EBME 361
Jamie Yu
02/28/18

Homework 5: Morphology (Mini)

1. I solved this problem programmatically in MATLAB. I created three structuring elements by constructing 3x3 matrices, setting the empty units to zero and filled units to one. I then applied the dilation operation using the imdilate() function, first for $A \oplus B$. Below is a visual of the matrices created and produced:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$A \oplus B = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Next, I followed the same procedure to find $A \oplus B \oplus C$:

$$C = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$A \oplus B \oplus C = \begin{bmatrix} -Inf & 0 & 1 & 1 & 1 & 0 & -Inf \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ -Inf & 0 & 1 & 1 & 1 & 0 & -Inf \end{bmatrix}$$

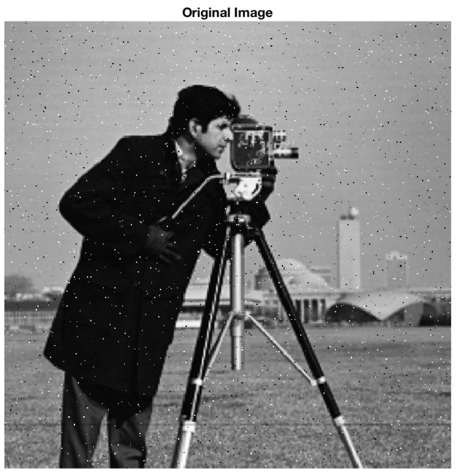
Below is the code used to apply these operations in MATLAB:

```
%% Problem 1

A = [0 1 0; 0 1 0; 0 1 0];
B = [0 0 0; 1 1 1; 0 0 0];
C = [0 1 0; 1 1 1; 0 1 0];

output1 = imdilate(A, B, 'full');
disp(output1);

output2 = imdilate(output1, C, 'full');
disp(output2);
```

   a. The output of same dimensions (3x3) resembles the input structuring element with padding around all edges of the original structuring element. This makes sense, because the dilation operation simply looks for a "hit", and since A and B are perpendicular, hits will occur for each filled unit. Furthermore, the output of same dimensions from $A \oplus B \oplus C$ resembles $A \oplus B$ because once the entire structuring element is filled with ones, it is already fully padded. The basic effect of dilation is enlargement of boundaries of regions of foreground (white or 1-valued) pixels. Since we specified 'full' for the imdilate() operation, the command performs dilation across a theoretically "infinite" plane, which is why we see values of –Inf at the edges where the operation did not occur, and zeros where there were misses and ones outside the 3x3 dimension where there were hits.

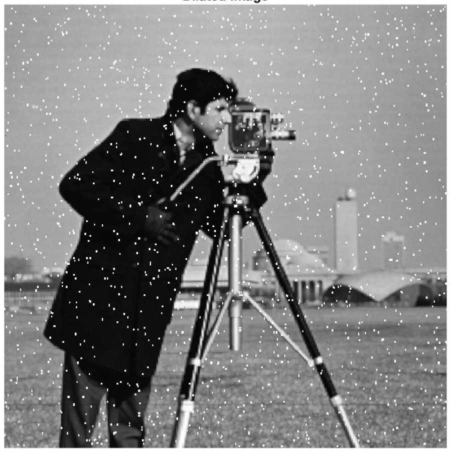
b. This is a useful operation for repairing breaks and intrusions. In image processing applications, this is extremely helpful when we have images with breaks, or unclosed edges. Additionally, it can be useful for removing "pepper" noise to "fill" the holes in the image.

2. The original image is shown below. It contains both salt and pepper noise. The objective of this problem was to investigate the use of dilation, opening, and closing, and understand which works best.



a. Below is the output image after applying dilation to the image. In the image, we see the successful removal of "pepper" noise. However, the "salt" noise has increased in size as a result.

