Uncertainties:
Representation and Propagation
&
Line Extraction from Range data

Uncertainty Representation

Section **4.1.3** of the book

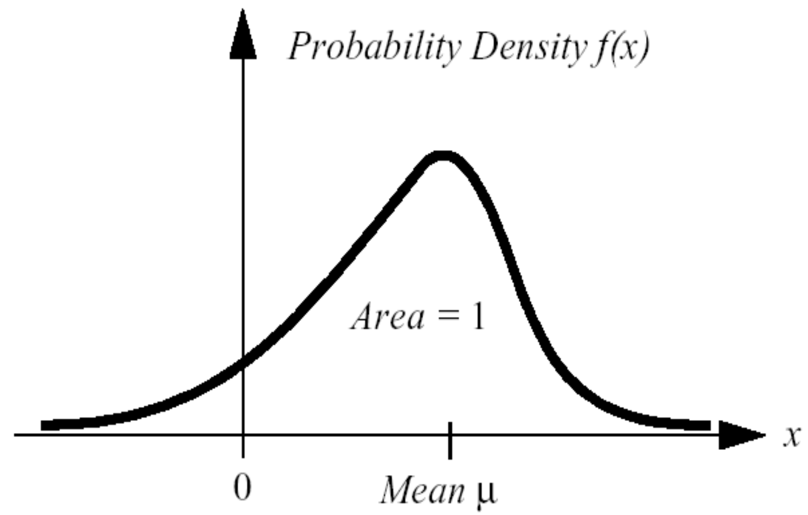
- Sensing in the real world is **always uncertain**
  - How can uncertainty be **represented** or quantified?
  - How does uncertainty **propagate**?

    fusing uncertain inputs into a system, what is the resulting uncertainty?
  - What is the merit of all this for mobile robotics?

Uncertainty Representation (2)

- Use a Probability Density Function (PDF) to characterize the statistical properties of a variable X:

$$\int_{-\infty}^{\infty} f(x)dx = 1$$



- Expected value of variable X:

$$\mu = E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

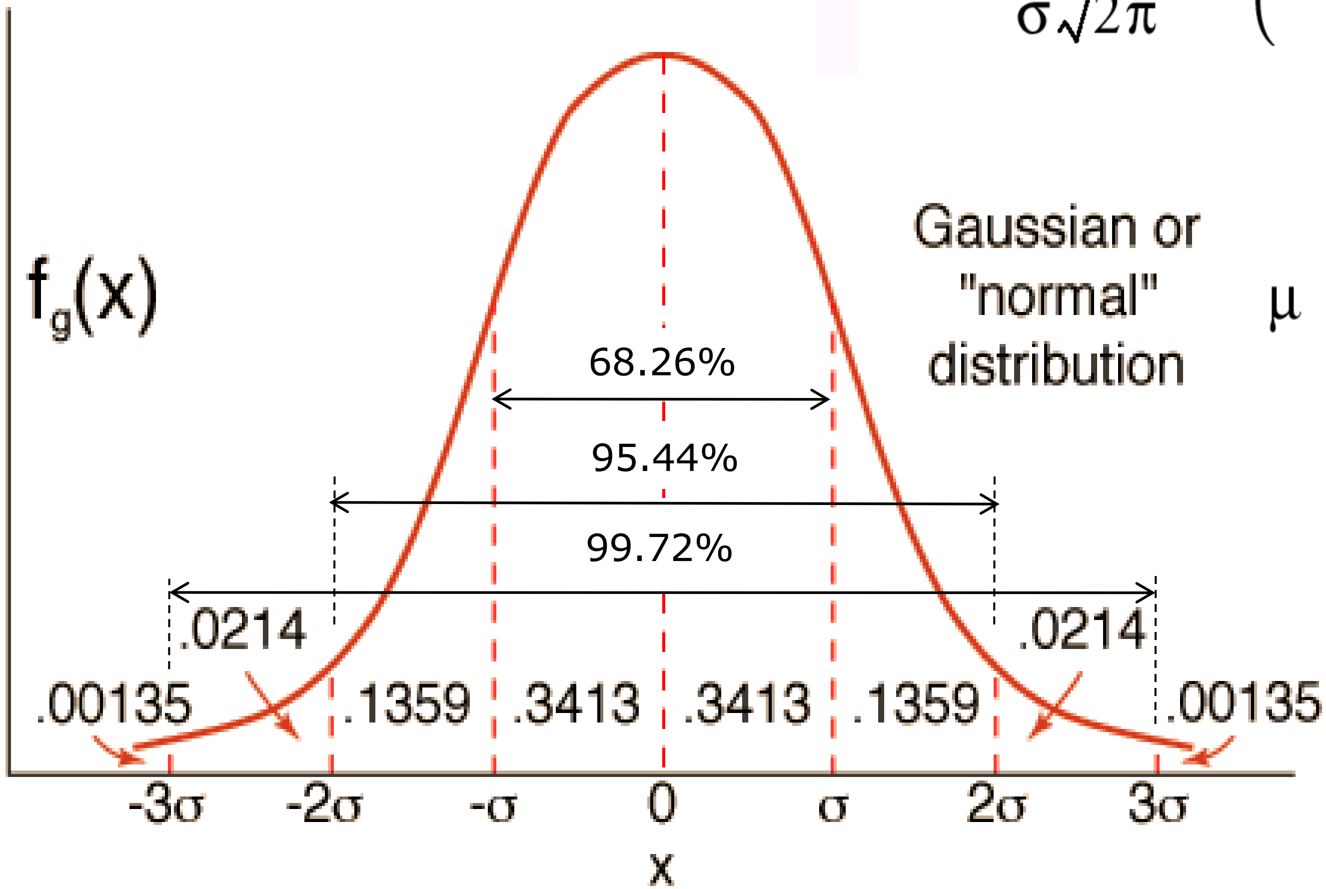
- Variance of variable X

$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx$$

Gaussian Distribution

- Most common PDF for characterizing uncertainties: Gaussian

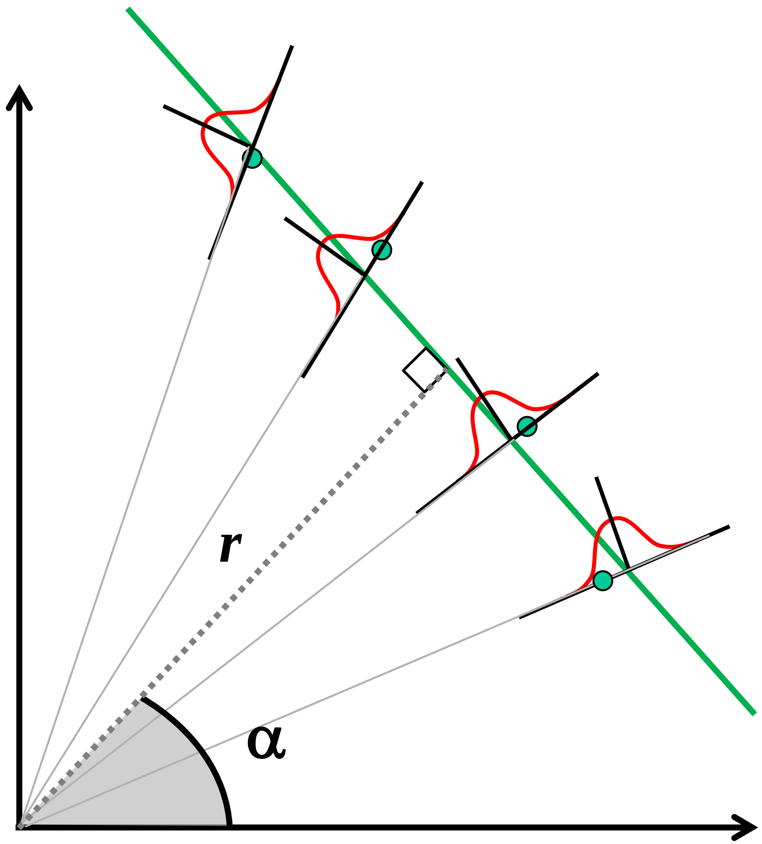
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$



$\mu = 0$ and $\sigma = 1$

The Error Propagation Law

- Imagine extracting a line based on point measurements with uncertainties.

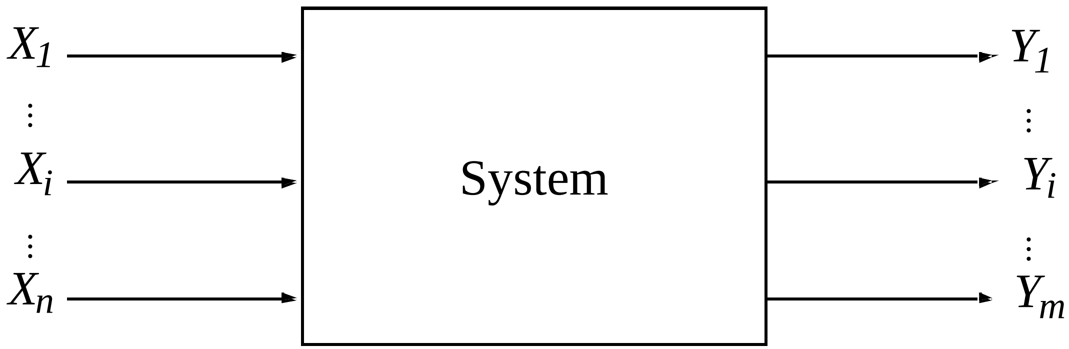


- Model parameters in polar coordinates [(r, α) uniquely identifies a line]

- The question:
  - What is the uncertainty of the extracted line knowing the uncertainties of the measurement points that contribute to it ?

The Error Propagation Law

Error propagation in a multiple-input multi-output system with *n* inputs and *m* outputs.

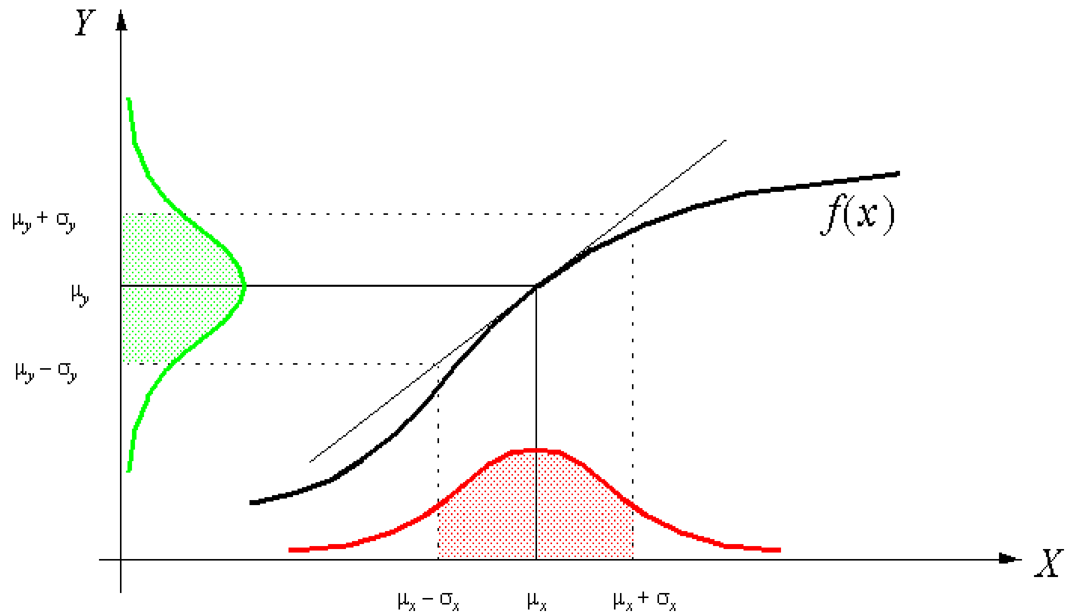


$$Y_j = f_j(X_1 \ldots X_n)$$

The Error Propagation Law

- 1D case of a nonlinear error propagation problem
- It can be shown that the output covariance matrix C_Y is given by the error propagation law:

$$C_Y = F_X C_X F_X^T$$

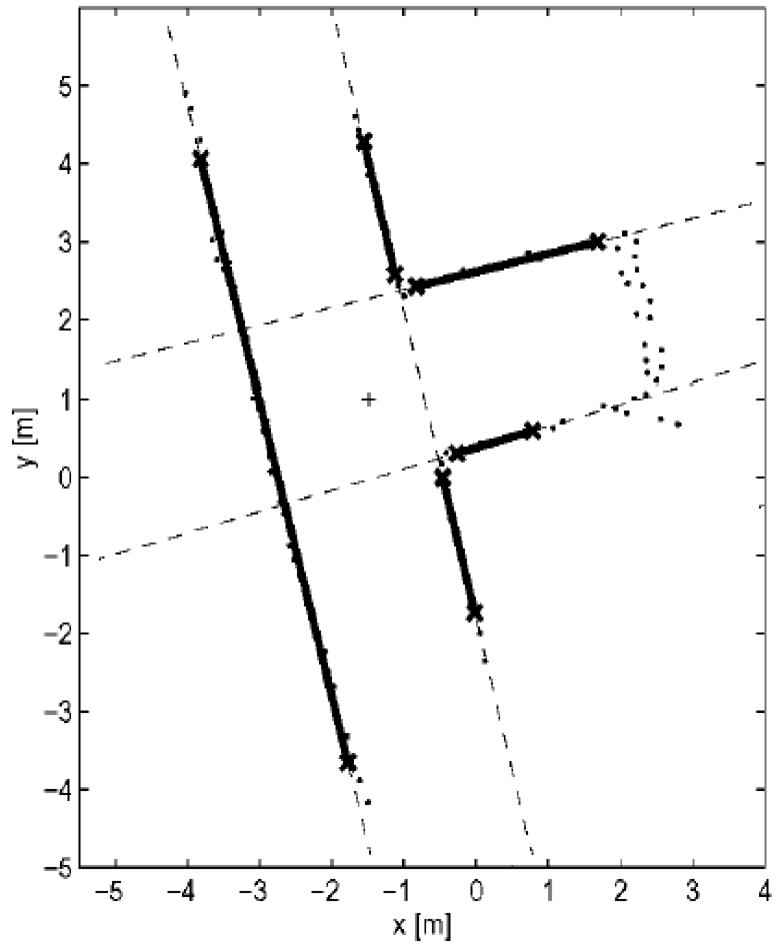


- where
  - C_X: covariance matrix representing the input uncertainties
  - C_Y: covariance matrix representing the propagated uncertainties for the outputs.
  - F_X: is the ***Jacobian*** matrix defined as:

$$F_X = \begin{bmatrix} \frac{\partial f_1}{\partial X_1} & \cdots & \frac{\partial f_1}{\partial X_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial f_m}{\partial X_1} & \cdots & \frac{\partial f_m}{\partial X_n} \end{bmatrix}$$

  - which is the transpose of the gradient of f(X).

