PRAKTIKUM 9

Restorasi Citra

Materi:

- Noise
- Restorasi Citra

Tujuan Praktikum:

- Mahasiswa dapat melakukan mengembalikan citra yang terkena *noise* ke citra semula

A PENYAJIAN

Restorasi Citra

Restorasi citra merupakan proses untuk mengurangi atau menghilangkan degradasi pada citra. Degradasi yang muncul dapat berupa *noise* atau efek optik atau distorsi saat melakukan akuisisi citra, seperti tidak fokusnya kamera yang menghasilkan citra *blur*. Adapun perbedaan *image enhancement* dan *image restoration* dapat dilihat sebagai berikut:

Tabel 1 Tabel perbandingan *image enhancement* dan *image restoration*

Image Enhancement	*Image Restoration*
Bersifat subjektif	Bersifat objektif
Digunakan untuk memanipulasi citra	Merekonstruksi citra untuk memperbaiki kualitas citra
Memperbaiki tampilan citra untuk tujuan tertentu	Memperbaiki citra yang sudah terkena *noise*

Degradasi citra dapat dimodelkan seperti berikut:

$$g(x, y) = h(x, y) * f(x, y) + n(x, y)$$

Keterangan :

g(x,y) = The degraded image
h(x,y) = spatial representation of the degradation function
**= convolution, spatial filtering*
f(x,y) = input image
n(x,y) = additive noise term

Pada model tersebut, dilakukan *blur filtering* h(x, y) pada citra asli f(x, y). Kemudian terdapat fungsi *random error* n(x, y) atau kita kenal dengan *noise*. Maka, untuk menghasilkan kembali citra asli kita dapat melakukannya seperti berikut.

$$f(x, y) = g(x, y) - n(x, y) / h(x, y)$$

Contoh model degradasi diatas dapat direstorasi pada domain spasial, tetapi ada juga noise yang hanya bisa direstorasi di domain frekuensi. Biasanya restorasi citra dilakukan karena kita sudah mengetahui dan menduga model noise yang terdapat pada citra, sehingga penanganannya akan berbeda-beda tergantung model *noise*-nya.

Noise

Noise merupakan degradasi pada sinyal citra yang terjadi akibat gangguan eksternal. *Error* yang terlihat akan bergantung pada jenis gangguan yang terjadi. Sumber dari noise pada citra dapat dilihat sebagai berikut:

- *Error* muncul ketika suatu citra dikirim secara elektronik dari suatu tempat ke tempat lain melalui satelit, kabel, atau nirkabel
- Sensor panas ketika mengambil gambar
- Nilai ISO yang tinggi pada kamera yang menyebabkan sensor kamera lebih cepat menangkap cahaya
- Ukuran sensor pada kamera

Noise merupakan *random error* yang dapat dimodelkan sebagai fungsi aditif seperti yang dimodelkan pada model degradasi diatas. Beberapa tipe noise yang diperkenalkan adalah *Salt* and *pepper noise, gaussian noise, speckle noise*, dan *periodic noise*.

Salt and Pepper Noise

Secara visual, *noise salt* & *pepper* terdiri dari piksel hitam dan putih yang tersebar pada seluruh citra. Sering disebut *noise impuls*, karena disebabkan oleh gangguan yang tiba-tiba pada sinyal citra. *Noise* ini dapat diatasi dengan *filter mean*, dan *filter median*. Citra *salt and pepper noise* dapat dilihat pada Gambar 1.

Gambar 1 Citra *Salt and pepper noise*

Gaussian Noise

Penyebab adanya *gaussian noise* adalah fluktuasi acak pada sinyal yang ditambahkan pada citra. *Gaussian noise* merupakan *white noise* ideal yang disebabkan oleh fluktuasi acak dalam sinyal karena penyebarannya terdistribusi normal. *Noise* ini dapat dikurangi dengan *filter* spasial seperti *median filtering*, *gaussian smoothing*, dan *mean filtering*. Jika citra dinotasikan dengan I dan *gaussian noise* dengan N, maka model degradasinya yaitu (I + N). Citra *gaussian noise* dapat dilihat pada Gambar 2.

