**New Bulgarian University**

**Department of Informatics**

**Program:** "Networking Technologies"

---

# Project

**On**

**NETB603: Programming Practice**

**On the topic of**

***Colour Image Processing***

Sofia, Bulgaria

2023

## Team B

## Problem 1:

### Colour Image Processing

Using programming language C++ and Qt framework implement an application that processes color images. The application has the following features:

- Graphical user interface that allows the user to open image files and process it;
- It supports the majority of standard raster image formats (jpg, png, bmp, tiff) and can both load and save these type of files;
- The program supports the following color models: RGB, CMY, CMYK and HSI;
- The program allows the user to the image pixel data between the above color models;
- The program displays the histograms for any of the color models;
- The program allows the user to modify a given value of a given color channel globally for the entire image, and displays the resulting image;
- The resulting image can be stored as a file.

# Solution:

## Contents

1. Prerequisites ....................................................................................................4
2. Overview ....................................................................................................4
3. How does the application work ....................................................................................................4
4. Project structure....................................................................................................8
5. Source code ....................................................................................................9
5.1. Colour Models....................................................................................................9
5.2. Color Conversion ....................................................................................................9
5.2.1 RGB to CMY to RGB ....................................................................................................10
5.2.2 RGB to CMYK to RGB ....................................................................................................12
5.2.3 RGB to HSI to RGB....................................................................................................14
5.3 Histogram widget ....................................................................................................16
5.4 Mainwindow ....................................................................................................20

## 1. Prerequisites

To build the application successfully the developer would need:

- QT 6.4.3 or higher
- QT Charts package
- tempFiles folder into build directory

## 2. Overview

The project was developed using the C++ language and QT framework. The aim of the application is to process colour images. The following features have been implemented:

- Graphical user interface that allows the user to open image files, process them and save the changes;
- Support for the majority of standard raster formats for both “load” and “save” options
- Support of the following colour models: RGB, CMY, CMYK and HSI
- Visualization of histograms for all of the supported colour models
- Image colour transformation, depending on the selected colour model
- Optional change of the red, green and blue colour channels
- Optional colour pixel swap

## 3. How does the application work

Upon starting the application the main working window is loaded. It is equally divided in two. The left half is responsible for opening, saving and displaying an image and the right half visualizes the histogram of that image.

A dropdown menu is present on the right side, as well. It allows the user to make a selection from a number of colour models – RGB, CMY, CMYK and

HSI, which will then change the histogram and the picture's colours, in the event a colour model, different from RGB is chosen.

On the far left we have an options bar, containing several buttons. The first three let the user change the colour channel of the loaded picture. The fourth one prompts the user to pick two colours, the first one will be found and the second one will be put in its place. The next button will revert the image back to its original state and reload the histogram. And finally, the last button will display a text box with all of the key combination shortcuts that are recognized by the program and their function.

