邻域测度直方图均衡化

一、原因分析

直方图均衡化方法是一对一或者多对一的映射关系，即原图像的某一灰度级或某几个灰度级只能映射为均衡化图像的一个灰度级，因此不能实现理想的均衡。

二、改进思路

要想实现直方图的理想均衡化，就必须破除传统直方图均衡化方法所蕴含的一对一或者多对一映射关系的理论前提，实现灰度级多对多的映射关系。

三、技术路线

1、邻域测度

邻域测度( 或邻域算子)定义为:

f'(x,y)= f(x,y)+ k[f(x,y)- fg(x, y))]

k>0,是锐化系数。

下面解释公式的物理含义。当f(x,y)比它 的8邻域均值大时，变换后邻域测度将比f(x,y)大;相应的，当f(x,y)比它 的8邻域均值小时，变换后邻域测度将比f(x,y)小。因此，邻域测度(或邻域算子)可以看作为一个锐化算子，k (锐化系数)的大小决定了锐化的强度。

2、排序

对邻域测度空间的值进行由小到大的排序。

3、均匀分段

排序完成后，按照原始图像的灰度级数进行均匀分段。例如，如果原始图像是256灰度级的，则均匀分为256段，每段的像素的数目基本相等，最多相差1。

4、均衡化映射

按分段的先后顺序，每段中的数据分别赋值为0,1,…L-1 (L为灰度级数)。然后，每段中的每个数据根据在排序过程中保存的位置关系，映射回图像中。

四、程序实现

以下程序使用c++、opencv、在ubuntu中编程实现。C++程序，输入参数为2个，第一个参数为锐化参数k，第二个参数为图像名。

1.C++程序

#include <iostream>

#include <algorithm>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <vector>

#include <string>

using namespace std;

typedef struct

{

int index;

int pixel;

} sort\_st;

bool compare(sort\_st a, sort\_st b)

{

return a.pixel < b.pixel; //降序

}

// step 1 计算邻域测度

cv::Mat Neighborhood\_measurement(double k, cv::Mat image)

{

cout << "邻域测度" << endl;

cout << "锐化系数" << k << endl;

cv::Mat image\_mean9;

cv::Mat Neighborhood\_measurement\_space;

cv::blur(image, image\_mean9, cv::Size(3, 3));

cv::imshow("均值滤波", image\_mean9);

cv::waitKey(1000);

Neighborhood\_measurement\_space = (9.0 / 8.0 \* k + 1.0) \* image - k \* 9.0 / 8.0 \* image\_mean9;

cv::imshow("邻域测度空间", Neighborhood\_measurement\_space);

cv::waitKey(1000);

return Neighborhood\_measurement\_space;

}

// step 2 排序

vector<sort\_st> sort\_pix(cv::Mat image)

{

cout << "排序" << endl;

int w = image.cols; // - 宽

int h = image.rows; // - 高

vector<sort\_st> sort\_array(w \* h);

for (int i = 0; i < sort\_array.size(); ++i)

{

sort\_array[i].index = i;

}

for (int i = 0; i < w \* h; ++i)

{

int channels = image.channels();

if (channels == 1)

{

//得到初始位置的迭代器

cv::Mat\_<uchar>::iterator it = image.begin<uchar>();

//得到终止位置的迭代器

cv::Mat\_<uchar>::iterator itend = image.end<uchar>();

// - 获得像素

int pixel = \*(it + i);

sort\_array[i].pixel = pixel;

}

}

sort(sort\_array.begin(), sort\_array.end(), compare);

// - 输出看一下是否是排序

/\*

for (int i = 0; i < w \* h; ++i)

{

cout << sort\_array[i].index << ":" << sort\_array[i].pixel << " \*\*";

}

\*/

return sort\_array;

}

// step 3 灰度级别分段

int split\_pix(vector<sort\_st> sort\_pix\_index)

{

cout << "灰度级别分段" << endl;

int x = 0;

x = int(sort\_pix\_index[sort\_pix\_index.size()].pixel - sort\_pix\_index[0].pixel) + 1;

cout << "灰度等级:" << x << endl;

return x;

}

int split\_pix(cv::Mat &image0)

{

int x = 0;

double minVal = 0.0;

double maxVal = 0.0;

cv::minMaxLoc(image0, &minVal, &maxVal);

x = int((maxVal - minVal)) + 1;

cout << "原始灰度等级:" << minVal << ":" << maxVal << "共" << x << "个级别" << endl;

return x;

}

// step 4 灰度级映射

cv::Mat balanced\_mapping(cv::Mat image, vector<sort\_st> sort\_pix\_index, int x)

{

cout << "灰度级映射" << endl;

cv::Mat balanced\_mapping\_image = image \* 0;

int pixel = 0;

int level = 1;

for (int i = 0; i < sort\_pix\_index.size(); ++i)

{

//得到初始位置的迭代器

cv::Mat\_<uchar>::iterator it = balanced\_mapping\_image.begin<uchar>();

int xxx = sort\_pix\_index[i].index;

\*(it + xxx) = pixel;

if (i > sort\_pix\_index.size() \* level / x)

{

++pixel;

++level;

}

// cout << pixel << " " << xxx << " ";

}

return balanced\_mapping\_image;

}

//绘制直方图，src为输入的图像，histImage为输出的直方图，name是输出直方图的窗口名称

double drawHistImg(cv::Mat &src, cv::Mat &histImage, std::string name, double maxValue0)

