Elements of Image (Pre)-Processing and Feature Detection

Acknowledgements: slides from Steven Seitz, Aleosha Efros, David Forsyth, and Gonzalez & Woods
Image Processing Basics

- **Point Processing**
  - gamma correction
  - window-center correction
  - histogram equalization

- **Filtering** (linear and non-linear)
  - mean, Gaussian, and median filters
  - image gradients, Laplacian
  - normalized cross-correlation (NCC)
  - etc…: Fourier, Gabor, wavelets (Szeliski, Sec 3.4-3.5)

- **Other features**
  - Harris corners, MOPS, SIFT, etc.

Extra Reading:
- Szeliski, Sec 3.1: intensities, colors
- Szeliski, Sec 3.2-3.3: contrast edges, texture, templates, patches
- Szeliski, Sec. 4.1: Harris corners, MOPS, SIFT, etc.
Summary of image transformations

- An **image processing** operation (or transformation) typically defines a new image $g$ in terms of an existing image $f$.

*Examples:*
An **image processing** operation (or transformation) typically defines a new image $g$ in terms of an existing image $f$.

**Examples:**

- **Geometric (domain) transformation:**
  - What kinds of operations $g(x, y) = f(t_x(x, y), t_y(x, y))$
Summary of image transformations

- An image processing operation (or transformation) typically defines a new image $g$ in terms of an existing image $f$.

**Examples:**

- **Geometric (domain) transformation:**
  - What kinds of operations
  
  $$g(x, y) = f(t_x(x, y), t_y(x, y))$$

- **Range transformation:**
  - What kinds of operations
  
  $$g(x, y) = t(f(x, y))$$
Summary of image transformations

An image processing operation (or transformation) typically defines a new image $g$ in terms of an existing image $f$.

**Examples:**

- **Geometric (domain) transformation:**
  - What kinds of operations
  
  \[ g(x, y) = f(t_x(x, y), t_y(x, y)) \]

- **Range transformation:**
  - What kinds of operations
  
  \[ g(x, y) = t(f(x, y)) \]

- **Filtering** also generates new images from an existing image
  
  \[ g(x, y) = \int_{|u|<\varepsilon} \int_{|v|<\varepsilon} h(u, v) \cdot f(x-u, y-v) \, du \, dv \]

  - more on filtering later
Point Processing

\[ g(x, y) = t(f(x, y)) \]

for each original image intensity value \( I \) function \( t(\cdot) \) returns a transformed intensity value \( t(I) \).

\[ I' = t(I) \]

**Important:** every pixel is for itself
- spatial information is ignored!

**What can point processing do?**
(we will focus on grey scale images, see Szeliski 3.1 for examples of point processing for color images)
Point Processing:

Examples of gray-scale transforms $t$

$$I' = t(I)$$

**FIGURE 3.3** Some basic gray-level transformation functions used for image enhancement.
Point Processing:

Negative

\[ t(I) = 255 - I \]

\[ g(x, y) = t(f(x, y)) = 255 - f(x, y) \]
**Point Processing:**

**Power-law transformations**

*FIGURE 3.6* Plots of the equation $s = cr^\gamma$ for various values of $\gamma$ ($c = 1$ in all cases).
Point Processing:

Gamma Correction

**FIGURE 3.7**
(a) Linear-wedge gray-scale image. 
(b) Response of monitor to linear wedge. 
(c) Gamma-corrected wedge. 
(d) Output of monitor.

