谭树杰 11849060

Lab5 Report

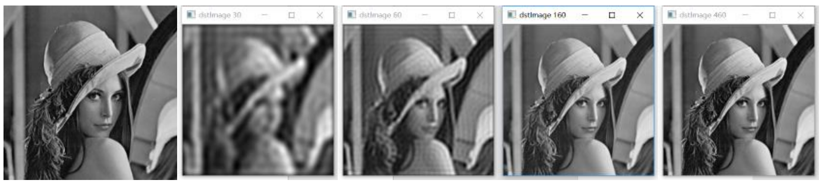
The main() function of this project is shown below:

```
int main(){
    // 1. Filter lena.pgm and bridge.pgm using IDLPF, BLPF, GLPF
    // and compare the image qualities with different cutoff frequencies
    /*
    // I just replace lena.pgm with bridge.pgm to process bridge.pgm using filters
    cv::Mat srcImage = imread("..\\images\\lena.pgm", CV_LOAD_IMAGE_GRAYSCALE);
    if (!srcImage.data)
    return -1;
    cv::Mat dstImage = IDLPF(srcImage, 30); // IDLPF
    cv::imshow("dstImage 30", dstImage);
    dstImage = IDLPF(srcImage, 60);
    cv::imshow("dstImage 60", dstImage);
    dstImage = IDLPF(srcImage, 160);
    cv::imshow("dstImage 160", dstImage);
    dstImage = IDLPF(srcImage, 460);
    cv::imshow("dstImage 460", dstImage);
    cv::Mat dstImage = BLPF(srcImage, 30, 2);   //BLPF
    cv::imshow("dstImage 30", dstImage);
    dstImage = BLPF(srcImage, 60, 2);
    cv::imshow("dstImage 60", dstImage);
    dstImage = BLPF(srcImage, 160, 2);
    cv::imshow("dstImage 160", dstImage);
    dstImage = BLPF(srcImage, 460, 2);
    cv::imshow("dstImage 460", dstImage);
    cv::Mat dstImage = GLPF(srcImage, 30);      // GLPF
    cv::imshow("dstImage 30", dstImage);
    dstImage = GLPF(srcImage, 60);
    cv::imshow("dstImage 60", dstImage);
    dstImage = GLPF(srcImage, 160);
    cv::imshow("dstImage 160", dstImage);
    dstImage = GLPF(srcImage, 460);
    cv::imshow("dstImage 460", dstImage);
    */
    // 2. Use HPF and thresholding to sharpen fingerprint1.pgm and
    // fingerprint2.pgm
    // replace fingerprint1 with fingerprint2 to get processing of fingerprint2.pgm
    cv::Mat srcImage = imread("..\\images\\fingerprint1.pgm", CV_LOAD_IMAGE_GRAYSCALE);
    if (!srcImage.data)
        return -1;
    cv::Mat dstImage = BHPF(srcImage, 50, 4);
    cv::imshow("fingerprint1", dstImage);
    cv::waitKey(0);
    return 0;
}
```

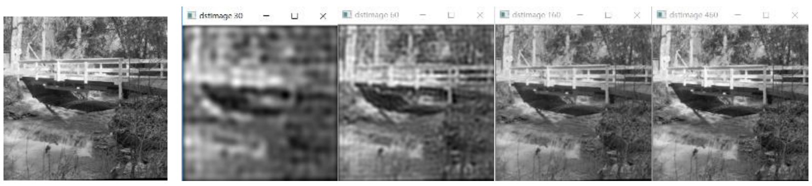
1. Filter lena.pgm and bridge.pgm using IDLPF, BLPF, GLPF and compare the image qualities with different cutoff frequencies

(1) filter lena.pgm and bridge.pgm using IDLPF

The lena.pgm and processed images using D0 = 30, 60, 160, 460 are shown below:



The bridge.pgm and processed images using D0 = 30, 60, 160, 460 are shown below:



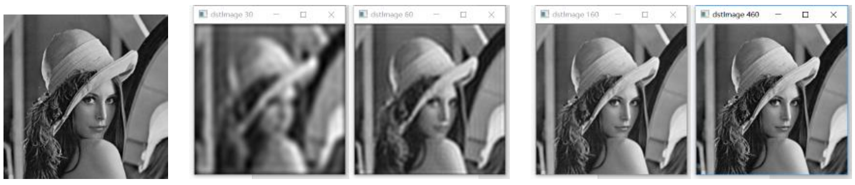
Analysis: From the images above, we can see the blurring and ringing through the increment of cutoff frequency D0. When D0 = 30, we can hardly recognize the objects in the image, for that we only reserve small low frequencies of the images. As D0 increasing, the images become more and more clear and the ringing reduces. Because more and more high frequency information is reserved.

