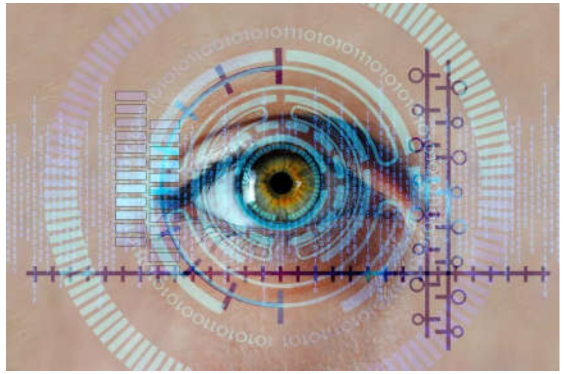
北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY



Computer Vision

Lecture 5 Line Detection

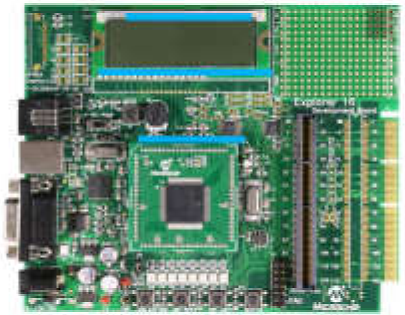
School of Computer Science and Technology

Ying Fu

fuying@bit.edu.cn

Line Detection

- Why detect lines? Many objects characterized by presence of straight lines


- Wait, why aren't we done just by running edge detection?

Outline

- Hough transform
- RANSAC

Intro to Hough transform

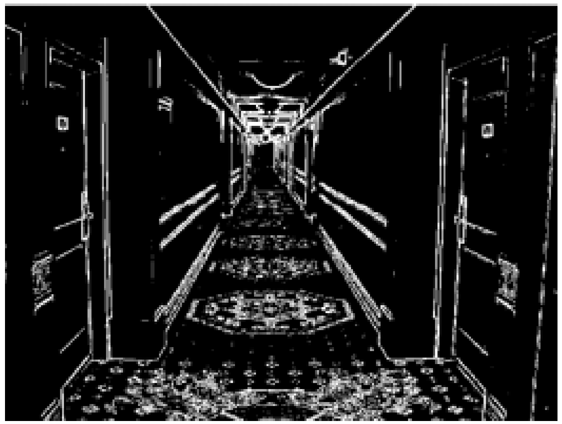
- The Hough transform (HT) can be used to detect lines.
- It was introduced in 1962 (Hough 1962) and first used to find lines in images a decade later (Duda1972).
- Our goal with the Hough Transform is to find the location of lines in images.
- Hough transform can detect lines, circles and other structures ONLY if their parametric equation is known.
- It can give robust detection under noise and partial occlusion

Prior to Hough transform

- Assume that we have performed edge detection, for example, by thresholding the gradient magnitude image.
- Thus, we have some pixels that may partially describe the boundary of some objects.

Input Image

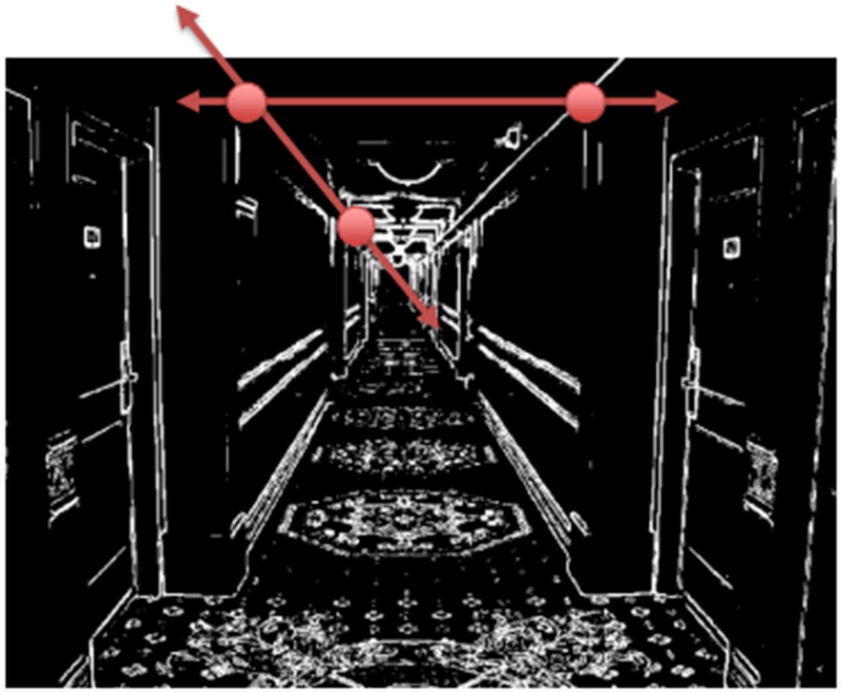
Image Gradients



Edge map(binary image)

Naïve Line Detection

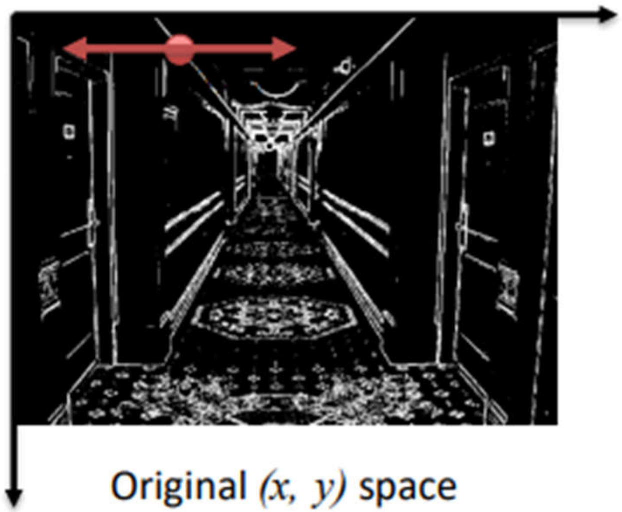
- For every pair of edge pixels
  - Compute equation of line
  - Check if other pixels satisfy equation
- Complexity?
  - O(N^2) for an image with N edge pixels
- We can do better with the Hough Transform!



Edge map (binary image)

Detecting lines using Hough transform

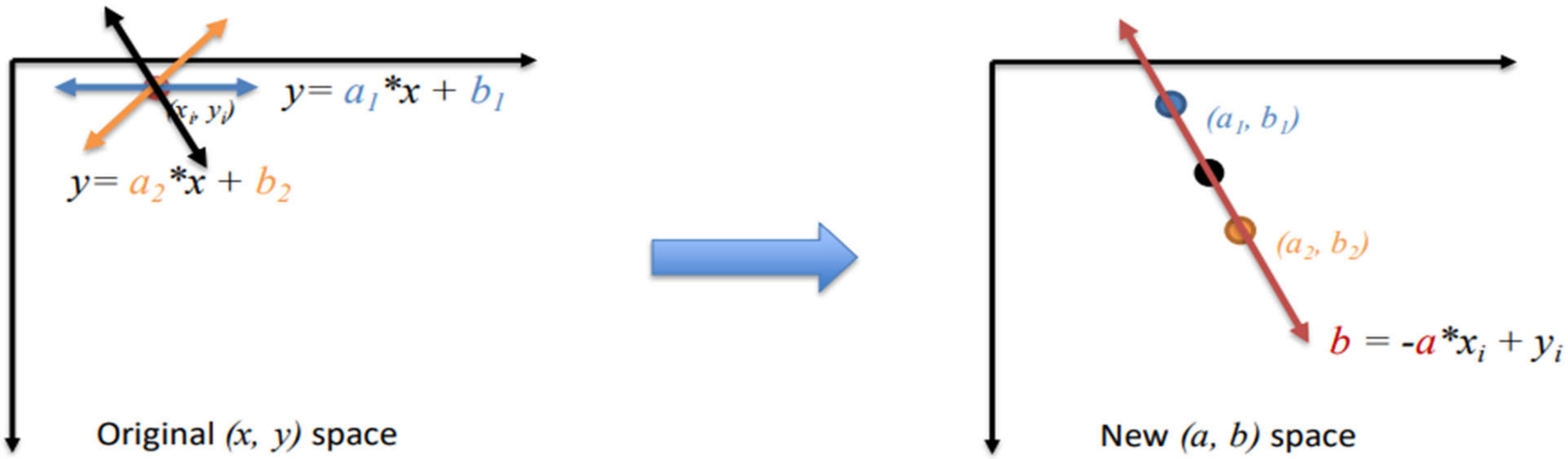
- We wish to find sets of pixels that make up straight lines.
- First step is to transform edge points into a new space.
- Consider an edge point of known coordinates (x_i, y_i):
  - –There are many potential lines passing through the point (x_i, y_i).
- This family of lines have the form $y_i = a * x_i + b$.



Original *(x, y)* space

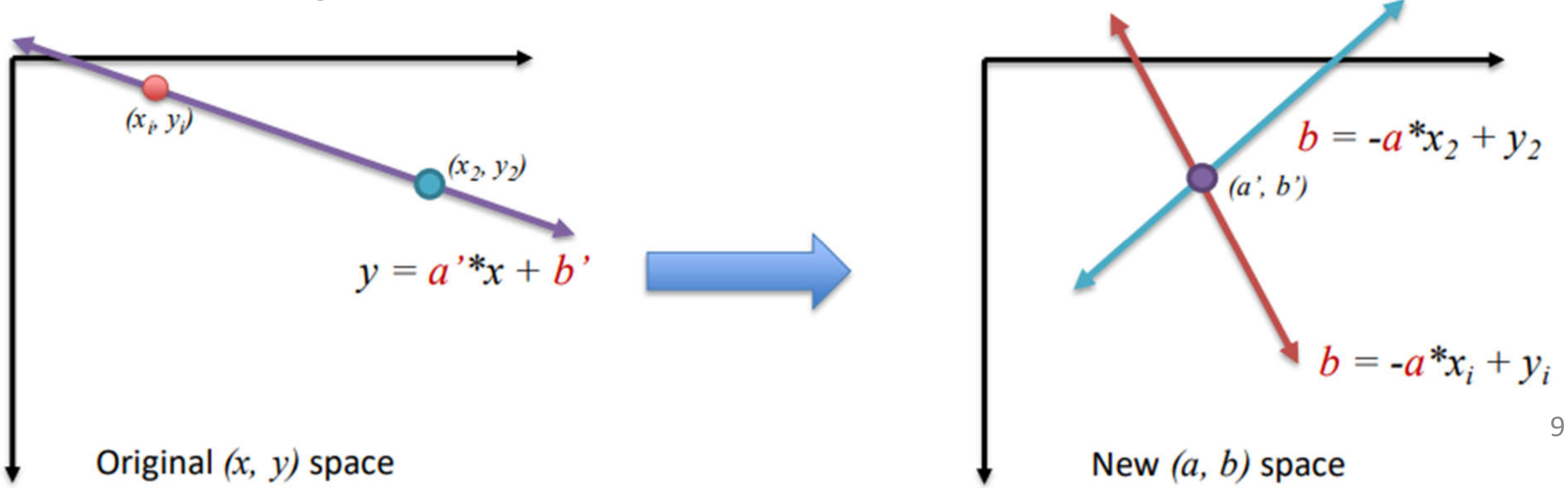
Detecting lines using Hough transform

- This family of lines have the form $y_i = a * x_i + b$.
- Note (x_i, y_i) are constants, while (a, b) can change. This gives rise to a new space where (a, b) are the variables.
- That means, a point (x_i, y_i) transforms into a line in the (a, b) space: $b = -x_i * a + y_i$.