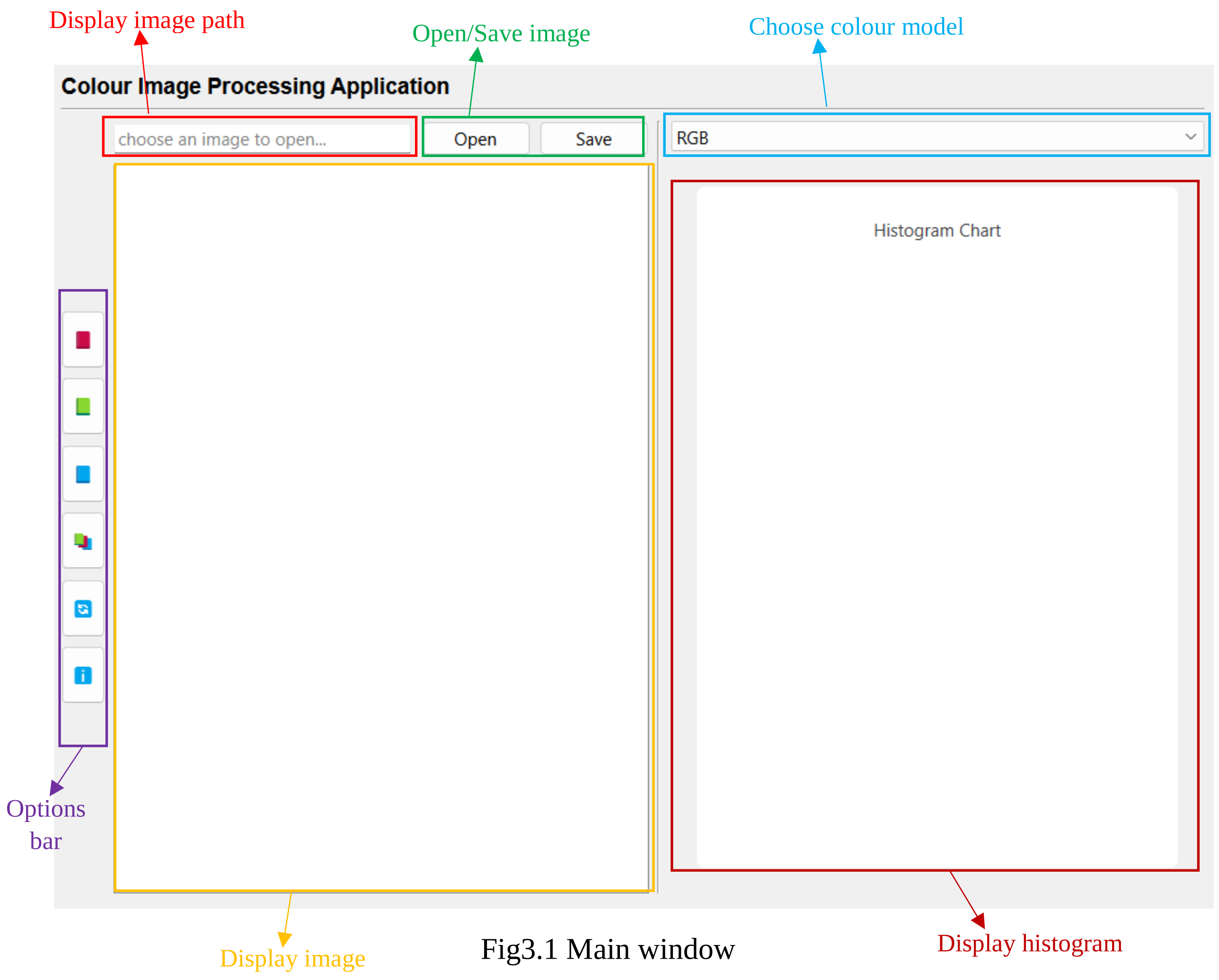


Fig3.1 Main window

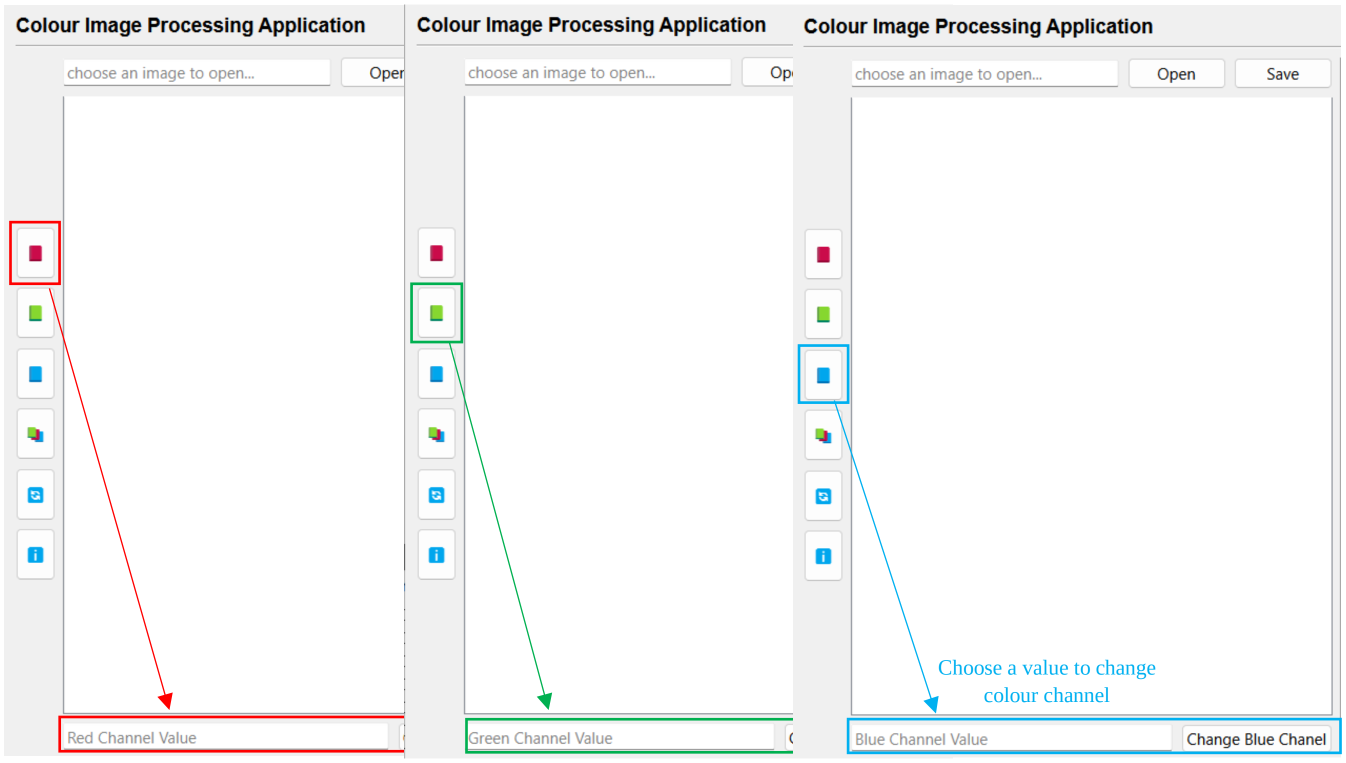

Colour Image Processing Application
choose an image to open...
Open
Save
Red Channel Value
Green Channel Value
Blue Channel Value
Change Blue Chanel
Choose a value to change colour channel

