Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

ELECTIVE 3 - CPE 0332 - 1

DIGITAL IMAGE PROCESSING

ACTIVITY 2

Submitted by:
Dela Paz, Kzyrell A.

Submitted to:

Engr. Evelyn B. Carolino

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

I. MATLAB

Source Code

```
% To find Neighbour of a given Pixel
a=magic(5);
disp('a='); disp(a);
b=input('Enter the row < size of the Matrix');
c=input(' Enter the Column < size of matrix');
disp('Element'); disp(a(b,c));
% 4 Point Neighbour
N4=[a(b+1,c), a(b-1,c), a(b,c+1), a(b,c-1)];
disp('N4='); disp(N4);
%8 Point Neighbour
N8=[a(b+1,c), a(b-1,c), a(b,c+1), a(b,c-1), a(b+1,c+1), a(b+1,c-1),
a(b-1,c-1), a(b-1,c+1)]; disp('N8=');
disp(N8);
%Diagonal Neighbour
ND=[ a(b+1,c+1), a(b+1,c-1), a(b-1,c-1), a(b-1,c+1)];
disp('ND='); disp(ND);
```

```
>> lab2first
a=
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

Enter the row < size of the Matrix 3
 Enter the Column < size of matrix 3
Element
    13

N4=
    19     7    20     6

N8=
    19     7    20     6    21    12     5    14

ND=
    21    12     5    14

>>
```

Figure 1. Neighbour of 4, 8 and Diagonal point

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

Source Code

```
% Load image
I = imread('picforlab1.jpg');
subplot(2,2,1);
imshow(I);
title('Original Image');

% Get scaling factor from user
s = input('Enter Scaling Factor: ');

% Manual scaling using imresize alternative
I_scaled = manualResize(I, s);
subplot(2,2,2);
imshow(I_scaled);
title('Scaled Image');

% Manual rotation by 60 degrees
I_rot60 = manualRotate(I_scaled, 60);
subplot(2,2,3);
imshow(I_rot60);
title('Rotated Image 60deg');

% Manual rotation by 45 degrees
I_rot45 = manualRotate(I_scaled, 45);
subplot(2,2,4);
imshow(I_rot45);
title('Rotated Image 45deg');

%% Manual Resize Function
function resized_img = manualResize(img, scale_factor)
    [rows, cols, channels] = size(img);

    % Calculate new dimensions
```

```
    new_rows = round(rows * scale_factor);
    new_cols = round(cols * scale_factor);

    % Create coordinate grids for original and new images
    [X_new, Y_new] = meshgrid(1:new_cols, 1:new_rows);

    % Map new coordinates back to original image coordinates
    X_orig = (X_new - 1) / scale_factor + 1;
    Y_orig = (Y_new - 1) / scale_factor + 1;

    % Initialize output image
    resized_img = zeros(new_rows, new_cols, channels, class(img));

    % Perform interpolation for each channel
    for c = 1:channels
        resized_img(:,:,c) = interp2(double(img(:,:,c)), X_orig,
Y_orig, 'linear', 0);
    end

    % Convert back to original data type
    resized_img = cast(resized_img, class(img));
end

%% Manual Rotate Function
function rotated_img = manualRotate(img, angle_deg)
    [rows, cols, channels] = size(img);

    % Convert angle to radians
    theta = deg2rad(angle_deg);

    % Calculate rotation matrix
    cos_theta = cos(theta);
    sin_theta = sin(theta);

    % Find corners of original image
```

```
    corners = [1, 1; cols, 1; cols, rows; 1, rows]';
    center_orig = [cols/2; rows/2];

    % Translate corners to origin, rotate, then translate back
    corners_centered = corners - center_orig;
    rotation_matrix = [cos_theta, -sin_theta; sin_theta, cos_theta];
    rotated_corners = rotation_matrix * corners_centered;

    % Find bounding box of rotated image
    min_x = min(rotated_corners(1,:));
    max_x = max(rotated_corners(1,:));
    min_y = min(rotated_corners(2,:));
    max_y = max(rotated_corners(2,:));

    % Calculate new image dimensions
    new_cols = ceil(max_x - min_x);
    new_rows = ceil(max_y - min_y);

    % Create coordinate grids for new image
    [X_new, Y_new] = meshgrid(1:new_cols, 1:new_rows);

    % Center of new image
    center_new = [new_cols/2; new_rows/2];

    % Convert new image coordinates to original image coordinates
    X_centered = X_new - center_new(1);
    Y_centered = Y_new - center_new(2);

    % Apply inverse rotation
    X_orig = cos_theta * X_centered + sin_theta * Y_centered +
center_orig(1);
    Y_orig = -sin_theta * X_centered + cos_theta * Y_centered +
center_orig(2);

    % Initialize output image
```

```
    rotated_img = zeros(new_rows, new_cols, channels, class(img));

    % Perform interpolation for each channel
    for c = 1:channels
        rotated_img(:,:,c) = interp2(double(img(:,:,c)), X_orig,
Y_orig, 'linear', 0);
    end

    % Convert back to original data type
    rotated_img = cast(rotated_img, class(img));
end
```

