```
/**
 * Project: DotMatrixPrinter
 * Name: Graham Burgsma
 * Created on 16 April, 2016
 */
public class Globals {
    public static final int SLIDER_START_DISTANCE = 20;
    public static final int MAX_SLIDER_DISTANCE = 1100;
    public static final int PRINT_X_SPACING = 20;
    public static final int paletteSize = 6;
    public static final int PRINT_THRESHOLD = -1; //-1 ignores edge detection, good for superman
}
```

```java
import lejos.pc.comm.NXTConnector;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;

/**
 * Project: DotPrinter
 * Name: Graham Burgsma
 * Created on 30 March, 2016
 */
public class DotPrinterPC {

    private int[][] printMatrix;

    public DotPrinterPC() {
        ImageProcessor imageProcessor = new ImageProcessor("superman.jpg");
        imageProcessor.sobelEdgeDetector();

        printMatrix = imageProcessor.imageToMatrix();
        imageProcessor.saveMatrixToFile();

        try {
            sendPrintMatrix();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new DotPrinterPC();
    }


    private void sendPrintMatrix() throws IOException {
        NXTConnector conn = new NXTConnector();

        if (!conn.connectTo("usb://")) {
            System.err.println("No NXT found using USB");
            System.exit(1);
        }

        DataOutputStream outputStream = new DataOutputStream(conn.getOutputStream());
        DataInputStream inputStream = new DataInputStream(conn.getInputStream());

        for (int i = 1; i < Globals.paletteSize; i++) {

            inputStream.readBoolean();
            //write dimensions
            outputStream.writeInt(printMatrix.length);
            outputStream.writeInt(printMatrix[0].length);
            outputStream.flush();
            inputStream.readBoolean();

            for (int[] y : printMatrix) {
                for (int x : y) {
                    System.out.print(x);

                    outputStream.writeInt(x);
                }
                System.out.println("");
                outputStream.flush();
                inputStream.readBoolean();
            }
        }
        outputStream.close();
        conn.close();
    }
}
```

```
import lejos.nxt.*;
import lejos.nxt.comm.USB;
import lejos.nxt.comm.USBConnection;
import lejos.util.Delay;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.IOException;

/**
 * Project: DotPrinter
 * Name: Graham Burgsma
 * Created on 30 March, 2016
 */

public class DotPrinterNXT {

    private static final int PEN_DOWN_ROTATION = -48;
    private static final int PEN_UP_ROTATION = -38;
    private static final int PAPER_INCREMENT = 18;
    private int colourIteration = 1;
    private int height;
    private int width;
    private DataInputStream inputStream;
    private DataOutputStream outputStream;

    public DotPrinterNXT() {
        setup();
        changePen();

        USBConnection conn = USB.waitForConnection();
        inputStream = conn.openDataInputStream();
        outputStream = conn.openDataOutputStream();

        for (int i = 1; i < Globals.paletteSize; i++) {
            Motor.B.rotateTo(0, false);
            Motor.C.rotateTo(PEN_UP_ROTATION, false);
            Delay.msDelay(300);

            try {
                printMatrix();
            } catch (IOException e) {
                e.printStackTrace();
            }

            if (colourIteration == Globals.paletteSize - 1) {
                try {
                    inputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                conn.close();
                drawBorder();
                ejectPaper();
            } else {
                reversePaperFeed();
                colourIteration++;
                LCD.clear(0);

                changePen();
            }
        }
    }

    public static void main(String[] args) {
        new DotPrinterNXT();
    }

    private void changePen() {
        if (colourIteration == 1) {
            Sound.playSample(new File("robotred.wav"));
            LCD.drawString("Insert RED", 0, 0);
        } else if (colourIteration == 2) {
            Sound.playSample(new File("robotgreen.wav"));
            LCD.drawString("Insert GREEN", 0, 0);
        } else if (colourIteration == 3) {
            Sound.playSample(new File("robotblue.wav"));
            LCD.drawString("Insert BLUE", 0, 0);
        } else if (colourIteration == 4) {
            LCD.drawString("Insert YELLOW", 0, 0);
        } else if (colourIteration == 5) {
            Sound.playSample(new File("robotblack.wav"));
            LCD.drawString("Insert BLACK", 0, 0);
        }
        LCD.drawString("Press Button", 0, 1);
