Ceng 466 - Take Home Exam 3 A_1

Eda Özyılmaz
2171882

Hilal Ünal
2172112

I. Introduction

This document is designed as a report for CENG466, Fundamentals of Image Processing, Take Home Exam 3. This report includes explanations of our methodology and analysis of the results we received while developing our code. The report mainly explains the methodology and rationale behind our algorithm design. Comments and explanations can be found in the following section.

II. Questions

In this homework, there are three questions which are Question 1 - Object Counting, Question 2 - Image Segmentation, and Question 3 - Apple Detection. The questions and our answers, analysis, and methods to solve these questions are explained in detail in the following sub-sections.

A. Question 1 - Object Counting

In this question we were asked to find the flying jet objects from 6 different given images. After founding the jets, we should count them and print the number of flying jets in the image.

We first get the images by using the *imread(..)* function. Then we translate the given image into a gray image by using *rgb2gray(image)*. By using the gray image, we calculated a binary image. to find out which pixels are going to be 1 which are going to be 0, we looked for the gray images' pixel values by using Matlab tools. For the first image if the gray image's pixel value is larger than 50, the binary value of that pixel is 0. Otherwise, it is 1. We found the limit which is 50 for this image, by looking into the jets' pixel values on the gray image. The jets mostly have value less than 50; therefore, if the pixel value is larger than 50, the binary image has 0 as pixel value. This limit is different for every image.

After creating image's binary version, we realised that image has some holes on the detected objects. Therefore, for the first image we created a 5x5 binary matrix to fill the holes on the detected objects. Then we used *imclose(binary_image,matrix)* to fill the holes in the binary image. Then we calculated the matrix named result with the following function *bwlabel(binary_image)* which returns the label matrix that contains labels for the 8-connected objects found in 2-D binary image.

We also created a function to remove the unwanted components from the result but for the image 1 it was not necessary. this function simply takes the result image and an index number which is the index of the unwanted component and it makes the pixel values of that component 0. This function is needed for other images.

After removing unwanted object we again used the *bwlabel(result)* function. This function returns result and n that represents the counted component number. So we used the following lines to save the image and to print the object counts:

```
imwrite(result, 'part1_A1.png');
fprintf('The number of flying jets in image A1 is %d', n);
```

For the other images the basic algorithm is the same; however, some of the limit numbers are different, such as the limit when we calculated the binary image, the size of the binary matrix that is used to fill the holes in the objects, and index of the unwanted component to remove it. The appropriate values for these variables of other images can be seen below.

For the image 2, we again calculated the limit values to create binary image by using its gray image. The limits we found are the followings, *A2_gray(y,x) > 110 || A2_gray(y,x) < 65* if this condition holds then the pixel (y,x) of the created binary image is 0, otherwise it is 1. To fill the holes we created a 1x1 matrix. To remove the mountain object from the binary image, we used the function we created for unwanted objects whose index is 1. When we done all of these steps we had a image which also contains edge of the mountain which can be seen on the Figure 1. Therefore, after making research we decided to use the following code lines to overcome this issue:

```
result=bwareaopen(result,500);
se = strel('disk',5);
result = imclose(result,se);
```

This codes removes the unwanted components if they are smaller than 5 pixels for this image. The final image after applying these part can be seen on the Figure 2.

For the image 3, we calculated the limit values to create binary image by using its gray image. The limits we found are the followings, *A3_gray(y,x) > 60* if this condition holds then the pixel (y,x) of the created binary image is 0, otherwise it is 1. To fill the holes we created a 9x9 matrix. To remove the sky from the binary image, we used the function that we created for unwanted objects with index 1.

