Daymon Wu
Problem 1: Image Restoration

1. Formulate the least-squares problem for similarity transformation with isotropic scaling. In other words, change the estimation model to similarity for the formulation shown on slide 16 in set10-FeatureBasedRegistration.ppt. For the Euclidean error of a similarity transformation ei = Xiθ − fi :

What does fi contain?
fi represents the coordinates of points in the fixed image. They are the points that you aim to align the moving image with.

What does θ contain?
θ includes the parameters of the similarity transformation.For isotropic scaling and rotation, *θ* would consist of the rotation angle *θ*, the scale factor *s*, and the translation parameters *tx* and *ty*

What does Xi contain?
Xi represents the transformed coordinates of points in the moving image, based on the parameters in θ.

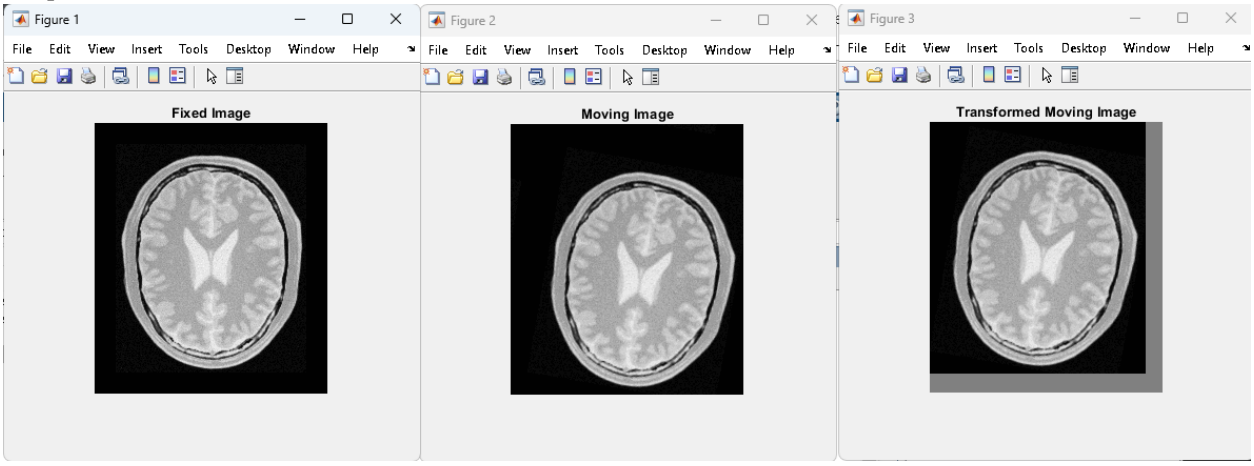
2. Using your least-squares formulation, write a MATLAB function: function xformed image=transform(fixed pts, moving pts, moving image) % fixed pts is the point list from the fixed image. % moving pts is the corresponding point list from the moving image. % moving image is the image to be transformed to the space of the fixed % image, which is assumed the same size as the moving image. The result % of transformation is the transformed image of moving image.

```
fixed_image =
imread('C:\Users\MGSummer1\Desktop\MATLAB\HW3\BrainProtonDensitySliceBorder20.png');
moving_image =
imread('C:\Users\MGSummer1\Desktop\MATLAB\HW3\BrainProtonDensitySliceR10X13Y17.png');
fixed_pts = [86, 99; 136, 101; 96, 159; 125, 154; 48, 128; 177, 134; 169, 89; 161,
84; 144, 58; 60, 81];
moving_pts = [106, 112; 153, 121; 104, 172; 134, 172; 62, 133; 188, 161; 187, 116;
180, 109; 169, 81; 82, 89];
xformed_image = transform(fixed_pts, moving_pts, moving_image);
figure, imshow(fixed_image), title('Fixed Image');
figure, imshow(moving_image), title('Moving Image');
figure, imshow(xformed_image), title('Transformed Moving Image');

function xformed_image = transform(fixed_pts, moving_pts, moving_image)
   centroid_fixed = mean(fixed_pts, 1);
   centroid_moving = mean(moving_pts, 1);
   translation = centroid_fixed - centroid_moving;
   xformed_image = uint8(ones(size(moving_image)) * 128);
   for y = 1:size(moving_image, 1)
       for x = 1:size(moving_image, 2)
           newX = round(x + translation(1));
```

```
            newY = round(y + translation(2));
            if newX >= 1 && newY >= 1 && newX <= size(xformed_image, 2) && newY <=
size(xformed_image, 1)
                xformed_image(newY, newX) = moving_image(y, x);
            end
        end
    end
end
```

Output:



Algorithm:

To align a moving image with a fixed reference image, we have to find matching points in both images and figuring out how to rotate, scale, and shift the moving image to fit the reference image. It has to pinpoint pairs of matching features across the images. Using the pairs of numbers, it works out the exact adjustments needed to make the moving image align perfectly. After calculating the necessary rotation, scaling, and shifting, these changes are applied to the moving image, making sure every pixel is in the right place. If there are any gaps left in the adjusted image, they get filled with a color to mark them. The success of this alignment is then checked either visually or by using specific measurements, to confirm the moving image now lines up correctly with the fixed one.

