applied sciences

MDPI

Article

Reconstructing Nerve Structures from Unorganized Points

Jelena Kljajić [1,2,*], Goran Kvaščev [1] and Željko Đurović [1]

[1] School of Electrical Engineering, University of Belgrade, 11000 Belgrade, Serbia
[2] Institute Mihajlo Pupin, University of Belgrade, 11000 Belgrade, Serbia
* Correspondence: jelena.kljajic@pupin.rs

Abstract: Realistic sensory feedback is paramount for amputees as it improves prosthetic limb control and boosts functionality, safety, and overall quality of life. This sensory restoration relies on the direct electrostimulation of residual peripheral nerves. Computational models are instrumental in simulating these neurostimulation effects, offering solutions to the complexities tied to extensive animal/human trials and costly materials. Central to these models is the detailed mapping of nerve geometry, necessitating the delineation of internal nerve structures, such as fascicles, across various cross-sections. In our modeling process, we faced the challenge of organizing an originally unstructured set of points into coherent contours. We introduced a parameter-free curve-reconstruction algorithm that combines valley-seeking clustering, an adaptive Kalman filter, and the nearest neighbor classification technique. While intuitively simple for humans, the task of reconstructing multiple open and/or closed lines with pronounced corners from a nonuniform point set is daunting for many algorithms. Additionally, the precise differentiation of adjacent curves, commonly encountered in realistic nerve models, remains a formidable challenge even for top-tier algorithms. Our proposed method adeptly navigates the complexities inherent to nerve structure reconstruction. While our algorithm is chiefly designed for closed curves, as dictated by nerve geometry, we believe it can be reconfigured with appropriate code adjustments to handle open curves. Beyond neuroprosthetics, our proposed model has the potential to be applied and spark innovations in biomedicine and a variety of other fields.

Keywords: nerve geometry; curve reconstruction; unorganized points; multiple curves; neuroprosthetics; neurostimulation

check for updates

Citation: Kljajić, J.; Kvaščev, G.; Đurović, Ž. Reconstructing Nerve Structures from Unorganized Points. *Appl. Sci.* **2023**, *13*, 11421. https://doi.org/10.3390/app132011421

Academic Editors: Bo Yang, Bo Jin and Xiaoyan Chen

Received: 3 September 2023
Revised: 11 October 2023
Accepted: 14 October 2023
Published: 18 October 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

There are a number of tasks we humans tend to solve effortlessly and almost intuitively, which machines find complicated and often require intricate algorithms for solving them. One such task is connecting seemingly randomized points into a meaningful shape. This type of 2D reconstruction is fundamental across a myriad of application areas: digital image processing, computer vision, reverse engineering, material quality control, 3D medical image reconstruction, and handwriting recognition, among others. In our case, the need for the development of such a technique for line reconstruction emerged during the in silico modeling of a human peripheral nerve.

A 2017 study showed that approximately 57.7-million people worldwide were living with limb amputation caused by a traumatic injury [1]. This study did not account for other common causes of extremity loss, such as diabetic neuropathy, cancer, infection, or burn injuries. Recent advances in the field of neurotechnology have enabled scientists to develop new methods that could help amputees [2]. Today, there are numerous prostheses available for both upper and lower limbs. However, there is significant room for improvement in restoring sensory feedback to amputees.

Nerve fibers are responsible for transmitting information between the Central Nervous System (CNS) and the peripheral effectors, and vice versa. These fibers bundle together to form nerve fascicles, which are essentially groups or bundles of nerve fibers within a

nerve. Fascicles often exhibit intricate patterns of crossings and branchings. Recent studies have highlighted that direct electrostimulation of the remaining portions of a peripheral nerve can successfully generate physiologically relevant sensations, such as touch, pressure, vibration, and tingling [3–7]. Owing to the significant variability in peripheral nerve anatomy and its unique characteristics, it is unfeasible to standardize electrode positioning and stimulation protocols. Instead of resorting to clinical animal or human trials, which are generally complicated, risky, and bring a high cost, in silico modeling offers a promising alternative [3,8–10]. The combination of the Finite Element Method (FEM) with Neuron Modeling (hybrid electro-neural modeling) could simulate the effects of the electrical stimulation of residual peripheral nerves, i.e., the nerve fibers (axons) contained within, thus facilitating electrode positioning and stimulation paradigm optimization. Detailed and realistic nerve structure modeling would undoubtedly enhance the precision of these models. An in silico method for nerve contour reconstruction would certainly be an integral step. We introduced a technique specifically designed for this purpose, but one that could, with some minor adjustments, also find its use in a handful of other disciplines.

The persistent challenge of curve (re)construction from an unorganized set of points, coupled with the need for a reliable solution, has motivated numerous scientists to address this issue. However, this is not an easy task. Its complexity largely depends on the assignment, particularly the characteristics of the point set and the desired curve shape. A nonuniform point distribution, the high proximity of contours, as well as sharp corners they could contain are just some of the most-common problems such algorithms can encounter. However, few have successfully tackled these challenges. Existing techniques vary in their performance and in how they formulate and approach these tasks. More about line types each of them is able to reconstruct, their similarities, and their differences will be discussed in Section 2.

Section 3 presents the motivation for this study and gives a detailed insight into the problem that needed to be addressed during realistic nerve contour reconstruction. Section 4 describes the innovative way in which the authors approached the problem, which combines techniques such as valley-seeking clustering, the adaptive Kalman filter, and Nearest Neighbor (NN) classification.

Section 5 presents the reconstruction results generated in Matlab and the comparison with the representative state-of-the-art techniques, as well as outlines key considerations for future implementations of this algorithm. Finally, Section 6 sums up the advantages of the proposed algorithm and puts it into the context of potential future improvements and usages.

2. Related Work

During the last several decades, numerous methods dedicated to line reconstruction from a set of unstructured points have been presented [11]. In general, these techniques can be classified based on their ability to handle lines, as well as sets of points, of different properties. In this paper, we highlight the most-prominent techniques, differentiated by their responses to the following questions:

1. Is the point distribution uniform or nonuniform?
2. Are the lines smooth, or do they contain sharp corners?
3. Can the method handle both open and closed curves?

Additionally, while some algorithms can reconstruct only a single contour at a time, others are capable of detecting multiple lines. The majority of the techniques focus only on curves without self-intersection.

The earlier techniques, such as α-shapes [12,13], β-skeletons [14], minimum spanning trees [15], and r-regular shapes [16], were limited to a uniform point distribution and could reconstruct only smooth lines, without sharp corners.

The following methods brought improvement in the aspect of working with a nonuniform set of points: Crust [17,18], Nearest Neighbor Crust (NN-Crust) [19], Conservative

Crust [20], Gathan [21], GathanG [22], Discur [23], and Vicur [24], as well as the algorithms that are based on the Traveling Salesman Problem (TSP) [25,26].

The Crust algorithm, first introduced in [17], constructs a graph on a planar point set. It is designed for the reconstruction of smooth closed curves, under the condition that the points are dense enough in areas with higher curvature. Gold [18] introduced a simplified version of the original Crust algorithm. Similarly, Dey and Kumar [19] proposed their NN-Crust method, which introduces the nearest neighbor technique. The following method Conservative Crust [20] is suitable for the reconstruction of both open and closed lines.

However, the biggest difficulty for all the above-mentioned methods is the reconstruction of sharp angles. The problem occurs when two non-adjacent points are nearest neighbors, which leads to their incorrect connection (Figure 1). Giesen was the first to offer a solution to this problem [25], developing a reconstruction algorithm that could reconstruct open or closed non-smooth curves. This method, as well as the method proposed by Althaus and Mehlhorn [26] are based on the Traveling Salesman Problem (TSP). However, due to the way the TSP algorithm works, neither one of those two methods can reconstruct more than a single curve at a time, while the latter method can handle only closed curves.

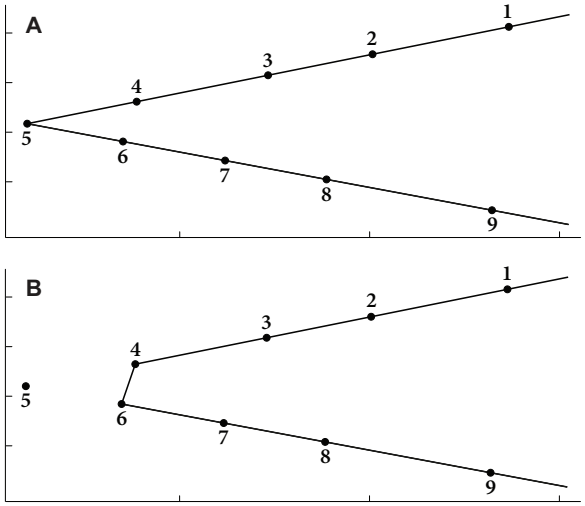


Figure 1. An example of a curve with a sharp corner. (**A**) Correct reconstruction. (**B**) A problem found in many methods: due to the proximity between the Points 4 and 6, the corner Point 5 is missed.

Dey and Wenger proposed a method called Gathan [21], which can work with curves that contain sharp corners. The same authors then presented GathanG [22], which guarantees closedcurve reconstruction, with both smooth and corner parts, under specified conditions. Funke and Ramos [27] introduced a method able to reconstruct both open and/or closed single or multiple curves with or without sharp corners. However, its limited adoption is often attributed to specific sampling conditions and the lack of open-source code availability.

The algorithms Discur [23] and Vicur [24] bring a new perspective, inspired by the way human vision works: nearness (two nearest points are usually connected) and smoothness (the connected points tend to form a smooth curve). Both techniques enable reconstruction of single or multiple open and closed curvesand/or lines with sharp corners. Discur requires a very dense sampling near the sharp corner area, while Vicur brings an advancement in that aspect.

