CAP 4453
Robot Vision
Dr. Gonzalo Vaca-Castaño
gonzalo.vacacastano@ucf.edu
Administrative details

• Issues submitting homework
Credits

• Slides comes directly from:
  • Ioannis (Yannis) Gkioulekas (CMU)
  • Kris Kitani.
  • Fredo Durand (MIT).
  • James Hays (Georgia Tech).
  • Yogesh S Rawat (UCF)
  • Noah Snavely (Cornell)
Short Review from last class
Warping with different transformations

- Translation
- Affine
- Projective (homography)
View warping

original view  synthetic top view  synthetic side view

What are these black areas near the boundaries?
Virtual camera rotations

original view

synthetic rotations
Image rectification

two original images

rectified and stitched
Image warping

\[(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\]

\[(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_n, y'_n)\]
## Recap: Two Common Optimization Problems

<table>
<thead>
<tr>
<th>Problem statement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimize $|Ax - b|^2$</td>
<td>$x = \left( A^T A \right)^{-1} A^T b$</td>
</tr>
<tr>
<td>least squares solution to $Ax = b$</td>
<td></td>
</tr>
<tr>
<td>minimize $x^T A^T Ax$ s.t. $x^T x = 1$</td>
<td>$[v, \lambda] = \text{eig}(A^T A)$</td>
</tr>
<tr>
<td>non-trivial lsq solution to $Ax = 0$</td>
<td>$\lambda_1 &lt; \lambda_{2..n} : x = v_1$</td>
</tr>
</tbody>
</table>

```
import numpy as np
x, resid, rank, s = np.linalg.lstsq(A, b)
```
Affine transformations

• Matrix form

\[
\begin{bmatrix}
    x_1 & y_1 & 1 & 0 & 0 & 0 \\
    0 & 0 & 0 & x_1 & y_1 & 1 \\
    x_2 & y_2 & 1 & 0 & 0 & 0 \\
    0 & 0 & 0 & x_2 & y_2 & 1 \\
    \vdots \\
    x_n & y_n & 1 & 0 & 0 & 0 \\
    0 & 0 & 0 & x_n & y_n & 1
\end{bmatrix} \begin{bmatrix}
a \\
b \\
c \\
d \\
e \\
f
\end{bmatrix} = \begin{bmatrix}
x'_1 \\
y'_1 \\
x'_2 \\
y'_2 \\
x'_n \\
y'_n
\end{bmatrix} = \begin{bmatrix}
A \\
t
\end{bmatrix}
\]
Solving for homographies

\[
\begin{bmatrix}
   x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\
   0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\
   \vdots \\
   x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\
   0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n
\end{bmatrix}
\begin{bmatrix}
   h_{00} \\
   h_{01} \\
   h_{02} \\
   h_{10} \\
   h_{11} \\
   h_{12} \\
   h_{20} \\
   h_{21} \\
   h_{22}
\end{bmatrix}
= \begin{bmatrix}
   0 \\
   0 \\
   \vdots \\
   0 \\
   0
\end{bmatrix}
\]

\[
A \quad 2n \times 9 
\quad h \quad 9 
\quad 0 \quad 2n
\]

Defines a least squares problem: minimize \( \|Ah - 0\|^2 \)

- Since \( h \) is only defined up to scale, solve for unit vector \( \hat{h} \)
- Solution: \( \hat{h} = \text{eigenvector of } A^T A \) with smallest eigenvalue
- Works with 4 or more points
Recap: Two Common Optimization Problems

<table>
<thead>
<tr>
<th>Problem statement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimize $|Ax - b|^2$</td>
<td>$x = (A^T A)^{-1} A^T b$</td>
</tr>
<tr>
<td>least squares solution to $Ax = b$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem statement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimize $x^T A^T A x$ s.t. $x^T x = 1$</td>
<td>$[v, \lambda] = \text{eig}(A^T A)$</td>
</tr>
<tr>
<td>non-trivial lsq solution to $Ax = 0$</td>
<td>$\lambda_1 &lt; \lambda_{2..n} : x = v_1$</td>
</tr>
</tbody>
</table>
Image warping

