Filtering

Introduction to Computer Vision
CSE 152
Lecture 8

Announcements

• HW 1 will be due on Monday
• HW2 will be available in the next couple days.
• See links on web page for reading on binary image processing (e-reserves)
• No class on Tuesday: Makeup class

Binary System Summary

1. Acquire images and binarize (tresholding, color labels, etc.).
2. Possibly clean up image using morphological operators.
3. Determine regions (blobs) using connected component exploration
4. Compute position, area, and orientation of each blob using moments
5. Compute features that are rotation, scale, and translation invariant using Moments (e.g., Eigenvalues of normalized moments).

Threshold selection with the P-Tile Method

• If the size of the object is approx. known, pick T s.t. the area under the histogram corresponds to the size of the object:

“Peakiness” Detection Algorithm

• Find the two highest local maxima that minimum distance apart: \( g_i \) and \( g_j \)
• Find lowest point between them: \( g_k \)
• Measure “peakiness”:
  \[- \min(H(g_i),H(g_j))/H(g_k) \]
• Find \((g_i,g_k,g_j)\) with highest peakiness

Four & Eight Connectedness

Four Connected
Eight Connected
Recursive Labeling
Connected Component Exploration

Properties extracted from binary image
- A tree showing containment of regions
- Properties of a region
  1. Genus – number of holes
  2. Centroid
  3. Area
  4. Perimeter
  5. Moments (e.g., measure of elongation)
  6. Number of "extrema" (indentations, bulges)
  7. Skeleton

Moments

(related to moments of inertia)

\[ S = \{(x, y) | f(x, y) = 1\} \]

Given a pair of non-negative integers (j,k) the discrete \((j,k)^{th}\) moment of \(S\) is:

\[ M_{jk}(S) = \sum_{(x,y) \in S} x^j y^k \]

The order of the \(M_{jk}\) moment is \(j+k\).

\[ M_{jk} = \sum_{x=1}^{n} \sum_{y=1}^{m} B(x,y)x^j y^k \]

- Fast way to implement computation over \(n\) by \(m\) image or window
- One object

Moments: Centroid

\[ S = \{(x, y) | f(x, y) = 1\} \]

Example:

\[ M_{jk}(S) = \sum_{(x,y) \in S} x^j y^k \]

\[ M_{00}(S) = \sum_{(x,y) \in S} x^0 y^0 = \sum_{(x,y) \in S} 1 = \text{\#}(S) \]

Area of \(S\):

Area: Moment \(M_{00}\)

\[ S = \{(x, y) | f(x, y) = 1\} \]

Example:

\[ M_{00}(S) = \sum_{(x,y) \in S} x^0 y^0 = \sum_{(x,y) \in S} 1 = \text{\#}(S) \]

Shape recognition by Moments

Recognition could be done by comparing moments

However, moments \(M_{jk}\) are not invariant under:
- Translation
- Scaling
- Rotation
- Skewing
Central Moments
\[ S = \{(x, y) | f(x, y) = 1\} \]
\[ \bar{x} = \frac{M_{10}(S)}{M_{00}(S)} \quad \bar{y} = \frac{M_{01}(S)}{M_{00}(S)} \]

Given a pair of non-negative integers \((j,k)\) the central \((j,k)^{th}\) moment of \(S\) is given by:
\[ \mu_{jk}(S) = \sum_{(x,y) \in S} (x-\bar{x})^j (y-\bar{y})^k \]

Or the central moments can be computed from precomputed regular moments
\[ \mu_{jk} = \sum_{i=1}^{n} \sum_{m=1}^{n} \left[ \sum_{i,j} \frac{(x-i)(y-j)}{i} \right] \]

Translation by \(T = (a,b)\):
\[ S_T = \{(x', y') | x' = x + a, y' = y + b, (x,y) \in S\} \]
\[ \bar{x}' = \frac{M_{10}(S_T)}{M_{00}(S_T)} - \bar{x} + a \quad \bar{y}' = \frac{M_{01}(S_T)}{M_{00}(S_T)} - \bar{y} + b \]

Translation INVARIANT!

