Lecture 6

Detecting Lines

So far: discrete derivatives in 3 ways

$$\frac{df}{dx} = f[x] - f[x-1]$$ Backward

$$= f[x+1] - f[x]$$ Forward

$$= \frac{1}{2}(f[x+1] - f[x-1])$$ Central but we can drop the 1/2

So far: Designing filters that perform differentiation

- Using Backward differentiation:

0	1	-1

$$g[n,m] = f[n,m] - f[n,m-1]$$

- Using Forward differentiation:

1	-1	0

$$g[n,m] = f[n,m+1] - f[n,m]$$

- Using Central differentiation:

1	0	-1

$$g[n,m] = f[n,m+1] - f[n,m-1]$$

So far: Calculating gradient magnitude and direction

$$\text{Given function } f[n,m]$$

$$\text{Gradient filter } \nabla f[n,m] = \begin{bmatrix} \frac{df}{dn} \\ \frac{df}{dm} \end{bmatrix} = \begin{bmatrix} f_n \\ f_m \end{bmatrix}$$

$$\text{Gradient magnitude } |\nabla f[n,m]| = \sqrt{f_n^2 + f_m^2}$$

$$\text{Gradient direction } \theta = \tan^{-1}(\frac{f_m}{f_n})$$

Today's agenda

- Sobel Edge detector
- Canny edge detector
- Hough Transform
- RANSAC

Optional reading:
Szeliski, Computer Vision: Algorithms and Applications, 2nd Edition
Sections 7.1, 8.1.4

Today's agenda

- Sobel Edge detector
- Canny edge detector
- Hough Transform
- RANSAC

Optional reading:
Szeliski, Computer Vision: Algorithms and Applications, 2nd Edition
Sections 7.1, 8.1.4

Sobel Operator

- uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives
- one for horizontal changes, and one for vertical

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Sobel Operation

- Smoothing + differentiation

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} +1 & 0 & -1 \end{bmatrix}$$

Gaussian smoothing

differentiation

Sobel Operation

- Magnitude:

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

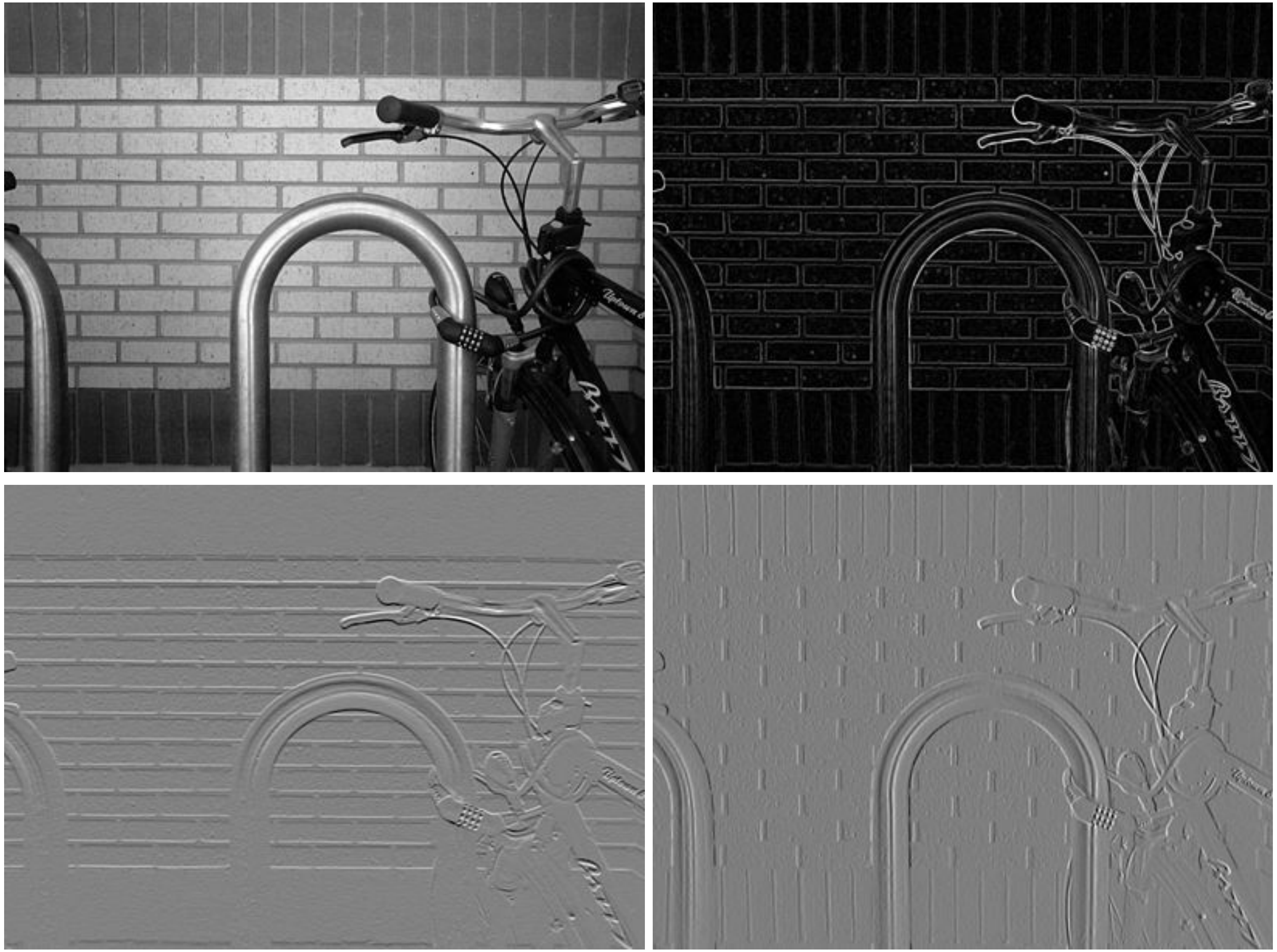
- Angle or direction of the gradient:

$$\mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

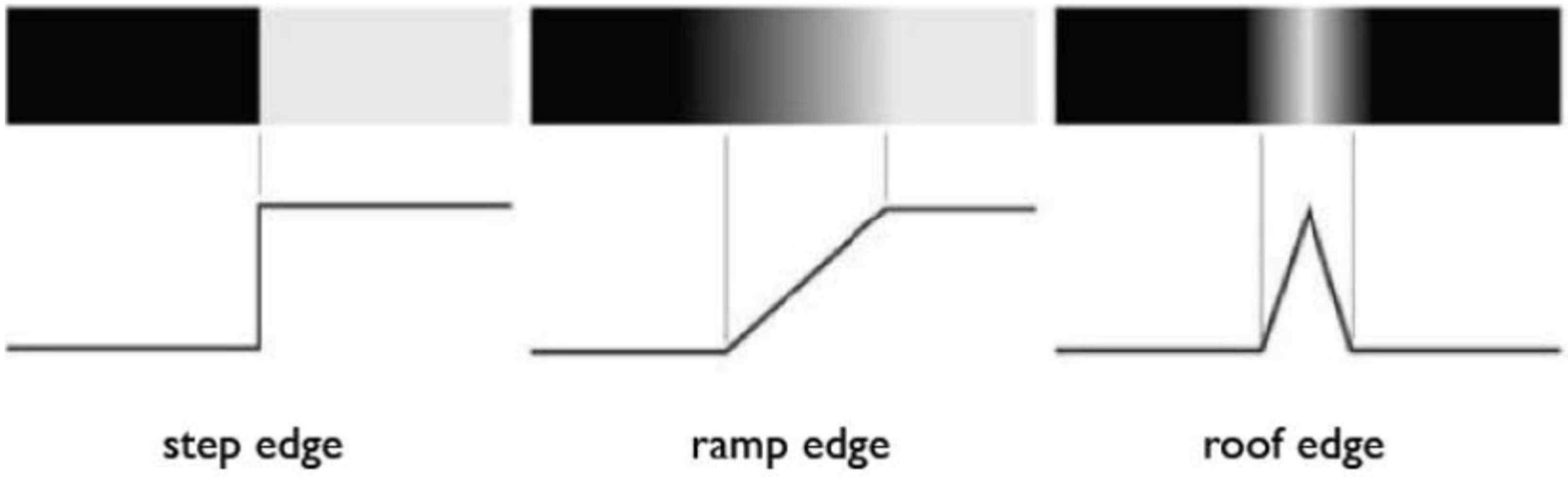
Sobel Filter example

Step 1: Calculate the gradient magnitude at every pixel location.

Step 2: Threshold the values to generate a binary image



Sobel Filter Problems



- Poor Localization (Trigger response in multiple adjacent pixels)
- Thresholding value favors certain directions over others
  - Can miss oblique edges more than horizontal or vertical edges
  - False negatives

What we will learn today

- Sobel Edge detector
- Canny edge detector
- Hough Transform
- RANSAC

So far: A simple edge detector

- This theorem gives us a very useful property:

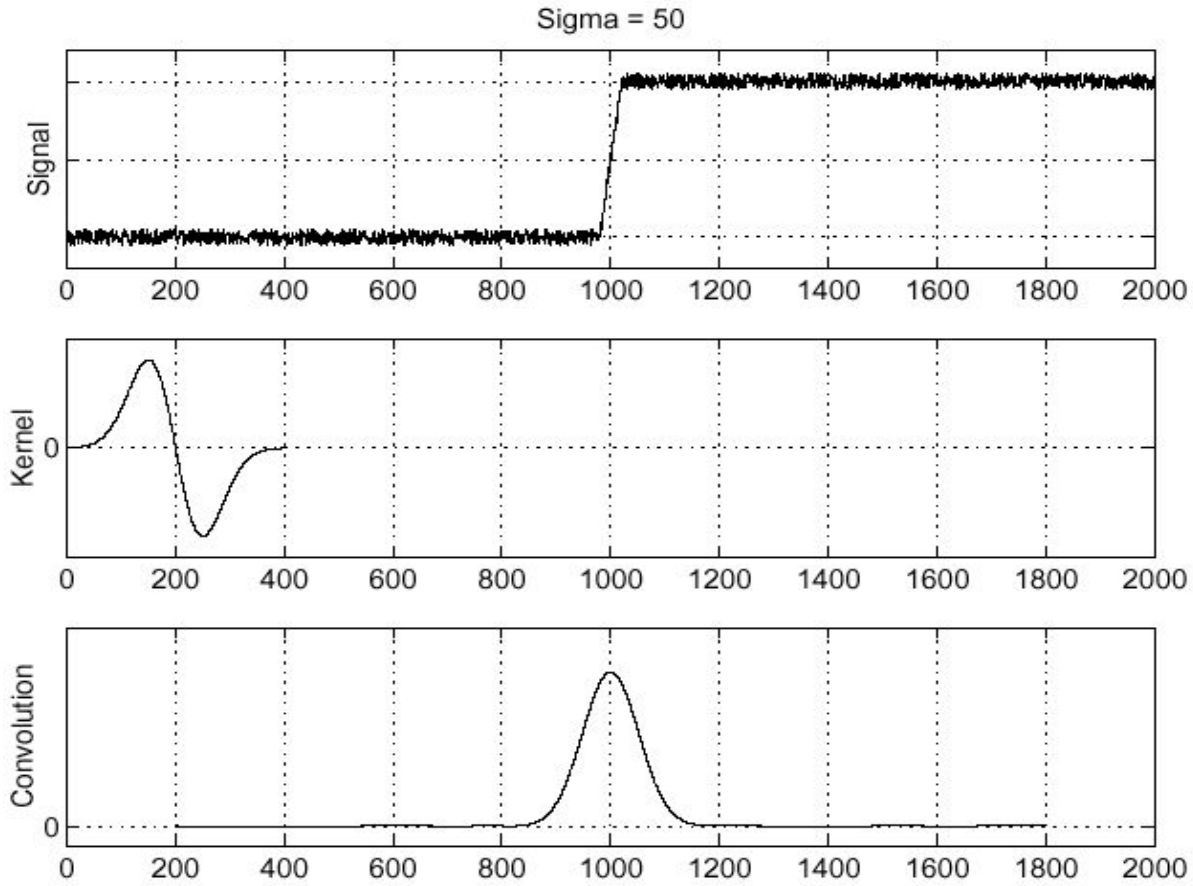
$$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$$

- This saves us one operation:

f

$\frac{d}{dx} g$

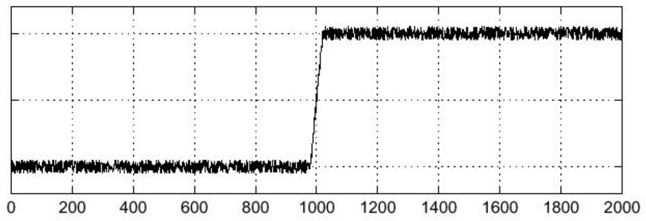
$f * \frac{d}{dx} g$



Source: S. Seitz

Canny edge detector

- This is probably the most widely used edge detector in computer vision
- **Theoretical model**: optimal edge detection when pixels are corrupted by additive Gaussian noise
- Theory shows that first derivative of the Gaussian closely approximates the operator that optimizes the product of ***signal-to-noise ratio***