The method HNN-Crust, introduced in [28], deals with sampling conditions and improves them. Ohrhallinger et al. showed that their algorithm requires less-dense sampling than the prior methods of the same type (the Crust family of proximity-based reconstruction algorithms). The main deficiency of this method is it not being able to successfully reconstruct contours with sharp corners or two lines in close proximity.

3. Motivation of This Study

The method described in this paper was developed while working on a framework called ProprioStim [29], which focuses on planning biomimetic stimulation strategies to restore natural proprioceptive sensation, primarily for individuals with peripheral nerve damage, such as amputees. One of the framework's key contributions is its departure from the traditional methods of generating in silico nerve models. The mentioned study demonstrated that the innovative approach of 3D modeling, which faithfully reconstructs peripheral nerves with consideration for their intricate complexity, branching, and internal nerve structures, yields significantly different results compared to the commonly used extruded models [8,9,30]. This approach substantially enhances the accuracy of the simulation.

During the development of ProprioStim, previously collected experimental data were utilized, including histological images of transverse sections of the sciatic nerves from two transfemoral amputees (as the one shown in Figure 2A) [4]. They were utilized for CAD modeling of the detailed fascicular nerve geometry (Figure 2B,C), which was then used to create an FEM model in the Comsol software (version 5.5). The spatial distribution of electrical potentials, obtained during the process, along with the nerve geometry were imported into Matlab for the purpose of calculating the recruitment of individual nerve fibers within the fascicle. Both nerves had similar cross-sectional dimensions (approximately 20 mm $\times$ 7.5 mm). The acquired histologies allowed for the modeling of nerve segments with a length of 20 mm along the longitudinal (z) axis. The nerve segments were discretized into 19 equidistant cross-sections for manipulation in Matlab, with a layer-to-layer spacing of 100 μm.

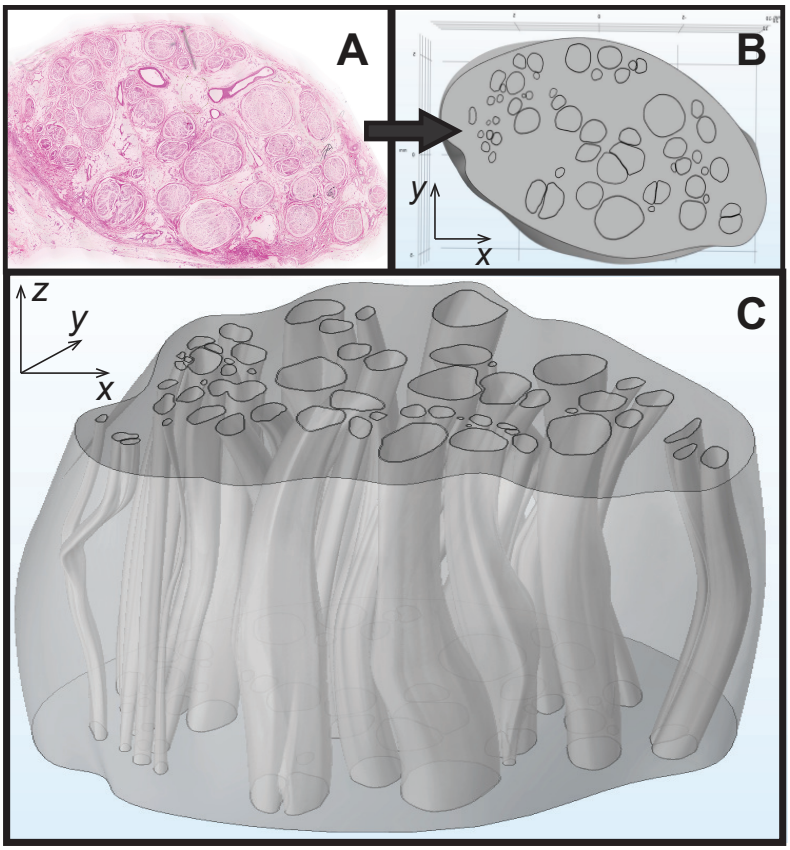


Figure 2. (**A**) Histological image captured from a layer of the sciatic nerve from a transfemoral amputee. (**B**) The associated nerve segmentation derived from this cross-section. (**C**) The reconstruction of the geometry of a peripheral human nerve, illustrating the branching of fascicles.

The motivation behind the method presented in this paper emerged as a result of the challenges encountered during the data transfer of nerve geometry from Comsol to Matlab. The vertices of the contours, which correspond to cross-sectional slices of nerve fascicles, initially became randomized upon import into Matlab, requiring a sorting process. Figure 3 depicts an example of a problem this paper focused on. The contour should be (re)constructed as precisely as possible from the nonuniformly distributed sample points (Figure 3A). Figure 3B illustrates its expected shape. If the points were connected according to the initial randomized order, the resulting contour would look as depicted in Figure 3C. Figure 3D shows an enlarged view of the reconstruction presented in Figure 3C, while four randomly chosen points are shown as references in both Figure 3C,D.

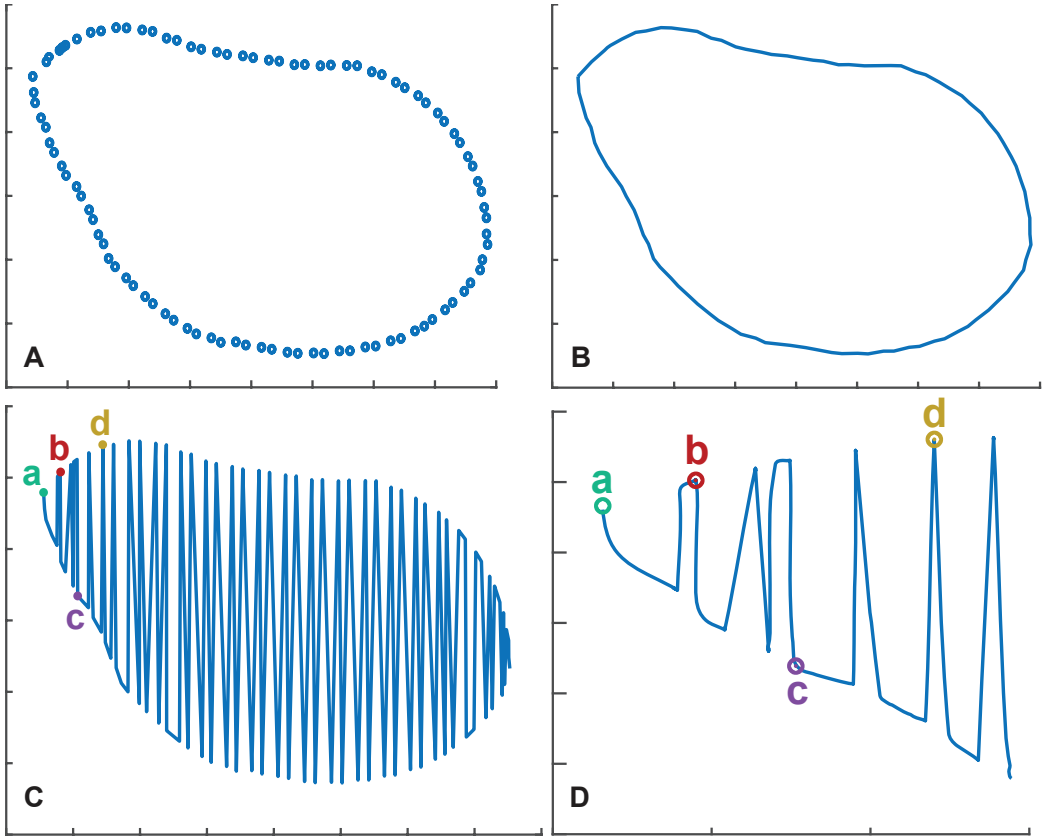


Figure 3. (**A**) Collection of unorganized points to be linked into a curve. (**B**) The anticipated curve's form. (**C**) The outcome after linking the initial, unsorted points. (**D**) A magnified view of (**C**).

4. Materials and Methods

4.1. Initializing the Algorithm and the Clusters

The algorithm works recurrently and processes each cross-sectional level of each nerve fascicle individually. It starts by loading all the points from one level. As the points are initially unsorted, the goal is to organize and group them into clusters, i.e., connect them into lines. The initial point of the first cluster can be sampled from the input set randomly. Each following cluster from the same level is initialized by the first remaining, i.e., unsorted point from the input set.

The part of the algorithm that sorts points into corresponding clusters is based on the valley-seeking clustering method. This is a nonparametric technique that estimates the gradient of a density function [31–33]. Since a higher density of this function means more samples, our method's objective is to find the regions with the highest density function and, thus, detect individual clusters.

The graph theoretic valley-seeking technique involves the following procedure: an area $G(X)$ around point X is selected (Figure 4):

$$G(X) = \{Y : d(Y, X) \leq r\}, \tag{1}$$

where r is the radius of the area $G(X)$. A measure of the distance $d(Y, X)$ is specified by:

$$d^2(Y, X) = (Y - X)^T A^{-1}(Y - X), \tag{2}$$

where A is the metric to measure the distance. If the data about the distribution statistics are insufficient, the Euclidean metric $A = I$ can be used, in which case the area $G(X)$ will take the form of a circle. This assumption is applicable to the problem we tackled (Figure 4).

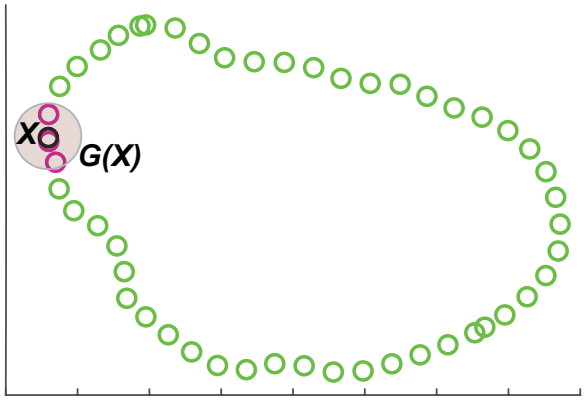


Figure 4. Graphical representation of valley-seeking clustering implemented in the algorithm. The point X (black) and the area $G(X)$ around it (shaded region); all the points located in this area (pink) are candidates for the next point Y in the cluster.