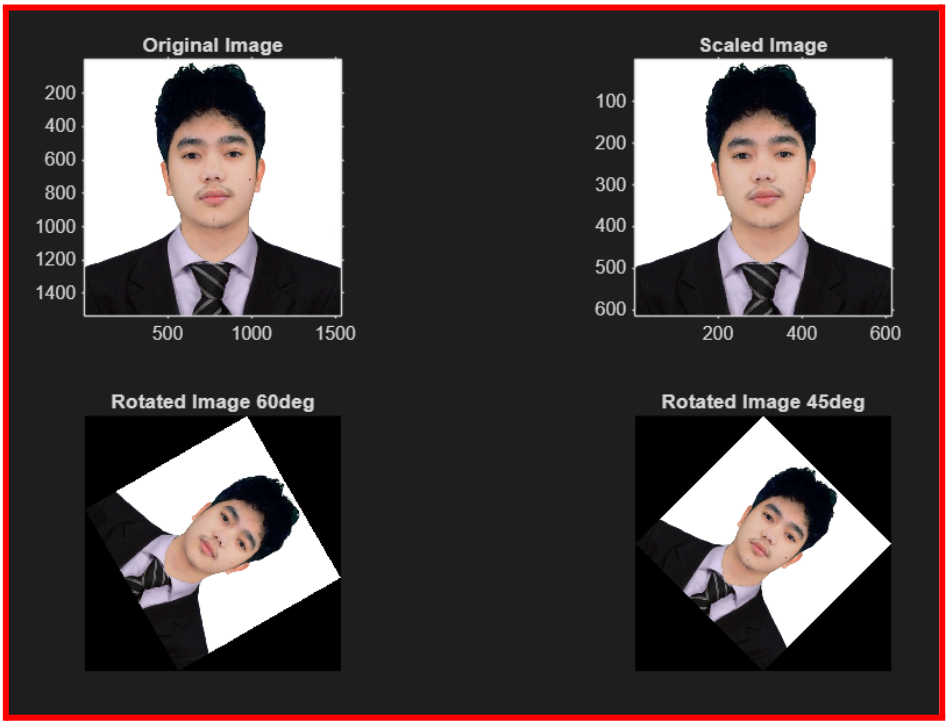

Figure 2. Scaling and Rotation

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

Source Code

```
% Display color image and resized versions using different
interpolation methods
% Load and display original image
I = imread('formalpicturekzy.jpg');
figure;
subplot(2,2,1);
imshow(I);
title('Original Image');

% Resize using different methods
methods = {'bilinear', 'nearest', 'bicubic'};
titles = {'Bilinear Image', 'Nearest Image', 'Bicubic Image'};

for i = 1:3
    % Resize image to half size
    resized = imresize(I, 0.5, methods{i});

    subplot(2,2,i+1);
    imshow(resized);
    title(titles{i});

    % Add axis with scale
    axis on;
    set(gca, 'XTick', 0:200:size(resized,2));
    set(gca, 'YTick', 0:200:size(resized,1));
    xlabel('X');
    ylabel('Y');
end

% Add axis to original image too
subplot(2,2,1);
axis on;
set(gca, 'XTick', 0:200:size(I,2));
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

```
set(gca, 'YTick', 0:200:size(I,1));
xlabel('X');
ylabel('Y');

% Adjust layout
sgtitle('Image Resizing Comparison');
```

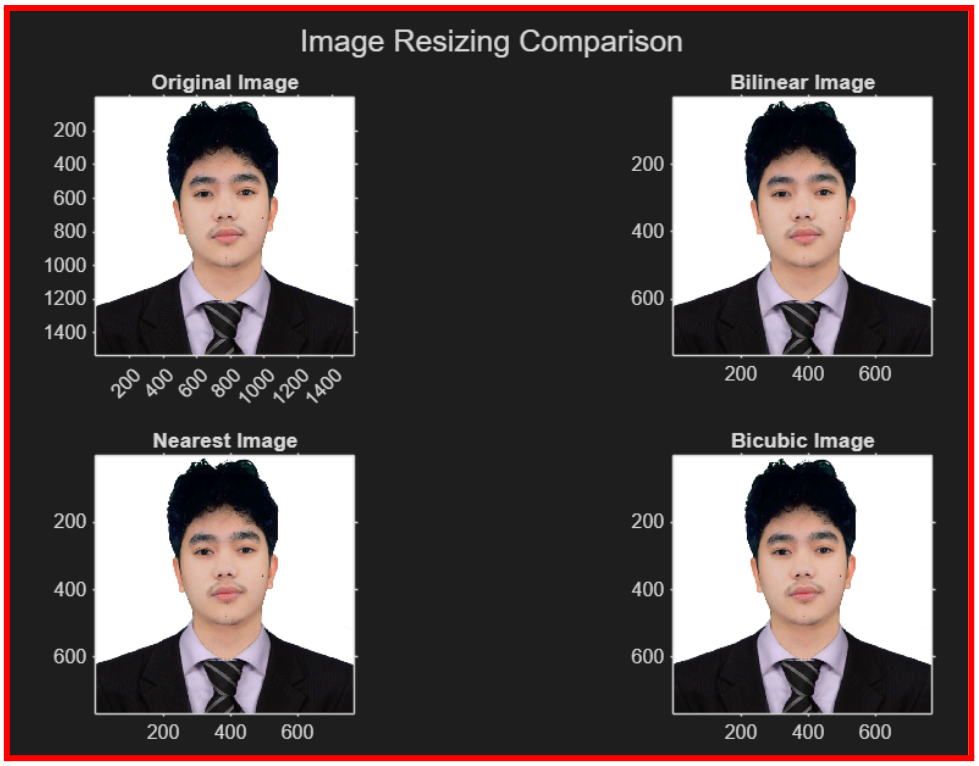

Figure 3. Image Resizing comparison

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

II. JAVA

Source Code

```java
import java.util.Scanner;

public class lab2firstjava {

    public static int[][] magicSquare() {
        // Create the magic square matrix
        return new int[][]{
            {17, 24, 1, 8, 15},
            {23, 5, 7, 14, 16},
            {4, 6, 13, 20, 22},
            {10, 12, 19, 21, 3},
            {11, 18, 25, 2, 9}
        };
    }

    public static void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int i = 0; i < row.length; i++) {
                System.out.print(row[i]);
                if (i < row.length - 1) System.out.print(" ");
            }
            System.out.println();
        }
    }

    public static void printArray(int[] array) {
        for (int i = 0; i < array.length; i++) {
            System.out.print(array[i]);
            if (i < array.length - 1) System.out.print(" ");
        }
        System.out.println();
    }
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // To find Neighbour of a given Pixel
        int[][] a = magicSquare();
        System.out.println("a=");
        printMatrix(a);

