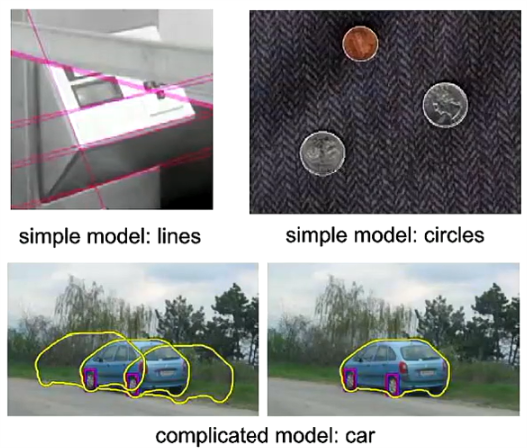
# Robust Model Fitting

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

## Fitting

* We’ve learned how to detect edges, corners and blobs. Now what?
* We’d like to form a higher-level, more compact representation of the features in the image by grouping multiple features according to a simple model
* Choose a parametric model to represent a set of features



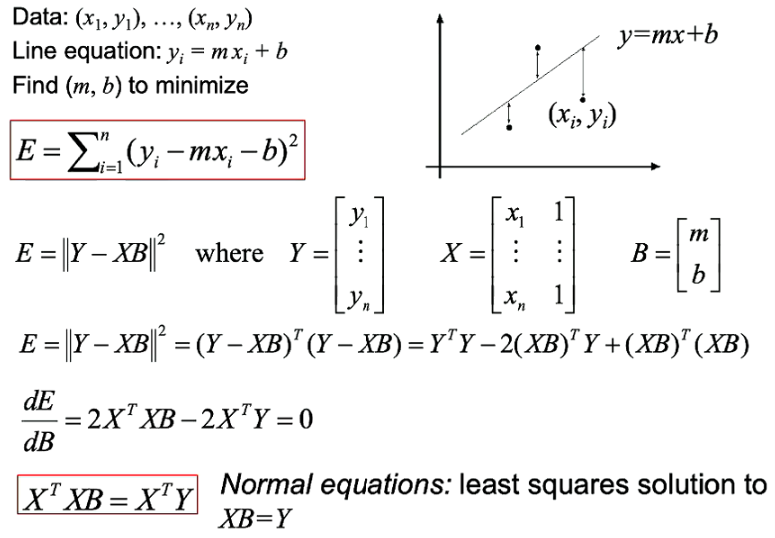
## Fitting: Issues

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

## Fitting: Overview

* If we know which points belong to the line, how do we find the optimal line parameters?
  + Least squares
* What if there are outliers?
  + Robust fitting, RANSAC
* What if there are many lines?
  + Voting methods: RANSAC, Hough transform
* What if we’re not even sure it’s a line?
  + Model selection (not covered)

## Least square line fitting

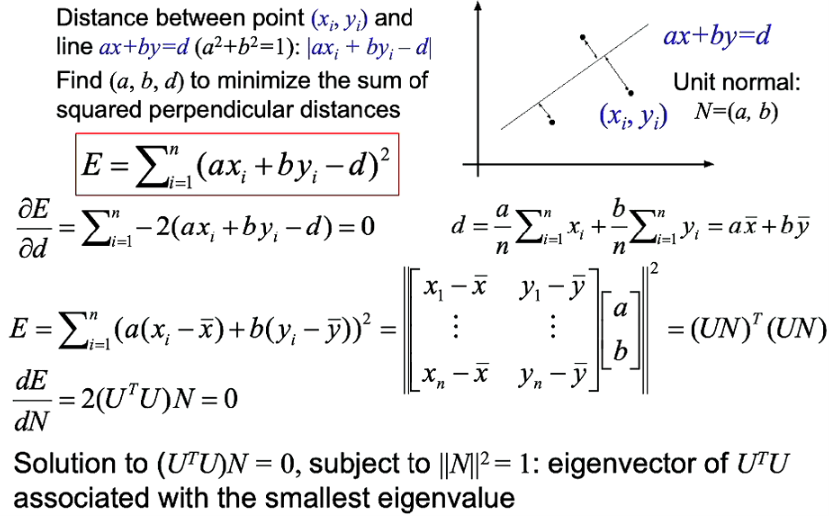


If the line is vertical, this is useless approach, since errors are huge.