{

const int bins = 256;

int hist\_size[] = {bins};

float range[] = {0, 256};

const float \*ranges[] = {range};

cv::MatND hist;

int channels[] = {0};

cv::calcHist(&src, 1, channels, cv::Mat(), hist, 1, hist\_size, ranges, true, false);

double maxValue;

cv::minMaxLoc(hist, 0, &maxValue, 0, 0);

int scale = 1;

int histHeight = 256 \* 3;

if (maxValue0 != 0)

{

maxValue = maxValue0;

}

for (int i = 0; i < 2 \* bins; ++i)

{

float binValue = hist.at<float>(i);

int height = cvRound(binValue \* histHeight / maxValue);

cv::rectangle(histImage, cv::Point(5 \* i \* scale, histHeight), cv::Point((5 \* i + 1) \* scale, histHeight - height), cv::Scalar(255));

cv::imshow(name, histImage);

}

return maxValue;

}

int main(int argc, char \*\*argv)

{

double k = atof(argv[1]);

string image\_name = string(argv[2]) + ".png";

cout << image\_name;

cv::Mat image0;

image0 = cv::imread(image\_name, 0);

cv::Mat gray;

// cvtColor(image0, gray, CV\_BGR2GRAY);

// cv::imshow(image\_name, gray);

// cv::imwrite(image\_name, gray);

cv::imshow("原图像", image0);

cv::waitKey(1000);

// step 1 计算邻域测度

cout << "step 1" << endl;

cv::Mat Neighborhood\_measurement\_space;

Neighborhood\_measurement\_space = Neighborhood\_measurement(k, image0);

image\_name = string(argv[2]) + "Neighborhood\_measurement\_space.png";

cv::imwrite(image\_name, Neighborhood\_measurement\_space);

// step 2 获得像素值排序的索引

cout << "step 2 " << endl;

vector<sort\_st> sort\_pix\_index;

sort\_pix\_index = sort\_pix(Neighborhood\_measurement\_space);

// step 3 获取像素灰度级

cout << "step 3 " << endl;

int x = 0;

x = split\_pix(image0);

// step 4 灰度级映射R

cout << "step 4 " << endl;

cv::Mat result;

result = balanced\_mapping(image0, sort\_pix\_index, 256);

cv::imshow("结果", result);

cv::waitKey(1000);

image\_name = string(argv[2]) + "result.png";

cv::imwrite(image\_name, result);

// step 5 绘制灰度直方图

double maxValue0 = 0;

cv::Mat image0\_HistImage = cv::Mat::zeros(256 \* 3, 256 \* 5, CV\_8UC1);

cv::Mat Neighborhood\_measurement\_space\_HistImage = cv::Mat::zeros(256 \* 3, 256 \* 5, CV\_8UC1);

cv::Mat result\_HistImage = cv::Mat::zeros(256 \* 3, 256 \* 5, CV\_8UC1);

maxValue0 = drawHistImg(image0, image0\_HistImage, "原图像直方图", maxValue0);

maxValue0 = drawHistImg(Neighborhood\_measurement\_space, Neighborhood\_measurement\_space\_HistImage, "邻域测度空间图像直方图", maxValue0);

maxValue0 = drawHistImg(result, result\_HistImage, "结果图像直方图", maxValue0);

cvWaitKey(1000);

image\_name = string(argv[2]) + "image0\_HistImage.png";

cv::imwrite(image\_name, image0\_HistImage);

image\_name = string(argv[2]) + "Neighborhood\_measurement\_space\_HistImage.png";

cv::imwrite(image\_name, Neighborhood\_measurement\_space\_HistImage);

image\_name = string(argv[2]) + "result\_HistImage.png";

cv::imwrite(image\_name, result\_HistImage);

return 0;

}

2.CMakelists.txt配置文件

cmake\_minimum\_required(VERSION 2.8)

project(Neighborhood\_measuremen)

#set(CMAKE\_BUILD\_TYPE Debug)

set(CMAKE\_BUILD\_TYPE Release)

find\_package(OpenCV 3.0 QUIET)

if(NOT OpenCV\_FOUND)

find\_package(OpenCV 2.4.3 QUIET)

if(NOT OpenCV\_FOUND)

message(FATAL\_ERROR "OpenCV > 2.4.3 not found.")

endif()

endif()

set(CMAKE\_RUNTIME\_OUTPUT\_DIRECTORY ${PROJECT\_SOURCE\_DIR})

add\_executable(main

main.cpp

)

target\_link\_libraries(main

${OpenCV\_LIBS}

)

五、实验结果

1.图像对比

使用给定图像，设置不同的参数k进行了对比实验、具体结果如下。分别展示了原图像及其统计直方图、不同k值的领域测度图及其统计直方图、不同k值的直方图均衡化结果及其统计直方图。





2.结果分析：

锐化系数k，影响使用邻域测度法时候，所获得的图像的锐化强度。

当k>0时，k越大获得的邻域测度图像的锐化强度越强。当k值过大时，最终产生的图像结果噪声会增大影响直方图均衡化的结果。

当k=0时，获得的邻域测度图像的锐化强度不变。

当k<0时，获得的邻域测度图像会变模糊，有低通滤波的效果。同时最终获得的直方图均衡化的结果也会变得模糊，有低通滤波的效果。

k值的选取并不影响最终直方图均衡化后的统计直方图结果。