Energy Management for IoT - Report Lab 02

Flavia Caforio (s257750) - Samuele Yves Cerini (s256813)

February 27, 2020

Contents

1 Introduction

The goal of this second laboratory is to demonstrate how image manipulation can be used to tradeoff image quality to reduce the energy power consumption in emissive displays, like OLED ones. We will implement and test different techniques to modify a set of demonstrative images: we will evaluate the effect of the modifications both qualitatively and quantitatively. We will also evaluate the gains (if any) in terms of power consumption, trying to define a trade-off between the image quality and the gains obtained. Finally, as an additional requirement, we will visualize these images using a proper OLED (thus, emissive) display. The overall laboratory, and hence, the report, is divided into 2 main parts:

- Image manipulation, Distortion estimation and Power consumption
- Interfacing with the external OLED display and image manipulation on-the-edge

2 Part 1: Image Manipulation | Distortion Estimation | Power Consumption

In the first part we will talk about how the changes made to the images have been implemented.

2.1 The MATLAB Script

In the first part of the `Matlab` script `lab2.m` the images are collected in `original_vector_images` and also the lab color space version in `original_vector_images_lab` useful for computing the Euclidean distance that we'll talk about later. In addition, the power cost is calculated using the `image_power.m` function and collected in `original_vector_power`. There are 20 images to process, 5 of which are screenshots. The `Matlab` script is then subdivided according to the type of manipulation that has been made to the original images with the related calculations to calculate the distortion and power difference between the original and the modified image.

2.2 Power Consumption Estimation

In order to calculate the power of the image it is necessary to compute and sum the power of the single pixel plus the baseline power. The power of the single pixel is $P_{pixel}(R, G, B) = w_r * R^\gamma + w_g * G^\gamma + w_b * B^\gamma$ where RGB are the values of the single color channel in the pixel which emphasizes the fact that energy is color-dependent. The following coefficients are used that refer to a specific OLED display.

```
% Power constants
wr = 2.13636845e-7;
w0 = 1.48169521e-6;
wg = 1.77746705e-7;
wb = 2.14348309e-7;
gamma = 0.7755;
```

So at the end we can compute the total power using the following formula $P_{image} = w_o + \sum_{i=1}^{N} P_{pixel}(i)$

2.3 Distortion Estimation

There are two ways to estimate image distortion. The distortion computation is based on numerical values but the distortion perceptible to the human eye is very different and difficult to evaluate. For example images can tolerate some approximations which are hardly visible, but but can improve the power saving.

2.3.1 Euclidean Distance

The Euclidean distance is computed for each pixel in the `LAB` space where L is the luminance and a and b are two color channels, and corresponds to the difference of these components between the two images. In the `Matlab` function `eucl_distance.m` receives the two images in the lab space and computes the Euclidean distance. This metric is based on luminance but also on colors.

```
 % Evaluating the distortion by computing the euclidean distance between pixels

eps = 0;
for i = 1:n(1)
    for j = 1:n(2)
        eps = eps + sqrt( (L(i,j)-L2(i,j))^2 + (a(i,j)-a2(i,j))^2 + (b(i,j)-b2(i,j))^2 );
    end
end
```

2.3.2 SSIM

A very valid alterative is the computation of the Mean Structural Similarity Index (MSSIM). The calculation is already implemented in the `Matlab` `ssim()` function. As we will see later, it is the most faithful metric from the point of view of distortion perception.

2.4 Image Manipulations

Here we will talk about the changes that have been made to the images and their results and considerations. The result paragraph of each modification reports the results based on the image #15 in figure 1 and a screenshot (image #17) in figure 2 to get an idea of the findings. The data collected in the tables of results are the power saving value and, as parameter to evaluate the visual distortion of the image, we have chosen to show the MSSIM values.

We have to keep in mind that each image has its own characteristics and that the results can also be very different.

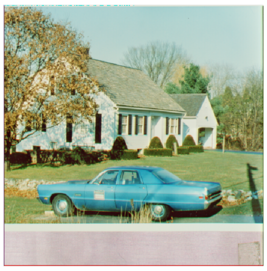

Figure 1: Demo image #15

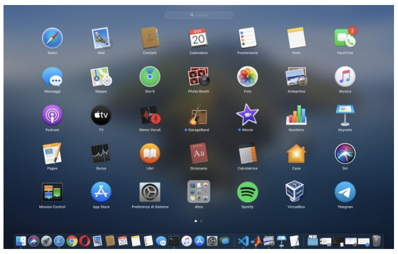

Figure 2: Screenshot image #17

2.4.1 Colors Manipulation

The first manipulation that has been done is specifically on colors. Manipulating colors is the same as backlight dimming in transmissive displays, so in OLED emissive devices we try to reduce display consumption by using the dependency of energy on color values. The reduction was made in such a way that there was a gradual distortion of the color. This basically means reducing the color value of each pixel by an increasing percentage. It was implemented by iterating 10 to 100 percent color reduction on all three RGB values of each pixel. For each image and for each percentage the Euclidean distance, MSSIM and energy saving are estimated and collected. In the function `color_reduction.m` we call the function `single_color_reduction.m` for each color. Inside this last function the reduction to be applied is proportional to the maximum value of the color and applied to the channel as shown in the following code.

```
% reduction => 255 : 100 = x : perc_reduction
reduction = (perc_reduction * 255) / 100;
for i = 1:n(1)
    for j = 1:n(2)
        I_d(i, j, channel) = I(i, j, channel) - reduction;
    end
end
```

2.4.1.1 Results

Color reduction is a pixel-wise transformation strategy that has a great impact on energy. As we can see from the data collected already with a 20% of color reduction has a good margin of power saving for most of the images. The results are as we expected, i.e. a reduction in colour has certainly had an impact on energy savings at the expense of the similarity between the original and the modified image, which is darker and obviously different as the savings increase.

