Energy Management for IoT - Report Lab 02

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

February 20, 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

2.1 The MATLAB Script

2.2 Power Consumption Estimation

2.3 Distortion Estimation

2.3.1 Euclidean Distance

2.3.2 SSIM

2.4 Image Manipulations

2.4.1 Colors Manipulation

2.4.1.1 Results

2.4.2 Histogram Equalization

2.4.2.1 Results

2.4.3 Luminance Reduction

2.4.3.1 Results

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 to the power consumption gains obtained with the modified image. We can observe this plot in figure (1). From this plot 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 (3). 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.

At the same time, we decided to save a comparison image for each degree of color reduction percentage. In the following figure (2) we can see what a color reduction of the 20% looks like.

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 (4), 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 an example (refer also to figure (3)): 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 final quality of 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 that compares, for each image, the power consumption gains obtained, the Euclidean Distance and the SSIM parameter.

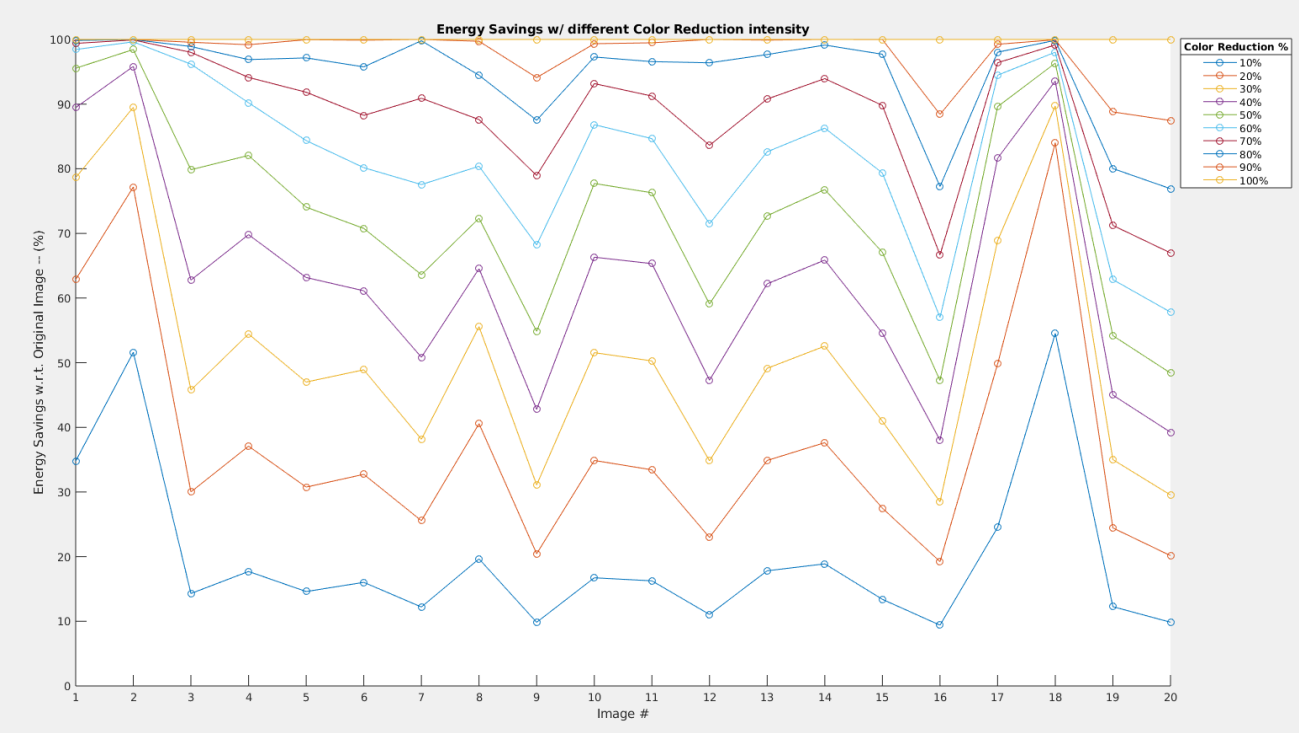

Figure 1: The plot represents, for each of the 15 images (on the x axis) and for each color reduction percentage applied to it (the different curves), the overall power consumption gains obtained.

Figure 2: 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 3: 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.

