Infrared MATLAB Segmentation

EE5532 Module 2 Assignment

Matthew Spencer

02/14/2021

CONTENTS

Introduction

This module explores using MATLAB to program a custom segmentation algorithm that segments a data set of infrared camera images to locate people based on a previously supplied training data set. The images used in this module are taken from the open-source OTCBVS datasets publicly available online [1].

The first set of images is treated as " training" data for future data sets. The training data is run through the MATLAB algorithm to gather pertinent information that can be used in detecting "blobs" (clusters) of pixels that contain humans in the second set of images.. Human body heat appears brighter in the images due to their bodies being warmer than the background environment, and is therefore assigned brighter pixels by the infrared camera [2]. Therefore, this body-warmth data can be analyzed during the segmentation training within a normalized probability histogram [3] of each infrared grayscale image, where the brighter pixel values can be isolated. (The X axis is the pixel values, and the Y axis is the probability of that pixel appearing in the image.)

This report outlines the results of using the histogram analysis during the training portion of the algorithm, and then goes into the processes and different steps involved when finding and detecting human clusters based on the beginning testing data. These steps include manipulating the inputted images by diluting/eroding them to more easily extract each desired cluster, isolating the clusters by "tracing" the outline of each by detecting the bright pixel values next to the dark background pixels, and then building-in some basic rules to determine if an extracted cluster is a human or not to display the results.

Theory

After implementing the histogram equalization and detection algorithms summarized in the introduction, the custom MATLAB program should be able to successfully detect and locate human "blobs" within a similar directory of images as the training set. This recognition will be done by placing red rectangles over the human blobs found within the data set images.

Materials

- Windows 10 PC
    - 10400 i5 Intel Processor
    - 16 GB Ram
- MATLAB R2020a [4]

Training

The first directory of images are read into the program as the training data set. Each training image has its histogram equalization probability plot calculated using MATLAB's histogram [5] functionality. An example of the first training image [Figure 1] probability histogram can be seen below in Figure 2. The training set used for this report was the first OTCBVS data set, set "00001".

Figure 1: Training Set Image "img_00001.png"

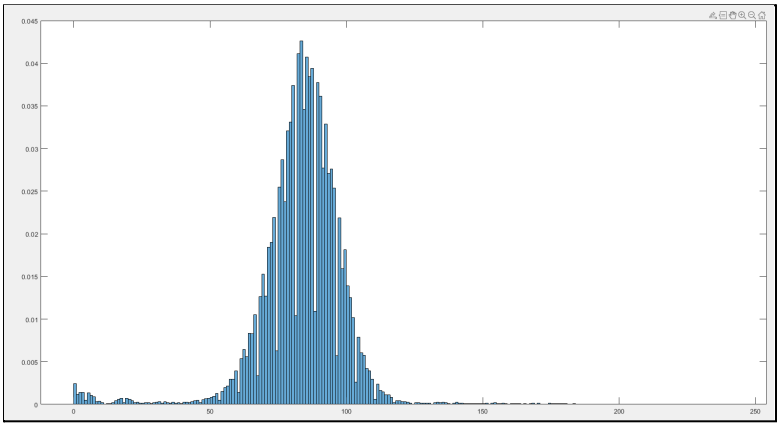

Figure 2: Training Set Probability Histogram for "img_00001.png"

Since humans are some of the brightest features represented in the infrared image as seen in Figure 1, we can assume that their values are represented by a number of standard deviations away towards the right side of the normalized mean observed in Figure 2. Therefore, the algorithm keeps a running set of pixel values that lie in the upper 99.99% of the histograms gathered from each training image. The median value of this data set is then used to locate potential human clusters in the future data set for segmenting.

Binary Segmentation

After the median pixel value is found from the training set, the next step is to find clusters of pixels at that value or greater and separate them from the image's background. These clusters (or blobs) are what the algorithm[1] guesses as being people in the image. This next step of segmentation re-assigns the grayscale pixel values for the found cluster to be the brightest pixel values possible, (which is 256 for an 8-bit grayscale image), and the lowest pixel value for pixels belonging to the background image (0). An example of this segmentation can be seen below in Figure 3.

