CSci 4270 and 6270
Spring 2025
Lectures 7 and 8: Local Feature Extraction and Description
Thursday, January 30 and Monday February 3 (after the quiz)

Overview

- "Features" and "feature extraction" are fundamental to computer vision.
- Local vs. global:
  - Could be "local" to small image regions
  - Could be "global", image-wide summaries
- Today's focus: local "hand-crafted" features
- Why?
  - Illustrate key issues and introduce widely-used techniques
  - Tool in fun problems in HW 3
- We'll study global features (and perhaps some other local ones) later in the semester.
- Scheduling note: we will skip over some of the details of these lectures to stay on track with the syllabus. What we cover in class and study in homework assignments and practice problems will determine what you are responsible for knowing.
- There will be no separate Lecture 8 handout.

Local Features: Edges

- Edge elements — "edgels"
- Goal: extract image locations of sharpest intensity change.
  - Intuitively, edges are "significant events" in images.
  - This is a "boundary first" approach to image analysis, whereas the problem area known as *segmentation* is a "region first" approach.
- A single pixel edge is called an *edgel*, short for "edge element".
- Steps of edge(l) detection
  - Smoothing (Gaussian)
  - Differentiation, gradient magnitude and direction
    - Some methods use second order derivatives, but we will not.
  - Non-maximum suppression, sub-pixel interpolation, thresholding.
  - Linking into edgel chains

Differential Operations

- Directional derivatives and gradiaent magnitude and direction calculation.n
- Let I_s be the smoothed image.
- Compute $I_x = \partial I_s/\partial x$ and $I_y = \partial I_s/\partial y$ using discrete differentiation (convolution).
- Then we can compute the gradient direction and magnitude as

$$I_m(x,y) = \|I_x(x,y)^2 + I_y(x,y)^2\|^{1/2}$$
$$I_\theta(x,y) = \text{atan2}(I_y(x,y), I_x(x,y))$$

  - Notice with atan2 the y derivative is first.

- The direction is "normal" to the direction of the edge contour and points in the direction of most rapid intensity increase.

Other Derivative Convolution Kernels

(Sometimes these are called "operators".)

- As we've discussed, other partial derivative operators have been used. For example, the Sobel and Prewitt $\partial f/\partial x$ kernels are

$$\begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{pmatrix}$$

  respectively.

- Notice that there is implicit smoothing in these operators, and the magnitudes of the derivatives are scaled up.

Additional Steps are Needed Following Smoothing and Gradient Computation

- After gradient computation, we still just have an image.
- Pure thresholding leaves us with pixel locations of strong gradients, not edges (edge elements).
- We want to "extract" the edges, both individually and in contours.
- Decisions about what is a true edge and what is not are made in groups of edgels.
- We'll sketch a picture in class to illustrate this.

Non-Maximum Suppression

- For each pixel, suppress it as a "non-edge" if a neighboring pixel has a stronger gradient.
- We'll start with a 1-d image where the gradients are shown in the following (with the column index above the boxes and the gradient magnitude in the boxes):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	6	7	8	11	15	18	17	8	6	4	8	10	8	4

  and demonstrate the importance of non-maximum suppression and not recording the values "in-place" in the array.

- In 2d it is important to think about what should should remain after non-maximum suppression.
- We will focus our discussion on the following example where the gradient magnitudes are shown as integers (assume blanks boxes have gradient 0):

	0	1	2	3	4	5	6	7
0						85	94	71
1					78	100	96	
2				55	82	98	80	
3			49	80	102	99	65	
4			71	100	104	71		
5		72	90	89	68			
6	73	91	92	76				
7	80	99	80	45				

- Discussion:
  - What happens if we don't consider gradient direction during non-maximum suppression?
  - How to use gradient direction?
  - Discretization effects must be considered here, since the determination of who is a "neighbor" may not be symmetric.
- Finally, candidate edgels are the pixels that are left after non-maximum suppression.
  - This can include locations with very low gradient magnitudes!

Peak Finding / Interpolation: Sub-Pixel Localization of Edges

Still only know pixel locations of peaks, so...

