高级图像处理与分析课程实验



高级图像处理与分析课程实验报告

学号 JG15225001

姓名 祁振东

日期 2016年4月28日

|  |  |
| --- | --- |
| **实验名称** | 灰度变换 |
| **实验内容** | 1、 灰度图像的DFT和IDFT。 具体内容：利用OpenCV提供的cvDFT函数对图像进行DFT和IDFT变换  2、利用理想高通和低通滤波器对灰度图像进行频域滤波 具体内容：利用cvDFT函数实现DFT，在频域上利用理想高通和低通滤波器进行滤波，并把滤波过后的图像显示在屏幕上（观察振铃现象），要求截止频率可输入。  3、利用布特沃斯高通和低通滤波器对灰度图像进行频域滤波。 具体内容：利用cvDFT函数实现DFT，在频域上进行利用布特沃斯高通和低通滤波器进行滤波，并把滤波过后的图像显示在屏幕上（观察振铃现象），要求截止频率和n可输入。 |
| **实验完成情况** | 代码见下面 |
| （包括完成的 |
| 实验内容及 |
| 每个实验的 |
| 完成程度。 |
| 注意要贴出 |
| 每个实验的 |
| 核心代码） |
| **实验中的问题** | 把中心点移到图像的中心  公式的计算还是比较简单的 |
| （包括在实验 |
| 中遇到的问 |
| 题，以及解 |
| 决问题的方 |
| 法） |
| **实验结果** | 三个实验分别放在了三个项目中了。 |
| （实验完成后 |
| 的源码和打 |
| 包文件的说 |
| 明） |

1、空域到频域

|  |
| --- |
| #include <opencv\highgui.h>  #include <opencv2\core\core.hpp>  #include <opencv2\imgproc\imgproc.hpp>  #include <cv.h>  #include <algorithm>  #include <numeric>  #include <vector>  #define path "..\\..\\pic\\lz.jpg"  using namespace cv;  using namespace std;  int main()  {  Mat src = imread(path);  cvtColor(src, src, CV\_BGR2GRAY);//变换成灰度图像  if (!src.data) cout << "open imge error";  imshow("原始图像",src);  int col = getOptimalDFTSize(src.cols);  int row = getOptimalDFTSize(src.rows);  cout << src.cols << src.rows << endl;  //填充边界  copyMakeBorder(src, src, 0, col - src.cols, 0, row - src.rows, BORDER\_CONSTANT, Scalar::all(0));  //imshow("a", src);  Mat planes[] = {Mat\_<float>(src),Mat::zeros(src.size(),CV\_32F)};  Mat compleximg;  merge(planes, 2, compleximg);//compleximg用来存放变换的结果  dft(compleximg, compleximg);  split(compleximg, planes);  magnitude(planes[0], planes[1], planes[0]);  Mat magnitudeimg = planes[0];  magnitudeimg += Scalar::all(1);  log(magnitudeimg, magnitudeimg);//log处理  magnitudeimg = magnitudeimg(Rect(0, 0, magnitudeimg.cols&-2, magnitudeimg.rows&-2));  int cx = magnitudeimg.cols / 2;  int cy = magnitudeimg.rows / 2;  //中心移动到原点  Mat q0(magnitudeimg, Rect(0, 0, cx, cy));  Mat q1(magnitudeimg, Rect(cx, 0, cx, cy));  Mat q2(magnitudeimg, Rect(0, cy, cx, cy));  Mat q3(magnitudeimg, Rect(cx, cy, cx, cy));  Mat tmp;  q0.copyTo(tmp);  q3.copyTo(q0);  tmp.copyTo(q3);  q1.copyTo(tmp);  q2.copyTo(q1);  tmp.copyTo(q2);  normalize(magnitudeimg, magnitudeimg, 0, 1, CV\_MINMAX);//归一化处理  imshow("DFT变换后的图像", magnitudeimg);  cout <<src.cols << src.rows << endl;  waitKey(0);  return 0;  } |

