Group Number:1
Students: Berkay Yaldız, Bevan Deniz Çılğın, Melih Can Zerin

EE433 PROJECT REPORT

In this project, conversion of RGB image into grayscale image and conversion of grayscale image into RGB image are implemented in LabVIEW. Necessary codes are written in C language and dll functions are constructed in LabWindows. These dll functions are called from LabView.

Representation of Images

A digital image is a 2-dimensional signal. These two dimensions indicates the locations of the pixels of the image. It also has color dimension(s) according to a color model such as RGB, grayscale, CMY, HSI etc. Each color dimension can take values between 0 and 255, which indicate the intensity of that color. They can be represented using 8 bits ($256=2^8$). For example, a grayscale image has only color, dimension, therefore each pixel of it is represented by 8 bits in memory. On the other hand, an RGB image has 3 color dimensions, therefore each pixel of it is represented by 8×3=24 bits.

RGB Images

RGB images consist of 3 image dimensions: red (R), green (G) and blue (B). Each dimension takes values between 0 and 255 and the color of a pixel is determined by the combination of these 3 values. When all of them are 0, the color is black and when all of them are 255, the color is white. An extension of the RGB model is RGBA model, which has a fourth dimension, namely alpha (A). Alpha indicates the opacity level. When alpha is 0, the image is totally transparent, when it is 255, the image is totally opaque. Hence, an RGBA image pixel contains 24+8=32 bits. Labview uses ARGB convention when holding the images in the memory.

Grayscale Images

Grayscale images have one color dimension: gray (G). Its pixels take values between 0 and 255. One pixel is represented by 8 bits.

RGB to Grayscale Conversion

Since the dimension of the original image is higher than that of the converted image, the transformation is uniquely represented by the following formula:

$$G = 0.299R + 0.587G + 0.114B \qquad (1)$$

where G in the left hand side is gray pixel value, R is red pixel value, G in the right hand side is green pixel value and B is blue pixel value.

Grayscale to RGB Conversion

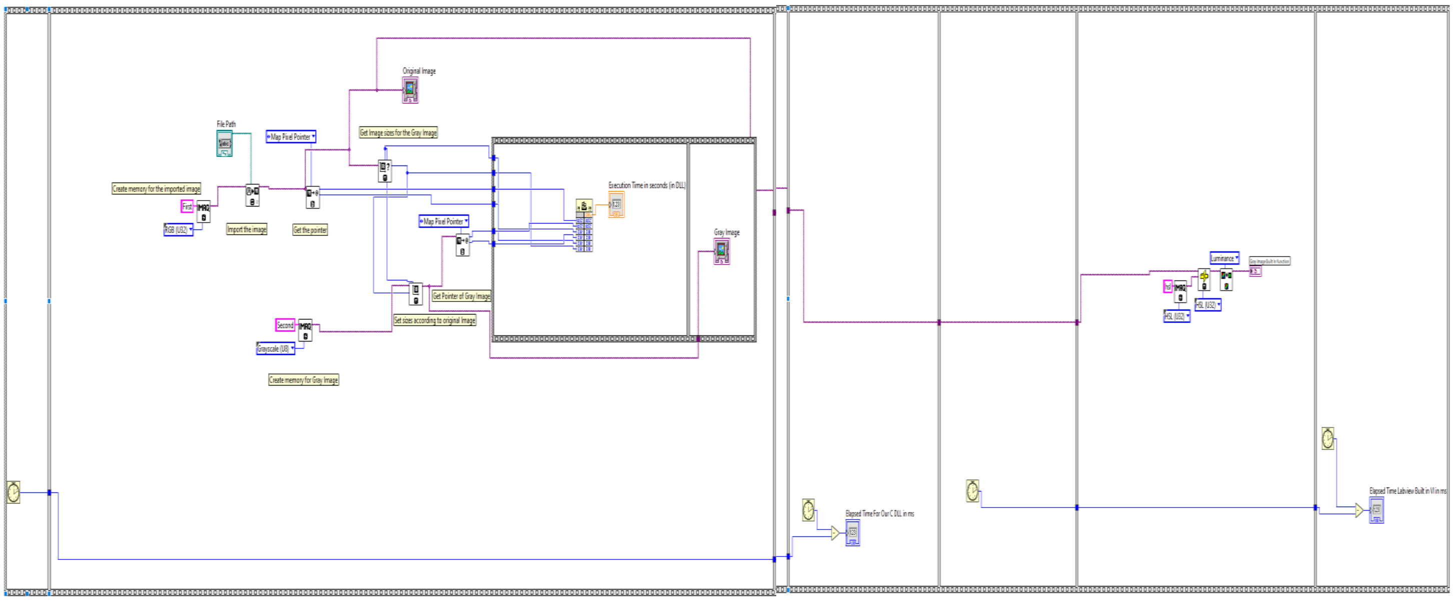
Grayscale to RGB conversion is done by using color maps. Color maps hold to actual color values corresponding to the pixel’s grayscale value. To convert the grayscale image to an RGB image, iteration through pixels of the image should be performed and every grayscale value should be changed with color values according to the color maps. While performing this, it should be noted that the grayscale value is consisting of 8 bits and the ARGB channel is consisting of 32 bits. Thus, obtained ARGB values should be used to form a 32 bit structure by using bit shifting.

