

**UNIVERSIDADE DE SANTIAGO DE
COMPOSTELA**



ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

**Segmentación de datos LiDAR terrestre
aplicada a seguridad vial**

Autor:
Samuel Soutullo Sobral

Tutores:
David López Vilariño
Francisco Fernández Rivera

Grao en Enxeñaría Informática

Julio 2018

Trabajo de Fin de Grado presentando en la Escuela Técnica Superior de Ingeniería de la Universidad de Santiago de Compostela para la obtención del Grado en Ingeniería Informática



D. David López Vilariño, Profesor del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, y **D. Francisco Fernández Rivera**, Profesor del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela,

INFORMAN:

Que la presente memoria, titulada *Segmentación de datos LiDAR terrestre aplicada a seguridad vial*, presentada por **D. Samuel Soutullo Sobral** para superar los créditos correspondientes al Trabajo de Fin de Grado de la titulación de Grado en Ingeniería Informática, se realizó bajo nuestra tutela en el Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela.

Y para que así conste a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a 1 de Julio de 2018:

El tutor,

El cotutor,

El alumno,

David López Vilariño Francisco Fernández Rivera Samuel Soutullo Sobral

Agradecimientos

A mis padres, Josefa Sobral y Ramón Soutullo por educarme en los valores que me han llevado a donde estoy hoy.

Resumen

El conocimiento del lugar de residencia de los alumnos y la caracterización de los viales urbanos del entorno de los centros educativos, son elementos fundamentales en el diseño de los trayectos peatonales más adecuados para que éstos se desplacen diariamente a sus centros de estudio. Los caminos escolares seguros, que la DGT está intentando promocionar junto con la Federación de Municipios y Provincias, ya se han implantado en diferentes ciudades gracias a técnicos de algunos ayuntamientos y centros de enseñanza interesados en fomentar la movilidad sostenible desde edades tempranas.

El proceso habitual de diseño de estas rutas escolares requiere realizar un estudio individualizado en cada centro mediante encuestas directas a alumnos, responsables educativos y técnicos municipales de movilidad. En función de la ubicación de los domicilios de los alumnos interesados en participar, y considerando los conocimientos del entorno urbano de quienes diseñan la ruta segura, se establecen caminos adecuados a las circunstancias particulares detectadas en cada curso escolar.

Conseguir el domicilio de todos los alumnos de un centro escolar y sus hábitos de movilidad, es una tarea que puede realizarse mediante sistemas web de recopilación de datos geolocalizados. Sin embargo, el conocimiento de forma periódica y automática del estado de los viales en el entorno de los centros educativos, requeriría el empleo de métodos de captura masiva de datos mediante tecnologías de teledetección cuya utilización empieza a implantarse en análisis de ciertas ciudades, pero sobre lo que es necesario investigar para su aplicación en estudios de movilidad.

El proyecto Big-GeoMove, en el que se enmarca este Trabajo de Fin de Grado, plantea crear nuevos indicadores que permitan caracterizar los viales urbanos en función de múltiples fuentes de datos mediante técnicas Big Geo-Data y de análisis geoestadístico, que permitan parametrizar con gran detalle todos los elementos de las calles (aceras, pavimento, bordillos, pendientes, escaleras, arbolado, etc.). Se están empleando técnicas de captura y generación de nubes de puntos (mediante sensores LiDAR y restitución fotogramétrica) para su posterior estudio y clasificación de las geometrías de las calles.

En este Trabajo de Fin de Grado, partiendo de algoritmos ya implementados por el grupo de investigación al que pertenecen los tutores para el análisis de

nubes de puntos LiDAR aerotransportado, se plantea implementar una solución que permita segmentar nubes de puntos LiDAR terrestre para su posterior uso en el reconocimiento de estructuras viales tales como aceras, pavimento, bordillos, etc.

Índice general

1. Introducción	1
1.1. Trabajo realizado	1
1.1.1. Objetivos generales	1
1.1.2. Documentación que conforma la memoria	1
1.1.3. Descripción del sistema	1
1.2. LiDAR	2
1.2.1. Recolección de datos LiDAR	3
1.2.2. Aplicaciones de LiDAR	5
1.3. Estado del arte y motivación	6
2. Planificación y presupuestos	9
2.1. EDT del proyecto	9
2.1.1. Tareas del EDT	9
2.2. Planificación temporal del proyecto	12
2.3. Metodología de desarrollo	12
2.4. Estimación de costes y recursos	14
2.4.1. Costes personales	14
2.4.2. Costes materiales	15
2.4.3. Costes totales	17
2.5. Gestión de riesgos	17
2.5.1. Definiciones	17
2.5.2. Identificación de riesgos	19
3. Especificación de requisitos	25
3.1. Identificación de requisitos	25
3.1.1. Requisitos funcionales	26
3.1.2. Requisitos no funcionales	28
3.2. Casos de uso	29
3.2.1. Diagrama de casos de uso	29
3.2.2. Definición de casos de uso	30
3.3. Formato de ficheros	30
3.3.1. Fichero de entrada	30
3.3.2. Fichero de salida	31

4. Diseño	33
4.1. Algoritmo basado en grafos	34
4.1.1. Algoritmo desarrollado por el grupo de investigación	34
4.1.2. Algoritmo modificado	37
4.2. Algoritmo basado crecimiento de regiones	40
4.2.1. Modificaciones al fichero de entrada	40
4.2.2. Funcionamiento del algoritmo	42
4.2.3. Estructura del programa	46
4.3. Equipamiento <i>hardware</i> y <i>software</i>	48
4.3.1. Equipamiento <i>software</i>	48
4.3.2. Equipamiento <i>hardware</i>	49
5. Implementación	51
5.1. Algoritmo basado en grafos	51
5.1.1. Cálculo de vecinos	52
5.1.2. Segmentación	54
5.2. Algoritmo basado en crecimiento de regiones	57
5.2.1. Cálculo de vecinos	57
5.2.2. Etapa de segmentación	58
6. Resultados	61
6.1. Calidad de las segmentaciones	61
6.1.1. Nube de puntos 1	61
6.1.2. Nube de puntos 2	67
6.1.3. Perspectiva general de la nube de puntos	69
6.2. Comparación de rendimientos	69
6.2.1. Características del equipo	70
6.2.2. Método de medición	70
6.2.3. Evaluación de rendimientos	71
7. Conclusiones y posibles ampliaciones	77
7.1. Conclusiones	77
7.2. Posibles ampliaciones	78
A. Manuales técnicos	79
A.1. Instrucciones de compilación	79
A.1.1. Programa del nuevo algoritmo basado en grafos	79
A.1.2. Programa del algoritmo basado en crecimiento de regiones	79
B. Manuales de usuario	81
B.1. Segmentador basado en grafos	81
B.1.1. Códigos de iteraciones	82
B.1.2. Ejemplos de uso	83

B.2. Segmentador basado en crecimiento de regiones	84
B.2.1. Parámetros opcionales	84
B.2.2. Ejemplos de uso	85
Bibliografía	87

X

Índice de figuras

1.1.	Nube de puntos 3D obtenida mediante tecnología LiDAR.	3
1.2.	Obtención de datos LiDAR de forma aérea.	4
1.3.	Nube de puntos de LiDAR aéreo.	4
1.4.	Nube de puntos de LiDAR terrestre.	5
1.5.	Nube de puntos de LiDAR terrestre sin segmentar.	6
1.6.	Nube de puntos de LiDAR terrestre segmentada mediante el algoritmo de LiDAR aéreo.	7
2.1.	EDT del proyecto.	10
2.2.	Diagrama de Gantt del proyecto.	13
3.1.	Diagrama de casos de uso.	29
3.2.	Ejemplo de fichero de entrada	31
3.3.	Ejemplo de fichero de salida	32
4.1.	Diagrama de flujo del algoritmo de partida.	35
4.2.	Diagrama de flujo del algoritmo modificado.	38
4.3.	Ejemplo de fichero de entrada con los dos nuevos parámetros. . .	41
4.4.	Diagrama de flujo general del nuevo algoritmo.	42
4.5.	Diagrama de flujo de una etapa del nuevo algoritmo.	45
4.6.	Representación estructural del nuevo programa desarrollado. . .	47
5.1.	Bucle principal del nuevo algoritmo de grafos.	52
5.2.	Porción de código asociado al cálculo de vecinos.	52
5.3.	Porción de código asociada al <i>parseo</i> del argumento de entrada. .	54
5.4.	Porción de código asociado al almacenamiento de referencias a características a emplear.	55
5.5.	Estructura de datos en la que se almacena cada punto.	56
5.6.	Bucle de creación y asignación de peso de aristas.	56
5.7.	Código asociado a la fusión de componentes.	57
5.8.	Porción de código asociada a la obtención de vecinos.	58
5.9.	Porción de código asociada a una etapa de segmentación.	59
6.1.	Nube de puntos 1 sin segmentar.	62
6.2.	Nube de puntos 1 segmentada en función de planicidad.	63

6.3. Nube de puntos 1 segmentada en función de planicidad y normales.	64
6.4. Nube de puntos 1 segmentada en función de planicidad, normales e intensidad.	65
6.5. Nube de puntos 1 segmentada con falsos positivos por un uso muy agresivo del parámetro de intensidad.	66
6.6. Nube de puntos 1 segmentada completamente.	66
6.7. Nube de puntos 2 sin segmentar.	68
6.8. Nube de puntos 2 segmentada completamente.	68
6.9. Nube de puntos 2 segmentada desde una perspectiva general.	69
6.10. Ejemplo de salida del comando <code>time</code>	71
6.11. Gráfico comparativo de picos de uso de memoria.	74
6.12. Gráfico comparativo de tiempos de ejecución.	75

Índice de cuadros

1.1.	Relación de documentos anexos	2
2.1.	Costes de recursos humanos.	15
2.2.	Costes asociados a productos físicos.	16
2.3.	Consumo eléctrico de los dispositivos.	17
2.4.	Costes totales del proyecto.	17
2.5.	Definición de impacto de los riesgos.	18
2.6.	Definición de probabilidad de los riesgos.	18
2.7.	Definición de exposición a los riesgos.	18
6.1.	Picos de usos de memoria de los diferentes algoritmos (en gigabytes)	72
6.2.	Tiempos de ejecución de los diferentes algoritmos (en segundos) .	73

Capítulo 1

Introducción

1.1. Trabajo realizado

1.1.1. Objetivos generales

Los objetivos generales del presente Trabajo de Fin de Grado son los que a continuación se enumeran:

1. **OB-01:** Analizar el algoritmo de segmentación de datos LiDAR aerotransportado y la influencia de sus parámetros para obtener las combinaciones de ellos más adecuadas para el nuevo caso de uso que concierne a este proyecto: LiDAR terrestre.
2. **OB-02:** Modificar el algoritmo ya implementado para su uso en el reconocimiento de los tipos de algunas estructuras relacionadas con el objetivo de este proyecto (pasos de cebra, aceras, bordillos, obstáculos variados, etc.).
3. **OB-03:** Validar los resultados obtenidos en términos de la calidad de la segmentación y del rendimiento computacional.

1.1.2. Documentación que conforma la memoria

La tabla 1.1 recoge los documentos que se entregan como anexos a la presente memoria. Para cada uno de ellos se indica el nombre del documento, el *software* de visualización y una descripción del mismo.

1.1.3. Descripción del sistema

El resultado de este trabajo es una aplicación escrita en C que, trabajando con datos de LiDAR terrestre, es capaz de segmentar diferentes estructuras viales (pasos de cebra, aceras, bordillos, obstáculos variados, etc.) que pueden posteriormente servir para identificar caminos seguros escolares.

Nombre del documento	Software de visualización	Descripción
UseCase.mdj	StarUML 2	Diagrama de casos de uso.
Design.mdj	StarUML 2	Fichero que contiene todos los diagramas de diseño: diagramas de flujo de los algoritmos y el diagrama de estructura del programa desarrollado.
Gantt.mpp	Microsoft Project 2016	Planificación temporal, uso y costes de recursos humanos.
Performances.xlsx	Microsoft Excel 2016	Tablas y gráficas de rendimiento de los algoritmos

Cuadro 1.1: Relación de documentos anexos

Para la segmentación de estas estructuras, es necesario analizar características de los puntos que permitan agruparlos. Dichas características incluyen, entre otras, posición, reflectividad o vector normal del plano formado por los puntos vecinos.

1.2. LiDAR

LiDAR (*Light Detection and Ranging*) es un método de sensado remoto que utiliza un láser para medir distancias. Los pulsos de luz son emitidos por un escáner láser, que cuando alcanzan un obstáculo, son reflejados de vuelta hacia dicho escáner. En base a la ubicación del escáner, la dirección del haz, y el tiempo transcurrido entre la emisión del pulso y el retorno al escáner, se puede calcular la posición 3D del punto en el que impactó el haz. Durante el proceso de sensado, el escáner emite millones de pulsos, lo que permite obtener una nube de puntos en tres dimensiones altamente precisa, que se puede usar para estimar la estructura 3D del área escaneada.

A mayores de la ubicación 3D de cada punto, también se suelen trabajar con otros parámetros cuando se tratan datos LiDAR. Uno de los parámetros más usados en el tratamiento de datos LiDAR es la intensidad. Dado que no todas las superficies que se escanean mediante el haz de luz son espejos perfectos, la cantidad de fotones que se reflejan en la superficie escaneada y regresan al escáner, será menor que la cantidad de fotones que fueron emitidos en primer lugar. Diferentes superficies con diferentes propiedades físicas reflejarán en mayor o menor medida el haz de luz emitido por el escáner. Por tanto, a cada punto escaneado se le puede asociar un valor que determine la intensidad del haz de luz retornado por la superficie.

La figura 1.1 muestra un ejemplo de nube de puntos 3D obtenida mediante tecnología LiDAR, en la que se incorpora el parámetro de intensidad. Los puntos más oscuros representan puntos de menor intensidad mientras que los más claros, de mayor intensidad.



Figura 1.1: Nube de puntos 3D obtenida mediante tecnología LiDAR.

1.2.1. Recolección de datos LiDAR

Existen principalmente dos formas de recolectar datos LiDAR: de manera aérea y de manera terrestre. En esta sección se comentará de manera breve en que consiste cada una de estas formas y las principales diferencias entre ambas.

LiDAR aéreo

Se dice que se está usando LiDAR aéreo cuando los datos fueron recogidos mediante un aparato volador que tiene acoplado un escáner LiDAR. Durante el vuelo, el escáner va lanzando pulsos de luz hacia el suelo, que rebotan y son recogidos por el propio escáner tal y como se ha explicado anteriormente. Con esta técnica se obtiene una nube de puntos 3D (modelo) del terreno escaneado desde una perspectiva aérea. La figura 1.2 representa de manera gráfica esta forma de obtención de datos LiDAR.

Las nubes de puntos de LiDAR aéreo, al ser escaneadas desde el aire, no reflejan estructuras verticales tales como paredes o troncos de árboles. Además, la densidad de puntos en este tipo de nubes suele ser menor que cuando se trata de nubes LiDAR escaneadas mediante otros métodos. La figura 1.3 es una nube de puntos LiDAR obtenida de esta forma, en la que se pueden observar estas particularidades.



Figura 1.2: Obtención de datos LiDAR de forma aérea.

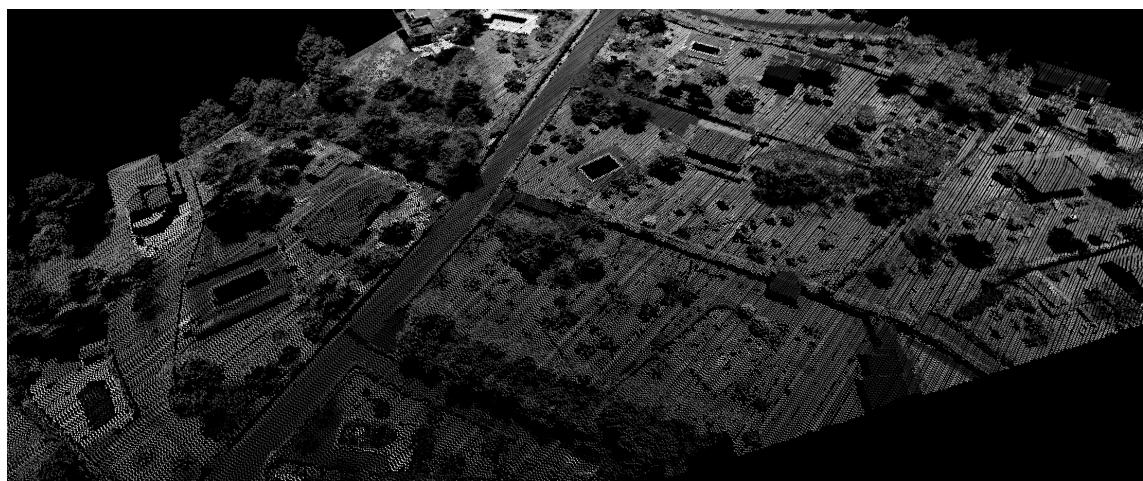


Figura 1.3: Nube de puntos de LiDAR aéreo.

LiDAR terrestre

Se dice que se está usando LiDAR terrestre cuando los datos fueron recogidos mediante un vehículo terrestre en movimiento (por ejemplo un coche) que tiene acoplado un escáner LiDAR. Este cambio de perspectiva con respecto al LiDAR aéreo hace que las nubes de puntos terrestres cuenten con ciertas particularidades, como la presencia de estructuras verticales tales como paredes o troncos de árboles, así como, generalmente, una densidad de puntos considerablemente mayor [2]. La figura 1.4 es una nube de puntos LiDAR obtenida de esta forma, en la que se pueden observar estas particularidades.



Figura 1.4: Nube de puntos de LiDAR terrestre.

1.2.2. Aplicaciones de LiDAR

La recolección y tratamiento de datos LiDAR son útiles cada vez en una mayor cantidad de escenarios distintos. Se podrían destacar, entre otros muchos, los siguientes contextos de aplicación: [1]

- **Agricultura:** puede ayudar a determinar donde se necesita aplicar fertilizante. Mediante un barrido LiDAR se obtiene un modelo sobre el campo de cosecha, sobre el que se pueden determinar zonas de alta, media o baja fertilidad. Esto indicaría donde se debe aplicar fertilizante para mejorar la producción.

- **Vehículos autónomos:** los vehículos autónomos pueden usar tecnología LiDAR para la detección de obstáculos durante la realización de sus trayectos, de forma que puedan transportar a los pasajeros de forma segura.
- **Robótica:** se usa para percepción del entorno y clasificación de objetos. Dado que LiDAR puede proveer un modelo 3D del terreno, puede ser esencial para la movilidad de muchos tipos de robots.
- **Videojuegos:** algunos juegos recientes de simulación de carreras están empezando a incluir circuitos generados a partir de nubes LiDAR 3D, lo que permite replicarlos con precisión milimétrica y de manera automática.

1.3. Estado del arte y motivación

De cara al desarrollo del proyecto, se parte de un algoritmo implementado por el grupo de investigación que dirige este Trabajo de Fin de Grado, utilizado para la segmentación de datos LiDAR aéreo. Como ya se ha comentado anteriormente, los datos LiDAR terrestre tienen ciertas particularidades que no tienen los datos LiDAR aéreo. Como consecuencia, el algoritmo actualmente desarrollado no realiza una segmentación de una calidad lo suficientemente buena como para considerarla válida sobre nubes de LiDAR terrestre. La figura 1.5 muestra una nube de puntos LiDAR terrestre sin segmentar, mientras que la figura 1.6 muestra la misma nube segmentada mediante el algoritmo desarrollado previamente para LiDAR aéreo.



Figura 1.5: Nube de puntos de LiDAR terrestre sin segmentar.

Cada color en la figura 1.6 representa un conjunto de puntos que pertenecen al mismo grupo. Como se puede comprobar, la segmentación dista bastante de

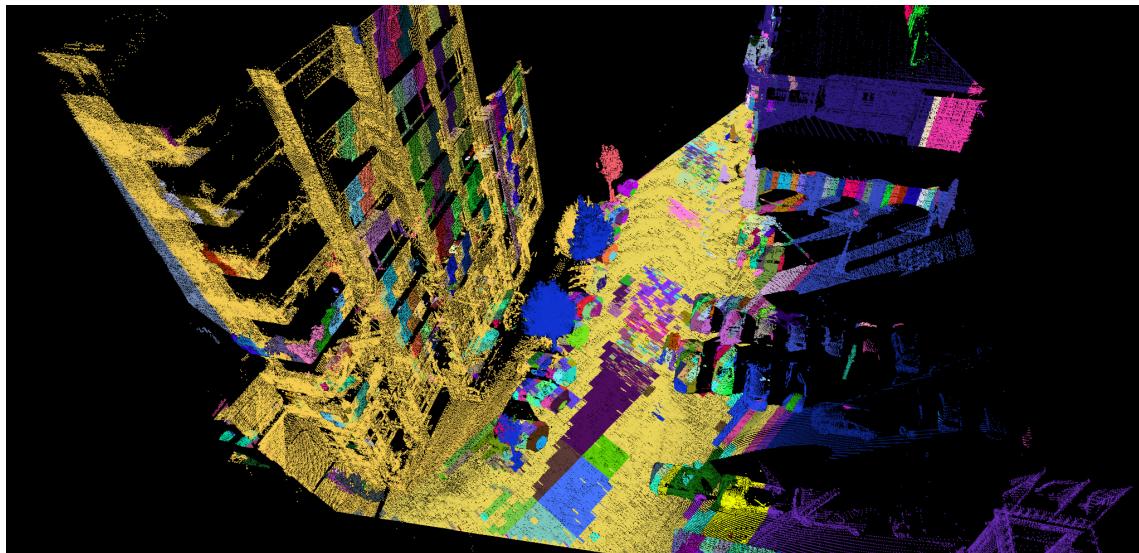


Figura 1.6: Nube de puntos de LiDAR terrestre segmentada mediante el algoritmo de LiDAR aéreo.

ser la óptima. En primer lugar, se produce una sobresegmentación en algunos tramos de la carretera, esto es, se crean más grupos de los que debería en lugares de la carretera que, aparentemente, no son lo suficientemente diferentes entre sí como para que deban ser separados en distintos grupos. Sumado a esto, no hay rastro del paso de peatones, así como otras marcas viales, que se mezclan con los grupos de la carretera.

Por otra parte, en lo que se refiere al rendimiento, también se pueden realizar mejoras. Dado que el algoritmo fue inicialmente concebido para trabajar con nubes de puntos de LiDAR aéreo, que suelen tener muchos menos puntos por área de superficie, no es posible realizar segmentaciones sobre nubes de puntos de más de 10 millones de puntos en la gran mayoría de equipos de escritorio, debido a un alto consumo de memoria. En otras palabras, el consumo de memoria escala demasiado rápido en función del tamaño de la nube de puntos a segmentar.