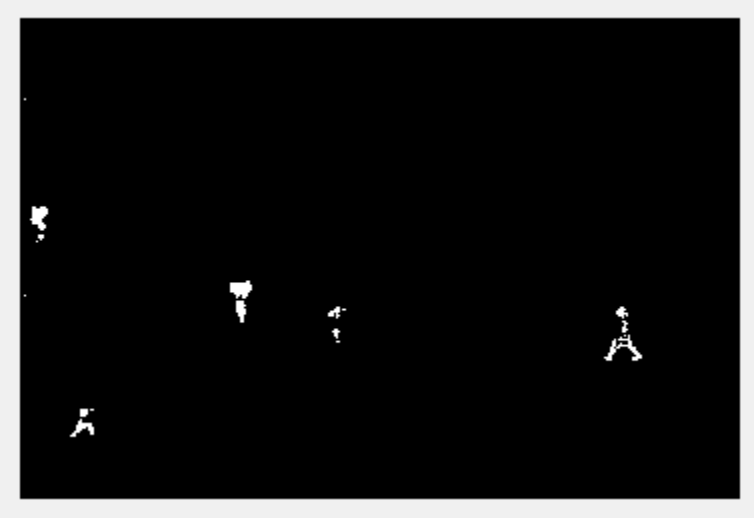

Figure 3: Binary Segmentation of Training Set Image "img_00001.png"

It is shown in Figure 3 above that these pixel clusters are fragmented from each other. In order to better identify the scale of each cluster, dilation must be done to better represent each cluster of pixels.

Dilation

In order to better piece together clusters that may originate from the same object in an image, the algorithm dilates the found pixels to bring them closer together. This is done by choosing a dilation size *d*, which forms an *d* x *d* neighborhood of pixels which iterates over each brightly segmented pixel and changes the surrounding pixels to 256 as well. This in effect "stretches" the cluster, and attempts to fill it in. The result is a blockier looking cluster, as seen in Figure 4.

[1] Matlab algorithms for each step in this report can be viewed at the end within the Appendix.

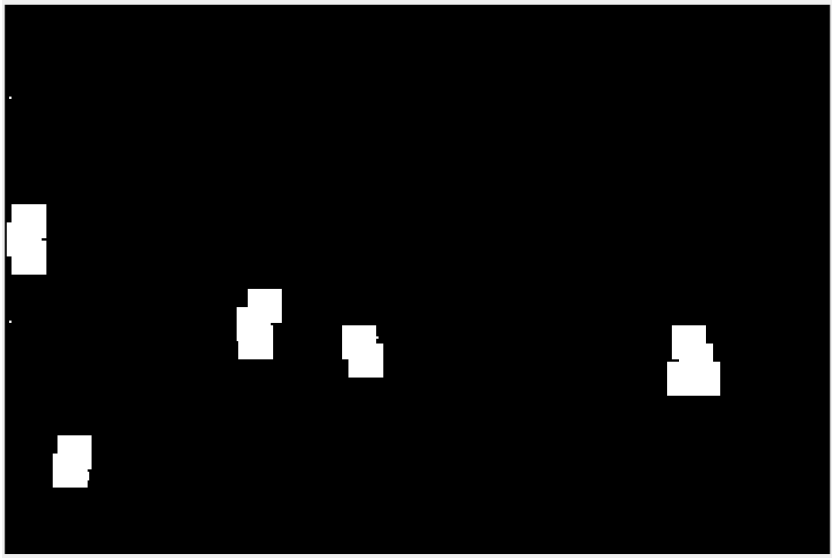

Figure 4: Dilation of Training Set Image “img_00001.png” where d = 7

Erosion

Some of the unwanted stretching effects of dilation can be partially corrected by then “eroding” the sides of the newly merged clusters. This is done by finding each cluster and applying the opposite calculations done during the dilation process. The difference in this calculation is that the erosion will only happen on the outside of the cluster once an edge has been detected within the neighborhood, shaving off e pixels. The variable e is chosen to be the floor of the value of $\frac{d}{2}$ for this report’s erosion algorithm step. This effect is shown below in Figure 5.

