Color, Edge and Texture

Ying Wu
Electrical Engineering & Computer Science
Northwestern University
Evanston, IL 60208
yingwu@ece.northwestern.edu

Contents

1 Color Models .......................................................... 2
  1.1 CIE XYZ Color Model ............................................. 2
  1.2 RGB Color Model ................................................ 3
  1.3 HSV Color Model ................................................ 3

2 Edge ...................................................................... 4
  2.1 Edge Detection ...................................................... 4
  2.2 Hough Transformation ............................................ 6

3 Texture .................................................................. 8
  3.1 Texture ............................................................... 8
  3.2 Gabor Filtering .................................................... 8
1 Color Models

Different applications will choose different color models. For example, if you are doing color-based segmentation or tracking, it is not a good idea to use the RGB color model.

1.1 CIE XYZ Color Model

The International Committee on Illumination (CIE) defines three standard primaries X, Y, and Z. And any color could be represented by a linear combination of such three primaries, i.e.,

\[ C_\lambda = \lambda_x X + \lambda_y Y + \lambda_z Z \]

When normalized against the luminance \( \frac{\lambda_x}{\lambda_x + \lambda_y + \lambda_z} \), \( \frac{\lambda_y}{\lambda_x + \lambda_y + \lambda_z} \), \( \frac{\lambda_z}{\lambda_x + \lambda_y + \lambda_z} \),

with \( x + y + z = 1 \). Any color can be represented with just \((x, y)\) components.

![Figure 1: (a) CIE chromaticity diagram, (b) Three primaries can only generate colors inside or on the bounding edges of the triangle, i.e., no set of three primaries can be additively combined to generate all colors.](image)

It is impossible to obtain CIE X,Y,Z primaries, because for some wavelength the value of their spectral radiance is negative, (i.e., not additively anymore). However, given color matching functions alone, one can specify the X,Y,Z coordinates of a color and hence describe it.
1.2 RGB Color Model

A color can be represented by a linear combination of red (630nm), green (530nm) and blue (450nm). So, the colors represented by RGB color model are in the RGB cubic.

1.3 HSV Color Model

HSV color model is a nonlinear transformation of the RGB color model. Human eyes can distinguish about 128 different hues and 130 saturations and 23 shades.

H is hue, S is saturation and V is value. S and V are from 0 to 1, and H is an angle from 0 to $2\pi$. White has $V = 1$ and $S = 0$, and black has $V = 0$, and undetermined H and S. When a color of $V = 1$ and $S = 1$, it is a pure color. Any color could be made by adding white and black into a pure color. For example, we starts at a pure blue with $H = 240^\circ$ and $S = 1$, $V = 1$. To produce a light blue, we can add some white, which reduce S, while keeping H and V. To produce a deep blue, we can add some black, which reduce V, while keeping H and S. And we can also produce other type of blue by adding both white and black which reduce both S and V, but the hue remains the same. This is quite useful, because, say if we want to detect a specific color in an illumination-changing environment, we can always use its H and S components by discarding the V component.

The implementation of the mapping from the RGB space to HSV space is easy. The pieces of code segments are listed below.

2 Edge

Edges are fundamental to image processing.
#include <math.h>

//Input:  h,s,v in range [0,1]
//Output: r,g,b in range [0,1]

void HSV2RGB(float h, float s, float v, float* r, float* g, float* b)
{
    int i;
    float aa, bb, cc, f;
    if (s==0) // grayscale
        *r = *g = *b = v;
    else {
        if (h==1.0) h=0;
        h *= 6.0;
        i = ffloor(h);
        f = h - i;
        aa = v*(1-s);
        bb = v*(1-(s*f));
        cc = v*(1-(s*(1-f)));
        switch(i){
            case 0: *r = v; *g = cc; *b = aa; break;
            case 1: *r = bb; *g = v; *b = aa; break;
            case 2: *r = aa; *g = v; *b = cc; break;
            case 3: *r = aa; *g = bb; *b = v; break;
            case 4: *r = cc; *g = aa; *b = v; break;
            case 5: *r = v; *g = aa; *b = bb; break;
        }
    }
}  

Figure 3: HSV color pixel to RGB color pixel

2.1 Edge Detection

There are two type of approaches to edge detection. One uses image derivatives and edges are detected by finding the local maxima of the image derivatives. The other one uses Laplacians and edges are detected by finding the zero-crossing of image Laplacians. We have image derivative:

$$\nabla I(x,y) = \begin{bmatrix} \partial I(x,y)/\partial x \\ \partial I(x,y)/\partial y \end{bmatrix}$$

And the Laplacian is:

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$$

Before applying image derivatives and Laplacians, we generally apply a low-pass filter S, e.g., a Gaussian filter, to smooth the image. So, we have:

$$K_{\nabla\nabla}(S \otimes I) = (K_{\nabla\nabla} S) \otimes I = \nabla S \otimes I$$

And

$$K_{\nabla^2\nabla^2}(S \otimes I) = (K_{\nabla^2\nabla^2} S) \otimes I = \nabla^2 S \otimes I$$
#include <math.h>
#define NO_HUE -1

// Input: h, s, v in range [0, 1]
// Output: r, g, b in range [0, 1]
void RGB2HSV(float r, float g, float b, float* h, float* s, float* v)
{
    float max = MAX(r, g, b), min = MIN(r, g, b);
    float delta = max - min;
    *v = max;
    if(max!=0.0)
    {
        *s = delta/max;
    }
    else
    {
        *s = 0.0;
    }
    if (*s==0.0) *h = NO_HUE;
    else{
    if (r==max)
        *h = (g-b)/delta;
    else if(g==max)
        *h = 2+(b-r)/delta;
    else if (b==max)
        *h = 4+(r-g)/delta;
    *h *= 60.0;
    if(*h<0) *h += 360.0;
    *h /= 360.0;
    }
}

Figure 4: RGB color pixel to HSV color pixel

It means that we can apply \( \nabla \) or \( \nabla^2 \) on the low-pass filter \( S \), e.g., a Gaussian filter, to get a kernel. Then convolute the image with such a kernel to get the edge.

\[
\begin{align*}
  s(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \\
  \frac{\partial s(x, y)}{\partial x} &= -\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2\sigma^2}\right) \frac{x}{\sqrt{2\pi}\sigma^3} \exp\left(-\frac{x^2}{2\sigma^2}\right) \\
  \frac{\partial s(x, y)}{\partial y} &= -\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) \frac{y}{\sqrt{2\pi}\sigma^3} \exp\left(-\frac{y^2}{2\sigma^2}\right) \\
  \nabla^2 s(x, y) &= \frac{1}{2\pi\sigma^4} \left(\frac{x^2 + y^2}{\sigma^2} - 2\right) \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)
\end{align*}
\]

Some simple edge detector can also be used for simplicity. We can use such detectors as kernels to convolute with images to get image gradient. For example, the Prewitt detector is:

\[
\begin{bmatrix}
  -1 & -1 & -1 \\
  0 & 0 & 0 \\
  1 & 1 & 1
\end{bmatrix}
\quad\text{and}\quad
\begin{bmatrix}
  -1 & 0 & 1 \\
  -1 & 0 & 1 \\
  -1 & 0 & 1
\end{bmatrix}
\]

5
And the Sobel detector is:

\[
\begin{bmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1
\end{bmatrix}
\text{ and }
\begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{bmatrix}
\]

Thresholding the magnitudes of the image gradients, we can get edge points from images. However, nearly all edge detectors notoriously fail at corners, because the assumption that the estimation of the partial derivatives in the x and y directions estimate the gradient will not hold anymore. For example, at sharp corners like \( \Lambda \), these partial derivatives estimates will be very poor.

But we can have a remedy for that. Within a small window, we look at the variation of the orientations of image gradient, i.e.,

\[
H = \sum_{\text{window}} (\nabla I)(\nabla I)^T
\]

\[
= \sum_{\text{window}} \begin{bmatrix}
\frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \\
\frac{\partial I}{\partial y} & \frac{\partial I}{\partial y}
\end{bmatrix}
\]

Then we examine the eigenvalues (\( \lambda_1 \) and \( \lambda_2 \)) of \( H \). If both \( \lambda_1 \) and \( \lambda_2 \) are small, we say that there is no edge within the small window. If \( \lambda_1 \) is large, but \( \lambda_2 \) is small, we say the window may contain one of more edges. But if both \( \lambda_1 \) and \( \lambda_2 \) are large, we can say there is a corner in the window.

### 2.2 Hough Transformation

After edge detection, what we obtain are a set of edge points. Many heuristic method of edge following and edge linking, for example, dynamic programming approach, were proposed to obtain continuous edge curves. However, in some special cases, we know in advance that the curves of interests are straight lines. And the question we shall ask is that, how can we detect lines in images?

Hough transformation is a approach to this problem. Assuming we have a prior knowledge of the object, say, we know they are lines or circles, Hough transformation is to map the curve space to a parametric space, and the detection of a specific curve becomes detect salient points in the parametric space.

In X-Y plane, a line can be represented by:

\[
y = mx + c = f(x; m, c)
\]

where \( m \) and \( c \) are parameters of the line. We can also write:

\[
c = -xm + y = g(m; x, y)
\]

which means that in M-C plane, it is still a line with \( x \) and \( y \) as its parameters.

- A point \((x_0, y_0)\) in X-Y plane will be mapped to a line, i.e., \( c = (-x_0)m + y_0 \), in M-C plane, which is shown in Figure 5;
A set of co-linear points, e.g., points are on a line $y = m_0x + y_0$, in X-Y plane, will be mapped to a set of lines intersecting at the point $(m_0, c_0)$ in M-C plane.

For each edge point in the edge map, we map it to the M-C space. Since a line in the edge map consist of many co-linear edge points, the intersection of such mapped lines will become very strong. We can either thresholding or finding the local maximum to get such intersection, whose coordinates in M-C plane is the parameter of the line we detected. The process can be summarized as:

1. Edge detection to get edge map $E$;
2. ∀ edge point $(x_k, y_k)$, map to a line $c = -x_km + y_k$ in M-C plane $A(m, c)$. If the line passes through $(m_i, c_j)$, then $A(m_i, c_j) = A(m_i, c_j) + 1$.
3. find $(m^*, c^*) = \max A(m_i, c_j)$, then the line detected in X-Y plane is $y = m^*x + c^*$.

You may noticed that when the line is a vertical line, which means that $m = \infty$. Then we need an unnecessary huge computation based on this method. Fortunately, we can convert the Euclidean system to a polar system, i.e., a line will be represented by

$$
\begin{align*}
d & = r_i \sin(\theta - \phi_i) \\
r_i & = \sqrt{x_i^2 + y_i^2} \\
\phi_i & = \tan^{-1}\frac{y_i}{x_i}
\end{align*}
$$

in which $(d, \theta)$ are parameters, and $\theta \in [0, 2\pi]$, as shown in Figure 6. For a vertical line, $\theta = \pi/2$. And we have

$$
\theta = \phi_i + \arcsin \frac{d}{r_i}
$$

Question(a): a point in X-Y plane will be mapped to the $\theta$-$d$ space. So, what will it look like in the $\theta$-$d$ plane? Question(b): if we want to detect a circle, how can we generalize the idea of Hough transformation?
3 Texture

3.1 Texture

Texture is easy to recognize, but hard to define. Also, texture depends on scales. At different scales, textures may look very different. There are three standard research topics regarding to texture:

- Texture representation
- Texture synthesis
- Shape from texture

We may cover some topics on texture synthesis in our later lectures.

Texture representation is to find the textons, and then describe their spatial layout. For example, detecting dots and bars by image filtering and modelling the probabilistic distribution of such dots and bars. So, the texture representation problem is to represent textures in terms of the response of a collection of filters. Many people have come up with different image filters, but what filters should we use? Unfortunately, there is no any canonical answers, and no any rule of thumb.

3.2 Gabor Filtering

Gabor filters are Gaussian filters modulated by sinusoid functions. For example:

\[ G_s(x, y) = \cos(k_x x + k_y y) \exp(-\frac{x^2 + y^2}{2\sigma^2}) \]

\[ G_a(x, y) = \sin(k_x x + k_y y) \exp(-\frac{x^2 + y^2}{2\sigma^2}) \]

Gabor filters can have different scales and different orientations, as show in Figure 7. Figure 7(a) shows the 2-D plots of Gabor filter at different orientations. When filtering image with Gabor filters, edges will produce different responses by different filters. Figure 8 shows a filtering result of a human face by Gabor filters of different scales.
Figure 7: Gabor Filters (a) 2-D plots, (b) 3-D plots

Figure 8: Gabor filtering of a human face