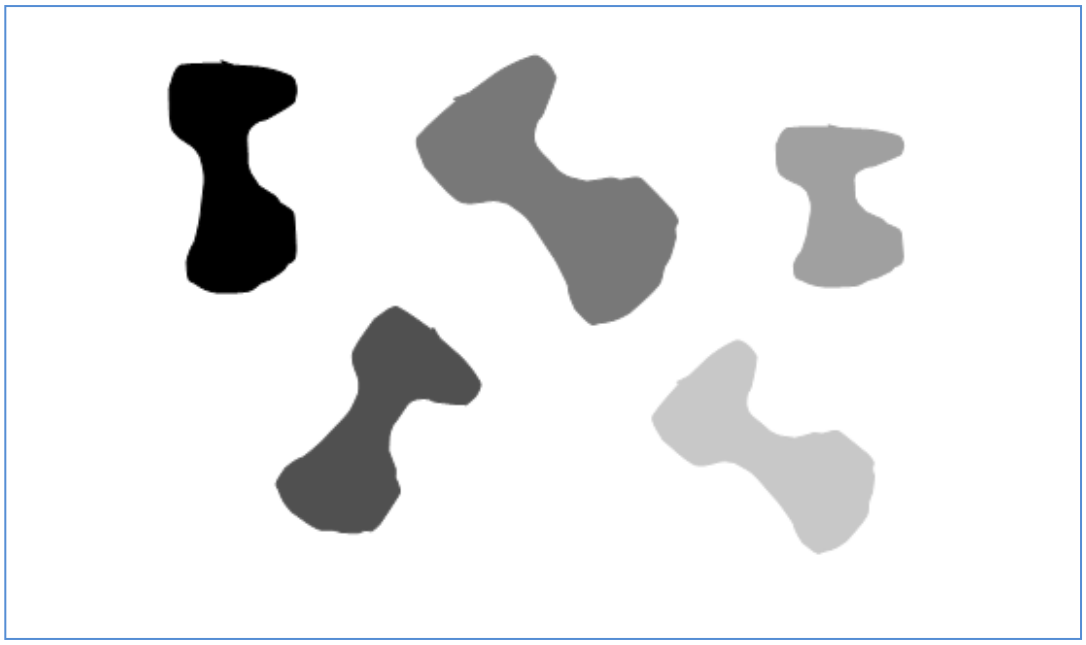
นายพีรณัฐ ธารทะเลทอง 550610530

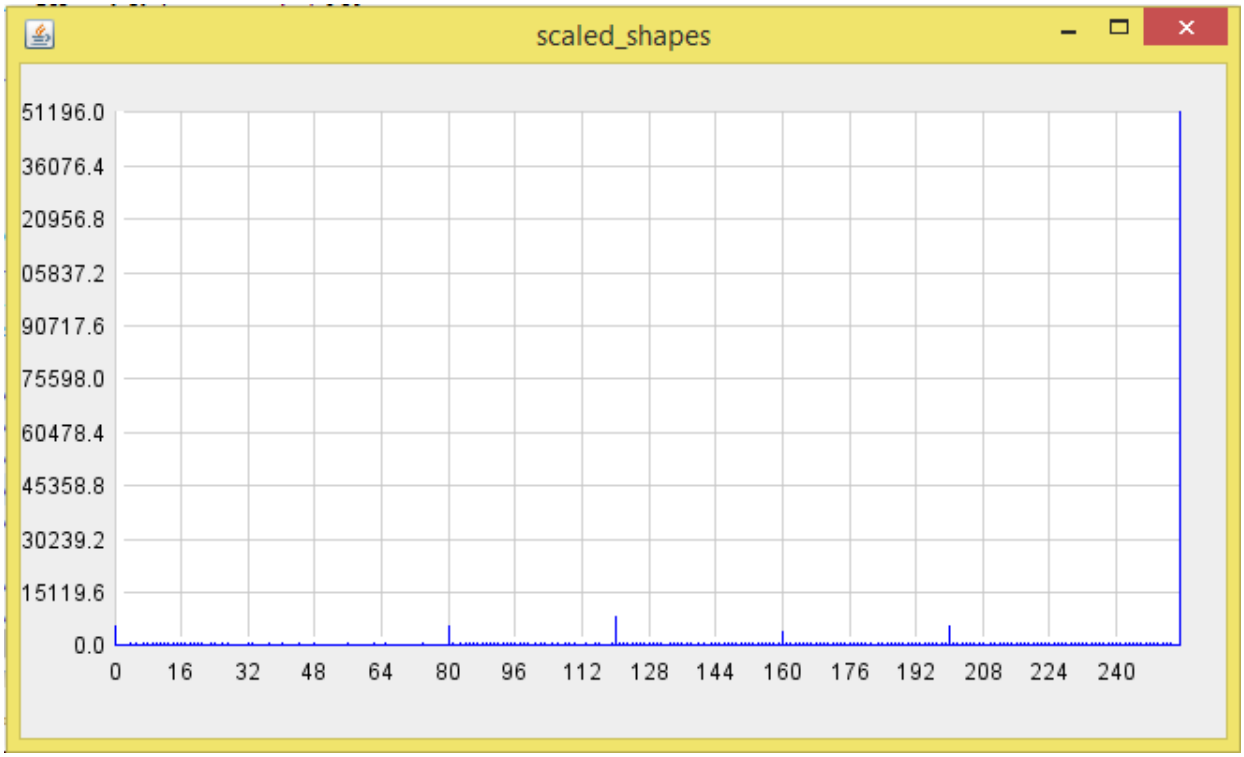
Digital Image Processing (261453)

Computer Assignment 1

1. Histogram and Object Moment



รูป input



Histogram

จาก Histogram จะเห็นว่ามีค่าที่โดดขึ้นมา 6 ค่า และค่าที่ระดับ 255 คือสีขาวที่เป็น background ดังนั้น 5 ค่าที่เหลือคือ object โดยแต่ละ object มีค่า gray level ที่ 0, 80, 120, 160, 200

ค่า Quantity จะขึ้นอยู่กับรูปร่างและขนาดของรูป Object สังเกตุจากค่า Quantity ของ Object 1, 2, 5 ที่มีค่าใกล้เคียงกัน เพราะว่ามีขนาดและรูปร่างใกล้เคียงกัน แค่หมุนรูปเฉยๆ

