CS 2770: Grouping: edges, lines, circles, and segments

PhD. Nils Murrugarra-Llerena
nem177@pitt.edu
University of Pittsburgh

Plan for this lecture

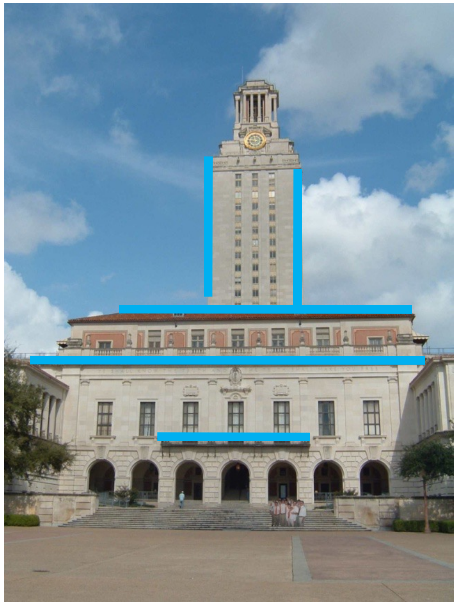
- Lines
  - Find which edge points are collinear or belong to another shape e.g. circle
  - Automatically detect and ignore outliers

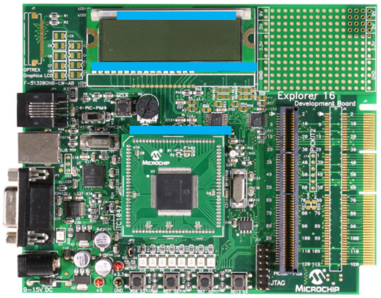
- Segments
  - Find which pixels form a consistent region
  - Clustering (e.g. K-means)

Line detection (fitting)

- Why fit lines?

  Many objects characterized by presence of straight lines

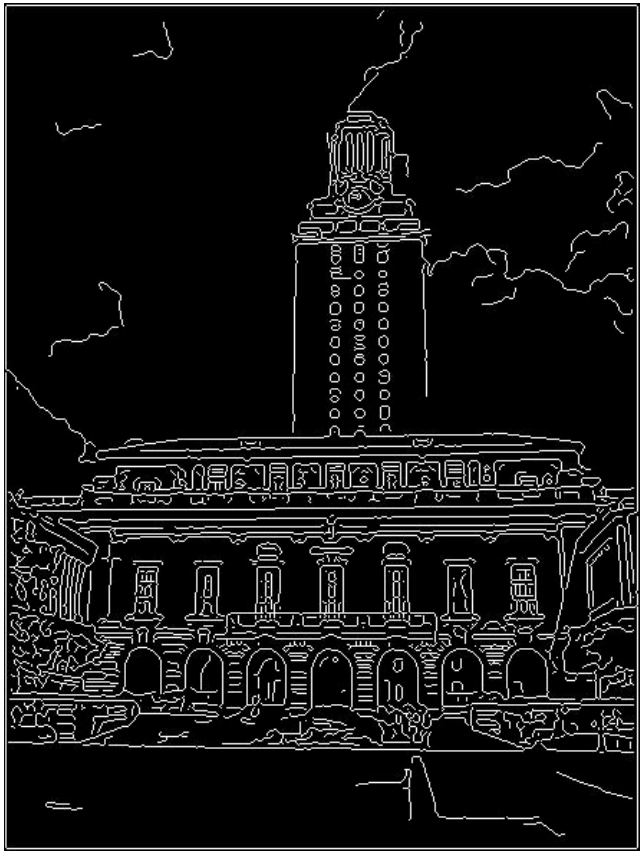




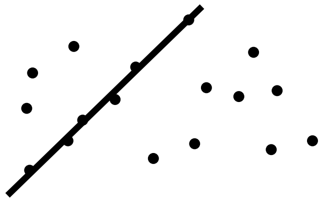
- Why aren’t we done just by running edge detection?

Kristen Grauman

Difficulty of Line Fitting



Adapted from Kristen Grauman

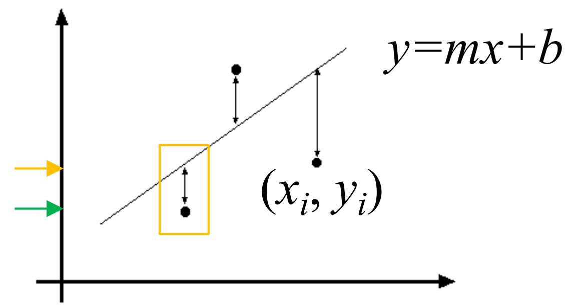


- **Noise** in measured edge points, orientations:
  - e.g. edges not collinear where they should be
  - how to detect true underlying parameters?
- **Extra** edge points (clutter):
  - which points go with which line, if any?
- Only some parts of each line detected, and some parts are **missing:**
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

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



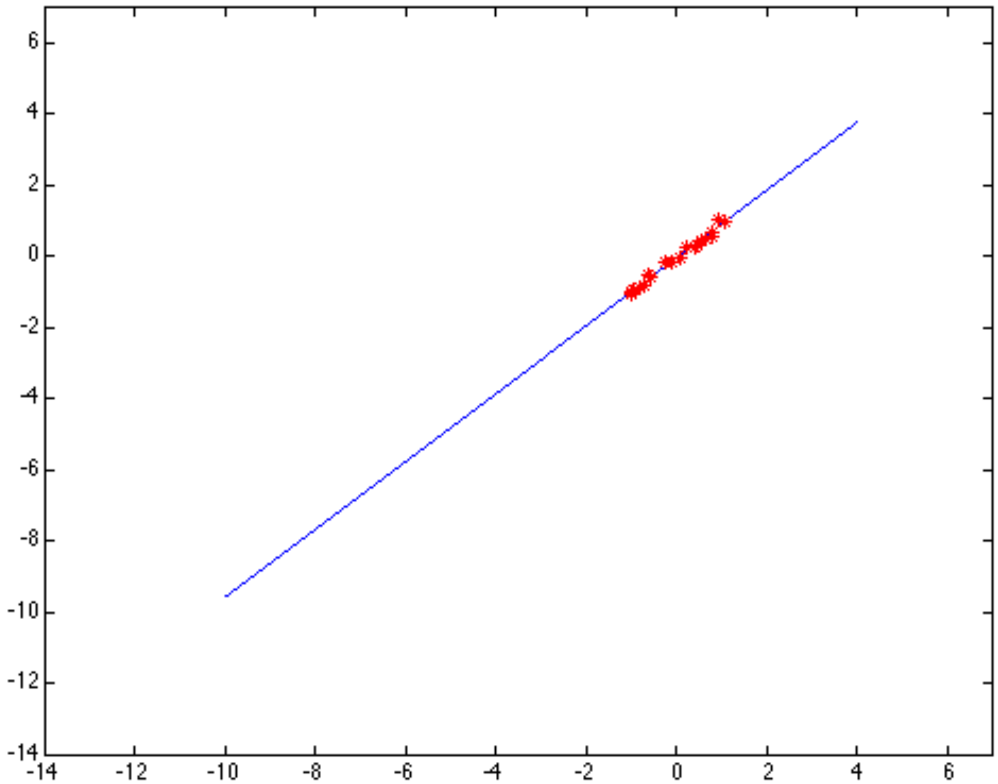
You want to find a single line that “explains” all of the points in your data, but data may be noisy!