How do we find point correspondences automatically?
Robot Vision

11. Feature points detection
Outline

• Motivation

• Detecting key points
  • Harris corner detector
  • Blob detection
Location Recognition
Robot Localization
Image matching
Structure from motion
3D photosynth
Image matching
Matching
Where are the corresponding points?
Application: KeyPoint Matching

1. Find a set of distinctive key-points
2. Define a region around each key-point
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors
Finding interest points
The aperture problem

• Individual pixels are ambiguous
• Idea: Look at whole patches!
Pick a point in the image. Find it again in the next image.

What type of feature would you select?
Pick a point in the image. 
Find it again in the next image.

What type of feature would you select?
Pick a point in the image.
Find it again in the next image.

*What type of feature would you select?*

a corner
What is an interest point?
Properties of interest points algorithm

• Detect all (or most) true interest points
• No false interest points
• Well localized
• Robust with respect to noise
• Efficient detection
• Detect points that are repeatable and distinctive
Outline

• Motivation

• Detecting key points
  • Harris corner detector
  • Blob detection
Corner detection: Possible approaches

• Based on brightness of images
  • Usually image derivatives

• Based on boundary extraction
  • First step edge detection
  • Curvature analysis of edges
Corner Detection: Basic Idea

- We should easily recognize the point by looking through a small window.
- Shifting a window in any direction should give a large change in intensity.

"flat" region: no change in all directions

"edge": no change along the edge direction

"corner": significant change in all directions

Source: A. Efros
Harris corner detector
Harris Detector


1. Compute x and y derivatives of image

\[ I_x = G^x_\sigma \ast I \quad I_y = G^y_\sigma \ast I \]

2. Compute products of derivatives at every pixel

\[ I_{x^2} = I_x \cdot I_x \quad I_{y^2} = I_y \cdot I_y \quad I_{xy} = I_x \cdot I_y \]

3. Compute the sums of the products of derivatives at each pixel

\[ S_{x^2} = G^{\sigma'} \ast I_{x^2} \quad S_{y^2} = G^{\sigma'} \ast I_{y^2} \quad S_{xy} = G^{\sigma'} \ast I_{xy} \]
Harris Detector


4. Define the matrix at each pixel

\[ M(x, y) = \begin{bmatrix} S_{x^2}(x, y) & S_{xy}(x, y) \\ S_{xy}(x, y) & S_{y^2}(x, y) \end{bmatrix} \]

5. Compute the response of the detector at each pixel

\[ R = \det M - k(\text{trace}M)^2 \]

6. Threshold on value of R; compute non-max suppression.
Corner Detection: Basic Idea

• We should easily recognize the point by looking through a small window

• Shifting a window in any direction should give a large change in intensity

“flat” region: no change in all directions

“edge”: no change along the edge direction

“corner”: significant change in all directions

Source: A. Efros
Corner detection the math

• Consider shifting the window $W_{nn}$ by $(u,v)$
  • how do the pixels in W change?
• Write pixels in window as a vector:

$$\phi_0 = [I(0, 0), I(0, 1), \ldots, I(n, n)]$$
$$\phi_1 = [I(0 + u, 0 + v), I(0 + u, 1 + v), \ldots, I(n + u, n + v)]$$

$$E(u, v) = \|\phi_0 - \phi_1\|_2^2$$
Corner detection: the math

Consider shifting the window $W$ by $(u,v)$

- how do the pixels in $W$ change?
- compare each pixel before and after by summing up the squared differences (SSD)
- this defines an SSD “error” $E(u,v)$:

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

- We want $E(u,v)$ to be as high as possible for all $u, v$!
Small motion assumption

Taylor Series expansion of $I$:

$$I(x+u, y+v) = I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion $(u,v)$ is small, then first order approximation is good

$$I(x + u, y + v) \approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

$$\approx I(x, y) + [I_x \ I_y] \begin{bmatrix} u \\ v \end{bmatrix}$$

shorthand: $I_x = \frac{\partial I}{\partial x}$

Plugging this into the formula on the previous slide...
Corner detection: the math

Consider shifting the window $W$ by $(u,v)$

- define an SSD “error” $E(u,v)$:

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

\[
\approx \sum_{(x,y) \in W} \left[ I(x, y) + I_x u + I_y v - I(x, y) \right]^2
\]

\[
\approx \sum_{(x,y) \in W} \left[ I_x u + I_y v \right]^2
\]
Corner detection: the math

Consider shifting the window $W$ by $(u,v)$
- define an “error” $E(u,v)$:

$$E(u,v) \approx \sum_{(x,y) \in W} [I_x u + I_y v]^2$$

$$\approx A u^2 + 2B uv + C v^2$$

$$A = \sum_{(x,y) \in W} I_x^2 \quad B = \sum_{(x,y) \in W} I_x I_y \quad C = \sum_{(x,y) \in W} I_y^2$$

- Thus, $E(u,v)$ is locally approximated as a quadratic error function
A more general formulation

• Maybe all pixels in the patch are not equally important
• Consider a “window function” $w(x, y)$ that acts as weights
• $E(u, v) = \sum_{(x,y) \in W} w(x, y) [I(x + u, y + v) - I(x, y)]^2$
• Case till now:
  • $w(x,y) = 1$ inside the window, 0 otherwise
Using a window function

• Change in appearance of window \( w(x,y) \) for the shift \([u,v]\):

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

Window function

Shifted intensity

Intensity

Window function \( w(x,y) = \)

1 in window, 0 outside

Gaussian

Source: R. Szeliski
Redoing the derivation using a window function

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

\[ \approx \sum_{x,y \in W} w(x,y) [I(x, y) + uI_x(x, y) + vI_y(x, y) - I(x, y)]^2 \]

\[ = \sum_{x,y \in W} w(x,y) [uI_x(x, y) + vI_y(x, y)]^2 \]

\[ = \sum_{x,y \in W} w(x,y) [u^2I_x(x, y)^2 + v^2I_y(x, y)^2 + 2uvI_x(x, y)I_y(x, y)] \]
Redoing the derivation using a window function

\[ E(u, v) \approx \sum_{x, y \in W} w(x, y)[u^2 I_x(x, y)^2 + v^2 I_y(x, y)^2 + 2uv I_x(x, y)I_y(x, y)] \]

\[ = Au^2 + 2Buv + Cv^2 \]

\[ A = \sum_{x, y \in W} w(x, y)I_x(x, y)^2 \]

\[ B = \sum_{x, y \in W} w(x, y)I_x(x, y)I_y(x, y) \]

\[ C = \sum_{x, y \in W} w(x, y)I_y(x, y)^2 \]
The second moment matrix

\[
E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}
\]

\[
M = \sum_{x, y \in W} w(x, y) \begin{bmatrix} I_x(x, y)^2 & I_x(x, y)I_y(x, y) \\ I_x(x, y)I_y(x, y) & I_y(x, y)^2 \end{bmatrix}
\]
The second moment matrix

\[ E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \]

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

Second moment matrix
Recall that we want \( E(u,v) \) to be as large as possible for all \( u,v \)

What does this mean in terms of \( M \)?
\[ E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \]

\[
A = \sum_{(x,y) \in W} I_x^2 \\
B = \sum_{(x,y) \in W} I_x I_y \\
C = \sum_{(x,y) \in W} I_y^2
\]

\[
M = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\
M \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
E(u, v) = 0 \quad \forall u, v
\]

Flat patch: \[
I_x = 0 \\
I_y = 0
\]
\[ E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \]

\[ A = \sum_{(x,y) \in W} I_x^2 \]

\[ B = \sum_{(x,y) \in W} I_x I_y \]

\[ C = \sum_{(x,y) \in W} I_y^2 \]

