Computer Vision 600100

Counting Starfish

Student ID: 201601620

Date: **August 3, 2020**

Deadline: Monday 3rd August 2020 by 2pm

Image Processing Pipeline

In tackling the task to create an image processing pipeline I decided to create a linear pipeline. The pipeline I create starts with loading in the image and viewing the data for the image. A histogram is then made of the image to see the values in the 255 greyscale of the image (see figure 1).

```
% Step 1 Read in an image

% orig = original image that is going to be passed through the pipeline
orig = imread('starfish.jpg');
star = orig;
imshow(orig);

% Checking the image data that gets stored in the workspace
whos orig;

% Seeing what the image contrast is like to ensure that the full range is
% used. Equalising spreads the intensity values over the full range of the image.
figure
imhist(star);

equalized = histeq(star);

figure
imhist(equalized);

figure
imshowpair(star,equalized,'montage');
```

```
Name        Size                Bytes  Class    Attributes

orig        362x438x3          475668  uint8
```

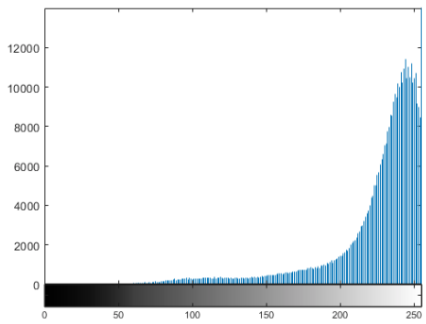

Figure 1 - Histogram of the unaltered "starfish.jpg"

As seen in figure 1, there is also some equalizing done on the image using the function *histeq* to see how a more equalized image would fare. This was not the best idea as the route I chose to take would work better if the image wasn't across the whole grayscale and more isolated to one area.

The next step in the pipeline was to filter the image. This was done using a custom function, *filterImage*. This function takes in the image in question (e.g. starfish.jpg) and then converts it into a grayscale image. This turns the image from 3D into a 2D image, this allows the ability to use the *medfilt2* function to remove the noise. Removing the noise allows for a much clearer image (see figure 2) and will help to make a more accurate image segmenting process.

```
% imshowpair(star, edge(rgb2gray(star), 'Canny'), 'montage');
%
% hold off;

% Step 2.1 filter the image

function star = filterImage(star)
    star = rgb2gray(star);
    star = medfilt2(star);
    imshow(star);
end


% Step 2.2 Threshold the image
function star = binaryObjectDetection(star,orig)
    thresh = graythresh(star);
    imshowpair(orig,star,'montage');

    % Step 2.3 Binarize the image
```

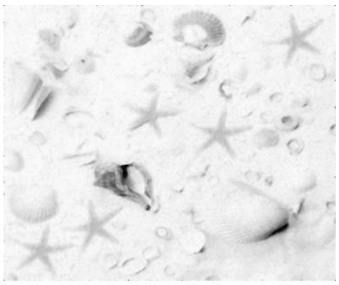

Figure 2 - filterImage function with denoised image

Once filterImage has taken place we now have a grayscale image which we can work with. This will be used in the next step which is thresholding and binarizing the image. I used *graythresh* so that the best grayscale levels were selected. This allows the image to become a binary image instead of a 2D image (essentially black and white/1 or 0). Using the function *imcomplement* inverts the binary image to bring the objects to the foreground and send, what seems to be the background, into the background. This would be done by taking X and the function *imcomplement* inverts it to become ~X.

```
% Step 2.2 Threshold the image
function star = binaryObjectDetection(star,orig)
    thresh = graythresh(star);
    imshowpair(orig,star,'montage');

    % Step 2.3 Binarize the image

    mask = imbinarize(star,thresh);
    imshowpair(star,mask,'montage');

    % Step 2.4 Bring the image to the forefront

    star = imcomplement(mask);
    imshowpair(orig,star,'blend');

    % Step 2.5 Clean up the mask so that detecting the object is easier
    mask2 = bwareaopen(star, 350);
```

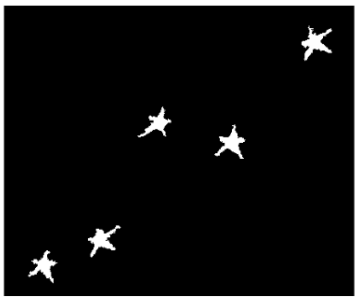

Figure 3 - The binarizing of the function binaryObjectDetection

As seen in figure 3, we can now use the binarized image to detect the objects within it. Using the function *bwareaopen* I tried to remove other objects that were visible such as the seashells, unfortunately this only removed the smaller objects. To remove the objects that were larger I had to use the *xor* function. This allowed for me to have 2 *bwareaopen* functions together which worked as a min and max. removing anything higher than the max and anything lower than the min. The two numbers used to remove objects of that amount of pixels were set through trial and error.

