Lecture 6

Lines and Features

So far: discrete derivatives in 3 ways

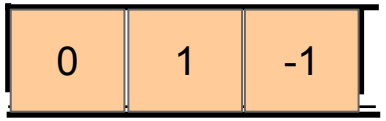
$$\frac{df}{dx} = f[x] - f[x-1]$$ Backward

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

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

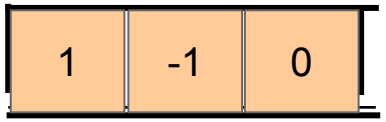
So far: Designing filters that perform differentiation

- Using Backward differentiation:



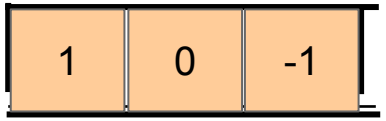
$$g[n,m] = f[n,m] - f[n,m-1]$$

- Using Forward differentiation:



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

- Using Central differentiation:



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

So far: Calculating gradient magnitude and direction

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

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

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

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

So far: A simple edge detector

- This theorem gives us a very useful property:

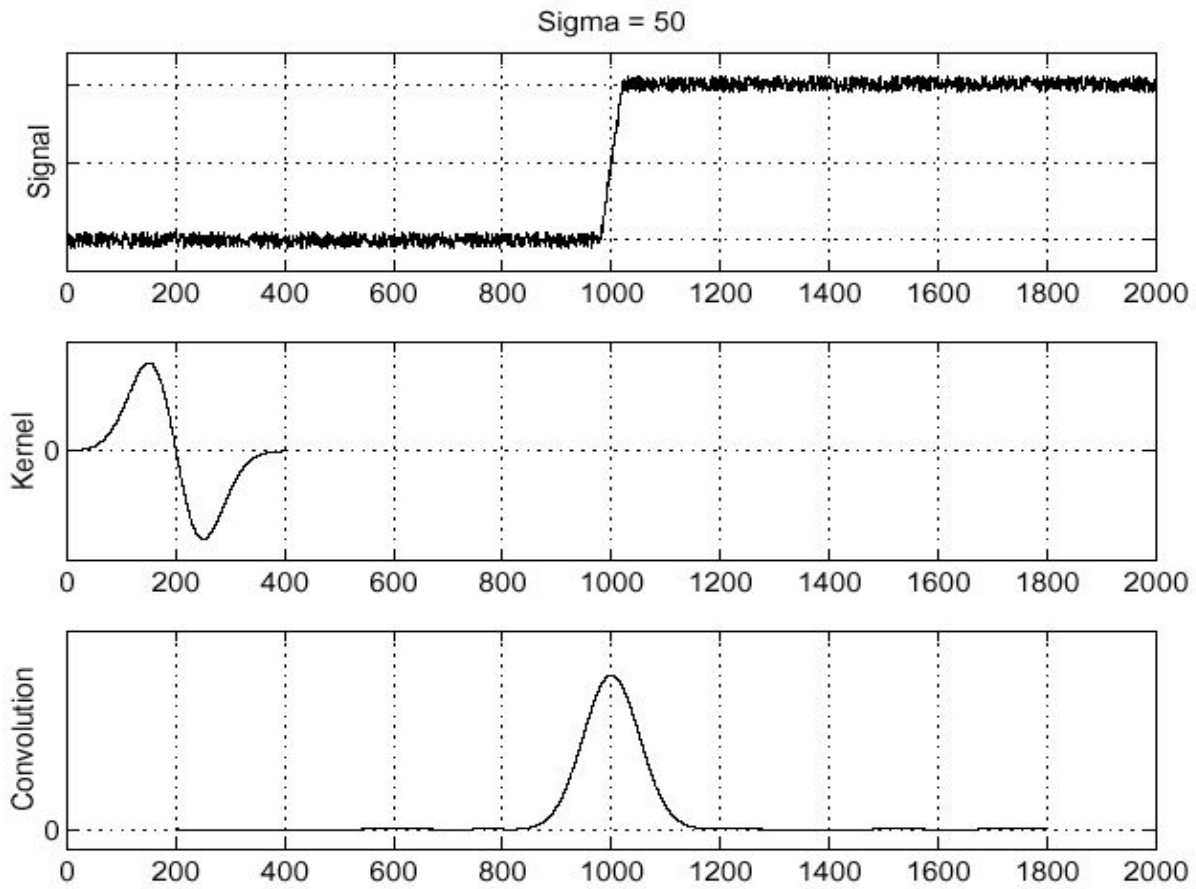
$$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$$

- This saves us one operation:

f

$\frac{d}{dx} g$

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



Source: S. Seitz

Today's agenda

- Canny edge detector
- Hough Transform
- RANSAC
- Local Invariant Features
- Harris Corner Detector

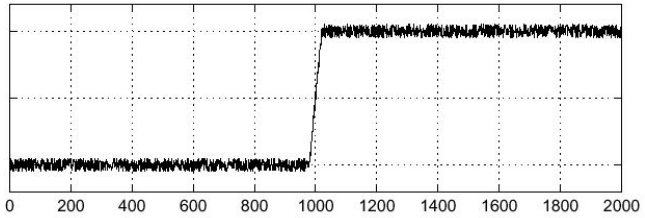
Optional reading:
Szeliski, Computer Vision: Algorithms and Applications, 2nd Edition
Sections 7.1, 8.1.4

What we will learn today

- Canny edge detector
- Hough Transform
- RANSAC
- Local Invariant Features
- Harris Corner Detector

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.

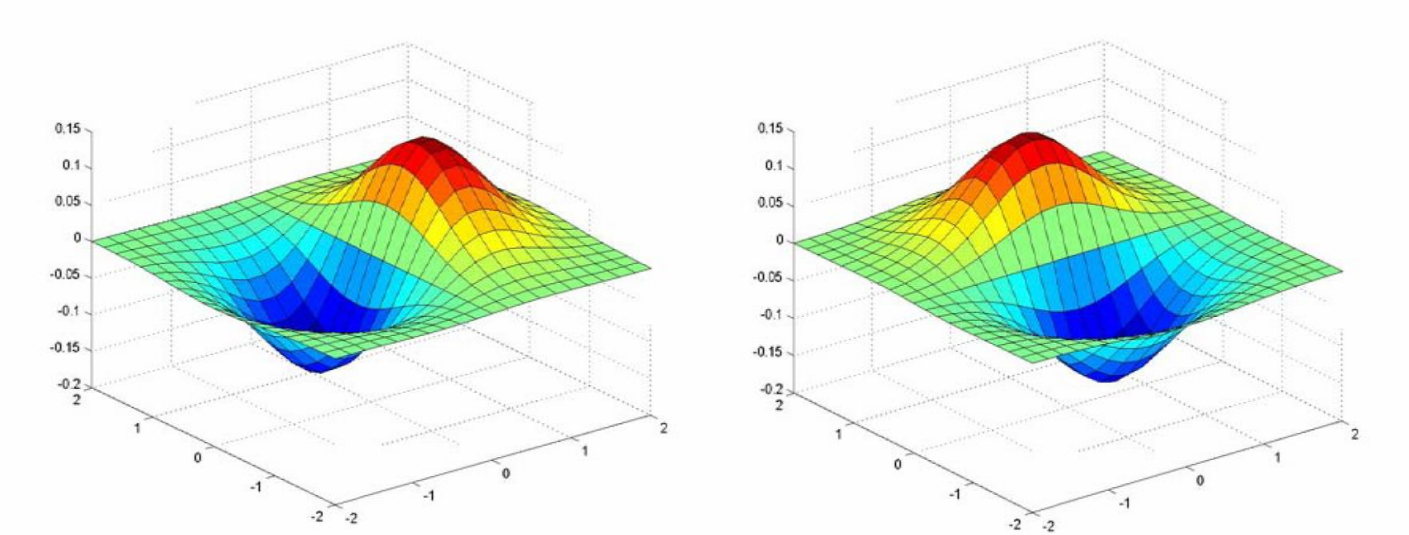
Canny edge detector

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

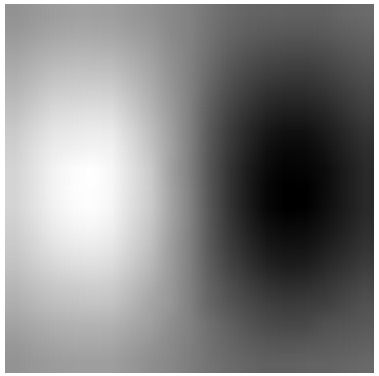
Derivative of Gaussian filter

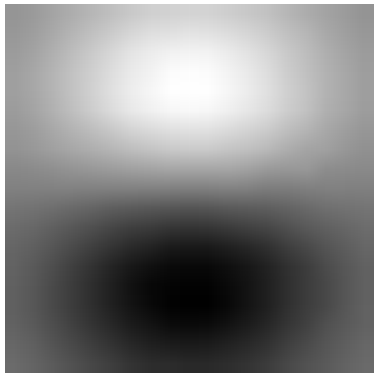


x-direction

y-direction

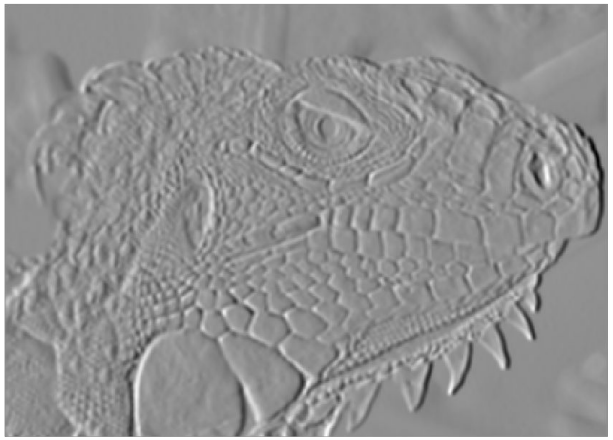
$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$



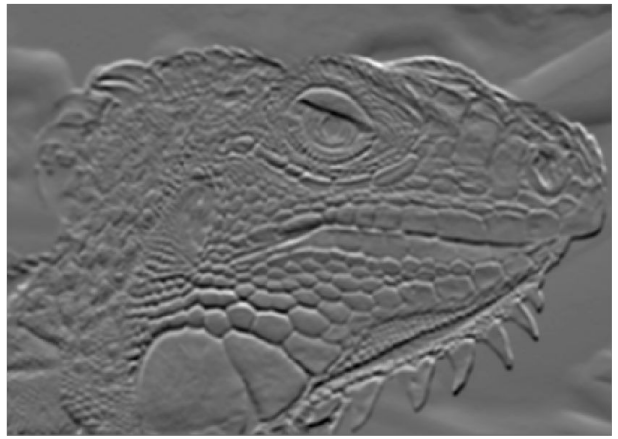


$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Compute gradients (DoG)



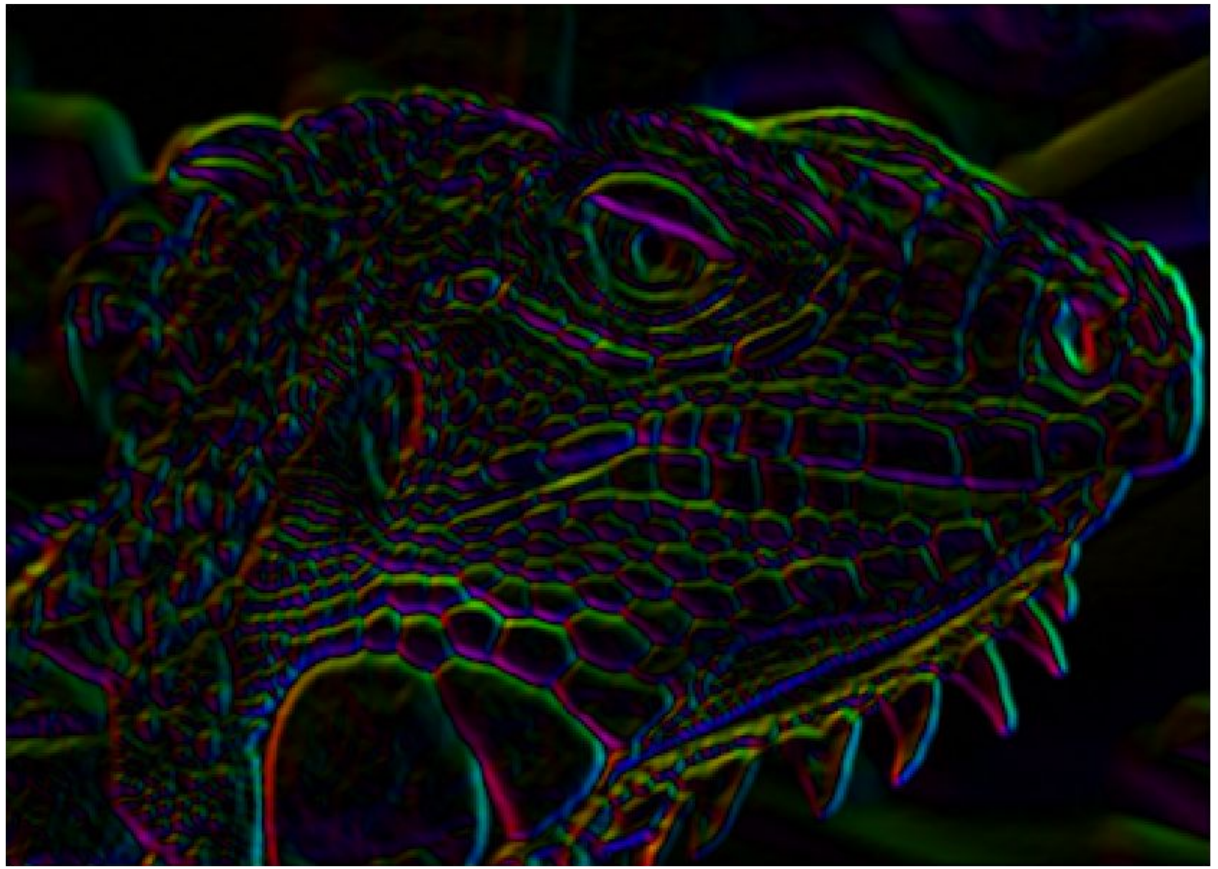
X-Derivative of Gaussian



Y-Derivative of Gaussian

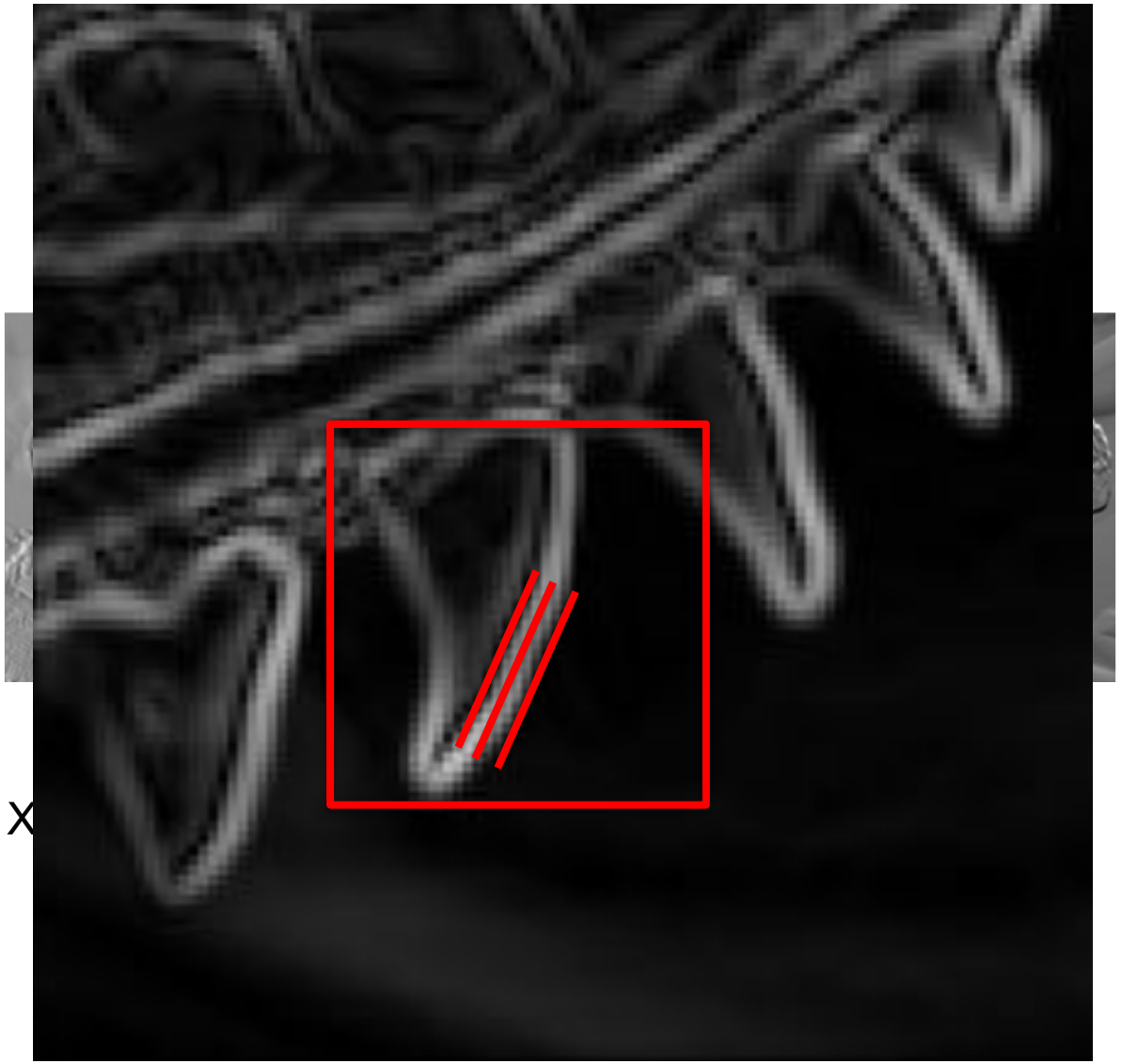
Gradient Magnitude

Get orientation at each pixel



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

Compute gradients (DoG)



Gradient Magnitude

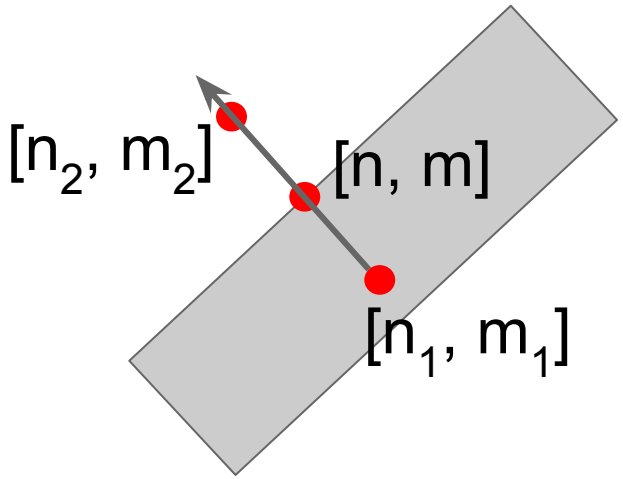
Canny edge detector

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

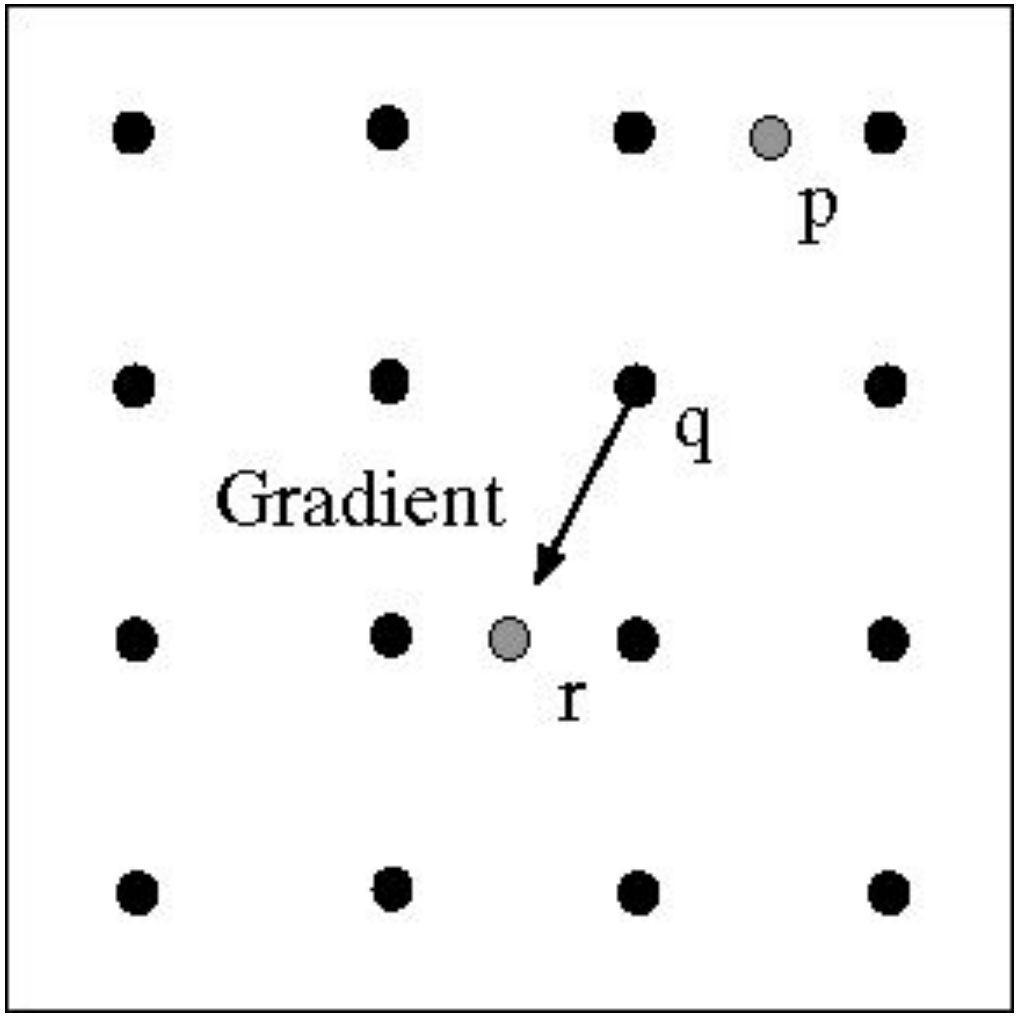
Remove spurious gradients



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

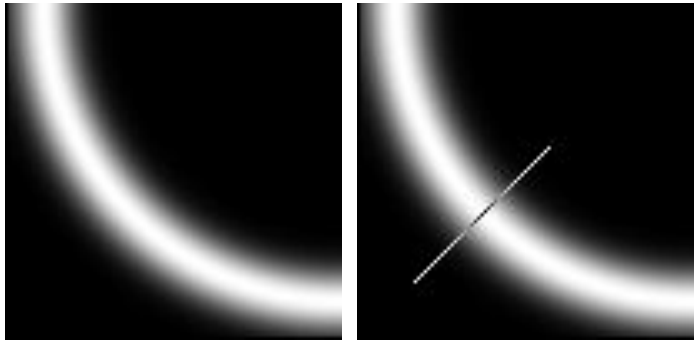
$$\text{If } G[n,m] = \begin{cases} G[n,m] & \text{if } G[n,m] > G[n_1,m_1] \text{ and } G[n,m] > G[n_2,m_2] \\ 0 & \text{otherwise} \end{cases}$$

