

***SCHOOL OF ELECTRONIC ENGINEERING***

电 子 工 程 学 院

**《图 像 分 析 与 处 理》**

**实验指导**

西安邮电大学

电 子 工 程 学 院

2025年5月

**实验三**  **基于Matlab的彩色图像处理**

一、实验目的与要求

1．了解彩色空间的基本概念及转换原理。

2．熟悉Matlab中彩色图像的表示方法。

3. 使用Matlab函数实现各类彩色空间和图像类型的转换。

4. 掌握彩色图像的平滑与锐化算法。

5. 实验Matlab对彩色图像进行分割的处理。

二、实验原理及知识点

彩色图像的数据不仅包含亮度信息，还包含颜色信息。彩色的表示方法是多样化的，最常见的是三基色模型，例如RGB（Red / Green / Blue，红绿蓝）三基色模型，利用RGB三基色可以混合成任意颜色。因此，数字图像处理中最通用面向硬件的模型是RGB模型，用于彩色监视器和彩色视频摄像机等，常规的彩色图像也都是用RGB三基色来表示的，每个像素包括RGB三基色数据，每个基色用1个字节（8位二进制位）表示，则每个像素的数据为3个字节（即24位二进制位），这就是人们常说的24位真彩色。CMY（青，品红，黄）、CMYK（青，品红，黄，黑）针对彩色打印机，HSI（色调，饱和度，亮度）模型更符合人描述颜色的方式，将图像分成彩色和灰度信息适合于本书给出的许多灰度处理技术。

RGB彩色模型每种颜色表现为其红、绿、蓝原始光谱分量，且基于笛卡尔坐标系统。如图1所示。RGB是图像色彩产生的理想模型（彩色照相机成像，监视器显示图像），但其用于彩色描述有较多限制。HSI模型将彩色图像中的亮度分量和彩色信息分离，是彩色图像处理的理想模型。



图1 RGB彩色空间

1. 彩色空间转换——从RGB到HSI的彩色转换

给定一个RGB彩色格式图像，每一个RGB像素的H、S、I分量为



若RGB值在[0,1]范围内，则S、I值均在[0,1]范围内，H值可通过除以360°归一化。

1. 彩色空间转换——从HSI到RGB的彩色转换

在[0，1]内给出HSI值，在相同值域找到RGB值，分三个区域（相隔120°）

1. 彩色图像平滑与锐化
2. 彩色图像平滑

彩色图像平滑，是将灰度图像的空域滤波拓展到彩色图像。不同的是，彩色图像平滑是对向量进行处理。如图2所示。

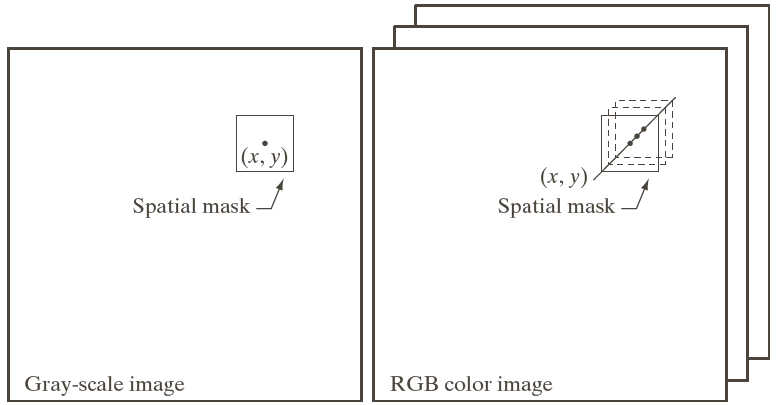


图2 灰度图像与彩色图像空域掩模处理原理

在RGB彩色空间中，令 Sxy表示中心在(x, y)的邻域的坐标集合，在该邻域中，RGB分量向量的平均值：



由向量加法可得：



基于邻域平均的平滑可在单个彩色平面上进行，其结果与采用RGB彩色向量平均相同。

对RGB空间各个分量平滑，改变了原始图像的颜色。仅对HSI空间的强度分量平滑，然后转换到RGB空间，保持了原始图像的色调和饱和度，即保持了原始图像的颜色。

1. 彩色图像锐化

由向量分析，向量的拉普拉斯算子定义为一个向量，其分量等于输入向量各独立分量的拉普拉斯算子。



向量c的拉普拉斯算子为：



可通过计算每一分量图像的拉普拉斯算子，计算全彩色图像的拉普拉斯算子。通过对RGB空间的红、绿、蓝分量计算拉普拉斯算子。仅对HSI空间的强度分量计算拉普拉斯算子，并与未改变的色调和饱和度分量组合。