Detecting lines using Hough transform

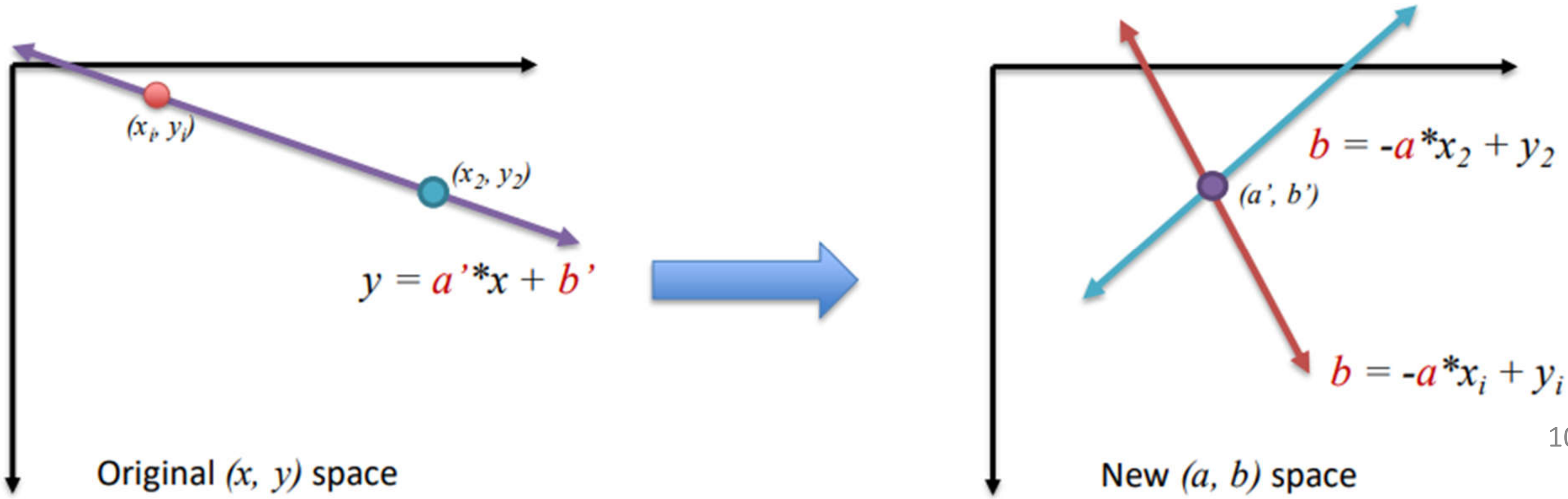
- This family of lines have the form $y_i = a * x_i + b$.
- Note (x_i, y_i) are constants, while (a, b) can change. This gives rise to a new space where (a, b) are the variables.
- That means, a point (x_i, y_i) transforms into a line in the (a, b) space: $b = -x_i * a + y_i$.
- Another edge point (x_2, y_2) will give rise to another line in the (a, b) space.



Detecting lines using Hough transform

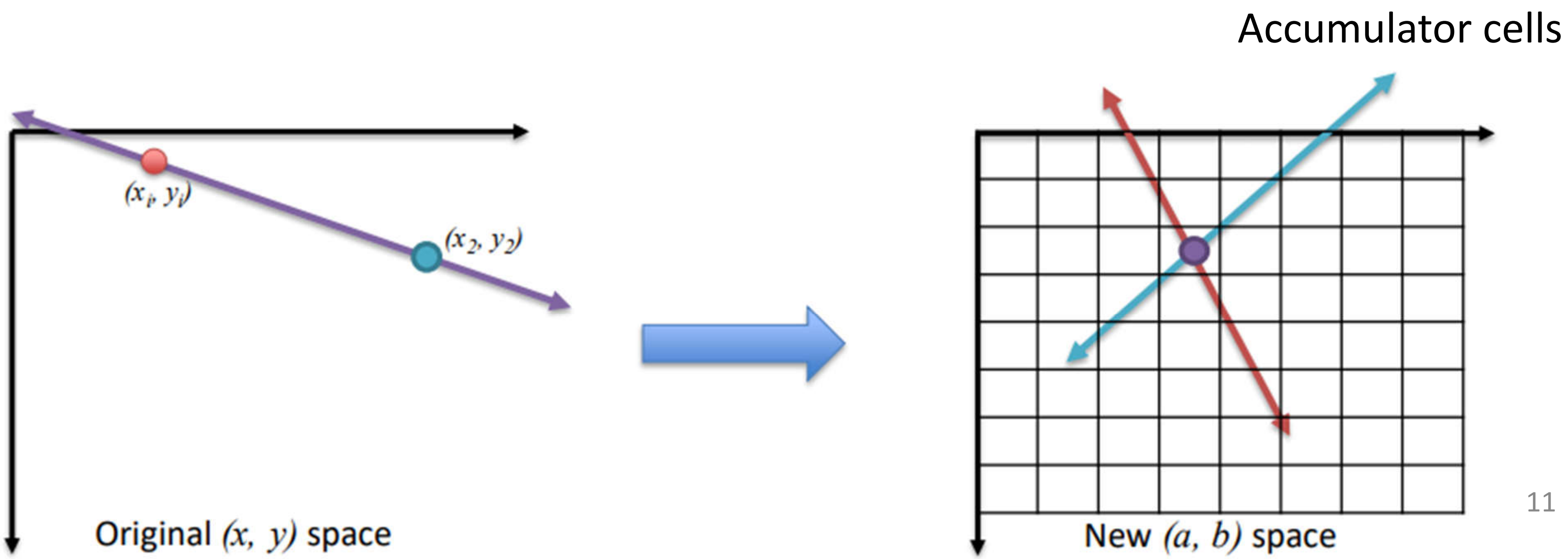
- Colinear points in the (x, y) space transform into lines in the (a, b) space that intersect at a single point (a', b').
- We can detect lines by finding such intersection points (a', b') in the (a, b) space.
- Our resulting line equation in the original space is

