SOUTH UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA

IMAGE AND VIDEO PROCESSING

CLASS PROJECT 3

# Fourier Transform

Author:
Jie YANG

Supervisor:
JIANHONG SHI

October 16, 2018

# CONTENTS

# Fourier Transform

## Introduction

In most common conditions, the talk about the image in space domain, which indicate the intensity distribution of a image. However, an image can be transformed to frequency domain by Fourier transform. Because image is discrete matrix, it is discrete Fourier transform (DFT).

In space domain, the matrix indicate power distribution in each frequency component. So that we can process image in frequency domain which can be very easy to do filtering. This powerful method nowadays applied broadly in lots of fields.

In this experiment, we will write DFT function to implement 2D DFT for all provided images and analyze their transformed images in frequency domain. We will also reconstruct lena.pgm using the magnitude and phase images in frequency domain respectively and analyze the results.

## Method

For each image, we can calculate its frequency component by **Eq.1**, $M, N$ is the length and width as image, respectively. This progress is DFT. After that, we can do reverse transform to reconstruct image,**Eq.1** which transform image from frequency domain to spatial domain.

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/M+vy/N)} \tag{1}$$

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(ux/M+vy/N)} \tag{2}$$

In frequency domain, the low frequency component at the corner while high frequency at the center. To make it clear, we can shift original point to the image center, so that from center to edge, the frequency gradually decrease. This shift step can be done base on **Eq.3**. by multiply $(-1)^{(x+y)}$ in frequency domain or spatial domain before DFT.

$$\begin{aligned} f(x,y)(-1)^{x+y} &\Leftrightarrow F(u-M/2, v-N/2) \\ f(x-M/2, y-N/2) &\Leftrightarrow F(u,v)(-1)^{u+v} \end{aligned} \tag{3}$$

The full steps in this lab is indicated in **Eq.4**. First, we multiply image with $(-1)^{(x+y)}$ , then do DFT to transform image to frequency domain. For reverse DFT (IDFT), we can do DFT to frequency image then get its real part, finally multiply it with $(-1)^{(x+y)}$ to reconstruct image.

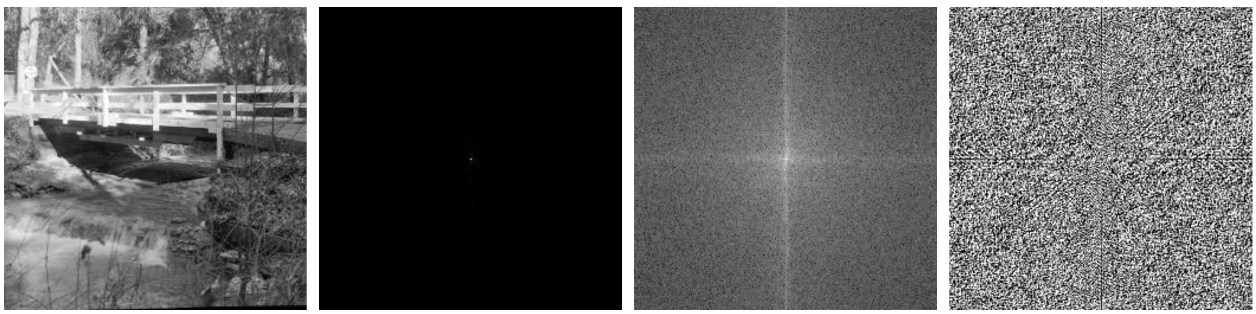
$$f(x,y) \xrightarrow{*(-1)^{x+y}} g(x,y) \xrightarrow{\mathfrak{F}} F(u,v) \xrightarrow{\mathfrak{F}} f(u,v) \xrightarrow{\mathfrak{Re}} \mathfrak{Re}(f(u,v)) \xrightarrow{*(-1)^{x+y}} f(x,y) \tag{4}$$

## Results

# DFT

We find that after DFT, the magnitude is two weak to see, so I choose to plot the log form of magnitude.

