Published in Image Processing On Line on 2017–07–19.
Submitted on 2017–03–20, accepted on 2017–06–12.
ISSN 2105–1232 © 2017 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.2017.208

Iterative Hough Transform for Line Detection in 3D Point Clouds

Christoph Dalitz, Tilman Schramke, Manuel Jeltsch

Institute for Pattern Recognition (iPattern)
Niederrhein University of Applied Sciences, Krefeld, Germany
(christoph.dalitz@hs-niederrhein.de,
tilman.schramke@stud.hn.de,
manuel.jeltsch@gmail.com)

Communicated by Bertrand Kerautret *Demo edited by* Bertrand Kerautret

Abstract

The Hough transform is a voting scheme for locating geometric objects in point clouds. This paper describes its application for detecting lines in three dimensional point clouds. For parameter quantization, a recently proposed method for Hough parameter space regularization is used. The voting process is done in an iterative way by selecting the line with the most votes and removing the corresponding points in each step. To overcome the inherent inaccuracies of the parameter space discretization, each line is estimated with an orthogonal least squares fit among the candidate points returned from the Hough transform.

Source Code

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

Supplementary Material

Six reference data sets are provided with the article. The data sets contain both synthetic data and experimental data of radioactive beams recorded in an active target time projection chamber (AT-TPC).

Keywords: Hough transform; object recognition; 3D point clouds; orthogonal least squares fit

[1] https://doi.org/10.5201/ipol.2017.208

1 Introduction

In 1962 Hough patented a method for finding lines in 2D images [3]. Meanwhile the underlying idea has been generalized for the detection of a wide variety of parametric shapes, all of which are covered by the term *Hough transform* [5]. This article is devoted to its application for detecting lines in three dimensional data.

Unlike other shape detection algorithms, the Hough transform does not work on raster images, but on point clouds. Some 3D sensing methods like Active Target Time Projection Chamber (AT-TPC) [8] or Light Detection and Ranging (LIDAR) [2] directly yield point clouds, while others like Nuclear Magnetic Resonance Spectroscopy (NMR) yield voxel rasters. Applying the Hough transform in the latter case thus requires a pre-processing step for filtering candidate points. This might even be necessary on native point cloud data, because otherwise the characteristic points go unnoticed among a majority of points carrying no shape information. The present article assumes that this pre-processing, if necessary, has already been done.

The algorithm takes a point cloud $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$ as input. Its objective is to find lines close to which large numbers of points from X are located. The idea of the Hough transform is to make the infinite space of all possible lines finite by a discretization of the parameter space, and to let each point "vote" for all lines to which it belongs in this parameter space. Parameter cells with many votes then correspond to lines with many points.

In [4], Jeltsch et al. have proposed a new scheme to discretize the Hough parameter space of 3D lines. The scheme is based on Roberts' minimal and optimal line representation [6] and on a tessellation of a platonic solid for determining the set of line orientations. The present implementation is based on this scheme, but additionally addresses the following two points:

- Selecting the lines from local maxima in the voting space, as proposed in [4] ("non maximum suppression"), can still lead to many nearby lines, because each point votes for many lines. We circumvent this problem by an iterative application of the Hough transform and by removing the points of a detected line after each iteration.

- To improve the accuracy of the detected lines, an orthogonal least squares fit is made through the points of each line.

This article is organized as follows: Section 2 describes both the complete algorithm and its individual steps in detail, Section 5 describes the online demo provided on the web page of this article[2], and Section 6 describes the test data set provided on the same web page and the results of the algorithm on these data sets.

2 Algorithm

In a narrow sense, the term *Hough transform* refers to building an accumulator array with each cell representing one set of parameters from the discretized parameter space, and to increase for each point cloud element the counter of all cells to which this point might belong. The resulting accumulator array can thus be considered a "transform" of the original point cloud into the parameter space. This Hough transform is a subroutine in our *iterative Hough transform* algorithm, which consists of the following steps (see Algorithm 1):

1) Discretization of the parameter space for all lines crossing the point cloud volume.

2) Hough transform of the point cloud X based on the discretization from step 1.

[2] https://doi.org/10.5201/ipol.2017.208

Algorithm 1: IterativeHoughTransform

input : point cloud $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$, step width dx,
minimum vote count n_{min}, maximum number of lines k_{max}
output: lines in vector form $L = \{(\vec{a}_i, \vec{b}_i) \mid i = 1, \ldots, k\}$
compute direction vectors $B = \{\vec{b}_1, \ldots, \vec{b}_{N_1}\}$ *see sections 2.1 and 2.2*
compute x'-discretization $X' = \{x'_1, \ldots, x'_{N_2}\}$
compute y'-discretization $Y' = \{y'_1, \ldots, y'_{N_3}\}$
$L \leftarrow \{\}$
accumulator array $A \leftarrow \mathit{HoughTransform}(X, B, X', Y', \text{“+”})$ *see Section 2.3 (Algorithm 2)*
repeat
  $(\vec{a}, \vec{b}) \leftarrow$ line corresponding to maximum of A *see Equations (1) and (3)*
  $Y \leftarrow \{\}$
  foreach $\vec{x} \in X$ **do**
    if $\|\vec{a} + \langle\vec{b}, \vec{x} - \vec{a}\rangle\vec{b} - \vec{x}\| < dx$ **then** *orthogonal distance to line less than dx*
      $Y \leftarrow Y \cup \{\vec{x}\}$
  $(\vec{a}, \vec{b}) \leftarrow$ orthogonal least squares fitted line to Y *see Section 2.4*
  $Y \leftarrow \{\}$
  foreach $\vec{x} \in X$ **do**
    if $\|\vec{a} + \langle\vec{b}, \vec{x} - \vec{a}\rangle\vec{b} - \vec{x}\| < dx$ **then** *orthogonal distance to line less than dx*
      $Y \leftarrow Y \cup \{\vec{x}\}$
  $X \leftarrow X \setminus Y$
  $A \leftarrow \mathit{HoughTransform}(Y, B, X', Y', \text{“−”})$ *remove points from accumulator array*
  if $|Y| \geq n_{min}$ **then** $L \leftarrow L \cup \{(\vec{a}, \vec{b})\}$
until $|Y| < n_{min}$ *or* $|L| = k_{max}$ *not enough points on line or sufficient number of lines found*

3) Determination of the line parameters corresponding to the highest voted accumulator cell.