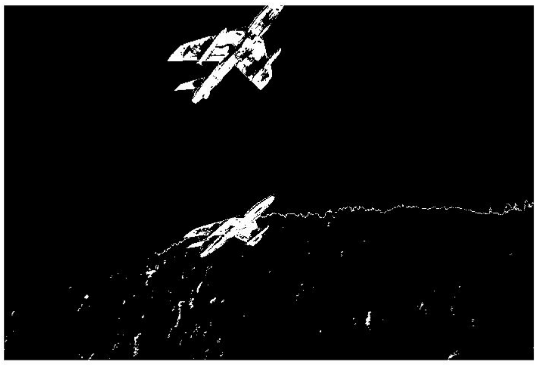

Fig. 1. Binary Image with Edges

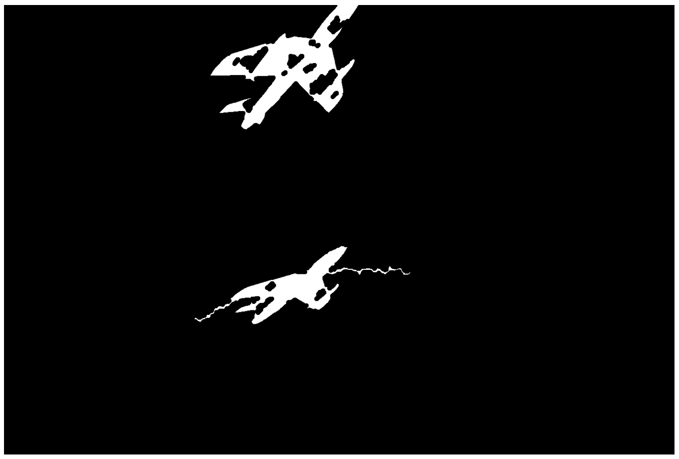

Fig. 2. Binary Image without Unwanted Components

For the image 4, the limits we found are the followings *A4_gray(y,x) > 40* if this condition holds then the pixel (y,x) of the created binary image is 0, otherwise it is 1. To fill the holes we created a 1x1 matrix. To remove the sky from the binary image, we used the function that we created for unwanted objects with index 1. We again used the codes which we used for image 2 with *se = strel('disk',2);* new index this time to get rid of the unwanted small components.

For the image 5, the limits we found are the followings *A5_gray(y,x) > 90 A5_gray(y,x) < 130* if this condition holds then the pixel (y,x) of the created binary image is 0, otherwise it is 1. To fill the holes we created a 7x7 matrix. To remove the sky from the binary image, we used the function that we created for unwanted objects with index 1. We again used the *se = strel('disk',20);* with new index this time to get rid of the unwanted small components.

For the image 6, the limits we found are the followings *A6_gray(y,x) > 10 A6_gray(y,x) < 235* if this condition holds then the pixel (y,x) of the created binary image is 0, otherwise it is 1. To fill the holes we created a 1x1 matrix. To remove the mountain and sky from the binary image, we used the function that we created for unwanted objects with index 1. We again used the *result=bwareaopen(result,500);* to get rid of the unwanted small components.

B. Question 2 - Image Segmentation

For this question, we were asked to use two different for image segmentation on given first 25 images from the Berkeley Dataset.

For the first algorithm, we used algorithm we find in the web page of Matlab for the Image Segmentation, that segments color images into regions. For every image we call *colImgSeg(..)* functions with the parameters input image, number of bins that gonna use, window size and number of classes that needed for segmentation.

colImgSeg(..) gives gray image as an output, so we use *ind2rgb(image, colormap)* function that gives a color for every segmented regions. We use *colormap(jet(256)* as a colormap.

As second algorithm, we used existing The Watershed Transform for image segmentation. To use Watershed Transform, we first translated our image in to a binary image by using the following *binImg = ∼ im2bw(C1, graythresh(C1));*. Then we calculated the distance transform which is the distance from every pixel to the nearest nonzero-valued pixel of this binary image with the following *D = -bwdist(∼ binImg);*. Then we used the watershed function *L = watershed(D);*. After this function we got the label, and we converted it to RGB by using *rgb_C1 = label2rgb(L,'jet',[.5 .5 .5]);*.

We applied both of the algorithms to all given 25 images. We found that the first algorithm is better than the second one because for some images second watershed algorithm didn't compute clear segmented images. However, first algorithm generates better and clear segmented images. The segmented images of the 100080.jpg with the algorithm 1 can be seen as Figure 3 and with the algorithm 2 can be seen as Figure 4.

