PRAKTIKUM 7

Fourier Transform

Materi:

- Image Transformation
- Fourier Transformation 1D
- Discrete Fourier Transform 2D

Tujuan Praktikum:

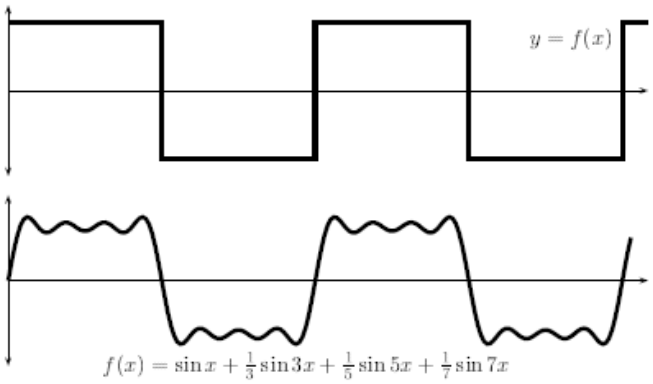
- Mahasiswa dapat melakukan transformasi pada citra menggunakan fourier trasnform
- Mahasiswa dapat melakukan filtering pada domain frekuensi

A. PENYAJIAN

Teori Dasar Fourier

Transformasi citra memegang peranan penting dalam pengolahan citra digital. Pada dasarnya transformasi bertujuan untuk mengubah domain spasial atau domain waktu ke dalam ruang frekuensi. Transformasi citra dapat digunakan untuk *image filtering*, *image restoration* dan *image compression* (*image encoding)*. Transformasi citra juga dapat memungkinkan pemisahan sinyal dengan noise dan frekuensi dapat menjelaskan karakteristik *periodic motion*.

Semua hal yang ada di muka bumi ini dapat digambarkan secara virtual menggunakan gelombang sinyal (*waveform*). Waveform merupakan fungsi waktu, fungsi ruang, atau fungsi dengan variable lain sesuai konteksnya. Sebagai contoh gelombang suara, elektromagnetik, data time series dan citra dapat direpresentasikan menggunakan sinyal. Pada dasarnya, serumit apapun gelombang yang dihasilkan ternyata dibentuk oleh penjumlahan nilai sinusoid pada frekuensi yang berbeda-beda. Contohnya dapat dilihat pada Gambar 1.



Gambar 1 Gelombang dan aproksimasi trigonometriknya

Fourier Transform pada 1D

Transformasi fourier dapat dilakukan pada data 1D maupun 2D. Transformasi fourier berlaku untuk sinyal kontinyu dan sinyal diskret. Fungsi transform terdiri dari faktor skala dan fungsi basis. Fungsi basis berfungsi seperti kernel pada konvolusi, dalam hal fourier fungsi kernelnya adalah cos dan sin. Berikut merupakan fungsi transformasi fourier untuk 1D:

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi ux} dx$$

Fungsi Basis

Faktor Skala

Note: $e^{ik} = \cos k + i \sin k \qquad i = \sqrt{-1}$

$$e^{\pm ix} = \cos(x) \pm i \sin(x)$$

$$F(u) = \int_{\infty}^{-\infty} f(x)(\cos 2\pi ux - i \sin 2\pi ux)\, dx$$

Fungsi kontinyu Fourier Transform 1D

$$F(k) = \left\langle f(x), b_k(x) \right\rangle = \sum_{x=0}^{N-1} f(x) e^{\frac{-2\pi ikx}{N}}$$

k = 0, 1, 2, ..., N-1

Fungsi diskret Fourier Transform 1D

Persamaan tersebut mengubah data dalam domain spasial f(x) kedalam domain frekuensi f(u). Mengubah data dari domain spasial kedalam domain frekuensi disebut dengan **transformasi Fourier**.