4) Finding all points $Y \subseteq X$ close (i.e., distance less than cell width) to the line.

5) Determination of the optimal line going through Y with an orthogonal least squares fit.

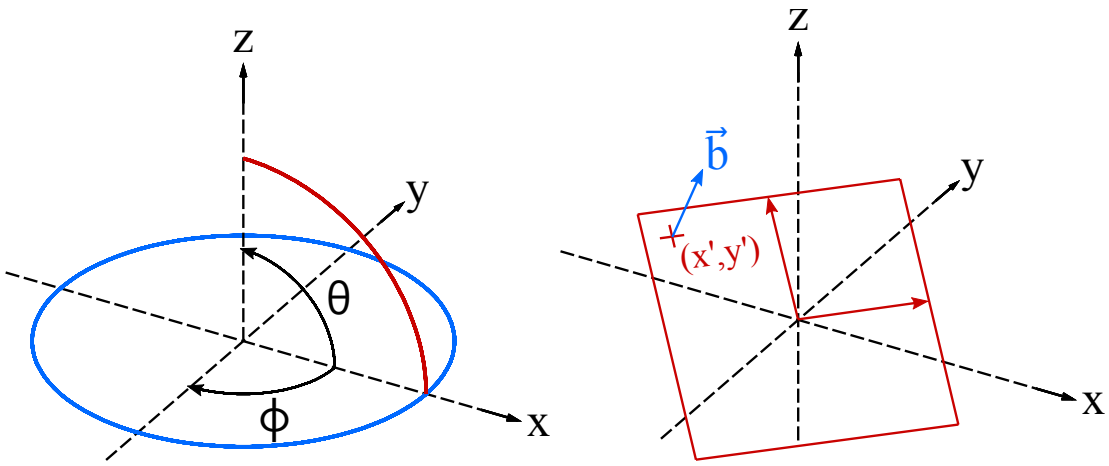
6) Finding all points from X close to the fitted line and their removal from X and from the accumulator array.

7) Repetition of steps 2 to 6 until X contains too few points or the specified number of lines has been found.

The algorithm returns the lines obtained from the least squares fits. The individual steps are described in detail in the following subsections.

2.1 Non-redundant Line Representation

The text book line representation is the vector form $\vec{a} + t\vec{b}$, where $\vec{a}$ is a point on the line and $\vec{b}$, with $\|\vec{b}\| = 1$, is the direction of the line. The *direction* of a line can be specified by the two parameters ϕ, for horizontal orientation (*azimuth*), and θ, for altitude (*elevation*) (see Figure 1(a)). The directional



(a) meaning of ϕ and θ (b) meaning of x' and y'

Figure 1: Roberts' line representation with *azimuth* ϕ and *elevation* θ (left) and the intersection point (x', y') between the direction vector $\vec{b}$ and the perpendicular plane crossing the origin (right).

vector $\vec{b}$ is obtained from θ and ϕ through

$$\vec{b} = \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} \cos\phi\cos\theta \\ \sin\phi\cos\theta \\ \sin\theta \end{pmatrix}. \tag{1}$$

Since two anti-parallel directional vectors describe the same line, the vectors $\vec{b}$ have to be confined to a half-space for the representation to be unique. This can be achieved by restricting the angle ranges to $0 \leq \theta \leq \frac{\pi}{2}$ and $-\pi < \phi \leq \pi$. The remaining redundancy due to anti-parallel vector pairs in the (x, y)-plane ($b_z = 0$) is removed with the restrictions $b_y \geq 0$ if $b_z = 0$ and $b_x = 1$ if $b_y = b_z = 0$.

When the *position* of the line is represented by an arbitrary anchor point, this leads to three parameters, one of which is redundant. Roberts' optimal line representation [6] is one way to remove this redundancy. Roberts first defined a plane which passes through the origin and is perpendicular to the line. The two parameters x' and y' are then defined as the coordinates of the intersection of the line and the plane in the plane's own 2D coordinate frame (see Figure 1(b)). From an arbitrary point $\vec{p} = (p_x, p_y, p_z)$ on the line, the parameters x' and y' are obtained with

$$x' = \left(1 - \frac{b_x^2}{1+b_z}\right) p_x - \left(\frac{b_x b_y}{1+b_z}\right) p_y - b_x p_z \quad \text{and} \quad y' = -\left(\frac{b_x b_y}{1+b_z}\right) p_x + \left(1 - \frac{b_y^2}{1+b_z}\right) p_y - b_y p_z. \tag{2}$$

A point $\vec{p}$ on the line can in turn be obtained with

$$\vec{p} = x' \cdot \begin{pmatrix} 1 - b_x^2/(1+b_z) \\ -b_x b_y/(1+b_z) \\ -b_x \end{pmatrix} + y' \cdot \begin{pmatrix} -b_x b_y/(1+b_z) \\ 1 - b_y^2/(1+b_z) \\ -b_y \end{pmatrix}. \tag{3}$$

