FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Multimedia and New Services

## Lab 3 - Image Processing in Matlab

**Mário Santos**
**up201503406**

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

February 1, 2021

# Multimedia and New Services
# Lab 3 - Image Processing in Matlab

**Mário Santos**
**up201503406**

Mestrado Integrado em Engenharia Informática e Computação

February 1, 2021

# Contents

# List of Figures

# List of Tables

# Chapter 1

# 1. Blue Background Extraction

In this chapter we will discuss the multiple scripts written for the purpose of splitting an image into RGB channels, creating a mask based on the information of the blue channel, and using this mask to extract images.

## 1.1 Basic Segmentation

Based on the script ***segmentBB.m*** and the set of Matlab built-in functions already learnt, we wrote a Matlab script that:

- Imports a coloured image with a blue background and presents that image on the screen
- Separates each RGB component in a different matrix and visualises each one on the screen
- Displays the B channel histogram on the screen using the built-in function imhist for the user to decide on a suitable threshold value
- Uses the matrix with the B component to identify the foreground objects. One possibility for doing this is to
- Asks the user to input the threshold value
- Using that threshold, it copies from the B matrix to a new matrix (with the same dimensions) only the pixel values that are below that threshold. Putting those pixels with the value 255 and all the others with value 0, creating a black and white picture
- Shows the black and white segmented image on the screen and creates in the disk a new file with that image

The following is the code for the script that was written, *simple_segment.m*:

```
function simple_segment (inputImg)

% Imports a coloured image with a blue background and presents that image on the
% screen;

img = imread(inputImg);
if size(img,3) ~= 3
    img=cat(3,img,img,img);
end

figure(1), imshow(img),title('Original')

% get image dimensions: an RGB image has three planes
[height, width, planes] = size(img);

% Separates each RGB component in a different matrix, visualises each one on the
% screen and writes to disk;

if(planes==3)
    r = img(:, :, 1);         % red channel
    g = img(:, :, 2);         % green channel
    b = img(:, :, 3);          % blue channel
end

figure(2), imshow(r),title('Red Channel');
filepath = append('figures/',extractBefore(inputImg, '.'),'_red_channel.png');
imwrite(r,filepath);

figure(3), imshow(g),title('Green Channel');
filepath = append('figures/',extractBefore(inputImg, '.'),'_green_channel.png');
imwrite(g,filepath);

figure(4), imshow(b),title('Blue Channel');
filepath = append('figures/',extractBefore(inputImg, '.'),'_blue_channel.png');
imwrite(b,filepath);

% Generate the histogram of the Blue Channel with the built-in function imhist

figure(5),imhist(b),title('Blue Channel Histogram');
filepath = append('figures/',extractBefore(inputImg, '.'),'_blue_channel_histogram.png');
saveas(gcf, filepath)

% Ask the user to input the threshold value

threshold = input('Input threshold:');
```

```

% Get image dimensions

[height,width,~] = size(img);

% Uses the matrix with the B component to identify the foreground objects (the jumping
%man or the Christmas bulbs or the bird). Using a threshold, copy from the B matrix to a new matrix only
%the pixel values that are below that threshold.
%When doing that you may put those pixels with the value 255 and all the others with value 0 (you will create a black and white
%picture).

BWforeground = zeros(height,width);
for i = 1:height
    for j = 1:width
        if(b(i,j)<threshold)
           BWforeground(i,j)=255;
        end
    end
end

% Shows the black and white segmented image on the screen and creates in the disk a new file with that image.

figure(6),imshow(BWforeground),title('Segmented Image');
filepath = append('figures/',extractBefore(inputImg, '.'),'_',int2str(threshold),'_segmented.png');
imwrite(BWforeground,filepath);

% Make experiments with different threshold values and with different images.

threshold = input('Input higher threshold:');

BWforeground = zeros(height,width);
for i = 1:height
    for j = 1:width
        if(b(i,j)<threshold)
            BWforeground(i,j)=255;
        end
    end
end

figure(7),imshow(BWforeground),title('Segmented Image (High Threshold)');
filepath = append('figures/',extractBefore(inputImg, '.'),'_',int2str(threshold),'_segmented_high.png');
imwrite(BWforeground,filepath);

```

```
threshold = input('Input lower threshold:');

BWforeground = zeros(height,width);
for i = 1:height
    for j = 1:width
        if(b(i,j)<threshold)
            BWforeground(i,j)=255;
        end
    end
end

figure(8),imshow(BWforeground),title('Segmented Image (Low Threshold)');
filepath = append('figures/',extractBefore(inputImg, '.'),'_',int2str(threshold),'
    _segmented_low.png');
imwrite(BWforeground,filepath);

close all;
```

And the following were the results obtained from experimenting with this script on the images supplied:

(a) Original image

(b) Red Channel

(c) Green Channel

(d) Blue Channel

Figure 1.1: Original image and RGB channels

We can observe that the brightest pixels in the Blue channel correspond to the blue background, the sky. However, we can also note that part of the bird and bird feeders' are also bright in this channel, which might lead to an erroneous removal of those pixels when segmenting the image.