Figure 4: Dilation of Training Set Image “img_00001.png” where n = 7

Extraction

Each cluster is then individually extracted by tracing around each cluster's edge. The method used in this report works by moving through the binary segmented image from top-to-bottom, left-to-right, and then starting a trace on unvisited edges when discovered. (The edges that have been visited are remembered in a separate matrix.) This is done using a highly edge-case-oriented tracing algorithm, where a cursor is placed at the edge and told which direction to travel depending on the pixel's current direction and edge. A brief summary of this tracing algorithm's decision making is outlined in the following table, Table 1 and Table 2.

Neighboring Pixel Values to Check Current Edge 0 = Background Pixel 1 = Human Cluster Pixel				Action
Up	Right	Down	Left	
0	0	0	0	Nothing
0	0	0	1	Pre-check Right
0	0	1	0	Pre-check Up
0	0	1	1	Move Down
0	1	0	0	Pre-check Left
0	1	0	1	Pre-check Right/Left
0	1	1	0	Move Right
0	1	1	1	Move Right
1	0	0	0	Pre-check Down
1	0	0	1	Move Left
1	0	1	0	Pre-check Up/Down
1	0	1	1	Move Down
1	1	0	0	Move Up
1	1	0	1	Move Left
1	1	1	0	Move Up
1	1	1	1	(See Table 2)

Table 1: Tracing Algorithm Binary Decision Tree Part 1

Direction of Travel	Action
Up	Move Left
Right	Move Up
Down	Move Right
Left	Move Down

Table 2: Tracing Algorithm Decision Tree for a Pixel with No Edge

Each cluster extracted has their maximum and minimum values recorded in order to find appropriate bounding box coordinates. Every cluster must meet a certain set of manual rules and requirements hard-coded into the algorithm to be extracted, such as having a larger height than width (common for human blobs) [6], and having greater than a certain number of pixels. The training data currently saves the extracted clusters' maximum and minimum values from the training set. These values are saved in the program, but are not used at the time of writing this report. Future experimentation and study will have to be done to implement these variables within the data segmentation step.

Data Segmentation

Finally, the data set to be segmented is loaded into the program and put through the binary segmentation, dilation, erosion, and extraction steps from before. The result of this process leaves clusters and their bounding box coordinates extracted from after the erosion step (see Figure 4), which enables MATLAB's rectangle [7] function to draw a red bounding box over the unaltered data set image for observation.

The data set to be segmented was chosen to be a group of images from a similar time and location as the training data set, OTCBVS set "00007". Further training data sets will have to be loaded and configured to reliably extract human clusters.

Results

A few interesting images from the segmented data set and their extracted clusters can be seen below in Figures 5 - 12.

