Image & Video Processing

Assignment 1

Albert Cerfeda

Contents

19 March 2023
Università della Svizzera italiana
Faculty of Informatics
Switzerland

1 Point Operations [20 points]

1.1 Tone mapping & Linearization [2 points]

The provided image `ferrari.jpg` has been gamma corrected in order to take into account the non-linear conversion from intensity value to the actual emitted light on the screen. In this exercise we perform various transformations on the provided image:

- Linearize the image back by inverting the gamma correction.
  We know that the image has been encoded with gamma $\gamma = \frac{1}{2.2}$, so the inverse operation is to raise the intensity values to the power of $\gamma = 2.2$. Results are shown in Figure 1b.
- Increase the brightness of the image by multiplying the intensity values by a constant factor.
  Let us increase the brightness by a factor of 2. Results are shown in Figure 1c.
- Enhance the contrast of the image by applying an exponential function to the intensity values. Results are shown in Figure 1d.

```
im_lin = im.^2.2; % 1.1 Linearize the image
im_lin = im.*2;   % 1.2 Increase the brightness
im_con = im.^0.7; % 1.3 Enhance the contrast
```

(a) Original `ferrari.jpg`. (b) `ferrari_lin.jpg`. Linearized contrast

(c) `ferrari_bri.jpg`. Scaled brightness (d) `ferrari_con.jpg`. Exponential contrast.

Figure 1: Tonemapping procedure comparison.

1.2 Color correction [4 points]

During image acquisition, the color of the light sources affect the color of the captured image. Our eyes, unlike cameras, are able to adapt to the color of the light source and thus we are able to see the correct colors. In order to represent the colors as they are seen by our eyes, we need to perform white balancing. We implement different white-balancing transformations:

- Pixel-based correction
  We choose a pixel in the image that appears gray, and we balance the color channels around the chosen pixel's color. Results are shown in Figure 3b.

- Gray-world assumption
  We assume that the average color of the image is gray so, instead of balancing the image colors around some pixel, we balance the channel mean itself. Results are shown in Figure 3c.

```
% ...
channelmean = mean(mean(im))
% 2.1 Pixel-based correction
figure(); imshow(im);
coords = int32(ginput(1));
px_color = im(coords(2), coords(1), 1:3);
gain = px_color./channelmean;
im_pxcorr = im.*gain;

% 2.2 Gray-world assumption
gain = channelmean/mean(channelmean);
im_gwa = im./gain;
```

Figure 2: Matlab code performing various intensity transformations on the image `ferrari.jpg`.

(a) Original image (b) Pixel-based correction (c) Gray-world assumption correction

Figure 3: Color correction procedures comparison.

1.3 Histograms [1 point]

Image histograms show us the distribution of the different pixel values for each color channel. Let us implement a function to compute RGB image histograms.

```
function [distribution] = compute_distribution(channel)
    distribution = cell2mat(arrayfun(@(x) sum(sum(channel==x)), 0:255,'UniformOutput',false));
end
```

In the implementation, every intensity value in the `0:255` range gets mapped to the number of occurrences of the value in the channel.

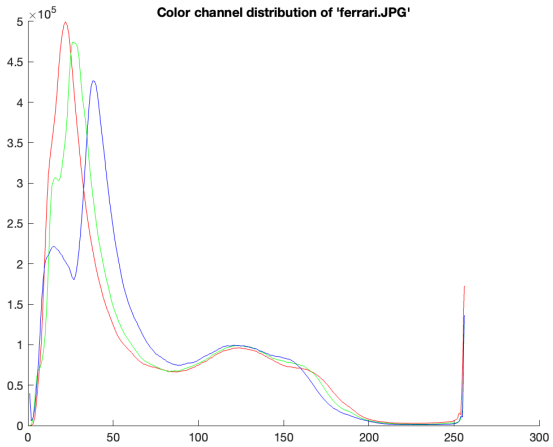


Figure 4: Color channel distribution of the image `ferrari.JPG`.

1.4 Histogram Equalization [5 points + 2 Bonus]

Histogram equalization is a tecnique used to redistribute pixel intensities in order to improve details in low-contrast regions. We compress the details of the intensity regions that do not contain much information, and we expand the details of the regions that contain more information.

