura de cada punto en la nube de puntos, pero el problema de esto es que las nubes son irregulares y existen partes donde hay más densidad de puntos que en otras, donde los puntos muestreados tendrán una distribución no uniforme. Con el fin de tener una representación fiel y precisa, optamos por usar un *KD-tree* y sacrificar un poco de rendimiento.

En el ejemplo 5.4 podemos ver cómo a partir de una nube de puntos, obtenemos su mapa de alturas.

```

1 def compute_canopy_height_model(point_cloud_height_model,
2     resolution):
3     min_coords_height_model = np.min(point_cloud_height_model,
4         axis=0)
5     max_coords_height_model = np.max(point_cloud_height_model,
6         axis=0)
7
8     x_grid = np.arange(min_coords_height_model[0],
9         max_coords_height_model[0], resolution)
10    y_grid = np.arange(min_coords_height_model[1],
11        max_coords_height_model[1], resolution)
12    xx, yy = np.meshgrid(x_grid, y_grid)
13    xy_flat = np.column_stack((xx.ravel(), yy.ravel()))
14
15    kdtree = cKDTree(point_cloud_height_model[:, :2])
16
17    _, indices = kdtree.query(xy_flat, k=1)
18
19    canopy_height = point_cloud_height_model[indices, 2]
20
21    canopy_height_model = canopy_height.reshape(xx.shape)
22    return canopy_height_model

```

Listing 5.1: Código para obtener el mapa de alturas del dorsel

5.1.5 Detección usando Máximos

Haciendo uso del mapa de alturas obtenido antes, aplicando un umbral y un tamaño de filtro específico, se pueden identificar los puntos que tienen potencial de ser árboles.

En primer lugar, se realiza un filtrado de máximo local en el mapa de alturas. Este proceso consiste en examinar cada punto y determinar si es el valor máximo dentro de un vecindario definido por el tamaño de filtro establecido. La distancia de búsqueda del máximo local está determinada por el tamaño del vecindario. Esto lo podemos ver como la variable *filter_size* en la implementación 5.3 y con *threshold*, ambos valores son definidos previamente como constantes.

A continuación, se crea una máscara que identifica los puntos en el mapa de alturas que cumplen dos condiciones: deben ser iguales a los máximos locales encontrados y deben ser mayores que el umbral establecido. Esto se logra mediante una comparación elemento a elemento entre el mapa de alturas y la imagen resultante del filtrado de máximo local.

Por último, se obtienen los índices correspondientes a los puntos que cumplen las condiciones establecidas en la máscara. Estos índices representan las ubicaciones de los árboles detectados en el mapa de alturas.

```

1 def detect_trees_in_max(canopy_height_model, threshold,
2                         filter_size):
3
4     neighborhood_size = (filter_size, filter_size)
5     local_max = maximum_filter(canopy_height_model,
6                                 footprint=np.ones(neighborhood_size), mode='constant')
7     tree_mask = (canopy_height_model == local_max) &
8     (canopy_height_model > threshold)
9     tree_indexes = np.where(tree_mask)
10
11
12 return tree_indexes

```

Listing 5.2: Código para la búsqueda de máximos

La figura 5.4d muestra los puntos máximos dentro del mapa de alturas obtenido previamente

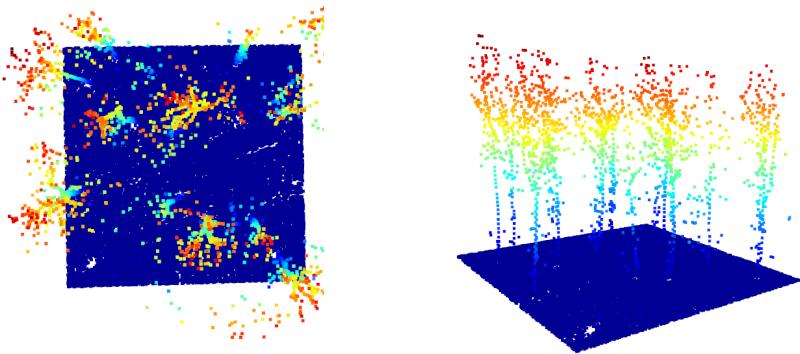
5.1.6 Detección de árboles mediante árboles

Una vez obtenidos los puntos con potencial de ser árboles, obtendremos una sección cilíndrica de puntos con centro en ese punto. Esta sección tiene un radio que determinaremos, y dentro de ella se tienen que encontrar los puntos que conforman ese *árbol*.

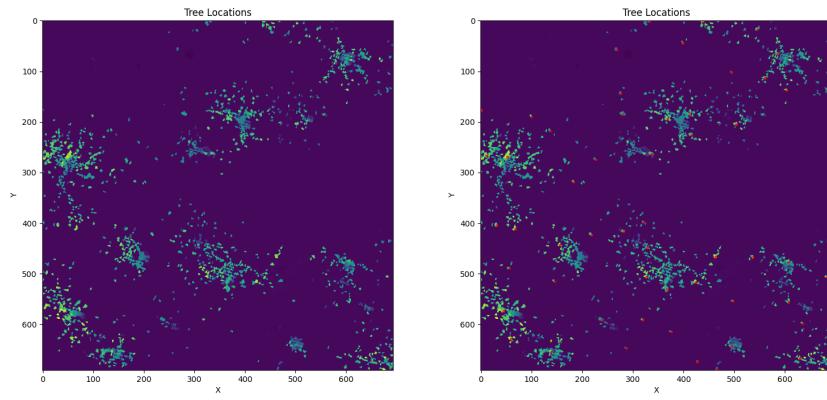
Los troncos de los árboles tienden a tener una estructura que forma una línea, aunque en ciertas especies, llegada cierta altura, comienza a tener muchas ramificaciones; hasta ese punto forma una línea.

Lo que buscamos es ser capaces de encontrar esa linealidad en esa sección. Para eso, la partiremos en *slices* de la misma altura y obtendremos el centroide. Las slices tendrán un tamaño fijo que definimos previamente. Un centroide representa el centro geométrico de los puntos de esa rodaja. Con una lista de estos, obtendremos un score en función de que también se ajuste a una línea, y en función de si es mayor a un umbral o no, se determinará si es realmente un árbol.

Para obtener este valor de que tan bien se ajustan a una linea usaremos el método de los mínimos cuadrados. Este método busca minimizar la suma de los cuadrados de las diferentes ordenadas. La ecuación de una línea en 3D se representa como $z = ax + by + c$, donde z es la coordenada vertical y x e y son las coordenadas horizontales. Los coeficientes a , b , y c se determinan para minimizar la función de error:



(a) Nube de puntos Original desde arriba (b) Nube de puntos Original vista lateral



(c) Mapa de alturas (d) Mapa de alturas con los Máximos

Figura 5.4: Ejemplo de Obtencion de arboles mediante maximos

$$E(a, b, c) = \sum_{i=1}^n (z_i - (ax_i + by_i + c))^2$$

donde n es el número de puntos y (x_i, y_i, z_i) son las coordenadas de cada punto. La optimización se realiza para encontrar los valores de a , b , y c que minimizan esta función, lo que permite construir la ecuación de la línea que mejor se ajusta a los datos.

Si nuestro algoritmo solo tuviera en cuenta eso podrían surgir problemas al encontrarse con ramificaciones o con puntos que aun que estén dentro del cilindro pueden ser de otro árbol o ruido. Con el fin de hacer el algoritmo más robusto, antes de calcular el centroide de cada slice, se aplica un algoritmo al conjunto de puntos que la conforman y se aplica un algoritmo de *clustering*, en nuestro caso usamos DBSCAN, que se basa en la agrupación en la densidad de puntos en una zona. En una primera implementación, en el caso de tener más de

un centroide en una slice podríamos seleccionar solo el que se ajuste mejor a una recta con los centroides previos, pero con esta decisión estaríamos descartando centroides que aunque en el contexto de esa sección se ajustan peor, en el contexto de todo el árbol pueden tener un mejor ajuste.

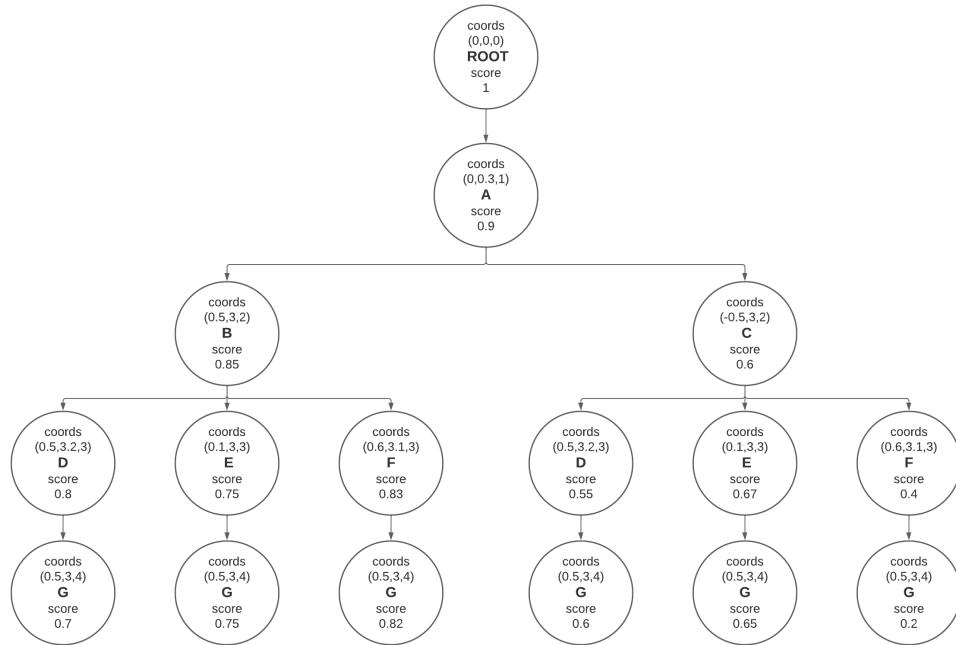


Figura 5.5: Ejemplo de un árbol de centroides

Por lo que para evitar esto se guardan todos en una estructura de árbol que contiene las coordenadas del centroide y el ajuste a una linea con el resto de centroides, a cada nodo se le añaden los slices del siguiente slice. En la figura 5.5 vemos como sería uno de estos árboles. Al tener el árbol lo que haremos será mirar los nodos hoja y mirar cuál es el que tiene mayor score, es decir, que mejor se ajuste a una línea y a partir de ese nodo iremos recorriendo la estructura de datos hasta la raíz para obtener todos los centroides por los que pasa.

Para terminar mostraremos el código que realiza esta parte y lo comentaremos en relación al algoritmos explicado. Lo primero que hacemos para cada puntos es obtener la sección cilíndrica con el radio que definimos. A continuación para evitar detectar arbustos o maleza comprobamos si en ese cilindro hay un mínimo de puntos, este es un parámetro constante definido como constante.

Si tiene mas de el mínimo de puntos se procede a llamar a la función que divide ese cilindro en diferentes secciones y obtiene los centroides mediante el algoritmo que mencionamos antes. Una vez con los centroides volvemos a calcular el ajuste de los puntos y mediante un

umbral decidimos si ese punto es realmente un árbol o no.

