İhsan Doğramaci Bilkent University

Introduction to Computer Vision
CS 484

Assignment: Homework #1

Authors:

Yusuf Dalva
Bilkent ID: 21602867

March 6, 2020

Contents

Question 1 **3**

Question 2 **5**

Question 3 **9**

Question 4 **12**

Image 3a.png 12

Image 3b.png 14

References **17**

Question 1

As the question implicitly states that usage image processing learning libraries are allowed for this assignment, the implementation provided for this question follows a brute force approach. To be precise, to create the histogram the script provided skims through the image inputted and counts the occurrences of each image pixel[1], which is in the range **[0,255]**. The program works on a counter based logic and after the counting operation is performed the histogram representing the intensity of the pixels in the mentioned range is provided. The code for this question is provided below (*histogram.m*):

```
function [] = histogram(img)
    dim = size(img);
    hist = zeros(1,256);
    for x = 1:dim(1)
        for y = 1:dim(2)
            hist(1,img(x,y) + 1) = hist(1,img(x,y) + 1) + 1;
        end
    end
    px = (0:255);
    bar(px, hist);
end
```

As it can be understood by the function declaration, the function returns nothing and takes an input image to be analyzed. Before starting the counting operation, at the initialization stage all of the occurrence counts are considered as 0 and then the counting operation begins. As the bar chart that is going to be plotted will not be a graph of a continuous function, the bar chart is selected and the visualization is performed. Due to the similarities in the function signature, to prevent any confusion the **implot** method has been selected to verify the function written. As a result of the execution the following histograms are obtained for the first image (*1a.png*), for clarity the image itself is provided first:

Figure 1: Image *1a.png* which was an image from the dataset provided for the assignment

The histogram obtained by the implementationn:

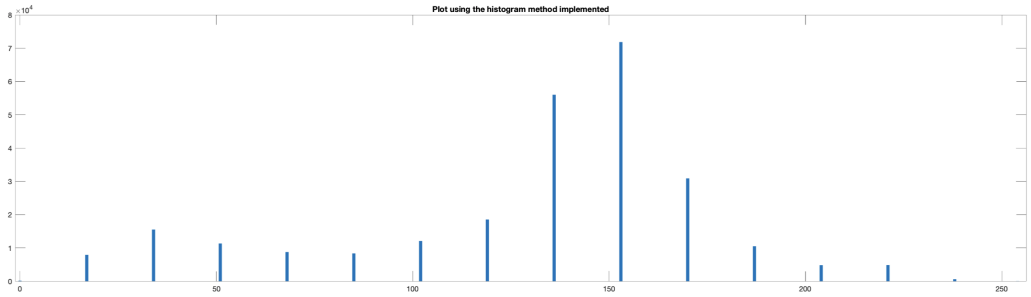

Figure 2: Histogram generated by the implementation for the assignment for image 1a.png

The first figure is provided by the implementation provided in this section and the figure generated by the imhist method is given as the second plot. In the same way the second image provided for this question with the respective plots is given below (*1b.png*):

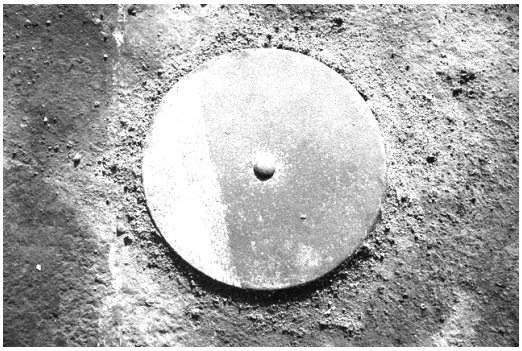

Figure 3: Image *1a.png* which was an image from the dataset provided for the assignment

The histogram obtained by the implementation:

