Basics of Color Image Processing

- Color transformations or color mappings
  - processing pixels of each color plane based strictly on their values and not on their spatial coordinates;
  - equivalent to intensity transforms of gray scale images
- Spatial processing of individual color planes
  - spatial (neighborhood) filtering of individual color planes
  - analogous to spatial filtering of gray scale images
- Color vector processing
  - techniques based on processing all components of a color image simultaneously;
  - each color point is a vector in RGB coordinate system

Basics of Color Image Processing

- Color pixels are vectors of the form:
  \[
  c = \begin{bmatrix}
  c_R \\
  c_G \\
  c_B
  \end{bmatrix} = \begin{bmatrix}
  c_R(x, y) \\
  c_G(x, y) \\
  c_B(x, y)
  \end{bmatrix} = \begin{bmatrix}
  R \\
  G \\
  B
  \end{bmatrix} = \begin{bmatrix}
  R(x, y) \\
  G(x, y) \\
  B(x, y)
  \end{bmatrix}
  \]

- Image of size \( M \times N \) has \( MN \) vectors \( c(x, y) \) for \( x=0, 1, \ldots, M-1 \), \( y=0, 1, \ldots, N-1 \)

Conversion to Other Color Spaces

- Other color spaces (or color models) include:
  - NTSC – used for television in the U.S.; luminence, hue and saturation (YIQ)
  - YCbCr – used in digital video; luminence (Y), difference between blue and reference, difference between red and reference
  - HSV – (hue, saturation, value) is used to select colors from a color wheel or palette; best match to perceptual color scale
  - CMY and CMYK – (cyan, magenta, yellow, optional black) used for color printers and copies
  - HSI – human descriptions of color objects (hue, saturation, intensity)
Gray Scale Transformations

- imcomplement; computes negative of image
- histeq; depends on gray level distribution, but a fixed transformation once parameters of distribution estimated from image data
- imadjust; needs user input (e.g., to specify gamma and break points) but best done interactively on color images using ICE (Interactive Color Editor)

Color Transformations

- Transformations of color images of the form:
  \[ s_j = T_i(r_i), \quad i = 1,2,...,n \]
  \[ r_i \] and \[ s_j \] are the color components of input and output images
  \[ n \] is the dimension of (number of color components) the color space of \[ r_i \]
  \[ T_i \] is the full color transformation (or mapping) function

- Transformations of monochrome images:
  \[ s_j = T_i(r), \quad i = 1,2,...,n \]
  \[ r \] denotes gray level values
  \[ n \] denotes the number of color components in \[ s_j \]

- Mapping of gray levels into arbitrary colors called pseudocolor transformation or pseudocolor mapping

- Monochrome and color image processing can be made equivalent if we let \[ r_i = r_j = r \]

ICE – Interactive Color Editor

- ICE can be used to specify shapes and control points of color transformations (interp1q and spline) in a user interactive manner
- MATLAB format for call to ICE is:
  \[ g = \text{ice}('Property Name', 'property Value', ...); \]
  \[ g \] is the processed image file
  \[ 'Property Name' \] is one of 3 choices, 'image', 'space' and 'wait'
  \[ f = \text{imread('parrots.tif')}; g = \text{ice('image', f)}; \]

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>An RGB or monochrome input image to be transformed by interactively specified mappings.</td>
</tr>
<tr>
<td>space</td>
<td>The color space of the components to be modified. Possible values are 'gray', 'rgb', 'hsv', 'hsl', 'hls', 'true', or 'gray'.</td>
</tr>
<tr>
<td>wait</td>
<td>Whether interactive processing is to occur.</td>
</tr>
</tbody>
</table>

TABLE 1.7

<table>
<thead>
<tr>
<th>Value of the function f</th>
<th>Handle of the mapped image file</th>
</tr>
</thead>
</table>

Color Transformations – Graphical Specification of Mapping Function

- Linear interpolation in MATLAB:
  \[ z = \text{interp1q}(x, y, xi); \]
  \[ z \] is a column vector containing values of the linearly interpolated function \[ z \] at points \[ xi \]
- Column vectors \[ x \] and \[ y \] specify the horizontal and vertical coordinate pairs of the underlying control points
- Elements of \[ x \] must increase monotonically
- Length of \[ z \] is equal to length of \[ xi \]
- \[ z = \text{interp1q}([0 0.255], [0 255], [0:255]); \] produces a 256 element one-to-one mapping connecting control points (0,0) to (255,255) with \[ z = [0 1 2 ... 255] \]
- Spline interpolation in MATLAB:
  \[ z = \text{spine}(x, y, xi); \] if \[ y \] contains two or more elements than \[ x \], the extra elements are treated as the spline slope at the endpoints
ICE — Interactive Color Editor