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

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

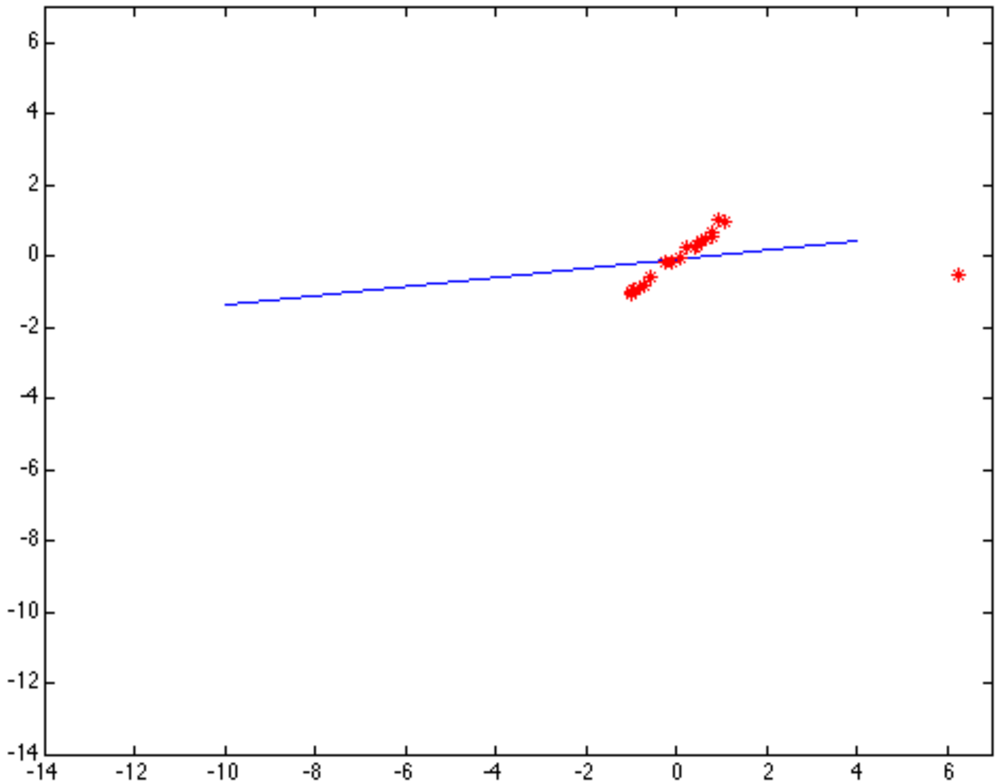
Adapted from Svetlana Lazebnik

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

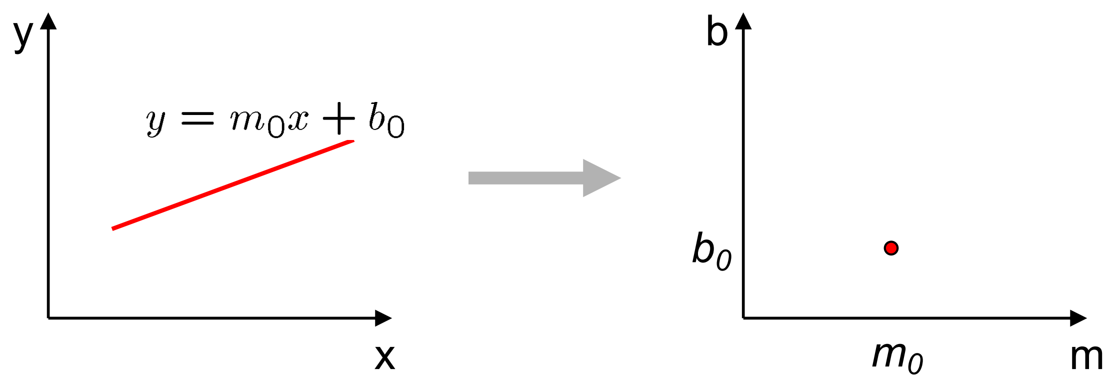


image space

Hough (parameter) 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

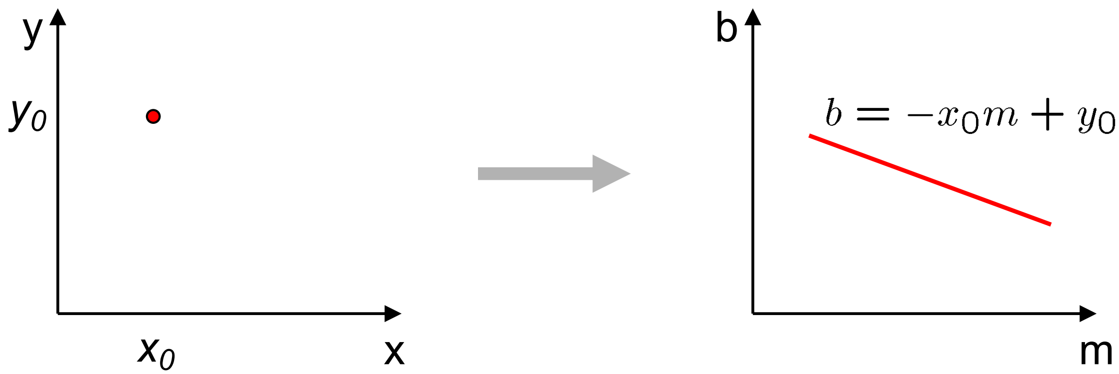


image space

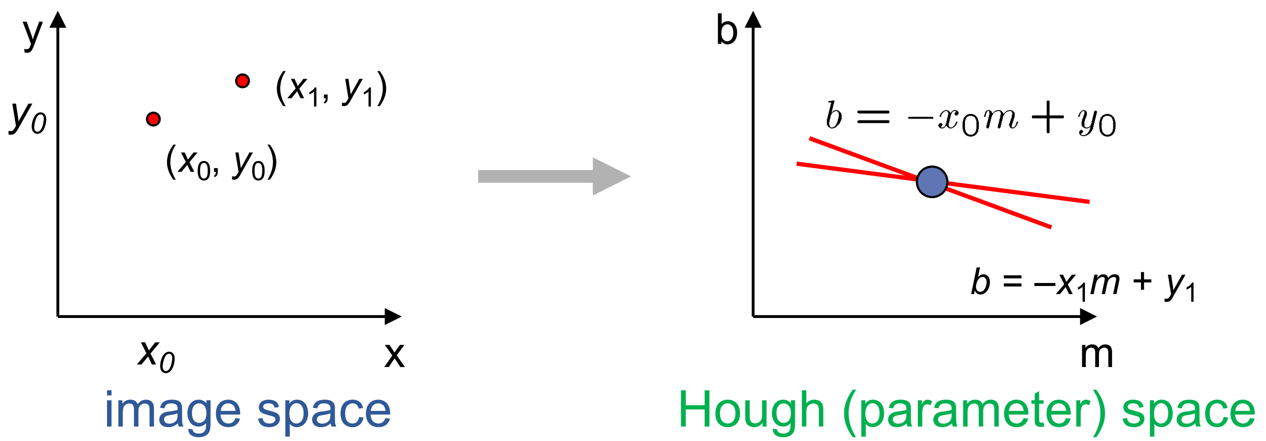
Hough (parameter) space

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

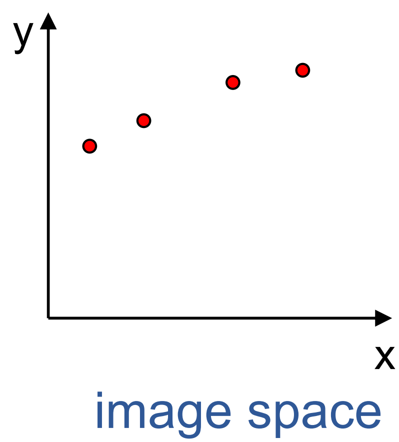


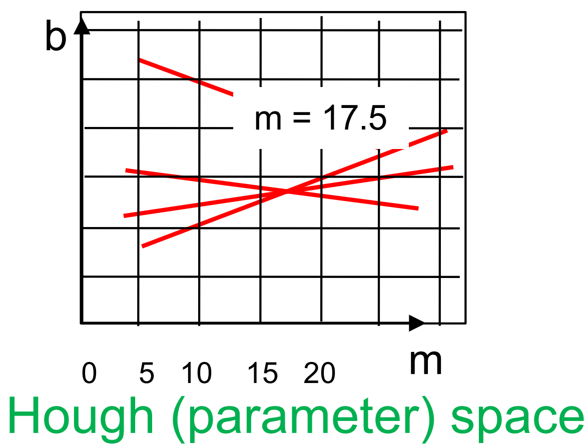
What are the line parameters for the line that contains both (x_0, y_0) and (x_1, y_1)?

- 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



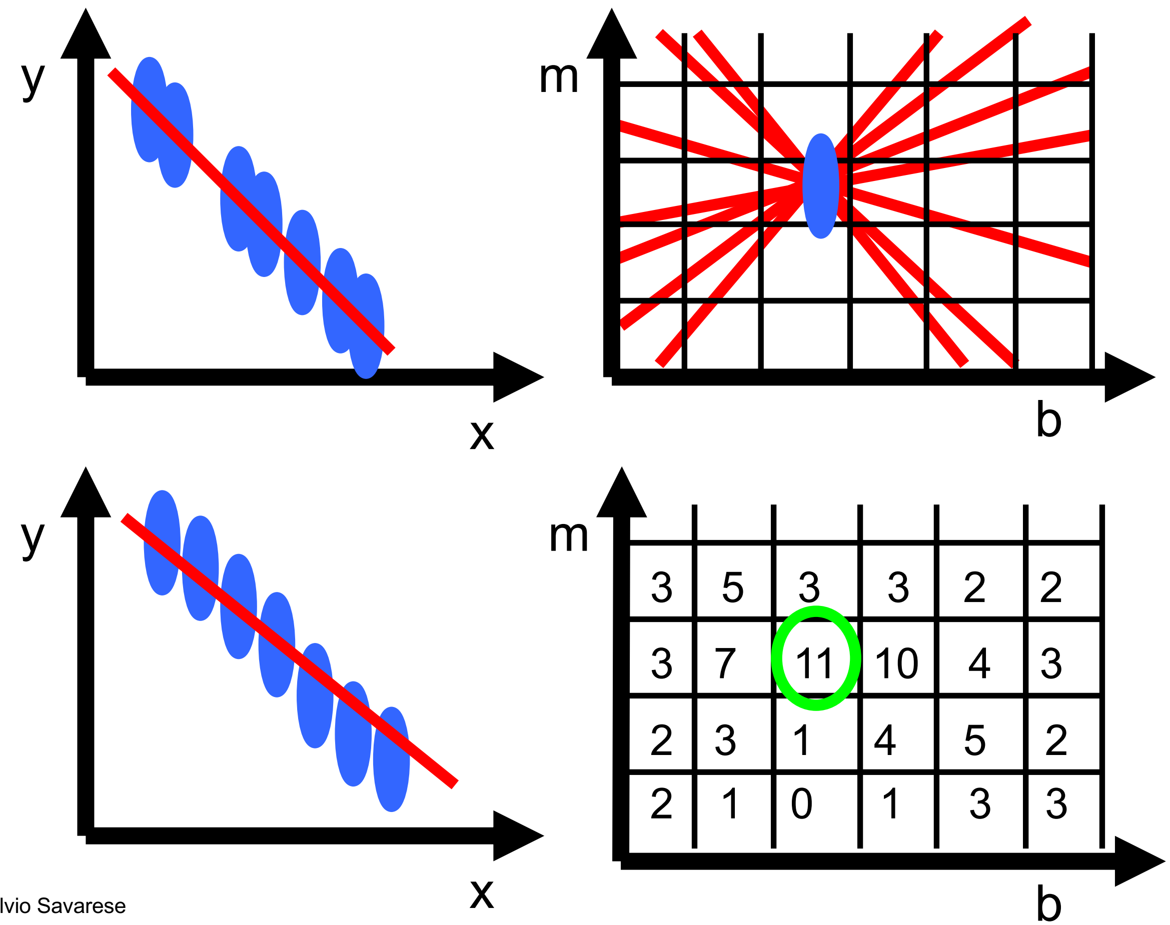


How can we use this to find the most likely parameters (m,b) for the most prominent line in the image space?

- 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

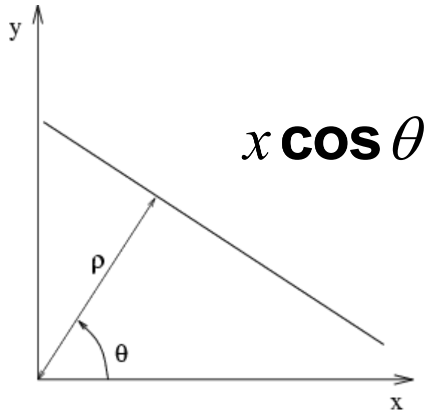
Finding lines in an image: Hough space



Adapted from Silvio Savarese

Parameter space representation

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



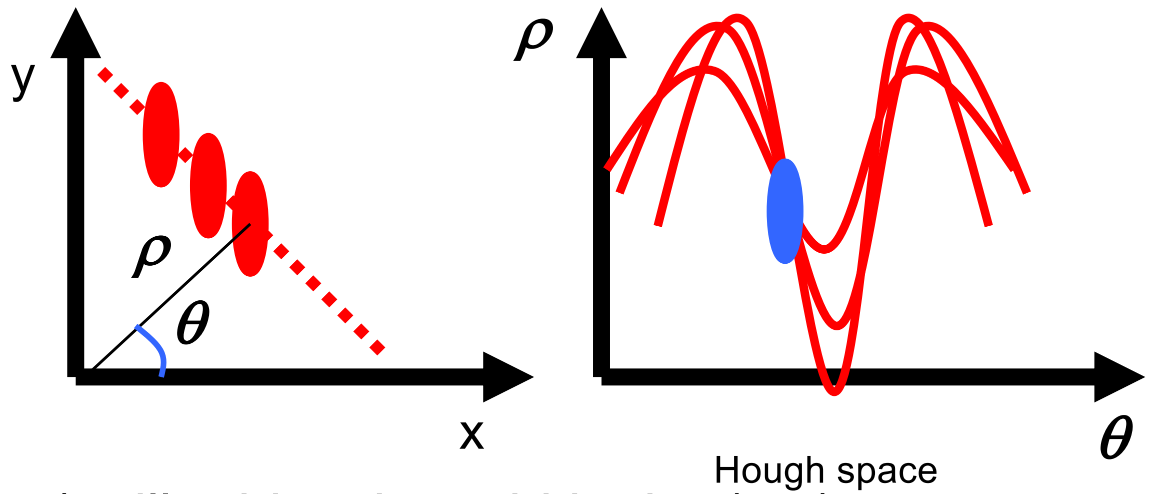
$$x\cos\theta + y\sin\theta = \rho$$

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*

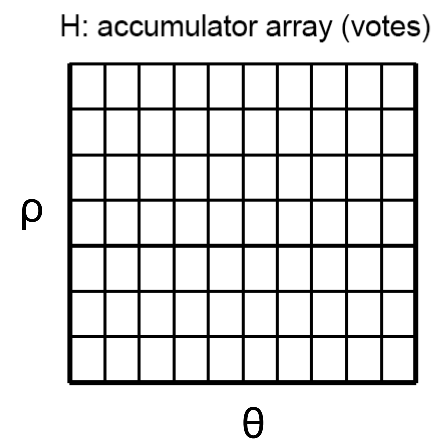


Each point (x,y) will add a sinusoid in the (θ,ρ) parameter space

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

  ρ = x cos θ + y sin θ

  H(θ, ρ) = H(θ, ρ) + 1

  end

  end

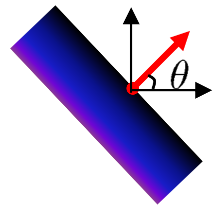
Why only until 180 degrees?

