CS 1674: Grouping: edges, lines, circles, and segments

PhD. Nils Murrugarra-Llerena
nem177@pitt.edu

University of Pittsburgh

Plan for this lecture

- Edges
  - Extract gradients and threshold
- Lines and circles
  - Find which edge points are collinear or belong to another shape e.g. circle
  - Automatically detect and ignore outliers
- Segments
  - Find which pixels form a consistent region
  - Clustering (e.g. K-means)

Edge detection

- **Goal**: map image from 2d array of pixels to a set of curves or line segments or contours.
- **Why?**

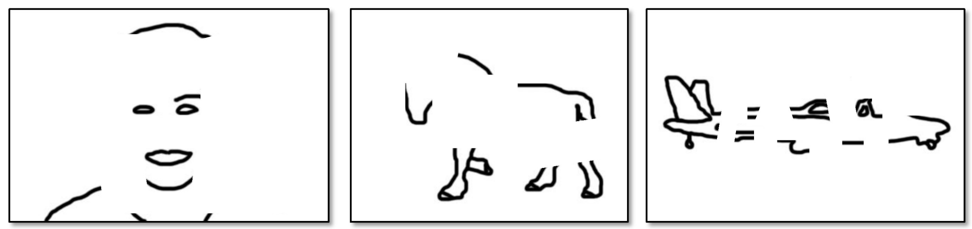


Figure from J. Shotton et al., PAMI 2007

- **Main idea**: look for strong gradients, post-process

Source: K. Grauman

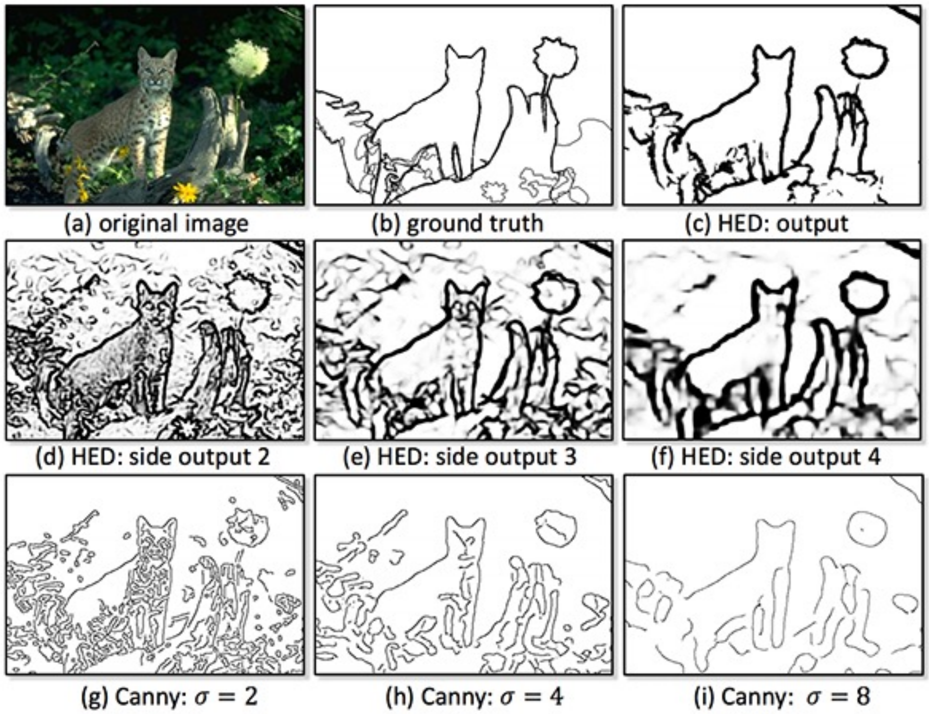
Designing an edge detector

- Criteria for a good edge detector
  - Good categorization (edge vs not edge)
    - find all real edges, ignoring noise or other artifacts
  - Good localization
    - detect edges as close as possible to the true edges
    - return one point only for each true edge point (true edges = the edges humans drew on an image)

- Cues of edge detection
  - Bottom-up: Differences in color, intensity, or texture across the boundary
  - Top-down: Continuity and closure, high-level knowledge

Adapted from L. Fei-Fei

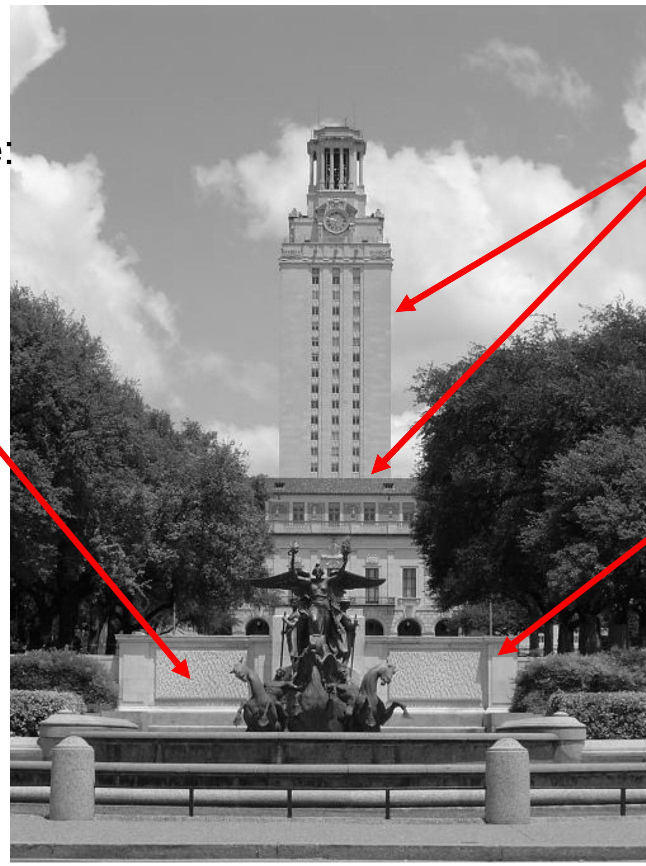
Examples of edge detection results



Xie and Tu, [Holistically-Nested Edge Detection](), ICCV 2015

What causes an edge?

Reflectance change:
appearance
information, texture



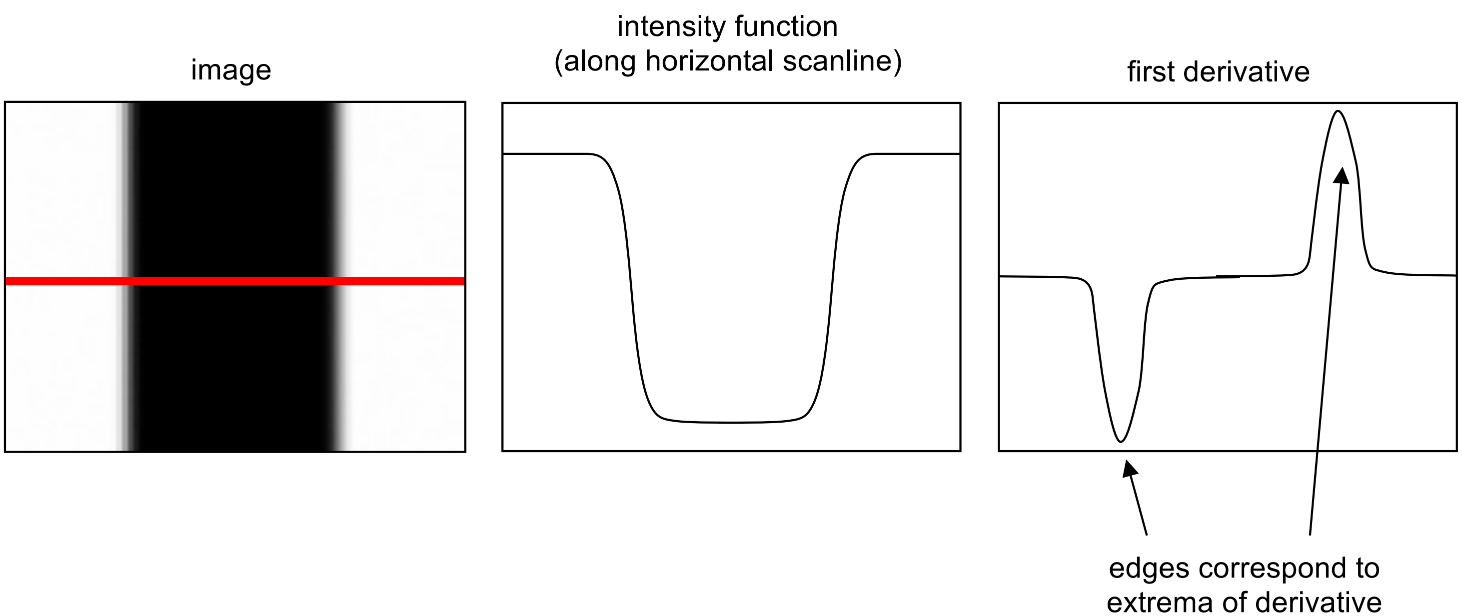
Depth
discontinuity:
object boundary

Cast shadows

Adapted from K. Grauman

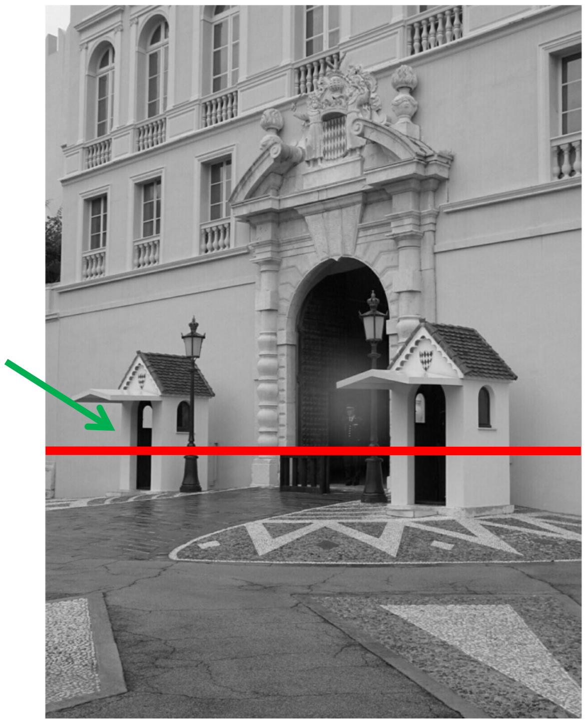
Characterizing edges

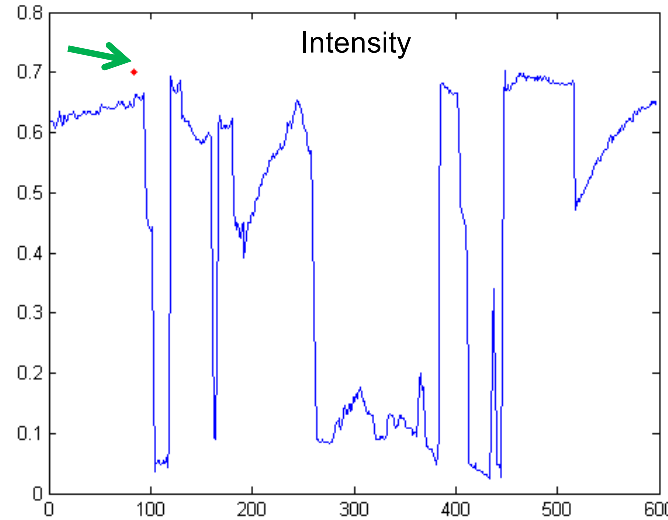
- An edge is a place of rapid change in the image intensity function

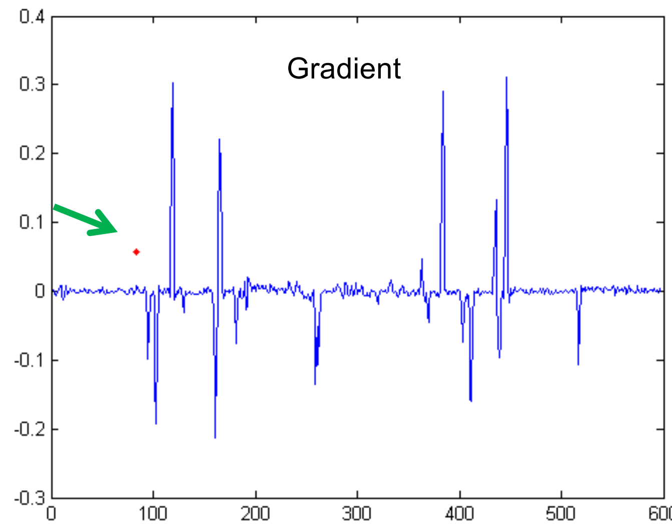


Source: L. Lazebnik

Intensity profile







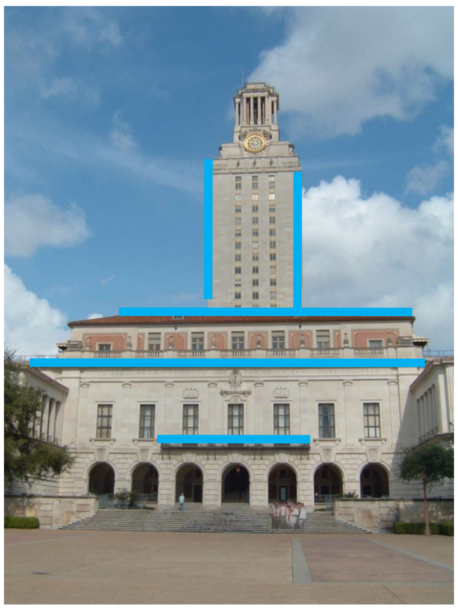
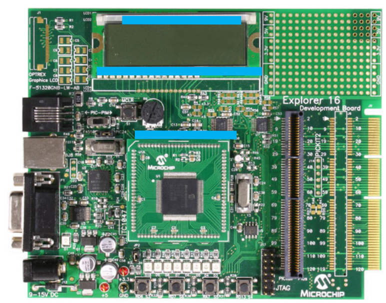
Source: D. Hoiem

Plan for this lecture

- Edges
  - Extract gradients and threshold
- Lines and circles
  - Find which edge points are collinear or belong to another shape e.g. circle
  - Automatically detect and ignore outliers
- Segments
  - Find which pixels form a consistent region
  - Clustering (e.g. K-means)

Line detection (fitting)

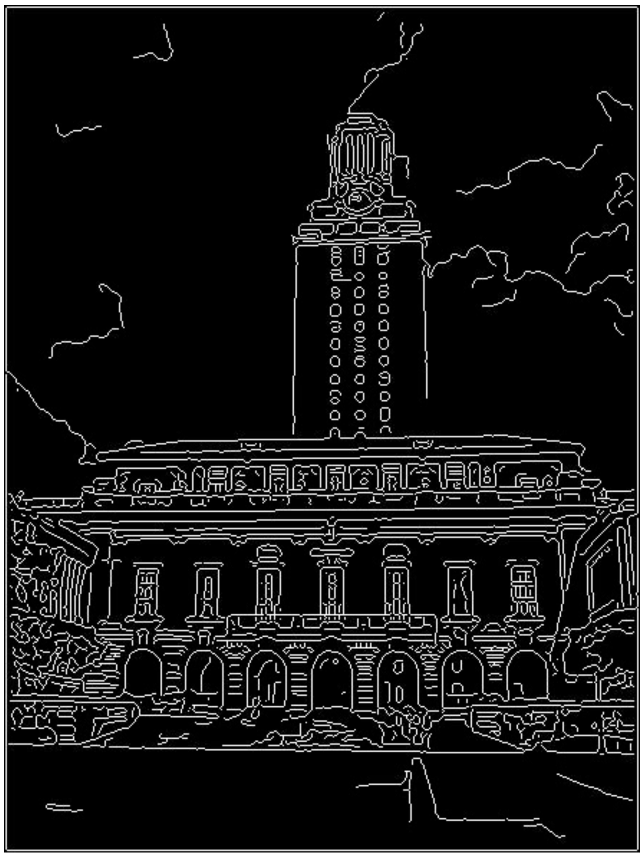
- Why fit lines?
  Many objects characterized by presence of straight lines

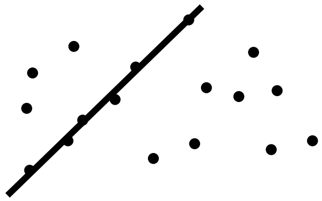

- Why aren't we done just by running edge detection?

