IMAGE PROCESSING ON LINE

Published in Image Processing On Line on 2019–01–19.
Submitted on 2018–09–03, accepted on 2018–12–13.
ISSN 2105–1232 © 2019 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2019.234

TriplClust: An Algorithm for Curve Detection in 3D Point Clouds

Christoph Dalitz, Jens Wilberg, Lukas Aymans

Institute for Pattern Recognition (iPattern)
Niederrhein University of Applied Sciences, Krefeld, Germany
(christoph.dalitz@hs-niederrhein.de)

Abstract

In this article, we describe an algorithm for detecting and separating curves in 3D point clouds without making a priori assumptions about their parametric shape. The algorithm is called "TriplClust" because it is based on the idea of clustering point triplets instead of the original points. We define a distance measure on point triplets and then apply a single link hierarchical clustering on the triplets. The clustering process can be controlled by several parameters, which are described in detail, and suggestions for reasonable choices for these parameters based on the input data are made. Moreover, we suggest a simple criterion for stopping the single link clustering automatically.

Source Code

The reviewed C++ source code for this algorithm is available from the web page of this article[1]. Compilation and usage instructions are included in the README.txt file of the archive.

Supplementary Material

Six reference data sets are provided with the article. These stem from different application areas: particle tracks in an active target time projection chamber (AT-TPC), ball trajectories optically recorded from a tennis match, an airborne LIDAR scan of a high-voltage pole, and thresholded radar data of vehicle tracks. Additionally, there are two synthetically generated point clouds containing parabolic curves with and without random noise.

Keywords: object recognition; 3D point clouds; hierarchical clustering

[1] https://doi.org/10.5201/ipol.2019.234

Christoph Dalitz, Jens Wilberg, Lukas Aymans, *TriplClust: An Algorithm for Curve Detection in 3D Point Clouds*, Image Processing On Line, 8 (2019), pp. 26–46. https://doi.org/10.5201/ipol.2019.234

1 Introduction

Curve detection in 3D point clouds is a problem that occurs in many applications and for a wide variety of sensor data. Typical use cases are the identification of particle tracks in Time Projection Chambers [3], roadside detection in ground based mobile LIDAR data [17], roof ridge detection in airborne LIDAR data [5], or object tracking in sports events [13]. Usually, a curve detection algorithm pursues three aims:

a) Discrimination between points representing noise and those belonging to curves.

b) Partitioning of the curve points into sets representing different curves.

c) Description of the curves in parametric form, e.g. as line segments or splines.

Parametric approaches to curve detection in noisy point clouds start out from step c): they use a particular parametric description of the curve shape (in the simplest case, e.g. straight lines) and search for parameter values representing shapes with many points. This requires the shape of the curves to be known a priori, and it leads to a voting scheme for parameter values, which can be either done exhaustively for all points (Hough transform [4]) or by random sampling (RANSAC [6, 1]).

In the present article, we follow a *non-parametric* approach, which means that the shape of the curves is not assumed to be known beforehand. We reformulate the problem as a clustering problem and only address points a) and b) of the aims listed above. The problem is thus to partition the set of points into an unknown number of possibly overlapping clusters representing curves, and an additional distinct cluster representing noise. An example for the expected result can be seen in Figure 1. Point c) is not addressed in the present article and can be added as a post-processing step, if needed, for example with the method described in [7].

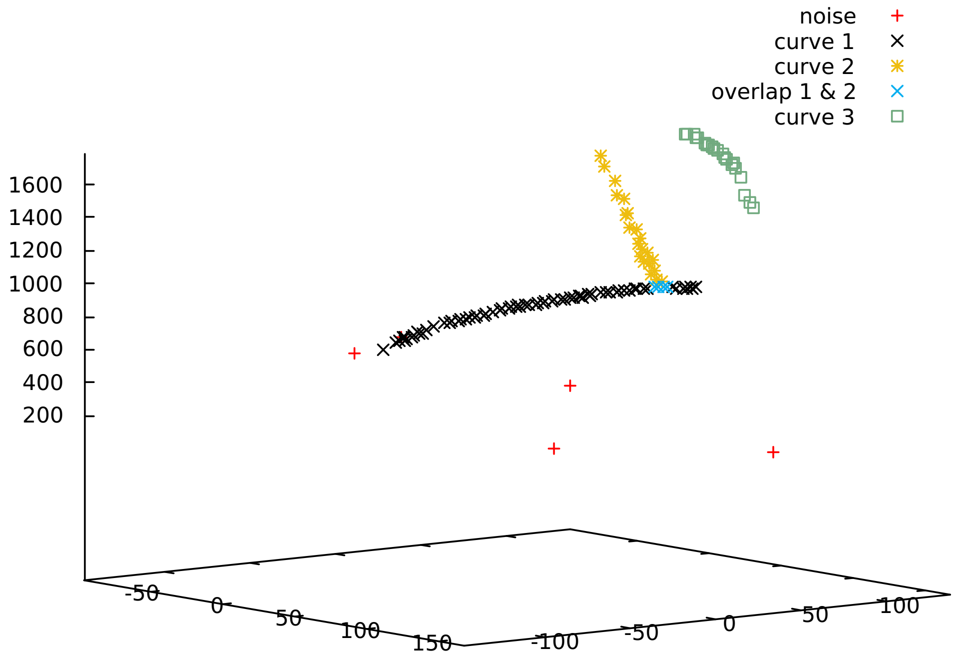


Figure 1: Ideal clustering result for a point cloud with one separated curve and two intersecting curves: the noise cluster and curve 3 are distinct from the other clusters, while the two clusters representing curves 1 & 2 overlap at the vertex.

Our algorithm is based on first building point triplets, which are then partitioned into clusters, such that each cluster represents a track. The idea of grouping triplets was recently suggested by Lezama et al. for detecting continuations in 2D dot patterns [8]. Their algorithm looked for approximately symmetric triplets, and the grouping was based on overlapping points. Whether a triplet was considered "approximately symmetric" was based on a threshold e_{max} for an error that measures the deviation from symmetry. Although that algorithm could be generalized to 3D in a straightforward way, it is applicable only in the special case of locally approximately equidistant points without much random perpendicular spread around the actual curve. As these assumptions do not hold in many real-world 3D data, we change both steps of the algorithm: firstly, we do not look for symmetric triplets, but for triplets with approximately collinear branches, and, secondly, we group the triplets with a single link hierarchical clustering utilizing an appropriately defined triplet distance measure.

Originally, we had developed the algorithm for particle track detection in active target time projection chambers (AT-TPC) [3]. Now we present a reference implementation of the algorithm in standard C++ that does not require any third party libraries. Additionally, we explain in detail the meaning of the parameters for controlling the algorithm and suggest default values for the parameters that can be automatically computed from properties of the input point cloud. In our previous study [3], the parameters were instead optimized on ground truth data. Moreover, we introduce another threshold parameter d_{max} for the maximum gap width within a curve, and we present a new method for automatically stopping the single link clustering process.

This article is organized as follows: Section 2 describes the algorithm in detail, Section 3 describes the meaning of the parameters and suggests reasonable defaults for their values, Section 4 estimates the computational complexity of the algorithm, Section 6 presents some problematic cases for the algorithm and how these can be improved by parameter tweaking, Section 7 describes the online demo provided on the web page of this article[2], and Section 8 describes the test data set provided on the same web page and the results of the algorithm on these data sets.

2 Algorithm

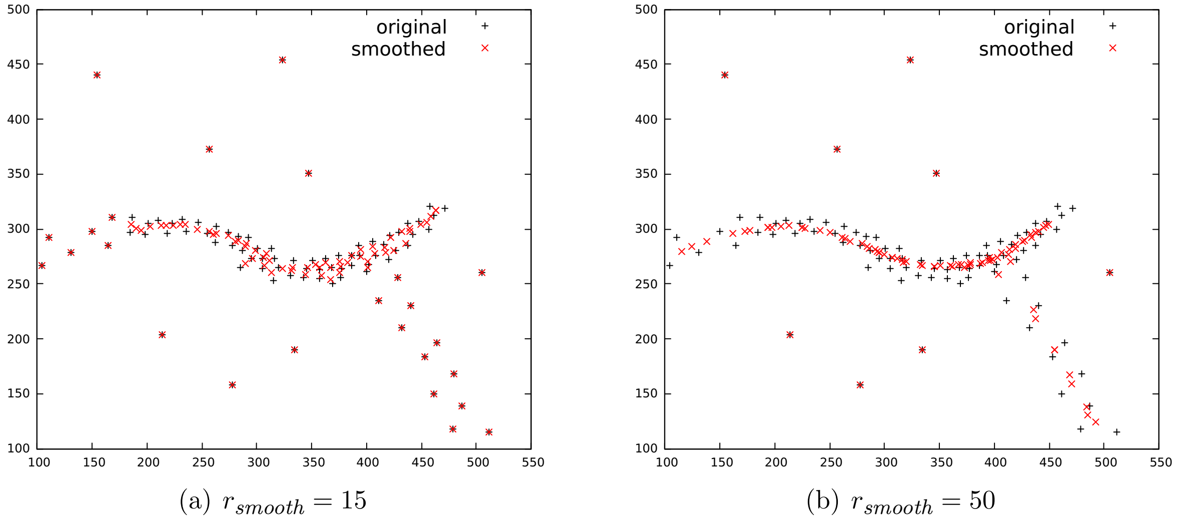
The TriplClust algorithm groups the points into clusters representing curves or noise, where clusters representing curves can overlap at vertex points. It consists of the following four steps:

1. Smoothing by position averaging of neighboring points.
2. Finding triplets of approximately collinear points.
3. Single link hierarchical clustering of the triplets.
4. Pruning by removal of small clusters and (optionally) by splitting clusters with large gaps.

