Feature Detection and Matching

CSE 6367: Computer Vision

Instructor: William J. Beksi

Introduction

- **Feature detection** and **matching** are an essential component of many computer vision applications
- What kinds of *features* should be *detected* and then *matched* in a given image?

Introduction

- The first kind of feature that we may notice are specific locations in the images, e.g. mountain peaks, building corners, doorways, patches of snow, etc.
- These kinds of localized features are called **keypoint features** or **interest points** (or even **corners**) and are described by the appearance of patches of pixels surrounding the point location

Introduction

- Another class important features are **edges**, e.g. the profile of mountains against the sky
- These kinds of features can be matched based on their orientation and local appearance (edge profiles) and can also be good indicators of object boundaries and **occlusion** events in image sequences
- Edges can be grouped into longer **curves** and **straight line segments**, which can be directly matched or analyzed to find **vanishing points** and thus internal and external camera parameters

Introduction

(a)

(b)

(c)

(d)

- A variety of feature detectors and descriptors can be used to analyze, describe, and match images

Point Features

- **Point features** can be used to find a sparse set of corresponding locations in different images
- This is often done as a precursor to computing camera pose, which is a prerequisite for computing a dense set of correspondences using stereo matching
- Such correspondences can also be used to align different images, e.g. when stitching image mosaics or performing video stabilization

Point Features

- Point features are used extensively to perform object instance and category recognition
- A key advantage of keypoints is that they permit matching even in the presence of clutter (occlusion) and large scale and orientation changes
- Feature-based correspondence techniques have been used since the early days of stereo matching and have gained popularity for image-stitching applications as well as fully automated 3D modeling

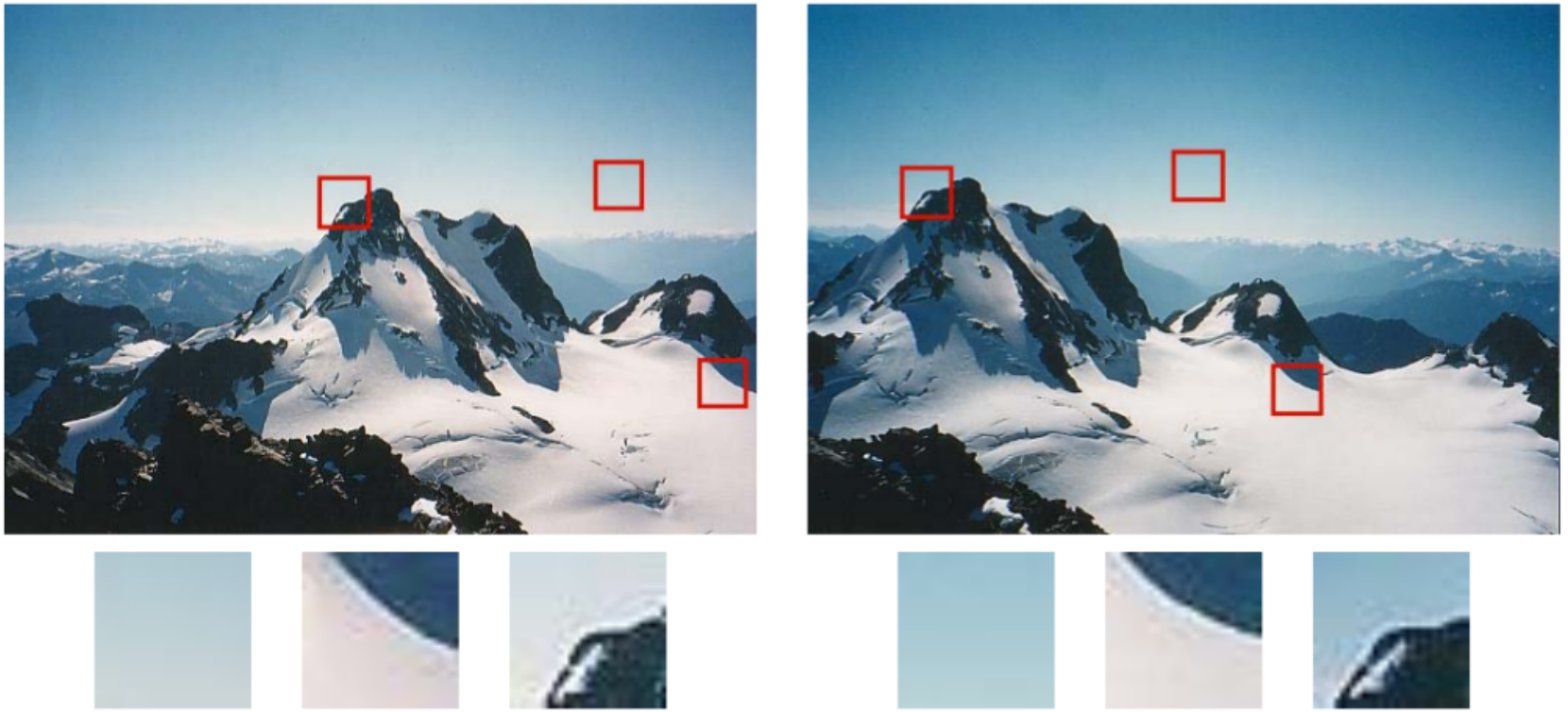
Point Features

- There are two main approaches to finding feature points and their correspondences:
  - The first is to find features in one image that can be accurately *tracked* using a local search technique such as correlation or least squares
  - The second is to independently detect features in all the images under consideration and then *match* features based on their local appearance
- The first approach is more suitable when images are taken from nearby viewpoints or in rapid succession (e.g. video images)
- The second approach is more suitable when a large number of motion or appearance change is expected (e.g. object recognition)

Point Features

- The keypoint detection and matching pipeline can be split into four separate stages:
  - **Feature detection** (extraction) stage - each image is searched for locations that are likely to match well in other images
  - **Feature description** stage - each region around detected keypoints is converted into a more compact and stable (invariant) *descriptor* that can be matched against other descriptors
  - **Feature matching** stage - efficiently searches for likely matching candidates in other images
  - **Feature tracking** stage - is an alternative to the third stage that only searches a small neighborhood around each detected feature and is therefore more suitable for video processing

Feature Detectors

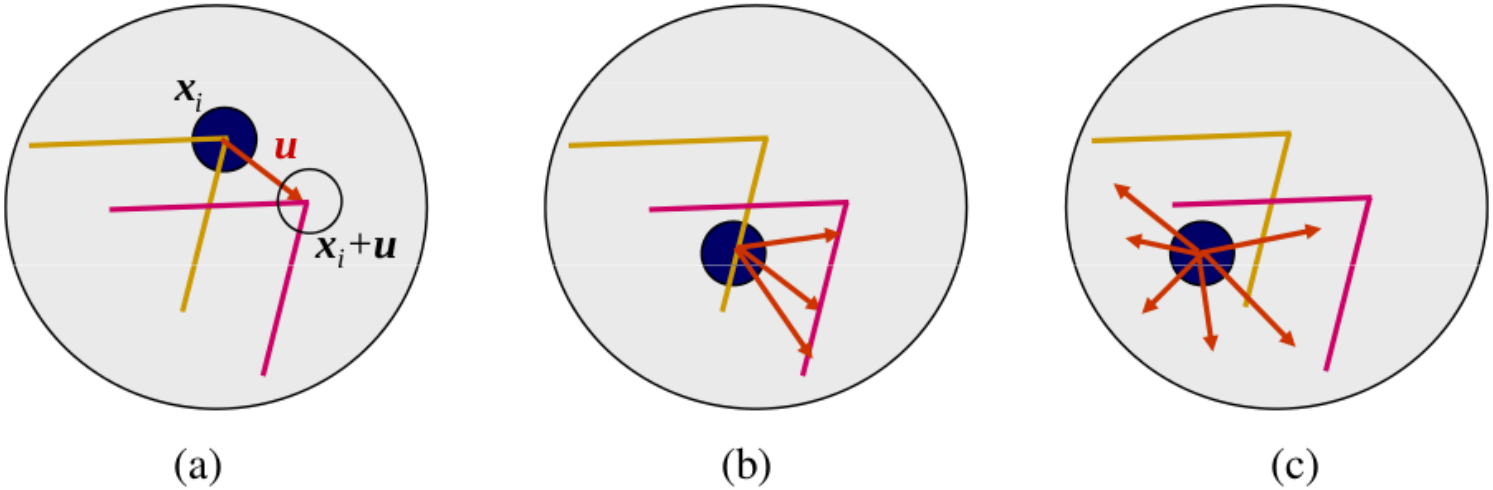


- How can we find image locations where we can reliably find correspondences with other images, i.e. what are good features to track?