$$y = a' * x + b'.$$



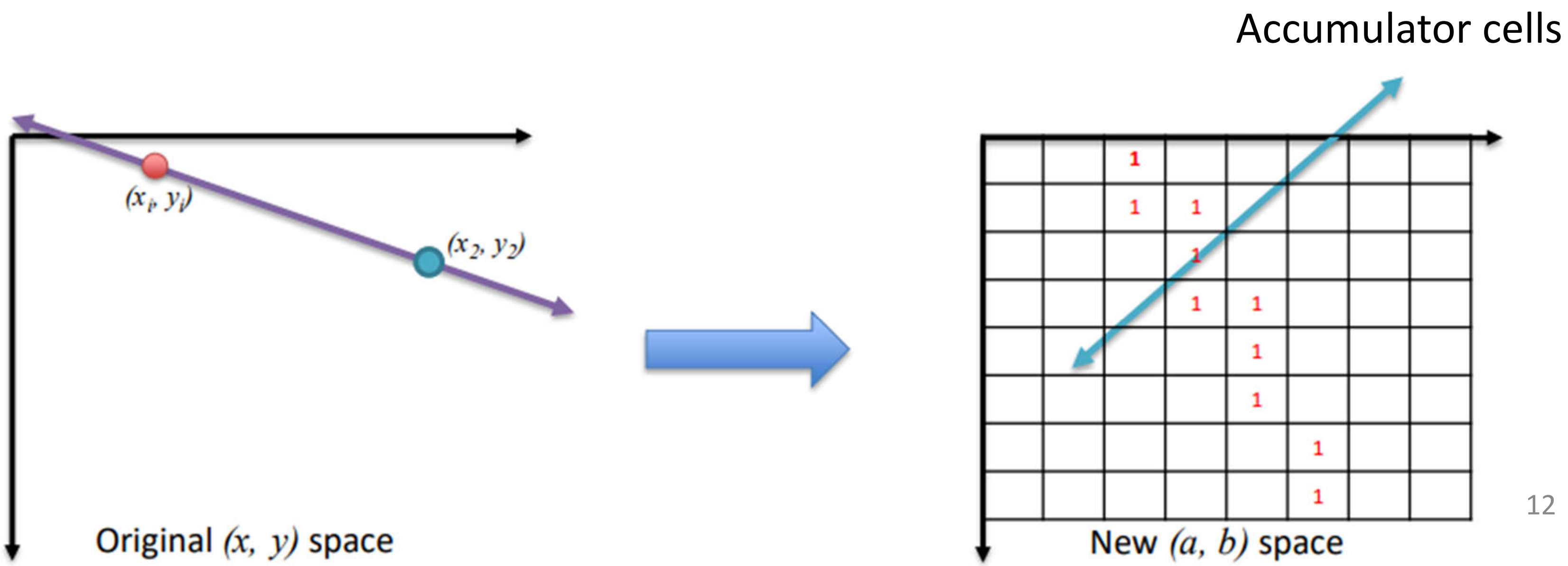
Detecting lines using Hough transform

- We efficiently find the intersection points in the (a, b) space by quantizing it into cells.
- Instead of transforming a point to an explicit line, we vote on the discrete cells that are 'activated' by the transformed line in (a, b) .



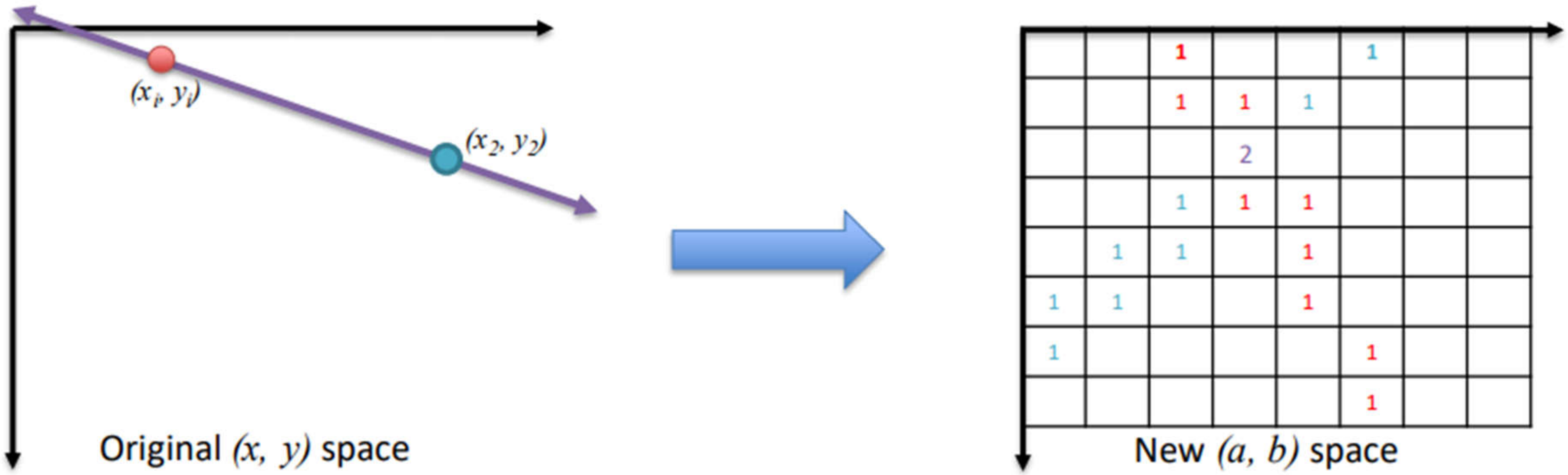
Detecting lines using Hough transform

- We efficiently find the intersection points in the (a, b) space by quantizing it into cells.
- Instead of transforming a point to an explicit line, we vote on the discrete cells that are ‘activated’ by the transformed line in (a, b) .



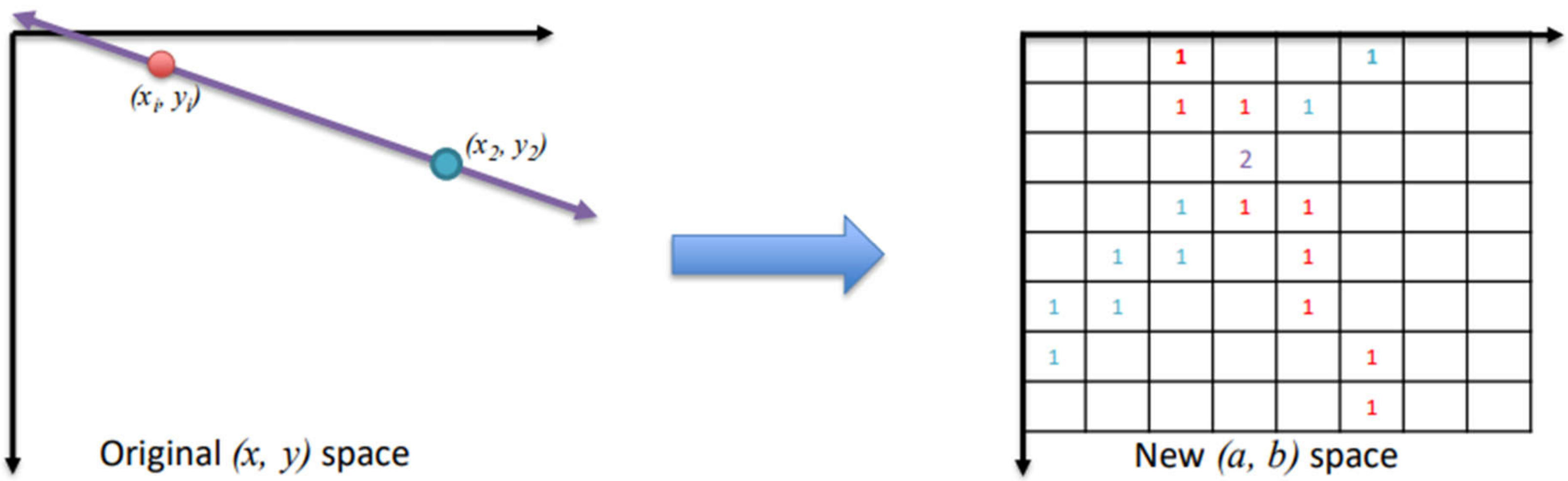
Detecting lines using Hough transform

- We efficiently find the intersection points in the (a, b) space by quantizing it into cells.
- Instead of transforming a point to an explicit line, we vote on the discrete cells that are 'activated' by the transformed line in (a, b).
- Cells that receive more than a certain number of votes are assumed to correspond to lines in (x, y) space.



