(More) Algorithms for Cameras: Edge Detection Modeling Cameras/Objects

Connelly Barnes

Acknowledgment: Many slides from James Hays, also Derek Hoiem Grauman&Leibe 2008

Outline

- **Edge Detection: Canny, etc.**
- Modeling cameras/objects:
  - Model Fitting: Hough Transform and RANSAC
  - Modeling Multiple Cameras
  - Optical Flow

Canny edge detector

- This is probably the most widely used edge detector in computer vision
- Theoretical model: step-edges corrupted by additive Gaussian noise
- Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of *signal-to-noise ratio* and localization

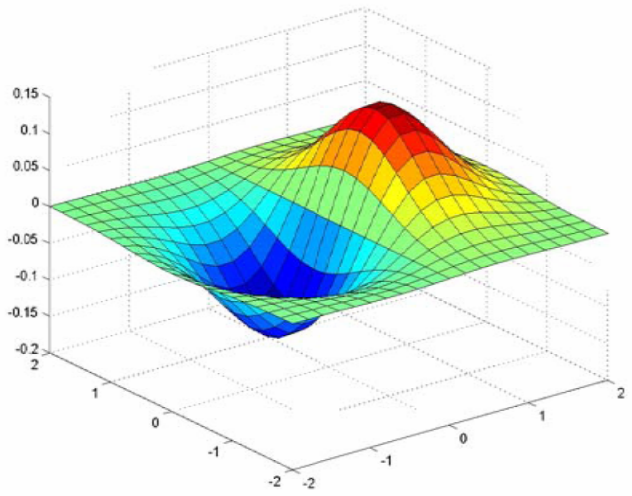
J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Source: L. Fei-Fei

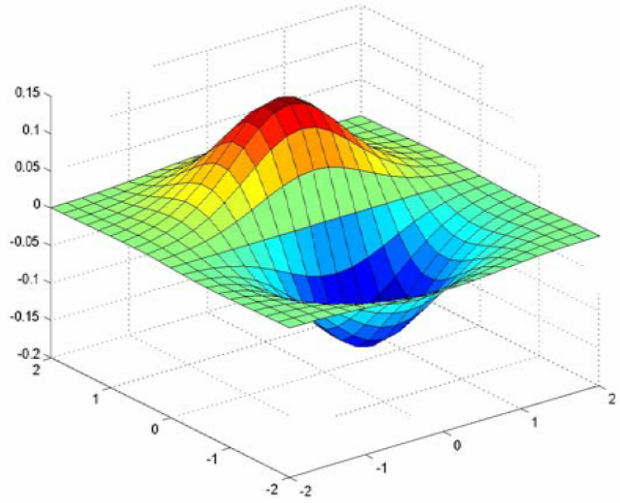
Example

original image (Lena)

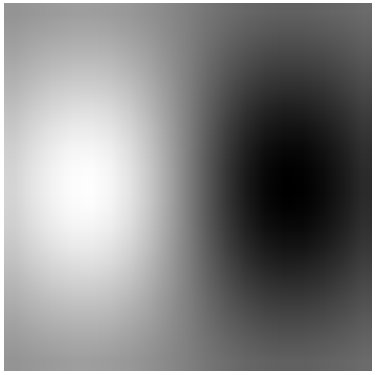
Derivative of Gaussian filter

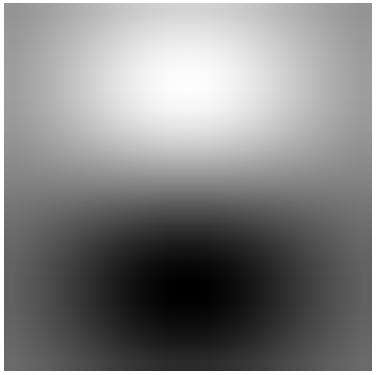


x-direction

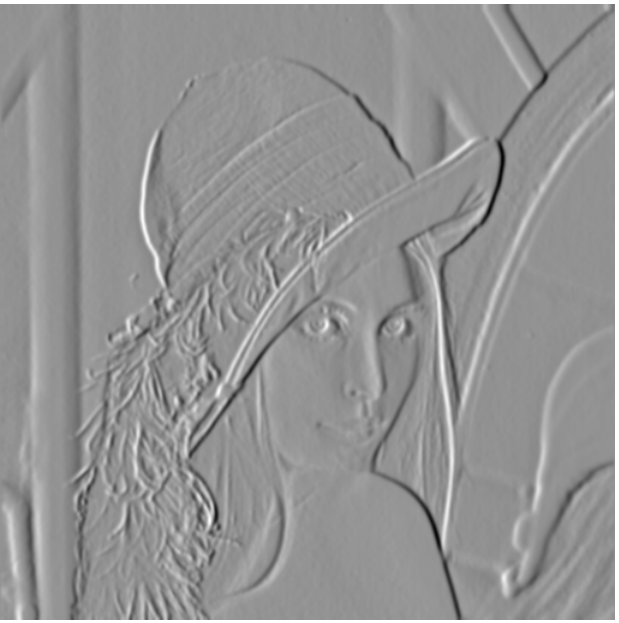


y-direction





Compute Gradients (DoG)



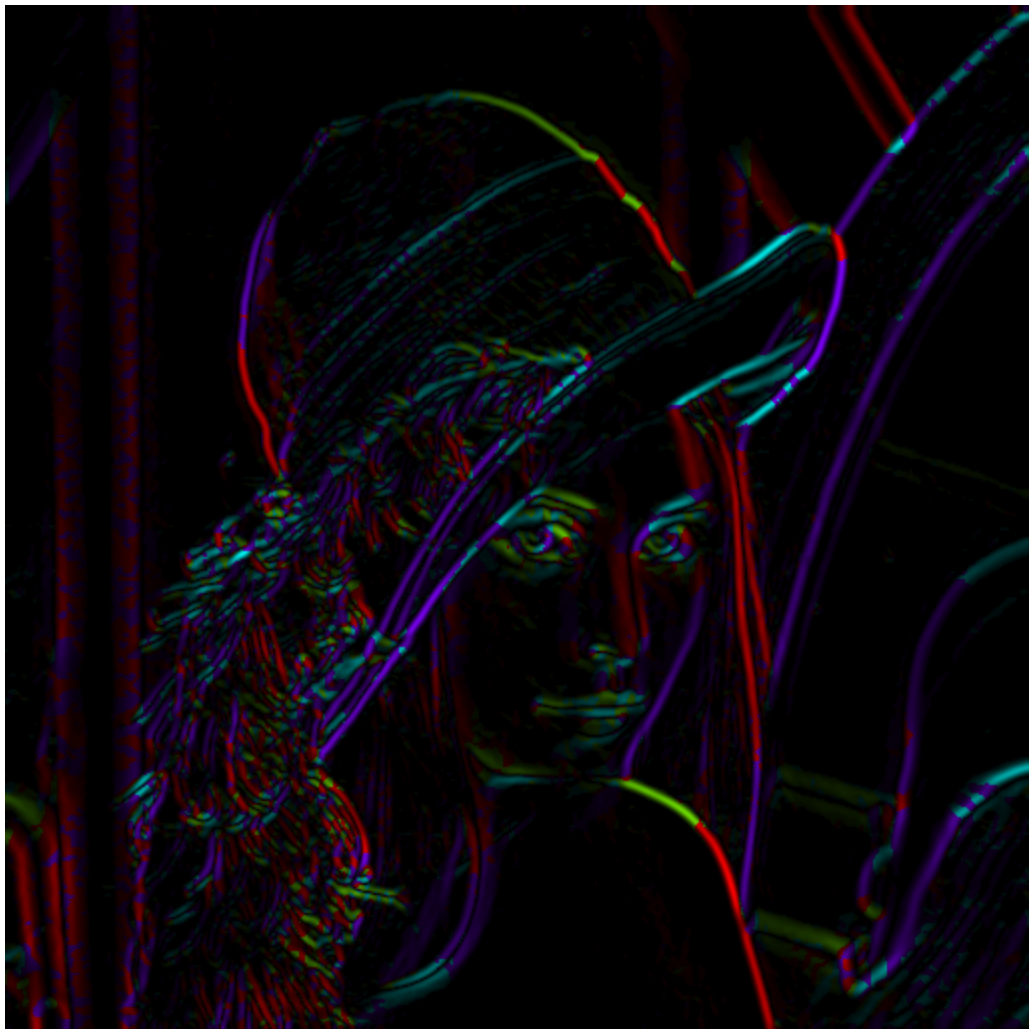
X-Derivative of Gaussian

Y-Derivative of Gaussian

Gradient Magnitude

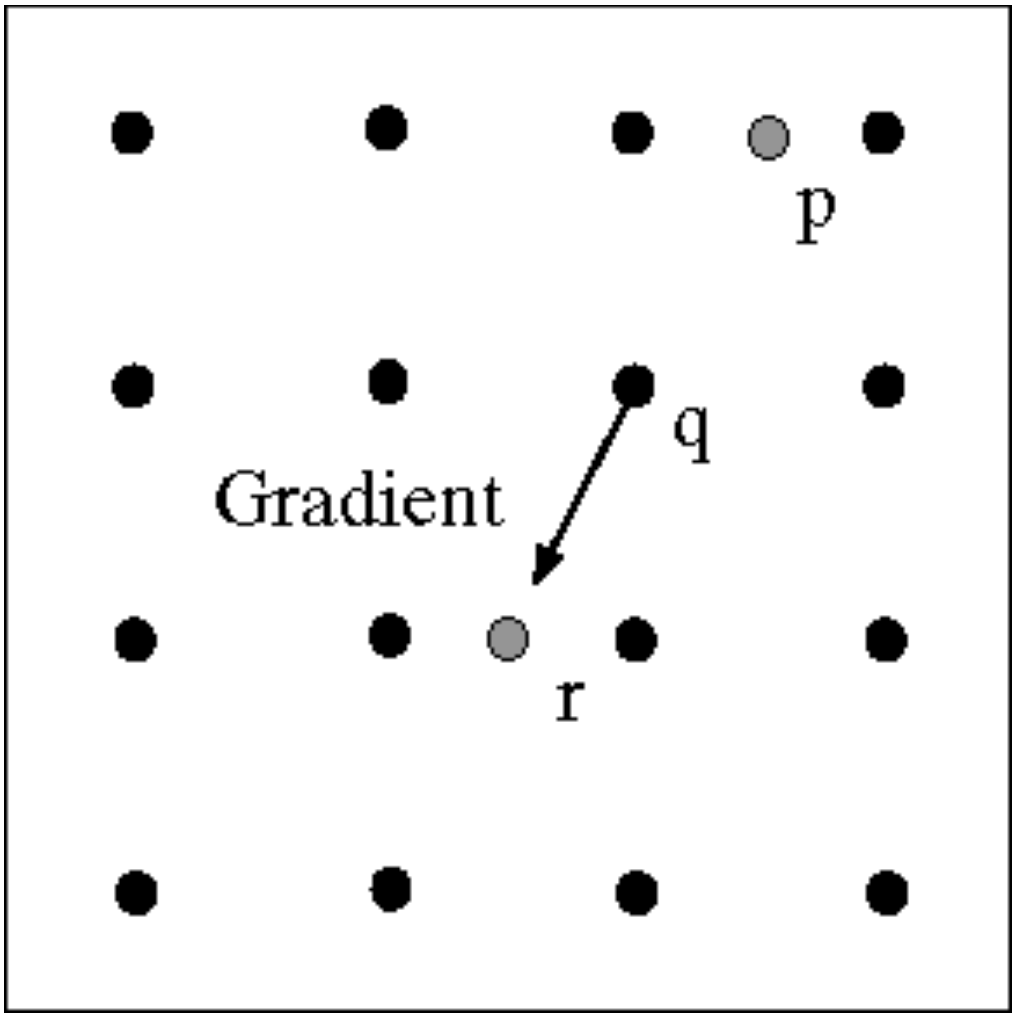
Get Orientation at Each Pixel

- Threshold at minimum level
- Get orientation

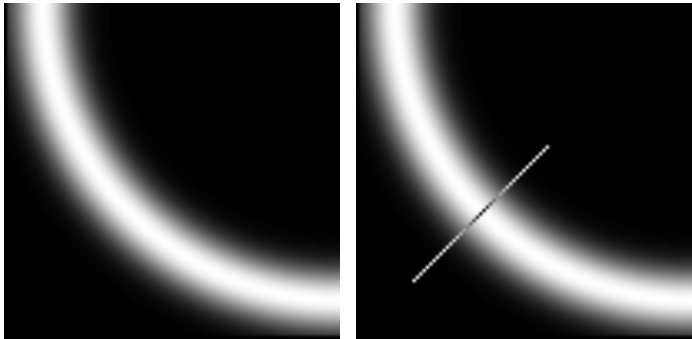


theta = atan2(gy, gx)

Non-maximum suppression for each orientation (“thinning”)



At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.