- Find the value(s) of (θ*, ρ*) where H(θ, ρ) is a local maximum
  - The detected line in the image is given by

    ρ* = x cos θ* + y sin θ*

Svetlana Lazebnik

Incorporating Image Gradients



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

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

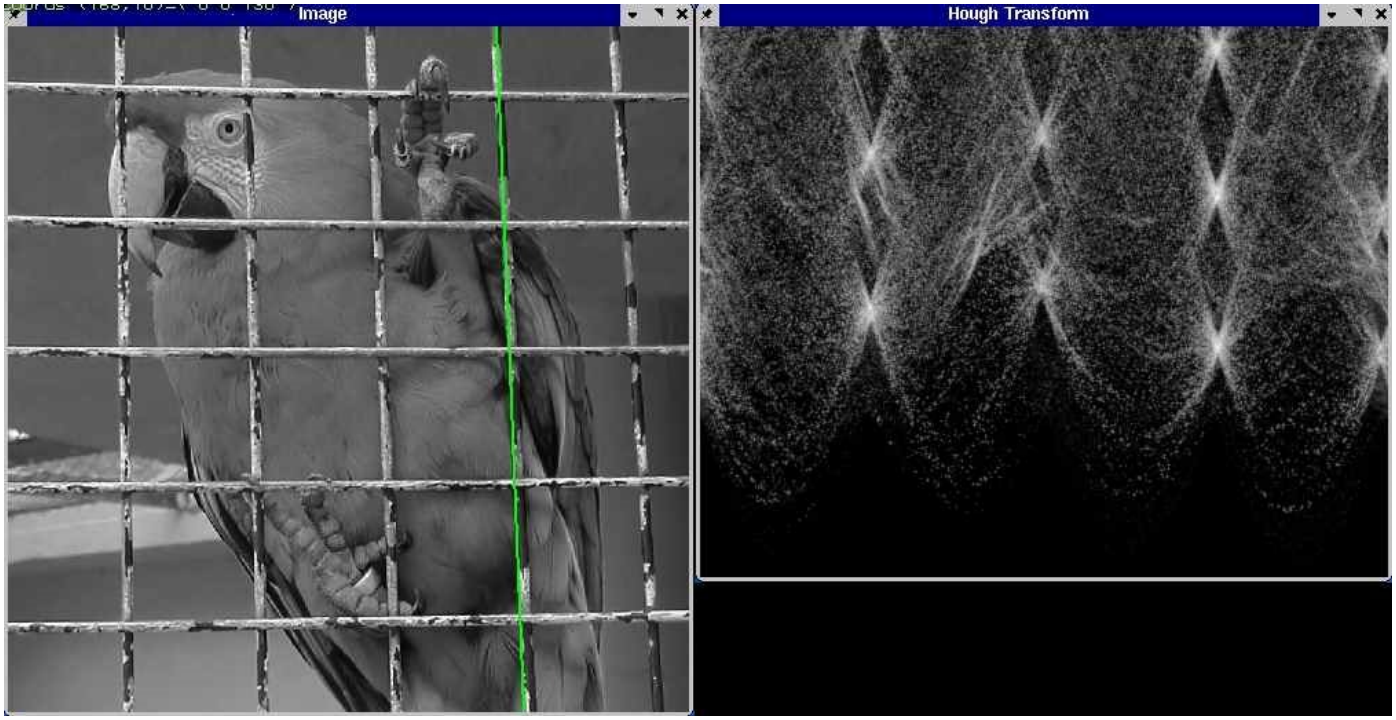
- Recall: when we detect an edge point, we also know its gradient direction
- But this means that the line is uniquely determined!