- **Global equalization**
  We calculate the cumulative distribution function of the images' Value channel distribution and use it to map the intensities from the old image into the new, equalized one. Results are shown in Figure 5b.

```
function [equalized_channel, original_brightness_distribution] = ...
        global_histogram_equalization(imhsv)
    channel = imhsv(:,:,3);
    % Cumulative distribution function of the image histogram
    [original_brightness_distribution, ~ ] = imhist(channel);
    cdf = cumsum(original_brightness_distribution);
    cdf = cdf / cdf(end);
    % Apply the transformation
    equalized_channel = cdf(round(channel*255+1));
end
```

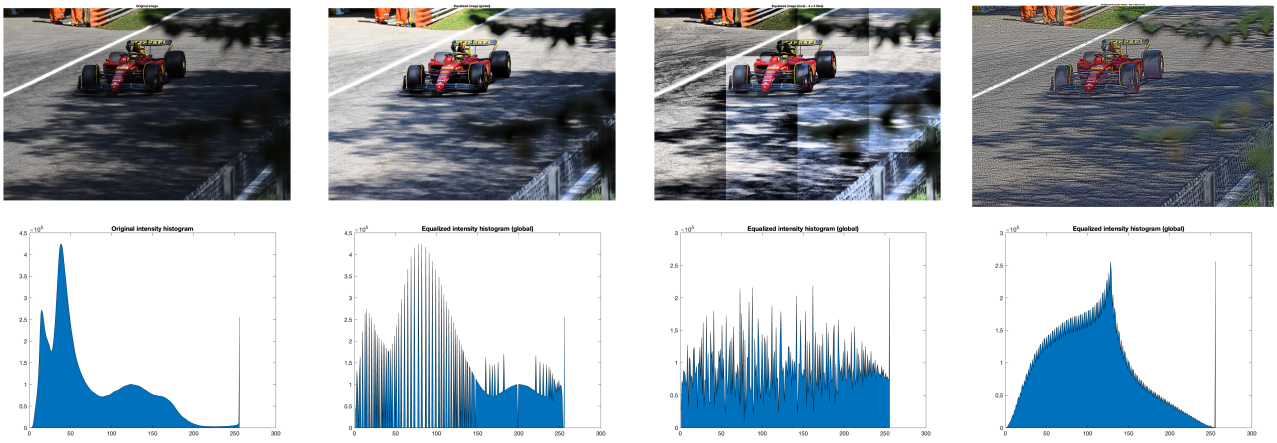
- **Local equalization**
  We subdivide the frame into tiles and we equalize them separately using global equalization. Results are shown in Figure 5c.

```
function [imhsv_equalized] = local_histogram_equalization(imhsv, TILENO)
    imhsv_equalized = imhsv;
    im_size = size(imhsv_equalized);
    tile_size = floor((im_size(1:2)/TILENO));
    for row = 0:TILENO-1
        for col = 0:TILENO-1
            imageUB = round(([row col].*tile_size)) + 1;
            imageLB = round(([row col].*tile_size) + tile_size);
            tile = imhsv_equalized(imageUB(1):imageLB(1),imageUB(2):imageLB(2),:);
            [equalized_tile, ~ ] = global_histogram_equalization(tile);
            imhsv_equalized(imageUB(1):imageLB(1),imageUB(2):imageLB(2),3) = equalized_tile;
        end
    end
end
```

- **Locally adaptive equalization**
  We equalize each pixel by using a moving window that has the pixel at its center. We run the standard global equalization on the window and we apply the transformation to the pixel. Results are shown in Figure 5d.

```
function [imhsv_equalized] = locally_adaptive_histogram_equalization(imhsv, TILESIZE)
    imhsv_equalized = imhsv;
    im_size = size(imhsv_equalized);
```

```
        for y = 1:im_size(1)
            for x = 1:im_size(2)
                imageUB = round(max([[y x] - (TILESIZE/2); 1 1]));
                imageLB = round(min([[y x] + (TILESIZE/2); im_size(1:2)]));
                tile = imhsv_equalized(imageUB(1):imageLB(1), imageUB(2):imageLB(2),:);
                [equalized_tile, ~ ] = global_histogram_equalization(tile);
                imhsv_equalized(y,x,3) = equalized_tile(y-imageUB(1)+1, x-imageUB(2)+1);
            end
        end
    end
```



(a) Original image. (b) Global eq. (c) Local eq. (4x4 tiles) (d) Locally adaptive eq.

