INTERNATIONAL DOCTORAL
SCHOOL OF THE USC

Miguel
Yermo García

PhD Thesis

# HPC Solutions for ALS Point Cloud Processing in Pathfinding and Powerline Detection and Characterization

Santiago de Compostela, 2024

**Doctoral Programme in Information Technology Research**

Ph.D. Thesis

# HPC SOLUTIONS FOR ALS POINT CLOUD PROCESSING IN PATHFINDING AND POWERLINE DETECTION AND CHARACTERIZATION

Miguel Yermo García

Directors: Prof. Dr. Francisco Fernández Rivera and Prof. Dr. Tomás Fernández Pena

Tutor: Prof. Dr. Francisco Fernández Rivera

**DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY RESEARCH**

SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

*Never confuse education with intelligence, you can have a Ph.D. and still be an idiot.*
— Richard P. Feynman, *Nobel Prize in Physics.*

*Certainty of death. Small chance of success. What are we waiting for?*
— Gimli, son of Glóin, *The Lord of the Rings.*

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**To my family, friends, and advisors.**

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

## Acknowledgements

Thanks to Francisco F. Rivera, José C. Cabaleiro, David L. Vilariño, and Tomás F. Pena, for their guidance and support. All four should be considered my advisors equally.

Thanks to Jorge Martínez, my brother-in-arms during the first two years at CiTIUS, for helping me whenever I needed it in the arduous path of learning to program in C from scratch while starting my research.

Thanks to Ruben Laso, for **const**antly feeding back my desire to optimize with his eagerness to optimize everything. Thanks to him, I acquired the knowledge of several years of C++ in just a few months.

Thanks to Silvia Alcaraz, the best rubber ducky anyone could wish for. She was always available to listen and give well-formed opinions and advice on any subject.

Thanks to Alberto Esmorís for our endless chats about any topic one can imagine and for his great ideas, which have undoubtedly helped me throughout this thesis.

May 17, 2024

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

## Additional acknowledgements

I had the opportunity to start this doctoral thesis thanks to the experience gained working in the Department of Applied Mathematics at the University of Santiago de Compostela. I will be forever grateful to Pilar Salgado, María Dolores Gómez, and Alfredo Bermúdez for giving me the opportunity to be a researcher.

Thanks to Juan Ángel Lorenzo del Castillo for his warm welcome and support during my stay at the CY Tech.

Thanks to Bernhard Höfle and his team at the University of Heidelberg for the year we spent working together.

This work has received financial support from the Consellería de Cultura, Educación e Ordenación Universitaria (accreditation ED431G-2019/04, ED431C-2022/16) and the European Regional Development Fund (ERDF), which acknowledges the CiTIUS-Research Center in Intelligent Technologies of the University of Santiago de Compostela as a Research Center of the Galician University System. This work was also supported by the Ministry of Economy and Competitiveness, Government of Spain, under Grant PID2019-104834GB-I00 and Grant PID2022-141623NB-I00. Data services offered by the OpenTopography Facility were utilized and supported by the National Science Foundation under NSF Award 1948997, Award 1948994, and Award 1948857. We also acknowledge the *Centro de Supercomputación de Galicia* (CESGA) for providing computational resources and outstanding support. We also would like to thank Babcock Mission Critical Service SAU, currently AVINCIS TECHNICS Y AVINCIS ESPAÑA UTE, for providing data of Galicia, as part of the Civil UAVs Initiative.

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# Contents

**Resumo** **1**

**1 Introduction** **9**
1.1 Objectives . . . 14
1.2 Thesis outline . . . 14
1.3 Contributions . . . 15

**2 LiDAR technology** **17**
2.1 The physics behind a LiDAR sensor . . . 19
2.1.1 Distance measurement: Point coordinates . . . 19
2.1.2 Reflection of the laser beam: Intensity . . . 26
2.1.3 Laser beam divergence: Number of returns . . . 29
2.2 The engineering behind a LiDAR sensor . . . 31
2.2.1 Controlling the scanning direction . . . 31
2.2.2 Controlling the scanning translation . . . 33
2.2.3 From distances to coordinates . . . 34
2.2.4 Storing the data . . . 36

**3 High-performance point cloud algorithms and data structures** **43**
3.1 Neighborhood computation . . . 44
3.2 Point cloud data structures . . . 49
3.2.1 $k$-d tree . . . 50
3.2.2 Octree . . . 52
3.3 Performance evaluation . . . 52
3.3.1 Datasets . . . 53
3.3.2 Libraries . . . 55
3.3.3 Results . . . 56
3.4 High-performance computing for neighbor searching . . . 58
3.5 Conclusions . . . 62

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**4 Optimal pathfinder in 3D point clouds** **63**
4.1 Graph construction . . . 68
4.2 LCP computation . . . 72
4.2.1 Algorithms to obtain the optimal path . . . 77
4.2.2 Storing the graph in memory . . . 80
4.3 Examples and results . . . 80
4.3.1 Point cloud segmentation . . . 81
4.3.2 Graph construction . . . 83
4.3.3 Slopes . . . 86
4.3.4 Roughness . . . 87
4.3.5 NDVI . . . 88
4.4 Computational performance . . . 89
4.4.1 Results on Test A . . . 91
4.4.2 Results on Test B . . . 93
4.5 Discussion . . . 93
4.5.1 Number of points considered . . . 94
4.5.2 Trafficability of the path . . . 95
4.6 Conclusions . . . 97

**5 Powerline detection and characterization** **99**
5.1 Powerline detection algorithm . . . 102
5.1.1 Identifying vertically isolated points via a height-based filter . . . 103
5.1.2 Detecting straight lines: The Hough transform . . . 105
5.1.3 Powerline clusterization . . . 115
5.1.4 Powerline characterization . . . 120
5.1.5 Validation of the characterization procedure . . . 125
5.2 Results and experiments . . . 125
5.2.1 Results on the Diablo Canyon dataset . . . 126
5.2.2 Results on the Lugo dataset . . . 128
5.2.3 Results on the DALES dataset . . . 129
5.2.4 Quantitative evaluation . . . 131
5.2.5 Parameters sensitivity analysis . . . 138
5.2.6 Influence of the point density . . . 139
5.3 Computational performance . . . 140
5.4 Discussion . . . 144
5.5 Conclusions . . . 147

**6 Conclusions** **151**



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Bibliography** **155**

**List of Figures** **179**

**List of Tables** **185**

**List of Algorithms** **187**

**Acronyms** **189**

**Appendix A Machines** **193**

**Appendix B Kernel isoefficiency** **197**

**Appendix C Copyright and permissions** **201**
C.1 Articles . . . . . 201
C.2 Figures . . . . . 202



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# Resumo

Esta tese de doutoramento aborda o procesamento de nubes de puntos 3D con técnicas de computación de altas prestacións. Unha nube de puntos é unha representación virtual tridimensional dunha área determinada, e existen numerosas tecnoloxías para a súa obtención, todas englobadas no sensado remoto.

O sensado remoto é unha técnica que permite obter información sobre as características físicas dun obxecto sen ter contacto directo con el. As raíces do sensado remoto remóntanse a principios do s. XIX, cando a invención da cámara fixo posible capturar imaxes da Terra dende o aire. No ano 1858, o fotógrafo francés Gaspard-Félix Tournachon, realizou a primeira fotografía aérea dende un globo. A partir de entón, a tecnoloxía foi evolucionando, sobre todo ó longo do s. XX. A principios deste século, usáronse aeronaves tripuladas para capturar fotografías aéreas co propósito de ser utilizadas pola intelixencia militar na Primera Guerra Mundial. Foi na Segunda Guerra Mundial cando se ampliou o rango do espectro electromagnético aproveitable polas cámaras, incluindo dende as microondas ata o infravermello.

As diferentes tecnoloxías que permiten o sensado remoto poden dividirse en dous grandes grupos: por un lado, o sensado remoto activo, onde o sensor emite unha sinal e recolle a súa reflexión; e por outro, o sensado remoto pasivo, onde o sensor recolle a radiación emitida polo obxecto. A información recollida e posteriormente analizada, procesada, e pódese aplicar a numerosos campos, dende agricultura e xestión de recursos naturais ata planificación urbana e monitorización ambiental.

Dentro do sensado remoto activo, unha das técnicas mais empregadas é a tecnoloxía LiDAR, que permite obter nubes de puntos con un moi alto nivel de precisión. LiDAR é unha técnica de sensado remoto activa, que permite a medida de distancias entre calqueira obstáculo e o sensor. A técnica basease na emisión dun pulso de luz láser, que reflíctese nos obstáculos e retorna ao sensor. A medición do tempo que transcorre entre a emisión e a recepción do pulso permite calcular a distancia entre o sensor e o obstáculo. A primeira proposta de utilizar a luz para medir unha distancia foi feita polo físico ruso A. A. Lebedev en 1936, onde conseguiu medir un obstáculo a 3,5 km de distancia cun erro de 2 m.

Coñecendo a orientación do sensor e a súa posición no momento da emisión do laser, é posible calcular as coordenadas do rebote con precisión. Repetir este proceso centos

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

de miles de veces por segundo permite obter unha nube de puntos 3D da contorna cun gran nivel de detalle. As orixes dos sensores LiDAR remóntanse a mediados do século XX, coa invención do laser.

Para calcular a distancia á que se atopa un obstáculo, e necesario calcular o tempo que tarda o laser en viaxar dende o sensor ata o obxecto e voltar ao sensor. Para o cómputo do tempo, existen dous enfoques dependendo do tipo de feixe laser utilizado. No caso de utilizar pulsos discretos, a emisión dun pulso moi corto e intenso de laser activa un fotodíodo, que á súa vez activa un contador. Este contador volve a activarse cando o pulso de laser retorna ao sensor. A diferenza entre os dous valores do contador é proporcional á distancia entre o sensor e o obxecto. A frecuencia do oscilador depende da precisión que se desexa acadar na medición. Por exemplo, para obter unha resolución de 10 cm, necesitaríase un oscilador capaz de medir intervalos de 0,66 ns. A maior frecuencia, maior resolución do sensor. No caso de utilizar pulsos continuos, a medición baséase na interferencia entre o pulso emitido e o reflectido. A diferenza de fase entre os dous pulsos é proporcional á distancia entre o sensor e o obxecto. Para poder medir a diferenza de fase, débese modular a frecuencia ou a amplitude do laser emitido.

As nubes de puntos son unha fonte de información moi importante de cara á análise de todo tipo de escenarios: dende zonas rurais ou boscosas ata zonas urbanas. Os sensores LiDAR do estado da arte son capaces de tomar medicións con frecuencias de centos de miles de puntos por segundo. Así, a cantidade de datos final obtida é masiva, polo que o procesamento destes datos require dun grande esforzo computacional.

No Capítulo 2 explícanse os principais conceptos relacionados coa tecnoloxía LiDAR: a historia da súa creación a partir dos avances científicos de principios do s. XX, así como a física que hai detrás dos fenómenos involucrados na propagación da luz, a adquisición do rebote dun feixe laser e a medición to tempo de voo. Ademáis, explícanse os métodos principais utilizados nos sensores LiDAR para deflectar o laser e rexistrar a posición do sensor en todo momento, ambas necesarias para a obtención dunha nube de puntos xeorreferenciada. Finalmente, explícase cal é o estándar de almacenamento de datos LiDAR e por que sobresae con respecto a outras alternativas.

Á hora de procesar nubes de puntos, un dos principais recursos utilizados para obter información útil é a análise da veciñanza dos puntos, sendo esta o conxunto de puntos situados dentro dunha determinada zona xeométrica co respecto ó punto analizado. Polo tanto, a capacidade de obter estas veciñanzas de forma eficiente é unha tarefa fundamental no procesamento de nubes de puntos. Por iso, o uso dunha estrutura de datos adecuada é crucial. No Capítulo 3 abórdase tanto o estudo das estruturas de datos máis empregadas na actualidade como a súa implantación na linguaxe de programación elexida para o desenvolvemento dos algoritmos: C++. No procesamento de nubes de puntos, os lugares xeométricos ao redor dun punto que definen a súa veciñanza son típicamente cadrados ou círculos, no caso bidimensional, ou cubos e esferas, no caso tridimensional.

Nese mesmo capítulo, faise un estudo comparativo das librerías mais populares que



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

implementan as dúas estruturas de datos máis empregadas no procesamento de nubes de puntos 3D: o Octree e o $k$-d tree. O estudo, realizado utilizando diversos conxuntos de datos obtidos por distintos sensores e dende distintas plataformas, demostra que o $k$-d tree é a estrutura de datos mais axeitada á hora de buscar os $k$ veciños máis próximos, mentres que o Octree debe ser a estrutura de datos elixida cando a busca de veciños faise nunha xeometría definida. Con respecto á paralelización dos algoritmos, emprégase o paradigma de memoria compartida. Este paradigma de paralelismo permite que todas as unidades de procesamento teñan acceso a todo o conxunto de datos a procesar. Para obter unha bo escalabilidade dos algoritmos implementados, é crucial elexir unha estratexia de balanceo de carga adecuada. OpenMP, a API utilizada para paralelizar os algoritmos, permite varias estratexias de balanceo de carga. Para a busca de veciñanzas, sen dúbida o algoritmo máis empregado no procesamento de nubes de puntos, fixéronse probas coas principais estratexias para repartir a carga que proporciona OpenMP: estática, dinámica e guiada. O estudo conclúe que neste caso non existen diferenzas significativas entre elas. Ademais, realizouse un estudio da escalabilidade da búsqueda de veciñanzas para distintas xeometrías en función do seu tamaño. Os resultados demostran que a escalabilidade dos algoritmos implementados e moi competitiva cando os radios utilizados son inferiores a 4 m. Finalmente, explícase que a maioria de probas realizáronse en máquinas con diferentes nodos NUMA, polo que definir a afinidade dos fíos para que estean o mais próximos aos datos como sexa posible é crucial para obter o mellor rendemento posible das aplicacións desenroladas.

O Capítulo 4 é o primeiro dos dous adicados á presentación de solucións de procesamento de nubes de puntos 3D. Concretamente, preséntase un método para a determinación da ruta menos custosa entre dous puntos calesquera nunha nube de puntos LiDAR, tanto clasificada como sen clasificar. A busca dunha ruta entre dous puntos pode clasificarse como local ou global. Nas buscas locais, cada paso da ruta vaise calculando coa información disponible nese momento. Este tipo de técnicas pódense atopar habitualmente en navegación autónoma co fin de evitar obstáculos. Por outra banda, na busca de rutas global, todo o escenario está dispoñible dende un principio, e o obxectivo é atopar a ruta que minimice unha determinada función de custo. Habitualmente, a procura de rutas global apoiase na xeración dun modelo dixital do terreno. Isto é unha rasterización da escena, onde a cada cela se lle atribúe un valor de altura determinado. Así, tipicamente, a busca da ruta óptima entre dous puntos soamente ten en conta as diferenzas de altura, e por tanto as pendentes, do terreo, sen aplicar técnicas para evitar obstáculos ou terreo mal condicionado.

Ao longo desde capítulo, explícase o proceso de segmentación da nube de puntos utilizando algoritmos de clusterización baseados en densidade, o cal permitirá definir os nodos do grafo que eventualmente se utilizará para aplicar o algoritmo de obtención do camiño óptimo. Probáronse distintos parámetros para a realización da segmentación, e escolléronse os mais axeitados de maneira empírica. Para construir o grafo, desenvolveuse un método para determinar a topoloxía do mesmo, de tal maneira que os nodos queden conectados entre si de maneira axeitada por medio de aristas. Para



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

determinar as devanditas conexións utilizouse a teselación de Voronoi, a cal obtívose a partir da triangulación de Delaunay. A teselación de Voronoi relaciona nodos moi alonxados cando estos atópanse no borde das nubes de puntos, especialmente en xeometrías convexas. Por elo, implementouse unha solución para eliminar as aristas consideradas demasiado longas. A continuación, procédese a poboar cada arista dos grafos cos puntos LiDAR contidos nela, de tal maneira que para cada arista sexa posible calcular características xeométricas de maneira moi precisa. Así, utilizando os puntos contidos en cada arista, calcúlase o plano que mellor axusta a dita arista co algoritmo SVD. Este plano permite o cálculo da pendente tanto na dirección da arista como na dirección transversal. Ademais, a partir do plano, é posible calcular a rugosidade da superficie contida nela, o que posteriormente permitirá ao algoritmo de busca evitar áreas con presenza de obstáculos e superficies irregulares. De existir, a información sobre a presenza de estradas na nube de puntos é aproveitada de tal manera que o algoritmo de busca intente seguir a ruta marcada por estas. Por outra banda, se os datos LiDAR conteñen información da banda espectral infravermella, a información sobre a presenza de vexetación viva é incorporada ao algoritmo de busca, de tal maneira que se eviten as zonas con vexetación, especialmente os cultivos e plantacións.

No Capítulo 5 preséntase un método para a detección e caracterización de liñas eléctricas en nubes de puntos LiDAR. O método proposto é capaz de detectar liñas eléctricas de calqueira voltaxe, tanto en zonas rurais como urbanas, e é agnóstico co respecto á presenza de liñas eléctricas na nube de puntos. O método consiste en catro etapas ben diferenciadas. Primeiro, realízase un filtrado dos puntos baseados na súa altura. A continuación, aplícase a transformada de Hough sobre esos puntos para detectar liñas rectas na proxección no espacio bidimensional. As liñas detectadas procésanse posteriormente para determinar o seu número, no caso de que existan diferentes liñas apiladas no plano vertical. Finalmente, para cada liña individual, procédese ao seu modelado matemático utilizando a ecuación da catenaria para eixos arbitrarios.

O filtrado dos puntos consiste na busca de puntos que estén verticalmente illados, é dicir, que non teñan puntos a unha certa distancia por debaixo deles. Calculando a veciñanza para cada nube de punto, e computando a porcentaxe de puntos que cumplen certas condicións con respecto á altura e ao grosor estimado do cable, determínase se é un punto candidato a formar parte dunha liña eléctrica. Este proceso realízase de maneira iterativa para poder detectar aqueles casos onde existan conductores eléctricos distribuidos en múltiples alturas, o que é común no caso de liñas eléctricas de alta voltaxe.

A continuación, sobre os puntos que pasaron o anterior filtro, aplícase unha implementación da transformada de Hough adaptada para o seu funcionamento en nubes de puntos co obxecto de detectar conxuntos de puntos que estén aliñados. Unha parte importante da transformada de Hough é a busca de máximos locais no espacio de Hough. Para evitar múltiples detección dunha mesma línea, débense aplicar técnicas de supresión de máximos. As técnicas revisadas están baseadas en diferentes heurísti-



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

cas e requiren polo menos un parámetro. Dado que a elección do parámetro debe ser precisa para evitar a presenza de falsos positivos ou falsos negativos na detección de liñas, implementouse un método que permite facer esta supresión de manera exacta sen utilizar parámetros. Así, unha vez se detecta un máximo no espacio de Hough, as curvas xeradas por tódolos puntos pertencentes a ese máximo son eliminadas do espazo de Hough. Isto provoca a atenuación dos máximos locais arredor do máximo detectado de maneira exacta sen utilizar heurísticas, o que elimina a posibilidade de detectar falsos positivos ou falsos negativos.

Unha vez detectadas as liñas eléctricas, procédese á súa caracterización. Primeiramente, e dado que a transformada de Hough aplícase sobre o plano 2D, as líneas detectadas deben ser clusterizadas, xa que poden coexistir varias no mesmo plano vertical. Isto é algo que ocorre a miúdo en tendidos de alta tensión. A clusterización foi deseñada para traballar con nubes de puntos de moi pouca densidade, e con liñas eléctricas que presenten ausencia de datos nalgunha parte do seu percorrido. A clusterización está construída sobre o algoritmo RANSAC. Esto permite clusterizar os conductores eléctricos correctamente malia que sexan moi pouco densos ou existan tramos con ausencia de puntos. A clusterización realízase coa axuda do polinomio que mellor se axusta ás curvas, que será utilizado na última etapa do algoritmo. O dominio da liña detectada divídese en intervalos, e para os puntos mais baixos de cada intervalo calcúlase o polinomio que mellor axusta aos datos. A curva detectada é eliminada e o proceso repítese ata que non existan mais conductores que detectar.

Sobre os clusters, procédese a realizar a súa modelización matemática. Neste caso, a solución foi deseñada de tal maneira que non precise a aportación dun valor inicial para ningún dos parámetros que definen a curva de cada conductor. Se ben é sinxelo dar estimacións para os parámetros correspondentes á traslación dos eixos reducidos, a estimación da relación entre a tensión do conductor e a súa masa lineal non é trivial. Neste traballo, a partir da lonxitude do conductor, calculada a partir do polinomio xerado na etapa anterior, derívanse e resólvense numéricamente todas as ecuacións necesarias para o cómputo dos parámetros que definen a curva catenaria. O cómputo da lonxitude realízase a través de integración numérica, utilizando o método de Simpson. A continuación, o método de Brent e utilizado para resolver numéricamente a ecuación resultante do sistema de ecuacións plantexado para calcular os parámetros da catenaria. O método de Brent require dun intervalo inicial onde a solución será buscada. Dito intervalo é calculado de maneira unívoca heurísticamente, sen necesidade de proporcionar a estimación de ningún parámetro. Cos datos obtidos, é posible obter unha última ecuación que será resolta co método de Newton-Rhapson, o que finalmente proporciona os valores dos parámetros buscados para cada curva. Coñecendo a caracterización da curva, é posible calcular de maneira exacta a distancia de cada punto á curva matemática que define cada conductor.

O método de detección e caracterización de liñas eléctricas foi avaliado tanto cualitativamente como cuantitativamente utilizando tres nubes de puntos obtidas con LiDAR aerotransportado. A ausencia de nubes de puntos clasificadas é un mal endémico



USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

no campo do procesamento de nubes de puntos, polo que dúas das nubes utilizadas foron etiquetadas manualmente. A terceira nube de puntos corresponde ao conxunto de datos de DALES, a nube de referencia por excelencia para a clasificación de nubes de puntos LiDAR aéreas. A avaliación cualitativa amosa que o algoritmo é robusto na presenza de múltiples liñas eléctricas nunha misma escena, detectando todos os conductores eléctricos presentes nas escenas, independentemente de que estén agrupados nun mesmo plano vertical, ou se crucen entre eles. Ademais, os conductores múltiples que rotan o seu plano entre o inicio e o final do seu percorrido tamén son detectados correctamente. A transformada de Hough é capaz de eliminar os posibles falsos positivos detectados na primeira etapa do algoritmo. Estes puntos normalmente corresponden a copas de árbores ou ramas saíntes cunha altura considerable. Co respecto ao número de liñas eléctricas detectadas, non se detectaron falsos positivos nin falsos negativos a nivel de liña. A nivel de punto, a avaliación debe facerse de maneira cuantitativa. Para cada conxunto de datos, computáronse as métricas clásicas de avaliación de algoritmos de clasificación: sensibilidade, precision, exactitude, especificidade e puntuación F1. Con respecto á avaliación da modelización dos conductores eléctricos, usáronse como métricas a raíz cadrada do erro cuadrático medio e a media do erro absoluto. Cabe destacar que a comparación directa destas métricas con outros traballos non é posible, xa que a maioría dos traballos relacionados coa detección de liñas eléctricas non usan nubes de puntos adquiridas con plataformas aerotransportadas. A pesar disto, os resultados obtidos son moi competitivos con respecto a outros traballos que empregan nubes de puntos terrestres ou tomadas con traxectorias paralelas ás propias liñas eléctricas a analizar. Dos traballos analizados, o método proposto é o único que é capaz de detectar liñas eléctricas de calqueira voltaxe en nubes de densidades tan baixas como 3 puntos/m$^2$. De media, acadouse un 97,2 % de precisión, 87,0 % de sensibilidade e 91,8 % de $F_1$-score, cun RMSE medio de 0,14 m.

Tódolos algoritmos desenrolados están implementados para poder ser executados en paralelo en memoria compartida. Así, utilizouse a linguaxe C++ para a implementación e a API OpenMP para a súa paralelización. A paralelización dos algoritmos foi deseñada para ser escalable, polo que realizáronse probas de rendemento en diferentes máquinas para cada un dos algoritmos. Normalmente, a paralelización faise a nivel de punto, xa que adoita a ser no só doada de implementar, se non que é a que mellor escalabilidade otorga. Isto demóstrase cos estudos realizados para cada algoritmo, onde utilizouse o extinto supercomputador Finisterrae II e o actual Finisterrae III para analizar o comportamento do algoritmo frente ó número de fíos utilizado para distintos tamaños de conxuntos de datos. No caso do algoritmo de busca do camiño óptimo, un computador de escritorio ofrece unha escalabilidade notable, conseguindo eficiencias do 80 % no caso de utilizar todos os fíos dispoñibles. No Finisterrae II, a escalabilidade mantense para un número de fíos elevado, e ademais dase superlinealidade para tamaños de problema o suficientemente pequenos. No caso do algoritmo de detección de liñas eléctricas, onde utilizouse o Finisterrae III, a escalabilidade é tamén moi competitiva sempre e cando os datos non estean distribuidos entre distintos no-



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

dos NUMA. O fenómeno da superlinealidade tamén aparece na etapa de votación da transformada de Hough.



USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

## Chapter 1

# Introduction

Remote sensing involves gathering information about the physical characteristics of an object through non-contact methods, using techniques to detect and measure reflected (active remote sensing) or emitted energy (passive remote sensing). This information is then analyzed, processed, and applied to innumerable fields, ranging from agriculture and natural resource management to urban planning and environmental monitoring.

The roots of remote sensing can be traced back to the early 19th century when the invention of the camera made it possible to capture images of the Earth from above. In 1858, Gaspard-Félix Tournachon, a French photographer and balloonist, captured the first aerial photograph using a tethered balloon. Unfortunately, photography has not survived until the present. This marked the beginning of aerial photography, which later became an essential tool for remote sensing.

The development of remote sensing as a scientific discipline accelerated during the 20th century. In the early 1900s, manned aircraft were used to capture aerial photographs for military reconnaissance purposes during World War I. This led to advancements in aerial photography techniques and the development of specialized cameras. In the interwar period, the design of cameras was improved and adapted to their use in aircraft, which ultimately led to the first accurate measurements from photographs, known as photogrammetry. In this period, many companies were funded to develop civil applications of aerial mapping.

Until World War II, remote sensing was limited to the visible range of the electromagnetic spectrum despite the invisible spectrum being known several decades before. During WWII, the use of the spectrum extended to the infrared and microwave ranges for information purposes, such as the detection of enemy aircraft or the deployment of German V-2 rockets.

Another period in history where significant advancements in remote sensing were achieved was during the Cold War, where mutual aerial surveillance between the two superpowers of the time was a common activity. The development of jet-propelled aircraft, capable of reaching extremely high altitudes, forced the development of better

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

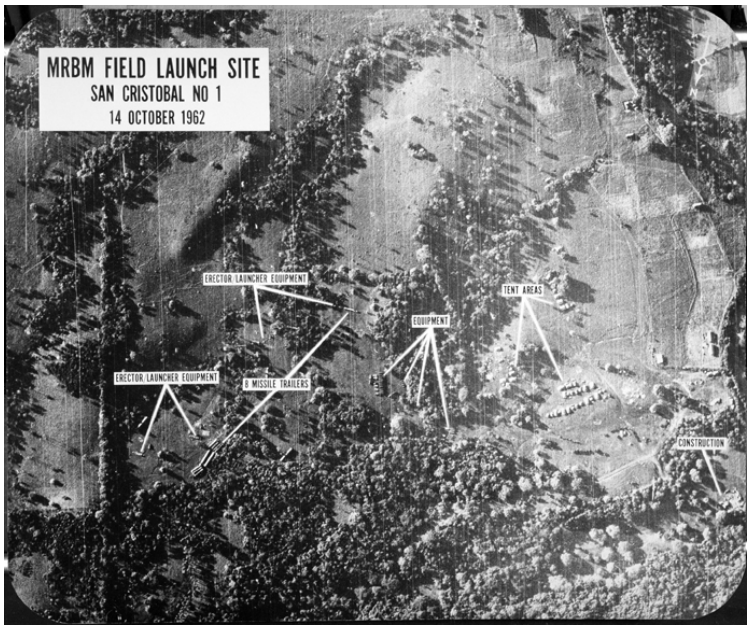



**Figure 1.1:** Aerial photograph of MRBM Field Launch Site No.1 in San Cristobal, Cuba, on 14/10/1962 [1]. Public domain.

sensors. Arguably, the best contribution of aerial remote sensing in the Cold War era was during the Cuban Missile Crisis in 1962, when the U.S. detected the early stages of a massive introduction of ballistic missile facilities in Cuba (Figure 1.1) with photographic means. This early detection, not expected by the Soviet bloc, allowed the incident to be defused peacefully.

In parallel to these events, the space race between the United States and the Soviet Union enabled the launch of the first Earth-observing satellites, initiating the era of space-based remote sensing. In 1959, the U.S. launched the first successful weather satellite, Explorer 6, which provided the first image of the Earth from space (see Figure 1.2). Subsequent missions, such as Landsat (beginning in 1972) and the series of NOAA satellites (starting in 1970), provided high-resolution images of the Earth's surface. Meanwhile, the Soviet Union deployed a range of satellites, including the Elektron series for meteorological observations, the Meteor series for weather monitoring, the Resurs series for Earth resource mapping, and the Cosmos series for various scientific and military purposes. These satellites greatly expanded our ability to monitor the Earth's surface and atmosphere and gather valuable data for meteorology, environmental monitoring, and resource management.

The development of remote sensing platforms ran in parallel with that of sensors,



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Figure 1.2:** First image of the Earth taken from outside the Earth itself [2]. Public domain.

which became more sophisticated and provided higher spatial and spectral resolutions. In addition to visible light imagery, sensors were developed to capture data in other parts of the electromagnetic spectrum, such as thermal infrared and microwave wavelengths. This expansion enabled scientists to gather information about various aspects of the Earth, including vegetation health, atmospheric conditions, land use and cover, oceanography, and more.

At the dawn of the 21st century, remote sensing has become an integral part of numerous scientific disciplines and applications. It plays a crucial role in environmental monitoring, disaster management, infrastructure preservation, urban planning, archaeology, agriculture, forestry, and natural resource management, among others. Furthermore, the availability of open data and the development of advanced processing techniques have expanded remote sensing applications. These advancements allow for automated analysis, change detection, and the extraction of valuable information from large volumes of data.

In the context of remote sensing, sensors can be classified as either active or passive, depending on how they acquire information about the Earth's surface. The distinction lies in how they interact with the target and the energy source used for sensing.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Passive sensors are the most common type used in remote sensing. They detect and record naturally occurring energy, such as sunlight or thermal radiation, reflected or emitted by the Earth's surface. These sensors do not emit their own energy but rely on the energy already present in the environment. The most familiar example of a passive sensor is a traditional camera, which captures visible light reflected by objects. Passive sensors can also be sensitive to other parts of the electromagnetic spectrum, including infrared, microwave, and radio waves.

Passive remote sensing provides valuable information about the physical properties of the Earth's surface, such as vegetation cover, land use, and surface temperature. It is widely used in applications such as land cover mapping, monitoring of urban growth, and vegetation health assessments. Satellites like the Landsat series and the Sentinel satellites employ passive sensors to acquire multi-spectral or hyperspectral data, enabling detailed analysis of Earth's features.

On the other hand, active sensors emit their own energy and measure the reflected or scattered signals to gather information about the Earth's surface. These sensors illuminate the target with pulses of energy, such as radio waves or laser beams, and measure the time it takes for the energy to return. By analyzing the characteristics of the returned energy, active sensors provide precise information about the target's properties.

Active remote sensing is particularly useful when passive sensors are limited, such as in cloud-covered areas or during nighttime. It also provides additional information, such as surface elevation and subsurface features. Radio detection and ranging (radar) is a commonly used active sensing technology that employs microwave energy to measure the target's distance, speed, and other properties. Light Detection And Ranging (LiDAR) is another active sensing technique that uses laser pulses to measure distances and create detailed three-dimensional maps of the Earth's surface.

Active sensors find applications in many fields, including topographic mapping, forest structure analysis, and climate modeling. They are used in satellites like NASA's Shuttle Radar Topography Mission (SRTM) and the Ice, Cloud, and land Elevation Satellite (ICESat), as well as in airborne and terrestrial systems for detailed mapping and monitoring purposes.

Remote sensing has a rich history that spans several centuries. It evolved from early aerial photography to space-based observations, providing valuable insights into the Earth's surface and atmosphere. The continuous advancements in sensor technology and data analysis techniques have revolutionized the field and made remote sensing an indispensable tool for understanding and managing our planet's resources.

Several remote sensing techniques provide valuable information about the environment. No technique is inherently better than others; they have different characteristics that make the collected information more suitable depending on the specific application. For example, taking two photographs of an object from slightly different positions allows one to compute the distance to the photographed object using the



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

parallax technique. While this is true, it involves a tedious computation process so that a better approach would be using a rangefinder.

The algorithms presented in this thesis work on n-dimensional data, which is defined as:

$$\mathcal{D} = \{p_1, p_2, \ldots, p_m\} \text{ where } p_i \in \mathbb{R}^{m \times n} \tag{1.1}$$

where $p_i$ is the $i$-th point, $m$ the number of points, and $n$ is the number of features of each point. Three of the $n$ features each point has describe its position in the 3-dimensional space forming the subset $\mathcal{P} \in \mathbb{R}^{m \times 3}$, which is commonly known as *point cloud*. A point cloud is a 3-dimensional representation of the environment, where each point represents a sample. Point clouds present several advantages over images. Firstly, they offer a direct representation of 3D spatial information, capturing the intricate geometry and shape of objects within a scene. Unlike images, point clouds remain unaffected by lighting variations and texture discrepancies, making them ideal for environments with challenging lighting conditions or limited surface details.

Some techniques that allow the collection of 3D point clouds are Multi-Beam Echo Sounding (MBES), tomography using Synthetic Aperture Radar (SAR), photogrammetry, and LiDAR.

Regarding the MBES, it implies the utilization of acoustic waves to compute different distances. The utilization of this type of sensor is suitable when mapping the seabed since the sound propagation in a liquid medium is superior to any other type of electromagnetic wave. Using sound to map the surface of a forest does not seem like a good idea.

The SAR technique uses microwaves to produce a 3D map of its surroundings. Typically, the SAR sensors are not limited by their line-of-sight, which means they can detect objects not directly visible by the sensors. Nevertheless, the resolution of the SAR is much lower than LiDAR's, which can provide 3D mappings with a very high degree of detail.

Another technique that allows the collection of 3D information is photogrammetry, which involves the production of 3D point clouds by collecting a set of different 2D images. Its main advantage is that visual information can be added easily to the resulting point cloud. On the other hand, areas with low contrast cannot be reconstructed properly. The main consequence of this is that the precision of the reconstruction is not uniform across the point cloud, and it could depend on whether the given area is well-illuminated or not.

Finally, LiDAR is the technique that allows the most precise 3D data collection, providing highly accurate reconstructions of the scanned areas. This makes the LiDAR data suitable to perform different geometrical analyses, such as the computation of the distance between different elements, the detection of irregularities in surfaces, and so on. Also, its ability to penetrate dense vegetation makes the LiDAR data suitable for performing forestry-related analysis.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

It is worth mentioning that the fusion of the data produced by different sensors is an open research field. These techniques allow all the valuable information different sensors provide to be available for processing. An example of data fusion applications is the combination of LiDAR data with hyperspectral images. While LiDAR data provides valuable information about the 3D distribution of an environment, the hyperspectral data allows the identification of different elements based on their spectral signatures. One possible application is the characterization of individual trees in a forest, using LiDAR data to model the tree in terms of its geometry and using the hyperspectral data to determine the tree species.

## 1.1 Objectives

The work presented in this Ph.D. aims to provide more complete solutions than those encountered in the state-of-the-art literature regarding two problems: powerline detection and optimal path computation in 3D point clouds.

- Powerline detection: Existing methods for powerline detection in 3D point clouds might have limitations, such as needing high-density point clouds or being able to process only one powerline at a time. Our research proposes a novel approach for powerline detection that addresses these shortcomings, being able to detect multiple powerlines in any scenario and process them no matter their configuration.
- Optimal path computation: Current techniques for optimal path computation in 3D point clouds depend on the rasterization of the point cloud to properly work, losing a significant amount of the information the point cloud provides. Typically, only the terrain slopes are considered to compute the optimal path between two points. Our work proposes using all the information available in the point cloud to find the optimal path.

Given the potentially large size of the point clouds, the algorithms were developed primarily focusing on optimizing performance. In particular, they were designed to leverage shared-memory architectures. This design choice enables the code to exploit the processing capabilities of multiprocessor computer systems that can access a unified memory space.

## 1.2 Thesis outline

The rest of this manuscript is organized as follows:

- Chapter 2 describes the principles behind LiDAR sensors and how the 3D point clouds are built from the acquisition stage to the final product.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

- Chapter 3 describes the typical data structures to store point clouds and the algorithms used to compute the local features of every point in a given point cloud.

- Chapter 4 describes the computation of the optimal path between two points in a 3D point cloud obtained with LiDAR technology, attending to different terrain characteristics.

- Chapter 5 describes the methods to detect and characterize different types of powerlines across different scenarios.

## 1.3 Contributions

- Articles directly related to the thesis:

  - Articles in peer-reviewed journals:

    * M. Yermo, F. F. Rivera, J. C. Cabaleiro, D. L. Vilariño, and T. F. Pena, "A fast and optimal pathfinder using airborne LiDAR data", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 183, pp. 482–495, 2022, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2021.11.014
      **Impact factor (JCR 2022):** 12.700, Q1.
      **Category:** GEOGRAPHY, PHYSICAL.
      **Rank:** 1/49.
      **Impact factor (SJR 2022):** 3.760, Q1.
      **Category:** COMPUTERS IN EARTH SCIENCES.
      **Rank:** 2/69.
      **Contributions:** Design, development, and optimization of the algorithms. Design the experiments, analyze the results, and write the article.

    * M. Yermo, R. Laso, O. G. Lorenzo, J. C. Cabaleiro, T. F. Pena, F. F. Rivera, and D. L. Vilariño, "Powerline Detection and Characterization in General-Purpose Airborne LiDAR Surveys", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024. DOI: https://doi.org/10.1109/JSTARS.2024.3396522
      **Impact factor (JCR 2023)**: 5.500, Q1.
      **Category:** GEOGRAPHY, PHYSICAL.
      **Rank:** 6/49.
      **Impact factor (SJR 2023)**: 1.434 Q1.
      **Category:** COMPUTERS IN EARTH SCIENCES.
      **Rank:** 5/69.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Contributions:** Design, development, and optimization of the algorithms. Design the experiments, analyze the results, and write the article.

- Articles published in international conferences:
  - M. Yermo, J. Martínez, O. G. Lorenzo, D. L. Vilariño, J. C. Cabaleiro, T. F. Pena, and F. F. Rivera, "Automatic detection and characterisation of power lines and their surroundings using LiDAR data", *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W13, pp. 1161–1168, 2019. DOI: 10.5194/isprs-archives-XLII-2-W13-1161-2019. [Online]. Available: https://isprs-archives.copernicus.org/articles/XLII-2-W13/1161/2019/
    **Contributions:** Design, development, and optimization of the algorithms. Design the experiments, analyze the results, and write the article.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

## Chapter 2

# LiDAR technology

*Any sufficiently advanced technology is indistinguishable from magic.*
— Arthur C. Clarke, *writer.*

As many other times in history, the theoretical work of one of the greatest physicists led to the development of new technology. In 1917, Albert Einstein published his work *On the quantum theory of radiation* [6], where the phenomena known as *stimulated emission* was presented. The *stimulated emission* consists of the interaction of a photon with an electron of an excited atom, causing the emission of an identical photon in terms of frequency, polarization, phase, and direction due to the decay of the excited electron to a lower energy level.

Suppose we have a medium containing a certain number of atoms, the vast majority of which are in an excited state. When a single photon interacts with one of those atoms, an identical photon is produced, which will continue to interact with other excited atoms, resulting in a cascade of identical photons sharing their phase and frequency with each other.

The knowledge provided by Einstein led Joseph Weber to be the first to introduce the concept of Microwave Amplification by Stimulated Emission of Radiation (maser) and Light Amplification by Stimulated Emission of Radiation (laser) in 1952. In 1954, a team of scientists consisting of Charles H. Townes, James P. Gordon, and Herbert J. Zeiger built the first maser during their time at Columbia University. Nikolay Basov and Alexander Prokhorov also built a prototype independently. Townes, Basov, and Prokhorov were awarded the Nobel Prize in Physics in 1964 for their invention. The maser was an innovative device designed to amplify microwave signals using the principle of stimulated emission.

In 1957, an American physicist, Gordon Gould, coined the term laser while working on his doctoral thesis and filed a patent application for the laser in 1959. Nevertheless, the construction of the first laser is credited to Theodore Harold Maiman, who developed and patented the first ruby-based laser in 1960 [7].

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Over the years, the practical applications of the laser have proven to be highly diverse. Among these applications are optic communication (optic fiber), medicine (surgical eye corrections, tattoo removal), optical data storage (CD, DVD, Bluray), and scientific research (gravitational waves discovery). However, as is common when a new technology is developed, the very first application was found by the military, using the laser as a rangefinder.

The first proposal of using light to measure distance is attributed to a Russian physicist, A. A. Lebedev. In 1936, the first attempt at measuring a distance using light pulses was tested. A range of 3.5 km was measured with 2-3 m accuracy. In the 1960s and 1970s, the first military laser systems were developed as rangefinders, proximity fuses, and weapon guides. It was in the mid-1990s that advancements in electronics and computing allowed the construction of commercially viable laser based systems to be used for topographic mapping systems [8], nowadays known as LiDAR.

LiDAR can be defined as a remote sensing technology that uses the detected rebounds of emitted laser beams to measure the distance the beam traveled from its emission to its encounter with an obstacle. LiDAR sensors can take only one direct measurement: the lapse of time between the emission and the return of an emitted laser beam. The distance from the emitter to the obstacle can then be derived from this measurement. Furthermore, to accurately compute the coordinates where the laser beam ends its brief journey before returning to the sensor, some additional components are needed: a Global Positioning System (GPS), to know with a high precision degree the geographical position of the sensor, and the Inertial Measurement Unit (IMU), a device that can keep track of the rotation angles with respect to a fixed reference frame in the case the sensor is moving.

Over the years, as sensors have become cheaper and smaller, access to this type of technology has been democratized, with very affordable commercial solutions available. Some of the most important LiDAR sensor vendors are RIEGL [9], Leica [10], Velodyne [11], Optech [12], and Livox [13]. The work of these and other companies towards the commercialization of LiDAR sensors has allowed the birth of a wide range of applications across various fields. Some key applications include autonomous vehicles, where LiDAR enables object detection, obstacle avoidance, and precise mapping for safe navigation [14]. In mapping and surveying, LiDAR is widely used for high-precision topographic mapping, land surveying, and creating detailed 3D models of landscapes, buildings, and infrastructure. In particular, one of the most widespread applications in the topography field is the creation of Digital Terrain Model (DTM) [15], a rasterized representation of the ground surface. Environmental monitoring benefits from LiDAR ability to assess forest canopy structure [16, 17], vegetation health [18], coastal erosion [19], and water quality [20]. LiDAR also plays a significant role in urban planning and smart cities, generating detailed city models, infrastructure monitoring [21], and implementing smart city initiatives. Archaeology and cultural heritage preservation benefit from LiDAR's ability to identify and map archaeological sites [22] and uncover hidden structures [23]. In forestry and agricul-



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ture, LiDAR assists in forest inventory [24], and precision agriculture [25]. LiDAR applications extend to earthquake and natural disaster management, where it aids in hazard assessment [26], landslide monitoring [26], and flood modeling [27]. Civil engineering and infrastructure inspection benefit from LiDAR ability to assess structural integrity, monitor construction projects, and conduct infrastructure inspections [28, 29].

## 2.1 The physics behind a LiDAR sensor

Each LiDAR point contains valuable information that contributes to a comprehensive understanding of the scanned area. The fundamental data stored in a LiDAR point includes coordinates, intensity, number of rebounds, scan direction, and GPS timestamp, among others. The coordinates provide the exact spatial location of the point within the 3D point cloud, enabling accurate mapping and spatial analysis. The intensity value reflects the strength of the laser reflection, providing insights into the object's surface properties or material composition at that location. The number of rebounds indicates the number of times the laser pulse bounces off objects before returning to the sensor, conveying depth information and aiding in creating detailed 3D models. Lastly, the scan direction indicates the orientation of the LiDAR sensor during data acquisition, which is crucial for aligning multiple scans and creating a complete representation of the scanned area. Together, these attributes demonstrate the intertwined relationship between the physics of the detection (coordinates, intensity, and number of rebounds) and the engineering considerations (scan direction, GPS timestamp, data storage, registration, and synchronization) in LiDAR data collection and analysis.

### 2.1.1 Distance measurement: Point coordinates

A LiDAR sensor can be thought of as a high-frequency laser rangefinder. Its only purpose is to measure, in an indirect way, the distance traveled for each emitted laser beam. Depending on the type of emitted pulses, the measurements are performed in different ways: using discrete pulses or using a continuous wave.

#### Discrete pulse

In this model, an intense and very short laser pulse is emitted, and its rebound is detected. The laser pulse emitted undergoes energy diversion through a beam splitter, directing a small fraction of its energy toward a photodiode. The photodiode's activation triggers a timing mechanism. Typically, the triggering process is achieved through straightforward thresholding techniques. The reflected pulse is directed towards a second photodiode, whose activation stops the timing, and the Time of Flight



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

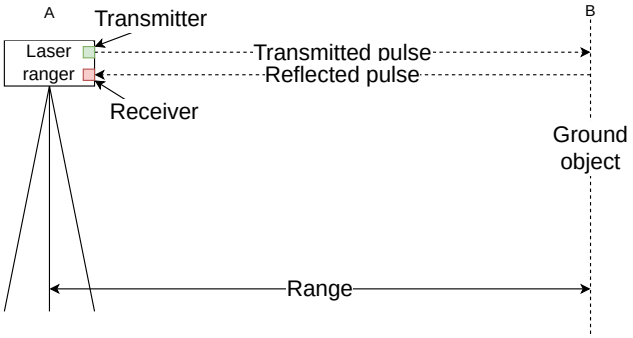



**Figure 2.1:** Computation of the TOF using the discrete pulse method.

(TOF) is measured by simple subtraction of the times between activations. In practical implementations, the timing mechanism is implemented as a high-speed counter governed by a highly stable oscillator. The needed frequency of the oscillator depends directly on the desired precision of the measurements. For example, in order to achieve a ranging resolution of 10 cm, the timer needs to be capable of accurately measuring intervals as small as 0.66 ns, which is the time needed for a pulse to perform the round trip to the obstacle and back to the sensor. This time corresponds to an oscillator's frequency of 1.49 GHz. The higher the frequencies achieved, the smaller the distances that can be measured. The basic operation of the discrete pulse method is illustrated in Figure 2.1.

When using the discrete pulse method to measure the distance to a given obstacle, the emitter usually has to wait until the last measurement time is computed to shoot the next laser pulse. Therefore, the smaller the distances expected to be measured, the higher the frequency of measurements can be. The number of measurements per second that can be made is known as Pulse Repetition Frequency (PRF). Typically, ground-based detectors have a PRF greater than airborne or spaceborne sensors. This deficiency can be mitigated by techniques that allow multiple flying pulses simultaneously [30], so a new pulse can be emitted before the last one is measured back in the sensor, notably increasing the data acquisition rate. The maximum distance a pulse can travel before the emission of the next is known as the *unambiguous range*. When an object is located beyond the unambiguous range, determining which pulse returns to the sensor becomes uncertain because multiple pulses are present in the air simultaneously. To expand the unambiguous range, several time intervals between pulses are used. As long as the sequence of time intervals remains unique and does not repeat, the most recent laser pulse that traveled from the sensor to the target and back can be accurately identified. This enables the range measurement without any



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ambiguity. However, there will still be a range ambiguity at a specific distance, which depends on when the pulse pattern repeats itself. The number of different time intervals between pulses determines the maximum number of pulses that can be present in the air simultaneously. If we initially had an unambiguous range of, 10 km with a fixed PRF, and a slight change in the time between pulses is introduced for ten consecutive pulses, the resulting unambiguous range would be approximately 100 km.

The traveled distance of the laser pulse is computed using the TOF as shown in the following equation:

$$L = \frac{c \cdot t}{2} \tag{2.1}$$

where $c$ is the speed of light, $t$ is the TOF, and $L$ is the distance the laser beam travels from the sensor to the encountered obstacle. Of course, no measure is perfect, so the computed distance will have an associated uncertainty, which can be computed by propagating the uncertainties [31] of the actual measurements as follows:

$$s(L) = \sqrt{\left(\frac{\partial L}{\partial c} \cdot s(c)\right)^2 + \left(\frac{\partial L}{\partial t} \cdot s(t)\right)^2} \approx \frac{c \cdot s(t)}{2} \tag{2.2}$$

where $s(L)$ is the uncertainty of the distance, $c$ is the speed of light, and $s(c)$ its uncertainty, which is considered negligible compared to the uncertainty of the TOF, $s(t)$. The assumption that the speed of light is accurately known and equal to its value in the vacuum should be noted. Actually, the speed of light in the air is slightly lower and varies based on factors such as air temperature, pressure, and humidity, but these effects are often negligible. With this in mind, it is reasonable to assume that the main direction of the noise in the data generated by a laser scanner will be longitudinal with respect to the laser beam axis.

### Continuous Wave (CW)

In this case, a continuous beam is emitted instead of pulses, and the TOF is computed using the phase shift method. In devices utilizing the CW method, the emitted beam is modulated by a signal layered onto it for measurement purposes. This modulation signal is generated and kept constant with the help of a stable frequency oscillator. Depending on whether the amplitude or the frequency is modulated, the processes are commonly referred to as Amplitude Modulation (AM) or Frequency Modulation (FM). The reflected beam preserves the modulation so it can be compared with the original signal, which allows the computation of the TOF and, therefore, the distance to the obstacle. In the following, we will briefly describe both modulation modes.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

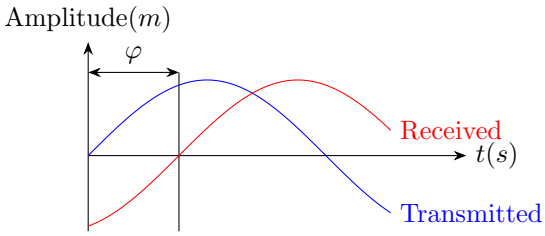



**Figure 2.2:** Measure of the phase difference between transmitted and received waves.

### CW: Amplitude Modulation

In the case of AM, the measurement is typically split into the integer number of wavelengths and its fractional part. Thus, the distance can be computed as follows:

$$L = \frac{M\lambda}{2} + \frac{\varphi}{2\pi} \cdot \frac{\lambda}{2} \tag{2.3}$$

where $M$ is the number of complete wavelengths, $\lambda$ is the wavelength itself, and $\varphi$ is the phase difference, as shown in Figure 2.2. In the first term of the equations, $M\lambda$ gives the total distance in multiples of $\lambda$, and it must be halved to take into account only the one-way trip of the beam. The second term corresponds to the distance the measured phase difference represents. Depending on the measurement conditions, the value of M may not be known, so the correct distance cannot be computed. The challenge of identifying the value of $M$ is often referred to as the *ambiguity resolution problem* [32, 33].

There are different methods to compute the distance unambiguously attending to the relation between $L$ and $\lambda$:

- $L \leq \frac{\lambda}{2}$: In this case, $M = 0$, so the distance can be computed directly from the phase difference:

$$L = \frac{\varphi}{2\pi} \cdot \frac{\lambda}{2} \tag{2.4}$$

  Assuming that the uncertainty of the wavelength is negligible compared to that of the phase difference, the distance uncertainty is governed by the following expression:

$$s(L) = \frac{\lambda s(\varphi)}{4\pi} \tag{2.5}$$

  It is clear that the main source of the distance uncertainty is the angle resolution $s(\varphi)$, which is constant no matter the wavelength used. For computing large distances, a very long wavelength must be used, so the uncertainty in the distance increases. For example, let us say that the phase difference can be measured with



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Table 2.1:** Example of using several discrete frequencies to determine a range uniquely. In bold, the significant digits that are added to the final distance measurement.

| $\lambda$ (m) | p (m) | L (m) |
|---|---|---|
| 10.00 | **2.76** | 2.76 |
| 100.0 | **4**2.8 | 42.76 |
| 1 000 | **3**43 | 342.76 |
| 10 000 | **6**340 | 6 342.76 |

a precision of 0.1°. In the case of a ranging distance of 1 000 m, the minimum needed wavelength is $\lambda = 2\,000$ m, which yields an uncertainty of $s(L) = 0.28$ cm, which leads to an inaccurate measurement. For this reason, the method is only used when mapping indoor areas or where the maximum range does not exceed 100 m.

- $L > \frac{\lambda}{2}$ In this case, the distance cannot be uniquely determined due to the ambiguity in the value of $M$. The ambiguity can be solved by modifying the frequency of the measuring wave. Two methods exist for uniquely determining the range depending on whether the frequency modifications are discrete or continuous.

  Discrete variations: The key to this method is understanding that the phase shift provides an ambiguous distance value that is always less than the wavelength used. By sending several waves, each with frequencies that are multiples of 10 relative to each other, the obtained distances can be combined to compute the actual range of the obstacle. For instance, using a wave with a wavelength of 10 m yields both the integer and decimal parts of the distance. The next digit is obtained by multiplying the wavelength by 10, and so on, until the desired range is achieved. One must consider that the measurement uncertainty increases as the wavelength increases. Continuous variations of the frequency can also be used to mitigate this issue. Table 2.1 shows an example of this procedure. This method requires prior knowledge about the range of distances to be measured to adjust the frequencies appropriately.

  Continuous variations: The reasoning behind this method is to compute the value of $M$ by solving a determined system of equations. The measurement starts with a given frequency. This frequency must be chosen so that the desired ranging accuracy can be reached given the phase shift accuracy. Once the phase shift is measured, the frequency of the measuring wave is slowly and continuously increased until the measurement of the phase shift matches the one caused by the first used frequency. At this point, we have



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

the following system of equations:

$$\begin{aligned} L &= \frac{M_1 \cdot \lambda_1}{2} + \frac{\varphi}{2\pi} \cdot \frac{\lambda_1}{2} \\ L &= \frac{M_2 \cdot \lambda_2}{2} + \frac{\varphi}{2\pi} \cdot \frac{\lambda_2}{2} \end{aligned} \tag{2.6}$$

where the values of $L$ and $M_i$ are unknown, whereas $\lambda_i$ is known. However, since the frequency has been increased to cause a phase shift of $2\pi$ rad, the relation $M_2 = M_1 + 1$ is fulfilled, which allows computing the distance unambiguously:

$$L = \frac{\lambda_1 \lambda_2}{2(\lambda_1 - \lambda_2)} + \frac{\varphi}{2\pi} \cdot \frac{\lambda_1}{2} \tag{2.7}$$

where $\lambda_1$ and $\lambda_2$ are the wavelengths producing the same phase shift.

In practical terms, the computation of the total number of integer wavelengths is not trivial. Specifically, the design of systems to accurately measure ranges by means of AM or FM is still open [34, 35].

### CW: Frequency Modulation

A method for computing the distance using FM is known as Frequency-Sweeping Interferometry (FSI) [36]. By analyzing the interference spectrum between the measuring wave and the returning carrier wave, the TOF can be calculated. The key of this method is the application of a well-known modulation linear function to the carrier wave, so its frequency increases linearly with time. This causes a chirped wave. Usually, the modulation function used is the *saw tooth* function for a double reason: its linear behavior allows it to compute the TOF easily, and its modulation is easy to implement. The periodic form of the *saw tooth* function is defined as follows:

$$u(t) = u_0 + \Delta u \ (ft - \lfloor ft \rfloor) \tag{2.8}$$

where $t$ is the time, $f$ the frequency of the modulation repetition, $\Delta u$ the optical frequency range and $u_0$ the baseline frequency. A full period of modulation, $T = \frac{1}{f}$, is known as a sweeping cycle. When a beam returns to the sensor, its frequency is the same as when it was emitted. Nevertheless, the frequency of the measuring wave continued to increase following the behavior given by the modulating signal so that both waves will have different frequencies. The absolute difference of these frequencies, $f_{\text{beat}} = |f_A - f_B|$, is known as *beat frequency*. Using this value and the known slope of the *saw tooth* function, the optical path length can be obtained:

$$\kappa = \frac{\Delta u}{T} = f \Delta u = \frac{f_{\text{beat}}}{\tau} \tag{2.9}$$



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

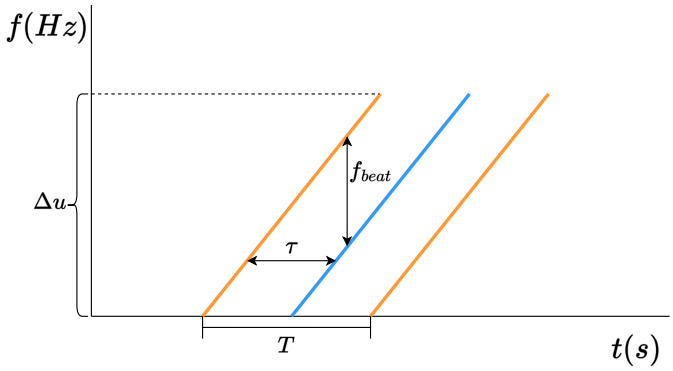



**Figure 2.3:** Beam frequency variation with the time. In orange is the frequency of the measuring signal. In blue, the extrapolated frequency of a returning wave after $\tau$ seconds of traveling.

where $\kappa$ is the slope of the *saw tooth* function and $\tau$ is the time between the emission and the reception of a modulated beam. A graphical representation of these values is depicted in Figure 2.3. Knowing that $\tau$ corresponds to the round trip, it is possible to compute the distance traveled to the obstacle as:

$$\tau = \frac{T f_B}{\Delta u};\ \tau = \frac{2L}{c}; L = \frac{cT f_{\text{beat}}}{2\Delta u} \tag{2.10}$$

where $L$ is the distance from the emitter to a given obstacle. At this point, only the computation of $f_{\text{beat}}$ remains to calculate the value of $L$ properly.

The computation of $f_{\text{beat}}$ is carried out through the spectral analysis of the interference signal formed from the measuring wave and the returning carrier wave. If the slope of the modulating function is low enough, the measuring signal and the carrier signal will have very close frequencies. By means of the Fourier transform, it is possible to compute the value of those frequencies, whose difference is precisely the value of $f_{\text{beat}}$, which, in the end, allows the computation of the distance traveled by the beam.

Figure 2.4 shows an example of the modulation function, the different signals, and the Fourier transform. In this example, the time shift between the measuring and carrier signals is $t = 1\,\mu\text{s}$. The carrier wave was emitted at $t_0 = 0.02\,\text{ms}$ at a frequency of $f_0 = 1.80\,\text{MHz}$. At the time of the return, the measuring wave's frequency was $f_r = 1.84\,\text{MHz}$. Both frequencies show a clear peak in the Fourier transform (Figure 2.4, bottom). The period of the modulating function was fixed to $T = 0.05\,\text{ms}$, and $\Delta u = 2\,\text{MHz}$. With this data, the range measurement is $L = 150\,\text{m}$, which validates Equation 2.10.

The practical implementations of the modulation functions are never perfectly linear but quasi-linear, causing a variation in the beating frequency. Consequently,



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

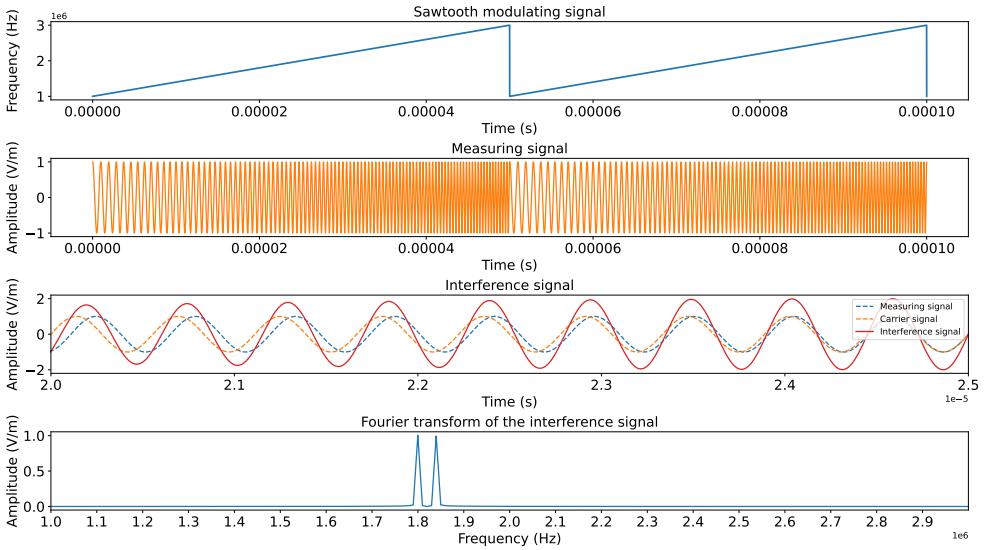



**Figure 2.4:** Extraction of the beat frequency using the measuring and the returning carrier signal by means of the Fourier transform in the generated interference.

the interference signal's spectrum becomes dispersed. Extensive research has been conducted to examine the impact of nonlinear tuning [37, 38]

The obtained uncertainty measurements are typically sufficient for LiDAR purposes. Despite the presence of additional sources of uncertainty, the CW method is generally regarded as more accurate than the pulse method because the phase difference can be computed with a higher precision than the TOF. Even so, the processing times needed by the CW methods are higher than those of the discrete pulse methods, discouraging its usage in high-frequency scanning applications. Also, attending to Equation 2.5, the uncertainty in the computed distance can be controlled by means of the employed wavelength to perform the measurement. Typically, determining the actual uncertainty is only available to manufacturing companies since the measurement uncertainties are directly related to the quality of the measuring hardware employed. These uncertainties are often offered in the sensor datasheet [39] provided by its vendor.

### 2.1.2 Reflection of the laser beam: Intensity

We studied how the rebound of the emitted laser pulse is collected by the sensor to carry out the distance computation of the obstacle. Therefore, discussing how the



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

pulses are reflected in the scanned surface is natural. Unless the scanned object is an ideal mirror, which could be an interesting case study, when a laser beam impacts a given surface, the condensed energy of the beam is nearly evenly distributed in all directions of the space. This behavior is known as Lambertian reflectance [40]. The laser beam is formed by a finite number of photons, and their distribution after the reflection follows Lambert's cosine law [41]:

$$I = I_0 \cos(\theta) \tag{2.11}$$

where $I_0$ is the emitted intensity, which could be thought as the number of photons of the laser beam, $\theta$ is the angle subtended by the normal of the scanned object and the laser beam, and $I$ the reflected intensity. Although Lambertian surfaces are ideal, they are rarely encountered in real cases, so the approximation is good enough for most applications.

The distribution of the reflected intensity could lead to thinking that the received intensity depends on the scanning angle, which is not true in all cases. When the scanned object area is equal or greater than the illuminated area, and it is located at a distance $L$ of the sensor, measured along the direction of the laser beam, the total reflected intensity in the unit of area is $I_R = I\omega A$, where $I$ is the reflected intensity computed using Equation 2.11, $\omega$ is the solid angle subtended by the illuminated area viewed from the measuring position (the sensor), and $A$ is the area of the illuminated surface. To compute the fraction of intensity entering the sensor, the area of the sensor's aperture ($A_0$) and its solid angle subtended from the illuminated area ($w_0$) must be considered. So, we can write:

$$I_r = \frac{I_0 \cos(\theta)\omega A}{w_0 A_0 \cos(\theta)} = \frac{I_0 A \omega}{A_0 w_0} \tag{2.12}$$

where $I_r$ is the intensity received by the sensor. Note that there is no dependence on the angle of the observation since the cosine terms of both Lambert's law and the subtended solid angle are canceled out. This allows the detection of reflections from practically any angle, as long as the sensor is sensitive enough.

Intuitively, since the scanned object is larger than the beam footprint, the whole surface contributes to the reflection of the beam. Nevertheless, when the object is smaller than the footprint, its cross-section must be considered in the computations. This cross-section is typically computed as $\sigma = 4\pi\rho r^2 \cos\theta$, where $\rho$ is the total hemispherical reflectance of the surface, and $r$ is its radius.

The distribution of the reflected energy follows the well-known inverse-squared law: as the wavefront expands, its area increases with the square of the distance from the source (see Figure 2.5). Consequently, the same amount of energy is spread over a larger area as the radius of the wavefront increases. This results in a decrease in energy density. Neglecting energy losses due to the heating of the illuminated surface



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

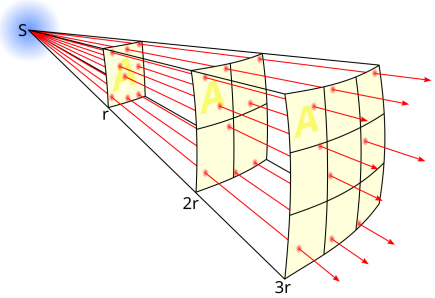



**Figure 2.5:** Depiction of the inverse-squared law [42]. Three snapshots of a wavefront are shown. CC BY-SA 3.0.

and atmospheric scattering and applying the energy conservation law, the following relation must be satisfied:

$$P_t = \int_0^{2\pi} \int_0^{\frac{\pi}{2}} P_r(\varphi, \theta)\ d\varphi d\theta \tag{2.13}$$

where $P_t$ is the power transmitted from the source, $P_r(\varphi, \theta)$ is the power measured in any direction of the space, $\theta$ is the polar angle, and $\varphi$ is the azimuthal angle. Note that the integration limits describe a perfect semi-sphere.

The measured energy of a rebound depends on several factors, whose relation is shown in the following equation:

$$P_r = P_t \frac{\sigma A_{\text{rec}}}{\pi L^2 A_{\text{illum}}} \eta_{\text{atm}} \eta_{\text{sys}} \tag{2.14}$$

where $A_{\text{rec}}$ is the area of the receiving sensor, $A_{\text{illum}}$ is the area illuminated, $L$ is the distance to the obstacle, $\sigma$ is the cross section of the area illuminated, $\eta_{\text{atm}}$ is the atmosphere's transmission efficiency, $\eta_{\text{sys}}$ is the receiver efficiency, and $P_t$ and $P_r$ are the transmitted and received power, respectively. The ratio $\sigma / A_{\text{illum}}$ corresponds to the percentage of the illuminated spot visible to the sensor. Multiplying that equation for the emission time of a given pulse and isolating $P_t$, the needed transmitted energy to receive a significant rebound can be computed as follows:

$$E_t = E_r \frac{\pi L^2 A_{\text{illum}}}{\sigma A_{\text{rec}} \eta_{\text{atm}} \eta_{\text{sys}}} \tag{2.15}$$

Of course, the amount of energy that must be transmitted to detect a rebound will depend directly on the detection methods and the sensor's sensitivity. The detected intensity in the receiver is stored along with other data taken in a given measurement.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

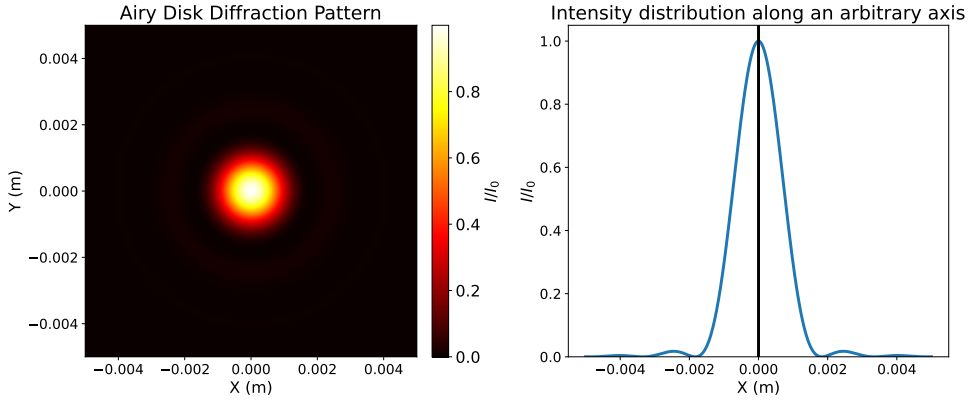



**Figure 2.6:** Intensity distribution through a circular aperture. Left: Airy disk diffraction pattern. Right: Intensity distribution along an arbitrary axis.

### 2.1.3 Laser beam divergence: Number of returns

The divergence of a beam refers to the spreading of the beam's energy distribution over distance, which causes an increase in the measurable diameter of the beam. Although the laser beam is well collimated after its emission from the sensor, a certain amount of beam divergence is unavoidable due to the diffraction of light. One of the main sources of this diffraction is the scattering of the photons that make up the beam, resulting from their interaction with atmospheric particles. There are primarily two types of scattering based on the size ratio between the wavelength and the particles with which the beam interacts. The scattering phenomenon for particles smaller than the wavelength follows the Rayleigh scattering [43]. When the particle size is comparable to the wavelength, the scattering behavior is governed by Mie scattering [44]. While Rayleigh scattering is isotropic in practical terms, Mie scattering refers to the dispersion of light around the direction of the beam. Typically, Rayleigh scattering is caused by molecules present in the atmosphere, while Mie scattering occurs due to the presence of fog, sand, or dust in the environment.

The other source of diffraction is the aperture of the emitter. When a collimated laser beam passes through a circular aperture, a particular diffraction pattern appears, known as the Airy pattern [45]. For a laser beam with $\lambda = 150$ nm, an aperture of 0.01 m, the diffraction pattern at 10 m of the aperture is shown in Figure 2.6. This pattern consists of a well-illuminated circular disk (the Airy disk) surrounded by concentric rings of decreasing intensity. The Airy pattern intensity distribution obeys the following equation [46]:



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

$$I(\theta) = I_0 \left( \frac{2J_1(kr\sin\theta)}{kr\sin\theta} \right)^2 \tag{2.16}$$

where $I_o$ is the intensity of the Airy disk's central spot, $k = \frac{2\pi}{\lambda}$ is the wavenumber, $r$ is the aperture radius, $\theta$ is the angle subtended between the center of the aperture and the observation point, which is the divergence itself, and $J_1$ is the Bessel function of first order. The intensity distribution is the same along any arbitrary axis, and it is shown in Figure 2.6. The first zero of the Bessel function corresponds to the first dark ring in the pattern, and $J_1(x) = 0$ when $x = 3.83$, which leads to $kr\sin\theta = 3.83$. From this expression, the divergence of the first dark disc can be derived:

$$\theta = 1.22\frac{\lambda}{2r} \tag{2.17}$$

where the approximation $\sin\theta \approx \theta$ was used, which is justified by the fact that, in the far field diffraction, $\theta \approx 0$. With the expression for the angular diffraction, it is possible to derive a good estimator for the expected diameter of the laser footprint at a distance $L$:

$$D = 2r + 1.22\frac{\lambda}{2r}L \tag{2.18}$$

where $2r$ is the diameter of the aperture, $\lambda$ the wavelength, and $L$ the distance traveled by the laser beam.

The main consequence of the beam divergence is the illumination of a circular (if the beam is perpendicular to the encountered obstacle) or elliptical area (otherwise). Thus, when only part of the diameter of the diverging pulse comes across an obstacle, that part of the pulse is reflected while the rest of the pulse keeps traveling until it encounters more obstacles. This effect can be exploited by LiDAR sensors to record multiple returns from a single pulse. The number of returns that can be recorded by a LiDAR sensor depends on the beam divergence and the distance to the encountered obstacle. The higher the divergence, the higher the number of returns that can be produced. For instance, the size of an illuminated spot produced at 1 km may vary from 30 cm to 1 m, depending on the emitter aperture and the wavelength used. It is up to the manufacturer to implement the ability to detect multiple returns. Common sampling methods in commercial LiDAR are, but not limited to, the following: recording of the most significant return, i.e., the return with the highest intensity; recording of the first return; recording of the last return; recording of all returns. The sampling method and the laser beam divergence are usually specified in the sensor datasheet [39].



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

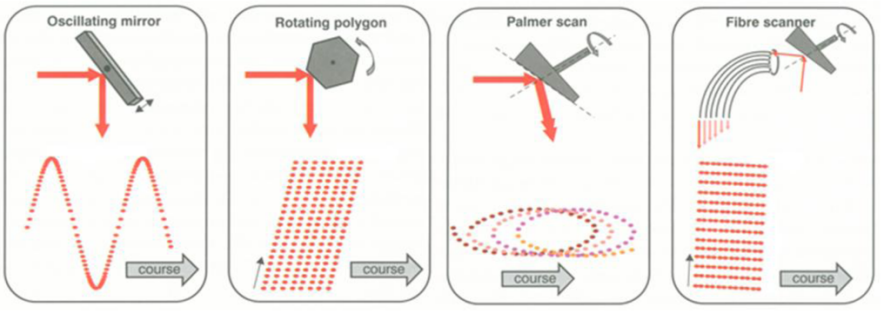



**Figure 2.7:** Mirrors commonly used to deflect the incoming laser beam and the corresponding scanning patterns produced [47]. CC BY 3.0.

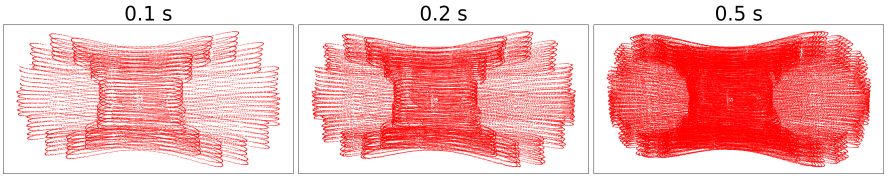



**Figure 2.8:** Scan pattern produced by the Livox Horizon [48] at 0.1 s, 0.2 s, and 0.5 s into the scanning process. This scanner can produce up to 480 000 points per second.

## 2.2 The engineering behind a LiDAR sensor

The last section exposed and discussed the physics behind measuring a single distance. Nevertheless, the goal of a LiDAR sensor is to collect enough data to represent its surroundings in a 3D point cloud precisely. This section presents the needed elements to go from 1D distance measurements to 3D point clouds.

### 2.2.1 Controlling the scanning direction

A LiDAR sensor by itself is only capable of measuring the distance to the nearest point aligned with the sensor's optics. In order to produce a set of measurements of different points, the trajectory of the output laser beam must be manipulated. This is often achieved by exploiting the reflection of the laser in a mobile mirror. These sensors are under the category of mechanical scanners. Usually, mirrors' movement is periodic, and the 2D profile acquired in one mirror oscillation is known as scanline [49]. The most common mirrors used and their associated scanning patterns are shown in Figure 2.7. The *oscillating mirror* is the least common of the reflecting devices. The changes in the inertia at the end of the oscillations cause these types of mirrors to be



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

small and light to allow a high scanning frequency. In contrast, *rotating polygon* and *Palmer scan* types do not suffer from the inertia problem, as they only need to reach the nominal rotational speed at the beginning of the scanning process and keep the angular velocity constant.

The pattern produced by the oscillating mirror has a sinusoidal shape, while the rotating mirror produces straight lines at a higher speed. On the other hand, the Palmer mirror produces a circular scanning pattern. It is possible to produce straight scanlines using a Palmer mirror through optic fiber, which is used to convert the rotational pattern into a linear one (*fibre scanner* in Figure 2.7). This solution is often used when a significantly reduced number of mobile parts is needed, usually when obtaining a high-speed scanning device is a design requirement.

It is worth mentioning that with the introduction of mirrors, the scanned object will not always be aligned with the receiving sensor; this must be considered when choosing the correct sensor size. The maximum scanning angle where the detection of the rebound is still possible is known as Field Of View (FOV). For each emitted pulse, the angle with respect to the sensor axis is recorded and stored along with other valuable information about the scanned point.

Livox [13] have developed sensors whose scanning patterns are unique and non-repetitive, so these sensors are able to scan a wide area located in its front without the need to move the sensors. These types of sensors are commonly encountered in autonomous driving applications [14], where continuous scanning is needed to get real-time 3D data of the vehicle's surroundings. The scanning pattern produced by these sensors is shown in Figure 2.8. Information about the configuration of inner mirrors was not found in this case.

LiDAR sensors without mobile parts are also available, known as non-mechanical sensors or solid-state LiDAR. To achieve high-performance scanning capabilities, these sensors utilize fixed optics and semiconductor-based components, including Micro-electromechanical Systems (MEMS), where tiny moving mirrors are replaced with micro-mirror arrays controlled by electric signals. Such sensors offer compactness, low power consumption, and high reliability, making them suitable for automotive collision avoidance and industrial automation applications. Additionally, Flash LiDAR systems utilize arrays of solid-state photodetectors to capture an entire scene instantaneously, eliminating the need for mechanical scanning. This approach enables rapid data acquisition and real-time 3D mapping, making Flash LiDAR sensors invaluable for urban planning, forestry, and disaster management applications. Finally, Photonics Integrated Circuit (PIC) sensors integrate multiple optical components onto a single chip, enabling miniaturization and cost reduction while maintaining high performance. These emerging solid-state LiDAR technologies promise to unlock new possibilities in remote sensing, enabling more efficient and versatile solutions for a wide range of industries and applications.

Finally, a new type of sensor exploiting the phenomenon of Brillouin scattering [50] was presented in [51]. Brillouin scattering refers to the interaction of light with the



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

material waves in a medium. It allows the refractive index of a medium to be altered through acoustic waves in the material. Thus, by modulating the frequency of the acoustic waves, the refractive index of the medium can be controlled, which, in turn, allows the laser beam to be deflected at any desired angle.

### 2.2.2 Controlling the scanning translation

By manipulating the scanning direction, the resulting patterns generated by the LiDAR system can take the form of a single straight line or a circle. To comprehensively map a specific environment, the LiDAR sensor needs to be repositioned rather than staying stationary. According to the type of platform used to move the scanner, the origin of the collected data is classified into some of the following groups, ordered by the sensor's height over the ground:

- Terrestrial Laser Scanning (TLS): The TLS platforms are typically tripods where the sensor is mounted, despite some authors including terrestrial vehicles in this category [52]. The tripod's position is fixed, but it provides rotational movement to the laser scanned to scan the surroundings. One of the main applications of this type of platform is forest inventory-related labors [53].

- Mobile Laser Scanning (MLS): In this configuration, the sensor is mounted in a terrestrial vehicle, and the vehicle's surroundings along its path are scanned. Usually, a rotary mount is used. The most common applications of these types of scanning are the characterization of urban elements, such as zebra crossing zones [54]. Other applications are road surface reconstruction, building facet modeling, generation of high-definition maps for their application in autonomous driving, and traffic sign detection. A deep insight into these applications is presented in [55].

- Unmanned Laser Scanning (ULS): The ULS refers to attaching the scanner to a Unmanned Aerial Vehicle (UAV). The utilization of LiDAR sensors on UAVs is gaining significant popularity across diverse applications such as agriculture [56], civil infrastructure [57], forestry inventory [58], and powerline inspections [28]. UAV platforms offer flexibility, granting accessibility to hard-to-reach areas, and enabling rapid data collection capabilities.

- Airborne Laser Scanning (ALS): The ALS is the most common type of LiDAR platform. In this case, the sensor is mounted in an airborne vehicle like an airplane or helicopter. ALS LiDAR finds extensive applications in diverse fields including river mapping [59], topographic mapping and land cover classification [60], forestry management [61], flood modeling [62], among many others. By offering intricate and accurate elevation data, it serves as a valuable instrument for comprehending and examining the Earth's surface on a broad scale.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

- Spaceborn Laser Scanning (SLS): SLS implies the utilization of LiDAR technology from satellite platforms for remote sensing and data collection purposes. In this context, LiDAR sensors are mounted on satellites orbiting Earth to capture information about the Earth's surface and its features. The most important platform in this category is the Ice, Cloud, and land Elevation Satellite 2 (ICESat-2), whose applications range from studying ice sheets and glaciers to monitoring vegetation and forests. An interesting application of SLS is the study of aerosols and clouds from space [63].

### 2.2.3 From distances to coordinates

The process of converting the measured distances to actual coordinates may vary depending on the type of platform used in the data acquisition process. This Ph.D. thesis is focused on processing data obtained through sensors mounted in ALS platforms, so the exposition of this topic will be based on that type of platform.

Along with the Laser Ranging Unit (LRU), which is used to measure the distance from the scanner to the illuminated spot, several devices must synchronously collect data to allow the accurate computation of the scanned spot coordinates from the measured distance. The Control, Monitoring, and Recording Unit (CMRU) is responsible for keeping control of the synchronization between the different devices. The device responsible for providing position and orientation data is the Positioning and Orientation System (POS), which, in turn, consists of a GPS and an IMU. With these devices, the position of the scanned spots can be accurately computed.

- The CMRU could be considered the heart of the data acquisition process. Its main function is to synchronize the activation of the laser emission with the rotation of the mirrors. Also, it is responsible for storing the collected data (the measured distance, the intensity of the collected pulse, the current scanning angle, the waveform, the return number, and the GPS timestamp of the measurement) in a storing device. In this case, the timestamps are needed for later synchronization with the POS unit, often carried out offline.

- The GPS is responsible for providing the aircraft's position with a very high degree of precision and generating precise timestamps for each measurement of the POS. It is common to use GPS ground stations at most 25 km from the aircraft to achieve centimeter-level accuracies in the position measurement. By knowing the exact position of the GPS ground station, the errors the GPS measurements might present can be appropriately corrected. This technique is known as Differential GPS (DGPS). The GPS signal must travel from the orbiting satellites to the corresponding station through the entire atmosphere, so the quality of the signal could be affected by the atmospheric conditions, which is the primary source of the measurement errors in the position. Finally,



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

the timestamps provided by the GPS are crucial to associate each measurement with the spatial situation of the platform, as well as the instant accelerations suffered due to vibrations.

- The IMU provides data about the platform's accelerations and orientations at a given time. This unit must be as close as possible to the scanner device to accurately take the measurements. The IMU's ability to measure accelerations is useful to cancel the effect that platform vibrations may have on the measurement. Typically, the IMUs record data at a frequency of 100 Hz. The IMU can also record the three orientation angles of the aircraft at any given moment. The information provided by the angles is essential for the correct computation of the actual coordinates of the measurements taken since the orientation of the local reference system is needed.

Finally, the information about the relative position of the LiDAR sensor with respect to the GPS and the IMU is needed, and it is up to the user to provide these parameters depending on the configuration used.

In the case that the POS is not built inside the LiDAR sensor, two files are stored in drives attached to the sensor. First, a LiDAR raw data file containing all the data related to the laser beam measurements collected through the survey. Second, a POS file where the instantaneous positions, accelerations, and rotations of the platform are stored, along with their GPS timestamps. The data is then synchronized between these two files. The main issue in synchronization is the difference between the measurement frequency of the GPS timestamps and the scan frequency of the LiDAR sensor. Typically, the measurement frequency of the POS is in the order of hundreds of Hertz, while the scanning frequency of the laser device is three orders of magnitude higher. Clearly, the relation between both types of data is not injective. To overcome this issue, the usual method is to record the timestamp at which the scan of a given scanline begins. By knowing the scanning frequency, the timestamps of the remaining points forming the scanline can be easily derived by linear interpolation.

After synchronization and registration, a raw LiDAR file is available with all the collected data. The fields included in this file are, in order: GPS time, spatial coordinates in WGS84, roll, pitch, yaw, distance, intensity, and scanning angle.

Note that the coordinates in this file correspond to the platform's position, not to the measurements themselves.

The process of combining all the acquired data and transforming them into actual 3D coordinates is called registration or georeferencing. Often, the coordinate system chosen to carry out the registration is the World Geodetic System 1984 (WGS84) [64]. The origin of this coordinate system is located at the Earth's mass center, with an uncertainty of 2 cm, and it allows to locate any point of the Earth, without using any references, using three coordinates $(x, y, z)$. These coordinates represent the vector from the Earth's center of mass until the represented point.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

The registration process is as follows. Let $\mathbf{r_p}$ be the vector representing the point whose distance was computed using a LiDAR sensor. At the moment of the measurement, the position vector $\mathbf{r_L}$ of the LiDAR sensor is known. Additionally, the vector $\mathbf{s}$ joining the LiDAR sensor with the illuminated spot can be computed offline using the distance measured and the orientation of the sensor. Thus, the registration process can be defined as the computation of the $\mathbf{r_p}$ vector:

$$\mathbf{r_p} = \mathbf{r_L} + \mathbf{s} \tag{2.19}$$

As the position of the plane is well-known thanks to the GPS data, the computation of $\mathbf{r_L}$ is straightforward since most GPS offer data in the WGS84 standard. Nevertheless, the computation of $\mathbf{s}$ requires transforming from the IMU reference frame to that of the sensor. Let be $\mathbf{\Lambda}_i^j$ the transformation matrix between an $i$-th reference frame with respect to the $j$-th. The georeferenced position of the illuminated target can be computed as:

$$\mathbf{r_p} = \mathbf{r_L} + \mathbf{\Lambda}_H^{\text{WGS84}} \cdot \mathbf{\Lambda}_{\text{IMU}}^H \cdot \mathbf{\Lambda}_L^{\text{IMU}} \cdot \mathbf{s} \tag{2.20}$$

where $\mathbf{\Lambda}_H^{\text{WGS84}}$ is the rotation matrix of the WGS84 reference frame with respect to local horizontal coordinate system, $\mathbf{\Lambda}_{\text{IMU}}^H$ is the same but for the IMU reference frame, and $\mathbf{\Lambda}_L^{\text{IMU}}$ is the matrix to transform from the IMU reference frame to that of the sensor. Usually, the IMU and the sensors are aligned, so $\mathbf{\Lambda}_L^{\text{IMU}} = \mathbf{I}$. The misalignment that may exist between these two devices is characterized in the calibration process and taken into account in the registration process.

### 2.2.4 Storing the data

Once the different data collected by all the devices has been merged and synchronized, it must be stored in files to facilitate its distribution and manipulation. There are plenty of file formats in which a point cloud could be stored, some of which are as follows:

- Polygon File Format (PLY): A flexible format that allows the addition of custom properties to each LiDAR point. Supports both ASCII and binary encoding. This format was originally thought to store 3D meshes.
- Wavefront OBJ (OBJ): Similar to the PLY format, it was primarily designed to store 3D polygonal models, but since it allows the storing of point cloud information without the need for connectivity information, it is feasible to use it to store 3D point clouds.
- Plain coordinates file (XYZ): A free ASCII format in which every row corresponds to a given measurement. It is up to the user to decide the format of each row, as well as to implement the needed I/O operations for its processing.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

- Point Cloud Data (PCD): A file format developed for use in the Point Cloud Library (PCL) [65]. The header stores certain properties of the point cloud, such as the number of entries or the information of each entry. The point cloud data are stored immediately after the header.

- FastBinary (FB): A file format used by the TerraSolid suite [66]. The format is not openly documented.

Among all the formats, there is one that stands out above the rest and has become the *de facto* standard in the LiDAR industry: the LASer (LAS) format. The format was developed by the American Society for Photogrammetry and Remote Sensing (ASPRS), and it encodes the point cloud data and metadata into a binary file, whose description and specification can be found in the ASPRS LAS Specification 1.4 - R14 [67]. The publication date of the latest standard is July 9, 2019. The key to the popularity of the LAS format is the efficient and flexible storage structure for large-scale point cloud data, which allows the implementation of fast and efficient compression algorithms, being the most popular the one created by Martin Isenburg [68]. The compression algorithm implemented by Isenburg is lossless, non-progressive, streaming, order-preserving, and provides random access. These features, along with an average compression ratio of 12.1, enable easy data exchange and compatibility between different software applications and LAS processing workflows.

The main structure of a LAS file is shown in Table 2.2. The file is divided into four blocks. The first one is the header block, which contains information about the contained data, such as the number of points, the date of the surveys, the program that generated the file, the bounds of the data, the scale, or the memory footprint of each entry. It also contains the offset, in bytes, to the location of the point data itself, as well as the number of Extended Variable Length Records (EVLR). The second block contains the Variable Length Records (VLR), which allows adding custom data to the file. Any number of VLRs is allowed, as long as the offset in bytes to the point data records can be stored in 4 bytes. The VLR is formed by a header block, in which the size of the actual record immediately after the header is stored, allowing the VLRs to be of any size. The complete description of a VLR record is shown in Table 2.3.

The third block is the point data block, beginning after the VLR block. Each point record must have exactly the same size. To fulfill the different needs the users may have, up to ten point data formats exist in the LAS 1.4 specification, depending on the available data for each point record. One of the most complete data record formats is shown in Table 2.4.

After the point records block, the EVLR begins. The EVLRs are practically identical to the VLRs, but a larger payload is allowed since these are located at the end of the file. This is reflected by the fact that the *Record Length After Header* field is 8 bytes long.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Table 2.2:** LAS format definition.

| Public Header Block |
|---|
| VLR |
| Point Data Records |
| EVLR |

**Table 2.3:** Variable Length Record Header

| **Item** | **Format** | **Size** | **Required** |
|---|---|---|---|
| Reserved | unsigned short | 2 bytes | - |
| User ID | char[16] | 16 bytes | yes |
| Record ID | unsigned short | 2 bytes | yes |
| Record Length After Header | unsigned short | 2 bytes | yes |
| Description | char[32] | 32 bytes | - |
| Actual record | - | - | - |

The description of the fields of a point record, using the format shown in Table 2.4, is as follows:

- **X, Y, and Z**: The coordinates of the points are stored as long integers to allow very high values without loss of precision. The reconstruction of the actual coordinates value is carried out when reading the LAS file using the offset and scale stored in the main header block as:

$$\begin{aligned} X_{\text{coordinate}} &= X_{\text{record}} \times X_{\text{scale}} + X_{\text{offset}} \\ Y_{\text{coordinate}} &= Y_{\text{record}} \times Y_{\text{scale}} + Y_{\text{offset}} \\ Z_{\text{coordinate}} &= Z_{\text{record}} \times Z_{\text{scale}} + Z_{\text{offset}} \end{aligned} \tag{2.21}$$

  The absolute values of the coordinates may be large, and it is the user's responsibility to choose the appropriate numeric type that avoids any possible overflow.

- **Intensity**: The integer representation of the pulse return magnitude. The value must always be normalized to a 16-bit unsigned value, which allows merging data from sensors with different intensity dynamic ranges.

- **Return number**: When a laser pulse has many returns due to its encounter with multiple obstacles, this number stores the order of the specific rebound.

- **Number of Returns**: For a given pulse, this number stores the number of total rebounds experimented by the laser pulse. Note that information regarding the return number and the number of returns are stored in the same byte.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Table 2.4:** Point Data Record Format 8

| Field | Format | Size | Required |
|---|---|---|---|
| X | long | 4 bytes | yes |
| Y | long | 4 bytes | yes |
| Z | long | 4 bytes | yes |
| Intensity | unsigned short | 2 bytes | no |
| Return Number | 4 bits (bits 0-3) | 4 bits | yes |
| Number of Returns | 4 bits (bits 4-7) | 4 bits | yes |
| Classification Flags | 4 bits (bits 0-3) | 4 bits | no |
| Scanner Channel | 2 bits (bits 4-5) | 2 bits | yes |
| Scan Direction Flag | 1 bit (bit 6) | 1 bit | yes |
| Edge of Flight Line | 1 bit (bit 7) | 1 bit | yes |
| Classification | unsigned char | 1 byte | yes |
| User Data | unsigned char | 1 byte | no |
| Scan Angle | short | 2 bytes | yes |
| Point Source ID | unsigned short | 2 bytes | yes |
| GPS Time | double | 8 bytes | yes |
| Red | unsigned short | 2 bytes | yes |
| Green | unsigned short | 2 bytes | yes |
| Blue | unsigned short | 2 bytes | yes |
| NIR | unsigned short | 2 bytes | yes |

- **Classification flags**: These 4 bits store special characteristics associated with the point record.
  - Bit 0 (Synthetic) If set, this point was created artificially.
  - Bit 1 (Key-Point) If set, thinning algorithms cannot remove this point, which usually removes a fraction of the points.
  - Bit 2 (Withheld) If set, this point should not be included in processing.
  - Bit 3 (Overlap) If set, this point is within the overlap region between flight strips.
- **Scanner Channel**: In the case of using multi-channel sensors, it encodes the ID of the scanner head that records the measurement using two bits.
- **Scan Direction Flag**: It is used to store the direction of the scanner at the time of the record. If the bit is set, it indicates that the scanner was moving from left to right and vice versa.
- **Edge of Flight Line**: This bit indicates if the current record is the end of a given scanline.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

- **Classification**: This byte encodes the key of a lookup table with the format shown in Table 2.5, which allows the association of a point record with a real-world element.
- **User Data**: This field is used at the user's discretion.
- **Point Source ID**: This value represents the source of origin for a given point. A source refers to a collection of data that exhibits temporal consistency, such as a flight line or sortie number for airborne systems.
- **GPS Time**: the timestamp at which the point record was taken. It could be stored in two formats: GPS Week Time or Adjusted Standard GPS Time. The information on the format used is stored in the file header.
- **RGB**: The Red, Green, and Blue image channel values associated with the record. These values are usually obtained from an imaging sensor.
- **NIR**: The near-infrared image channel values associated with the record.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Table 2.5:** ASPRS Standard Point Classes

| **Value (Bits 0:4)** | **Meaning** |
|---|---|
| 0 | Created, Never Classified |
| 1 | Unclassified (Unknown) |
| 2 | Ground |
| 3 | Low Vegetation |
| 4 | Medium Vegetation |
| 5 | High Vegetation |
| 6 | Building |
| 7 | Low Point (Noise) |
| 8 | *Reserved* |
| 9 | Water |
| 10 | Rail |
| 11 | Road Surface |
| 12 | *Reserved* |
| 13 | Wire - Guard (Shield) |
| 14 | Wire - Conductor (Phase) |
| 15 | Transmission Tower |
| 16 | Wire - Structure Connector |
| 17 | Bridge Deck |
| 18 | High Noise |
| 19 | Overhead Structure |
| 20 | Ignored Ground |
| 21 | Snow |
| 22 | Temporal Exclusion |
| 23-63 | *Reserved* |
| 64-225 | User Definable |



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# Chapter 3

# High-performance point cloud algorithms and data structures

*When I go to bed in the evening, I think about cache locality and data structures.*
— Michal "kovarex" Kovařík, *Factorio technology director.*

The efficient extraction of meaningful information from point clouds is crucial for object recognition, scene understanding, and environmental modeling tasks. One of the fundamental operations in point cloud processing is the search for neighboring points around a specific point. This operation serves as the basis for tasks like segmentation, feature extraction, data mining, convolutions, and clusterization.

To efficiently handle point clouds, their irregular structure must be taken into account to be able to design and implement efficient algorithms to process them. Contrary to the images, where the data is stored in a regular grid, and applying algorithms involving the surrounding pixels is straightforward, the irregular and 3D nature of point clouds makes the efficient implementation of algorithms more complex. Furthermore, images only have neighbor pixels in 2D, while the point clouds have an extra spatial dimension.

The features a point has can be classified according to its source: one set of features is derived from the scanning process itself, such as geometrical coordinates, the intensity, the scan angle, the GPS time, etc. The other features are obtained by processing the point cloud point by point. For example, the number of neighbors a point has, different geometrical descriptors, etc. Many geometrical 3D features derived from the eigenvalues of a point exist. These features describe the local geometry of the point, and they are used in many algorithms of point cloud processing. Some of these features are linearity, planarity, scattering, omnivariance, anisotropy, eigenentropy, and curvature [69]. One important feature that can also be derived from the eigenvalues is the normal of the best-fitting plane of a given point.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Typical techniques to compute the eigenvalues and eigenvectors of a point are Principal Component Analysis (PCA) and Singular Value Decomposition (SVD). Both require the computation of the covariance matrix of the analyzed point, which in turn needs the information of the neighboring points.

The great diversity of point cloud acquisition methods (ALS, MLS, TLS, etc.) has led to the creation of a wide variety of point cloud types in terms of density and distribution. Depending on these characteristics, not all methods for neighborhood analysis of a given point and not all data structures used to store the point cloud will be equally suitable.

In this chapter, a comprehensive explanation of what a neighborhood is and a comparative analysis of various data structures combined with neighboring search methods across different types of point clouds are presented. The goal is to provide insights into the strengths, weaknesses, and applicability of these methods in diverse scenarios. By understanding the performance of neighboring search techniques under various conditions, researchers and practitioners can make informed decisions when selecting an appropriate method for their specific point cloud processing tasks. The motivation behind the research stems from the observation that state-of-the-art data structures and neighboring search methods may exhibit varying levels of efficiency depending on the nature of the point cloud data. Factors such as varying point densities, irregular distributions, high dimensionalities, and the presence of noise lead to a deep evaluation of existing methods.

## 3.1 Neighborhood computation

Computing neighborhoods in a point cloud is a crucial task due to the irregular nature of the data itself. The neighborhood of a given data point is composed of the set of surrounding points that are *close* to the analyzed point, and it provides information about the local structure of the point cloud, which is essential in its processing [70]. What *close* means depends on the application and the algorithm. The most common neighboring methods encountered in the state-of-the-art literature are two: Fixed-radius neighborhood [71] and $K$-Nearest Neighbors (KNN) [72, 73]. While the former queries the point cloud to retrieve all the points in a surrounding defined through kernel function, the latter computes the $k$-nearest points using any valid metric, being $k$ the only parameter to be set.

Regarding the fixed-radius neighborhood of a point $p_i$, its neighborhood is defined as follows:

$$\mathcal{N}(p_i) = \{p_j \in \mathcal{P} \wedge p_j \in \mathcal{K}(p_j, p_i, r)\} \tag{3.1}$$

where $p_i$ is the analyzed point, $p_j$ is any other point in the point cloud, $\mathcal{K}(p_j, p_i, r)$ is the set defined by a kernel function that defines the neighborhood of a point, and $r$ is



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

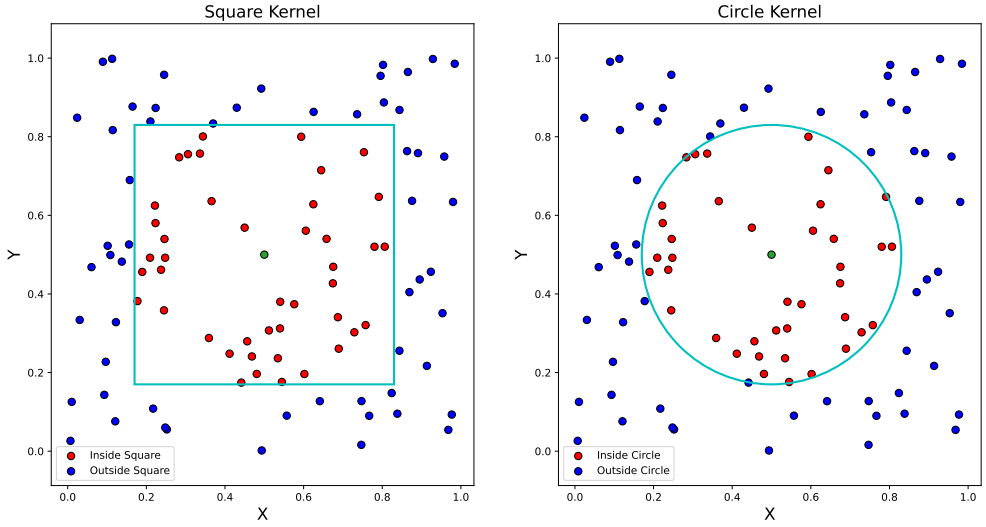



**Figure 3.1:** 2D kernels. The left image shows the square kernel and the right image shows the circular kernel.

a parameter that controls the kernel size. The mathematical definition of the kernel depends on the type of kernel used.

The typical approach is to use a spherical kernel composed of all the points lying inside a sphere of radius $R$ centered in the queried point. A variation of this method is to not consider the third spatial dimension (typically the vertical, $z$), producing a cylindrical neighborhood, often used in point clouds obtained in ALS [74]. While these two kernels are the most common, any custom geometry can be defined and used to carry out the queries: cubes and their 2D counterpart squares, toroids, ellipses, and many more.

The 2D kernels are defined only using two of the three spatial dimensions of the points, typically the $x$ and $y$ coordinates in ALS point clouds. The square and the circular kernels are the most common, and they are defined as follows:

$$\mathcal{K}_{\text{square}}(p_i, p_j, R) = \{p_j \in \mathcal{P} : |p_{i\lambda} - p_{j\lambda}| \leq R \; \forall \lambda = \{x, y\}\} \tag{3.2}$$

$$\mathcal{K}_{\text{circle}}(p_i, p_j, R) = \{p_j \in \mathcal{P} : (p_{ix} - p_{jx})^2 + (p_{iy} - p_{jy})^2 \leq R^2\} \tag{3.3}$$

The resulting neighborhoods using the 2D kernels are shown in Figure 3.1.

Regarding the typical 3D kernels, the cube and the sphere, their definitions are as follows:



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

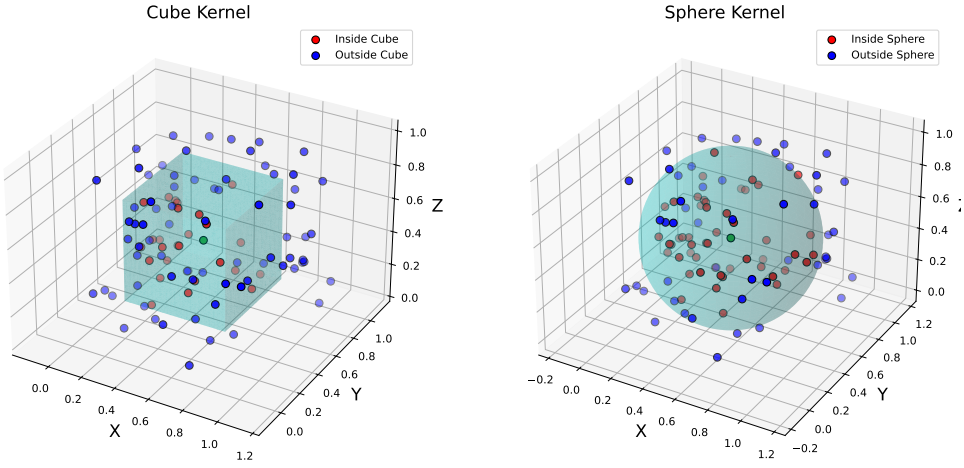



**Figure 3.2:** 3D kernels. The left image shows the cube kernel and the right image shows the spherical kernel.

$$\mathcal{K}_{\text{cube}}(p_i, p_j, R) = \{p_j \in \mathcal{P} : |p_{i\lambda} - p_{j\lambda}| \leq R \ \forall \lambda = \{x, y, z\}\} \tag{3.4}$$

$$\mathcal{K}_{\text{sphere}}(p_i, p_j, R) = \{p_j \in \mathcal{P} : (p_{ix} - p_{jx})^2 + (p_{iy} - p_{jy})^2 + (p_{iz} - p_{jz})^2 \leq R^2\} \tag{3.5}$$

The resulting neighborhoods using the 3D kernels are shown in Figure 3.2.

Despite these being the most commonly used kernels, any geometry can be defined and used to perform the queries. For example, the torus kernel and the ring kernel can be defined as follows:

$$\mathcal{K}_{\text{torus}}(p_i, p_j, R, r) = \{p_j \in \mathcal{P} : (\sqrt{p_{ix}^2 + p_{jy}^2} - R^2)^2 + p_{jz}^2 \leq r^2\} \tag{3.6}$$

$$\mathcal{K}_{\text{ring}}(p_i, p_j, r_1, r_2) = \{p_j \in \mathcal{P} : p_j \in \mathcal{K}_{\text{circle}}(p_i, p_j, r_2),\ p_j \notin \mathcal{K}_{\text{circle}}(p_i, p_j, r_1)\} \tag{3.7}$$

where $R$ is the major radius of the torus and $r$ the minor radius.

Examples of these kernels are shown in Figure 3.3.

As for KNN, the neighborhood $\mathcal{N}$ of a point $p_i$ is defined as follows:

$$\mathcal{N}_k(p_i) = \{p_j \in \mathcal{P} : ||p_i - p_j|| \leq ||p_i - p_l|| \ \forall p_l \in \mathcal{P} \setminus \{p_j\}\} \tag{3.8}$$

where $\mathcal{N}_k(p_i)$ is the neighborhood of the point $p_i$, $p_j$ and $p_l$ are any pair of points, $\mathcal{P}$ is the point cloud, $||\cdot||$ is any norm, and $k$ is the number of neighbors to be retrieved.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

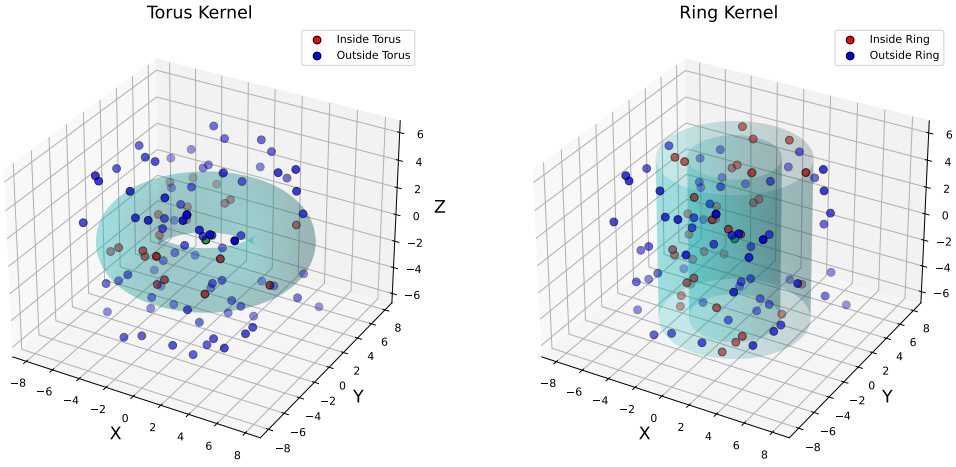



**Figure 3.3:** Torus and ring kernels.

An example of this kind of neighborhood is shown in Figure 3.4

If the point being queried is located in a low-density area, the $k$ closest neighbors may include points too far away to be geometrically meaningful. This could be detrimental to the quality of the local descriptors computed on the point, such as linearity, planarity, or eigenentropy. To overcome this issue, some authors use a mixed approach. For example, Want et al. [75] employs a fixed-radius sphere as a kernel, but only $k$ randomly selected neighbors are included.

Which type of neighborhood to use depends on the application. When processing point clouds, the neighborhood computation is usually the most computationally expensive operation [76], so choosing the correct approach is crucial in terms of accuracy and saving computing time. Both types of search methods depend on a parameter, which must be chosen carefully. Due to the irregular nature of point clouds, a fixed-radius search can be appropriate for some areas of the cloud but may fail in others. This occurs because of the variations of point densities across a single real dataset. To avoid this problem, some authors suggest the use of different scales across the dataset to retrieve the neighborhood. For example, [69] presented a strategy to adapt the query parameter for each point. Nevertheless, an improvement of this technique is proposed in [77, 78] for $k$-NN queries, and in [79, 80] for fixed-radius neighborhoods. Notwithstanding, the computational cost of all these methods is still significant.

To reduce query times, some authors rely on stochastic sampling methods, such as the Poisson Disk Sampling [81], which produces evenly distributed sets of points in a specific region. This is the case of [82], where the random sampling is also accelerated



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

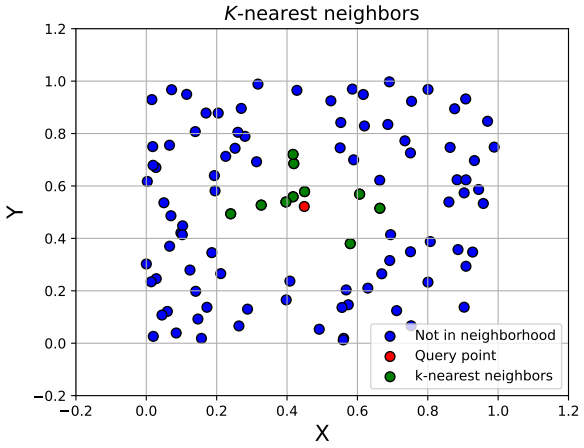



**Figure 3.4:** Example of a $k$-NN neighborhood for $k = 10$.

by means of GPU support to keep a high number of neighbors while reducing the computation times. To compute multiscale neighborhoods in reasonable times, [76] opted to perform a subsampling of the dataset by using a grid, which allows them to control the size of the neighborhoods. In the realm of real-time applications, [83] used a GPU-accelerated algorithm to detect obstacles for mobile vehicles. The hardware accelerated methods are also useful to process massive point clouds, as is the case of [84], where GPUs are used to accelerate the computation of triangular irregular networks.

It is worth mentioning the method used in [49] to compute the neighborhood of a given point. The authors exploit the geometry of the sensor's scanlines to transform the point cloud into a convenient space that assigns to each point two indexes: one corresponding to the position of the point inside a scanline and the other corresponding to the scanline the point belongs to. By considering the characteristics of the sensor in terms of distances between points, distances between scanlines, and the scan angle of each point, the spherical neighborhood can be retrieved in $\mathcal{O}1$. Nevertheless, this method is only valid when the sensor is known, and its scanning pattern is regular and predictable.

The naïve approach to computing the aforementioned types of neighborhoods is to check, for every point in the dataset, whether they meet the needed conditions to be part of the neighborhood. This method could be computationally suitable if the datasets are composed of a few hundred points (see Figure 3.5). From that number of points, the time difference between methods grows to reach two and a half orders of magnitude for a dataset of 1 000 000 points. State-of-the-art datasets range from tens



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

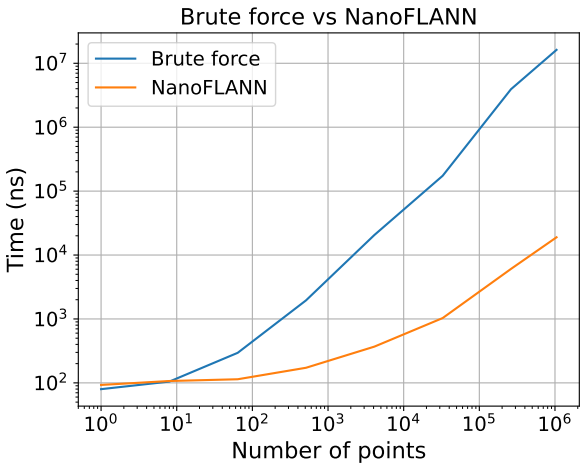



**Figure 3.5:** Comparison of query times using a specialized library versus the brute force approach.

to thousands of millions of points, so the brute force approach is not suitable, and specialized data structures to efficiently perform neighborhood searches are needed.

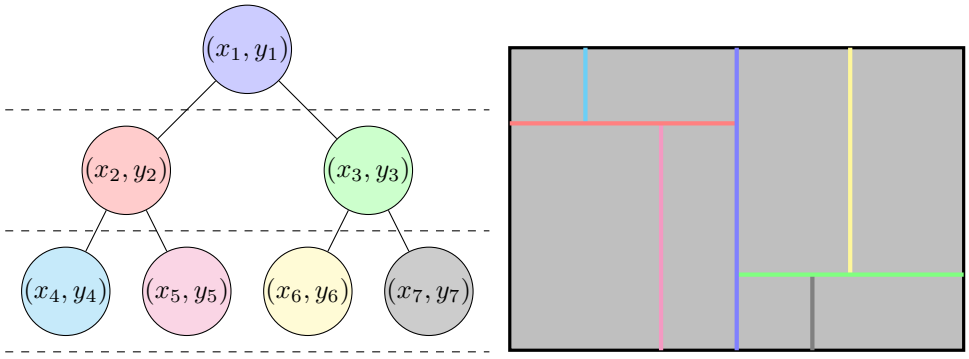
## 3.2 Point cloud data structures

A number of data structures were proposed to speed up the query times. The $k$-d tree [85] is a tree-based data structure in which each node has a maximum of two children. In the $k$-d tree, the data domain is partitioned by using ($k$-1)-dimensional planes, and each division created by each plane corresponds to a node of the tree. The Octree [86] is also a tree-based data structure, but each node has exactly eight children nodes. Other not-so-common data structures are the Ball Tree [87], which can be thought of as exactly the same as a $k$-d tree, but using ($k$-1)-dimensional spheres instead of planes. The R-Tree [88] is similar to the Octree, but the space partitioning is carried out by computing smaller bounding boxes for each level of the tree. To check whether two elements are nearby, the collision of the bounding boxes must be computed, which is usually a computationally cheap operation.

The most commonly used of the aforementioned data structures to carry out neighboring searches are the $k$-d tree and the Octree. $k$-d tree is known for its effectiveness in multidimensional data for nearest neighbor searches [49], whereas Octree is widely utilized for organizing point clouds into voxels, simplifying the exploitation of the Octree structure for neighbor searching [89]. Octree is also used in hybrid spatial indexing structures for massive point clouds, enhancing the efficiency of spatial indexing [90].



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA




(a) $k$-d tree with 3 levels. (b) 2D domain of the $k$-d tree.

**Figure 3.6:** Example of a $k$-d tree partitioning a 2D dataset. The data is recursively divided between nodes that are split using a hyperplane. The hyperplanes are orthogonal to each other.

In terms of efficiency, $k$-d tree is known for its complexity of $\mathcal{O}(n \log(n))$ for the nearest-neighbor search in point clouds with $n$ points [27]. However, it has been noted that even when using the $k$-d tree algorithm, the efficiency of the KNN method is limited in some cases [91]. Concerning the Octree, it has been broadly employed in fast index constructing and neighbor searching methods for 3D point clouds, particularly in the context of LiDAR data [92].

Furthermore, in the context of graph-based networks and point cloud analysis, $k$-d tree is utilized for conducting point-to-node $k$-nearest neighbor searches, allowing for the systematic adjustment of the network's receptive field [93]. Additionally, $k$-d tree is used in the establishment of a three-dimensional point cloud registration based on entropy and particle swarm optimization, where the relationship of points is established by $k$-d tree for finding the $k$-nearest neighbor of a point [94]. Octree, alternatively, has been used in the context of 3D change detection, where it is used for adaptive thresholds based on local point cloud density [95].