- At each surviving peak, fit a parabola to the gradient magnitudes along the gradient direction.
- Offset the edge location along the normal based on the location of the peak.
- Yields the subpixel edge location
- While fitting a parabola sounds like an expensive computation, we will see in class that it reduces to straightforward arithmetic.
- Per usual we will start with 1d and then consider 2d.

Edge Linking

Group edgels into chains:

- Tangent direction is 90° rotated from the gradient direction.
- Record "next" and "previous" edgels at each edge (think of a doubly-linked list) — the edges "ahead" and "behind" the current one.
- Resolving ambiguities:
  - Find candidates for linking based on consistency of position **and** orientation.
  - If there are two candidates ahead (or behind) and they are consistent with each other, choose the four-connected neighbor first
  - If they aren't consistent, then this is a branching location. We can either
    - Terminate the chain or
    - Attempt to find the stronger link and continue it.
- Result is a series of edgel chains.

Edge Thresholding

We still haven't decided which points are really "edgels", so we need to apply a threshold:

- Could use a single global threshold, perhaps computed from the image gradient statistics.
  - For example, assume the edgels occupy no more than a certain percentage of the image and use the histogram to compute a threshold.
- Could compute different thresholds in different (but overlapping) regions, perhaps interpolating between them to generate (potentially) different thresholds at each pixel.
- Could apply a threshold based on the average gradient magnitude of an edgel chain.
- Many other ideas.

Examples

Using Python and OpenCV, we will examine edge detection results on a number of images and consider how well it works.

Summary – Edgel Detection

- The "Canny" edge detection method, which we summarized here, is often used as a baseline method.
- Techniques used here are needed in many guises throughout computer vision:
  - Smoothing and differentiation
  - Interpolation / subpixel localization
  - Non-maximum suppression
  - Thresholding

Interest Point / Keypoint Extraction: Motivating Example

- How do we recognize that three of these pictures show the same thing (the front of my house) and the other two show something different?

- We need to *describe* each image in a way that allows them to be matched (or not), despite differences in
  - Viewpoint
  - Content
  - Orientation
  - Scale
- To do this, we extract **and** describe (as a vector) features in a way that is as invariant as possible to these differences

Specific Goals

- Corners: between edgels and lines
  - Also called *interest points* or *keypoints*.
- Want to find the same regions repeatedly in multiple images as the basis for matching the images:
  - If we find the same regions, the images match. If we don't, the images do not match.
- Therefore, we need:
  - A way of extracting repeatable regions — centered at interest points.
  - A way of describing the regions in an invariant way so that we can match them.

Topics

- Detection:
  - Harris and its variations
  - Laplacian
  - Hessian
- Orientation and scale
- Description

Starting intuition: Find the region where the intensity is varying significantly in multiple directions.

Harris Measure

- Intensity gradient captures the intensity variation at a pixel
- Look for locations where the intensity gradient direction (where the gradient itself is significant) is varying substantially in multiple directions.
- Start by computing the intensity derivatives at each point in an image.
  - Call the image $\mathbf{I}$ and the derivative images $\mathbf{I}_x$ and $\mathbf{I}_y$.
  - As usual, we apply Gaussian smoothing first, with std deviation σ.
- Then we compute the outer product of the gradient vectors at each pixel, so that we effectively have a matrix at each pixel location:

$$\begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \tag{1}$$

- Finally, we convolve this matrix (one entry at a time) with another Gaussian to produce the weighted average of the gradient outer products over a neighborhood. This value is computed at every point:

$$\mathbf{H}(x,y) = \sum_{(u,v)\in\mathcal{N}(x,y)} w(u-x, v-y) \begin{pmatrix} I_x^2(u,v) & I_x(u,v)I_y(u,v) \\ I_x(u,v)I_y(u,v) & I_y^2(u,v) \end{pmatrix} \tag{2}$$

  Notes:

  - $w(i,j)$ is the Gaussian weight function.
  - $\mathcal{N}(x,y)$ is a neighborhood of pixels centers at (x,y), usually determined by the σ of the Gaussian function.

- At each pixel location (x, y), we then compute the eigenvalues of $\mathbf{H}(x, y)$.
  - Call these λ_1 and λ_2, with the convention that $\lambda_1 \geq \lambda_2 \geq 0$.
- We will think about types of image neighborhoods that cause these values to be lower or higher...