Source: D. Forsyth

Before Non-max Suppression

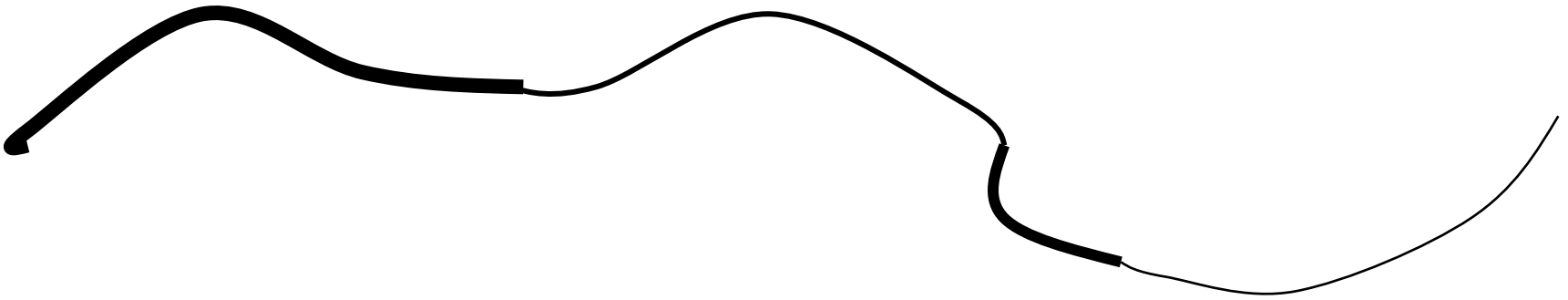
After non-max suppression

Hysteresis thresholding

- Threshold at low/high levels to get weak/strong edge pixels
- Do connected components, starting from strong edge pixels

Hysteresis thresholding

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.



Source: S. Seitz

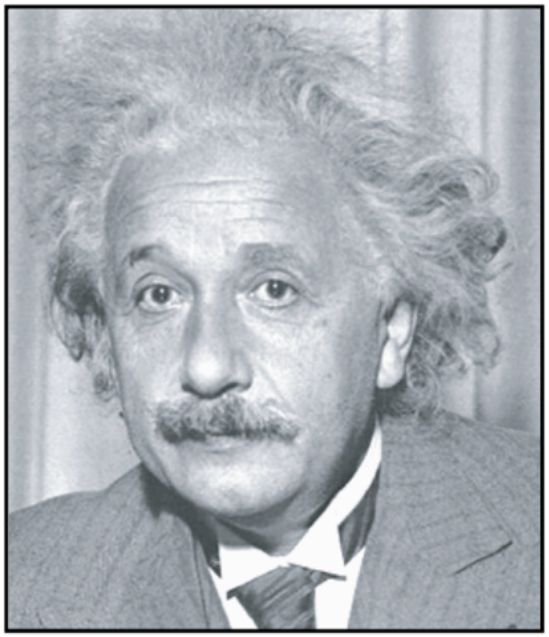
Final Canny Edges

Implementations

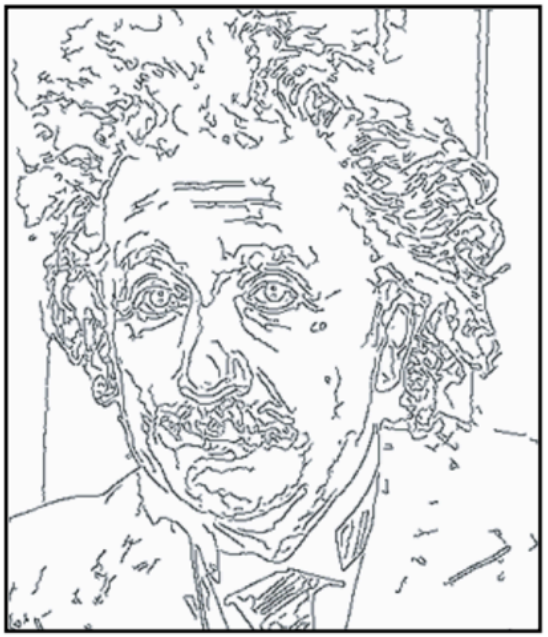
- MATLAB: edge(im, ‘canny’)
- Python: skimage.filter.canny()
- C++: OpenCV canny()

Smoother Edges

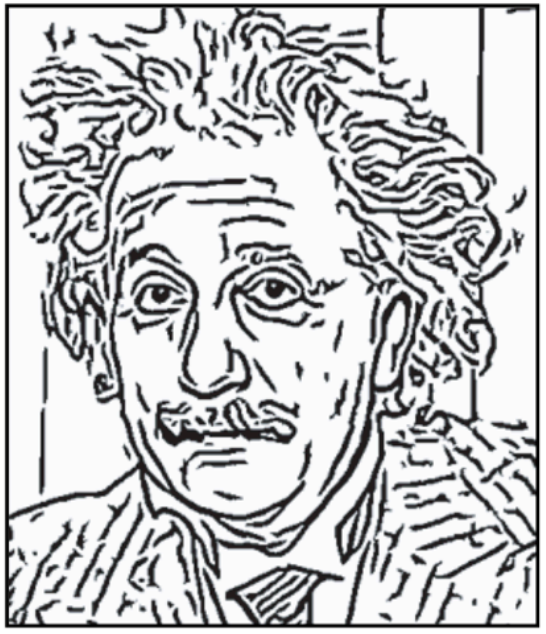
- Canny faithfully tracks along noisy edges
- Kang et al. 2007 “Coherent Line Drawing”:
- Smoother edges by blurring vector field along the edge direction before line extraction.



(a) Input



(b) Canny



Kang et al. 2007

Outline

- Edge Detection: Canny, etc.
- **Basics of modeling cameras/objects:**
  - **Model Fitting: Hough Transform and RANSAC**
  - Modeling Multiple Cameras
  - Optical Flow

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

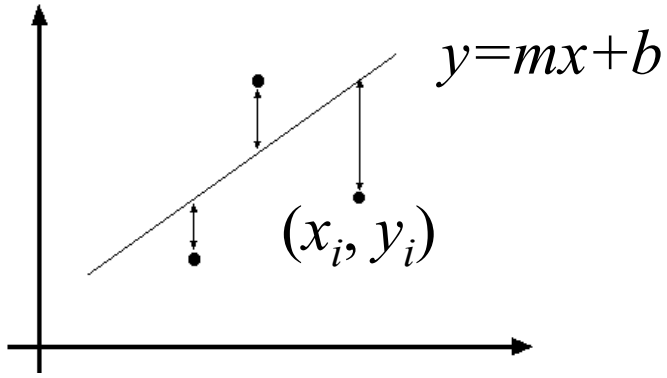
- Global optimization / Search for parameters
  - Least squares fit
  - Robust least squares
  - Iterative closest point (ICP)

- Hypothesize and test
  - Generalized Hough transform
  - RANSAC

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



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

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

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

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

Matlab: `p = A \ y;`

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

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.

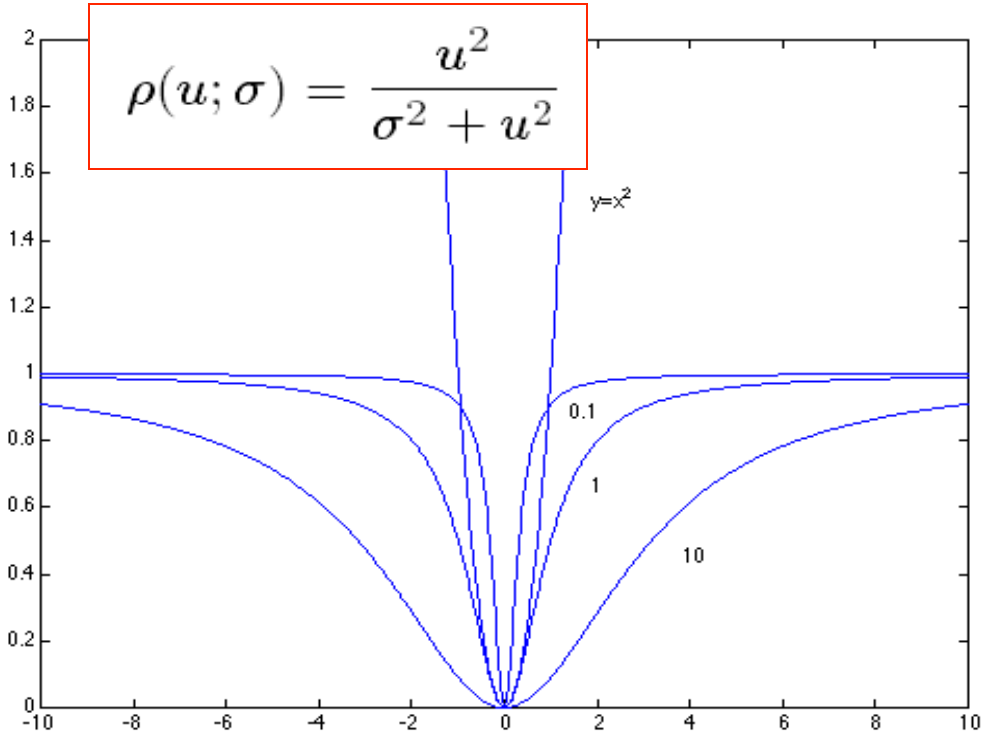
Robust least squares (to deal with outliers)

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

Slide from S. Savarese

Robust Estimator

1. Initialize: e.g., choose θ by least squares fit and $\sigma = 1.5 \cdot \text{median}(error)$
2. Choose params to minimize: $\sum_i \frac{error(\theta, data_i)^2}{\sigma^2 + error(\theta, data_i)^2}$
   - E.g., numerical optimization
3. Compute new $\sigma = 1.5 \cdot \text{median}(error)$
4. Repeat (2) and (3) until convergence

Hough Transform: Outline

1. Create a grid of parameter values

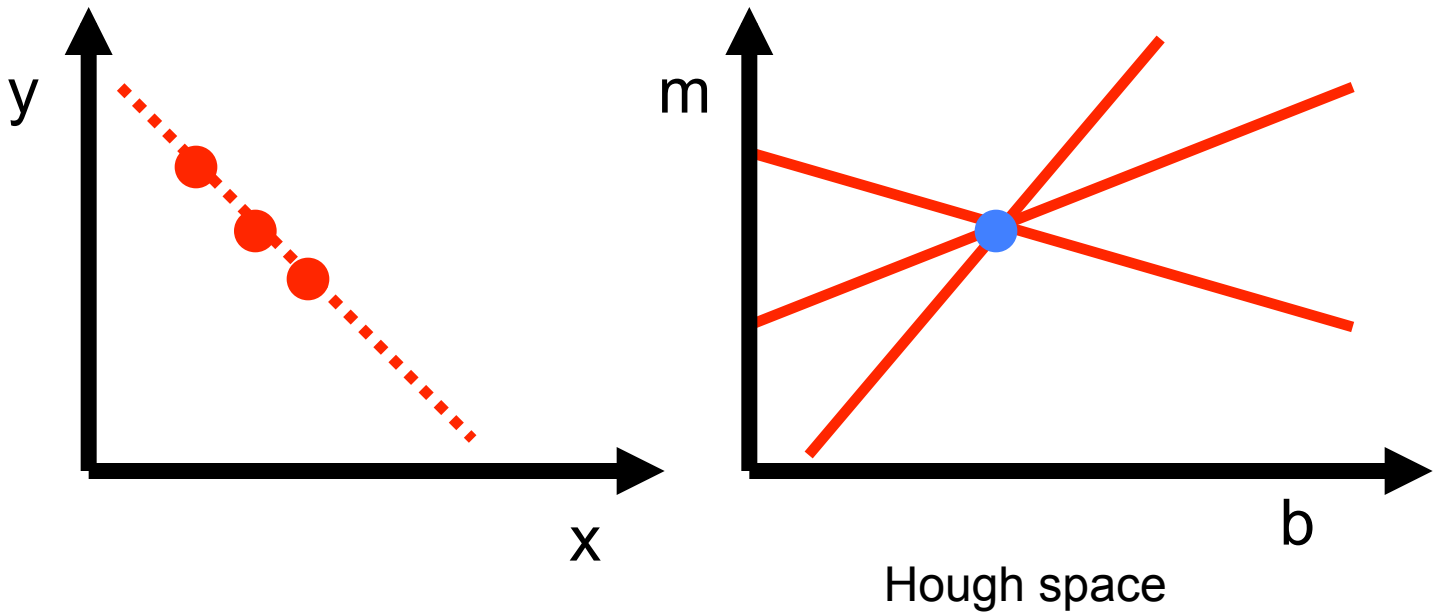
2. Each point votes for a set of parameters, incrementing those values in grid

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

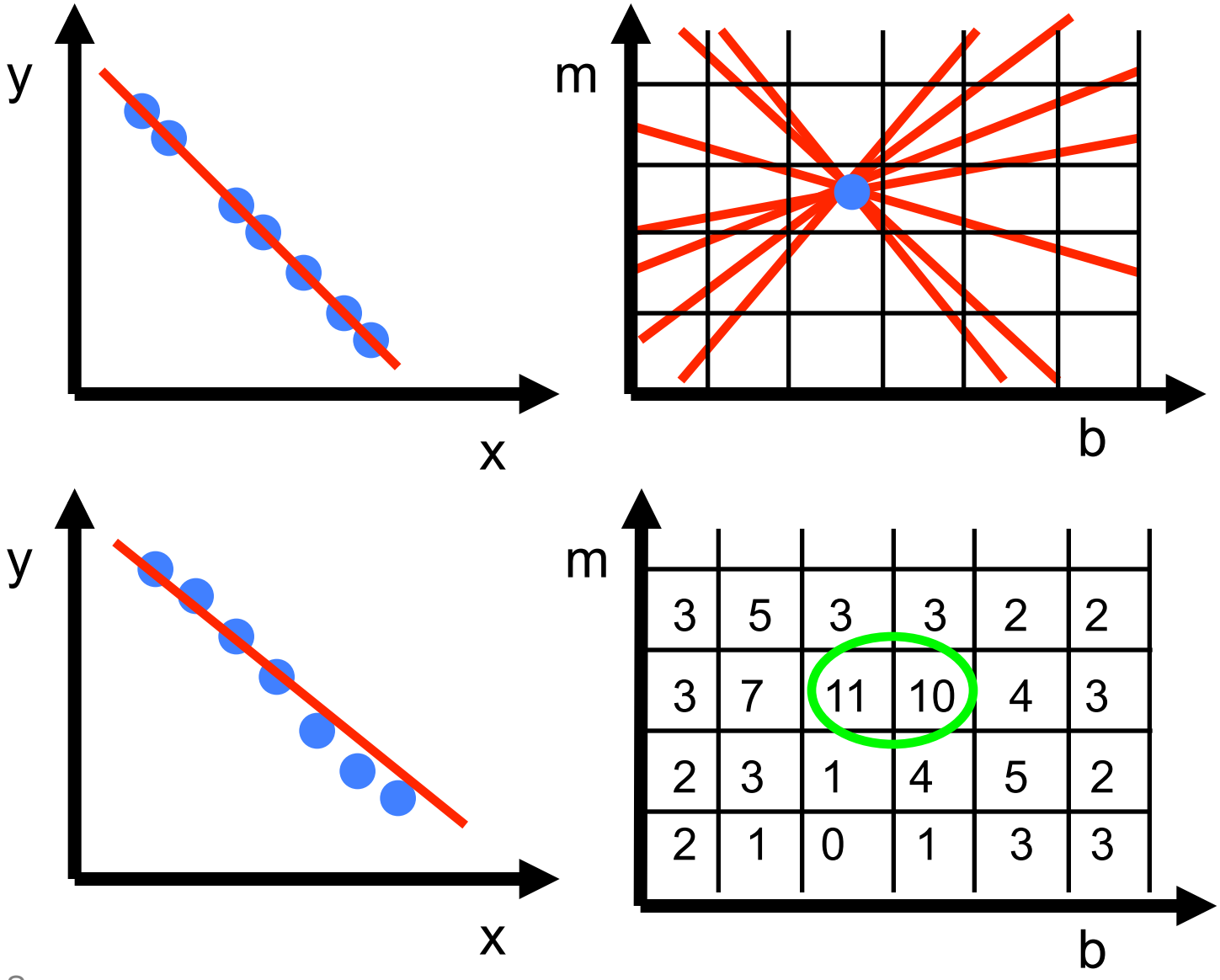
Given a set of points, find the curve or line that explains the data points best



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

Slide from S. Savarese

Hough transform



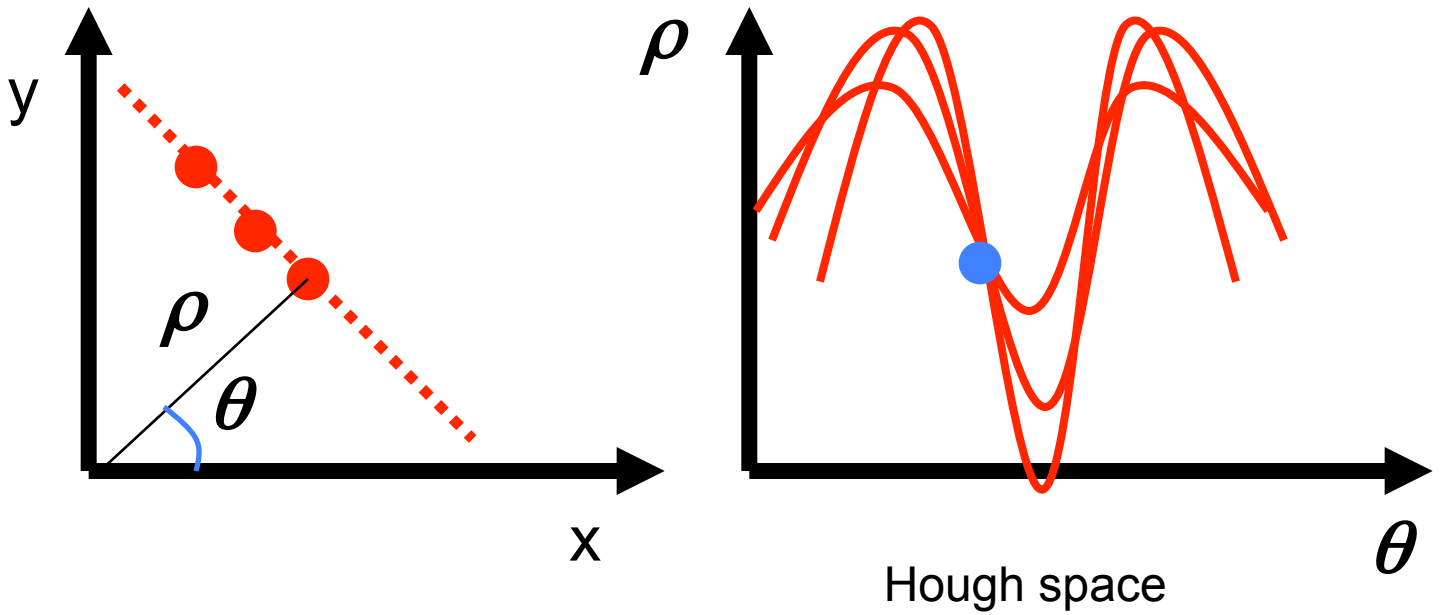
Slide from S. Savarese

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…

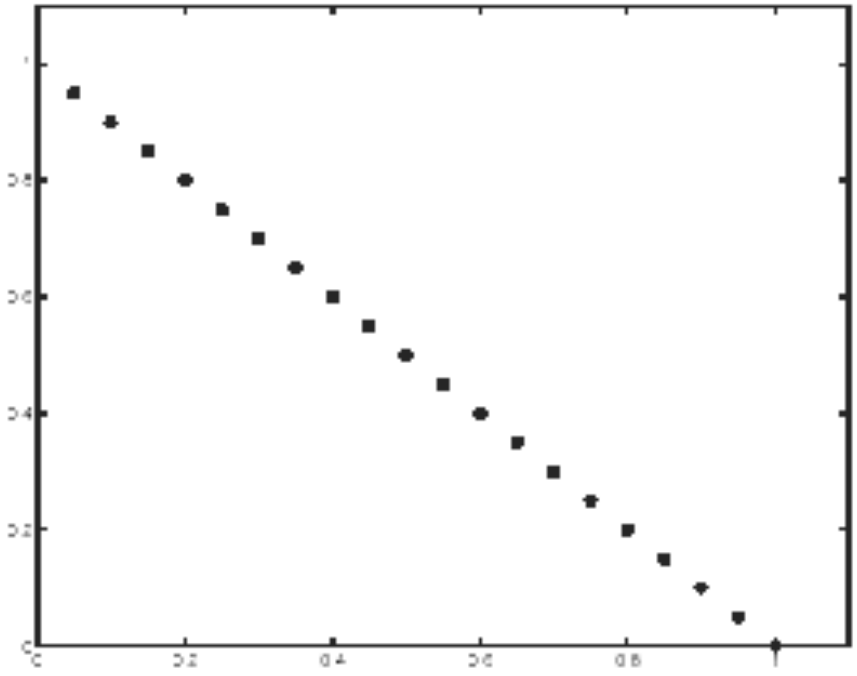
Use a polar representation for the parameter space