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

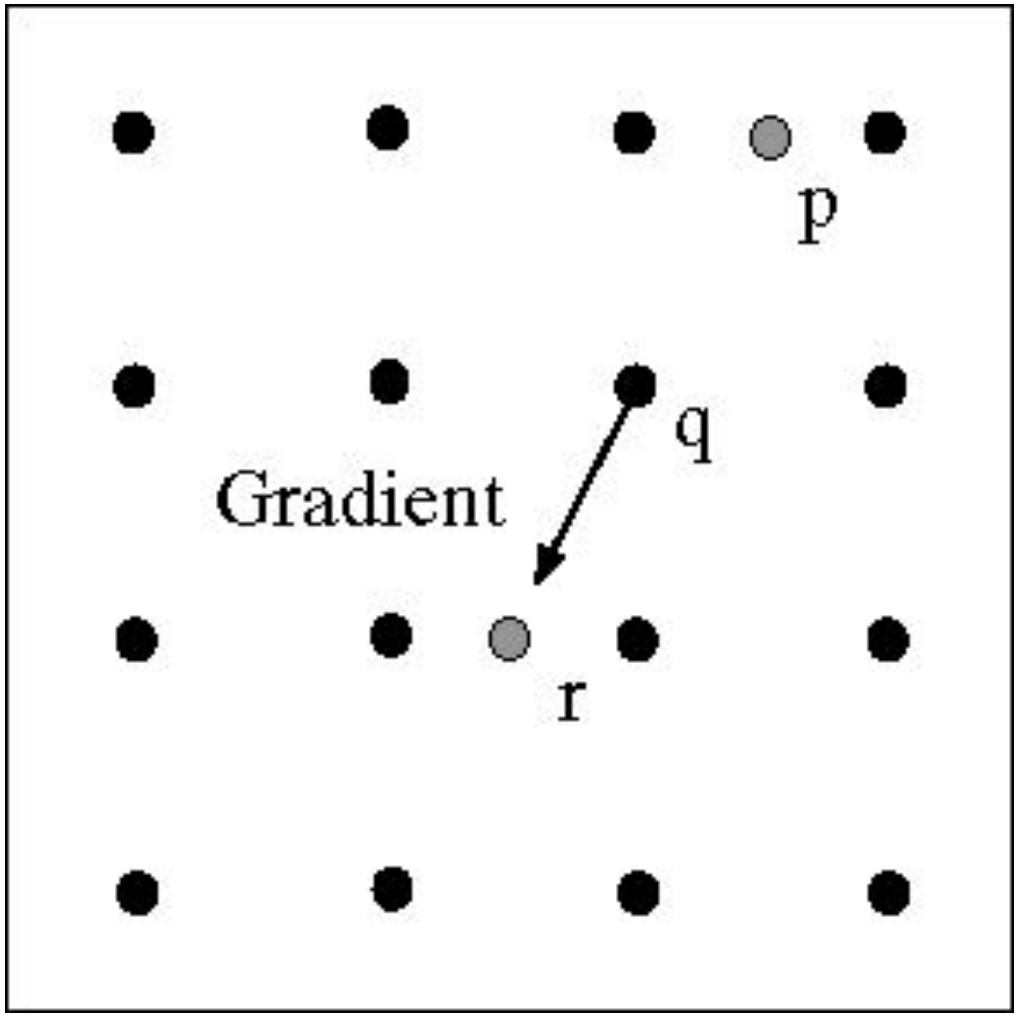


q is a maximum if the value is larger than those at both p and at r.

How should we calculate magnitude at G?



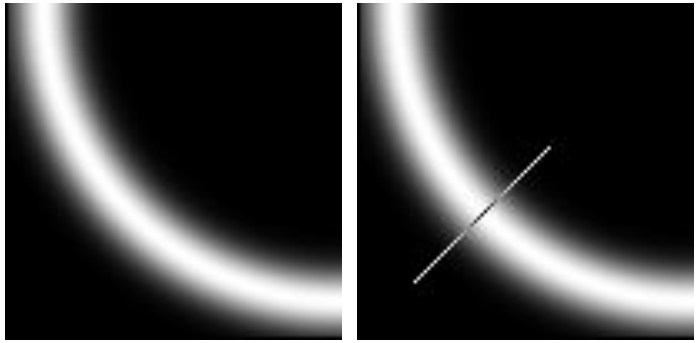
What if $p = [n_1, m_1]$ **or** $r = [n_2, m_2]$, is not a pixel location



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

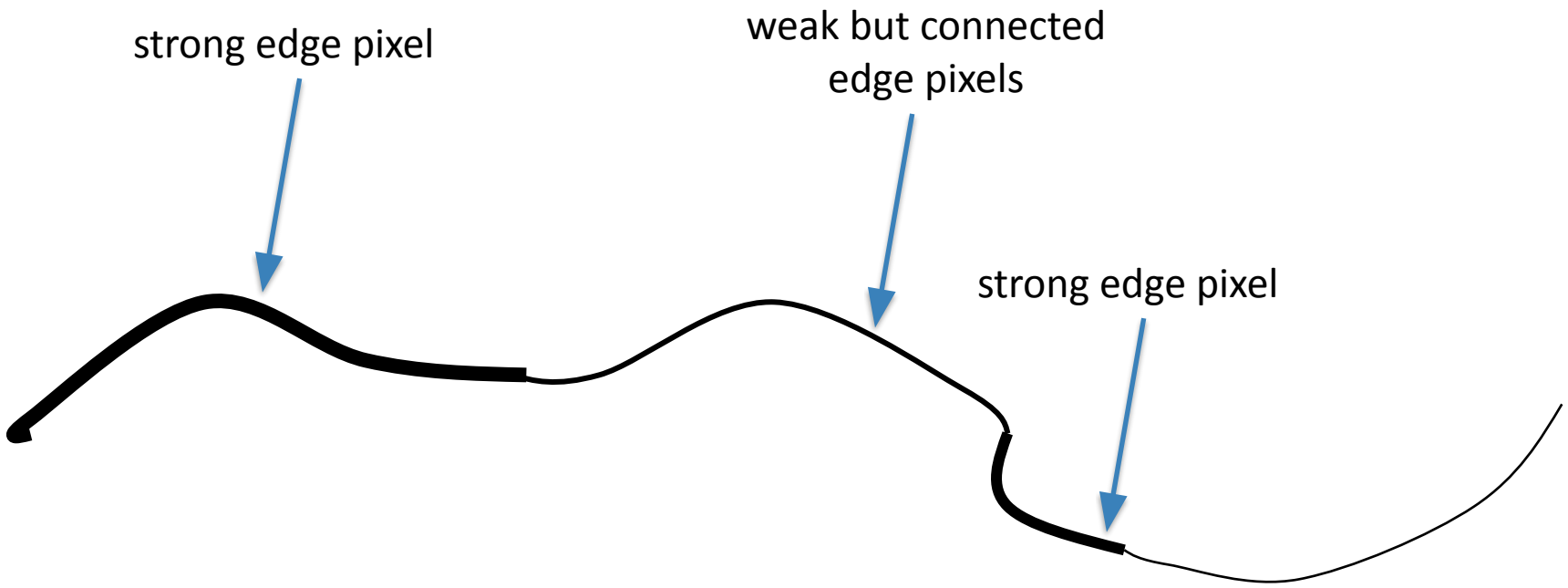
- Avoid breaking edges near the threshold value
- Define two thresholds: **Low** and **High**
  - If less than Low => **not an edge**
  - If greater than High => **strong edge**
  - If between Low and High => **weak edge**

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 $\sigma = 2$

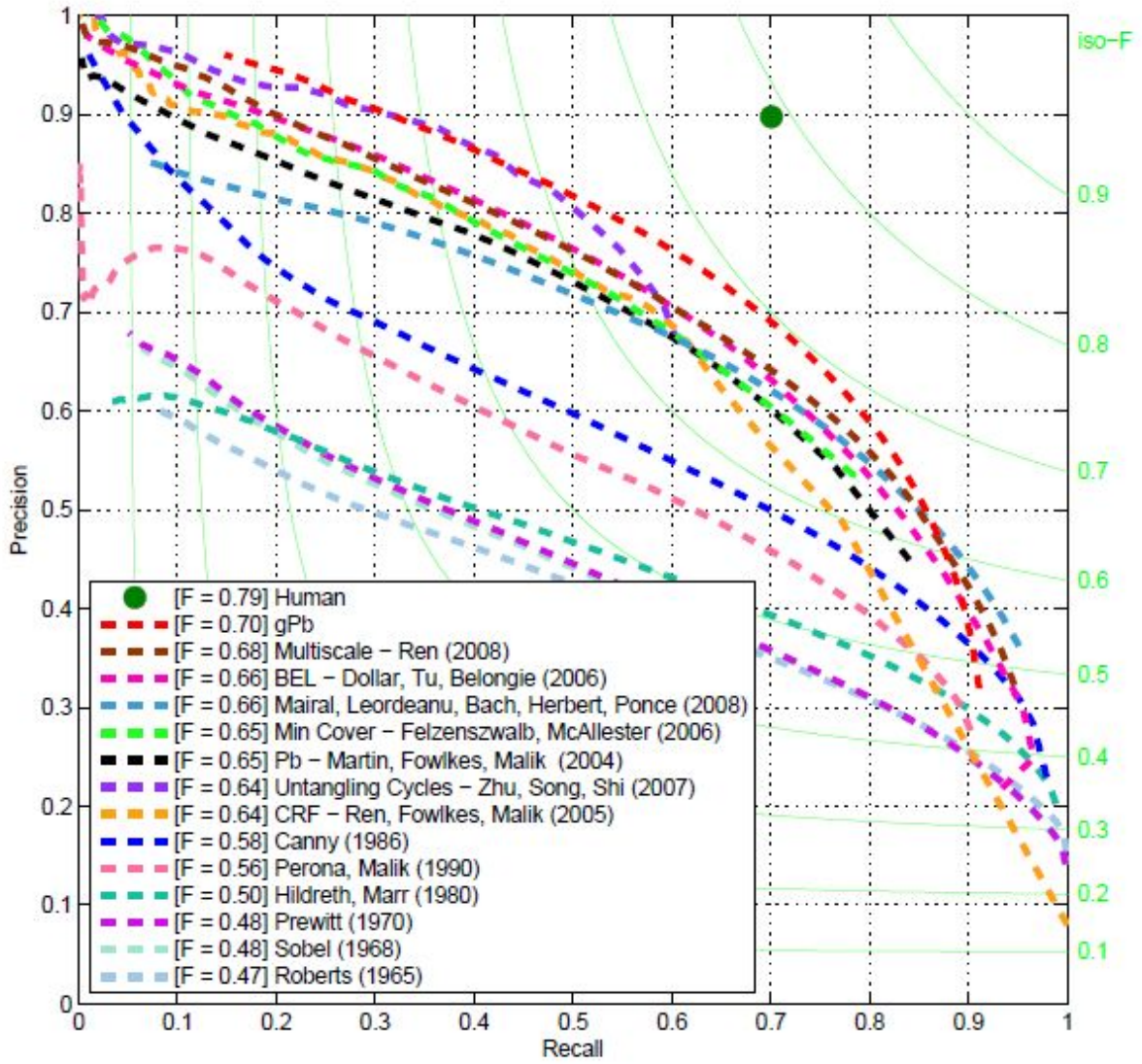
The choice of σ depends on desired behavior

- 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

- Canny edge detector
- Hough Transform
- Model fitting for line detection: RANSAC
- Local Invariant Features
- Harris Corner Detector

Hough transform

How Transform edge detections into lines

Original image

Edge image

Hough transform

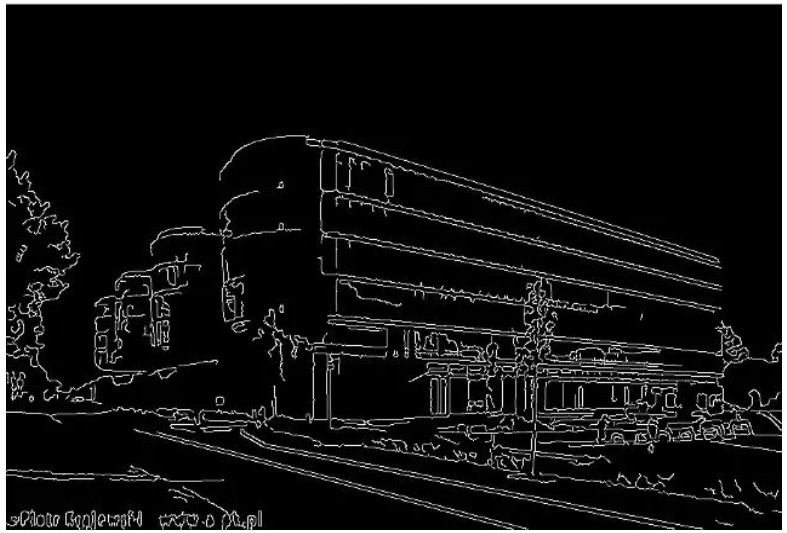
- It was introduced in 1962 (Hough 1962) and first used to find lines in images a decade later (Duda 1972).

- **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:

$$y = a*x + b$$

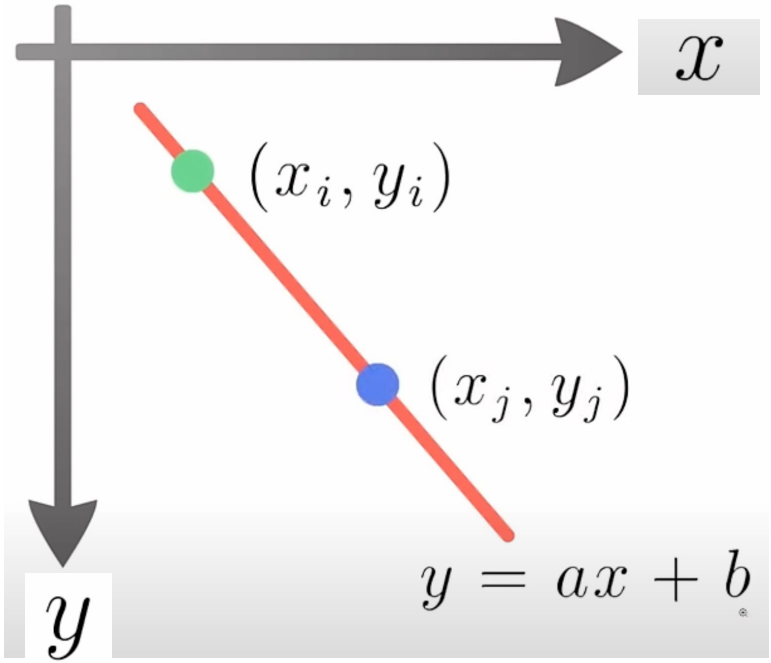
- How do we calculate the parameters (a, b)?

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

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

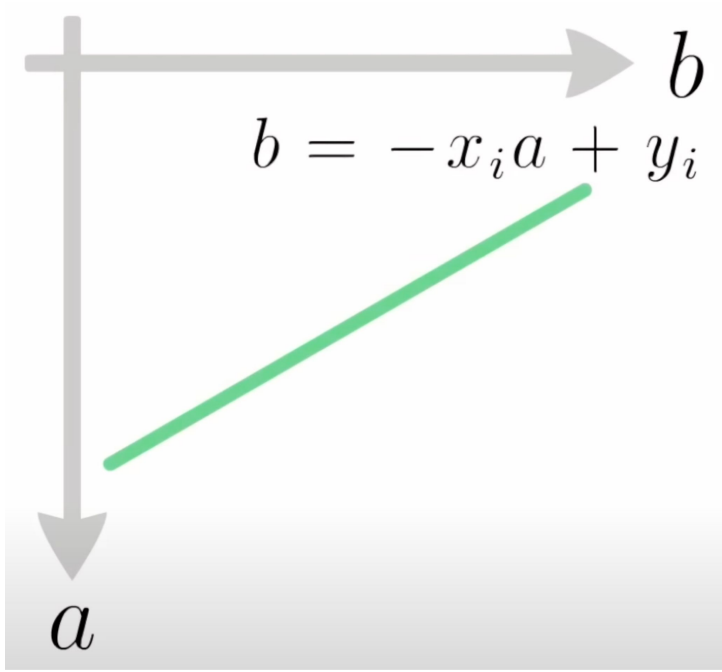
$$y_i = a*x_i + b$$

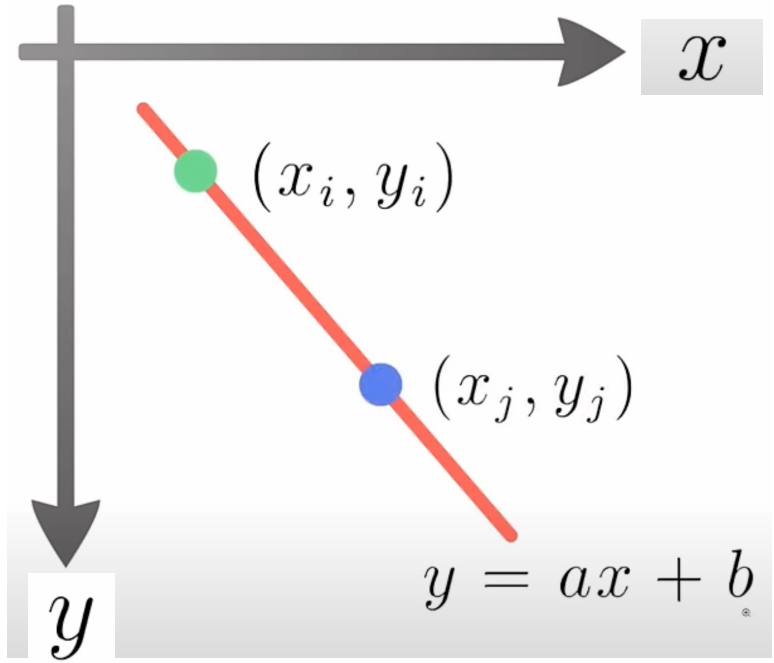
- This equation can be rewritten as follows:
  - $b = -a*x_i + y_i$

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.

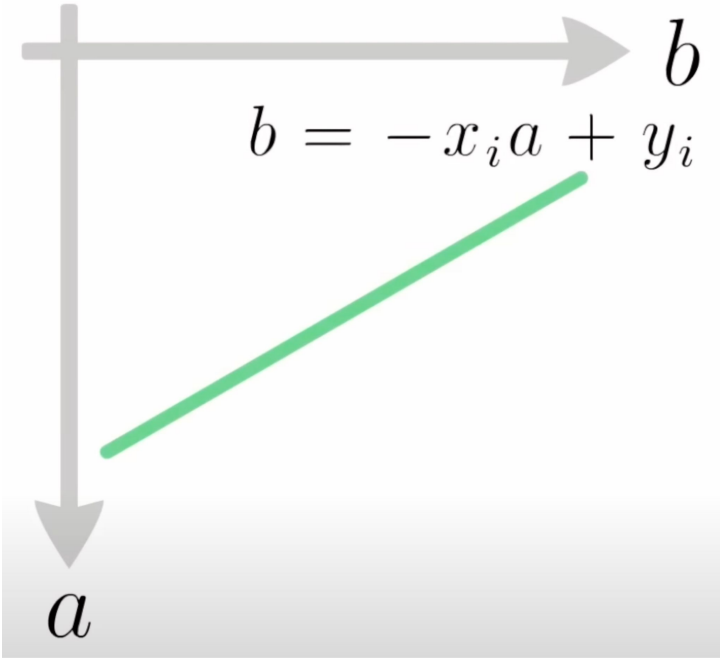




The Hough transform

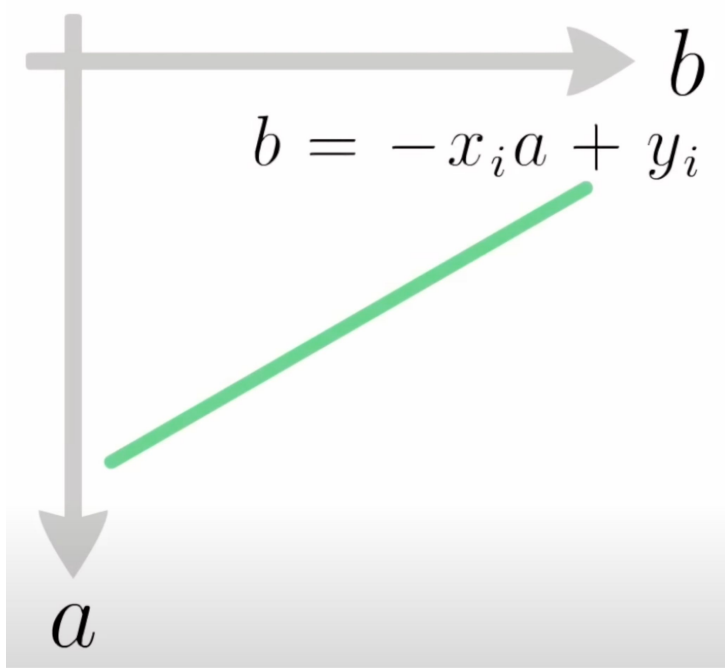
$y_i = a^*x_i + b$

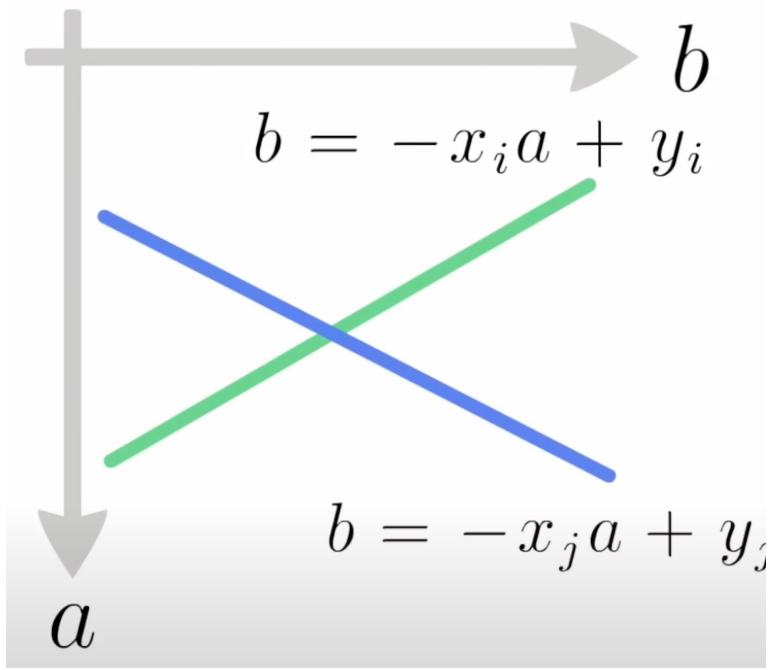
- $b = -a^*x_i + y_i$
- If our coordinates were (a,b) instead of (x, y):
  - We could say the above equation is a line in (a,b)-space
  - parameterized by x and y.
  - So: one point (x_i,y_i) gives a line in (a,b) space.



