Home Work II

Cheng Xinhao*

Dalian University of Technology
cxh@mail.dlut.edu.cn

March 23, 2019

I. Different color space in image processing I

i. RGB color:

The most widely used color space in computer technology is the RGB color space, which is a model closely related to the structure of the human visual system. Depending on the structure of the human eye, all colors can be viewed as three basic colors - different combinations of red, green, and blue, which are used by most displays. For a three-channel color digital image for each image pixel (x, y), it is necessary to indicate three vector components R, G, B.

ii. Lab color space:

The Lab color space is a color mode developed by the CIE (International Commission on Illumination). Any color in nature can be expressed in Lab space, its color space is larger than RGB space. The Lab color space takes the coordinate Lab, where L is bright; the positive number of a represents red, the negative end represents green; the positive number of b represents yellow, and the negative end represents blue.

iii. CIE-lab/luv(XYZ) color space:

The CIE International Standard Lighting Commission established a series of color space standards for the visible spectrum in 1931. It has three basic quantities, represented by X, Y, and Z. It can represent any color by X, Y, and Z. The values of X, Y, and Z can be expressed linearly by R, G, and B, relative to the RGB color space. , XYZ color space can contain almost all colors that humans can feel.

iv. YCBCR color space:

This color space is mainly based on the fact that the human eye is sensitive to the luminance ratio to the chromaticity, and separates the color component from the luminance component. The principles of early black and white televisions and color televisions came from this. Y represent the luminance, and CB and CR are blue and red concentration offset components.

v. CMYK color space:

CMYK (cyan, magenta, yellow) color space is used in the printing industry. The printing industry expresses colorful colors and tones through the overprinting of different dot area ratios of cyan (C), product (M), and yellow (Y) inks. This is the CMY color space of the three primary colors. In actual printing, four colors of cyan (C), product (M), yellow (Y), and black (BK) are generally used, and the black version is added to the darkness in the middle of printing. When the red, green, and blue primary colors are mixed, white is produced, but black is produced when the three primary colors of cyan, magenta, and yellow are mixed.

```
im1 = imread('color1.jpg');

im1_1 = rgb2lab(im1);
figure
imshowpair(im1,im1_1,'montage');
title('rgb color  to lab color');

im1_2 = ...
      rgb2lab(im1,'Whitepoint','d50');
```

```
figure
imshowpair(im1,im1_2,'montage');
title('rgb color  to lab color with ...
    parameter');

figure
imshow(im1_1(:,:,1),[0,100]);%display ...
    the L* component of the LAB image
title('show the L* component of the ...
    LAB image')

im2 = imread('color2.jpg');
im2_1 = rgb2xyz(im2);
figure
imshowpair(im2,im2_1,'montage');
title('convert from rgb color to ...
    xyz color')

im2_2 = xyz2rgb(im2_1);
figure
imshowpair(im2_1,im2_2,'montage');
title('reverse from the xyz color ...
    to rgb color')

im1_3 = rgb2ycbcr(im1);
figure
imshowpair(im1,im1_3,'montage');
title('convert from rgb color to ...
    ycbcr color');

im1_4 = ycbcr2rgb(im1_2);
figure
imshowpair(im1_2,im1_4,'montage');
title('reverse from the ycbcr color ...
    to rgb color');
```

II. Different filters in image processing I

i. Gaussian filter:

- `h = fspecial(âĂŸgaussianâĂŹ,hsize, ... sigma)`

returns a rotationally symmetric Gaussian low-pass filter of size hsize with standard deviation sigma. Not recommended. Use imgaussfilt or imgaussfilt3 instead.

The Gaussian filter is a linear filter that effectively suppresses noise and smoothes the image. The sigma determines the degree of blurring of the Gaussian fuzzy kernel. Mathematically, the effect of the sigma on the shape of the curve, the smaller the sigma, the higher the sharper the curve, the larger the sigma, the lower the curve and the smoother the curve. Therefore, the smaller the Gaussian radius, the smaller the blur and the larger the Gaussian radius, the greater the degree of blur. In other words, the smaller the sigma, the more concentrated the numerical distribution, and the larger the sigma, the more dispersed the numerical distribution.