1. RGB空间中的彩色图像分割

分割是把一幅图像分成区域的处理。在RGB图像中分割具有特定色彩范围的目标：选一个色彩样本集代表感兴趣的颜色，令向量a代表其均值（平均色彩），分割即是对每一RGB像素进行分类，判断其色彩是否在特定范围内。

若以欧式距离作为相似性测度，设*z*为待分类像素，若*z*与*a*间欧式距离小于阈值*D0*，则认为*z*与*a*相似。*z*与*a*间欧式距离为：



把上式进行推广，可得另一距离测度：



*C*是色彩样本集的协方差矩阵。

可采用平方距离作为测度，以避免开方运算，但计算量仍很大。这时，可采用中心在*a*的立方体，其在各维上的长度与色彩样本集的相应标准差成正比。RGB空间彩色图像分割原理如图3所示。



图3 RGB空间彩色图像分割原理

三、实验内容及步骤

1．启动Matlab程序，设置Matlab的程序组中的当前活动文件夹；

2．利用imread( )函数读取一幅图像，将图像数据存入一个数组中；

3．利用imshow( )函数来显示图像；

4．编写Matlab程序，将彩色图像从RGB空间转换为HSI空间，分别显示其H分量、S分量、I分量；

5．编写Matlab程序，将彩色图像从HSI空间转换为RGB空间，分别显示R分量、G分量、B分量；

6．采用7×7盒式均值滤波器对彩色图像的R、G和B分量进行相同平滑，同屏显示原图像和平滑后图像，对结果进行说明；

7．采用7×7盒式均值滤波器对彩色图像的红分量进行平滑，其它分量不变，同屏显示原图像和平滑后图像，对结果进行说明；

8．将图像转换到HSI空间，仅对I分量进行25×25的盒式均值滤波平滑，H、S分量不变，之后在转换回RGB空间显示，对结果进行说明；

9．采用直方图均衡方法，对彩色图像的R、G和B分量进行增强，同屏显示原图像和增强后图像，对结果进行说明；

10. 编写Matlab程序，利用3×3拉普拉斯算子对彩色图像进行处理，之后与原图进行相应运算实现彩色图像的锐化处理，对结果进行说明；

11. 编写Matlab程序，将彩色图像在RGB空间进行彩色分割，并观察其效果；

12．将以上每一步的函数执行语句拷贝下来，写入实验报告，将得到的对应图像结果拷贝下来放入实验报告，并进行结果分析。

四、考核要点

1. 熟悉在Matlab中如何完成彩色图像的不同色空间的转换及分量显示。

2. 熟悉在Matlab中如何实现彩色图像的平滑与锐化。

3. 熟悉在Matlab中如何实现彩色图像的分割处理。

五、实验报告要求

描述实验目的、实验原理及实验基本步骤，用数据和图像给出各个步骤中取得的实验结果，并进行相应的数据分析，就实验中遇到的问题及解决方法进行描述，回答思考题，给出实验总结及程序清单。

六、思考题

(1) 在RGB空间中进行彩色图像平滑，会改变图像的色彩吗？为什么？

七、实验图像

Fig.1 flower.jpg Fig.2 lean.jpg

Fig.3 horse.bmp Fig.4 Peppers.bmp



Fig.5 fruits.jpg

八、参考代码

**1、RGBtoHSI函数**

function [H,S,I]=RGBtoHSI(imagrgb)

[X,Y,Z]=size(imagrgb);

image=im2double(imagrgb);

R=image(:,:,1);

G=image(:,:,2);

B=image(:,:,3);

H=zeros(X,Y);

