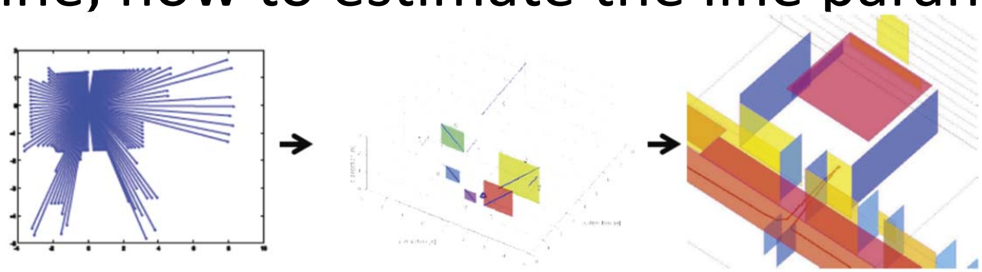
CMPE 185 Autonomous Mobile Robots

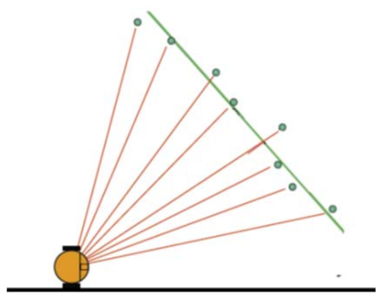
Perception: Feature Extraction Based on Range Data

Dr. Wencen Wu
Computer Engineering Department
San Jose State University

Line Extraction Problem

- Given range data, how do we extract line segments (or planes)?
- These features (line segments) can be used to build maps or be compared with an existing map.
- Three main problems in line extraction in unknown environments
  - 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?





Line Extraction: Representing a line

- A line can be represented by

$$y = ax + b$$

$y = ?$

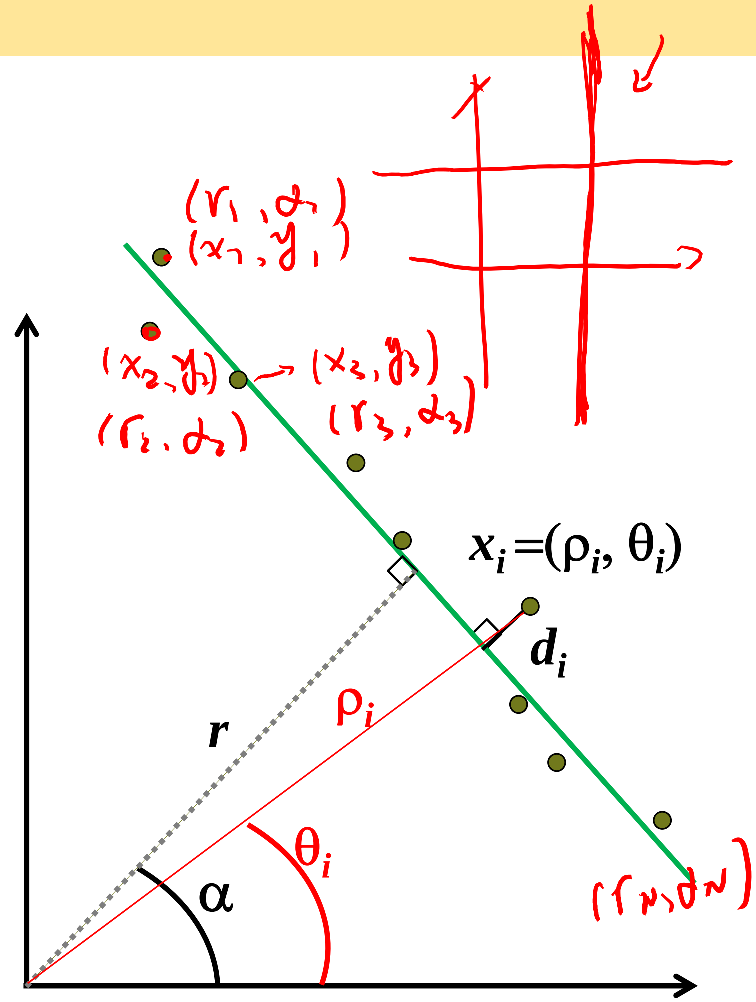
- Q: how to represent a vertical line?

- Polar coordinate $\boldsymbol{x} = (r, \alpha)$
  - r: distance from the origin to the close point on the line
  - α: angle of the line
  - $r = x * cos\alpha + y * sin\alpha$

- The line can represented by

$$y = \frac{-cos\alpha}{sin\alpha} * x + \frac{r}{sin\alpha}$$

for $\alpha \in [0{,}180]$ and $r \in \mathbf{R}$
or $\alpha \in [0{,}360]$ and $r \geq 0$



$$\begin{cases} y_1 = -\frac{\cos\alpha}{\sin\alpha} x_1 + \frac{r}{\sin\alpha} \\ \vdots \\ y_N = -\frac{\cos\alpha}{\sin\alpha} x_N + \frac{r}{\sin\alpha} \end{cases}$$

N

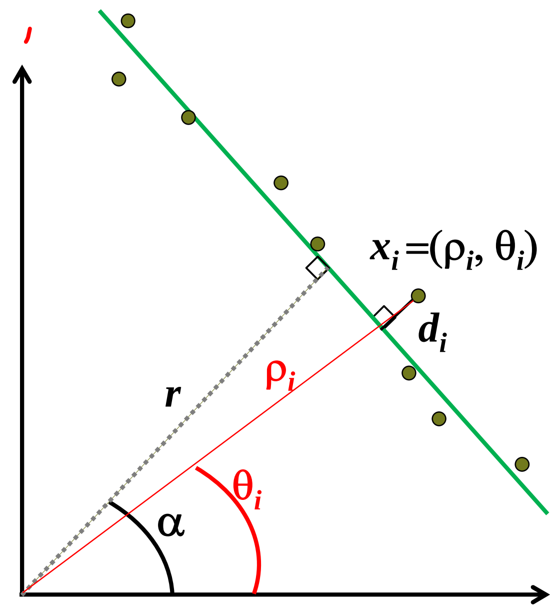
Line Extraction Problem

- Given a measurement vector of *N* range and bearing measurements $\boldsymbol{x_i} = (\rho_i, \theta_i)$, what are the parameters (*r*, α) that define a line feature for these measurements.

pointing angle of sensor θ_i [deg]	range ρ_i [m]
0	0.5197
5	0.4404
10	0.4850
15	0.4222
20	0.4132
25	0.4371
30	0.3912
35	0.3949
40	0.3919
45	0.4276
50	0.4075
55	0.3956
60	0.4053
65	0.4752
70	0.5032
75	0.5273
80	0.4879