2.2 Parameter Space Discretization

For the discretization of the line direction $\vec{b}$, we use the method by Jeltsch et al. [4], which is based on tessellation of Platonic solids. There are only a finite number of Platonic solids, and the solid with the highest number of vertices is the *icosahedron* (see Figure 2(a)). It has 12 vertices, the coordinates of which can be described with the golden section ratio r

$$\mathcal{I} = \{(0, \pm 1, \pm r), (\pm 1, \pm r, 0), (\pm r, 0, \pm 1)\}, \quad \text{with} \quad r = \frac{1}{2}\left(1 + \sqrt{5}\right). \tag{4}$$

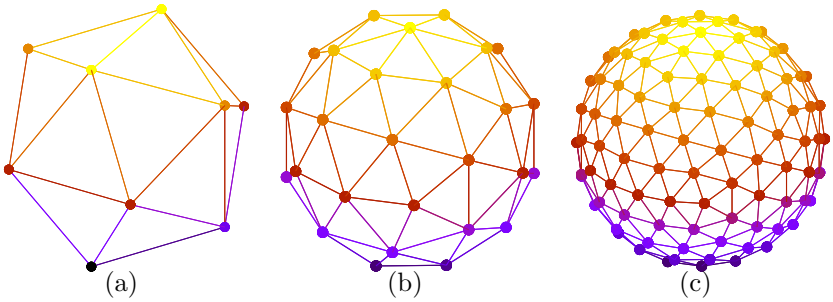


Figure 2: Icosahedron after $0, 1, 2$ tessellation steps.

To obtain more direction vectors than the 12 icosahedron vertices, each triangle on the surface of the icosahedron can be divided into four new ones using polygon triangulation by inserting a new vertex $\vec{w}$ between each pair of adjacent vertices $(\vec{v}_i, \vec{v}_j)$ of the triangle and normalizing its length

$$\vec{w} = (\vec{v}_i + \vec{v}_j) \Big/ \|\vec{v}_i + \vec{v}_j\|. \tag{5}$$

Doing so for all of the polygon triangles results in a new vertex for each edge and three new edges for each new vertex. This can be repeated as often as necessary for the desired level of granularity. We stop after four icosahedron subdivision steps, which leads to 1281 different direction vectors.

For the discretization of the (x', y') plane, it is essential to first translate the point cloud such that the center of its bounding box coincides with the origin. Otherwise, the Hough space would become unnecessarily large with many cells representing lines outside the point cloud bounding box. This centering is done by translating each point as follows

$$\vec{x}_i \leftarrow \vec{x}_i - \vec{c} \quad \text{with} \quad \vec{c} = \frac{1}{2} \begin{pmatrix} x_{max} + x_{min} \\ y_{max} + y_{min} \\ z_{max} + z_{min} \end{pmatrix}. \tag{6}$$

The maximum range of x' and y' then goes from $-\|\vec{d}/2\|$ to $+\|\vec{d}/2\|$, where $\vec{d} = (x_{max} - x_{min}, y_{max} - y_{min}, z_{max} - z_{min})$ is the diagonal of the point cloud bounding box.

2.3 Hough Transform

The transformation of the point cloud into the parameter space described in the preceding subsections is detailed in Algorithm 2. Note that we have included an option to decrease an already filled accumulator array A instead of increasing it for each point (see the last parameter *op*). This is necessary to implement the iterative Hough transform in an efficient way, as explained in the next subsection.

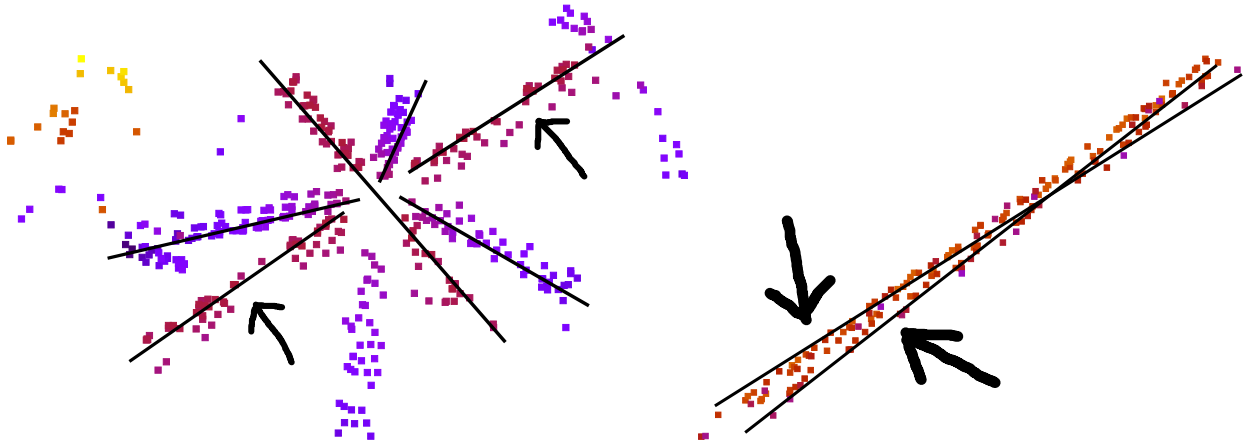
2.4 Iterative Line Detection and Post-processing

While the Hough transform provides a method for transforming a point cloud into a voting array in the parameter space, the problem still remains of how to find the actual lines in this voting array. A common approach is to search for local maxima in this array, also known as a "non maximum suppression" [1]. Whilst this avoids obtaining lines from directly adjacent parameter cells, it can nevertheless yield close by lines separated by cells with lower vote counts. An example from roof ridge detection as presented by Jeltsch et al. in [4] is shown in Figure 3.