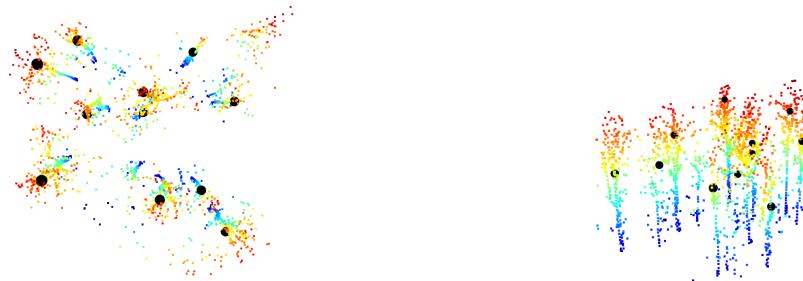
Ahora con la figura 6.3 veremos un ejemplo de como seria el proceso para la obtención de los centroides. La figura 5.7a es la nube de puntos de la sección tubular, en este ejemplo vemos claramente el tronco. A continuación en 5.7b se ejemplifica como dividimos esa nube de puntos en 7 slices, para cada uno obtendremos los centroides y después de entre todos escogeremos los que mejor forme una linea 5.7c y 5.7d.

```

1 def detect_trees(point_cloud, query_coords, radius_threshold):
2     # create a list to save the locations of the trees that are
3     # tubular
4     tubular_tree_locations = []
5     no_tubular_tree_locations = []
6
7     for query_coord in query_coords:
8
9         distances = np.linalg.norm(point_cloud[:, :2] -
10             query_coord[:2], axis=1)
11         filtered_indices = np.where(distances <=
12             radius_threshold)[0]
13         filtered_points = point_cloud[filtered_indices]
14
15         if len(filtered_points) < min_tree_samples:
16             continue
17         else:
18             pcd = o3d.geometry.PointCloud()
19             pcd.points = o3d.utility.Vector3dVector(filtered_points)
20
21             centroids = slice_and_get_points_tree(filtered_points,
22             slice_height)
23             centroids = np.array(centroids)
24
25             if len(centroids) > 4 and not np.isnan(centroids).any():
26
27                 r2 = calculate_r_squared(centroids)
28                 is_line = r2 > r_squared_threshold
29
30                 if is_line:
31                     tubular_tree_locations.append(query_coord)
32                 else:
33                     no_tubular_tree_locations.append(query_coord)
34
35     return tubular_tree_locations

```

Listing 5.3: Código para la búsqueda de máximos



(a) Puntos detectamos arboles vista superior (b) Puntos detectamos arboles vista lateral

Figura 5.6: Ejemplo de puntos detectados como arboles, puntos negros son un árbol

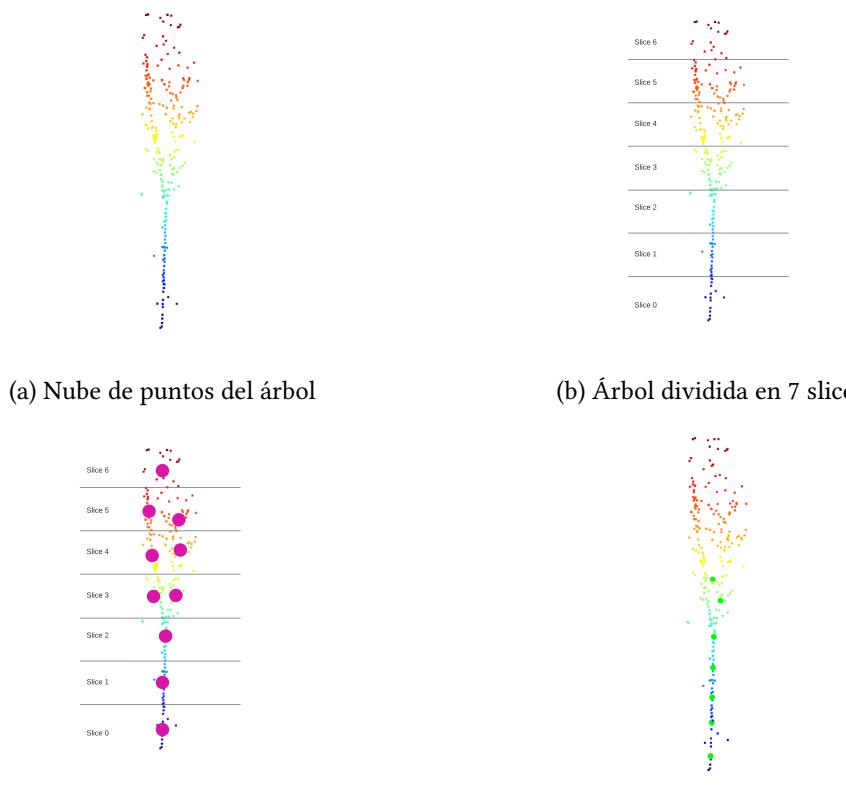


Figura 5.7: Ejemplo del proceso de obtención de los slices

Capítulo 6

Resultados Experimentales

EN este capítulo expondremos los resultados obtenidos al ejecutar nuestro algoritmo en un entorno de pruebas. También explicaremos este entorno y como se obtuvieron los datasets utilizados para la obtención de las métricas.

6.1 Entorno de pruebas

A continuación explicaremos como se obtuvieron los datasets para realizar donde se realizaron estas pruebas. Para la obtención del dataset se siguió un procedimiento que podemos ver en la figura 6.1 ahora explicaremos cada paso

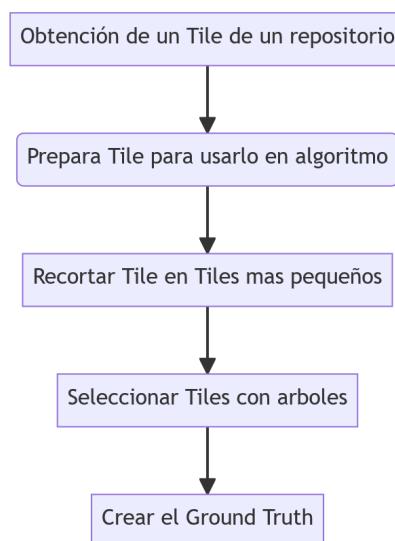
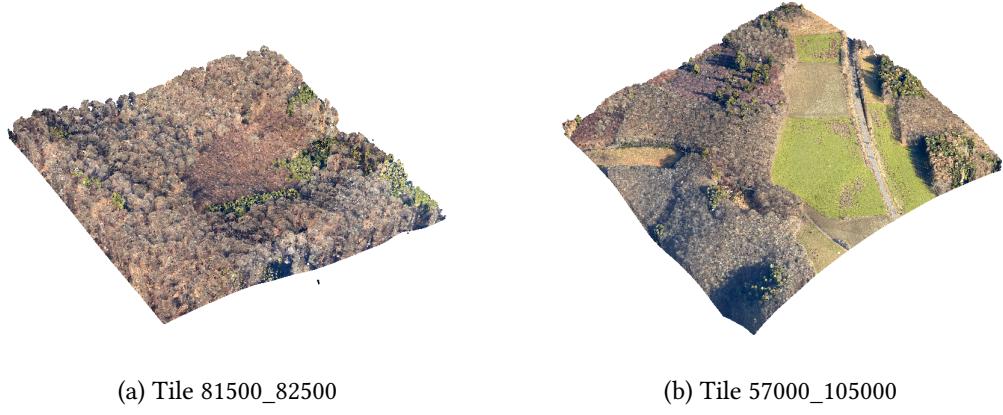