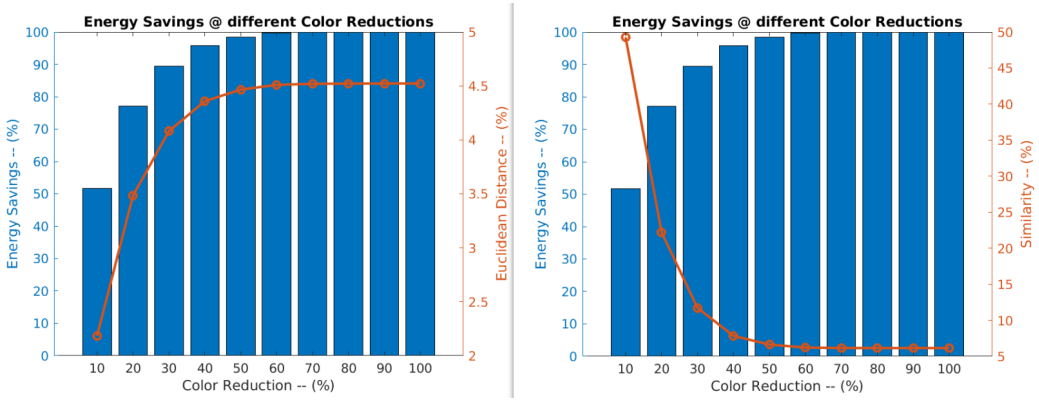

Figure 4: Two graphs regarding image #2, representing the tradeoff between the power consumption gains obtained with 10 different color reduction profiles and the two techniques used to assert the image distortion (on the left the Euclidean Distance, on the right the SSIM similarity index).

This summarized plot can be observed in figure (**??**). From the left graph, the one representing the power consumption, it 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 90%. Once again, the Euclidean Distance, especially given the previous results when evaluating the Color Reduction, cannot be considered as a good metric to evaluate the image distortion as perceived by the human eye.

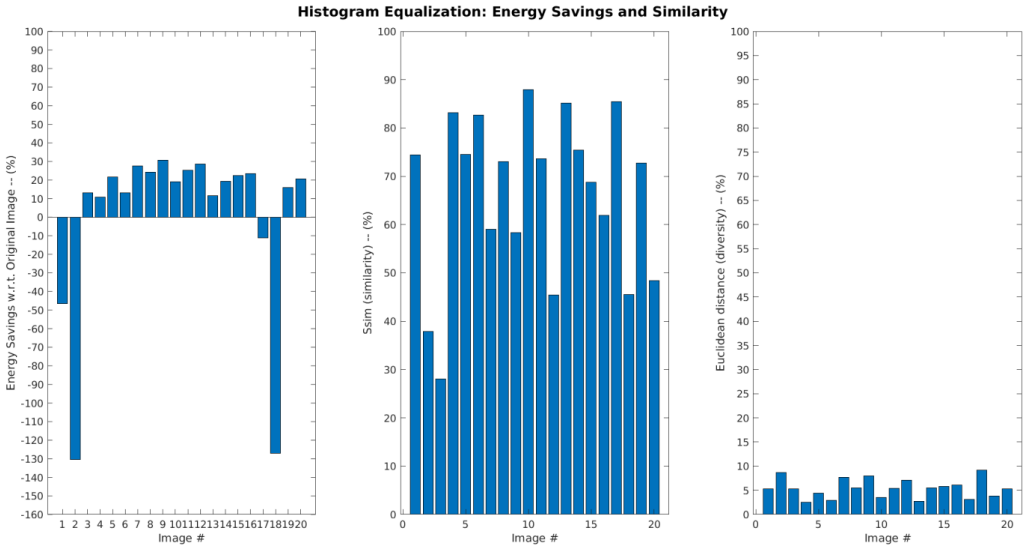

Figure 5: Summarized plot representing, for each image of the set, the power consumption, the Euclidean Distance and the SSIM parameter obtained after the Histogram Equalization.

From now on, we will discuss some of the most interesting results obtained when considering a single image at a time. In figure (6) we can, for example, observe the histogram Equalization applied to the second image of the entire set: this image demonstrated that the Histogram Equalization can in fact increase the power consumption of an image. In this case, we can see that the application of the equalizer "stretches" the entire histogram: in the original images most of the colors were concentrated on the left half of the histogram, demonstrating that the original image is in fact very dark. 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 (7). Here, the original image (iamge #13) presents an histogram which is already "stretched" enough. The application of the equalization hence stretches even more the histogram, especially for the red channel. As a result, the image 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 enhances quality of the image. Such result suggests that an Histogram Equalization transformation is best suited to images that are overall "equilibrate" in terms of color