- 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
```

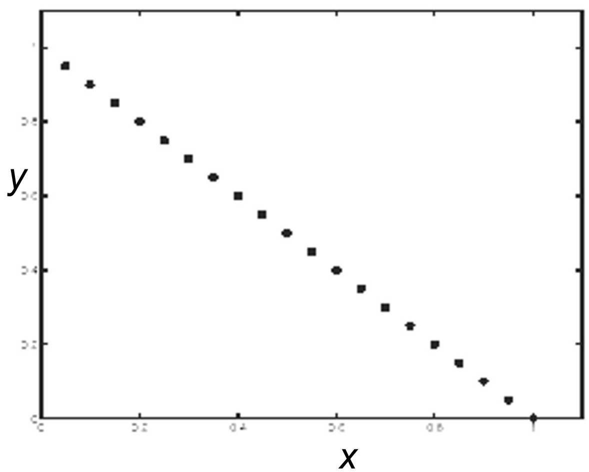
Svetlana Lazebnik

Hough transform example



Derek Hoiem

Impact of noise on Hough



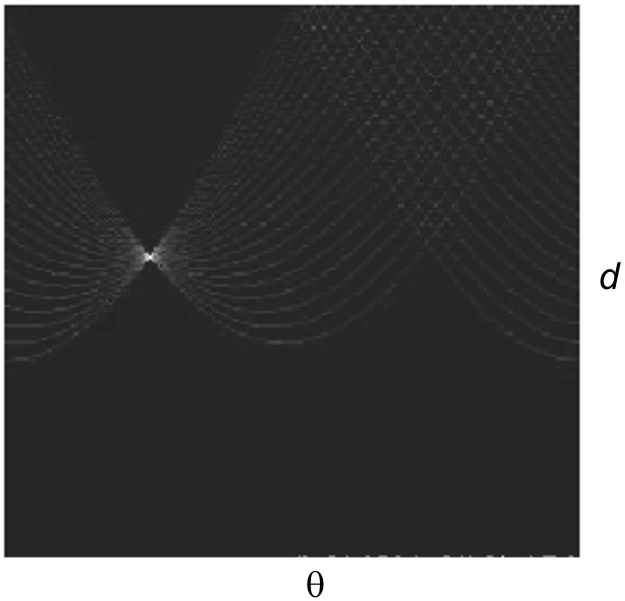
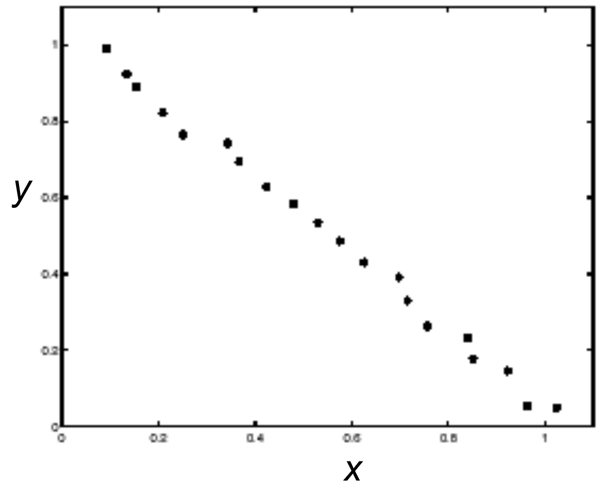


Image space edge coordinates

Votes

Kristen Grauman

Impact of noise on Hough



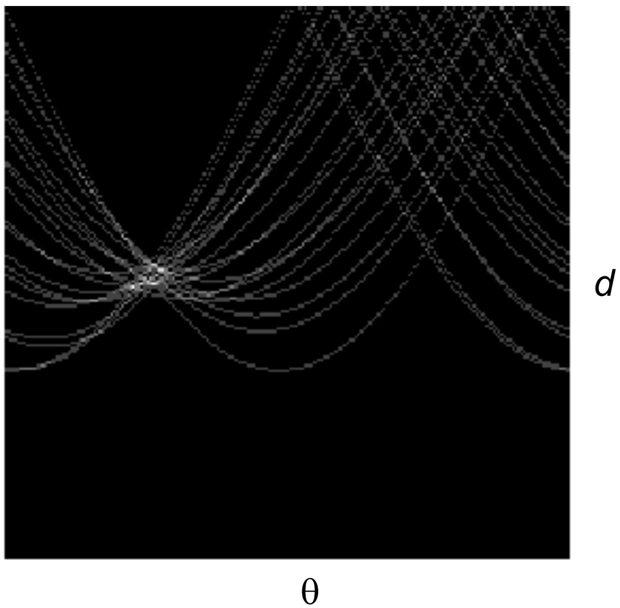


Image space edge coordinates

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

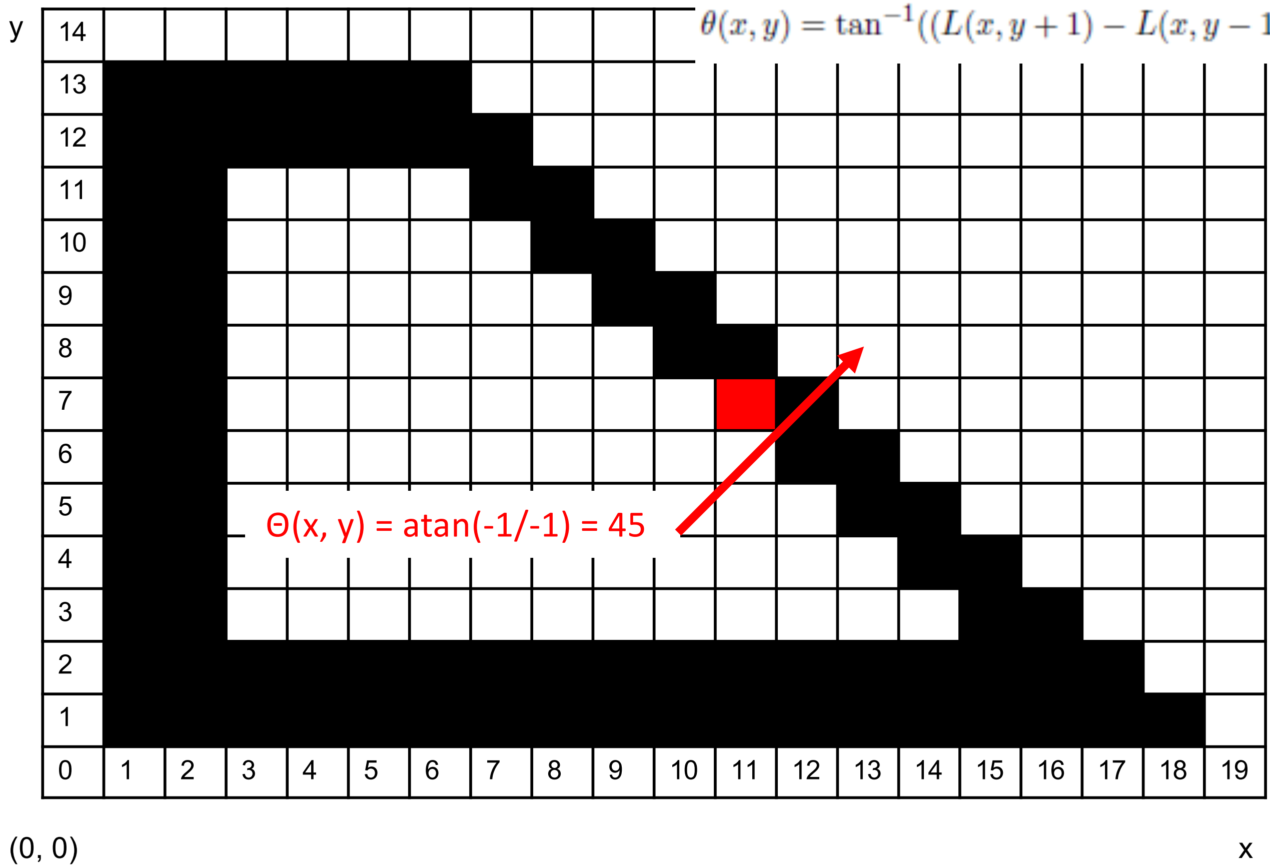
  Too coarse ? Too fine

  - **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)

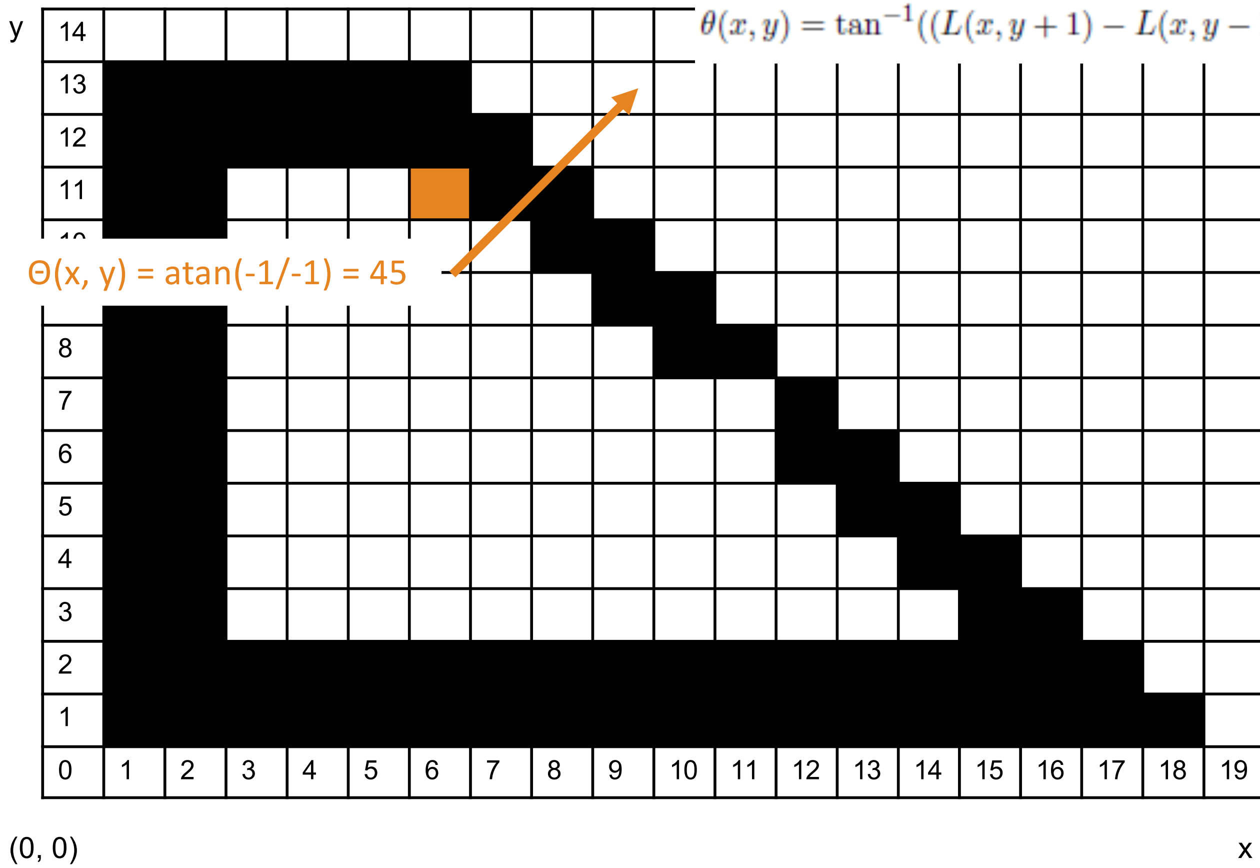
ρ = 11 cos(45) + 7 sin(45)

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)

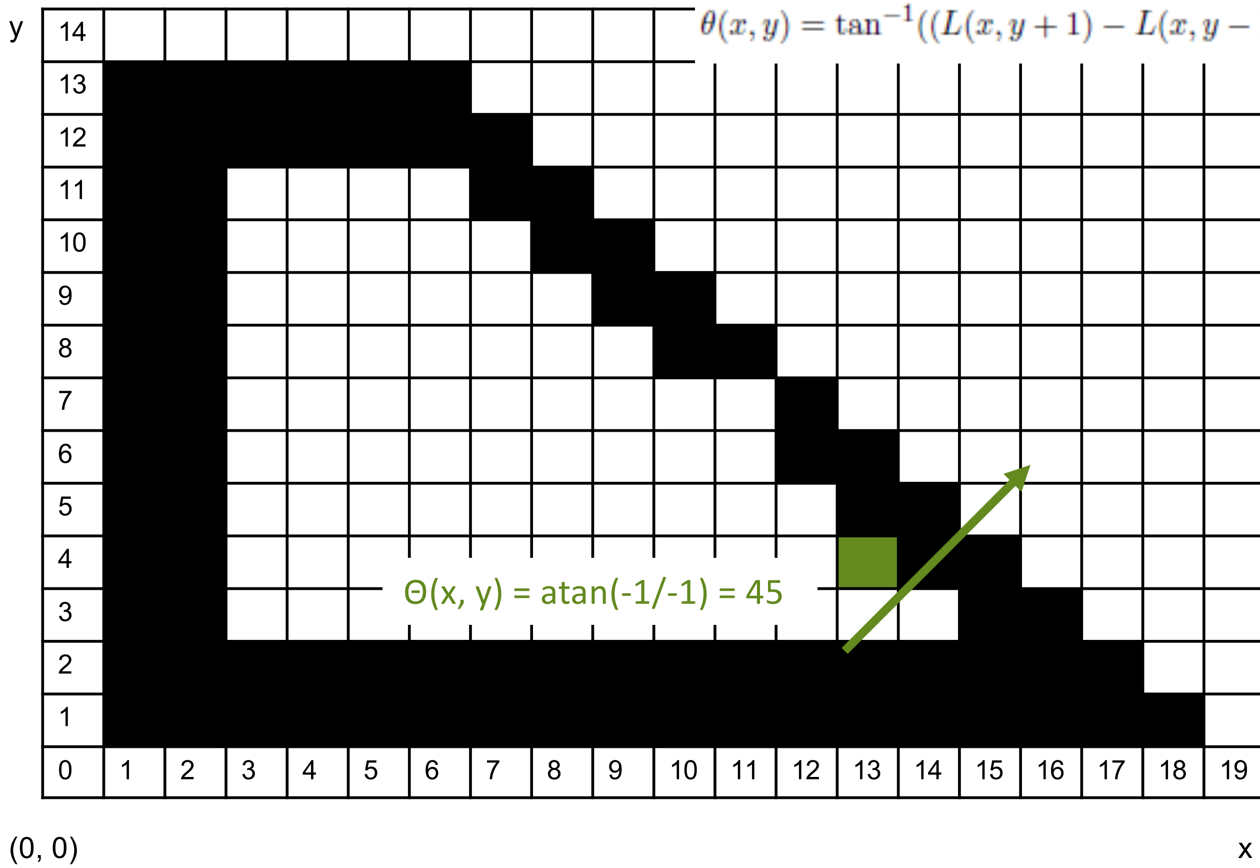
ρ = 6 cos(45) + 11 sin(45)

ρ = 12.02

H(θ=45, ρ= 12.02) = 1

Hough Transform: Example

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



ρ = x cos(Θ) + y sin(Θ)

ρ = 13 cos(45) + 4 sin(45)

ρ = 12.02

H(θ=45, ρ= 12.02) = 2

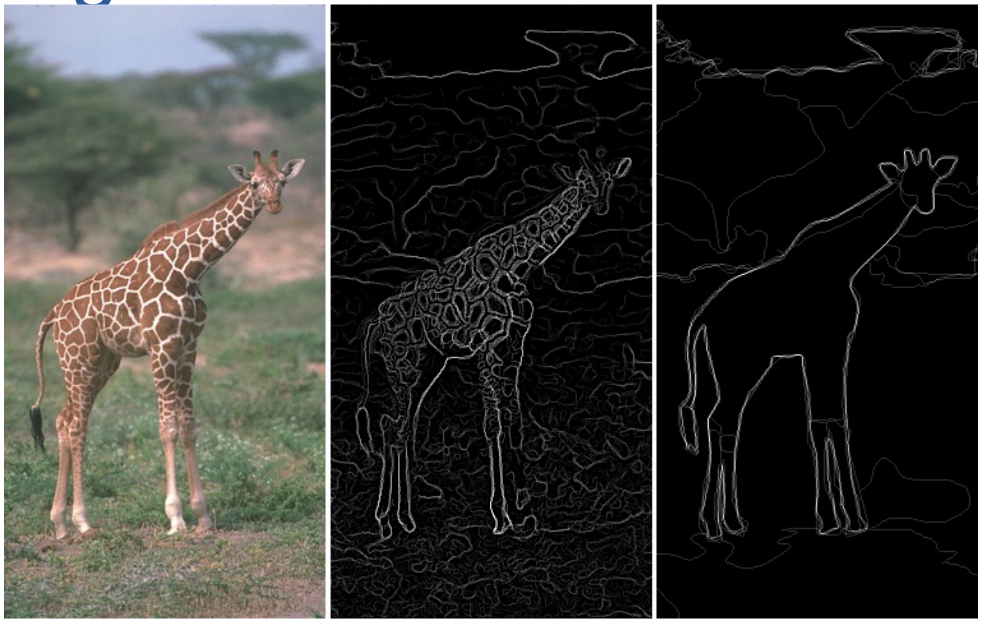
H(θ=45, ρ=12.73) = 1
H(θ=45, ρ= 12.02) = 2

Plan for today

- Lines
  - 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



