Op

C

本节为**opencv数字图像处理**（12）：**图像复原与重建**的第三小节，**逆滤波、维纳滤波、约束最小二乘方滤波和几何均值滤波**，主要包括：四种滤波复原图像的数学推导以及维纳滤波的C++实现。

C

## 1. 逆滤波器

emsp; 若退化函数已知或可以得到一个估计，最简单的图像复原方法就是直接做逆滤波，用退化函数除退化图像的傅立叶变换来计算原始图形的傅立叶变换的估计即：

$$\hat{F}(u,v)=\frac{G(u,v)}{H(u,v)}$$

展开计算为：

$$\hat{F}(u,v)=F(u,v)+\frac{N(u,v)}{H(u,v)}$$

如果退化噪声为0或很小，噪声就会支配估计值$\hat{F}(u,v)$，这时候经常需要限制滤波的频率，使其接近原点。

## 1. 最小均方误差滤波/维纳滤波

这种滤波方法建立在图像和噪声都是随机变量的基础上，目标是找出未污染图像$f(x,y)$的一个估计$\hat{f}$，使它们之间的均方误差最小，误差度量由下式给出：

$$e^2=E\{(f-\hat{f})^2\}$$

假设噪声和图像不相关，二者中有一个有零均值且估计中的灰度级是退化图像中灰度级的线性函数，则误差函数的最小值在频率与中由下式给出：

$$\hat{F}(u,v)=\left[\frac{H^*(u,v)S_f(u,v)}{S_f(u,v)|H(u,v)|^2+S_\eta(u,v)}\right]G(u,v)=\left[\frac{H^*(u,v)}{|H(u,v)|^2+S_\eta(u,v)/S_f(u,v)}\right]G(u,v)$$
$$=\left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2+S_\eta(u,v)/S_f(u,v)}\right]G(u,v)$$

这个结果就是维纳滤波，方括号中的项组成的滤波器称为最小均方差滤波器或最小二乘误差滤波器。式子中：$H(u,v)$ =退化函数；$H*(u,v)$=$H(u,v)$的复共轭；$|H(u,v)|^2=H*(u,v)H(u,v)$；$S_{eta}(u,v)=|N(u,v)|^2$ =噪声的功率谱。

如果噪声为0，那么噪声功率谱消失，维纳滤波简化为逆滤波。

信噪比是一个基于噪声和未退化图像的功率谱为基础的，频率域可用下式近似：

$$\mathrm{SNR}=\frac{\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}|F(u,v)|^2}{\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}|N(u,v)|^2}$$

携带低噪声的图像SNR较高，是表征复原算法的性能的一个重要量度。

均方误差也可以这样描述：

$$\mathrm{MSE}=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[f(x,y)-\hat{f}(x,y)\right]^2$$

而如果把复原图像考虑为“信号”，复原图像和原图像的差考虑为噪声，则空间域中信噪比可定义如下：

$$\mathrm{SNR}=\frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[f(x,y)-\hat{f}(x,y)\right]^2}$$

$f$和$\hat{f}$越接近，这个比值越大。【这个定量与感觉上的图像质量没有很好的必然关系】

当处理白噪声时$|N(u,v)|^2$是一个常数，但是未退化图像和噪声的功率谱通常未知或不能估计，则可用下式近似：

$$\hat{F}(u,v)=\left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2+K}\right]G(u,v)$$