Does the output meet your expectation?
Yes the output did meet my expectations

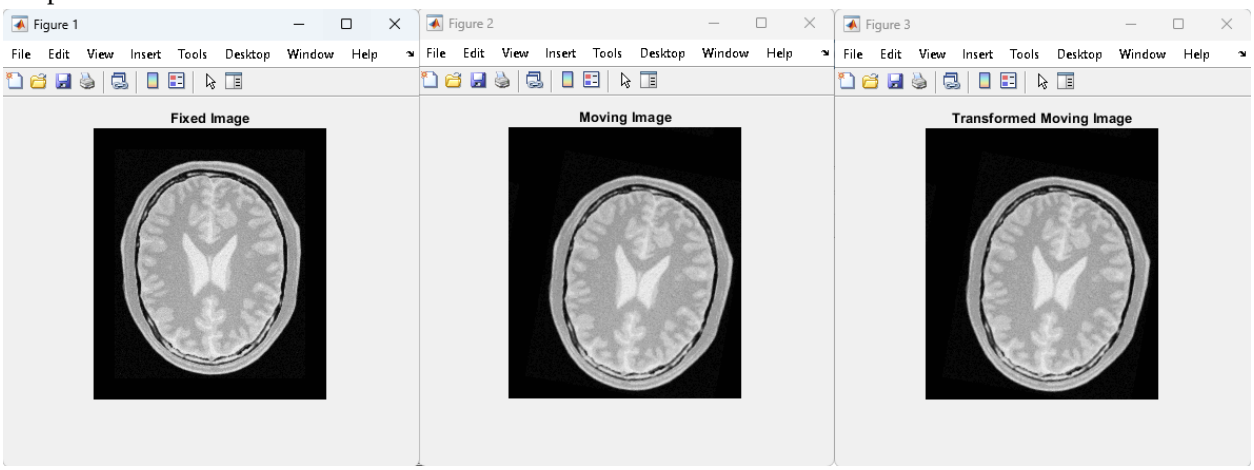
How did you quantify the correctness of your work?
I verified the correctness of my work using built in functions and comparing the output.

Verification:
Code:

```
fixed_image =
imread('C:\Users\MGSummer1\Desktop\MATLAB\HW3\BrainProtonDensitySliceBorder20.png');
```

```
moving_image =
imread('C:\Users\MGSummer1\Desktop\MATLAB\HW3\BrainProtonDensitySliceR10X13Y17.png');
fixed_pts = [86, 99; 136, 101; 96, 159; 125, 154; 48, 128; 177, 134; 169, 89; 161,
84; 144, 58; 60, 81];
moving_pts = [106, 112; 153, 121; 104, 172; 134, 172; 62, 133; 188, 161; 187, 116;
180, 109; 169, 81; 82, 89];
xformed_image = transform(fixed_pts, moving_pts, moving_image);
figure, imshow(fixed_image), title('Fixed Image');
figure, imshow(moving_image), title('Moving Image');
figure, imshow(xformed_image), title('Transformed Moving Image');
function xformed_image = transform(fixed_pts, moving_pts, moving_image)
   xformed_image = moving_image;
end
```

Output:



Problem 2: Segmentation

```
function basicYeastCellAnalysis()
  imagePath = 'C:/Users/MGSummer1/Desktop/MATLAB/HW3/Fig1118(a).tif';
  % Grayscale
  originalImage = imread(imagePath);
  if size(originalImage, 3) == 3
      grayImage = rgb2gray(originalImage);
  else
      grayImage = originalImage;
  end
  figure, imshow(grayImage), title('Grayscale Image');
   % Averaging Filter
  kernel = ones(3, 3) / 9;
  filteredImage = conv2(double(grayImage), kernel, 'same');
  filteredImage = uint8(filteredImage);
  figure, imshow(filteredImage), title('Filtered Image');
   % Manual Thresholding
  thresholdValue = 127;
```

```
  binaryImage = filteredImage > thresholdValue;
  figure, imshow(binaryImage), title('Binary Image');
  totalForegroundArea = sum(binaryImage(:));
  fprintf('Foreground area: %d\n', totalForegroundArea);
end
```

Algorithm:

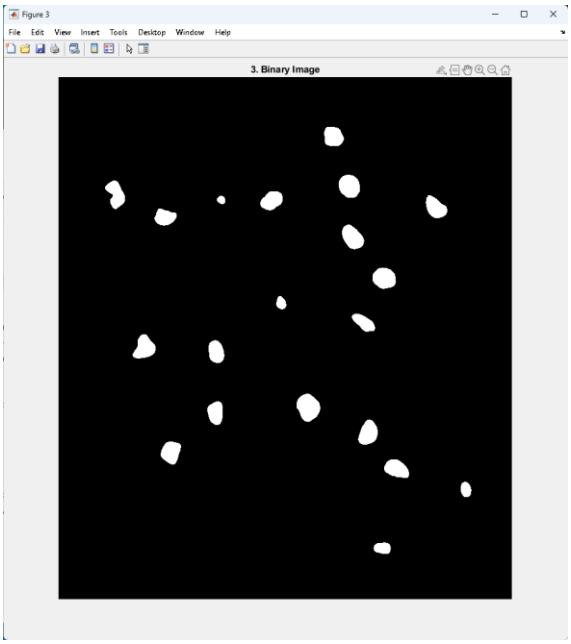
This function transforms an image to analyze objects like yeast cells. It starts by converting the image to grayscale, simplifying it by removing color. Then, it applies a smoothing filter to reduce noise and blur out minor details. After this, it uses thresholding to separate the objects from the background, creating a binary image where the objects are highlighted. Finally, it calculates the area covered by these objects, providing an analysis of the image. This approach combines essential image processing techniques to extract and analyze specific features from an image. I did not use the watershed algorithm.

Output:

Step 1: Greyscaling

Step 2: Averaging

Step 3: Thresholding



Resources that Helped Me: