Descriptors

EECS 442 – David Fouhey
Winter 2022, University of Michigan

https://web.eecs.umich.edu/~fouhey/teaching/EECS442_W22/
Recap: Motivation

1: find corners+features

Image credit: M. Brown
Last Time – Gradients

Image gradients – treat image like function of x, y – gives edges, corners, etc.

\[ \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x}, 0 \end{bmatrix} \]  
\[ \nabla f = \begin{bmatrix} 0, \frac{\partial f}{\partial y} \end{bmatrix} \]  
\[ \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \end{bmatrix} \]

Figure credit: S. Seitz
Last Time – Corner Detection

Can localize the location, or any shift $\rightarrow$ big intensity change.

“flat” region: no change in all directions

“edge”: no change along the edge direction

“corner”: significant change in all directions

Diagram credit: S. Lazebnik
Last Time – Corner Detection

Zoom-In at x, y

Window with and w/o Offset

“Window”
At x+u, y+v
Here: u=-2, v=-3

“Window”
At x, y
Last Time – Corner Detection

Zoom-In at \(x,y\)

Error (Sum Sqs) for \(u,v\) offset

\[
E(u, v) = \sum_{(x,y) \in W} (I[x + u, y + v] - I[x, y])^2
\]

\[
\left \begin{array}{c}
\begin{array}{c}
\end{array}
\end{array}\right )^2
\]
Last Time – Corner Detection

TL;DR: Taylor expansion for error $E(u,v)$. All terms in equation are sums of image gradients and in $M$

\[
M = \begin{bmatrix}
\sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\
\sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2
\end{bmatrix}
\]

$I_x = I_x$ at point $(x,y)$, $I_y = I_y$ at point $(x,y)$

Optional Directions
Amounts

Should know
Can compute at each pixel

\[
V^{-1} \begin{bmatrix}
\lambda_1 & 0 \\
0 & \lambda_2
\end{bmatrix} V
\]
Putting It Together

\[ R = \det(M) - \alpha \trace(M)^2 = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2 \]

**Remake of standard diagram from S. Lazebnik from original Harris paper.**
In Practice

1. Compute partial derivatives $I_x$, $I_y$ per pixel
2. Compute $\mathbf{M}$ at each pixel, using Gaussian weighting $w$

$$\mathbf{M} = \begin{bmatrix} \sum_{x,y\in W} w(x, y)I_x^2 & \sum_{x,y\in W} w(x, y)I_xI_y \\ \sum_{x,y\in W} w(x, y)I_xI_y & \sum_{x,y\in W} w(x, y)I_y^2 \end{bmatrix}$$


Slide credit: S. Lazebnik
In Practice

1. Compute partial derivatives $I_x$, $I_y$ per pixel
2. Compute $M$ at each pixel, using Gaussian weighting $w$
3. Compute response function $R$

$$R = \det(M) - \alpha \text{trace}(M)^2$$
$$= \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$


Slide credit: S. Lazebnik
Computing R

Slide credit: S. Lazebnik
Computing R

Slide credit: S. Lazebnik
In Practice

1. Compute partial derivatives I_x, I_y per pixel
2. Compute M at each pixel, using Gaussian weighting w
3. Compute response function R
4. Threshold R


Slide credit: S. Lazebnik
In Practice

1. Compute partial derivatives $I_x$, $I_y$ per pixel
2. Compute $M$ at each pixel, using Gaussian weighting $w$
3. Compute response function $R$
4. Threshold $R$
5. Take only local maxima (called non-maxima suppression)


Slide credit: S. Lazebnik
Thresholded
Final Results
Desirable Properties

If our detectors are repeatable, they should be:

- **Invariant** to some things: image is transformed and corners remain the same

- **Covariant/equivariant** with some things: image is transformed and corners transform with it.

Slide credit: S. Lazebnik
Recall Motivating Problem

Images may be different in lighting and geometry
Affine Intensity Change

\[ I_{\text{new}} = aI_{\text{old}} + b \]

\( M \) only depends on derivatives, so \( b \) is irrelevant

But \( a \) scales derivatives and there’s a threshold

Partially invariant to affine intensity changes
All done with convolution. Convolution is translation equivariant.

Equivariant with translation
Rotations just cause the corner rotation matrix to change. Eigenvalues remain the same.

Equivariant with rotation

Slide credit: S. Lazebnik
Image Scaling

One pixel can become many pixels and vice-versa.

Not equivariant with scaling

How do we fix this?

Slide credit: S. Lazebnik
Recap: Motivation

1: find corners+features
2: match based on local image data
How?

Image credit: M. Brown
Today

• Fixing scaling by making detectors in both location **and scale**
• Enabling matching between features by **describing regions**
Key Idea: Scale Space
Left to right: each image is half-sized
Upsampled with big pixels below

Note: I’m also slightly blurring to prevent aliasing (https://en.wikipedia.org/wiki/Aliasing)
Key Idea: Scale Space
Left to right: each image is half-sized
If I apply a KxK filter, how much of the original image does it see in each image?

1/2 → 1/2 → 1/2

Note: I’m also slightly blurring to prevent aliasing (https://en.wikipedia.org/wiki/Aliasing)
Solution to Scales

Try them all!

Harris Detection

Harris Detection

Harris Detection

Harris Detection

See: Multi-Image Matching using Multi-Scale Oriented Patches, Brown et al. CVPR 2005
Blob Detection

Another detector (has some nice properties)

Find maxima and minima of blob filter response in scale and space

Slide credit: N. Snavely
Gaussian Derivatives

Gaussian

\[
\begin{align*}
\frac{\partial}{\partial y} g \\
\frac{\partial}{\partial x} g \\
\frac{\partial^2}{\partial^2 y} g \\
\frac{\partial^2}{\partial^2 x} g
\end{align*}
\]
Laplacian of Gaussian (LoG)

\[
\frac{\partial^2}{\partial^2 y} \frac{g}{g} + \frac{\partial^2}{\partial^2 x} \frac{g}{g}
\]

Slight detail: for technical reasons, you need to scale the Laplacian of Gaussian if you want to compare across sigmas.

\[
\nabla^2_{norm} = \sigma^2\left(\frac{\partial^2}{\partial x^2} g + \frac{\partial^2}{\partial y^2} g\right)
\]
Edge Detection with LoG

\[ f \]

\[ \frac{\partial^2}{\partial^2 x} g \]

\[ f \ast \frac{\partial^2}{\partial^2 x} g \]

Edge = Zero-crossing

Modern remake of classic S. Seitz slide
Edge Detection with LoG

Modern remake of classic S. Seitz slide
Edge Detection with LoG

What happens if we make input 1 unit wide?

$f * \frac{\partial^2}{\partial^2 x} g = \text{Zero-crossings}$

Modern remake of classic S. Seitz slide
Edge Detection with LoG

Modern remake of classic S. Seitz slide
Scale Selection

Given binary circle and Laplacian filter of scale $\sigma$, we can compute the response as a function of the scale.

<table>
<thead>
<tr>
<th>Image</th>
<th>$\sigma = 2$</th>
<th>$\sigma = 6$</th>
<th>$\sigma = 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius: 8</td>
<td>R: 0.02</td>
<td>R: 2.9</td>
<td>R: 1.8</td>
</tr>
</tbody>
</table>
Characteristic Scale

Characteristic scale of a blob is the scale that produces the maximum response.

Image  

Abs. Response

---

Scale-space blob detector

1. Convolve image with scale-normalized Laplacian at several scales
Scale-space blob detector: Example
Scale-space blob detector: Example

\[
\text{sigma} = 11.9912
\]

Slide credit: S. Lazebnik
Scale-space blob detector

1. Convolve image with scale-normalized Laplacian at several scales
2. Find maxima of squared Laplacian response in scale-space

Slide credit: S. Lazebnik
Finding Maxima

Point i,j is maxima (minima if you flip sign) in image I if it’s bigger than all neighbors

for y=range(i-1,i+1+1):
    for x in range(j-1,j+1+1):
        if y == i and x == j: continue
        #below has to be true
        $I[y,x] < I[i,j]$
Scale Space

Blue lines are image-space neighbors (should be just one pixel over but that’s impossible to draw)

