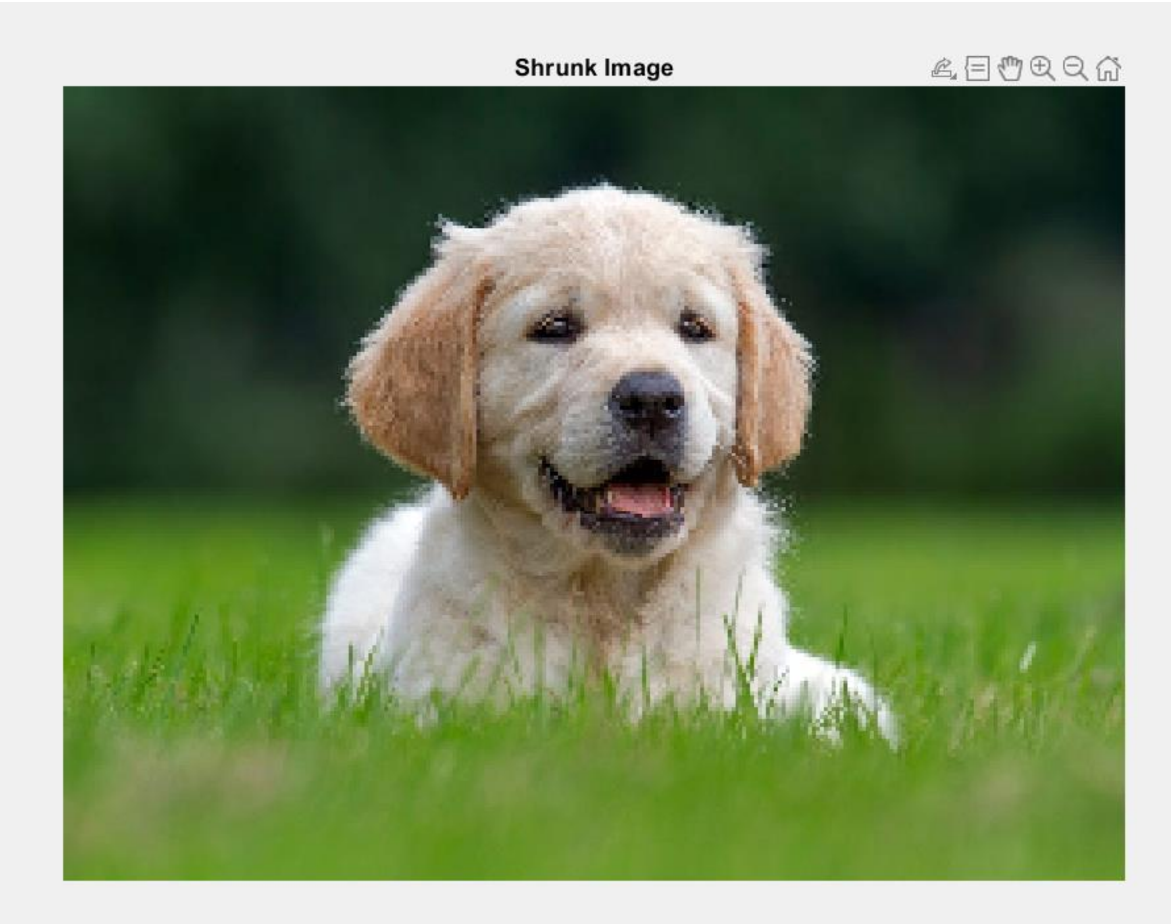
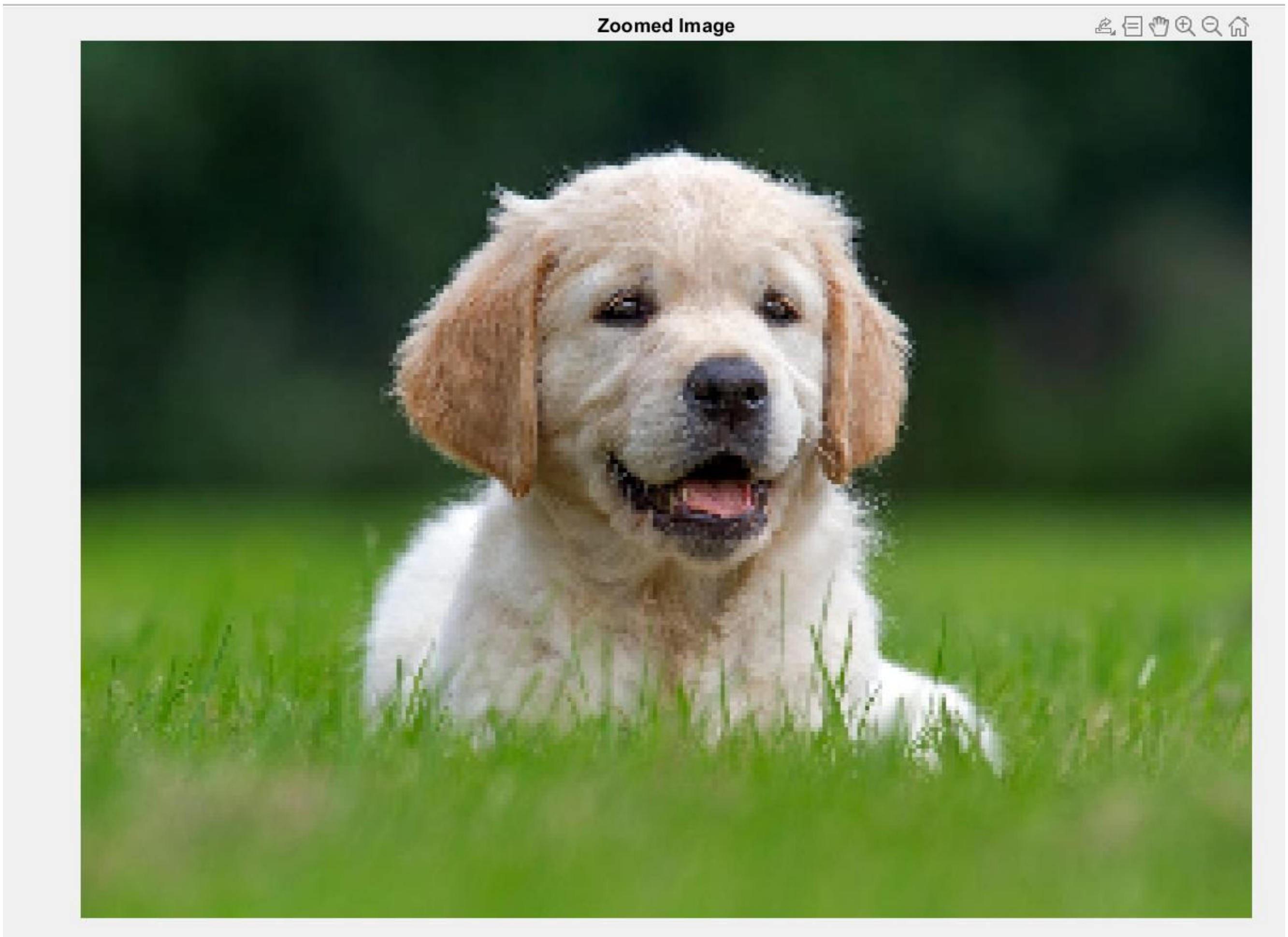
1) **Problem 1**: Images Scaling by Pixel Replication

```
% Load the image
originalImage = imread('ImgA.jpg'); % Replace with the
actual image file path

% (a) Shrink the image by a factor of 4 in each dimension
shrinkFactor = 4;
shrunkImage = customImageScale(originalImage,
shrinkFactor);

% Display the shrunk image
imshow(shrunkImage);
title('Shrunk Image');

% (b) Zoom the image back to its original size
zoomFactor = 1 / shrinkFactor;
zoomedImage = customImageScale(shrunkImage, zoomFactor);

% Display the zoomed image
figure;
imshow(zoomedImage);
title('Zoomed Image');



function scaledImage = customImageScale(inputImage,
scaleFactor)
    % Get the dimensions of the input image
    [height, width, channels] = size(inputImage);

    % Calculate the new dimensions for the scaled image
    newHeight = floor(height / scaleFactor);
    newWidth = floor(width / scaleFactor);

    % Initialize the scaled image matrix
    scaledImage = zeros(newHeight, newWidth, channels,
'uint8'); % Assuming it's a uint8 image

    % Loop through the scaled image and replicate or
decimate pixels
    for i = 1:newHeight
        for j = 1:newWidth
            % Calculate the corresponding pixel
coordinates in the input image
            origX = round(i * scaleFactor);
            origY = round(j * scaleFactor);
```

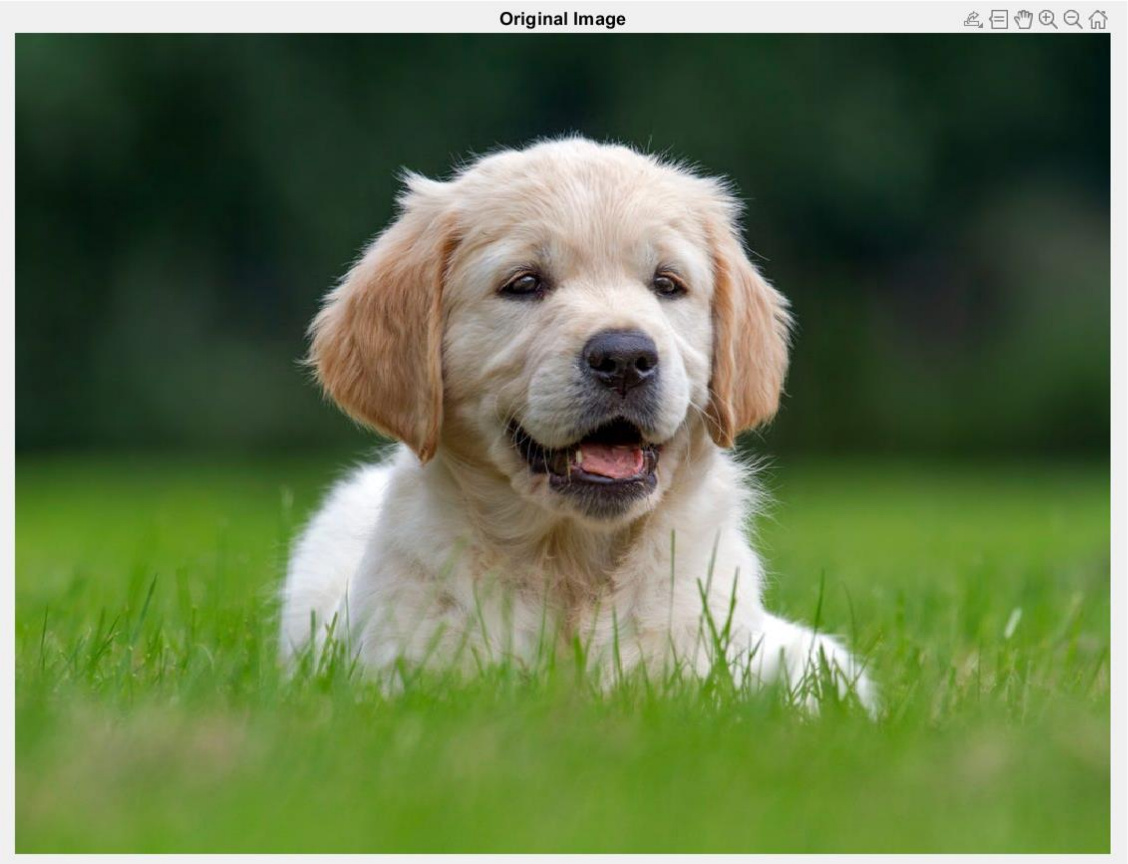
```
            % Ensure the coordinates are within the input
image boundaries
            origX = max(1, min(origX, height));
            origY = max(1, min(origY, width));

            % Copy the pixel value to the scaled image
            scaledImage(i, j, :) = inputImage(origX,
origY, :);
        end
    end
end
```



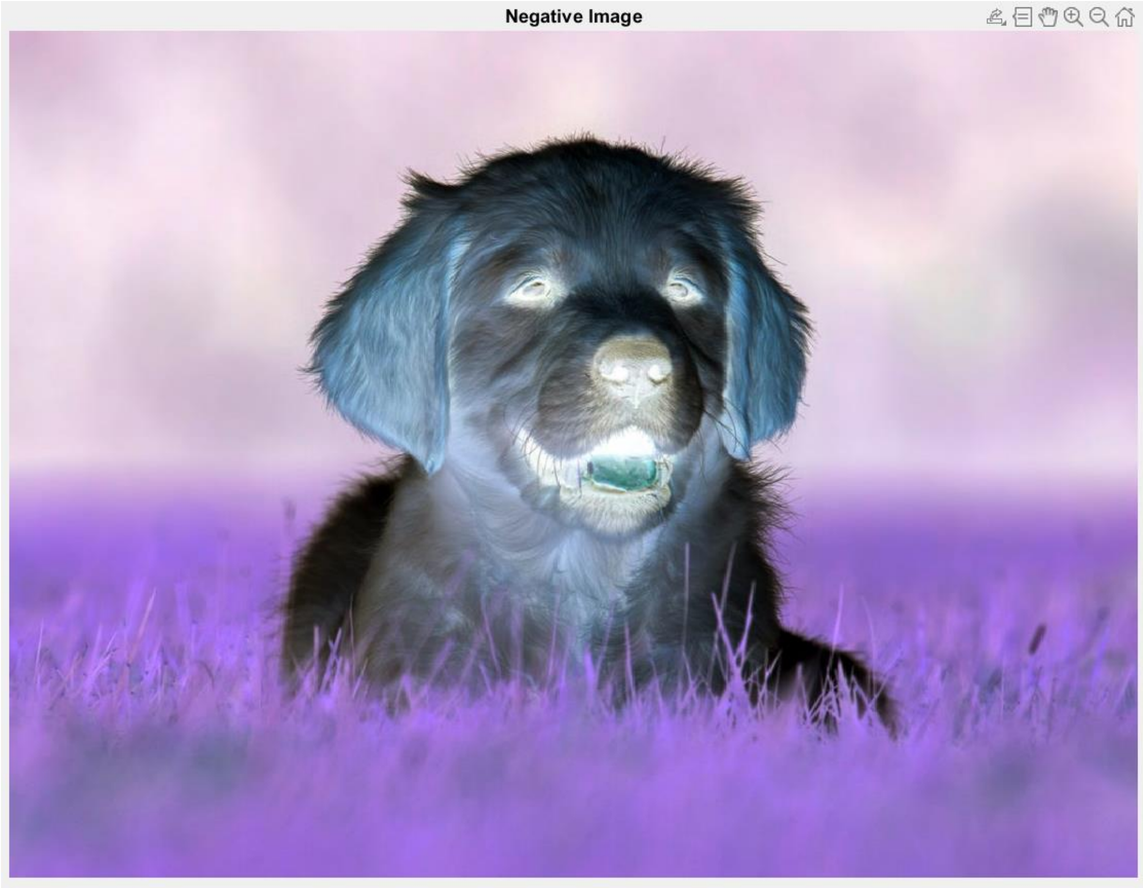



**2)** a) Read and display an image

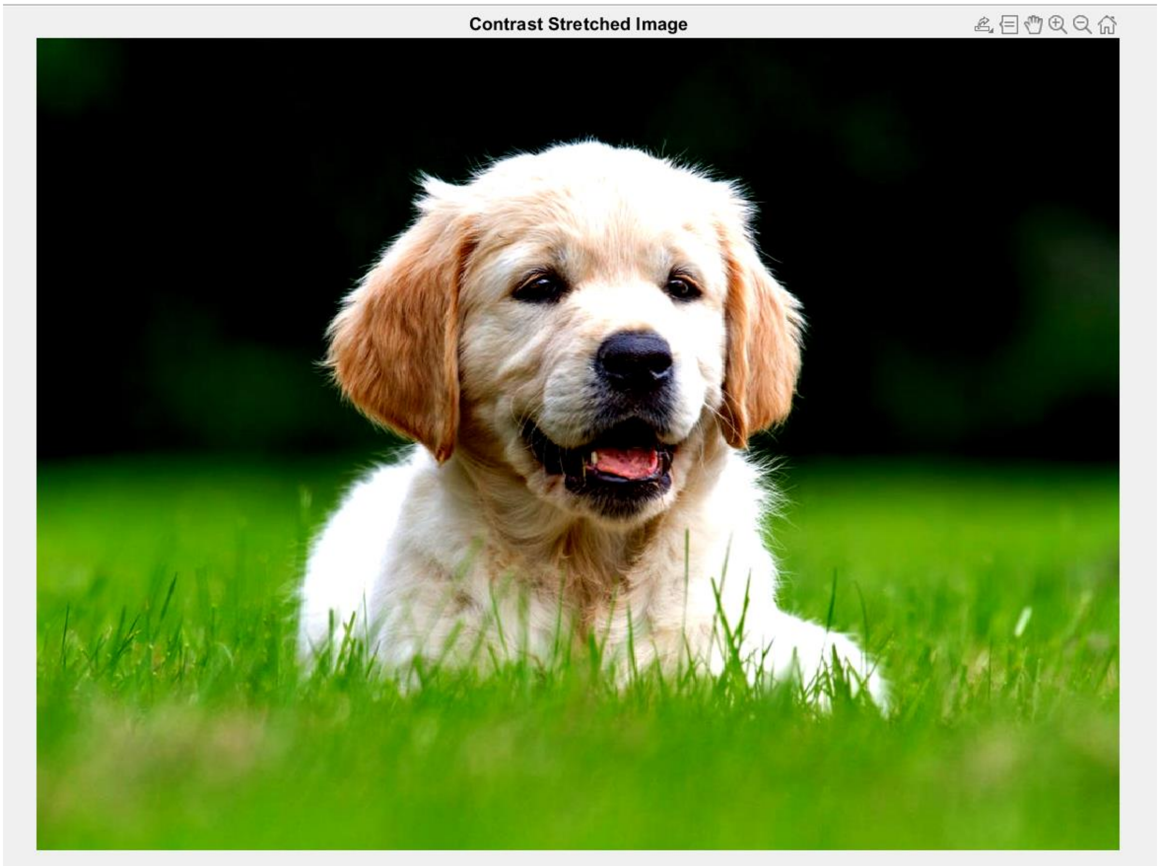
```
originalImage = imread('ImgA.jpg'); % Replace
'input_image.jpg' with your image file

% Create a figure
figure;

% Display the original image
subplot(2, 2, [1, 2]);
imshow(originalImage);
title('Original Image');
```