Block Diagram of RGBtoGray.vi

In order to get an RGB image from a file and convert it to grayscale, the following procedure is followed:

Group Number:1
Students: Berkay Yaldız, Bevan Deniz Çılğın, Melih Can Zerin

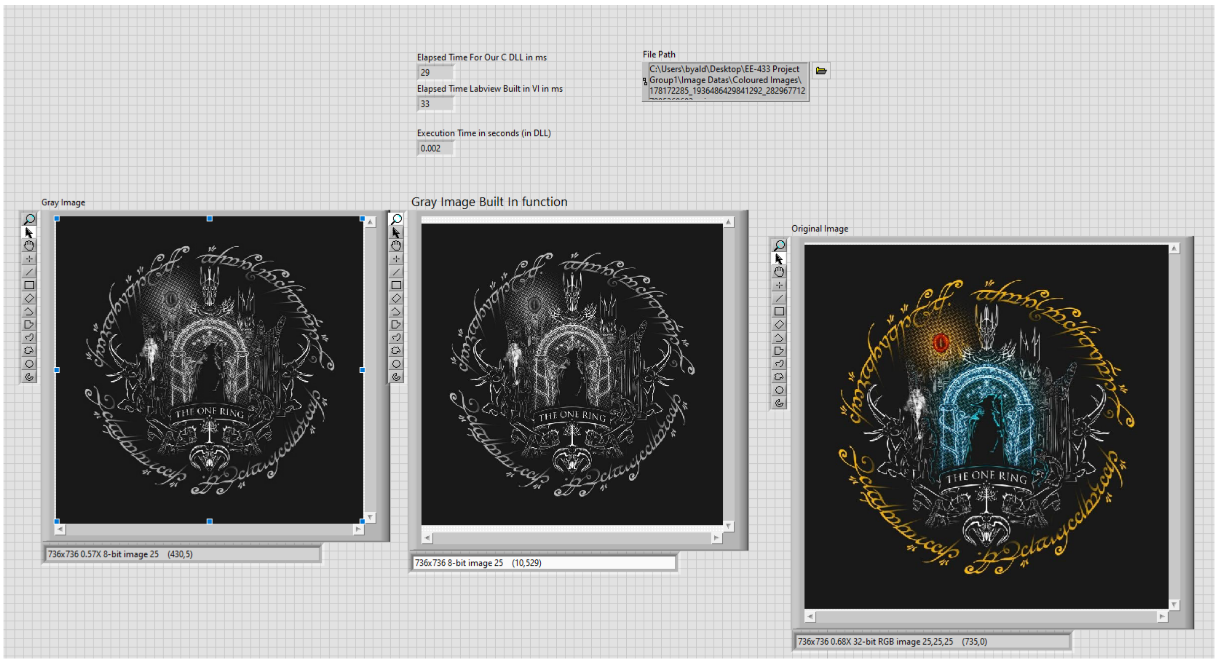
1. An IMAQ Create VI is used to create a memory location for the input RGB image. “Image Type” input is chosen as “RGB (U32)”. “New Image” output of this VI is fed to an IMAQ ReadFile 2 VI.
2. An IMAQ ReadFile 2 VI reads an image file whose location is given from the “File Path” input. The pixels of the image are transformed into the image type given from the “Image” input. The “Image Out” output of this VI is given to a IMAQ GetImagePixelPtr VI as input.
3. An IMAQ GetImagePixelPtr VI is used to obtain the pointers on the pixels of the image given from its “Image” input. The “Pixel Pointer Out” returns the pointer on the pixels. “LineWidth” output of the VI gives the total number of pixels in one row of the input image. (This number is the summation of the horizontal resolution of an image, borders and left and right alignments. Therefore, the horizontal resolution of the image and line width are not necessarily equal.) The two outputs taken from this VI are given to our dll function.
4. An IMAQ GetImageSize VI takes the RGB image as input from “Image” input and outputs the size of it as “X Resolution” and “Y Resolution” values. These outputs are given to the dll function as inputs.
5. Another IMAQ Create VI is used to create a memory location for the output grayscale image. The “Image Type” input is chosen as “Grayscale (U8)”. The “New Image” output of this VI is sent to an IMAQ SetImageSize VI.
6. IMAQ SetImageSize VI adjusts the size of the output grayscale image to the input RGB image size, which is obtained by the IMAQ GetImageSize VI. The output of this block is given to an IMAQ GetImagePixelPtr as input.
7. The IMAQ GetImagePixelPtr gives the pointer on the grayscale output image as output. This output and the “Linewidth” output of this VI are given to the dll function as input.
8. In the dll function, RGB image is converted into grayscale image, using the formula (1). A detailed explanation of the C code inside the dll function is given right after this section. The converted grayscale image and the execution time of the C code are displayed as outputs in the front panel. A flat sequence structure is used to first run the code and get the execution time, and then display the image using the memory location information of it taken from the IMAQ Create VI output.
9. In order to calculate the execution time of the whole process explained above, which is the time of our code to convert an RGB image into grayscale image, flat sequence structure is used. In the first subdiagram, a Tick Count (ms) Function is placed. In the second subdiagram, the whole block diagram of the process explained above is executed. In the third subdiagram, another Tick Count (ms) Function is placed. The output of the first Tick Clock is subtracted from the second one and the execution time of our code is obtained. In the fourth and sixth subdiagrams, Tick Count (ms) Functions are placed. In the fifth subdiagram, we used the first method in [1] to convert RGB image to grayscale using built-in LabVIEW functions, which is converting RGB image into HSL image and then extracting the L plane. To do this, first IMAQ Create VI is used to allocate memory for HSL image. Then, IMAQ Cast Image VI is used to convert RGB image into HSL image. After that, IMAQ ExtractSingleColorPlane VI is used to extract the luminance plane of the HSL image. The output of this VI is displayed as the converted grayscale image output. The time difference between the Tick Count outputs of the fourth and sixth subdiagrams is calculated in the sixth subdiagram, which is the time taken by the built-in LabVIEW functions to convert an RGB image into grayscale image.


