EE795: Computer Vision and Intelligent Systems

Spring 2012
TTh 17:30-18:45 FDH 204

Lecture 10
130221

http://www.ee.unlv.edu/~b1morris/ecg795/

Outline

- Review
  - Canny Edge Detector
  - Hough Transform
- Feature-Based Alignment
- Image Warping
- 2D Alignment Using Least Squares

Quantifying Performance

- Confusion matrix-based metrics
  - Binary {1,0} classification tasks

		actual value		
predicted outcome		p	n	total
	p'	TP	FP	P'
	n'	FN	TN	N'
	total	P	N	

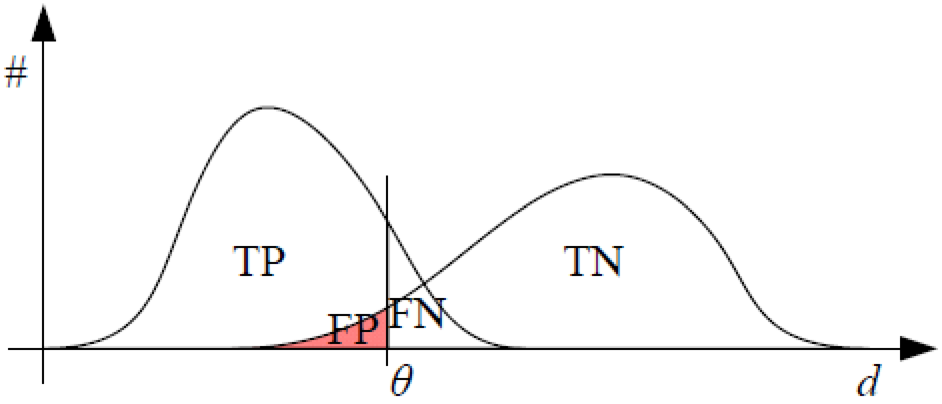
- True positives (TP) - # correct matches
- False negatives (FN) - # of missed matches
- False positives (FP) - # of incorrect matches
- True negatives (TN) - # of non-matches that are correctly rejected

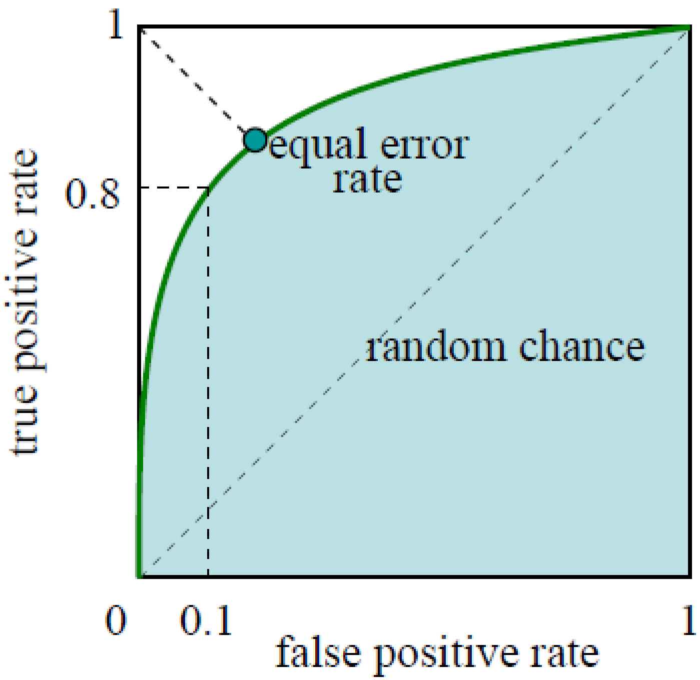
- A wide range of metrics can be defined

- True positive rate (TPR) (sensitivity)
  - $TPR = \frac{TP}{TP+FN} = \frac{TP}{P}$
  - Document retrieval → recall – fraction of relevant documents found
- False positive rate (FPR)
  - $FPR = \frac{FP}{FP+TN} = \frac{FP}{N}$
- Positive predicted value (PPV)
  - $PPV = \frac{TP}{TP+FP} = \frac{TP}{P'}$
  - Document retrieval → precision – number of relevant documents are returned
- Accuracy (ACC)
  - $ACC = \frac{TP+TN}{P+N}$

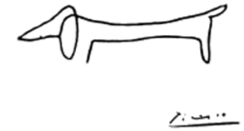
Receiver Operating Characteristic (ROC)

- Evaluate matching performance based on threshold
  - Examine all thresholds θ to map out performance curve
- Best performance in upper left corner
  - Area under the curve (AUC) is a ROC performance metric



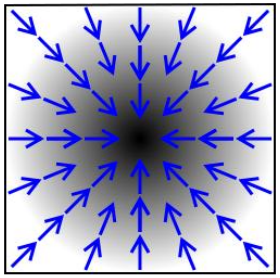


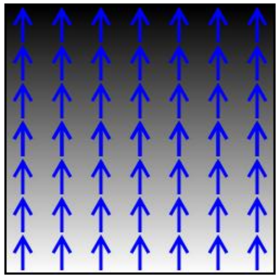
Edges



- 2D point features only provide a limited number of "good" locations for matching
- Edges are plentiful and carry semantic significance