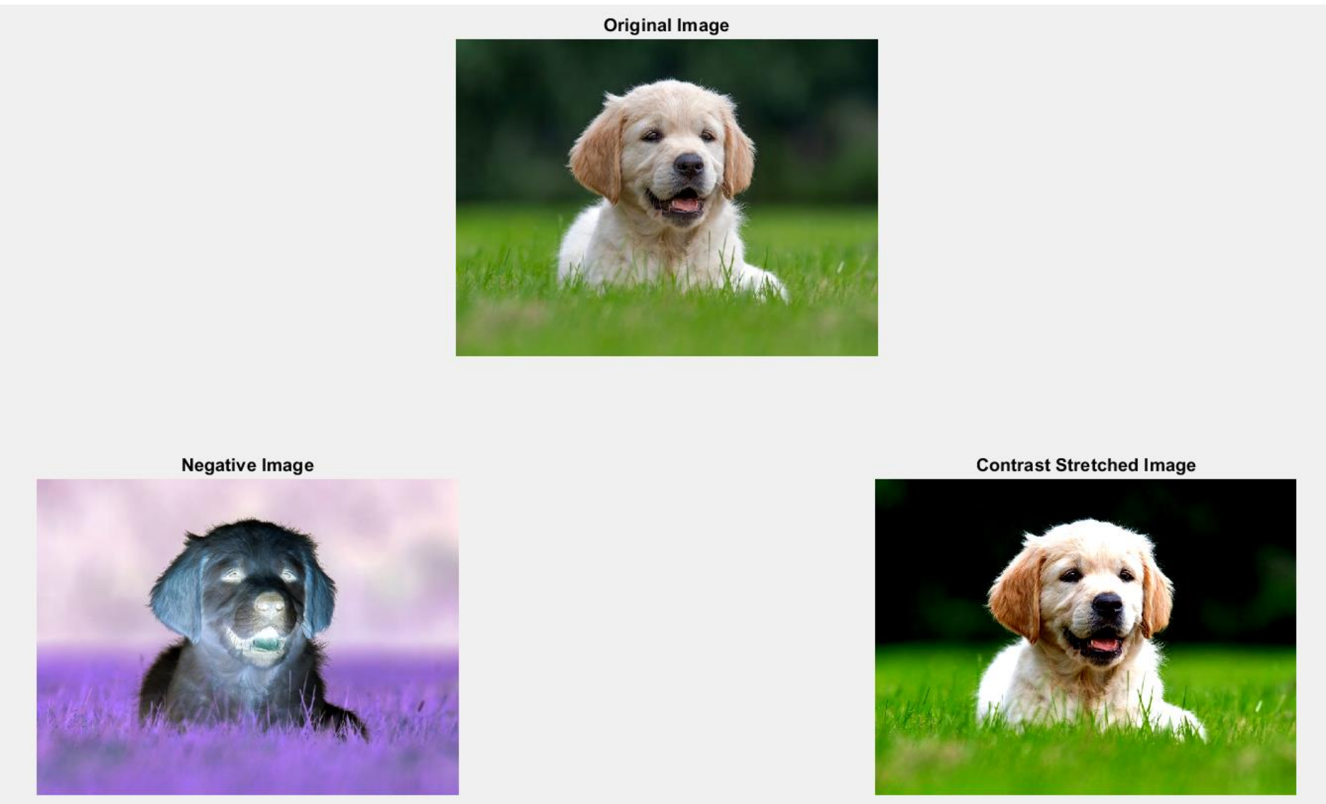



```
(b) Calculate the negative of the image and display it
negativeImage = 255 - originalImage; % Invert pixel
values
subplot(2, 2, 3);
imshow(negativeImage);
title('Negative Image');
```




```
c)Define the contrast stretching parameters

lowIn = 0.2;   % Input range minimum (adjust as needed)
highIn = 0.8;  % Input range maximum (adjust as needed)
lowOut = 0;    % Output range minimum (typically 0)
highOut = 1;   % Output range maximum (typically 1)

% Perform contrast stretching
stretchedImage = imadjust(originalImage, [lowIn, highIn],
[lowOut, highOut]);

% Display the stretched image
subplot(2, 2, 4);
imshow(stretchedImage);
title('Contrast Stretched Image');
```