Gamma Measuring Applet:  
http://www.cs.berkeley.edu/~efros/java/gamma/gamma.html
Point Processing:
Enhancing Image via Gamma Correction

FIGURE 3.9
(a) Aerial image.
(b)–(d) Results of applying the transformation in Eq. (3.2-3) with
$c = 1$ and
$\gamma = 3.0, 4.0, \text{and} 5.0, \text{respectively.}
(\text{Original image for this example courtesy of NASA.})
Point Processing:
Understanding Image Histograms

Image Brightness

Image Contrast

probability of intensity $i : p(i) = \frac{n_i}{n}$

---number of pixels with intensity $i$

---total number of pixels in the image
Point Processing:

Contrast Stretching

\[ T(r) \]

\[ (r_1, s_1) \]

\[ (r_2, s_2) \]

\[ 0 \]

\[ \frac{L}{4} \]

\[ \frac{L}{2} \]

\[ \frac{3L}{4} \]

\[ L - 1 \]

\[ \frac{L}{2} \]

\[ \frac{L}{2} \]

\[ \frac{L}{2} \]

\[ \frac{L}{2} \]

FIGURE 3.10
Contrast stretching. (a) Form of transformation function. (b) A low-contrast image. (c) Result of contrast stretching. (d) Result of thresholding. (Original image courtesy of Dr. Roger Hadey, Research School of Biological Sciences, Australian National University, Canberra, Australia.)
Point Processing:

Contrast Stretching

<table>
<thead>
<tr>
<th>Original images</th>
<th>Histogram corrected images</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.jpg" alt="Image 1" /></td>
<td><img src="image2.jpg" alt="Image 2" /></td>
</tr>
<tr>
<td><img src="image3.jpg" alt="Image 3" /></td>
<td><img src="image4.jpg" alt="Image 4" /></td>
</tr>
</tbody>
</table>

1) 
2)
Point Processing:

Contrast Stretching

Original images

Histogram corrected images

3)

4)
One way to automatically select transformation $t$:

**Histogram Equalization**

**FIGURE 3.18**  
Transformation functions (1) through (4) were obtained from the histograms of the images in Fig.3.17(a), using Eq. (3.3-8).

$$t(i) = \sum_{j=0}^{i} p(j) = \sum_{j=0}^{i} \frac{n_j}{n}$$

$=$ cumulative distribution of image intensities

...see Gonzalez and Woods, Sec3.3.1, for more details
Point processing

Histogram Equalization

\[ t(i) = \sum_{j=0}^{i} p(j) = \sum_{j=0}^{i} \frac{n_j}{n} = \text{cumulative distribution of image intensities} \]

Why does that work?

Answer in probability theory:

\( I \) – random variable with \textit{probability} distribution \( p(i) \) over \( i \) in \([0,1]\)

If \( t(i) \) is a \textit{cumulative} distribution of \( I \) then

\( I' = t(I) \) – is a random variable with \textit{uniform} distribution over its range \([0,1]\)

That is, transform image \( I' \) will have a uniformly-spread histogram (good contrast)
Point Processing:

**Window-Center adjustment**

![Graph showing the relationship between input gray level and output gray level. The graph illustrates a linear mapping between the high dynamic range image and the monitor's dynamic range.](image)
Point Processing:

Window-Center adjustment

Output gray level (monitor's dynamic range)

Input gray level (high dynamic range image)
Point Processing:

Window-Center adjustment

- Input gray level
- Output gray level

- Window
- Center

- 0
- 256
- 60000
Point Processing:

Window-Center adjustment

Window = 4000
Center = 500
Point Processing:

Window-Center adjustment

Window = 2000
Center = 500
Point Processing:

Window-Center adjustment

Window = 800
Center = 500
Point Processing: 

Window-Center adjustment

Window = 0
Center = 500

If \( \text{window}=0 \) then we get binary image *thresholding*
Point Processing:

Window-Center adjustment

Window = 800
Center = 500

Window = 800
Center = 1160
Neighborhood Processing (or filtering)

Q: What happens if I reshuffle all pixels within the image?

