ISPRS Journal of Photogrammetry and Remote Sensing 183 (2022) 482–495

Contents lists available at ScienceDirect

ISPRS Journal of Photogrammetry and Remote Sensing

journal homepage: www.elsevier.com/locate/isprsjprs

A fast and optimal pathfinder using airborne LiDAR data

Miguel Yermo [a,*], Francisco F. Rivera [a,b], José C. Cabaleiro [a,b], David L. Vilariño [b], Tomás F. Pena [a,b]

[a] Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Rúa de Jenaro de la Fuente Domínguez, Santiago de Compostela 15782, Spain

[b] Departamento de Electrónica e Computación, Escola Técnica Superior de Enxeñaría, Universidade de Santiago de Compostela, Rúa Lope Gómez de Marzoa, Santiago de Compostela 15782, Spain

ARTICLE INFO

Keywords:
DTM
Airborne point cloud
Pathfinding
Parallel computing

ABSTRACT

Determining the optimal path between two points in a 3D point cloud is a problem that have been addressed in many different situations: from road planning and escape routes determination, to network routing and facility layout. This problem is addressed using different input information, being 3D point clouds one of the most valuables. Its main utility is to save costs, whatever the field of application is. In this paper, we present a fast algorithm to determine the least cost path in an Airborne Laser Scanning point cloud. In some situations, like finding escape routes for instance, computing the solution in a very short time is crucial, and there are not many works developed in this theme. State of the art methods are mainly based on a digital terrain model (DTM) for calculating these routes, and these methods do not reflect well the topography along the edges of the graph. Also, the use of a DTM leads to a significant loss of both information and precision when calculating the characteristics of possible routes between two points. In this paper, a new method that does not require the use of a DTM and is suitable for airborne point clouds, whether they are classified or not, is proposed. The problem is modeled by defining a graph using the information given by a segmentation and a Voronoi Tessellation of the point cloud. The performance tests show that the algorithm is able to compute the optimal path between two points by processing up to 678,820 points per second in a point cloud of 40,000,000 points and 16 km^2 of extension.

1. Introduction

LiDAR technology is a remote sensing technique that allows the 3D modeling of any environment. It has a wide range of practical applications in different fields, such as archaeology, agriculture, and autonomous vehicles. Finding the least cost path is a topic that has been widely studied in recent years. In this work, we aim to provide a fast method to find the Least Cost Path (LCP) in a point cloud obtained from airborne laser scanning without the need of using a Digital Terrain Model (DTM) to rasterize the scene and, therefore, avoiding the drawbacks of generating a rasterization to compute different terrain features.

Being able to find the least cost path accurately is an essential subject in many areas, such as civil works involving the design of roads Parsakhoo and Jajouzadeh (2016), transmission lines Bagli et al. (2011) or rails Cowen et al. (2000). The benefit of taking optimal routes into account is twofold: On the one hand, the construction costs are lowered to the minimum by applying these techniques. On the other hand, the transportation costs are also lowered since the built routes are optimal in terms of the cost to be transversed.

The primary strategy to obtain the LCP from an airborne point cloud is to rasterize the scene, typically applying some well-known algorithms to compute the DTM and using the information provided by the DTM to find the optimal routes. In this way, the problem resembles an image and can be addressed by image processing approaches. There are two main drawbacks when using a rasterization: loss of information due to the modelization of the point cloud into a DTM, which ignores a large number of points, and introduction of errors depending on the chosen DTM algorithm Aguilar et al. (2005), Zhou and Liu (2004), Fisher and Tate (2006).

Furthermore, in some situations, the time required to compute a solution is as important as the accuracy. For example, when computing escape routes in emergencies Wang et al. (2014), Campbell et al. (2017), Liu et al. (2006), the routes must be re-evaluated when the landscape conditions change. To the best of our knowledge, no author has

* Corresponding author.
E-mail address: miguel.yermo@usc.es (M. Yermo).

https://doi.org/10.1016/j.isprsjprs.2021.11.014
Received 3 August 2021; Received in revised form 11 October 2021; Accepted 10 November 2021
Available online 7 December 2021
0924-2716/© 2021 The Author(s). Published by Elsevier B.V. on behalf of International Society for Photogrammetry and Remote Sensing, Inc. (ISPRS). This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

performed a computational cost analysis or an optimization of an algorithm to find the optimal path over a 3D point cloud.

In this work, we propose a LCP computation algorithm that is both accurate and computationally efficient. Regarding the accuracy, every point available in the point cloud has been taken into account to compute the slopes, the terrain roughness, the trafficability of the paths, and the presence of low vegetation, instead of performing a rasterization of the scene that produces a lack of information. Regarding the execution times, several parallelization techniques were implemented and analyzed to reduce the algorithm's computational burden.

The sequence of the paper is structured as follows: In Section 2 an up-to-date overview of some of the most relevant methods to find the LCP over point cloud is presented. Section 3 describes in detail the algorithm used to build the graph, compute the weights of its links and find the least cost path between two points. Section 4 shows the LCPs computed using these techniques with several point clouds. Section 5 shows several performance tests and the parallelization strategies that have been adopted. In Section 6, we discuss the advantages of this algorithm over the state of the art. Finally, in Section 7, the conclusions of the paper are exposed.

2. Related work

When solving the problem of finding the least-cost path, the use of nodes and links and, therefore, a graph is the general approach. In the literature, it is possible to find a great diversity of LCP algorithms: Dijkstra's Algorithm Dijkstra (1959), A* Hart et al. (1968), the Bellman-Ford's algorithm, published independently in Bellman (1958) and Ford (1956), and the Floyd-Warshall's algorithm Floyd (1962) among others. Every one of these algorithms relies on the use of a graph, so both the way a graph is built and the cost is assigned to their links play an important role in the robustness of the method and in the quality of the solution. Of those LCP algorithms, most common is Dijkstra's, used for example in Sarı and Sen (2017),Parsakhoo and Jajouzadeh (2016), Rees (2004) and Contreras and Chung (2007). Dijkstra's algorithm is also built into the network and spatial analyst extensions of ArcGIS software.

Path planning can be divided into two main branches: global and local pathfinding. In the case of global pathfinding, the aim is to find the optimal path connecting any two points in a static point cloud, where the obstacles' position, size, and shape are known in advance. In the local case, the information provided periodically by several sensors (LiDAR, Radar) is used for dynamic obstacle detection Campbell et al. (2020).

Regarding global pathfinding, a method is proposed in González de Santos et al. (2021), where the point cloud of a building is segmented and discretized in a 3D grid. The grid is later used to apply pathfinding algorithms between different rooms of the building. In Zheng et al. (2020), a 3D path planning algorithm for unmanned aerial vehicles (UAVs) is proposed. The algorithm uses a point cloud as input, and several techniques are used to prevent the discretization of the cloud and find the optimal path between two points while avoiding obstacles.

Local path planning is usually combined with global pathfinding. For example, in Li et al. (2021) terrestrial LiDAR point clouds were used to detect obstacles on a route that has been precomputed using OpenStreetMap and Dijkstra's algorithm. In Benelmir et al. (2020), Dijkstra's algorithm is also used to precompute a path, and corrections to the route are calculated by applying the simulated annealing technique. A LiDAR sensor is used in Zhang et al. (2017) to construct a 2D grid, later used to compute a global path. The same sensor is used for obstacle detection.

Many authors have addressed the problem of finding the LCP by using DTMs over a specific area. For example, Metz et al. (2011) compares the effectiveness of using the LCP against other methods in order to extract drainage networks in DTMs of a given area. In Bagli et al. (2011), the cheapest route for building powerlines is found using a combination of multi-criteria evaluation and LCP. The selection of a highway route also could be made by using the LCP computed over a DTM, as shown in Sarı and Sen (2017). Parsakhoo and Jajouzadeh (2016) compute the route for a road in the forest with minimum construction cost through the use of the LCP. In Rees (2004), LCPs are calculated by applying Dijkstra's algorithm to a DTM in order to compare the calculated LCPs with the existing footpaths. Another suitable application of the computation of the LCP over a DTM is finding the localization of optimal log landing, as presented in Contreras and Chung (2007). Liu et al. (2006) developed a LCP based method in order to find evacuation routes in flood disasters areas. In Aguiar et al. (2021), different LCP algorithms were presented to minimize the economic cost and environmental impact of road construction in the Amazon rainforest. Finally, Flisberg et al. (2021) used a DTM to compute the optimal log extraction routes in forest areas while minimizing the negative impact that these operations have on soil and water.

In addition, other authors directly use remote sensing data to find the LCP in a given scene with different applications. For example, in Campbell et al. (2017), the effects that several factors have on travel rates were studied by using airborne LiDAR data, but the use of a DTM is still needed to compute both the slope and the roughness of the terrain, and, therefore, the LCP. In Peng et al. (2017), an airborne LiDAR point cloud and the derived Digital elevation model (DEM) were used to find ventilation corridors through the LCP. The authors addressed that using a 10 m resolution grid could lead to the omission of very detailed features. In Verbrugghe et al. (2017), a 5 m resolution grid obtained using LiDAR data in the rural areas and photogrammetry data in the urban areas are used to locate the Roman road network using LCP over a cost map. Another example of the use of LiDAR-derived DTM is provided in Jones et al. (2008). In this case, a 1 m resolution DTM is used to carry out the hydrologic analysis of low-relief landscapes.