Hough Transform Algorithm

- For each (x, y) edge point:
  - Vote on cells that satisfy the corresponding (a, b) line equation
- Find cells with more votes than threshold.
- Complexity?
  - Linear on number of edge points
  - Linear on number of accumulator cells



Output of Hough transform

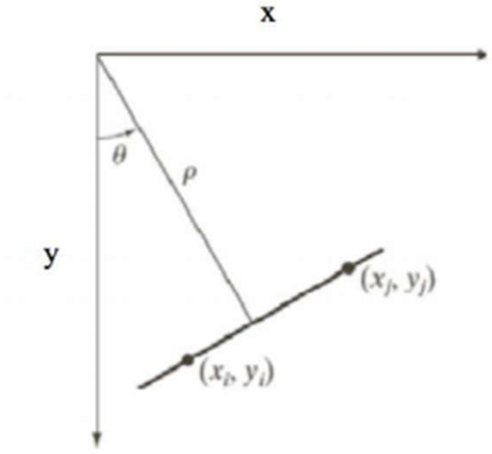
- Here are the top 20 most voted lines in the image

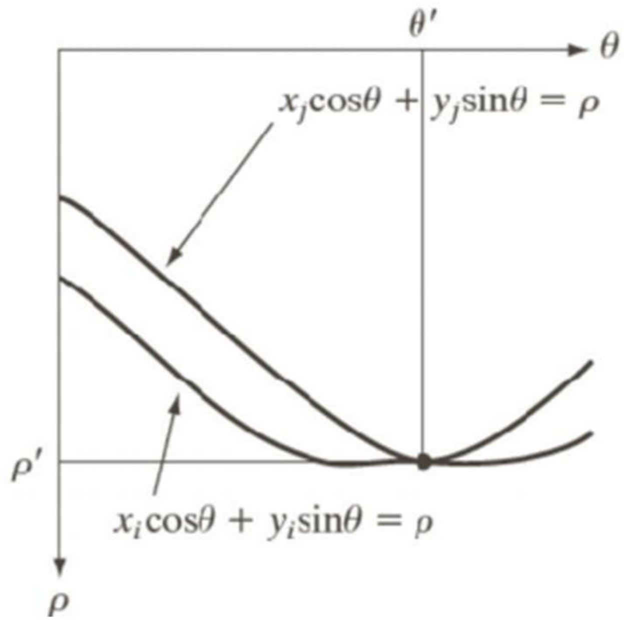
Other Hough transformations

- We can represent lines as polar coordinates instead of $y = a * x + b$
- Polar coordinate representation:

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

- A vertical line will have $\theta = 90^{\circ}$ and ρ equal to the intercept with the x-axis.
- A horizontal line will have $\theta = 0^{\circ}$ and ρ equal to the intercept with the y-axis.
- Note that lines in (x, y) space are not lines in (ρ, θ) space, unlike (a, b) space



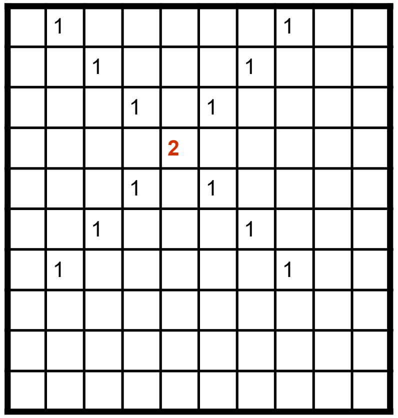


Problems with parameterization

- Euclidean coordinate

How big does the accumulator need to be for the parameterization (a,b) ?

A(a,b)



The space of a is huge!

$$-\infty \leq a \leq \infty$$

The space of b is huge!

$$-\infty \leq b \leq \infty$$

Better Parameterization

- Polar coordinate

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

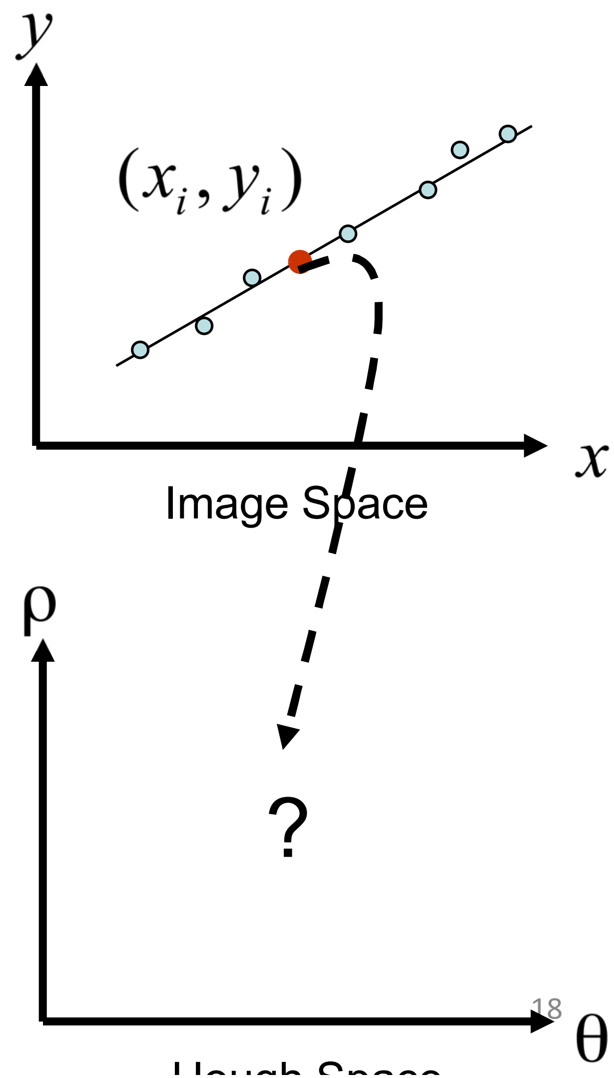
Given points (x_i, y_i) find (ρ, θ)