Sedangkan persamaan untuk mengembalikan data dari domain frekuensi kedalam fungsi spasial disebut dengan **Inverse Fourier Transform** yang dinyatakan dengan,

$$f(x) = \int_{\infty}^{\infty} F(u) e^{i2\pi ux} du = \int_{\infty}^{\infty} F(u)\{\cos(2\pi ux) + i \sin(2\pi ux)\} du$$

Fungsi Kontinyu Inverse Fourier Transform 1D

$$f(x) = \frac{1}{N} \sum_{k=0}^{N-1} F(k)\, e^{\frac{2\pi ikx}{N}} \qquad x = 0, 1, 2, \ldots, N-1$$

Fungsi Diskret Inverse Fourier Transform 1D

Fourier Transform pada 2D

Transformasi juga dapat dilakukan pada kasus 2D seperti citra. Fungsi yang digunakan untuk mentransformasi citra adalah *Discrete Fourier Transform* (DFT) 2D. Transformasi Fourier Diskrit (DFT) 2 Dimensi adalah tranformasi fourier diskrit yang dikenakan pada fungsi 2D (fungsi dengan dua variabel bebas), yang didefinisikan sebagai berikut :

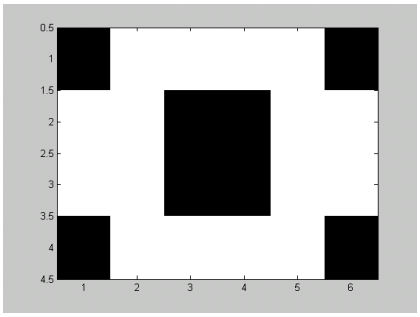
$$F(k_1,k_2) = \sum_{n_1=0}^{N_1} \sum_{n_2=0}^{N_2} f(n_1,n_2).e^{-j2\pi T(k_1 n_1/N_1 + k_2 n_2/N_2)}$$

contoh :

Diketahui f(x,y) adalah sebagai berikut :

0	1	1	1	1	0
1	1	0	0	1	1
1	1	0	0	1	1
0	1	1	1	1	0

Bila digambarkan hasilnya adalah sebagai berikut :



Gambar 2 Contoh citra dalam *f(x, y)*

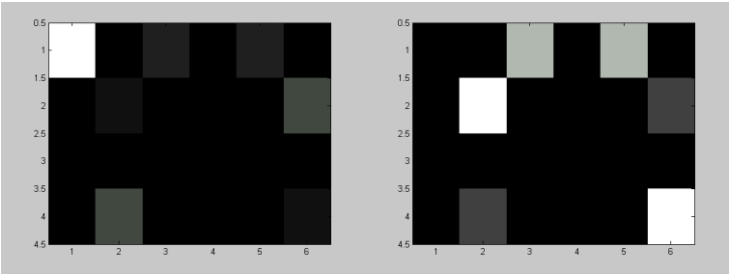
DFT dari fungsi f(x,y) di atas adalah :

$$F(k_1,k_2) = \sum_{n_1=0}^{4} \sum_{n_2=0}^{6} f(n_1,n_2).e^{-j2\pi T(k_1 n_1/4 + k_2 n_2/6)}$$

Hasil dari DFT adalah sebagai berikut :

16	0	-2 - 3.46i	0	-2 + 3.46i	0
0	-1.27 - 4.73i	0	0	0	4.73 - 1.27i
0	0	0	0	0	0
0	-4.73+ 1.27i	0	0	0	1.27 + 4.73i

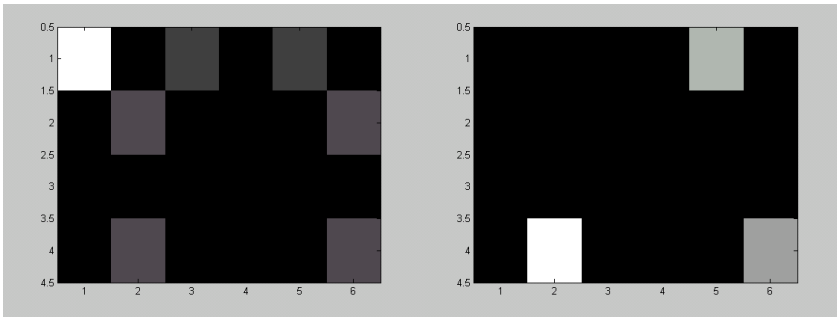
Secara Grafis dapat ditunjukkan bahwa :



