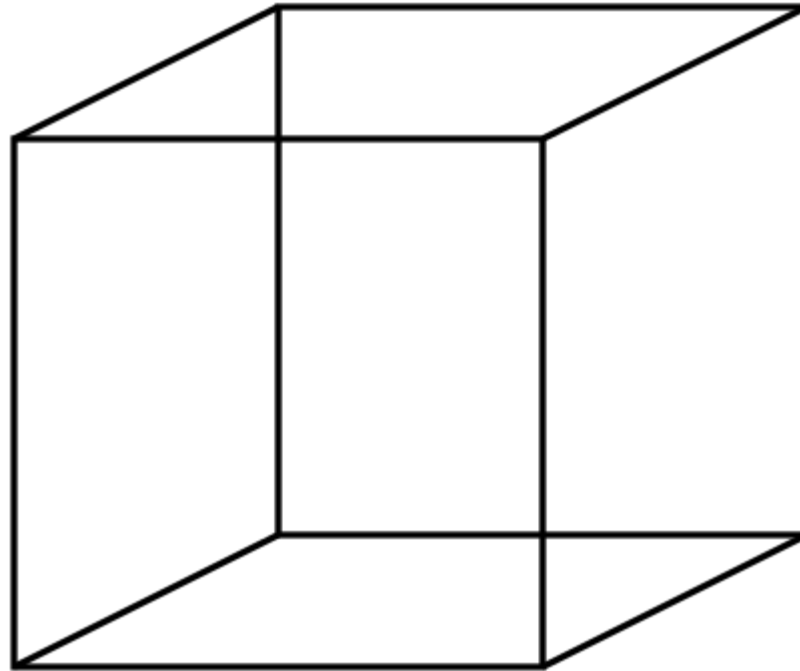
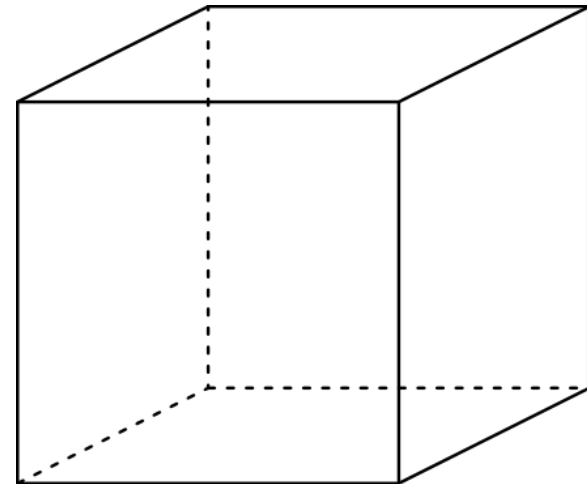
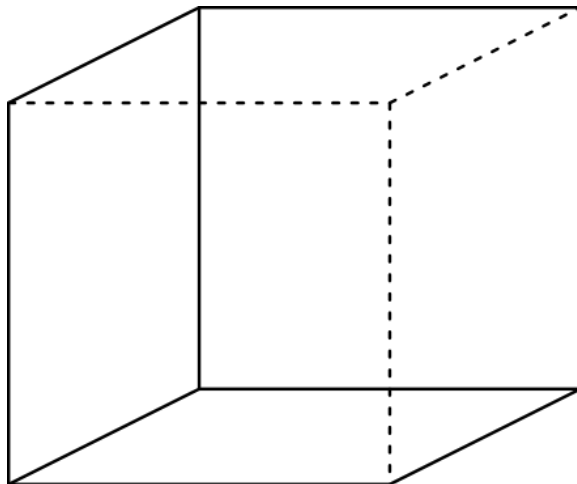
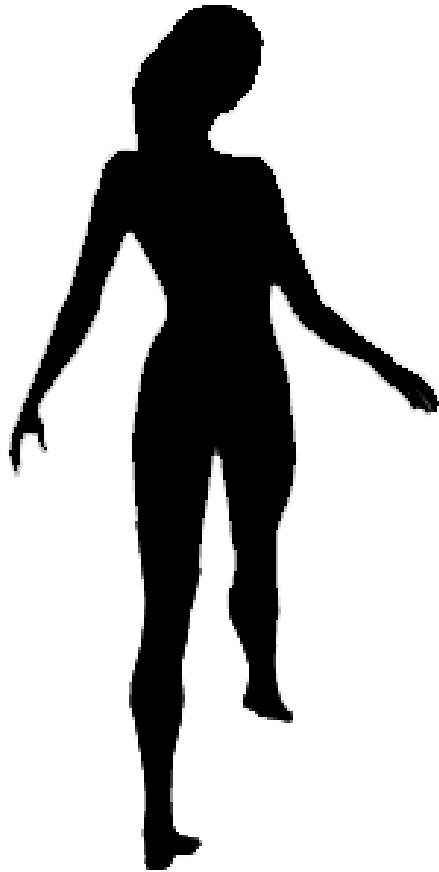


Multi-stable Perception



Necker Cube





Spinning dancer illusion, Nobuyuki Kayahara



Feature Matching and Robust Fitting

Read Szeliski 7.4.2 and 2.1

Computer Vision

James Hays

Project 2

Project 2: SIFT Local Feature Matching

CS 4476
Fall 2023

Brief

- Due: Check [Canvas](#) for up to date information
- Project materials including report template: [Project 2](#)
- Hand-in: through [Gradescope](#)
- Required files: <your_gt_username>.zip, <your_gt_username>_proj2.pdf

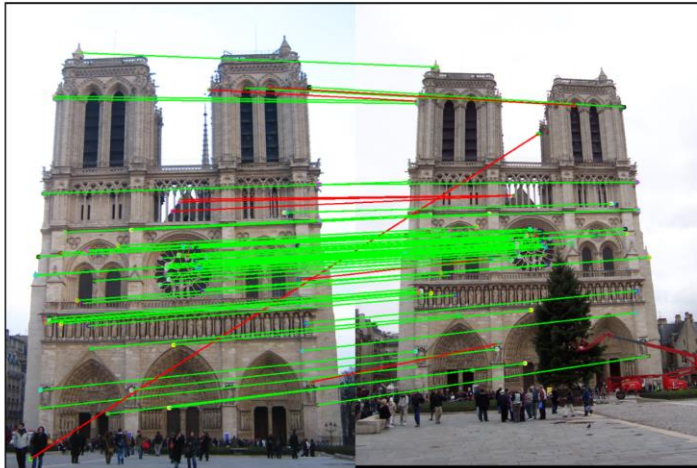


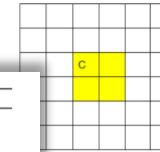
Figure 1: The top 100 most confident local feature matches from a baseline implementation of project 2. In this case, 89 were correct (lines shown in green), and 11 were incorrect (lines shown in red).

Overview

The goal of this assignment is to create a local feature matching algorithm using techniques described in Szeliski chapter 7.1. The pipeline we suggest is a simplified version of the famous **SIFT** pipeline. The pipeline we suggest is a simplified version of the famous **SIFT** pipeline. The pipeline we suggest is a simplified version of the famous **SIFT** pipeline.

Algorithm 1: Harris Corner Detector

Compute the horizontal and vertical derivatives I_x and I_y of the image by convolving the original image with a Sobel filter;
Compute the three images corresponding to the outer products of these gradients. (The matrix A is defined as follows.)
Compute the determinant D and the trace T of the matrix A .
Compute the corner response $R = \frac{D^2}{T^3}$.
Threshold R and report them as detected feature point locations.



Now, the yellow cells could all be considered the center. Please enter throughout this project.

In this part, you will have to code feature distances, and `match_features_ratio_test()` to perform feature lists.

part4_sift_descriptor.py

As described in the lecture materials and Szeliski 7.1.2. We'll use "Root SIFT" from a 2012 CVPR paper (linked here) to get a SIFT descriptor.

An unweighted 1D histogram with 3 bins could be defined over half-open intervals $h = [2, 1, 2]$.

3 bins and bin edges has each item weighted by some value. The bin edges are $w = [2, 3, 1, 0, 0]$, and the same bin edges histogram weight at a pixel is the magnitude of the image gradient.

Implement the following:

`get_gradients()`: Retrieves gradient magnitudes and orientations of the image.

`concatenate_histograms()`: Retrieves a feature consisting of concatenated histograms.

`get_descriptor()`: Retrieves a feature from a single point.

`match_features_ratio_test()`: Compares feature vectors corresponding to our interest points from an image.

The pipeline on the Notre Dame image is at least 80%. Note that `epsilon` (close to 0) and think about why this could be happening.

Project Exploration

- How big should the window around each feature be?
- How many orientations should each histogram have? Modify `get_gradients()` and report the results.

- Take two images of a building or structure near you. Save them in the `additional_data/` folder of the project and run your SIFT pipeline on them. Analyze the results - why do you think our pipeline may have performed well or poorly for the given image pair? Is there anything about the building that is helpful or detrimental to feature matching?

do a project report using the template slides provided. Remove any slides, as this will affect the grading process. In the report you will describe your algorithm and any other way. Then you will show and discuss the results of the experiment for what you should include in your report. A good conclusion from the experiments. You must convert the report into PDF and then assign each PDF page to the relevant question.

Use the slides given in the template deck to describe your implementation. You will receive full credit for your extra credit implementations.

part2.py

The starter code provided in the starter code includes file handling, visualizations, and placeholder versions of the three functions listed below.

The starter code includes truth evaluation in the starter code as well. `evaluate_matches()` will return True or False based on hand-provided matches. The starter code includes two other image pairs (Mount Rushmore and Notre-Dame) and uncommenting the appropriate lines in `project-2.ipynb`.

You should use your performance according to `evaluate_matches()` to compare your implementation to the initial Notre Dame image. The starter code will give you a baseline.

Helper functions

`concatenate()`, `np.flipplr()`, `np.flipud()`, `np.histogram()`, `np.meshgrid()`, `np.reshape()`, `np.sort()`.