Original Image
File Path
Map Pixel Pointer
Get Image sizes for the Gray Image
Create memory for the imported image
First
RGB (U32)
Import the image
Get the pointer
Map Pixel Pointer
Execution Time in seconds (in DLL)
Gray Image
Get Pointer of Gray Image
Set sizes according to original Image
Second
Grayscale (U8)
Create memory for Gray Image
Luminance
Gray Image Built In function
HSL (U32)
Elapsed Time For Our C DLL in ms
Elapsed Time Labview Built in VI in ms

RGBtoGray.dll

- We iterate through the input image and read every pixel value in hexadecimal form. Consequently, we obtain a 32-bit structure. Byte 0, byte 1, and byte 2 of this structure correspond to the blue, green, and red components of this pixel, respectively.
- To obtain these RGB values, bitwise and (&) and shift (>>) operators are used. The bytes other than the one corresponding to the desired channel are masked. For example, if the blue channel is wanted, 0x000000FF is used for masking and byte 0 is obtained only. In this case, it is not necessary to use a shift operation since the obtained result is in the byte 0 position. However, we need to use shift red and green components to obtain correct values. Red components are shifted by 16 bits and green components are shifted by 8 bits.
- RGB channel values are used to find the grayscale value. The formula for this conversion is $G = 0.299R + 0.587G + 0.114B$. The obtained value is assigned to the grayscale image's corresponding pixel.
- After one row is completely read, the algorithm moves to the next row by increasing the current location in the image by LVLineWidthX – LVWidth (X is either RGB or Gray). Subtracting LVWidth enables returning back to the beginning of the row that was read and adding LVLineWidthX makes the beginning of the next row the new location.

Results

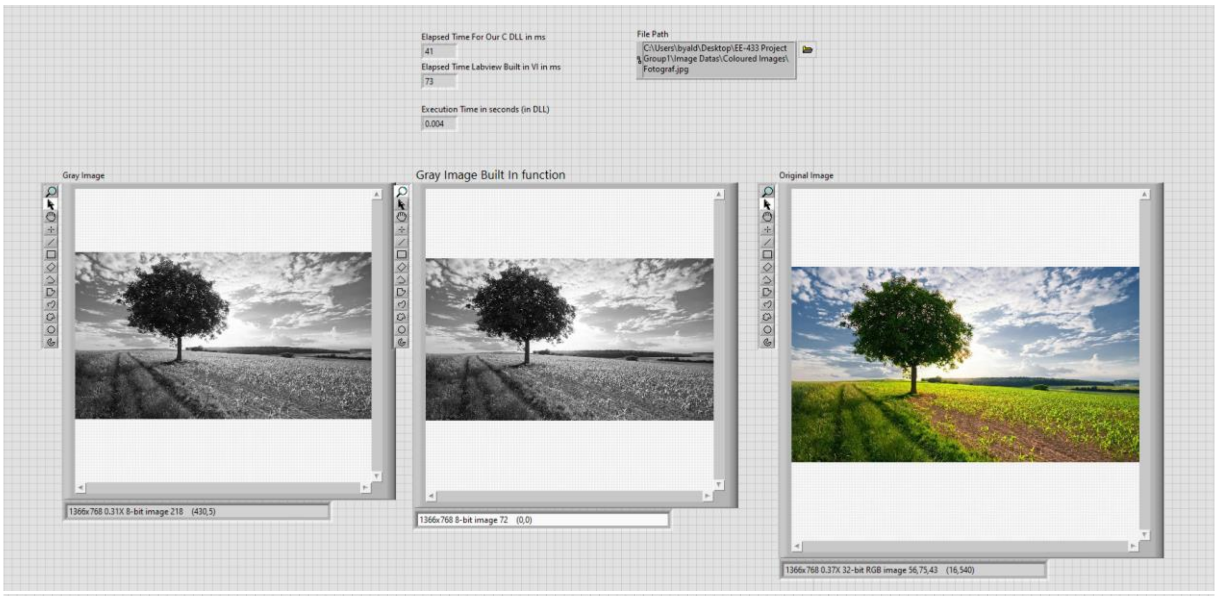

Elapsed Time For Our C DLL in ms
29
Elapsed Time Labview Built in VI in ms
33
Execution Time in seconds (in DLL)
0.002
File Path
C:\Users\byald\Desktop\EE-433 Project Group1\Image Datas\Coloured Images\178172285_1936486429841292_282967712