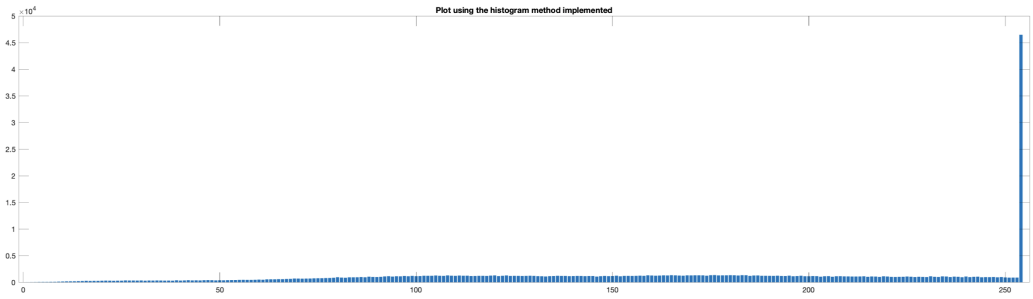

Figure 4: Histogram generated by the implementation for the assignment for image 1b.png

Note that in order to clarify that the method is correct, the second plot is visualized in a way that two graphs are overlapping, which clearly shows that they are identical in behaviour.

Question 2

In order to solve the automated thresholding problem, Otsu's method has been implemented as mentioned in the problem statement. Before providing and explaining the function implemented, the ideas and mathematical background behind Otsu's thresholding method will be explained.
Otsu's idea depends on setting a threshold that minimizes the variance between the pixels that are from the same class[2]. As the most general form, the within group variance that is going to be minimized is shown by the following equation:

$$\sigma_I{}^2 = q_1(t) * \sigma_1{}^2 + q_2(t) * \sigma_2{}^2 \tag{1}$$

For *eqn. 1*, the symbol $\sigma_I{}^2$ represents the variance between two classes, where $\sigma_1{}^2$ represents the variance between the first class of pixels and $\sigma_2{}^2$ is for the second class. Finally the numbers used ($q_1(t)$ and $q_2(t)$) for the weighted variance calculation shows the cumulative probability of the occurrence of the pixels which are shown below:

$$q_1(t) = \sum_{i=1}^{t-1} P(i) \tag{2}$$

$$q_2(t) = \sum_{i=t}^{L} P(i) \tag{3}$$

For equations 2 and 3, the variable P(i) shows the occurrence probability of pixel i in the input image and L represents upper limit of pixels, which is 255 here. For the calculation of the variable given in equation 1, the following equation is going to be used:

$$\sigma_1^2 = \sum_{i=0}^{t-1} [i - \mu_1(t)]^2 P(i)/q_1(t) \tag{4}$$

$$\sigma_2^2 = \sum_{i=t}^{L} [i - \mu_2(t)]^2 P(i)/q_2(t) \tag{5}$$

Considering these equations, equation 1 can be reconstructed in a way which is the following:

$$\sigma^2 = \sum_{i=0}^{t-1} [i - \mu_1(t)]^2 P(i) + [\mu_1(t) - \mu]^2 q_1(t) \tag{6}$$

Doing the necessary simplifications:

$$\sigma^2 = \sigma_I^2 + q_1(t)[1 - q_1(t)][\mu_1(t) - \mu_2(t)]^2 \tag{7}$$

As it can be observed from equation 7, in order to minimize the term σ_I^2 the term $q_1(t)[1-q_1(t)][\mu_1(t)-\mu_2(t)]^2$ is going to be maximized. By using this approach, all of the possible pixels starting from 0 to 255 will be attempted as a threshold value and a t value will be obtained which corresponds to the optimal separation. The implementation of the Otsu's Thresholding algorithm is given below (*otsu_thresholding.m*):

```
function[binary_image] = otsu_threshold(source_image)
    %% First step is to compute the histogram
    dim = size(source_image);
    hist = zeros(1,256);
    for x = 1:dim(1)
        for y = 1:dim(2)
            hist(1,source_image(x,y) + 1) = hist(1,source_image(x,y) + 1) + 1;
        end
    end
    %% Computing the mean and variance of the image
    prob_hist = hist;
    img_sum = 0;
    dist_dim = size(prob_hist);
    disp(dist_dim);
    for x = 1:dist_dim(2)
        img_sum = img_sum + (x - 1) * prob_hist(x);
    end
    %% Computing moving variances (for minimum within class variance)
    t = 0;
    q_1 = 0; \% q_1(t) -- initially q_1(0) = 0
    sum_1 = 0;
    m_1 = 0; \% m_1(t) -- initially m_1(0) = 0
    m_2 = 0;
    var = (q_1 * (1 - q_1) * (m_1 - m_2)^2);
    for x = 1:dist_dim(2)
        q_1 = q_1 + prob_hist(x);
        if q_1 == 0
            continue;
        end
        q_2 = (dim(1) * dim(2)) - q_1;
        if q_2 == 0
            break;
        end
        sum_1 = sum_1 + ((x - 1) * prob_hist(x));
        m_1 = sum_1 / q_1;
        m_2 = (img_sum - sum_1) /q_2;
        t_var = (q_1 * q_2 * (m_1 - m_2)^2);
        if t_var > var
            var = t_var;
            t = x - 1;
        end
    end
    disp(t);
    %% Change the input matrix according to the threshold value
    dim = size(source_image);
    binary_image = zeros(dim(1), dim(2));
    for x = 1:dim(1)
        for y = 1:dim(2)
            if source_image(x,y) < t
                binary_image(x,y) = 0;
            else
                binary_image(x,y) = 255;
            end
        end
```

```
    end
end
```