Normalized Moments
\[ S = \{(x, y) | f(x, y) = 1\} \]
\[ \mu_{jk}(S) = \sum_{(x,y) \in S} (x-\bar{x})^j (y-\bar{y})^k \]
\[ \sigma_x = \sqrt{\frac{\mu_{20}(S)}{M_{00}(S)}} \quad \sigma_y = \sqrt{\frac{\mu_{02}(S)}{M_{00}(S)}} \]

Given a pair of non-negative integers \((j,k)\) the normalized \((j,k)^{th}\) moment of \(S\) is given by:
\[ m_{jk}(S) = \sum_{(x,y) \in S} \left[ \frac{(x-\bar{x})^j}{\sigma_x^j} \right] \left[ \frac{(y-\bar{y})^k}{\sigma_y^k} \right] \]

Normalized moments are scale and translation invariant.

Region orientation from Second Moment Matrix
1. Compute second centralized moment matrix
\[
\begin{bmatrix}
\mu_{20} & \mu_{11} \\
\mu_{11} & \mu_{11}
\end{bmatrix}
\]
• Symmetric, positive definite matrix
• Positive Eigenvalues
• Orthogonal Eigenvectors
2. Compute Eigenvectors of Moment Matrix to obtain orientation
3. Eigenvalues are independent of orientation, translation!

Moments
• Regular Moments \(M_{jk}\)
• Central Moments \(\mu_{jk}\): Translation invariant
• Normalized Moments \(m_{jk}\): Translation and scale Invariant
• Eigenvalues of Second Moment Matrix: translation, scale, and rotation invariant.
• Hu Moments: Higher than second order, translation, rotation, and scale invariant
Binarization using Color

- Objects in robocup are distinguished by color.
- How do you binarize the image so that pixels where ball is located are labeled with 1, and other locations are 0?
- Let \( C_b = (r \ g \ b)^T \) be the color of the ball.

\[
0 = \begin{cases} 1 & \text{if } ||c(u,v) - c_b||^2 \leq \epsilon \\ 0 & \text{otherwise} \end{cases}
\]

Better alternative (why?)
- Convert \( c(u,v) \) to HSV space \( H(u,v), S(u,v), V(u,v) \)
- Convert \( c_b \) to HSV
- Check that HS distance is less than threshold \( \epsilon \) and brightness \( V \) is greater than a threshold \( V > \tau \)

Color Blob tracking

- Color-based tracker gets lost on white knight: Same Color

What is image filtering?

- Modify the pixels in an image based on some function of a local neighborhood of the pixels.

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\quad \text{Local image data}
\]

\[
\begin{array}{ccc}
1 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
\end{array}
\quad \text{Modified image data = }
\]

(From Bill Freeman)

Linear functions

- Simplest: linear filtering.
  - Replace each pixel by a linear combination of its neighbors.
- The prescription for the linear combination is called the "convolution kernel".

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\quad \text{Local image data}
\]

\[
\begin{array}{ccc}
0 & 1 & 0 \\
1 & 0 & 1 \\
0 & 1 & 0 \\
\end{array}
\quad \text{kernel}
\]

\[
\begin{array}{ccc}
1 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
\end{array}
\quad \text{Modified image data = }
\]

(Freeman)
Smoothing by Averaging

Kernel: $\boldsymbol{\mathbf{K}}$

- General process:
  - Form new image whose pixels are a weighted sum of original pixel values, using the same set of weights at each point.
- Properties
  - Output is a linear function of the input
  - Output is a shift-invariant function of the input (i.e., shift the input image two pixels to the left; the output is shifted two pixels to the left)

Linear Filters

- Example: smoothing by averaging
  - Form the average of pixels in a neighbourhood
- Example: smoothing with a Gaussian
  - Form a weighted average of pixels in a neighbourhood
- Example: finding a derivative
  - Form a weighted average of pixels in a neighbourhood

Convolution

$R = K \ast I$

Kernel size is $m+1 \times m+1$

$$R(i, j) = \sum_{k=-2}^{2} \sum_{h=-2}^{2} K(h, k)I(i-h, j-k)$$
Convolution: \( R = K \ast I \)

\[
R(i, j) = \sum_{h=1}^{m} \sum_{k=1}^{m} K(h,k)I(i-h,j-k)
\]

Kernel size is \( m+1 \) by \( m+1 \)

\( m = 2 \)
Convolution: \( R = K \ast I \)

Kernel size is \( m+1 \) by \( m+1 \)

Impulse Response

Linear filtering (warm-up slide)

Linear filtering (no change)
Shifted one Pixel to the left

Original

Shifted one Pixel to the left

Original

Blurred (filter applied in both dimensions).