$(0, 0.5197) \Leftarrow \vec{x}_1$

$(60, 0.4053) \rightarrow \vec{x}_{13}$



Line Extraction

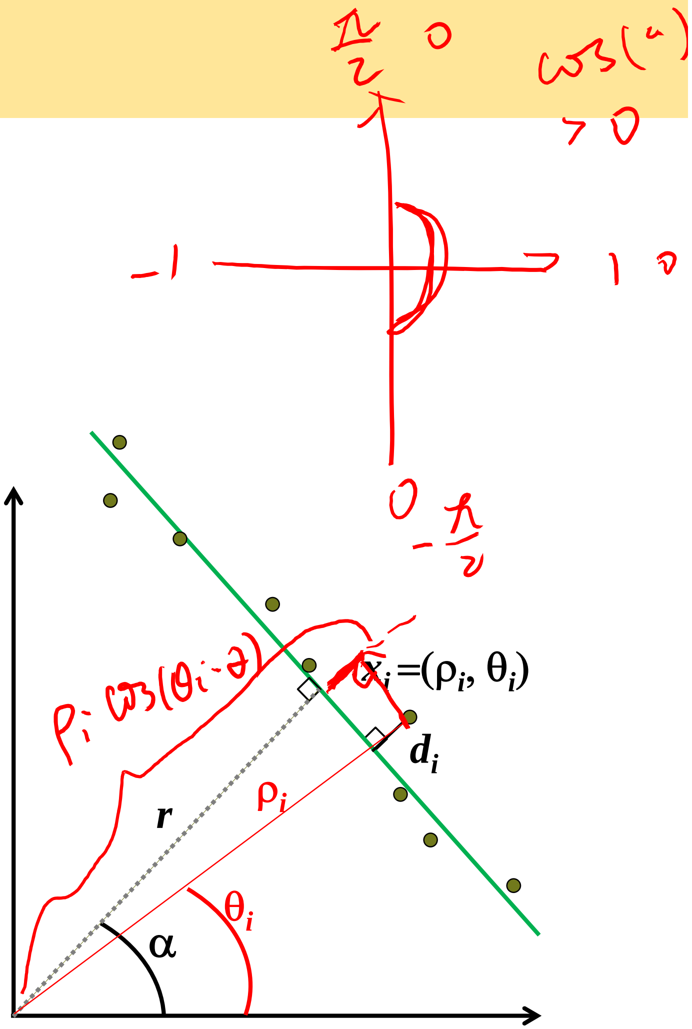
- Polar measurement $\boldsymbol{x_i} = (\rho_i, \theta_i)$
- The corresponding Euclidean coordinate:
  - $x_i = \rho_i cos\theta_i$
  - $y_i = \rho_i sin\theta_i$

- Point-Line distance

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

- If each measurement is equally uncertain, the sum of squared errors:

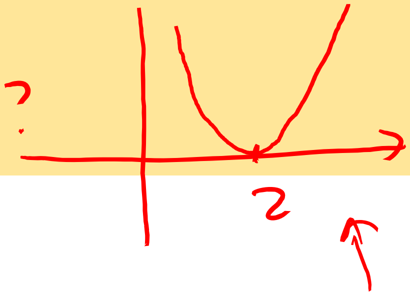
$$S = \sum_i d_i^2 = \sum_i (\rho_i \cos(\theta_i - \alpha) - r)^2$$



Line Extraction

$y = \frac{1}{2}(x-2)^2$

find x, that minimize y?



$\frac{dy}{dx} = \frac{1}{2} \cdot 2 \cdot (x-2)$

$= x-2 = 0 \rightarrow x=2$

- Each sensor measurement may have its own, unique uncertainty σ_i^2

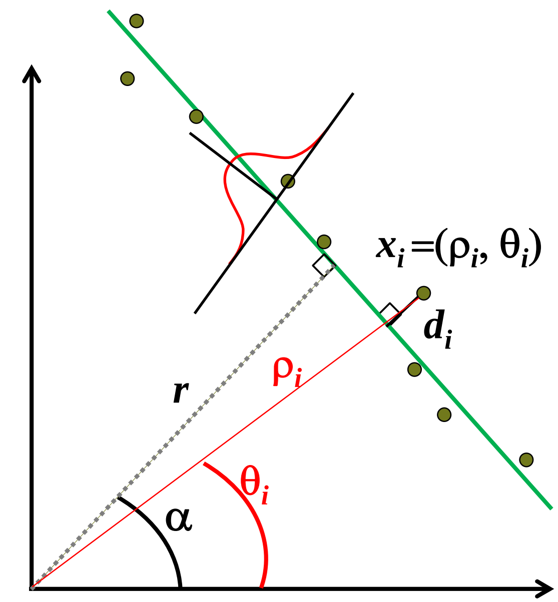
cost function

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

$$w_i = \frac{1}{\sigma_i^2}$$

- Goal: minimize S when selecting (r, α)

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



- "Unweighted Least Squares"
- "Weighted Least Squares"

Line Extraction

$N=4.$

$\alpha = \frac{1}{2}\mathrm{atan}\left(\frac{\Sigma\rho_i^2\sin 2\theta_i - \frac{2}{1}\Sigma\Sigma(\rho_1\rho_2\cos\theta_1\sin\theta_2 + \rho_1\rho_3\cos\theta_1\sin\theta_3 + \rho_1\rho_4\cos\theta_1\sin\theta_4 + (2,3) + (2,4) + (3,4))}{\Sigma\rho_i^2\cos 2\theta_i}\right.$

$\rho_1^2\sin 2\theta_1 + \rho_2^2\sin 2\theta_2 - \cdots$

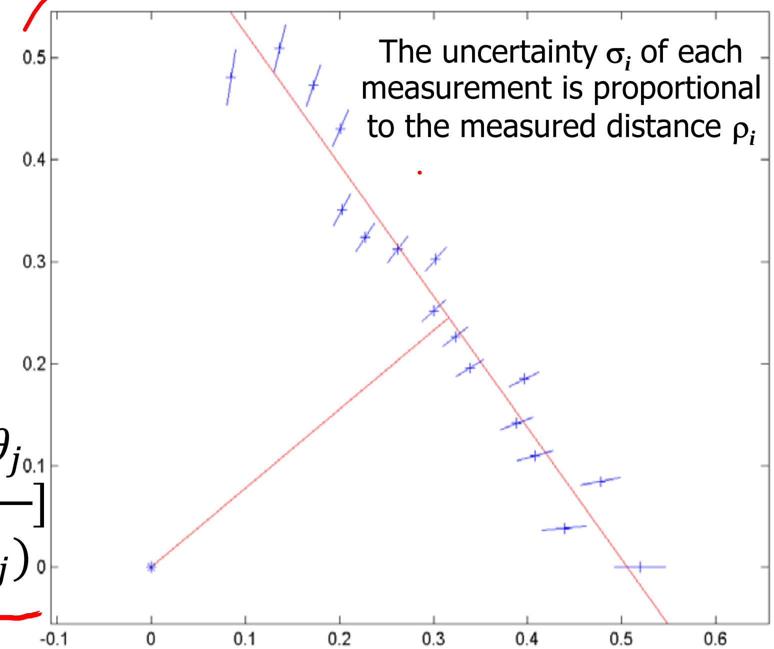
- Weighted least squares and solving the equations

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

- The line parameters are

$$\alpha = \frac{1}{2}\mathrm{atan}\left[\frac{\Sigma w_i\rho_i^2 sin2\theta_i - \frac{2}{\sum w_i}\sum\sum w_i w_j \rho_i\rho_j cos\theta_i sin\theta_j}{\Sigma w_i\rho_i^2 cos2\theta_i - \frac{1}{\sum w_i}\sum\sum w_i w_j\rho_i\rho_j cos(\theta_i+\theta_j)}\right]$$