Kristen Grauman

Difficulty of Line Fitting



Adapted from Kristen Grauman

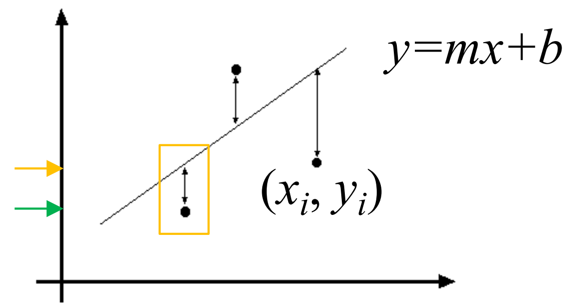


- **Noise** in measured edge points, orientations:
  - e.g. edges not collinear where they should be
  - how to detect true underlying parameters?
- **Extra** edge points (clutter):
  - which points go with which line, if any?
- Only some parts of each line detected, and some parts are **missing:**
  - how to find a line that bridges missing evidence?

Least squares line fitting

- Data: $(x_1, y_1), \ldots, (x_n, y_n)$
- Line equation: $y_i = m\,x_i + b$
- Find (m, b) to minimize

$$\boxed{E = \sum_{i=1}^{n} (\underbrace{mx_i + b}_{\text{where line you found tells you point is along y axis}} - \underbrace{y_i}_{\text{where point really is along y axis}})^2}$$



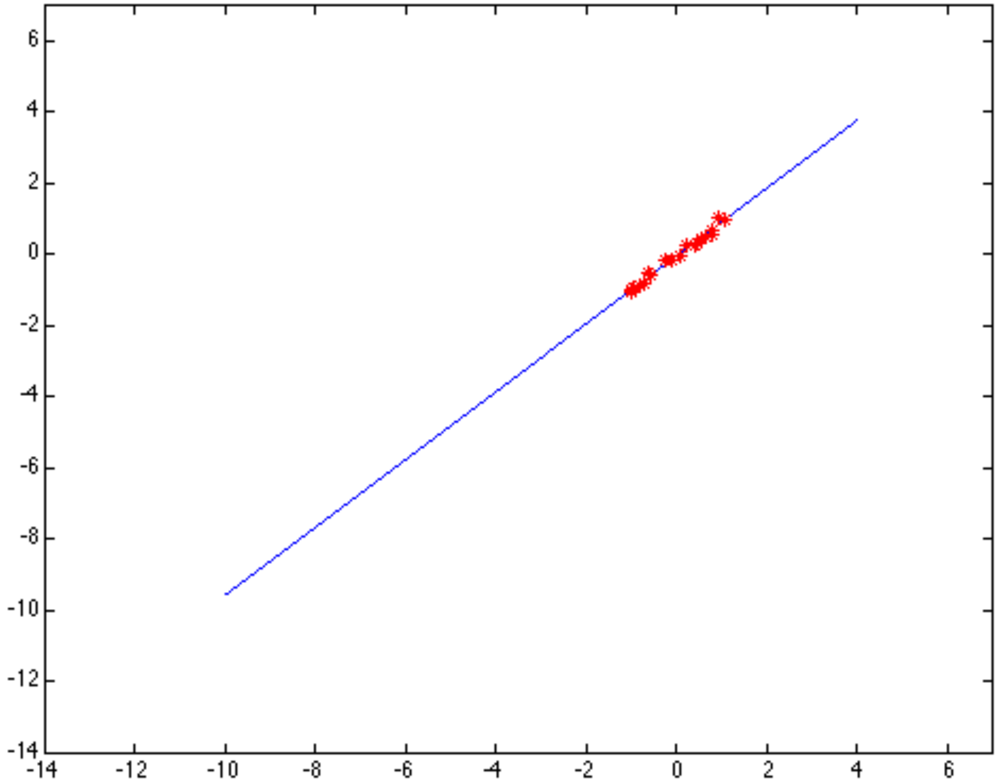
You want to find a single line that "explains" all of the points in your data, but data may be noisy!

$$E = \sum_{i=1}^{n} \left(\begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - y_i \right)^2 = \left\| \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 = \|\mathbf{Ap} - \mathbf{y}\|^2$$

Adapted from Svetlana Lazebnik

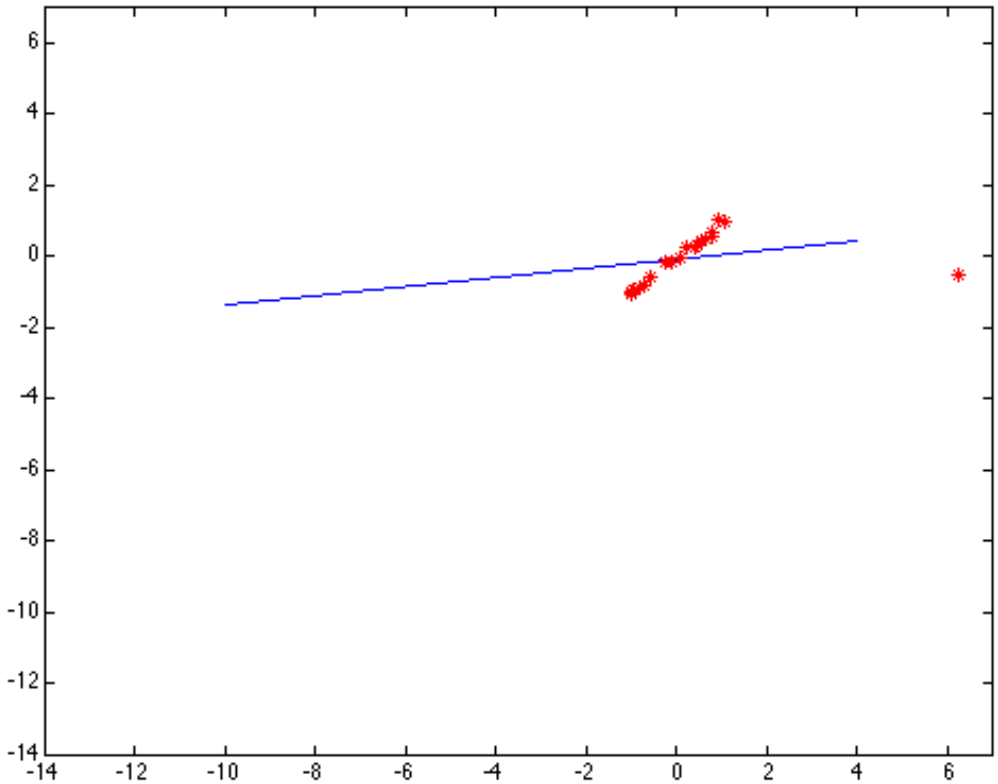
Matlab: `p = A \ y;` or `p = pinv(A)*y;`

Outliers affect least squares fit



Kristen Grauman

Outliers affect least squares fit



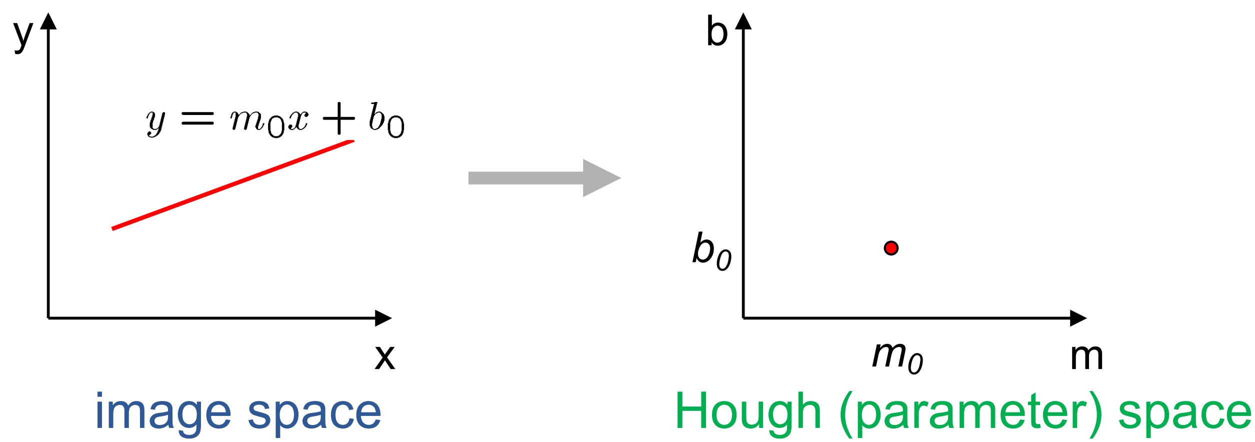
Kristen Grauman

Dealing with outliers: Voting

- **Voting** is a general technique where we let the features *vote for all models that are compatible with it*.
  - Cycle through features, cast votes for model parameters.
  - Look for model parameters that receive *a lot of votes*.
- Noise & clutter features?
  - They will cast votes too, *but* typically their votes should be inconsistent with the majority of "good" features.
- Common techniques
  - Hough transform
  - RANSAC

Adapted from Kristen Grauman

Finding lines in an image: Hough space



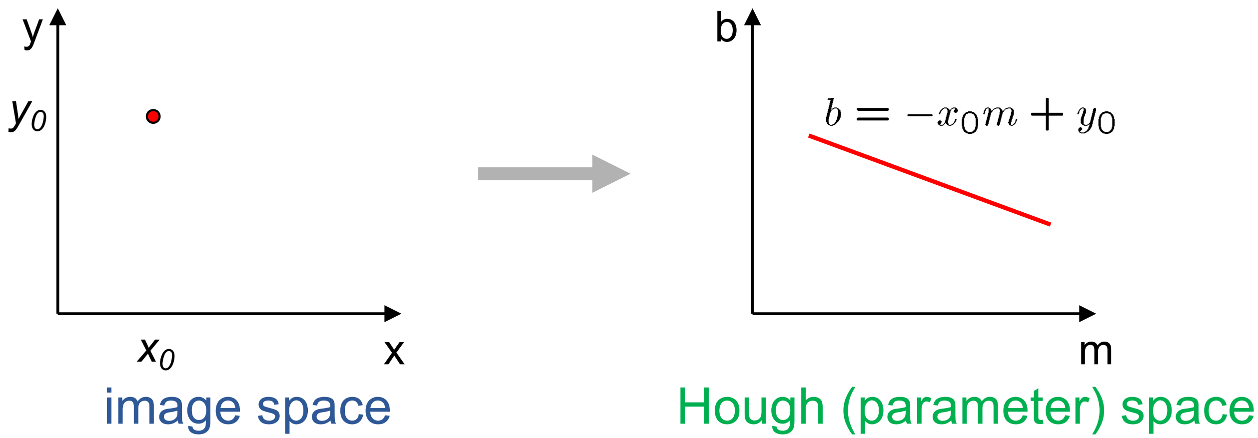
Connection between image (x,y) and Hough (m,b) spaces

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

$$y = m_0 x + b_0$$

Steve Seitz

Finding lines in an image: Hough space

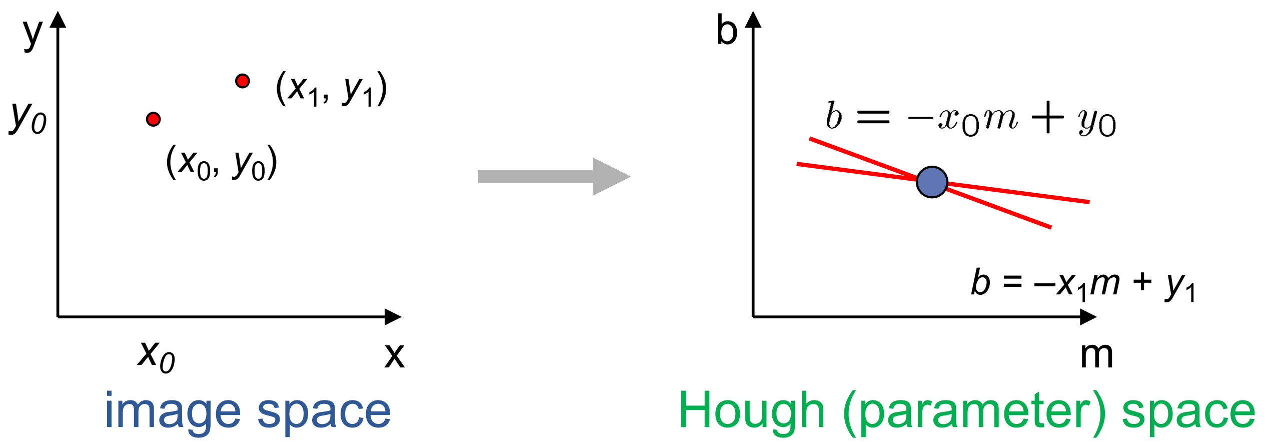


Connection between image (x,y) and Hough (m,b) spaces

- A line in the image corresponds to a point in Hough space $y = m_0 x + b_0$
- What does a point (x_0, y_0) in the image space map to?
  - Answer: the solutions of $b = -x_0 m + y_0$
  - This is a line in Hough space
  - Given a pair of points (x,y), find all (m,b) such that y = mx + b

Adapted from Steve Seitz

Finding lines in an image: Hough space

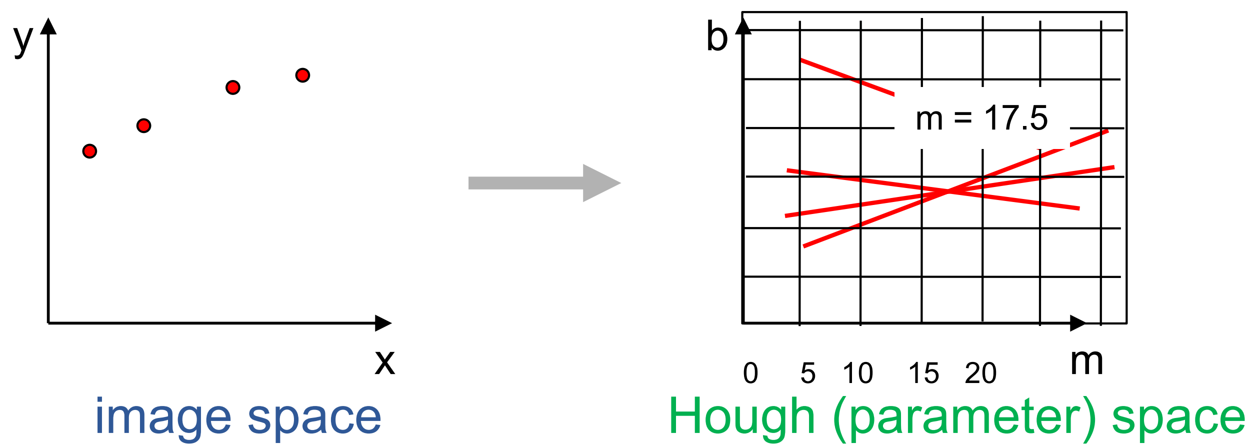


What are the line parameters for the line that contains both (x_0, y_0) and (x_1, y_1)?