Gray Image
736x736 0.57X 8-bit image 25 (430,5)

Gray Image Built In function
736x736 8-bit image 25 (10,529)

Original Image
736x736 0.68X 32-bit RGB image 25,25,25 (735,0)

Group Number:1
Students: Berkay Yaldız, Bevan Deniz Çılğın, Melih Can Zerin



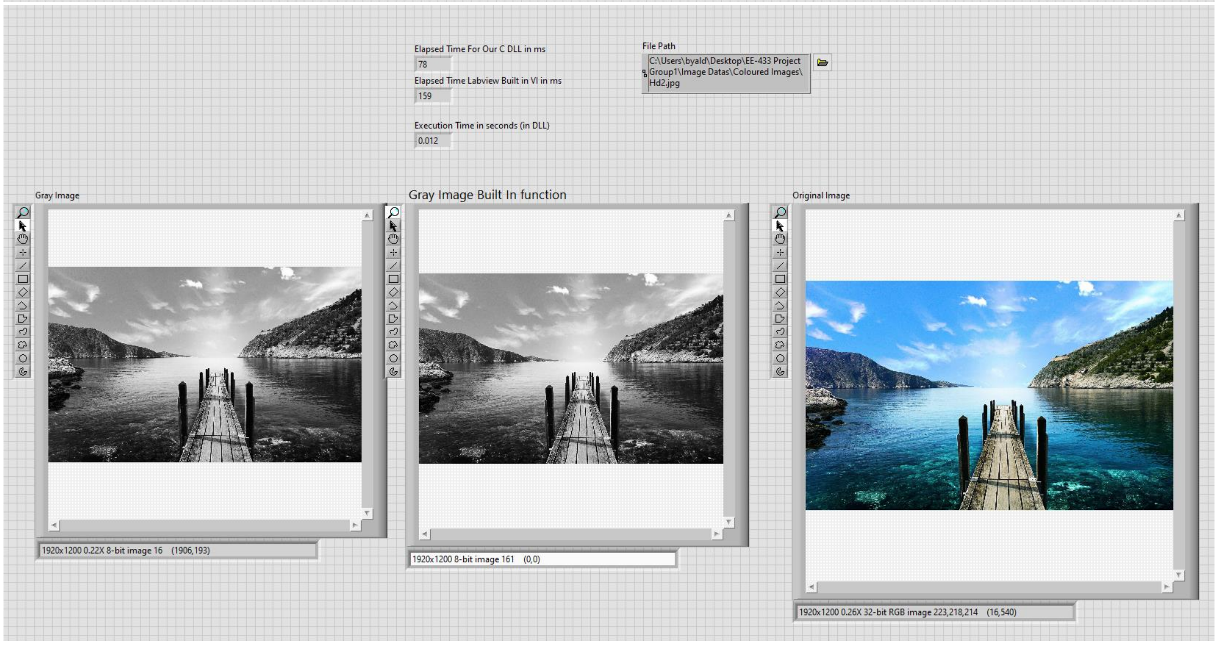
Elapsed Time For Our C DLL in ms
41
Elapsed Time Labview Built in VI in ms
73
Execution Time in seconds (in DLL)
0.004

File Path
C:\Users\byald\Desktop\EE-433 Project Group1\Image Datas\Coloured Images\Fotograf.jpg

Gray Image
1366x768 0.31X 8-bit image 218   (430,5)

Gray Image Built In function
1366x768 8-bit image 72   (0,0)

Original Image
1366x768 0.37X 32-bit RGB image 56,75,43   (16,540)



Elapsed Time For Our C DLL in ms
78
Elapsed Time Labview Built in VI in ms
159
Execution Time in seconds (in DLL)
0.012

File Path
C:\Users\byald\Desktop\EE-433 Project Group1\Image Datas\Coloured Images\Hd2.jpg

Gray Image
1920x1200 0.22X 8-bit image 16   (1906,193)

Gray Image Built In function
1920x1200 8-bit image 161   (0,0)

Original Image
1920x1200 0.26X 32-bit RGB image 223,218,214   (16,540)

As we can see from the results, as the dimensions of the image increase, our dll implementation performs better and better than the built in vi.

Block Diagram of GraytoRGB.vi

- IMAQ Create VI is used to allocate memory for the input image and IMAQ ReadFile 2 VI imports the gray image.
- Imported image is given as input to the IMAQ GetImagePixelPtr VI to obtain a pointer on the pixels of an image. Moreover, it gives the LineWidth of the image as an output which is the number of pixels in a row of the input image. This parameter is necessary for performing iteration on the pixels of the image and it is used in our shared libraries.
- We expect an RGB image as an output. Thus, memory is allocated for this RGB image by using IMAQ Create VI. The size of this image should be the same as the gray image.