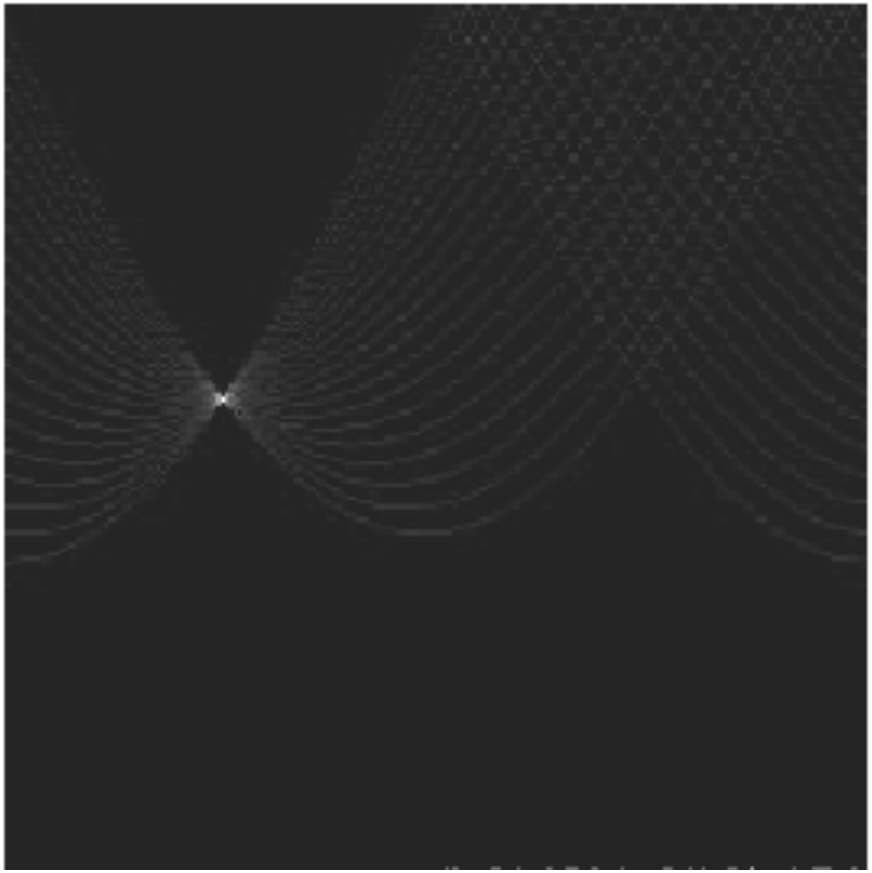
$$x\cos\theta + y\sin\theta = \rho$$

Slide from S. Savarese

Hough transform - experiments



features

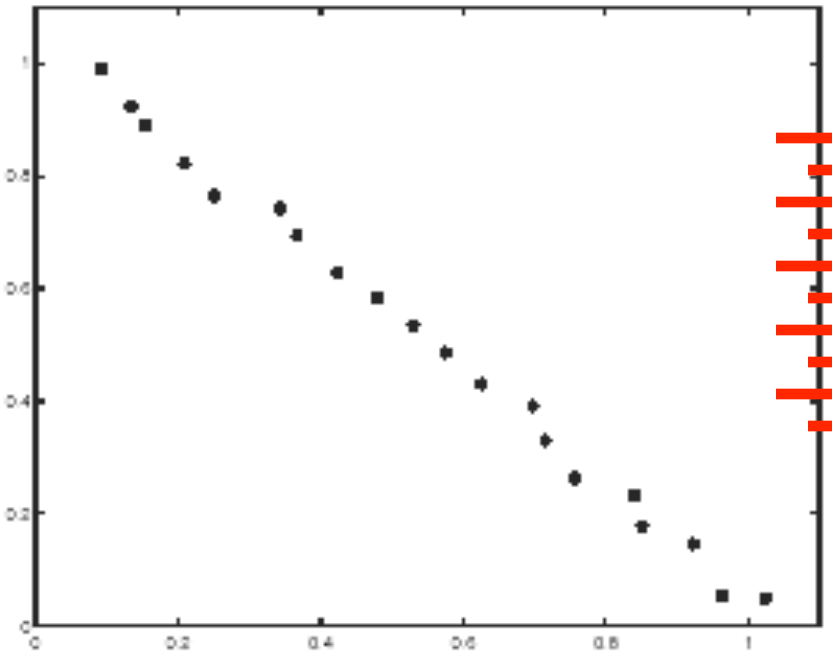


votes

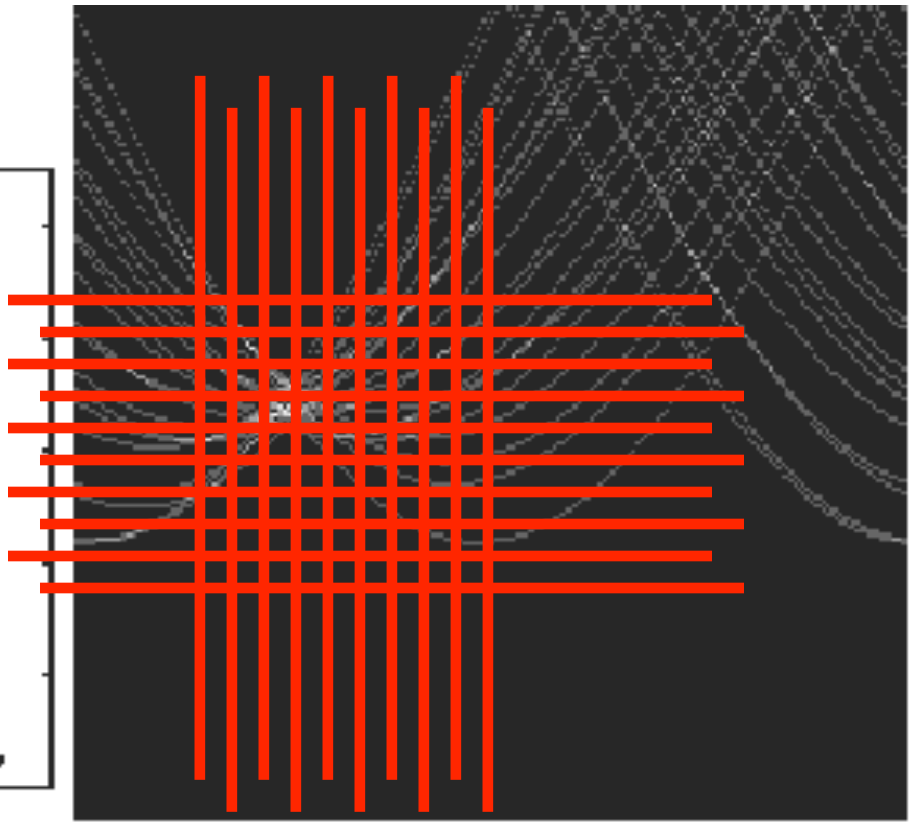
Slide from S. Savarese

Hough transform - experiments

Noisy data



features

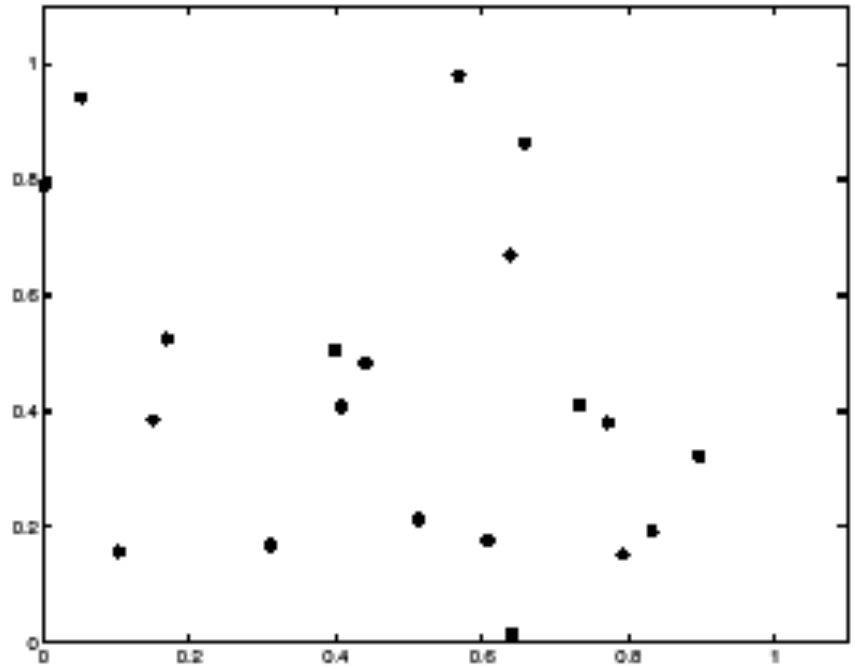


votes

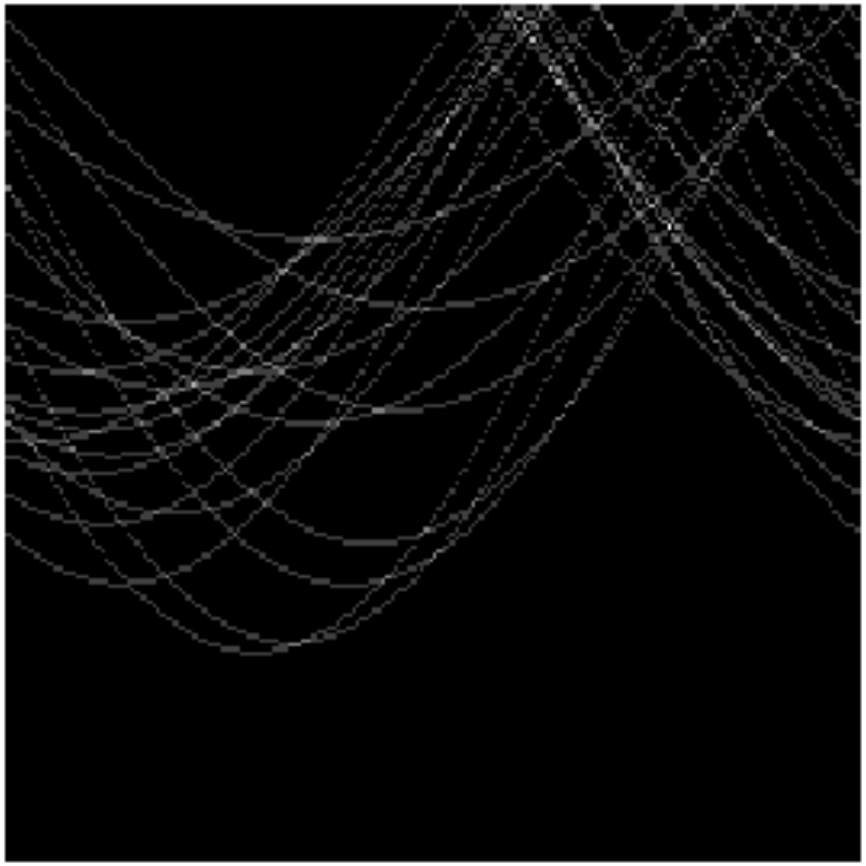
Need to adjust grid size or smooth

Slide from S. Savarese

Hough transform - experiments



features



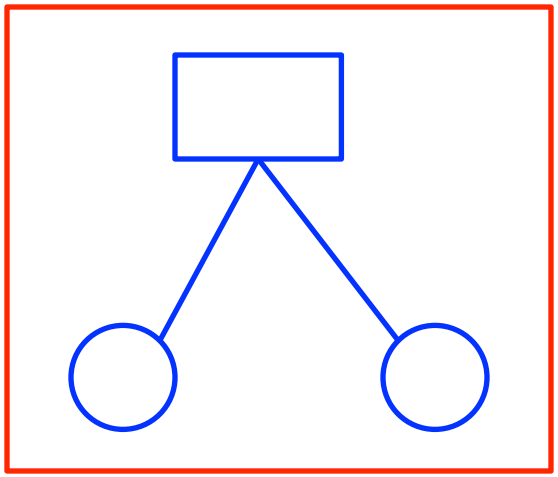
votes

Issue: spurious peaks due to uniform noise

Slide from S. Savarese

Hough Transform

- How would we find circles?
  - Of fixed radius
  - Of unknown radius
- How would we detect an object with several parts?



Hough transform conclusions

Good

- Robust to outliers: each point votes separately
- Fairly efficient (much 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

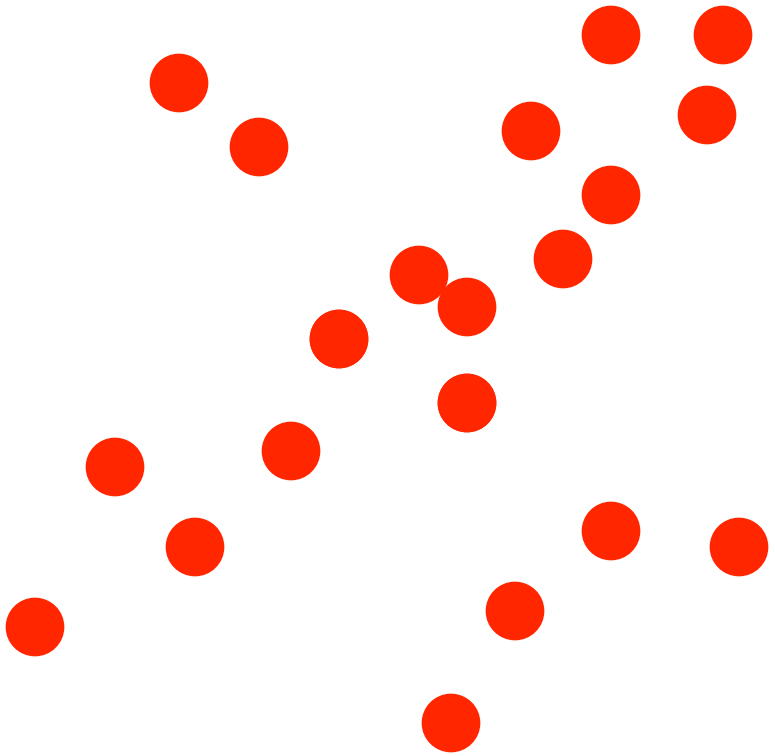
Common applications

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

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in ‘81.



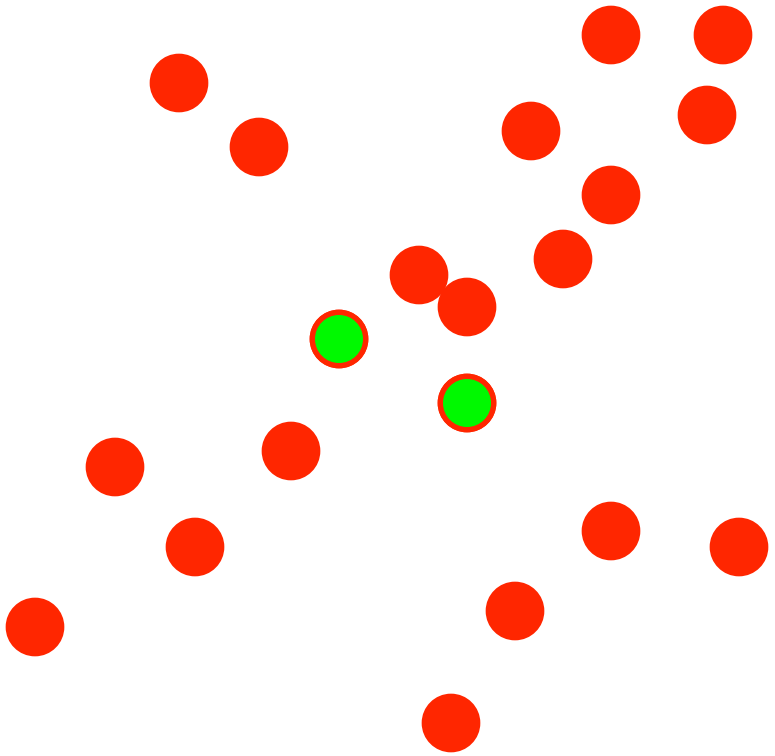
Algorithm:

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:

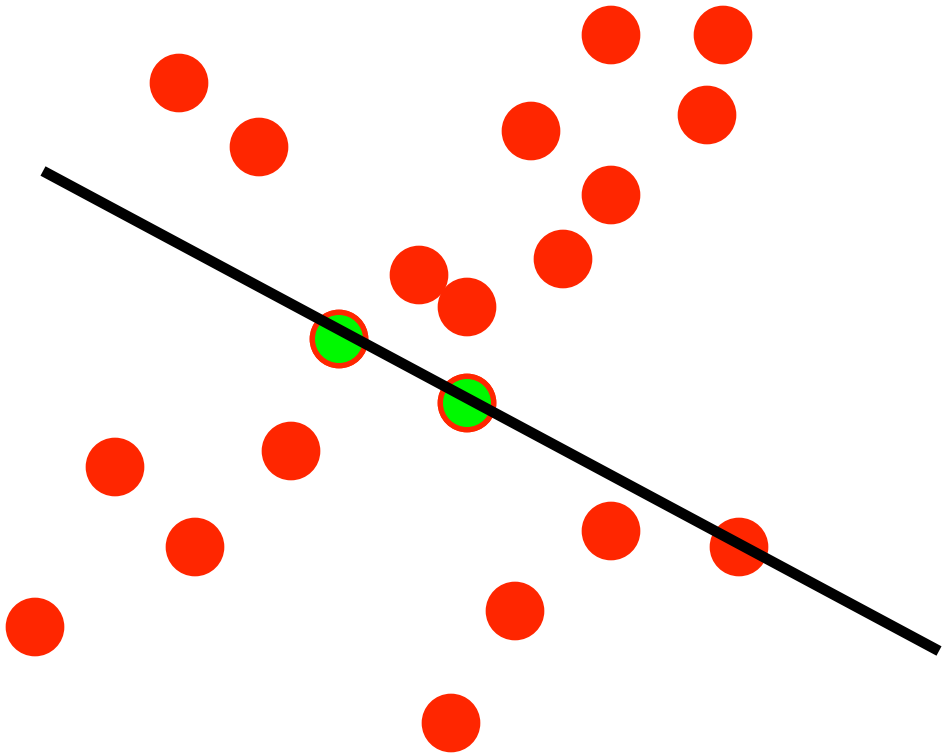
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

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

Illustration by Savarese

RANSAC

Line fitting example



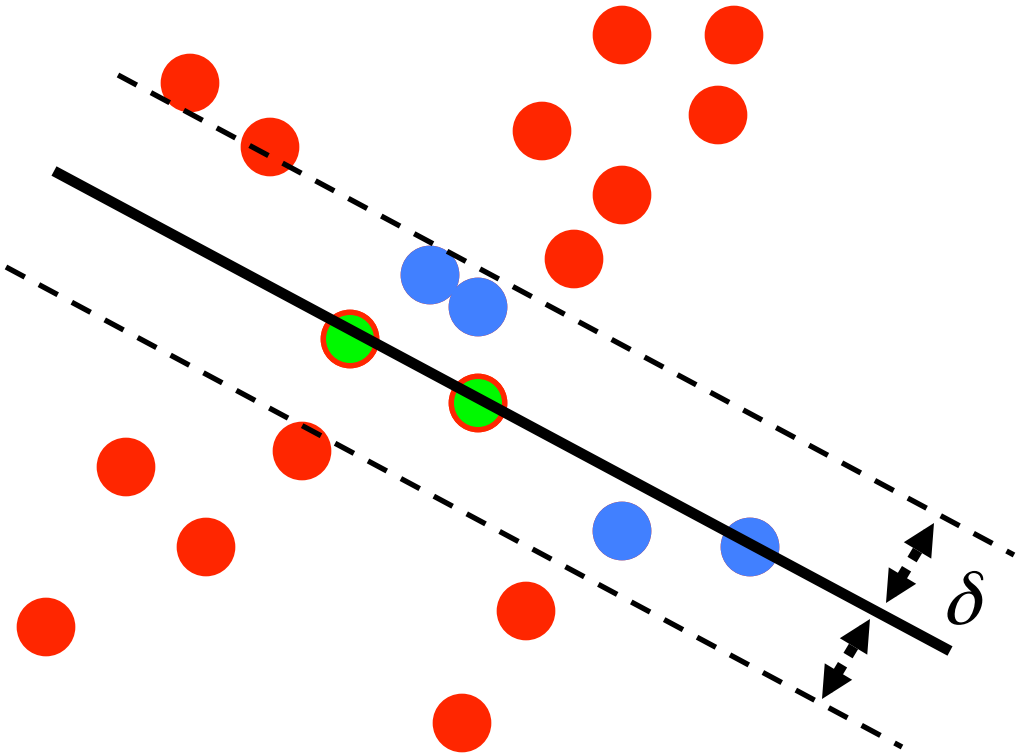
Algorithm:

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



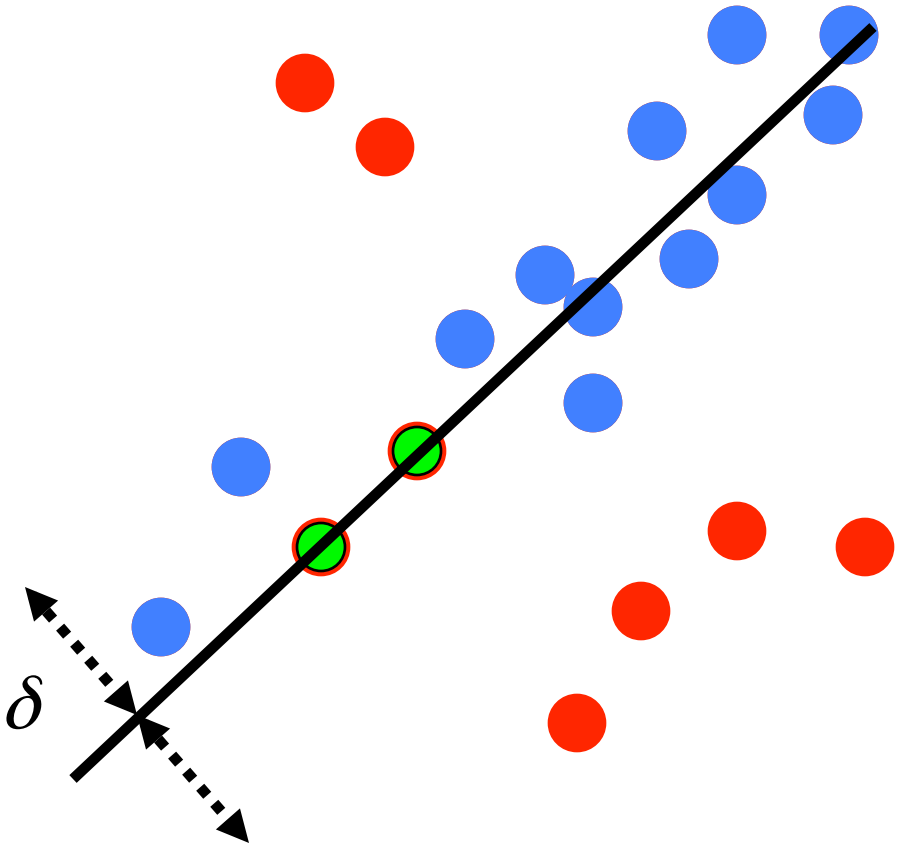
$$N_I = 6$$

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



$N_I = 14$

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

How to choose parameters?

- Number of samples *N*
  - Choose *N* so that, with probability *p*, at least one random sample is free from outliers (e.g. *p*=0.99) (outlier ratio: *e*)
- Number of sampled points *s*
  - Minimum number needed to fit the model
- Distance threshold δ
  - Choose δ so that a good point with noise is likely (e.g., prob=0.95) within threshold
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$

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

	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

modified from M. Pollefeys

RANSAC conclusions

Good

- Robust to outliers
- Applicable for larger number of objective function parameters than Hough transform
- Optimization parameters are easier to choose than Hough transform

Bad

- Computational time grows quickly with fraction of outliers and number of parameters
- Not good for getting multiple fits

Common applications

- Computing a homography (e.g., image stitching)
- Estimating fundamental matrix (relating two views)

Outline

- Edge Detection: Canny, etc.
- **Basics of modeling cameras/objects:**
  - Model Fitting: Hough Transform and RANSAC
  - **Modeling Multiple Cameras**
  - Optical Flow

Homography

- In classic games, e.g. Mario Kart, the ground is just a texture mapped plane:

  Mario Kart (YouTube)

- Any two images *a* and *b* of a planar surface are related by a homography.
- In homogeneous coordinates:

$$p_a = \begin{bmatrix} x_a \\ y_a \\ 1 \end{bmatrix}, p'_b = \begin{bmatrix} w' x_b \\ w' y_b \\ w' \end{bmatrix}, \mathbf{H}_{ab} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

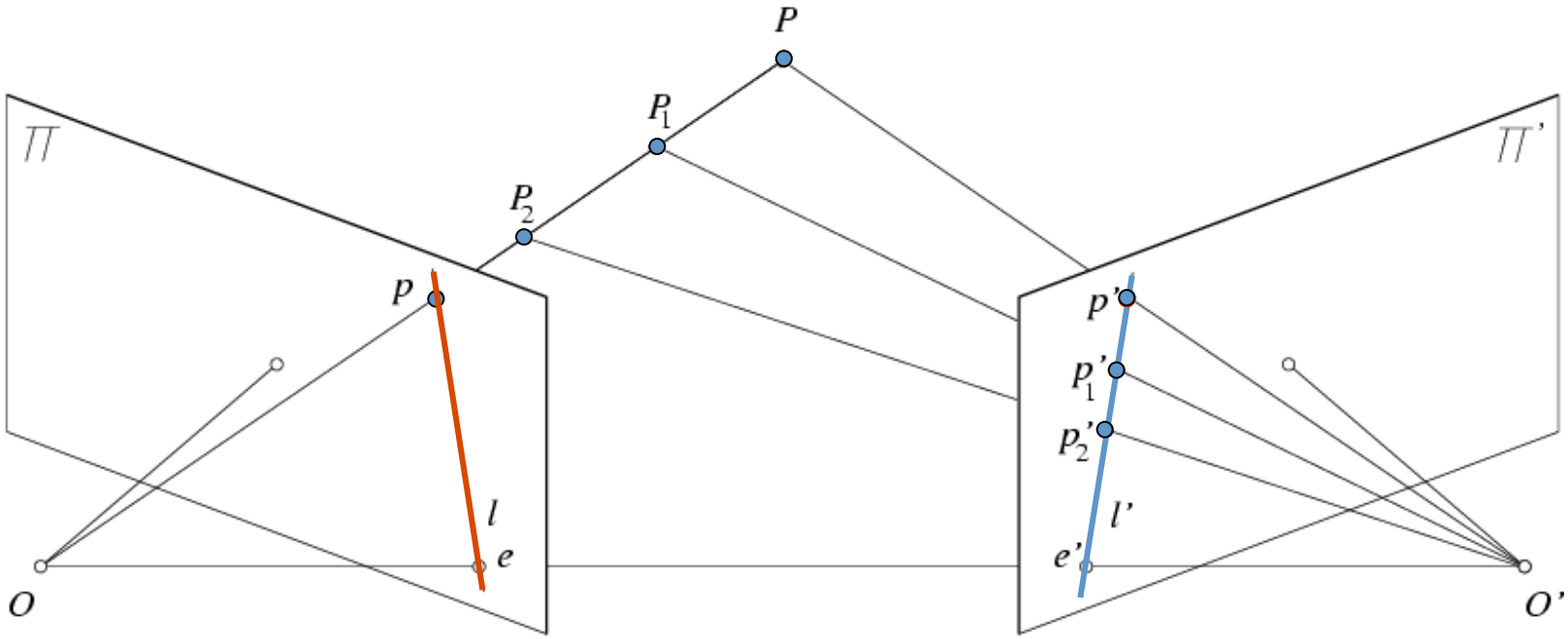
Homography

- Discussion questions:
- If we had two images of a planar scene, how would we find the homography between them?
- How many corresponding points would we need to fit the homography?

Point p_a in image *a*, point p'_b in image *b*

$$p_a = \begin{bmatrix} x_a \\ y_a \\ 1 \end{bmatrix}, p'_b = \begin{bmatrix} w'x_b \\ w'y_b \\ w' \end{bmatrix}, \mathbf{H}_{ab} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

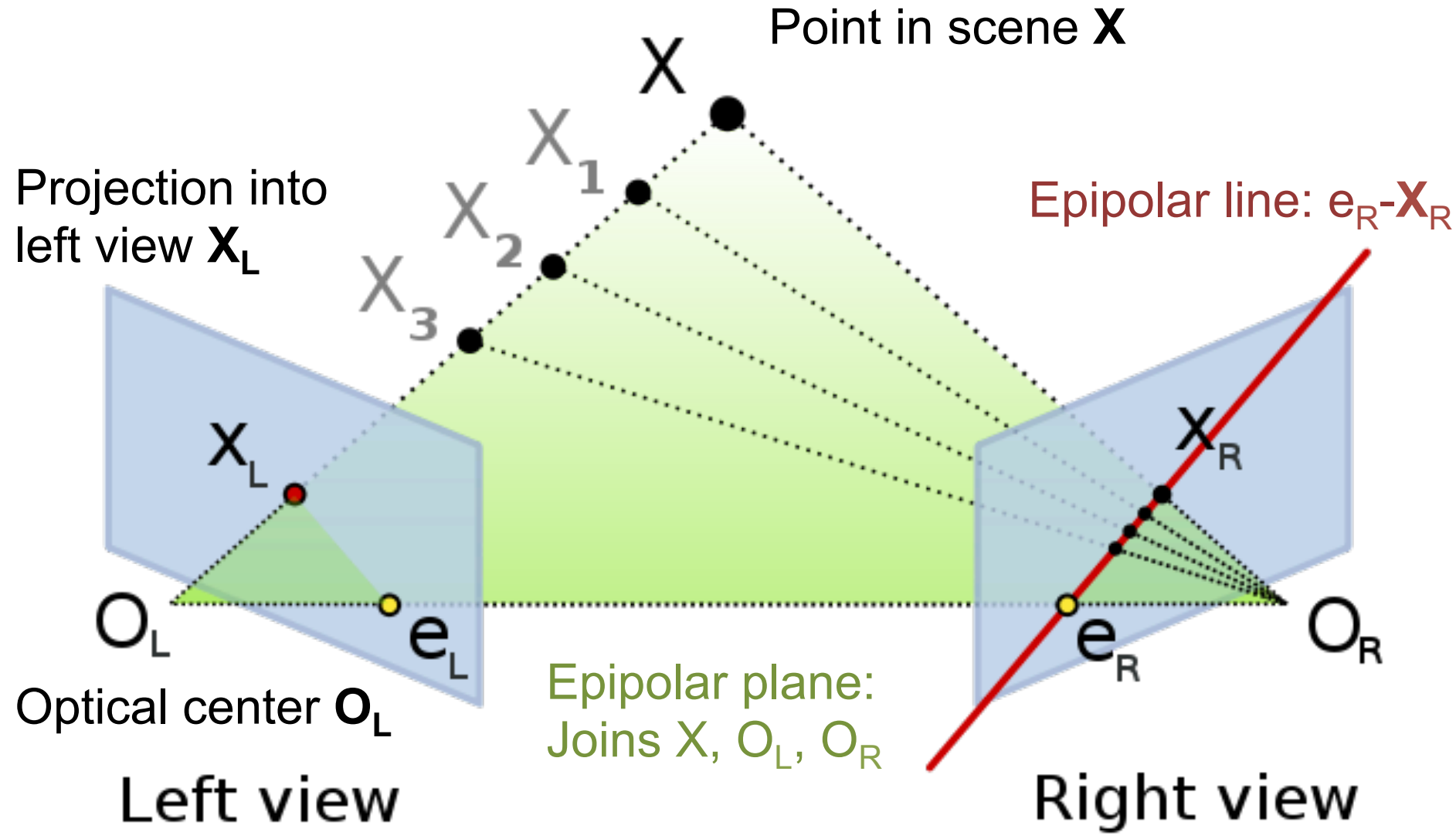
Epipolar constraint



Geometry of two views constrains where the corresponding pixel for some image point in the first view must occur in the second view.

- It must be on the line carved out by a plane connecting the world point and optical centers.

Epipolar Geometry



Epipolar geometry: terms

- **Baseline**: line joining the camera centers
- **Epipole**: point of intersection of baseline with image plane
- **Epipolar plane**: plane containing baseline and world point
- **Epipolar line**: intersection of epipolar plane with the image plane

- All epipolar lines intersect at the epipole
- An epipolar plane intersects the left and right image planes in epipolar lines

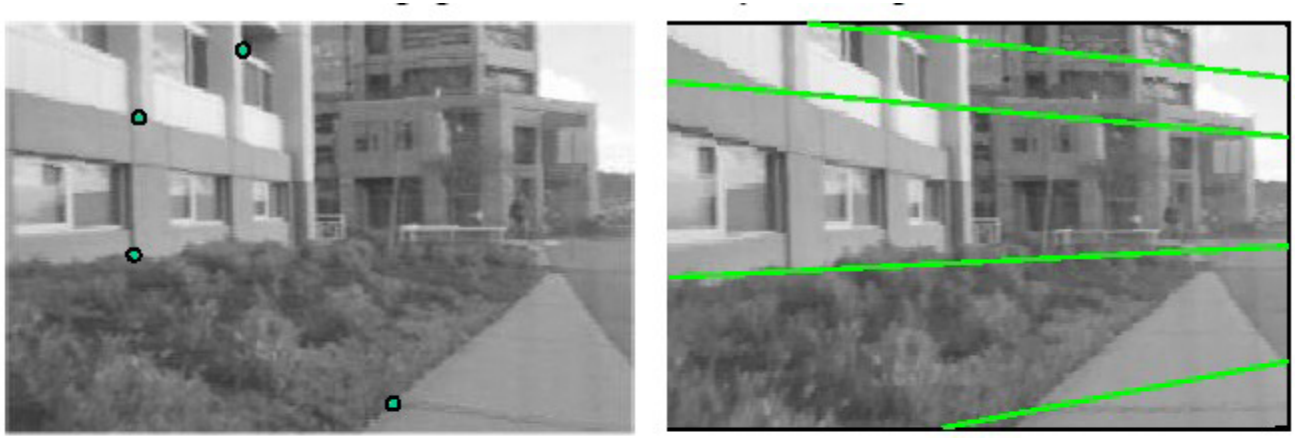
Why is the epipolar constraint useful?

Epipolar constraint

This is useful because it reduces the correspondence problem to a 1D search along an epipolar line.

Image from Andrew Zisserman

Example



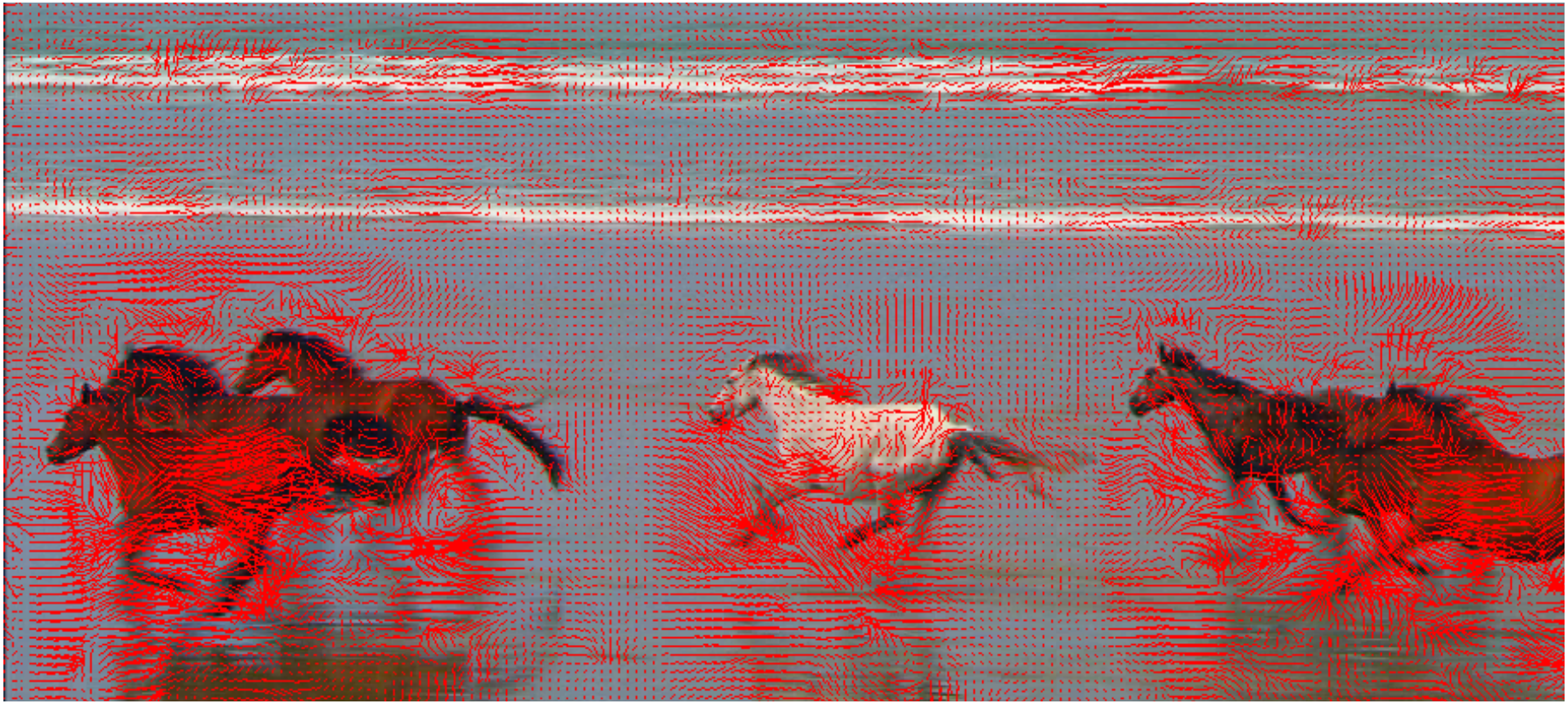
Can often **rectify** a pair of images so the epipolar lines become horizontal.

Outline

- Edge Detection: Canny, etc.
- **Basics of modeling cameras/objects:**
  - Model Fitting: Hough Transform and RANSAC
  - Modeling Multiple Cameras
  - **Optical Flow**

Optical Flow

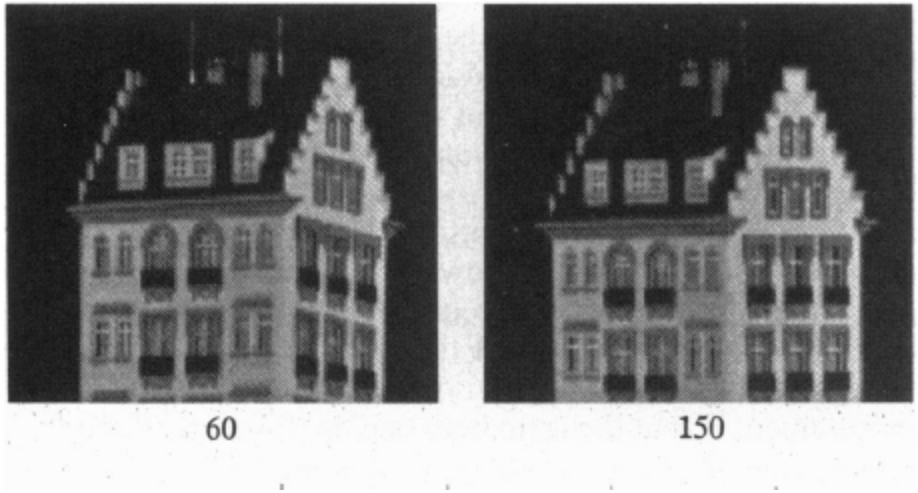
- Problem: given a video, how can we find the correspondences between pixels in subsequent frames?