- It is the intersection of the lines $b = -x_0 m + y_0$ and $b = -x_1 m + y_1$

Steve Seitz

Finding lines in an image: Hough space

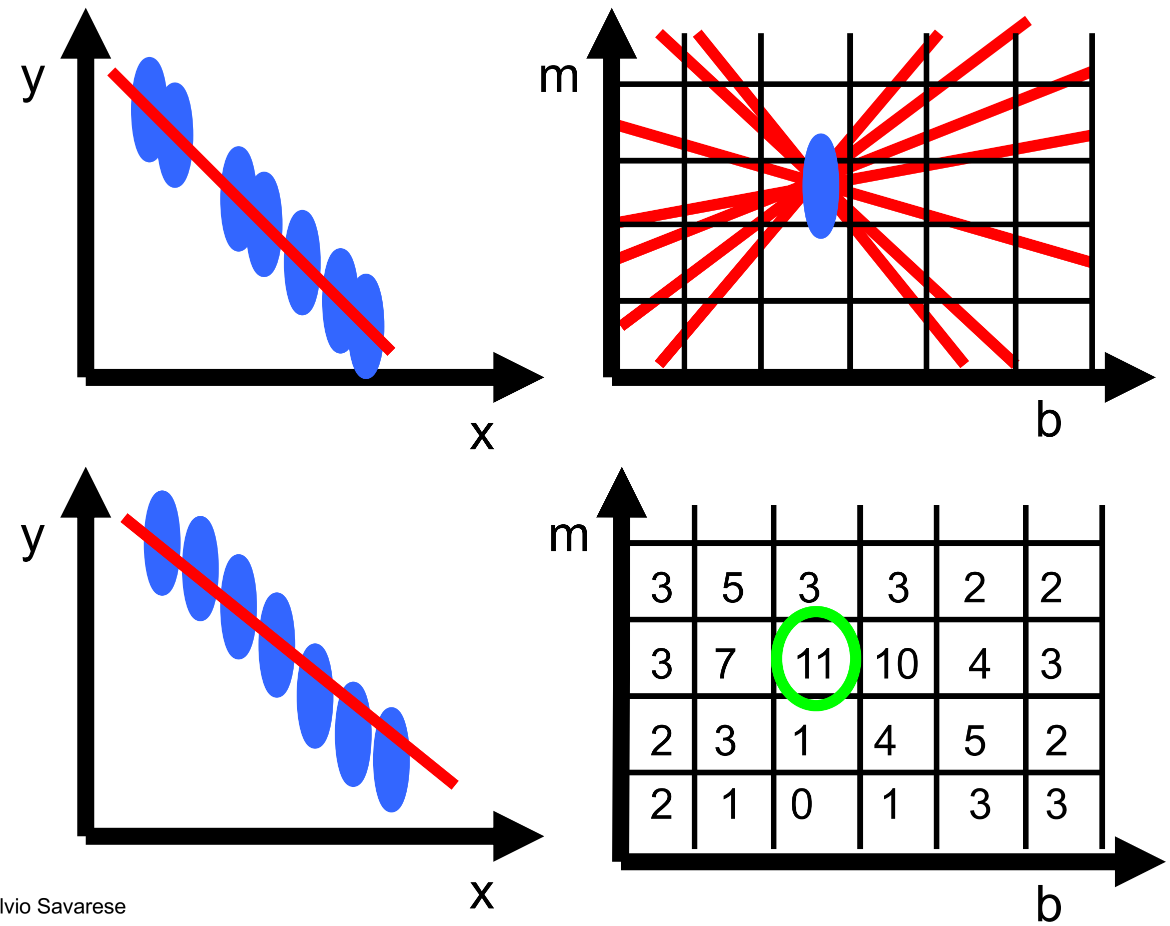


How can we use this to find the most likely parameters (m,b) for the most prominent line in the image space?

- Let each edge point in image space *vote* for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.

Steve Seitz

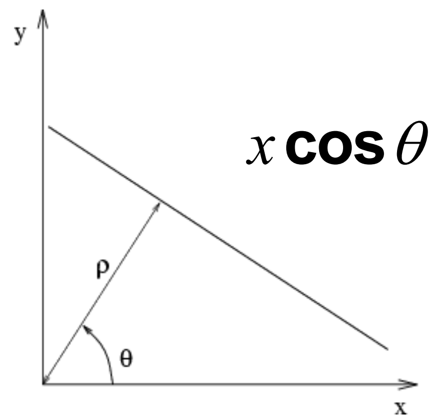
Finding lines in an image: Hough space



Adapted from Silvio Savarese

Parameter space representation

- Problems with the (m, b) space:
  - Unbounded parameter domains
  - Vertical lines require infinite m
- Alternative: *polar representation*



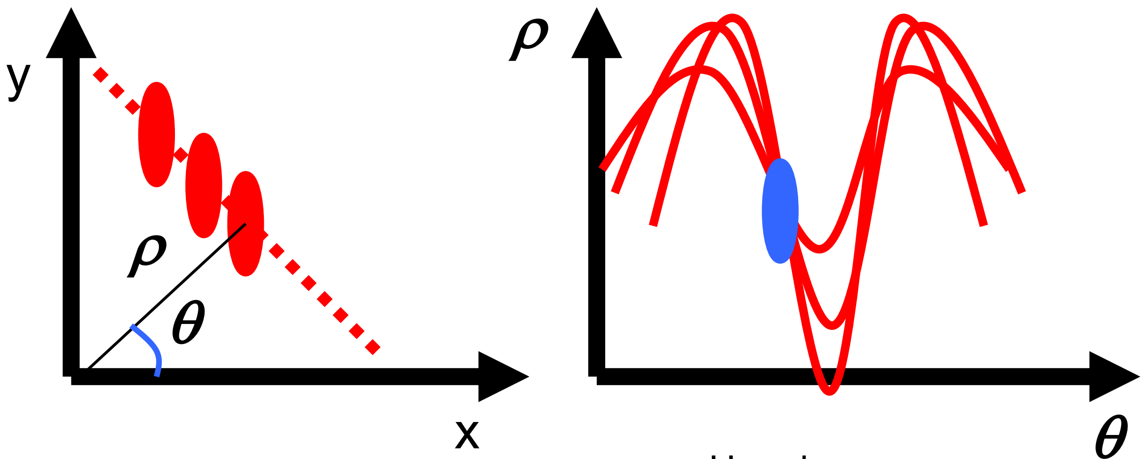
$$x\cos\theta + y\sin\theta = \rho$$

Each point (x,y) will add a sinusoid in the (θ,ρ) parameter space

Svetlana Lazebnik

Parameter space representation

- Problems with the (m,b) space:
  - Unbounded parameter domains
  - Vertical lines require infinite m
- Alternative: *polar representation*



Each point (x,y) will add a sinusoid in the (θ,ρ) parameter space

Svetlana Lazebnik

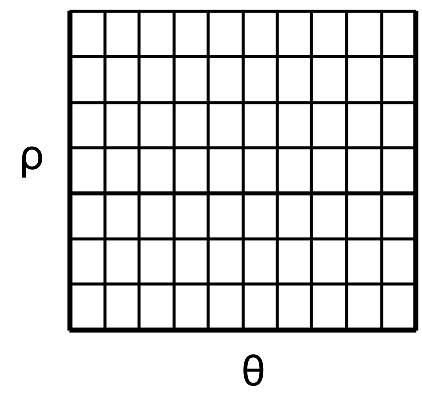
Algorithm outline: Hough transform

- Initialize accumulator H to all zeros

- For each edge point (x,y) in the image
  - For θ = 0 to 180
    - $\rho = x \cos \theta + y \sin \theta$
    - $H(\theta, \rho) = H(\theta, \rho) + 1$
  - end
- end

Why only until 180 degrees?

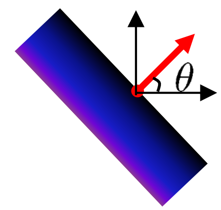
H: accumulator array (votes)



- Find the value(s) of (θ^*, ρ^*) where $H(\theta, \rho)$ is a local maximum
  - The detected line in the image is given by $\rho^* = x \cos \theta^* + y \sin \theta^*$

Svetlana Lazebnik

Incorporating Image Gradients



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

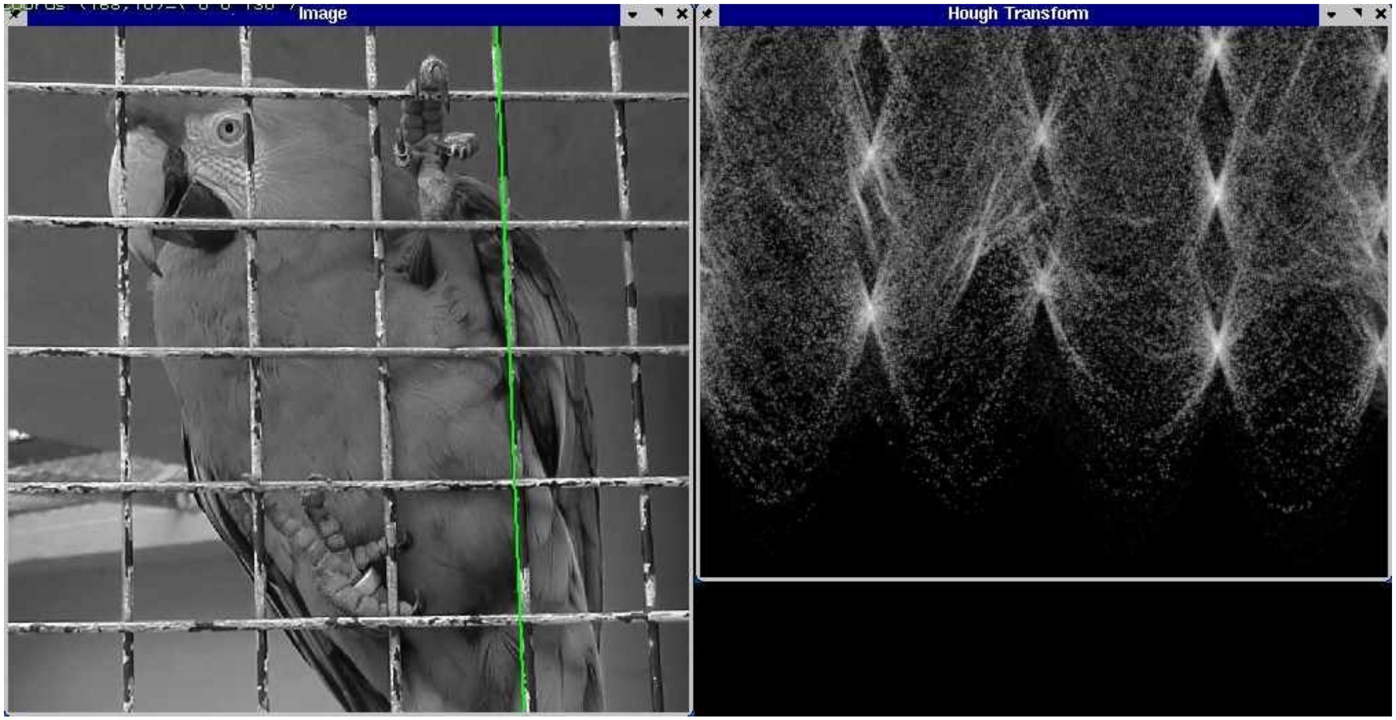
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- Recall: when we detect an edge point, we also know its gradient direction
- But this means that the line is uniquely determined!
- Modified Hough transform:

For each edge point (x,y) in the image
    θ = gradient orientation at (x,y)
    ρ = x cos θ + y sin θ
    H(θ, ρ) = H(θ, ρ) + 1
end

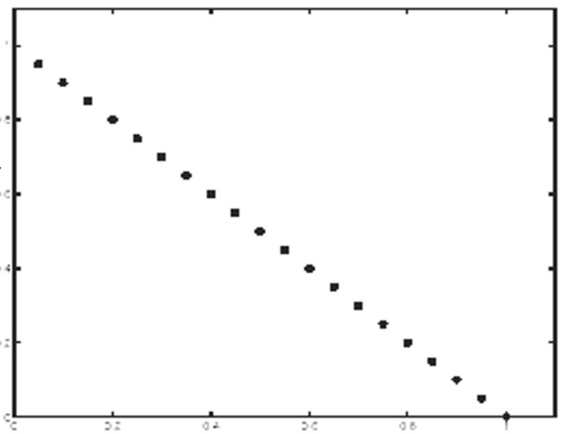
Svetlana Lazebnik

Hough transform example



Derek Hoiem

Impact of noise on Hough



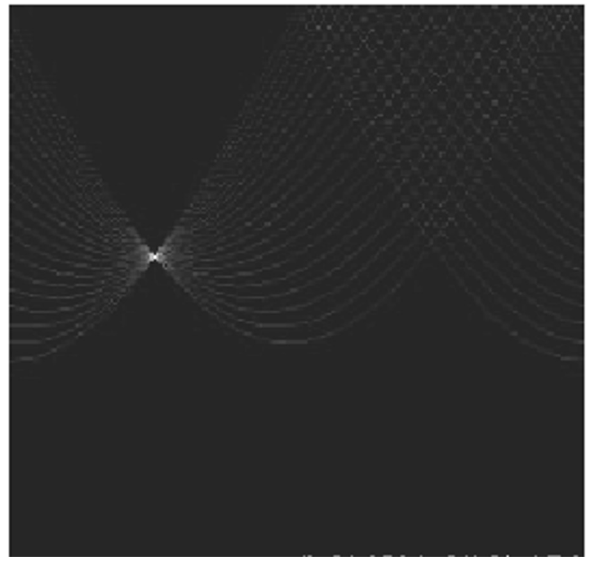


Image space edge coordinates

Votes

Kristen Grauman

Impact of noise on Hough

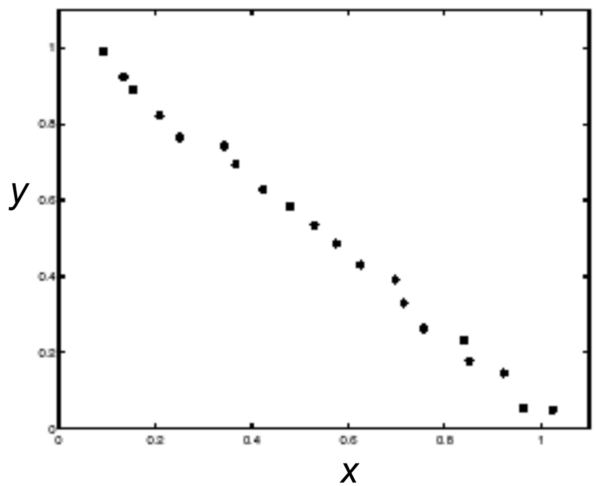
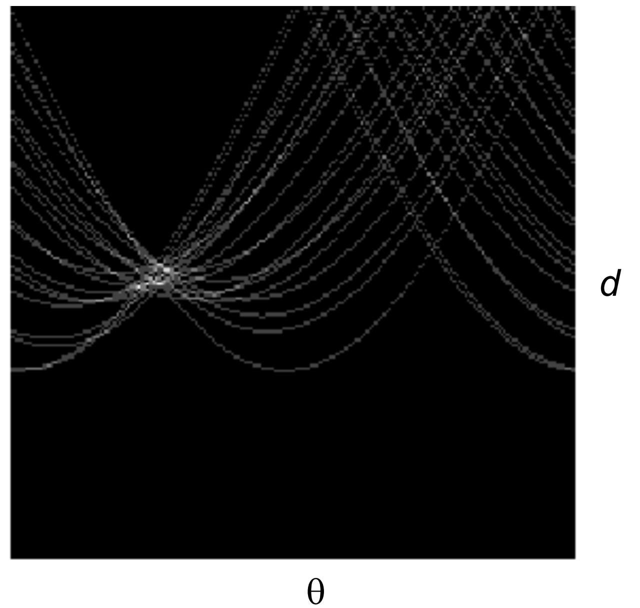


Image space edge coordinates



Votes

What difficulty does this present for an implementation?

Kristen Grauman

Voting: practical tips

- Minimize irrelevant tokens first (reduce noise)
- Choose a good grid / discretization

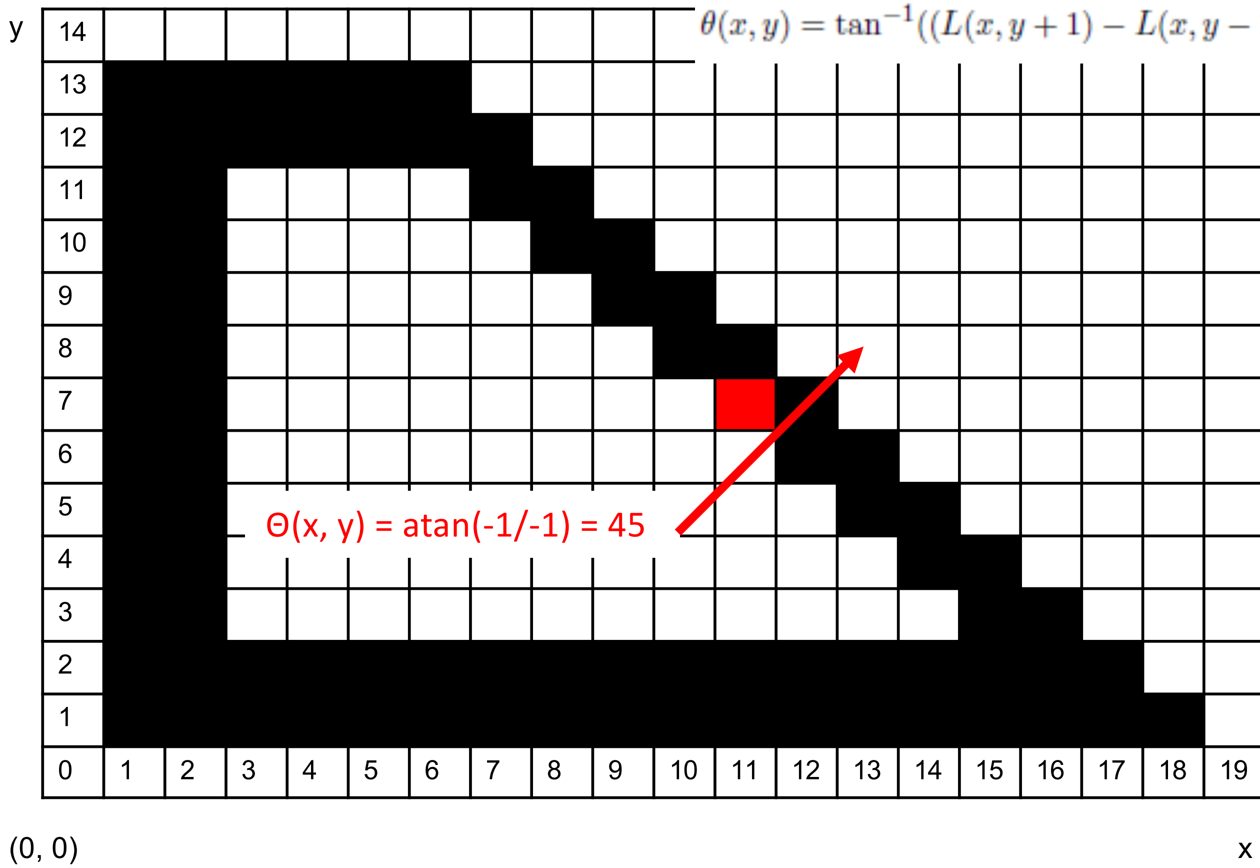
  Too coarse ← ? → Too fine

  - **Too coarse:** large votes obtained when too many different lines correspond to a single bucket
  - **Too fine:** miss lines because points that are not exactly collinear cast votes for different buckets
- Vote for neighbors (smoothing in accumulator array)
- Use direction of edge to reduce parameters by 1
- To read back which points voted for “winning” peaks, keep tags on the votes

Kristen Grauman

Hough Transform: Example

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$



ρ = x cos(Θ) + y sin(Θ)

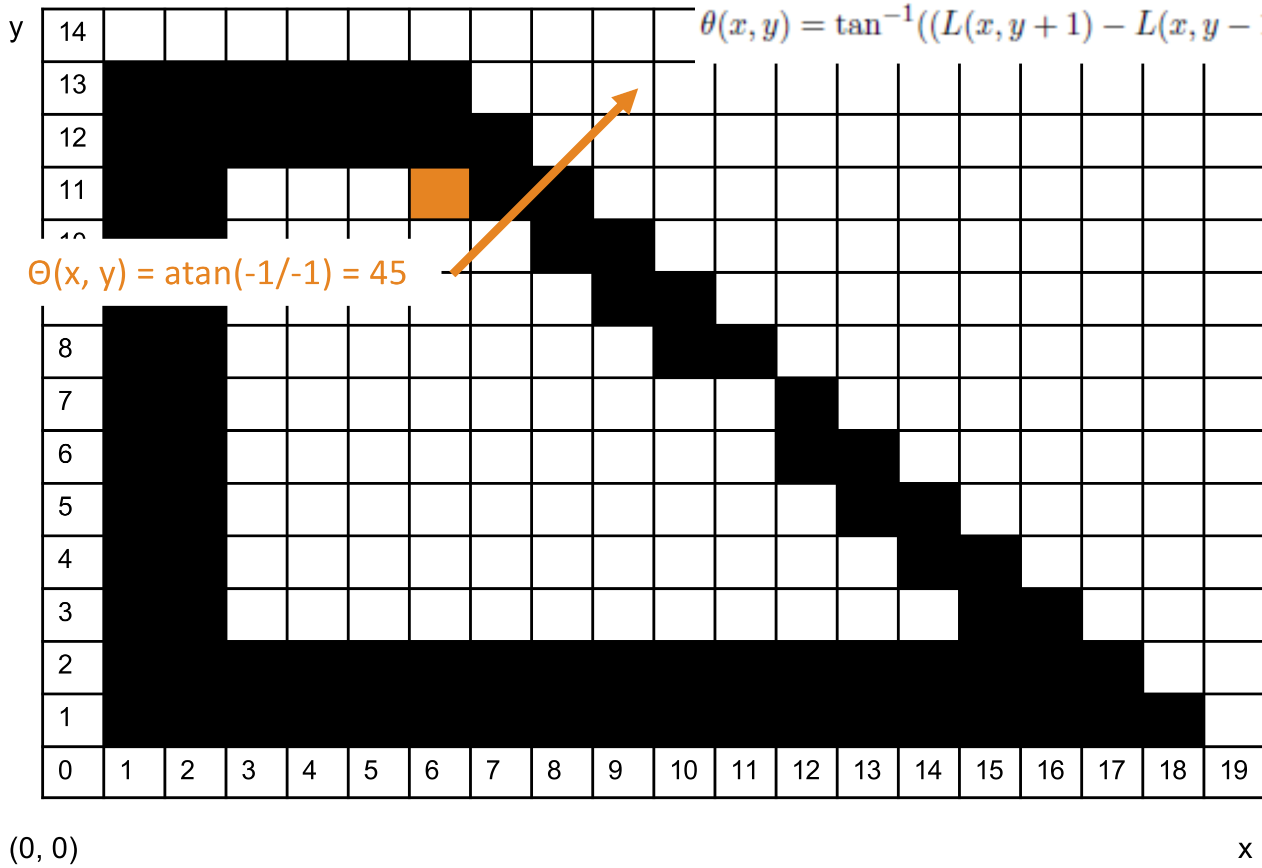
ρ = 11 cos(45) + 7 sin(45)

ρ = 12.73

H(θ=45, ρ=12.73) = 1

Hough Transform: Example

$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$



ρ = x cos(Θ) + y sin(Θ)

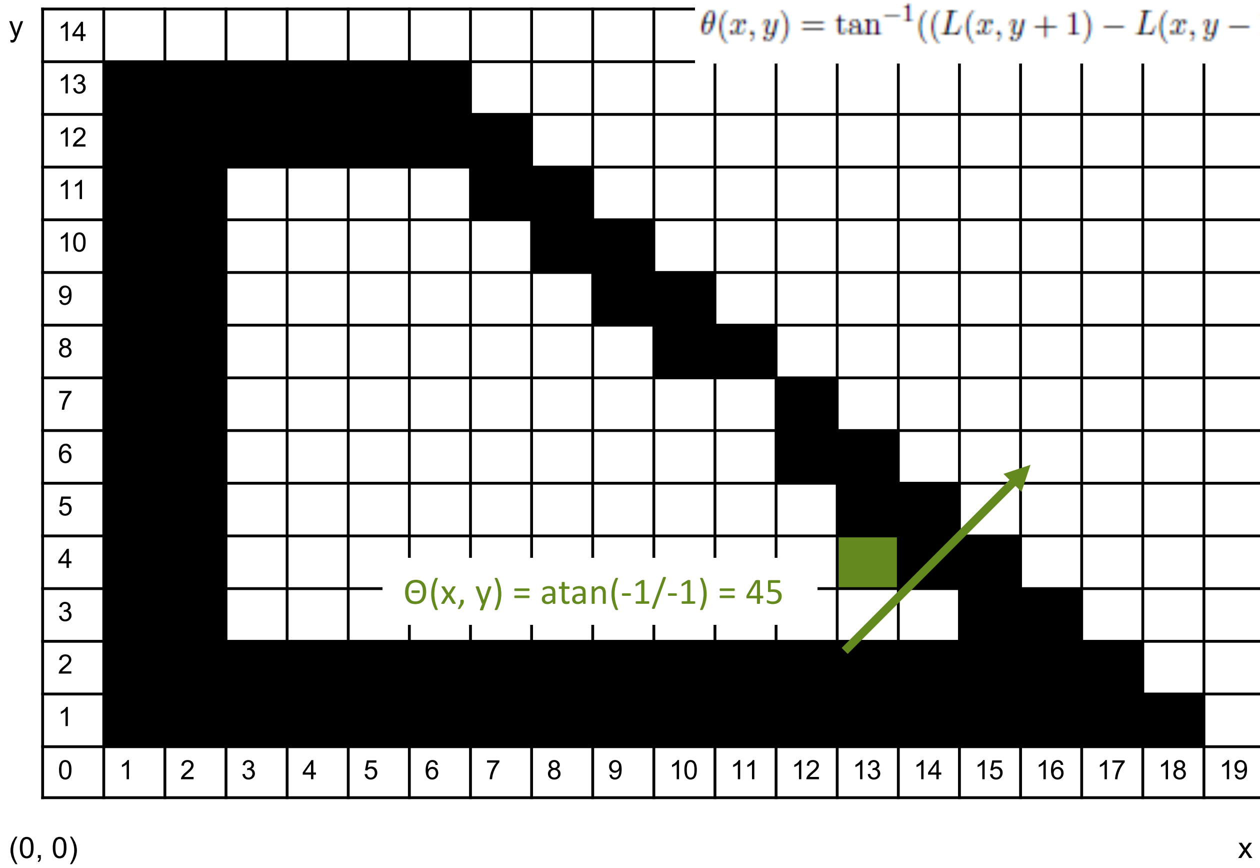
ρ = 6 cos(45) + 11 sin(45)

ρ = 12.02

H(θ=45, ρ= 12.02) = 1

Hough Transform: Example

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$



$\rho = x \cos(\Theta) + y \sin(\Theta)$

$\rho = 13 \cos(45) + 4 \sin(45)$

$\rho = 12.02$

$H(\theta=45, \rho= 12.02) = 2$

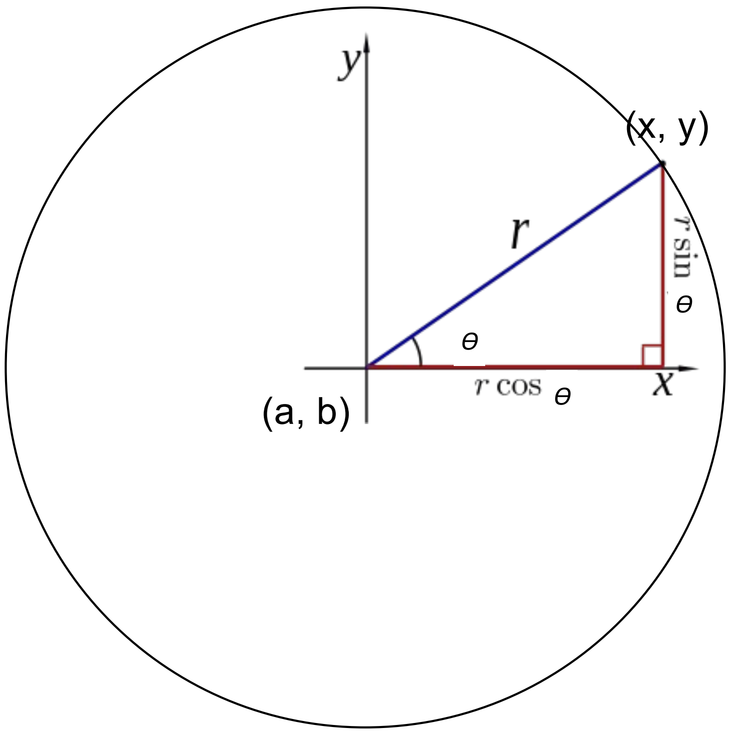
$H(\theta=45, \rho=12.73) = 1$
$H(\theta=45, \rho= 12.02) = 2$

Hough Transform for Circles

- A circle with radius *r* and center (*a, b*) can be described as:

$x = a + r\cos(\theta)$

$y = b + r\sin(\theta)$



Hough Transform for Circles

- Circle: center (*a*, *b*) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

  - For a fixed radius r, unknown gradient direction

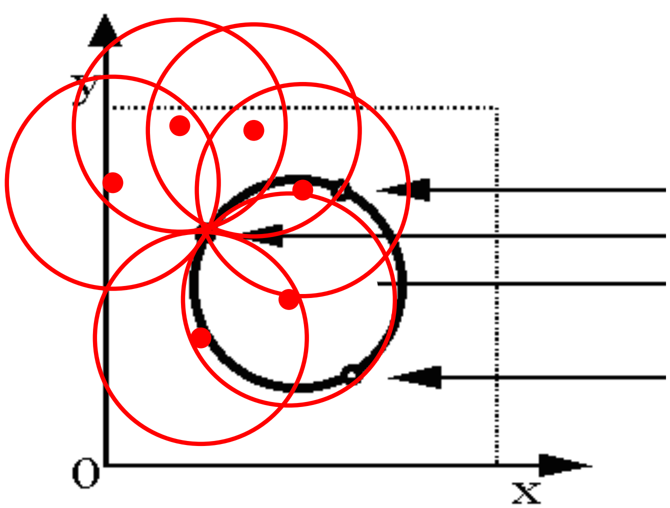
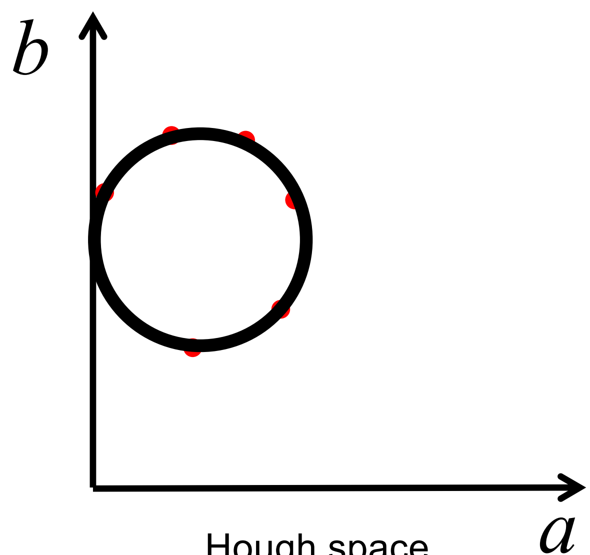


Image space



Hough space

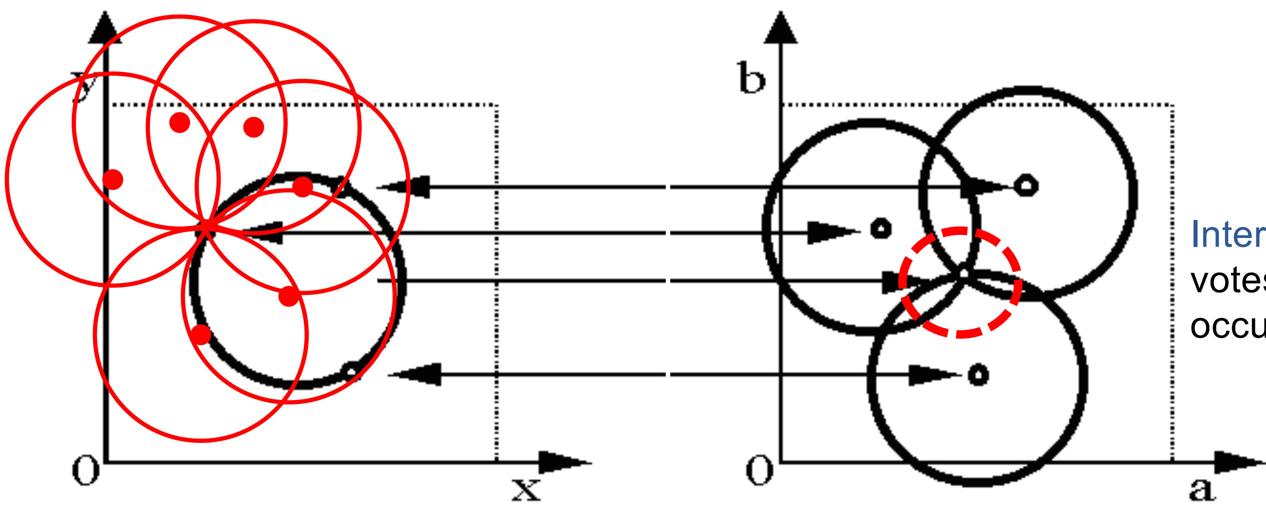
Kristen Grauman

Hough Transform for Circles

- Circle: center (*a, b*) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

  - For a fixed radius r, unknown gradient direction



Intersection: most votes for center occur here.