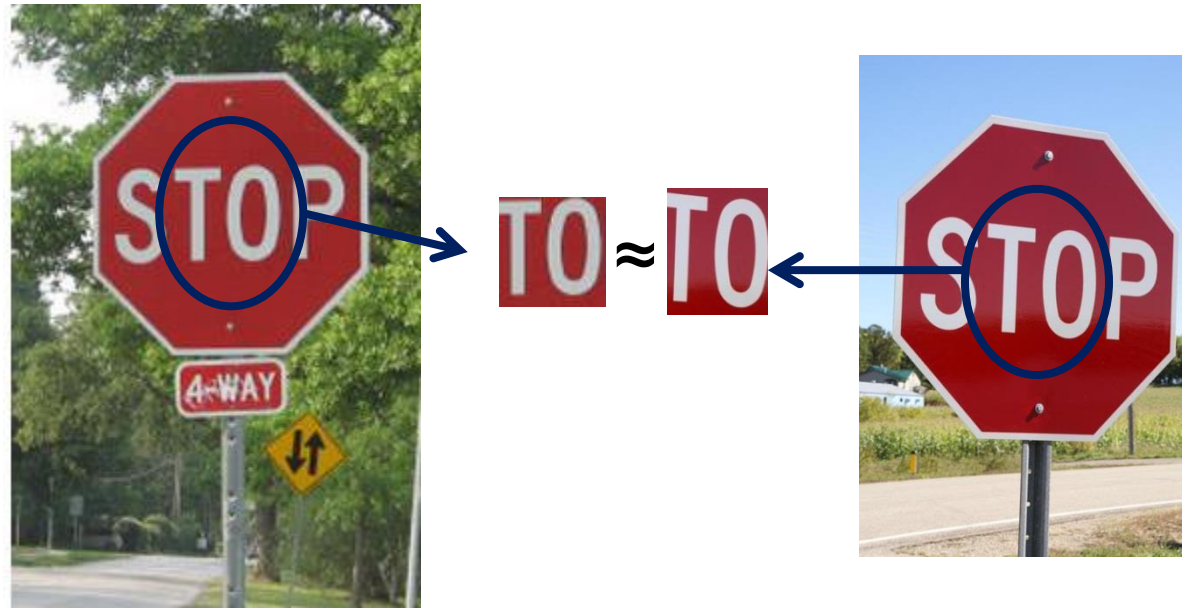
`torch.from_numpy()`, `torch.median()`, `torch.nn.functional.conv2d()`, `torch.nn.Parameter`, `torch.stack()`.

For visualization, you might find `torch.meshgrid`, `torch.norm`, `matplotlib.pyplot.imshow`.

Please use `torch.nn.Conv2d` or `torch.nn.functional.conv2d` from other libraries (e.g., `cv.filter2D()`, `scipy.ndimage`).

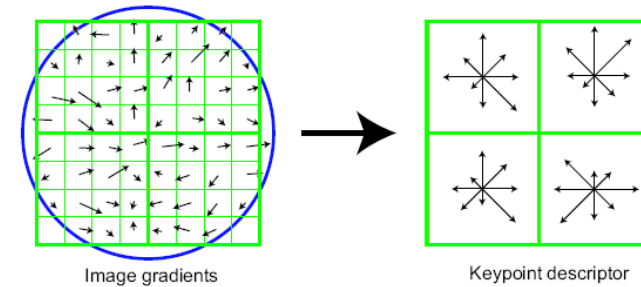
This section: correspondence and alignment

- Correspondence: matching points, patches, edges, or regions across images

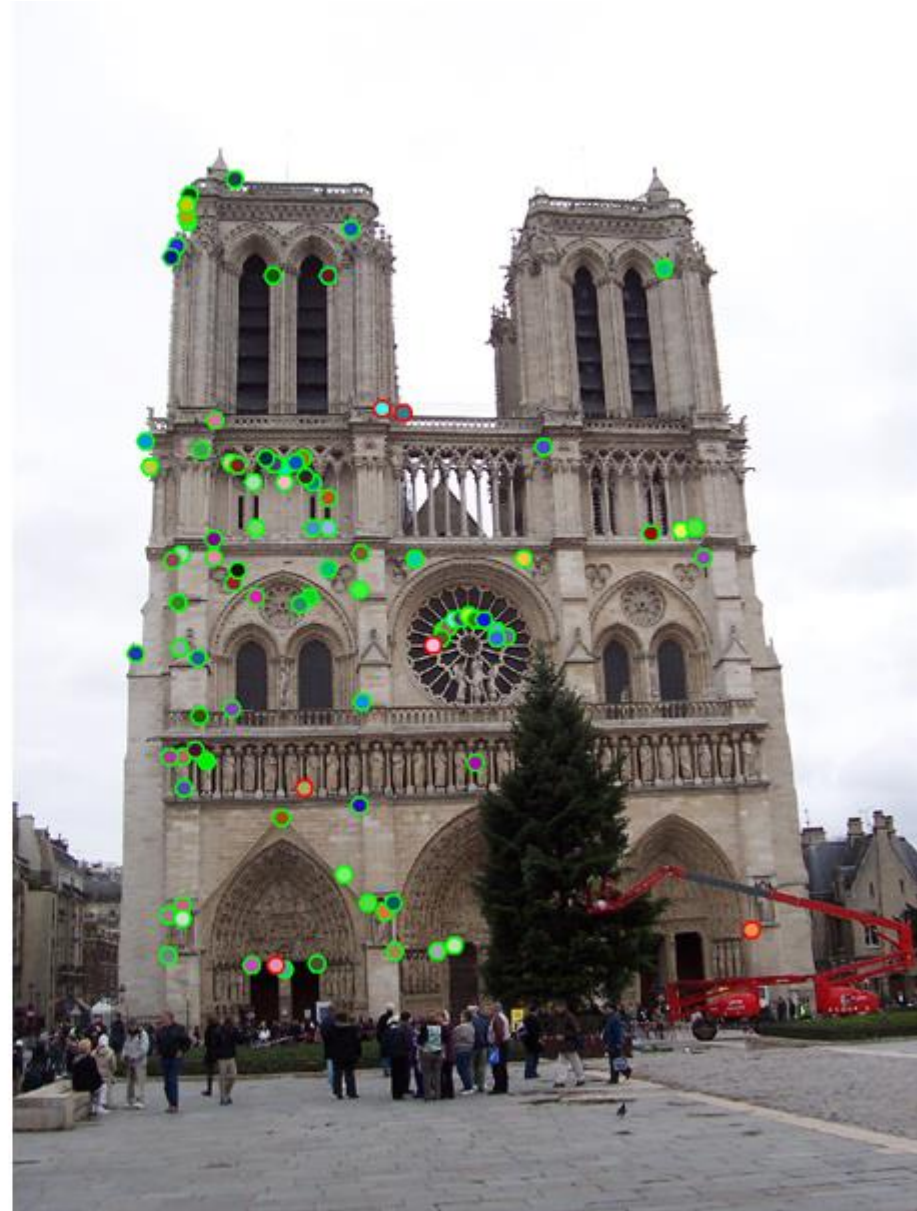
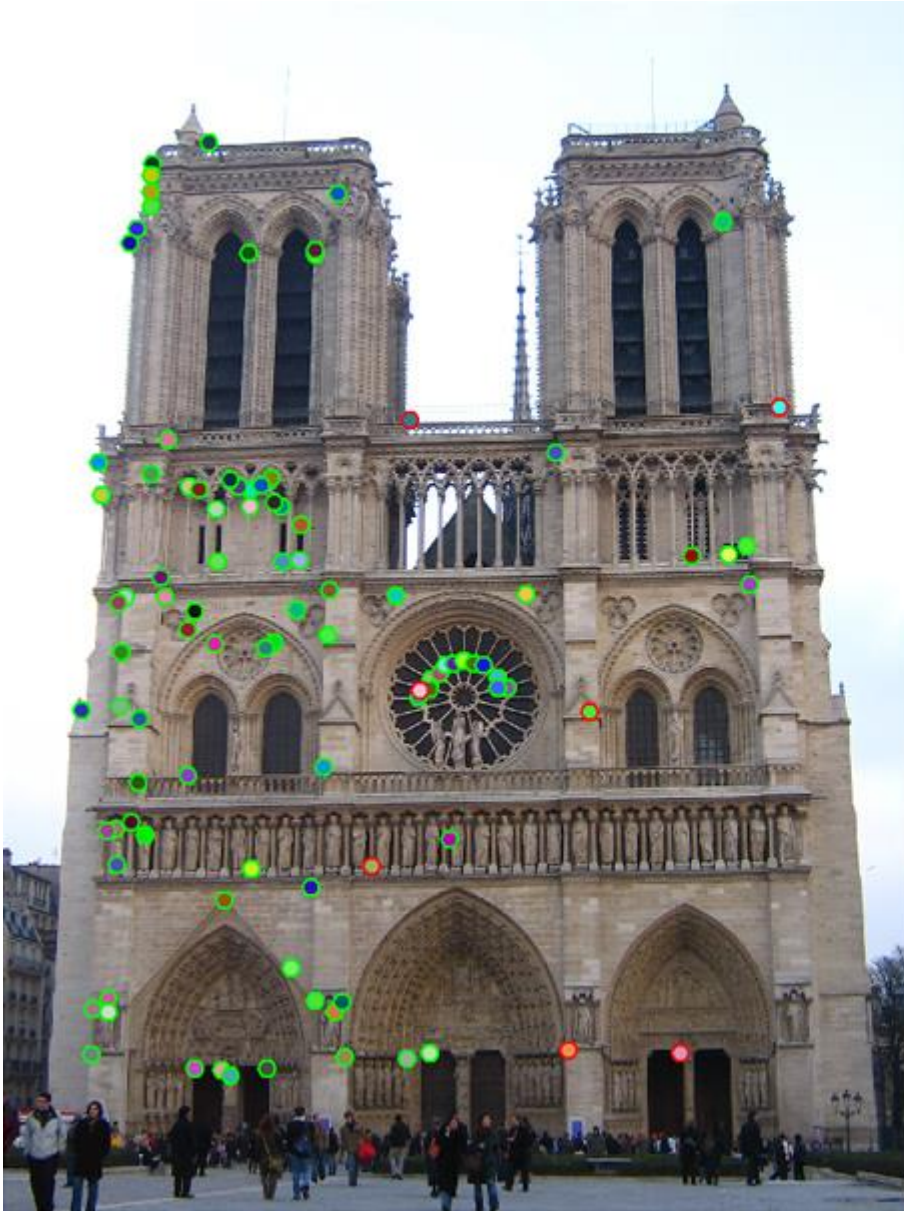


Review: Local Descriptors

- Most features can be thought of as templates, histograms (counts), or combinations
- The ideal descriptor should be
 - Robust and Distinctive
 - Compact and Efficient
- Most available descriptors focus on edge/gradient information
 - Capture texture information
 - Color rarely used



Can we refine this further?