Feature Detectors

- Textureless patches are nearly impossible to localize
- Patches with large contrast changes (gradients) are easier to localize, although straight line segments at a single orientation suffer from the **aperture problem** (i.e. it is only possible to align the patches along the direction normal to the edge direction)
- Patches with gradients in at least two (significantly) different orientations are the easiest to localize

Feature Detectors



- Aperture problems for different image patches: (a) stable ("corner-like") flow; (b) classic aperture problem (barber-pole illusion); (c) textureless region

Feature Detectors

- The simplest possible matching criterion for comparing two image patches is their (weighted) summed square difference

$$E_{\mathrm{WSSD}}(\mathbf{u}) = \sum_i w(\mathbf{x}_i)[I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2$$

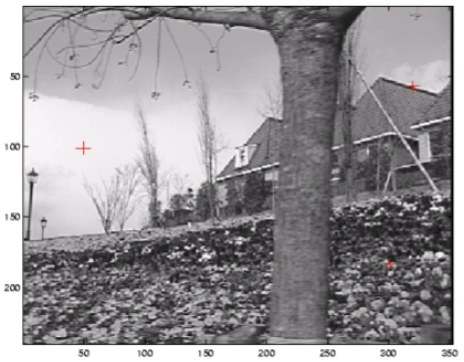
where I_0 and I_1 are the two images being compared, $\mathbf{u} = [u, v]$ is the *displacement* vector, $w(\mathbf{x})$ is a spatially varying weighting (or window) function, and i is over all pixels in the patch

Feature Detectors

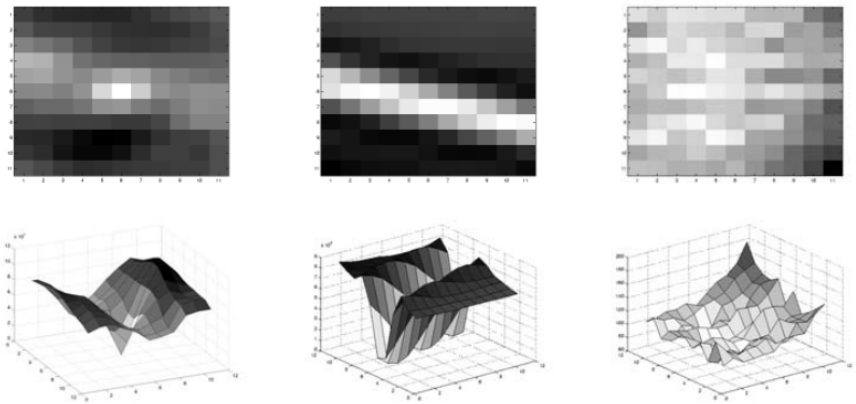
- When performing feature detection, we do not know which other image locations the feature will end up being matched against
- Therefore, we can only compute how stable this metric is w.r.t small variations in position $\Delta\mathbf{u}$ by comparing an image patch against itself which is known as an **auto-correlation surface** function

$$E_{\mathrm{AC}}(\Delta\mathbf{u}) = \sum_i w(\mathbf{x}_i)[I_0(\mathbf{x}_i + \Delta\mathbf{u}) - I_0(\mathbf{x}_i)]^2$$

Feature Detectors



(a)



(b) (c) (d)

- Three auto-correlation surfaces: (a) original image; (b) flower bed patch (good unique minimum); (c) roof edge patch (1D aperture problem); (d) cloud patch (no good peak)

Feature Detectors

- Using a Taylor series expansion of the image function $I_0(\mathbf{x}_i + \Delta\mathbf{u}) \approx I_0(\mathbf{x}_i) + \nabla I_0(\mathbf{x}_i) \cdot \Delta\mathbf{u}$, we can approximate the auto-correlation surface as

$$\begin{aligned} E_{\mathrm{AC}}(\Delta\mathbf{u}) &= \sum_i w(\mathbf{x}_i)[I_0(\mathbf{x}_i + \Delta\mathbf{u}) - I_0(\mathbf{x}_i)]^2 \\ &\approx \sum_i w(\mathbf{x}_i)[I_0(\mathbf{x}_i) + \nabla I_0(\mathbf{x}_i) \cdot \Delta\mathbf{u} - I_0(\mathbf{x}_i)]^2 \\ &= \sum_i w(\mathbf{x}_i)[\nabla I_0(\mathbf{x}_i) \cdot \Delta\mathbf{u}]^2 \\ &= \Delta\mathbf{u}^T A \Delta\mathbf{u} \end{aligned}$$

where $\nabla I_0(\mathbf{x}_i) = (\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y})(\mathbf{x}_i)$ is the **image gradient** at $\mathbf{x}_i$

Feature Detectors

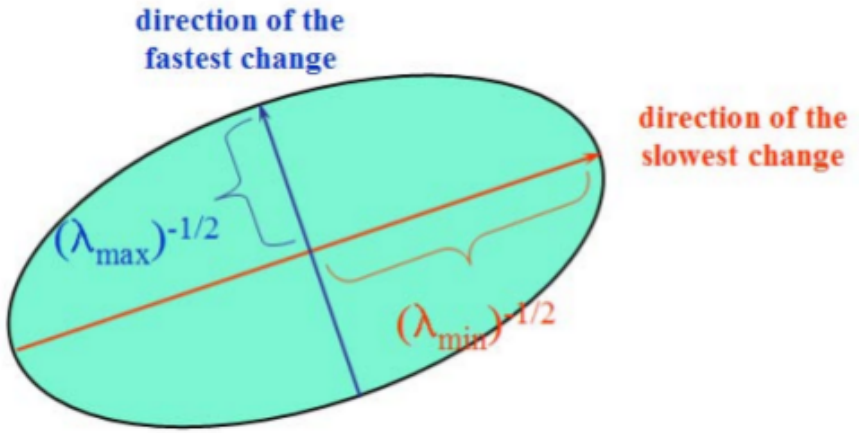
- The auto-correlation matrix A can be written as

$$A = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

  where the weighted summations have been replaced with discrete convolutions using the weighting kernel w

- The inverse of A provides a lower bound on the uncertainty in the location of a matching pixel and thus is a useful indicator of which patches can be reliably matched

Feature Detectors



- Uncertainty ellipse corresponding to an eigenvalue analysis of the auto-correlation matrix A
- Since the larger uncertainty depends on the smaller eigenvalue, it makes sense to find maxima in the smaller eigenvalue to locate good features to track

Harris Detector

- The minimum eigenvalue λ_0 is not the only quantity that can be used to find keypoints
- The **Harris** detector makes use of a simpler quantity

$$\det(A) - \alpha \text{trace}(A)^2 = \lambda_0 \lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$$

with $\alpha = 0.06$

- Unlike eigenvalue analysis, this quantity does not require the use of square roots and yet is still rotationally invariant
- It also downweights edge-like features where $\lambda_1 \gg \lambda_0$

Interest Operator Responses

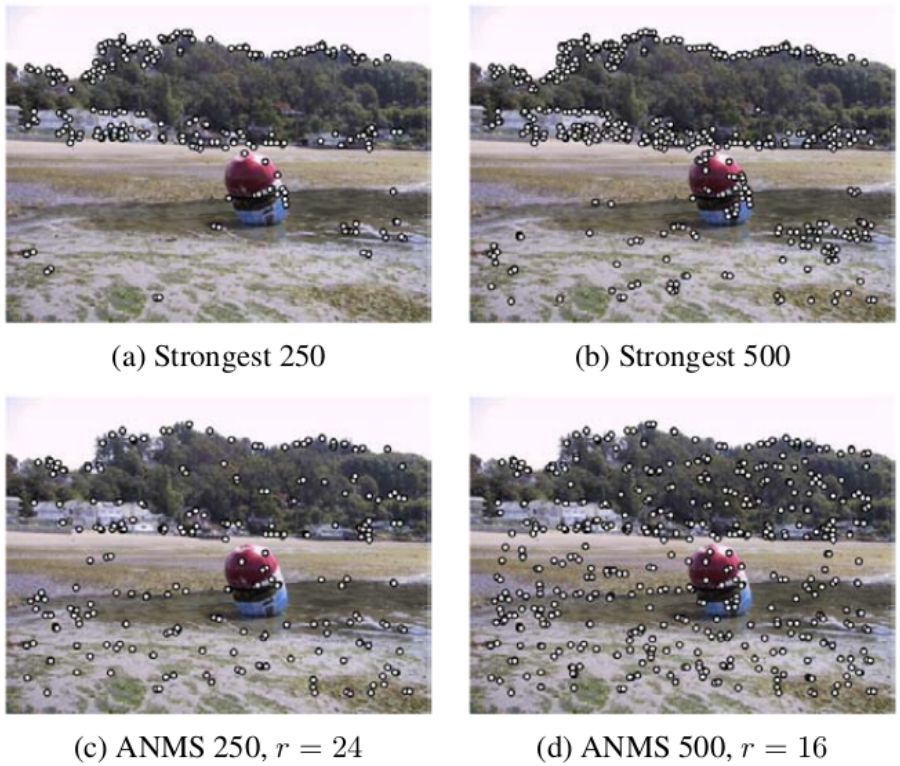
(a) (b) (c)

- (a) sample image; (b) Harris response; (c) DoG response
- The circle sizes and colors indicate the scale at which each interest point was detected
- Note how the two detectors then to respond at complementary locations

Adaptive Non-Maximal Suppression Detector

- While most feature detectors simply look for local maxima in the interest function, this can lead to an uneven distribution of feature points across the image (e.g. points will be denser in regions of higher contrast)
- To mitigate this problem the **adaptive non-maximal suppression** (ANMS) detector only detects features that are both local maxima and whose response value is significantly (10%) greater than its neighbors within a radius r

Adaptive Non-Maximal Suppression Detector



(a) Strongest 250 (b) Strongest 500

(c) ANMS 250, $r = 24$ (d) ANMS 500, $r = 16$

- The upper two images show the strongest 250 and 500 interest points
- The lower two images show the interest points selected with ANMS along with the suppression radius r

Outline of a Basic Feature Detection Algorithm

1. Compute the horizontal and vertical derivatives of the image I_x and I_y by convolving the original image with derivatives of Gaussians
2. Compute the three images corresponding to the outer products of these gradients (the matrix A is symmetric, so only three entries are needed)
3. Convolve each of these images with a larger Gaussian
4. Compute a scalar interest measure using one of the discussed formulas
5. Find local maxima above a certain threshold and report them as detected feature point locations

Measuring Repeatability

- The **repeatability** of a feature detector is defined as the frequency in which keypoints detected in one image are found within ϵ (say $\epsilon = 1.5$) pixels of the corresponding location in a transformed image
- The **information content** available at each detected feature point can be defined as the entropy of a set of rotationally invariant local grayscale descriptors

Scale Invariance

- In many situations, detecting features at the finest scale possible may not be appropriate (e.g. when matching images with little high frequency detail, fine-scale features may not exist
- One solution is to extract features at a variety of scales, e.g. by performing the same operations at multiple resolutions in a pyramid and then matching features at the same level

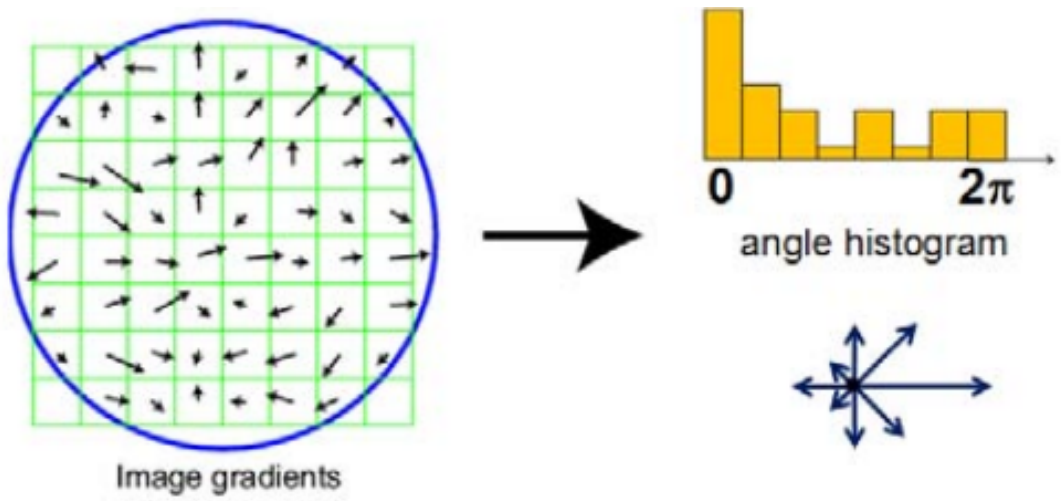
Scale Invariance

- This kind of approach is suitable when the images being matched do not undergo large scale changes
- However, for most object recognition applications, the scale of the object in the image is unknown
- Thus, instead of extracting features at many different scales and then matching all of them, it is more efficient to extract features that are stable in both location *and* scale

Rotational Invariance and Orientation Estimation

- In addition to dealing with scale changes, most image matching and object recognition algorithms need to deal with in-plane image rotation
- One way to address this problem is to design descriptors that are rotationally invariant, however such descriptors have poor discriminability (i.e. they map different looking patches to the same descriptor)
- A better method is to estimate a **dominate orientation** at each detected keypoint, then a scaled and oriented patch around the detected point can be extracted and used to form a feature descriptor

Dominate Orientation Estimate

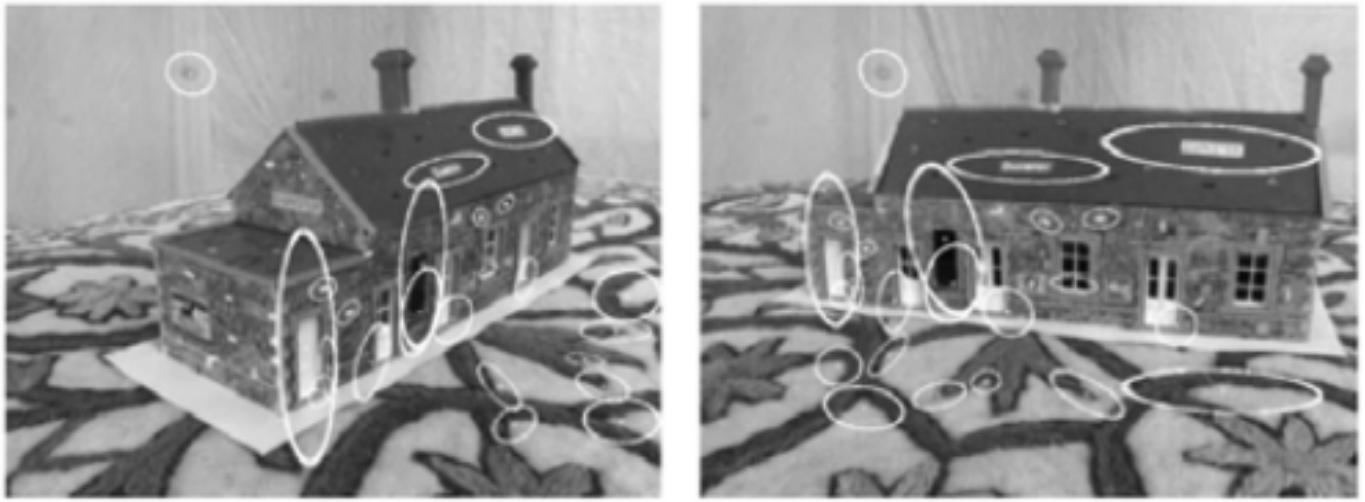


- A dominate orientation estimate can be computed by creating a histogram of all the gradient orientations (weighted by their magnitude or after thresholding out small gradients) and then finding the significant peaks in this distribution

Affine Invariance

- Affine invariant detectors not only respond at consistent locations after scale and orientation changes, they also respond consistently across affine deformations such as (local) perspective foreshortening
- Affine invariance may be introduced by fitting an ellipse to the auto-correlation or Hessian matrix (using eigenvalue analysis) and then using the principle axes and ratios of this fit as the affine coordinate frame

Affine Region Detectors

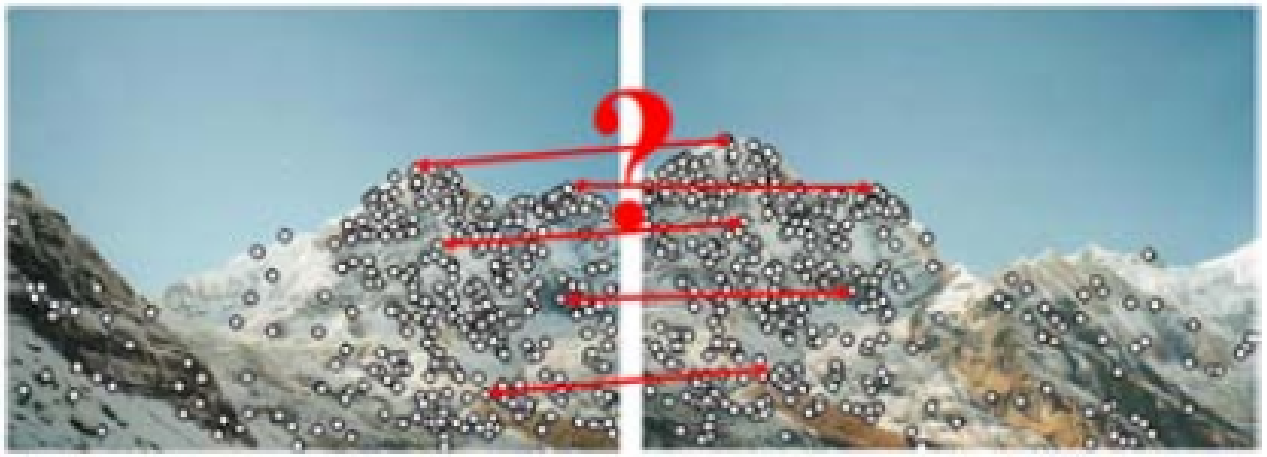


- Affine region detectors used to match two images taken from dramatically different viewpoints

Feature Descriptors

- After detecting features (keypoints) we must **match** them, i.e. determine which features come from corresponding locations in different images
- The local appearance of features will change in orientation and scale (sometimes even undergo affine deformations)
- Therefore, we are interested in extracting a local scale, orientation, or affine frame estimate and then using this to resample the patch before forming the feature descriptor

Feature Descriptors

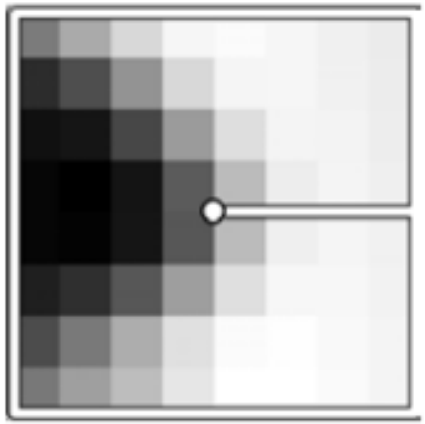


- Even after compensating for these changes, the local appearance of image patches will usually vary from image to image
- How can we extract local descriptors that are invariant to inter-image variations yet still discriminative enough to establish correct correspondences?

Bias and Gain Normalization (MOPS)

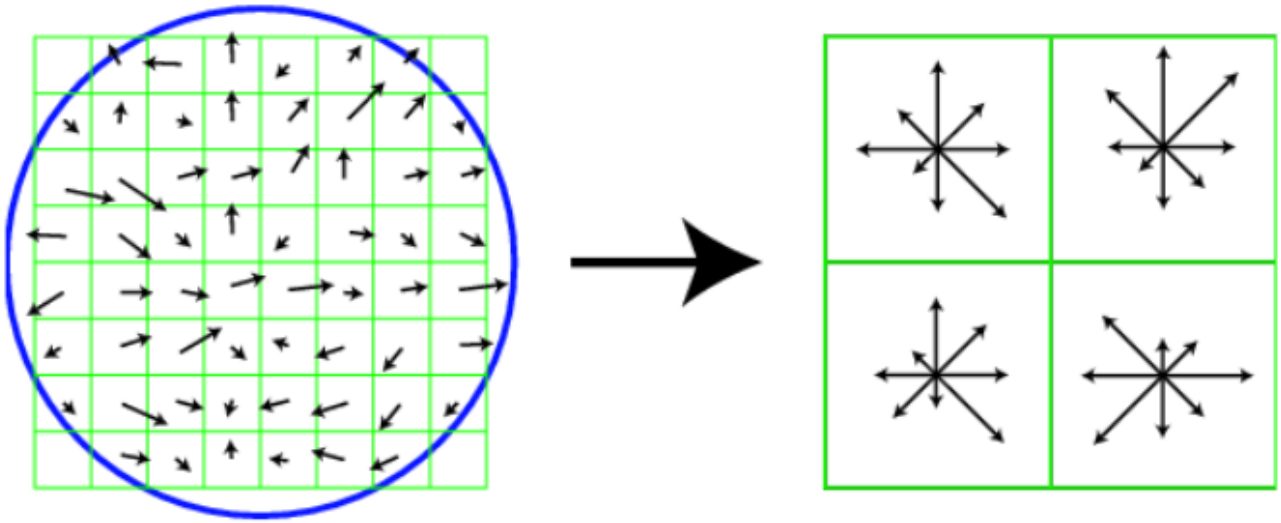
- For tasks that do not exhibit large amounts of foreshortening (e.g. image stitching) simple normalized intensity patches perform reasonably well and are simple to implement
- To compensate for slight inaccuracies in the feature point detector (location, orientation, and scale), these **multi-scale oriented patches** (MOPS) are sampled at a spacing of five pixels relative to the detection scale

Bias and Gain Normalization (MOPS)



- MOPS descriptors are formed using an 8×8 sampling of bias and gain normalized intensity values
- This low frequency sampling gives the features some robustness to interest point location error and is achieved by sampling at a higher pyramid level than the detection scale

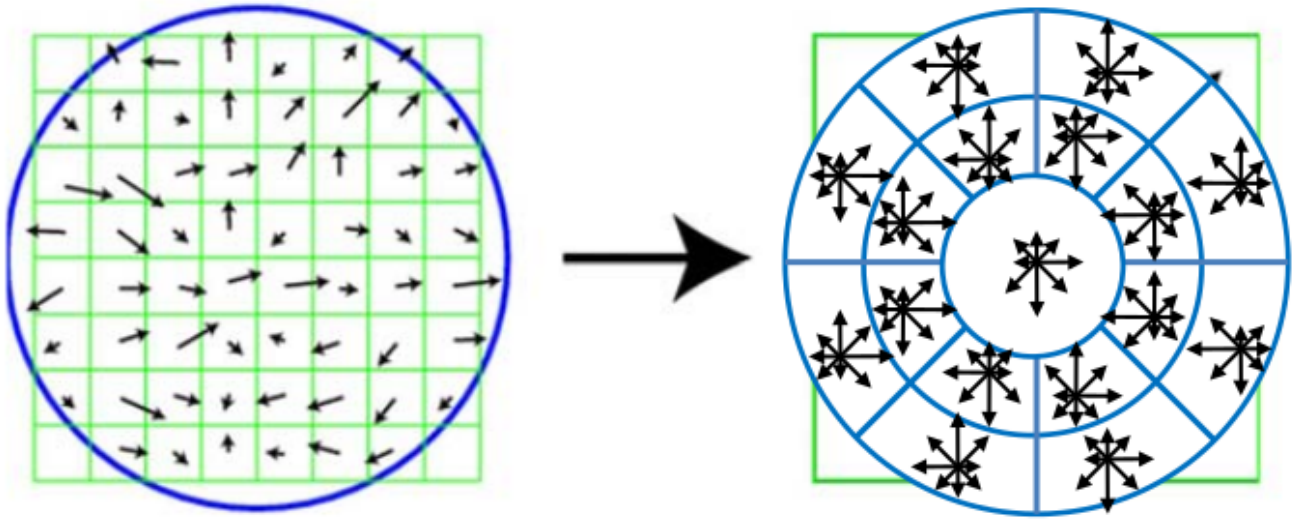
Scale Invariant Feature Transform (SIFT)



(a) image gradients (b) keypoint descriptor

- A schematic representation of **SIFT**: (a) gradient orientations and magnitudes are computed at each pixel and weighted by a Gaussian fall-off function (blue circle); (b) a weighted gradient orientation histogram is then computed in each subregion using trilinear interpolation

Gradient Location-Orientation Histogram (GLOH)



(a) image gradients (b) keypoint descriptor

- The **GLOH** descriptor is a variant on SIFT that uses a log-polar binning structure instead of square bins to compute orientation histograms
- The 272-dimensional descriptor is then projected onto a 128-dimensional descriptor using PCA trained on a large database

Performance of Local Descriptors

- The field of feature descriptors continues to rapidly evolve from handcrafted to data driven descriptors
- In addition to optimizing for repeatability across *all* object classes, it is also possible to develop class- or instance-specific feature detectors that maximize *discriminability* from other classes

Feature Tracking

- Once we have extracted features and their descriptors from two or more images, the next step is to establish some preliminary feature matches between these images
- This problem can be divided into two separate components:
  - Select a **matching strategy** to determine which correspondences are passed on to the next stage for further processing
  - Devise efficient **algorithms** and **data structures** to perform this matching as quickly as possible

Application: Recognizing Objects in a Cluttered Scene

- Two of the training images in the database are shown on the left
- These are matched to the cluttered scene using SIFT features (small squares)
- The affine warp of each recognized database image is shown as a larger parallelogram

Matching Strategy and Error Rates

- Determining which feature matches are reasonable to process further depends on the context in which the matching is being performed
- To begin, we assume that the feature descriptors have been designed so that Euclidean (vector magnitude) distances in feature space can be directly used for ranking potential matches

Matching Strategy and Error Rates

- Given a Euclidean distance metric, the simplest matching strategy is to set a threshold (maximum distance) and return all matches from other images (within this threshold)
- Setting the threshold too high results in too many **false positives**, i.e. incorrect matches being returned
- Setting the threshold too low results in too many **false negatives**, i.e. too many correct matches being missed

Matching Strategy and Error Rates

- We can quantify the performance of a matching algorithm at a particular threshold by first counting the number of true and false matches and match failures using the following definitions:
  - TP - true positives (number of correct matches)
  - FN - false negatives (matches that were not correctly detected)
  - FP - false positives (proposed matches that are incorrect)
  - TN - true negatives (non-matches that were correctly rejected)

Matching Strategy and Error Rates

	True matches	True non-matches		
Predicted matches	TP = 18	FP = 4	P' = 22	PPV = 0.82
Predicted non-matches	FN = 2	TN = 76	N' = 78	
	P = 20	N = 80	Total = 100	
	TPR = 0.90	FPR = 0.05		ACC = 0.94

- The number of matches correctly and incorrectly estimated by a feature matching algorithm can be tabulated in a **confusion matrix** (contingency table)

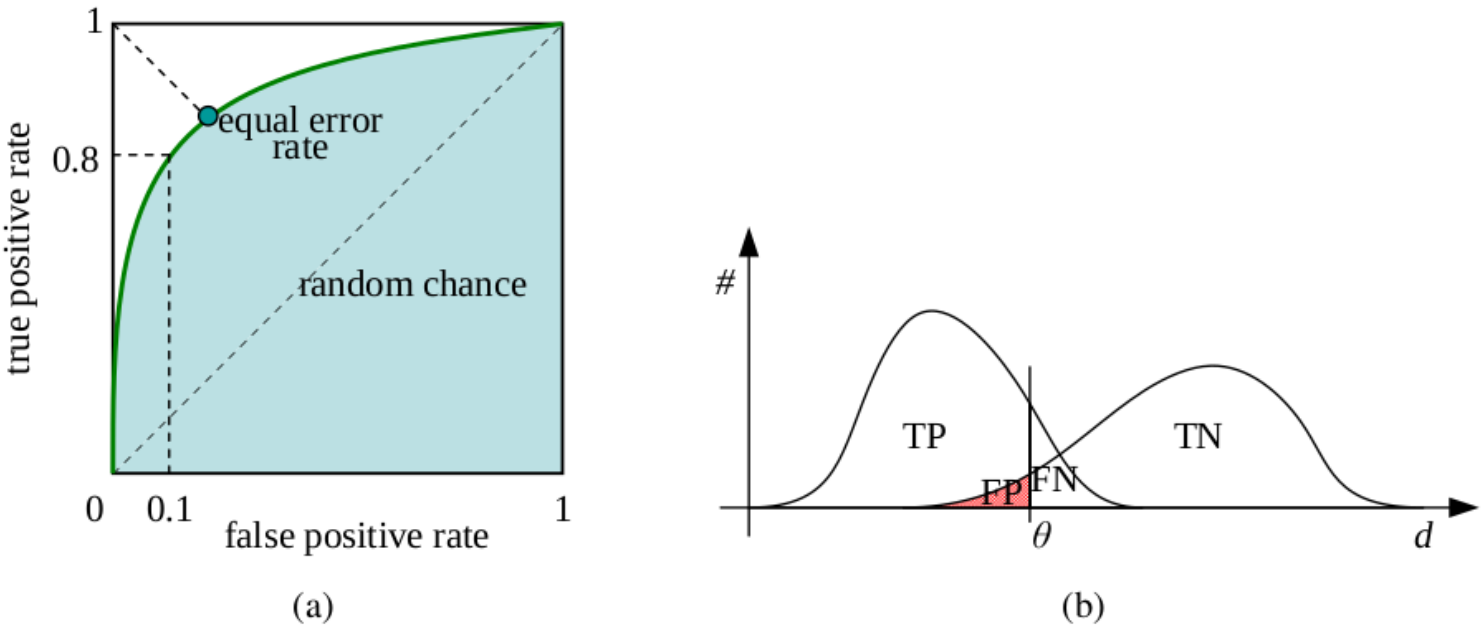
Matching Strategy and Error Rates

- We can convert these numbers into unit rates by defining the following quantities:
  - true positive rate, $\text{TPR} = \frac{\text{TP}}{\text{TP+FN}} = \frac{\text{TP}}{\text{P}}$
  - false positive rate, $\text{FPR} = \frac{\text{FP}}{\text{FP+TN}} = \frac{\text{FP}}{\text{N}}$
  - positive predictive value, $\text{PPV} = \frac{\text{TP}}{\text{TP+FP}} = \frac{\text{TP}}{\text{P}}$
  - accuracy, $\text{ACC} = \frac{\text{TP+TN}}{\text{P+N}}$

Matching Strategy and Error Rates

- Any particular matching strategy can be rated by the TPR and FPR numbers
- Ideally, the true positive rate will be close to 1 and the false positive rate close to 0
- As we vary the matching threshold, we obtain a family of such points which are collectively known as the **receiver operating characteristic** (ROC) curve

Matching Strategy and Error Rates



- (a) the ROC curve plots the true positive rate against the false positive rate for a particular combination of feature extraction and matching algorithms
- (b) the distribution of positives (matches) and negatives (non-matches) as a function of inter-feature distance d

Efficient Matching

- Once we have decided on a matching strategy, we still need to efficiently search for potential candidates
- This can be done by devising an **indexing structure**, such as a multi-dimensional search tree or a hash table, to rapidly search for features near a given feature

Efficient Matching

- An indexing structure can be built for each image independently (useful if we want to only consider certain potential matches, e.g. searching for a particular object)
- Alternatively, an indexing structure can be built globally for all the images in a given database, which can potentially be faster since it removes the need to iterate over each image

Feature Tracking

- An alternative to independently finding features in all candidate images and then matching them is to find a set of likely feature locations in the first image and then *search* for their corresponding locations in subsequent images
- This kind of *detect then track* approach is more widely used for video tracking applications where the amount of motion and appearance deformation between adjacent frames is expected to be small

Feature Tracking

- The process of selecting good features to track is closely related to selecting good features for more general recognition applications
- In practice, regions containing high gradients in both directions, i.e. which have high eigenvalues in the auto-correlation matrix, provide stable locations at which to find correspondences

Feature Tracking

- If features are being tracked over longer image sequences, their appearance can undergo larger changes
- We must then decide whether to continue matching against the originally detected patch (feature) or to re-sample each subsequent frame at the matching location
- The former strategy is prone to failure as the original patch can undergo appearance changes such as foreshortening
- The latter strategy runs the risk of the feature drifting from its original location to some other location in the image

Kanada-Lucas-Tomasi (KLT) Tracker

- A preferable solution to the tracking problem is to compare the original patch to later image locations using an *affine* motion model
- The **Kanada-Lucas-Tomasi** (KLT) tracker first compares patches in neighboring frames using a translational model
- The tracker then uses the location estimates produced by this model to initialize an affine registration between the patch in the current frame and the base frame where a feature was first detected

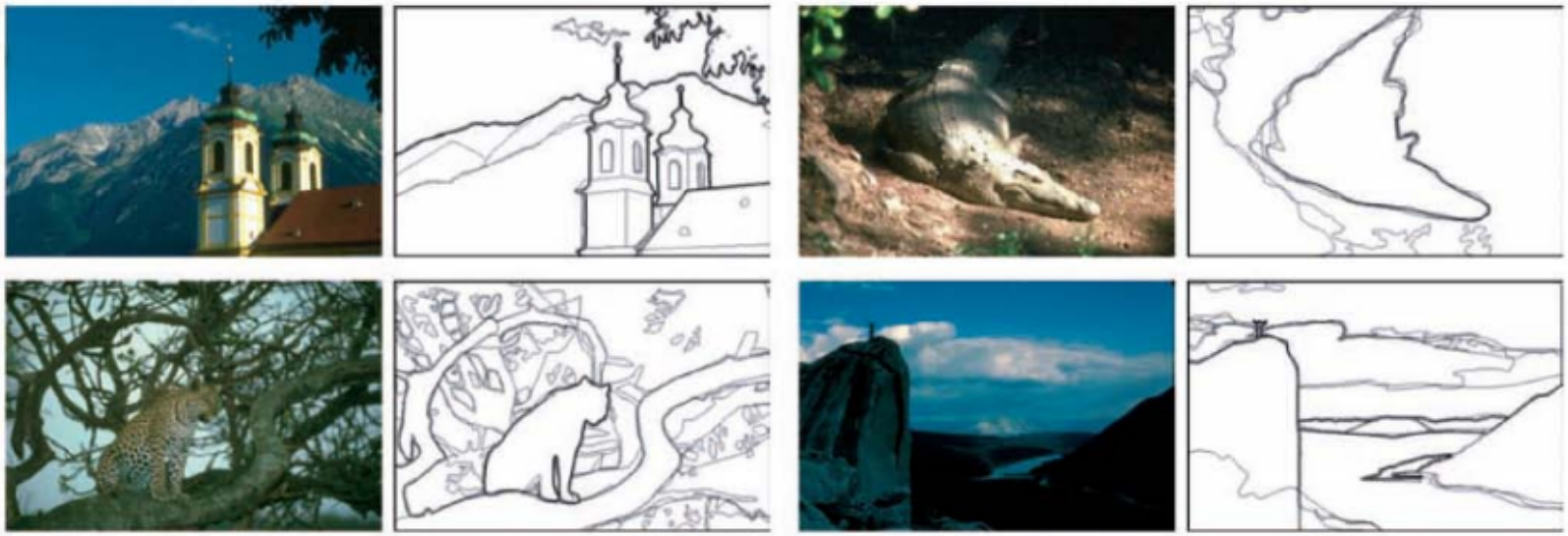
Review Question

- Compare and contrast local and global feature descriptors
- In what computer vision applications would it be more desirable to use a local rather than global descriptor, and vice versa?

Edge Features

- While interest points are useful for finding image locations that can be accurately matched 2D, edge points are far more plentiful and often carry semantic associations
- For example, the boundaries of objects, which also correspond to occlusion events in 3D, are usually delineated by visible contours

Edge Detection



- Given an image, how can we find the most "salient" or "strongest" edges or the object boundaries?

Edge Detection

- Qualitatively, edges occur at boundaries between regions of different color, intensity, or texture
- Unfortunately, segmenting an image into coherent regions is a difficult task
- Often, it is preferable to detect edges using only purely local information

Edge Detection

- Under such conditions, a reasonable approach is to define an edge as the location of **rapid intensity variation**
- If we think of an image as a height field, then on such a surface, edges occur at locations of **steep slopes**, or equivalently, in regions of closely packed contour lines (on a topographic map)

Edge Detection

- Mathematically, we define the slope and direction of a surface through its gradient

$$\mathbf{J}(\mathbf{x}) = \nabla I(\mathbf{x}) = (\frac{\partial I}{\partial x}\frac{\partial I}{\partial y})(\mathbf{x})$$

  where the local gradient vector **J** points in the direction of **steepest ascent** in the intensity function

- Its magnitude is an indication of the slope or strength of the variation, while its orientation points in a direction *perpendicular* to the local contour

Edge Detection

- Unfortunately, taking image derivatives accentuates high frequencies and hence amplifies noise since the proportion of noise to signal is larger at high frequencies
- Therefore, it is good practice to smooth the image with a low-pass filter prior to computing the gradient
- Because we would like the response of our edge detector to be independent of orientation, a circularly symmetric smoothing filter is desirable

Edge Detection

- Since differentiation is a linear operation, it commutes with other linear filtering operations
- The gradient of the smoothed image can therefore be written as

$$\mathbf{J}_\sigma(\mathbf{x}) = \nabla[G_\sigma(\mathbf{x}) * I(\mathbf{x})] = [\nabla G_\sigma](\mathbf{x}) * I(\mathbf{x})$$

i.e. we can convolve the image with the horizontal and vertical derivatives of the Gaussian kernel function

$$\nabla G_\sigma(\mathbf{x}) = (\frac{\partial G_\sigma}{\partial x}\frac{\partial G_\sigma}{\partial y})(\mathbf{x}) = [-x - y]\frac{1}{\sigma^3}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

where σ indicates the width of the Gaussian

Edge Detection

- For many applications, we want to thin such a continuous gradient image to only return isolated edges, i.e. as single pixels at discrete locations along the edge contours
- This can be achieved by looking for *maxima* in the edge strength (gradient magnitude) in a direction *perpendicular* to the edge orientation, i.e. along the gradient direction

Edge Detection

- Finding this maximum corresponds to taking a directional derivative of the strength field in the direction of the gradient and then looking for zero crossings
- The desired directional derivative is equivalent to the dot product between a second gradient operator and the results of the first

$$S_\sigma(\mathbf{x}) = \nabla \cdot \mathbf{J}_\sigma(\mathbf{x}) = [\nabla^2 G_\sigma](\mathbf{x}) * I(\mathbf{x})]$$

Edge Detection

- The gradient operator dot product with the gradient is called the **Laplacian**
- The convolution kernel

$$\nabla^2 G_\sigma(\mathbf{x}) = \frac{1}{\sigma^3}\left(2 - \frac{x^2+y^2}{2\sigma^2}\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

is therefore called the **Laplacian of Gaussian** (LoG) kernel

Edge Detection

- This kernel can be split into two separable parts

$$\nabla^2 G_\sigma(\mathbf{x}) = \frac{1}{\sigma^3}\left(1 - \frac{x^2}{2\sigma^2}\right) G_\sigma(x) G_\sigma(y) + \frac{1}{\sigma^3}\left(1 - \frac{y^2}{2\sigma^2}\right) G_\sigma(x) G_\sigma(y)$$

  which allows for a much more efficient implementation using separable filtering

- In practice, it is quite common to replace the LoG convolution with a DoG computation since the kernel shapes are qualitatively similar

Edge Detection

- Once we have computed the sign function $S(\mathbf{x})$, we must find its **zero crossings** and convert these into edge elements (**edgels**)
- An easy way to detect and represent zero crossings is to look for adjacent pixel locations $\mathbf{x}_i$ and $\mathbf{x}_j$ where the sign changes value, i.e. $[S(\mathbf{x}_i) > 0] \neq [S(\mathbf{x}_j) > 0]$
- The sub-pixel location of this crossing can be obtained by computing the "x-intercept" of the "line" connecting $S(\mathbf{x}_i)$ and $S(\mathbf{x}_j)$

$$\mathbf{x}_z = \frac{\mathbf{x}_i S(\mathbf{x}_j) - \mathbf{x}_j S(\mathbf{x}_i)}{S(\mathbf{x}_j) - S(\mathbf{x}_i)}$$

Edge Linking

- While isolated edges can be useful in a variety of applications, they become even more useful when linked into continuous contours
- If edges have been detected using zero crossings of some function, then linking them up is straightforward since adjacent edgels share common endpoints
- Linking edgels into chains involves sorting them and then picking an unlinked edgel and following its neighbors in both directions

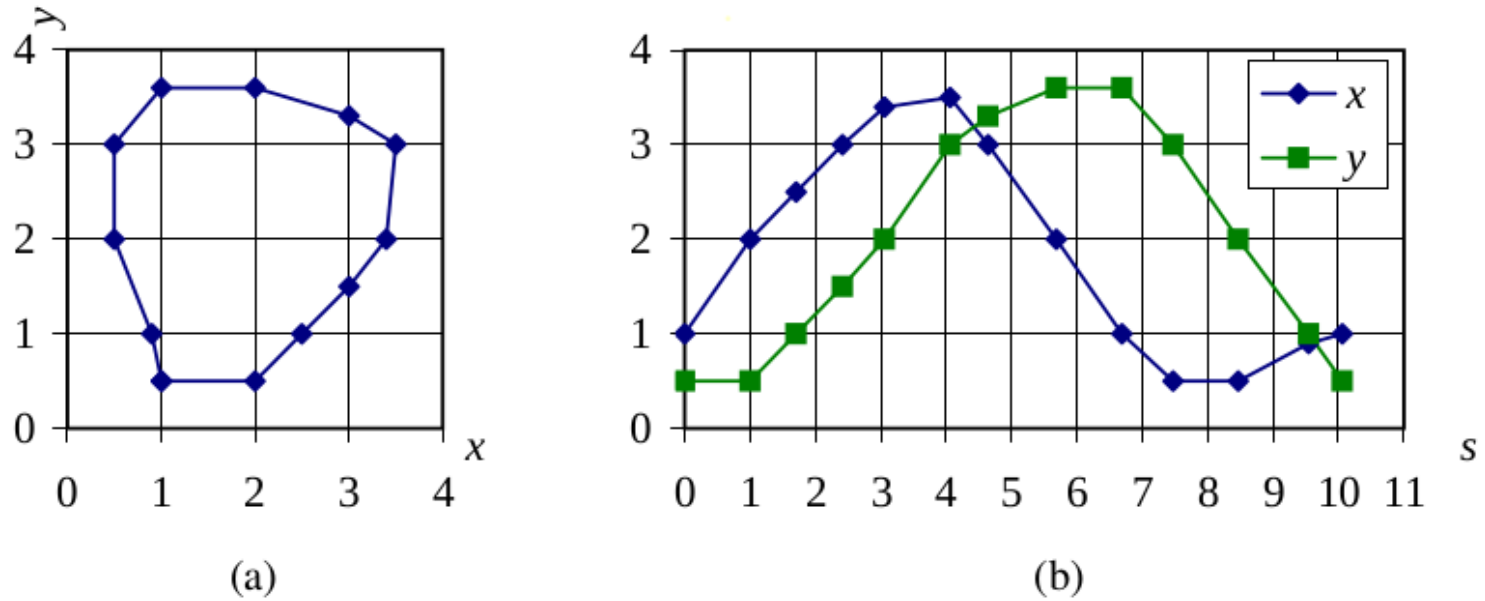
Edge Linking

- Once the edgels have been linked into chains, we can apply an optional thresholding with **hysteresis** to remove low-strength contour segments
- The basic idea of hysteresis is to set two different thresholds and allow a curve being tracked above the higher threshold to dip in strength down to the lower threshold

Edge Linking

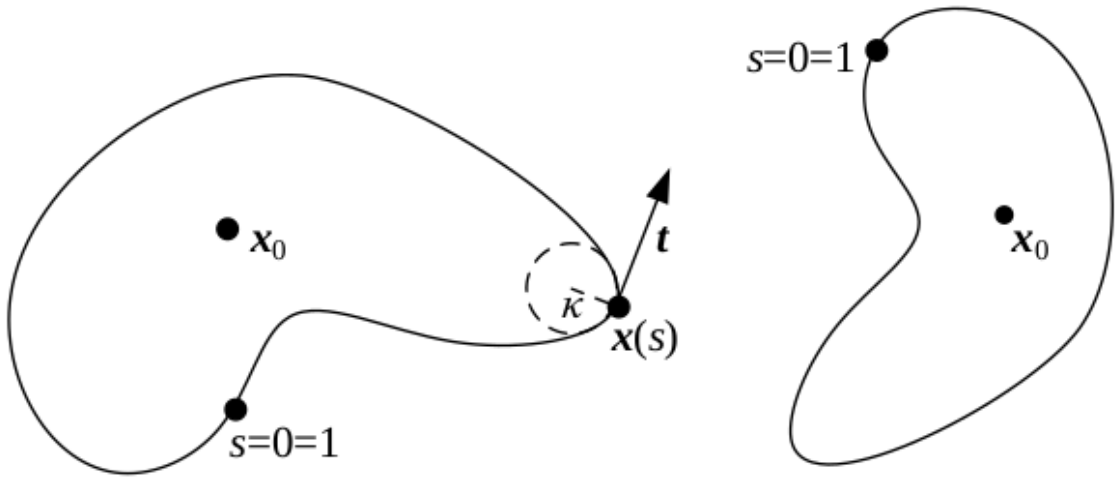
- Linked edgel lists can be encoded more compactly using a variety of alternative representations such as the **arc length parameterization** of a contour, $\mathbf{x}_s$, where s denotes the arc length along a curve
- The advantage of the arc-length parameterization is that it makes matching and processing (e.g. smoothing) operations much easier

Edge Linking



- Arc-length parameterization of a contour: (a) discrete points along the contour are first transcribed as (b) (x, y) pairs along the arc length s
- This curve can then be regularly re-sampled or converted into alternative (e.g. Fourier) representations

Edge Linking



- Matching two contours using their arc-length parameterization
- If both curves are normalized to unit length, $s \in [0, 1]$ and centered around their centroid $\mathbf{x}_0$, they will have the same descriptor up to an overall "temporal" shift (due to different starting points for $s = 0$) and a phase (x-y) shift (due to rotation)

Review Question

- What are the basic steps of the Canny edge detection algorithm?
- Using MATLAB, compare the results of the Sobel, Prewitt, Roberts, and Canny edge detectors on an image of your choice