The triplet clustering is then transformed into a point clustering by labeling each point with the cluster label of the triplet to which it has been assigned in step 2). Points not belonging to any triplet or only assigned to triplets that do not belong to a cluster are labeled as noise. As one point can belong to more than one triplet, the resulting clusters can overlap and some points can obtain simultaneous cluster labels. It should be noted that, although steps 2)-4) are done on the smoothed data points, the point indices are not lost during smoothing, so that the final clustering actually is a grouping of the *original* points.

The individual steps and the meaning of the parameters are described in detail in the following subsections. Each step can be controlled by external parameters which are discussed in Section 3.

[2] https://doi.org/10.5201/ipol.2019.234



(a) $r_{smooth} = 15$

(b) $r_{smooth} = 50$

Figure 2: Effect of the position smoothing with two different radii r_{smooth}. In this example, all z-components are zero so that distances are not distorted by perspective projection.

2.1 Position Smoothing

A spread of points perpendicular to the curve has the effect that triplets of adjacent points can have a branch angle considerably different from 180°, which would result in many missed triplets that actually join points on the same curve. It is thus desirable to reduce the spread of the points around the curves. To this end, we perform a smoothing operation.

We simply replace the coordinates of each point $\vec{p}$ by the mean $\sum_{i=1}^{k} \vec{q}_i/k$ of the points $\vec{q}_1, \ldots, \vec{q}_k$ in its neighborhood. A point $\vec{q}_i$ is considered to belong to the neighborhood of $\vec{p}$, if its distance $\|\vec{q}_i - \vec{p}\|$ is less than a threshold r_{smooth}. In regions with a low point density, solely the point $\vec{p}$ itself will fall into its own neighborhood, which means that the position of $\vec{p}$ remains unchanged. In regions with a high point density, several points will fall into the neighborhood and the position of points close to curves is moved towards the middle axis of the curve.

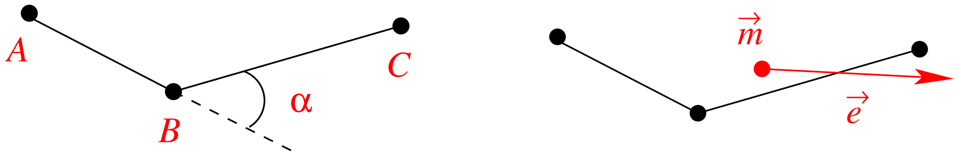
Figure 2 shows the effect of this smoothing operation with two different radii r_{smooth}. For the smaller radius in Figure 2(a), points in curve regions with a low point density remain unaltered (see the left part of the upper curve and the curve at the lower right), while points in curve regions with a high point density are moved towards the middle axis of the curve. In Figure 2(b), the radius is chosen so high that all curve points are moved towards the respective middle axis. This has the side effects, however, that curves are made shorter at the end points (see the right end of the upper curve), and that curves are fused at vertices. In this example, the latter effect introduces a gap in the lower right curve. It is thus important that r_{smooth} is chosen appropriately.

2.2 Triplet Grouping

The second step in the algorithm consists of building groups of three approximately collinear points, i.e. *triplets*. An example of a triplet is shown in Figure 3(a). Let the indices of the points A, B, and C in the point cloud be i, j, and k, that is $A = \vec{q}_i$, $B = \vec{q}_j$, and $C = \vec{q}_k$. The cosine of the angle α between the triplet branches is given by

$$\cos(\alpha) = \frac{\langle \overline{AB}, \overline{BC} \rangle}{\|\overline{AB}\| \cdot \|\overline{BC}\|} = \frac{\langle \vec{q}_j - \vec{q}_i, \vec{q}_k - \vec{q}_j \rangle}{\|\vec{q}_j - \vec{q}_i\| \cdot \|\vec{q}_k - \vec{q}_j\|}. \tag{1}$$

For the hierarchical clustering described in the following subsection, each triplet is represented



(a) nodes A, B, C and angle α (b) midpoint $\vec{m}$ and direction $\vec{e}$

Figure 3: Properties of a triplet.

by two vectors, the midpoint $\vec{m}$ and the direction $\vec{e}$ between the outer points

$$\vec{m} = \frac{1}{3}\left(\vec{q}_i + \vec{q}_j + \vec{q}_k\right) \quad \text{and} \quad \vec{e} = \frac{\vec{q}_k - \vec{q}_i}{\|\vec{q}_k - \vec{q}_i\|}. \tag{2}$$

The triplets are computed from the point cloud in a loop over all points. Each point is considered as a possible midpoint B, and all points among its $k_{triplet}$ nearest neighbors are tried as candidate points A and B. The triplet is discarded, if the triplet angle α is greater than a threshold, or, equivalently, if $\cos(\alpha)$ is below a threshold

$$\cos(\alpha) = \frac{\langle \overline{AB}, \overline{BC} \rangle}{\|\overline{AB}\| \cdot \|\overline{BC}\|} < 1 - a_{triplet}. \tag{3}$$

From the remaining triplets of each midpoint, only the $n_{triplet}$ triplets with the smallest angle α, or, equivalently, with the greatest $\cos(\alpha)$ are kept. For n points, the total number of triplets is thus not more than $n \cdot n_{triplet}$.

2.3 Hierarchical Clustering

The third step of the algorithm consists in a hierarchical clustering, which starts with each triplet as a separate cluster and merges two clusters in each iteration. The procedure with the most simple (threshold based) stopping criterion is formally described in Algorithm 1. It depends on a distance measure $cdist(C_i, C_j)$ on clusters C_i that can be constructed from a distance measure $d(x, y)$ on triplets x, y in different ways, known as *complete link*, *average link*, or *single link* clustering [16]. In our situation, only the single link method is appropriate because, for curved tracks, the angle distance between triplets of the same cluster can become large[3]. The single link method defines the cluster distance as

$$cdist(C_i, C_j) = \min\{d(x, y) \mid x \in C_i, y \in C_j\}. \tag{4}$$

This leads to the question of how to define a distance measure d on triplets.

2.3.1 Triplet Distance

Let us start with the observation that two triplets (A_i, B_i, C_i) and (A_j, B_j, C_j) are similar when three conditions hold:

1. The perpendicular distance $d_1^\perp$ between the midpoint $\vec{m}_i$ and the extrapolated line $\vec{m}_j + \lambda\vec{e}_j$ is small, which is

$$d_1^\perp = \|\vec{m}_j - \vec{m}_i + \langle \vec{m}_i - \vec{m}_j, \vec{e}_j \rangle \cdot \vec{e}_j\|. \tag{5}$$

[3] This does not hold when the curves are known to be *straight lines*. In this special case, the average link or complete link might be appropriate, too.

Algorithm 1: Hierarchical clustering with stopping threshold

```
input  : set of triplets $X = \{x_1, \dots, x_m\}$, threshold $t_{cluster}$ on cluster distance
output: triplet clustering $M = \{C_1, \dots, C_k\}$
$M \leftarrow \{C_i = \{x_i\}, i = 1, \dots, m\}$
for $i = 1, \dots, m-1$ do
    from all pairs $C_i, C_j \in M$, select the pair with smallest distance $cdist(C_i, C_j)$
    if $cdist(C_i, C_j) > t_{cluster}$ then
        break
    $C_h \leftarrow C_i \cup C_j$
    $M \leftarrow (M \setminus \{C_i, C_j\}) \cup \{C_h\}$
return $M$
```

2. The perpendicular distance $d_2^\perp$ between the midpoint $\vec{m}_j$ and the extrapolated line $\vec{m}_i + \lambda \vec{e}_i$ is small, which is
$$d_2^\perp = \|\vec{m}_i - \vec{m}_j + \langle \vec{m}_j - \vec{m}_i, \vec{e}_i \rangle \cdot \vec{e}_i\|. \tag{6}$$

3. The angle φ between their direction vectors $\vec{e}$ is small, which is
$$\varphi = \cos^{-1}\left(|\langle \vec{e}_i, \vec{e}_j \rangle|\right). \tag{7}$$

We combine these three distance measures into a single measure via

$$d\Big((A_i, B_i, C_i), (A_j, B_j, C_j)\Big) = \frac{\max\{d_1^\perp, d_2^\perp\}}{s_{cluster}} + |\tan\varphi|. \tag{8}$$

We use the tangent as an angle distance measure, and not one minus the cosine, because the tangent goes to infinity as $\varphi \to \pm\pi/2$, which means that perpendicular triplets have an infinite distance, regardless of their spatial distance. The scale factor $s_{cluster}$ allows for controlling the relative effect of angle and perpendicular distance.

It should be noted that Equation (8) does not define a metric in its strict mathematical meaning, because neither does a zero distance guarantee identity, nor is the triangle inequality satisfied. This is no problem, however, because the hierarchical clustering does not require a metric, but merely a dissimilarity measure. This even holds for the *fastcluster* implementation that we have used, as mentioned in [11, p. 3].