## Problems with vertical least squares

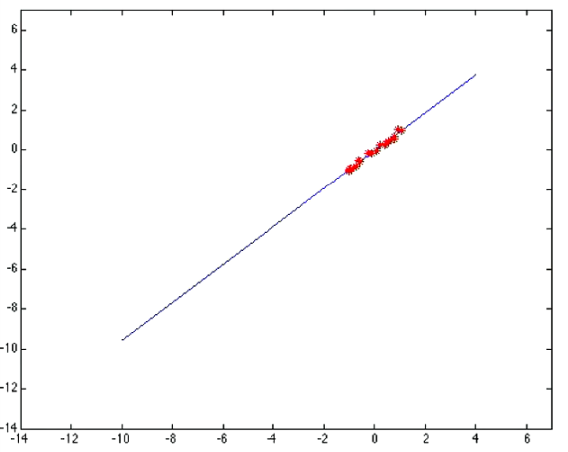
* Not rotation-invariant
* Fails completely for vertical lines

## Total least squares

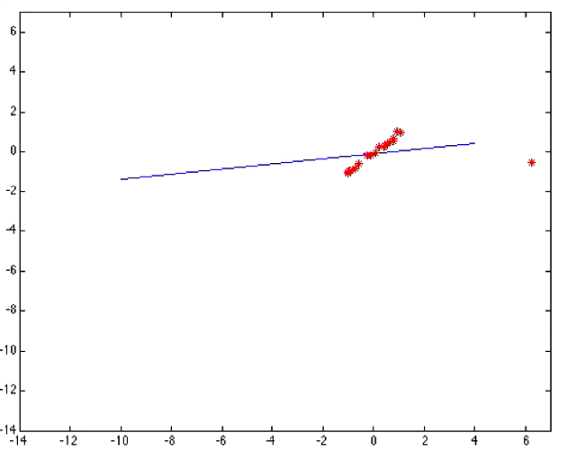


## Least squares: Robustness to noise

Least squares fit to the red points:



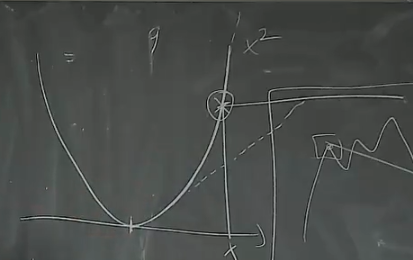
Least squares fit with an outlier:



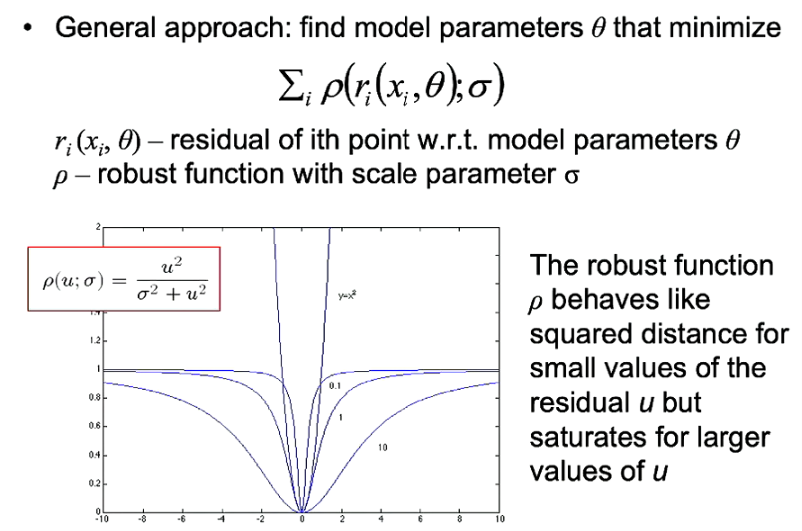
The error increases quadratically, that’s why the error is huge even due to a single sample