Gambar 2 Citra *gaussian noise*

Speckle Noise

Meskipun secara visual *gaussian* dan *speckle noise* terlihat mirip, tetapi keduanya memiliki pemodelan yang berbeda. *Speckle noise* dapat dimodelkan dengan nilai acak yang dikalikan dengan nilai pixel pada citra. Model degradasinya dapat dituliskan dengan persamaan I(1 + N). Penanganan untuk *speckle noise* dengan menggunakan *filter* spasial. Citra *speckle noise* dapat dilihat pada Gambar 3.

Gambar 3 Citra *speckle noise*

Periodic Noise

Berbeda dengan *noise* lain yang acak, periodic noise merupakan noise yang memiliki pola. Citra yang terdegradasi oleh *noise* ini akan tampak terdapat garis-garis pada citra. Untuk menghilangkan noise ini, dibutuhkan filtering pada domain frekuensi seperti *band reject filtering* dan *notch filter*. Citra *periodic noise* dapat dilihat pada Gambar 4.

Gambar 4 Citra *periodic noise*

B LATIHAN

Domain spasial

Berikut ini merupakan latihan membuat program yang dapat menambahkan dan merestorasi *noise* di domain spasial berupa *salt and pepper noise*, *gaussian noise*, dan *speckle noise*. Citra cameraman.jpg tersedia di LMS pada LKP 9.

```
#include <iostream>
#include <cv.h>
#include <highgui.h>
#include <math.h>

using namespace cv;
using namespace std;

void Add_salt_pepper_noise(Mat &src, Mat &dst, int n)
{
    dst = src.clone();
    int i, j, p, q;
    for(int k=0; k<n; k++)
    {
        i = rand()%dst.cols;
        j = rand()%dst.rows;

        p = rand()%dst.cols;
        q = rand()%dst.rows;


        if(dst.channels() == 1)
        {
            dst.at<uchar>(j,i) = 255;
            dst.at<uchar>(q,p) = 0;
        }
        else if(dst.channels()==3)
        {
            dst.at<Vec3b>(j,i)[0]=255;
            dst.at<Vec3b>(j,i)[1]=255;
            dst.at<Vec3b>(j,i)[2]=255;

            dst.at<Vec3b>(q,p)[0]=0;
            dst.at<Vec3b>(q,p)[1]=0;
            dst.at<Vec3b>(q,p)[2]=0;
        }
    }
}

void Add_gaussian_noise(Mat &src, Mat &dst, float mean, float std)
{
    //gaussian noise
    dst = src.clone();
    Mat gaussian_noise = dst.clone();
    //fills array with normally-distributed random numbers with the
specified mean and the standard deviation
    randn(gaussian_noise,mean,std);
    dst += gaussian_noise;
}
```

```
void Add_speckle_noise(Mat &src, Mat &dst, float mean, float std)
{
    //speckle noise
    dst = src.clone();
    Mat speckle_noise = src.clone();
    randn(speckle_noise,mean,std);
    Mat one_mat = src.clone();
    one_mat = Scalar::all(1);
    dst = src.mul(one_mat + speckle_noise);
}

int main()
{
     //------read image file----------------
     string filename = "cameraman.jpg";
     Mat original = imread(filename,0);
     //-------add salt and pepper noise------
     Mat snpImage;
     Add_salt_pepper_noise(original, snpImage, 1000);
     //-------add gaussian noise ------------
     Mat gnImage;
     Add_gaussian_noise(original, gnImage, 0, 30);
     //-------add speckle noise -------------
     Mat spImage;
     Add_speckle_noise(original, spImage, 0, 0.3);
     //-------show image with noise----------
     imshow(filename+" original", original);
     imshow(filename+" salt and pepper noise", snpImage);
     imshow(filename+" gaussian noise", gnImage);
     imshow(filename+" speckle noise", spImage);
     //-------salt and pepper noise restoration section—
     Mat snpImageMean;
     Mat snpImageMedian;
     blur(snpImage, snpImageMean, Size( 3, 3 ), Point(-1,-1));
     medianBlur(snpImage, snpImageMedian, 5);
     //-------gaussian noise restoration section---------
     Mat gnImageMean;
     Mat gnImageMedian;
     blur(gnImage, gnImageMean, Size( 3, 5 ), Point(-1,-1));
     medianBlur(snpImage, gnImageMedian, 5);
     //-------speckle noise restoration section----------
     Mat spImageMean;
     Mat spImageMedian;
     blur(spImage, spImageMean, Size( 3, 3 ), Point(-1,-1));
     medianBlur(snpImage, spImageMedian, 3);
     //-------show image after restoration---------------
     imshow(filename+" salt and pepper noise restoration, f = mean",
      snpImageMean);
     imshow(filename+" salt and pepper noise restoration, f = median",
      snpImageMedian);
     imshow(filename+" gaussian noise restoration, f = mean",
      gnImageMean);
     imshow(filename+" gaussian noise restoration, f = median",
      gnImageMedian);
     imshow(filename+" speckle noise restoration, f = mean",
      spImageMean);
```

```
     imshow(filename+" speckle noise restoration, f = median",
      spImageMedian);

//=================================================================
         waitKey(0);
         return 0;
}
```