S=zeros(X,Y);%先赋值

for i=1:X

for j=1:Y

num=0.5\*(R(i,j)-G(i,j)+R(i,j)-B(i,j));

den=sqrt((R(i,j)-G(i,j))^2+(R(i,j)-B(i,j))\*(G(i,j)-B(i,j)));

theta=acos(num/(den+eps)); %·防止为0

if (B(i,j)<=G(i,j))

H(i,j)=theta;

else

H(i,j)=2\*pi-theta;

end

minx=min(R(i,j),G(i,j));

minx=min(B(i,j),minx);

S(i,j)=1-3/(R(i,j)+G(i,j)+B(i,j))\*minx;

end

end

I=1/3\*(R+G+B);

**2、HIStoRGB函数**

function image=HSItoRGB(H,S,I)

[X,Y,Z]=size(I);

image=zeros(X,Y,3);

R=zeros(X,Y);

G=zeros(X,Y);

B=zeros(X,Y);

for i=1:X

for j=1:Y

if (H(i,j)<=(2\*pi/3))

b=I(i,j)\*(1-S(i,j));

r=I(i,j)\*(1+S(i,j)\*cos(H(i,j))/cos(pi/3-H(i,j)));

g=3\*I(i,j)-R(i,j)-B(i,j);

image(i,j,1)=r;

image(i,j,2)=g;

image(i,j,3)=b;

else if(H(i,j)<=(4\*pi/3))

H=H-2\*pi/3;

b=3\*I(i,j)-R(i,j)-G(i,j);

r=I(i,j)\*(1-S(i,j));

g=I(i,j)\*(1+S(i,j)\*cos(H(i,j))/cos(pi/3-H(i,j)));

image(i,j,1)=r;

image(i,j,2)=g;

image(i,j,3)=b;

else if(H(i,j)<=2\*pi)

H=H-4\*pi/3;

g=I(i,j)\*(1-S(i,j));

b=I(i,j)\*(1+S(i,j)\*cos(H(i,j))/cos(pi/3-H(i,j)));

r=3\*I(i,j)-R(i,j)-G(i,j);

image(i,j,1)=r;

image(i,j,2)=g;

image(i,j,3)=b;

end

end

end

end

end

**主函数：**

%==读取分量==

lena\_color=imread('d:\test\_images\lean.jpg','jpg');

[H,S,I] = RGBtoHSI(lena\_color);

%===HIS分量=====

subplot(2,2,1);imshow(lena\_color);title('origin');

H=H/(2\*pi)\*255;%范围0~255

subplot(2,2,2);imshow(H);title('H');

subplot(2,2,3);imshow(S);title('S');

subplot(2,2,4);imshow(I);title('I');

%====HIStoRGB

lena\_rgb=HSItoRGB(H,S,I);

lena\_rgb\_c=im2double(lena\_rgb);

R=lena\_rgb\_c(:,:,1);

G=lena\_rgb\_c(:,:,2);

B=lena\_rgb\_c(:,:,3);

figure(2);

subplot(2,2,1);imshow(lena\_rgb\_c);title('HSItoRGB');

subplot(2,2,2);imshow(R);title('R');

subplot(2,2,3);imshow(G);title('G');

subplot(2,2,4);imshow(B);title('B');

**3、彩色图像平滑与锐化**

%==RGB每个分量滤波============

lena\_color=imread('d:\test\_images\lean.jpg','jpg');

lena\_red=lena\_color(:,:,1);

lena\_green=lena\_color(:,:,2);

lena\_blue=lena\_color(:,:,3);

H=1/49\*ones(7,7);

lena\_red1=imfilter(lena\_red,H);

lena\_green1=imfilter(lena\_green,H);

lena\_blue1=imfilter(lena\_blue,H);

%===组合图像========================

lena(:,:,1)=lena\_red1;

lena(:,:,2)=lena\_green1;

lena(:,:,3)=lena\_blue1;

figure(1);

subplot(1,2,1);imshow(lena\_color);title('origin');

subplot(1,2,2);imshow(lena);title('7\*7');

%==仅对红色分量滤波¨============

lena\_color=imread('d:\test\_images\lean.jpg','jpg');

lena\_red=lena\_color(:,:,1);

H=1/49\*ones(7,7);

lena\_red1=imfilter(lena\_red,H);