2.3.2 Stopping Criterion

A crucial point for hierarchical clustering is the criterion for stopping the agglomeration process. Algorithm 1 uses the most simple stopping criterion: a fixed threshold $t_{cluster}$ on the distance *cdist* for merging two clusters. This is a viable approach when the range of reasonable distances is fixed, e.g. by choosing the scale factor $s_{cluster}$ in such a way that the distance measure (8) becomes approximately scale invariant. Possible choices for $s_{cluster}$ and $t_{cluster}$ are discussed in Section 3.

A different approach is to automatically stop the clustering based on some inherent property of the data. The usual approach is to define an "internal cluster index" that measures the ratio of within and between cluster distances and to choose the number of clusters that optimizes this index [10]. In our situation of single link clustering, these internal indices are not applicable, however, because the within cluster distance can be infinite, e.g. when a curve is bent in such a way that two triplets on the curve are perpendicular, which results in an infinite distance (8).

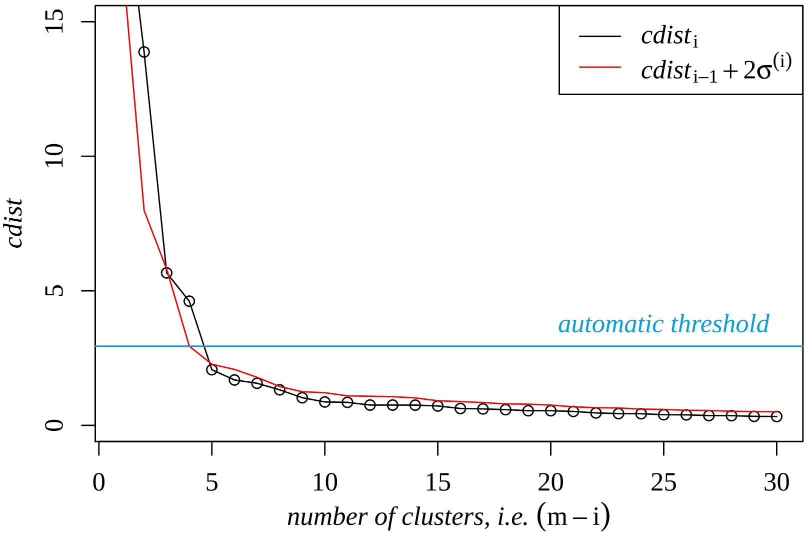


Figure 4: An example how the automatic threshold (9) stops the clustering at the first unusually large jump. Note that the number of clusters is not i, but $m - i$, so that the hierarchical clustering proceeds from right to left.

We therefore use a simple criterion that is based on the increase of the distance *cdist* in a merge step. Let $cdist_1 \leq cdist_2 \leq \ldots \leq cdist_{m-1}$ be the single link distance of the clusters that are merged in line 6 of Algorithm 1. The clustering is stopped at the first step $i > m/2$ for which $cdist_i$ is unexpectedly large. Here we define “unexpectedly” as greater than a two sigma interval around $cdist_{i-1}$

$$cdist_i > cdist_{i-1} + 2 \cdot \sigma^{(i)} \quad \text{with } \sigma^{(i)} = \sqrt{\frac{1}{i-1} \sum_{k=1}^{i} \left(\overline{cdist}^{(i)} - cdist_k \right)^2}. \tag{9}$$

Note that both the mean distance $\overline{cdist}^{(i)}$ and the standard deviation $\sigma^{(i)}$ are computed *including* the i-th value $cdist_i$, because otherwise the clustering would be stopped too early. We apply the test for the stopping rule only for $i > m/2$ to make sure that enough data goes into the computation of the statistic $\sigma^{(i)}$. Moreover, the clustering is not stopped when $\overline{cdist}^{(i)}$ is zero, or when $cdist_i$ is very small (less than 10^{-8}), in order to avoid a too early stopping. Figure 4 shows a typical example of how the automatic criterion (9) stops the clustering at the “elbow” of the cluster distances *cdist*.

The source code of our reference implementation that is available from the web page of this article[4] includes an R-script file for creating a plot as shown in Figure 4. This can be useful for finding a better threshold if the automatic criterion fails to yield a decent cluster number. See the file `README.txt` in the source code archive for details.

2.4 Pruning

To distinguish actual tracks from random clusters due to noise, it is necessary to remove some of the clusters in a post-processing step. We use a very simple rule and remove all clusters containing less than $m_{cluster}$ triplets. This cluster removal is done at the *triplet level*, which means that all triplets of the specific clusters are removed. If the data is known to be almost noise free, $m_{cluster}$ can be set to two. As this is rarely the case, $m_{cluster}$ should be set higher to control for the noise level.

Another optional pruning step is applied at the *point level*, i.e. after the triplet clusters have been transformed into clusters of points. As the distance measure given by Equation (8) does not consider the spatial distance between the points of the two triplets, it is possible that gaps occur

[4] https://doi.org/10.5201/ipol.2019.234

Table 1: Overview over the algorithm and the external parameters controlling each step.

Step	*Parameter*	*Default value*
1) neighborhood smoothing	r_{smooth} = neighbor distance	$2 \cdot d_{NN}$
2) triplet building	$k_{triplet}$ = tested neighbors of triplet mid point $n_{triplet}$ = max number of triplets to one mid point $a_{triplet} = 1 - \cos\alpha$, where α is the angle between the two triplet branches	19 2 0.03
3) triplet clustering	$s_{cluster}$ = distance scale factor in metric $t_{cluster}$ = threshold for *cdist* in clustering	$d_{NN}/3$ auto
4) pruning	$m_{cluster}$ = min number of triplets per cluster d_{max} = max gap width within cluster	5 none

in the clusters representing curves. For application cases in which this is not wanted, we introduce another parameter d_{max} and split up clusters at gaps greater than d_{max}. This is done for each cluster as follows:

1. We build the minimum spanning tree (MST) of the cluster *points* (not the triplets!), where the edge weight is the Euclidean distance between the points. This special case of a MST is known as the *Euclidean MST* [9].

2. Every edge with a weight greater than d_{max} is removed from the MST.

3. If edges have been removed, the tree is decomposed into its connected components. Connected components with less than $m_{cluster} + 2$ points[5] are discarded, and the remaining components are the new clusters.

3 Parameters

The algorithm can be controlled by several parameters which are listed in Table 1. Some of these parameters are thresholds on distances. Although these thresholds can be provided by an operator, it is preferable to have reasonable guesses for them based on a *characteristic length* that is inherent to the data. This makes the default parameter values scale invariant, which means that the algorithm yields the same result when all input points are scaled by the same factor. We therefore first describe a way to obtain this characteristic length, and then describe the different parameters from Table 1, except for the parameter $m_{cluster}$ which is self-explanatory.

3.1 Automatic Scale Factor: Characteristic Length d_{NN}

The distance between neighboring points within the same curve can be considered as a *characteristic length* of the data. It is also an indicator for the spread of points around the middle axis of the curve. To obtain an estimator for the point distance, we compute, for each point, the distance to its nearest neighbors. The first quartile of all these distances is our characteristic length d_{NN}.

We have chosen the first quartile, because typically noise points have a greater distance than points within a curve. Otherwise the concept of a curve would become meaningless. When all curves have a similar point density, the lower quartile is a robust estimator for the on-curve distance even for signal to noise ratios slightly less than one. This value can be considered as a scale parameter, because when all coordinates are scaled with the same factor, d_{NN} will scale with this factor, too.

[5] For $n_{triplet} = 1$, this is the minimum number of points for a cluster that consists of at least $m_{cluster}$ triplets.

3.2 Smoothing Radius: r_{smooth}

As shown in Figure 2(b), the neighborhood smoothing has the effect of moving points towards the medial axis of the curve. This means that it can be set to zero (no smoothing), when all points are known to lie on the curve and no random spread of the points perpendicular to the axis is present. If a spread is present, however, it should be set to the size of this spread. Under the assumption that points are randomly homogeneously shifted in all space directions, a reasonable guess for this spread is $r_{smooth} = 2 \cdot d_{NN}$.

To allow for an interactive control of the smoothing effect, the *triplclust* program has an optional Parameter `-v` (verbosity) that can be used to create a gnuplot file *debug_smoothed.gnuplot* for visualizing the smoothing result like in Figure 2.

3.3 Triplet Thresholds: $a_{triplet}$, $k_{triplet}$, $n_{triplet}$

These parameters control which groups of three points are acceptable triplets of points possibly belonging to the same curve. According to Equation (3), the default value $a_{triplet} = 0.03$ corresponds to a threshold of $|\alpha| < 14.07°$, where α is the angle between the triplet branches (see Figure 3(a)).