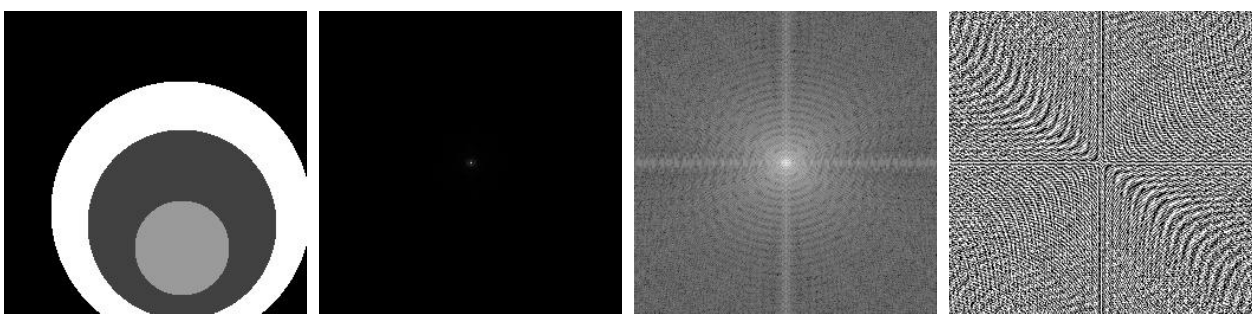
From bridge image, the main is at low frequency part, it has bright line on vertical and horizontal axis. The vertical one is brighter that horizontal one. Because the brighter part in image is the bridge while it is on horizontal axis.**Fig.1**. The phase image seems to be no order.



(a) original (b) magnitude (c) log magnitude (d) phase

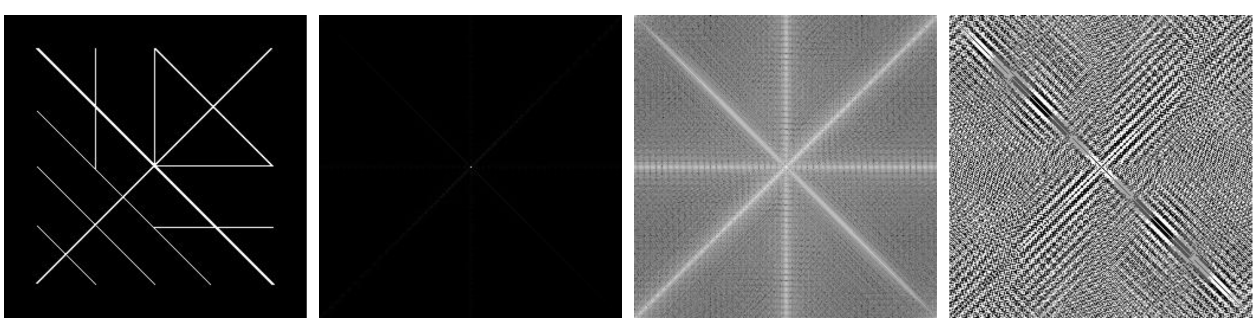
Figure 1: Fourier transform of bridge image

For circles image,**Fig.2** there are several circle line around center and phase image also be symmetrical



(a) original (b) magnitude (c) log magnitude (d) phase

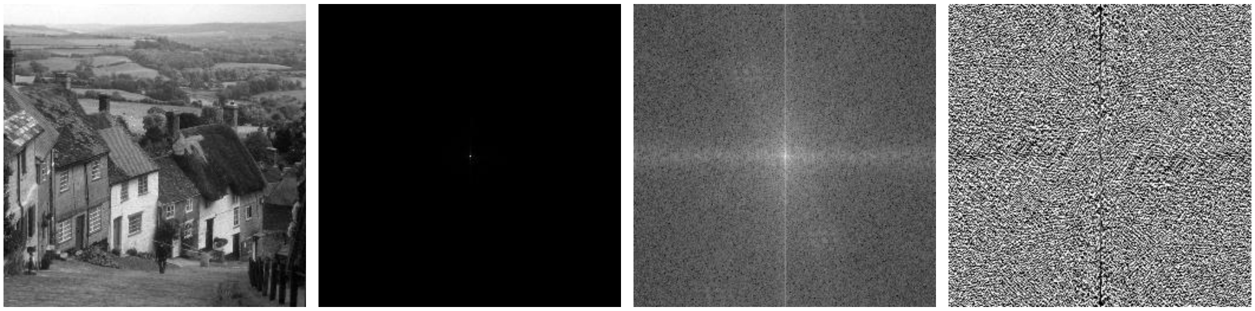
Figure 2: Fourier transform of circles image