Most DTMs are obtained from a point cloud generated using the data acquired from either airborne laser scanning (ALS) or aerial imagery. Rasterizing point clouds into a DTM has two significant side effects: First, it implies an intrinsic source of errors. For example, when multiple echoes are registered in forested areas, points with similar horizontal coordinates but different elevations are detected, which makes it difficult to represent these points on a regular grid Axelsson (1999). In addition, the interpolation of height in cells without points may be needed Özcan and Ünsalan (2017). Second, the rasterization of a point cloud assigns a single value for each cell based on the set of points contained in it. Two important error sources of DTM are those derived from the interpolation of LiDAR data and those resulting from the differences between the actual shape of the terrain surface being modeled and its representation as a DTM Fisher and Tate (2006). Thus, the choice of the interpolation method has a direct impact on the accuracy of the interpolated heights from scattered data in DTMs Aguilar et al. (2005). Furthermore, when using a DTM to compute landscape slopes, variations of the slope could occur in areas of landscape smaller than the DTM resolution. For example, Contreras and Chung (2007) points that his 50 m resolution grid could not accurately describe terrain conditions. A study of how DTM data precision, grid resolution, and orientation affect the calculation of the slope of the terrain was carried out in Zhou and Liu (2004).

Regarding the computation of the slopes when using a DTM, several techniques are typically used. For example, Campbell et al. (2017) transformed the height difference and distance between adjacent cells in a slope by using trigonometric techniques. In Rees (2004), the slope percentage was computed as dh/dx, where dh is the height gained, and dx is the horizontal distance traveled. Several methods to compute the slope in a grid are analyzed in Tang et al. (2013). In these methods, the standard approach uses 3×3 moving windows to compute the finite differences or the local polynomial surface fits. In the best cases, these methods yield a maximum of 8 points to compute the slope of a given cell and only two points in the worst case.

The main problem of the methods mentioned above is that they compute a single slope value for each DTM cell, which implies that the cost to traverse the path between two adjacent cells has a weight

associated with one of the two cells, not with the path traveled. Furthermore, the direction of the slope is ignored, giving the same importance to an uphill slope, a downhill slope, and traversing the slope perpendicularly, not gaining or losing any height at all. In this paper, we propose an accurate method to compute the slope (both transversal and longitudinal) of a link taking into account the topology of the terrain. In Hoppe et al. (1992), each point's normal is defined as the normal of the best-fitting plane, which is computed using the least square method with the neighbors of the target point. On the one hand, a comparison of several normal estimation methods is provided in Klasing et al. (2009), which are mostly divided into two categories: optimization-based methods and averaging methods. The optimization methods are based on the minimization or maximization of some target criteria, such as the distance of every point to a local plane or the angle between the tangential vectors and the normal vector. On the other hand, the averaging methods are based on averaging the normal vectors of all possible triangles formed by a point and its neighbors. Finally, Klasing et al. (2009) concluded that the optimization-based methods, specifically *PlaneSVD*, *PlanePCA* and *QuadTransSVD* outperform the averaging methods in terms of speed, even in small neighborhoods.

Regarding slope factors, many efforts were performed over the years to quantify the cost of traverse a steep terrain effectively. The oldest known estimation for human walk speed was performed by William W. Naismith in 1892 Naismith (1892). Several authors have developed their own models. For example, the Tobler's Hiking Function, presented in Tobler (1993), is based on empirical results and it is commonly used in GIS travel time computations Irmischer and Clarke (2018). Different effort factors are proposed in Contreras and Chung (2007) as function of the slope (see Table 2). In Davey et al. (1994), a mathematical model was derived by collecting experimental data of a runner in a treadmill with different inclinations. A polynomial model for computing the cost of traverse a slope was developed in Rees (2004), based on unpublished studies conducted by the same author. Another relationship between the slopes and the travel rates was presented in Irmischer and Clarke (2018), where the modelization of the movement cost as a function of the slope is performed through exponential functions. In Campbell et al. (2019), three probability distribution functions (Laplace, Gauss, and Lorentz) were fitted to data obtained from 421,247 hikes, jogs, and runs. They recommend using the Lorentz distribution, as it outperforms the other ones in a more significant number of situations.

3. Pathfinding approach

In this work, we propose an algorithm to find the LCP using an airborne point cloud as input. The path-planning is based on minimizing the cost of the global path in terms of slope, roughness and presence of low vegetation. The proposed algorithm works both with classified and unclassified point clouds. Particularly, in the case in which the classification information is provided, the algorithm can determine the trafficability of the path based on the classification of the points surrounding them. Also, the rasterization of the point cloud is avoided to preserve the information of every point in the scene. A comparison of the methods with other path-planning algorithms can be seen in Table 1.

The algorithm starts by performing a segmentation of the point cloud. This segmentation will serve as input to construct a directed weighted graph. The graph is defined by nodes (vertices) and edges (links) joining two neighboring nodes. The heart of the algorithm is the weight computation of every edge contained in the graph. In this paper, a combination of several factors is taken into account to compute the final weight of every edge: the edge length, the slope, the roughness, and the NDVI value. Also, if available, the classification of the points is taken into account to determine whether the edge is trafficable or part of a road. Once the graph is weighted, Dijkstra's algorithm is applied to determine the optimal path between any pair of points. A flowchart of the algorithm is shown in Fig. 1.

3.1. The neighborhoods

One of the most resource demanding operations when processing 3D point clouds is the extraction of the neighborhood of each point, which allows to compute different descriptors accurately, such as the tangent plane, its normal, the roughness and the curvature, and apply different algorithms, such as noise reduction or segmentation. In Klasing et al. (2009), it is stated that the most natural data structure to represent the neighborhood of a point is an undirected graph, and that there are two types of graphs commonly used for the estimation of the normal vector of a point: the k nearest neighbors and the Delaunay tessellation.

In our work, we propose to work mainly with two types of neighborhoods: On the one hand, those provided by a graph built using a Delaunay tessellation. These neighborhoods will be later used to determine which segments contribute to the optimal path. On the other hand, the local neighborhood of each point of the cloud, whose information is useful to calculate the plane to which the point belongs and associated factors, such as the slope, the roughness or the NDVI.

To determine the neighborhood of each point, the method used in Martínez et al. (2016) has been chosen. The neighborhood of a point p is defined as all the points within a voxel of radius r centered at p. The reason for using a voxel instead of a sphere is a lower computational cost when checking if a point belongs to the neighborhood. Point clouds are organized in a data structure called octree to accelerate the search of neighbors. The octree allows to check the neighborhoods with reduced computing effort as only the tree's leaves close to the point have to be analyzed. Moreover, since the octree is a read-only data structure, it is possible to perform operations using parallel techniques in distributed memory, which speeds up the neighborhood calculation considerably.

3.2. The graph

As stated in the introduction, several authors rely on a regular grid to discretize the map and perform the path search. Despite being a valid strategy, it has some drawbacks. By using a grid, there is an implicit directionality when searching the LCP, since an individual cell could be 4-connected or 8-connected (the neighbors are those cells sharing an edge or a corner). This leads to a loss of accuracy and a predictable path,

Table 1
Table showing the differences between different path-planning algorithms.

Reference	Local/ Global	Discretization	Static/ Dynamic	Indoor/ Outdoor	Weights
González de Santos et al. (2021)	Both	Yes	Both	Indoor	No
Li et al. (2021)	Both	No	Both	Outdoor	No
Zheng et al. (2020)	Global	No	Static	Outdoor	No
Zhang et al. (2017)	Both	Yes	Both	Indoor	No
Contreras and Chung (2007)	Global	Yes	Static	Outdoor	Slopes
Parsakhoo and Jajouzadeh (2016)	Global	Yes	Static	Outdoor	Slopes Construction cost
Rees (2004)	Global	Yes	Static	Outdoor	Slopes
Flisberg et al. (2021)	Global	Yes	Static	Outdoor	Length
Sarı and Sen (2017)	Global	Yes	Static	Outdoor	Length
Proposed Method	Global	No	Static	Outdoor	Length, slopes roughness and NDVI

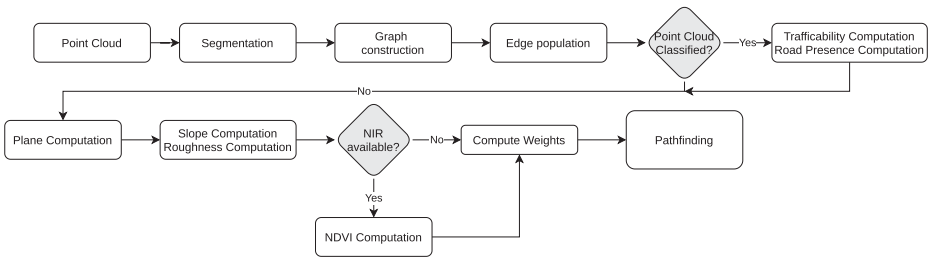


Fig. 1. Flowchart of the pathfinding algorithm.

where each node could have eight possible outputs with previously known directions. Also, when using a rasterized partition of the scene, the terrain details between two adjacent nodes are not considered.