Finalmente, también hay margen de mejora con respecto al funcionamiento de ciertas partes del algoritmo. Una de ellas es el funcionamiento de las etapas. En una primera etapa se juntan todos los puntos con sus vecinos similares para formar grupos. Para mejorar la segmentación, se ejecuta una segunda etapa que funciona de manera similar, pero tomando como entrada los grupos generados en la etapa anterior, en lugar de todos los puntos de la nube. El problema reside en que la flexibilidad del código en este sentido es bastante baja. Para ejecutar las dos etapas se debe invocar al programa de forma manual en dos ocasiones. Si se desearan hacer más de dos etapas, el código actual no lo permite. Además, se generan bastantes ficheros temporales. Por tanto, se puede abordar la optimización del código en este sentido.

Capítulo 2

Planificación y presupuestos

2.1. EDT del proyecto

En esta sección se mostrará y explicará el EDT del proyecto. En otras palabras, se enunciarán las tareas asociadas al desarrollo de dicho proyecto, y se dará una breve explicación sobre cada una de ellas. La figura 2.1 recoge la EDT (estructura de desglose del trabajo) del proyecto.

2.1.1. Tareas del EDT

En esta sección se explicarán las tareas del EDT. En concreto, solo las tareas hoja (las que en la figura 2.1 no cuentan con ninguna hija), dado que ello será suficiente para entenderlas todas, pues el resto de tareas cumplen una simple función de agrupación de tareas más pequeñas.

1. **Análisis bibliográfico:** se leerá y estudiará diferente documentación (manuales, *papers* del grupo de investigación, etc.) que pueda resultar útil o incluso esencial para el desarrollo del proyecto.
2. **Análisis de requisitos:** se determinarán de manera precisa e inequívoca los requisitos del programa que se va a desarrollar. Esto incluye, entre otros muchos aspectos, que entradas va a recibir, que salidas se esperan, y con qué rendimiento se espera que opere el programa.
3. **Estudio de funcionamiento del código:** dado que en principio se partirá de un código ya desarrollado por el grupo de investigadores que dirige este proyecto, resulta imprescindible entender como éste opera. Es preciso comprender de manera muy profunda el funcionamiento del algoritmo implementado mediante dicho código.
4. **Estudio exhaustivo de la influencia de los parámetros en los resultados:** una vez entendido el algoritmo implementado previamente por los

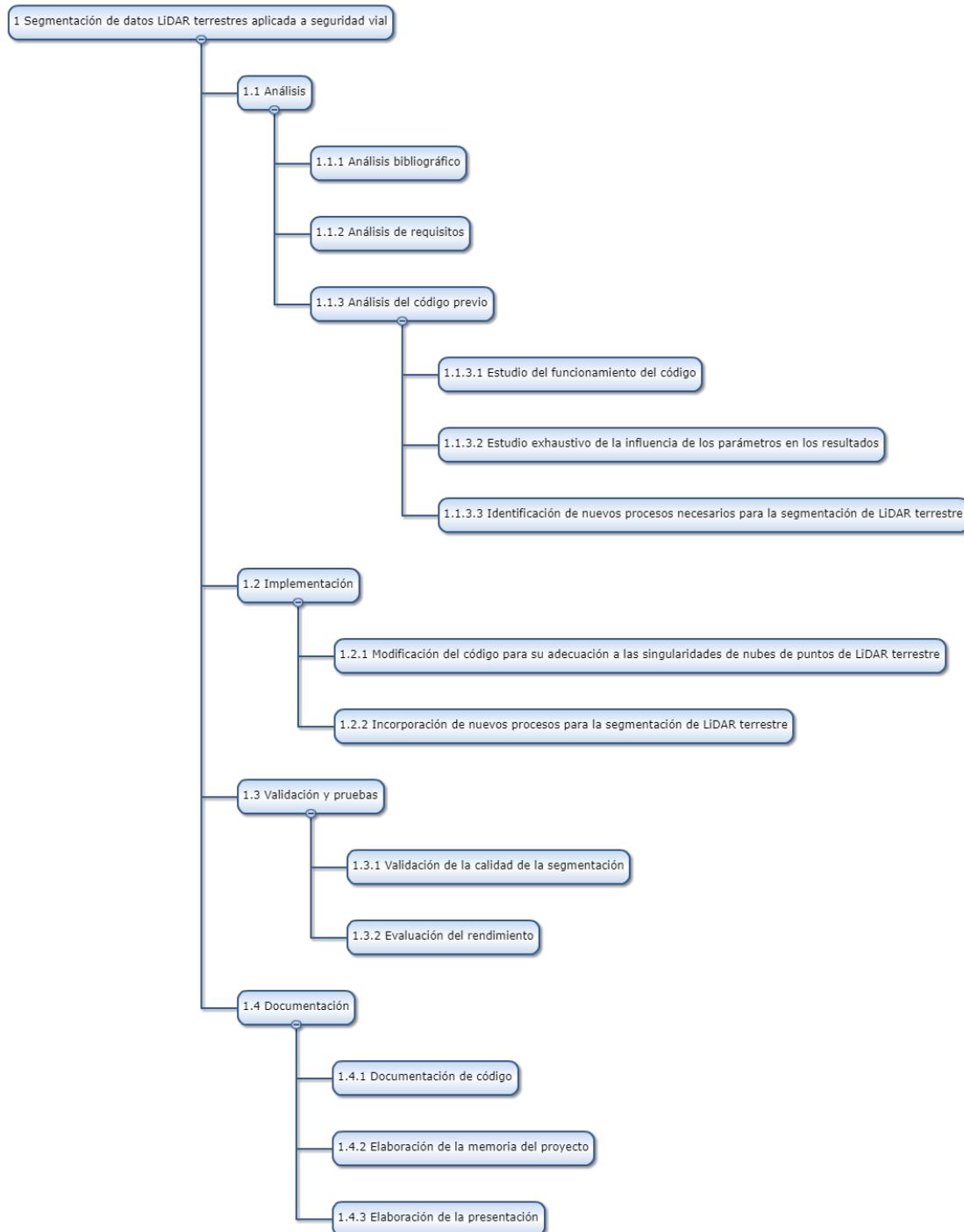


Figura 2.1: EDT del proyecto.

miembros del grupo de investigación, será preciso realizar diferentes pruebas variando los diferentes parámetros con los que cuente dicho algoritmo para obtener una perspectiva precisa de como estos afectan al funcionamiento del mismo.

5. **Identificación de nuevos procesos necesarios para la segmentación de LiDAR terrestre:** dadas las diferencias entre LiDAR aéreo y LiDAR terrestre, y que el código desarrollado por el grupo de investigación está pensado para LiDAR aéreo, es posible que haya que desarrollar de cero nuevas partes del programa ya implementado o incluso un nuevo programa. Es imprescindible tratar de identificar estos nuevos procesos lo antes posible.
6. **Modificación del código para su adecuación a las singularidades de LiDAR terrestre:** dadas nuevamente las diferencias entre LiDAR aéreo y terrestre, parte del trabajo será modificar ciertas partes del código ya implementado para que funcione correctamente en nubes de LiDAR terrestre.
7. **Incorporación de nuevos procesos para la segmentación de LiDAR terrestre:** tras la identificación de nuevos procesos mencionada anteriormente, será necesario implementarlos para incorporarlos a la segmentación de los datos LiDAR terrestre.
8. **Validación de la calidad de la segmentación:** una vez finalizado el desarrollo, será necesario validar el nuevo programa obtenido. La principal forma de hacer esto es mediante la validación de la calidad de la segmentación que se obtiene como resultado de la ejecución del mismo. Ante la imposibilidad de realizar dicha validación de forma automatizada, la forma de proceder será la visualización de los datos segmentados y la observación minuciosa de dicha visualización para comprobar que se corresponde con lo que se espera.
9. **Evaluación del rendimiento:** el nuevo programa deberá funcionar con nubes de varias decenas de millones de puntos. Es imprescindible que la segmentación se realice correctamente, y además en un tiempo razonable. Se debe comprobar que esto se cumple. La definición de tiempo razonable en este contexto se fijará posteriormente, en el capítulo de especificación de requisitos (capítulo 3).
10. **Documentación de código:** dado que se van a desarrollar nuevo código, es imprescindible quede debidamente documentado para facilitar futuras revisiones o modificaciones. Basándose en la forma de trabajo actual del grupo de investigación, esto se llevará a cabo principalmente mediante la inclusión de comentarios en el código y la redacción de pequeños ficheros de texto plano.

11. **Elaboración de la memoria del proyecto:** redacción de la presente memoria.
12. **Elaboración de la presentación:** elaboración de la presentación que se empleará en el momento de la defensa del Trabajo de Fin de Grado.

2.2. Planificación temporal del proyecto

Partiendo de que las tareas necesarias para la realización del proyecto se recogen en el EDT, de cara a la planificación del desarrollo solo falta determinar que tareas se realizará en que momento. Para ello se debe no solo estimar una duración de las mismas, sino también determinar las dependencias existentes que impidan que varias tareas se realicen de forma paralela. Todo ello se recoge en diagrama de Gantt plasmado en la figura 2.2.

Se ha estimado un tiempo de trabajo de aproximadamente 16 semanas, con un total de 26 horas de trabajo por semana. Esto es, un total de 416 horas, o lo que es lo mismo, una jornada laboral de 5,2 horas de lunes a viernes.

Como se puede observar, prácticamente ninguna tarea tiene asignado el 100 % de un recurso humano. Esto se debe a que la memoria se elabora en paralelo al desarrollo del trabajo. En particular, se puede comprobar que el alumno dedica un 75 % de la jornada a la realización de toda tarea no relacionada con el desarrollo de la memoria, y un 25 % a la elaboración de dicha memoria.

2.3. Metodología de desarrollo

Para el desarrollo de este proyecto se partirá de un algoritmo ya desarrollado por el grupo de investigación al que pertenecen los tutores del proyecto. El objetivo será, en principio, modificar dicho algoritmo para adaptarlo a una nueva casuística.

Tras realizar un estudio de varias metodologías, se ha decidido que la metodología más adecuada en este contexto es la de un ciclo de vida basado en incrementos. En él, cada incremento constituye un período de desarrollo que concluye con una versión usable del programa que se está desarrollando, la cual debería ser mejor que la anterior y ofrecer más funcionalidades, o en el peor de los casos igualar sus resultados.

En este sentido, la metodología incremental congenia bastante bien con la naturaleza de este proyecto. En cada incremento se realizarían diferentes modificaciones al algoritmo que supondrían una mejora a versiones anteriores. Este proceso culminaría en el último incremento, en el que se desarrollaría la versión final del algoritmo.

Por último, dado que el alumno mantendrá reuniones con los tutores de proyecto cada 15 días, cada una de estas reuniones podría servir para validar los

2.3. METODOLOGÍA DE DESARROLLO

13

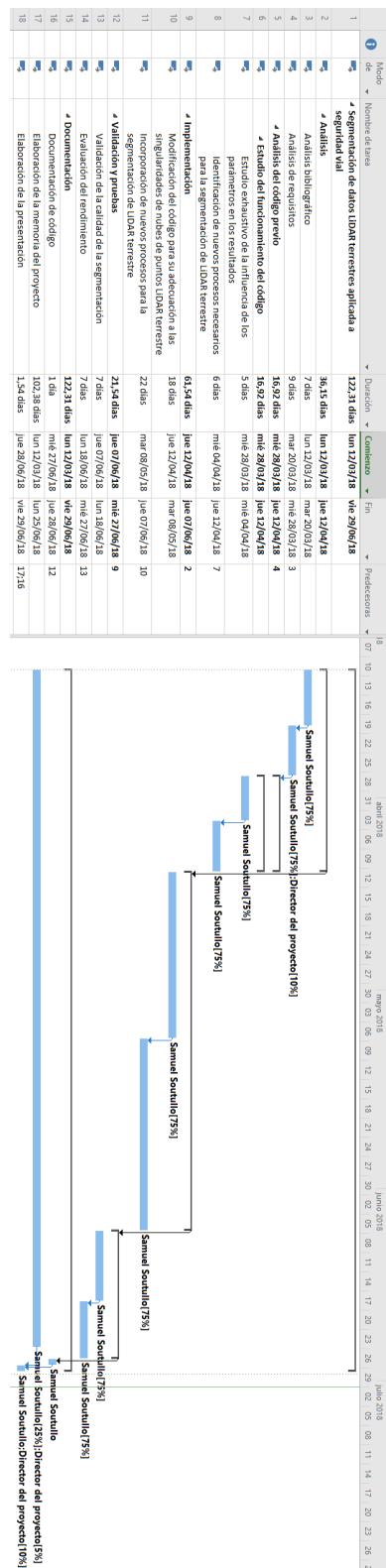


Figura 2.2: Diagrama de Gantt del proyecto.

cambios realizados durante el incremento correspondiente, y prepararse para el siguiente.

2.4. Estimación de costes y recursos

La estimación de costes y recursos se realizará en función de las tareas especificadas en la sección 2.2. Los costes serán presentados divididos en dos grupos: costes personales y costes materiales.

Los costes personales son aquellos en los que se incurría en caso de que los recursos humanos involucrados en el desarrollo del proyecto percibiesen un salario en función de las horas invertidas en la realización del mismo.

Los costes materiales, por otra parte son aquellos relacionados con la adquisición de *hardware* y/o *software*, el mantenimiento de los mismos, incluyendo costes indirectos como el asociado al consumo eléctrico, y, en definitiva, todo coste no asociado directamente a los recursos humanos.

2.4.1. Costes personales

Para estimar los costes asociados de forma directa a recursos humanos, se asumirá que el proyecto es desarrollado por dos individuos que desempeñan dos roles distintos: director de proyecto y becario de investigación.

- El **director de proyecto** supervisa y dirige el proyecto. Por otra parte, también puede realizar labores puntuales de asesoramiento al becario de investigación. En este caso se trata, por tanto, de los tutores del Trabajo de Fin de Grado.
- El **becario de investigación** es el encargado principal de desarrollar el proyecto. Para ello, deberá desempeñar, entre otras tareas: análisis, diseño y codificación del algoritmo y validación y pruebas del mismo, así como la elaboración de la documentación correspondiente. En este caso se trata, por tanto, del alumno que ha desarrollado el Trabajo de Fin de Grado.

El cuadro 2.1 recoge los costes de cada uno de los recursos humanos mencionados. Los costes por hora se especifican en bruto, por lo que incluyen todos los impuestos. Cabe destacar que han sido estimados conforme a la normativa de la Universidad [9]. El número de horas de trabajo desempeñadas por cada uno de los recursos se ha calculado en función de la planificación temporal recogida en el apartado 2.2.

Sumando cada uno de los costes totales recogidos en el cuadro 2.1 se obtiene que los costes personales totales ascienden a 8050 €. Como se ha mencionado anteriormente, esta estimación incluye impuestos.

Rol	Coste/Hora	Horas de trabajo	Coste total
Tutor del proyecto	25 €/hora	25 horas	625 €
Becario de investigación	18 €/hora	412,5 horas	7425 €

Cuadro 2.1: Costes de recursos humanos.

2.4.2. Costes materiales

En lo que respecta a la estimación de costes materiales se analizarán los costes asociados a productos físicos, los costes asociados a productos software, y otros costos materiales varios.

Costes asociados a productos físicos

No se ha necesitado adquirir ningún producto de manera expresa para el proyecto, sino que ya se contaba de antemano con todo el material físico necesario para el desarrollo del mismo. Sin embargo, aún así es necesario imputar el coste asociado al uso de dichos productos.

Para imputar el coste asociado al uso de los productos se debe introducir el concepto de vida útil de un producto. La vida útil de un producto consiste en la cantidad total de tiempo que este va a ser usado desde su adquisición hasta que se cese el uso del mismo. Se incluye no solo el tiempo dedicado a este proyecto, sino a cualquier otra tarea que dicho producto pueda desempeñar.

Conociendo el precio por el que se ha comprado el producto y su vida útil, resulta muy sencillo calcular el coste asociado al uso del mismo en el contexto de este proyecto. Simplemente hay que hallar que fracción de la vida útil del producto supone el tiempo de desarrollo de este proyecto y multiplicarlo por el precio de adquisición del mismo.

El cuadro 2.2 recoge los costes asociados a los productos físicos. Para cada producto se da la siguiente información:

- **P.V.P:** el precio por el que fue adquirido en el momento de su compra
- **Vida útil:** una estimación de la vida útil, concepto explicado anteriormente, en meses.
- **Coste/Mes:** el coste por mes del producto teniendo en cuenta su vida útil. Para calcularlo simplemente se divide el P.V.P entre los meses de vida útil.
- **Coste final:** el coste final del producto de cara al desarrollo del proyecto. Para calcularlo simplemente se multiplica el coste por mes, por el número de meses que dura el desarrollo del proyecto, en este caso 4 meses.

Por lo tanto, el total asociado a costes de productos físicos asciende a 67,32 €.

Producto	P.V.P	Vida útil	Coste/Mes	Coste final
Equipo portátil	1000 €	84 meses	11,90 €/mes	47,60 €
Monitores externos	350 €	120 meses	2,92 €/mes	11,68 €
Teclado externo	35 €	36 meses	0,97 €/mes	3,88 €
Ratón externo	50 €	48 meses	1,04 €/mes	4,16 €

Cuadro 2.2: Costes asociados a productos físicos.

Costes asociados a productos software

En este caso todo el *software* empleado se ha podido adquirir de manera gratuita, bien sea porque había sido desarrollado previamente por el grupo de investigación que dirige este Trabajo de Fin de Grado, porque se trata de *software* libre, o porque se ha podido adquirir de manera gratuita al ser estudiante universitario.

Otros costes materiales

En esta última sección se tratarán los gastos menores indirectos que no encajan en ninguna de las categorías hasta ahora tratadas. En particular, se considerarán costes asociados a desplazamientos y consumo eléctrico.

- Los **desplazamientos** están justificados por la necesidad de mantener reuniones presenciales de forma periódica con los tutores del proyecto. Si bien durante la mayor parte del tiempo de desarrollo del mismo, el alumno ha residido en Santiago de Compostela, durante las últimas semanas, por el contrario, lo ha hecho cerca de la ciudad de Vigo.

Para realizar dichos desplazamientos se usará el tren como medio de transporte. El coste de ida y vuelta a Santiago de Compostela asciende a 21,10 €. Teniendo en cuenta la realización de un total de 3 desplazamientos de este tipo, el coste total asociado a esta categoría asciende a 63,30 €.

- El **consumo eléctrico** radica del uso del equipo portátil y los monitores. Resulta sencillo calcular el coste asociado a este consumo eléctrico conociendo la potencia eléctrica del producto, facilitada por el fabricante, y el precio del kWh de la tarifa estándar de *Gas Natural Fenosa*, que es de 0,148051 €/kWh. [3]

El cuadro 2.3 recoge el coste asociado al consumo eléctrico de cada uno de los dispositivos empleados durante el desarrollo del proyecto.

Sumando los costes asociados a cada uno de los dispositivos, se obtiene un coste total de 13,81 €.

Producto	Potencia	Tiempo de uso	Energía	Coste
Equipo portátil	120 W	412,5 h	49,50 kWh	7,33 €
Monitor 1	65.7 W	412,5 h	27,10 kWh	4,01 €
Monitor 2	40.5 W	412,5 h	16,71 kWh	2,47 €

Cuadro 2.3: Consumo eléctrico de los dispositivos.

Por tanto, los costes asociados a desplazamientos y consumo eléctrico suman un total de 77,11 €.

Se desprecian otros costes indirectos como material fungible, limpieza, etc. Con todo estos costes indirectos suponen menos del 21 % del coste del proyecto, que es el valor usado por la USC en la mayoría de los proyectos de investigación.

2.4.3. Costes totales

Tras haber analizado los costes de diferentes categorías, solo falta proporcionar un valor final de costes totales asociados al proyecto. El cuadro 2.4 resume todos los costes analizados anteriormente.

Tipo de coste	Valor total
Costes personales	8050 €
Costes materiales	67,32 €
Productos físicos	0 €
Otros	77,11 €

Cuadro 2.4: Costes totales del proyecto.

Sumando todos los costes recogidos en el cuadro resumen 2.4, se obtiene que el total de costes asociados al proyecto es de 8194.43 €.

2.5. Gestión de riesgos

2.5.1. Definiciones

Análisis cualitativo de riesgos

La realización correcta de un análisis de riesgos depende, fundamentalmente, de que se identifiquen correctamente los distintos niveles de probabilidad e impacto de los riesgos específicos. El impacto representa las consecuencias de la ocurrencia del riesgo en términos de plazo, esfuerzo o coste. El cuadro 2.5 recoge su valoración en el contexto del proyecto:

Valoración del impacto	
Impacto	Repercusión en plazo, calidad, esfuerzo o coste
Muy bajo	<5 %
Bajo	Entre 5 % y 10 %
Moderado	Entre 10 % y 20 %
Alto	Entre 20 %,y 40 %
Muy alto	>40 %

Cuadro 2.5: Definición de impacto de los riesgos.

Valoración de la probabilidad	
Probabilidad	Ocurrencia del riesgo
Baja	<= 30 % (poco probable)
Moderada	Entre 30 % y 80 % (muy probable)
Alta	>= 80 % (casi segura)

Cuadro 2.6: Definición de probabilidad de los riesgos.

La probabilidad representa la expectativa de la ocurrencia real del riesgo. El cuadro 2.6 recoge su valoración en el contexto del proyecto:

Una matriz de probabilidad e impacto permite prever el nivel de exposición del proyecto a un riesgo concreto. Es necesario estudiar la exposición frente a un riesgo ya que es la forma más eficiente de priorizar riesgos. El cuadro 2.7 representa esta matriz de probabilidad e impacto y define, en función de los valores que pueden tomar estos, los diferentes niveles de exposición a los riesgos.

Nivel de exposición al riesgo					
Probabilidad / Impacto	Muy bajo	Bajo	Moderado	Alto	Muy alto
Baja	Muy baja	Baja	Baja	Moderada	Alta
Moderada	Muy baja	Baja	Moderada	Alta	Muy alta
Alta	Baja	Moderada	Alta	Alta	Muy alta

Cuadro 2.7: Definición de exposición a los riesgos.

Indicadores

Un aspecto muy importante en la gestión de riesgos es la identificación de los indicadores. Estos consisten en la descripción de situaciones que podrían indicar que el riesgo podría estar materializándose o a punto de materializarse. Especificar los indicadores de un riesgo permite tratarlos en el momento adecuado.

Estrategias para el tratamiento de riesgos

Las cuatro estrategias que se barajarán a la hora de abordar amenazas o riesgos que pueden tener impactos negativos son las siguientes:

- **Evitar:** se actúa para neutralizar la amenaza o para proteger al proyecto de su impacto, de forma que a efectos prácticos es imposible que se manifiesten las consecuencias negativas del riesgo durante el proyecto. Por lo general, implica cambiar el plan para la dirección del proyecto.
- **Transferir:** se delega el impacto de la amenaza en un tercero, junto con la responsabilidad de la respuesta. Casi siempre implica el pago de una prima de riesgo a la parte que consume dicho riesgo.
- **Mitigar:** se actúa para reducir la probabilidad de ocurrencia o el impacto del riesgo, reduciendo en todo caso el nivel de exposición. Se basa en la premisa de que adoptar acciones tempranas para reducir la exposición al riesgo a menudo es más eficaz que tratar de reparar el daño una vez éste ha ocurrido.
- **Aceptar:** se decide reconocer el riesgo y no tomar ninguna medida a menos que éste se materialice.

2.5.2. Identificación de riesgos

RSK-001	Las herramientas necesarias para el desarrollo del proyecto no funcionan como se esperaba
Descripción	Es posible que las herramientas tales como entornos de desarrollo integrados (<i>IDE</i>), visualizadores de nubes de puntos, etc., funcionen de manera errónea o distinta a la esperada. Ello daría lugar a la necesidad de invertir tiempo para solucionar estos problemas, lo que podría reducir el ritmo de avance del proyecto. Consecuentemente, se podrían producir incumplimientos en los plazos.
Probabilidad	Baja
Impacto	Alto
Exposición	Moderada
Indicadores	Aparición de constantes problemas en las herramientas que el alumno utiliza y sobre las que no tiene control a bajo nivel, que provoquen un retraso superior al 10 % con respecto a la planificación.
Acción	Mitigar
Tratamiento	Para reducir las consecuencias negativas derivadas de la materialización de este riesgo, se deben realizar pequeñas pruebas en una fase lo más temprana posible de todas las herramientas que deberán ser usadas a lo largo del desarrollo del proyecto. De esta forma, si se detectasen problemas con alguna de ellas, habría margen para poder solucionarlos mientras no sea todavía necesario usar dicha herramienta.

RSK-002	Los ciclos de revisión y/o decisión con los tutores del proyecto son más lentos de lo esperado
Descripción	Durante el desarrollo del proyecto, la comunicación con los tutores será constante, con el objetivo de obtener <i>feedback</i> de forma que el producto final cuente con la calidad esperada. Sin embargo, los ciclos de revisión y/o decisión con los tutores pueden ser más lentos de lo esperado por diversas razones, como la indisponibilidad de los propios tutores o del alumno para mantener una reunión en un momento dado. El avance en el desarrollo del proyecto dependerá de dichos ciclos de revisión y/o decisión. En consecuencia, se podría dar lugar a incumplimientos en los plazos.
Probabilidad	Moderada
Impacto	Alto
Exposición	Alta
Indicadores	Aparición de períodos de tiempo en los que el desarrollo del proyecto se ve interrumpido únicamente y exclusivamente por la imposibilidad de realizar una sesión de revisión y/o decisión con los tutores del proyecto, que provoquen un retraso superior al 10 % con respecto a la planificación.
Acción	Evitar
Tratamiento	Se puede evitar esta situación si desde el principio se establece una rutina de reuniones periódicas. Por ejemplo, a partir del momento en el que se empieza a trabajar en el proyecto mantener una reunión semanal, siempre el mismo día y a la misma hora. Se deberá asistir a estas reuniones incluso si el avance en el proyecto no es demasiado relevante, ya que de esta forma se refuerza el carácter rutinario de las mismas.

RSK-003	Un retraso en una tarea produce retrasos en cascada en las tareas dependientes
Descripción	Cuando una tarea se ubica en el camino crítico de la planificación, si la duración de dicha tarea aumenta, el tiempo de desarrollo del proyecto también aumentará. En consecuencia, retrasos en ciertas tareas podrían dar lugar a incumplimientos en los plazos.
Probabilidad	Moderada
Impacto	Moderado
Exposición	Moderada
Indicadores	Al inputar el retraso que se está produciendo en la tarea en el fichero de planificación, el software de planificación reporta un aumento en la duración del proyecto.
Acción	Mitigar
Tratamiento	En el momento en el que se detecta este problema se debe volver a encarar el desarrollo del proyecto para cumplir los planes previstos. Para ello se pueden o bien descartar tareas superfluas, o, si no las hubiere, reducir el alcance o la calidad del resultado asociados a algunas de ellas. De este modo, se amortiguan los retrasos surgidos.

RSK-004	Pérdida de información
Descripción	La pérdida de información puede llegar a suponer un problema catastrófico, puesto que toda aquella información que haya sido generada durante el desarrollo del proyecto, y que se pierda de manera irremediable, supone una vuelta atrás en la realización de tareas. Por tanto, es particularmente grave si se produce en un estado avanzado del proyecto.
Probabilidad	Baja
Impacto	Muy alto
Exposición	Alta
Indicadores	La información solo está guardada en una localización.
Acción	Evitar
Tratamiento	Se puede evitar este riesgo de manera sencilla respaldando toda la información asociada al proyecto en la nube. Sería particularmente interesante integrar esto con los procesos de gestión de la configuración, usando conocidos servicios como <i>GitHub</i> .

RSK-005	Falta de nubes de puntos de prueba
Descripción	La fase de pruebas es fundamental para asegurar la calidad del producto software desarrollado. En este proyecto en particular, dado que la funcionalidad del programa que se está desarrollando consiste en segmentar nubes de puntos, habrá que verificar que esto se realiza correctamente probando a realizar segmentaciones y validando las mismas. Para realizar una correcta validación del programa se necesitan, o bien muchas nubes de puntos, o bien tan sólo una o dos si estás cuentan con una cantidad elevada de estructuras a segmentar. De no contar con esto, podría llegar a suceder que el programa funcionase correctamente, por pura coincidencia, con cierto banco de pruebas, pero que al tratar de usarlo con otras nubes de puntos se produjesen errores notables en la segmentación.
Probabilidad	Moderada
Impacto	Moderado
Exposición	Moderada
Indicadores	Se observa que la cantidad de nubes de puntos ofrecidas al alumno por el grupo de investigación, disponibles para ser probadas, es inferior a tres y que además en dichas nubes de puntos solo existe una estructura relevante para ser segmentada.
Acción	Transferir
Tratamiento	Se delega en los tutores del proyecto, y el grupo de investigación al que pertenecen, la responsabilidad en este sentido. Siempre y cuando la segmentación se realice correctamente para los casos de prueba facilitados, el trabajo realizado por el alumno debe considerarse correcto, ya que éste no tuvo forma posible de detectar errores con otras nubes de puntos, al no tener acceso a las mismas.

RSK-006	Para ser realizadas, ciertas tareas necesitan una formación académica con la que no se cuenta
Descripción	Dado que el alumno no ha trabajado con nubes de puntos LiDAR en ningún momento durante la realización del grado, ni tan siquiera tampoco con algoritmos de segmentación de imágenes de cualquier tipo, es posible que carezca, en un primer momento, de la formación necesaria para acometer la realización de ciertas tareas de la manera más correcta. Esto podría derivar en el desarrollo de un producto de baja calidad, o un incumplimiento de los plazos al no poder trabajar a una velocidad apropiada.
Probabilidad	Alta
Impacto	Alto
Exposición	Alta
Indicadores	Se detecta que, en ciertos momentos, no es posible realizar ciertas tareas por el simple hecho de no contar con los conocimientos necesarios para ello. Un posible ejemplo en este sentido podría ser la imposibilidad de entender el código facilitado por los tutores del proyecto. Esto provoca un retraso superior al 10 % con respecto a la planificación.
Acción	Evitar
Tratamiento	Antes de comenzar a desarrollar el proyecto, resulta de vital importancia realizar una labor de estudio bibliográfico muy intensiva. Los tutores facilitarán siempre todos los recursos, ya sean artículos en Internet o publicaciones científicas, necesarios para entender el proyecto y todo lo que lo rodea.

RSK-007	El algoritmo del que se parte no tiene potencial para funcionar correctamente con la nueva casuística
Descripción	Se parte de un algoritmo para realizar segmentaciones sobre LiDAR aéreo que funciona correctamente en estos casos, pero no lo hace en el caso de nubes de puntos de LiDAR terrestre. Se asume que realizando pequeñas modificaciones sobre el mismo, o ajustando ciertos parámetros se podrá solucionar este problema. Sin embargo, al tratarse esto se trata más de una suposición que de un hecho constatado es posible que dicha asunción sea incorrecta. Existe la posibilidad, por tanto, de que el algoritmo nunca pueda funcionar correctamente bajo la nueva casuística.
Probabilidad	Baja
Impacto	Alto
Exposición	Moderada
Indicadores	Tras invertir cierta cantidad de tiempo modificando el código y variando parámetros no se obtienen resultados esperanzadores.
Acción	Mitigar
Tratamiento	Dado que el algoritmo de grafos desarrollado por el grupo de investigación se trata de un algoritmo de segmentación de propósito general, podría proponerse, como alternativa, un algoritmo nuevo diseñado específicamente para el dominio del proyecto. En otras palabras, si el algoritmo de partida no segmenta con la suficiente calidad las estructuras más interesantes de cara al proyecto (pasos de peatones, bordillos, aceras, etc.) se debería desarrollar un algoritmo diseñado específicamente para este propósito, aunque no funcione en otros escenarios.

Capítulo 3

Especificación de requisitos

En este capítulo se recoge la especificación del sistema en forma de requisitos. Para ello se fijaran los requisitos funcionales y no funcionales del sistema, los casos de uso, y los formatos de los ficheros que se manejarán.

3.1. Identificación de requisitos

En esta sección se identifican y especifican los requisitos funcionales y no funcionales del sistema.

En primer lugar, se tratarán los requisitos funcionales, que tendrán un código de identificación que seguirá el formato **RF-XXX**, donde **XXX** es un identificador numérico único de tres cifras. En segundo lugar se tratarán los requisitos no funcionales del sistema, que tendrán un código de identificación que seguirá el formato **RNF-XXX**.

Para cada requisito, además de su identificador y su título, se especificarán los siguientes campos:

- **Descripción:** una descripción del requisito lo suficientemente detallada como para que su interpretación sea única.
- **Importancia:** la importancia de implementar la funcionalidad asociada al requisito en el producto final. Puede tomar los valores:
 - **Esencial:** el producto no tiene sentido si no se implementa la funcionalidad asociada a este requisito.
 - **Deseable:** el producto es válido si no se implementa la funcionalidad asociada a este requisito. No obstante implementarla sería bastante útil y positivo para el producto.
 - **Estimulante:** la funcionalidad asociada a este requisito tiene poco interés. Si bien su implementación podría resultar positiva para el

producto, es recomendable realizarla solamente si implica la inversión de pocos recursos.

- **Criterio de validación:** especifica la forma de determinar, dado el producto final, que el requisito se ha implementado correctamente.

Los requisitos no funcionales contarán con el siguiente campo, a mayores de los anteriormente descritos:

- **Tipo:** el tipo de requisito no funcional que se está especificando. Son tipos comunes: rendimiento, usabilidad, mantenibilidad, seguridad, etc.

3.1.1. Requisitos funcionales

RF-001	Lectura y carga en memoria de nubes de puntos
Descripción	Dado un fichero de entrada, que representa una nube de puntos según el formato especificado en la sección 3.3.1, el programa deberá poder leerlo correctamente, cargando todos los puntos con sus características en memoria. Esto permitirá la posterior ejecución del algoritmo de segmentación.
Importancia	Esencial
Criterio de validación	El programa se ejecuta y comienza a leer el fichero que se le indique. Una vez termine de realizar dicha lectura, si se observa el mapa de memoria del programa será posible comprobar como se han cargado, sin errores ni modificaciones, los mismos puntos que los que contenía el fichero.

RF-002	Segmentación de carreteras
Descripción	Los puntos asociados a una zona de carretera se deberán separar de todos los demás. Se trata de un proceso única y exclusivamente de segmentación, no de clasificación. Por tanto, no será necesario determinar si un punto pertenece a una carretera o no, sino simplemente separarlo de aquellos que no pertenecen, y juntarlo con aquellos sí pertenecen.
Importancia	Deseable
Criterio de validación	Al visualizar la salida de la ejecución del algoritmo, se puede comprobar como todos los puntos asociados a una misma carretera se incluyen en un mismo grupo, el cual no contiene ningún punto que no pertenezca a dicha carretera.

RF-003	Segmentación de líneas de carretera
Descripción	Los puntos asociados a una línea de carretera (línea de paso de peatones, línea de separación de carriles, ...) se deberán separar de todos los demás. Se trata de un proceso única y exclusivamente de segmentación, no de clasificación. Por tanto, no será necesario determinar si un punto pertenece a una línea de carretera o no, sino simplemente separarlo de aquellos que no pertenecen, y juntarlo con aquellos sí pertenecen.
Importancia	Deseable
Criterio de validación	Al visualizar la salida de la ejecución del algoritmo, se puede comprobar como todos los puntos asociados a una misma línea de carretera se incluyen en un mismo grupo, el cual no contiene ningún punto que no pertenezca a dicha línea.

RF-004	Segmentación de bordillos
Descripción	Los puntos asociados a un bordillo de una acera se deberán separar de todos los demás. Se trata de un proceso única y exclusivamente de segmentación, no de clasificación. Por tanto, no será necesario determinar si un punto pertenece a un bordillo o no, sino simplemente separarlo de aquellos que no pertenecen, y juntarlo con aquellos sí pertenecen.
Importancia	Deseable
Criterio de validación	Al visualizar la salida de la ejecución del algoritmo, se puede comprobar como todos los puntos asociados a un mismo bordillo se incluyen en un mismo grupo, el cual no contiene ningún punto que no pertenezca a dicho bordillo.

RF-005	Segmentación de otros obstáculos
Descripción	Los puntos asociados a un obstáculo (farola, semáforo, señal de tráfico, viandante, ...) se deberán separar de todos los demás. Se trata de un proceso única y exclusivamente de segmentación, no de clasificación. Por tanto, no será necesario determinar si un punto pertenece a un obstáculo o no, sino simplemente separarlo de aquellos que no pertenecen, y juntarlo con aquellos sí pertenecen.
Importancia	Estimulante
Criterio de validación	Al visualizar la salida de la ejecución del algoritmo, se puede comprobar como todos los puntos asociados a un mismo obstáculo se incluyen en un mismo grupo, el cual no contiene ningún punto que no pertenezca a dicho obstáculo.

RF-006	Escritura en fichero de los resultados de la segmentación
Descripción	Una vez realizada la segmentación, se deberá escribir el resultado en un fichero según el formato especificado en la sección 3.3.2.
Importancia	Esencial
Criterio de validación	El programa termina de ejecutarse y como resultado se crea un nuevo fichero. Dicho fichero, deberá ser poder cargado correctamente por el visualizador de nubes de puntos desarrollado por el grupo de investigación. Al hacerlo, el visualizador mostrará el resultado esperado tras la ejecución del algoritmo de segmentación.

3.1.2. Requisitos no funcionales

RNF-001	Será posible segmentar nubes de tamaño considerable
Tipo	Rendimiento
Descripción	El programa debe poder cargar y segmentar nubes de tamaño considerable. En particular, deberá poder segmentar nubes de hasta 20 millones de puntos ejecutándose en un equipo de sobremesa que cuente con 16 GB de RAM.
Importancia	Deseable
Criterio de validación	Al ejecutar el algoritmo sobre una nube de 20 millones de puntos, la memoria el sistema no desborda. Además, la ejecución del algoritmo llega a finalizar correctamente en algún momento.

RNF-002	La segmentación se deberá realizar en tiempos inferiores a una hora para nubes de formadas por unas decenas de millones de puntos
Tipo	Rendimiento
Descripción	El programa debe poder ejecutarse de manera correcta y completa, es decir, realizar una segmentación adecuada, en un tiempo razonable teniendo en cuenta el tamaño de la nube que se pretende segmentar. En concreto, deberá segmentar una nube de 20 millones de puntos en menos de 15 minutos, al ser ejecutado en un equipo de gama media-alta o superior.
Importancia	Deseable
Criterio de validación	Al ejecutar el algoritmo sobre una nube de 20 millones de puntos en un equipo moderno, la ejecución del mismo termina de forma correcta en menos de 15 minutos.

RNF-003	El código desarrollado debe ser legible y mantenible
Tipo	Mantenibilidad
Descripción	Es muy probable que el código desarrollado deba ser modificado en un futuro por los miembros del grupo de investigación, para adaptarlo a nuevas situaciones (nubes de puntos de mayor tamaño, puntos con más características de las contempladas actualmente, ...). Por ello, el código desarrollado debe ser entendible por cualquier ser humano, y modificable de manera sencilla, en la medida de lo posible, para poder ser adaptado a este tipo de nuevas situaciones.
Importancia	Deseable
Criterio de validación	Los directores del proyecto pueden entender el código desarrollado y están de acuerdo en que sería fácilmente modificable para adaptarlo a futuras situaciones.

3.2. Casos de uso

Al tratarse este de un proyecto de investigación, el programa derivado del mismo tiene un propósito muy específico: segmentar nubes de puntos LiDAR terrestre. Si bien cuando se hablaba de requisitos (sección 3.1) se especificaban diversas particularidades que tendría que cumplir el producto final, lo cierto es que, en lo que respecta al uso, intervendrá un único actor, el usuario, usando siempre el programa para el mismo propósito, segmentar una nube de puntos LiDAR.

Como consecuencia de lo descrito en el párrafo anterior, tan sólo existe un caso de uso: segmentar nube de puntos. El actor también será único: el usuario.

3.2.1. Diagrama de casos de uso

La figura 3.1 representa la asociación entre el actor y el caso de uso asociados a este proyecto. Al tratarse de un único actor y un único caso de uso, como se puede comprobar, se trata de un diagrama simple.



Figura 3.1: Diagrama de casos de uso.

3.2.2. Definición de casos de uso

En esta sección se definirá el caso de uso anteriormente mencionado. Se especificará: actores implicados, precondiciones, postcondiciones, escenario principal y escenario alternativo.

CU-01	Segmentar nube de puntos
Actores	Usuario
Precondiciones	Se dispone de una nube de puntos LiDAR terrestre sin segmentar
Postcondiciones	Se genera la segmentación asociada a la nube de puntos LiDAR terrestre que se ha introducido
Escenario principal	<ol style="list-style-type: none"> 1. El usuario indica el fichero asociado a la nube de puntos a segmentar. 2. El usuario indica los diferentes valores de los parámetros que afectan la ejecución del algoritmo. 3. El programa procesa la nube de puntos y realiza la segmentación. 4. El programa escribe la nube de puntos segmentada en un fichero.
Escenario alternativo	<ol style="list-style-type: none"> 1. El usuario indica un fichero a segmentar que no existe. 2. El programa detecta este error y aborta su ejecución de manera controlada, informando al usuario del error.

3.3. Formato de ficheros

Como todo programa de computadora, el segmentador que se está desarrollando tiene unas entradas y unas salidas como resultado de su ejecución. En este caso, la entrada consiste en el fichero que contiene la nube de puntos a segmentar, y la salida es otro fichero que representa la misma nube de puntos, pero donde cada uno de ellos está asociado a un determinado grupo de segmentación.

En esta sección se recogen los formatos de estos dos ficheros, que el programa desarrollado deberá respetar.

3.3.1. Fichero de entrada

Tal y como se ha mencionado anteriormente, el fichero de entrada representa la nube de puntos a segmentar. Consiste, esencialmente, en una secuencia de puntos, cada uno de ellos con diferentes características.

Así pues, cada línea de este fichero representa un punto de la nube de puntos. Para cada punto, sus diferentes características estarán separadas por espacios en blanco o tabuladores. Todas ellas, tienen formato numérico. La figura 3.2 muestra las diez primeras líneas de un fichero de ejemplo, lo que ayudará a comprender este formato.

```
561456.666000 4814497.633000 31.928000 0.000000 -0.511000 -0.212000 0.833000
561456.668000 4814497.620000 31.955000 0.000000 -0.581000 -0.296000 0.758000
561456.675000 4814497.621000 31.980000 0.000000 -0.602000 -0.348000 0.719000
561456.856000 4814497.628000 32.225000 0.000000 0.853000 0.521000 0.012000
561456.884000 4814497.630000 32.421000 0.000000 -0.826000 -0.559000 0.076000
561456.820000 4814497.658000 30.462000 0.000000 -0.634000 -0.318000 0.705000
561456.842000 4814497.654000 32.076000 0.000000 -0.802000 -0.263000 0.537000
561456.866000 4814497.657000 32.240000 0.000000 0.864000 0.504000 0.005000
561456.875000 4814497.647000 32.292000 0.000000 0.850000 0.526000 0.036000
561456.883000 4814497.637000 32.354000 0.000000 0.846000 0.532000 0.022000
```

Figura 3.2: Ejemplo de fichero de entrada

Los tres primeros campos representan las coordenadas (x, y, z) del punto. El siguiente, representa la intensidad de rebote del punto. Ya se ha explicado el concepto de intensidad en la sección 1.2. Finalmente, los tres últimos campos representan un vector unitario, que es el vector normal del plano formado por el punto en particular y sus vecinos más cercanos.

De ser necesario para mejorar la segmentación, podrían añadirse más campos para cada punto. Sin embargo, en principio, esta es la especificación de fichero de la que se parte.

Finalmente, destacar que los puntos no tienen por qué encontrarse en ningún orden determinado, por lo que no puede realizarse ninguna asunción en este respecto.

3.3.2. Fichero de salida

Tal y como se ha mencionado anteriormente, el fichero de salida representa la nube de puntos segmentada. Consiste, esencialmente, en la misma secuencia de puntos que existía en el fichero de entrada, pero con un campo nuevo que permita discernir a que grupo de segmentación se ha asignado dicho punto.

La figura 3.3 muestra las diez primeras líneas de un fichero de salida de ejemplo, lo que ayudará a comprender este formato.

Los siete primeros campos coinciden con los del fichero de entrada, explicado en la sección 3.3.1, por lo que no se volverá incidir sobre ellos.

Los dos siguientes campos son irrelevantes para este algoritmo, pero es necesaria su presencia en el fichero de salida. El visualizador desarrollado por el grupo de investigación lee más características de las que se necesitan en el contexto de este proyecto. Se introducen estos campos de relleno sin significado para cumplir

```

561456.666000 4814497.633000 31.928000 0 -0.511000 -0.212000 0.833000 0 0 0 0 0 0
561456.668000 4814497.620000 31.955000 0 -0.581000 -0.296000 0.758000 0 0 0 0 0 0
561456.675000 4814497.621000 31.980000 0 -0.602000 -0.348000 0.719000 0 0 0 0 0 0
561456.856000 4814497.628000 32.225000 0 0.853000 0.521000 0.012000 0 0 1 0 0 0
561456.884000 4814497.630000 32.421000 0 -0.826000 -0.559000 0.076000 0 0 2 0 0 0
561456.820000 4814497.658000 30.462000 0 -0.634000 -0.318000 0.705000 0 0 3 0 0 0
561456.842000 4814497.654000 32.076000 0 -0.802000 -0.263000 0.537000 0 0 1 0 0 0
561456.866000 4814497.657000 32.240000 0 0.864000 0.504000 0.005000 0 0 1 0 0 0
561456.875000 4814497.647000 32.292000 0 0.850000 0.526000 0.036000 0 0 1 0 0 0
561456.883000 4814497.637000 32.354000 0 0.846000 0.532000 0.022000 0 0 1 0 0 0

```

Figura 3.3: Ejemplo de fichero de salida

con el formato de entrada que el visualizador espera, de forma que sea posible validar los resultados.