Object 1

Gray level : 0

Center of mass : x = 116.130408532904, y = 85.51298047896961

Central moments : U20 = 1100035.4952706748, U02 = 6345057.412758953

Quantity : 0.3015311112447032

Object 2

Gray level : 80

Center of mass : x = 189.08030669895078, y = 215.0449959644875

Central moments : U20 = 2456890.0379338027, U02 = 4941000.965899928

Quantity : 0.30119331814209316

Object 3

Gray level : 120

Center of mass : x = 280.5477487050073, y = 95.14982069331917

Central moments : U20 = 8460663.084340539, U02 = 7875173.002258037

Quantity : 0.2881819480564569

Object 4

Gray level : 160

Center of mass : x = 428.01040462427744, y = 100.97976878612717

Central moments : U20 = 975991.6254335531, U02 = 2194070.5838150145

Quantity : 0.2647985406502529

Object 5

Gray level : 200

Center of mass : x = 391.4387487386478, y = 227.5313824419778

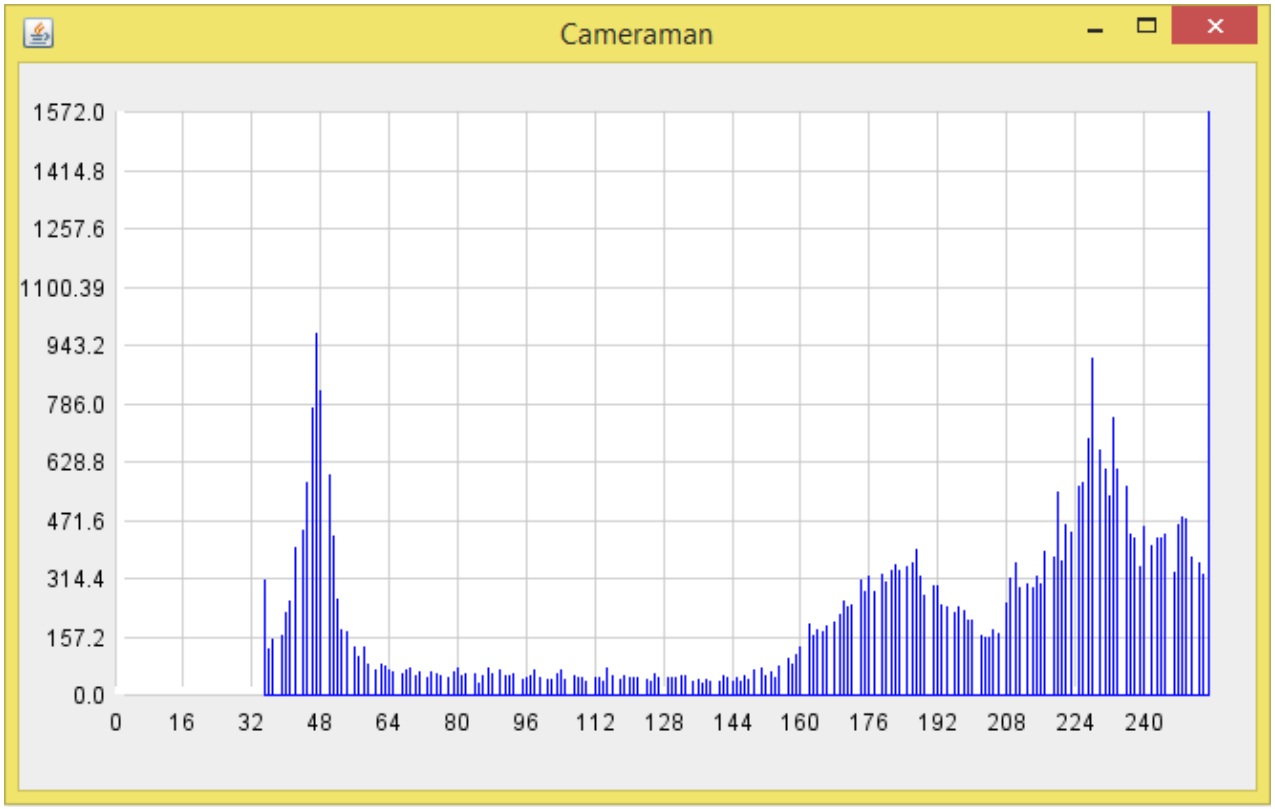
Central moments : U20 = 4792344.160242188, U02 = 3007131.8700302355

Quantity : 0.3176713949367689

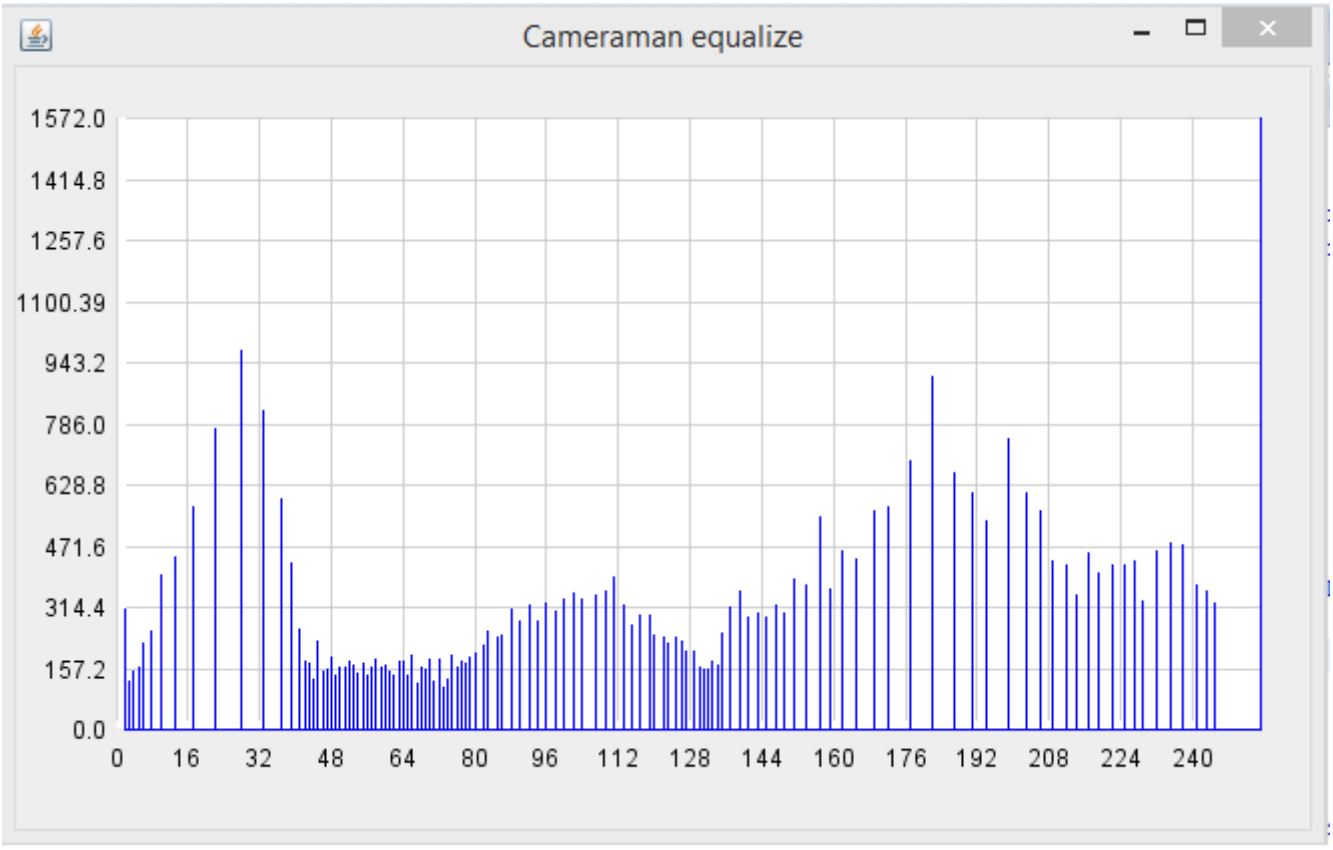
2. Point Operations

เริ่มจากหา Histogram ของรูป จากนั้นหาร Histogram แต่ละค่าด้วยพื้นที่ของรูป จะได้ Probability mass function จากนั้นหา cdf โดยการบวกค่าก่อนหน้าของ Probability mass function มาเรื่อยๆ จากนั้นคูณ cdf แต่ละค่าด้วยค่า max gray level และปัดเศษให้เป็นจำนวนเต็ม สุดท้ายเราจะได้ gray level ใหม่ และเปลี่ยนค่า gray level ของรูปให้เป็น gray level ใหม่ ก็จะได้รูป output ที่ผ่านการทำ Histogram equalization

