Topic 4:

Local analysis of image patches

- What do we mean by an image “patch”?
- Applications of local image analysis
- Visualizing 1D and 2D intensity functions
Local Image Patches

So far, we have considered pixels completely independently of each other (as RGB values or, as vectors \([R, G, B]\))

In reality, photos have a great deal of structure. This structure can be analyzed at a local level (e.g., small groups of nearby pixels) or a global one (e.g., entire image).
Local Image Patches

Qualitatively, we can think of many different types of patches in an image.

Patches corresponding to a “corner” in the image.
Local Image Patches

Qualitatively, we can think of many different types of patches in an image.

Patches corresponding to an “edge” in the image.
Local Image Patches

Qualitatively, we can think of many different types of patches in an image.

- Patches of uniform texture
Local Image Patches

Qualitatively, we can think of many different types of patches in an image.

Patches that originate from a single surface.
Local Image Patches

Qualitatively, we can think of many different types of patches in an image

Or patches with perceptually-significant “features”
Local Image Patches

When is a group of pixels considered a local patch?

The notion of a patch is relative. It can be a single pixel.
Local Image Patches

When is a group of pixels considered a local patch?

There is no answer to this question!

The notion of a patch is relative. It can be a single pixel.
Local Image Patches

When is a group of pixels considered a local patch?

There is no answer to this question!

The notion of a patch is relative. It can be the entire image.
Local Image Patches

We will begin with mathematical properties and methods that apply mostly to very small patches (e.g., 3x3)

... and eventually consider descriptions that apply to entire images
Topic 4:

Local analysis of image patches

- What do we mean by an image “patch”?
- Applications of local image analysis
- Visualizing 1D and 2D intensity functions
Patches: Why Do We Care?

Many applications...

- Recognition
- Inspection
- Video-based tracking
- Special effects
Face Recognition and Analysis

http://petapixel.com/2012/03/30/facial-recognition-software-guesses-age-based-on-a-photo/
Tracking

M. Zervos, H. BenShitrit and P. Fua, Real time multi-object tracking using multiple cameras
Editing & Manipulating Photos

Object removal from a photo

(Criminisi et al, CVPR 2003)
Editing & Manipulating Photos

Colorization of black and white photos

Original (B&W)  New (Color)

(Levin & Weiss, SIGGRAPH 2004)
Editing & Manipulating Photos

Scissoring objects from a photo

source images

composite image
Giving Photos a “Painted” Look

From P. Litwinowicz’s SIGGRAPH’97 paper “Processing Images and Videos for an Impressionist Effect”
Topic 4:

Local analysis of image patches

- What do we mean by an image “patch”?
- Applications of local image analysis
- Visualizing 1D and 2D image patches as intensity functions
Visualizing An Image as a Surface in 3D

Gray-scale image

A gray-scale image is like a function $I(x,y)$
Image $\leftrightarrow$ Surface in 3D

Gray-scale image $I(x, y)$

Surface $z = I(x, y)$

And we can visualize this function in 3D
Image $\Leftrightarrow$ Surface in 3D

Gray-scale image $I(x,y)$

Surface $z = I(x,y)$

- The height of the surface at $(x,y)$ is $I(x,y)$
- The surface contains point $(x,y,I(x,y))$
Image ↔ Surface in 3D

Gray-scale image

Image patch

The same applies to image patches
Image $\leftrightarrow$ Surface in 3D

Gray-scale image

Image patch

Surface patch $z = I(x, y)$

Patches have their own coordinate system.
BTW, notice image noise
Another way of visualizing image data is as a graph in 2D.
Image row or column ⇔ Graph in 2D

Gray-scale image

Graph in 2D

Point \((x_0, I(x_0, y_0))\)

And of course, we can do this for a 1D patch.
Today we’ll learn about

4.1. Today’s lecture is about modeling image data taking into account more than one (potentially noisy) single pixel.

We will focus on 1D patches.

Methods include:

- Computing derivatives of 1D patches using polynomial fitting via Least-squares, weighted least squares and RANSAC
where are we, and what will come after?