C++实现来自**这篇博文**：

```
#include <iostream>
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

void calcPSF(Mat& outputImg, Size filterSize, int len, double theta);
void fftshift(const Mat& inputImg, Mat& outputImg);
void filter2DFreq(const Mat& inputImg, Mat& outputImg, const Mat& H);
void calcWnrFilter(const Mat& input_h_PSF, Mat& output_G, double nsr);
void edgetaper(const Mat& inputImg, Mat& outputImg, double gamma = 5.0, double beta = 0.2);
```

```
int LEN = 50;
int THETA = 360;
int snr = 8000;
Mat imgIn;
Rect roi;
static void onChange(int pos, void* userInput);

int main(int argc, char* argv[])
{
 string strInFileName = "1.JPG";

 imgIn = imread(strInFileName, IMREAD_GRAYSCALE);
 if (imgIn.empty()) //check whether the image is loaded or not
 {
  cout << "ERROR : Image cannot be loaded..!!" << endl;
  return -1;
 }
 imshow("src", imgIn);

 // it needs to process even image only
 roi = Rect(0, 0, imgIn.cols & -2, imgIn.rows & -2);
 imgIn = imgIn(roi);
 cv::namedWindow("inverse");

 createTrackbar("LEN", "inverse", &LEN, 200, onChange, &imgIn);
 onChange(0, 0);
 createTrackbar("THETA", "inverse", &THETA, 360, onChange, &imgIn);
 onChange(0, 0);
 createTrackbar("snr", "inverse", &snr, 10000, onChange, &imgIn);
 onChange(0, 0);
 imshow("inverse", imgIn);
 cv::waitKey(0);

 return 0;
}

void calcPSF(Mat& outputImg, Size filterSize, int len, double theta)
{
 Mat h(filterSize, CV_32F, Scalar(0));
 Point point(filterSize.width / 2, filterSize.height / 2);
 ellipse(h, point, Size(0, cvRound(float(len) / 2.0)), 90.0 - theta,
  0, 360, Scalar(255), FILLED);
 Scalar summa = sum(h);
```

```
 outputImg = h / summa[0];
 Mat tmp;
 normalize(outputImg, tmp, 1, 0, NORM_MINMAX);
 imshow("psf", tmp);
}
void fftshift(const Mat& inputImg, Mat& outputImg)
{
 outputImg = inputImg.clone();
 int cx = outputImg.cols / 2;
 int cy = outputImg.rows / 2;
 Mat q0(outputImg, Rect(0, 0, cx, cy));
 Mat q1(outputImg, Rect(cx, 0, cx, cy));
 Mat q2(outputImg, Rect(0, cy, cx, cy));
 Mat q3(outputImg, 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);
}
void filter2DFreq(const Mat& inputImg, Mat& outputImg, const Mat& H)
{
 Mat planes[2] = { Mat_<float>(inputImg.clone()), Mat::zeros(inputImg.size(), CV_32F) };
 Mat complexI;
 merge(planes, 2, complexI);
 dft(complexI, complexI, DFT_SCALE);
 Mat planesH[2] = { Mat_<float>(H.clone()), Mat::zeros(H.size(), CV_32F) };
 Mat complexH;
 merge(planesH, 2, complexH);
 Mat complexIH;
 mulSpectrums(complexI, complexH, complexIH, 0);
 idft(complexIH, complexIH);
 split(complexIH, planes);
 outputImg = planes[0];
}
void calcWnrFilter(const Mat& input_h_PSF, Mat& output_G, double nsr)
{
 Mat h_PSF_shifted;
 fftshift(input_h_PSF, h_PSF_shifted);
 Mat planes[2] = { Mat_<float>(h_PSF_shifted.clone()), Mat::zeros(h_PSF_shifted.size(), CV_32F) };
 Mat complexI;
 merge(planes, 2, complexI);
 dft(complexI, complexI);
 split(complexI, planes);
 Mat denom;
```

```
 pow(abs(planes[0]), 2, denom);
 denom += nsr;
 divide(planes[0], denom, output_G);
}
void edgetaper(const Mat& inputImg, Mat& outputImg, double gamma, double beta)
{
 int Nx = inputImg.cols;
 int Ny = inputImg.rows;
 Mat w1(1, Nx, CV_32F, Scalar(0));
 Mat w2(Ny, 1, CV_32F, Scalar(0));
 float* p1 = w1.ptr<float>(0);
 float* p2 = w2.ptr<float>(0);
 float dx = float(2.0 * CV_PI / Nx);
 float x = float(-CV_PI);
 for (int i = 0; i < Nx; i++)
 {
  p1[i] = float(0.5 * (tanh((x + gamma / 2) / beta) - tanh((x - gamma / 2) / beta)));
  x += dx;
 }
 float dy = float(2.0 * CV_PI / Ny);
 float y = float(-CV_PI);
 for (int i = 0; i < Ny; i++)
 {
  p2[i] = float(0.5 * (tanh((y + gamma / 2) / beta) - tanh((y - gamma / 2) / beta)));
  y += dy;
 }
 Mat w = w2 * w1;
 multiply(inputImg, w, outputImg);
}

// Trackbar call back function
static void onChange(int, void* userInput)
{
 Mat imgOut;
 //Hw calculation (start)
 Mat Hw, h;
 calcPSF(h, roi.size(), LEN, (double)THETA);
 calcWnrFilter(h, Hw, 1.0 / double(snr));
 //Hw calculation (stop)
 imgIn.convertTo(imgIn, CV_32F);
 edgetaper(imgIn, imgIn);
 // filtering (start)
 filter2DFreq(imgIn(roi), imgOut, Hw);
 // filtering (stop)
 imgOut.convertTo(imgOut, CV_8U);
```

```
    normalize(imgOut, imgOut, 0, 255, NORM_MINMAX);
    //    imwrite("result.jpg", imgOut);
    imshow("inverse", imgOut);
}
```