- The Harris corner measure is

$$h(x, y) = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 \tag{3}$$

  where κ is a tunable parameter, usually in the range of about $[0.004 \ldots 0.015]$, but it can be much lowere

- The Harris measure can be computed from just the determinant and trace, avoiding the square root calculation necessary to compute the eigenvalues:

$$h(x, y) = \det \mathbf{H}(x, y) - \kappa \operatorname{trace}^2 \mathbf{H}(x, y) \tag{4}$$

- Others use simpler measures such as
  - $\det \mathbf{H}(x, y)$
  - $\lambda_2(x, y)$ (the smaller of the two eigenvalues)
- Locations become Harris corners:
  - The Harris measure is above a threshold — determined by statistics on the Harris responses.
  - No other higher values of the measure are found over an area of a user-specified width. This is a form of *non-maximum suppression.*

Demo

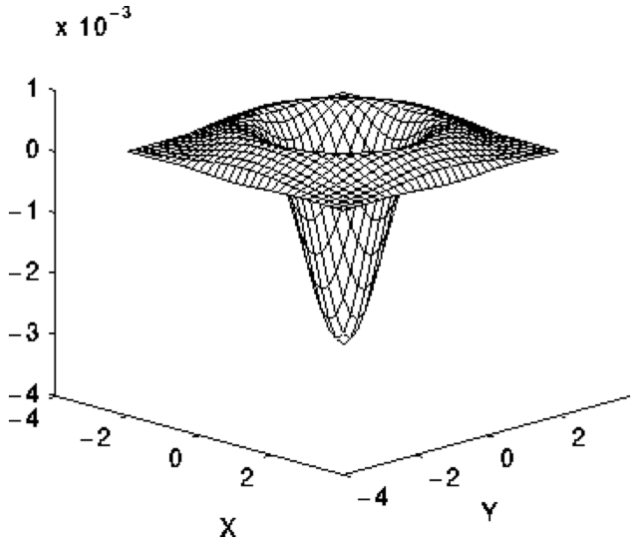
- At this point we will switch to a Jupyter notebook demo
- One important feature is the use of dilation to achieve a form of non-maximum suppression:

Laplacian

- Second derivative operator
- Equation:

$$\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{5}$$

- Plot:



- Discrete version is the simple convolution, which we will derive in class:

	-1	0	1
-1	0	1	0
0	1	-4	1
1	0	1	0

- Of course, we apply Gaussian smoothing first.
- Yields a scalar value at each pixel, with negative values as well as positive.
- Detection therefore becomes a problem of finding negative minimum and positive maxima.

Hessian

- The Hessian uses the second derivative of intensity in place of the outer products of the first derivative to produce a 2x2 matrix at each pixel:

$$\mathbf{H}(x,y) = \sum_{(u,v)in\mathcal{N}(x,y)} w(u-x, v-y) \begin{pmatrix} \frac{\partial^2 I}{\partial x^2}(u,v) & \frac{\partial^2 I}{\partial x \partial y}(u,v) \\ \frac{\partial^2 I}{\partial x \partial y}(u,v) & \frac{\partial^2 I}{\partial y^2}(u,v) \end{pmatrix} \quad (6)$$

  where w is the Gaussian smoothing function.

  - Note that the superscript 2 above indicates the 2nd partial derivative operation instead of a squaring operation.

- We will look at the discrete form of these second partial derivatives in class.

- Like the Harris measure, various combinations of the determinant, the trace and the eigenvalues are used to generate a final measure at each point, but like the Laplacian, detection should be considered at both positive and negative extrema.