Color reduction percentage	im#15 power saving %	im#15 ssim %	im#17 power saving %	im#17 ssim %
10	13.0	96.4	24.5	88.6
20	27.5	86.6	49.8	56.7
30	40.9	73.1	68.8	32.3
40	54.5	55.6	81.6	17.7
50	67.0	37.8	89.6	8.7
60	79.3	21.2	94.4	4.0
70	89.7	9.0	96.4	2.1
80	97.7	1.2	98.0	0.8
90	99.9	0.0	99.2	0.2
100	99.9	0.0	99.9	0.0

2.4.2 Histogram Equalization

Histogram equalization is not pixel-wise transformation but rather about brightness/contrast modifications (hence, related to the overall image). For this reason it is necessary to switch to HSV space: here we modify only the third element (called *Value*), that is associated to the brightness of the image, implemented as in the following code.

```
HSV_img = rgb2hsv(original_vector_images{i});
temp = HSV_img(:, :, 3);
HSV_img_histeq = histeq(temp);
HSV_mod = HSV_img;
HSV_mod(:, :, 3) = HSV_img_histeq;
```

It is very important to remember that after histogram equalization the returned HSV values are, even after completing the reverse transformation into the RGB space, between 0 and 1. Hence, for the purpose of power saving calculations and distortion evaluation parameters it is necessary to transform this values of type `double` into `integer` and this is possible through the function Matlab `im2uint8()`.

2.4.2.1 Results

The images resulting from histogram equalization are for the most part convenient from the point of view of power saving with very high similarity. Images for which the energy saving is negative, on the other hand, mainly concern images already dark at the beginning where the contrast has been increased and for this reason they are brighter and therefore consume more. The images #15 and #17 chosen as a sample are a perfect example because they behave in a diametrically opposed way. These considerations can be referred also to the figure 7.

Hist equalization	Power saving %	MSSIM %
Image 15	22.4	68.78
Image 17	-11.0	85.41

2.4.3 Luminance Reduction

As an optional transformation we have chosen to use a luminance transformation in the HSV space via the function we have implemented in `luminance_reduction.m`. The function is so implemented:

```
reduction = alpha / 100;
HSV_img_reduced = HSV_img;
HSV_img_reduced(:, :, 3) = HSV_img(:, :, 3) - reduction;
```

Where alpha is iterated from 10 to 100 percentage for all images as we did for color reduction. Alpha is then divided by 100 as the values in the HSV space range from 0 to 1. Therefore also here we have to go back to integer values after the transformation and return in the RGB space. Luminance reduction is an approach that we wanted to analyze because we can experience it every day with our mobile phones when we lower or increase the brightness of the phone display. Therefore, it is interesting to see the related effects on energy saving.

2.4.3.1 Results

Working only on luminance is not very efficient in terms of image distortion, as it becomes very dark even at low percentages, very similar to the results of color reduction approach. However, the best compromise we can see is that with a 10% reduction we can have positive power saving percentages and an acceptable similarity.

Luminance reduction percentage	im#15 power saving %	im#15 ssim %	im#17 power saving %	im#17 ssim %
10	11.5	96.9	19.4	90.5
20	23.7	87.3	41.2	59.9
30	35.8	73.6	61.4	33.8
40	48.4	55.8	74.8	19.2
50	61.2	37.2	84.5	9.9
60	74.2	20.4	92.8	4.7
70	86.2	8.7	95.2	2.5
80	95.9	1.7	97.3	1.0
90	99.9	0.0	99.0	0.2
100	99.9	0.0	99.9	0.0

2.5 Plot Creation

In order to quantitatively compare the results obtained with the different manipulations, considering all the images in our possession, we decided to automatically build some comparisons plots that could clearly help us defining some tradeoffs.

2.5.1 Color Manipulations Plots

For this very first image manipulation technique, we decided to summarize all the modifications made into a single plot, comparing the percentage of the color reduction applied against the power consumption gains obtained with the modified image. We reported the plot in figure (3). As an example, it is evident how the first two images present bigger gains with respect to all the other images: this is due to their original intrinsic darkness. The second image, which is subject to a 20% color reduction, is reported in figure (5). In a similar way, we can appreciate the lower power consumption gains on image #9, which is, on the other hand, very bright in its original form. Of course, by increasing the color reduction percentage all the images become more and more similar in terms of power consumption but at the same time the quality of the image becomes way too unacceptable.

Starting from image #16 (i.e. the subset of images representing all the screenshots taken from our PC) we can notice big improvements on the power consumption domain, especially for image #17 (the MacOS application drawer) and image #18 (the dark-themed text editor). Especially for this last image, it is clear how a dark background can positively impact the power consumption of emissive displays.

For this same purpose, we decided to save a comparison image for each degree of color reduction percentage in order to qualitatively assess the quality of the final image. In figure (4) we can see what a color reduction of the 20% looks like on an already dark image.

Finally, to evaluate the energy consumption gains alongside a quantitative evaluation of the image distortion, we plotted, for each image in the image set, a graph that shows the trends of both the power consumption and the euclidean (or SSIM) metric. As we can see from figure (6), each x value corresponds to a different color reduction percentage, on the left y axis we reported the energy savings obtained and on the right y axis we reported the Euclidean Distance (or SSIM) values.

