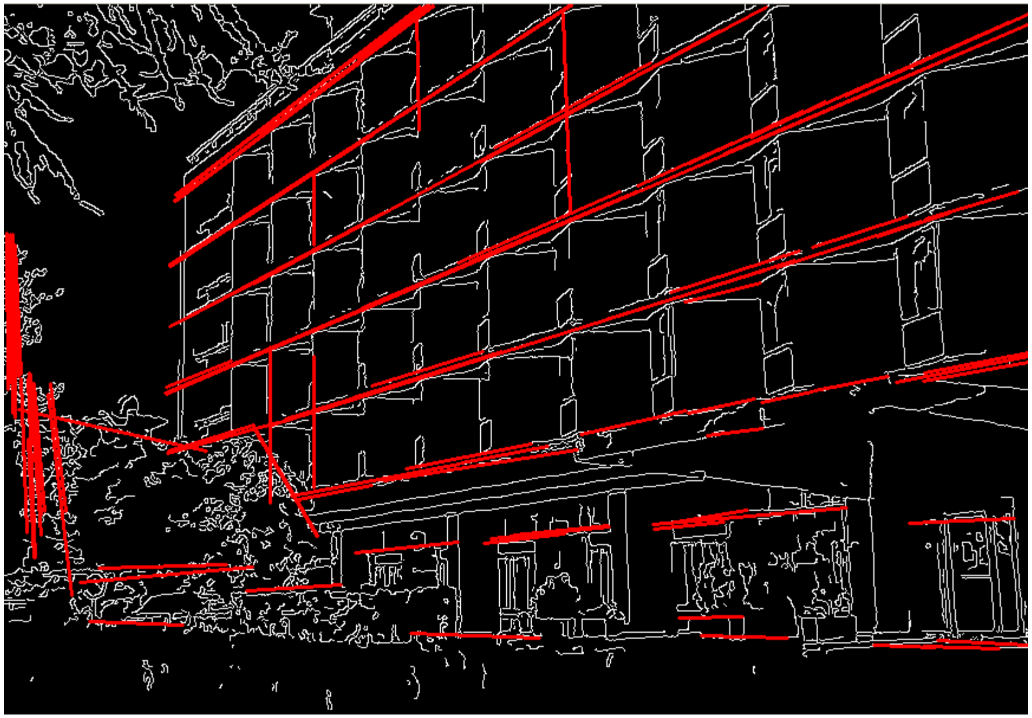
Fitting

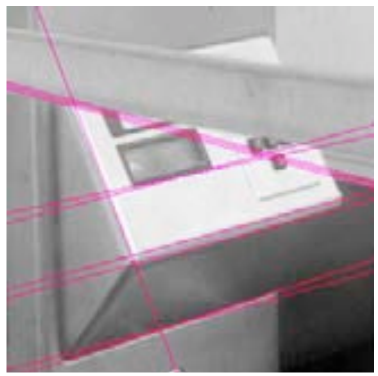


Fitting

- We've learned how to detect edges, corners, blobs. Now what?
- We would like to form a higher-level, more compact representation of the features in the image by grouping multiple features according to a simple model

Fitting

- Choose a *parametric model* to represent a set of features



simple model: lines

simple model: circles

complicated model: car

Source: K. Grauman

Fitting: Challenges

Case study: Line detection

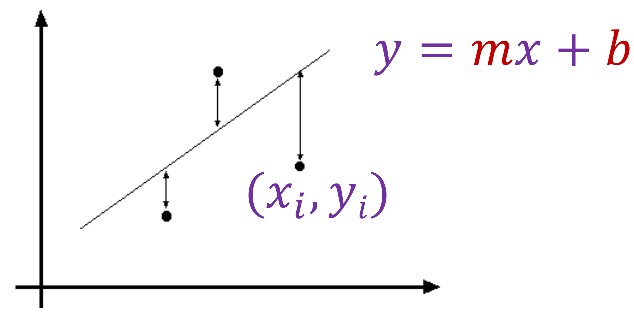
- **Noise** in the measured feature locations
- **Extraneous data:** clutter (outliers), multiple lines
- **Missing data:** occlusions

Fitting: Overview

- Least squares line fitting
- Robust fitting
- RANSAC

Least squares line fitting: First attempt

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



$$E = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

- Equivalent to finding least-squares solution to:

$$\underset{X}{\begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}} \underset{B}{\begin{pmatrix} m \\ b \end{pmatrix}} = \underset{Y}{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}$$

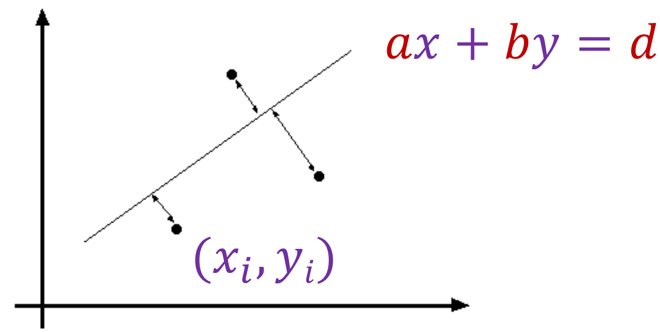
- Solution is given by $X^T X B = X^T Y$

Is this a good solution?

- Slope-intercept parametrization fails for vertical lines
- Solution is not equivariant w.r.t. rotation

Total least squares

- Line parametrization: $ax + by = d$
  - (a, b) is the *unit normal* to the line (i.e., $a^2 + b^2 = 1$)
  - d is the distance between the line and the origin



- Perpendicular distance between point (x_i, y_i) and line $ax + by = d$ (assuming $a^2 + b^2 = 1$):

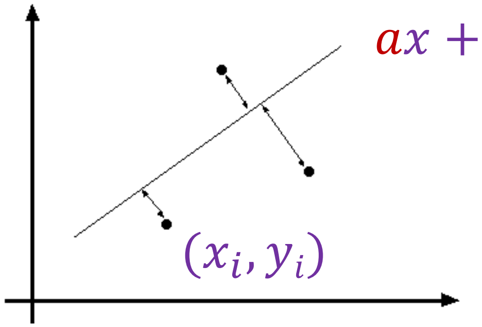
$$|ax_i + by_i - d|$$

- Objective function:

$$E = \sum_{i=1}^{n} (ax_i + by_i - d)^2$$

Total least squares

$$E = \sum_{i=1}^{n} (ax_i + by_i - d)^2$$



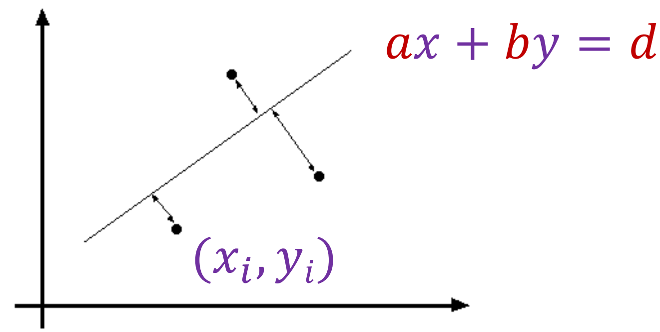
- Solve for d first:

$$\frac{\partial E}{\partial d} = -2 \sum_{i=1}^{n} (ax_i + by_i - d) = 0$$

$$d = \frac{a}{n} \sum_{i=1}^{n} x_i + \frac{b}{n} \sum_{i=1}^{n} y_i = a\bar{x} + b\bar{y}$$

Total least squares

$$E = \sum_{i=1}^{n} (ax_i + by_i - d)^2$$

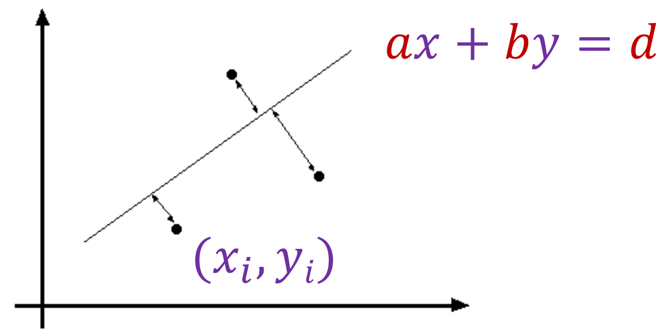


- Solve for d first: $d = a\bar{x} + b\bar{y}$
- Plugging back in:

$$E = \sum_{i=1}^{n} (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2$$

Total least squares

$$E = \sum_{i=1}^{n} (ax_i + by_i - d)^2$$



- Solve for d first: $d = a\bar{x} + b\bar{y}$
- Plugging back in:

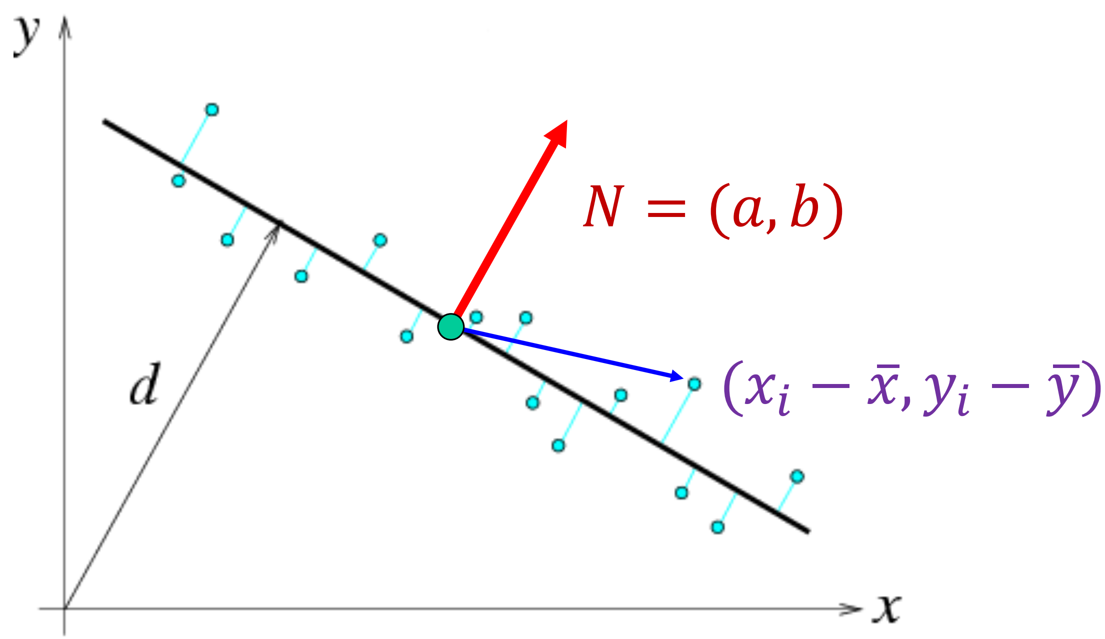
$$E = \sum_{i=1}^{n} (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2 = \left\| \underset{U}{\begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix}} \underset{N}{\begin{pmatrix} a \\ b \end{pmatrix}} \right\|^2$$

- We want to find N that minimizes $\|UN\|^2$ subject to $\|N\|^2 = 1$
  - Solution is given by the eigenvector of $U^T U$ associated with the smallest eigenvalue

Total least squares

$$U = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \qquad U^T U = \begin{bmatrix} \sum_{i=1}^{n}(x_i - \bar{x})^2 & \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \\ \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) & \sum_{i=1}^{n}(y_i - \bar{y})^2 \end{bmatrix}$$

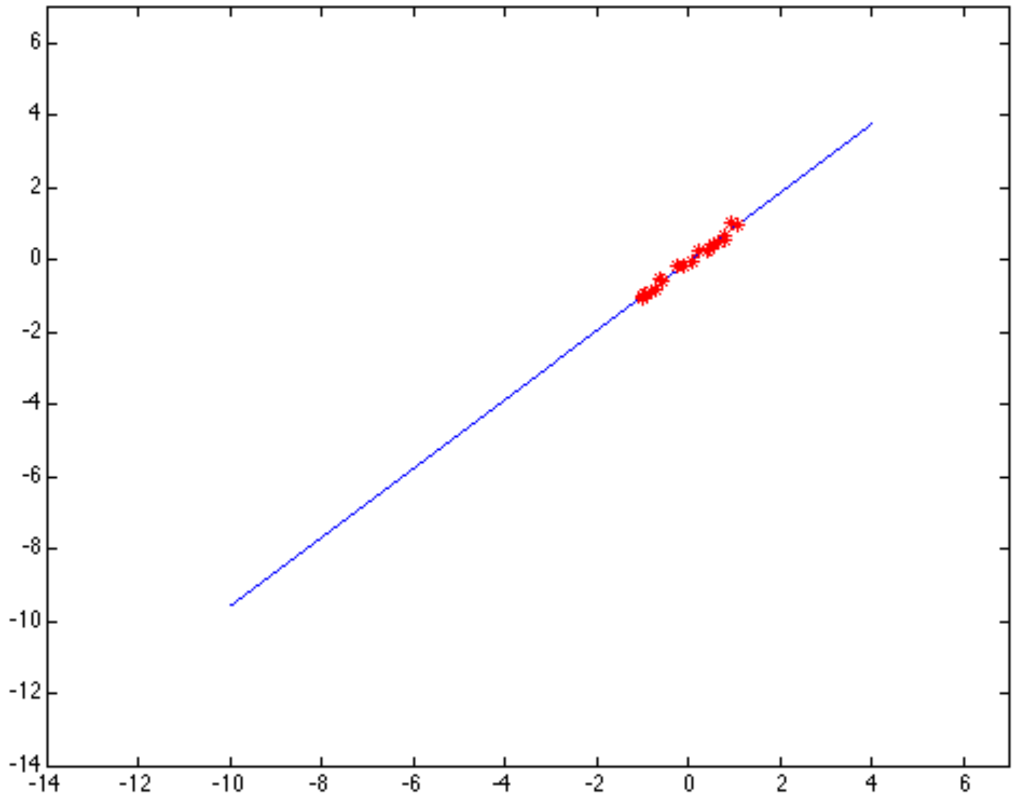
second moment matrix



F&P (2nd ed.) sec. 22.1

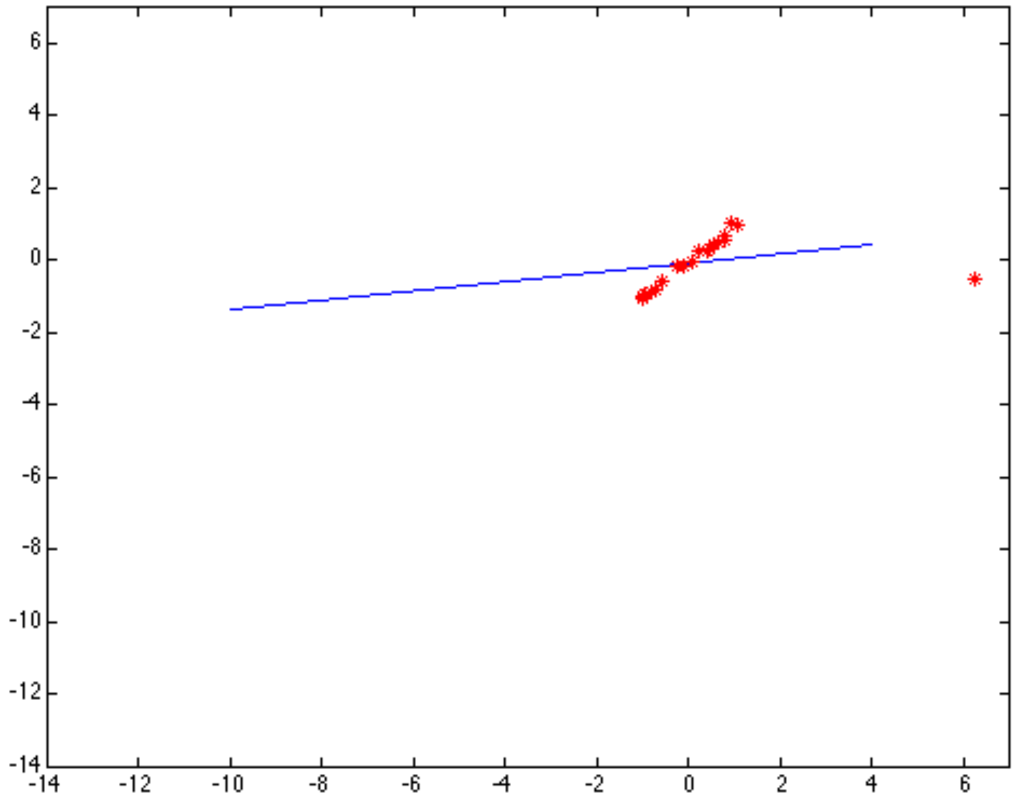
Least squares: Robustness to noise

- Least squares fit to the red points:



Least squares: Robustness to noise

- Least squares fit with an outlier:



Problem: squared error heavily penalizes outliers

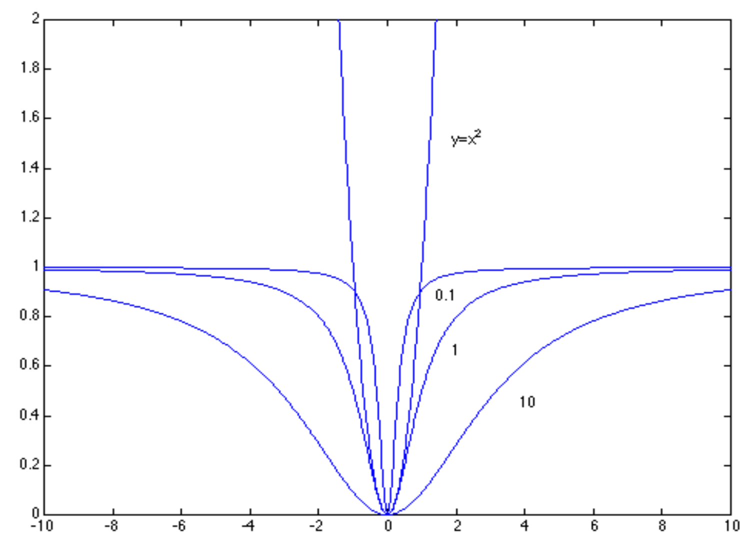
Robust estimators

- General approach: find model parameters θ that minimize

$$\sum_i \rho_\sigma(r(x_i;\theta))$$

$r(x_i;\theta)$: residual of x_i w.r.t. model parameters θ

ρ_σ: robust function with scale parameter σ, e.g., $\rho_\sigma(u) = \frac{u^2}{\sigma^2+u^2}$



Robust estimators

- General approach: find model parameters θ that minimize

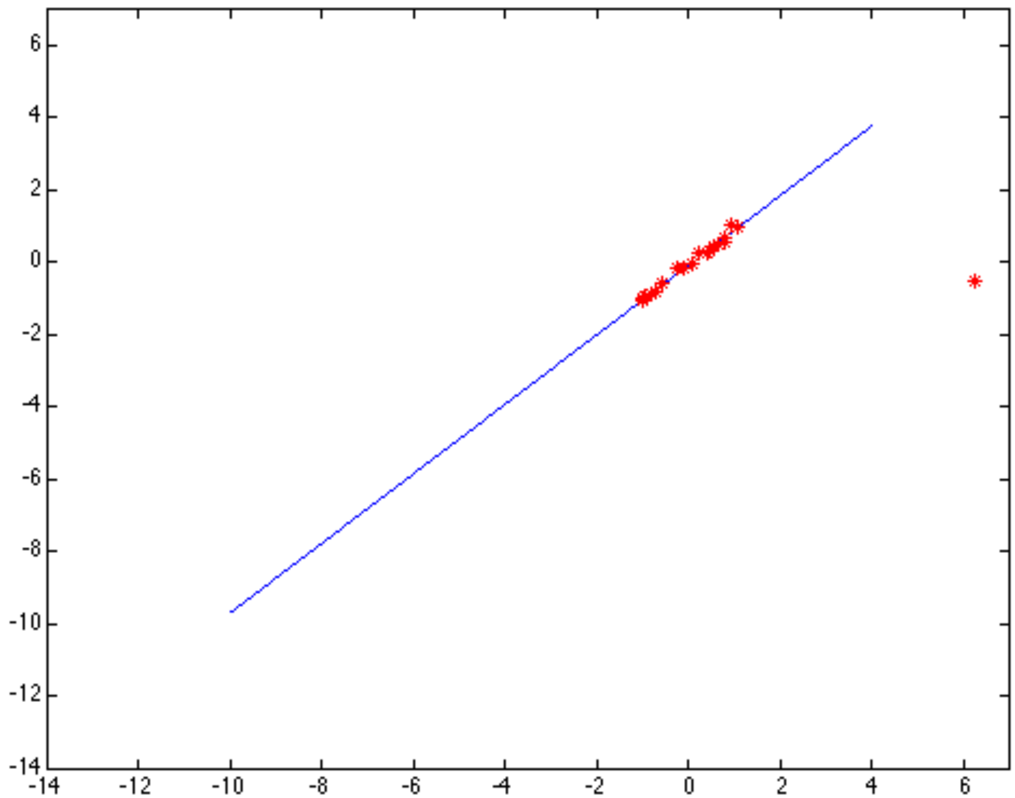
$$\sum_i \rho_\sigma(r(x_i;\theta))$$

$r(x_i;\theta)$: residual of x_i w.r.t. model parameters θ

ρ_σ: robust function with scale parameter σ, e.g., $\rho_\sigma(u) = \frac{u^2}{\sigma^2+u^2}$

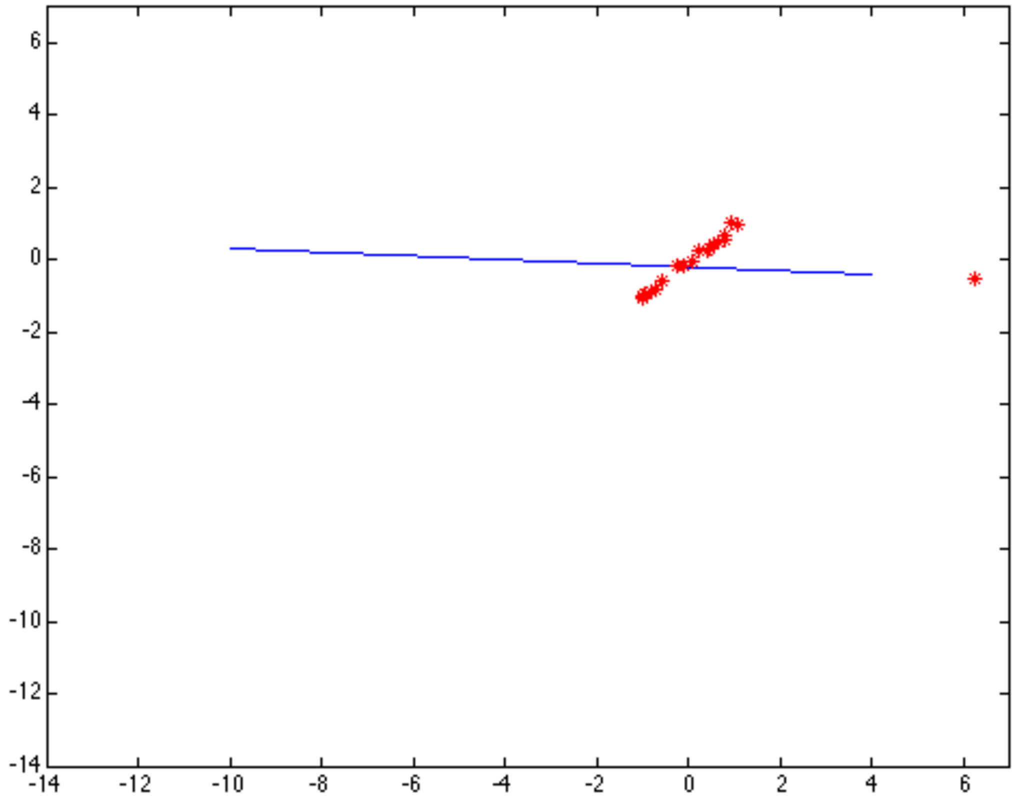
- Nonlinear optimization problem that must be solved iteratively
  - Least squares solution can be used for initialization
  - Scale of robust function should be chosen carefully

Choosing the scale: Just right



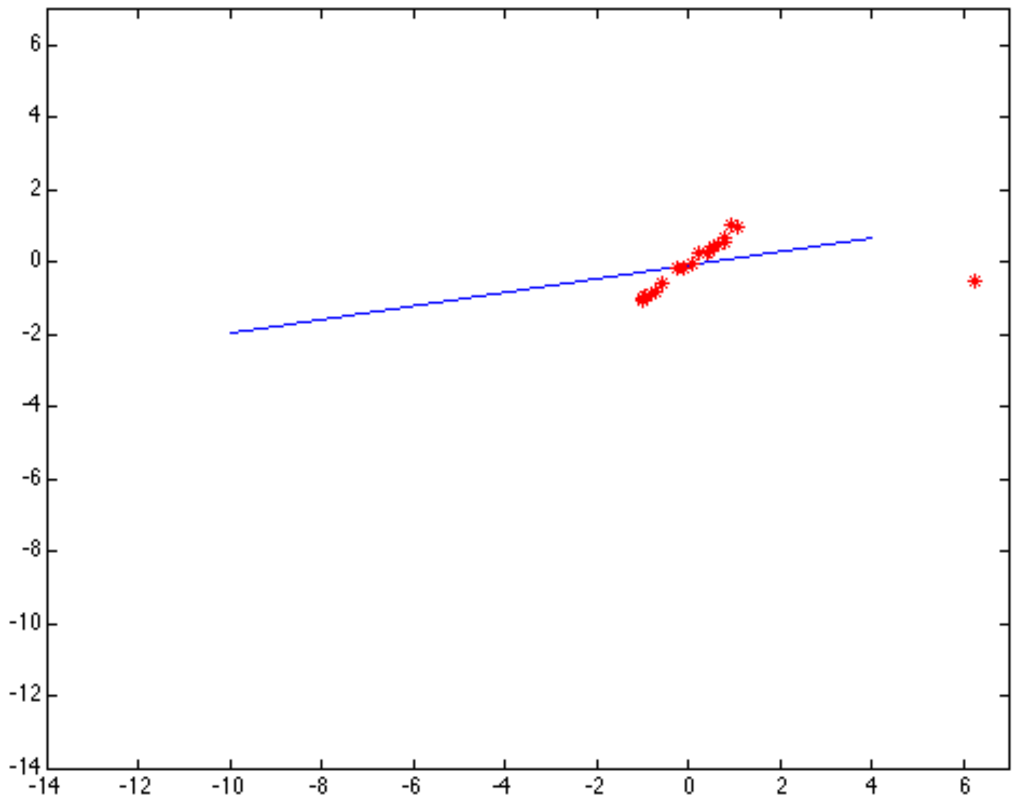
The effect of the outlier is minimized

Choosing the scale: Too small



The error value is almost the same for every point and the fit is very poor

Choosing the scale: Too large



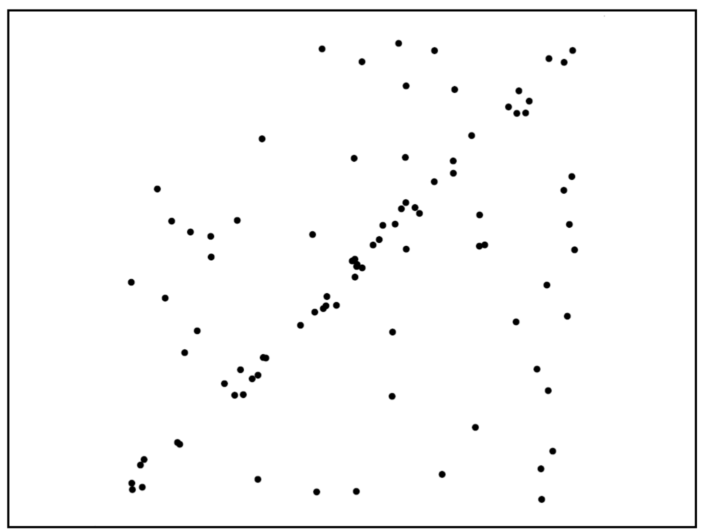
Behaves much the same as least squares

Fitting: Overview

- Least squares line fitting
- Robust fitting
- RANSAC

Voting schemes

- Robust fitting can deal with a few outliers – what if we have very many?
- Basic idea: let each point *vote* for all the models that are compatible with it
  - Hopefully the outliers will not vote consistently for any single model
  - The model that receives the most votes is the best fit to the image

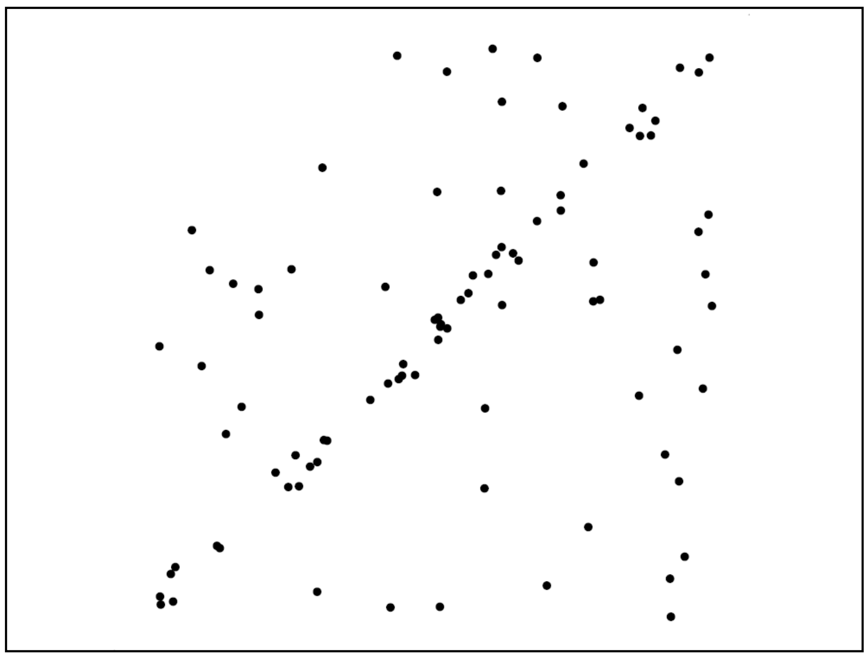


RANSAC

- Random sample consensus: very general framework for model fitting in the presence of outliers
- Outline:
  - Randomly choose a small initial subset of points
  - Fit a model to that subset
  - Find all inlier points that are “close” to the model and reject the rest as outliers
  - Do this many times and choose the model with the most inliers

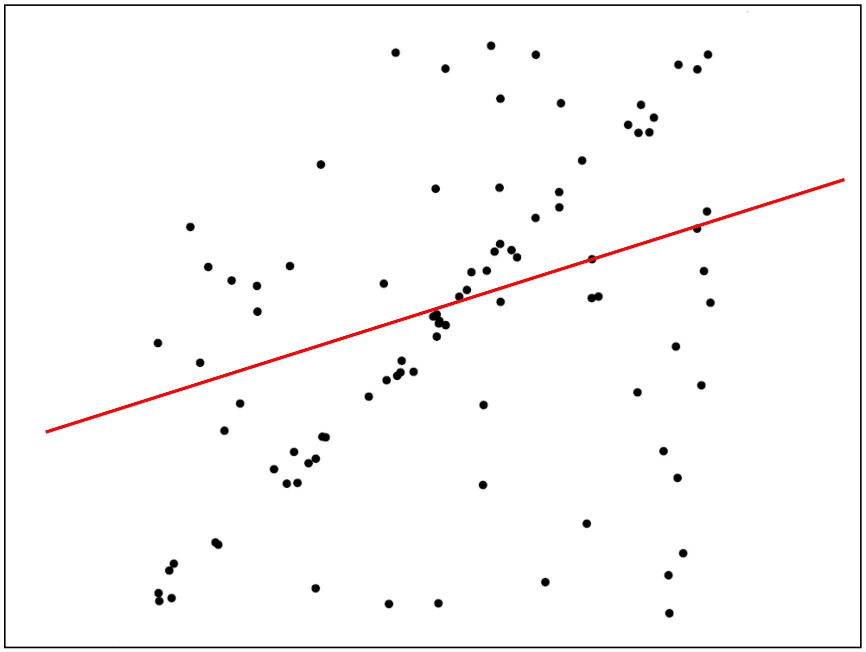
M. Fischler and R. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981

RANSAC for line fitting example



Source: R. Raguram

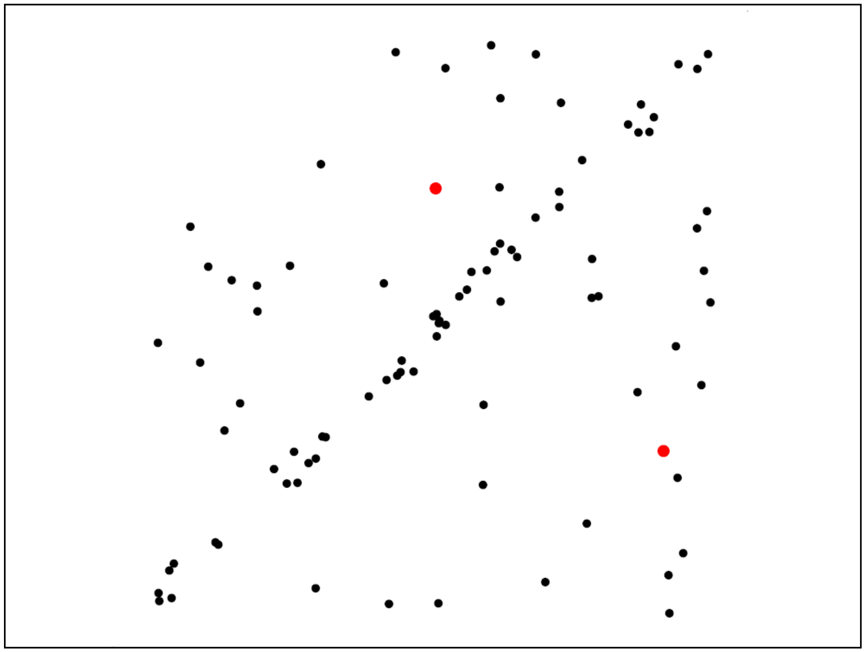
RANSAC for line fitting example



Least-squares fit

Source: R. Raguram

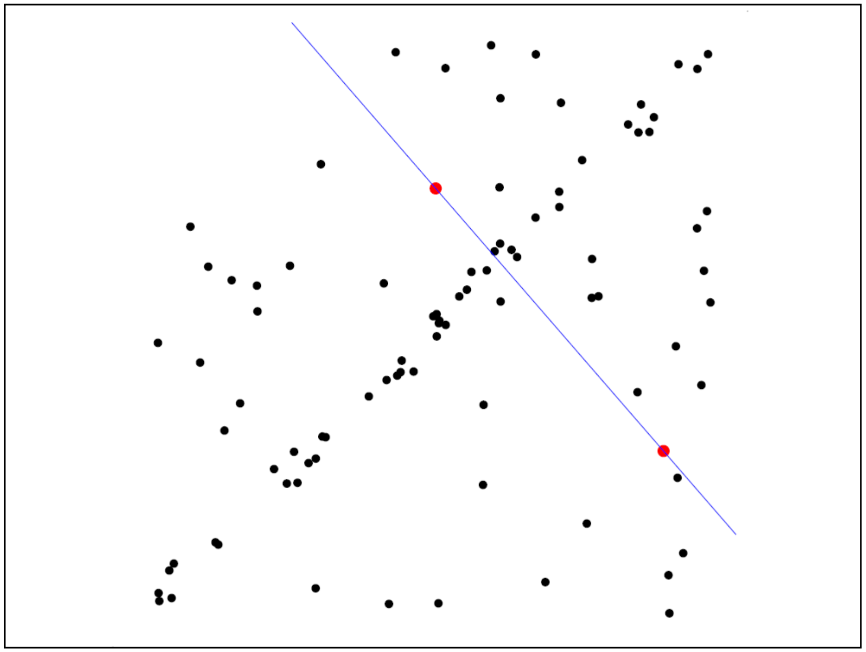
RANSAC for line fitting example



1. Randomly select minimal subset of points

Source: R. Raguram

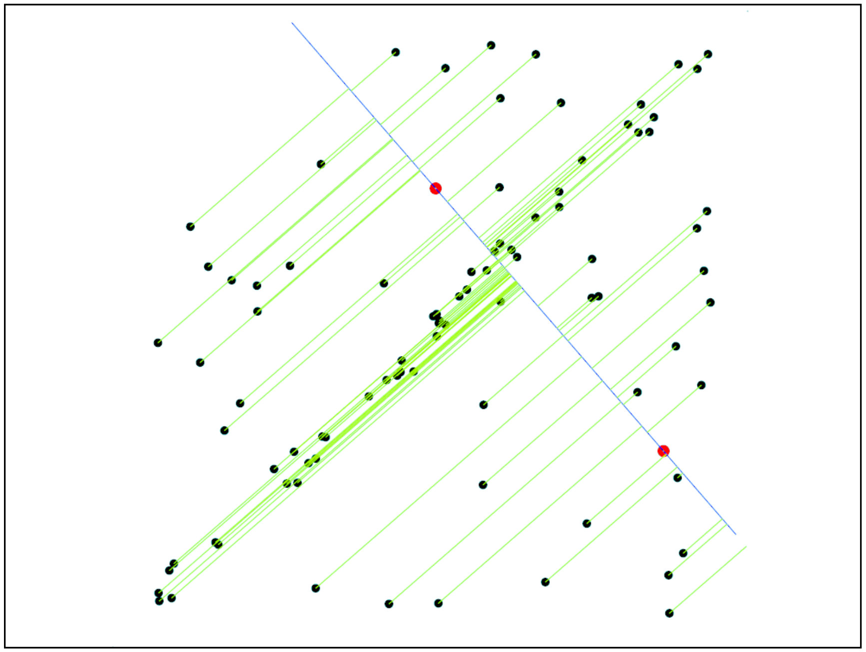
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model

Source: R. Raguram

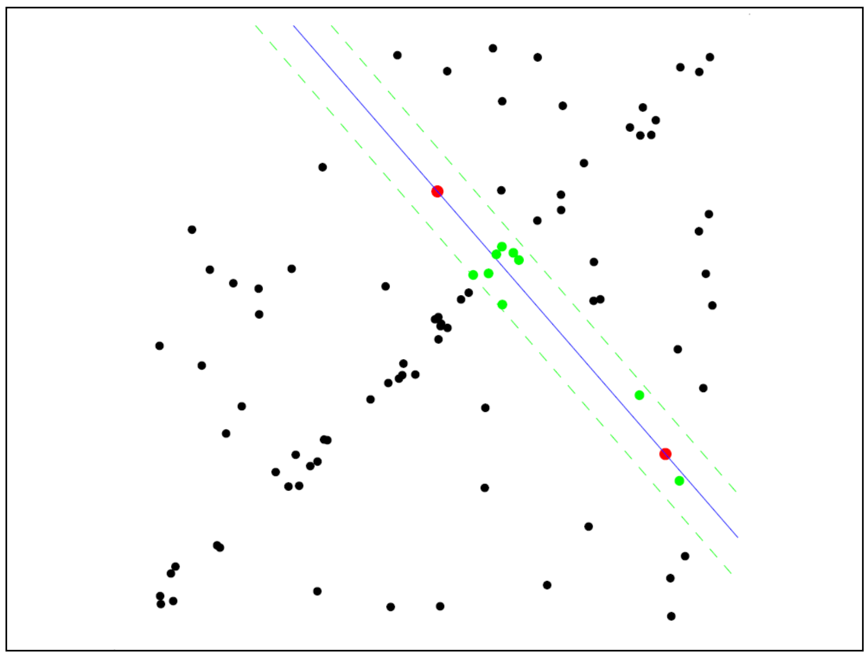
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function

Source: R. Raguram

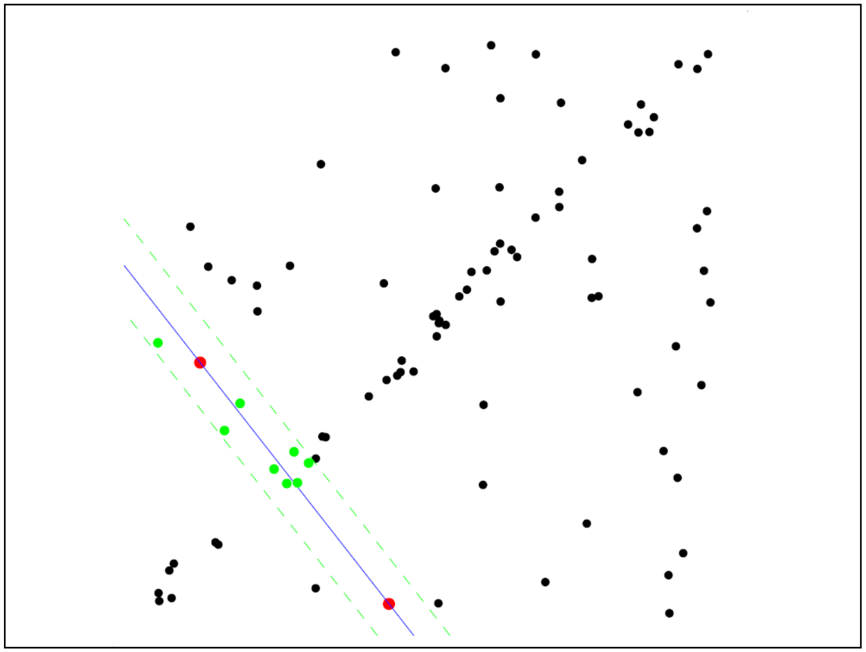
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model

Source: R. Raguram

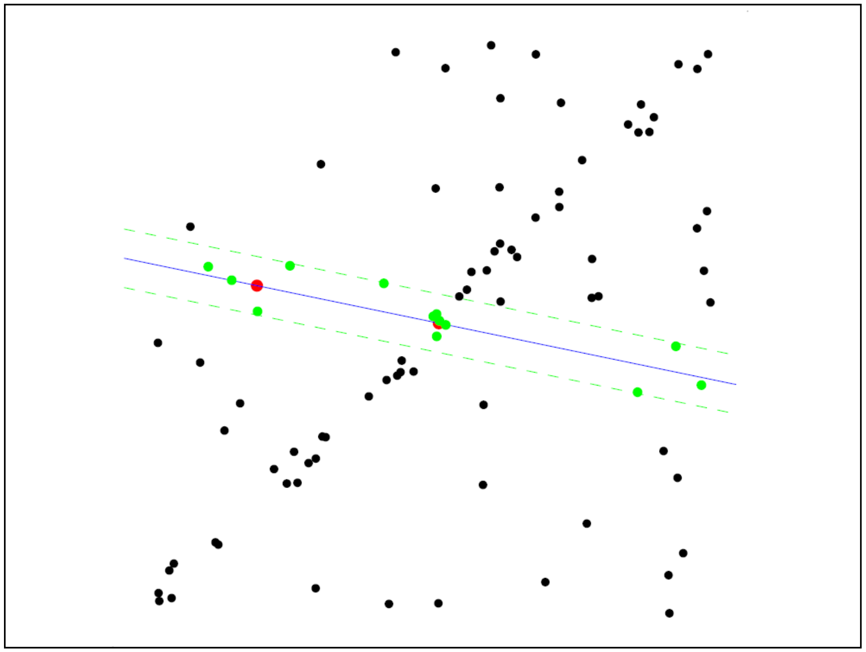
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example

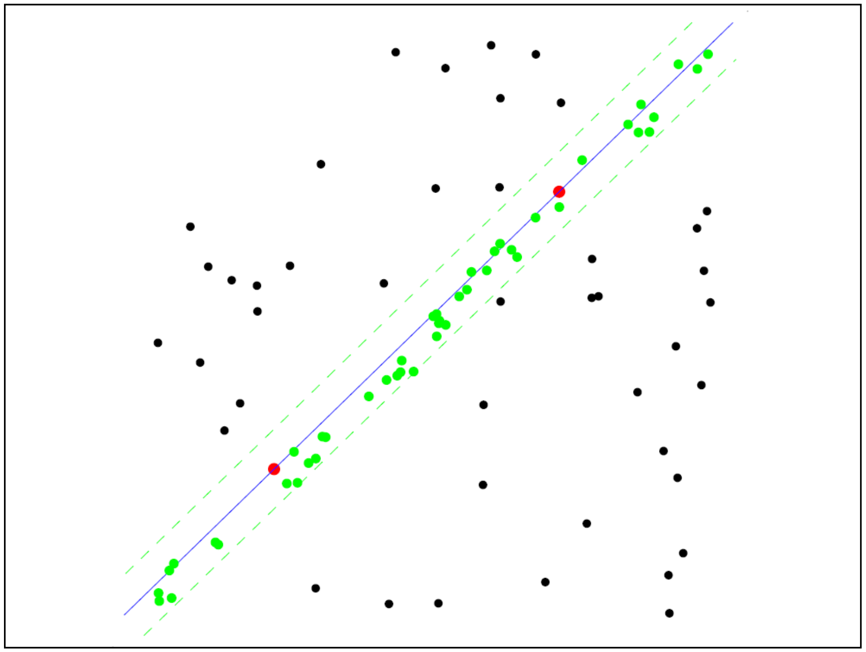


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example

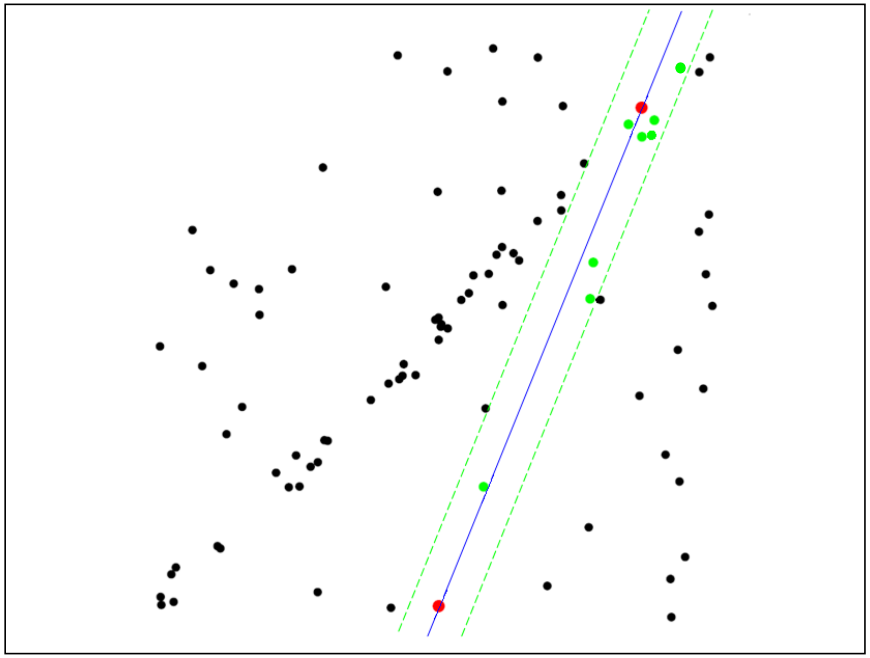
Uncontaminated sample



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC loop

Repeat N times:

- Draw s points uniformly at random
- Fit model to these s points
- Find *inliers* to the model among the remaining points (points whose distance or residual w.r.t. model is less than t)
- If there are d or more inliers, accept the model and refit using all inliers

RANSAC: Choosing the parameters

- Initial number of points s
  - Typically minimum number needed to fit the model
- Distance threshold t for inliers
  - Need suitable assumptions, e.g., given zero-mean Gaussian noise with std. dev. σ, $t = 1.96\sigma$ will give ~95% probability of capturing all inliers
- Consensus set size d
  - Should match expected inlier ratio

Adapted from M. Pollefeys

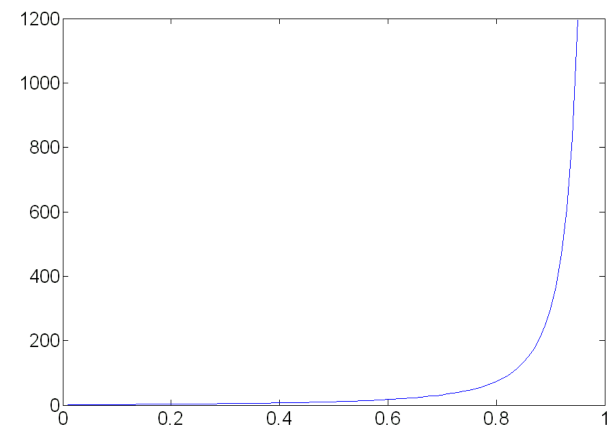
RANSAC: Choosing the parameters

- Choosing the number of iterations (initial samples) N:
  - Choose N so that, with probability p (e.g. 99%), at least one initial sample is free from outliers
  - Assuming an outlier ratio of e:

$$(1-(1-e)^s)^N = 1-p$$

$$N = \log(1-p)\,/\log(1-(1-e)^s)$$

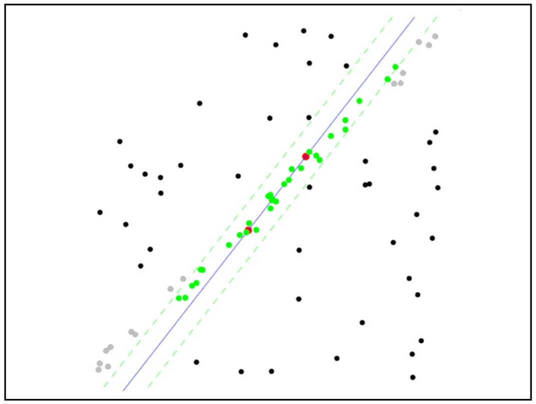
	proportion of outliers e						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177



Source: M. Pollefeys

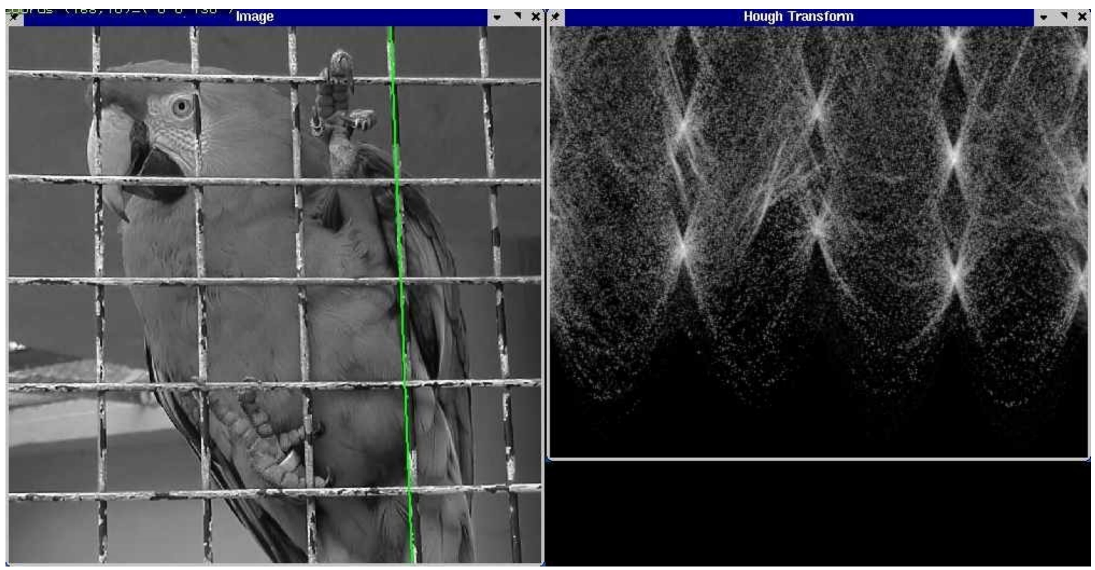
RANSAC pros and cons

- Pros
  - Simple and general
  - Applicable to many different problems
  - Often works well in practice
- Cons
  - Lots of parameters to set
  - Number of iterations grows exponentially as outlier ratio increases
  - Can’t always get a good initialization of the model based on the minimum number of samples



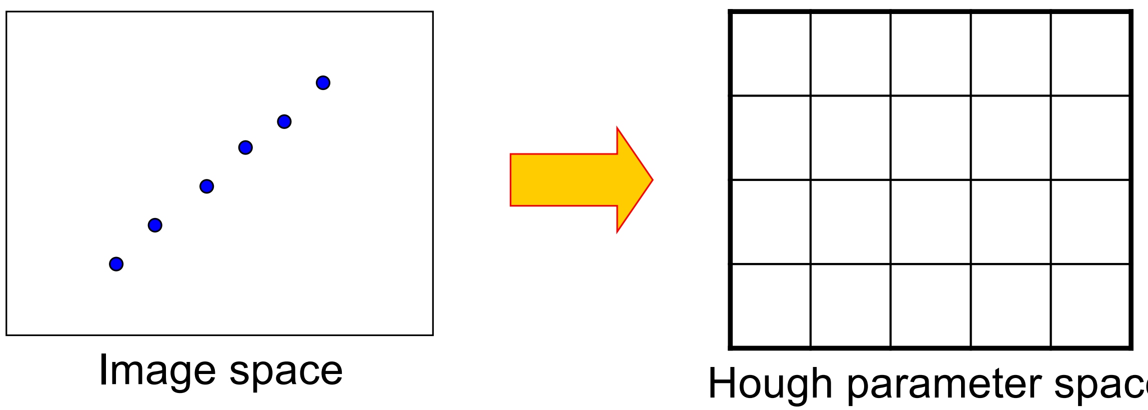
Fitting: Overview

- Least squares line fitting
- Robust fitting
- RANSAC
- Hough transform



Hough transform

- Possibly the earliest voting scheme – but still useful!
  - Discretize parameter space into bins
  - For each feature point in the image, put a vote in every bin in the parameter space that could have generated this point
  - Find bins that have the most votes



P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures*, Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

Hough transform

- What does a **line** in the image space correspond to?
  - A **point** in the parameter space

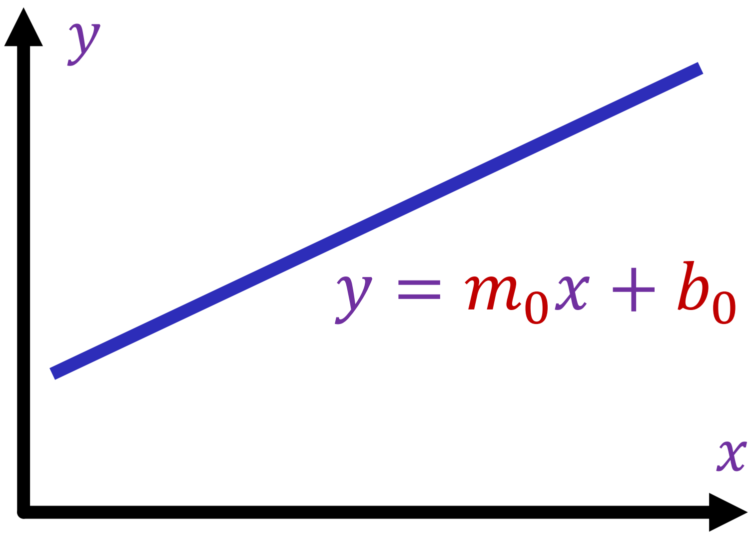
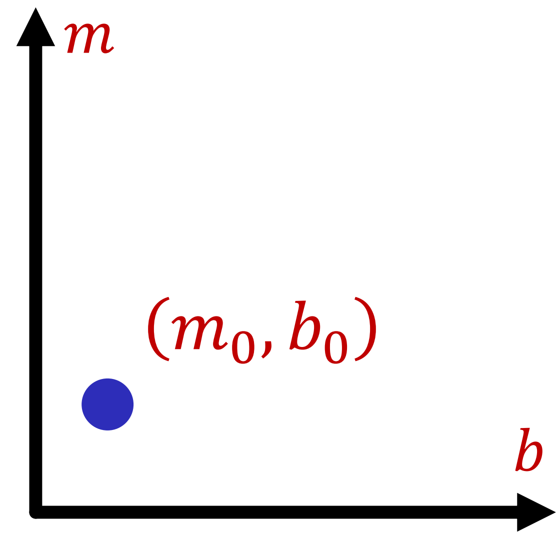


Image space



Parameter space

Hough transform

- What does a **point** in the image space correspond to?
  - A **line** in the parameter space: all (m, b) that satisfy $-b = x_0 m - y_0$

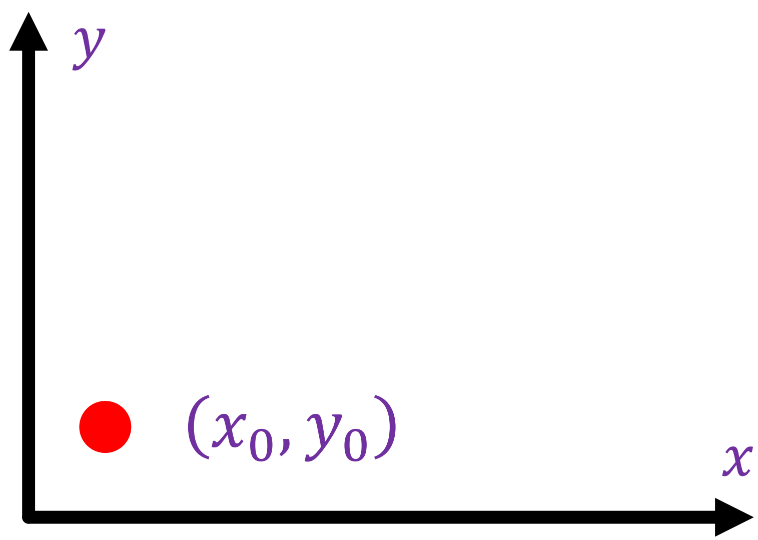
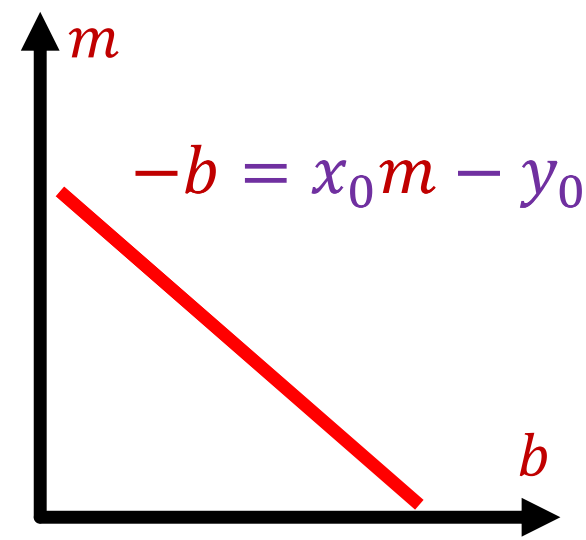


Image space



Parameter space

Hough transform

- What about **two points** in the image space?
  - A **point** in the parameter space, corresponding to the unique line that passes through both points

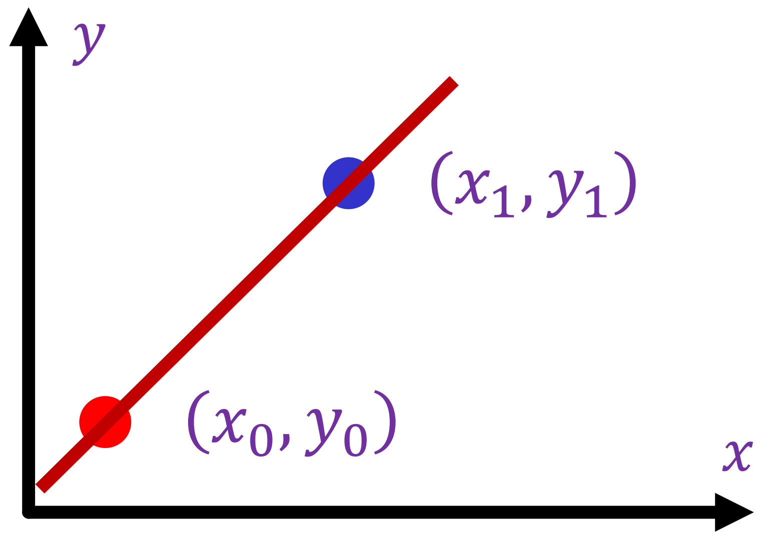
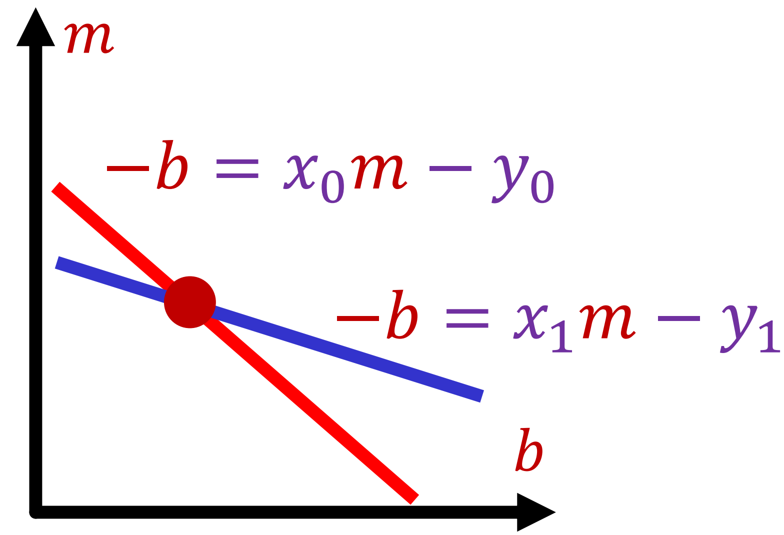


Image space



Parameter space

Hough transform

- What about **many points** in the image space?
  - Plot all the lines in the parameter space and try to find a spot where a large number of them intersect