Our goal is to build a graph that does not rely on a rasterized scene. In our method, a segmentation is carried out over a scene, which might be previously classified or not. So, if the scene is classified, then the segmentation will only consider the points classified as ground and roads since they are considered the unique trafficable classes, and, if not, the segmentation is performed over the whole ensemble of points.

The segmentation groups the points by proximity, following a region-growing strategy. Two parameters rule the segmentation: The radius (R_s) and the maximum number of points (N_s). Once a seed point is selected, the points that will belong to that group can not be further than a distance R_s from that point. The creation of the group is stopped when the maximum number of points is reached or when there are no more points to add in the defined neighborhood.

In order to determine the neighboring information between groups, a Voronoi diagram is computed using the centroids of the resulting groups obtained from the segmentation. The Voronoi diagram partitions the scene into irregular polygons with any number of edges. 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, because of the nature of the Voronoi partition. 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.

At this point, a bi-directed graph is defined. The edges of the graph can be traversed in both directions, giving a total of two edges joining every pair of groups. Also, since we intend an edge to contain the weight of traversing the interior of a single group, without taking into account neighboring groups, each edge joining two groups is split into four edges: one edge joining the centroid of the group with the nearest point of the boundary with a neighboring group, and another joining this same point to the centroid of the neighboring group. This process is also carried out in the opposite direction, yielding four edges per pair of groups. By doing this, it is possible to quantify the weights of the path separately depending on the direction chosen to cross the corresponding edges and take into account the topography of the interior of the groups.

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 less the number of points in every group, the more the number of nodes conforming the graph. This versatility allows adapting the graph to every type of point cloud (rural areas, roads, urban environments, etc.).

3.3. The edges

The edge (also called a link) between two adjacent nodes (vertices) is defined as the set of points contained inside a rectangle surrounding an edge. This implementation has a remarkable advantage against using a DTM: it will lead to a more precise feature computation since every point of the point cloud is available to compute the weights. The rectangle is built using as a parameter the so-called Agent Width (A_w), which is the width reference used to compute the LCP. The rectangle is defined by projecting two parallel lines at both sides of the edge at a distance of $A_w/2$. By using these lines, the rectangle is closed, and the edge is built. The area covered by the edge can be seen in Fig. 2a.

Once the edge area has been determined, its bounding box is computed. Using the center of this bounding box, a neighbor search is performed using as radius $R = 0.5 \cdot max(R_x, R_y)$, as shown in Fig. 2b. In this way, we make sure that no point is left out of the edge area. Then we check whether each one of the points belonging to the neighborhood is inside the edge or not. Since the width and length of each rectangle are known, it is possible to compute the coordinates of the four corners, and therefore the equations of the lines that are part of each rectangle. By knowing the equations of the edge borders, computation of whether a point is inside and edge or not is straightforward. The points contained in the edge are shown in yellow in Fig. 2c.

Once the edges are well defined, their trafficability can be established. Trafficability is defined as the capacity of an agent to go through a specific terrain. In order to compute this feature quantitatively, the number of points belonging to roads or ground is considered. So, the edge is tagged as trafficable if the ratio of ground or road points is lower than a certain threshold, as given by Eq. 1:

$$\frac{n_t}{n_p} \geqslant T_t \tag{1}$$

where n_t is the number of points classified as road or ground, n_p is the total number of points in the edge and $T_t \in [0, 1]$ is the Trafficability Threshold, which corresponds to the minimum ratio of trafficable points an edge must have in order to be considered as trafficable. The edges not passing this filter are removed from the graph, and therefore the neighboring information of the nodes must be updated accordingly.

3.4. Edge features

The optimal path between two points is the one that minimizes the required effort to go through it. So, this might not necessarily correspond to the shortest path because of the landscape characteristics. Slopes, roughness, and the presence of vegetation are some of the main features that affect the effort needed to go through a path. In order to find the optimal path, it is needed to assign to each edge a cost of traveling, which is a quantitative measure of the effort needed to traverse that link. This effort can be estimated using the point cloud information directly.

In this paper, as a particular case, the following factors are considered to measure the cost of traveling: length of the edge, road presence, slopes, both longitudinal and transversal, roughness, and vegetation presence. These features are computed using the points belonging to the rectangle assigned to each one of the links.

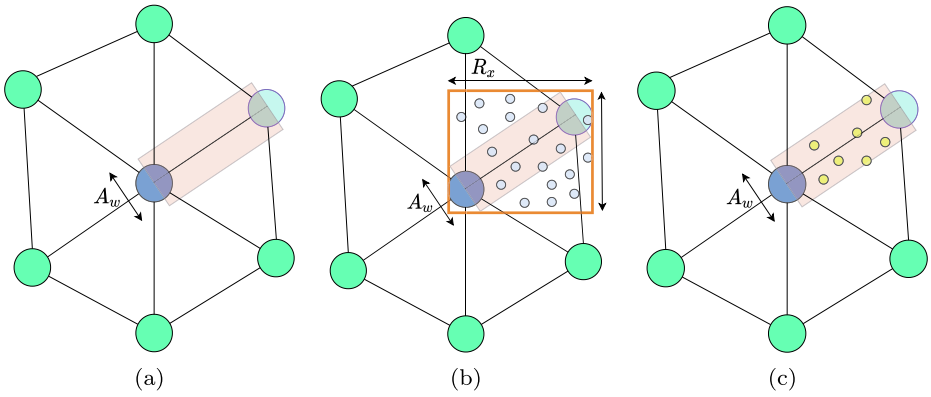


Fig. 2. Determination of the points lying inside a given edge. The blue circle is the graph node whose edges are being built. The neighbor used to build the edge is shown in light blue. The nodes in green are graph nodes to be processed later. The gray circles are points found in the initial search, and the yellow ones are those finally included in the edge. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.4.1. The planes

Computing the slope of an edge is equivalent to computing the plane's normal formed by the points belonging to that edge. In Klasing et al. (2009), a comparison of different methods for estimation of surface normals is presented. In this paper, 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 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 to 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 of the points in each edge is computed using the following expression:

$$\bar{\mathbf{r}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{r_i} \tag{2}$$

where $\mathbf{r_i}$ is the 3D position of each point and $\bar{\mathbf{r}} = (\bar{x}, \bar{y}, \bar{z})$ is the centroid of the set of points. The matrix $\mathcal{P}$ is then defined as follows:

$$\mathcal{P} = \begin{pmatrix} \mathbf{x} - \bar{x} \\ \mathbf{y} - \bar{y} \\ \mathbf{z} - \bar{z} \end{pmatrix} \tag{3}$$

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 can be finally 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

$$\pi \equiv n_x x + n_y y + n_z z + d = 0 \tag{4}$$

where d is

$$d = -(n_x \cdot \bar{x} + n_y \cdot \bar{y} + n_z \cdot \bar{z}) \tag{5}$$

A final consideration must be taken into account regarding the orientation of the computed normals. 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.

3.4.2. The slopes

The slope of each edge is computed as the angle subtended between the normal vector of its plane and a vertical vector. This slope corresponds to the maximum value of the slope, which is the value in the downhill direction of the plane, known as slope aspect, and can be computed using the gradient operator in two dimensions over the normal vector of the plane. The slope aspect is then used to compute the slope magnitude both along the longitudinal (s_l) and perpendicular (s_t) direction of the edge.

3.4.3. Road presence

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. Thus, the ratio between road and not-road points in an edge is computed as shown in Eq. 6:

$$\frac{n_r}{n_p} \geqslant T_r \tag{6}$$

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 Road Ratio, a user-selectable parameter that corresponds to the ratio of road points in the edge.

3.4.4. Roughness

This feature provides information about the terrain smoothness at low-scale. The presence of bumps, vegetation, rocks, or smalls obstacles will increase the roughness, and so it will increase the cost of traversing these areas. In Glenn et al. (2006) the roughness is defined as the standard deviation of the height of the LiDAR points relative to a given surface. This surface is obtained by applying a thin-plate spline interpolation over a previously performed rasterization of the map, and where the lowest point of each cell is used in the aforementioned interpolation. This method provides a single value of the roughness for each cell of the rasterized map.

In this paper, the roughness is defined in a similar way, and can be computed by using two different surface references. On the one hand, the roughness of an edge is calculated as the standard deviation of the distances of the points contained in it to the best fitting plane of the edge. On the other hand, the roughness of each point on the map is calculated as the standard deviation of the distances of its neighboring points to the best fitting plane formed by those neighbors. The differences between the two methods are clear: one method only provides one roughness value per edge, similar to what is done in Glenn et al. (2006). Our method establishes a value of roughness for each point contained in the scene based on its surrounding points, thus allowing a more detailed analysis of terrain features using this descriptor. Note that this gain in accuracy implies a higher computational cost.

Once the roughness is computed by any of the mentioned methods,

these values should be translated to weight factors. In the case of roughness at the edge level, it is enough to design a function that translates the roughness value of each edge to its corresponding weight. For the case of roughness at the point level, it is necessary to define a descriptor that translates the roughness contribution of each point contained in the edge to a single weight value.

To the best of our knowledge, there are no references of a transit cost modeling as a function of terrain roughness. Since non-rough terrains should be prioritized, the cost of traverse rough terrain should be higher than smooth terrain.