A: It’s histogram won’t change.
No point processing will be affected…

Images contain a lot of “spatial information”

Readings: Szeliski, Sec 3.2-3.3
Let’s start with 1D image (a signal): \( f[i] \)

A very general and useful class of transforms are the **linear transforms** of \( f \), defined by a matrix \( M \)

\[
M[i, j] f[i] = g[i] = \sum_{j=1} M[i, j] f[j]
\]
Neighborhood Processing (filtering)

Linear image transforms

Let's start with 1D image (a signal): $f[i]$
Neighborhood Processing (filtering)

Linear image transforms

Let’s start with 1D image (a signal): \( f[i] \)

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\[
\frac{1}{2}
\begin{bmatrix}
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]
**Neighborhood Processing (filtering)**

**Linear shift-invariant filters**

This pattern is very common:
- same entries in each row
- all non-zero entries near the diagonal

It is known as a **linear shift-invariant filter** and is represented by a **kernel** (or **mask**) $h$:

$$h[i] = [a \ b \ c]$$

and can be written (for kernel of size $2k+1$) as:

$$g[i] = \sum_{u=-k}^{k} h[u] \cdot f[i+u]$$

The above allows negative filter indices. When you implement need to use: $h[u+k]$ instead of $h[u]$. 

$$g = M \cdot f$$
Neighborhood Processing (filtering)

2D linear transforms

We can do the same thing for 2D images by concatenating all of the rows into one long vector (in a “raster-scan” order):

\[ \hat{f}[i] = f[\lfloor i/m \rfloor, i\%m] \]
Neighborhood Processing (filtering)

2D filtering

A 2D image \( f[i,j] \) can be filtered by a 2D kernel \( h[u,v] \) to produce an output image \( g[i,j] \):

\[
g[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] \cdot f[i + u, j + v]
\]

This is called a **cross-correlation** operation and written:

\[
g = h \circ f
\]

\( h \) is called the “filter,” “kernel,” or “mask.”
Neighborhood Processing (filtering)

2D filtering

A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

\[
g[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u, v] \cdot f[i-u, j-v]
\]

It is written:

\[
g = h \ast f = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[-u,-v] \cdot f[i+u, j+v]
\]

How does convolution differ from cross-correlation?

If \( h[u, v] = h[-u,-v] \) then there is no difference between convolution and cross-correlation.

Convolution has additional “technical” properties: **commutativity**, **associativity**. Also, “nice” properties wrt Fourier analysis. (see Szeliski Sec 3.2, Gonzalez and Woods Sec. 4.6.4)
2D filtering

Noise

Filtering is useful for noise reduction...

(side effects: blurring)

Common types of noise:

- **Salt and pepper noise**: random occurrences of black and white pixels
- **Impulse noise**: random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution
Practical noise reduction

How can we “smooth” away noise in a single image?

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>130</td>
<td>110</td>
<td>120</td>
<td>110</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>110</td>
<td>90</td>
<td>100</td>
<td>90</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>130</td>
<td>100</td>
<td>90</td>
<td>130</td>
<td>110</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>120</td>
<td>100</td>
<td>130</td>
<td>110</td>
<td>120</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>110</td>
<td>80</td>
<td>120</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Neighborhood Processing (filtering)

Mean filtering

\[ F[x, y] \]

\[ G[x, y] \]
### Neighborhood Processing (filtering)

#### Mean filtering

**$F[x, y]$**

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>90</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>90</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**$G[x, y]$**

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>20</td>
<td>40</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>60</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>60</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>50</td>
<td>80</td>
<td>80</td>
<td>90</td>
<td>60</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>50</td>
<td>80</td>
<td>80</td>
<td>90</td>
<td>60</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>20</td>
<td>30</td>
<td>50</td>
<td>50</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Effect of mean filters

3x3

5x5

7x7

Gaussian noise

Salt and pepper noise
Mean kernel

What’s the kernel for a 3x3 mean filter?

\[ F[x, y] \]

\[ H[u, v] \]
Neighborhood Processing (filtering)

Gaussian Filtering

- A Gaussian kernel gives less weight to pixels further from the center of the window.