*mathwork document suggest use imgaussfiltæĹŰimgaussfilt3 instead of gaussian filter*ãĂĆ

- `B = imgaussfilt(A,sigma)` imageA with a 2-D Gaussian smoothing kernel with standard deviation specified by sigma.
- `B = imgaussfilt(___,Name,Value)` uses name-value pair arguments to control aspects of the filtering.

ii. Mean filter:

- `h = fspecial('average',hsize)`

The mean filter is a common filter in image processing and is mainly used to smooth noise. Its principle is mainly to use the average value of pixels around a pixel to achieve smooth noise.

iii. Median filter:

The median filtering method is a non-linear smoothing technique that sets the gray value of each pixel to the median of the gray values of all pixels in a neighborhood window at that point. Often used to treat salt and pepper noise Median filtering usually uses a sliding window with an odd number of points. The median value of the gray value in the window is used instead of the gray value of the center point. In fact, the gray value in this window is sorted, and then the value is assigned to the center point. Commonly used median filter windows are wired, square, circular, and cross-shaped.

iv. Circular averaging filter:

- `h = fspecial('disk',radius)`

The parameter is radius representing the radius of the area. The default value is 5.returns a

circular averaging filter (pillbox) within the square matrix of size 2*radius+1

v. Motion filter:

- `h = fspecial('motion',len,theta)`

There are two parameters, indicating that the camera moves len pixels in the counterclockwise direction at the theta angle, the default value of len is 9, and the default value of theta is 0.

vi. Prewitt filter:

- `h = fspecial(âĂŸprewitt')`

returns a 3-by-3 filter that emphasizes horizontal edges by approximating a vertical gradient. To emphasize vertical edges, transpose the filter hâĂŹ. The size is [3 3], no parameters. This means that the filter is unique and the elements are fixed.

vii. Laplace Gaussian filter:

- `h = fspecial('log',hsize,sigma)`

The Laplace Gaussian operator has two parameters. hsize indicates the template size. The default value is [3,3]. sigma is the standard deviation of the filter. The unit is pixel. The default value is 0.5.

viii. Laplace filter:

- `h = fspecial('laplacian',alpha)`

The parameter alpha is used to control the shape of the operator. The value range is [0, 1]. The default value is 0.2. The Laplacian filter can no longer specify the size as the filter described above, but must be 3ÃŮ3.

```
h1 = fspecial('motion',20,45);
h2 = fspecial('gaussian',3,0.1);
figure;
imshow(im1(:,:,1));

im1_4 = imfilter(im1(:,1,:),h1);
figure
imshowpair(im1,im1_4,'montage');
title('create a filter and filter ...
    an image');

im1_5 = imfilter(im1(:,:,1),h2);
figure
imshowpair(im1,im1_5,'montage');
title('create a filter and filter ...
    an image');
```

III. ADD NOISE TO AN IMAGE I

i. The description of mathwork:

- `J = imnoise(I,'gaussian')`
- `J = imnoise(I,'gaussian',m)`
- `J = imnoise(I,'gaussian',m,var_gauss)`
- `J = imnoise(I,'poisson')`
- `J = imnoise(I,'salt & pepper')`
- `J = imnoise(I,'salt & pepper',d)`
- `J = imnoise(I,'speckle',var_speckle)`

IV. DENOISE METHOD: I

i. Averaging filter

```
im3_1 = imnoise(im3,'salt & ...
    pepper',0.02);
figure
imshowpair(im3,im3_1,'montage');
title('add sal&pepper noise');

im3_2 = imnoise(im3,'gaussian',0,0.02);
figure
imshowpair(im3,im3_2,'montage');
title('add gaussian noise');

h3 = fspecial('average',5);
im3_3 = imfilter(im3_2,h3);
figure
imshowpair(im3,im3_3,'montage');
title('remove gaussian noise with ...
    averaging filter');
```

ii. Median filter

```
im3_4 = medfilt2(im3_1);
figure
imshowpair(im3_1,im3_4,'montage');
```

```
title('remove salt&pepper noise ...
      with median filter');
```

iii. Adding and average

combine 100 images together and average it.

```
j= imnoise(im3,'gaussian',0,0.02);
figure
imshow(j);
title('after noise');
H1=zeros(size(im3));
for i=1:100
   j=imnoise(im3,'gaussian',0,0.02);
   H1=H1+double(j);
end
H=H1/100;
figure
imshow(uint8(H));
title('denoise by add and averaging');
```

iv. Non-local average

We always smooth the mean value around a target pixel, like the median filter, the Non-local mean filtering means that it uses all the pixels in the image, and these pixels are weighted averaged according to some similarity. The filtered image is sharp and does not lose detail. Find similar regions in the image in units of image blocks, and then average these regions to better filter out Gaussian noise in the image.