The following histogram was generated for this images' Blue channel.

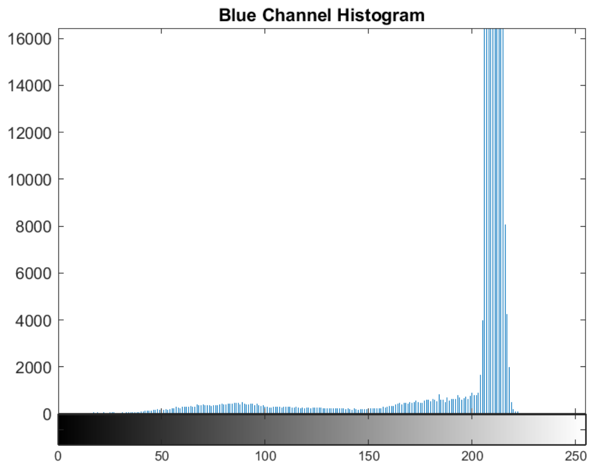



Figure 1.2: Blue Channel Histogram

We can observe that the zone with the highest density of blue pixels is around 200Hz. Based on this we experimented with several threshold values, obtaining the following results:

(a) Foreground image using lower than suitable threshold (190)

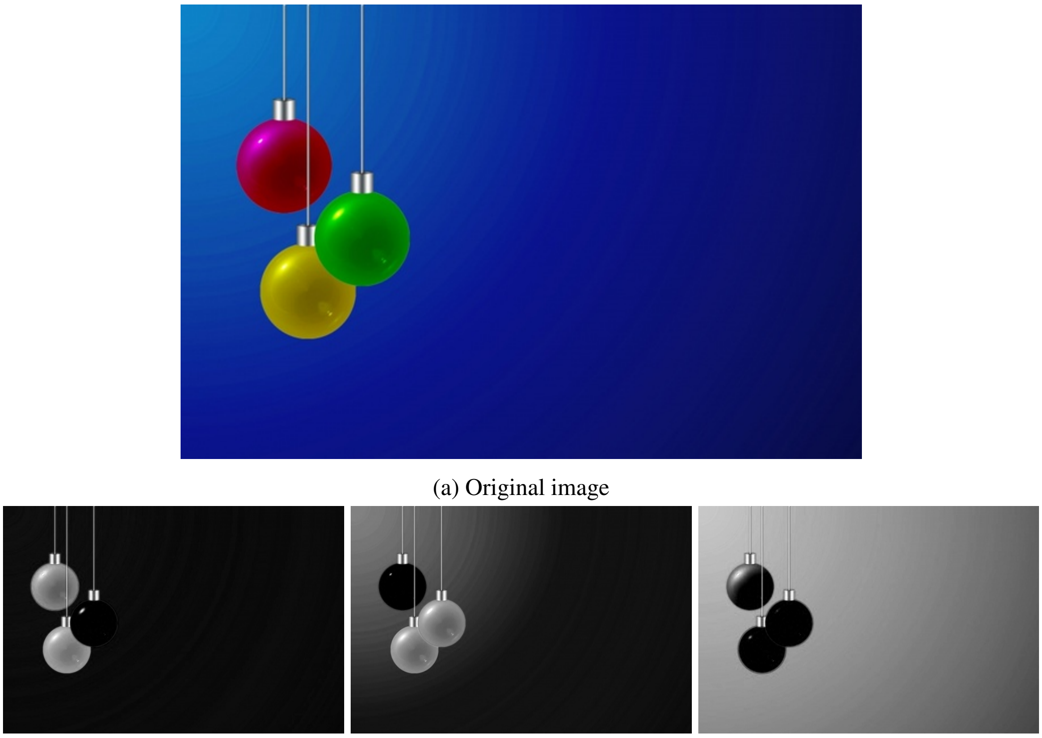
(b) Foreground image using suitable threshold (204)

(c) Foreground image using higher than suitable threshold

Figure 1.3: B&W Foreground Segmented Images using different threshold values (210)

Using a threshold value of 190Hz, the script successfully removes the blue background, however it also removes parts of the bird, namely the wings. When using a value of 204Hz, we are able to isolate the bird and the bird feeder. Using a slightly higher value, 210Hz, we no longer isolate the bird and the bird feeder and include some of the darker parts of the blue background, resulting in an unclear and noisy image.

Now looking at image ***christmasBB.jpg***, we can observe that the brightest pixels in the Blue channel also correspond to the blue background, the sky. Similarly to the previous image, part of the foreground is also bright in the Blue channel, particularly the reflection of light on the Christmas balls.



(a) Original image

(b) Red Channel

(c) Green Channel

(d) Blue Channel

Figure 1.4: Original image and RGB channels

The following histogram was generated for this images' Blue channel.