Image space

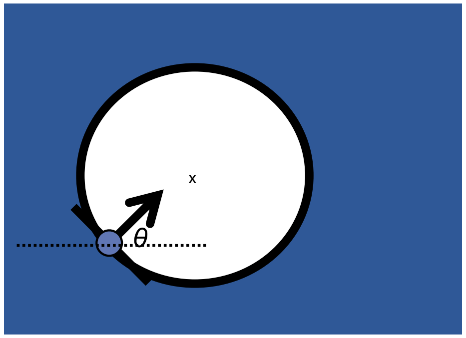
Hough space

Kristen Grauman

Hough Transform for Circles

$x = a + r\cos(\theta)$
$y = b + r\sin(\theta)$

For every edge pixel (x,y) :
    For each possible radius value r:
        For each possible gradient direction θ:
        // or use estimated gradient at (x,y)
            $a = x - r\cos(\theta)$ // column
            $b = y - r\sin(\theta)$ // row
            $H[a,b,r]$ += 1
        end
    end
end

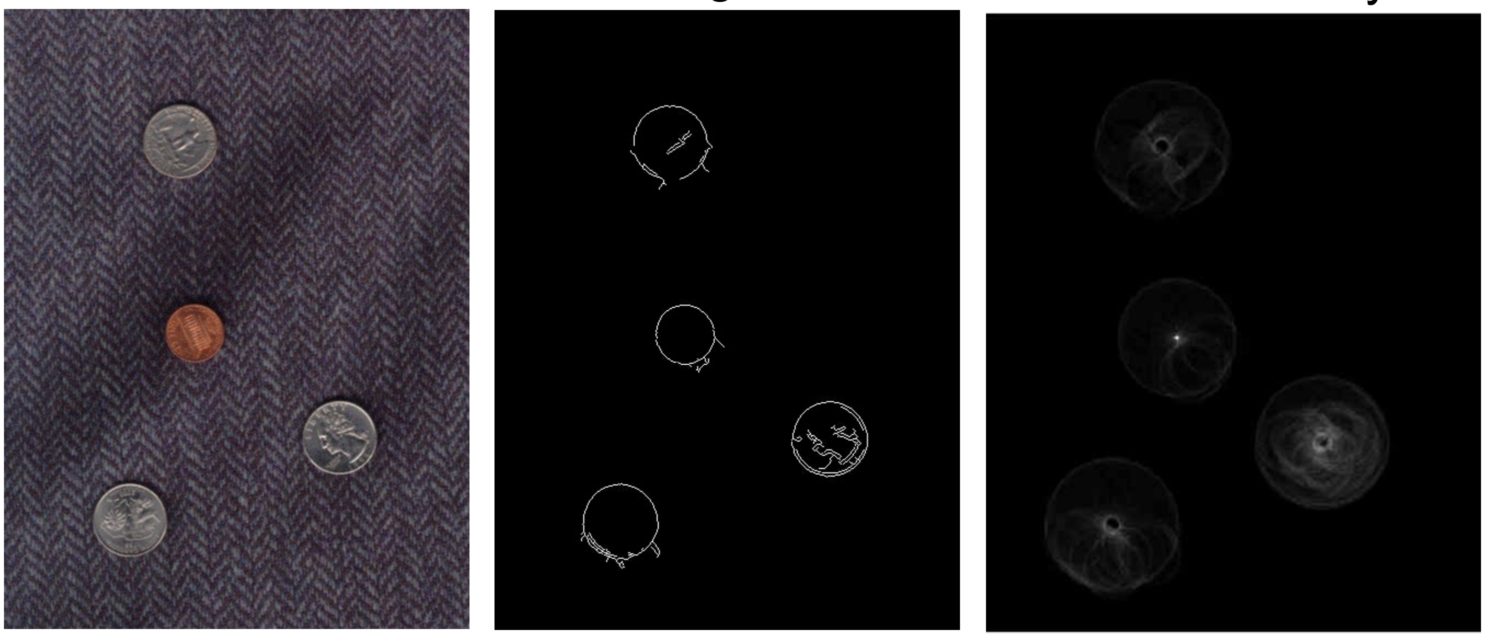


Your homework!

Modified from Kristen Grauman

Example: Detecting Circles with Hough

Original | Edges | Votes: Penny

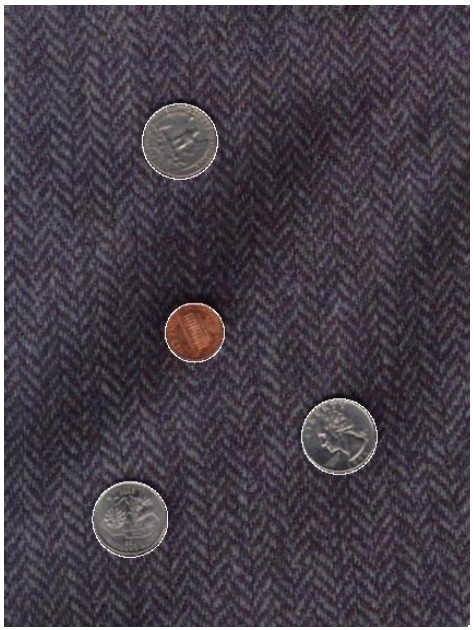


Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

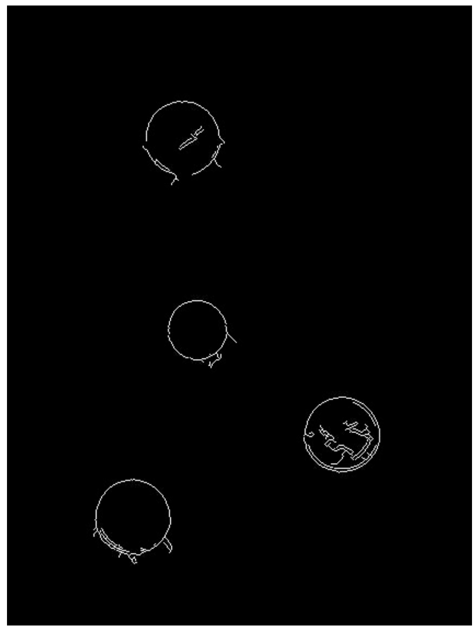
Kristen Grauman, images from Vivek Kwatra

Example: Detecting Circles with Hough

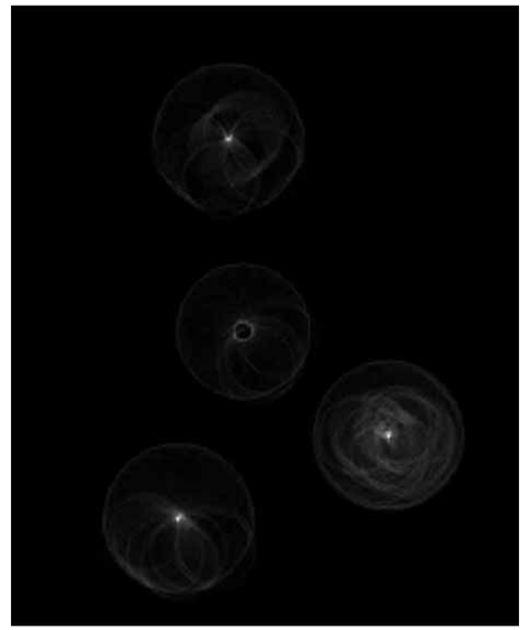
Original



Edges



Votes: Quarter



Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

Kristen Grauman, images from Vivek Kwatra

Hough transform: pros and cons

Pros

- All points are processed independently, so can cope with occlusion, gaps
- Some robustness to noise: noise points *unlikely* to contribute *consistently* to any single bin
- Can detect multiple instances of a model in a single pass

Cons

- Complexity of search time for maxima increases exponentially with the number of model parameters
  - If 3 parameters and 10 choices for each, search is $O(10^3)$
- Quantization: can be tricky to pick a good grid size

Adapted from Kristen Grauman

Hough transform

[Hough Transform Demo]

RANSAC

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

Kristen Grauman

RANSAC: General Form

RANSAC loop:

1. Randomly select a *seed group* of ***s*** points on which to base model estimate (e.g. ***s***=2 for a line)
2. Fit model to these ***s*** points
3. Find *inliers* to this model (i.e., points whose distance from the line is less than ***t***)
4. Repeat ***N*** times

- Keep the model with the largest number of inliers

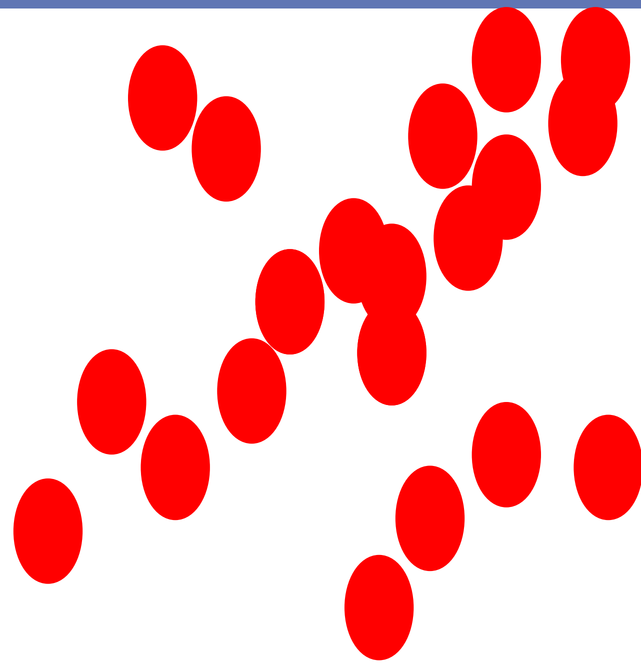
Adapted from Kristen Grauman and Svetlana Lazebnik

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in '81.

Line fitting example



Algorithm:

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

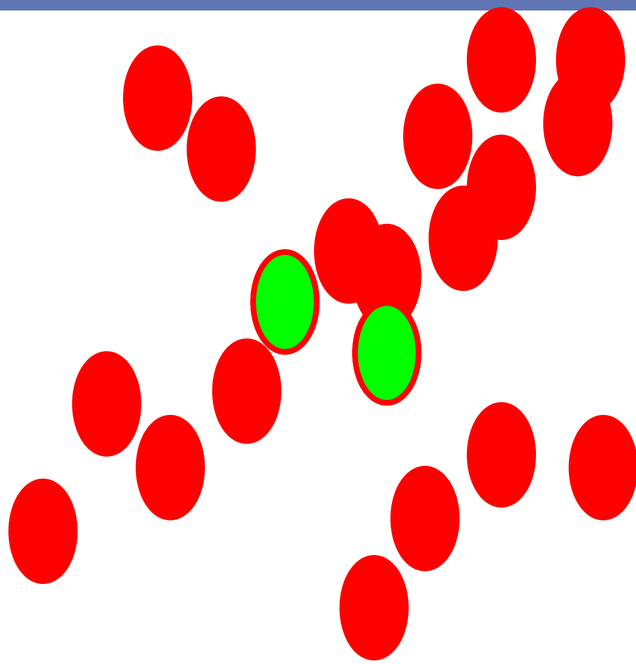
Repeat 1-3 until the best model is found with high confidence

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in '81.

Line fitting example



Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

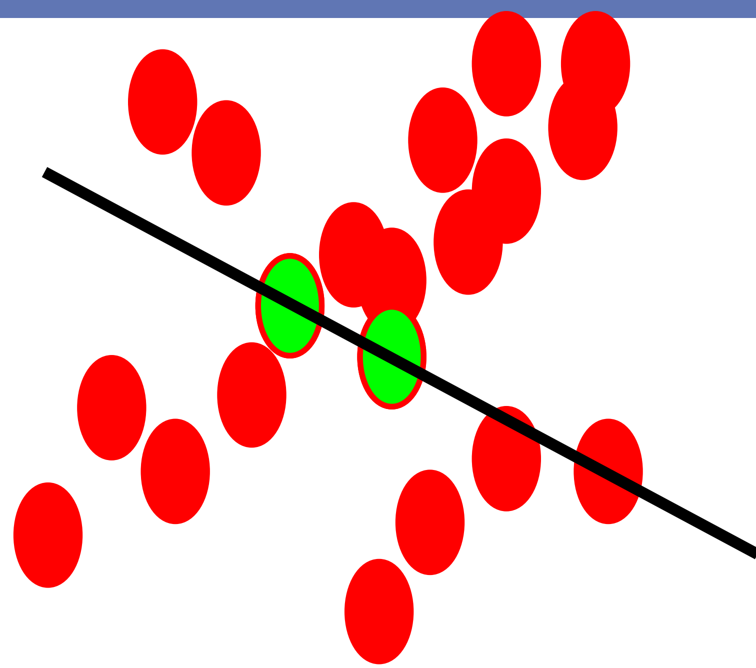
Repeat 1-3 until the best model is found with high confidence

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in '81.

Line fitting example



Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

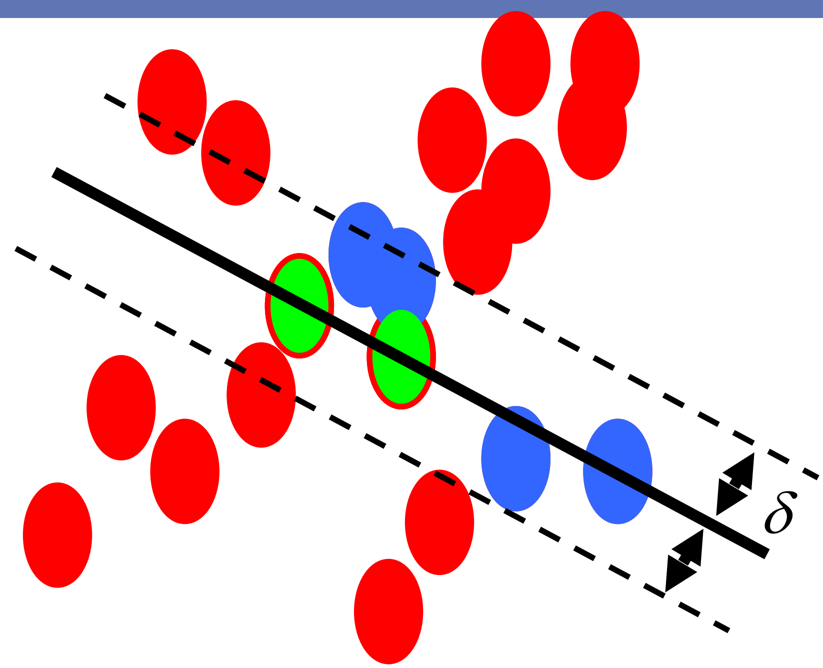
Repeat 1-3 until the best model is found with high confidence

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in '81.

Line fitting example



$$N_I = 6$$

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

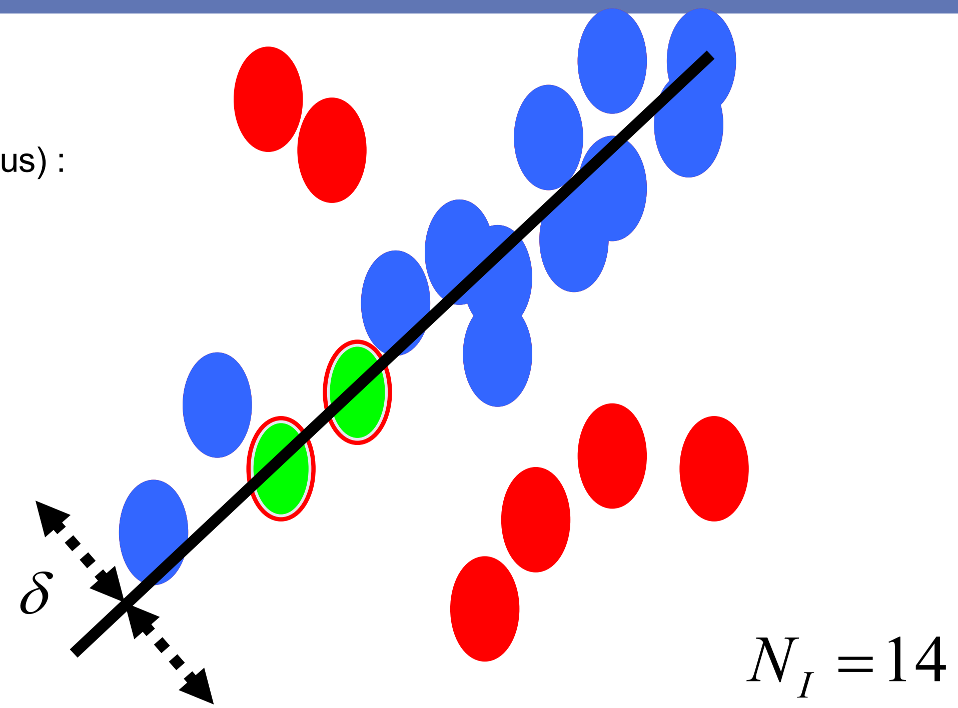
Repeat 1-3 until the best model is found with high confidence

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in '81.