Example: line extraction from laser scans



Line Extraction (1)

- Point-Line distance

$$\rho_i \cos(\theta_i - \alpha) - r = d_i$$

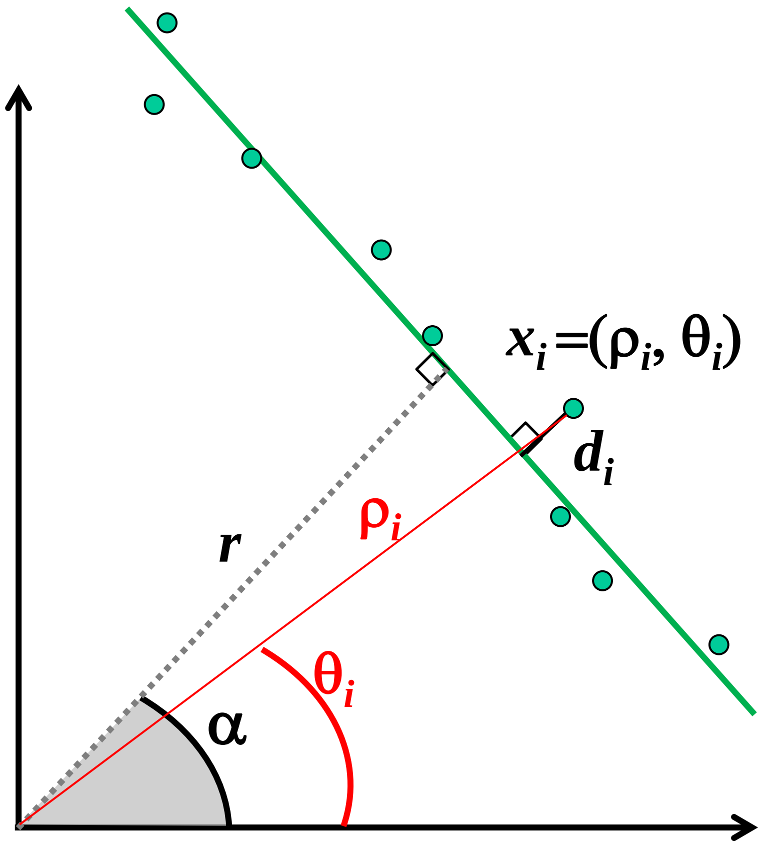
- If each measurement is equally uncertain then sum of sq. errors:

$$S = \sum_i d_i^2 = \sum_i (\rho_i \cos(\theta_i - \alpha) - r)^2$$

- Goal: minimize ***S*** when selecting (***r***, **α**)
  ⇨ solve the system

$$\frac{\partial S}{\partial \alpha} = 0 \qquad \frac{\partial S}{\partial r} = 0$$

- **"Unweighted Least Squares"**



Line Extraction (2)

- Point-Line distance

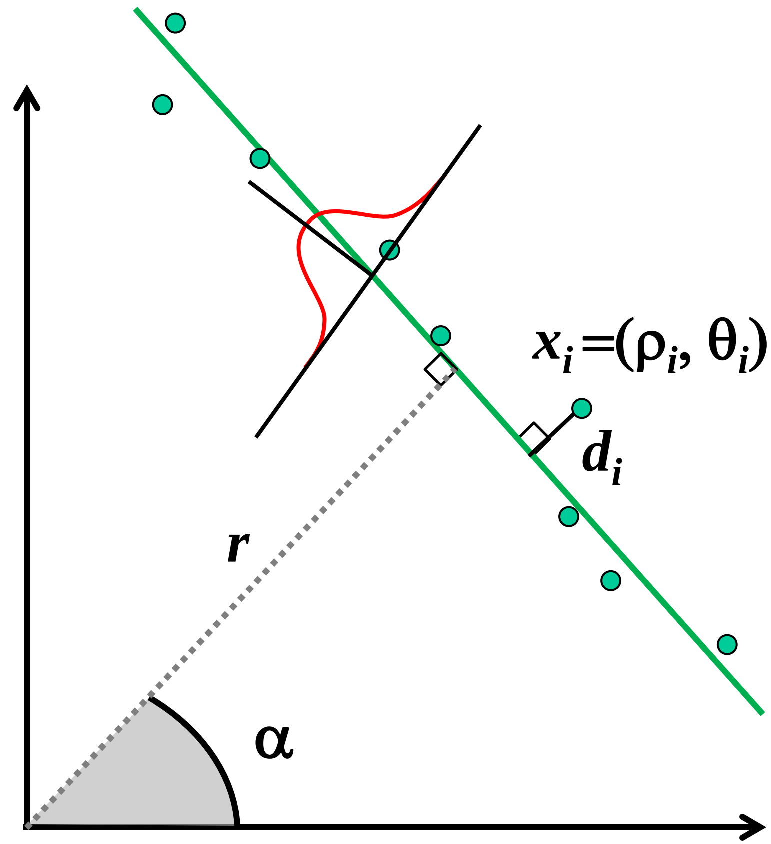
$$\rho_i \cos(\theta_i - \alpha) - r = d_i$$

- Each sensor measurement, may have its own, unique uncertainty

$$S = \sum w_i d_i^2 = \sum w_i (\rho_i \cos(\theta_i - \alpha) - r)^2$$

$$w_i = 1/\sigma_i^2$$

- **Weighted Least Squares**



Line Extraction (2)

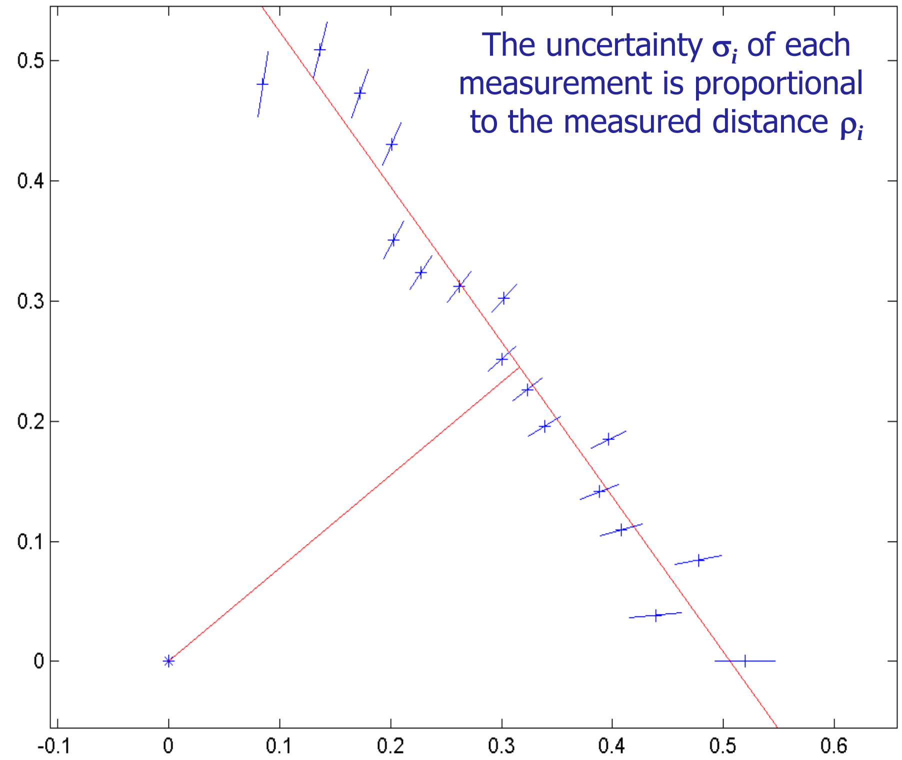
- Weighted least squares and solving the system:

$$\frac{\partial S}{\partial \alpha} = 0 \qquad \frac{\partial S}{\partial r} = 0$$

- Gives the line parameters:

$$\alpha = \frac{1}{2}\operatorname{atan}\left(\frac{\sum w_i \rho_i^2 \sin 2\theta_i - \frac{2}{\sum w_i}\sum\sum w_i w_j \rho_i \rho_j \cos\theta_i \sin\theta_j}{\sum w_i \rho_i^2 \cos 2\theta_i - \frac{1}{\sum w_i}\sum\sum w_i w_j \rho_i \rho_j \cos(\theta_i + \theta_j)}\right)$$

$$r = \frac{\sum w_i \rho_i \cos(\theta_i - \alpha)}{\sum w_i}$$



- If $\rho_i \sim N(\hat{\rho}_i, \sigma_{\rho_i}{}^2)$, $\theta_i \sim N(\hat{\theta}_i, \sigma_{\theta_i}{}^2)$ what is the uncertainty in the line (r, α) ?

Error Propagation: Line extraction

The uncertainty of **each measurement** $x_i=(\rho_i, \theta_i)$ is described by the covariance matrix:

$$C_{x_i} = \begin{bmatrix} \sigma_{\rho_i}{}^2 & 0 \\ 0 & \sigma_{\theta_i}{}^2 \end{bmatrix}$$

Assuming that ρ_i, θ_i are independent

The uncertainty in the **line** (r, α) is described by the covariance matrix:

$$C_{\alpha r} = \begin{bmatrix} \sigma_{\alpha}{}^2 & \sigma_{\alpha r} \\ \sigma_{r\alpha} & \sigma_{r}{}^2 \end{bmatrix} = ?$$

Define:

$$C_x = \begin{bmatrix} diag(\sigma_\rho{}^2) & 0 \\ 0 & diag(\sigma_\theta{}^2) \end{bmatrix} = \begin{bmatrix} \dots & 0 & 0 & . & 0 & 0 & . \\ . & \sigma_{\rho_i}{}^2 & 0 & . & 0 & 0 & . \\ . & 0 & \sigma_{\rho_{i+1}}{}^2 & . & 0 & 0 & . \\ . & . & . & \dots & . & . & . \\ . & 0 & 0 & . & \sigma_{\theta_i}{}^2 & 0 & . \\ . & 0 & 0 & . & 0 & \sigma_{\theta_i+1}{}^2 & . \\ . & . & . & . & . & . & \dots \end{bmatrix}$$

$2n \times 2n$

Jacobian:

$$F_{\rho\theta} = \begin{bmatrix} \dots & \frac{\partial \alpha}{\partial \rho_i} & \frac{\partial \alpha}{\partial \rho_{i+1}} & \dots & \frac{\partial \alpha}{\partial \theta_i} & \frac{\partial \alpha}{\partial \theta_{i+1}} & \dots \\ \dots & \frac{\partial r}{\partial \rho_i} & \frac{\partial r}{\partial \rho_{i+1}} & \dots & \frac{\partial r}{\partial \theta_i} & \frac{\partial r}{\partial \theta_{i+1}} & \dots \end{bmatrix}$$

From Error Propagation Law

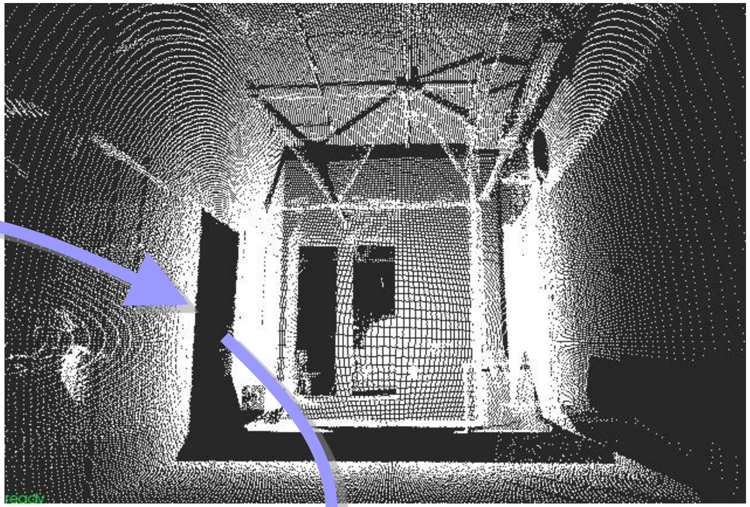
$$C_{\alpha r} = F_{\rho\theta} C_x F_{\rho\theta}{}^T$$

Feature Extraction from Range Data:
Line extraction

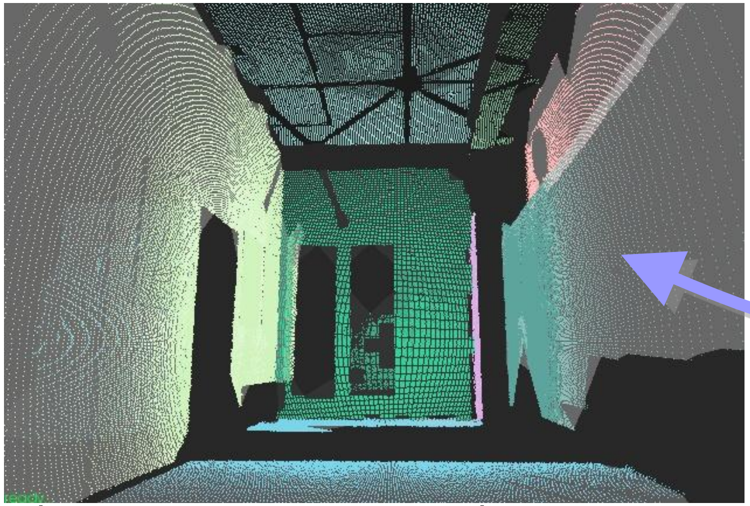
Split and merge
Linear regression
RANSAC
Hough-Transform

Extracting Features from Range Data

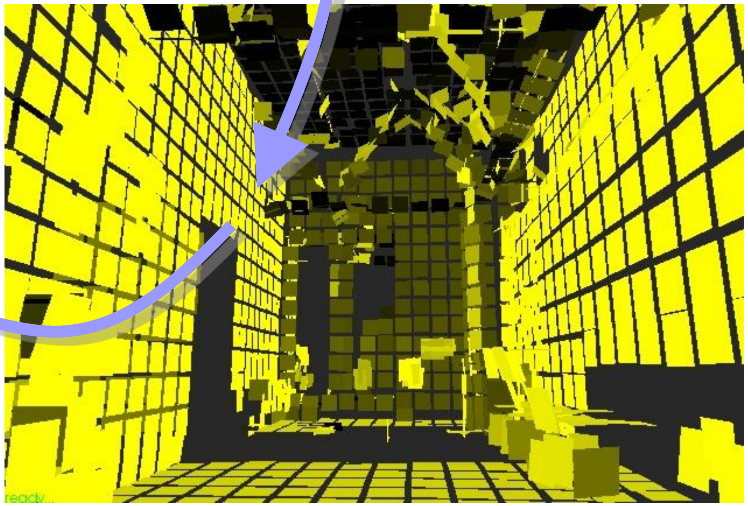
photograph of corridor at ASL



raw 3D scan



plane segmentation result



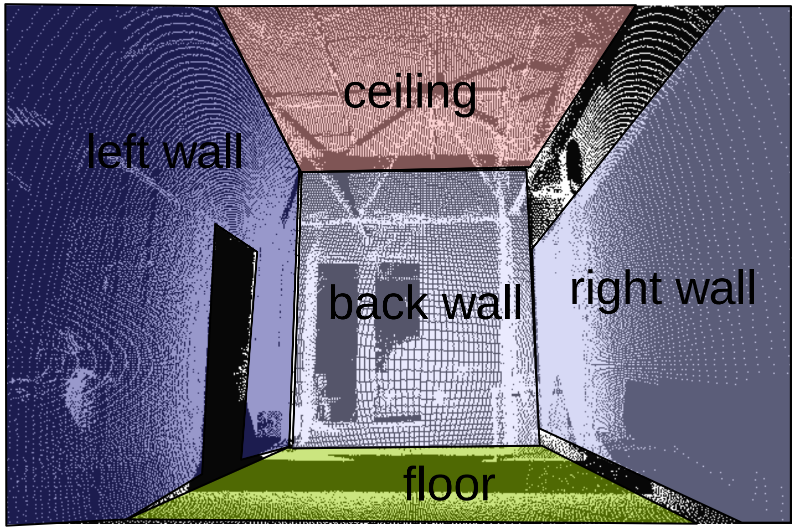
extracted planes for every cube

Extracting Features from Range Data

- goal: extract planar features from a dense point cloud

- example:

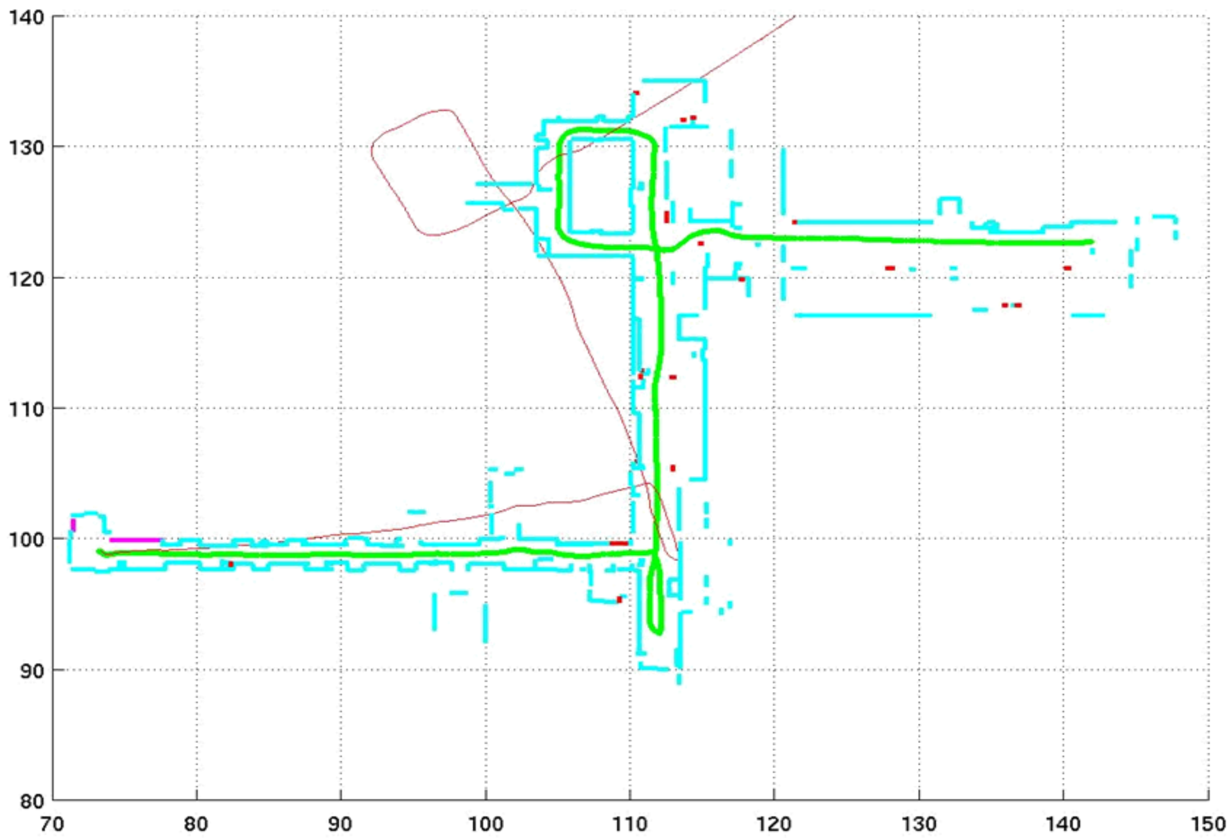
example scene showing a part of a corridor of the lab



same scene represented as dense point cloud generated by a rotating laser scanner