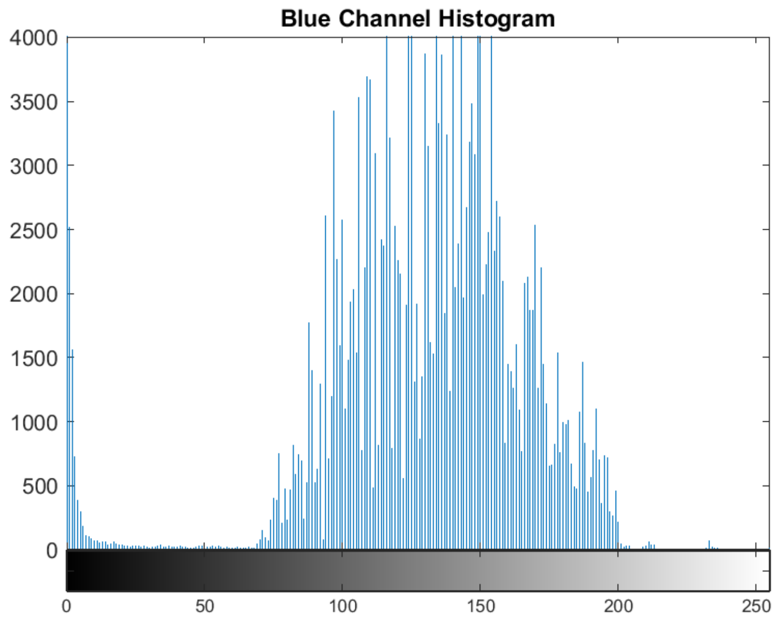
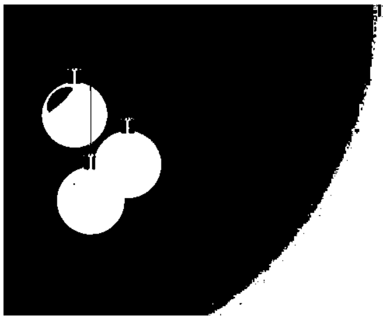



Figure 1.5: Blue Channel Histogram

In this case, we can see the highest density of blue pixels is ranged between approximately 75Hz and 200Hz. Based on this we experimented with several threshold values, obtaining the following results:

(a) Foreground image using lower than suitable threshold (70)

(b) Foreground image using suitable threshold (94)



(c) Foreground image using higher than suitable threshold (122)

Figure 1.6: B&W Foreground Segmented Images using different threshold values

Using a threshold value of 70Hz, the script successfully removes the blue background, however it also removes parts of the Christmas balls, namely the reflections. When using a value of 94Hz, we are able to isolate the foreground more clearly, however parts of the background are also erroneously detected on the right edge of the image. Using a slightly higher value, 122Hz, we achieve an almost perfect outline of the Christmas ball, however almost half of the background is also erroneously detected as foreground.

Now looking at image ***jump.jpg***, we can observe that the brightest pixels in the Blue channel correspond to the white mountain. The pixels representing the sky are also bright however they are darker than the mountain.

(a) Original image

(b) Red Channel (c) Green Channel (d) Blue Channel

Figure 1.7: Original image and RGB channels

The following histogram was generated for this images' Blue channel.

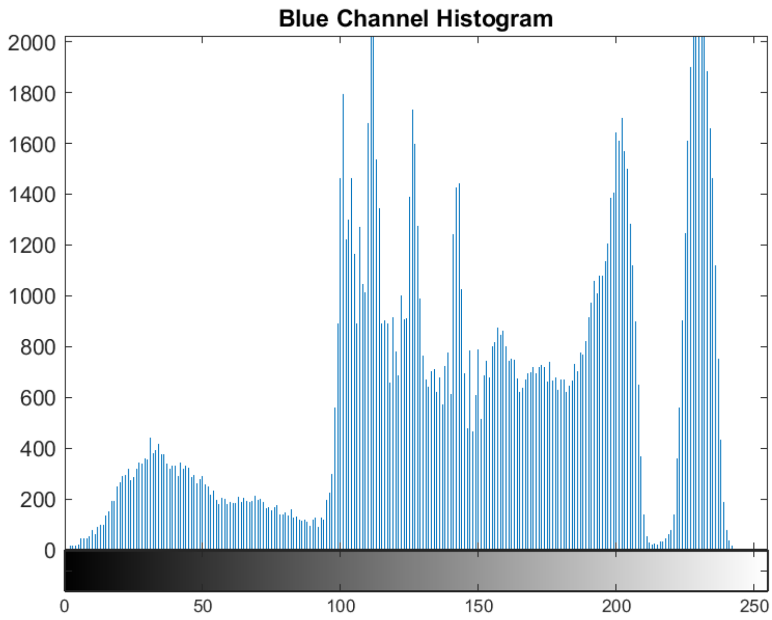



Figure 1.8: Blue Channel Histogram

In this case, we can see the highest density of blue pixels is ranged between approximately 100 to 200Hz and also between 220Hz to 240Hz. It is expected that one of these peaks corresponds to pixels in the white mountain while the other peak corresponds to the blue sky. Considering that when analysing the blue channel image the mountain is brightest, we can assume that the peak between 220Hz to 240Hz corresponds to the white mountain. Based on this we experimented with several threshold values, obtaining the following results:

(a) Foreground image using lower than suitable threshold (80)

(b) Foreground image using suitable threshold (95)

(c) Foreground image using higher than suitable threshold (220)

Figure 1.9: B&W Foreground Segmented Images
using different threshold values

Using a threshold value of 80Hz, the script successfully removes the blue sky and mountain background, however it also removes parts of the jumper, namely his torso and arms. When using a value of 95Hz, we are able to isolate the jumper more clearly, and the background is completely gone. Using a slightly higher value, 220Hz, we detect everything in the original image except for the mountain. We can thus conclude that the peak in the 220Hz to 240Hz range corresponded to the mountain.

### 1.1.1 Questions and Answers

Now that we have interpreted the results, we provide the answers to the more specific questions posed.

- **Could there be low blue value in zones of the background?**
  **Yes**, if the pixel has a blue value under the threshold that was set it will still be visible.
- **Could there be high blue values in some parts of the foreground objects?**
  **No**, if a suitable threshold is set than the pixels whose blue values are over the threshold will not be visible, regardless of whether they are in the back or foreground.
- **Is it always true that a pixel with a high value in the B component is always blue?**
  **No**, pixels with high blue values can be multiple different colors, for example, pixels with a maximum blue value (255), and maximum red and green values will result in a white pixel. This is clear when considering our experiments with the image ***jump.jpg***, where many of the high blue value pixels were in fact from the white mountain.

## 1.2 Alternative Segmentation

Based on the script *segmentBB.m* and and the set of Matlab built-in functions already learnt, we wrote a Matlab script that:

- Imports a coloured image with a blue background and presents that image on the screen
- Separates each RGB component in a different matrix and visualises each one on the screen
- Considering that a pixel is really blue if it has high values in the B component and low values in the other components, computes the "blueness" of a pixel using the equation

$$blueness = B - max(R, G)$$

- Asks the user to input the threshold value based on the blueness factor
- Creates a new black and white image with the pixels of the foreground objects with value 255 and all the other with value zero.
- Shows the black and white segmented image on the screen and creates in the disk a new file with that image

The following is the code for the script that was written, *simple_segment.m*:

```
function alternative_segment (inputImg)

% Imports a coloured image with a blue background and presents that image on the
% screen;

img = imread(inputImg);
if size(img,3) ~= 3
    img=cat(3,img,img,img);
end

figure(1), imshow(img),title('Original')

% get image dimensions: an RGB image has three planes
[height, width, planes] = size(img);

% Separates each RGB component in a different matrix, visualises each one on the
% screen and writes to disk;

if(planes==3)
    r = img(:, :, 1);          % red channel
    g = img(:, :, 2);          % green channel
    b = img(:, :, 3);           % blue channel
end

```

```
figure(2), imshow(r),title('Red Channel');

figure(3), imshow(g),title('Green Channel');

figure(4), imshow(b),title('Blue Channel');

% Calculate Blueness using formula Blueness = B - max(R,G)

blueness = b;
for i=1:height
    for j=1:width
        blueness(i,j) = b(i,j)-max(r(i,j),g(i,j));
    end
end

% Plot the histogram of the Blueness with the built-in function imhist

figure(5),imhist(blueness),title('Blueness Histogram');
filepath = append('figures/',extractBefore(inputImg, '.'),'_blueness_channel_histogram.png');
saveas(gcf, filepath);

% Ask the user to input the threshold value

threshold = input('Input threshold:');

% Uses the matrix with the B component to identify the foreground objects (the jumping
%man or the Christmas bulbs or the bird). Using a threshold, copy from the B matrix to a new matrix only
%the pixel values that are below that threshold.
%When doing that you may put those pixels with the value 255 and all the others with value 0 (you will create a black and white
%picture).

BWforeground = zeros(height,width);
for i = 1:height
    for j = 1:width
        if(blueness(i,j)<threshold)
           BWforeground(i,j)=255;
        end
    end
end

% Shows the black and white segmented image on the screen and creates in the disk a new file with that image.

figure(6),imshow(BWforeground),title('Segmented Image');
```

```
filepath = append('figures/',extractBefore(inputImg, '.'),'_',int2str(threshold),'
    _alt_segmented.png');
imwrite(BWforeground,filepath);

close all;
```

And the following were the results from experimenting with this script on the images supplied: Starting with the image *birdBB.jpg*, its Blueness histogram is depicted in the following figure:

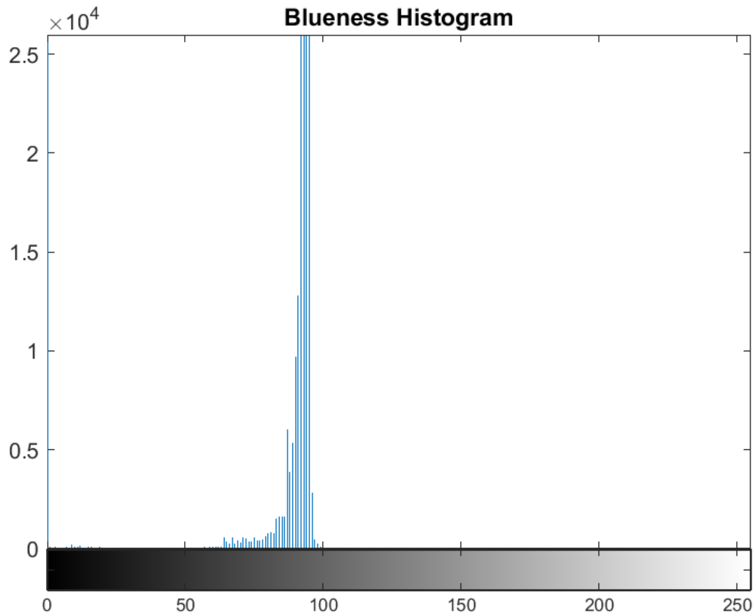



Figure 1.10: Blueness Channel Histogram

We can see that the highest density of Blueness ranges between 60Hz to 100 Hz, based on this we experimented with two threshold values and obtained the following results:

(a) Foreground image using suitable threshold (83)

(b) Foreground image using higher than suitable threshold (86)

Figure 1.11: B&W Foreground Alternative Segmented Images using different threshold values

Using a threshold value of 83Hz successfully removes the background and identifies the foreground, however parts of the birds wings were also removed. It can be observed that increasing the threshold by simply 3Hz, results in parts of the background being erroneously detected as foreground.

Now experimenting with the image *christmasBB.jpg*, its Blueness histogram is depicted in the following figure:

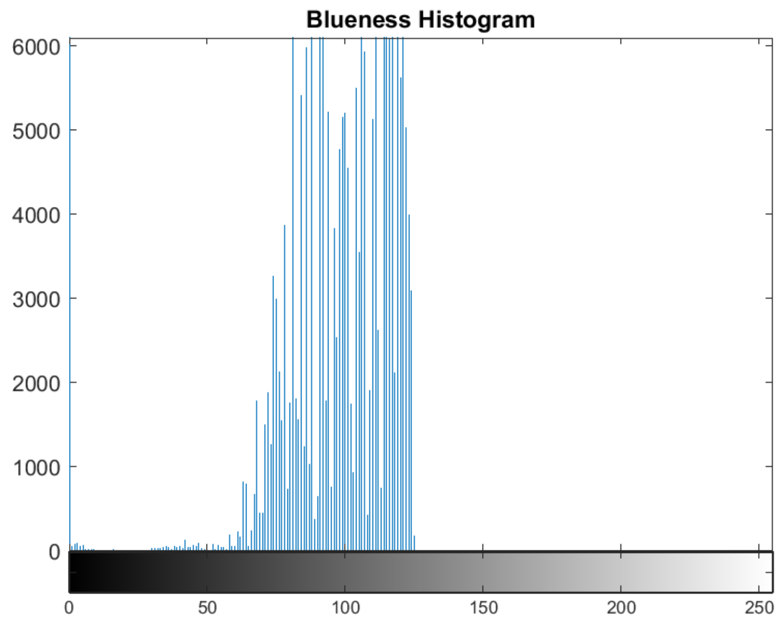



Figure 1.12: Blueness Channel Histogram

We can see that the highest density of Blueness ranges between approximately 60Hz to 130Hz, based on this we experimented with two threshold values and obtained the following results:

(a) Foreground image using suitable threshold (68)

(b) Foreground image using higher than suitable threshold (70)

Figure 1.13: B&W Foreground Alternative Segmented Images using different threshold values

Using a threshold value of 68Hz successfully removes the background and identifies the foreground, additionally, unlike the simple segment script, it successfully identified the reflections on the christmas balls as foreground, while excluding the majority of the background. It can be observed that increasing the threshold by simply 2Hz, results in parts of the background being erroneously detected as foreground (top left corner of the image).