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

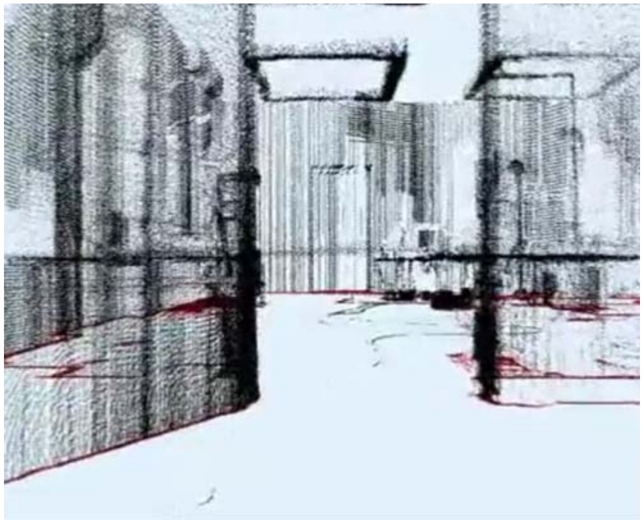


$\rho_1\rho_2\cos(\theta_1+\theta_2) + \rho_1\rho_3\cos(\theta_1+\theta_3)$
$+ \rho_1\rho_4\cos(\theta_1+\theta_4) + \rho_2\rho_3\cos(\theta_2+\theta_3)$
$+ \rho_2\rho_4\cos(\theta_2+\theta_4) + \rho_3\rho_4\cos(\theta_3+\theta_4)$

Line Extraction from a Point Cloud

- Extract lines from a point cloud (e.g. range scan)
- 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:
  - Split-and-merge
  - Linear regression
  - RANSAC
  - Hough-Transform



Line Extraction – Split and Merge (standard)

- Popular algorithm, originates from Computer Vision.
- A recursive procedure of fitting and splitting.

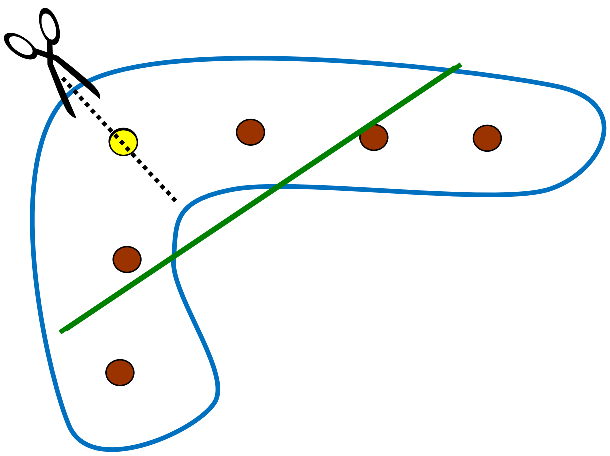
Let **S** be the set of all data points

Split

-
-
-

Merge

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



Line Extraction – Split and Merge (standard)

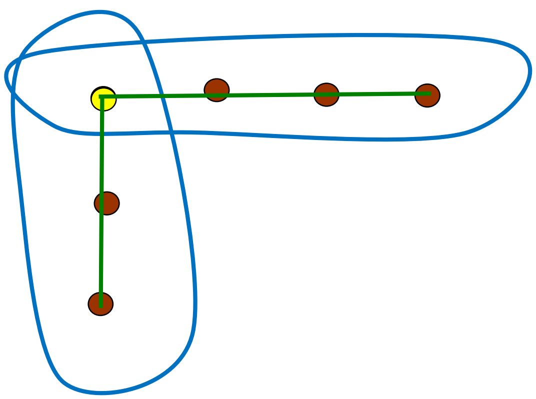
Let **S** be the set of all data points

Split

-
-
- If distance > threshold ⇨ split set & repeat with left and right point sets

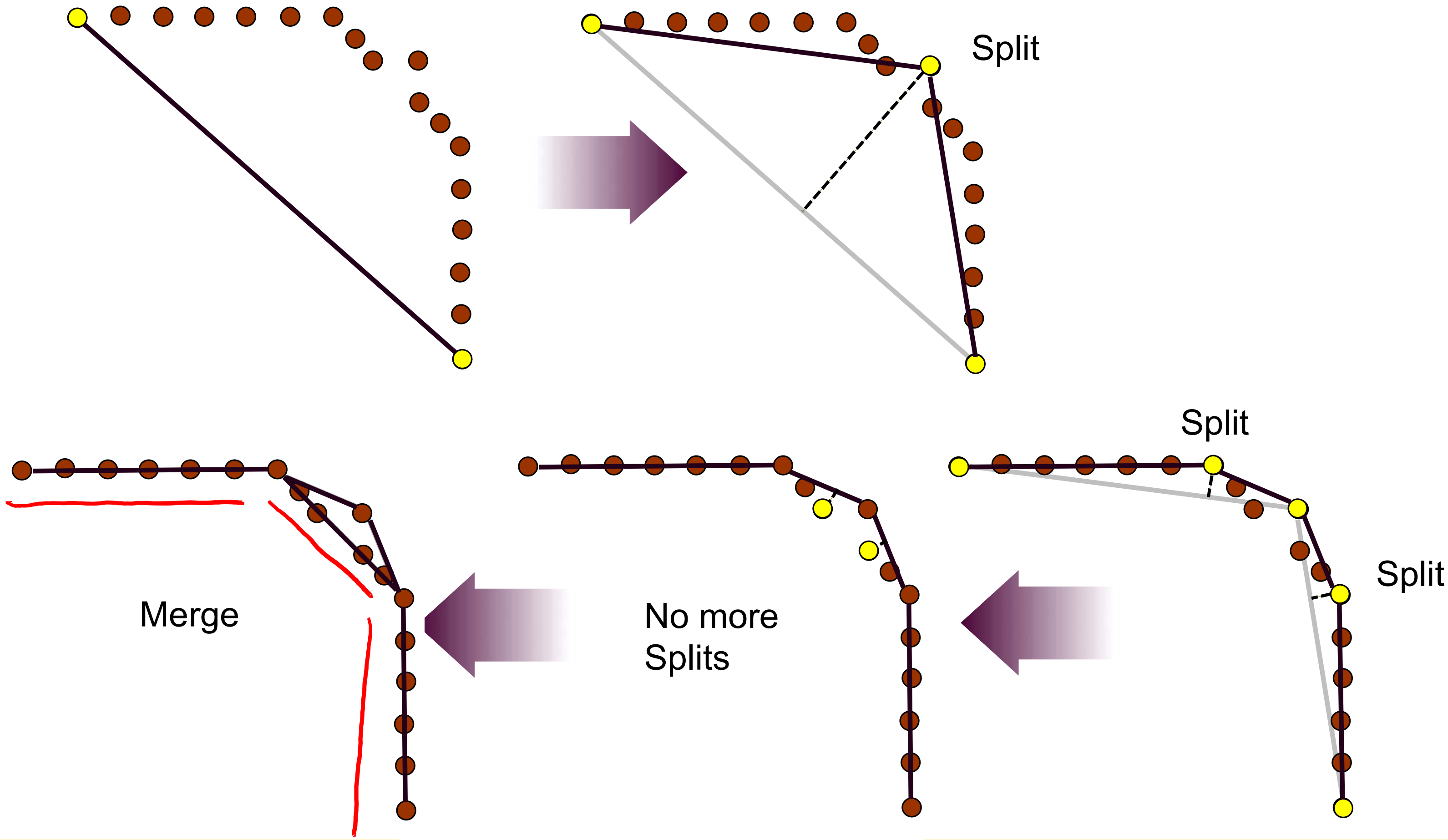
Merge

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



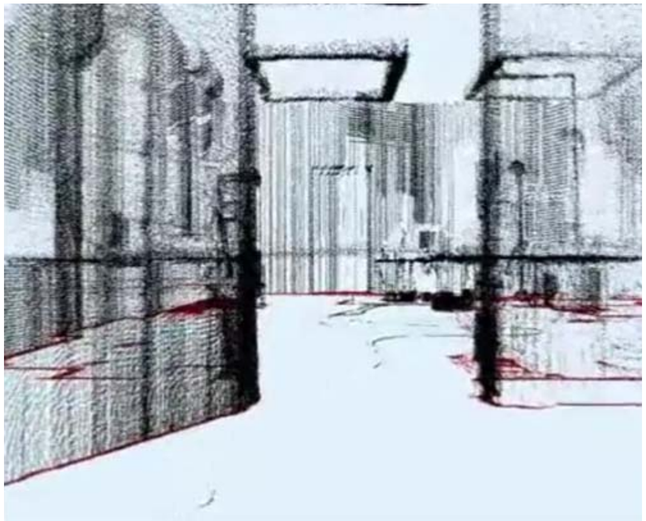
Line Extraction – Split and Merge (iterative end-point-fit)