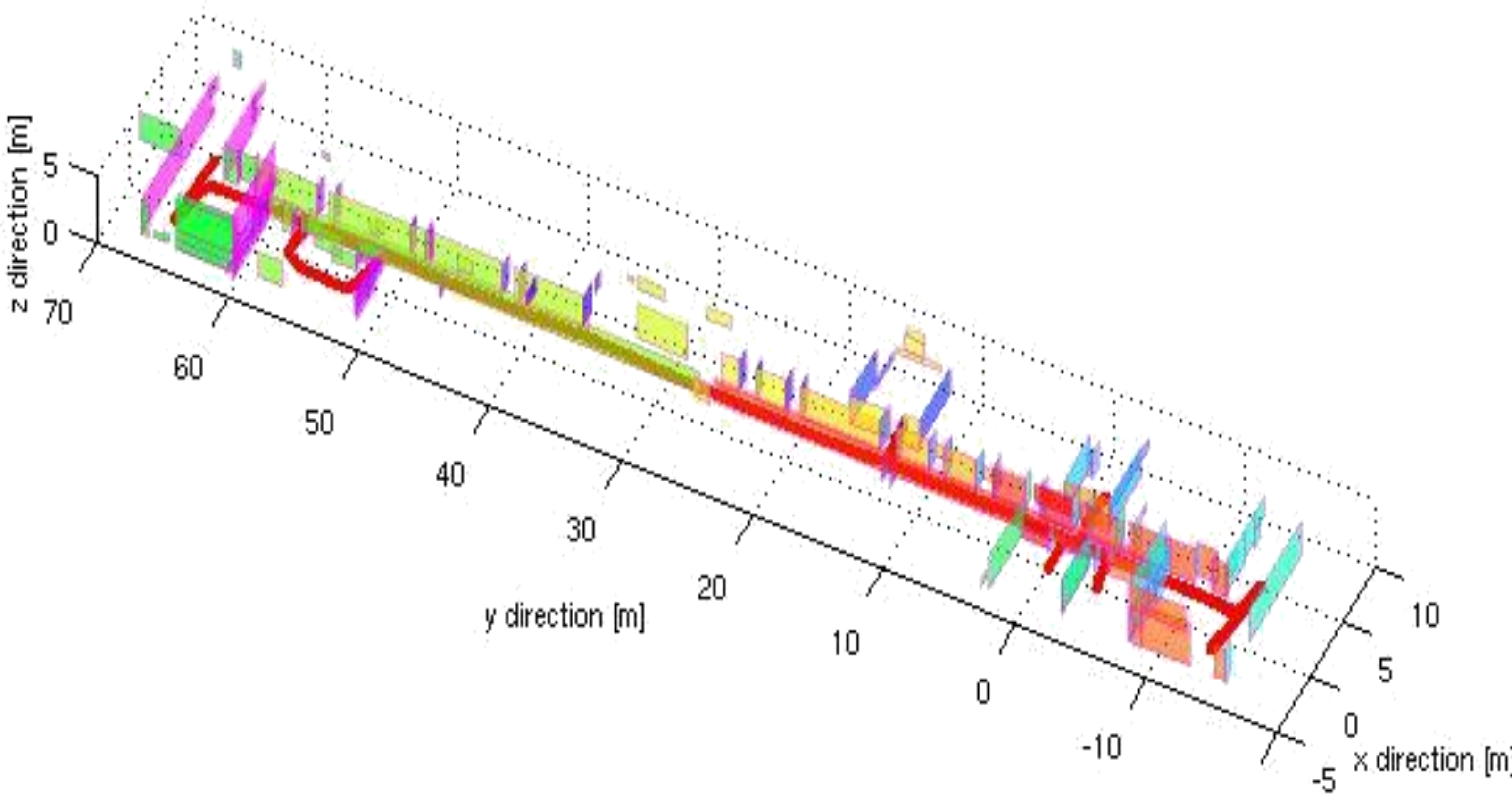
Extracting Features from Range Data

- Map of the ASL hallway built using line segments



Extracting Features from Range Data

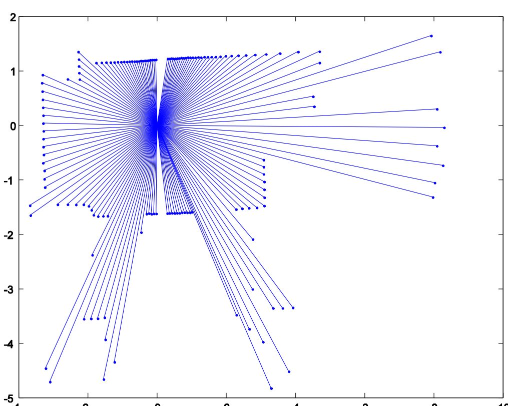
- Map of the ASL hallway built using orthogonal planes constructed from line segments

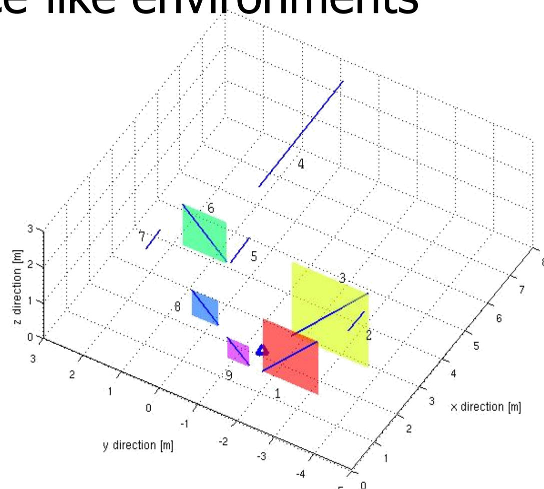


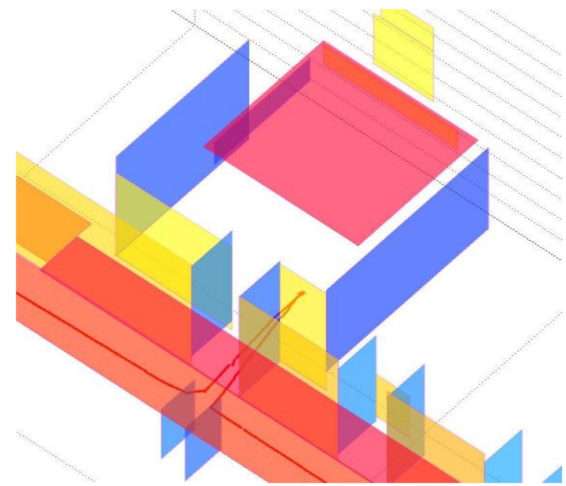
Features from Range Data: Motivation

- Point Cloud ⇨ extract Lines / Planes
- Why Features:
  - Raw data: huge amount of data to be stored
  - Compact features require less storage
  - Provide rich and accurate information
  - Basis for high level features (e.g. more abstract features, objects)

- Here, we will study line segments
  - The simplest geometric structure
  - Suitable for most office-like environments



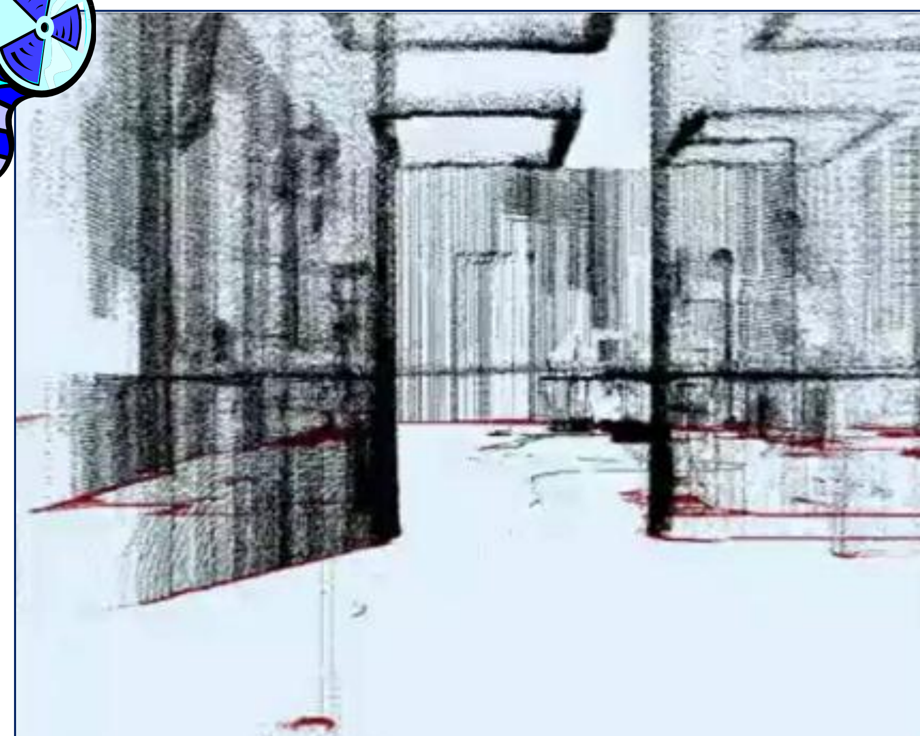




Line Extraction: The Problem

Extract lines from a Range scan (i.e. a point cloud)

- Three main problems:
  - How many lines are there?
  - **Segmentation**: Which points belong to which line ?
  - **Line Fitting/Extraction**: Given points that belong to a line, how to estimate the line parameters ?



- Algorithms we will see:
  1. Split and merge
  2. Linear regression
  3. RANSAC
  4. Hough-Transform

Algorithm 1: Split-and-Merge (standard)

- The most popular algorithm which is originated from computer vision.
- A recursive procedure of fitting and splitting.
- A slightly different version, called Iterative-End-Point-Fit, simply connects the end points for line fitting.

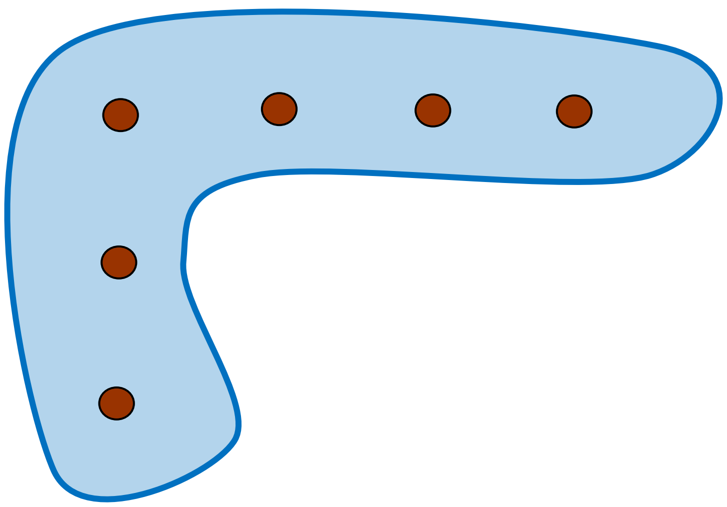
Initialise set **S** to contain all points

Split

- Fit a line to points in current set **S**
- Find the most distant point to the line
- If distance > threshold ⇨ split & repeat with left and right point sets

Merge

- If two consecutive segments are close/collinear enough, obtain the common line and find the most distant point
- If distance <= threshold, merge both segments



Algorithm 1: Split-and-Merge (standard)

- The most popular algorithm which is originated from computer vision.
- A recursive procedure of fitting and splitting.
- A slightly different version, called Iterative-End-Point-Fit, simply connects the end points for line fitting.

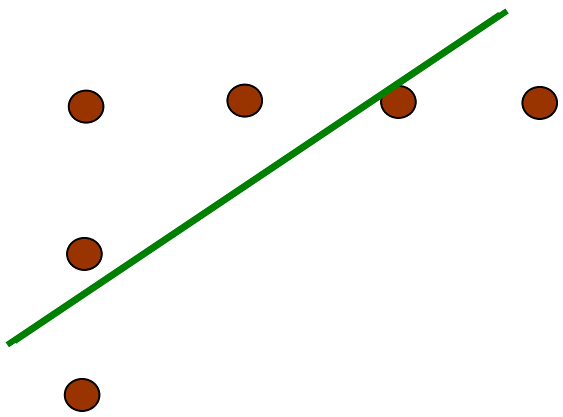
Initialise set **S** to contain all points

Split

- Fit a line to points in current set **S**
- Find the most distant point to the line
- If distance > threshold ⇨ split & repeat with left and right point sets

Merge

- If two consecutive segments are close/collinear enough, obtain the common line and find the most distant point
- If distance <= threshold, merge both segments



Algorithm 1: Split-and-Merge (standard)

- The most popular algorithm which is originated from computer vision.
- A recursive procedure of fitting and splitting.
- A slightly different version, called Iterative-End-Point-Fit, simply connects the end points for line fitting.

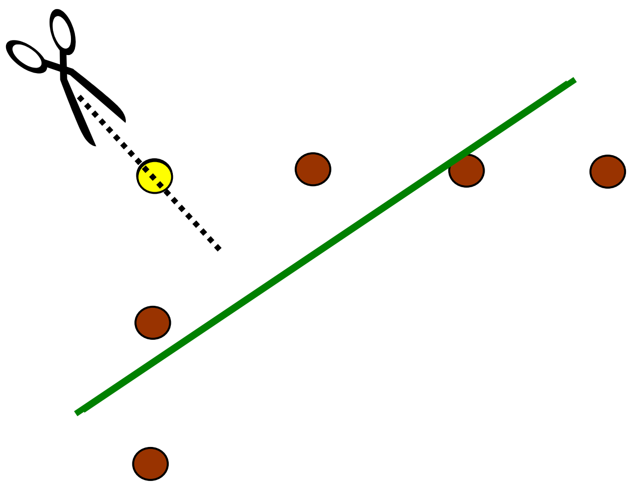
Initialise set **S** to contain all points

Split

- Fit a line to points in current set **S**
- Find the most distant point to the line
- If distance > threshold ⇨ split & repeat with left and right point sets

Merge

- If two consecutive segments are close/collinear enough, obtain the common line and find the most distant point
- If distance <= threshold, merge both segments



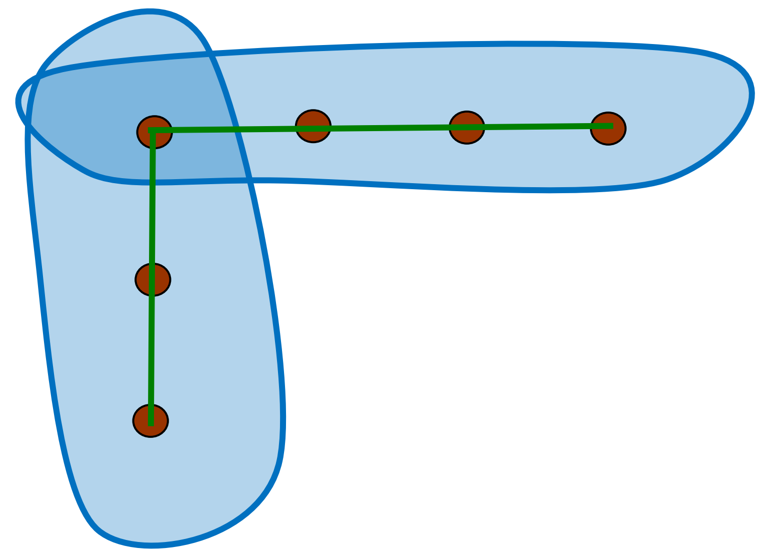
Algorithm 1: Split-and-Merge (standard)

- The most popular algorithm which is originated from computer vision.
- A recursive procedure of fitting and splitting.
- A slightly different version, called Iterative-End-Point-Fit, simply connects the end points for line fitting.

