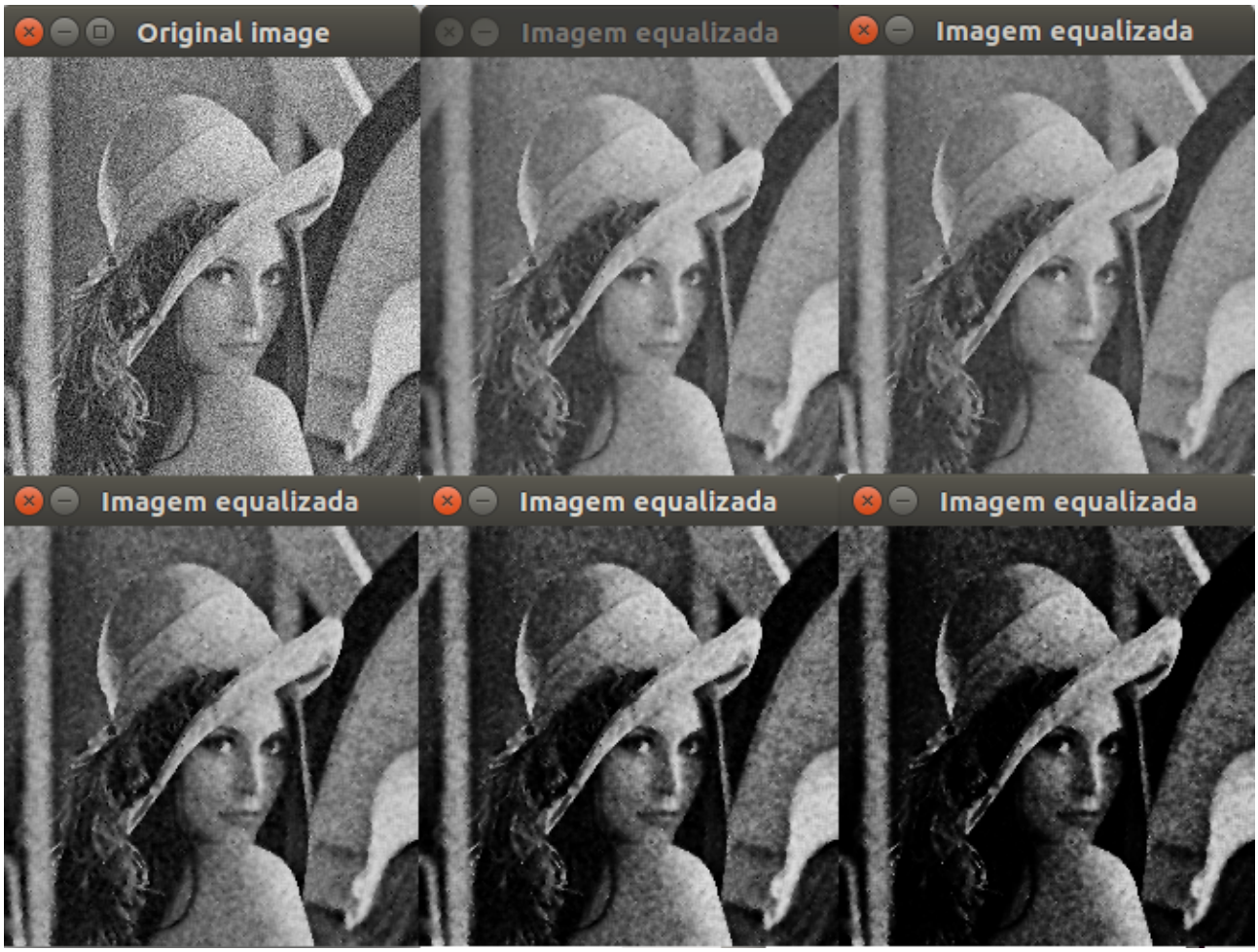
Nícolas Maduro

The codes are appended at the end.

