Digital Image Processing (CS/ECE 545)
Lecture 11: Geometric Operations, Comparing Images and Future Directions

Prof Emmanuel Agu

Computer Science Dept.
Worcester Polytechnic Institute (WPI)
Geometric Operations

- Filters, point operations change intensity
- Pixel position (and geometry) unchanged
- Geometric operations: change image geometry
- **Examples**: translating, rotating, scaling an image

Examples of Geometric operations
Geometric Operations

- Example applications of geometric operations:
  - Zooming images, windows to arbitrary size
  - Computer graphics: deform textures and map to arbitrary surfaces

- **Definition:** Geometric operation transforms image $I$ to new image $I'$ by modifying coordinates of image pixels

\[ I(x, y) \rightarrow I'(x', y') \]

- Intensity value originally at $(x,y)$ moved to new position $(x',y')$

Example: Translation geometric operation moves value at $(x,y)$ to $(x + d_x, y + d_y)$
Geometric Operations

- Since image coordinates can only be discrete values, some transformations may yield \((x', y')\) that’s not discrete
- **Solution:** interpolate nearby values
Simple Mappings

- **Translation:** (shift) by a vector \((d_x, d_y)\)

\[
T_x : x' = x + d_x \quad \text{or} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \end{pmatrix}
\]

- **Scaling:** (contracting or stretching) along x or y axis by a factor \(s_x\) or \(s_y\)

\[
T_x : x' = s_x \cdot x \quad \text{or} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}
\]

Simple Mappings

- **Shearing:** along x and y axis by factor $b_x$ and $b_y$

  $$
  T_x : x' = x + b_x \cdot y \\
  T_y : y' = y + b_y \cdot x
  $$
  or
  $$
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} =
  \begin{pmatrix}
  1 & b_x \\
  b_y & 1
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  $$

- **Rotation:** the image by an angle $\alpha$

  $$
  T_x : x' = x \cdot \cos \alpha - y \cdot \sin \alpha \\
  T_y : y' = x \cdot \sin \alpha + y \cdot \cos \alpha
  $$
  or
  $$
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} =
  \begin{pmatrix}
  \cos \alpha & -\sin \alpha \\
  \sin \alpha & \cos \alpha
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  $$
Image Flipping & Rotation by 90 degrees

- We can achieve 90, 180 degree rotation easily
- Basic idea: Look up a transformed pixel address instead of the current one
- To flip an image upside down:
  - At pixel location $xy$, look up the color at location $x(1-y)$

- For horizontal flip:
  - At pixel location $xy$, look up $(1-x)y$

- Rotating an image 90 degrees counterclockwise:
  - At pixel location $xy$, look up $(y, 1-x)$
**Image Flipping, Rotation and Warping**

- **Image warping**: we can use a function to select which pixel somewhere else in the image to look up.
- For example: apply function on both texel coordinates \((x, y)\)

\[ x = x + y \cdot \sin(\pi \cdot x) \]
Homogeneous Coordinates

- Notation useful for converting scaling, translation, rotating into point-matrix multiplication
- To convert ordinary coordinates into homogeneous coordinates

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} \text{ converts to } \mathbf{\hat{x}} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ h \end{pmatrix} = \begin{pmatrix} h \times x \\ h \times y \\ h \end{pmatrix}$$
Affine (3-Point) Mapping

- Can use homogeneous coordinates to rewrite translation, rotation, scaling, etc. as vector-matrix multiplication

\[
\begin{pmatrix}
  x' \\
  y' \\
  1
\end{pmatrix}
= \begin{pmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  0 & 0 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
\]

- **Affine mapping**: Can then derive values of matrix that achieve desired transformation (or combination of transformations)

- Inverse of transform matrix is **inverse mapping**
Affine (3-Point) Mapping

- What’s so special about affine mapping?
- Maps
  - straight lines -> straight lines,
  - triangles -> triangles
  - rectangles -> parallelograms
  - Parallel lines -> parallel lines
- Distance ratio on lines do not change
Non-Linear Image Warps

(a) Original Twirl Ripple Spherical
(b) (d) Twirl
(c) Ripple
(f) Spherical
### Twirl

- **Notation:** Instead using texture colors at \((x',y')\), use texture colors at twirled \((x,y)\) location