%===组合图像============================

lena(:,:,1)=lena\_red1;

lena(:,:,2)=lena\_color(:,:,2);

lena(:,:,3)=lena\_color(:,:,3);

figure(1);

subplot(1,2,1);imshow(lena\_color);title('origin');

subplot(1,2,2);imshow(lena);title('7\*7red');

%==仅对I分量滤波¨============

fc=imread('d:\test\_images\lean.jpg','jpg');

figure,imshow(fc);

h=rgb2hsi(fc); %转换到HIS彩色空间

H=h(:,:,1); %色度分量

S=h(:,:,2); %饱和度分量

I=h(:,:,3); %亮度分量

w=fspecial('average',25); %25×25的方形平滑算子

I\_filtered=imfilter(I,w,'replicate'); %对亮度分量进行平滑操作

H1=cat(3,H,S,I\_filtered); %将H、S、I三个分量重新组合起来f=hsi2rgb(H1); %转换到RGB彩色空间

f=min(f,1); %f的值控制在[0,1]之间

figure,imshow(f); %彩色图像平滑处理

%==RGB每个分量直方图均衡¨============

lena\_color=imread('d:\test\_images\lena.jpg','jpg');

lena\_red=lena\_color(:,:,1);

lena\_green=lena\_color(:,:,2);

lena\_blue=lena\_color(:,:,3);

lena\_red1=histeq(lena\_red);

lena\_green1=histeq(lena\_green);

lena\_blue1=histeq(lena\_blue);

%===组合===================

lena(:,:,1)=lena\_red1;

lena(:,:,2)=lena\_green1;

lena(:,:,3)=lena\_blue1;

figure(1);

subplot(1,2,1);imshow(lena\_color);title('origin');

subplot(1,2,2);imshow(lena);title('jungheng');

%==彩色图像锐化¨============

fb=imread('d:\test\_images\lena.jpg','jpg');

lapmask=[1 1 1;1 -8 1;1 1 1]; %拉普拉斯算子

g=imfilter(fb,lapmask,'replicate'); %拉普拉斯算子处理

figure,imshow(g);

%图像锐化后与原图像相减

fen=imsubtract(fb,g);

figure,imshow(fen); %显示差值图像

**4、RGB空间中的彩色图像分割**

主程序：

clear all;

close all;

f=imread('d:\test\_images\lena.jpg','jpg');

imshow(f);

mask=roipoly(f); %采用交互式的方式提取感兴趣的区域ROI

red=immultiply(mask, f(:, :, 1));

green=immultiply(mask, f(:, :, 2));

blue=immultiply(mask, f(:, :, 3));

g=cat(3, red, green, blue);

figure; imshow(g); %显示感兴趣的区域

[M, N, K]=size(g);

I=reshape(g, M\*N, 3);

idx=find(mask);

I= double(I(idx, 1:3));

[C, m]=covmatrix(I); %计算ROI的协方差矩阵和均值

d=diag(C);

sd=sqrt(d); %计算ROI中RGB个分量的标准差

E25=colorseg('eclidean', f, 25, m) %25是根据最大标准差而取，通常为最大标准差的倍数。按照欧式距离计算

figure; imshow(E25);

E25=colorseg('mahalanobis', f, 25, m) %25是根据最大标准差而取，通常为最大标准差的倍数。按照马氏距离计算

figure; imshow(E25);

子程序（1）：

function [C, m] = covmatrix(X)

%COVMATRIX Computes the covariance matrix of a vector population.

% [C, M] = COVMATRIX(X) computes the covariance matrix C and the

% mean vector M of a vector population organized as the rows of

% matrix X. C is of size N-by-N and M is of size N-by-1, where N is

% the dimension of the vectors (the number of columns of X).

% Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins

% Digital Image Processing Using MATLAB, Prentice-Hall, 2004

% $Revision: 1.4 $

[K, n] = size(X);

X = double(X);

if n == 1 % Handle special case.

C = 0;

m = X;

else

% Compute an unbiased estimate of m.

m = sum(X, 1)/K;

% Subtract the mean from each row of X.

X = X - m(ones(K, 1), :);

% Compute an unbiased estimate of C. Note that the product is

% X'\*X because the vectors are rows of X.

C = (X'\*X)/(K - 1);

m = m'; % Convert to a column vector.

end

子程序（2）：

function [X, R] = imstack2vectors(S, MASK)

%IMSTACK2VECTORS Extracts vectors from an image stack.