### 3.2.1 $k$-d tree

The $k$-d tree is a binary tree because a node cannot have more than two descendants. The $k$-d tree is built by recursively partitioning the data domain into two half spaces along the $k$-th dimension, being $k$ the number of features, including the spatial coordinates, that each entry in the dataset has. Typically, the stop condition when building the $k$-d tree is defined through the leaf size, i.e., the maximum number of points a node may have not to be partitioned again or the maximum depth of the tree. The axis along which the partition is carried out is a hyperplane, and its dimensionality is



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Algorithm 3.1** K-Dimensional Tree Construction

**Input:** $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$: set of points
`depth`: current depth of the tree
`k`: dimensionality of the KDTree

```
1: function BUILDKDTREE(P, depth, k)
2:     if P = ∅ then                                   ▷ Recursive base case
3:         return null
4:     axis ← mod(depth, k)
5:     split ← findSplitAxis(P, axis)  ▷ There are different strategies to find the split axis
6:     P_left = {p ∈ P : p ≤ split}
7:     P_right = {p ∈ P : p > split}
8:     node ← new Node()
9:     idx ← |P_left| − 1
10:    node.point ← p_idx                              ▷ Last point of P_left
11:    node.axis ← axis
12:    node.split ← split
13:    node.leftChild ← buildKDTree(P_left, depth + 1, k)
14:    node.rightChild ← buildKDTree(P_right, depth + 1, k)
15:    return node
```

**Algorithm 3.2** Compute Octant Idx

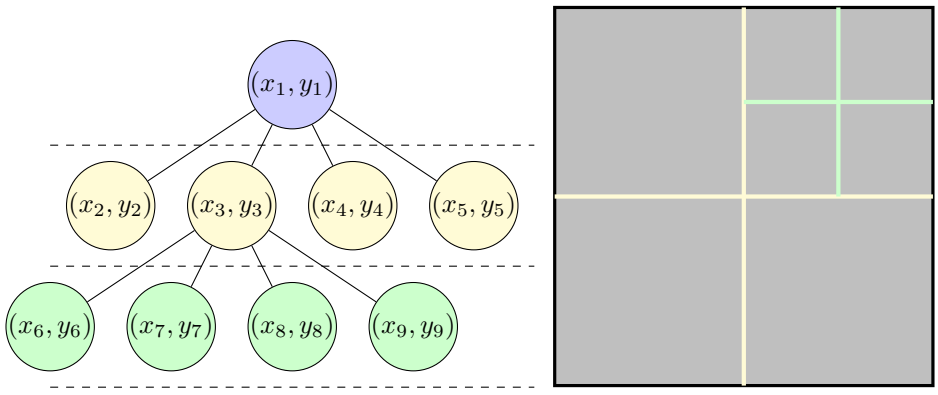
d **Input:** $p$: point to be inserted
$octant$: parent octant
**Output:** $idx$: octant child index where the point will be inserted

```
1: procedure OCTANTIDX(p, octant)
2:     idx ← 0
3:     if p.x > octant.center.x then idx |= 4
4:     if p.y > octant.center.y then idx |= 2
5:     if p.z > octant.center.z then idx |= 1
6:     return idx
```

exactly $k$-1. There are several strategies to compute the position of this hyperplane, being the most common the median or some kind of heuristic [96] of the data contained in the node to be partitioned. Once a partition is performed using a hyperplane, the next partition is carried out using the next axis, which is always orthogonal to the previous one. An example of a $k$-d tree is shown in Figure 3.6. The process to build a $k$-d tree is shown in Algorithm 3.1.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA




(a) Octree with 3 levels. (b) 2D domain of the Octree.

**Figure 3.7:** Example of an Octree partitioning a 2D dataset.

### 3.2.2 Octree

The Octree is a tree where each node has exactly eight children nodes. A node with no children is considered a leaf and contains the actual partitioned data. The space partitioning, in this case, is carried out using voxels: The root of the Octree is a voxel whose size's length is exactly the maximum radius of the minimum bounding box of the dataset. Each voxel is then iteratively subdivided into eight equal voxels by halving its length. The partitioning is usually stopped when a fixed number of points for a given voxel is reached. An example of a 2D Octree is shown in Figure 3.7. The process to build an Octree is shown in Algorithm 3.3. Note that the eight children are created at once only when the current node is full, as shown in Algorithm 3.4. The process to decide in which octant a point is inserted is shown in Algorithm 3.2. Finally, the insertion of a point is shown in Algorithm 3.5.

## 3.3 Performance evaluation

Both $k$-d tree and Octree play crucial roles in neighbor searches in point clouds, with $k$-d tree being known for its efficiency in multidimensional data and nearest neighbor searches, while Octree is widely utilized for organizing point clouds into voxels and in fast index constructing and fixed-radius neighbor searching methods for 3D point clouds. To the best of our knowledge, these data structures were reviewed and compared only two times in the literature. In [97], five implementations of $k$-d tree, one Octree, and one R-Tree are compared in terms of performance when used for k-nearest neighbors and fixed-radius queries, focusing on its use in shape registration. The authors use both artificial and real point clouds in their experiments. All the



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Algorithm 3.3** Octree construction function

**Input:** $\mathcal{P}$: set of points

1: **procedure** OCTREE($\mathcal{P}$)
2: $x_{min},\ y_{min},\ z_{min} \leftarrow \min\{p_x : p \in \mathcal{P}\},\ \min\{p_y : p \in \mathcal{P}\}, \min\{p_z : p \in \mathcal{P}\}$
3: $x_{max},\ y_{max},\ z_{max} \leftarrow \max\{p_x : p \in \mathcal{P}\},\ \max\{p_y : p \in \mathcal{P}\}, \max\{p_z : p \in \mathcal{P}\}$
4: `octant.center` $\leftarrow 0.5 \cdot (x_{max} + x_{min}), 0.5 \cdot (y_{max} + y_{min}), \leftarrow 0.5 \cdot (z_{max} + z_{min})$
5: `octant.r` $\leftarrow \max\{x_{max} - x_{min}, y_{max} - y_{min}, z_{max} - z_{min}\}$ ▷ Octant radius
6: `octant.min` $\leftarrow$ `octant.center` $-$ `octant.r`
7: `octant.max` $\leftarrow$ `octant.center` $+$ `octant.r`
8: **for all** $p \in \mathcal{P}$ **do**
9: `insertPoint`(octant, $p$) ▷ Insert point in the current octant

**Algorithm 3.4** Create child octants

**Input:** *octant*: parent octant

1: **procedure** CREATEOCTANTS(*octant*)
2: **for** $i \leftarrow 0, 7$ **do**
3: `newCenter` $\leftarrow$ `octant.center`
4: `newCenter.x` $\leftarrow$ `newCenter.x` + `octant.r` $\cdot$ (i & 4) $\neq 0$ ? 0.5 : -0.5
5: `newCenter.y` $\leftarrow$ `newCenter.y` + `octant.r` $\cdot$ (i & 2) $\neq 0$ ? 0.5 : -0.5
6: `newCenter.z` $\leftarrow$ `newCenter.z` + `octant.r` $\cdot$ (i & 1) $\neq 0$ ? 0.5 : -0.5
7: `octant.octants[i]` $\leftarrow$ `new Octant(newCenter, 0.5 · octant.r)`

real point clouds are terrestrial and created with static sensors. The other review is presented in [98], where a single unstructured mesh composed of 152,7 million nodes and 152,1 million hexahedral elements is used as a study case. In that review, a deep comparison between 20 open-source C/C++ libraries implementing different search methods is presented. Nevertheless, the study is carried out using synthetic data whose spatial distribution is unknown.

### 3.3.1 Datasets

In this chapter, we present a performance evaluation of the $k$-d tree and Octree data structures when used for k-nearest neighbor searches and fixed-radius queries across several types of point clouds. Our aim is to analyze the influence the different kernel sizes have on the query times for both data structures. The datasets were chosen to cover a wide range of point densities and extensions. We also used datasets that are open, publicly available, and widely used in the literature. These data sets represent different types of scenarios and platforms, as well as different LiDAR sensors. The



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Algorithm 3.5** Insert point in the given octant

**Input:** *octant*: parent octant
*p*: point to be inserted

```
1:  procedure insertPoint(octant, p)
2:      idx ← 0
3:      if octant.isLeaf() then                          ▷ If current octant has no children
4:          if octant.isEmpty() then                     ▷ If current octant has no points
5:              octant.points.add(p)
6:          else
7:              if |octant.points| > MAX_POINTS then     ▷ If octant is full
8:                  octant.createOctants()
9:                  for all p ∈ octant.points do         ▷ Move all points to children
10:                     idx ← octant.octantIdx(p)
11:                     octant.octants[idx].insertPoint(p)  ▷ Insert point in child
12:                 idx ← octant.octantIdx(p)
13:                 octant.octants[idx].points.add(p)
14:             else                                     ▷ Insert point in the current octant
15:                 octant.points.add(p)
16:     else                                             ▷ If octant has children, insert point there
17:         idx ← octant.octantIdx(p)
18:         octant.octants[idx].points.add(p)
```

| Name | Type | Sensor | Points | Extension | Scene | Density (pts/m$^2$) |
|---|---|---|---|---|---|---|
| DALES | ALS | Riegl Q1560 | 505 311 573 | 10.000 km$^2$ | Multiple | 50.53 |
| sg28_2 | TLS | - | 170 158 281 | 0.262 km$^2$ | Rural | 649.85 |
| sg28_4 | TLS | - | 258 720 948 | 0.050 km$^2$ | Urban | 5 181.00 |
| hallway_6 | TLS | Matterport Camera | 3 840 664 | 151.089 m$^2$ | Indoor | 25 419.88 |
| Paris | MLS | Velodyne HDL-32E | 80 000 000 | 0.048 km$^2$ | Urban | 1 030.83 |
| Lille | MLS | Velodyne HDL-32E | 50 000 000 | 0.798 km$^2$ | Urban | 100.27 |
| Speulderbos | ULS | Riegl VUX-1UAV | 79 221 314 | 0.033 km$^2$ | Forestry | 2 378.46 |

**Table 3.1:** Characteristics of the datasets used in the comparison. The sensor information used in the Semantic3D dataset is not publicly available.

chosen datasets and their main characteristics are depicted in Table 3.1.

The DALES [99] dataset is a well-known benchmark for evaluating deep learning classification models acquired using airborne laser scanning. The dataset is split in squared chunks of 250 000 m$^2$. For the purposes of this work, all the chunks were merged into one single point cloud where the benchmarks will be performed.

Regarding terrestrial laser scanning, two datasets were chosen. For outdoor areas,



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

two scenes from the Semantic3D [100] dataset were selected. The *sg28_2* scene corresponds to a farm in a rural area, and the *sg28_4* scene is a point cloud representing an urban environment. Other scenes of this dataset correspond to streets, which are better represented in mobile laser scanning datasets. The sensor information used for the acquisition of this dataset was not publicly divulged. An example of an indoor point cloud is also included and was obtained from the S3DIS [101] dataset. The selected scene is the one containing the greater number of LiDAR points: hallway_6.

MLS is often used to capture urban environments, which are properly represented by the Paris-Lille-3D [102] dataset. In this case, the whole dataset is not used as a study case, differentiating between Paris and Lille.

Finally, to the best of our knowledge, there are no ULS datasets that are the benchmark reference in the field. In this case, we have chosen the Speulderbos [103] dataset. Despite covering a relatively small area (0.018 $km^2$), it contains around 80 M points, so its utility relies on researching the behavior of the reviewed data structures in extremely dense point clouds.

The datasets are distributed in several formats: LAS, ASCII, and binary PLY. All were converted to the well-known LAS format.

### 3.3.2 Libraries

The libraries to carry out the study were chosen based on several criteria: Firstly, they must be publicly available and open-source. Secondly, they must be widely used in the literature. Thirdly, they must be coded in C or C++, as those languages and the tools they provide are the most used in High-Performance Computing (HPC) applications [104, 105]. The chosen libraries are the PCL [65], Boost [106], nanoflann [107], and unibnOctree [86].

The PCL is a widely used library in the field of point cloud processing. It provides a wide range of functionalities, including the implementation of a $k$-d tree and a Octree data structures. The selected $k$-d tree implementation in PCL was the one developed in the Fast Library for Approximate Nearest Neighbors (FLANN) library [108], which is a library for approximate nearest neighbor searches. Both data structures in PCL allow performing fixed radius searches, using spheres or voxels, and KNN searches.

The Boost library does not implement either a $k$-d tree or a Octree data structure, but a R-Tree, which allows for both radius-based searches and KNN.

The nanoflann library is based on the FLANN library, but it is more lightweight and faster, according to their authors. Unlike FLANN, where non-exact searches are allowed, the results of nanoflann are always exact. This library uses a $k$-d tree to perform fixed-radius searches and KNN.

Finally, the unibnOctree library is a C++ library that implements a Octree data structure. It is widely used in the literature for organizing point clouds into voxels and in fast index constructing and fixed-radius neighbor searching methods for 3D point clouds. Unlike normal Octree, each level of the Octree keeps track of all the points



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

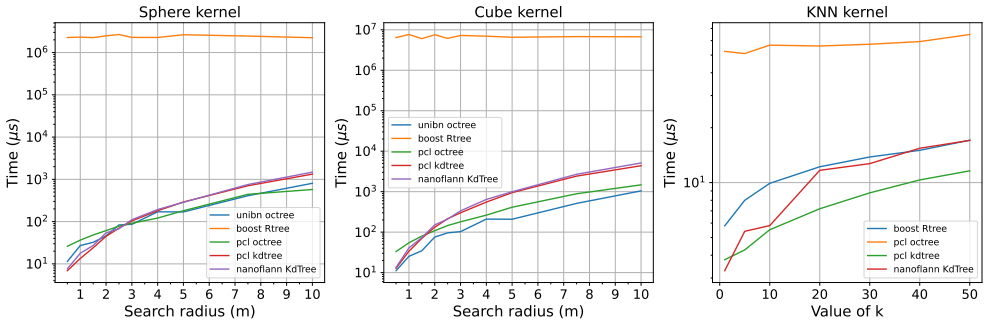



**Figure 3.8:** Query times for the DALES dataset using different libraries and search methods.

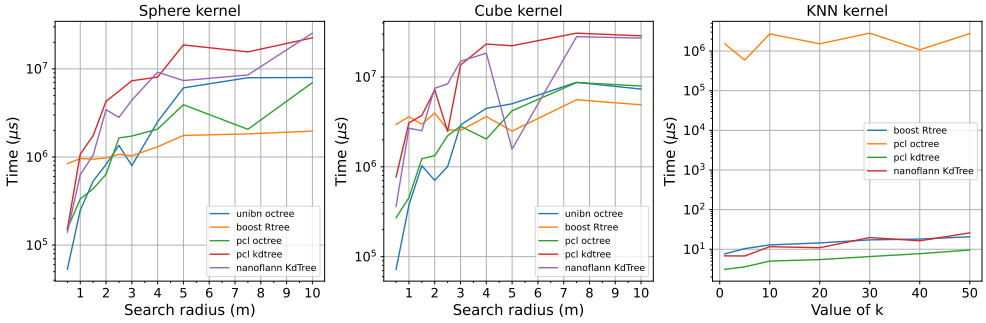



**Figure 3.9:** Query times for the *sg28_2* dataset using different libraries and search methods.

contained in the descendent octants. This allows for faster queries when the used kernel contains entirely a given octant. The library is able to perform fixed-radius searches but not for KNN.

### 3.3.3 Results

For each dataset and library, different radii and $k$ values were used to perform the queries. The query times are shown on a logarithmic scale, and the results are shown in milliseconds. The range of tested values is:

$$r = \{0.5, 1.0, 1.5, 2.02.5, 3.0, 4.0, 5.0, 7.5, 10.0\} \text{ m} \tag{3.9}$$

$$k = \{1, 5, 10, 20, 30, 40, 50\} \tag{3.10}$$



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

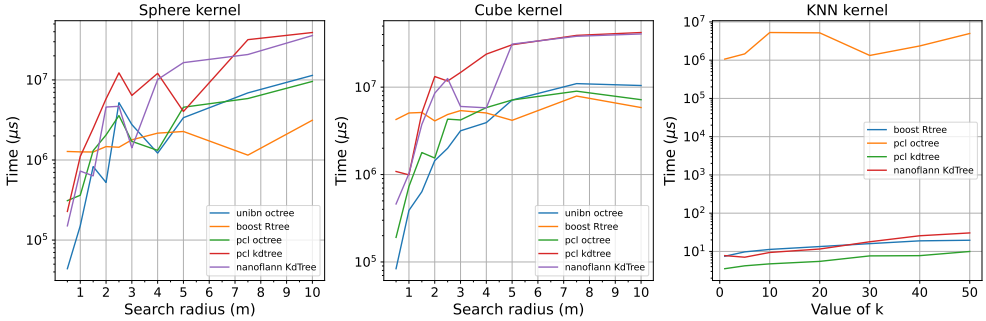



**Figure 3.10:** Query times for the *sg28_4* dataset using different libraries and search methods.

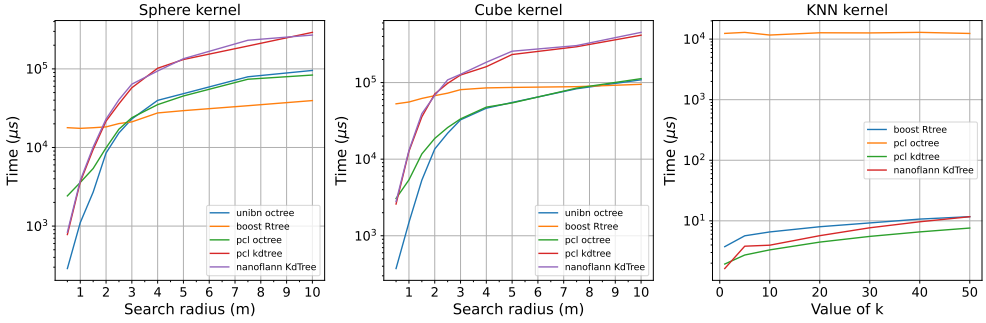



**Figure 3.11:** Query times for the hallway_6 dataset using different libraries and search methods.

Each query was run for 1 second or once if that time was exceeded. The shown results for each value are the mean of all the query times. The results are shown in Figures 3.8, 3.9, 3.10, 3.11, 3.12, 3.13, and 3.14.

As for the fixed-radius searches, the performance of the unibnOctree and the PCL Octree are similar, being unibnOctree slightly faster for small search radii, which occurs in all the analyzed datasets. Additionally, the other pair of libraries that show similar performance are the PCL $k$-d tree and nanoflann. Their performance for small radii values is similar to unibnOctree and PCL Octree, but for larger radii values, they perform worse, averaging half an order of magnitude that unibnOCtree and PCL Octree for most search radii. As for the Boost R-Tree, it is the slowest library for fixed-radius searches. Its search time is almost constant for all the search radii, but it is several orders of magnitude slower in four of the datasets while showing similar performance in *sg28_2*, *sg28_4*, and hallway_6 datasets.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

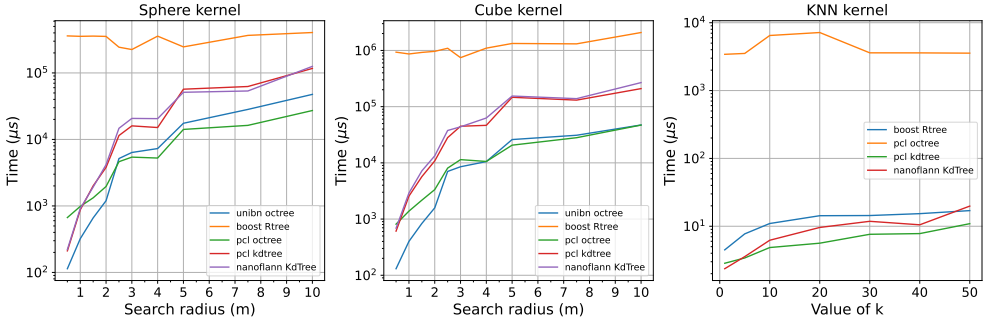



**Figure 3.12:** Query times for the Paris dataset using different libraries and search methods.

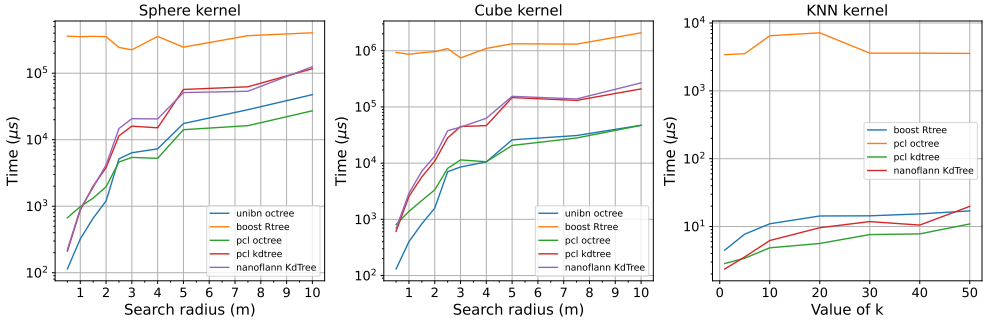



**Figure 3.13:** Query times for the Lille dataset using different libraries and search methods.

Concerning the KNN searches, Boost R-Tree is at least two orders of magnitude slower than the rest of the libraries in all cases. The other three (nanoflann, PCL Octree, and PCL $k$-d tree) show similar performances across all the datasets, being PCL $k$-d tree slightly faster in all cases.

## 3.4 High-performance computing for neighbor searching

When dealing with large amounts of data, it's often not feasible to process them using a conventional computer. Although the processing power of personal computers has multiplied in recent years, so has the volume of generated data. In this regard, HPC refers to the ability to process data and perform calculations using supercomputers and computer clusters. The decreasing cost of LiDAR sensors and advancements in



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

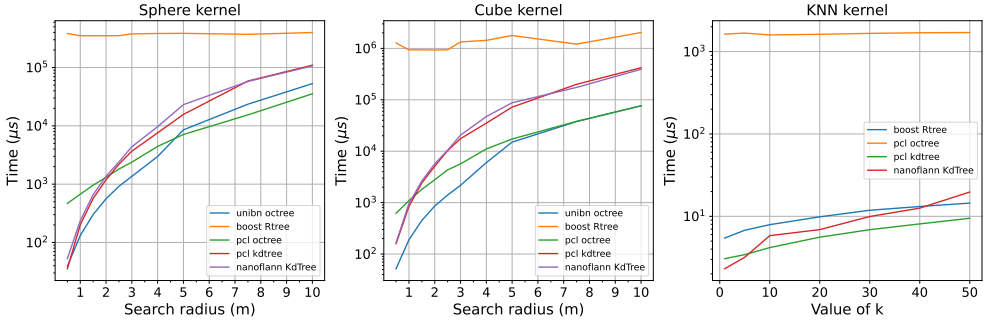



**Figure 3.14:** Query times for the Speulderbos dataset using different libraries and search methods.

data acquisition techniques have led to the generation of increasingly larger point clouds. Therefore, the use of HPC techniques and systems is necessary to process them within a reasonable timeframe. The ultimate goal of employing HPC in data processing is to reduce computation time, which can be motivated by three factors: reducing computation time, decreasing energy consumption, and cutting economic costs. Thus, when developing algorithms for point cloud processing, it's crucial to be able to run experiments within a reasonable time frame, substantially speeding up the iterative process of designing, implementing, and evaluating algorithms.

Algorithm design must consider the hardware on which it will be executed and parallelized to utilize the available hardware efficiently.

As mentioned at the beginning of this chapter, the most common operation when processing point clouds is neighbor searching. Therefore, parallelizing this task is crucial to minimize processing times for a given point cloud. Once the data structure for neighboring searches is chosen, the final computation time for a given neighborhood depends mainly on two factors: the type of kernel used and the local data distribution around the analyzed point. Regarding the type of kernel, the most computationally expensive operation is checking whether a point lies within the kernel. The more complex the geometry, the more costly this calculation becomes. Concerning the local distribution of points, it affects the number of times a point needs to be checked if it is inside a kernel or not.

The neighborhoods of two different points may overlap, but their computation is performed independently. Therefore, most point cloud processing algorithms are embarrassingly parallel, meaning that parallelization at the point level is not only easy to implement but is often the best available option. Thus, the shared memory parallelization paradigm has been chosen. All algorithms presented in this thesis have been implemented in C++20, one of the most commonly used languages in HPC [104, 105], using the OpenMP API for parallelization. Shared memory programming assumes



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

that all processors have access to the entire dataset, so performance gain is achieved by assigning each processor to work on only a portion of the dataset. In this parallel paradigm, locality, load balance, and affinity are the topics on which the developer must focus to improve the performance of their codes. All of them are analyzed next.

Given the irregular nature of point clouds, it's not difficult to imagine that workloads may be unbalanced across different computing units. To deal with this circumstance, OpenMP offers various tools to control workload distribution among available threads. Specifically, the schedule option allows configuring how loop iterations are distributed among threads. The three main configurations are static, dynamic, and guided. The static configuration distributes iterations evenly among threads, while the dynamic configuration distributes iterations so that each thread receives a number of iterations based on those it has completed. The guided configuration is a combination of the previous two, distributing iterations so that threads receive a decreasing number of iterations as they progress. Intuitively, it's deduced that both the dynamic and guided options are most suitable for the type of processing to be performed.

To determine the best possible scheduling, performance tests have been conducted on different architectures regarding neighbor searching. Specifically, search times for neighbors with the four most commonly used kernels —square, circle, cube, and sphere— have been measured using Machine 3 and Machine 2, with a radius of 0.5 m, and the results are shown in Figure 3.15. The measured times correspond to the neighborhood computation of all the points contained in a tile of the DALES dataset [109]. The times were computed as the median of ten executions in each machine. All the scheduling strategies showed similar scalabilities, being the simplest kernels —square and cube— the ones that benefit the most from parallelization. For the rest of the experiments in this thesis, the *guided* scheduling strategy has been chosen. Since most of the algorithms are based on neighbor searches, the upper bound of the speedup for any implemented algorithm is dictated by the speedup of the kernel type used.

The parameter that strongly influences the neighborhood search time, and therefore, its scalability, is the search radius. The larger the search radius, the more points need to be checked to determine if they are inside the kernel. The influence the different radii have in the scalability was studied using Machine 3 and Machine 2, and the results for the circle kernel are shown in Figure 3.17 and Figure 3.16, respectively. The results for the rest of the kernels are shown in Appendix B. The scheduling strategy was set to *guided*.

In Machine 3, the scalability is excellent for radii up to 2 m, with an efficiency higher than 0.9 for most of the threads. When the search radius increases, the number of threads where the efficiency starts to drop decreases. The effect the different Non-Uniform Memory Access (NUMA) nodes have on the scalability is only observed for a search radius of 8 m, where there are sudden drops every 10 threads approximately. As for Machine 2, the scalability is noticeably worse when more than one NUMA node comes into play. The scalability for a high number of threads for radii below 2 m is around 0.6, whereas for the Machine 3, the efficiency is around 0.85. The scalability for



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

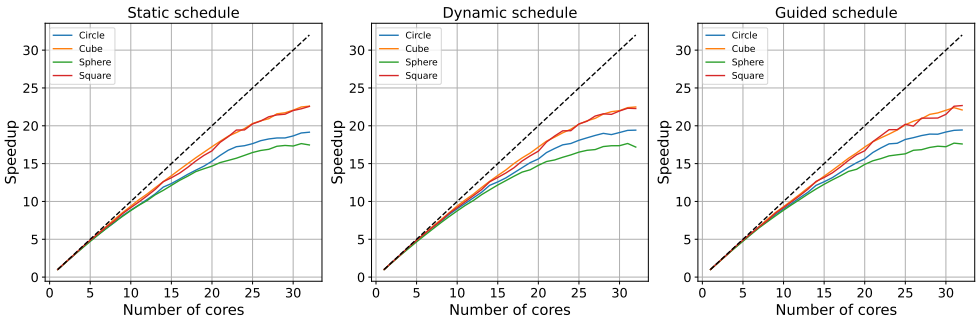



**Figure 3.15:** Speedup for different kernels using three OpenMP scheduling strategies.

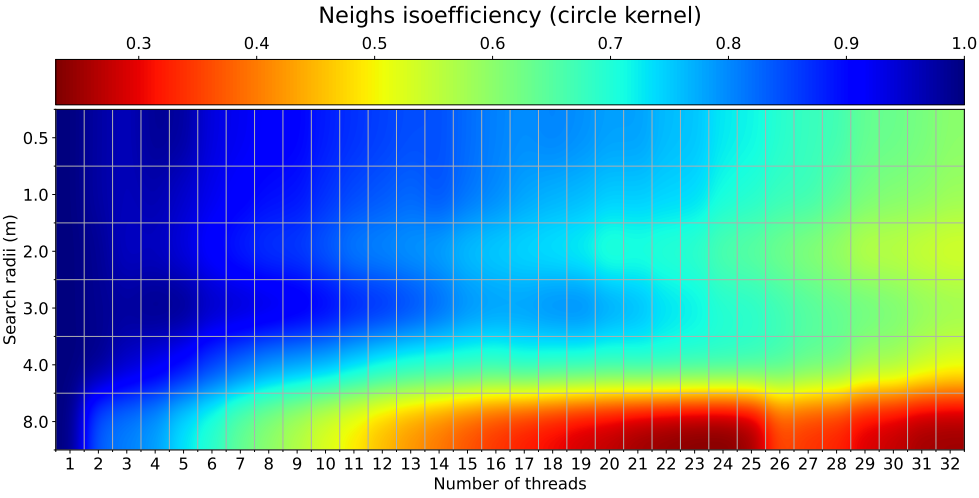



**Figure 3.16:** Isoefficiency matrix for the circle kernel in Machine 2.

the highest search radius is poor in both machines, probably due to the high amount of octants that must be checked for each point.

Another tool provided by OpenMP is the affinity policies. These policies allow binding threads to specific cores, which can be useful to avoid cache misses and to reduce the time spent on thread migration. Typically, it is cheaper to move a thread to the data than to move the data to the thread [110]. The thread affinity policies allow the binding of a thread to a specific hardware resource. In the case of machines



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

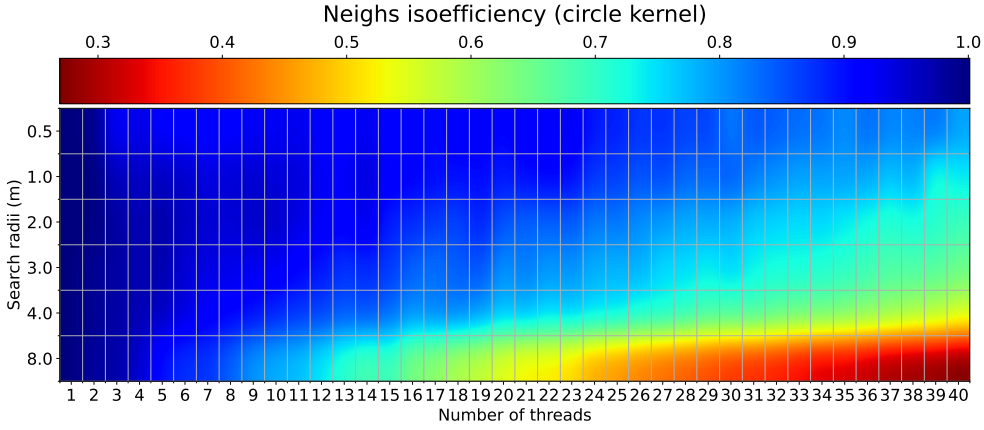



**Figure 3.17:** Isoefficiency matrix for the circle kernel in Machine 3.

with multiple sockets and NUMA nodes, it is important to bind threads to the same socket or NUMA node to avoid the overhead of accessing data from a different node. Several experiments carried out in this thesis use machines with several sockets and NUMA nodes, so the thread affinity policies have been configured to place all the execution threads in the same socket and node.

## 3.5 Conclusions

Based on the performance analysis of the different libraries and query methods, the data structure chosen to work with is the unibnOctree. This decision is based on the performance of the fixed-radius searches, where the unibnOctree is the fastest library for all the datasets. While it may not be the fastest for search radii starting from 3 m, its performance matches the fastest library in those ranges. Also, the most used radii in the algorithms developed in this thesis are below the 3 m threshold. Furthermore, unibnOctree is a header-only library, which makes it easier to integrate into the developed algorithms, while PCL, the other possible candidate for fixed-radius searches, must be compiled and have several dependencies.

Regarding the scheduling strategies, all of them show similar results, so the *guided* strategy was chosen for the rest of the experiments of this thesis. We have shown that the scalability of the different kernels in the neighborhood search is close to 1 when the search radius is below 3 m and drops with the neighborhood radius.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Chapter 4

# Optimal pathfinder in 3D point clouds

*Greed, for lack of a better word, is good. Greed is right. Greed works.*
— Gordon Gekko, *Wall Street.*

The Least Cost Path (LCP), also known as optimal path or minimum-weighted path, is a concept widely used in physics, mathematics, engineering, or geography. It is typically defined as the route between two points that minimizes the cost of travel from one of them to the other. The LCP has a different definition in each of the aforementioned fields, but they all agree to define it as the path that minimizes a given mathematical function.

Thus, in physics, the free motion of any body follows the trajectory that minimizes the energy required to perform it (known as the principle of least action). In mathematics, the LCP is the trajectory that minimizes the value of a given cost function (for example, the distance). In the engineering field, the LCP is typically used to minimize the cost of construction of different infrastructures, such as roads [111], railways [112], or power transmission grids [113]. In geography, the LCP is the trajectory that minimizes the cost of traversing the path. The definition of cost in this case may vary depending on the application: it could be the distance, the time, the energy, the risk, etc. Finding the LCP in the field of geography is also known as *path planning* or *path finding*.

Depending on the scope of the problem, path planning can be classified into two categories: local and global. On the one hand, local path planning is the computation of the LCP in a small area, typically in the order of a few meters, using the information from different sensors obtained in real time in order to avoid immediate obstacles. This also implies that local versions of the optimal path are found, whose aggregation may not necessarily be the optimal global path. On the other hand, global path planning is the computation of the LCP in a big geographical area, typically in the order of

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

kilometers, where the information of the whole area is known. Local path planning is typically used in robotics, where the robot must compute the LCP in a small area to avoid obstacles and reach a goal, whereas global path planning is typically used in geography, where the LCP is computed in large geographical areas in order to find the optimal route between two points.

A combination of local and global path planning is presented in [27], where the information provided by OpenStreetMap is used for global path planning, and corrections to that data are performed through real-time information collected using LiDAR sensors. Typically, in local path planning, the cost of traversing the path is not taken into account, as is the case of [114, 115].

The computation of the LCP in large geographical areas has become feasible in the last decades thanks to the appearance of remote sensing techniques that allow a very precise and accurate acquisition of the terrain surface. Particularly, as described in Chapter 2, LiDAR sensors are able to capture the terrain surface with very high resolution, allowing the computation of the LCP.

Typically, the main strategy to compute the LCP by using a 3D point cloud is to rasterize the scene by calculating the DTM, which results in a 2D raster image where each pixel contains the elevation of the terrain surface at that point.

A large number of works in the literature follow the same strategy in order to compute the LCP: a graph is built from the DTM, where each cell is a node and the edges are the connections between the cells, so every node has eight neighbors, with the exception of the borders and the corners of the raster. The cost of traversing each edge is then computed by using the elevation of the cells that are connected by the edge. That cost is known as the weight of each edge in the graph. Then, the LCP is computed by using a graph search algorithm.

Computing the LCP by using the DTM is widely encountered in the literature. In [116], Metz et al. present an improvement of an existing algorithm that uses the LCP on a DTM to compute the drainage networks in a given area. Sarı et al. [117] use the computation of the LCP in terms of economic costs over a DTM to select the route where a highway will be built. A similar application is presented in [111], where the authors compute the trajectory of a forest road with the minimum construction cost. Also, Ress et al. [118] use the LCP computed over a DTM to compare the optimal routes with the already present footpaths. Another application is the utilization of the LCP in determining the localization of optimal log landing, as elucidated by Contreras et al. in [119]. Also, a notable example is presented in [120], where Liu et al. develop an LCP-based method for identifying evacuation routes in areas prone to flood disasters. More recently, Aguiar et al. [121] explore the application of different LCP algorithms to minimize both economic costs and environmental impacts associated with road construction in the expansive Amazon rainforest. Lastly, Flisberg et al. [122] employ a DTM to compute optimal log extraction routes in forested areas, with a primary focus on minimizing the adverse effects of these operations on soil and water.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

| Reference | Local / Global | Discretization | Indoor / Outdoor | Weights | Algorithm |
|---|---|---|---|---|---|
| [27] | Both | No | Outdoor | No | Dijkstra |
| [114] | Global | No | Outdoor | No | RRT |
| [115] | Both | Yes | Indoor | No | AD* |
| [119] | Global | Yes | Outdoor | Slopes | Dijkstra |
| [111] | Global | Yes | Outdoor | Slopes<br>Construction cost | Dijkstra |
| [118] | Global | Yes | Outdoor | Slopes | Dijkstra |
| [122] | Global | Yes | Outdoor | Length | Own implementation |
| [117] | Global | Yes | Outdoor | Length | Dijkstra |
| Our proposal | Global | No | Outdoor | Length, slopes<br>roughness and NDVI | Dijkstra |

**Table 4.1:** Characteristics for state-of-the-art path-planning on 3D point clouds.

However, the use of the DTM to compute the LCP presents several issues. Firstly, and since only a single height value is assigned to each cell of the DTM, it does not contain any information about the obstacles that may exist in the study area, but only about the terrain elevation in a given point. Due to the nature of the laser scanning process, the registration of multiple echoes poses a challenge in forested regions: points with identical horizontal coordinates but varying elevations are identified, introducing difficulty in their representation on a standard grid, as highlighted in [123]. Secondly, the absence of data in certain regions must be managed, typically by interpolation of the values of the nearby cells [124]. The interpolation may lead to differences between the real terrain surface and the one represented in the DTM [125]. Note that different interpolation algorithms may lead to different outcomes and may have a big impact on the final modelization [126]. Finally, the choice of the cell size is critical to represent the terrain variations across different areas properly. For example, in [119], a 50 m resolution raster is used, and the authors state that the terrain conditions cannot be accurately described. Zhou et al. [127] outline the effects that precision and resolution may have on the final DTM.

Several works do not use weights when computing the LCP, as shown in Table 4.1. In that case, path planning consists of avoiding obstacles to reach the destination. Some works use the distance between nodes as the cost function to compute the LCP. However, in most applications, the distance is not the only factor that must be taken into account when computing the LCP. For example, the slope of the terrain is also a very important factor since it determines the difficulty of traversing the terrain. There are several methods encountered in the literature to perform the slope compu-



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

tation. Thus, in their study, Campbell et al. [128] employ trigonometric techniques to transform the height difference and distance between adjacent cells on a slope. In contrast, Rees [118] computes the slope percentage as $\frac{dh}{dx}$, where $dh$ represents the gained height, and $dx$ denotes the horizontal traveled distance. The computation of slope in a grid has been explored by Tang et al. [129], who analyzes different methods. The conventional approach in these methods involves the use of $3 \times 3$ moving windows to compute finite differences or local polynomial surface fits. Despite the variety of techniques, the maximum number of points used for slope computation ranges from eight in the best cases to only two points in the worst-case scenarios.

The primary limitation of the aforementioned methods lies in their computation of a single slope for each DTM cell. This approach implies that the cost associated with traversing the path between two adjacent cells is attributed to one of the two cells rather than to the actual path taken. Additionally, these methods overlook the direction of the slope, treating uphill and downhill slopes, as well as traversing the slope perpendicularly (without gaining or losing any height) with equal significance.

Other methods to compute the slope imply the computation of the best fitting plane for each DTM cell. Typically, the height data of the cell itself and its neighbors are used for this purpose. This is the case exposed in [130], where the best-fitting plane is computed using the least-squares method, and the slope is derived from the plane's normal vector. Klasing et al. [131] perform an in-depth study of the slope computation methods using approximate planes. This work divides the methods into two main categories: optimization-based methods and averaging methods. Optimization methods rely on minimizing or maximizing specific criteria, such as the distance of each point to a local plane or the angle between tangential vectors and the normal vector. Conversely, averaging methods involve the computation of normal vectors based on the averaging of all possible triangles formed by a given point and its neighboring points. Comparing both categories, authors find that optimization-based methods, particularly PlaneSVD, PlanePCA, and QuadTransSVD, exhibit lower run times compared to averaging methods, even in smaller neighborhoods.

We have already mentioned that most authors use the length between cells as a weight factor to compute the LCP. In that case, the value of the length can be used directly as the weight value. In the case of taking the slopes into account, an estimation of the effort needed to traverse a steep terrain usually defines the weight. The oldest known estimation of the effort to climb a hill was provided by William W. Naismith [132] in 1892: "*an hour for every three miles on the map, with an additional hour for every 2,000 feet of ascent*". In the following years, several authors added some corrections to the Naismith proposal [133], while others developed their own models. One of the most-known models is the Tobler's hiking function [134]. This function was derived from empirical results, and it is the go-to method to compute travel times in Geographical Information Systems (GIS) [135]. In the work presented in [136], the data of several hikers were collected using a treadmill with different inclinations. They found that the relation between the slope and the speed is an inverse exponential. In



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

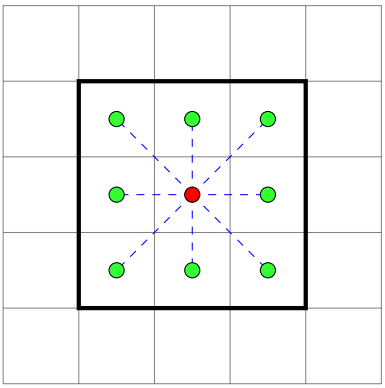


**Figure 4.1:** Neighboring cells in a DTM grid. The red circle is the node whose neighbors are being computed. The green cells are the eight neighbors of the red node.

a study carried out by Rees [118], the author used his own unpublished data to fit the uphill speed function to a second-degree polynomial function. A similar study is presented by Irmischer and Clarke [135], where they conclude that the movement cost as a function of the slope follows an exponential relation. The most complete study is presented by Campbell et al. [137], where the collected data of 421,247 hikes, jogs, and runs were fitted to three probability distribution functions: Lorentz, Gauss, and Laplace. The authors recommend the use of the Lorentz distribution since it outperforms the other ones in a vast number of situations. Finally, a study comparing several methods to estimate the effort of climbing a hill is presented in [138].

As we have just discussed, most of the proposed methods to compute the LCP using 3D point clouds rely on the information provided by a DTM, and, typically, only the length of the path and the value of the slopes are used to build the function to be minimized. In the approach presented in this thesis, we not only avoid the need to compute the DTM, but we propose a slope computation method that takes into account all the points available in the cloud. Also, the roughness of the terrain is computed and plays an important fact in the cost-minimization process. Finally, we propose the use of the Near-Infrared (NIR) band, when available, to demonstrate the potential of the information provided to search a path that avoids crops and other types of living vegetation. As a consequence of this approach, we avoid the loss of information associated with the use of the DTM and increase the accuracy by considering every point in the cloud to estimate the slope.

This work was motivated by the need to find a safe route in emergency situations as fast as possible. Thus, this algorithm is being used together with the one presented in [139] to obtain the fastest route between the landing zone of a UAV that carries medical material and the site of an emergency.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

## 4.1 Graph construction

In our proposal, as the above-mentioned ones, to compute the LCP, a graph to work with must be defined. The state-of-art works use the rasterization obtained from the computation of the DTM to build the graph, using the center of each cell as the nodes of the graph and linking those nodes with the adjacent cells, as shown in Figure 4.1. This results in each node having eight links, whose orientations are always vertical, horizontal, and diagonal. The main benefit of this approach is the regularity of this graph, making it easy to work with it because of its similarity with images. However, this method has a main drawback: the topology of the resulting graph is agnostic with respect to the point cloud itself. To address this issue, we adopt a segmentation approach focusing solely on the spatial proximity of points within the point cloud.

In general, a graph is defined as follows:

$$\mathcal{G} = (V, E) \tag{4.1}$$

where $V$ is the set of vertices (nodes) and $E$ is the set of edges or links that connect the vertices of the graph. The flowchart of the steps needed to build the graph and compute the LCP is shown in Figure 4.2.

The graph nodes are defined via a segmentation process starting from a random seed point, and progressively incorporating nearby points into the same group if their distance falls below a predefined threshold. This iterative process continues until no further points can be added to the group. The algorithm responsible for segmenting the point cloud is a variation of the well-known Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [140], without considering isolated points. The outcome of this segmentation yields a collection of groups, each containing points closely situated to one another. In this stage of the algorithm, the resolution of the graph can be controlled indirectly through any of the two parameters that govern the segmentation process: the maximum number of points per group $N_s$ and the radius of the neighborhood $R_s$. $N_s$ governs the size of the groups, taking into account the density of the point cloud: areas with low density will yield larger groups than areas with high density. On the other hand, $R_s$ limits the size of the groups with its value, useful for indirectly controlling the number of groups of the resultant segmentation. Note that by using this method, the information of every point in the scene is preserved. The segmentation can be adapted to form groups from hundreds or thousands of points to just one. The smaller the number of points in each group, the greater the number of nodes conforming the graph. This versatility allows the graph to be adapted to every type of point cloud (rural areas, roads, urban environments, etc.). Finally, this approach has the advantage of preserving the information of the point cloud since the points belonging to each group are available to compute the weights of the edges.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

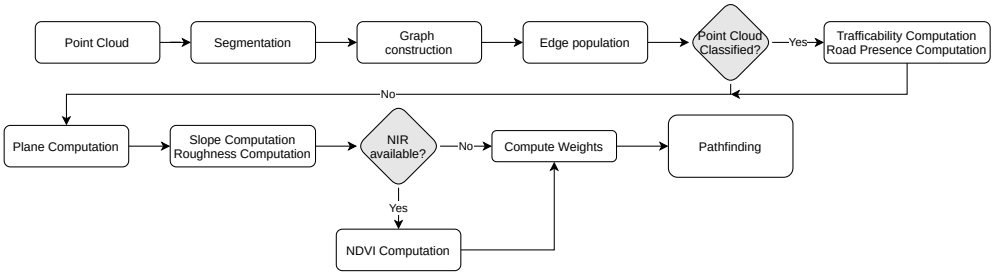



**Figure 4.2:** Flowchart of the pathfinding algorithm.

Subsequently, we will use these groups' centroids as nodes for the graph construction, forming the set $V$.

Once the nodes are defined, it remains to decide which nodes are connected by an edge, which yields a complete graph. In the state-of-art works, the edges are defined using the neighboring cells provided by the DTM. In contrast, our approach uses the neighboring information provided by a Voronoi tessellation [141]. By using the centroids computed in the segmentation, we build the Voronoi diagram, which partitions the dataset into irregular polygons. Each polygon contains just one centroid of the segmented groups, and the points inside the polygon are guaranteed to be closer to the centroid of that polygon than to any other centroid. Thus, two groups are considered neighbors if their corresponding Voronoi polygons have a common side. This method could give spurious neighboring information, especially on the groups owning points belonging to the scene's borders. The Voronoi partition tends to build infinite area polygons in the borders of the datasets, leading to false positives in the neighboring information. The mean and the standard deviation of the edges' length in the scene are computed to remove the false positives. Then, those edges whose length is greater than three times the standard deviation are removed.

Several algorithms to compute the Voronoi tessellation are found in the bibliography. The first computationally viable were all developed in the 1980s. Fortune's Sweep-Line [142] is lauded for its efficiency $\mathcal{O}(n \log n)$ in both Voronoi diagram constructions and Delaunay triangulations [143] (the dual of the Voronoi tessellation, see Figure 4.3). This algorithm offers excellent versatility and handles degenerate cases effectively. It stands as the go-to method for large datasets due to its robust performance and well-established foundation. However, its implementation complexity requires careful consideration when choosing this approach. Another algorithm is the Bowyer-Watson Incremental Insertion [144, 145]. This algorithm shines with its simple and intuitive implementation, making it particularly suitable for educational purposes or small datasets. While its $\mathcal{O}(n^2)$ worst-case time complexity makes it less efficient



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

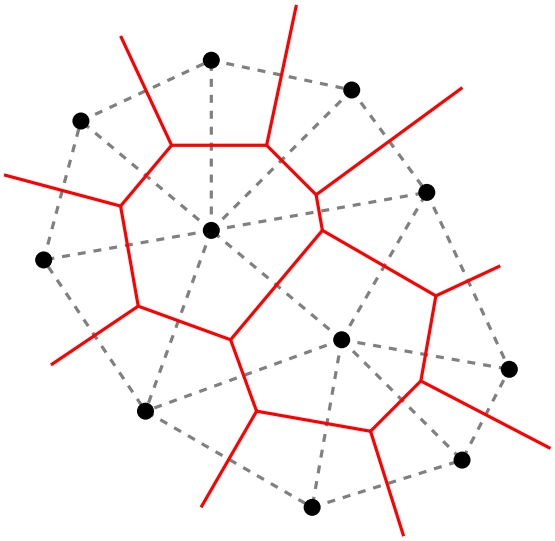


**Figure 4.3:** Voronoi and Delaunay tessellations on a random set of nodes. The black circles are the nodes of the graph. The red lines are the Voronoi tessellation edges, and the dashed back lines are the corresponding dual graph: the Delaunay triangulation.

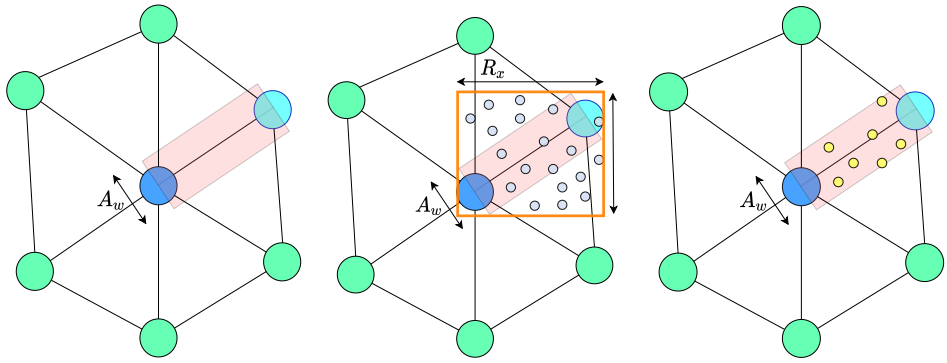
for large-scale scenarios, its ease of understanding and practical implementation can be valuable assets in initial explorations. Notice that the algorithm was independently developed by Bowyer and Watson and was published in different papers in the same issue of the journal. Another popular method is the Lloyd's Relaxation [146]. Taking an iterative approach, this algorithm excels in applications involving point sets with specific properties like uniform density. Its low memory requirements offer an advantage, but its convergence and stopping criteria can be more challenging to handle compared to other methods. While not the most widely used in general-purpose applications, it finds its niche in specific scenarios where its strengths can be leveraged effectively. While these algorithms are 2D-oriented, 3D versions also exist in the literature. Fortune's Sweep, Watson's [147], and Lloyd's Relaxation algorithms were extended to be applied on 3D point clouds.

Apart from these classical methods, some modern approaches offer better performance and are more robust to degenerate cases where two or more points are collinear or coincident. The most notable are the algorithms based on the Delaunay triangulation, a well-studied problem in computational geometry. Several algorithms have been developed to solve this problem, the most notable being the incremental algorithm, the divide-and-conquer algorithm, and the sweep-line algorithm.

The Delaunay triangulation $\mathcal{T}(V)$ is composed by all the triangles $\triangle_{v_i,v_j,v_k}$ formed by the $\{v_i, v_j, v_k\}$ triplet of vertices satisfying that $S(v_i, v_j, v_k) \cap V = \{v_i, v_j, v_k\}$, where



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA




(a) Creation of the rectangle around a given edge.

(b) Neighborhood computation around the edge.

(c) Inclusion of the points in the edge.

**Figure 4.4:** Determination of the points lying inside a given edge. The blue circle is the graph node whose edges are being built. The neighbor used to build the edge is shown in light blue. The nodes in green are graph nodes that will be processed later. The gray circles are points found in the initial search, and the yellow ones are those finally included in the edge.

$S(v_i, v_j, v_k)$ is the circumcircle of the triangle $\triangle_{v_i,v_j,v_k}$, and $V$ is the set of graph nodes. In our proposal, we opted for using the C++ library implemented in [148] to compute Delaunay's triangulation. This library is a port to C++ of the JavaScript version available at [149]. This implementation is based on the work and ideas presented in [150–152].

After the triangulation is obtained, the neighbors of each node are extracted by looking at the Delaunay's triangulation edges that are connected to the current node. The totality of the connections between nodes forms the set $E$. At this point, a bi-directed graph $\mathcal{G} = (V, E)$ is defined. The edges of the graph can be traversed in both directions, giving a total of two edges joining every pair of nodes. This is done to reflect the different costs of traversing the same edge in opposite directions.

To assign LiDAR points to every edge, an encompassing rectangle around the edge is defined. This approach offers a notable advantage over utilizing a DTM: it leads to a more precise feature computation as it allows for the inclusion of every point within the point cloud to compute the weights accurately. The rectangle's dimensions are determined by a parameter called the Agent Width ($A_w$), which serves as the reference width for computing the LCP. To construct the rectangle, two parallel lines are projected from either side of the edge, each extending a distance of $A_w/2$. By connecting these lines, a closed rectangle enclosing the edge is formed. The enclosed area represents the extent covered by the edge, as depicted in Figure 4.4(a).

To compute the points inside the defined rectangle, a square kernel is used centered



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

in the edge's bounding box. The radius of the kernel is set to $R = 0.5 \cdot \max(R_x, R_y)$, where $R_x$ and $R_y$ are the dimensions of the edge's bounding box, as shown in Figure 4.4(b). The election of the radius is chosen so no points inside the edge are left outside the kernel. For each point in the kernel, whether they lie inside the edge or not is determined. This is possible since the coordinates of the corners of the rectangle defining the edge are known. The points that remain in the edge at the end of the process are shown in yellow in Figure 4.4(c).

Once the edges have been defined, their *trafficability* can be quantitatively assessed in the case the input point cloud is classified. Trafficability refers to the capability of an agent to traverse a specific terrain type. To achieve a quantitative measure, the number of points classified as belonging to roads or ground within the edge is considered. Consequently, an edge is classified as *trafficable* if the proportion of ground or road points surpasses a predefined threshold, denoted by $T_t$ in Equation 4.2:

$$\frac{n_t}{n_p} \geq T_t \tag{4.2}$$

where $n_t$ represents the number of points classified as road or ground, $n_p$ denotes the total number of points within the edge, and $T_t \in [0, 1]$ is the *Trafficability Threshold*. This threshold defines the minimum proportion of trafficable points an edge must possess to be considered trafficable. Edges that fail to meet this criterion are excluded from the graph, so a subsequent update of the nodes' neighboring information to reflect this change is needed.

## 4.2 LCP computation

The most efficient route between two points is determined by minimizing the exertion required for traversal. Therefore, the optimal path may not always be the shortest, as various landscape characteristics come into play. Factors such as slopes, terrain roughness, and vegetation density significantly influence the effort required to navigate a path. To find the optimal route, it is necessary to assign a quantitative measure of traversal effort to each edge derived from the point cloud data.

In this thesis, we consider several factors to assess traversal costs, including edge length, road presence, longitudinal and transversal slopes, terrain roughness, and vegetation density. These attributes are computed based on the points encompassed within the assigned rectangle for each link, serving as specific parameters for cost evaluation.

The slope of a given edge in the graph is given by the slope of the best-fitting plane for the points within the edge. Several methods for computing the best-fitting plane exist. For example, in [131] a comparison of different methods for the estimation of surface normals is presented. In this work, the Plane Singular Value Decomposition (PlaneSVD) method was chosen. Although PlaneSVD is computationally expensive, it is more numerically stable in a broader range of cases than other methods since the



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

decomposed matrix requires weaker conditions. The algorithm can only be applied to those edges containing at least 3 points. In order to compute the PlaneSVD, a matrix $\mathcal{P}_{3\times n}$ is defined, in which each column corresponds to the mean-normalized spatial coordinates of each point, and $n$ is the number of points belonging to the current edge. Furthermore, the centroid of each dimension must be subtracted from each row of the matrix $\mathcal{P}$ in order to obtain a 0-main row-wise matrix.

To obtain the matrix $\mathcal{P}$, the centroid $\bar{\mathbf{r}} = (\bar{x}, \bar{y}, \bar{z})$ of the set of points in each edge is computed using the following expression:

$$\bar{\mathbf{r}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{r_i} \tag{4.3}$$

where $\mathbf{r_i} = (x_i, y_i, z_i)$ is the 3D position of each point. Matrix $\mathcal{P}$ is then defined as:

$$\mathcal{P} = \begin{pmatrix} \mathbf{x} - \bar{x} \\ \mathbf{y} - \bar{y} \\ \mathbf{z} - \bar{z} \end{pmatrix} \tag{4.4}$$

where $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ are the vectors containing the $x$, $y$ and $z$ coordinates of every point in the edge. The SVD method is then applied over this matrix.