理想低通和理想高通

|  |
| --- |
| #include "opencv2/opencv.hpp"  #include <iostream>  using namespace cv;  using namespace std;  #define PI2 2\*3.141592654  int main()  {  Mat image = imread("..\\..\\pic\\lz.jpg");  cvtColor(image, image, CV\_BGR2GRAY);//变换成灰度图像  ///////////////快速傅里叶变换////////  int oph = getOptimalDFTSize(image.rows);  int opw = getOptimalDFTSize(image.cols);  //Mat padded;  copyMakeBorder(image, image, 0, oph - image.rows, 0, opw - image.cols, BORDER\_CONSTANT, Scalar::all(0));  Mat temp[] = { Mat\_<float>(image), Mat::zeros(image.size(), CV\_32F) };  Mat complexI;  merge(temp, 2, complexI);  dft(complexI, complexI);  split(complexI, temp);  magnitude(temp[0], temp[1], temp[0]);  Mat magnitudeimg = temp[0];  magnitudeimg += Scalar::all(1);  log(magnitudeimg, magnitudeimg);//log处理  magnitudeimg = magnitudeimg(Rect(0, 0, magnitudeimg.cols&-2, magnitudeimg.rows&-2));  int cx = magnitudeimg.cols / 2;  int cy = magnitudeimg.rows / 2;  //中心移动到原点  Mat q0(magnitudeimg, Rect(0, 0, cx, cy));  Mat q1(magnitudeimg, Rect(cx, 0, cx, cy));  Mat q2(magnitudeimg, Rect(0, cy, cx, cy));  Mat q3(magnitudeimg, Rect(cx, cy, cx, cy));  Mat tmp;  q0.copyTo(tmp);  q3.copyTo(q0);  tmp.copyTo(q3);  q1.copyTo(tmp);  q2.copyTo(q1);  tmp.copyTo(q2);  normalize(magnitudeimg, magnitudeimg, 0, 1, CV\_MINMAX);//归一化处理  imshow("DFT频谱", magnitudeimg);  //////////////////////////频域滤波//////  //生成频域滤波核  Mat LXDT(image.size(), CV\_32FC2);  Mat LXGT(image.size(), CV\_32FC2);  float D0 = 500\*500;  // cout << "输入截止频率" << endl;  // cin >> D0;  for(int i=0; i<oph; i++)  {  float \*p = LXDT.ptr<float>(i);  float \*q = LXGT.ptr<float>(i);  for(int j=0; j<opw; j++)  { //高斯低通，高斯高通  //float d = pow(i-oph/2, 2) + pow(j-opw/2, 2);  //p[2\*j] = expf(-d / D0);  //p[2\*j+1] = expf(-d / D0);  //q[2\*j] = 1 - expf(-d / D0);  //q[2\*j+1] = 1 - expf(-d / D0);  float d = pow(i-oph/2, 2) + pow(j-opw/2, 2);  //d = sqrt(d);  if (D0 > d)  {  p[2\*j] = 0;  p[2 \* j + 1] = 0;  q[2 \* j] = 255;  q[2 \* j + 1] = 255 ;  }  else  {  p[2 \* j] = 255;  p[2 \* j + 1] = 255;  q[2 \* j] = 0;  q[2 \* j + 1] = 0;  }  }  }  //理想低通滤波， 理想高通滤波  multiply(complexI, LXDT, LXDT);  multiply(complexI, LXGT, LXGT);  //傅里叶反变换  dft(LXDT, LXDT, CV\_DXT\_INVERSE);  dft(LXGT, LXGT, CV\_DXT\_INVERSE);  Mat dstBlur[2], dstSharpen[2];  split(LXDT, dstBlur);  split(LXGT, dstSharpen);  for(int i=0; i<oph; i++) //中心化  {  float \*p = dstBlur[0].ptr<float>(i);  float \*q = dstSharpen[0].ptr<float>(i);  for(int j=0; j<opw; j++)  {  p[j] = p[j] \* pow(-1, i+j);  q[j] = q[j] \* pow(-1, i+j);  }  }  normalize(dstBlur[0], dstBlur[0], 1, 0, CV\_MINMAX);  normalize(dstSharpen[0], dstSharpen[0], 1, 0, CV\_MINMAX);  imshow("理想低通滤波",dstBlur[0]);  imshow("理想高通滤波",dstSharpen[0]);  waitKey(0);  return 0;  } |

