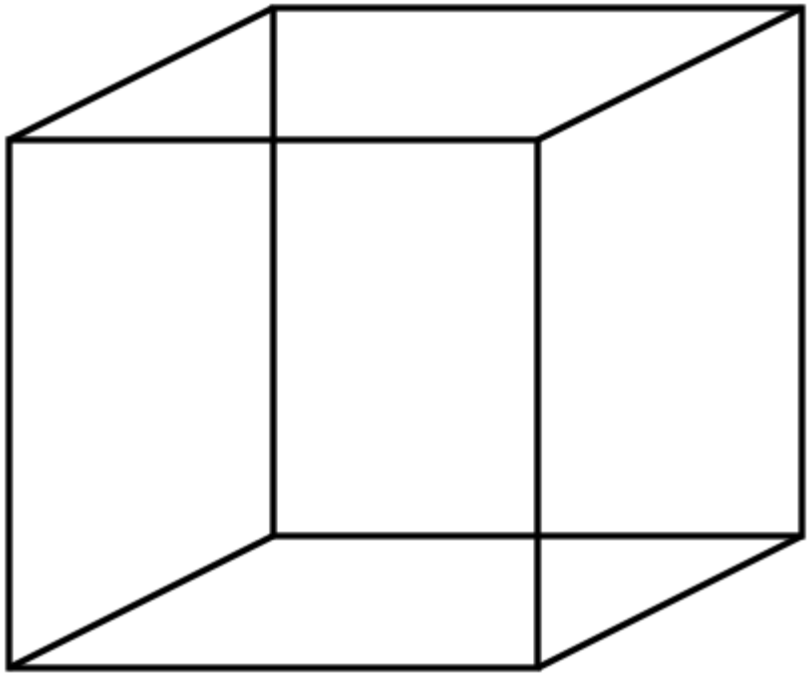
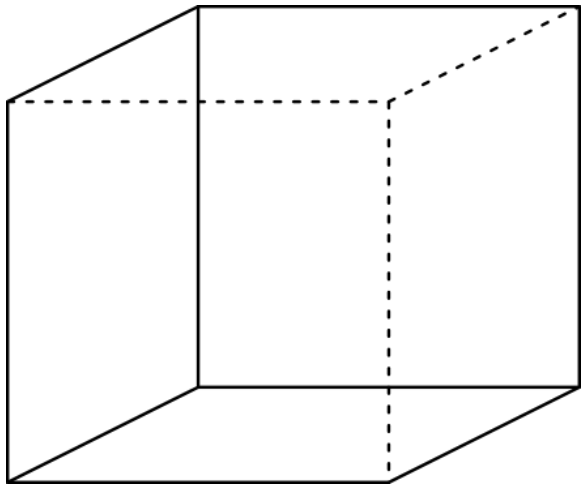
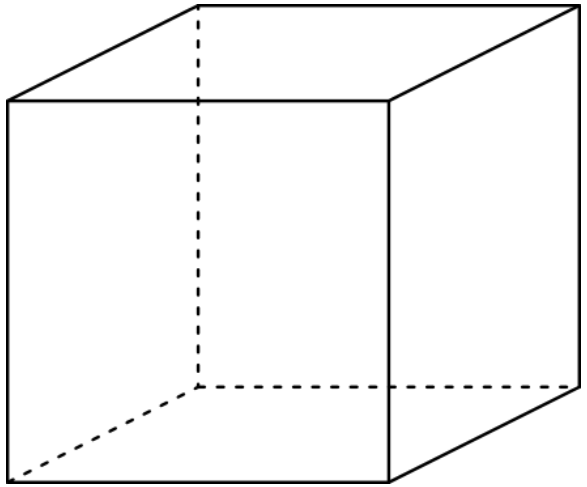
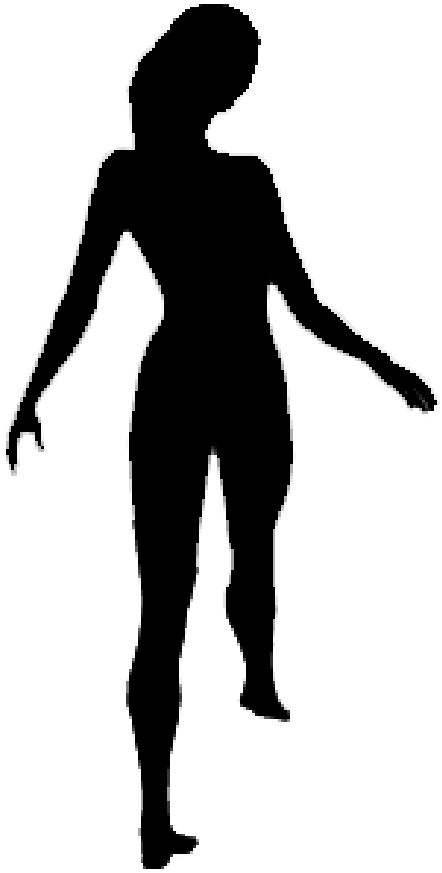
Multi-stable Perception



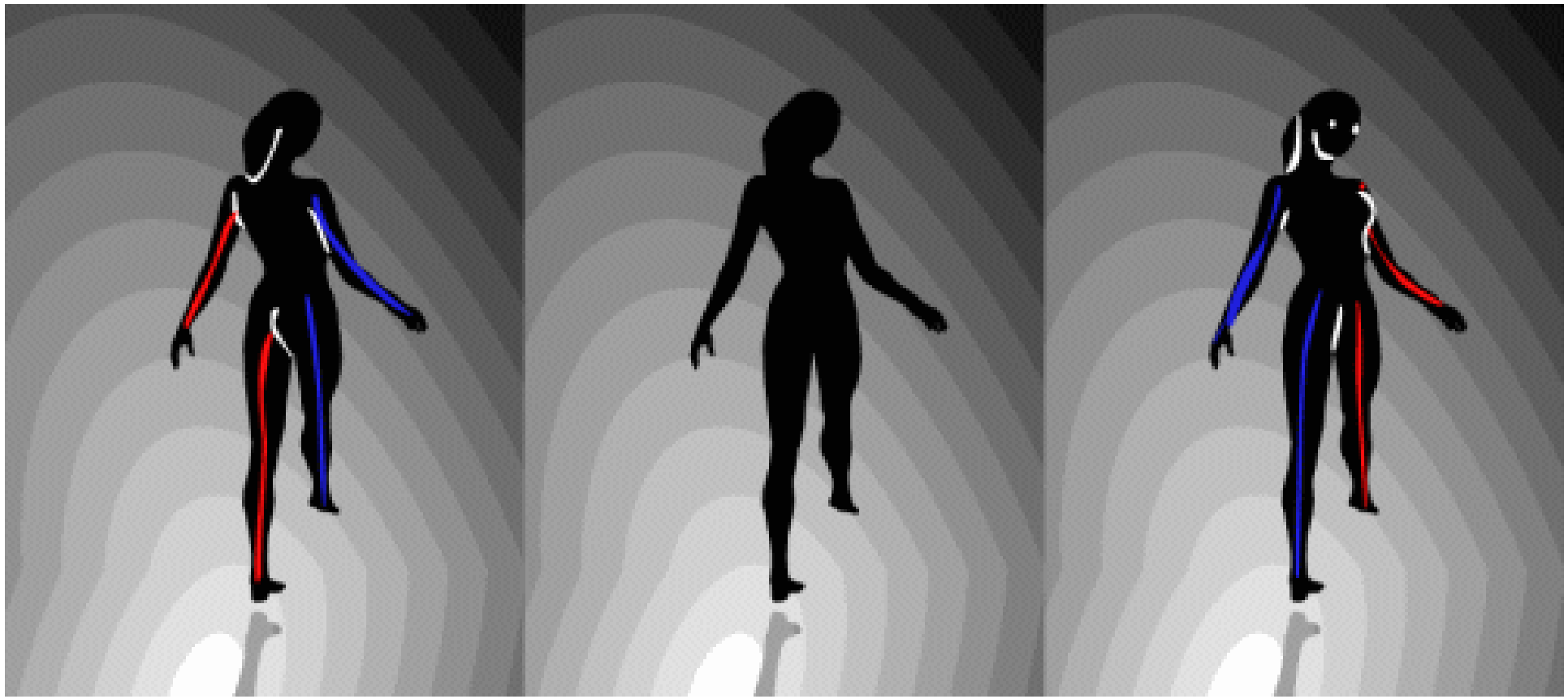
Necker Cube







Spinning dancer illusion, Nobuyuki Kayahara



Feature Matching and Robust Fitting

Read Szeliski 7.4.2 and 2.1

Computer Vision

James Hays

Acknowledgment: Many slides from Derek Hoiem and Grauman&Leibe 2008 AAAI Tutorial

Project 2

Project 2: SIFT Local Feature Matching and Camera Calibration

Brief

- Due: Check Canvas for up to date information
- Project materials including report template: zip file on Canvas
- Hand-in: through Gradescope
- Required files: `<your_gt_username>.zip`, `<your_gt_username>_proj2.pdf`

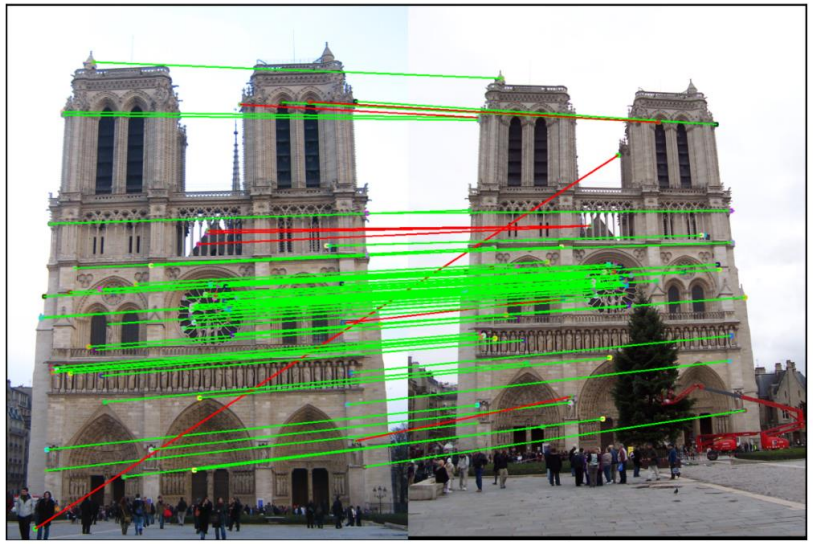


Figure 1: The top 100 most confident local feature matches from a baseline implementation of project 2. In this case, 89 were correct (lines shown in green), and 11 were incorrect (lines shown in red).

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 matching – multiple views of the same physical

for environment installation.

t-2.ipynb

unity checks are passing by running `pytest tests`

submission once you've finished the project using username>

ps of a local feature matching algorithm (detecting matching feature vectors). We'll implement two ganized as follows:

(see Szeliski 7.1.1)

patch feature in `part2_patch_descriptor.py` (see

e Szeliski 7.1.3)

art4_sift_descriptor.py (see Szeliski 7.1.2)

is_corner.py)

ed in the lecture materials and Szeliski 7.1.1.

tion 7.8 of book, p. 424)

$$w * \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \quad (1)$$

liscrete convolutions with the weighting kernel w

ation matrix A as:

$$\text{trace}(A)^2 \quad (2)$$

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 symmetric, so only three entries are needed.);

(Equation 2) discussed above.;

rt them as detected feature point locations.;

ll out the following methods in `part1_harris_corner`

ents using the Sobel filter.

er responses over the entire image (the previously

ppression using max-pooling. You can use PyTorch

nts from the entire image (the previously imple-

rt1_harris_corner.py:

Gaussian kernel (this is essentially the same as your

s of the input image. This makes use of your

n using just NumPy. This manual implementation ext step.

border that we can't create a useful SIFT window

u do not need to worry about scale invariance or orner detector. The original paper by Chris Harris e found here.

S (part2_patch_descriptor.py)

ll implement a bare-bones feature descriptor in mage intensity patches as your local feature. See lized_patch_descriptors()

center of a square window, as shown in Figure

Notre Dame is around 40 – 45% and Mt Rushmore

eature_matching.py)

"nearest neighbor distance ratio test") method of rials and Szeliski 7.1.3 (page 444). See equation tio test the easiest should have a greater tendency

- `projection()`: Projects homogeneous world coordinates $[X, Y, Z, 1]$ to non-homogeneous image coordinates (u, v). Given projection matrix M, the equations that accomplish this are (4) and (5).
- `n_matrix()`: Solves for the camera projection matrix using a system of equations nding 2D and 3D points.

`enter()`: Computes the camera center location in world coordinates.

lamental matrix

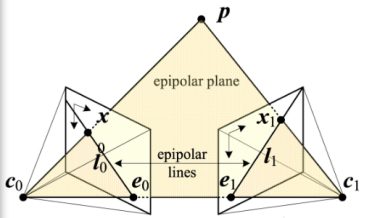


Figure 2: Two-camera setup. Reference: Szeliski, p. 682.

project is estimating the mapping of points in one image to lines in another by matrix. This will require you to use similar methods to those in part 4. We will ding point locations listed in `pts2d-pic_a.txt` and `pts2d-pic_b.txt`. Recall that mental matrix is:

$$\begin{pmatrix} u' & v' & 1 \end{pmatrix} \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = 0 \quad (9)$$

ge A, and a point $(u', v', 1)$ in image B. See Appendix A for the full derivation. atrix is sometimes defined as the transpose of the above matrix with the left and ed. Both are valid fundamental matrices, but the visualization functions in the se the above form.

is matrix equations is:

$$\begin{pmatrix} u' & v' & 1 \end{pmatrix} \begin{pmatrix} f_{11}u + f_{12}v + f_{13} \\ f_{21}u + f_{22}v + f_{23} \\ f_{31}u + f_{32}v + f_{33} \end{pmatrix} = 0 \quad (10)$$

$$f_{12}vu' + f_{13}u' + f_{21}uv' + f_{22}vv' + f_{23}v' + f_{31}u + f_{32}v + f_{33}) = 0 \quad (11)$$

sion equations? Given corresponding points you get one equation per point pair. can solve this (why 8?). Similar to part 4, there's an issue here where the matrix e and the degenerate zero solution solves these equations. So you need to solve u used in part 4 of first fixing the scale and then solving the regression.

e of F is full rank; however, a proper fundamental matrix is a rank 2. As such we rder to do this, we can decompose F using singular value decomposition into the

- 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?

port using the template slides provided s, as this will affect the grading process u will describe your algorithm and any you will show and discuss the results of should include in your report. A good from the experiments. You must convert ach PDF page to the relevant question

n in the template deck to describe your lit for your extra credit implementations

tarter code includes file handling, visual- lder versions of the three functions listed

in the starter code as well. `evaluate_` based on hand-provided matches . The ther image pairs (Mount Rushmore and he appropriate lines in `project-2.ipynb`.

our performance according to `evaluate_` *overfit* to the initial Notre Dame image re and in the starter code will give you

`fliplr()`, `np.flipud()`, `np.histogram()`, ape(), `np.sort()`.

(), `torch.median()`, `torch.nn.functional` rameter, `torch.stack()`.

might find `torch.meshgrid`, `torch.norm`,

orch.nn.Conv2d or `torch.nn.functional` er libraries (e.g., `cv.filter2D()`, `scipy`.

Part 6: Homography with RANSAC

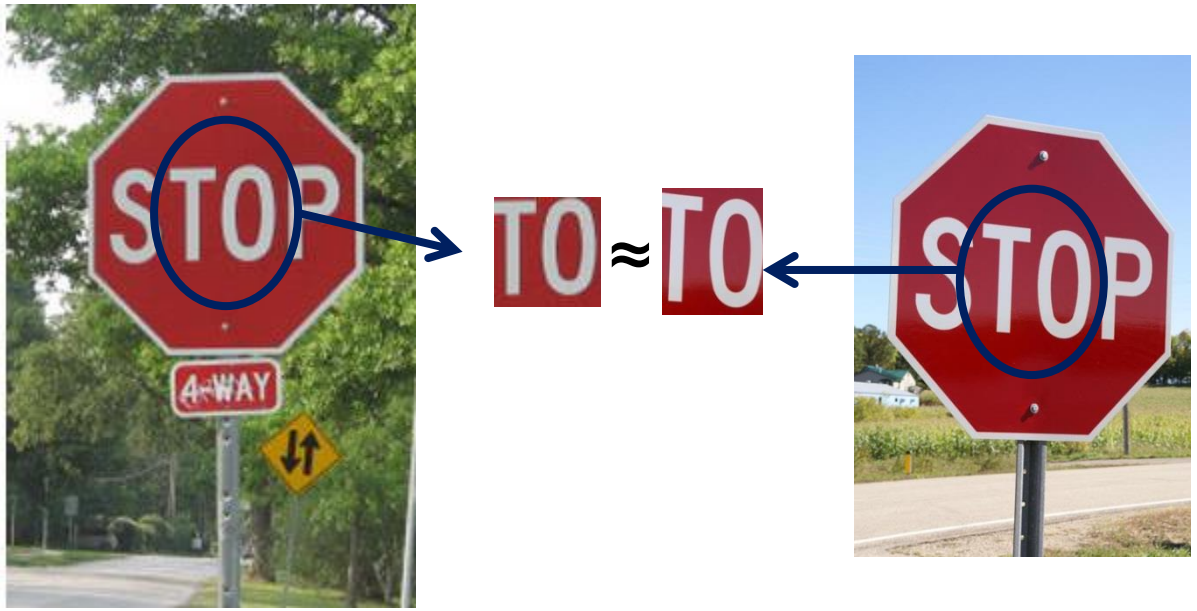
For two photographs of a scene, it's unlikely that you'd have perfect point correspondence with which to solve for the homography matrix. So, next you are going to compute the homography with point correspondences computed using SIFT. As discussed in class, a direct least-squares solution alone is not appropriate in this scenario due to the presence of multiple outliers. In order to estimate the homography from this noisy data, you'll need to use RANSAC. This technique is especially powerful for aligning images of planar surfaces, like building facades or the ground plane.

