CSE 559A: Computer Vision

Fall 2020: T-R: 11:30-12:50pm @ Wrighton 300 / Zoom

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Adith Boloor, Patrick Williams

http://www.cse.wustl.edu/~ayan/courses/cse559a/

Sep 29, 2020
• Reminder: Office hours happening on Friday and Tuesday mornings.
• Next class will be broadcast from Wrighton 300:
  ■ You can still choose to attend over Zoom.
  ■ I will monitor the live chat and answer questions like before.
  ■ If you attend in person, you can choose to ask questions verbally or through the chat.
• Waitlist:
  ■ Tomorrow is the drop deadline. Will close enrollment today evening.
  ■ Target enrollment is 50. We are now at 57.
  ■ Will enroll from waitlist if more than 7 people drop by late afternoon.
APPLICATION: EDGE DETECTION

Smoothing + Derivative

\[ I_x = \partial_x \ast (G_\sigma \ast X) = (\partial_x \ast G_\sigma) \ast X = G_{x;\sigma} \ast X \quad I_y = G_{y;\sigma} \ast X \]

\[ G_x = \frac{-x}{2\pi\sigma^4} \exp \left( -\frac{x^2 + y^2}{2\sigma^2} \right) \quad G_y = \frac{-y}{2\pi\sigma^4} \exp \left( -\frac{x^2 + y^2}{2\sigma^2} \right) \]

\[ G_\theta = \frac{-(x \cos \theta + y \sin \theta)}{2\pi\sigma^4} \exp \left( -\frac{x^2 + y^2}{2\sigma^2} \right) \]

\[ I_\theta = I_x \cos \theta + I_y \sin \theta \]  

Just need to convolve twice. Gives us an expression for derivative along every direction.
APPLICATION: EDGE DETECTION

Smoothing + Derivative

\[ I_\theta = I_x \cos \theta + I_y \sin \theta \]
APPLICATION: EDGE DETECTION

Smoothing + Derivative

\[ I_\theta[n] = I_x[n] \cos \theta + I_y[n] \sin \theta \]
APPLICATION: EDGE DETECTION

Smoothing + Derivative

\[ I_\theta[n] = I_x[n] \cos \theta + I_y[n] \sin \theta \]

\[ H[n] = \sqrt{I_x^2[n] + I_y^2[n]} = \max_\theta I_\theta[n] \]
APPLICATION: EDGE DETECTION

Smoothing + Derivative

\[ I_\theta[n] = I_x[n] \cos \theta + I_y[n] \sin \theta \]

\[ H[n] = \sqrt{I_x^2[n] + I_y^2[n]} = \max_{\theta} I_\theta[n] \]

\[ \Theta[n] = \text{atan2}(I_y, I_x) = \arg \max_{\theta} I_\theta[n] \]
APPLICATION: EDGE DETECTION

Smoothing + Derivative

\[ I_\theta[n] = I_x[n] \cos \theta + I_y[n] \sin \theta \]

\[ H[n] = \sqrt{I_x^2[n] + I_y^2[n]} = \max_\theta I_\theta[n] \]

\[ \Theta[n] = \text{atan2}(I_y, I_x) = \arg \max_\theta I_\theta[n] \]

Gives us gradient magnitude and direction.

Often applied even to filters that aren't "steerable" like DoG.
APPLICATION: EDGE DETECTION

$I_x$

$I_y$

$I_{45°}$
APPLICATION: EDGE DETECTION

$I_x$  

$I_y$  

$H$
APPLICATION: EDGE DETECTION

\[ I_x \quad I_y \quad H > \epsilon \]
APPLICATION: EDGE DETECTION

\[ I_x \quad I_y \quad H > \epsilon \]
Extensions

- Non-maxima Supression: Keep an edge pixel only if its magnitude is higher than its neighbors along the direction of the derivative.

\[
\begin{array}{ccc}
  e & f & g \\
  b & a & c \\
  h & j & k \\
\end{array}
\]

Declare edge if a above threshold and:
- \( a > b \) and \( a > c \) if \( \theta = 0 \)
- \( a > f \) and \( a > j \) if \( \theta = 90 \)
- \( a > e \) and \( a > k \) if \( \theta = 45 \)
- ....
APPLICATION: EDGE DETECTION

Extensions

- Non-maxima Supression: Keep an edge pixel only if its magnitude is higher than its neighbors along the direction of the derivative.

```
  e f g
 b a c
 h j k
```

Declare edge if a above threshold and:
- $a > b$ and $a > c$ if $\theta = 0$
- $a > f$ and $a > j$ if $\theta = 90$
- $a > e$ and $a > k$ if $\theta = 45$
- ....