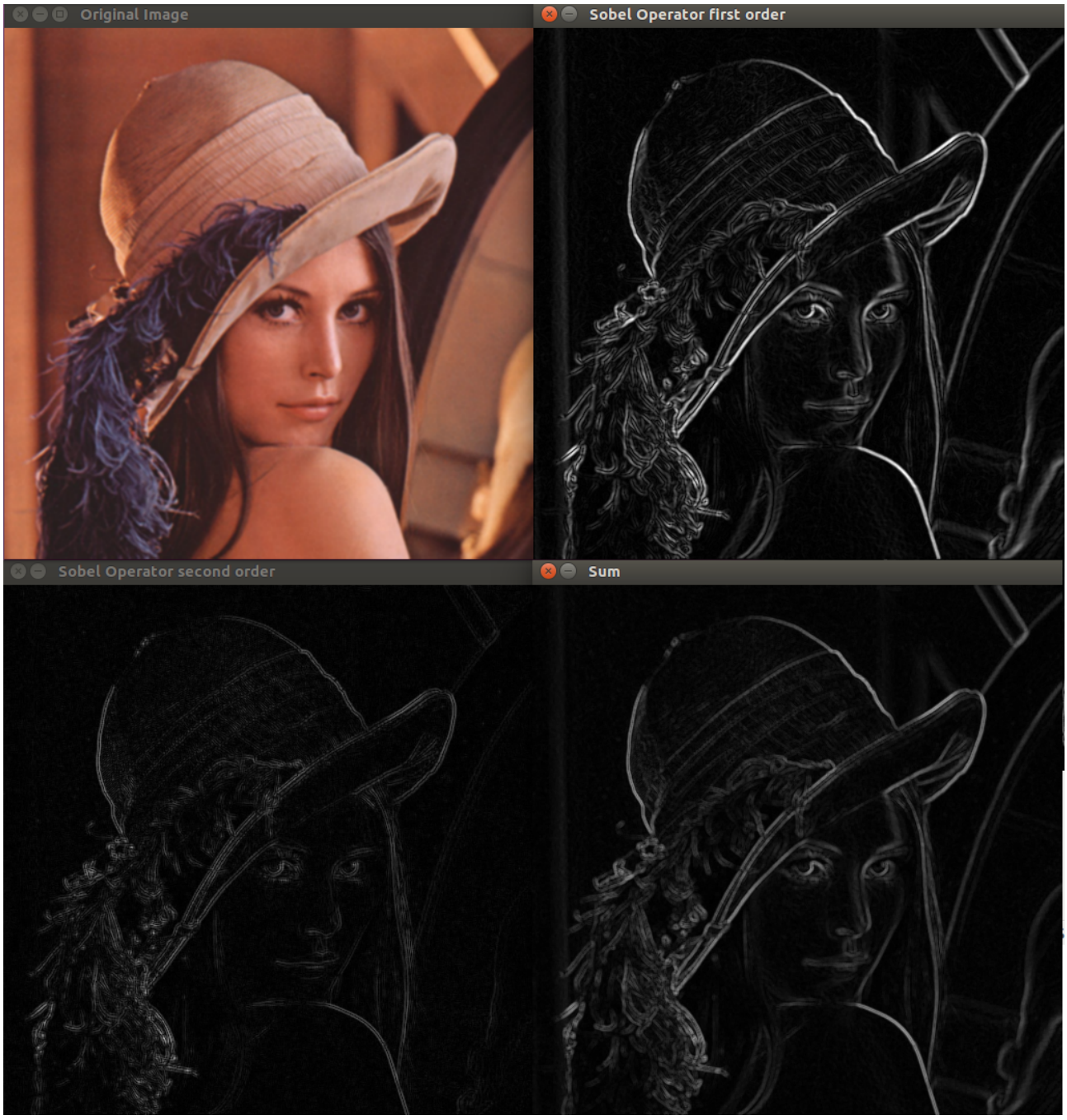
Exercise 1



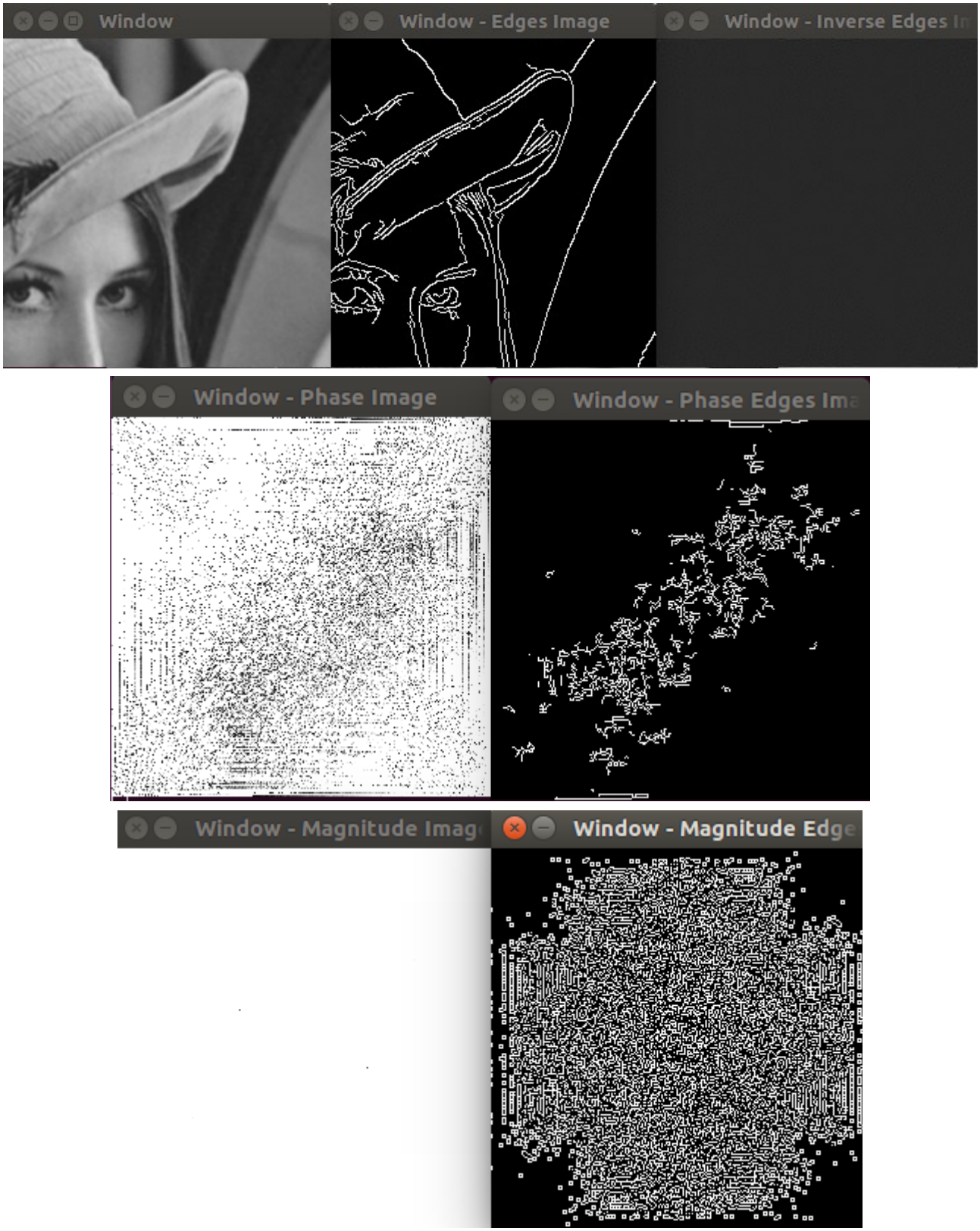
It was used in this example sigma equal to 80 and r is equal to 0, 0.5, 1 and 2 from left to right, respectivamente. As expected, when larger R, darker is the image.

Exercise 2

The edge detector work by tests using both first-order and second-order derivatives. The derivatives are calculated with the Sobel Operator.



Exercise 3



The edge detector applied in phase and magnitude windows creates uninterpretable images even when was apply inverse Fourier transform. It wasn't possible count the numbers of pixels being in the thresholding edge because the edge map thresholding was perform in Canny function from OpenCV.

```cpp
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <cmath>

using namespace cv;
using namespace std;

Mat histogramEqualization(Mat histogram) {
    float Q = 0;
    int Gmax = histogram.rows;
    Mat newHistogram(histogram.clone());
    for (int i = 1; i < Gmax; i++) {
        newHistogram.at<float>(i, 0) += newHistogram.at<float>(i - 1, 0);
    }
    return newHistogram;
}

Mat computeHistogramEqualized(Mat histogram, Mat img, float r) {
    Mat newImg(img.clone());
    float sum = 0;
    for (unsigned int w = 0; w < histogram.rows; w++)
        sum += histogram.at<float>(w, 0);

    for (unsigned int i = 0; i < img.rows; i++) {
        for (unsigned int j = 0; j < img.cols; j++) {
            newImg.at<uchar>(i, j) = img.at<uchar>(i, j) *
            pow(histogram.at<float>(img.at<uchar>(i, j), 0), r);
        }
    }
    return newImg;
}

Mat getHistogram1C(Mat img) {
    int histSize = 256;
    float range[] = { 0, 256 } ;
    const float* histRange = { range };
    bool uniform = true;
    bool accumulate = false;
    Mat hist;
    calcHist( &img, 1, 0, Mat(), hist, 1, &histSize, &histRange, uniform,
    accumulate );
    hist /= (img.cols * img.rows);
    return hist;
}

Mat drawHistogram1C(Mat histogram) {
    int histSize = histogram.rows;
    int hist_w = 512; int hist_h = 400;
    int bin_w = cvRound( (double) hist_w / histSize );
    Mat histImage( hist_h, hist_w, CV_8UC3, Scalar( 0, 0, 0) );
    normalize(histogram, histogram, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
    for ( int i = 1; i < histSize; i++ ) {
        line( histImage, Point( bin_w * (i - 1), hist_h -
        cvRound(histogram.at<float>(i - 1)) ) ,