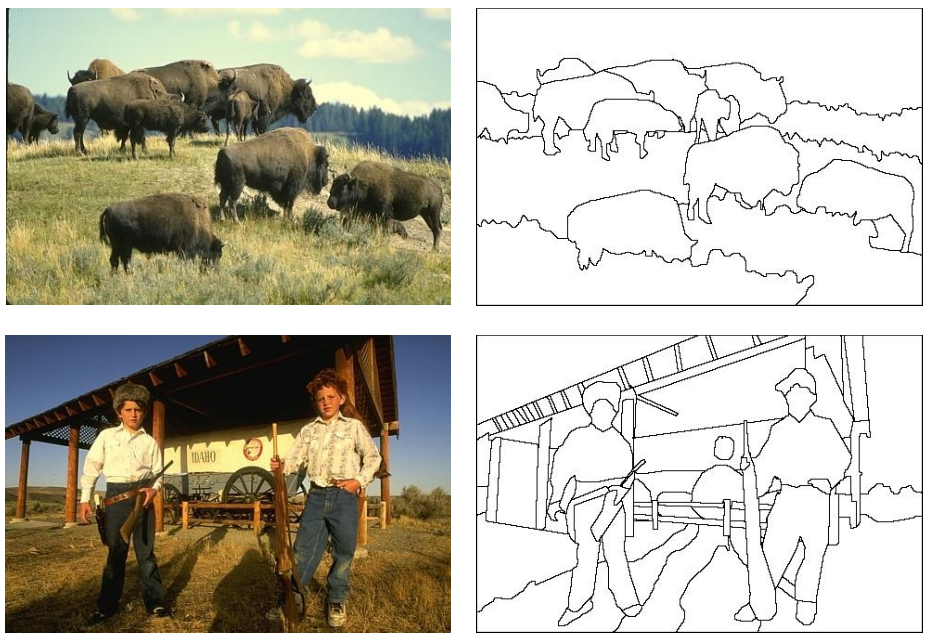
- Edges: More low-level; don’t need to be closed
- Segments: Ideally one segment for each semantic group/object; should include closed contours

Figure adapted from J. Hays

The goals of segmentation

- Separate image into coherent “objects”

image human segmentation

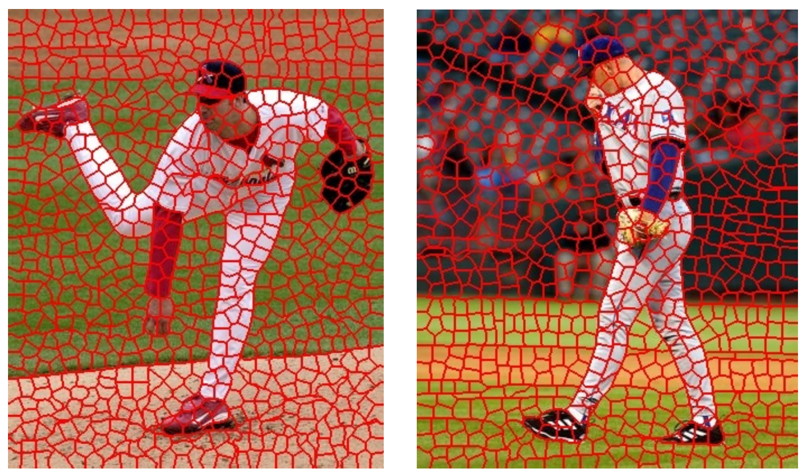


Source: L. Lazebnik

The goals of segmentation

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

“superpixels”



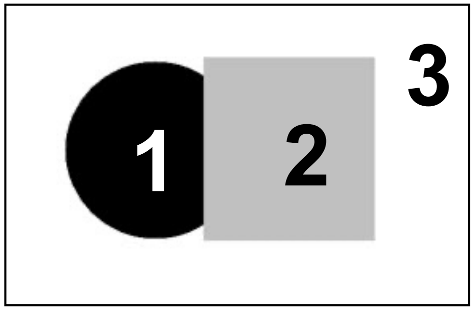
X. Ren and J. Malik. [Learning a classification model for segmentation](). ICCV 2003.

Source: L. Lazebnik

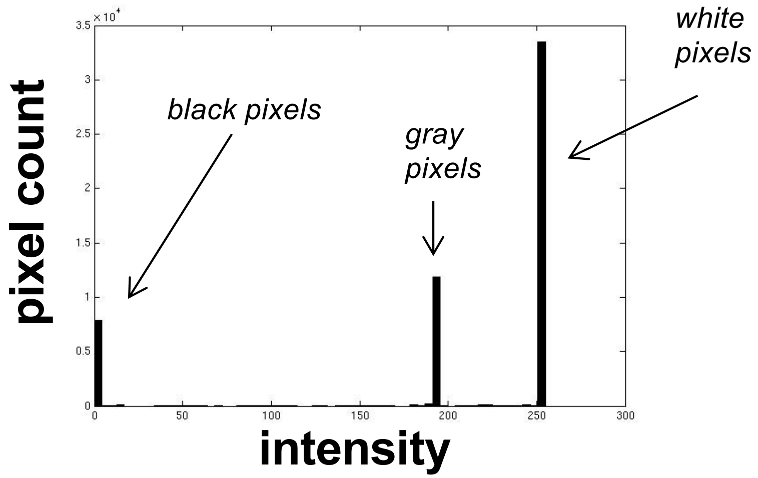
Similarity

Slide: K. Grauman

Image Segmentation: Toy Example



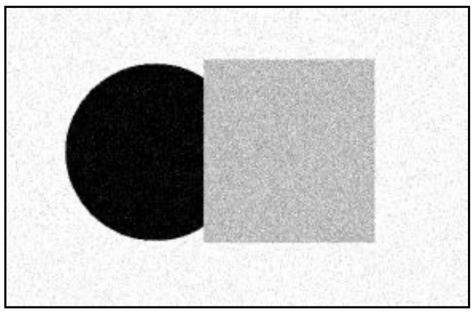
input image



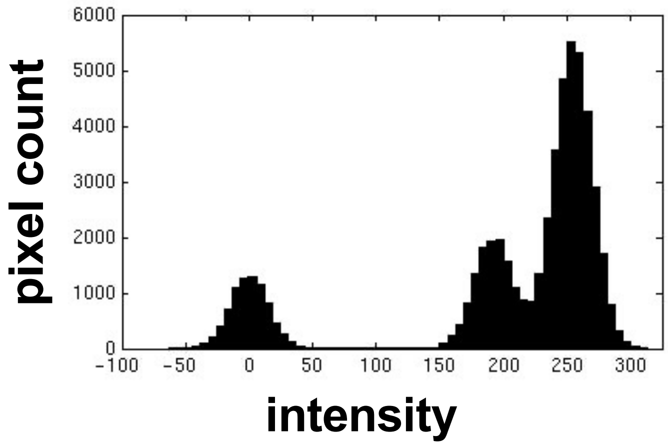
- These intensities define the three groups.
- We could label every pixel in the image according to which of these primary intensities it is.
  - i.e., *segment* the image based on the intensity feature.
- What if the image isn't quite so simple?

Source: K. Grauman

Image Segmentation: Toy Example



input image

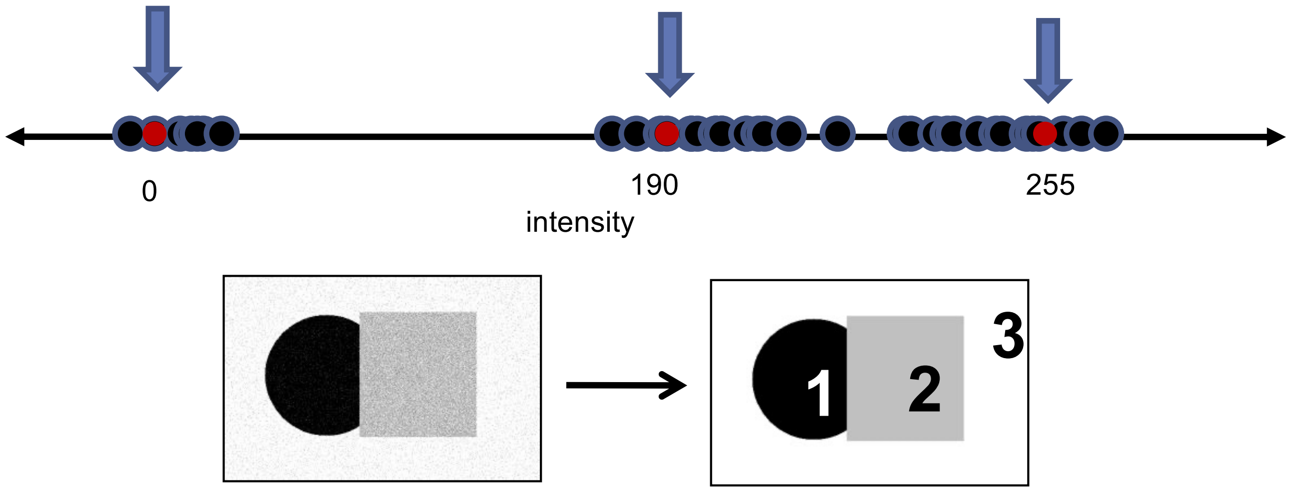