You'll use these putative point correspondences and RANSAC to find the "best" homography matrix. You will iteratively choose a minimal set of point correspondences (4 for a homography), solve for the homography matrix, and then count the number of inliers. Inliers in this context will be point correspondences that "agree" with the estimated homography. In order to count how many inliers a homography has, you'll need a distance metric. For a homography, this is typically the reprojection error: the geometric distance between a point in the second image (x') and the location predicted by mapping its corresponding point from the first image (Hx). You'll need to pick a threshold between inliers and outliers; your results are very sensitive to this threshold, so explore a range of values. You don't want to be too permissive about what you consider an inlier, nor do you want to be too conservative. Return the homography with the most inliers.

Recall from lecture the expected number of iterations of RANSAC to find the "right" solution in the presence of outliers. For example, if half of your input correspondences are wrong, then you have a $(0.5)^4 = 6.25\%$

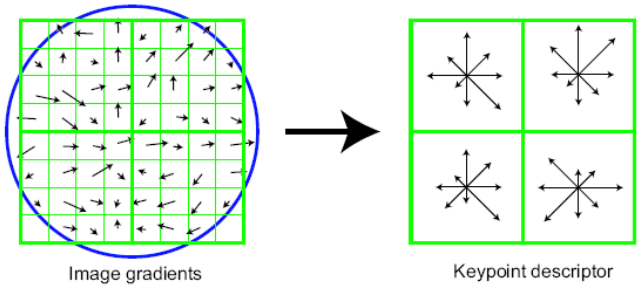
This section: correspondence and alignment

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



Review: Local Descriptors

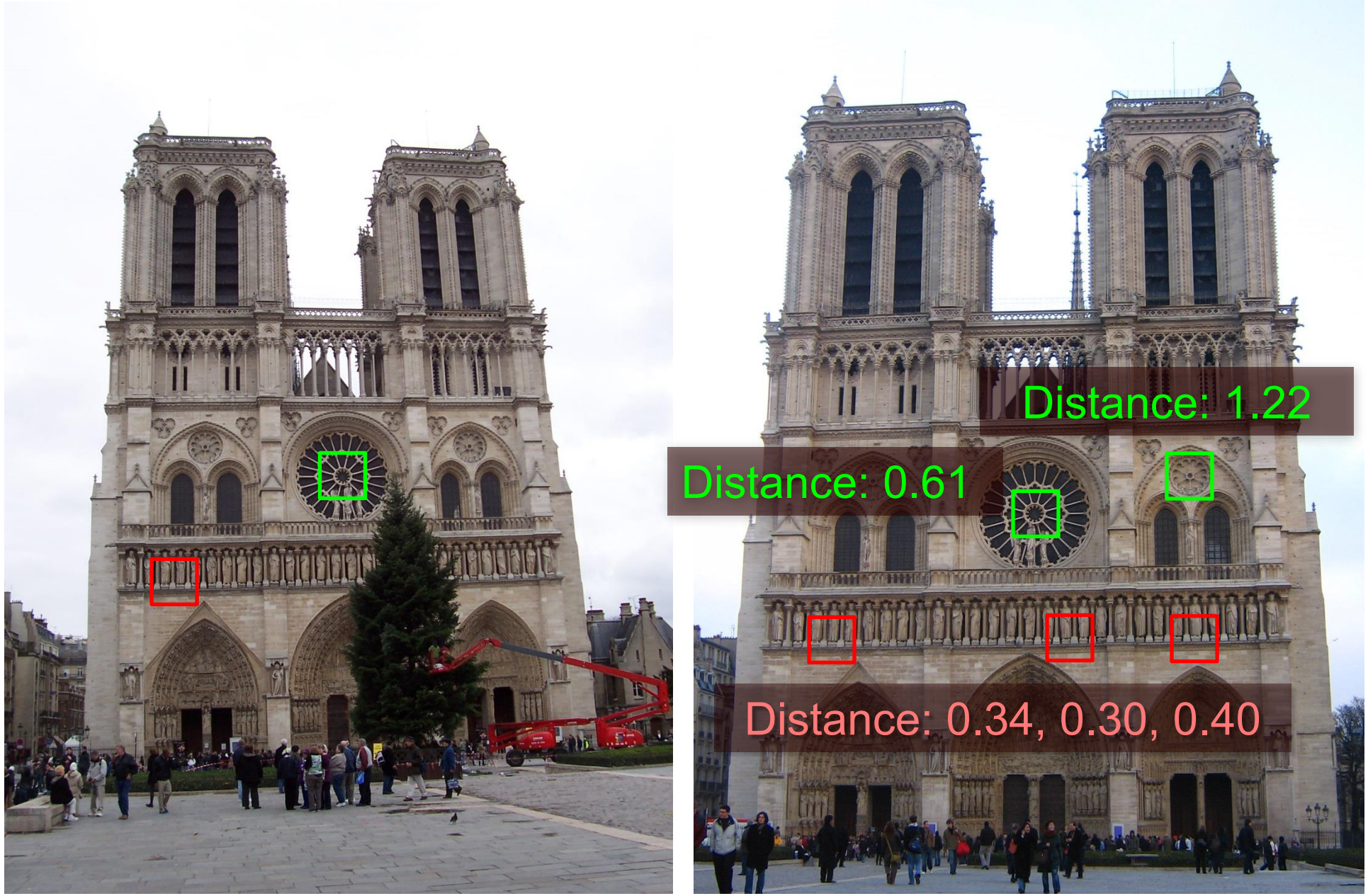
- Most features can be thought of as templates, histograms (counts), or combinations
- The ideal descriptor should be
  - Robust and Distinctive
  - Compact and Efficient



- Most available descriptors focus on edge/gradient information
  - Capture texture information
  - Color rarely used

Matching

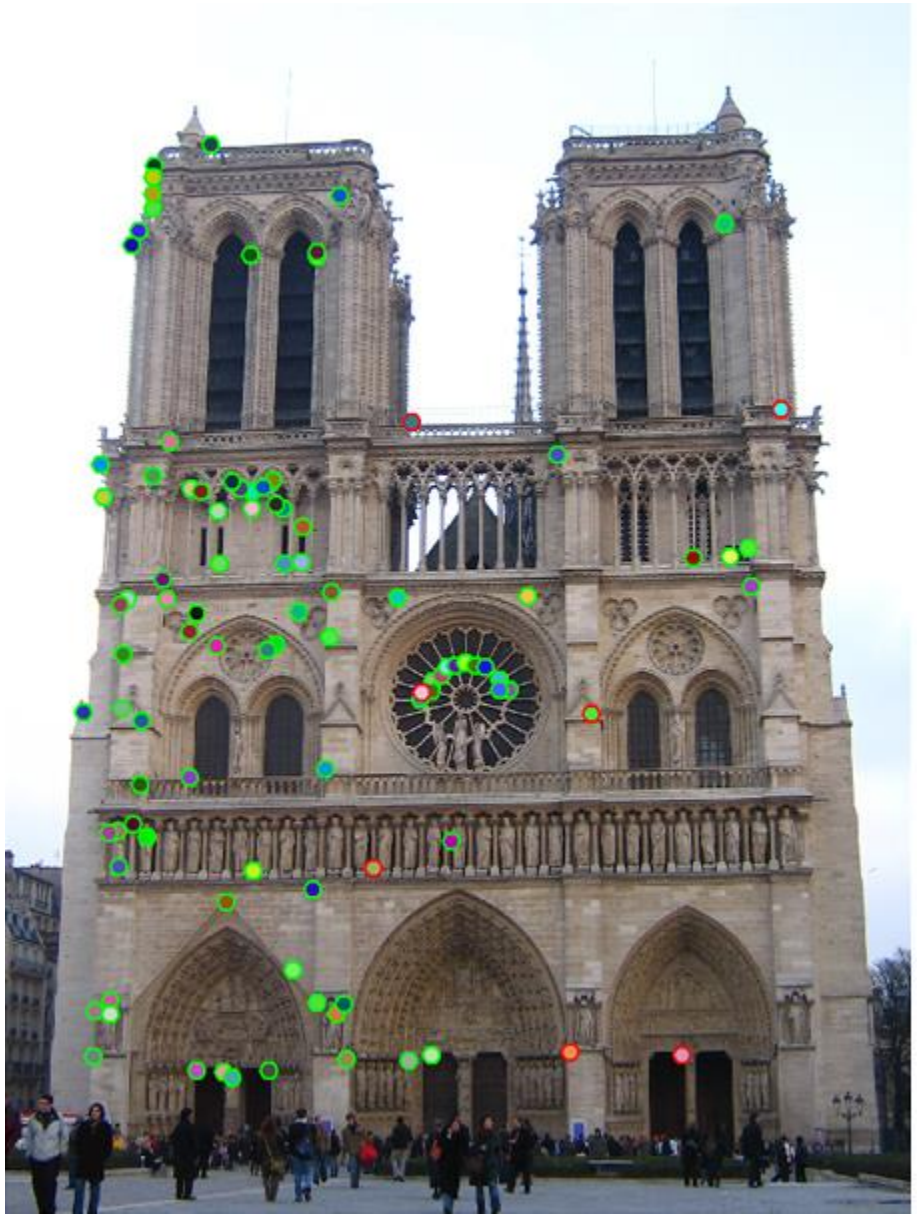
- Simplest approach: Pick the nearest neighbor. Threshold on absolute distance
- Problem: Lots of self similarity in many photos

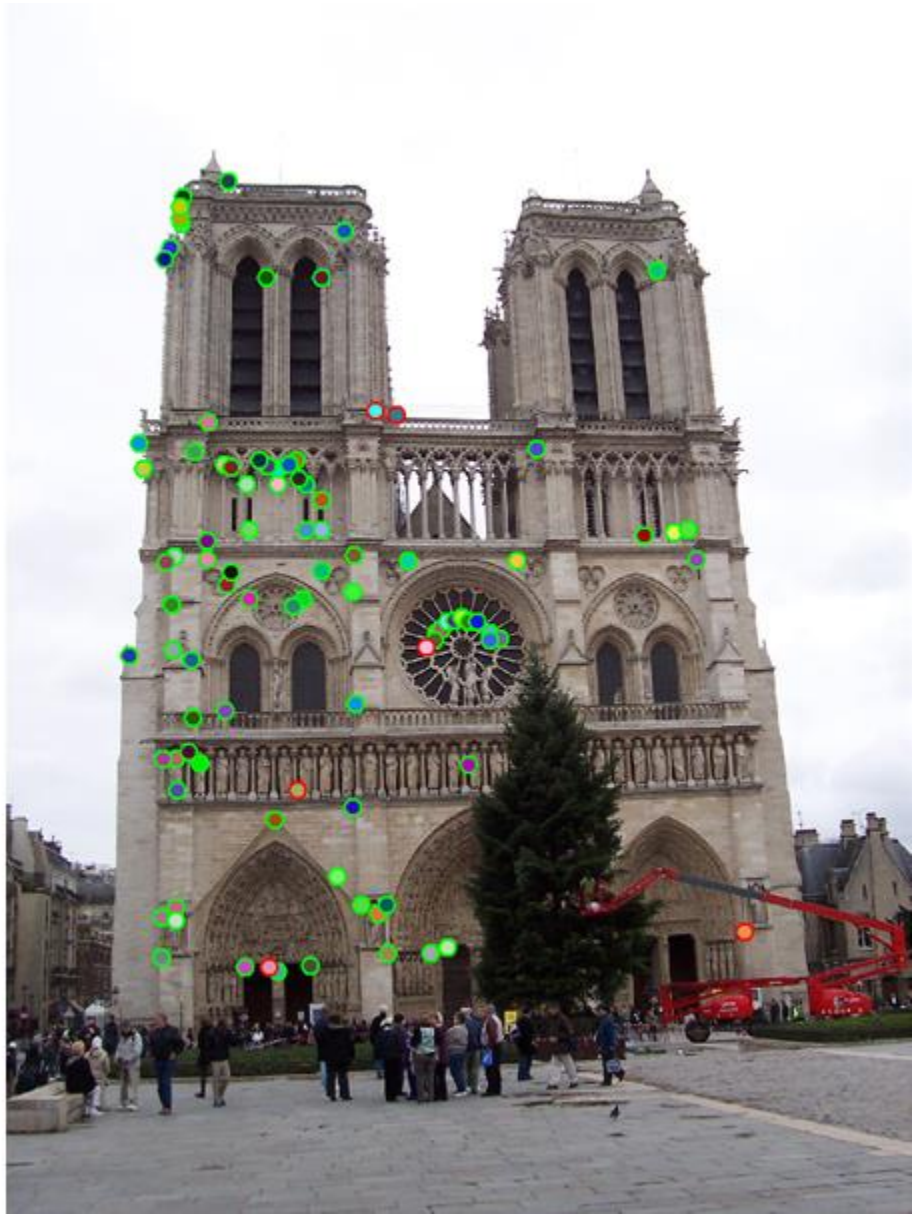

Distance: 1.22
Distance: 0.61
Distance: 0.34, 0.30, 0.40

Nearest Neighbor Distance Ratio

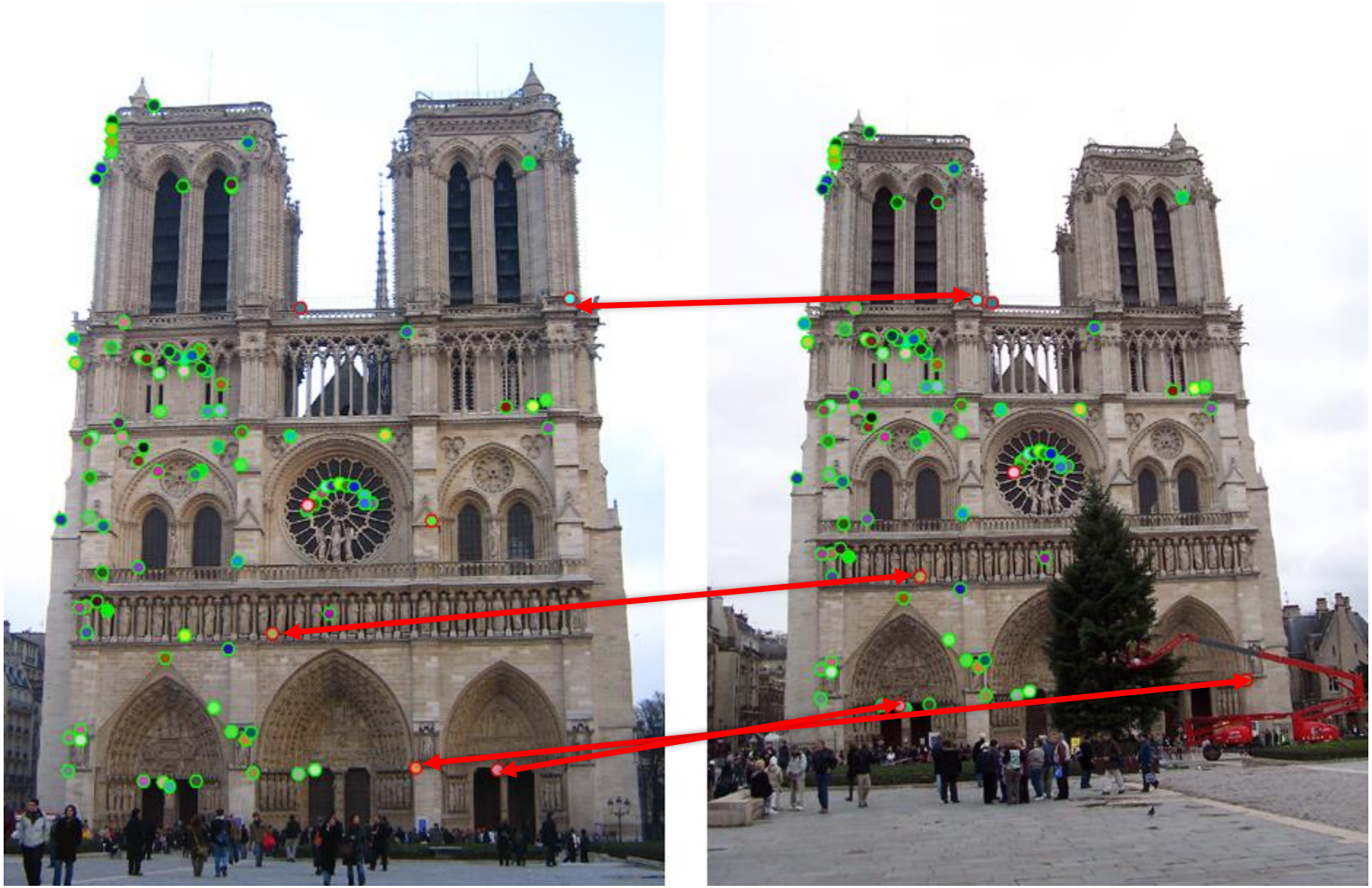
- $\frac{NN1}{NN2}$ where NN1 is the distance to the first nearest neighbor and NN2 is the distance to the second nearest neighbor.
- Sorting by this ratio (into ascending order) puts matches in order of confidence (in descending order of confidence).