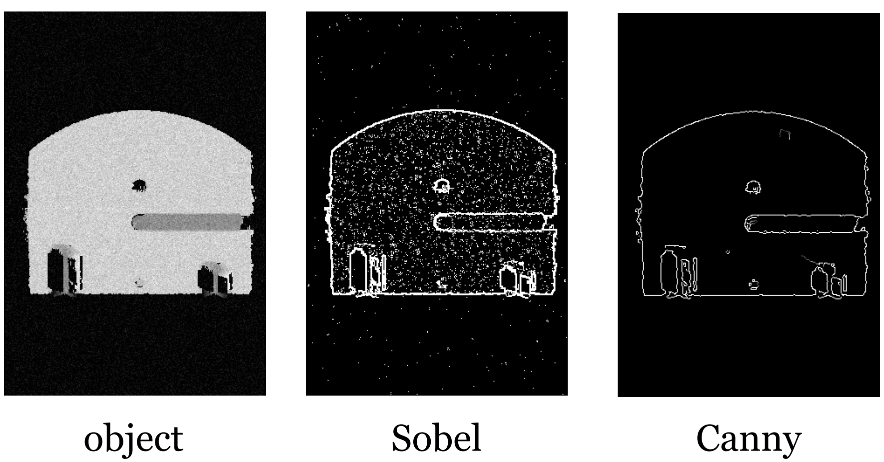
- Edges detected by gradient – slope and direction
  - $J(x) = \nabla I(x) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)(x)$
    - Smooth with Gaussian kernel before computation
  - $J_\sigma(x) = \nabla[G_\sigma(x) * I(x)] = \nabla[G_\sigma(x)] * I(x)$
    - $\nabla G_\sigma(x) = \left(\frac{\partial G_\sigma}{\partial x}, \frac{\partial I G_\sigma}{\partial y}\right)(x) = [-x, -y]\frac{1}{\sigma^3}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$
- Sharper edges obtained by Laplacian (2nd derivative)
  - $S_\sigma(x) = \nabla \cdot J_\sigma(x) = [\nabla^2 G_\sigma(x) * I(x)]$
  - Laplacian of Gaussian (LoG) kernel
    - $\nabla^2 G_\sigma(x) = \frac{1}{\sigma^3}\left(2 - \frac{x^2+y^2}{2\sigma^2}\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$
    - Can be approximated with difference of Gaussian (DoG) kernel



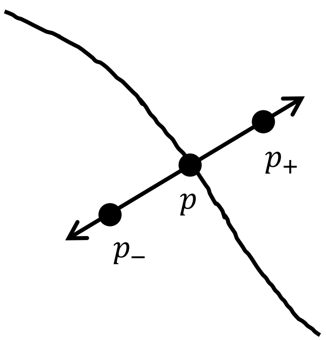


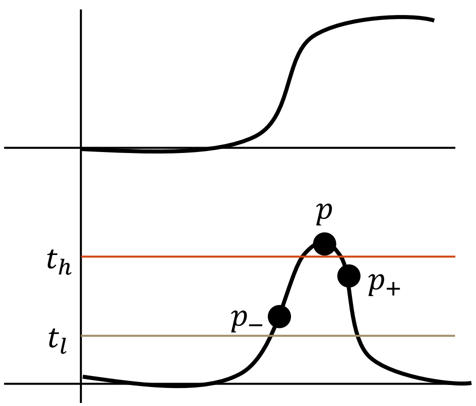
Canny Edge Detection

- Popular edge detection algorithm that produces a thin lines
- 1) Smooth with Gaussian kernel
- 2) Compute gradient
  - Determine magnitude and orientation (45 degree 8-connected neighborhood)



- 3) Use non-maximal suppression to get thin edges
  - Compare edge value to neighbor edgels in gradient direction