- Now how to determine the three main intensities that define our groups?

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

Source: K. Grauman

Image Segmentation: Toy Example



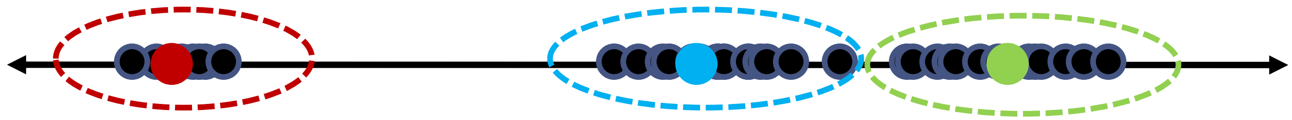
- Goal: choose three "centers" as the representative intensities, and label every pixel according to which of these centers it is nearest to.
- Best cluster centers are those that minimize *sum of squared differences* (SSD) between all points and their nearest cluster center c_i:

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

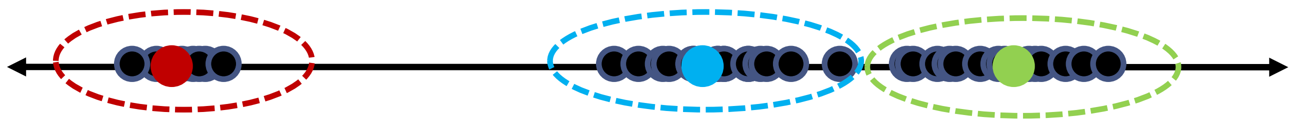
Source: K. Grauman

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

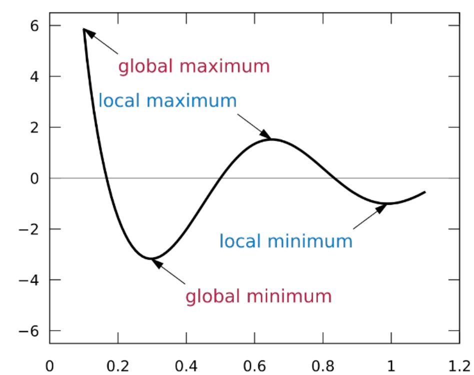
- **Basic idea**: randomly initialize the *k* cluster centers, and iterate between the two steps we just saw.

1. *Randomly* initialize the cluster centers, $c_1, \ldots, c_K$
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:

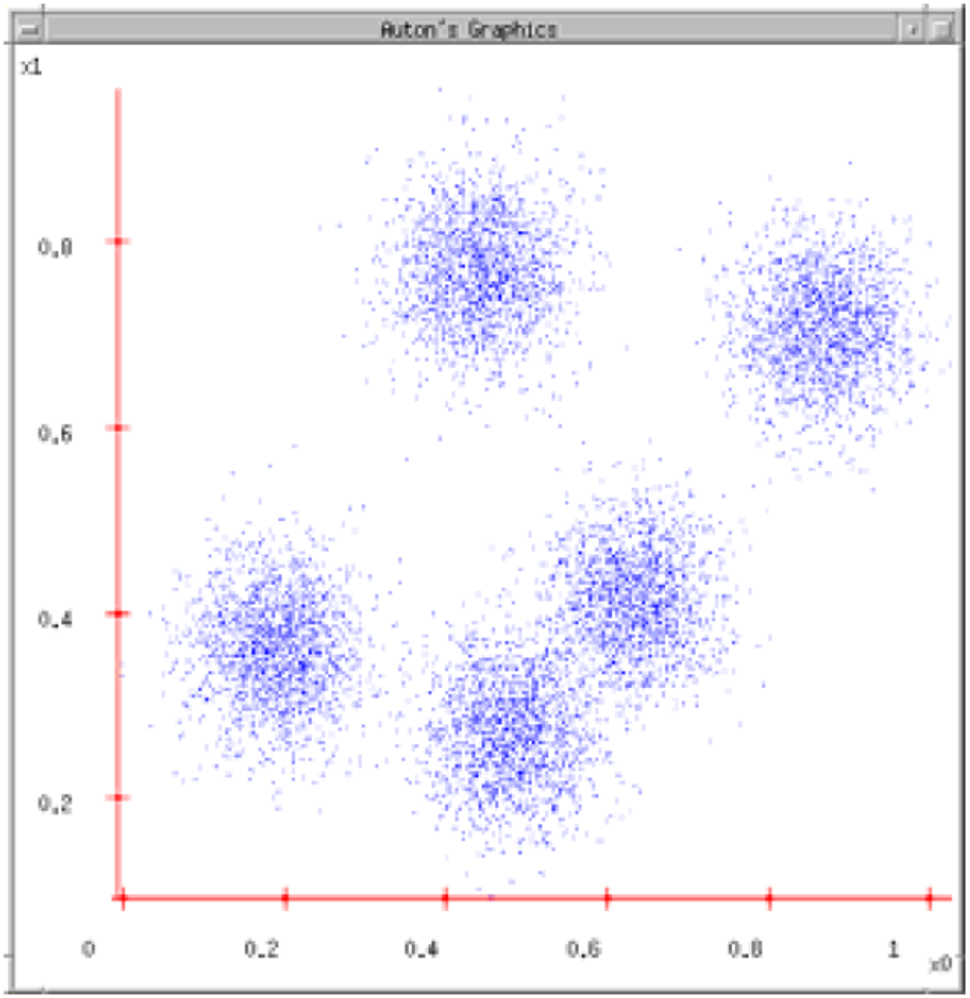
$$\sum_{\text{clusters } i} \sum_{\text{points p in cluster } i} ||p - c_i||^2$$



Slide: Steve Seitz, image: Wikipedia

K-means

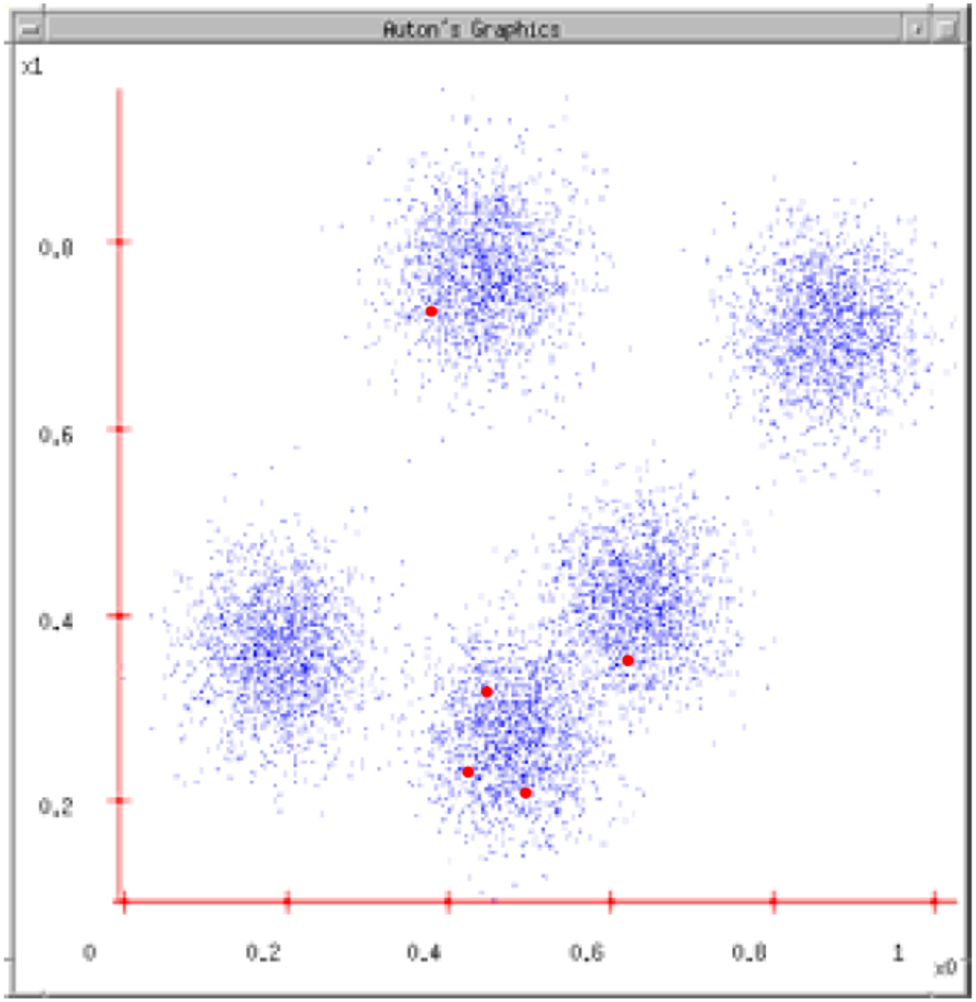