From this figure we can easily understand that the SSIM technique (used to evaluate the image distortion) is the one that better reflects the real image quality as seen by the human eye. As an example (refer also to figure (5)): notice that at 20% of color reduction the SSIM index indicates that the similarity of the modified image against the original one is slightly below the 22%. By qualitatively comparing the two images we can appreciate in fact the over-distorted condition of the modified image. On the other hand, the Euclidean Distance, for the same color reduction percentage, tops at a 3.45% distance from the original image: this obviously does not reflect the real final distortion perceived on the modified image.

2.5.2 Histogram Equalization Plots

Similar plots have been made also for the Histogram Equalization image manipulations. For each image of the set, we extracted a figure containing the original image and its related color histogram, compared with the modified image (that has been histogram equalized) and its related color histogram. Finally, we made a plot

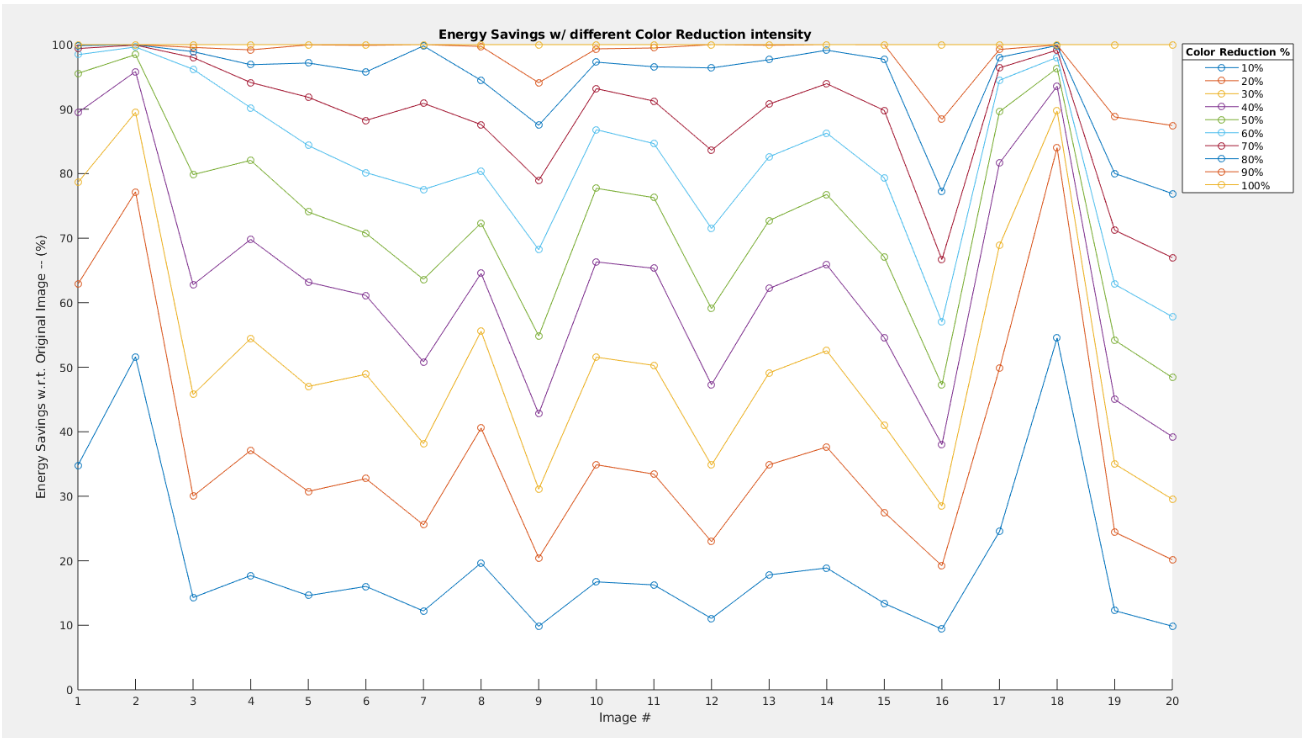

Figure 3: The plot represents, for each of the 15 images (on the x axis) and for each color reduction percentage applied to it (the different curves), the overall power consumption gains obtained.

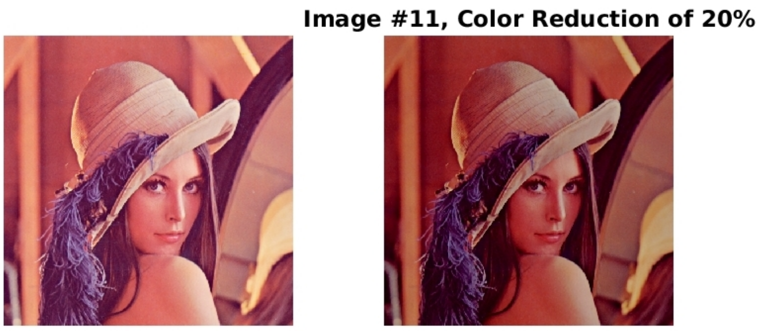

Figure 4: On the left the original image, on the right the same image with all colors reduced by the 20%. The image is still recognizable and the overall quality is still acceptable. The modified image presents a power consumption gain equal to the 32%.