Can we refine this further?

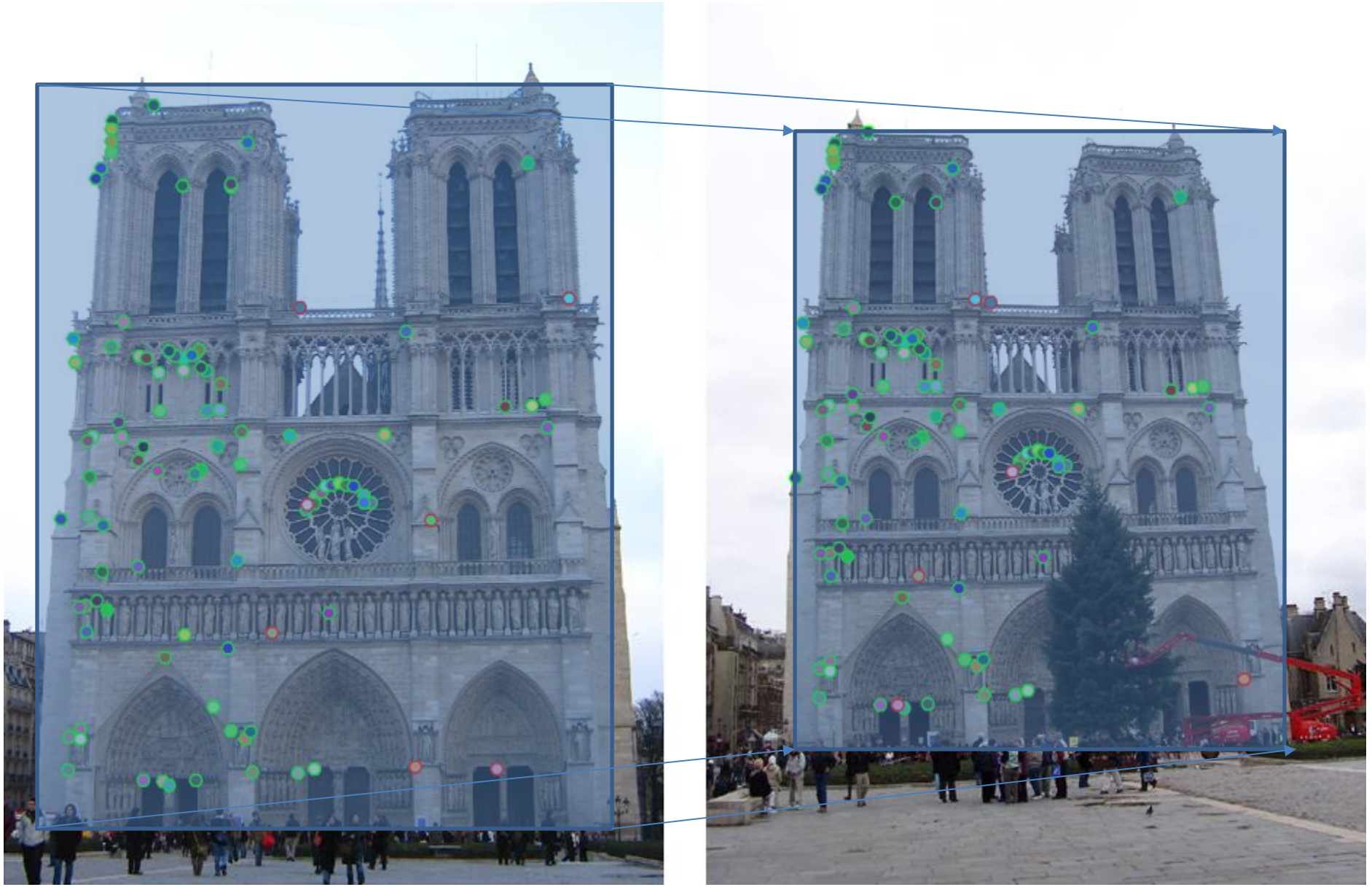




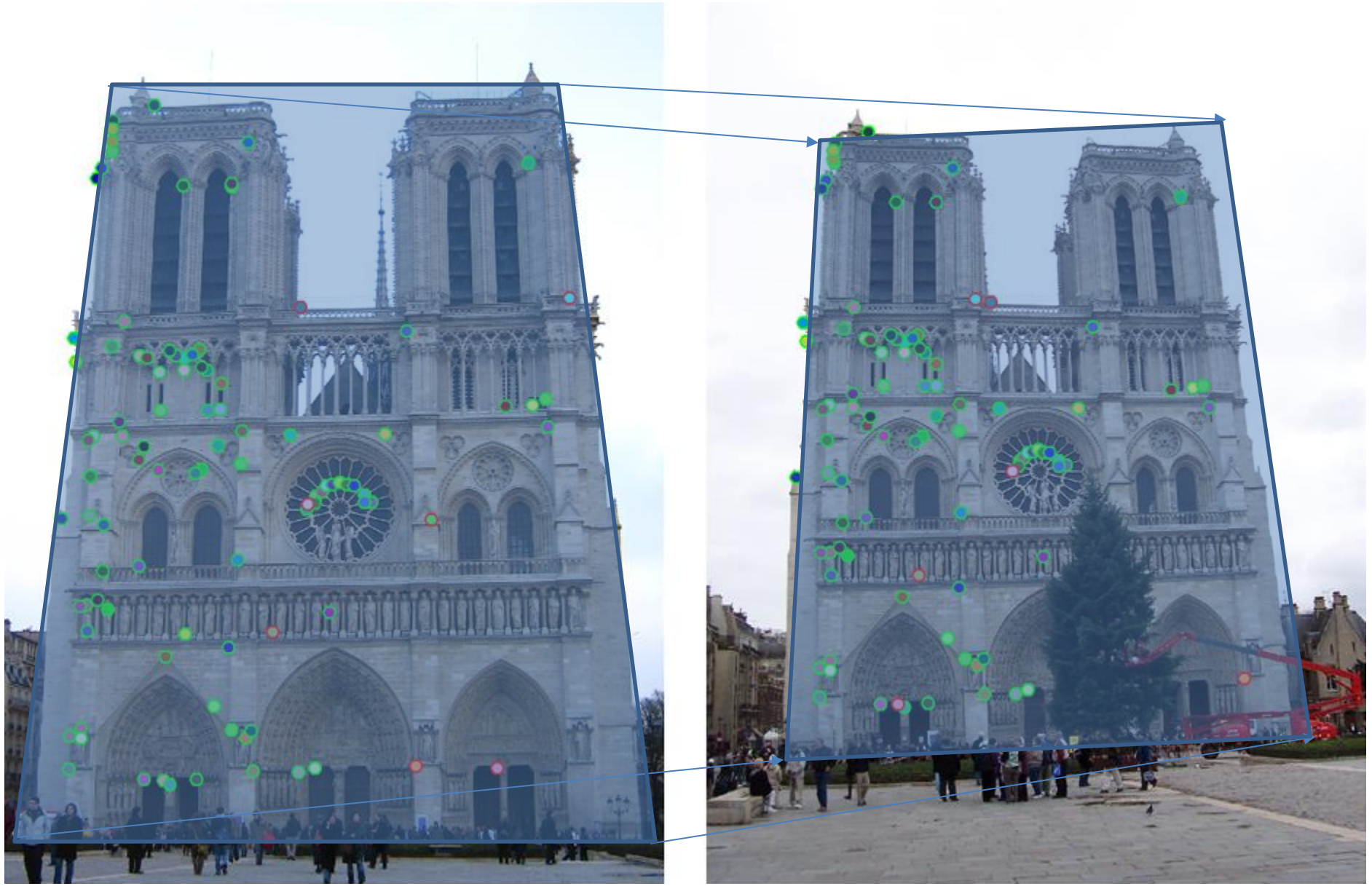
Can we refine this further?



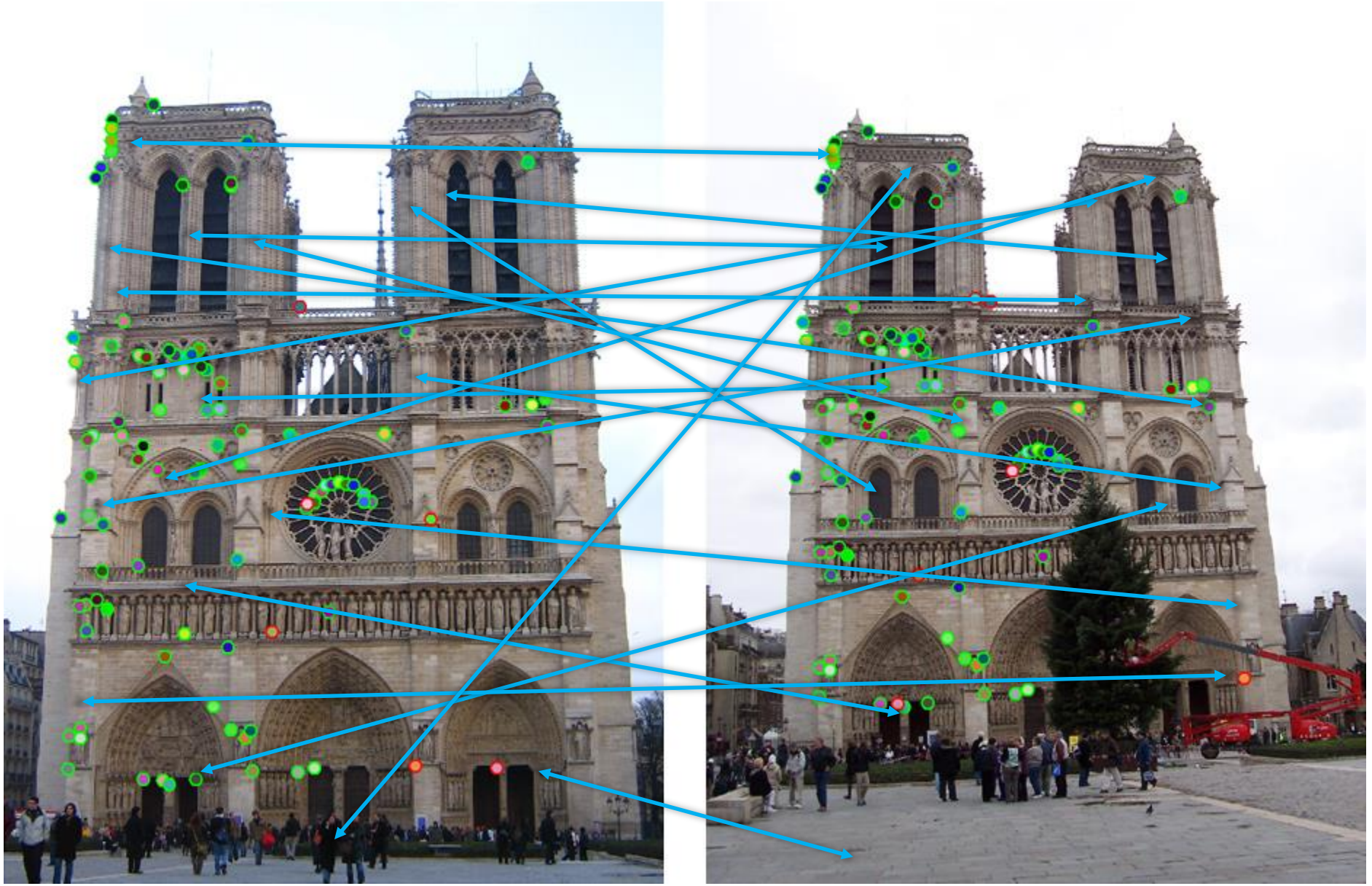
Can we refine this further?



Can we refine this further?

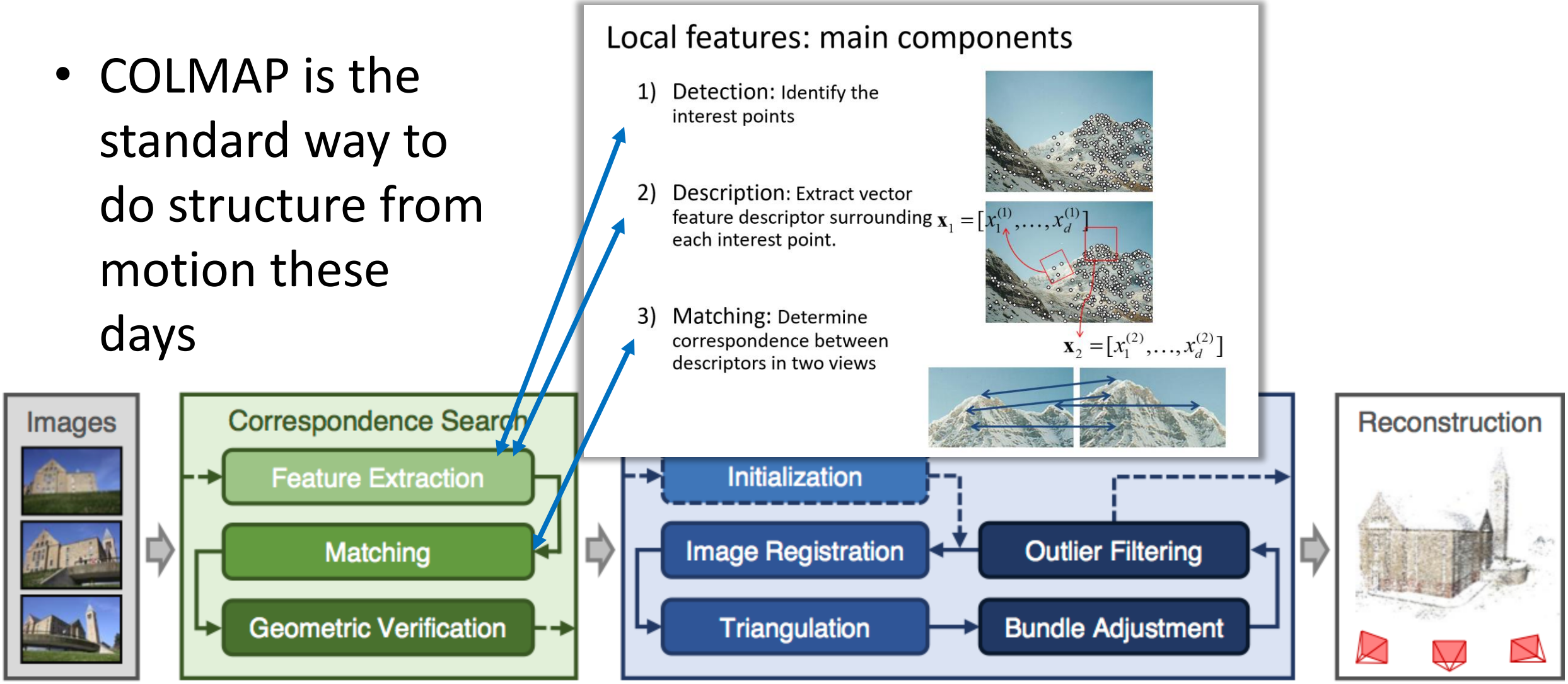


What is the space of allowable correspondences?



What is “Geometric Verification”?

- COLMAP is the standard way to do structure from motion these days



“Structure-From-Motion Revisited”. Johannes L. Schonberger, Jan-Michael Frahm; CVPR 2016
5k+ citations

Part 6: Homography with RANSAC

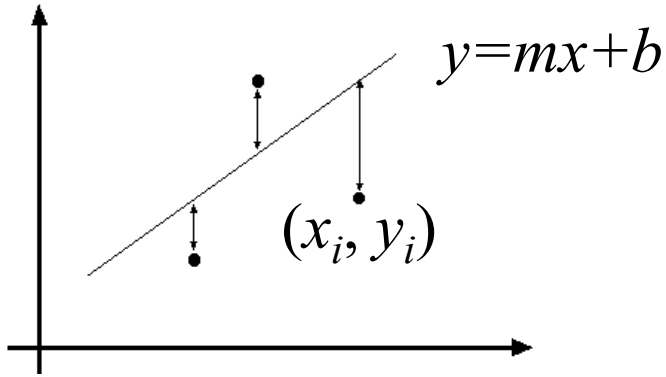
For two photographs of a scene, it's unlikely that you'd have perfect point correspondence with which to solve for the homography matrix. So, next you are going to compute the homography with point correspondences computed using SIFT. As discussed in class, a direct least-squares solution alone is not appropriate in this scenario due to the presence of multiple outliers. In order to estimate the homography from this noisy data, you'll need to use RANSAC. This technique is especially powerful for aligning images of planar surfaces, like building facades or the ground plane.

You'll use these putative point correspondences and RANSAC to find the "best" homography matrix. You will iteratively choose a minimal set of point correspondences (4 for a homography), solve for the homography matrix, and then count the number of inliers. Inliers in this context will be point correspondences that "agree" with the estimated homography. In order to count how many inliers a homography has, you'll need a distance metric. For a homography, this is typically the reprojection error: the geometric distance between a point in the second image (x') and the location predicted by mapping its corresponding point from the first image (Hx). You'll need to pick a threshold between inliers and outliers; your results are very sensitive to this threshold, so explore a range of values. You don't want to be too permissive about what you consider an inlier, nor do you want to be too conservative. Return the homography with the most inliers.