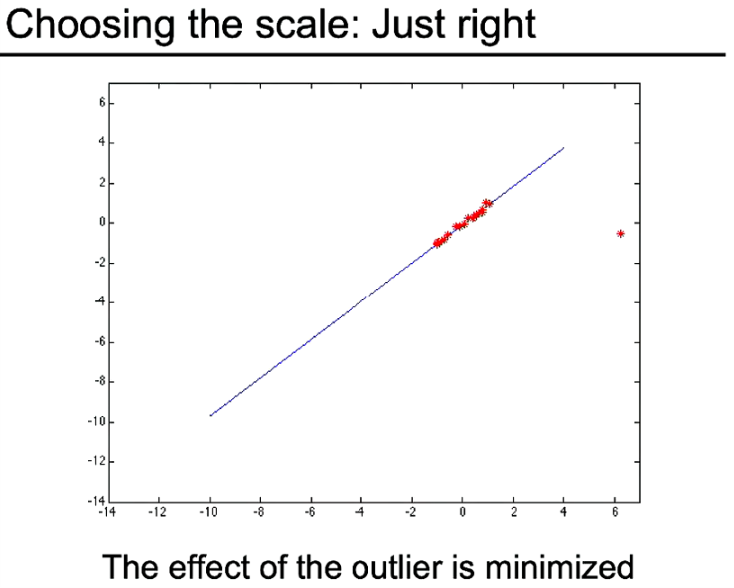
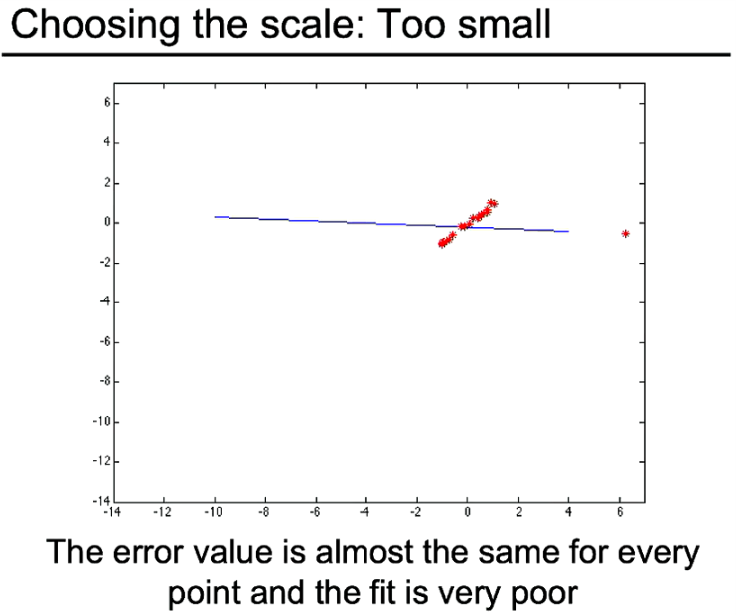
Problem: squared error heavily penalizes outliers

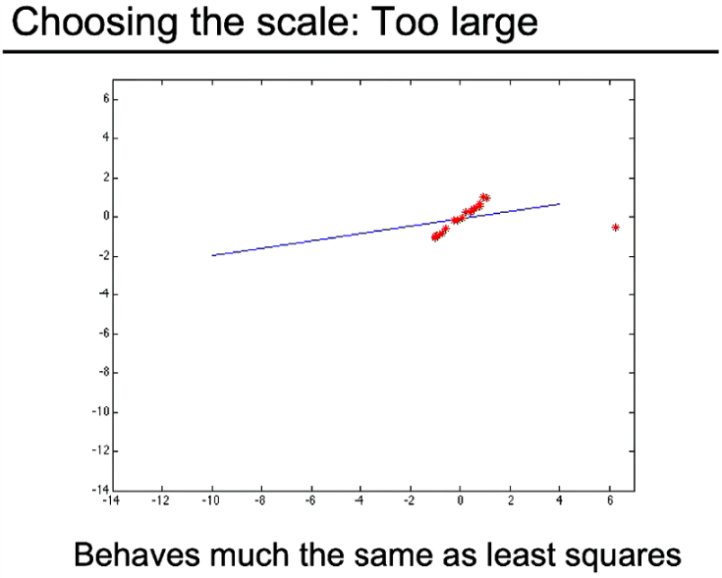
## Robust estimators



You saturate the error at some point, so it does not apply huge penalties. To solve the discontinuity, you smooth it.