Fig3.2 Channel colour change

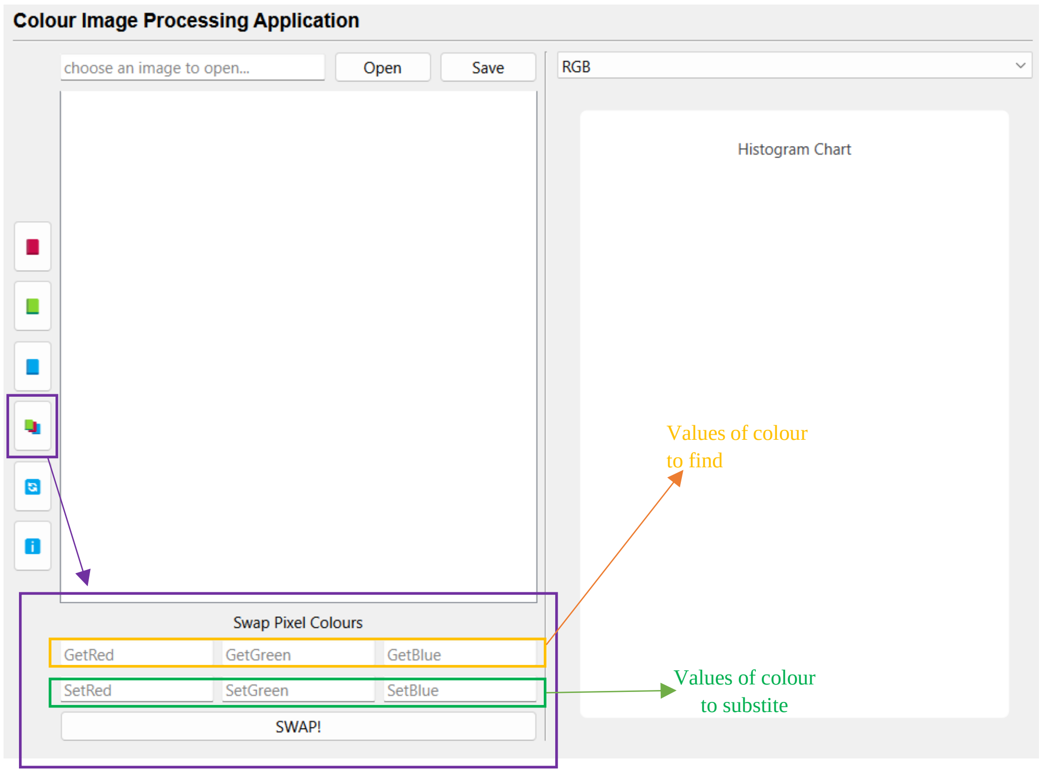


Fig3.3 Single Colour Change

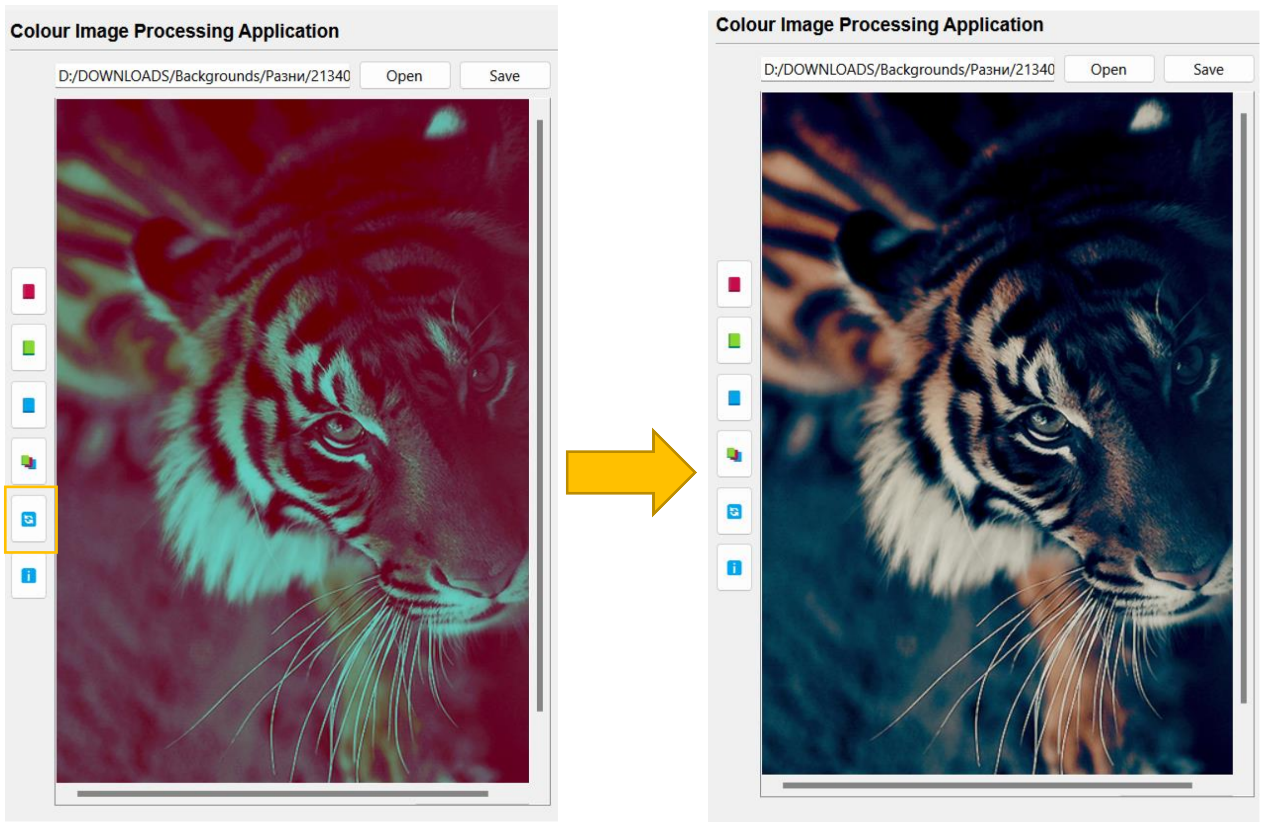
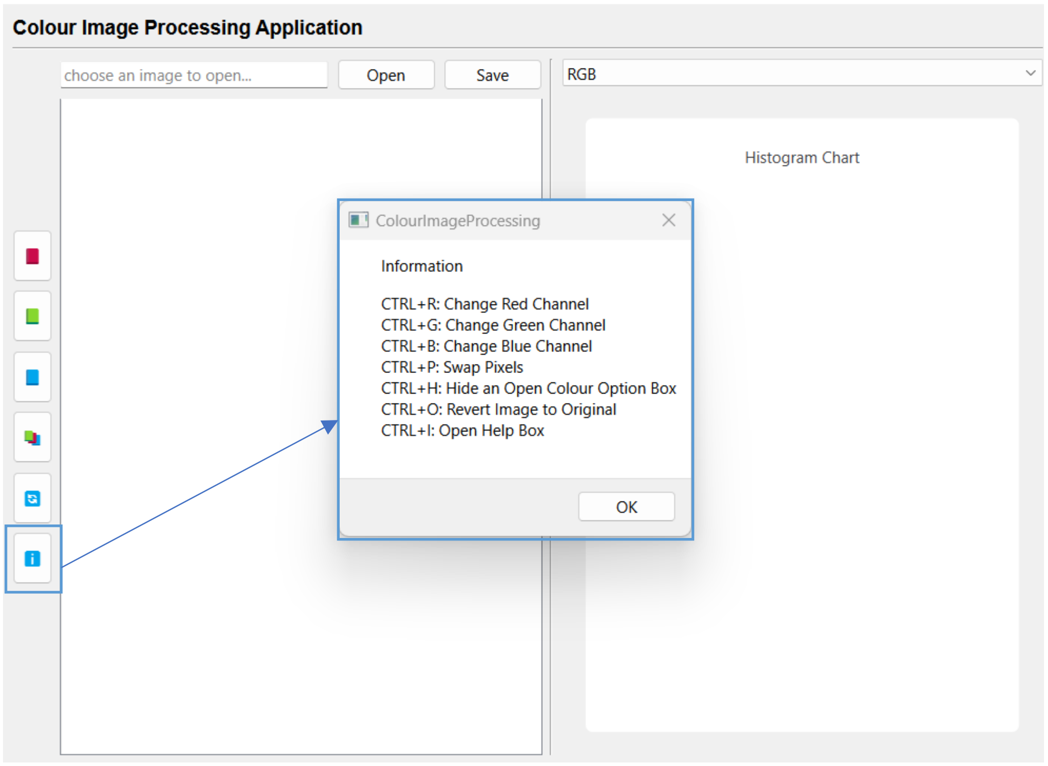

Colour Image Processing Application
D:/DOWNLOADS/Backgrounds/Разни/21340
Open
Save
Colour Image Processing Application
D:/DOWNLOADS/Backgrounds/Разни/21340
Open
Save