Restorasi yang digunakan pada latihan ini ialah *mean filter*, dan *median filter*. Hasil menjalankan program diatas dapat dilihat pada Gambar 5, Gambar 6, Gambar 7, Gambar 8, dan Gambar 9.

Gambar 5 Citra asli

Gambar 6 Citra hasil penambahan *noise*

Gambar 7 Citra hasil restorasi pada *salt and pepper noise*

Gambar 8 Citra hasil restorasi pada *gaussian noise*

Gambar 9 Citra hasil restorasi pada *speckle noise*

Domain frekuensi

Berikut ini merupakan latihan membuat program yang dapat menambahkan *periodic noise* di domain spasial dan merestorasi *periodic noise* di domain frekuensi.

```
#include <iostream>
#include <cv.h>
#include <highgui.h>
#include <math.h>
#define PERIODIC_NOISE_HORIZONTAL 1
#define PERIODIC_NOISE_VERTICAL 2

using namespace cv;
using namespace std;

void Add_periodic_noise(Mat &src, Mat &dst, double density, int thickness,
int feather, double opacity, int periodic_noise_mode)
{
    dst = src.clone();
    //create noise model
    Mat noiseModel = Mat(dst.rows, dst.cols, CV_8UC1, double(0));
    //create original image selection
    Mat imageSelection = Mat(dst.rows, dst.cols, CV_8UC1, double(255));
    int nNoise = 0;
    if (density > 0) nNoise = round(4/(density));

    if(periodic_noise_mode == PERIODIC_NOISE_VERTICAL)
    {
        for(int y = 0; y < noiseModel.rows; y++)
```

```
        {
            uchar* px = noiseModel.ptr<uchar>(y);
            uchar* pxSel = imageSelection.ptr<uchar>(y);
            for(int x = 0; x < noiseModel.cols; x++)
            {
                if(x%nNoise == 0)
                {
                    if(thickness <= 1){ px[x] = 255; pxSel[x] = 0;}
                    if(thickness > 1)
                    {
                        for(int n = -thickness/2 ; n < thickness/2; n++)
                        {
                            if(x+n > 0 && x+n < noiseModel.cols)
                            { px[x+n] = 255; pxSel[x+n] = 0; }
                        }
                    }
                }
            }
        }
    }
    else if(periodic_noise_mode == PERIODIC_NOISE_HORIZONTAL)
    {
        for(int y = 0; y < noiseModel.rows; y++)
        {
            for(int x = 0; x < noiseModel.cols; x++)
            {
                if(y%nNoise == 0)
                {
                    if(thickness <= 1){
                       noiseModel.at<uchar>(y,x) = 255;
                       imageSelection.at<uchar>(y,x) = 0;}
                    if(thickness > 1)