Image
Radius: 8

$\sigma = 2$
R: 0.02

$\sigma = 6$
R: 2.9

$\sigma = 10$
R: 1.8
Scale Space

Red lines are the scale-space neighbors

Image
Radius: 8

\( \sigma = 2 \)
R: 0.02

\( \sigma = 6 \)
R: 2.9

\( \sigma = 10 \)
R: 1.8
Finding Maxima

Suppose \( I[:, :, k] \) is image at scale \( k \). Point \( i, j, k \) is maxima (minima if you flip sign) in image \( I \) if:

for \( y = \text{range}(i-1, i+1+1) \):
  for \( x \) in \( \text{range}(j-1, j+1+1) \):
    for \( c \) in \( \text{range}(k-1, k+1+1) \):
      if \( y == i \) and \( x == j \) and \( c == k \):
        continue
      #below has to be true
      \( I[y, x, c] < I[i, j, k] \)
Scale-space blob detector: Example

Slide credit: S. Lazebnik
Efficient implementation

- Approximating the Laplacian with a difference of Gaussians:

\[
L = \sigma^2 \left( G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)
\]

(Laplacian)

\[
DoG = G(x, y, k\sigma) - G(x, y, \sigma)
\]

(Difference of Gaussians)
Efficient implementation


Slide credit: S. Lazebnik
Problem 1 Solved

• How do we deal with scales: try them all
• Why is this efficient?

Vast majority of effort is in the first and second scales

\[ 1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{4^i} \ldots = \frac{4}{3} \]
Problem 2 – Describing Features

Image – 40
1/2 size, rot. 45°
Lightened +40

Full Image

100x100 crop
at Glasses
Problem 2 – Describing Features

Once we’ve found a corner/blobs, we can’t just use the image nearby. What about:

1. Scale?
2. Rotation?
3. Additive light?
Handling Scale

Given characteristic scale (maximum Laplacian response), we can just rescale image.
Handling Rotation

Given window, can compute “dominant orientation” and then rotate image

Slide credit: S. Lazebnik
Scale and Rotation

SIFT features at characteristic scales and dominant orientations

Scale and Rotation

1. Compute gradients
2. Build histogram (2x2 here, 4x4 in practice)

Gradients ignore global illumination changes
SIFT Descriptors

- In principle: build a histogram of the gradients
- In reality: quite complicated
  - Gaussian weighting: smooth response
  - Normalization: reduces illumination effects
  - Clamping
  - Tons of more stuff
Properties of SIFT

• Can handle: up to ~60 degree out-of-plane rotation, changes of illumination
• Fast, efficient, code available (but was patented)
Feature Descriptors

Think of feature as some non-linear filter that maps pixels to 128D feature

128D vector $\mathbf{x}$
Instance Matching

\[ \|x_1 - x_2\| = 0.61 \]

\[ \|x_1 - x_3\| = 1.22 \]

Example credit: J. Hays
Instance Matching

\[ \| x_4 - x_5 \| = 0.34 \]
\[ \| x_4 - x_6 \| = 0.30 \]
\[ \| x_4 - x_6 \| = 0.40 \]

Example credit: J. Hays
2\textsuperscript{nd} Nearest Neighbor Trick

- Given a feature $x_q$, nearest neighbor to $x$ is a good match, but distances can’t be thresholded.
- Instead, find nearest neighbor ($x_{1NN}$) and second nearest neighbor ($x_{2NN}$). This ratio is a good test for matches:

$$r = \frac{\|x_q - x_{1NN}\|}{\|x_q - x_{2NN}\|}$$
2nd Nearest Neighbor Trick

Figure from David G. Lowe. "Distinctive image features from scale-invariant keypoints." *IJCV* 60 (2), pp. 91-110, 2004.
So Far; What’s Next?

1: find corners+features
2: match based on local image data
3: next time: compute offsets from matches
Extra Reading for the Curious
Aside: This Trick is Common

Given a 50x16 person detector, how do I detect:
(a) 250x80 (b) 150x48 (c) 100x32 (d) 25x8 people?

Sample people from image
Aside: This Trick is Common

Detecting all the people
The red box is a fixed size

Sample people from image
Aside: This Trick is Common

Detecting all the people
The red box is a fixed size

Sample people from image
Aside: This Trick is Common

Detecting all the people
The red box is a fixed size

Sample people from image
Affine adaptation

Consider the second moment matrix of the window containing the blob:

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

Recall:

$$[u \ v] M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$

This ellipse visualizes the “characteristic shape” of the window.
Affine adaptation example

Scale-invariant regions (blobs)
Affine adaptation example

Affine-adapted blobs