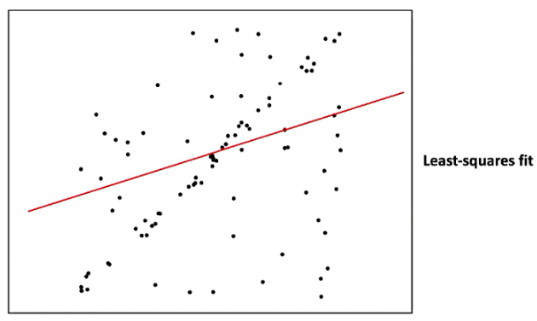
## Robust estimation: Details

* Robust fitting is a nonlinear optimization problem that must be solved iteratively
* Least squares solution can be used for initialization
* Scale of robust function should be chosen adaptively based on median residual

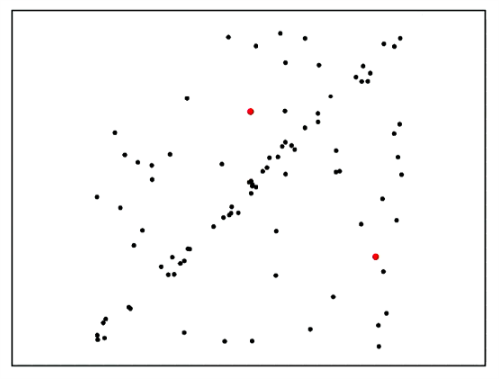
## RANSAC: RAndom SAmpled Consensus

* Robust fitting can deal with a few outliers, but what if we have many?
* It is a general framework for model fitting in presence of outliers
* Outline
  + Choose a small subset of points uniformly at random
  + Fit a model to that subset
  + Find all remaining points that are close to the model and reject the rest as outliers
  + Do this many times and choose the best model

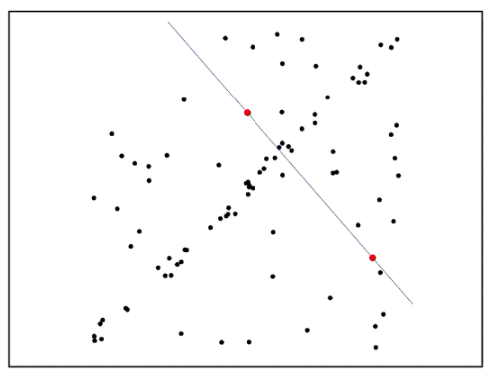
## RANSAC for line fitting example



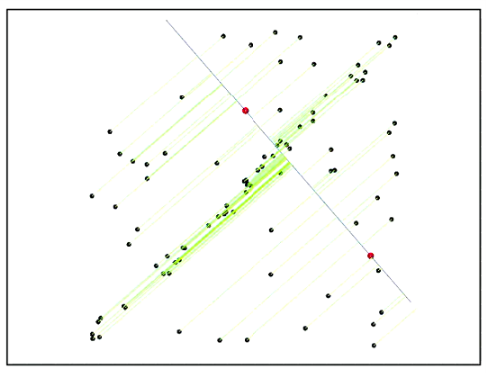
1. Randomly select minimal subset of points