(a) original (b) magnitude (c) log magnitude (d) phase

Figure 3: Fourier transform of crosses image

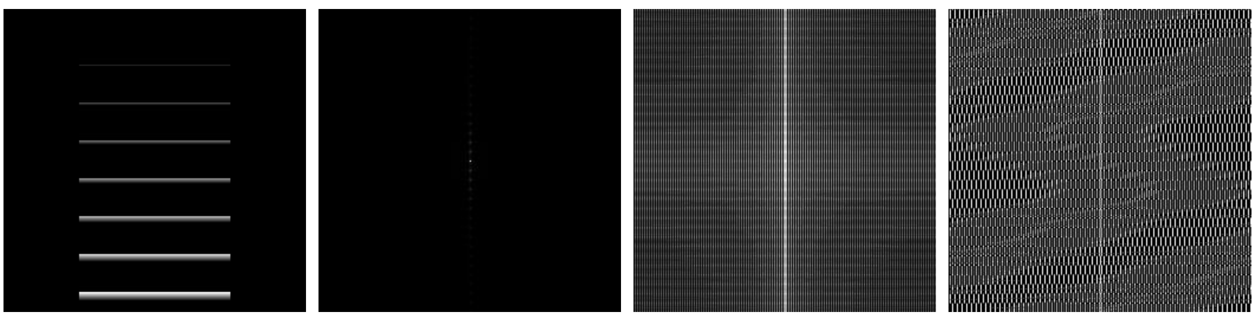
For crosses image,**Fig.3** there are several diagonal vertical and horizontal lines, so that we can see for lines in magnitude image, the phase image also be ordered.



(a) original (b) magnitude (c) log magnitude (d) phase

Figure 4: Fourier transform of goldhill image

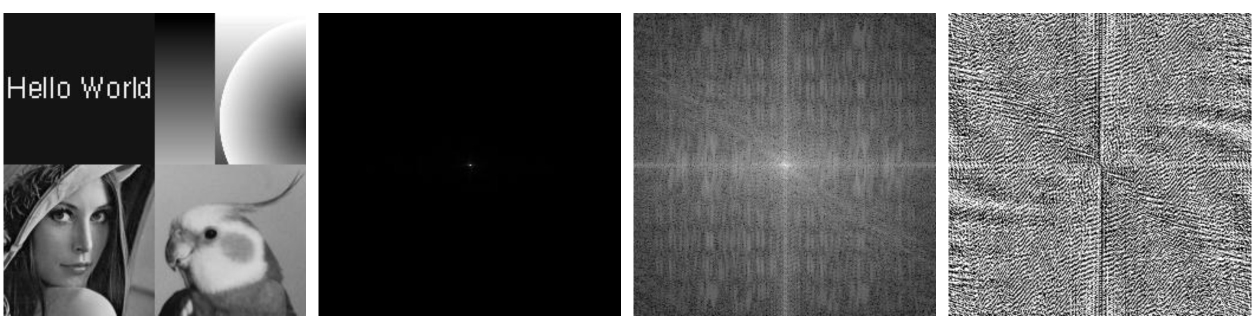
For goldhill image,**Fig.4**, it has house with white walls, which induce bright vertical line and closer horizontal areal in magnitude image, in phase image, we can see vertical line.