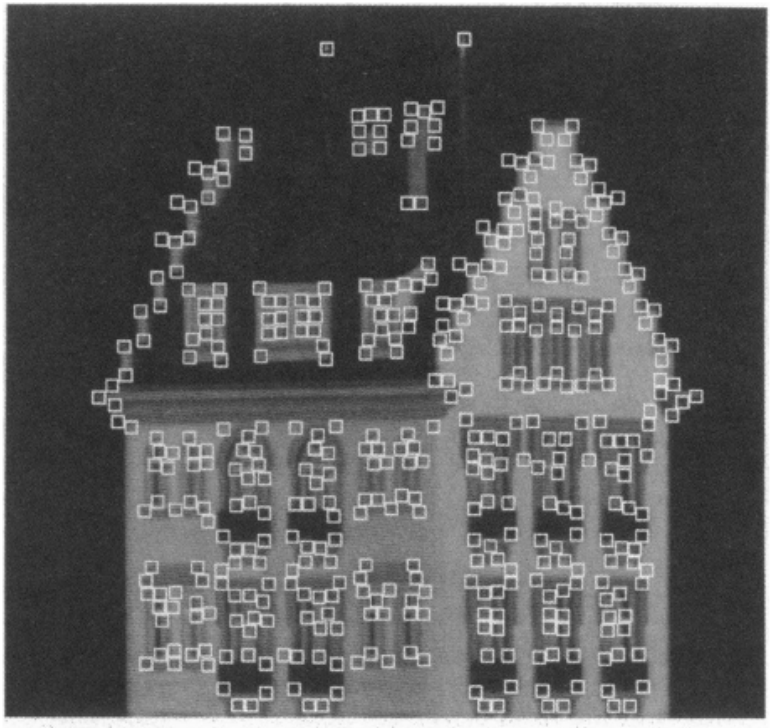


Source: Paul Sastrasinh, Brown University CS 1290

Feature tracking

- Many problems, such as structure from motion require matching points
- If motion is small, tracking is an easy way to get them

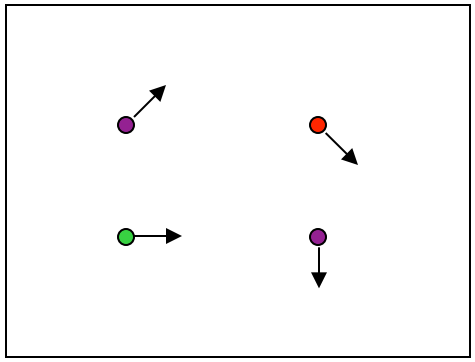




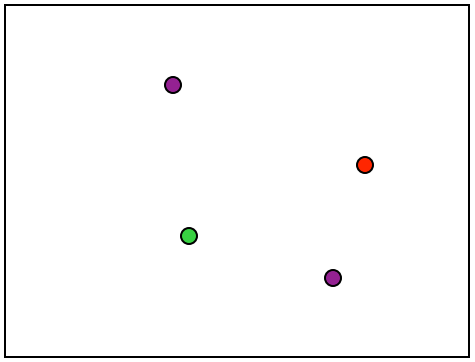
Feature tracking

- Challenges
  - Figure out which features can be tracked
  - Efficiently track across frames
  - Some points may change appearance over time (e.g., due to rotation, moving into shadows, etc.)
  - Drift: small errors can accumulate as appearance model is updated
  - Points may appear or disappear: need to be able to add/delete tracked points

Feature tracking



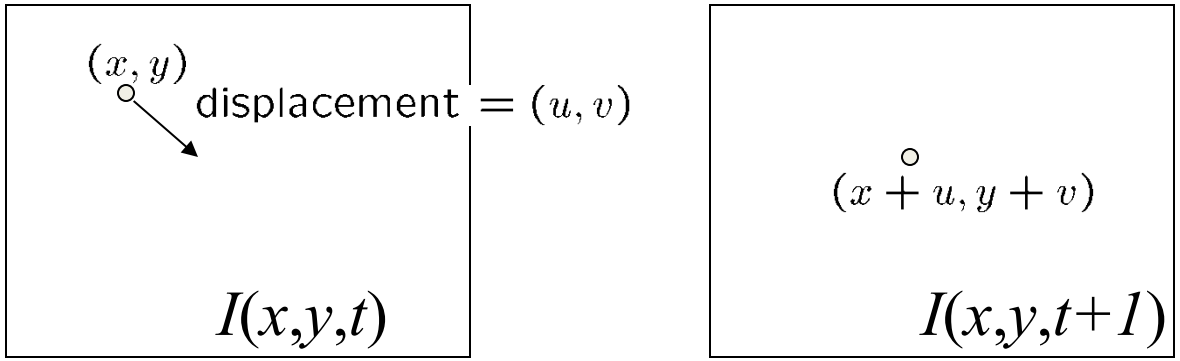
$I(x,y,t)$



$I(x,y,t+1)$

- Given two subsequent frames, estimate the point translation
- Key assumptions of Lucas-Kanade Tracker
  - **Brightness constancy:** projection of the same point looks the same in every frame
  - **Small motion:** points do not move very far
  - **Spatial coherence:** points move like their neighbors

The brightness constancy constraint



- Brightness Constancy Equation:

$$I(x,y,t) = I(x+u, y+v, t+1)$$

Take Taylor expansion of $I(x+u, y+v, t+1)$ at (x,y,t) to linearize the right side:

Image derivative along x Derivative along t

$$I(x+u,y+v,t+1) \approx I(x,y,t) + I_x \cdot u + I_y \cdot v + I_t$$

$$I(x+u,y+v,t+1) - I(x,y,t) = +I_x \cdot u + I_y \cdot v + I_t$$

Hence, $I_x \cdot u + I_y \cdot v + I_t \approx 0 \quad \rightarrow \nabla I \cdot \left[u \;\; v\right]^T + I_t = 0$

The brightness constancy constraint

Can we use this equation to recover image motion (u,v) at each pixel?

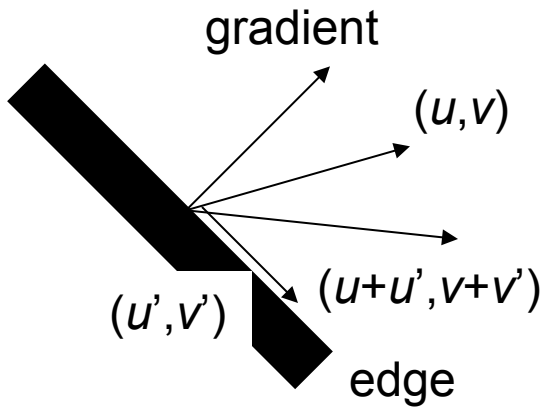
$$\nabla I \cdot \begin{bmatrix} u & v \end{bmatrix}^T + I_t = 0$$

- How many equations and unknowns per pixel?
  - One equation (this is a scalar equation!), two unknowns (u,v)

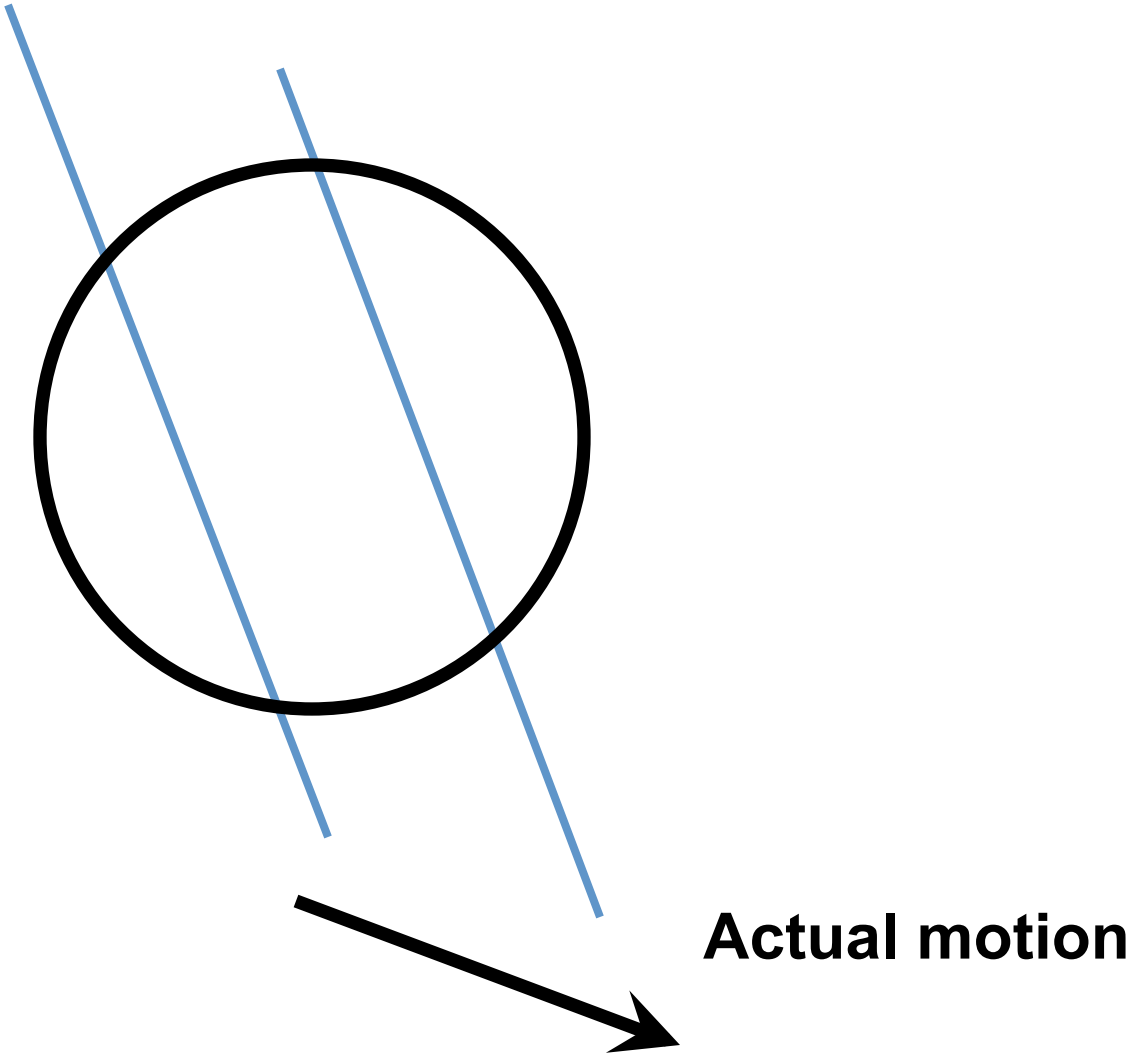
The component of the motion perpendicular to the gradient (i.e., parallel to the edge) cannot be measured

If (u, v) satisfies the equation, so does (u+u', v+v') if

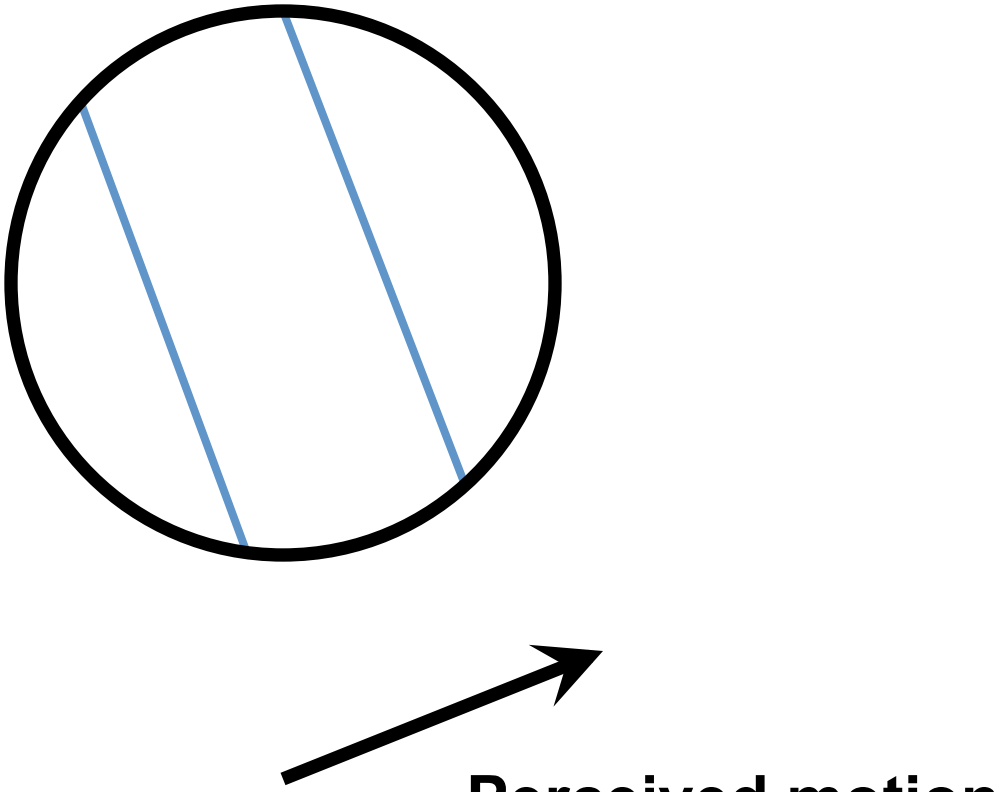
$$\nabla I \cdot \begin{bmatrix} u' & v' \end{bmatrix}^T = 0$$