- Iterative end-point-fit: simply connects the end points for line fitting



Line Extraction from a Point Cloud

- Extract lines from a point cloud (e.g. range scan)
- 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:
  - Split-and-merge
  - Linear regression
  - RANSAC
  - Hough-Transform



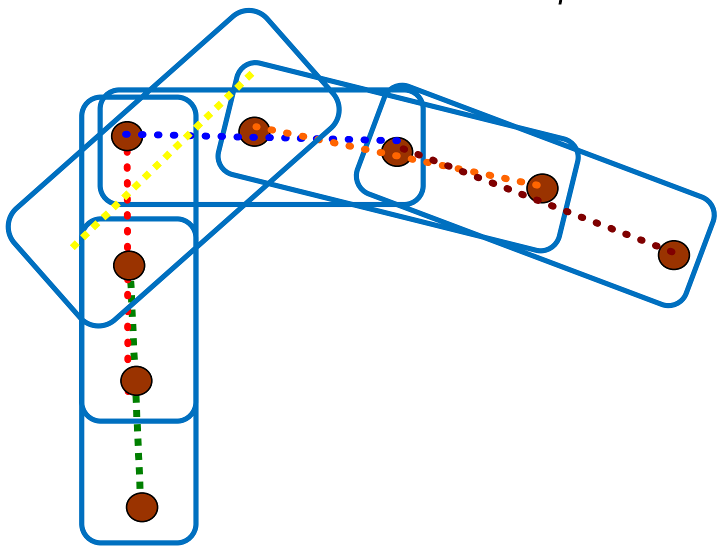
Line Extraction – Line Regression

- "Sliding window" of size N_f points
- Fit line-segment to all points in each window

Line-Regression

- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Merge overlapping line segments + re-compute line parameters for each segment

$N_f = 3$

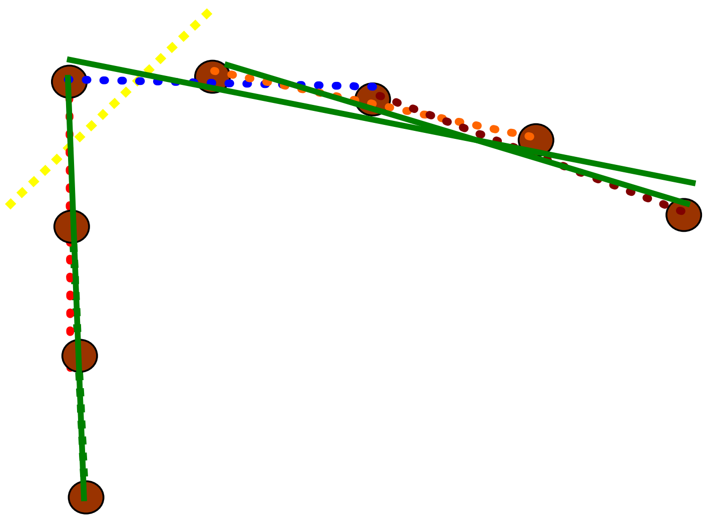


Line Extraction – Line Regression

$N_f = 3$

Line-Regression

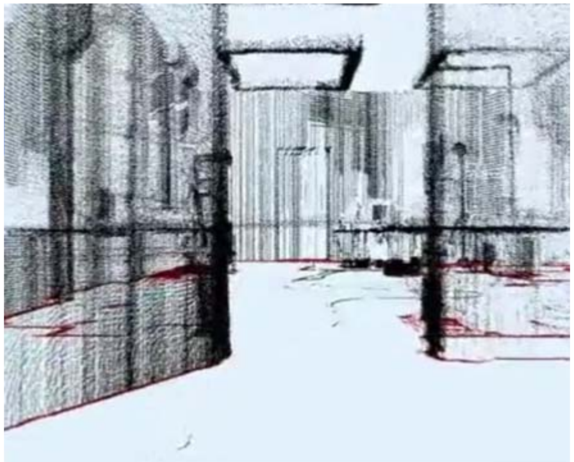
- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Merge overlapping line segments + re-compute line parameters for each segment



Line Extraction from a Point Cloud

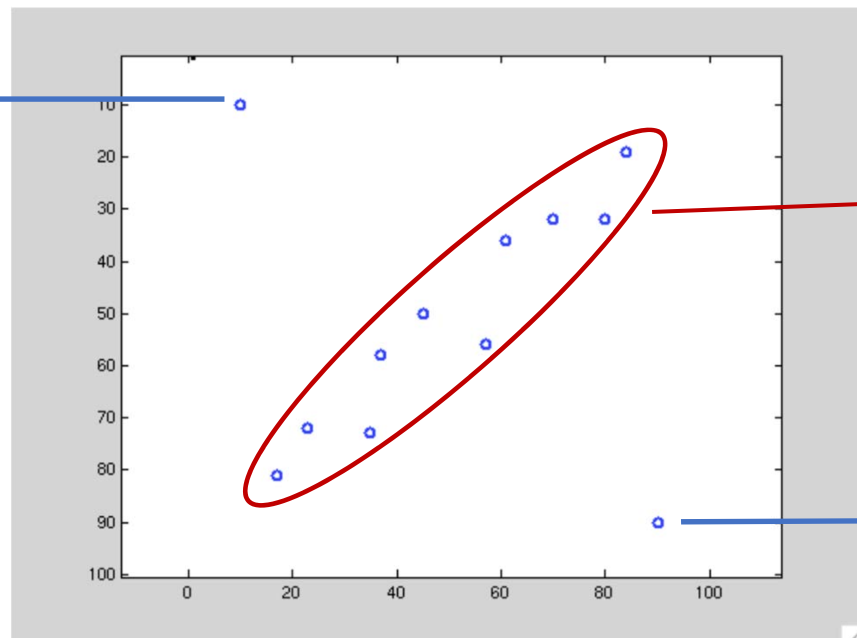
- Extract lines from a point cloud (e.g. range scan)
- Three main problems:
  - How many lines are there?
  - **Segmentation**: Which points belong to which line?
  - **Line Fitting/Extraction**: Given points that belong to a line, how to estimate the line parameters?