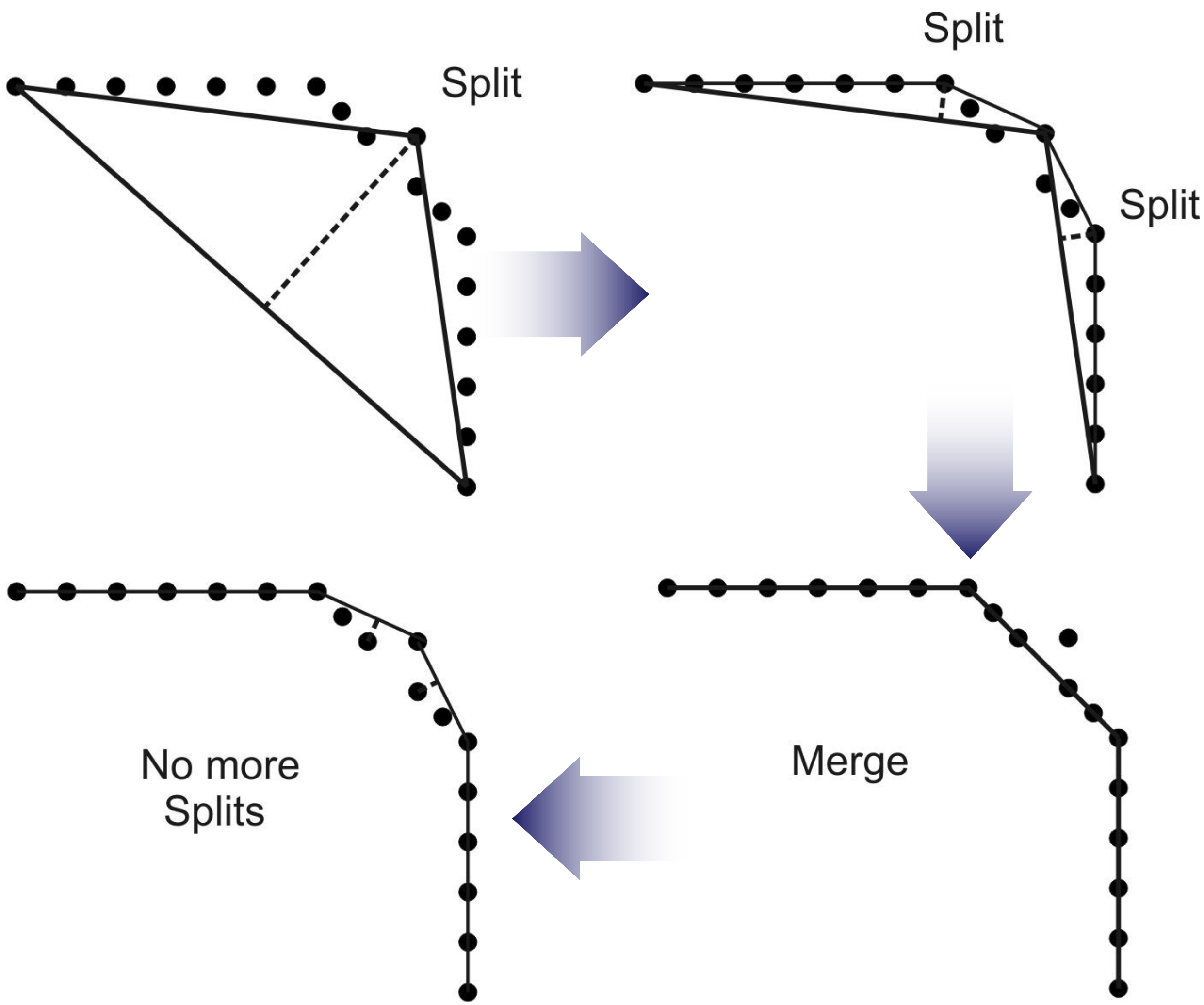
Initialise set **S** to contain all points

Split
- Fit a line to points in current set **S**
- Find the most distant point to the line
- If distance > threshold ⇨ split & repeat with left and right point sets

Merge
- If two consecutive segments are close/collinear enough, obtain the common line and find the most distant point
- If distance <= threshold, merge both segments



Algorithm 1: Split-and-Merge (Iterative-End-Point-Fit)



Algorithm 1: Split-and-Merge

Algorithm 1: *Split-and-Merge*

1. Initial: set s_1 consists of N points. Put s_1 in a list L
2. Fit a line to the next set s_i in L
3. Detect point P with maximum distance d_P to the line
4. If d_P is less than a threshold, continue (go to step 2)
5. Otherwise, split s_i at P into s_{i1} and s_{i2}, replace s_i in L by s_{i1} and s_{i2}, continue (go to 2)
6. When all sets (segments) in L have been checked, merge collinear segments.

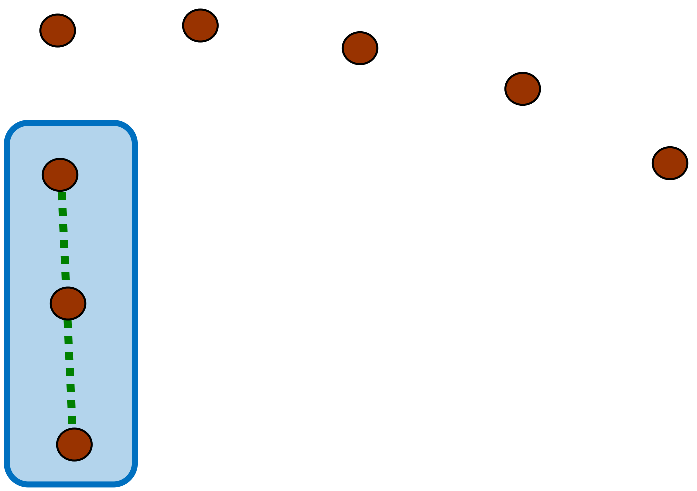
Algorithm 2: Line-Regression

- Uses a "sliding window" of size N_f
- The points within each "sliding window" are fitted by a segment

Line-Regression

- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Compute a **Line Fidelity Array**: each element contains the sum of Mahalanobis distances between 3 consecutive windows
  (Mahalanobis distance used as a measure of similarity)
- Scan Fidelity array for consecutive elements < threshold (using a clustering algorithm).
- For every Fidelity Array element < Threshold, construct a new line segment
- Merge overlapping line segments + recompute line parameters for each segment

$N_f = 3$



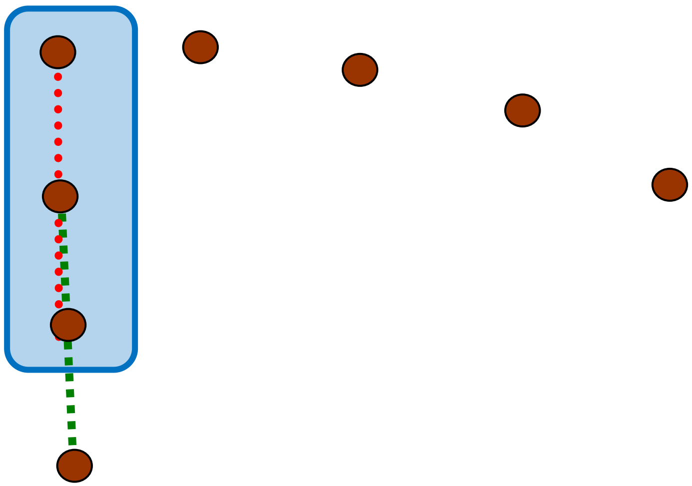
Algorithm 2: Line-Regression

- Uses a "sliding window" of size N_f
- The points within each "sliding window" are fitted by a segment

Line-Regression

- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Compute a **Line Fidelity Array**: each element contains the sum of Mahalanobis distances between 3 consecutive windows (Mahalanobis distance used as a measure of similarity)
- Scan Fidelity array for consecutive elements < threshold (using a clustering algorithm).
- For every Fidelity Array element < Threshold, construct a new line segment
- Merge overlapping line segments + recompute line parameters for each segment

$N_f = 3$



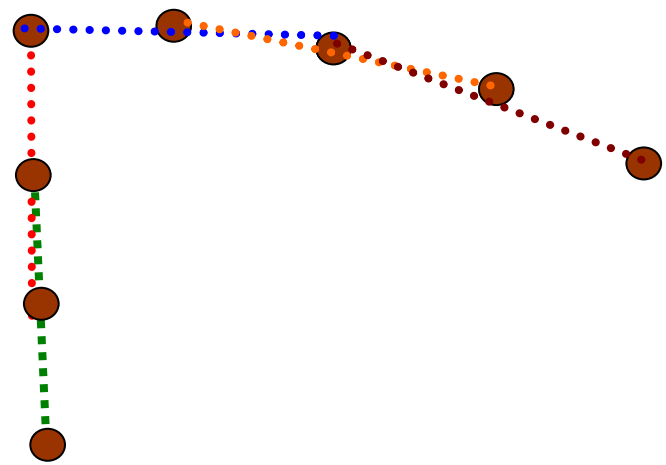
Algorithm 2: Line-Regression

- Uses a "sliding window" of size N_f
- The points within each "sliding window" are fitted by a segment

Line-Regression

- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Compute a **Line Fidelity Array**: each element contains the sum of Mahalanobis distances between 3 consecutive windows
(Mahalanobis distance used as a measure of similarity)
- Scan Fidelity array for consecutive elements < threshold (using a clustering algorithm).
- For every Fidelity Array element < Threshold, construct a new line segment
- Merge overlapping line segments + recompute line parameters for each segment

$N_f = 3$



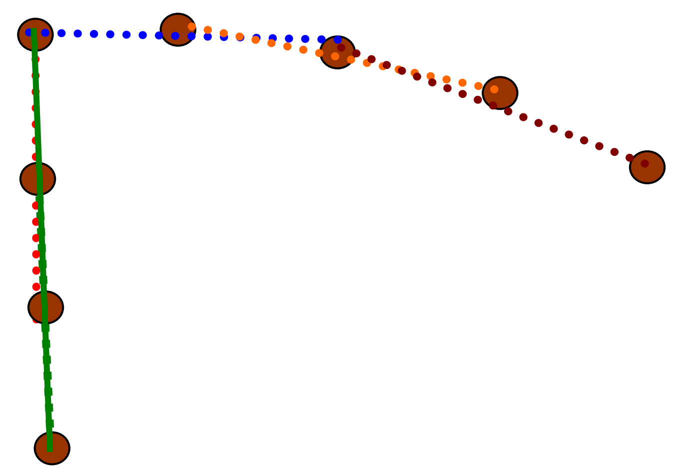
Algorithm 2: Line-Regression

- Uses a "sliding window" of size N_f
- The points within each "sliding window" are fitted by a segment
- Then adjacent segments are merged if their line parameters are close

Line-Regression

- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Compute a **Line Fidelity Array**: each element contains the sum of Mahalanobis distances between 3 consecutive windows
  (Mahalanobis distance used as a measure of similarity)
- Scan Fidelity array for consecutive elements < threshold (using a clustering algorithm).
- For every Fidelity Array element < Threshold, construct a new line segment
- Merge overlapping line segments + recompute line parameters for each segment

$N_f = 3$



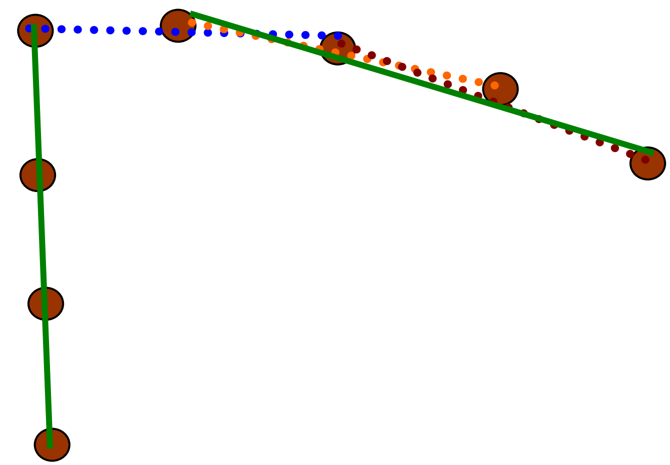
Algorithm 2: Line-Regression

- Uses a "sliding window" of size N_f
- The points within each "sliding window" are fitted by a segment
- Then adjacent segments are merged if their line parameters are close

Line-Regression

- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Compute a **Line Fidelity Array**: each element contains the sum of Mahalanobis distances between 3 consecutive windows
  (Mahalanobis distance used as a measure of similarity)
- Scan Fidelity array for consecutive elements < threshold (using a clustering algorithm).
- For every Fidelity Array element < Threshold, construct a new line segment
- Merge overlapping line segments + recompute line parameters for each segment

$N_f = 3$



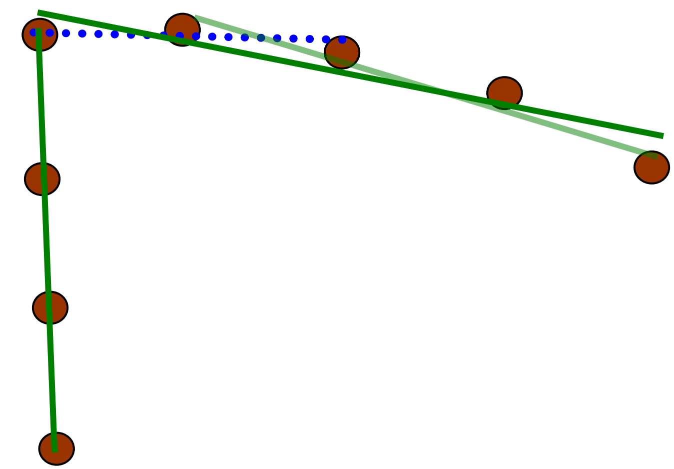
Algorithm 2: Line-Regression

- Uses a "sliding window" of size N_f
- The points within each "sliding window" are fitted by a segment
- Then adjacent segments are merged if their line parameters are close

Line-Regression

- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Compute a **Line Fidelity Array**: each element contains the sum of Mahalanobis distances between 3 consecutive windows (Mahalanobis distance used as a measure of similarity)
- Scan Fidelity array for consecutive elements < threshold (using a clustering algorithm).
- For every Fidelity Array element < Threshold, construct a new line segment
- Merge overlapping line segments + recompute line parameters for each segment

$N_f = 3$

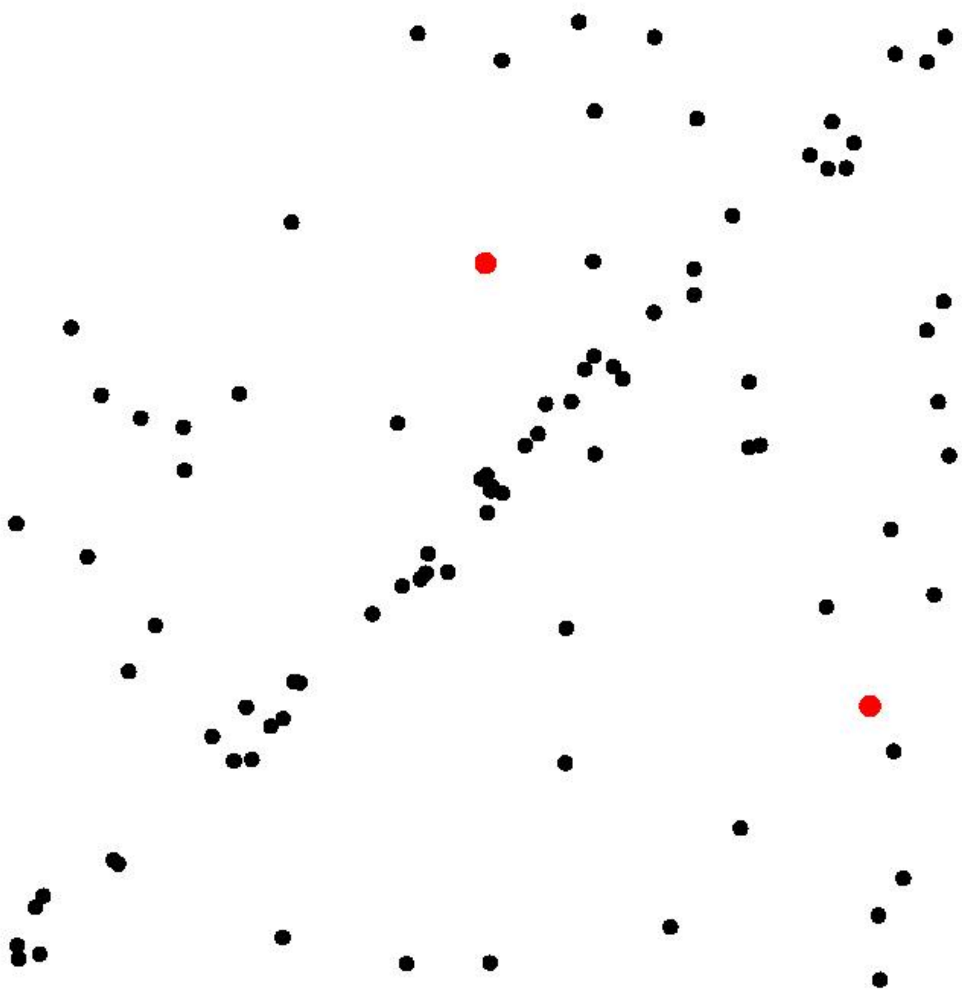


Algorithm 3: RANSAC

- **RANSAC** = **RAN**dom **SA**mple **C**onsensus.
- It is a generic and robust fitting algorithm of models in the presence of outliers (i.e. points which do not satisfy a model)
- Generally applicable algorithm to any problem where the goal is to **identify the inliers which satisfy a predefined model**.
- Typical applications in robotics are: line extraction from 2D range data, plane extraction from 3D range data, feature matching, structure from motion, …
- RANSAC is an **iterative** method and is **non-deterministic** in that the probability to find a set free of outliers increases as more iterations are used
- <u>Drawback: A nondeterministic method, results are different between runs.</u>