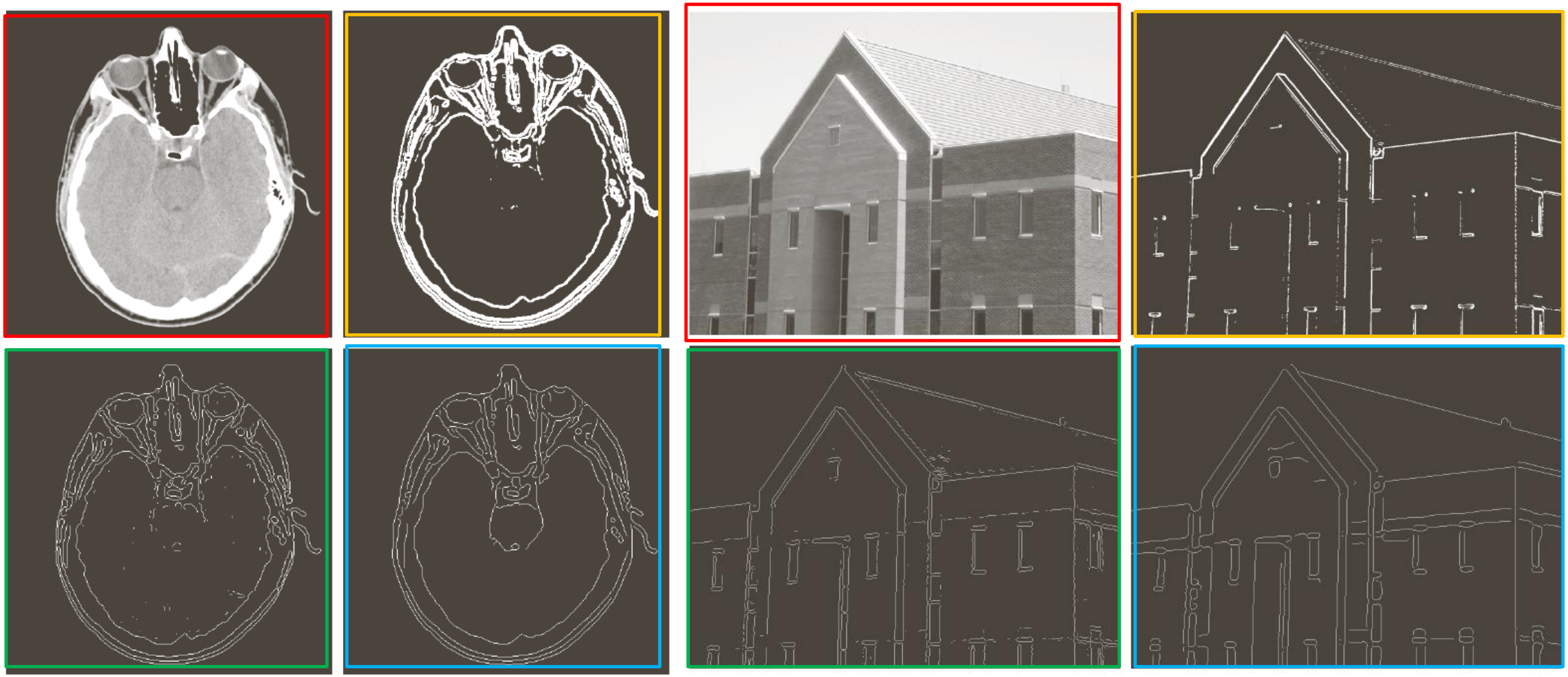




- 4) Use hysteresis thresholding to prevent streaking
  - High threshold to detect edge pixel, low threshold to trace the edge

http://homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm

Canny Edge Detection Results



- Original image
- Thresholded gradient of smoothed image (thick lines)
- Marr-Hildreth algorithm
- Canny algorithm (low noise, thin lines)

Lines

- Edges and curves make up contours of natural objects
  - Man-made world uses straight lines

- 3D lines can be used to determine vanishing points and do camera calibration
- Estimate pose of 3D scene

Hough Transform

- Lines in the real-world can be broken, collinear, or occluded
  - Combine these collinear line segments into a larger extended line
- Hough transform creates a parameter space for the line
  - Every pixel votes for a family of lines passing through it
  - Potential lines are those bins (accumulator cells) with high count
- Uses global rather than local information

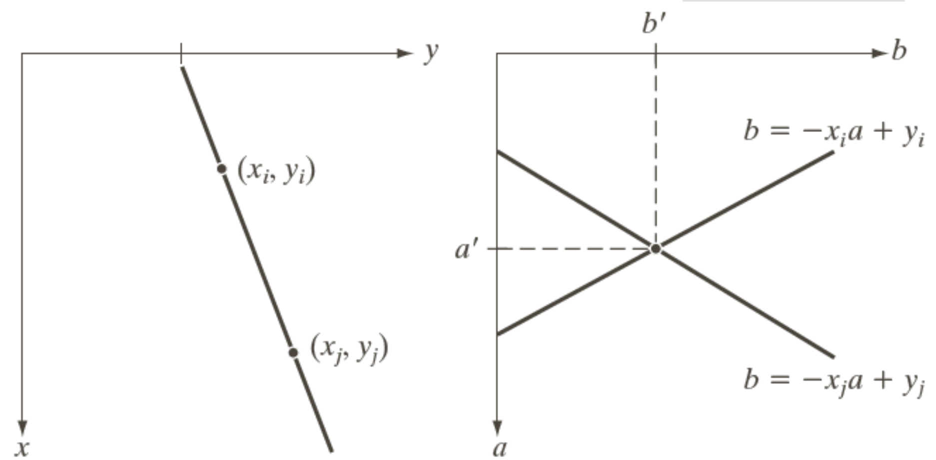
- See `hough.m, radon.m` in Matlab

Hough Transform Insight

- Want to search for all points that lie on a line
  - This is a large search (take two points and count the number of edgels)
- Infinite lines pass through a single point (x_i, y_i)
  - $y_i = ax_i + b$
    - Select any a, b
- Reparameterize
  - $b = -x_i a + y_i$
  - ab-space representation has single line defined by point (x_i, y_i)