- **Twirl?**
  - Rotate image by angle \(\alpha\) at center or anchor point \((x_c,y_c)\)
  - Increasingly rotate image as radial distance \(r\) from center increases (up to \(r_{\text{max}}\))
  - Image unchanged outside radial distance \(r_{\text{max}}\)

\[
T^{-1}_x : x = \begin{cases} 
    x_c + r \cdot \cos(\beta) & \text{for } r \leq r_{\text{max}} \\
    x' & \text{for } r > r_{\text{max}},
\end{cases}
\]

\[
T^{-1}_y : y = \begin{cases} 
    y_c + r \cdot \sin(\beta) & \text{for } r \leq r_{\text{max}} \\
    y' & \text{for } r > r_{\text{max}},
\end{cases}
\]

with

\[
\begin{align*}
    d_x &= x' - x_c, & r &= \sqrt{d_x^2 + d_y^2}, \\
    d_y &= y' - y_c, & \beta &= \arctan(d_y, d_x) + \alpha \cdot \left(\frac{r_{\text{max}} - r}{r_{\text{max}}}\right).
\end{align*}
\]
Ripple

- Ripple causes wavelike displacement of image along both the x and y directions.

\[
T_x^{-1} : \quad x = x' + a_x \cdot \sin\left(\frac{2\pi \cdot y'}{\tau_x}\right),
\]
\[
T_y^{-1} : \quad y = y' + a_y \cdot \sin\left(\frac{2\pi \cdot x'}{\tau_y}\right).
\]

- Sample values for parameters (in pixels) are
  - \(\tau_x = 120\)
  - \(\tau_y = 250\)
  - \(a_x = 10\)
  - \(a_y = 15\)
**Spherical Transformation**

- Imitates viewing image through a lens placed over image.
- Lens parameters: center \((x_c, y_c)\), lens radius \(r_{\text{max}}\) and refraction index \(\rho\).
- Sample values \(\rho = 1.8\) and \(r_{\text{max}} = \) half image width.

\[
T_x^{-1}: x = x' - \begin{cases} 
  z \cdot \tan(\beta_x) & \text{for } r \leq r_{\text{max}}, \\
  0 & \text{for } r > r_{\text{max}}, 
\end{cases}
\]

\[
T_y^{-1}: y = y' - \begin{cases} 
  z \cdot \tan(\beta_y) & \text{for } r \leq r_{\text{max}}, \\
  0 & \text{for } r > r_{\text{max}}, 
\end{cases}
\]

\[
d_x = x' - x_c, \quad r = \sqrt{d_x^2 + d_y^2}, \\
d_y = y' - y_c, \quad z = \sqrt{r_{\text{max}}^2 - r^2}, \\
\beta_x = (1 - \frac{1}{\rho}) \cdot \sin^{-1}\left(\frac{d_x}{\sqrt{(d_x^2 + z^2)}}\right), \\
\beta_y = (1 - \frac{1}{\rho}) \cdot \sin^{-1}\left(\frac{d_y}{\sqrt{(d_y^2 + z^2)}}\right).
\]
Image Warping
How to tell if 2 Images are same?

- Pixel by pixel comparison?
  - Makes sense only if pictures taken from same angle, same lighting, etc

- Noise, quantization, etc introduces differences
  - Human may say images are same even with numerical differences
Comparing Images

- Better approach: Template matching
  - Identify similar sub-images (called template) within 2 images

- Applications?
  - Match left and right picture of stereo images
  - Find particular pattern in scene
  - Track moving pattern through image sequence
Template Matching

- Basic idea
  - Move given pattern (template) over search image
  - Measure difference between template and sub-images at different positions
  - Record positions where highest similarity is found
Template Matching

- Difficult issues?
  - What is distance (difference) measure?
  - What levels of difference should be considered a match?
  - How are results affected when brightness or contrast changes?
Template Matching in Intensity Images

- Consider problem of finding a template (reference image) $R$ within a search image
- Can be restated as **Finding positions in which contents of $R$ are most similar to the corresponding subimage of $I$**
- If we denote $R$ shifted by some distance $(r,s)$ by

$$R_{r,s}(u,v) = R(u-r, v-s)$$
Template Matching in Intensity Images

- We can restate template matching problem as:
- Finding the offset \((r,s)\) such that the similarity between the shifted reference image \(R_{r,s}\) and corresponding subimage \(I\) is a maximum

\[
R_{r,s}(u,v) = R(u-r,v-s)
\]

- Solving this problem involves solving many sub-problems
Distance between Image Patterns

- Many measures proposed to compute distance between the shifted reference image $R_{r,s}$ and corresponding subimage $I$
Distance between Image Patterns

- Many measures proposed to compute distance between the shifted reference image \( R_{r,s} \) and corresponding subimage \( I \):