- Algorithms we will see:
  - Split-and-merge
  - Linear regression
  - RANSAC
  - Hough-Transform



Line Extraction – RANSAC

- RANSAC = RANdom Sample Consensus
- A generic & robust fitting algorithm of models **in the presence of outliers** (i.e. points which do not satisfy a model)



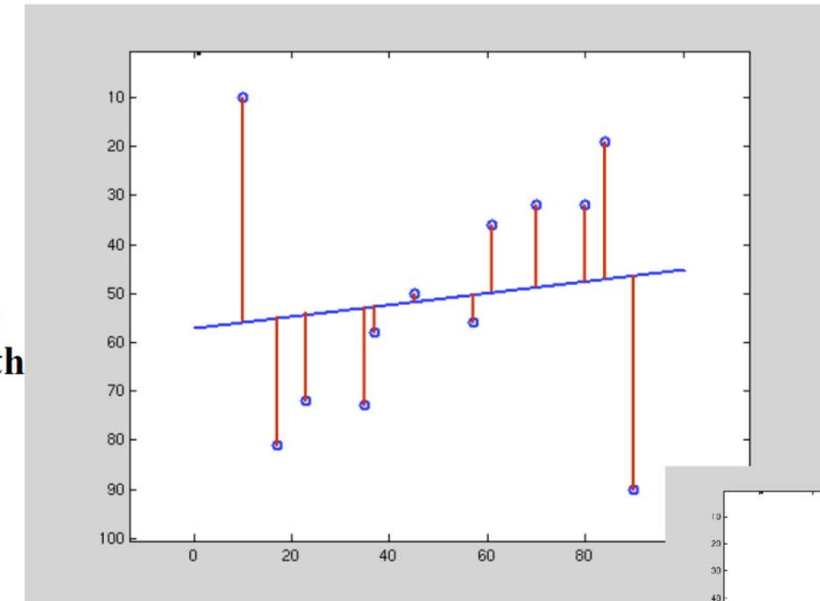
M. Fischler & R. C.Bolles. RANndom SAmple Consensus:
A paradigm for model fitting with applicatlons to image analysis and automated cartography. Graphics and Image Processing, **1981**.

Line Extraction – RANSAC

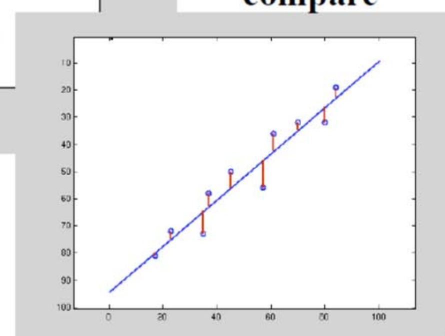
- RANSAC = RANdom Sample Consensus

Least squares estimation is sensitive to outliers, so that a few outliers can greatly skew the result.

Least squares regression with outliers



compare



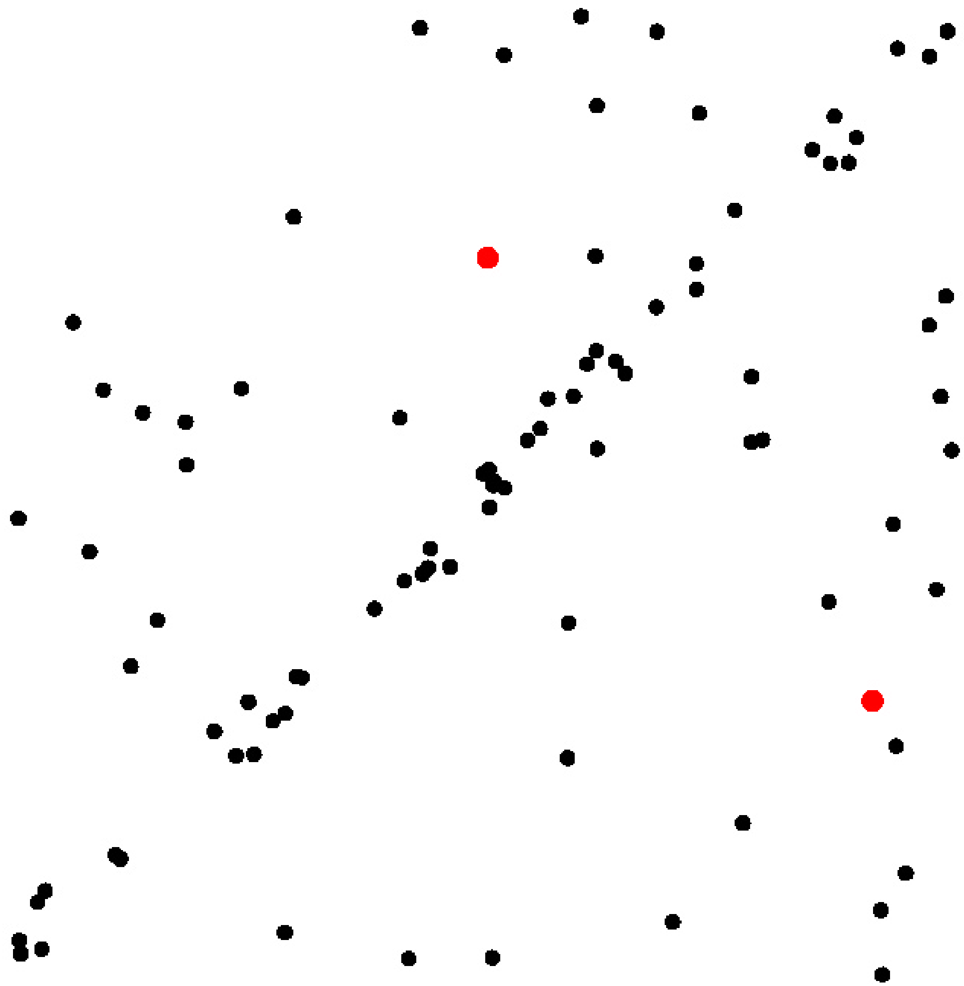
Solution: Estimation methods that are robust to outliers.

Line Extraction – RANSAC

- RANSAC = RANdom Sample Consensus
- Can be applied in general 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 data, feature matching, structure from motion, camera calibration, homography estimation, etc.
- RANSAC is **iterative** and **non-deterministic:** the probability to find a set free of outliers increases as more iterations are used
- Drawback: a non-deterministic method, results are different between runs.