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.



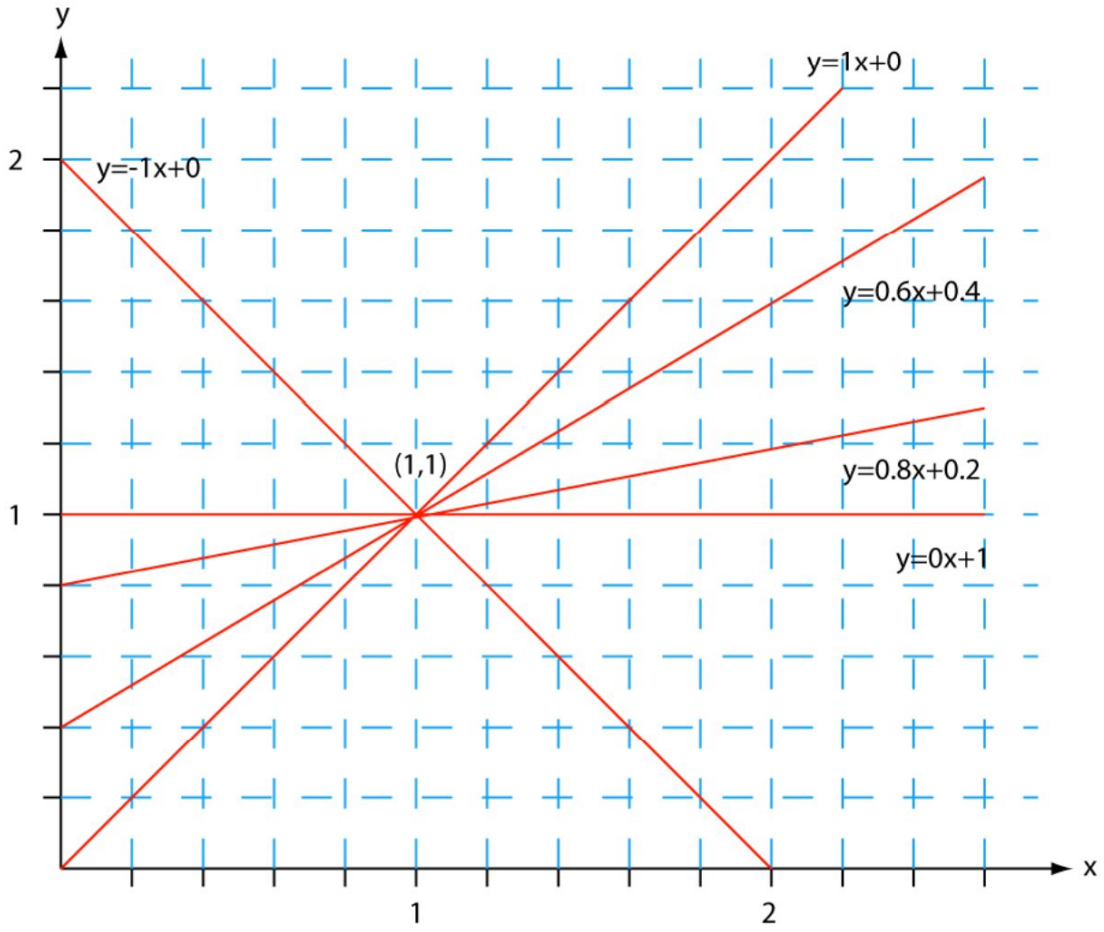


The Hough transform

- 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.

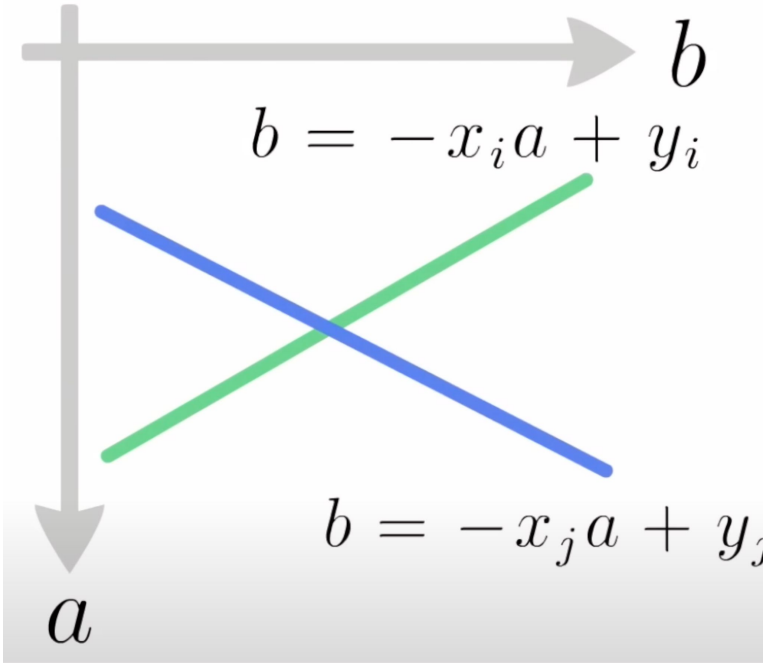


The need to quantize and “vote”

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.

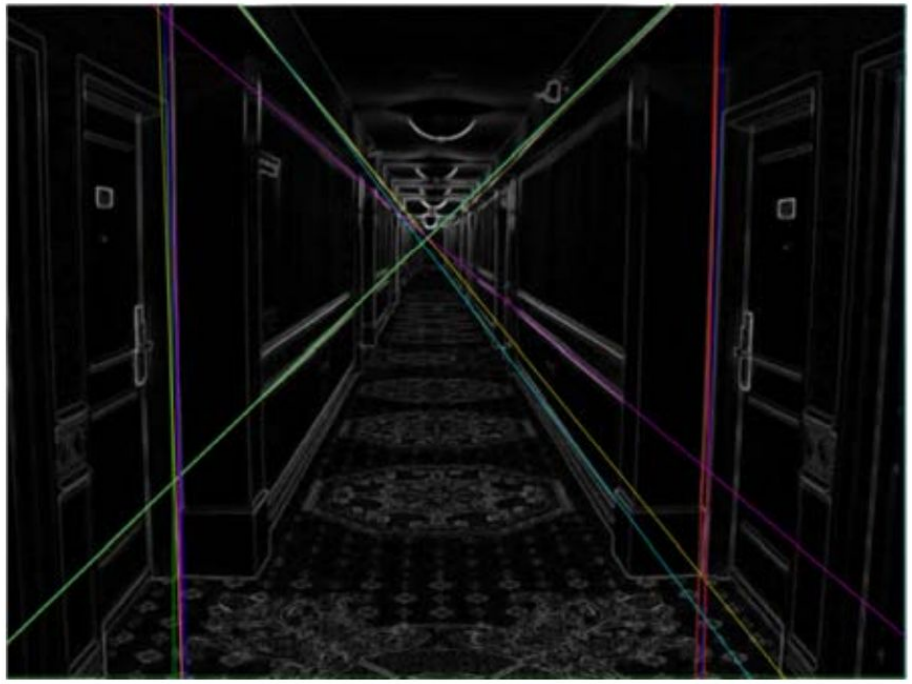
So, we need to count how many lines intersect at a point and keep the ones with high count

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:

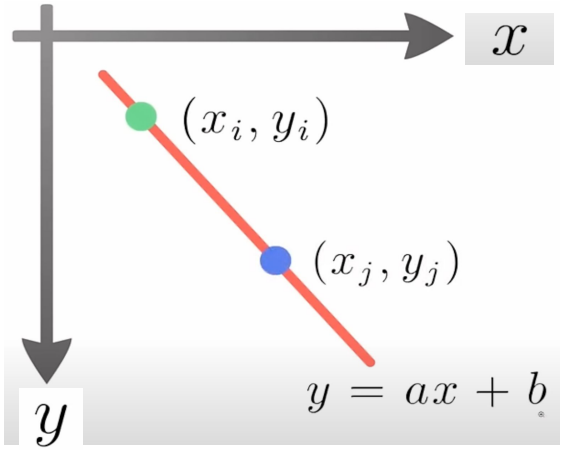


Other Hough transformations

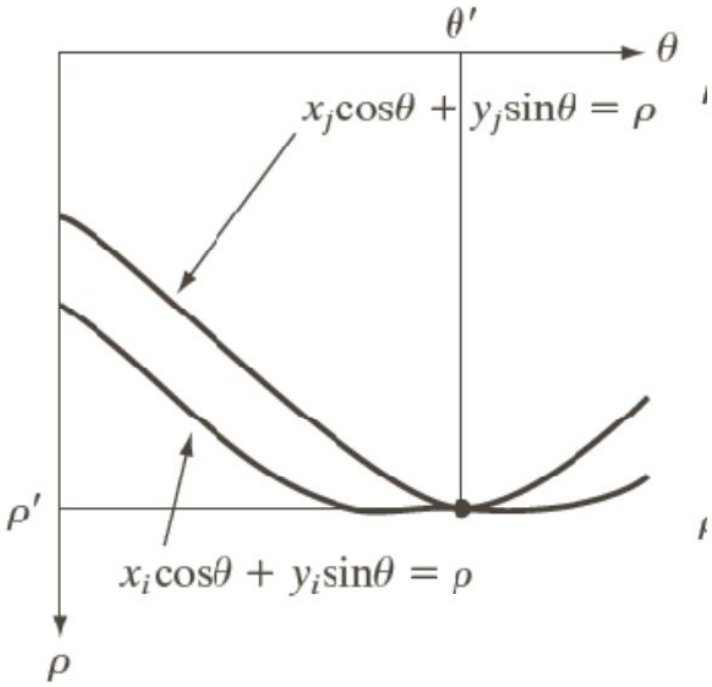
- We can represent lines as polar coordinates instead of y = a*x + b
- Polar coordinate representation:
  - x*cosθ + y*sinθ = ρ
- We can transform points in (x, y) space to curves in (ρ θ)-space
  - (x y) and (ρ θ)?

Other Hough transformations

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



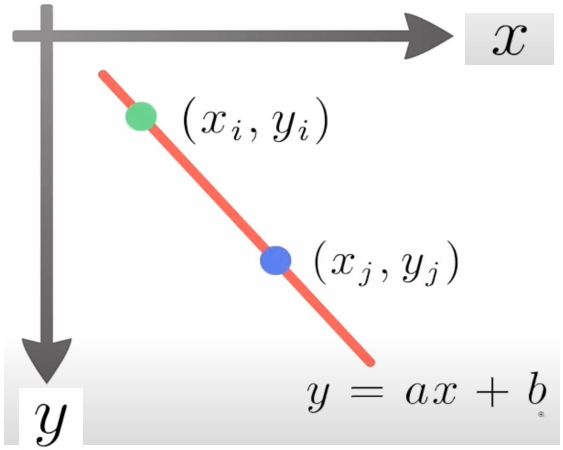
- Curves in (ρ, θ)-space intersect similarly like in (a, b)-space.

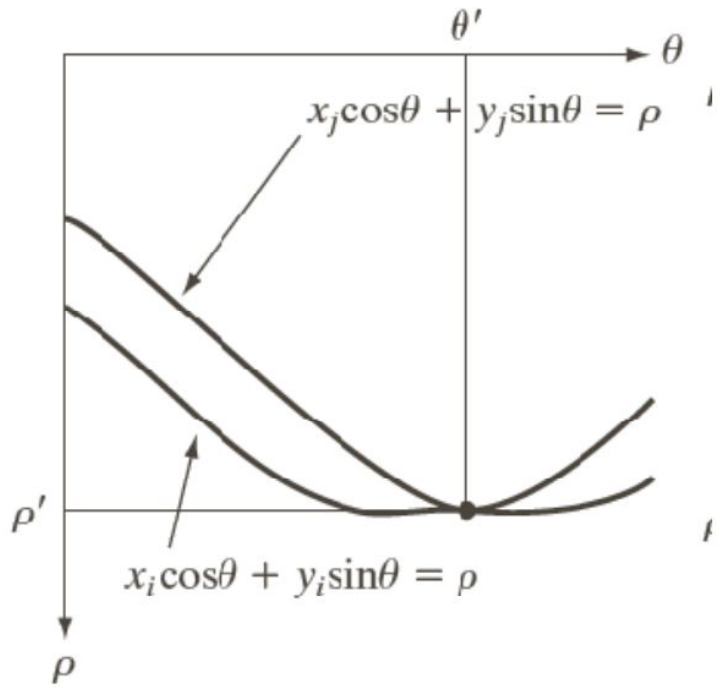


Other Hough transformations

- x*cosθ + y*sinθ = ρ

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



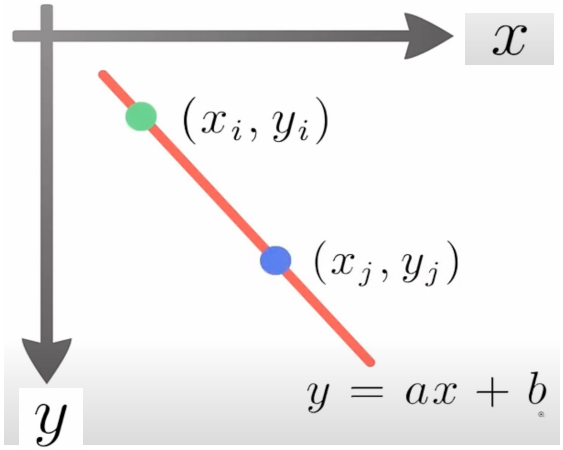


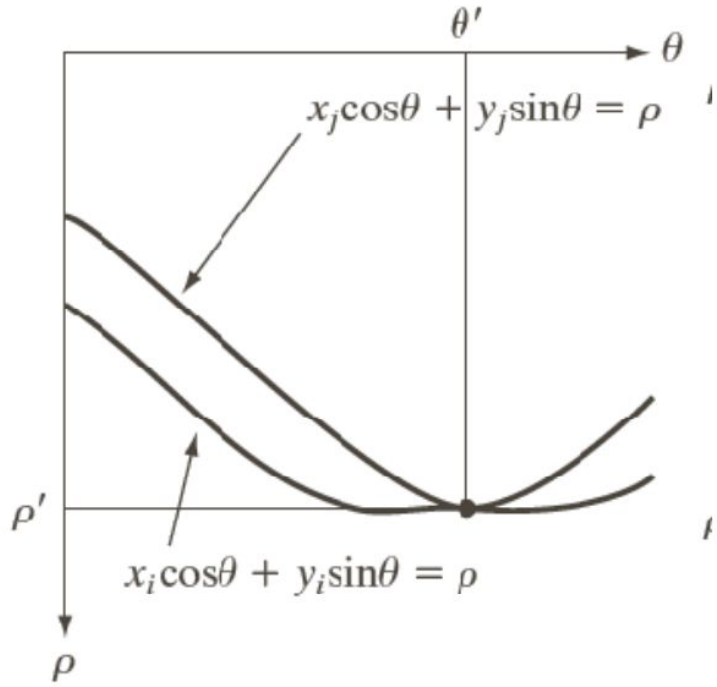
Other Hough transformations

- x*cosθ + y*sinθ = ρ

- 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

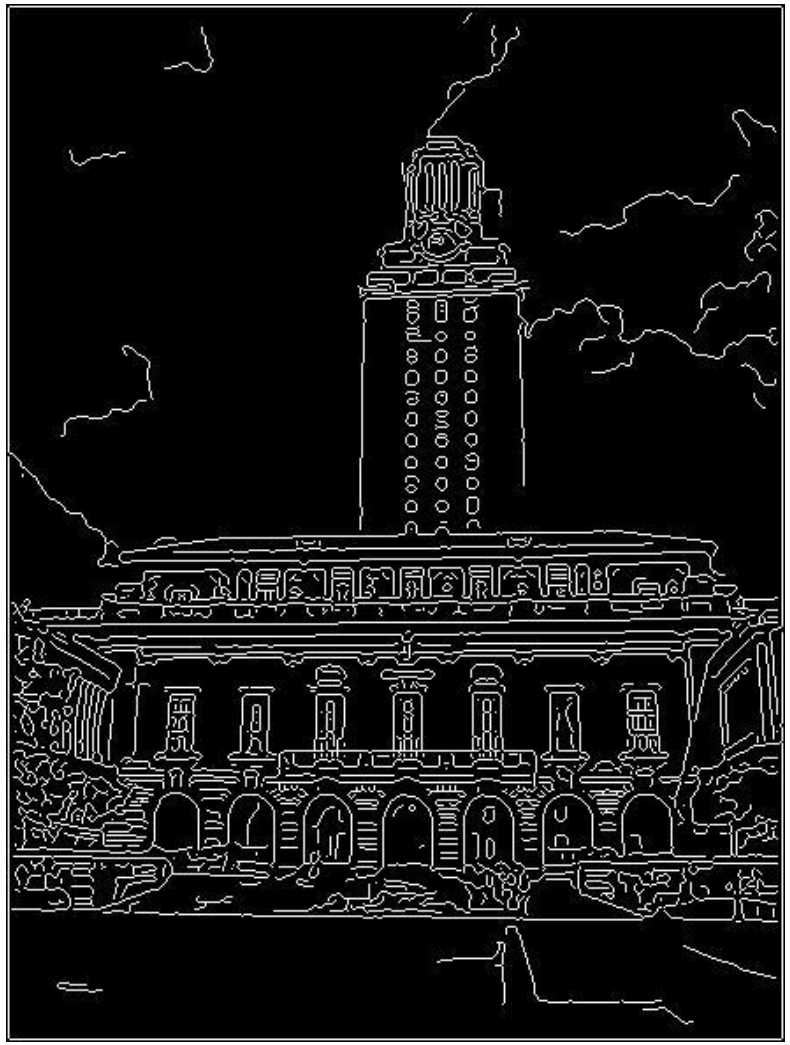
- Canny edge detector
- Hough Transform
- RANSAC
- Local Invariant Features
- Harris Corner Detector

Why is Hough transform inefficient?

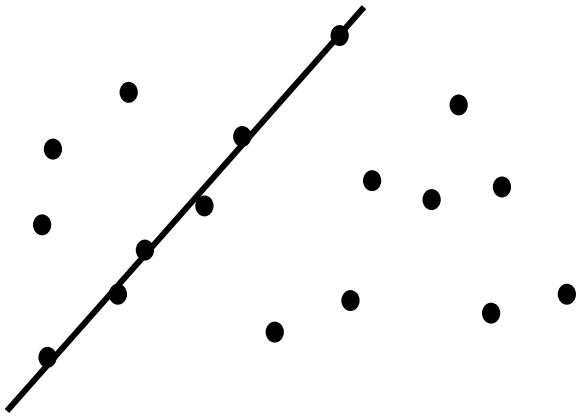
- It's not feasible to check all pairs of points to calculate possible lines. For example, Hough Transform algorithm runs in $O(N^2)$.
- Voting is a general technique where we let the each point vote for all models that are compatible with it.
  - Iterate through features, cast votes for parameters.
  - 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

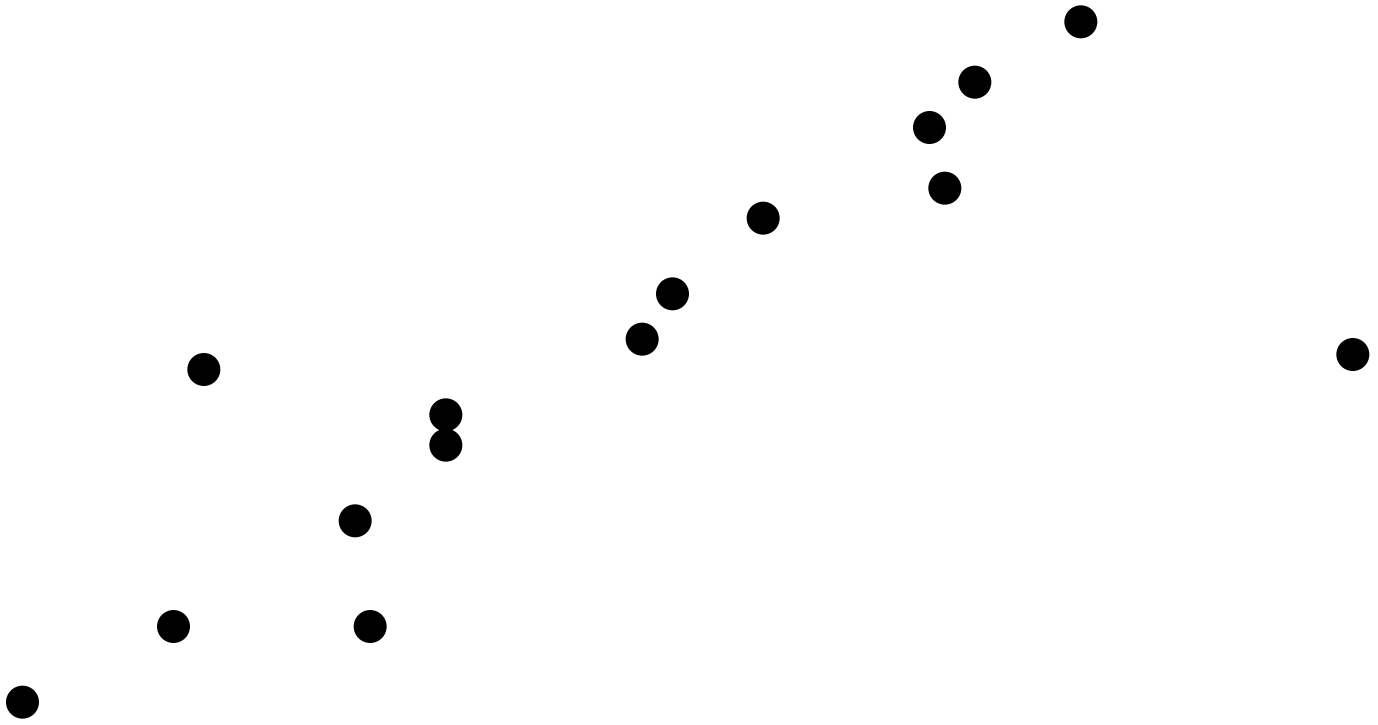
RANSAC [Fischler & Bolles 1981]

- 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

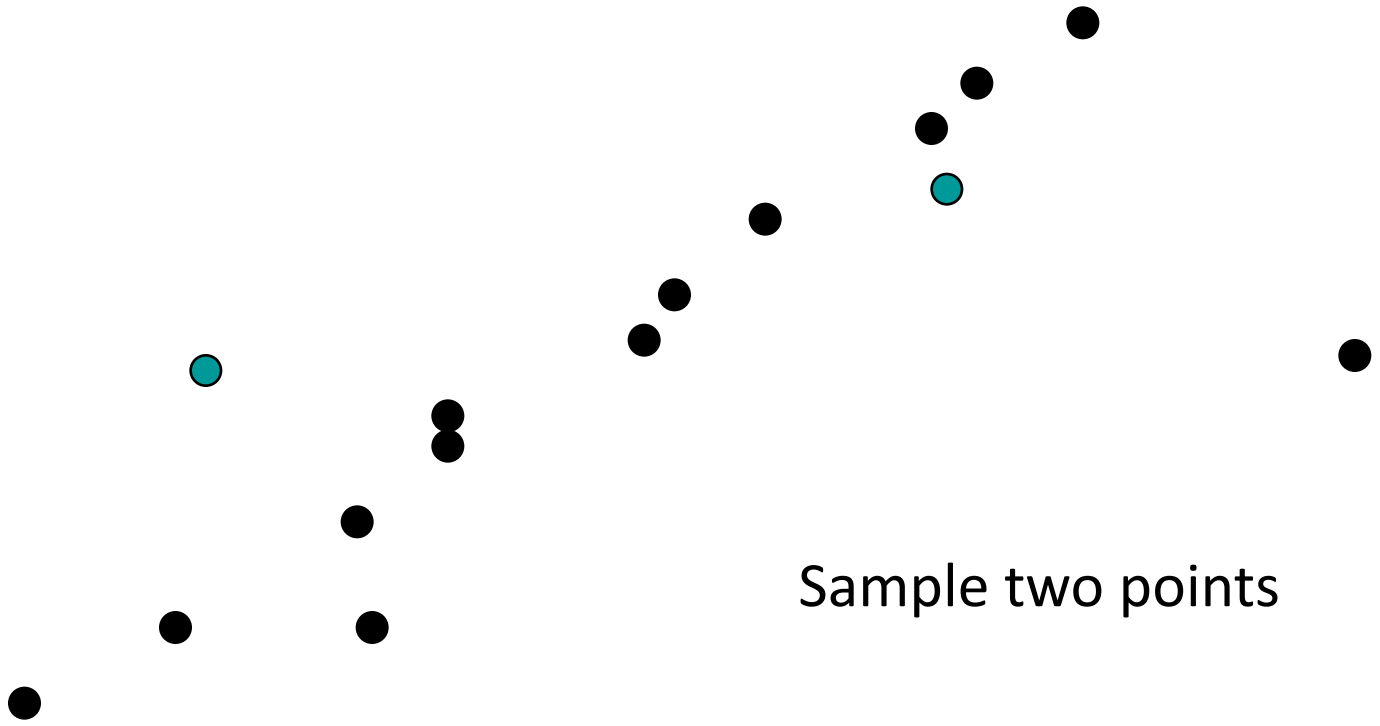
- Task: Estimate the best line
  - *Let's randomly select a subset of points and calculate a line*



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

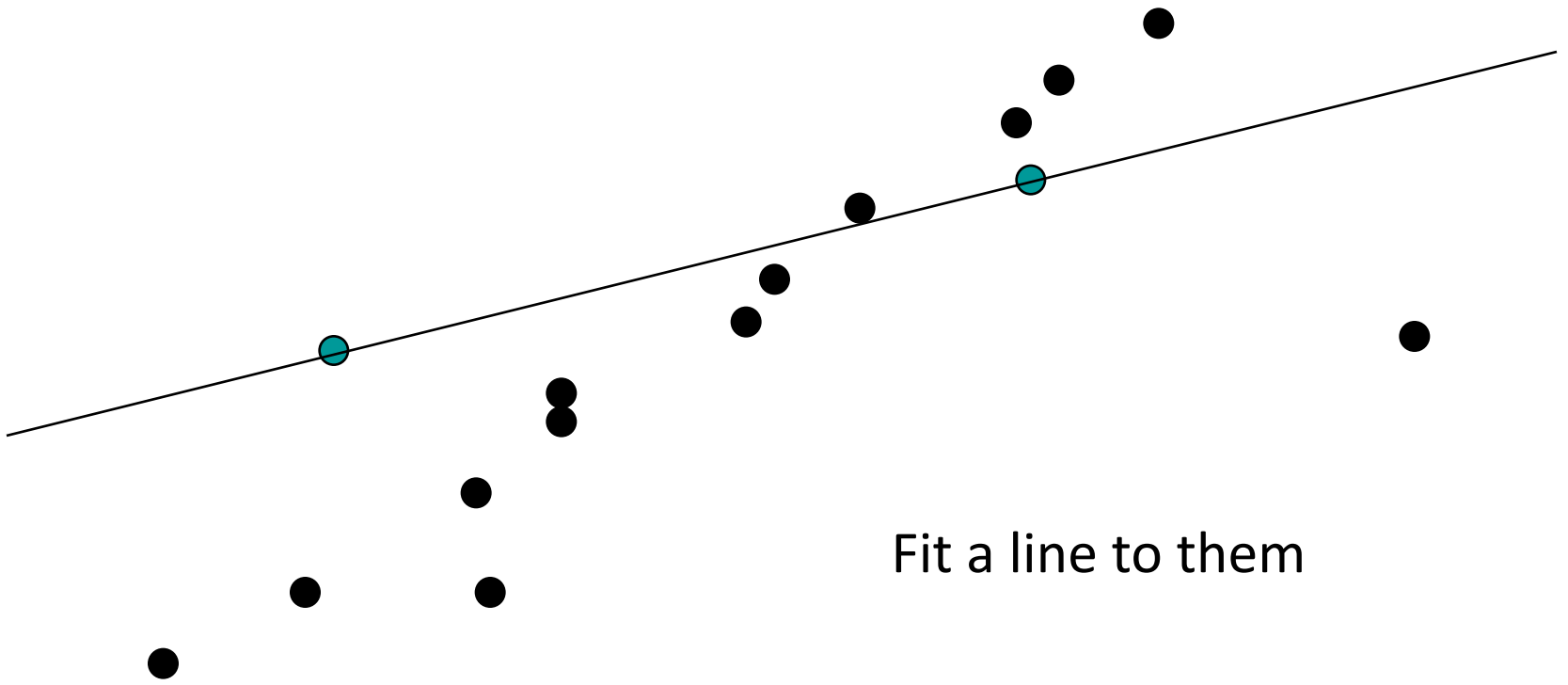
- Task: Estimate the best line
  - Let's select only 2 points as an example



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

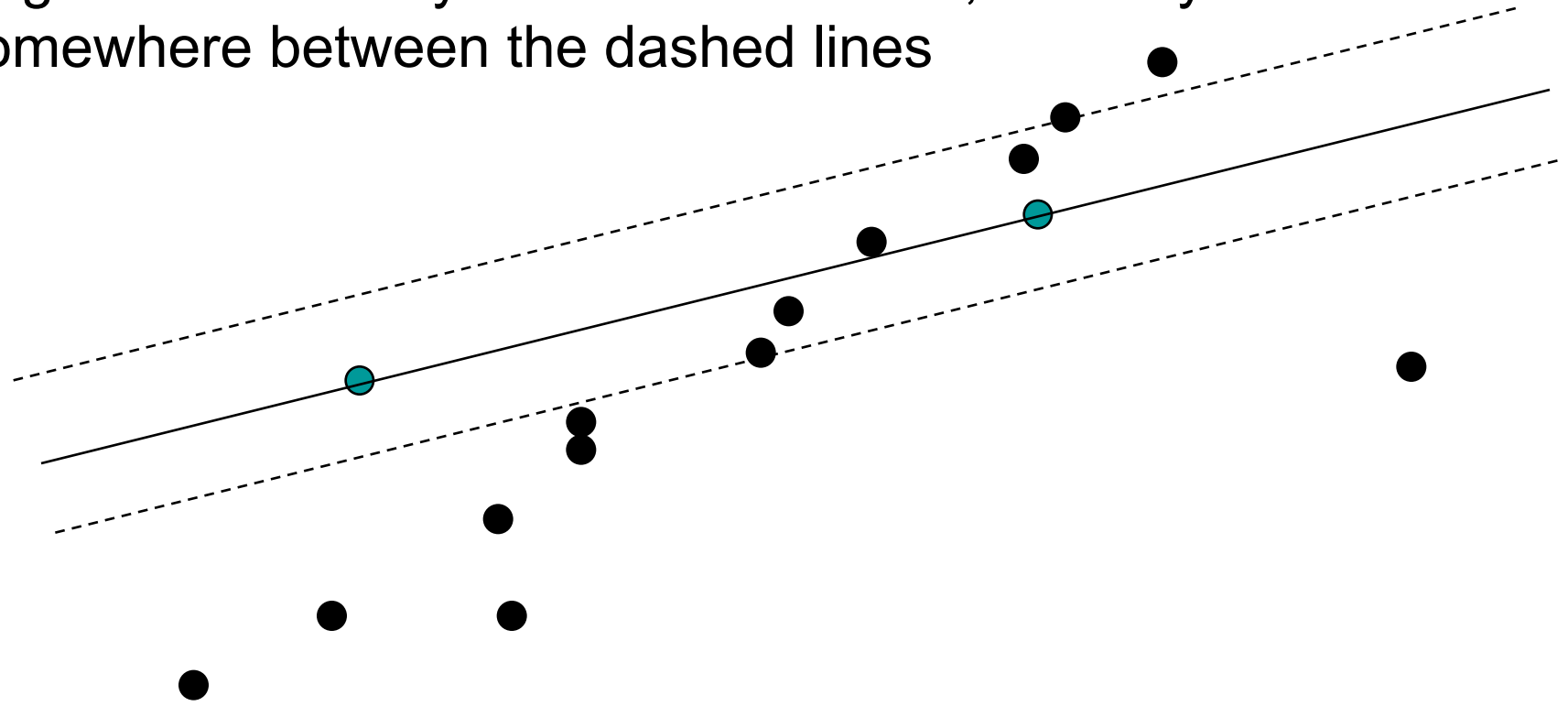
- Task: Estimate the best line
  - Calculate the line parameters



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

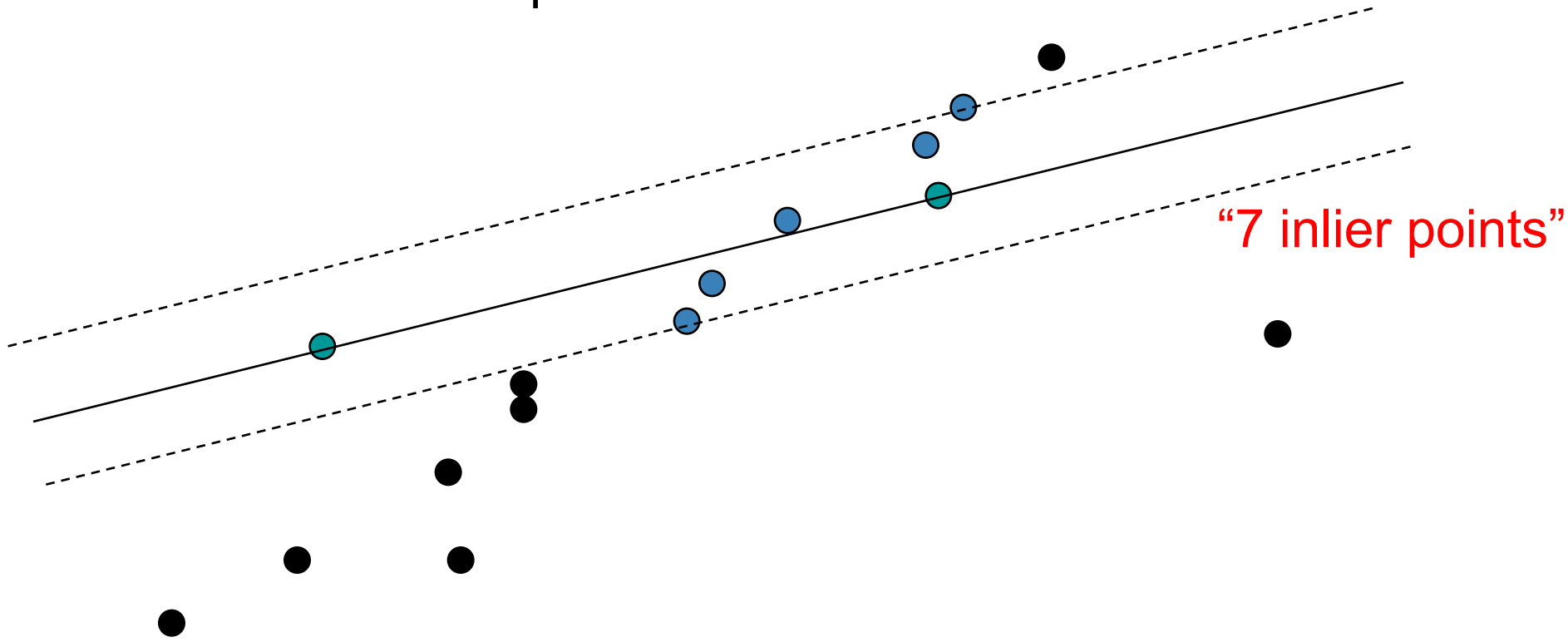
- Task: Estimate the best line
  - Edges can be noisy. To account for this, let's say that the line is somewhere between the dashed lines



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

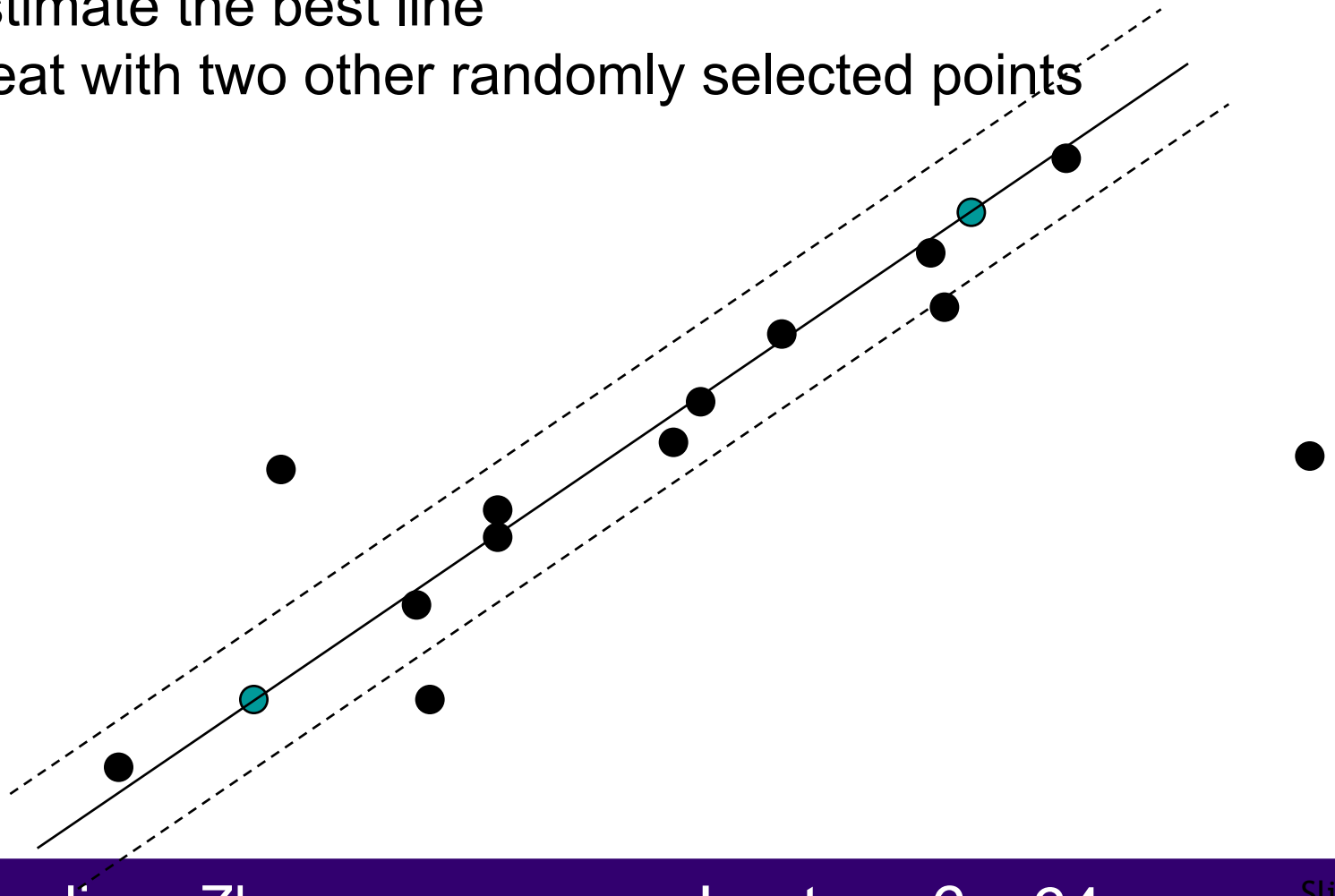
- Task: Estimate the best line
  - Calculate the number of points that lie within the dashed lines



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

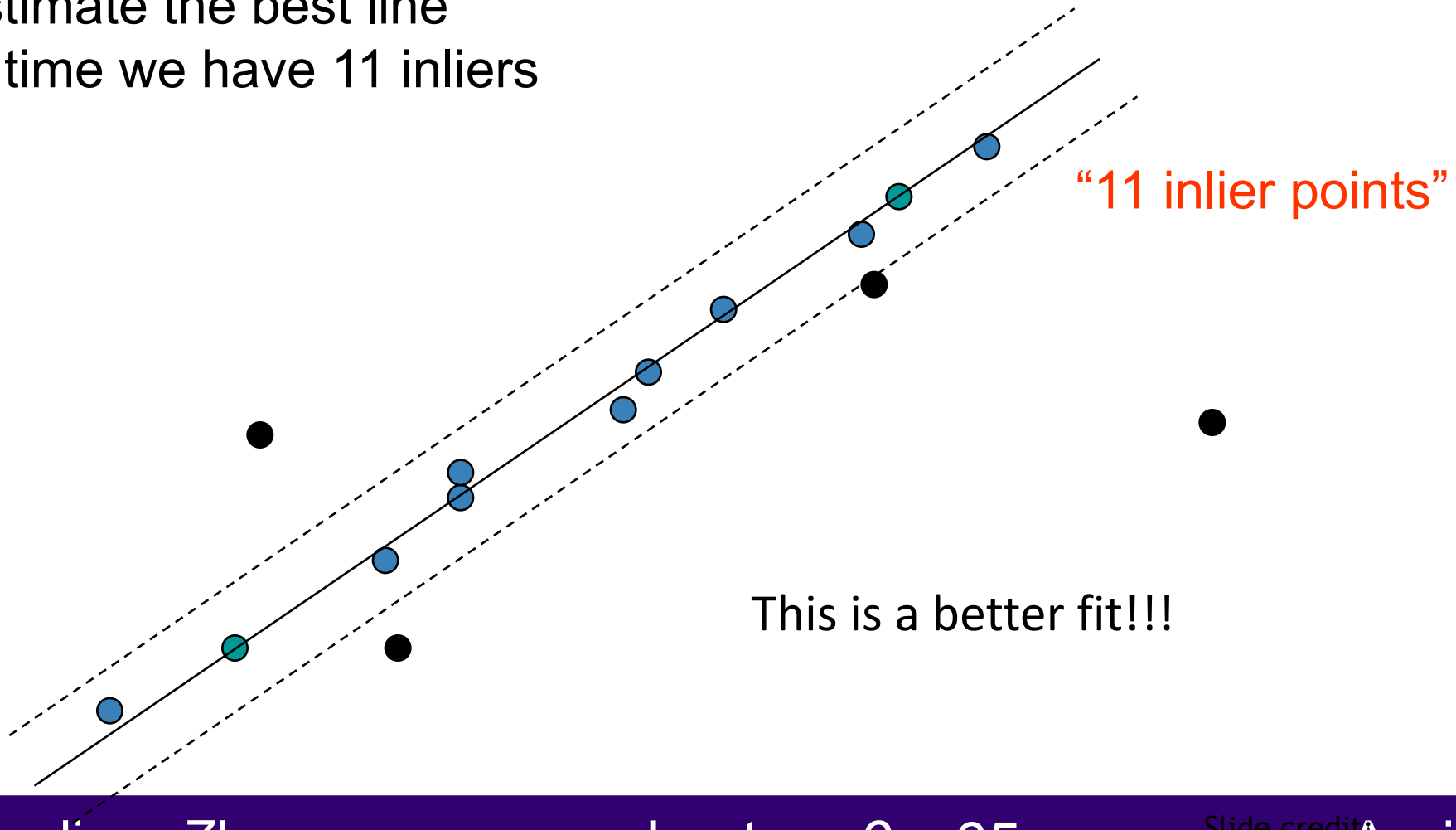
- Task: Estimate the best line
  - Repeat with two other randomly selected points



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

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



Slide credit: Jinxiang Chai

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)

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

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

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

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

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

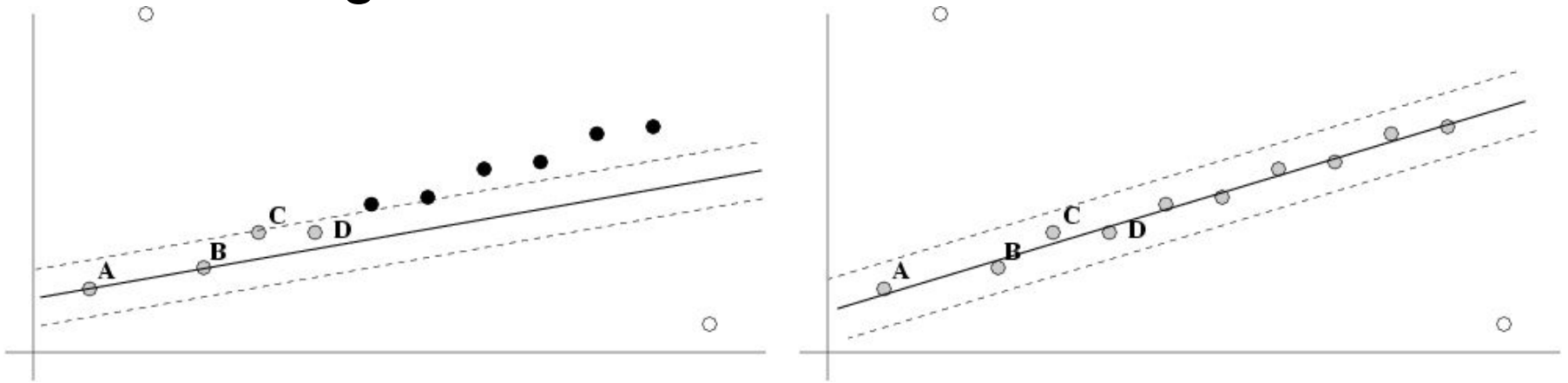
If number of inliers in the best line is < *m*, return no line

Else re-calculate the final parameters with all the inliers

Slide credit: Kristen Grauman

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.



Slide credit: David Lowe

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

Slide credit: Kristen Grauman

The hyperparameters

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

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

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

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
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

Slide credit: Kristen Grauman

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: 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: Pros and Cons

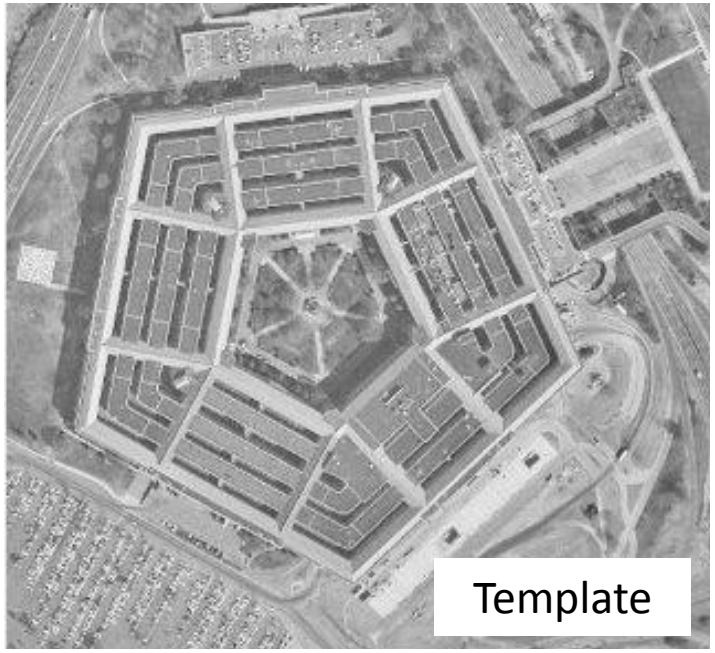
- **Pros**:
  - General method suited for a wide range of parameter fitting problems
  - Easy to implement and easy to calculate its failure rate
- **Cons**:
  - Only handles a moderate percentage of outliers without cost blowing up
  - Many real problems have high rate of outliers (but sometimes selective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

Today's agenda

- Canny edge detector
- Hough Transform
- RANSAC
- Local Invariant Features
- Harris Corner Detector

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

Image matching: a challenging problem

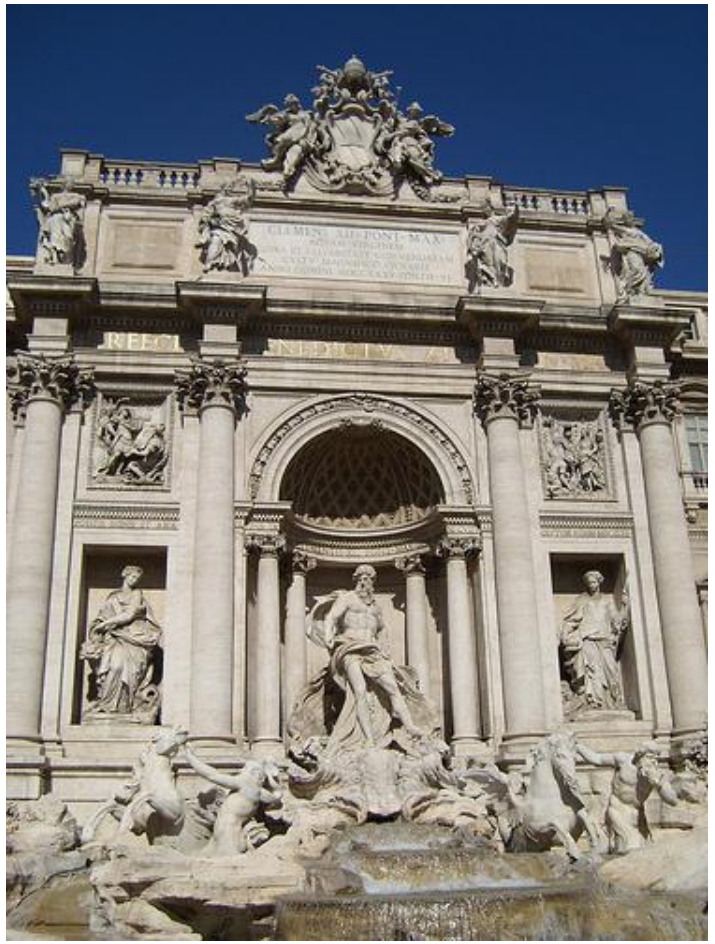


Q1. Will cross-correlation work?

Q2. Can we use match the lines?

Challenge: Perspective / viewpoint changes

by Diva Sian



by swashford

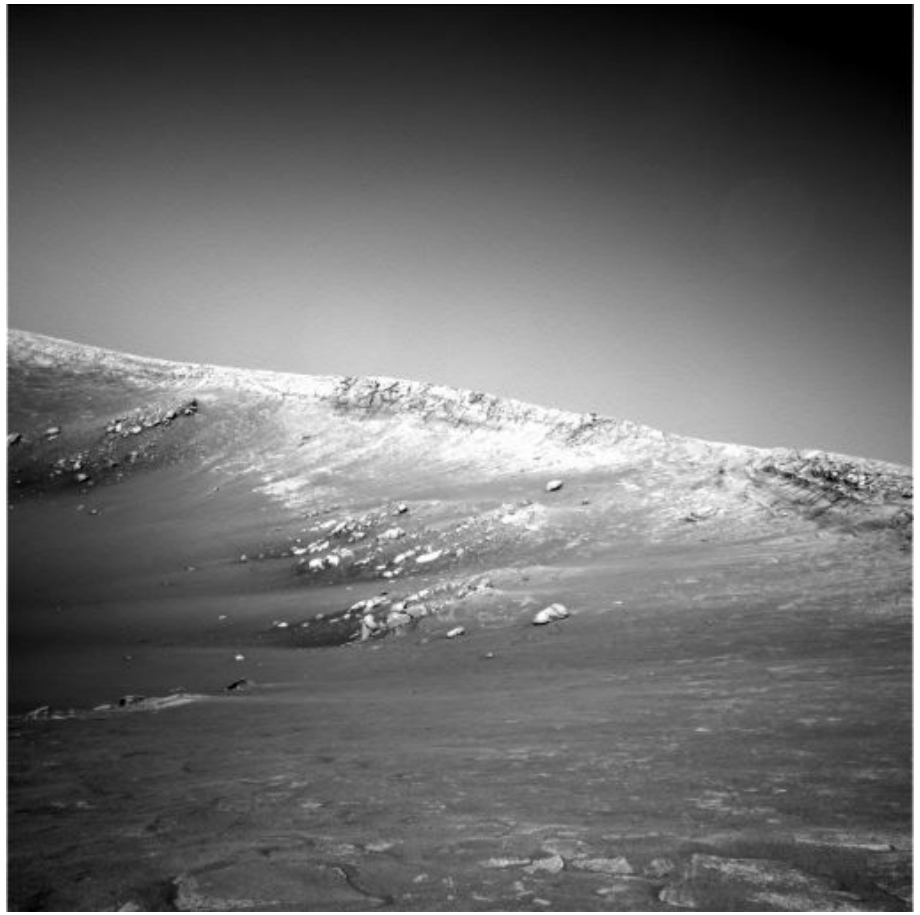
Challenge: partial observability

by Diva Sian

by scgbt

Slide Credit: Steve Seitz

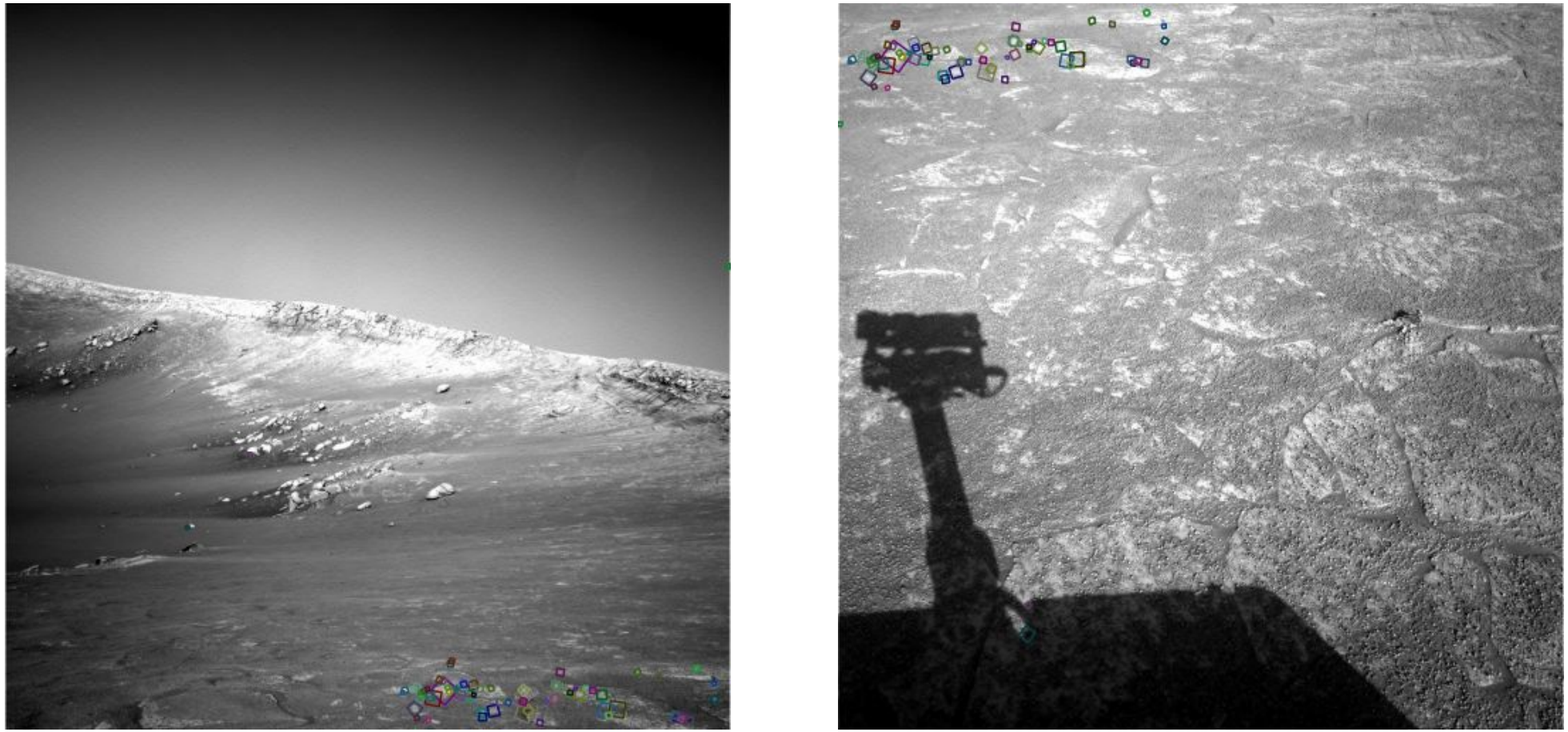
Challenge even for us



NASA Mars Rover images

Slide Credit: Steve Seitz

Answer Below (Look for tiny colored squares)



NASA Mars Rover images with SIFT feature matches
(Figure by Noah Snavely)

Slide Credit: Steve Seitz

Intuition behind how to match images

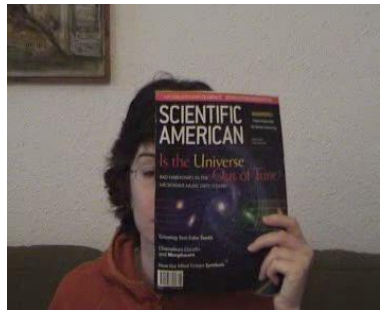
- Find matching patches
- Check to make sure enough patches

What do we need?

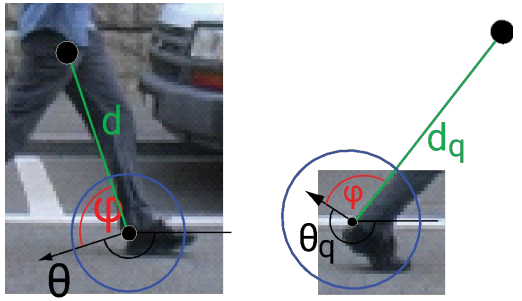
- We need to identify patches
- We need to learn to a way to describe each patch
- We need an algorithm to match the description between two patches

Motivation for using local features

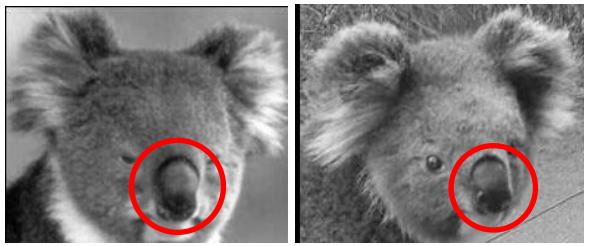
- Global representations have major limitations
- Instead, describe and match only local regions
- Increased robustness to
  - Occlusions



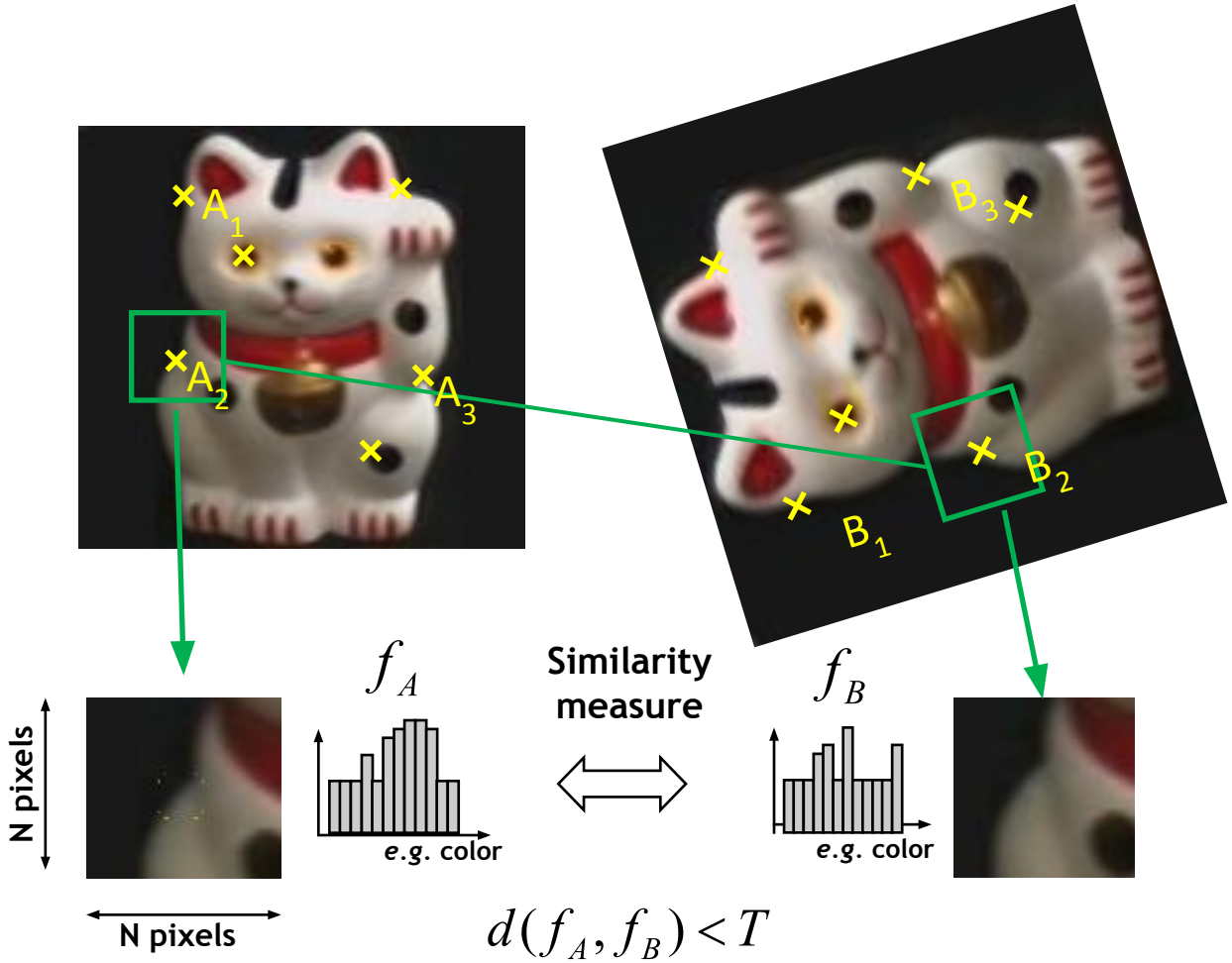
  - Articulation



  - Intra-category variations



General Approach



1. Find a set of distinctive key-points

2. Define a region/patch around each keypoint

3. Describe and normalize the region content

4. Compute a local descriptor from the normalized region

5. Match local descriptors

Common Requirements

- Problem 1: How should we choose the key-points?
  - We want to detect the same points independently in both images

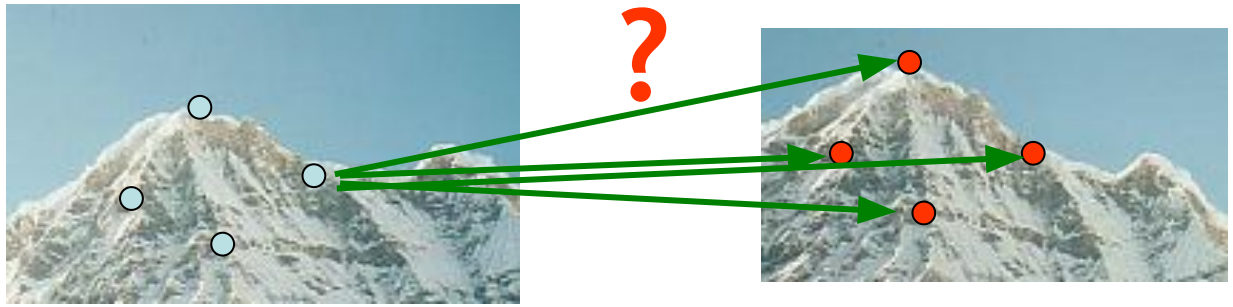
No chance to match if the key-points aren't the same

We need a repeatable detector!

Slide credit: Darya Frolova, Denis Simakov

Common Requirements

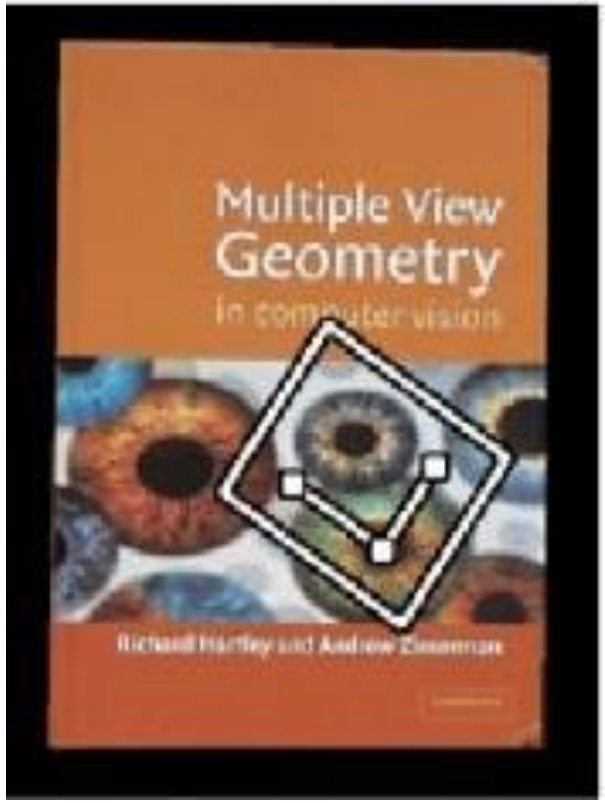
- Problem 1: How should we choose the key-points?
  - Detect the same point independently in both images
- Problem 2: How should we describe each patch?
  - For each point correctly recognize the corresponding one

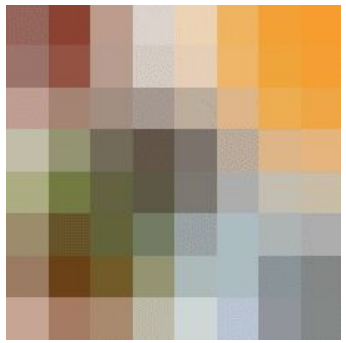


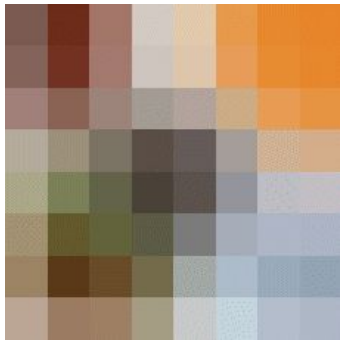
We need a reliable and distinctive descriptor!

Slide credit: Darya Frolova, Denis Simakov

Descriptions should be invariant to rotation and translation

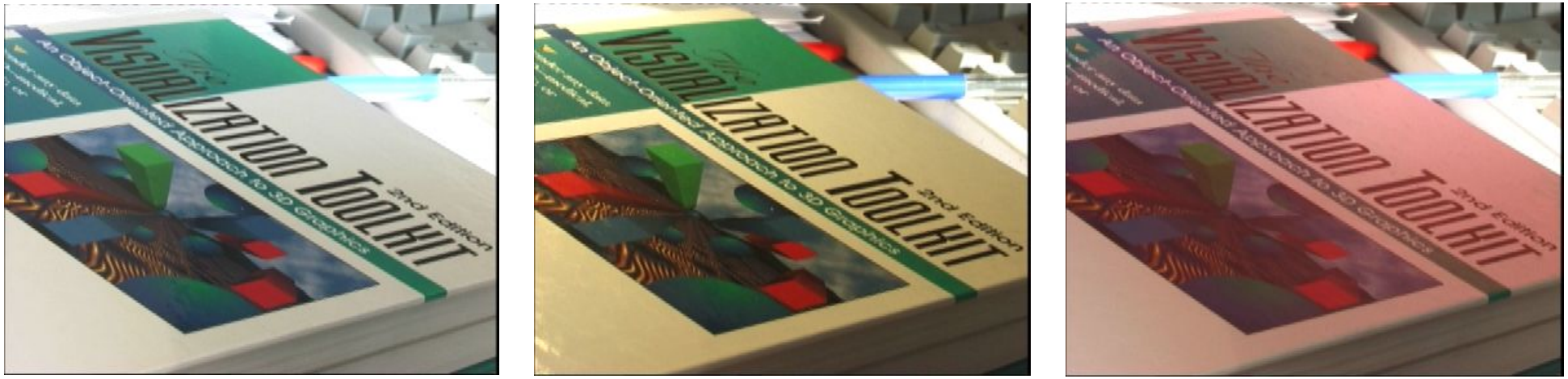






Slide credit: Steve Seitz

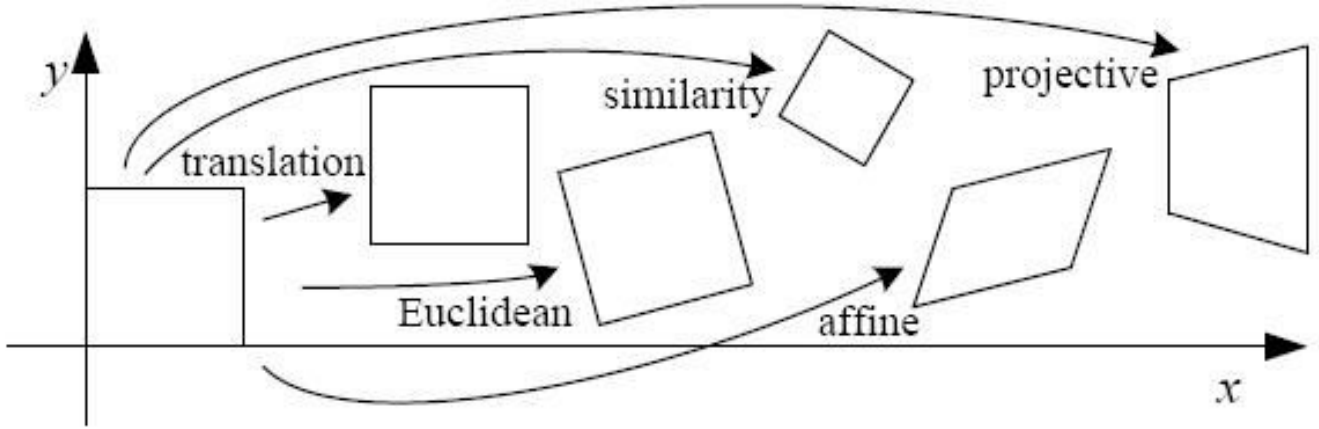
Descriptions should be invariant to photometric transformations

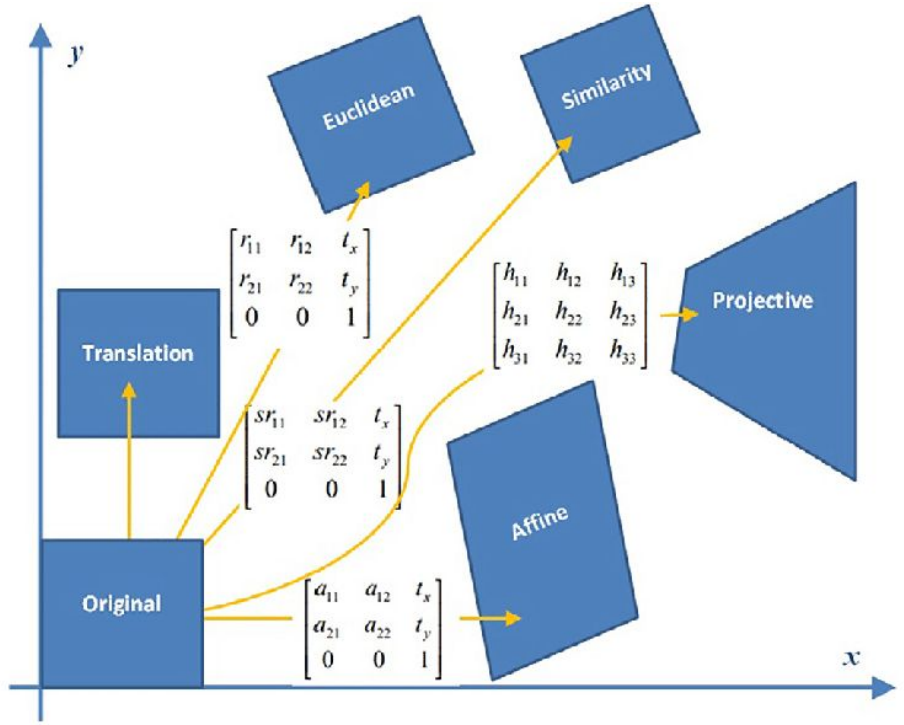


- Often modeled as a linear transformation:
  - Scaling + Offset

Slide credit: Tinne Tuytelaars

Levels of geometric transformations





Slide credit: Bastian Leibe

Requirements for Local Features

- Patch selection needs to be **repeatable** and **accurate**
  - **Invariant** to translation, rotation, scale changes
  - **Robust** to out-of-plane (≈affine) transformations
  - **Robust** to lighting variations, noise, blur, quantization
- **Locality**: Features are local, therefore robust to occlusion and clutter.
- **Quantity**: We need a sufficient number of regions to cover the object.
- **Distinctiveness**: The regions should contain “interesting” structure.
- **Efficiency**: Close to real-time performance.

Slide credit: Bastian Leibe

Many existing feature detectors available

- Hessian & Harris [Beaudet ‘78], [Harris ‘88]
- Laplacian, DoG [Lindeberg ‘98], [Lowe ‘99]
- Harris-/Hessian-Laplace [Mikolajczyk & Schmid ‘01]
- Harris-/Hessian-Affine [Mikolajczyk & Schmid ‘04]
- EBR and IBR [Tuytelaars & Van Gool ‘04]
- MSER [Matas ‘02]
- Salient Regions [Kadir & Brady ‘01]
- Neural networks [Krichevsky ‘12]

- *Those detectors have become a basic building block for many applications in Computer Vision.*

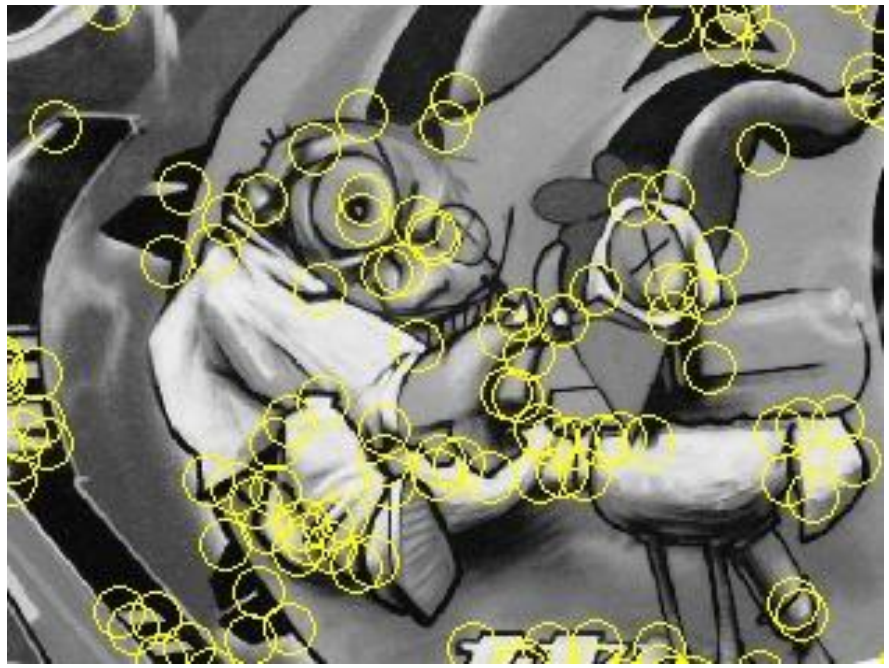
Slide credit: Bastian Leibe

Today's agenda

- Canny edge detector
- Hough Transform
- RANSAC
- Local Invariant Features
- Harris Corner Detector

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

Keypoint Localization



- **Goals:**
  - Repeatable detection
  - Precise localization
  - Interesting content

intuition⇒ *Look for 2D signal changes (*LSI systems strike again*)*

Slide credit: Bastian Leibe

What are good patches?

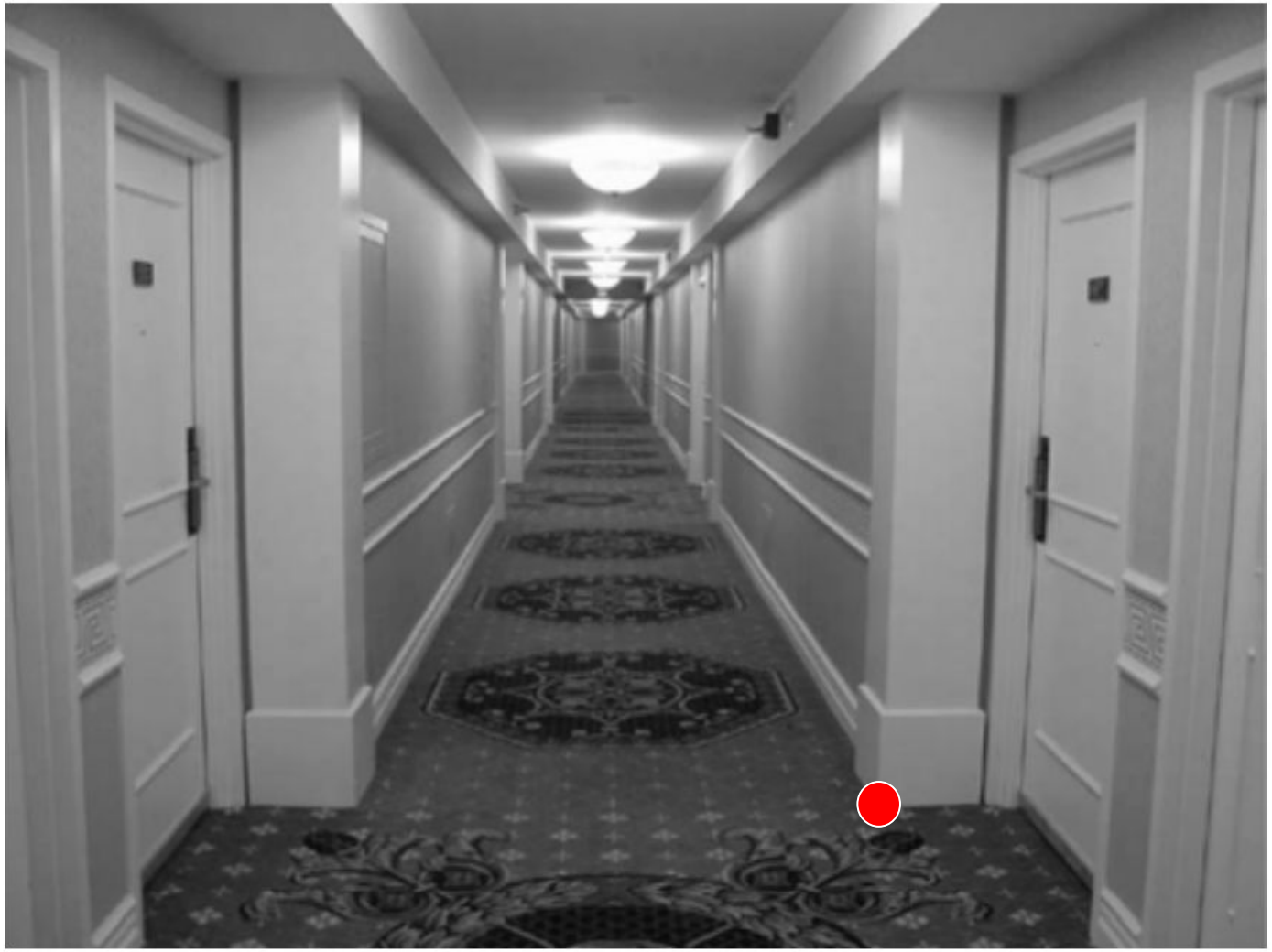
Q. Is this a good patch for image matching?

What are good patches?

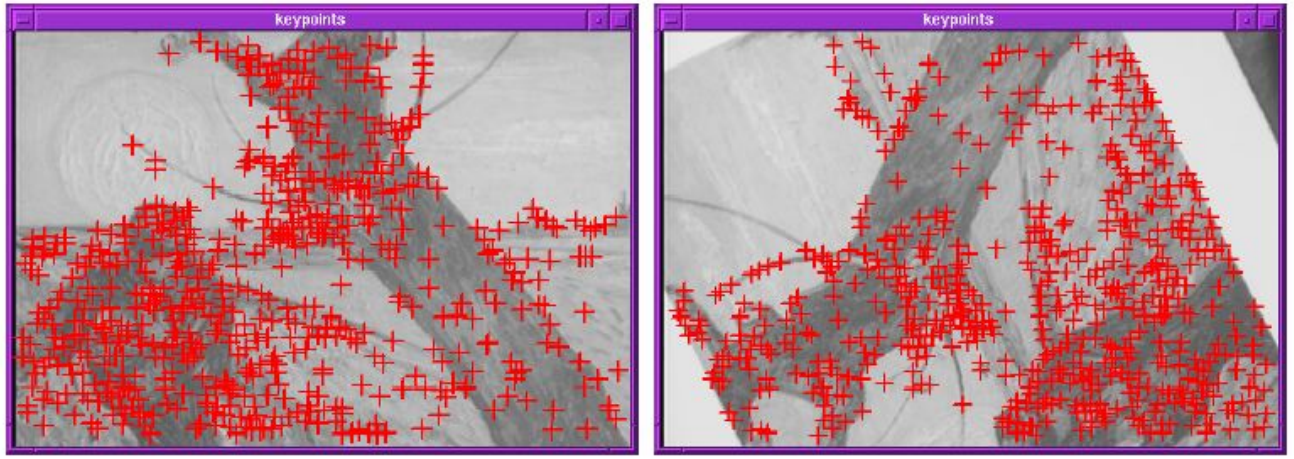
Q. What about this one?

What are good patches?

Q. Let's try another one?



Finding Corners



How do we find corners using LSI systems?

- The image gradient around a corner has two or more dominant directions

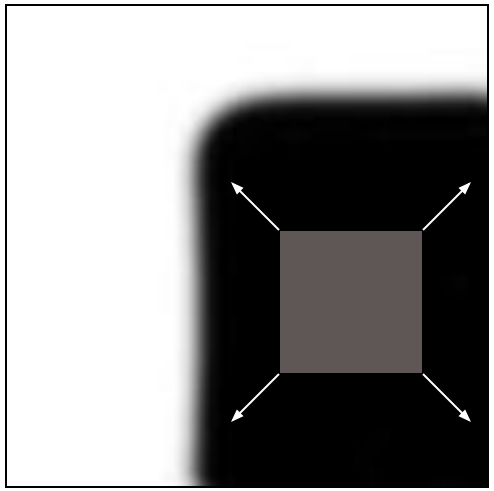
Corners are **repeatable** and **distinctive**

C.Harris and M.Stephens. "A Combined Corner and Edge Detector."
Proceedings of the 4th Alvey Vision Conference, 1988.

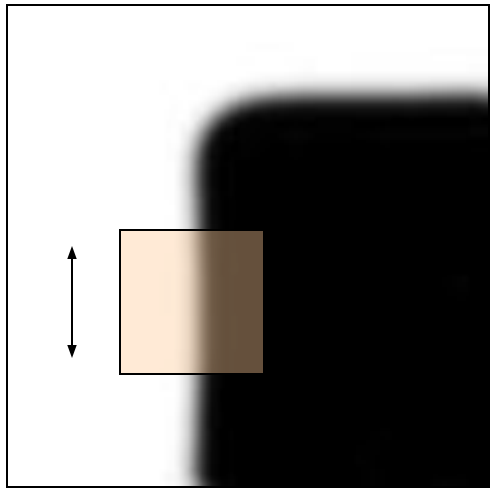
Slide credit: Svetlana Lazebnik

Corners are distinctive key-points

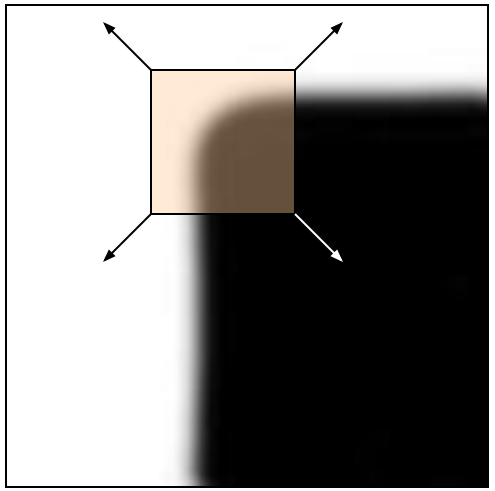
- We should easily recognize the corner point by looking through a small image patch (*locality*)
- Shifting the window in *any direction* should give *a large change* in intensity (*good localization*)



"flat" region: no change in all directions



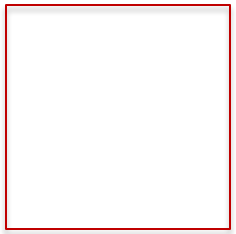
"edge": no change along the edge direction



"corner": significant change in all directions

Slide credit: Alyosha Efros

Flat patches have small image gradients

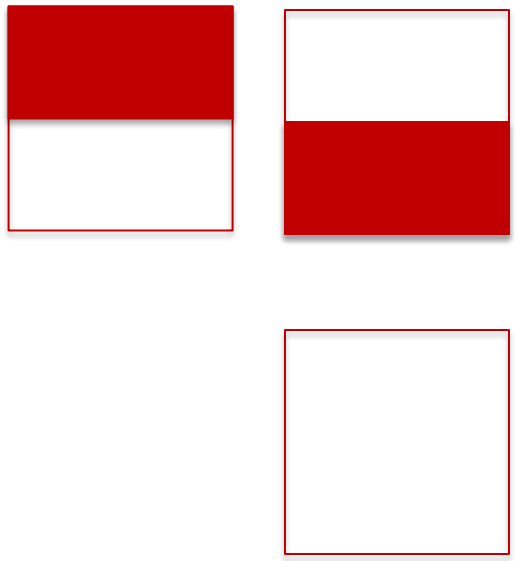


$\sum I_x^2$ → Small

$\sum I_y^2$ → Small

Flat

Edges have high gradient in one direction



$\sum I_x^2$ → Small

$\sum I_y^2$ → Large

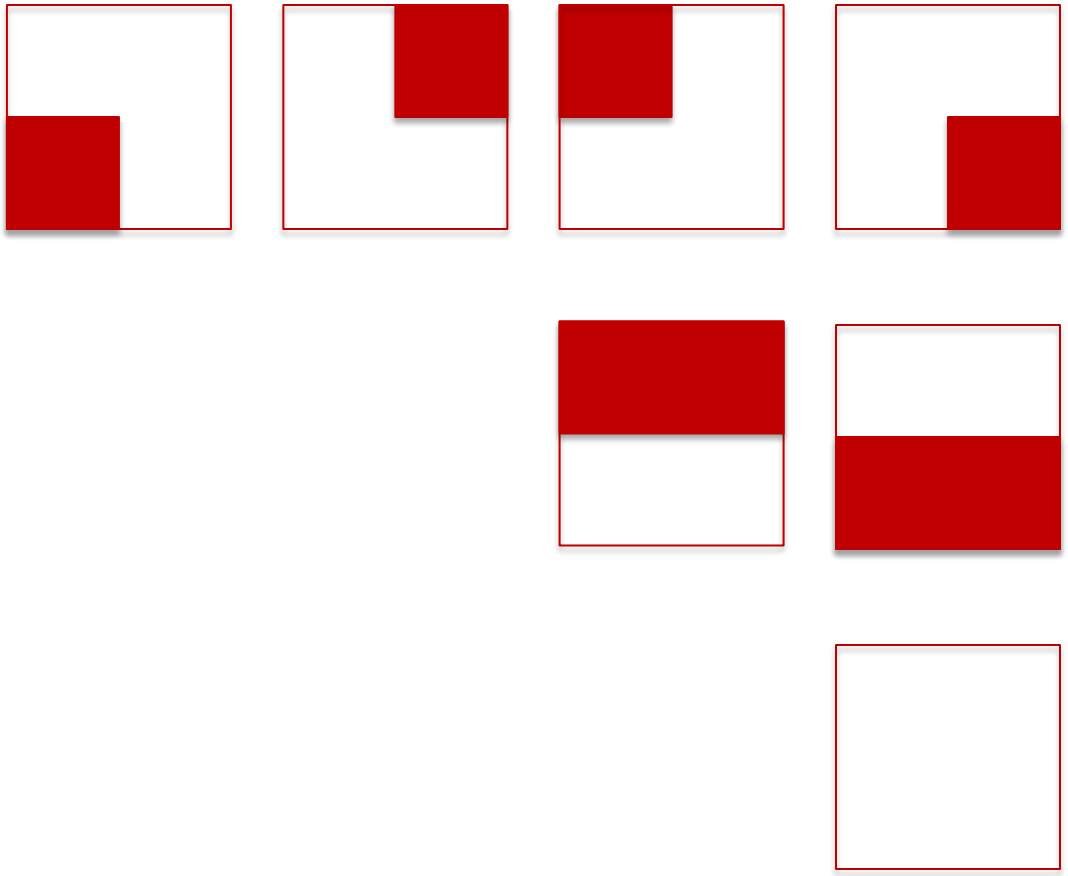
Edge

$\sum I_x^2$ → Small

$\sum I_y^2$ → Small

Flat

Corners versus edges



$\sum I_x^2$ → Large

$\sum I_y^2$ → Large

Corner

$\sum I_x^2$ → Small

$\sum I_y^2$ → Large

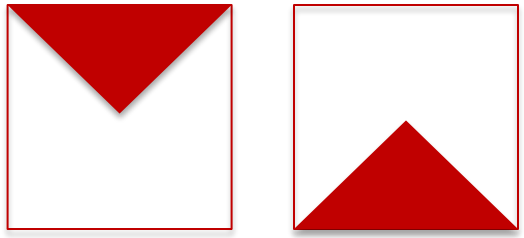
Edge

$\sum I_x^2$ → Small

$\sum I_y^2$ → Small

Flat

Generalizing to corners in any direction



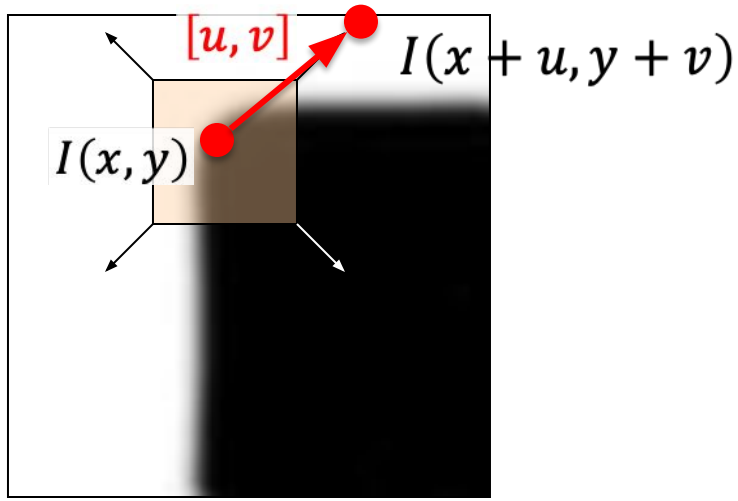
$$\sum I_x^2 \rightarrow ??$$

$$\sum I_y^2 \rightarrow ??$$

Corner

Harris Detector Formulation

- Find patches that result in large change of pixel values when shifted in *any direction.*
- When we shift by $[u, v]$, the intensity change at the center pixel is:



"corner":
significant change in all directions

- Measure change as intensity difference:

$$(I(x+u, y+v) - I(x, y))$$

- That's for a single point, but we have to accumulate over the patch or "small window" around that point...

Harris Detector Formulation

- When we shift by $[u, v]$, the change in intensity for the "small window" is:

$$E(u,v) = \sum_{x,y} w(x,y)\,[I(x+u, y+v) - I(x,y)]^2$$

Sum over window

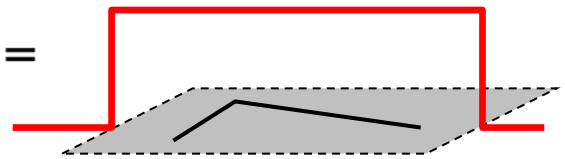
Window function

Shifted intensity

Intensity

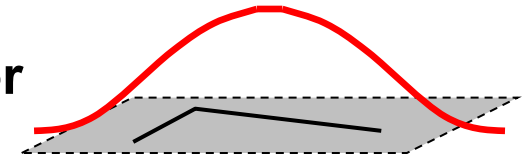
Intensity change

Window function $w(x,y) =$



1 in window, 0 outside

or



Gaussian

Slide credit: Rick Szeliski

Change in intensity function

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$$

We can rewrite the shifted intensity using Taylor's expansion:

$$I(x+u, y+v) \approx I(x,y) + I_x u + I_y v$$

Substituting it back into E(u, v):

$$E(u,v) = \sum_{x,y} w(x,y)[I_x u + I_y v]^2$$

Re-writing E:

$$E(u,v) = \sum_{x,y} w(x,y)[I_x u + I_y v]^2$$

Re-writing E:

$$E(u,v) = \sum_{x,y} w(x,y)[I_x u + I_y v]^2$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} I_x u & I_y v \end{bmatrix} \begin{bmatrix} I_x u \\ I_y v \end{bmatrix}$$

Re-writing E:

$$E(u,v) = \sum_{x,y} w(x,y)[I_x u + I_y v]^2$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} I_x u & I_y v \end{bmatrix} \begin{bmatrix} I_x u \\ I_y v \end{bmatrix}$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x u \\ I_y v \end{bmatrix}$$

Re-writing E:

$$E(u,v) = \sum_{x,y} w(x,y)[I_x u + I_y v]^2$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} I_x u & I_y v \end{bmatrix} \begin{bmatrix} I_x u \\ I_y v \end{bmatrix}$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x u \\ I_y v \end{bmatrix}$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Re-writing E:

$$E(u,v) = \sum_{x,y} w(x,y)[I_x u + I_y v]^2$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} I_x u & I_y v \end{bmatrix} \begin{bmatrix} I_x u \\ I_y v \end{bmatrix}$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x u \\ I_y v \end{bmatrix}$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Re-writing E:

$$
\begin{aligned}
E(u,v) &= \sum_{x,y} w(x,y)[I_x u + I_y v]^2 \\
&= \sum_{x,y} w(x,y) \begin{bmatrix} I_x u & I_y v \end{bmatrix} \begin{bmatrix} I_x u \\ I_y v \end{bmatrix} \\
&= \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x u \\ I_y v \end{bmatrix} \\
&= \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\
&= \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\
&= w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}
\end{aligned}
$$

Re-writing E:

$$E(u,v) = w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Re-writing E:

$$E(u,v) = w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$= w(x,y) \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix}$$

Re-writing E:

$$E(u,v) = w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$= w(x,y) \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

Does anyone know what this part of the equation is?

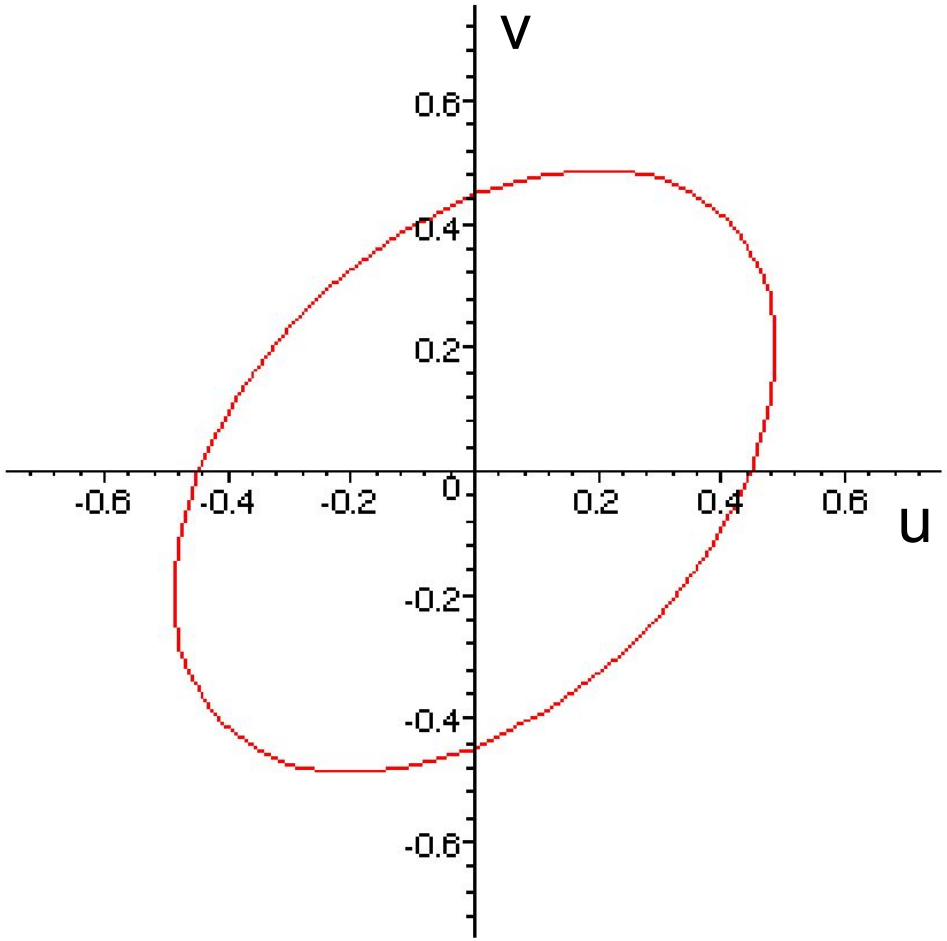
$$M = \begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix}$$

It’s the equation of an ellipse

$$5u^2 - 4uv + 5v^2 = 1$$

$$\begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \begin{bmatrix} 5 & -2 \\ -2 & 5 \end{bmatrix}$$



Change in intensity in a patch

- So, using Taylor's expansion, the change in intensity in an image patch:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a 2×2 matrix computed from image derivatives:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Gradient with respect to x, times gradient with respect to y

Sum over image region – the area we are checking for corner

Harris Detector Formulation

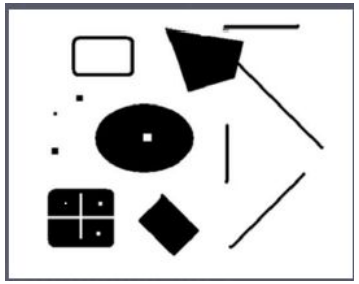
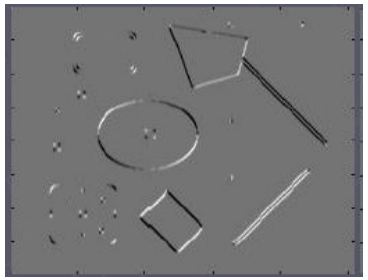


Image I

I_x

I_y



$I_x I_y$

where M is a 2×2 matrix computed from image derivatives:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

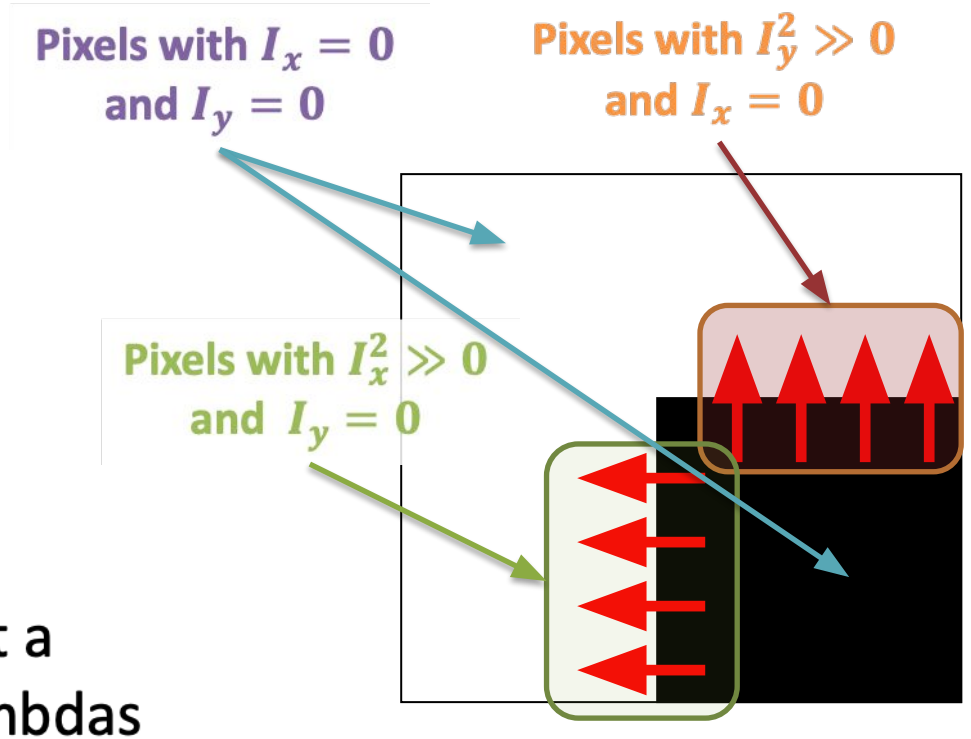
Gradient with respect to x, times gradient with respect to y

Sum over image region – the area we are checking for corner

$$M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

What Does This Matrix Reveal?

- First, let's consider an axis-aligned corner.
- In that case, the dominant gradient directions align with the x or the y axis
- $M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$



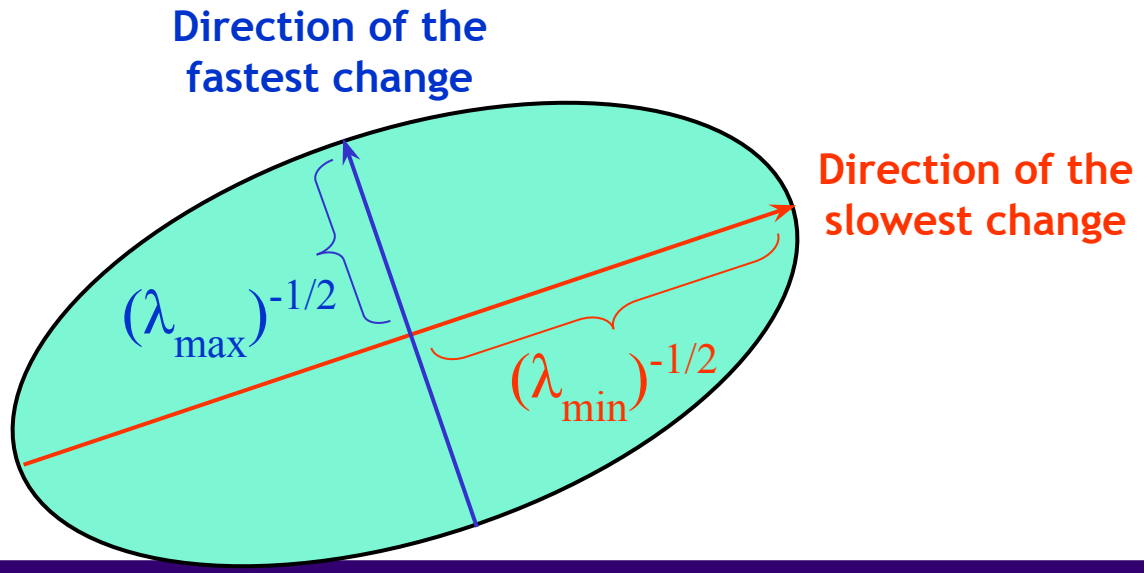
- This means: if either λ is close to 0, then this is not a corner, so look for image windows where both lambdas are large.
- What if we have a corner that is not aligned with the image axes?

General Case

- Since $M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$ is symmetric, we can re-rewrite $M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$

(Eigenvalue decomposition)

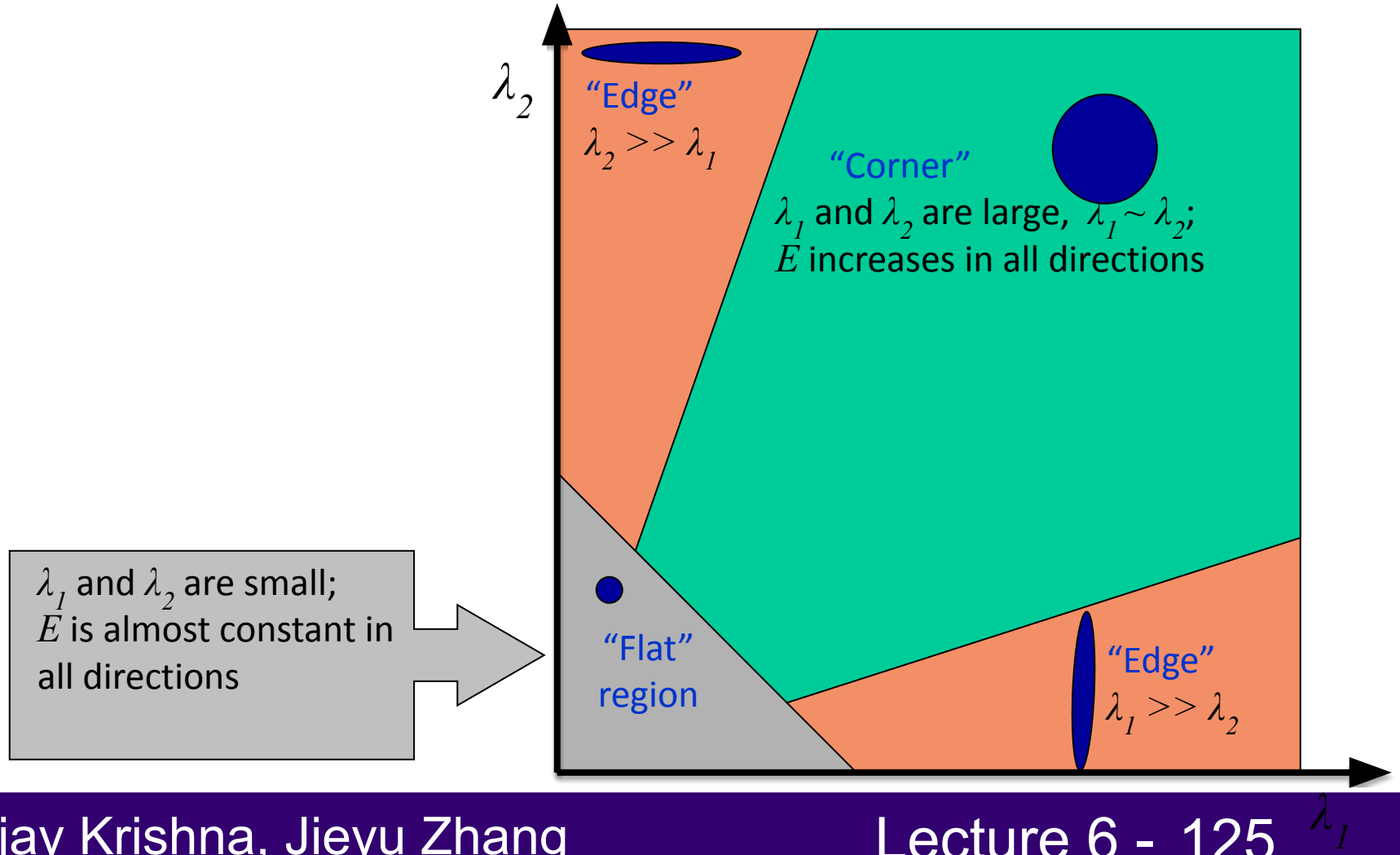
- We can think of M as an ellipse with its axis lengths determined by the eigenvalues λ_1 and λ_2; and its orientation determined by R



- A rotated corner would produce the same eigenvalues as its non-rotated version.