Our proposal is the weight to be 1 when the roughness is 0, i.e., when the edge is completely flat, and higher for different values of roughness thereafter. Thus, if the modeled roughness weight function is given by $f_r(r)$, for any value r of roughness, the following must be satisfied:

$$\forall r \in \mathbb{R}_0^+ : f_r(0) = 1;\ f_r(b) > f_r(a)\ \forall\ b > a \tag{7}$$

Note that the contribution of the roughness of each point must be translated into one single value, and the result must be independent of the number of points contained in the edge, so the average roughness computed by using the points of the edge is chosen in that case. Finally, the roughness factor is computed from the roughness value through Eq. 7. For testing purposes, the relationship between the roughness value and its weight was defined as $f(r) = 1 + r$ in all cases, since this criterion meets the one defined in Eq. 7.

3.4.5. The NDVI

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

$$\mathrm{NDVI} = \frac{\mathrm{NIR} - \mathrm{R}}{\mathrm{NIR} + \mathrm{R}} \tag{8}$$

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 shows the presence of living vegetation, and values close to 0 and negative show its absence.

So, for every point of each edge, the NDVI is computed if the corresponding data is available. Finally, the average value of the NDVI is assigned to the corresponding edge.

If NIR data are not available, it is possible to compute several Vegetation Indices (VI) by means of RGB information, as shown in Lussem et al. (2018), following a similar approach.

3.5. The weights

A directed, non-negative weighted graph is built. In this paper, the graph will be weighted with the following proposed cost function for each edge:

$$W = L{\cdot}P{\cdot}S_l{\cdot}S_t{\cdot}R{\cdot}N \tag{9}$$

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 the Euclidean distance between two adjacent nodes. In order to quantify the advantages of traveling by roads, P must meet the following condition: $P = 1$ if the edge is not classified as road, and $0 < P < 1$ otherwise.

Regarding the slope factors, S_l and S_t, the criterion designed by Contreras and Chung (2007) was chosen, and it is detailed in Table 2.

As it was mentioned before, R refers to the weight of an edge in terms of its roughness, which is computed by Eq. 7. Some example values are shown in Table 3. To the best of our knowledge, no studies to date have derived any model of the cost of traversing a surface attending to its 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% Campbell et al. (2017).

Table 2
Longitudinal and Transversal slope factors.Contreras and Chung (2007).

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

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 Eq. 9, $L \in (0,\infty)$. The other parameters are multiplicative factors that increase or decrease this value depending on the feature they represent. 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 and vice versa. The only factor, besides L, that can have a value less than 1.0 is P since its purpose is to reduce the weight of an edge if considered a road.

3.6. Algorithms to obtain the optimal path

Two of the more commonly used algorithms for performing path searches are the A* Hart et al. (1968) and the Dijkstra's algorithm Dijkstra (1959). 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 selection of the optimization function is crucial since some of them could lead to overestimating the path cost, leading to sub-optimal solutions of the global path. The advantage of the A* algorithm over Dijkstra's algorithm is that A* is considerably faster in most situations. Since our goal is to provide the optimal path every time, Dijkstra's algorithm is the selected algorithm to carry out the LCP search. In our proposal, it runs over the well-known Compressed Sparse Row (CSR) method to store sparse matrices.

Let N_e be the number of edges in the graph and N_n the number of nodes. In our method, the equivalent to an adjacency matrix is implemented by using two arrays, A and B. These two arrays efficiently store the information regarding the number and the identity of neighbors of every node conforming the graph. A is an $1 \times N_n$ array storing the starting index of the B array, where the corresponding edges are stored. B is an $1 \times N_e$ array that contains the identity of the neighbors of every node. Furthermore, an additional array (W) of dimension $1 \times N_e$ is defined to store the weights of each one of the edges in a similar way that B stores the identity of the neighbors. Thus, the weight of the edge linking node i and its neighbor j of that edge is W_{A_i+j}. The most significant drawback of this data structure is that it needs to be rebuilt to apply any changes to the graph. Since the data structure is only established

Table 3
Roughness factors.

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

after the edges are computed and the weights are assigned, this issue can be ignored.

4. Examples and results

The point clouds selected for testing are airborne LiDAR point clouds, and they were obtained from both public and private repositories. The dataset containing infrared information correspond to the Spanish region of La Rioja, and it were provided by the Spanish Geographic Institute and the Government of La Rioja, in the frame of the National aerial orthophotography plan (PNOA). The orthomosaic of this region is shown in Fig. 3c. This cloud have a point density of 3 p/m^2. Furthermore, a more dense set of point clouds that do not include infrared information were used, whose average density is 15 p/m^2, and were privately provided by Babcock International Babcock International (2020). These point clouds correspond to rural areas surrounding the city of Lugo (Spain) and can be seen in Fig. 3a and b.

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 Martínez et al. (2016), which includes the methods described in Martínez Sánchez et al. (2020). Several variations of the segmentation parameters N_s and R_s are tested for each point cloud to obtain different edge mean length, group sizes, and number of neighbors per group. Therefore, different configurations of the trafficability graph can be generated. As default values for these experiments, $T_r = 0.9$ is used to determine whether an edge is considered a road or not. Also, $T_t = 0.9$ is used to determine the trafficability of an edge, and the edges with longitudinal slopes greater than 0.30 are considered as non-trafficable. Users can easily change these parameters.

Also, a comparison between grid-based methods and our method is carried out to show the advantages of not using predefined cells to divide the point cloud. A study of the influence for each one of the weights included in Eq. 9 was carried out.

4.1. Segmentation

As stated initially, a segmentation of the already classified point cloud is carried out to obtain groups to build up a trafficability graph. In this graph, each group is represented by a node. Eventually, the groups might contain even one point, yielding a graph where each node corresponds to one point in the cloud. The advantages and disadvantages of this situation are analyzed in this section.

The choice of segmentation parameters, described in Section 3.2, will depend on the average point density of the point cloud. Let be ρ the average point density (in p/m^2). Since we are using square-shaped neighborhoods, a given neighborhood around a seed will have an average of ρR_s^2 points. So, depending on the chosen values of these parameters, certain combinations will yield identical segmentations of the scene. Furthermore, these parameters will rule, in an indirect way, the length of the edges of the graph. To show results, a test case with the following combinations of these two parameters were considered:

(a) (b) (c)

Fig. 3. Orthomosaics of the areas used in the different tests.

$$R_s = \{1.0, 2.5, 5.0, 7.5, 10.0\} m$$
$$N_s = \{100, 500, 1000, 2500, 5000, 7500, 10000\} points \quad (10)$$

So, by exploiting all the combinations, 35 segmentations were generated. Some of the most representatives are shown in Fig. 4. 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 using the centroids of every group, the presence of deformed groups is a problem because the computed centroid could be outside the group. To deal with this issue, we choose as default values $R_s = 2.5$ m and $N_s = 2500$ points, which yield to a compact and uniform segmentation, where all the groups are convex.

4.2. Graph

As an example, the Voronoi Tessellation of the segmentation in Fig. 4b ($R_s = 2.5\ m, N_s = 2500$) is shown in Fig. 5a. 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. Fig. 5b shows 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: cells whose centroids are far apart in the scene are considered neighbors because of the very long edges keeping contact with several cells. A detailed example of this phenomenon is shown in Fig. 5c, where the graph resulting from the Voronoi Tessellation is shown. As expected, there are very long edges in the borders that must be removed. 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. 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 Fig. 6. The color codification of the different classes shown follows the LAS v1.4 specification, which applies to every figure showing classification data. This graph now sticks perfectly to the graph borders. Also, the graph is built only over groups of segmented points classified as road or ground, ignoring buildings (in red) and vegetation (in green).

4.3. 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, as shown in Section 3.4.1. This method allows the extraction of the actual slope of the terrain and the transversal slope value, which is also essential to 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 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. The overlap is not an issue since the same point can be used several times in different planes. Only the points close to the nodes are shared by two edges.

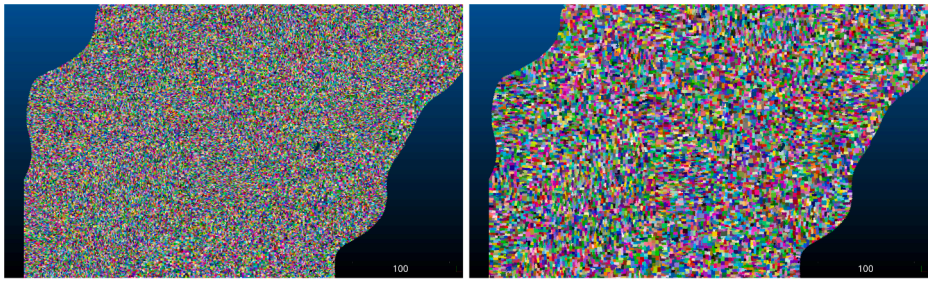
4.4. Roughness

As discussed in Section 3.4.4, roughness can be computed at edge level or point level. The difference in accuracy between the two methods is significant, and so is their computational cost, which will be discussed later.