Contrast Stretched Image


Original Image
Negative Image
Contrast Stretched Image

**3)**

```
% Load the image
originalImage = imread('ImgA.jpg'); % Replace with the
actual image file path

% Rotate the image by 30 degrees
angleDegrees = 30;
rotatedImage = rotateImage(originalImage, angleDegrees);

% Display the rotated image
imshow(rotatedImage);
title('Rotated Image');



function rotatedImage = rotateImage(inputImage,
angleDegrees)
    % Convert the angle from degrees to radians
    angleRadians = deg2rad(angleDegrees);

    % Get the dimensions of the input image
    [height, width, channels] = size(inputImage);

    % Calculate the center point for rotation
    centerX = width / 2;
    centerY = height / 2;

    % Create a transformation matrix for rotation
    rotationMatrix = [cos(angleRadians) -
sin(angleRadians) 0;
                      sin(angleRadians)
cos(angleRadians) 0;
                      0                 0
1];

    % Initialize the rotated image matrix
    rotatedImage = zeros(height, width, channels,
'uint8');

    % Loop through the rotated image and apply the
transformation
    for i = 1:height
        for j = 1:width
```

```
                % Apply the inverse transformation to find
the corresponding pixel in the original image
                transformedPoint = [j - centerX; i - centerY;
1];
                originalPoint = rotationMatrix \
transformedPoint;

                % Interpolate the pixel value from the
original image
                x = originalPoint(1) + centerX;
                y = originalPoint(2) + centerY;

                % Check if the pixel coordinates are within
bounds
                if x >= 1 && x <= width && y >= 1 && y <=
height
                    % Perform bilinear interpolation
                    x1 = floor(x);
                    x2 = min(ceil(x), width);
                    y1 = floor(y);
                    y2 = min(ceil(y), height);

                    % Calculate interpolation weights
                    wx2 = x - x1;
                    wx1 = 1 - wx2;
                    wy2 = y - y1;
                    wy1 = 1 - wy2;

                    % Interpolate pixel values
                    for c = 1:channels
                        pixelValue = wx1 * wy1 *
double(inputImage(y1, x1, c)) + ...
                                     wx2 * wy1 *
double(inputImage(y1, x2, c)) + ...
                                     wx1 * wy2 *
double(inputImage(y2, x1, c)) + ...
                                     wx2 * wy2 *
double(inputImage(y2, x2, c));
                        rotatedImage(i, j, c) =
uint8(pixelValue);
                    end
                end
            end
        end
end
```

**Rotated Image**

## Problem 4

$V = \{0, 1\}$

a)

| | | | | |
|---|---|---|---|---|
| 3 | 1 | 2 | 1 | (q) |
| 2 | 2 | 0 | 2 | |
| 1 | 2 | 1 | 1 | |
| 1 | 0 | 1 | 2 | |

4- path is not possible;

Since we can't move diagonally &

the path ends at (2,3) value = 0

8-path

| | | | | | |
|---|---|---|---|---|---|
| | 3 | 1 | 2 | 1 | (q) |
| | 2 | 2 | 0 | 2 | |
| | 1 | 2 | 1 | 1 | |
| (p) | 1 | 0 | 1 | 2 | |

path length = $\underline{\underline{4}}$

b) Co-ordinates of $p(x,y) = (0,0)$

Co-ordinates of $q(s,t) = (3,3)$

i) $D_4(p,q) = |x-3| + |y-t|$

$= |0-3| + |0-3|$

$= 3+3$

$= \underline{6 \text{ units}}$

ii) $D_8(p,q) = \max(|x-s|, |y-t|)$

$= \max(|0-3|\ |0-3|)$

$= \max(3,3)$

$= \underline{3 \text{ units}}$

The D4 & D8 distances are solely determined by number of steps in horizontal & vertical directions, and they do not consider the specific pattern of movements taken as long as they are within the valid grid directions.