El siguiente campo es el identificativo del grupo de segmentación. Todos los puntos con el mismo identificador pertenecerán al mismo grupo, según el algoritmo. Por ejemplo, todos los puntos que pertenezcan a la misma carretera contarán con el mismo valor identificador. Se trata de un número entero que puede tomar cualquier valor positivo, incluyendo el cero.

Finalmente, los tres últimos campos son, nuevamente, campos de relleno para adaptarse al formato de entrada del visualizador, que no tienen relevancia en el contexto de este proyecto.

Capítulo 4

Diseño

En este capítulo se tratarán de explicar, desde una perspectiva de alto nivel, los algoritmo desarrollados para la realización del presente Trabajo de Fin de Grado. Al este tratarse de un proyecto puramente de investigación, en el que el flujo de desarrollo del mismo va variando en función de los resultados intermedios que se obtienen, resulta imposible realizar una fase de diseño previa a la implementación. Por lo tanto, los diagramas y demás figuras recogidas en esta sección han sido elaboradas después del desarrollo de los correspondientes algoritmos, y pretenden simplemente facilitar la comprensión del trabajo realizado.

Como se comentaba en la sección 1.3, en principio, se partía de un algoritmo desarrollado por el grupo de investigación, para la segmentación de datos LiDAR aéreo. En esta misma sección se indicaba que el algoritmo no solo daba lugar a malos resultados para datos LiDAR terrestre, sino que era mejorable en otros muchos aspectos. Este algoritmo, basado en grafos, ha podido ser mejorado en algunos de los aspectos propuestos por los directores del proyecto. Sin embargo, no se ha podido obtener una mejora significativa en los resultados de segmentación.

La posibilidad de que este algoritmo basado en grafos no se adaptase a la nueva casuística era un riesgo recogido en la sección 2.5.2. En particular, se trata del riesgo RSK-007. Como tratamiento de dicho riesgo, se recogía la posibilidad de desarrollar un algoritmo cuyo propósito específico sea la segmentación de las estructuras viales de interés para este proyecto, y así se ha hecho. Por tanto, en este capítulo se tratarán las cuestiones de diseño no solo relacionadas con el algoritmo de partida, basado en grafos, sino también con el nuevo algoritmo desarrollado, basado en crecimiento de regiones [8].

4.1. Algoritmo basado en grafos

4.1.1. Algoritmo desarrollado por el grupo de investigación

Para comprender el funcionamiento del algoritmo de partida se ha elaborado un diagrama de flujo, recogido en la figura 4.1. Este algoritmo ha sido desarrollado específicamente para datos de LiDAR aéreo por el grupo de investigación [6]. A continuación, se procederá a realizar una explicación de dicho algoritmo atendiendo a las componentes del citado diagrama de flujo.

- **Leer y cargar puntos:** se procede a leer el archivo que representa la nube de puntos a segmentar. Los puntos se cargan en memoria para poder ser tratados por el algoritmo.
- **Calcular vecinos:** se realiza un proceso de obtención de vecinos en función de las coordenadas u otras características. Para cada punto se obtiene un conjunto de puntos vecinos, que se encuentran cercanos al primero en función de dichas características. Por ejemplo, en caso de usar las coordenadas para el cálculo de vecinos, para cada punto se obtendría un conjunto de puntos próximos a él en función de su posición.
- **Crear grafo:** se crea un grafo donde cada uno de los puntos leídos y cargados anteriormente pasa a ser un nodo del mismo. Cada nodo se conecta mediante aristas con todos sus vecinos, los cuales fueron determinados en el paso anterior. A cada arista se le da un peso, en función de las características de los dos puntos asociados a los dos nodos conectados a dicha arista.
- **Crear componentes:** las componentes son, esencialmente, conjuntos de nodos del grafo que se encuentran en el mismo grupo. En este punto se crea una lista de componentes con tantas componentes como nodos tiene el grafo. Cada componente contendrá, por el momento, un solo nodo del grafo. Por tanto, en este instante, existe una correspondencia uno a uno entre componentes y nodos del grafo.
- **Ordenar aristas del grafo:** las aristas del grafo anteriormente creado son ordenadas en función del peso que les fue asignado. En particular, serán ordenadas de menor a mayor peso. Se realiza esta operación para, posteriormente, procesar las aristas en el orden adecuado y de manera eficiente.
- **¿Quedan aristas por procesar?:** las aristas se irán procesando una a una según el orden establecido anteriormente. Esta decisión indica si ya se han procesado todas las aristas o no.

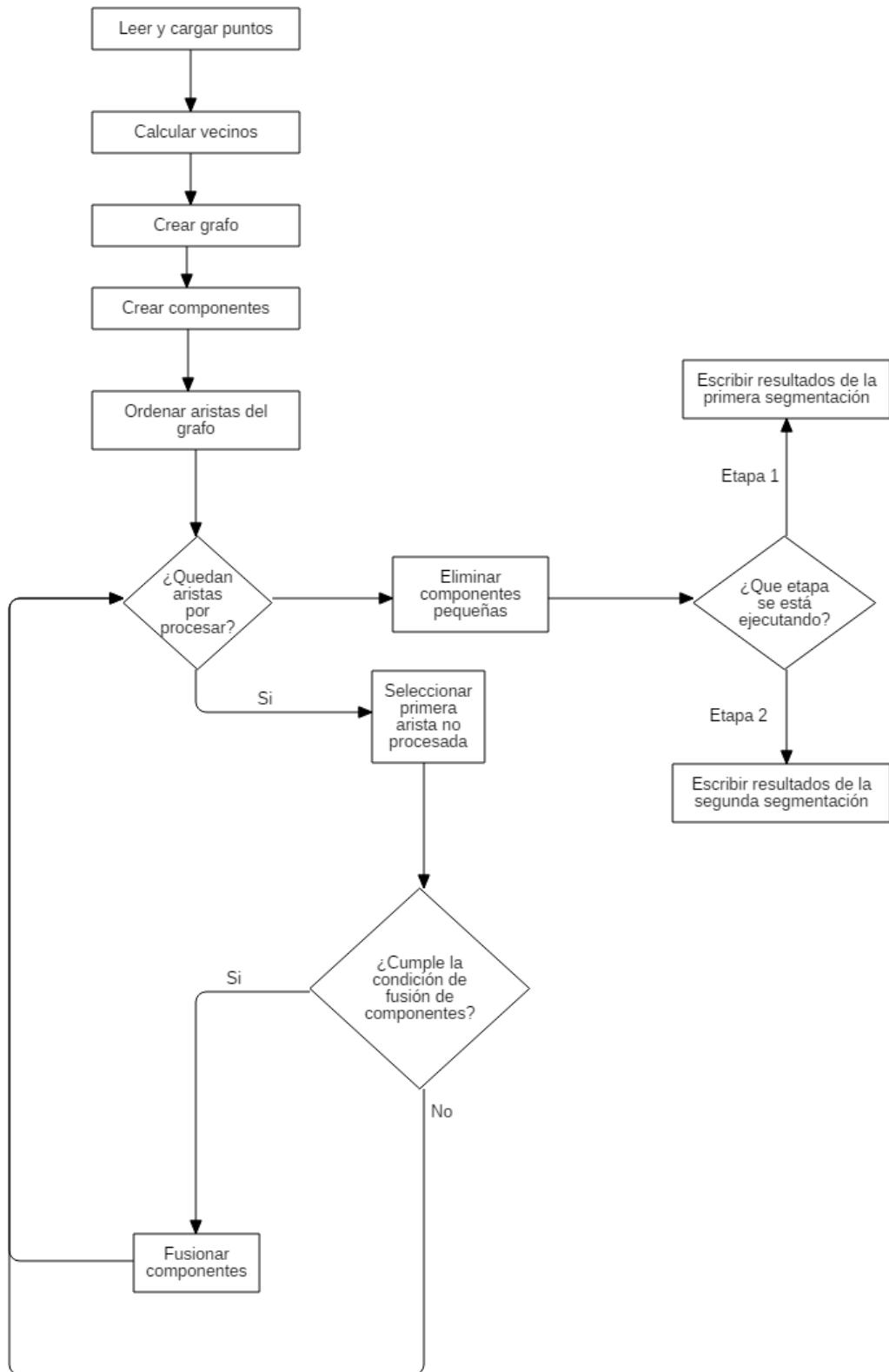


Figura 4.1: Diagrama de flujo del algoritmo de partida.

- **Seleccionar la primera arista no procesada:** localizar la arista de menor peso que no haya sido procesada en una iteración anterior.
- **¿Cumple la condición de fusión de componentes?:** se determina si las componentes asociadas a los nodos conectados por la arista actual deben ser fusionadas o no. La condición de fusión de componentes se cumple si el peso de la arista es menor que el menor de los pesos internos de las componentes que se pretenden fusionar más un *offset*. El peso interno de una componente es el peso de la mayor arista que une dos nodos que se encuentren dentro de la componente.
- **Fusionar componentes:** las dos componentes se fusionan. Los nodos que contenían ambas se mueven a una sola componente.
- **Eliminar componentes pequeñas:** para reducir las probabilidades de segmentaciones erróneas, se eliminan las componentes que contengan una cantidad de nodos (puntos) muy pequeña. Esto se basa en la premisa de que no deberían existir áreas segmentadas tan pequeñas en ninguna nube de puntos que cumpla unas condiciones normales.
- **¿Que etapa se está ejecutando?:** determina si se está ejecutando la etapa 1 o 2 del algoritmo. Se darán más detalles en este sentido en la sección *procedimiento de ejecución del algoritmo*. Para entender el funcionamiento general del algoritmo basta con saber que se ejecuta en dos etapas distintas y que esta decisión determina cual es la que se está ejecutando en un momento dado.
- **Escribir resultados de la primera segmentación:** se escribe la salida de la primera etapa, que está formada por diversos archivos temporales necesarios para la segunda etapa del algoritmo. Entre estos archivos se encuentra una nube de puntos temporal, donde cada punto representa cada una de las componentes resultantes de la ejecución del algoritmo. Se determinan las características de estos puntos mediante una media aritmética de todos los puntos originales asociados a la componente. También se escribe la nube de puntos original, pero añadiéndole un campo que representa el grupo al que pertenece, o lo que es lo mismo, el índice de la componente en la que se encuentra.
- **Escribir los resultados de la segunda segmentación:** se escribe la salida de la segunda etapa, que está formada por un archivo temporal. Se trata de un archivo de referencia al ID de los nuevos grupos creados en la segunda etapa de la segmentación. La utilidad de este fichero será explicada la sección *procedimiento de ejecución del algoritmo*.

Procedimiento de ejecución del algoritmo

Como se había comentado en la sección 1.3, el código de partida tenía un problema en lo que se refería a su flexibilidad y su procedimiento de ejecución. Ahora que ya se conoce el funcionamiento general del mismo, es posible comentar con mayor detalle los pasos necesarios para, partiendo de una nube sin segmentar, obtener dicha nube segmentada:

1. **Ejecutar el algoritmo en modo etapa 1:** se ejecuta el programa, indicando mediante un parámetro de entrada que se debe ejecutar la primera etapa. El algoritmo lee la nube de puntos original y se ejecuta de acuerdo a lo especificado en la figura 4.1 para la etapa 1.
2. **Ejecutar el algoritmo en modo etapa 2:** se ejecuta el programa, indicando nuevamente mediante un parámetro de entrada que se debe ejecutar la segunda etapa. El algoritmo lee la nube de puntos escrita tras la ejecución de la primera etapa, que está formada por tantos puntos como componentes existían al finalizar dicha etapa. El algoritmo se ejecuta conforme a lo especificado en la figura 4.1 para la etapa 2. Se escribe el archivo temporal de referencia de ID de nuevos grupos mencionado anteriormente.
3. **Ejecutar el programa *idtracker*:** antes de obtener la nube de puntos final segmentada, se debe ejecutar otro pequeño programa desarrollado por el grupo de investigación. Se trata de un programa que, en base a los archivos temporales creados tras la ejecución de la primera y segunda etapas del algoritmo, escribe un fichero que representa la nube de puntos original segmentada en función de los resultados de ambas etapas de segmentación. Este fichero está listo para ser visualizado.

4.1.2. Algoritmo modificado

Como se ha explicado, la flexibilidad y funcionamiento del algoritmo original no eran los óptimos, en particular en lo que se refiere a los pasos a realizar para obtener la segmentación final. Se han realizado modificaciones a este algoritmo, y se han podido suplir estas carencias.

La figura 4.2 recoge el diagrama de flujo del algoritmo de grafos modificado, y ayudará a comprender las modificaciones realizadas sobre el mismo desde una perspectiva de alto nivel. Nuevamente, se procederá a explicar este algoritmo modificando atendiendo a las componentes del citado diagrama de flujo.

- **Leer y cargar puntos:** se leen y se cargan los puntos, de igual manera que en el algoritmo original.
- **Crear grafo en función de los puntos actuales:** se crea un grafo en función de los puntos con los que se está trabajando. Este grafo tendrá tantos nodos como puntos. Por el momento, no tendrá ninguna arista.

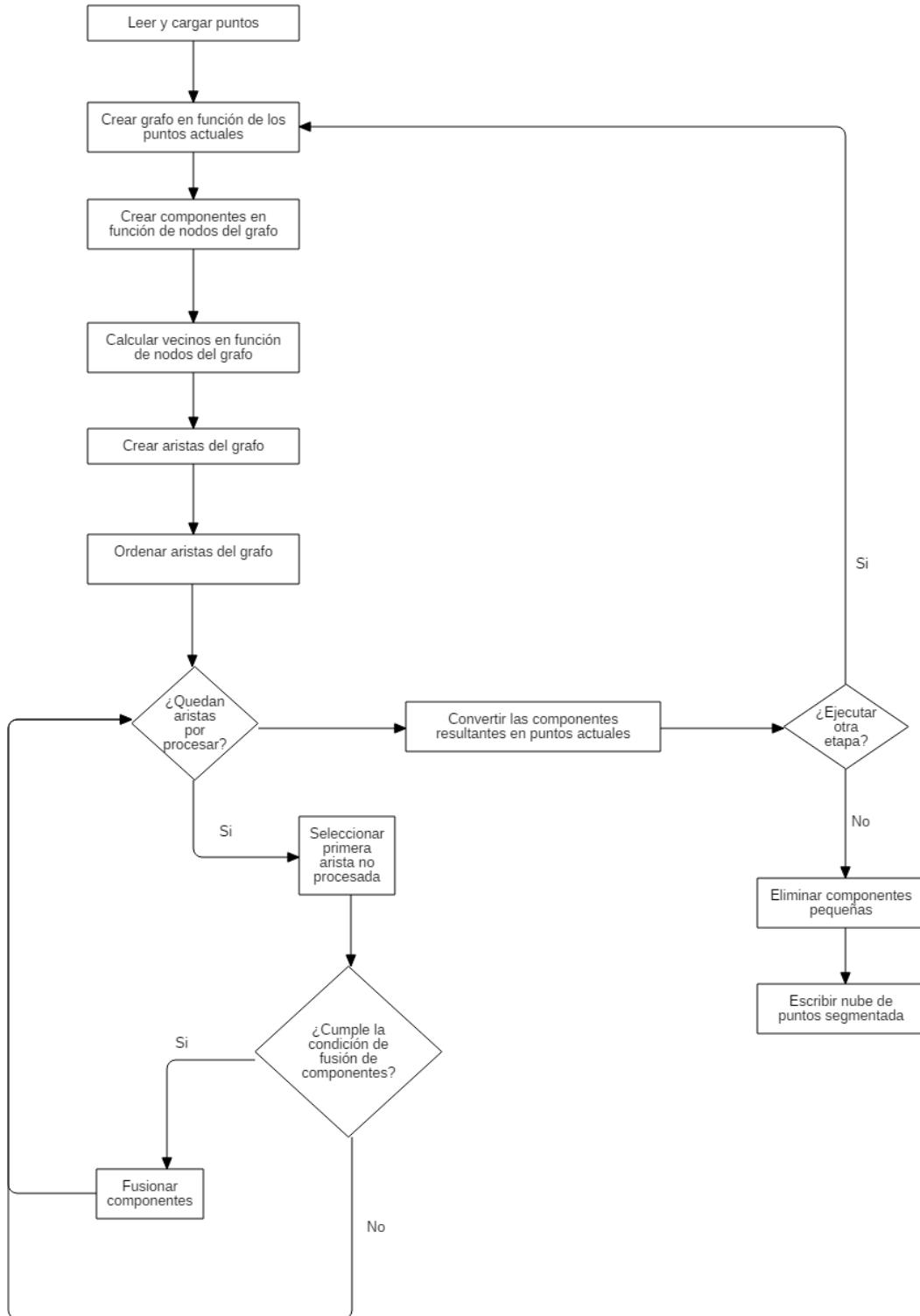


Figura 4.2: Diagrama de flujo del algoritmo modificado.

- **Crear componentes en función de nodos del grafo:** se crean las componentes de igual forma que en el algoritmo original, es decir, se crea una componente por cada nodo del grafo.
- **Calcular vecinos en función de nodos del grafo:** se realiza un cálculo de vecinos de manera similar a como se hacía en el algoritmo original. La diferencia en este caso es que se toman como puntos los nodos del grafo, mientras que en el algoritmo original este paso se realizaba en función de los puntos del fichero de entrada. Este pequeño cambio permitirá ejecutar tantas etapas como se desee sin tener que ejecutar el programa varias veces.
- **Crear aristas del grafo:** se conectan los nodos vecinos mediante aristas de manera similar a como se hacía en el algoritmo original, atendiendo a los resultados de cálculo de vecinos. Se le da un peso a cada arista de manera idéntica a como se hacía en el algoritmo original.
- **El flujo de ejecución de los siguientes bloques es idéntico al del algoritmo general, por lo que no serán descritos nuevamente:**
 - Ordenar aristas del grafo
 - Seleccionar primera arista no procesada
 - ¿Cumple la condición de fusión de componentes?
 - Fusionar componentes
- **Convertir las componentes resultantes en puntos actuales:** las componentes resultantes de la etapa actual de segmentación se convierten a puntos en memoria. Para ello, se hace la media aritmética de todas las características de los puntos de cada componente. Estos puntos se marcan como puntos actuales, que serán empleados en la siguiente iteración.
- **¿Ejecutar otra etapa?:** se decide si ejecutar otra etapa o no, con total flexibilidad en función de las preferencias del usuario.
- **Eliminar componentes pequeñas:** se eliminan las componentes pequeñas de manera similar a como se hacía en algoritmo original. La diferencia en este caso radica en que este paso se realiza tras ejecutar todas las etapas de segmentación. Esta decisión se basa en la premisa de que grupos que en principio son pequeños, pueden llegar a crecer fusionándose con otros tras la ejecución de varias etapas.
- **Escribir la nube de puntos segmentada:** tras la ejecución de todas las etapas de segmentación, se escribe directamente la nube de puntos segmentada lista para ser visualizada. No se usan, por tanto, archivos temporales intermedios.

Por tanto, este algoritmo soluciona parte de los problemas del algoritmo original. El número de etapas que ejecuta, así como las características que se tienen en cuenta en cada una de ellas, tanto para el cálculo de vecinos como para el cálculo de peso de las aristas, se decide de manera totalmente libre mediante argumentos de entrada. Se profundizará sobre estos argumentos de entrada en el apéndice B.

Sin embargo, a pesar de estos cambios no ha sido posible mejorar como escala el uso de memoria en función del tamaño de la nube de puntos. Por tanto, no es posible segmentar nubes de puntos más grandes con este algoritmo que con el algoritmo original. Tampoco se han podido obtener mejoras relevantes en lo que se refiere a la calidad de la segmentación final. Es por ello que se ha decidido desarrollar un algoritmo completamente nuevo, cuyos aspectos de diseño serán explicados en la sección 4.2.

4.2. Algoritmo basado crecimiento de regiones

Como se ha mencionado anteriormente, para tratar de obtener segmentaciones de mayor calidad en el contexto de este proyecto y mejorar la eficiencia en el uso de memoria del algoritmo de grafos, se ha procedido a desarrollar un algoritmo nuevo. En esta sección, mediante diversos diagramas, se tratará de explicar la estructura del programa desarrollado y el funcionamiento del algoritmo, desde una perspectiva de alto nivel.

4.2.1. Modificaciones al fichero de entrada

Antes de comentar el funcionamiento del algoritmo, es importante comentar que, en su versión final, trabaja con ficheros de entrada con una especificación distinta a la de partida. Cuando se describía dicha especificación, en la sección 3.3.1, se comentaba, precisamente, que podrían llegar a añadirse más campos si esto servía para mejorar la calidad de la segmentación, y así ha resultado ser finalmente.

Para entender la necesidad y naturaleza de estos campos es importante entender como se calculan el vector normal de cada punto. El grupo de investigación cuenta con un programa, que partiendo de un fichero que representa una nube de puntos sin normales, genera un fichero que representa la misma nube de puntos pero con los vectores normales ya calculados. Dicho programa, basa su cálculo de normales en la búsqueda del mejor plano de un punto y sus vecinos. En otras palabras, para calcular la normal de un punto, considera puntos vecinos que se encuentren en un radio determinado, calcula el mejor plano para dichos puntos, y entonces asume que la normal del punto será igual a la normal de este plano.

El problema de este método es que no es del todo preciso. En particular, durante el desarrollo de este proyecto, se detectó un problema con las normales de los bordillos de las aceras. Dado que los bordillos son estructuras relativamente

pequeñas, el método de calcular el mejor plano en función de los puntos vecinos no ofrece resultados precisos. Mientras que se esperaban dos grupos de puntos con normales perpendiculares (los correspondientes a las partes superior y frontal del bordillo, respectivamente), se obtenía que todos los puntos del bordillo tenían una normal prácticamente homogénea.

Se decidió que la solución más adecuada para este problema no era reducir el radio de cálculo de normales, sino incluir en el fichero de la nube de puntos dos parámetros que indicasen la calidad del plano generado para el cálculo de la misma. En zonas donde el cálculo de las normales se realizase sin problemas de ningún tipo, como zonas planas de carreteras, estos parámetros tendrían valores muy bajos. En zonas donde el cálculo de las normales fuese conflictivo, y el plano calculado tuviese un margen de error amplio, estos parámetros tendrían valores muy altos.