Bagian Real Bagian Imaginer

Gambar 3 Contoh hasil DFT 2D

Hasil DFT dalam bentuk magnitude dan phase adalah sebagai berikut :

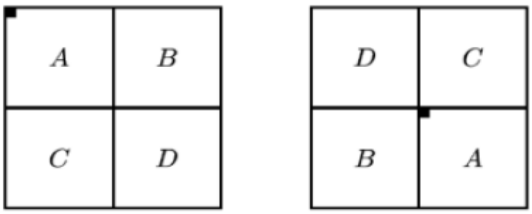


Magnitude Phase

Gambar 4 Contoh hasil DFT 2D dalam magnitude dan phase

Magnitude (frekuensi spektrum) sinusoid menunjukkan kontras citra selisih antara warna paling gelap dan warna paling terang. *Phase* menunjukkan bagaimana gelombang sinyal digeser terhadap titik awal.

Ilustrasi shifting pada DFT:



An FFT After shifting

B. LATIHAN

Membuat program yang dapat melakukan filtering pada domain frekuensi hasil discrete fourier transform pada citra menggunakan fungsi OpenCV.

butterworth.cpp

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

#define BUTTERWORTH_LPF 1
```

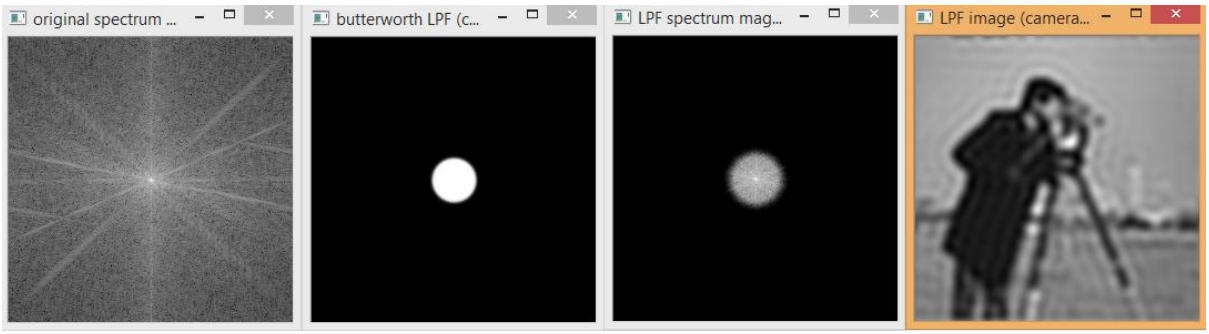
```
#define BUTTERWORTH_HPF 2

using namespace cv;

void shiftDFT(Mat& fImage )
{
    Mat tmp, q0, q1, q2, q3;

    // first crop the image, if it has an odd number of rows or columns
    fImage = fImage(Rect(0, 0, fImage.cols & -2, fImage.rows & -2));
    int cx = fImage.cols/2;
    int cy = fImage.rows/2;
    // rearrange the quadrants of Fourier image
    // so that the origin is at the image center
    q0 = fImage(Rect(0, 0, cx, cy));
    q1 = fImage(Rect(cx, 0, cx, cy));
    q2 = fImage(Rect(0, cy, cx, cy));
    q3 = fImage(Rect(cx, cy, cx, cy));

    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);

    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);
}

Mat create_spectrum_magnitude_display(Mat& complexImg, bool rearrange)
{
    Mat planes[2];
    // compute magnitude spectrum (N.B. for display)
    // compute log(1 + sqrt(Re(DFT(img))**2 + Im(DFT(img))**2))
    split(complexImg, planes);
    magnitude(planes[0], planes[1], planes[0]);