Once the starfish objects are isolated, they are then put through an *imfill* function. This was done to ensure that the objects were full of little to no holes in them. Once this was done some extra morphology needed to be used to make the objects clearer. This was done using some structured elements to dilate and close the objects using the functions *imclose* and *imdilate*. This too was achieved through trial and error of the radius this allowed us to then have “fat” star fish shapes (see figure 4).

```
    %Using an xor so that the larger ojects such as the shells will be removed
    mask3 = xor(bwareaopen(star, 350), bwareaopen(star,630));
    imshowpair(mask3,mask2,'montage');

    star = imfill(mask3,'holes');
    imshow(star);

    % Step 3
    close_se = strel('disk',8,4);
    dilate_se = strel('disk',5,4);
    closeBW = imclose(star,close_se);
    dilateBW = imdilate(closeBW,dilate_se);
    figure
    subplot(1,3,1);
    imshow(star);
    subplot(1,3,2);
    imshow(closeBW);
    subplot(1,3,3);
    imshow(dilateBW);

    star = dilateBW;

end
```

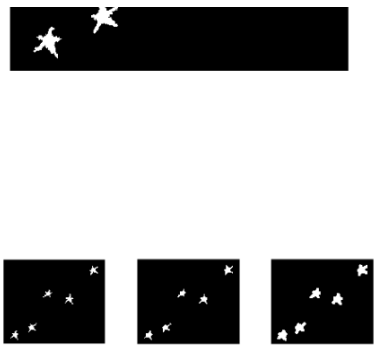

Figure 4 - The morphology of the function binaryObjectDetection

The next step is to detect the objects individually within the binary image. This process starts in the *objectCount* function which labels each “blob” individually. This then provides us with a number of “blobs” that are in the binary image. For just a better visual aid I also used *label2rgb* to provide a figure displaying the detected starfish in different colours (see figure 5).

```
function rgb = objectCount(star, orig)

    [l, connected] = bwlabel(star,4);

    rgb = label2rgb(l,'hsv','k');

    connected = string(connected);

    connected

    figure
    imshow(rgb);

    objectHighlighting(rgb, star, connected, orig);
end
```

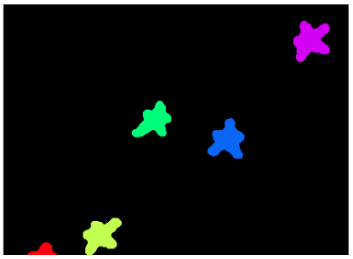

Figure 5 - The counting of the objects using function objectCount

Finally the last step was to then show the binary image overlaying on top of the original image and then using *regionprops* we could outline each starfish and title the figure with the appropriate amount detected (see figure 6).

```
function objectHighlighting(rgb, star, connected, orig)
    info = regionprops(rgb,orig,'BoundingBox');
    numberOfBlobs = size(info, 1);

    figure
    subplot(1,1,1);
    imshowpair(orig,star,'blend');
    titlestr = append("There are ", connected, " starfish");
    title(titlestr);
    axis image;
    hold on;
    bounds = bwboundaries(star);
    numberOfBounds = size(bounds,1);
    for k = 1 : numberOfBounds
        thisBoundary = bounds{k};
        plot(thisBoundary(:,2), thisBoundary(:,1), 'r', 'LineWidth', 2);
    end
    hold off;
end
```

Figure 6 - Detected star fish, highlighted using the function objectHighlighting

Results

Figure 7 - Starfish.jpg result after passing through the pipeline

As seen in figure 7, the starfish were roughly outlined on the starfish.jpg file after being put through the pipeline. The correct amount of starfish was detected.

Figure 8 - starfish_noise5.jpg result

As seen in figure 8, the starfish were roughly outlined on the starfish_noise5.jpg file after being put through the pipeline. The correct amount of starfish was detected.

Figure 9 - starfish_5.jpg result

As seen in figure 9, the starfish were roughly outlined on the starfish_5.jpg file after being put through the pipeline. Not all the star fish were detected.

Discussion

The way the filter and the image segmentation were done could have been done in a more dynamic way. Using the image data would probably be better to make the segmentation and the filter dynamic so that it would change slightly between images put through the pipeline.

The pipeline itself, being linear, could be changed to be a bit more complex allowing for a more accurate result. Using a parallel pipeline could work to make a more accurate result. This way you can merge 2 pipelines to make a more complex pipeline.

Edge detection could have helped in the more complex images to help find the objects a lot easier.

References