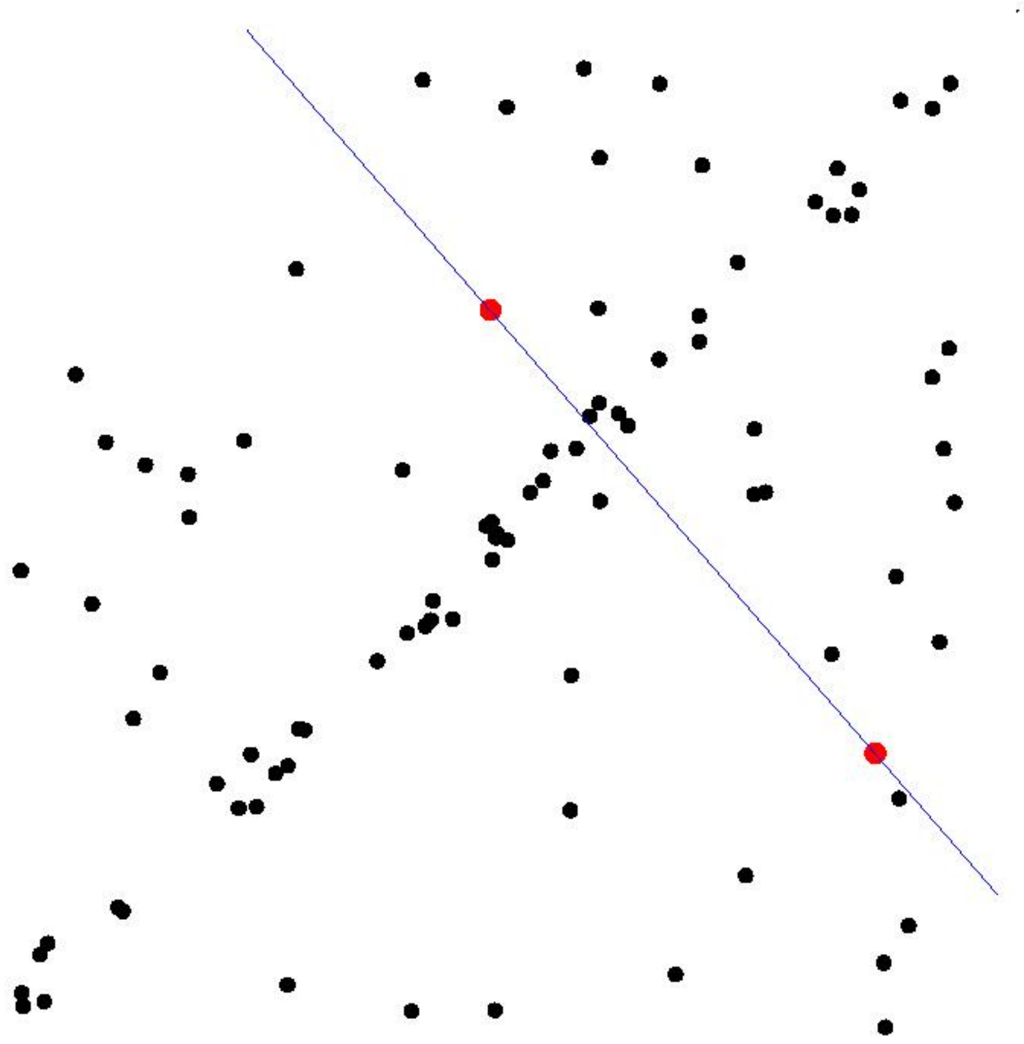
Algorithm 3: RANSAC

Algorithm 3: RANSAC



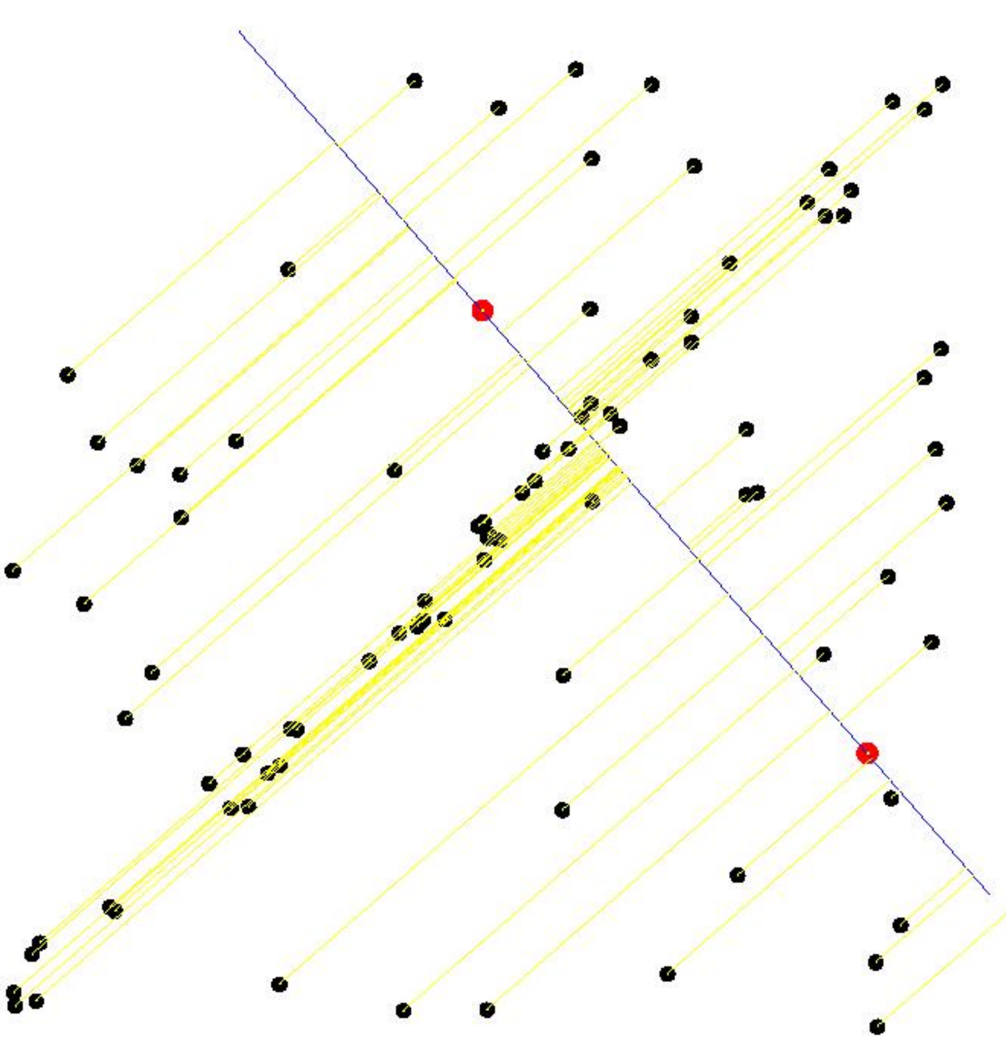
- **Select sample of 2 points at random**

Algorithm 3: RANSAC



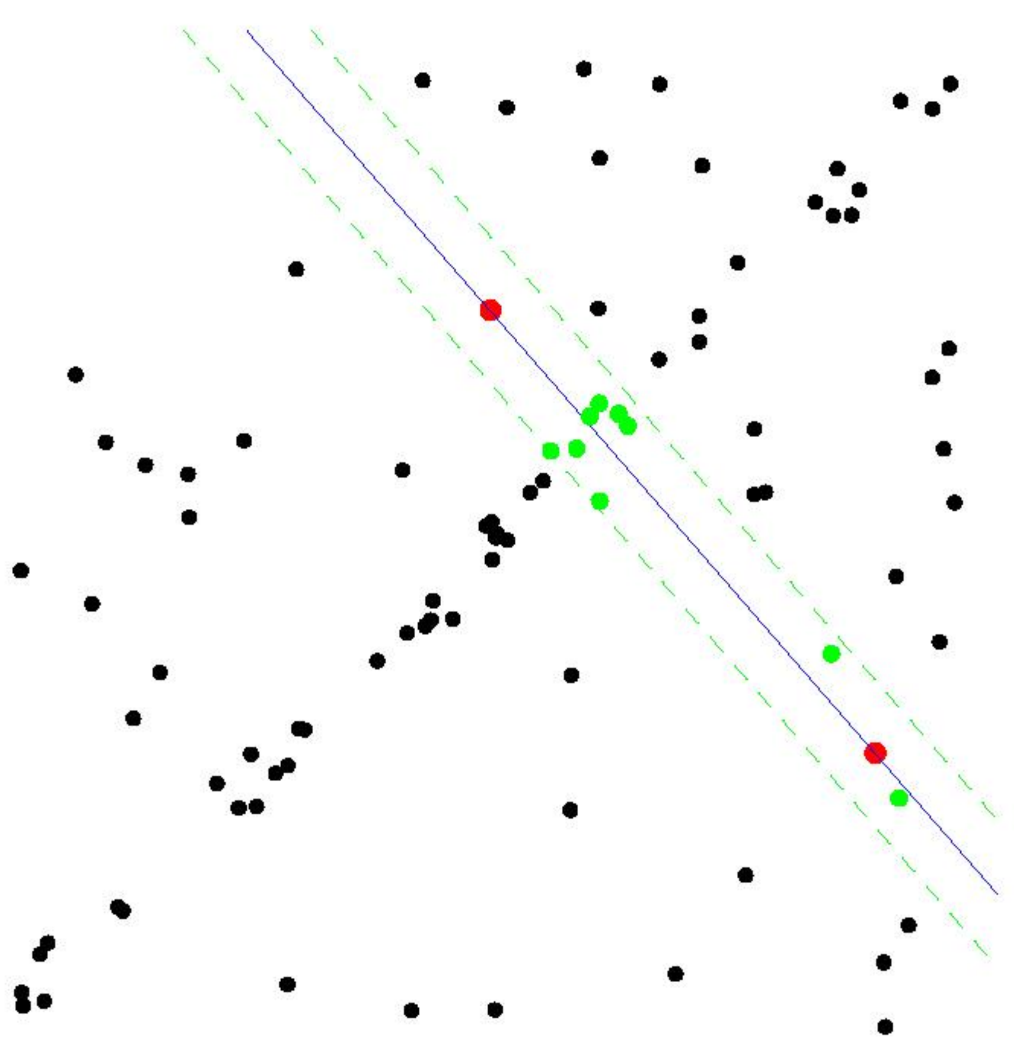
- Select sample of 2 points at random
- **Calculate model parameters that fit the data in the sample**

RANSAC



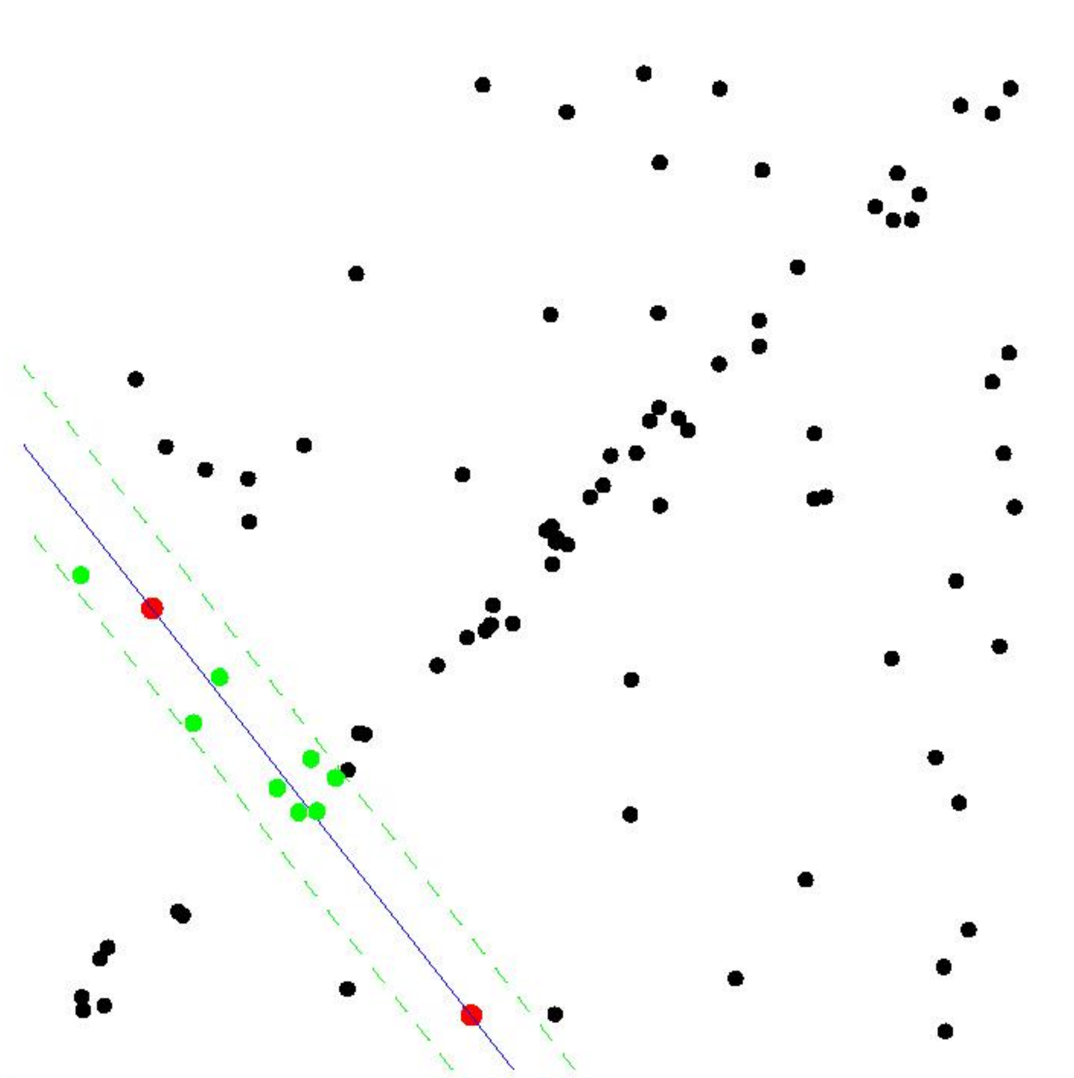
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- **Calculate error function for each data point**

Algorithm 3: RANSAC



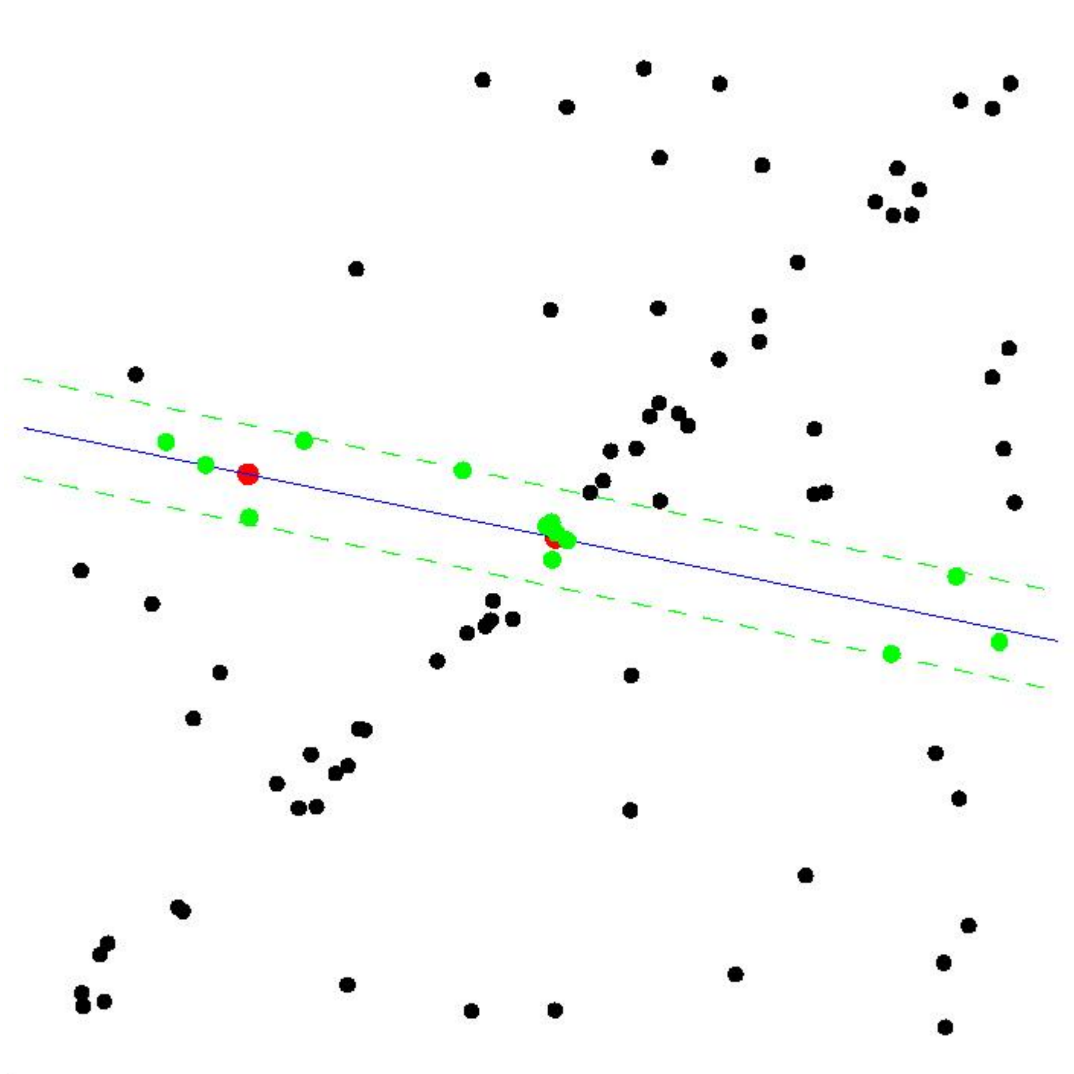
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- **Select data that support current hypothesis**