Hough Space Sinusoid

$$0 \leq \theta \leq 2\pi$$

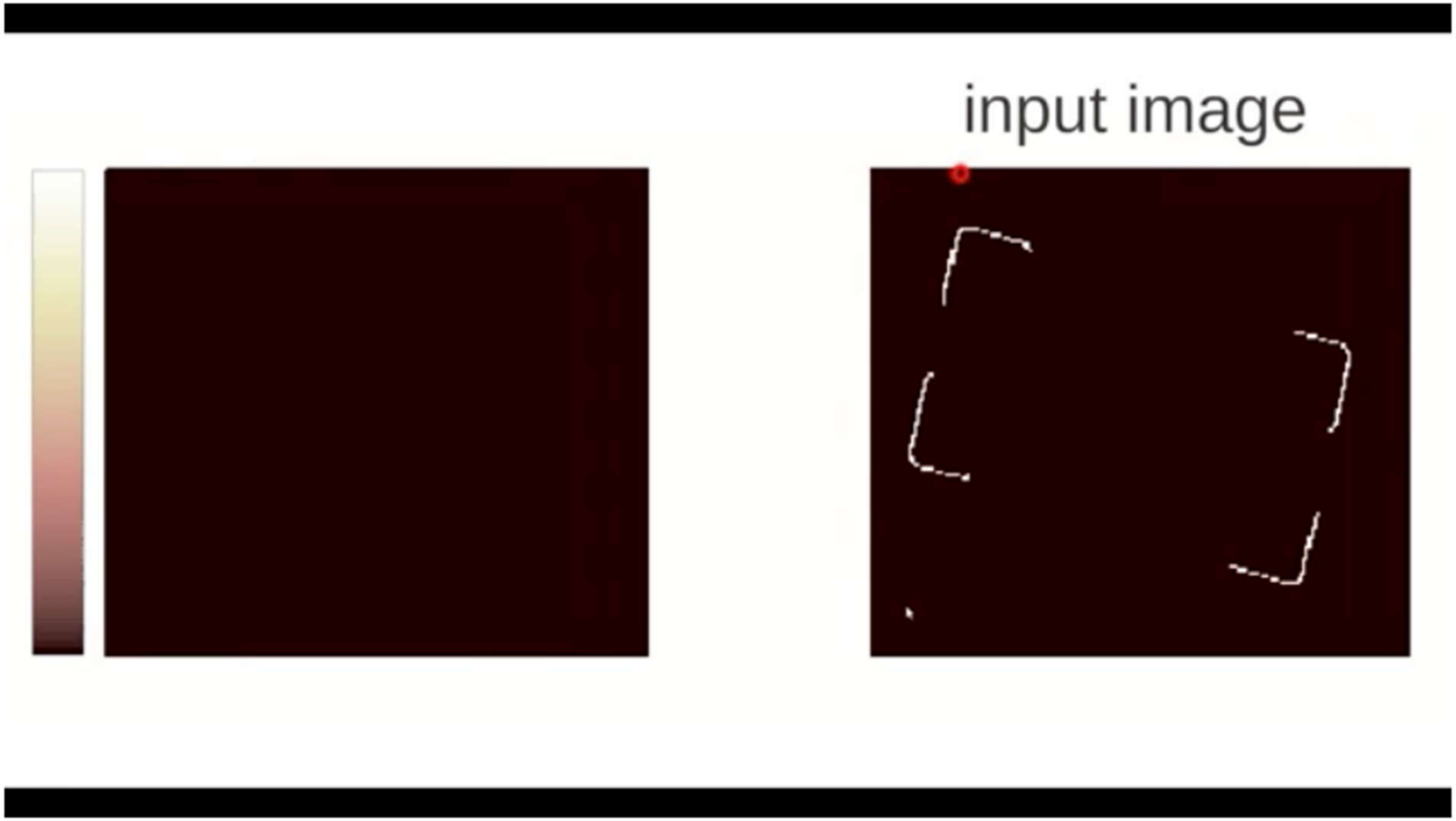
$$0 \leq \rho \leq \rho_{max}$$

(Finite Accumulator Array Size)



Example video

- Demo



Remarks

- Advantages:
  - Conceptually simple.
  - Easy implementation.
  - Handles missing and occluded data very gracefully.
  - Can be adapted to many types of forms, not just lines.
- Disadvantages:
  - Computationally complex for objects with many parameters.
  - Looks for only one single type of object.
  - Can be “fooled” by “apparent lines”.
  - The length and the position of a line segment cannot be determined.
  - Co-linear line segments cannot be separated.

Outline

- Hough transform
- RANSAC

RANSAC [Fischler & Bolles 1981]

- RANdom SAmple Consensus
- Approach: we want to avoid the impact of outliers, so let's look for "inliers", and use only those
- Intuition: if an outlier is chosen to compute the current fit, then the resulting line won't have much support from rest of the points.

RANSAC loop

Repeat for k iterations:

① Randomly select a seed group of points on which to perform a model estimate (e.g., a group of edge points)

② Compute model parameters from seed group

③ Calculate distances and find inliers to this model

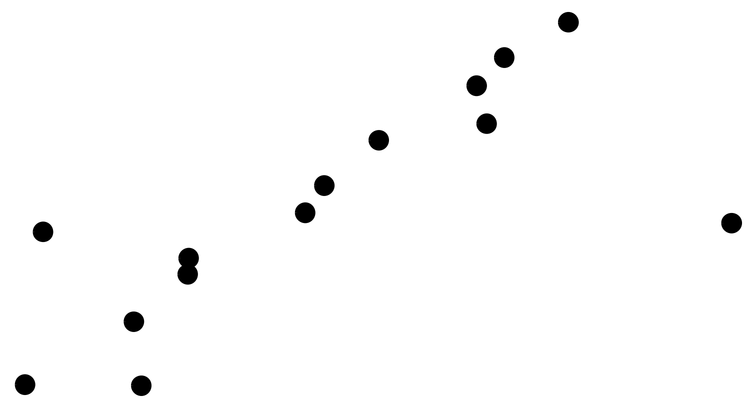
④ If the number of inliers is sufficiently large, re-compute least-squares estimate of model on all of the inliers

–Keep the model with the largest number of inliers

https://zhuanlan.zhihu.com/p/36301702

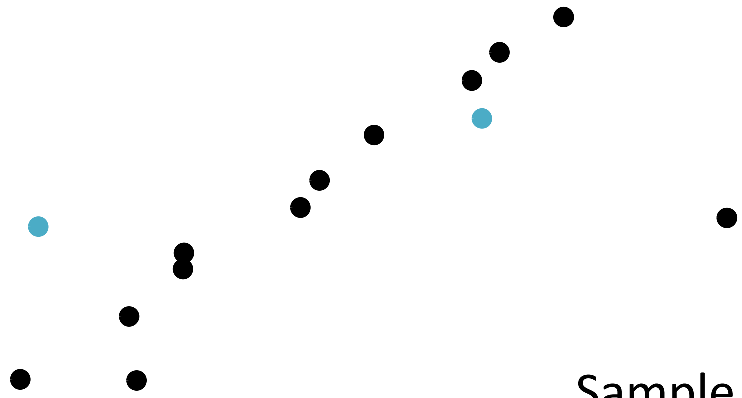
RANSAC Line Fitting Example

- Task: Estimate the best line
  - How many points do we need to estimate the line?