% [X, R] = imstack2vectors(S, MASK) extracts vectors from S, which

% is an M-by-N-by-n stack array of n registered images of size

% M-by-N each (see Fig. 11.24). The extracted vectors are arranged

% as the rows of array X. Input MASK is an M-by-N logical or

% numeric image with nonzero values (1s if it is a logical array)

% in the locations where elements of S are to be used in forming X

% and 0s in locations to be ignored. The number of row vectors in X

% is equal to the number of nonzero elements of MASK. If MASK is

% omitted, all M\*N locations are used in forming X. A simple way

% to obtain MASK interactively is to use function roipoly. Finally,

% R is an array whose rows are the 2-D coordinates containing the

% region locations in MASK from which the vectors in S were

% extracted to form X.

% Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins

% Digital Image Processing Using MATLAB, Prentice-Hall, 2004

% $Revision: 1.6 $ $Date: 2003/11/21 14:37:21 $

% Preliminaries.

[M, N, n] = size(S);

if nargin == 1

MASK = true(M, N);

else

MASK = MASK ~= 0;

end

% Find the set of locations where the vectors will be kept before

% MASK is changed later in the program.

[I, J] = find(MASK);

R = [I, J];

% Now find X.

% First reshape S into X by turning each set of n values along the third

% dimension of S so that it becomes a row of X. The order is from top to

% bottom along the first column, the second column, and so on.

Q = M\*N;

X = reshape(S, Q, n);

% Now reshape MASK so that it corresponds to the right locations

% vertically along the elements of X.

MASK = reshape(MASK, Q, 1);

% Keep the rows of X at locations where MASK is not 0.

X = X(MASK, :);

子程序（3）：

function I = colorseg(varargin)

%COLORSEG Performs segmentation of a color image.

% S = COLORSEG('EUCLIDEAN', F, T, M) performs segmentation of color

% image F using a Euclidean measure of similarity. M is a 1-by-3

% vector representing the average color used for segmentation (this

% is the center of the sphere in Fig. 6.26 of DIPUM). T is the

% threshold against which the distances are compared.

%

% S = COLORSEG('MAHALANOBIS', F, T, M, C) performs segmentation of

% color image F using the Mahalanobis distance as a measure of

% similarity. C is the 3-by-3 covariance matrix of the sample color

% vectors of the class of interest. See function covmatrix for the

% computation of C and M.

%

% S is the segmented image (a binary matrix) in which 0s denote the

% background.

% Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins

% Digital Image Processing Using MATLAB, Prentice-Hall, 2004

% $Revision: 1.5 $ $Date: 2003/11/21 14:28:34 $

% Preliminaries.

% Recall that varargin is a cell array.

f = varargin{2};

if (ndims(f) ~= 3) | (size(f, 3) ~= 3)

error('Input image must be RGB.');

end

M = size(f, 1); N = size(f, 2);

% Convert f to vector format using function imstack2vectors.

[f, L] = imstack2vectors(f);

f = double(f);

% Initialize I as a column vector. It will be reshaped later

% into an image.

I = zeros(M\*N, 1);

T = varargin{3};

m = varargin{4};

m = m(:)'; % Make sure that m is a row vector.

if length(varargin) == 4

method = 'euclidean';

elseif length(varargin) == 5

method = 'mahalanobis';

else

error('Wrong number of inputs.');

end

switch method

case 'euclidean'

% Compute the Euclidean distance between all rows of X and m. See

% Section 12.2 of DIPUM for an explanation of the following

% expression. D(i) is the Euclidean distance between vector X(i,:)

% and vector m.

p = length(f);

D = sqrt(sum(abs(f - repmat(m, p, 1)).^2, 2));

case 'mahalanobis'

C = varargin{5};

D = mahalanobis(f, C, m);

otherwise

error('Unknown segmentation method.')

end

% D is a vector of size MN-by-1 containing the distance computations

% from all the color pixels to vector m. Find the distances <= T.

J = find(D <= T);

% Set the values of I(J) to 1. These are the segmented

% color pixels.

I(J) = 1;

% Reshape I into an M-by-N image.

I = reshape(I, M, N);