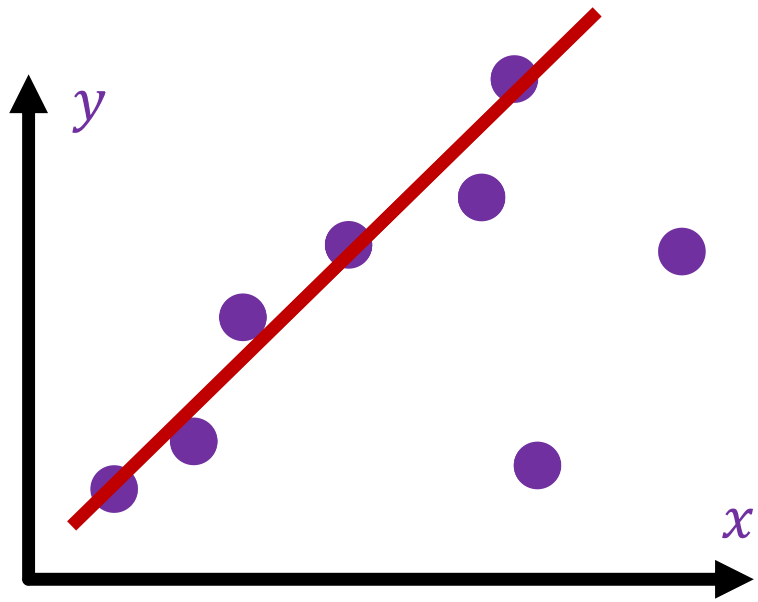
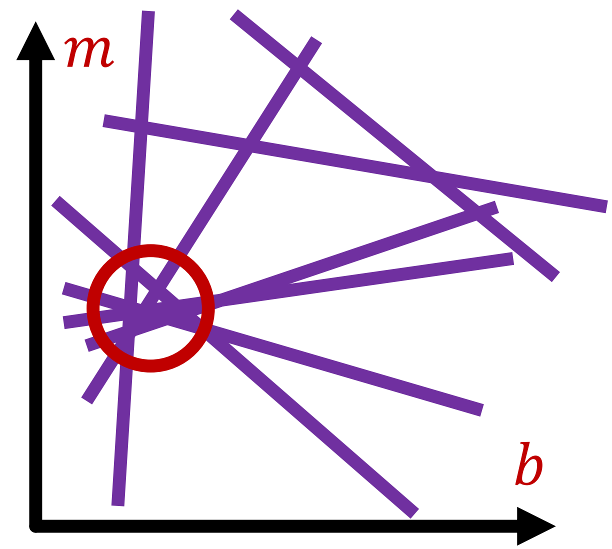


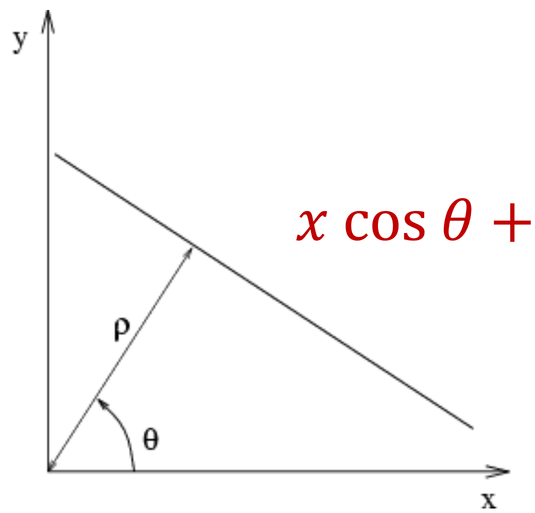
Image space



Parameter space

Parameter space representation

- In practice, we don't want to use the (m, b) space!
  - Unbounded parameter domains
  - Vertical lines require infinite m
- Alternative: *polar representation*
  - Each image point (x, y) yields a *sinusoid* in the (θ, ρ) parameter space



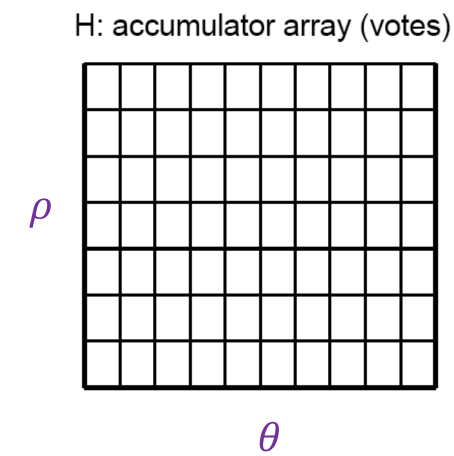
$$x \cos \theta + y \sin \theta = \rho$$

Algorithm outline

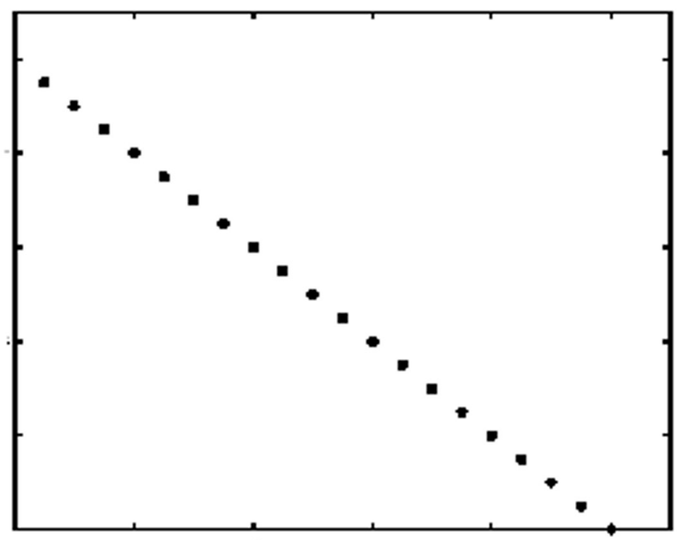
- Initialize accumulator H to all zeros
- For each feature point (x, y)
  - For $\theta = 0$ to 180

    $\rho = x \cos\theta + y \sin\theta$
    $H(\theta, \rho) \mathrel{+}= 1$

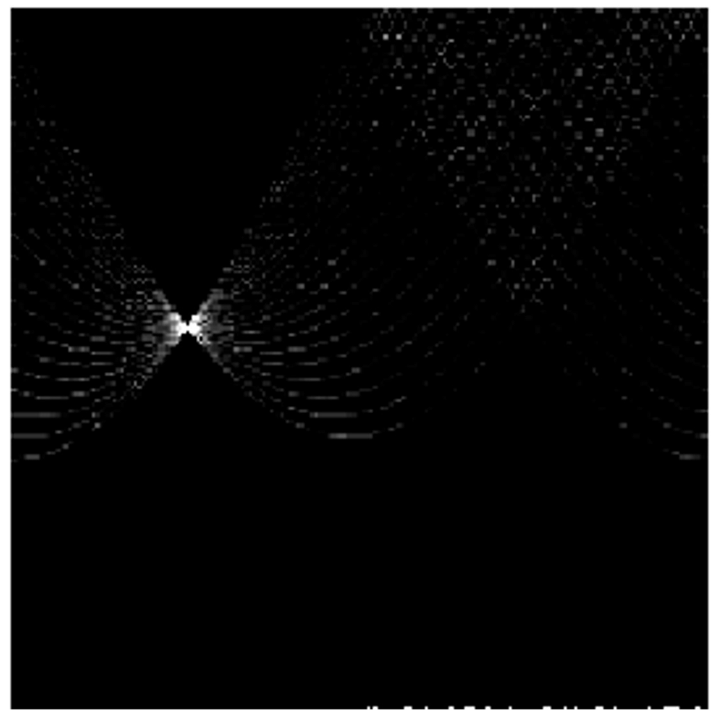
- Find the value(s) of (θ, ρ) where $H(\theta, \rho)$ is a local maximum (perform NMS on the accumulator array)
  - The detected line in the image is given by $\rho = x \cos\theta + y \sin\theta$