Dilated Image

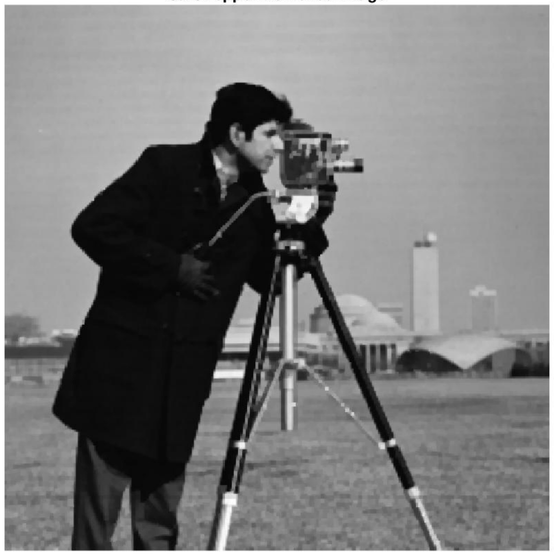


To perform dilation, I created a structuring element using the MATLAB function, strel(). I specified a rectangular shape, and defined the dimensions of the structuring element as 3x2. I then used imdilate() to apply the structuring element to the original image. The code can be found below:

```
%% Problem 2
close all; clear; clc;

% Load the image.
cameraMan = imread('cameraman.png');

fig1 = figure(1);
imshow(cameraMan); title('Original Image');

% a. Create a structuring element to dilate image to remove
pepper noise.
SE_dilate = strel('rectangle', [3, 2]);
dilatedCameraMan = imdilate(cameraMan, SE_dilate);
fig2 = figure(2);
imshow(dilatedCameraMan); title('Dilated Image');
```

b. Below is the output image after applying closing and opening operations to the original image. We are abler to see successful remove of both “salt” and “pepper” noise.

Salt/Pepper Removed Image

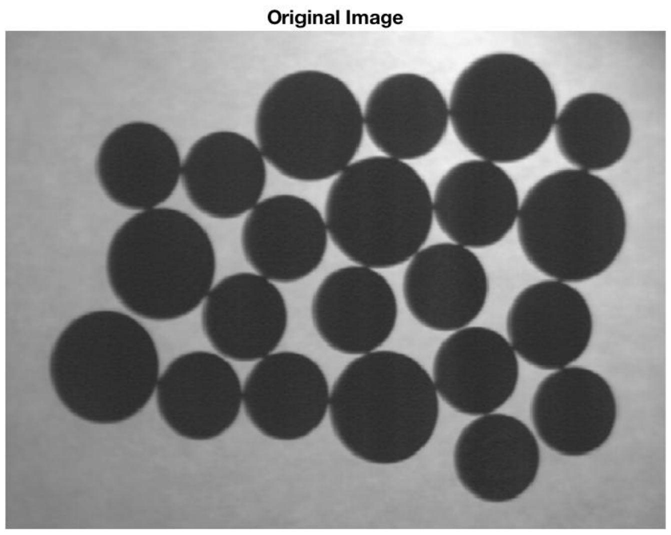


I designed the open-close gray scale morphology operation by first attempting to close and then open the image. However, I found this would not successfully remove the salt noise. I then concluded on performing closing operation followed by opening operation. I created a structuring element of dimensions 4x4, and used the imclose() and imopen() functions. The code can be found below:

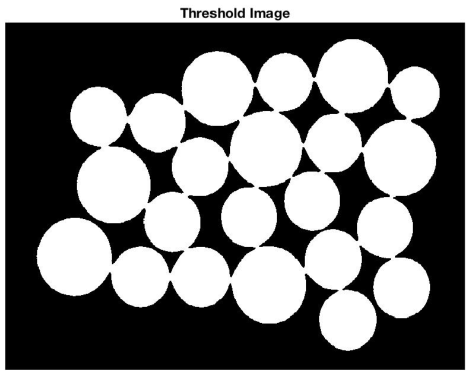
```
% b. Design an open-close gray scale morphology operation to
reduce both
% salt and pepper noise. We perform consecutive closing and
opening
% operations on the image to first remove the pepper (close the
"holes" in
% the image and then the salt (open the "noise").
SE = strel('rectangle', [4, 4]);
closedCameraMan = imclose(cameraMan, SE);
finalCameraMan = imopen(closedCameraMan, SE);
fig3 = figure(3);
imshow(finalCameraMan); title('Salt/Pepper Removed Image');

saveas(fig1, 'hw5_fig1.jpg');
saveas(fig2, 'hw5_fig2.jpg');
saveas(fig3, 'hw5_fig3.jpg');
```

c. From comparing the two images from parts (a) and (b), I can conclude that an open-close operation is much better than simply applying a dilation to an image. This has to do with the fact that dilation distorts regions of pixels indiscriminately. In contrast, opening and closing operations are combinations of erosion *and* dilation, which will reduce some of the distortion effect.

3. This problem involved separating disks against a light background. The original image can be seen below:



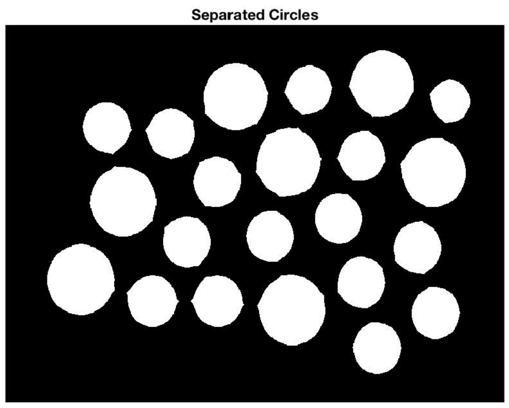
   a. I applied a final threshold value of 90 pixels to the image. I chose this value after playing around with values, and determining this would produce the binary image. Since the image has a light background and dark circles, I found the threshold image by evaluating a logical statement finding the pixels in the original image less than the threshold. The output image and code used can be seen below:



```
%% Problem 3
close all; clear; clc;

% Load the image.
circles = imread('Circles.png');

fig4 = figure(4);
imshow(circles); title('Original Image');

% a. Perform thresholding of the image. First convert rgb image
to gray, then
% perform gray thresholding.
```