Fitting: find the parameters of a model that best fit the data

Alignment: find the parameters of the transformation that best align matched points

Fitting and Alignment

- Design challenges
 - Design a suitable **goodness of fit** measure
 - Similarity should reflect application goals
 - Encode robustness to outliers and noise
 - Design an **optimization** method
 - Avoid local optima
 - Find best parameters quickly

Fitting and Alignment: Methods

- Global optimization / Search for parameters
 - Least squares fit
 - Robust least squares
 - Other parameter search methods
- Hypothesize and test
 - Generalized Hough transform
 - RANSAC

Fitting and Alignment: Methods

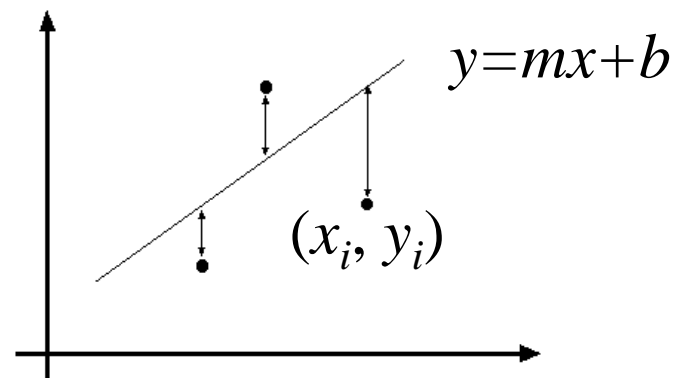
- Global optimization / Search for parameters
 - Least squares fit
 - Robust least squares
 - Other parameter search methods
- Hypothesize and test
 - Generalized Hough transform
 - RANSAC

Simple example: Fitting a line

Least squares line fitting

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

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



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

$$= \mathbf{y}^T \mathbf{y} - 2(\mathbf{A}\mathbf{p})^T \mathbf{y} + (\mathbf{A}\mathbf{p})^T (\mathbf{A}\mathbf{p})$$

Matlab: $\mathbf{p} = \mathbf{A} \setminus \mathbf{y};$

$$\frac{dE}{d\mathbf{p}} = 2\mathbf{A}^T \mathbf{A}\mathbf{p} - 2\mathbf{A}^T \mathbf{y} = 0$$

Python: $\mathbf{p} =$
`numpy.linalg.lstsq(A, y)`

$$\mathbf{A}^T \mathbf{A}\mathbf{p} = \mathbf{A}^T \mathbf{y} \Rightarrow \mathbf{p} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y}$$

Least squares (global) optimization

Good

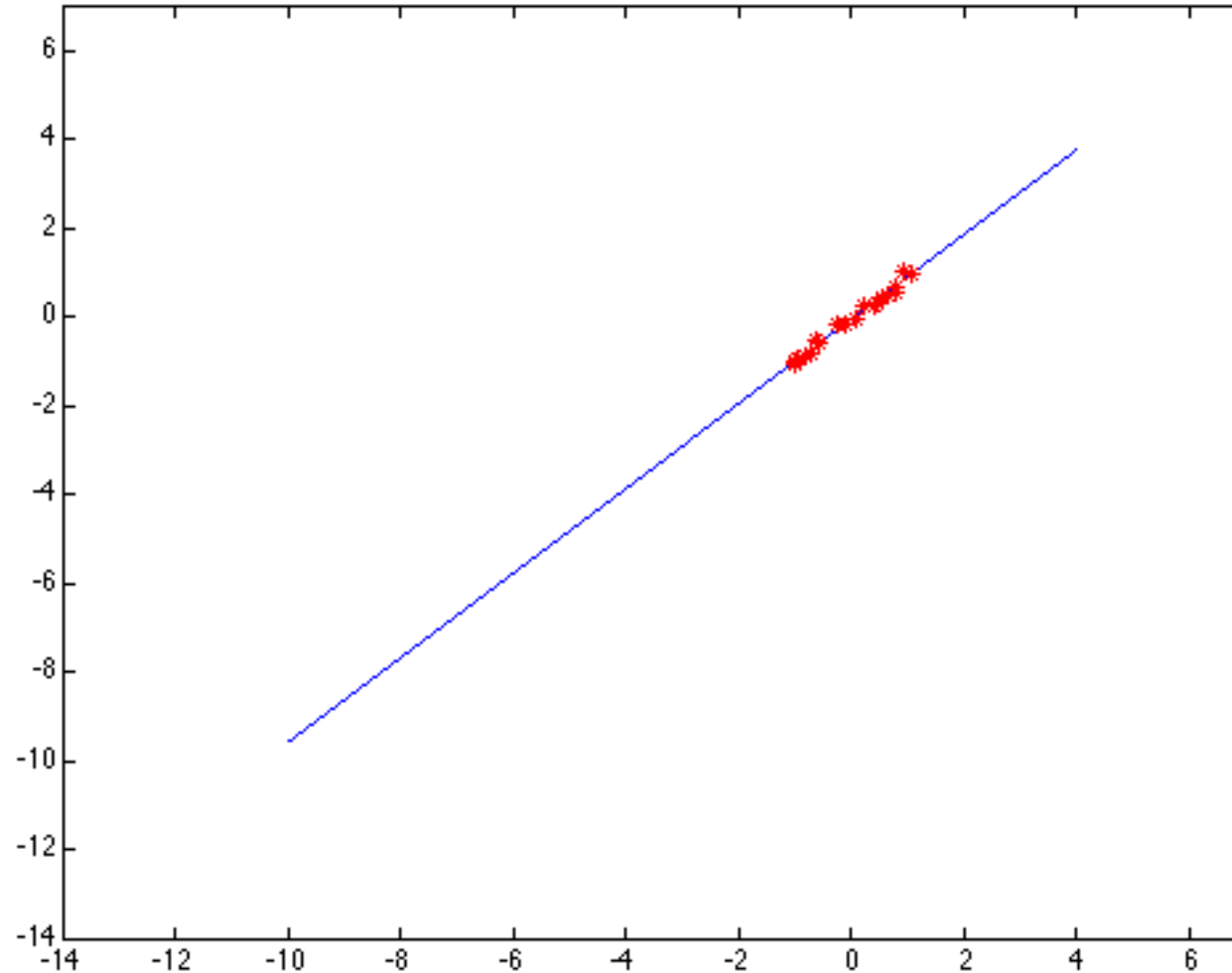
- Clearly specified objective
- Optimization is easy

Bad

- May not be what you want to optimize
- Sensitive to outliers
 - Bad matches, extra points
- Doesn't allow you to get multiple good fits
 - Detecting multiple objects, lines, etc.

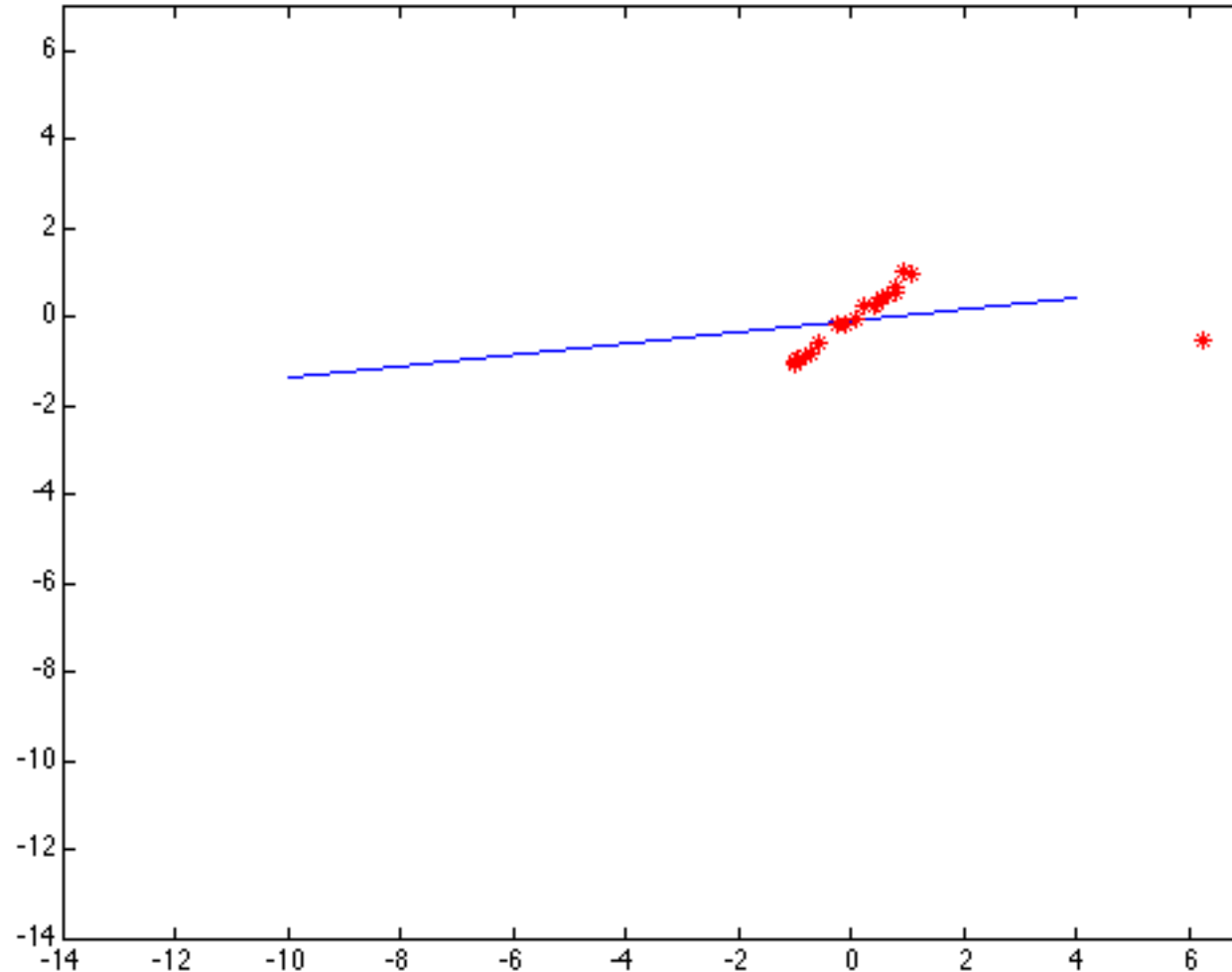
Least squares: Robustness to noise

- Least squares fit to the red points:



Least squares: Robustness to noise

- Least squares fit with an outlier:



Problem: squared error heavily penalizes outliers

Fitting and Alignment: Methods

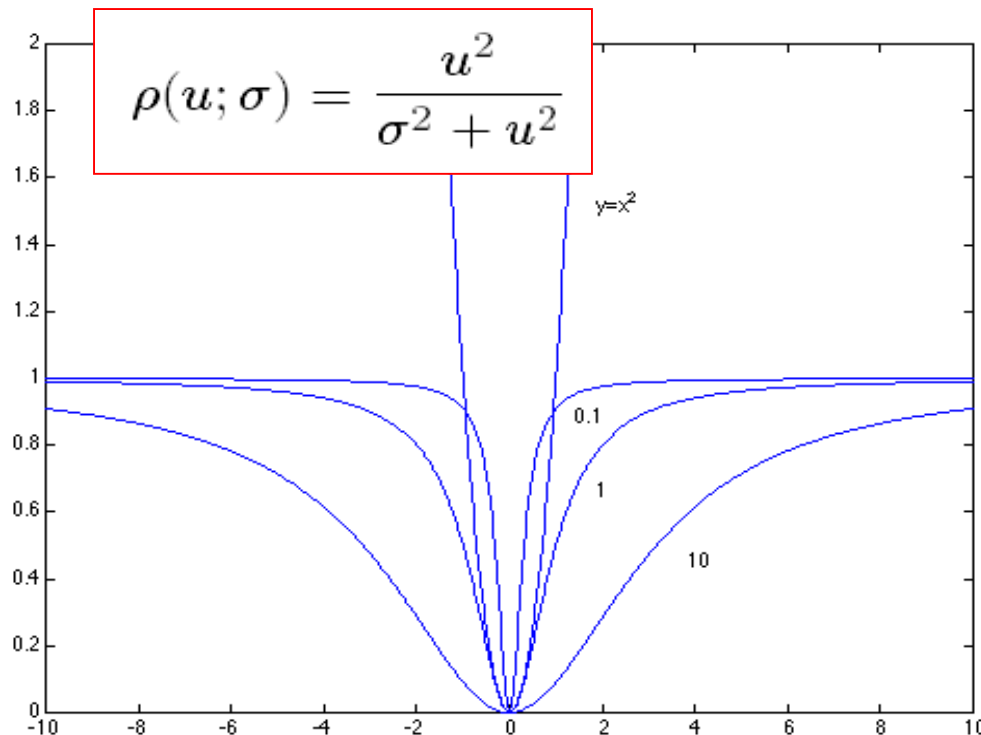
- Global optimization / Search for parameters
 - Least squares fit
 - Robust least squares
 - Other parameter search methods
- Hypothesize and test
 - Generalized Hough transform
 - RANSAC