Figure 5: “img_00001.bmp” from Segmentation Data Set

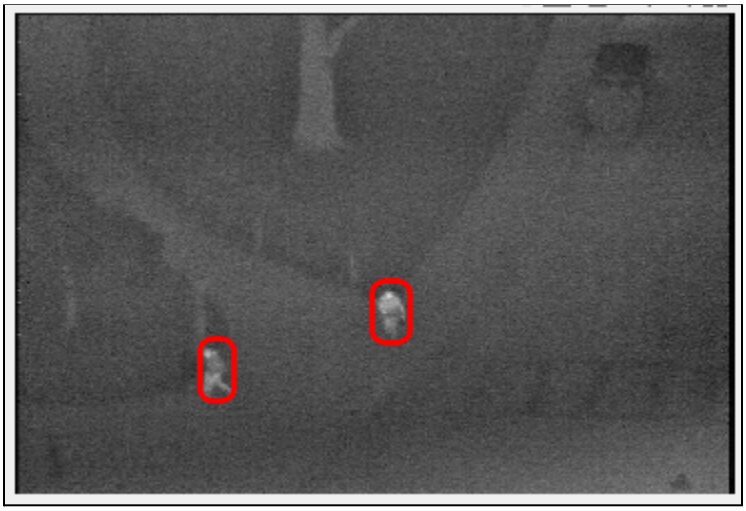

Figure 6: Segmented “img_00001.bmp” from Segmentation Data Set

Figure 7: “img_00004.bmp” from Segmentation Data Set

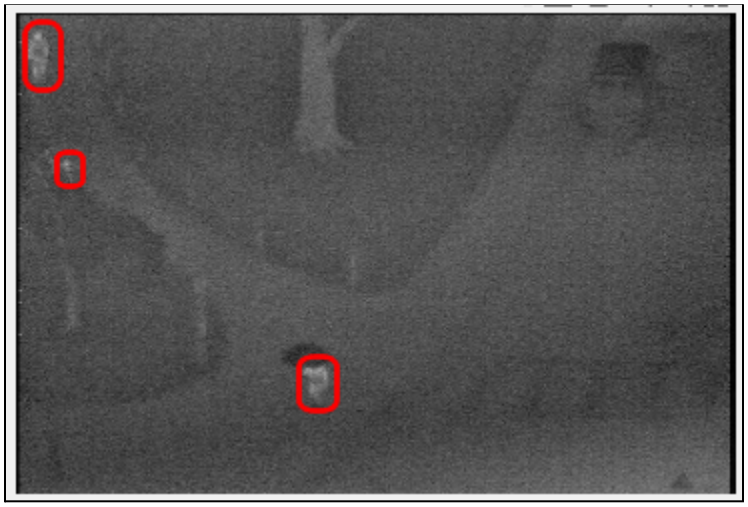

Figure 8: Segmented “img_00004.bmp” from Segmentation Data Set

Figure 9: “img_00013.bmp” from Segmentation Data Set

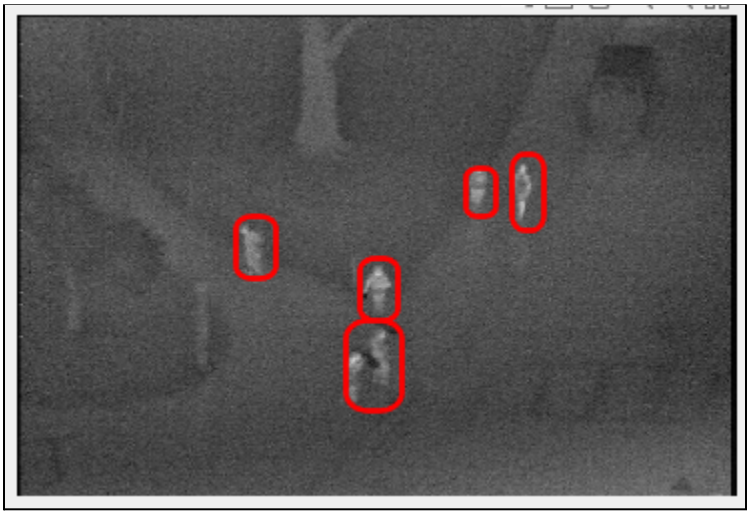

Figure 10: Segmented “img_00013.bmp” from Segmentation Data Set

Figure 11: “img_00015.bmp” from Segmentation Data Set

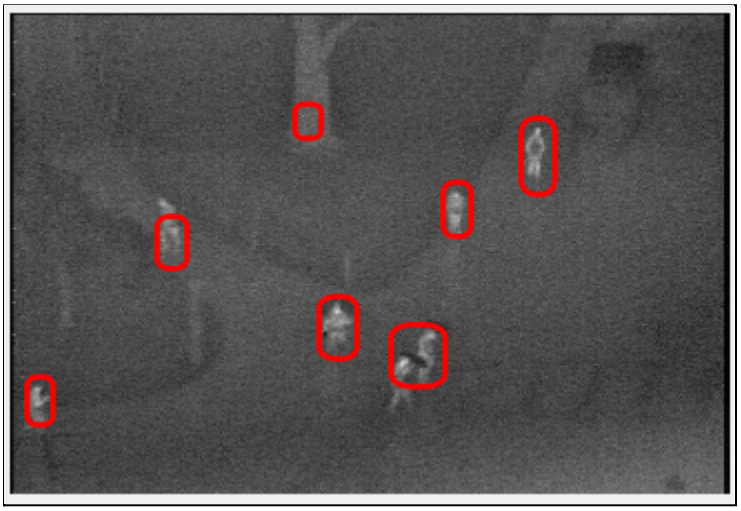

Figure 12: Segmented “img_00015.bmp” from Segmentation Data Set

Conclusion

Further experimentation and study is required to iron out the faults in the current segmentation implementation, as seen in Figures 10 and 12. Both figures have two people "merge" into one large cluster due to the close proximity to each other during the dilation phase. This could be tweaked by further fine tuning the dilation algorithm used. There was also a weird noise effect in Figure 12, where the extraction falsely extracted a non-human cluster at the base of the tree. This could potentially be corrected by incorporating the height and width data gathered during training, or by reading in more training data sets.

In summary, the segmentation algorithm did a decent job in extracting humans from the infrared images supplied by the OTCBVS dataset, and having MATLAB place red outlines over the desired clusters. There is also room for improvements that can be made towards the current segmentation algorithm outlined in this report for future studies.

References

1. "OTCBVS Benchmark Dataset Collection", *Dataset 1: OSU Thermal Pedestrian Database*, 2021, [Online] Available: http://vcipl-okstate.org/pbvs/bench/, [Accessed: Feb. 14, 2021]

2. "Infrared", *Wikipedia*, Fed. 13, 2021, [Online] Available: https://en.wikipedia.org/wiki/Infrared, [Accessed: Feb. 14, 2021]

