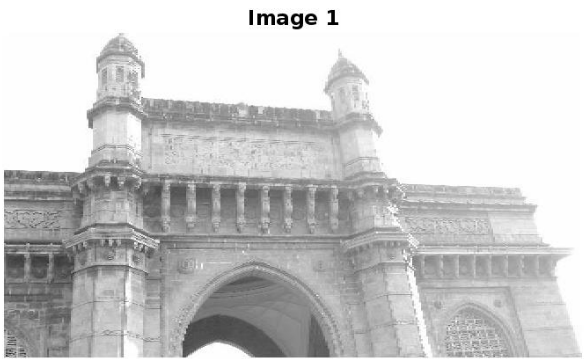
PROBLEM 6(a,b) :FEATURE POINT SELECTION

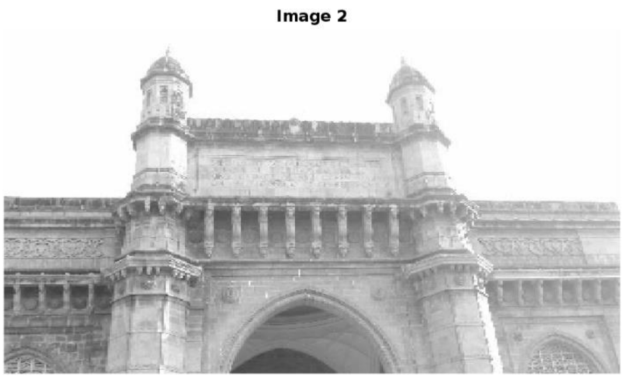
CODE:

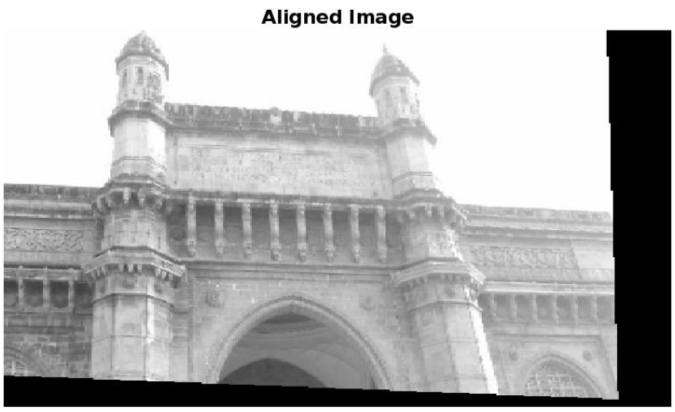
```
% Load original images
im1 = imread('goi1.jpg'); % Original image 1
im2 = imread('goi2.jpg'); % Original image 2

% Convert images to double precision
im1 = double(im1);
im2 = double(im2);

% Initialize arrays to store the points
x1 = zeros(1, 12);
y1 = zeros(1, 12);
x2 = zeros(1, 12);
y2 = zeros(1, 12);

% Manually select 12 pairs of corresponding points
for i = 1:12
    % Select point from the first image
    figure(1);
    imshow(im1 / 255);
    title(sprintf('Select point %d from Image 1', i));
    [x1(i), y1(i)] = ginput(1);

    % Select corresponding point from the second image
    figure(2);
    imshow(im2 / 255);
    title(sprintf('Select corresponding point %d from Image 2', i));
    [x2(i), y2(i)] = ginput(1);
end

% Prepare new images for annotation
new_image1 = im1;
new_image2 = im2;

% Annotate new_image1 with control points
figure;
imshow(new_image1 / 255); % Display image with normalization
title('Control Points on New Image 1');
hold on;
for i = 1:length(x1)
    plot(x1(i), y1(i), 'ro', 'MarkerSize', 8, 'LineWidth', 2);
    text(x1(i), y1(i), num2str(i), 'Color', 'yellow', 'FontSize', 12, 'FontWeight', 'bold');
end
hold off;

% Annotate new_image2 with control points
figure;
imshow(new_image2 / 255); % Display image with normalization
title('Control Points on New Image 2');
hold on;
for i = 1:length(x2)
    plot(x2(i), y2(i), 'go', 'MarkerSize', 8, 'LineWidth', 2);
    text(x2(i), y2(i), num2str(i), 'Color', 'cyan', 'FontSize', 12, 'FontWeight', 'bold');
end
hold off;

% Prepare the points for the affine transformation estimation
movingPoints = [x2', y2']; % Points from the second image
fixedPoints = [x1', y1'];  % Corresponding points from the first image

% Estimate the affine transformation matrix
tform = fitgeotrans(movingPoints, fixedPoints, 'affine');

% Extract the affine transformation matrix
problem6_a_matrix = tform.T; % 3x3 affine transformation matrix


% Display the matrix
disp('Affine Transformation Matrix (problem6_a_matrix):');
disp(problem6_a_matrix);

% Apply the transformation to align the second image with the first
```

```
alignedImage = imwarp(im2 / 255, tform, 'OutputView', imref2d(size(im1)));

% Display the aligned image
figure;
imshow(alignedImage);
title('Aligned Image');

% Overlay the aligned image with the first image to visualize the transformation
figure;
imshowpair(im1 / 255, alignedImage, 'blend');
title('Overlay of Image 1 and Aligned Image 2');
```

RESULT(A,B):







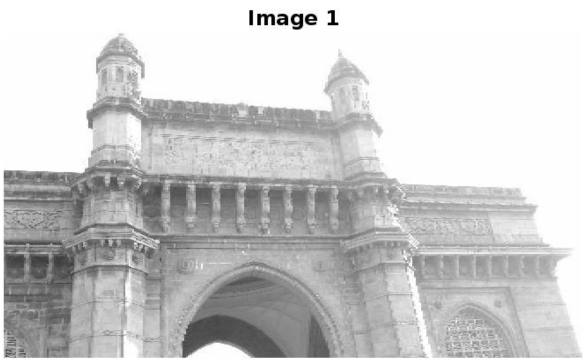
PROBLEM 6(C) : NEAREST NEIGHBOR INTERPOLATION

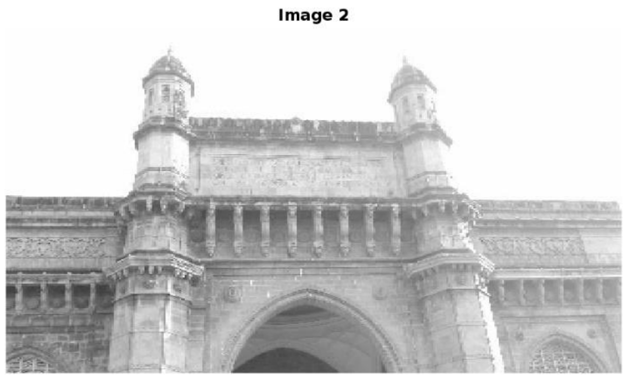
CODE:

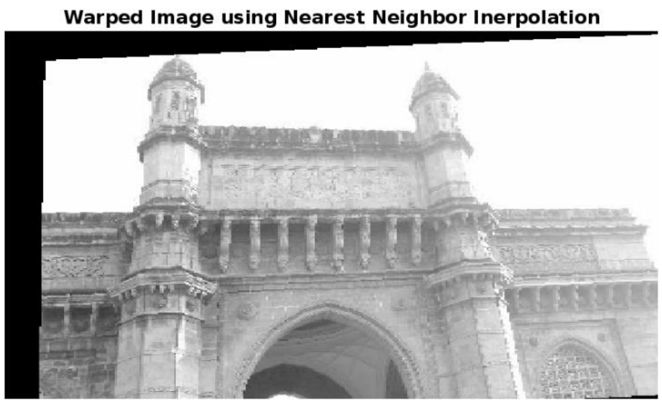
```
% Load images
im1 = imread('goi1.jpg');
im2 = imread('goi2.jpg');

% Convert images to double precision
im1 = double(im1);
im2 = double(im2);

% Assume affine_matrix is the affine transformation matrix obtained from control points
% Replace with your actual matrix
% affine_matrix = [Your generated matrix here];

% Invert the affine transformation matrix for backward warping
inv_matrix = inv(problem6_a_matrix);
```

```
% Get the dimensions of the second image (output image)
[rows, cols, channels] = size(im2);

% Initialize the output (warped) image
warped_image = zeros(rows, cols, channels);

% Perform nearest-neighbor interpolation
for i = 1:rows
    for j = 1:cols
        % Apply the inverse affine transformation to map output (warped) coordinates to input
(im1) coordinates
        original_coords = inv_matrix * [j; i; 1];

        % Extract the original x and y coordinates
        x_orig = round(original_coords(1));
        y_orig = round(original_coords(2));

        % Check if the original coordinates are within the bounds of the first image
        if x_orig >= 1 && x_orig <= size(im1, 2) && y_orig >= 1 && y_orig <= size(im1, 1)
            % Assign the pixel value from the original image to the warped image
            warped_image(i, j, :) = im1(y_orig, x_orig, :);
        end
    end
end

% Display the warped image
figure;
imshow(warped_image / 255);
title('Warped Image using Nearest Neighbor Interpolation');

% Overlay the warped image with the second image for comparison
figure;
imshowpair(im2 / 255, warped_image / 255, 'blend');
title('Overlay of Image 2 and Warped Image 1 with Nearest Neighbor Interpolation')
```

RESULT(C):





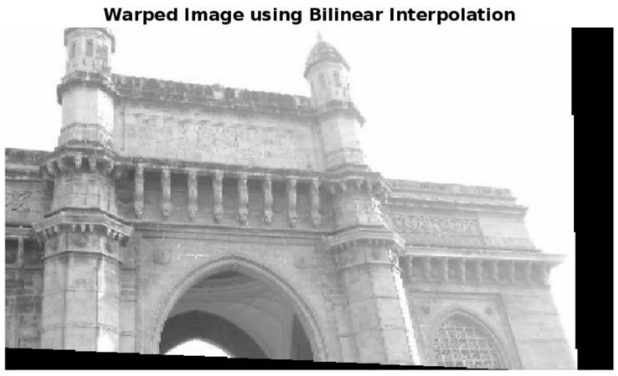


PROBLEM 6(D) : BILINEAR INTERPOLATION

CODE:

```
% Load images
im1 = imread('goi1.jpg');
im2 = imread('goi2.jpg');

% Convert images to double precision
im1 = double(im1);
im2 = double(im2);

% Assume problem6_a_matrix is already generated from the previous step
% Here it's just a placeholder; replace it with your actual matrix
% problem6_a_matrix = [Your generated matrix here];

% Get the dimensions of the second image (output image)
[rows, cols, channels] = size(im2);

% Initialize the output (warped) image
warped_image = zeros(rows, cols, channels);

% Invert the affine transformation matrix for backward warping
inv_matrix = inv(problem6_a_matrix);

% Perform bilinear interpolation
for i = 1:rows
    for j = 1:cols
        % Apply the inverse affine transformation to map output (warped) coordinates to input
(im1) coordinates
        original_coords = inv_matrix * [j; i; 1];

        % Extract the original x and y coordinates
        x_orig = original_coords(1);
        y_orig = original_coords(2);

        % Find the integer coordinates of the four nearest neighbors
        x1 = floor(x_orig);
        y1 = floor(y_orig);
        x2 = ceil(x_orig);
        y2 = ceil(y_orig);

        % Check if the coordinates are within the bounds of the image
        if x1 >= 1 && x1 < size(im1, 2) && y1 >= 1 && y1 < size(im1, 1)
            % Get pixel values for the four neighboring pixels
            Q11 = im1(y1, x1, :);
            Q12 = im1(y1, x2, :);
            Q21 = im1(y2, x1, :);
            Q22 = im1(y2, x2, :);

            % Compute weights for bilinear interpolation
            alpha = x_orig - x1;
            beta = y_orig - y1;

            % Compute the interpolated value
            top = (1 - alpha) * Q11 + alpha * Q12;
            bottom = (1 - alpha) * Q21 + alpha * Q22;
            warped_image(i, j, :) = (1 - beta) * top + beta * bottom;
        end
    end
end

% Display the warped image
figure;
imshow(warped_image / 255);
title('Warped Image using Bilinear Interpolation');

% Overlay the warped image with the second image for comparison
figure;
imshowpair(im2 / 255, warped_image / 255, 'blend');
title('Overlay of Image 2 and Warped Image 1 with Bilinear Interpolation');
```

RESULT (D):



PROBLEM 6(E):

What Happens When Points Are Collinear?

- Non-Uniqueness: An affine transformation should correspond to translation, rotation, scaling, and shearing between two images. For an accurate estimation of this transformation, the points you select should cover different parts of the image and span the plane 2D; that is, all of them should not lie on the same line. Collinear points do not give enough information to uniquely describe the transformation. As such, this can result in a very inaccurate matrix or even a non-unique one.
- Low Dimensionality: Affine transformations deal in 2D space, meaning that the points chosen should ideally give information in both the x and y directions. Collinear points only vary along one dimension, hence effectively making the problem a 1D case. This can then cause the equations used to calculate the affine transformation matrix to become dependent on each other; that is, a situation known as degeneracy. In other words, this means that there will not exist a unique solution to the system of equations, and no correct transformation can be determined.
- Poor Estimation: Even if an affine transformation matrix is computed from collinear points, it may not correctly estimate the transformation between images. Such a matrix will still be good at doing simple operations like translation or scaling but will not capture correctly more complex transformations, and large errors will show up when the transformation is applied.