En consecuencia, estos dos parámetros podrían usarse para la segmentación de bordillos de aceras, ya que serían de las pocas estructuras de interés para el proyecto que tendrían un valor alto para estos parámetros.

Por tanto, con la inclusión de estos dos nuevos parámetros el formato del fichero de entrada para este algoritmo pasó a ser el representado en la figura 4.3.

```
561456.666000 4814497.633000 31.928000 0.000000 -0.511000 -0.212000 0.833000 0.054 0.073
561456.668000 4814497.620000 31.955000 0.000000 -0.581000 -0.296000 0.758000 0.050 0.118
561456.675000 4814497.621000 31.980000 0.000000 -0.602000 -0.348000 0.719000 0.055 0.142
561456.856000 4814497.628000 32.225000 0.000000 0.853000 0.521000 0.012000 0.046 0.509
561456.884000 4814497.630000 32.421000 0.000000 -0.826000 -0.559000 0.076000 0.027 0.521
561456.820000 4814497.658000 30.462000 0.000000 -0.634000 -0.318000 0.705000 0.100 0.661
561456.842000 4814497.654000 32.076000 0.000000 -0.802000 -0.263000 0.537000 0.080 0.343
561456.866000 4814497.657000 32.240000 0.000000 0.864000 0.504000 0.005000 0.037 0.526
561456.875000 4814497.647000 32.292000 0.000000 0.850000 0.526000 0.036000 0.018 0.556
561456.883000 4814497.637000 32.354000 0.000000 0.846000 0.532000 0.022000 0.017 0.551
```

Figura 4.3: Ejemplo de fichero de entrada con los dos nuevos parámetros.

Los siete primeros campos coinciden con los de la especificación inicial del fichero de entrada, explicada en la sección 3.3.1. Los campos octavo y noveno representan los dos nuevos parámetros que indican la planicidad de una zona. Tal y como se explico antes, valores bajos (típicos en carreteras) indican alta planicidad, mientras que valores bajos (típicos en bordillos de aceras) indican baja planicidad. El campo octavo se conoce como *flatness*, mientras que el noveno como *roughness*.

- ***Flatness***: representa la planicidad de la vecindad, un valor 0 representa planicidad máxima.
- ***Roughness***: es la distancia en metros del punto al plano que mejor se ajusta su vecindad.

4.2.2. Funcionamiento del algoritmo

Para comprender el funcionamiento del nuevo algoritmo desarrollado, basado en crecimiento de regiones, se ha elaborado un diagrama de flujo, recogido en la figura 4.4. A continuación, se procederá a realizar una explicación de dicho algoritmo atendiendo a las componentes del citado diagrama de flujo.

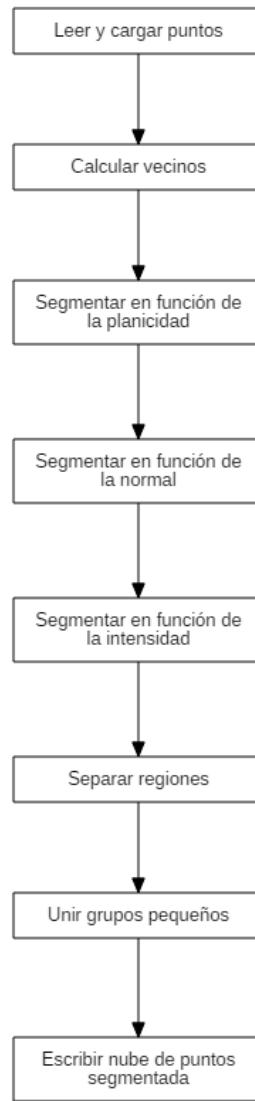


Figura 4.4: Diagrama de flujo general del nuevo algoritmo.

- **Leer y cargar puntos:** se procede a leer el archivo que representa la nube de puntos a segmentar. Los puntos se cargan en memoria para poder ser tratados por el algoritmo.

- **Calcular vecinos:** se realiza un proceso de obtención de vecinos en función de las coordenadas. Para cada punto se obtiene un conjunto de puntos vecinos, que se encuentran cercanos al primero en función de dichas características. Es importante incidir en el hecho de que en este algoritmo los vecinos se calculan únicamente en función de su posición, nunca atendiendo a ninguna otra característica.
- **Segmentar en función de la planicidad:** se separan los puntos en función de su valor de planicidad, teniendo en cuenta los dos nuevos parámetros comentados anteriormente. Esto permite separar puntos de bordillos de aquellos que no lo son.
- **Segmentar en función de la normal:** partiendo de los grupos del paso anterior, se separan los puntos en función de su normal. De esta forma, cada uno de los grupos se vuelve a segmentar, separando estructuras tales como fachadas de edificios de carreteras, al tener estos normales perpendiculares.
- **Segmentar en función de la intensidad:** partiendo de los grupos del paso anterior, se separan los puntos en función de su valor de intensidad. De esta forma, cada uno de los grupos se vuelve a segmentar, separando, principalmente, puntos correspondientes a líneas de carretera de la carretera propiamente dicha, ya que estas líneas al estar pintadas de color blanco cuentan con una intensidad considerablemente mayor.
- **Separar regiones:** para posteriores etapas de clasificación es imprescindible que dos estructuras distintas, aunque sean del mismo tipo, estén en grupos de segmentación distintos. Por ejemplo, no es permisible que dos pasos de peatones distintos se encuentren dentro del mismo grupo. Para realizar esta división se realiza una separación de regiones en función de los vecinos anteriormente calculados. De esta forma, cada uno de los grupos segmentados hasta este momento, se segmenta en grupos más pequeños. Si un punto es vecino de otro, y este a su vez es vecino de un tercero (y así sucesivamente), se considera que todos son vecinos entre sí, y permanecen en el mismo grupo. En el momento en el que se rompe esta cadena de vecindad se crea otro grupo que estará formado por otros puntos que sean vecinos entre sí.
- **Unir grupos pequeños:** los grupos pequeños resultantes no se eliminan sin más, como se hacía en el algoritmo de grafos. En su lugar, se incluyen en los grupos más próximos. Se hace de esta forma basándose en la premisa de que los grupos pequeños se forman por errores de segmentación, siendo lo más probable que los puntos que se encuentran dentro de éstos pertenezcan a un grupo de mayor tamaño que se encuentra en una posición muy cercana.

Se trata por tanto de un algoritmo de segmentación específico u orientado a dominio. Esto es, el algoritmo ha sido diseñado para segmentar específicamente estructuras viales en entornos urbanos sobre nubes de puntos LiDAR terrestre, y no cualquier tipo de estructura en cualquier tipo de nubes de puntos LiDAR. Es por ello que el algoritmo está dividido en tres etapas, cada una de las cuales tiene como objetivo segmentar ciertos tipos de estructuras en particular.

Funcionamiento de una etapa de segmentación del algoritmo

La segmentación en función de planicidad, en función de la normal y en función de la intensidad son tres etapas de segmentación del algoritmo que funcionan de manera muy similar. Resulta interesante, para comprender como funciona el algoritmo, comprender como funcionan estas etapas. El diagrama de flujo recogido en la figura 4.5 representa este funcionamiento.

- **¿Quedan puntos por procesar?**: los puntos se irán procesando en el mismo orden que en el fichero de entrada. Esta decisión determina si en la etapa de segmentación actual ya se han procesado todos los puntos o no.
- **Seleccionar el primer punto no procesado**: se selecciona el primer punto no procesado y se prepara para evaluarlo.
- **¿Es similar a algún centroide?**: existe un centroide, por cada uno de los grupos de segmentación que se van creando a medida que se procesan los puntos. El centroide de un grupo es un punto que representa a dicho grupo. Para que un punto cualquiera pueda ser añadido a un determinado grupo de segmentación, debe ser lo suficientemente similar al centroide. Esta decisión determina si el punto actual es suficientemente similar a algún centroide de los grupos de segmentación creados hasta el momento.
- **Incluir el punto actual en el grupo correspondiente al centroide**: dado que el punto actual es suficientemente similar a algún centroide de los grupos creados hasta el momento, se entiende que este punto debe pertenecer al mismo grupo que dicho centroide. Por tanto, se añade a este grupo.
- **Crear nuevo grupo que tenga como centroide al actual**: dado que el punto actual no es lo suficientemente similar a ningún centroide, se entiende que no debe incluirse en ninguno de los grupos creados hasta el momento. Por tanto, se crea un nuevo grupo de puntos, y el punto actual pasa a ser el centroide de este nuevo grupo.

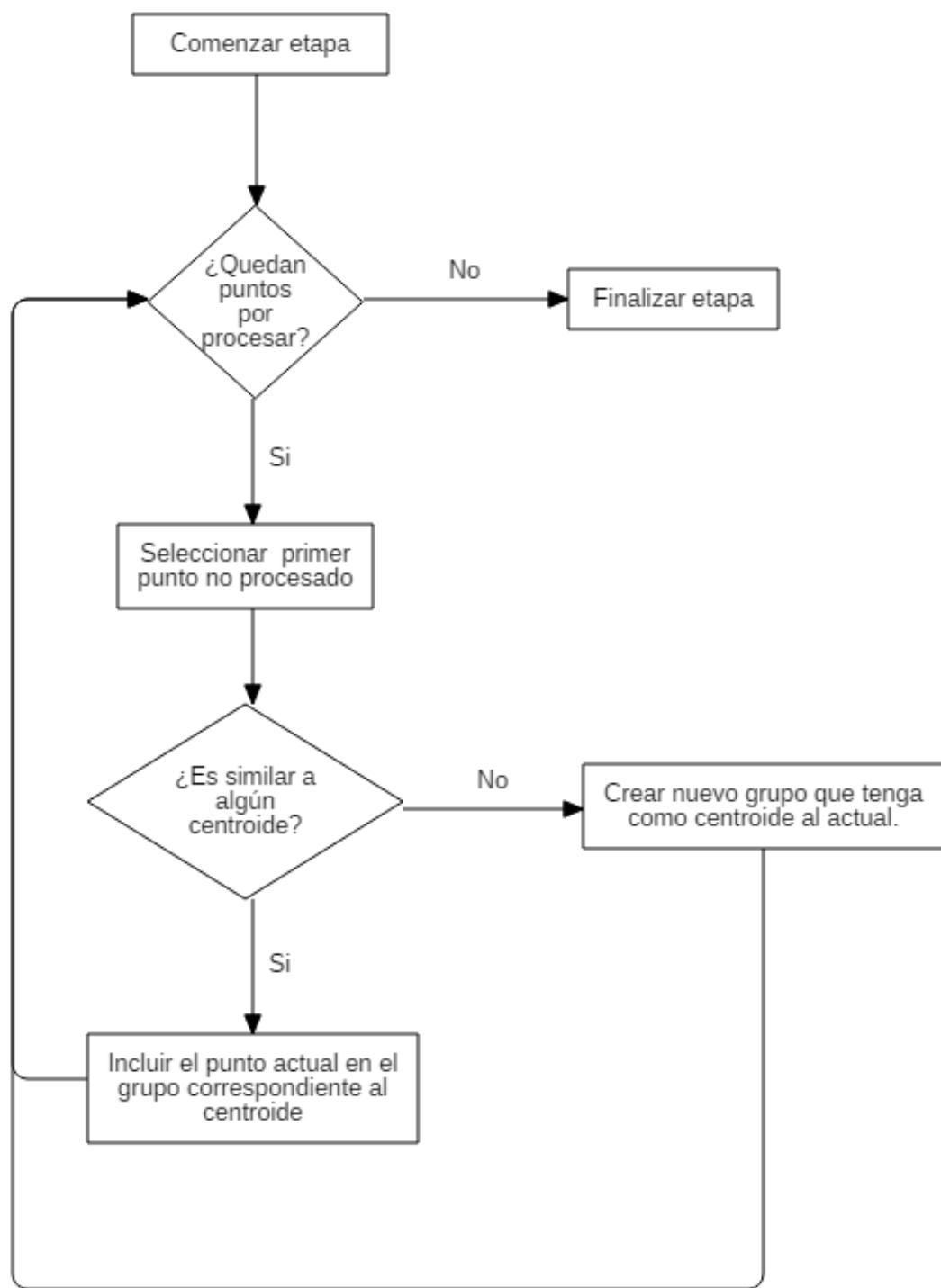


Figura 4.5: Diagrama de flujo de una etapa del nuevo algoritmo.

4.2.3. Estructura del programa

Una vez comentado el funcionamiento del algoritmo, es interesante también tratar la estructura del programa desarrollado. No se ha hecho esta consideración para el caso del programa del algoritmo de grafos, puesto que esta ya había sido definida por el grupo de investigación en el momento de su desarrollo inicial. Sin embargo, en este caso el programa ha sido creado de cero, por lo que se ha considerado que sí merece la pena comentar este aspecto.

La figura 4.6 representa, mediante lenguaje UML, la estructura del programa desarrollado. Como se ha desarrollado un programa en C, y UML se trata de un lenguaje de modelado orientado objetos, este diagrama no debe ser interpretado de forma literal, sino como una forma de comprender que archivos de código conforman el programa y las funciones que se pueden encontrar dentro de cada uno.

Antes de comentarlo detalladamente, es importante realizar un par de aclaraciones, ya que no está usando UML de manera estándar, debido a la situación comentada en el párrafo anterior.

En primer lugar, cada caja representa una dupla de archivos de código, un archivo .c y su correspondiente cabecera en forma de archivo .h. Esta regla cuenta con la excepción de la caja llamada *main*, que no cuenta con un archivo de cabecera y por tanto solo representaría el fichero *main.c*.

En segundo lugar, realizar una pequeña puntualización sobre los dos distintos estereotipos que se pueden encontrar en cada una de las cajas: «Type» y «File». Cuando una caja cuenta con el estereotipo «Type» es porque la correspondiente dupla de archivos .c y .h representan una estructura de datos con sus diferentes funciones de creación, modificación, y destrucción. Por el contrario, el estereotipo «File» representa simplemente una colección de funciones codificadas en un fichero .c y declaradas en un fichero .h.

Finalmente, se puede observar en el modelo que ciertas funciones cuentan con acceso público y otras con acceso privado. Esto representa la accesibilidad desde fuera del fichero .c asociado a la caja correspondiente, que depende únicamente de que se encuentre declarada en el correspondiente .h. En otras palabras, las funciones declaradas en el archivo .h correspondiente tienen acceso público, y las que solo se encuentran codificadas en el fichero .c tienen acceso privado.

A continuación, se procederá a comentar la estructura del programa en función de las diferentes componentes del diagrama recogido en la figura 4.6.

- **Arguments:** se trata de una estructura de datos encargada de procesar los argumentos de entrada que recibe el programa, que permiten adaptar el algoritmo a diferentes nubes de puntos. Se comentarán detalladamente los argumentos de entrada disponibles para el usuario en el apéndice B.
- **ArrayList:** se trata de una estructura de datos que permite la creación y manejo de forma sencilla de *arrays* dinámicos, muy útiles durante la ejecu-

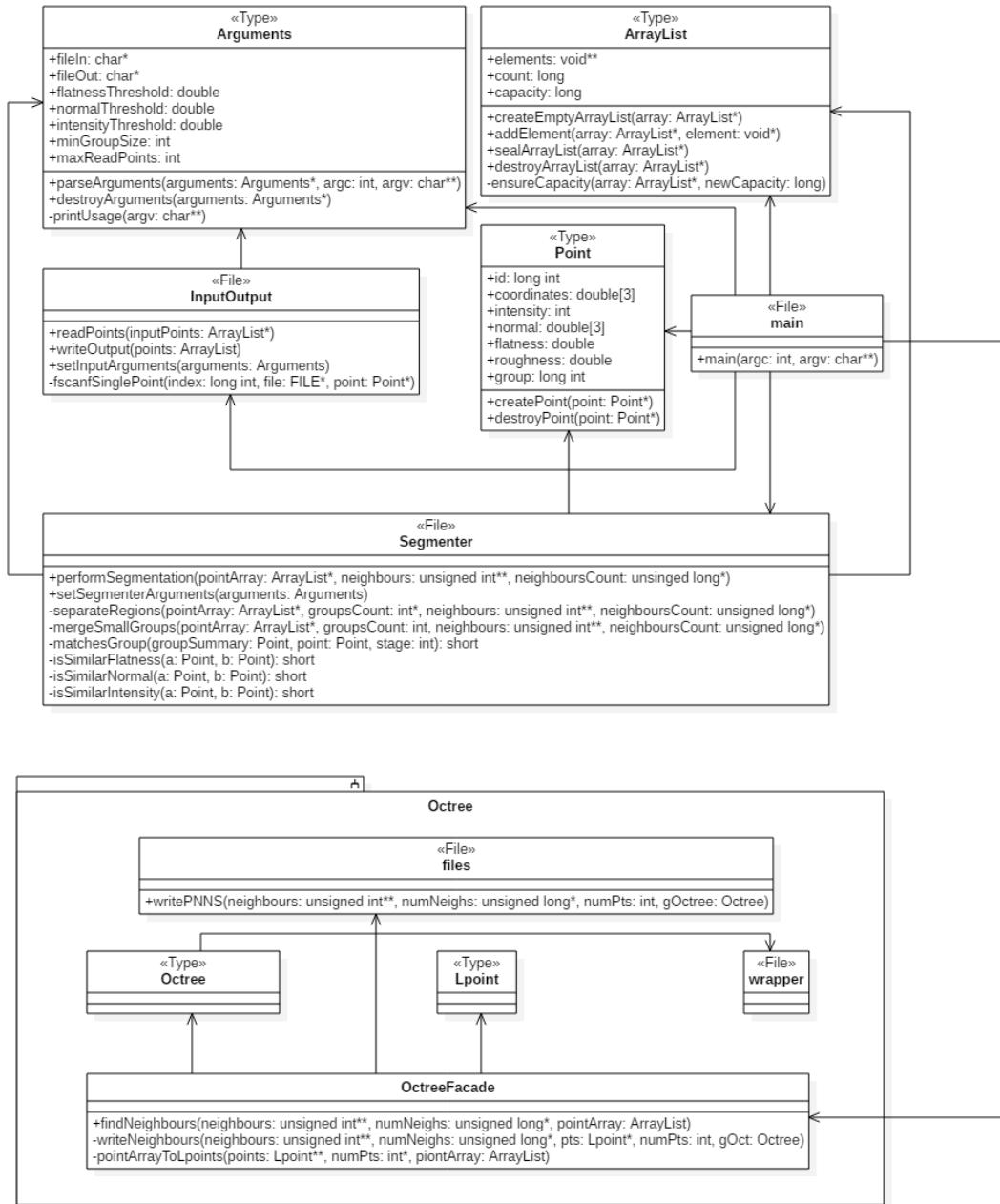


Figura 4.6: Representación estructural del nuevo programa desarrollado.

ción del algoritmo. No consiste en una lista enlazada, sino en un *array* que se expande de forma automática, en el que todos los elementos se encuentran de forma contigua en memoria.

- **InputOutput:** se trata de un módulo encargado de parte de entrada y salida del programa. Realiza la lectura y carga en memoria de la nube de puntos al comienzo de la ejecución del programa, y escribe los resultados de la segmentación cuando el algoritmo finaliza.
- **main:** módulo principal del programa, donde comienza la ejecución. Invoca funciones de los diferentes módulos según corresponda.
- **Octree:** se trata de un subsistema desarrollado principalmente por el grupo de investigación [5]. Sin entrar en detalles sobre su funcionamiento interno, es importante conocer que se trata de la parte del programa que realiza el cálculo de vecinos. Todos los módulos dentro de este subsistema han sido desarrollados de manera íntegra por el grupo de investigación, excepto *OctreeFacade*, creado por el alumno para integrarlo dentro del nuevo segmentador desarrollado.
 - **OctreeFacade:** cuenta con la función a la que se debe llamar desde fuera del subsistema *Octree* para realizar el cálculo vecinos. Como resultado de la llamada a esta función se obtendrá, para cada punto, un conjunto de puntos que se encuentran en su vecindad.
- **Point:** se trata de una estructura de datos encargada de almacenar los datos de cada uno de los puntos, para que puedan ser manejados de manera sencilla y legible.
- **Segmenter:** se trata del módulo que realiza la segmentación propiamente dicha. Aquí se realizan todas las etapas de segmentación y la separación de regiones.

4.3. Equipamiento *hardware* y *software*

En esta sección se comentarán las herramientas *software*, así como el *hardware* que se ha usado para el desarrollo de este Trabajo de Fin de Grado.

4.3.1. Equipamiento *software*

En primer lugar, en lo que respecta a la redacción de esta memoria se ha usado la tecnología de *LaTeX*. De esta forma, es posible usar de forma cómoda la plantilla requerida por la Universidad. Como entorno, se ha usado la herramienta

online *Overleaf*, ya que esta ofrece características tales como almacenamiento en la nube de documentos y control de versiones automático.

Para realizar la planificación temporal y presupuestaria del proyecto se ha usado la herramienta *Microsoft Project*. Se trata de la herramienta más conocida para realizar este tipo de tareas. Su versatilidad y su facilidad de uso la hacen candidata para usarla en prácticamente cualquier tipo de proyecto.

Para la creación de los diferentes diagramas recogidos a lo largo de esta memoria, se ha usado la herramienta *StarUML 2*, una de las más conocidas de su tipo, que además, es totalmente gratuita.

Para la codificación de los algoritmos se ha usado el entorno de desarrollo *CLion*, uno de los más conocidos y reputados para el desarrollo de programas de cualquier tipo en los lenguajes *C* o *C++*. Esta aplicación es gratuita para estudiantes universitarios.

Para la visualización de las diferentes nubes de puntos utilizadas a lo largo del proyecto, tanto aquellas segmentadas como aquellas sin segmentar, se ha usado el visualizador desarrollado previamente por el grupo de investigación al que pertenecen los directores del proyecto.

Finalmente, para realizar la gestión de la configuración se ha utilizado la herramienta *Git*, en conjunción con el servicio *Github* para mantener respaldos en la nube.

4.3.2. Equipamiento *hardware*

El proyecto se ha desarrollado única y exclusivamente usando el ordenador portátil del alumno, que tiene las características recogidas en la sección 6.2.1.

Capítulo 5

Implementación

En este capítulo se profundizarán sobre los detalles de implementación de los nuevos algoritmos desarrollados. Para ello, se expondrán y comentarán las porciones de código más relevantes para cada una de las fases de cada uno de los algoritmos, recogidos en los diagramas de flujo que se han explicado a lo largo del capítulo 4.

Se comenzará comentando el nuevo algoritmo de grafos, y se finalizará comentando el algoritmo basado en crecimiento de regiones.

5.1. Algoritmo basado en grafos

Para el desarrollo del nuevo algoritmo basado en grafos los principales cambios realizados se centran en el orden en el que se realizan las diferentes etapas, así como la conexión entre estas. Por el contrario, se han realizado pocos cambios en lo que se refiere a la parte interna de cada una de las fases. Por tanto, se comenzará comentando el funcionamiento de bucle principal, en el que se pueden observar llamadas a funciones que ejecutan las fases más relevantes del algoritmo.

El código asociado a la figura 5.1 muestra el bucle principal del nuevo algoritmo. Nótese que para simplificar la explicación, no se incluye realmente todo el código que se encuentra en el programa, sino solo aquellas líneas más relevantes.

Las dos primeras líneas de código sirven para crear un nuevo grafo en función de los puntos actuales. Posteriormente, en la tercera línea, se crea la lista de componentes en función de los nodos del grafo.

La cuarta línea de código, la llamada a la función `KNN()` sirve para realizar el cálculo de vecinos. La llamada a la función `segmentation()`, que se encuentra en la quinta línea, añade las aristas al grafo y realiza una iteración del proceso de segmentación propiamente dicho.

Finalmente, la sexta línea de código, en la cual se encuentra la llamada a la función `summarizeComponentsToPoints()` convierte las nuevas componentes en los puntos a segmentar para la siguiente iteración.

```

createGraph(&graph , currentPointsCount );
fillGraphWithPoints(&graph , currentPoints , currentPointsCount );
createComponentList(&graph , &components );

KNN( results , distances , graph , neighboursPerPoint ,
      parsedArgs . neighboursCodes [ i ] );
segmentation(&graph , &components , results ,
      distances , currentPointsCount , neighboursPerPoint ,
      parsedArgs . graphsCodes [ i ] );

summarizeComponentsToPoints(&currentPoints ,
      &currentPointsCount , graph , components ,
      points , pointsCount );

```

Figura 5.1: Bucle principal del nuevo algoritmo de grafos.

Al tratarse del bucle principal, todo este código se ejecutaría una vez por cada una de las fases de segmentación del algoritmo. Recuérdese que el usuario tiene total libertad en lo que se refiere a la elección de fases del algoritmo a ejecutar.

No se explicará el significado de todas la variables involucradas en esta porción de código, pues solo se pretende ilustrar que operaciones realiza el algoritmo en cada iteración y en qué orden, sin entrar en mayores detalles por el momento.

Una vez vista la parte del bucle principal del algoritmo, es necesario comentar brevemente el funcionamiento y los cambios realizados a dos de los procesos más relevantes: el cálculo de vecinos y la segmentación en función del grafo.

5.1.1. Cálculo de vecinos

Para el cálculo de vecinos, en este caso se usa una librería conocida como *FLANN - Fast Library for Approximate Nearest Neighbors* [4]. La porción de código que realiza el cálculo de vecinos propiamente dicho se recoge en la figura 5.2.

```

index = flann_build_index(dataset , (int) graph->nodeNum ,
      columns , &speedup , &flannParameters );

flann_find_nearest_neighbors_index(index , dataset ,
      (int) graph->nodeNum , results , distances ,
      neighboursPerPoint + 1 , &flannParameters );

```

Figura 5.2: Porción de código asociado al cálculo de vecinos.

En esta porción de código, como se puede comprobar se realiza la llamada a

dos funciones de la librería. Lo más relevante, por tanto, es el rol que desempeñan cada una de las variables que se pasan como argumentos de las funciones:

- **dataset**: se trata del conjunto de datos sobre el que realizar el cálculo de vecinos. Esto es, un array de tipo `float` con las características sobre las que se desee realizar el cálculo de vecindad (coordenadas, intensidad, normales, o una combinación de éstas), de todos los puntos.
- **graph->nodeNum**: es un número entero que representa la cantidad de puntos totales.
- **columns**: es un número entero que representa la cantidad de características por cada punto sobre las cuales se realizará el cálculo de vecindad. Por ejemplo, si se realizan sobre coordenadas, serían tres características (x, y, z).
- **speedup**: la librería usa esta variable para almacenar un cociente de ganancia de velocidad al haberlo usado.
- **flannParameters**: una estructura con diferentes argumentos que permite variar el comportamiento de la búsqueda de vecinos.
- **results**: el resultado de la búsqueda de vecinos. En este *array* de números enteros, la librería escribirá, para cada punto, los índices de sus vecinos.
- **distances**: las distancias entre vecinos. Se trata de un *array* de números en punto flotante. La librería escribirá, siguiendo el mismo orden que en el caso del *array* **results**, la distancia al vecino.
- **neighboursPerPoint**: número entero que especifica cuantos vecinos se desean hallar para cada punto.

Durante el desarrollo de esta parte del algoritmo, para garantizar la flexibilidad del mismo, fue crucial generalizar la forma en la que se genera el *array* **dataset**. El usuario puede escoger las características que serán empleadas para el cálculo de vecinos, usando para ello los argumentos de entrada conforme se explica en el apéndice B.

Para ello, el primer paso consiste en *parsear* el argumento de entrada del usuario, pasando de una simple secuencia de caracteres, a almacenar en memoria de forma cómoda banderas que indiquen que características se deben usar para el cálculo de vecinos, y cuales no. La porción de código asociada a la figura 5.3 realiza esta tarea.

De esta forma, se obtiene un *array* formado por unos y ceros, en el que cada posición representa una característica. Si una posición tiene valor cero la característica correspondiente no será usada.

```

void parseCode(short *parsedCode, const char *code) {
    for (int i = 0; i < PARSED_SIZE; ++i) {
        parsedCode[i] = 0;
    }

    for (int i = 0; code[i] != '\0'; i++) {
        switch(code[i]) {
            case 'x': parsedCode[X_PARSED_INDEX] = 1; break;
            case 'y': parsedCode[Y_PARSED_INDEX] = 1; break;
            case 'z': parsedCode[Z_PARSED_INDEX] = 1; break;
            case 'p': for (int j = X_PARSED_INDEX; j <= Z_PARSED_INDEX ; ++j)
                parsedCode[j] = 1;
            break;
            case 'i': parsedCode[LPARSED_INDEX] = 1; break;
            case 'q': parsedCode[NX_PARSED_INDEX] = 1; break;
            case 'w': parsedCode[NY_PARSED_INDEX] = 1; break;
            case 'e': parsedCode[NZ_PARSED_INDEX] = 1; break;
            case 'n': for (int j = NX_PARSED_INDEX; j <= NZ_PARSED_INDEX ; ++j)
                parsedCode[j] = 1;
            break;
            case 'a': for (int j = X_PARSED_INDEX; j <= NZ_PARSED_INDEX ; ++j)
                parsedCode[j] = 1;
            break;
        }
    }
}

```

Figura 5.3: Porción de código asociada al *parseo* del argumento de entrada.

A continuación, reutilizando el espacio de memoria creado para este *array*, se procede a almacenar datos que permitan llenar la variable *dataset* correctamente. Lo que se hace es almacenar en este *array* auxiliar referencias a las características que se van a usar, en el orden en el que serán escritas en el *dataset*, así como contar cuantas son. El código asociado a esto se encuentra en la figura 5.4.

Finalmente, se escriben los valores de las características en el *array* *dataset* atendiendo a la información proporcionada por el *array* auxiliar de referencias. Se muestra este código en la figura 5.5.

5.1.2. Segmentación

Para la realización de la segmentación en función del grafo, el primer paso consiste en crear aristas entre los nodos. Recuérdese que dichas aristas se creaban siempre entre nodos vecinos, obtenidos en la fase anteriormente comentada. A cada una de estas aristas se le daba un peso determinado en función de las características de los puntos. Este paso se realiza en la porción de código plasmada en la figura 5.6.

La llamada a la función *setWeight()* crea una arista en el grafo *graph*, asociada a los puntos con índice *i* y *knnResults[currentIndex]*, que es el índice de uno de los vecinos del punto con índice *i*. Para el cálculo del peso será tenido en cuenta el valor de distancias entre los puntos dado por la librería de cálculo de vecinos (variable *distancess[currentIndex]*). También será tenida en cuenta

```

void parseNeighboursCode(const char *code, int *columns, short *parsedCode) {
    *columns = 0;

    parseCode(parsedCode, code);

    if(parsedCode[X_PARSED_INDEX]) {
        parsedCode[*columns] = X_PARSED_INDEX;
        (*columns)++;
    }

    if(parsedCode[Y_PARSED_INDEX]) {
        parsedCode[*columns] = Y_PARSED_INDEX;
        (*columns)++;
    }

    if(parsedCode[Z_PARSED_INDEX]) {
        parsedCode[*columns] = Z_PARSED_INDEX;
        (*columns)++;
    }

    if(parsedCode[I_PARSED_INDEX]) {
        parsedCode[*columns] = I_PARSED_INDEX;
        (*columns)++;
    }

    if(parsedCode[NX_PARSED_INDEX]) {
        parsedCode[*columns] = NX_PARSED_INDEX;
        (*columns)++;
    }

    if(parsedCode[NY_PARSED_INDEX]) {
        parsedCode[*columns] = NY_PARSED_INDEX;
        (*columns)++;
    }

    if(parsedCode[NZ_PARSED_INDEX]) {
        parsedCode[*columns] = NZ_PARSED_INDEX;
        (*columns)++;
    }
}

```

Figura 5.4: Porción de código asociado al almacenamiento de referencias a características a emplear.

```

for (int i = 0; i < graph->nodeNum; ++i) {
    for (int j = 0; j < columns; ++j) {
        switch(parsedCode[j]) {
            case X_PARSED_INDEX: dataset[i * columns + j] =
                (float) (graphNodes[i].element.x * X_OFFSET); break;

            case Y_PARSED_INDEX: dataset[i * columns + j] =
                (float) (graphNodes[i].element.y * Y_OFFSET); break;

            case Z_PARSED_INDEX: dataset[i * columns + j] =
                (float) (graphNodes[i].element.z * Z_OFFSET); break;

            case I_PARSED_INDEX: dataset[i * columns + j] =
                (float) (graphNodes[i].element.intensity * I_OFFSET); break;

            case NX_PARSED_INDEX: dataset[i * columns + j] =
                (float) (graphNodes[i].element.normal[0] * N_OFFSET); break;

            case NY_PARSED_INDEX: dataset[i * columns + j] =
                (float) (graphNodes[i].element.normal[1] * N_OFFSET); break;

            case NZ_PARSED_INDEX: dataset[i * columns + j] =
                (float) (graphNodes[i].element.normal[2] * N_OFFSET); break;
        }
    }
}

```

Figura 5.5: Estructura de datos en la que se almacena cada punto.

```

for (long int i = 0; i < pointsCount; ++i) {
    for (long int j = 0; j < neighboursPerPoint + 1; ++j) {
        setWeight(graph, distances[currentIndex], i, knnResults[currentIndex], code);
        currentIndex++;
    }
}

```

Figura 5.6: Bucle de creación y asignación de peso de aristas.

la secuencia de caracteres recogida en la variable `code`, que es el argumento de entrada del usuario que determina que características tener en cuenta para calcular los pesos. Este código será *parseado* de forma muy similar a como se hacía para el cálculo de vecinos, por lo que no se profundizará en ello de nuevo.

El siguiente paso, como ya se ha explicado anteriormente, consiste en ordenar las aristas del grafo en función de su peso. No se profundizará en esto, pues la ordenación se realiza usando las librerías estándar del lenguaje C, mediante el método de *Quicksort* [7].

Finalmente, se recorren estas aristas y se van juntando las componentes según proceda. Esto se puede observar en la porción de código recogida en la figura 5.7.

```
for (long int i = 0; i < edgeCount; ++i) {
    data = edges[i].data;
    row = edges[i].row;
    col = edges[i].col;

    component1 = graphNodes[row].componentId;
    component2 = graphNodes[col].componentId;

    if (!compInt(*graph, data, &component1, &component2)) {
        mergeComponents(graph, components, &component1, &component2, data);
    }
}
```

Figura 5.7: Código asociado a la fusión de componentes.

En este código, la variable `data` representa el peso de la arista actual, mientras que `row` y `col` son los índices de los nodos conectados por dicha arista. La llamada a la función `compInt` decide si los componentes se deben juntar o no, atendiendo a la condición explicada previamente en la sección 4.1.1. Finalmente, la función `mergeComponents` fusiona las dos componentes, haciendo que todos los nodos asociados a ambas pasen a estar en una única componente.

5.2. Algoritmo basado en crecimiento de regiones

Para profundizar en el funcionamiento de este algoritmo, se comentarán secciones de código asociadas a dos de las partes más relevantes del mismo: el cálculo de vecinos y cada una de las etapas de segmentación.

5.2.1. Cálculo de vecinos

Para el cálculo de vecinos, en este caso se usa un algoritmo previamente desarrollado por los miembros del grupo de investigación. Al tratarse de un módulo no desarrollado durante la realización de este proyecto, pero que cumple a la perfección la funcionalidad que se necesita para la realización del mismo, no se

comentará como funciona detalladamente a nivel interno, sino simplemente como se obtienen los vecinos de cada punto desde una perspectiva de caja negra.

La porción de código donde realmente se obtienen los vecinos se refleja en la figura 5.8.

```

for (i = 0; i < numPts; i++) {
    for (int k = 0; numNeighs[i] < 8; ++k) {
        pointNeighs = searchNeighbors3D(&pts[i], gOct,
                                         (float) (SEARCH_RADIUS + SEARCH_RADIUS_INCREMENT * k),
                                         numNeighs + i);
    }

    neighbours[i] = malloc(numNeighs[i] * sizeof(unsigned long));
    for (j = 0; j < numNeighs[i]; j++) {
        neighbours[i][j] = pointNeighs[j]->id;
    }
}

```

Figura 5.8: Porción de código asociada a la obtención de vecinos.

Como se puede comprobar la línea más relevante de esta porción de código es aquella en la que se llama a la función `searchNeighbors3D`. Se trata de una función completamente desarrollada por el grupo de investigación antes de la realización de este proyecto, que devuelve, en este para el punto representado por la variable `pts[i]`, un conjunto de puntos vecinos, que se encuentren en un radio concreto, en este caso determinado por el valor de la expresión `SEARCH_RADIUS + SEARCH_RADIUS_INCREMENT * k`. La cantidad de vecinos encontrados se escribe en la variable `numNeighs[i]`.

En este caso se anida la llamada a esta función en un bucle. Con ello, lo que se pretende es empezar a buscar vecinos en un radio de búsqueda muy pequeño e irlo ampliando poco a poco hasta obtener 8 vecinos o más. Con esto, se consigue un número de vecinos por punto razonable (mayor que 8, pero en el mismo orden de magnitud), independientemente de que la densidad de puntos varíe en diferentes zonas. Es deseable que el número de vecinos se encuentre dentro de unos márgenes, ya de que de ser muy pequeño podría provocar problemas de sobresegmentación, al impedir el crecimiento en la fase de separación de regiones, y de ser muy grande, podría provocar desbordamientos de memoria.

Las siguientes líneas de código simplemente extraen los datos relevantes para el funcionamiento del resto del programa. En este caso, no interesa tener todas las características para cada vecino de cada punto, sino que simplemente basta con tener los identificadores que los referencien.

5.2.2. Etapa de segmentación

El código mostrado en la figura 5.9 se corresponde con el que se ejecuta en cada etapa de segmentación.

```

for (int pointIndex = 0; pointIndex < (*pointArray)->count; ++pointIndex) {
    int matched = -1;

    for (int groupIndex = 0; groupIndex < groups->count; ++groupIndex) {
        if(matchesGroup(groups->elements[groupIndex],
                          (*pointArray)->elements[pointIndex], stage)) {
            matched = groupIndex;
            break;
        }
    }

    if(matched >= 0) {
        newGroups[pointIndex] = matched;
    } else {
        newGroups[pointIndex] = groups->count;
        addElement(&groups, (*pointArray)->elements[pointIndex]);
    }
}

```

Figura 5.9: Porción de código asociada a una etapa de segmentación.

Como se puede observar, se comienza con un bucle que recorre todos los puntos de la nube. Para cada punto, se evalúa si este encaja en alguno de los grupos. Esto se determina mediante sucesivas llamadas (una por grupo existente) a la función `matchesGroup`. Esta función devolverá valor verdadero si el punto pertenece al grupo. Como se puede ser, en caso de ser así se cambia el valor de una variable, de forma que contenga el índice del grupo correspondiente. Posteriormente, se comprueba el valor de esta variable. Si tiene un índice asociado a un grupo válido, el punto se incluye en el grupo. Por el contrario, si esto no sucede, bien sea porque el punto no encaja en ningún grupo, o todavía no existe ningún grupo, se crea un nuevo grupo en el que se incluye el punto.

Por último, es preciso realizar unas aclaraciones en lo que respecta a la llamada a la función `matchesGroup`. Como se puede comprobar, recibe tres argumentos. Estos son, el punto líder el grupo actual, el punto que se está comparando con el grupo, y la variable entera `stage`, que determina en que etapa de segmentación se encuentra el algoritmo actualmente. En función del valor de esta variable, la comparación entre punto actual y punto líder se realizará en función de la planicidad, el vector normal o la intensidad.

Capítulo 6

Resultados

En este capítulo se expondrán los resultados obtenidos por los algoritmos desarrollados. Debido a que, como se comentó anteriormente, no se ha conseguido una mejora significativa de los resultados modificando el algoritmo de grafos, esta capítulo se centrará, principalmente, en las segmentaciones obtenidas mediante el algoritmo basado en crecimiento de regiones.

Como parte de estos resultados, también se comentarán las mejoras de rendimiento obtenidas, realizando una comparación de los diferentes algoritmos en lo que respecta a tiempos de ejecución y uso de memoria.

6.1. Calidad de las segmentaciones

En esta sección se expondrán y comentarán los resultados del segmentador basado en crecimiento de regiones. Para ello, se analizarán las segmentaciones de dos nubes de puntos distintas.

6.1.1. Nube de puntos 1

La primera nube de puntos que será comentada en este documento está formada por un total de 18.286.338 puntos. Para cada punto se dispone de la siguiente información:

- **Coordenadas** (posición)
- **Intensidad**
- **Vector normal**
- **Coeficientes de planicidad** (*flatness* y *roughness*)

Dado que el algoritmo basado en crecimiento de regiones desarrollado está conformado por varias etapas, cada una de las cuales realiza segmentaciones atendiendo a diferentes características de los puntos, y que están desacopladas entre

sí, es posible visualizar los resultados intermedios, que son las salidas de cada una de estas etapas.

Recordando lo visto en la sección 4.2.2, el orden de estas etapas es: segmentación en función de planicidad, segmentación en función de normales, segmentación en función de intensidad, y, finalmente, separación de regiones. Las figuras 6.1 hasta 6.6 recogen los resultados de cada una de estas etapas. Estos resultados serán analizados, uno por uno, en los próximos párrafos.

Nube de puntos sin segmentar

No obstante, antes de pasar a comentar los resultados obtenidos, es importante realizar un pequeño análisis de la nube de puntos original. La figura 6.1 es una imagen de la escena sin segmentar. Las zonas más blancas representan zonas de mayor intensidad.

Como se puede observar, existe un paso de peatones en la parte inferior de su imagen, diferenciable del resto de la carretera gracias a su mayor intensidad. Aparte de éstas, se pueden visualizar otras líneas también pintadas en la misma carretera. Por otra parte, la escena también contiene aceras con sus respectivos bordillos. Éstas se encuentran a la izquierda del paso de peatones, y en la parte superior de la imagen, al lado derecho de la carretera.



Figura 6.1: Nube de puntos 1 sin segmentar.

Nube de puntos segmentada tras la primera etapa del algoritmo

Una vez comentadas las características de la nube de puntos sin segmentar, se pueden empezar a comentar los resultados de la segmentación. Se comentará el resultado intermedio asociado a ejecutar sólo la primera etapa del algoritmo:

la segmentación en función de la planicidad. La figura 6.2 recoge los resultados de esta etapa.

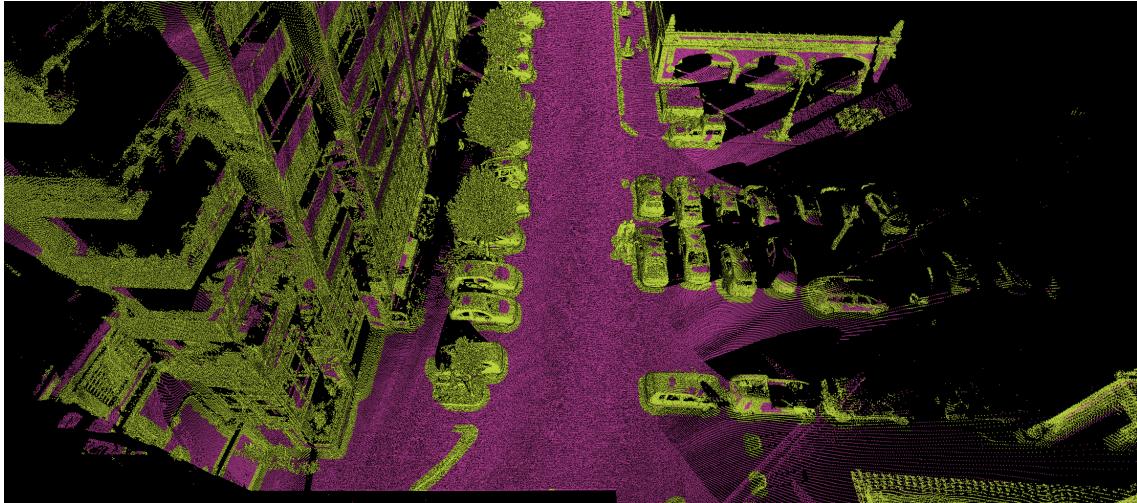


Figura 6.2: Nube de puntos 1 segmentada en función de planicidad.

Como se puede comprobar, en esta figura los puntos solo son de dos colores: fucsia y amarillo. Los puntos fucsia son aquellos que se encuentran en zonas que el algoritmo ha considerado que son planas (principalmente carreteras y aceras), mientras que los puntos amarillos se encuentran en zonas no planas (bordillos, árboles, ...).

Esta etapa tiene el objetivo único de segmentar los bordillos, pues son las únicas estructuras de interés para el proyecto que se pueden segmentar atendiendo exclusivamente a su planicidad. Como consecuencia, podría llegar a cuestionarse la efectividad del algoritmo basándose en la premisa de que los puntos asociados a bordillos no son los únicos que se encuentran en el grupo amarillo. Si bien esto es cierto, el problema es simplemente que en este punto no se ha ejecutado la fase de separación de regiones. Cuando esto se haga, cada uno de los bordillos quedará separado del resto de puntos con baja planicidad.

Nube de puntos segmentada tras las dos primeras etapas del algoritmo

Como se mencionó anteriormente, tras la etapa de segmentación en función de la planicidad se procede a segmentar en función de las normales. La figura 6.3 recoge los resultados intermedios tras la ejecución de esta segunda etapa, es decir, tras realizar una segmentación en función de la planicidad y los vectores normales.