Robust least squares (to deal with outliers)

General approach:

minimize
$$\sum_i \rho(u_i(x_i, \theta); \sigma) \quad u^2 = \sum_{i=1}^n (y_i - mx_i - b)^2$$

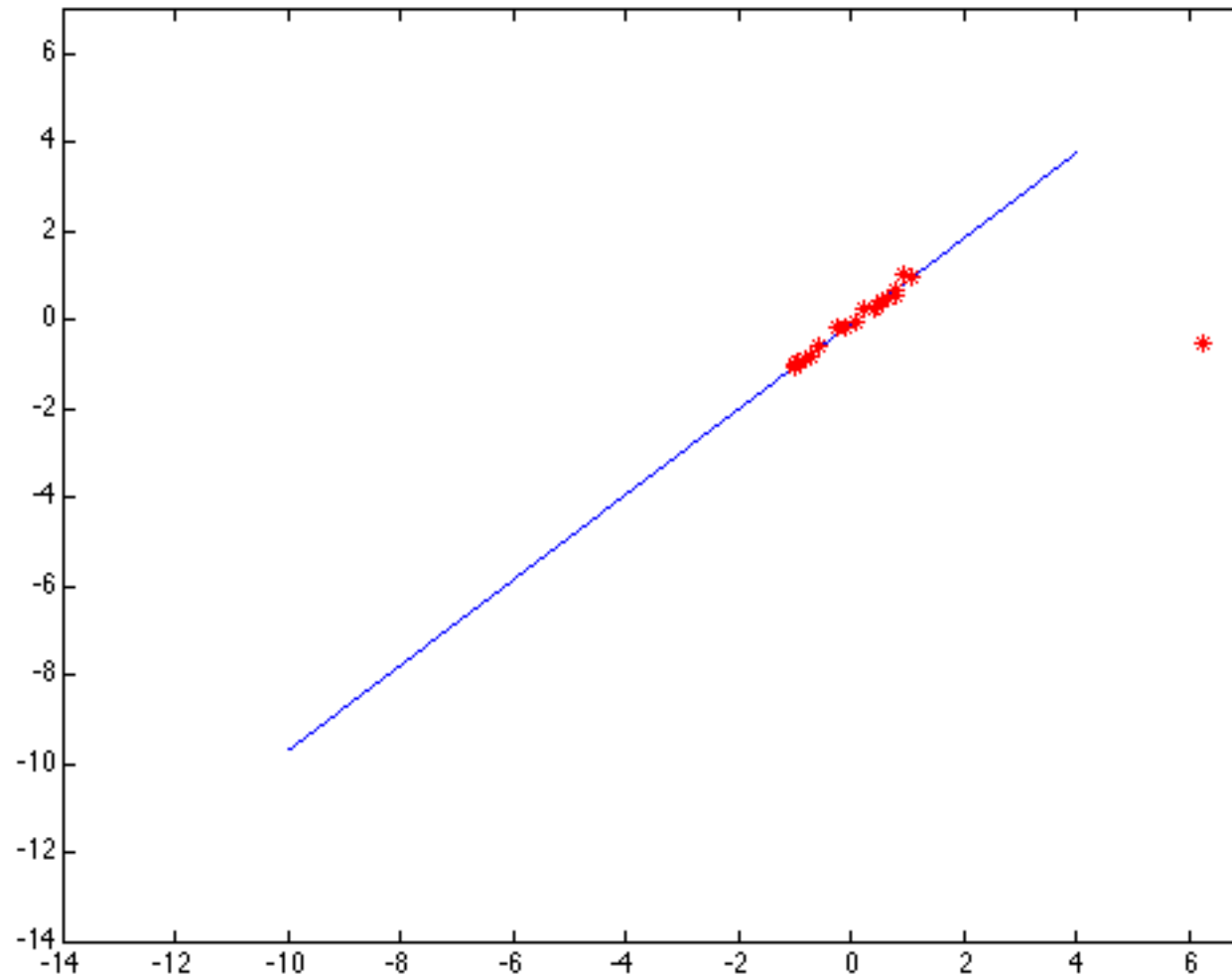
$u_i(x_i, \theta)$ – residual of i^{th} point w.r.t. model parameters ϑ
 ρ – robust function with scale parameter σ



The robust function ρ

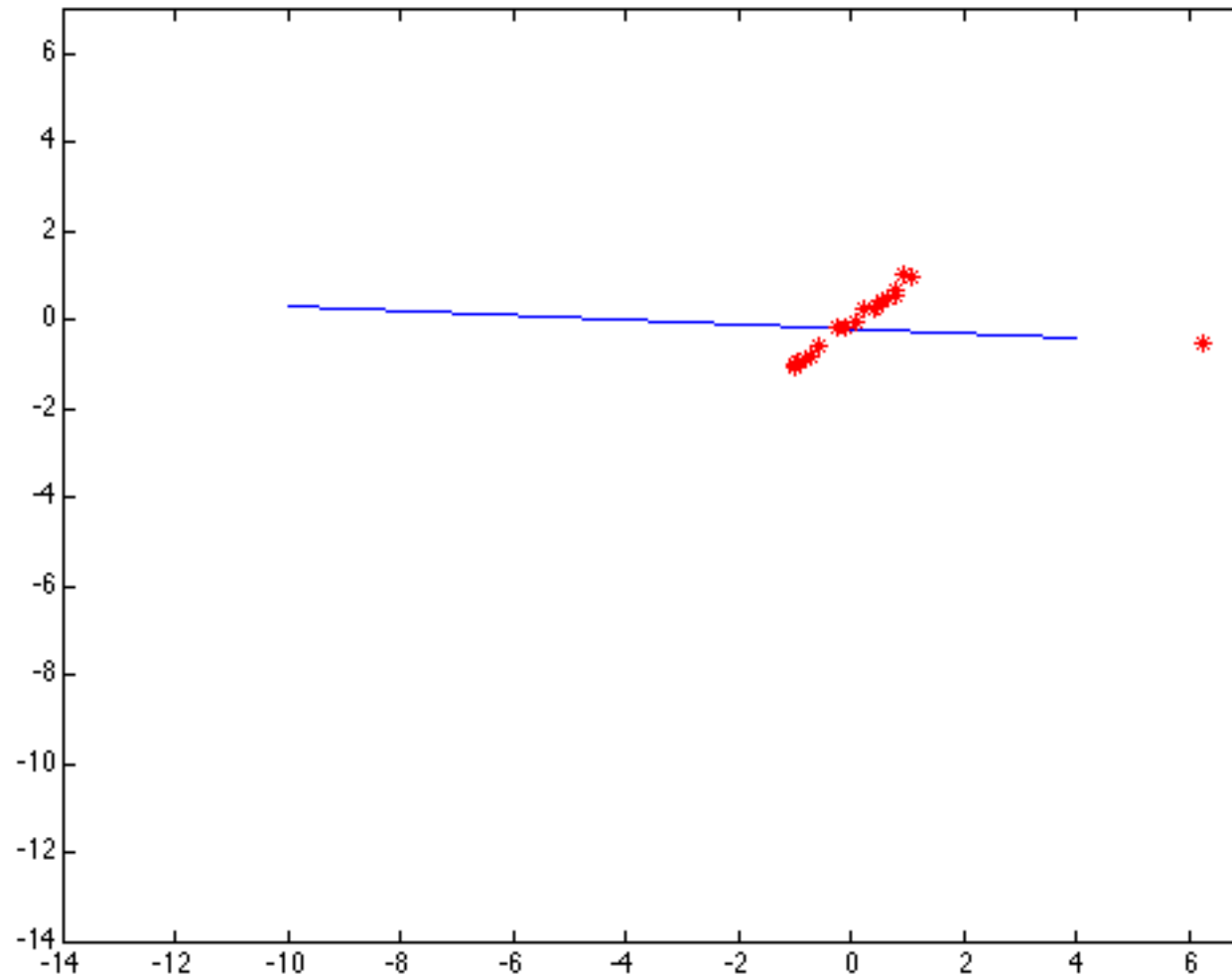
- Favors a configuration with small residuals
- Constant penalty for large residuals