All the points that can be found inside the area $G(X)$ are taken into consideration as potential candidates for the point X's successor. The local mean $M(X)$ can be calculated as:

$$M(X) = E\{(Y - X) \mid G(X)\} = \int_{G(X)} (Y - X) \frac{f(Y)}{u_0} \mathrm{d}Y, \tag{3}$$

where $f(Y)$ is the a priori density function in the point Y and u_0 is the probability that Y belongs to the area $G(X)$ [32]. Under the assumption that the area $G(X)$ is small enough, the probability u_0 can be expressed as:

$$u_0 = \int_{G(X)} f(Y) \mathrm{d}Y \cong f(X)v, \tag{4}$$

where v would be the volume of the area $G(X)$. By implementing Equation (4) in Equation (3) and expanding $f(Y)$ from Equation (3) by a Taylor series, we can write the following:

$$M(X) \cong \int_{G(X)} (Y - X) \frac{f(X) + (Y - X)^T \bigtriangledown f(X)}{f(X)v} \mathrm{d}Y = \frac{r^2}{n+2} A \frac{\bigtriangledown f(X)}{f(X)}, \tag{5}$$

where $n = dim\{X\}$ and $\bigtriangledown f(X)$ is the gradient of the density function in the area $G(X)$. From Equation (5), we can derive the normalized gradient:

$$\frac{\bigtriangledown f(X)}{f(X)} \cong \frac{n+2}{r^2} A^{-1} M(X), \tag{6}$$

where $M(X)$ can be estimated based on the sample of all the points contained in the area $G(X)$, as their mean value.

For finding the next point of the cluster, i.e., the successor of the point X, we can define a measure S_{xy}:

$$S_{xy} = \left\| \frac{\bigtriangledown f(X)}{f(X)} \right\| \cos\theta_{xy}, \tag{7}$$

where S_{xy} is defined for each point Y from the area $G(X)$. $\left\| \frac{\bigtriangledown f(X)}{f(X)} \right\|$ is defined by Equation (6), while θ_{xy} is the angle between vectors $\|\bigtriangledown f(X)\|$ and $(Y - X)$. The point that maximizes the measure S_{xy} is chosen as X's successor. In practical terms, for all the analyzed points surrounding point X, the first portion of the expression in Equation (7), $\left\| \frac{\bigtriangledown f(X)}{f(X)} \right\|$, remains constant with each new point selection. This observation underscores that the measure S_{xy} and the rationale behind defining the successor are solely influenced by the value of $\cos\theta_{xy}$.

The proper choice of the area $G(X)$ is of great importance for the method's initialization. The key parameter that affects the size of this area is r from (1). This parameter should be based on the points' density. Based on their experience, the authors suggest that it should be estimated as:

$$r = k \times \frac{max(X_{max} - X_{min}, Y_{max} - Y_{min})}{N/2}, \tag{8}$$

where (X_{max}, X_{min}) and (Y_{max}, Y_{min}) are pairs of maximum and minimum values of all the points from the input set by the x-axis and y-axis, respectively, and N is the total number of points in the input set. The parameter k is calculated as:

$$k = \begin{cases} 4, & \text{if } \frac{N}{averDist} < 200 \\ 5, & \text{if } 200 \leq \frac{N}{averDist} < 240 \\ 6, & \text{otherwise} \end{cases} \tag{9}$$

where $averDist$ is defined as:

$$averDist = \frac{sum(minDist)}{N} \tag{10}$$

and where $minDist$ is an array, each element of which contains the distance from one point to its closest point in the set.

While the valley-seeking clustering method is highly applicable in many cases, it is not always able to successfully follow the dynamics of the trajectory. The problem it could encounter is a sudden change in the direction while forming the point order, as exemplified in Figure 5. To address this issue, we employed the classic valley-seeking method exclusively during the cluster initialization phase, specifically when determining the first two points of the cluster. This initial step defines the direction in which the cluster will be constructed, i.e., in which the next point will be searched for.

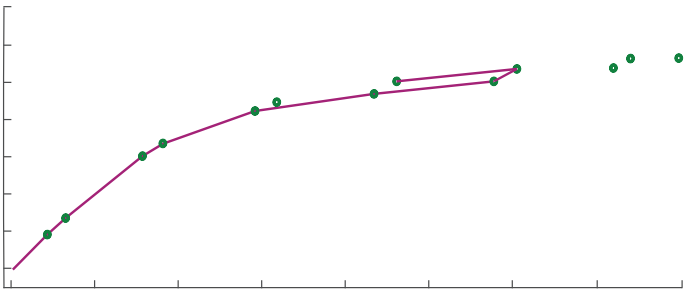


Figure 5. Unexpected and undesired shifts in the clustering direction, resulting from the use of the classic valley-seeking approach.

4.2. Trajectory Renewing

The objective of this part of the algorithm is forming an oriented graph that will, starting from the two points determined during the cluster initialization, result in a sequence of points. Once connected, the points from this sequence will form a line. The authors found that the way to accomplish this without the risk of diverging, as described in the previous section, is by combining the valley-seeking clustering method with an adaptive Kalman filter [34–39].

If we associate point-by-point contour reconstruction with target tracking, as in the example in Figure 6, the state vector $\mathbf{x}_{k+1}$ at time step $k+1$ can be expressed as:

$$\mathbf{x}_{k+1} = \begin{bmatrix} x_{k+1} & \dot{x}_{k+1} & y_{k+1} & \dot{y}_{k+1} \end{bmatrix}^T, \tag{11}$$

where x_{k+1} and y_{k+1} are the position coordinates, while $\dot{x}_{k+1}$ and $\dot{y}_{k+1}$ are the speeds along those two coordinates at time step $k+1$. The Kalman filter has the task of estimating the future states, i.e., the point positions, based on the available measurements.

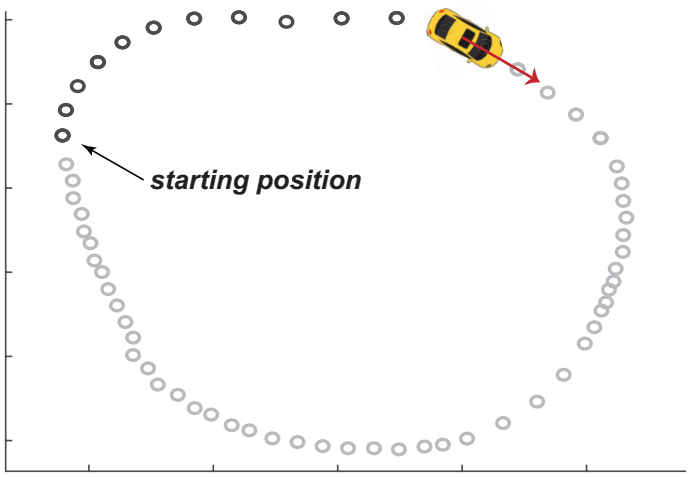


Figure 6. Representation of similarity between nerve contour reconstruction and target tracking (e.g., a car) at different time steps. Black and gray points represent the previous and future trajectory, respectively. The processed data indicate the vehicle positions sampled at equal time step. A higher point density indicates a higher speed of the vehicle.

If we adopt the constant velocity model, i.e., we assume that the velocity is constant between two consecutive sampling steps, we can write the following:

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{w}_{k+1}, \tag{12}$$

$$\mathbf{z}_{k+1} = \mathbf{H}\mathbf{x}_{k+1} + \mathbf{v}_{k+1}, \tag{13}$$

where $\mathbf{F}$ is the state transition matrix, $\mathbf{z}_{k+1}$ is the measurement at time step $k+1$, $\mathbf{H}$ is the measurement (i.e., observation) matrix, while $\mathbf{w}_{k+1}$ and $\mathbf{v}_{k+1}$ are the process noise and the measurement noise, respectively. The transition matrix is given by:

$$\mathbf{F} = \begin{bmatrix} 1 & \triangle t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \triangle t \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

where, for simplicity reasons, it was assumed that $\triangle t = 1$, and where the measurement matrix is:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{15}$$

With the nature of the particular points in mind, we can assume that the measurement noise $\mathbf{v}_{k+1}$ is negligible. On the other hand, the process noise $\mathbf{w}_{k+1}$ is very important, since its fluctuations indicate changes in trajectory, i.e., the presence of a maneuver.