        System.out.print("Enter the row < size of the Matrix: ");
        int b = scanner.nextInt();
        System.out.print("Enter the Column < size of matrix: ");
        int c = scanner.nextInt();

        System.out.println("Element");
        System.out.println(a[b-1][c-1]); // Convert to 0-based
indexing

        // Adjust indices for 0-based indexing
        b = b - 1;
        c = c - 1;

        // 4 Point Neighbour
        int[] N4 = {a[b+1][c], a[b-1][c], a[b][c+1], a[b][c-1]};
        System.out.println("N4=");
        printArray(N4);

        // 8 Point Neighbour
        int[] N8 = {a[b+1][c], a[b-1][c], a[b][c+1], a[b][c-1],
                    a[b+1][c+1], a[b+1][c-1], a[b-1][c-1],
a[b-1][c+1]};
        System.out.println("N8=");
        printArray(N8);

        // Diagonal Neighbour
        int[] ND = {a[b+1][c+1], a[b+1][c-1], a[b-1][c-1],
```

```
a[b-1][c+1]};
            System.out.println("ND=");
            printArray(ND);


            scanner.close();
        }
}
```

```
a=
17 24 1 8 15
23 5 7 14 16
4 6 13 20 22
10 12 19 21 3
11 18 25 2 9
Enter the row < size of the Matrix: 3
Enter the Column < size of matrix: 3
Element
13
N4=
19 7 20 6
N8=
19 7 20 6 21 12 5 14
ND=
21 12 5 14
PS C:\Users\DELL\Documents\PLM Files\4th Year\Midyear - 3rd Year\Elective 3>
```

Figure 4. Neighbour of 4, 8 and Diagonal point

```
Enter Scaling Factor: 0.08
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

Source Code

```
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.*;
import java.util.Scanner;

public class lab2secondjava {

    public static void main(String[] args) {
        try {
            // Load image
            BufferedImage originalImage = ImageIO.read(new
File("picforlab1.jpg"));

            // Get scaling factor from user
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter Scaling Factor: ");
            double scaleFactor = scanner.nextDouble();

            // Manual scaling
            BufferedImage scaledImage = manualResize(originalImage,
scaleFactor);

            // Manual rotation by 60 degrees
            BufferedImage rotated60 = manualRotate(scaledImage, 60);

            // Manual rotation by 45 degrees
            BufferedImage rotated45 = manualRotate(scaledImage, 45);

            // Display images in a window
            displayImages(originalImage, scaledImage, rotated60,
```

```
rotated45);

            // Save processed images
            ImageIO.write(scaledImage, "jpg", new
File("scaled_image.jpg"));
            ImageIO.write(rotated60, "jpg", new
File("rotated_60deg.jpg"));
            ImageIO.write(rotated45, "jpg", new
File("rotated_45deg.jpg"));

            scanner.close();

        } catch (IOException e) {
            System.err.println("Error loading image: " +
e.getMessage());
        }
    }

    // Manual resize function using bilinear interpolation
    public static BufferedImage manualResize(BufferedImage img,
double scaleFactor) {
        int originalWidth = img.getWidth();
        int originalHeight = img.getHeight();

        // Calculate new dimensions
        int newWidth = (int) Math.round(originalWidth * scaleFactor);
        int newHeight = (int) Math.round(originalHeight *
scaleFactor);

        BufferedImage resizedImage = new BufferedImage(newWidth,
newHeight, img.getType());

        // Perform bilinear interpolation
        for (int y = 0; y < newHeight; y++) {
            for (int x = 0; x < newWidth; x++) {
```

```
                // Map new coordinates back to original image
coordinates
                double origX = (x / scaleFactor);
                double origY = (y / scaleFactor);

                // Get the interpolated color
                int interpolatedColor = bilinearInterpolation(img,
origX, origY);
                resizedImage.setRGB(x, y, interpolatedColor);
            }
        }

        return resizedImage;
    }

    // Manual rotate function
    public static BufferedImage manualRotate(BufferedImage img,
double angleDeg) {
        int width = img.getWidth();
        int height = img.getHeight();

        // Convert angle to radians
        double theta = Math.toRadians(angleDeg);
        double cosTheta = Math.cos(theta);
        double sinTheta = Math.sin(theta);

        // Find corners of original image
        double[][] corners = {{0, 0}, {width-1, 0}, {width-1,
height-1}, {0, height-1}};
        double centerX = width / 2.0;
        double centerY = height / 2.0;

        // Find bounding box of rotated image
        double minX = Double.MAX_VALUE, maxX = Double.MIN_VALUE;
        double minY = Double.MAX_VALUE, maxY = Double.MIN_VALUE;
```

```
        for (double[] corner : corners) {
            double x = corner[0] - centerX;
            double y = corner[1] - centerY;

            double rotatedX = x * cosTheta - y * sinTheta;
            double rotatedY = x * sinTheta + y * cosTheta;

            minX = Math.min(minX, rotatedX);
            maxX = Math.max(maxX, rotatedX);
            minY = Math.min(minY, rotatedY);
            maxY = Math.max(maxY, rotatedY);
        }

        // Calculate new image dimensions
        int newWidth = (int) Math.ceil(maxX - minX);
        int newHeight = (int) Math.ceil(maxY - minY);

        BufferedImage rotatedImage = new BufferedImage(newWidth,
newHeight, img.getType());

        // Center of new image
        double newCenterX = newWidth / 2.0;
        double newCenterY = newHeight / 2.0;

        // Fill the rotated image
        for (int y = 0; y < newHeight; y++) {
            for (int x = 0; x < newWidth; x++) {
                // Convert new image coordinates to original image
coordinates
                double xCentered = x - newCenterX;
                double yCentered = y - newCenterY;

                // Apply inverse rotation
                double origX = cosTheta * xCentered + sinTheta *
```

```
yCentered + centerX;
                double origY = -sinTheta * xCentered + cosTheta *
yCentered + centerY;

                // Get interpolated color if within bounds
                if (origX >= 0 && origX < width && origY >= 0 &&
origY < height) {
                    int interpolatedColor =
bilinearInterpolation(img, origX, origY);
                    rotatedImage.setRGB(x, y, interpolatedColor);
                } else {
                    rotatedImage.setRGB(x, y, 0); // Black background
                }
            }
        }

        return rotatedImage;
    }

    // Bilinear interpolation helper function
    private static int bilinearInterpolation(BufferedImage img,
double x, double y) {
        int x1 = (int) Math.floor(x);
        int y1 = (int) Math.floor(y);
        int x2 = Math.min(x1 + 1, img.getWidth() - 1);
        int y2 = Math.min(y1 + 1, img.getHeight() - 1);

        // Ensure coordinates are within bounds
        x1 = Math.max(0, Math.min(x1, img.getWidth() - 1));
        y1 = Math.max(0, Math.min(y1, img.getHeight() - 1));

        double dx = x - x1;
        double dy = y - y1;

        // Get the four surrounding pixels
```

```
        int rgb11 = img.getRGB(x1, y1);
        int rgb21 = img.getRGB(x2, y1);
        int rgb12 = img.getRGB(x1, y2);
        int rgb22 = img.getRGB(x2, y2);

        // Extract RGB components
        int[] c11 = {(rgb11 >> 16) & 0xFF, (rgb11 >> 8) & 0xFF, rgb11
& 0xFF};
        int[] c21 = {(rgb21 >> 16) & 0xFF, (rgb21 >> 8) & 0xFF, rgb21
& 0xFF};
        int[] c12 = {(rgb12 >> 16) & 0xFF, (rgb12 >> 8) & 0xFF, rgb12
& 0xFF};
        int[] c22 = {(rgb22 >> 16) & 0xFF, (rgb22 >> 8) & 0xFF, rgb22
& 0xFF};

        // Interpolate each channel
        int[] result = new int[3];
        for (int i = 0; i < 3; i++) {
            double top = c11[i] * (1 - dx) + c21[i] * dx;
            double bottom = c12[i] * (1 - dx) + c22[i] * dx;
            result[i] = (int) Math.round(top * (1 - dy) + bottom *
dy);
            result[i] = Math.max(0, Math.min(255, result[i])); //
Clamp to [0, 255]
        }

        return (result[0] << 16) | (result[1] << 8) | result[2];
    }

    // Display images in a JFrame
    private static void displayImages(BufferedImage original,
BufferedImage scaled,
                                      BufferedImage rotated60,
BufferedImage rotated45) {
        JFrame frame = new JFrame("Image Processing Results");
```

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new GridLayout(2, 2));

        // Scale images for display if they're too large
        int maxDisplaySize = 300;

        frame.add(createImagePanel(scaleForDisplay(original,
maxDisplaySize), "Original Image"));
        frame.add(createImagePanel(scaleForDisplay(scaled,
maxDisplaySize), "Scaled Image"));
        frame.add(createImagePanel(scaleForDisplay(rotated60,
maxDisplaySize), "Rotated 60deg"));
        frame.add(createImagePanel(scaleForDisplay(rotated45,
maxDisplaySize), "Rotated 45deg"));

        frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    private static JPanel createImagePanel(BufferedImage img, String
title) {
        JPanel panel = new JPanel(new BorderLayout());
        panel.add(new JLabel(title), BorderLayout.NORTH);
        panel.add(new JLabel(new ImageIcon(img)),
BorderLayout.CENTER);
        panel.setBorder(BorderFactory.createEtchedBorder());
        return panel;
    }

    private static BufferedImage scaleForDisplay(BufferedImage img,
int maxSize) {
        int width = img.getWidth();
        int height = img.getHeight();
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

```
        if (width <= maxSize && height <= maxSize) {
            return img;
        }

        double scale = Math.min((double) maxSize / width, (double)
maxSize / height);
        int newWidth = (int) (width * scale);
        int newHeight = (int) (height * scale);

        BufferedImage scaled = new BufferedImage(newWidth, newHeight,
img.getType());
        Graphics2D g2d = scaled.createGraphics();
        g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BILINEAR);
        g2d.drawImage(img, 0, 0, newWidth, newHeight, null);
        g2d.dispose();

        return scaled;
    }
}
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

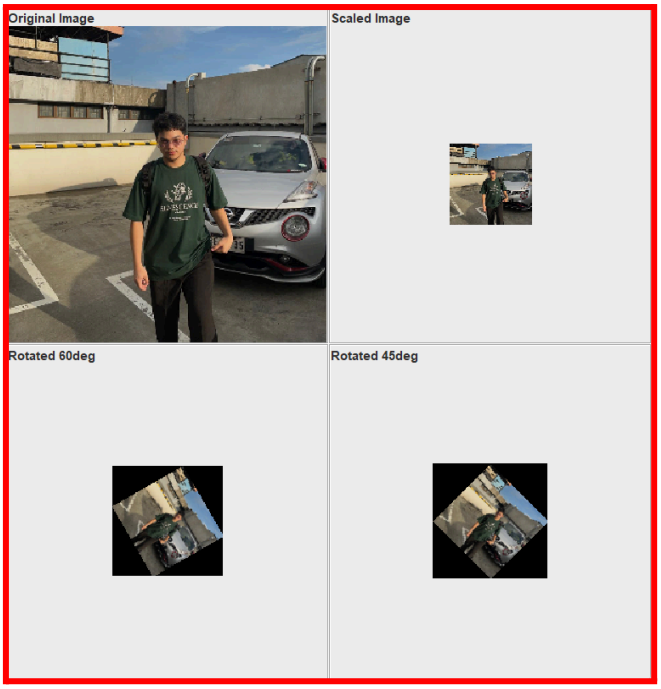

Figure 5. Scaling and Rotation

Source Code

```
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;
import javax.swing.*;

public class lab2thirdjava extends JPanel {
    private final BufferedImage[] images;
    private final String[] titles = {
        "Original Image", "Bilinear Image", "Nearest Image", "Bicubic
Image"
    };

    public lab2thirdjava(BufferedImage original) {
        int newWidth = original.getWidth() / 2;
```

```
        int newHeight = original.getHeight() / 2;

        images = new BufferedImage[4];
        images[0] = original;
        images[1] = resizeImage(original, newWidth, newHeight,
RenderingHints.VALUE_INTERPOLATION_BILINEAR);
        images[2] = resizeImage(original, newWidth, newHeight,
RenderingHints.VALUE_INTERPOLATION_NEAREST_NEIGHBOR);
        images[3] = resizeImage(original, newWidth, newHeight,
RenderingHints.VALUE_INTERPOLATION_BICUBIC);
    }

    private BufferedImage resizeImage(BufferedImage src, int width,
int height, Object interpolationHint) {
        BufferedImage resized = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = resized.createGraphics();
        g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
interpolationHint);
        g2d.drawImage(src, 0, 0, width, height, null);
        g2d.dispose();
        return resized;
    }

    private int getGridSpacing(int size) {
        if (size <= 500) return 100;
        else if (size <= 1000) return 200;
        else if (size <= 2000) return 400;
        else return 500;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        int pad = 40; // Padding between cells
```

```
int labelHeight = 20;
int tickSize = 5;
int tickFontSize = 10;
int rows = 2;
int cols = 2;

int panelW = getWidth();
int panelH = getHeight();
int cellW = (panelW - (cols + 1) * pad) / cols;
int cellH = (panelH - (rows + 1) * pad) / rows;

Graphics2D g2d = (Graphics2D) g;
g2d.setFont(new Font("SansSerif", Font.BOLD, 14));
g2d.setColor(Color.BLACK);

for (int i = 0; i < images.length; i++) {
    int row = i / cols;
    int col = i % cols;

    int x = pad + col * (cellW + pad);
    int y = pad + row * (cellH + pad);

    BufferedImage img = images[i];

    // --- Centered title above the image ---
    FontMetrics fm = g2d.getFontMetrics();
    int titleWidth = fm.stringWidth(titles[i]);
    int titleX = x + (cellW - titleWidth) / 2;
    g2d.drawString(titles[i], titleX, y);

    int imgX = x;
    int imgY = y + labelHeight;

    int imgW = cellW;
    int imgH = cellH - labelHeight;
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

```
            // --- Draw image ---
            g2d.drawImage(img, imgX, imgY, imgW, imgH, null);

            // --- Grid overlay ---
            int gridSpacing = getGridSpacing(img.getWidth());
            g2d.setColor(new Color(255, 255, 255, 80));
            for (int gx = 0; gx <= imgW; gx += gridSpacing * imgW /
img.getWidth()) {
                g2d.drawLine(imgX + gx, imgY, imgX + gx, imgY +
imgH);
            }
            for (int gy = 0; gy <= imgH; gy += gridSpacing * imgH /
img.getHeight()) {
                g2d.drawLine(imgX, imgY + gy, imgX + imgW, imgY +
gy);
            }

            // --- Axis ticks and labels ---
            g2d.setColor(Color.BLACK);
            g2d.setFont(new Font("SansSerif", Font.PLAIN,
tickFontSize));

            int xTickSpacing = getGridSpacing(img.getWidth());
            int yTickSpacing = getGridSpacing(img.getHeight());

            for (int gx = 0; gx <= img.getWidth(); gx +=
xTickSpacing) {
                int screenX = imgX + gx * imgW / img.getWidth();
                g2d.drawLine(screenX, imgY + imgH, screenX, imgY +
imgH + tickSize);
                String label = Integer.toString(gx);
                int strWidth =
g2d.getFontMetrics().stringWidth(label);
                g2d.drawString(label, screenX - strWidth / 2, imgY +
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

```
imgH + tickSize + 12);
            }

            for (int gy = 0; gy <= img.getHeight(); gy +=
yTickSpacing) {
                int screenY = imgY + gy * imgH / img.getHeight();
                g2d.drawLine(imgX - tickSize, screenY, imgX,
screenY);
                g2d.drawString(Integer.toString(gy), imgX - tickSize
- 25, screenY + 5);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        BufferedImage img = ImageIO.read(new
File("formalpicturekzy.jpg"));

        JFrame frame = new JFrame("Image Resizing with Grid and
Axes");
        lab2thirdjava panel = new lab2thirdjava(img);
        frame.add(panel);
        frame.setSize(1200, 900);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

Figure 6. Image Resizing Comparison

III. PYTHON

Source Code

```
import numpy as np

def magic_square(n):
    """Create a magic square like MATLAB's magic() function"""
    return np.array([[17, 24, 1, 8, 15],
                     [23, 5, 7, 14, 16],
                     [4, 6, 13, 20, 22],
                     [10, 12, 19, 21, 3],
                     [11, 18, 25, 2, 9]])

def print_array(arr):
    """Print array without brackets"""
    print(' '.join(map(str, arr)))
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

```
# To find Neighbour of a given Pixel
a = magic_square(5)
print('a=')
for row in a:
    print(' '.join(map(str, row)))

b = int(input('Enter the row < size of the Matrix: '))
c = int(input('Enter the Column < size of matrix: '))
print('Element'); print(a[b-1, c-1])  # Convert to 0-based indexing

# Adjust indices for 0-based indexing
b, c = b-1, c-1

# 4 Point Neighbour
N4 = [int(a[b+1,c]), int(a[b-1,c]), int(a[b,c+1]), int(a[b,c-1])]
print('N4='); print_array(N4)

# 8 Point Neighbour
N8 = [int(a[b+1,c]), int(a[b-1,c]), int(a[b,c+1]), int(a[b,c-1]),
int(a[b+1,c+1]), int(a[b+1,c-1]), int(a[b-1,c-1]), int(a[b-1,c+1])]
print('N8='); print_array(N8)

# Diagonal Neighbour
ND = [int(a[b+1,c+1]), int(a[b+1,c-1]), int(a[b-1,c-1]),
int(a[b-1,c+1])]
print('ND='); print_array(ND)
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

```
a=
17 24 1 8 15
23 5 7 14 16
4 6 13 20 22
10 12 19 21 3
11 18 25 2 9
Enter the row < size of the Matrix: 3
Enter the Column < size of matrix:  3
Element
13
N4=
19 7 20 6
N8=
19 7 20 6 21 12 5 14
ND=
21 12 5 14
PS C:\Users\DELL\Documents\PLM Files\4th Year\Midyear - 3rd Year\Elective 3>
```

Figure 7. Neighbour of 4,8 and Diagonal point

Source Code

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import math

def manual_resize(img_array, scale_factor):
    """
    Manually resize image using bilinear interpolation
    """
    if len(img_array.shape) == 3:
        rows, cols, channels = img_array.shape
    else:
        rows, cols = img_array.shape
        channels = 1
        img_array = img_array[:, :, np.newaxis]

    # Calculate new dimensions
```

```
    new_rows = int(round(rows * scale_factor))
    new_cols = int(round(cols * scale_factor))

    # Create coordinate grids for new image
    x_new = np.arange(new_cols)
    y_new = np.arange(new_rows)
    X_new, Y_new = np.meshgrid(x_new, y_new)

    # Map new coordinates back to original image coordinates
    X_orig = (X_new) / scale_factor
    Y_orig = (Y_new) / scale_factor

    # Initialize output image
    resized_img = np.zeros((new_rows, new_cols, channels),
dtype=img_array.dtype)

    # Perform bilinear interpolation for each channel
    for c in range(channels):
        resized_img[:, :, c] = bilinear_interpolation(img_array[:, :,
c], X_orig, Y_orig)

    # Remove single channel dimension if original was grayscale
    if channels == 1 and len(img_array.shape) == 2:
        resized_img = resized_img[:, :, 0]

    return resized_img

def manual_rotate(img_array, angle_deg):
    """
    Manually rotate image
    """
    if len(img_array.shape) == 3:
        rows, cols, channels = img_array.shape
    else:
        rows, cols = img_array.shape
```

```
        channels = 1
        img_array = img_array[:, :, np.newaxis]

    # Convert angle to radians
    theta = math.radians(angle_deg)
    cos_theta = math.cos(theta)
    sin_theta = math.sin(theta)

    # Find corners of original image
    corners = np.array([[0, 0], [cols-1, 0], [cols-1, rows-1], [0,
rows-1]]).T
    center_orig = np.array([cols/2, rows/2])

    # Translate corners to origin, rotate, then find bounding box
    corners_centered = corners - center_orig.reshape(-1, 1)
    rotation_matrix = np.array([[cos_theta, -sin_theta],
                                [sin_theta, cos_theta]])
    rotated_corners = rotation_matrix @ corners_centered

    # Find bounding box of rotated image
    min_x = np.min(rotated_corners[0, :])
    max_x = np.max(rotated_corners[0, :])
    min_y = np.min(rotated_corners[1, :])
    max_y = np.max(rotated_corners[1, :])

    # Calculate new image dimensions
    new_cols = int(math.ceil(max_x - min_x))
    new_rows = int(math.ceil(max_y - min_y))

    # Create coordinate grids for new image
    x_new = np.arange(new_cols)
    y_new = np.arange(new_rows)
    X_new, Y_new = np.meshgrid(x_new, y_new)

    # Center of new image
```

```
    center_new = np.array([new_cols/2, new_rows/2])

    # Convert new image coordinates to original image coordinates
    X_centered = X_new - center_new[0]
    Y_centered = Y_new - center_new[1]

    # Apply inverse rotation
    X_orig = cos_theta * X_centered + sin_theta * Y_centered +
center_orig[0]
    Y_orig = -sin_theta * X_centered + cos_theta * Y_centered +
center_orig[1]

    # Initialize output image
    rotated_img = np.zeros((new_rows, new_cols, channels),
dtype=img_array.dtype)

    # Perform interpolation for each channel
    for c in range(channels):
        rotated_img[:, :, c] = bilinear_interpolation(img_array[:, :,
c], X_orig, Y_orig)

    # Remove single channel dimension if original was grayscale
    if channels == 1 and len(img_array.shape) == 2:
        rotated_img = rotated_img[:, :, 0]

    return rotated_img

def bilinear_interpolation(img_channel, X_coords, Y_coords):
    """
    Perform bilinear interpolation on a single channel
    """
    rows, cols = img_channel.shape
    output_shape = X_coords.shape
    result = np.zeros(output_shape, dtype=img_channel.dtype)
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

```
# Flatten coordinates for easier processing
X_flat = X_coords.flatten()
Y_flat = Y_coords.flatten()

for i, (x, y) in enumerate(zip(X_flat, Y_flat)):
    # Check bounds
    if x < 0 or x >= cols-1 or y < 0 or y >= rows-1:
        continue

    # Get integer coordinates
    x1 = int(np.floor(x))
    y1 = int(np.floor(y))
    x2 = min(x1 + 1, cols - 1)
    y2 = min(y1 + 1, rows - 1)

    # Calculate weights
    dx = x - x1
    dy = y - y1

    # Get four surrounding pixels
    c11 = img_channel[y1, x1]
    c21 = img_channel[y1, x2]
    c12 = img_channel[y2, x1]
    c22 = img_channel[y2, x2]

    # Bilinear interpolation
    top = c11 * (1 - dx) + c21 * dx
    bottom = c12 * (1 - dx) + c22 * dx
    interpolated_value = top * (1 - dy) + bottom * dy

    # Convert back to 2D index
    row_idx = i // output_shape[1]
    col_idx = i % output_shape[1]
    result[row_idx, col_idx] = interpolated_value
```

```
    return result

def main():
    """
    Main function to demonstrate image processing
    """
    try:
        # Load image
        img = Image.open('picforlab1.jpg')
        img_array = np.array(img)

        # Get scaling factor from user
        scale_factor = float(input("Enter Scaling Factor: "))

        # Manual scaling
        img_scaled = manual_resize(img_array, scale_factor)

        # Manual rotation by 60 degrees
        img_rot60 = manual_rotate(img_scaled, 60)

        # Manual rotation by 45 degrees
        img_rot45 = manual_rotate(img_scaled, 45)

        # Display images
        plt.figure(figsize=(12, 8))

        plt.subplot(2, 2, 1)
        plt.imshow(img_array, cmap='gray' if len(img_array.shape) ==
2 else None)
        plt.title('Original Image')
        plt.axis('on')

        plt.subplot(2, 2, 2)
        plt.imshow(img_scaled, cmap='gray' if len(img_scaled.shape)
== 2 else None)
```

```
        plt.title('Scaled Image')
        plt.axis('on')

        plt.subplot(2, 2, 3)
        plt.imshow(img_rot60, cmap='gray' if len(img_rot60.shape) ==
2 else None)
        plt.title('Rotated Image 60deg')
        plt.axis('on')

        plt.subplot(2, 2, 4)
        plt.imshow(img_rot45, cmap='gray' if len(img_rot45.shape) ==
2 else None)
        plt.title('Rotated Image 45deg')
        plt.axis('on')

        plt.tight_layout()
        plt.show()

        # Save processed images

Image.fromarray(img_scaled.astype(np.uint8)).save('scaled_image.jpg')

Image.fromarray(img_rot60.astype(np.uint8)).save('rotated_60deg.jpg')

Image.fromarray(img_rot45.astype(np.uint8)).save('rotated_45deg.jpg')

        print("Processed images saved!")

    except FileNotFoundError:
        print("Error: Could not find 'circuit.jpg'. Please make sure
the image file exists.")
    except Exception as e:
        print(f"Error: {e}")

# Alternative using OpenCV (if available)
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

```
def opencv_version():
    """
    Alternative implementation using OpenCV for comparison
    """
    try:
        import cv2

        # Load image
        img = cv2.imread('circuit.jpg')
        if img is None:
            print("Error: Could not load image")
            return

        # Get scaling factor
        scale_factor = float(input("Enter Scaling Factor: "))

        # Resize using OpenCV
        height, width = img.shape[:2]
        new_height = int(height * scale_factor)
        new_width = int(width * scale_factor)
        img_scaled = cv2.resize(img, (new_width, new_height),
interpolation=cv2.INTER_LINEAR)

        # Rotate using OpenCV
        center = (new_width // 2, new_height // 2)
        rotation_matrix_60 = cv2.getRotationMatrix2D(center, 60, 1.0)
        rotation_matrix_45 = cv2.getRotationMatrix2D(center, 45, 1.0)

        img_rot60 = cv2.warpAffine(img_scaled, rotation_matrix_60,
(new_width, new_height))
        img_rot45 = cv2.warpAffine(img_scaled, rotation_matrix_45,
(new_width, new_height))

        # Display using matplotlib (convert BGR to RGB)
        plt.figure(figsize=(12, 8))
```

```
        plt.subplot(2, 2, 1)
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title('Original Image')
        plt.axis('on')

        plt.subplot(2, 2, 2)
        plt.imshow(cv2.cvtColor(img_scaled, cv2.COLOR_BGR2RGB))
        plt.title('Scaled Image')
        plt.axis('on')

        plt.subplot(2, 2, 3)
        plt.imshow(cv2.cvtColor(img_rot60, cv2.COLOR_BGR2RGB))
        plt.title('Rotated Image 60deg')
        plt.axis('on')

        plt.subplot(2, 2, 4)
        plt.imshow(cv2.cvtColor(img_rot45, cv2.COLOR_BGR2RGB))
        plt.title('Rotated Image 45deg')
        plt.axis('on')

        plt.tight_layout()
        plt.show()

    except ImportError:
        print("OpenCV not installed. Use the manual implementation
instead.")

if __name__ == "__main__":
    main()
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