(a) original (b) magnitude (c) log magnitude (d) phase

Figure 5: Fourier transform of horiz image

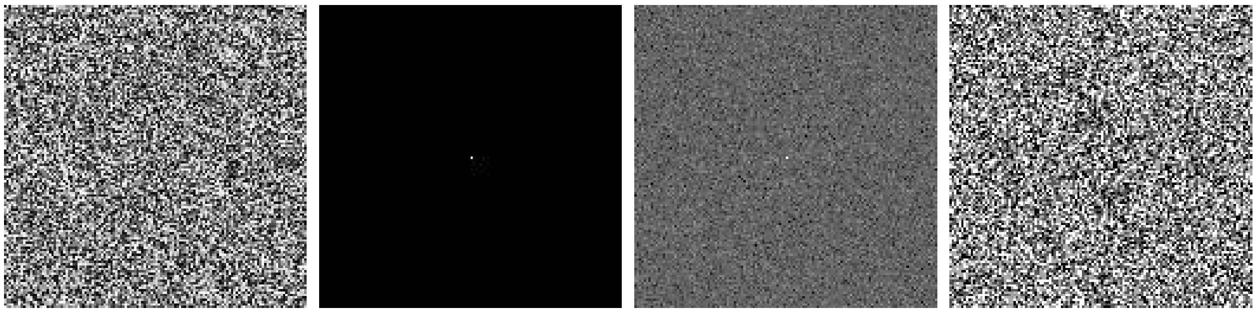
For horiz image,**Fig.5**, it has several vertical lines in image, so that we can see a bright vertical line in magnitude image and the phase image be very ordered.




(a) original (b) magnitude (c) log magnitude (d) phase

Figure 6: Fourier transform of montage image

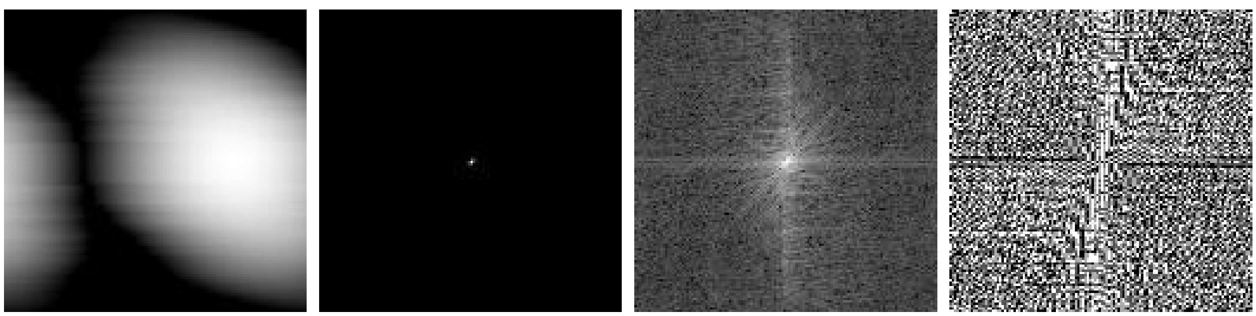
For montage image **Fig.6**, it be divided into four parts, however, the magnitude image is not the combine of part parts individually, each part has similar magnitude image. The phase image also be divided into four parts.



(a) original (b) magnitude (c) log magnitude (d) phase

Figure 7: Fourier transform of noise image

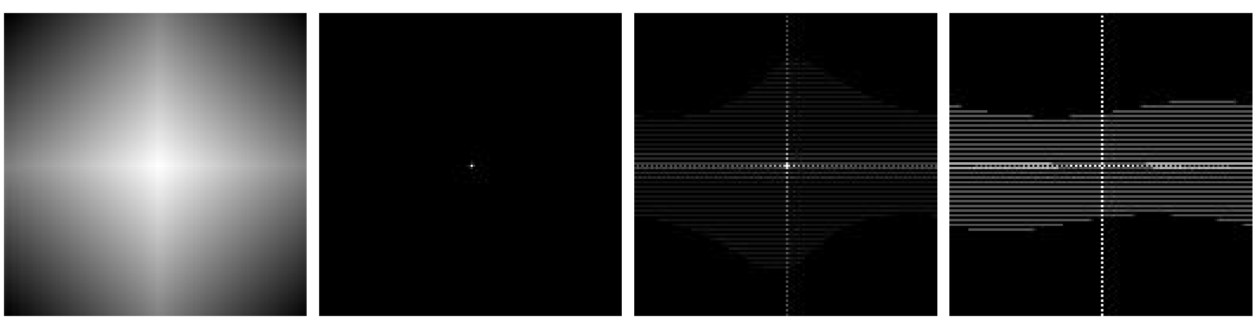
For noise image **Fig.7**, the magnitude image is smoother than original image while most power concentrate in low frequency part. Its phase image is orderless like original image.