a b

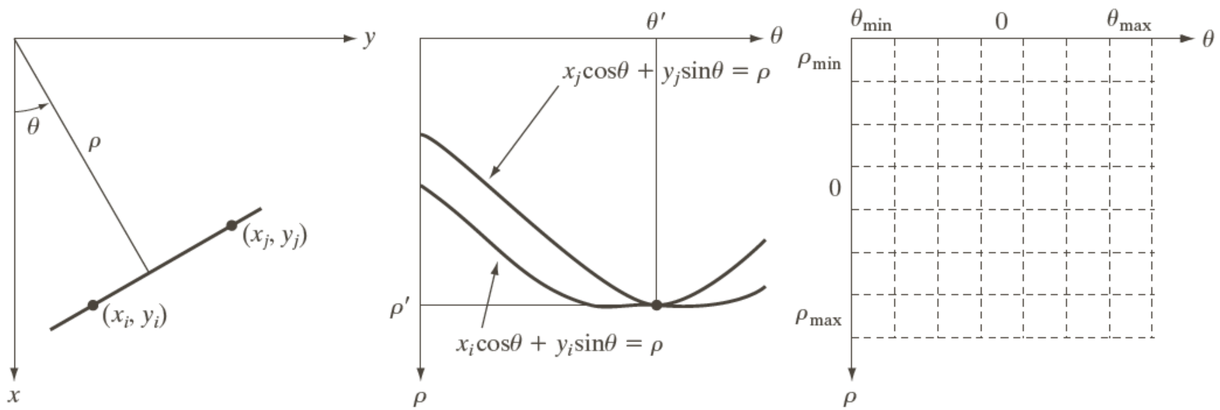
FIGURE 10.31 (a) xy-plane. (b) Parameter space.



- All points on a line will intersect in parameter space
  - Divide parameter space into cells/bins and accumulate votes across all a and b values for a particular point
  - Cells with high count are indicative of many points voting for the same line parameters (a, b)

Hough Transform in Practice

- Use a polar parameterization of a line – why?



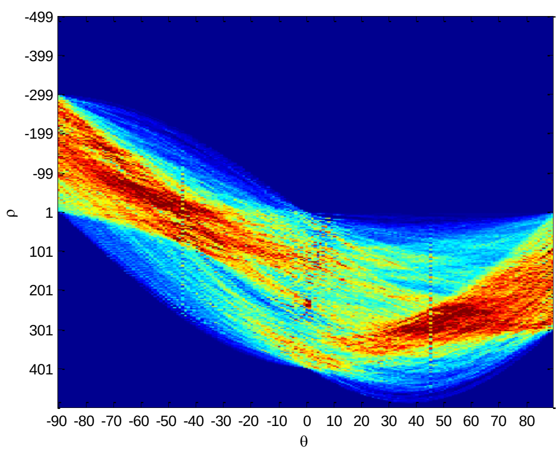
- After finding bins of high count, need to verify edge
  - Find the extent of the edge (edges do not go across the whole image)
- This technique can be extended to other shapes like circles

Hough Transform Example I

Input image

Grayscale

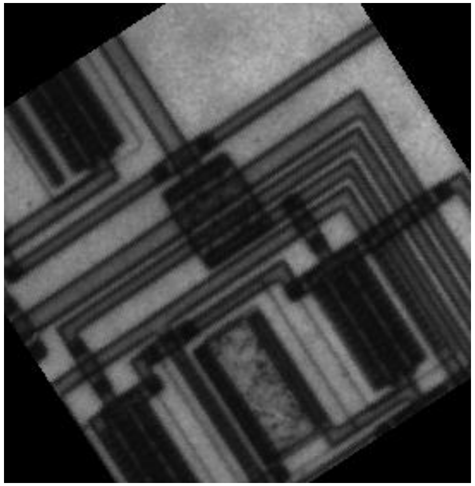
Canny edge image

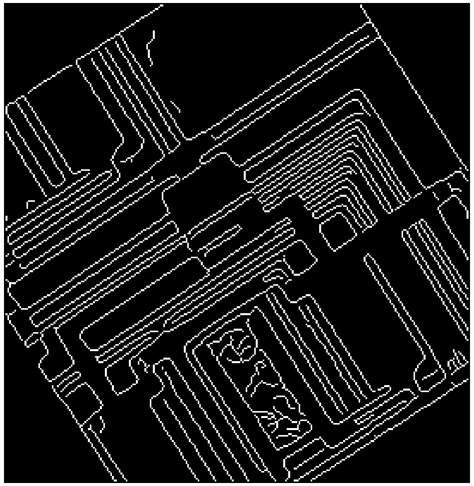


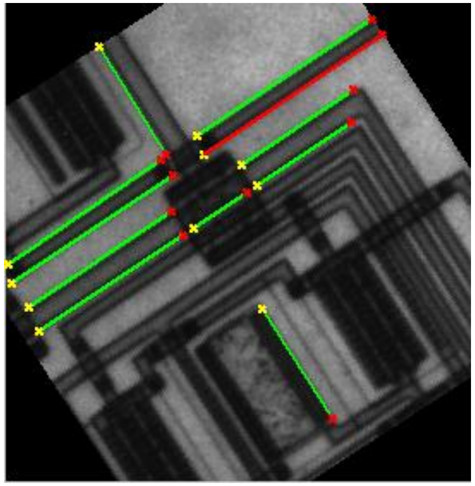
Hough space

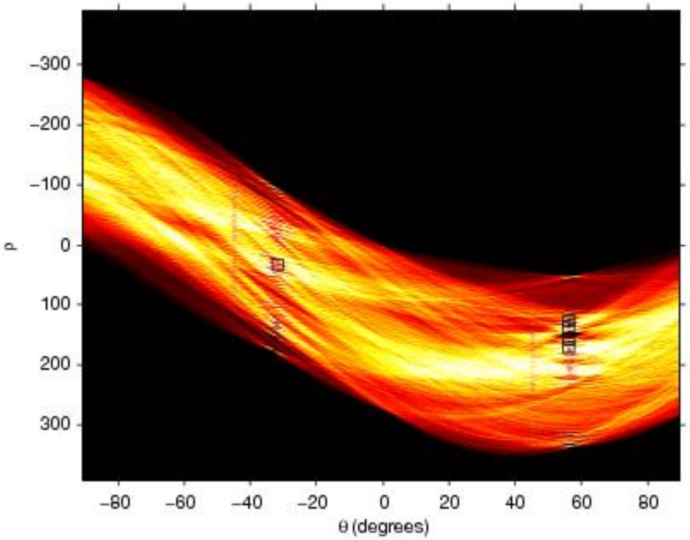
Top edges

Hough Transform Example II









http://www.mathworks.com/help/images/analyzing-images.html

Feature-Based Alignment

- After detecting and matching features, may want to verify if the matches are geometrically consistent
  - Can feature displacements be described by 2D and 3D geometric transformations

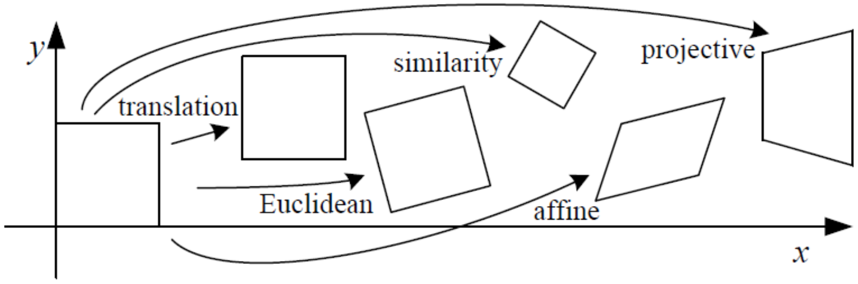


Figure 6.2 Basic set of 2D planar transformations

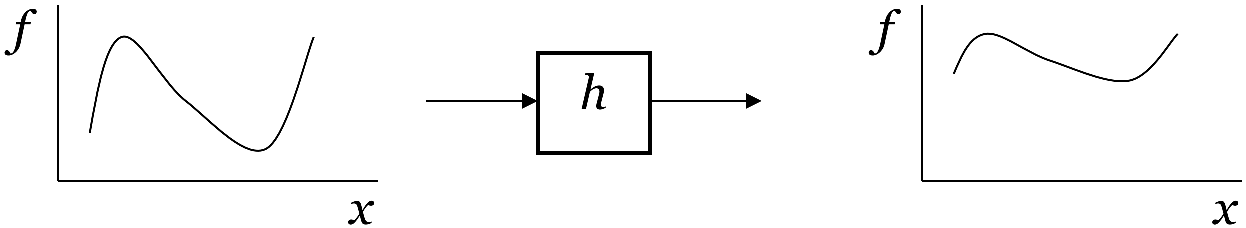
- Provides
- Geometric registration
  - 2D/3D mapping between images
- Pose estimation
  - Camera position with respect to a known 3D scene/object
- Intrinsic camera calibration
  - Find internal parameters of cameras (e.g. focal length, radial distortion)

Motion models

- What happens when we take two images with a camera and try to align them?
- translation?
- rotation?
- scale?
- affine?
- perspective?