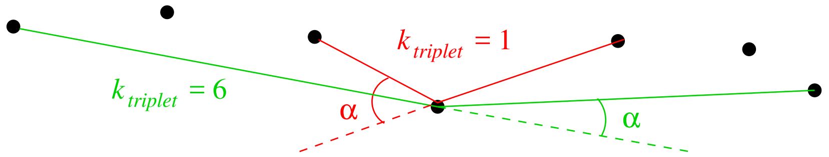
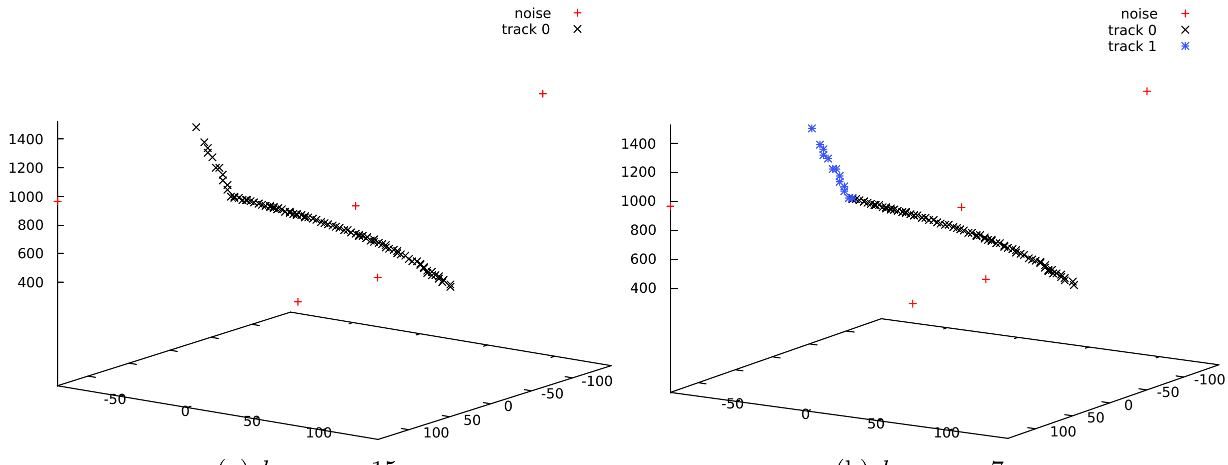


Figure 5: Triplets of points on the same curve tend to have a smaller branch angle α when the distance between the points is greater. The red triplet is built from the nearest neighbors of the midpoint, whilst the green triplet is the triplet with the smallest angle among the 6 nearest neighbors of the midpoint.



(a) $k_{triplet} = 15$

(b) $k_{triplet} = 7$

Figure 6: Effect of $k_{triplet}$: the smaller value splits the curve at the kink, the larger value doesn't.

Depending on the point density and the spread of the points around the curve, a threshold of $|\alpha| < 14.07°$ might rule out triplets of adjacent points. This can be compensated by a larger value of the parameter $k_{triplet}$, because triplets of points on the same curve with a greater distance have a smaller angle α than triplets of closer points. Figure 5 shows an example.

This does not mean, however, that $k_{triplet}$ should be chosen as large as possible, because a greater value increases the risk that intersecting curves are merged because triplets join points from different curves across a vertex. An example is shown in Figure 6.

The parameter $n_{triplet}$ was introduced primarily for performance reasons. Increasing it considerably increases the number of triplets used as input for the clustering. For small point clouds, this can improve the clustering result, but for large point clouds, the runtime increase was not justified by the results in the test cases that we tried. We therefore recommend to leave it at its default value, or even to decrease it for large point clouds with many points.

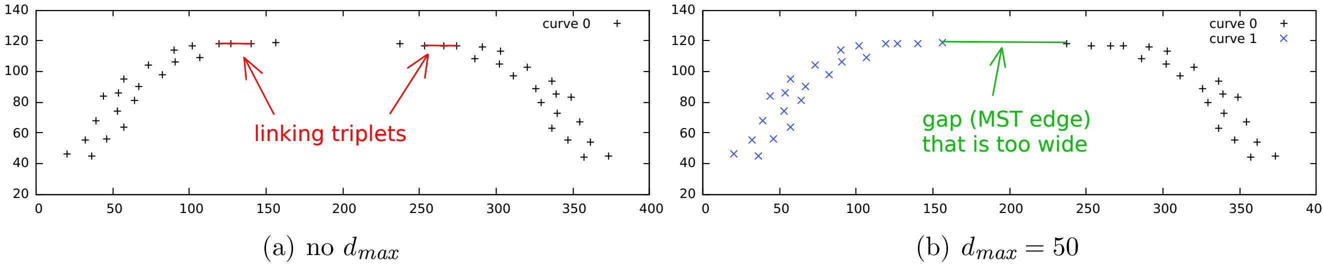
3.4 Clustering Control: $s_{cluster}$, $t_{cluster}$, d_{max}

The parameter $s_{cluster}$ controls the weight of the spatial distance with respect to the directional difference between two triplets, as can be seen in our definition of the triplet distance, Equation (8)

$$d\Big((A_i, B_i, C_i), (A_j, B_j, C_j)\Big) = \frac{\max\{d_1^{\perp}, d_2^{\perp}\}}{s_{cluster}} + |\tan\varphi|. \qquad (8, \text{repeated})$$

As the unit of length for measuring the point coordinates is arbitrary, it is important to scale $s_{cluster}$ with the characteristic length of the data d_{NN}, for which we use the lower quartile of all nearest neighbor distances (see Section 3.1). The experimental parameter optimization performed in the study [3] yielded optimal values between one half and one fifth of d_{NN}. We therefore recommend a default value of $s_{cluster} \approx d_{NN}/3$.

The cutoff threshold $t_{cluster}$ for the clustering process depends on the actual values for the triplet distances given by Equation (8). This means that a change of the parameter $s_{cluster}$ can make a change in the parameter $t_{cluster}$ necessary, too. We therefore suggest, to try first the automatic choice of $t_{cluster}$ as it is described in Section 2.3.2. Our reference implementation provides an option for printing the resulting threshold value. When this results in too many or too few clusters, different values for $t_{cluster}$ can be tried, starting from the value determined by the automatic stopping criterion. In the experimental study [3], the value $s_{cluster} = d_{NN}/3$ corresponded approximately with values $t_{cluster}$ in the range between 2 and 3.5.



(a) no d_{max}

(b) $d_{max} = 50$

Figure 7: 2D example (all z-components zero) showing the effect of d_{max} for splitting up clusters at large gaps. The example was processed with $r_{smooth} = 0$ so that the triplets were built from the displayed points.

As the distance measure (8) does not take the translational distance into account, the clustering can overcome large gaps in a curve, provided the continuation after the gap is an extrapolation of the curve before the gap. Due to the single link method, it is sufficient if two triplets of the different branches have a similar angle and their mutual extensions come close. A typical example can be seen in Figure 7(a). In some applications, this is the desired behavior. If not, large gaps can be split up by setting the parameter d_{max} to the maximum of the permitted gap width (see Figure 7(b)).

4 Time and Space Complexity

The *time complexity* is given by the order of the slowest step among the four sequential steps listed in Section 2. Here is a runtime analysis of the individual steps where n is the number of points in the point cloud:

1. Both the estimation of d_{NN} and the neighborhood smoothing are $O(n^2)$ operations when they are implemented by simple loops. The nearest neighbor retrieval can be reduced to $O(n \log n)$ time with a kd-tree, albeit only on average.

2. The number of candidate triplets, from which approximately collinear triplets are selected, is $\binom{n}{3}$, which is $O(n^3)$. As we do not try all possible triplets, but only search for candidate points among the $k_{triplet}$ nearest neighbors, the runtime reduces to $O(k_{triplet}^3 \cdot n \log n)$, where the factor $n \log n$ stems from the all-nearest-neighbor search in the kd-tree.

3. The runtime complexity of Algorithm 1 is $O(m^3)$ where m is the number of triplets [16], but this can be reduced to $O(m^2)$ with the utilization of Rohlf's MST-algorithm [11, 14]. As we make the restriction of keeping only the $n_{triplet}$ "best" triplets for each candidate midpoint, the number of triplets is only $O(n)$, not $O(n^3)$, as it were if we kept all triplets below the given collinearity threshold $a_{triplet}$. The runtime of the clustering step is thus $O(n^2)$.

4. As we cannot have more clusters than triplets, the runtime of the pruning is $O(m)$, which is $O(n)$ due to the restriction mentioned in the preceding step.

The runtime is thus dominated by the hierarchical clustering and is $O(n^2)$ in total. For the nearest neighbor and range search, we have used the C++ kd-tree implementation that one of us had written for the Gamera framework [2], and which we have extended with a range search method. For the hierarchical clustering, we have used the library *fastcluster* [12]. On a point cloud consisting of 666 points with much noise and strongly curved tracks, which resulted in a large number of candidate triplets, the runtime was about 0.5 seconds on an Intel Core i5-6500 CPU @ 3.20GHz processor. The clustering step took about 70% of the total runtime.

The *space complexity* is dominated by the clustering, too. When m is the number of triplets, the clustering requires the computation of a condensed distance matrix which has $\binom{m}{2}$ entries. As explained above, the number of triplets is $O(n)$, which results in a space complexity of $O(n^2)$.