รูป Input

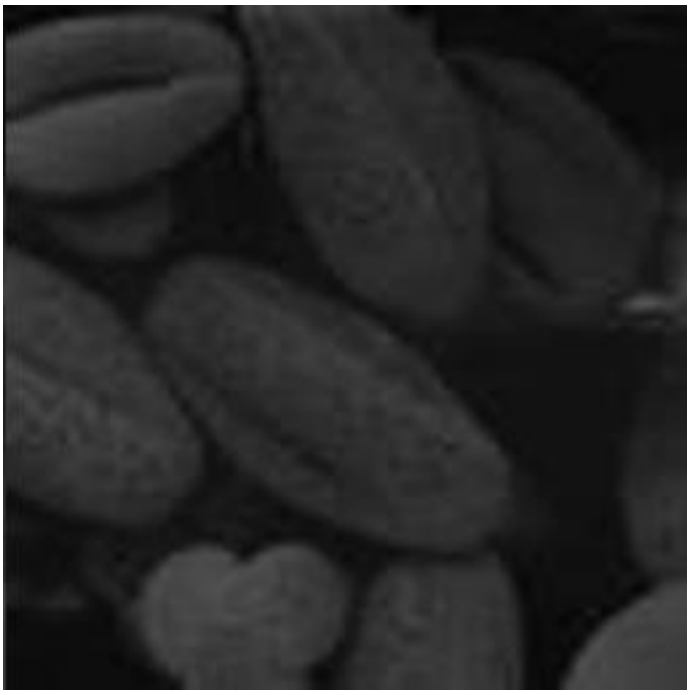


Histogram input

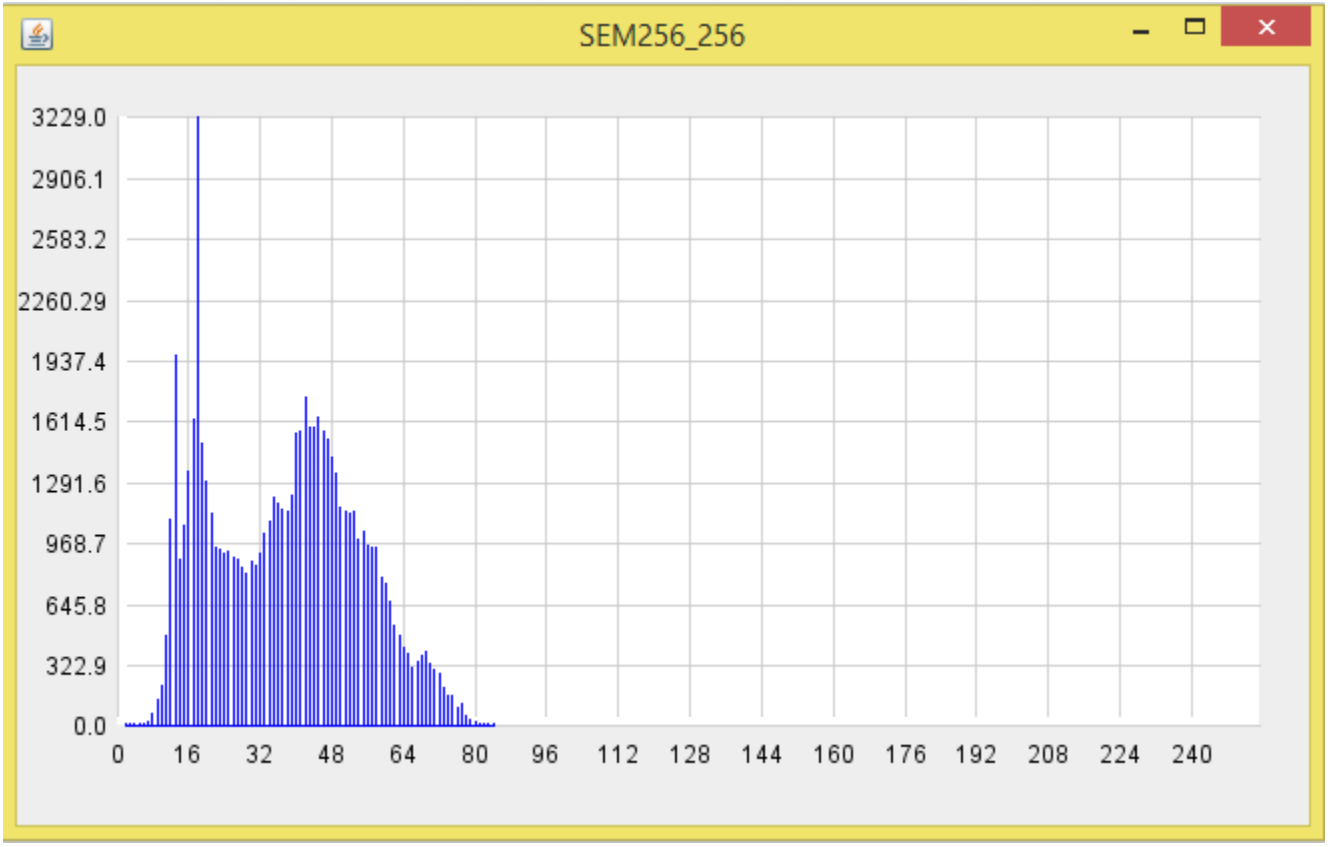


รูป Output

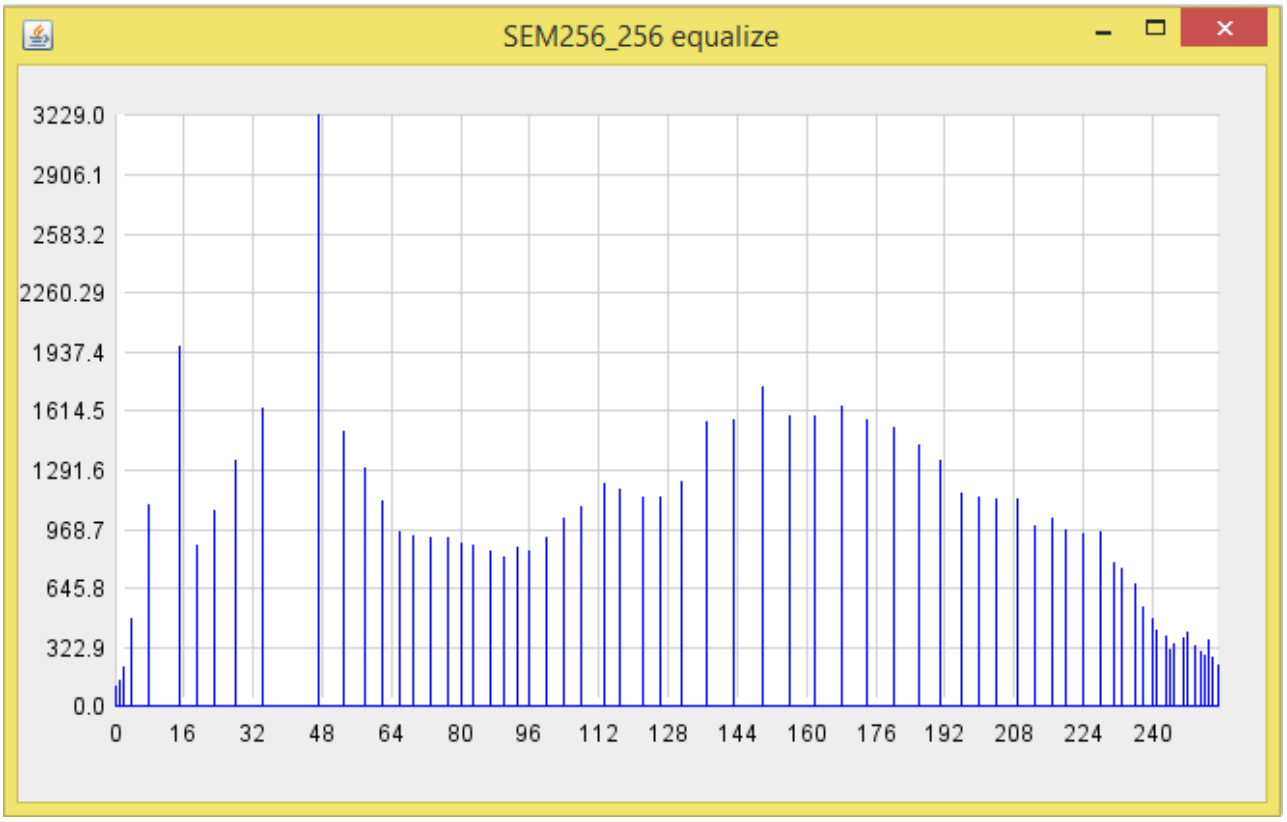
Histogram output



รูป Input



Histogram input



รูป Output

Histogram output

จากการทำ Histogram Equalization ทำให้รูป output ที่ได้ มีความถี่แต่ละ gray level ใกล้เคียงกันมากขึ้น