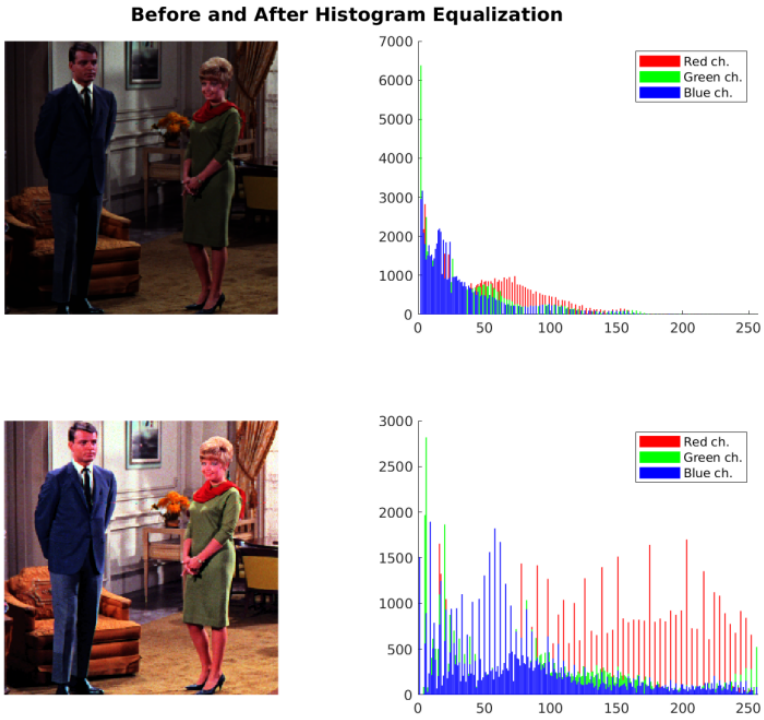

Figure 6: An example of Histogram Equalization that actually increases the power consumption of an image: the color histogram is stretched up to the most bright zones (as a rule of thumb, the more the image is bright, the more its power consumption). This image, once equalized, results in a power consumption increased by 130%.

presence in its 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 (8) 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 is clear even, with a fast comparison, that the both plots (1 and 8) are really similar in terms of curve trends.

In the following figure (9) 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 be pushing 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 (10) and figure (11). 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, suggesting 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 gains hits 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.

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

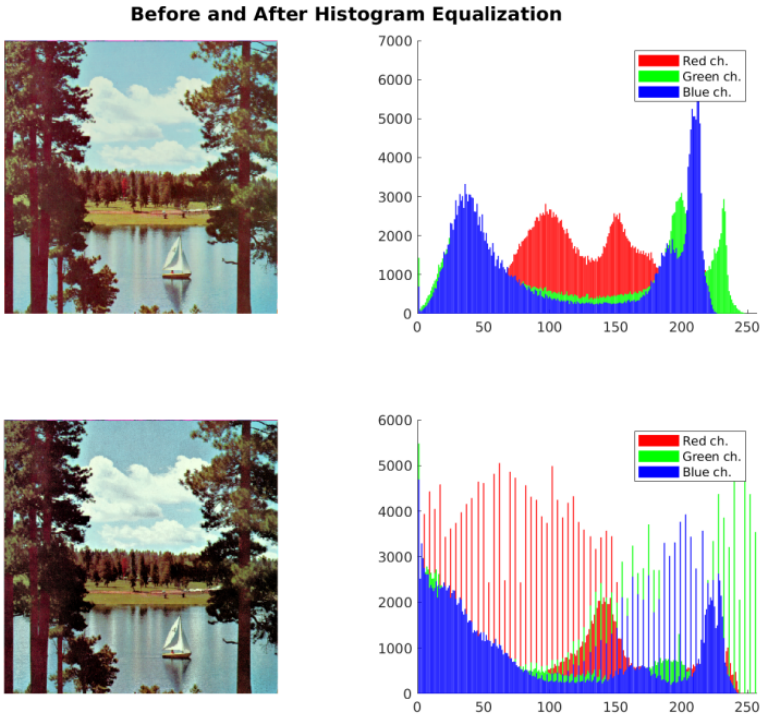

Figure 7: An example of good Histogram Equalization. The image appear nearly the same as the original, but with a power consumption that is reduced by the 30%.