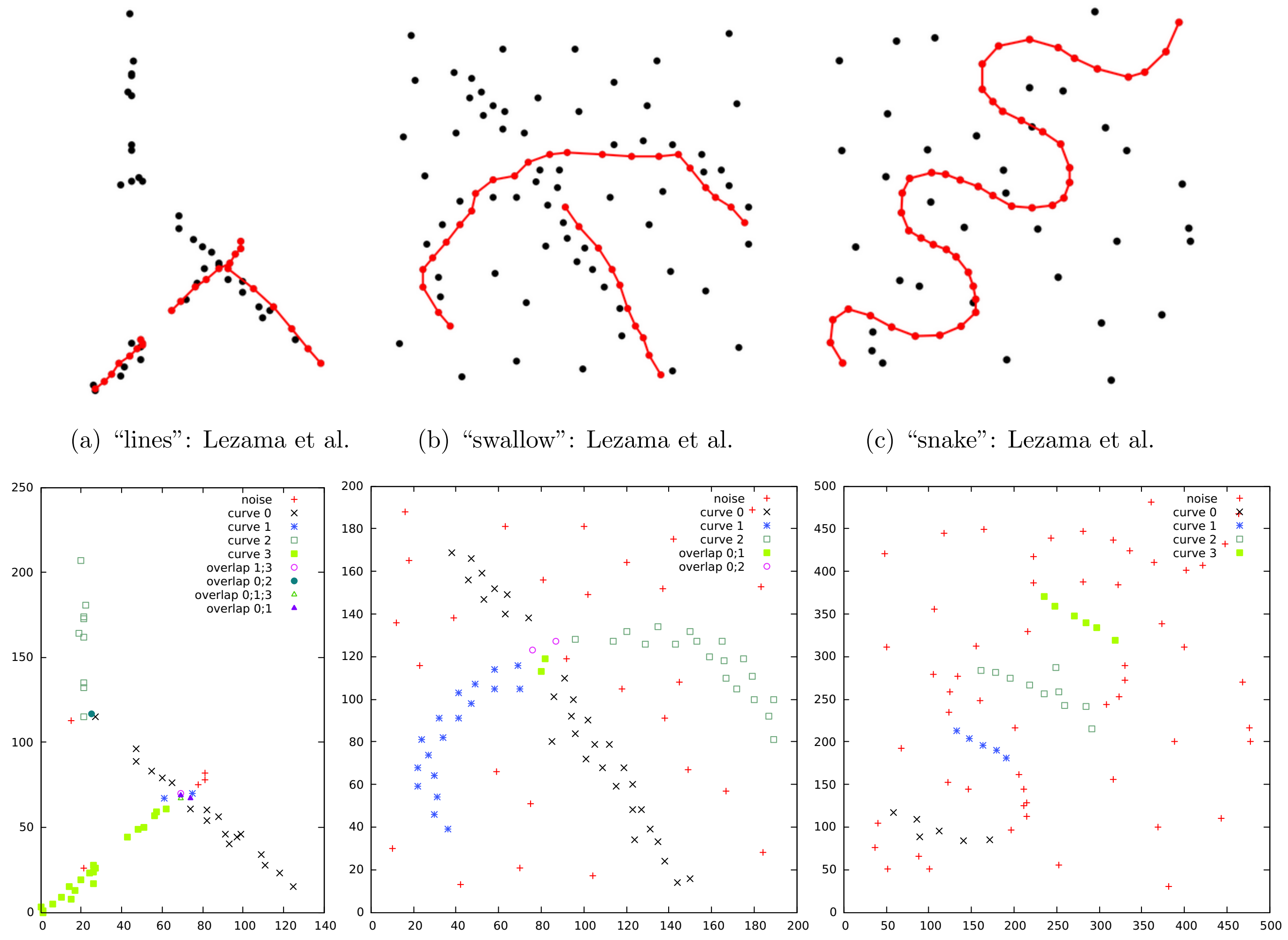
5 Comparison with other IPOL algorithms

Two other algorithms have been published in IPOL that also aim at finding grouping structures in point clouds: an unsupervised algorithm for detecting good continuations in dot patterns [8], and the iterative Hough transform [4]. It is thus interesting to consider examples demonstrating the limitations or benefits of TriplClust with respect to these algorithms.

5.1 Good Continuation in Dot Patterns

This algorithm by Lezama et al. works on 2D data only. We therefore have created two 2D point clouds: one with three "lines" randomly sampled and shifted, and one with a manually drawn "swallow" with noise points. Additionally, we have used a third point cloud ("snake") that Lezama et al. presented as a working example of their algorithm in Figure 5 of their publication [8]. It should be noted that the continuation algorithm was devised to work on approximately equidistantly sampled skeleton curves, whilst TriplClust was designed to work on points randomly spread along curves. Both algorithms thus aim at different use cases, but it is nevertheless interesting to compare their results. As the online demo for the algorithm of Lezama does not support larger point clouds, we have resorted to small examples and thus had to reduce the TriplClust parameter $k_{triplet}$ from its default value 19 to 7.

The results of the two algorithms are shown in Figure 8. For the "lines" example (Figures 8(a) and 8(d)), the algorithm by Lezama et al. extracts parts of the lines as a skeleton, but one line



(a) “lines”: Lezama et al. (b) “swallow”: Lezama et al. (c) “snake”: Lezama et al.

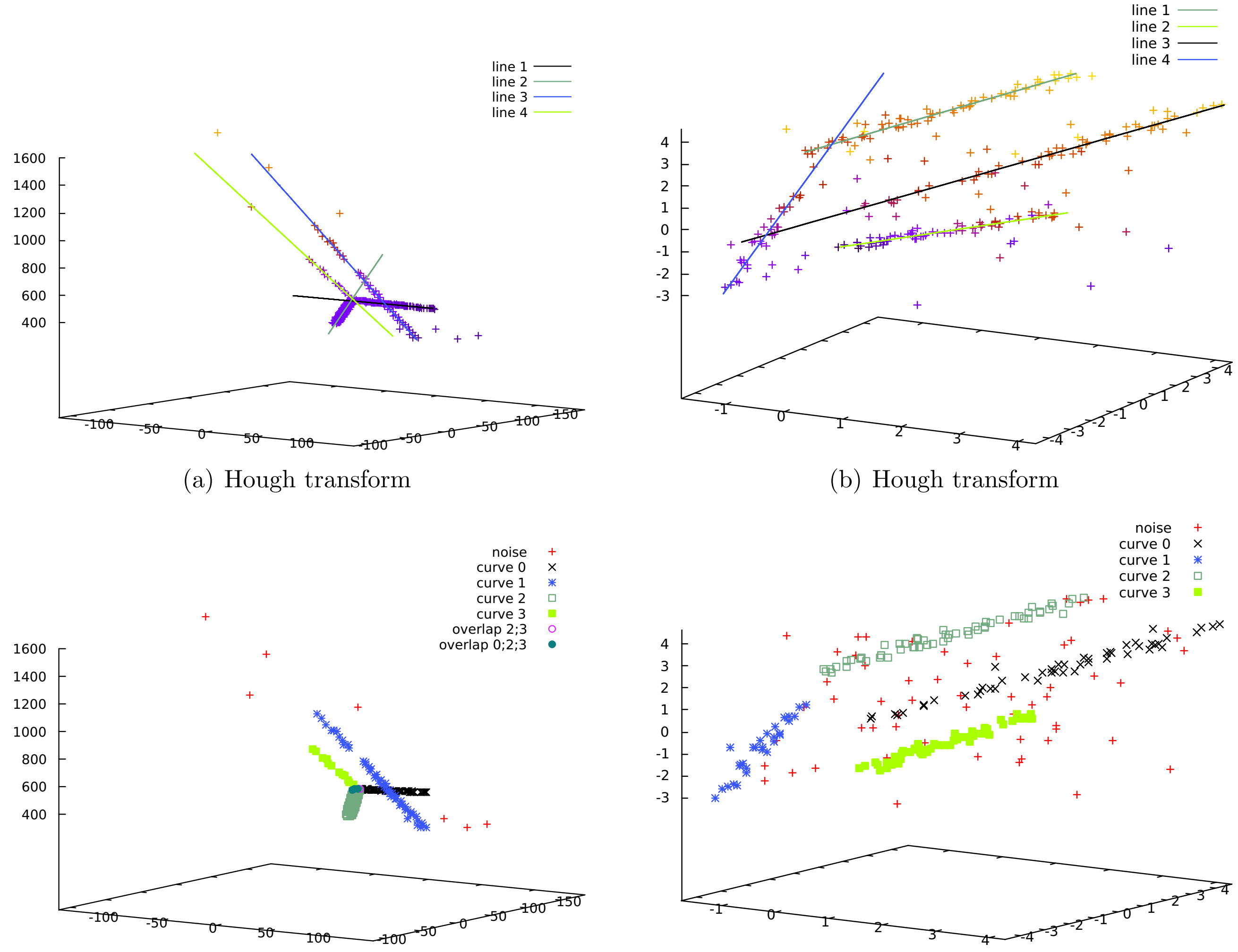
(d) “lines”: TriplClust (e) “swallow”: TriplClust (f) “snake”: TriplClust with $t_{cluster} = 2$

Figure 8: Comparison of the algorithm by Lezama et al. (top row) with TriplClust (bottom row) on 2D examples. As the curves consisted of only few points, we have reduced the parameter $k_{triplet}$ to 7.

is interrupted and one line is missed. Whilst TriplClust finds all lines, it detects an additional cluster (curve 1) near the crossing point of curves 0 and 3. For the “swallow” example (Figures 8(b) and 8(e)), the algorithm by Lezama et al. extracts part of the skeleton, but misses the top of the “swallow”. In this example, TriplClust even joins the top with the body of the “swallow”, and also separates the object correctly from the noise. In the “snake” example (Figures 8(c) and 8(f)), the algorithm by Lezama et al. perfectly extracts the skeleton because the snake is already skeletonized and locally approximately equidistantly sampled. This is an example for which TriplClust does not work, even with a manually set threshold $t_{cluster} = 2$, because the curve is only thinly sampled and the triplet clustering does not merge triplets across the sharp bends.