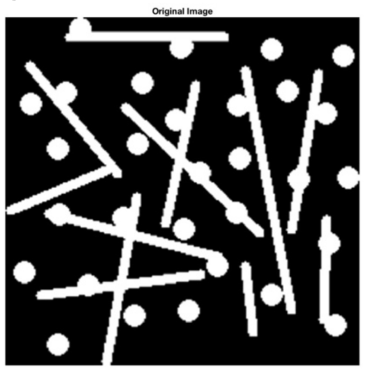
```
grayCircles = rgb2gray(circles);

% Define threshold at pixel value 90 to create a binary image.
threshold = 90;
thresholdCircles = grayCircles < threshold;

fig5 = figure(5);
imshow(thresholdCircles); title('Threshold Image');
```

b. I used the erosion operation, imerode() in MATLAB on the threshold image to separate the disks in the image. I chose a structuring element of disk shape and diameter of 11. The output image and code used can be found below:

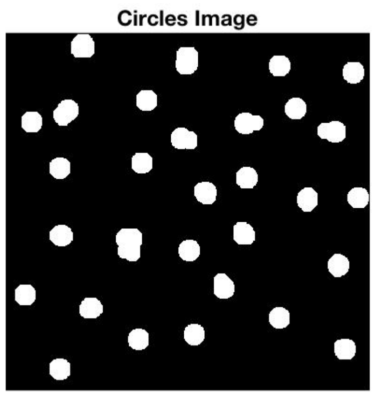


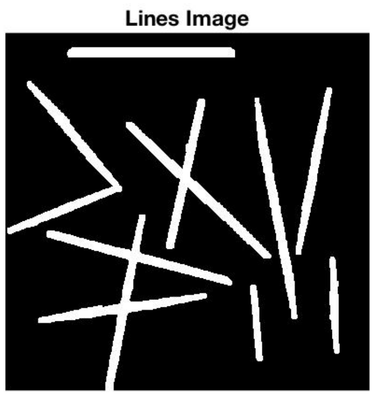
```
% b. Use erosion operation to separate the circles distinctly.
SE_erode = strel('disk', 11);
erodedCircles = imerode(thresholdCircles, SE_erode);

fig6 = figure(6);
imshow(erodedCircles); title('Separated Circles');

saveas(fig4, 'hw5_fig4.jpg');
saveas(fig5, 'hw5_fig5.jpg');
saveas(fig6, 'hw5_fig6.jpg');
```

4. This problem involved separating an image with multiple shapes using the opening operation. The original image can be seen below:



a. I determined the proper opening element size to be 19 pixels and disk-shaped in order to get an image with just circles. The lines-only image was computed using a series of structuring elements. I used seven line-shaped structuring elements of different lengths and angles after playing around with the options. An easy example of using a structuring element to get a line would be the horizontal line at the top of the image. For this line, we would specify a length similar to length found in the picture, and set the angle of rotation to 0. However, it took adjusting to figure out the structuring elements for other lines at offset angles. In the end, I generalized the line segments into a group of horizontal and a group of vertical lines, using an OR operator. The output images and code used can be found below:





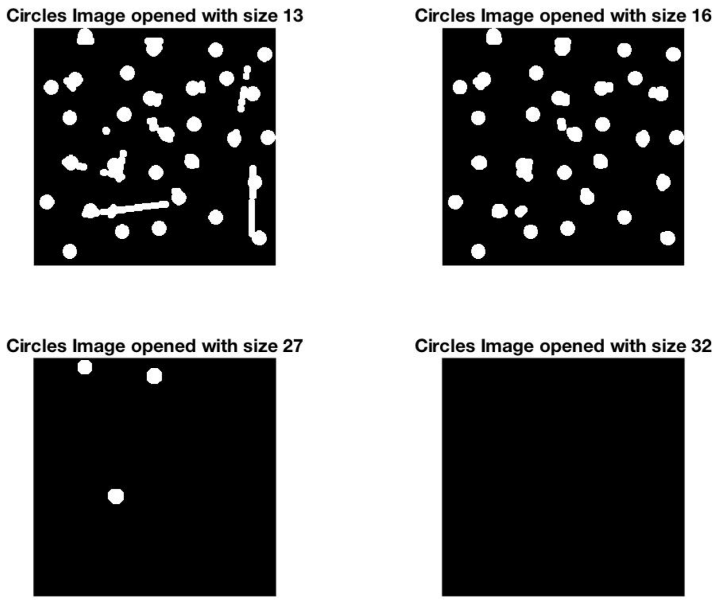
```
%% Problem 4
clear; clc;

% Load the image.
circlesLines = imread('Circle_and_Lines.png');

% Convert to 2D image (gray-scale);
circlesLines = rgb2gray(circlesLines);

% Find the threshold.
circleLevel = graythresh(circlesLines);

% Binarize the image.
circlesLines = imbinarize(circlesLines, circleLevel);

fig7 = figure(7);
imshow(circlesLines); title('Original Image');
saveas(fig7, 'hw5_fig7.jpg');

circlesImage = imopen(circlesLines, strel('disk', 19));
lines1 = imopen(circlesLines, strel('line', 150, 0));
lines2 = imopen(circlesLines, strel('line', 145, 275));
lines3 = imopen(circlesLines, strel('line', 200, 20));
lines4 = imopen(circlesLines, strel('line', 200, 77));
lines5 = imopen(circlesLines, strel('line', 110, 135));
```

```
lines6 = imopen(circlesLines, strel('line', 160, -15));
lines7 = imopen(circlesLines, strel('line', 200, 5));

horizontalLines = lines1 | lines5 | lines6 | lines7;
verticalLines = lines2 | lines3 | lines4;

linesImage = horizontalLines | verticalLines;
linesImage = imopen(linesImage, strel('disk', 7));

fig8 = figure(8);
subplot(1,2,1); imshow(circlesImage);
title('Circles Image');
subplot(1,2,2); imshow(linesImage);
title('Lines Image');
saveas(fig8, 'hw5_fig8.jpg');
```

b. The drawback/limitation of this method is clear to see in the implementation of the lines-only image. It required seven separate structuring elements to get a lines-only image, due to the fact that the lines were all at different angles. While similar-shaped objects such a circles are easy to separate, this method does not work well for asymmetrical shapes. Additionally, we notice slight deformation in the circles-only image.

   This method works well in separating objects in a binary image. In comparison to just dilation or erosion alone, it works much better. While dilation works well in fixing breaks and intrusions, it would not work well in this case because we are trying to separate items. Similarly, erosion would not work well either. While we would be able to use it to separate some circles from lines that are touching, getting the two objects in separate images would be difficult. Opening works best in comparison to dilation and erosion alone.

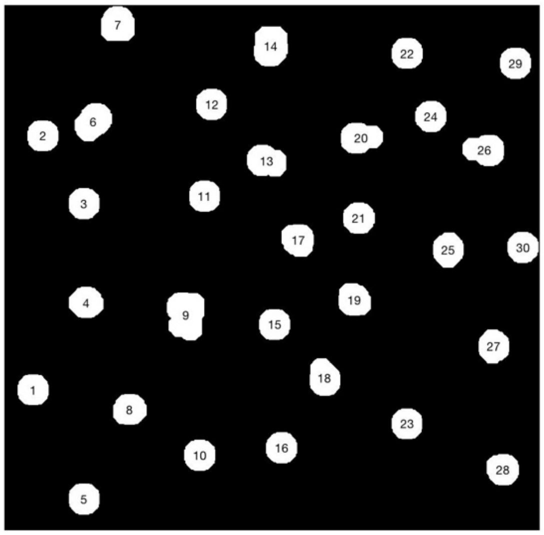
c. I tested various SE sizes ranging from 13 pixels to 32. The code and resulting images can be found below:


Circles Image opened with size 13
Circles Image opened with size 16
Circles Image opened with size 27
Circles Image opened with size 32

```
%% c. Compare different SE sizes.
SE_open = strel('disk', 13);
circlesImage = imopen(circlesLines, SE_open);

fig9 = figure(9);
subplot(2,2,1); imshow(circlesImage);
title('Circles Image opened with size 13');

SE_open = strel('disk', 16);
circlesImage = imopen(circlesLines, SE_open);

fig9;
subplot(2,2,2); imshow(circlesImage);
title('Circles Image opened with size 16');

SE_open = strel('disk', 27);
circlesImage = imopen(circlesLines, SE_open);

fig9;
subplot(2,2,3); imshow(circlesImage);
title('Circles Image opened with size 27');

SE_open = strel('disk', 32);
circlesImage = imopen(circlesLines, SE_open);

fig9;
subplot(2,2,4); imshow(circlesImage);
title('Circles Image opened with size 32');

saveas(fig9, 'hw5_fig9.jpg');
```