- Sum of absolute differences:

\[
d_A(r, s) = \sum_{(i,j) \in R} |I(r+i, s+j) - R(i, j)|
\]

- Maximum difference:

\[
d_M(r, s) = \max_{(i,j) \in R} |I(r+i, s+j) - R(i, j)|
\]

- Sum of squared differences (also called N-dimensional Euclidean distance):

\[
d_E(r, s) = \left[ \sum_{(i,j) \in R} (I(r+i, s+j) - R(i, j))^2 \right]^{1/2}
\]
Distance and Correlation

- Best matching position between shifted reference image $R_{r,s}$ and subimage $I$ minimizes square of $d_E$ which can be expanded as

$$d_E^2(r, s) = \sum_{(i,j) \in R} (I(r+i, s+j) - R(i, j))^2$$

$$= \sum_{(i,j) \in R} I^2(r+i, s+j) + \sum_{(i,j) \in R} R^2(i, j) - 2 \sum_{(i,j) \in R} I(r+i, s+j) \cdot R(i, j)$$

- $B$ term is a constant, independent of $r$, $s$ and can be ignored
- $A$ term is sum of squared values within subimage $I$ at current offset $r$, $s$
Distance and Correlation

\[ d^2_E(r, s) = \sum_{(i,j) \in R} (I(r+i, s+j) - R(i, j))^2 \]

\[ = \sum_{(i,j) \in R} I^2(r+i, s+j) + \sum_{(i,j) \in R} R^2(i, j) - 2 \sum_{(i,j) \in R} I(r+i, s+j) \cdot R(i, j) \]

- \( C(r,s) \) term is **linear cross correlation** between \( I \) and \( R \) defined as

\[
(I \otimes R)(r, s) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(r+i, s+j) \cdot R(i, j)
\]

- Since \( R \) and \( I \) are assumed to be zero outside their boundaries

\[
\sum_{i=0}^{w_R-1} \sum_{j=0}^{h_R-1} I(r+i, s+j) \cdot R(i, j) = \sum_{(i,j) \in R} I(r+i, s+j) \cdot R(i, j)
\]

- **Note**: Correlation is similar to linear convolution
- Min value of \( d^2_E(r,s) \) corresponds to max value of \( (I \otimes R)(r, s) \)
Normalized Cross Correlation

- Unfortunately, A term is not constant in most images
- Thus cross correlation result varies with intensity changes in image \( I \)
- **Normalized cross correlation** considers energy in \( I \) and \( R \)

\[
C_N(r, s) = \frac{C(r, s)}{\sqrt{A(r, s) \cdot B}} = \frac{C(r, s)}{\sqrt{A(r, s) \cdot \sqrt{B}}}
\]

\[
= \sum_{(i,j) \in R} I(r+i, s+j) \cdot R(i, j)
\]

\[
\left[ \sum_{(i,j) \in R} I^2(r+i, s+j) \right]^{1/2} \cdot \left[ \sum_{(i,j) \in R} R^2(i, j) \right]^{1/2}
\]

- \( C_N(r, s) \) is a local distance measure, is in [0,1] range
- \( C_N(r, s) = 1 \) indicates maximum match
- \( C_N(r, s) = 0 \) indicates images are very dissimilar
Correlation Coefficient

- **Correlation coefficient**: Use differences between $I$ and $R$ and their average values

\[ C_L(r, s) = \frac{\sum_{(i,j) \in R} (I(r+i, s+j) - \bar{I}(r, s)) \cdot (R(i, j) - \bar{R})}{\left[ \sum_{(i,j) \in R} (I(r+i, s+j) - \bar{I}_{r,s})^2 \right]^{1/2} \cdot \left[ \sum_{(i,j) \in R} (R(i, j) - \bar{R})^2 \right]^{1/2}} \]

where the average values are defined as

\[ \bar{I}_{r,s} = \frac{1}{K} \cdot \sum_{(i,j) \in R} I(r+i, s+j) \quad \text{and} \quad \bar{R} = \frac{1}{K} \cdot \sum_{(i,j) \in R} R(i, j) \]

- $K$ is number of pixels in reference image $R$
- $C_L(r, s)$ can be rewritten as

\[ C_L(r, s) = \frac{\sum_{(i,j) \in R} (I(r+i, s+j) \cdot R(i, j)) - K \cdot \bar{I}_{r,s} \cdot \bar{R}}{\left[ \sum_{(i,j) \in R} I^2(r+i, s+j) - K \cdot \bar{I}_{r,s}^2 \right]^{1/2} \cdot S_R} \]
Correlation Coefficient Algorithm

1: CorrelationCoefficient \((I, R)\)

- **\(I(u, v)\):** search image of size \(w_I \times h_I\)
- **\(R(i, j)\):** reference image of size \(w_R \times h_R\)

Returns \(C(r, s)\) containing the values of the correlation coefficient between \(I\) and \(R\) positioned at \((r, s)\).

**Step 1—Initialize:**

2: \(K \leftarrow w_R \cdot h_R\)
3: \(\Sigma_R \leftarrow 0, \Sigma_{R2} \leftarrow 0\)
4: \textbf{for} \(i \leftarrow 0 \ldots (w_R - 1)\) \textbf{do}
5: \hspace{1em} \textbf{for} \(j \leftarrow 0 \ldots (h_R - 1)\) \textbf{do}
6: \hspace{2em} \(\Sigma_R \leftarrow \Sigma_R + R(i, j)\)
7: \hspace{2em} \(\Sigma_{R2} \leftarrow \Sigma_{R2} + (R(i, j))^2\)
8: \(\bar{R} \leftarrow \Sigma_R/K\) \hspace{2em} \(\triangleright \text{Eqn. (17.8)}\)
9: \(S_R \leftarrow \sqrt{\Sigma_{R2} - K \cdot \bar{R}^2} = \sqrt{\Sigma_{R2} - \Sigma_{R}^2/K}\) \hspace{2em} \(\triangleright \text{Eqn. (17.10)}\)

**Step 2—Compute the correlation map:**

10: \(C \leftarrow \text{new map of size } (w_I - w_R + 1) \times (h_I - h_R + 1), C(r, s) \in \mathbb{R}\)
11: \textbf{for} \(r \leftarrow 0 \ldots (w_I - w_R)\) \textbf{do}
12: \hspace{1em} \textbf{for} \(s \leftarrow 0 \ldots (h_I - h_R)\) \textbf{do} \hspace{2em} \(\triangleright \text{place } R\text{ at position } (r, s)\)
13: \hspace{2em} Compute correlation coefficient for position \((r, s)\):
14: \hspace{3em} \(\Sigma_I \leftarrow 0, \Sigma_{I2} \leftarrow 0, \Sigma_{IR} \leftarrow 0\)
15: \hspace{3em} \textbf{for} \(i \leftarrow 0 \ldots (w_R - 1)\) \textbf{do}
16: \hspace{4em} \textbf{for} \(j \leftarrow 0 \ldots (h_R - 1)\) \textbf{do}
17: \hspace{5em} \(a_I \leftarrow I(r+i, s+j)\)
18: \hspace{5em} \(a_R \leftarrow R(i, j)\)
19: \hspace{5em} \(\Sigma_I \leftarrow \Sigma_I + a_I\)
20: \hspace{5em} \(\Sigma_{I2} \leftarrow \Sigma_{I2} + a_I^2\)
21: \hspace{5em} \(\Sigma_{IR} \leftarrow \Sigma_{IR} + a_I \cdot a_R\)
22: \(I_{r,s} \leftarrow \Sigma_I/K\) \hspace{2em} \(\triangleright \text{Eqn. (17.8)}\)
23: \(C(r, s) \leftarrow \frac{\Sigma_{IR} - K \cdot I_{r,s} \cdot \bar{R}}{\sqrt{\Sigma_{I2} - K \cdot I_{r,s}^2 \cdot S_R}} = \frac{\Sigma_{IR} - \Sigma_I \cdot \bar{R}}{\sqrt{\Sigma_{I2} - \Sigma_I^2/K \cdot S_R}}\)
24: \textbf{return} \(C\). \hspace{2em} \(\triangleright C(r, s) \in [-1, 1]\)
class CorrCoeffMatcher {
    FloatProcessor I; // image
    FloatProcessor R; // template
    int wI, hI; // width/height of image
    int wR, hR; // width/height of template
    int K; // size of template

    float meanR; // mean value of template (\bar{R})
    float varR; // square root of template variance (\sigma_R)

    public CorrCoeffMatcher( // constructor method
        FloatProcessor img, // search image (I)
        FloatProcessor ref) // reference image (R)
    {
        I = img;
        R = ref;
        wI = I.getWidth();
        hI = I.getHeight();
        wR = R.getWidth();
        hR = R.getHeight();
        K = wR * hR;

        // compute the mean (\bar{R}) and variance term (S_R) of the template:
        float sumR = 0; // \sum_R = \sum R(i,j)
        float sumR2 = 0; // \sum_{R^2} = \sum R^2(i,j)
        for (int j = 0; j < hR; j++) {
            for (int i = 0; i < wR; i++) {
                float aR = R.getf(i, j);
                sumR += aR;
                sumR2 += aR * aR;
            }
        }
        meanR = sumR / K; // \bar{R} = \frac{\sum R(i,j)}{K}
        varR = (float) Math.sqrt(sumR2 - K * meanR * meanR);
    }

    // continued...
public FloatProcessor computeMatch() {
    FloatProcessor C = new FloatProcessor(wI-wR+1, hI-hR+1);
    for (int r = 0; r <= wI-wR; r++) {
        for (int s = 0; s <= hI-hR; s++) {
            float d = getMatchValue(r, s);
            C.setf(r, s, d);
        }
    }
    return C;
}

float getMatchValue(int r, int s) {
    float sumI = 0;  // $\Sigma_I = \sum I(r+i, s+j)$
    float sumI2 = 0;  // $\Sigma_{I^2} = \sum (I(r+i, s+j))^2$
    float sumIR = 0;  // $\Sigma_{IR} = \sum I(r+i, s+j) \cdot R(i, j)$
    for (int j = 0; j < hR; j++) {
        for (int i = 0; i < wR; i++) {
            float aI = I.getf(r+i, s+j);
            float aR = R.getf(i, j);
            sumI += aI;
            sumI2 += aI * aI;
            sumIR += aI * aR;
        }
    }
    float meanI = sumI / K;  // $\bar{I}_{r,s} = \Sigma_I/K$
    return (sumIR - K * meanI * meanR) /
            ((float)Math.sqrt(sumI2 - K * meanI * meanI) * varR);
}

} // end of class CorrCoeffMatcher
Examples and Discussion

- We now compare these distance metrics
- **Original image I:** Repetitive flower pattern
- **Reference image R:** one instance of repetitive pattern extracted from I

Now compute various distance measures for this I and R
Examples and Discussion

- **Sum of absolute differences**: performs okay but affected by global intensity changes

\[ d_A(r, s) = \sum_{(i,j) \in R} |I(r + i, s + j) - R(i, j)| \]

- **Maximum difference**: Responds more to lighting intensity changes than pattern similarity

\[ d_M(r, s) = \max_{(i,j) \in R} |I(r+i, s+j) - R(i, j)| \]
Examples and Discussion

- **Sum of squared (euclidean) distances:** performs okay but affected by global intensity changes

\[ d_E(r, s) = \left[ \sum_{(i,j) \in R} (I(r+i, s+j) - R(i,j))^2 \right]^{1/2} \]

- **Global cross correlation:** Local maxima at true template position, but is dominated by high-intensity responses in brighter image parts

\[ (I \otimes R)(r, s) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(r+i, s+j) \cdot R(i,j) \]
Examples and Discussion

- **Normalized cross correlation**: results similar to euclidean distance (affected by global intensity changes)

\[
\sum_{(i,j) \in R} I(r+i, s+j) \cdot R(i, j) \\
\left[ \sum_{(i,j) \in R} I^2(r+i, s+j) \right]^{1/2} \cdot \left[ \sum_{(i,j) \in R} R^2(i, j) \right]^{1/2}
\]

- **Correlation coefficient**: yields best results. Distinct peaks produced for all 6 template instances, unaffected by lighting

\[
C_L(r, s) = \sum_{(i,j) \in R} (I(r+i, s+j) \cdot R(i, j)) - K \cdot \bar{I}_{r,s} \cdot \bar{R} \\
\left[ \sum_{(i,j) \in R} I^2(r+i, s+j) - K \cdot \bar{I}_{r,s}^2 \right]^{1/2} \cdot S_R
\]
Effects of Changing Intensity

- To explore effects of globally changing intensity, raise intensity of reference image $R$ by 50 units
- Distinct peaks disappear in Euclidean distance
- Correlation coefficient unchanged, robust measure in realistic lighting conditions
Euclidean Distance under Global Intensity Changes

- Local peaks disappear as template intensity (and thus distance) is increased
Shape of Template

- Template does not have to be rectangular
- Some applications use circular, elliptical or custom-shaped templates
- Non-rectangular templates stored in rectangular array, but pixels in template marked using a mask
- More generally, a weighted function can be applied to template elements
Matching under Rotation and Scaling