5.2 Iterative Hough Transform

The iterative Hough transform searches, in each iteration, for the straight line that is close to most of the points. A comparison with TriplClust thus only makes sense for the special case of linear curves. Figure 9 shows the result of the iterative Hough transform (top row) and TriplClust (bottom row) on two sample data from the IPOL online demo that accompanies [4]. As the curves are known to be straight lines, the parameter $a_{triplet}$ can also be decreased, e.g. to 0.003, but for the sample data



(a) Hough transform

(b) Hough transform

(c) TriplClust

(d) TriplClust

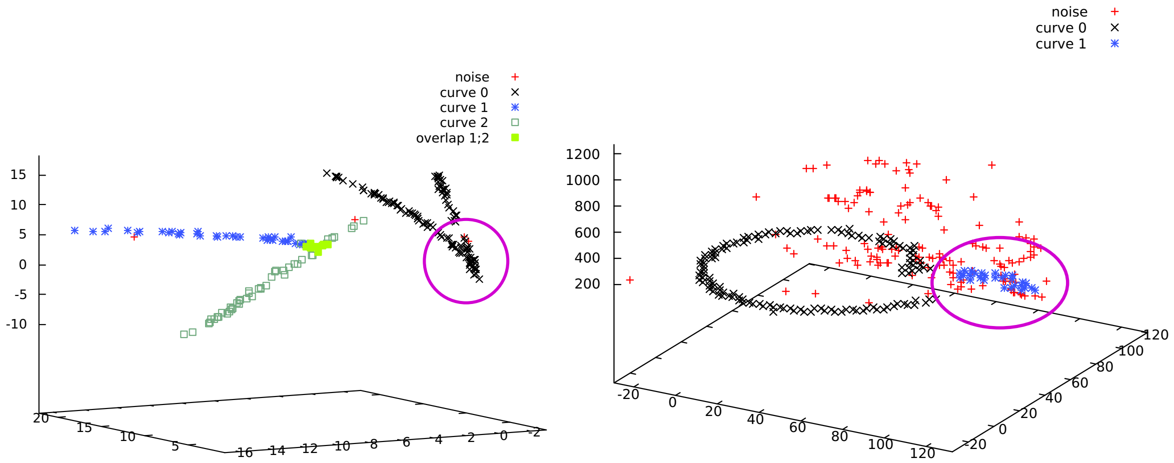
Figure 9: Comparison of the iterative Hough transform (top row) with TriplClust (bottom row). The Hough transform requires a parameter *minvotes*, which we have set to 10.

shown in Figure 9, this was not necessary.

The results of both algorithms are very similar, and the lines returned by the Hough transform can be extracted from the TriplClust output, too, by means of an orthogonal least squares fit through the points of each cluster [15]. Compared to the iterative Hough transform, TriplClust has the advantage that line intersections are automatically detected as points belonging to more than one cluster (see Figure 9(c)). Moreover, it does not require a space quantization (parameter dx of the Hough transform). The Hough transform is, however, more robust with respect to noise: when we added more random noise to the data from Figures 9(d), we had to reduce the parameter $a_{triplet}$. In the extensive study [3], TriplClust performed better than the iterative Hough transform on AT-TPC data containing only straight particle tracks.

6 Cases of Failure

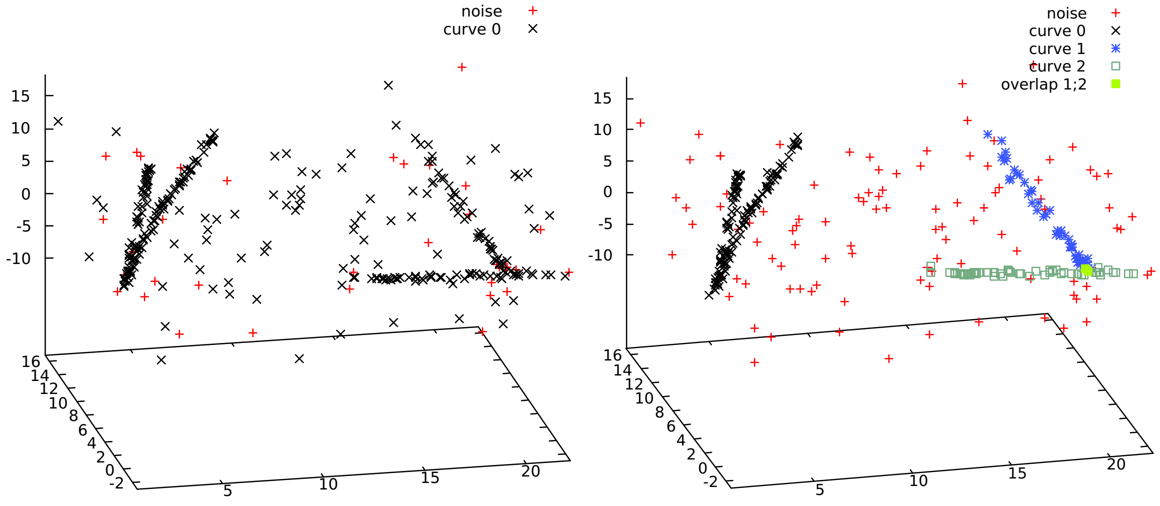
There are a number of situations in which TriplClust cannot detect the correct number of clusters. One is shown in Figure 10(a): here “curve 0” actually consists of *two* different curves that are touching tangentially. This has the effect that triplets from the different curves around the touch point are very close and create a link between the two curves.



(a) merged clusters

(b) falsely detected cluster

Figure 10: Examples where the algorithm does not detect the correct number of clusters.



(a) default $r_{smooth} = 2 \cdot d_{NN} = 0.39$

(b) $r_{smooth} = 1.0$

Figure 11: Examples where the characteristic length d_{NN} is too small for sufficient smoothing when the default value for r_{smooth} is used. All other parameters have been set to their default values.

Other problems can occur due to noise. In Figure 10(b), part of the noise has been recognized as a cluster due to alignment of triplets by chance. This occurs in Figure 11(a) too, where the triplets in the noise even create links between the actual curves, thereby resulting in only a single cluster.

The point cloud in Figure 11 shows another problem: when the points on the curves are quite dense, the estimation of the characteristic length d_{NN} is too low, which means that too little smoothing is done. This has the effect that the angle of the triplets on the curves vary considerably, which leads to linkage with some noise triplets. In this example, the problem can be solved by setting r_{smooth} greater than the default value, as can be seen in Figure 11(b)

7 Online Demo and Command Line Program

The source code of a command line program `triplclust` that implements this algorithm is provided on the web page of this article[6]. On the same website, there is also an online demo built on top of this command line program. The users can choose among several pre-uploaded data sets or upload their own cloud of 3D or 2D points in the following file format:

- The line separator is the newline character (`\n`).
- Lines starting with a hash symbol (`#`) are comments.
- Each line contains the 3D coordinates of one point as three floating point numbers separated by single spaces, as in the following example:

```
# point cloud data from a wondrous experiment
41.7201 138.2140 -648.0000
0.0001 -138.2140 -440.0000
2.4543 -136.8650 -436.8000
```

The online demo only supports a space character as field separator, but, for the command line program, a different field separator can be specified with the parameter *-delim*. When each line only contains two coordinates, the data are interpreted as 2D points, i.e. the z component of each point is set to zero. Columns after the third column are silently ignored. When the point cloud data has been selected, the following parameters can be chosen:

Smoothing radius (r). This is r_{smooth} which is described in Section 3.2. Can be given as an absolute number or a multiple of d_{NN}, e.g. `5.3dNN`. Defaults to `2dNN`.

NNs for triplet building (k). This is $k_{triplet}$ which is described in Section 2.2. Defaults to 19.

Triplets per point (n). This is $n_{triplet}$ which is described in Section 2.2. Defaults to 2.

Triplet angle threshold (a). This is $a_{triplet}$ which is described in Section 2.2. Defaults to 0.03, which corresponds to an angle $\alpha = \cos^{-1}(1 - 0.03) \approx 14°$.

Distance scale factor (s). This is $s_{cluster}$ which is described in Section 2.3.1. Can be given as an absolute number or a multiple of d_{NN}, e.g. `0.5dNN`. Defaults to `0.33dNN`.

Clustering threshold (t). This is $t_{cluster}$ which is described in Section 2.3.2. Can be given as an absolute number or `auto`, which corresponds to the automatic threshold selection described in Section 2.3.2. Defaults to `auto`.

Min triplets per cluster (m). This is $m_{cluster}$, an integer number. Defaults to 5.

Max gap width within cluster ($dmax$). This is d_{max} which is described in Section 2.4. Can be given as a number or `none`. Defaults to `none`.