(a) original (b) magnitude (c) log magnitude (d) phase

Figure 8: Fourier transform of rampe128 image

For rampe128 **Fig.8**, the main bright part is vertical or horizontal, so that there are two lines in magnitude image. For phase image, the high angle concentrate in horizontal axis.

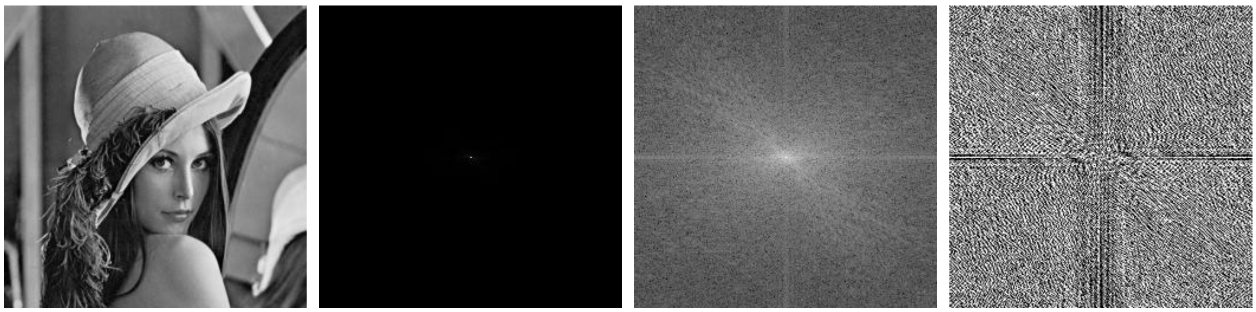


(a) original (b) magnitude (c) log magnitude (d) phase

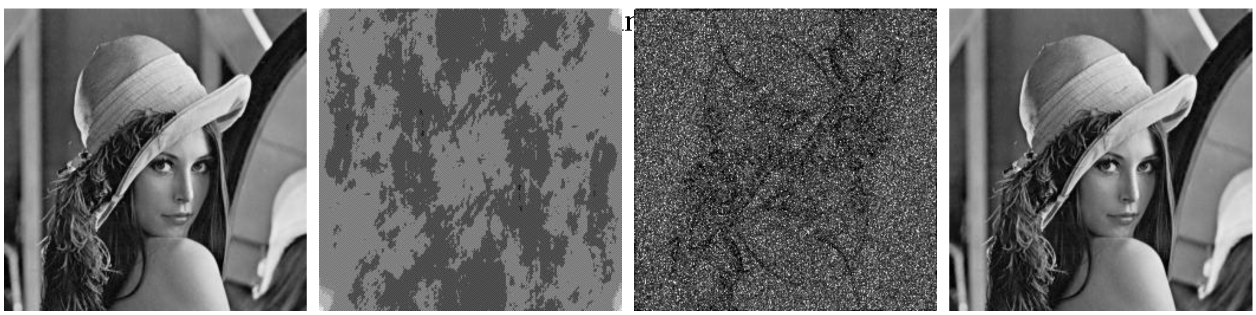
Figure 9: Fourier transform of rampr128 image

## Image reconstruction

We choose lena image to do reconstruction in this experiment. From magnitude image, main power are in low frequency and two axis in magnitude and phase image.**Fig.11**.



(a) original (b) magnitude (c) log magnitude (d) phase



(a) original (b) magnitude (c) log phase (d) magnitude & phase

Figure 11: Reconstructed lena image

When only use magnitude image to do reconstruction, we can not see the lena in reconstructed image. However, when only use phase to reconstruct, after log enhancement and hist equalization, we can see the outline of lena. When we use magnitude and phase information together to do reconstruction, we can reconstruct image perfectly.**Fig.11**

# Discussion

In this experiment, we can find that when the magnitude image contain the intensity information of the image, when we rotate image, its magnitude image will also rote but the phase dose not. The phase image contain the phase information of each frequency component.