Algorithm 3: RANSAC



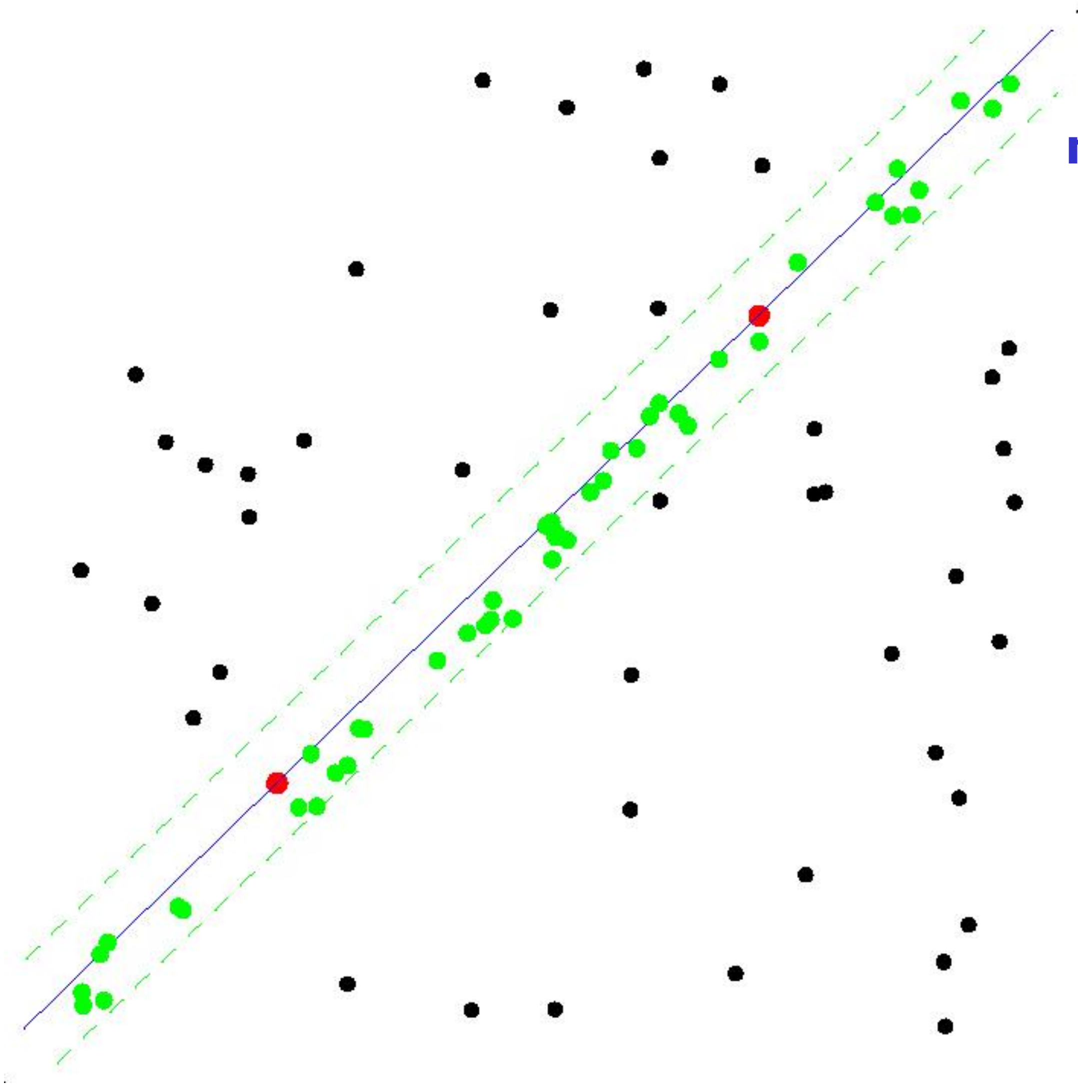
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that support current hypothesis
- **Repeat sampling**

Algorithm 3: RANSAC



- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that support current hypothesis
- **Repeat sampling**

Algorithm 3: RANSAC



Set with the maximum number of inliers obtained within k iterations

Algorithm 3: RANSAC

Algorithm 4: *RANSAC* *(for line extraction from 2D range data)*

1. Initial: let A be a set of N points
2. **repeat**
3. Randomly select a sample of 2 points from A
4. Fit a line through the 2 points
5. Compute the distances of all other points to this line
6. Construct the inlier set (i.e. count the number of points with distance to the line $< d$)
7. Store these inliers
8. **until** Maximum number of iterations k reached
9. The set with the maximum number of inliers is chosen as a solution to the problem

Algorithm 3: RANSAC

How many iterations does RANSAC need?

- We cannot know in advance if the observed set contains the max. no. inliers
  ⇨ ideally: check all possible combinations of 2 points in a dataset of **N** points.
- No. all pairwise combinations: **N(N-1)/2**
  ⇨ computationally infeasible if **N** is too large.
  example: laser scan of **360** points ⇨ need to check all 360*359/2= **64,620** possibilities!
- Do we really need to check all possibilities or can we stop RANSAC after iterations?
  Checking a subset of combinations is enough if we have a **rough** estimate of the percentage of inliers in our dataset
- This can be done in a probabilistic way

Algorithm 3: RANSAC

How many iterations does RANSAC need?

- w = number of inliers / N
  where N : tot. no. data points
  ⇨ w : fraction of inliers in the dataset = probability of selecting an inlier-point

- Let p : probability of finding a set of points free of outliers

- Assumption: the 2 points necessary to estimate a line are selected independently
  ⇨ w^2 = prob. that both points are inliers
  ⇨ $1-w^2$ = prob. that at least one of these two points is an outlier

- Let k : no. RANSAC iterations executed so far
  ⇨ $(1-w^2)^k$ = prob. that RANSAC never selects two points that are both inliers.
  ⇨ $1-p = (1-w^2)^k$ and therefore :

$$k = \frac{\log(1-p)}{\log(1-w^2)}$$

Algorithm 3: RANSAC

How many iterations does RANSAC need?

- The number of iterations ***k*** is

$$k = \frac{\log(1-p)}{\log(1-w^2)}$$

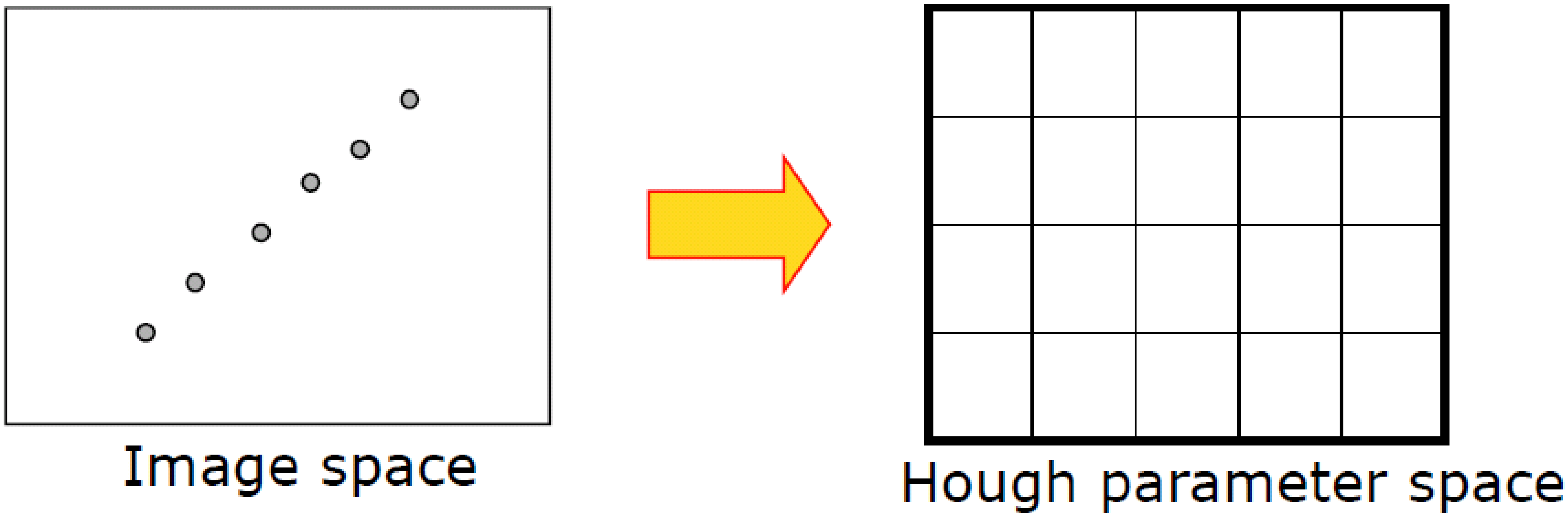
⇨ knowing the fraction of inliers ***w***, after ***k*** RANSAC iterations we will have a prob. ***p*** of finding a set of points free of outliers.

- Example: if we want a probability of success ***p***=99% and we know that ***w***=50% ⇨ ***k***=16 iterations, which is much less than the number of all possible combinations!

- In practice we need only a rough estimate of ***w***. More advanced implementations of RANSAC estimate the fraction of inliers & adaptively set it on every iteration.