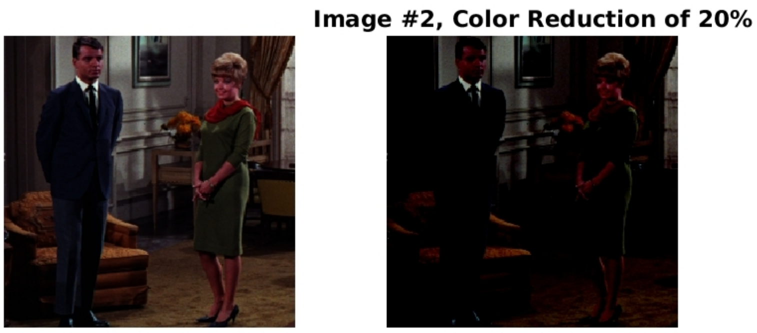

Figure 5: On the left the original image, on the right the same image with all colors reduced by the 20%. The image is too distorted and the quality loss is unacceptable. This is due to the intrinsic darkness of the original image.

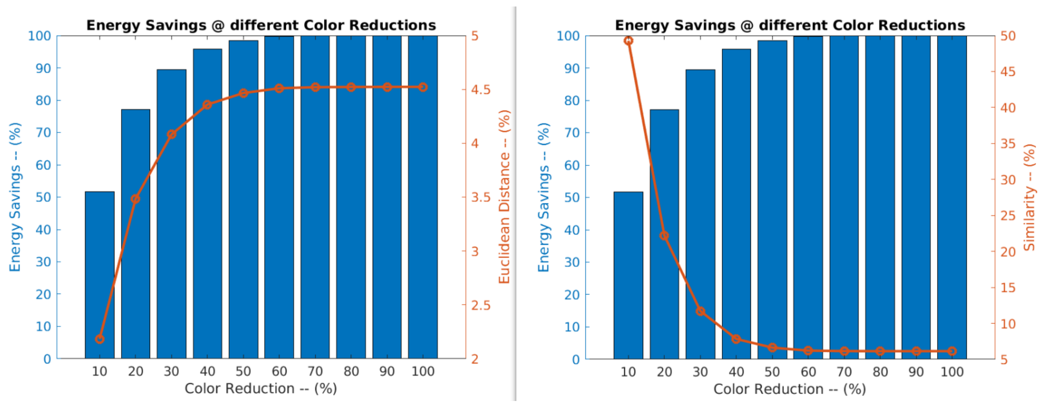

Figure 6: Two graphs regarding image #2, representing the tradeoff between the power consumption gains obtained with 10 different color reduction profiles and the two techniques used to assert the image distortion (on the left the Euclidean Distance, on the right the SSIM similarity index).

that compares, for each image, the power consumption gains obtained, the Euclidean Distance and the SSIM parameter.

This summarized plot can be observed in figure (7). From the left graph (the one representing the power consumption) it is possible to see that some images, for example image #1, #2 and the screenshots #17 and #18 suffer from an increase of the power consumption up to 130%. At the same time, all the other images present a power consumption reduction which is less evident with respect to the Color Reduction technique. However, the similarity of most of the images is between the 40% and the 60%, with some images peaking even above the 80%. Once again, the Euclidean Distance, especially considering the previous results obtained when evaluating the Color Reduction, cannot be considered as a good metric to evaluate the image distortion as perceived by the human eye.

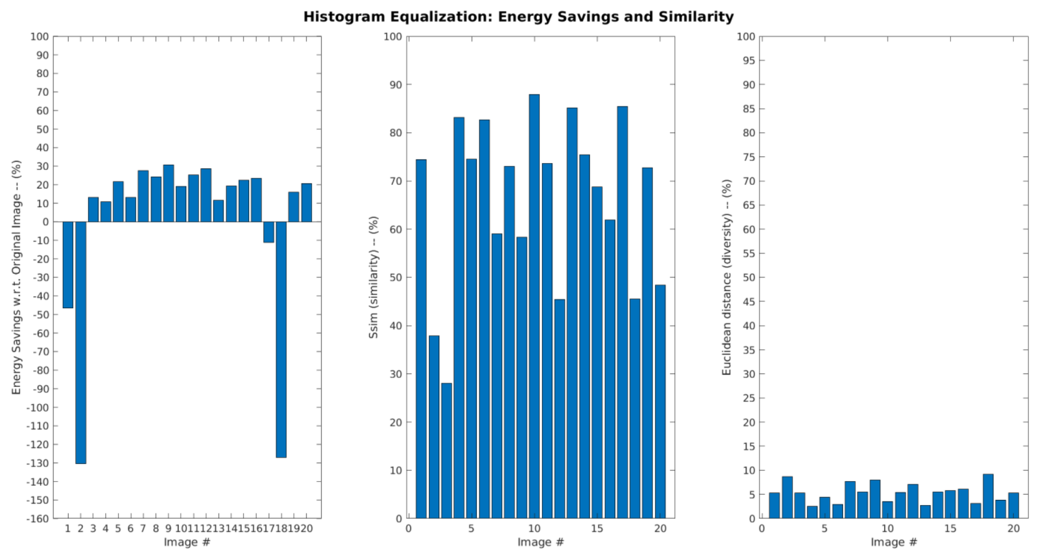

Figure 7: Summarized plot representing, for each image of the set, the power consumption, the Euclidean Distance and the SSIM parameter obtained after the Histogram Equalization.

From now on, we will discuss some of the most interesting results obtained by considering a single image at a time. In figure (8) we can, for example, observe the Histogram Equalization applied to the second image of the entire set: this image demonstrates that the equalization process can in fact increase the power consumption of an image. In this case, we can see that the application of the equalizer "stretches" horizontally the entire histogram: in the original images most of the colors were concentrated on the left half of the histogram, proving the overall darkness of the original image. The equalization "expanded" the concentration of the colors also to the right part of the histogram, occupying also the more bright zones of the histogram (the ones near the value 255).