3. R. Keim, "The Normal Distribution: Understanding Histograms and Probability", *All About Circuits*, Aug. 7, 2020, [Online] Available: https://www.allaboutcircuits.com/technical-articles/normal-distribution-understanding-histograms-probability/, [Accessed: Feb. 14, 2021]

4. "MATLAB", *MathWorks*, R2020b, 2021, [Online] Available: https://www.mathworks.com/products/matlab.html, [Accessed: Feb. 14, 2021]

5. MATLAB, "Histogram", *MathWorks*, r2020b, 2021, [Online] Available: https://www.mathworks.com/help/matlab/ref/matlab.graphics.chart.primitive.histogram.html, [Accessed: Feb. 14, 2021]

6. M. Roggemann, “EE5532 Scoring Computer Vision”, *Michigan Technological University*, 2021, [Online] Available: https://mtu.instructure.com/courses/1347065/files/92729453?wrap=1, [Accessed: Feb. 14, 2021]

7. MATLAB, “Rectangle”, *MathWorks*, r2020b, 2021, [Online] Available: https://www.mathworks.com/help/matlab/ref/rectangle.html, [Accessed: Feb. 14, 2021]

Appendix

```
% view each image
fprintf("Learning from test data...\n")
for i=1:numImages
    % get image name
    imageName = fscanf(fid, '%c', 13);
    %fprintf("%s\n", imageName);

    % Get the current image
    fname = fullfile(SEQ_DIR, imageName);
    Im = imread(fname);
    imagesc(Im);

    % Compute img histogram, normalize its probabilities
    hist = histogram(Im, 'Normalization', 'probability');
    maxProb  = 0;
    maxPixel = 0;
    [~, n] = size(hist.Values);
    for k=1:n
       if (hist.Values(1,k) > maxProb)
          maxProb  = hist.Values(1,k);
          maxPixel = k;
       end
    end

    % Find the light pixel vales 2 std deviation away from mean
    hist2std = 0;
    for k=1:n
       if (k > maxPixel && hist.Values(1,k) <= 0.000001)
          hist2std = k;
          break;
       end
    end

    % Save the std value
    stdMem(1,i) = hist2std;

    % Move to the next image
    fgets(fid);
end

% Get the median pixel value for segmentation
med = median(stdMem);

% Close file and open it again
fclose(fid);
```