ทำให้ รูป Camera man ที่มีสีค่อนข้างสว่าง มืดลง และรูป SEM256_256 ที่มีสีค่อนข้างมืด สว่างขึ้น หลังผ่านการทำ Histogram Equalization

3. Algebraic Operations

Excess Green

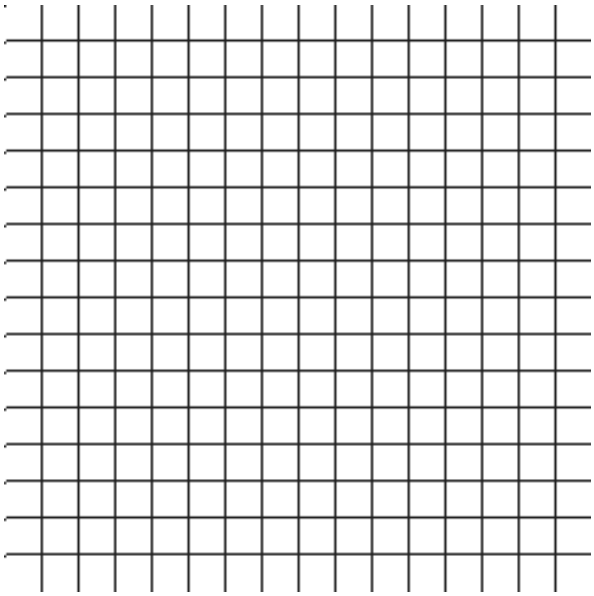
Intensity

red-blue difference

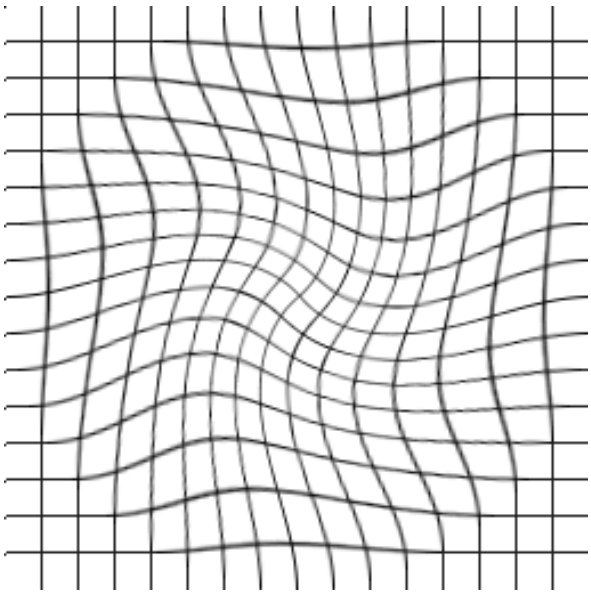
Excess Red (ทำให้ค่าสีแดง เด่นขึ้นมา)

4. Geometric Operations

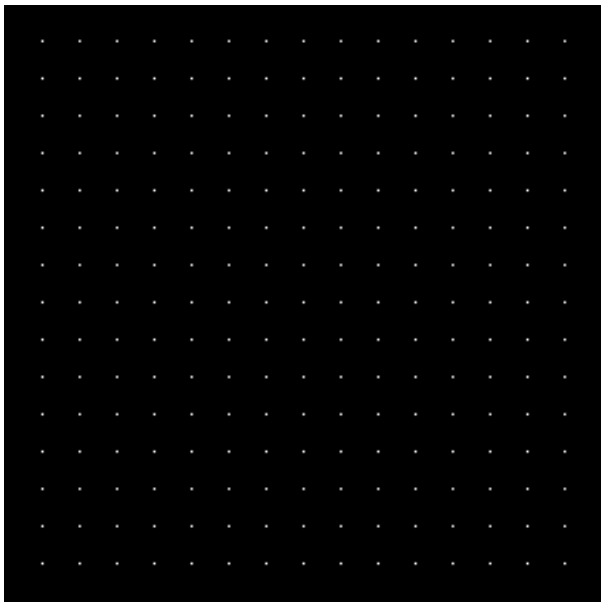
ในการทำ Control grid interpolation เพื่อทำให้รูปที่เบี้ยว กลับมาเป็นรูปปกติ เริ่มจากการหาจุดตัดแต่ละเส้นของรูป grid.pgm และรูป distgrid.pgm โดยรูป grid.pgm หาจุดตัดโดยการทำ Convolute ด้วย kernel รูปบวก ที่มี origin ตรงกลาง รูป output จะได้รูปที่มีค่าตรงจุดตัดมากกว่าจุดอื่นๆ และแปลงรูปที่ได้เป็น binary โดยตัด threshold ที่ 1000 และหาตำแหน่งของจุดนั้นต่อไป ส่วนจุดตัดในรูป distgrid.pgm ได้มาจากเทพต้าร์ที่ใช้มือจิ้มเองครับ เมื่อได้จุดตัดของสองรูปมาแล้ว ก็ใช้ วิธี Gaussian Elimination เพื่อแก้สมการ จากนั้นนำค่าทุก pixel ของรูป output มาแทนในสมการ จะได้ตำแหน่งของรูปเบี้ยว และใช้วิธี Nearest neightbor interpolation เพื่อปัดค่า และนำค่า gray level ที่ตำแหน่งนั้น มาแทนในรูป output



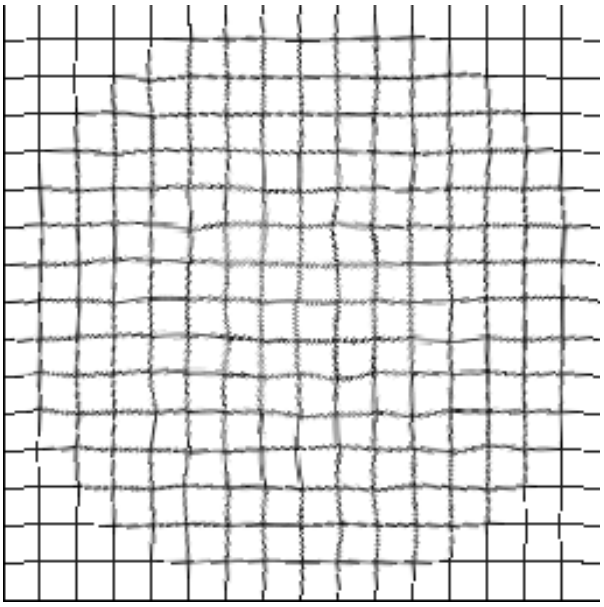
รูป grid



รูป distgrid



รูป grid หลัง convolute



รูป distgrid หลังทำ Control grid interpolation

รูป distlenna

รูป distlenna หลังทำ Control grid interpolation

Code (โค๊ดทั้งหมดอยู่ที่ https://github.com/porpeeranut/Digital_Image_Processing_Assignment1)