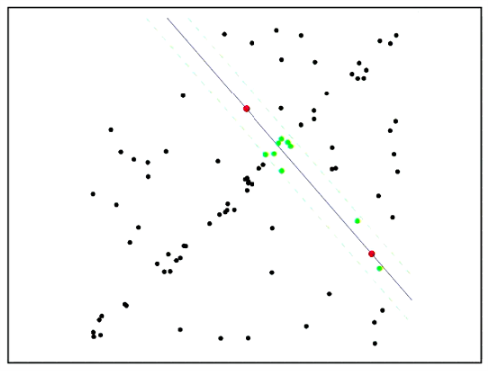
1. Hypothesize a model



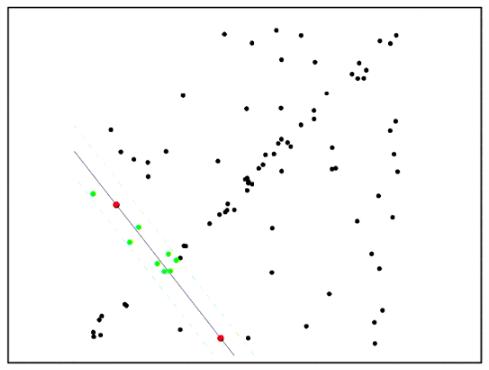
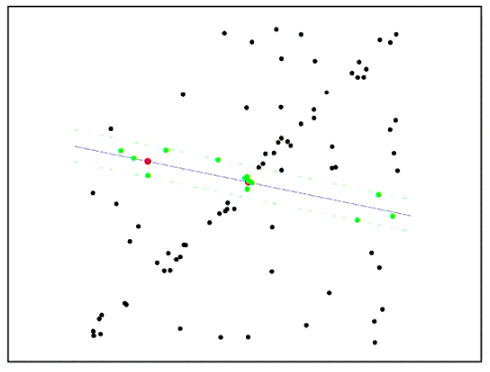
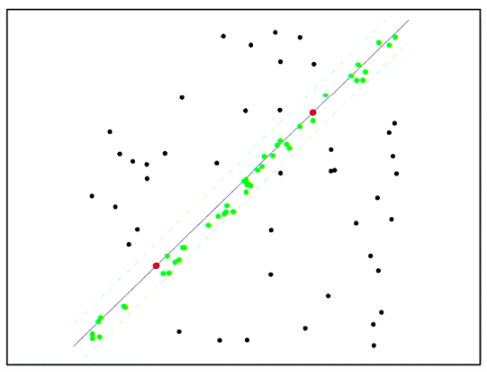
1. Compute error function



1. Select points consistent with model



1. Repeat hypothesize-and-verify loop



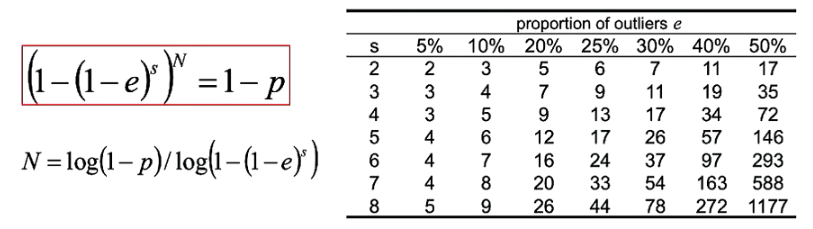
## RANSAC for line fitting

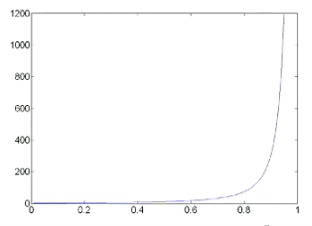
Repeat N times:

* Draw “s” points uniformly at random
* Fit line to these “s” points
* Find inliers to this line among the remaining points (i.e. points whose distance from the line is less than “t”) t = np.sqrt(3.84)\*2
* If there are “d” or more inliers, accept the line and refit using all inliers

## Choosing the parameters

* Initial number of points “s”
  + Typically, minimum number needed to fit the model
* Distance threshold “t”
  + Choose “t” so probability for inlier is “p” (e.g. 0.95)
  + Zero-mean Gaussian noise with std. dev.
* Number of samples “N”
  + Choose “N” so that, the probability “p”, at least one random sample is free from outliers (e.g. 0.99) (outlier ratio “e”)