J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

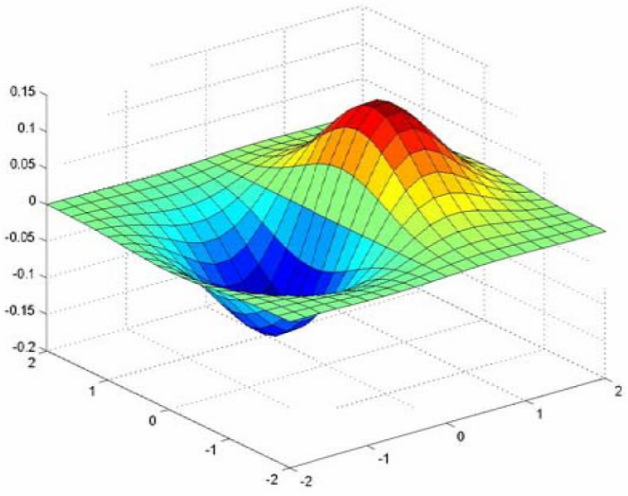
Canny edge detector

1. Suppress Noise
2. Compute gradient magnitude and direction
3. Apply Non-Maximum Suppression
    - Assures minimal response
4. Use hysteresis and connectivity analysis to detect edges

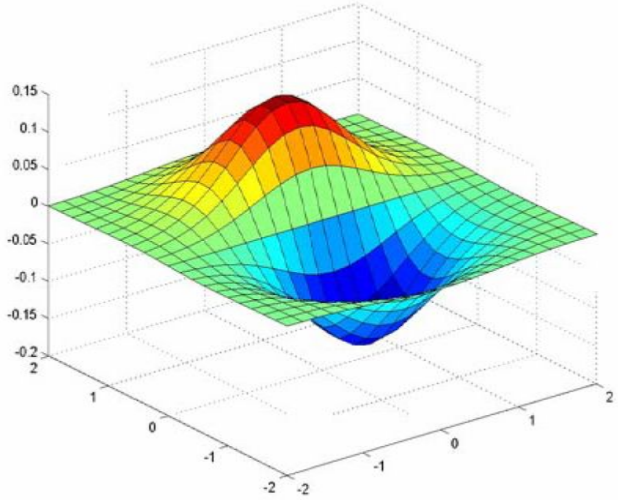
Example

- original image

Derivative of Gaussian filter

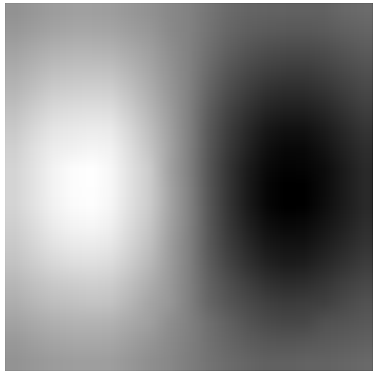


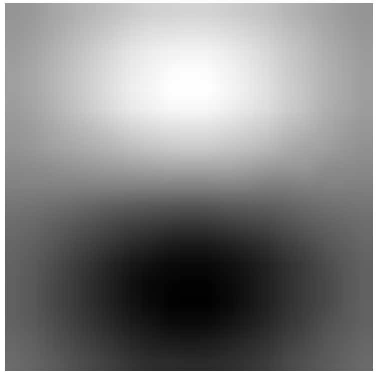
x-direction



y-direction

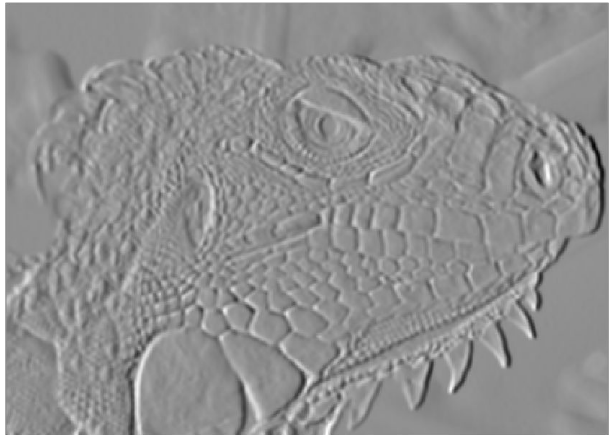
$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$



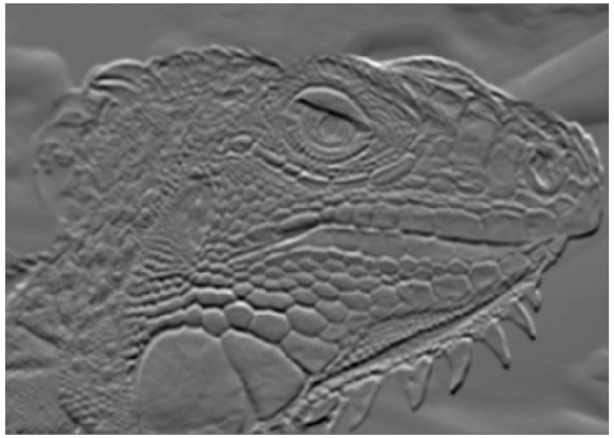


$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Compute gradients (DoG)



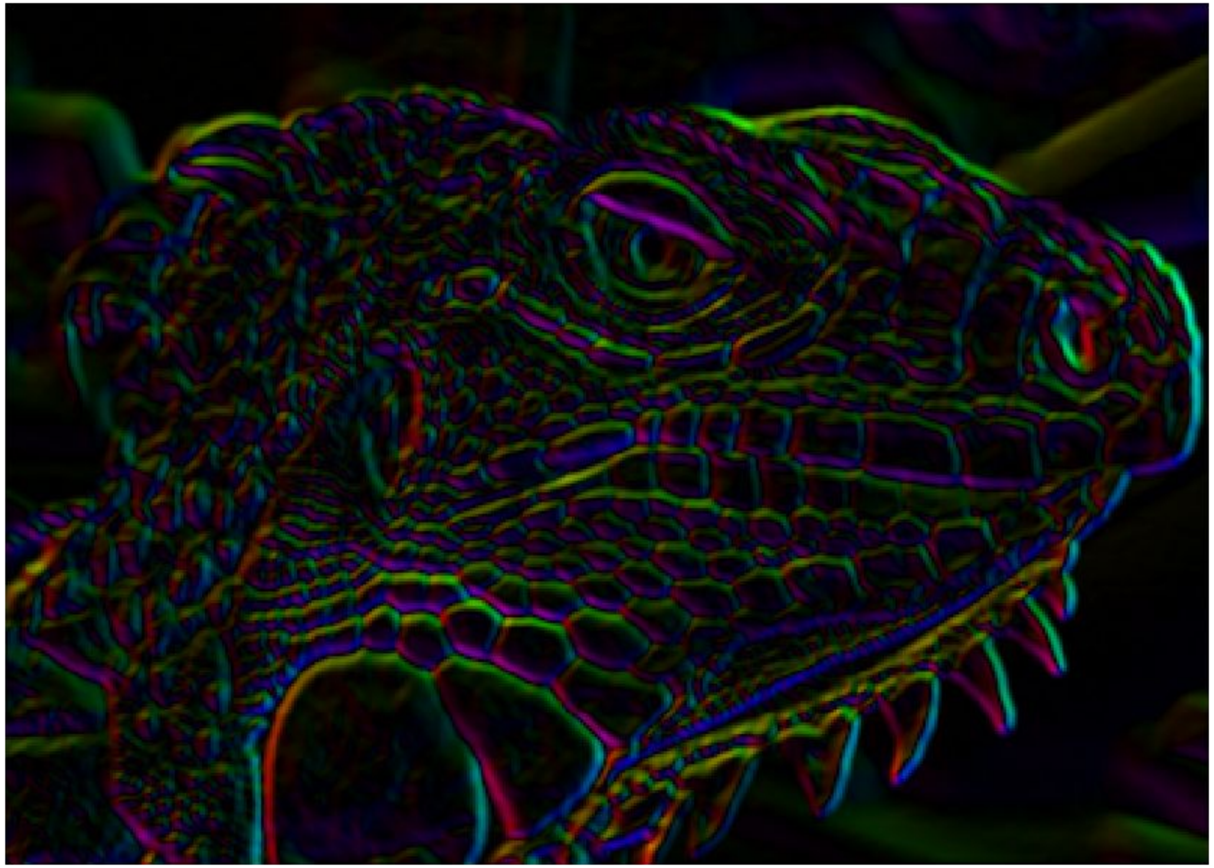
X-Derivative of Gaussian



Y-Derivative of Gaussian

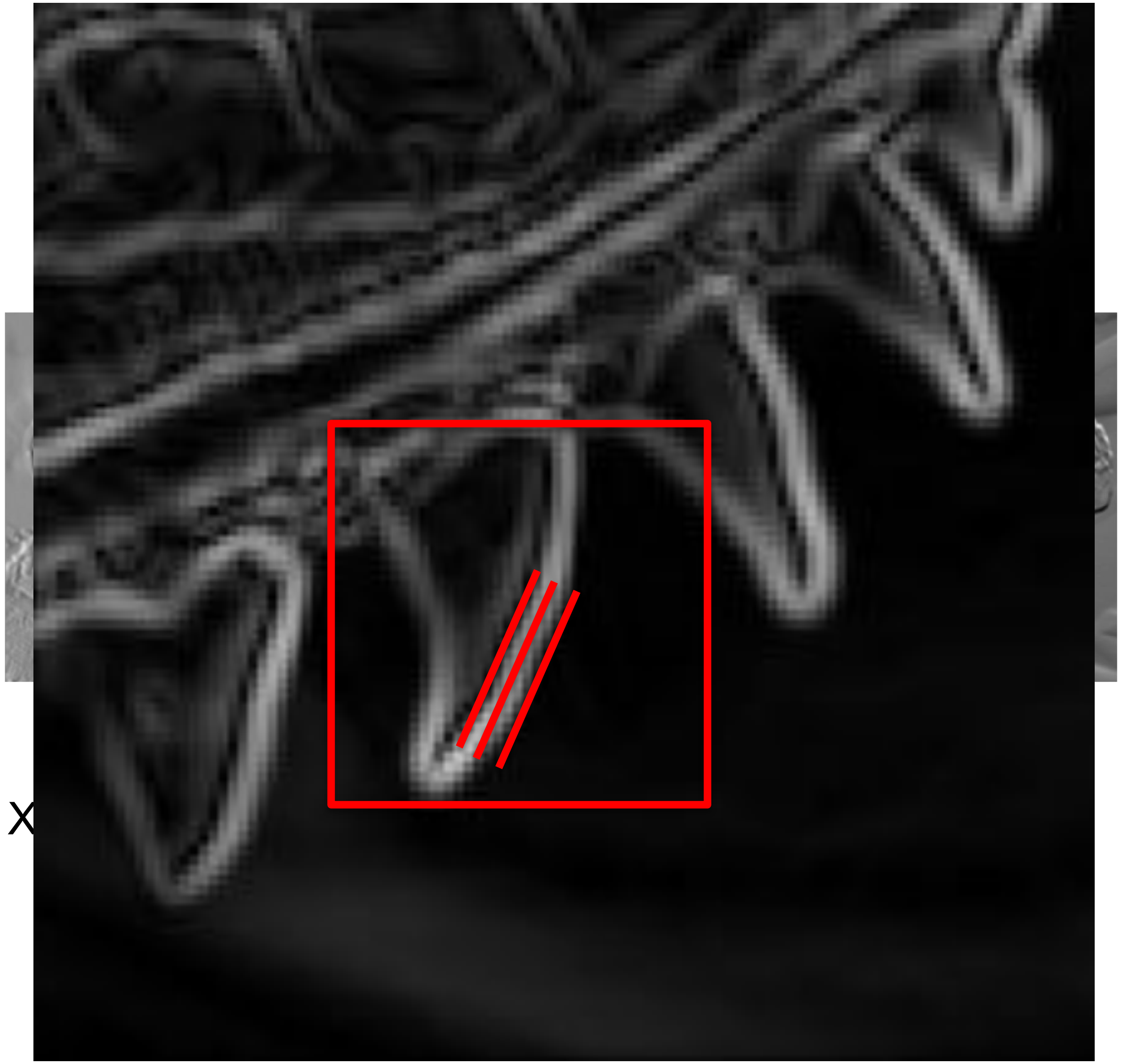
Gradient Magnitude

Get orientation at each pixel



$$\Theta = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

Compute gradients (DoG)



X

Gradient Magnitude

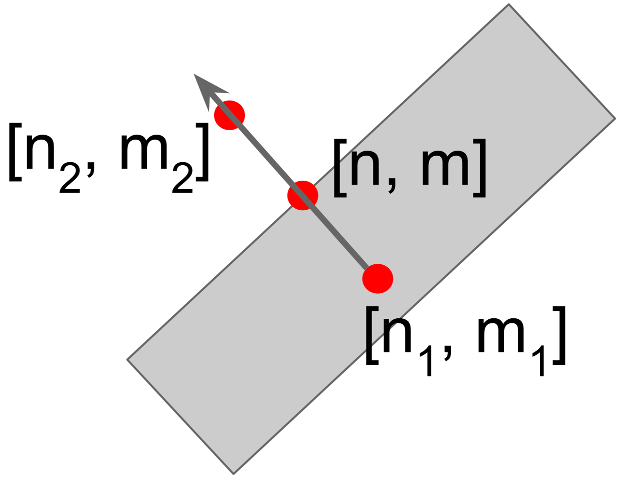
Canny edge detector

1. Suppress Noise
2. Compute gradient magnitude and direction
3. Apply Non-Maximum Suppression
   - Assures minimal response

Non-maximum suppression

- **Edge occurs where gradient reaches a maxima**
- **Suppress non-maxima** pixels even if it passes threshold
- Assume only points along the angle directions
  - **Suppress all pixels** in the direction which are not maxima

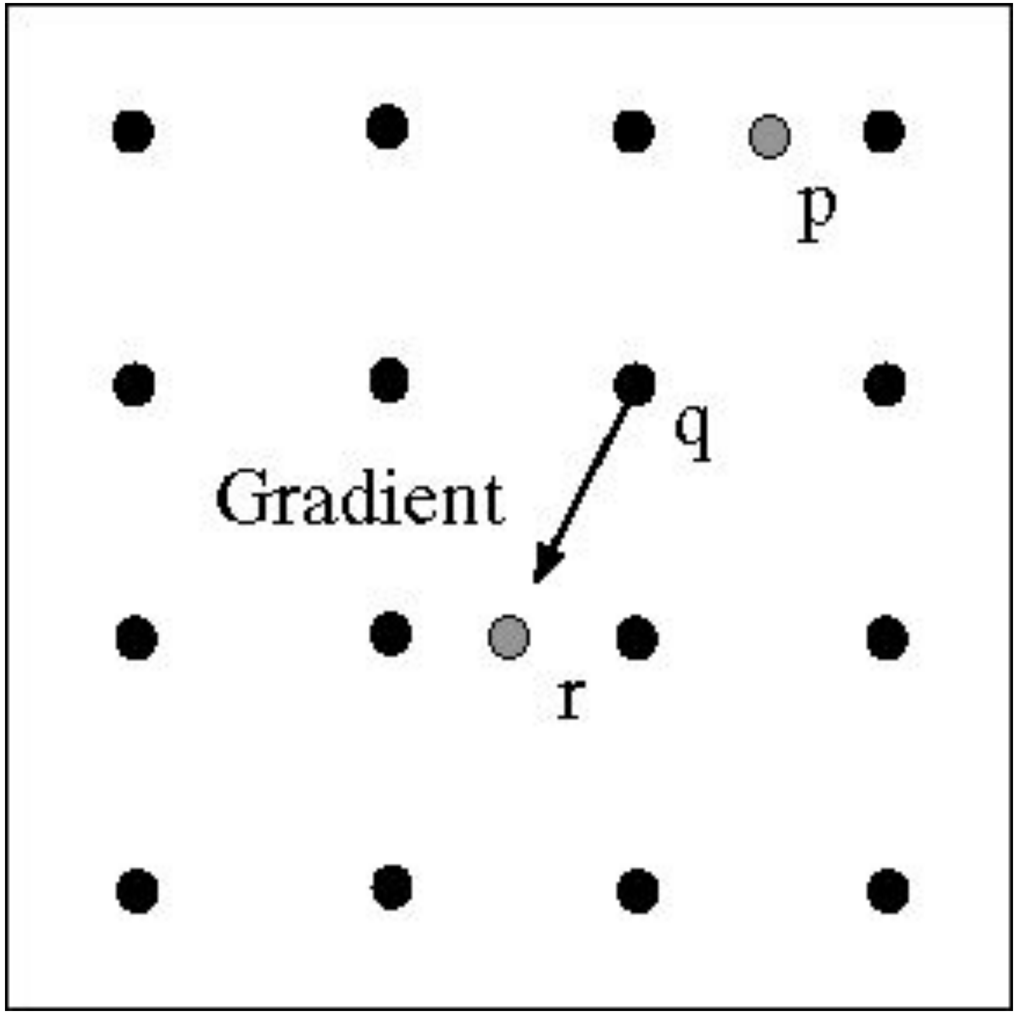
Remove spurious gradients



$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

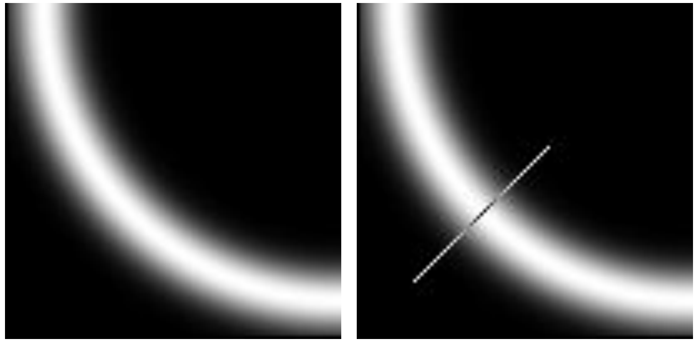
$$\text{If } G[n,m] = \begin{cases} G[n,m] & \text{if } G[n,m] > G[n_1,m_1] \text{ and } G[n,m] > G[n_2,m_2] \\ 0 & \text{otherwise} \end{cases}$$

What if $p = [n_1, m_1]$ or $r = [n_2, m_2]$, is not a pixel location

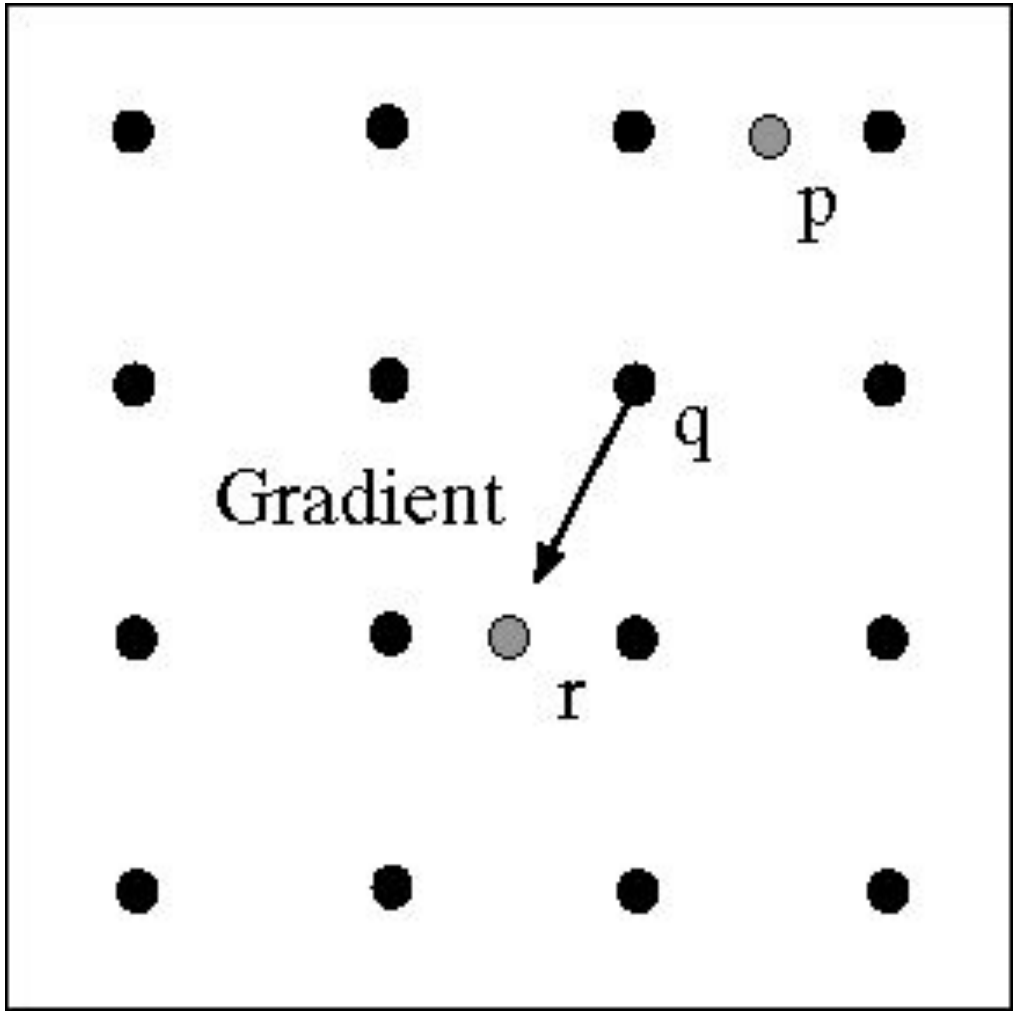


q is a maximum if the value is larger than those at both p and at r.

How should we calculate magnitude at G?



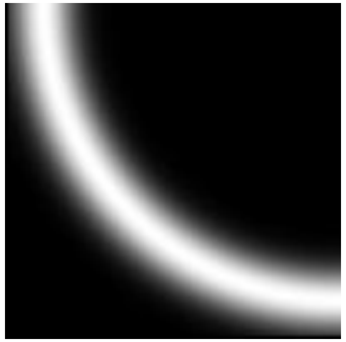
What if $p = [n_1, m_1]$ or $r = [n_2, m_2]$, is not a pixel location



q is a maximum if the value is larger than those at both p and at r.

How should we calculate magnitude at G?

p and r are weighted averaged values of top k=8 closest pixel locations



Non-max Suppression

Before

After

Canny edge detector

1. Suppress Noise
2. Compute gradient magnitude and direction
3. Apply Non-Maximum Suppression
   - Assures minimal response
4. Use hysteresis and connectivity analysis to detect edges

Problem: if your threshold is too high (left) or too low (right), you have too many or too few edges

Hysteresis thresholding

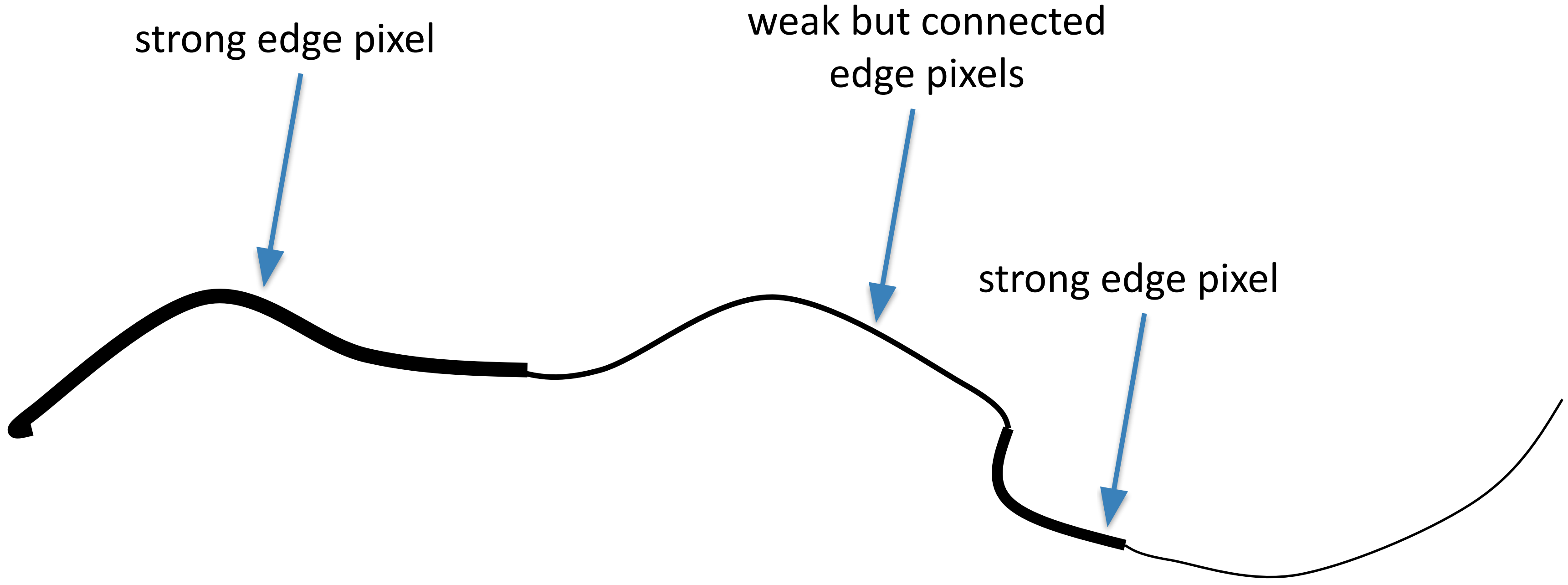
- Avoid breaking edges near the threshold value
- Define two thresholds: **Low** and **High**
  - If less than Low => **not an edge**
  - If greater than High => **strong edge**
  - If between Low and High => **weak edge**

Hysteresis thresholding

If the gradient at a pixel is

- above High, declare it as an ‘strong edge pixel’
- below Low, declare it as a “non-edge-pixel”
- between Low and High
  - Consider its neighbors iteratively then declare it an “edge pixel” if it is connected to an ‘strong edge pixel’ directly or via pixels between Low and High

Hysteresis thresholding



Source: S. Seitz

Final Canny Edges

Canny edge detector

1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
   - Thin multi-pixel wide “ridges” down to single pixel width
4. Thresholding and linking (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

Effect of σ (Gaussian kernel spread/size)

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

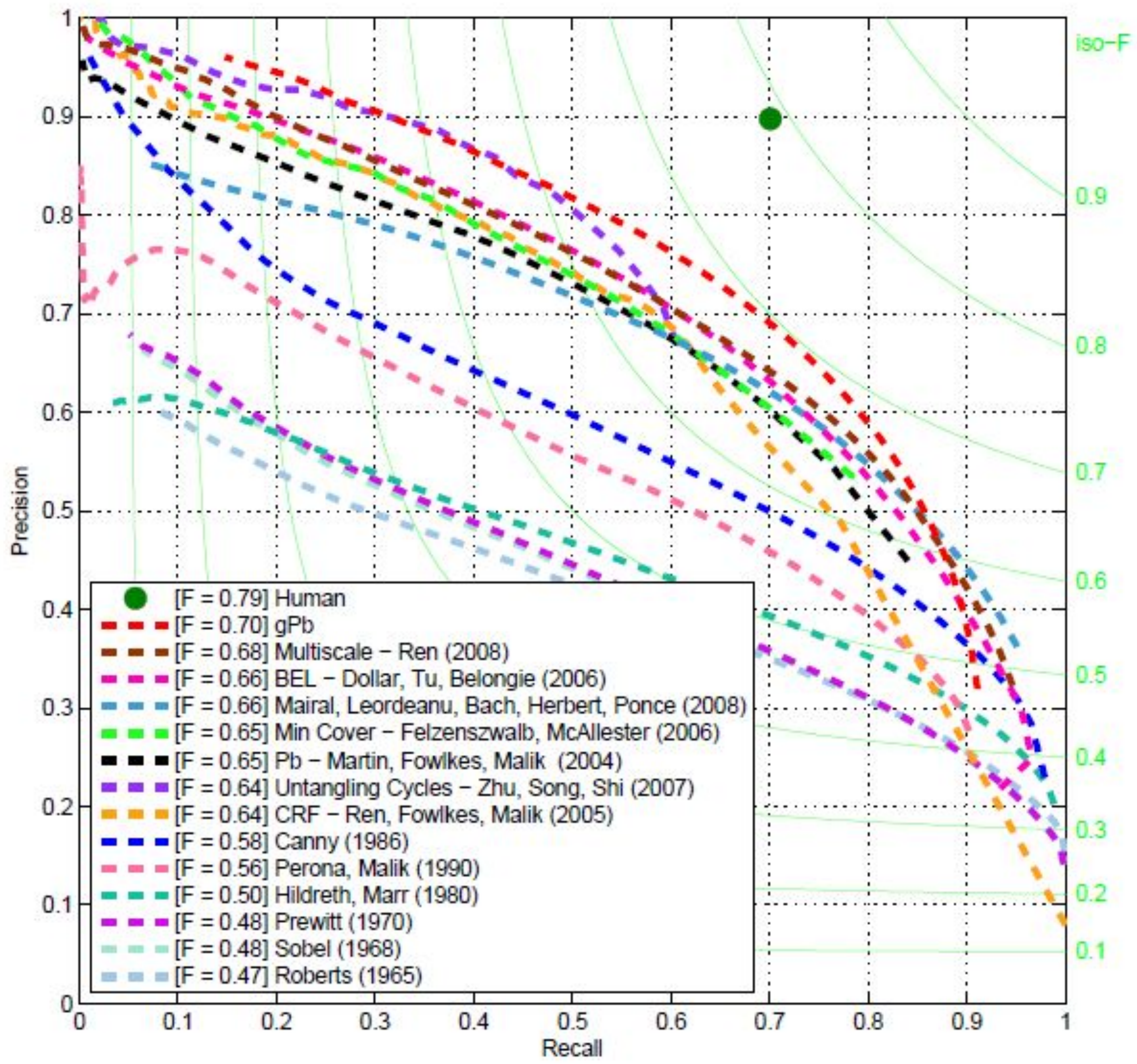
The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

Gradients (e.g. Canny)

Human

45 years of edge detection



Source: Arbelaez, Maire, Fowlkes, and Malik. TPAMI 2011 (pdf)

What we will learn today

- Sobel Edge detector
- Canny edge detector
- Hough Transform
- RANSAC

Hough transform

How Transform edge detections into lines

Original image

Edge image

Hough transform

- It was introduced in 1962 (Hough 1962) and first used to find lines in images a decade later (Duda 1972).

- **Caveat**: 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

Input to Hough transform algorithm

- We have performed some edge detection (Sobel filter, Canny Edge detector, etc.), including a thresholding of the edge magnitude image.
- Thus, we have some pixels that may partially describe the boundary of some objects.

Edge image

Detecting lines using Hough transform

- We wish to find sets of pixels that make up straight lines.
- Instead of using [n, m], this might be easier to do with (x, y)

How do we transform [n, m] to (x, y)?

- Simple: We assume
  - $n = y$,
  - $m = x$.

- So, $f[n, m] = f[y, x]$

Detecting lines using Hough transform

- Consider a line that passes through two points in the image
  - (x_1, y_1) and (x_2, y_2)
- Straight lines that pass that point have the form:

$$y = a*x + b$$

- How do we calculate the parameters (a, b)?

$$a = (y_2 - y_1) / (x_2 - x_1)$$
$$b = y_1 - a \times x_1$$

Detecting lines using Hough transform

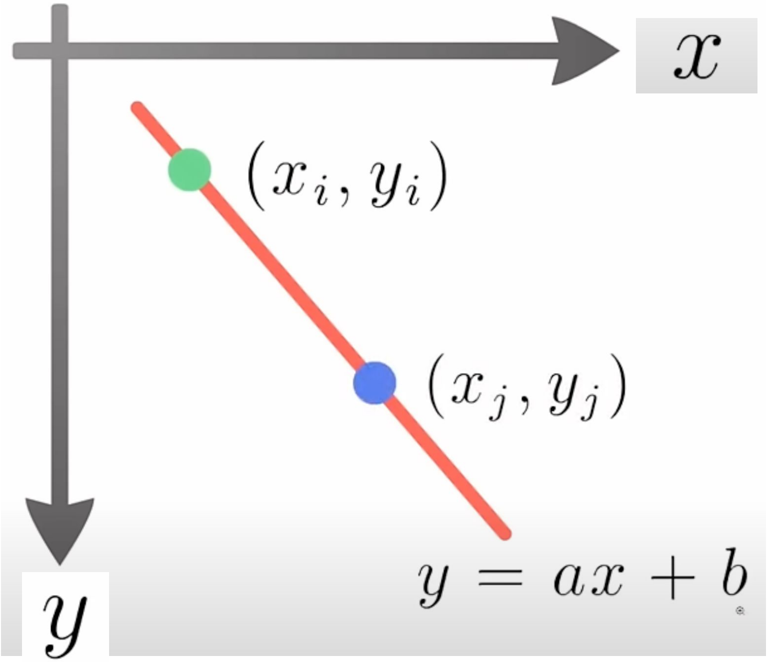
- Consider a line that passes through two points in the image
  - (x_1, y_1) and (x_2, y_2)

- Straight lines that pass that point have the form:

  $$y = a*x + b$$

- How do we calculate the parameters (a, b)?

  $$a = (y_2 - y_1) / (x_2 - x_1)$$
  $$b = y_1 - a \times x_1$$



Detecting lines using Hough transform

- Problem: We don't know which pairs of edge points belong to the same line.

- That's where Hough transform comes in!

The Hough transform

- Consider a line that passes through a single point in the image
  - (x_i, y_i)

- All straight lines that pass that point have the form:

$$y_i = a * x_i + b$$

The Hough transform

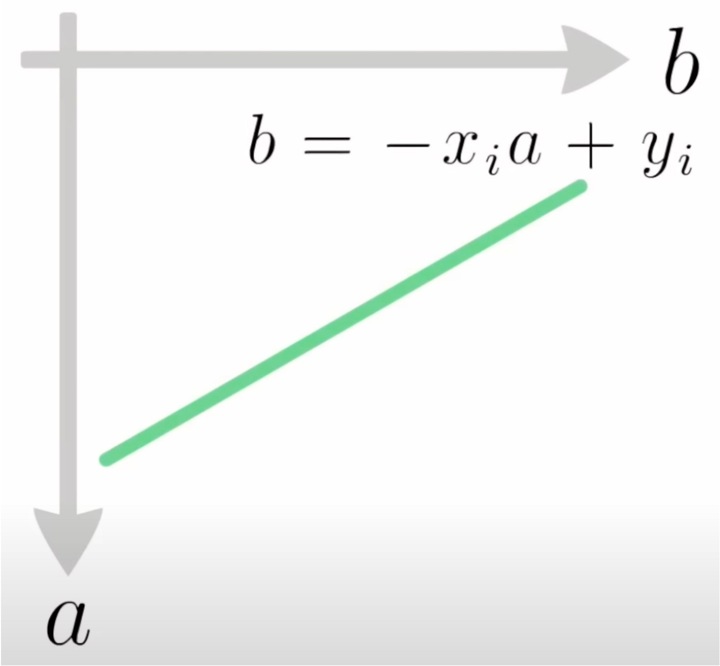
$$\boxed{y_i = a * x_i + b}$$

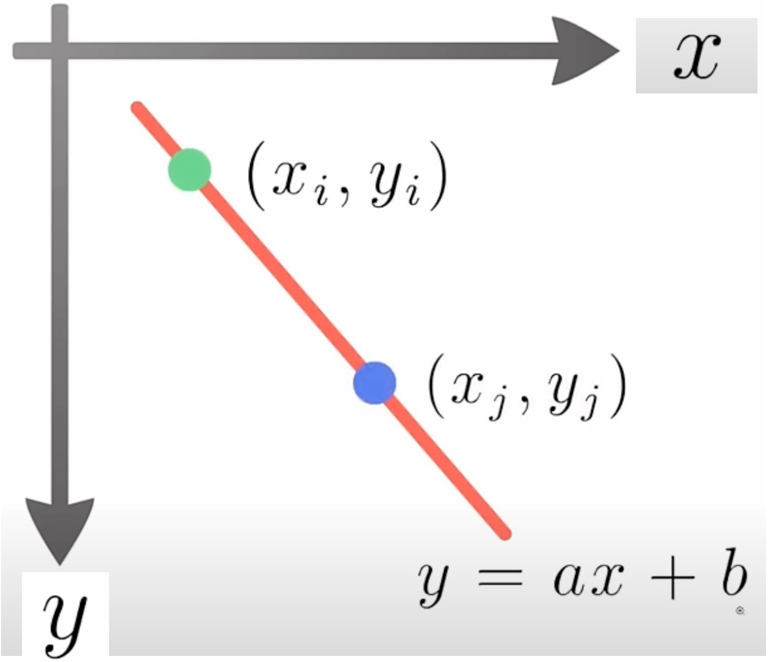
- This equation can be rewritten as follows:
    - $b = -a * x_i + y_i$

The Hough transform

$y_i = a*x_i + b$

- This equation can be rewritten as follows:
  - $b = -a*x_i + y_i$
  - We can now consider x and y as parameters
  - a and b as coordinates.

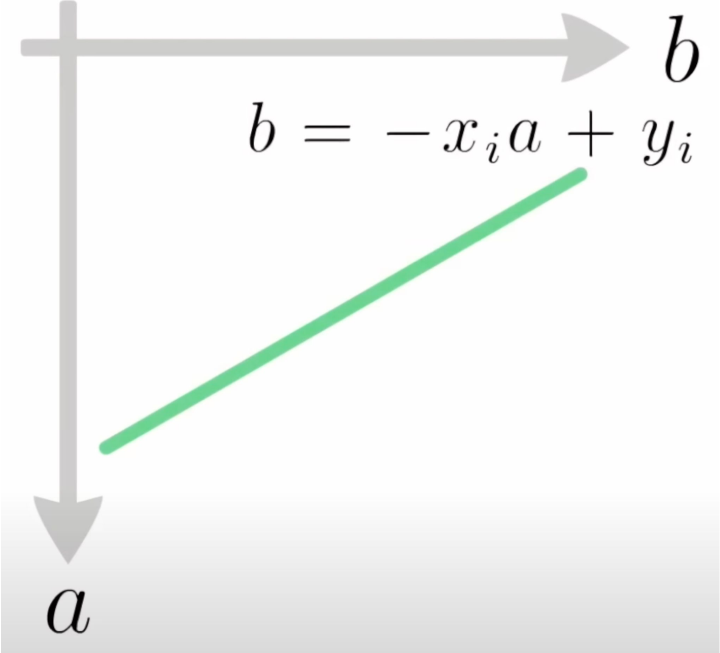




The Hough transform

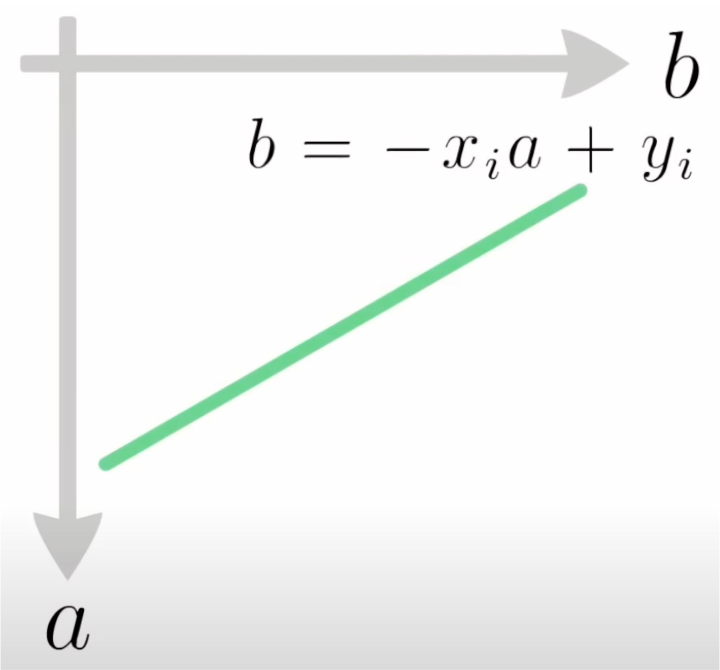
$y_i = a^*x_i + b$

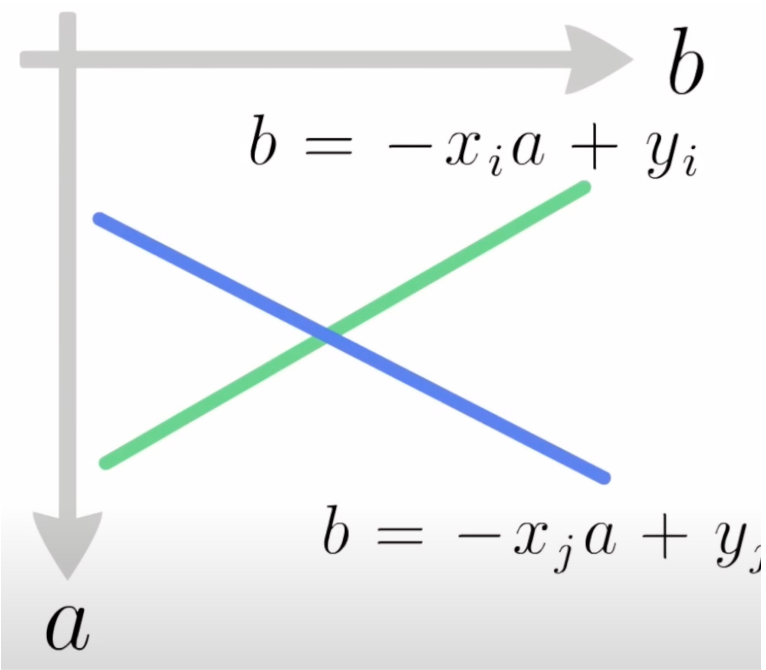
- $b = -a^*x_i + y_i$
- If our coordinates were (a,b) instead of (x, y):
  - We could say the above equation is a line in (a,b)-space
  - parameterized by x and y.
  - So: one point (x_i,y_i) gives a line in (a,b) space.