RANSAC Line Fitting Example

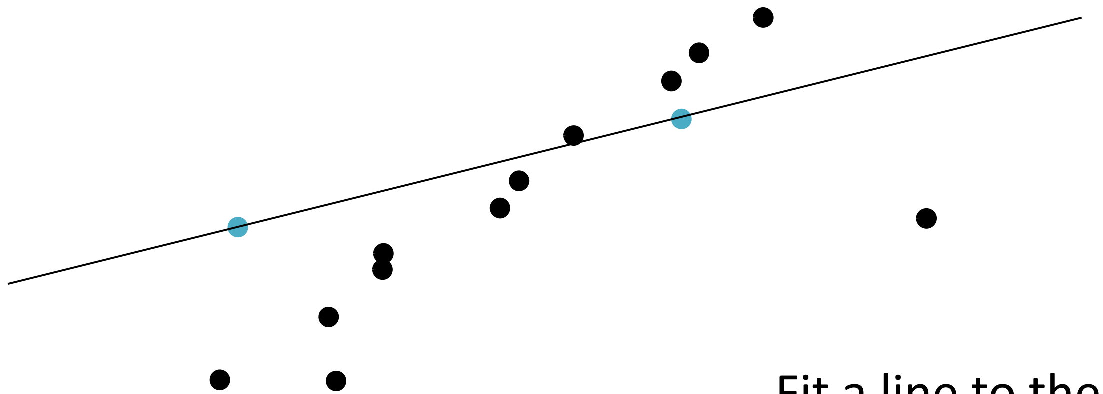
- Task: Estimate the best line



Sample two points

RANSAC Line Fitting Example

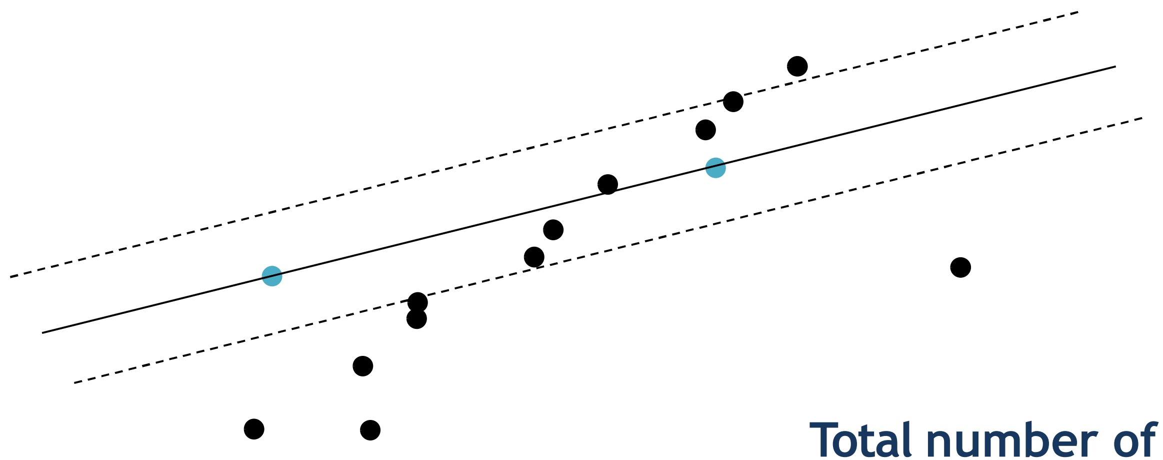
- Task: Estimate the best line



Fit a line to them

RANSAC Line Fitting Example

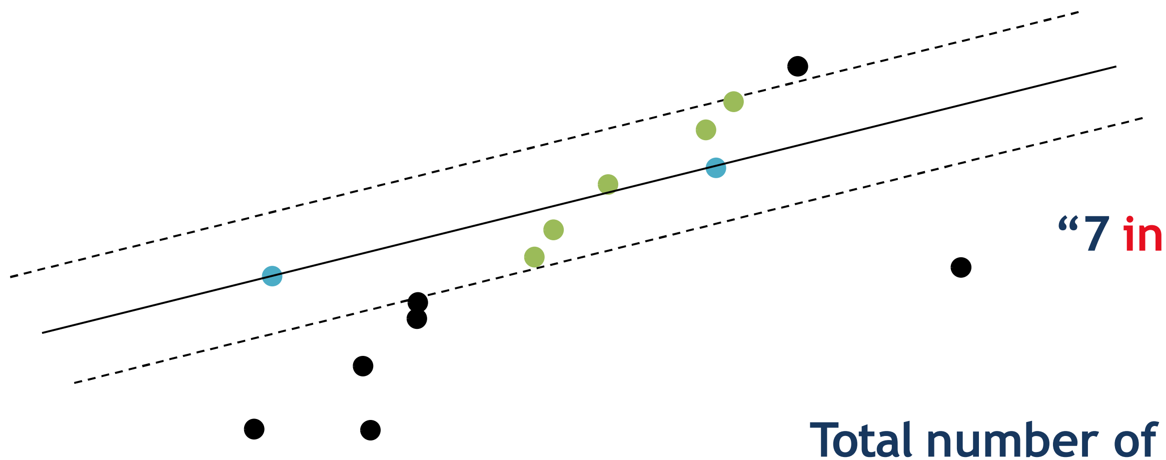
- Task: Estimate the best line



Total number of points within a threshold of line.

RANSAC Line Fitting Example

- Task: Estimate the best line

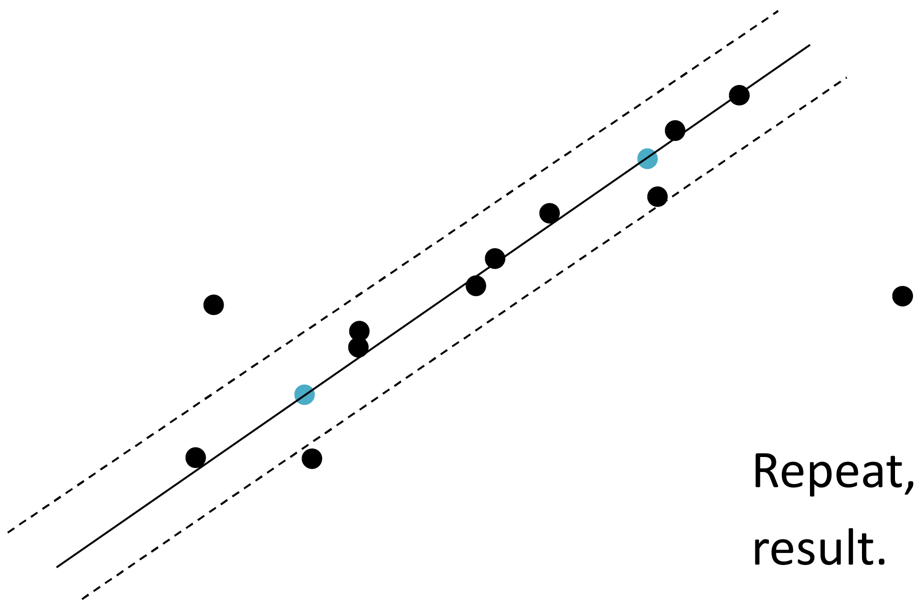


"7 inlier points"

Total number of points within a threshold of line.