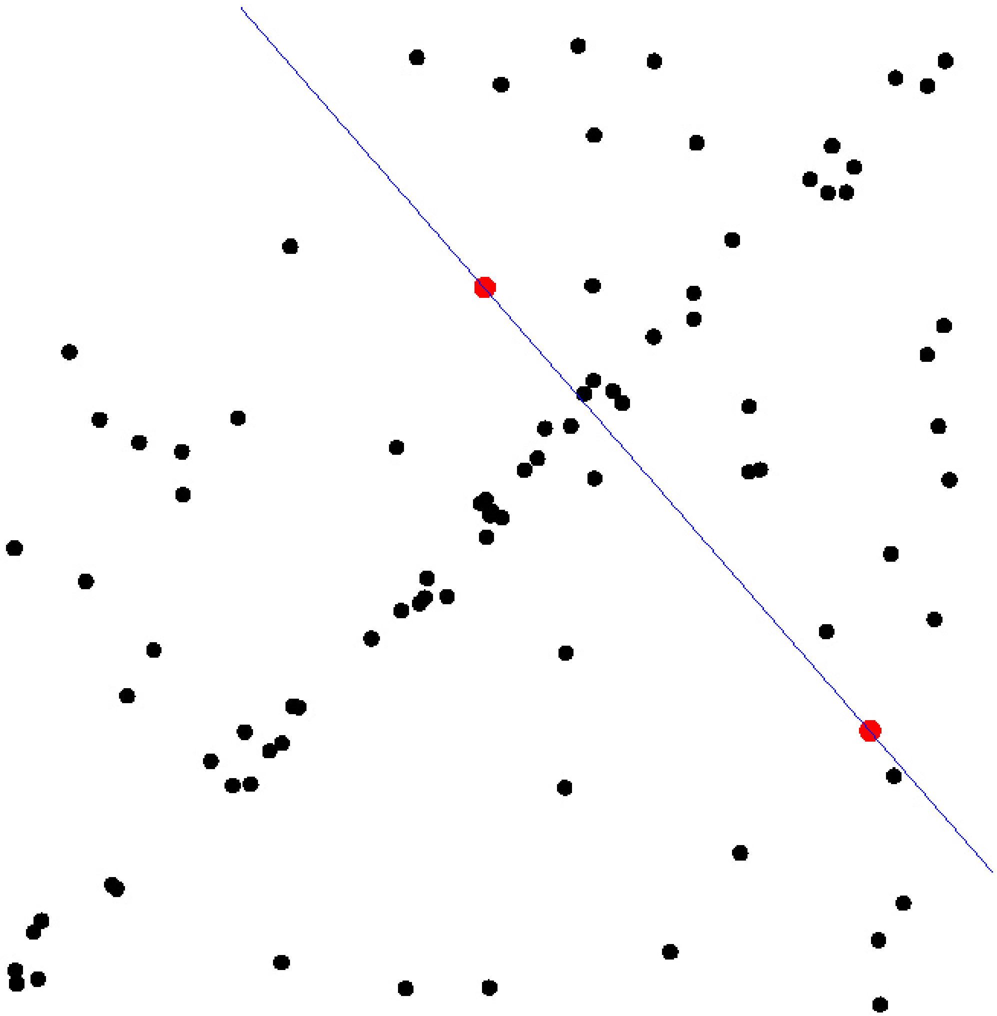
M. Fischler & R. C.Bolles. RANndom SAmple Consensus:
A paradigm for model fitting with applicatlons to image analysis and automated cartography. Graphics and Image Processing, **1981**.

Line Extraction – RANSAC



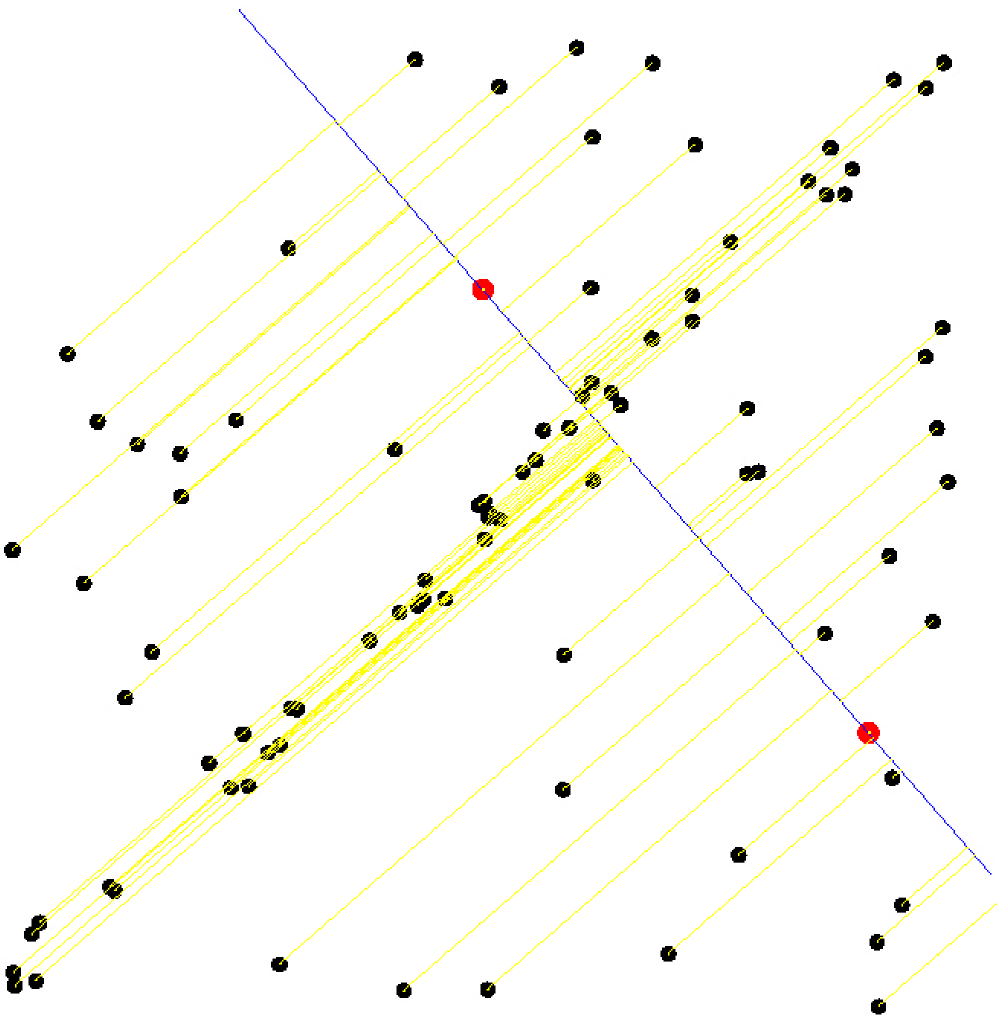
- Select a sample of two points at random

Line Extraction – RANSAC



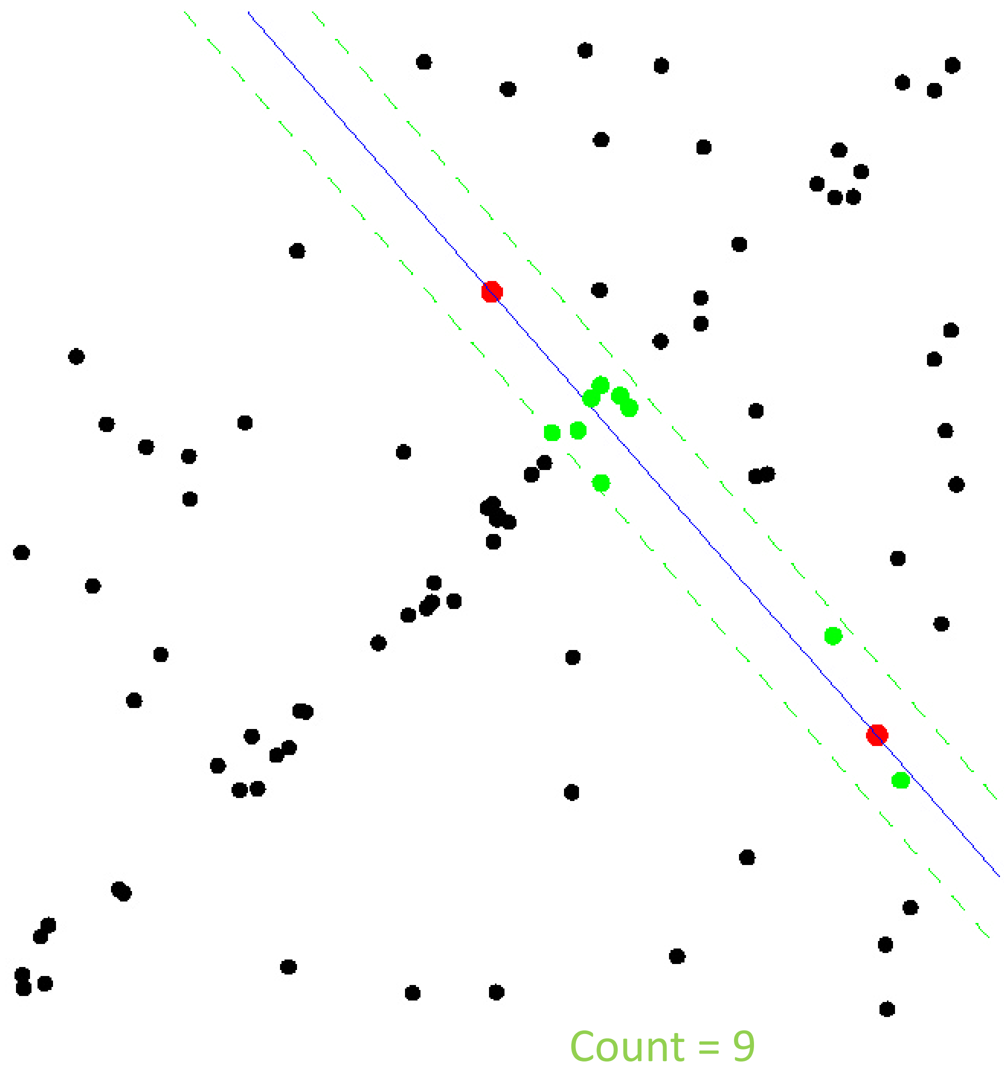
- Select a sample of two points at random
- Calculate model parameters that fit the data in the sample