The left-singular vectors obtained from this matrix are an orthonormal base corresponding to the distribution of the points currently analyzed. The third vector of this base is the one that matches the normal vector $\mathbf{n}$ of the searched plane. Thus, the equation of the plane for each edge is

$$\pi \equiv n_x x + n_y y + n_z z + d = 0 \tag{4.5}$$

where d is

$$d = -(n_x \cdot \bar{x} + n_y \cdot \bar{y} + n_z \cdot \bar{z}) \tag{4.6}$$

A final consideration must be taken into account regarding the orientation of the computed normals in this application. To compute the slope of a plane, its normal must face upwards. Thus, all the normals whose $z$ component is negative are flipped to meet this requisite.

The slope of each edge is computed as the angle subtended between the normal vector of its plane and the vector $\mathbf{k}$, i.e., the unitary vertical vector. The resulting angle corresponds to the slope in the downhill direction, which is the maximum slope among all the directions. This is known as the slope aspect and can be computed using the gradient operator in two dimensions over the normal vector of the plane, as shown in Equation 4.7. Its magnitude is given by Equation 4.8.

$$\boldsymbol{\nabla}\pi = n_x \mathbf{i} + n_y \mathbf{j} \tag{4.7}$$



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

$$s = \arccos\left(\frac{n_z}{\sqrt{n_x^2 + n_y^2 + n_z^2}}\right) \tag{4.8}$$

Both the slope aspect $\boldsymbol{\nabla}\pi$ and the magnitude $s$ are used to compute the slope magnitude both along the longitudinal ($s_l$) and perpendicular ($s_t$) directions of the edge:

$$s_l = s\boldsymbol{\nabla}\pi \cdot \mathbf{u} \tag{4.9}$$

$$s_t = s\boldsymbol{\nabla}\pi \cdot \mathbf{u}_\perp \tag{4.10}$$

where $s_l$ is the slope value in the direction $\mathbf{u}$ of the edge, $s_t$ is the value in the transversal direction $\mathbf{u}_\perp$. These values will be used later to compute the cost of traverse each edge.

The algorithm presented in this thesis was designed with the following in mind: traveling through roads is usually easier than going through other types of terrain. Thus, the presence of roads in the scene is taken into account when computing the weights of the edges. With this in mind, each edge is classified as road or not road based on the proportion of road points within the edge:

$$\frac{n_r}{n_p} \geq T_r \tag{4.11}$$

where $n_r$ is the number of points classified as road, $n_p$ is the total number of points in the edges, and $T_r \in [0, 1]$ is the ratio of points in the edge classified as road, a user-selectable parameter that corresponds to the ratio of road points in the edge. The information of whether an edge belongs to a road or not is used later in the weight computation as long as classification information is available.

Roughness offers insights into the terrain's smoothness at a small scale. Factors such as bumps, vegetation, rocks, or minor obstacles contribute to increasing roughness, consequently raising the traversal cost in such areas. In [153], roughness is characterized by the standard deviation of LiDAR point heights relative to a designated surface. This surface is derived from a thin-plate spline interpolation applied to a previously rasterized map, utilizing the lowest point within each cell for interpolation. This approach yields a singular roughness value for each cell of the rasterized map.

In our work, roughness is defined similarly but can be computed using two distinct surface references. Firstly, the roughness of an edge is determined by calculating the standard deviation of the distances from the points it contains to the plane of the edge. Secondly, the roughness of each point on the map is evaluated by computing the standard deviation of distances from its neighboring points to the best-fitting plane formed by those neighbors. The distinction between the two methods is evident: while



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

one yields a single roughness value per edge similar to [153], our approach assigns a roughness value to each point in the scene based on its neighboring points, which allows for a more intricate analysis of terrain features. It is important to note that this increase in accuracy comes with a higher computational cost.

Once roughness is computed using any of the methods described, the values need to be converted into weight factors. For roughness calculated at the edge level, a function can be derived to directly translate the roughness value of each edge into its corresponding weight. However, for roughness calculated at point level, a descriptor is required to translate the roughness contribution of each point within the edge into a single weight value.

As far as our research indicates, there are no existing references about transit cost modeling based on terrain roughness. However, some empirical results were found. For example, traveling along rough ground surfaces (roughness $= 3.57 \cdot 10^{-2}$ m) reduces the travel rate by 19% [128]. Given that traversing non-rough terrains should be prioritized, the cost of traversing rough terrain should be greater than smooth terrain. Our proposal suggests assigning a weight of 1 when the roughness is 0, indicating completely flat terrain and increasing the weight for higher roughness values accordingly. Therefore, if the roughness weight function is denoted by $f_r(r) : \mathbb{R} \to \mathbb{R}$ for any roughness value $r$, it should satisfy the following condition:

$$\forall r \in \mathbb{R}_0^+ : f_r(0) = 1;\ f_r(b) > f_r(a)\ \forall\ b > a \tag{4.12}$$

It's important to note that when translating the contribution of roughness from each point into a single value, the result must remain independent of the number of points within the edge. In this scenario, the average roughness computed using the points of the edge is chosen.

The Normalized Difference Vegetation Index (NDVI) provides information about the presence of living vegetation in the points cloud since it tends to reflect the infrared spectra better than other objects. It can be computed only when near-infrared data NIR is available in the point cloud, and it is defined as:

$$\mathrm{NDVI} = \frac{\mathrm{NIR} - \mathrm{R}}{\mathrm{NIR} + \mathrm{R}} \tag{4.13}$$

where NIR is the reflectance of the near-infrared band, and $R$ is the reflectance of the red band. The range of possible values of the NDVI varies from $-1$ to 1. High and positive values of the NDVI show the presence of living vegetation, and values close to 0 and negative show its absence.

To the best of our knowledge, there are no studies that attempt to relate the NDVI index to a weight factor for the trafficability of a path so that we will follow a similar approach to that used with the calculation of roughness weights. The NDVI of each point on an edge will contribute to the total weight associated with the NDVI. The function used to map the NDVI value to a weight is as follows:



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

| Longitudinal Slope | Factor | Transversal Slope | Factor |
|---|---|---|---|
| $0.00 - 0.05$ | 1.0 | $0.00 - 0.15$ | 1.0 |
| $0.06 - 0.10$ | 1.5 | $0.16 - 0.30$ | 1.5 |
| $0.11 - 0.15$ | 2.5 | $0.31 - 0.45$ | 2.5 |
| $0.16 - 0.20$ | 3.5 | $0.46 - 0.60$ | 3.5 |
| $0.21 - 0.25$ | 5.0 | $0.61 - 0.75$ | 5.0 |
| $0.26 - 0.30$ | 6.0 | $0.76 - 0.90$ | 6.0 |
| $> 0.30$ | 10.0 | $> 0.90$ | 10.0 |

**Table 4.2:** Longitudinal and Transversal slope factors[119].

$$f_n(x) = \max(0, x), \ x \in [-1, 1] \tag{4.14}$$

where $x$ is the NDVI value for a given point. With the application of this function, only the positive NDVI values are mapped into a positive weight since they are the values that represent living vegetation.

Let $E = \{p_1, \ p_2, \ \ldots, \ p_n\}$ be the set of points contained in an edge. The weight of an edge related to the NDVI is defined as follows:

$$W_{ndvi} = \frac{1}{n} \sum_{i=1}^{n} f_n(p_i^{\text{ndvi}}) = \frac{1}{n} \sum_{i=1}^{n} \max(0, p_i^{\text{ndvi}}) \tag{4.15}$$

where $p_i^{\text{ndvi}}$ is the NDVI value of the $i$-th point and $f_n$ the function defined in Equation 4.14.

So, for every point of each edge, the NDVI is computed if the corresponding data is available. Alternatively, if NIR data are not available, it is possible to compute several vegetation indices by means of RGB information, as shown in [154], following a similar approach.

Considering all the edge features, a bi-directed, non-negative graph is built. In this thesis, the graph will be weighted with the following proposed cost function for each edge:

$$W = L \cdot P \cdot S_l \cdot S_t \cdot R \cdot N \tag{4.16}$$

where $W$ is the total weight of the considered edge; $L$ is its length, in meters; $P$ is the factor regarding the presence of roads; $S_l$ is the longitudinal slope factor; $S_t$ corresponds to the transversal slope factor; $R$ is the terrain roughness factor, and $N$ is the NDVI factor.

The computation of $L$ is straightforward as it corresponds to the Euclidean distance between two adjacent nodes. In order to quantify the advantages of traveling by roads,



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

| Roughness (m) | Factor |
|---|---|
| 0.00 | 1.0 |
| 0.05 | 1.05 |
| 0.15 | 1.15 |
| 0.25 | 1.25 |
| 0.35 | 1.35 |

**Table 4.3:** Roughness factors.

$P$ must meet the following conditions: $P = 1$ if the edge is not classified as road, and $0 < P < 1$ otherwise. The closer the value to 0, the easier it is to traverse a road edge.

Regarding the slope factors, $S_l$ and $S_t$, the criterion designed by [119] was chosen, and it is detailed in Table 4.2.

As mentioned before, $R$ refers to the weight of an edge in terms of its roughness, which is computed by Equation 4.12. Some example values are shown in Table 4.3.

The optimal path between two points is the one that minimizes the cost of traversing it. Thus, the edges with smaller values of $W$ will be more likely to be part of the solution. Regarding the parameter values of the Equation 4.16, $L \in (0, \infty)$. The other parameters are multiplicative factors that increase or decrease this value depending on the feature they quantify. The base value of each parameter is 1.0. The higher the value of a parameter, the more it will increase the overall value of $W$. The only factor besides $L$ that can have a value lesser than 1.0 is $P$ since its purpose is to reduce the weight of an edge if it is considered as part of a road.

### 4.2.1 Algorithms to obtain the optimal path

The optimal path between two points in a weighted graph can be computed by using several algorithms. Two of the more commonly used algorithms are the A* [155] and the Dijkstra's algorithm [156]. Unlike Dijkstra's algorithm, A* is a heuristic method where the user, besides having to compute the weighting of the graph, has to choose a heuristic function to estimate the cost of the path from a given node to the goal. The A* algorithm is optimal when the heuristic function is admissible, i.e., when it never overestimates the cost of reaching the destination. When using only edge length for weighting the graph, the straightforward solution is to select the Euclidean distance as the heuristic function, as it is the lowest possible length a path may have. Usually, the A* algorithm discards routes that could have a lesser overall weight when the routes are not directly oriented toward the destination. The advantage of the A* algorithm over Dijkstra's algorithm is that A* is considerably faster in most situations. Some variations of the A* algorithm have been proposed to overcome some of its limitations. The Iterative-Deepening A* (IDA*) [157] tries to



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

| | Dijkstra | A* | IDA* | D* | Anytime-D* | Bellman-Ford | Floyd-Warshall |
|---|---|---|---|---|---|---|---|
| Purpose | Single-source | Single-source | Single-source | Single-source | Single-source | Single-source | All-pairs |
| Negative Weights | No | No | No | No | No | Yes | Yes |
| Negative Cycles | No | No | No | No | No | Yes | No |
| Optimal Path | Yes | Heuristic | Heuristic | Heuristic | Heuristic | Yes | Yes |
| Time Complexity | $\mathcal{O}(\|E\| + \|V\| \log\|V\|)$ | $\mathcal{O}(\|E\| \log\|V\|)$ | - | - | - | $\mathcal{O}(\|V\| \cdot \|E\|)$ | $\mathcal{O}(\|V^3\|)$ |
| Space Complexity | $\mathcal{O}(\|E\| + \|V\|)$ | $\mathcal{O}(\|V\|)$ | - | - | - | $\mathcal{O}(\|V\|)$ | $\mathcal{O}(\|V^2\|)$ |

**Table 4.4:** Comparison of the different algorithms to compute the optimal path.

reduce the amount of memory consumed by A*. Instead of storing all the unvisited nodes, IDA* only stores the nodes of the current path. The Dynamic A* (D*) [158] algorithm is an incremental version of A*, where the path is recomputed every time a change in the graph is detected. This algorithm is suitable when the graph is expected to change over time. To achieve that efficiently, techniques like edge relaxation and data structures such as the priority queue are used to minimize the computational overhead of computing the new path. The Anytime-D* [159] is a variation of the D* algorithm that provides a sub-optimal solution at any time, and the solution is improved as the algorithm runs.

Other well-known algorithms are the Bellman-Ford's algorithm [160, 161], similar to Dijkstra's but capable of handling negative weights but no negative cycles, or the Floyd-Warshall's algorithm [162], which computes the shortest path between all pairs of nodes in a graph, even in the presence of negative cycles.

Since our goal is to provide the optimal path every time, A* and any of its variants are discarded. We also discard the Bellman-Ford and the Floyd-Warshall algorithms because they are slower than Dijkstra's due to their capability to manage negative weights and negative cycles, a feature we do not need in our case. Thus, Dijkstra is the selected algorithm to carry out the LCP search in this work. The goal of every algorithm and both the time and space complexities are shown in Table 4.4.

The implementation of Dijkstra's algorithm used in this work is shown in Algorithm 4.1. This implementation uses a priority queue, as introduced by Fredman and Tarjan [163], to always have $\mathcal{O}(1)$ access to the node with the smallest weight, instead of searching the node with the minimum weight in each iteration of the execution. This reduces the algorithm complexity from the original $\mathcal{O}(|V|^2)$ to $\mathcal{O}(|E| + |V| \log|V|)$. As shown in line 15 of the Algorithm 4.1, for each node, its neighbors are analyzed to update their weights. The data structure chosen to store the graph ensures that the neighbors of each node are stored consecutively in memory. This allows the loading of all the neighbors at once in the CPU cache, reducing the time to access the neighbors of each node, which is crucial to achieving the best performance of the algorithm.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Algorithm 4.1** Dijkstra algorithm

**Input:** $n_s$: start node
$n_e$: end node
$\mathcal{G}(V, E)$: graph
**Output:** `path`: Vector of nodes conforming the path

```
1:  procedure DIJKSTRA(n_s, n_e, G(V, E))
2:      pathFound ← 0                                          ▷ Path found flag
3:      V ← ∅                                                  ▷ Set of visited nodes
4:      T ← ∞                                                  ▷ Set of weights to the start node
5:      pq ← PriorityQueue()
6:      T[n_s] ← 0                                             ▷ Distance to the start node is 0
7:      V ← V ∪ {n_s}
8:      pq.push((0, n_s))                                      ▷ Push the start node to the priority queue
9:      while pq.empty do() = False
10:         w, n ← pq.pop()                                    ▷ Get element with lowest weight
11:         V ← V ∪ {n}                                        ▷ Nodes in this set already have definitive weights
12:         if n = n_e then
13:             pathFound ← 1
14:             break
15:         for n' ∈ G.neighbors(n) do                         ▷ For every neighbouring node
16:             if n' ∉ V then                                 ▷ The node weight must not be definitive
17:                 currNeightWeight ← w + G.distance(n, n')
18:                 if currNeightWeight < T[n'] then
19:                     pq.push((currNeightWeight, n'))
20:                     T[n'] ← currNeightWeight
21:                     cameFrom[n'] ← n
22:     if pathFound = 0 then                                  ▷ No path found
23:         return ∅
24:     path ← ∅                                               ▷ Path reconstruction
25:     n ← n_e
26:     while n ≠ n_s do
27:         path.push(n)
28:         n ← cameFrom[n]
29:     path.push(n_s)
30:     return path
```



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

### 4.2.2 Storing the graph in memory

Typically, the graph's topology information is stored using an adjacency matrix or an adjacency list. The first method is more efficient regarding time complexity, but it is unsuitable for big graphs since the memory requirements are $\mathcal{O}(V^2)$. In our case, where each node is only connected with its nearby neighbors, most of the elements of the matrix would be zero. The latter is a more memory-efficient method in terms of not wasting memory: each node stores a list of its neighbors. The drawback of this data structure is that since the location of each node in memory is unknown, the potential benefits of accessing elements located in nearby memory locations (cache locality) are not exploited. To overcome these issues, we opted for the well-known Compressed Sparse Row (CSR) data structure.

The CSR data structure stores a graph in a memory-efficient way and favors the exploitation of cache locality. As advanced in Section 4.2.1, our implementation of the selected LCP algorithm vastly exploits this data structure.

The data structure we propose is composed of three arrays ($A$, $B$, $W$). $A$ is an unsigned integer cumulative array, defined as $A_0 = 0$ and $A_i = A_{i-1} + N_i$, where $N_i$ is the number of neighboring nodes of the $i$-th node. By using this array, the number of neighbors of the $i$-th node can be retrieved as $A_{i+1} - A_i$. Note that $A$ has exactly $|V| + 1$ elements. The array $B$ stores the IDs of the neighbors of each node. Thus, the $j$-th neighbors of the $i$-th node is retrieved a $B_{A_i+j}$. Finally, the array $W$ stores the weights of the edges. The weight of the $j$-th edge coming out of the $i$-th node is retrieved as $W_{A_i+j}$. Both $B$ and $W$ have $|E|$ elements. The total size of the graph using this data structure is $2|E|+|V|+1$. The primary limitation of this data structure lies in its requirement for complete reconstruction whenever modifications are made to the graph. Given that the structure is formed only after edge computations and weight assignments, this drawback may be overlooked.

Usually, the number of edges is much greater than the number of nodes. Let's assume a graph such as $|E| < 2^{32} - 1$. Both the $A$ and $B$ arrays will be stored as 32-bit unsigned integers, and the $W$ array will be stored as 32-bit floating point numbers. Assuming an average of 6 edges per node, when the Dijkstra's algorithm is checking the total weight to the neighbors of a given node, $6 \times 32 \times 2$ bytes are loaded into the cache since both the IDs of the neighbors and the weights are required. The reduced size achieved by the CSR data structure allows for big chunks of the graph to be completely stored in the cache of current computers, which is a key factor in the performance of Dijkstra's algorithm.

## 4.3 Examples and results

The datasets selected for testing our pathfinder proposal are airborne LiDAR point clouds obtained from both public and private repositories. The datasets containing infrared information correspond to the Spanish region of La Rioja, and they were



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

provided by the Spanish Geographic Institute and the Government of La Rioja, in the frame of the National Plan of Aerial Orthophotography (PNOA). These clouds have a point density of 3 pts/m$^2$. Furthermore, a more dense set of point clouds that do not include infrared information were used, whose average density is 15 pts/m$^2$, and were privately provided by Babcock International [164]. These point clouds correspond to rural areas surrounding the city of Lugo (Spain). The orthomosaics of these areas are shown in Figure 4.5.

The classification of the scenes into the classes vegetation, ground, buildings, and roads was carried out by using an improved version of the point cloud classifier described in [165], which includes the methods described in [166]. Several variations of the segmentation parameters $N_s$ and $R_s$, described in Section 4.1, are tested for each point cloud to obtain different edge mean lengths, group sizes, and number of neighbors per group. Therefore, different configurations of the trafficability graph can be generated. Default values for these experiments were defined as follows. A value for the ratio of points in the edge classified as road $T_r$ of 0.9 is used to determine whether an edge is considered part of a road. Also, $T_t = 0.9$ is used as the Trafficability Threshold to determine the trafficability of an edge. Finally, edges with longitudinal slopes greater than 0.30 are considered non-trafficable. Users can easily tune these parameters according to the features of their scenes.

We compare grid-based methods with our proposal in order to show the advantages of not using predefined cells to divide the point cloud. A study of the influence for each of the weights included in Equation 4.16 was performed as well.

### 4.3.1 Point cloud segmentation

As stated initially, the already classified point cloud is segmented to obtain groups to build a trafficability graph. In this graph, each group is represented by a node. Eventually, the groups might contain even only one point, yielding a graph where each node corresponds to one point in the cloud. This section analyzes the advantages and disadvantages of this situation.

The choice of segmentation parameters, described in Section 4.1, will depend on the average point density of the point cloud. Let be $\rho$ the average point density (in pts/m$^2$). A given neighborhood around a seed will have an average of $\rho R_s^2$ points since we are using square-shaped neighborhoods. So, depending on the chosen values of these parameters, certain combinations will yield identical segmentations of the scene. Furthermore, these parameters will indirectly rule the length of the edges of the graph. To show results, a test case with the following combinations of these two parameters is considered:

$$\begin{aligned} R_s &= \{1.0,\ 2.5,\ 5.0,\ 7.5,\ 10.0\}\ \text{m} \\ N_s &= \{100,\ 500,\ 1\,000,\ 2\,500,\ 5\,000,\ 7\,500,\ 10\,000\}\ \text{points} \end{aligned} \tag{4.17}$$



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

(a) Surroundings of Lugo.

(b) Surroundings of Lugo with a road present.

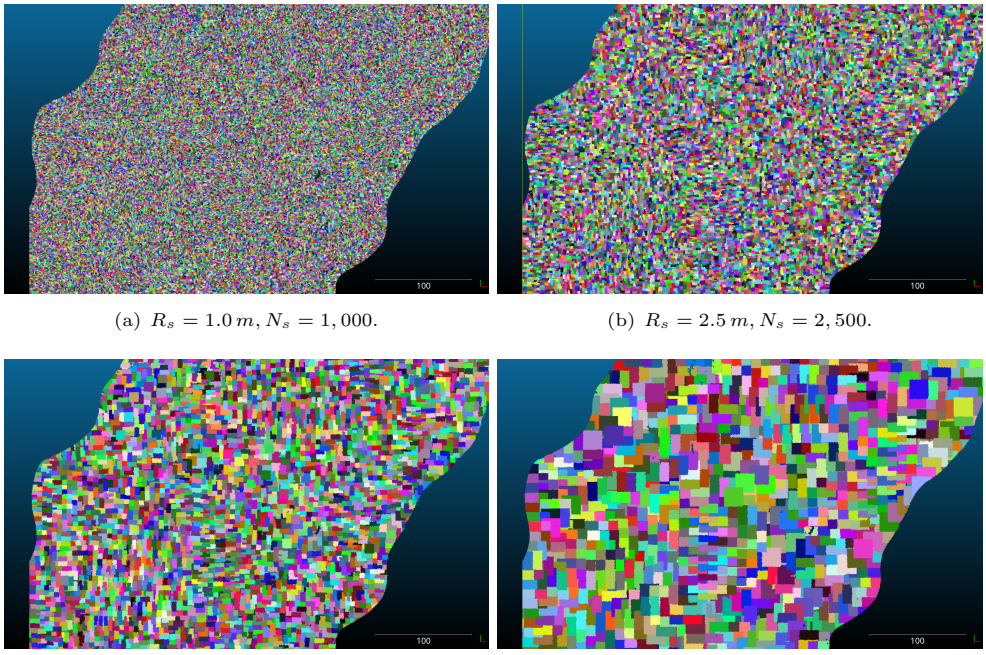
(c) La Rioja dataset.

**Figure 4.5:** Orthomosaics of the areas used in the different tests.

So, by exploiting all the combinations, 35 segmentations were generated. Figure 4.6 shows some of the most representative. The results show that the segmentation groups are more deformed as their size increases. Since the position of the nodes in the graph is computed by using the centroids of every group, the presence of concave groups is a problem since the computed centroid may be, in fact, outside the group. To avoid



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(a) $R_s = 1.0\,m, N_s = 1,000$.

(b) $R_s = 2.5\,m, N_s = 2,500$.

(c) $R_s = 5.0\,m, N_s = 2,500$.

(d) $R_s = 10.0\,m, N_s = 10,000$.

**Figure 4.6:** Segmentations generated with different combination of the parameters $R_s$ and $N_s$. The scale shown in the bottom right corner of each map is given in meters.

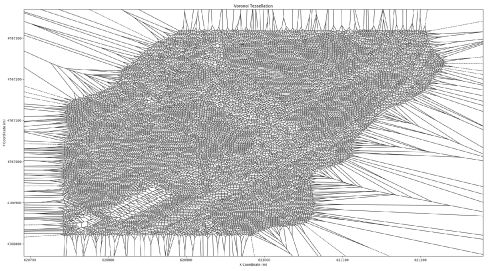
this issue, we choose as default values $R_s = 2.5$ m and $N_s = 2,500$ points, which yield a compact and uniform segmentation, where all the groups are convex.

### 4.3.2 Graph construction

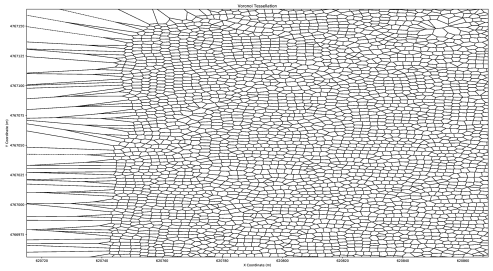
As an example, the Voronoi Tessellation of the segmentation in Figure 4.6(b) ($R_s = 2.5$ m, $N_s = 2,500$) is shown in Figure 4.7(a). In this scene, there are 20 460 Voronoi Cells (graph nodes) and 121 742 total neighbors (graph edges), yielding an average of 6 neighbors per node. As previously established, the outer cells must be treated as special cases. Figure 4.7(b) shows a detailed view as an example of this situation. Due to the lack of points outside the point cloud limits, the Voronoi cells tend to be very large, producing poor quality neighboring information because cells whose centroids are far apart in the scene are considered neighbors due to the very long edges that maintain contact with several cells. A more detailed example of this phenomenon is shown in Figure 4.7(c), where the graph resulting from the Voronoi Tessellation is



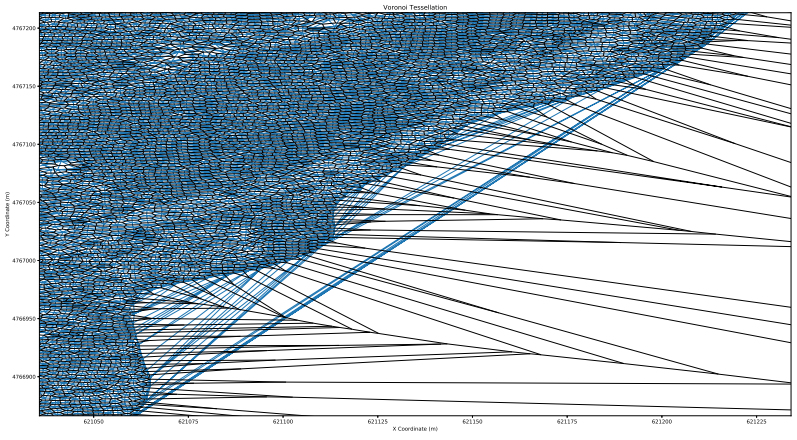
USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(a) Voronoi tessellation of a given point cloud.



(b) Detail of the Voronoi tessellation.



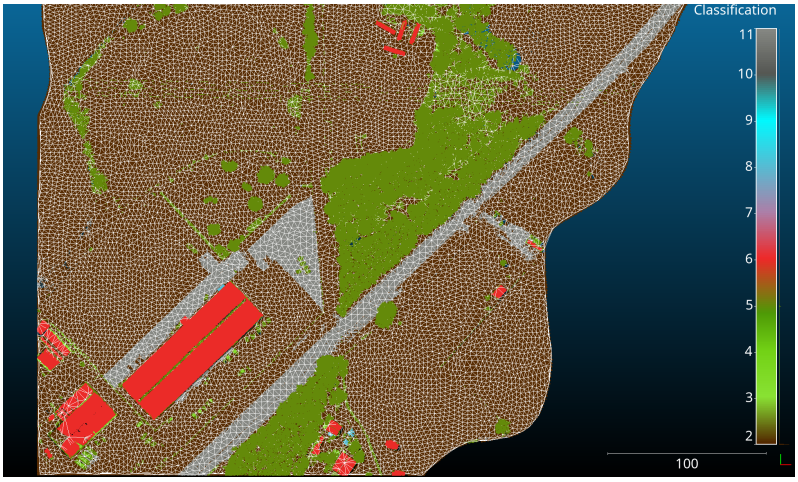

(c) Detail of long edges in the borders of the point cloud in the Delaunay triangulation.

**Figure 4.7:** Voronoi tessellation of a point cloud.

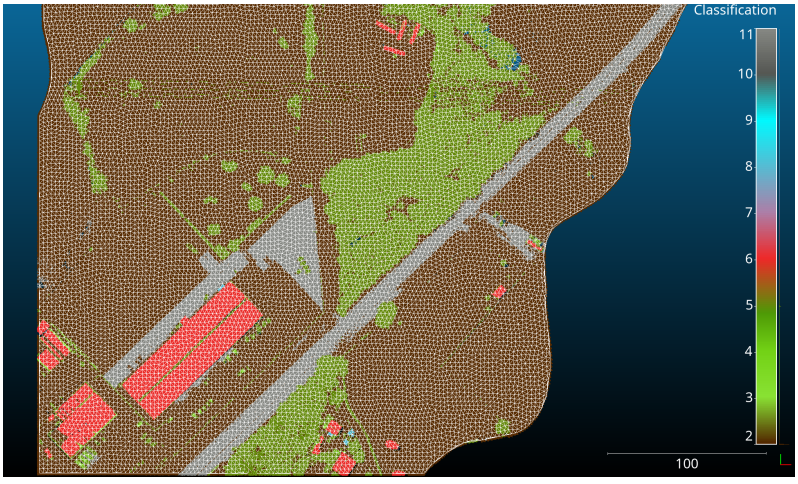
shown. As expected, there are very large edges in the borders that must not be taken into account in the next stages. This effect is magnified when the shape of the borders in the scene is convex. As the size of the inner cells is almost homogeneous, the length of the inner edges is also quite regular, so removing the spurious border edges with misleading information is straightforward with this information. An example of the resulting graph over a classified and a non-classified point cloud is shown in Figure 4.8. This graph now sticks perfectly to the graph borders. Also, the graph in Figure 4.8(a) is built only over groups of segmented points classified as road or ground, ignoring buildings (in red) and vegetation (in green). The case ignoring the classification is



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA




(a) Graph with classification data.



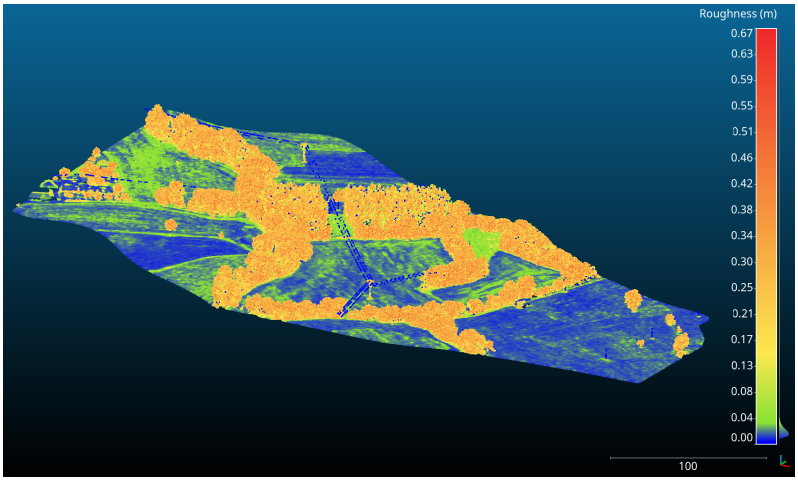

(b) Graph without classification data.

**Figure 4.8:** Resulting graph using the neighboring information of the Voronoi Tessellation.

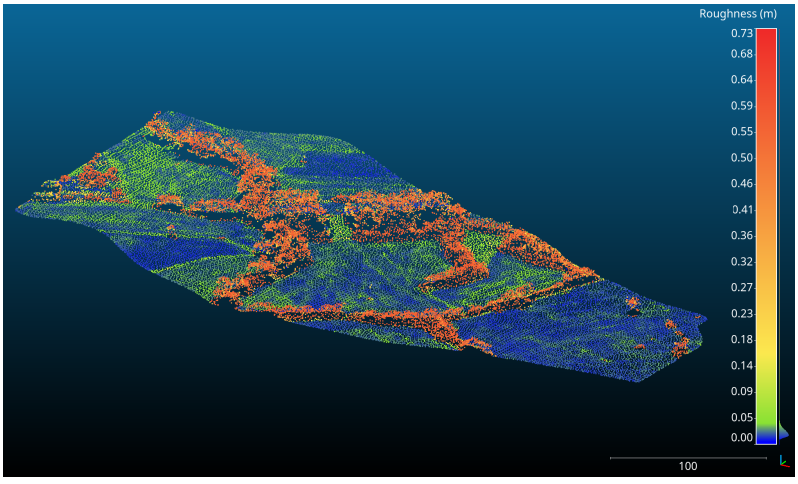
shown in Figure 4.8(b).



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA




(a) Roughness computed at point level.




(b) Roughness computed at edge level.

**Figure 4.9:** Surface roughness.

### 4.3.3 Slopes

The slope is directly computed over the edge linking two nodes in the graph by using the totality of the points in that area. This method allows the extraction of the actual slope of the terrain in addition to the transversal slope, which is also important to



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

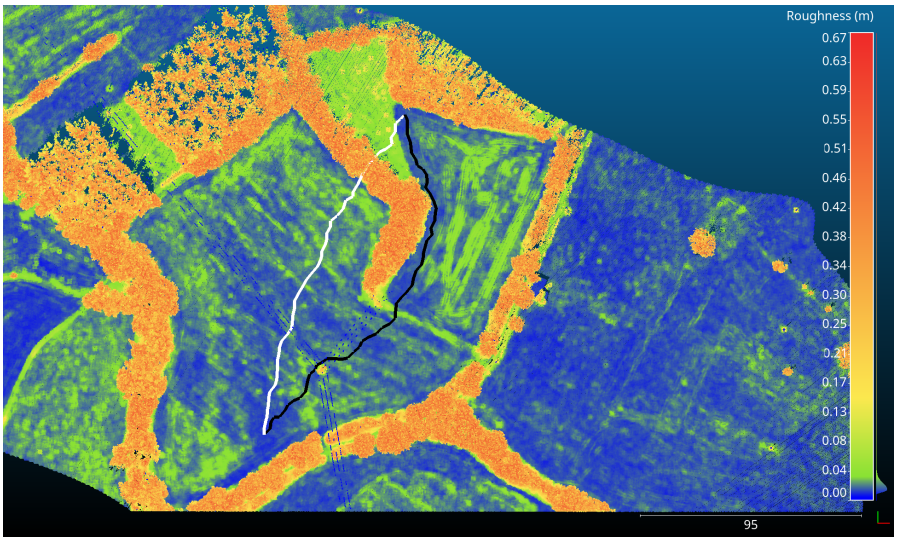



**Figure 4.10:** Example of paths computed with and without considering the roughness. The path with the white color corresponds to that computed without including the roughness in the weight computation, and the black path is the one including it.

the weight computation of the given path. Note that, for each node, $2n$ slopes are computed, where $n$ is the number of edges going out from the given node and towards its neighbors.

Also, each edge is treated as an individual entity, holding a certain number of 3D points. Note that one point may belong to more than one edge because of the overlap that could appear in the proximity of the nodes. This is not an issue since the same point can be used several times in different planes.

### 4.3.4 Roughness

As discussed in Section 4.2, roughness can be computed at edge level or at point level. The difference in accuracy between the two methods is significant, but so is their computational cost, which will be discussed later.

An example of the results of roughness, both at edge and point level, is shown in Figure 4.9. As can be observed, the roughness intervals are very similar in both scenes: $(0.00, 0.67)$ m for point-level roughness and $(0.00, 0.70)$ m for edge-level roughness. The roughness values in the interval $(0.0, 0.1)$ m correspond to points or edges that belong to the ground and which will be used later in the search for paths. The higher roughness values correspond to points or edges that lie on the edges of buildings or



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

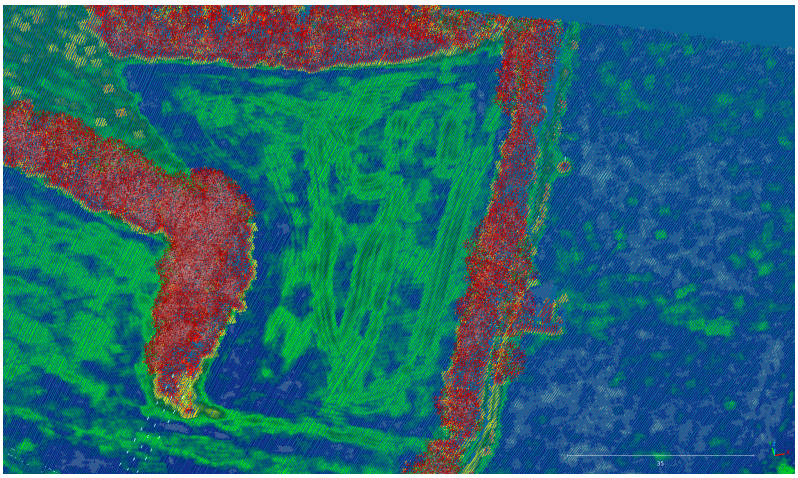


**Figure 4.11:** Detail of tire tracks in the ground.

on high vegetation. Note that these values are more widely distributed when the calculation is at point level than when it is at edge level.

Paying attention to the ground areas, the distribution of roughness is very similar in both cases, obviating the evident loss of precision.

For testing purposes, the relationship between roughness value and weight was uniformly defined as $f(r) = 1 + r$ across all cases, as it meets the criteria imposed by Equation 4.12. This weight definition causes the path to go through less rough areas (blue) rather than rougher areas (green). Figure 4.10 shows an example of the difference between including and not including the roughness in the weight computation. Note how the effect of taking into account the roughness in the algorithm has two effects: On one hand, the black path lies in areas in which the roughness is the lowest; on the other hand, since the obstacles are expected to have a high roughness value (for instance, high vegetation, represented mostly in red in Figure 4.10), the inclusion of the roughness in the computation allows the algorithm to avoid obstacles even if they are not classified as such. In Figure 4.11, the detail of how the roughness computation highlights the presence of tire tracks in the ground is shown. The pathfinder avoids these areas since the roughness is higher than the average roughness of the ground.

### 4.3.5 NDVI

The NDVI can only be computed at the point level. If the point cloud is not already classified, the NDVI is useful, when available, to avoid areas with the presence of crops or low vegetation. Also, it is useful to force the cost of the path to be the lowest when



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

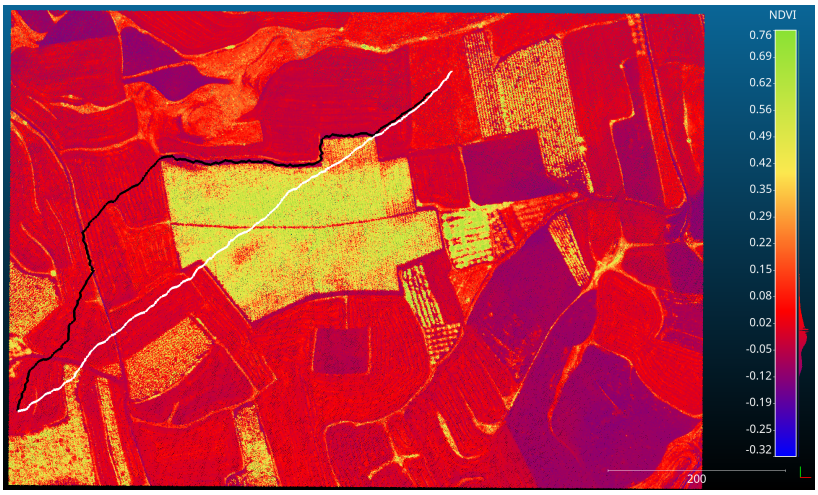



**Figure 4.12:** Path found using the weights given by the NDVI values. In white, the path found without using NDVI values is shown. In black, the same path is found using the NDVI values of each edge.

using human-made roads since these roads have a very low NDVI value.

An example of the behavior of the Dijkstra's algorithm when searching paths taking as weight factors both the NDVI and the length of an edge is shown in Figure 4.12, where two paths are depicted. The white path has been calculated taking into account only the length of the edges as a weight factor, while in the black path, the NDVI values have also been taken into account. Note that the white path follows an almost straight trajectory from the starting point to the end (as straight as the built graph allows), crossing the crop in the central part of the map. However, when NDVI values are considered in the cost function, the path found, in black, determines that it is better to go around the crop to reach the target. Furthermore, the behavior explained above is observed: human-made roads and paths tend to have a lower NDVI value than raw land and crops, so the optimal path will prioritize roads and paths, as they will have a lower total weight than their surroundings. This effect can be observed in the left side of Figure 4.12, where part of the black path runs along a road.

## 4.4 Computational performance

Two tests were carried out to evaluate the computational performance of our algorithm using a C++17 implementation. La Rioja maps were used in both tests due to the availability of the infrared information needed in one step of the algorithm.

The first test (Test A) was carried out in a map of 14 076 958 points and 4 $km^2$,



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

with a density of 3.52 pts/m$^2$. The analysis of the algorithm running on this map was performed using the Machine 4. The second test (Test B) was performed over a larger map, containing 54 850 074 points for a total area of 16 km$^2$ and a density of 3.43 pts/m$^2$. This experiment was carried out on the Machine 1. In both cases, the optimization flag was -O2.

Regarding the parallelization strategy, a shared memory model has been chosen, and mainly, the OpenMP v4.5 [167] library was used.

Shared-memory parallelization models are very suitable for accelerating computations on a large number of entities. The main advantage of this model is that it is not necessary to modify the code already written to execute it in parallel. So it is easy to parallelize at the node, edge, or point level. On the other hand, this model limits the size of the maps to be used to those whose size does not exceed the computer's memory.

In this case, it is possible to parallelize almost the entire code, except for the graph creation algorithm (which involves segmenting the scene and calculating the Delaunay triangulation) and searching for the optimal path between two points because of their inherent dependencies. It has been experimentally proven that the execution time of these two algorithms represents a small percentage of the total time. Amdahl's law can be applied to evaluate the maximum achievable speedup as a function of the number of threads used. This law can be formulated as follows:

$$S = \frac{1}{(1-p) + \frac{p}{n}} \tag{4.18}$$

where $p$ is the percentage of time consumed by the parallelizable parts with respect to the total execution time using only one thread, $n$ is the maximum achievable speedup, i.e., the number of threads used in the execution, and $S$ is the maximum achievable speedup in the whole execution taking into account the non-parallel parts of the code.

Another useful descriptor that shows the quality of the parallelization is the efficiency, defined as the quotient between the achieved speedup and the number of threads used:

$$E = \frac{S}{n} \tag{4.19}$$

where $n$ is the number of threads.

Two parallelizable tasks constitute the main bottleneck of the execution. On one side, the creation of every edge of the graph implies the computation of both the points that fit inside them and the best-fitting plane. On the other hand, the roughness computation at point level implies the determination of the points surrounding every point in the scene and the computation of the best-fitting plane as well. The speed of the neighborhood computation is directly related to the density of the local area where the point is located. Since in ALS point clouds the density may have significant variations, the *dynamic schedule* strategy of OpenMP is selected to avoid an unbalanced



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

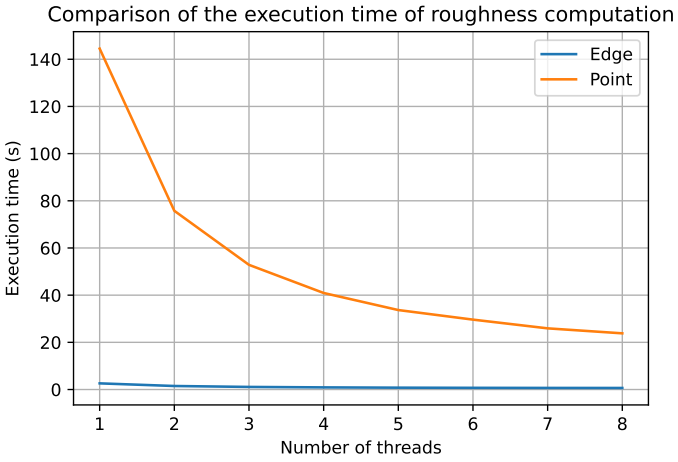



**Figure 4.13:** Execution times in the cases when computing the roughness at edge level and at point level.

parallelization. So, the loop iterations are dynamically distributed among the threads to ensure that every thread picks up new chunks of work as they become available. This approach is suitable when the workload of each iteration is unpredictable, as this is the case.

As already mentioned in Section 4.2, roughness can be computed at edge or point level, being both computations mutually exclusive. The cost of computing the roughness at edge level is low compared to the computation at point level, as shown in Figure 4.13, but also implies a loss of precision. For this reason, the scalability analysis of the algorithm was performed considering the computation of the roughness at point level.

### 4.4.1 Results on Test A

The achieved speedup per number of threads in Machine 4 is shown in Figure 4.14. In the case of 8 threads, the achieved speedup is 4.81 while the theoretical maximum speedup is 6.05, yielding an efficiency of 0.8.

A parameter that strongly influences the achieved speedup is the problem size. In fact, when the algorithm is applied over a very small point cloud (in the order of tens of thousands of points), the overhead introduced due to the parallelization mechanisms eventually can not pay off the obtained gain of time. This effect is clearly depicted in Figure 4.15. Note that when reaching a problem size of 100 000 points approximately, the overhead introduced starts to be compensated with the speedup obtained. It can



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

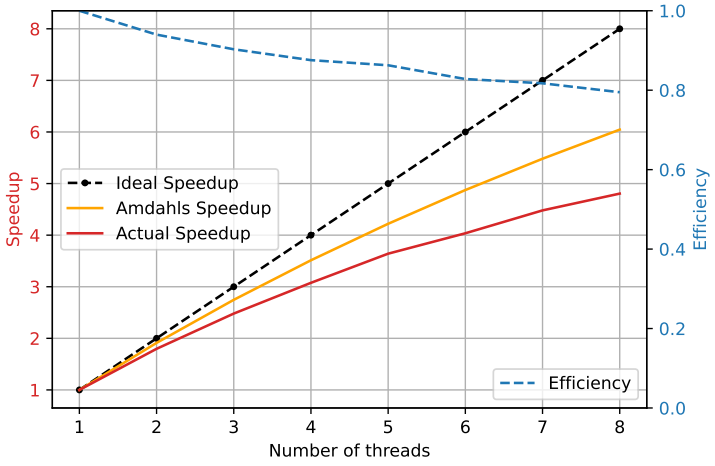



**Figure 4.14:** Speedup and efficiency achieved on Test A. Problem size: 14 000 000 points.

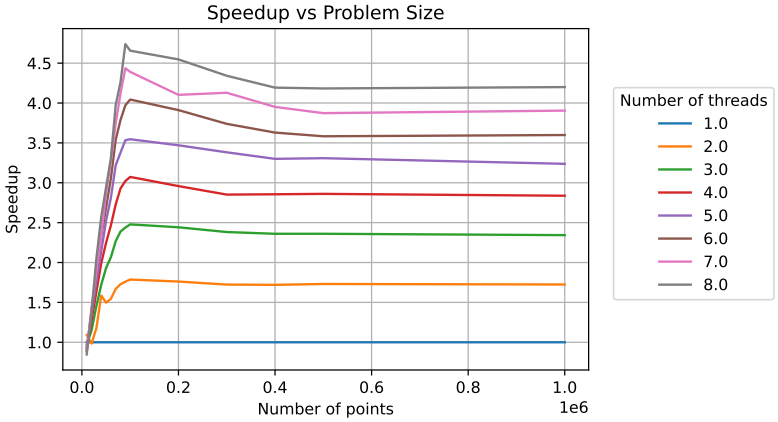



**Figure 4.15:** Achieved speedup as a function of the problem size, given in number of points, for Test A.

be observed as well that the speedup scales quickly and then remains almost constant from that point on.

Finally, the combined effect of the problem size and the number of threads can be expressed through an isoefficiency matrix [168], as depicted in Figure 4.16. Note that the efficiency scales well both with the number of threads used and the problem size,



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

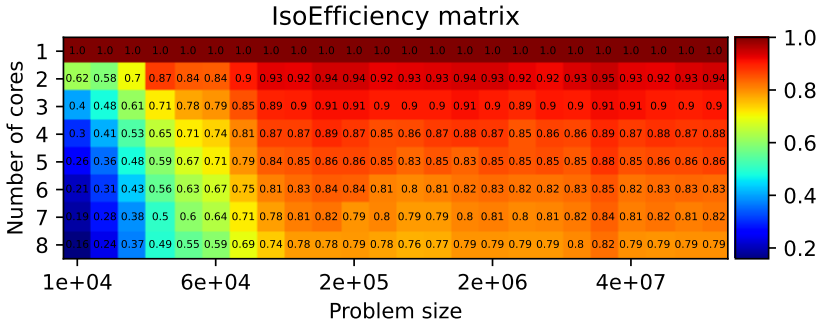



**Figure 4.16:** Efficiency matrix for Test A.

remaining almost constant when the problem size is big enough. This means that the parallelization performed presents a good scaling.

### 4.4.2 Results on Test B

In this case, the Machine 1 was used to carry out the study. The speedup achieved per number of threads is shown in Figure 4.17. The behavior is similar to the one shown in Test A when the number of threads in both cases matches. Beyond that, the speedup keeps growing parallel to the theoretical speedup, keeping the efficiency in every core at a value of around 0.75.

In order to analyze the influence of the problem size on the achieved speedup, different numbers of points were considered. The results shown here were computed as the median of 30 different executions in the same conditions. In Figure 4.18, the isoefficiency matrix is shown. In this case, the efficiency is close to 1 when the number of threads is low. As the number of threads increases, the efficiency drops until it remains almost constant at a value around 0.75. The values of the efficiency are greater than 1.0 in the first column of the isoefficiency matrix for a very specific number of threads. This effect is known as superlinear speedup [169] and can be observed in highly parallel codes when the size of the problem divided by the number of threads is less than the size of the available cache in each processor.

## 4.5 Discussion

In this section we remark the advantages of our approach compared with the state-of-the-art methods, specifically the utilization of all the information provided in the point cloud, and the analysis of the trafficability of the edges in the graph.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

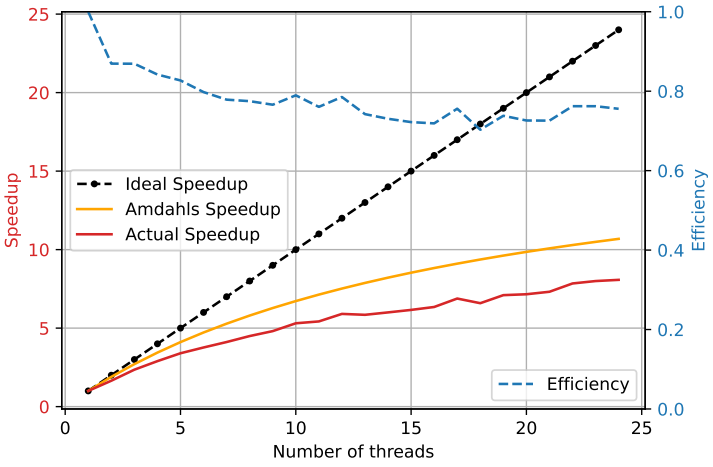



**Figure 4.17:** Speedup for Test B. Problem size: 40 000 000 of points.

We have described several properties used to compute the weight of each edge: the length, the slopes, the roughness, and the NDVI, if applicable. It is important to remark again that we compute these properties using every point in each edge instead of using discrete values over rasterized cells, so there is no loss of information associated with a rasterization of the scene. When a DTM is used to find the optimal path, groups of tenths or even hundreds of points are reduced to a single value: the one of the cell where the points lie. For the sake of comparison, we have implemented the DTM described in [170] to highlight the advantages of our method.

### 4.5.1 Number of points considered

In [118], an area of $9 \times 12\text{km}^2$ is represented by a Digital Elevation Model (DEM) of 50 m resolution. Therefore, each cell has an area of $2\,500\,\text{pts/m}^2$. This implies that using a point resolution of only $1\,\text{pts/m}^2$ (which is a very pessimistic estimation) is enough to condense the information of 2 500 points in a single height value. This is summarized in reducing a point cloud containing 108 000 000 points in a DEM holding the information in only 43 200 points, losing up to 99.96 % of the information.

A similar issue occurs when computing the roughness weight factor. In [153], a DEM of 5 m resolution, where each cell contains 5 to 50 data points, is used to compute both the slope and the roughness of the terrain. The roughness is then computed as the standard deviation of the height of each point over an interpolated surface, whose resolution is the same as the DEM. By using an interpolation, the roughness is actually being computed over a virtual surface instead of a real surface defined by the points.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

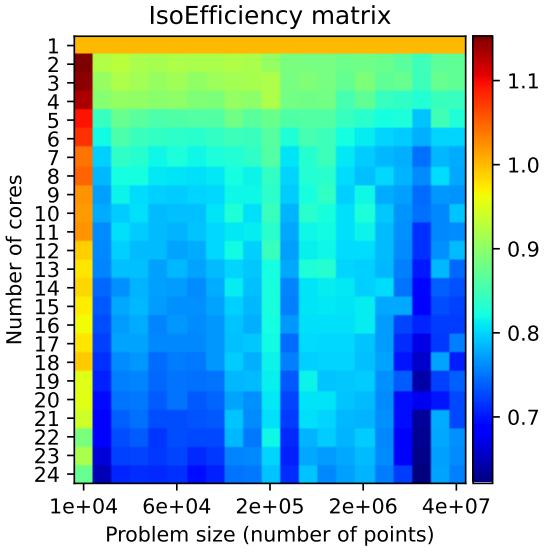



**Figure 4.18:** Efficiency as a function of the number of threads and the problem size for Test B.

A similar approach is used in [128], defining a fine-scale DTM of 0.25 m of resolution instead of actual points in order to compute the roughness. In our work, the roughness is also computed using the information of the whole available set of points.

### 4.5.2 Trafficability of the path

When computing the cost of traversing a path between two points, it is important to have enough knowledge of the terrain to determine if a given agent (person or vehicle) could traverse that path. As far as we know, no authors have presented that kind of analysis to date.

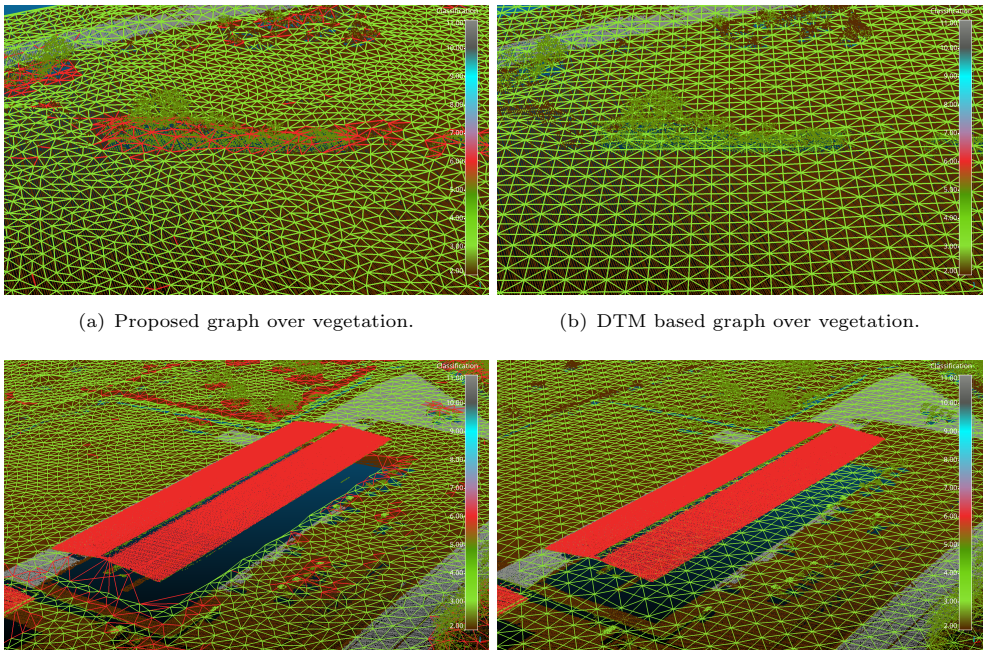
In this work, and only when the point cloud is previously classified, the trafficability graph is computed by using the expression given in Equation 4.2. The application of this equation splits the edges between trafficable and no trafficable, and another trafficability analysis based on the slope will be carried out later.

For example, in Figure 4.19, the differences between our graph and a DTM-based graph are shown. The pictures focus on a little area of isolated vegetation, including a building. The trafficable edges are shown in green, and the non-trafficable ones are in red. In Figure 4.19(a), it can be seen that the vegetation is non-trafficable. However, that same area is trafficable in the graph based on the DTM information.

Also, no previous work details the issues when a building is encountered in the



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(a) Proposed graph over vegetation.

(b) DTM based graph over vegetation.

(c) Proposed graph over a building.

(d) DTM based graph over a building.

**Figure 4.19:** Trafficability graph over different areas.

scene, as in Figure 4.19(c) and Figure 4.19(d). The DTM is able to compute the terrain level below the roof of the buildings (Figure 4.19(d)), but then those edges are incorrectly managed as trafficable. In our method (Figure 4.19(c)), since the nodes and neighboring information are computed from a segmentation carried out over already classified road and ground points, this problem does not appear.