main_1.java

```
public class main_1 {
	public static void main(String[] args) {
		String fileName;
		fileName = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\scaled_shapes.pgm";
		int[][] pic = func.getPGMdata(fileName);
		int[] Histogram = func.getHistogramAndPrintGrayLV(pic, 100);
		func.showGraph(Histogram, "scaled_shapes");

		//	get grayLV each obj
		int[] grayLVeachObj = new int[6];
		int j = 0, threshold = 100;
		for (int i = 0; i < Histogram.length; i++) {
			if (Histogram[i] > threshold)
				grayLVeachObj[j++] = i;
		 }

		//	loop each obj
		for (int i = 0; i < grayLVeachObj.length; i++) {
			String fileNameout = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\1scaled_shapes"+i+".pgm";
			func.writePGMfile(fileNameout, func.getBW0255_imageFromGrayLVThreshold(pic, grayLVeachObj[i]));
			int[][] pic01_lv0 = func.getBW01_imageFromGrayLVThreshold(pic, grayLVeachObj[i]);
			double m00 = func.getMomentFromBW01(pic01_lv0, 0, 0);
			double m10 = func.getMomentFromBW01(pic01_lv0, 1, 0);
```

```
                    double m01 = func.getMomentFromBW01(pic01_lv0, 0, 1);
                    double x = m10/m00;
                    double y = m01/m00;

                    double n20 = func.getNormalizedMomentFromBW01(pic01_lv0, 2, 0, x, y);
                    double n02 = func.getNormalizedMomentFromBW01(pic01_lv0, 0, 2, x, y);
                    double quantity = n20+n02;
                    System.out.println("x "+x+", y "+y+", quantity "+quantity);
                    System.out.println();
                }
        }
}
```

main_2.java

```
public class main_2 {
	public static void main(String[] args) {
		String fileName1, fileName2;
		fileName1 = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\Cameraman.pgm";
		fileName2 = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\SEM256_256.pgm";
		int[][] pic1 = func.getPGMdata(fileName1);
		int[][] pic2 = func.getPGMdata(fileName2);
		int[] Histogram1 = func.getHistogram(pic1);
		int[] Histogram2 = func.getHistogram(pic2);
		func.showGraph(Histogram1, "Cameraman");
		func.showGraph(Histogram2, "SEM256_256");

		double area = pic1.length*pic1[0].length;
		pic1 = func.histogramEqualize(Histogram1, area, pic1);
		String fileNameout = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\2Cameraman.pgm";
		func.writePGMfile(fileNameout, pic1);
		Histogram1 = func.getHistogram(pic1);
		func.showGraph(Histogram1, "Cameraman equalize");

		area = pic2.length*pic2[0].length;
		pic2 = func.histogramEqualize(Histogram2, area, pic2);
		fileNameout = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\2SEM256_256.pgm";
```

```
                func.writePGMfile(fileNameout, pic2);
                Histogram2 = func.getHistogram(pic2);
                func.showGraph(Histogram2, "SEM256_256 equalize");
        }
}
```

main_3.java

```
public class main_3 {
        public static void main(String[] args) {
                 String fileR, fileG, fileB;
                fileB = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image
Processing\\HW2\\SanFranPeak_blue.pgm";
                fileG = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image
Processing\\HW2\\SanFranPeak_green.pgm";
                fileR = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image
Processing\\HW2\\SanFranPeak_red.pgm";
                int[][] picB = func.getPGMdata(fileB);
                int[][] picR = func.getPGMdata(fileR);
                int[][] picG = func.getPGMdata(fileG);
                int[] histR = func.getHistogram(picR);
                int[] histG = func.getHistogram(picG);
                int[] histB = func.getHistogram(picB);
                func.showGraph(histB, "SanFranPeak_blue");
                func.showGraph(histG, "SanFranPeak_green");
                func.showGraph(histR, "SanFranPeak_red");

                int[][] picExcessGreen = new int[picB.length][picB[0].length];
                int[][] picRedBlueDiff = new int[picB.length][picB[0].length];
                int[][] picIntensity = new int[picB.length][picB[0].length];
                int[][] picExcessRed = new int[picB.length][picB[0].length];
                //---------------------------------------------------------
                for (int row = 0; row < picB.length; row++) {
                        for (int col = 0; col < picB[0].length; col++) {
```

```
                int val = 2*picG[row][col] - picR[row][col] - picB[row][col];
                if (val < 0)
                        val = 0;
                if (val > 255)
                        val = 255;
                picExcessGreen[row][col] = val;
        }
 }
//-----------------------------------------------------------
for (int row = 0; row < picB.length; row++) {
        for (int col = 0; col < picB[0].length; col++) {
                int val = picR[row][col] - picB[row][col];
                if (val < 0)
                        val = 0;
                if (val > 255)
                        val = 255;
                picRedBlueDiff[row][col] = val;
        }
 }
//-----------------------------------------------------------
for (int row = 0; row < picB.length; row++) {
        for (int col = 0; col < picB[0].length; col++) {
                int val = (picR[row][col] + picB[row][col] + picG[row][col])/3;
                if (val < 0)
                        val = 0;
                if (val > 255)
                        val = 255;
                picIntensity[row][col] = val;
         }
}
//-----------------------------------------------------------
for (int row = 0; row < picB.length; row++) {
```

```
                for (int col = 0; col < picB[0].length; col++) {
                        int val = 2*picR[row][col] - picG[row][col] - picB[row][col];
                        if (val < 0)
                                val = 0;
                        if (val > 255)
                                val = 255;
                        picExcessRed[row][col] = val;
                }
        }
        //---------------------------------------------------------
        String fileExcessGreen = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\3ExcessGreen.pgm";
        String fileRedBlueDiff = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\3RedBlueDiff.pgm";
        String fileIntensity = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\3Intensity.pgm";
        String fileExcessRed = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\3ExcessRed.pgm";
        func.writePGMfile(fileExcessGreen, picExcessGreen);
        func.writePGMfile(fileRedBlueDiff, picRedBlueDiff);
        func.writePGMfile(fileIntensity, picIntensity);
        func.writePGMfile(fileExcessRed, picExcessRed);
    }
}
```

main_4.java

```
public class main_4 {
	public static void main(String[] args) {
		String fileGrid, fileDisgrid, fileLena;
		fileDisgrid = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\distgrid.pgm";
		fileLena = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\distlenna.pgm";
		fileGrid = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image Processing\\HW2\\grid.pgm";
		int[][] picGrid = func.getPGMdata(fileGrid);
		int[][] picDisgrid = func.getPGMdata(fileDisgrid);
		int[][] picLena = func.getPGMdata(fileLena);
		int[][] newPic = new int[picLena.length][picLena[0].length];

		// Convolute to find cross point in grid file
		int[][] F = new int[picGrid.length][picGrid[0].length];
		for (int row = 0; row < picGrid.length; row++) {
			for (int col = 0; col < picGrid[0].length; col++) {
				F[row][col] = 255 - picGrid[row][col];
			}
		}
		int[][] G = new int[][]{
				{ 0, 1, 0 },
				{ 1, 1, 1 },
				{ 0, 1, 0 }};
		int[][] C = new int[picGrid.length][picGrid[0].length];
```

```
                C = func.convoluteOriginCenter(F, G);

                String fileNameout = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image
Processing\\HW2\\4GridConvolute.pgm";
                func.writePGMfile(fileNameout, C);

                int[][][] gridXY = new int[17][17][2];
                int[][][] disgridXY = func.disgridXY;
                gridXY = func.findGridXY(C);

                newPic = func.controlGrid(gridXY, disgridXY, picDisgrid);
                fileNameout = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image
Processing\\HW2\\4disgridAfter.pgm";
                func.writePGMfile(fileNameout, newPic);

                newPic = func.controlGrid(gridXY, disgridXY, picLena);
                fileNameout = "D:\\Google Drive\\CMU\\3rd\\semester 2\\261453 Digital Image
Processing\\HW2\\4lenaAfter.pgm";
                func.writePGMfile(fileNameout, newPic);
        }
}
```

Func.java

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.RenderingHints;
import java.awt.Stroke;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JFrame;
import javax.swing.JPanel;



public class func extends JPanel {
    private int padding = 25;
    private int labelPadding = 25;
```