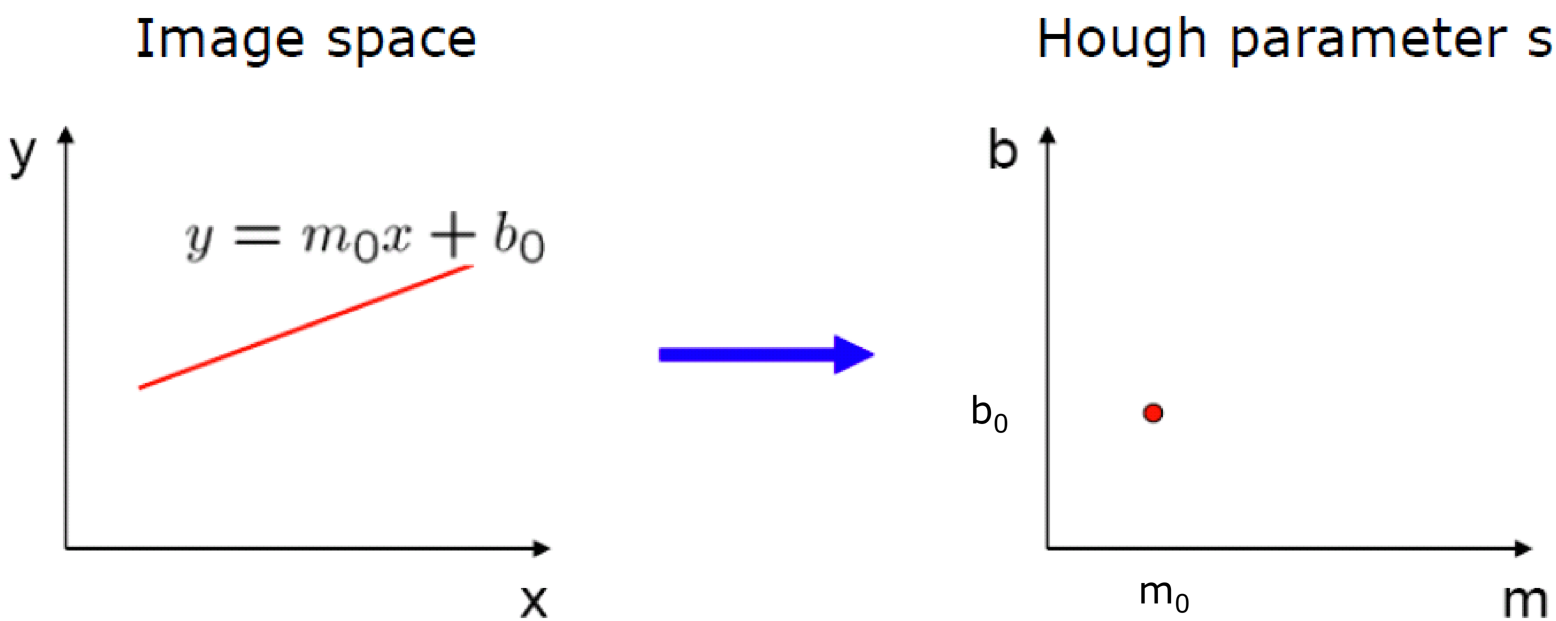
Algorithm 4: Hough-Transform

- Hough Transform uses a voting scheme



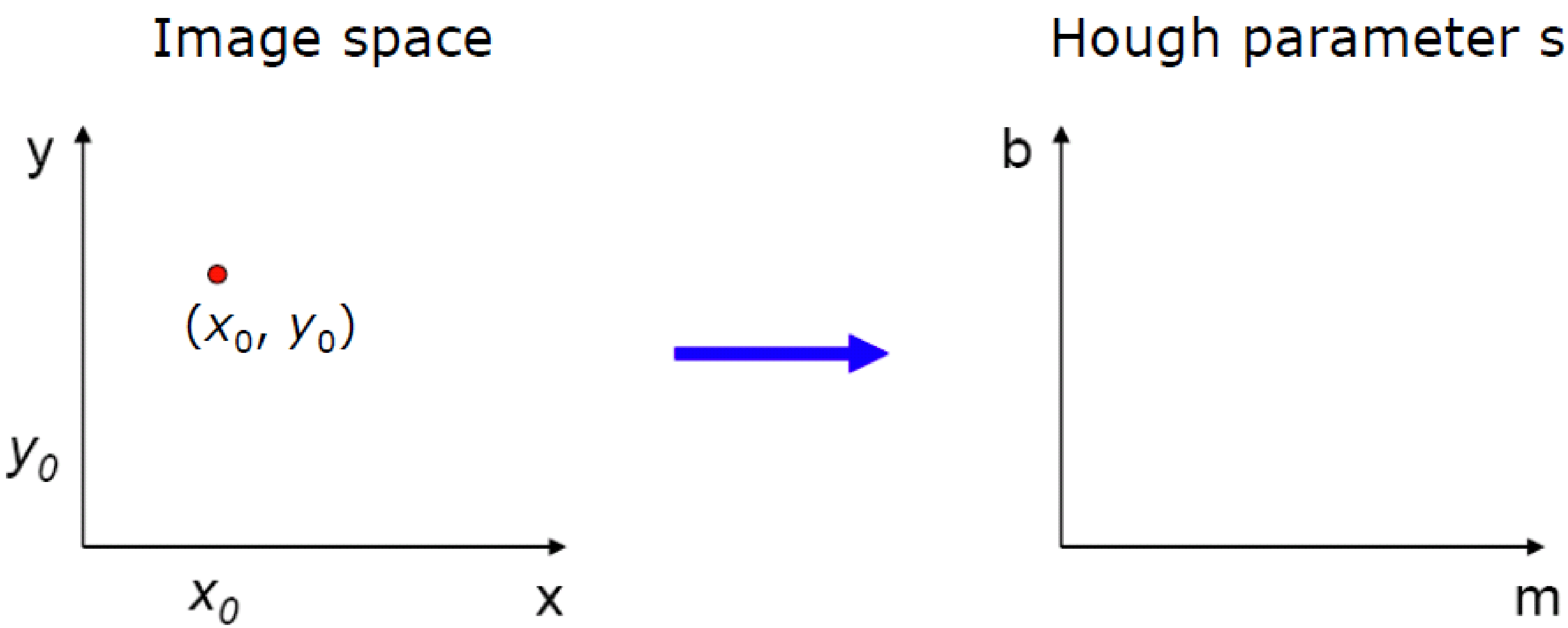
Algorithm 4: Hough-Transform

- A line in the image corresponds to a point in Hough space



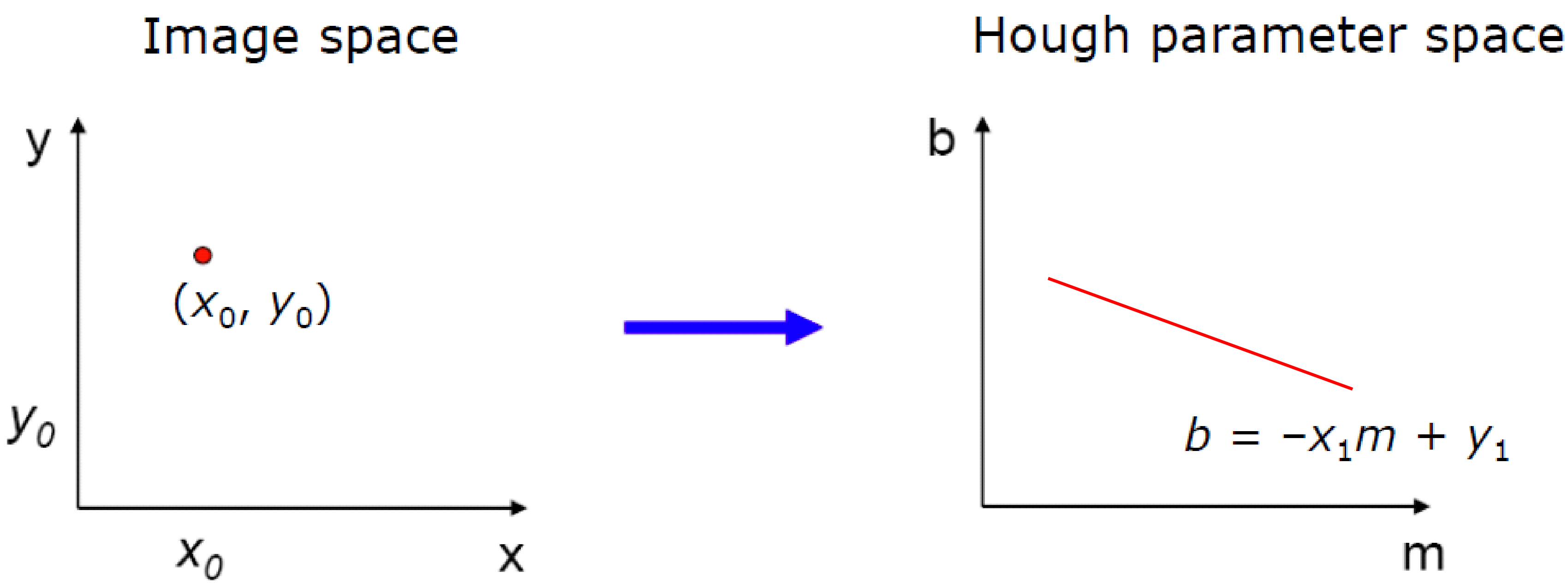
Algorithm 4: Hough-Transform

- What does a point (x_0, y_0) in the image space map to in the Hough space?



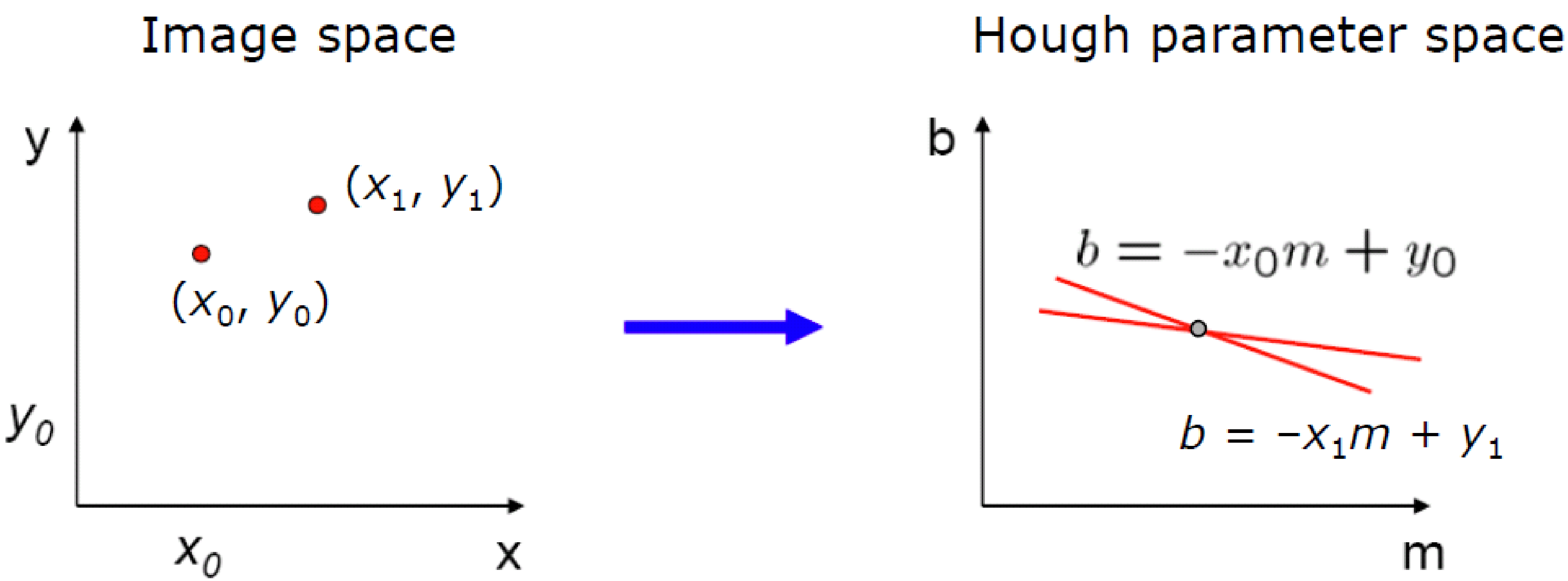
Algorithm 4: Hough-Transform

- What does a point (x_0, y_0) in the image space map to in the Hough space?

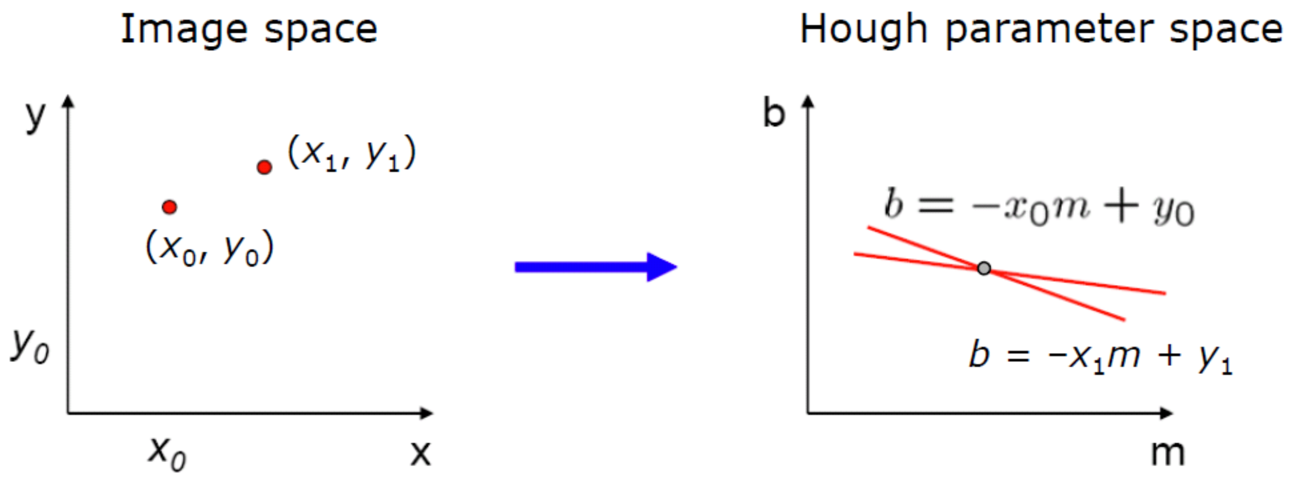


Algorithm 4: Hough-Transform

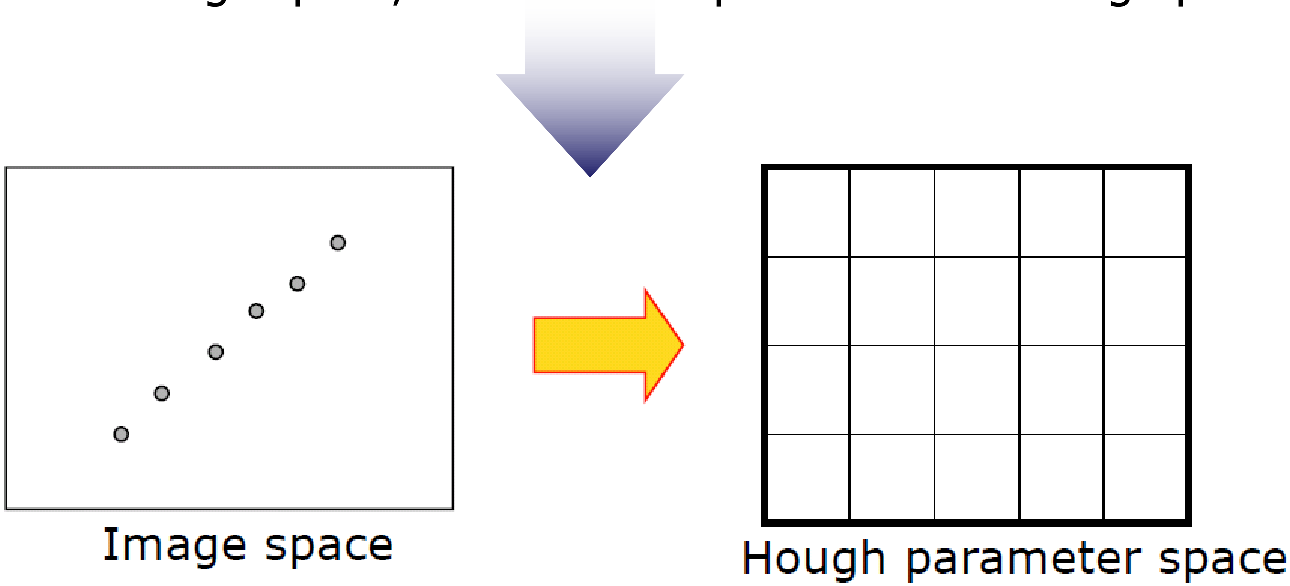
- Where is the line that contains both (x_0, y_0) and (x_1, y_1)?
  - It is the intersection of the lines $b = -x_0 m + y_0$ and $b = -x_1 m + y_1$



Algorithm 4: Hough-Transform



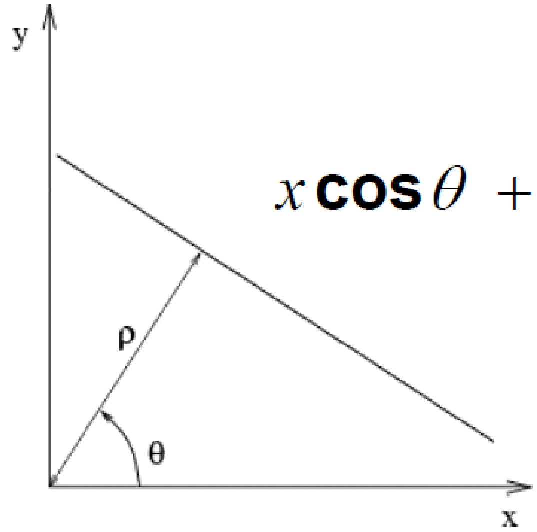
- Each point in image space, votes for line-parameters in Hough parameter space



Algorithm 4: Hough-Transform

- Problems with the (m,b) space:
  - Unbounded parameter domain
  - Vertical lines require infinite m

- Alternative: polar representation



$$x \cos\theta + y \sin\theta = \rho$$

Each point in image space will map to a sinusoid in the (θ,ρ) parameter space

Algorithm 4: Hough-Transform

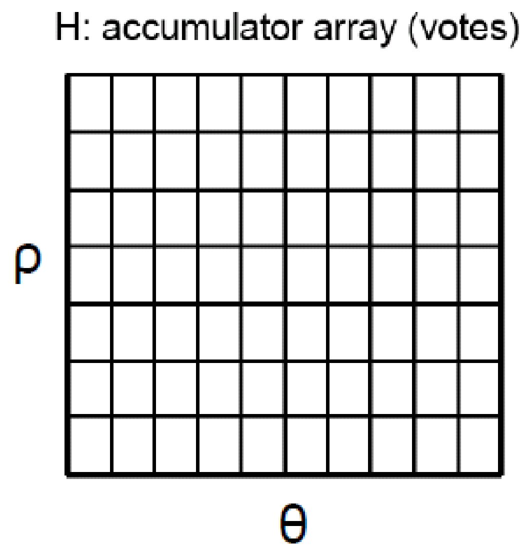
1. Initialize accumulator H to all zeros

2. **for** each edge point (x,y) in the image
   - **for** all θ in [0,180]
     - Compute $\rho = x \cos\theta + y \sin\theta$
     - $H(\theta, \rho) = H(\theta, \rho) + 1$
   - **end**

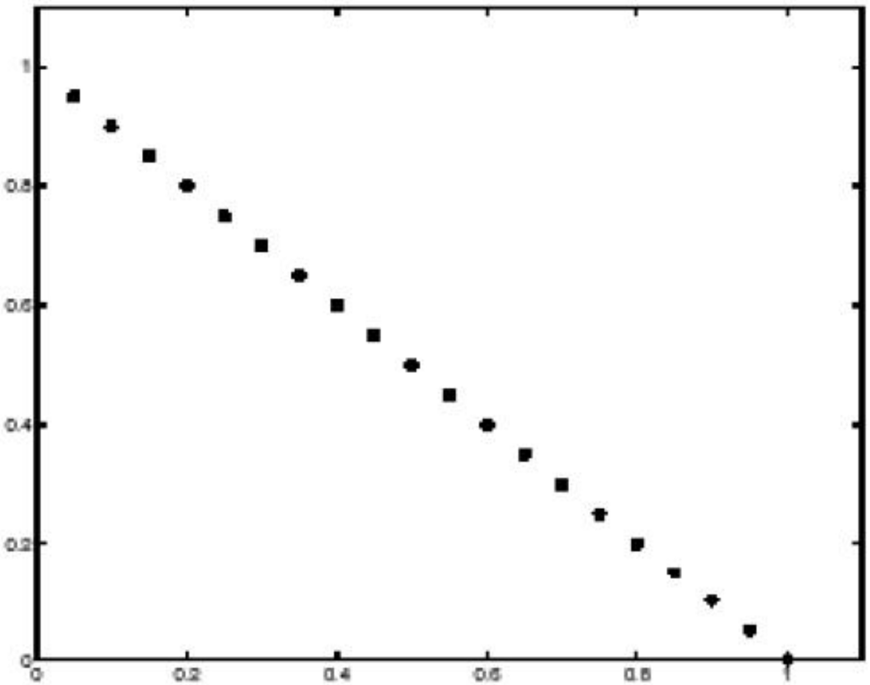
   end

3. Find the values of (θ, ρ) where $H(\theta, \rho)$ is a local maximum

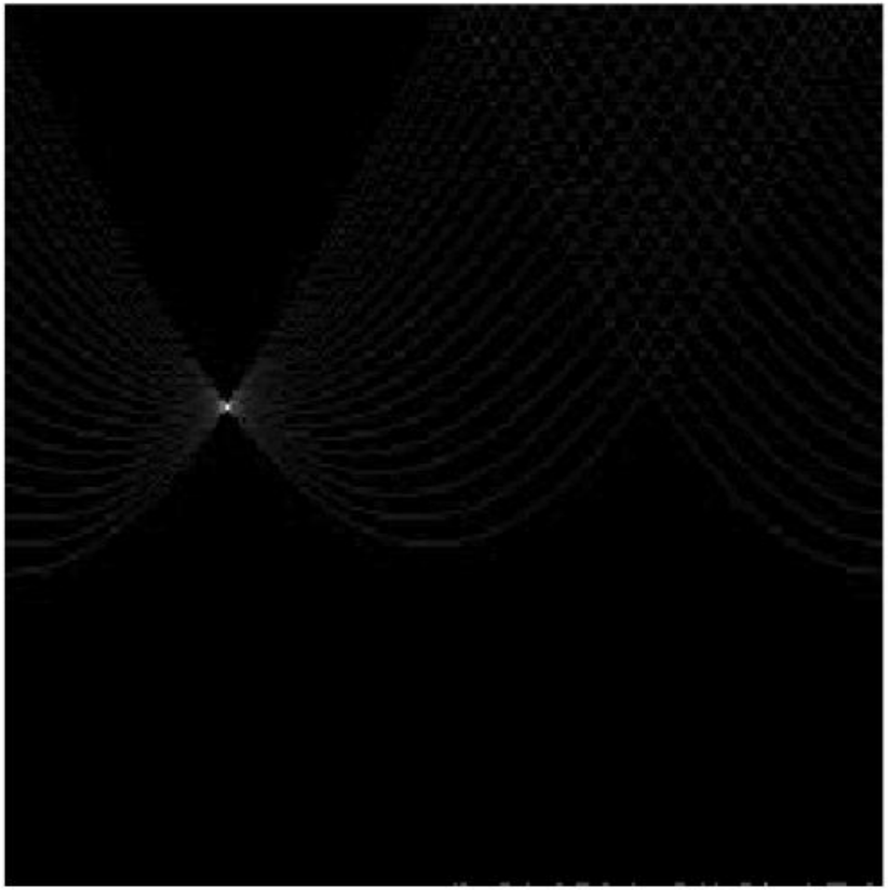
4. The detected line in the image is given by $\rho = x \cos\theta + y \sin\theta$