Appendix 1: Module_2.m Compute Histograms

```
% Dilate image
d = 7;
neighborhood = -d:1:d;
NHOOD = length(neighborhood);
r = d + 2;
while r < nRows-d-2
    skip = 0;
    c = d + 2;
    while c < nCols-d-2
        if (newImg(r,c) > 0)
           for p=1:NHOOD
              x = neighborhood(1,p);
              for q=1:NHOOD
                 y = neighborhood(1,q);
                 newImg(r+x,c+y) = 256;
              end
           end
           c = c + d + 1;
           skip = 1;
        else
           c = c + 1;
        end
    end
    if (skip == 1)
       r = r + d + 1;
    else
       r = r + 1;
    end
end
```

Appendix 2: Module_2.m Dilation Algorithm

```
% Erode image
erode = ceil(d/2);
r     = d + 1;
while r < nRows-d-2
    c = d + 1;
    while c < nCols-d-2
        skipC = 0;
        if (newImg(r,c) > 0)
           if (newImg(r,c-d) == 0)
              for p=1:erode
                 newImg(r,c-d+p) = 0;
              end
              skipC = 1;
           end
           if (newImg(r,c+d) == 0)
               for p=1:erode
                 newImg(r,c+d-p) = 0;
               end
               skipC = 1;
           end
        end
        if (skipC == 1)
            c = c + erode + 1;
        else
            c = c + 1;
        end
    end
    r = r + 1;
end
```

Appendix 3: Module_2.m Erosion Algorithm

```
% Keep track of blobs that have been found
visited = zeros(nRows, nCols);

% Draw an outline around the blobs
% Check to make sure height > width
r = 1;
while r < nRows
    c = 1;
    while c < nCols
        m = 1;
        if (newImg(r,c) > 0 && visited(r,c) == 0)
            p      = r;
            q      = c;
            up     = 0;
            right  = 0;
            down   = 0;
            left   = 0;
            go     = 1;
            check  = 0;
            maxR   = -1;
            minR   = 99999;
            maxC   = -1;
            minC   = 99999;
            %fprintf("(%d,%d)\n", p,q);
            while (go == 1)
                %fprintf("(%d,%d)\n", p,q);
                %fprintf("%d,%d,%d,%d\n",newImg(p-1,q), newImg(p,q+1), newImg(p+1,q), newImg(p,q-1));
                %fprintf("Direction: %d,%d,%d,%d\n", up, right, down, left);
                %fprintf("max and mins: %d,%d,%d,%d\n", maxR, minR, maxC, minC);

                % Check for a valid starting point
                if (up == 0 && right == 0 && down == 0 && left == 0) &&...
                   ((newImg(p-1,q) == 1 && newImg(p,q+1) == 1 && newImg(p+1,q) == 1 && newImg(p,q-1) == 1) ||...
                    (newImg(p-1,q) == 1 && newImg(p,q+1) == 0 && newImg(p+1,q) == 0 && newImg(p,q-1) == 0) ||...
                    (newImg(p-1,q) == 0 && newImg(p,q+1) == 1 && newImg(p+1,q) == 0 && newImg(p,q-1) == 0) ||...
                    (newImg(p-1,q) == 0 && newImg(p,q+1) == 0 && newImg(p+1,q) == 1 && newImg(p,q-1) == 0) ||...
                    (newImg(p-1,q) == 0 && newImg(p,q+1) == 0 && newImg(p+1,q) == 0 && newImg(p,q-1) == 1) ||...
                    (newImg(p-1,q) == 0 && newImg(p,q+1) == 1 && newImg(p+1,q) == 0 && newImg(p,q-1) == 1) ||...
                    (newImg(p-1,q) == 1 && newImg(p,q+1) == 0 && newImg(p+1,q) == 1 && newImg(p,q-1) == 0))
                    go = 0;
                end
                if (check == 0) &&...
                   ((newImg(p-1,q) == 1 && newImg(p,q+1) == 1 && newImg(p+1,q) == 1 && newImg(p,q-1) == 1) ||...
                    (newImg(p-1,q) == 1 && newImg(p,q+1) == 0 && newImg(p+1,q) == 0 && newImg(p,q-1) == 0) ||...
                    (newImg(p-1,q) == 0 && newImg(p,q+1) == 1 && newImg(p+1,q) == 0 && newImg(p,q-1) == 0) ||...
                    (newImg(p-1,q) == 0 && newImg(p,q+1) == 0 && newImg(p+1,q) == 1 && newImg(p,q-1) == 0) ||...
                    (newImg(p-1,q) == 0 && newImg(p,q+1) == 0 && newImg(p+1,q) == 0 && newImg(p,q-1) == 1) ||...
                    (newImg(p-1,q) == 0 && newImg(p,q+1) == 1 && newImg(p+1,q) == 0 && newImg(p,q-1) == 1) ||...
                    (newImg(p-1,q) == 1 && newImg(p,q+1) == 0 && newImg(p+1,q) == 1 && newImg(p,q-1) == 0))
                    go = 0;
                end
```

Appendix 4: Module_2.m Extraction Algorithm Part 1

```
% Precheck right and move
if (right == 1) &&...
   ((newImg(p-1,q+1) == 0 && newImg(p,q+2) == 1 && newImg(p+1,q+1) == 0 && newImg(p,q) == 1) ||...
    (newImg(p-1,q+1) == 0 && newImg(p,q+2) == 0 && newImg(p+1,q+1) == 0 && newImg(p,q) == 1))
    newImg(p,q+1) = 0;
elseif (right == 1)
    q = q + 1;
    if (p > maxR)
       maxR = p;
    end
    if (p < minR)
       minR = p;
    end
    if (q > maxC)
       maxC = q;
    end
    if (q < minC)
       minC = q;
    end
    visited(p,q) = 1;
    check = 1;
    m = m + 1;

% Precheck down and move
elseif (down == 1) &&...
   ((newImg(p,q) == 1 && newImg(p+1,q+1) == 0 && newImg(p+2,q) == 1 && newImg(p+1,q-1) == 0) ||...
    (newImg(p,q) == 1 && newImg(p+1,q+1) == 0 && newImg(p+2,q) == 0 && newImg(p+1,q-1) == 0))
    newImg(p+1,q) = 0;
elseif (down == 1)
    p = p + 1;
    if (p > maxR)
       maxR = p;
    end
    if (p < minR)
       minR = p;
    end
    if (q > maxC)
       maxC = q;
    end
    if (q < minC)
       minC = q;
    end
    visited(p,q) = 1;
    check = 1;
    m = m + 1;
```

Appendix 5: Module_2.m Extraction Algorithm Part 2

```
% Precheck left and move
elseif (left == 1) &&...
    ((newImg(p-1,q-1) == 0 && newImg(p,q) == 1 && newImg(p-1,q-1) == 0 && newImg(p,q-2) == 1) ||...
     (newImg(p-1,q-1) == 0 && newImg(p,q) == 1 && newImg(p-1,q-1) == 0 && newImg(p,q-2) == 0))
    newImg(p,q-1) = 0;
elseif (left == 1)
    q = q - 1;
    if (p > maxR)
       maxR = p;
    end
    if (p < minR)
       minR = p;
    end
    if (q > maxC)
       maxC = q;
    end
    if (q < minC)
       minC = q;
    end
    visited(p,q) = 1;
    check = 1;
    m = m + 1;

% Precheck up and move
elseif (up == 1) &&...
    ((newImg(p-2,q) == 1 && newImg(p-1,q+1) == 0 && newImg(p,q) == 1 && newImg(p-1,q-1) == 0) ||...
    (newImg(p-2,q) == 0 && newImg(p-1,q+1) == 0 && newImg(p,q) == 1 && newImg(p-1,q-1) == 0))
     newImg(p-1,q) = 0;
elseif (up == 1)
    p = p - 1;
    if (p > maxR)
       maxR = p;
    end
    if (p < minR)
       minR = p;
    end
    if (q > maxC)
       maxC = q;
    end
    if (q < minC)
       minC = q;
    end
    visited(p,q) = 1;
    check = 1;
    m = m + 1;
end

% Travel Right
if (newImg(p-1,q) == 0 && newImg(p,q+1) == 1 && newImg(p+1,q) == 1 && newImg(p,q-1) == 1) ||...
   (newImg(p-1,q) == 0 && newImg(p,q+1) == 1 && newImg(p+1,q) == 1 && newImg(p,q-1) == 0) ||...
   ((newImg(p-1,q) == 1 && newImg(p,q+1) == 1 && newImg(p+1,q) == 1 && newImg(p,q-1) == 1) &&...
    down == 1)
    up    = 0;
    right = 1;
    down  = 0;
    left  = 0;
```