We circumvent this problem by only looking for the highest voted cell. The points belonging to the corresponding line are then found and removed from the point cloud. On the remaining point

Algorithm 2: HoughTransform

input : point cloud $X = \{\vec{x}_1, \dots, \vec{x}_n\}$, direction vectors $B = \{\vec{b}_1, \dots, \vec{b}_{N_1}\}$,
x'-discretization $X' = \{x'_1, \dots, x'_{N_2}\}$, y'-discretization $Y' = \{y'_1, \dots, y'_{N_3}\}$,
operation *op* ("+" for addition, "−" for subtraction)
output: voting array A of size $N_1 \times N_2 \times N_3$
if *op* = "+" **then** $A(\vec{b}_i, x'_j, y'_k) \leftarrow 0$ for all $\vec{b}_i, x'_j, y'_k$
for $\vec{x} \in X$ **do**
  for $\vec{b}_i \in B$ **do**
    $(x', y') \leftarrow$ computed after Equation (2)
    $(x'_j, y'_k) \leftarrow$ nearest neighbor to (x', y') from $X' \times Y'$
    if *op* = "+" **then** $A(\vec{b}_i, x'_j, y'_k) \leftarrow A(\vec{b}_i, x'_j, y'_k) + 1$
    else $A(\vec{b}_i, x'_j, y'_k) \leftarrow A(\vec{b}_i, x'_j, y'_k) - 1$



(a) one line detected as two (b) another split line

Figure 3: Finding maxima with a non-maximum suppression as done in [4] can lead to multiple nearby detected lines (images from [4]).

cloud, the Hough transform is applied again, which results in an iterative scheme for finding lines. A naïve application of this scheme by recomputing the Hough transform at each step would result in a runtime that is quadratic in the number of points. This can be reduced to linear runtime, however, by keeping the accumulator array between the different steps, such that the accumulator array only needs to be *decreased* for the removed points. This way, each point from the point cloud is subject to a Hough transform only twice: once for the initial build up of the accumulator array, and a second time for its removal from the array. This is the reason why we introduced the input parameter *op* in Algorithm 2. The criterion for a point $\vec{x}$ belonging to a line given in vector form $\vec{a} + t\vec{b}$ is that its perpendicular distance to the line is less than the cell width dx in the (x', y') plane of the Hough space

$$\|\vec{x} - (\vec{a} + t\vec{b})\| \leq dx \quad \text{with,} \quad t = \langle \vec{b}, \vec{x} - \vec{a} \rangle. \tag{7}$$

Another problem of the Hough transform is its inherent inaccuracy due to the discretization of the parameter space. This inaccuracy cannot be overcome by a finer granularity of the parameter space discretization, because when the parameter cell width becomes too small, points belonging to the same line eventually vote for different lines. To overcome this problem, we collect the points belonging to the line with the most votes and compute a new line with an orthogonal least squares fit through these points. This leads to a high accuracy while at the same time allowing for sufficient

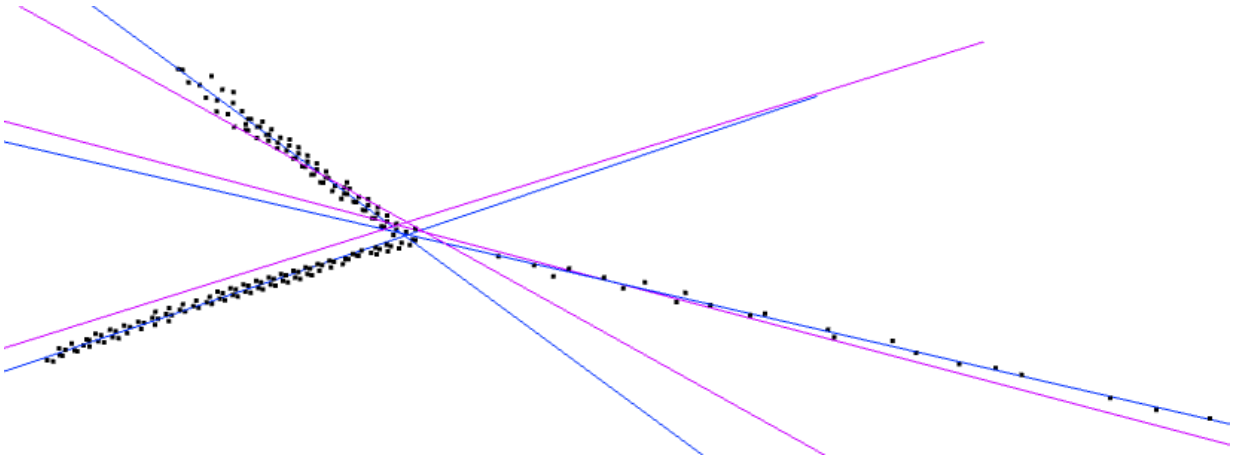


Figure 4: Accuracy improvement over the original lines returned by the Hough transform (magenta) by orthogonal least squares fitted lines (blue)

tolerance in the granularity of the Hough space. Figure 4 shows an example of the improvements achieved with the least squares fit.

Our problem of fitting a line through 3D points $\vec{x}_1, \ldots, \vec{x}_n$ is a special case of orthogonal least squares fitting of linear manifolds, which was studied by Späth in [7]. He showed that an anchor point $\vec{a}$ of the fitted line is given by the center of mass

$$\vec{a} = \frac{1}{n} \sum_{i=1}^{n} \vec{x}_i, \tag{8}$$

and that the direction vector $\vec{b}$ can be determined by two mathematically equivalent methods:

1) $\vec{b}$ is the right singular vector $\vec{v}_1$ corresponding to the largest singular value $s_1 \geq \ldots \geq s_n$ in the singular value decomposition (SVD) $Q = USV^T$ of the matrix built from the centered data points
$$Q^T = (\vec{q}_1, \ldots, \vec{q}_n) \quad \text{with} \quad \vec{q}_i = \vec{x}_i - \vec{a}, \tag{9}$$

2) $\vec{b}$ is the eigenvector corresponding to the largest eigenvalue of Q^TQ. Q^TQ is identical to the scatter matrix, or $(n-1)$ times the covariance matrix of the data points $\vec{x}_1, \ldots, \vec{x}_n$. Thus, $\vec{b}$ is simply the principal component obtained from a principal component analysis (PCA) [9].