An example of a good histogram equalization can be seen in figure (9). Here, the original image (image #13) presents an histogram which is already "stretched" enough to cover the entire x axis uniformly. The application of the equalization hence stretches even more the histogram, especially for the red channel. As a result, the image

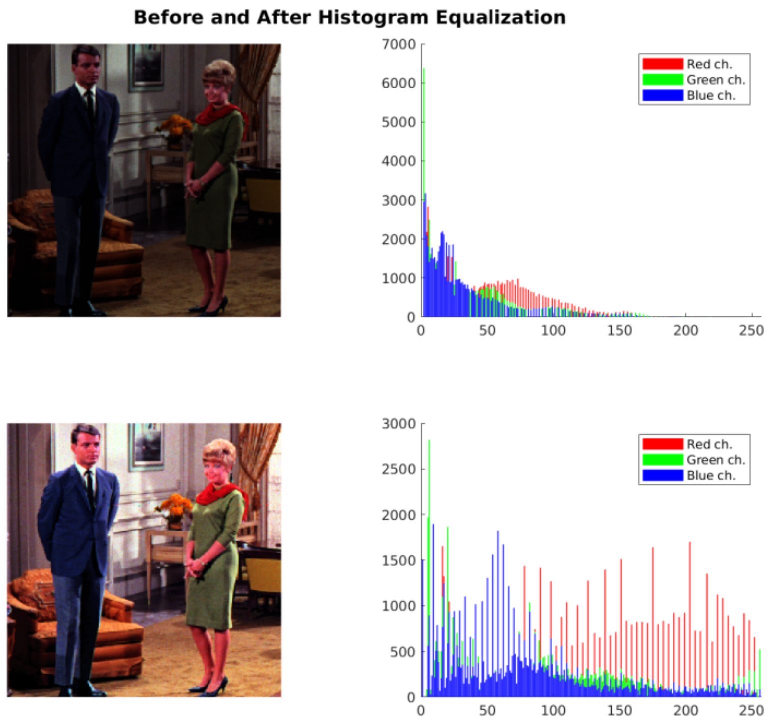

Figure 8: An example of Histogram Equalization that actually increases the power consumption of an image: the color histogram is stretched up to the most bright zones (as a rule of thumb, the more the brightness of the image, the more its power consumption). This image, once equalized, results in a power consumption increased by 130%.

appears really similar to the original one, however with darker shadows and an overall better exposure. If we now see the power consumption in the summarizing plot, we can see that image #13 has a power consumption which is reduced of the 30%, which is a very good result considering the overall quality of the final image. Such result suggests that an Histogram Equalization transformation is best suited to images that are overall "equilibrate" in terms of color presence in their related histogram.

2.5.3 Luminance Reduction Plots

In this last section we finally analyze the results obtained when reducing the luminance of an image. The results are overall similar to what we have obtained with the color reduction (the transformations are in fact similar).

In figure (10) we report a very similar plot (although different in the values represented) to what we already illustrated for the color reduction process. In this case, the luminance reduction appears to be "less aggressive" in terms of gains of power consumption. For example, for image #1, we were able to obtain with a 10% of color reduction, a gain in power consumption of the 52%. On the other hand, with a 10% of luminance reduction, we obtained a gain in power consumption equal to the 44%. Similar reductions are observable also with other images. However, it should be noted that the both plots (3 and 10) are really similar in terms of curve trends.

In the following figure (11) we compared the results obtained with the same image (image #11) for both color reduction and luminance reduction. From this comparison we can see the differences, small but still significant, between the two techniques. From this comparison the luminance reduction seems to deliver a better image, with colors not too distorted, especially if we compare it against the strong red deviation of the skin color that we find in the color reduced image.

An interesting behavior has been obtained with the screenshots taken from our PC (image #18), as we can see from figure (12) and figure (13). From the first picture we can appreciate that a small color reduction does not affect too much the overall look of the dark-theme applied to the application, confirming once again that dark-themes are in fact beneficial when applied to emissive displays. Finally, from the second picture we can also appreciate the fact that, once the 20% of reduction threshold is overcome, the trend of the power consumption faces more and more resistivity, suggesting that further luminance reductions will probably impact too much the image quality for very limited gains obtained on the power consumption domain.

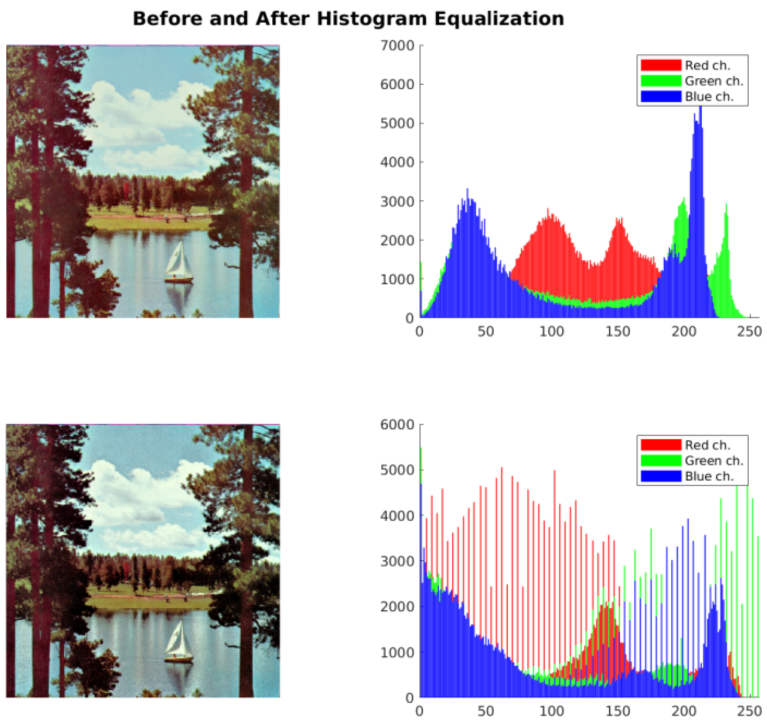

Figure 9: An example of good Histogram Equalization. The image appear nearly the same as the original, but with a power consumption that is reduced by the 30%.