Recall from lecture the expected number of iterations of RANSAC to find the "right" solution in the presence of outliers. For example, if half of your input correspondences are wrong, then you have a $(0.5)^4 = 6.25\%$

But first…

Let’s talk about how we even find the parameters for a model

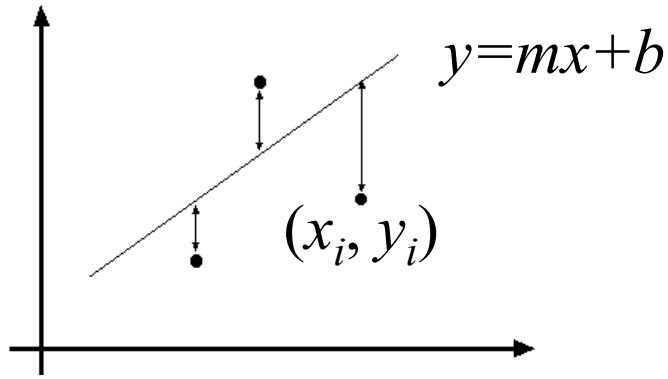


Fitting: find the parameters of a model that best fit the data

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
- Iterative Closest Points (ICP)

Fitting and Alignment: Methods

- Global optimization / Search for parameters
  - Least squares fit
  - Robust least squares
  - Other parameter search methods

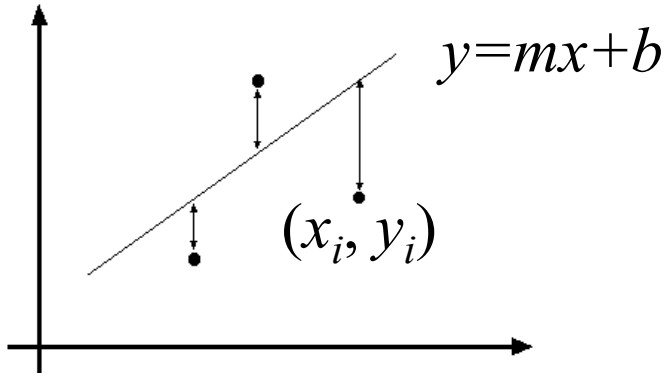
- Hypothesize and test
  - Generalized Hough transform
  - RANSAC

- Iterative Closest Points (ICP)

Simple example: Fitting a line

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



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

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

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

$$\frac{dE}{dp} = 2\mathbf{A}^T \mathbf{Ap} - 2\mathbf{A}^T \mathbf{y} = 0$$

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

Matlab: `p = A \ y;`

Python:
```
p = numpy.linalg.lstsq(A, y)
```

Modified from S. Lazebnik

Least squares (global) optimization

Good

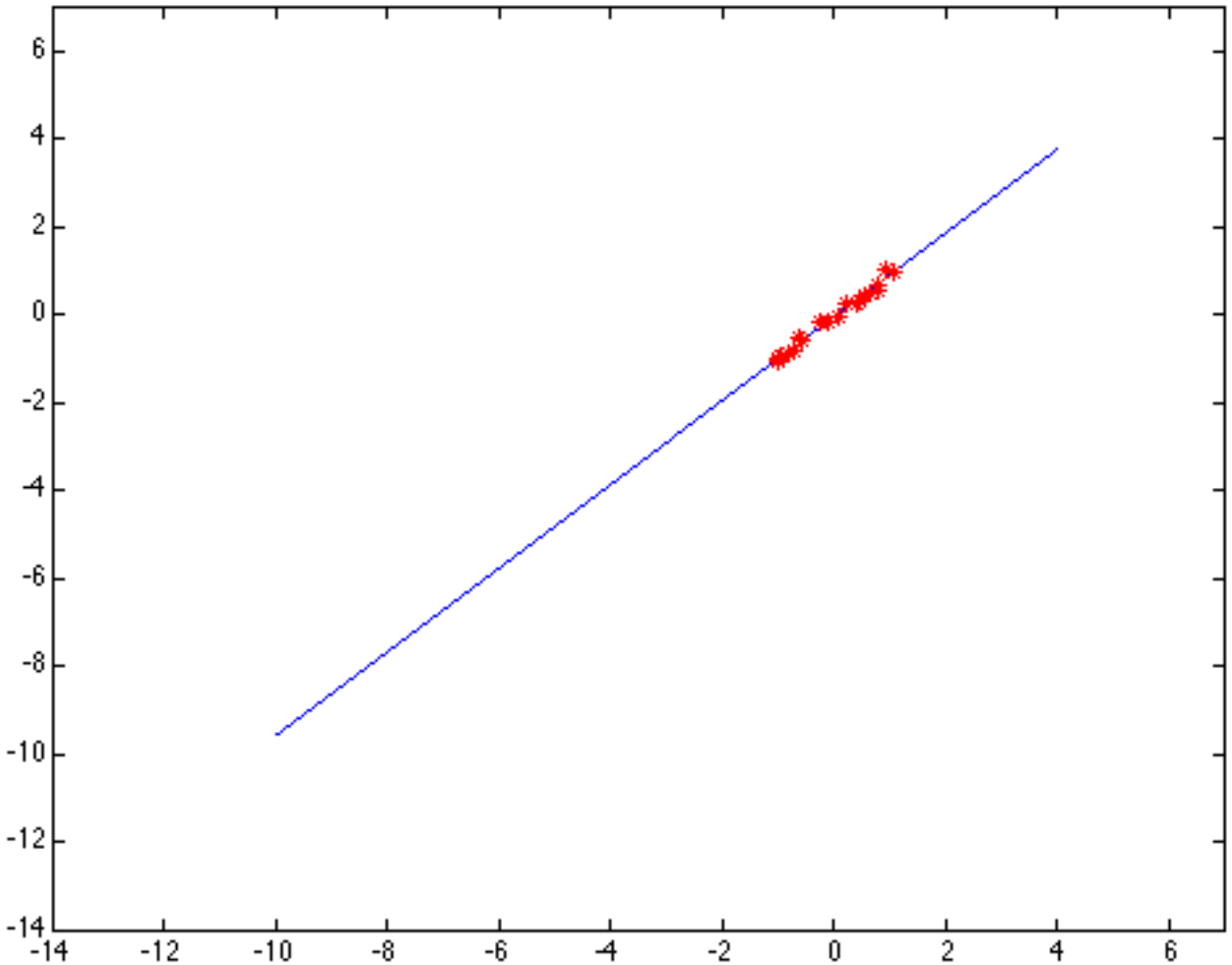
- Clearly specified objective
- Optimization is easy

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.

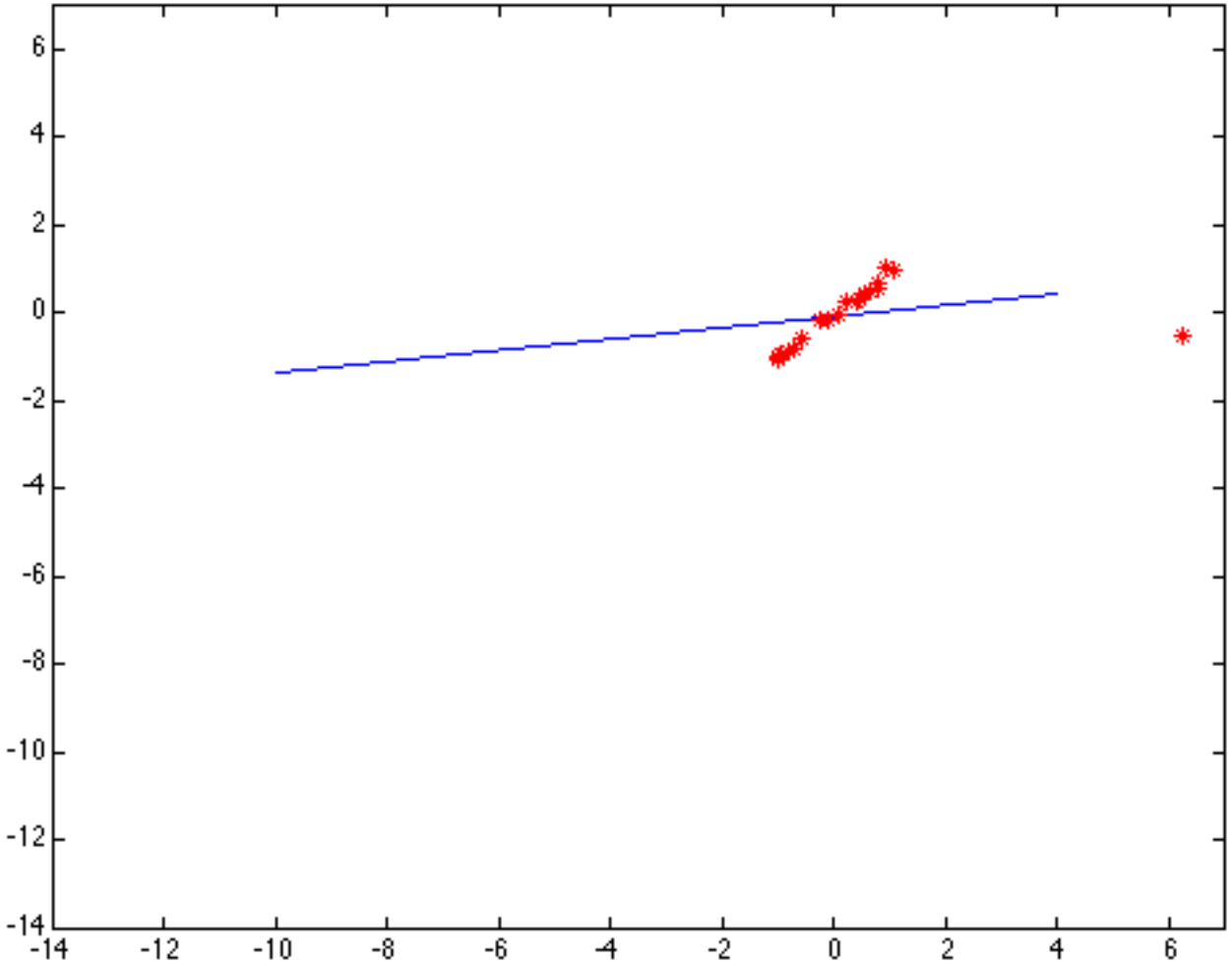
Least squares: Robustness to noise

- 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
- Iterative Closest Points (ICP)

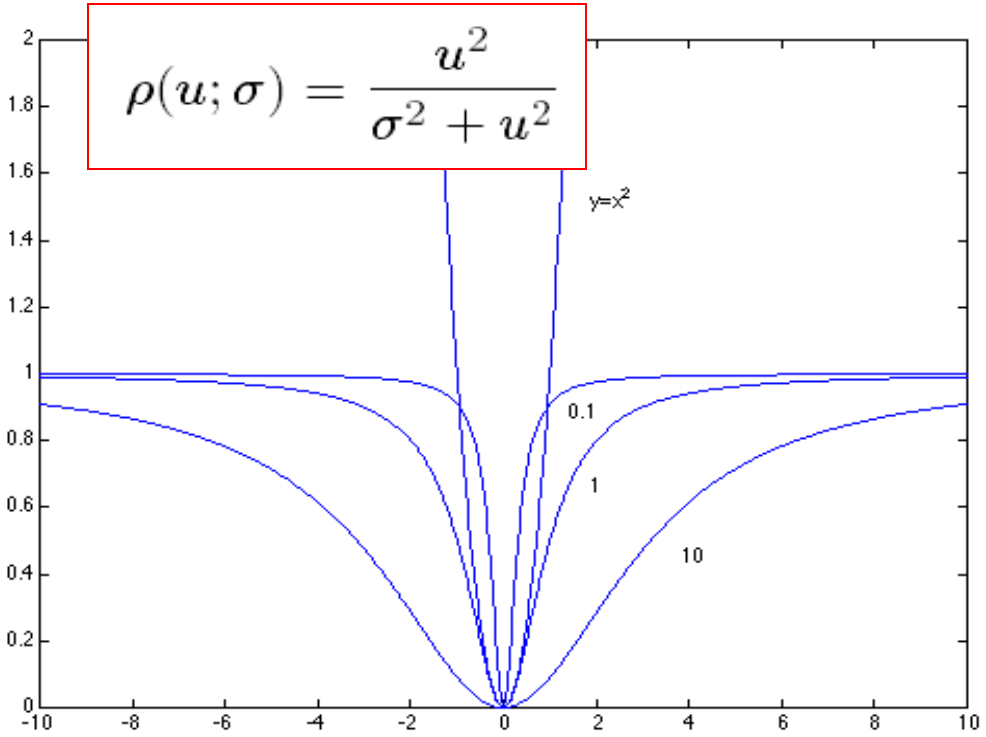
Robust least squares (to deal with outliers)

General approach:

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

$u_i(x_i, \theta)$ – residual of ith point w.r.t. model parameters ϑ

ρ – robust function with scale parameter σ

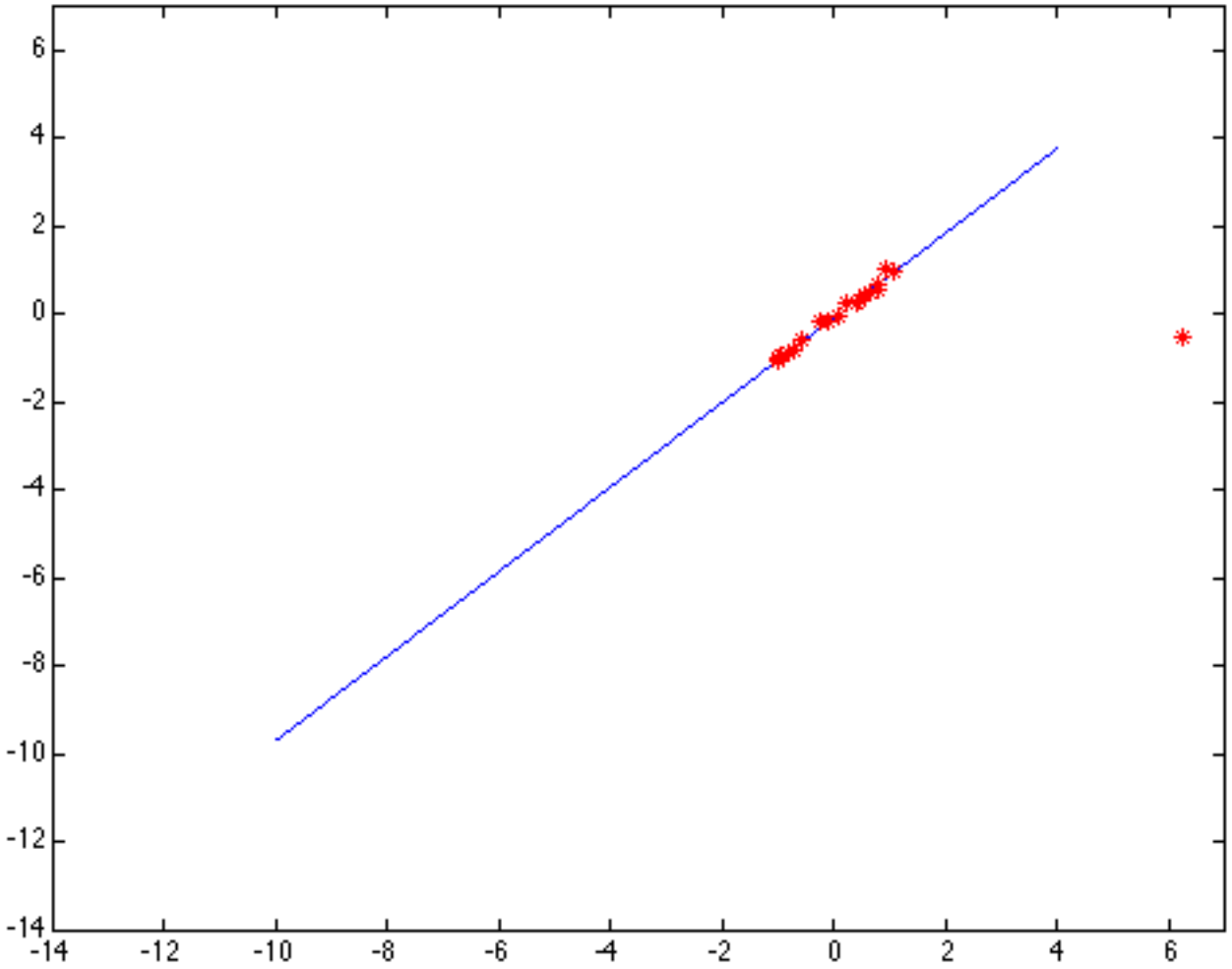


The robust function ρ

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

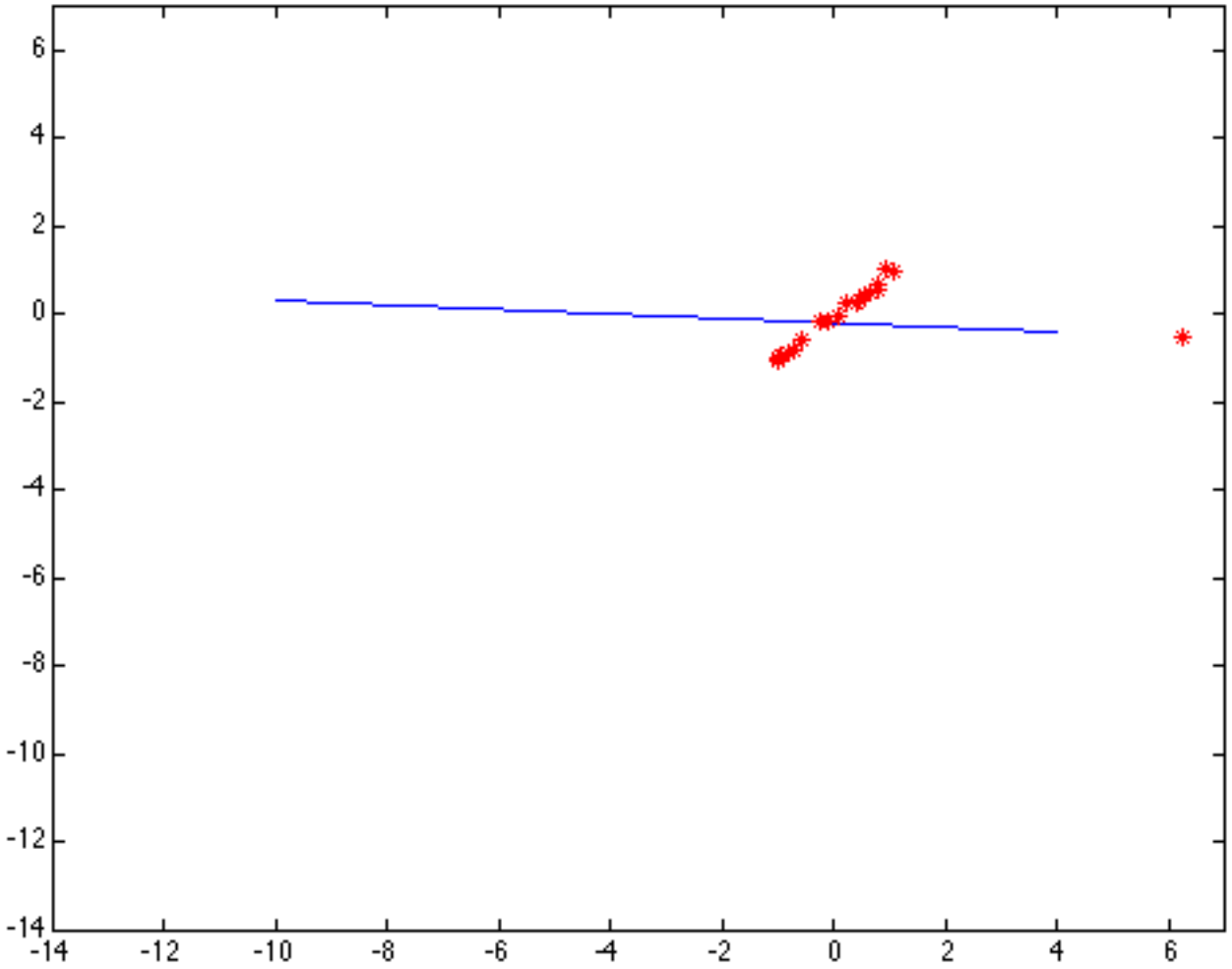
Slide from S. Savarese

Choosing the scale: Just right



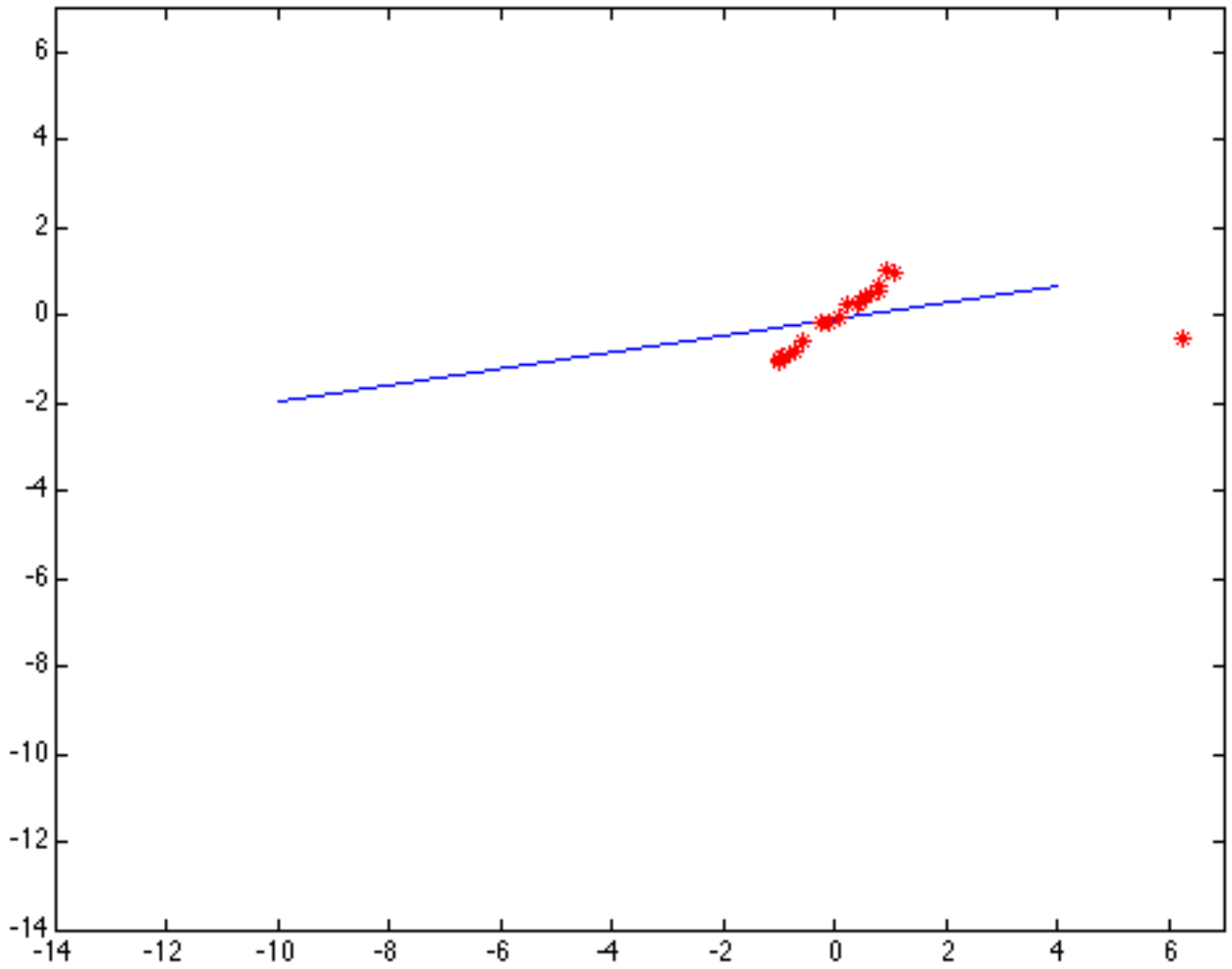
The effect of the outlier is minimized

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
- Iterative Closest Points (ICP)

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

- Iterative Closest Points (ICP)

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
- Iterative Closest Points (ICP)

Hough Transform: Outline

1. Create a grid of parameter values

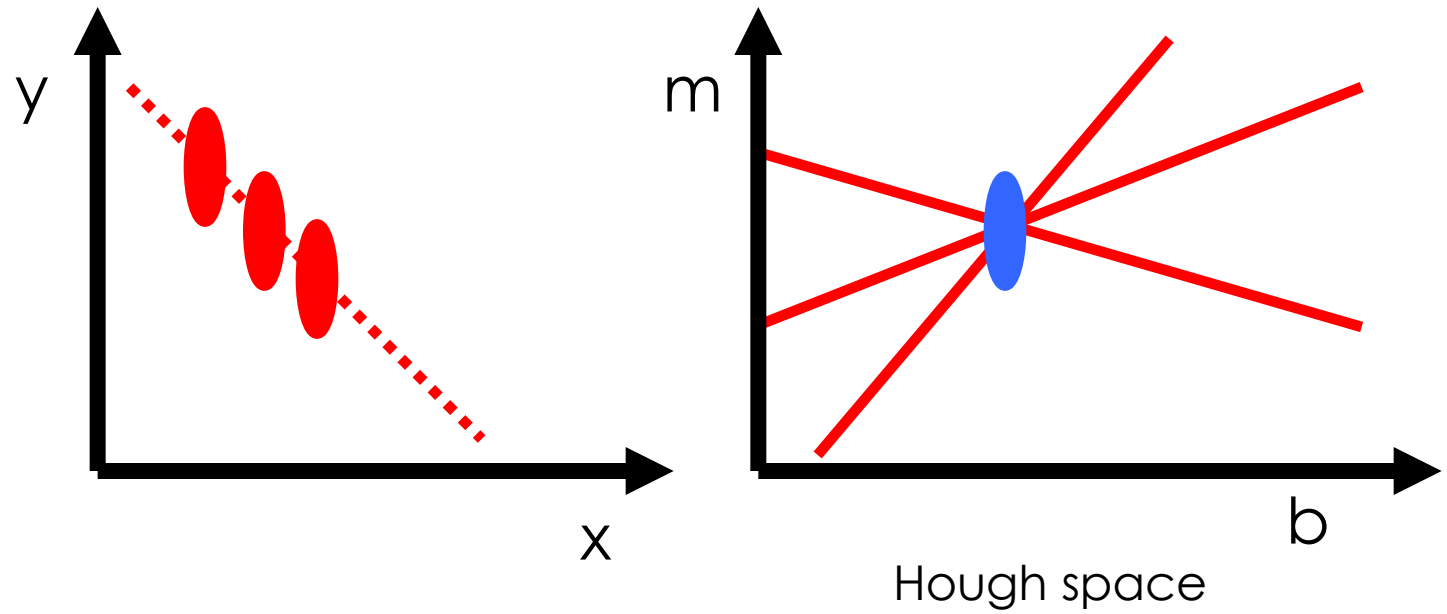
2. Each point (or observation of correspondence) votes for a set of parameters, incrementing those values in grid

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

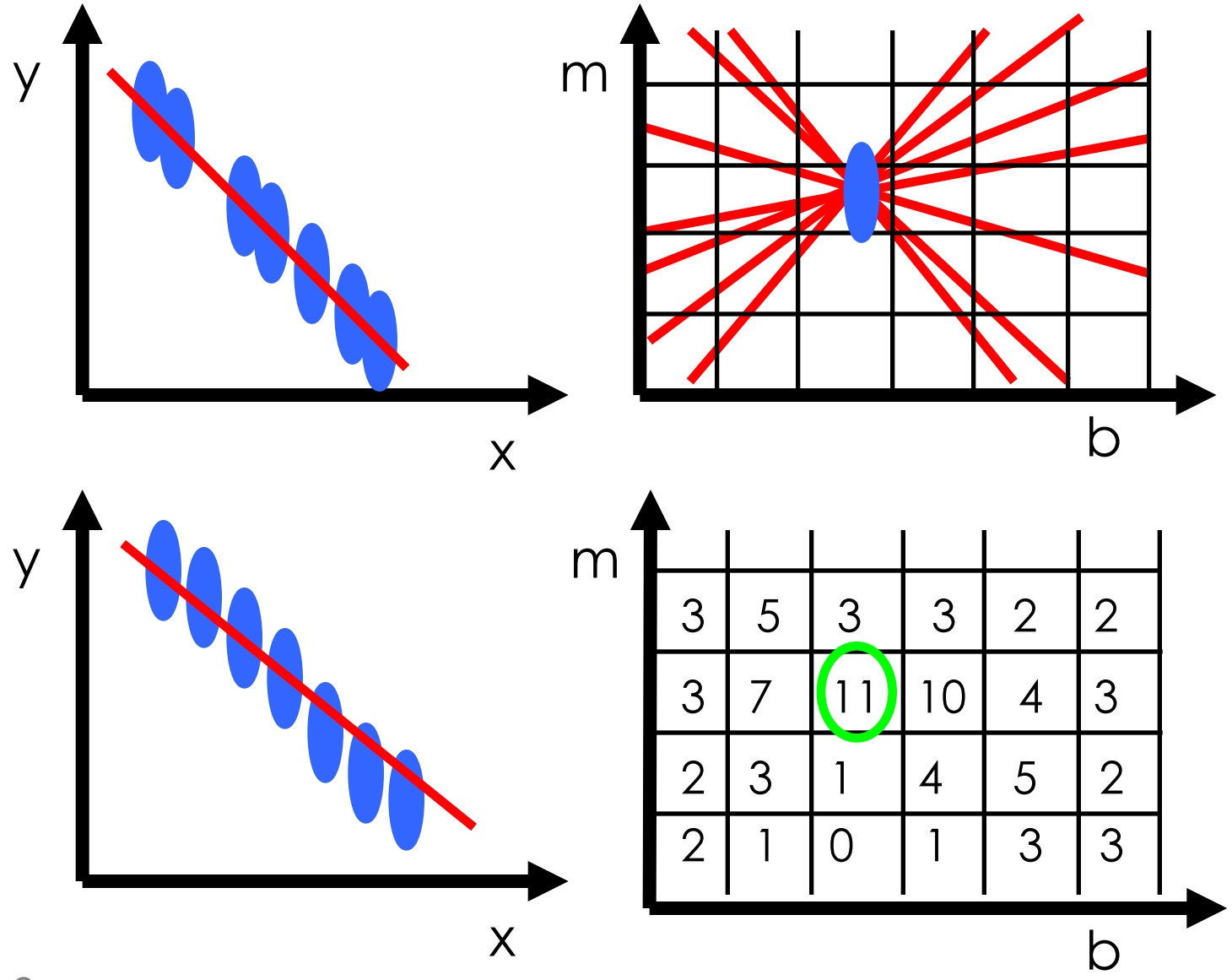
Given a set of points, find the line that explains the data points best



$$y = m\,x + b$$

Slide from S. Savarese

Hough transform



Slide from S. Savarese

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…

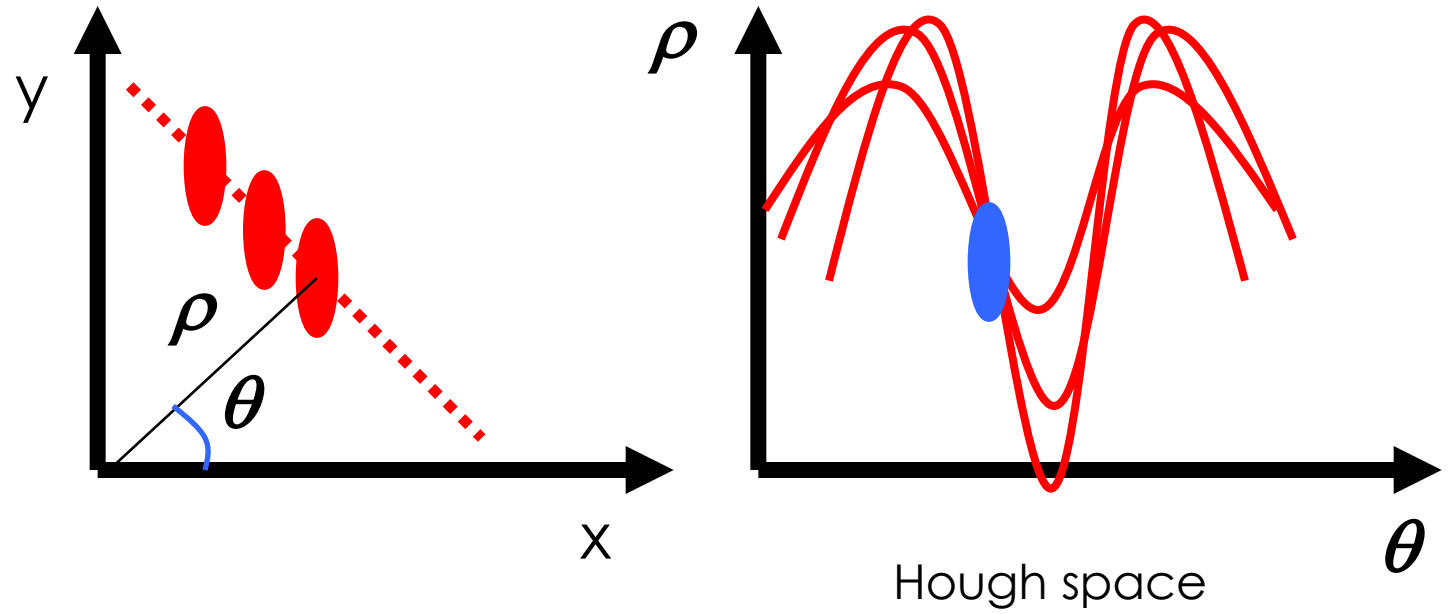
Slide from S. Savarese

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…

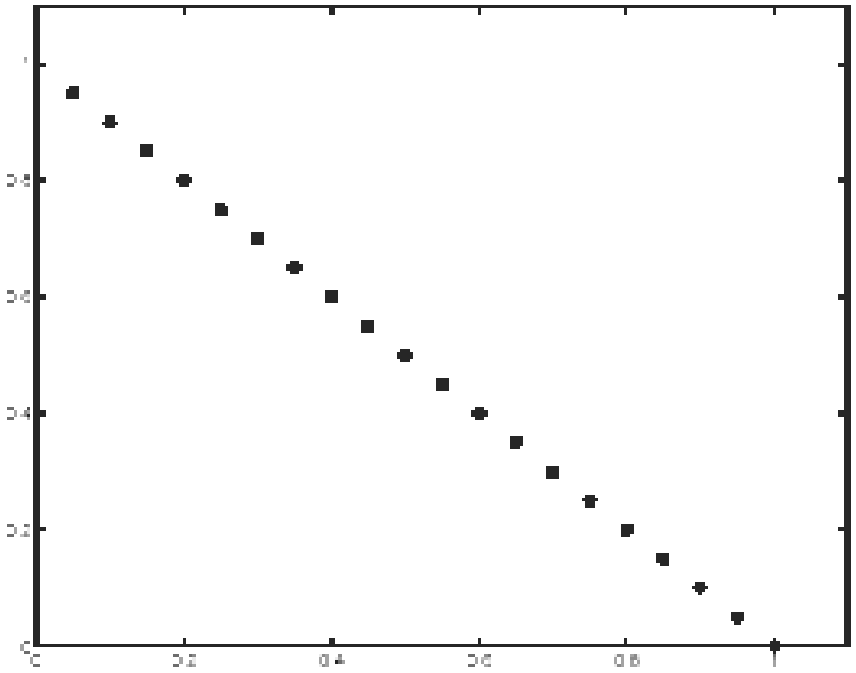
Use a polar representation for the parameter space



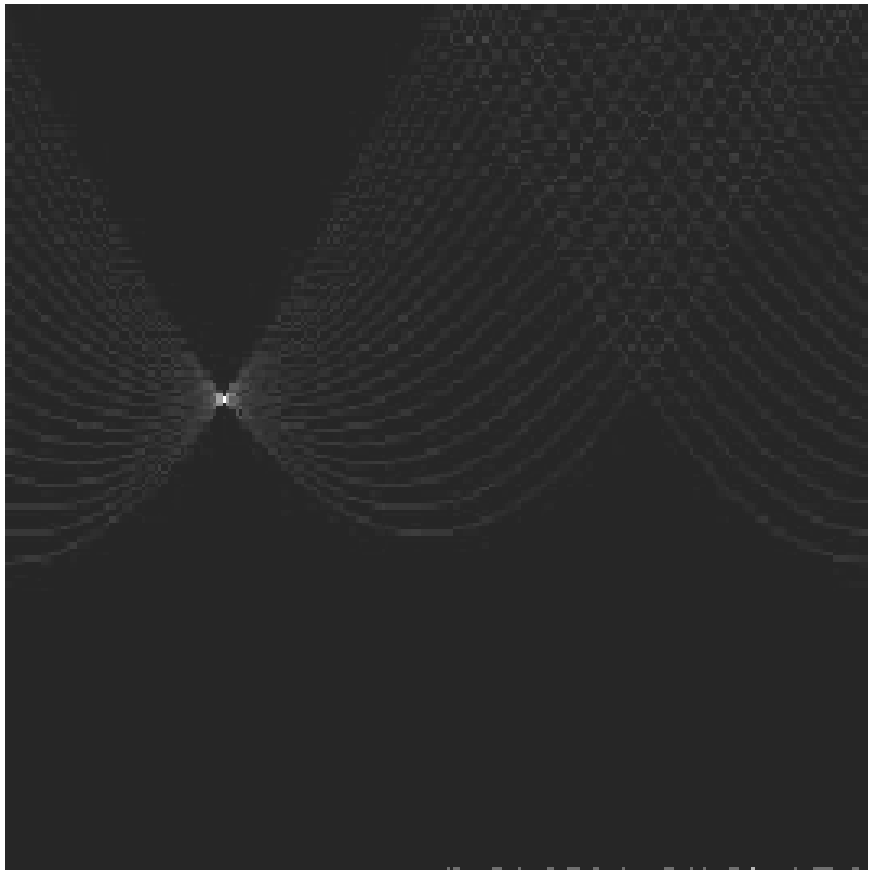
$$x \cos\theta + y \sin\theta = \rho$$

Slide from S. Savarese

Hough transform - experiments



features

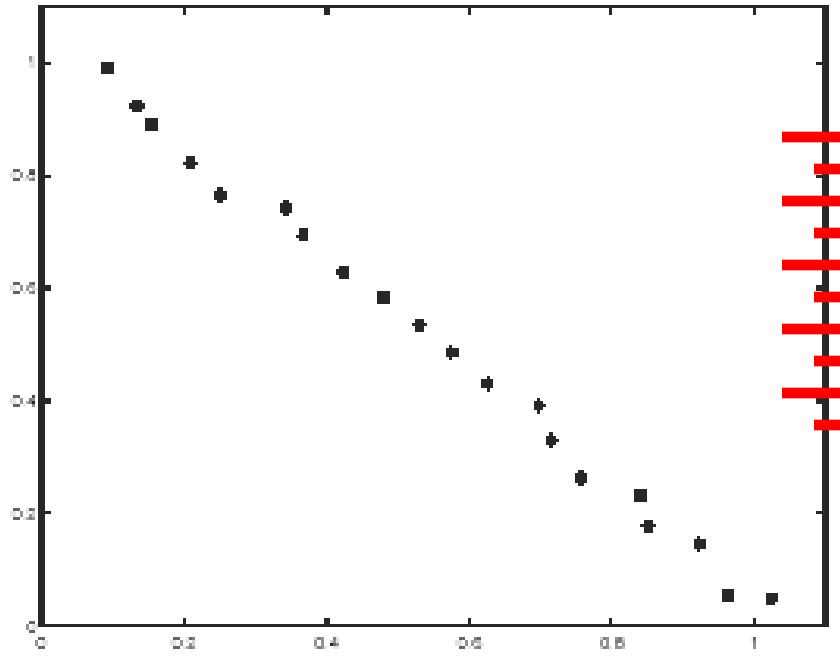


votes

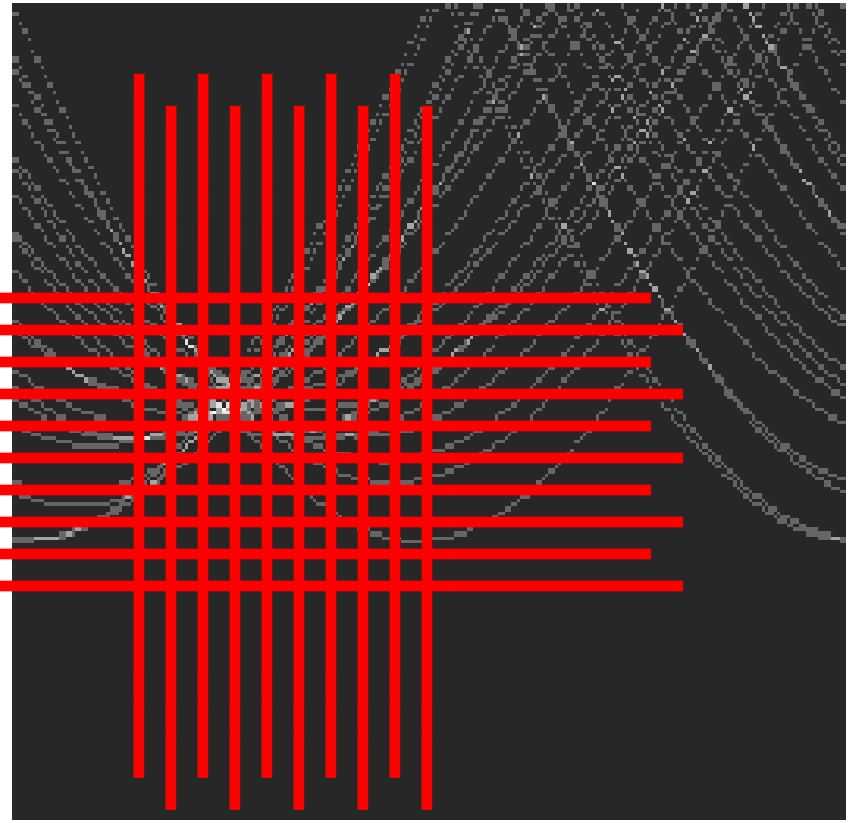
Slide from S. Savarese

Hough transform - experiments

Noisy data



features

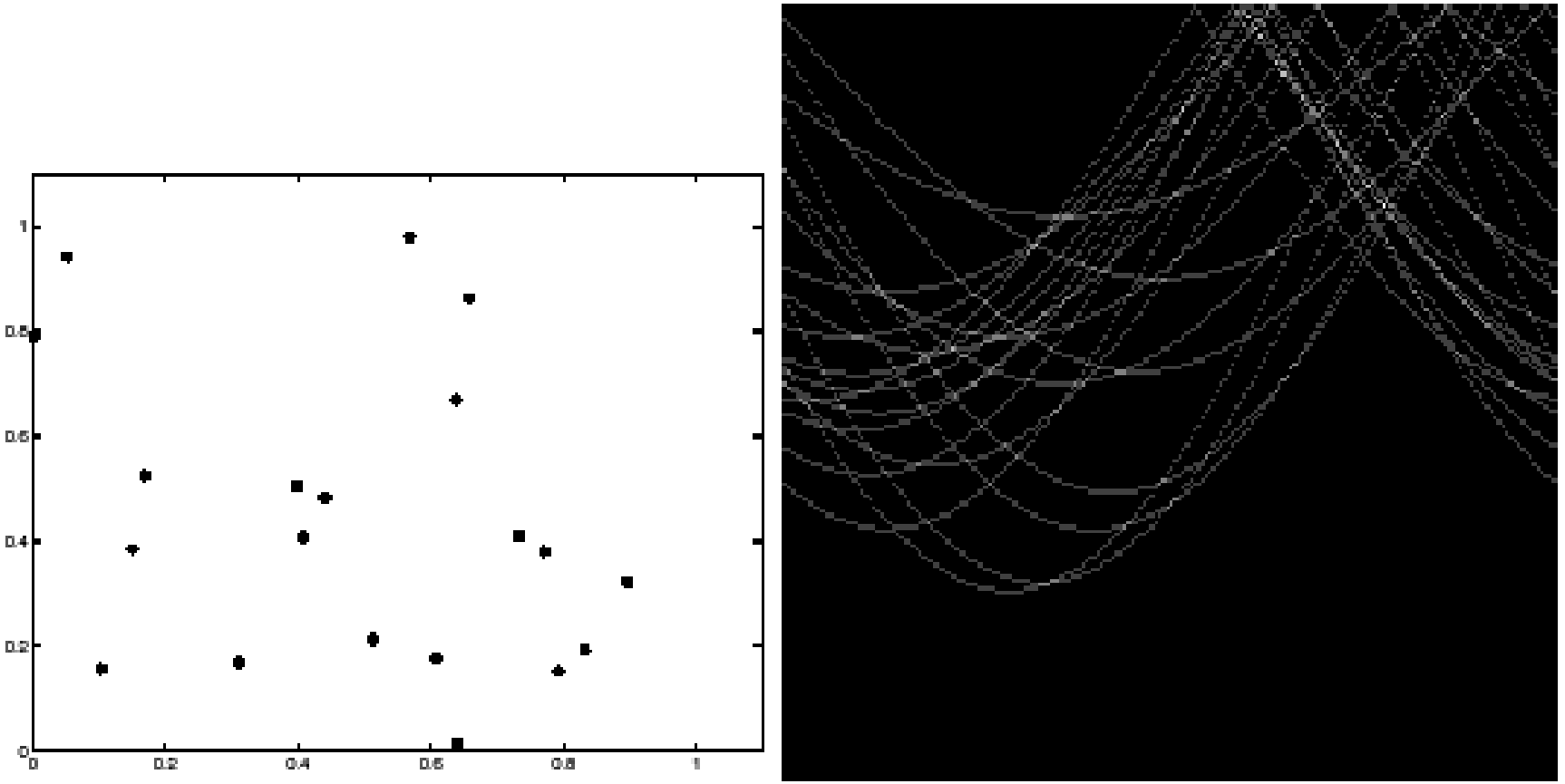


votes

Need to adjust grid size or smooth

Slide from S. Savarese

Hough transform - experiments



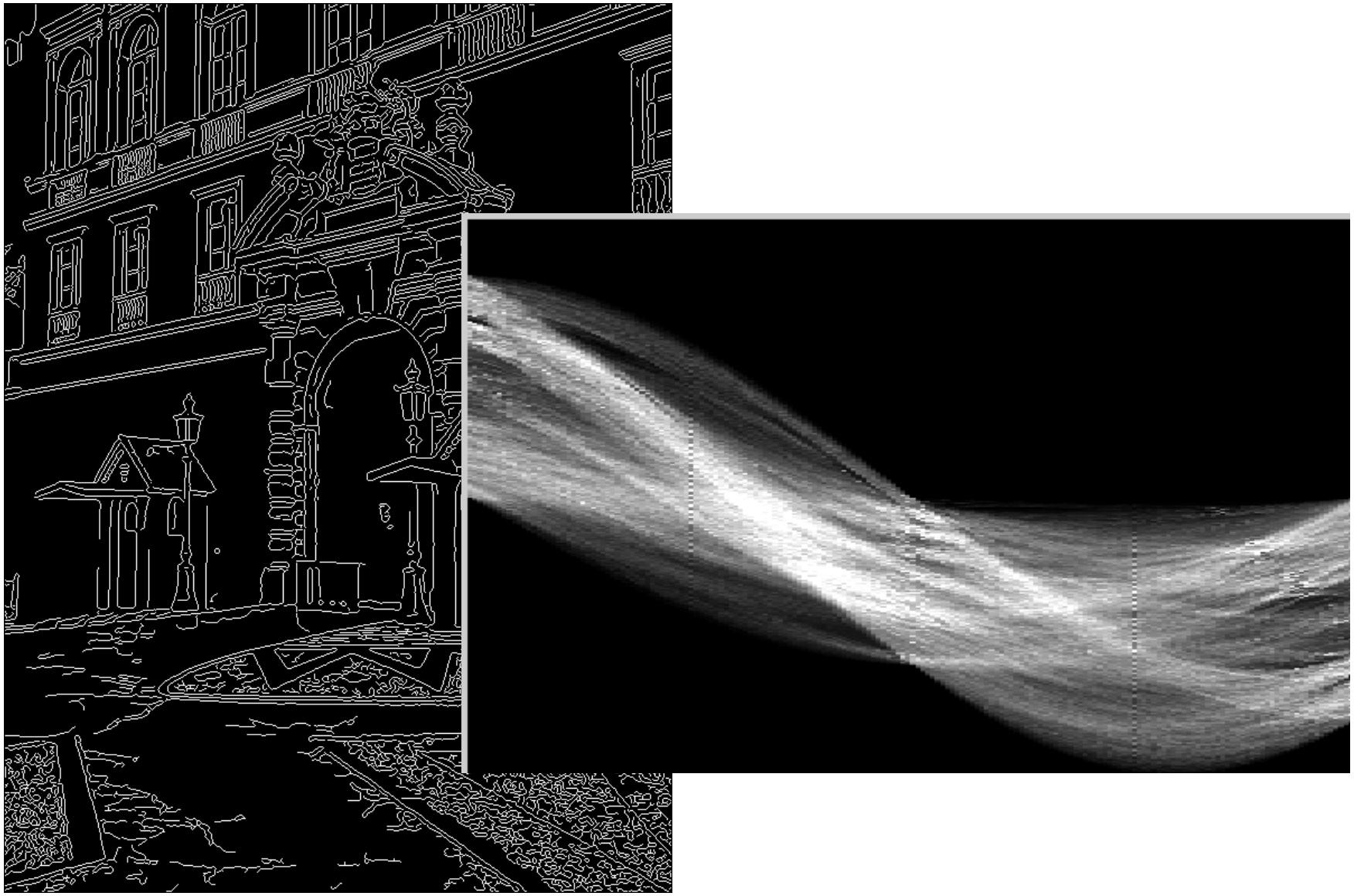
features votes

Issue: spurious peaks due to uniform noise

Slide from S. Savarese

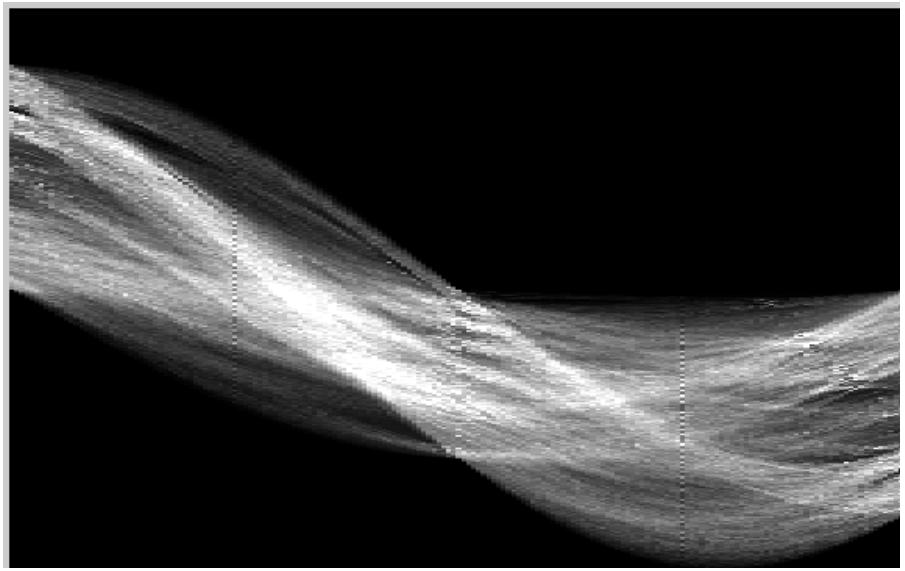
1. Image → Canny Edge Detection

2. Canny → Hough votes

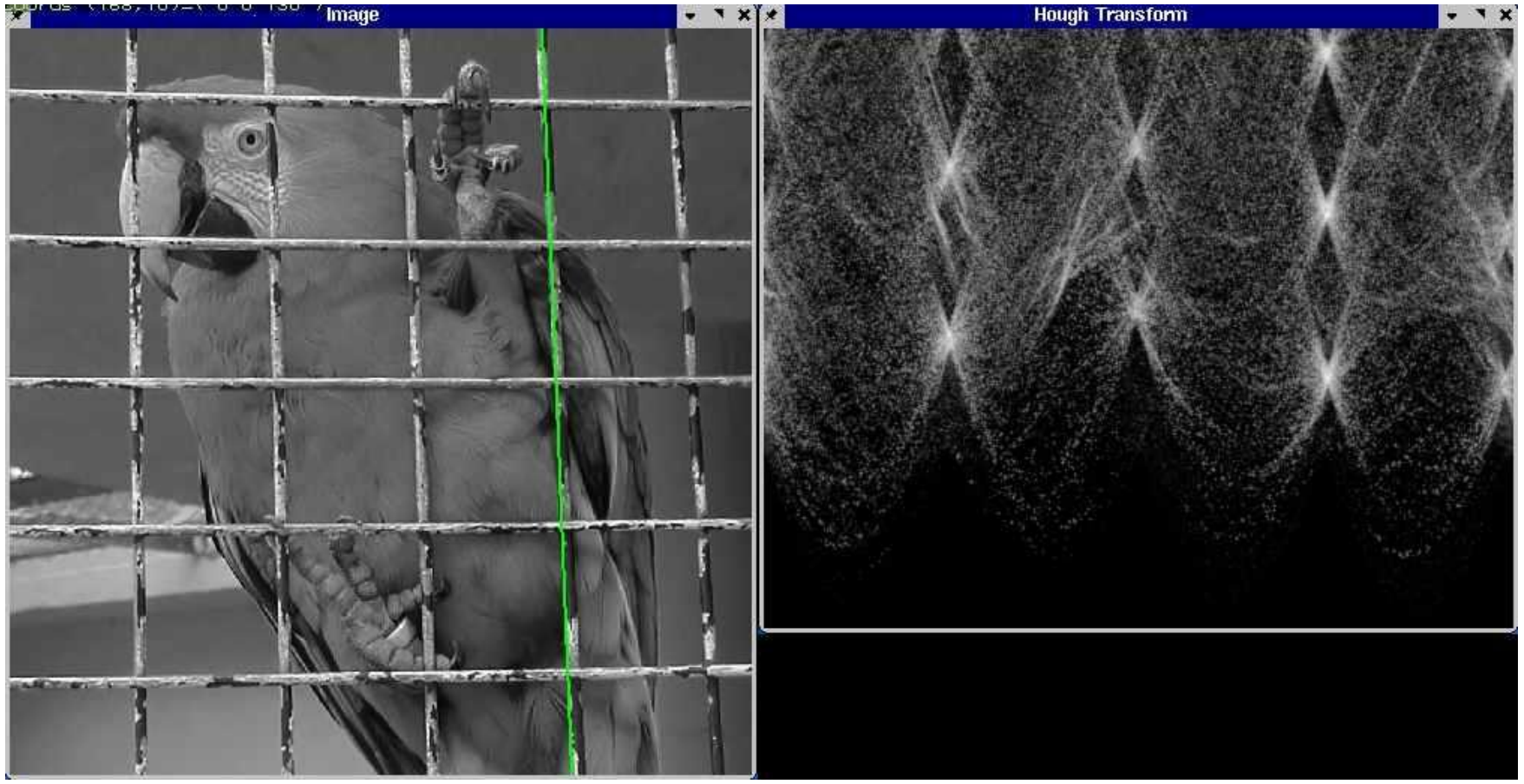


3. Hough votes → Edges

Find peaks and post-process



Hough transform example