Original

Practice with linear filters

Original
**Practice with linear filters**

**Sharpening example**
- Accentuates differences with local average

**Sharpening**
- before
- after

**Filters are 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

**Smoothing by Averaging**
- Kernel

**Properties of Continuous Convolution**
(Same for discrete)
- Let f, g, h be images and let * denote convolution
- Commutative: \( f * g = g * f \)
- Associative: \( f * (g * h) = (f * g) * h \)
- Linear: for scalars a & b and images f, g, h
  \( (af + bg) * h = af * h + bg * h \)
- Differentiation rule
  \[
  \frac{\partial}{\partial x} (f * g) = \frac{\partial f}{\partial x} * g = f * \frac{\partial g}{\partial x}
  \]
**Numerical Derivatives**

Take Taylor series expansion of \( f(x) \) about \( x_0 \)

\[
f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2} f''(x_0)(x-x_0)^2 + \ldots
\]

Consider Samples taken at increments of \( h \) and first two terms, we have

\[
f(x_0+h) = f(x_0) + f'(x_0)h + \frac{1}{2} f''(x_0)h^2
\]

\[
f(x_0-h) = f(x_0) - f'(x_0)h + \frac{1}{2} f''(x_0)h^2
\]

Subtracting and adding \( f(x_0+h) \) and \( f(x_0-h) \) respectively yields

\[
f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h}
\]

\[
f''(x_0) = \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2}
\]

Convolve with kernels:

- First Derivative: \([-1/2h\ 0\ 1/2h]\)
- Second Derivative: \([1/h^2\ -2/h^2\ 1/h^2]\)

**Computing Image Derivatives**

- Often, we can measure \( h \) in units of pixels. So, \( h=1 \) and the derivative kernels become:
  - First Derivative: \([-1/2\ 0\ 1/2]\)
  - Second Derivative: \([-1\ 2\ -1]\)
- When looking for local maxima of the first derivative, it is sufficient to use the kernel \([-1\ 0\ 1]\). Why?
- When computing derivatives in the \( x \) and \( y \) directions, use:

\[
\frac{d}{dx} = \begin{bmatrix} -1/2 & 0 & 1/2 \end{bmatrix}
\]

\[
\frac{d}{dy} = \begin{bmatrix} -1/2 & 0 & 1/2 \end{bmatrix}
\]

**Filtering to reduce noise**

- Noise is what we’re not interested in.
  - We’ll discuss simple, low-level noise today:
    - Light fluctuations; Sensor noise; Quantization effects; Finite precision
  - Not complex: shadows; extraneous objects.
- A pixel’s neighborhood contains information about its intensity.
- Averaging noise reduces its effect.

**Additive noise**

- \( I = S + N \). Noise doesn’t depend on signal.
- We’ll consider:

\[
I_i = s_i + n_i \text{ with } E(n_i) = 0
\]

\( s_i \) deterministic. \( n_i \) a random var. \( n_i, n_j \) independent for \( i \neq j \)

\( n_i, n_j \) identically distributed

**Average Filter**

- Mask with positive entries, that sum 1.
- Replaces each pixel with an average of its neighborhood.
- If all weights are equal, it is called a BOX filter.
Does it reduce noise?

- Intuitively, takes out small variations.

\[ I(i, j) = \hat{I}(i, j) + N(i, j) \quad \text{with} \quad N(i, j) \sim N(0, \sigma) \]

\[ O(i, j) = \frac{1}{m} \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} I(i-h, j-k) + N(i-h, j-k) = \]

\[ = \frac{1}{m} \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} \hat{I}(i-h, j-k) + \frac{1}{m} \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} N(i-h, j-k) \]

\[ E(\hat{N}(i, j)) = 0 \]

\[ E(\hat{N}^2(i, j)) = \frac{1}{m} m \sigma^2 = \frac{\sigma^2}{m} \Rightarrow \hat{N}(i, j) \sim N(0, \frac{\sigma}{\sqrt{m}}) \]

(Camps)

An Isotropic Gaussian

- The picture shows a smoothing kernel proportional to

\[ e^{-\frac{x^2+y^2}{2\sigma^2}} \]

(which is a reasonable model of a circularly symmetric fuzzy blob)