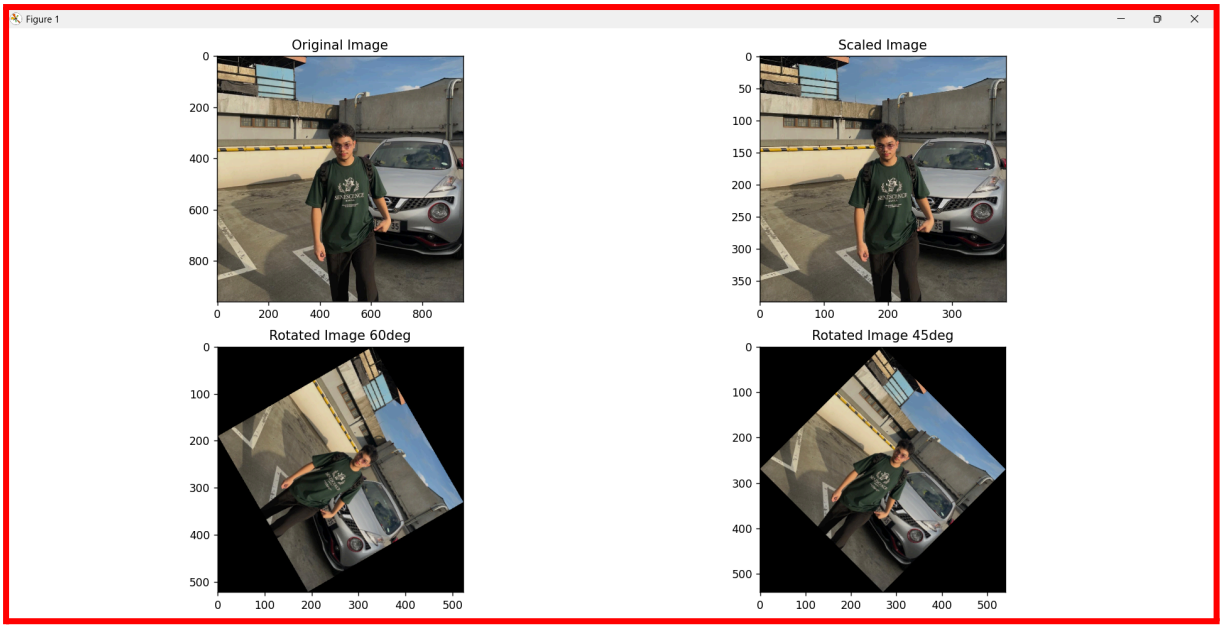

Figure 8. Scaling and Rotation

Source Code

```
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

# Load and display original image
img = Image.open('formalpicturekzy.jpg')

# Calculate new size (50% of original)
new_size = (img.width // 2, img.height // 2)

# Resize using different methods
resized_images = {
    'Original Image': img,
    'Bilinear Image': img.resize(new_size, Image.BILINEAR),
    'Nearest Image': img.resize(new_size, Image.NEAREST),
    'Bicubic Image': img.resize(new_size, Image.BICUBIC)
}
```

Republic of the Philippines
PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Muralla Street, Intramuros
Manila 1002, Philippines

COLLEGE OF ENGINEERING AND TECHNOLOGY
Computer Engineering Department

```
# Create subplot display
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.flatten()

# Display images with proper axis formatting
for i, (title, image) in enumerate(resized_images.items()):
    axes[i].imshow(np.array(image), extent=[0, image.width,
image.height, 0])
    axes[i].set_title(title)

    if title == 'Original Image':
        # Original image - ticks every 100 pixels
        axes[i].set_xticks(np.arange(100, image.width + 100, 100))
        axes[i].set_yticks(np.arange(100, image.height + 100, 100))
    else:
        # Resized images - ticks every 500 pixels
        max_x = max(2500, image.width * 2)  # Ensure we show up to
2500
        max_y = max(2000, image.height * 2)  # Ensure we show up to
2000
        axes[i].set_xticks(np.arange(500, max_x + 500, 500))
        axes[i].set_yticks(np.arange(500, max_y + 500, 500))

    # Set axis limits to match image bounds
    axes[i].set_xlim(0, image.width)
    axes[i].set_ylim(image.height, 0)  # Invert Y-axis to match image
coordinates

    # Style the axes
    axes[i].grid(False)
    axes[i].tick_params(labelsize=8)

plt.tight_layout()
plt.show()
```

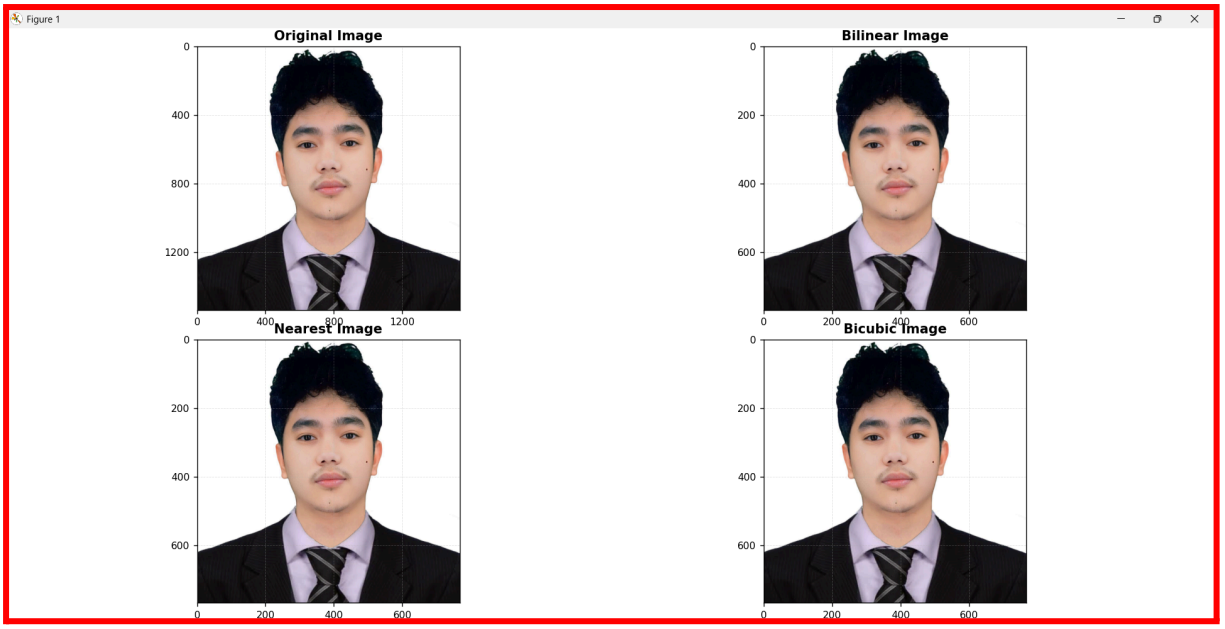

Figure 9. Image Resizing Comparison