The Hough transform

- So: one point (x_i,y_i) gives a line in (a,b) space.
- Another point (x_j,y_j) will give rise to another line in (a,b)-space.



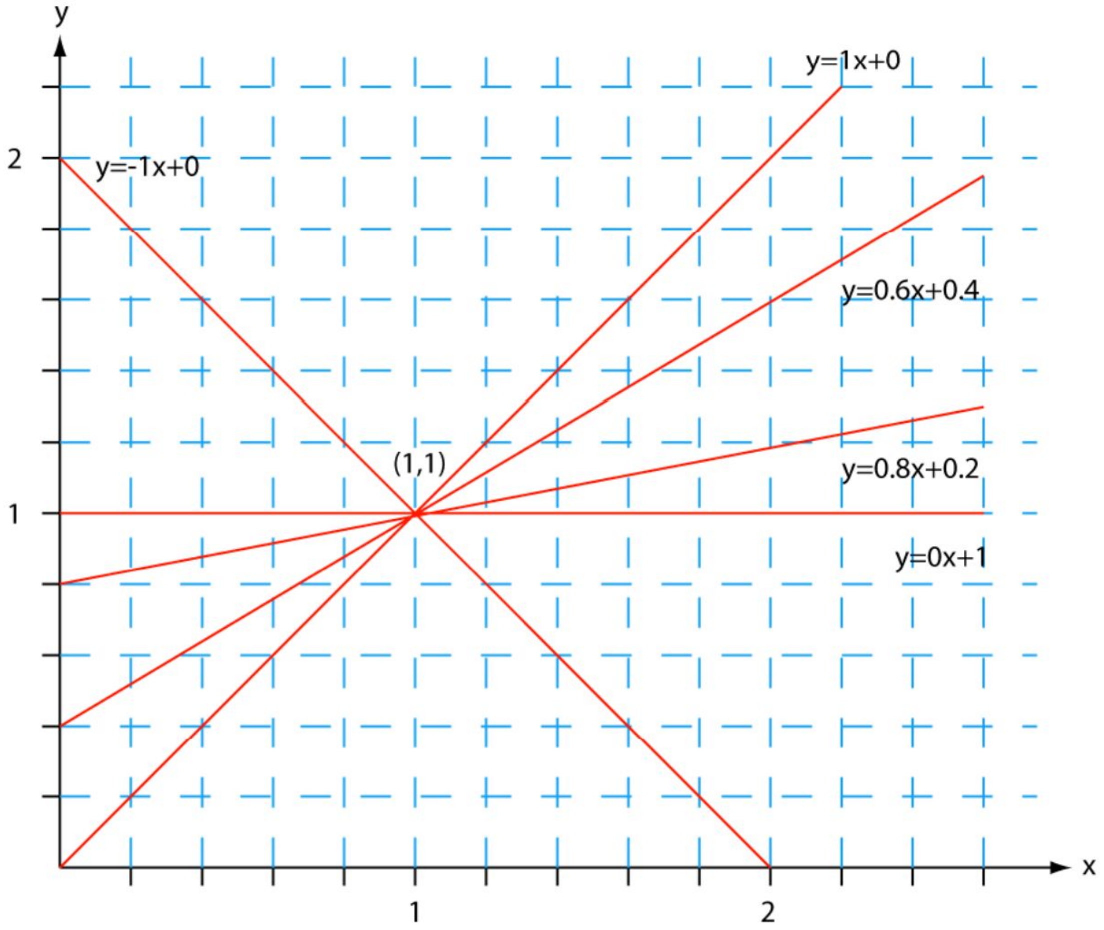


The Hough transform

- Doing this for 6 edge points will result in an graph like the one on the right.

- In (a,b) space these lines will intersect in a point **(a’, b’)**
  - On the right, a’ = 1, b’ = 1

- All points on the line defined by (x_i, y_i) and (x_j, y_j) in (x, y)-space will parameterize lines that intersect in (a’, b’) in (a,b) space.

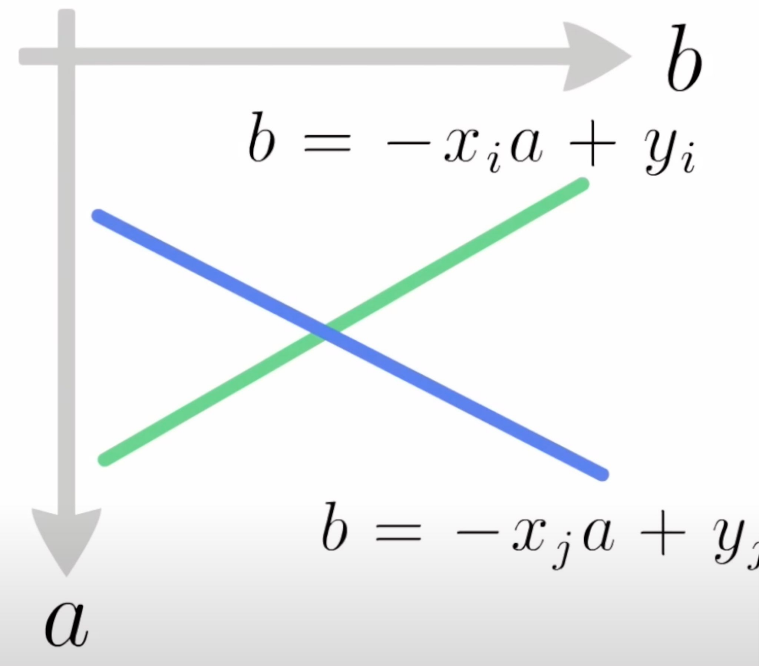


The need to quantize and “vote”

Not all intersections will be valid lines.

Consider two edge points that are not part of a real edge:

- They might still intersect in (a, b) space.



Problem: How do we identify intersections that are belong to the same edge versus random points?

Intuition behind voting

The more lines intersect at the same (a’, b’) point, the more likely y=a’x + b’ is a real edge in the image.

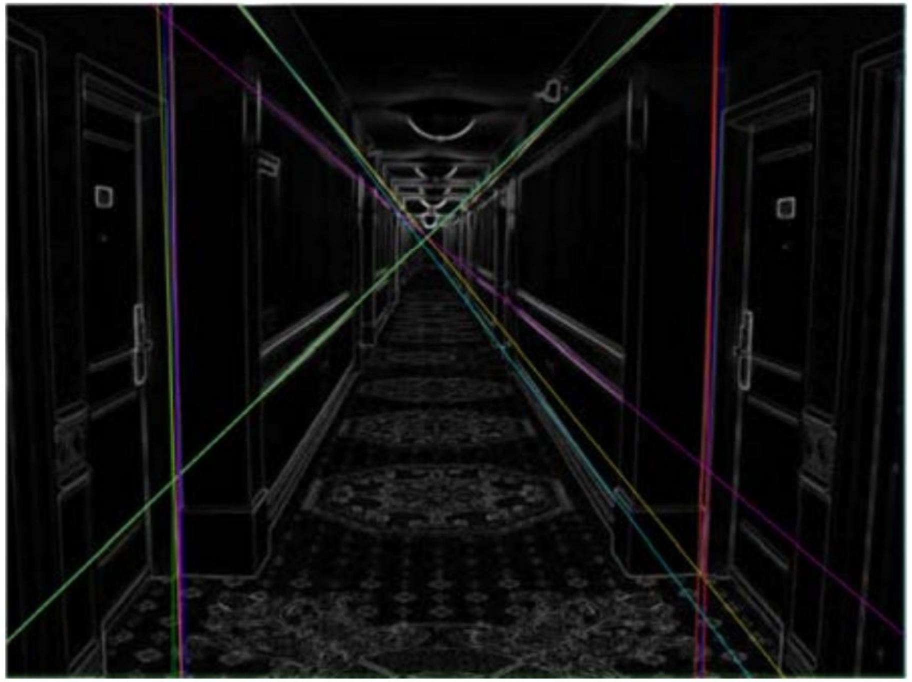
So, we need to count how many lines intersect at a point and keep the ones with high count

Counting in quantized (a, b)-space

1. Quantize the parameter space (a b) by dividing it into cells
   a. $[[a_{min}, a_{max}],[b_{min},b_{max}]]$
2. For each pair of points (x_i, y_i) and (x_j, y_j), find the intersection (a',b') in (a,b)-space.
3. Increase the value of a cell in the range $[[a_{min}, a_{max}],[b_{min},b_{max}]]$ that (a', b') belongs to.
4. Cells receiving more than a certain number of counts (also called 'votes') are assumed to correspond to lines in (x,y) space.

Output of Hough transform

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



Other Hough transformations

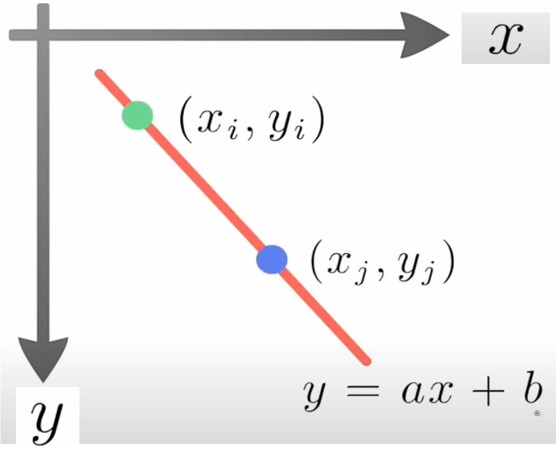
- We can represent lines as polar coordinates instead of $y = a*x + b$

- Polar coordinate representation:
  - $x*\cos\theta + y*\sin\theta = \rho$

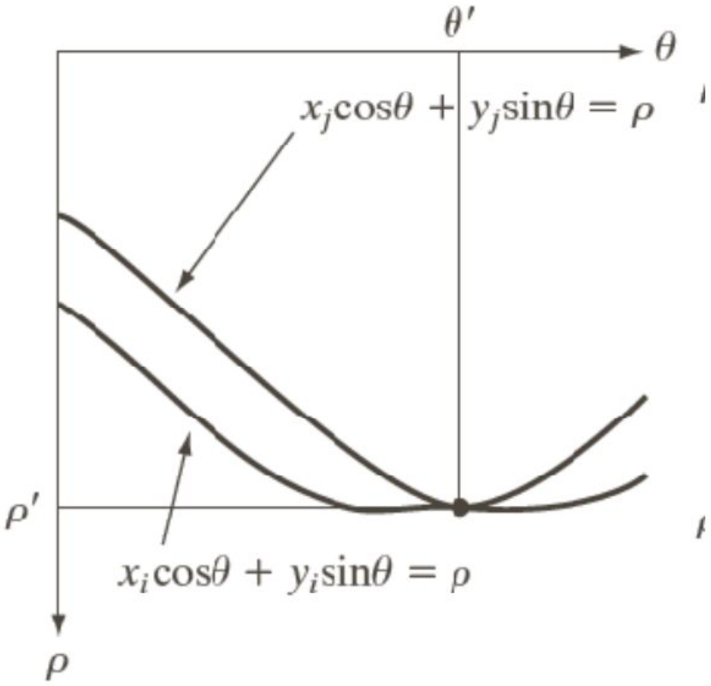
- We can transform points in (x, y) space to curves in $(\rho\ \theta)$-space
  - $(x\ y)$ and $(\rho\ \theta)$?

Other Hough transformations

- Note that lines in (x, y)-space are not lines in (ρ, θ)-space



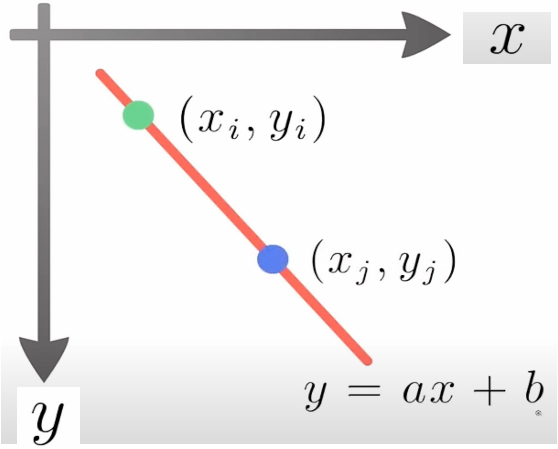
- Curves in (ρ, θ)-space intersect similarly like in (a, b)-space.