Choosing the scale: Just right



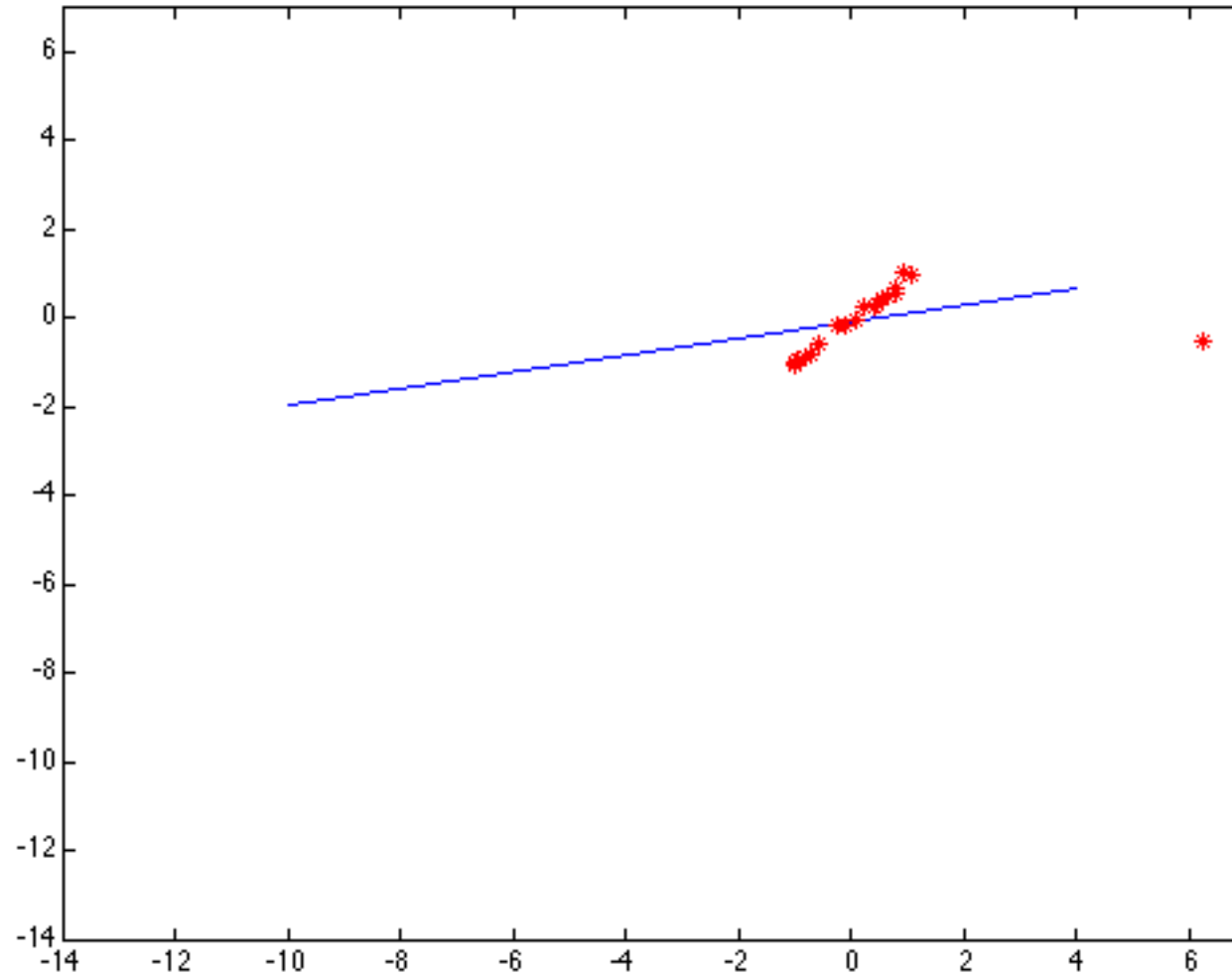
The effect of the outlier is minimized

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

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

Fitting and Alignment: Methods

- Global optimization / Search for parameters
 - Least squares fit
 - Robust least squares
 - Other parameter search methods
- Hypothesize and test
 - Generalized Hough transform
 - RANSAC

Other ways to search for parameters (for when no closed form solution exists)

- Line search (see also “coordinate descent”)
 1. For each parameter, step through values and choose value that gives best fit
 2. Repeat (1) until no parameter changes
- Grid search
 1. Propose several sets of parameters, evenly sampled in the joint set
 2. Choose best (or top few) and sample joint parameters around the current best; repeat
- Gradient descent
 1. Provide initial position (e.g., random)
 2. Locally search for better parameters by following gradient

Fitting and Alignment: Methods

- Global optimization / Search for parameters
 - Least squares fit
 - Robust least squares
 - Other parameter search methods
- Hypothesize and test
 - Generalized Hough transform
 - RANSAC

Fitting and Alignment: Methods

- Global optimization / Search for parameters
 - Least squares fit
 - Robust least squares
 - Other parameter search methods
- Hypothesize and test
 - Generalized Hough transform
 - RANSAC

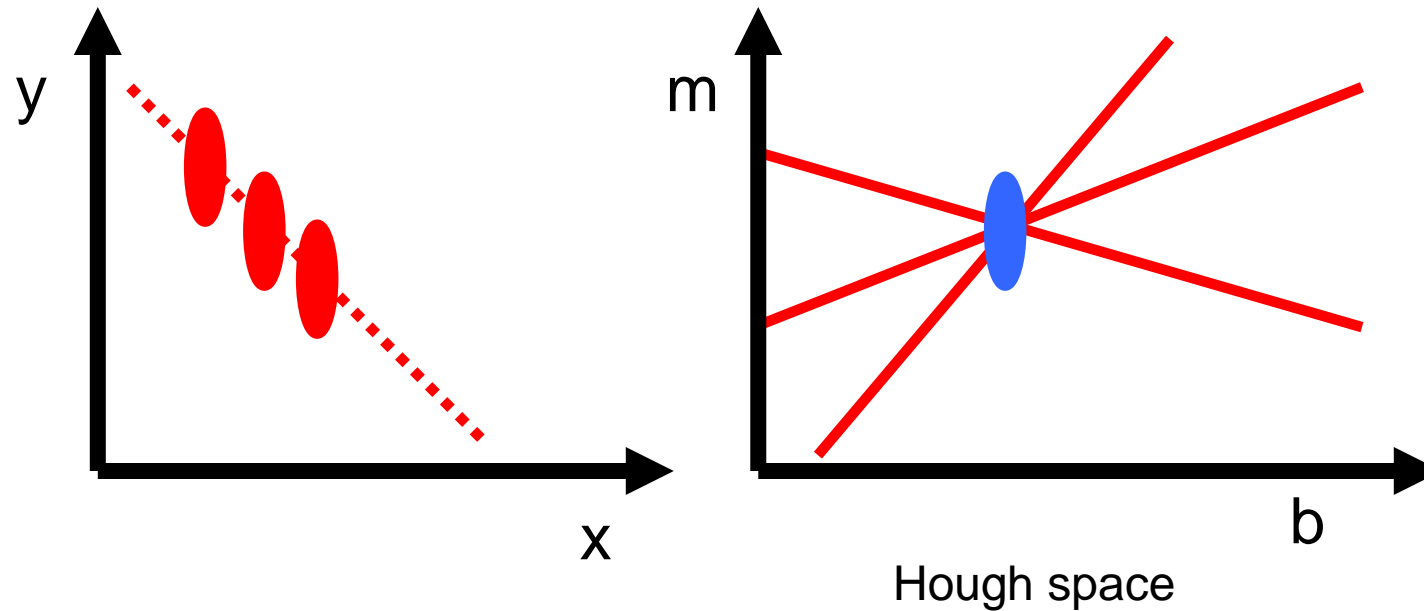
Hough Transform: Outline

1. Create a grid of parameter values
2. Each point votes for a set of parameters, incrementing those values in grid
3. Find maximum or local maxima in grid

Hough transform

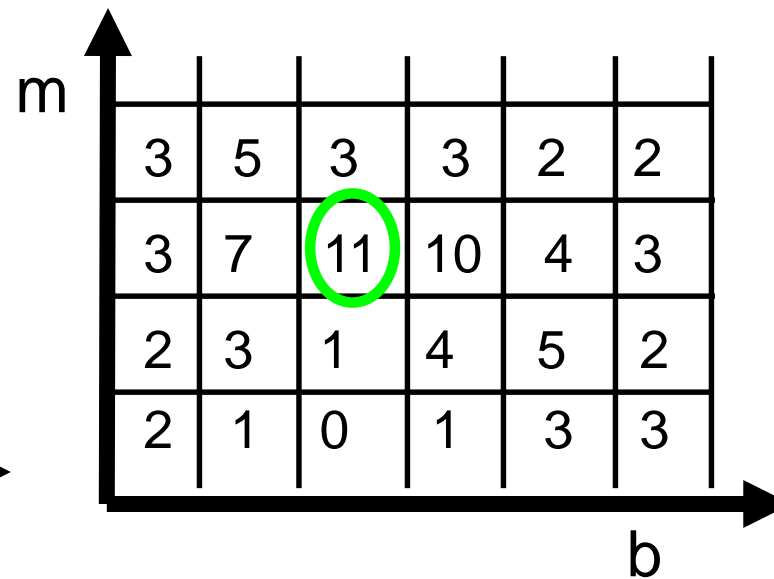
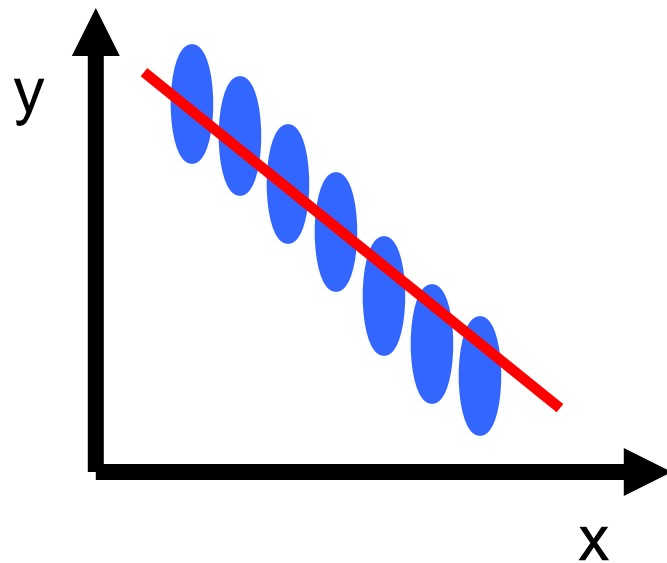
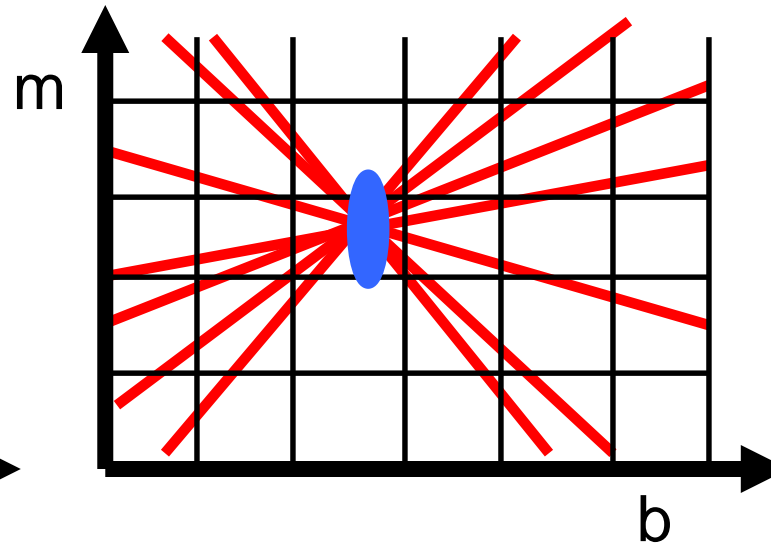
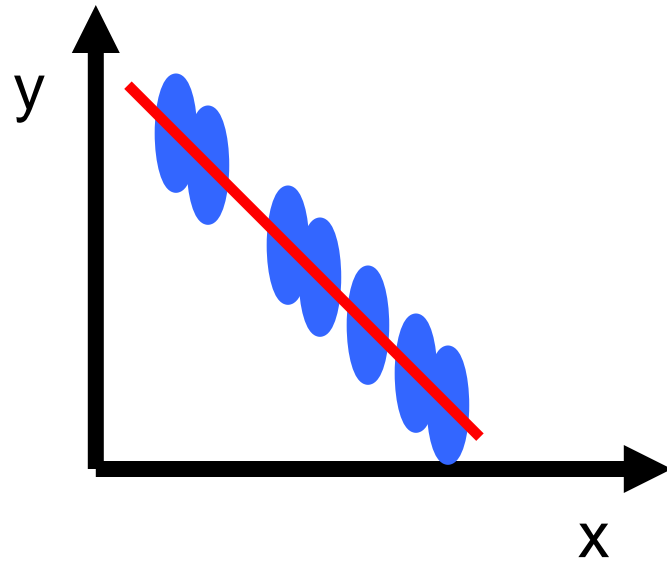
P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures*, Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