- Canny: Keep a lower magnitude edge pixel if it has a higher edge magnitude neighbor.
  Two thresholds (hysteresis)

- Second derivative filters.

See Szeliski Section 4.2
Edges are isolated per-pixel labels
Pool them together to detect scene structure: E.g., Lines

Edges are isolated per-pixel labels
Pool them together to detect scene structure: E.g., Lines

Missed detections
Clutter
Occlusions

Edges are isolated per-pixel labels
Hough Transform
- Consider ALL possible lines (on a 2D plane)
Hough Transform

- Consider ALL possible lines (on a 2D plane)
Hough Transform

- Consider ALL possible lines (on a 2D plane)
- This is a two dimensional search space, could parameterize it in different ways.

\[ r = x \cos \theta + y \sin \theta \]
Hough Transform

- Consider ALL possible lines (on a 2D plane)
- This is a two dimensional search space, could parameterize it in different ways.

\[ r = x \cos \theta + y \sin \theta \]
\[ \theta \in [-\pi/2, \pi/2] \]
\[ r \in [-r_{\text{max}}, r_{\text{max}}] \]
\[ r = x \cos \theta + y \sin \theta \]

**Hough Transform**

Image where each pixel corresponds to some discrete value of \( r \) and \( \theta \)

Discretize this space into a bunch of buckets.
r = x \cos \theta + y \sin \theta

Hough Transform

Discretize this space into a bunch of buckets.
$r = x \cos \theta + y \sin \theta$

Hough Transform

Each edge pixel casts a vote for all lines it could belong to.

Compute by plugging in $x, y$ into equation: get a value of $r$ for each $\theta$ (get a sinusoid)
$r = x \cos \theta + y \sin \theta$

**Hough Transform**

Each edge pixel casts a vote for all lines it could belong to.

Compute by plugging in $x,y$ into equation: get a value of $r$ for each $\theta$ (get a sinusoid)
\[ r = x \cos \theta + y \sin \theta \]

Each edge pixel casts a vote for all lines it could belong to.

Compute by plugging in \( x, y \) into equation: get a value of \( r \) for each \( \theta \) (get a sinusoid)
$r = x \cos \theta + y \sin \theta$

Hough Transform

Each edge pixel casts a vote for all lines it could belong to.

Compute by plugging in $x, y$ into equation: get a value of $r$ for each $\theta$ (get a sinusoid)
Each edge pixel casts a vote for all lines it could belong to.

Compute by plugging in $x,y$ into equation: get a value of $r$ for each $\theta$ (get a sinusoid)

$r = x \cos \theta + y \sin \theta$

Hough Transform
$r = x \cos \theta + y \sin \theta$

**Hough Transform**

Each edge pixel casts a vote for all lines it could belong to.

Compute by plugging in $x,y$ into equation: get a value of $r$ for each $\theta$ (get a sinusoid)

Do this for all pixels and see which 'bins' get the most votes.
\[ r = x \cos \theta + y \sin \theta \]

**Hough Transform**

Each edge pixel casts a vote for all lines it could belong to.

Compute by plugging in \(x,y\) into equation: get a value of \(r\) for each \(\theta\) (get a sinusoid)

Do this for all pixels and see which 'bins' get the most votes.

**Variants**
- Each edge pixel only casts one vote based on angle. (with or without sign)
- Vote weighted by magnitude of gradient.
- Exclusive vote (select dominant line, remove vote from its pixels for other lines)

- Use same idea for line segments, circles, ellipses ...
Other Neighborhood Operations

**Median Filter / Order Statistics**

\[ Y[n] = \text{Median}\{X[n - n']\}_{N[n'] = 1} \]

- Neighborhood function \(N[n'] \in \{0, 1\}\)
- Often better at removing outliers than convolution.