The IDLPF is implemented in the following code:

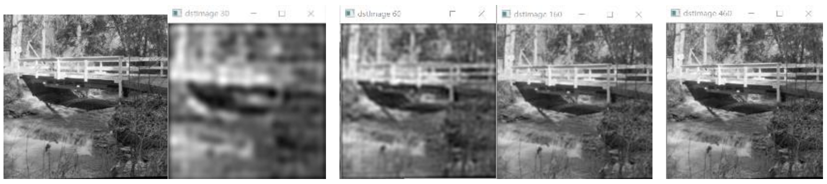
```
cv::Mat IDLPF(cv::Mat srcImage, int D0){
    CV_Assert(srcImage.data != NULL);

    int P = 2 * srcImage.rows;  // 1. obatain padding parameters P and Q
    int Q = 2 * srcImage.cols;
    cv::Mat dstImage;
    copyMakeBorder(srcImage, dstImage, 0, P - srcImage.rows,  // 2. form a padding
image
        0, Q - srcImage.cols, BORDER_CONSTANT, Scalar::all(0) );
    dstImage.convertTo(dstImage, CV_64F);
    for (int x = 0; x < srcImage.rows; x++){    // 3. center the image
        for (int y = 0; y < srcImage.cols; y++){
            dstImage.ptr<double>(x)[y] *= pow(-1, x + y);
        }
    }
    // 4. compute the DFT
    cv::Mat planes[] = { cv::Mat_<double>(dstImage), cv::Mat::zeros(dstImage.size(),
CV_64F) };
    cv::Mat dftMat;
    merge(planes, 2, dftMat);
    dft(dftMat, dftMat);

    cv::Mat ILMat(P, Q, CV_64F, Scalar(0));  // 5. calculate ideal lowpass filter
matrix and form the product
    for (int u = 0; u < P; u++){
        for (int v = 0; v < Q; v++){
            if ((u - P / 2) * (u - P / 2) + (v - Q / 2) * (v - Q / 2) <= D0 * D0){
                ILMat.ptr<double>(u)[v] = 1;
            }
        }
    }
    split(dftMat, planes);
    planes[0] = planes[0].mul(ILMat);
    planes[1] = planes[1].mul(ILMat);
    merge(planes, 2, dftMat);      // get G(u,v) now
    idft(dftMat, dftMat);                    //6. obtain the processed image
    split(dftMat, planes);
    dstImage = planes[0];            // select real part in order to ignore parasitic
    for (int x = 0; x < dstImage.rows; x++){
        for (int y = 0; y < dstImage.cols; y++){
            dstImage.ptr<double>(x)[y] *= pow(-1, x + y);
        }
    }
    Mat dst(dstImage, Rect(0, 0, P/2, Q/2));    //7. obtain the final result, extract
```

(2) filter lena.pgm and bridge.pgm using BLPF

The lena.pgm and processed images using N = 2 and D0 = 30, 60, 160, 460 are shown below:



The bridge.pgm and processed images using N = 2 and D0 = 30, 60, 160, 460 are shown below:



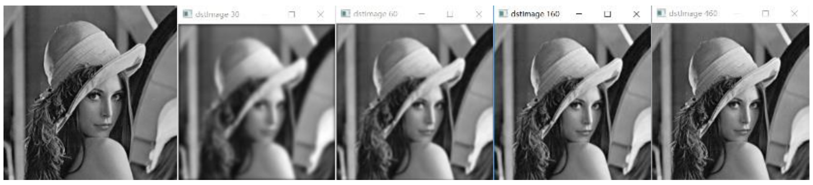
Analysis: Not as IDLPF, the ringing is not visible in any images above. Because BLPF is of smooth transition between low and high frequencies. But ringing can become significant in hihg order BLPF.