During reconstruction step, we find that, phase image has for information that magnitude image, because we can see outline of image through phase image that just use magnitude image. The reason is that, if we use phase image only, we suppose that all intensity of each frequency component to be same, that likely to weight each component at each pixel, however, low frequency has higher amplitude while high frequency has low amplitude. This weight will increase high frequency component, for edge pixel, it most composed of high frequency, so we can see the outline of image in reconstruct image only with phase.

When we only use magnitude image to reconstruct image, it is same to shift all phase to zero for each pixel, because the phase of lena image of each pixel is not same, this shift let pixel value to be random, so we can not see image in reconstruct image.

The reconstruct step indicate that phase image contains more information than magnitude image.

# Supplementary

This is the code used in this project.

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <string>
#define _USE_MATH_DEFINES
#include <complex.h>
#include<math.h>
#include "PUABLIC.h"


using namespace cv;
using namespace std;


Mat Hiseq(Mat ima){
  Mat img = Mat::zeros(ima.rows, ima.cols, CV_8UC1);
  ima.convertTo(img, CV_8UC1, 1, 0);
  //cout <<'\n'<< img << endl;

  Mat IM = Mat::zeros(img.rows, img.cols, CV_8UC1);

  double minv = 0.0, maxv = 0.0;
  double* minp = &minv;
  double* maxp = &maxv;
  minMaxIdx(img, minp, maxp);
  //cout << *minp << endl;
  //cout << *maxp << endl;
  int L = *maxp - *minp + 1;
  IM = (img - *minp)*255/L;




  return IM;


}

Mat getlayer(Mat src, int k){
  double M = src.rows;
```

```
    double N = src.cols;
    Mat layer (M, N, CV_64FC1);
    for (int x = 0; x < src.rows; ++x){
      for (int y = 0; y < src.cols; ++y){
        layer.at<double>(x, y) = src.at<Vec4d>(x, y)[k];


      }
    }
    return layer;

}


Mat mydft2(Mat src){

    double M = src.rows;
    double N = src.cols;
    Mat DFTI(M, N, CV_64FC4);
    //Mat DFTI(M, N, CV_64FC2, Scalar_<double>(0.0, 0.0));
    //Mat img(3, 4, CV_64FC2, Scalar_<double>(12.625, 3.141592653));


    for (int x = 0; x < src.rows; ++x){
      for (int y = 0; y < src.cols; ++y){
        complex<double> temp = 0;
        for (int u = 0; u < src.rows; ++u){
          for (int v = 0; v < src.cols; ++v){
            //Complex a = Complex< float >::Complex(0, 2 * M_PI*(u*x / M + v*y / N));
            double aa = v*y/N;
            //cout << v << "  v\t" << y << "  y \t"<< aa << "  aa\t"<< endl;

            complex<double> a(0, -1*2 * M_PI*(u*x / M + v*y / N));
            //cout << "+++" << x << '\t' << u << '\t' << y<< '\t'<<v<<endl;
            //cout << "++++++++++" << 'u*x ' << u*x << 'v*y/N' << v*y / N<<(u*x / M + v*y / N) << '\t' << 2 * M_PI*(u*x / M + v*y / N) << '\t' << a << endl;

            double val= src.at<double>(u, v);

            temp = temp + val*pow((-1) ,(u+v))*exp(a);
          }
        }
        //cout << x << "\t" << y << "val\t" << temp << endl;

        //DFTI.at<std::complex<double> >(x, y) = temp;

        DFTI.at<Vec4d>(x, y)[0] = temp.real();
        DFTI.at<Vec4d>(x, y)[1] = temp.imag();
        DFTI.at<Vec4d>(x, y)[2] = abs(temp);
        DFTI.at<Vec4d>(x, y)[3] = arg(temp);
        //cout << DFTI << endl;
```

```
      }
    }




    return DFTI;