Figura 6.1: Diagrama de flujo para obtención del dataset

6.1.1 Obtencion del tile de un repositorio

Primero tenemos que buscar un repositorio de datos LiDAR y seleccionar los tiles que usaremos. En nuestro caso obtamos por usar el repositorio de Luxemburgo [25] como ya se comento en el sección 4.1. Se escogió este principalmente por su densidad de puntos que cumple con lo requerido y por ser libre para que cualquiera pueda usarlo. Para nuestro dataset escogimos los tiles con identificador 57000_105000 y 81500_82500.



6.1.2 Preparación del Tile

Como ya comentamos en el capitulo 5 necesitamos obtener un tile con las alturas normalizadas y con el ultimo retorno. A partir de eso necesitamos la versión con suelo para el mapa de alturas y otro sin el suelo para la detección. La explicación de como obtenerlos la encontramos en el capitulo 5.1.1 , 5.1.2 y 5.1.3.

6.1.3 Recortar Tile en Tiles mas pequeños

Para el entorno de prueba se decidió optar por recortar los tiles en trozos mas pequeños. Esto se decidió por que el ground truth se hace manera manual y por lo tanto generar uno grande seria prácticamente imposible y se cometerían muchos errores. Además también eliminan puntos que no son de interés como campos de cultivos o casas que lo único que hacen es ralentizar el proceso de prueba.

Para esta tarea se realizó un pequeño programa para recortar un tile grande en tiles mas pequeños cuadrados y todos del mismo tamaño. Después de esto escogeremos manualmente varios donde veamos que haya arboles y con diferentes tipos. En nuestro caso el dataset esta

formado por 70 de estos tiles y con un total de 547 arboles de diferentes especies para que las métricas obtenidas sean reales.

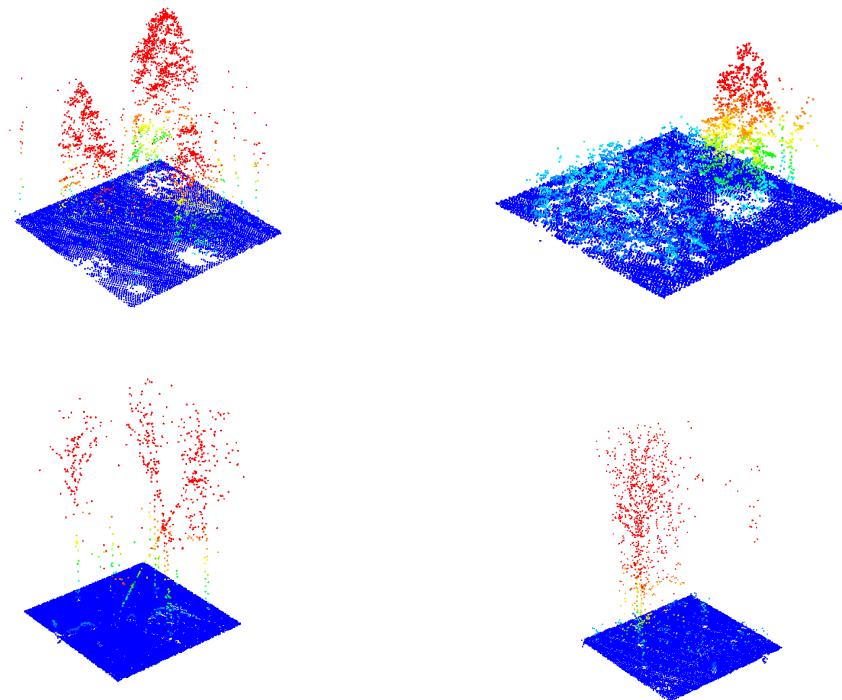


Figura 6.3: Ejemplo de tiles recortados

6.1.4 Obtencion del Ground Truth

Una vez con los tiles preparados necesitamos un ground truth con la ubicación de los arboles. Para esta tarea tambien se desarrollo un pequeño programa para marcar estos mediante una interfaz gráfica como vemos en la figura 6.4. En esta interfaz se nos muestra la nube de puntos y al hacer *Shift + Click Izquierdo* marcamos el punto donde esta el árbol, idealmente marcaremos donde se encuentre el tronco, en los casos donde no se pueda saber bien si es el tronco marcaremos la copa del árbol.

Una vez los seleccionemos todos al cerrar la interfaz se nos generara un archivo csv con las coordenadas de los puntos. Este archivo es el que usaremos en un futuro para comprobar la precisión de la detección.