- Other ops: \(Y[n] = \max / \min\{X[n - n']\}_{N[n'] > 0}\)

Morphological Operations

- Conducted on binary images ($X[n] \in \{0, 1\}$)
- Erosion: $Y[n] = \text{AND} \{X[n - n']\}_{N[n'] = 1}$
- Dilation: $Y[n] = \text{OR} \{X[n - n']\}_{N[n'] = 1}$
Other Neighborhood Operations

Morphological Operations

- Conducted on binary images ($X[n] \in \{0, 1\}$)
- Erosion: $Y[n] = \text{AND} \{X[n - n'] \}_{N[n'] = 1} \quad (1 \text{ if all neighbors } 1)$
- Dilation: $Y[n] = \text{OR} \{X[n - n'] \}_{N[n'] = 1} \quad (1 \text{ if any neighbor } 1)$
- Opening: Erosion followed by Dilation
- Closing: Dilation followed by Erosion

See Szeliski Sec 3.3.2
Denoising by Smoothing (with a Gaussian filter):

\[
X
\]
BILATERAL FILTERING

Denoising by Smoothing (with a Gaussian filter):

\[ X \quad Y = X \ast G \]
BILATERAL FILTERING

Denoising by Smoothing (with a Gaussian filter):

\[ X \]

\[ Y = X \ast G \]

\[ Y[n] = \sum_{n'} G[n']X[n - n'] \]
BILATERAL FILTERING

Denoising by Smoothing (with a Gaussian filter):

\[ X \]

\[ Y = X * G \]

\[ G'[n_1, n_2] = G[n_1 - n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} \right) \]

\[ \sum_{n_2} G'[n_1, n_2] = 1 \]

\[ Y[n] = \sum_{n'} G[n']X[n - n'] \]

\[ Y[n_1] = \sum_{n_2} G'[n_1, n_2]X[n_2] \]
Denoising by Smoothing (with a Gaussian filter):

\[ Y = X \ast G \]
BILATERAL FILTERING

Denoising by Smoothing (with a Gaussian filter):

\[ Y = X * G \]
Denoising by Smoothing (with a Gaussian filter):

\[ Y = X \ast G \]
BILATERAL FILTERING

Denoising by Smoothing (with a Gaussian filter):

\[ X \]

\[ Y = X * G \]

\[ G'[n_1, n_2] = G[n_1 - n_2] \propto \exp \left( - \frac{|n_1 - n_2|^2}{2\sigma^2} \right) \]

\[ \sum_{n_2} G'[n_1, n_2] = 1 \]
BILATERAL FILTERING

Denoising by Smoothing (with a Gaussian filter):

\[ X \]

\[ B[n_1, n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_f^2} \right) \]

\[ \sum_{n_2} B[n_1, n_2] = 1 \]

Make the filter weights data dependent!
Denoising by Smoothing (with a Gaussian filter):

\[ X \]

\[ B[n_1, n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_f^2} \right) \]

\[ \sum_{n_2} B[n_1, n_2] = 1 \]
Denoising by Smoothing (with a Gaussian filter):

\[ X \]

\[
B[n_1, n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_i^2} \right)
\]

\[
\sum_{n_2} B[n_1, n_2] = 1
\]
BILATERAL FILTERING

Denoising by Smoothing (with a Gaussian filter):

\[
X
\]

\[
B[n_1, n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_I^2} \right)
\]

\[
\sum_{n_2} B[n_1, n_2] = 1
\]
BILATERAL FILTERING

Denoising with a Bilateral Filter

\[ B[n_1, n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_f^2} \right) \]
Denoising with a Bilateral Filter

\[ B[n_1, n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_f^2} \right) \]
Denoising with a Bilateral Filter

\[ B[n_1, n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_I^2} \right) \]

\( \sigma_I \) Medium
Denoising with a Bilateral Filter

\[ B[n_1, n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_I^2} \right) \]
Denoising with a Bilateral Filter

\[ B[n_1, n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_I^2} \right) \]
BILATERAL FILTERING

Denoising with a Bilateral Filter

\[ B[n_1, n_2] \propto \exp \left( -\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_f^2} \right) \]

\( \sigma_f \) Low Repeated

Gaussian Filter Result
BILATERAL FILTERING

- *Guided Bilateral Filter*: $B[n_1, n_2]$ based on a separate image $Z[n]$: depth, infra-red, etc.
- Far less efficient than convolution
  - Filter also has to be computed, normalized, at each output location.
  - Efficient Datastructures Possible
- Further Reading:
  - Paris et al., SIGGRAPH/CVPR Course on Bilateral Filtering
  - Recent work on using this for inference, best paper runner up at ECCV 2016
Quick Recap: Complex Numbers

- A complex number $f = x + jy$ where $x$ and $y$ are scalar numbers.
  - $j = \sqrt{-1}$ (EE convention: we use $j$ instead of $i$)
  - $x$ and $y$ are called the real and imaginary components of $f$

Think of $f$ as a 2-D vector with special definitions of addition, multiplication, etc.

- $(x_1 + jy_1) + (x_2 + jy_2) = (x_1 + x_2) + j(y_1 + y_2)\)
- $(x_1 + jy_1) \times (x_2 + jy_2) = (x_1x_2 - y_1y_2) + j(x_2y_1 + x_1y_2)\)
- $(x_1 + jy_1) \times x_2 = x_1x_2 + jy_1x_2$
- Conjugate: $x + jy = x - jy = x + j(-y)$
- Magnitude: $(x + jy) \times (x + jy) = x^2 + y^2$
Quick Recap: Complex Numbers

Euler’s Formula

- \( \exp(j\theta) = \cos \theta + j \sin \theta \)
- \( x + jy = M \exp(j\theta) \)
  - \( M = \sqrt{x^2 + y^2}, \theta = \tan^{-1}(y, x) \)
  - \( \theta \) is called the “phase”
- \( M \exp(j\theta) = M \exp(-j\theta) \)
- \( (x + jy) \times \exp(j\theta_0) = M \exp(j(\theta + \theta_0)) \)
  - Preserves magnitude, adds to phase
- \( \exp(j0) = 1 \)
- \( \exp(jN\pi) = 1 \) where \( N \) is an even integer, and \( = -1 \) where \( N \) is an odd integer.
  - Real in both cases
The Discrete 2D Fourier Transform

$$\mathcal{F}[X] = F[u, v] = \frac{1}{WH} \sum_{n_x=0}^{W-1} \sum_{n_y=0}^{H-1} X[n_x, n_y] \exp\left(-j 2\pi \left(\frac{u n_x}{W} + \frac{v n_y}{H}\right)\right)$$

$$\exp(j \theta) = \cos \theta + j \sin \theta$$

- Defined for a single-channel / grayscale image $X$.
- $F$ is a “complex valued” array indexed by integers $u$, $v$.
- Each $F[u, v]$ depends on the intensities at all pixels.
The Discrete 2D Fourier Transform

\[ \mathcal{F}[X] = F[u, v] = \frac{1}{WH} \sum_{n_x=0}^{W-1} \sum_{n_y=0}^{H-1} X[n_x, n_y] \exp(-j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right)) \]

\[ \exp(j \theta) = \cos \theta + j \sin \theta \]


\[ \exp(-j 2\pi \left( \frac{(u + W) n_x}{W} + \frac{v n_y}{H} \right)) = \exp(-j 2\pi \left( \frac{u n_x}{W} + n_x + \frac{v n_y}{H} \right)) \]

\[ = \exp(-j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) - j 2n_x\pi) = \exp(-j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right)) \times \exp(-j 2n_x\pi) \]

\[ = \exp(-j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right)) \]
The Discrete 2D Fourier Transform

\[ \mathcal{F}[X] = F[u, v] = \frac{1}{WH} \sum_{n_x=0}^{W-1} \sum_{n_y=0}^{H-1} X[n_x, n_y] \exp\left(-j 2\pi \left(\frac{u n_x}{W} + \frac{v n_y}{H}\right)\right) \]

\[ \exp(j \theta) = \cos \theta + j \sin \theta \]

- Therefore, we typically store \( F[u, v] \) for \( u \in \{0, \ldots, W - 1\}, v \in \{0, \ldots, H - 1\} \).
- Can think of \( F[u, v] \) as a complex-valued “image” with the same number of pixels as \( X \).