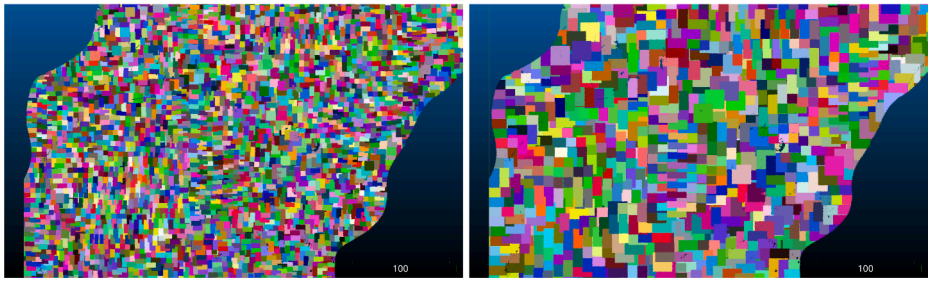
An example of the calculation of roughness, both at edge and point level, is shown in Fig. 7. As it can be observed, the roughness intervals are very similar in both scenes: $(0.00, 0.67)$ m for point-level roughness and $(0.00, 0.70)$ m for edge-level roughness. The roughness values in the interval $(0.0, 0.1)$ m correspond to points or edges that belong to the ground and will be used later in the path searching. The higher roughness values correspond to points or edges on the corners of buildings or high vegetation. Note that these values are more widely distributed when the calculation is at point level than 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.

Roughness is taken into account in the path-finding algorithm, such that the path prefers to run through smoother areas (blue) rather than rougher areas (green). The effect of the roughness in the weight computation was analyzed, and an example is shown in Fig. 8. Note how the consequence of taking into account the roughness in the algorithm

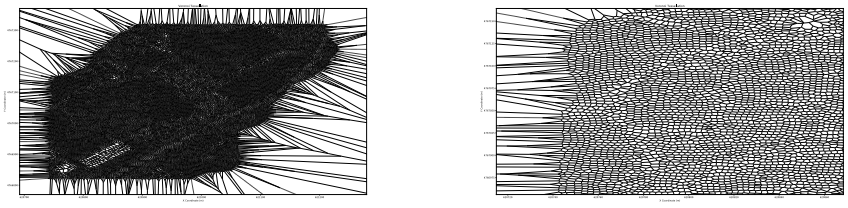


(a) $R_s = 1.0\ m\ N_s = 1,000$ (b) $R_s = 2.5\ m\ N_s = 2,500$

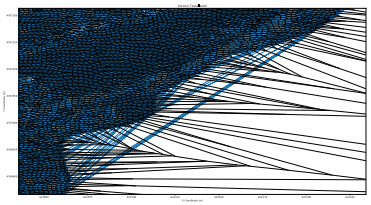


(c) $R_s = 5.0\ m\ N_s = 2,500$ (d) $R_s = 10.0\ m\ N_s = 10,000$

Fig. 4. Segmentations generated with different combinations of the parameters R_s and N_s. The scale shown in the bottom right corner of each map is given in meters.

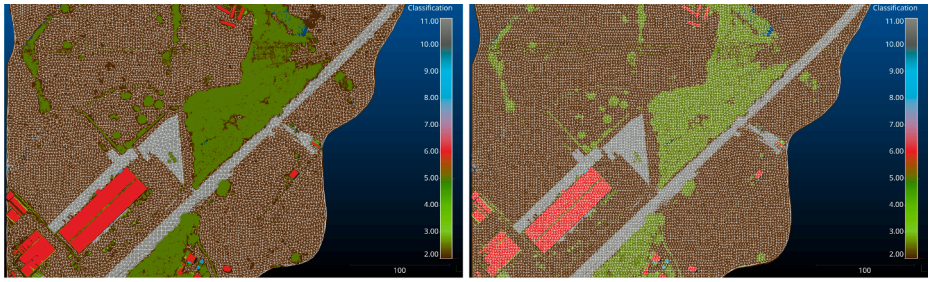


(a) Voronoi Tessellation of a given point cloud (b) Detail of the Voronoi Tessellation



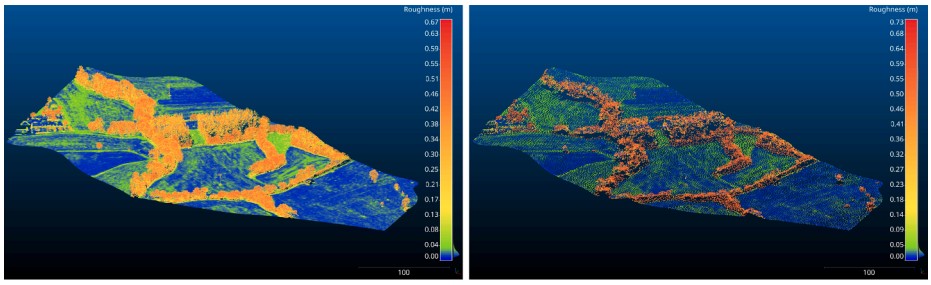
(c) Detail of long edges in the borders of the point cloud

Fig. 5. Voronoi tessellation of a point cloud.



(a) (b)

Fig. 6. Resulting graph using the neighboring information of the Voronoi Tessellation. (a) Graph with classification data. (b) Graph without classification data.



(a) (b)

Fig. 7. Surface roughness. Roughness computed at point level. (b) Roughness computed at edge level.

has two effects: On one hand, the path lies on areas in which the roughness is the lowest. On the other hand, since the obstacles are expected to have a high roughness value (high vegetation, represented primarily in orange + in Fig. 8 for instance), the inclusion of the roughness in the computation allows the algorithm to avoid obstacles even if they are not classified as such.

4.5. NDVI

The Normalized Difference Vegetation Index can only be computed at point level. If the point cloud is not already classified, the NDVI is helpful, when available, to avoid areas with the presence of crops or low vegetation. Also, it is helpful to force the path's cost to be the lowest when using human-made roads since they have a low NDVI value.

To the best of our knowledge, no studies attempt to relate the NDVI index to a weight factor for the trafficability of a path, so we will follow a similar approach to that used with the calculation of roughness weights. The NDVI of each point of 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:

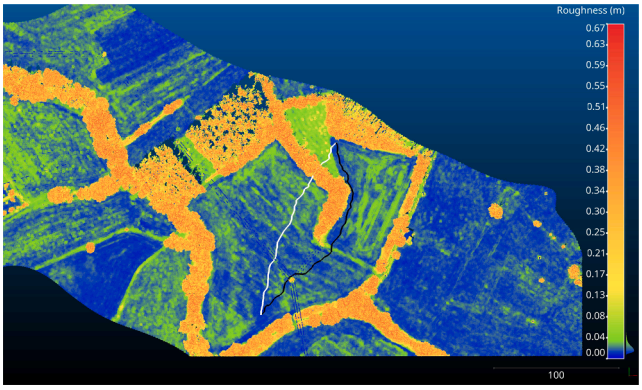


Fig. 8. Example of paths computed with and without taking the roughness into account. The path with the white color corresponds to that computed without including the roughness in the weight computation, and the black path is the one including it.

$$f_n(x) = \max(0, x), \;\; x \in [-1, 1] \tag{11}$$

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:

$$W_{ndvi} = \frac{1}{n}\sum_{i=1}^{n} f_n(p_i^{ndvi}) = \frac{1}{n}\sum_{i=1}^{n} \max(0, p_i^{ndvi}) \tag{12}$$

where p_i^{ndvi} is the NDVI value of the ith point and f_n the function defined in Eq. 11.

An example of the behavior of Dijkstra's Algorithm when searching paths taking as weight factors both the NDVI and the length of an edge is shown in Fig. 9, where two paths are depicted. The white path has been calculated considering 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 algorithm 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 on the left side of Fig. 9.

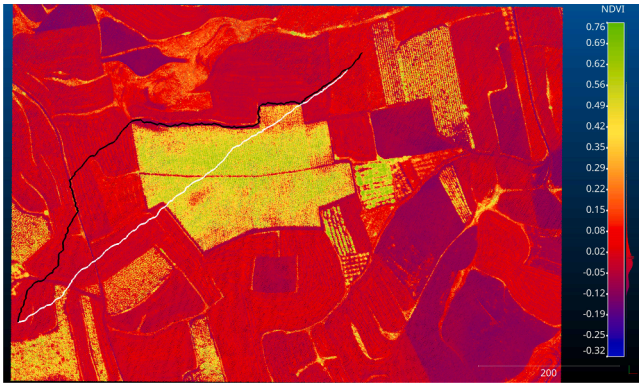


Fig. 9. Path found using the weights given by the NDVI values. In blue, the path found without using NDVI values is shown. In white, the same path found using the NDVI values of each edge.

5. Computational performance

In order to evaluate the computational performance, two tests were carried out using a C++ implementation of the algorithm (ISO/IEC 14882:2017). In both tests, La Rioja maps were used due to 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, with a density of 3.52 p/m^2. The algorithm running on this map was analyzed using an Intel(R) Core(TM) i7-9700 K CPU @ 3.60 GHz, with eight physical cores and 32 GiB of RAM. The compilation has been done with GNU g++ 9.3.0. 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 p/m^2. This experiment was conducted on an Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50 GHz, with 24 physical cores spread over two sockets. The RAM was 64 GiB. In this case, the compilation has been done with GNU g++ 10.1.0. In both cases, the optimization flag was -O2.