• Subtopics:
  1. Local analysis of 1D image patches (today)
  2. Local analysis of 2D curve patches
  3. Local analysis of 2D image patches
Local Analysis of Image Patches: Outline

As graph in 2D

As curve in 2D

As surface in 3D

\[ z = I(x, y) \]
Topic 4:

Local analysis of image patches

• Subtopics:
  1. Local analysis of 1D image patches
  2. Local analysis of 2D curve patches
  3. Local analysis of 2D image patches
Topic 4.1: Local analysis of 1D image patches

- Taylor series approximation of 1D intensity patches
- Estimating derivatives of 1D intensity patches
  - Least-squares fitting
  - Weighted least-squares fitting
  - Robust polynomial fitting: RANSAC
Topic 4.1:

Local analysis of 1D image patches

- Taylor series approximation of 1D intensity patches
  - Estimating derivatives of 1D intensity patches:
    Least-squares fitting
    Weighted least-squares fitting
    Robust polynomial fitting: RANSAC
Least-Squares Polynomial Fitting

Taylor approximation: Fit a polynomial to the pixel intensities in a patch

- All pixels contribute equally to estimate of derivative(s) at patch center (i.e., at $x=0$)
Taylor-Series Approximation of $I(x)$

If we knew the derivatives of $I(x)$ at $x=0$, we can approximate $I(x)$ using the Taylor Series:

$$I(x) = I(0) + x \cdot \frac{dI}{dx}(0) + \frac{1}{2} x^2 \frac{d^2I}{dx^2}(0) + \frac{1}{6} x^3 \frac{d^3I}{dx^3}(0) + \ldots + \frac{1}{n!} x^n \frac{d^nI}{dx^n}(0) + R_{n+1}(x)$$

<table>
<thead>
<tr>
<th>Order</th>
<th>Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0th</td>
<td>$I(0)$</td>
</tr>
<tr>
<td>1st</td>
<td>1st-order approx. of $I$</td>
</tr>
<tr>
<td>2nd</td>
<td>2nd-order approx. of $I$</td>
</tr>
<tr>
<td>n-th</td>
<td>n-th order approx</td>
</tr>
</tbody>
</table>

As graph in 2D
Taylor-Series Approximation of \( I(x) \)

If we knew the derivatives of \( I(x) \) at \( x=0 \), we can approximate \( I(x) \) using the Taylor Series:

\[
I(x) = I(0) + x \cdot \frac{d I}{dx}(0) + \frac{x^2}{2} \frac{d^2 I}{dx^2}(0) + \ldots + \frac{1}{n!} x^n \frac{d^n I}{dx^n}(0) + R_{n+1}(x)
\]

The residual \( R_{n+1}(x) \) satisfies

\[
\lim_{x \to 0} R_{n+1}(x) = 0
\]
Taylor-Series Approximation of $I(x)$

If we knew the derivatives of $I(x)$ at $x=0$, we can approximate $I(x)$ using the Taylor Series:

$$I(x) = I(0) + x \cdot \frac{dI}{dx}(0) + \frac{1}{2} x^2 \frac{d^2I}{dx^2}(0) + \cdots + \frac{1}{n!} x^n \frac{d^nI}{dx^n}(0) + R_{n+1}(x)$$

The approximation is best at the origin and degrades from there.
Taylor-Series Approximation of $I(x)$

As graph in 2D

The $n$-th order Taylor series expansion of $I(x)$, near the patch center ($x=0$) can then be written in matrix form as:

$$I(x) \approx \left[ 1 \quad x \quad \frac{1}{2}x^2 \quad \frac{1}{6}x^3 \quad \ldots \quad \frac{1}{n!}x^n \right] \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{d^2I}{dx^2}(0) \\ \vdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

Note that an approximated value for $I(x)$ will depend on $n+1$ coefficients: the intensity derivatives at $I(0)$.
Taylor-Series Approximation of $I(x)$

As graph in 2D

Example: $0^{th}$ order approximation

$$I(x) \approx \left[ 1 \right]$$

$$I(x) = I(0)$$
Taylor-Series Approximation of $I(x)$

As graph in 2D

Example: 1\textsuperscript{st} order approximation

$$I(x) \approx \left[ 1 \ x \right]$$

$$I(x) = I(0) + x \cdot \frac{dI}{dx}(0)$$
Taylor-Series Approximation of $I(x)$

As graph in 2D

Example: 2\textsuperscript{nd} order approximation

$$I(x) \approx \begin{bmatrix} 1 & x & \frac{1}{2}x^2 \end{bmatrix}$$

$$I(x) = I(0) + x \cdot \frac{dI}{dx}(0) + \frac{x^2}{2} \cdot \frac{d^2I}{dx^2}(0)$$
Taylor-Series Approximation of $I(x)$

As graph in 2D

\[
I(x) \approx \left[ 1 \times \frac{1}{2}x^2 \frac{1}{6}x^3 \ldots \frac{1}{n!}x^n \right]
\]

\[
\begin{bmatrix}
I(0) \\
\frac{dI}{dx}(0) \\
\frac{d^2I}{dx^2}(0) \\
\vdots \\
\frac{d^nI}{dx^n}(0)
\end{bmatrix}
\]
Taylor-Series Approximation of $I(x)$

But do we know the derivatives?

\[
\begin{pmatrix}
I^{(0)} \\
\frac{d I}{d x}^{(0)} \\
\frac{d^2 I}{d x^2}^{(0)} \\
\vdots \\
\frac{d^n I}{d x^n}^{(0)}
\end{pmatrix}
\]
Taylor-Series Approximation of I(x)

But do we know the derivatives?

No, but we can estimate them!
Taylor-Series Approximation of $I(x)$

And can we estimate them for the entire row?
Taylor-Series Approximation of I(x)

And can we estimate them for the entire row? Yes, but pixel by pixel.

In fact...
Applying the same operation on multiple patches

A “sliding window” algorithm is a common approach to patch-based operations

The algorithm goes as follows:
A “sliding window” algorithm is a common approach to patch-based operations.

The algorithm goes as follows:

1. Define a “pixel window” using a window size and a window center.

Applying the same operation on multiple patches
A “sliding window” algorithm is a common approach to patch-based operations.

The algorithm goes as follows:

1. Define a “pixel window” using a window size and a window center.
2. Apply whatever operation in mind to that patch.
3. Move the window center one pixel to define a new window.
4. Repeat steps 1-3.

Applying the same operation on multiple patches.
A “sliding window” algorithm is a common approach to patch-based operations.

The algorithm goes as follows:

1. Define a “pixel window” using a window size and a window center.
2. Apply whatever operation in mind to that patch.
3. Move the window center one pixel to define a new window.
4. Repeat steps 1-3.
A "sliding window" algorithm is a common approach to patch-based operations.

The algorithm goes as follows:

1. Define a “pixel window” using a window size and a window center.
2. Apply whatever operation in mind to that patch.
3. Move the window center one pixel to define a new window.
4. Repeat steps 1-3.
A “sliding window” algorithm is a common approach to patch-based operations.

The algorithm goes as follows:

1. Define a “pixel window” using a window size and a window center.
2. Apply whatever operation in mind to that patch.
3. Move the window center one pixel to define a new window.
4. Repeat steps 1-3.
A “sliding window” algorithm is a common approach to patch-based operations.

The algorithm goes as follows:

1. Define a “pixel window” using a window size and a window center.
2. Apply whatever operation in mind to that patch.
3. Move the window center one pixel to define a new window.
4. Repeat steps 1-3.

Applying the same operation on multiple patches...
Estimating Derivatives For Image Row r

“Sliding window” algorithm:

• Define a “pixel window” centered at pixel (w,r)

• Fit n-degree poly to window’s intensities (usually n=1 or 2)

• Assign the poly’s derivatives at x=0 to pixel at window’s center

• “Slide” window one pixel over, so that it is centered at pixel (w+1,r)

• Repeat 1-4 until window reaches right image border
Estimating Derivatives For Image Row r

“Sliding window” algorithm:

- Define a “pixel window” centered at pixel \((w,r)\)
- Fit n-degree poly to window’s intensities (usually n=1 or 2)
- Assign the poly’s derivatives at \(x=0\) to pixel at window’s center
- “Slide” window one pixel over, so that it is centered at pixel \((w+1,r)\)
- Repeat 1-4 until window reaches right image border
Estimating Derivatives For Image Row r

“Sliding window” algorithm:

• Define a “pixel window” centered at pixel (w, r)

• Fit n-degree poly to window’s intensities (usually n=1 or 2)

• Assign the poly’s derivatives at x=0 to pixel at window’s center

• “Slide” window one pixel over, so that it is centered at pixel (w+1, r)

• Repeat 1-4 until window reaches right image border
Estimating Derivatives For Image Row r

“Sliding window” algorithm:
- Define a “pixel window” centered at pixel \((w, r)\)
- Fit \(n\)-degree poly to window’s intensities (usually \(n=1\) or 2)
- Assign the poly’s derivatives at \(x=0\) to pixel at window’s center
- “Slide” window one pixel over, so that it is centered at pixel \((w+1, r)\)
- Repeat 1-4 until window reaches right image border
Estimating Derivatives For Image Row r

“Sliding window” algorithm:

- Define a “pixel window” centered at pixel (w,r)
- Fit n-degree poly to window’s intensities (usually n=1 or 2)
- Assign the poly’s derivatives at x=0 to pixel at window’s center
- “Slide” window one pixel over, so that it is centered at pixel (w+1,r)
- Repeat 1-4 until window reaches right image border
Estimating Derivatives For Image Row $r$

“Sliding window” algorithm:
- Define a “pixel window” centered at pixel $(w,r)$
- Fit $n$-degree poly to window’s intensities (usually $n=1$ or $2$)
- Assign the poly’s derivatives at $x=0$ to pixel at window’s center
- “Slide” window one pixel over, so that it is centered at pixel $(w+1,r)$
- Repeat 1-4 until window reaches right image border
Topic 4.1: Local analysis of 1D image patches

- Taylor series approximation of 1D intensity patches
- Estimating derivatives of 1D intensity patches:
  - Least-squares fitting
  - Weighted least-squares fitting
  - Robust polynomial fitting: RANSAC
Taylor-Series Approximation of $I(x)$

As graph in 2D

How to estimate the Taylor series approximation from image data?
Taylor-Series Approximation of $I(x)$

As graph in 2D

Surprise!

The $n^{th}$ degree Taylor approximation can be estimated using a linear system of equations (which we can represent in matrix form).

This is Least Squares!
Taylor-Series Approximation of $I(x)$

As graph in 2D

We know that the Taylor series is:

$$I(x) = I(0) + x \cdot \frac{dI}{dx}(0) + \frac{1}{2} x^2 \frac{d^2I}{dx^2}(0) + \ldots + \frac{1}{n!} x^n \frac{d^nI}{dx^n}(0) + R_{n+1}(x)$$
Taylor-Series Approximation of $I(x)$

As graph in 2D

We know that the Taylor series is:

\[
I(x) = I(0) + x \cdot \frac{dI}{dx}(0) + \frac{1}{2} x^2 \frac{d^2I}{dx^2}(0) + \cdots + \frac{1}{n!} x^n \frac{d^nI}{dx^n}(0)
\]

The derivatives are unknown.
Taylor-Series Approximation of \( I(x) \)

As graph in 2D

We know that the Taylor series is:

\[
I(x) = I(0) - x \frac{dI(0)}{dx} + \frac{1}{2} x^2 \frac{d^2I(0)}{dx^2} - \ldots + \frac{1}{n!} x^n \frac{d^nI(0)}{dx^n}
\]

But the coefficients are known
Taylor-Series Approximation of $I(x)$

As graph in 2D

The n-th order Taylor series expansion of $I(x)$, near the patch center ($x=0$) can then be written in matrix form as:

$$I(x) \approx \left[ 1 \ x \ \frac{1}{2!}x^2 \ \frac{1}{6!}x^3 \ \ldots \ \frac{1}{n!}x^n \right] \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{d^2I}{dx^2}(0) \\ \ldots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$
The n-th order Taylor series expansion of $I(x)$, near the patch center ($x=0$) can then be written in matrix form as:

$$I(x) \approx \begin{bmatrix} 1 & x & \frac{1}{2}x^2 & \frac{1}{6}x^3 & \cdots & \frac{1}{n!}x^n \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{d^2I}{dx^2}(0) \\ \cdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

Patch (2w+1 pixels)

\[\begin{array}{cccc}
\vdots & x=-w & x=0 & x=2 & x=w \\
\hline
S & \ddots & \ddots & \ddots & \ddots \\
\frac{dI}{dx}(0) & \frac{d^2I}{dx^2}(0) & \cdots & \frac{d^nI}{dx^n}(0)
\end{array}\]
Taylor-Series Approximation of $I(x)$

As graph in 2D

The n-th order Taylor series expansion of $I(x)$, near the patch center ($x=0$) can then be written in matrix form as:

$$I(x) \approx \begin{bmatrix} 1 & x & \frac{1}{2}x^2 & \frac{1}{6}x^3 & \cdots & \frac{1}{n!}x^n \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{d^2I}{dx^2}(0) \\ \vdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

for $x \in (-W, W)$

2w+1 equations to estimate n+1 unknowns
Least-Squares Polynomial Fitting of $I(x)$

As graph in 2D

The equations define the system:

$$ I_{(2w+1)\times1} = X_{(2w+1)\times(n+1)} d_{(n+1)\times1} $$

Intensities (known) \quad \uparrow \quad \text{positions (known)} \quad \uparrow \quad \text{derivatives (unknown)}

Solving linear system in terms of $d$ minimizes the "fit error"

$$ \| I - Xd \|^2 $$
Least-Squares Polynomial Fitting of $I(x)$

We could then do $v=Xd$ to get an estimate for all pixels in the patch in $(-w, \ldots, 0, \ldots, w)$.
Least-Squares Polynomial Fitting of $I(x)$

Example

\[ \| I - Xd \|^2 \]

Solution $d$ is called a **least-squares fit**

- This solution minimizes the 2-norm (i.e., the length) of the error vector $(I - v)$:
  \[
  \left( \sum_{i=1}^{2w+1} (I_i - v_i)^2 \right)^{1/2}
  \]
0th-Order (Constant) Estimation of I(x)

Special case:

\[ I \]

\[ I_1, I_2, I_3, I_4, I_5, I_6, I_7 \]

\[ -3, -2, -1, 0, 1, 2, 3 \]

- Solution minimizes
  \[ \sum_{i=1}^{2w+1} (I_i - \bar{d}_i)^2 \]

- Solution is the mean intensity of the patch:
  \[ \bar{d}_i = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i \]

Patch (2w+1 pixels)

\[
\begin{array}{ccccccc}
  x=-w & x=-w+1 & \cdots & x=0 & x=1 & \cdots & x=w \\
  I_1 & I_2 & \cdots & I_{2w+1} & \cdots & \cdots & I_{2w+1} \\
\end{array}
\]

\[
I_{(2w+1) \times 1} = X_{(2w+1) \times 1} \cdot d_{d \times 1}
\]

\[
\begin{bmatrix}
I_1 \\
I_2 \\
\vdots \\
I_{2w+1}
\end{bmatrix} =
\begin{bmatrix}
1 \\
1 \\
\vdots \\
1
\end{bmatrix}
\cdot
\begin{bmatrix}
d_1 \\
\end{bmatrix}
\]

\[
I(0) = \bar{I}
\]

Solving linear system in terms of \(d\) minimizes the "fit error"

\[ \| I - Xd \|^2 \]
0th-Order (Constant) Estimation of $I(x)$

Special case:

- Solution minimizes
  $$\sum_{i=1}^{2w+1} (I_i - d_i)^2$$

- Solution is the mean intensity of the patch:
  $$d_i = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i$$
0th-Order (Constant) Estimation of I(x)

Special case:

Proof

- Let \( E(x) = \sum_{i=1}^{2w+1} (I_i - x)^2 \)

- Solution minimizes \( \sum_{i=1}^{2w+1} (I_i - d_i)^2 \)

- Solution is the mean intensity of the patch:

\[
    d_i = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i
\]
0th-Order (Constant) Estimation of \( I(x) \)

**Proof**

- Let \( E(x) = \sum_{i=1}^{2w+1} (I_i - x)^2 \)
- At the minimum of \( E(x) \), the derivative \( \frac{d}{dx} E(x) \) must be zero.

- Solution minimizes
  \[ \sum_{i=1}^{2w+1} (I_i - d_i)^2 \]

- Solution is the mean intensity of the patch:
  \[ d_i = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i \]
0th-Order (Constant) Estimation of I(x)

Special case:

- Solution minimizes
  \[ \sum_{i=1}^{2w+1} (I_i - d_i)^2 \]
  \[ \sum_{i=1}^{2w+1} I_i \]

- Solution is the mean intensity of the patch:
  \[ d_i = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i \]

\[ \frac{2w+1}{2w+1} \sum_{i=1}^{2w+1} I_i \]

**Proof**

- Let \( E(x) = \sum_{i=1}^{2w+1} (I_i - x)^2 \)
  \[ \sum_{i=1}^{2w+1} \]
  \[ \sum_{i=1}^{2w+1} \]

- At the minimum of \( E(x) \), the derivative \( \frac{d}{dx} E(x) \) must be zero:

  \[ \frac{d}{dx} E(x) = \sum_{i=1}^{2w+1} \frac{d}{dx} \left[ (I_i - x)^2 \right] \]
  \[ \sum_{i=1}^{2w+1} 2(I_i - x) \cdot (-1) \]
  \[ = -2 \left( \sum_{i=1}^{2w+1} (I_i - x) \right) \]
  \[ = -2 \left( \sum_{i=1}^{2w+1} I_i \right) \cdot x \]
0th-Order (Constant) Estimation of I(x)

Special case:

• Solution minimizes
  \[
  \sum_{i=1}^{2w+1} (I_i - d_i)^2
  \]

• Solution is the mean intensity of the patch:
  \[
  d_i = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i
  \]

Proof:

• Let \( E(x) = \sum_{i=1}^{2w+1} (I_i - x)^2 \)

• At the minimum of \( E(x) \), the derivative \( \frac{d}{dx} E(x) \) must be zero

\[
\frac{d}{dx} E(x) = \sum_{i=1}^{2w+1} \frac{d}{dx} \left[ (I_i - x)^2 \right]
\]

\[
= \sum_{i=1}^{2w+1} 2 (I_i - x) \cdot (-1)
\]

\[
= -2 \left[ \sum_{i=1}^{2w+1} (I_i - x) \right]
\]

\[
= -2 \left( \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i \right) + 2(2w+1) \times
\]

\[
\frac{d}{dx} E(x) = 0 \iff x = \frac{1}{2w+1} \left( \sum_{i=1}^{2w+1} I_i \right)
\]
1st-Order (Linear) Estimation of I(x)

Special case:

- Solution minimizes sum of "metrical" distances between line and image intensities.

- Gives us an estimate of $I(0)$ and $\frac{dI(0)}{dx}$ (i.e. value & derivative at 0).
2\textsuperscript{nd}-Order (Quadratic) Estimation of I(x)

Special case:

- Fits a parabola/ hyperbola/ ellipse
- Gives us an estimate of 1\textsuperscript{st} & 2\textsuperscript{nd} image derivative at patch center

\[
\begin{bmatrix}
I_1 \\
I_2 \\
\vdots \\
I_n
\end{bmatrix} = 
\begin{bmatrix}
1 & -3 & 9/2 \\
1 & -2 & z \\
1 & 3 & 9/2
\end{bmatrix} 
\begin{bmatrix}
d_1 \\
d_2 \\
d_3
\end{bmatrix} 
\rightarrow \frac{d^2 I}{dx^2}(0)
\]
2\textsuperscript{nd}-Order (Quadratic) Estimation of I(x)

Note how all pixels in the window contribute equally to the estimate around the center of the window!
Topic 4.1: Local analysis of 1D image patches

• Taylor series approximation of 1D intensity patches
• Estimating derivatives of 1D intensity patches:
  • Least-squares fitting
  • Weighted least-squares fitting
  • Robust polynomial fitting: RANSAC
Weighted Least Squares Polynomial Fitting

Scenario #1:

- Fit polynomial to ALL pixel intensities in a patch
Scenario #2:

- Fit polynomial to all the pixel intensities in the patch
- Pixels contribute to estimate of derivative(s) at center according to a weight function $\Omega(x)$
Q: Will the estimate of \( \frac{dI}{dx} (0) \) be the same or different in the two cases below? (Assume a 1st order fit.)

case #1

\[ -w \quad 0 \quad x \quad w \]

best-fit line

---

case #2

\[ -w \quad 0 \quad x \quad w \]

\( < \) best-fit line
Polynomial Fitting: A Linear Formulation

Q: Will the estimate of \( \frac{dI}{dx} (0) \) be the same or different in the two cases below? (Assume a 1st order fit)

Case #1

Ans: the values will differ because all patch pixels contribute equally to the linear system!
Weighted Least-Squares Estimation of $I(x)$

Q: How can we bias our estimate of $\frac{d}{dx} I(o)$ toward the patch center?

**Case #1**

- Weight function $\omega(x) = e^{-x^2}$

**Case #2**

Idea: Weigh pixels near center more than pixels away from it.
Weighted Least-Squares Estimation of $I(x)$

Q: How can we bias our estimate of $\frac{d}{dx} I(x)$ toward the patch center?

**Case #1**

New equation for pixel $x$:

$$\Omega(x) I(x) = \Omega(x) \begin{bmatrix} 1 & x & \frac{1}{2} x^2 & \frac{1}{6} x^3 & \ldots & \frac{1}{n!} x^n \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{d^2I}{dx^2}(0) \\ \ldots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

**Case #2**

Weight function $\Omega(x)$ (e.g., $\Omega(x) = e^{-x^2}$)
QA: How can we bias our estimate of $\frac{dI(x)}{dx}$ toward the patch center?

- **Case #1**
- **Case #2**

Weight function $\omega(x)$ (e.g., $\omega(x) = e^{-x^2}$)

Patch (2w+1 pixels):

$$
\begin{bmatrix}
I_1 & I_2 & \cdots & I_{w+1} & \cdots & I_{2w+1}
\end{bmatrix}
$$

$$
\begin{bmatrix}
\omega_1 & 0 & \cdots & 0 & \cdots & \omega_{2w+1}
\end{bmatrix} I = \begin{bmatrix} \omega_1 & \cdots & \omega_{2w+1} \end{bmatrix} X d
$$

Solution $d$ minimizes the norm

$$
\left\| \begin{bmatrix}
\omega_1 & 0 & \cdots & 0 & \cdots & \omega_{2w+1}
\end{bmatrix} (I - Xd) \right\|^2
$$
We could then do $v = Xd$ to get an estimate of $I(x)$ for all pixels in the patch in $(-w, \ldots, 0, \ldots, w)$.