By using this function two images given for this question are thresholded. The results of this operation is given below:

- **Image 2a.png**:

Figure 5: Image 2a.png provided as a part of the dataset for the assignment

The resulting image after applying Otsu's Thresholding algorithm is as follows:

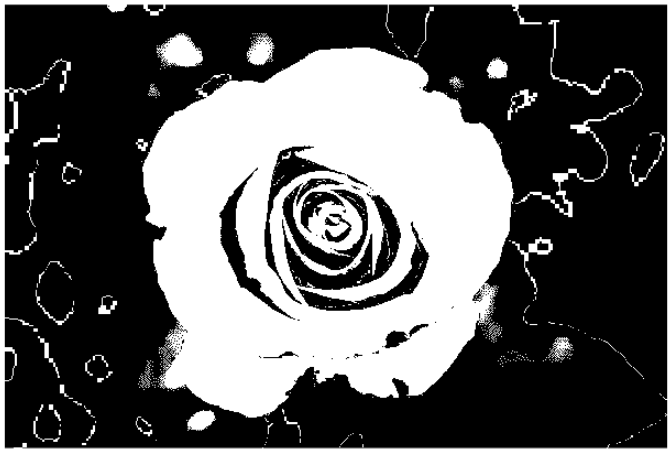

Figure 6: Image 2a.png after applying Otsu's Thresholding algorithm

- **Image 2b.png**:

Figure 7: Image 2b.png provided as a part of the dataset for the assignment

The resulting image after applying Otsu's Thresholding algorithm is as follows:

Figure 8: Image 2b.png after applying Otsu's Thresholding algorithm

As it can be observed from the images, Otsu's thresholding algorithm is not able to distinguish the image into two classes by finding a threshold value in a automated way. In order to observe the reason of this problem the pixel intensity distriburtion of the images are observed. The histograms representing the pixel intensities of the images are provided below:

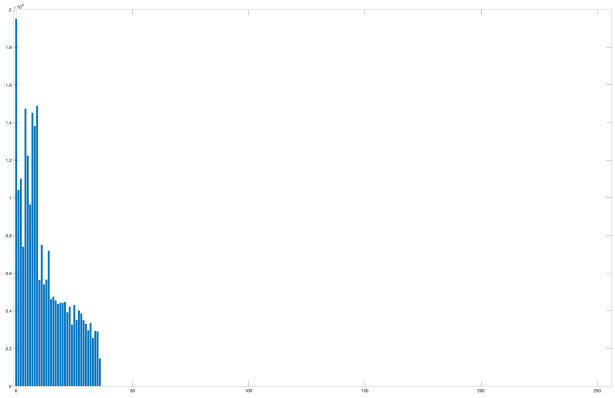

Figure 9: Histogram representing the pixel intensity of image 2a.png

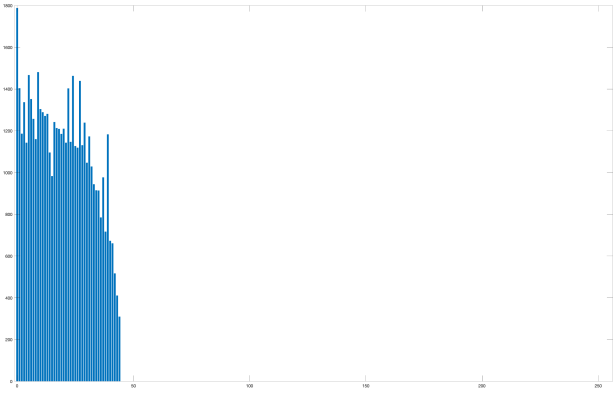

Figure 10: Histogram representing the pixel intensity of image 2b.png

As it can be observed from figure 9 & 10, the variance between the intensity of the pixels is not that distinguishable for image 2b.png. Due to this fact, the selected point (threshold) does not separate the pixels such that the different pixel clusters that are showing some certain areas are distinguished. In other words, since the pixel intensity values for the second image are really close to each other, the probability of getting some pixel when a random location is selected from the image is more close to equal compared to the first image. Due to that fact, the weighted calculation mentioned by equation 1 is not able to separate the clusters in an meaningful way. As a result of this, thresholding performs better for image 2a.png than 2b.png. This conclusion supports the idea explained considering the histograms given by figure 9 and 10.

Question 3

For this question, the morphological operations erosion and dilation are implemented. The implementation for dilation operation is given below(*dilation.m*):

```
function[dilated_image] = dilation(source_image, struct_el)
    dim = size(source_image);
    dilated_image = zeros(dim(1), dim(2));
    struct_dim = size(struct_el);
    %% Zero padding the input image
    % Padding in order to apply the window to every pixel
    input_img = padarray(source_image, [floor(struct_dim(1)/2),
        floor(struct_dim(2)/2)], 0, "both");
    pad_dim = size(input_img);
```

```
    %% Traversing through image pixels
    for x = ceil(struct_dim(1)/2): pad_dim(1) - floor(struct_dim(1)/2)
        for y = ceil(struct_dim(2)/2): pad_dim(2) - floor(struct_dim(2)/2)
            % If there is a hit in the image by the structuring element,
            % pixel (x,y) set to 1
            x_bound = 0;
            y_bound = 0;
            if mod(size(struct_el, 1), 2) == 1
                x_bound = [x - floor(struct_dim(1) / 2),
                    x + floor(struct_dim(1) / 2)];
            else
                x_bound = [x - ceil(struct_dim(1) / 2) + 1,
                    x + ceil(struct_dim(1) / 2)];
            end
            if mod(size(struct_el,2),2) == 1
                y_bound = [y - floor(struct_dim(2) / 2),
                    y + floor(struct_dim(2) / 2)];
            else
                y_bound = [y - ceil(struct_dim(1) / 2) + 1,
                    y + ceil(struct_dim(1) / 2)];
            end
            window = input_img(x_bound(1): x_bound(2), y_bound(1): y_bound(2));
            flag = 0;
            for i = 1:size(window, 1)
                for j = 1:size(window,2)
                    if struct_el(i,j) == 1 && window(i,j) == 1
                        dilated_image(x - ceil(struct_dim(1) / 2) + 1,
                            y - ceil(struct_dim(2) / 2) + 1) = 1;
                        flag = 1;
                        break;
                    end
                end
                if flag == 1
                    break;
                end
            end
            if flag == 0
                dilated_image(x - ceil(struct_dim(1) / 2) + 1,
                    y - ceil(struct_dim(2) / 2) + 1) = 0;
            end
        end
    end
end
```

According to the definition of the dilation operation, the image has been traversed in terms of windows that is equal to the size of the structure element, which can be considered as a mask. Since all of the image pixels will be traversed and due to the nature of the image doing that operation is not possible, padding method has been used. In the dilation operation, it is enough to identify one pixel in the window which is one that shares the same locality with any pixel 1 in the structure element[2]. Tracking this is done by the double for loop that traverses the window cropped. Since finding a single 1 in the window is enough in dilation, the padding is considered as 0 padding, in order not to effect the results of the operation. For this assignment, the center of the structure element is taken as the origin, which explains the operation of division by 2. Depending on whether the dimension of the structure element is odd or even, the window cropping process changes, due to that these two cases are taken as two separate cases and implementation is done accordingly.