Line Extraction – RANSAC



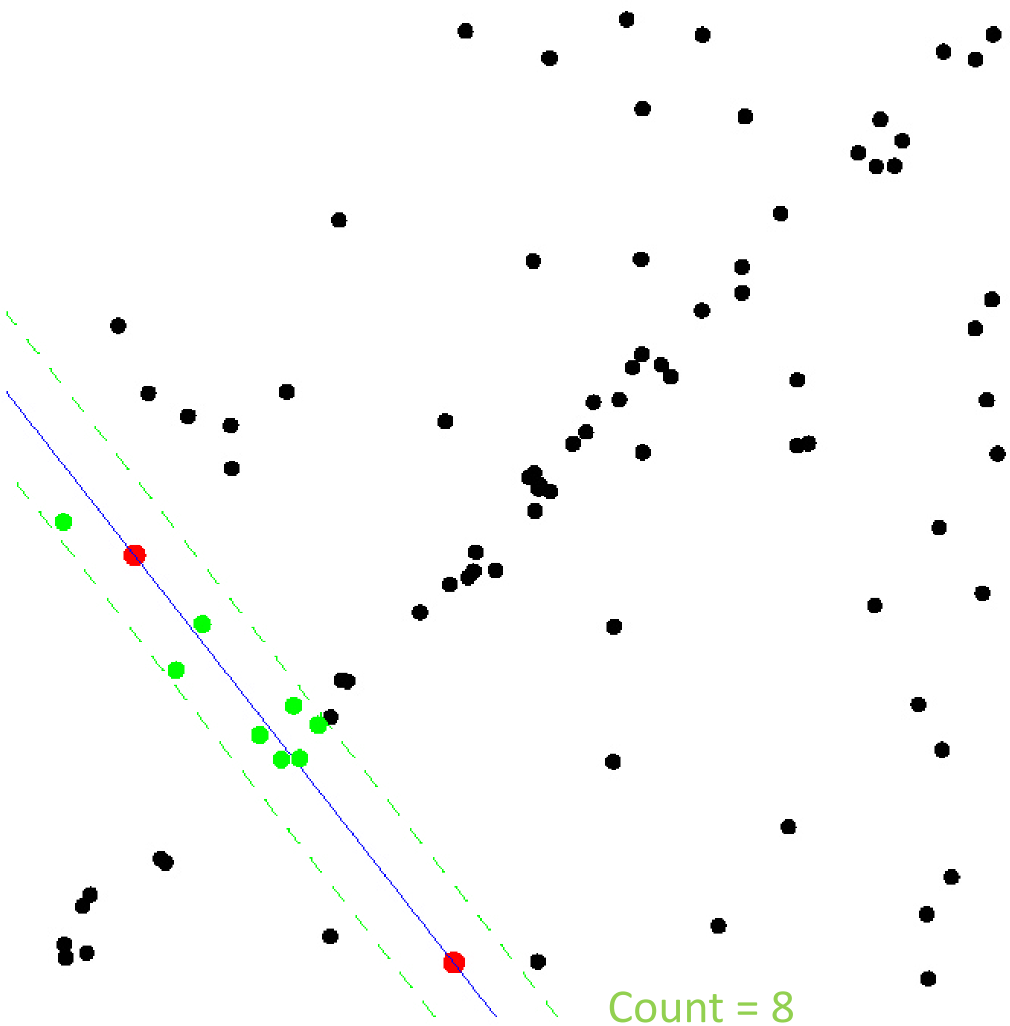
- Select a sample of two points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point

Line Extraction – RANSAC



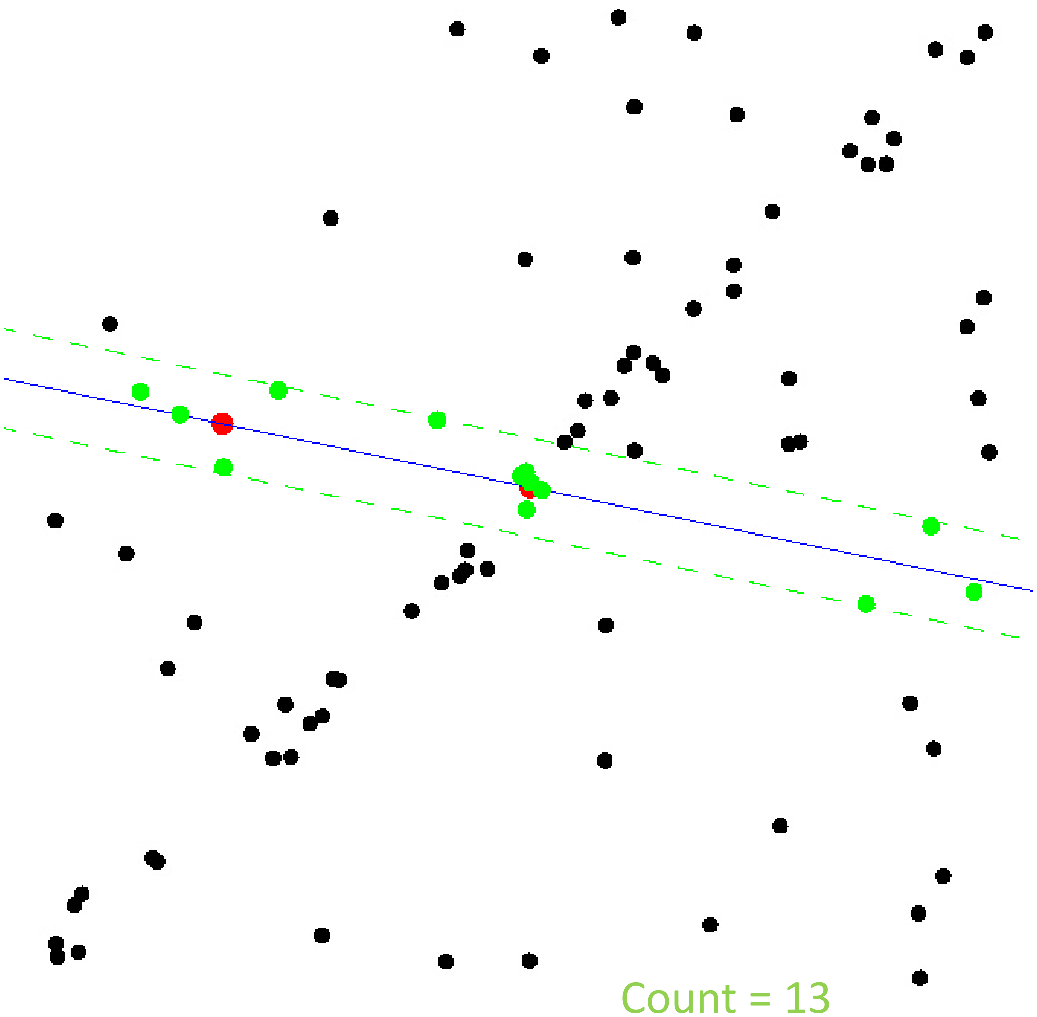
- Select a sample of two points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that supports current hypothesis