Another parameter influencing trafficability is the slope. In our work, edges whose slope is greater than $S_t$, the slope threshold, are considered non-trafficable since the algorithm aims to find a route that can be traversed without much effort or special equipment. Table 4.2 in Section 4.2 shows several weights for longitudinal and transversal slopes. Slopes greater than 30% (16.70°) are weighted with a very high cost but still could be traversed. In cases where the only possible path includes a very steep slope, the model described in [119] returns a very costly yet possible route. In our work, edges whose slope is greater than $S_t = 36.4\%$, which corresponds to 20°, are removed from the graph. Thus, in the case that classification information is not available, the roof of the building would not be trafficable either because the edges joining the ground and the roof would be deleted due to the slope threshold, as can



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

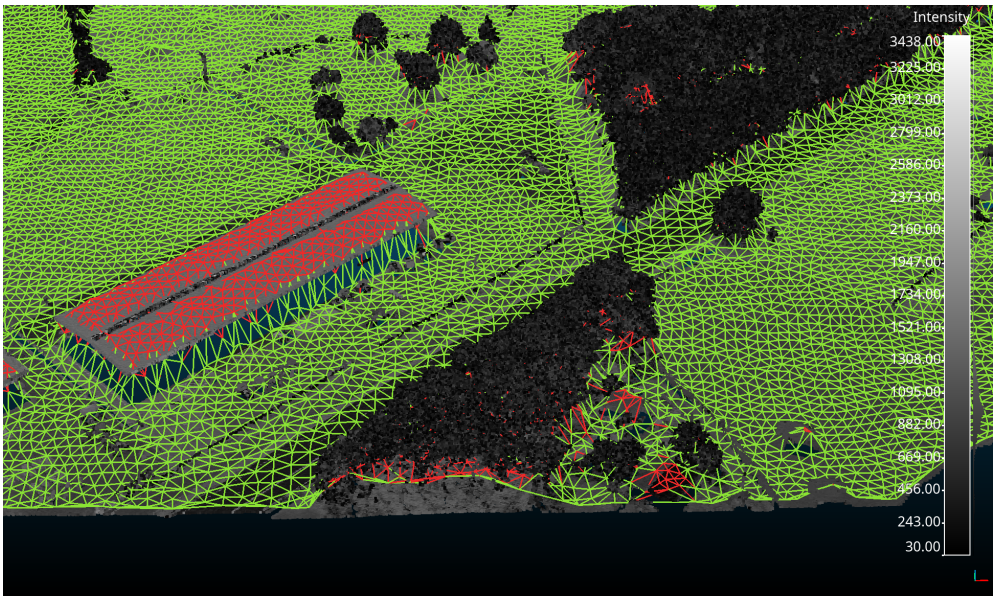



**Figure 4.20:** Trafficability graph over a non-classified building.

be seen in Figure 4.20.

## 4.6 Conclusions

In this chapter, a model to find the optimal path in terms of transitability between two points has been presented. The advantages of this model are, on the one hand, the use of Dijkstra's algorithm with an efficient implementation, which guarantees that the route found is the optimal one in terms of cost among all the possible solutions and, on the other hand, the use of the entire point cloud when calculating the cost functions, avoiding the rasterization of the scene by means of a DTM with the consequent loss of accuracy. Unlike the works where a DTM is used as a basis for path planning, in this work, the presence of obstacles, such as rocks, bumps, walls, or high vegetation, is taken into account thanks to the computation of the roughness at point level. Also, if the input point cloud is classified, the individual trafficability of each edge can be computed based on the classification of the points contained in every edge. In addition, two novel methods have been proposed for defining the cost of traversing a path based on two different properties: the NDVI value and the roughness value of the terrain. With this, the possibility of picking the paths along roads and tracks has been



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

achieved, even when the scene is not classified. The codes to execute this approach were efficiently parallelized on shared memory multicore systems using OpenMP. A novel study of the computational costs for such algorithms was carried out, which demonstrates the good scalability of the algorithm in manycore systems.

The contributions of this chapter were published in *Elsevier* and were extracted from:

- M. Yermo, F. F. Rivera, J. C. Cabaleiro, D. L. Vilariño, and T. F. Pena, "A fast and optimal pathfinder using airborne LiDAR data", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 183, pp. 482–495, 2022, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2021.11.014.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# Chapter 5

# Powerline detection and characterization

*Nature always tends to act in the simplest way.*
— Daniel Bernoulli, *Mathematician and Physicist.*

Nowadays, access to electricity is a fundamental and essential need for virtually the entire world's population. With the rise of remote work, even minor power outages can be a significant problem for individuals and companies. For this reason, it is necessary to ensure the correct operation of powerlines, both the high-voltage powerlines responsible for transporting electricity over long distances and the low-voltage powerlines responsible for the final energy delivery to the infrastructures.

Powerline inspection is a task that must be carried out periodically to ensure the safe and correct operation of the entire electrical network. In particular, one of these tasks is to inspect the surroundings of powerlines (both high- and low-voltage) and remove surrounding vegetation and obstacles that may be hazardous. Since on-site inspection can be costly in terms of man work, time, and financial resources, as well as dangerous and error-prone, the use of automated or remote sensing techniques has spread quickly to perform such tasks. One of the first remote sensing techniques used for powerline inspection is SAR. In particular, the use of data obtained from SARs for identifying powerline towers is proposed in [171]. Another possibility for powerline inspection is using optical satellite imagery, which is used for coarse-grained inspections, i.e., detecting potentially hazardous vegetation around a known powerline [172, 173]. Other techniques based on optical imaging are the image-based inspection methods [115, 174], where powerline structures (usually high-voltage) are extracted from aerial images. More specifically, most image-based techniques focus on photogrammetry since the 3D information it provides is of great importance when detecting and characterizing obstacles. In this line, different proposals exist to compute distances in 3 dimensions using images [175, 176]. Additionally, the study of the feasibility of using

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

hyperspectral imagery together with LiDAR data has been carried out in [177], where it is shown that the inclusion of LiDAR data significantly increases the accuracy of obstacle classification.

In the past few years, the adoption of robots for powerline inspection has become increasingly common. These robots are specifically built to utilize the powerline conductors as tracks, allowing them to move along and examine the powerline infrastructure. Nevertheless, there are challenges associated with positioning the robot on the line and ensuring it can navigate around obstacles, which can hinder the inspection process [178].

The reduced costs of both UAVs and LiDAR sensors, coupled with their versatility and the superior precision offered by LiDAR compared to other remote sensing methods, have resulted in widespread adoption of LiDAR solutions over alternative options [115, 179].

The current state-of-the-art 3D point cloud analysis for powerlines primarily revolves around two main areas of interest. Firstly, there is a strong emphasis on powerline inspection, which involves identifying potential problems along a given powerline, such as damaged conductors or hazardous vegetation nearby. Secondly, considerable efforts are dedicated to developing techniques for reconstructing and characterizing the powerlines themselves so the distances of nearby objects can be computed accurately with respect to the conductors themselves.

For the case of powerline inspection, Nardinocchi et al. [180] present a fully automatic method. In this work, the point cloud is rasterized using an arbitrary cell size over a data set obtained from a flight over a high-voltage powerline. At this point, the highest point inside each cell is considered a powerline conductor. Then, after selecting a seed point, those points showing a height difference smaller than a given threshold are added to a wire list and become the seed of the next iteration. Once the wires are classified, several cylinders with different sizes are defined transversally to the detected wires. Points inside these cylinders are considered potentially dangerous to the powerline. The authors report an agreement of 97.53% in classifying wires and pylons when comparing their results with those obtained with TerraScan. Regarding the computational efficiency of the algorithm, a naïve estimation of the powerline length that can be processed as a function of the grid size is also proposed.

A different approach to powerline inspection is presented in [181]. In this work, a flight made on purpose over a powerline is used to demonstrate the results of the developed algorithm. The algorithm consists of several stages: first, the extraction of the ground points using the cloth simulation filtering algorithm. After that, a voxelization of the remaining points in voxels of $5 \times 5 \times 5$ m is performed. They use the geometric properties of a hanging wire to determine a seed voxel that for sure contains powerline points. The neighbor voxels are analyzed from that seed and labeled as powerlines if their absolute slope difference is below a fixed threshold. In this case, the analysis of dangerous objects near the powerline is carried out by computing the distance from potentially risky points to the concave hull of the detected powerlines.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

The reported qualities for classifying powerlines and pylons are 94.63% and 67.36%, respectively.

In [182], Huang et al. introduce a method to compute the distance between a given powerline and its surrounding obstacles. As in the approaches described above, the point clouds are obtained with a specific flight over a powerline, and the data are rasterized on the XOY plane. The cell size used in the rasterization is obtained from the pylon planimetric footprints. Therefore, prior knowledge of the powerline's existence and characteristics is needed. Using the vertical distribution of the points inside a given cell, they determine whether the points in a given cell belong to an object, a pylon, or a conductor. Then, each electric conductor is segmented individually using a projection of the points in a plane perpendicular to the direction of the conductors. Once the conductors are segmented, the tower locations are obtained using the direction of the conductors as a guide. Finally, with the powerline completely segmented, distances to the nearest obstacles are computed to determine if they are a potential hazard. Regarding the performance analysis of the algorithm, the authors provide measurements of the number of distances per second that can be computed with a given system.

Regarding the reconstruction of powerlines, an image-based model is proposed in [183]. This work presents a method to classify conductors, pylons, insulators, and shield wires. A dataset of 48 point clouds is used to validate the application of the algorithm. Nevertheless, the 48 presented point clouds are sections of the same 80 km length and 80 m width powerline. The data was acquired with specific flights over the powerlines. A recall of 94.90% for towers and 99.58% for conductors is obtained from a study on 25 datasets. Regarding the computational performance of the method, one reference to the classification method used in [184] was found.

Chen et al. [185] introduce another method that combines the powerline's modeling and anomaly detection. The data is acquired by flying a UAV along a high-voltage powerline. The extraction of the powerline is carried out using a pipeline-style algorithm. First, the ground points are removed from the scene. After that, the pylons are detected following a feature map method. Then, several clustering techniques are applied between two detected pylons to extract and model the individual conductors. Similar to the technique presented in [182], the vertical distribution of the points is analyzed in the span between two pylons to detect peaks, which correspond to the electrical conductors at several height levels. The authors use conditional Euclidean clustering to classify the individual conductors. Once the electrical conductors are identified, each one is modeled through a projection to the XOY plane using the well-known catenary equation. After the powerline is modeled, the computation of distances to the closest obstacles is carried out.

Another work involving the extraction of powerlines using terrestrial mobile LiDAR data is presented in [186]. In this case, a height filter is used to distinguish between ground and wire points. Euclidean distance clustering is applied to cluster the power transmission lines. The distance parameter used in this clustering is fixed, so a lack of



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

data in the conductor could cause a single wire to be detected as two or more different conductors.

As we have just seen, most state-of-the-art algorithms are built in the context of on-purpose flights over existing powerlines in which the LiDAR scanning is performed. Thus, the data used in these algorithms are strips containing a high-voltage powerline, which implies that the development of the algorithm is drastically simplified. Also, the density of the 3D point clouds used is typically very high, which implies the need for significant storage capacity and computational processing. However, despite having high-density point clouds, no efforts were made to optimize the algorithms' computational performance. Also, a single span of powerlines is analyzed at a time, and no author considers the possibility of having two different powerlines in the same scene, crossing one over another, or even several sets of conductors coming out from a single pylon in different directions. Furthermore, when carried out, the clustering of the individual conductors does not consider situations such as the lack of points or the conductor overlapping in the vertical dimension that is common among high-voltage powerlines. Finally, most approaches group points into voxels or use some rasterization of the point cloud to simplify the algorithms, yielding a loss of precision.

This chapter presents an algorithm for classifying and modeling powerlines that address several issues not managed by state-of-art works. Thus, our algorithm can be applied to any general-purpose airborne LiDAR point cloud, and it can detect, classify, and model several powerlines present in a single point cloud. So, there is no need to perform a specific flight along the powerline to be analyzed or to have prior knowledge of the shape or the presence of a powerline in a given point cloud. Furthermore, and to the best of our knowledge, the proposed algorithm is the only one that can be applied to point clouds whose density is as low as $5\,\mathrm{pts/m^2}$. Besides, as the size of the 3D point clouds can be huge, there is a need for different algorithms that can solve this problem in an affordable time. Also, a significant effort was made to improve the algorithm's computational efficiency in manycore systems. Finally, we developed a Non-Maximum Suppression (NMS) method related to the object detection algorithm, which does not depend on heuristics to work properly.

## 5.1 Powerline detection algorithm

The proposed powerlines detection algorithm consists of four main stages, depicted in Figure 5.1. The algorithm's input is a 3D point cloud, and of all the information the cloud can include, it only uses the points' geometrical coordinates. The first stage of the algorithm is the Transmission line Point height-based Filter (TPF), which filters out points that are not likely to belong to a powerline. The second stage is a Hough transform, which detects straight lines in the non-filtered points. After the Hough transform, a clustering algorithm is applied to group the detected lines into individual wires. Finally, each cluster is modeled and characterized as a catenary curve [187,



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

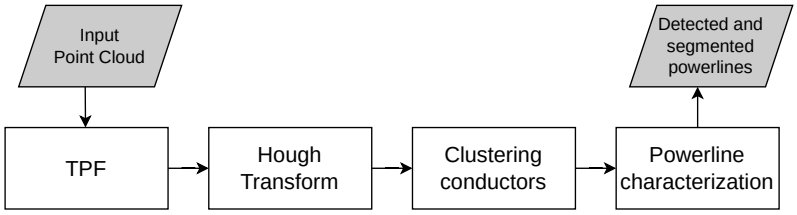



**Figure 5.1:** Powerline detection algorithm flowchart.

188]. The whole method is designed as a pipeline, where the output of one stage is used as the input of the next one.

### 5.1.1 Identifying vertically isolated points via a height-based filter

A typical point cloud is formed by tenths of millions of points. Thus, the first step of the algorithm is to reduce the number of points to be processed in the next step, in which straight lines will be detected. To do so, we have developed the TPF. The TPF is based on the assumption that the points belonging to a powerline are vertically isolated from the rest of the points in the scene, which are kept and used as input to the next stage.

When detecting straight lines in an image, it is common to use the Canny edge detector [189] to highlight only those points that could be part of a straight line. The goal of the TPF is somehow similar. As we are only interested in those points potentially belonging to a powerline conductor, the TPF is used to remove any points that are not likely to be part of the infrastructure.

The different steps of the algorithm are depicted in Figure 5.2. The TPF implies computing the neighborhood of every point available in the dataset. Among the different neighborhood-defining kernels that could be used, we selected the circular kernel, defined in Equation 3.3, since it is invariant against rotations and has infinite length in the vertical direction. So, the neighborhood $N_i$ of a given point is defined as

$$N_i := \mathcal{K}_{\text{circle}}(p_i, p_j, R) \tag{5.1}$$

where $p_i$ is the point whose neighborhood is being computed, $p_j$ any other point, $R$ the kernel radius, and $\mathcal{K}_{\text{circle}}$ the circular kernel.

The algorithm analyzes the neighborhood of every point to identify isolated points by computing the percentage of points $C_i$ with a vertical distance greater than a height threshold $H_{th}$. If this percentage exceeds a given threshold, $C_{th}$, the point is labeled as a powerline candidate. This method avoids needing a DTM, preserving accuracy.

The analysis involves computing two subsets within the neighborhood of a given point $p_i$. The first subset, denoted $W_i$, includes all points in the neighborhood that do



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

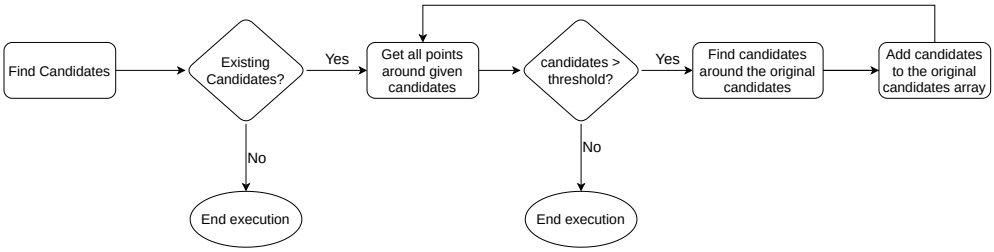



**Figure 5.2:** TPF flowchart.

not belong to the same conductor as the analyzed point. This subset is necessary to address false negatives caused by neighboring points belonging to the same conductor below $p_i$. To mitigate this, the wire thickness ($W_{th}$) is introduced as an estimated conductor thickness. So, for the $i$-th point, the $W_i$ subset is defined as

$$W_i := \{n_j \in N_i : |n_{jz} - p_{iz}| \ > \ W_{th}\} \tag{5.2}$$

where $n_j$ and $n_{jz}$ are the $j$-th neighbor and its height, respectively, $p_{iz}$ the height of the analyzed point, and $W_{th}$ the wire thickness.

The second subset $H_i$ is defined from $W_i$ via

$$H_i := \{w_j \in W_i : w_{jz} + H_{th} < p_{iz}\} \tag{5.3}$$

where $w_{jz}$ is the height of the $j$-th point belonging to $W_i$. Thus, $H_i$ includes those points that are lower than $H_{th}$ with respect to the analyzed point.

Finally, a point is labeled as a powerline candidate if the following condition is met:

$$C_i = \frac{|H_i|}{|W_i|} \geq C_{th} \tag{5.4}$$

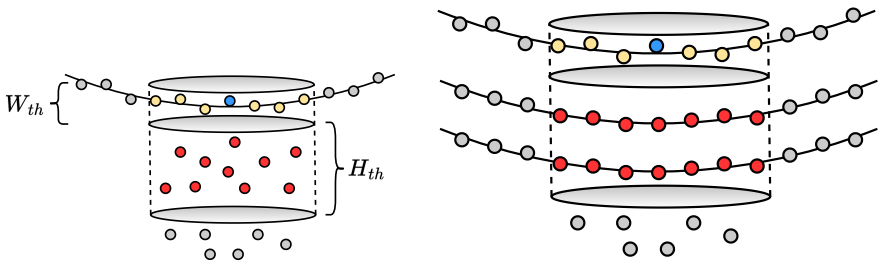
where $|H_i|$ and $|W_i|$ are the cardinalities of their respective sets corresponding to the $i$-th point, $C_{th}$ the ratio threshold, and $C_i$ the computed ratio of the $i$-th point.

This procedure is illustrated in Figure 5.3(a). The blue point is the one being analyzed. $N_i$ is formed by all points contained in the depicted cylinder. The yellow points are inside the limits of the $W_{th}$ parameter, so they are excluded from $N_i$, forming the $W_i$ set. The red points are inside the limit imposed by $H_{th}$, so they are excluded from $W_i$ to form $H_i$. These two sets are then used to compute $C_i$ and compare its value against $C_{th}$.

In 3D point clouds with high-voltage powerlines, multiple conductors often lie in the same vertical plane (Figure 5.3(b)). Thus, points of upper conductors are



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA




(a) Single powerline. (b) Multilevel powerline.

**Figure 5.3:** Application of the TPF algorithm to a point belonging to a single powerline and a multilevel powerline.

typically not labeled as powerline candidates due to the presence of points from other conductors below them. To address this, our TPF algorithm is executed iteratively, excluding already labeled points. This causes the points in multilevel powerlines to be labeled as candidates from the bottom conductors to the top.

Since the points not labeled as candidates in the first iterations will be close to the points that are labeled as candidates, the subsequent executions of TPF are only carried out in those points that are close enough to the initial candidates. This approach avoids analyzing areas where no points were labeled initially, reducing unnecessary overhead and focusing on areas more likely to contain powerlines. A neighborhood search using a circular kernel around the found candidates in the first iteration is used to build the new set of points that will be used as seed in the subsequent iterations of the TPF.

Note that if two candidate points are close enough, their neighborhoods are likely to overlap, which leads to the presence of repeated points in the set used as seed for the next TPF iteration. Despite only points not labeled as candidates are processed by the TPF, in a multithreaded parallel implementation of this algorithm, such as the one we have carried out, this could result in a repeated point being added twice to the candidate set if it is processed at the same time by two different threads. Instead of using a mutual exclusion mechanism to prevent duplicates, the points are stored in a Set, which is an associative container that stores a unique, sorted set of values.

The consequences of having duplicate points in the input set of the next stage of the algorithm are not negligible, as will be explained in the next section.

### 5.1.2 Detecting straight lines: The Hough transform

The Hough transform is an algorithm to detect parametric shapes in an image. The initial conception of the algorithm was patented by Paul Hough in 1962 [190]. The idea of detecting straight lines reached Paul Hough while seeking means to automate the



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

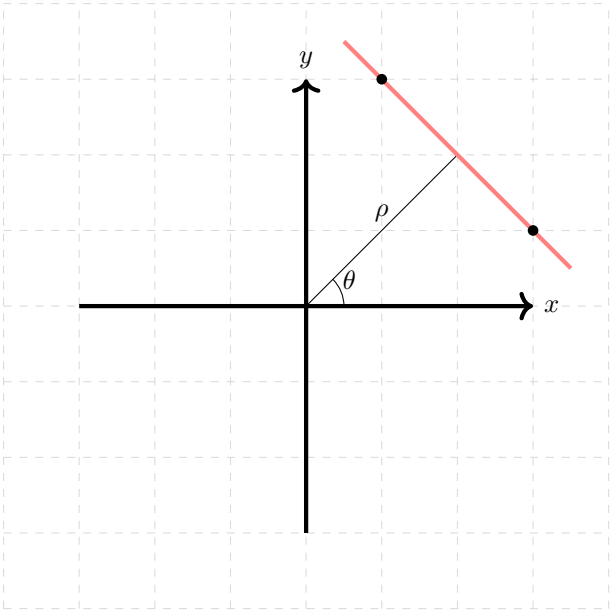



**Figure 5.4:** Example of two points sharing a line in the Euclidean space.

detection of subatomic particle tracks in bubble chamber photographs. Nevertheless, the implementation of Hough had the problem of using the equation of a line in its canonical form, which is not defined for vertical lines. Later, the algorithm was redefined by Richard Duda and Peter Hart [191] to use the equation of a line in its normal form, solving the problem of detecting vertical lines. The implementation of Duda and Hart evolved to more general algorithms that can detect any parametric shapes, such as ellipses [192] and arbitrary shapes [193]. More recently, the Hough transform was adapted from its use in images to point clouds, allowing the detection of lines [194], planes [195], or spheres [196] in the 3D space.

The Hough transform for 3D point clouds is a one-to-many application $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ that maps each point in the Euclidean space into a curve in the Hough space $\mathcal{H}$, defined as:

$$\mathcal{H}(\theta, \rho) = \{(x, y, z) \in \mathbb{R}^3 : \rho = x \cos\theta + y \sin\theta \ \forall\theta \in [0, \pi)\} \tag{5.5}$$

where $\theta$ the angle between the $x$ axis and the line that passes through $O = (0, 0, 0)$ and $p = (x, y, z)$, and $\rho$ the distance between the origin and the line that passes through $p$ at a given angle $\theta$. The meaning of the pair $(\theta, \rho)$ is shown in Figure 5.4. Since only $(x, y)$ are used in the transform, the projection of the points onto the $z = 0$ plane is done implicitly.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

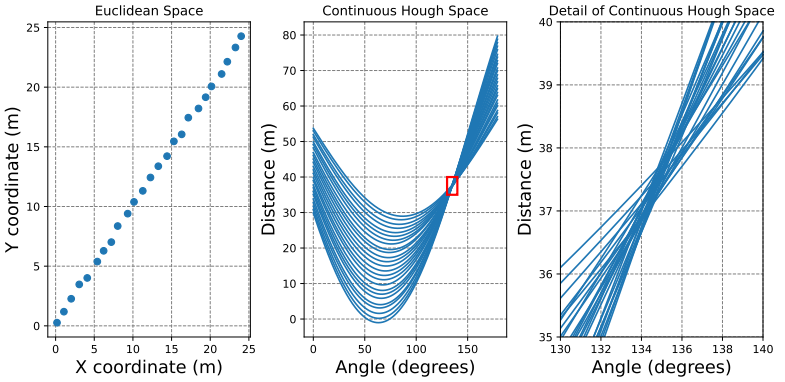



**Figure 5.5:** Continuous version of the Hough transform.

The idea behind the Equation 5.5 is that a single point in the Euclidean space corresponds to a sinusoidal curve in the Hough space. Following this reasoning, each point in the Hough space corresponds to a straight line in the Euclidean space. Thus, $n$ aligned points in the Euclidean space are mapped into $n$ curves in the Hough space that intersect in a single point, which corresponds to the straight line that passes through the $n$ points.

Equation 5.5 shows the Hough transform in its continuous form. Nevertheless, due to the continuous nature of the point clouds, the continuous transform is not suitable for finding the intersection points in the Hough space since only a perfectly aligned set of points would intersect at exactly one point in the Hough space, which is unlikely to occur. Figure 5.5 shows an example of this phenomenon. In this figure, the points in the Euclidean space are mapped into curves in the Hough space. As can be observed, in practice, the curves intersect not in a single point but in a small area around a point. Thus, the Hough space is discretized in order to be able to detect the intersection points. The discretized version of the Hough space is known as *accumulator*.

The accumulator is a matrix $\mathcal{A}_{n\times m}$ initialized to 0, where $n$ is the number of lines drawn through each point $p$ in the Euclidean space, and $m$ the number of discrete values that the distance of those lines to the origin may have. Both $n$ and $m$ are controlled by the angle step, $A_s$, defined as the minimum angle between two detectable straight lines, and the grid spacing, $G_s$, which is the minimum separation between detectable parallel lines. So, the dimensions of the accumulator are defined as:

$$n = \left\lfloor \frac{\pi}{A_s} \right\rfloor, \; m = \left\lfloor \frac{\rho_{max}}{G_s} \right\rfloor \tag{5.6}$$

where $\rho_{max}$ is the diagonal of the axis-aligned bounding box containing all the points



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

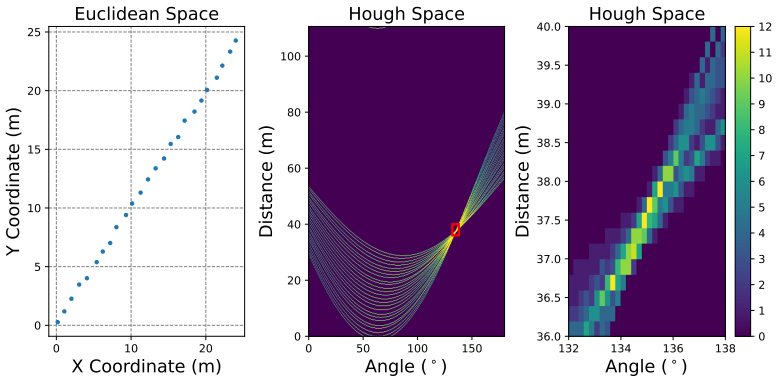



**Figure 5.6:** Discrete version of the Hough transform. In this case $A_s = 0.2$ and $G_s = 0.2$. The Hough space is colored according to the number of votes in each bin.

in the Euclidean space. Since the coordinates of the points can be very large, and the indexes are only defined in the interval $[0, \rho_{max}]$, the bounding box is translated by placing its center in the absolute origin of coordinates. In addition, half the distance of the diagonal is added to the calculated distance of a given line to ensure that all distances are positive, avoiding problems with the index computation.

So, for each pair of values $(\theta, \rho)$, the corresponding indexes are computed, and the accumulator's value in those indexes is incremented by one. The accumulator is filled by doing this with all the existing points in the Euclidean space. This process is commonly known as *voting*. The implemented algorithm is shown in Algorithm 5.1, and the results of the voting process in the discrete version of the Hough space are shown in Figure 5.6. The general workflow of the implemented Hough transform is shown in Figure 5.7.

Note a discrepancy between the Equation 5.5 and the implemented version in line 21 of Algorithm 5.1. As mentioned above, the implemented version of the voting procedure includes translating the point to the center of the bounding box. As the distance from the origin to the center was added to ensure that all the distances were positive, negative indexes in the accumulator were avoided.

After the voting procedure, the problem of finding straight lines in the Euclidean space is delegated to the task of finding the intersection points in the Hough space, which correspond to the highest populated bins in the accumulator. After finding the highest bin, its coordinates are stored, and the value of the bin is set to 0, so the next highest valued bin will be the next detected straight line. Nevertheless, despite the discretization, there are several bins with a significant number of votes around the current highest valued bin, as depicted in Figure 5.6. This is because the discretization of the Hough space is very fine, which causes a significant number of



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

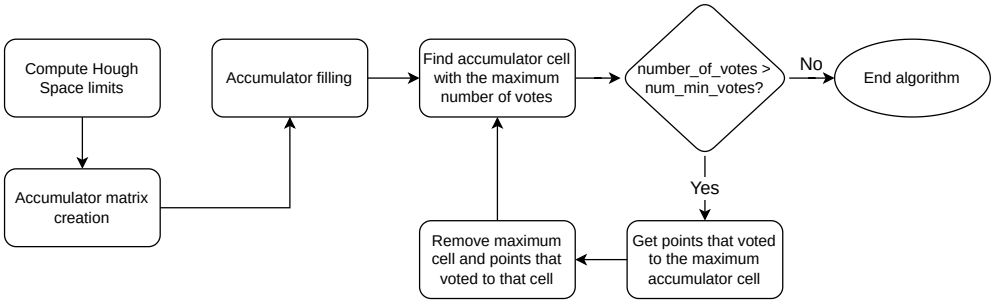



**Figure 5.7:** Flowchart of the Hough transform algorithm for 3D point clouds.

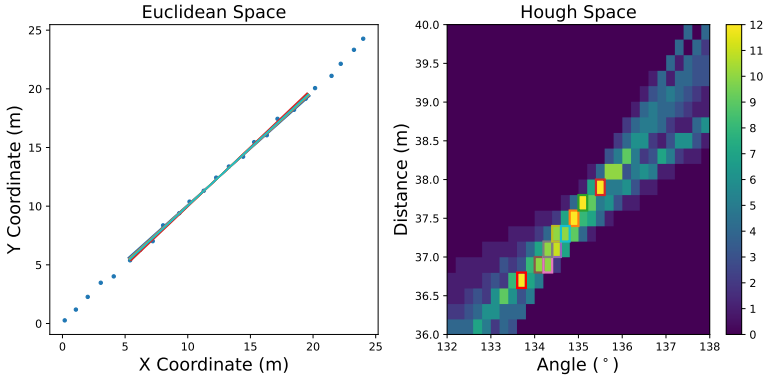



**Figure 5.8:** Multiple detections of the same line in the Euclidean space due to the discretization of the Hough space.

points belonging to a particular straight line in the Euclidean space to also belong to other very close Hough lines. This fact will produce several detections of the same line in the Euclidean space, affecting the recall of the algorithm. Figure 5.8 shows an example of this phenomenon. To solve this problem, a post-processing technique known as NMS is used, which will be described in the next section.

### Heuristicless non-maximum suppression

In any object detection algorithm, the detection is usually carried out using a sliding window to select an area of interest to be classified. The selection of the size of the step movement is crucial to perform a good detection: if the step is large enough, several objects of interest could be ignored. Conversely, a tiny step size could lead to



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Algorithm 5.1** Hough Space construction and voting procedure

**Input:** $\mathcal{P}$: set of points
$A_s$: angle step
$G_s$: grid space
**Output:** $\mathcal{A}$: Populated accumulator

**procedure** Hough space $(\mathcal{P}, A_s, G_s)$
  $x_{min},\ y_{min} \leftarrow \min\{p_x : p \in \mathcal{P}\},\ \min\{p_y : p \in \mathcal{P}\}$ ▷ Min x and y values
  $x_{max},\ y_{max} \leftarrow \max\{p_x : p \in \mathcal{P}\},\ \max\{p_y : p \in \mathcal{P}\}$ ▷ Max x and y values
  $x_c \leftarrow 0.5 \cdot (x_{max} + x_{min})$ ▷ x-coordinate of the bounding box center
  $y_c \leftarrow 0.5 \cdot (y_{max} + y_{min})$ ▷ y-coordinate of the bounding box center
  $w \leftarrow x_{max} - x_{min}$ ▷ Bounding box width
  $h \leftarrow y_{max} - y_{min}$ ▷ Bounding box height
  $\rho_{max} \leftarrow \sqrt{w^2 + h^2}$ ▷ Computation of the maximum distance to the origin
  $r \leftarrow 0.5 \cdot \rho_{max}$ ▷ Radius of the bounding box
  $n \leftarrow \left\lfloor \frac{\pi}{A_s} \right\rfloor$ ▷ Number of angle bins
  $m \leftarrow \left\lfloor \frac{\rho_{max}}{G_s} \right\rfloor$ ▷ Number of distance bins
  $\mathcal{A} \leftarrow \mathbf{0}$ ▷ Initialize all bins of the accumulator to 0
  coses $\leftarrow \mathbf{0}$ ▷ Initialize all cosines to 0
  sines $\leftarrow \mathbf{0}$ ▷ Initialize all sines to 0
  **for** $i \leftarrow 0, n$ **do** ▷ Pre-computation of trigonometric values
    $\theta \leftarrow i \cdot A_s$
    coses$[i] \leftarrow \cos\theta$ ▷ Computation of cosines
    sines$[i] \leftarrow \sin\theta$ ▷ Computation of sines
  **for all** $p \in \mathcal{P}$ **do** ▷ Actual voting procedure
    **for** $i \leftarrow 0, n$ **do**
      $\rho \leftarrow r + (p_x - x_c) \cdot \text{coses}[i] + (p_y - y_c) \cdot \text{sines}[i]$ ▷ Distance value
      $j \leftarrow \left\lfloor \frac{\rho}{G_s} \right\rfloor$ ▷ Distance index
      $\mathcal{A}_{i,j} := \mathcal{A}_{i,j} + 1$ ▷ Voting
  **return** $\mathcal{A}$



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

several detections of the same object. The latter is usually preferred since windows belonging to the same object can be merged in a process known as (NMS).

NMS is a post-processing technique that aims to remove redundant detections of the same object. Iteratively, for each detection, the algorithm checks whether another detection with a higher confidence score and a high overlap exists. If so, the current detection is removed. The algorithm is repeated until no more detections are removed. This technique is used in some of the most top-notch object detection algorithms, such as YOLO [197], R-CNN [198], and Viola & Jones [199]. Several techniques for performing NMS have been proposed in the literature, being the most common the greedy approach (GreedyNMS [200]), which consists of selecting the detection with the highest confidence score and removing all the neighboring detections in the search space. This approach is repeated until no more detections remain. The greedy approach is simple and fast, but it is not optimal since it can remove detections that do not belong to the actual detected object. The GreedyNMS method requires a parameter to select the threshold for the overlap between detections. The correct selection of this parameter is crucial to obtain an accurate detection. On the one hand, choosing a value that is too high would lead to the deletion of close detections, resulting in false negatives. On the other hand, opting for a value that is too low would fail to eliminate all redundant detections, resulting in a significant number of false negatives and compromising the algorithm's recall. As stated in [200], "GreedyNMS is doomed to sacrifice precision or recall independent of its parameter".

An example of the application of GreedyNMS is shown in Figure 5.9. The plot corresponding to the Euclidean space shows three close parallel lines (top-left plot) that consist of different numbers of points (50, 40, and 15). The corresponding bins of the three parallel lines are highlighted in the Hough space shown in the top-right plot. Note that some cells surrounding the highest valued bin have a higher value than the bin corresponding to the last detected line. Unless the value to perform the GreedyNMS is selected correctly, those cells will cause a false positive if the suppression radius is not big enough or a false negative if the suppression radius is chosen by excess. This is shown in Figure 5.9, where we can see how a suppression radius of 1 causes a false positive, highlighted in the bottom-left plot, whereas a suppression radius of 3, shown in the bottom-right plot, causes a false negative since the bin corresponding to the second line is suppressed before being detected, yielding only two detected lines instead of three.

Other implementations of NMS in the literature are the Soft-NMS [201], an alternative in which the scores of the overlapping objects are reduced as a continuous function of their overlap with the current detection; the Adaptive NMS [202], which propose the use of a dynamic threshold for the overlap between detections to improve the detection of pedestrians in crowds; and the Distance-NMS [203], in which the overlap between detections and their distance between center points are used simultaneously to suppress redundant detections of the same object. Nevertheless, all these methods are based on the greedy approach, and thus, they require the selection of a



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Algorithm 5.2** Voting Elements Suppresion algorithm

**Input:** $\mathcal{H}$: accumulator
$\mathcal{P}$: set of points
$\mathcal{D}$: set of points already belonging to a line
$n_{th}$: bin value threshold
$\theta_{idx}, \rho_{idx}$: Hough Space coordinates of the current detected line
coses, sines: pre-computed trigonometric values

**Output:** $\mathcal{A}$: Suppressed accumulator
$\mathcal{P}_{sel}$: Array of points belonging to the current detected line

1: **procedure** VES($\mathcal{H}, \mathcal{P}, \mathcal{D}, n_{th}$, $\theta_{idx}, \rho_{idx}$, coses, sines)
2: $\mathcal{P}_{sel} \leftarrow \emptyset$ ▷ Array of points belonging to the current detected line
3: $n \leftarrow \text{size}(\mathcal{H}, 1)$ ▷ Number of angles used in the voting process
4: **for all** $p \in \mathcal{P}$ **do**
5: **if** $p \in \mathcal{D}$ **then**
6: **continue**
7: $\hat{\rho} \leftarrow r + (p_x - x_c) \cdot \text{coses}[\theta_{idx}] + (p_y - y_c) \cdot \text{sines}[\theta_{idx}]$ ▷ Distance value
8: $\hat{\rho}_{idx} \leftarrow \left\lfloor \frac{\rho}{G_s} \right\rfloor$ ▷ Distance index
9: **if** $\rho_{idx} \neq \hat{\rho}_{idx}$ **then** ▷ The point $p$ didn't vote for the current selected bin
10: **continue**
11: $\mathcal{D} \leftarrow \mathcal{D} \cup \{p\}$ ▷ Label the point as belonging to a line
12: $\mathcal{P}_{sel} \cup \{p\}$ ▷ Add the point to the array of points belonging to the current detected line
13: **for** $\theta_{idx}^{del} \leftarrow 0, n$ **do** ▷ Removal of the curve generated by $p$
14: $\rho^{del} \leftarrow r + (p_x - x_c) \cdot \text{coses}[\theta_{idx}^{del}] + (p_y - y_c) \cdot \text{sines}[\theta_{idx}^{del}]$
15: $\rho_{idx}^{del} \leftarrow \left\lfloor \frac{\rho^{del}}{G_s} \right\rfloor$
16: $\mathcal{A}[\theta_{idx}^{del}, \rho_{idx}^{del}] \leftarrow \mathcal{A}[\theta_{idx}^{del}, \rho_{idx}^{del}] - 1$ ▷ Actual removal of the votes generated by $p$
17: **return** $\mathcal{A}, \mathcal{P}_{sel}, \mathcal{D}$



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

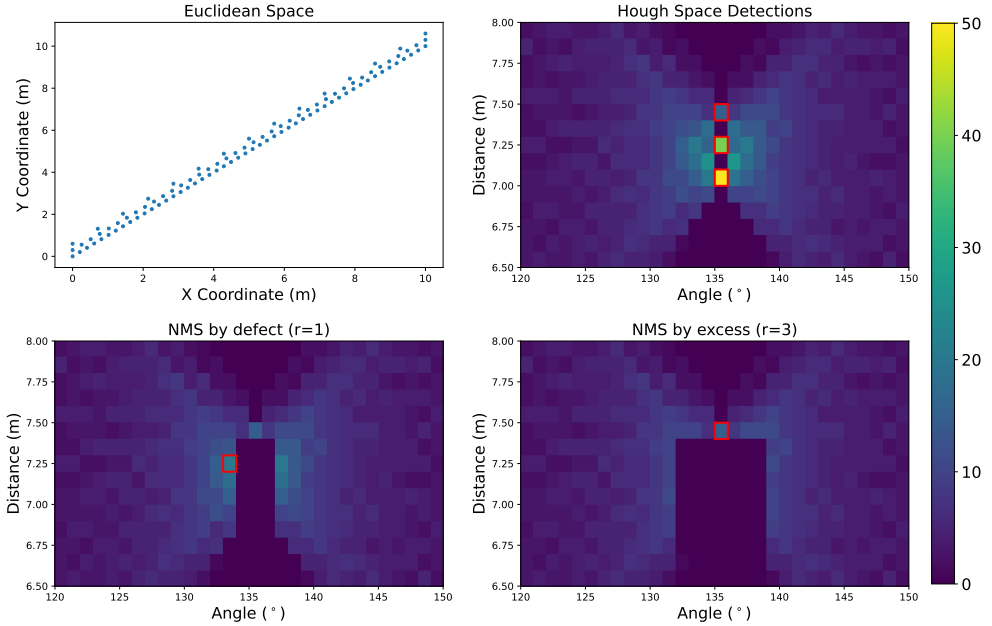



**Figure 5.9:** GreedyNMS algorithm.

threshold to decide whether two detections belong to the same object or not.

To address these issues, we propose a new algorithm called Voting Elements Suppression (VES). Instead of simply removing the highest bin and its surrounding areas, we remove the curves present in the Hough space that correspond to all the points belonging to the line represented by the current highest valued bin. We do not remove the surroundings but instead attenuate them in the exact way required to avoid any false positives or negatives in subsequent detections. VES (Algorithm 5.2) returns the accumulator in the state that it would have if the detected line were not present in the given point cloud. The procedure is applied for each maximum bin encountered in the accumulator. Once the value of the highest bin is lower than a certain threshold $n_{th}$, the process is stopped, and the line detection ends. This method allows not only the detection of the position of a given line but also the identification of which points belong to each line, which is helpful for further analysis of the detected lines. The algorithm's behavior applied over the data of Figure 5.9 is shown in Figure 5.10. Note how the maximum is completely removed, and the surroundings are attenuated in the exact way needed so as not to cause any false positives or negatives in subsequent detections. The deletion of the curves in a real example is shown in Figure 5.11.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

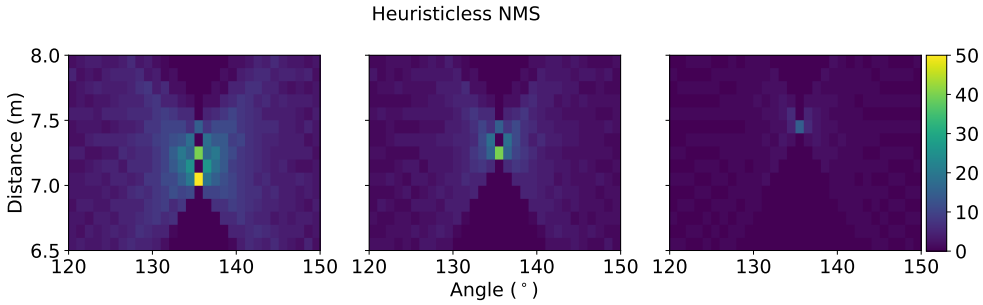
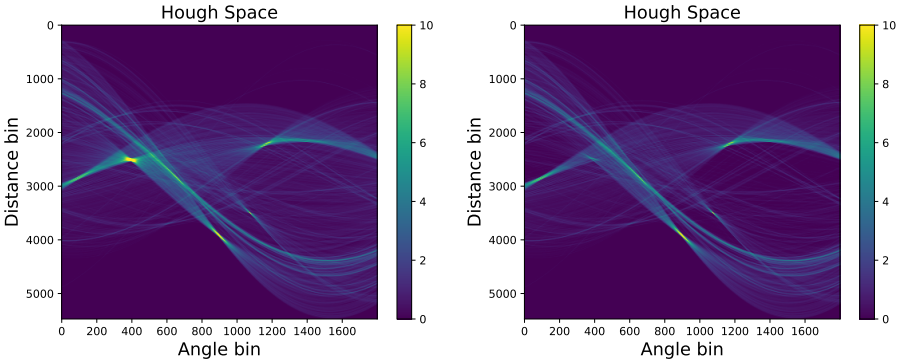



**Figure 5.10:** VES: the heuristicless NMS algorithm.




(a) Origin Hough Space.

(b) Deletion of first three curves.

**Figure 5.11:** Hough transform curve deletion example.

This method has several advantages compared to the state-of-art NMS algorithms. First, only the most probable line is associated with each of the current lines present in the Euclidean space. Second, detecting very close parallel lines is possible since the NMS is carried out without parameters. Third, since the surroundings of a given maximum are correctly attenuated, a false positive is not possible. On the other hand, the method introduces a computational burden that the classic NMS does not have: the suppression of the curves has the same time complexity as the initial voting stage, and that suppression is performed once per line present in the scene.

Suppressing non-maximum bins by deleting the curves presents an issue when duplicated points are present in the input of the Hough transform. When a point is present multiple times in the voting stage, its corresponding curve in the Hough space



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

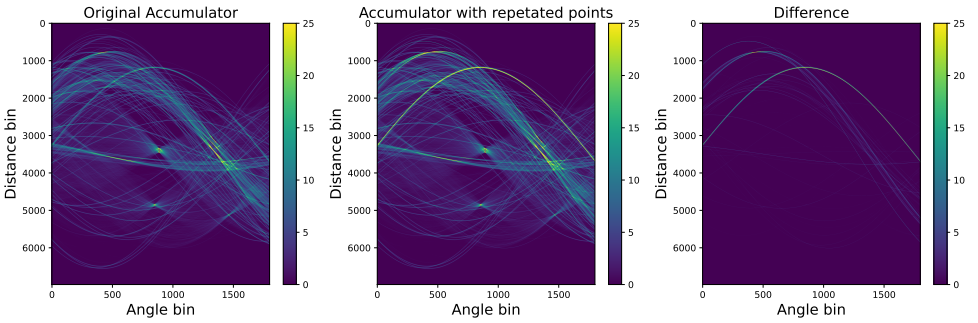



**Figure 5.12:** Detail of Hough spaces with and without repeated points and their difference.

will be artificially populated, as shown in Figure 5.12. When iteratively suppressing curves, some bins of the curves generated by the duplicated points may have higher values than an actual bin representing a straight line to be detected, so when a bin of the aforementioned curve is detected as a maximum, all the curves that also contain that bin are suppressed. As a consequence, the value of the bins representing an actual straight line where the points associated with the deleted curves belong decreases. This leads to false positives in the detection of straight lines.

### 5.1.3 Powerline clusterization

The algorithms presented so far allow the detection of straight lines. Nevertheless, due to the application of the Hough transform using as input data a projection of the candidates onto the $z = 0$ plane, several lines sharing the same vertical plane are detected as only one line. Previous works combining the Hough transform with LiDAR data faced challenges distinguishing vertically spread conductors using 2D projections [182, 186, 204].

In [204], a *minimum linkage hierarchical clustering approach* is used to group the detected lines into corridor segments. Nevertheless, that work does not consider the case of having multiple, vertically distributed powerline conductors. In [182], points detected as belonging to powerline conductors are projected onto a plane perpendicular to the direction of the powerline itself. The authors claim that points belonging to the same conductor converge into the same cluster in the projected plane. This is true as long as there is no overlap between the conductors in the vertical dimension, which often occurs when the height difference between the start and end pylons is significant. In [186], the authors use a Euclidean distance clustering with a fixed radius to group the multiple vertically spread detected lines into powerline conductors. Notwithstanding, as stated in [205], the point densities of powerlines are uneven, which



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

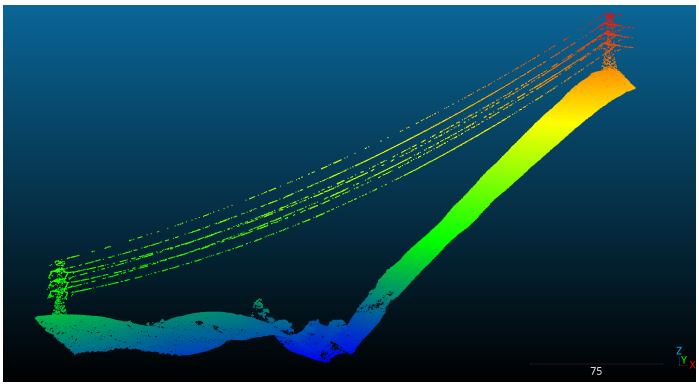


**Figure 5.13:** Powerline to be used as an example in the clustering algorithm.

is a problem when using a fixed radius clustering method. In [182], it is explained that the distance between two consecutive points belonging to the same conductor could be greater than the distance between two parallel conductors (both in the horizontal and the vertical planes).

To address these issues, we propose a method to group the detected lines into individual powerline conductors. The proposed method is robust against uneven and very low densities and to the steep slopes that the conductors may have in the proximity of the pylons when the height difference between them is significant. The algorithm accomplishes two tasks: determining the number of conductors sharing a vertical plane and segmenting them individually, which is essential for detecting objects near powerlines and analyzing the powerline characteristics through the analysis of its shape.

To illustrate the different steps of the algorithm, we will use a triple catenary, extracted using the Hough transform on the Diablo Canyon dataset [206] as an example. The corresponding powerline is shown in Figure 5.13.

The first step of the clusterization is to project each detected line to the vertical plane containing it. Let $\mathcal{P}$ be the points of a straight line detected using the Hough transform. Since the Hough transform preserves the dimensionality of the points, the orientation of the vertical plane containing the conductors is arbitrary. To project the points into the vertical plane in which they are contained, the transformation $D : \mathbb{R}^3 \to \mathbb{R}^2$ is applied to the set $\mathcal{P}$ defined as:

$$D(p_x, p_y, p_z) := \left( \sqrt{(p_x - o_x)^2 + (p_y - o_y)^2}, p_z \right) \tag{5.7}$$

being the point $o$ defined as

$$o = (o_x, o_y) := (\min \{p_x : p \in \mathcal{P}\}, \min \{p_y : p \in \mathcal{P}\}) \tag{5.8}$$



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

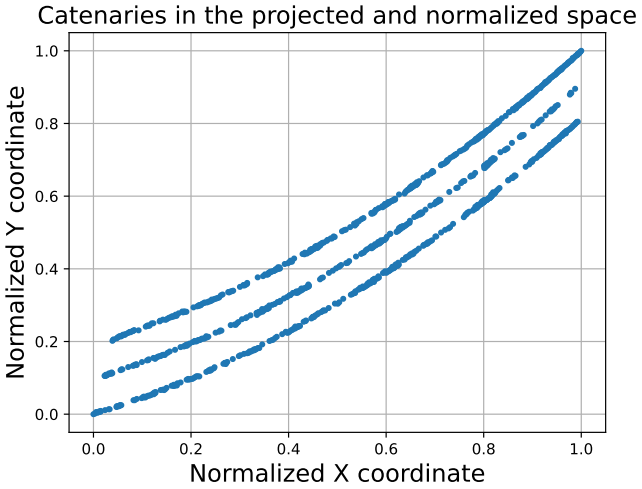



**Figure 5.14:** Normalization of the powerline used as an example in the clustering algorithm.

After the transformation, point coordinates are normalized to the $(0, 1)$ interval to improve the numerical stability of subsequent steps. The transformed and normalized points of the example powerline are shown in Figure 5.14.

The second step is the clustering itself, exploiting the fact that clusters consist of quasi-parallel curves, which can be modeled using $k$-order polynomials or the general catenary equation. Using the latter requires a nonlinear fitting method (e.g., the Levenberg-Marquardt algorithm [207, 208]) with an unknown number of iterations and initial parameter estimation. Polynomial fitting doesn't require initial parameter estimation and can be done at once using the Moore-Penrose inverse [209], making it computationally more efficient. With this, it is reasonable to perform the fitting using a $k$-order polynomial. Since the actual catenary equation is an even function, only even degree best fitting polynomials are relevant. Let $\boldsymbol{x} = (x_1, \ldots, x_m)$ and $\boldsymbol{y} = (y_1, \ldots, y_m)$ be the two sets of coordinates corresponding to $m$ points in the normalized space. The algorithm goal is to get $n$ polynomials $P_i(x)$ with coefficients $\boldsymbol{w}_i = (w_0, \ldots, w_{k+1})$ that fit every curve in the dataset. Therefore, the number of polynomials $n$ will match the number of conductors sharing the same vertical plane.

A proper selection of the points to be fitted is crucial to the algorithm's performance. Fitting a set of points belonging to different curves will cause a poor fitting. To choose the points to be fit, the x-axis is equally divided into $n_{\text{bin}}$ intervals, where $n_{\text{bin}} = \sqrt{N}$, being $N = |\mathcal{P}|$. In each interval $\mathcal{S}_i$, the point $p = \min p_y : p \in \mathcal{S}_i$ is selected as a suitable sample, as shown in Figure 5.15. In the case of uneven point density in a given conductor, the horizontal distance between two consecutive points



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

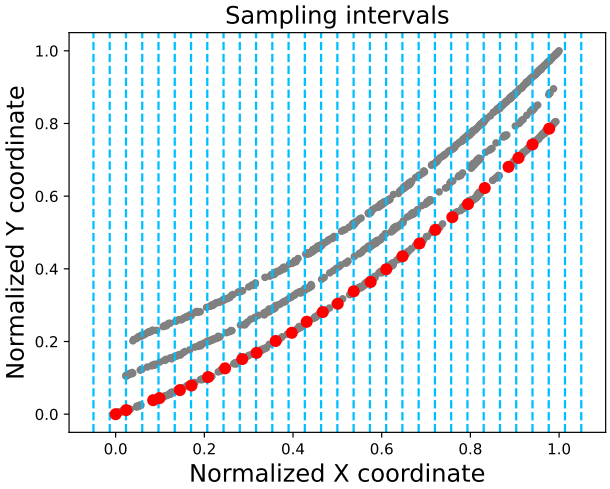



**Figure 5.15:** Sampling intervals for the RANSAC algorithm. The lowest points encountered in each interval are in red.

may be greater than the interval width, which may cause the selected point in that interval to be susceptible to belonging to one of the upper conductors if they exist.

To address this issue, the fitting process is carried out through the RAndom SAmple Consensus (RANSAC) algorithm [210], which is robust when outliers are present. Typically, the RANSAC algorithm needs to be fed with two parameters: the maximum number of trials ($n_{\text{trials}}$) before stopping and returning the best found fit and the number of samples ($n_{\text{samples}}$) to be used in the fitting process.

The number of samples is chosen so the fitting step meets the Rouché-Frobenius theorem, i.e., $n_{\text{samples}} = k + 1$. By doing this, we ensure that the found solution for a given fitting is unique and optimal. The number of trials is chosen so every possible combination of $n_{\text{samples}}$ without repetition is tested before returning the best fit. So, the number of trials is defined as

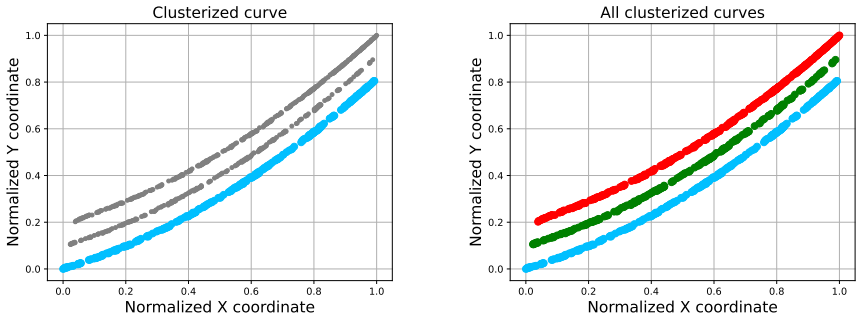
$$n_{\text{trials}} = \binom{n_{\text{bin}}}{n_{\text{samples}}} = \frac{n_{\text{bin}}!}{n_{\text{samples}}!(n_{\text{bin}} - n_{\text{samples}})!} \tag{5.9}$$

Since the fitting method is computationally cheap, this is an affordable solution. Finally, the found polynomial $\boldsymbol{w}_i$ is used to build the vector of predictions $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$, i.e., the vector containing the predictions for every point in the subset. Therefore, the points belonging to the lowest catenary are those satisfying:

$$|\hat{y}_i - y_i| < \epsilon \tag{5.10}$$



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA




(a) Clusterization of the first curve.

(b) Clusterization of all the curves.

**Figure 5.16:** Clusterization of the points using the curve fit with the lowest points in the dataset.

where $\epsilon$ is the maximum error for a point to be included in the current clusterized curve. Of course, this value must be chosen to be less than the vertical separation between parallel curves.

The already clusterized points (as shown in Figure 5.16(a)) are then removed (or labeled as already clusterized), and the process is repeated until no more curves are encountered, yielding to a proper clusterization of the individual conductors (as shown in Figure 5.16(b)).