Regarding the parallelization strategy, a shared memory model has been chosen, and particularly the OpenMP v4.5 OpenMP Architecture Review Board (2015) 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 map size to those whose size does not exceed the available memory.

In this case, it is possible to parallelize almost the entire code, with the sole exception of the graph creation algorithm (which involves segmenting the scene and calculating the Delaunay triangulation) and the search for the optimal path between two points because of their inherent dependencies. Although it has been experimentally proven that the execution time of these two algorithms represents a tiny percentage of the total time, it is appropriate to apply Amdahl's law (Eq. 13) to evaluate the maximum speed-up that can be achieved as a function of the number of cores used:

$$S = \frac{1}{(1-p) + \frac{p}{s}} \tag{13}$$

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

Another valid descriptor that shows the quality of the parallelization is the efficiency, defined as the quotient between the achieved speed-up and the number of cores used:

$$E = \frac{S}{n} \tag{14}$$

where n is the number of cores.

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.

The workload of these functions is directly related to the point density of the area where the point or edge is located. This will lead to an unbalanced parallelization, so, in this case, the *dynamic schedule* of OpenMP is used.

The computations of the roughness, both at edge and point level, are mutually exclusive. The cost of computing the roughness at edge level is low compared to the computation at point level, as shown in Fig. 10, but also implies a loss of precision. For this reason, the scalability analysis of the algorithm was performed considering the computation of the

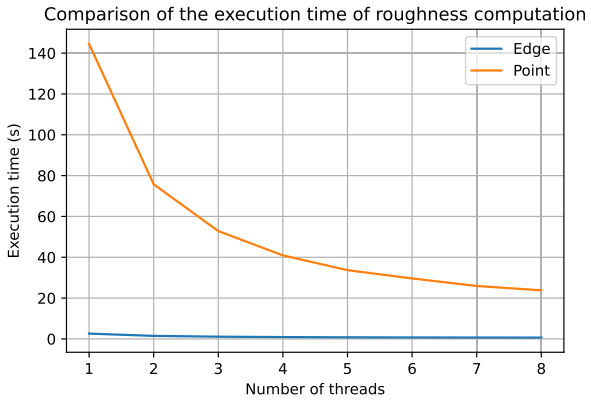


Fig. 10. Execution times in the cases when computing the roughness at edge level and at point level.

roughness at point level.

5.1. Results on Test A

The achieved speed-up per number of cores used in the desktop computer is shown in Fig. 11:

In the case of eight cores, the achieved speed-up is 4.81 while the theoretical maximum speed-up is 6.05, yielding an efficiency of 0.8.

Another parameter that strongly influences the achieved speed-up is the problem size. When the algorithm is applied over a tiny point cloud (in the order of tenths of thousands of points), it is possible that the overhead introduced due to the parallelization mechanisms does not pay off the obtained gain of time. This effect is depicted in Fig. 12. Note that when reaching a problem size of 100,000 points approximately, the overhead introduced starts to be compensated with the speed-up obtained. It can be observed that the speed-up scales quickly and then remains almost constant from that point.

Finally, the combined effect of both the problem size and the number of cores can be expressed through an isoefficiency matrix Grama et al. (2003), as depicted in Fig. 13. Note that the efficiency scales well both with the number of cores used and the problem size, remaining almost constant when the problem size is big enough. That means that the parallelization performed presents a good scaling.

5.2. Results on Test B

In this case, the speed-up achieved per number of cores is shown in Fig. 14. The behavior is similar to Test A when the number of cores in both cases matches. Beyond that, the speed-up keeps growing parallel to the theoretical speed-up, keeping the efficiency around 0.75.

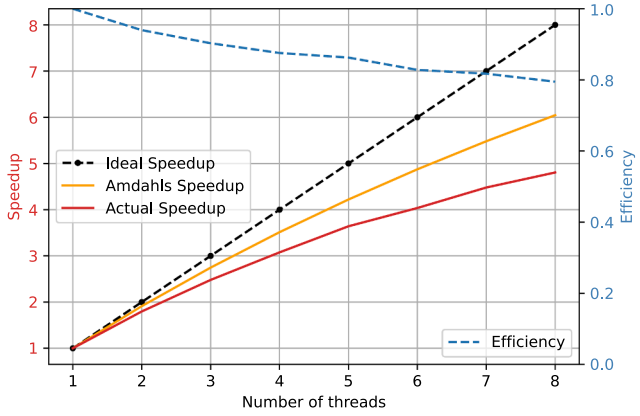


Fig. 11. Speed-up and efficiency achieved on Test A. Problem size: 14,000,000 points.

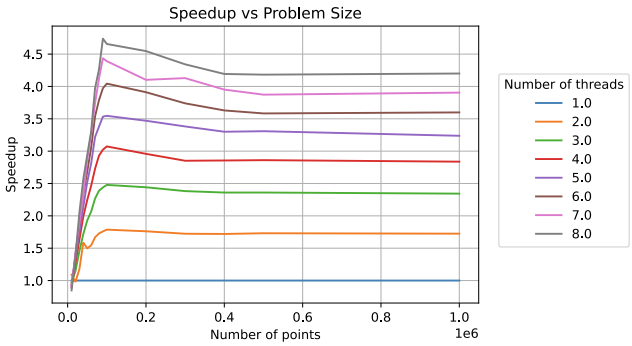


Fig. 12. Achieved speed-up in function of the problem size, given in number of points, for Test A.

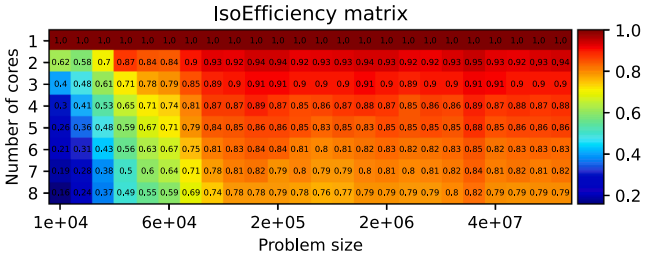


Fig. 13. Efficiency matrix for Test A.

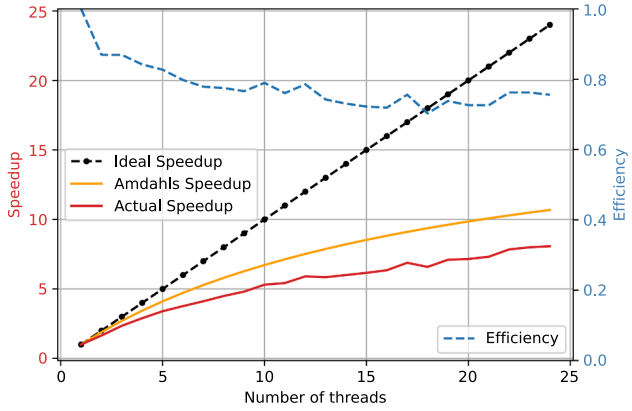


Fig. 14. Speed-up for Test B. Problem size: 40,000,000 of points.

In order to analyze the influence of the problem size in the achieved speed-up, several point cloud sizes were chosen when executing the algorithm. The results shown here were computed as the median of 30 different executions in the same conditions. In Fig. 15, the isoefficiency map is shown. In this case, the efficiency is close to 1 when the number of cores used is low. As the number of cores 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 cores. This effect is known as superlinear speed-up Ristov et al. (2016), and can be observed in highly parallel codes when the size of the problem divided by the number of cores is less than the amount of available cache in each processor.

6. Discussion

Over the graph used to compute the LCP, the properties enumerated in Section 3 are analyzed. It is important to remark that these properties

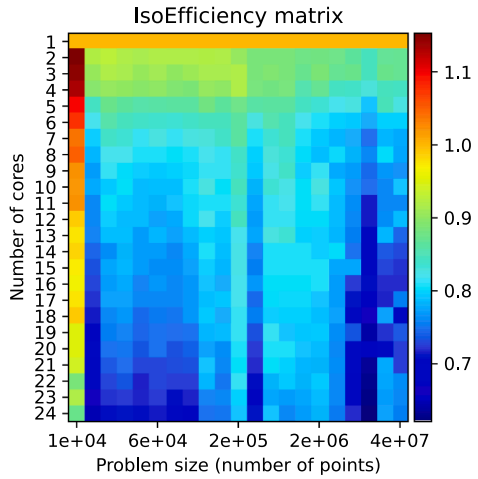


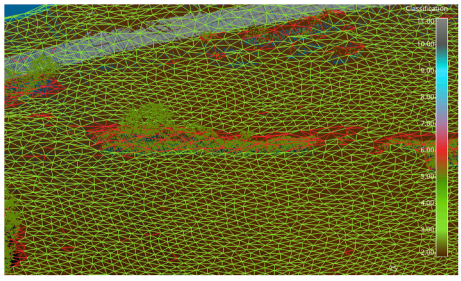
Fig. 15. Efficiency as function of the number of cores and the problem size.

are computed at the level of each edge, instead of using discrete values over rasterized cells, so there is not loss of information associated to the 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. When used, the DTM was implemented based on Pingel et al. (2013).