Given a set of points, find the curve or line that explains the data points best



$$y = m x + b$$

Hough transform



Hough transform

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

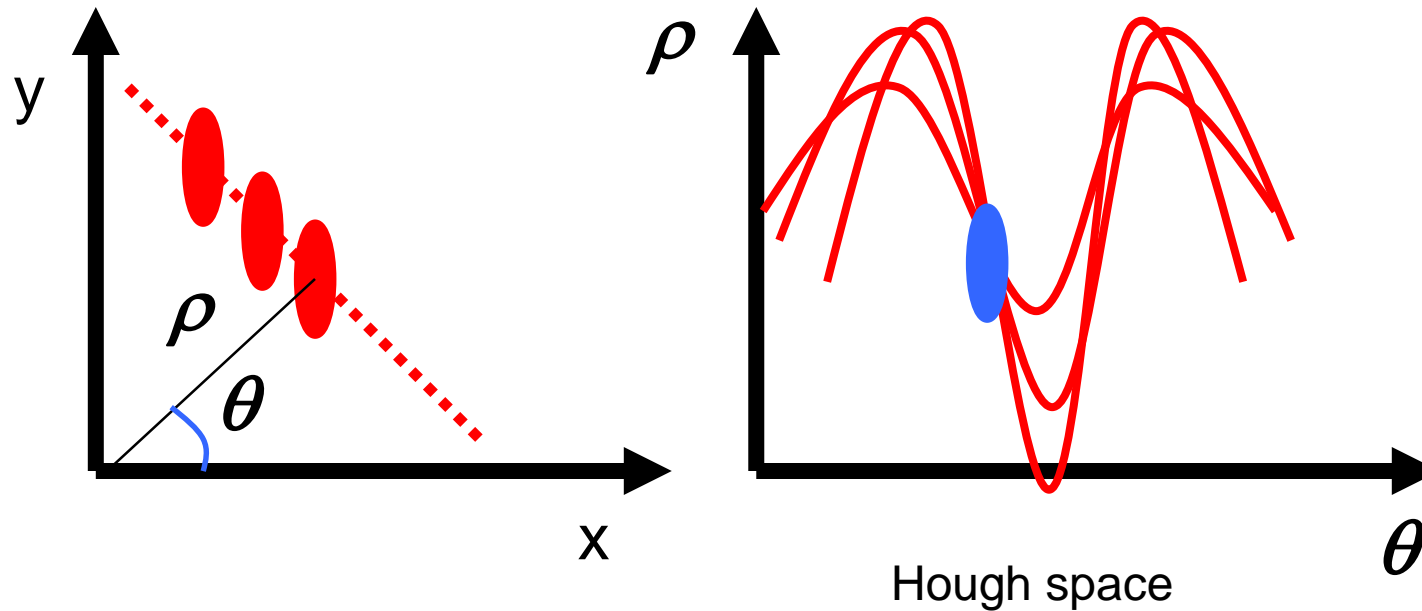
Issue : parameter space $[m,b]$ is unbounded...

Hough transform

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

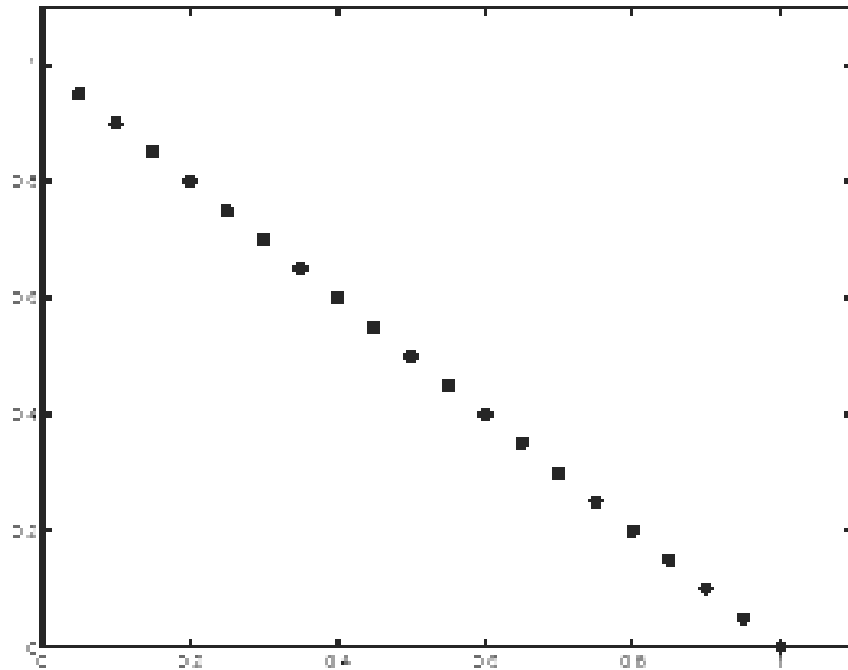
Issue : parameter space $[m,b]$ is unbounded...