Image Warping

- image filtering: change *range* of image
  - $g(x) = h(f(x))$



- image warping: change *domain* of image
  - $g(x) = f(h(x))$

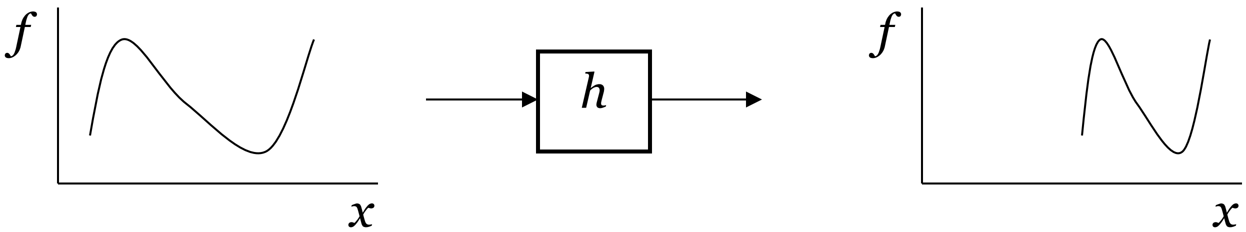
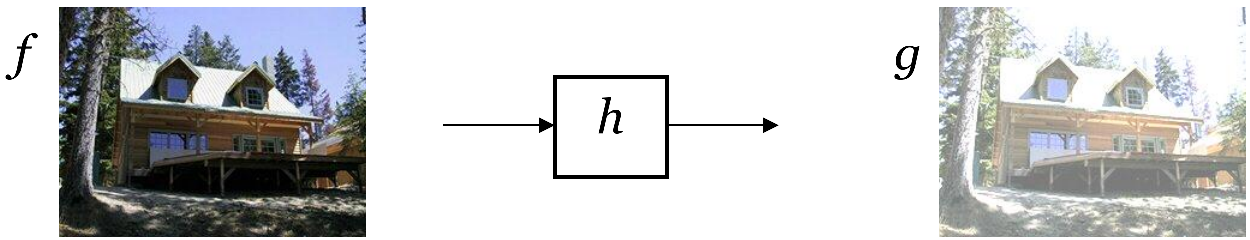
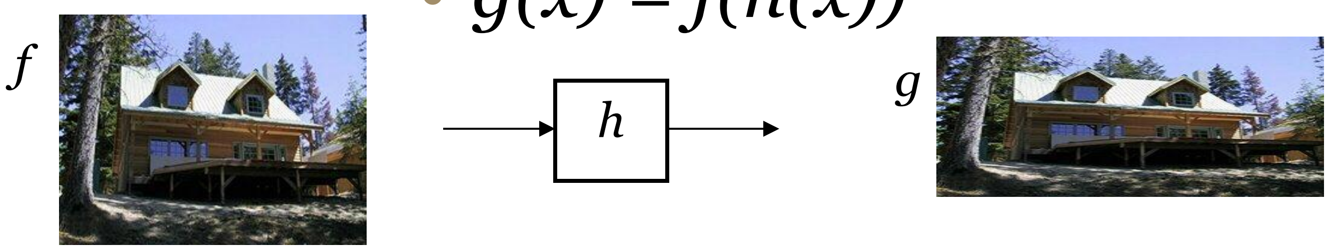


Image Warping

- image filtering: change *range* of image
  - $g(x) = h(f(x))$



- image warping: change *domain* of image
  - $g(x) = f(h(x))$



Parametric (global) warping

- Examples of parametric warps:


translation


rotation


aspect


affine


perspective


cylindrical

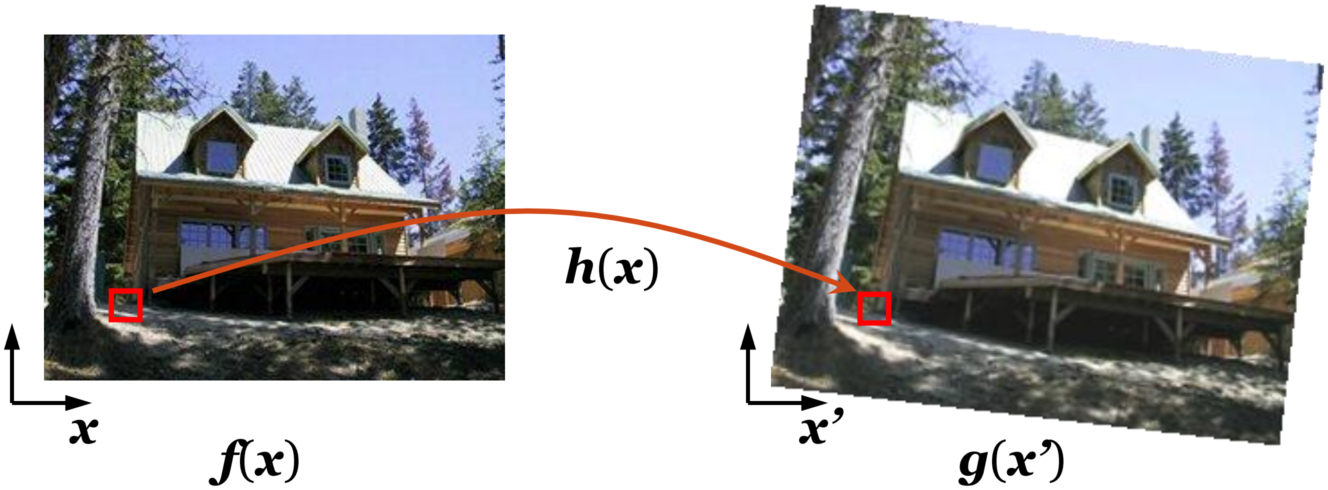
2D coordinate transformations

- translation: $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ $\quad \boldsymbol{x} = (x,y)$
- rotation: $\boldsymbol{x}' = \boldsymbol{R}\,\boldsymbol{x} + \boldsymbol{t}$
- similarity: $\boldsymbol{x}' = s\,\boldsymbol{R}\,\boldsymbol{x} + \boldsymbol{t}$
- affine: $\boldsymbol{x}' = \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{t}$
- perspective: $\underline{\boldsymbol{x}}' \cong \boldsymbol{H}\,\underline{\boldsymbol{x}}$ $\quad \underline{\boldsymbol{x}} = (x,y,1)$

  ($\underline{\boldsymbol{x}}$ is a *homogeneous* coordinate)
- These all form a nested *group* (closed w/ inv.)