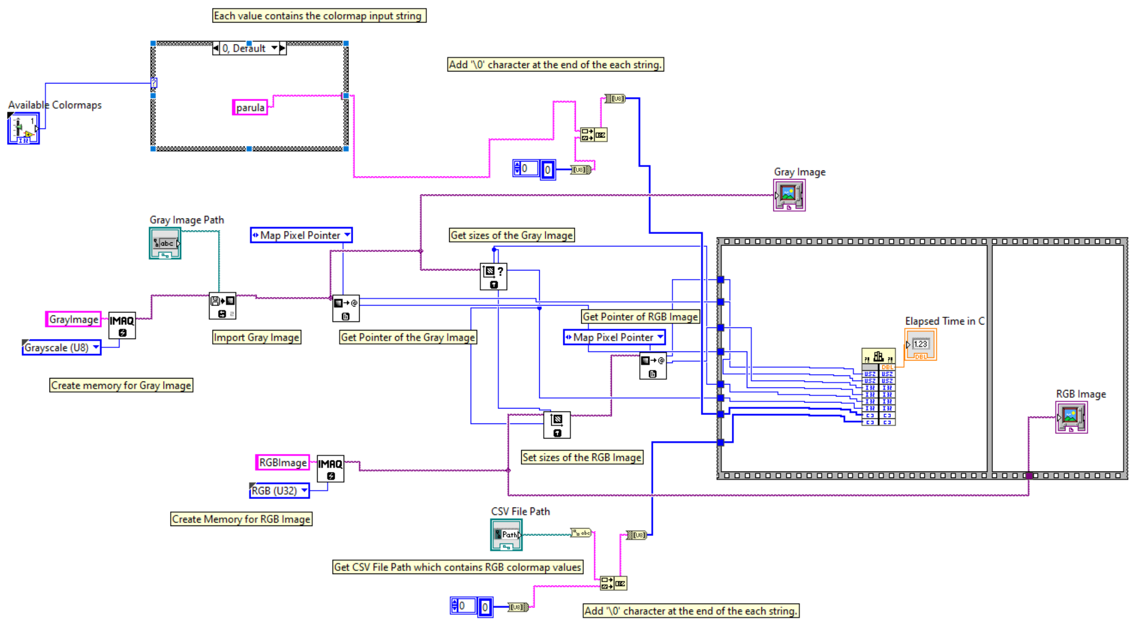
Group Number:1
Students: Berkay Yaldız, Bevan Deniz Çılğın, Melih Can Zerin

To arrange the size of our output image, we determine the gray image's size by IMAQ GetImageSize VI and find LVWidth and LVHeight values. Using these values and IMAQ SetImageSize VI, RGB image size is determined.

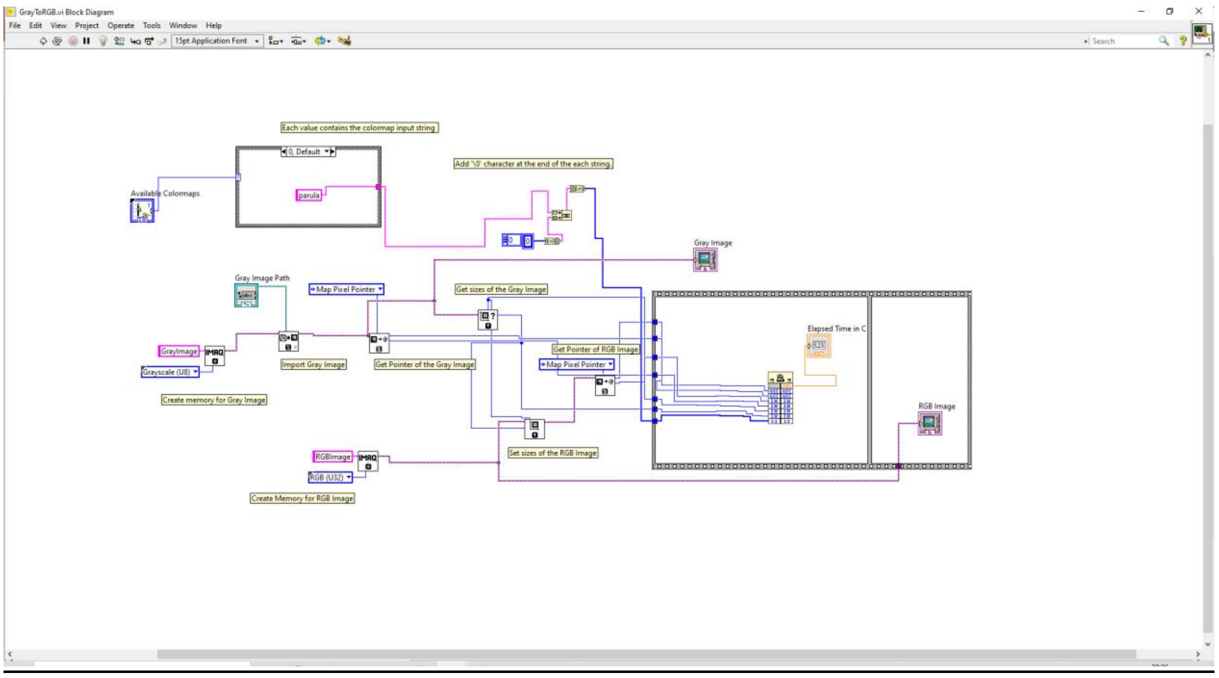
- After these arrangements for the RGB image, we get the pointer on the pixels of the image and LineWidth of the image.