Prediction can be defined as:

$$\hat{\mathbf{x}}_{k+1/k} = \mathbf{F}\hat{\mathbf{x}}_{k/k}, \tag{16}$$

$$\mathbf{P}_{k+1/k} = \mathbf{F}\mathbf{P}_{k/k}\mathbf{F}^T + \mathbf{Q}_{k+1}, \tag{17}$$

while updating can be expressed through the following set of equations:

$$\hat{\mathbf{x}}_{k+1/k+1} = \hat{\mathbf{x}}_{k+1/k} + \mathbf{K}_{k+1}(\mathbf{z}_{k+1} - \mathbf{H}\hat{\mathbf{x}}_{k+1/k}), \tag{18}$$

$$\mathbf{P}_{k+1/k+1} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}_{k+1/k}, \tag{19}$$

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1/k}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k+1/k}\mathbf{H}^T + \mathbf{R})^{-1}, \tag{20}$$

where $\hat{\mathbf{x}}_{k+1/k}$ denotes the estimate for the following time step $k+1$, $\mathbf{P}_{k+1/k}$ and $\mathbf{P}_{k+1/k+1}$ denote the covariance matrices of the state estimations $\hat{\mathbf{x}}_{k+1/k}$ and $\hat{\mathbf{x}}_{k+1/k+1}$, respectively, $\mathbf{Q}_{k+1}$ denotes the covariance matrix of the process noise, $\mathbf{R}$ denotes the covariance matrix of the measurement noise, while $\mathbf{K}_{k+1}$ is the Kalman gain. Assuming that the individual components of the process noise and the measurement noise are uncorrelated, we can write that:

$$\mathbf{R} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}, \tag{21}$$

$$\mathbf{Q}_{k+1} = \begin{bmatrix} \frac{q^3}{3} & \frac{q^2}{2} & 0 & 0 \\ \frac{q^2}{2} & q & 0 & 0 \\ 0 & 0 & \frac{q^3}{3} & \frac{q^2}{2} \\ 0 & 0 & \frac{q^2}{2} & q \end{bmatrix}, \tag{22}$$

where σ_x^2 and σ_y^2 are the variances of the x and y coordinate measurement noises, respectively, while the parameter q must be chosen in advance. The parameter q represents a deviation from the assumed constant velocity model. Since it is generally unknown, here, we adopted its value as $q = 1$, while having in mind that the adaptation described in the following paragraphs will be able to indirectly model its unfixed value.

In order to detect the deviation from a linear trajectory, it is necessary to adapt the Kalman filter. We used the adaptation approach through the fading factor S_{k+1} [36–39]. This implies changing the covariance matrix of the state estimation Equation (17):

$$\mathbf{P}_{k+1/k} = S_{k+1}\mathbf{F}\mathbf{P}_{k/k}\mathbf{F}^T + \mathbf{Q}_{k+1}. \tag{23}$$

The fading factor S_{k+1} represents the measure of model nonstationarity at time step $k+1$. It is based on the deviation of the theoretic covariance matrix ($\mathbf{M}_{k+1}$) from the covariance matrix estimated on a sample ($\mathbf{N}_{k+1}$) for the transformed prediction error sequence $\mathbf{HF}\{\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1/k}\}$:

$$S_{k+1} = max\{1, tr(\mathbf{N}_{k+1})/tr(\mathbf{M}_{k+1})\}, \tag{24}$$

and where:

$$\mathbf{M}_{k+1} = \mathbf{H}\mathbf{F}\mathbf{P}_{k/k}\mathbf{F}^T\mathbf{H}^T, \tag{25}$$

$$\mathbf{N}_{k+1} = \mathbf{P}_{V_{k+1}} - \mathbf{H}\mathbf{Q}_{k+1}\mathbf{H}^T - \mathbf{R}, \tag{26}$$

where $P_{V_{k+1}}$ is the covariance matrix of the predicted residual vector. The predicted residual vector and its covariance matrix are given by:

$$\mathbf{V}_{k+1} = \mathbf{z}_{k+1} - \mathbf{H}\mathbf{x}_{k+1/k}, \tag{27}$$

$$\mathbf{P}_{V_{k+1}} = E(V_{k+1}V_{k+1}^T) \cong \frac{1}{k}\sum_{i=1}^{k} \mathbf{V}_{k+1}\mathbf{V}_{k+1}^T. \tag{28}$$

In an ideal case, when the trajectory is linear and the distances between the consecutive points are equal, $\mathbf{M}_k$ and $\mathbf{N}_k$ share the same value. In this case, the Kalman filter works in nominal mode ($S_k = 1$). The adaptability of the Kalman filter is displayed through the situation when the measurements start to significantly differ from the predicted ones. This situation has a direct effect on the value of $\mathbf{N}_k$, which then as a result increases the value of the fading factor ($S_k > 1$) and of the covariance matrix of the state estimation. As a result, the algorithm depends less on the previous and more on the new measurements.

After each state vector estimation $\hat{\mathbf{x}}_{k+1/k}$, an area $G(X)$ is formed around each newly found point, i.e., graph node, according to Relations (1), (2), and (8). For each unassigned point from $G(X)$, a measure of accordance between the points and the trajectory is set. It is calculated in the manner of (7):

$$S_{xy} = \cos\alpha_{xy}, \tag{29}$$

where α_{xy} is the angle between the vector $(Y - X)$ and the speed vector $\dot{x}_{k+1}\mathbf{i}_x + \dot{y}_{k+1}\mathbf{i}_y$, where $\mathbf{i}_x$ and $\mathbf{i}_y$ are orthogonal unit vectors along the x- and y-axis, respectively. The point that maximizes the measure S_{xy} is chosen as the successor of the point X.

4.3. Adding Omitted Points

The previously described procedure for arranging points into a cluster implies using the angle between the vector $(Y - X)$ and the vector $\left\|\bigtriangledown f(X)\right\|$ during the initialization or the speed vector during the trajectory-renewing process. However, this leaves the possibility of missing some points that should be placed in the oriented graph between the nodes X and Y. The solution to this problem can be found in the implementation of the Nearest Neighbor (NN) algorithm after each step of the previously described procedure. In order to detect the omitted points and organize them correctly, we define a sector of a circle with its center in the point X:

$$\Xi(X) = \{Z : \left\|X - Z\right\| \leq r_1, \angle(Y - X, Z - X) < \alpha\}, \tag{30}$$

where $r_1 = \left\|X - Y\right\|$ is the radius and the angle α is one half of the sector's central angle (Figure 7). The angle α must be small enough so that the points that should belong to the other clusters are not encapsulated and added to the current cluster/graph. On the other hand, α must also be large enough to prevent any of the points that should belong to the current cluster being left outside. All the detected points should be placed between the points X and Y and sorted by ascending distance from the point X.

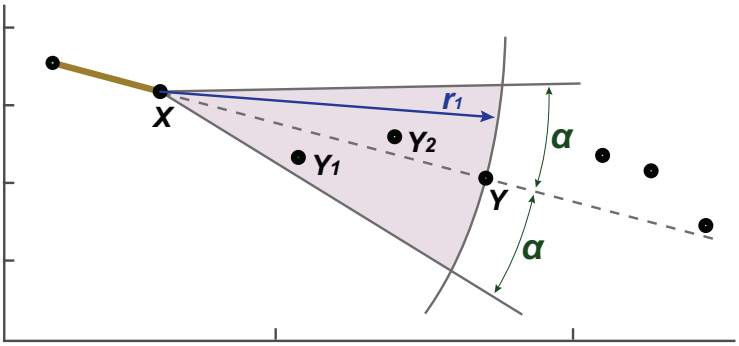


Figure 7. The NN approach used in our method. The search is performed in the sector of a circle (shaded area), where r_1 is the radius and 2α is the central angle of the sector. The sector's center is placed in the point X, while its bisection passes through the point Y. In this example, the resulting order of points in the graph would be: $\{X, Y_1, Y_2, Y\}$.

4.4. Finalizing the Individual Clusters and Forming the Final Contours

The parameter S_{xy} from the formula (29), or the formula (7), is used as a criterion for finalizing the current cluster/graph. S_{xy} with a positive value indicates that the successor Y of the point X was found. A negative value of S_{xy} suggests that the found node does not have its successor, i.e., that the algorithm has reached the root of the graph. The situation where S_{xy} is zero represents a borderline case. In terms of the probability density function, this case would correspond to a "flat" local maximum, where more than one potential maximum value can be found.

At the end of each iteration, which involves the analysis and incorporation of point(s) into the current cluster, the algorithm checks for any unsorted points remaining in the area $G(Y)$ surrounding the cluster's last point (Y). If such points are detected, the previously described procedure is repeated, focusing on the point Y (as illustrated in Figure 8, and the accompanying pseudocode, which can be found in Appendix A). If the control does not return/find any potential candidates in the area $G(Y)$, the current cluster is finalized. New clusters are formed as long as there are still some unsorted points left.

Once all the clusters are formed, the algorithm checks which ones should be connected, in order to get the final nerve contours. For that purpose, around each cluster's terminal node X and its predecessor X_p, an area in the form of a circle sector is made (as shown in Figure 9):

$$\Xi(X) = \{Z : \|X - Z\| \leq r_1, \angle(X - X_p, Z - X) < \theta\}, \tag{31}$$

where r_1 and θ are empirically derived values of the sector's radius and of one half of the sector's central angle, respectively. When some other cluster's terminal node Y belongs to $\Xi(X)$, that cluster is considered to be an extension/continuation of the current cluster.

Remarks: Two evaluations used before the final contours are formed will be reported in the following paragraphs.

The initial evaluation indirectly assesses the dimensions of the selected area $G(X)$. If, in post-cluster formation, there is a pronounced number of extremely small clusters (consisting of three points or fewer, determined empirically), as in example in Figure 10, the algorithm deduces that the chosen $G(X)$ area was undersized. As a result, the entire process restarts with an expanded radius for $G(X)$.

In the second evaluation, clusters containing fewer than n points, where n was empirically set to 1 or 2 points, are considered for elimination. These smaller clusters, often located close to larger ones, typically do not significantly influence the final reconstruction, but can pose challenges during the final contour-formation stage.