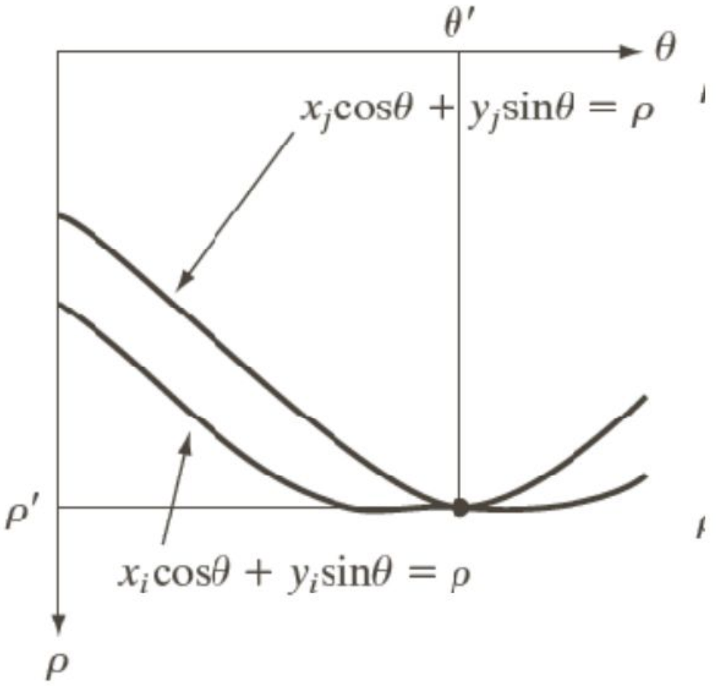


Other Hough transformations

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

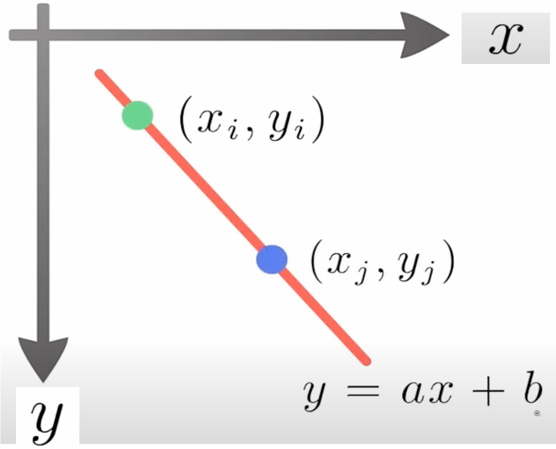
- Q. For a vertical line in (x, y)-space, what are the θ and ρ values?

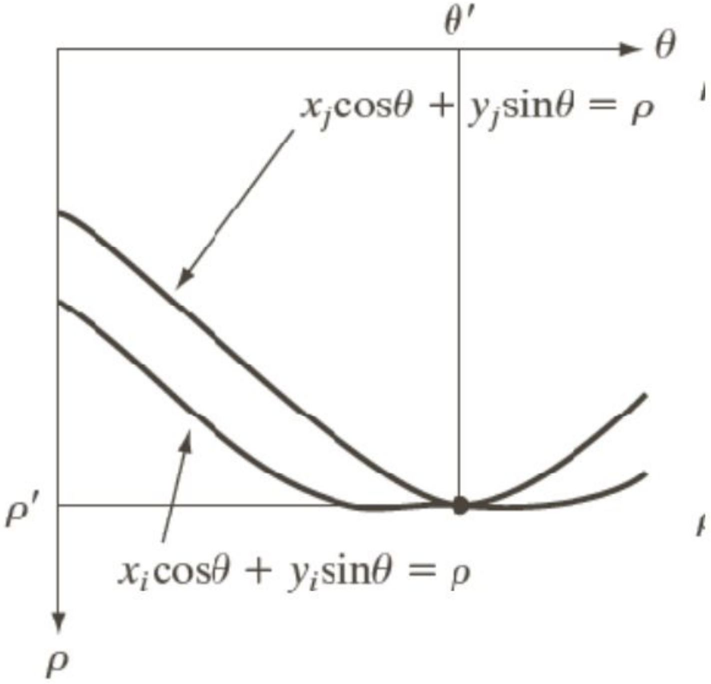




Other Hough transformations

- $x*\cos\theta + y*\sin\theta = \rho$
- Q. For a vertical line in (x, y)-space, what are the θ and ρ values?
  - θ=0, ρ=x
- Q. For a horizontal line in (x, y)-space, what are the θ and ρ values?





Hough transform remarks

- **Advantages**:
  - Conceptually simple.
  - Easy implementation
  - Handles missing and occluded data very gracefully.
  - Can be adapted to many types of forms, not just lines

Hough transform 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 shapes with many parameters.
  - Looks for only one single shape 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.
  - Runs in $O(N^2)$ since all pairs of points should be considered

What we will learn today

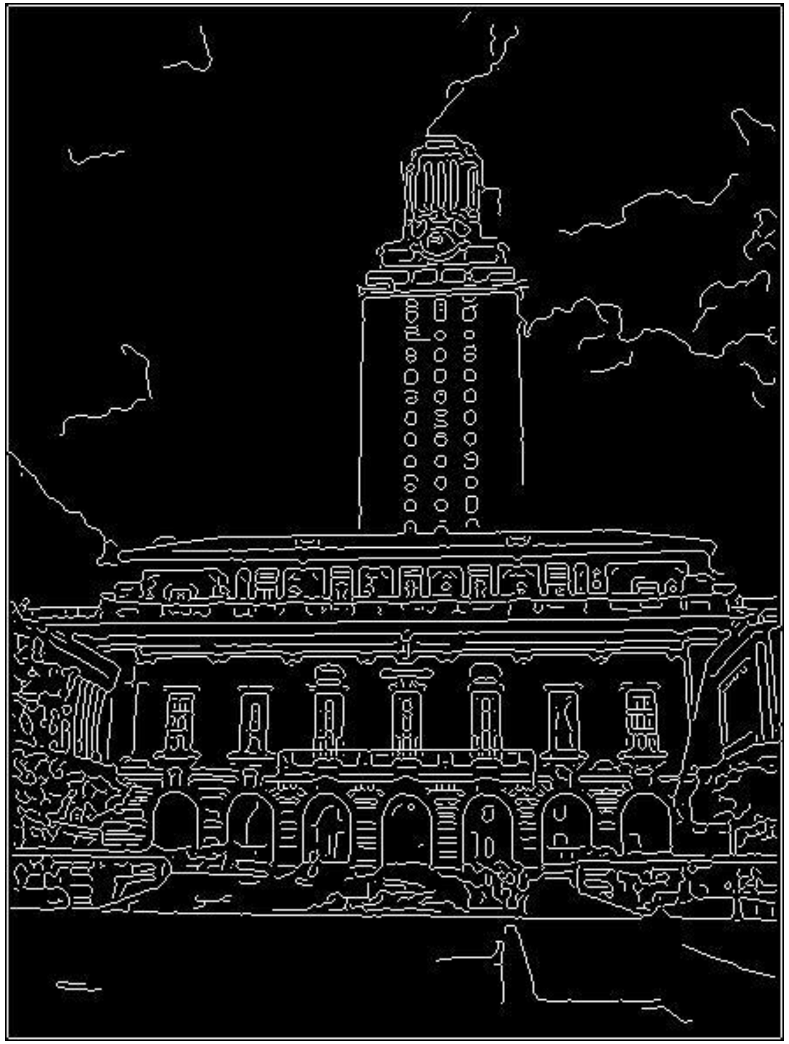
- Sobel Edge detector
- Canny edge detector
- Hough Transform
- RANSAC

Why is Hough transform inefficient?

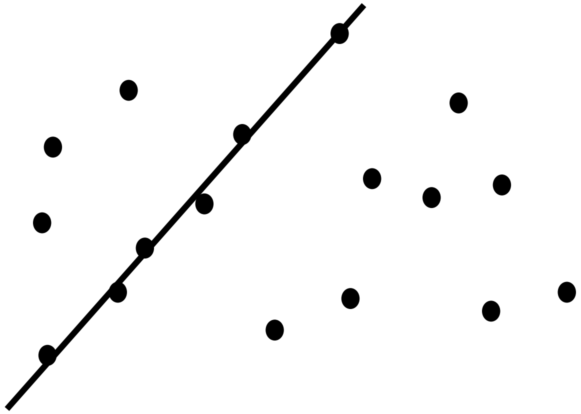
- It's not feasible to check all pairs of points to calculate possible lines. For example, Hough Transform algorithm runs in $O(N^2)$.
- Voting is a general technique where we let the each point vote for all models that are compatible with it.
  - Iterate through features, cast votes for parameters.
  - Filter parameters that receive a lot of votes.
- **Problem:** Noisy points will cast votes too, *but* typically their votes should be inconsistent with the majority of "good" edge points.

Slide credit: Kristen Grauman

Difficulty of voting for lines



- Noisy edge points, cast inconsistent votes:
  - Can we identify them without iterating over all pairs?
- Only some parts of each line detected, and some parts are missing:
  - How do we find a line that bridges missing evidence?
- Noise in measured edge points, orientations:
  - How to detect true underlying parameters?



Slide credit: Kristen Grauman

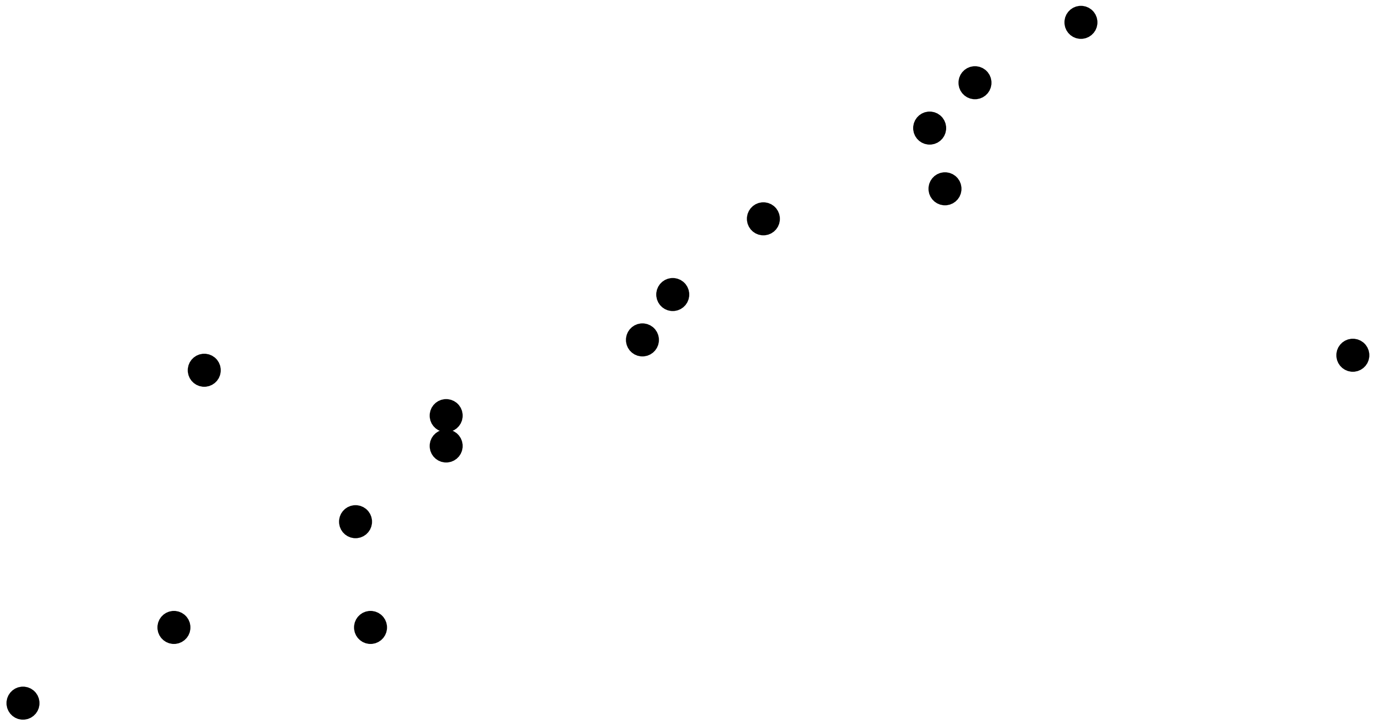
RANSAC [Fischler & Bolles 1981]

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

Slide credit: Kristen Grauman

RANSAC Line Fitting Example

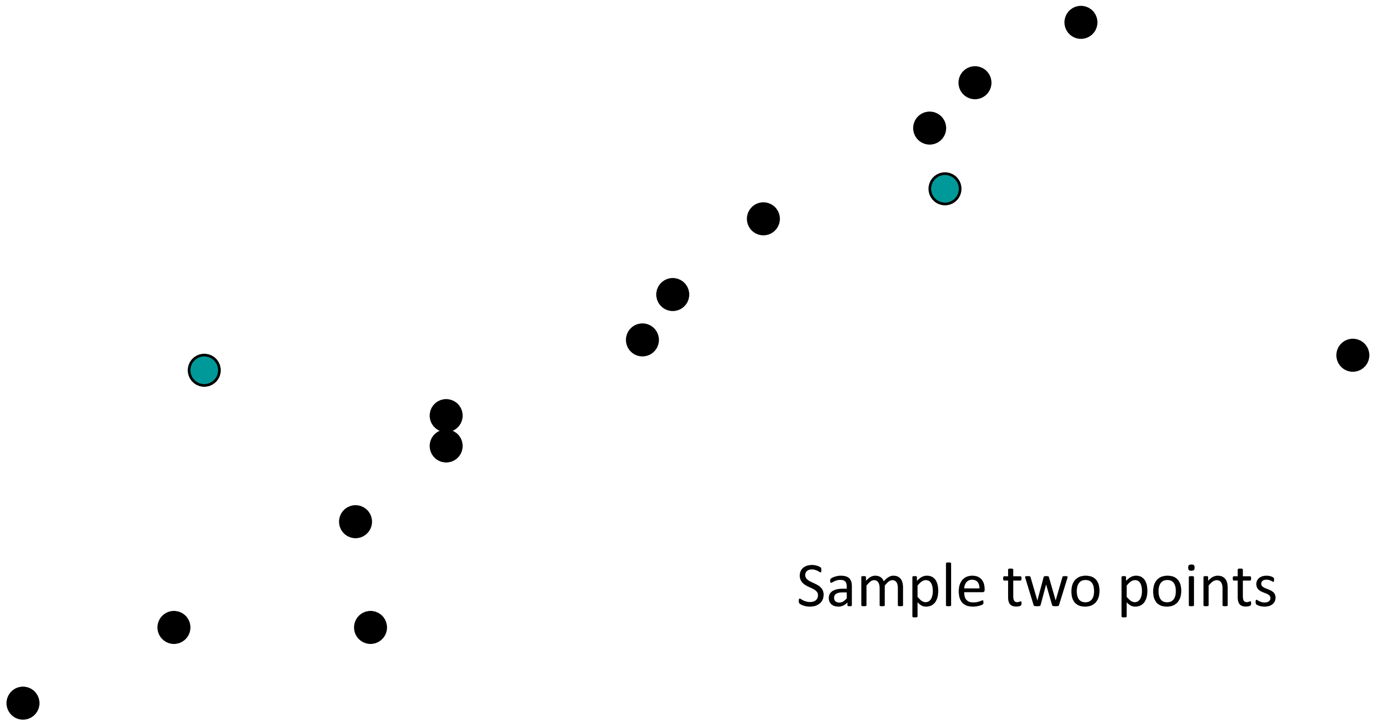
- Task: Estimate the best line
  - *Let's randomly select a subset of points and calculate a line*