        Motor.C.rotateTo(0, true);
        Motor.B.rotateTo(Globals.MAX_SLIDER_DISTANCE / 2, true);
        Button.waitForAnyPress();
        LCD.clear();
    }

    private void setup() {
        Motor.C.setSpeed(500);
        resetSlider();
        resetPen();
        Motor.B.rotateTo(1300, false);
        feedPaperIn();

        Motor.A.resetTachoCount();
    }

    private void drawBorder() {
        Motor.A.setSpeed(150);
        Motor.A.rotateTo(0, false);

        Motor.B.rotateTo(0, false);
        Motor.C.rotateTo(PEN_DOWN_ROTATION, false);
        Motor.B.rotateTo((width * Globals.PRINT_X_SPACING) + Globals.SLIDER_START_DISTANCE
  + Globals.PRINT_X_SPACING, false);

        Motor.A.rotateTo((height * PAPER_INCREMENT) + PAPER_INCREMENT, false);
        Motor.B.rotateTo(0, false);
        Motor.A.rotateTo(0, false);

        Motor.C.rotateTo(0, false);
    }

    private void resetSlider() {
        TouchSensor touch = new TouchSensor(SensorPort.S1);
        while (!touch.isPressed()) {
            Motor.B.backward();
            Delay.msDelay(50);
        }
        Motor.B.stop();
        Motor.B.resetTachoCount();
    }

    private void resetPen() {
        Motor.C.setStallThreshold(50, 300);
        Motor.C.setSpeed(50);

        while (!Motor.C.isStalled()) {
            Motor.C.forward();
            Delay.msDelay(100);
        }
        Motor.C.resetTachoCount();
        Motor.C.stop();
    }

    private void printMatrix() throws IOException {
        outputStream.writeBoolean(true);
        outputStream.flush();
        height = inputStream.readInt();
        width = inputStream.readInt();
        outputStream.writeBoolean(true);
        outputStream.flush();

        LCD.drawString(colourIteration == 1 ? "RED" : colourIteration == 2 ? "GREEN" : colourIteration ==
  3 ? "BLUE" : "BLACK", 0, 3);

        for (int y = 0; y < height; y++) {
            LCD.drawString("Line: " + (y + 1) + "/" + height, 0, 4);
            for (int x = 0; x < width; x++) {
                if (inputStream.readInt() == colourIteration) {
                    Motor.B.rotateTo((x * Globals.PRINT_X_SPACING) + Globals.
SLIDER_START_DISTANCE, false);
                    drawDot();
                }
            }
            outputStream.writeBoolean(true); //Give ready symbol
            outputStream.flush();

            resetSlider();
            incrementPaperFeed();
        }
    }

    private void drawDot() {
        Motor.C.setSpeed(230); //was 200

        Motor.C.rotateTo(PEN_DOWN_ROTATION);
        Motor.C.rotateTo(PEN_UP_ROTATION);
    }

    private void incrementPaperFeed() {
        Motor.A.setSpeed(50);
        Motor.A.rotate(PAPER_INCREMENT, false);
    }

    private void reversePaperFeed() {
        Motor.A.rotateTo(0, false);
    }

    private void feedPaperIn() {
        new ColorSensor(SensorPort.S3);
        while (SensorPort.S3.readValue() <= 1) {
            Motor.A.forward();
        }
        Motor.A.stop();
    }

    private void ejectPaper() {
        Motor.A.setSpeed(400);
        LCD.clear();
        LCD.drawString("Press to STOP", 0, 0);

        while (!Button.ENTER.isDown()) {
            Motor.A.forward();
            Delay.msDelay(100);
        }
        Motor.A.stop();
    }
}
```

```
import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

/**
 * Project: DotPrinter
 * Name: Graham Burgsma
 * Created on 30 March, 2016
 */

public class ImageProcessor {

    public static final int[] palette = {Color.white.getRGB(), Color.red.getRGB(), Color.green.getRGB(), 
Color.blue.getRGB(), Color.yellow.getRGB(), Color.black.getRGB()};
    private int MAX_PRINT_WIDTH = 5;
    private BufferedImage originalImage, edgeImage;
    private int[][] printMatrix;

    public ImageProcessor(String imageName) {
        MAX_PRINT_WIDTH = (Globals.MAX_SLIDER_DISTANCE / Globals.PRINT_X_SPACING) - (
Globals.SLIDER_START_DISTANCE / Globals.PRINT_X_SPACING);

        try {
            originalImage = ImageIO.read(new File("images/" + imageName));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void sobelEdgeDetector() {
        SobelEdgeDetection sobelEdgeDetection = new SobelEdgeDetection(originalImage);
        edgeImage = sobelEdgeDetection.process();

        saveImage(edgeImage, "sobel.jpg");
    }

    public int[][] imageToMatrix() {
        int height = edgeImage.getHeight() / (edgeImage.getWidth() / MAX_PRINT_WIDTH);

        Image imageEdge = edgeImage.getScaledInstance(MAX_PRINT_WIDTH, height, Image.
SCALE_AREA_AVERAGING);
        Image imageOriginal = originalImage.getScaledInstance(MAX_PRINT_WIDTH, height, Image.