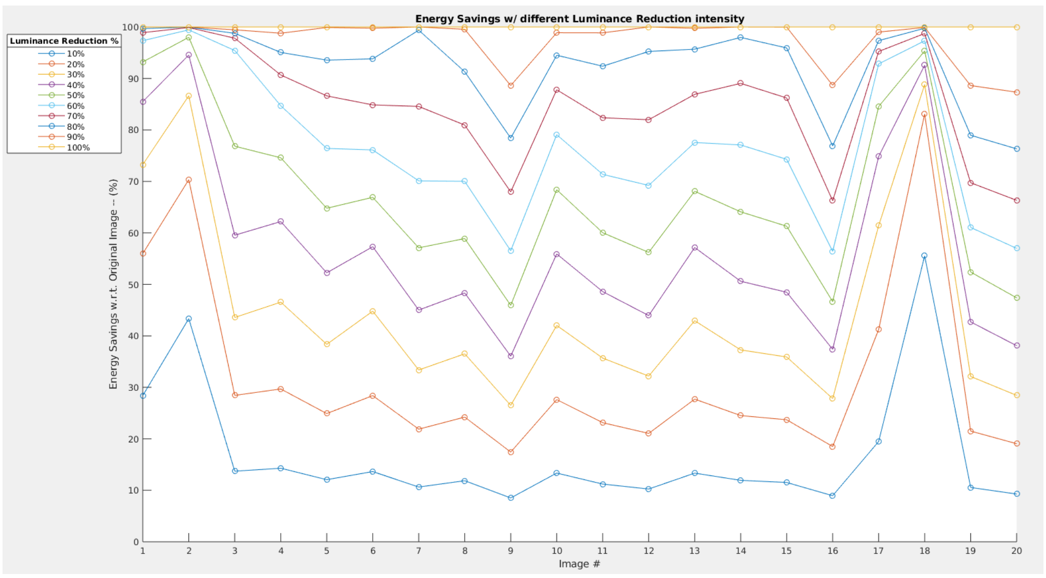

Figure 10: The plot represents, for each of the 15 images (on the x axis) and for each luminance reduction percentage applied to it (the different curves), the overall power consumption gains obtained.

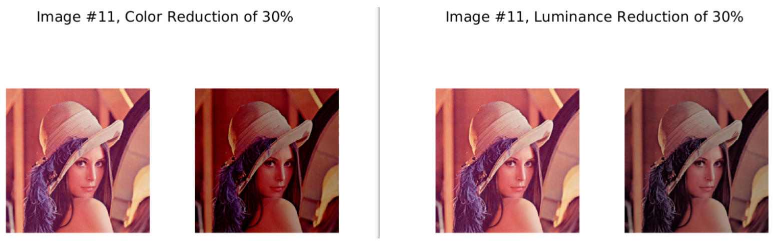
Image #11, Color Reduction of 30%
Image #11, Luminance Reduction of 30%

Figure 11: Comparison between the Color Reduction and the Luminance Reduction techniques. In this case, the Luminance Reduction delivers colors that are more faithful to the original image.

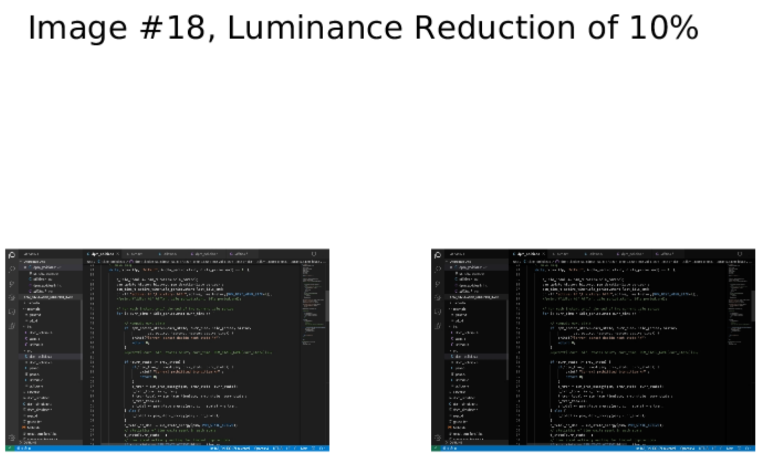

Figure 12: A screenshot of an application supporting a dark theme (image #18).