Como se puede comprobar, la cantidad de grupos diferentes de la nube de puntos ha aumentado considerablemente. Esto se debe a que se ha incrementado la cantidad de grupos al realizar una fase más de la segmentación. Cada color representa un grupo de segmentación diferente.

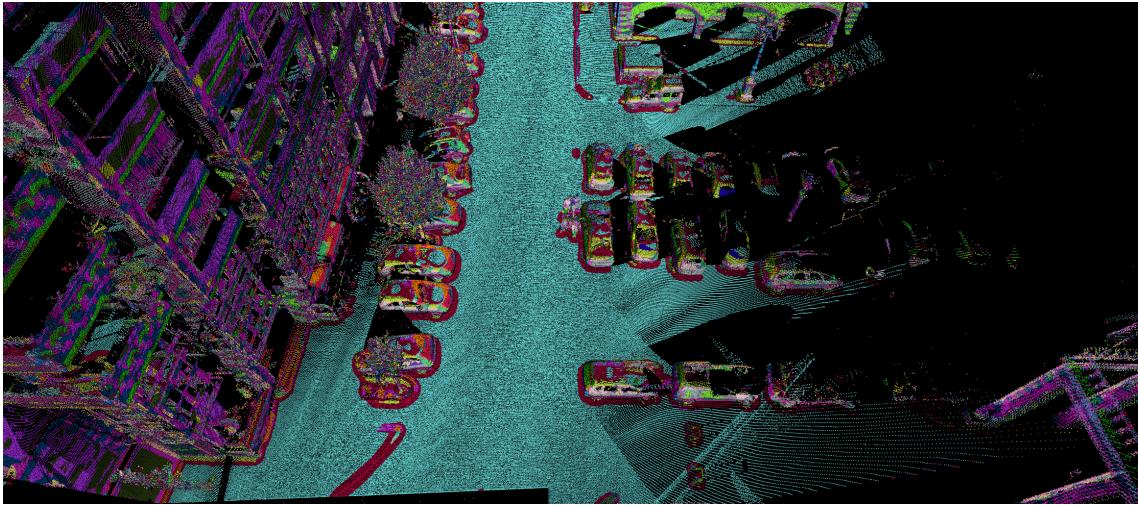


Figura 6.3: Nube de puntos 1 segmentada en función de planicidad y normales.

Como se puede comprobar, tras la ejecución de la primera etapa existían fachadas del mismo color que la carretera y la acera. Esto se debe a que ambas tenían bajos niveles de planicidad. Ahora, al tener en cuenta también las direcciones de los vectores normales, estas fachadas y la carretera se separan, al estar sus planos en ángulos completamente distintos.

Nube de puntos segmentada tras las tres primeras etapas del algoritmo

La tercera etapa del algoritmo es la segmentación en función de la intensidad. Esta etapa tiene como objetivo más evidente la segmentación de estructuras tales como pasos de peatones y otras líneas de carretera. La figura 6.4 recoge los resultados intermedios tras la ejecución de esta etapa.

Como se puede comprobar, la diferencia principal entre el resultado de esta etapa y el resultado de la etapa anterior, es que se segmenta el paso de peatones y otras líneas de la carretera. Estas estructuras aparecen ahora en un grupo representado en color naranja. Segmentar este tipo de estructuras era, precisamente, el objetivo principal de esta etapa.

Es importante señalar que las líneas centrales del paso de peatones no se han podido segmentar con la misma claridad que las líneas exteriores. Si se observa la imagen asociada a la escena original (figura 6.1), se puede comprobar que estas líneas centrales se encuentran significativamente deterioradas. Como consecuencia, no tienen la intensidad de rebote necesaria como para que el algoritmo las segmente, separándolas de la carretera.

El usuario puede regular la sensibilidad del algoritmo a la hora de segmentar en función de la intensidad. En este caso, si bien aumentar esta sensibilidad ayudaría a que el paso de peatones se segmentase con mayor claridad, también introduciría la presencia de falsos positivos (se segmentarían partes de la carretera

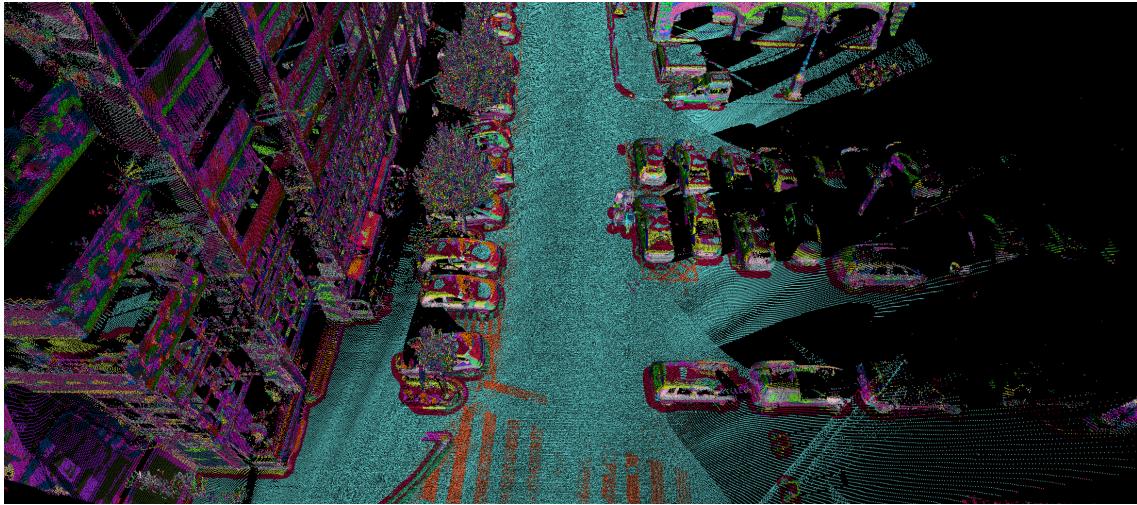


Figura 6.4: Nube de puntos 1 segmentada en función de planicidad, normales e intensidad.

en las que realmente no hay líneas de ningún tipo). La selección de parámetros adecuada es una cuestión de compromiso entre estos factores. Se puede observar el resultado de escoger un parámetro de intensidad muy agresivo en la segmentación plasmada en la figura 6.5, en la que la parte izquierda de la carretera se incluye en el mismo grupo que el paso de peatones.

Lo cierto es que, de cara a una fase posterior de clasificación, en la cual partiendo de los resultados de esta segmentación se determinase donde hay pasos de peatones y donde no, la presencia de falsos positivos sería más perjudicial que la desaparición de pequeñas partes de algunas líneas del paso de peatones, como las que se pueden observar en esta segmentación en particular. El algoritmo podría detectar el paso de peatones al haber sido segmentadas con bastante claridad aproximadamente 8 líneas que tienen forma típica de paso de peatones. Por tanto, en este sentido, se considera que la segmentación tiene la calidad suficiente en el contexto del proyecto en el que se enmarca.

Nube de puntos segmentada completamente

La cuarta y última etapa del algoritmo es la separación de regiones. Como se explicó anteriormente, esta etapa tiene como objetivo separar diferentes estructuras, que si bien pueden ser del mismo tipo, no pueden permanecer dentro del mismo grupo de segmentación (por ejemplo, dos pasos de peatones distintos). La figura 6.6 muestra el resultado de esta etapa, que es el resultado final de la segmentación de la nube de puntos.

Como se puede observar, como consecuencia de la ejecución de esta etapa cada línea del paso de peatones pasa a formar parte de un grupo distinto. Este es el comportamiento esperado y no perjudica una posterior fase de clasificación,

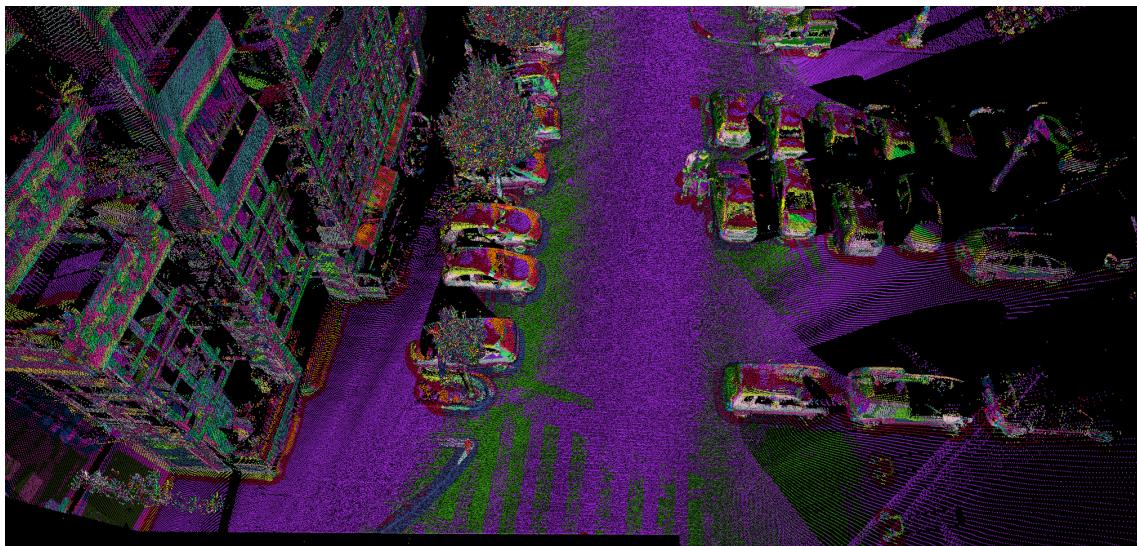


Figura 6.5: Nube de puntos 1 segmentada con falsos positivos por un uso muy agresivo del parámetro de intensidad.

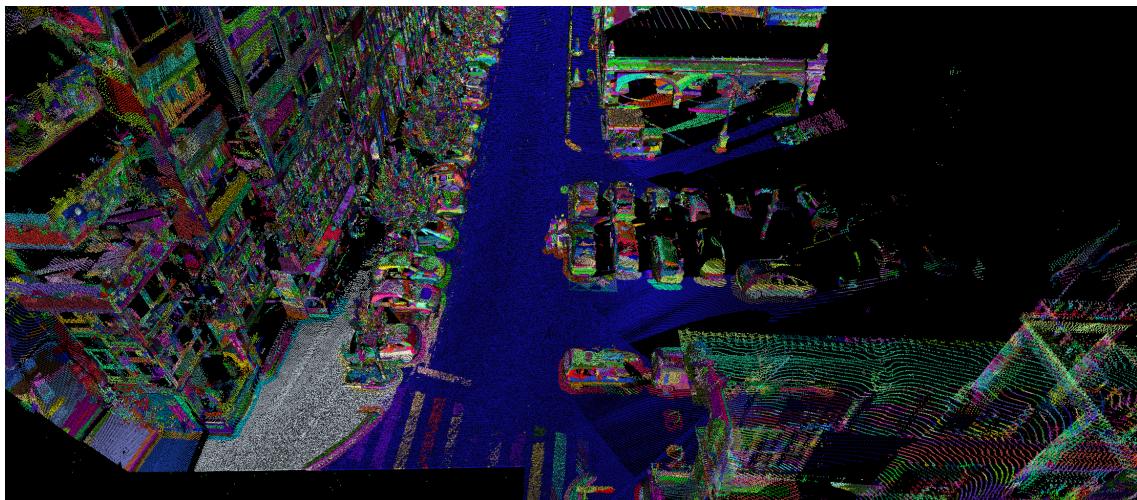


Figura 6.6: Nube de puntos 1 segmentada completamente.

puesto que la mayoría de clasificadores basan su funcionamiento en agregar diferentes grupos de segmentación para determinar la presencia de un elemento de cierta clase. En este caso, un posible clasificador agregaría los grupos asociados a cada una de las líneas del paso de peatones y determinaría que, todas ellas, forman parte de la clase paso de peatones.

El problema, como se comentó antes, existiría si una estructura que no forma parte del paso de peatones estuviese incluida en el grupo asociado a dicho paso de peatones. En otras palabras, si un futuro posible clasificador se viese obligado a disgregar los grupos para clasificar diferentes estructuras, entonces la segmentación no sería correcta. No es este el caso, la segmentación se realiza de forma correcta conforme a lo deseado y esperado.

6.1.2. Nube de puntos 2

La segunda nube de puntos está formada por un total de 26.318.212 puntos. Para cada punto se dispone de la siguiente información:

- **Coordenadas** (posición)
- **Intensidad**
- **Vector normal**
- **Coeficientes de planicidad** (*flatness* y *roughness*)

Nube de puntos sin segmentar

Al tratarse de una nube de puntos de un tamaño tan grande, cuenta con muchas estructuras a segmentar. Existen muchos pasos de peatones y líneas, así como bordillos y otras estructuras de interés a lo largo y ancho de esta nube de puntos. Sin embargo, una de las zonas más relevantes de la nube de puntos es la que se encuadra dentro de la escena recogida en la figura 6.7. Se trata de una zona particularmente interesante porque reúne, en unos pocos metros, dos pasos de peatones distintos y otras marcas viales pintadas en la carretera, además de un bordillo.

Nube de puntos segmentada completamente

La figura 6.8 plasma los resultados de la segmentación completa, es decir, una vez ejecutadas todas las etapas.

Como se puede comprobar, se obtiene una segmentación de bastante calidad, en la que se separan perfectamente los pasos de peatones y el resto de marcas viales de la carretera propiamente dicha. También se puede observar como se segmenta la parte más pronunciada del bordillo en el lado izquierdo de la escena, es decir, su totalidad salvo la rampa del paso de peatones.



Figura 6.7: Nube de puntos 2 sin segmentar.

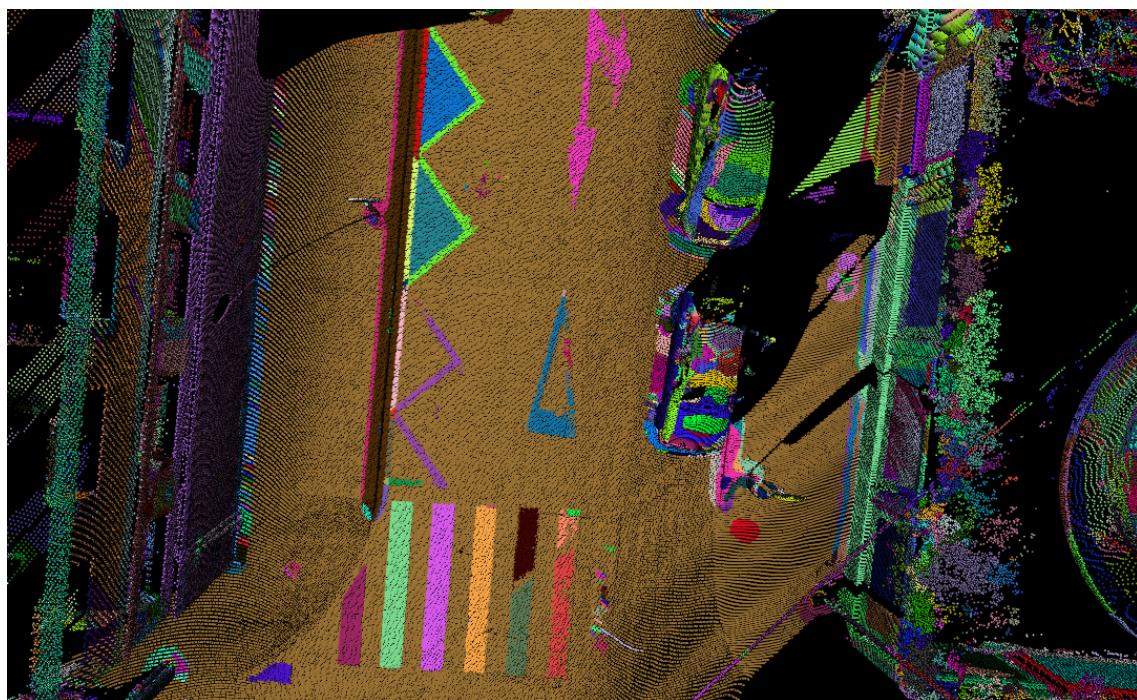


Figura 6.8: Nube de puntos 2 segmentada completamente.

6.1.3. Perspectiva general de la nube de puntos

La figura 6.9 recoge la segmentación de esta misma nube de puntos desde una perspectiva más general. En ella, se puede observar claramente el carácter específico del dominio. Como se puede comprobar, la carretera, y dentro de ésta, las marcas viales, son claramente segmentados con una calidad considerablemente buena. Sin embargo, la segmentación que se produce en árboles, coches y fachadas es bastante mejorable. Este es el comportamiento esperado del algoritmo. No se trata de realizar una segmentación perfecta de toda la nube, sino de aquellas estructuras críticas para el proyecto. Es por ello que se trata de un algoritmo de propósito específico.

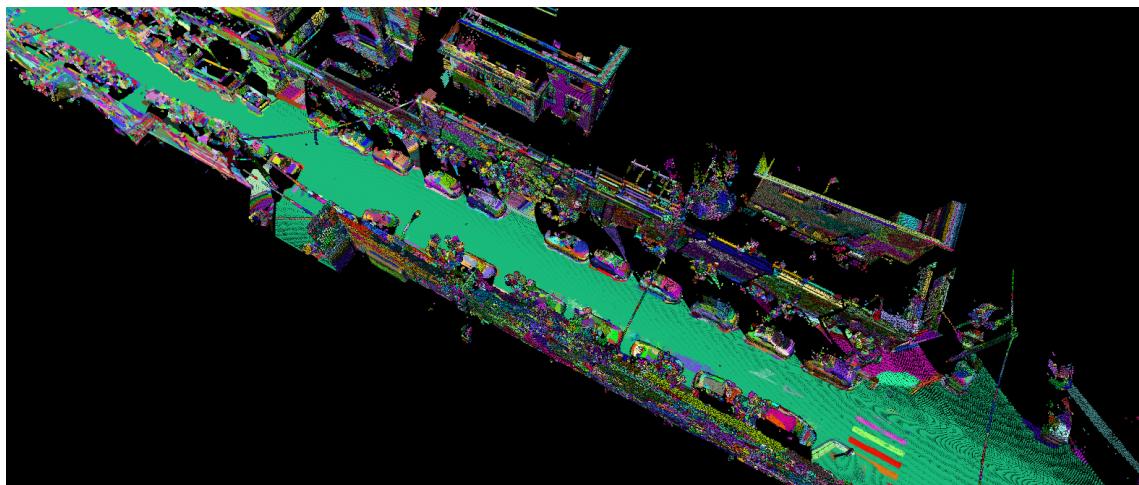


Figura 6.9: Nube de puntos 2 segmentada desde una perspectiva general.

6.2. Comparación de rendimientos

En esta sección se realizarán comparaciones de rendimientos de los tres algoritmos tratados a lo largo de este Trabajo de Fin de Grado: el algoritmo de grafos original desarrollado por el grupo de investigación, el algoritmo de grafos modificado por el alumno, y el nuevo algoritmo basado en crecimiento de regiones, también desarrollado por el alumno.

En primer lugar, se indicarán las especificaciones del equipo sobre el cual se ejecutaron las pruebas de rendimiento. A continuación, se explicará el método de medición de tiempos de ejecución y usos de memoria. Finalmente se presentarán y comentarán los resultados obtenidos.

6.2.1. Características del equipo

El equipo en el que se realizaron las pruebas de rendimientos cuenta con las siguientes características:

- **Modelo:** MSI GP72 2QE Leopard Pro
- **CPU:** Intel Core i7 5700HQ @ 2.70GHz
- **Memoria RAM:** 2x8GB DDR3L-1600MHz (16GB en total)
- **Disco duro:** 1TB SATA 7200RPM (HGST Travelstar 7K1000 HTS721010A9E630)
- **Tarjeta gráfica:** NVIDIA GeForce GTX 950M (2GB DDR3)
- **Compilador:** gcc 5.4.0

6.2.2. Método de medición

Para cada uno de los algoritmos, se han obtenido datos de tiempos de ejecución y picos de máximo uso de memoria con nubes de puntos de diferentes tamaños. Ello ha sido posible al uso de una de las versiones más recientes del comando `time`, que proporciona no solo tiempos de ejecución de procesos, sino otras muchas estadísticas sobre los mismos, entre los que se encuentra el pico de uso máximo de memoria.

Este comando ha sido ejecutado sobre un sistema operativo *Ubuntu 16.04.4 LTS* de la siguiente forma:

```
/usr/bin/time -v proceso
```

Donde `proceso` representa el comando a ejecutar sobre el que se desean realizar las mediciones.

Para que sirva a modo de ejemplo, se mostrará el resultado de realizar las mediciones sobre el comando `ls`. Se obtiene la salida reflejada en la figura 6.10 (se ha eliminado la propia salida del comando `ls`, ya que no resulta relevante).

Las líneas que se han tenido en cuenta para evaluar los rendimientos de los diferentes algoritmos han sido *Elapsed (wall clock) time*, que mide el tiempo total de ejecución del programa y *Maximum resident set size* que mide el pico máximo de uso de memoria.

Usando este comando se ha creado un *script* en *Bash* que, partiendo de una nube de 26.318.212 de puntos, va generando nubes de diferentes tamaños y ejecuta los diferentes algoritmos sobre cada una de estas nubes, realizando las mediciones de rendimiento. De esta forma, se obtienen de forma automática las estadísticas de rendimiento para todos los algoritmos con nubes de diferentes tamaños.

```

samuel@Ada:~$ /usr/bin/time -v ls
      Command being timed: "ls"
      User time (seconds): 0.00
      System time (seconds): 0.01
      Percent of CPU this job got: 115%
      Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.01
      Average shared text size (kbytes): 0
      Average unshared data size (kbytes): 0
      Average stack size (kbytes): 0
      Average total size (kbytes): 0
      Maximum resident set size (kbytes): 1000
      Average resident set size (kbytes): 0
      Major (requiring I/O) page faults: 0
      Minor (reclaiming a frame) page faults: 300
      Voluntary context switches: 0
      Involuntary context switches: 0
      Swaps: 0
      File system inputs: 0
      File system outputs: 0
      Socket messages sent: 0
      Socket messages received: 0
      Signals delivered: 0
      Page size (bytes): 4096
      Exit status: 0

```

Figura 6.10: Ejemplo de salida del comando `time`

6.2.3. Evaluación de rendimientos

La tabla 6.1 recoge los picos de uso de memoria de los tres algoritmos ejecutándose sobre nubes de puntos de diferentes tamaños. En concreto, se presentan datos de mediciones rendimiento sobre nubes que van desde 1.000.000 de puntos hasta 26.318.212, con un paso de 1.000.000. Se disponen de datos con un paso de 500.000, pero se ha decidido omitirlos para elaborar una tabla más corta y legible.