The scatter matrix Q^TQ is a 3×3 matrix, while Q is a $n \times 3$ matrix. One should therefore expect method 2 to be much faster. On the other hand, the scatter matrix needs to be computed first, which is an $O(n)$ operation. It is thus not directly obvious which method is faster. To compare both methods, we created a point cloud with $n = 10^6$ points randomly varying around a straight line. On this point cloud, we tested the performance of LAPACK[3] with both methods, as they were wrapped in the functions *svd* and *eigen* of the R language for statistical computing[4]. The PCA (method 2) was more than hundred times faster than the SVD (method 1), so that we settled for method 2 in our implementation of the orthogonal least squares fit.

3 Time and Space Complexity

The runtime of the Hough transform in Algorithm 2 is $O(nN_1)$, where $n = |X|$ is the number of points in the point cloud X, and N_1 is the number of directions B, i.e., $N_1 = 1281$ in our implementation.

[3] http://www.netlib.org/lapack/
[4] https://www.r-project.org/

The Hough transform is called once with all n points, and then in the i-th iteration with n_i points whereby $\sum n_i \leq n$. The total runtime of all Hough transform calls is thus $O(2nN_1)$.

In the worst case, the repeat-loop is run n/n_{min} times. The search for points from X belonging to the line is run twice and thus requires in total $O(2n)$ comparisons. The PCA in the orthogonal least squares fit has a constant runtime, but the computation of the covariance matrix requires $O(n)$ elementary arithmetic operations. Algorithm 1 thus has a (note that $N_1 = 1281$ in our case)

$$\text{runtime complexity} \in O(2nN_1 + 3n^2/n_{min}). \tag{10}$$

The space complexity results from the storage requirements of the point cloud X and the accumulator array A. Note that the orthogonal least squares fit does not contribute to the space complexity because the covariance matrix is of size 3×3. When the three coordinates of each point are stored as *double*, the point cloud space requirement is $3n \times$ sizeof(double), where $n = |X|$ is the number of points. When the counters in the accumulator array A are of type *int*, its space requirement is $N_1 \times N_2 \times N_3 \times$ sizeof(int), where $N_1 = 1281$ is the number of directions and $N_2 = N_3 = 2\|\vec{c}\|/dx$, with $\vec{c}$ given by Equation (6), is the number of subdivisions in each direction of the $x'y'$ plane. Algorithm 1 thus has a

$$\text{space complexity} \in O(3n \times \text{sizeof(double)} + N_1 \times N_2 \times N_3 \times \text{sizeof(int)}). \tag{11}$$

4 Cases of Failure

The algorithm can fail for three reasons:

- there is not enough memory for the point cloud or the Hough space accumulator array;
- overflow in an accumulator array counter;
- the point cloud contains many identical points, which unluckily lead in one iteration step to a point cloud with *all* points identical.

As explained in Section 3, the memory requirement of the algorithm is $3n \times$ sizeof(double) $+ N_1 \times N_2 \times N_3 \times$ sizeof(int). The algorithm can thus run out of memory because there are too many points in the cloud, or because the granularity of the Hough space is too fine so that it has too many cells. As N_1 is constantly set to $N_1 = 1281$ in our implementation, the size of the Hough space only depends on $N_2 = N_3 = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2 + (z_{max} - z_{min})^2}/dx$.

Another problem could be overflow of a Hough cell because too many points belong to the corresponding line. As the cell counters are of type *int*, overflow cannot occur when the number of points is less than $2^{8 \cdot \text{sizeof(int)} - 1}$.

When, during one iteration of the repeat-loop in Algorithm 1, the point cloud X consists solely of points with the same coordinates, its covariance matrix is zero and has no non-zero eigenvector. The orthogonal least squares fit will then yield no line direction $\vec{b}$ and the algorithm must stop. When this occurs in the first step, no line is returned at all.

5 Online Demo

An online demo for this algorithm is provided on the web page of this article[5]. The user can choose among several pre-uploaded data sets or upload his own point cloud data in the following file format:

[5] https://doi.org/10.5201/ipol.2017.208

- 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 commas, 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
```

When the point cloud data has been selected, the following parameters can be chosen:

xy step width (*dx*). The Hough space cell width in the $x'y'$ plane. When $dx = 0$ (default), it is automatically set to

$$dx = \frac{\sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2 + (z_{max} - z_{min})^2}}{64},$$

which results in $64 \times 64 = 2^{12}$ cells in the $x'y'$ plane. When dx is chosen such that it results in more than 10^8 Hough cells, an error is thrown.

maximum number of lines (*nlines*). The maximum number of lines returned. When $nlines = 0$ (default), all lines are returned.

minimum vote count (*minvotes*). Lines with less than *minvotes* are not returned. When $minvotes < 2$ (default), all lines are returned.

The detected lines are returned in vector form with the following information:

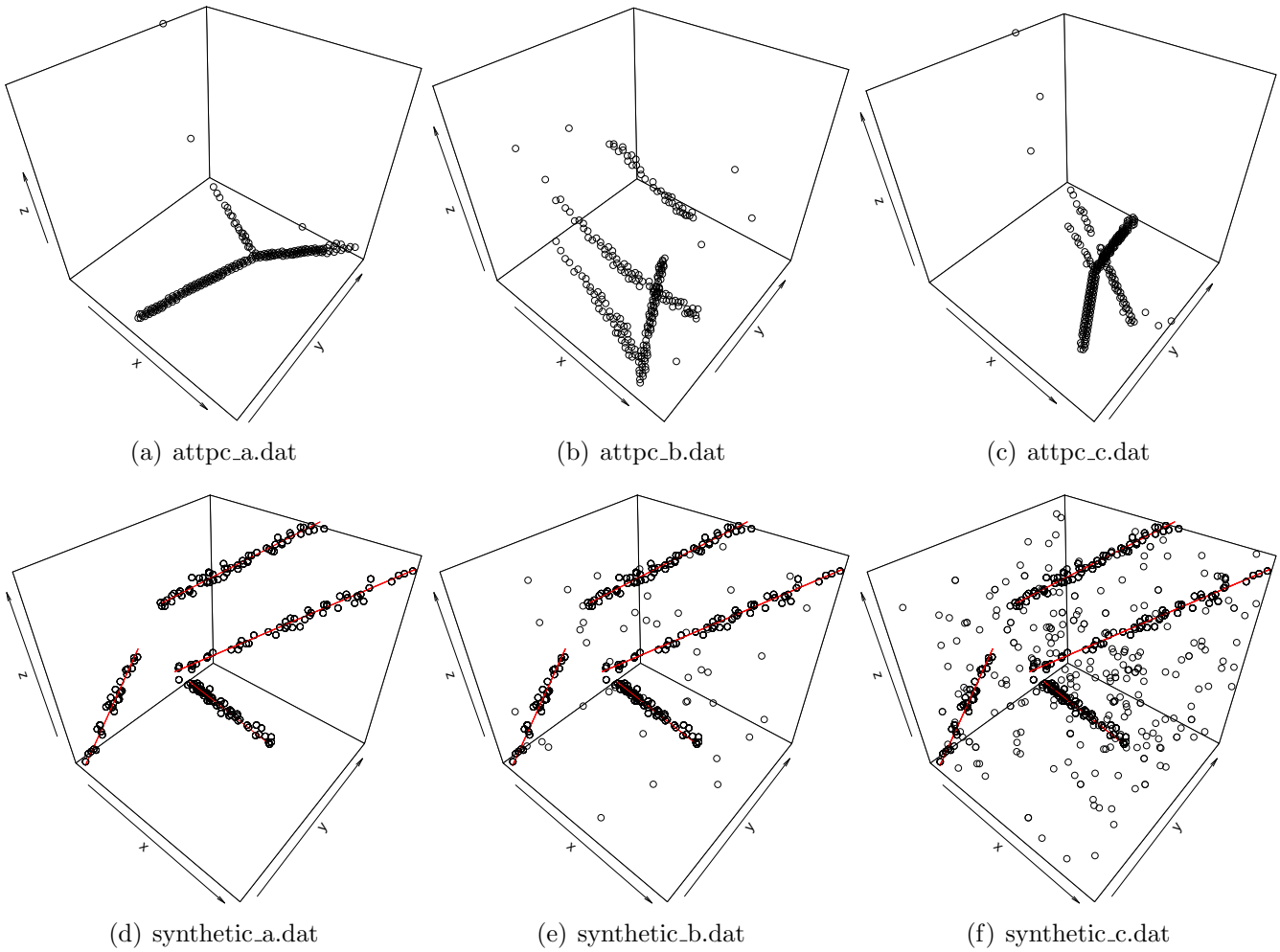
`npoints=`n`, a=(`a_x`,`a_y`,`a_z`), b=(`b_x`,`b_y`,`b_z`)`

where n is the number of points that have been assigned to the line after the least-squares-fit, $\vec{a} = (a_x, a_y, a_z)$ is the anchor point given by Equation (8), and $\vec{b} = (b_x, b_y, b_z)$ is the line direction with $\|\vec{b}\| = 1$. The detected lines are returned in decreasing order of the Hough space vote counts (but not necessarily in decreasing order of *npoints*[6]) as in the following example:

```
npoints=58, a=(2.5682,-123.8986,0.3456), b=(0.0000,0.7071,0.7071)
npoints=32, a=(-2.4576,34.8665,0.0000), b=(0.7071,0.0000,0.7071)
npoints=3, a=(7.0056,-12.7867,8.5634), b=(0.5773,0.5773,0.5773)
```

6 Reference Data Sets

As supplementary material, we provide six data sets, of which three are real 3D sensor data and three are synthetically generated point clouds. The real data stem from the active target time projection chamber (AT-TPC) at the National Superconducting Cyclotron Laboratory, Michigan State University [8]. We have chosen these data because they are from a real use case and do not require any pre-processing. Moreover, the iterative Hough transform, as presented in this article, is going to be integrated into a data analysis framework for AT-TPC experiments[7].



(a) attpc_a.dat (b) attpc_b.dat (c) attpc_c.dat

(d) synthetic_a.dat (e) synthetic_b.dat (f) synthetic_c.dat

Figure 5: The data sets provided with the online demo. For the synthetic data, the ground truth lines are drawn, too, in red.

6.1 Data Set Description

The real data files show the following characteristics with $dx = 15$:

attpc_a.dat has 206 points, of which 203 belong to three lines:

```
npoints=111, a=(25.51,-42.76,474.42), b=(0.2890, 0.5544, 0.7805)
npoints=70, a=(92.38,64.98,521.05), b=(0.5072, 0.6880, -0.5191)
npoints=22, a=(33.47,9.66,744.58), b=(0.1187, 0.0389, -0.9922)
```

attpc_b.dat has 238 points, of which 231 belong to four lines:

```
npoints=77, a=(52.43,52.83,459.86), b=(0.5690, -0.5415, 0.6188)
npoints=70, a=(69.56,28.17,1018.70), b=(0.1262, 0.0649, -0.9899)
npoints=48, a=(55.52,9.98,812.80), b=(0.1162, 0.0202, -0.9930)
npoints=36, a=(75.60,38.14,1496.36), b=(0.1101, 0.0709, -0.9914)
```

attpc_c.dat has 255 points, of which 248 belong to four lines:

[6] After the least-squares-fit, all points close to the line are collected, which generally yields a different set of points than those that have voted for the Hough cell.

[7] https://github.com/ATTPC/ATTPCROOTv2