Basic illustration



features



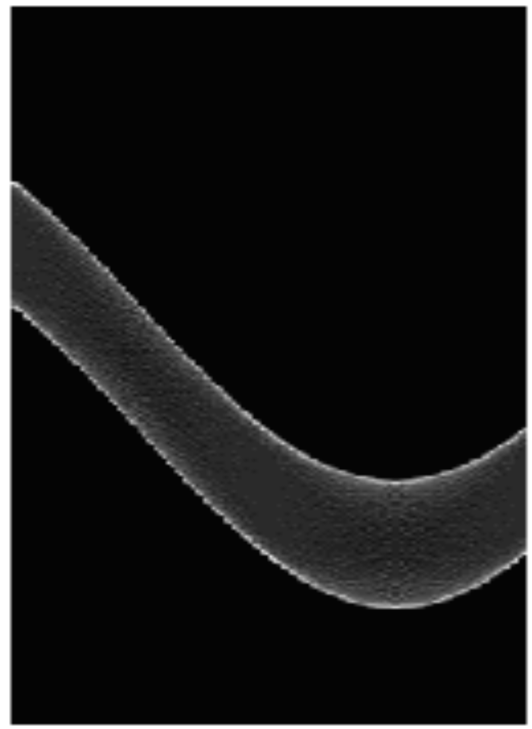
votes

Hough transform demo

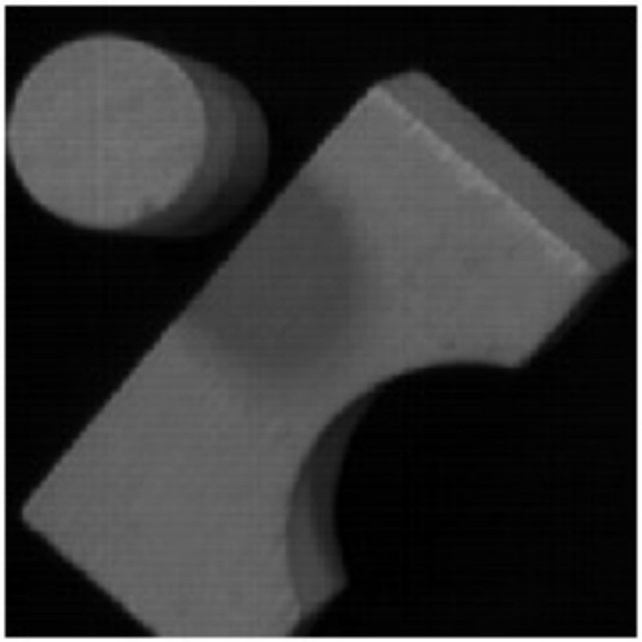
Other shapes

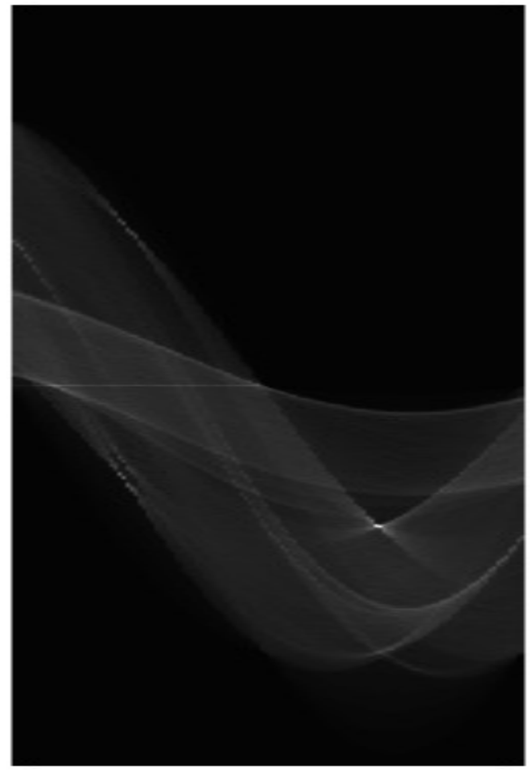
Square

Circle

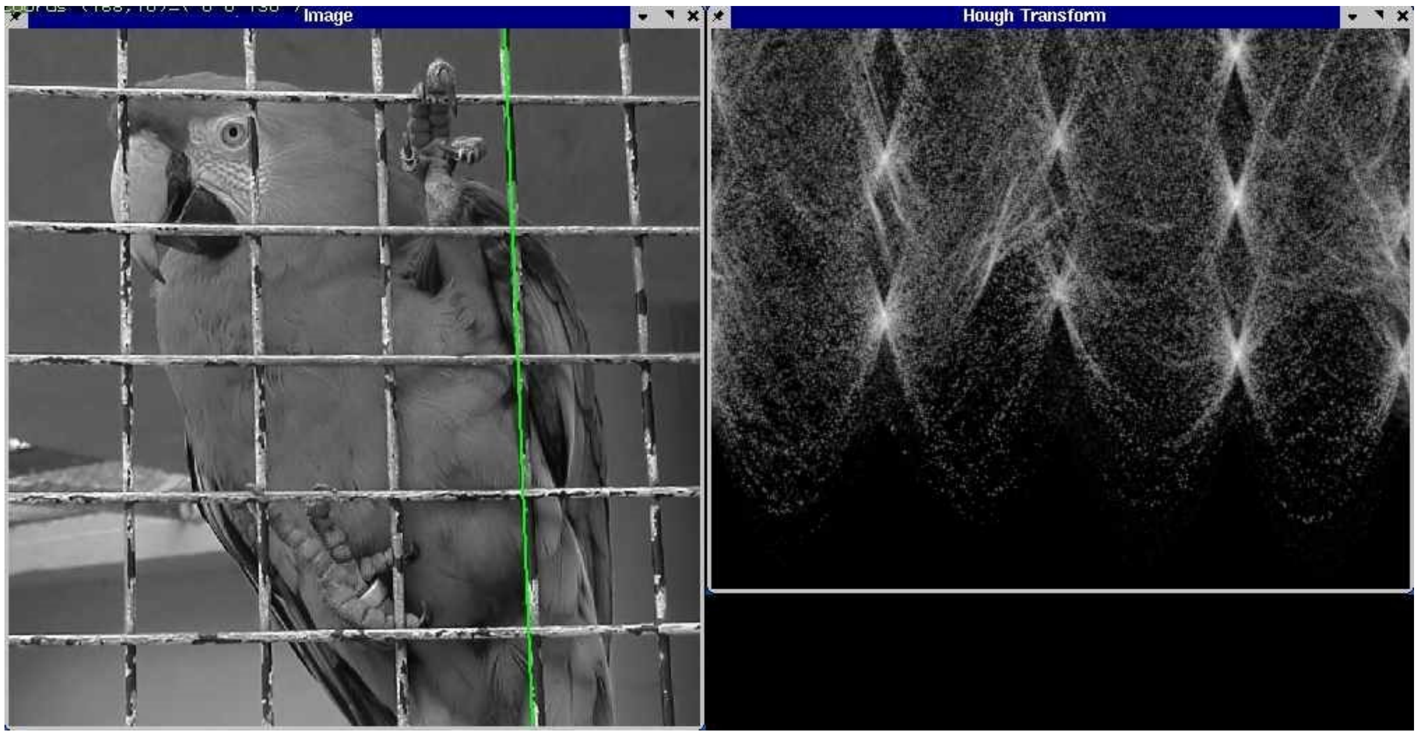


Several lines



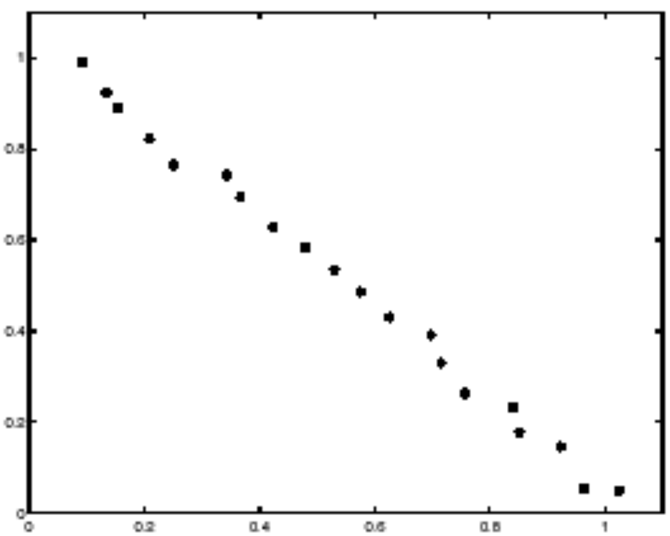


A more complicated image



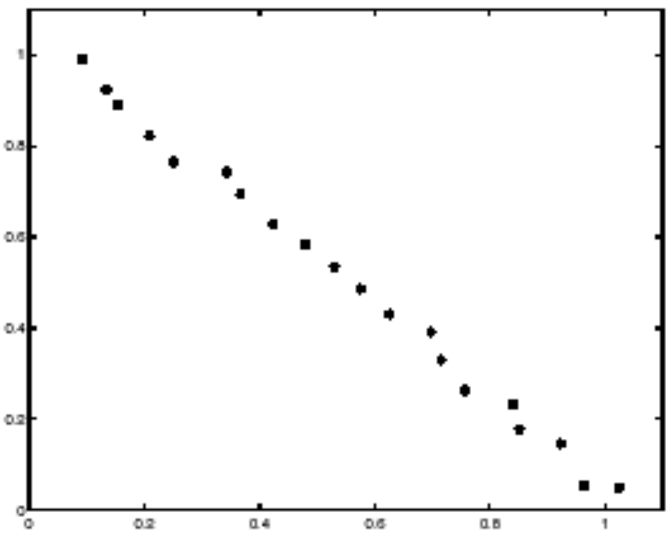
Source

Effect of noise

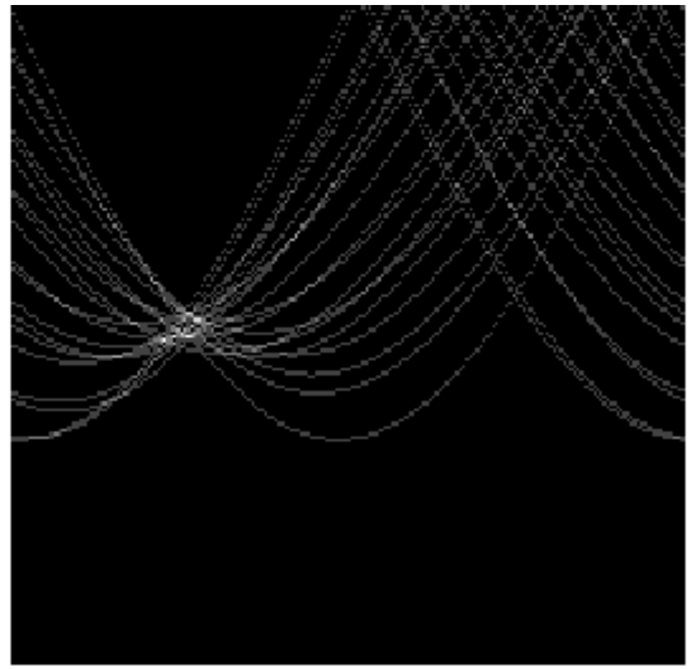


features

Effect of noise



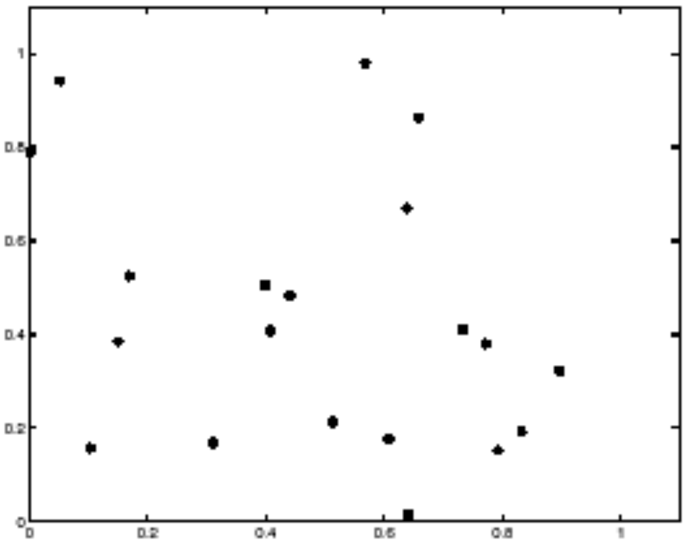
features



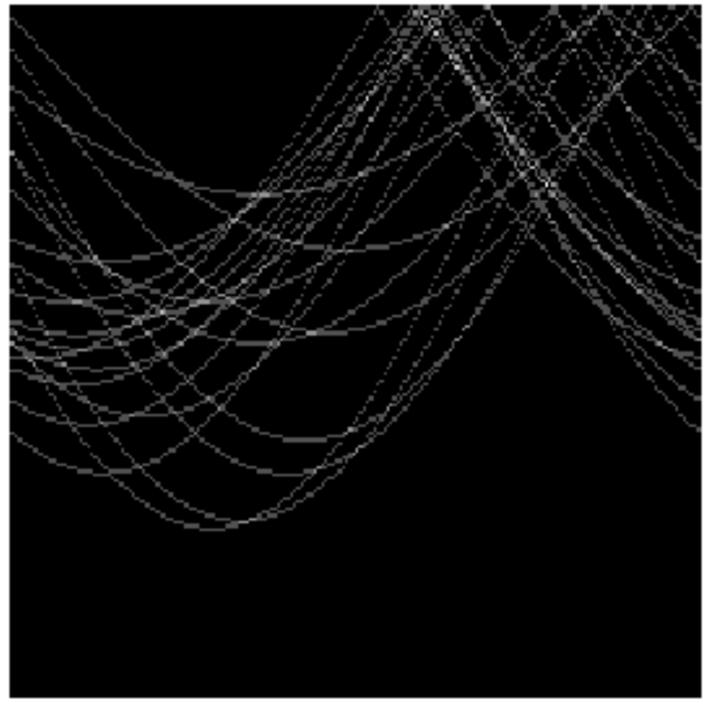
votes

Peak gets fuzzy and hard to locate

Effect of outliers



features



votes

Uniform noise can lead to spurious peaks in the array

Dealing with noise

- How to choose a good grid discretization?
  - **Too coarse:** large votes obtained when too many different lines correspond to a single bucket
  - **Too fine:** miss lines because some points that are not exactly collinear cast votes for different buckets
- Increment neighboring bins (smoothing in accumulator array)
- Try to get rid of irrelevant features
  - E.g., take only edge points with significant gradient magnitude

Hough transform: Pros and cons

- Pros
  - Can deal with non-locality and occlusion
  - Can detect multiple instances of a model
  - Some robustness to noise: noise points unlikely to contribute consistently to any single bin
  - Leads to a surprisingly general strategy for shape localization (more on this next)
- Cons
  - Complexity increases exponentially with the number of model parameters – in practice, not used beyond three or four dimensions
  - Non-target shapes can produce spurious peaks in parameter space
  - It’s hard to pick a good grid size

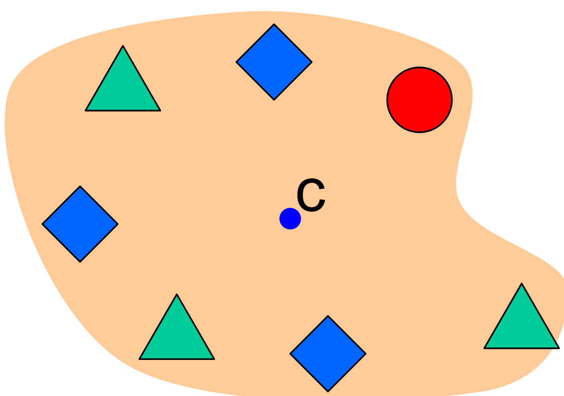
Fitting: Overview

- Least squares line fitting
- Robust fitting
- RANSAC
- Hough transform
- Generalized Hough transform

Generalized Hough transform

- We want to find a template defined by its reference point (center) and several distinct types of landmark points in stable spatial configuration

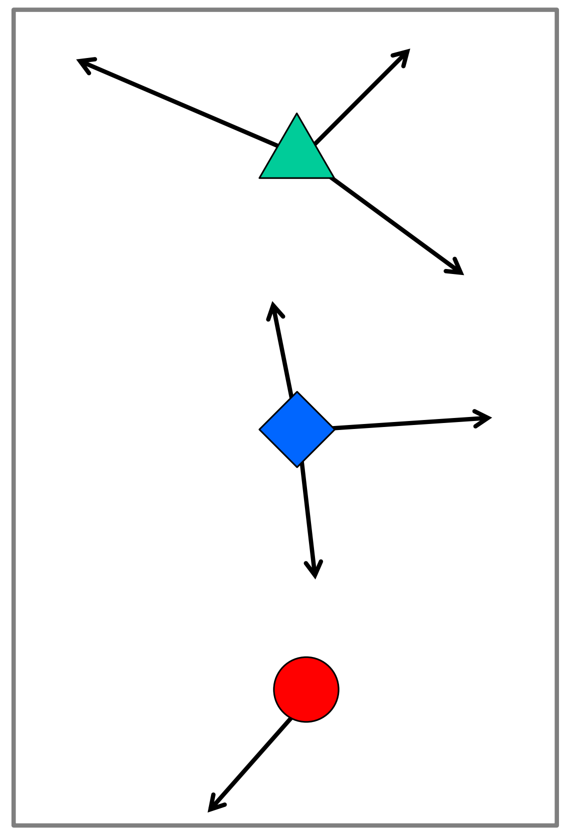
Template



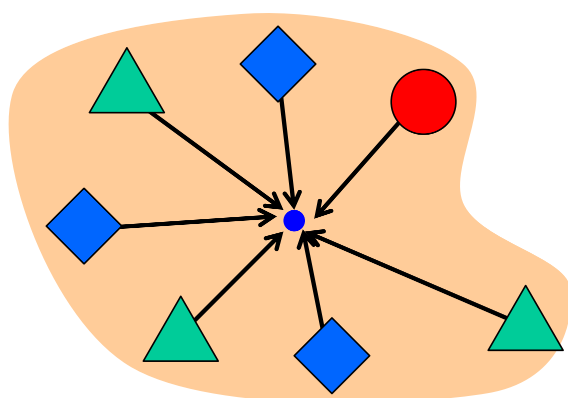
Generalized Hough transform

- Template representation: for each type of landmark point, store all possible displacement vectors towards the center

Model



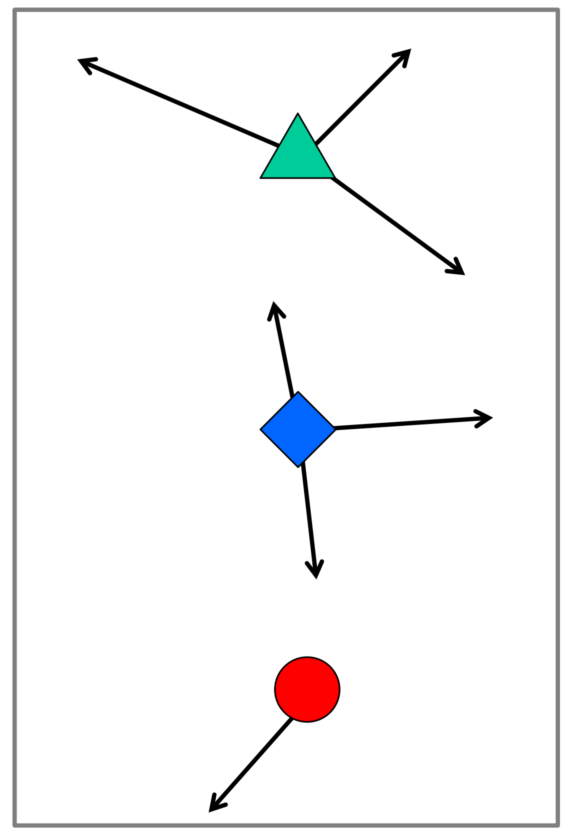
Template



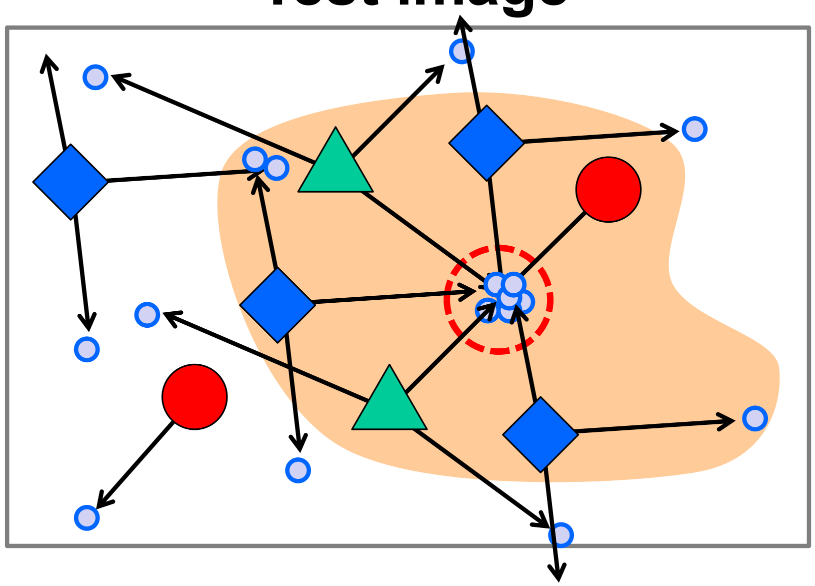
Generalized Hough transform

- Detecting the template:
  - For each feature in a new image, look up that feature type in the model and vote for the possible center locations associated with that type in the model

Model

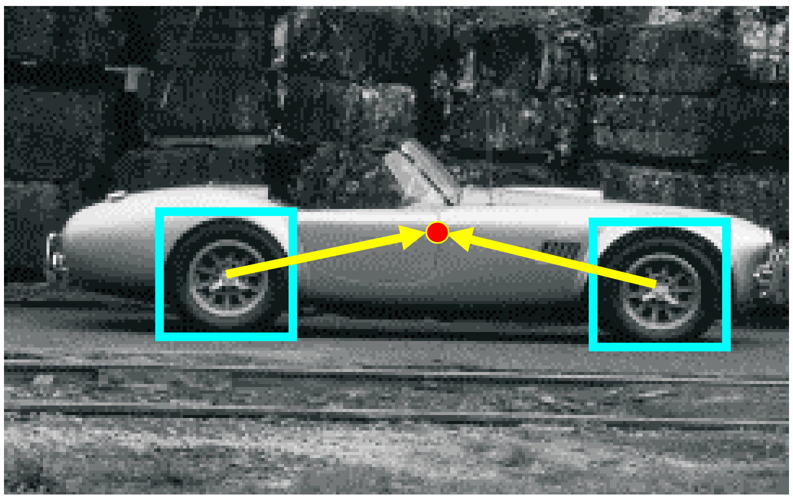


Test image

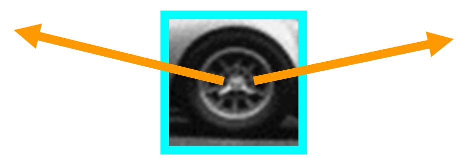


Application in recognition

- Index displacements by “visual codeword”



training image



visual codeword with displacement vectors

B. Leibe, A. Leonardis, and B. Schiele, Combined Object Categorization and Segmentation with an Implicit Shape Model, ECCV Workshop on Statistical Learning in Computer Vision 2004