6.2 Equipo de Pruebas

Una vez con el dataset obtenido, comentaremos el equipo donde se realizaron estas pruebas. Para los test usamos una PC de sobremesa con las especificaciones que podemos ver en esta tabla

Especificación	Valor
CPU	Ryzen 5 1600 6c 12t 3.20 GHz
Memoria RAM	16 gb 2666 mhz
GPU	Nvidia GTX 1060 6 gb
OS	Windows 10 Home 22H2

Tabla 6.1: Tabla con el desglose de los costes humanos

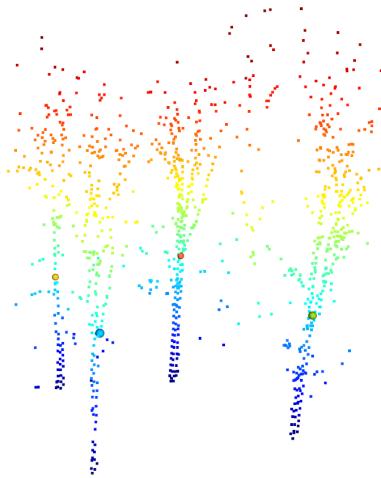


Figura 6.4: Programa para marcar los arboles

6.3 Parámetros de Prueba

Antes de revisar los resultados de las pruebas, es esencial definir los parámetros utilizados en su realización. En primer lugar, estableceremos la resolución del mapa de alturas, que influye en la calidad y la resolución del mapa. Esta variable se establecerá en **0.005**. A continuación, determinaremos el tamaño de la vecindad, en este caso, **80**, y el umbral necesario para determinar si un valor es un máximo, que se fijará en **0.3**.

En lo que respecta a la detección por estratos, estableceremos el número mínimo de puntos requeridos para considerar que un objeto es un árbol, que será **40**. Además, definiremos el tamaño de los "slices" como **1**, y el umbral necesario para determinar si el ajuste de la línea es suficientemente bueno, que será **0.1**.

A continuación, presentaremos estos valores en una tabla 6.2 para una visualización más rápida.

Especificación	Valor
Resolución del mapa de altura	0.005
Tamaño de vecindad	80
Umbral detección en máximos	0.3
Mínimo de puntos por estrato	40
Tamaño de "Slice"	1
Umbral Linealidad	0.1

Tabla 6.2: Tabla con el valor de las constantes

6.4 Resultados

En esta sección comentaremos los resultados obtenidos y las métricas obtenidas tras los test en el conjunto de datos obtenido mediante el proceso anteriormente explicado.

Para poder entender lo que pasa en el sistema necesitamos una manera de cuantificar que tan bien esta funcionando, para esta cuantificación se usaron las métricas mas comunes en el campo de clasificación, la *precisión*, la *especialidad*, la *exactitud* y la *sensibilidad*. A mayores con estés valores obtendremos el *F1 Score*

La obtención se explica en la imagen 6.5, donde a la derecha vemos la matriz de confusión, esta es una herramienta utilizada para evaluar el rendimiento en los modelos de clasificación. Esta matriz organiza las predicciones hechas en un modelo de cuatro categorías:

- **Verdaderos Positivos (True Positives, TP)** : Representa los casos en los que el modelo predice correctamente a que clase pertenece. En nuestro caso seria cuando detecta un árbol correctamente.
- **Verdaderos Negativos (True Negatives, TN)** : Representa los casos en los que el modelo predice que no pertenece a una clase determinada y realmente ese objeto no pertenece a esa clase. Seria por ejemplo cuando detecta que un punto no es un árbol, en nuestro caso este valor no lo podemos obtener.

- **Falsos Postivos (False Positives, FP)** : Representa los casos en los que el modelo predijo incorrectamente a que pertenece a una clase determinada. En nuestro dominio seria cuando detecta que hay un árbol cuando realmente no lo hay.
- **Falsos Negativos (False Negatives, FN)** : Representa cuando el modelo predice que no pertenece a una clase cuando realmente si lo hace. Un ejemplo seria cuando el sistema detecta que no hay un árbol pero realmente si lo hay.

Una vez explicada la matriz de confusión explicaremos el significado de las métricas que hay en esta, usaremos todas menos la especialidad ya que no consideramos el caso no existir arboles.

- **Precisión** : La precisión representa la proporción de predicciones positivas correctas (TP) en relación a todas las predicciones positivas (TP + TF). Esta métrica es útil para identificar que la detección de los arboles sea precisa y confiable.
- **Exactitud** : La exactitud representa la proporción de predicciones correctas (TP + TN) en relación al total de predicciones (TP + TN + FP + FN). Esta metrica nos indica que tambien esta clasificando los arboles que realmente son arboles y los que no son realmente arboles. En casos donde el dataset no este balanceado, por ejemplo donde haya mas especies que puedan favorecer al algoritmo esta metrica no es suficiente.
- **Exhaustividad** : La sensibilidad representa la proporción de predicciones correctas (TP) en relación a todos las predicciones verdaderas reales (TP + FN). Esta métrica nos permite ver si nuestro sistema es capaz de detectar la mayoría de los arboles evitando los falsos negativos.
- **Especialidad** : La especificidad representa la proporción de predicciones correctas (TP) en relación a todos los casos verdaderamente negativos (TN + FP). Esta métrica nos permite evaluar la capacidad de evitar detecciones erróneas en nuestro sistema, como por ejemplo en el caso de detectar un arbusto como árbol.
- **F1 Score** : Esta métrica combina la precisión y Sensibilidad en una sola métrica. Es importante no solo ser capaces de detectar arboles sino tambien asegurar que la detección sea precisa.

$$F1Score = \frac{2 \cdot \text{Precisión} \cdot \text{Exhaustividad}}{\text{Precisión} + \text{Exhaustividad}}$$

Una vez con esto cabe remarcar que este sistema solo detecta si hay arboles, no detecta que no hay, por lo que las métricas como la Especialidad o Exactitud están acotadas para el