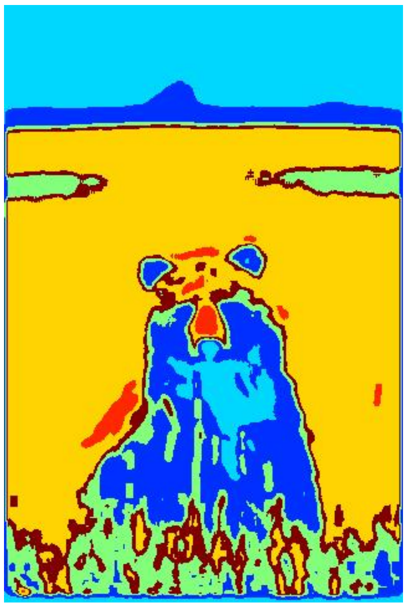

Fig. 3. 100080.jpg with Algorithm 1

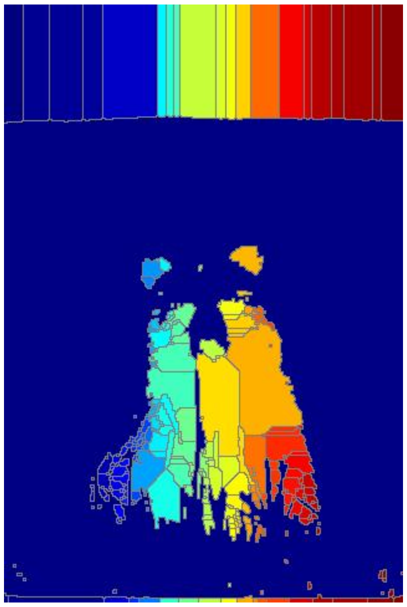

Fig. 4. 100080.jpg with Algorithm 2 - Watershed

As you can see from the Figure 3 and 4, the first algorithm segments the given image better than the second algorithm. For example in this image first algorithm even segments the grass and the background while second algorithm does not segment the background and the grass very well.

C. *Question 3 - Apple Detection*

For the last question we were given 5 images that contains different fruits and we are asked to detect only apples from these images with their original color. Therefore, we created an apple detection algorithm.

We first tried to detect apples by looking into the gray image of the original image by using *rgb2gray(..)* as we did in Question 1. However in this example apples and the some other fruits have nearly the same gray values so we couldn't detect the apples by using this way.

Then after making research, we figured out that using HVS instead of RGB can be more effective. This way we were able to separate yellow from red and green more easily.

We decided that using colors to write a generic algorithm for apple detection is going to be harder than we expected, so we make some research and find out that we can use shapes to detect images.

We write a mask function that takes the image, number of apples in the image, radius interval and finds the apples. First we detect edges from the gray image using *edge(image,'sobel')* and to make edges more definite we use *imfill(image,'holes')* and *bwareaopen(image,20)* functions. After that we use *imfindcircles* to detect circles in the image, which returns center, radius and metric of the circles that has the radius between the given interval.

After finding circles as many as the apple count in the pictures, we create a new image, color it using the colors and coordinates same as the apples in the original image have.

We get better results than when we try to detect apples by colors this way. Problems that we face with using this method are, first is that C4 take a little while to compile because it is a big image and we create a new image with C4 as an reference. The second problem is that, for C5 our algorithm can only detect 5 of the apples and one circle where not an apple exist.

III. Conclusion

With this homework, we detected objects and count them from given images, we used existing image segmentation methods and learned how they works, and we created an apple detection mask. We learned how to develop object detection algorithms by using Matlab.

For the first question, we were asked to detect flying jets from given images. We used the given images gray values and separated the unwanted objects by looking into pixels' gray values. Even though, some unwanted objects are eliminated after using gray value limitations, there were still some unwanted objects on the image. We used different approaches to overcome this problem. For some images we used *imclose(..)*, or *bwareaopen(..)* functions to remove unwanted parts. After this processes we obtained the flying jets as we aimed.

For the question 2, we used 2 different existing image segmentation algorithms on given Berkeley Dataset. After making research we found the two algorithms that we explained in the related section. By looking into the resulted segmented images we understood that the first algorithm we found is better than the second in terms of image segmentation quality.

For the final question, we generated an apple detection mask. While developing this mask we first tried to detect the apples' color and then separate the from the image. However, this is not resulted as we expected because some other fruits' pixels have similar colors as apples. Then we tried to detect the circles in the given images. We created a new generic function for detecting the apples in all five images. This mask function takes the circle's radius range and the apple count, and returns the circles' center coordinates and their radius. By looking into these given values we detected the apples from the given images.

References

[1] Athi. (2011, Jun). Color Image Segmentation. [Online] from https://www.mathworks.com/matlabcentral/fileexchange/25257-color-image-segmentation

[2] Imfindcircles. [Online] from https://www.mathworks.com/help/images/ref/imfindcircles.

[3] Detect and Measure Circular Objects in an Image. [Online] from https://www.mathworks.com/help/images/detect-and-measure-circular-objects-in-an-image.html

[4] Edge. [Online] from https://www.mathworks.com/help/images/ref/edge.html