- In addition to the images, a path file of the CSV file that contains the RGB channel values for color maps is provided and a case structure to enable a user to select the color map type is constructed. However, this approach is not efficient especially for the real-time and video processing, so we decided to embed necessary color map matrices to dll and use it according to user input without needing to open, close the csv file over and over again, travelling unnecessary color maps. Both implementations can be seen in our project.

- Finally, RGBimage, Grayimage, LVLineWidthRGB, LVLineWidthGray, LVWidth,LVHeight,ColorMap and CSVPath are given as input for the Gray_to_RGB.dll.


Each value contains the colormap input string
0, Default
parula
Available Colormaps
Add '\0' character at the end of the each string.
Gray Image
Gray Image Path
Map Pixel Pointer
Get sizes of the Gray Image
GrayImage
Grayscale (U8)
Import Gray Image
Get Pointer of the Gray Image
Get Pointer of RGB Image
Map Pixel Pointer
Elapsed Time in C
Create memory for Gray Image
RGBImage
RGB (U32)
Set sizes of the RGB Image
RGB Image
Create Memory for RGB Image
CSV File Path
Get CSV File Path which contains RGB colormap values
Add '\0' character at the end of the each string.

Group Number:1
Students: Berkay Yaldız, Bevan Deniz Çılğın, Melih Can Zerin


GrayToRGB.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Each value contains the colormap input string
0, Default
Available Colormaps
parula
Add '\0' character at the end of the each string.
Gray Image
Gray Image Path
Map Pixel Pointer
Get sizes of the Gray Image
GrayImage
IMAQ
Grayscale (U8)
Import Gray Image
Get Pointer of the Gray Image
Get Pointer of RGB Image
Map Pixel Pointer
Elapsed Time in C
Create memory for Gray Image
Set sizes of the RGB Image
RGB Image
RGBImage
IMAQ
RGB (U32)
Create Memory for RGB Image

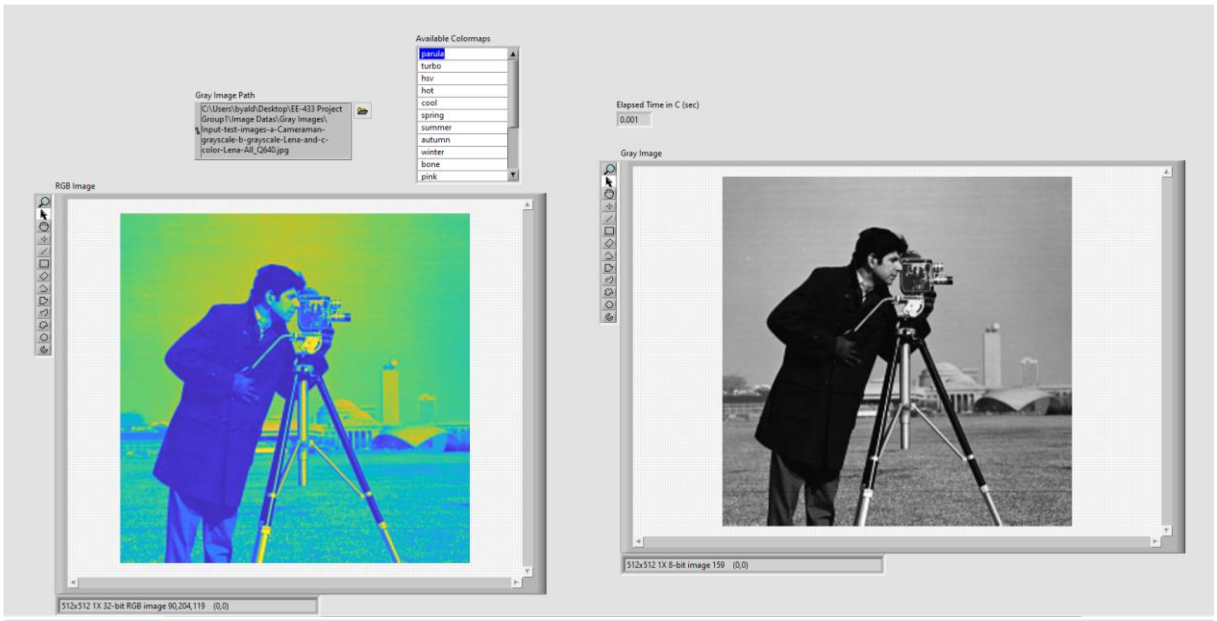
Second implementation where it does not need a csv file.

GraytoRGB.dll

- Color map matrices hold the RGB channel values for related grayscale values. We found these matrices by using Matlab, converted to uint8, and put them in a CSV file to use in the algorithm (We embedded color map matrices directly into our second implementation dll and used it accordingly, which is much faster).
- According to input color map type, related columns of the CSV file are put in a matrix called C.
- After these steps, iteration on the grayscale image is performed and every pixel's value is read. Then, RGB channel values are extracted from the matrix C. RGB channel values are located at the row that has the number equal to the pixel's grayscale value.
- Obtained channel values are used to construct the 32-bit structure. Bytes that correspond to alpha are selected as FF and shifted left by 24 bits. Red and green values are shifted left by 16 and 8 bits respectively. Shift operation is not performed on blue value since it is already located at byte 0.
- The constructed structure is assigned as the corresponding pixel's value in the RGB image that is being constructed.
- After operations are done for a row, the algorithm moves to the next row by using LVLineWidth and LVWidth. When the iteration ends for the grayscale image, RGB image construction is done.