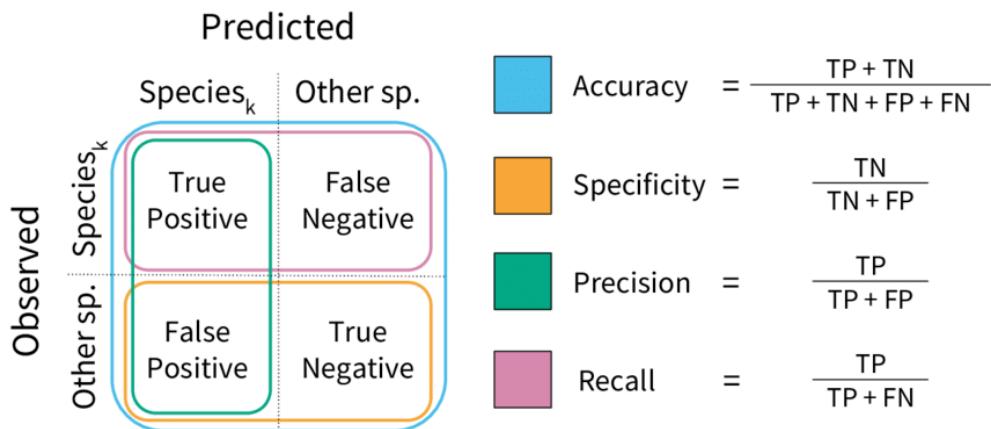


Figura 6.5: Matriz de confusión con la manera de obtener las métricas usadas.[5]

caso exclusivo de existir arboles y no consideran el caso de no existir arboles. Para comprobar si un árbol esta bien clasificado usamos las coordenadas del ground truth y comparamos si el punto que detectado esta cerca de un punto que es detectado como árbol, para esto definimos una distancia de margen de error.

Con esto aclarado procedemos a mostrar los resultados obtenidos en la tabla 6.3 que contiene los valores de la matriz de confusión y en la tabla 6.4 que contiene las métricas.

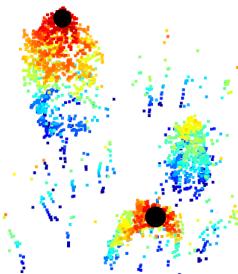
True positives	False Negatives	False Positives
486	300	61

Tabla 6.3: Valores de la matriz de confusión

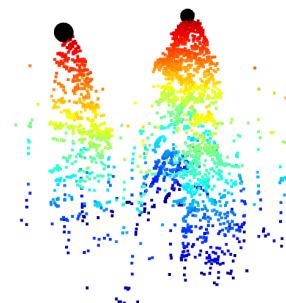
Accuracy	Precision	Recall	F1 Score
59 %	68 %	89 %	70 %

Tabla 6.4: Métricas obtenidas con los valores de la matriz

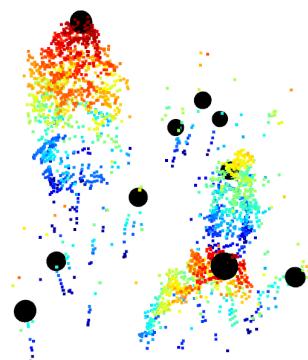
Si miramos la precisión podemos ver como nuestro sistema tiene un valor de 0.59 % en un principio puede parecer baja pero no tiene que ser así. El dataset esta conformado por arboles de los dos principales tipos de bosques que existen en Luxemburgo, bosques caducifolios y bosques de coníferas [26]. Esto se hizo con el fin de tener un conjunto de datos fiel a la realidad del país del cual se tomaron los datos, el problema de esto es que en los bosques de coníferas como las piceas estas tiene mucho mas ramaje mucho mas denso y con las densidades que trabajamos perdemos el tronco al obtener el ultimo retorno, en el caso de bosques caducifolios habitados por robles, castaños y hayas si la densidad de puntos es buena nos permite ver claramente el tronco y nuestro algoritmo obtiene una mayor precisión.