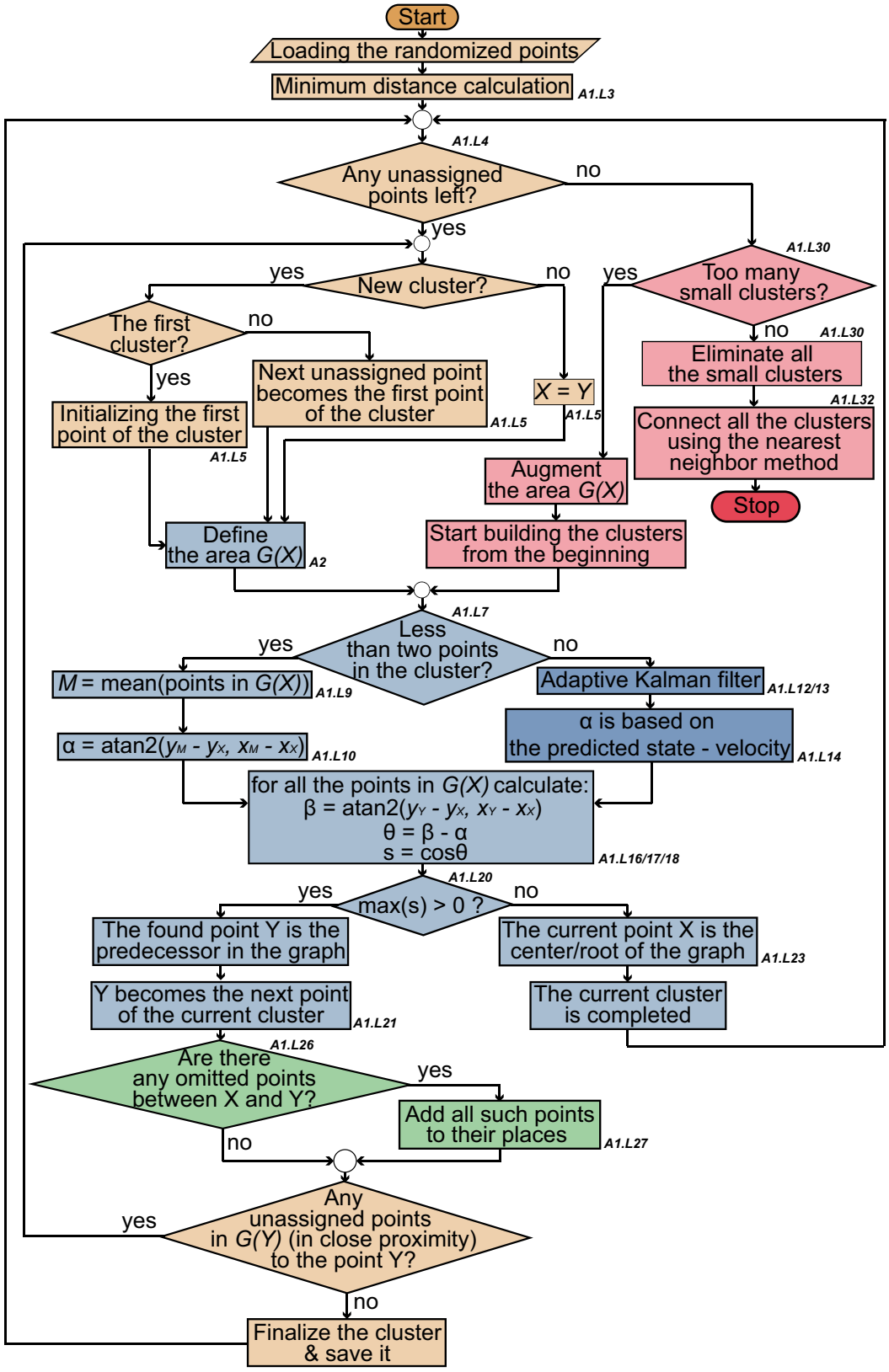


Figure 8. Flowchart with the algorithm steps. Each color represents a specific part of the algorithm: initialization of the algorithm and finalization of individual clusters (yellow), valley-seeking clustering and adaptive Kalman filter (blue), check for omitted points through NN classification (green), and steps for the formation of final contours (red). The flowchart complements the pseudocode provided in Appendix A, such that A*x*.L*y* in the flowchart relates to the *y*th line of the algorithm *x* in Appendix A.

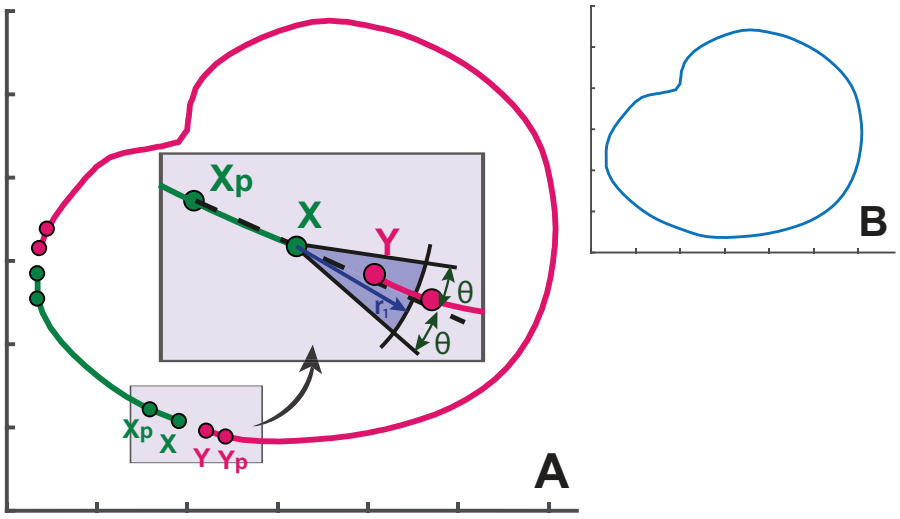


Figure 9. An example of two clusters/lines (**A**), which the algorithm recognizes as one closed contour and merges into one cluster (**B**).

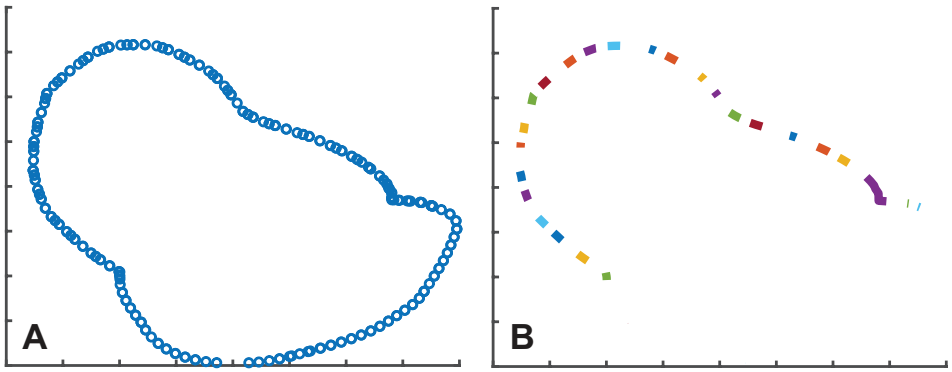


Figure 10. (**A**) Input points for one of the contours. (**B**) Inadequate area selection (G(X)) in the valley-seeking clustering step of the algorithm. When the chosen radius is too small, the algorithm struggles to create an adequate number of sufficiently large clusters, resulting in multiple clusters containing only one or two points. This leads to the "missing" segment of the contour on the right.

5. Results and Discussion

The combination of methods and procedures reported in the previous section enables the reconstruction of one (Figure 9) or several contours (Figure 11).

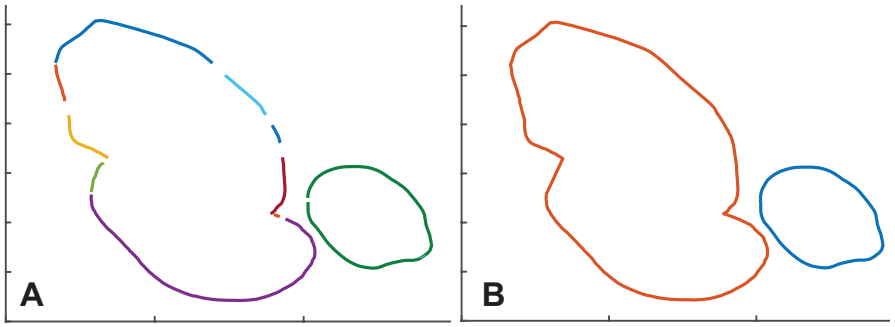


Figure 11. An example of a cross-section that belongs to a complex, branching fascicle. (**A**) Individual clusters generated by the algorithm. (**B**) The final contours result from the merging of these clusters. Notably, the algorithm achieves the successful reconstruction of sharp corners and effectively distinguishes between closely positioned contours.

A complete reconstruction of one complex (i.e., that has at least one branching along its length) nerve fascicle is shown in Figure 12. Fascicle shapes, dimensions, and branching locations differ across nerves and individuals. The example from Figure 12 shows a fascicle branching from a single section (in the lower part) into two sections (in the upper part of the figure). Indeed, nerve structures such as this one can present the biggest challenge to the algorithm, due to the complexity of nerve cross-sections at the branching points. This intricacy can lead to semi-connected contours with pronounced angles or contours in near proximity to one another, etc.

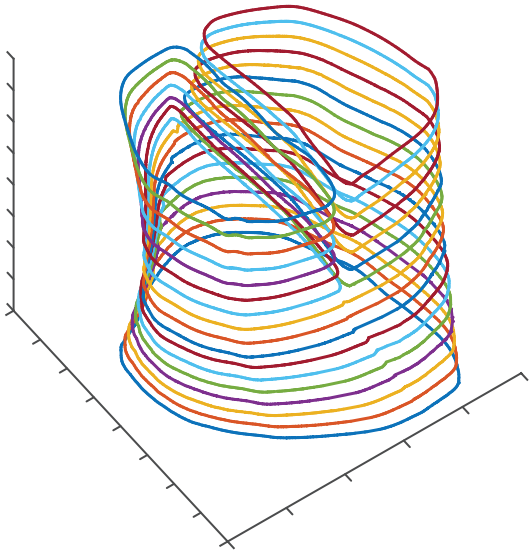


Figure 12. Nerve fascicle reconstruction by cross-sectional levels. Each color represents a reconstructed contour. Fascicle branching is visible in the upper segment of the figure.

Figure 13 displays a segment (due to the clarity of the representation) of the reconstructed fascicles from one of the two subjects.

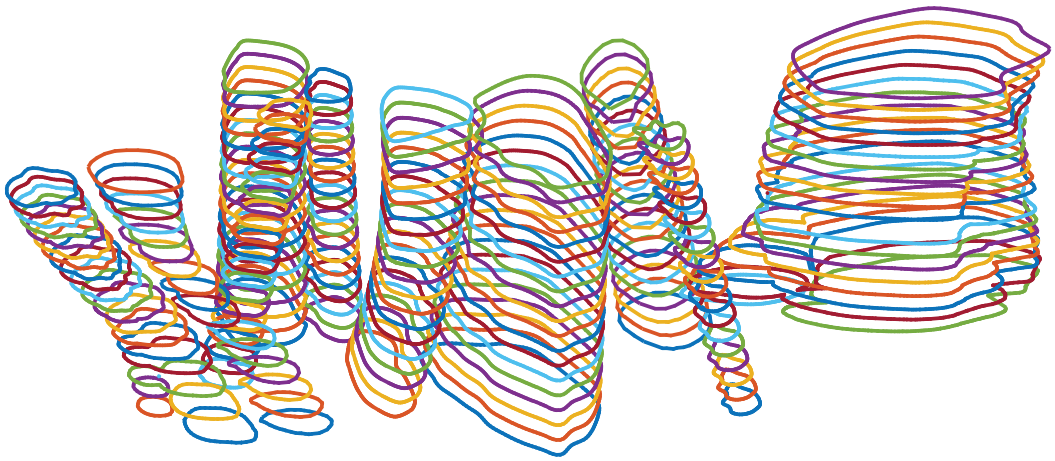


Figure 13. A section of nerve fascicles is shown for one of the subjects. Visible on the far **left** and **right** are two complex fascicles with branching.

In the context of comparing our algorithm with existing state-of-the-art methods for contour reconstruction from 2D point sets, we primarily focused on analyzing two algorithms: HNN-Crust [28] and Vicur [24]. Our selection of these two was informed by a thorough review paper [11], where they particularly distinguished themselves from other methods in an experimental study. We also selected these two algorithms because, as highlighted in Section 2, they represent cutting-edge techniques within their respective algorithm categories. HNN-Crust is the latest iteration in the Crust algorithm lineage,

introducing new enhancements to this widely used algorithm category. On the other hand, Vicur is an evolved version of the acclaimed Discur algorithm. Given that it was developed by authors previously involved with Discur, we opted to analyze this advanced iteration, Vicur.

While HNN-Crust and Vicur adeptly handle a broad range of reconstruction scenarios, they encountered challenges with specific nerve contour cases presented in this study. Particularly in situations requiring the differentiation of two closely situated contours, our newly introduced algorithm demonstrated a higher success rate than Vicur (Figure 14) and HNN-Crust (Figure 15).

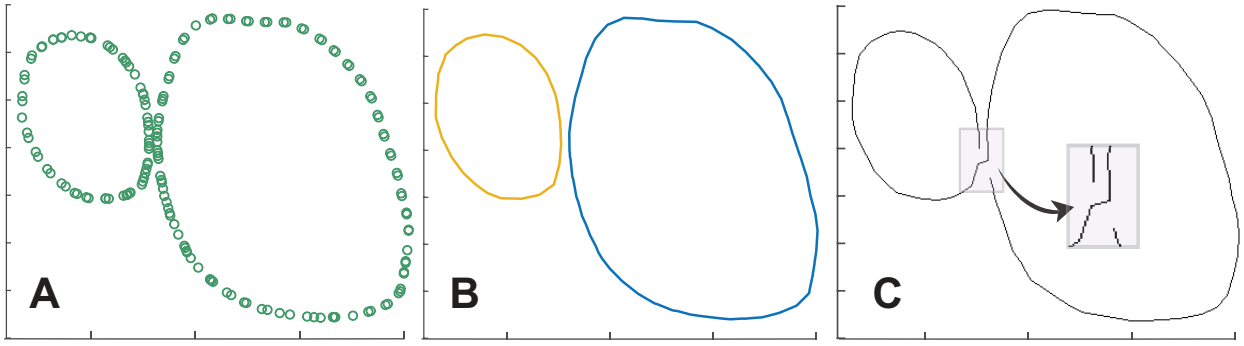


Figure 14. A comparison of the Vicur algorithm and our method. (**A**) Input points; (**B**) reconstruction performed with our approach; (**C**) reconstruction carried out using the Vicur technique. Due to the closeness of the two contours to be reconstructed and the point distribution in the segment where those contours are in close proximity, the Vicur algorithm mistakenly skipped from one contour to another and fused them together.

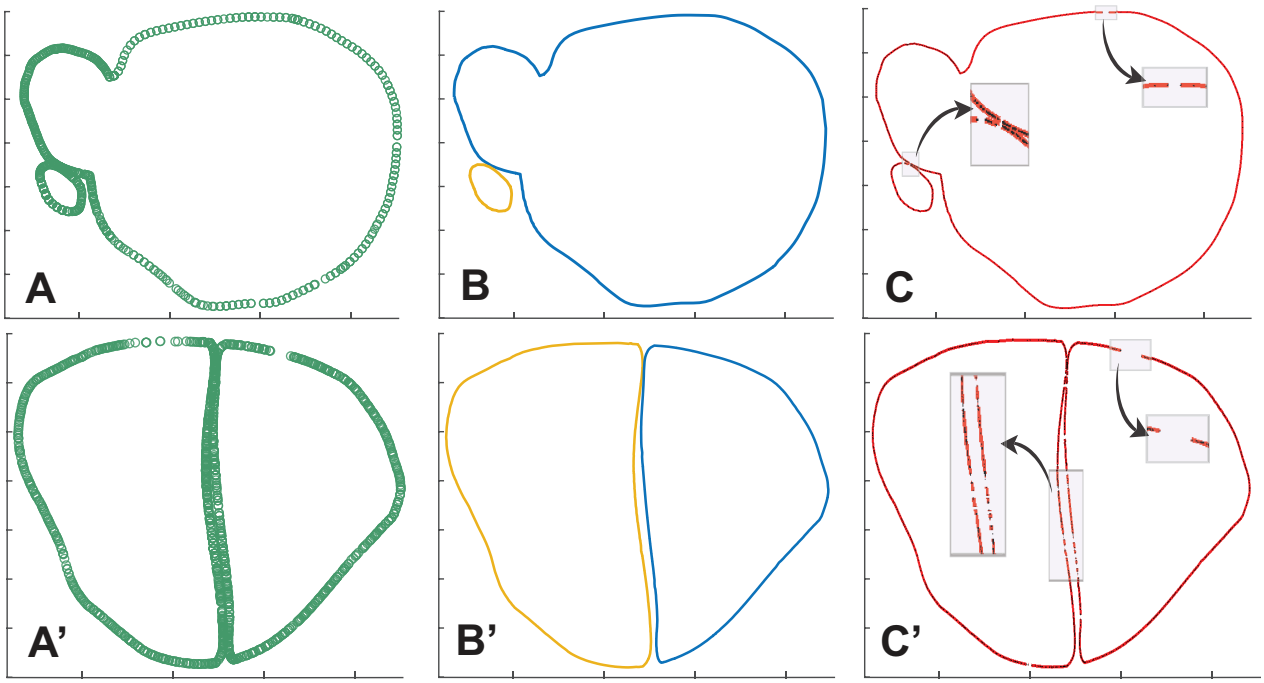


Figure 15. Two illustrative comparisons between the HNN-Crust algorithm and our approach. (**A**,**A′**) Input points; (**B**,**B′**) reconstructions accomplished using our method; (**C**,**C′**) reconstructions performed through the HNN-Crust algorithm. The reconstructions performed by HNN-Crust in (**C**) and especially in (**C′**) exhibit interruptions along the contours and an overall inability to clearly distinguish two adjacent contours.

A comparative analysis between our proposed method, HNN-Crust, and the Vicur algorithm is presented in Table 1. Given that we observed minimal variation among the three methods except in instances of fascicle branching, we opted to specifically analyze and present statistics for fascicles exhibiting branching throughout their length. The results represent an average taken from all the cross-sections of such fascicles. This indicates that, while most of the analyzed contours are straightforward, a few within each fascicle are intricate, characterized by sharp corners and notably closely spaced contours. The first column displays the Root-Mean-Squared Error (RMSE) values of each of the methods in comparison to the expert curve. The second column indicates the percentage of points that were omitted during reconstruction, relative to the total number of points in the initial set. The third column presents the count of unintended gaps or interruptions in the reconstructed curve.

Table 1. Statistical comparison between methods *.

Algorithm	RMSE	% of Omitted Points	No. of Gaps per Fascicle
our method	0.020	1.94 (%)	0
HNN-Crust	0.043	0.4 (%)	9.5
Vicur	0.019	0 (%)	0.2

* Data for branching fascicles.

Table 1 reveals that, when considering the RMSE, Vicur and our method produce closely aligned results. However, in the context of missed points, Vicur excels over both our strategy and the HNN-Crust approach, seamlessly retaining every point from the input set. This is, in fact, the main liability of our method. Conversely, in terms of faithfully reconstructing continuous curves, both HNN-Crust and Vicur demonstrate inconsistencies, especially in the scenarios illustrated in Figures 14 and 15.

Emphasizing the importance of the precise reconstruction of neighboring contours, particularly fascicle branches, Table 2 showcases statistical data from realistic nerve models derived from [29] relevant to this study. The data reveal the count of fascicles recognized in the available nerve sections, as well as the overall number of fascicle branchings occurring along their paths. Notably, about a third of all detected fascicles in the examined nerve sections exhibited at least one branching, with some presenting multiple branches. It us noteworthy that a heightened resolution in the count of cross-sectional layers amplifies the chances of encountering such scenarios, as depicted in Figures 14 and 15.

Table 2. Statistical details concerning two realistic nerve models used in this study.

Subject	Total No. of Fascicles	No. of Branchings	No. of Complex Fascicles (%)
Subject 1	49	20	≈33%
Subject 2	47	21	≈28%

While the presented algorithm effectively performs reconstructions for its intended and similar applications, it is important to acknowledge its potential limitations:

1. Given that our algorithm adopts the valley-seeking clustering method, the primary limitation of our approach lies in its inherent characteristic of omitting certain points deemed non-essential for reconstruction. Hence, it is not particularly tailored for reconstructions reliant on a small number of points. In such cases, there is an increased risk of overlooking or disregarding points that the algorithm determines as not aligning with the intended "line" (see Figure 16). However, this shortcoming diminishes with reconstructions based on a larger point set.
2. Considering the sensitivity of the results to the selection of specific parameter values (k, defined in Equation (9), α, from Equation (30), or θ, from Equation (31)), it is important to reevaluate and adjust these parameter values when dealing with a significantly

different dataset. Further details regarding the values of these parameters will be provided in the subsequent paragraphs.

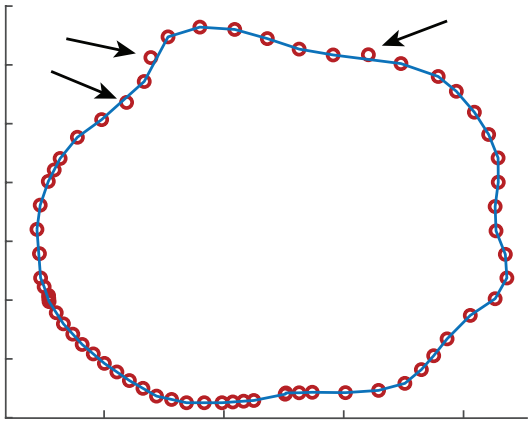


Figure 16. Certain points (marked with arrows) that the algorithm deems redundant are omitted during the reconstruction process.

The parameter k is defined in Equation (9). It is worth noting that, for future parameter selection, a value of 5 was suitable in roughly 93% of cases examined in this study. However, values of 4 and 6 reflect special cases and should be considered when setting up this method with initial values.

The parameter α, as introduced in Equation (30), plays a vital role in defining the area where omitted points will be sought. Achieving an optimal value for α must be balanced due to the reasons and logic explained in Section 4.3. In our experimentation, we explored a range of values from 2° to 90°, as these values align with the intended purpose; values below 2° result in a region that is too narrow, while values exceeding 90° yield regions that are excessively wide. It is noteworthy that, for nearly 97% of cases, any value within this range yields satisfactory reconstructions. However, in special cases, such as the one illustrated in the lower row of Figure 15, an angle between 5° and 25° proves necessary. In the context of nerve contour reconstruction, we determined that the optimal value for α is 8°.

The parameter θ, as outlined in Equation (31), plays a pivotal role in consolidating clusters to formulate the final contour. When deliberating on the suitable value for this parameter, there was no predefined rationale regarding its permissible range. Unlike α, which intuitively had a "forward" direction, θ possessed greater flexibility in its potential values. Consequently, we conducted experiments involving both the lower and upper limits of the angles that permit successful reconstruction. We systematically recorded the values that resulted in successful reconstructions, and these findings are summarized in Table 3, which presents statistical information regarding the range, i.e., the minimum and maximum angle values. As a practical rule of thumb, we identified the optimal value for θ to be 45°.

Table 3. Statistical analysis of minimum and maximum angle ranges for successful reconstruction with parameter θ.

	Minimum Angles for Successful Reconstruction	**Maximum Angles for Successful Reconstruction**
Mean value	17.34°	252.5°
Standard deviation	13.55°	115.97°
Minimum	5°	110°
Maximum	45°	360°

6. Conclusions

In this paper, we present an algorithm designed to reconstruct realistic nerve cross-sectional contours from unsorted points. Our approach addresses the complex geometry of human nerves' internal structures, characterized by frequent fascicle crossings and branching, presenting the algorithm with challenges such as handling sharp corners and closely positioned individual contours that require precise differentiation.

This innovative method has significant potential contributions to the field of neuroprostheses, particularly in simulating natural sensations in amputated limbs. We recognize the ongoing advancements needed in this area. Additionally, this study built upon our previous work in modeling nerve geometry, as detailed in [29].

Our newly developed method offers several key advantages when compared to existing approaches for curve reconstruction from unarranged points:

- Handling nonuniform point sets: Our method excels in working with nonuniform sets of points and is capable of reconstructing one or more contours, even when they are not necessarily smooth.
- Reconstructing closely positioned contours: An essential advantage of our algorithm is its proficiency in reconstructing closely positioned contours, a task where many existing methods face challenges.
- Automation: Once the parameters are set for the input values and remain relatively consistent, our algorithm operates without the need for further manual input from the user.
- Dynamics and fascicle model: Unlike many existing methods that primarily focus on geometric considerations, our algorithm stands out by incorporating dynamics and implicitly integrating a fascicle model through an adaptive Kalman filter.
- Innovative combination: To our knowledge, there is currently no algorithm for curve reconstruction from randomized points that combines valley-seeking clustering with a Kalman filter and NN classification method.

A comparative analysis with state-of-the-art methods demonstrates that our approach excels in cases involving intricate internal nerve structures, making it particularly suitable for such scenarios. While it may not be ideal for cases with a small number of input points, it outperforms other methods in situations similar to those presented here.

In this paper, we primarily focused on reconstructing closed contours due to the nature of the contours examined. However, our method has the potential to reconstruct open lines with minor algorithm modifications. In future work, we plan on expanding the algorithm's capabilities to handle open curves and, potentially, make the parameters more robust to the input values.

Beyond its primary purpose, this innovative method has broader potential applications and can serve as inspiration or a foundational technique in various fields. These encompass 2D and 3D image processing, target tracking (see Figure 6), and spatial analysis of linearly arranged points across diverse domains, including biomedicine. Its impact extends not only to the field of simulation planning in neuroprosthetics, but also to virtual surgeries and many other specialized domains.

Author Contributions: Conceptualization, J.K. and Ž.Đ.; methodology, J.K. and Ž.Đ.; software, J.K.; validation, J.K., G.K. and Ž.Đ.; formal analysis, J.K.; investigation, J.K.; data curation, J.K.; writing—original draft preparation, J.K.; writing—review and editing, J.K., G.K. and Ž.Đ.; visualization, J.K.; supervision, Ž.Đ. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are available from the corresponding author upon request.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The approach in this pseudocode provides a deeper dive into the procedure introduced in Section 4.

Algorithm A1 Algorithm for reconstruction from unorganized points.

```
function RECONSTRUCTION(unorganizedPoints)
    Initialize necessary variables and data structures.
    Compute minimum distances between all points in the input dataset (variable minDist referenced in Equation (10).
    while not all points are sorted do
        Determine a start for a new cluster.
        while current cluster is incomplete do
            if cluster is starting then
                Estimate average direction of nearby points using classic valley-seeking approach:
                Calculate mean position M of candidate surrounding points.
                Calculate angle α based on direction from current point to M.
            else
                Predict next direction using adaptive Kalman filter:
                Use Kalman filter to predict direction of cluster growth.
                Calculate angle α based on estimated velocity components from Kalman filter.
            end if
            for each candidate point in surrounding do
                Calculate its angle β with respect to the current point.
                Determine the similarity score S between α and β.
            end for
            if a clear maximum value in S exists then
                Choose the point corresponding to max S value as the next point.
            else
                Handle special cases: current point as the root of the graph or multiple potential next points.
            end if
            Add the chosen point to the current graph and cluster.
            if omitted points exist between current and next point then
                Include them in the cluster (as in Section 4.3).
            end if
        end while
        Two evaluations described in Section 4.4 and in Algorithm A4 of this Appendix A.
    end while
    Merge clusters that belong to the same curves.
    return clusters and graph.
end function
```

Algorithm A2 Define area *G*.

```
function DEFINE_AREA_G(unorganizedPointsData, minDistances, currentClusterPoint)
    Determine minimum and maximum x and y coordinates of the points
    Compute the average of minimum distances between the points
    Define a radius r, as in Equations (8)–(10)
    Create a logical mesh grid representing the area
    Create a logical circle in the image with center at the current point and radius r
    Determine all points that lie within the circle
    Return indices of these points
end function
```

Algorithm A3 Adaptive Kalman filter.

```
function KALMAN_FILTER_ADAPTIVE(unorganizedPointsData, currentClusterPoint, startingClusterPoint, previousStatePrediction, previousUncertaintyPrediction, residualHistory)
    Define system parameters and matrices (according to Equations (14), (15), (21), and (22))
    if Only two initial cluster points are chosen then
        Extract starting and current position
        Set initial state prediction based on positions
        Define initial state uncertainty
        Predict next state and its uncertainty
        Initialize history of residuals
        return Predicted state, its uncertainty, and residuals history
    else
        Get current measurement from data
        Update state estimation using previous state and measurement
        Predict next state
        Update history of measurement residuals
        Adjust prediction based on historical residuals (adaptive factor)
        Predict uncertainty of the next state using the adaptive factor
        return Predicted state, its uncertainty, and updated residuals history
    end if
end function
```

Algorithm A4 Evaluations from Section 4.4.

```
function CHECKUNUSEDPOINTS(definedClusters)
    Initialize the counter of small clusters (three points or less)
    if over more than half of the clusters are small then
        return True                          ▷ Recommend repeating with a larger area
    else
        Eliminate clusters consisting of 1 or 2 points.
        return False
    end if
end function
```

Algorithm A5 Forming the final contours, as shown in Figure 9.

```
function FORMFINALCONTOUR(definedClusters)
    Sort clusters by size
    if there is only one cluster then
        Close the cluster, in order to get a closed contour.
    else
        Set starting point at end of largest cluster
        while unsorted clusters left do
            Define search criteria as elaborated in Section 4.4 in Equation (31)
            Identify next connecting cluster
            if connection found then
                Mark the added cluster as sorted
                Append its points to current contour
                Update starting point
            end if
            Close the current contour
        end while
    end if
    return contours
end function
```

References

1. McDonald, C.L.; Westcott-McCoy, S.; Weaver, M.R.; Haagsma, J.; Kartin, D. Global prevalence of traumatic non-fatal limb amputation. *Prosthetics Orthot. Int.* **2021**, *45*, 105–115. [CrossRef] [PubMed]
2. Raspopovic, S. Advancing limb neural prostheses. *Science* **2020**. *370*, 290–291. [CrossRef]
3. Raspopovic, S.; Valle, G.; Petrini, F.M. Sensory feedback for limb prostheses in amputees. *Nat. Mater.* **2021**, *20*, 925–939. [CrossRef] [PubMed]
4. Petrini, F.M.; Bumbasirevic, M.; Valle, G.; Ilic, V.; Mijović, P.; Čvančara, P.; Barberi, F.; Katic, N.; Bortolotti, D.; Andreu, D.; et al. Sensory feedback restoration in leg amputees improves walking speed, metabolic cost and phantom pain. *Nat. Med.* **2019**, *25*, 1356–1363. [CrossRef]
5. Valle, G.; Saliji, A.; Fogle, E.; Cimolato, A.; Petrini, F.M.; Raspopovic, S. Mechanisms of neuro-robotic prosthesis operation in leg amputees. *Sci. Adv.* **2021**, *7*, eabd8354. [CrossRef] [PubMed]
6. Preatoni, G.; Valle, G.; Petrini, F.M.; Raspopovic, S. Lightening the perceived prosthesis weight with neural embodiment promoted by sensory feedback. *Curr. Biol.* **2021**, *31*, 1065–1071. [CrossRef] [PubMed]
7. Charkhkar, H.; Shell, C.E.; Marasco, P.D.; Pinault, G.J.; Tyler, D.J.; Triolo, R.J. High-density peripheral nerve cuffs restore natural sensation to individuals with lower-limb amputations. *J. Neural Eng.* **2018**, *15*, 056002. [CrossRef]
8. Raspopovic, S.; Petrini, F.M.; Zelechowski, M.; Valle, G. Framework for the development of neuroprostheses: From basic understanding by sciatic and median nerves models to bionic legs and hands. *Proc. IEEE* **2016**, *105*, 34–49. [CrossRef]
9. Zelechowski, M.; Valle, G.; Raspopovic, S. A computational model to design neural interfaces for lower-limb sensory neuroprostheses. *J. Neuroeng. Rehabil.* **2020**, *17*, 1–13. [CrossRef]
10. Romeni, S.; Valle, G.; Mazzoni, A.; Micera, S. Tutorial: A computational framework for the design and optimization of peripheral neural interfaces. *Nat. Protoc.* **2020**, *15*, 3129–3153. [CrossRef]
11. Ohrhallinger, S.; Peethambaran, J.; Parakkat, A.D.; Dey, T.K.; Muthuganapathy, R. 2d points curve reconstruction survey and benchmark. In *Computer Graphics Forum*; Wiley Online Library: Hoboken, NJ, USA, 2021; Volume 40, pp. 611–632.
12. Bernardini, F.; Bajaj, C.L. Sampling and Reconstructing Manifolds Using Alpha-Shapes. 1997. Available online: https://www.researchgate.net/publication/2258823_Sampling_and_Reconstructing_Manifolds_Using_Alpha-Shapes (accessed on 16 October 2023).
13. Edelsbrunner, H.; Kirkpatrick, D.; Seidel, R. On the shape of a set of points in the plane. *IEEE Trans. Inf. Theory* **1983**, *29*, 551–559. [CrossRef]
14. Kirkpatrick, D.G.; Radke, J.D. A framework for computational morphology. In *Machine Intelligence and Pattern Recognition*; Elsevier: Amsterdam, The Netherlands, 1985; Volume 2, pp. 217–248.
15. De Figueiredo, L.H.; de Miranda Gomes, J. Computational morphology of curves. *Vis. Comput.* **1994**, *11*, 105–112. [CrossRef]
16. Attali, D. r-Regular shape reconstruction from unorganized points. In Proceedings of the Thirteenth Annual Symposium on Computational Geometry, Nice, France, 4–6 June 1997; pp. 248–253.
17. Amenta, N.; Bern, M.; Eppstein, D. The crust and the β-skeleton: Combinatorial curve reconstruction. *Graph. Model. Image Process.* **1998**, *60*, 125–135. [CrossRef]
18. Gold, C. Crust and anti-crust: A one-step boundary and skeleton extraction algorithm. In Proceedings of the Fifteenth Annual Symposium on Computational Geometry, Miami Beach, FL, USA, 13–16 June 1999; pp. 189–196.
19. Dey, T.K.; Kumar, P. A Simple Provable Algorithm for Curve Reconstruction. In Proceedings of the SODA, Baltimore, MD, USA, 17–19 January 1999; Volume 99, pp. 893–894.
20. Dey, T.K.; Mehlhorn, K.; Ramos, E.A. Curve reconstruction: Connecting dots with good reason. In Proceedings of the Fifteenth Annual Symposium on Computational Geometry, Miami Beach, FL, USA, 13–16 June 1999; pp. 197–206.
21. Dey, T.K.; Wenger, R. Reconstruction curves with sharp corners. In Proceedings of the Sixteenth Annual Symposium on Computational Geometry, Hong Kong, China, 12–14 June 2000; pp. 233–241.
22. Dey, T.K.; Wenger, R. Fast reconstruction of curves with sharp corners. *Int. J. Comput. Geom. Appl.* **2002**, *12*, 353–400. [CrossRef]
23. Zeng, Y.; Nguyen, T.A.; Yan, B.; Li, S. A distance-based parameter free algorithm for curve reconstruction. *Comput.-Aided Des.* **2008**, *40*, 210–222. [CrossRef]
24. Nguyen, T.A.; Zeng, Y. Vicur: A human-vision-based algorithm for curve reconstruction. *Robot. Comput.-Integr. Manuf.* **2008**, *24*, 824–834. [CrossRef]
25. Giesen, J. Curve reconstruction, the traveling salesman problem and menger's theorem on length. In Proceedings of the Fifteenth annual Symposium on Computational Geometry, Miami Beach, FL, USA, 13–16 June 1999; pp. 207–216.
26. Althaus, E.; Mehlhorn, K. Traveling salesman-based curve reconstruction in polynomial time. *SIAM J. Comput.* **2001**, *31*, 27–66. [CrossRef]
27. Funke, S.; Ramos, E.A. Reconstructing a collection of curves with corners and endpoints. In Proceedings of the SODA, Washington, DC, USA, 7–9 January 2001; pp. 344–353.
28. Ohrhallinger, S.; Mitchell, S.A.; Wimmer, M. Curve reconstruction with many fewer samples. In *Computer Graphics Forum*; Wiley Online Library: Hoboken, NJ, USA, 2016; Volume 35, pp. 167–176.
29. Cimolato, A.; Ciotti, F.; Kljajić, J.; Valle, G.; Raspopovic, S. Symbiotic electroneural and musculoskeletal framework to encode proprioception via neurostimulation: ProprioStim. *Iscience* **2023**, *26*, 106248. [CrossRef] [PubMed]

30. Raspopovic, S.; Capogrosso, M.; Micera, S. A computational model for the stimulation of rat sciatic nerve using a transverse intrafascicular multichannel electrode. *IEEE Trans. Neural Syst. Rehabil. Eng.* **2011**, *19*, 333–344. [CrossRef]
31. Koontz, W.L.; Fukunaga, K. A nonparametric valley-seeking technique for cluster analysis. *IEEE Trans. Comput.* **1972**, *100*, 171–178. [CrossRef]
32. Fukunaga, K. *Introduction to Statistical Pattern Recognition*; Elsevier: Amsterdam, The Netherlands, 2013.
33. Gokcay, E.; Principe, J.C. Information theoretic clustering. *IEEE Trans. Pattern Anal. Mach. Intell.* **2002**, *24*, 158–171. [CrossRef]
34. Kalman, R.E. A New Approach to Linear Filtering and Prediction Problems. 1960. Available online: https://asmedigitalcollection.asme.org/fluidsengineering/article-abstract/82/1/35/397706/A-New-Approach-to-Linear-Filtering-and-Prediction (accessed on 16 October 2023).
35. Zarchan, P. *Progress in Astronautics and Aeronautics: Fundamentals of Kalman Filtering: A Practical Approach*; Aiaa: Reston, VA, USA, 2005; Volume 208.
36. Jiang, C.; Zhang, S.B.; Zhang, Q.Z. Adaptive estimation of multiple fading factors for GPS/INS integrated navigation systems. *Sensors* **2017**, *17*, 1254. [CrossRef] [PubMed]
37. Hu, C.; Chen, W.; Chen, Y.; Liu, D. Adaptive Kalman filtering for vehicle navigation. *J. Glob. Position. Syst.* **2003**, *2*, 42–47. [CrossRef]
38. Xia, Q.; Rao, M.; Ying, Y.; Shen, X. Adaptive fading Kalman filter with an application. *Automatica* **1994**, *30*, 1333–1338. [CrossRef]
39. Lee, T.S. Theory and application of adaptive fading memory Kalman filters. *IEEE Trans. Circuits Syst.* **1988**, *35*, 474–477. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.