1. Ask user how many clusters they'd like. *(e.g. k=5)*



Source: A. Moore

K-means

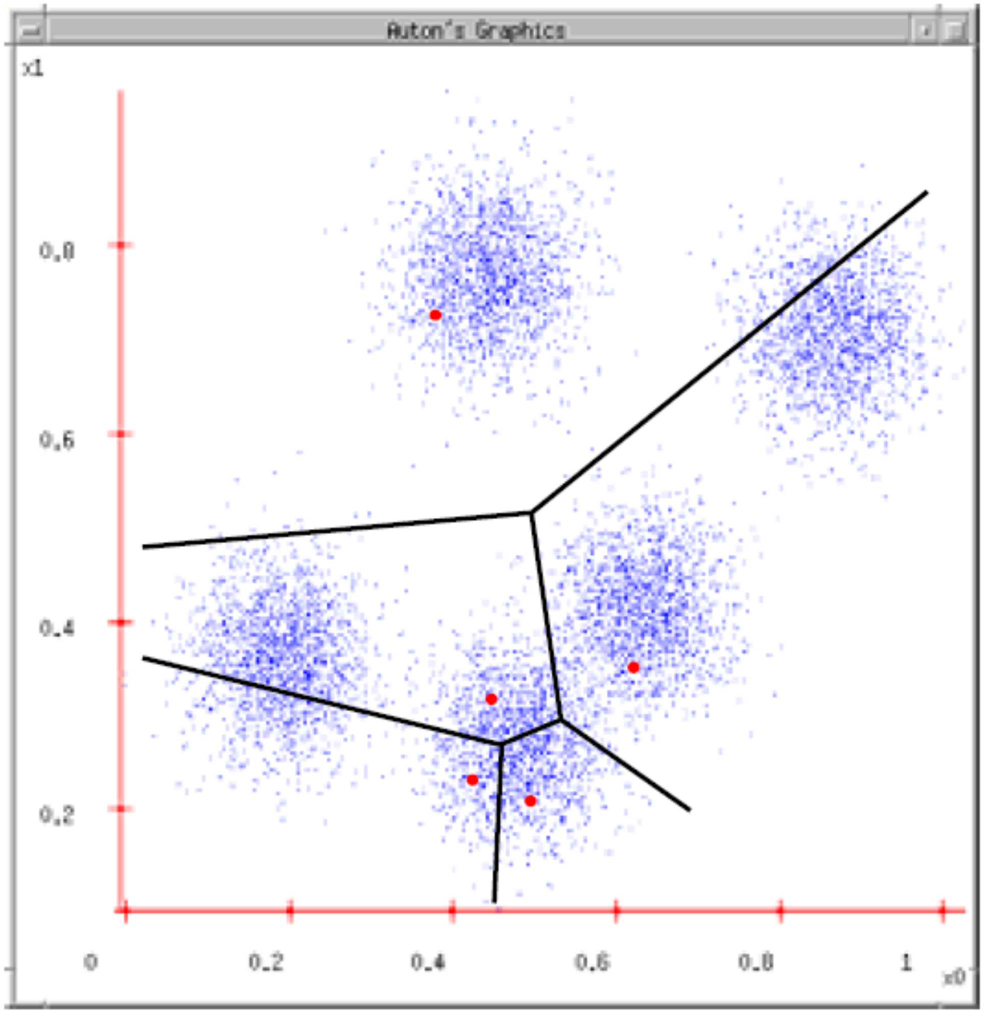
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations



Source: A. Moore

K-means

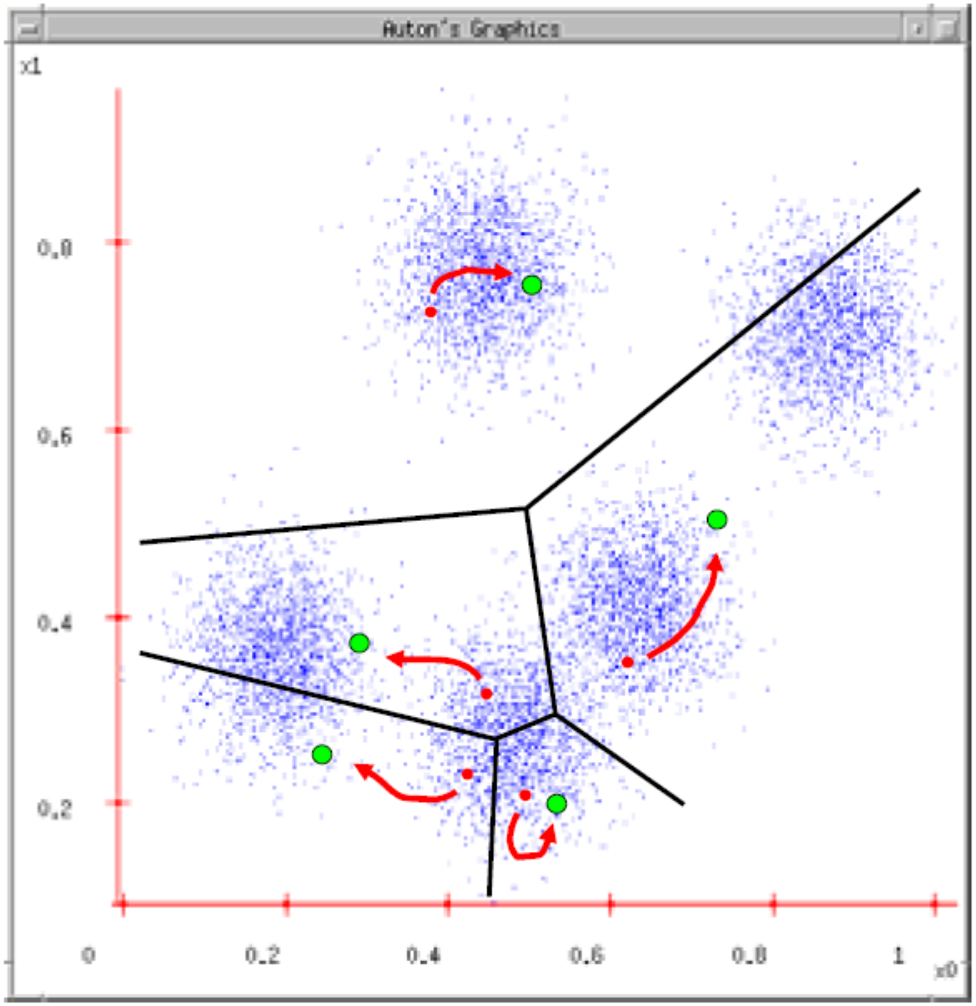
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



Source: A. Moore

K-means

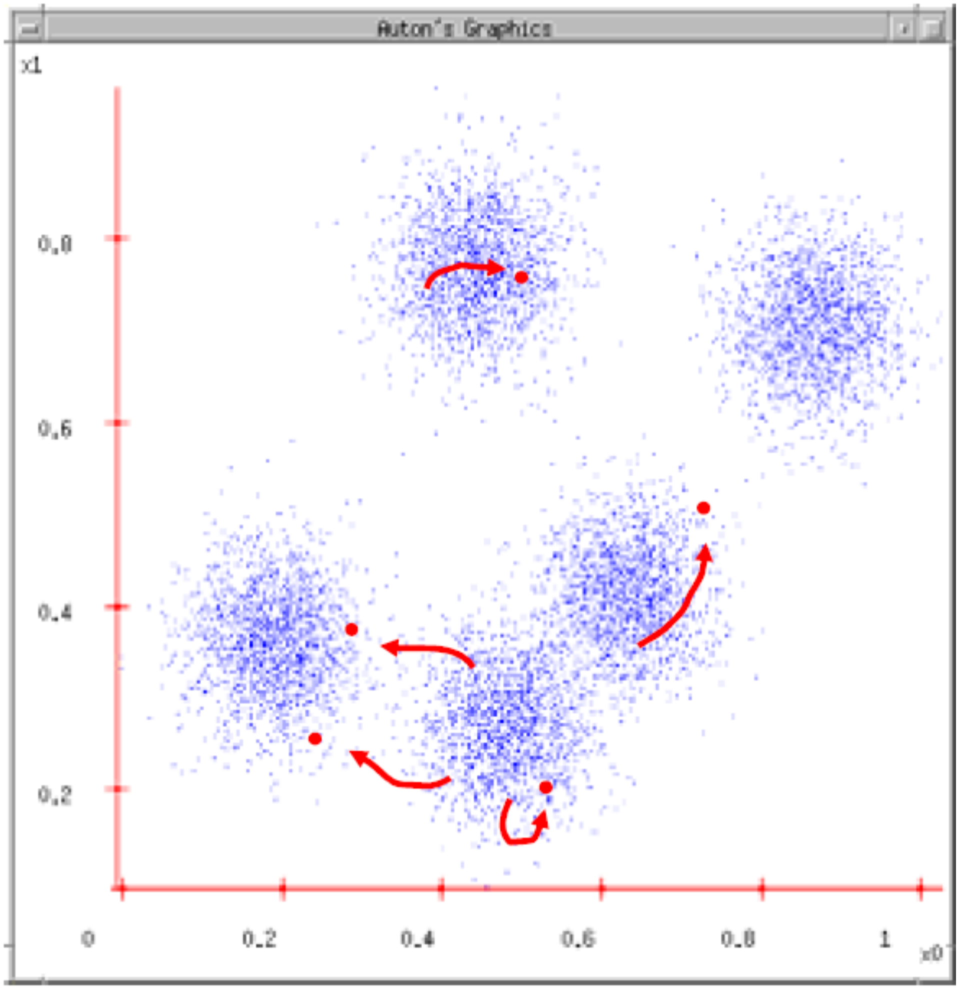
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



Source: A. Moore

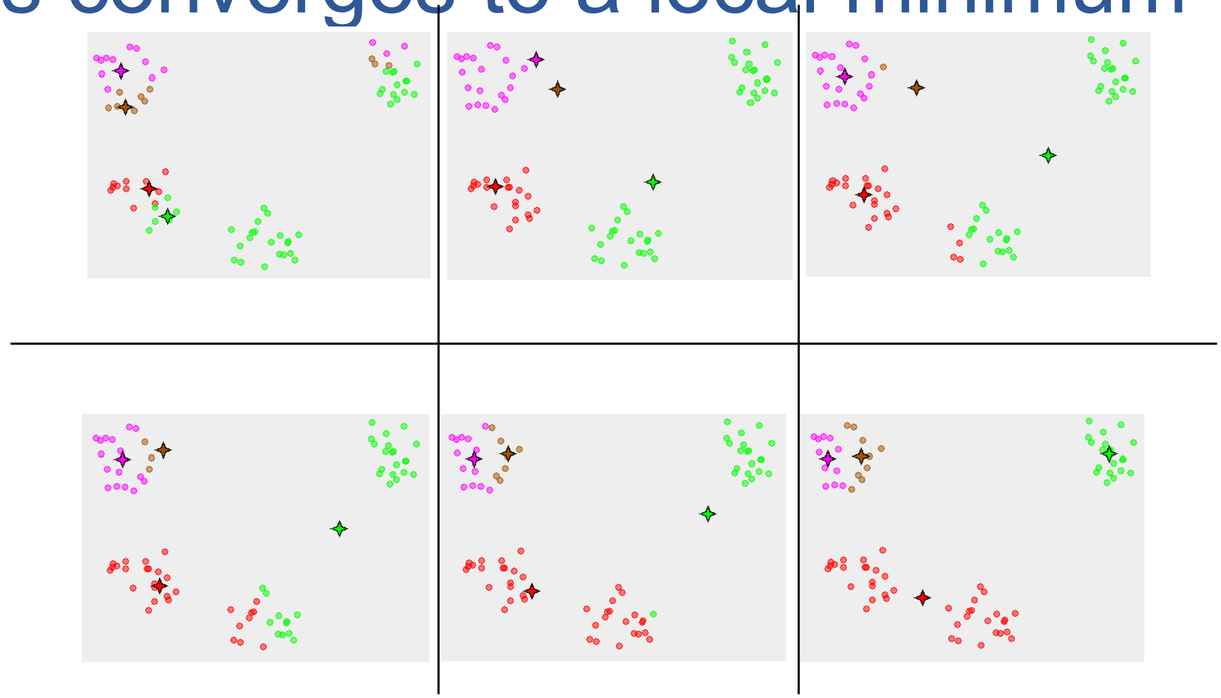
K-means

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

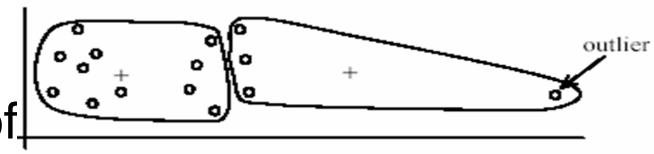
K-means: pros and cons

Pros

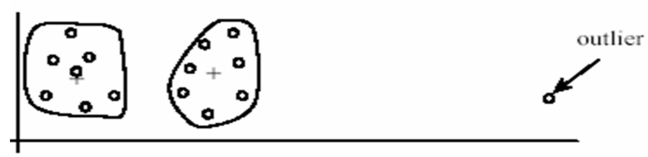
- Simple, fast to compute
- Converges to local minimum of within-cluster squared error

Cons/issues

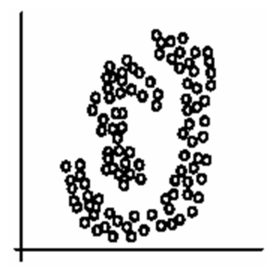
- Setting k?
  - One way: silhouette coefficient
- Sensitive to initial centers
  - Use heuristics or output of another method
- Sensitive to outliers
- Detects spherical clusters



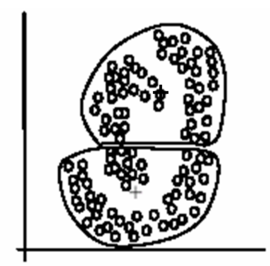
(A): Undesirable clusters



(B): Ideal clusters



(A): Two natural clusters



(B): *k*-means clusters

Adapted from K. Grauman