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

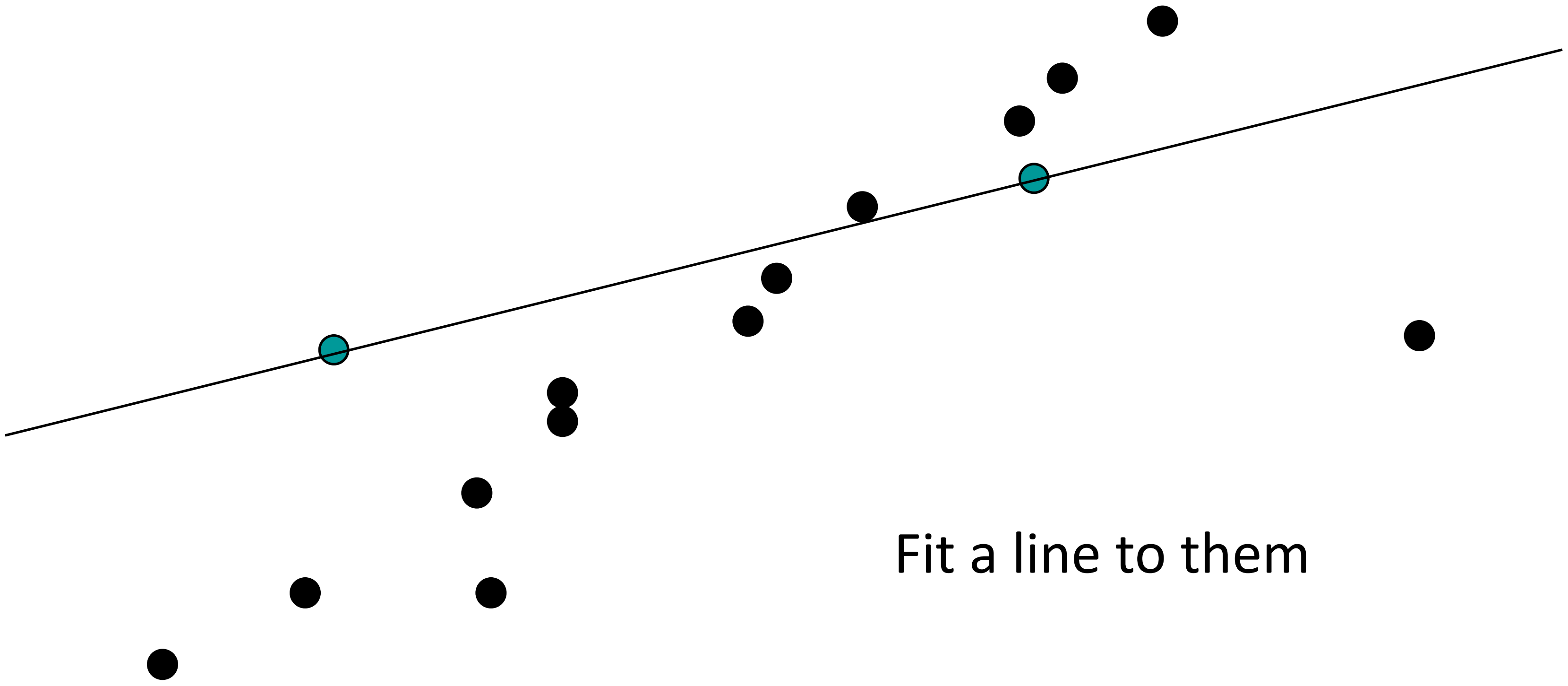
- Task: Estimate the best line
  - Let's select only 2 points as an example



Slide credit: Jinxiang Chai

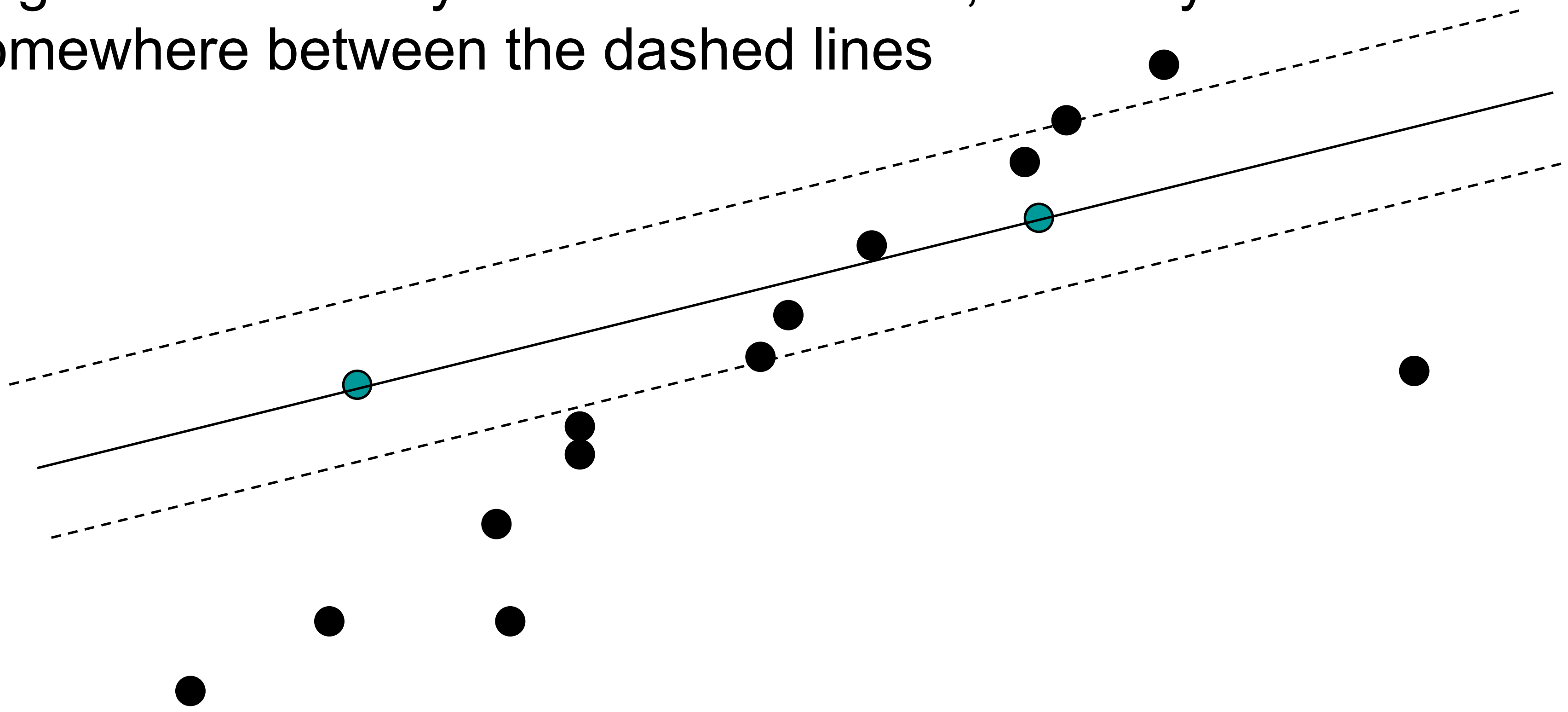
RANSAC Line Fitting Example

- Task: Estimate the best line
  - Calculate the line parameters



RANSAC Line Fitting Example

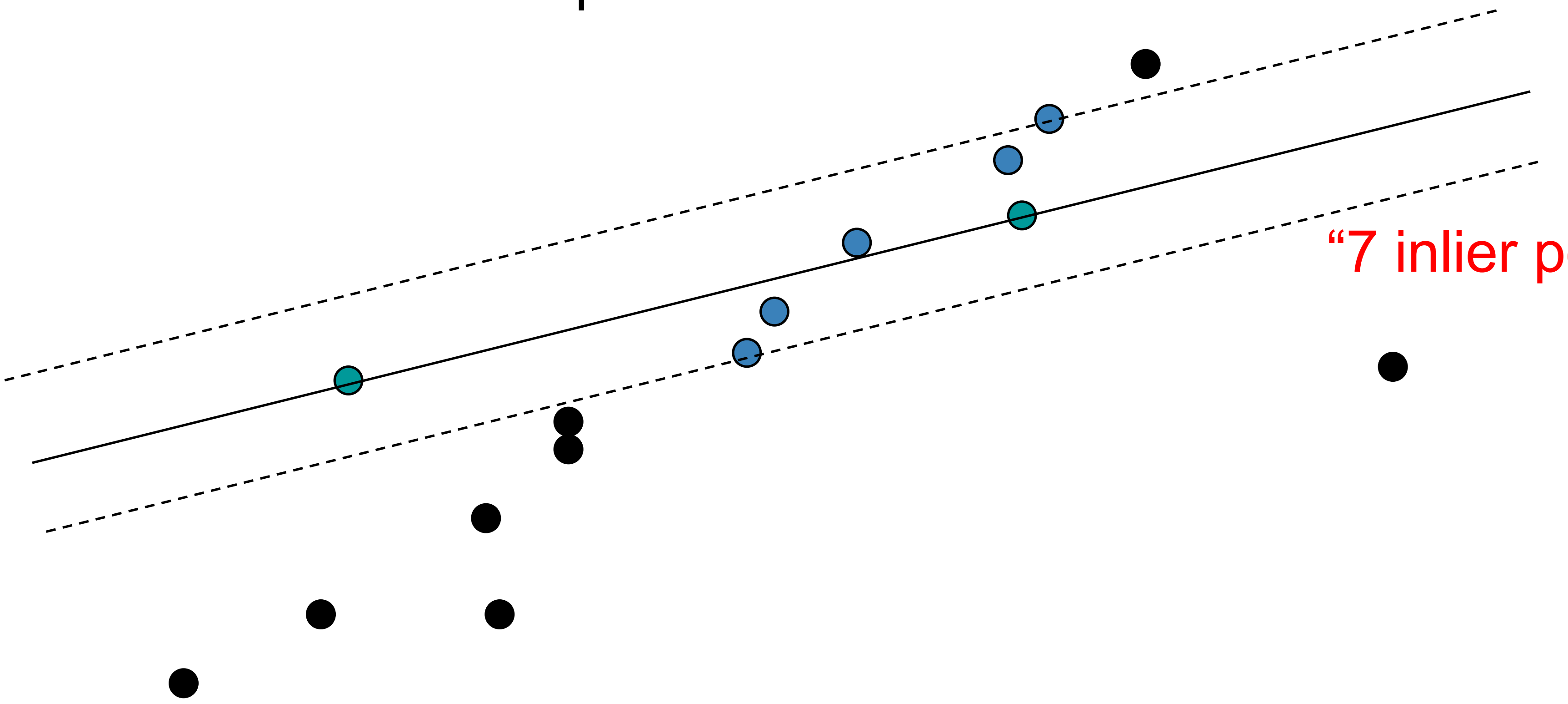
- Task: Estimate the best line
  - Edges can be noisy. To account for this, let's say that the line is somewhere between the dashed lines