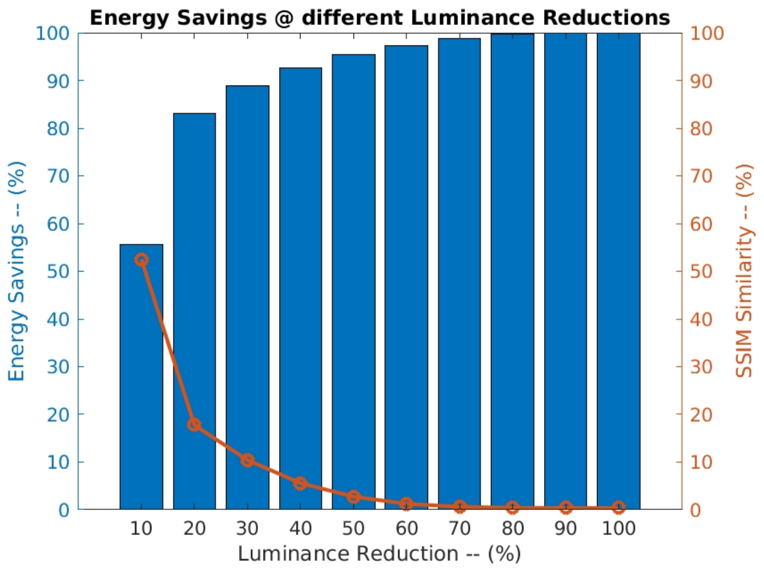

Figure 13: The overall power consumption behavior of the same screenshot of figure (12). We can see that a luminance reduction of the 10% is sufficient to obtain a power consumption which is reduced up to the 55%.

2.5.4 Conclusions

All the results obtained allow us to define some basic approaches to image manipulation. For example, we can infer that the color reduction technique cannot always be applied to all images: some images (like the one depicted in figure (5)) suffer from distortion even if affected by small manipulations.

In general, all of the three techniques we tested cannot be applied by default on all images without exploring the tradeoffs and evaluating carefully the results obtained image-per-image. Some of them provide in fact very good results on some images while very poor on other ones. Hence, it is not possible to define a "golden" manipulation technique able to return good results on the power consumption domain and poor image distortion for all the images.

If we consider the Color Reduction technique, a good policy will evaluate carefully the original image power consumption (tightly related to the overall darkness of an image), defining a threshold that, if surpassed, will resort to the application of the technique. Hence, images like image #1 and #2 will be kept in their original form, meanwhile images like image #9 will be color reduced.

A similar approach can be taken also considering the histogram equalization: from our experiments it is clear that static images like the screenshots obtained from our PC will suffer from a heavily degraded image quality if the equalization is applied. Histogram Equalization is better suited for photos and videos, where the overall image does not have an intrinsic "flat" or "unbalanced" histogram. Hence, we can apply some sort of filtering of the input image to avoid to apply this transformations to images we know in advance will be too distorted or will lead to an increased power consumption.

Finally, the luminance reduction technique demonstrated to be very effective if applied on our screenshots, where the majority of the image area is predominantly light or dark. Once again, we can think of a good automatic policy applying an histogram equalization on multimedia applications (like video-streaming ones) and luminance reduction on static applications (like text editors, spreadsheets and textual webpages).

3 Part 2: Interfacing w/ an external OLED display | Image manipulation on-the-edge

For this second part of the laboratory we were required to use some additional equipment to load and test our set of images on a real emissive display. We were provided, in fact, with an *Arduino UNO* board and an OLED Display by *Newhaven Display* connected to it. The goal is to asses the real impact on visual quality for the modifications made in the first part of the laboratory. Finally, this process allowed us also to learn how to interface our `MATLAB` environment with an 8-bit microcontroller, especially from the point of view of the serial interconnection protocol. In the following sections we will discuss both sides of the project: the `MATLAB` related one and the *Arduino* related one. As a final task, we will discuss the peculiarities observed when displaying the images on the OLED display and our attempt to manipulate the images by directly leverage the microcontroller logic.

3.1 The `MATLAB` Script

In the following sections we will discuss the `MATLAB` script (`BoardUpload.m`) used to send the images to the *Arduino* microcontroller.

3.1.1 Image Manipulation Prior Transfer

Before sending the images directly to the *Arduino* board, some modifications to each image have to be made considering the characteristics of the provided OLED display. This display has in fact a resolution of 128x128 pixels and its controller handles the data of each pixel not in a "classical" `RGB` schema, but on an "inverted" `BGR` schema. Finally, each pixel, which on classical displays is encoded into the 3 `RGB` 8-bit channels, is here encoded into 6-bit channels.

Our script starts by loading into memory the image that has to be sent to the microcontroller. The image is then resized to match the 128x128 resolution of the OLED display. After that, each of the 3 available channels is then transformed from an 8-bit notation into a 6-bit notation (to do that, a simple shift right operation by two positions has to be performed).

Now the image (if no additional transformation has to be done on the PC side) is ready to be sent to the microcontroller. In the following section we explain how this procedure is handled from the `MATLAB` perspective.

3.1.2 Serial Communication

The *Arduino UNO* board communicates using the same USB cable it uses as a power supply: the communication is serial and the `UART` protocol has to be used. `MATLAB` already provides the `serial()` method to correctly configure and interface our PC with additional Serial hardware like the one we have. To create a serial connection it is firstly mandatory to acquire the serial port *Arduino* is connected to: in our case, using a *Windows* PC, the serial port was called `COM3`.

While defining the serial connection using the `serial()` method we need to also define the classical configuration parameters a `UART` connection requires like the *BaudRate*, the presence of the parity bit (none, odd, even, mark, space), the number of data bits (5, 6, 7 or 8-bit), the byte order (big/little endian) etc... In our case, the default configuration provided by the method was sufficient: the only modification to be done was the one related to the BaudRate, configured at 115200 `BAUD`.