SCALE_AREA_AVERAGING);

        BufferedImage resizedImage = toBufferedImage(imageEdge);
        BufferedImage resizedImageOriginal = toBufferedImage(imageOriginal);

        int[][] imageMatrix = new int[resizedImage.getHeight()][resizedImage.getWidth()];

        for (int y = 0; y < resizedImage.getHeight(); y++) {
            for (int x = 0; x < resizedImage.getWidth(); x++) {
                if (getRed(resizedImage.getRGB(x, y)) + getGreen(resizedImage.getRGB(x, y)) + getBlue(
resizedImage.getRGB(x, y)) > Globals.PRINT_THRESHOLD) {
                    int minDistance = Integer.MAX_VALUE;
                    int closestColour = 0;

                    for (int i = 0; i < palette.length; i++) {
                        int distance = getDistance(resizedImageOriginal.getRGB(x, y), palette[i]);
                        if (distance < minDistance) {
                            minDistance = distance;
                            closestColour = i;
                        }
                    }
                    imageMatrix[y][x] = closestColour;
                } else {
                    imageMatrix[y][x] = 0;
                }
            }
        }

        saveImage(resizedImage, "scaledImage.jpg");

        printMatrix = imageMatrix;
        return imageMatrix;
    }

    private int getDistance(int color1, int color2) {
        return ((int) (Math.pow(getRed(color2) - getRed(color1), 2) + Math.pow(getGreen(color2) - getGreen(
color1), 2) + Math.pow(getBlue(color2) - getBlue(color1), 2)));
    }

    private int getRed(int rgb) {
        return (rgb >> 16) & 0xFF;
    }

    private int getGreen(int rgb) {
        return (rgb >> 8) & 0xFF;
    }

    private int getBlue(int rgb) {
        return rgb & 0xFF;
    }

    private BufferedImage toBufferedImage(Image img) {
        if (img instanceof BufferedImage) {
            return (BufferedImage) img;
        }

        BufferedImage bimage = new BufferedImage(img.getWidth(null), img.getHeight(null), BufferedImage.
TYPE_INT_ARGB);

        Graphics2D bGr = bimage.createGraphics();
        bGr.drawImage(img, 0, 0, null);
        bGr.dispose();

        return bimage;
    }

    public void saveMatrixToFile() {
        System.out.println(printMatrix.length);
        System.out.println(printMatrix[0].length);
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter(new File("matrix.txt")));

            for (int y = 0; y < printMatrix.length; y++) {
                for (int x = 0; x < printMatrix[0].length; x++) {
                    writer.write(String.valueOf(printMatrix[y][x]));
                    writer.write(',');
                }
                writer.write("\n");
            }
            writer.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void saveImage(BufferedImage image, String fileName) {
        File outputfile = new File("images/" + fileName);
        try {
            ImageIO.write(image, "png", outputfile);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.awt.*;
import java.awt.color.ColorSpace;
import java.awt.image.BufferedImage;
import java.awt.image.ColorConvertOp;

/**
 * Project: DotMatrixPrinter
 * Name: grahamburgsma
 * Created on 29 April, 2016
 */
public class SobelEdgeDetection {

    BufferedImage image;

    public SobelEdgeDetection(BufferedImage image) {
        this.image = image;
    }

    BufferedImage process() {
        BufferedImage greyImage = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.
TYPE_BYTE_GRAY);
        BufferedImage edgeImage = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.
TYPE_BYTE_GRAY);

        ColorConvertOp colorConvert = new ColorConvertOp(ColorSpace.getInstance(ColorSpace.
CS_GRAY), null);
        colorConvert.filter(image, greyImage);

        int count, averageVertical, averageHorizontal;
        int arrayVertical[] = {1, 2, 1, 0, 0, 0, -1, -2, -1};
        int arrayHorizontal[] = {1, 0, -1, 2, 0, -2, 1, 0, -1};

        for (int x = 0; x < greyImage.getWidth(); x++) {
            for (int y = 0; y < greyImage.getHeight(); y++) {
                count = 0;
                averageVertical = 0;
                averageHorizontal = 0;
                for (int i = x - 1; i <= x + 1; i++) {
                    for (int j = y - 1; j <= y + 1; j++) {
                        if (j > 0 && j < greyImage.getHeight() && i > 0 && i < greyImage.getWidth()) {
                            averageVertical += getRed(greyImage.getRGB(i, j)) * arrayVertical[count];
                            averageHorizontal += getRed(greyImage.getRGB(i, j)) * arrayHorizontal[count];
                        }
                        count++;
                    }
                }

                averageVertical = averageVertical < 0 ? 0 : averageVertical > 255 ? 255 : averageVertical;
                averageHorizontal = averageHorizontal < 0 ? 0 : averageHorizontal > 255 ? 255 :
averageHorizontal;

                int newColor = (int) Math.sqrt(Math.pow(averageVertical, 2) + Math.pow(averageHorizontal, 2))
;

                newColor = newColor < 0 ? 0 : newColor > 255 ? 255 : newColor;

                edgeImage.setRGB(x, y, new Color(newColor, newColor, newColor).getRGB());
            }
        }

        return edgeImage;
    }

    private int getRed(int rgb) {
        return (rgb >> 16) & 0xFF;
    }
}
```