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 0 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 90 & 0 & 90 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

\[
\frac{1}{16} \cdot \begin{pmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{pmatrix}
\]

\[
H[u, v]
\]

\[
h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}
\]

This kernel is an approximation of a Gaussian function:
Neighborhood Processing (filtering)

Mean vs. Gaussian filtering
Neighborhood Processing (filtering)

Median filters

- A Median Filter operates over a window by selecting the median intensity in the window.

- What advantage does a median filter have over a mean filter?

- Is a median filter a kind of convolution?

  - No, median filter is an example of non-linear filtering
Comparison: salt and pepper noise

3x3

5x5

7x7
Comparison: Gaussian noise

3x3

5x5

7x7
Reading: Forsyth & Ponce, 8.1-8.2

Differentiation and convolution

- Recall

\[
\frac{\partial}{\partial x} f = \lim_{\varepsilon \to 0} \left( \frac{f(x + \varepsilon, y) - f(x, y)}{\varepsilon} \right)
\]

- Now this is linear and shift invariant, so must be the result of a convolution.

- We could approximate this as

\[
\frac{\partial}{\partial x} f \approx \frac{f(x_{i+1}, y) - f(x_{i-1}, y)}{2 \cdot \Delta x}
\]

\[= \nabla_x * f \quad \text{(convolution)}\]

with kernel

\[
\begin{vmatrix}
\frac{1}{2\Delta x} & 0 & 0 & 0 \\
1 & 0 & -1 \\
0 & 0 & 0 \\
\end{vmatrix}
\]

\[\nabla_x [u, v]\]

sometimes this may not be a very good way to do things, as we shall see
Differentiation and convolution

Recall

\[ \frac{\partial}{\partial x} f = \lim_{\varepsilon \to 0} \left( \frac{f(x + \varepsilon, y) - f(x, y)}{\varepsilon} \right) \]

Now this is linear and shift invariant, so must be the result of a convolution.

We could approximate this as

\[ \frac{\partial}{\partial x} f \approx \frac{f(x_{i+1}, y) - f(x_{i-1}, y)}{2 \cdot \Delta x} \]

\[ = \nabla_x \ast f \quad \text{(convolution)} \]

with kernel

\[
\begin{bmatrix}
\frac{1}{2\Delta x} & 1 & 0 & -1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

\[ \nabla_x[u, v] \]

sometimes this may not be a very good way to do things, as we shall see
Finite differences responding to noise

Increasing noise ->
(this is zero mean additive gaussian noise)
Finite differences and noise

- Finite difference filters respond strongly to noise
  - obvious reason: image noise results in pixels that look very different from their neighbours
- Generally, the larger the noise the stronger the response

- What is to be done?
  - intuitively, most pixels in images look quite a lot like their neighbours
  - this is true even at an edge; along the edge they’re similar, across the edge they’re not
  - suggests that smoothing the image should help, by forcing pixels different to their neighbours (=noise pixels?) to look more like neighbours
Smoothing and Differentiation

■ Issue: noise
  • smooth before differentiation
  • two convolutions: smooth, and then differentiate?
  • actually, no - we can use a derivative of Gaussian filter
    – because differentiation is convolution, and convolution is associative
      \[ \nabla_x \ast (H \ast f) = (\nabla_x \ast H) \ast f \]

\[ \nabla_x \ast H \quad \nabla_y \ast H \]
\[(\nabla_x \ast H) \ast f\]

The scale of the smoothing filter affects derivative estimates, and also the semantics of the edges recovered.
**Sobel derivative kernels**

\[
\begin{align*}
\frac{\partial}{\partial x} f &= \frac{1}{8\Delta x} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \nabla_x [u, v] \\
\frac{\partial}{\partial y} f &= \frac{1}{8\Delta y} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \nabla_y [u, v]
\end{align*}
\]
Image Gradients

- Recall for a function of two (or more) variables \( f(x, y) \)

\[
\nabla f = \begin{bmatrix}
\frac{\partial f}{\partial x} \\
\frac{\partial f}{\partial y}
\end{bmatrix} \approx \begin{bmatrix}
\nabla_x * f \\
\nabla_y * f
\end{bmatrix}
\]

Gradient at point \((x,y)\)

- A two (or more) dimensional vector

- The absolute value \( |\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \approx \sqrt{\left(\nabla_x * f\right)^2 + \left(\nabla_y * f\right)^2} \)

  is large at image boundaries

- The direction of the gradient corresponds to the direction of the “steepest ascend” - normally gradient is orthogonal to object boundaries in the image.

- Small image gradients in low textured areas
Comment: vector $\nabla f$ is independent of specific coordinate system

- Equivalently, gradient of function $f(p)$ at point $p \in \mathbb{R}^2$ can be defined as a vector $\nabla f$ s.t. for any unit vector $\vec{n}$

$$\nabla f \cdot \vec{n} = \frac{\partial f}{\partial \vec{n}} \approx \frac{f(p + \epsilon \cdot \vec{n}) - f(p)}{\epsilon}$$

Gradient at point $p$
dot product
directional derivative of function $f$ along direction $\vec{n}$

- pure vector algebra, specific coordinate system is irrelevant
- works for functions of two, three, or any larger number of variables
- previous slide gives a specific way for computing coordinates $(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$ of vector $\nabla f$ w.r.t. given orthogonal basis (axis X and Y).
Image Gradient

- Typical application where image gradients are used is *image edge* detection
  - find points with large image gradients

```
"edge features"
```

```
Canny edge detector suppresses non-extrema Gradient points
```
Second Image Derivatives
(Laplace operator $\Delta f$)

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \left[ \frac{\partial}{\partial x} \quad \frac{\partial}{\partial y} \right] \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot f = \nabla \cdot \nabla f$$

“divergence of gradient”

rotationally invariant second derivative for 2D functions

rate of change for the rate of change in x-direction

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial f}{\partial x} \left( \frac{+1}{2} \right) - \frac{\partial f}{\partial x} \left( \frac{-1}{2} \right)$$

rate of change for the rate of change in y-direction

$$\frac{\partial^2 f}{\partial y^2} = \frac{\partial f}{\partial y} \left( \frac{+1}{2} \right) - \frac{\partial f}{\partial y} \left( \frac{-1}{2} \right)$$
Laplacian of a Gaussian (LoG)

\[ \Delta \ast G \] image should be smoothed a bit first

\[ \text{LoG}(x, y) = -\frac{1}{\pi \sigma^4} \left[ 1 - \frac{x^2 + y^2}{2\sigma^2} \right] e^{-\frac{x^2 + y^2}{2\sigma^2}} \]

MATLAB: logfilt = fspecial('log',25,4);

Second Image Derivatives
(Laplace operator $\Delta f$)

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

For simplicity, assume $f(x,y) = \text{const}(y)$. Then, Laplacian of $f$ is simply a second derivative of $f(x) = f(x,y)$.

Application: **Laplacian Zero Crossings** are used for edge detection
(alternative to methods computing Gradient extrema)

Laplacian of a Gaussian (LoG)

$\Delta \ast G$  
image should be smoothed a bit first

Unsharp masking

- What does blurring take away?

\[ U = I - G \ast I \]

\[ \begin{align*}
- & = \\
unsharp mask & = \\
\alpha & = \\
unsharp mask & = \\
\end{align*} \]
Unsharp masking

\[(1 + \alpha)I - \alpha \cdot G^* I \approx [(1 + \alpha)G_{\sigma_1} - \alpha \cdot G_{\sigma_2}] \ast I\]

\[U = I - G^* I\]

\[I + \alpha \cdot U\]
Unsharp masking

MATLAB

Imrgb = imread('file.jpg');
im = im2double(rgb2gray(imrgb));
g = fspecial('gaussian', 25,4);
imblur = conv2(im,g,'same');
imagesc([im imblur])
imagesc([im im+.4*(im-imblur)])

unsharp mask kernel can be seen as a difference of two Gaussians (DoG) with $\sigma_1 << \sigma_2$.

$DoG \equiv G_{\sigma_1} - G_{\sigma_2} \approx LoG$

Filters and Templates

- Applying a filter at some point can be seen as taking a dot-product between the image and some vector.
- Filtering the image is a set of dot products.

Insight
- Filters may look like the effects they are intended to find.
- Filters find effects they look like.

\[ \nabla_x \ast H \]
\[ \nabla_y \ast H \]
Normalized Cross-Correlation (NCC)

- filtering as a **dot product**
- now measure the angle:

NCC output is filter output divided by root of the sum of squares of values over which filter lies

\[
g[t] = \frac{h \cdot f_t}{|h| \cdot |f_t|} = \cos(\alpha)
\]

**cross-correlation of** *h* and *f* at \(t=(x,y)\)

\[
\sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u, v] \cdot f[x+u, y+v]
\]

- division makes this a non-linear operation

- vector lengths \(|z| = \sqrt{\sum_{i=1}^{n} z_i^2}\)
Normalized Cross-Correlation (NCC)

- filtering as a dot product
- now measure the angle:

NCC output is filter output divided by root of the sum of squares of values over which filter lies

\[ g[t] = \frac{(h - \bar{h}) \cdot (f_t - \bar{f}_t)}{|h - \bar{h}| \cdot |f_t - \bar{f}_t|} \]

**Tricks:**
- subtract template average \( \bar{h} \)
  (to give zero output for constant regions, reduces response to irrelevant background)
- subtract patch average \( \bar{f}_t \) when computing the normalizing constant (i.e. subtract the image mean in the neighborhood)
Normalized Cross-Correlation (NCC)

- filtering as a **dot product**
- now **measure the angle:**
  NCC output is filter output divided by root of the sum of squares of values over which filter lies

**Tricks:**
- subtract *template average* $\bar{h}$
  (to give zero output for constant regions, reduces response to irrelevant background)
- subtract *patch average* $\bar{f}_t$ when computing the normalizing constant (i.e. subtract the image mean in the neighborhood)

```
f_t = (x, y)
```

```
\[ \sigma_z \equiv \sqrt{ \frac{1}{n} \sum_{i=1}^{n} (z_i - \bar{z})^2 } = \sqrt{ \frac{1}{n} \cdot |z - \bar{z}| } \]
```
Normalized Cross-Correlation (NCC)

- filtering as a **dot product**
- now **measure the angle**: NCC output is filter output divided by root of the sum of squares of values over which filter lies

**Tricks:**
- subtract *template average* $\bar{h}$ (to give zero output for constant regions, reduces response to irrelevant background)
- subtract *patch average* $\bar{f}_t$ when computing the normalizing constant (i.e. subtract the image mean in the neighborhood)

\[ g[t] = \frac{\text{cov}(h, f_t)}{\sigma_h \cdot \sigma_{f_t}} \]

*standard in statistics correlation coefficient*

- $\rho$ between $h$ and $f_t$

equivocally using statistical term cov (covariance)

$\text{cov}(a, b) \equiv E(a - \bar{a})(b - \bar{b}) = \frac{1}{n} \sum_{i=1}^{n} (a_i - \bar{a})(b_i - \bar{b}) = \frac{(a - \bar{a}) \cdot (b - \bar{b})}{n}$
Normalized Cross-Correlation (NCC)

templates

points of interest or feature points

pictures from Silvio Savarese
Feature points are used for:

- Image alignment (homography, fundamental matrix)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- … other
Harris corner detector

- C.Harris, M.Stephens. “A Combined Corner and Edge Detector”. 1988
The 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
Harris Detector: Basic Idea