**Simple Approach:**
- Store multiple rotated and scaled versions of template
- Computationally prohibitive

**Alternate approaches:**
- Matching in logarithmic-polar space (complicated!)
- Affine matching use local statistical features invariant under affine image transformations (including rotation and scaling)
Matching Binary Images

- **Direct Comparison:**
  - Count the number of identical pixels in search image and template
  - Small total difference when most pixels are same
- **Problem:** Small shift, rotation or distortion of image create high distance
- **Need a more tolerant measure**

![Binary Images](image.png)
The Distance Transform

- For every position \((u,v)\) in the search image \(I\), record distance to closest foreground pixel

- So, for binary image

\[
\begin{align*}
FG(I) &= \{ p \mid I(p) = 1 \} \\
BG(I) &= \{ p \mid I(p) = 0 \}
\end{align*}
\]

- Distance transform is defined as

\[
D(p) = \min_{p' \in FG(I)} \text{dist}(p, p')
\]

- Examples of distance measures are Euclidean distance

\[
d_E(p, p') = \| p - p' \| = \sqrt{(u-u')^2 + (v-v')^2} \in \mathbb{R}^+
\]

- Or Manhattan distance

\[
d_M(p, p') = |u - u'| + |v - v'| \in \mathbb{N}_0
\]
Distance Transform Example

- Example using Manhattan distance
Chamfer Algorithm

- Efficient method to compute distance transform
- Similar to sequential region labeling
- Traverses image twice
  - First, starting at upper left corner of image, propagates distance values downward in diagonal direction
  - Second traversal starts at bottom right, proceeds in opposite direction (bottom to top)
- For each traversal, the following masks is used for propagating distance values

\[
M^L = \begin{bmatrix}
  m_2^L & m_3^L & m_4^L \\
  m_1^L & \times & . \\
  . & . & .
\end{bmatrix}
\quad M^R = \begin{bmatrix}
  \cdot & \cdot & \cdot \\
  \cdot & \times & m_1^R \\
  m_4^R & m_3^R & m_2^R
\end{bmatrix}
\]
Chamfer Distance

- Specifically, for masks for Manhattan distance

\[
M^L_M = \begin{bmatrix}
2 & 1 & 2 \\
1 & \times & . \\
. & . & .
\end{bmatrix}
\quad \quad
M^R_M = \begin{bmatrix}
. & . & . \\
. & \times & 1 \\
2 & 1 & 2
\end{bmatrix}
\]

- And masks for Euclidean distance

\[
M^L_E = \begin{bmatrix}
\sqrt{2} & 1 & \sqrt{2} \\
1 & \times & . \\
. & . & .
\end{bmatrix}
\quad \quad
M^R_E = \begin{bmatrix}
. & . & . \\
. & \times & 1 \\
\sqrt{2} & 1 & \sqrt{2}
\end{bmatrix}
\]

- Floating point-operations can be avoided using distance masks with scaled integer values for Euclidean distance such as

\[
M^L_{E'} = \begin{bmatrix}
4 & 3 & 4 \\
3 & \times & . \\
. & . & .
\end{bmatrix}
\quad \quad
M^R_{E'} = \begin{bmatrix}
. & . & . \\
. & \times & 3 \\
4 & 3 & 4
\end{bmatrix}
\]
Chamfer Matching

- Uses distance transform for matching binary images
- Finds points of maximum agreement between binary search image \( I \) and binary reference image \( R \)
- Accumulates values of distance transform as match score \( Q \)
- At each position, \((r,s)\) of the template \( R \), distance values to all foreground pixels are accumulated

\[
Q(r, s) = \frac{1}{K} \cdot \sum_{(i,j)\in FG(R)} D(r + i, s + j)
\]

where \( K = |FG(R)| \) is number of foreground pixels in template \( R \)

- Zero \( Q \) score = maximum match
- Large \( Q \) score = large deviations
- Best match corresponds to global minimum of \( Q \)
Chamfer Matching

1: **CHAMFERMATCH** \( (I, R) \)
   \( I \): binary search image of size \( w_I \times h_I \)
   \( R \): binary reference image of size \( w_R \times h_R \)
   Returns a two-dimensional map of match scores.

**Step 1—initialize:**
2: \( D \leftarrow \text{DISTANCE TRANSFORM}(I) \) \( \triangleright \) see Alg. 17.2
3: \( K \leftarrow \text{number of foreground pixels in } R \)
4: \( Q \leftarrow \text{new match map of size } (w_I-w_R+1) \times (h_I-h_R+1), \quad Q(r, s) \in \mathbb{R} \)

**Step 2—compute the match score:**
5: for \( r \leftarrow 0 \ldots (w_I-w_R) \) do \( \triangleright \) place \( R \) at \( (r, s) \)
6:   for \( s \leftarrow 0 \ldots (h_I-h_R) \) do
   Get match score for template placed at \( (r, s) \):
7:     \( q \leftarrow 0 \)
8:     for \( i \leftarrow 0 \ldots (w_R-1) \) do
9:       for \( j \leftarrow 0 \ldots (h_R-1) \) do
10:          if \( R(i, j) = 1 \) then \( \triangleright \) foreground pixel in template
11:             \( q \leftarrow q + D(r+i, s+j) \)
12:     \( Q(r, s) \leftarrow q/K \)
13: return \( Q \).

- Compute distance transform \( D \) of image using Chamfer algorithm
- Accumulate sum of distance values
- For all foreground pixels in template \( R \)
- Results stored in 2D match map \( D \)
Comparing Direct Pixel comparison and Chamfer Matching

- Chamfer match score $Q$ much smoother than direct comparison
  - Distinct peaks in places of high similarity
1: \textbf{DistanceTransform} \((I)\)
\hspace{1em} \(I\): binary image of size \(M \times N\).
Returns the distance transform of image \(I\).

\textbf{STEP 1—INITIALIZE:}
2: \(D \leftarrow \text{new} \ distance \ map \ \text{of size} \ M \times N, \ D(u, v) \in \mathbb{R}\)
3: \textbf{for} all image coordinates \((u, v)\) \textbf{do}
4: \hspace{1em} \textbf{if} \(I(u, v) = 1\) \textbf{then}
5: \hspace{2em} \(D(u, v) \leftarrow 0\) \hspace{1em} \triangleright \text{foreground pixel (zero distance)}
6: \hspace{1em} \textbf{else}
7: \hspace{2em} \(D(u, v) \leftarrow \infty\) \hspace{1em} \triangleright \text{background pixel (infinite distance)}

\textbf{STEP 2—L→R PASS (using distance mask} \(M_{L}^{R} = m_{i}^{L}\):)
8: \textbf{for} \(v \leftarrow 1, 2, \ldots, N-1\) \textbf{do} \hspace{1em} \triangleright \text{top} → \text{bottom}
9: \hspace{2em} \textbf{for} \(u \leftarrow 1, 2, \ldots, M-2\) \textbf{do} \hspace{1em} \triangleright \text{left} → \text{right}
10: \hspace{3em} \textbf{if} \(D(u, v) > 0\) \textbf{then}
11: \hspace{4em} \(d_{1} \leftarrow m_{1}^{L} + D(u-1, v)\)
12: \hspace{4em} \(d_{2} \leftarrow m_{2}^{L} + D(u-1, v-1)\)
13: \hspace{4em} \(d_{3} \leftarrow m_{3}^{L} + D(u, v-1)\)
14: \hspace{4em} \(d_{4} \leftarrow m_{4}^{L} + D(u+1, v-1)\)
15: \hspace{3em} \(D(u, v) \leftarrow \min(d_{1}, d_{2}, d_{3}, d_{4})\)

\textbf{STEP 3—R→L PASS (using distance mask} \(M_{R}^{L} = m_{i}^{R}\):)
16: \textbf{for} \(v \leftarrow N-2, \ldots, 1, 0\) \textbf{do} \hspace{1em} \triangleright \text{bottom} → \text{top}
17: \hspace{2em} \textbf{for} \(u \leftarrow M-2, \ldots, 2, 1\) \textbf{do} \hspace{1em} \triangleright \text{right} → \text{left}
18: \hspace{3em} \textbf{if} \(D(u, v) > 0\) \textbf{then}
19: \hspace{4em} \(d_{1} \leftarrow m_{1}^{R} + D(u+1, v)\)
20: \hspace{4em} \(d_{2} \leftarrow m_{2}^{R} + D(u+1, v+1)\)
21: \hspace{4em} \(d_{3} \leftarrow m_{3}^{R} + D(u, v+1)\)
22: \hspace{4em} \(d_{4} \leftarrow m_{4}^{R} + D(u-1, v+1)\)
23: \hspace{3em} \(D(u, v) \leftarrow \min(D(u, v), d_{1}, d_{2}, d_{3}, d_{4})\)

24: \textbf{return} \(D\).
Distance Transform using Chamfer Algorithm
Digital Image Processing (CS/ECE 545)
Lecture 11: Future Directions

Prof Emmanuel Agu

Computer Science Dept.
Worcester Polytechnic Institute (WPI)
Recall: Electromagnetic Spectrum and IP

- Images can be made from any form of EM radiation

From Wikipedia
Recall: Images from Different EM Radiation

- Radar imaging (radio waves)
- Magnetic Resonance Imaging (MRI) (Radio waves)
- Microwave imaging
- Infrared imaging
- Photographs
- Ultraviolet imaging telescopes
- X-rays and Computed tomography
- Positron emission tomography (gamma rays)
- Ultrasound (not EM waves)
Medical Imaging Example Technologies

- XRay
- Computerized tomography
- Mammogram
- Nuclear magnetic resonance
- Positron Emission Tomography
- Single Photon Emission Computerized Tomography
- Ultrasound imaging
XRay

- Imaging body internals using electromagnetic waves of wavelength 0.01 to 10 nanometers
Computerized Tomography

- **Tomography**: Cross-sectional image formed from projections
- **Example**: XRay Computerized tomography of human brain
- Virtual slices allow human to see inside without cutting open
Ultrasound

- Uses sound waves undetectable by human ear
- Non-invasive imaging, used for imaging unborn babies
Computer Vision

- Vision builds on Image processing
- Inverse problem to computer graphics

Vision and graphics

Images $\xrightarrow{\text{Vision}}$ Model

Graphics $\xleftarrow{\text{Inverse problems: analysis and synthesis}}$

Courtesy Grauman U of Texas
Why do we need Computer Vision?

- Explosion of visual content
- Let computers help humans with “easy” tasks
Computer Vision

- **Classic CV task:** Recognize objects in image
- **First step:** Describe images using distinct features (textures, colors, edges, etc)
- **Outputs of image processing = inputs for CV**

Courtesy Grauman U of Texas
Grouping & fitting

Clustering, segmentation, fitting; what parts belong together?

Hough transform, etc
Recognition and learning

Recognizing objects and categories, learning techniques

Courtesy Grauman U of Texas
Digital Forensics

- Detecting when images have been tampered with
- Has been around for a long time
- Example: 1961 Grigoriy Nelyubov, one of astronauts removed from image of Russian astronauts on moon for misbehavior
Computational Photography

- Traditional camera: only configurable settings
- Computational camera: More parts programmable
  - Programmable illumination: complex flash patterns
  - Programmable apertures, shutter, etc
  - Programmable image processing
- What’s possible?
Tone Mapping, Color Correction on Camera

- Courtesy Fredo Durand MIT
Depth from Image using programmable aperture

- Courtesy Bill Freeman, MIT
ReFocus badly focussed Images

- Courtesy Fredo Durand MIT
Computational Photography

- What’s possible?
  - Deblurring: Take out motion blur artifacts
Exam Overview

- Wednesday, April 30, 2014, in-class
- Midterm covered up to lecture 5 (Corner Detection)
- Final covers lecture 6 till today’s class (lecture 11)
- Can bring:
  - One page cheat-sheet, hand-written (not typed)
  - Calculator
- Will test:
  - Theoretical concepts
  - Mathematics
  - Algorithms
  - Programming
  - ImageJ knowledge (program structure and some commands)
What am I Really Testing?

- Understanding of
  - concepts (NOT only programming)
  - programming (pseudocode/syntax)

- Test that:
  - you can plug in numbers by hand to check your programs
  - you did the projects
  - you understand what you did in projects
General Advise

- **Read your projects** and refresh memory of what you did
- **Read the slides**: worst case – if you understand slides, you’re more than 50% prepared
- Focus on **Mathematical results, concepts, algorithms**
- Plug numbers: calculate by hand
- Try to **predict subtle changes** to algorithm.. What ifs?..
- **Past exams**: One sample final will be on website
- All lectures have references. Look at refs to focus reading
- Do all readings I asked you to do on your own
Grading Policy

- I try to give as much partial credit as possible
- In time constraints, laying out outline of solution gets you healthy chunk of points
- Try to write something for each question
- Many questions will be easy, exponentially harder to score higher in exam
Topics

- Curve Detection
- Morphological Filters
- Regions in Binary Images
- Color Images
- Introduction to Spectral Techniques
- Discrete Fourier Transform
- Geometrical Operations
- Comparing Images
- Future Directions
References

- University of Utah, CS 4640: Image Processing Basics, Spring 2012
- Rutgers University, CS 334, Introduction to Imaging and Multimedia, Fall 2012
- CS 376 Slides, Computer Vision, Fall 2011