```
    //private Color lineColor = new Color(44, 102, 230, 180);
    private Color lineColor = Color.BLUE;
    private Color pointColor = new Color(100, 100, 100, 180);
    private Color gridColor = new Color(200, 200, 200, 200);
    private static final Stroke GRAPH_STROKE = new BasicStroke(1f);
    private int pointWidth = 4;
    private int numberYDivisions = 10;
    private List<Double> Values;

    public func(List<Double> Values) {
        this.Values = Values;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        double xScale = ((double) getWidth() - (2 * padding) - labelPadding) / (Values.size() - 1);
        double yScale = ((double) getHeight() - 2 * padding - labelPadding) / (getMaxScore() -
getMinScore());

        List<Point> graphPoints = new ArrayList<>();
        for (int i = 0; i < Values.size(); i++) {
            int x1 = (int) (i * xScale + padding + labelPadding);
            int y1 = (int) ((getMaxScore() - Values.get(i)) * yScale + padding);
            if (y1 != (int) ((getMaxScore() - 0) * yScale + padding)) {
```

```
                graphPoints.add(new Point(x1, (int) ((getMaxScore() - 0) * yScale + padding)));
            graphPoints.add(new Point(x1, y1));
            graphPoints.add(new Point(x1, (int) ((getMaxScore() - 0) * yScale + padding)));
        }
    }

    // draw white background
    g2.setColor(Color.WHITE);
    g2.fillRect(padding + labelPadding, padding, getWidth() - (2 * padding) - labelPadding, getHeight() -
2 * padding - labelPadding);
    g2.setColor(Color.BLACK);

    // create hatch marks and grid lines for y axis.
    for (int i = 0; i < numberYDivisions + 1; i++) {
        int x0 = padding + labelPadding;
        int x1 = pointWidth + padding + labelPadding;
        int y0 = getHeight() - ((i * (getHeight() - padding * 2 - labelPadding)) / numberYDivisions +
padding + labelPadding);
        int y1 = y0;
        if (Values.size() > 0) {
            g2.setColor(gridColor);
            g2.drawLine(padding + labelPadding + 1 + pointWidth, y0, getWidth() - padding, y1);
            g2.setColor(Color.BLACK);
            String yLabel = ((int) ((getMinScore() + (getMaxScore() - getMinScore()) * ((i * 1.0) /
numberYDivisions)) * 100)) / 100.0 + "";
            FontMetrics metrics = g2.getFontMetrics();
            int labelWidth = metrics.stringWidth(yLabel);
            g2.drawString(yLabel, x0 - labelWidth - 5, y0 + (metrics.getHeight() / 2) - 3);
```

```
            }
            //g2.drawLine(x0, y0, x1, y1);
        }

        // and for x axis
        for (int i = 0; i < Values.size(); i++) {
            if (Values.size() > 1) {
                int x0 = i * (getWidth() - padding * 2 - labelPadding) / (Values.size() - 1) + padding +
labelPadding;
                int x1 = x0;
                int y0 = getHeight() - padding - labelPadding;
                int y1 = y0 - pointWidth;
                if ((i % ((int) ((Values.size() / 17.0)) + 1)) == 0) {
                    g2.setColor(gridColor);
                    g2.drawLine(x0, getHeight() - padding - labelPadding - 1 - pointWidth, x1, padding);
                    g2.setColor(Color.BLACK);
                    String xLabel = i + "";
                    FontMetrics metrics = g2.getFontMetrics();
                    int labelWidth = metrics.stringWidth(xLabel);
                    g2.drawString(xLabel, x0 - labelWidth / 2, y0 + metrics.getHeight() + 3);
                }
                //g2.drawLine(x0, y0, x1, y1);
            }
        }

        // create x and y axes
        /*g2.drawLine(padding + labelPadding, getHeight() - padding - labelPadding, padding +
labelPadding, padding);
```

```
        g2.drawLine(padding + labelPadding, getHeight() - padding - labelPadding, getWidth() - padding,
getHeight() - padding - labelPadding);*/

        Stroke oldStroke = g2.getStroke();
        g2.setColor(lineColor);
        g2.setStroke(GRAPH_STROKE);
        for (int i = 0; i < graphPoints.size() - 1; i++) {
            int x1 = graphPoints.get(i).x;
            int y1 = graphPoints.get(i).y;
            int x2 = graphPoints.get(i + 1).x;
            int y2 = graphPoints.get(i + 1).y;
            g2.drawLine(x1, y1, x2, y2);
        }

        //          draw point
        /*g2.setStroke(oldStroke);
        g2.setColor(pointColor);
        for (int i = 0; i < graphPoints.size(); i++) {
            int x = graphPoints.get(i).x - pointWidth / 2;
            int y = graphPoints.get(i).y - pointWidth / 2;
            int ovalW = pointWidth;
            int ovalH = pointWidth;
            g2.fillOval(x, y, ovalW, ovalH);
        }*/
    }

//    @Override
//    public Dimension getPreferredSize() {
```

```
//        return new Dimension(width, heigth);
//    }
    public double getMinScore() {
        double minScore = Double.MAX_VALUE;
        for (Double score : Values) {
            minScore = Math.min(minScore, score);
        }
        return minScore;
    }

    public double getMaxScore() {
        double maxScore = Double.MIN_VALUE;
        for (Double score : Values) {
            maxScore = Math.max(maxScore, score);
        }
        return maxScore;
    }

    public void setValues(List<Double> Values) {
        this.Values = Values;
        invalidate();
        this.repaint();
    }

    public List<Double> getValues() {
        return Values;
    }
```

```
public static void showGraph(int[] histrogram, String head) {
      List<Double> yData = new ArrayList<>();
    for (int i = 0; i < histrogram.length; i++) {
      yData.add((double) histrogram[i]);
    }
    func mainPanel = new func(yData);
    mainPanel.setPreferredSize(new Dimension(800, 600));
    JFrame frame = new JFrame(head);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(mainPanel);
    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}

public func() {}

      public static int[] getHistogram(int[][] picData) {
                int[] Histogram = new int[256];
                for (int row = 0; row < picData.length; row++) {
                          for (int col = 0; col < picData[0].length; col++) {
                                    Histogram[picData[row][col]]++;
                          }
                 }
                return Histogram;
      }

      public static int[][] getBW01_imageFromGrayLVThreshold(int[][] picIn, int threshold) {
```

```
        int[][] picOut = new int[picIn.length][picIn[0].length];
        for (int row = 0; row < picIn.length; row++) {
                for (int col = 0; col < picIn[0].length; col++) {
                        if (picIn[row][col] == threshold)
                                picOut[row][col] = 1;
                        else
                                picOut[row][col] = 0;
                 }
        }
        return picOut;
}

public static int[][] getBW0255_imageFromGrayLVThreshold(int[][] picIn, int threshold) {
        int[][] picOut = new int[picIn.length][picIn[0].length];
        for (int row = 0; row < picIn.length; row++) {
                for (int col = 0; col < picIn[0].length; col++) {
                        if (picIn[row][col] == threshold)
                                picOut[row][col] = 255;
                        else
                                picOut[row][col] = 0;
                 }
         }
        return picOut;
}

public static int getMomentFromBW01(int[][] pic, int p, int q) {
        int m = 0;
        for (int y = 0; y < pic.length; y++) {
```

```
                for (int x = 0; x < pic[0].length; x++) {
                        m += Math.pow(x, p)*Math.pow(y, q)*pic[y][x];
                }
        }
        return m;
}

public static double getCentralMomentFromBW01(int[][] pic, int p, int q, double x_, double y_) {
        double u = 0;
        for (int y = 0; y < pic.length; y++) {
                for (int x = 0; x < pic[0].length; x++) {
                        u += Math.pow(x-x_, p)*Math.pow(y-y_, q)*pic[y][x];
                }
         }
        return u;
}

public static double getNormalizedMomentFromBW01(int[][] pic, int p, int q, double x_, double
y_) {
        double Upq = getCentralMomentFromBW01(pic, p, q, x_, y_);
        double U00 = getCentralMomentFromBW01(pic, 0, 0, x_, y_);
        System.out.println("U"+p+q+" "+Upq+", U00 "+U00);
        double n = Upq/Math.pow(U00, ((p+q)/2)+1);
        return n;
}

public static int[] getHistogramAndPrintGrayLV(int[][] picData, int threshold) {
        int[] Histogram = new int[256];
```