              Point( bin_w * (i), hist_h - cvRound(histogram.at<float>(i)) ),
              Scalar( 255, 255, 255), 2, 8, 0  );
    }
    return histImage;
}

Mat sigmaFilter(Mat imgNoise, uchar sigma) {
    Mat img(imgNoise.clone());

    for (unsigned int i = 0; i < imgNoise.rows; i++) {
        for (unsigned int j = 0; j < imgNoise.cols; j++) {
            long int sum = 0, cont = 0;
            for (int ii = -1; ii <= 1; ii++)
                if (i + ii >= 0 && i + ii < imgNoise.rows)
```

```
68                      for (int jj = -1; jj <= 1; jj++)
69                          if (j + jj >= 0 && j + jj < imgNoise.cols)
70                              if (abs(imgNoise.at<uchar>(i, j) -
                                imgNoise.at<uchar>(i + ii, j + jj)) < sigma) {
71                                  sum = sum + imgNoise.at<uchar>(i + ii, j + jj);
72                                  cont++;
73                              }
74              if (cont)
75                  img.at<uchar>(i, j) = (int)sum / cont;
76          }
77      }
78      return  img;
79  }
80
81
82  int main (int argv, char** args) {
83      Mat img = imread(args[1], CV LOAD IMAGE GRAYSCALE);
84      uchar sigma = atoi(args[2]);
85      float r = atoi(args[3]) / 10.0;
86      Mat filteredImg = sigmaFilter(img, sigma);
87
88      Mat hist = getHistogram1C(filteredImg);
89      Mat equaHist = histogramEqualization(hist);
90      Mat equaImg = computeHistogramEqualized(equaHist, filteredImg, r);
91
92      namedWindow("Original image", CV WINDOW AUTOSIZE );
93      imshow("Original image", img );
94      namedWindow("Sigma Filter image", CV WINDOW AUTOSIZE );
95      imshow("Sigma Filter image", filteredImg);
96      namedWindow("Imagem equalizada", CV WINDOW AUTOSIZE );
97      imshow("Imagem equalizada", equaImg);
98
99      waitKey(0);
100
101 }
102
```

```cpp
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <cmath>

using namespace cv;
using namespace std;

Mat sobelOperator(Mat img, int order) {
    int scale = 1, delta = 0, ddepth = CV 16S;
    Mat grayImg;

    GaussianBlur( img, img, Size(3, 3), 0, 0, BORDER DEFAULT );

    cvtColor( img, grayImg, CV BGR2GRAY );
    Mat imgDer, imgDerX, imgDerY, gradImgDerX, gradImgDerY;

    Sobel( grayImg, imgDerX, ddepth, order, 0, 3, scale, delta, BORDER DEFAULT );
    convertScaleAbs( imgDerX, gradImgDerX );