Experimenting with the image *jumper.jpg*, its Blueness histogram is depicted in the following figure:

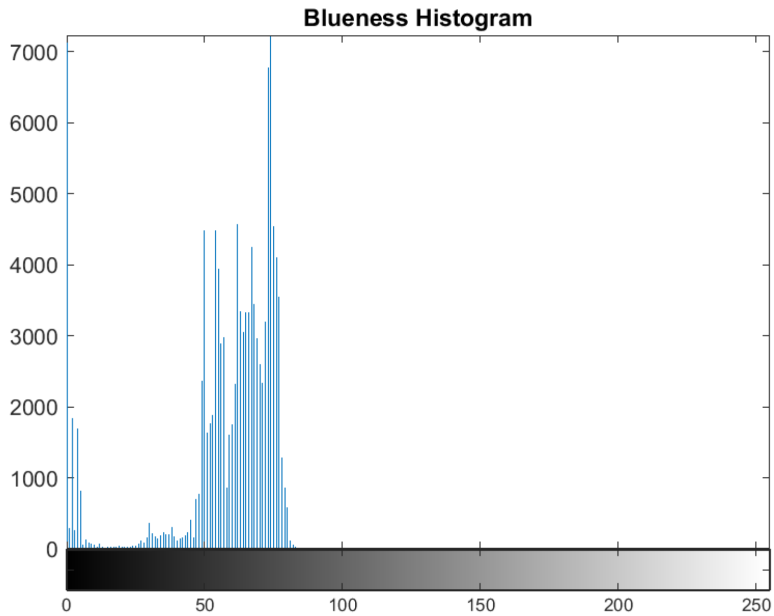
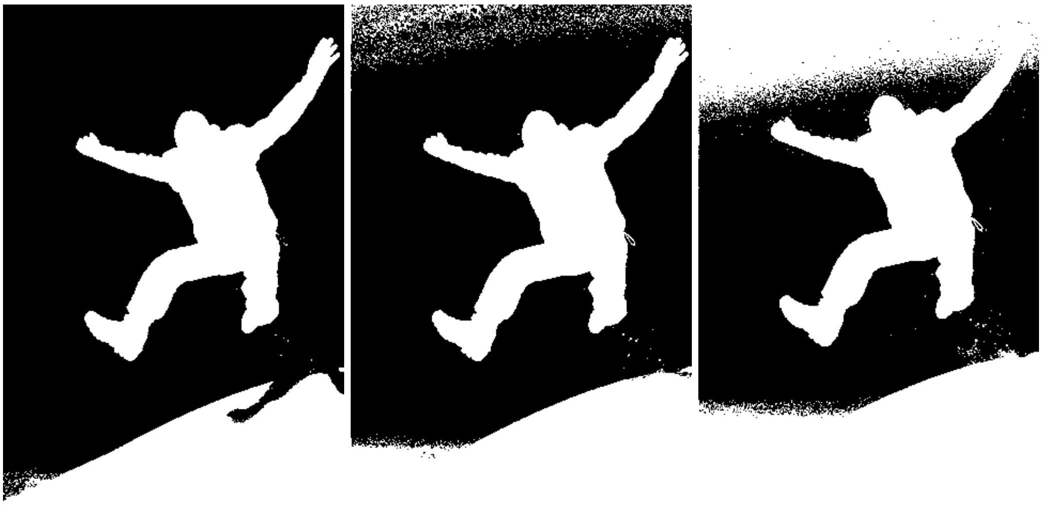



Figure 1.14: Blue Channel Histogram

We can see that the highest density of Blueness ranges between approximately 50Hz to 100Hz. We can see that in comparison to the Blue channel histogram, the Blueness histogram allows a better identification of the pixels that are closer to blue, and don't just have a high blue channel value. This allows for a better identification of the correct threshold to use to remove the actual blue pixels. Based on this we experimented with three threshold values and obtained the following results:



(a) Foreground image using threshold of 30 (30)

(b) Foreground image using threshold of 50 (50)

(c) Foreground image using lower than
threshold of (60)

Using a threshold value of 30 successfully removes the background blue sky but includes the white mountain in the foreground. Increasing the value to 50 and over simply increases the number of background pixels that are identified as foreground. In this case we were not able to achieve as good results as we did using the simple segment.

Overall the alternative segment did not prove superior to the simple segment, despite requiring more processing (calculations). In comparison to the simple segment script, it achieved a very similar result with the *birdBB.jpg* image, better results for *christmas.jpg* and regarding the *jump.jpg* image, if the mountain is to be considered as foreground, it provided a better performance, otherwise, if the goal was to isolate the jumper alone, it provided worst performance.

Figure 1.15: B&W Foreground Alternative Segmented Images
using different threshold values

The following table provides an overview of the threshold values used for each experiment:

| **Image** | **Simple Segment** | | | **Alternative Segment** | | |
|---|---|---|---|---|---|---|
| | **T1** | **T2** | **T3** | **T1** | **T2** | T3 |
| birdBB.jpg | 190 | 204 | 210 | 83 | 86 | - |
| christmasBB.jpg | 70 | 94 | 122 | 68 | 70 | - |
| jump.jpg | 80 | 95 | 220 | 30 | 50 | 60 |

Table 1.1: Overview of threshold values used
in each experiment for each image

# Chapter 2

# 2. Adding Objects to Another Image

In this chapter we will discuss the the script written for the purpose of fusing 2 images, as well as modifications to the previously written scripts so that the foreground image is colored after removing the background.

## 2.1 Superimposing Images

Based on the script “segmentBB.m” we wrote a Matlab script that superimposes 2 images, using the built-in Matlab function *imfuse*.