(a) Árboles detectados vista superior



(b) Árboles detectados vista lateral



(c) Ground Truth

Figura 6.6: Ejemplo de Detección en Coníferas

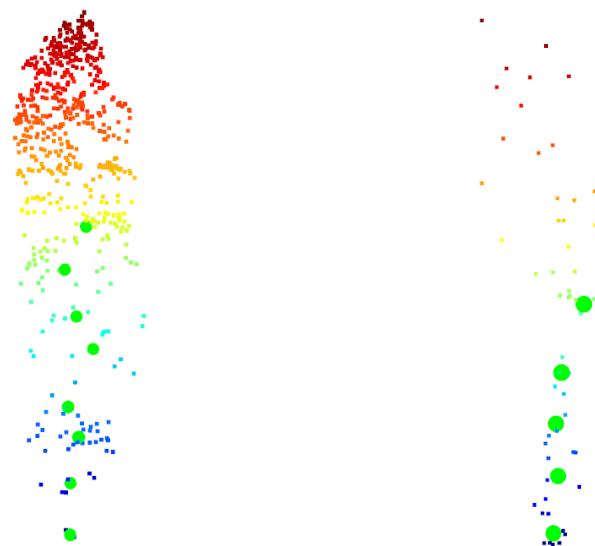


Figura 6.7: Ejemplo de los centroides que son detectados como árboles

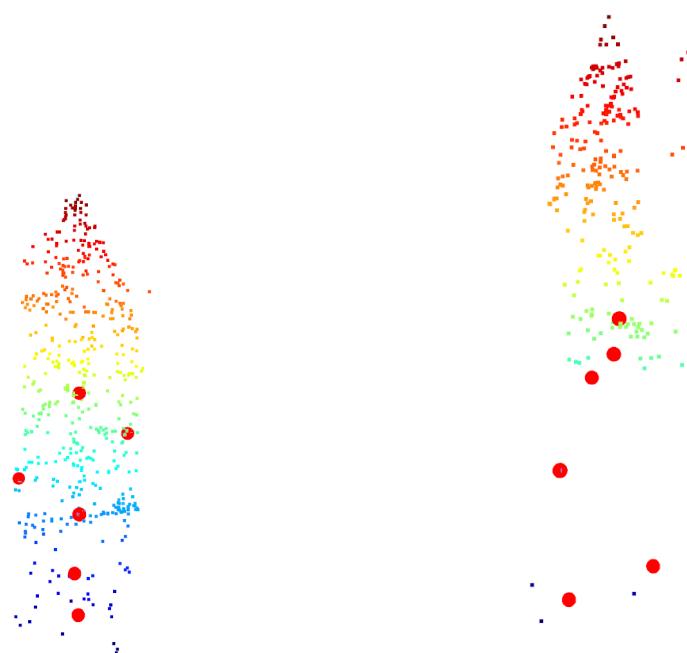


Figura 6.8: Ejemplo de los centroides que no son detectados como árboles



(a) Imagen de un roble



(b) Imagen de una picea

Si observamos el Recall podemos ver como nuestro sistema realmente detecta muchos árboles pero al mismo tiempo muchos son incorrectos, esto puede ser dado por los parámetros utilizados, aquí también influye el tipo de árbol, por lo general en los caducifolios las copas son mas extensas por lo que el tamaño de vecindad es mayor por lo que si el valor fuera demasiado bajo podría detectar varios árboles donde solo existe uno. Por la contra en el caso de coníferas las copas suelen ser mas pequeñas y en un mismo espacio hay mas por lo que el valor de vecindad alto podría dar a que solo obtuviéramos un árbol donde existen varios. Por esto mismo para tener un resultado lo mas realista posible se busco un valor intermedio que no beneficiara en exceso a ninguno de los dos.

En la figura 6.6 podemos ver un ejemplo de un tile donde las coníferas están bastante definidas pero los pequeños árboles como tiene una densidad e puntos muy baja el algoritmo los detecta como si fueran maleza. Además de esto en las figuras 6.7 y 6.8 podemos ver ejemplos de árboles donde los centroides forman una linea bastante definida y ese punto se clasifica como árbol y el caso contrario donde vemos que no tienen forma lineal por lo tanto se rechaza.

Para corroborar esto se escogieron 29 de los 70 tiles anteriores donde predominan los caducifolios y donde la nube de puntos tiene una buena densidad. Esta vez los parámetros los ajustamos para este tipo de bosque y nos quedaron como vemos en la tabla 6.5

Tras ejecutar el mismo algoritmo en este dataset obtenemos los resultados que podemos ver en la tabla 6.6 y 6.7 que podemos ver que son totalmente diferentes.

Aquí vemos como ahora no tenemos una precisión mayor y nuestro sistema es mas exacto, la precisión tiene un porcentaje de mejora del 20 % lo que es bastante, el Recall se mantiene por la cantidad de árboles que detectamos sigue siendo alta. En general podemos ver mejores resultados este tipo de entorno, pero aun así esto podría estar condicionado por la densidad de puntos, para un análisis mas exhaustivo sería necesario tener un conjunto de datos de pruebas

Especificación	Valor
Resolución del mapa de altura	0.005
Tamaño de vecindad	140
Umbral detección en máximos	0.3
Mínimo de puntos por estrato	40
Tamaño de "Slice"	1
Umbral Linealidad	0.1

Tabla 6.5: Tabla con el valor de las constantes para el test en caducifolios

True positives	False Negatives	False Positives
139	15	46

Tabla 6.6: Valores de la matriz de confusión en el nuevo dataset

Accuracy	Precision	Recall	F1 Score
72 %	82 %	90 %	83 %

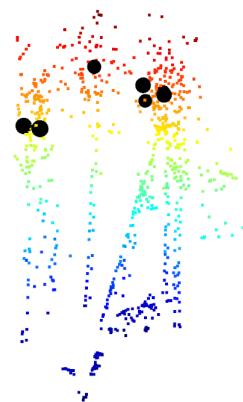
Tabla 6.7: Métricas obtenidas con los valores de la matriz en el nuevo dataset

con varias especies y con densidades de puntos mas altas que estas, estas se podrían obtener mediante escaneos terrestres o con drones con un sensor equipado.

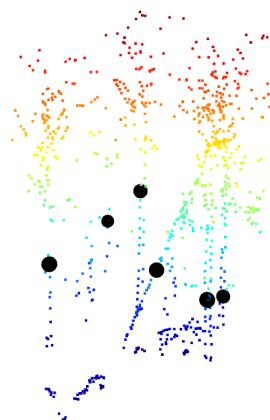
Por último comentar los tiempos de ejecución, la ejecución del entorno de prueba tomo 59 segundos lo que es un valor bastante alto si lo relacionamos con lo que realmente tardaríamos en procesar 2 tiles enteros. Cada Tile lo dividimos en 320 subtiles aproximadamente, es decir los dos tiles serian 740 subtiles. Si hacemos una regla de tres con los 59 segundos que tarda en ejecutar 70 tiles tendríamos que tardaría 10 minutos en procesar los 2 tiles. En el capítulo 7 comentaremos opciones para mejorar este resultado.



(a) Árboles detectados vista superior



(b) Árboles detectados vista lateral



(c) Ground Truth

Figura 6.10: Ejemplo de Detección en Caducifolios



Figura 6.11: Ejemplo de los centroides que forman una linea para caducifolios

Capítulo 7

Conclusións

En este ultimo capitulo se expondrán las conclusiones obtenidas tras realizar este proyecto y los aprendizajes realizados. También Comentaremos que posibles lineas de futuro se podrían seguir para mejorar el trabajo realizado hasta ahora.

7.1 Conclusiones

7.2 Trabajo Futuro

En este proyecto, se emplearon técnicas geométricas para determinar la presencia de árboles. Durante la fase inicial, se consideró la utilización de una red neuronal como *PointNet* [3] para esta tarea. Sin embargo, esta aproximación presentaba un desafío en términos de la necesidad de una amplia y diversa base de datos de miles de árboles de distintas especies, con el fin de asegurar la robustez del sistema. Debido a los plazos ajustados del proyecto, se optó por descartar esta opción.

En relación al rendimiento del código desarrollado, una mejora potencial consistiría en lograr compatibilidad con librerías GPU como *CUDA* o *OpenCL*. Esto permitiría aprovechar la capacidad de procesamiento gráfico para ejecutar en paralelo las numerosas operaciones necesarias en el procesamiento de nubes de puntos. Otra forma de ganar rendimiento sería portar el código a Rust un lenguaje mucho mas rápido y eficiente que Python.