Vertical edge: \( I_y = 0 \)

\[ M = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \]

\[ M \begin{bmatrix} 0 \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \]

\[ E(0, v) = 0 \ \forall v \]
\[ E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \]

\[
A = \sum_{(x,y) \in W} I_x^2 \\
B = \sum_{(x,y) \in W} I_x I_y \\
C = \sum_{(x,y) \in W} I_y^2
\]

Horizontal edge: \( I_x = 0 \)

\[ M^r = \begin{bmatrix} 0 & 0 \\ 0 & C \end{bmatrix} \]

\[
M \begin{bmatrix} u \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
E(u, 0) = 0 \ \forall u
\]
What about edges in arbitrary orientation?
Solutions to \( Mx = 0 \) are directions for which \( E \) is 0: window can slide in this direction without changing appearance.
\[ E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \]

Solutions to \( Mx = 0 \) are directions for which \( E \) is 0: window can slide in this direction without changing appearance.

For corners, we want no such directions to exist.
Harris Detector

1. Compute x and y derivatives of image
   \[ I_x = G^x_{\sigma} \ast I \quad I_y = G^y_{\sigma} \ast I \]

2. Compute products of derivatives at every pixel
   \[ I_{x^2} = I_x \cdot I_x \quad I_{y^2} = I_y \cdot I_y \quad I_{xy} = I_x \cdot I_y \]

3. Compute the sums of the products of derivatives at each pixel
   \[ S_{xy} = G_{\sigma} \ast I_{xy} \quad S_{y^2} = G_{\sigma} \ast I_{y^2} \quad S_{x^2} = G_{\sigma} \ast I_{x^2} \]

4. Define the matrix at each pixel
   \[ M(x, y) = \begin{bmatrix} S_{x^2}(x, y) & S_{xy}(x, y) \\ S_{xy}(x, y) & S_{y^2}(x, y) \end{bmatrix} \]

5. Compute the response of the detector at each pixel
   \[ R = \det M - k(\text{trace}M)^2 \]

6. Threshold on value of R; compute non-max suppression.

\[ E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} = \text{Constant} \]
Visualization as an ellipse

Since $M$ is symmetric, we have

$$M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

We can visualize $M$ as an ellipse with axis lengths determined by the eigenvalues and orientation determined by $R$.

Ellipse equation:

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

$\lambda_1$ - direction of the fastest change

$\lambda_2$ - direction of the slowest change

$(\lambda_{\text{max}})^{-1/2}$

$(\lambda_{\text{min}})^{-1/2}$
SVD

\[ A = U \Sigma V^{-1} \]

\[ \Sigma = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_N \end{bmatrix} \]

\[ U, V = \text{orthogonal matrix} \quad \rightarrow \quad U^{-1} = U^T \]

\[ \sigma_i = \sqrt{\lambda_i} \quad \sigma = \text{singular value} \]

\[ \lambda = \text{eigenvalue of } A^\dagger A \]

For a square symmetric matrix

- \( U, V \) becomes Rotation Matrix \( R \)
- Diagonal matrix has eigenvalues of \( A \)
Compute eigenvalues and eigenvectors

\[
Me = \lambda e \\
(M - \lambda I)e = 0
\]
1. Compute the determinant of
   \[ M - \lambda I \]
   (returns a polynomial)
Compute eigenvalues and eigenvectors

\[ Me = \lambda e \]

\[ (M - \lambda I)e = 0 \]

1. Compute the determinant of
   \[ M - \lambda I \]
   (returns a polynomial)

2. Find the roots of polynomial
   \[ \det(M - \lambda I) = 0 \]
   (returns eigenvalues)
1. Compute the determinant of $M - \lambda I$ (returns a polynomial)

2. Find the roots of polynomial $\det(M - \lambda I) = 0$ (returns eigenvalues)

3. For each eigenvalue, solve $(M - \lambda I)e = 0$ (returns eigenvectors)
Eigenvalues & Eigenvector computation example

• Compute eigenvalues, eigenvectors of \( A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \).