Application in recognition

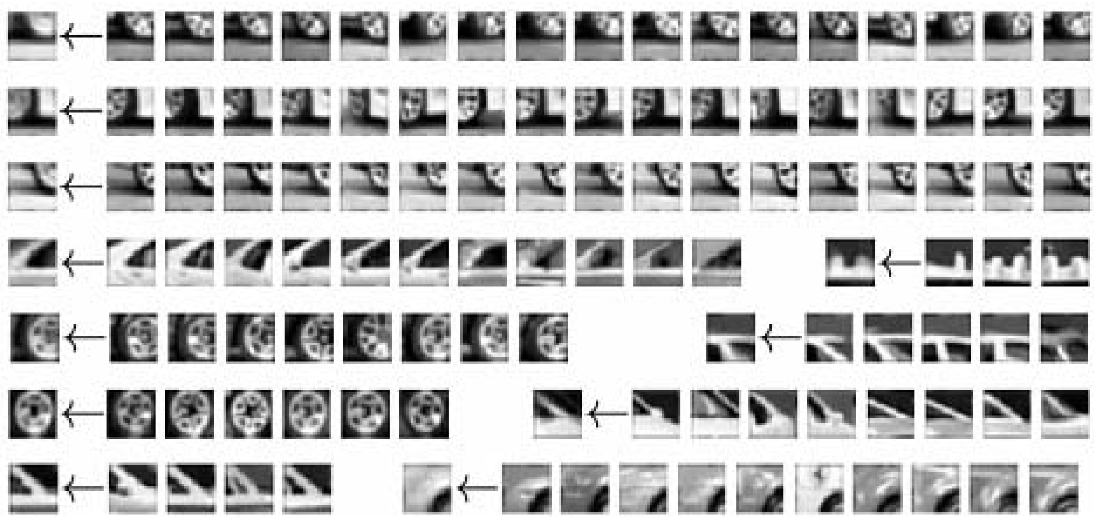
- Index displacements by “visual codeword”

test image

B. Leibe, A. Leonardis, and B. Schiele, [Combined Object Categorization and Segmentation with an Implicit Shape Model](), ECCV Workshop on Statistical Learning in Computer Vision 2004

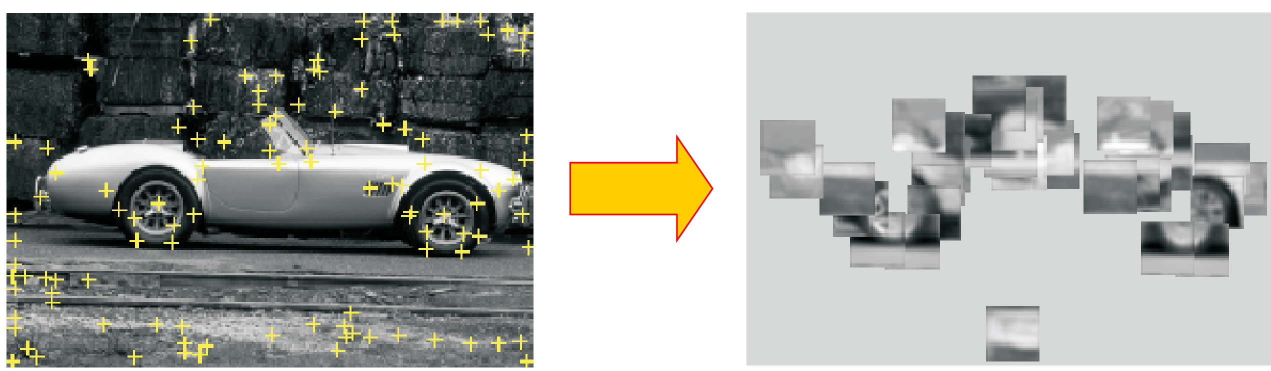
Implicit shape models: Training

1. Build *codebook* of patches around extracted interest points using clustering



Implicit shape models: Training

1. Build *codebook* of patches around extracted interest points using clustering
2. Map the patch around each interest point to closest codebook entry



Building a codebook

Get many images

1. find interest points;
2. find windows;
3. build feature vectors

We believe there are k kinds of interest point

a wheel; a bush; a doorhandle; a window; etc.

Wheels mostly look like one another (but not exactly) and not like windows

We know k (get to this later)

Write:

$\mathbf{x}_i$ Feature vector of k'th example interest point

$\mathbf{c}_j$ Feature vector of j'th center, which describes j'th *kind* (unknown)

Building a codebook

We believe there are k kinds of interest point

- a wheel; a bush; a doorhandle; a window; etc.
- Wheels mostly look like one another (but not exactly) and not like windows
  - So each wheel (say) feature vector should be close to one another
    - encode this with a wheel center, etc

Write:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i\text{'th vector is of } k\text{'th type} \\ 0 & \text{otherwise} \end{cases}$$

Notice:

$$\sum_i \delta_{ij} = 1$$

(Also, we don't know these!)

Building a codebook

Idea:

Choose δ_{ij} and $\mathbf{c}_j$

To minimize:

$$\sum_{ij} \delta_{ij} \left[(\mathbf{x}_i - \mathbf{c}_j)^T (\mathbf{x}_i - \mathbf{c}_j) \right]$$

So that: example takes the kind of the closest center
and center is close to all examples of that kind

Approximation

Minimize:

$$\sum_{ij} \delta_{ij} \left[(\mathbf{x}_i - \mathbf{c}_j)^T (\mathbf{x}_i - \mathbf{c}_j) \right]$$

Is intractable as an exact problem.

BUT

- if you know deltas, easy to choose c's
  - center is average of all points of that kind
- if you know centers, easy to choose deltas
  - each point goes to closest center

Issues

How to start?

centers are randomly chosen from data

quite good, better to follow

When to stop?

center locations change little; OR

deltas do not change

What about kinds with no examples?

(you can't average 0 points!)

choose center randomly from data

Algorithm: K-means

Procedure: 10.1 *K-Means Clustering*

Choose k. Now choose k data points $\mathbf{c}_j$ to act as cluster centers. Until the cluster centers change very little

- Allocate each data point to cluster whose center is nearest.
- Now ensure that every cluster has at least one data point; one way to do this is by supplying empty clusters with a point chosen at random from points far from their cluster center.
- Replace the cluster centers with the mean of the elements in their clusters.

Starting K-means

It turns out the choice of initial estimates of the centers can matter a lot, too. One natural strategy for initializing k-means is to choose k data items at random, then use each as an initial cluster center. This approach is widely used, but has some difficulties. The quality of the clustering can depend quite a lot on initialization, and an unlucky choice of initial points might result in a poor clustering. One (again quite widely adopted) strategy for managing this is to initialize several times, and choose the clustering that performs best in your application. Another strategy, which has quite good theoretical properties and a good reputation, is known as *k-means++*. You choose a point $\mathbf{x}$ uniformly and at random from the dataset to be the first cluster center. Then you compute the squared distance between that point and each other point; write $d_i^2(\mathbf{x})$ for the distance from the i'th point to the first center. You now choose the other $k-1$ cluster centers as IID draws from the probability distribution

$$\frac{d_i^2(\mathbf{x})}{\sum_u d_u^2(\mathbf{x})}.$$

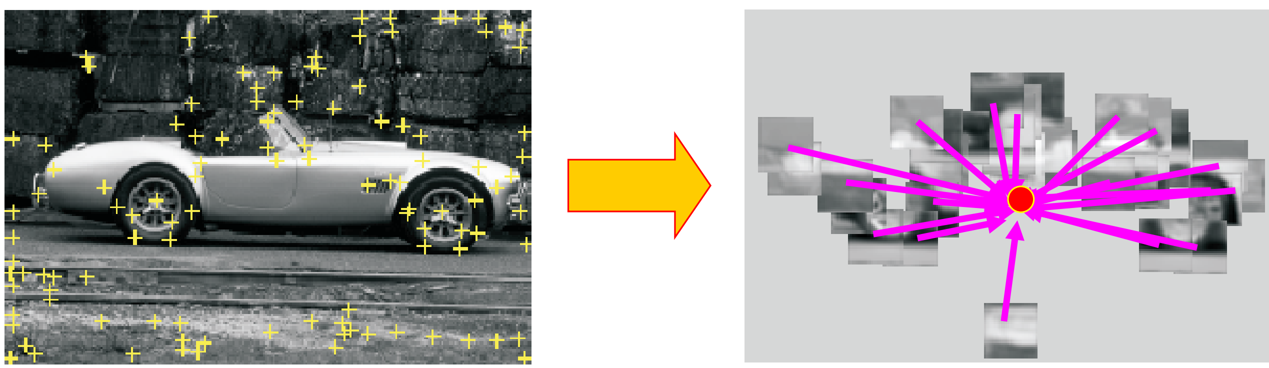
Vector Quantization

Represent a feature vector by its kind by

1. Finding the closest center
2. Recording its number (1…k) --- this is its code

Implicit shape models: Training

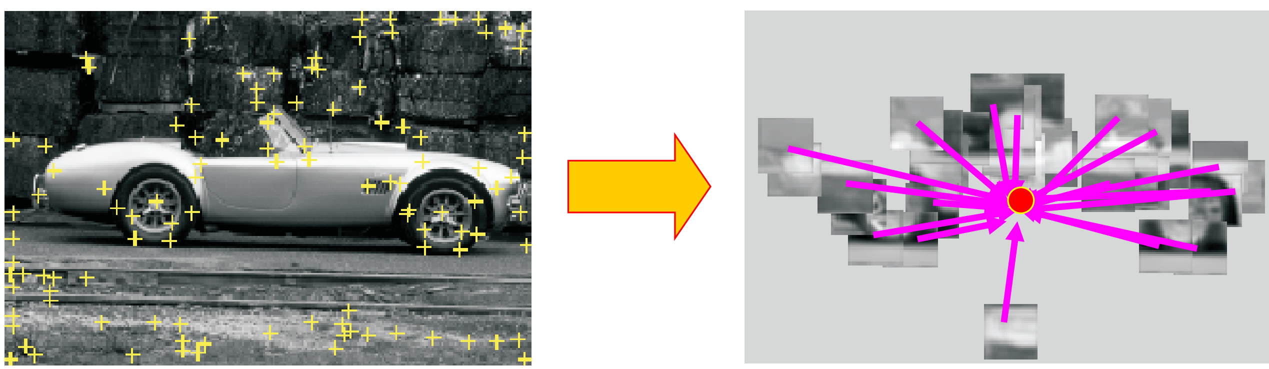
1. Build *codebook* of patches around extracted interest points using clustering
2. Map the patch around each interest point to closest codebook entry
3. For each codebook entry, store all positions it was found, relative to object center



Implicit shape models: Training

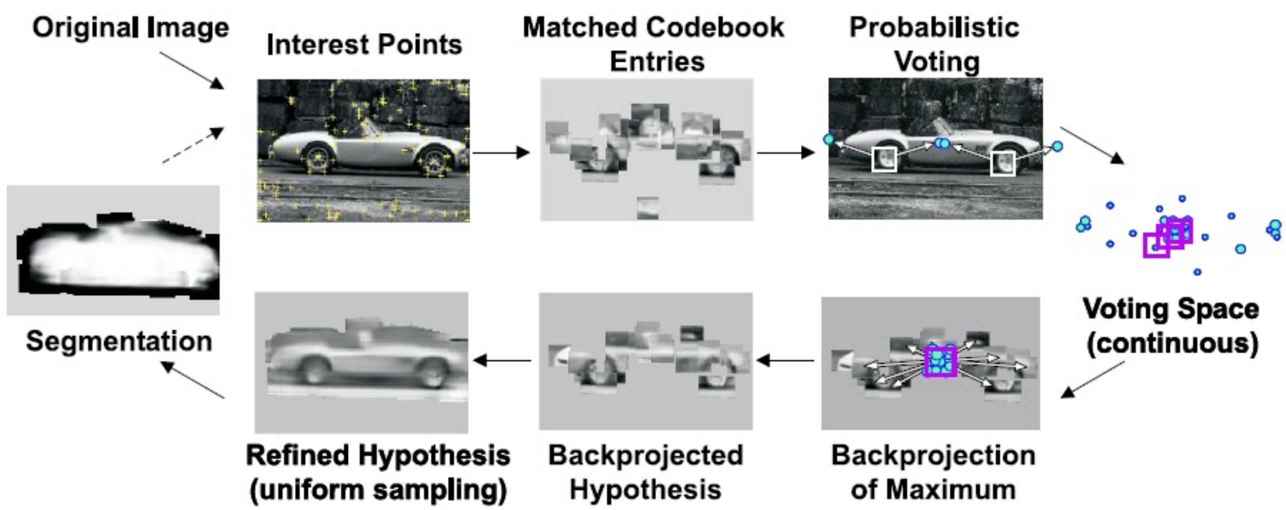
For each codebook entry, store all positions it was found, relative to object center

Recall you know location, orientation and scale for each codebook entry – so each ”knows” where the object center should be

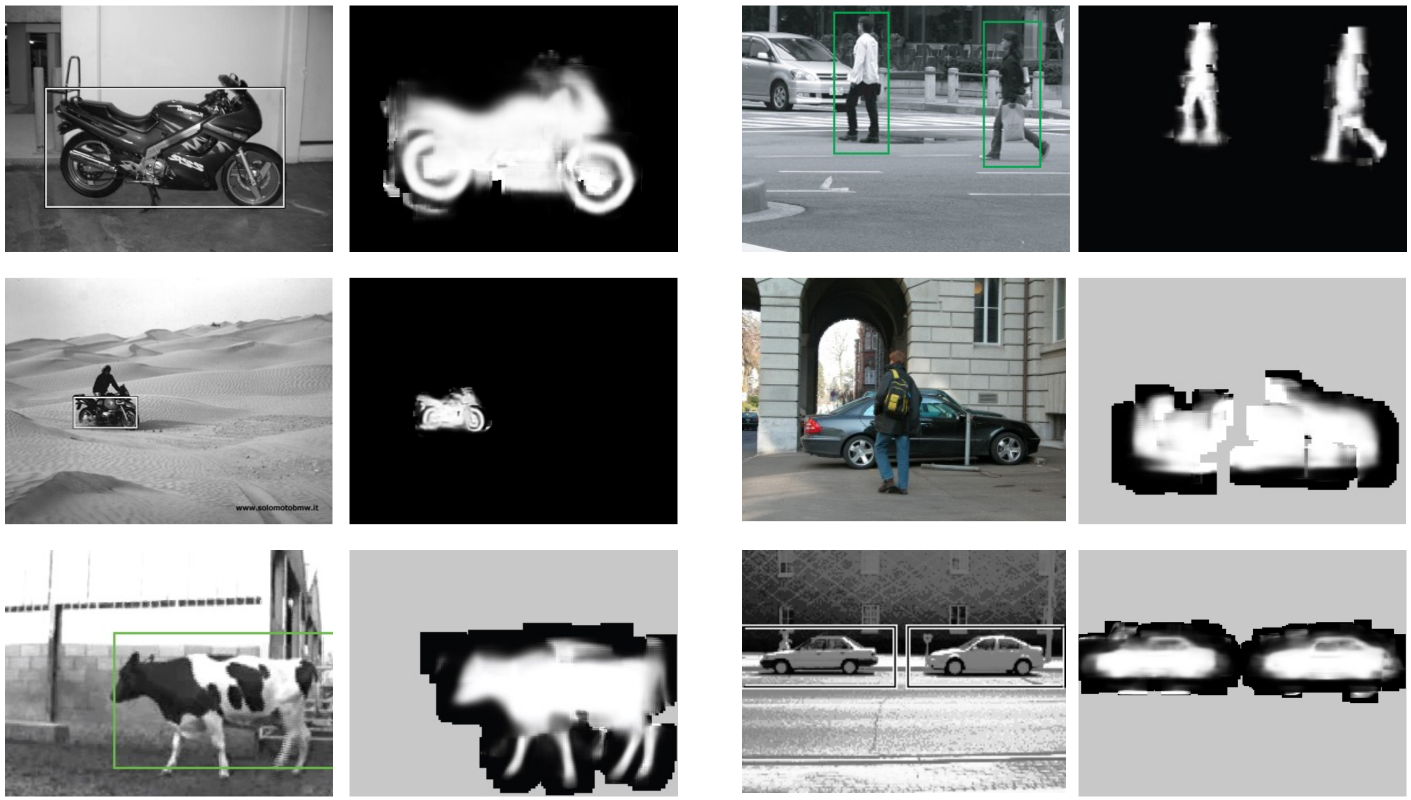


Implicit shape models: Testing

1. Given test image, extract patches, match to codebook entry
2. Cast votes for possible positions of object center
3. Search for maxima in voting space
4. Extract weighted segmentation mask based on stored masks for the codebook occurrences