Apéndices

Apéndice A

Material adicional

EXEMPLO de capítulo con formato de apéndice, onde se pode incluír material adicional que non teña cabida no corpo principal do documento, suxeito á limitación de 80 páxinas establecida no regulamento de TFGs.

APÉNDICE A. MATERIAL ADICIONAL

Bibliografía

- [1] C. Torresan, A. Berton, F. Carotenuto, U. Chiavetta, F. Miglietta, A. Zaldei, and B. Gioli, “Development and performance assessment of a low-cost uav laser scanner system (lasuav),” *Remote Sensing*, vol. 10, no. 7, 2018. [En línea]. Disponible en: <https://www.mdpi.com/2072-4292/10/7/1094>
- [2] P.-M. Difrancesco, D. Bonneau, and D. Hutchinson, “The implications of m3c2 projection diameter on 3d semi-automated rockfall extraction from sequential terrestrial laser scanning point clouds,” *Remote Sensing*, vol. 12, p. 1885, 06 2020.
- [3] C. Ruizhongtai Qi, H. Su, K. Mo, and L. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” 12 2016.
- [4] H. Omar, N. O, N. AD, A. Faidi, and A. R. Kassim, “Airborne lidar for estimating above-ground biomass in dipterocarp forests of malaysia,” 10 2015.
- [5] C. Anderson, “The ccb-id approach to tree species mapping with airborne imaging spectroscopy,” *PeerJ*, vol. 6, p. e5666, 10 2018.
- [6] H. Ritchie and M. Roser, “Forests and deforestation,” *Our World in Data*, 2021, <https://ourworldindata.org/forests-and-deforestation>.
- [7] W. B. Group, “Forest area (en: <https://data.worldbank.org/indicator/AG.LND.FRST.ZS?locations=SE>
- [8] S. Royo and M. Ballesta-Garcia, “An overview of lidar imaging systems for autonomous vehicles,” *Applied Sciences*, vol. 9, p. 4093, 09 2019.
- [9] F. Rodríguez-Puerta, E. Gómez-García, S. Martín-García, F. Pérez-Rodríguez, and E. Prada, “Uav-based lidar scanning for individual tree detection and height measurement in young forest permanent trials,” *Remote Sensing*, vol. 14, no. 1, 2022. [En línea]. Disponible en: <https://www.mdpi.com/2072-4292/14/1/170>

- [10] J. Schindling and C. Gibbes, “Lidar as a tool for archaeological research: a case study,” *Archaeological and Anthropological Sciences*, vol. 6, 12 2014.
- [11] G. Rivera, R. Porras, R. Florencia, and J. P. Sánchez-Solís, “Lidar applications in precision agriculture for cultivating crops: A review of recent advances,” *Computers and Electronics in Agriculture*, vol. 207, p. 107737, 2023. [En línea]. Disponible en: <https://www.sciencedirect.com/science/article/pii/S0168169923001254>
- [12] J. Lesparre and B. Gorte, “Simplified 3d city models from lidar,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXIX-B2, pp. 1–4, 07 2012.
- [13] T. A. S. for Photogrammetry Remote Sensing, “LAS Classification Standard,” *ASPRS Standards*, 03 2019. [En línea]. Disponible en: https://www.asprs.org/wp-content/uploads/2019/03/LAS_1_4_r14.pdf
- [14] P. D. Team, *PDAL: Point Data Abstraction Library*, 2012. [En línea]. Disponible en: <https://pdal.io/>
- [15] “Open3d a modern library for 3d data processing.” [En línea]. Disponible en: <http://www.open3d.org/>
- [16] “Whiteboxtools.” [En línea]. Disponible en: <https://www.whiteboxgeo.com/>
- [17] P. S. Foundation, *Python Programming Language*, 1991. [En línea]. Disponible en: <https://www.python.org/>
- [18] I. Anaconda, *Conda: Package and Environment Manager*, 2012. [En línea]. Disponible en: <https://conda.io/>
- [19] B. Carvalho, C. Henrique, and C. Mello, “Scrum agile product development method - literature review, analysis and classification,” *Product: Management Development*, vol. 9, pp. 39–49, 01 2011.
- [20] Y. Pu, D. Xu, H. Wang, X. Li, and X. Xu, “A new strategy for individual tree detection and segmentation from leaf-on and leaf-off uav-lidar point clouds based on automatic detection of seed points,” *Remote Sensing*, vol. 15, no. 6, 2023. [En línea]. Disponible en: <https://www.mdpi.com/2072-4292/15/6/1619>
- [21] Y. Zhang, Y. Tan, Y. Onda, A. Hashimoto, T. Gomi, C. Chiu, and S. Inokoshi, “A tree detection method based on trunk point cloud section in dense plantation forest using drone lidar data,” *Forest Ecosystems*, vol. 10, p. 100088, 2023. [En línea]. Disponible en: <https://www.sciencedirect.com/science/article/pii/S2197562023000039>

- [22] “Estimacion del salario de un programador junior,” 2023. [En línea]. Disponible en: <https://www.hackaboss.com/blog/cuanto-gana-programador-salario-espana>
- [23] “Tablas retributivas udc,” 2023. [En línea]. Disponible en: https://www.udc.es/es/gobierno/equipo_reitoral/xerencia/servizos/retribucions_seguridade_social_e_accion_social/taboas_retributivas/
- [24] “Costes del software pycharm,” 2023. [En línea]. Disponible en: <https://www.jetbrains.com/pycharm/buy/?var=1#commercial?billing=monthly>
- [25] “Lidar 2019 - relevé 3d du territoire luxembourgeois,” 2019. [En línea]. Disponible en: <https://data.public.lu/en/datasets/lidar-2019-releve-3d-du-territoire-luxembourgeois>
- [26] T. Mirgain, “National forestry accounting plan luxembourg,” Administration de l’environnement, Tech. Rep., 12 2018. [En línea]. Disponible en: <https://environnement.public.lu/dam-assets/documents/for%C3%AAt/nfap/NFAP-2018.pdf>