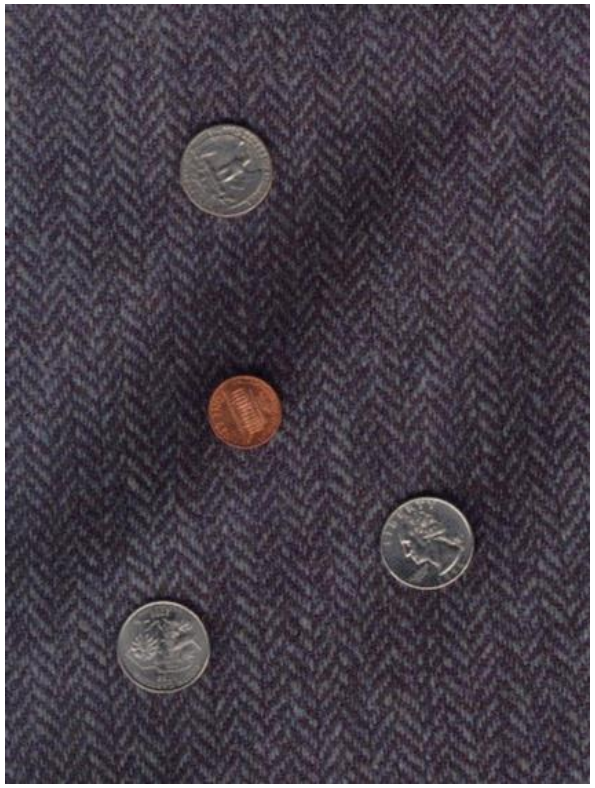
http://ostatic.com/files/images/ss_hough.jpg

Hough Transform

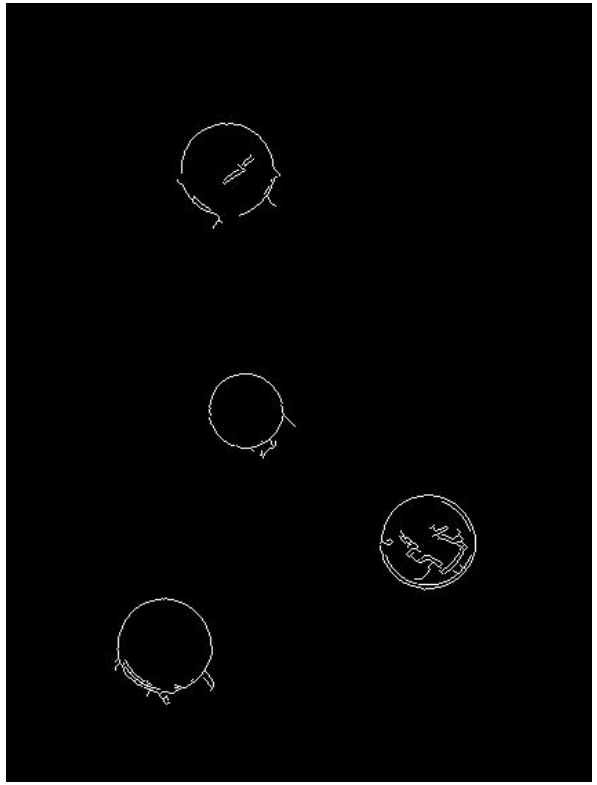
- How would we find circles?
  - Of fixed radius
  - Of unknown radius
  - Of unknown radius but with known edge orientation

Example: detecting circles with Hough

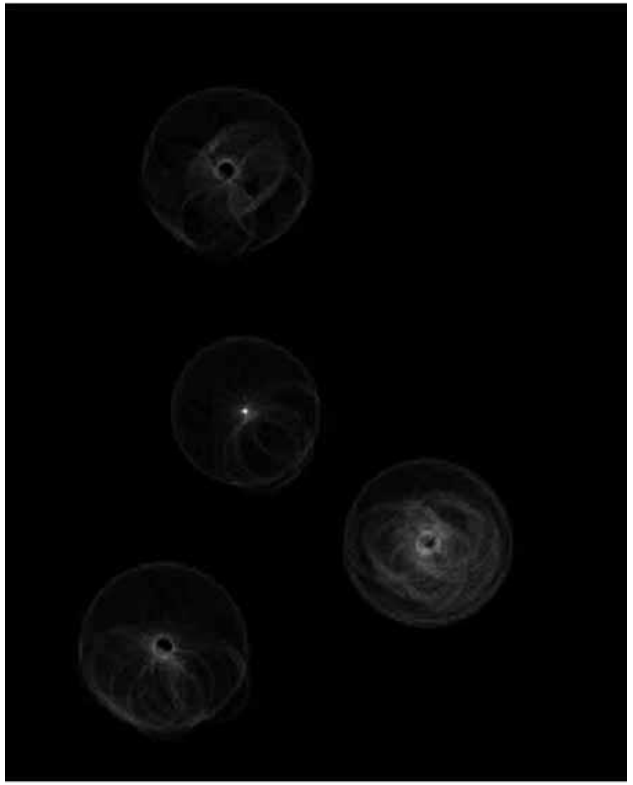
Original



Edges



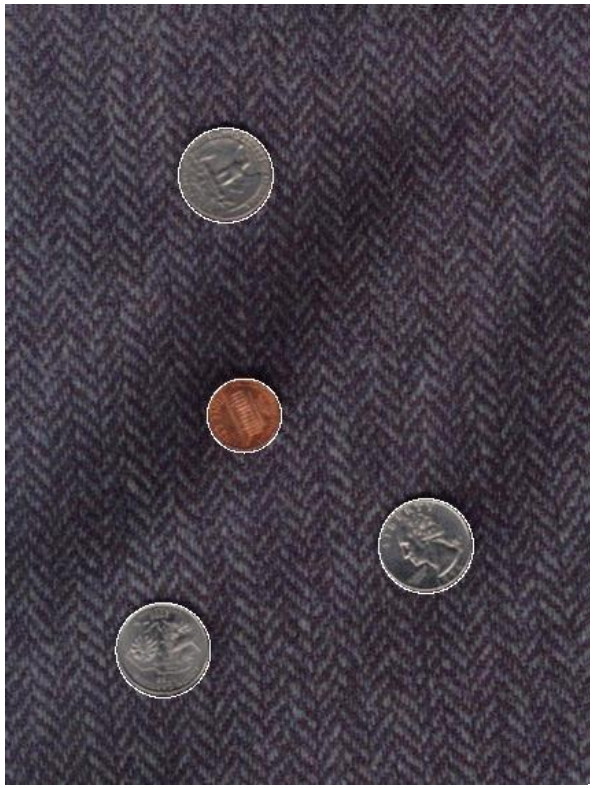
Votes: Penny



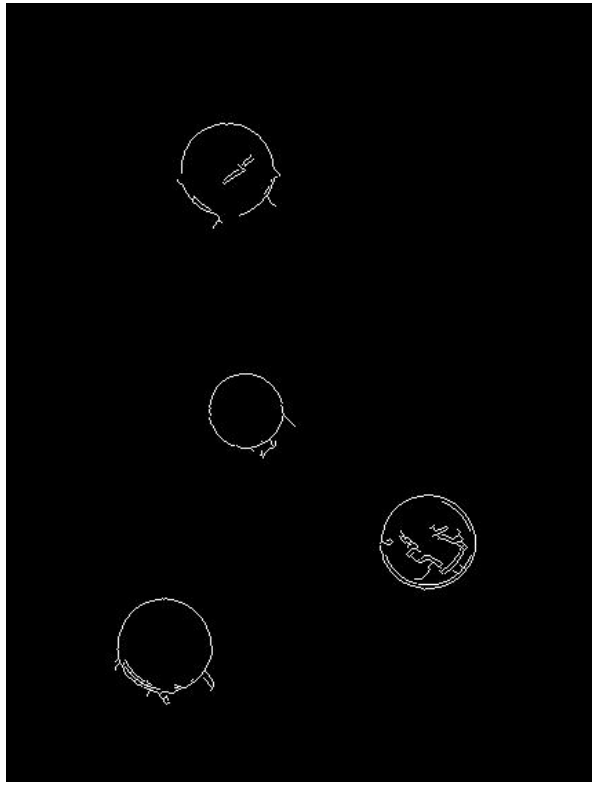
Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

Example: detecting circles with Hough

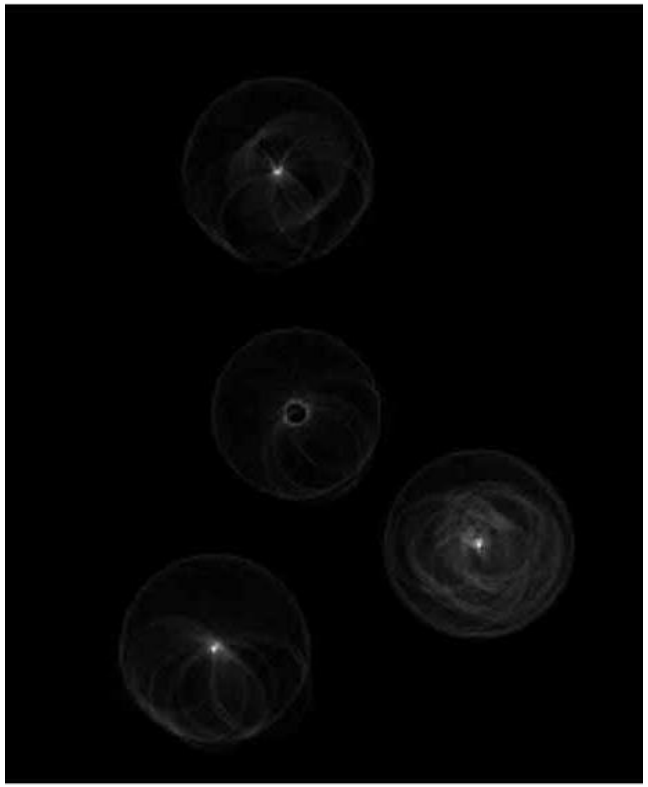
Combined detections



Edges



Votes: Quarter



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

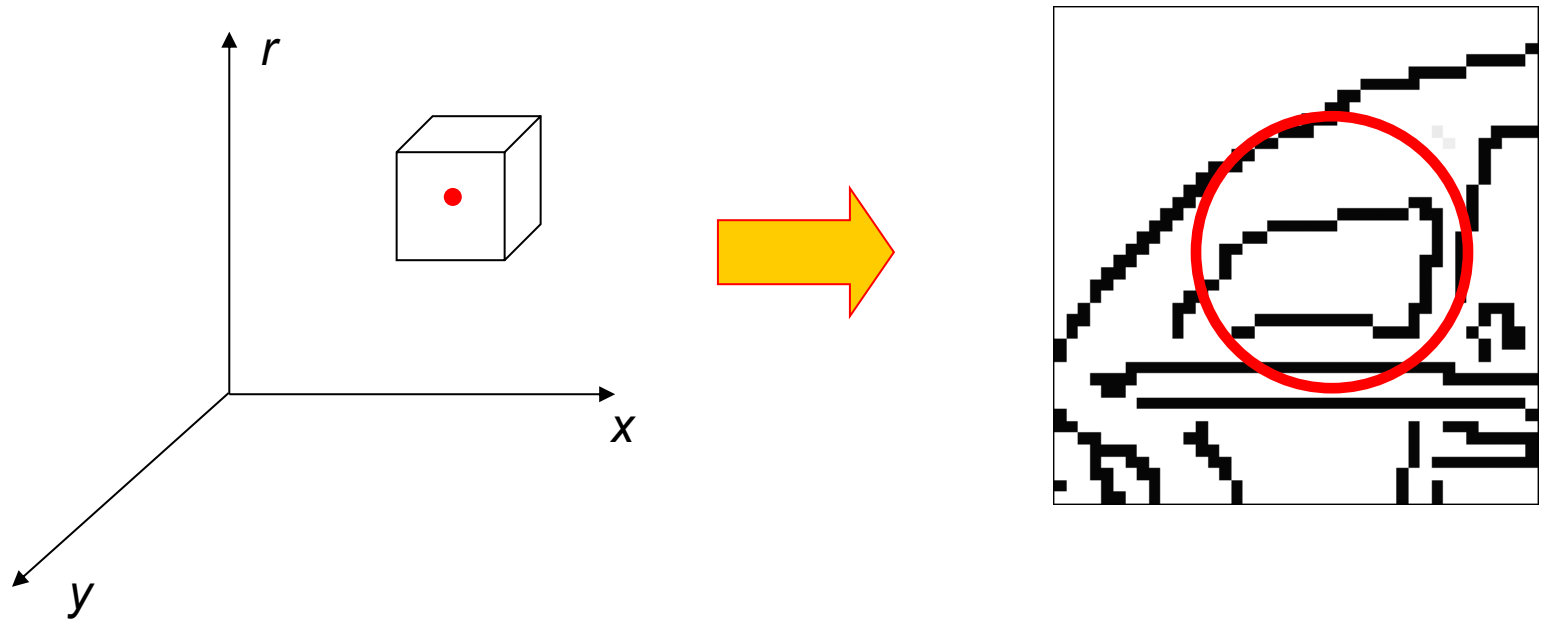
Coin finding sample images from: Vivek Kwatra