Interpreting the Eigenvalues

- Classification of image points using eigenvalues of M:

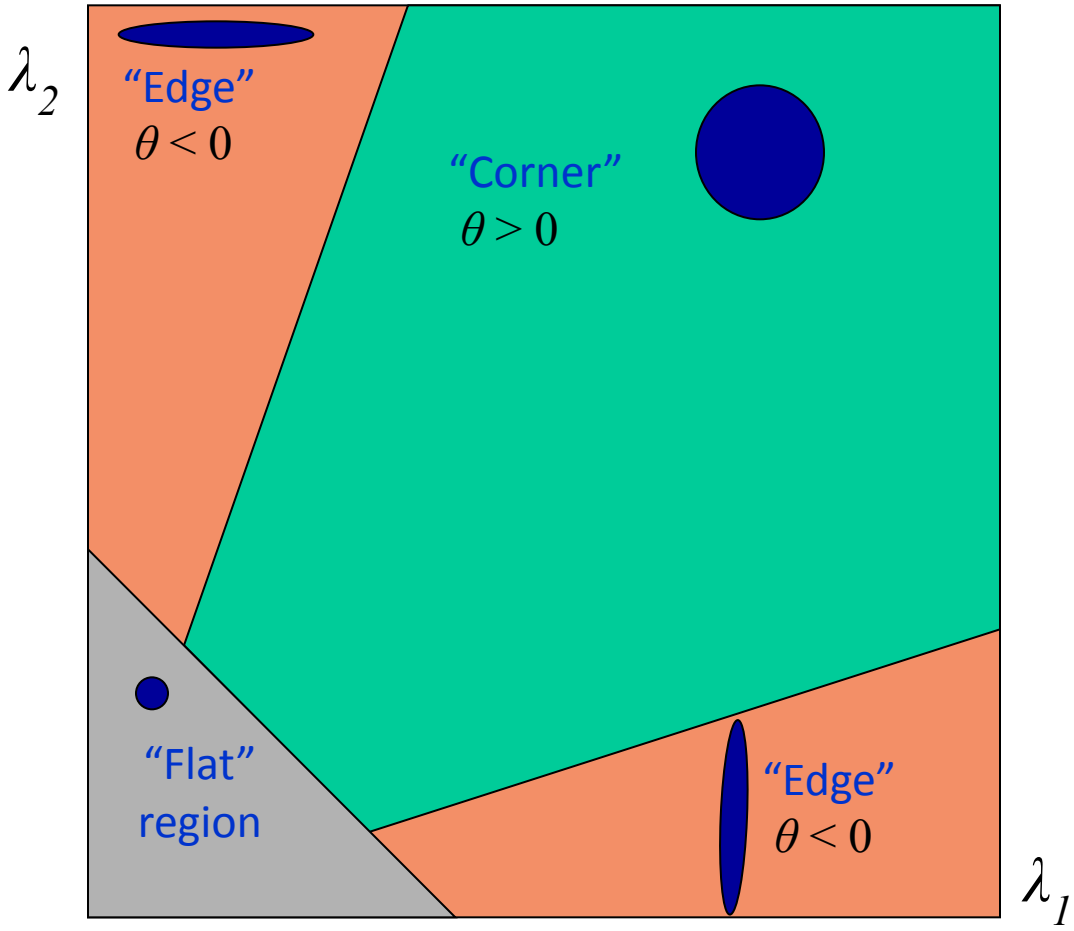


Slide credit: Kristen Grauman

Corner Response Function

$$\theta = \det(M) - \alpha \,\mathrm{trace}(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

- Fast approximation
  - Avoid computing the eigenvalues
  - α: constant (0.04 to 0.06)



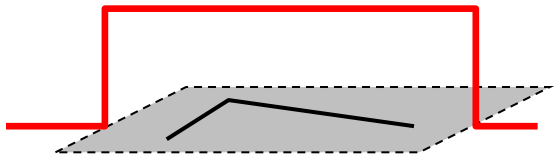
Slide credit: Kristen Grauman

Window Function $w(x,y)$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- Option 1: uniform window
  - Sum over square window

$$M = \sum_{x,y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

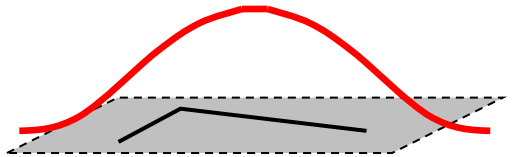


1 in window, 0 outside

  - Problem: not rotation invariant

- Option 2: Smooth with Gaussian
  - Gaussian already performs weighted sum

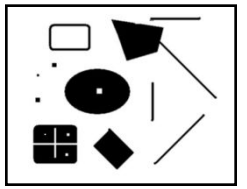
$$M = g(\sigma) * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



Gaussian

  - Result is rotation invariant

Summary: Harris Detector [Harris88]

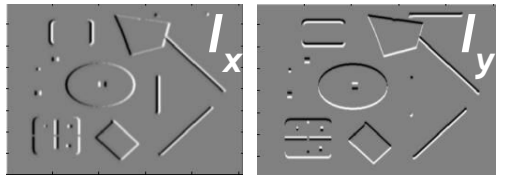


- Compute second moment matrix (autocorrelation matrix)

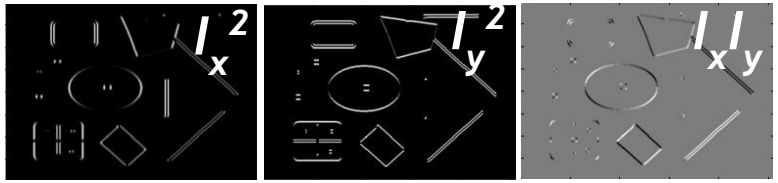
$$M(\sigma_I,\sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

σ_D: for Gaussian in the derivative calculation
σ_I: for Gaussian in the windowing function

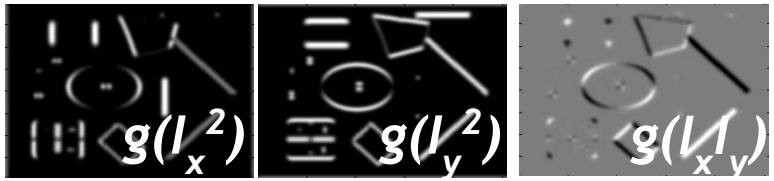
1. Image derivatives



2. Square of derivatives



3. Gaussian filter $g(\sigma_I)$



4. Cornerness function - two strong eigenvalues

$$\theta = \det[M(\sigma_I,\sigma_D)] - \alpha[\text{trace}(M(\sigma_I,\sigma_D))]^2$$
$$= g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$$

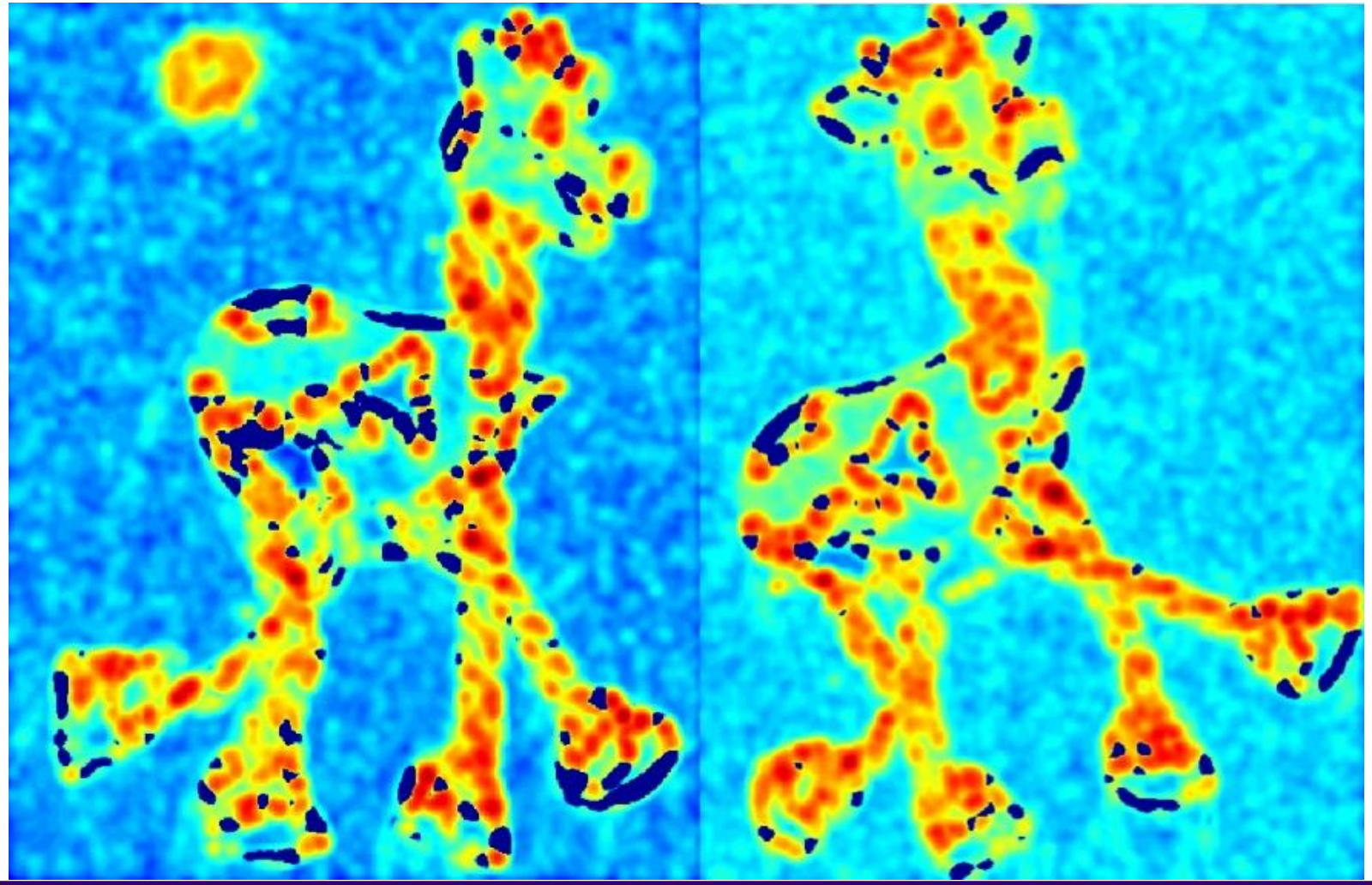
5. Perform non-maximum suppression

Harris Detector: Example

- Input Image

Harris Detector: Example

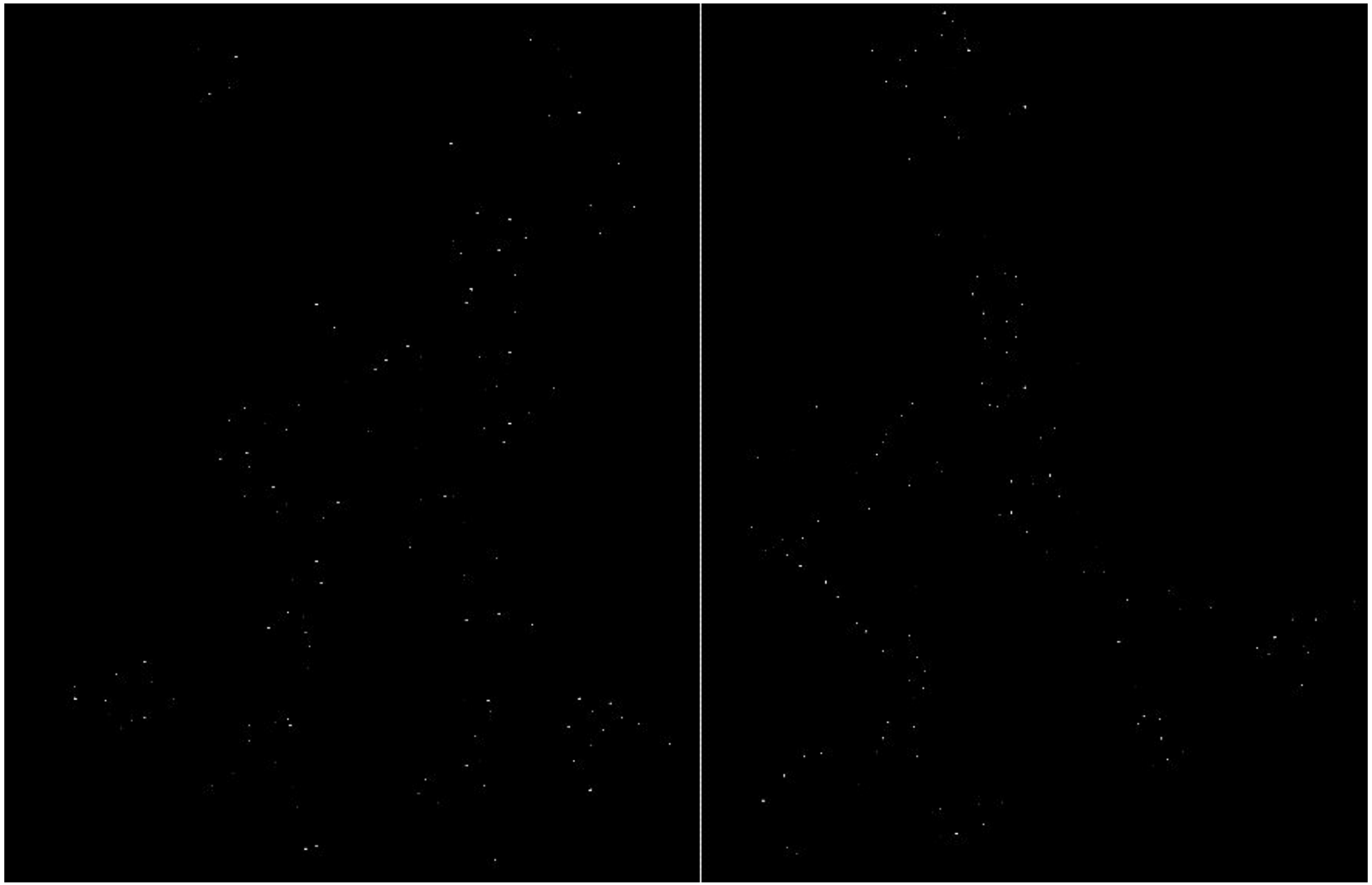
- Input Image
- Compute corner response function θ



Slide adapted from Daya Frolova, Denis Simakov

Harris Detector: Example

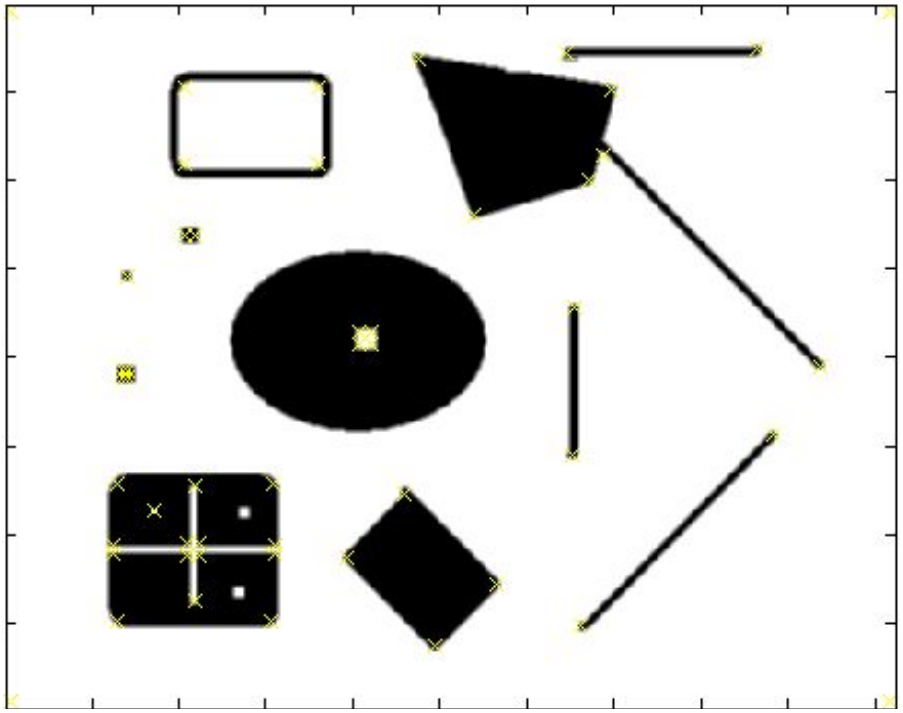
- Input Image
- Compute corner response function θ
- Take only the local maxima of θ, where θ > threshold



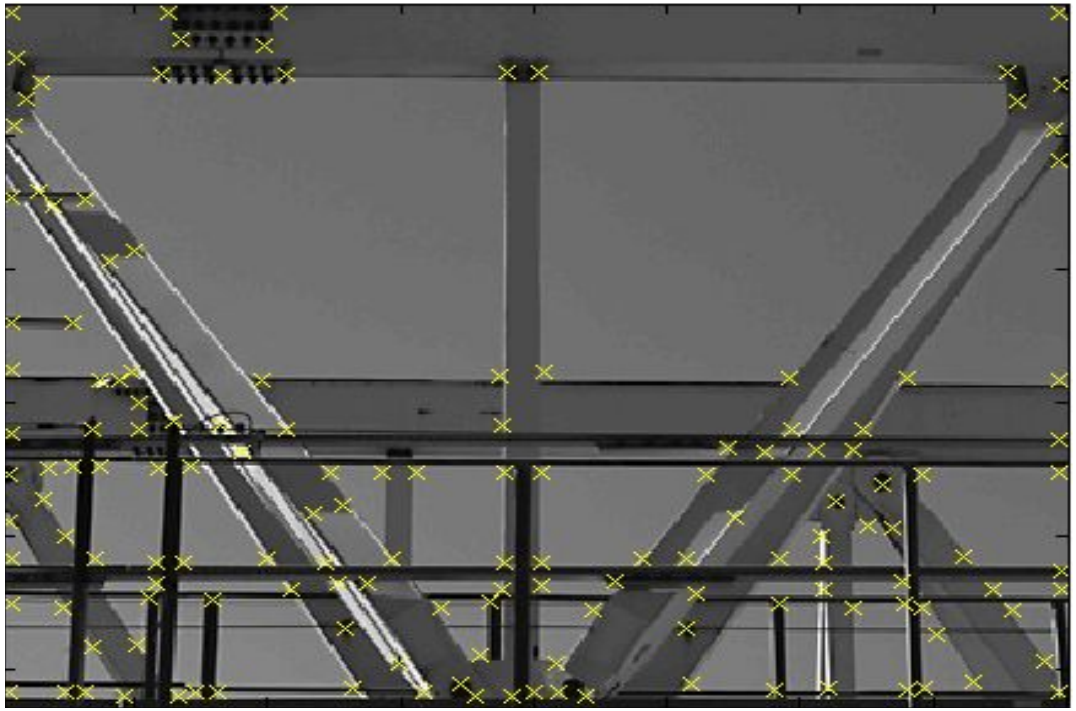
Harris Detector: Example

- Input Image
- Compute corner response function θ
- Take only the local maxima of θ, where θ > threshold

Harris Detector – Responses [Harris88]



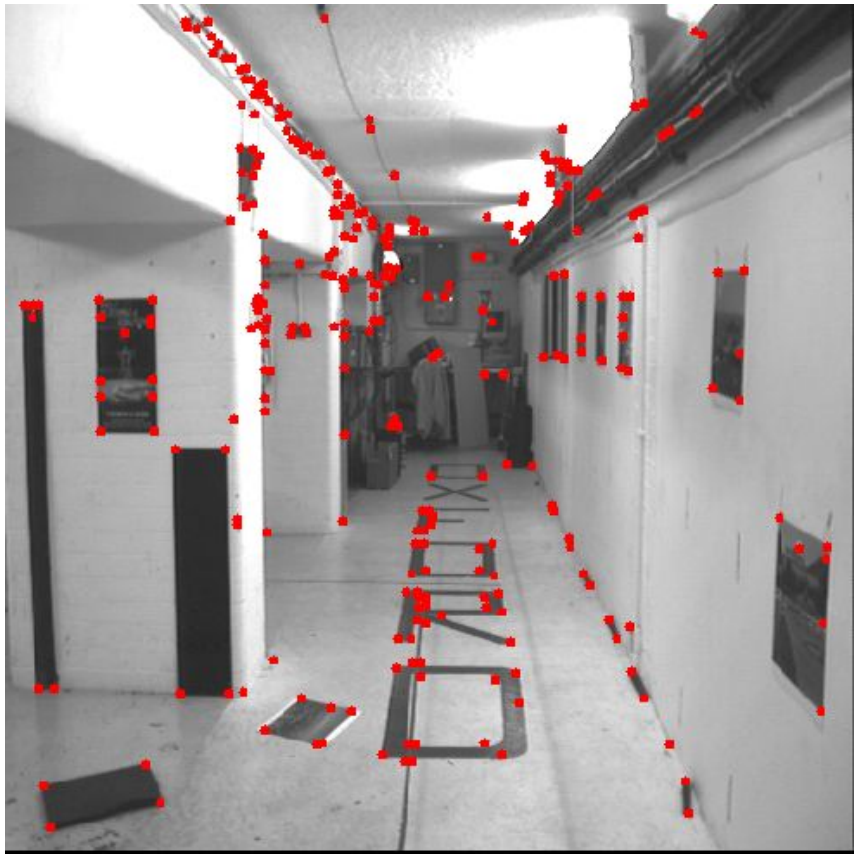
Effect: A very precise corner detector.



Harris Detector – Responses [Harris88]

Slide credit: Krystian Mikolajczyk

Harris Detector – Responses [Harris88]



- Results are great for finding correspondences matches between images

Slide credit: Kristen Grauman

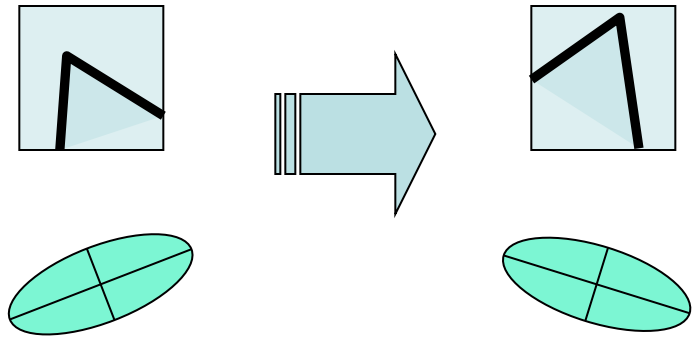
Harris Detector: Properties

- Translation invariance?

Slide credit: Kristen Grauman

Harris Detector: Properties

- Translation invariance
- Rotation invariance?

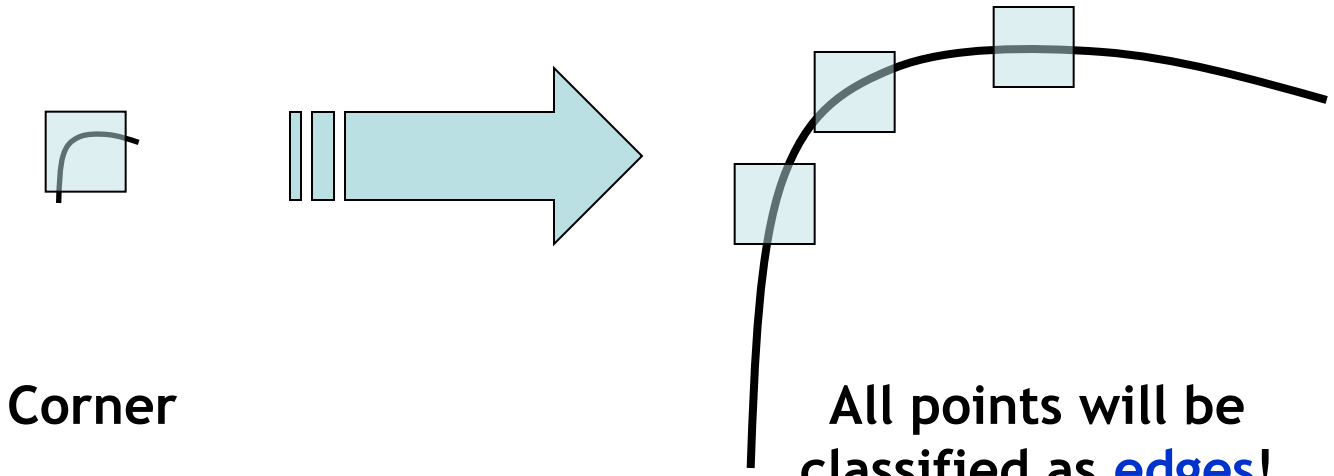


Ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner response θ **is invariant to image rotation**

Harris Detector: Properties

- Translation invariance
- Rotation invariance
- Scale invariance?



***Not* invariant to image scale!**

Summary

- Canny edge detector
- Hough Transform
- Model fitting for line detection: RANSAC
- Local Invariant Features
- Harris Corner Detector

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

Next time

Detectors and descriptors