- This solution minimizes the 2-norm (i.e., the length) of the weighted error vector: $$(\sum_{i=1}^{2w+1} \omega_i (I_i - v_i)^2)^{1/2}$$
Topic 4.1:
Local analysis of 1D image patches

- Taylor series approximation of 1D intensity patches
- Estimating derivatives of 1D intensity patches:
  - Least-squares fitting
  - Weighted least-squares fitting
  - Robust polynomial fitting: RANSAC
Robust Polynomial Fitting

Scenario #3:

- Fit polynomial only to SOME pixel intensities in a patch (the “inliers”)
Robust Polynomial Fitting

But how can we tell between inliers and outliers?
We can’t. At least not before we fit a model.
Here’s our problem: find the inliers, fit a polynomial to them:

Given:
- $n =$ degree of poly
- $p =$ fraction of inliers
- $t =$ fit threshold
- $p_s =$ success probability

![Graph showing polynomial fitting using RANSAC](image-url)
Example: Line fitting using RANSAC (i.e., n=2 unknown polynomial coefficients)

- Step 1: Randomly choose n pixels from the patch
RANSAC Algorithm

Step 2: Fit the poly using the chosen pixels/intensities
RANSAC Algorithm

Step 3: Count pixels with vertical distance < threshold t

\[ \#\text{pixels} = 5 \]
• Step 4: If there aren’t “enough” such pixels, REPEAT
(not more than K times)

\[ w = 7 \]
\[ \# \text{pixels} = 5 \]
\[ p = 0.85 \]

Example:
• Suppose we know that there at most 2 outliers pixels.
  Then,
  \[ p = \frac{2w+1-2}{2w+1} = \frac{13}{15} \approx 85\% \]
RANSAC Algorithm

How about these two?
RANSAC Algorithm

Step 4: If there are “enough” such pixels, STOP
Label them as “inliers” & do a least-squares fit to the INLIER pixels only

\[ w = 7 \]
\[ \# \text{pixels} = 13 \]
\[ p = 0.85 \]

\[ p(2w+1) = 0.85 \cdot 13 = 13 \implies \text{success!} \]
RANSAC Algorithm

Step 4: If there are “enough” such pixels, STOP
Label them as “inliers” & do a least-squares fit
to the INLIER pixels only
RANSAC Algorithm

Eventually, after “enough” trials, there must be some likelihood of having chosen \(n+1\) inliers to fit the model.
RANSAC Algorithm

Eventually, after “enough” trials, there must be some likelihood of having chosen \( n+1 \) inliers to fit the model.

How many trials are enough then?
RANSAC Algorithm

Given:
- \( n = \text{degree of poly} \)
- \( p = \text{fraction of inliers} \)
- \( t = \text{fit threshold} \)
- \( p_s = \text{success probability} \)

Repeat at most \( K \) times:
1. Randomly choose \( n+1 \) pixels
2. Fit \( n \)-degree poly
3. Count pixels whose vertical distance from poly is \(< t\)
4. If there are at least \((2w+1)p\) pixels, EXIT LOOP
   a. Label them as inliers
   b. Fit \( n \)-degree poly to all inlier pixels

Q: What should \( K \) be?
- Probability we choose an inlier pixel: \( p \)
- Probability we choose \((n+1)\) inlier pixels: \( p^{n+1} \)
- Prob at least 1 outlier chosen: \( 1 - p^{n+1} \)
- Prob at least 1 outlier chosen in all \( K \) trials: \( (1 - p^{n+1})^K \)
**RANSAC Algorithm**

**Given:**
- $n = \text{degree of poly}$
- $p = \text{fraction of inliers}$
- $t = \text{fit threshold}$
- $p_s = \text{success probability}$

**Repeat at most $K$ times:**

1. Randomly choose $n+1$ pixels
2. Fit $n$-degree poly
3. Count pixels whose vertical distance from poly is $< t$
4. If there are at least $(2w+1)p$ pixels, EXIT LOOP
   - a. Label them as inliers
   - b. Fit $n$-degree poly to all inlier pixels

**Q: What should $K$ be?**

- Probability we choose an inlier pixel: $P$
- Probability we choose $(n+1)$ inlier pixels: $p^{n+1}$
- Prob at least 1 outlier chosen: $1 - P$
- Prob at least 1 outlier chosen in all $K$ trials: $(1 - P^{n+1})^K$
- Failure probability: $(1 - P^{n+1})^K$
- Success probability: $p_s = 1 - (1 - P^{n+1})^K$
- By taking logs on both sides

$$K = \frac{\log (1 - p_s)}{\log (1 - P^{n+1})}$$