    Sobel( grayImg, imgDerY, ddepth, 0, order, 3, scale, delta, BORDER DEFAULT );
    convertScaleAbs( imgDerY, gradImgDerY);

    addWeighted( gradImgDerX, 0.5, gradImgDerY, 0.5, 0, imgDer);
    return imgDer;
}

int main (int argv, char** args) {
    Mat img = imread(args[1]);
    if ( !img.data )
        return -1;
    int order = 2;
    Mat imgDer2 = sobelOperator(img, order);
    Mat imgDer =  sobelOperator(img, order - 1);

    Mat sum = (imgDer + imgDer2) / 2;

    namedWindow("Original Image", CV WINDOW AUTOSIZE );
    imshow("Original Image", img );
    namedWindow("Sobel Operator first order", CV WINDOW AUTOSIZE );
    imshow("Sobel Operator first order", imgDer);
    namedWindow("Sobel Operator second order", CV WINDOW AUTOSIZE );
    imshow("Sobel Operator second order", imgDer2);
    namedWindow("Sum", CV WINDOW AUTOSIZE );
    imshow("Sum", sum);
    waitKey(0);
}

```

```cpp
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>

using namespace cv;
using namespace std;

void computeDft(Mat img, Mat *phaImg, Mat *magImg) {
    Mat paddedImg;
    int m = getOptimalDFTSize( img.rows );
    int n = getOptimalDFTSize( img.cols );
    copyMakeBorder(img, paddedImg, 0, m - img.rows, 0, n - img.cols,
    BORDER CONSTANT, Scalar::all(0));

    Mat planesImg[] = {Mat <float>(paddedImg), Mat::zeros(paddedImg.size(), CV 32F)};
    Mat complexImg;
    merge(planesImg, 2, complexImg);

    dft(complexImg, complexImg);
    split(complexImg, planesImg);

    magnitude(planesImg[0], planesImg[1], *magImg);
    phase(planesImg[0], planesImg[1], *phaImg);
}

Mat CannyThreshold(Mat img, int thr1, int thr2){
    Mat edgesImg;
    Canny( img, edgesImg, thr1, thr2, 3 );
    return edgesImg;
}

Mat computeInverseDft(Mat phaImg,Mat magImg){
    Mat plane[2];
    polarToCart(magImg, phaImg, plane[0], plane[1]);

    Mat inverseComplexImg, inverseImg;

    merge(plane, 2, inverseComplexImg);
    dft(inverseComplexImg, inverseImg , DFT INVERSE | DFT REAL OUTPUT);
    normalize(inverseImg, inverseImg, 0, 1, CV MINMAX);

    return inverseImg;
}

void CallBackFunc(int event, int x, int y, int flags, void* param){
    Mat* img = (Mat*) param;
    int windowsSize = 250;

    namedWindow("Window",   WINDOW AUTOSIZE);
    namedWindow("Window - Edges Image", WINDOW AUTOSIZE);
    namedWindow("Window - Phase Edges Image",   WINDOW AUTOSIZE);