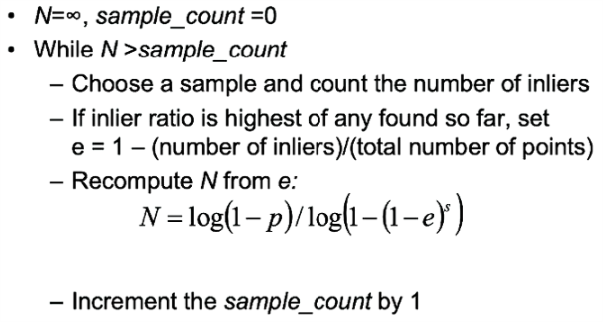




Inliers are the points that are kind of fit if a hypothetic line

## Adaptively determining the number of samples

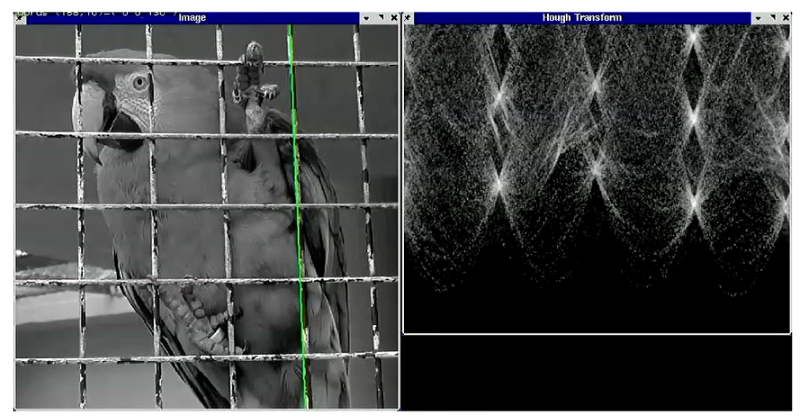
* Outlier ratio “e” is often unknown a priori, so pick worst case, e.g. 50%, and adapt if more inliers found, e.g. 80% would yield e = 0.2
* Adaptive procedure:



## RANSAC pros and cons

* Pros
  + Simple and general
  + Applicable to many different problems
  + Often works well in practice
* Cons
  + Lots of parameters to tune
  + Does not work well for low inlier rations (too many iterations or can fail completely)
  + Can’t always get a good initialization of the model based on the minimum number of samples

## Fitting: The Hough Transform

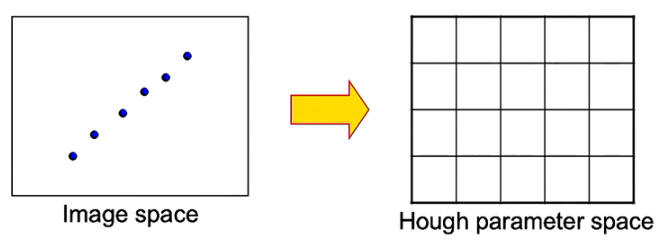


## Voting schemes

* Let each feature vote for all the models that are compatible with it
* The noise features will not vote consistently for any single model, hopefully
* Missing data does not matter as long as there are enough features remaining to agree on a good model

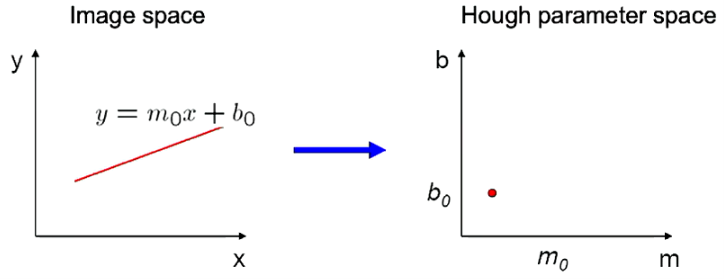
## Hough transform

* An early type of voting scheme
* General outline:
  + Discrete parameter space into bins
  + For each feature point in the image, put a vote in every bin in the parameters space that could have generated this point
  + Find bins that have the most votes

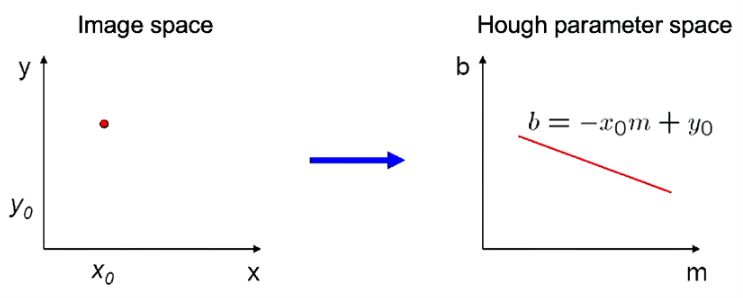


## Parameter space representation

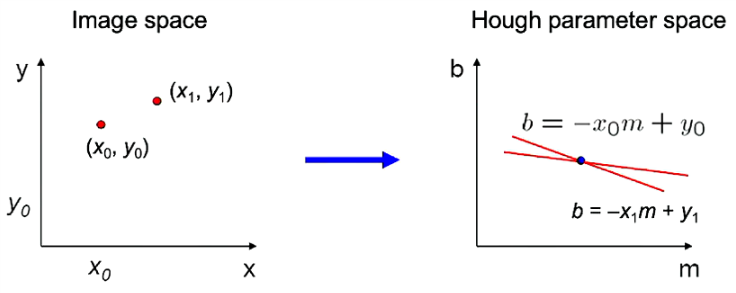
* A line in the image corresponds to a point in the Hough space