• Determinant of the matrix \((A - \lambda I)\) equals zero are the eigenvalues

\[
|A - \lambda I| = \begin{vmatrix} 2 - \lambda & 1 \\ 1 & 2 - \lambda \end{vmatrix} = (2 - \lambda)(2 - \lambda) - 1 = 3 - 4\lambda + \lambda^2.
\]

• Setting the characteristic polynomial equal to zero, it has roots at \(\lambda=1\) and \(\lambda=3\), which are the two eigenvalues of A.
Eigenvalues & Eigenvector computation example

• Compute eigenvalues, eigenvectors of

\[
A = \begin{bmatrix}
2 & 1 \\
1 & 2
\end{bmatrix}.
\]

• For \(\lambda=1\),

\[
(A - I)v_{\lambda=1} = \begin{bmatrix}
1 & 1 \\
1 & 1
\end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\]

\(1v_1 + 1v_2 = 0\)

\[
v_{\lambda=1} = \begin{bmatrix} v_1 \\ -v_1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}
\]

Any nonzero vector with \(v_1 = -v_2\) solves this equation.

For \(\lambda=3\),

\[
(A - 3I)v_{\lambda=3} = \begin{bmatrix}
-1 & 1 \\
1 & -1
\end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\]

\(-1v_1 + 1v_2 = 0;\)

\(1v_1 - 1v_2 = 0\)

Any nonzero vector with \(v_1 = v_2\) solves this equation. Therefore,

\[
v_{\lambda=3} = \begin{bmatrix} v_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\]
Eigenvalues and eigenvectors of $M$

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

- Define shift directions with the smallest and largest change in error
- $x_{\text{max}}$ = direction of largest increase in $E$
- $\lambda_{\text{max}}$ = amount of increase in direction $x_{\text{max}}$
- $x_{\text{min}}$ = direction of smallest increase in $E$
- $\lambda_{\text{min}}$ = amount of increase in direction $x_{\text{min}}$

\[ M \cdot x_{\text{max}} = \lambda_{\text{max}} x_{\text{max}} \]
\[ M \cdot x_{\text{min}} = \lambda_{\text{min}} x_{\text{min}} \]
Interpreting the eigenvalues

\[ \lambda_{\text{max}} \approx \lambda_{\text{min}} \gg 0 \]

Corner

\[ \lambda_{\text{max}} \approx \lambda_{\text{min}} \gg 0 \]
E very high in all directions

Edge

\[ \lambda_{\text{max}} \gg \lambda_{\text{min}}, \lambda_{\text{min}} \approx 0 \]
E remains close to 0 along \( x_{\text{min}} \)

Flat patch

\[ \lambda_{\text{max}}, \lambda_{\text{min}} \text{ are small; } E \text{ is almost 0 in all directions} \]
Use threshold on eigenvalues to detect corners

Think of a function to score ‘cornerness’
Use threshold on eigenvalues to detect corners

Think of a function to score ‘cornerness’
Use threshold on eigenvalues to detect corners

(\text{a function of})

\( R = \min(\lambda_1, \lambda_2) \)
Corner response function

\[ R = \min(\lambda_1, \lambda_2) \]
Use threshold on eigenvalues to detect corners

(a function of )

Eigenvalues need to be bigger than one.

\[ R = \lambda_1 \lambda_2 - \kappa (\lambda_1 + \lambda_2)^2 \]

Can compute this more efficiently...
Corner response function

\[ R = \det(M) - \alpha \text{trace}(M)^2 = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2 \]
Use threshold on eigenvalues to detect corners