After that, it is only matter to open the connection using the `fopen()` method: if the serial port is not busy or occupied by another process, the connection will allow us to send the bytes representing our image to the *Arduino* board. Just after opening the connection we invoke the function `send_image(file_pointer, img_R, img_G, img_B)` that we defined in order to automatically handle all the byte-transferring process. When invoking it, we complete the last manipulation to the image needed to interface properly with the display controller: we swap the `RGB` channels into the new `BGR` notation. This function is defined in the `send_image.m` script: by leveraging the `fwrite()` function we send, pixel after pixel, the 3 bytes corresponding to the 3 channels that compose each pixel (i.e. at each iteration only one pixel is sent, hence requiring 128x128 iterations to send a complete image).

3.2 The Arduino Program

As a second task necessary to complete the image transfer it is required to implement the code that allows the *Arduino* board to correctly handle the reception of the image from the serial connection initialized by the `MATLAB` script. The provided code contains a set of high-level library functions that allow to complete some basic operations and to correctly interface the board with the display controller.

The *Arduino* sketch starts by the definition of some pre-processor commands that allow us to compile (or not) some sections of the code (we will discuss them later). Also, some variables are defined, like the `rcv_data[8]` array of `char`s that is used to load into memory the 8 different Bytes received from the serial connection. The first operation completed after the reset of the board is the setup of the interconnection to the OLED display and the setup of the display controller. Once these operations completed, we can set the starting position (i.e. the display cursor) from which the images will be shown on the physical display.

Finally, after this setup section, the code loops indefinitely until a serial connection is started by the `MATLAB` script (that will trigger the `serialEvent()` interrupt routine).

3.2.1 Image Reception

Now, assuming a byte has been sent by the `MATLAB` script through the serial connection, the microcontroller will start to execute the `serialEvent()` routine. Before explaining what the code implemented will do next it is necessary to explain how an *Arduino* board "reads" the data sent through the serial port. In fact, each character sent through the serial connection is interpreted like an `UTF-8` 8-bit character. If we now recall that the `MATLAB` script sends 8 characters at a time (corresponding to a single Byte of a single channel) we can understand that each one of these characters will be interpreted by *Arduino* like and 8-bit character itself. Hence, if we send the `"0100|1001"` Byte using `MATLAB`, *Arduino* will interpret this Byte as an array of 8 different Bytes, where each element of the array represents the ASCII code either of the character '`0`' or of the character '`1`'.

Hence, our *Arduino* code will read the entire sequence of 8 characters sent over the serial connection and place it into the `rcv_data[8]` array of `char`s. That done, we start a loop of 8 iterations in order to "translate" each ASCII code received into the actual bit value and push this value into a single `char` variable (called `data`) containing the reconstructed 8-bit sequence.

Once the sequence reconstructed, it is sent to the display controller using the `OLED_Data_128128RGB(data)` function. By doing that, we have transferred the first Byte, corresponding to one of the three Bytes needed to complete a single pixel.

In the following figure (14) we can see the same image which is color reduced by `MATLAB` on the left side of the screen and in its original form on the right side of the screen.

3.2.2 Image Manipulation

Since the Bytes sent to *Arduino* on the serial port cannot be stored in RAM because of its limited capacity, it is necessary to modify the single Bytes *on-the-fly* just after reception and before sending them again to the display controller.

To do that we added a piece of code, protected by the pre-processor directive `IMG_EDIT` in order to mask or unmask the code prior compilation. We decided to opt for a color manipulation technique, similar to the first manipulation technique we mentioned for the first part of the report. Also in this case, we defined a constant called `COLOR_REDUCTION_PERC` representing the percentage of reduction we want to apply to each channel of the image.

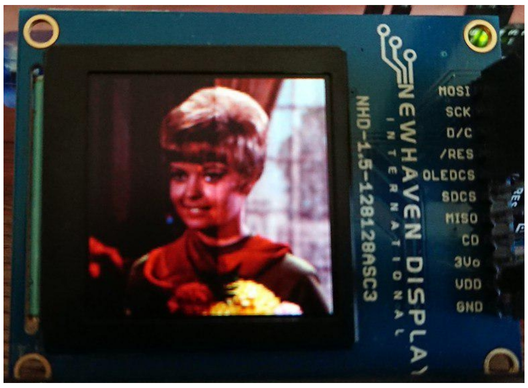

Figure 14: The OLED display representing the same image color reduced by `MATLAB` on the left side of the screen and in its original form on the right side of the screen. The image was loaded in around 20-30 seconds.

Since the color reduction must be applied *on-the-fly*, the computation is done in the `serialEvent()` routine: here the reduction value is subtracted to the original value and the new color reduced `data` variable is sent to the display controller. In the following picture (15) we reported a photo of the OLED display showing a comparison between our image, which is color reduced by `MATLAB` (on the right side of the screen) and the same image but color reduced *on-the-fly* by *Arduino* (on the left side of the screen). From this picture it is possible to notice the clear distortion caused by the color reduction applied at run-time to the original image, making such technique not feasible for everyday operations.

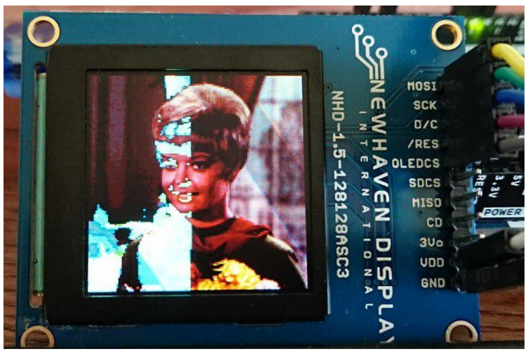

Figure 15: The OLED display representing the same image with the two different color reductions applied: on the left the color reduction (10%) done by *Arduino*, on the right the color reduction (10%) done by MATLAB.