    Mat mag = (planes[0]).clone();
    mag += Scalar::all(1);
    log(mag, mag);

    if (rearrange)
    {
        // re-arrange the quaderants
        shiftDFT(mag);
    }

    normalize(mag, mag, 0, 1, CV_MINMAX);
    return mag;
}

Mat dft_create_spectrum_magnitude(Mat imgGray)
```

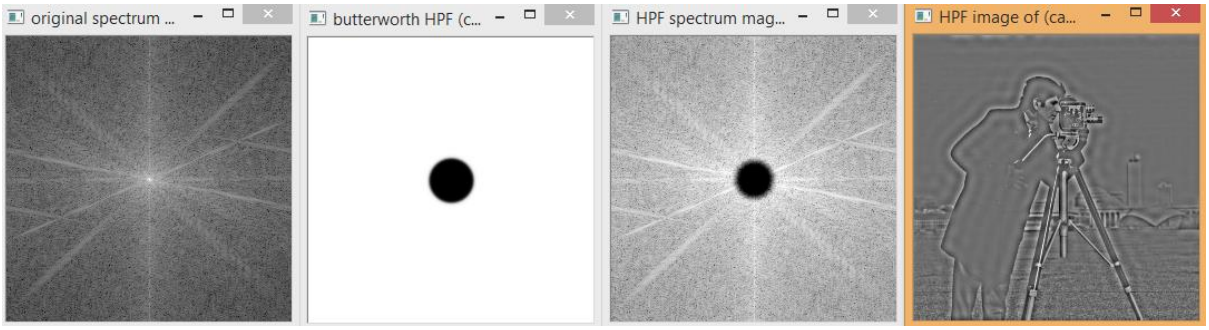
```
{
    Mat padded, complexImg;
    Mat planes[2];
    Mat dst;

    int N, M; // fourier image sizes

    // setup the DFT image sizes
    M = getOptimalDFTSize( imgGray.rows );
    N = getOptimalDFTSize( imgGray.cols );
    // setup the DFT images
    copyMakeBorder(imgGray, padded, 0, M - imgGray.rows, 0, N - imgGray.cols,
BORDER_CONSTANT, Scalar::all(0));
    planes[0] = Mat_<float>(padded);
    planes[1] = Mat::zeros(padded.size(), CV_32F);
    merge(planes, 2, complexImg);

    dft(complexImg, complexImg);// do the DFT
    dst = create_spectrum_magnitude_display(complexImg, true);// create spectrum
    magnitude

    return dst;
}

void create_butterworth_lowpass_filter(Mat &dft_Filter, int D, int n)
{
    Mat tmp = Mat(dft_Filter.rows, dft_Filter.cols, CV_32F);
    Point centre = Point(dft_Filter.rows / 2, dft_Filter.cols / 2);
    double radius;
    // based on the forumla in the IP notes (p. 130 of 2009/10 version)
    // see also HIPR2 on-line
    for(int i = 0; i < dft_Filter.rows; i++)
    {
        for(int j = 0; j < dft_Filter.cols; j++)
        {
            radius = (double) sqrt(pow((i - centre.x), 2.0) + pow((double) (j -
centre.y), 2.0));
            tmp.at<float>(i,j) = (float) ( 1 / (1 + pow((double) (radius /  D),
(double) (2 * n))));
        }
    }
    Mat toMerge[] = {tmp, tmp};
    merge(toMerge, 2, dft_Filter);
}

void create_butterworth_highpass_filter(Mat &dft_Filter, int D, int n)
{
    Mat tmp = Mat(dft_Filter.rows, dft_Filter.cols, CV_32F);
    Point centre = Point(dft_Filter.rows / 2, dft_Filter.cols / 2);
    double radius;
    // based on the forumla in the IP notes (p. 130 of 2009/10 version)
```

```
    // see also HIPR2 on-line
    for(int i = 0; i < dft_Filter.rows; i++)
    {
        for(int j = 0; j < dft_Filter.cols; j++)
        {
            radius = (double) sqrt(pow((i - centre.x), 2.0) + pow((double) (j -
centre.y), 2.0));
            tmp.at<float>(i,j) = (float) (1/(1+pow((double) (D /  radius),
(double) (2 * n))));
        }
    }
    Mat toMerge[] = {tmp, tmp};
    merge(toMerge, 2, dft_Filter);
}

void butterworth_filter(Mat imgGray, string title, int mode, int radius, int
order)
{
    Mat filtered_img;
    Mat padded, complexImg, filter, butterworth_filt;
    Mat planes[2], filtered_img_mag;

    int N, M; // fourier image sizes

    // setup the DFT image sizes
    M = getOptimalDFTSize( imgGray.rows );
    N = getOptimalDFTSize( imgGray.cols );