Slide credit: Jinxiang Chai

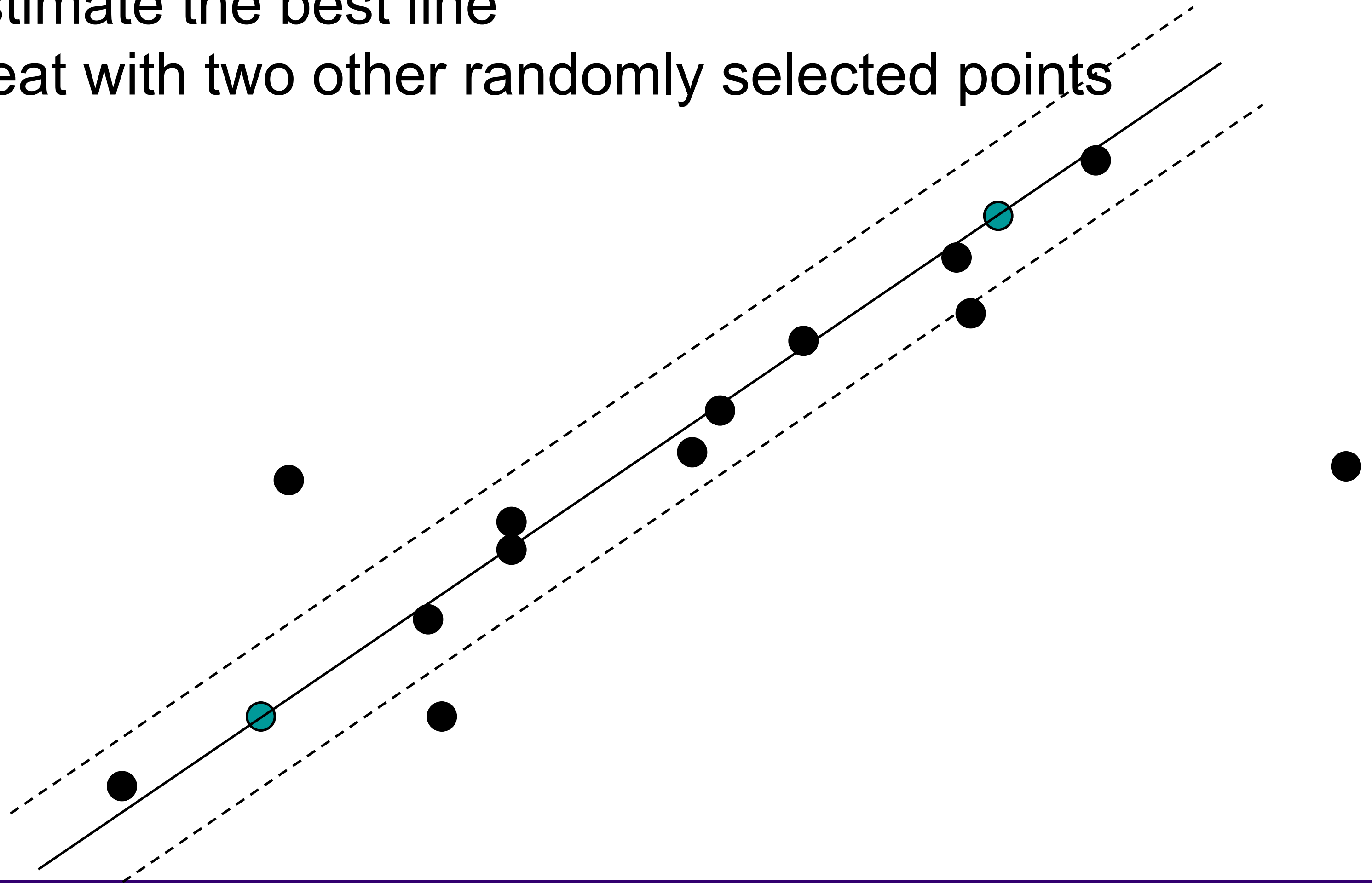
RANSAC Line Fitting Example

- Task: Estimate the best line
  - Calculate the number of points that lie within the dashed lines



RANSAC Line Fitting Example

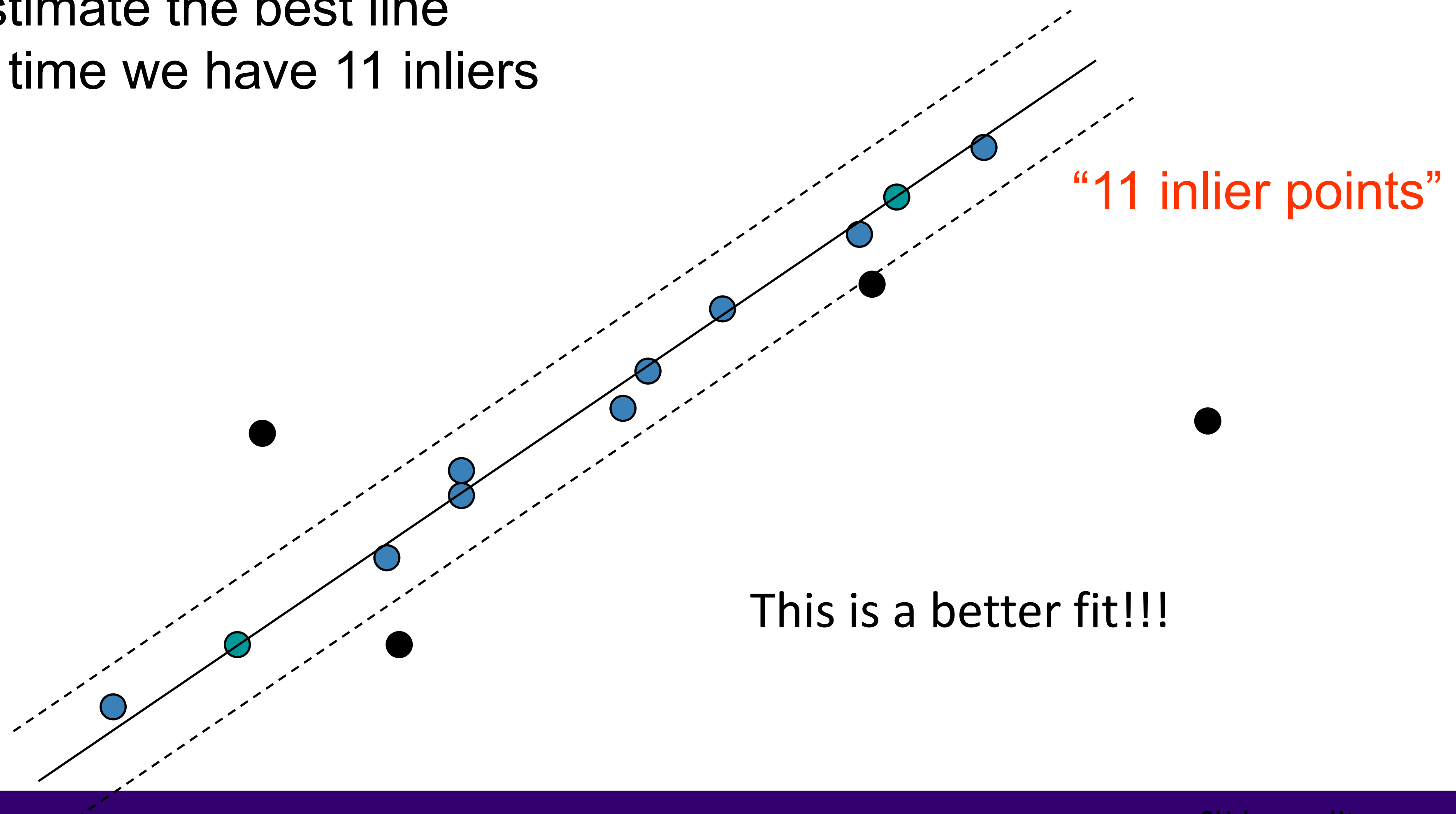
- Task: Estimate the best line
  - Repeat with two other randomly selected points



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

- Task: Estimate the best line
  - This time we have 11 inliers



The RANSAC algorithm [Fischler & Bolles 1981]

RANSAC loop:

Repeat for k iterations:

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

The RANSAC algorithm [Fischler & Bolles 1981]

RANSAC loop:

Repeat for k iterations:

1. Randomly select a *seed* subset of points on which to perform a model estimate (e.g., a group of edge points)
2. Compute parameters from seed group

The RANSAC algorithm [Fischler & Bolles 1981]

RANSAC loop:

Repeat for k iterations:

1. Randomly select a *seed* subset of points on which to perform a model estimate (e.g., a group of edge points)
2. Compute parameters from seed group
3. Find inliers for these parameters

Slide credit: Kristen Grauman

The RANSAC algorithm [Fischler & Bolles 1981]

<u>RANSAC loop</u>:

Repeat for k iterations:

1. Randomly select a *seed* subset of points on which to perform a model estimate (e.g., a group of edge points)
2. Compute parameters from seed group
3. Find inliers for these parameters
4. If the number of inliers is larger than the best so far, save these parameters and the inliers

Slide credit: Kristen Grauman

The RANSAC algorithm [Fischler & Bolles 1981]

RANSAC loop:

Repeat for k iterations:

1. Randomly select a *seed* subset of points on which to perform a model estimate (e.g., a group of edge points)
2. Compute parameters from seed group
3. Find inliers for these parameters
4. If the number of inliers is larger than the best so far, save these parameters and the inliers

If number of inliers in the best line is < m, return no line

The RANSAC algorithm [Fischler & Bolles 1981]

RANSAC loop:

Repeat for k iterations:

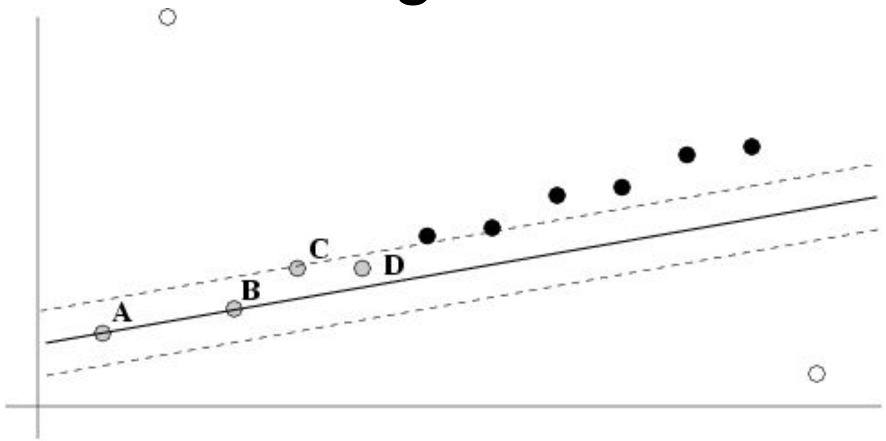
1. Randomly select a *seed* subset of points on which to perform a model estimate (e.g., a group of edge points)
2. Compute parameters from seed group
3. Find inliers for these parameters
4. If the number of inliers is larger than the best so far, save these parameters and the inliers

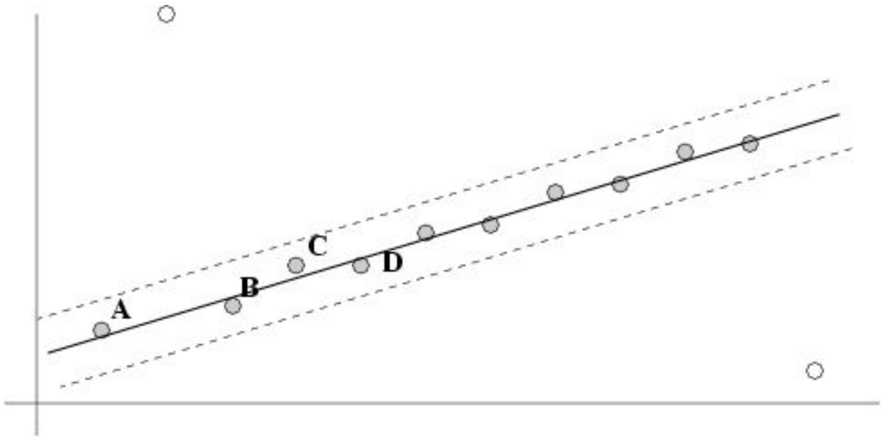
If number of inliers in the best line is < m, return no line

Else re-calculate the final parameters with all the inliers

Final step: Refining the parameters

- The best parameters were computed using a seed set of n points.
- We use these points to find the inliers.
- We can improve the parameters by estimating over all inliers (e.g. with standard least-squares minimization).
- But this may change the inliers, so repeat this last step until there is no change in inliers.





The RANSAC algorithm [Fischler & Bolles 1981]

<u>RANSAC loop</u>:

Repeat for k iterations:

1. Randomly select a *seed* subset of points on which to perform a model estimate (e.g., a group of edge points)
2. Compute parameters from seed group
3. Find inliers for these parameters
4. If the number of inliers is larger than the best so far, save these parameters and the inliers

If number of inliers in the best line is < m, return no line

Else re-calculate the final parameters with all the inliers

Slide credit: Kristen Grauman

The hyperparameters

1. How many points to sample in the seed set?
   a. We used 2 in the example above

The hyperparameters

1. How many points to sample in the seed set?
   a. We used 2 in the example above
2. How many times should we repeat?
   a. More repetitions increase computation but increase chances of finding best line

Slide credit: Kristen Grauman

The hyperparameters

1. How many points to sample in the seed set?
   a. We used 2 in the example above
2. How many times should we repeat?
   a. More repetitions increase computation but increase chances of finding best line
3. The threshold for the dashed lines
   a. Larger the gap between dashed lines, the more false positive inliers
   b. Smaller the gap, the more false negatives outliers

The hyperparameters

1. How many points to sample in the seed set?
   a. We used 2 in the example above
2. How many times should we repeat?
   a. More repetitions increase computation but increase chances of finding best line
3. The threshold for the dashed lines
   a. Larger the gap between dashed lines, the more false positive inliers
   b. Smaller the gap, the more false negatives outliers
4. The minimum number of inliers to confidently claim there is a line
   a. Smaller the number, the more false negative lines
   b. Larger the number, the fewer lines we will find

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

Slide credit: David Lowe

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: $1 - (1 - w^n)^k$

$\Rightarrow$ Choose k high enough to keep this below desired failure rate.

Slide credit: David Lowe

RANSAC: Pros and Cons

- **Pros**:
  - General method suited for a wide range of parameter 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

Summary

- Sobel Edge detector
- Canny edge detector
- Hough Transform
- RANSAC

Optional reading:
Szeliski, Computer Vision: Algorithms and Applications, 2nd Edition
Sections 7.1, 8.1.4

Next time

Detectors and descriptors