Figure 5: Comparison of different histogram equalization techniques.

1.5 Manual histogram equalization [2 points]

1.6 Thresholding & matting [4 points]

The chroma key implementation is rather straightforward. We let the user choose the pixel color that will be used when deriving the mask to isolate the cat subject from the background. Since the green background is not homogeneous, we define a set of threshold values for each color channel in order to take into account the variability of the background. We then apply the mask to the original image and replace the pixels that are not masked with the background image. The results are shown in Figure 6d.

1.7 How many bits is enough [2 points]

2 Bonus: use your own pictures [3 points]

I took multiple pictures of myself in different poses in my hallway. I took a cleanplate picture of my hallway and used it to compute the masks for all the other pictures. Once I had all the masks, I composited every masked picture on top of the cleanplate image. The result is shown in Figure 8b. Please note that the pictures were taken in challenging lighting conditions, so the results are not perfect. Improving the masking would require more time and effort, but I think the results are still quite good.

(a) Original image.

(b) Chroma Key mask.

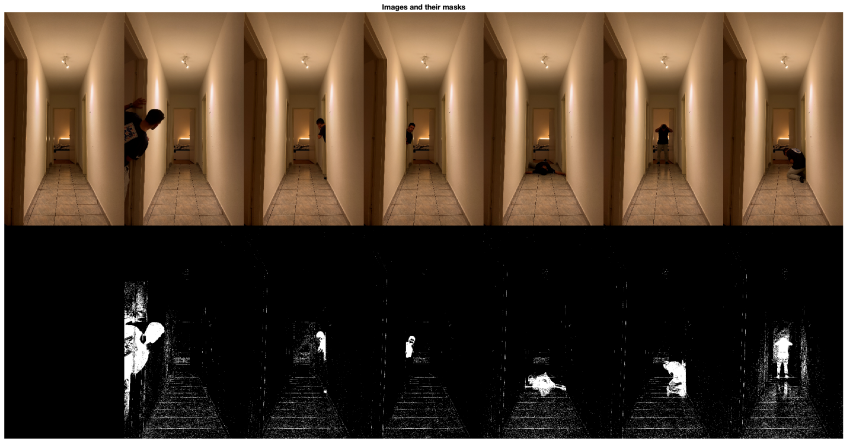
(c) Background image.

(d) Composited result image.

Figure 6: Chroma key compositing

```
im = imread("./media/cat.jpg");
imhsv = rgb2hsv(im);
im_spacebg = imread("./media/spacebg.jpg");

im = (double(im)./255);
im_spacebg = (double(im_spacebg)./255);

% Lets the user select a pixel in the image to key out.
figure()
imshow(im);
coords = int32(ginput(1));
px_hsv = imhsv(coords(2), coords(1), 1:3);

%  Each channel may deviate of these amounts from the selected pixel.
hue_deviation = 0.05;
saturation_deviation = 0.1; % += 10% saturation
brightness_deviation = 0.4; % += 40% darker/lighter

% Creates the mask for each pixel whose channels are within the deviations
mask = ~((imhsv(:,:,1) > px_hsv(1) - hue_deviation) & (imhsv(:,:,1) < px_hsv(1) + hue_deviation) & ...
        (imhsv(:,:,2) > px_hsv(2) - saturation_deviation) & (imhsv(:,:,2) < px_hsv(2) + saturation_deviation) & ...
        (imhsv(:,:,3) > px_hsv(3) - brightness_deviation) & (imhsv(:,:,3) < px_hsv(3) + brightness_deviation));

im_mask = comp(ones(size(im)),mask,zeros(size(im)));
im_comp = comp(im,mask,im_spacebg);

function res = comp(a,mask,b)
    % Masks image 'a' on top of image 'b'
    res = a.*mask + b.*(~mask);
end
```

Figure 7: Matlab code performing background replacement on image `cat.jpg`.



(a) Individual pictures and relative masks.

(b) Final composited image

Figure 8: Mourning the death of Albert