```
npoints=115, a=(26.48,82.03,487.00), b=(0.1679, -0.5659, 0.8072)
npoints=70, a=(76.53,-38.88,558.03), b=(0.4430, -0.7577, -0.4791)
npoints=51, a=(70.98,11.14,657.54), b=(0.1189, 0.0144, -0.9928)
npoints=12, a=(29.04,7.79,735.20), b=(0.1261, 0.0384, -0.9913)
```

The synthetic data has been created by randomly sampling points on predefined lines with the x, y, and z coordinates randomly shuffled by a normal distribution with zero mean. Afterwards, uniform random noise points have been added. The R script *generate_synthetic_data.r* used for generating the data files is provided with the supplementary material.

The synthetic data files work well with $dx = 0.5$ and have the following characteristics:

synthetic_a.dat has 200 points belonging to four lines:

```
npoints=60, a=(0.0000,3.0000,3.0000), b=(0.5774,0.5774,0.5774)
npoints=60, a=(1.0000,0.0000,-1.0000), b=(-0.5774,0.5774,-0.5774)
npoints=50, a=(2.0000,2.0000,2.0000), b=(0.5774,0.5774,0.5774)
npoints=30, a=(-1.0000,-3.0000,-1.0000), b=(0.0000,0.7071,0.7071)
```

synthetic_b.dat is identical to *synthetic_a.dat*, but with 50 noise points added.

synthetic_c.dat is identical to *synthetic_a.dat*, but with 200 noise points added.

Plots of the all point clouds are shown in Figure 5.

6.2 Hough Transform Results on the Data Sets

In the real data sets, not all points belong to lines from particle trajectories, but with $dx = 15$ and $minvotes = 3$ all lines are correctly returned. This is achieved with the following command line:

```
hough3dlines -dx 15 -minvotes 3 attpc_c.dat
```

When no lower bound to the number of votes is given, additional lines containing only two points are returned, which are shown in blue in Figure 6. When dx becomes too large, lines are missed and erroneous lines are returned as can be seen in the second row of Figure 6 for $dx = 40$. This is because too many points not belonging to line are collected within a distance dx.

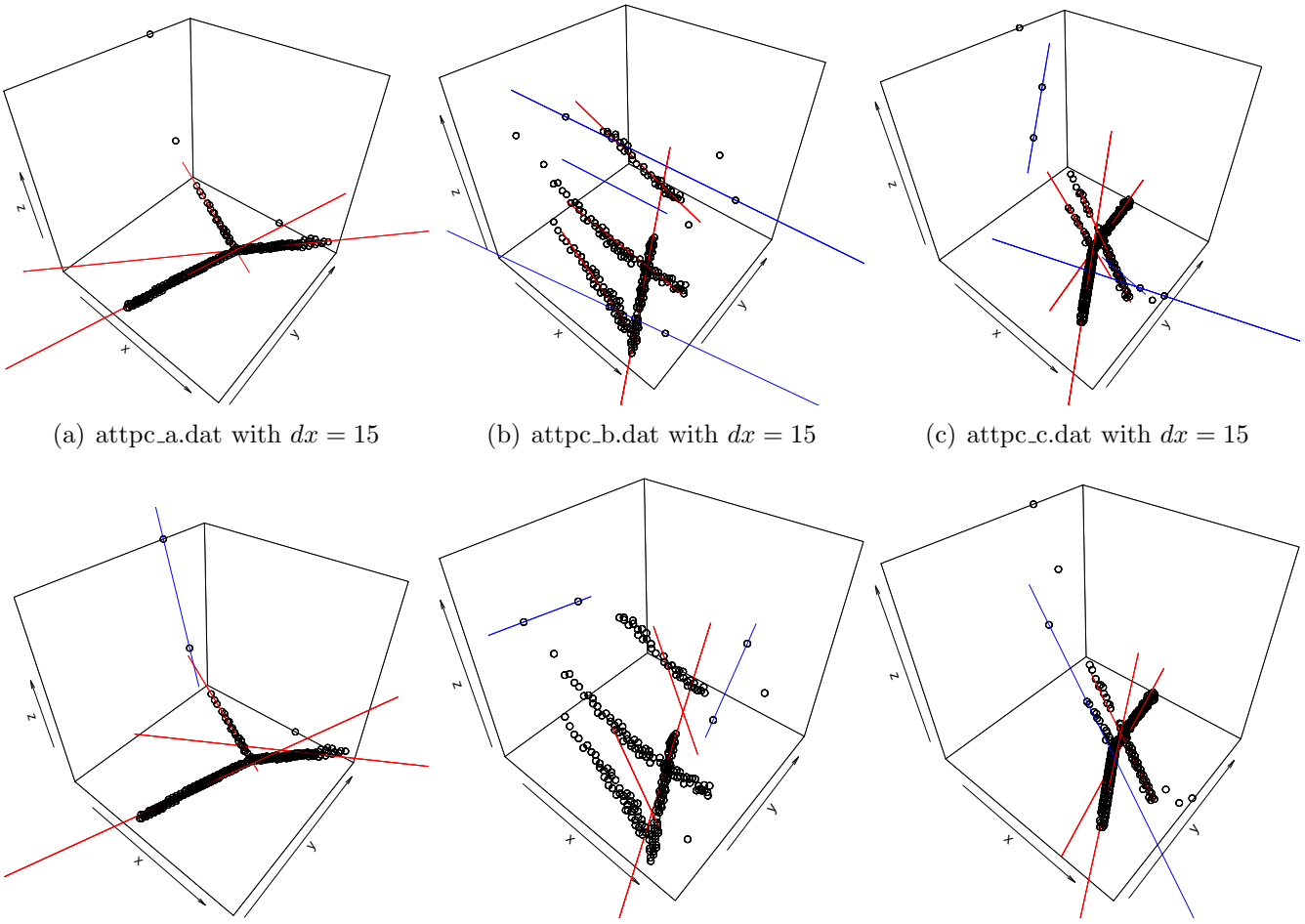
For the synthetic data with many noise points (*synthetic_c.dat*) and $dx = 0.5$, in total 37 lines are found, when no restriction on the minimum vote count is made. The correct four lines are found, however, for $minvotes > 10$ with the following command line:

```
hough3dlines -dx 0.5 -minvotes 20 synthetic_c.dat
```

When dx is set too small, the same problem shown in Figure 3 occurs: only four lines are found, which all have less than 20 votes and three of these lines belong to the same ground truth line (see Figure 7(a)). For the choice $dx = 1.0$, the four highest voted lines still correspond to the ground truth lines, but the next highest votes line has 23 votes.

7 Conclusion

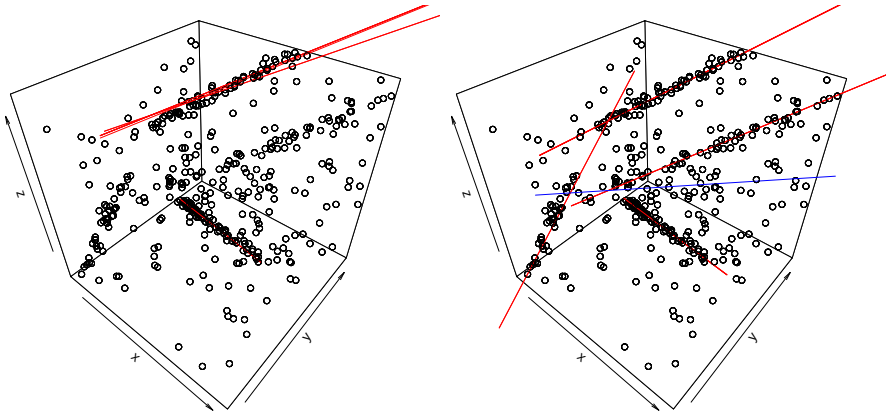
The iterative Hough transform is an algorithm that can find lines even in noisy images. The final orthogonal least squares fit in our implementation overcomes the accuracy limitation that is inherent to the Hough transform due to parameter space quantization. As the examples in Section 6.2 show, it is however important that the spread dx of the points around the lines is approximately specified as an input parameter.



(a) attpc_a.dat with $dx = 15$ (b) attpc_b.dat with $dx = 15$ (c) attpc_c.dat with $dx = 15$

(d) attpc_a.dat with $dx = 40$ (e) attpc_b.dat with $dx = 40$ (f) attpc_c.dat with $dx = 40$

Figure 6: Results of the algorithm on the real data sets with $dx = 15$ and $dx = 40$. Red lines are lines with more than two votes. Only for an appropriate choice of dx, the results are good.



(a) synthetic_c.dat with $dx = 0.1$ (b) synthetic_c.dat with $dx = 1.0$ and $minvotes = 20$

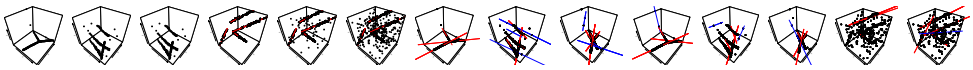
Figure 7: Results of the algorithm on the synthetic data with noise for $dx = 0.1$ and $dx = 1$. The blue line does not correspond to a ground truth line.

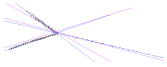
Acknowledgment

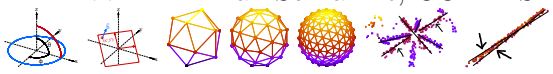
The authors are grateful to Yassid Ayyad Limonge for providing sample data from the active target time projection chamber (AT-TPC) at the National Superconducting Cyclotron Laboratory, Michi-

gan State University.

Image Credits


Christoph Dalitz, CC-BY-SA


Tilman Schramke, CC-BY-SA


Manuel Jeltsch, CC-BY-SA

References

[1] W. Burger and M.J. Burge, *Principles of Digital Image Processing - Core Algorithms*, Springer, London, 2009.

[2] J. Carter, K. Schmid, K. Waters, L. Betzhold, B. Hadley, R. Mataosky, and J. Halleran, *Lidar 101: An introduction to LIDAR technology, data, and applications*, National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center. Charleston, SC, (2012).

[3] P.V.C. Hough, *Method and means for recognizing complex patterns*, 1962. US Patent 3,069,654.

[4] M. Jeltsch, C. Dalitz, and R. Pohle-Fröhlich, *Hough parameter space regularisation for line detection in 3D*, in International Conference on Computer Vision Theory and Applications (VISAPP), 2016, pp. 345–352.

[5] P. Mukhopadhyay and B.B. Chaudhuri, *A survey of Hough transform*, Pattern Recognition, 48 (2015), pp. 993–1010.

[6] K.S. Roberts, *A new representation for a line*, in Computer Vision and Pattern Recognition CVPR'88, 1988, pp. 635–640.

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

[8] D. Suzuki, M. Ford, D. Bazin, W. Mittig, W.G. Lynch, T. Ahn, S. Aune, E. Galyaev, A. Fritsch, J. Gilbert, F. Montes, A. Shore, J. Yurkon, J.J. Kolata, J. Browne, A. Howard, A.L. Roberts, and X.D. Tang, *Prototype AT-TPC: Toward a new generation active target time projection chamber for radioactive beam experiments*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 691 (2012), pp. 39–54.

[9] A. Webb, *Statistical Pattern Recognition*, John Wiley & Sons, 2 ed., 2002.