                    {
                        for(int n = -thickness/2 ; n < thickness/2; n++)
                        {
                            if(y+n > 0 && y+n < noiseModel.rows)
                            { noiseModel.at<uchar>(y+n,x) = 255;
                              imageSelection.at<uchar>(y+n,x) = 0; }
                        }
                    }
                }
            }
        }
    }

    if (feather > 1)
    {
        if(feather%2 == 0) feather += 1;
        blur(noiseModel, noiseModel, Size(feather,feather));
    }
    //apply noise to image
    double beta = 1 - opacity;
    addWeighted( noiseModel, opacity, src, beta, 0.0, dst);

    Mat tempNoise = Mat(dst.rows, dst.cols, CV_8UC1, double(0));
    Mat tempDst = Mat(dst.rows, dst.cols, CV_8UC1, double(0));
```

```
    dst.copyTo(tempNoise, noiseModel);
    src.copyTo(tempDst, imageSelection);

    for(int y = 0; y < dst.rows; y++)
    {
        uchar* pxtNoise = tempNoise.ptr<uchar>(y);
        uchar* pxtDst = tempDst.ptr<uchar>(y);
        uchar* pxDst = dst.ptr<uchar>(y);
        for(int x = 0; x < dst.cols; x++)
        {
            if(pxtNoise[x] > 0) pxDst[x] = pxtNoise[x];
            else{ pxDst[x] = pxtDst[x];}
        }
    }
    imshow("periodic noise model", noiseModel);
}

Mat computeDFT(Mat image) {
    /*
    http://opencv.itseez.com/doc/tutorials/core/discrete_fourier_transfor
m/discrete_fourier_transform.html
    */
    Mat padded;
    int m = getOptimalDFTSize( image.rows );
    int n = getOptimalDFTSize( image.cols );
    copyMakeBorder(image, padded, 0, m-image.rows, 0, n-image.cols,
     BORDER_CONSTANT, Scalar::all(0));
    Mat planes[] = {Mat_<float>(padded), Mat::zeros(padded.size(),
     CV_32F)};
    Mat complex;
    merge(planes, 2, complex);
    dft(complex, complex, DFT_COMPLEX_OUTPUT);
    return complex;
}

void shift(Mat magI) {

    // crop if it has an odd number of rows or columns
    magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));

    int cx = magI.cols/2;
    int cy = magI.rows/2;

    Mat q0(magI, Rect(0, 0, cx, cy));
    Mat q1(magI, Rect(cx, 0, cx, cy));  // Top-Right
    Mat q2(magI, Rect(0, cy, cx, cy));  // Bottom-Left
    Mat q3(magI, Rect(cx, cy, cx, cy)); // Bottom-Right

    Mat tmp; // swap quadrants (Top-Left with Bottom-Right)
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    q1.copyTo(tmp); // swap quadrant (Top-Right with Bottom-Left)
    q2.copyTo(q1);
    tmp.copyTo(q2);
}
```