En primer lugar, es preciso aclarar por qué no hay datos para gran parte de las ejecuciones de los dos algoritmos de grafos. Esto se debe a que, a partir de cierto número de puntos a procesar, el uso de memoria es tan alto, que no es posible finalizar correctamente la ejecución del algoritmo. Obsérvese como de hecho, los datos comienzan a desaparecer cuando el uso de memoria se aproxima a 14 GB, que junto a 2GB que probablemente están siendo usados por el sistema y otros procesos, llenan los 16 GB de memoria RAM disponibles. Aclarado esto, se puede proceder a comentar los resultados obtenidos.

Como se puede comprobar, el algoritmo de grafos modificado tiene un uso de memoria ligeramente superior al original. Esto es debido a que la posibilidad de efectuar varias etapas del algoritmo en una sola ejecución, así como la ausencia de creación de ficheros temporales, tiene un costo asociado en el uso de memoria. Es preciso almacenar más datos, sobre todo durante la transición entre diferentes etapas.

Por otro lado, resulta realmente interesante el bajo uso de memoria del algoritmo basado en crecimiento de regiones, considerablemente menor a ambos

Número de puntos	Grafos original	Grafos modificado	Algoritmo nuevo
1.000.000	1,1044	1,2626	0,4672
2.000.000	2,2060	2,5219	0,9190
3.000.000	3,3076	3,9452	1,3786
4.000.000	4,4091	5,2590	1,8404
5.000.000	5,5107	6,8463	2,3003
6.000.000	6,6122	8,2149	2,7868
7.000.000	7,7138	9,6771	3,2646
8.000.000	8,8154	10,9085	3,7067
9.000.000	9,9169	12,6531	4,1629
10.000.000	11,0185	13,6010	4,6125
11.000.000	12,1200	-	5,0603
12.000.000	13,2216	-	5,5364
13.000.000	-	-	6,0082
14.000.000	-	-	6,4808
15.000.000	-	-	7,0230
16.000.000	-	-	7,5374
17.000.000	-	-	8,0578
18.000.000	-	-	8,5845
19.000.000	-	-	9,0995
20.000.000	-	-	9,5926
21.000.000	-	-	10,1045
22.000.000	-	-	10,6040
23.000.000	-	-	11,0903
24.000.000	-	-	11,5554
25.000.000	-	-	12,0453
26.000.000	-	-	12,5063
26.318.212	-	-	12,6702

Cuadro 6.1: Picos de usos de memoria de los diferentes algoritmos (en gigabytes)

algoritmos basados en grafos. Esto le permite realizar segmentaciones sobre nubes de tamaño considerablemente mayor. Obsérvese como mientras no es posible ejecutar los algoritmos de grafos con nubes de más de 12.000.000 de puntos, el algoritmo basado en crecimiento de regiones llega a segmentar la totalidad de la nube, de 26.318.212 puntos, sin llegar a consumir 13 GB de RAM.

Una vez comentados los resultados en lo que respecta a los picos de uso de memoria, solo falta comentar los tiempos de ejecución. La tabla 6.2 recoge los datos asociados a estos tiempos de ejecución. Nuevamente, no se han podido obtener mediciones válidas para parte de las ejecuciones de los algoritmos de grafos, por los problemas de uso de memoria anteriormente comentados.

Número de puntos	Grafos original	Grafos modificado	Algoritmo nuevo
1.000.000	18,94	15,51	13,69
2.000.000	38,18	31,43	26,88
3.000.000	59,58	49,60	40,87
4.000.000	82,90	66,90	55,11
5.000.000	102,72	87,28	69,43
6.000.000	127,99	106,31	85,08
7.000.000	166,12	137,48	100,61
8.000.000	187,99	157,37	114,92
9.000.000	196,12	161,12	129,69
10.000.000	215,65	180,70	144,90
11.000.000	236,98	-	162,54
12.000.000	257,75	-	175,06
13.000.000	-	-	190,01
14.000.000	-	-	205,29
15.000.000	-	-	222,11
16.000.000	-	-	239,39
17.000.000	-	-	258,16
18.000.000	-	-	276,51
19.000.000	-	-	294,38
20.000.000	-	-	309,69
21.000.000	-	-	328,79
22.000.000	-	-	344,76
23.000.000	-	-	361,54
24.000.000	-	-	380,82
25.000.000	-	-	401,15
26.000.000	-	-	417,65
26.318.212	-	-	423,46

Cuadro 6.2: Tiempos de ejecución de los diferentes algoritmos (en segundos)

En esta ocasión se puede comprobar como el algoritmo de grafos modifica-

do supone una mejora sobre el original. Es altamente probable que esta mejora radique, principalmente, en el hecho de que ya no se necesitan leer ni escribir ficheros temporales. Las escrituras en disco son unas de las operaciones más costosas que puede realizar un computador. Por ello, reducir al máximo la cantidad de este tipo de operaciones puede tener un impacto positivo muy notable en el rendimiento de un programa, y este parece ser el caso.

Por otro lado, nuevamente el algoritmo basado en crecimiento de regiones parece ser notablemente más eficiente que los dos basados en grafos, ya no solo en uso de memoria, sino también en tiempos de ejecución.

Finalmente, para visualizar los datos anteriormente presentados de manera más cómoda, se han representado mediante gráficas. Las figuras 6.11 y 6.12 representan los datos de usos de memoria y tiempos de ejecución respectivamente.

Estas gráficas permiten observar como, tanto el uso de memoria como los tiempos de ejecución escalan de manera lineal (salvo por pequeñas perturbaciones despreciables) para los tres algoritmos. También permite comparar de forma visual los usos de memoria y los tiempos de ejecución de los tres algoritmos. En este sentido, no hay nada nuevo que comentar, el algoritmo basado en crecimiento de regiones es más rápido y usa menos memoria que los dos algoritmos de grafos, cada uno de los cuales supera al otro o bien solo en tiempo de ejecución, o bien solo en uso de memoria.

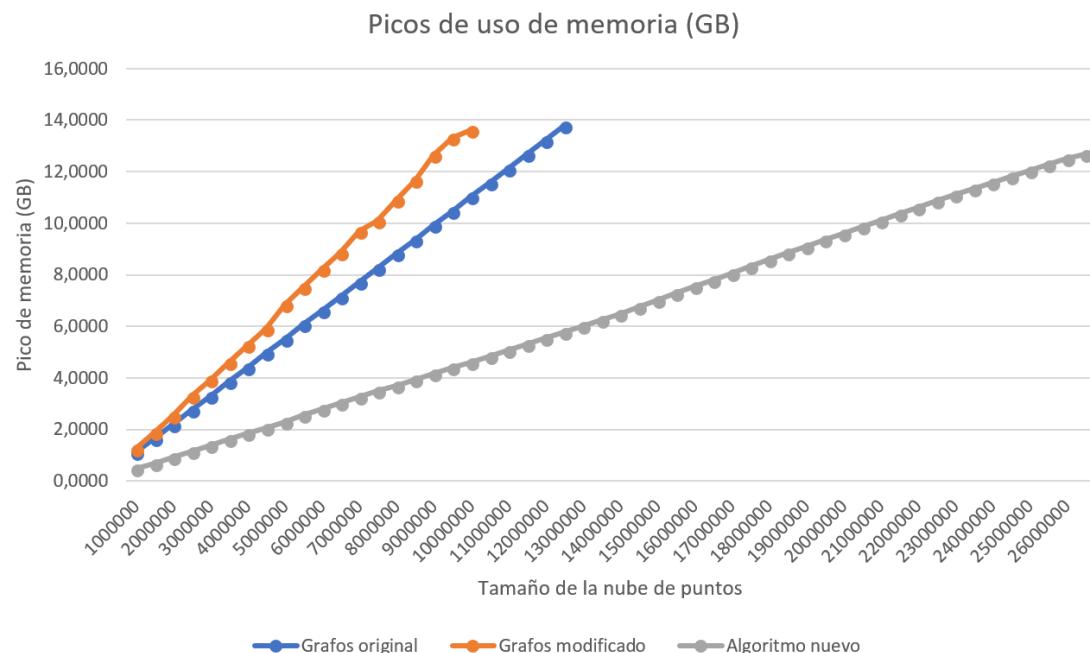


Figura 6.11: Gráfico comparativo de picos de uso de memoria.



Figura 6.12: Gráfico comparativo de tiempos de ejecución.

Capítulo 7

Conclusiones y posibles ampliaciones

7.1. Conclusiones

El objetivo principal de este Trabajo de Fin de Grado consistía en desarrollar un algoritmo capaz de segmentar estructuras viales de interés en el contexto del proyecto en el que se enmarca. Se partía de un algoritmo de propósito general, además de otros muchos programas y herramientas desarrollados por el grupo de investigación (visualizador LiDAR, algoritmo Octree, etc.)

El algoritmo de partida no daba lugar a segmentaciones que pudiesen llegar a ser usadas para el reconocimiento de cualquiera de las estructuras viales relevantes para el proyecto (pasos de peatones, bordillos, ...). Al tratarse de un segmentador de propósito general, pensado para segmentar cualquier tipo de estructuras de una nube de puntos LiDAR, se propuso, como contrapartida, un nuevo algoritmo de propósito específico, diseñado para segmentar las estructuras viales urbanas interesantes para este proyecto en particular.

Como resultado, se ha desarrollado el algoritmo de segmentación basado en crecimiento de regiones, capaz de segmentar de manera bastante robusta y clara este tipo de estructuras. Se ha demostrado, además, que el uso de segmentadores de propósito general puede no ser la mejor opción para aplicaciones específicas como las de este Trabajo de Fin de Grado.

Este nuevo segmentador mejora al anterior no solo en lo que se refiere a la calidad de las segmentaciones en nubes de puntos LiDAR terrestre para este dominio en particular, si no también en lo que se refiere al rendimiento. Como se ha podido comprobar, tiene un tiempo de ejecución y un uso de memoria considerablemente más bajos que el algoritmo original, lo que permite segmentar nubes que contengan una mayor cantidad de puntos.

No obstante, este nuevo segmentador no es la única aportación de valor de este Trabajo de Fin de Grado. Se ha conseguido flexibilizar y mejorar el funcio-

namiento del algoritmo de grafos original.

Por último, este Trabajo de Fin de Grado demuestra que el reconocimiento de estructuras viales mediante datos LiDAR es una tarea viable. Los productos aquí desarrollados no acometen esta tarea por completo, pero sientan unas bases muy importantes y resuelven algunos de los problemas más importantes para la realización de la misma.

7.2. Posibles ampliaciones

Como posibles ampliaciones, podría ser interesante incorporar más datos a las nubes de puntos a segmentar (por ejemplo, datos de color RGB), para poder obtener segmentaciones más fiables y de mayor calidad. Con este tipo de datos se podrían segmentar ciertas estructuras con un mayor nivel de fiabilidad, como por ejemplo las aceras, las cuales actualmente no se logran separar de las carreteras si no existe un bordillo lo suficientemente pronunciado.

Por otra parte, considero que el siguiente paso sería realizar un clasificador, que, partiendo de los resultados del segmentador desarrollado, clasifique las diferentes estructuras de interés, así como la extracción de las principales características de los elementos viales identificados.

Apéndice A

Manuales técnicos

A.1. Instrucciones de compilación

A.1.1. Programa del nuevo algoritmo basado en grafos

Antes de proceder a la compilación del programa asociado al nuevo algoritmo basado en grafos, es necesario tener instalada la librería *FLANN - Fast Library for Approximate Nearest Neighbors* [4]. En la mayoría de distribuciones *Linux* basadas en *Debian*, esta librería se encuentra en los repositorios, por lo que basta con ejecutar el siguiente comando para instalar su última versión:

```
sudo apt install libflann1.8
```

De no ser posible esta vía de instalación, la descarga se podría efectuar desde la web oficial del desarrollador [4].

Una vez hecho esto, dado que se proporciona un fichero *makefile* junto al código, basta con ejecutar el comando `make` sobre el mismo directorio sobre el que se encuentra dicho fichero, junto al código fuente. Tras realizar este paso, se genera el fichero *GraphSegmenter*, que es el ejecutable asociado al programa.

A.1.2. Programa del algoritmo basado en crecimiento de regiones

En este caso, no se necesita instalar ninguna librería para poder compilar el programa. Nuevamente, se proporciona un fichero *makefile* junto al código. Por tanto, basta con ejecutar el comando `make` sobre el directorio sobre el que se encuentra dicho fichero, junto al código fuente. Tras realizar este paso, se genera el fichero *Segmenter*, que es el ejecutable asociado al programa.

Apéndice B

Manuales de usuario

En este apéndice se explicarán las posibilidades que ofrecen al usuario los dos segmentadores desarrollados y como usarlos. Esta explicación se enfocará principalmente en el uso de los parámetros de entrada de los programas codificados, ya que esta es la única forma de la cual dispone el usuario para interactuar con los programas y modificar los comportamientos predeterminados.

B.1. Segmentador basado en grafos

La plantilla del comando a emplear para ejecutar el nuevo segmentador basado en grafos es la siguiente:

```
./GraphSegmenter ficheroEntrada.xyz ficheroSalida.xyz iteracion1  
[iteracion2] [...]
```

Donde cada una de las partes del comando representa lo siguiente:

- **./GraphSegmenter**: ejecutable compilado del segmentador.
- **ficheroEntrada.xyz**: fichero de entrada. Ruta del fichero que contiene la nube de puntos a segmentar.
- **ficheroSalida.xyz**: fichero de salida. Ruta del fichero en el que se desea que se escriba el resultado de la segmentación, es decir, la nube segmentada lista para ser visualizada.
- **iteracion1**: el código que indica en base a qué características de los puntos se realizará la búsqueda de vecinos y el cálculo de pesos de las aristas en la primera iteración. Se explicará el formato de este código a continuación.
- **[iteracion2]**: análogo a **iteracion1** pero para la segunda iteración. Este argumento es opcional.

- [...] : uno o más códigos opcionales, análogos a `iteracion1` e `[iteracion2]` que especifican el comportamiento de las iteraciones correspondientes.

B.1.1. Códigos de iteraciones

Los códigos `iteracion1` e `iteracion2` mencionados anteriormente, que permiten modificar que características de los puntos se tienen en cuenta en cada una de las iteraciones, tienen el siguiente formato general:

`vecinos:aristas:repeticiones`

Donde cada una de las partes del código representa lo siguiente:

- **vecinos**: secuencia de caracteres que representa las características de los puntos a tener en cuenta para el cálculo de vecinos.
- **aristas**: secuencia de caracteres que representa las características de los puntos a tener en cuenta para el cálculo de pesos de aristas.
- **repeticiones**: número entero que indica la cantidad de iteraciones reales a realizar con los criterios actuales. Permite abreviar el comando de ejecución del algoritmo, ya que de esta forma no es necesario repetir un mismo código varias veces si se quieren realizar varias iteraciones del segmentador usando los mismos criterios.

Las cadenas de caracteres **vecinos** y **aristas** tienen exactamente el mismo formato y restricciones. Son cadenas de caracteres, en las cuales cada carácter representa una o varias características del punto a tener en cuenta. En particular, estas cadenas pueden contener los siguientes caracteres:

- **x**: coordenadas *x* de los puntos.
- **y**: coordenadas *y* de los puntos.
- **z**: coordenadas *z* de los puntos.
- **p**: posiciones de los puntos, es decir, coordenadas *x, y, z* de los mismos.
- **i**: intensidades de los puntos.
- **q**: componentes *x* de los vectores normales de los puntos.
- **w**: componentes *y* de los vectores normales de los puntos.
- **e**: componentes *z* de los vectores normales de los puntos.

- n : vectores normales de los puntos, es decir, las componentes x, y, z de las normales.
- a : todas las características de los puntos: posiciones, intensidades y vectores normales.

Para formar las cadenas de caracteres **vecinos** y **aristas** los caracteres anteriormente comentados se pueden combinar de cualquier forma y en cualquier orden. Las características correspondientes a los caracteres que se encuentren en la cadena serán tenidos en cuenta para realizar los cálculos correspondientes.

Por ejemplo, la cadena p_i , o su equivalente xyz_i , indica que se tengan en cuenta las coordenadas x, y, z de los puntos, así como sus intensidades para realizar los cálculos.

Cabe destacar que si las características a tener en cuenta se repiten, no se reportará ningún error. Siempre se tendrá en cuenta el resultado de realizar la unión de los conjuntos de las características asociadas a cada uno de los caracteres. Por ejemplo, la cadena de caracteres $xyzyp$ es totalmente equivalente a p .

B.1.2. Ejemplos de uso

A continuación, se muestran y explican dos ejemplos de ejecuciones del segmentador, para clarificar lo anteriormente explicado.

```
./GraphSegmenter ficheroEntrada.xyz ficheroSalida.xyz pi:n:3
./GraphSegmenter ficheroEntrada.xyz ficheroSalida.xyz p:z:1 pi:x:3
n:i:2
```

El primer comando se podría interpretar de forma textual como: “realizar tres iteraciones con cálculo de vecinos atendiendo a las coordenadas x, y, z de los puntos, así como su intensidad; y teniendo en cuenta las normales para calcular los pesos de las aristas.”

El segundo comando se podría interpretar de forma textual como: “realizar una iteración con cálculo de vecinos atendiendo a las coordenadas x, y, z de los puntos; y teniendo en cuenta la coordenada z para calcular los pesos de las aristas. Después, realizar tres iteraciones con cálculo de vecinos atendiendo a las coordenadas x, y, z de los puntos, así como su intensidad; y teniendo en cuenta la coordenada x para calcular los pesos de las aristas. Finalmente, realizar dos iteraciones con cálculo de vecinos atendiendo a los vectores normales de los puntos; y teniendo en cuenta la intensidad para calcular los pesos de las aristas”

B.2. Segmentador basado en crecimiento de regiones

Para ejecutar el segmentador basado en crecimiento de regiones usando el valor por defecto en todos los parámetros, se debe introducir la siguiente línea por consola de comandos:

```
./Segmenter ficheroEntrada.xyz ficheroSalida.xyz
```

Donde cada una de la partes del comando representa lo siguiente:

- **./Segmenter**: ejecutable compilado del segmentador.
- **ficheroEntrada.xyz**: fichero de entrada. Ruta del fichero que contiene la nube de puntos a segmentar.
- **ficheroSalida.xyz**: fichero de salida. Ruta del fichero en el que se desea que se escriba el resultado de la segmentación, es decir, la nube segmentada lista para ser visualizada.

B.2.1. Parámetros opcionales

A este comando base, se le pueden añadir varios parámetros opcionales para customizar la ejecución del algoritmo:

- **f=X**: permite customizar el valor umbral de planicidad para separar bordillos de carreteras. X puede tomar cualquier valor de número en punto flotante. El valor por defecto de este parámetro es 0.005 (equivalente a introducir el argumento **f=0.005**).
- **n=X**: permite customizar el valor que indica cual debe ser la diferencia entre dos vectores normales para considerar que pertenecen a planos distintos. X puede tomar cualquier valor de número en punto flotante. El valor por defecto de este parámetro es 0.25 (equivalente a introducir el argumento **n=0.25**).
- **i=X**: permite customizar el valor umbral de intensidad para separar estructuras como pasos de peatones basándose en esta característica. X puede tomar cualquier valor de número entero. El valor por defecto de este parámetro es de 18000 (equivalente a introducir el argumento **i=18000**).
- **s=X**: permite customizar el número de puntos que debe tener un grupo para ser considerado grupo pequeño, que debe ser fusionado con uno grande. X puede tomar cualquier valor de número entero. El valor por defecto de este parámetro es de 20 (equivalente a introducir el argumento **s=20**).

- **r=X**: permite limitar el número de puntos a procesar por el segmentador. Puede ser útil para realizar pequeñas pruebas, segmentando solo una parte de una nube de puntos en lugar de la totalidad de la misma, logrando tiempos de ejecución más rápidos. X puede tomar cualquier valor de número entero. Si no se introduce este parámetro, se leerán todos los puntos del archivo de entrada.

Los argumentos anteriormente explicados pueden ser introducidos en cualquier orden.

B.2.2. Ejemplos de uso

A continuación, se muestra y explica un ejemplo de ejecución del segmentador incluyendo alguno de estos parámetros.

```
./Segmenter ficheroEntrada.xyz ficheroSalida.xyz r=3000000 n=0.3
i=3500
```

Este comando, se podría interpretar de forma textual como: “realizar una segmentación sobre los primeros tres millones de puntos de la nube almacenada en el fichero **ficheroEntrada.xyz**, tomando como valor de separación de normales 0.3, y como valor de separación de intensidad 3500, y escribir el resultado de la segmentación en el fichero **ficheroSalida.xyz**”.

Bibliografía

- [1] Carter, Jamie; Keil Schmid; Kirk Waters; Lindy Betzhold; Brian Hadley; Rebecca Mataosky; Jennifer Halleran, "Lidar 101: An Introduction to Lidar Technology, Data, and Applications."(NOAA) Coastal Services Center, 2012 (<https://coast.noaa.gov/data/digitalcoast/pdf/lidar-101.pdf>)
- [2] Lovas Tamás, "Data acquisition and integration. Laser Scanning", 2010 (http://www.tankonyvtar.hu/en/tartalom/tamop425/0027_DAI4/ch01.html)
- [3] Condiciones generales de contrato. Gas Natural Fenosa. Disponible en https://www.gasnaturalfenosa.es/hogar/luz_o_gas/contratar_luz_o_gas/tarifas_luz_y_gas/tarifas_estables. Consultado el 21 de junio de 2018.
- [4] FLANN - Fast Library for Approximate Nearest Neighbors (<http://www.cs.ubc.ca/research/flann/>). Consultado el 1 de julio de 2018.
- [5] Jorge Martínez, Francisco F. Rivera, José C. Cabaleiro, David L. Vilariño, Tomás F. Pena, "A rule-based classification from a region-growing segmentation of airborne lidar", <https://dx.doi.org/10.11117/12.2240750>
- [6] David L. Vilariño, José C. Cabaleiro, Jorge Martínez, Francisco F. Rivera, Tomás F. Pena, "Graph-based approach for airborne light detection and ranging segmentation", *Journal of Applied Remote Sensing*, SPIE, Vol. 11, No. 1, marzo 2017. (<https://dx.doi.org/10.11117/1.JRS.11.015020>)
- [7] Quicksort. Artículo de Wikipedia (<https://en.wikipedia.org/wiki/Quicksort>). Consultado el 25 de junio de 2018.
- [8] R.C. Gonzalez e R.E. Woods, *Digital image processing*, 3^a edición, Prentice Hall, New York, 2007.
- [9] Normativa de la Universidad de Santiago de Compostela para la contratación de personal con cargo a actividades de investigación, desarrollo e innovación (I+D+i) (http://www.usc.es/export9/sites/webinstitucional/gl/normativa/descargas/documentos/Normativa_para_a_contratacion_de_persoal_con_cargo_a_actividades_de_investigacion_desenvolvimento_e_innovacion.pdf)