## 2. 约束最小二乘方滤波

虽然维纳滤波中可以根据上式来进行估计，但是很少得到合适的解，我们将图像受噪声污染表达为如下的矩阵形式：

$$g = Hf + \eta$$

考虑一个带约束的最小准则函数$C$，定义如下：

$$C = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\nabla^2 f(x,y)\right]^2$$

约束为：

$$\left\|g - H\hat{f}\right\|^2 = \|\eta\|^2$$

在频率域这个最佳化问题的解决方案由如下公式给出：

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)}{|H(u,v)|^2 + \gamma|P(u,v)|^2}\right]G(u,v)$$

$\gamma$是一个参数，调整以满足上2式，$P(u,v)$是函数:

$$p(x,y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

的傅立叶变换，当$\gamma = 0$时最小二乘方滤波退化为直接逆滤波。

$\gamma$可以交互地手动调整，当然也可以迭代计算，步骤如下。

定义一个残差向量$r$为：

$$r = g - H\hat{f}$$

由上3式知，$\hat{F}(u,v)$是$\gamma$的函数，所以$r$是该参数的函数，可以证明：

$$\phi(\gamma) = r^T r = \|r\|^2$$

是$\gamma$的单调递增函数，我们需要调整$\gamma$使得:

$$\|r\|^2 = \|\eta\|^2 \pm a$$

若$||r||^2 = ||\eta||^2$，那么可以严格满足上述约束，具体地，寻找一个$\gamma$值得方法：

- 指定$\gamma$的初始值
- 计算$||r||^2$
- 满足上1式则停止；否则，若$||r||^2 < ||\eta||^2 - a$，增大$\gamma$，若$||r||^2 = ||\eta||^2 + a$，则减少$\gamma$。然后返回步骤2。重新计算最佳估计$\hat{F}(u,v)$　　上面步骤中，为了计算$||r||^2$，由上式3【从此处向上数第三个列出的公式】有：

$$R(u,v) = G(u,v) - H(u,v)\hat{F}(u,v)$$

计算$R(u,v)$的傅里叶反变换得$r(x,y)$，有：

$$\|r\|^2 = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} r^2(x,y)$$

计算$||\eta||^2$的话，首先考虑正负图像的噪声方差：

$$\sigma_\eta^2 = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\eta(x,y) - m_\eta\right]^2$$

其中：

$$m_\eta = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\eta(x,y)$$

为样本均值。注意到上2式【从此处向上数第二个列出的公式】的双重求和即为$||\eta||^2$，给出如下表达式：

$$\|\eta\|^2 = MN\left[\sigma_\eta^2 + m_\eta^2\right]$$

也就是说，仅仅使用噪声的均值和方差的知识，就可以实现一个最佳复原算法，但必须假设噪声和图像灰度值不相关。但是需要指出，约束最小二乘方意义小的最佳复原并不是视觉效果上最好。

### 3. 几何均值滤波

几何均值滤波是维纳滤波的推广：

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)}{|H(u,v)|^2}\right]^\alpha \left[\frac{H^*(u,v)}{|H(u,v)|^2 + \beta\left[\frac{S_\eta(u,v)}{S_f(u,v)}\right]}\right]^{1-\alpha} G(u,v)$$

当$\alpha = 1$，退化为直接逆滤波，当$\alpha = 0$，参数维纳滤波器，参数维纳滤波器在$\beta = 1$时还原为标准的维纳滤波器。如果$\alpha = 1/2$，则滤波器变成相同幂次的两个量的积，也就是几何均值的定义。当$\beta = 1$,随着$\alpha$减小到$1/2$以下，滤波器性能越来越接近逆滤波器，增大到$1/2$以上，越来越接近维纳滤波器。当$\alpha = 1/2$且$\beta = 1$时，称为谱均衡滤波器。

---

*欢迎扫描二维码关注微信公众号 深度学习与数学　　[每天获取免费的大数据、AI等相关的学习资源、经典和最新的深度学习相关的论文研读，算法和其他互联网技能的学习，概率论、线性代数等*

*高等数学知识的回顾*