The following is the code written:

```
function superimpose(backgroundInput,foregroundInput)

background = imread(backgroundInput);
foreground = imread(foregroundInput);

outputImage=imfuse(background,foreground);
figure(1), imshow(uint8(outputImage)),title('Fused image');

filepath = append('figures/', extractBefore(backgroundInput, '_'),'_',extractBefore(foregroundInput,'_'),'_fused.png');

imwrite(outputImage,filepath);
```

We used the previously generated images, particularly the Christmas balls and the jumper and superimposed them, resulting in the following image:

(a) B&W Foreground of 'christmasBB.jpg'

(b) B&W Foreground of 'jump.jpg'

(c) Fused Images

Figure 2.1: Superimposed B&W images using *imfuse*

The default method for the built-in *imfuse* function is named 'falsecolor'. This generates an RGB image that displays both images superimposed in different colors. In this example we can observe that the jumper was colored in magenta and the christmas balls and remainder of the *christmasBB.jpg* image background (bottom right corner) were colored in green.

## 2.2 Colored Foreground

Based on the script *segmentBB.m* we modified the simple and alternative segment scripts, in order to generate a colored image instead of black and white.

The following is the code for *simple_segment_color.m*:

```
function simple_segment_color (inputImg)

% Imports a coloured image with a blue background and presents that image on the
% screen;

img = imread(inputImg);
if size(img,3) ~= 3
    img=cat(3,img,img,img);
end

figure(1), imshow(img),title('Original')

% get image dimensions: an RGB image has three planes
[height, width, planes] = size(img);

% Separates each RGB component in a different matrix, visualises each one on the
% screen and writes to disk;

if(planes==3)
    r = img(:, :, 1);          % red channel
    g = img(:, :, 2);          % green channel
    b = img(:, :, 3);           % blue channel
end

figure(2), imshow(r),title('Red Channel');
filepath = append('figures/',extractBefore(inputImg, '.'),'_red_channel.png');
imwrite(r,filepath);

figure(3), imshow(g),title('Green Channel');
filepath = append('figures/',extractBefore(inputImg, '.'),'_green_channel.png');
imwrite(g,filepath);

figure(4), imshow(b),title('Blue Channel');
filepath = append('figures/',extractBefore(inputImg, '.'),'_blue_channel.png');
imwrite(b,filepath);
```

```
% Generate the histogram of the Blue Channel with the built-in function imhist

figure(5),imhist(b),title('Blue Channel Histogram');
filepath = append('figures/',extractBefore(inputImg, '.'),'_blue_channel_histogram.png');
saveas(gcf, filepath)

% Ask the user to input the threshold value

threshold = input('Input threshold:');

% Uses the matrix with the B component to identify the foreground objects (the jumping
%man or the Christmas bulbs or the bird). Using a threshold, copy from the B matrix to a new matrix only
%the pixel values that are below that threshold.
%When doing that you may put those pixels with the value 255 and all the others with value 0 (you will create a black and white
%picture).

BWforeground = zeros(height,width);
for i = 1:height
    for j = 1:width
        if(b(i,j)<threshold)
           BWforeground(i,j)=255;
        end
    end
end

% Shows the black and white segmented image on the screen and creates in the disk a new file with that image.

figure(6),imshow(BWforeground),title('B&W Segmented Image');
filepath = append('figures/',extractBefore(inputImg, '.'),'_',int2str(threshold),'_segmented.png');
imwrite(BWforeground,filepath);

% obtain the full color representation of the foreground objects
foregroundR=zeros(height, width);
foregroundG=zeros(height, width);
foregroundB=zeros(height, width);
for i=1:height
    for j=1:width
        if(BWforeground(i,j)==255)
            foregroundR(i,j)=r(i,j);
            foregroundG(i,j)=g(i,j);
            foregroundB(i,j)=b(i,j);
        end
```

```
    end
end
foregroundRGB=cat(3,uint8(foregroundR),uint8(foregroundG),uint8(foregroundB));

%
figure(7),imshow(foregroundRGB),title('Coloured Foreground');
filepath = append('figures/',extractBefore(inputImg, '.'),'_',int2str(threshold),'_segmented_colored.png');
imwrite(foregroundRGB,filepath);

close all;
```