Figure 5.17 shows an example in which multiple catenaries created using different parameters are fitted using our algorithm. These catenaries are not parallel and were built with different catenary parameters and densities. Also, some data were removed to simulate the absence of data in some areas of the conductors. Despite the differences between the catenaries, the algorithm is able to clusterize them properly. The initial sampling intervals and the fitted curves are shown in the first row of the figure. The second row is where the usage of RANSAC is justified. Due to the introduced lack of data, not all the points selected in the sampling stage belong to the same curve. Nevertheless, the RANSAC algorithm is able to fit the curve properly. Represented with circles are the inlier points, i.e., the points used to fit the corresponding curve, and represented with crosses are those points considered outliers by the algorithm and discarded for their use in the fitting process. The last row in the figure shows the clusterization results. The points belonging to the same curve are represented with the same color. As can be observed, the algorithm can clusterize the points correctly, even when the curves are not parallel and when the density of the points is uneven.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

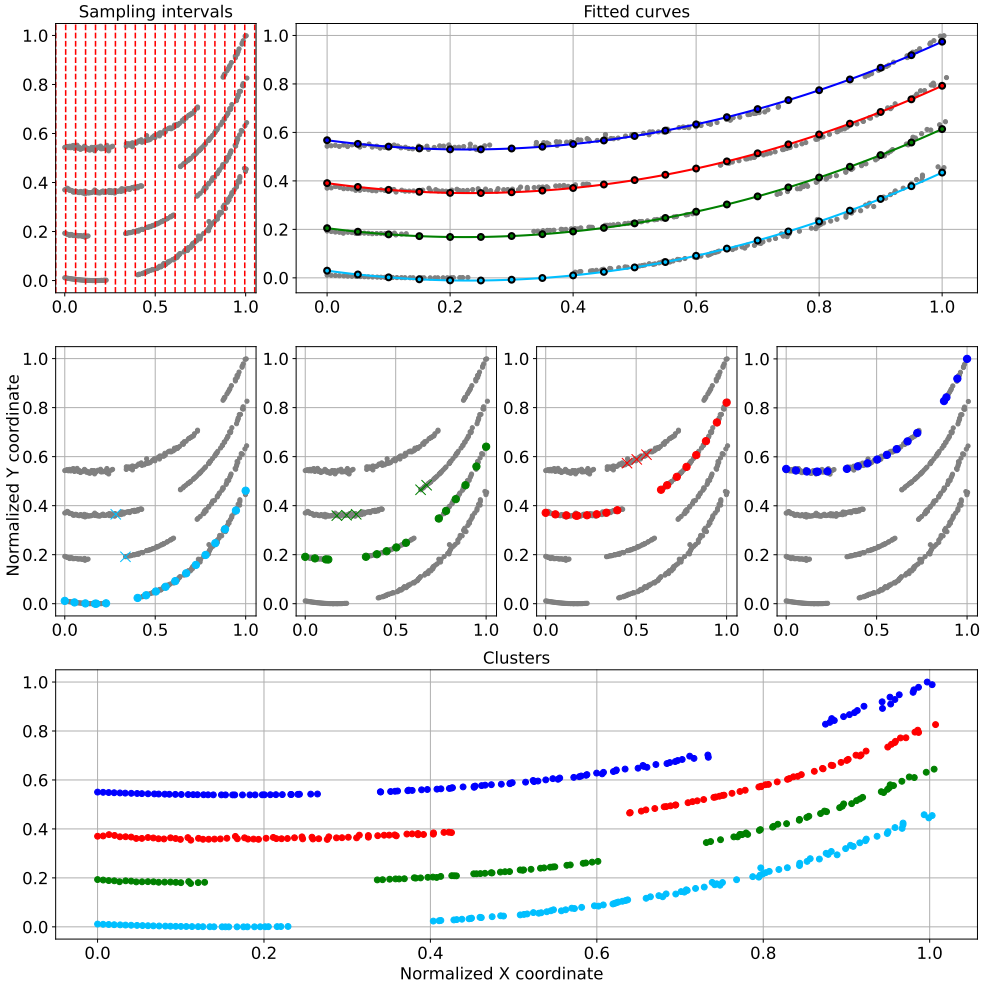



**Figure 5.17:** Clusterization of multiple, non-parallel, uneven catenaries.

### 5.1.4 Powerline characterization

Once the powerline conductors are detected and clusterized, the next step is to characterize them. Characterizing the powerline conductors is a crucial step for analyzing the powerline itself since it allows us to determine the sag of the conductors, the distance between them, and the height of the pylons. This information is useful for analyzing



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

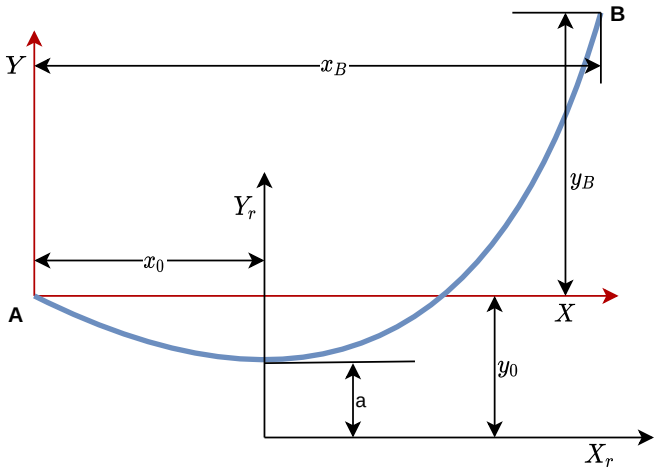



**Figure 5.18:** Diagram of a catenary curve.

the powerline and the objects near it. In particular, the sag of the conductors is a key factor in determining the clearance between the powerline and the objects.

The shape of a hanging cable is governed by a curve known as catenary [187, 188]. A catenary curve and its relevant parameters are shown in Figure 5.18. Typically, the catenary is represented using one of the two following frames of reference: the absolute or the reduced axis reference. The former is an orthogonal frame of reference whose origin is located in one of the two anchors of the catenary. The anchors are defined as the points attached to the structures that support them, typically pylons. In this case, the left anchor point was chosen as the origin of this reference system, formed by the $(X, Y)$ axis. On the other hand, the reduced axis reference, $(X_r, Y_r)$, is placed so its origin is vertically aligned with the lowest point of the curve. In the case where both anchor points are exactly at the same height, and the density of the conductor is uniform, the lowest point of the curve is located exactly at the midpoint between both anchor points. In that case, the catenary is defined by $y(x) = a\cosh\left(\frac{x}{a}\right)$. Nevertheless, this rarely occurs in real powerlines, so the catenary is defined with respect to the absolute axis reference system as

$$y(x) = y_0 + a\cosh\left(\frac{x - x_0}{a}\right) \tag{5.11}$$

where $(x_0, y_0)$ are the translation parameters from the reduced to the absolute axis and $a$ is the catenary parameter, which defines the curvature of the catenary. Physically, this parameter represents the relation between the tension of the conductor and its



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

lineal density. Mathematically, its value is equal to the distance of the lowest point of the curve to the $X_r$ axis.

Since we know the position of the LiDAR points belonging to the curve, one technique to compute the lowest point of the curve, and therefore $(x_0, y_0)$, is to retrieve the point with the minimum $y$ coordinate. Nevertheless, we are working with airborne LiDAR data, and, as explained in Section 2.1.1, the greatest error in the measurement is presented in the direction of the laser beam. This implies that the lowest LiDAR point may not correspond to the actual lowest conductor point.

Computing $(x_0, y_0, a)$ is not trivial because the equation governing a hanging cable's shape is transcendental, so it must be solved using numerical methods. Some works in the literature present methods to solve this equation. For example, in [182] Huang et al. state that the equation can be solved using the least-squares method, but they do not provide any details of the procedure. Another work that solves the equation using the least-squares method is [185], where the authors claim to use the Ceres solver [211] to solve it without providing more insights. In [186], Guan et al. claim that the reconstruction of catenary curves is used to determine the optimal values for $(x_0, y_0, a)$, but the explanation of the method is not present in that work. In [183], the Particle Swarm Optimization method [212] is used to determine the $(x_0, y_0, a)$ that best fit the data, using as initial values the coordinates of the lowest point in the catenary and a random value for $a$.

Solving Equation 5.11 implies finding the parameters $(x_0, y_0, a)$ that minimize the sum of the residuals $S$:

$$S = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{5.12}$$

where $y_i$ is the value of Equation 5.11 for a given $x_i$ and $\hat{y}_i$ is the prediction made with the current computed parameters $(x_0, y_0, a)$.

Several algorithms are suitable to find the set of parameters that minimize Equation 5.12. For example, the Gauss-Newton method [213, 214], which is the basis of most existing algorithms to solve non-linear least squares problems. This method uses a linear approximation of the model around the solution to iteratively find the best set of parameters. The Gradient descent method [215] is also suitable for solving this kind of problem. The direction towards the solution is found by computing the gradient of the objective function. A combination of these two methods is presented in the Levenberg-Marquardt [207, 208] algorithm, which is a modification of the Gauss-Newton method that adds a damping factor to vary between the gradient descent and the Gauss-Newton updates.

All these algorithms need an initial estimation of the model parameters. While giving an initial estimation of $(x_0, y_0)$ is straightforward, as shown in [183], the initial guess for $a$ is not. A poorly chosen initial value for this parameter may cause the optimization algorithms to diverge. Thus, we have developed a method to compute



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

the best-fitting parameters for the catenary equation without needing an initial guess for any of them.

It is possible to compute the unknowns $(x_0, y_0, a)$ using the position of the anchor points of the curve, $(x_A, y_A)$, $(x_B, y_B)$, plus the length of the curve $l$. Using these conditions, the following system of equations is considered:

$$\begin{cases} l = a \sinh\left(\frac{x_0}{a}\right) + a \sinh\left(\frac{x_B - x_0}{a}\right) \\ y_A - y_0 = a \cosh\left(\frac{x_A - x_0}{a}\right) \\ y_B - y_0 = a \cosh\left(\frac{x_B - x_0}{a}\right) \end{cases} \tag{5.13}$$

This system can be reduced to

$$\sqrt{l^2 - y_B^2} = 2a \sinh\left(\frac{x_B}{2a}\right) \tag{5.14}$$

Here, $(x_B, y_B)$ are known. $l$ can be computed with the arc length formula applied to the fitted polynomial obtained in the clusterization stage:

$$l = \int_A^B \sqrt{1 + \left(\frac{dP_i(x)}{dx}\right)^2}\, dx \tag{5.15}$$

which must be solved via numerical integration. Once the value of $l$ is known, only the $a$ variable remains unknown in Equation 5.14, which can be expressed as follows:

$$f(\lambda) = \sinh \lambda - k\lambda \tag{5.16}$$

where $\lambda = \frac{x_B}{2a}$ and $k = \frac{\sqrt{l^2 - y_B^2}}{x_B}$. This transcendental equation can be numerically solved using Brent's method [216]. We chose this method due to its superior convergence ratio compared to traditional root-finding algorithms, such as the secant, bisection, or Newton's. Brent's method requires the initial interval containing the searched root as input parameters. The graph of Equation 5.16 is shown in Figure 5.19. For values $k \in \mathbb{R} : k \leq 1$ or $k \in i\mathbb{R}$, the equation only has the solution $\lambda = 0$, which corresponds to the case where the existence of a conductor is physically impossible. There are three different scenarios causing that:

- The length $l$ of the catenary is 0. In that case, $k \in i\mathbb{R}$.
- $x_B$ is infinite, which leads to $k = 0$.
- $l = y_B$, which is impossible since the minimum possible value for $l$ is $\sqrt{x_B^2 + y_B^2}$, so the condition is reduced to $x_B = 0$, i.e. both anchor points are the same, and the $k$ value would be undefined.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

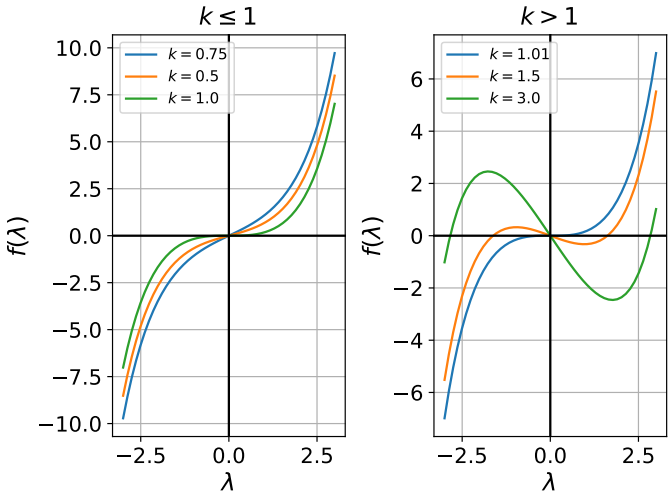



**Figure 5.19:** Graph of $f(\lambda)$ for several k values.

When none of the above three conditions are met, i.e., $k > 1$, there are always three roots, two of them ($\lambda < 0$ and $\lambda = 0$) physically meaningless. Therefore, Brent's algorithm allows us to find the positive root of Equation 5.16. The initial search interval is defined as $(\epsilon, b)$, being $\epsilon$ a tiny positive value such as $f(\epsilon) < 0$, and $b$ the first encountered value such as $b > 0$, $f(b) > 0$ and $f(\epsilon) \cdot f(b) < 0$. Once $\lambda$ is computed, the value of $a$ is easily calculated as $a = \dfrac{x_B}{2\lambda}$.

After computing $a$, solving another transcendental equation to compute $x_0$ is necessary. By subtracting the equations of the catenary passing through points A and B, the following function is obtained:

$$f(x_0) = a\cosh(\epsilon_A) - a\cosh(\epsilon_B) + y_B - y_A \tag{5.17}$$

where $\epsilon_A = \dfrac{x_A - x_0}{a}$ and $\epsilon_B = \dfrac{x_B - x_0}{a}$. As shown in Figure 5.18, $x_0$ must be between both anchor points. This allows us to use the Newton-Raphson [213, 217] method to obtain the roots of Equation 5.17 since it only requires an approximate value of the function root as input. In this case, the midpoint between both anchor points is chosen.

Finally, once $x_0$ is computed, the computation of $y_0$ is straightforward using the catenary equation.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Table 5.1:** Comparison of the error in the catenary parameters numerically computed when using different polynomials and the actual catenary equation.

| | $P_2$ | $P_3$ | $P_4$ | $P_6$ | $P_8$ | $Cat$ |
|---|---|---|---|---|---|---|
| $x_0$ | 77.48 % | 22.64 % | 5.09 % | 0.17 % | 0.13 % | 0.08 % |
| $y_0$ | 17.57 % | 11.28 % | 1.85 % | 0.05 % | 0.05 % | 0.03 % |
| $a$ | 29.38 % | 8.92 % | 1.99 % | 0.05 % | 0.05 % | 0.03 % |
| $l$ | 5.05 % | 0.95 % | 0.27 % | 0.01 % | 0.01 % | 0.02 % |

### 5.1.5 Validation of the characterization procedure

Numerical estimation of catenary coefficients gives off an error term since none of the mentioned methods is exact. By using a polynomial fitting, the main error source is controlled by the polynomial degree, whereas by using the catenary equation directly, the chosen non-linear algorithm is the main source of error.

Let $x_0 = 1.0$ m, $y_0 = 2.0$ m, and $a = 5.0$ m define a catenary in the interval $x \in [-2.5, 10]$ m. Its length, computed using Equation 5.13, is $l = 18.5037$ m. Table 5.1 shows the errors in parameter estimation, which decrease as the polynomial degree increases. While the error is minimal when the actual catenary equation is a regressor in the fitting procedure, the disadvantages outweigh the slight improvement compared to the 8th-degree polynomial fitting, which justifies its use.

## 5.2 Results and experiments

Several datasets were used to validate the proposed method. The first point cloud corresponds to a subset of the Diablo Canyon dataset obtained from OpenTopography [206]. This dataset contains, along with high-voltage powerlines, some power transformers that are interesting for testing the algorithm's robustness. The second dataset, privately provided by Babcock International [164], contains mid- and low-voltage powerlines and corresponds to the surroundings of the city of Lugo (Spain). The third dataset is the well-known DALES dataset [109], the reference ALS benchmark for testing classification algorithms. The data was collected in the city of Surrey in British Columbia, Canada. This dataset is split into 40 disjoint tiles, containing on average 12 million points each. Not every tile contains powerlines, so we have selected four tiles to carry out the experiments: 5105_54460 (dales10), 5130_54355 (dales17), 5135_54435 (dales19), and 5180_54485 (dales36), according to the dataset naming convention. The number of points of the four tiles combined is 51 249 744 points. A summary of the characteristics of the datasets is shown in Table 5.2. Their orthomosaics and the corresponding point clouds are shown in Figures 5.20, 5.23, and 5.25.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Table 5.2:** Summary of the datasets used for validation.

| Dataset | Number of Points | Density (pts/m$^2$) |
|---|---|---|
| Diablo Canyon | 8 097 766 | 8.93 |
| Lugo | 2 520 403 | 15.00 |
| dales10 | 12 954 374 | 51.81 |
| dales17 | 12 148 800 | 48.60 |
| dales19 | 14 196 538 | 56.79 |
| dales36 | 11 950 032 | 47.80 |

The selection of datasets aimed to address three notable gaps identified in current state-of-the-art research. Firstly, there is a limited representation of high-voltage electrical towers at varying heights above ground level in existing literature [218]. Secondly, existing studies predominantly focus on analyzing point clouds featuring a single powerline, neglecting scenarios with multiple powerlines. Lastly, mid- and low-voltage powerlines are not usually considered to be at the same point cloud.

### 5.2.1 Results on the Diablo Canyon dataset

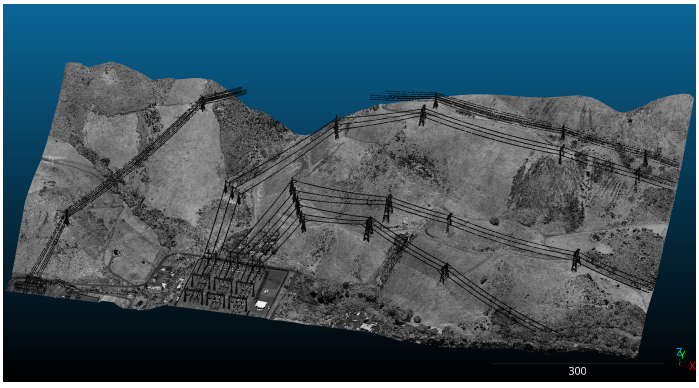
The outcomes of the TPF applied to the Diablo Canyon dataset are depicted in Figure 5.22, utilizing parameters $R = 1.0$,m, $H_{\text{th}} = 3.5$,m, $C_{\text{th}} = 0.90$, and $W_{\text{th}} = 0.15$,m. It's worth noting that there is a discontinuity in the northern conductors due to missing data in the original dataset. Notably, the TPF effectively eliminates all non-conductor points, aside from a few scattered points at lower levels and surrounding the catenaries. The algorithm achieves this by iteratively running in the initial filter regions to detect top catenaries of multi-level powerlines. Ultimately, all conductors within the scene are accurately preserved for subsequent processing.

The point cloud generated by the TPF underwent the adapted Hough transform applied to its 2D projection onto the XOY plane, employing as parameters $A_s = 0.1°$, $G_s = 0.1$ m, and $n_{\min} = 25$. Figure 5.21(a) illustrates each detected line, distinguished by different colors. Notably, the Hough transform effectively eliminates spurious points, retaining those predominantly associated with electric conductors. An interesting situation, absent in previous literature, emerges in the bottom left of Figure 5.21(a): a pair of conductors are anchored at one end in a horizontal plane and at the other end in a vertical plane. A detailed view is provided in Figure 5.21(b). Despite the conductors being discernible, minor discrepancies occur where the conductor plane rotates along its axis. Such behavior is expected, as the Hough transform may encounter challenges with such geometries. Notably, the conductors from the electrical transformer are accurately identified as well.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

(a) Diablo Canyon orthomosaic.



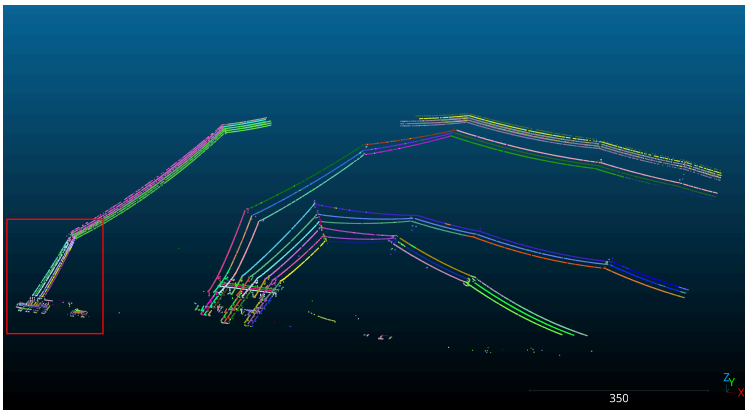

(b) Diablo Canyon point cloud.

**Figure 5.20:** Diablo Canyon dataset.

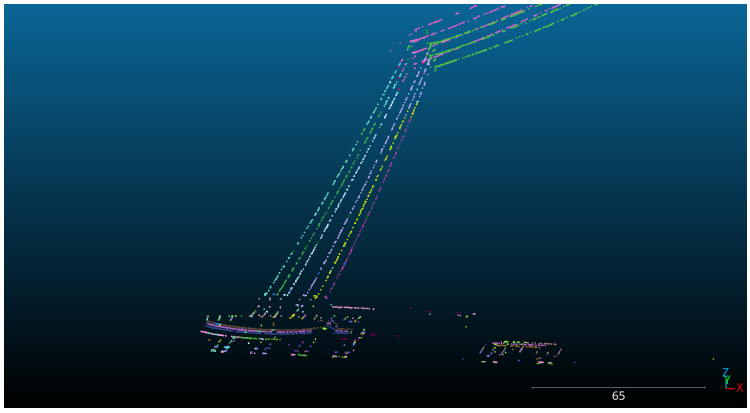
Within the results of the Hough transform, an interesting effect emerges where certain points are dispersed among parallel conductors associated with another detected conductor. This phenomenon arises from the algorithm's reliance on the number of points to identify straight lines, disregarding their proximity. Consequently, points aligned with a previously identified line are included in a different conductor. This effect, present across the entire point cloud in Figure 5.21(a), was overlooked in prior literature as it primarily concentrated on detecting individual powerlines in isolation.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA




(a) Hough transform results on Diablo Canyon.




(b) Detail of powerline going from vertical to horizontal distribution.

**Figure 5.21:** Results of the Hough transform applied on the Diablo Canyon dataset.

### 5.2.2 Results on the Lugo dataset

The experiment employed parameters identical to those in the previous case to showcase the algorithm's versatility across diverse datasets, encompassing mid- to low-voltage powerlines with no multiple conductors sharing the same vertical plane. The results (Figure 5.24(a)) demonstrate precise preservation of all conductors following the initial filter but with some presence of spurious points. Subsequently, the Hough transform (Figure 5.24(b)) erroneously labels certain points due to alignment with longer conductors. However, these issues can be readily rectified by analyzing the



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

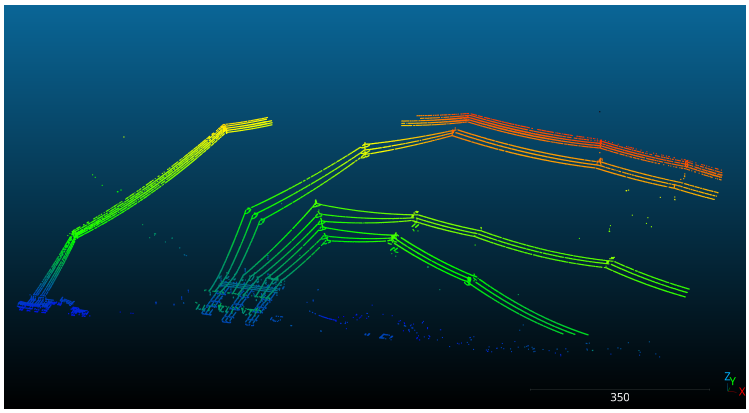



**Figure 5.22:** Results of the TPF applied to the Diablo Canyon dataset.

local neighborhood of each point. A closer examination of a low-voltage powerline intersecting beneath a mid-voltage powerline is depicted in Figure 5.24(c), showcasing the algorithm's capability to detect intersecting powerlines, which is not covered in the reviewed literature.

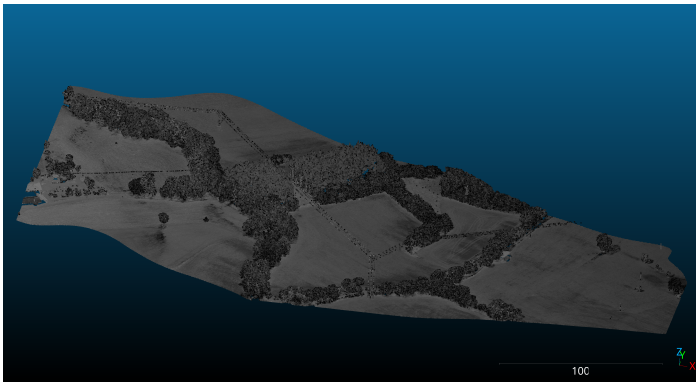
### 5.2.3 Results on the DALES dataset

The experiment used the same parameters as those employed with Diablo Canyon. Note that across the selected tiles, a spectrum of powerlines, ranging from high- to low-voltage, is present. Notably, the algorithm correctly detected all powerlines without needing adjustments to the initial parameters. However, in dales19, certain segments of a powerline were filtered out by the initial stage of the algorithm (Figure 5.26(a)). This is due to the absence of points below those particular powerline slices, as typical LiDAR scanners typically lack wavelengths suitable to carry out water detection. Consequently, during candidate selection, the value of $C_i$ fails to surpass the required threshold, given the set $H_i$ is empty. In dales10, a low-voltage powerline situated within an urban area was accurately identified (Figure 5.26(b)). Despite its proximity to adjacent vegetation, the algorithm successfully discerned the powerline without encountering any impediments. In dales17, a scene featuring both high- and low-voltage powerlines was encountered, with the latter crossing beneath the former. The algorithm identified both powerlines with precision. The outcomes of the algorithm on this tile are depicted in Figure 5.26(c). Finally, dales10 serves as proof of the algorithm's capability in detecting powerlines of different voltages within urban settings. In this particular tile, three distinct powerlines, each with differing voltage levels, run in parallel (Figure 5.26(d)), with the low-voltage variant constituting a



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

(a) Lugo orthomosaic.



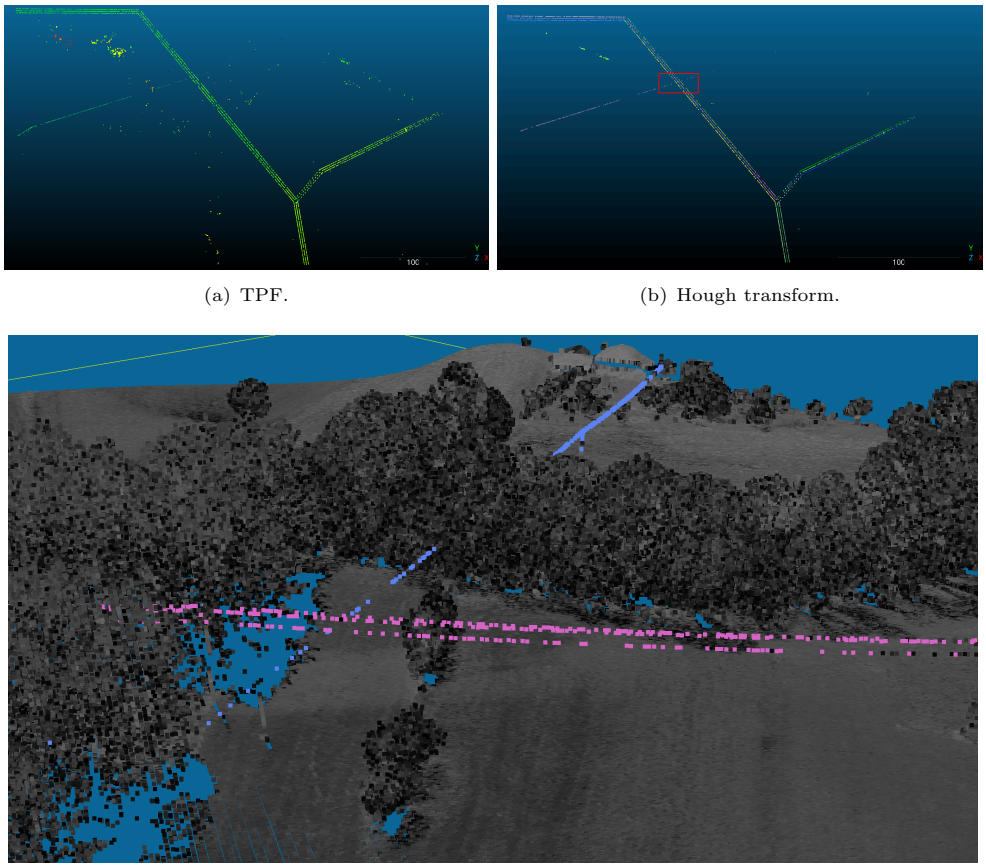

(b) Lugo point cloud.

**Figure 5.23:** Lugo dataset.

multilevel powerline. The algorithm effectively detected all the powerlines present. However, it's worth noting that distribution powerlines responsible for supplying electricity to households were not identified in this instance (Figure 5.26(e)) because their distance to the ground is lower than the $H_{\text{th}}$ parameter used in the experiment, or the amount of points in the powerline is not greater than $n_{\text{min}}$.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(a) TPF.

(b) Hough transform.

(c) Detail of crossing powerlines.

**Figure 5.24:** Different stages of the powerline detection algorithm applied to the Lugo dataset.

### 5.2.4 Quantitative evaluation

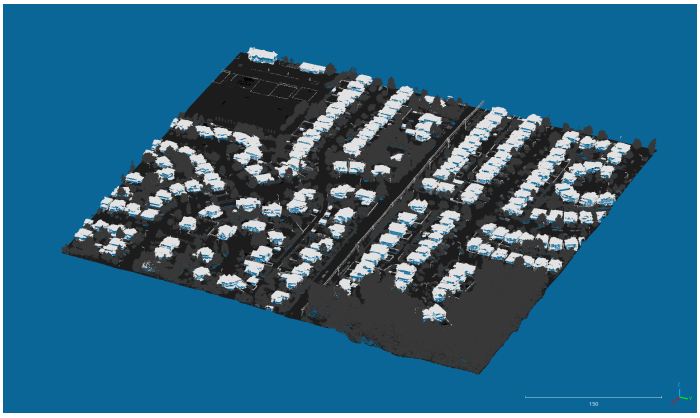
In order to evaluate the performance of the proposed algorithm, the results were compared with the ground truth corresponding to each dataset. Diablo Canyon and Lugo datasets were labeled by hand using CloudCompare [219]. In the DALES dataset, the labels corresponding to powerlines' conductors were changed from 5 to 14 to comply with the LAS Specification v1.4 [67] and to unify the evaluation method with the rest of the datasets.

When evaluating the performance of the powerline classification, several metrics are used: recall, precision, specificity, accuracy, and $F_1$-score [220].



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

(a) DALES orthomosaic.



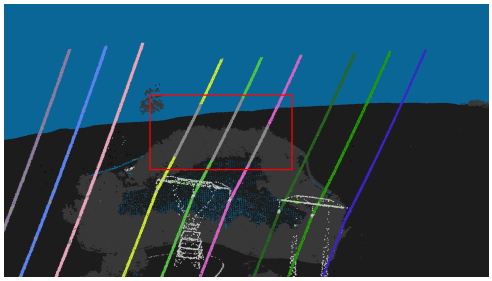
(b) DALES point cloud.

**Figure 5.25:** DALES dataset.

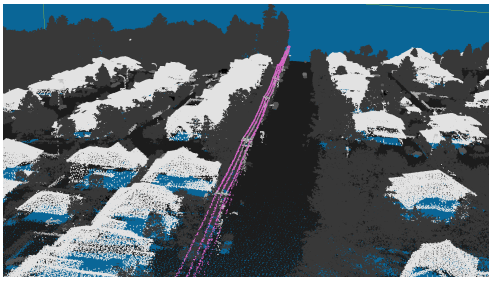
Recall (Equation 5.18), also known as sensitivity or true positive rate, measures the proportion of true positives (TP) correctly identified by a classifier, indicating its ability to classify points while minimizing false negatives (FN) correctly. Precision (Equation 5.19), on the other hand, assesses the accuracy of positive predictions made by the classifier, representing the proportion of TP among all positive predictions and indicating the classifier's ability to minimize false positives. Specificity (Equation 5.20), or the true negative rate, evaluates the classifier's capability to identify true negatives (TN), thereby minimizing false positives (FP). Accuracy (Equation 5.21) provides an overall measure of a classifier's correctness by considering the ratio of correctly predicted cases to the total number of cases. Lastly, the $F_1$-score (Equation 5.22), calculated as the harmonic mean of precision and recall, offers a balanced measure of



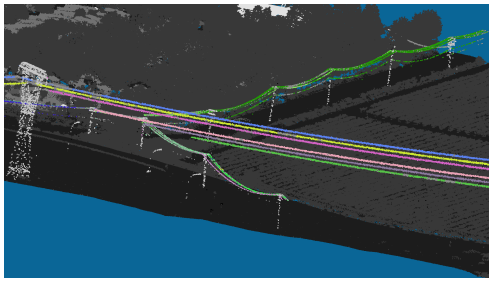
USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



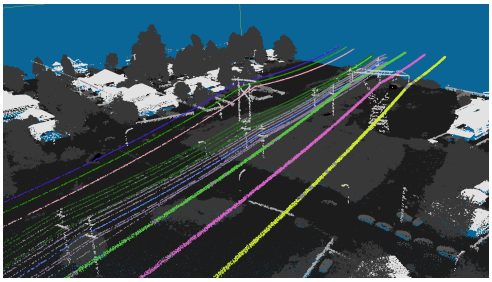
(a) False negatives over water.



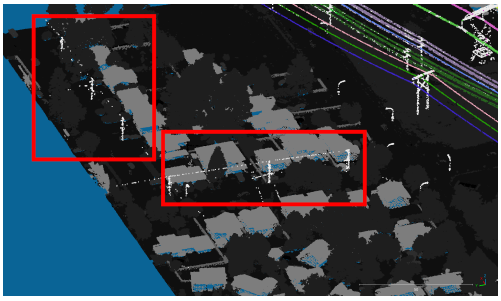
(b) Mid-voltage powerline nearby high vegetation.



(c) Low-voltage powerline crossing below a mid-voltage powerline.



(d) Powerlines with different voltages run in parallel in an urban environment.



(e) Not detected low-voltage powerlines.

**Figure 5.26:** Results on Dales dataset.

a classifier's performance, particularly useful for imbalanced datasets as it accounts for both false positives and false negatives. These metrics collectively provide a comprehensive assessment of a classifier's strengths and weaknesses, aiding in informed decision-making and model refinement.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.18}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.19}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{5.20}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{5.21}$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.22}$$

Regarding the evaluation of the powerline characterization, the Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and $\text{R}^2$ are used since they are commonly used metrics to assess performance in regression models. RMSE quantifies the average magnitude of errors between predicted and actual values, penalizing larger errors more heavily than smaller ones (Equation 5.23). Lower RMSE values indicate better model performance, with predictions closer to the actual values. Similarly, MAE measures the average absolute magnitude of errors, providing another perspective on prediction accuracy (Equation 5.24). A lower MAE suggests better model performance, with smaller deviations between predicted and actual values. Finally, $R^2$ represents the proportion of variance in the dependent variable explained by the independent variables (Equation 5.25). Higher $R^2$ values indicate a better fit of the regression model to the data, capturing a larger portion of the variability in the response variable. Together, these metrics offer valuable insights into the accuracy and goodness of fit of regression models, aiding in model evaluation and comparison.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5.23}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{5.24}$$

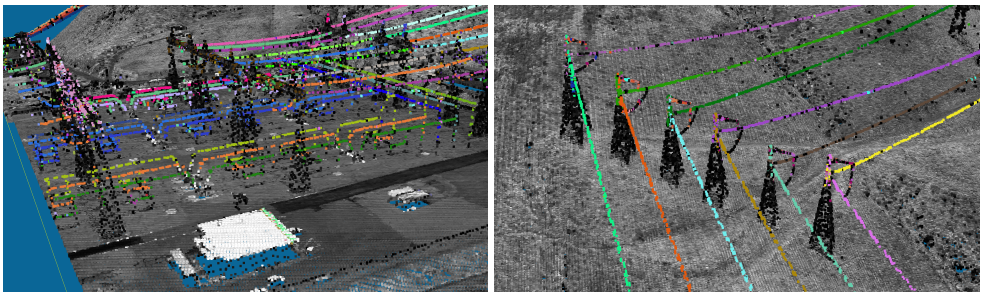
$$\text{R}^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.25}$$

where $y_i$ is the actual value, $\hat{y}_i$ is the predicted value using the model, $\bar{y}$ is the mean of the actual values, and $n$ is the number of samples.

For each dataset, the average of these metrics computed for all detected powerlines is presented in Table 5.3.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(a) Power transformer. The detected low lines across the power transformer were not labeled as powerlines.

(b) Pylons. The upper parts of the pylons are sparse enough to pass the first stage of the algorithm.

**Figure 5.27:** Detailed view of the Diablo Canyon dataset.

**Table 5.3:** Performance Metrics

| | **Diablo Canyon** | **Lugo** | **dales10** | **dales17** | **dales19** | **dales36** |
|---|---|---|---|---|---|---|
| TP | 33 917 | 2 281 | 65 384 | 46 983 | 70 706 | 6 664 |
| FP | 4 394 | 30 | 290 | 164 | 1 027 | 150 |
| TN | 8 053 473 | 2 517 881 | 12 877 758 | 12 095 895 | 14 115 326 | 11 941 821 |
| FN | 5 982 | 211 | 10 942 | 5 758 | 9 479 | 1 397 |
| Recall | 0.850 | 0.915 | 0.857 | 0.891 | 0.882 | 0.827 |
| Precision | 0.885 | 0.987 | 0.996 | 0.997 | 0.986 | 0.978 |
| Specificity | 0.999 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Accuracy | 0.999 | 1.000 | 0.999 | 1.000 | 0.999 | 1.000 |
| $F_1$-score | 0.867 | 0.950 | 0.921 | 0.941 | 0.931 | 0.896 |
| RMSE (m) | $0.080 \pm 0.015$ | $0.024 \pm 0.003$ | $0.128 \pm 0.096$ | $0.308 \pm 0.054$ | $0.221 \pm 0.066$ | $0.079 \pm 0.007$ |
| MAE (m) | $0.066 \pm 0.012$ | $0.019 \pm 0.003$ | $0.111 \pm 0.087$ | $0.262 \pm 0.035$ | $0.188 \pm 0.064$ | $0.066 \pm 0.008$ |
| $R^2$ | $0.999 \pm 0.001$ | $0.996 \pm 0.004$ | $0.988 \pm 0.012$ | $0.808 \pm 0.372$ | $0.991 \pm 0.001$ | $0.958 \pm 0.025$ |

The results for the Diablo Canyon dataset show an extremely high specificity and accuracy. This is due to the relation between the number of points belonging to powerlines' conductors and the total number of points in the dataset since only 0.493% of the existing points belong to wires. The values for recall and precision are explained by the number of FN (False Negatives) and FP (False Positives), respectively. The vast majority of FP correspond to the structure located at the bottom of the scene, which belong to a power transformer (Figure 5.27(a)). There are several hanging structures that are detected by the algorithm that were not considered powerlines since their nature was not known when labeling. The FN are caused indirectly by the low density of the dataset. The FN mainly belong to the pylons, which usually, in this dataset, are composed of sparse points that are not filtered in the first stage of the algorithm (Figure 5.27(b)). Since the points are aligned with the conductors



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

themselves, they are included as part of the detected catenaries.

Regarding the powerline characterization, the model demonstrates strong performance, as indicated by the low RMSE and MAE values of 0.080 m and 0.066 m, respectively. This suggests that the model's predictions are close to the actual values on average, with little deviation. Additionally, the high $R^2$ value of 0.999 indicates that the model explains nearly all the variance in the data, reflecting an excellent fit.

As for the results in Lugo, the specificity and accuracy are the highest (they are rounded to the third decimal) since the values of FP and FN are negligible with respect to TN (True Negatives). Compared with Diablo Canyon, the recall and precision are higher in this case since the dataset contains only mid and low-voltage powerlines. This kind of powerlines typically have simpler pylons whose points are correctly managed by our algorithm. Also, the density in this point cloud doubles the density of the Diablo Canyon dataset, which helps to detect the powerlines more accurately. In this dataset, the FN are mostly embedded in the correctly detected catenaries, as shown in Figure 5.28. This is a direct consequence of the parameters used in the Hough transform. The line detection is fine enough to add the points marked in Figure 5.28 as part of an independent line. Since each of those lines contains fewer points than $n_{\min}$, they are not detected as a straight line. Reducing the fineness of the parameters leads to those points being correctly detected at the expense of increasing the number of false positives. The catenary modeling algorithm is able to deal with missing points in a given catenary but not with points incorrectly classified as such, so these false negatives are assumable.

The performance for the Lugo dataset depicts outstanding model accuracy, with the lowest RMSE and MAE values among the analyzed sites, at 0.024 m and 0.019 m, respectively. These values indicate low errors in the model's predictions, showcasing its precision in estimating the target variable. Furthermore, the $R^2$ value of 0.996 suggests that the model accounts for a significant portion of the variance in the data, indicating a robust fit. Lugo stands out as a site where the model performs exceptionally well, demonstrating its reliability in predicting outcomes accurately.

Our algorithm exhibited competitive performance on various tiles of the DALES dataset for powerline detection. The recall scores ranged from 0.827 to 0.891, indicating the model consistently identified a substantial portion of true powerline points across different tiles. Precision remained consistently high, with values between 0.978 and 0.997, demonstrating a low rate of false positives. This suggests the algorithm effectively distinguished powerlines from other objects. Some of the detected conductors are labeled as pylons in the ground truth, as shown in Figure 5.29. Note that this slightly reduces the precision value of this dataset.

The model fitting in dales10 and dales19 exhibit relatively low RMSE and MAE values, indicating precise predictions, dales17 and dales36 show higher values, suggesting greater prediction errors. Notably, dales17 exhibits a notably lower $R^2$ value of 0.808, indicating that the model explains a smaller proportion of the variance in the data compared to other sites. Despite this, all DALES datasets display overall



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

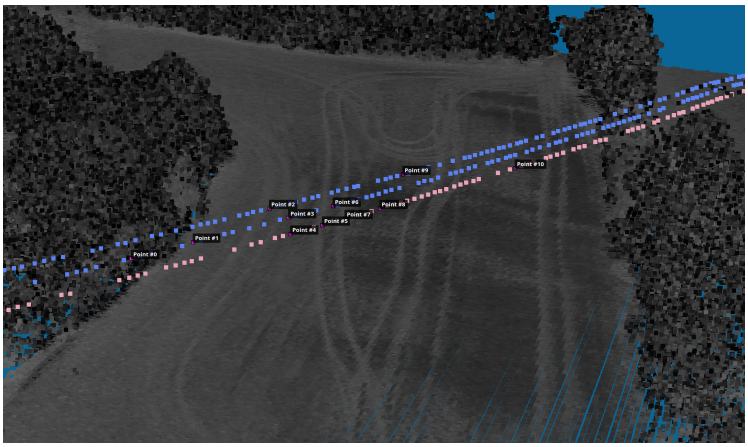



**Figure 5.28:** Detail of false negatives in the Lugo dataset.

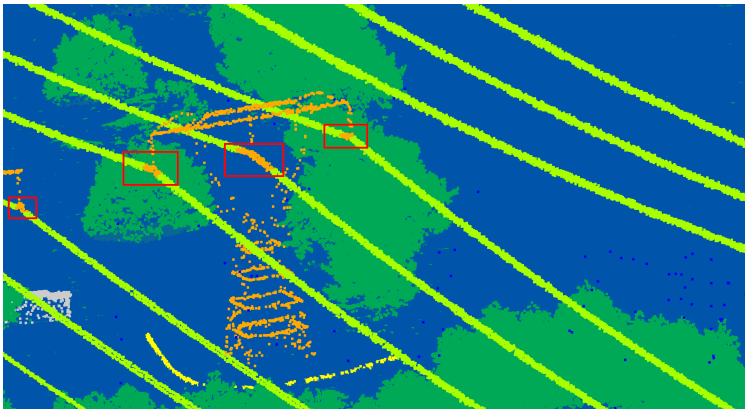


**Figure 5.29:** Detail of mislabeled conductors as pylons in the DALES dataset.

high accuracy, with $R^2$ values ranging from 0.958 to 0.991, highlighting the model's capability to capture the underlying trends in the datasets. These lower values can be explained by the nature of our algorithm, which uses the lower points in each catenary to perform the fit. In the Dales dataset, the wires could be as thick as 0.5 m, which explains the relatively high values for RMSE and MAE.

The specificity remained perfect for all subsets, signifying a flawless ability to differentiate powerlines from other elements within the scene. However, it's essential to acknowledge that a perfect specificity is due to the class imbalance in the dataset, where powerlines are a small proportion compared to other elements, as discussed in



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

the previous datasets.

Overall accuracy scores were very high, ranging from 0.999 to 1.000. However, it is important to consider the $F_1$-score, which balances precision and recall. Scores ranged from 0.896 to 0.941, demonstrating a good trade-off between these metrics. The variation is due to the wide range of different powerlines present across all tiles that were detected using the same parameters for the algorithm.

A comparison of the results with other state-of-the-art methods is shown in Table 5.4. The values of our method correspond to the average of the scores across all the tested datasets. The totality of the reviewed algorithms works with point clouds acquired specifically to detect and characterize powerlines, usually with flights parallel to the powerlines themselves, so their presence and position are known beforehand. Most methods achieve a precision higher than 0.9, which indicates that the state-of-the-art methods have a low rate of miss-classifying non-powerline points as powerlines. Our method achieves a precision of 0.972, which is slightly lower than the best value in the table. Secondly, all methods have a recall higher than 0.85 while our value is slightly lower. In any case, a direct comparison between methods is not possible due to the different characteristics of the acquisition methods and the datasets. As far as we know, this work is the first to use such a variety of datasets and types of powerlines to test the algorithm. The results show that our proposal is able to detect powerlines of any voltage, with less dense datasets, with any kind of pylon, and with a high degree of precision.

Regarding the RMSE, we achieved a better value than the only work found that uses ALS for reconstructing the powerlines. The other values were taken using MLS, implying a higher degree of precision in the point cloud, so a direct comparison is impossible.

An interesting work that uses ALS for powerline classification is proposed by Rousel et al. [221]. In this work, the results are quantified in terms of correctly classified conductor length and the number of correctly detected pylons. Based on this, they extrapolate the results assuming a constant density across all datasets to a point-level classification. This is the reason why we have not included this work in the comparison table. They reported results of 0.997 for the precision and 0.989 for the recall metrics. Furthermore, their method relies on having a detailed map of the position of the powerlines available when our proposal does not need such information, in addition to being agnostic with respect to the presence of powerlines in the analyzed point cloud.

### 5.2.5 Parameters sensitivity analysis

The sensitivity of the algorithm to the parameters was tested using the Diablo Canyon dataset. The parameters considered in this study were $R$, $H_{\text{th}}$, $W_{\text{th}}$, $C_{\text{th}}$, and $n_{\text{min}}$. Several values for each parameter were chosen to perform the analysis, and the combination of all parameters was tested, which yielded up to 1 024 experiments. The representation of each parameter is done by fixing the rest of the parameters to its



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Table 5.4:** Comparison of several powerline classification methods.

| Method | Platform | Corridor | Density (pts/m²) | Precision | Recall | $F_1$-Score | RMSE (m) |
|---|---|---|---|---|---|---|---|
| Dihkan et al. [181] | UAV | Yes | 17 | 0.971 | 0.974 | 0.976 | - |
| Huang et al. [182] | UAV | Yes | 50-90 | 0.990 | 0.992 | 0.991 | - |
| Chen et al. [185] | UAV | Yes | 35 | 0.965 | 0.948 | 0.956 | - |
| Shokri et al. [222] | MLS | Yes | - | **1.000** | 0.953 | 0.976 | 0.05 |
| Guan et al. [186] | MLS | Yes | 286 | 0.990 | 0.920 | 0.954 | 0.05 |
| Wang et al. [223] | MLS | Yes | - | 0.958 | 0.914 | 0.935 | - |
| McLaughlin [224] | ALS | Yes | - | 0.869 | - | - | - |
| Kim et al. [47] | ALS | Yes | 30 | 0.905 | 0.931 | 0.918 | - |
| Zhu et al. [225] | ALS | Yes | 55 | 0.933 | 0.980 | 0.956 | - |
| Ortega et al. [183] | ALS | Yes | 25.5 | 0.994 | **0.996** | **0.995** | 0.194 |
| Guo et al. [226] | ALS | Yes | 60 | 0.935 | 0.887 | 0.910 | - |
| Guo et al. [227] | ALS | Yes | - | 0.890 | 0.860 | 0.875 | - |
| Our Method | ALS | **No** | **5-20** | 0.972 | 0.870 | 0.918 | 0.140 |

default value. Both the default values and the chosen study range are shown in Table 5.5. The ranges were chosen for reasonable parameter values based on regulations and on the authors' experience. The sensitivity analysis results are shown in Figure 5.30. For each subplot, the precision, recall, and $F_1$-score are shown as a function of the corresponding variable while fixing the other four variables to the default value.

For $R$, the precision sightly drops with increasing values of $R$, while the recall does the opposite. This compensates for the $F_1$-score, which remains almost constant. In the case of $H_{\text{th}}$, the recall value remains practically constant across executions, while the precision slightly improves, reaching the maximum for 3.5 m, which supports the choice of the default parameter. In the case of $W_{\text{th}}$, the value of the precision quickly drops from 0.5 m. The reason is that when increasing the $W_{\text{th}}$, the number of points of the $W_i$ subset is reduced, so the value of $C_i$ is greater. Thus, more points pass the first stage of the algorithm, yielding more false positives. The recall also increases for the same reason since more true positives are detected in the first stage. As for the $C_{\text{th}}$, the recall rapidly drops from 0.9. This is due to the strong restriction the parameter imposes on the points to pass the TPF, so a lot of points belonging to electric wires do not reach the second stage of the algorithm. Regarding the $n_{\text{min}}$, the precision increases with the parameter value, reaching a perfect value for the higher values of $n_{\text{min}}$, at the cost of not detecting most of the wires, which causes the recall, and therefore the $F_1$-score, to drop significantly.

The maximum obtained $F_1$-score is 0.874 for the values $R = 2.0$, $H_{\text{th}} = 3.0$, $W_{\text{th}} = 0.5$, $C_{\text{th}} = 0.95$ and $n_{\text{min}} = 10$.

### 5.2.6 Influence of the point density

The influence of the point density was analyzed using the Diablo Canyon dataset. The point cloud was downsampled 20 times, in evenly spaced steps, from the original density of 8.93 pts/m$^2$ to 0.89 pts/m$^2$. The classification results in terms of the density



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**Table 5.5:** Parameters range for the sensitivity analysis.

| Parameter | Range | Default value |
|---|---|---|
| $R$ | $[1.0, 2.0, 3.0, 4.0]$ | 1.0 |
| $H_{\text{th}}$ | $[1.0, 2.0, 3.0, 3.5]$ | 3.5 |
| $W_{\text{th}}$ | $[0.15, 0.25, 0.5, 1.0]$ | 0.15 |
| $C_{\text{th}}$ | $[0.8, 0.9, 0.95, 0.99]$ | 0.90 |
| $n_{\text{min}}$ | $[10, 25, 50, 100]$ | 25 |

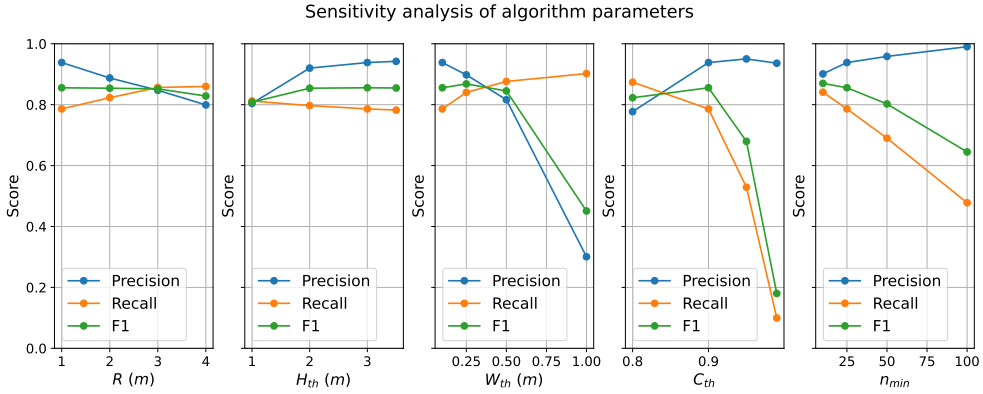



**Figure 5.30:** Sensitivity analysis of the algorithm parameters. The study was carried out on the Diablo Canyon dataset.

of the point cloud are shown in Figure 5.31. The curve $n_{\text{min}} = 25$ shows an almost constant $F_1$-score over $5\,\text{pts/m}^2$. The quality of the classification drops exponentially for low densities. Note that the algorithm parameters were unchanged for every density test. The drop in the $F_1$-score is mainly due to the fact that, below a certain density, the catenaries are made up of fewer points than $n_{\text{min}}$ and, therefore, are not detected by the Hough transform. When setting $n_{\text{min}} = 10$, the $F_1$-score remains almost constant until $3\,\text{pts/m}^2$.

## 5.3 Computational performance

To test the performance and scaling of the algorithm in manycore systems, a parallel C++20 implementation of the algorithm was developed and executed on the Diablo Canyon dataset.

Tests were carried out on Machine 2. Since the parallelization is based on a shared-memory model, only one node of the machine is used in the tests. Thread affinity was



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

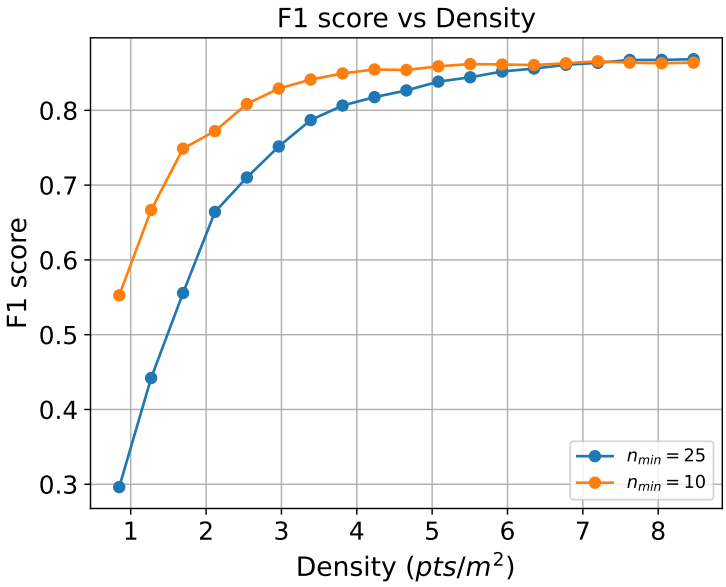



**Figure 5.31:** Influence of the point density on the algorithm. The study was carried out on the Diablo Canyon dataset.

set so that threads are located in the fewest NUMA nodes while still using one thread per core.

The stages of the method to be analyzed are the TPF and the Hough transform, see Figure 5.1, since the execution time of the post-processing is negligible compared to those. To measure the quality of the parallelization, besides execution times, we use the efficiency $\epsilon = \frac{S}{N}$, which is the ratio between the speedup $S$ and the number of threads used $N$. Note that $\epsilon \in [0, 1]$.

Figure 5.32 shows the execution times of the different phases of the algorithm, while Figure 5.33 shows the obtained speedup and efficiency. In the sequential implementation, powerlines are detected and processed in 37 s. When exploiting parallelism, the execution time is reduced to about 1.4 s seconds for each phase and less than 3 seconds in total with 32 cores.

The TPF algorithm scales particularly well, with efficiency above 80% up to 16 threads. With more cores, efficiency decreases because of the use of two NUMA nodes, making memory operations slower for remote threads. Similarly, the speedup of the Hough transform grows up to 16 threads and stalls from then on. Since the overhead of launching and synchronizing a pool of threads grows with the number of threads, there is a point where the benefits are not enough to compensate for that overhead. That



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

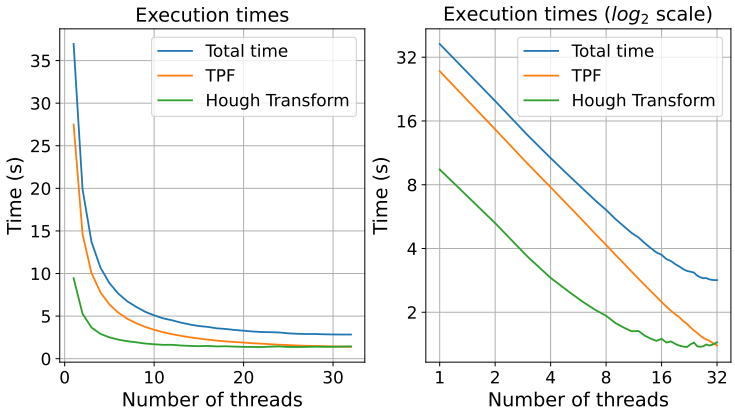



**Figure 5.32:** Execution times for Diablo Canyon dataset.

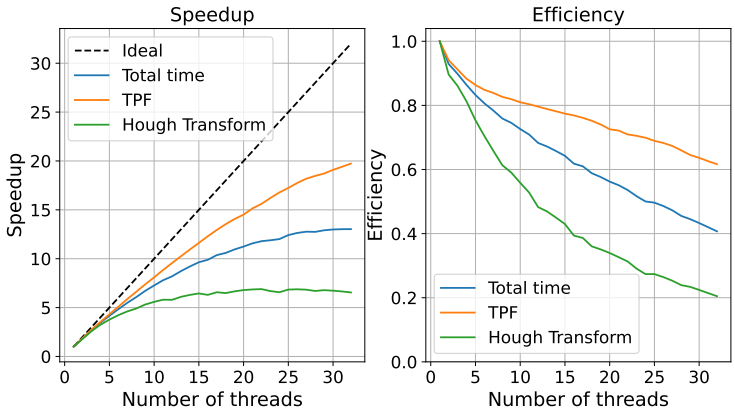



**Figure 5.33:** Speedup and efficiency for Diablo Canyon dataset.

affects the different stages of the Hough transform, such as maximum suppression, which already implies very short execution times for each call in sequential execution.