Group Number:1
Students: Berkay Yaldız, Bevan Deniz Çılğın, Melih Can Zerin

Results

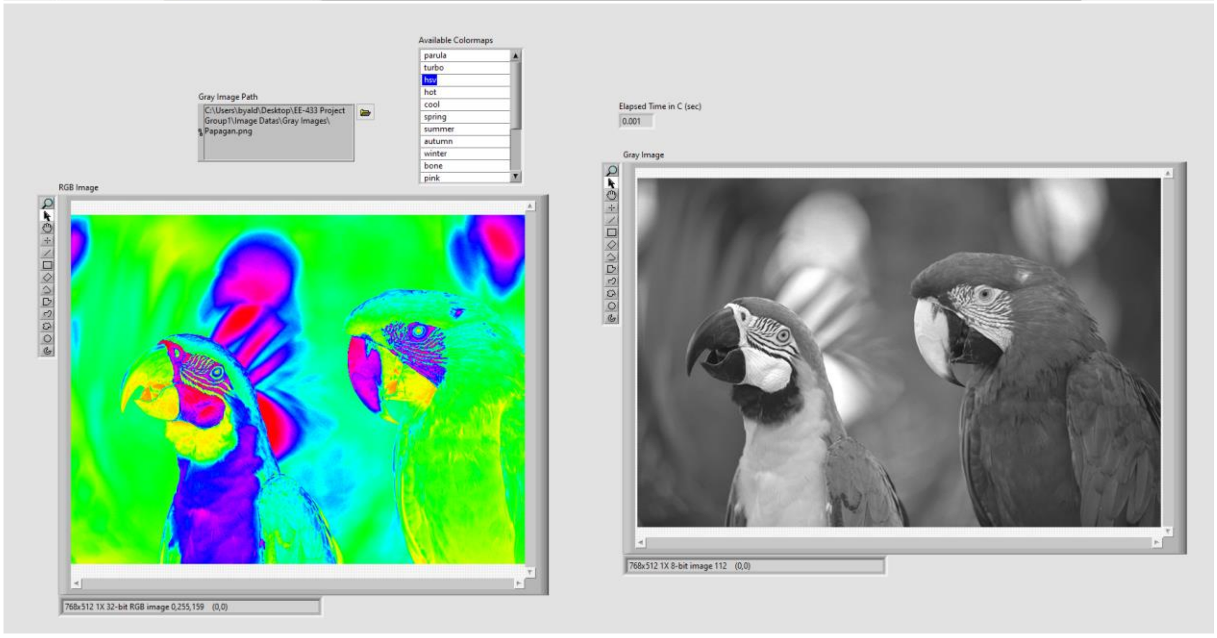

Gray Image Path
C:\Users\byald\Desktop\EE-433 Project Group1\Image Datas\Gray Images\Input-test-images-a-Cameraman-grayscale-b-grayscale-Lena-and-c-color-Lena-All_Q640.jpg

Available Colormaps: [✓ parula] turbo hsv hot cool spring summer autumn winter bone pink

Elapsed Time in C (sec): 0.001

RGB Image: 512x512 1X 32-bit RGB image 90,204,119 (0,0)

Gray Image: 512x512 1X 8-bit image 159 (0,0)


Gray Image Path
C:\Users\byald\Desktop\EE-433 Project Group1\Image Datas\Gray Images\Papagan.png

Available Colormaps: parula turbo [✓ hsv] hot cool spring summer autumn winter bone pink

Elapsed Time in C (sec): 0.001

RGB Image: 768x512 1X 32-bit RGB image 0,255,159 (0,0)

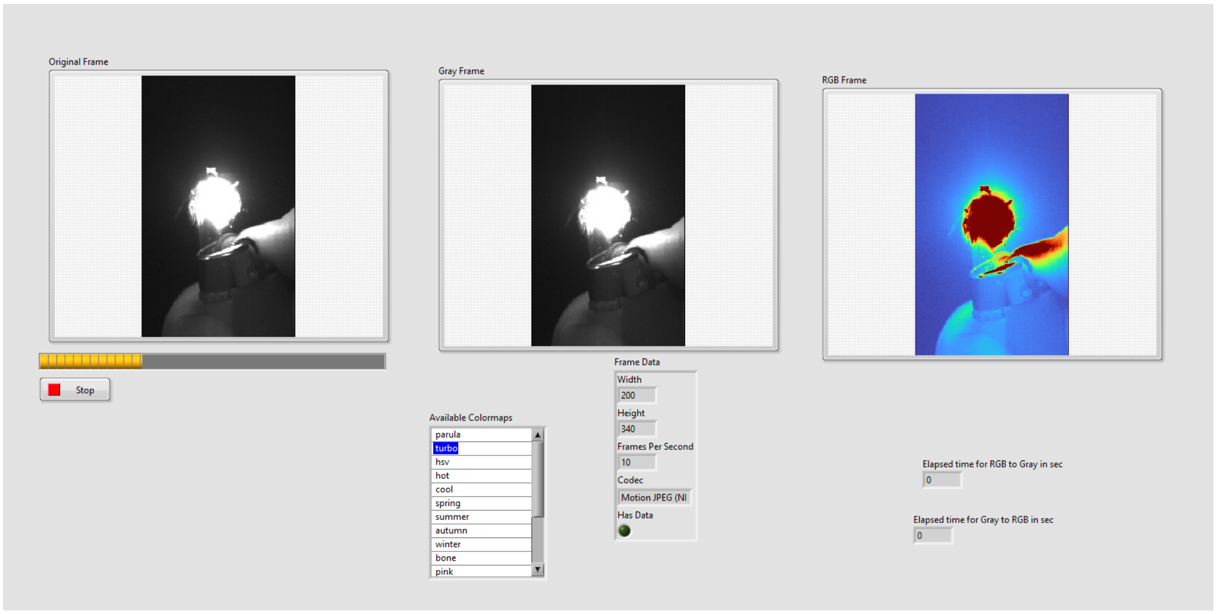
Gray Image: 768x512 1X 8-bit image 112 (0,0)