Line Extraction – RANSAC



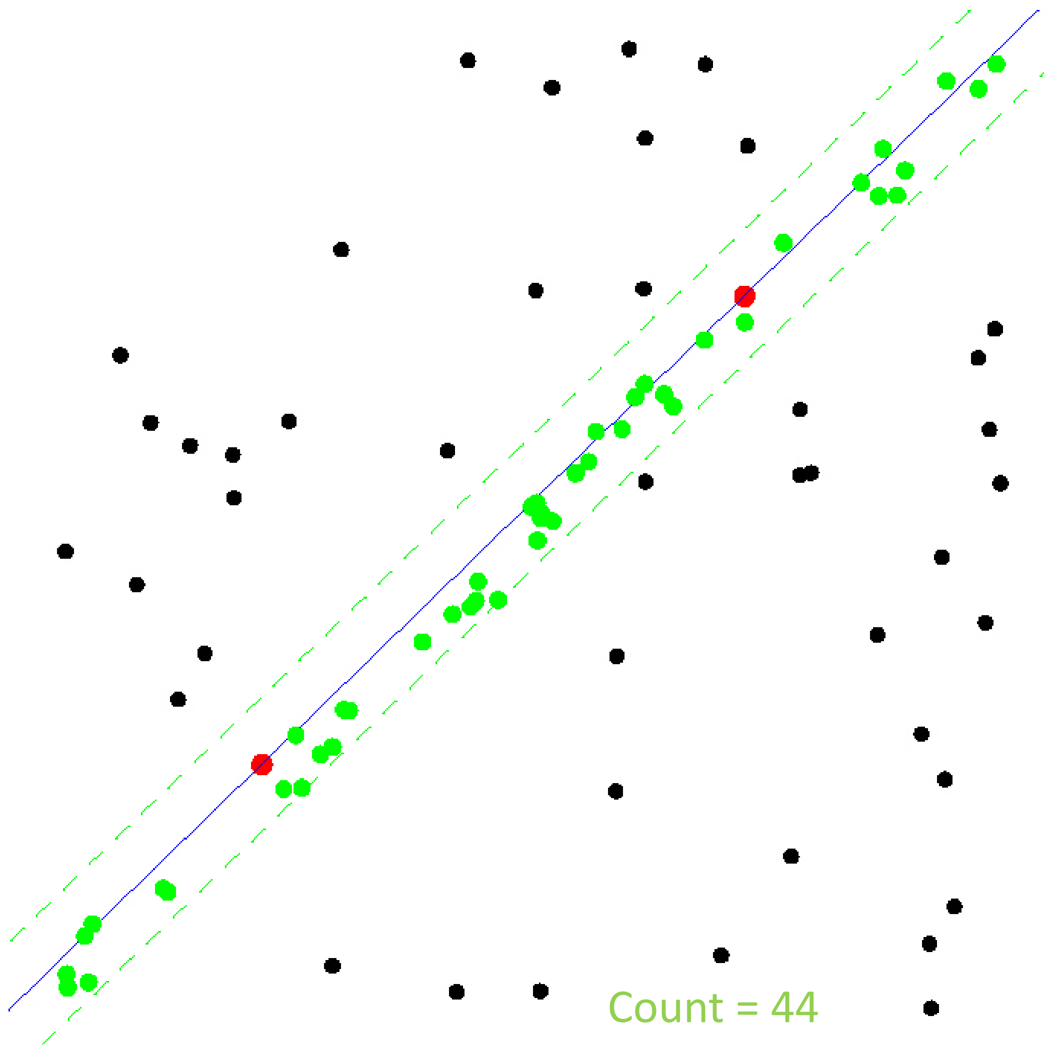
- Select a sample of two points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that supports current hypothesis
- Repeat sampling

Line Extraction – RANSAC



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

Line Extraction – RANSAC



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

Set with the maximum number of inliers obtained after *k* iterations

Line Extraction – RANSAC

How many iterations does RANSAC need? ***k* =?**

- Ideally: check all possible combinations of **2** points in a dataset of ***N*** points
- Number of all pairwise combinations: ***N(N-1)/2***
  - computationally infeasible if ***N*** is too large.
    example:
    10’000 points to fit a line through a need to check all 10’000 x 9’999/2 = **50 million** combinations!
- Do we really need to check all combinations or can we stop after some 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

Line Extraction – RANSAC

How many iterations does RANSAC need? *k* **=?**

- *N*: total number of data points
- *w*: number of inliers / *N*
  - *w* is the fraction of inliers in the dataset, the probability of selecting an inlier-point out of the dataset
- *p*: the desired probability of success
- The number of RANSAC iterations needed is

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

- **Example:** if we want a probability of success *p* = 99%, and we know that *w* = 50% → *k* = 16 iterations
  - dramatically fewer than the number of all possible combinations!

- In practice, we need only a rough estimate of *w*. There are more advanced variants of RANSAC

Line Extraction – RANSAC

RANSAC is not only for line extraction

```
Given:
    data – a set of observations
    model – a model to explain observed data points
    n – minimum number of data points required to estimate model parameters
    k – maximum number of iterations allowed in the algorithm
    t – threshold value to determine data points that are fit well by model
    d – number of close data points required to assert that a model fits well to data

Return:
    bestFit – model parameters which best fit the data (or nul if no good model is found)

iterations = 0
bestFit = nul
bestErr = something really large
while iterations < k {
    maybeInliers = n randomly selected values from data
    maybeModel = model parameters fitted to maybeInliers
    alsoInliers = empty set
    for every point in data not in maybeInliers {
        if point fits maybeModel with an error smaller than t
             add point to alsoInliers
    }
    if the number of elements in alsoInliers is > d {
        % this implies that we may have found a good model
        % now test how good it is
        betterModel = model parameters fitted to all points in maybeInliers and alsoInliers
        thisErr = a measure of how well betterModel fits these points
        if thisErr < bestErr {
            bestFit = betterModel
            bestErr = thisErr
        }
    }
    increment iterations
}
return bestFit
```

wikipedia

- Thank you!