Image Warping

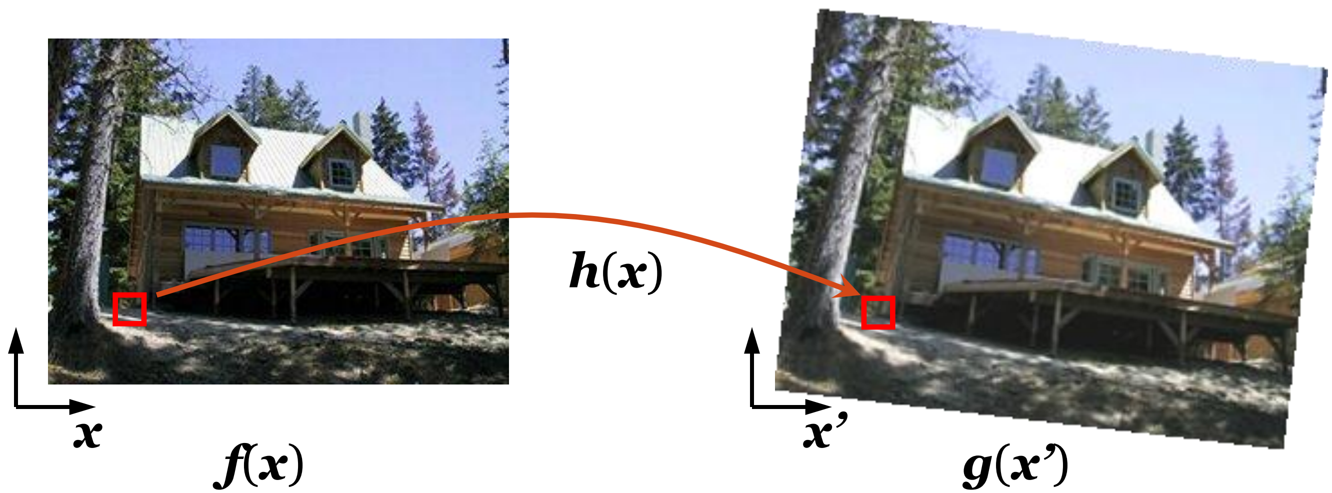
- Given a coordinate transform $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$ and a source image $\boldsymbol{f}(\boldsymbol{x})$, how do we compute a transformed image $\boldsymbol{g}(\boldsymbol{x}') = \boldsymbol{f}(\boldsymbol{h}(\boldsymbol{x}))$?



Forward Warping

- Send each pixel $\boldsymbol{f}(\boldsymbol{x})$ to its corresponding location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$ in $\boldsymbol{g}(\boldsymbol{x}')$

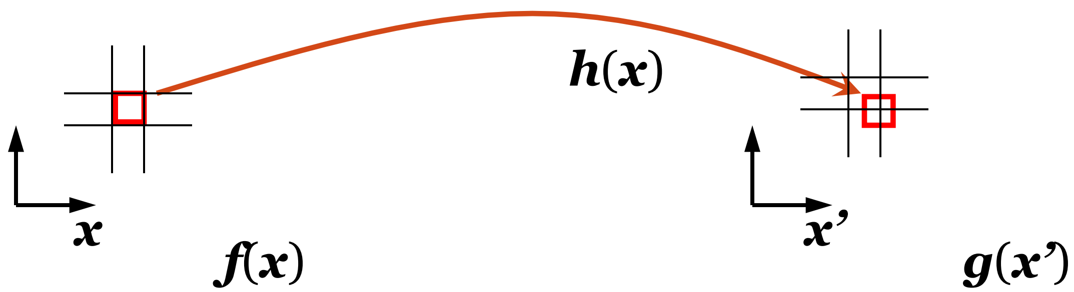
- What if pixel lands "between" two pixels?



Forward Warping

- Send each pixel $\boldsymbol{f}(\boldsymbol{x})$ to its corresponding location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$ in $\boldsymbol{g}(\boldsymbol{x}')$

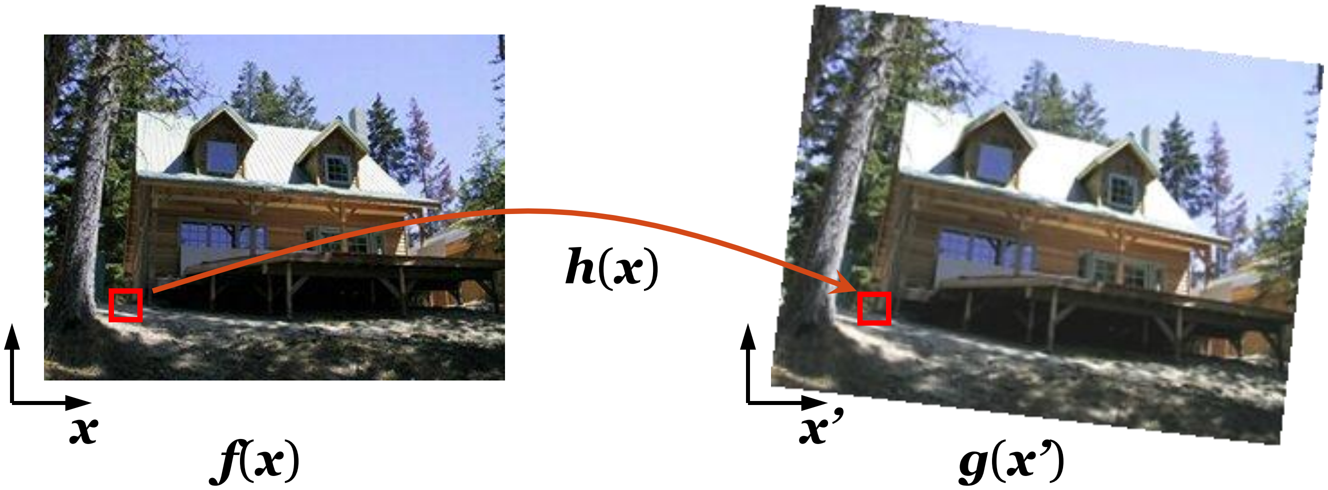
- What if pixel lands "between" two pixels?
- Answer: add "contribution" to several pixels, normalize later (*splatting*)
- See `griddata.m`