Appendix 6: Module_2.m Extraction Algorithm Part 3

```
    % Travel down
    elseif (newImg(p-1,q) == 0 && newImg(p,q+1) == 0 && newImg(p+1,q) == 1 && newImg(p,q-1) == 1) ||...
           (newImg(p-1,q) == 1 && newImg(p,q+1) == 0 && newImg(p+1,q) == 1 && newImg(p,q-1) == 1) ||...
           ((newImg(p-1,q) == 1 && newImg(p,q+1) == 1 && newImg(p+1,q) == 1 && newImg(p,q-1) == 1) &&...
            left == 1)
        up    = 0;
        right = 0;
        down  = 1;
        left  = 0;

    % Travel Left
    elseif (newImg(p-1,q) == 1 && newImg(p,q+1) == 1 && newImg(p+1,q) == 0 && newImg(p,q-1) == 1) ||...
           (newImg(p-1,q) == 1 && newImg(p,q+1) == 0 && newImg(p+1,q) == 0 && newImg(p,q-1) == 1) ||...
           ((newImg(p-1,q) == 1 && newImg(p,q+1) == 1 && newImg(p+1,q) == 1 && newImg(p,q-1) == 1) &&...
            up == 1)
        up    = 0;
        right = 0;
        down  = 0;
        left  = 1;

    % Travel Up
    elseif (newImg(p-1,q) == 1 && newImg(p,q+1) == 1 && newImg(p+1,q) == 0 && newImg(p,q-1) == 0) ||...
           (newImg(p-1,q) == 1 && newImg(p,q+1) == 1 && newImg(p+1,q) == 1 && newImg(p,q-1) == 0) ||...
           ((newImg(p-1,q) == 1 && newImg(p,q+1) == 1 && newImg(p+1,q) == 1 && newImg(p,q-1) == 1) &&...
            right == 1)
        up    = 1;
        right = 0;
        down  = 0;
        left  = 0;

    % Single pixel
    elseif (newImg(p-1,q) == 0 && newImg(p,q+1) == 0 && newImg(p+1,q) == 0 && newImg(p,q-1) == 0)
        go = 0;
    end

    % Reached the start
    if (p == r && q == c) && (check == 1)
        go = 0;
    end
end
```

Appendix 7: Module_2.m Extraction Algorithm Part 4

```
                    % Check to make sure width is less than height
                    if (width < height)
                        nBlob = nBlob + 1;
                        data(nBlob).values = [minC, minR, width, height];
                    end
                end
                c = c + 1;
            end
            r = r + 1;
        end

------------------------------------------------------------------------Seg Figures
        % Print segmented image
        figure;
        imshow(Im);
        pad = 5;
        for b=1:nBlob
            rectangle('Position',...
                [data(b).values(1),data(b).values(2),...
                data(b).values(3)+pad,data(b).values(4)+pad],...
                'Curvature',[0.8,0.4],...
                'EdgeColor', 'r',...
                'LineWidth', 2,...
                'LineStyle','-')
        end
        pause(0.1);
        fgets(fd);
    end

    disp('[DONE]');
    return;
```

Appendix 8: Module_2.m Display Extracted Figures