“flat” region: no change in all directions

“edge”: no change along the edge direction

“corner”: significant change in all directions
For any given image patch or window $w$ we should measure how it changes when shifted by $ds = \begin{bmatrix} u \\ v \end{bmatrix}$

Notation: let patch be defined by its support function $w(x,y)$ over image pixels
Harris Detector: Mathematics

Patch \( w \) change measure for shift \( ds = \begin{bmatrix} u \\ v \end{bmatrix} \):

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

Window function

Shifted intensity

Intensity

NOTE: window support functions \( w(x,y) = \)

1 in window, 0 outside

or

Gaussian (weighted) support
Harris Detector: Mathematics

Change of intensity for the shift $ds = \begin{bmatrix} u \\ v \end{bmatrix}$ assuming image gradient $\nabla I \equiv \begin{bmatrix} I_x \\ I_y \end{bmatrix}$

$I(x+u, y+v) - I(x, y) \approx I_x \cdot u + I_y \cdot v = ds^T \cdot \nabla I$

Rate of change for $I$ at $(x,y)$ in direction $(u,v) = ds$

(remember gradient definition on earlier slides!!!)

This is 2D analogue of 1st order Taylor expansion

$[I(x+u, y+v) - I(x, y)]^2 \approx ds^T \cdot \nabla I \cdot \nabla I^T \cdot ds$

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

$\approx ds^T \cdot \left( \sum_{x,y} w(x, y) \cdot \nabla I \cdot \nabla I^T \right) \cdot ds = ds^T \cdot M_w \cdot ds$
Harris Detector: Mathematics

Change of intensity for the **shift** \( ds = \begin{bmatrix} u \\ v \end{bmatrix} \) assuming **image gradient** \( \nabla I \equiv \begin{bmatrix} I_x \\ I_y \end{bmatrix} \)

\[
E_w(u, v) \equiv \begin{bmatrix} u & v \end{bmatrix} \cdot M_w \cdot \begin{bmatrix} u \\ v \end{bmatrix} = ds^T \cdot M_w \cdot ds
\]

where \( M_w \) is a 2×2 matrix computed from image derivatives inside patch \( w \)

**matrix** \( M \) is also called **Harris matrix** or **structure tensor**

\[
\begin{bmatrix}
I_x^2 & I_xI_y \\
I_yI_x & I_y^2
\end{bmatrix}
\]

This tells you how to compute \( M_w \) at any window \( w \) (t.e. any image patch)

\[
\cdots \left( \sum_{x, y} w(x, y) \cdot \nabla I \cdot \nabla I^T \right) \cdots
\]
Harris Detector: Mathematics

Change of intensity for the shift $ds = \begin{bmatrix} u \\ v \end{bmatrix}$ assuming image gradient $\nabla I \equiv \begin{bmatrix} I_x \\ I_y \end{bmatrix}$

$$E_w(u, v) \cong \begin{bmatrix} u & v \end{bmatrix} \cdot M_w \cdot \begin{bmatrix} u \\ v \end{bmatrix} = ds^T \cdot M_w \cdot ds$$

$M$ is a positive semi-definite matrix \hspace{1cm} (Exercise: show that $ds^T \cdot M \cdot ds \geq 0$ for any $ds$)

$M$ can be analyzed via its isolines, e.g. $ds^T \cdot M_w \cdot ds = 1$ (ellipsoid)

Points on this ellipsoid are shifts $ds=(u,v)$ giving the same value of energy $E(u,v)=1$.
Thus, the ellipsoid allows to visually compare sensitivity of energy $E$ to shifts $ds$ in different directions.

two eigen values of matrix $M_w$
Classification of image points using eigenvalues of $M$:

- $\lambda_1$ and $\lambda_2$ are small; $E$ is almost constant in all directions.
- $\lambda_1$ and $\lambda_2$ are large, $\lambda_1 \sim \lambda_2$; $E$ rapidly increases in all directions.
- $\lambda_1 \gg \lambda_2$.
Harris Detector: Mathematics