RANSAC Line Fitting Example

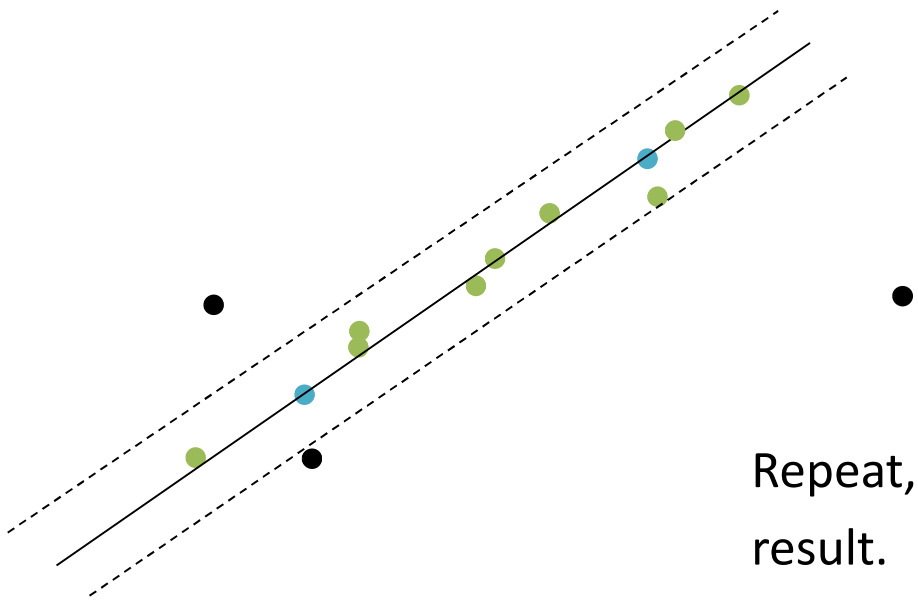
- Task: Estimate the best line



Repeat, until we get a good result.

RANSAC Line Fitting Example

- Task: Estimate the best line
  - How many points do we need to estimate the line?



“11 inlier points”

Repeat, until we get a good result.

RANSAC algorithm

Algorithm 15.4: RANSAC: fitting lines using random sample consensus

Determine:
  n — the smallest number of points required
  k — the number of iterations required
  t — the threshold used to identify a point that fits well
  d — the number of nearby points required
    to assert a model fits well
Until k iterations have occurred
  Draw a sample of n points from the data
    uniformly and at random
  Fit to that set of n points
  For each data point outside the sample
    Test the distance from the point to the line
      against t; if the distance from the point to the line
      is less than t, the point is close
  end
  If there are d or more points close to the line
    then there is a good fit. Refit the line using all
    these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion

RANSAC: How many iterations " k "?

- How many samples are needed?
  - Suppose w is fraction of inliers (points from line).
  - n points needed to define hypothesis (2 for lines)
  - k samples chosen.
- Prob. that a single sample of n points is correct: w^n
- Prob. that a single sample of n points fails: $1 - w^n$
- Prob. that all k samples fail is: $(1 - w^n)^k$
- Prob. that at least one of the k samples is correct: $1 - (1 - w^n)^k$

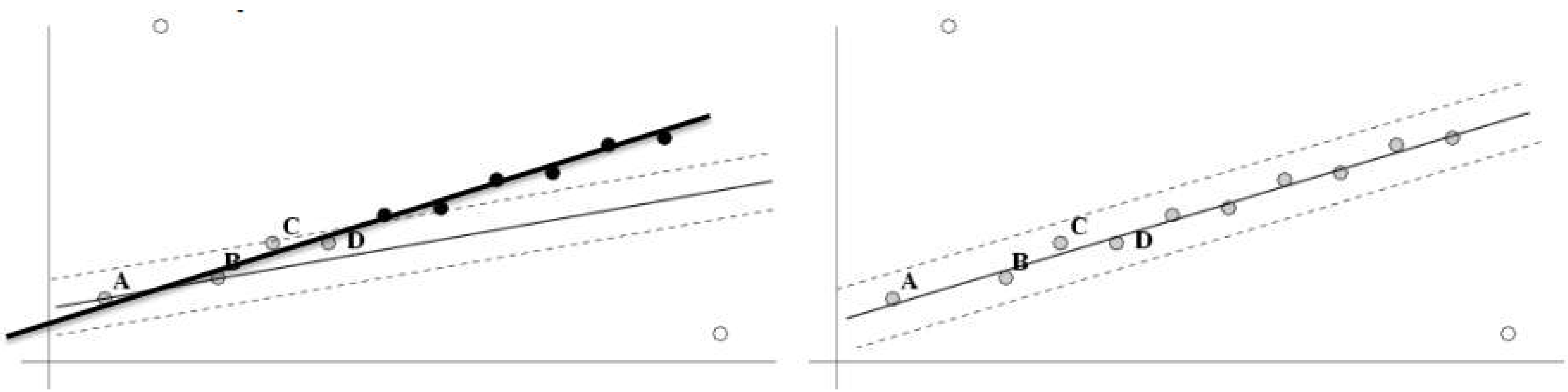
=> Choose k high enough to keep this below desired failure rate.

RANSAC: Computed k(p=0.99)

Sample size n	Proportion of outliers						
	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Refining RANSAC estimate

- RANSAC computes its best estimate from a minimal sample of n points, and divides all data points into inliers and outliers using this estimate.
- We can improve this initial estimate by estimation over all inliers (e.g. with standard least-squares minimization).
- But this may change inliers, so alternate fitting with re-classification as inlier/outlier.



RANSAC: Pros and Cons

- Pros:
  - General method suited for a wide range of model fitting problems
  - Easy to implement and easy to calculate its failure rate
- Cons:
  - Only handles a moderate percentage of outliers without cost blowing up
  - Many real problems have high rate of outliers (but sometimes selective choice of random subsets can help)
- A voting strategy, the Hough transform, can handle high percentage of outliers

References

- Basic reading:
  - Szeliski textbook, Chapter 3.2, 4,1