Can be implemented fairly efficiently using the FFT algorithm: \( O(n \log n) \) (often, FFT is used to refer to the operation itself)
The Discrete 2D Fourier Transform Pair

\[ F[X] = F[u, v] = \frac{1}{WH} \sum_{n_x=0}^{W-1} \sum_{n_y=0}^{H-1} X[n_x, n_y] \exp \left( -j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right) \]

\[ F^{-1}[F] = X[n_x, n_y] = \sum_{u=0}^{W-1} \sum_{v=0}^{H-1} F[u, v] \exp \left( j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right) \]

- If \( X \) is real-valued, \( F[-u, -v] = F[W - u, H - v] = \bar{F}[u, v] \), where \( \bar{F} \) implies complex conjugate.
- \( F[0, 0] \) is often called the DC component. It is the average intensity of \( X \). It is real if \( X \) is real.
- Only \( WH \) independent “numbers” in \( F[u, v] \) (counting real and imaginary separately) if \( X \) is real.
- Parseval’s Theorem: (energy preserving upto constant factor)

\[
\sum_{u,v} \|F[u, v]\|^2 = \sum_{u,v} F[u, v] \bar{F}[u, v] = \frac{1}{WH} \sum_{n_x, n_y} \|X[n_x, n_y]\|^2
\]
FOURIER TRANSFORM

The Discrete 2D Fourier Transform Pair

\[ \mathcal{F}[X] = F[u, v] = \frac{1}{WH} \sum_{n_x=0}^{W-1} \sum_{n_y=0}^{H-1} X[n_x, n_y] \exp\left(-j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right) \]

\[ F'[u, v] = F[u, v] \times \exp\left(-j 2\pi \left( \frac{u t_x}{W} + \frac{v t_y}{H} \right) \right) \]

\[ \mathcal{F}^{-1}[F'] = ? = \sum_{u=0}^{W-1} \sum_{v=0}^{H-1} F'[u, v] \exp\left(j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right) \]

for a fixed integers \( t_x, t_y \)
FOURIER TRANSFORM

The Discrete 2D Fourier Transform Pair

\[
F[X] = F[u, v] = \frac{1}{WH} \sum_{n_x=0}^{W-1} \sum_{n_y=0}^{H-1} X[n_x, n_y] \exp(-j 2\pi \left(\frac{u n_x}{W} + \frac{v n_y}{H}\right))
\]

\[
F'[u, v] = F[u, v] \times \exp(-j 2\pi \left(\frac{u t_x}{W} + \frac{v t_y}{H}\right))
\]

\[
F^{-1}[F'] = X[n_x + t_x, n_y + t_y] = \sum_{u=0}^{W-1} \sum_{v=0}^{H-1} F'[u, v] \exp(j 2\pi \left(\frac{u n_x}{W} + \frac{v n_y}{H}\right))
\]

for a fixed integers \(t_x, t_y\)

A change in the phase of the Fourier coefficients, that is linear in \(u, v\), leads to a translation in the image.
**FOURIER TRANSFORM**

\[
F[u, v] = \frac{1}{WH} \sum_{n_x=0}^{W-1} \sum_{n_y=0}^{H-1} X[n_x, n_y] \exp\left(-j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right)
\]

\[
X[n_x, n_y] = \sum_{u=0}^{W-1} \sum_{v=0}^{H-1} F[u, v] \exp\left(j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right)
\]

- \( F[u, v] \) and \( X[n_x, n_y] \) are both 2D array of the same size \( W \times H \).
- \( F \) is complex-valued (while \( X \) is typically real-valued)
- These equations are linear. So both the Fourier transform and its inverse are linear operations.
- But each \( F[u, v] \) depends on values of \( X[n_x, n_y] \) at ALL locations (not local like a convolution).
- Note that the \( \exp(\cdot) \) expressions in both are similar—one has a negative sign inside the \( \exp \), indicating a complex conjugate.
- But, in one, we hold \((u, v)\) fixed and sum over \((n_x, n_y)\). In the other, vice-versa.