Algorithm 4: Hough-Transform



features

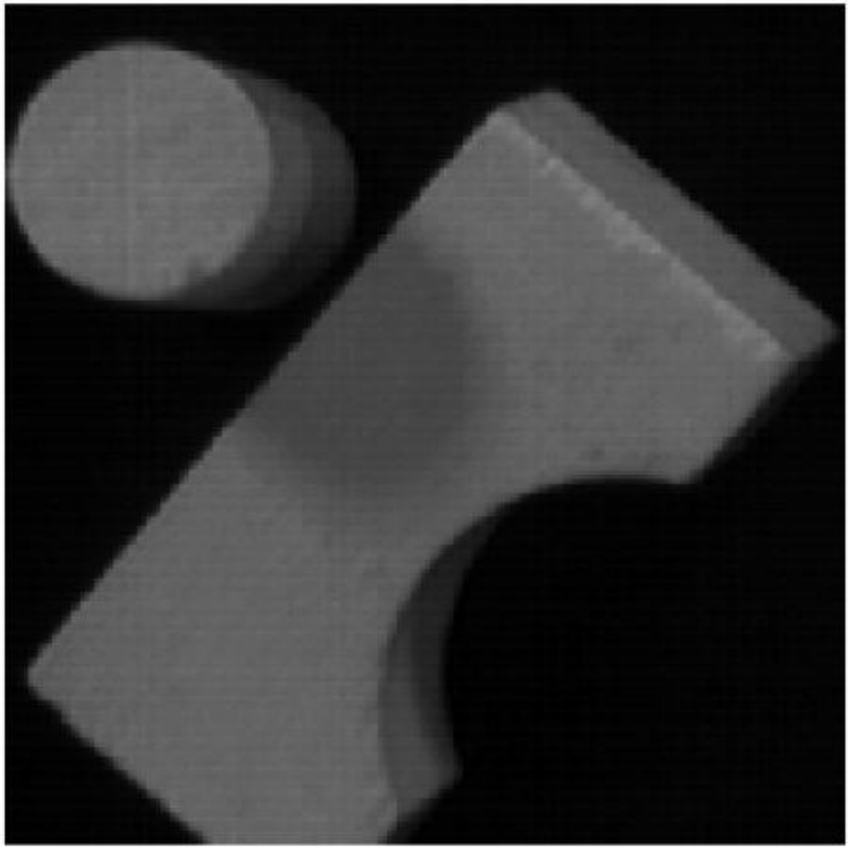


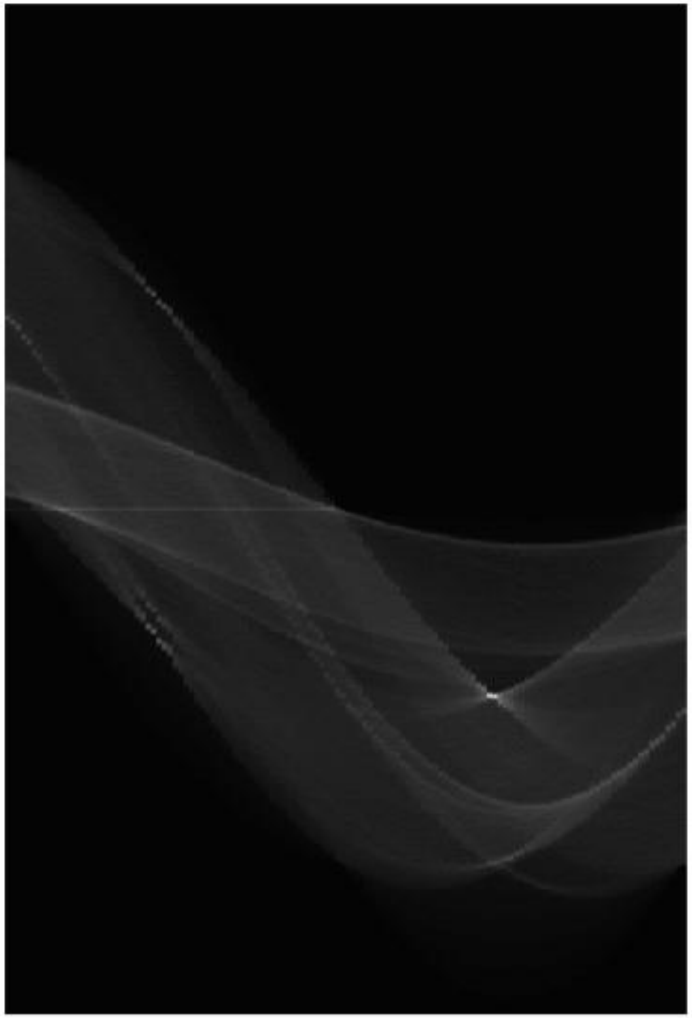
votes

Algorithm 4: Hough-Transform

Square

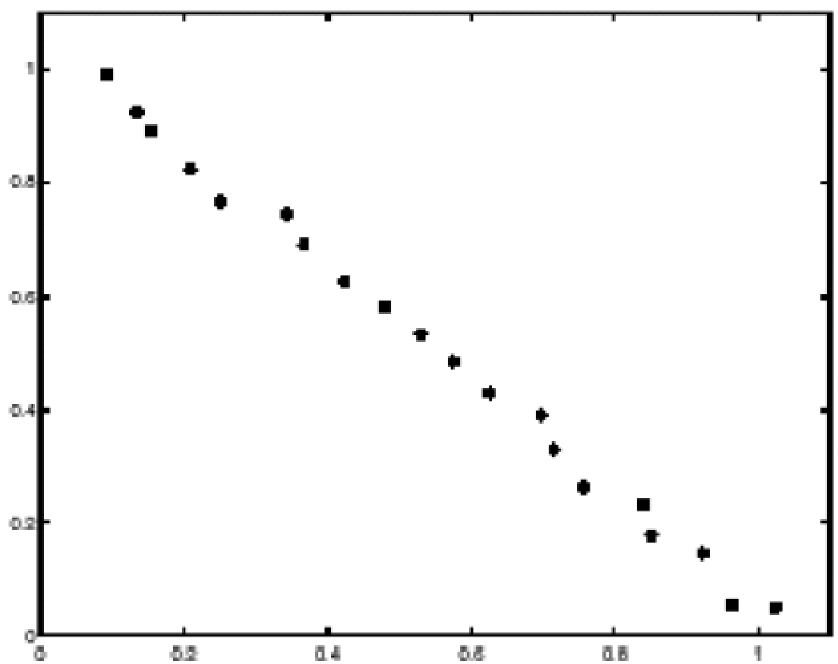
Algorithm 4: Hough-Transform



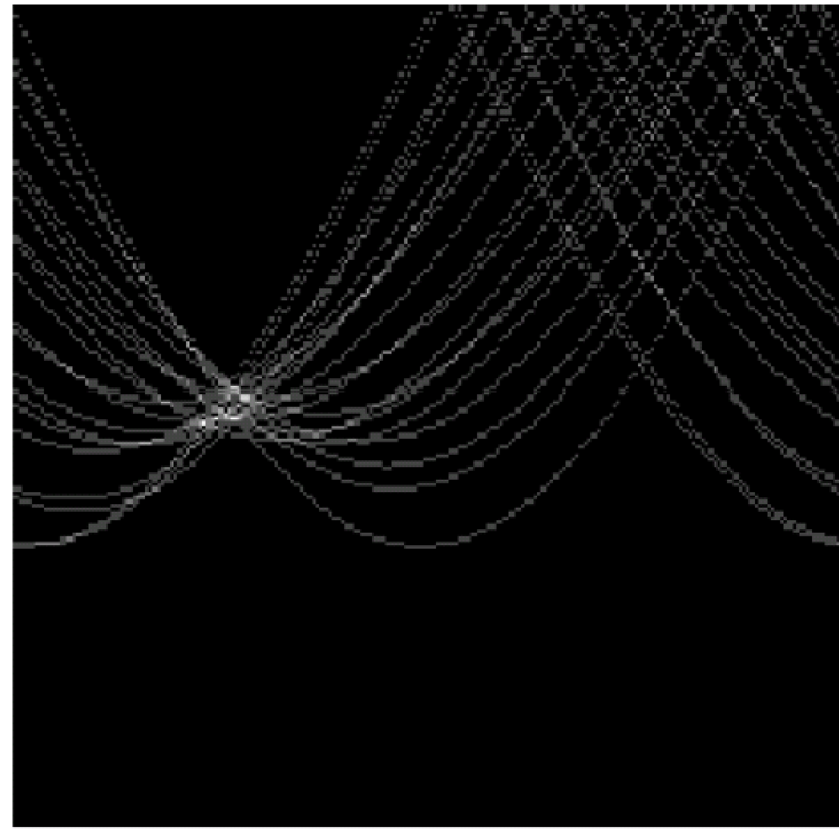


Algorithm 4: Hough-Transform

Effect of Noise



features

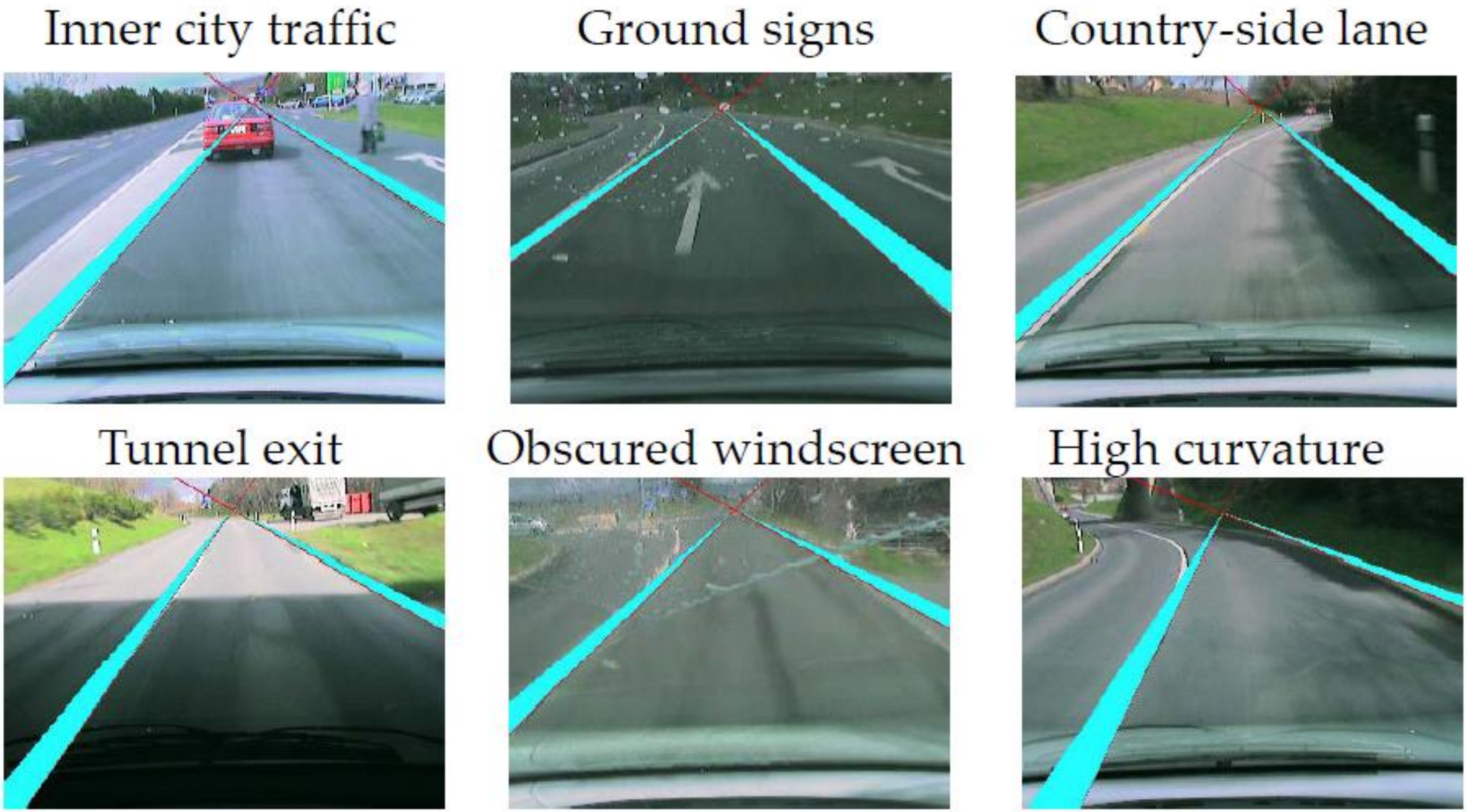


votes

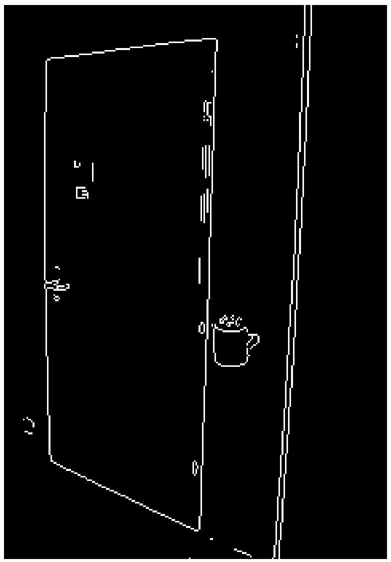
- Peak gets fuzzy and hard to locate

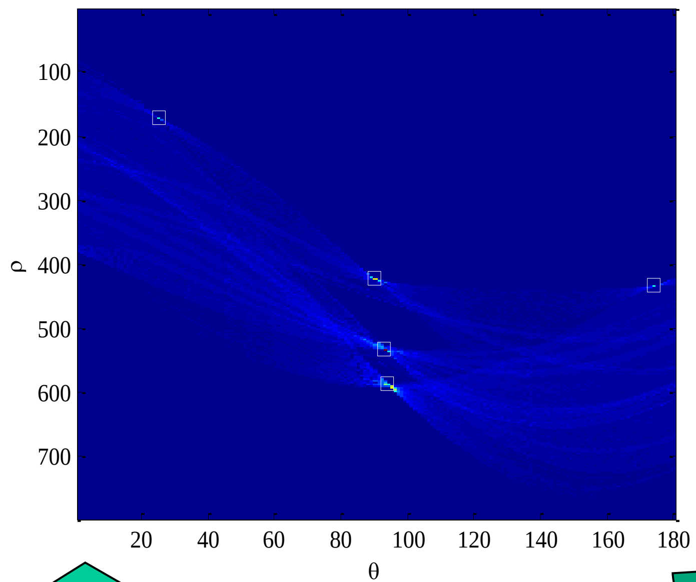
Algorithm 4: Hough-Transform

Application: Lane detection



Example – Door detection using Hough Transform





Hough Transform

Comparison of Line Extraction Algorithms

	Complexity	Speed (Hz)	False positives	Precision
Split-and-Merge	$N \log N$	**1500**	**10%**	**+++**
Incremental	$S\ N$	600	6%	+++
Line-Regression	$N\ N_f$	**400**	**10%**	**+++**
RANSAC	$S\ N\ k$	**30**	**30%**	**++++**
Hough-Transform	$S\ N\ N_C + S\ N_R\ N_C$	**10**	**30%**	**++++**
Expectation Maximization	$S\ N_1\ N_2\ N$	1	50%	++++

- Split-and-merge, Incremental and Line-Regression: fastest
  - Deterministic & make use of the sequential ordering of raw scan points (: points captured according to the rotation direction of the laser beam)
- If applied on randomly captured points only last 3 algorithms would segment all lines.
- RANSAC, HT and EM: produce greater precision ⇨ more robust to outliers