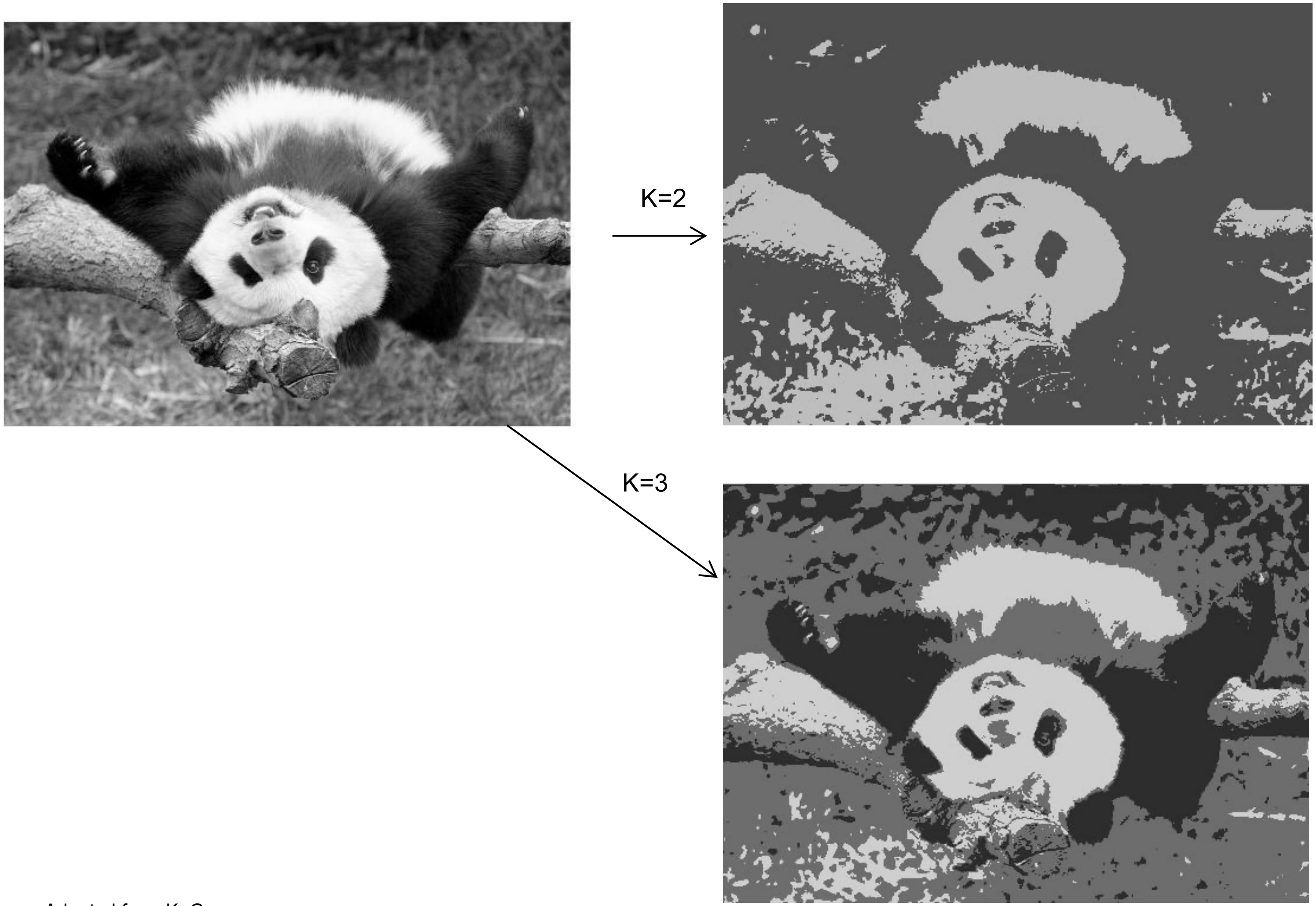
Segmentation as Clustering

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

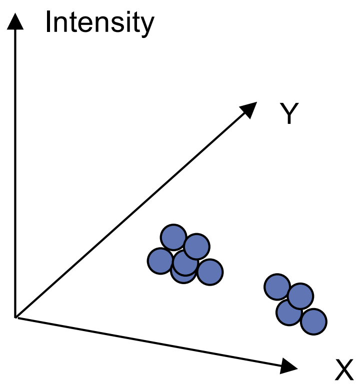
Clusters based on intensity similarity don’t have to be spatially coherent.

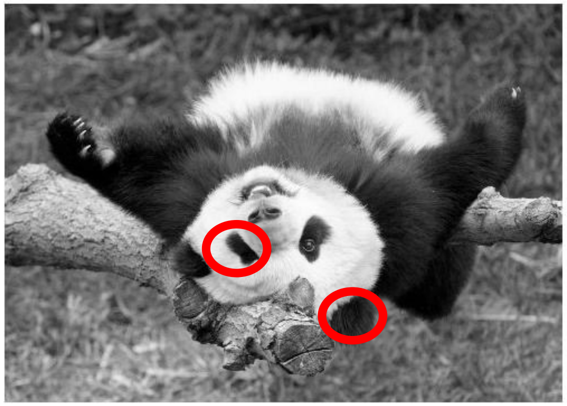
Source: K. Grauman

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





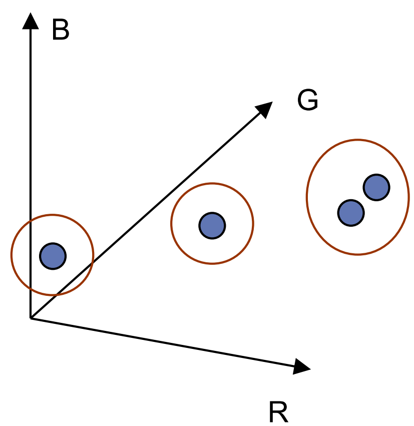
Both regions are black, but if we also include **position (x,y),** then we could group the two into distinct segments; way to encode both similarity & proximity.

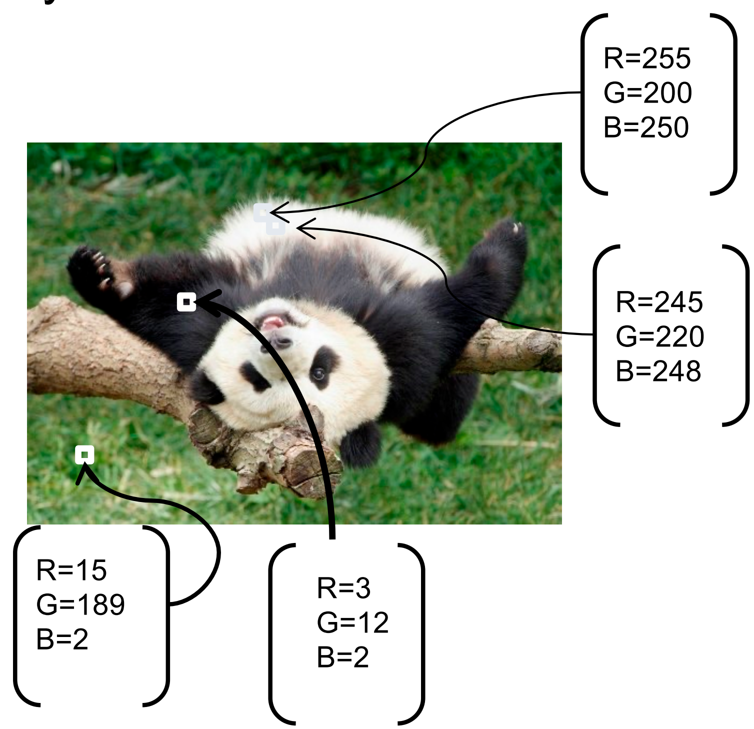
Source: K. Grauman

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



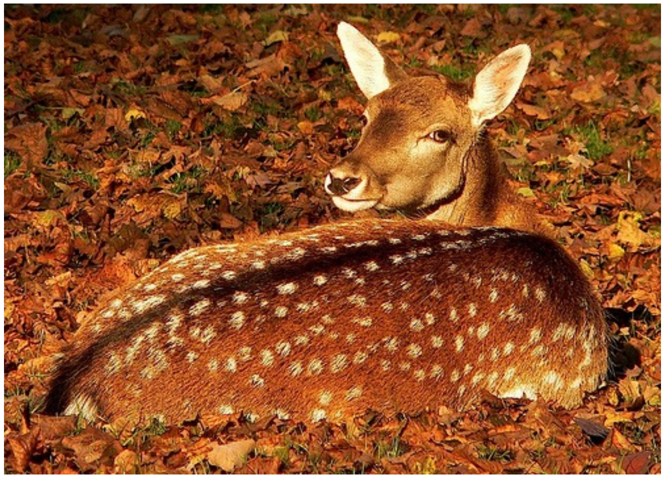


Feature space: color value (3-d)

Adapted from K. Grauman

Segmentation as Clustering

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

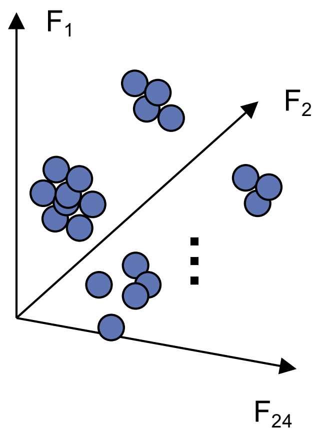


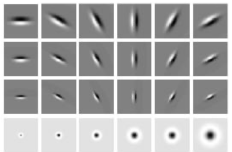
Source: L. Lazebnik

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



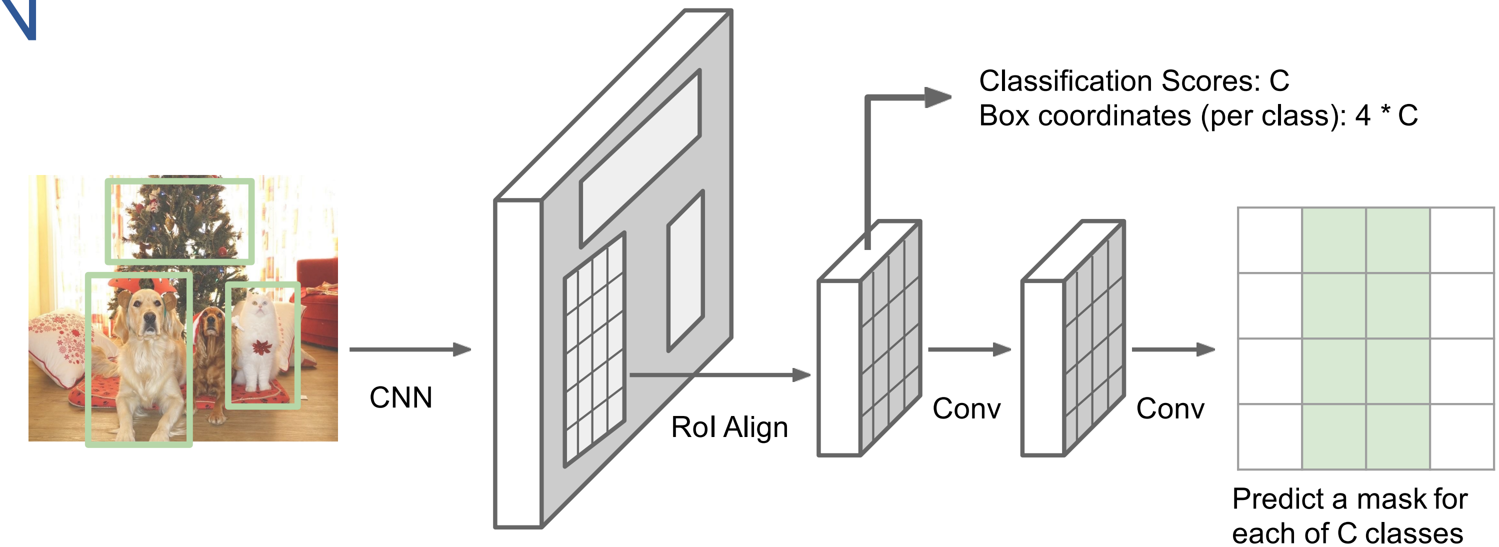


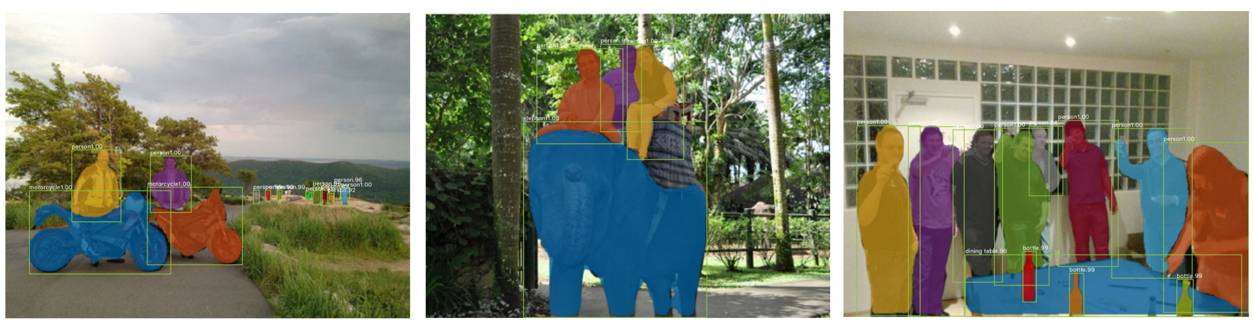
Filter bank of 24 filters

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.