The following is the code for *alternative_segment_color.m*:

```
function simple_segment_color (inputImg)

% Imports a coloured image with a blue background and presents that image on the
% screen;

img = imread(inputImg);
if size(img,3) ~= 3
    img=cat(3,img,img,img);
end

figure(1), imshow(img),title('Original')

% get image dimensions: an RGB image has three planes
[height, width, planes] = size(img);

% Separates each RGB component in a different matrix, visualises each one on the
% screen and writes to disk;

if(planes==3)
    r = img(:, :, 1);          % red channel
    g = img(:, :, 2);          % green channel
    b = img(:, :, 3);           % blue channel
end

figure(2), imshow(r),title('Red Channel');
filepath = append('figures/',extractBefore(inputImg, '.'),'_red_channel.png');
imwrite(r,filepath);

figure(3), imshow(g),title('Green Channel');
filepath = append('figures/',extractBefore(inputImg, '.'),'_green_channel.png');
imwrite(g,filepath);

figure(4), imshow(b),title('Blue Channel');
filepath = append('figures/',extractBefore(inputImg, '.'),'_blue_channel.png');
```

```
imwrite(b,filepath);

% Generate the histogram of the Blue Channel with the built-in function imhist

figure(5),imhist(b),title('Blue Channel Histogram');
filepath = append('figures/',extractBefore(inputImg, '.'),'_blue_channel_histogram.png');
saveas(gcf, filepath)

% Ask the user to input the threshold value

threshold = input('Input threshold:');

% Uses the matrix with the B component to identify the foreground objects (the jumping
%man or the Christmas bulbs or the bird). Using a threshold, copy from the B matrix to a new matrix only
%the pixel values that are below that threshold.
%When doing that you may put those pixels with the value 255 and all the others with value 0 (you will create a black and white
%picture).

BWforeground = zeros(height,width);
for i = 1:height
    for j = 1:width
        if(b(i,j)<threshold)
           BWforeground(i,j)=255;
        end
    end
end

% Shows the black and white segmented image on the screen and creates in the disk a new file with that image.

figure(6),imshow(BWforeground),title('B&W Segmented Image');
filepath = append('figures/',extractBefore(inputImg, '.'),'_',int2str(threshold),'_segmented.png');
imwrite(BWforeground,filepath);

% obtain the full color representation of the foreground objects
foregroundR=zeros(height, width);
foregroundG=zeros(height, width);
foregroundB=zeros(height, width);
for i=1:height
    for j=1:width
        if(BWforeground(i,j)==255)
            foregroundR(i,j)=r(i,j);
            foregroundG(i,j)=g(i,j);
            foregroundB(i,j)=b(i,j);
```

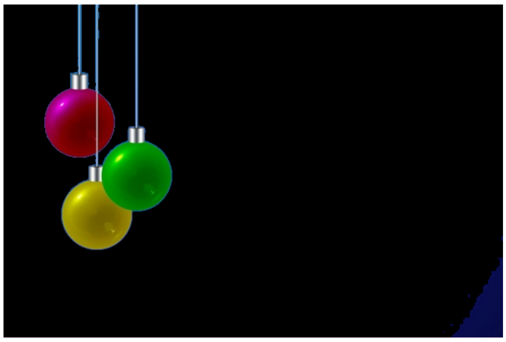
```
        end
    end
end
foregroundRGB=cat(3,uint8(foregroundR),uint8(foregroundG),uint8(foregroundB));

%
figure(7),imshow(foregroundRGB),title('Coloured Foreground');
filepath = append('figures/',extractBefore(inputImg, '.'),'_',int2str(threshold),'
    _segmented_colored.png');
imwrite(foregroundRGB,filepath);

close all;
```

The following are the resulting images:

(a) Colored Foreground of 'christmasBB.jpg'

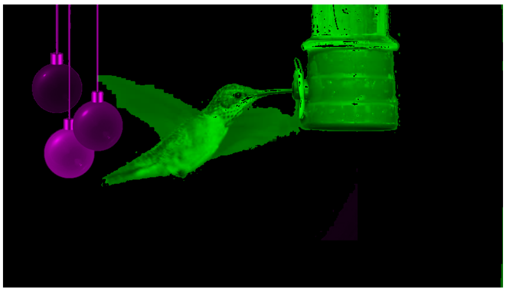


(b) Colored Foreground of 'christmasBB.jpg'

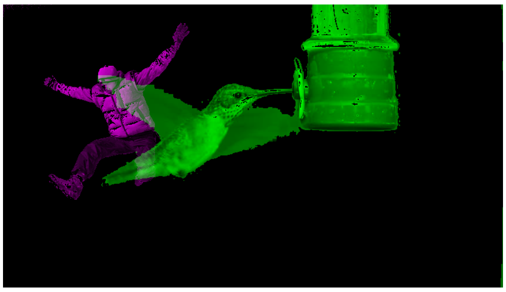
(c) Colored Foreground of 'jump.jpg'

Figure 2.2: Colored foreground images

Finally, we once more experimented using the built-in Matlab function *imfuse* using the generated colored foreground images, the following were our results:



(a) Superimposing colored foreground of 'birdBB.jpg' and 'christmasBB.jpg'



(b) Superimposing colored foreground of 'birdBB.jpg' and 'jump.jpg'

(c) Superimposing colored foreground of 'christmasBB.jpg' and 'jump.jpg'

Figure 2.3: Superimposed colored foreground images using *imfuse*

A lot more transparency can be observed in the resulting images, this is because unlike our previous experiments, the segmented images are not in full white, and thus have lower color intensities.