    namedWindow("Window - Magnitude Edges Image",   WINDOW AUTOSIZE);
    namedWindow("Window - Phase Image", WINDOW AUTOSIZE);
    namedWindow("Window - Magnitude Image", WINDOW AUTOSIZE);
    namedWindow("Window - Inverse Edges Image", WINDOW AUTOSIZE);

    if ( event == EVENT MOUSEMOVE )
    {
        unsigned int k, l;
        if (x > (windowsSize) / 2 &&    y > (windowsSize) / 2 && x < (*img).cols -
        (windowsSize) / 2 && y < (*img).rows - (windowsSize) / 2) {
            Mat zoomNoiseImg(windowsSize, windowsSize, CV 8U, Scalar(0));
            for (unsigned int i = x - (windowsSize) / 2, k = 0; i < x +
            (windowsSize) / 2; k++, i++) {
                for (unsigned int j = y - (windowsSize) / 2, l = 0; j < y +
                (windowsSize) / 2; l++, j++) {
                    zoomNoiseImg.at<uchar>(l, k) = (*img).at<uchar>(j, i);
                }
            }
```

```
67                Mat zoomImg;
68                blur( zoomNoiseImg, zoomImg, Size(3, 3));
69
70                Mat phaImg, magImg, absPhaImg, absMagImg;
71                computeDft(zoomImg, &phaImg, &magImg);
72
73                convertScaleAbs(phaImg, absPhaImg);
74                convertScaleAbs(magImg, absMagImg);
75
76                Mat edgesPhaImg = CannyThreshold(absPhaImg, 10, 30);
77                Mat edgesMagImg = CannyThreshold(absMagImg, 10, 30);
78
79                edgesPhaImg.convertTo(absPhaImg,CV 32F,1,0);
80                edgesMagImg.convertTo(absMagImg,CV 32F,1,0);
81
82                Mat inverseEdgeImg=computeInverseDft(absPhaImg,absMagImg);
83                Mat edgesImg = CannyThreshold(zoomImg,40,120);
84
85                imshow("Window", zoomImg);
86                imshow("Window - Edges Image", edgesImg);
87                imshow("Window - Phase Edges Image", edgesPhaImg);
88                imshow("Window - Magnitude Edges Image", edgesMagImg);
89                imshow("Window - Phase Image", phaImg);
90                imshow("Window - Magnitude Image", magImg);
91                imshow("Window - Inverse Edges Image", inverseEdgeImg);
92            }
93        }
94    }
95
96    int main (int argv, char** args) {
97        Mat img = imread(args[1], CV LOAD IMAGE GRAYSCALE);
98        namedWindow("Auto", WINDOW AUTOSIZE);
99        setMouseCallback("Auto", CallBackFunc, &img);
100       imshow("Auto", img);
101       waitKey(0);
102       return 0;
103   }
104
```