3、布特沃斯低通和高通

|  |
| --- |
| #include "opencv2/opencv.hpp"  #include <iostream>  using namespace cv;  using namespace std;  int main()  {  Mat image = imread("..\\..\\pic\\hc.jpg");  cvtColor(image, image, CV\_BGR2GRAY);//变换成灰度图像  ///////////////快速傅里叶变换////////  int oph = getOptimalDFTSize(image.rows);  int opw = getOptimalDFTSize(image.cols);  //Mat padded;  copyMakeBorder(image, image, 0, oph - image.rows, 0, opw - image.cols, BORDER\_CONSTANT, Scalar::all(0));  Mat temp[] = { Mat\_<float>(image), Mat::zeros(image.size(), CV\_32F) };  Mat complexI;  merge(temp, 2, complexI);  dft(complexI, complexI);  split(complexI, temp);  magnitude(temp[0], temp[1], temp[0]);  Mat magnitudeimg = temp[0];  magnitudeimg += Scalar::all(1);  log(magnitudeimg, magnitudeimg);//log处理  magnitudeimg = magnitudeimg(Rect(0, 0, magnitudeimg.cols&-2, magnitudeimg.rows&-2));  int cx = magnitudeimg.cols / 2;  int cy = magnitudeimg.rows / 2;  //中心移动到原点  Mat q0(magnitudeimg, Rect(0, 0, cx, cy));  Mat q1(magnitudeimg, Rect(cx, 0, cx, cy));  Mat q2(magnitudeimg, Rect(0, cy, cx, cy));  Mat q3(magnitudeimg, Rect(cx, cy, cx, cy));  Mat tmp;  q0.copyTo(tmp);  q3.copyTo(q0);  tmp.copyTo(q3);  q1.copyTo(tmp);  q2.copyTo(q1);  tmp.copyTo(q2);  normalize(magnitudeimg, magnitudeimg, 0, 1, CV\_MINMAX);//归一化处理  imshow("DFT频谱", magnitudeimg);  //////////////////////////频域滤波//////  //生成频域滤波核  Mat BTWSDT(image.size(), CV\_32FC2);  Mat BTWSGT(image.size(), CV\_32FC2);  float D0 = 100 \* 100;  // cout << "输入截止频率" << endl;  // cin >> D0;  for (int i = 0; i < oph; i++)  {  float \*p = BTWSDT.ptr<float>(i);  float \*q = BTWSGT.ptr<float>(i);  for (int j = 0; j < opw; j++)  {  float d = pow(i - oph / 2, 2) + pow(j - opw / 2, 2);  p[2 \* j] = 1.0 / (pow(d / D0, 4) + 1);  p[2 \* j + 1] = 1.0 / (pow(d / D0, 4) + 1);  q[2 \* j] = 1.0 / (pow(D0 / d, 4) + 1);  q[2 \* j + 1] = 1.0 / (pow(D0 / d, 4) + 1);  }  }  //布特沃斯低通滤波， 布特沃斯高通滤波  multiply(complexI, BTWSDT, BTWSDT);  multiply(complexI, BTWSGT, BTWSGT);  //傅里叶反变换  dft(BTWSDT, BTWSDT, CV\_DXT\_INVERSE);  dft(BTWSGT, BTWSGT, CV\_DXT\_INVERSE);  Mat dstBlur[2], dstSharpen[2];  split(BTWSDT, dstBlur);  split(BTWSGT, dstSharpen);  for (int i = 0; i < oph; i++) //中心化  {  float \*p = dstBlur[0].ptr<float>(i);  float \*q = dstSharpen[0].ptr<float>(i);  for (int j = 0; j < opw; j++)  {  p[j] = p[j] \* pow(-1, i + j);  q[j] = q[j] \* pow(-1, i + j);  }  }  normalize(dstBlur[0], dstBlur[0], 1, 0, CV\_MINMAX);  normalize(dstSharpen[0], dstSharpen[0], 1, 0, CV\_MINMAX);  imshow("布特沃斯低通滤波", dstBlur[0]);  imshow("布特沃斯高通滤波", dstSharpen[0]);  waitKey(0);  return 0;  } |