The erosion operation is implemented by the same way and the implementation for the function is given below(*erosion.m*):

```
function[eroded_image] = erosion(source_image, struct_el)
    dim = size(source_image);
```

```
    eroded_image = zeros(dim(1), dim(2));
    struct_dim = size(struct_el);
    %% One padding to the input image
    input_img = padarray(source_image, [floor(struct_dim(1) / 2),
        floor(struct_dim(2) / 2)], 1, "both");
    pad_dim = size(input_img);
    %% Traversing through image pixels
    for x = ceil(struct_dim(1)/2):pad_dim(1) - floor(struct_dim(1)/2)
        for y = ceil(struct_dim(2)/2):pad_dim(2) - floor(struct_dim(2)/2)
            % If there is a fit between the image part and the structuring
            % element, pixel set to 1
            x_bound = [];
            y_bound = [];
            if mod(size(struct_el, 1), 2) == 1
                x_bound = [x - floor(struct_dim(1) / 2),
                    x + floor(struct_dim(1) / 2)];
            else
                x_bound = [x - ceil(struct_dim(1) / 2) + 1,
                    x + ceil(struct_dim(1) / 2)];
            end
            if mod(size(struct_el,2),2) == 1
                y_bound = [y - floor(struct_dim(2) / 2),
                    y + floor(struct_dim(2) / 2)];
            else
                y_bound = [y - ceil(struct_dim(2) / 2) + 1,
                    y + ceil(struct_dim(2) / 2)];
            end
            window = input_img(x_bound(1): x_bound(2), y_bound(1): y_bound(2));
            flag = 0;
            for i = 1:size(window, 1)
                for j = 1:size(window,2)
                    if struct_el(i,j) == 1 && window(i,j) ~= 1
                        eroded_image(x - ceil(struct_dim(1) / 2) + 1,
                            y - ceil(struct_dim(2) / 2) + 1) = 0;
                        flag = 1;
                        break;
                    end
                end
                if flag == 1
                    break;
                end
            end
            if flag == 0
                eroded_image(x - ceil(struct_dim(1) / 2) + 1,
                    y - ceil(struct_dim(2) / 2) + 1) = 1;
            end
        end
    end
end
```

Just like for the dilation operation, the erosion operation does the traversing by cropping windows. The same padding methodology is used here too, since the natural boundaries of the image will not allow cropping the windows that contain in the boundaries of the image. In the erosion operation, the complete fit of the 1's in the structure element to the cropping window is deserved. In case that even a single pixel with value 0 sharing the same locality value with a 1 in the structure element, the result of the erosion operation is counted as 0. In order to provide a padding that does not effect the result of the erosion operation, the padding element is taken as 1. Since the complete fit is deserved, this kind of padding does not effect the result of the erosion operation. The calculation of the erosion result for a selected window is shown in the double for loop structure that traverses through the window. Just like the dilation operation, the erosion operation also separates the

cases where the dimension of the structure element is even or odd. These are taken as two separate cases, which is implemented by the if statement at the beginning of the nested loop structure.

For the construction of the resulting image, the dimension is identical as the input image and the padding added is not considered. The operation that considers the setting the pixels operation for the resulting image that considers the padding added and the pixels selected for the window which the operation is applied.

Question 4

The operations done for the images will be explained in two sections, where each section is dedicated to one image.

Image 3a.png

For this question, in addition to structuring elements like square and a straight line, some additional elements are defined. The elements defined are given below:

```
diag_5 = [1 0 0 0 0;
          0 1 0 0 0;
          0 0 1 0 0;
          0 0 0 1 0;
          0 0 0 0 1];
star_filter_3 = [0 1 0;
                 1 1 1;
                 0 1 0];
star_filter_5 = [0 0 1 0 0;
                 0 1 1 1 0;
                 1 1 1 1 1;
                 0 1 1 1 0;
                 0 0 1 0 0];
plus_filter = [0 0 1 0 0;
               0 0 1 0 0;
               1 1 1 1 1;
               0 0 1 0 0;
               0 0 1 0 0];
diagonal = [1 0 0;
            0 1 0;
            0 0 1 ];
se_1 = [1 1 0;
        0 1 1];
se_2 = [0 1 0;
        1 0 1;
        0 1 0];
```

For the thresholding part for this image, initially Otsu's thresholding algorithm is attempted. However, after inspecting the image and the histogram values, in order to eliminate the noise some other values are tried. As a result of the trial & error method, a good threshold has been defined as 70. The hishogram of the image is given below:

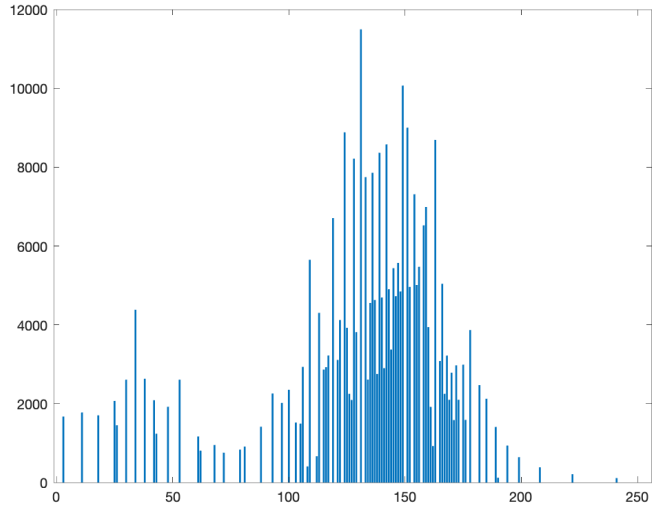

Figure 11: Histogram for image 3a.png

After applying the threshold value the following image has been obtained:

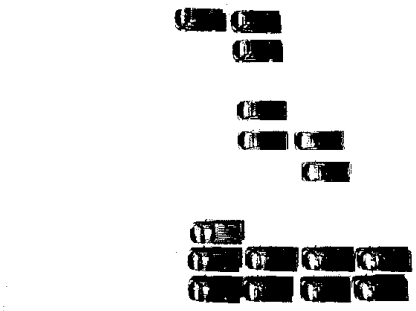

Figure 12: Thresholded version of image 3a.png

After deciding on the threshold for the image, the necessary dilation and erosion operations are performed. The sequence of the operations are given below:

```
res = dilation(~imbinarize(binary_image), star_filter_3);
res = erosion(res, star_filter_3);
res = dilation(res, ones(2,2));
res = erosion(res, ones(3,3));
res = dilation(res, ones(3,3));
res = erosion(res, ones(5,2));
```

Where res shows the resulting image for a particular moment. In order to cut the connection between cars first the erosion operation has been applied. Since the intersection of two distinct cars do not have a certain angle (which can clearly be observed in the cars at the bottom part of the image) a star shaped filter is used. After that, in order to recover from the extra area generated for the cars (which makes the cars even bigger), the closing operation has been performed. Finally, since some intersecting parts between cars, a triangular shaped filter has been applied in order to separate the cars that seem together as two horizontal lines. As a result of these operations, there are 12 areas that are separable. In the image, there are 16 cars present. At this instant, a certain trade off has been considered. By manipulating the image even more into almost rectangular operations, the cars were completely separable when opening and losing operations are repeated one after another. In order

to stick with the car characteristics, these way is not followed. There may be some way that can successfully separate cars by securing the characteristics but by experimentally it could not be found. The final results are given below:

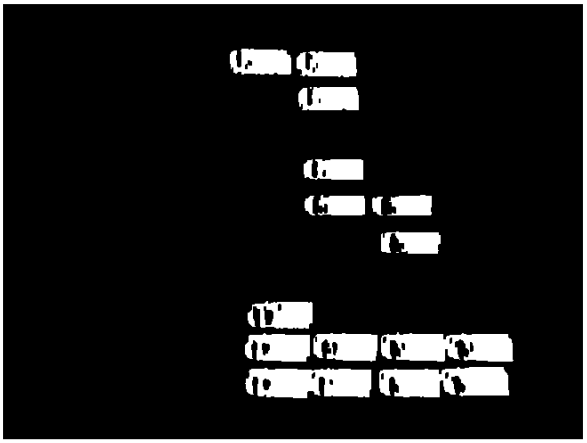

Figure 13: Image 3a.png after applying erosion and ilation operations

The labeled version:

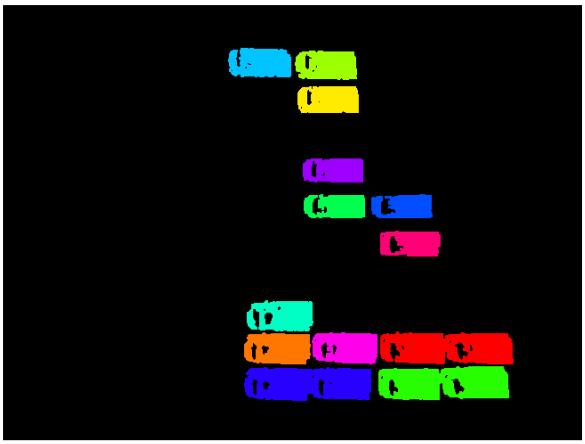

Figure 14: Labeling of image 3a.png

Image 3b.png

For this questions thresholding operation, Otsu's thresholding gave fairly good results in terms of separation. The verification of this result has been determined by using trail and error method. If the threshold has been set higher, preserving the characteristics of the planes are nearly impossible. Due to this fact, the thresholding value found by Otsu's algorithm has been used. The value found was 14. The thresholded image is given below:

Figure 15: Thresholded version of image 3b.png

In addition to the extra filters introduced in the previous part, two new structuring elements are defined which are given below:

```
rev_diagonal = [0 0 0 0 1;
                0 0 0 1 0;
                0 0 1 0 0;
                0 1 0 0 0;
                1 0 0 0 0];
cross = [1 0 0 0 1;
         0 1 0 1 0;
         0 0 1 0 0;
         0 1 0 1 0;
         1 0 0 0 1];
se = [1 1;
      0 1];
se_1 = [1 0; 1 1];
```

By using these structuring elements, the following operations are applied to the image:

```
res_b = imbinarize(binary_image_b);
res_b = erosion(res_b, star_filter_3);
res_b = dilation(res_b, star_filter_3);
res_b = dilation(res_b, ones(2,2));
res_b = erosion(res_b, rev_diagonal);
res_b = dilation(res_b, rev_diagonal);
res_b = dilation(res_b, diagonal);
res_b = erosion(res_b, se_1);
```

As the interactions of the planes with the lanes present do not follow any arbitrary shape, an opening operation that makes planes significant has been done with a star shaped structuring element. After that, In order to fill in the planes as much as possible, a square filter has been used. In the same way with the star filter, in order to expand the planes further, the element hat is an inverse diagonal has been used considering the shape of the lanes. Finally, as an attempt to eliminate the dotted lanes, the structural element defined as a corner has been used. After using these operations some of the planes were separable. However, the presence of the lanes in the image makes is incredibly hard for processing. With simple filters, the elimination of this noise was not a successful attempt. As a result of this bias, it is concluded that some more advanced processing techniques are required for this kind of processing different than morphological operations, or some way of optimizing the structural elements is required (There may be an usage of deep learning applications here). The images obtained as the result of the operations applied is shown below:

Figure 16: Image 3b.png after applying erosion and dilation operations

The corresponding labels for the image is given in the image below:

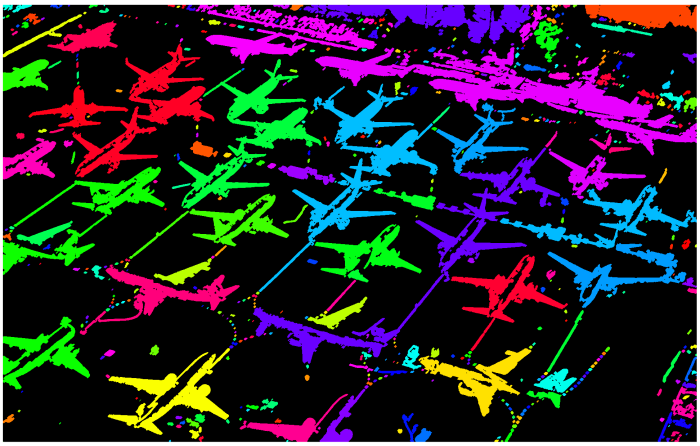

Figure 17: Labeling for image 3a.png

As it can be observed here, identifying the clusters that correspond to planes is possible. However, the noise element(lanes) create insensible amount of new labels as they are separated from the planes and not connected to any other piece in the image. Several combinations has been attempted and removal of these lanes was not possible.

References

Fisher, Perkins, W., & Wolfart. (2000). *Intensity histogram.* Retrieved 2020-03-04, from https://homepages.inf.ed.ac.uk/rbf/HIPR2/histgram.htm

Shapiro, L., & Stockman, G. (2001). *Computer vision.* Pearson.