Correspondence: Feature detection
A general pipeline for correspondence

1. If sparse correspondences are enough, choose points for which we will search for correspondences (feature points)

2. For each point (or every pixel if dense correspondence), describe point using a feature descriptor

3. Find best matching descriptors across two images (feature matching)

4. Use feature matches to perform downstream task, e.g., pose estimation
Characteristics of good feature points

- **Repeatability / invariance**
  - The same feature point can be found in several images despite geometric and photometric transformations

- **Saliency / distinctiveness**
  - Each feature point is distinctive
  - Fewer “false” matches
Goal: repeatability

- We want to detect (at least some of) the same points in both images.

No chance to find true matches!

- Yet we have to be able to run the detection procedure *independently* per image.
Goal: distinctiveness

- The feature point should be distinctive enough that it is easy to match
  - Should at least be distinctive from other patches nearby
The aperture problem
The aperture problem

- Individual pixels are ambiguous
- Idea: Look at whole patches!
The aperture problem

- Individual pixels are ambiguous
- Idea: Look at whole patches!
The aperture problem

• *Some local neighborhoods* are ambiguous
The aperture problem
Corner detection

• Main idea: Translating window should cause large differences in patch appearance
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 $W$ 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” $E(u,v)$:

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

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

- define an “error” $E(u,v)$:

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

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

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

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 \([u,v]\):

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

Window function

Shifted intensity

Intensity

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

1 in window, 0 outside

or

Gaussian

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

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

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

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

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

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

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

Second moment matrix
The 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:
\[
\begin{align*}
I_x &= 0 \\
I_y &= 0
\end{align*}
\]
\[ E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \]

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

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

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

Vertical edge: \( I_y = 0 \)

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

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

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

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

Horizontal edge: \( I_x = 0 \)

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

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

\[ E(u, 0) = 0 \ \forall u \]
What about edges in arbitrary orientation?
\[ E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \]

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

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.
Eigenvalues and eigenvectors of $M$

- $Mx = 0 \Rightarrow Mx = \lambda x$: $x$ is an eigenvector of $M$ with eigenvalue 0
- $M$ is 2 x 2, so it has 2 eigenvalues ($\lambda_{\text{max}}, \lambda_{\text{min}}$) with eigenvectors ($x_{\text{max}}, x_{\text{min}}$)
- $E(x_{\text{max}}) = x_{\text{max}}^T M x_{\text{max}} = \lambda_{\text{max}} \|x_{\text{max}}\|^2 = \lambda_{\text{max}}$ (eigenvectors have unit norm)
- $E(x_{\text{min}}) = x_{\text{min}}^T M x_{\text{min}} = \lambda_{\text{min}} \|x_{\text{min}}\|^2 = \lambda_{\text{min}}$
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 $E$
  - $\lambda_{\text{max}} = \text{amount of increase in direction } x_{\text{max}}$
  - $x_{\text{min}}$ = direction of smallest increase in $E$
  - $\lambda_{\text{min}} = \text{amount of increase in direction } x_{\text{min}}$
Interpreting the eigenvalues

- **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**: \(\lambda_{\text{max}}, \lambda_{\text{min}}\) are small; \(E\) is almost 0 in all directions
Computing the second moment matrix efficiently

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

- Window function \( w(x,y) \) typically a Gaussian centered on the window
  - \( w(x, y) = e^{-\frac{(x-x_0)^2}{\sigma^2} - \frac{(y-y_0)^2}{\sigma^2}} \)

- Need to compute this matrix efficiently for every window location
Computing the second moment matrix efficiently

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

- Step 1: Place \( k \times k \) window
- Step 2: Compute \( \sum_{x, y \in W} w(x, y)I_x(x, y)^2 = \sum_{x, y} e^{\frac{(x-x_0)^2}{\sigma^2}} - \frac{(y-y_0)^2}{\sigma^2} I_x(x, y)^2 \) (similarly other terms)
- This can be expressed as a convolution!
Computing the second moment matrix

• Compute image gradients $I_x, I_y$ (both of these are images)
  • Might want to blur with a Gaussian before doing this. Why?
• Compute $I_x^2, I_y^2, I_xI_y$ (these are images too)
• Convolve with windowing function (typically Gaussian)
• Assemble second moment matrix at every pixel
Corner detection: the math

How are $\lambda_{\text{max}}$, $x_{\text{max}}$, $\lambda_{\text{min}}$, and $x_{\text{min}}$ relevant for feature detection?

- Need a feature scoring function

Want $E(u,v)$ to be large for small shifts in all directions

- the minimum of $E(u,v)$ should be large, over all unit vectors $[u \ v]$
- this minimum is given by the smaller eigenvalue ($\lambda_{\text{min}}$) of $M$
Corner detection summary

Here’s what you do

• Compute the gradient at each point in the image
• Create the $M$ matrix from the entries in the gradient
• Compute the eigenvalues
• Find points with large response ($\lambda_{\text{min}} > \text{threshold}$)
• Choose those points where $\lambda_{\text{min}}$ is a local maximum as features
Corner detection summary

Here’s what you do

- Compute the gradient at each point in the image
- Create the $H$ matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response ($\lambda_{\text{min}} > \text{threshold}$)
- Choose those points where $\lambda_{\text{min}}$ is a local maximum as features.
The Harris operator

λ_{min} is a variant of the “Harris operator” for feature detection

\[ f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{\text{determinant}(H)}{\text{trace}(H)} \]

- The trace is the sum of the diagonals, i.e., trace(H) = h_{11} + h_{22}
- Very similar to λ_{min} but less expensive (no square root)
- Called the “Harris Corner Detector” or “Harris Operator”
  - Actually the Noble variant of the Harris Corner Detector
- Lots of other detectors, this is one of the most popular
Corner response function

\[ R = \det(M) - \alpha \text{trace}(M)^2 = \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2 \]
The Harris operator

Harris operator

$\lambda_{\text{min}}$
Harris Detector [Harris88]

- Second moment matrix

\[
\mu(\sigma_I, \sigma_D) = g(\sigma_I) \begin{bmatrix}
I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\
I_x I_y(\sigma_D) & I_y^2(\sigma_D)
\end{bmatrix}
\]

1. Image derivatives (optionally, blur first)

- \[\text{det } M = \lambda_1 \lambda_2\]
- \[\text{trace } M = \lambda_1 + \lambda_2\]

2. Square of derivatives

3. Gaussian filter \(g(\sigma_i)\)

4. Cornerness function – both eigenvalues are strong

\[
har = \text{det}[\mu(\sigma_I, \sigma_D)] - \alpha \left[\text{trace}(\mu(\sigma_I, \sigma_D))^2\right] = g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2
\]

5. Non-maxima suppression
Weighting the derivatives

• In practice, using a simple window $W$ doesn’t work too well

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

• Instead, we’ll *weight* each derivative value based on its distance from the center pixel

$$H = \sum_{(x,y) \in W} w_{x,y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$
Harris detector example
f value (red high, blue low)
Threshold ($f > value$)
Find local maxima of $f$
Harris features (in red)