```
        for (int row = 0; row < picData.length; row++) {
                for (int col = 0; col < picData[0].length; col++) {
                        Histogram[picData[row][col]]++;
                }
        }
        for (int i = 0; i < Histogram.length; i++) {
                if (Histogram[i] > threshold)
                        System.out.println("level "+i+", val "+Histogram[i]);
        }
        return Histogram;
}

public static int[][] getPGMdata(String fileName) {
        FileInputStream fin;
        int[][] picData = null;
        try {
                fin = new FileInputStream(fileName);
                int len;
                byte data[] = new byte[fin.available()];
                boolean isComment = false;
                boolean isWid = true;
                boolean isHigh = false;
                int line = 1;

                int width, height;
                String strWid = "", strHigh = "", strMaxGval = "";
                len = fin.read(data);
                for (int j = 0; j < len; j++) {
```

```
//      skip comment
if (data[j] == '#') {
        isComment = true;
        continue;
}
if (isComment) {
        if (data[j] == 0x0a) {
                isComment = false;
                line++;
        }
        continue;
}


if (line == 1) {
        if (data[j] == 0x0a) {
                line++;
                continue;
        }
        if (data[j] == 'P' && data[j+1] == '5') {
                j += 2;
                continue;
        } else {
                System.out.printf("not pgm file");
                return null;
        }
} else if (line == 2) {
        if (data[j] == ' ') {
                isWid = false;
```

```
                isHigh = true;
                continue;
            }
            if (data[j] == 0x0d)
                continue;
            if (data[j] == 0x0a) {
                line++;
                continue;
            }
            if (isWid)
                strWid += "" + (char)data[j];
            if (isHigh)
                strHigh += "" + (char)data[j];
        } else if (line == 3) {
            if (data[j] == 0x0d)
                continue;
            if (data[j] == 0x0a) {
                line++;
                continue;
            }
            strMaxGval += "" + (char)data[j];
        } else {
            width = Integer.valueOf(strWid);
            height = Integer.valueOf(strHigh);
            picData = new int[height][width];
            for (int row = 0; row < height; row++) {
           for (int col = 0; col < width; col++) {
            picData[row][col] = data[j] & 0xFF;
```

```
                        j++;
                    }
                }
                    break;
            }
        }

        System.out.print("wid "+strWid);
        System.out.print(", high "+strHigh);
        System.out.println(", max "+strMaxGval);
        return picData;
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}

public static void writePGMfile(String fileNamepath, int[][] pic) {
    FileOutputStream fos;
    try {
        fos = new FileOutputStream(fileNamepath);
        String head = "P5\r\n";
        head += pic[0].length + " " + pic.length + "\r\n";
        head += "255\r\n";
        fos.write(head.getBytes());
        for (int row = 0; row < pic.length; row++) {
```

```
                        for (int col = 0; col < pic[0].length; col++) {
                                fos.write((byte)pic[row][col]);
                        }
                }
                fos.close();
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
}

public static int[][] histogramEqualize(int[] Histo, double area, int[][] pic) {
        int maxGrayLV = 255;
        double[] HistDevideArea = new double[Histo.length];
        double[] PA = new double[Histo.length];
        int[] fDa = new int[Histo.length];

        //      find Ha(Da)/Area
        for (int i = 0;i < Histo.length;i++) {
                HistDevideArea[i] = Histo[i]/area;
        }

        //      find P(A(x,y))
        PA[0] = HistDevideArea[0];
        for (int i = 1;i < Histo.length;i++) {
                PA[i] = HistDevideArea[i] + PA[i-1];
        }
```

```
        //      find f(Da)
        for (int i = 0;i < Histo.length;i++) {
                fDa[i] = (int) Math.round(PA[i] * maxGrayLV);
        }

        // convert value in pic
        for (int row = 0; row < pic.length; row++) {
                for (int col = 0; col < pic[0].length; col++) {
                        pic[row][col] = fDa[pic[row][col]];
                }
        }
        return pic;
}

public static int[][] controlGrid(int[][][] gridXY, int[][][] disgridXY2, int[][] pic) {
        int x1, x2, x3, x4;
int y1, y2, y3, y4;
double[][] xy = new double[4][4];
double[] x_ = new double[4];
double[] y_ = new double[4];
double[] Wx = new double[4];
double[] Wy = new double[4];
double[][] V = new double[xy.length][];
        int[][] newPic = new int[pic.length][pic[0].length];
        for (int row = 0; row < gridXY.length-1; row++) {
       for (int col = 0; col < gridXY[0].length-1; col++) {
        x1 = gridXY[row][col][0];
```

```
y1 = gridXY[row][col][1];
x2 = gridXY[row][col+1][0];
y2 = gridXY[row][col+1][1];
x3 = gridXY[row+1][col][0];
y3 = gridXY[row+1][col][1];
x4 = gridXY[row+1][col+1][0];
y4 = gridXY[row+1][col+1][1];

//      x'1
xy[0][0] = x1;
xy[0][1] = y1;
xy[0][2] = x1*y1;
xy[0][3] = 1;

//      x'2
xy[1][0] = x2;
xy[1][1] = y2;
xy[1][2] = x2*y2;
xy[1][3] = 1;

//      x'3
xy[2][0] = x3;
xy[2][1] = y3;
xy[2][2] = x3*y3;
xy[2][3] = 1;

//      x'4
xy[3][0] = x4;
```

```
xy[3][1] = y4;
xy[3][2] = x4*y4;
xy[3][3] = 1;

x_[0] = disgridXY[row][col][0];
x_[1] = disgridXY[row][col+1][0];
x_[2] = disgridXY[row+1][col][0];
x_[3] = disgridXY[row+1][col+1][0];

y_[0] = disgridXY[row][col][1];
y_[1] = disgridXY[row][col+1][1];
y_[2] = disgridXY[row+1][col][1];
y_[3] = disgridXY[row+1][col+1][1];

for(int i = 0; i < xy.length; i++)
    V[i] = xy[i].clone();
Wx = gaussianElimination(V, x_.clone());

for(int i = 0; i < xy.length; i++)
    V[i] = xy[i].clone();
Wy = gaussianElimination(V, y_.clone());

for (int y = y1; y < y3; y++) {
        for (int x = x1; x < x2; x++) {
          int xp = (int) Math.round(Wx[0]*x + Wx[1]*y + Wx[2]*x*y + Wx[3]);
          int yp = (int) Math.round(Wy[0]*x + Wy[1]*y + Wy[2]*x*y + Wy[3]);
          if (xp > 255)
                  xp = 255;
```

```
                if (yp > 255)
                        yp = 255;

                newPic[y][x] = pic[yp][xp];
              }
          }
        }
          }
          return newPic;
}

public static int[][] convoluteOriginCenter(int[][] F, int[][] G) {
          //        flip G left-right
          int[][] tmp = new int[G.length][G[0].length];
          for (int row = 0; row < G.length; row++) {
        for (int col = 0; col < G[0].length; col++) {
          tmp[row][col] = G[row][(G[0].length-1) - col];
        }
    }
          //        flip G up-down
          for (int row = 0; row < G.length; row++) {
        for (int col = 0; col < G[0].length; col++) {
          G[row][col] = tmp[(G.length-1) - row][col];
        }
    }

          //        padding
          int topPadSize = (int) Math.floor(G.length/2);
```

```
    int leftPadSize = (int) Math.floor(G[0].length/2);
    int[][] C = new int[(int) (F.length + topPadSize*2)][(int) (F[0].length + leftPadSize*2)];
    for (int row = topPadSize; row < C.length - topPadSize; row++) {
   for (int col = leftPadSize; col < C[0].length - leftPadSize; col++) {
    C[row][col] = F[row - topPadSize][col - leftPadSize];
   }
}