(a function of $\lambda$)

$$R = \det(M) - \kappa \text{trace}^2(M)$$

- $R < 0$ (corner)
- $R > 0$
- $R \ll 0$
- $R < 0$

$$\lambda_1, \lambda_2$$

$$\det M = \lambda_1 \lambda_2$$
$$\text{trace} M = \lambda_1 + \lambda_2$$

$$\det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$$
$$\text{trace} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = a + d$$
Harris & Stephens (1988)

\[ R = \det(M) - \kappa \text{trace}^2(M) \]

Kanade & Tomasi (1994)

\[ R = \min(\lambda_1, \lambda_2) \]

Nobel (1998)

\[ R = \frac{\det(M)}{\text{trace}(M) + \epsilon} \]
Harris Detector

1. Compute $x$ and $y$ derivatives of image

\[ I_x = G^x_\sigma \ast I \quad I_y = G^y_\sigma \ast I \]

2. Compute products of derivatives at every pixel

\[ I_{x^2} = I_x \cdot I_x \quad I_{y^2} = I_y \cdot I_y \quad I_{xy} = I_x \cdot I_y \]

3. Compute the sums of the products of derivatives at each pixel

\[ S_{xy} = G^x_\sigma \ast I_{xy} \quad S_{y^2} = G^y_\sigma \ast I_{y^2} \quad S_{x^2} = G^x_\sigma \ast I_{x^2} \]

4. Define the matrix at each pixel

\[ M(x, y) = \begin{bmatrix} S_{x^2}(x, y) & S_{xy}(x, y) \\ S_{xy}(x, y) & S_{y^2}(x, y) \end{bmatrix} \]

5. Compute the response of the detector at each pixel

\[ R = \det M - k(\text{trace}M)^2 \]

6. Threshold on value of $R$; compute non-max suppression.
Final step: Non-maxima suppression

• Pick a pixel as corner if cornerness at patch centered on it > cornerness of neighboring pixels
• And if cornerness exceeds a threshold
Harris Detector

1. Compute x and y derivatives of image
   \[ I_x = G_\sigma^x * I \quad I_y = G_\sigma^y * I \]

2. Compute products of derivatives at every pixel
   \[ I_{x^2} = I_x * I_x \quad I_{y^2} = I_y * I_y \quad I_{xy} = I_x * I_y \]

3. Compute the sums of the products of derivatives at each pixel
   \[ S_{xy} = G_\sigma^x * I_{xy} \]
   \[ S_{y^2} = G_\sigma^y * I_{y^2} \]
   \[ S_{x^2} = G_\sigma^x * I_{x^2} \]

4. Define the matrix at each pixel
   \[ M(x, y) = \begin{bmatrix} S_{x^2}(x, y) & S_{xy}(x, y) \\ S_{xy}(x, y) & S_{y^2}(x, y) \end{bmatrix} \]

5. Compute the response of the detector at each pixel
   \[ R = \text{det} M - k(\text{trace} M)^2 \]

6. Threshold on value of R; compute non-max suppression.
Harris detector example
f value (red high, blue low)
Threshold \( (f > \text{value}) \)
Find local maxima of $f$
Harris features (in red)
Harris corner response is invariant to rotation

Ellipse rotates but its shape (\textit{eigenvalues}) remains the same

Corner response R is invariant to image rotation
Harris corner response is invariant to intensity changes

Partial invariance to *affine intensity* change

- Only derivatives are used $\Rightarrow$ invariance to intensity shift $I \rightarrow I + b$

- Intensity scale: $I \rightarrow a \, I$

\[ R \]

\[ x \text{ (image coordinate)} \]
The Harris detector is not invariant to changes in ...
The Harris corner detector is **not** invariant to scale
Multi-scale detection
How can we make a feature detector scale-invariant?
How can we automatically select the scale?
Scale invariant detection

Suppose you’re looking for corners

Key idea: find scale that gives local maximum of \textit{cornerness}
  • in both position and scale
  • One definition of \textit{cornerness}: the Harris operator
Intuitively…