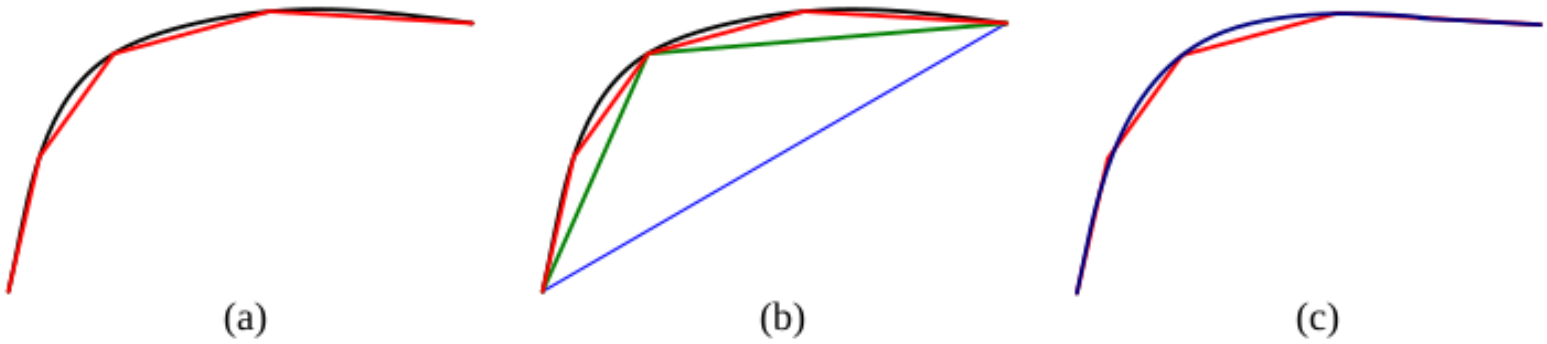
Detecting and Matching Lines

- Although edges and general curves are suitable for describing the contours of natural objects, our world is full of straight lines
- Detecting and matching these lines can be useful in a variety of computer vision applications

Successive Approximation

- In many applications we can approximate a curve as a piecewise-linear polyline or as a B-spline curve
- One simple technique is to recursively subdivide the curve at the point furthest away from the line joining the two endpoints
- Once the line simplification has been computed it can be used to approximate the original curve

Successive Approximation



- Approximating a curve (black) as a polyline or B-spline: (a) original curve and polyline approximation (red); (b) successive approximation by recursively finding points furthest away from the current approximation; (c) smooth interpolating spline (blue) fit to the polyline vertices

The Hough Transform

- While curve approximations with polylines can often lead to successful line extraction, lines in the real world are sometimes broken up into disconnected components or made up of many collinear line segments
- In many cases it is desirable to group such collinear segments into extended lines

The Hough Transform

- The **Hough transform** is a well-known technique for having edges "vote" for plausible line locations
- This is done by using the local information at each edgel to vote for a *single* accumulator cell

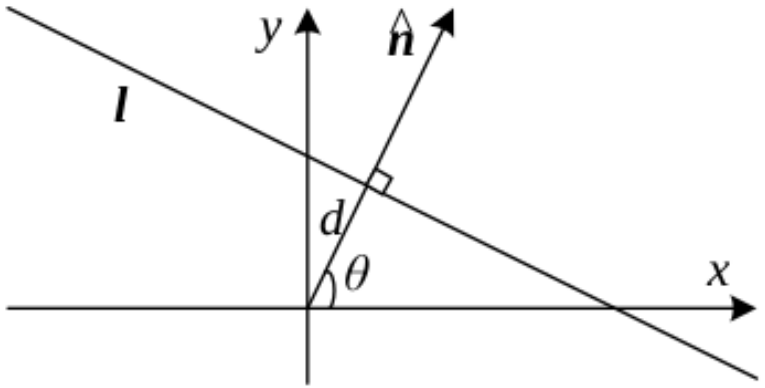
The Hough Transform

- To vote for line hypotheses, we use the 2D polar (θ, d) representation for lines and the normal-distance $(\hat{\mathbf{n}}, d)$ parameterization where

$$\theta = \tan^{-1} n_y / n_x$$

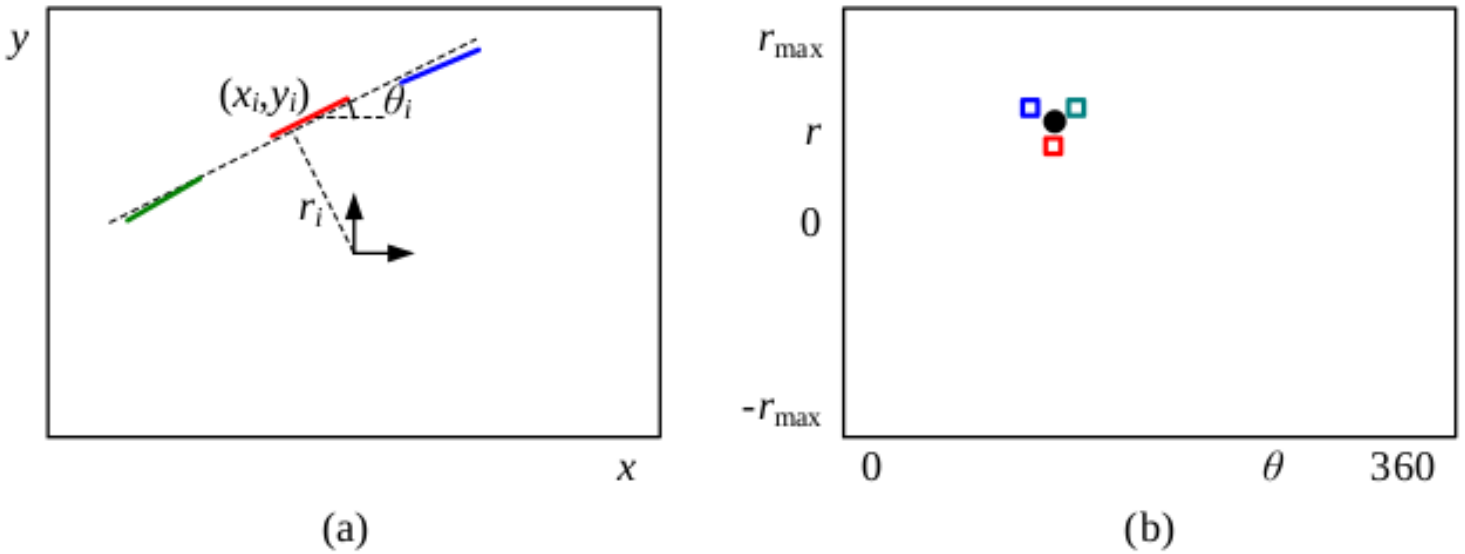
- The range of possible (θ, d) values is $[-180°, 180°] \times [-\sqrt{2}, \sqrt{2}]$ assuming that we are using normalized pixel coordinates that lie in $[-1, 1]$

The Hough Transform



- 2D line equation expressed in terms of the normal $\hat{\mathbf{n}}$ and distance to the origin d

The Hough Transform



- Oriented Hough transform: (a) an edgel re-parameterization in polar (r, θ) coordinates, with $\hat{\mathbf{n}}_i = (\cos\theta_i, \sin\theta_i)$ and $r_i = \hat{\mathbf{n}}_i \cdot \mathbf{x}_i$; (b) (r, θ) accumulator array, showing the votes for the three edgels marked in red, green, and blue

The Hough Transform

procedure *Hough*$(\{(x, y, \theta)\})$:

1. Clear the accumulator array.
2. For each detected edgel at location (x, y) and orientation $\theta = \tan^{-1} n_y/n_x$, compute the value of
$$d = x\, n_x + y\, n_y$$
and increment the accumulator corresponding to (θ, d).
3. Find the peaks in the accumulator corresponding to lines.
4. Optionally re-fit the lines to the constituent edgels.

- Outline of the Hough transform algorithm based on oriented edge segments

The Hough Transform

- The number of bins to use along each axis depends on the accuracy of the position and orientation estimate available at each edgel and the expected line density
- This parameter is best set experimentally with some test runs on sample imagery

Review Question

- How can the Hough transform be generalized to look for other geometric features such as ellipses?

Summary

- Detecting and matching features is a significant part of many computer vision systems
- We've explored some practical approaches to detecting features and also discussed how feature correspondences can be established across different images