The aperture problem



The aperture problem



The barber pole illusion

http://en.wikipedia.org/wiki/Barberpole_illusion

The barber pole illusion

http://en.wikipedia.org/wiki/Barberpole_illusion

Solving the ambiguity…

B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

- How to get more equations for a pixel?
- **Spatial coherence constraint**
- Assume the pixel's neighbors have the same (u,v)
  - If we use a 5x5 window, that gives us 25 equations per pixel

$$0 = I_t(\mathbf{p_i}) + \nabla I(\mathbf{p_i}) \cdot [u\ v]$$

$$\begin{bmatrix} I_x(\mathbf{p_1}) & I_y(\mathbf{p_1}) \\ I_x(\mathbf{p_2}) & I_y(\mathbf{p_2}) \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}}) & I_y(\mathbf{p_{25}}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p_1}) \\ I_t(\mathbf{p_2}) \\ \vdots \\ I_t(\mathbf{p_{25}}) \end{bmatrix}$$

Solving the ambiguity…

- Least squares problem:

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix}$$

$$\underset{25\text{x}2}{A} \quad \underset{2\text{x}1}{d} = \underset{25\text{x}1}{b}$$

Matching patches across images

- Overconstrained linear system

$$\begin{bmatrix} I_x(\mathbf{p_1}) & I_y(\mathbf{p_1}) \\ I_x(\mathbf{p_2}) & I_y(\mathbf{p_2}) \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}}) & I_y(\mathbf{p_{25}}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p_1}) \\ I_t(\mathbf{p_2}) \\ \vdots \\ I_t(\mathbf{p_{25}}) \end{bmatrix} \qquad \underset{25\text{x}2}{A} \quad \underset{2\text{x}1}{d} = \underset{25\text{x}1}{b}$$

Least squares solution for *d* given by $(A^T A)\ d = A^T b$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$A^T A$ $\qquad\qquad$ $A^T b$

The summations are over all pixels in the K x K window

Conditions for solvability

Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$A^T A$ $\qquad$ $A^T b$

When is this solvable? I.e., what are good points to track?

- $\mathbf{A^T A}$ should be invertible
- $\mathbf{A^T A}$ should not be too small due to noise
  - eigenvalues λ_1 and λ_2 of $\mathbf{A^T A}$ should not be too small
- $\mathbf{A^T A}$ should be well-conditioned
  - λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

Does this remind you of anything?

Criteria for Harris corner detector

Low-texture region

$$\sum \nabla I (\nabla I)^T$$

– gradients have small magnitude

– small λ_1, small λ_2

Edge

$$\sum \nabla I (\nabla I)^T$$

– gradients very large or very small

– large λ_1, small λ_2

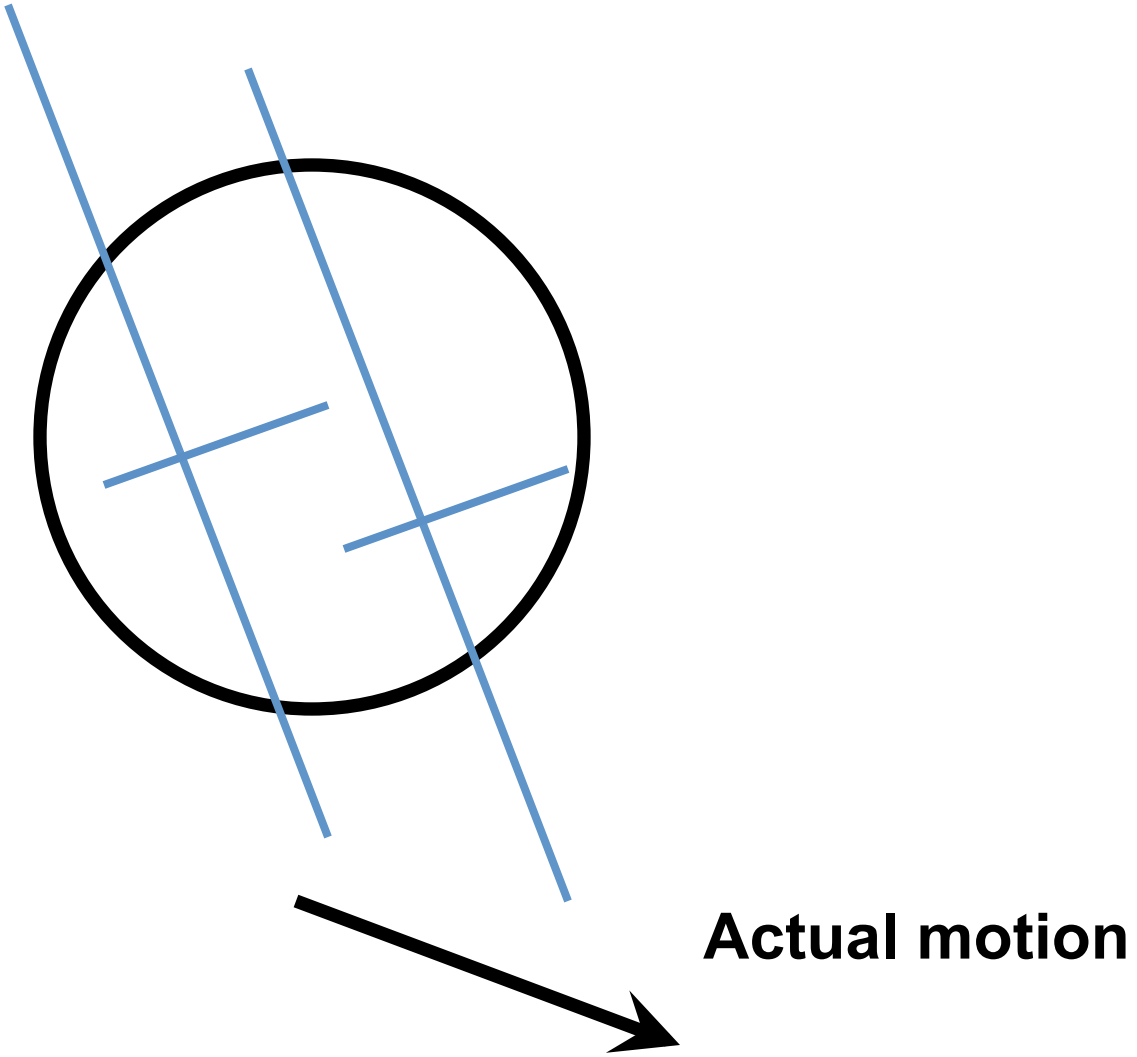
High-texture region

$$\sum \nabla I (\nabla I)^T$$

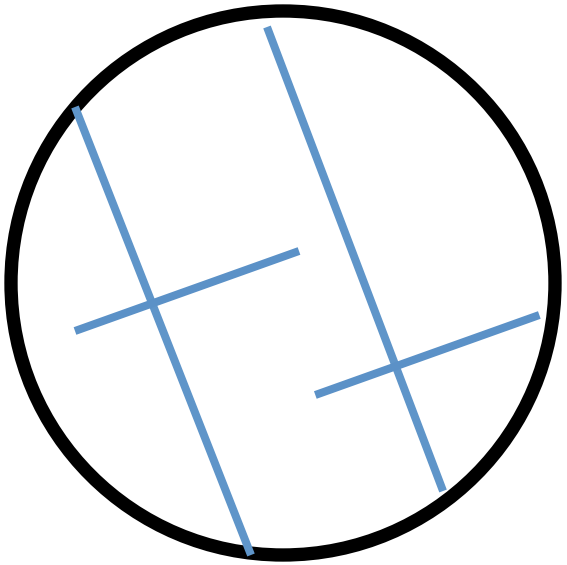
– gradients are different, large magnitudes

– large λ_1, large λ_2

The aperture problem resolved



The aperture problem resolved



Dealing with larger movements: Iterative refinement

1. Initialize (x',y') = (x,y)
2. Compute (u,v) by

Original (x,y) position

$I_t = I(x', y', t+1) - I(x, y, t)$

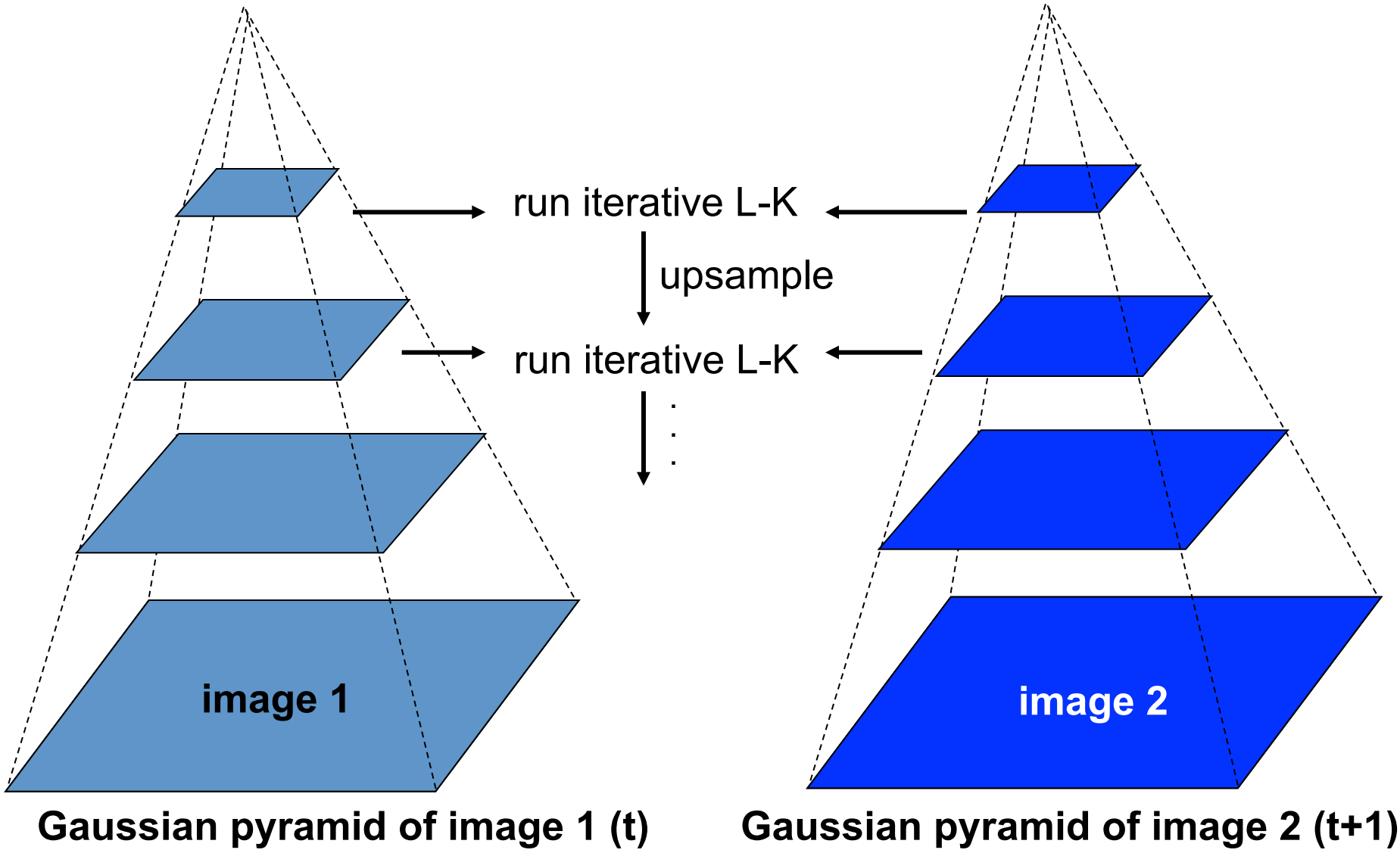
$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

2nd moment matrix for feature patch in first image

displacement

3. Shift window by (u, v): `x'=x'+u; y'=y'+v;`
4. Recalculate I_t
5. Repeat steps 2-4 until small change
   - Use interpolation for subpixel values

Dealing with larger movements: coarse-to-fine registration



Lucas-Kanade Optical Flow

- Same as Lucas-Kanade feature tracking, but for each pixel
  - As we saw, works better for textured pixels
- Operations can be done one frame at a time, rather than pixel by pixel
  - Efficient

Example of Optical Flow

YouTube Video