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

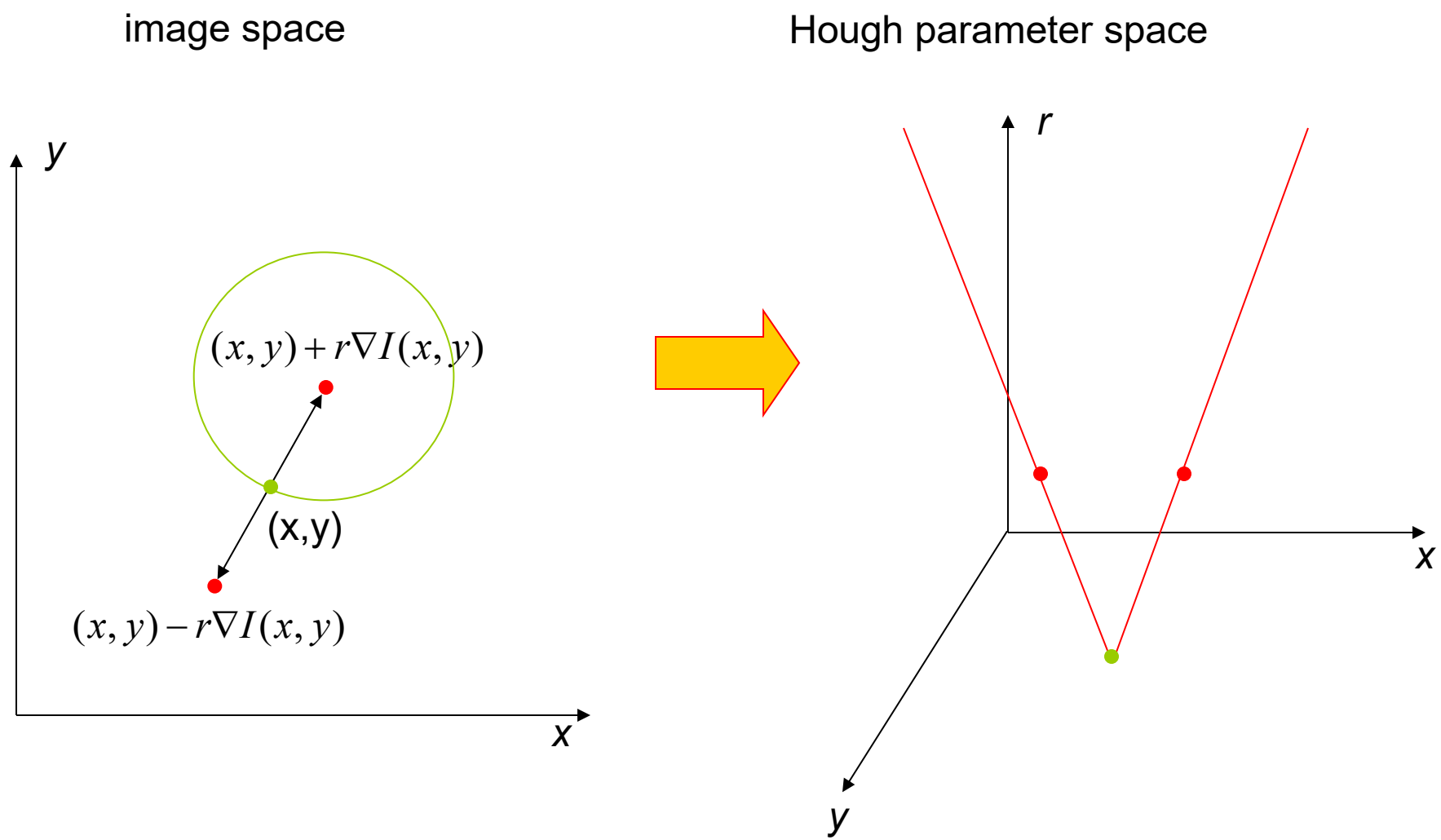
- Grid search equivalent procedure: for each (x,y,r), draw the corresponding circle in the image and compute its "support"



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



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)

Fitting and Alignment: Methods

- Global optimization / Search for parameters
  - Least squares fit
  - Robust least squares
  - Other parameter search methods

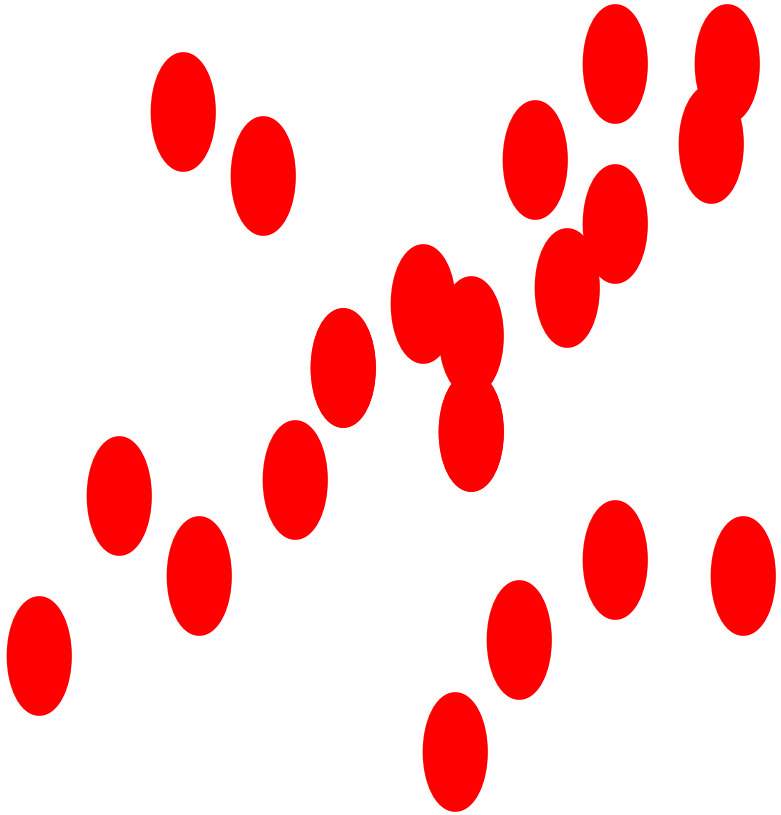
- Hypothesize and test
  - Generalized Hough transform
  - RANSAC

- Iterative Closest Points (ICP)

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in ‘81.



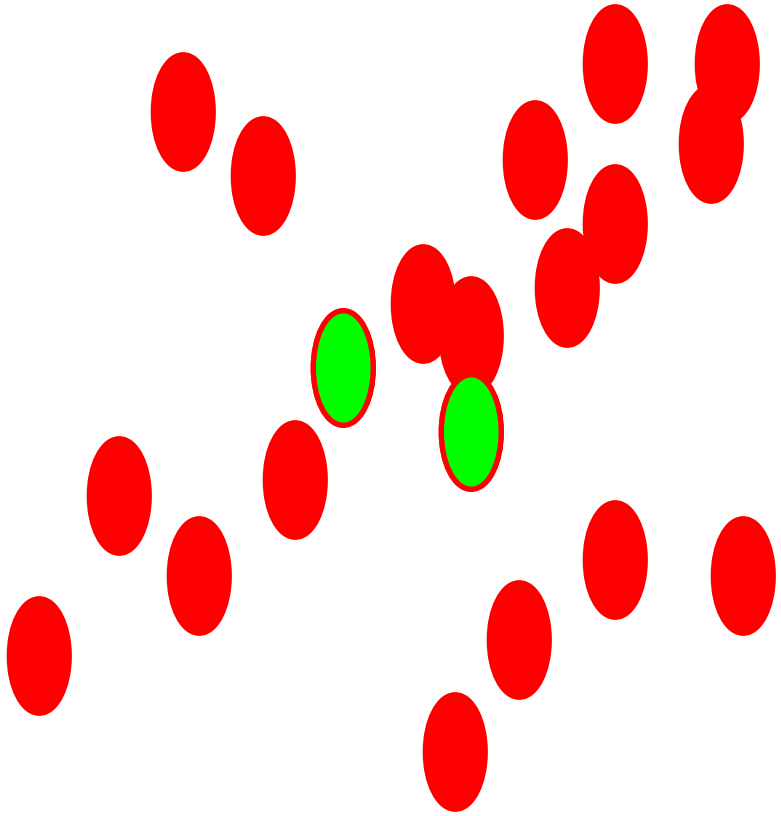
Algorithm:

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

Line fitting example



Algorithm:

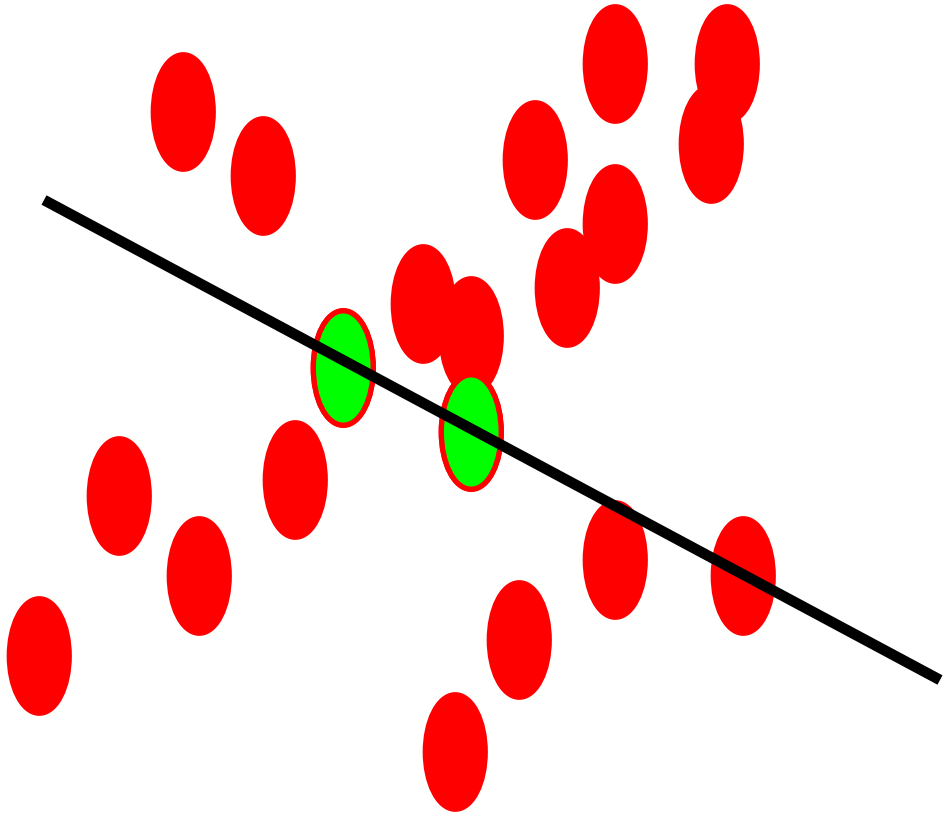
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

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

Illustration by Savarese

RANSAC

Line fitting example



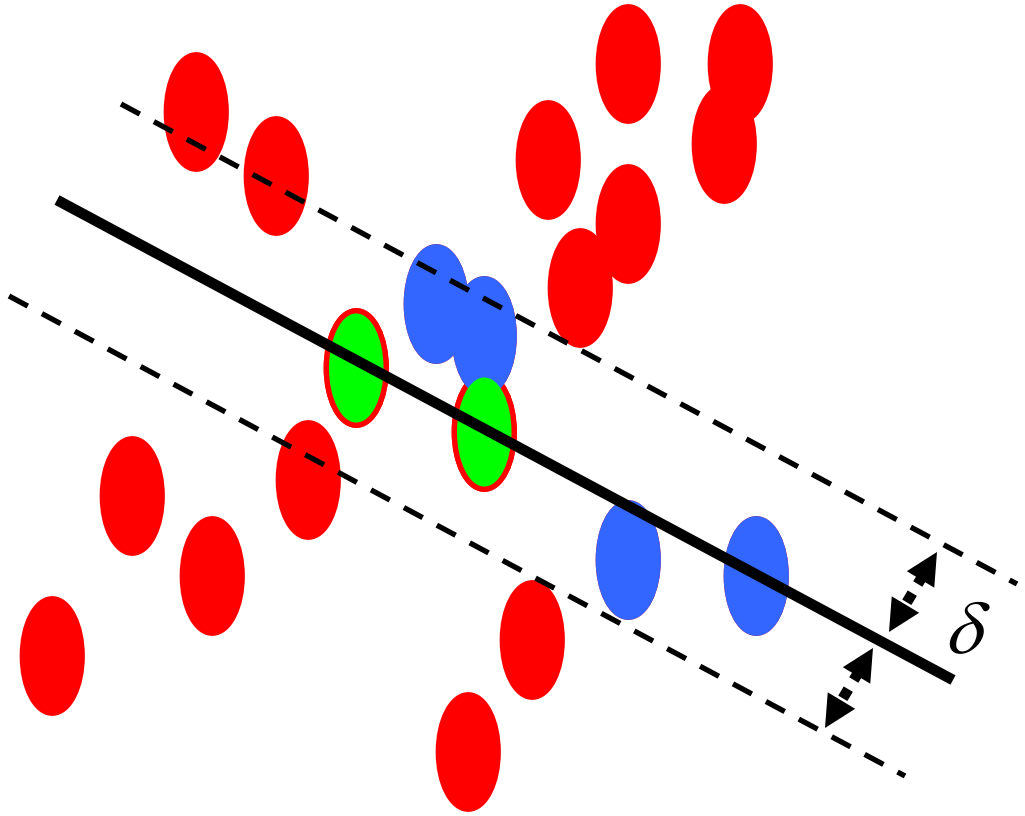
Algorithm:

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

Line fitting example



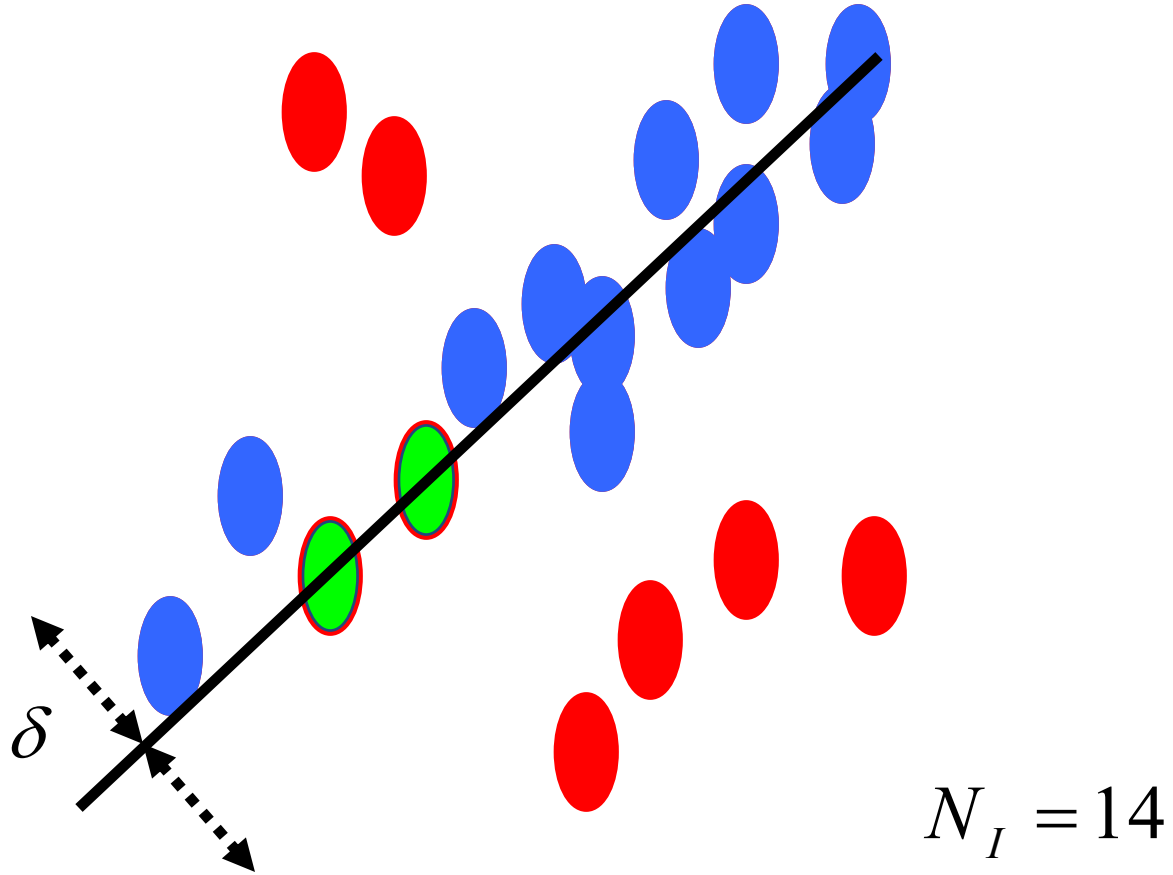
$$N_I = 6$$

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



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