V. The color balancing method I

i. Gray world

The gray world algorithm is based on the gray world assumption, which assumes that for an image with a large number of color variations, R, G, B. The average of the three components tends to the same gray value Gray. In the physical sense, the gray world method assumes that the mean of the average reflection of the natural scene on the light is generally fixed, and this value is approximately "gray". The color balance algorithm enforces this assumption on the image to be processed, and the effect of ambient light can be eliminated from the image to obtain the original scene image.

step 1: There are two ways to determine Gray 1.Take a fixed value (such as half of the brightest gray value) 2.Gray = (R+G+B)/3

step 2: Calculate the gain coefficients of the three channels R, G, and B

step 3: For each pixel C in the image, adjust its R, G, B components:

```
Image = imread('BlueImage.jpg');
r=Image(:,:,1);
g=Image(:,:,2);
b=Image(:,:,3);
avgR = mean(mean(r));
avgG = mean(mean(g));
avgB = mean(mean(b));
avgRGB = [avgR avgG avgB];
grayValue = (avgR + avgG + avgB)/3;
scaleValue = grayValue./avgRGB;
newI(:,:,1) = scaleValue(1) * r;
newI(:,:,2) = scaleValue(2) * g;
newI(:,:,3) = scaleValue(3) * b;
figure
imshowpair(Image,newI,'montage');
title('before and use the gray-world');
```

ii. Scale by max

step1:Calculate the sum of R,G,B of each pixel and save it to a temporary memory block

step2: Calculate the first 10

step3: Traversing each point in the image, calculating the average of the cumulative sum of the R,G,B components of all points where the R+G+B value is greater than T

step4: Quantize pixels to [0,255] for each point

```
Image = imread('redImage.jpg');
r=Image(:,:,1);
g=Image(:,:,2);
b=Image(:,:,3);
[x,y,z] = size(Image);
max = r(1,1) + g(1,1) + b(1,1);
avgR = 0;
avgG = 0;
avgB = 0;
count1 = 0;
k1 = 1;
k2 = 1;
k3 = 1;
S = 0;
for i=1:x
    for j=1:y
```

```
        sumR(k1) = r(i,j);
        k1 = k1 + 1;
        sumG(k3) = g(i,j);
        k2 = k2 + 1;
        sumB(k3) = b(i,j);
        k3 = k3 + 1;
        if((r(i,j) + g(i,j) + ...
            b(i,j)) > max)
            max = r(i,j) + g(i,j) + ...
                b(i,j);
        end
        end
    end
for w1 = 1:k1-1
    for e1 = w1 + 1 : k1-1
        if (sumR(w1) < sumR(e1))
            temp = sumR(w1);
            sumR(w1) = sumR(e1);
            sumR(e1) = temp;
        end

    end

end
for w2 = 1:k2-1
    for e2 = w2 + 1 : k2-1
        if (sumG(w2) < sumG(e2))
            temp = sumR(w1);
            sumG(w2) = sumG(e2);
            sumG(e2) = temp;
        end

    end

end
for w3 = 1:k3-1
    for e3 = w2 + 1 : k3-1
        if (sumB(w3) < sumB(e3))
            temp = sumB(w1);
            sumB(w3) = sumB(e3);
            sumB(e3) = temp;
        end

    end

end
for i=1:x
    for j=1:y
        if((r(i,j) + g(i,j) + ...
            b(i,j)) > ...
            (sumR(int32(k1/13))+sumG
        (int32(k2/13))+sumB(int32(k3/13))))
             avgR = avgR + r(i,j);
             avgG = avgG + g(i,j);
             avgB = avgB + b(i,j);
             count1 = count1 + 1;
        end
    end
end

paraR = uint32(max)/uint32(avgR);
paraG = uint32(max)/uint32(avgG);
paraB = uint32(max)/uint32(avgB);

NI(:,:,1) = uint32(r)* paraR;
NI(:,:,2) = uint32(g)* paraG;
NI(:,:,3) = uint32(b)* paraB;
figure;
imshow(Image);
figure;
imshow(NI);
```