It is interesting to study the scalability of the algorithm with the size of the point cloud used. The TPF algorithm relies on the neighborhood search of each point, so its scalability is limited by the search algorithm, as explained in Chapter 3. The isoefficiency is shown in Figure 5.34. For any problem size, the TPF show a similar scalability as the one achieved in the neighborhood search.

Notwithstanding, the study of the Hough transform parallelization is interesting since it is the most computationally expensive part of the algorithm. Concretely, the



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

VES algorithm, since it involves two nested loops, as shown in Algorithm 5.2.

The size problem of the Hough transform is governed by two factors. Firstly, the number of points that pass the TPF stage, which in turn depends on parameters such as $H_{\text{th}}$ or $C_{\text{th}}$. Secondly, the number of votes cast by each point, which is directly controlled by the parameter $A_s$, as shown in Equation 5.6. The variation of these parameters will lead to a different number of points considered in the Hough transform, which allows the study of the scalability of the algorithm.

The isoefficiency matrix [168] of the Hough transform algorithm is shown in Figure 5.35, where the efficiency was computed using Equation 4.19. The results show that the algorithm scalability is practically independent of the problem size since the efficiency along each row remains unaltered. Regarding the efficiency with the number of threads, the algorithm scales well up to 16 threads, with an average efficiency of 0.74.

In order to carry out a deeper analysis of the isoefficiency, different stages of the Hough transform were measured independently: the voting procedure (Algorithm 5.1), the VES (Algorithm 5.2), and the procedure of finding maximum values in the accumulator. The corresponding isoefficiency matrices are shown in Figure 5.36, Figure 5.37, and Figure 5.38.

Regarding the voting procedure, a soft increase in efficiency with the problem size is observed. Again, the best efficiencies are located in the range of 1 to 16 threads. Particularly, the best efficiencies in this range of threads start from a problem size of 330 616 points, reaching superlinearity [169] for bigger problem sizes. Superlinearity is due to the data locality when the size of the available cache grows as more cores come into play. This causes fewer accesses to the memory with respect to the case when only one thread is used, as more data is available directly in the cache.

As for the find maximum procedure, the same scenario is observed, with a soft increase of the efficiency with the problem size. Up to 0.80 efficiency is reached with 16 cores, depending on the problem size.

Finally, the efficiency of the VES procedure sightly drops when increasing the problem size for any number of threads. The achieved efficiencies below 16 threads are very competitive, taking into account the complexity of this part of the algorithm. Overall, the efficiency decreases as the problem size and the number of threads increases.

In every isoefficiency matrix, the efficiency decreases more rapidly as the frontier of 16 threads is crossed. This is due to the architecture of the machine (see Figure A.2), where each NUMA node has 16 cores. The latency of accessing the memory for those threads located in other NUMA nodes is higher, which negatively affects the execution time of the algorithm and, therefore, its efficiency.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

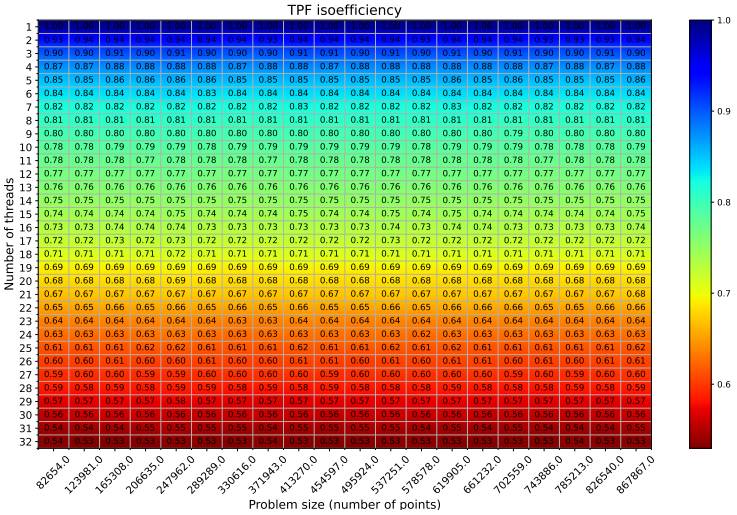



**Figure 5.34:** Isoefficiency matrix for the TPF algorithm.

## 5.4 Discussion

Some important aspects that can be analyzed when applying our proposal are the use of preprocessing stages to reduce the amount of input data, the influence of the parameters that rule the method, and the achievements obtained with different datasets, particularly DALES.

The TPF was designed not to rely on the use of a DTM, as it is not always available, and its generation may be computationally expensive depending on the generating algorithm chosen and the resolution used. In case the DTM is provided together with the dataset, it is possible to reduce the number of points to be checked as powerline candidates in the first stage of the algorithm by filtering out the ground points classified by using the information provided by the DTM. This would avoid the computational burden of applying the TPF to points belonging to the ground. Nevertheless, reducing the number of points to be checked in the TPF is not only applicable to DTMs but to any algorithm that can filter out other point classes, such as vegetation, which often are the most abundant in airborne point clouds in forested areas or buildings. The classification errors any of these algorithms may produce will be propagated to the TPF and, therefore, to the rest of the stages of the algorithm.

Regarding the TPF parameters, the default values for $R$, $H_{\text{th}}$, $W_{\text{th}}$, and $C_{\text{th}}$ are shown in Section 5.2.1. These values were chosen so the algorithm is valid for detecting



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

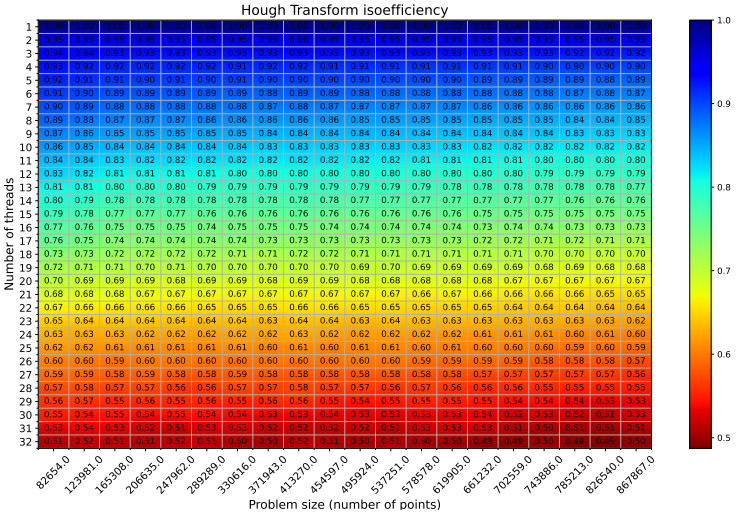



**Figure 5.35:** Isoefficiency matrix for the Hough transform algorithm.

powerlines of any voltage, but they can be adapted to the user's need. The value of $H_{\text{th}}$ can be thought of as the minimum height an electric conductor must have to be detected in this stage of the algorithm. Thus, this parameter can be set taking into account the local legislation of the analyzed area. The inclusion of $W_{\text{th}}$ was needed for improving the algorithm's performance in those cases where two or more conductors run in parallel so close to each other that they cannot be distinguished in an airborne point cloud. Also, for those powerlines with the highest voltages, the wires are thicker than usual. This, added to the fact that the greatest error in the LiDAR measurement occurs along the beam, can result in the cylinder created under the point being analyzed being populated by points of the conductor itself. The $W_{\text{th}}$ parameter prevents this situation. This parameter's ideal value is the wire's expected width, added to the average error of the sensor used. Since the algorithm must operate in areas where both the presence and the characteristics of a powerline are unknown, the value of this parameter has been chosen by excess. Finally, $C_{\text{th}}$ was chosen to be a ratio to make the algorithm independent of the point cloud density. A value of 1 would mean that the cylinder under the analyzed point must be empty to consider the point as a powerline candidate. The value of 0.9 gives some tolerance and allows the presence of a few points inside the cylinder, which yields the correct detection of powerlines very close to obstacles. None of these parameters has a direct impact on



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

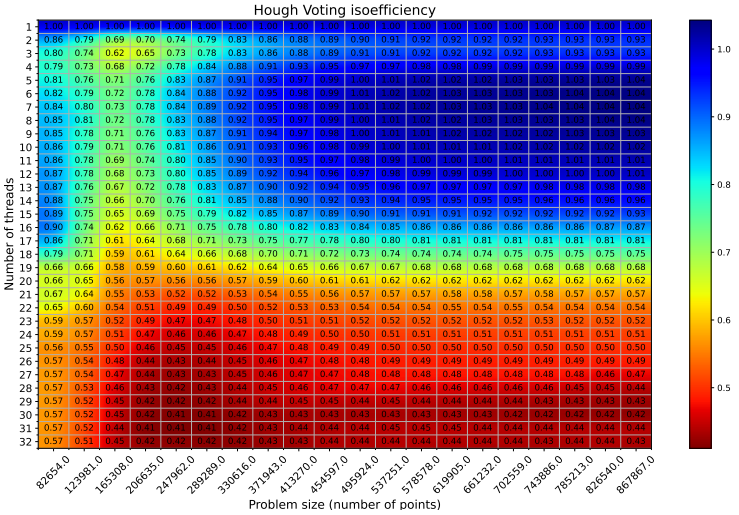



**Figure 5.36:** Isoefficiency matrix for the voting procedure.

the computational performance of the algorithm.

The parameters of the Hough transform ($A_s$, $G_s$, $n_{\min}$) control the resolution and the stop condition of the algorithm. While $A_s$ is used to control the angular resolution of the Hough transform, in combination with $G_s$, it determines the length of the longest single line that can be detected. When the distance between two voting points in the Hough transform is greater than $G_s \cdot \sin(A_s)$, they will have different bins in the accumulator for the same value of $\theta$, so the points will be part of different lines. The variation of $A_s$ deeply impacts the computational cost of the algorithm, as it directly controls the number of votes each point will cast. $n_{\min}$ is used to remove the lines with a low number of votes, which are likely to be spurious. This parameter does not have an impact on the computational performance.

Our study represents the first application, to the best of our knowledge, of a powerline detector on the well-known DALES dataset. Since its publication, this dataset has been the reference benchmark for airborne LiDAR data. The dataset is diverse enough to demonstrate the algorithm's effectiveness across a varied set of powerlines. The scarcity of publicly classified datasets for airborne LiDAR research is a common issue within the scientific community. This shortage hampers the validation and evaluation of algorithms in this field, often leading to manual labeling of datasets. Regarding the comparison of results with other works, a direct comparison has been



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

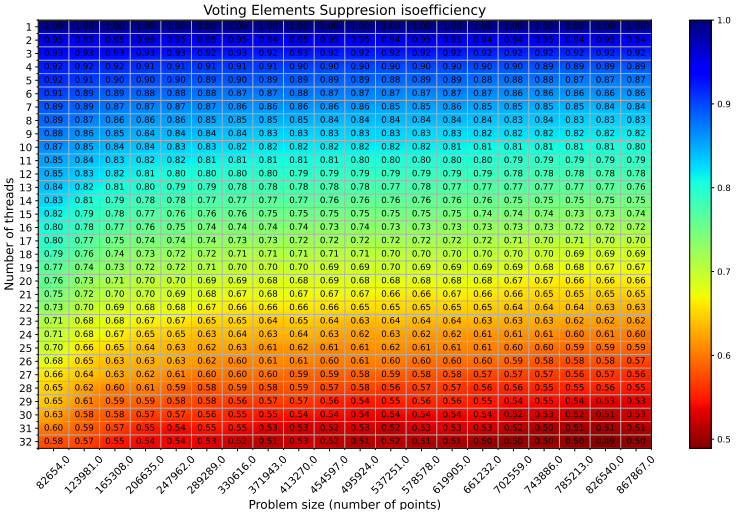



**Figure 5.37:** Isoefficiency matrix for the VES procedure.

challenging due to the lack of publicly available datasets typically utilized in previous studies. By utilizing DALES, we hope that future researchers directly compare their findings with our method under the same scenarios. The utilization of a widely accessible dataset like DALES serves to promote transparency and reproducibility and facilitates benchmarking efforts within the field. As more studies adopt standardized datasets, more meaningful comparisons and advancements in algorithmic approaches for airborne LiDAR analysis will be enabled.

## 5.5 Conclusions

This chapter presents a method to detect and characterize powerlines. The main contribution is that the method was designed to detect several ensembles of powerlines in an airborne point cloud that was not acquired specifically for this goal. Also, we have proven that the method is valid for detecting both high and low-voltage powerlines, even in the presence of overlapping, and the methods could be applied both to LiDAR and photogrammetric 3D point clouds.

Furthermore, when developing the Hough transform, we had to deal with the *non-maximum suppression* casuistic. Several authors have addressed NMS in the field of image-based object detection. We developed a non-parametric method that removes



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

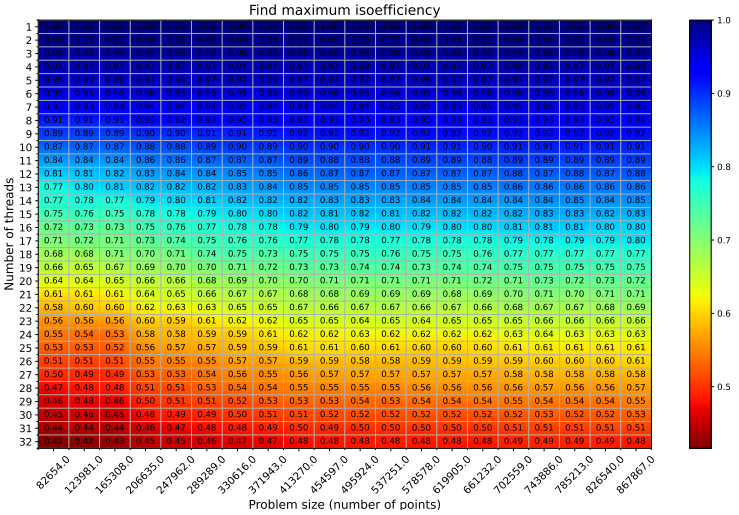



**Figure 5.38:** Isoefficiency matrix for the procedure of finding maximum values in the accumulator.

only the exact number of votes needed in the accumulator, and we proved that its parallel implementation is not only fast but very efficient in terms of taking advantage of all the physical cores that are available. Thus, the detection algorithm requires only a few parameters related to the approximate dimensions of the powerlines. These parameters could be tuned to adapt the algorithm to the detection of powerlines in different places since their geometry could vary from one region to another. An exhaustive analysis of their influence was performed, and default values were identified.

Since the 2-dimensional version of the Hough transform for point clouds was proposed, a method to determine whether a powerline is composed of several stacked vertical conductors was developed. With this method, not only are we able to determine the number of individual conductors that belong to a given powerline, but also to compute the catenary curve parameters even in conductors with point densities as low as $5\,\text{pts/m}^2$, or lack of points along their span.

Finally, a detailed study of the algorithm's computational performance was carried out, demonstrating that the method is very efficient in a manycore machine and has very high scalability.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

The contributions of this chapter were published in *IEEE* and were extracted from:

- M. Yermo, R. Laso, O. G. Lorenzo, J. C. Cabaleiro, T. F. Pena, F. F. Rivera, and D. L. Vilariño, "Powerline Detection and Characterization in General-Purpose Airborne LiDAR Surveys", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024. DOI: https://doi.org/10.1109/JSTARS.2024.3396522.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

## Chapter 6

# Conclusions

*This is Ripley, last survivor of the Nostromo, signing off.*
— Lt. Ellen Ripley, *Alien*.

This thesis addresses the problem of processing aerial LiDAR point clouds using HPC techniques. The first two chapters have served as an introduction to remote sensing and LiDAR technology, while the following three have presented the work carried out within the framework of this thesis. The main results obtained are summarized below, and future lines of research are proposed:

1. In Chapter 3, a study has been conducted on the most suitable data structures for executing the most used algorithm in LiDAR point cloud processing: neighbor search. An investigation of state-of-the-art libraries has been carried out, and the most suitable one for the purposes of this thesis has been selected. Furthermore, a study has been conducted on the scalability of neighbor search with different kernels when parallelized under the shared-memory paradigm. In this study, the best scheduling strategy and the best way to distribute threads on a given hardware have been determined. It has been demonstrated that neighbor search is highly parallelizable and that good results in terms of execution time and scalability can be achieved. Additionally, since most LiDAR point cloud processing algorithms depend on neighbor search, it has been shown that the maximum efficiency achievable in algorithms using this operation is limited by the efficiency of neighbor search. It is the opinion of this author that, due to the significant difference between the vertical dimension and the other dimensions in aerial LiDAR point clouds, using Octrees or $k$-d trees for neighbor search introduces significant overhead when these trees are very deep. This occurs when point clouds are of high density, which is becoming increasingly common due to the cost reduction of LiDAR data acquisition. Thus, designing a data structure that offers direct access to points considering only the two most extensive dimensions of the point cloud could outperform state-of-the-art methods

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

for fixed-radius neighbor search. Work is currently underway on such a data structure, and the results are promising.

2. In Chapter 4, an algorithm for pathfinding between two given points has been presented. Given a LiDAR point cloud, the algorithm calculates the least-cost route in terms of slope, terrain roughness, presence of vegetation, and presence of roads. To account for road presence, the point cloud must be classified beforehand. To consider vegetation presence, the available data must include NIR. From the point cloud segmentation, Voronoi tessellation is used to construct the graph nodes on which the optimal route will be calculated. Next, and this is where the most significant contribution is made compared to state-of-the-art methods, the graph edges are populated with all points contained within them. These points are used to calculate longitudinal and transverse slopes more accurately than any reviewed state-of-the-art method. Additionally, all points contribute to the calculation of roughness and vegetation presence. The cost functions used have been chosen to meet specific criteria, as there is little —in the case of slopes— or no —in the case of roughness and vegetation— literature on quantifying the effort of traversing terrain based on these parameters. The calculated route is always optimal in terms of cost, and this is guaranteed by the use of the Dijkstra algorithm. Furthermore, a scalability study of the algorithm on manycore machines has been conducted, demonstrating that algorithm parallelization is highly competitive in terms of efficiency.

3. In Chapter 5, an algorithm for detecting and characterizing powerlines has been presented. Given a general-purpose point cloud, the algorithm detects points susceptible to forming part of a powerline. This is done by studying the vertical distribution of points around each point in the cloud. Next, the Hough transform is applied to detect straight lines in the projection of points onto the 2D plane. Subsequently, for each line detected in the Hough transform, it is determined if there is more than one line in the same vertical plane, and they are clustered using a robust algorithm against low density and absence of points on a given conductor. Finally, the catenary curve defined by each conductor is modeled using numerical methods that do not require initial guesses for the curve parameters. The contributions of this algorithm compared to methods found in the state-of-the-art are several. First, the algorithm works on general-purpose point clouds, while most state-of-the-art methods require flights parallel to the powerline itself and very high densities to function correctly. Second, a method for suppressing maximums in the Hough transform that does not require parameters has been developed, so straight-line detection is done accurately. Third, a numerical method for modeling the catenary curve has been developed that does not require initial parameters, which can lead to method divergence if not carefully chosen, which is only sometimes possible. For algorithm validation, two point clouds have been manually labeled, and a reference benchmark in ALS has



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

been used: the DALES dataset. The results demonstrate that the algorithm can detect powerlines of any voltage in different scenarios, even if the lines cross over each other or change their rotation plane along their span. In quantitative terms, an average precision of 97.2 %, an average recall of 87.0 %, and an average $F_1$-score of 91.7 % have been achieved. Additionally, it has been demonstrated that the scalability when implementing the algorithm with parallelization in shared-memory is satisfactory in terms of both the number of threads and the problem size.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# Bibliography


[1] John F. Kennedy Library, *Aerial photograh of missiles in Cuba*, https://catalog.archives.gov/id/193926, Accessed on June 5, 2023.

[2] Kennedy Space Center, *60 years ago: First satellite image of Earth*, Kennedy Space Center Blog, Accessed on 24th May 2023, May 2023. [Online]. Available: https://www.kennedyspacecenter.com/blog/60-years-ago-first-satellite-image-of-earth.

[3] M. Yermo, F. F. Rivera, J. C. Cabaleiro, D. L. Vilariño, and T. F. Pena, "A fast and optimal pathfinder using airborne LiDAR data", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 183, pp. 482–495, 2022, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2021.11.014.

[4] M. Yermo *et al.*, "Powerline Detection and Characterization in General-Purpose Airborne LiDAR Surveys", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024. DOI: https://doi.org/10.1109/JSTARS.2024.3396522.

[5] M. Yermo *et al.*, "Automatic detection and characterisation of power lines and their surroundings using LiDAR data", *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W13, pp. 1161–1168, 2019. DOI: 10.5194/isprs-archives-XLII-2-W13-1161-2019. [Online]. Available: https://isprs-archives.copernicus.org/articles/XLII-2-W13/1161/2019/.

[6] A. Einstein, "On the quantum theory of radiation", *Physikalische Zeitschrift*, vol. 18, no. 18, pp. 121–128, 1917.

[7] T. H. Maiman, "Stimulated optical radiation in ruby", *Nature*, vol. 187, no. 4736, pp. 493–494, 1960, ISSN: 1476-4687. DOI: 10.1038/187493a0. [Online]. Available: https://doi.org/10.1038/187493a0.

[8] G. Petrie and C. K. Toth, "Introduction to laser ranging, profiling, and scanning", in *Topographic Laser Ranging and Scanning: Principles and Processing*, J. Shan, J. Shan, C. Toth, and C. Toth, Eds., CRC Press, 2009, ch. 1, pp. 29–


USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

88. DOI: 10.1201/9781420051438-1. [Online]. Available: https://doi.org/10.1201/9781420051438-1.

[9] RIEGL, *Riegl website*, http://www.riegl.com/, Accessed on May 11, 2023.

[10] Leica Geosystems, *Topographic LiDAR sensors*, https://leica-geosystems.com/products/airborne-systems/topographic-lidar-sensors, Accessed on May 11, 2023.

[11] Velodyne LiDAR, *Velodyne LiDAR website*, https://velodynelidar.com/, Accessed on May 11, 2023.

[12] Teledyne Optech, *Teledyne optech website*, https://www.teledyneoptech.com/en/home/, Accessed on May 11, 2023.

[13] LivoxTech, *Livoxtech website*, https://www.livoxtech.com/, Accessed on May 11, 2023.

[14] Y. Li and J. Ibanez-Guzman, "LiDAR for autonomous driving: The principles, challenges, and trends for automotive LiDAR and perception systems", *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, 2020. DOI: 10.1109/MSP.2020.2973615.

[15] N. Pfeifer and G. Mandlburger, "LiDAR data filtering and DTM generation", in *Topographic Laser Ranging and Scanning: Principles and Processing*, J. Shan, J. Shan, C. Toth, and C. Toth, Eds., CRC Press, 2009, ch. 11, pp. 349–378. DOI: 10.1201/9781420051438-11. [Online]. Available: https://doi.org/10.1201/9781420051438-11.

[16] K. Omasa, F. Hosoi, and A. Konishi, "3D LiDAR imaging for detecting and understanding plant responses and canopy structure", *Journal of Experimental Botany*, vol. 58, no. 4, pp. 881–898, Oct. 2006, ISSN: 0022-0957. DOI: 10.1093/jxb/erl142. eprint: https://academic.oup.com/jxb/article-pdf/58/4/881/1175299/erl142.pdf. [Online]. Available: https://doi.org/10.1093/jxb/erl142.

[17] M. Simard, N. Pinto, J. B. Fisher, and A. Baccini, "Mapping forest canopy height globally with spaceborne LiDAR", *Journal of Geophysical Research: Biogeosciences*, vol. 116, no. G4, 2011. DOI: 10.1029/2011JG001708.

[18] R. Boateng *et al.*, "Remote vegetation diagnostics in Ghana with a hyperspectral fluorescence LiDAR", *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 29, no. 4: Biophotonics, pp. 1–7, 2023. DOI: 10.1109/JSTQE.2023.3234022.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[19] J. Obu *et al.*, "Coastal erosion and mass wasting along the Canadian Beaufort Sea based on annual airborne LiDAR elevation data", *Geomorphology*, vol. 293, pp. 331–346, 2017, Permafrost and periglacial research from coasts to mountains, ISSN: 0169-555X. DOI: https://doi.org/10.1016/j.geomorph.2016.02.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169555X16300502.

[20] S. C. Palmer *et al.*, "Ultraviolet fluorescence LiDAR (UFL) as a measurement tool for water quality parameters in turbid lake conditions", *Remote Sensing*, vol. 5, no. 9, pp. 4405–4422, 2013, ISSN: 2072-4292. DOI: 10.3390/rs5094405. [Online]. Available: https://www.mdpi.com/2072-4292/5/9/4405.

[21] E. Kaartinen, K. Dunphy, and A. Sadhu, "LiDAR-based structural health monitoring: Applications in civil infrastructure systems", *Sensors*, vol. 22, no. 12, 2022, ISSN: 1424-8220. DOI: 10.3390/s22124610. [Online]. Available: https://www.mdpi.com/1424-8220/22/12/4610.

[22] A. F. Chase *et al.*, "Airborne LiDAR, archaeology, and the ancient Maya landscape at Caracol, Belize", *Journal of Archaeological Science*, vol. 38, no. 2, pp. 387–398, 2011, ISSN: 0305-4403. DOI: https://doi.org/10.1016/j.jas.2010.09.018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305440310003286.

[23] N. Masini *et al.*, "Medieval archaeology under the canopy with LiDAR. the (re)discovery of a medieval fortified settlement in southern Italy", *Remote Sensing*, vol. 10, no. 10, 2018, ISSN: 2072-4292. DOI: 10.3390/rs10101598. [Online]. Available: https://www.mdpi.com/2072-4292/10/10/1598.

[24] L. Wallace, A. Lucieer, C. Watson, and D. Turner, "Development of a UAV-LiDAR system with application to forest inventory", *Remote Sensing*, vol. 4, no. 6, pp. 1519–1543, 2012, ISSN: 2072-4292. DOI: 10.3390/rs4061519. [Online]. Available: https://www.mdpi.com/2072-4292/4/6/1519.

[25] G. Rivera, R. Porras, R. Florencia, and J. P. Sánchez-Solís, "LiDAR applications in precision agriculture for cultivating crops: A review of recent advances", *Computers and Electronics in Agriculture*, vol. 207, p. 107 737, 2023, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2023.107737. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169923001254.

[26] M. Jaboyedoff *et al.*, "Use of LiDAR in landslide investigations: A review", *Natural Hazards*, vol. 61, no. 1, pp. 5–28, 2012, ISSN: 1573-0840. DOI: 10.1007/s11069-010-9634-2. [Online]. Available: https://doi.org/10.1007/s11069-010-9634-2.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[27] B. Li *et al.*, "Application of LiDAR UAV for high-resolution flood modelling", *Water Resources Management*, vol. 35, no. 5, pp. 1433–1447, 2021, ISSN: 1573-1650. DOI: 10.1007/s11269-021-02783-w. [Online]. Available: https://doi.org/10.1007/s11269-021-02783-w.

[28] H. Guan *et al.*, "UAV-LiDAR aids automatic intelligent powerline inspection", *International Journal of Electrical Power & Energy Systems*, vol. 130, p. 106 987, 2021, ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2021.106987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061521002271.

[29] H. Zhou, C. Xu, X. Tang, S. Wang, and Z. Zhang, "A review of vision-laser-based civil infrastructure inspection and monitoring", *Sensors*, vol. 22, no. 15, 2022, ISSN: 1424-8220. DOI: 10.3390/s22155882. [Online]. Available: https://www.mdpi.com/1424-8220/22/15/5882.

[30] R. Roth, J. Thompson, *et al.*, "Practical application of multiple pulse in air (MPiA) LiDAR in large-area surveys", in *Proceedings of international archives of the photogrammetry, remote sensing and spatial information sciences*, vol. 37, 2008, pp. 183–188.

[31] H. Ku, "Notes on the use of propagation of error formulas", English, *Journal of Research of the National Bureau of Standards. Section C: Engineering and Instrumentation*, vol. 70C, p. 263, 4 Oct. 1966, ISSN: 0022-4316. DOI: 10.6028/jres.070C.025. [Online]. Available: https://dx.doi.org/10.6028/jres.070C.025.

[32] E. Jacobs and E. W. Ralston, "Ambiguity resolution in interferometry", *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-17, no. 6, pp. 766–780, 1981. DOI: 10.1109/TAES.1981.309127.

[33] J. A. Butt, D. Salido-Monzú, and A. Wieser, "Phase ambiguity resolution and mixed pixel detection in EDM with multiple modulation wavelengths", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 198, pp. 255–271, 2023, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2023.03.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271623000746.

[34] C. Zhang, Z. Zhang, Y. Tian, S. Y. Set, and S. Yamashita, "Comprehensive ranging disambiguation for amplitude-modulated continuous-wave laser scanner with focusing optics", *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–11, 2021. DOI: 10.1109/TIM.2020.3026441.

[35] G. Shi, F. Zhang, X.-H. Qu, and X. Meng, "High-resolution frequency-modulated continuous-wave laser ranging for precision distance metrology applications", *Optical Engineering*, vol. 53, no. 12, p. 122 402, 2014. DOI: 10.1117/1.OE.53.12.122402. [Online]. Available: https://doi.org/10.1117/1.OE.53.12.122402.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[36] A. P. Cabral and J. M. Rebordão, "Accuracy of frequency-sweeping interferometry for absolute distance metrology", *Optical Engineering*, vol. 46, no. 7, p. 073 602, 2007. DOI: 10.1117/1.2754308. [Online]. Available: https://doi.org/10.1117/1.2754308.

[37] J. J. Talamonti, R. B. Kay, and D. J. Krebs, "Numerical model estimating the capabilities and limitations of the Fast Fourier Transform technique in absolute interferometry", *Appl. Opt.*, vol. 35, no. 13, pp. 2182–2191, May 1996. DOI: 10.1364/AO.35.002182. [Online]. Available: https://opg.optica.org/ao/abstract.cfm?URI=ao-35-13-2182.

[38] L. Perret and P. Pfeiffer, "Sinusoidal nonlinearity in wavelength-sweeping interferometry", *Appl. Opt.*, vol. 46, no. 33, pp. 8074–8079, Nov. 2007. DOI: 10.1364/AO.46.008074. [Online]. Available: https://opg.optica.org/ao/abstract.cfm?URI=ao-46-33-8074.

[39] RIEGL, *RIEGL VQ-780 II-S datasheet*, http://www.riegl.com/uploads/tx_pxpriegldownloads/RIEGL_VQ-780II-S_Datasheet_2021-10-13.pdf, Accessed on May 11, 2023, p. 7.

[40] S. J. Koppal, "Lambertian reflectance", in *Computer Vision: A Reference Guide*, K. Ikeuchi, Ed. Boston, MA: Springer US, 2014, pp. 441–443, ISBN: 978-0-387-31439-6. DOI: 10.1007/978-0-387-31439-6_534. [Online]. Available: https://doi.org/10.1007/978-0-387-31439-6_534.

[41] J. H. Lambert, *Photometria*. 1760.

[42] Borb, *Inverse square law*, 2008. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=3816716.

[43] R. B. Miles, W. R. Lempert, and J. N. Forkey, "Laser Rayleigh scattering", *Measurement Science and Technology*, vol. 12, no. 5, R33, May 2001. DOI: 10.1088/0957-0233/12/5/201. [Online]. Available: https://dx.doi.org/10.1088/0957-0233/12/5/201.

[44] T. Wriedt, "Mie theory: A review", in *The Mie Theory: Basics and Applications*, W. Hergert and T. Wriedt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–71, ISBN: 978-3-642-28738-1. DOI: 10.1007/978-3-642-28738-1_2. [Online]. Available: https://doi.org/10.1007/978-3-642-28738-1_2.

[45] G. B. Airy, "On the diffraction of an object-glass with circular aperture", *Transactions of the Cambridge Philosophical Society*, vol. 5, p. 283, 1835.

[46] E. Hecht, *Optics*. Pearson Education India, 2012.

[47] S. Kim, I. Lee, and Y. J. Kwon, "Simulation of a Geiger-mode imaging LADAR system for performance assessment", *Sensors*, vol. 13, no. 7, pp. 8461–8489, 2013, ISSN: 1424-8220. DOI: 10.3390/s130708461. [Online]. Available: https://www.mdpi.com/1424-8220/13/7/8461.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[48] Livox Technology Co., Ltd., *Livox horizon user manual*, Accessed on 17th May, 2023, 2019. [Online]. Available: https://www.livoxtech.com/3296f540ecf5458a8829e01cf429798e/assets/horizon/Livox%20Horizon%20user%20manual%20v1.0.pdf.

[49] Q. Li, P. Yuan, Y. Lin, Y. Tong, and X. Liu, "Pointwise classification of mobile laser scanning point clouds of urban scenes using raw data", *Journal of Applied Remote Sensing*, vol. 15, no. 2, p. 024 523, 2021. DOI: 10.1117/1.JRS.15.024523. [Online]. Available: https://doi.org/10.1117/1.JRS.15.024523.

[50] Brillouin, Léon, "Diffusion de la lumière et des rayons X par un corps transparent homogène - influence de l'agitation thermique", *Ann. Phys.*, vol. 9, no. 17, pp. 88–122, 1922. DOI: 10.1051/anphys/192209170088. [Online]. Available: https://doi.org/10.1051/anphys/192209170088.

[51] B. Li, Q. Lin, and M. Li, "Frequency–angular resolving LiDAR using chip-scale acousto-optic beam steering", *Nature*, vol. 620, no. 7973, pp. 316–322, Aug. 2023, ISSN: 1476-4687. DOI: 10.1038/s41586-023-06201-6. [Online]. Available: https://doi.org/10.1038/s41586-023-06201-6.

[52] G. Petrie and C. K. Toth, "Terrestrial laser scanners", in *Topographic Laser Ranging and Scanning: Principles and Processing*, J. Shan, J. Shan, C. Toth, and C. Toth, Eds., CRC Press, 2009, ch. 12, pp. 29–88. DOI: 10.1201/9781420051438-2. [Online]. Available: https://doi.org/10.1201/9781420051438-2.

[53] X. Liang *et al.*, "Terrestrial laser scanning in forest inventories", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 115, pp. 63–77, 2016, Theme issue 'State-of-the-art in photogrammetry, remote sensing and spatial information science', ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2016.01.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271616000204.

[54] A. M. Esmorís, D. L. Vilariño, D. F. Arango, F.-A. Varela-García, J. C. Cabaleiro, and F. F. Rivera, "Characterizing zebra crossing zones using lidar data", *Computer-Aided Civil and Infrastructure Engineering*, vol. 38, no. 13, pp. 1767–1788, 2023. DOI: https://doi.org/10.1111/mice.12968. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/mice.12968. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12968.

[55] C. Wang, C. Wen, Y. Dai, S. Yu, and M. Liu, "Urban 3D modeling with mobile laser scanning: A review", *Virtual Reality & Intelligent Hardware*, vol. 2, no. 3, pp. 175–212, 2020, 3D Visual Processing and Reconstruction Special Issue, ISSN: 2096-5796. DOI: https://doi.org/10.1016/j.vrih.2020.05.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2096579620300395.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[56] N. Amarasingam, A. S. Ashan Salgadoe, K. Powell, L. F. Gonzalez, and S. Natarajan, "A review of UAV platforms, sensors, and applications for monitoring of sugarcane crops", *Remote Sensing Applications: Society and Environment*, vol. 26, p. 100 712, 2022, ISSN: 2352-9385. DOI: https://doi.org/10.1016/j.rsase.2022.100712. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352938522000209.

[57] S. Guan, Z. Zhu, and G. Wang, "A review on UAV-based remote sensing technologies for construction and civil applications", *Drones*, vol. 6, no. 5, 2022, ISSN: 2504-446X. DOI: 10.3390/drones6050117. [Online]. Available: https://www.mdpi.com/2504-446X/6/5/117.

[58] Q. Liu, S. Li, Z. Li, L. Fu, K. Hu, *et al.*, "Review on the applications of UAV-based LiDAR and photogrammetry in forestry.", *Scientia Silvae Sinicae*, vol. 53, no. 7, pp. 134–148, 2017.

[59] P. J. Kinzel, C. J. Legleiter, and J. M. Nelson, "Mapping river bathymetry with a small footprint green LiDAR: Applications and challenges", *JAWRA Journal of the American Water Resources Association*, vol. 49, no. 1, pp. 183–204, 2013. DOI: https://doi.org/10.1111/jawr.12008. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jawr.12008. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/jawr.12008.

[60] V. Wichmann, M. Bremer, J. Lindenberger, M. Rutzinger, C. Georges, and F. Petrini-Monteferri, "Evaluating the potential of multispectral airborne LiDAR for topographic mapping and land cover classification", *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-3/W5, pp. 113–119, 2015. DOI: 10.5194/isprsannals-II-3-W5-113-2015. [Online]. Available: https://isprs-annals.copernicus.org/articles/II-3-W5/113/2015/.

[61] L. R. Jarron, N. C. Coops, W. H. MacKenzie, P. Tompalski, and P. Dykstra, "Detection of sub-canopy forest structure using airborne LiDAR", *Remote Sensing of Environment*, vol. 244, p. 111 770, 2020, ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2020.111770. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425720301401.

[62] D. C. Mason, M. S. Horritt, N. M. Hunter, and P. D. Bates, "Use of fused airborne scanning laser altimetry and digital map data for urban flood modelling", *Hydrological Processes*, vol. 21, no. 11, pp. 1436–1447, 2007. DOI: https://doi.org/10.1002/hyp.6343. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/hyp.6343. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/hyp.6343.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[63] D. M. Winker *et al.*, "The CALIPSO mission: A global 3D view of aerosols and clouds", *Bulletin of the American Meteorological Society*, vol. 91, no. 9, pp. 1211–1230, 2010. DOI: https://doi.org/10.1175/2010BAMS3009.1. [Online]. Available: https://journals.ametsoc.org/view/journals/bams/91/9/2010bams3009_1.xml.

[64] National Geospatial-Intelligence Agency, *NGA.STND.0036_1.0.0_WGS84*, Accessed on 17th May, 2023, 2014. [Online]. Available: https://nsgreg.nga.mil/doc/view?i=4085.

[65] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)", in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[66] TerraSolid, *TerraSolid*, https://terrasolid.com/, Accessed: 2024.

[67] American Society for Photogrammetry and Remote Sensing, "ASPRS LAS Specification 1.4 - R14", ASPRS, Technical Report R14, 2014. [Online]. Available: https://www.asprs.org/wp-content/uploads/2019/03/LAS_1_4_r14.pdf.

[68] M. Isenburg, "LASzip: Lossless compression of LiDAR data", *Photogrammetric engineering and remote sensing*, vol. 79, no. 2, pp. 209–217, 2013.

[69] M. Weinmann, B. Jutzi, S. Hinz, and C. Mallet, "Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 105, pp. 286–304, 2015, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2015.01.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271615000349.

[70] Q. Xiang, Y. He, and D. Wen, "Adaptive deep learning-based neighborhood search method for point cloud", *Scientific Reports*, 2022. DOI: 10.1038/s41598-022-06200-z.

[71] V. Turau, "Fixed-radius near neighbors search", *Information Processing Letters*, 1991. DOI: 10.1016/0020-0190(91)90180-p.

[72] E. Fix and J. L. Hodges, "Discriminatory analysis. Nonparametric discrimination: Consistency properties", *International Statistical Review / Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989, ISSN: 03067734, 17515823. [Online]. Available: http://www.jstor.org/stable/1403797 (visited on 02/18/2024).

[73] T. Cover and P. Hart, "Nearest neighbor pattern classification", *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967. DOI: 10.1109/TIT.1967.1053964.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[74] S. Filin and N. Pfeifer, "Neighborhood systems for airborne laser data", *Photogrammetric Engineering and Remote Sensing*, vol. 71, no. 6, pp. 743–755, 2005. DOI: 10.14358/PERS.71.6.743. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-27444443641&doi=10.14358%2fPERS.71.6.743&partnerID=40&md5=20f5ee6ed3d6c95e80b3308e85e00007.

[75] L. Wang, Y. Huang, Y. Hou, S. Zhang, and J. Shan, "Graph attention convolution for point cloud semantic segmentation", in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 288–10 297. DOI: 10.1109/CVPR.2019.01054.

[76] H. Thomas, F. Goulette, J.-E. Deschaud, B. Marcotegui, and Y. LeGall, "Semantic classification of 3D point clouds with multiscale spherical neighborhoods", in *2018 International Conference on 3D Vision (3DV)*, 2018, pp. 390–398. DOI: 10.1109/3DV.2018.00052.

[77] T. Hackel, J. D. Wegner, and K. Schindler, "Fast semantic segmentation of 3D point clouds with strongly varying density", *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. III-3, pp. 177–184, 2016. DOI: 10.5194/isprs-annals-III-3-177-2016. [Online]. Available: https://isprs-annals.copernicus.org/articles/III-3/177/2016/.

[78] M. Pauly, R. Keiser, and M. Gross, "Multi-scale feature extraction on point-sampled surfaces", *Computer Graphics Forum*, vol. 22, no. 3, pp. 281–289, 2003. DOI: https://doi.org/10.1111/1467-8659.00675. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00675. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00675.

[79] N. Brodu and D. Lague, "3D terrestrial lidar data classification of complex natural scenes using a multi-scale dimensionality criterion: Applications in geomorphology", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 68, pp. 121–134, 2012, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2012.01.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271612000330.

[80] J. Niemeyer, F. Rottensteiner, and U. Soergel, "Contextual classification of lidar data and building object detection in urban areas", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 87, pp. 152–165, 2014, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2013.11.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271613002359.

[81] R. L. Cook, "Stochastic sampling in computer graphics", *ACM Trans. Graph.*, vol. 5, no. 1, pp. 51–72, Jan. 1986, ISSN: 0730-0301. DOI: 10.1145/7529.8927. [Online]. Available: https://doi.org/10.1145/7529.8927.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[82] E. Sevgen and S. Abdikan, "Classification of large-scale mobile laser scanning data in urban area with LightGBM", *Remote Sensing*, vol. 15, no. 15, 2023, ISSN: 2072-4292. DOI: 10.3390/rs15153787. [Online]. Available: https://www.mdpi.com/2072-4292/15/15/3787.

[83] Y. Tian, W. Song, L. Chen, Y. Sung, J. Kwak, and S. Sun, "A fast spatial clustering method for sparse LiDAR point clouds using GPU programming", *Sensors*, vol. 20, no. 8, 2020, ISSN: 1424-8220. DOI: 10.3390/s20082309. [Online]. Available: https://www.mdpi.com/1424-8220/20/8/2309.

[84] X. Zeng and W. He, "GPGPU-based parallel processing of massive LiDAR point cloud", in *MIPPR 2009: Medical Imaging, Parallel Processing of Images, and Optimization Techniques*, F. Zhang and F. Zhang, Eds., International Society for Optics and Photonics, vol. 7497, SPIE, 2009, p. 749 716. DOI: 10.1117/12.833740. [Online]. Available: https://doi.org/10.1117/12.833740.

[85] J. L. Bentley, "Multidimensional binary search trees used for associative searching", *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975, ISSN: 0001-0782. DOI: 10.1145/361002.361007. [Online]. Available: https://doi.org/10.1145/361002.361007.

[86] J. Behley, V. Steinhage, and A. B. Cremers, "Efficient radius neighbor search in three-dimensional point clouds", in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3625–3630. DOI: 10.1109/ICRA.2015.7139702.

[87] S. M. Omohundro, *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[88] A. Guttman, "R-trees: A dynamic index structure for spatial searching", in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '84, Boston, Massachusetts: Association for Computing Machinery, 1984, pp. 47–57, ISBN: 0897911288. DOI: 10.1145/602259.602266. [Online]. Available: https://doi.org/10.1145/602259.602266.

[89] E. Che and M. J. Olsen, "An efficient framework for mobile lidar trajectory reconstruction and mo-norvana segmentation", *Remote Sensing*, vol. 11, no. 7, 2019, ISSN: 2072-4292. DOI: 10.3390/rs11070836. [Online]. Available: https://www.mdpi.com/2072-4292/11/7/836.

[90] W. Wang, Y. Zhang, G. Ge, Q. Jiang, Y. Wang, and L. Hu, "A hybrid spatial indexing structure of massive point cloud based on octree and 3D R*-Tree", *Applied Sciences*, vol. 11, no. 20, 2021, ISSN: 2076-3417. DOI: 10.3390/app11209581. [Online]. Available: https://www.mdpi.com/2076-3417/11/20/9581.

[91] C. Luo, X. Li, N. Cheng, H. Li, S. Lei, and P. Li, *Mvp-net: Multiple view pointwise semantic segmentation of large-scale point clouds*, 2022. arXiv: 2201.12769 [cs.CV].



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[92] Y. Feng, M. Cen, and T. Zhang, "An algorithm of fast index constructing and neighbor searching for 3D LiDAR data", in *PIAGENG 2013: Intelligent Information, Control, and Communication Technology for Agricultural Engineering*, H. Tan, Ed., International Society for Optics and Photonics, vol. 8762, SPIE, 2013, 87620Y. DOI: 10.1117/12.2019658. [Online]. Available: https://doi.org/10.1117/12.2019658.

[93] J. Li, B. M. Chen, and G. H. Lee, "So-net: Self-organizing network for point cloud analysis", in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9397–9406. DOI: 10.1109/CVPR.2018.00979.

[94] X. Zhan, Y. Cai, and P. He, "A three-dimensional point cloud registration based on entropy and particle swarm optimization", *Advances in Mechanical Engineering*, vol. 10, no. 12, p. 1 687 814 018 814 330, 2018. DOI: 10.1177/1687814018814330. eprint: https://doi.org/10.1177/1687814018814330. [Online]. Available: https://doi.org/10.1177/1687814018814330.

[95] D. Liu, D. Li, M. Wang, and Z. Wang, "3D change detection using adaptive thresholds based on local point cloud density", *ISPRS International Journal of Geo-Information*, vol. 10, no. 3, 2021, ISSN: 2220-9964. DOI: 10.3390/ijgi10030127. [Online]. Available: https://www.mdpi.com/2220-9964/10/3/127.

[96] J. D. MacDonald and K. S. Booth, "Heuristics for ray tracing using space subdivision", *The Visual Computer*, vol. 6, no. 3, pp. 153–166, May 1990, ISSN: 1432-2315. DOI: 10.1007/BF01911006. [Online]. Available: https://doi.org/10.1007/BF01911006.

[97] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter, "Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration", *Journal of Software Engineering for Robotics*, vol. 3, no. 1, pp. 2–12, 2012.

[98] M. Lawson, W. Gropp, and J. Lofstead, "Exploring spatial indexing for accelerated feature retrieval in HPC", in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 605–614. DOI: 10.1109/CCGrid54584.2022.00070.

[99] N. Varney, V. K. Asari, and Q. Graehling, "DALES: A large-scale aerial LiDAR data set for semantic segmentation", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 186–187.

[100] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys, *Semantic3d.net: A new large-scale point cloud classification benchmark*, 2017. arXiv: 1704.03847 [cs.CV].

[101] I. Armeni *et al.*, "3D semantic parsing of large-scale indoor spaces", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[102] X. Roynard, J.-E. Deschaud, and F. Goulette, "Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification", *The International Journal of Robotics Research*, vol. 37, no. 6, pp. 545–557, 2018. DOI: 10.1177/0278364918767506.

[103] B. Brede *et al.*, "Non-destructive tree volume estimation through quantitative structure modelling: Comparing UAV laser scanning with terrestrial LIDAR", *Remote Sensing of Environment*, vol. 233, p. 111 355, 2019, ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2019.111355. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425719303748.

[104] I. Laguna, R. Marshall, K. Mohror, M. Ruefenacht, A. Skjellum, and N. Sultana, "A large-scale study of MPI usage in open-source HPC applications", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, Denver, Colorado: Association for Computing Machinery, 2019, ISBN: 9781450362290. DOI: 10.1145/3295500.3356176. [Online]. Available: https://doi.org/10.1145/3295500.3356176.

[105] V. Amaral *et al.*, "Programming languages for data-intensive HPC applications: A systematic mapping study", *Parallel Computing*, vol. 91, p. 102 584, 2020, ISSN: 0167-8191. DOI: https://doi.org/10.1016/j.parco.2019.102584. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819119301759.

[106] B. Schäling, *The boost C++ libraries*. XML press Laguna Hills, 2014, vol. 3.

[107] J. L. Blanco and P. K. Rai, *Nanoflann: A C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees*, https://github.com/jlblancoc/nanoflann, 2014.

[108] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.", *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[109] N. Varney, V. K. Asari, and Q. Graehling, "DALES: A large-scale aerial LiDAR data set for semantic segmentation", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 186–187.

[110] R. Van der Pas, E. Stotzer, and C. Terboven, *Using OpenMP—the next step: Affinity, Accelerators, Tasking, and SIMD*. MIT press, 2017, ISBN: 978-0-262-53478-9. DOI: 10.7551/mitpress/10031.001.0001.

[111] A. Parsakhoo and M. Jajouzadeh, "Determining an optimal path for forest road construction using Dijkstra's algorithm", *Journal of Forest Science*, vol. 62, no. 6, pp. 264–268, 2016. DOI: 10.17221/9/2016-JFS. [Online]. Available: https://doi.org/10.17221/9/2016-JFS.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[112] D. J. Cowen, J. R. Jensen, C. Hendrix, M. Hodgson, S. R. Schill, and F. Macchiaverna, "A GIS-assisted rail construction econometric model that incorporates LiDAR data", *Photogrammetric Engineering and Remote Sensing*, vol. 66, no. 11, pp. 1323–1328, 2000.

[113] S. Bagli, D. Geneletti, and F. Orsi, "Routeing of power lines through least-cost path analysis and multicriteria evaluation to minimise environmental impacts", *Environmental Impact Assessment Review*, vol. 31, no. 3, pp. 234–239, 2011, ISSN: 0195-9255. DOI: https://doi.org/10.1016/j.eiar.2010.10.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0195925510001393.

[114] Z. Zheng, T. R. Bewley, and F. Kuester, "Point cloud-based target-oriented 3D path planning for UAVs", in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 790–798. DOI: 10.1109/ICUAS48674.2020.9213894.

[115] Y. Zhang, X. Yuan, W. Li, and S. Chen, "Automatic power line inspection using UAV images", *Remote Sensing*, vol. 9, no. 8, 2017, ISSN: 2072-4292. DOI: 10.3390/rs9080824. [Online]. Available: https://www.mdpi.com/2072-4292/9/8/824.

[116] M. Metz, H. Mitasova, and R. S. Harmon, "Efficient extraction of drainage networks from massive, radar-based elevation models with least cost path search", *Hydrology and Earth System Sciences*, vol. 15, no. 2, pp. 667–678, 2011. DOI: 10.5194/hess-15-667-2011. [Online]. Available: https://hess.copernicus.org/articles/15/667/2011/.

[117] F. Sarı and M. Sen, "Least cost path algorithm design for highway route selection", *International Journal of Engineering and Geosciences*, vol. 2, no. 1, pp. 1–8, 2017. DOI: 10.26833/ijeg.285770.

[118] W. Rees, "Least-cost paths in mountainous terrain", *Computers & Geosciences*, vol. 30, no. 3, pp. 203–209, 2004, ISSN: 0098-3004. DOI: https://doi.org/10.1016/j.cageo.2003.11.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098300404000226.

[119] M. Contreras and W. Chung, "A computer approach to finding an optimal log landing location and analyzing influencing factors for ground-based timber harvesting", *Canadian Journal of Forest Research*, vol. 37, no. 2, pp. 276–292, 2007. DOI: 10.1139/x06-219. eprint: https://doi.org/10.1139/x06-219. [Online]. Available: https://doi.org/10.1139/x06-219.

[120] Y. Liu, M. Hatayama, and N. Okada, "Development of an adaptive evacuation route algorithm under flood disaster", *Annuals of Disaster Prevention Research Institute, Kyoto University*, vol. 49, pp. 189–195, 2006.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[121] M. O. Aguiar *et al.*, "Optimizing forest road planning in a sustainable forest management area in the Brazilian Amazon", *Journal of Environmental Management*, vol. 288, p. 112 332, 2021, ISSN: 0301-4797. DOI: https://doi.org/10.1016/j.jenvman.2021.112332. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301479721003947.

[122] P. Flisberg, M. Rönnqvist, E. Willén, M. Frisk, and G. Friberg, "Spatial optimization of ground-based primary extraction routes using the bestway decision support system", *Canadian Journal of Forest Research*, vol. 51, no. 5, pp. 675–691, 2021. DOI: 10.1139/cjfr-2020-0238. eprint: https://doi.org/10.1139/cjfr-2020-0238. [Online]. Available: https://doi.org/10.1139/cjfr-2020-0238.

[123] P. Axelsson, "Processing of laser scanner data—algorithms and applications", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 54, no. 2, pp. 138–147, 1999, ISSN: 0924-2716. DOI: https://doi.org/10.1016/S0924-2716(99)00008-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271699000088.

[124] A. H. Özcan and C. Ünsalan, "LiDAR data filtering and DTM generation using empirical mode decomposition", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 1, pp. 360–371, 2017. DOI: 10.1109/JSTARS.2016.2543464.

[125] P. F. Fisher and N. J. Tate, "Causes and consequences of error in digital elevation models", *Progress in Physical Geography: Earth and Environment*, vol. 30, no. 4, pp. 467–489, 2006. DOI: 10.1191/0309133306pp492ra. eprint: https://doi.org/10.1191/0309133306pp492ra. [Online]. Available: https://doi.org/10.1191/0309133306pp492ra.

[126] F. Aguilar, F. Agüera, A. M.A., and F. Carvajal, "Effects of terrain morphology, sampling density, and interpolation methods on grid DEM accuracy", *Photogrammetric Engineering and Remote Sensing*, vol. 71, pp. 805–816, Jul. 2005. DOI: 10.14358/PERS.71.7.805.

[127] Q. Zhou and X. Liu, "Analysis of errors of derived slope and aspect related to dem data properties", *Computers & Geosciences*, vol. 30, no. 4, pp. 369–378, 2004, Multidimensional geospatial technology for the geosciences, ISSN: 0098-3004. DOI: https://doi.org/10.1016/j.cageo.2003.07.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098300404000160.

[128] M. J. Campbell, P. E. Dennison, and B. W. Butler, "A LiDAR-based analysis of the effects of slope, vegetation density, and ground surface roughness on travel rates for wildland firefighter escape route mapping", *International journal of wildland fire*, vol. 26, no. 10, pp. 884–895, 2017. DOI: 10.1071/WF17031.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[129] J. Tang, P. Pilesjö, and A. Persson, "Estimating slope from raster data – a test of eight algorithms at different resolutions in flat and steep terrain", *Geodesy and Cartography*, vol. 39, pp. 41–52, Jun. 2013. DOI: 10.3846/20296991.2013.806702.

[130] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points", *SIGGRAPH Comput. Graph.*, vol. 26, no. 2, pp. 71–78, Jul. 1992, ISSN: 0097-8930. DOI: 10.1145/142920.134011. [Online]. Available: https://doi.org/10.1145/142920.134011.

[131] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, "Comparison of surface normal estimation methods for range sensing applications", in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 3206–3211. DOI: 10.1109/ROBOT.2009.5152493.

[132] W. W. Naismith, "Excursions. Cruach Ardran, Stobinian, and Ben More", *Scottish Mountaineering Club Journal*, p. 136, 1892.

[133] P. Scarf, "Route choice in mountain navigation, naismith's rule, and the equivalence of distance and climb", *J Sports Sci*, vol. 25, no. 6, pp. 719–726, Apr. 2007. DOI: 10.1080/02640410600874906.

[134] W. Tobler, *Three presentations on geographical analysis and modeling*. University of California, Santa Barbara: National Center for Geographical Analysis and Modeling, 1993.

[135] I. J. Irmischer and K. C. Clarke, "Measuring and modeling the speed of human navigation", *Cartography and Geographic Information Science*, vol. 45, no. 2, pp. 177–186, 2018. DOI: 10.1080/15230406.2017.1292150. eprint: https://doi.org/10.1080/15230406.2017.1292150. [Online]. Available: https://doi.org/10.1080/15230406.2017.1292150.

[136] R. C. Davey, M. Hayes, and J. M. Norman, "Running uphill: An experimental result and its applications", *The Journal of the Operational Research Society*, vol. 45, no. 1, pp. 25–29, 1994, ISSN: 01605682, 14769360. DOI: 10.2307/2583947. [Online]. Available: http://www.jstor.org/stable/2583947.

[137] M. J. Campbell, P. E. Dennison, B. W. Butler, and W. G. Page, "Using crowdsourced fitness tracker data to model the relationship between slope and travel rates", *Applied Geography*, vol. 106, pp. 93–107, 2019, ISSN: 0143-6228. DOI: https://doi.org/10.1016/j.apgeog.2019.03.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0143622818307859.