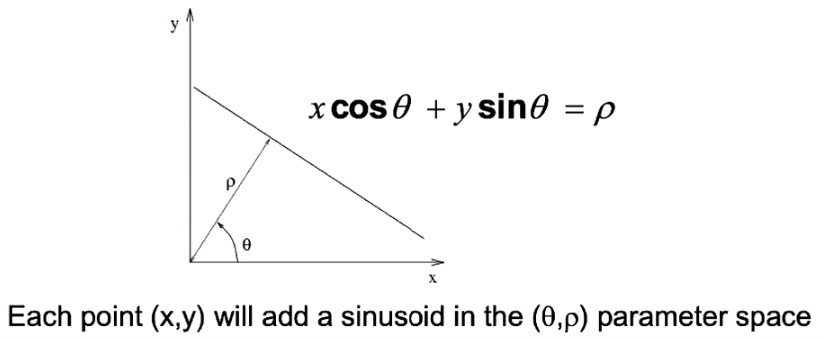
* A point in the image corresponds to a line in the Hough space



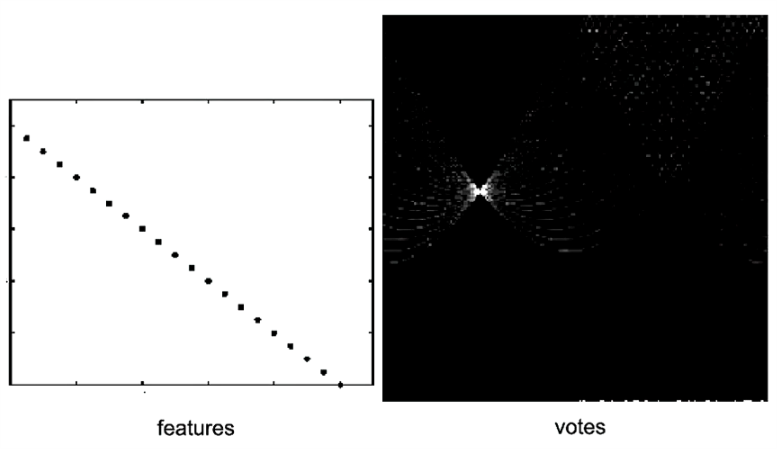
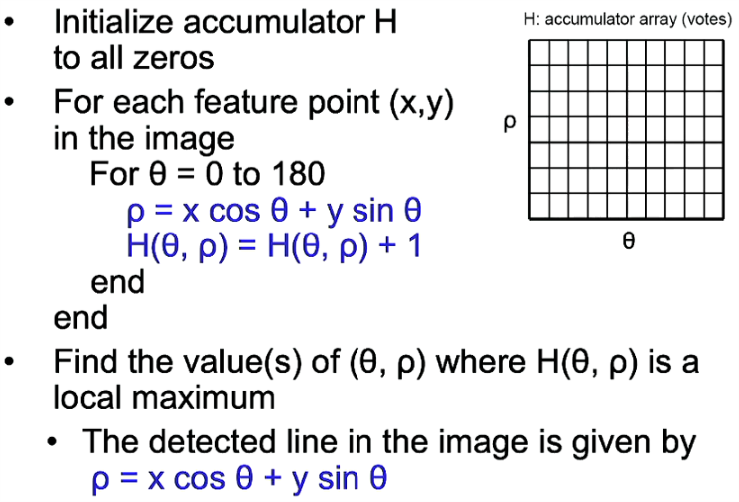
* Two points corresponds to two lines which will intersect in a point voted by the two points



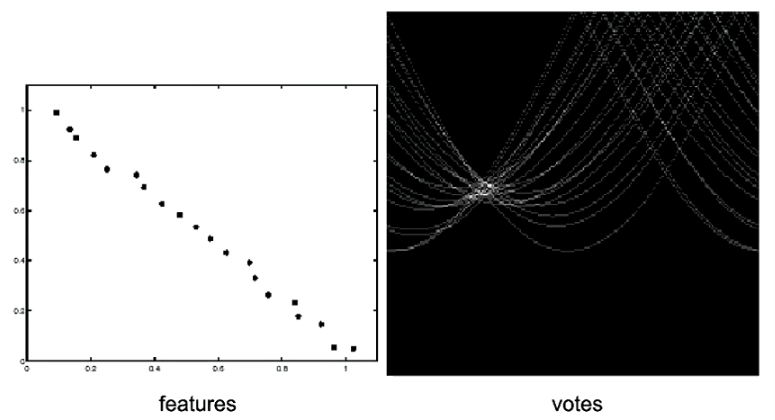
* Problems with the (m,b) space:
  + Unbounded parameter domains
  + Vertical lines require infinite m
* Alternative: polar representation