}



Mat myidft2(Mat reval, Mat imval){


    double M = reval.rows;
    double N = reval.cols;
    Mat IDFTI(M, N, CV_64FC4);

    for (int x = 0; x < reval.rows; ++x){
      for (int y = 0; y < reval.cols; ++y){
        complex<double> temp = 0;
        for (int u = 0; u < reval.rows; ++u){
          for (int v = 0; v < reval.cols; ++v){




            complex<double> a(0, 1 * 2 * M_PI*(u*x / M + v*y / N));
            //double reval = src.at<Vec4d>(x, y)[0];
            //double imval = src.at<Vec4d>(x, y)[1];
            double Re = reval.at<double>(u, v);
            double Im = imval.at<double>(u, v);


            complex<double> val(Re,Im);
            temp = temp + val*exp(a);
          }
        }

        temp = temp / (M*N)*pow(-1, (x + y));
        IDFTI.at<Vec4d>(x, y)[0] = temp.real();
        IDFTI.at<Vec4d>(x, y)[1] = temp.imag();
        IDFTI.at<Vec4d>(x, y)[2] = abs(temp);
        IDFTI.at<Vec4d>(x, y)[3] = arg(temp);


```

```
    }
  }




  return IDFTI;



}




string ImageNameList[] = { "bridge", "circles" ,"crosses","goldhill","horiz","lena...
    ","montage","noise","rampe128","rampr128"};
//string ImageNameList[] = { "test"};
ImgProp imgprop = {
  "D://graduated//Image_process//lab//PGM_images//",
  ".pgm" };
ImgProp imgsave = { "D://graduated//Image_process//lab//lab_report//lab4//...
    imagesave//",
".jpg" };

int len = sizeof(ImageNameList) / sizeof(ImageNameList[0]);// namelist length


void lab3_main()
{

  for (int i = 0; i < len;i++){
    imgprop.img_name = ImageNameList[i];
    imgsave.img_name = imgprop.img_name;
    string img_path = imgprop.img_fold + imgprop.img_name + imgprop.type;
    Mat ima = imread(img_path, 0); //read gray image
    imshow("original     " + imgprop.img_name, ima);

    imgsave.img = ima;

    imgsave.mark = "original";
    img_save(imgsave);

    Mat fimg;
    ima.convertTo(fimg, CV_64FC1, 1, 0);// no shift and scale
    Mat zoro = Mat::zeros(fimg.rows, fimg.cols, CV_64FC1);
    //*************     dft**************

    Mat DFT_img;

```

```cpp
        //cout << fimg << endl;
        DFT_img=mydft2(fimg);

        Mat magDFT_img = getlayer(DFT_img,2);
        Mat phaseDFT_img = getlayer(DFT_img,3);


        imgsave.img = Hiseq(magDFT_img);
        imgsave.mark = "DFT_mag";
        img_save(imgsave);

        imgsave.img = Hiseq(phaseDFT_img);
        imgsave.mark = "DFT_phase";
        img_save(imgsave);
        //*************    dft**************


        //*************    idft**************

        Mat IDFT_img_full = myidft2(getlayer(DFT_img, 0), getlayer(DFT_img, 1));
        Mat IDFT_img = getlayer(IDFT_img_full, 0);

        Mat magIDFT_img_full = myidft2(magDFT_img, zoro);
        Mat magIDFT_img = getlayer(magIDFT_img_full, 0);

        Mat phaseIDFT_img_full = myidft2(phaseDFT_img, zoro);
        Mat phaseIDFT_img = getlayer(phaseIDFT_img_full, 0);
        cout << IDFT_img << endl;
        imgsave.img = Hiseq(IDFT_img);
        cout << imgsave.img << endl;
        imgsave.mark = "IDFT_mag";

        imgsave.img = Hiseq(magIDFT_img);
        imgsave.mark = "IDFT_mag";
        img_save(imgsave);

        imgsave.img = Hiseq(phaseIDFT_img);
        imgsave.mark = "IDFT_phase";
        img_save(imgsave);



        //*************    idft**************


    }
}

}
```

C++ code for image processing