[138] J. Márquez Pérez, I. Vallejo Villalta, and J. I. Álvarez, "Estimación del tiempo de demora en rutas pedestres: Comparación de algoritmos", *GeoFocus. International Review of Geographical Information Science and Technology*, no. 15, pp. 47–74, Jun. 2015. [Online]. Available: https://www.geofocus.org/index.php/geofocus/article/view/408.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[139] O. G. Lorenzo, J. Martínez, D. L. Vilariño, T. F. Pena, J. C. Cabaleiro, and F. F. Rivera, "Parallel landing sites detection using LiDAR data on manycore systems", in *16th International Conference on Computational and Mathematical Methods in Science and Engineering*, 2016.

[140] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96, Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[141] G. Voronoi, "Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs.", *Journal für die reine und angewandte Mathematik (Crelles Journal)*, vol. 1908, no. 134, pp. 198–287, 1908. DOI: doi:10.1515/crll.1908.134.198. [Online]. Available: https://doi.org/10.1515/crll.1908.134.198.

[142] S. Fortune, "A sweepline algorithm for Voronoi diagrams", in *Proceedings of the Second Annual Symposium on Computational Geometry*, ser. SCG '86, Yorktown Heights, New York, USA: Association for Computing Machinery, 1986, pp. 313–322, ISBN: 0897911946. DOI: 10.1145/10515.10549. [Online]. Available: https://doi.org/10.1145/10515.10549.

[143] B. Delaunay *et al.*, "Sur la sphere vide", *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[144] A. Bowyer, "Computing Dirichlet tessellations", *The Computer Journal*, vol. 24, no. 2, pp. 162–166, Jan. 1981, ISSN: 0010-4620. DOI: 10.1093/comjnl/24.2.162. eprint: https://academic.oup.com/comjnl/article-pdf/24/2/162/967239/240162.pdf. [Online]. Available: https://doi.org/10.1093/comjnl/24.2.162.

[145] D. F. Watson, "Computing the N-dimensional Delaunay tessellation with application to Voronoi polytopes", *The Computer Journal*, vol. 24, no. 2, pp. 167–172, Jan. 1981, ISSN: 0010-4620. DOI: 10.1093/comjnl/24.2.167. eprint: https://academic.oup.com/comjnl/article-pdf/24/2/167/967258/240167.pdf. [Online]. Available: https://doi.org/10.1093/comjnl/24.2.167.

[146] S. Lloyd, "Least squares quantization in PCM", *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. DOI: 10.1109/TIT.1982.1056489.

[147] D. A. Field, "Implementing Watson's algorithm in three dimensions", in *Proceedings of the Second Annual Symposium on Computational Geometry*, ser. SCG '86, Yorktown Heights, New York, USA: Association for Computing Machinery, 1986, pp. 246–259, ISBN: 0897911946. DOI: 10.1145/10515.10542. [Online]. Available: https://doi.org/10.1145/10515.10542.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[148] V. Bilonenko, *Delaunator-cpp*, https://github.com/delfrrr/delaunator-cpp, 2018.

[149] V. Agafonkin, *Delaunator*, https://github.com/mapbox/delaunator, 2021.

[150] L. Yonghe, F. Jinming, and S. Yuehong, "A simple sweep-line Delaunay triangulation algorithm", *J. Algorithms Optimization*, vol. 1, no. 1, pp. 30–38, 2013.

[151] D. Sinclair, "S-hull: A fast radial sweep-hull routine for Delaunay triangulation", *CoRR*, vol. abs/1604.01428, 2016. arXiv: 1604.01428. [Online]. Available: http://arxiv.org/abs/1604.01428.

[152] A. Biniaz and G. Dastghaibyfard, "A faster circle-sweep Delaunay triangulation algorithm", *Advances in Engineering Software*, vol. 43, no. 1, pp. 1–13, 2012, ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2011.09.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965997811002420.

[153] N. F. Glenn, D. R. Streutker, D. J. Chadwick, G. D. Thackray, and S. J. Dorsch, "Analysis of LiDAR-derived topographic information for characterizing and differentiating landslide morphology and activity", *Geomorphology*, vol. 73, no. 1, pp. 131–148, 2006, ISSN: 0169-555X. DOI: https://doi.org/10.1016/j.geomorph.2005.07.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169555X05002047.

[154] U. Lussem, A. Bolten, M. L. Gnyp, J. Jasper, and G. Bareth, "Evaluation of RGB-based vegetation indices from UAV imagery to estimate forage yield in grassland", *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-3, pp. 1215–1219, 2018. DOI: 10.5194/isprs-archives-XLII-3-1215-2018. [Online]. Available: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-3/1215/2018/.

[155] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. DOI: 10.1109/TSSC.1968.300136.

[156] E. W. Dijkstra, "A note on two problems in connexion with graphs", *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959, ISSN: 0945-3245. DOI: 10.1007/BF01386390. [Online]. Available: https://doi.org/10.1007/BF01386390.

[157] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search", *Artificial Intelligence*, vol. 27, no. 1, pp. 97–109, 1985, ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(85)90084-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370285900840.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[158] C. Saranya, M. Unnikrishnan, S. A. Ali, D. Sheela, and D. V. Lalithambika, "Terrain based D* algorithm for path planning", *IFAC-PapersOnLine*, vol. 49, no. 1, pp. 178–182, 2016, 4th IFAC Conference on Advances in Control and Optimization of Dynamical Systems ACODS 2016, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2016.03.049. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896316300490.

[159] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: an anytime, replanning algorithm", in *Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'05, Monterey, California, USA: AAAI Press, 2005, pp. 262–271, ISBN: 1577352203.

[160] R. Bellman, "On a routing problem", *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958, ISSN: 0033569X, 15524485. [Online]. Available: http://www.jstor.org/stable/43634538.

[161] L. R. Ford Jr, "Network flow theory", Rand Corp Santa Monica Ca, Tech. Rep., 1956.

[162] R. W. Floyd, "Algorithm 97: Shortest path", *Commun. ACM*, vol. 5, no. 6, pp. 345–, Jun. 1962, ISSN: 0001-0782. DOI: 10.1145/367766.368168. [Online]. Available: http://doi.acm.org/10.1145/367766.368168.

[163] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms", *J. ACM*, vol. 34, no. 3, pp. 596–615, Jul. 1987, ISSN: 0004-5411. DOI: 10.1145/28869.28874. [Online]. Available: https://doi.org/10.1145/28869.28874.

[164] Babcock, *Babcock international. trusted to deliver*, Jul. 2020. [Online]. Available: https://www.babcockinternational.com/.

[165] J. Martínez, F. F. Rivera, J. C. Cabaleiro, D. L. Vilariño, T. F. Pena, and D. M. B., "A rule-based classification from a region-growing segmentation of airborne LiDAR", in *Image and Signal Processing for Remote Sensing XXII*, L. Bruzzone and F. Bovolo, Eds., International Society for Optics and Photonics, vol. 10004, SPIE, 2016, pp. 140–150. DOI: 10.1117/12.2240750. [Online]. Available: https://doi.org/10.1117/12.2240750.

[166] J. Martínez, F. F. Rivera, J. C. Cabaleiro, D. L. Vilariño, and T. F. Pena, "Automatic extraction of road points from airborne LiDAR based on bidirectional skewness balancing", *Remote Sensing*, vol. 12, no. 12, p. 21, 2020. DOI: 10.3390/rs12122025. [Online]. Available: http://dx.doi.org/10.3390/rs12122025.

[167] OpenMP Architecture Review Board, *OpenMP application program interface version 4.5*, Nov. 2015. [Online]. Available: https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[168] A. Grama, V. Kumar, A. Gupta, and G. Karypis, *Introduction to parallel computing*, 2nd ed. Pearson Education, 2003, ch. 5, pp. 212–217.

[169] S. Ristov, R. Prodan, M. Gusev, and K. Skala, "Superlinear speedup in HPC systems: Why and when?", in *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, IEEE, 2016, pp. 889–898.

[170] T. J. Pingel, K. C. Clarke, and W. A. McBride, "An improved simple morphological filter for the terrain classification of airborne LiDAR data", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 77, pp. 21–30, 2013, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2012.12.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924271613000026.

[171] R. Carande, M. Marra, D. Cronin, and P. Nagy, "Automated mapping using airborne IFSAR data", in *IGARSS '98. Sensing and Managing the Environment. 1998 IEEE International Geoscience and Remote Sensing. Symposium Proceedings. (Cat. No.98CH36174)*, vol. 1, 1998, 360–362 vol.1. DOI: 10.1109/IGARSS.1998.702905.

[172] J. Ahmad, A. S. Malik, and L. Xia, "Vegetation monitoring for high-voltage transmission line corridors using satellite stereo images", in *2011 National Postgraduate Conference*, 2011, pp. 1–5. DOI: 10.1109/NatPC.2011.6136337.

[173] Y. Kobayashi, G. G. Karady, G. T. Heydt, and R. G. Olsen, "The utilization of satellite images to identify trees endangering transmission lines", *IEEE Transactions on Power Delivery*, vol. 24, no. 3, pp. 1703–1709, 2009. DOI: 10.1109/TPWRD.2009.2022664.

[174] H. Zhang, W. Yang, H. Yu, H. Zhang, and G.-S. Xia, "Detecting power lines in UAV images with convolutional features and structured constraints", *Remote Sensing*, vol. 11, no. 11, 2019, ISSN: 2072-4292. DOI: 10.3390/rs11111342. [Online]. Available: https://www.mdpi.com/2072-4292/11/11/1342.

[175] G. Yan *et al.*, "An airborne multi-angle power line inspection system", in *2007 IEEE International Geoscience and Remote Sensing Symposium*, 2007, pp. 2913–2915. DOI: 10.1109/IGARSS.2007.4423453.

[176] W. Zhang, G. Yan, N. Wang, Q. Li, and W. Zhao, "Automatic 3D power line reconstruction of multi-angular imaging power line inspection system", in *Geoinformatics 2007: Remotely Sensed Data and Information*, W. Ju and S. Zhao, Eds., International Society for Optics and Photonics, vol. 6752, SPIE, 2007, pp. 597–605. DOI: 10.1117/12.760699. [Online]. Available: https://doi.org/10.1117/12.760699.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[177] M. Frank, Z. Pan, B. Raber, and C. Lenart, "Vegetation management of utility corridors using high-resolution hyperspectral imaging and LiDAR", in *2010 2nd Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing*, 2010, pp. 1–4. DOI: 10.1109/WHISPERS.2010.5594887.

[178] A. B. Alhassan, X. Zhang, H. Shen, and H. Xu, "Power transmission line inspection robots: A review, trends and challenges for future research", *International Journal of Electrical Power & Energy Systems*, vol. 118, p. 105 862, 2020, ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2020.105862. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061519324524.

[179] G. E. Teng *et al.*, "Mini-UAV LiDAR for power line inspection", *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W7, pp. 297–300, 2017. DOI: 10.5194/isprs-archives-XLII-2-W7-297-2017. [Online]. Available: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-2-W7/297/2017/.

[180] C. Nardinocchi, M. Balsi, and S. Esposito, "Fully automatic point cloud analysis for powerline corridor mapping", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 12, pp. 8637–8648, 2020. DOI: 10.1109/TGRS.2020.2989470.

[181] M. Dihkan and E. Mus, "Automatic detection of power transmission lines and risky object locations using UAV LiDAR data", *Arabian Journal of Geosciences*, vol. 14, no. 7, pp. 1–13, 2021. DOI: 10.1007/s12517-021-06947-1.

[182] Y. Huang, Y. Du, and W. Shi, "Fast and accurate power line corridor survey using spatial line clustering of point cloud", *Remote Sensing*, vol. 13, no. 8, 2021, ISSN: 2072-4292. DOI: 10.3390/rs13081571. [Online]. Available: https://www.mdpi.com/2072-4292/13/8/1571.

[183] S. Ortega, A. Trujillo, J. M. Santana, J. P. Suárez, and J. Santana, "Characterization and modeling of power line corridor elements from LiDAR point clouds", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 152, pp. 24–33, 2019, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2019.03.021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271619300905.

[184] S. Ortega, A. Trujillo-Pino, J. M. S. Núñez, and J. P. S. Rivero, "An image-based method to classify power lines in LiDAR point-clouds", in *12th International Tools and Methods of Competitive Engineering Symposium (TMCE 2018)*, Proceedings of TMCE 2018, May 2018.

[185] C. Chen, B. Yang, S. Song, X. Peng, and R. Huang, "Automatic clearance anomaly detection for transmission line corridors utilizing UAV-borne LiDAR data", *Remote Sensing*, vol. 10, no. 4, 2018, ISSN: 2072-4292. DOI: 10.3390/rs10040613. [Online]. Available: https://www.mdpi.com/2072-4292/10/4/613.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[186] H. Guan, Y. Yu, J. Li, Z. Ji, and Q. Zhang, "Extraction of power-transmission lines from vehicle-borne LiDAR data", *International Journal of Remote Sensing*, vol. 37, no. 1, pp. 229–247, 2016. DOI: 10.1080/01431161.2015.1125549. eprint: https://doi.org/10.1080/01431161.2015.1125549. [Online]. Available: https://doi.org/10.1080/01431161.2015.1125549.

[187] G. W. Leibniz, "De linea in quam flexile se pondere proprio curvat, eiusque usu insigni ad inveniendas quotcumque medias proportionales et logarithmos", *Acta Eruditorum*, vol. 10, pp. 277–281, Jun. 1691.

[188] G. W. Leibniz, "De solutionibus problematis catenarii vel funicularis in actis junii a. 1691, aliisque a dn. ib propositis", *Acta Eruditorum*, vol. 10, no. 1691, pp. 434–439, Sep. 1691.

[189] J. Canny, "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986, ISSN: 1939-3539. DOI: 10.1109/TPAMI.1986.4767851.

[190] P. V. Hough, "Method and means for recognizing complex patterns", US 3069654, Dec. 1962. [Online]. Available: https://www.osti.gov/biblio/4746348.

[191] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures", *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972, ISSN: 0001-0782. DOI: 10.1145/361237.361242. [Online]. Available: https://doi.org/10.1145/361237.361242.

[192] E. Davies, "Finding ellipses using the generalised Hough transform", *Pattern Recognition Letters*, vol. 9, no. 2, pp. 87–96, 1989, ISSN: 0167-8655. DOI: https://doi.org/10.1016/0167-8655(89)90041-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/016786558990041X.

[193] D. Ballard, "Generalizing the Hough transform to detect arbitrary shapes", *Pattern Recognition*, vol. 13, no. 2, pp. 111–122, 1981, ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(81)90009-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0031320381900091.

[194] C. Dalitz, T. Schramke, and M. Jeltsch, "Iterative Hough transform for line detection in 3D point clouds", *Image Processing On Line*, vol. 7, pp. 184–196, 2017, https://doi.org/10.5201/ipol.2017.208.

[195] D. Borrmann, J. Elseberg, K. Lingemann, and A. Nüchter, "The 3D Hough transform for plane detection in point clouds: A review and a new accumulator design", *3D Research*, vol. 2, no. 2, p. 3, 2011, ISSN: 2092-6731. DOI: 10.1007/3DRes.02(2011)3. [Online]. Available: https://doi.org/10.1007/3DRes.02(2011)3.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[196] M. Camurri, R. Vezzani, and R. Cucchiara, "3D Hough transform for sphere recognition on point clouds", *Machine Vision and Applications*, vol. 25, no. 7, pp. 1877–1891, 2014, ISSN: 1432-1769. DOI: 10.1007/s00138-014-0640-3. [Online]. Available: https://doi.org/10.1007/s00138-014-0640-3.

[197] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection", *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640.

[198] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", *CoRR*, vol. abs/1311.2524, 2013. arXiv: 1311.2524. [Online]. Available: http://arxiv.org/abs/1311.2524.

[199] P. Viola and M. J. Jones, "Robust real-time face detection", *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004, ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000013087.49260.fb. [Online]. Available: https://doi.org/10.1023/B:VISI.0000013087.49260.fb.

[200] J. H. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression", *CoRR*, vol. abs/1705.02950, 2017. arXiv: 1705.02950. [Online]. Available: http://arxiv.org/abs/1705.02950.

[201] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Improving object detection with one line of code", *CoRR*, vol. abs/1704.04503, 2017. arXiv: 1704.04503. [Online]. Available: http://arxiv.org/abs/1704.04503.

[202] S. Liu, D. Huang, and Y. Wang, "Adaptive NMS: refining pedestrian detection in a crowd", *CoRR*, vol. abs/1904.03629, 2019. arXiv: 1904.03629. [Online]. Available: http://arxiv.org/abs/1904.03629.

[203] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-IoU loss: Faster and better learning for bounding box regression", *CoRR*, vol. abs/1911.08287, 2019. arXiv: 1911.08287. [Online]. Available: http://arxiv.org/abs/1911.08287.

[204] T. Melzer and C. Briese, "Extraction and modeling of power lines from ALS point clouds", in *Proceedings of 28th Workshop of the Austrian Association for Pattern Recognition (ÖAGM)*, Hagenberg, Austria, Jun. 2004, pp. 47–54.

[205] Y. Liu, Z. Li, R. Hayward, R. Walker, and H. Jin, "Classification of airborne LiDAR intensity data using statistical analysis and Hough transform with application to power line corridors", in *2009 Digital Image Computing: Techniques and Applications*, 2009, pp. 462–467. DOI: 10.1109/DICTA.2009.83.

[206] PG&E Diablo Canyon Power Plant (DCPP), *Diablo Canyon, CA Central Coast.* [Online; accessed 18-Oct-2021], 2022. DOI: 10.5069/G9P26W19. [Online]. Available: https://portal.opentopography.org/datasets.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[207] K. Levenberg, "A method for the solution of certain non-linear problems in least squares", *Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944. DOI: https://doi.org/10.1090/qam/10666.

[208] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters", *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963. DOI: 10.1137/0111030. [Online]. Available: https://doi.org/10.1137/0111030.

[209] R. Penrose, "A generalized inverse for matrices", *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 51, no. 3, pp. 406–413, 1955. DOI: 10.1017/S0305004100030401.

[210] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography", *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692.

[211] S. Agarwal, K. Mierle, and T. C. S. Team, *Ceres Solver*, version 2.2, Oct. 2023. [Online]. Available: https://github.com/ceres-solver/ceres-solver.

[212] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory", in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43. DOI: 10.1109/MHS.1995.494215.

[213] I. Newton, *The Method of Fluxions and Infinite Series; with its Application to the Geometry of Curve-lines*. Henry Woodfall and John Nourse, 1736.

[214] C. F. Gauss, *Theoria motus corporum coelestium in sectionibus conicis solem ambientium auctore Carolo Friderico Gauss*. sumtibus Frid. Perthes et IH Besser, 1809.

[215] A. Cauchy *et al.*, "Méthode générale pour la résolution des systemes d'équations simultanées", *Comp. Rend. Sci. Paris*, vol. 25, no. 1847, pp. 536–538, 1847.

[216] R. P. Brent, *Algorithms for minimization without derivatives*. Prentice Hall, 1972, ch. 4.

[217] J. Raphson, *Analysis Aequationum Universalis Seu Ad Aequationes Algebraicas Resolvendas Methodus Generalis, & Expedita, Ex Nova Infinitarum Serierum Methodo, Deducta Ac Demonstrata*. Prostant venales apud Abelem Swalle, 1690.

[218] X. H. Chen, J. Q. Dai, Y. R. He, and W. W. Ma, "Power line extraction and analysis based on LiDAR", *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-3/W10, pp. 91–96, 2020. DOI: 10.5194/isprs-archives-XLII-3-W10-91-2020.

[219] *CloudCompare (version 2.13.1) [gpl software]*, 2024. [Online]. Available: http://www.cloudcompare.org/.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[220] D. M. W. Powers, *Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation*, 2020. arXiv: 2010.16061 [cs.LG].

[221] J. Roussel, A. Achim, and D. Auty, "Classification of high-voltage power line structures in low density ALS data acquired over broad non-urban areas", *PeerJ Computer Science*, vol. 7, e672, 2021. DOI: 10.7717/peerj-cs.672.

[222] D. Shokri, H. Rastiveis, W. A. Sarasua, A. Shams, and S. Homayouni, "A robust and efficient method for power lines extraction from mobile LiDAR point clouds", *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, vol. 89, no. 3, pp. 209–232, Jun. 2021, ISSN: 2512-2819. DOI: 10.1007/s41064-021-00155-y. [Online]. Available: https://doi.org/10.1007/s41064-021-00155-y.

[223] S. Wang, H. Wu, H. Yue, L. Yao, C. Liu, and H. Sun, "Automated extraction of tunnel electricity transmission system: An object-level approach with mobile laser scanning data", *International Journal of Applied Earth Observation and Geoinformation*, vol. 116, p. 103 136, 2023, ISSN: 1569-8432. DOI: https://doi.org/10.1016/j.jag.2022.103136. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569843222003247.

[224] R. McLaughlin, "Extracting transmission lines from airborne LIDAR data", *IEEE Geoscience and Remote Sensing Letters*, vol. 3, no. 2, pp. 222–226, 2006. DOI: 10.1109/LGRS.2005.863390.

[225] L. Zhu and J. Hyyppä, "Fully-automated power line extraction from airborne laser scanning point clouds in forest areas", *Remote Sensing*, vol. 6, no. 11, pp. 11 267–11 282, 2014, ISSN: 2072-4292. DOI: 10.3390/rs61111267. [Online]. Available: https://www.mdpi.com/2072-4292/6/11/11267.

[226] B. Guo, X. Huang, F. Zhang, and G. Sohn, "Classification of airborne laser scanning data using JointBoost", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 100, pp. 71–83, 2015, High-Resolution Earth Imaging for Geospatial Information, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2014.04.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271614001105.

[227] B. Guo, Q. Li, X. Huang, and C. Wang, "An improved method for power-line reconstruction from point cloud data", *Remote Sensing*, vol. 8, no. 1, 2016, ISSN: 2072-4292. DOI: 10.3390/rs8010036. [Online]. Available: https://www.mdpi.com/2072-4292/8/1/36.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# List of Figures


Fig. 1.1 Aerial photograph of MRBM Field Launch Site No.1 in San Cristobal, Cuba, on 14/10/1962 [1]. Public domain. . . . . 10
Fig. 1.2 First image of the Earth taken from outside the Earth itself [2]. Public domain. . . . . 11

Fig. 2.1 Computation of the TOF using the discrete pulse method. . . . . 20
Fig. 2.2 Measure of the phase difference between transmitted and received waves. 22
Fig. 2.3 Beam frequency variation with the time. In orange is the frequency of the measuring signal. In blue, the extrapolated frequency of a returning wave after $\tau$ seconds of traveling. . . . . 25
Fig. 2.4 Extraction of the beat frequency using the measuring and the returning carrier signal by means of the Fourier transform in the generated interference. . . . . 26
Fig. 2.5 Depiction of the inverse-squared law [42]. Three snapshots of a wavefront are shown. CC BY-SA 3.0. . . . . 28
Fig. 2.6 Intensity distribution through a circular aperture. Left: Airy disk diffraction pattern. Right: Intensity distribution along an arbitrary axis. . . . . 29
Fig. 2.7 Mirrors commonly used to deflect the incoming laser beam and the corresponding scanning patterns produced [47]. CC BY 3.0. . . . . 31
Fig. 2.8 Scan pattern produced by the Livox Horizon [48] at 0.1 s, 0.2 s, and 0.5 s into the scanning process. This scanner can produce up to 480 000 points per second. . . . . 31

Fig. 3.1 2D kernels. The left image shows the square kernel and the right image shows the circular kernel. . . . . 45
Fig. 3.2 3D kernels. The left image shows the cube kernel and the right image shows the spherical kernel. . . . . 46
Fig. 3.3 Torus and ring kernels. . . . . 47
Fig. 3.4 Example of a $k$-NN neighborhood for $k = 10$. . . . . 48


USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Fig. 3.5 Comparison of query times using a specialized library versus the brute force approach. . . . 49
Fig. 3.6 Example of a $k$-d tree partitioning a 2D dataset. The data is recursively divided between nodes that are split using a hyperplane. The hyperplanes are orthogonal to each other. . . . 50
Fig. 3.7 Example of an Octree partitioning a 2D dataset. . . . 52
Fig. 3.8 Query times for the DALES dataset using different libraries and search methods. . . . 56
Fig. 3.9 Query times for the *sg28_2* dataset using different libraries and search methods. . . . 56
Fig. 3.10 Query times for the *sg28_4* dataset using different libraries and search methods. . . . 57
Fig. 3.11 Query times for the hallway_6 dataset using different libraries and search methods. . . . 57
Fig. 3.12 Query times for the Paris dataset using different libraries and search methods. . . . 58
Fig. 3.13 Query times for the Lille dataset using different libraries and search methods. . . . 58
Fig. 3.14 Query times for the Speulderbos dataset using different libraries and search methods. . . . 59
Fig. 3.15 Speedup for different kernels using three OpenMP scheduling strategies. 61
Fig. 3.16 Isoefficiency matrix for the circle kernel in Machine 2. . . . 61
Fig. 3.17 Isoefficiency matrix for the circle kernel in Machine 3. . . . 62

Fig. 4.1 Neighboring cells in a DTM grid. The red circle is the node whose neighbors are being computed. The green cells are the eight neighbors of the red node. . . . 67
Fig. 4.2 Flowchart of the pathfinding algorithm. . . . 69
Fig. 4.3 Voronoi and Delaunay tessellations on a random set of nodes. The black circles are the nodes of the graph. The red lines are the Voronoi tessellation edges, and the dashed back lines are the corresponding dual graph: the Delaunay triangulation. . . . 70
Fig. 4.4 Determination of the points lying inside a given edge. The blue circle is the graph node whose edges are being built. The neighbor used to build the edge is shown in light blue. The nodes in green are graph nodes that will be processed later. The gray circles are points found in the initial search, and the yellow ones are those finally included in the edge. . . . 71
Fig. 4.5 Orthomosaics of the areas used in the different tests. . . . 82
Fig. 4.6 Segmentations generated with different combination of the parameters $R_s$ and $N_s$. The scale shown in the bottom right corner of each map is given in meters. . . . 83



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Fig. 4.7 Voronoi tessellation of a point cloud. . . . . 84
Fig. 4.8 Resulting graph using the neighboring information of the Voronoi Tessellation. . . . . 85
Fig. 4.9 Surface roughness. . . . . 86
Fig. 4.10 Example of paths computed with and without considering the roughness. The path with the white color corresponds to that computed without including the roughness in the weight computation, and the black path is the one including it. . . . . 87
Fig. 4.11 Detail of tire tracks in the ground. . . . . 88
Fig. 4.12 Path found using the weights given by the NDVI values. In white, the path found without using NDVI values is shown. In black, the same path is found using the NDVI values of each edge. . . . . 89
Fig. 4.13 Execution times in the cases when computing the roughness at edge level and at point level. . . . . 91
Fig. 4.14 Speedup and efficiency achieved on Test A. Problem size: 14 000 000 points. . . . . 92
Fig. 4.15 Achieved speedup as a function of the problem size, given in number of points, for Test A. . . . . 92
Fig. 4.16 Efficiency matrix for Test A. . . . . 93
Fig. 4.17 Speedup for Test B. Problem size: 40 000 000 of points. . . . . 94
Fig. 4.18 Efficiency as a function of the number of threads and the problem size for Test B. . . . . 95
Fig. 4.19 Trafficability graph over different areas. . . . . 96
Fig. 4.20 Trafficability graph over a non-classified building. . . . . 97

Fig. 5.1 Powerline detection algorithm flowchart. . . . . 103
Fig. 5.2 TPF flowchart. . . . . 104
Fig. 5.3 Application of the TPF algorithm to a point belonging to a single powerline and a multilevel powerline. . . . . 105
Fig. 5.4 Example of two points sharing a line in the Euclidean space. . . . . 106
Fig. 5.5 Continuous version of the Hough transform. . . . . 107
Fig. 5.6 Discrete version of the Hough transform. In this case $A_s = 0.2$ and $G_s = 0.2$. The Hough space is colored according to the number of votes in each bin. . . . . 108
Fig. 5.7 Flowchart of the Hough transform algorithm for 3D point clouds. . . 109
Fig. 5.8 Multiple detections of the same line in the Euclidean space due to the discretization of the Hough space. . . . . 109
Fig. 5.9 GreedyNMS algorithm. . . . . 113
Fig. 5.10 VES: the heuristicless NMS algorithm. . . . . 114
Fig. 5.11 Hough transform curve deletion example. . . . . 114
Fig. 5.12 Detail of Hough spaces with and without repeated points and their difference. . . . . 115



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Fig. 5.13 Powerline to be used as an example in the clustering algorithm. . . . 116
Fig. 5.14 Normalization of the powerline used as an example in the clustering algorithm. . . . 117
Fig. 5.15 Sampling intervals for the RANSAC algorithm. The lowest points encountered in each interval are in red. . . . 118
Fig. 5.16 Clusterization of the points using the curve fit with the lowest points in the dataset. . . . 119
Fig. 5.17 Clusterization of multiple, non-parallel, uneven catenaries. . . . 120
Fig. 5.18 Diagram of a catenary curve. . . . 121
Fig. 5.19 Graph of $f(\lambda)$ for several k values. . . . 124
Fig. 5.20 Diablo Canyon dataset. . . . 127
Fig. 5.21 Results of the Hough transform applied on the Diablo Canyon dataset. 128
Fig. 5.22 Results of the TPF applied to the Diablo Canyon dataset. . . . 129
Fig. 5.23 Lugo dataset. . . . 130
Fig. 5.24 Different stages of the powerline detection algorithm applied to the Lugo dataset. . . . 131
Fig. 5.25 DALES dataset. . . . 132
Fig. 5.26 Results on Dales dataset. . . . 133
Fig. 5.27 Detailed view of the Diablo Canyon dataset. . . . 135
Fig. 5.28 Detail of false negatives in the Lugo dataset. . . . 137
Fig. 5.29 Detail of mislabeled conductors as pylons in the DALES dataset. . . . 137
Fig. 5.30 Sensitivity analysis of the algorithm parameters. The study was carried out on the Diablo Canyon dataset. . . . 140
Fig. 5.31 Influence of the point density on the algorithm. The study was carried out on the Diablo Canyon dataset. . . . 141
Fig. 5.32 Execution times for Diablo Canyon dataset. . . . 142
Fig. 5.33 Speedup and efficiency for Diablo Canyon dataset. . . . 142
Fig. 5.34 Isoefficiency matrix for the TPF algorithm. . . . 144
Fig. 5.35 Isoefficiency matrix for the Hough transform algorithm. . . . 145
Fig. 5.36 Isoefficiency matrix for the voting procedure. . . . 146
Fig. 5.37 Isoefficiency matrix for the VES procedure. . . . 147
Fig. 5.38 Isoefficiency matrix for the procedure of finding maximum values in the accumulator. . . . 148

Fig. A.1 FT2 machine architecture. . . . 194
Fig. A.2 FT3 machine architecture. . . . 195
Fig. A.3 CTNUMA machine architecture. . . . 196
Fig. A.4 Desktop machine architecture. . . . 196

Fig. B.1 Isoefficiency matrix for the square kernel in Machine 3. . . . 197
Fig. B.2 Isoefficiency matrix for the square kernel in Machine 2. . . . 198
Fig. B.3 Isoefficiency matrix for the cube kernel in Machine 3. . . . 198



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Fig. B.4 Isoefficiency matrix for the cube kernel in Machine 2. . . . . 199
Fig. B.5 Isoefficiency matrix for the sphere kernel in Machine 3. . . . . 199
Fig. B.6 Isoefficiency matrix for the sphere kernel in Machine 2. . . . . 200



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# List of Tables


Tab. 2.1 Example of using several discrete frequencies to determine a range uniquely. In bold, the significant digits that are added to the final distance measurement. . . . . 23
Tab. 2.2 LAS format definition. . . . . 38
Tab. 2.3 Variable Length Record Header . . . . . 38
Tab. 2.4 Point Data Record Format 8 . . . . . 39
Tab. 2.5 ASPRS Standard Point Classes . . . . . 41

Tab. 3.1 Characteristics of the datasets used in the comparison. The sensor information used in the Semantic3D dataset is not publicly available. 54

Tab. 4.1 Characteristics for state-of-the-art path-planning on 3D point clouds. 65
Tab. 4.2 Longitudinal and Transversal slope factors[119]. . . . . 76
Tab. 4.3 Roughness factors. . . . . 77
Tab. 4.4 Comparison of the different algorithms to compute the optimal path. 78

Tab. 5.1 Comparison of the error in the catenary parameters numerically computed when using different polynomials and the actual catenary equation. . . . . 125
Tab. 5.2 Summary of the datasets used for validation. . . . . 126
Tab. 5.3 Performance Metrics . . . . . 135
Tab. 5.4 Comparison of several powerline classification methods. . . . . 139
Tab. 5.5 Parameters range for the sensitivity analysis. . . . . 140


USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# List of Algorithms


3.1 K-Dimensional Tree Construction . . . . . 51
3.2 Compute Octant Idx . . . . . 51
3.3 Octree construction function . . . . . 53
3.4 Create child octants . . . . . 53
3.5 Insert point in the given octant . . . . . 54
4.1 Dijkstra algorithm . . . . . 79
5.1 Hough Space construction and voting procedure . . . . . 110
5.2 Voting Elements Suppresion algorithm . . . . . 112


USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# Acronyms

**ALS** Airborne Laser Scanning 33, 34, 44, 45, 90, 125, 152

**AM** Amplitude Modulation 21, 22, 24

**ASPRS** American Society for Photogrammetry and Remote Sensing 37

**CMRU** Control, Monitoring, and Recording Unit 34

**CSR** Compressed Sparse Row 80

**CW** Continuous Wave 21, 22, 24, 26

**D*** Dynamic A* 78

**DBSCAN** Density-Based Spatial Clustering of Applications with Noise 68

**DEM** Digital Elevation Model 94

**DGPS** Differential GPS 34

**DTM** Digital Terrain Model 18, 64–69, 71, 94, 95, 97, 103, 144, 180

**EVLR** Extended Variable Length Records 37, 38

**FB** FastBinary 37

**FLANN** Fast Library for Approximate Nearest Neighbors 55

**FM** Frequency Modulation 21, 24

**FOV** Field Of View 32

**FSI** Frequency-Sweeping Interferometry 24

**GIS** Geographical Information Systems 66

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**GPS** Global Positioning System 18, 19, 34–36, 40, 43

**HPC** High-Performance Computing 55, 58, 59, 151

**ICESat** Ice, Cloud, and land Elevation Satellite 12

**ICESat-2** Ice, Cloud, and land Elevation Satellite 2 34

**IDA*** Iterative-Deepening A* 77, 78

**IMU** Inertial Measurement Unit 18, 34–36

**KNN** $K$-Nearest Neighbors 44, 46, 50, 55, 56, 58

**LAS** LASer 37, 38, 55

**laser** Light Amplification by Stimulated Emission of Radiation 17–21, 26, 27, 29–31, 33–35, 38

**LCP** Least Cost Path 63–68, 71, 78, 80

**LiDAR** Light Detection And Ranging 12–15, 18, 19, 26, 30–37, 50, 53, 55, 58, 64, 71, 74, 80, 100–102, 115, 122, 129, 145–147, 151, 152

**LRU** Laser Ranging Unit 34

**MAE** Mean Absolute Error 134

**maser** Microwave Amplification by Stimulated Emission of Radiation 17

**MBES** Multi-Beam Echo Sounding 13

**MEMS** Microelectromechanical Systems 32

**MLS** Mobile Laser Scanning 33, 44, 55

**NDVI** Normalized Difference Vegetation Index 75, 76, 88, 89, 97, 181

**NIR** Near-Infrared 67, 75, 76, 152

**NMS** Non-Maximum Suppression 102, 109, 111, 114, 147

**NUMA** Non-Uniform Memory Access 60, 62, 143, 193

**OBJ** Wavefront OBJ 36



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

**PCA** Principal Component Analysis 44

**PCD** Point Cloud Data 37

**PCL** Point Cloud Library 37, 55, 57, 58, 62

**PIC** Photonics Integrated Circuit 32

**PlaneSVD** Plane Singular Value Decomposition 72, 73

**PLY** Polygon File Format 36, 55

**PNOA** National Plan of Aerial Orthophotography 81

**POS** Positioning and Orientation System 34, 35

**PRF** Pulse Repetition Frequency 20, 21

**radar** Radio detection and ranging 12

**RANSAC** RAndom SAmple Consensus 118, 119

**RMSE** Root Mean Squared Error 134

**SAR** Synthetic Aperture Radar 13, 99

**SLS** Spaceborn Laser Scanning 34

**SRTM** NASA's Shuttle Radar Topography Mission 12

**SVD** Singular Value Decomposition 44, 73

**TLS** Terrestrial Laser Scanning 33, 44

**TOF** Time of Flight 19, 21, 24, 26

**TPF** Transmission line Point height-based Filter 102, 103, 105, 126, 139, 141–144, 181

**UAV** Unmanned Aerial Vehicle 33, 100

**ULS** Unmanned Laser Scanning 33, 55

**VES** Voting Elements Suppression 113, 143

**VLR** Variable Length Records 37, 38

**WGS84** World Geodetic System 1984 35, 36

**XYZ** Plain coordinates file 36



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

## Appendix A

# Machines

1. Finisterrae II (FT2): 2x Intel(R) Xeon(R) E5-2680 v3 CPU @ 2.50 GHz, with 12 physical cores each in two sockets. Sandy Bridge architecture. 251 GiB of RAM. GNU/Linux 4.18.0. The compiler is GCC 10.1.0. The architecture is shown in Figure A.1.

2. Finisterrae III (FT3): 2x Intel(R) Xeon(R) Platinum 8352Y CPU @ 2.20 GHz, with 32 physical cores each in two sockets and four NUMA nodes. Ice Lake architecture. 251 GiB of RAM. The operative system is GNU/Linux Kernel 4.18.0. The compiler is GCC 10.1.0. The architecture is shown in Figure A.2.

3. CTNUMA: 4x Intel(R) Xeon(R) CPU E5-4620 v4 @ 2.10 GHz, with 10 physical cores each in four sockets and four NUMA nodes. Broadwell architecture. 251 GiB of RAM. The operative system is GNU/Linux Kernel 4.18.0. The compiler is GCC 12.1.0. The architecture is shown in Figure A.3.

4. Desktop: Intel(R) Core(TM) i7-9700K CPU @ 3.60 GHz, with 8 physical cores in one NUMA socket. Coffee Lake architecture. 32 GiB of RAM. The operative system is GNU/Linux Kernel 5.15.0. The compiler is GCC 9.3.0. The architecture is shown in Figure A.4.

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

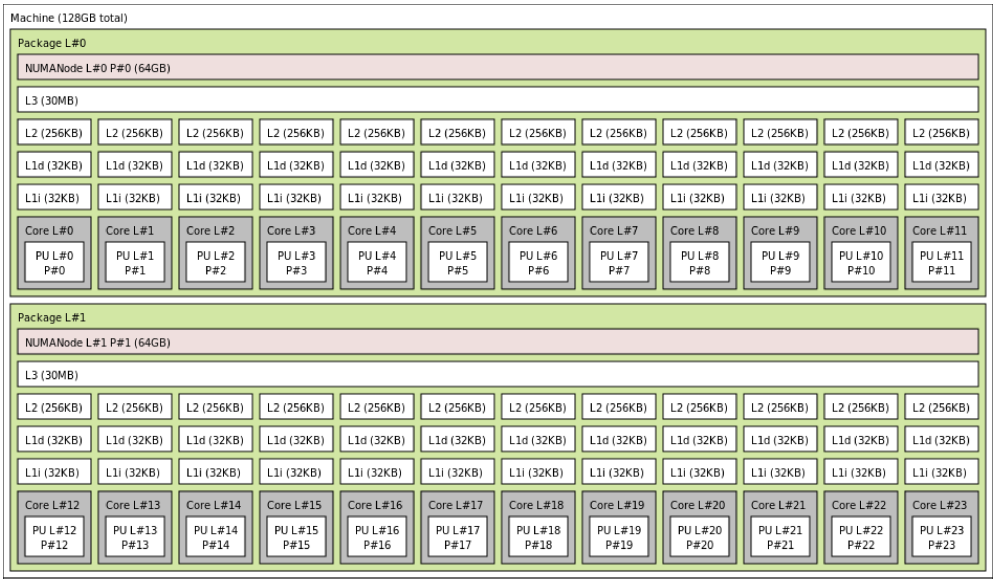



**Figure A.1:** FT2 machine architecture.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

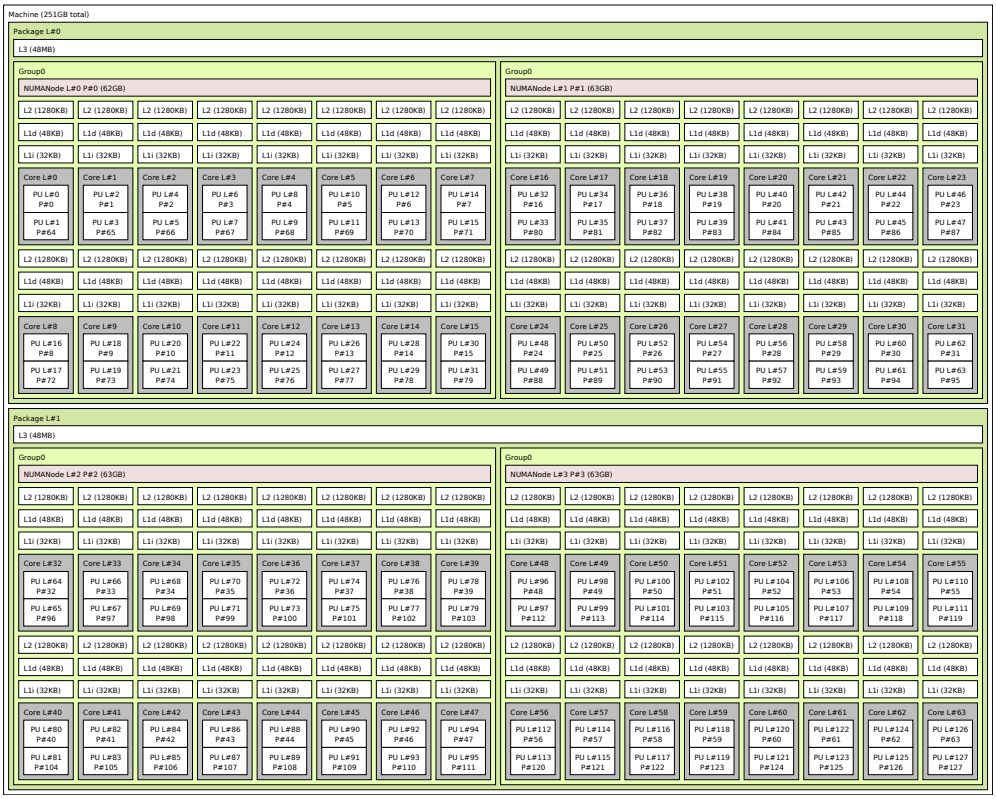



**Figure A.2:** FT3 machine architecture.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

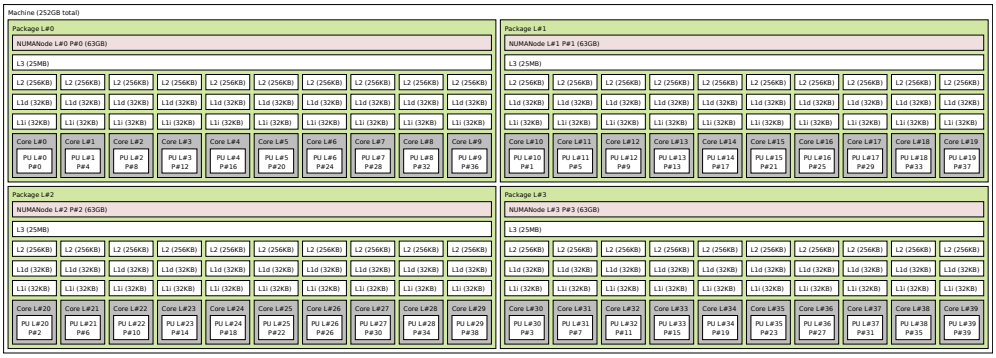



**Figure A.3:** CTNUMA machine architecture.

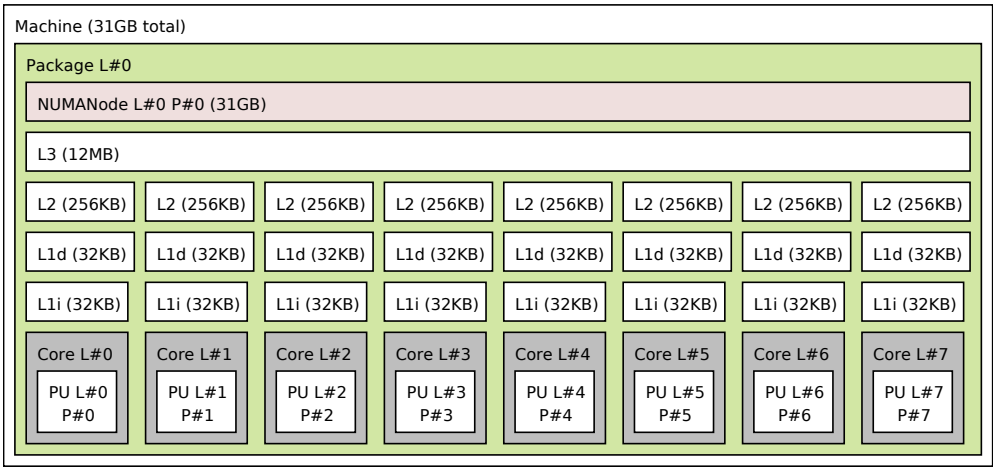



**Figure A.4:** Desktop machine architecture.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# Appendix B

# Kernel isoefficiency

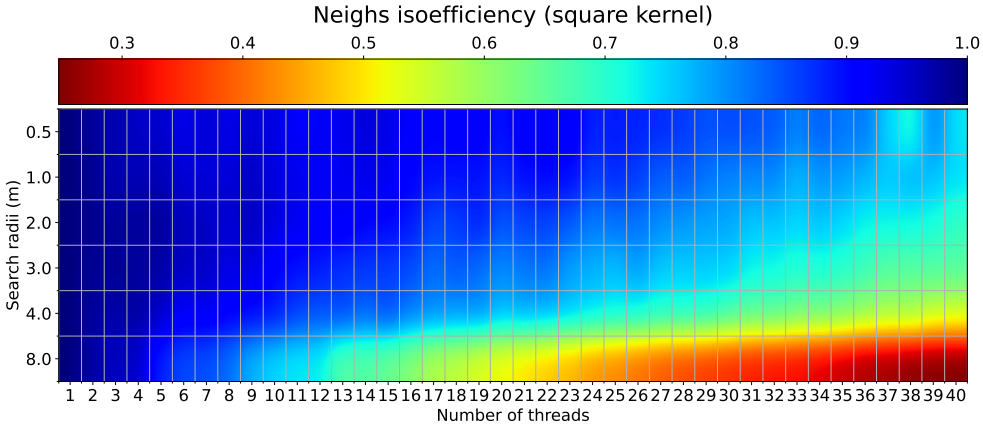



**Figure B.1:** Isoefficiency matrix for the square kernel in Machine 3.

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

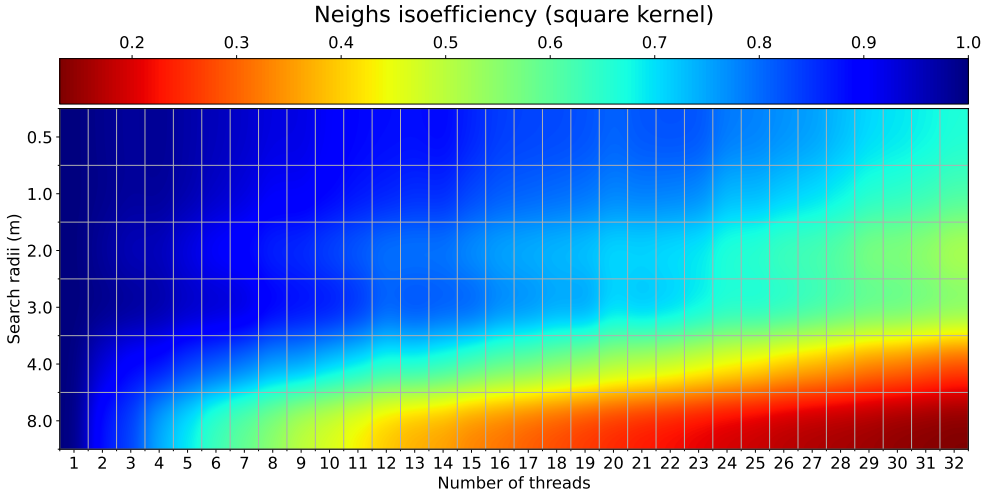



**Figure B.2:** Isoefficiency matrix for the square kernel in Machine 2.

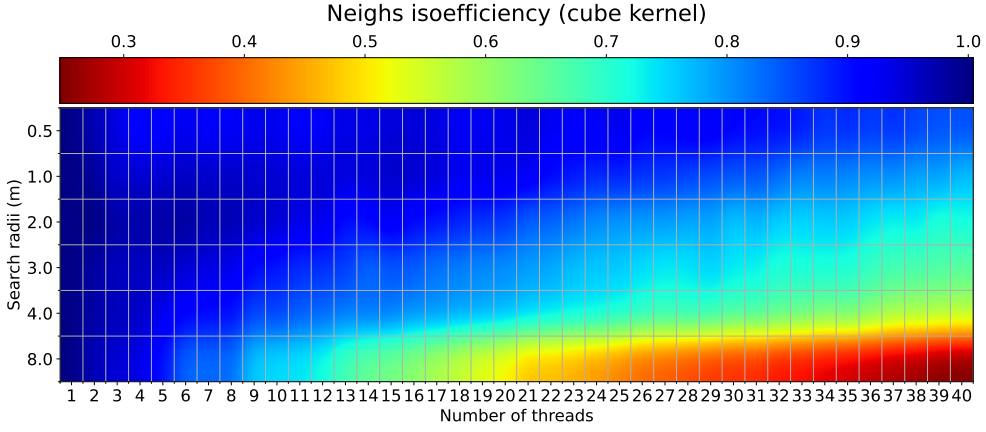



**Figure B.3:** Isoefficiency matrix for the cube kernel in Machine 3.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

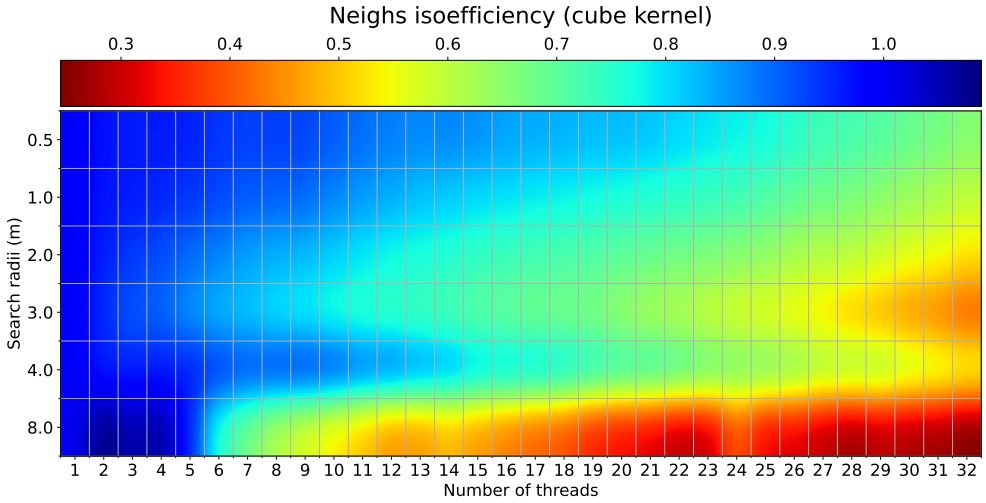



**Figure B.4:** Isoefficiency matrix for the cube kernel in Machine 2.

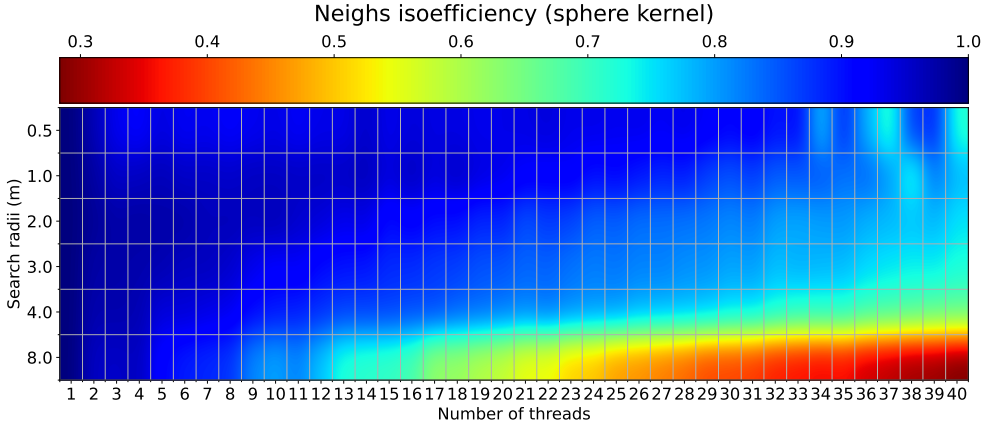



**Figure B.5:** Isoefficiency matrix for the sphere kernel in Machine 3.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

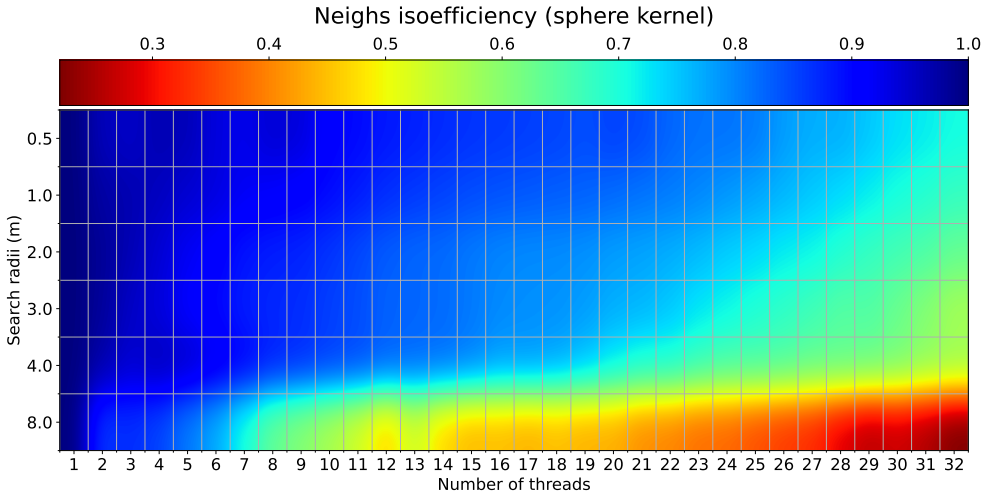



**Figure B.6:** Isoefficiency matrix for the sphere kernel in Machine 2.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

# Appendix C

# Copyright and permissions

## C.1 Articles

- M. Yermo *et al.*, "A fast and optimal pathfinder using airborne LiDAR data", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 183, pp. 482–495, 2022, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2021.11.014.


> The article was published under a Creative Commons Attribution 4.0 International License (CC BY 4.0). The license permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. For more information on this license, please visit: https://creativecommons.org/licenses/by/4.0/.
>
> Additionally, the journal states that permission is not required to reproduce the contents of the article: "*Please note that, as the author of this Elsevier article, you retain the right to include it in a thesis or dissertation, provided it is not published commercially. Permission is not required, but please ensure that you reference the journal as the original source. For more information on this and on your other retained rights, please visit:* https://www.elsevier.com/about/policies-and-standards/copyright#1-author-rights."


USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

- M. Yermo *et al.*, "Powerline Detection and Characterization in General-Purpose Airborne LiDAR Surveys", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024. DOI: https://doi.org/10.1109/JSTARS.2024.3396522.

The article was published under a Creative Commons Attribution 4.0 International License (CC BY 4.0). The license permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. For more information on this license, please visit: https://creativecommons.org/licenses/by/4.0/.

## C.2 Figures

Unless otherwise indicated, all figures are original creations. The figures that are not original creations are properly cited in the corresponding captions, are used under the terms of the licenses specified by the authors, and the licenses are specified in the captions of the figures.



USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

This thesis addresses the processing of LiDAR point clouds using high-performance computing techniques.
By employing efficient data structures and the shared-memory parallelization paradigm, two methods have been implemented for point cloud analysis.
First, a path planning algorithm is used to find the route between any two points within an airborne LiDAR point cloud, considering terrain features such as trafficability, slope, roughness, presence of vegetation, and roads.
It is guaranteed that the found route is optimal in terms of cost.
Second, the problem of detecting and characterizing powerlines in general-purpose airborne LiDAR point clouds has been tackled. The method can detect multiple powerlines in a given scene with a precision of 97.2%, and it can model the conductors with a mean error of 0.14 meters.