Measure of corner response:

\[ R = \frac{\det M}{\text{Trace } M} \]

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

*R should be large*  
(it implies that both \( \lambda \) are far from zero)
Harris Detector

The Algorithm:

• Find points with large corner response function $R$
  $R > \text{threshold}$

• Take the points of local maxima of $R$
Harris Detector: Workflow
Harris Detector: Workflow

Compute corner response $R$
Harris Detector: Workflow

Find points with large corner response: $R > \text{threshold}$
Harris Detector: Workflow

Take only the points of local maxima of $R$
Harris Detector: Workflow
Harris Detector: Some Properties

- Rotation invariance

Ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner response $\mathbf{R}$ is invariant to image rotation
Harris Detector: Some Properties

- Partial invariance to \textit{affine intensity} change

  ✓ Only derivatives are used \(\Rightarrow\) invariance to intensity shift \(I \rightarrow I + b\)

  ✓ Intensity scale: \(I \rightarrow aI\)

\[\text{threshold}\] \hspace{1cm} \text{features locations stay the same, but some may appear or disappear depending on gain } a \]
Harris Detector: Some Properties

- But: non-invariant to *image scale*!

All points will be classified as *edges*  
Corner!
Scale Invariant Detection

- Consider regions (e.g. circles) of different sizes around a point
- Regions of corresponding sizes will look the same in both images
Scale Invariant Detection

- The problem: how do we choose corresponding circles *independently* in each image?

- Choose the scale of the “best” corner
Other feature detectors

- *LoG* and *DoG* operators are also used to detect “features”
  
  - they find reliable “blob” features (at appropriate scale)

- these operators also respond to edges. To improve “selectivity”, post-processing is necessary.
  - e.g. eigen-values of the Harris matrix cold be used as in the corner operator.
    If the ratio of the eigen-values is too high, then the local image is regarded as too edge-like and the feature is rejected.
Other features

- MOPS, Hog, SIFT, ...

Features are characterized by **location** and **descriptor**

<table>
<thead>
<tr>
<th>Features</th>
<th>location</th>
<th>descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>any pixel</td>
<td>RGB vector</td>
</tr>
<tr>
<td>edge</td>
<td>Laplacian zero crossing</td>
<td>image gradient</td>
</tr>
<tr>
<td>corner</td>
<td>local max of $R$</td>
<td>magnitude of $R$</td>
</tr>
<tr>
<td>MOPS</td>
<td>corners</td>
<td>normalized intensity patch</td>
</tr>
<tr>
<td>HOG</td>
<td>LOG extrema points</td>
<td>gradient orientation histograms</td>
</tr>
<tr>
<td>SIFT</td>
<td>LOG extrema points</td>
<td></td>
</tr>
</tbody>
</table>
Feature descriptors

- We know how to detect points
- Next question: **How to match them?**

Point descriptor should be:
1. Invariant
2. Distinctive
Descriptors Invariant to Rotation

- Find local orientation

Dominant direction of gradient

- Extract image patches relative to this orientation
Multi-Scale Oriented Patches (MOPS)

- Interest points
  - Multi-scale Harris corners
  - Orientation from blurred gradient
  - Geometrically invariant to rotation

- Descriptor vector
  - Bias/gain normalized sampling of local patch (8x8)
  - Photometrically invariant to affine changes in intensity

[Brown, Szeliski, Winder, CVPR’2005]
Descriptor Vector

- Orientation = blurred gradient
- Rotation Invariant Frame
  - Scale-space position \((x, y, s)\) + orientation \((\theta)\)
Detections at multiple scales

Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.
MOPS descriptor vector

- 8x8 oriented patch
  - Sampled at 5 x scale
- Bias/gain normalisation: $I' = (I - \mu)/\sigma$