高级图像处理与分析课程实验报告5

学号	姓名	日期
SA18225428	许强	2019.04.05

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 （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）	3个模块全部完成
实验中的问题 （包括在实验 中遇到的问 题，以及解 决问题的方 法）	参考了网上的DFT IDFT代码
实验结果 （实验完成后 的源码和打 包文件的说 明）	代码注释中含有部分说明

```
//
// Created by XQ on 2019-03-29.
//

//https://www.jianshu.com/p/1c9ddc9a7b38

#include<iostream>
#include<string>
```

```cpp


#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/core/types_c.h>
#include <opencv2/core/core_c.h>
#include <opencv2/highgui/highgui.hpp>

using namespace std;
using namespace cv;

bool DFTAndIDFT(string &src, int flag);
bool idealLowPassFilterOrHighPassFilter(string &src, int flag, double f,int model);
bool idealHighPassFilter(string &src, int flag, double f);
bool butterworthLowPassFilterOrHighPassFilter(string &src, int flag, double f, int n, int model);







int main(){
    string str = "/Volumes/数据/图片/2k/lostwall.jpg";
    cout << "1.DFTAndIDFT:" << DFTAndIDFT(str,0) << endl;
    double f;
    int n;

    while (cout << "输入截止频率f 与 n:" && cin >> f >> n){
        cout << "2.1 idealLowPassFilter:" << idealLowPassFilterOrHighPassFilter(str, 0, f, 0) << endl;
        cout << "2.2 idealHighPassFilter:" << idealLowPassFilterOrHighPassFilter(str, 0, f, 1) << endl;
        cout << "3.1 ButterworthLowPassFilter:" << butterworthLowPassFilterOrHighPassFilter(str, 0, f, n, 0) << endl;
        cout << "3.2 ButterworthHighPassFilter:" << butterworthLowPassFilterOrHighPassFilter(str, 0, f, n, 1) << endl;
    }

    return 0;
}