Use a polar representation for the parameter space

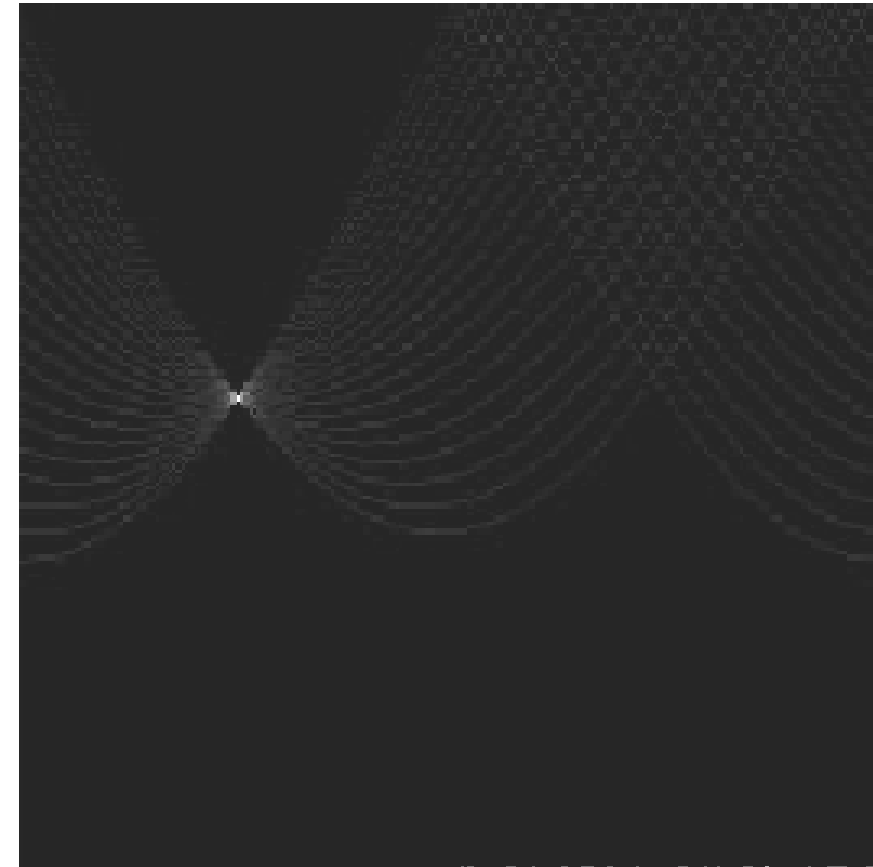


$$x \cos \theta + y \sin \theta = \rho$$

Hough transform - experiments

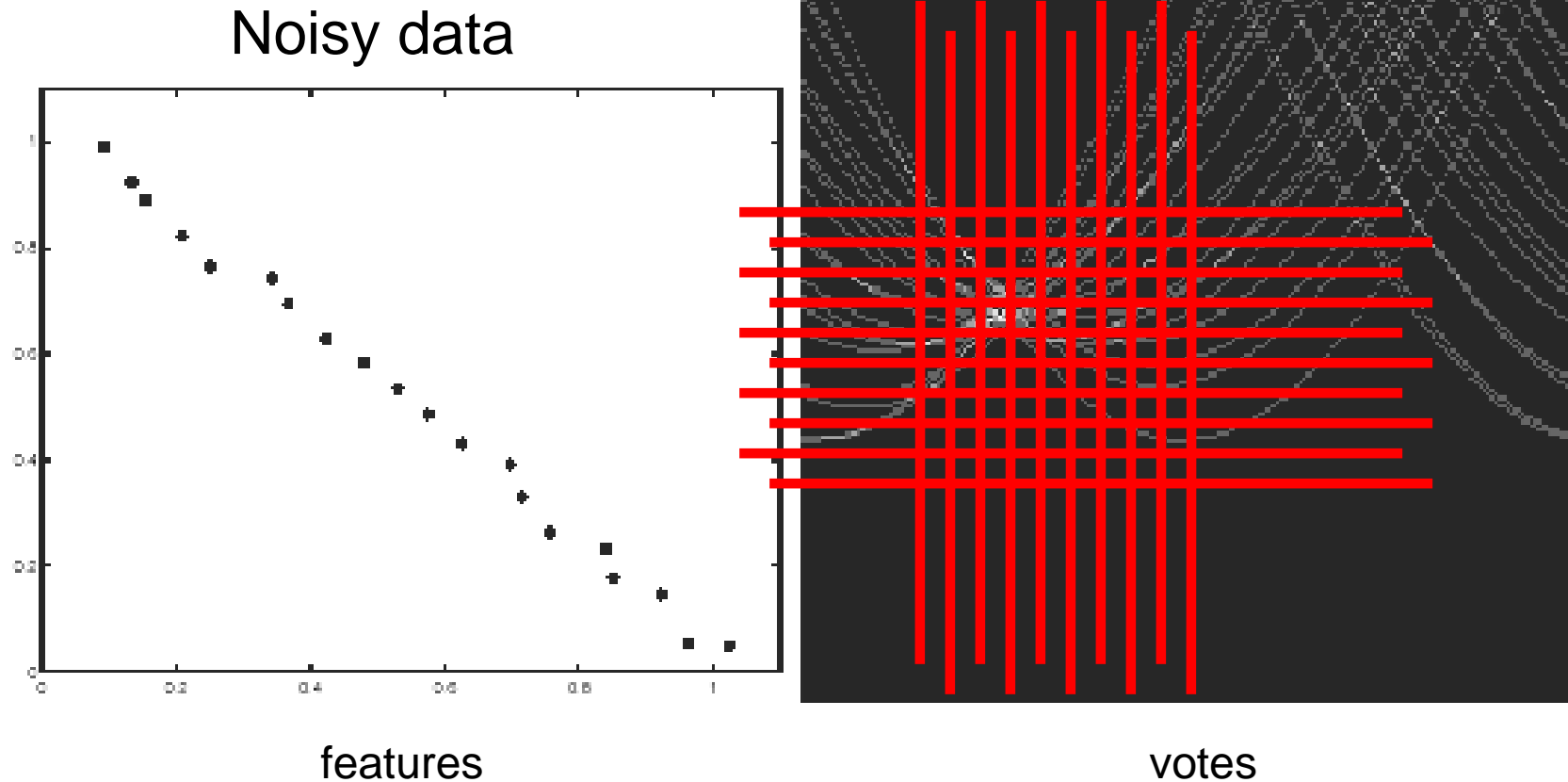


features



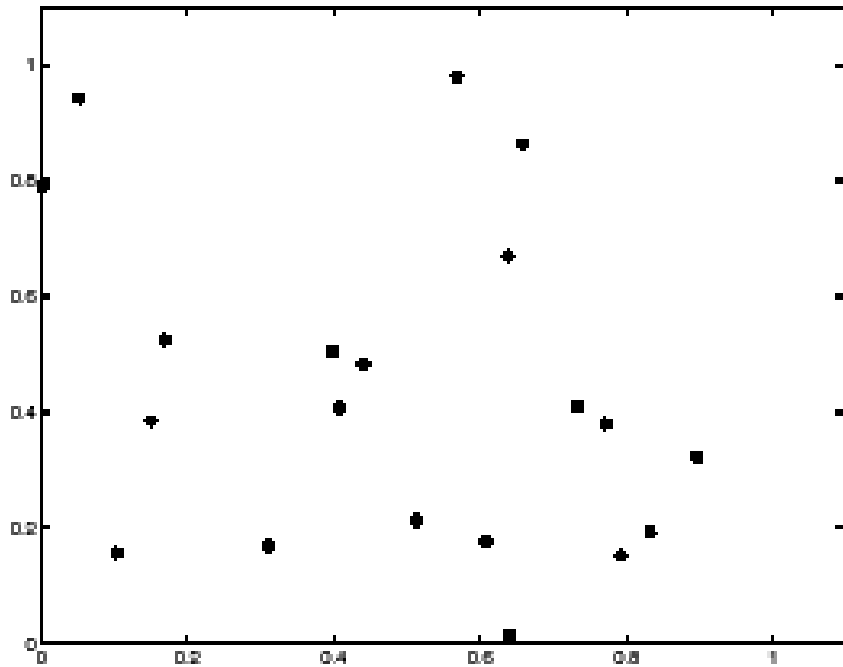
votes

Hough transform - experiments

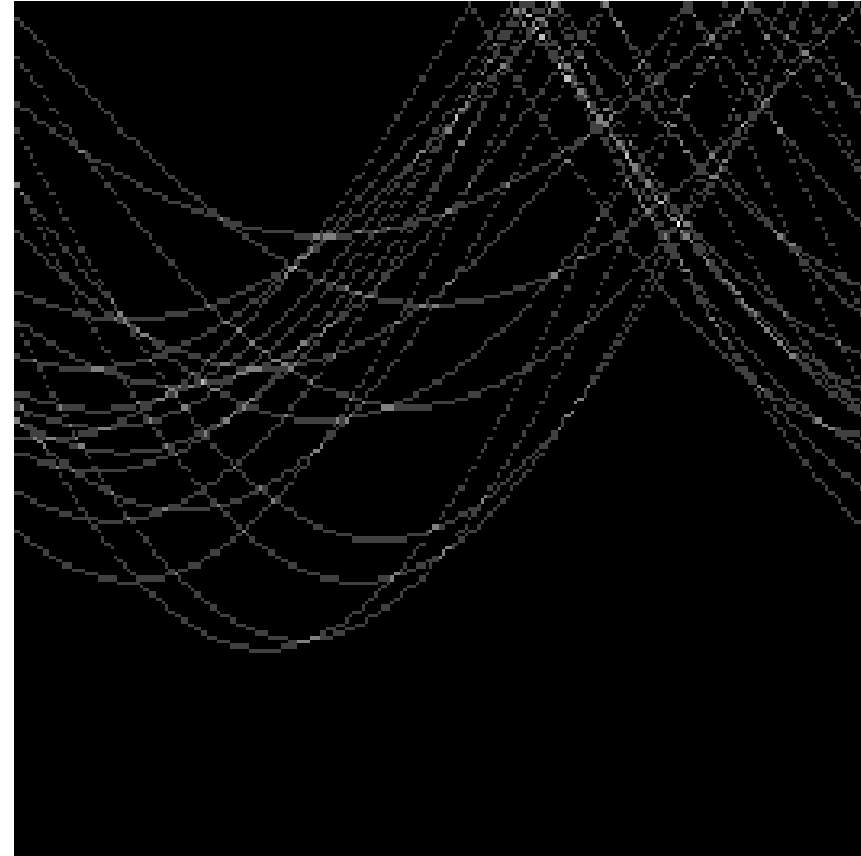


Need to adjust grid size or smooth

Hough transform - experiments



features



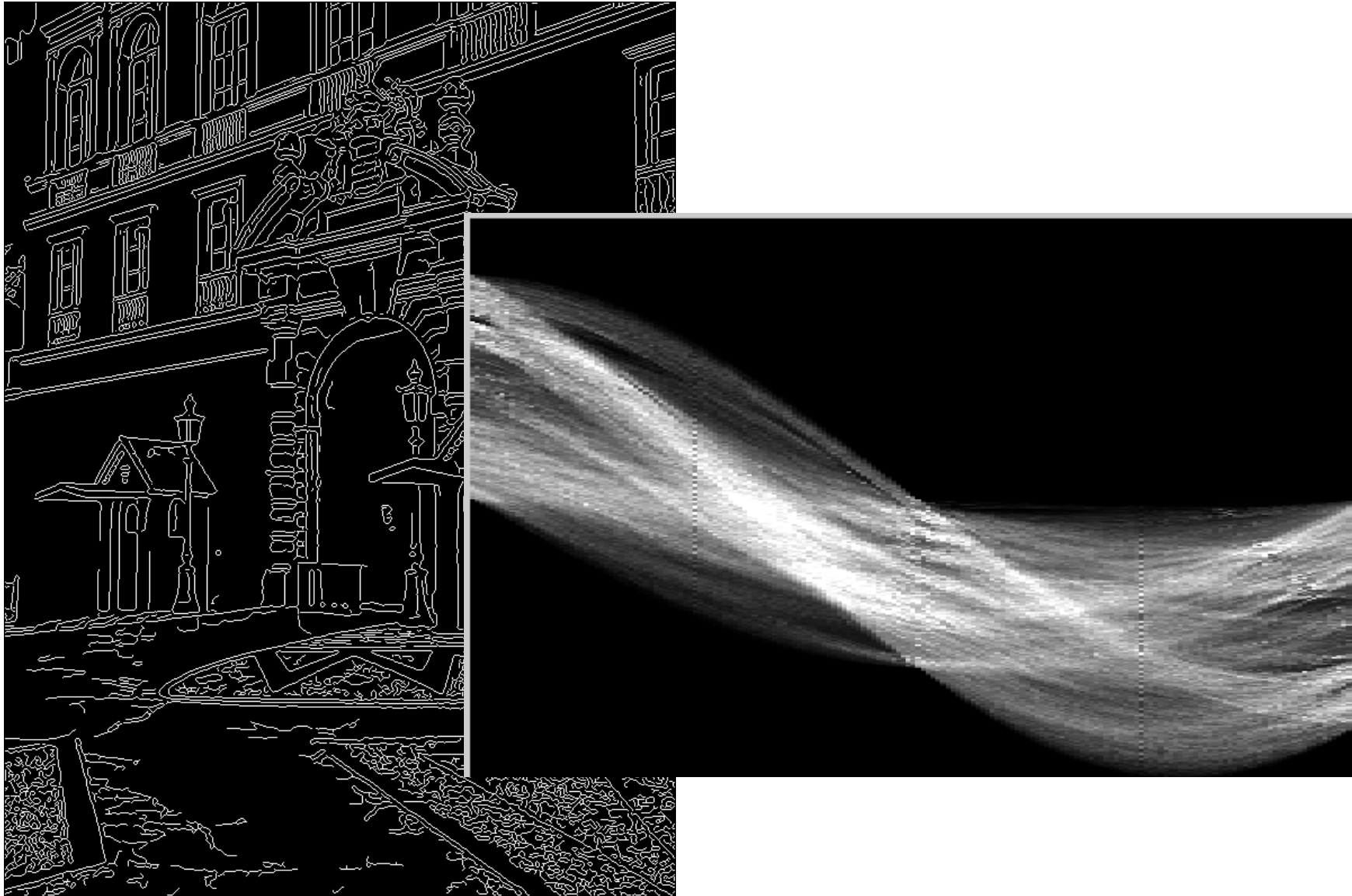
votes

Issue: spurious peaks due to uniform noise

1. Image → Canny Edge Detection

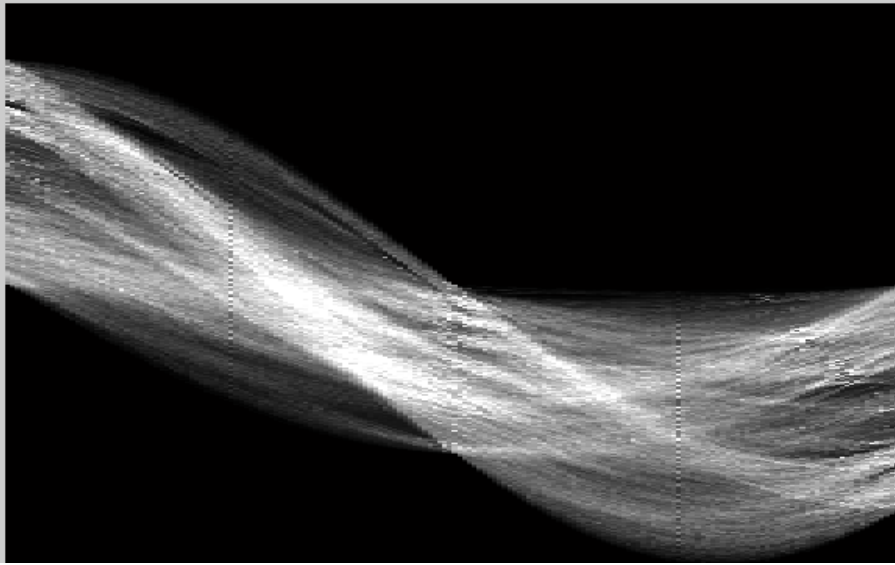


2. Canny \rightarrow Hough votes



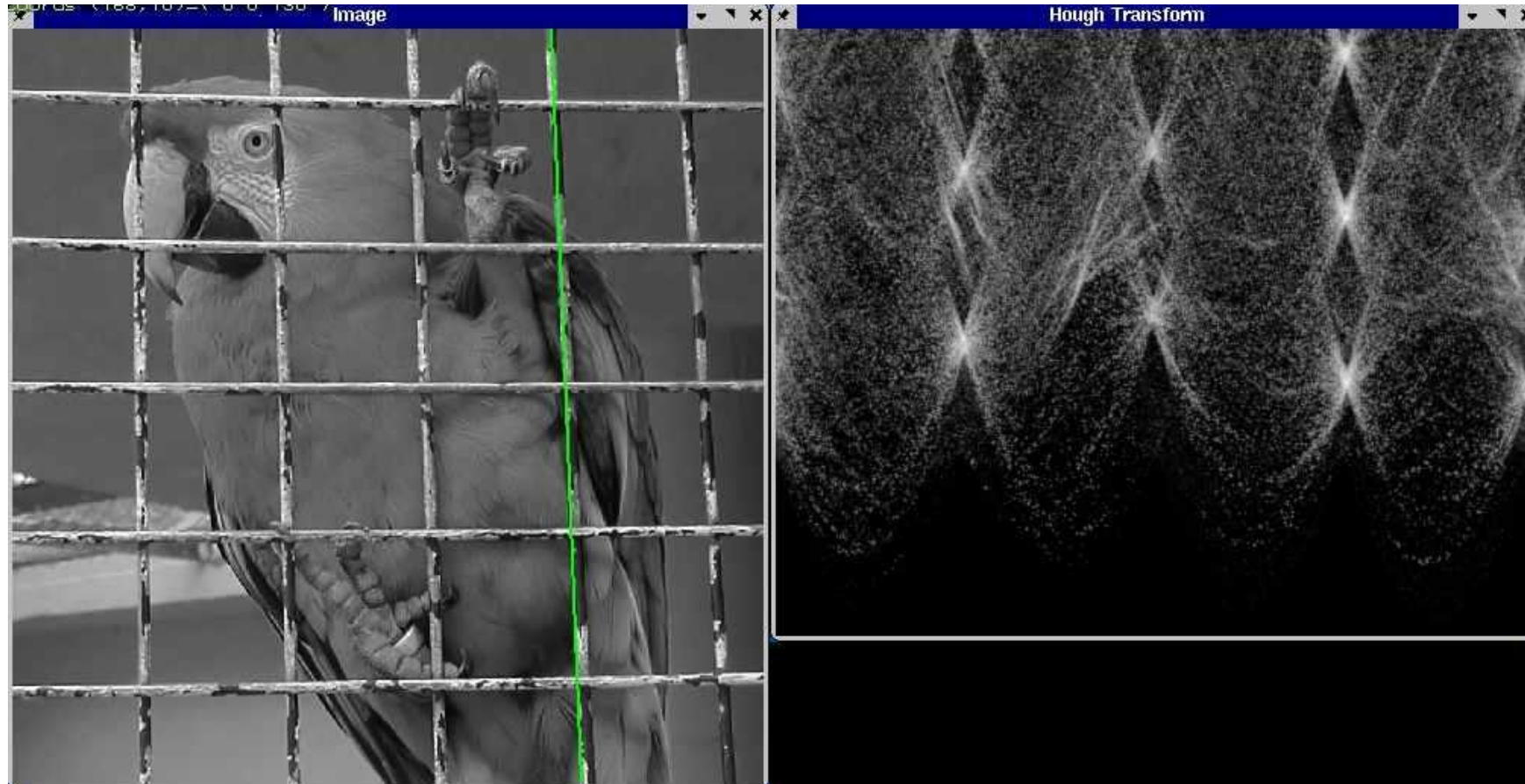
3. Hough votes \rightarrow Edges

Find peaks and post-process



Hough transform example

θ



ρ

Finding lines using Hough transform

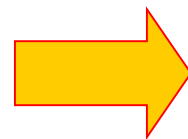
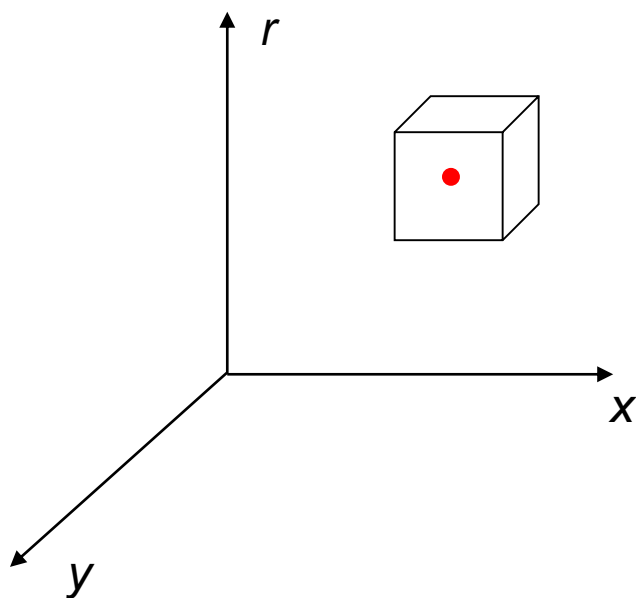
- Using m, b parameterization
- Using r, θ parameterization
 - Using oriented gradients
- Practical considerations
 - Bin size
 - Smoothing
 - Finding multiple lines
 - Finding line segments

Hough Transform