Line fitting example



Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC pros and cons

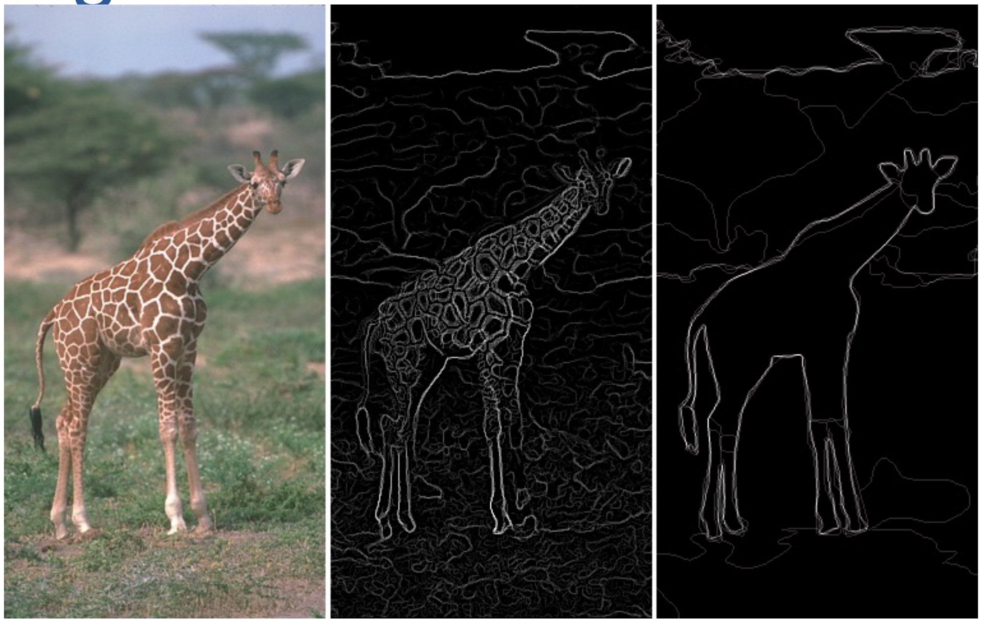
- Pros
  - Applicable to many different problems, e.g. image stitching, relating two views
  - Often works well in practice
- Cons
  - Lots of parameters to tune (see previous slide)
  - Doesn't work well for low inlier ratios (too many iterations, or can fail completely)

Adapted from Svetlana Lazebnik

Plan for today

- Edges
  - Extract gradients and threshold
- Lines and circles
  - Find which edge points are collinear or belong to another shape e.g. circle
  - Automatically detect and ignore outliers
- Segments
  - Find which pixels form a consistent region
  - Clustering (e.g. K-means)

Edges vs Segments



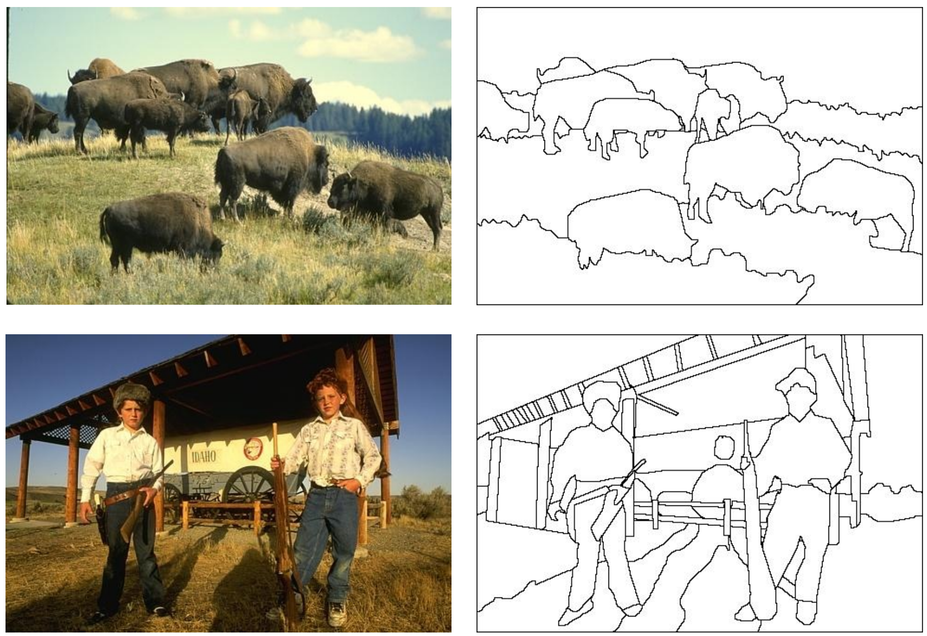
- Edges: More low-level; don't need to be closed
- Segments: Ideally one segment for each semantic group/object; should include closed contours

Figure adapted from J. Hays

The goals of segmentation

- Separate image into coherent “objects”

image　　human segmentation

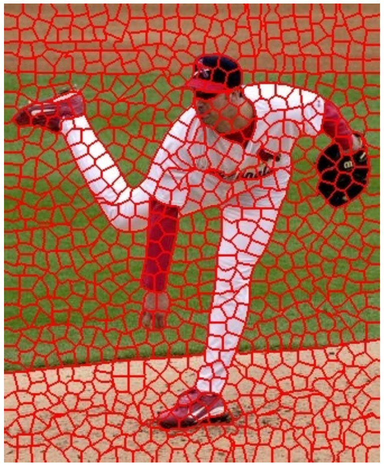


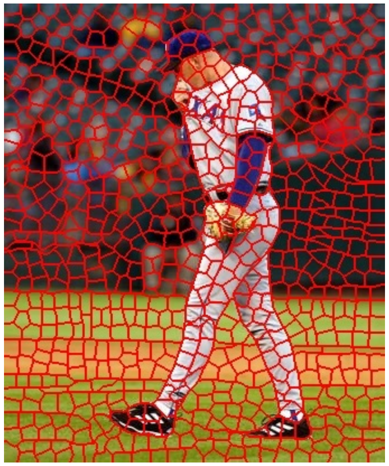
Source: L. Lazebnik

The goals of segmentation

- Separate image into coherent "objects"
- Group together similar-looking pixels for efficiency of further processing

"superpixels"





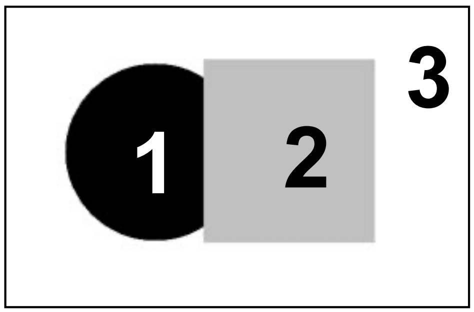
X. Ren and J. Malik. Learning a classification model for segmentation. ICCV 2003.

Source: L. Lazebnik

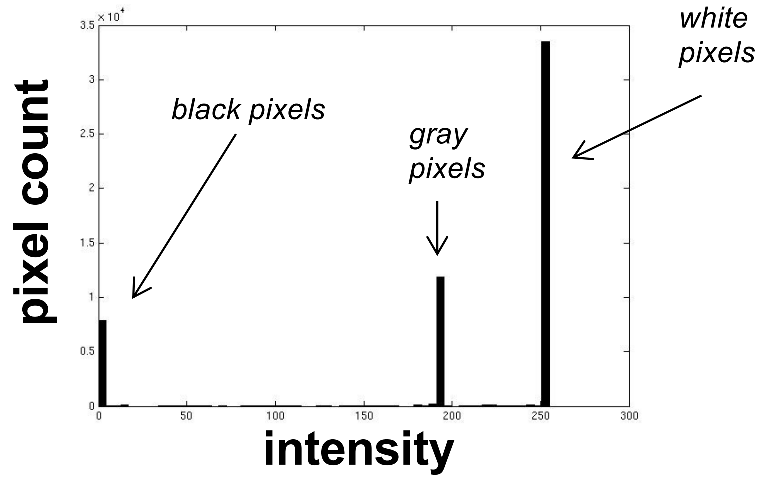
Similarity

Slide: K. Grauman

Image Segmentation: Toy Example



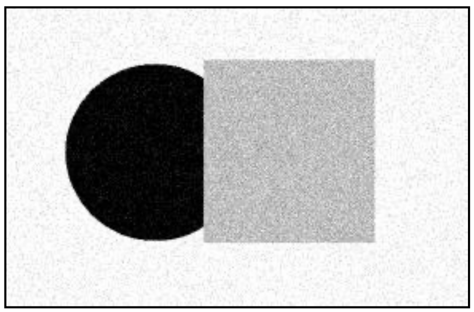
input image



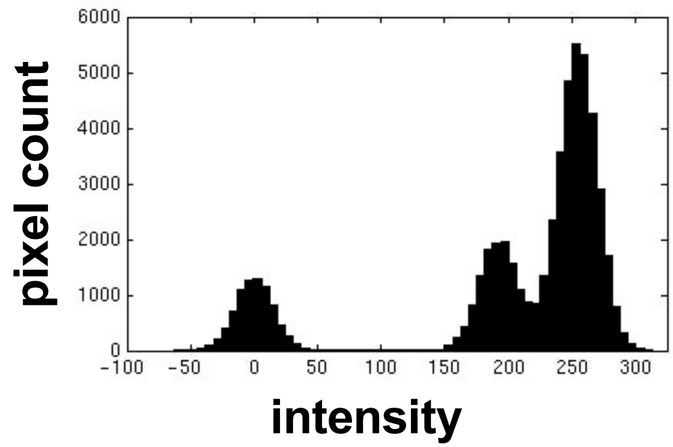
- These intensities define the three groups.
- We could label every pixel in the image according to which of these primary intensities it is.
    - i.e., *segment* the image based on the intensity feature.
- What if the image isn't quite so simple?

Source: K. Grauman

Image Segmentation: Toy Example



input image

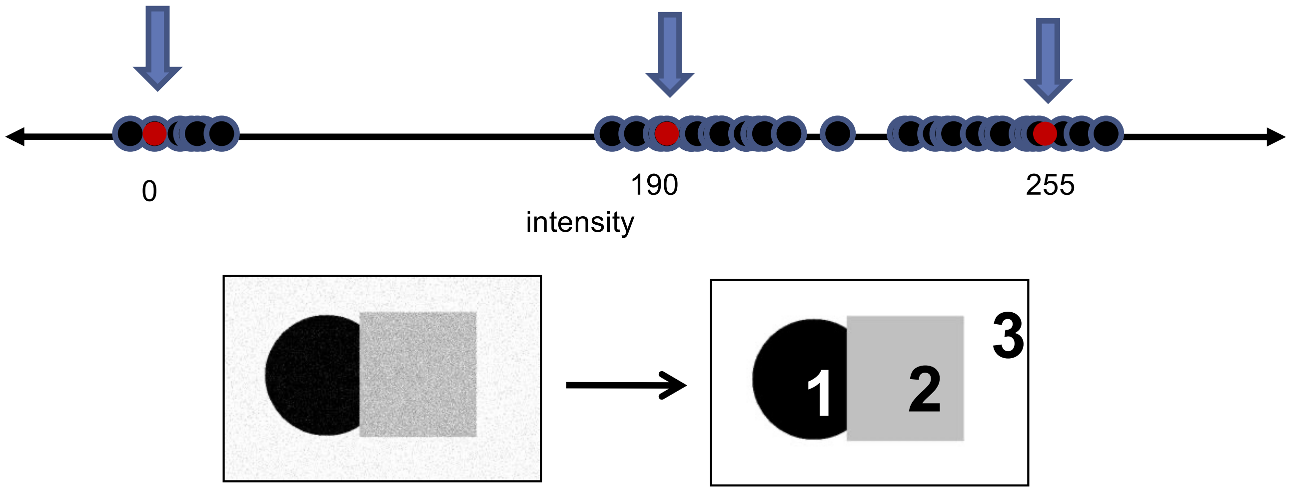


- Now how to determine the three main intensities that define our groups?

- We need to ***cluster.***

Source: K. Grauman

Image Segmentation: Toy Example



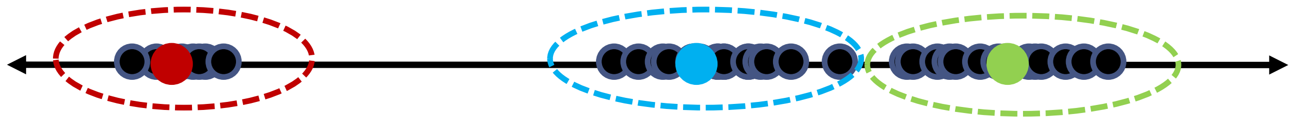
- Goal: choose three "centers" as the representative intensities, and label every pixel according to which of these centers it is nearest to.
- Best cluster centers are those that minimize *sum of squared differences* (SSD) between all points and their nearest cluster center c_i:

$$\sum_{\text{clusters } i} \sum_{\text{points p in cluster } i} ||p - c_i||^2$$

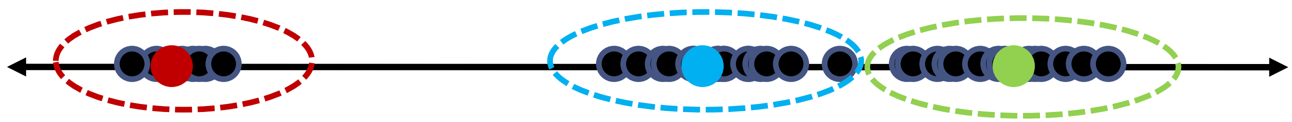
Source: K. Grauman

Clustering

- With this objective, it is a “chicken and egg” problem:
  - If we knew the **cluster centers**, we could allocate points to groups by assigning each to its closest center.



  - If we knew the **group memberships**, we could get the centers by computing the mean per group.



Source: K. Grauman

K-means clustering

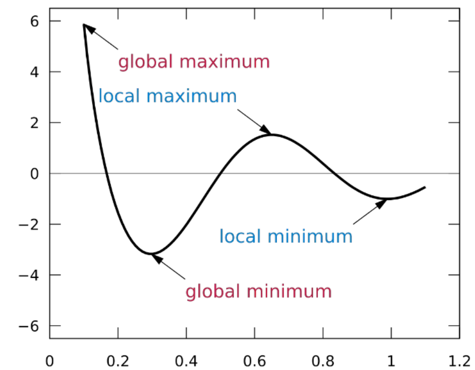
- **Basic idea**: randomly initialize the *k* cluster centers, and iterate between the two steps we just saw.

  1. *Randomly* initialize the cluster centers, $c_1, ..., c_K$
  2. *Given cluster centers*, determine points in each cluster
     - For each point p, find the closest c_i. Put p into cluster i
  3. *Given points in each cluster*, solve for c_i
     - Set c_i to be the mean of points in cluster i
  4. If c_i have changed, repeat Step 2

Properties

- Will always converge to *some* solution
- Can be a "local minimum" of objective:

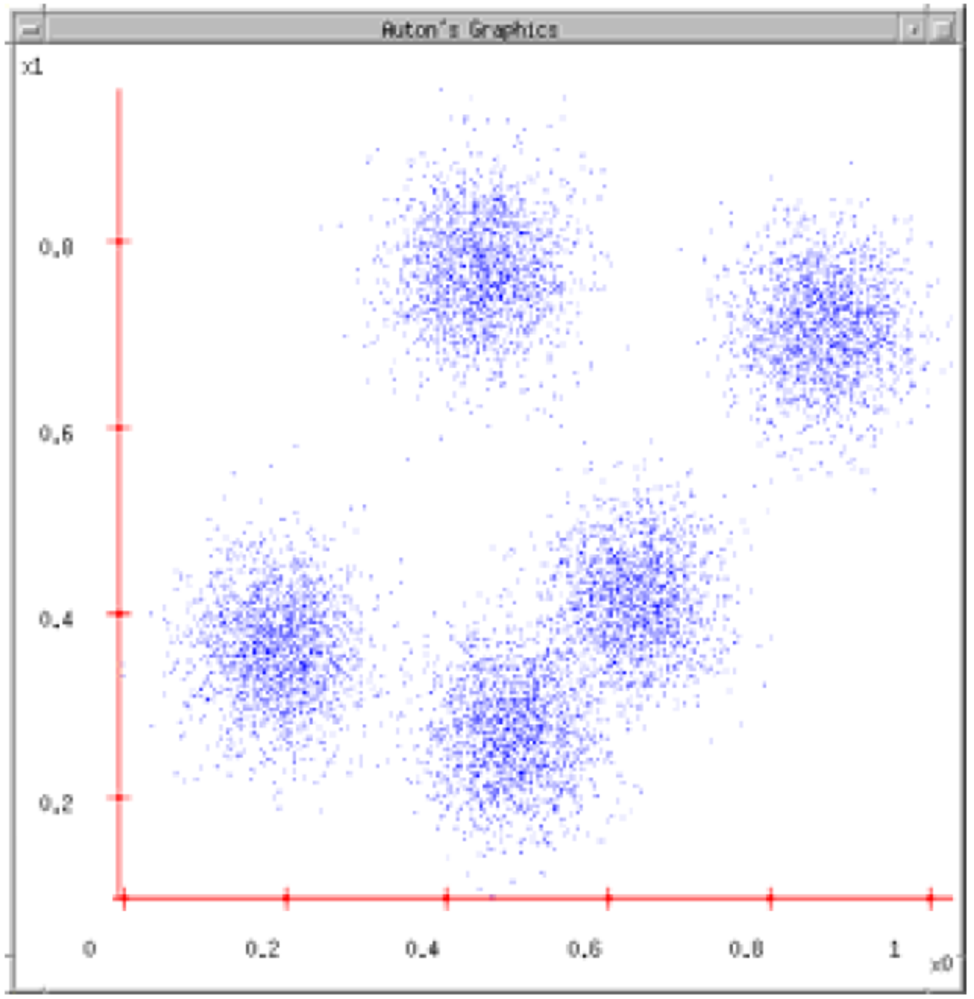
$$\sum_{\text{clusters } i} \sum_{\text{points p in cluster } i} ||p - c_i||^2$$



Slide: Steve Seitz, image: Wikipedia

K-means

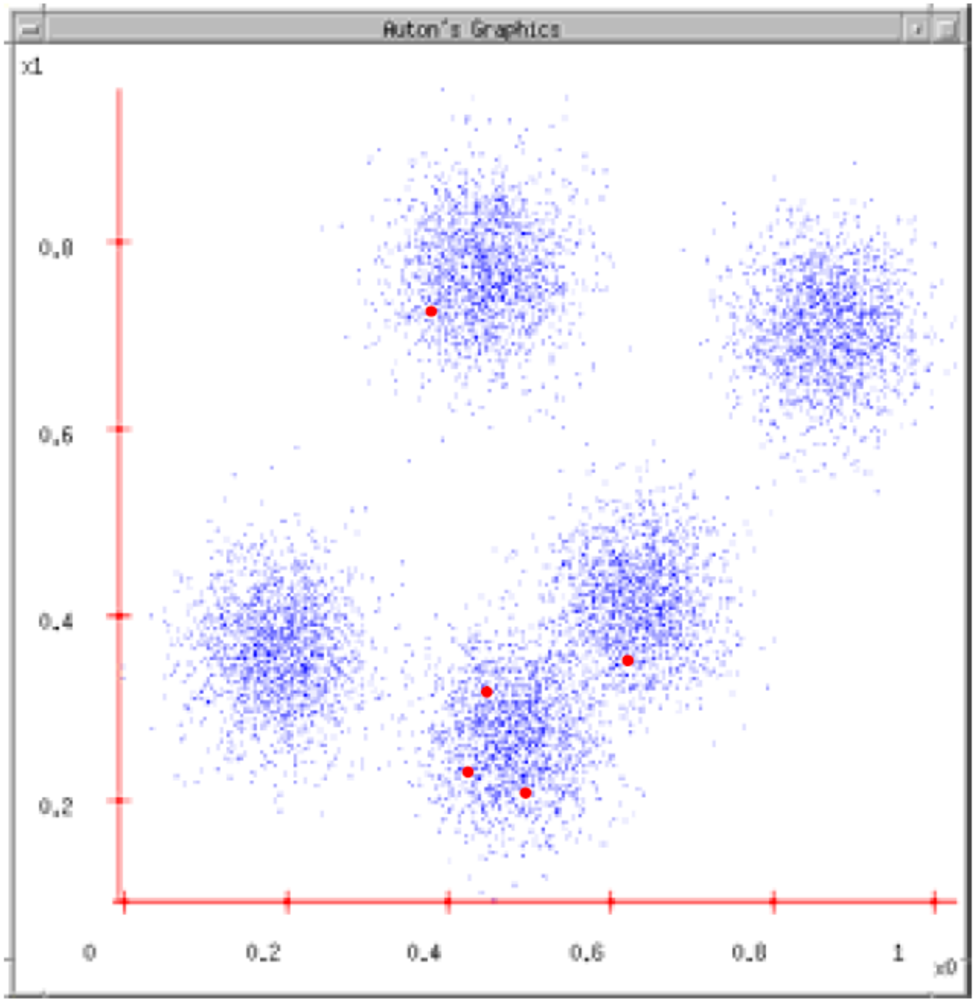
1. Ask user how many clusters they'd like. *(e.g. k=5)*



Source: A. Moore

K-means

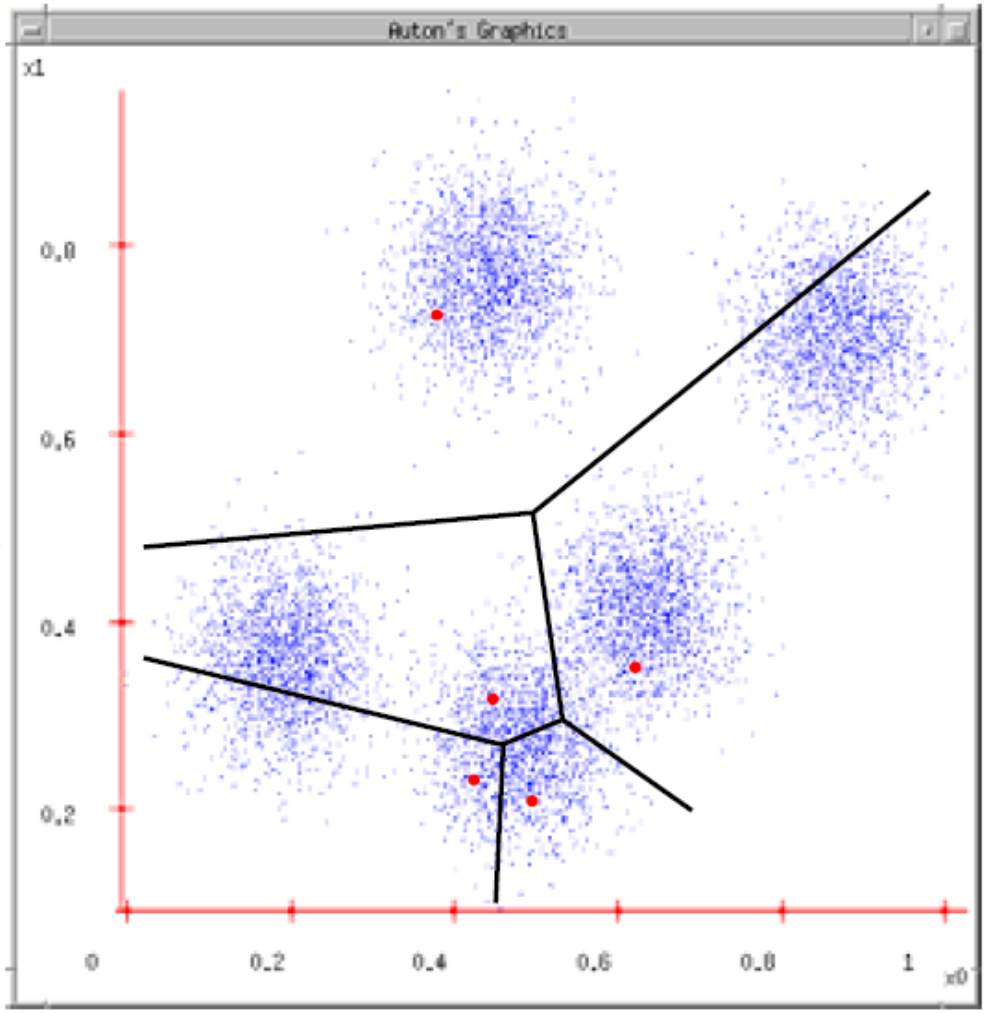
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations



Source: A. Moore

K-means

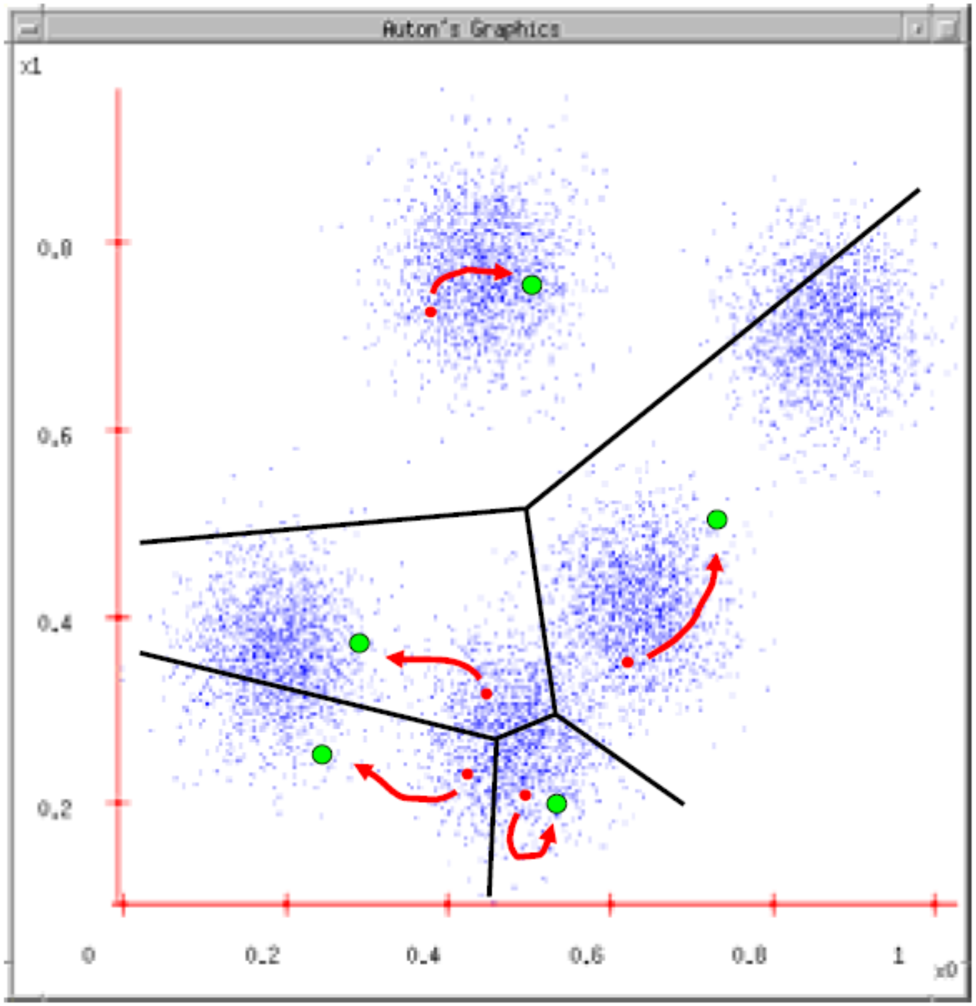
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



Source: A. Moore

K-means

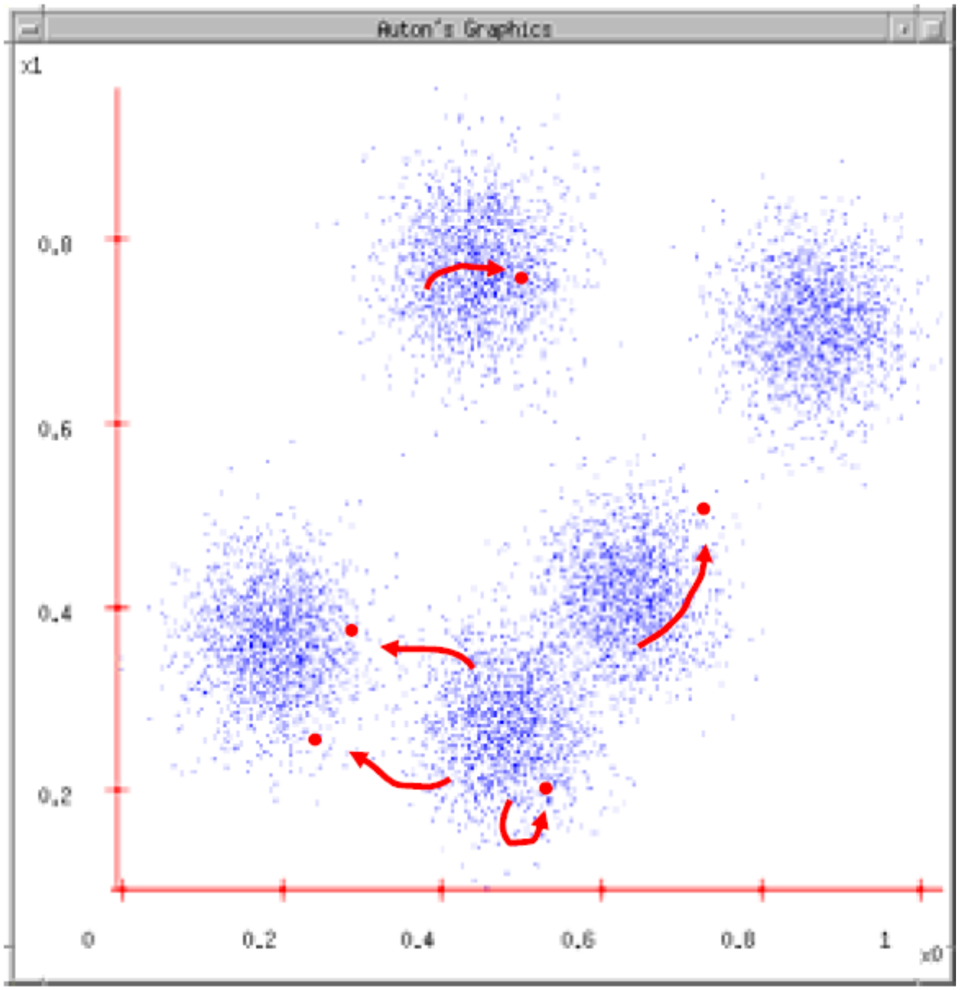
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



Source: A. Moore

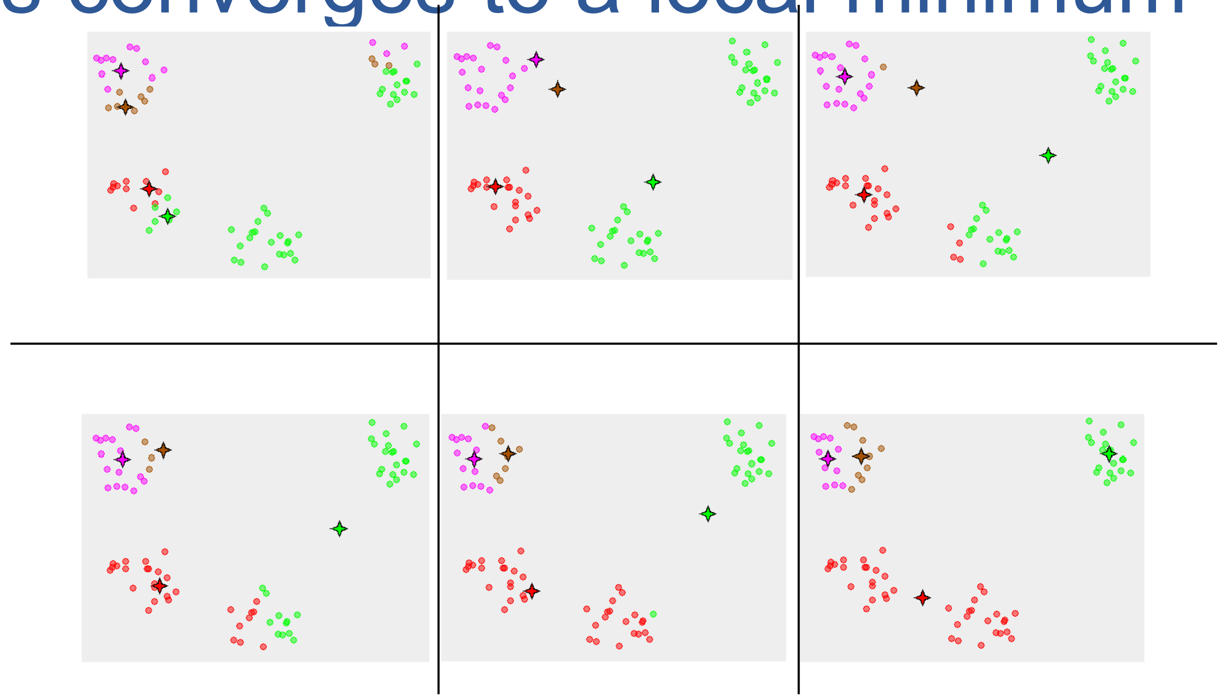
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!



Source: A. Moore

K-means converges to a local minimum



How can I try to fix this problem?

Adapted from James Hays

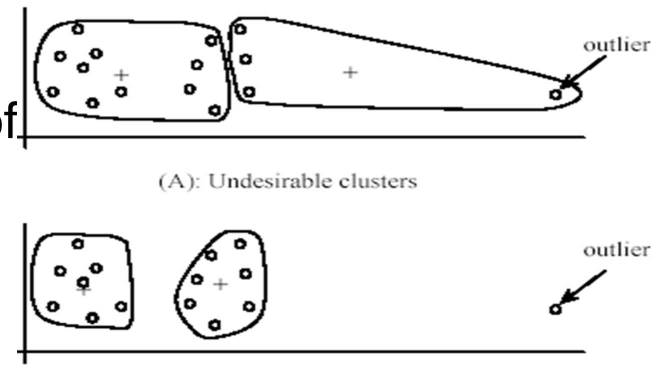
K-means: pros and cons

Pros

- Simple, fast to compute
- Converges to local minimum of within-cluster squared error

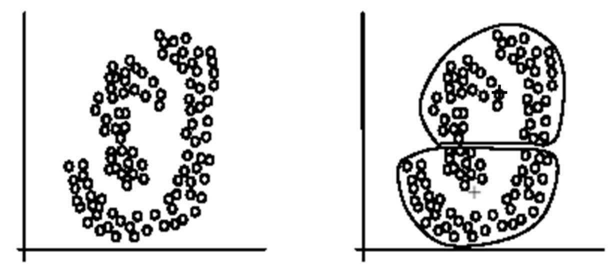
Cons/issues

- Setting k?
  - One way: silhouette coefficient
- Sensitive to initial centers
  - Use heuristics or output of another method
- Sensitive to outliers
- Detects spherical clusters



(A): Undesirable clusters

(B): Ideal clusters



(A): Two natural clusters (B): *k*-means clusters

Adapted from K. Grauman

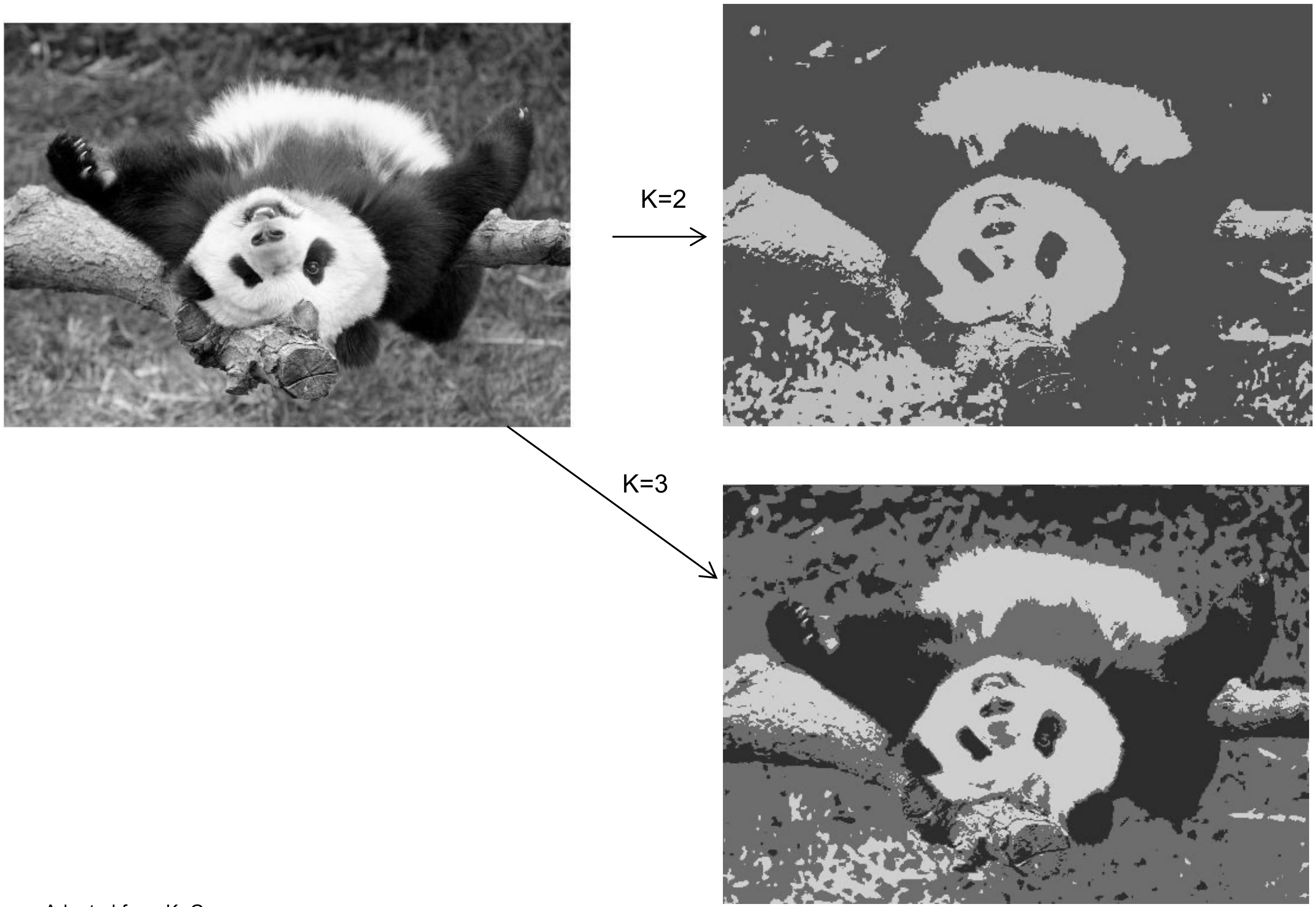
Segmentation as Clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **intensity** similarity

Feature space: intensity value (1-d)

Source: K. Grauman



Adapted from K. Grauman

Segmentation as Clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **intensity** similarity

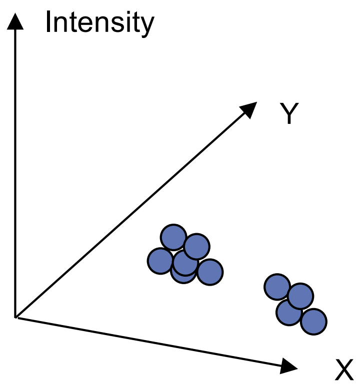
Clusters based on intensity similarity don’t have to be spatially coherent.

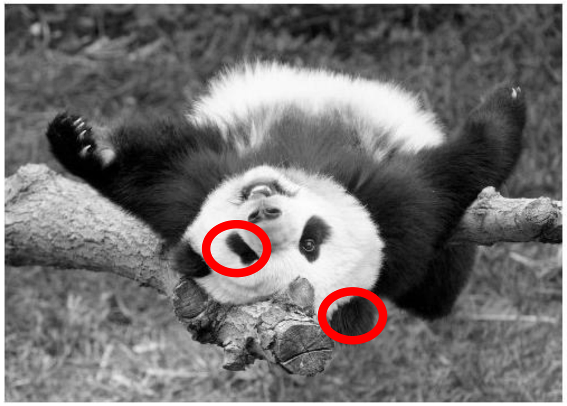
Source: K. Grauman

Segmentation as Clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **intensity+position** similarity





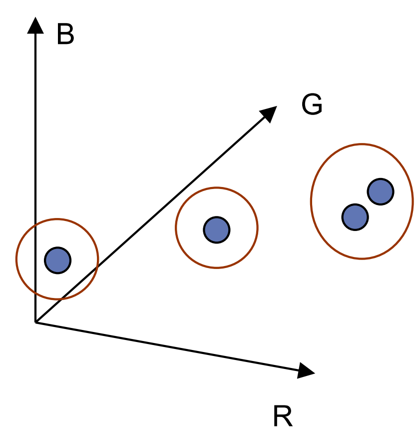
Both regions are black, but if we also include **position (x,y),** then we could group the two into distinct segments; way to encode both similarity & proximity.

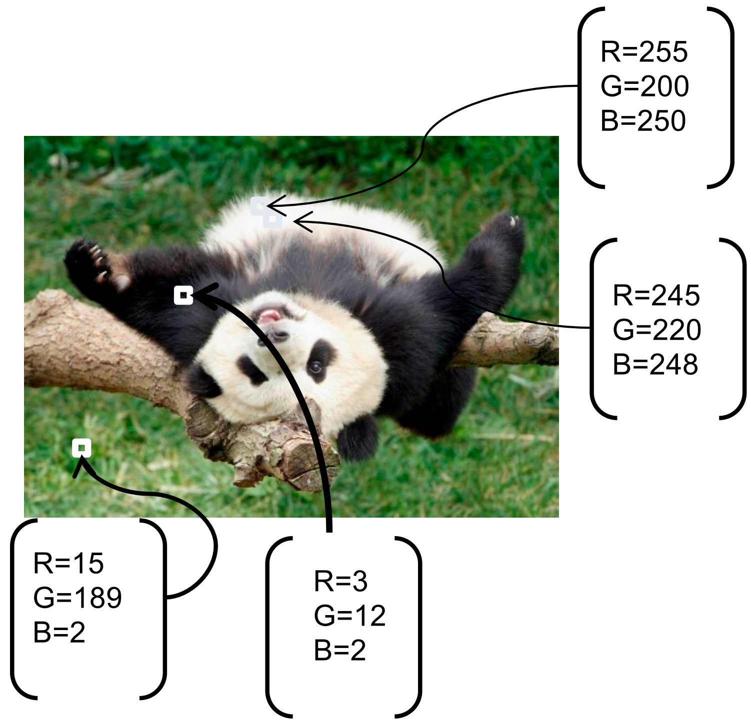
Source: K. Grauman

Segmentation as Clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **color** similarity



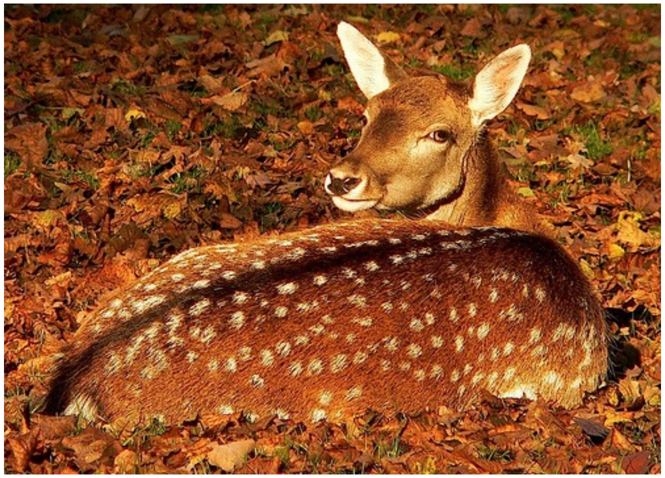


Feature space: color value (3-d)

Adapted from K. Grauman

Segmentation as Clustering

- Color, brightness, position alone are not enough to distinguish all regions…

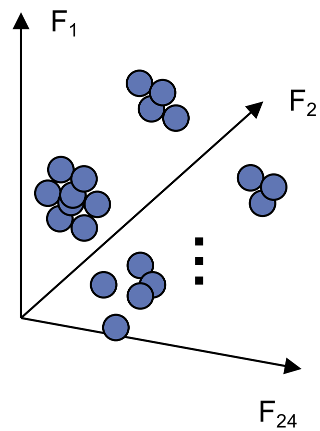


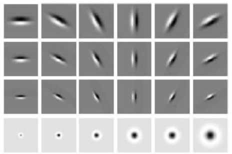
Source: L. Lazebnik

Segmentation as Clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **texture** similarity



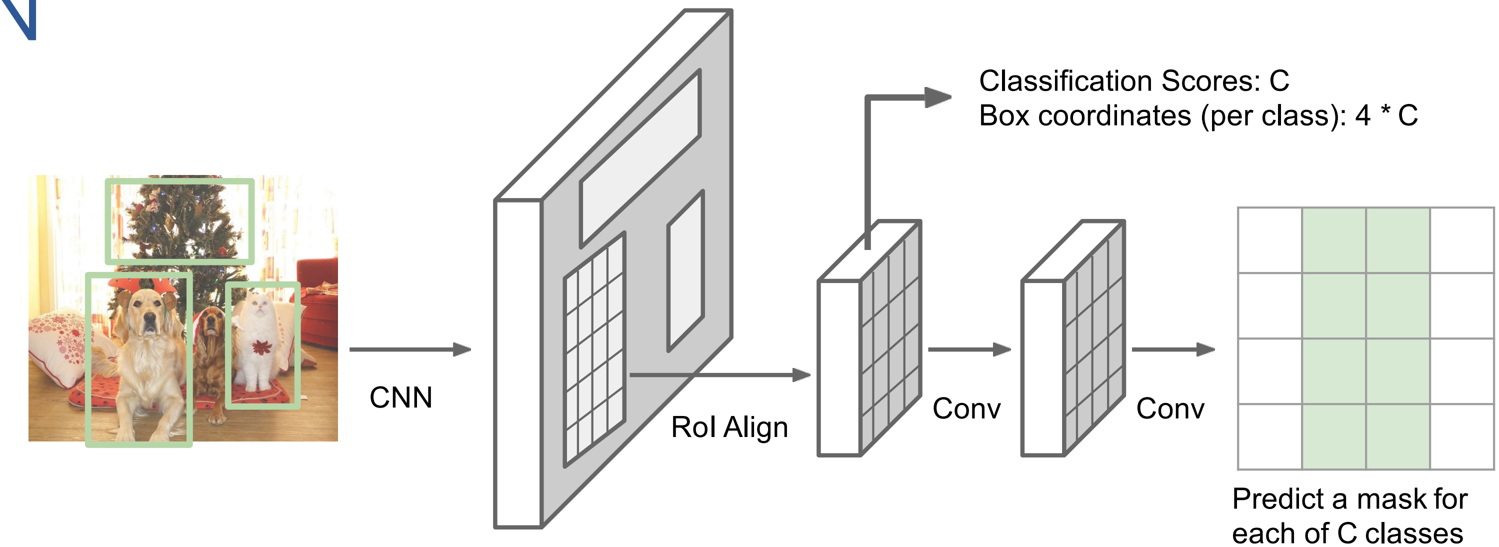


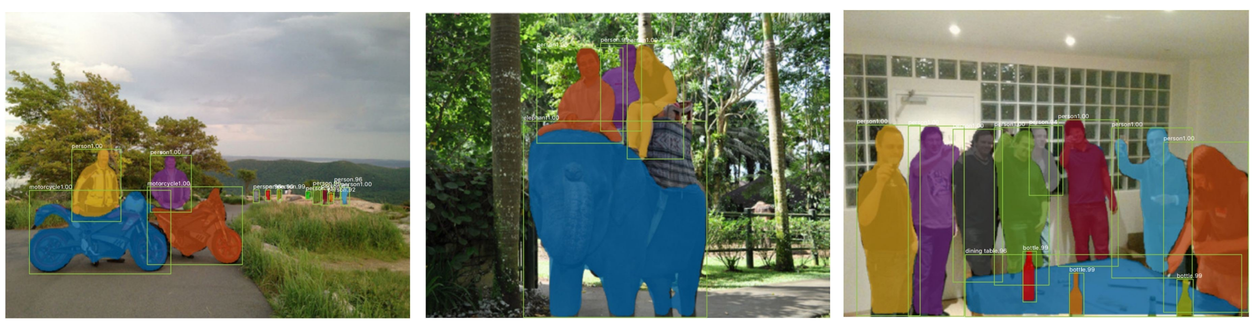
Filter bank of 24 filters

Feature space: filter bank responses (e.g., 24-d)

Source: K. Grauman

State-of-the-art (instance) segmentation: Mask R-CNN





He et al, "[Mask R-CNN]", ICCV 2017; slide adapted from Justin Johnson

Summary: classic approaches

- Edges: threshold gradient magnitude
- Lines: edge points vote for parameters of line, circle, etc. (works for general objects)
- Segments: use clustering (e.g. K-means) to group pixels by intensity, texture, etc.