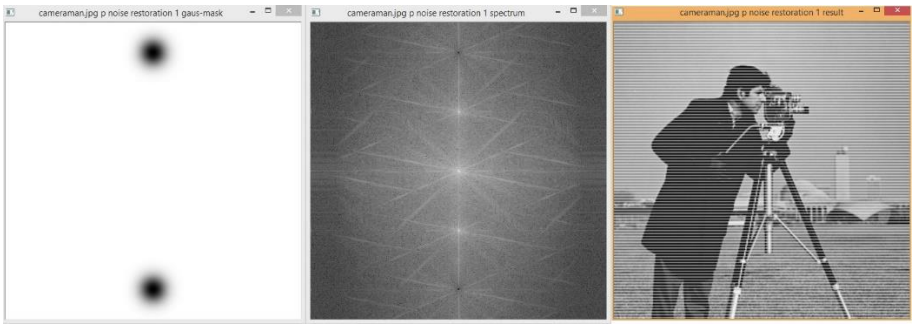
```
void updateResult(Mat complex, string title)
{
    Mat work;
    idft(complex, work);
    Mat planes[] = {Mat::zeros(complex.size(), CV_32F),
     Mat::zeros(complex.size(), CV_32F)};
    split(work, planes);

    magnitude(planes[0], planes[1], work);
    normalize(work, work, 0, 1, NORM_MINMAX);
    imshow(title+" result", work);
}

void updateMag(Mat complex, string title)
{

    Mat magI;
    Mat planes[] = {Mat::zeros(complex.size(), CV_32F),
     Mat::zeros(complex.size(), CV_32F)};
    split(complex, planes);

    magnitude(planes[0], planes[1], magI);

    // switch to logarithmic scale: log(1 + magnitude)
    magI += Scalar::all(1);
    log(magI, magI);

    shift(magI);
    normalize(magI, magI, 1, 0, NORM_INF);

    imshow(title+" spectrum", magI);
}

Mat createGausFilterMask(Size mask_size, int x, int y, int ksize, bool
normalization, bool invert) {
    // Some corrections if out of bounds
    if(x < (ksize / 2)) {
        ksize = x * 2;
    }
    if(y < (ksize / 2)) {
        ksize = y * 2;
    }
    if(mask_size.width - x < ksize / 2 ) {
        ksize = (mask_size.width - x ) * 2;
    }
    if(mask_size.height - y < ksize / 2 ) {
        ksize = (mask_size.height - y) * 2;
    }
    // call openCV gaussian kernel generator
    double sigma = -1;
    Mat kernelX = getGaussianKernel(ksize, sigma, CV_32F);
    Mat kernelY = getGaussianKernel(ksize, sigma, CV_32F);
    // create 2d gaus
    Mat kernel = kernelX * kernelY.t();
    // create empty mask
    Mat mask = Mat::zeros(mask_size, CV_32F);
    Mat maski = Mat::zeros(mask_size, CV_32F);
```

```
    // copy kernel to mask on x,y
    Mat pos(mask, Rect(x - ksize / 2, y - ksize / 2, ksize, ksize));
    kernel.copyTo(pos);

    // create mirrored mask
    Mat posi(maski, Rect(( mask_size.width - x) - ksize / 2,
     (mask_size.height - y) - ksize / 2, ksize, ksize));
    kernel.copyTo(posi);
    // add mirrored to mask
    add(mask, maski, mask);

    // transform mask to range 0..1
    if(normalization) {
        normalize(mask, mask, 0, 1, NORM_MINMAX);
    }

    // invert mask
    if(invert) {
        mask = Mat::ones(mask.size(), CV_32F) - mask;
    }
    return mask;
}

void filter_noise(Mat complex, string title, int x, int y, int
kernel_size)
{
    Mat mask = createGausFilterMask(complex.size(), x, y, kernel_size,
     true, true);
    // show the kernel
    imshow(title+" gaus-mask", mask);

    shift(mask);

    Mat planes[] = {Mat::zeros(complex.size(), CV_32F),
    Mat::zeros(complex.size(), CV_32F)};
    Mat kernel_spec;
    planes[0] = mask; // real
    planes[1] = mask; // imaginar
    merge(planes, 2, kernel_spec);

    mulSpectrums(complex, kernel_spec, complex, DFT_ROWS);
    updateMag(complex, title);
    updateResult(complex, title);
}

int main()
{
     //------read image file----------------
     string filename = "cameraman.jpg";
     Mat original = imread(filename,0);
     imshow(filename+" original", original);

     Mat pnImage;
     Add_periodic_noise(original, pnImage, 0.75, 3, 1,
      0.55, PERIODIC_NOISE_HORIZONTAL);
     imshow(filename+" periodic noise", pnImage);
```