Inverse Warping

- Get each pixel $g(x')$ from its corresponding location $x = h^{-1}(x')$ in $f(x)$

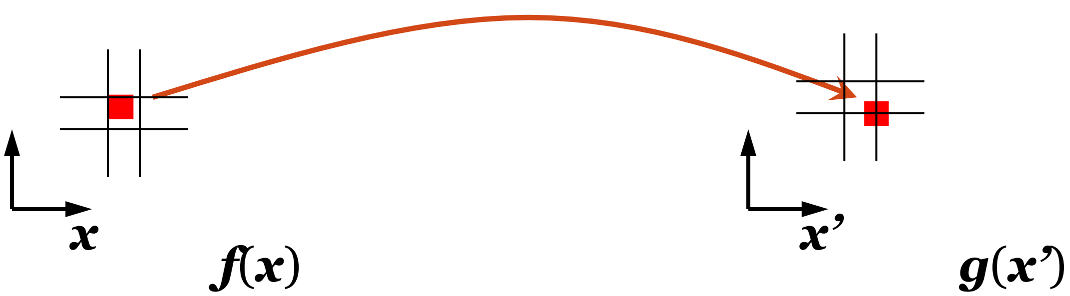
- What if pixel comes from “between” two pixels?



Inverse Warping

- Get each pixel $\boldsymbol{g}(\boldsymbol{x}')$ from its corresponding location $\boldsymbol{x} = \boldsymbol{h}^{-1}(\boldsymbol{x}')$ in $\boldsymbol{f}(\boldsymbol{x})$

- What if pixel comes from "between" two pixels?
- Answer: *resample* color value from *interpolated* (*prefiltered*) source image
- See `interp2.m`



Interpolation

- Possible interpolation filters:
  - nearest neighbor
  - bilinear
  - bicubic (interpolating)
  - sinc / FIR
- Needed to prevent "jaggies" and "texture crawl" (see demo)

Forward vs. Inverse Warping

- Which type of warping is better?

- Usually inverse warping is preferred
  - It eliminates holes
  - However, it requires an invertible warp function
    - Not always possible

Least Squares Alignment

- Given a set of matched features $\{(x_i, x_i')\}$, minimize sum of squared residual error
  - $E_{LS} = \sum_i \|r_i\|^2 = \sum_i \|f(x_i; p) - x_i'\|^2$
    - $f(x_i; p)$ - is the predicted location based on the transformation p

- The unknowns are the parameters p
  - Need to have a model for transformation
  - Estimate the parameters based on matched features

Linear Least Squares Alignment

- Many useful motion models have a linear relationship between motion and parameters p
  - $\Delta x = x' - x = J(x)p$
    - $J = \frac{\partial f}{\partial p}$ - the Jacobian of the transform f with respect to the motion parameters p
- Linear least squares
  - $E_{LLS} = \sum_i \|J(x_i)p - \Delta x_i\|^2 = p^T A p - 2p^T b + c$
    - Quadratic form
- The minimum is found by solving the normal equations
  - $Ap = b$
    - $A = \sum_i J^T(x_i)J(x_i)$ – Hessian matrix
    - $b = \sum_i J^T(x_i)\Delta x_i$
  - Gives the LLS estimate for the motion parameters

Jacobians of 2D Transformations

Transform	Matrix	Parameters p	Jacobian J
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
affine	$\begin{bmatrix} 1+a_{00} & a_{01} & t_x \\ a_{10} & 1+a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
projective	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \ldots, h_{21})$	(see Section 6.1.3)

Table 6.1 Jacobians of the 2D coordinate transformations $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p})$ shown in Table 2.1, where we have re-parameterized the motions so that they are identity for $\boldsymbol{p} = 0$.

Improving Motion Estimates

- A number of techniques can improve upon linear least squares
- Uncertainty weighting
  - Weight the matches based certainty of the match – texture in the match region
- Non-linear least squares
  - Iterative algorithm to guess parameters and iteratively improve guess
- Robust least squares
  - Explicitly handle outliers (bad matches) – don't use L2 norm
- RANSAC
  - Randomly select subset of corresponding points, compute initial estimate of p, count the inliers from all the other correspondences, good match has many inliers