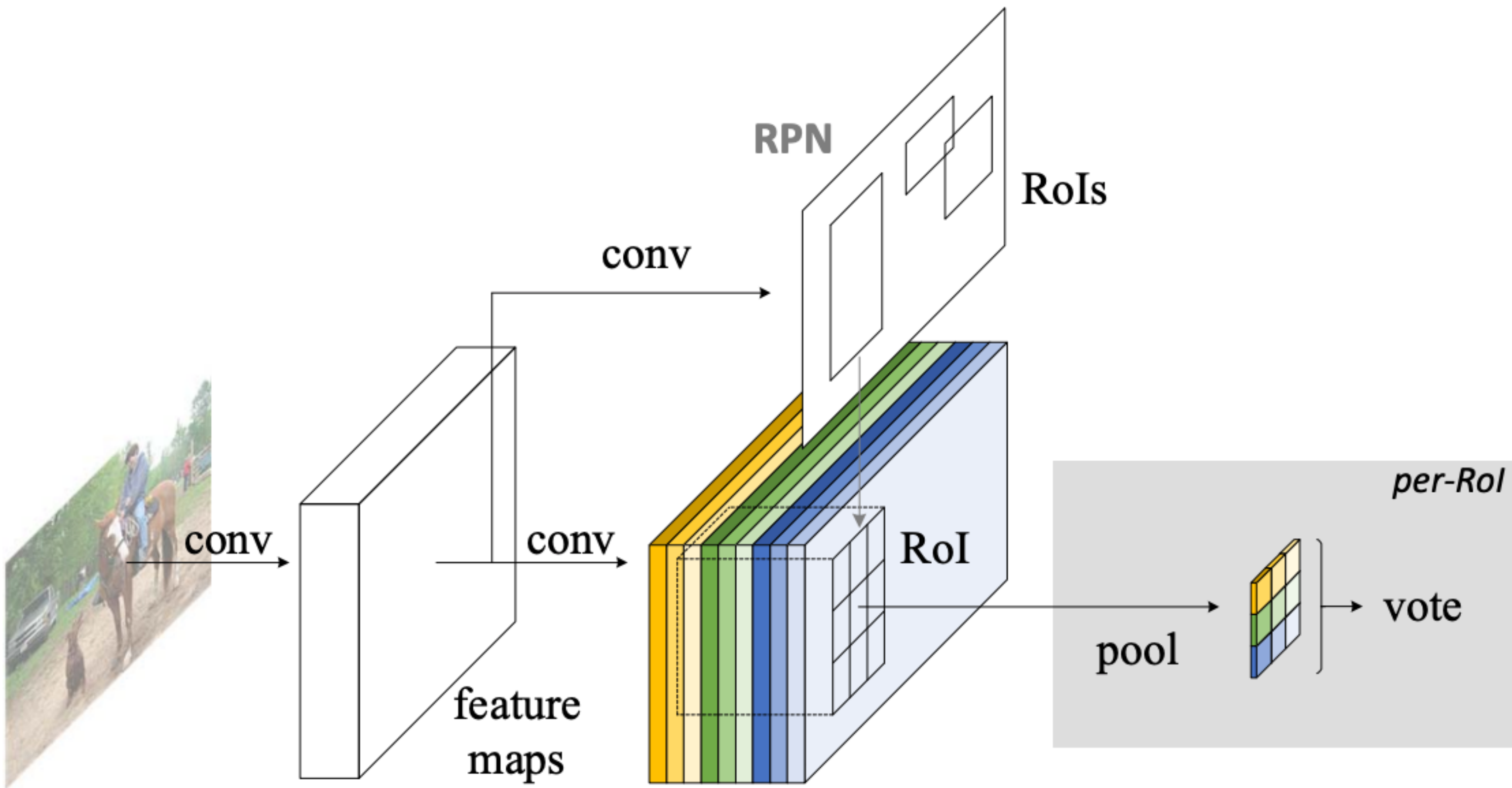


Additional examples



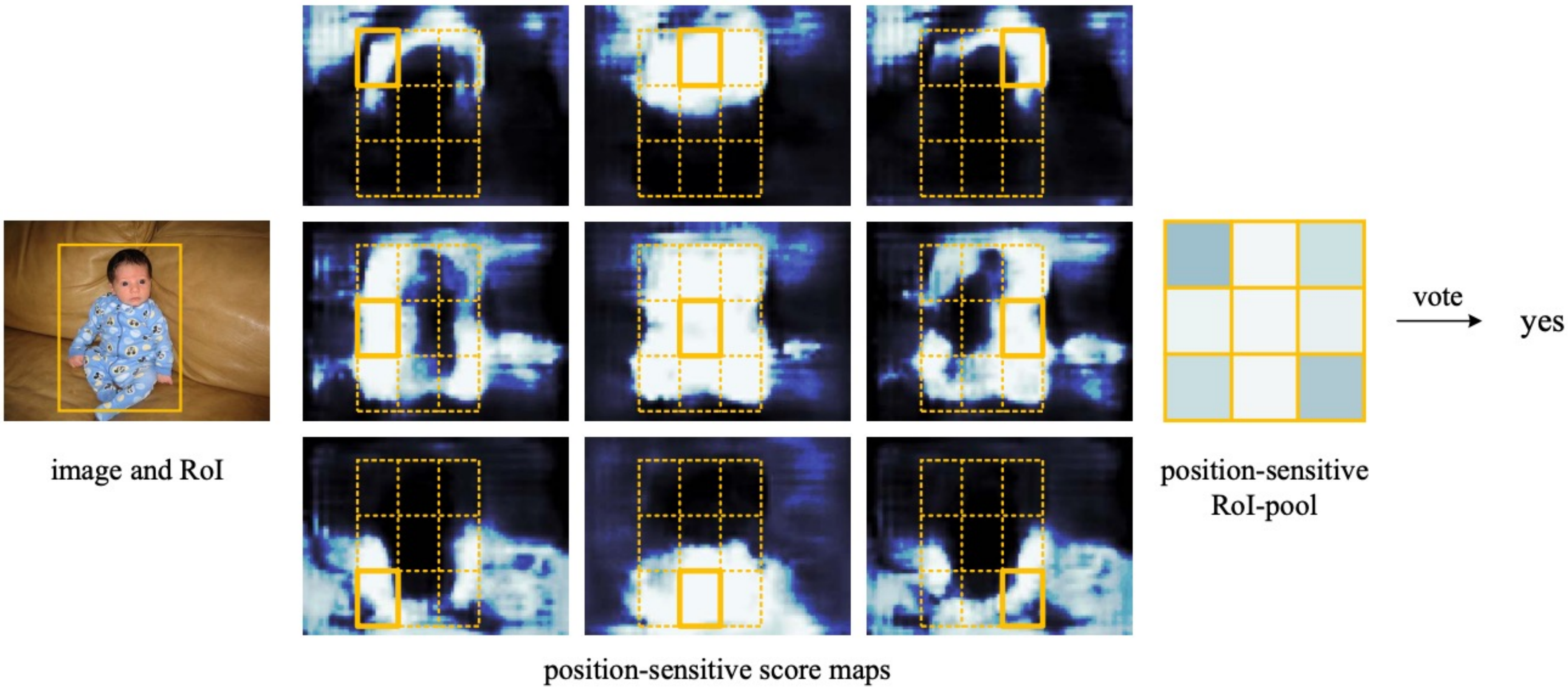
B. Leibe, A. Leonardis, and B. Schiele, Robust Object Detection with Interleaved Categorization and Segmentation, IJCV 77 (1-3), pp. 259-289, 2008.

A more recent example: Voting for detection



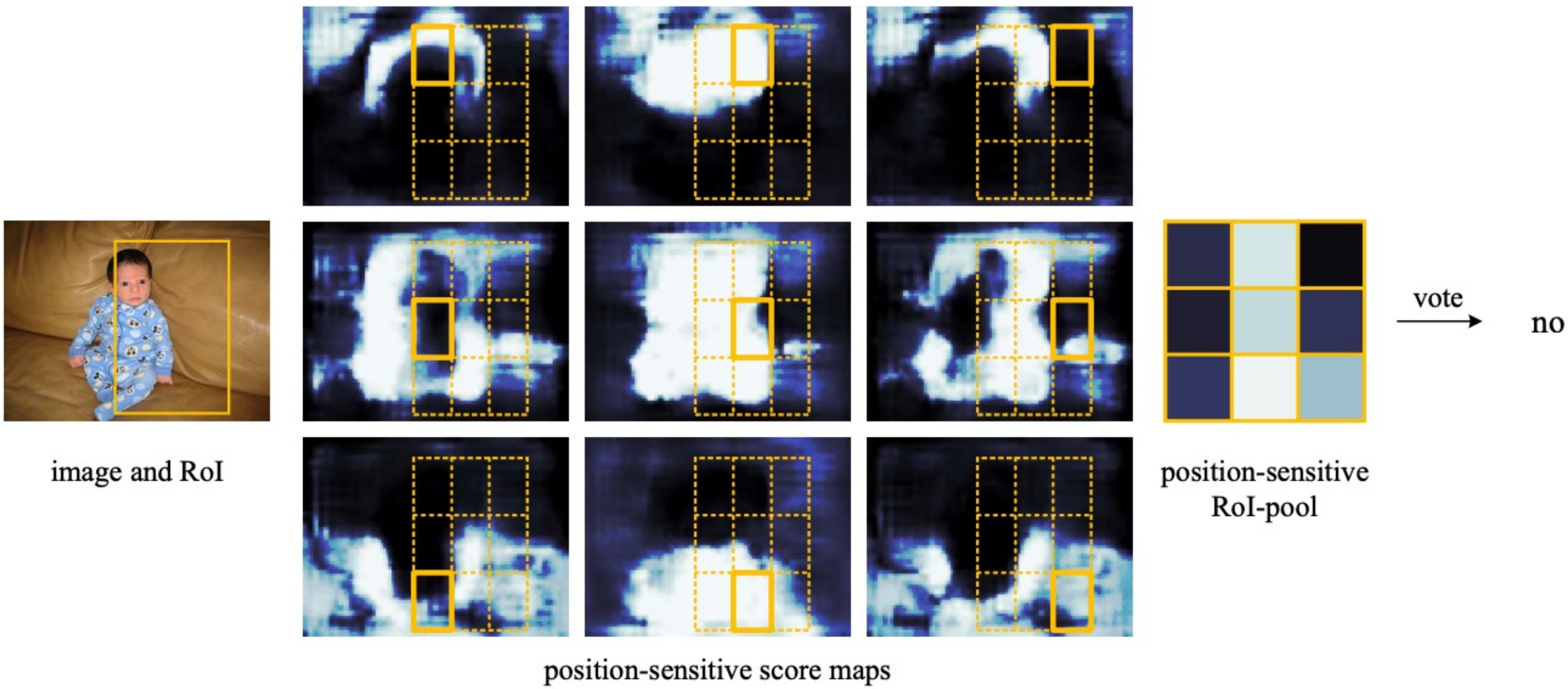
J. Dai et al. [R-FCN: Object Detection via Region-based Fully Convolutional Networks](). arXiv 2016

A more recent example: Voting for detection



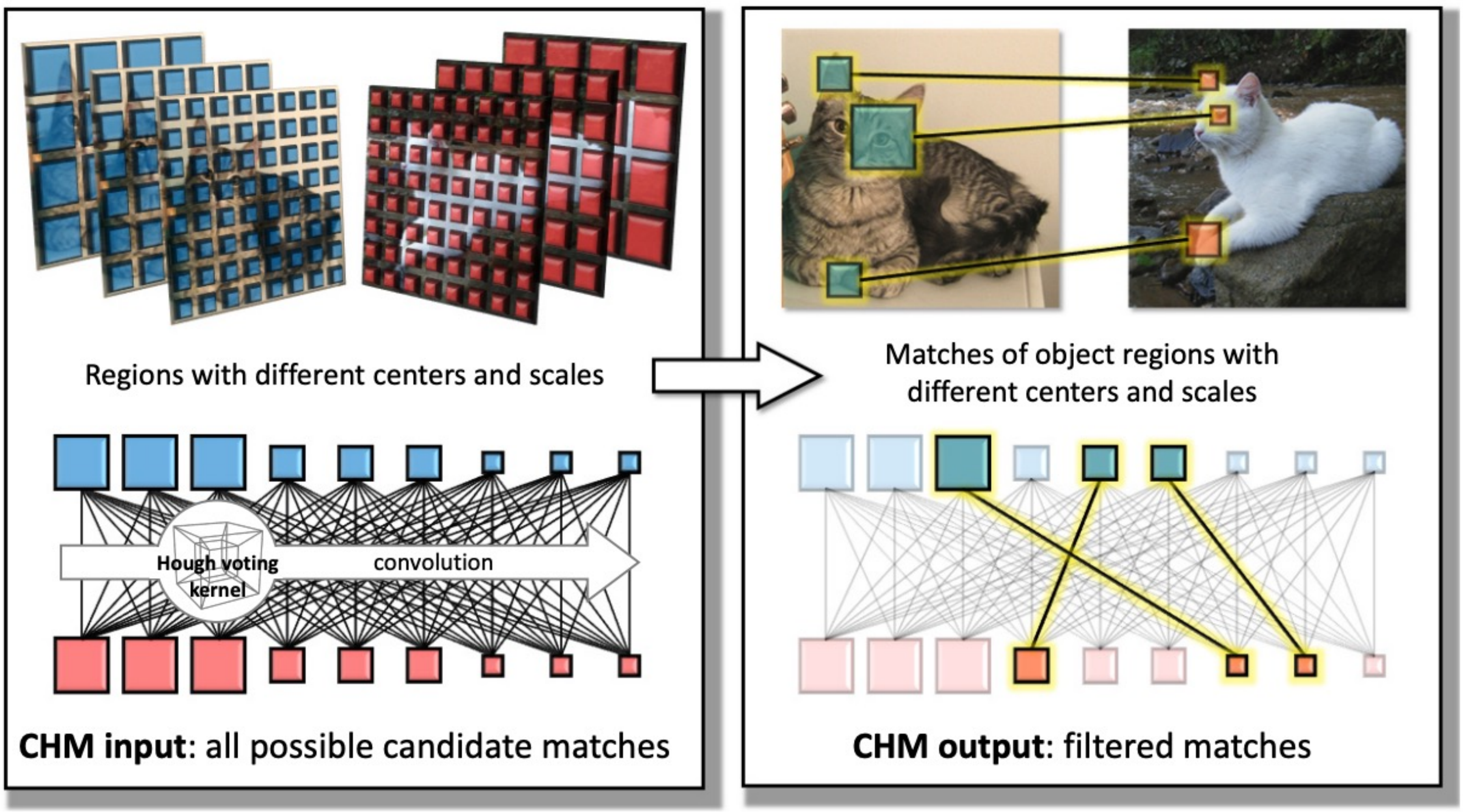
J. Dai et al. [R-FCN: Object Detection via Region-based Fully Convolutional Networks](). arXiv 2016

A more recent example: Voting for detection



J. Dai et al. [R-FCN: Object Detection via Region-based Fully Convolutional Networks](). arXiv 2016

Convolutional Hough matching networks



J. Min and M. Cho. Convolutional Hough matching networks. CVPR 2021