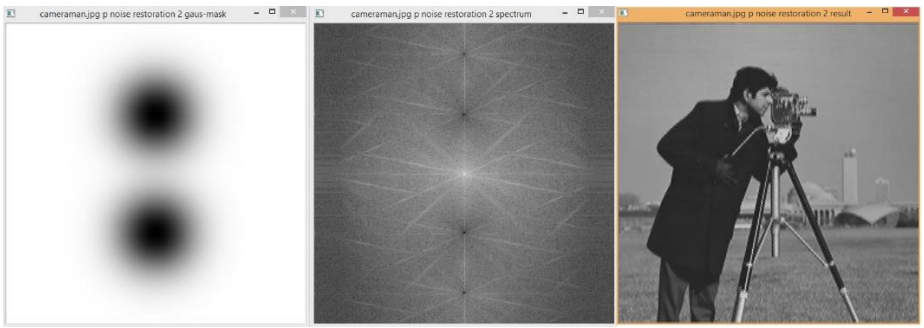
```
    //-------Transform to frequency domain--------------
    Mat complex = computeDFT(pnImage);
    //-------Apply filter to periodic noise image-------
    int kernel_size = 512;
    filter_noise(complex, filename+" p noise restoration 1", 256,
     52, kernel_size);
    filter_noise(complex, filename+" p noise restoration 2", 256,
     155, kernel_size);

    //================================================================
    waitKey(0);
    return 0;
}
```

Restorasi yang digunakan pada latihan ini ialah *gaussian filter* pada domain frekuensi dengan cara menutup letak *noise*. Hasil menjalankan program diatas dapat dilihat pada Gambar 10, Gambar 11, dan Gambar 12.

Gambar 10 Citra hasil penambahan *periodic noise*



Gambar 11 Citra hasil restorasi *periodic noise* (*filtering* 1)



Gambar 12 Citra hasil restorasi *periodic noise* (hasil akhir)

Nama :
NRP :
Nama Dosen :
Nama Asisten :

C TUGAS

Simpan kode program fungsi *outlier method* beserta *screenshot* semua citra hasil restorasi. *File* disimpan dengan format LKP9_NIM_Kelas dalam file .pdf.

Domain spasial

1 *Download* citra *salt and pepper noise*, *gaussian noise*, dan *speckle noise*.jpg pada LMS
2 Lakukanlah restorasi citra pada domain spasial menggunakan ***mean filter***, dan ***median filter***. **(Boleh menggunakan fungsi OpenCV).**
   - Restorasi citra boleh dilakukan beberapa kali secara berulang menggunakan *filter* dan ukuran yang sama hingga hasilnya bagus.
   - Tuliskan **jumlah *filtering*** dan **ukuran *filter*** yang digunakan.
   - *Screenshot* yang dilampirkan **setiap tahap** *filtering* dilakukan.
   - Lampirkan kode program bagian main saja.
3 Buatlah sebuah fungsi outlier_method **tanpa menggunakan fungsi OpenCV** pada citra *salt and pepper noise* (Fungsi OpenCV yang diperbolehkan: imread, waitKey, imshow, dan membaca citra *grayscale*).
   Dengan algoritma sebagai berikut:
   - Tentukan nilai D sebagai nilai *threshold*
   - Bandingkan tiap nilai piksel **p** dengan nilai rata-rata **m** dari 8 piksel tetangganya
   - Setiap piksel dilakukan pengecekan dengan kondisi
     if (|p – m| > D) **then** p = m
   - Lampirkan kode program fungsi outlier_method beserta *screenshot* hasil restorasi
   - Jelaskan secara singkat apakah algoritma *outlier method* lebih baik digunakan dibandingkan *mean filter* dan *median filter* untuk melakukan restorasi citra *salt and pepper noise*?

Domain frekuensi

1 *Download* citra periodic noise.jpg pada LMS
2 Lakukan restorasi citra pada periodic noise.jpg di domain frekeunsi menggunakan *gaussian filter* **(Boleh menggunakan fungsi OpenCV)**
3 Lampirkan kode program bagian main saja