Find local maxima in both **position** and **scale**
Automatic scale selection

Lindeberg et al., 1996

\[ M = \mu(x, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \otimes \begin{bmatrix} L_x^2(x, \sigma_D) & L_x L_y(x, \sigma_D) \\ L_x L_y(x, \sigma_D) & L_y^2(x, \sigma_D) \end{bmatrix} \]
Automatic scale selection

\[ M = \mu(x, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \odot \begin{bmatrix} L_x^2(x, \sigma_D) & L_x L_y(x, \sigma_D) \\ L_x L_y(x, \sigma_D) & L_y^2(x, \sigma_D) \end{bmatrix} \]

Increased
Automatic scale selection

\[ M = \mu(x, \sigma_I, \sigma_D) = \sigma^2_D g(\sigma_I) \otimes \begin{bmatrix} L_z^2(x, \sigma_D) & L_z y(x, \sigma_D) \\ L_z y(x, \sigma_D) & L_y^2(x, \sigma_D) \end{bmatrix} \]

Increased
Automatic scale selection

$M = \mu(x, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \otimes \begin{bmatrix} L_2^2(x, \sigma_D) & L_2 L_y(x, \sigma_D) \\ L_2 L_y(x, \sigma_D) & L_2^2(x, \sigma_D) \end{bmatrix}$

Increased
Automatic scale selection

\[ M = \mu(x, \sigma_I, \sigma_D) = \sigma_{D_I}^2 g(\sigma_I) \otimes \begin{bmatrix} L_x^2(x, \sigma_D) & L_x L_y(x, \sigma_D) \\ L_x L_y(x, \sigma_D) & L_y^2(x, \sigma_D) \end{bmatrix} \]

Increased
Automatic scale selection

$$M = \mu(x, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \otimes \begin{bmatrix} L_x^2(x, \sigma_D) & L_x L_y(x, \sigma_D) \\ L_x L_y(x, \sigma_D) & L_y^2(x, \sigma_D) \end{bmatrix}$$

$$f(L_{k..m}(x, \sigma))$$
Implementation

• Instead of computing $f$ for larger and larger windows, we can implement using a fixed window size with a Gaussian pyramid.
Gaussian pyramid implementation
How would you implement scale selection?
implementation

For each level of the Gaussian pyramid

compute feature response (e.g. Harris, Laplacian)

For each level of the Gaussian pyramid

if local maximum and cross-scale

save scale and location of feature \((x, y, s)\)
Blob detection
Scale-space blob detector: Example
Feature extraction: Corners and blobs
Formally...

Highest response when the signal has the same characteristic scale as the filter.
Another common definition of $f$

- The *Laplacian of Gaussian* (LoG)

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

(very similar to a Difference of Gaussians (DoG) – i.e. a Gaussian minus a slightly smaller Gaussian)
Laplacian of Gaussian

• “Blob” detector

• Find maxima and minima of LoG operator in space and scale
Scale-space blob detector: Example

\[ \text{sigma} = 11.9912 \]
Scale-space blob detector: Example
Scale selection

- At what scale does the Laplacian achieve a maximum response for a binary circle of radius \( r \)?
Characteristic scale

- We define the characteristic scale as the scale that produces peak of Laplacian response.

What happens if you apply different Laplacian filters?
jet color scale
blue: low, red: high
What happened when you applied different Laplacian filters?
What happened when you applied different Laplacian filters?
maximum response
optimal scale

2.1  4.2  6.0  9.8  15.5  17.0

Full size image

2.1  4.2  6.0  9.8  15.5  17.0

3/4 size image
optimal scale

Full size image

3/4 size image
cross-scale maximum

local maximum

4.2

local maximum

6.0

local maximum

9.8
Scale Invariant Detection

- Functions for determining scale

Kernels:

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

(Laplacian)

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

(Difference of Gaussians)

where Gaussian

\[ G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \]

Note: The LoG and DoG operators are both rotation equivariant
Basic reading:
• Szeliski textbook, Sections 4.1.
Questions?