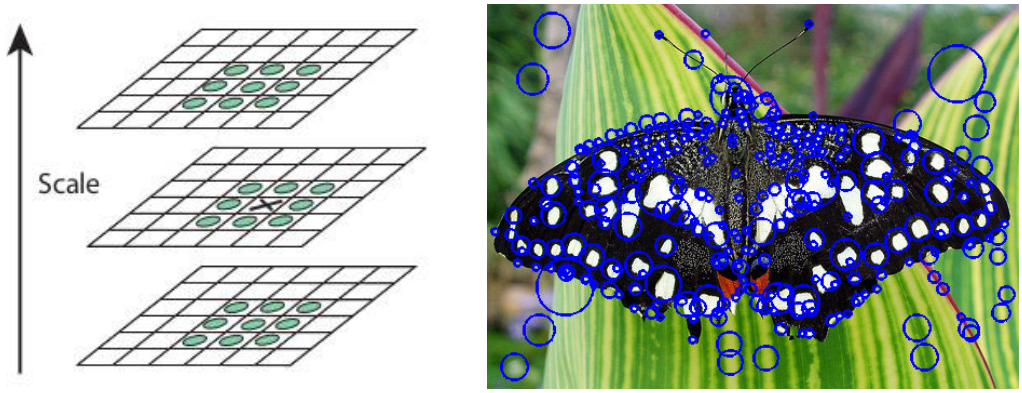


Figure 1: Left: In order for a point to be accepted as a keypoint it must be a local maximum in image coordinates and across scales and therefore it must be compared to 26 neighbors. Right: an example of keypoints of different sizes detected on an image (and shown on top of the original color image even though the keypoints were detected in the grayscale image.)

Scale Space

- By increasing σ used in the Gaussian smoothing we can make the keypoint detectors respond to larger and larger image features.
- For example, we can prove that for a disk of radius r we will achieve the strongest response when the σ of Gaussian smoothing is exactly $\sigma = r$.
- Therefore, we
  - Smooth the images at a relatively dense sampling of scales, downsampling at scale multiples of 2
  - Apply non-maximum suppression in three dimensions — two in image dimensions and one in the scale dimension.
- Sometimes we need to be careful to normalize derivatives across scales.

Subpixel Interpolation

- Two methods:
  - Fit a parabolic function in three dimensions — we looked at just a one-dimension version of this in our class on edge detection — and find the offset and peak value.
  - Look at the tangent directions of all the nearby edgels (or just look at gradient-weighted pixel locations) in the region around the keypoint and find the new keypoint location point that is closest to the lines.
- Both can require iterative re-estimation.

Description

- We want to "describe" a keypoint so that we can match with reasonably high probability in another image.
- Simplest idea is to just extract a square pixel neighborhood and compare it to keypoints found in a second image — e.g. compute the sum-squared difference of the intensities in the two neighborhoods.
  - Only works for small changes between images, such as in tracking
- Need to describe the region around a keypoint in a more "invariant" fashion.

- We start by extracting the information needed to define a similarity transformation between two keypoints:
  - Keypoint location and feature detection scale give us 3 out of the 4 needed pieces of information.
  - The 4th comes from finding the dominant orientation.
- Using this we could center, rotate and translate the pixels in a neighbor around the two keypoints we'd like to match and then we could compare intensities.

- We need something more if we need
  - Invariance to illumination changes too!
  - Fast search that does not require direction comparison between the neighborhoods surrounding each keypoint.
- Breakthrough came in Lowe's 2004 *International Journal of Computer Vision* paper: the Scale-Invariant Feature Transform (SIFT):
  - Describe each region in a 128-component vector
  - Fast nearest-neighbor search to find closest vector(s)
- Refer to this paper for many of the following details.
- Throughout our discussion we will use oriented and scaled copies of the center region of the letter **M**....

First Step: Direction at Keypoint from Gradient Histogram

At each keypoint:

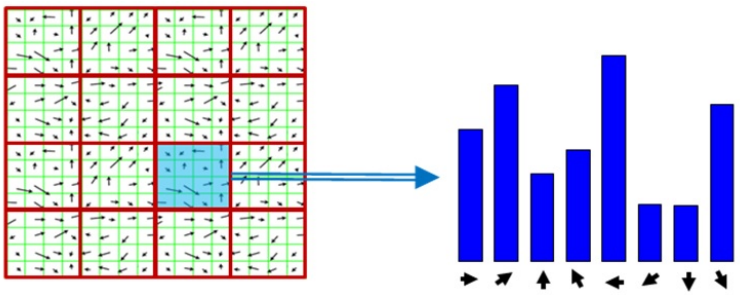
- Form a histogram of gradient directions, in increments of 10 degrees
- For each pixel in a neighborhood around the keypoint:
  - Convert its gradient direction to an angle θ using atan2.
  - Compute a weight, w, which is the product of (a) the value of the Gaussian function (yet again!) computed from the distance between the pixel location and the keypoint location, and (b) the gradient magnitude.
  - Split the weight w between the two closest orientation histogram bins to θ, so that, for example,
    - if θ is half-way between the centers of the two closest bins, each bin gets half of its weight.
    - if θ is 1/4 of the way between its closest bin center and its second closest, then the closest center gets 3/4 of the vote and the second closest gets 1/4.

    This is what's called "linear interpolation".

- Smooth the resulting histogram.
- Find the peak, and interpolate (just like sub-pixel location estimation during edge detection) to find the peak orientation and the peak histogram value.
  - Need to be correctly handle wrap-around of orientation, i.e. $0^\circ == 360^\circ$
- Form a keypoint from the location, orientation and scale.
- If there is another orientation peak whose height is within 80% of the first peak, form a separate keypoint using this (interpolated, of course) orientation.

SIFT Descriptor

- Lay down a 4x4 grid, centered on a keypoint, oriented with the x-axis along the keypoint direction.
  - Each square of the grid is 2σ on a side, where σ is the detection scale.
- Form an 8-component orientation histogram (45° increments) in each square of the grid.
- Here is an illustration with a horizontal orientation, showing the histogram in one bin:



- Each pixel falling within the 4x4 grid in the image votes into 8 locations in the histograms.
- In particular, it finds its four nearest grid locations, and its two nearest bins in orientation.
  - Denote the indices of the two nearest bin centers in x, by j and $j+1$ in y by i and $i+1$ (with $0 \leq i, j < 3$)
  - Denote the two nearest histogram bins centers in orientation by k and $k+1$ (with $0 \leq k < 7$).

- Then, each pixel votes just like it did for the orientation histogram above, except now its vote is split across 8 bins — four grid bins times two orientations for each grid bin.
  - This is done using linear interpolation in i, in j and in k and then multiplying them together to produce what is called *trilinear interpolation*.
- Notes:
  - Gaussian weighting is applied again so that points that are within the grid but are far from the keypoint center get a weaker vote and therefore less influence.
  - In general, since the keypoint orientation is not horizontal, the pixels and their gradient directions need to be rotated into the keypoint coordinate system.

- After all voting, the 16 orientation histograms are concatentated into a 128-component vector
- The vector's magnitude is normalized, which effectively makes it invariant to linear changes in intensity.
  - As a heuristic, any intensity value above 0.3 after this normalization is truncated at 0.3 and the vector is renormalized.
- The interpolation across 8 histogram entries for each pixel gives the SIFT descriptor some invariance to shifts in position, orientation and scale.
  - More advanced methods have been divised to estimate an elliptic region rather than just use a circle, giving some affine invariance.

Use in Matching

- For each image extract all keypoint and descriptors across the scales,
  - Usually 2,000 - 4,000 per image
- When matching between images, for each keypoint in one image, find its **two closest** descriptors in the other
  - Approximate nearest neighbors search techniques can make this very fast
- If d_1 is the distance (between descriptor vectors) to the closest descriptor and d_2 is the distance to the second closest, then this is considered a good match if

$$\frac{d_1}{d_2} < 0.8 \tag{7}$$

- Images that show the same object or scene under reasonably similar lighting conditions will produce a significant number of matches — 10-20 for hard cases, 50-100 or more for easier cases.
- Images that do not show the same object will have few matches.
- We will explore use of this matching in several contexts.

Summary

- Corners, interest points, descriptors: peak locations of strong intensity varation in two dimensions
- Localized in position and scale
- Oriented using a gradient histogram
- Summarized using SIFT descriptor
- While other descriptors with similar properties were proposed in the late 1990's and early 2000's, the SIFT descriptor had the largest impact, producing a huge change in the field of computer vision.
  - The conference and journal versions of David Lowe's SIFT paper have been cited about 100,000 times!