In general, the size of the structuring element modifies how well the separation of circles and lines occurs. When the structuring element size is small enough to "fit" inside the lines in the image, the lines will also show up in the circles image. However, when the size is large enough to *not* fit in the lines, but small enough to fit inside the circles, good separation will occur. In contrast, too large of a structuring element will cause no separation to occur. When we reach a pixel size of 32, the circles image is completely black. This indicates that the attempt at opening of the image does not work since the structuring element disc diameter is larger than the diameter of the circles.

d. I used the built-in MATLAB command logical() on the circles-only image to label the elements in the image. This command works in converting the elements of the matrix into logicals, holding a value of either 0 or 1 where distinctive objects exist. Therefore, every pixel containing black (background) is labeled as 0, while every pixel containing white (foreground) is labeled as 1. Using the command regionprops(), I then used the result of the logical array to find the centroid locations of each of the circles. Iterating through this array using a for loop, I then plotted numbers on top of each white circle for each centroid location found. The resulting image and code can be found below:

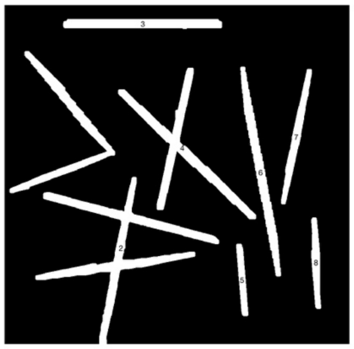


From this algorithm, **I found there to be 30 circles in the image.**

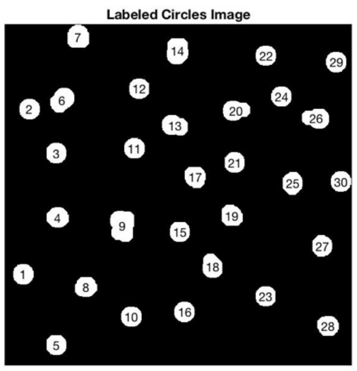
```
% Label the elements in each image.
circleLabels = logical(circlesImage);

s1 = regionprops(circleLabels, 'Centroid');
fig10 = figure(10);
imshow(circlesImage);
hold on
for k = 1:numel(s1)
    c = s1(k).Centroid;
    text(c(1), c(2), sprintf('%d', k), ...
        'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'middle');
```

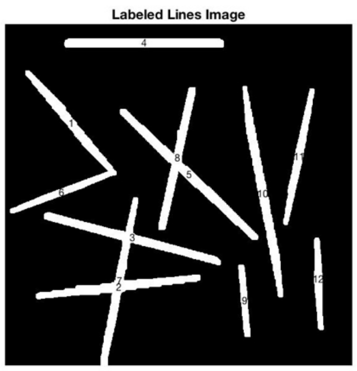
```
end
hold off

fprintf('There are %d circles in the image.\n', numel(s1));
saveas(fig10, 'hw5_fig10.jpg');
```

Similarly, I applied the same algorithm to the lines-only image. However, this was more difficult due to the fact that lines are overlapping. If we were to apply the algorithm directly, the count would be incorrect:



In order to get an accurate labeling and count of lines, I applied the logical() command separately on the horizontal and vertical lines. This way, it would be able to distinguish the overlapping lines objects as separate lines. Next, I performed the same algorithm of finding centroid of line segments, and labeling each line with text. I used two separate for loops to iterate over the vertical and horizontal lines. The output image and code is found below:





From this algorithm, **I found there to be 12 lines in the image.**

```
%% d. Write an algorithm to count the number of items in an
image.
close all; clear; clc;

circlesLines = imread('Circle_and_Lines.png');
circlesLines = rgb2gray(circlesLines);
falconLevel = graythresh(circlesLines);
circlesLines = imbinarize(circlesLines, falconLevel);
```

```
SE_open = strel('disk', 19);
circlesImage = imopen(circlesLines, SE_open);
lines1 = imopen(circlesLines, strel('line', 150, 0));
lines2 = imopen(circlesLines, strel('line', 145, 275));
lines3 = imopen(circlesLines, strel('line', 200, 20));
lines4 = imopen(circlesLines, strel('line', 200, 77));
lines5 = imopen(circlesLines, strel('line', 110, 135));
lines6 = imopen(circlesLines, strel('line', 160, -15));
lines7 = imopen(circlesLines, strel('line', 200, 5));

horizontalLines = lines1 | lines5 | lines6 | lines7;
verticalLines = lines2 | lines3 | lines4;

linesImage = horizontalLines | verticalLines;
linesImage = imopen(linesImage, strel('disk', 7));

% Label the elements in each image.
circleLabels = logical(circlesImage);

s1 = regionprops(circleLabels, 'Centroid');
fig10 = figure(10);
subplot(1,2,1); imshow(circlesImage); title('Labeled Circles
Image');
hold on
for j = 1:numel(s1)
    c = s1(j).Centroid;
    text(c(1), c(2), sprintf('%d', j), ...
        'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'middle');
end
hold off

fprintf('There are %d circles in the image.\n', numel(s1));
saveas(fig10, 'hw5_fig10.jpg');

% Label the elements in each image.
verticalLinesLabels = logical(verticalLines);
horizontalLinesLabels = logical(horizontalLines);

s2 = regionprops(horizontalLinesLabels, 'Centroid');
fig10;
subplot(1,2,2); imshow(linesImage); title('Labeled Lines Image');
hold on
for j = 1:numel(s2)
    c = s2(j).Centroid;
    text(c(1), c(2), sprintf('%d', j), ...
        'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'middle', 'FontSize', 5);
end

s3 = regionprops(verticalLinesLabels, 'Centroid');
for k = 1:numel(s3)
    i = k + j;
    c = s3(k).Centroid;
    text(c(1), c(2), sprintf('%d', i), ...
        'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'middle', 'FontSize', 5);
```

```
end
hold off

fprintf('There are %d lines in the image.\n', numel(s2) +
numel(s3));
saveas(fig10, 'hw5_fig10.jpg');
```

5.

a. My method for computing the skeleton of images consisted of several binary image operations. First, I threshold the individual images by computing the gray threshold using the command graythresh() on a gray-scale version of the original image. Then using the threshold, I computed a binary image using imbinarize() and the found threshold level. I also inverted the images using imcomplement(). From researching skeletonizing, I found that skeletons are foreground images, so it is necessary for us to represent the objects as foreground images before skeletonization. Next, I displayed the images to decide on the structuring elements needed to close the "holes" in the image. The inverted images can be seen below:

Original Falcon Image

Original Girl Image

Original Dog Image

Original Balloon Image

From observing the girl image, I see the hair as an area which needs closing. Additionally, I see the base of the balloon image as an area requiring closing. When you increase the size of the structuring element, more of the foreground image is merge together. For example, applying a disk-shaped structuring element with pixel size 20 will attempt to connect the feet of the dog together, as it assumes the gap between them is a "hole" to close. In contrast, smaller structuring units will keep more holes in the image. When we use a structuring unit of size 2, the holes in the balloon and girl's hair still exist. Below are the examples of different structuring element sizes:

Closed Falcon Image with SE size 20

Closed Girl Image with SE size 20

Closed Falcon Image with SE size 2

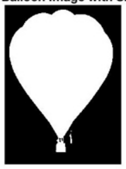
Closed Girl Image with SE size 2

Closed Dog Image with SE size 20

Closed Balloon Image with SE size 20

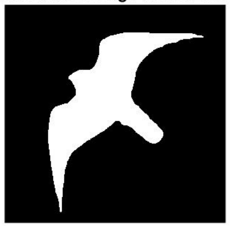
Closed Dog Image with SE size 2

Closed Balloon Image with SE size 2

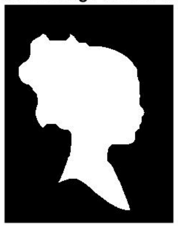


After trying various structuring element sizes, I decided on two structuring elements of disk-shape and pixel sizes 10 and 20. The 20-pixel SE will be used on the girl image, because the girl's hair requires more closing than do the other images. The 10-pixel SE will work for the other images, where we do not want as much closing to occur (for example, to keep the dog's feet separate). The final output is below along with the code used to create the closing operations:

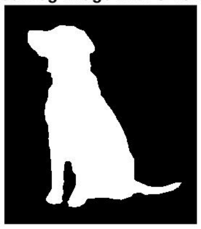
Closed Falcon Image with SE size 10



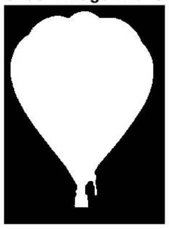
Closed Girl Image with SE size 20



Closed Dog Image with SE size 10



Closed Balloon Image with SE size 10



```
%% Problem 5
close all; clear; clc;

% Load images and perform thresholding.
falcon = imread('Falcon.png');
falcon = rgb2gray(falcon);
falconLevel = graythresh(falcon);

girl = imread('Girl.png');
girl = rgb2gray(girl);
girlLevel = graythresh(girl);
```

```
dog = imread('dog.png');
dog = rgb2gray(dog);
dogLevel = graythresh(dog);

balloon = imread('Balloon.png');
balloon = rgb2gray(balloon);
balloonLevel = graythresh(balloon);

% Binarize the images.
falcon = imcomplement(imbinarize(falcon, falconLevel));
girl = imcomplement(imbinarize(girl, girlLevel));
dog = imcomplement(imbinarize(dog, dogLevel));
balloon = imcomplement(imbinarize(balloon, balloonLevel));

% Zero-pad the left side so the balloon is not touching the edges
of the image.
[r, ~] = size(balloon);
balloon = [false(r,20), balloon];
[~, c] = size(balloon);
balloon = [false(20,c); balloon];

fig14 = figure(14);
subplot(2,2,1); imshow(falcon); title('Original Falcon Image');
subplot(2,2,2); imshow(girl); title('Original Girl Image');
subplot(2,2,3); imshow(dog); title('Original Dog Image');
subplot(2,2,4); imshow(balloon); title('Original Balloon Image');
saveas(fig14, 'hw5_fig14.jpg');

% Perform closing of images with disk shaped structuring element.
SE_close_med = strel('disk', 10);
SE_close_large = strel('disk', 20);

falconClosed = imclose(falcon, SE_close_med);
girlClosed = imclose(girl, SE_close_large);
dogClosed = imclose(dog, SE_close_med);
balloonClosed = imclose(balloon, SE_close_med);

fig15 = figure(15);
subplot(2,2,1); imshow(falconClosed); title('Closed Falcon Image
with SE size 10');
subplot(2,2,2); imshow(girlClosed); title('Closed Girl Image with
SE size 20');
subplot(2,2,3); imshow(dogClosed); title('Closed Dog Image with
SE size 10');
subplot(2,2,4); imshow(balloonClosed); title('Closed Balloon
Image with SE size 10');
saveas(fig15, 'hw5_fig15.jpg');
```

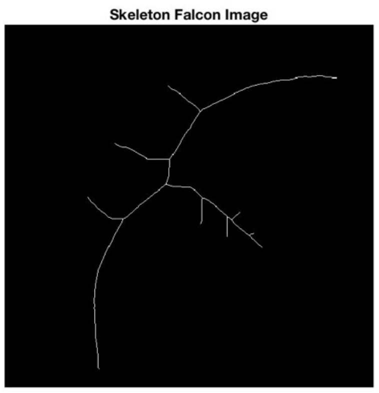
Finally, I applied skeletonization using the MATLAB function, bwmorph(). This function works by slowly thinning or eroding the surrounding boundaries of a binary object, while preserving the end points of line segments. The general algorithm that skeletonization follows is shown below:

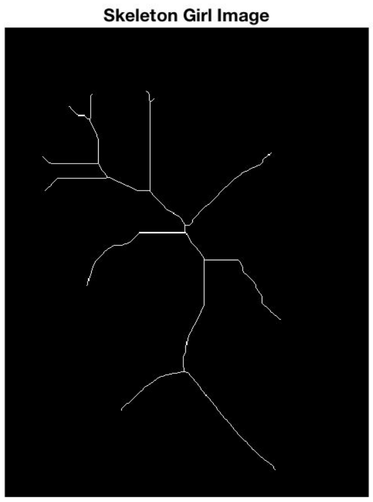
$$S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B$$

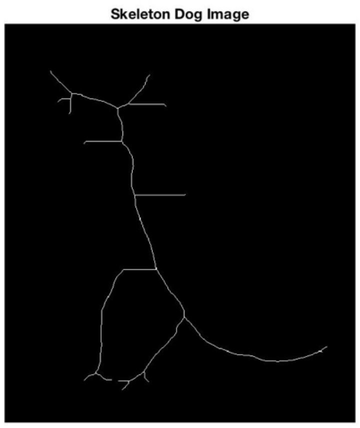
where,

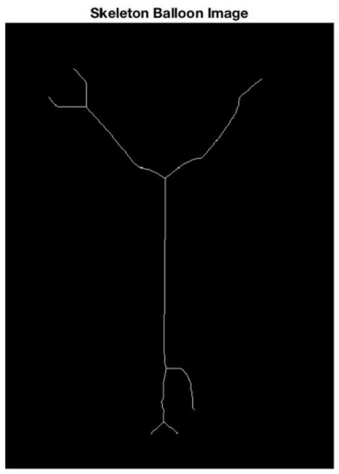
$$(A \ominus kB) = (\ldots(A \ominus B) \ominus B) \ldots) \ominus B$$
$$K = \max\{k \mid (A \ominus kB) \neq \emptyset\}$$

Skeletonization is an iterative process, where *k* successive erosions of A are performed by structuring element B. K represents the last iterative step before A erodes to an empty set. When I used bwmorph(), I passed in the ‘skel’ parameter to specify this algorithm. I also passed in *Inf*, which tells the function to perform infinite eroding operations, until the skeleton is found. Below are the output skeleton images and the code used to create them:









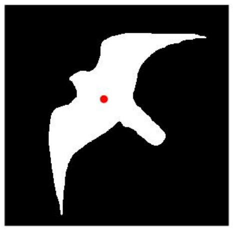
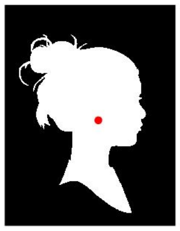
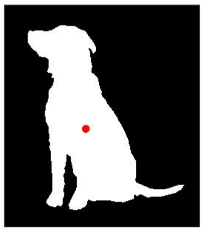
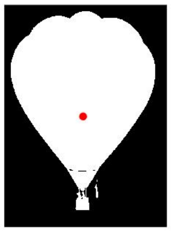
```
%% Create skeleton images from closed images.
skeletonizedFalcon = bwmorph(falconClosed, 'skel', Inf);
skeletonizedGirl = bwmorph(girlClosed, 'skel', Inf);
skeletonizedDog = bwmorph(dogClosed, 'skel', Inf);
skeletonizedBalloon = bwmorph(balloonClosed, 'skel', Inf);

fig16 = figure(16);
imshow(skeletonizedFalcon); title('Skeleton Falcon Image');
```

```
fig17 = figure(17);
imshow(skeletonizedGirl); title('Skeleton Girl Image');
fig18 = figure(18);
imshow(skeletonizedDog); title('Skeleton Dog Image');
fig19 = figure(19);
imshow(skeletonizedBalloon); title('Skeleton Balloon Image');

saveas(fig16, 'hw5_fig16.jpg');
saveas(fig17, 'hw5_fig17.jpg');
saveas(fig18, 'hw5_fig18.jpg');
saveas(fig19, 'hw5_fig19.jpg');
```

* NOTE: I decided to zero-pad the balloon image prior to image processing. I found that the balloon was not centered and was actually closer to the top left corner. By adding a few rows and columns of zeros to the image, I was able to prevent deformation.

b. The algorithm I created to calculate the location and center of mass of the shapes is as follows:
   i. Apply shrinking to the skeletonized image with infinite iterations using bwmorph() specifying 'shrink' and Inf parameters
      - This works by reducing the boundaries of the skeleton until a single point remains
   ii. Find the location of center of mass using find() to search for the remaining point
      - This works by searching the logical matrix for an index pair valued to 1 (remaining indices will hold 0 for black)
   iii. Use the indices found to plot the center of mass on the original images.

   The following images were displayed and the algorithm code is below:

```
%% Compute the center of mass of the shapes of the four images.
close all;

% Use shrinking operation with infinite iterations.
comFalcon = bwmorph(skeletonizedFalcon, 'shrink', Inf);
comGirl = bwmorph(skeletonizedGirl, 'shrink', Inf);
comDog = bwmorph(skeletonizedDog, 'shrink', Inf);
comBalloon = bwmorph(skeletonizedBalloon, 'shrink', Inf);

% Find the indices of center of mass locations.
[mFalcon, nFalcon] = find(comFalcon == 1);
[mGirl, nGirl] = find(comGirl == 1);
[mDog, nDog] = find(comDog == 1);
[mBalloon, nBalloon] = find(comBalloon == 1);

% Plot the center of mass on each of the images.
fig20 = figure(20); subplot(2,2,1);
imshow(falcon); hold on; plot(nFalcon, mFalcon, 'r.',
'MarkerSize', 15);
fig20; subplot(2,2,2);
imshow(girl); hold on; plot(nGirl, mGirl, 'r.', 'MarkerSize',
15);
fig20; subplot(2,2,3);
imshow(dog); hold on; plot(nDog, mDog, 'r.', 'MarkerSize', 15);
fig20; subplot(2,2,4);
imshow(balloon); hold on; plot(nBalloon, mBalloon, 'r.',
'MarkerSize', 15);
saveas(fig20, 'hw5_fig20.jpg');
```

c. To compute the area of each of the shape of images, I used the MATLAB function, bwarea(), which estimates the area of all of the foreground pixels in an image. It does so by summing the areas of each pixel in the image, by looking at each pixel's 2-by-2 neighborhood. The following areas were found and the code used is below:

Shape	**Area**
Falcon	53914
Girl	88950
Dog	90457
Balloon	17482

```
%% c. Compute the area of each of the shape of images.
close all;

areaFalcon = bwarea(falconClosed);
disp(areaFalcon);
areaGirl = bwarea(girlClosed);
disp(areaGirl);
areaDog = bwarea(dogClosed);
disp(areaDog);
areaBalloon = bwarea(balloonClosed);
disp(areaBalloon);
```