Pseudocolor Mappings

(a) X-ray image of weld with several cracks and porosities (bright white, streaks running through middle of the image)

(b) Pseudocolor version generated by mapping green and blue components of RGB image to blue and cyan components of X.TAL color map using ICE mapping functions

(c) Yellow now corresponds with cracks and porosities

Pseudo-Color Image

• pseudo-color is when a monochrome image is represented in RGB color space with independent mapping of each component
  – this results in pseudo-color image where gray levels are now in pseudo-color
  – used to make small changes in grayscale visible

Intensity transformed to 8 pseudo-color regions; image structure more clearly seen in pseudo-color image

Color Transformations – Inverse Map

• Inverse mappings:
  – useful for enhancing detail that is embedded in dark color regions, especially when the regions are dominant in size
  – complement of a primary color is the mixture of the other two primaries; red=green+blue=cyan; green=red+blue=magenta; blue=red+green=yellow; complement useful for enhancing detail embedded in dark regions of color

Two mouse movements to create negative mapping function – image complement equivalent function

Color Transformations – Pseudocolor Mappings

• when a monochrome image is represented in the RGB color space and the resulting components are mapped independently, the transformed result is a pseudocolor image in which input image grayscale levels have been replaced by arbitrary colors.
  – transformations that do this are useful because the human eye can distinguish between millions of colors – but relatively few shades of gray
  – pseudocolor mappings often used to make small changes in grayscale visible to the human eye, or to highlight important grayscale regions
  – primary use of pseudocolor is human visualization – i.e., the interpretation of grayscale events in an image or sequence of images via grayscale-to-color assignments
Example showing how color balancing or color correction greatly improves color image clarity and naturalness: (a) input image (in CMY format) is seen to be heavy in magenta; (b) ICE used to convert RGB image to CMY
\[ f_2 = \text{ice('image', } f_1, \text{'space', 'CMY'}) \]
and then transform magenta component independently of all other components; small change in magenta component (part (c)) has a significant impact on clarity and naturalness of the colors of the image.

ICE Examples – Monochrome and Color Contrast Manipulation

- Histogram of aerial photo using gamma-shaped mapping transformation
- Three color histograms of color photo using S-shaped mapping function -> image contrast increased, small effects on hue

ICE Example-Histogram Equalization

- Histogram equalization is a gray-level mapping process that seeks to produce monochrome images with uniform intensity histograms
  - required mapping function is the cumulative distribution function (CDF) of gray levels in input image
  - cannot successfully equalize components of a color image independently -> this leads to erroneous color
  - better approach is to spread color intensities uniformly, leaving the colors themselves unchanged
Spatial Filtering of Color Images

• Color image smoothing (spatial averaging)
  – smoothing of a monochrome image can be accomplished by multiplying all pixel values by the corresponding coefficients in the spatial mask (all 1s) and dividing by the total number of elements in the mask
  – smoothing a full-color image in RGB space is formulated in the same way as for gray-scale images, except that instead of single pixels, we now deal with vector values

Spatial Filtering of Color Images

Let \( S_c \) denote the set of coordinates defining a neighborhood centered at \((x, y)\) in a color image. The average of the RGB vectors in this neighborhood is:

\[
\mathbf{c}(x, y) - \frac{1}{K} \sum_{(i,j) \in S_c} \mathbf{c}(i, j)
\]

where \( K \) is the number of pixels in the neighborhood. From vector addition we can express this as:

\[
\begin{bmatrix}
\frac{1}{K} \sum_{(i,j) \in S_c} R(x, y) \\
\frac{1}{K} \sum_{(i,j) \in S_c} G(x, y) \\
\frac{1}{K} \sum_{(i,j) \in S_c} B(x, y)
\end{bmatrix}
\]

Spatial Filtering of Color Images

• Each component of vector same as result obtained by performing neighborhood averaging on each individual component image, using standard gray-scale neighborhood processing
• Thus smoothing by neighborhood averaging can be carried out directly in color vector space (i.e., on a per-image-plane basis)

Spatial Filtering of Color Images

Smoothing an RGB color image, \( fc \), with a linear spatial filter consists of the following three steps:
1. Extract the three component images – \( f_R = fc(:, :, 1) \); \( f_G = fc(:, :, 2) \); \( f_B = fc(:, :, 3) \);
2. Filter each component image individually. With \( w \) as a smoothing filter (generated using fspecial), smooth the red component image using – \( f_R_{\text{filtered}} = \text{imfilter}(f_R, w, \text{'replicate'}) \); similarly filter the other two component images
3. Reconstruct the filtered RGB image – \( fc_{\text{filtered}} = \text{cat}(3, f_R_{\text{filtered}}, f_G_{\text{filtered}}, f_B_{\text{filtered}}) \)

Can combine above operations using – \( fc_{\text{filtered}} = \text{imfilter}(fc, w, \text{'replicate'}) \);
Spatial Filtering Example

- we know from previous slides that smoothing the individual component images and forming a composite color image will be the same as smoothing the original RGB image using the imfilter command
  - Slide 7.24 (part (a)) shows the effects of smoothing using an averaging filter of size 25 x 25
  - investigate effects of smoothing only the intensity component of HSI (hue/saturation/intensity) version

Spatial Filtering Example

- MATLAB code for smoothing I component of HIS representation
  - h = rgb2hsi(fb); % convert to HIS representation
  - H = h(:,:,1); % hue component, see 2 slides later
  - S = h(:,:,2); % saturation component, see 2 slides later
  - I = h(:,:,3); % intensity component, see 2 slides later
  - w = fspecial(’average’, 25); % smoothing filter
  - t_filtered = imfilter(I, w, ’replicate’);
  - h = cat(3, H, S, t_filtered);
  - f = hsv2rgb(h); % back to RGB for comparison
  - imshow(f);

- Results shown in slide 7.24, part (b)
  - image is less blurred than smoothed RGB image; see effects due to lack of smoothing of H and S components
  - smoothing all three components leads to additional problems as seen in part (c) of the next slide

Color Image Sharpening

- Same general procedure as image smoothing but using a sharpening filter instead of a smoothing filter – e.g. Laplacian sharpener
  - Laplacian of vector c is of the form:
    \[
    \nabla^2[c(x,y)] = \frac{\partial^2 R(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2} + \frac{\partial^2 B(x,y)}{\partial z^2}
    \]

- Compute the Laplacian of a full color image by computing the Laplacian of each component image separately
Color Edge Detection Using Gradients

The gradient of a 2-D function, \( f(x,y) \), is defined as the vector:
\[
\nabla f = \left[ \frac{\partial f}{\partial x} \right] \left[ \frac{\partial f}{\partial y} \right]
\]

The magnitude of this vector is:
\[
| \nabla f | = \left( \left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2 \right)^{1/2}
\]
which is approximately equal to:
\[
| \nabla f | \approx \sqrt{\left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2}
\]

Gradient points in direction of the maximum rate of change of \( f \) at coordinates \((x,y)\). Angle at which maximum rate of change occurs is:
\[
\alpha(x,y) = \tan^{-1} \left( \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)
\]

Color Edge Detection Using Gradients

• Can compute derivatives at all points in image by convolving (using imfilter routine) the image separately with the two (Sobel) masks shown in previous slide, respectively.
• Can then approximate the gradient image by summing the absolute value of the two filtered images
• Sobel masks can be generated using function fspecial
• This method works well for gray-scale images for edge detection; not so well in RGB color space; therefore still need an effective gradient calculation for RGB images

Color Edge Detection Using Gradients

- In order to compute gradients for RGB color images need to compute the gradient of each component color image and then combine the results
  - unfortunately this is not the same as computing edges in RGB color space directly
  - problem is to define the gradient (magnitude and direction) of the vector \( c(x,y) = [R(x,y); G(x,y); B(x,y)] \)
Define vector dot products as:
\[
\mathbf{u} \cdot \mathbf{v} = u_x v_x + u_y v_y + u_z v_z
\]
Let \( \mathbf{u}, \mathbf{v} \) be unit vectors along the R, G and B axis of RGB color space.

Define the vectors:
\[
\mathbf{r} = \frac{\mathbf{R}}{\| \mathbf{R} \|}, \quad \mathbf{g} = \frac{\mathbf{G}}{\| \mathbf{G} \|}, \quad \mathbf{b} = \frac{\mathbf{B}}{\| \mathbf{B} \|}
\]

\[\mathbf{R} = r \mathbf{R} + g \mathbf{G} + b \mathbf{B}\]

Recall that R, G, and B (and consequently the g's) are functions of \( x \) and \( y \).

\[\frac{\partial}{\partial x} (\mathbf{R} \cdot \mathbf{g}) = \mathbf{g} \cdot \frac{\partial \mathbf{R}}{\partial x} + \mathbf{R} \cdot \frac{\partial \mathbf{g}}{\partial x}
\]

\[\frac{\partial}{\partial y} (\mathbf{R} \cdot \mathbf{g}) = \mathbf{g} \cdot \frac{\partial \mathbf{R}}{\partial y} + \mathbf{R} \cdot \frac{\partial \mathbf{g}}{\partial y}
\]

\[\mathbf{R} = \mathbf{r} \mathbf{R} + \mathbf{g} \mathbf{G} + \mathbf{b} \mathbf{B}\]

\[\theta = \arctan\left(\frac{g_y}{g_x}\right)
\]

The magnitude of the gradient in direction \( \theta \) is:
\[\left|\nabla \mathbf{R}\right| = \sqrt{\left(\frac{\partial \mathbf{R}}{\partial x}\right)^2 + \left(\frac{\partial \mathbf{R}}{\partial y}\right)^2}
\]

\[\nabla \mathbf{R} = \left(\frac{\partial \mathbf{R}}{\partial x}, \frac{\partial \mathbf{R}}{\partial y}\right)
\]

Image Segmentation in RGB Vector Space

Segmentation is a process that partitions an image into regions:
- Broad objective is to segment objects of a specified color range in an RGB image.
- Assume we are given a set of sample color points representative of a color (or a range of colors) of interest.
- Obtain an estimate of the average or mean color that we wish to segment, denoted by vector \( \mathbf{m} \).
- Objective of segmentation is to classify each RGB pixel in a given image as having a color in the specified range — or not.
- Hence we need a measure of similarity, e.g., the Euclidean distance.
Image Segmentation in RGB Vector Space

- Let $z$ denote an arbitrary point in the 3-D RGB space.
- We say that $z$ is similar to $m$ if the distance between them is less than a specified threshold, $T$.
- The Euclidean distance between $z$ and $m$ is given as:
  \[ D(z, m) = \sqrt{(z_x - m_x)^2 + (z_y - m_y)^2 + (z_z - m_z)^2}. \]
- The locus of points such that $D(z, m) < T$ is a solid sphere of radius $T$. Points within or on the surface of the sphere satisfy the specified color criterion; points outside the sphere do not.
- Coding these two sets of points in the image with black and white produces a binary, segmented image.

Image Segmentation in RGB Vector Space

- MATLAB Code for Segmentation:
  - $S = \text{colorseg}(\text{method}, f, T, \text{parameters});$
  - $\text{method}$ either ‘euclidean’ or ‘mahalanobis’
  - $f$ is the RGB color image to be segmented
  - $T$ is the distance threshold
  - input parameters: $m$ for ‘euclidean’ and $m$ and $C$ for ‘mahalanobis’
  - output $S$ is a two-level image (of same size as original) containing 0s in points failing the threshold test, and 1s in points that passed the test.

Image Segmentation in RGB Vector Space

- First obtain samples representing range of colors to be segmented (using ROI method):
  - $\text{mask} = \text{roi_poly}(f)$; % select region(s) interactively
  - $\text{red} = \text{im_multiply}(\text{mask}, f(:, :, 1))$; % identify red region
  - $\text{green} = \text{im_multiply}(\text{mask}, f(:, :, 2))$; % identify green region
  - $\text{blue} = \text{im_multiply}(\text{mask}, f(:, :, 3))$; % identify blue region
  - $\text{g} = \text{cat}(3, \text{red}, \text{green}, \text{blue});$
  - figure, imshow(g);
Image Segmentation in RGB Vector Space

- Meaningful segmentation obtained with ‘euclidean’ option using $T=25$ and $T=50$;
  - results using $T=75$ or $T=100$ produced significant oversegmentation
- Segmentation results obtained with ‘mahalanobis’ option using the same values of $T$ were significantly more accurate
  - reason is that the 3-D color data spread in the ROI is fitted much better in this case with an ellipsoid than with a sphere

Summary

- Discussed the basics of color image processing
  -- discussed basic color transformations such as gamma correction, histogram equalization, color balancing, etc.
  -- showed how to do some simple spatial filtering of color images based on smoothing (blurring) and sharpening (based on gradient computations)
  -- showed how to use the gradient to detect edges and to segment images into regions