```

```
50
51  /*灰度图像的 DFT 和 IDFT。
52   *
53   *具体内容：利用 OpenCV 提供的 cvDFT 函数对图像进行 DFT 和 IDFT 变换.
54   *
55   *
56   * */
57
58  bool DFTAndIDFT(string &src, int flag){
59      Mat image = imread(src,flag);
60      imshow("input", image);
61      int w = getOptimalDFTSize(image.cols);
62      int h = getOptimalDFTSize(image.rows);
63      Mat padded;
64      copyMakeBorder(image, padded, 0, h - image.rows, 0, w - image.cols,
    BORDER_CONSTANT, Scalar::all(0));//填充图像保存到padded中
65      Mat plane[] = { Mat_<float>(padded),Mat::zeros(padded.size(),CV_32F)
    };//创建通道
66      Mat complexIm;
67      merge(plane, 2, complexIm);//合并通道
68      dft(complexIm, complexIm);//进行傅立叶变换，结果保存在自身
69      split(complexIm, plane);//分离通道
70      magnitude(plane[0], plane[1], plane[0]);//获取幅度图像，0通道为实数通道，
    1为虚数，因为二维傅立叶变换结果是复数
71      int cx = padded.cols / 2;
72      int cy = padded.rows / 2;//一下的操作是移动图像，左上与右下交换位置，右上与
    左下交换位置
73      Mat temp;
74      Mat part1(plane[0], Rect(0, 0, cx, cy));
75      Mat part2(plane[0], Rect(cx, 0, cx, cy));
76      Mat part3(plane[0], Rect(0, cy, cx, cy));
77      Mat part4(plane[0], Rect(cx, cy, cx, cy));
78      part1.copyTo(temp);
79      part4.copyTo(part1);
80      temp.copyTo(part4);
81      part2.copyTo(temp);
82      part3.copyTo(part2);
83      temp.copyTo(part3);
84      /**/
85      Mat _complexim;
86      complexIm.copyTo(_complexim);//把变换结果复制一份，进行逆变换，也就是恢复原
    图
87      Mat iDft[] = {
    Mat::zeros(plane[0].size(),CV_32F),Mat::zeros(plane[0].size(),CV_32F)
    };//创建两个通道，类型为float，大小为填充后的尺寸
88      idft(_complexim, _complexim);//傅立叶逆变换
89      split(_complexim, iDft);//结果貌似也是复数
```

```
90      magnitude(iDft[0], iDft[1], iDft[0]);//分离通道，主要获取0通道
91      normalize(iDft[0], iDft[0], 1, 0, CV_MINMAX);//归一化处理，float类型的显示范围为0-1,大于1为白色，小于0为黑色
92      imshow("IDFT", iDft[0]);//显示逆变换
93      /**/
94      plane[0] += Scalar::all(1);//傅立叶变换后的图片不好分析，进行对数处理，结果比较好看
95      log(plane[0], plane[0]);
96      normalize(plane[0], plane[0], 1, 0, CV_MINMAX);
97      imshow("DFT", plane[0]);
98      waitKey(0);
99      destroyAllWindows();
100
101
102     return true;
103 }
104
105 /*利用理想高通和低通滤波器对灰度图像进行频域滤波
106  *
107  * 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。
108  *
109  * */
110 bool idealLowPassFilterOrHighPassFilter(string &src, int flag, double f, int model){
111     Mat image = imread(src, 0); // Read the file
112     imshow("input", image);
113     Mat img = image.clone();
114     //cvtColor(src, img, CV_BGR2GRAY);
115     //调整图像加速傅里叶变换
116     if(model == 0) cout << "低通 ";
117     if(model == 1) cout << "高通 ";
118     int M = getOptimalDFTSize(img.rows);
119     int N = getOptimalDFTSize(img.cols);
120     Mat padded;
121     copyMakeBorder(img, padded, 0, M - img.rows, 0, N - img.cols, BORDER_CONSTANT, Scalar::all(0));
122     //记录傅里叶变换的实部和虚部
123     Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
124     Mat complexImg;
125     merge(planes, 2, complexImg);
126     //进行傅里叶变换
127     dft(complexImg, complexImg);
128     //获取图像
129     Mat mag = complexImg;
```

```
130         mag = mag(Rect(0, 0, mag.cols & -2, mag.rows & -2));//这里为什么&上-2
    具体查看opencv文档
131         //其实是为了把行和列变成偶数 -2的二进制是11111111.......10 最后一位是0
132         //获取中心点坐标
133         int cx = mag.cols / 2;
134         int cy = mag.rows / 2;
135         //调整频域
136         Mat tmp;
137         Mat q0(mag, Rect(0, 0, cx, cy));
138         Mat q1(mag, Rect(cx, 0, cx, cy));
139         Mat q2(mag, Rect(0, cy, cx, cy));
140         Mat q3(mag, Rect(cx, cy, cx, cy));
141
142         q0.copyTo(tmp);
143         q3.copyTo(q0);
144         tmp.copyTo(q3);
145
146         q1.copyTo(tmp);
147         q2.copyTo(q1);
148         tmp.copyTo(q2);
149         //Do为自己设定的阀值具体看公式
150
151         //处理按公式保留中心部分
152         for (int y = 0; y < mag.rows; y++) {
153             auto * data = mag.ptr<double>(y);
154             for (int x = 0; x < mag.cols; x++) {
155                 double d = sqrt(pow((y - cy), 2) + pow((x - cx), 2));
156
157                 if(model == 0){
158                     if(d>f) data[x] = 0;//低通
159                 } else if(model == 1){
160                     if(d<f) data[x] = 0;//高通
161                 }
162             }
163         }
164         //再调整频域
165         q0.copyTo(tmp);
166         q3.copyTo(q0);
167         tmp.copyTo(q3);
168         q1.copyTo(tmp);
169         q2.copyTo(q1);
170         tmp.copyTo(q2);
171         //逆变换
172         Mat invDFT, invDFTcvt;
173         idft(mag, invDFT, DFT_SCALE | DFT_REAL_OUTPUT); // Applying IDFT
174         invDFT.convertTo(invDFTcvt, CV_8U);
175
```

```
176         imshow("idealLowPassFilterOrHighPassFilter", invDFTcvt);
177         waitKey(0);
178         destroyAllWindows();
179 
180 
181         return true;
182     }
183 
184     /*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。
185      *
186      * 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。
187      *
188      * */
189     bool butterworthLowPassFilterOrHighPassFilter(string &src, int flag, double f, int n, int model){
190         Mat image = imread(src, 0); // Read the file
191         imshow("原始图像", image);
192 
193         //H = 1 / (1+(D/D0)^2n)
194         Mat img = image.clone();
195         if(model == 0) cout << "低通 ";
196         if(model == 1) cout << "高通 ";
197         //cvtColor(src, img, CV_BGR2GRAY);
198         //调整图像加速傅里叶变换
199 
200         int M = getOptimalDFTSize(img.rows);
201         int N = getOptimalDFTSize(img.cols);
202         Mat padded;
203         copyMakeBorder(img, padded, 0, M - img.rows, 0, N - img.cols, BORDER_CONSTANT, Scalar::all(0));
204 
205         Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
206         Mat complexImg;
207         merge(planes, 2, complexImg);
208 
209         dft(complexImg, complexImg);
210 
211         Mat mag = complexImg;
212         mag = mag(Rect(0, 0, mag.cols & -2, mag.rows & -2));
213 
214         int cx = mag.cols / 2;
215         int cy = mag.rows / 2;
216 
217         Mat tmp;
218         Mat q0(mag, Rect(0, 0, cx, cy));
```

```
219         Mat q1(mag, Rect(cx, 0, cx, cy));
220         Mat q2(mag, Rect(0, cy, cx, cy));
221         Mat q3(mag, Rect(cx, cy, cx, cy));
222 
223         q0.copyTo(tmp);
224         q3.copyTo(q0);
225         tmp.copyTo(q3);
226 
227         q1.copyTo(tmp);
228         q2.copyTo(q1);
229         tmp.copyTo(q2);
230 
231 
232 
233         for (int y = 0; y < mag.rows; y++)
234         {
235             auto * data = mag.ptr<double>(y);
236             for (int x = 0; x < mag.cols; x++)
237             {
238                 //cout << data[x] << endl;
239                 double d = sqrt(pow((y - cy), 2) + pow((x - cx), 2));
240                 //cout << d << endl;
241                 double h = 0;
242 
243                 if(model == 0)  h = 1.0 / (1 + pow(d / f, 2 * n));
244                 if(model == 1)  h = 1.0 / (1 + pow(f / d, 2 * n));
245                 if (h <= 0.5)
246                 {
247                     data[x] = 0;
248                 }
249 
250             }
251         }
252         q0.copyTo(tmp);
253         q3.copyTo(q0);
254         tmp.copyTo(q3);
255         q1.copyTo(tmp);
256         q2.copyTo(q1);
257         tmp.copyTo(q2);
258         //逆变换
259         Mat invDFT, invDFTcvt;
260         idft(complexImg, invDFT, DFT_SCALE | DFT_REAL_OUTPUT); // Applying
    IDFT
261         invDFT.convertTo(invDFTcvt, CV_8U);
262         imshow("butterworthLowPassFilterOrHighPassFilter", invDFTcvt);
263 
264         waitKey(0);
```

```
265         destroyAllWindows();
266         return true;
267     }
```