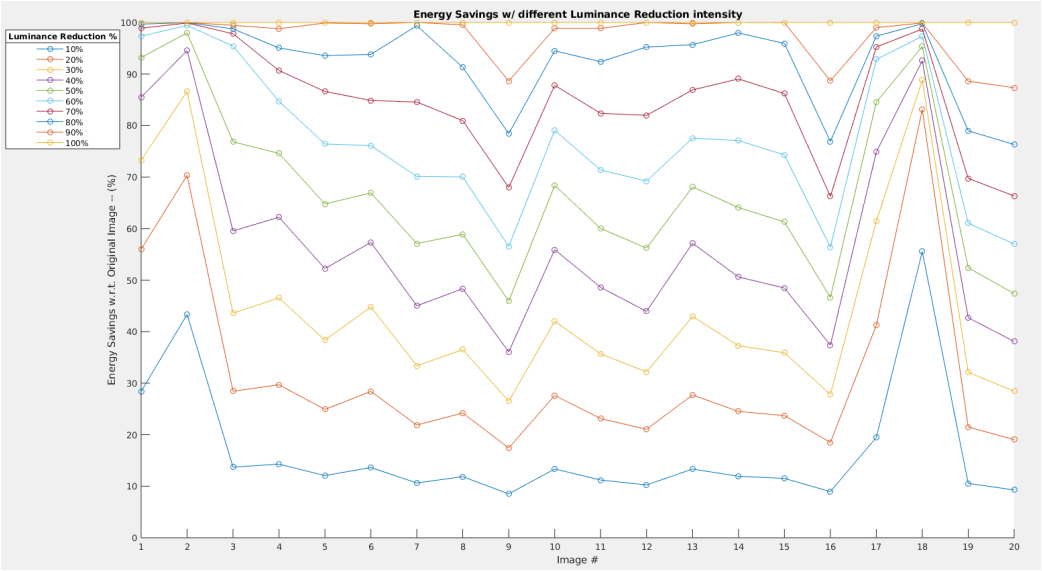

Figure 8: The plot represents, for each of the 15 images (on the x axis) and for each luminance reduction percentage applied to it (the different curves), the overall power consumption gains obtained.

Figure 9: 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.

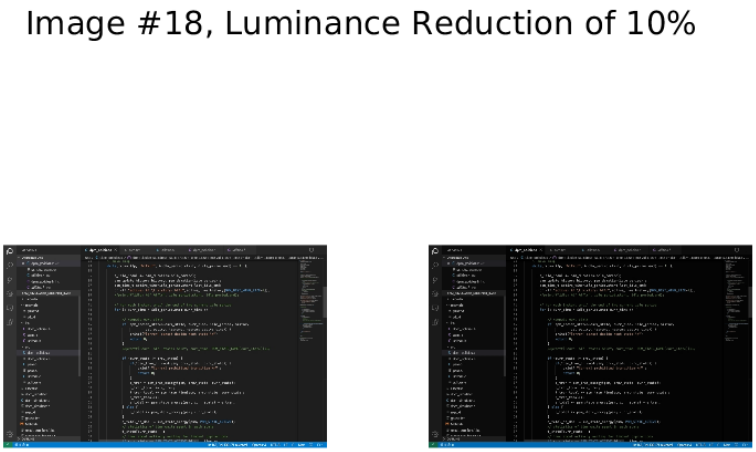

Figure 10: A screenshot of an application supporting a dark theme (image #18).

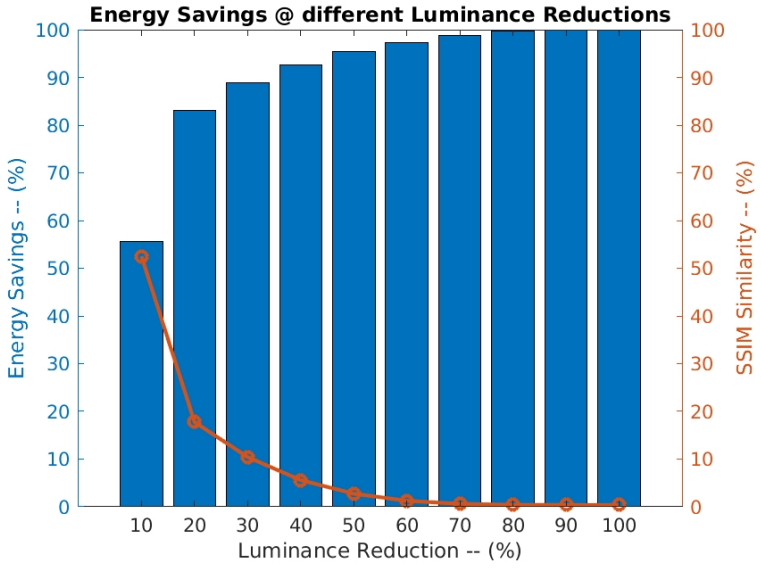

Figure 11: The overall power consumption behavior of the same screenshot of figure (10). We can see that a luminance reduction of the 10% is sufficient to obtain a power consumption which is reduced up to the 55%.

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.

Once the serial object obtained by executing the serial() method, 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 three 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 chars that is used to load into memory a single byte 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 (12) 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.

Figure 12: 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.

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.

Since the color reduction must be applied *on-the-fly*, the computation is done in the `serialEvent()` routine, enclosed between an `#if IMG_EDIT` pre-processor statement: here the reduction value is subtracted to the original value and the new color reduced `data` variable is sent to the display controller as we said in the previous section. In the following picture (13) 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 13: The OLED display representing the same image with the two different color reductions applied: on the left the color reduction done by *Arduino*, on the right the color reduction done by MATLAB.