The detected clusters are returned as labels assigned to the points in CSV format as a fourth column:

[6] https://doi.org/10.5201/ipol.2019.234

```
# Comment: curveID -1 represents noise
# x, y, z, curveID
2.4541,52.4586,64,-1
-66.2612,15.5519,679.6,0
-51.5365,7.05056,697.6,0
-49.0824,5.70141,701.2,-1
85.8941,18.4534,892,0;1
100.619,26.9547,895.6,1
88.3483,19.8025,895.6,0;1
103.073,28.3038,895.6,1
```

Note that the original point order is not changed, which means that the points are not ordered by cluster label. Points classified as noise have the label "-1", and for points belonging to more than one cluster, all cluster labels are given in a semicolon-separated list.

The web demo stores the above output in the downloadable file `result.csv` and displays a 3D plot of the detected clusters, where each cluster is highlighted by a different color. Moreover, the used parameters are stored in the downloadable file `output.txt`:

```
[Info] computed dnn: 2.90151
[Info] computed smoothed radius: 5.80302
[Info] computed distance scale: 0.870453
[Info] optimal cdist threshold: 3.6284
[Info] in pruning removed clusters: 3
```

This can be useful for trying out modified parameters after the algorithm has computed the default values.

8 Reference Data Sets

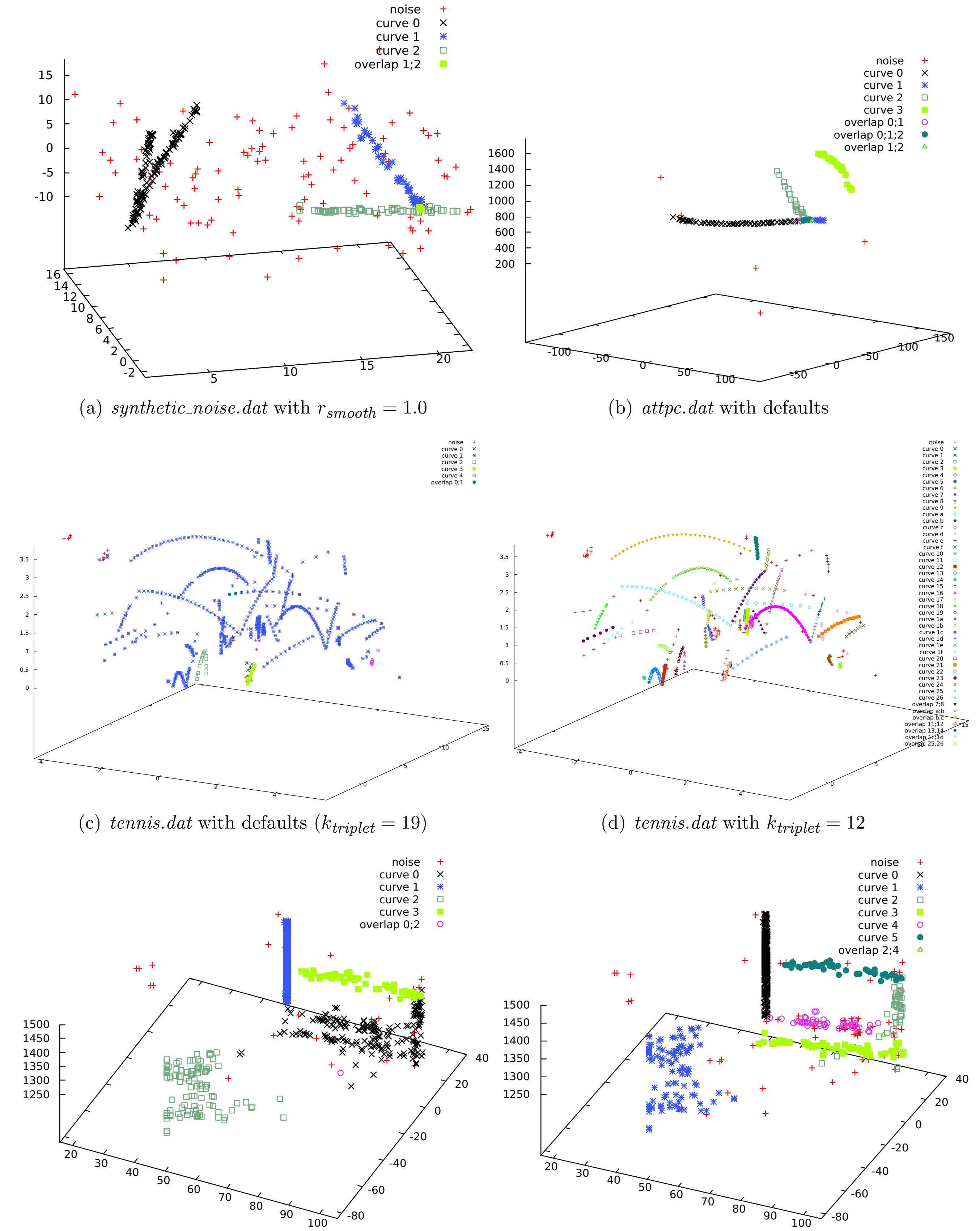
As supplementary material, we provide six point clouds, of which two are synthetically generated, and four are real 3D sensor data. The reference data sets can be found in the subdirectory `data` of the source code package.

The synthetic file `synthetic-clean.dat` contains 225 points that have been been randomly sampled from four quadratic splines with random Gaussian coordinate displacement *rnorm* with $\mu = 0$

$$\vec{x}(t) = \begin{pmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{pmatrix} \cdot \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix} + \begin{pmatrix} rnorm(0,\sigma) \\ rnorm(0,\sigma) \\ rnorm(0,\sigma) \end{pmatrix} \tag{10}$$

The file `synthetic-noise.dat` contains additional 100 random points uniformly sampled in the range of the bounding box of the four curves. As already discussed in Section 6, the automatically computed smoothing radius r_{smooth} is too small for the tracks in the noisy data to be separated. When the automatically computed value $r_{smooth} = 0.39$ is replaced by $r_{smooth} = 1.0$, all tracks and the noise are identified, as shown in Figure 12(a).

The file `attpc.dat` stems from the active target time projection chamber (AT-TPC) at the National Superconducting Cyclotron Laboratory, Michigan State University [1]. This was the kind of data for which we originally devised the algorithm [3]. It contains tracks of elementary particles in a nuclear physics experiment. The result of the algorithm is shown in Figure 12(b). All tracks and the vertex are correctly identified with the default parameters.



(a) *synthetic_noise.dat* with $r_{smooth} = 1.0$

(b) *attpc.dat* with defaults

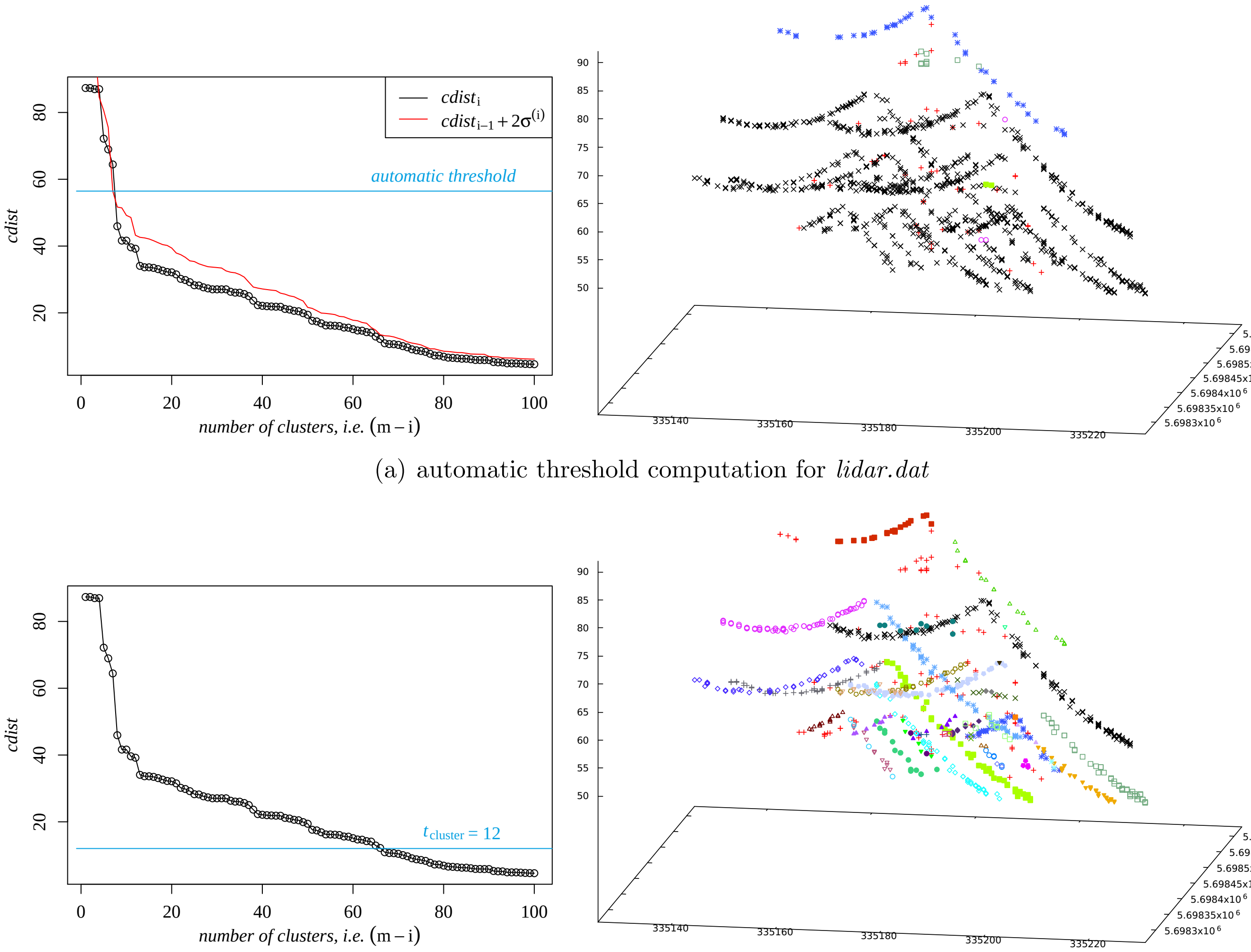
(c) *tennis.dat* with defaults ($k_{triplet} = 19$)

(d) *tennis.dat* with $k_{triplet} = 12$

(e) *radar.dat* with defaults ($a_{triplet} = 0.03$)

(f) *radar.dat* with $a_{triplet} = 0.003$

Figure 12: TriplClust results on the noisy synthetic data, the AT-TPC data, the tennis data, and the radar data. All parameters except the given parameters have been set to their default value listed in Table 1.



(a) automatic threshold computation for *lidar.dat*

(b) manually set $t_{cluster} = 12$ for *lidar.dat*

Figure 13: Progress of *cdist* and TriplClust results on the LIDAR data sets. The number of neighbors for triplet building has been set $k_{triplet} = 12$. Other parameters are set to their default value listed in Table 1.

The file `tennis.dat` has been extracted by by Vito Renò et al. from video recordings of a tennis match with the method described in [13]. The resulting data contained candidate points for tracking the tennis ball. From this data we have selected a contiguous sequence of 701 points. The original data also contained a time variable, which can be valuable for disambiguating close by curves. As this was not of interest in our case, we have discarded the time variable in our test file. In this point cloud, the point density within the curves is so low that the default value $k_{triplet} = 19$ is too large, because it leads to many triplets joining points from different curves. As can be seen in Figure 12(c), this has the effect of merging most curves. When $k_{triplet}$ is decreased, the clusters are split up and the individual curve segments are mostly correctly separated (Figure 12(d)).

The file `radar.dat` has been recorded with a 24 GHz FMCW-Radar kit by IMST GmbH. It only contains *two* spatial coordinates and the third coordinate is the recording time. This means that vertical straight lines correspond to non-moving objects, while moving objects create non-vertical curves. The data have been recorded at a crossroad and converted in a rough way to a point cloud by thresholding at the maximum minus 3dB per sweep. The three car tracks that are visible in the right half of Figure 12(e) are clustered into only two tracks (curve 0 and curve 3) with the default parameter values. The two merged tracks are split up, when more strictly collinear triplets are demanded by setting $a_{triplet} = 0.003$, as shown in Figure 12(f).

The file `lidar.dat` contains an airborne LIDAR scan of a high voltage pole with about seven dots per square meter. The original data in this excerpt from "Geobasisdaten der Kommunen und des Landes NRW 2014" had 5813 points, from which we have randomly sampled 1000 points for performance reasons. As this resulted in a lower point density along the curves, we have again decreased $k_{triplet}$ to 12. This is an interesting example where the automatic stopping criterion for the hierarchical clustering (see Section 2.3.2) yields poor results, as shown in Figure 13(a). The cluster merge distance *cdist* shows a number of leaps, but the first[7] leap at about $t_{cluster} = 12$ is too small to be detected by the automatic stopping criterion. When the threshold is set manually to this value, the clusters are split up and the high voltage lines are well detected (Figure 13(b)). As it should be, almost all of the power lines are split up into two curves at the top kink, with the notable exception of the black curve. This curve remains merged, even when $k_{triplet}$ is decreased.

9 Conclusion

TriplClust is an algorithm for segmenting point clouds into curves that already has demonstrated its usefulness for analyzing AT-TPC data in the previous study [3]. In this article, we have made suggestions for automatically computed default values for its parameters and for stopping the clustering process. As the reference examples show, these defaults do not work for all point clouds, but some parameters require tweaking, depending on the data point cloud. The reference implementation makes these modifications easy, by optionally returning the computed default values, which can then be used as a starting point for modifying some parameters.

The main idea of TriplClust is the reformulation of the curve detection problem as a clustering problem on point triplets. This reformulation requires an appropriate distance measure on triplets, for which Equation (8) is just one possibility. Our distance measure tries to combine the angle difference and the orthogonal distance into a single number, as it is required for a distance measure. It would be interesting to investigate which, and whether, other distance measures can improve the curve detection.

Acknowledgment

The authors are grateful to Yassid Ayyad for providing sample data from the active target time projection chamber (AT-TPC) at the National Superconducting Cyclotron Laboratory, Michigan State University, to Vito Renò for providing one of the tennis ball data sets recorded in the study [13], to Christoph Degen for providing the radar data, and to the Katasteramt Krefeld for providing us with the laser scan data from "Geobasisdaten der Kommunen und des Landes NRW 2014". Moreover, we thank the anonymous reviewers for their valuable comments.

References

[1] Y. Ayyad, W. Mittig, D. Bazin, S. Beceiro-Novo, and M. Cortesi, *Novel particle tracking algorithm based on the random sample consensus model for the active target time projection chamber (AT-TPC)*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 880 (2018), pp. 166–173. https://doi.org/10.1016/j.nima.2017.10.090.

[7] The clustering merge process proceeds from right to left. This means that the *first* leap is the *rightmost* leap in Figure 13.

[2] C. Dalitz, *Kd-trees for document layout analysis*, in Document Image Analysis with the Gamera Framework, Schriftenreihe des Fachbereichs Elektrotechnik und Informatik, Hochschule Niederrhein, C. Dalitz, ed., vol. 8, 2009, pp. 39–52.

[3] C. Dalitz, Y. Ayyad, J. Wilberg, L. Aymans, D. Bazin, and W. Mittig, *Automatic trajectory recognition in Active Target Time Projection Chambers data by means of hierarchical clustering*, Computer Physics Communications, 235 (2019), pp. 159–168. https://doi.org/10.1016/j.cpc.2018.09.010.

[4] C. Dalitz, T. Schramke, and M. Jeltsch, *Iterative Hough transform for line detection in 3D point clouds*, Image Processing On Line, 7 (2017), pp. 184–196. https://doi.org/10.5201/ipol.2017.208,.

[5] H. Fan, W. Yao, and Q. Fu, *Segmentation of sloped roofs from airborne LIDAR point clouds using ridge-based hierarchical decomposition*, Remote Sensing, 6 (2014), pp. 3284–3301. https://doi.org/10.3390/rs6043284.

[6] M.A. Fischler and R.C. Bolles, *Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography*, in Readings in Computer Vision, Martin A. Fischler, , and Oscar Firschein, eds., Morgan Kaufmann, San Francisco (CA), 1987, pp. 726–740.

[7] I-K. Lee, *Curve reconstruction from unorganized points*, Computer Aided Geometric Design, 17 (2000), pp. 161–177.

[8] J. Lezama, G. Randall, J-M. Morel, and R. Grompone von Gioi, *An unsupervised algorithm for detecting good continuation in dot patterns*, Image Processing On Line, 7 (2017), pp. 81–92. https://doi.org/10.5201/ipol.2017.176,.

[9] W.B. March, P. Ram, and A.G. Gray, *Fast Euclidean minimum spanning tree: algorithm, analysis, and applications*, in Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 603–612.

[10] G.W. Milligan, *A Monte Carlo study of thirty internal criterion measures for cluster analysis*, Psychometrika, 46 (1981), pp. 187–199.

[11] D. Müllner, *Modern hierarchical, agglomerative clustering algorithms*, 2011. http://adsabs.harvard.edu/abs/2011arXiv1109.2378M.

[12] D. Müllner, *fastcluster: Fast hierarchical, agglomerative clustering routines for R and Python*, Journal of Statistical Software, 53 (2013), pp. 1–18. https://doi.org/10.18637/jss.v053.i09.

[13] V. Renò, N. Mosca, M. Nitti, C. Guaragnella, T. D'Orazio, and E. Stella, *Real-time tracking of a tennis ball by combining 3D data and domain knowledge*, in Proceedings of International Conference on Technology and Innovation in Sports, Health and Wellbeing (TISHW), 2016, pp. 1–7. https://doi.org/10.1109/TISHW.2016.7847774,.

[14] F.J. Rohlf, *Algorithm 76: Hierarchical clustering using the minimum spanning tree*, The Computer Journal, 16 (1973), pp. 93–95.

[15] H. Späth, *Orthogonal least squares fitting with linear manifolds*, Numerische Mathematik, 48 (1986), pp. 441–445.

[16] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Academic Press, 4 ed., 2009.

[17] G. Zhao and J. Yuan, *Curb detection and tracking using 3D-LIDAR scanner*, in Proceedings of IEEE International Conference on Image Processing (ICIP), 2012, pp. 437–440. https://doi.org/10.1109/ICIP.2012.6466890,.