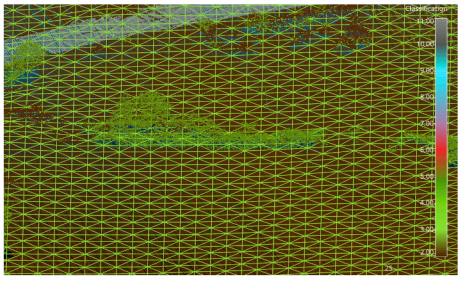
6.1. Number of points considered

In Rees (2004), an area of 9 km × 12 km is represented by a DEM of 50 m resolution. Therefore, each cell has an area of 2,500 m^2, which implies that using a point resolution of only 1 p/m^2 (which is a very pessimistic estimation) is enough to condense the information of 2,500 points in a single height value. Hence, a point cloud containing 108,000,000 points in a DEM holding the information in only 43,200 points, losing 99.96% of the information.

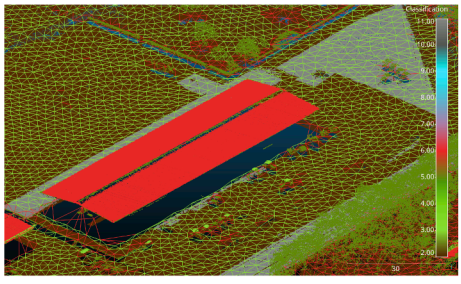
A similar issue occurs when computing the roughness weight factor. In Glenn et al. (2006), a DEM of 5 m resolution, where each cell contains 5–50 data points, is used to compute both the slope and the roughness of the terrain. The roughness is then computed as the standard deviation of the height of each point over an interpolated surface, whose resolution is the same as the DEM. By interpolating, the roughness is being computed



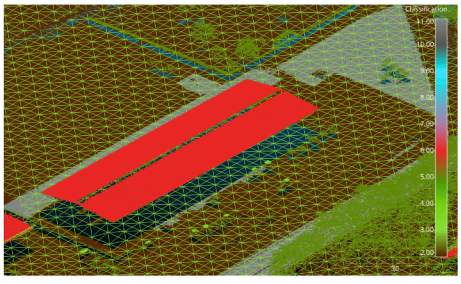
(a) Proposed graph over vegetation



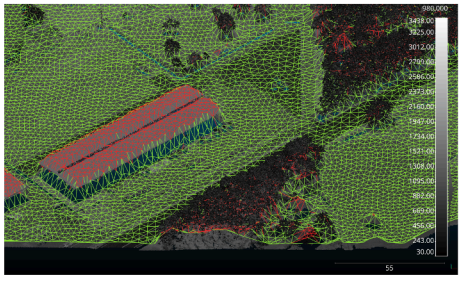
(b) DTM based graph over vegetation



(c) Proposed graph over a building



(d) DTM based graph over a building



(e) Proposed graph trafficability over a non classified building

Fig. 16. Trafficability graph over different areas.

over a virtual surface instead of an actual surface defined by the points. A similar approach is shown in Campbell et al. (2017), using 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 all the available points.

6.2. Trafficability of the path

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

In this paper, and only when the point cloud is previously classified, the trafficability graph is computed by using the expression given in Eq. 1. 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 Fig. 16, the differences between our graph and a DTM based graph are shown. The pictures focus on a little area of isolated vegetation and including a building. The trafficable edges are shown in green, and the non-trafficable ones in red. In Fig. 16a, it can be seen that the vegetation is not trafficable. However, that same area is trafficable in the graph built using DTM information.

Also, no author details the issues when a building is encountered in the scene, as it can be seen in Fig. 16b and c. The DTM can compute the terrain level below the roof of the buildings (Fig. 16c), but then those edges are incorrectly managed as trafficable. In our method (Fig. 16b), since the nodes and neighboring information are computed from a segmentation carried out over the already classified road and ground points, this problem does not appear.

Another parameter influencing the trafficability is the slope. In our work, edges whose slope is greater than S_t, the slope threshold, are considered as non-trafficable since the algorithm aims to find a route that can be traversed without much effort or special equipment. In Table 2, several weights for longitudinal and transversal slope are considered. Slopes greater than 30% (16.70°) are weighted with a very high cost but still could be traversed. For example, in the case where the only possible path includes a very steep slope, the model described in Contreras and Chung (2007) would return a very costly yet possible route. As a result, slopes greater than $S_t = 36.4\%$, which corresponds to 20°, should be 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 be seen in Fig. 16d.

7. Conclusions

In this paper, a model to find the optimal path in terms of transitability between two points has been developed. The advantage of this model is, on the one hand, the use of Dijkstra's algorithm in 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 through a DTM with the consequent loss of accuracy derived from such rasterization. Unlike the works where a DTM is used as a basis for path-planning, in this paper, 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 forcing the found paths to be along roads and tracks has been achieved, even when the scene is not classified. Also, a novel study of the computational costs for this type of algorithm was carried out, demonstrating the algorithm's good scalability in manycore systems. The motivation of this work was the need to find a safe route in emergencies as fast as possible. Thus, this algorithm, in combination with the work presented in Lorenzo et al. (2017), is used to obtain the fastest route between the landing zone of a medical UAV and the site of an emergency.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has received financial support from the Consellería de Cultura, Educación e Ordenación Universitaria (accreditation 2019-2022 ED431G-2019/04, reference competitive group 2019-2021, ED431C 2018/19) 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 (Grant No. PID2019-104834 GB-I00). We also acknowledge the Centro de Supercomputación de Galicia (CESGA) for the use of their computers.

References

Aguiar, M.O., Fernandes da Silva, G., Mauri, G.R., Ribeiro de Mendonça, A., de Oliveira, Junio, Santana, C., Marcatti, G.E., Marques da Silva, M.L., Ferreira da Silva, E., Figueiredo, E.O., Martins Silva, J.P., Silva, R.F., Santos, J.S., Lavagnoli, G. L., Claros Leite, C.C., 2021. Optimizing forest road planning in a sustainable forest management area in the brazilian amazon. J. Environ. Manage. 288, 112332. https://doi.org/10.1016/j.jenvman.2021.112332. URL: https://www.sciencedirect.com/science/article/pii/S0301479721003947.

Aguilar, F., Agüera, F., Aguilar, M.A., Carvajal, F., 2005. Effects of terrain morphology, sampling density, and interpolation methods on grid dem accuracy. Photogram. Eng. Remote Sens. 71, 805–816. https://doi.org/10.14358/PERS.71.7.805.

Axelsson, P., 1999. Processing of laser scanner data—algorithms and applications. ISPRS J. Photogram. Remote Sens. 54, 138–147. https://doi.org/10.1016/S0924-2716(99)00008-8. URL: https://www.sciencedirect.com/science/article/pii/S0924271699000088.

Babcock International, 2020. Babcock international. trusted to deliver. URL: https://www.babcockinternational.com/.

Bagli, S., Geneletti, D., Orsi, F., 2011. Routeing of power lines through least-cost path analysis and multicriteria evaluation to minimise environmental impacts. Environ. Impact Assess. Rev. 31, 234–239. https://doi.org/10.1016/j.eiar.2010.10.003. URL: https://www.sciencedirect.com/science/article/pii/S0195925510001393.

Bellman, R., 1958. On a routing problem. Q. Appl. Math. 16, 87–90. URL: http://www.jstor.org/stable/43634538.

Benelmir, R., Bitam, S., Mellouk, A., 2020. An efficient autonomous vehicle navigation scheme based on lidar sensor in vehicular network. In: 2020 IEEE 45th Conference on Local Computer Networks (LCN), pp. 349–352. https://doi.org/10.1109/LCN48667.2020.9314817.

Campbell, M.J., Dennison, P.E., Butler, B.W., 2017. A lidar-based analysis of the effects of slope, vegetation density, and ground surface roughness on travel rates for wildland firefighter escape route mapping. Int. J. Wildland Fire 26, 884–895. https://doi.org/10.1071/WF17031.

Campbell, M.J., Dennison, P.E., Butler, B.W., Page, W.G., 2019. Using crowdsourced fitness tracker data to model the relationship between slope and travel rates. Appl. Geogr. 106, 93–107. https://doi.org/10.1016/j.apgeog.2019.03.008. URL: https://www.sciencedirect.com/science/article/pii/S0143622818307859.

Campbell, S., O'Mahony, N., Carvalho, A., Krpalkova, L., Riordan, D., Walsh, J., 2020. Path planning techniques for mobile robots a review. In: 2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE), pp. 12–16. https://doi.org/10.1109/ICMRE49073.2020.9065187.

Contreras, M., Chung, W., 2007. A computer approach to finding an optimal log landing location and analyzing influencing factors for ground-based timber harvesting. Can. J. For. Res. 37, 276–292. https://doi.org/10.1139/x06-219.

Cowen, D.J., Jensen, J.R., Hendrix, C., Hodgson, M., Schill, S.R., Macchiaverna, F., 2000. A gis-assisted rail construction econometric model that incorporates lidar data. Photogram. Eng. Remote Sens. 66, 1323–1328.