    //      convolute
    int max = 0;
    for (int Cy = topPadSize;Cy < C.length - topPadSize; Cy++) {
            for (int Cx = leftPadSize;Cx < C[0].length - leftPadSize; Cx++) {
                    int val = 0;
                    for (int row = 0; row < G.length; row++) {
                  for (int col = 0; col < G[0].length; col++)
                    val += G[row][col] * C[(Cy-topPadSize)+row][(Cx-leftPadSize)+col];
               }
                    /*if (max < val)
                            max = val;*/
                    if (val > 1000)
                            val = 255;
                    else
                            val = 0;
                    F[Cy-topPadSize][Cx-leftPadSize] = val;
            }
    }
    //System.out.println("max "+max);
    return F;
```

```
  }


  public static int[][][] findGridXY(int[][] image0255) {
            int[][][] gridXY = new int[17][17][2];
            int x;
            int y = 1;
            for (int i = 0; i < 17; i++) {
                    gridXY[0][i][0] = i*16;
            gridXY[0][i][1] = 0;
            if (i != 0)
                    gridXY[0][i][0]--;
}
            for (int row = 0; row < image0255.length; row++) {
                    gridXY[y][0][0] = 0;
            gridXY[y][0][1] = y*16;
            x = 1;
           for (int col = 0; col < image0255[0].length; col++) {
            if (image0255[row][col] == 255) {
                    gridXY[y][x][0] = col;
                    gridXY[y][x++][1] = row;
                    if (x == 16) {
                            x = 1;
                            gridXY[y][16][0] = 255;
                            gridXY[y][16][1] = y*16;
                            y++;
                    }
            }
          }
```

```
        }
            for (int i = 0; i < 17; i++) {
                gridXY[16][i][0] = i*16;
            gridXY[16][i][1] = 255;
            if (i != 0)
                gridXY[16][i][0]--;
    }
            return gridXY;
      }

      private static final double EPSILON = 1e-10;

// Gaussian elimination with partial pivoting
public static double[] gaussianElimination(double[][] A, double[] b) {
    int N  = b.length;

    for (int p = 0; p < N; p++) {

        // find pivot row and swap
        int max = p;
        for (int i = p + 1; i < N; i++) {
            if (Math.abs(A[i][p]) > Math.abs(A[max][p])) {
                max = i;
            }
        }
        double[] temp = A[p]; A[p] = A[max]; A[max] = temp;
        double   t    = b[p]; b[p] = b[max]; b[max] = t;
```

```
        // singular or nearly singular
        if (Math.abs(A[p][p]) <= EPSILON) {
            throw new RuntimeException("Matrix is singular or nearly singular");
        }

        // pivot within A and b
        for (int i = p + 1; i < N; i++) {
            double alpha = A[i][p] / A[p][p];
            b[i] -= alpha * b[p];
            for (int j = p; j < N; j++) {
                A[i][j] -= alpha * A[p][j];
            }
        }
    }

    // back substitution
    double[] x = new double[N];
    for (int i = N - 1; i >= 0; i--) {
        double sum = 0.0;
        for (int j = i + 1; j < N; j++) {
            sum += A[i][j] * x[j];
        }
        x[i] = (b[i] - sum) / A[i][i];
    }
    return x;
}

      public static int[][][] disgridXY = new int[][][] {
```

```
//        x1, y1, x2, y2, x3, y3, x4, y4
{{0, 0},{16, 0},{32, 0},{48, 0},{64, 0},{80, 0},{96, 0},{112, 0},{128, 0},{144, 0},{160, 0},{176, 0},{192, 0},{208, 0},{224, 0},{240, 0},{256, 0}},
{{0, 16},{16, 16},{32, 16},{48, 16},{64, 16},{79, 16},{97, 17},{114, 19},{130, 18},{146, 19},{160, 18},{176, 17},{192, 16},{208, 16},{224, 16},{240, 16},{256, 16}},
{{0, 32},{16, 32},{33, 32},{48, 32},{67, 31},{85, 35},{103, 37},{121, 40},{136, 42},{150, 43},{162, 41},{177, 37},{192, 35},{208, 32},{224, 32},{240, 31},{256, 32}},
{{0, 48},{16, 48},{32, 48},{51, 49},{72, 49},{94, 53},{112, 56},{128, 60},{141, 63},{154, 65},{166, 65},{178, 62},{192, 57},{206, 52},{224, 48},{240, 48},{256, 48}},
{{0, 64},{16, 64},{34, 64},{56, 63},{80, 66},{99, 68},{116, 72},{132, 76},{144, 80},{156, 84},{167, 85},{177, 83},{190, 80},{204, 74},{222, 66},{240, 64},{256, 64}},
{{0, 80},{16, 80},{37, 78},{63, 78},{84, 78},{103, 81},{119, 85},{132, 89},{144, 94},{154, 100},{165, 103},{176, 102},{188, 100},{203, 94},{221, 85},{240, 80},{256, 80}},
{{0, 96},{16, 96},{41, 93},{65, 91},{86, 90},{102, 90},{118, 96},{130, 102},{141, 108},{152, 116},{161, 117},{172, 119},{184, 116},{200, 112},{217, 105},{237, 97},{256, 96}},
{{0, 112},{18, 110},{42, 106},{65, 103},{84, 101},{100, 102},{114, 105},{127, 112},{136, 119},{145, 126},{154, 130},{167, 132},{180, 132},{196, 128},{215, 122},{237, 115},{256, 112}},
{{0, 128},{19, 126},{41, 120},{64, 113},{81, 112},{96, 112},{109, 115},{121, 120},{129, 128},{137, 135},{148, 141},{161, 143},{174, 144},{193, 142},{213, 137},{236, 131},{256, 128}},
{{0, 144},{18, 141},{40, 135},{60, 129},{76, 125},{90, 124},{101, 125},{113, 129},{121, 136},{131, 144},{142, 150},{156, 154},{172, 154},{190, 154},{212, 149},{236, 145},{256, 144}},
{{0, 160},{17, 160},{38, 151},{57, 144},{72, 140},{85, 138},{96, 138},{106, 141},{115, 148},{126, 153},{138, 161},{153, 165},{169, 167},{190, 167},{214, 163},{238, 161},{256, 160}},
{{0, 176},{16, 177},{34, 170},{53, 162},{66, 156},{81, 153},{92, 153},{102, 156},{112, 158},{124, 165},{137, 171},{153, 174},{171, 178},{192, 178},{217, 177},{240, 176},{256, 176}},
{{0, 192},{17, 192},{33, 191},{51, 182},{66, 175},{78, 170},{90, 169},{101, 172},{113, 176},{124, 181},{139, 184},{155, 188},{174, 189},{198, 193},{221, 192},{240, 192},{256, 192}},
```

{{0, 208},{16, 208},{31, 208},{49, 204},{64, 197},{80, 193},{89, 190},{101, 190},{113, 191},{128, 195},{144, 198},{161, 203},{182, 205},{204, 206},{224, 208},{240, 208},{256, 208}},

{{0, 224, 0},{16, 224},{32, 224},{48, 223},{63, 221},{80, 217},{92, 213},{106, 212},{119, 212},{133, 215},{150, 217},{168, 220},{189, 222},{208, 224},{223, 224},{241, 224},{256, 224}},

{{0, 240},{16, 240},{32, 240},{48, 240},{64, 240},{80, 239},{95, 238},{110, 237},{125, 236},{142, 237},{158, 238},{175, 239},{192, 240},{208, 240},{224, 240},{240, 240},{256, 240}},

{{0, 256},{16, 256},{32, 256},{48, 256},{64, 256},{80, 256},{96, 256},{112, 256},{128, 256},{144, 256},{160, 256},{176, 256},{192, 256},{208, 256},{224, 256},{240, 256},{256, 256}}};

}