## Algorithm outline

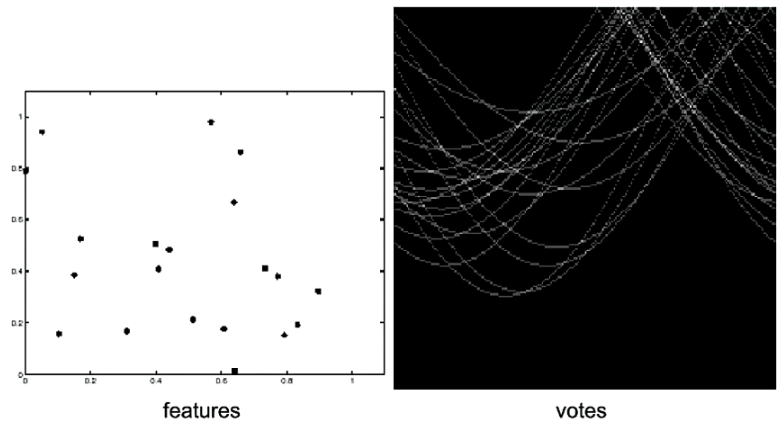


## Effect of noise



Peak gets fuzzy and hard to locate

## Random points



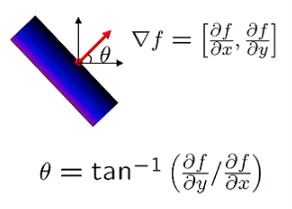
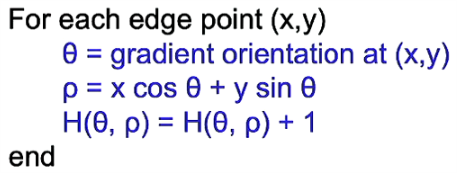
Uniform noise can lead to spurious peaks in the array

## Dealing with noise

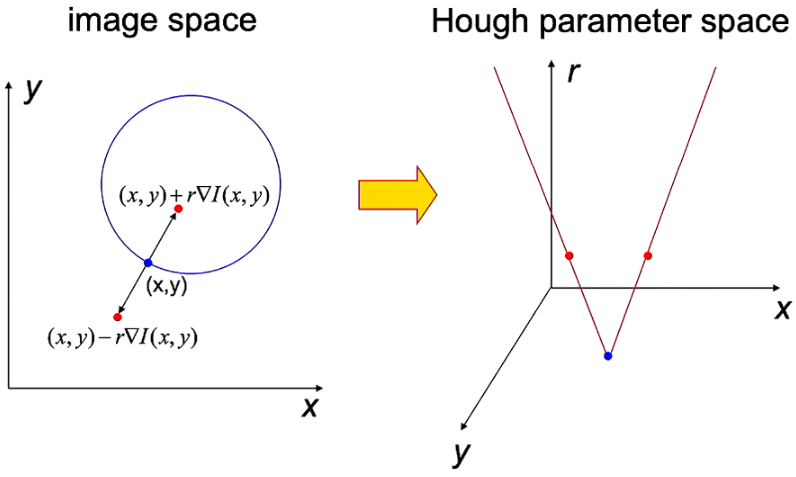
* Choose a good grid / discretization
  + Too coarse: large votes obtained when too many different lines correspond to a single bucket
  + Too fine: miss lines because some points that are not exactly collinear cast votes for different buckets
* Increment neighboring bins (smoothing in accumulator array)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

## Incorporating image gradients

* 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:



## Hough transform for circles



* Conceptually equivalent procedure: for each (x,y,r), draw the corresponding circle in the image and compute its “support”

## 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
* Cons
  + Complexity of search time increases exponentially with the number of model parameters
  + Non-target shapes can produce spurious peaks in parameters spaces
  + It’s hard to pick a good grid size