    // setup the DFT images
    copyMakeBorder(imgGray, padded, 0, M - imgGray.rows, 0, N - imgGray.cols,
BORDER_CONSTANT, Scalar::all(0));
    planes[0] = Mat_<float>(padded);
    planes[1] = Mat::zeros(padded.size(), CV_32F);
    merge(planes, 2, complexImg);

    dft(complexImg, complexImg);// do the DFT

    // construct the  filter (same size as complex image)

    filter = Mat(complexImg.rows, complexImg.cols, CV_32F, double(0));
    Mat complexFiltered = complexImg.clone();//clone to apply filter

    //default windows name
    string filter_name = "butterworth LPF ("+title+")";
    string filtered_spectrum_magnitude = "LPF spectrum magnitude ("+title+")";
    string filtered_image = "LPF image ("+title+")";

    if(mode == BUTTERWORTH_LPF) create_butterworth_lowpass_filter(filter, radius,
order);
    else if (mode == BUTTERWORTH_HPF)
    {
```

```
        create_butterworth_highpass_filter(filter, radius, order);
        filter_name = "butterworth HPF ("+title+")";
        filtered_spectrum_magnitude = "HPF spectrum magnitude ("+title+")";
        filtered_image = "HPF image of ("+title+")";
    }
    //==========Apply butterworth filter=============//
    shiftDFT(complexFiltered);
    mulSpectrums(complexFiltered, filter, complexFiltered, 0);
    shiftDFT(complexFiltered);
    filtered_img_mag = create_spectrum_magnitude_display(complexFiltered,
    true);// create spectrum magnitude spectrum for display
    idft(complexFiltered, complexFiltered); // do inverse DFT on filtered image

    // split into planes and extract plane 0 as output image
    split(complexFiltered, planes);
    normalize(planes[0], filtered_img, 0, 1, CV_MINMAX);

    // do the same with the filter image
    split(filter, planes);
    normalize(planes[0], butterworth_filt, 0, 1, CV_MINMAX);

    //=======display image in window==================//
    //filtered image
    imshow(filtered_spectrum_magnitude, filtered_img_mag);
    imshow(filtered_image, filtered_img);
    imshow(filter_name, butterworth_filt);
}

int main()
{
    Mat original = imread("cameraman.png",0);
    Mat dst = dft_create_spectrum_magnitude(original);

    imshow("original spectrum magnitude",dst);
    butterworth_filter(original, "cameraman 1", BUTTERWORTH_HPF,20,15);
    butterworth_filter(original, "cameraman 2", BUTTERWORTH_LPF,20,15);

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

Output butterworth.cpp




original spectrum ...
butterworth HPF (c...
HPF spectrum mag...
HPF image of (ca...

Nama :

NRP :

Nama Dosen :

Nama Asisten :

C. Lembar Kerja Praktikum

Simpan kode program beserta screenshot citra spectrum magnitude original, spectrum magnitude hasil filtering, dan filter yang digunakan. **File disimpan dengan format LKP7_NIM_Kelas dalam file .pdf.**

1. Download citra cameraman.png pada LMS
2. Baca citra cameraman.png dalam format grayscale
3. Lakukanlah butterwortfiltering low pass dan high pass pada citra cameraman.png dengan ketentuan sebagai berikut:
   a. Percobaan terhadap nilai radius (low pass dan high pass)
      - Radius = 5, order = 8
      - Radius = 10, order = 8
      - Radius = 20, order = 8
   b. Percobaan terhadap nilai order (low pass dan high pass)
      - Radius = 10, order = 4
      - Radius = 10, order = 8
      - Radius = 10, order = 16
4. Bandingkan hasil citra yang diperoleh! Bagaimana pengaruh nilai radius dan order terhadap output citra!
5. Boleh menggunakan fungsi OpenCV

D. Daftar pustaka

http://www.thefouriertransform.com/#introduction

http://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm

https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/

[Alasdair McAndrew] An Introduction to Digital Image (BookZZ.org)