Davey, R.C., Hayes, M., Norman, J.M., 1994. Running uphill: An experimental result and its applications. J. Oper. Res. Soc. 45, 25–29. https://doi.org/10.2307/2583947. URL: http://www.jstor.org/stable/2583947.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numer. Math. 1, 269–271. https://doi.org/10.1007/BF01386390.

Fisher, P.F., Tate, N.J., 2006. Causes and consequences of error in digital elevation models. Prog. Phys. Geogr. Earth Environ. 30, 467–489. https://doi.org/10.1191/0309133306pp492ra.

Flisberg, P., Rönnqvist, M., Willén, E., Frisk, M., Friberg, G., 2021. Spatial optimization of ground-based primary extraction routes using the bestway decision support system. Can. J. For. Res. 51, 675–691. https://doi.org/10.1139/cjfr-2020-0238.

Floyd, R.W., 1962. Algorithm 97: Shortest path. Commun. ACM 5, 345. https://doi.org/10.1145/367766.368168.

Ford Jr, L.R., 1956. Network flow theory. Technical Report. Rand Corp Santa Monica Ca.

Glenn, N.F., Streutker, D.R., Chadwick, D.J., Thackray, G.D., Dorsch, S.J., 2006. Analysis of lidar-derived topographic information for characterizing and differentiating landslide morphology and activity. Geomorphology 73, 131–148. https://doi.org/10.1016/j.geomorph.2005.07.006. URL: https://www.sciencedirect.com/science/article/pii/S0169555X05002047.

González de Santos, L.M., Frías Nores, E., Martínez Sánchez, J., González Jorge, H., 2021. Indoor path-planning algorithm for uav-based contact inspection. Sensors 21. https://doi.org/10.3390/s21020642.

Grama, A., Kumar, V., Gupta, A., Karypis, G., 2003. Introduction to parallel computing, second ed. Pearson Education.

Hart, P.E., Nilsson, N.J., Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybernet. 4, 100–107. https://doi.org/10.1109/TSSC.1968.300136.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W., 1992. Surface reconstruction from unorganized points. SIGGRAPH Comput. Graph. 26, 71–s78. https://doi.org/10.1145/142920.134011.

Irmischer, I.J., Clarke, K.C., 2018. Measuring and modeling the speed of human navigation. Cartogr. Geogr. Inform. Sci. 45, 177–186. https://doi.org/10.1080/15230406.2017.1292150.

Jones, K.L., Poole, G.C., O'Daniel, S.J., Mertes, L.A., Stanford, J.A., 2008. Surface hydrology of low-relief landscapes: Assessing surface water flow impedance using lidar-derived digital elevation models. Remote Sens. Environ. 112, 4148–4158. https://doi.org/10.1016/j.rse.2008.01.024. URL: https://www.sciencedirect.com/science/article/pii/S0034425708002113, applications of Remote Sensing to Monitoring Freshwater and Estuarine Systems.

Klasing, K., Althoff, D., Wollherr, D., Buss, M., 2009. Comparison of surface normal estimation methods for range sensing applications. In: 2009 IEEE International Conference on Robotics and Automation, pp. 3206–3211. https://doi.org/10.1109/ROBOT.2009.5152493.

Li, J., Qin, H., Wang, J., Li, J., 2021. Openstreetmap-based autonomous navigation for the four wheel-legged robot via 3d-lidar and ccd camera. IEEE Trans. Industr. Electron. https://doi.org/10.1109/TIE.2021.3070508, 1–1.

Liu, Y., Hatayama, M., Okada, N., 2006. Development of an adaptive evacuation route algorithm under flood disaster. Annuals of Disaster Prevention Research Institute, Kyoto University 49, 189–195.

Lorenzo, O.G., Martínez, J., Vilariño, D.L., Pena, T.F., Cabaleiro, J.C., Rivera, F.F., 2017. Landing sites detection using LiDAR data on manycore systems. J. Supercomput. 73, 557–575. https://doi.org/10.1007/s11227-016-1912-7.

Lussem, U., Bolten, A., Gnyp, M.L., Jasper, J., Bareth, G., 2018. Evaluation of rgb-based vegetation indices from uav imagery to estimate forage yield in grassland. Int. Arch. Photogram. Remote Sens. Spatial Inform. Sci. XLII-3 1215–1219. https://doi.org/10.5194/isprs-archives-XLII-3-1215-2018. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-3/1215/2018/.

Martínez, J., Rivera, F.F., Cabaleiro, J.C., Vilariño, D.L., Pena, T.F., Miranda B, David, 2016. A rule-based classification from a region-growing segmentation of airborne lidar. In: Bruzzone, L., Bovolo, F. (Eds.), Image and Signal Processing for Remote Sensing XXII. International Society for Optics and Photonics. SPIE, pp. 140–150. https://doi.org/10.1117/12.2240750.

Martínez Sánchez, J., Fernández Rivera, F., Cabaleiro Domínguez, J.C., López Vilariño, D., Fernández Pena, T., 2020. Automatic extraction of road points from airborne lidar based on bidirectional skewness balancing. Remote Sens. 12 https://doi.org/10.3390/rs12122025. URL: https://www.mdpi.com/2072-4292/12/12/2025.

Metz, M., Mitasova, H., Harmon, R.S., 2011. Efficient extraction of drainage networks from massive, radar-based elevation models with least cost path search. Hydrol. Earth Syst. Sci. 15, 667–678. https://doi.org/10.5194/hess-15-667-2011. URL: https://hess.copernicus.org/articles/15/667/2011/.

Naismith, W.W., 1892. Excursions. Cruach Ardran, Stobinian, and Ben More. Scottish Mountaineering Club J. 136.

OpenMP Architecture Review Board, 2015. OpenMP application program interface version 4.5. URL: https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf.

Özcan, A.H., Ünsalan, C., 2017. Lidar data filtering and dtm generation using empirical mode decomposition. IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens. 10, 360–371. https://doi.org/10.1109/JSTARS.2016.2543464.

Parsakhoo, A., Jajouzadeh, M., 2016. Determining an optimal path for forest road construction using dijkstra's algorithm. J. For. Sci. 62, 264–268. https://doi.org/10.17221/9/2016-JFS.

Peng, F., Wong, M.S., Wan, Y., Nichol, J.E., 2017. Modeling of urban wind ventilation using high resolution airborne lidar data. Comput. Environ. Urban Syst. 64, 81–90. https://doi.org/10.1016/j.compenvurbsys.2017.01.003. URL: http://www.sciencedirect.com/science/article/pii/S0198971516301740.

Pingel, T.J., Clarke, K.C., McBride, W.A., 2013. An improved simple morphological filter for the terrain classification of airborne lidar data. ISPRS J. Photogram. Remote Sens. 77, 21–30. https://doi.org/10.1016/j.isprsjprs.2012.12.002. URL: http://www.sciencedirect.com/science/article/pii/S0924271613000026.

Rees, W., 2004. Least-cost paths in mountainous terrain. Comput. Geosci. 30, 203–209. https://doi.org/10.1016/j.cageo.2003.11.001. URL: https://www.sciencedirect.com/science/article/pii/S0098300404000226.

Ristov, S., Prodan, R., Gusev, M., Skala, K., 2016. Superlinear speedup in hpc systems: Why and when?. In: 2016 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 889–898.

Sarı, F., Sen, M., 2017. Least cost path algorithm design for highway route selection. Int. J. Eng. Geosci. 2, 1–8. https://doi.org/10.26833/ijeg.285770.

Tang, J., Pilesjö, P., Persson, A., 2013. Estimating slope from raster data - a test of eight algorithms at different resolutions in flat and steep terrain. Geodesy Cartogr. 39, 41–52. https://doi.org/10.3846/20296991.2013.806702.

Tobler, W., 1993. Three presentations on geographical analysis and modeling. National Center for Geographical Analysis and Modeling, University of California, Santa Barbara.

Verbrugghe, G., De Clercq, W., Van Eetvelde, V., 2017. Routes across the civitas menapiorum: using least cost paths and gis to locate the roman roads of sandy flanders. J. Histor. Geogr. 57, 76–88. https://doi.org/10.1016/j.jhg.2017.06.006. URL: https://www.sciencedirect.com/science/article/pii/S0305748817301123.

Wang, Z., Zlatanova, S., Moreno, A., van Oosterom, P., Toro, C., 2014. A data model for route planning in the case of forest fires. Comput. Geosci. 68, 1–10. https://doi.org/10.1016/j.cageo.2014.03.013. URL: https://www.sciencedirect.com/science/article/pii/S0098300414000636.

Zhang, C., Wang, J., Li, J., Yan, M., 2017. 2d map building and path planning based on lidar. In: 2017 4th International Conference on Information Science and Control Engineering (ICISCE), pp. 783–787. https://doi.org/10.1109/ICISCE.2017.167.

Zheng, Z., Bewley, T.R., Kuester, F., 2020. Point cloud-based target-oriented 3d path planning for uavs. In: 2020 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 790–798. https://doi.org/10.1109/ICUAS48674.2020.9213894.

Zhou, Q., Liu, X., 2004. Analysis of errors of derived slope and aspect related to dem data properties. Comput. Geosci. 30, 369–378. https://doi.org/10.1016/j.cageo.2003.07.005. URL: https://www.sciencedirect.com/science/article/pii/S0098300404000160, multidimensional geospatial technology for the geosciences.