ImportedVideoConversion.vi

In order to convert the color model of a video file input, we take an AVI file as input. Since a video is actually an image set (at each time instant, there is an image), we make the conversion using these images. As in the image color conversion cases, we first allocate memory for an image, then we import the image at the first time instant of the video and make the color conversion. For the following time instances, we do not allocate memory for the corresponding images again and again, because after displaying the image at the previous time instant, we use the same memory location for the image at the next time instant. Therefore, setting the dimensions of output image to those of the input image is also not done again and again.

The part for reading an AVI file is taken from the following LabVIEW example: [2]

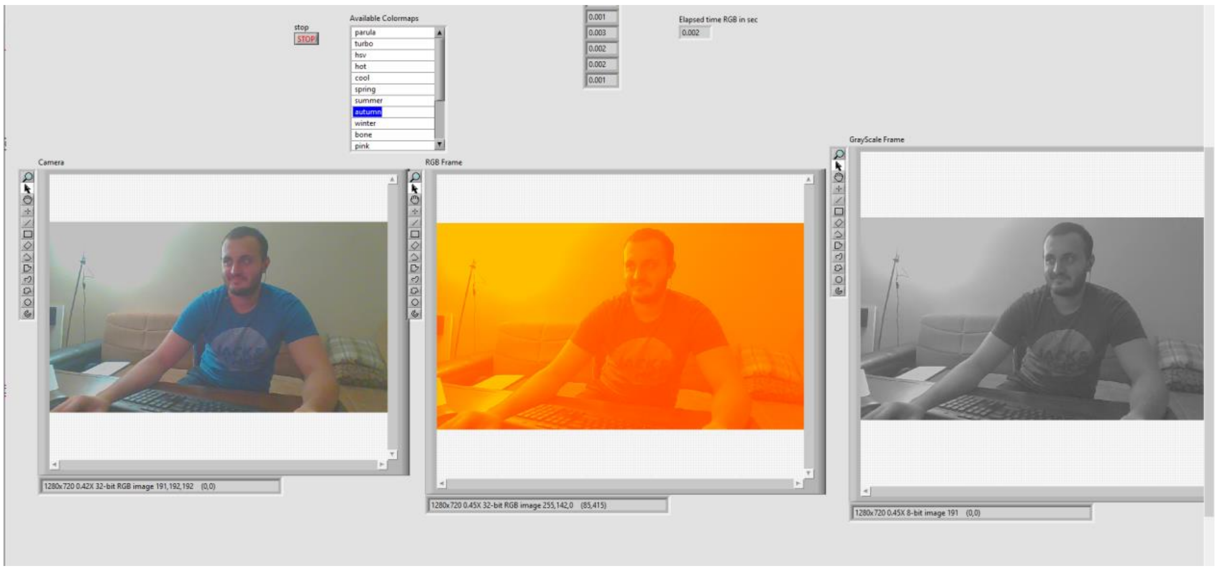
Group Number:1
Students: Berkay Yaldız, Bevan Deniz Çılğın, Melih Can Zerin



Width 200
Height 340
Frames Per Second 10
Codec Motion JPEG (NI
Has Data ●
Elapsed time for RGB to Gray in sec 0
Elapsed time for Gray to RGB in sec 0

RealTimeConversions.vi

In order for LabVIEW to reach the camera of the device, IMAQdx Open Camera VI and IMAQdx Configure Grab VI are used. IMAQdx Grab2 VI takes the most current frame to the output in a for loop. After taking the frame, the process is same as in the ImportedVideoConversion.vi.



0.001
0.003
0.002
0.002
0.001
Elapsed time RGB in sec 0.002
1280x720 0.42X 32-bit RGB image 191,192,192 (0,0)
1280x720 0.45X 32-bit RGB image 255,142,0 (85,415)
1280x720 0.45X 8-bit image 191 (0,0)

References

[1] https://forums.ni.com/t5/Example-Code/IMAQ-Color-Image-to-Grayscale/ta-p/3535022?profile.language=en

[2] https://zone.ni.com/reference/en-XX/help/370281AG-01/imaqvision/imaq_avi_read_frame/