The BLPF is implemented in the following code:

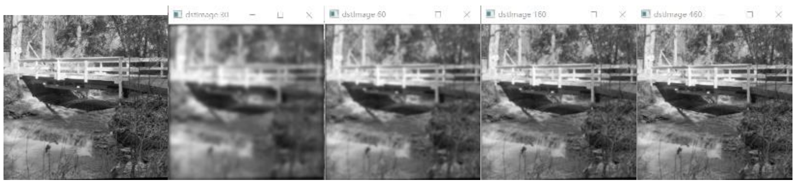
```
cv::Mat BLPF(cv::Mat srcImage, int D0, int N){
    CV_Assert(srcImage.data != NULL);

    int P = 2 * srcImage.rows;  // 1. obatain padding parameters P and Q
    int Q = 2 * srcImage.cols;
    cv::Mat dstImage;
    copyMakeBorder(srcImage, dstImage, 0, P - srcImage.rows,  // 2. form a padding
image
        0, Q - srcImage.cols, BORDER_CONSTANT, Scalar::all(0));
    dstImage.convertTo(dstImage, CV_64F);
    for (int x = 0; x < srcImage.rows; x++){    // 3. center the image
        for (int y = 0; y < srcImage.cols; y++){
            dstImage.ptr<double>(x)[y] *= pow(-1, x + y);
        }
    }

    // 4. compute the DFT
    cv::Mat planes[] = { cv::Mat_<double>(dstImage), cv::Mat::zeros(dstImage.size(),
CV_64F) };
    cv::Mat dftMat;
    merge(planes, 2, dftMat);
    dft(dftMat, dftMat);

    cv::Mat BwMat(P, Q, CV_64F, Scalar(0));  // 5. calculate Butterworth lowpass filter
matrix and form the product
    for (int u = 0; u < P; u++){
        for (int v = 0; v < Q; v++){
            double distSqure = (u - P / 2) * (u - P / 2) + (v - Q / 2) * (v - Q / 2);
                double tmp = pow(distSqure, N);
                BwMat.ptr<double>(u)[v] = 1 / (1 + tmp / pow(D0, 2 * N) );
        }
    }
    split(dftMat, planes);
    planes[0] = planes[0].mul(BwMat);
    planes[1] = planes[1].mul(BwMat);
    merge(planes, 2, dftMat);      // get G(u,v) now

    idft(dftMat, dftMat);                    //6. obtain the processed image
    split(dftMat, planes);
    dstImage = planes[0];            // select real part in order to ignore parasitic
    for (int x = 0; x < dstImage.rows; x++){
        for (int y = 0; y < dstImage.cols; y++){
            dstImage.ptr<double>(x)[y] *= pow(-1, x + y);
        }
```

(3) filter lena.pgm and bridge.pgm using GLPF

The lena.pgm and processed images using D0 = 30, 60, 160, 460 are shown below:



The bridge.pgm and processed images using D0 = 30, 60, 160, 460 are shown below:



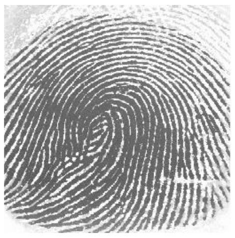
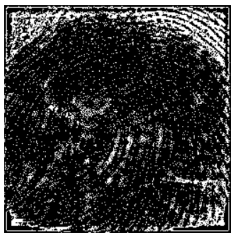
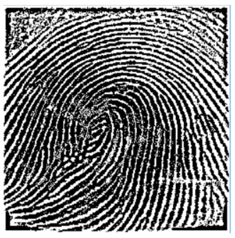
Analysis: We observe a smooth transition when cutoff frequency is increasing. The ringing is more invisible in any images above. Because the profile of GLPF is not as tight as the profile of the BLPF of order 2.

The GLPF is implemented in the following code:

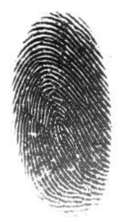
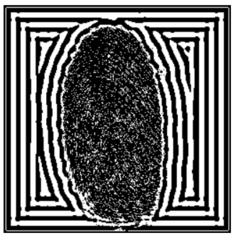
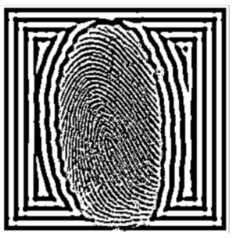
```
cv::Mat GLPF(cv::Mat srcImage, int D0){
    CV_Assert(srcImage.data != NULL);

    int P = 2 * srcImage.rows;  // 1. obatain padding parameters P and Q
    int Q = 2 * srcImage.cols;
    cv::Mat dstImage;
    copyMakeBorder(srcImage, dstImage, 0, P - srcImage.rows,  // 2. form a padding
image
        0, Q - srcImage.cols, BORDER_CONSTANT, Scalar::all(0));
    dstImage.convertTo(dstImage, CV_64F);
    for (int x = 0; x < srcImage.rows; x++){    // 3. center the image
        for (int y = 0; y < srcImage.cols; y++){
            dstImage.ptr<double>(x)[y] *= pow(-1, x + y);
        }
    }

    // 4. compute the DFT
    cv::Mat planes[] = { cv::Mat_<double>(dstImage), cv::Mat::zeros(dstImage.size(),
CV_64F) };
    cv::Mat dftMat;
    merge(planes, 2, dftMat);
    dft(dftMat, dftMat);

    cv::Mat GMat(P, Q, CV_64F, Scalar(0));  // 5. calculate Gaussian lowpass filter
matrix and form the product
    for (int u = 0; u < P; u++){
        for (int v = 0; v < Q; v++){
            double distSqure = (u - P / 2) * (u - P / 2) + (v - Q / 2) * (v - Q / 2);
            GMat.ptr<double>(u)[v] = exp(-distSqure / (2 * D0 * D0) );
        }
    }
    split(dftMat, planes);
    planes[0] = planes[0].mul(GMat);
    planes[1] = planes[1].mul(GMat);
    merge(planes, 2, dftMat);      // get G(u,v) now

    idft(dftMat, dftMat);                  //6. obtain the processed image
    split(dftMat, planes);
    dstImage = planes[0];            // select real part in order to ignore parasitic
    for (int x = 0; x < dstImage.rows; x++){
        for (int y = 0; y < dstImage.cols; y++){
            dstImage.ptr<double>(x)[y] *= pow(-1, x + y);
        }
    }
```

2. Use HPF and thresholding to sharpen fingerprint1.pgm and fingerprint2.pgm

The fingerprint1.pgm and its image after BHPF and thresholding are shown below

The fingerprint2.pgm and its image after BHPF and thresholding are shown below

Analysis: We can see the enhancement of print ridges and the reduction of smudges in fingerprint1.pgm and fingerprint2.pgm clearly. The highpass filter won't change the high frequencies contained in the ridges. In contrast, the filter reduces low frequency components corresponding to the smudges and background. That's why we achieved the effects above.

The GHPF and thresholding is implemented in the following code:

```
cv::Mat BHPF(cv::Mat srcImage, int D0, int N){
    CV_Assert(srcImage.data != NULL);

    int P = 2 * srcImage.rows;  // 1. obtain padding parameters P and Q
    int Q = 2 * srcImage.cols;
    cv::Mat dstImage;
    copyMakeBorder(srcImage, dstImage, 0, P - srcImage.rows,  // 2. form a padding
image
        0, Q - srcImage.cols, BORDER_CONSTANT, Scalar::all(0));
    dstImage.convertTo(dstImage, CV_64F);
    for (int x = 0; x < srcImage.rows; x++){    // 3. center the image
        for (int y = 0; y < srcImage.cols; y++){
            dstImage.ptr<double>(x)[y] *= pow(-1, x + y);
        }
    }

    // 4. compute the DFT
    cv::Mat planes[] = { cv::Mat_<double>(dstImage), cv::Mat::zeros(dstImage.size(),
CV_64F) };
    cv::Mat dftMat;
    merge(planes, 2, dftMat);
    dft(dftMat, dftMat);

    cv::Mat BwMat(P, Q, CV_64F, Scalar(0));  // 5. calculate Butterworth lowpass filter
matrix and form the product
    for (int u = 0; u < P; u++){
        for (int v = 0; v < Q; v++){
            double distSqure = (u - P / 2) * (u - P / 2) + (v - Q / 2) * (v - Q / 2);
                double tmp = pow(distSqure, N);
                BwMat.ptr<double>(u)[v] = 1 / (1 + pow(D0, 2 * N) / tmp);
        }
    }
    split(dftMat, planes);
    planes[0] = planes[0].mul(BwMat);
    planes[1] = planes[1].mul(BwMat);
    merge(planes, 2, dftMat);      // get G(u,v) now

    idft(dftMat, dftMat);                    //6. obtain the processed image
    split(dftMat, planes);
    dstImage = planes[0];           // select real part in order to ignore parasitic
    for (int x = 0; x < dstImage.rows; x++){
        for (int y = 0; y < dstImage.cols; y++){
            dstImage.ptr<double>(x)[y] *= pow(-1, x + y);
        }
```