- How would we find circles?
 - Of fixed radius
 - Of unknown radius
 - Of unknown radius but with known edge orientation

Hough transform for circles

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

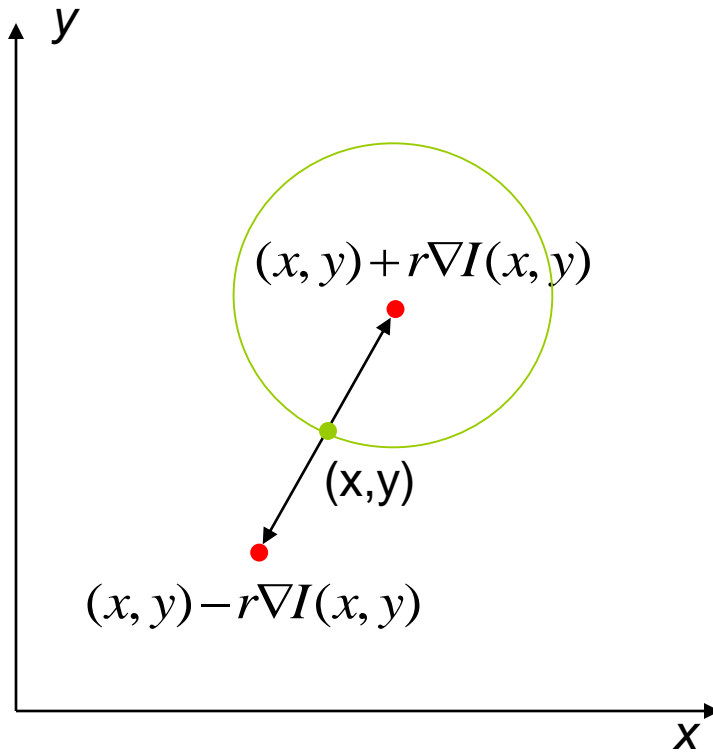


Hough Transform

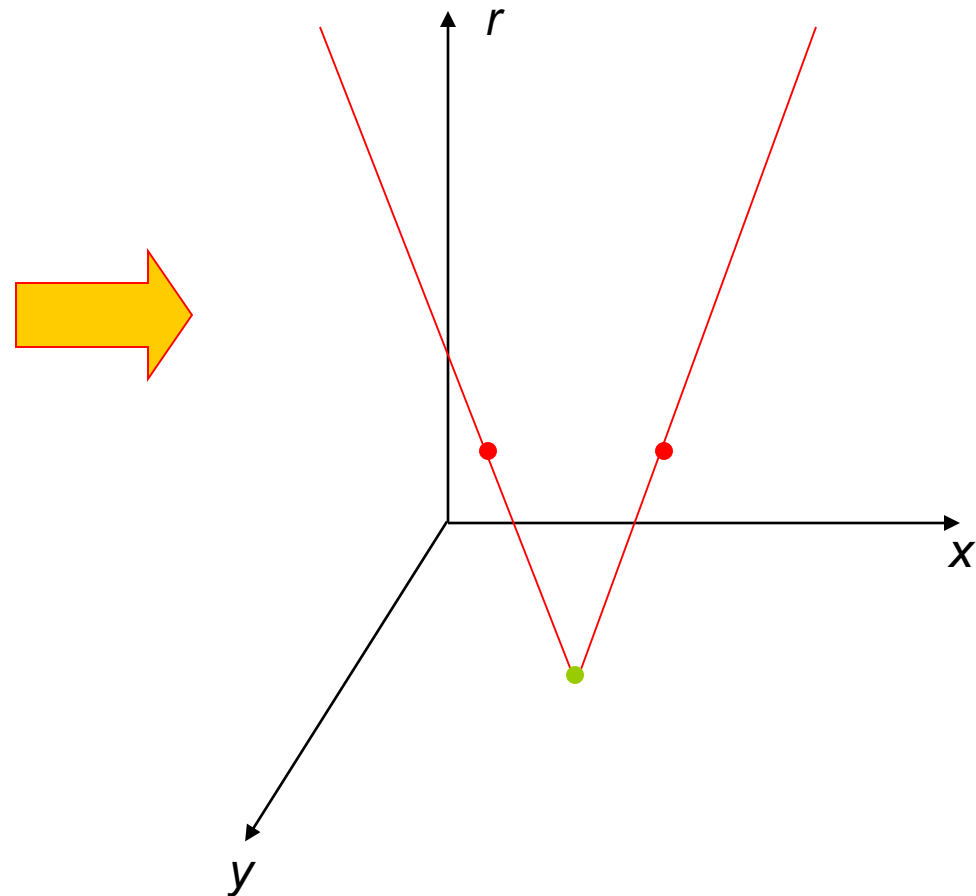
- How would we find circles?
 - Of fixed radius
 - Of unknown radius
 - Of unknown radius but with known edge orientation

Hough transform for circles

image space



Hough parameter space



Hough transform conclusions

Good

- Robust to outliers: each point votes separately
- Fairly efficient (often faster than trying all sets of parameters)
- Provides multiple good fits

Bad

- Some sensitivity to noise
- Bin size trades off between noise tolerance, precision, and speed/memory
 - Can be hard to find sweet spot
- Not suitable for more than a few parameters
 - grid size grows exponentially

Common applications

- Line fitting (also circles, ellipses, etc.)
- Object instance recognition (parameters are affine transform)
- Object category recognition (parameters are position/scale)