Fig3.4 Revert image back to original


Colour Image Processing Application
choose an image to open...
Open
Save
RGB
Histogram Chart
ColourImageProcessing
OK

Information

CTRL+R: Change Red Channel
CTRL+G: Change Green Channel
CTRL+B: Change Blue Channel
CTRL+P: Swap Pixels
CTRL+H: Hide an Open Colour Option Box
CTRL+O: Revert Image to Original
CTRL+I: Open Help Box

Fig.3.5 Information

## 4. Project structure

The project consists of 16 files in total – 8 headers and, respectively, 8 .cpp files. The four colour models have been divided into four individual classes. These colour model classes have all been used in the ColourConversion class, which holds all of the formulas that were needed for the model conversions to be made. The formulas in question are then put to use in the HistogramWidget class, which is responsible for the visualization of the histograms as well as the change of the loaded picture’s colour, all determined by the currently chosen colour model. The MainWindow class is the class that creates the GUI and implements the HistogramWidget, together with some additional functions for colour channel change and colour swap. On fig.4.1 a structure diagram is presented.

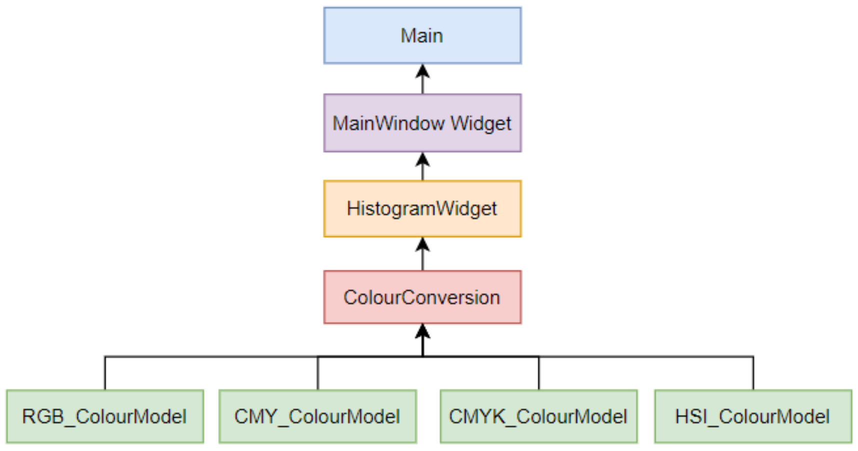


Fig.4.1

# 5. Source code

## 5.1. Colour Models

There are four classes responsible for the four different colour models – RGB_ColourModel.h/.cpp, CMY_ColourModel.h/.cpp, CMYK_ColourModel.h/.cpp and HIS_ColourModel.h/.cpp. These classes are similar in structure.

All of the classes have a default constructor and a copy constructor, which accepts a set of values, corresponding to the number of colours the chosen model uses, for RGB, CMY, HSI – three and for CMYK – four. A copy constructor has been created as it is later used in the ColorConversion class and HistogramWidget where a copy initialization of the objects is made.

Furthermore, all of the classes hold setters and getters for all of the colours of the models. There is also a private section, which has three/four parameters, which are set with the set functions and can be returned with the get functions. At the bottom of each header there is a friend class declaration of ColorConversion. This allows ColorConversion to have access to all of the members – public and private, of all of the colour model classes, as we want to be able to fully modify the parameters that the objects hold.

## 5.2. Color Conversion

The colour conversion class is the class that is fully responsible for all of the calculations that are needed for the histogram visualization and image change. This class holds several functions: RGB to CMY, RGB to CMYK, RGB to HSI and their respective reverse functions. All of the functions accept a colour model object as a parameter and return the new one that has been created.

## 5.2.1 RGB to CMY to RGB

The RGB colour model is a model, in which there are three primary colours – red, green and blue, which are added together in various ways to reproduce a broad selection of colours. While CMY, which is frequently associated with colour printing, uses cyan, magenta and yellow as the primary colours, and red, green and blue are left as secondary ones.

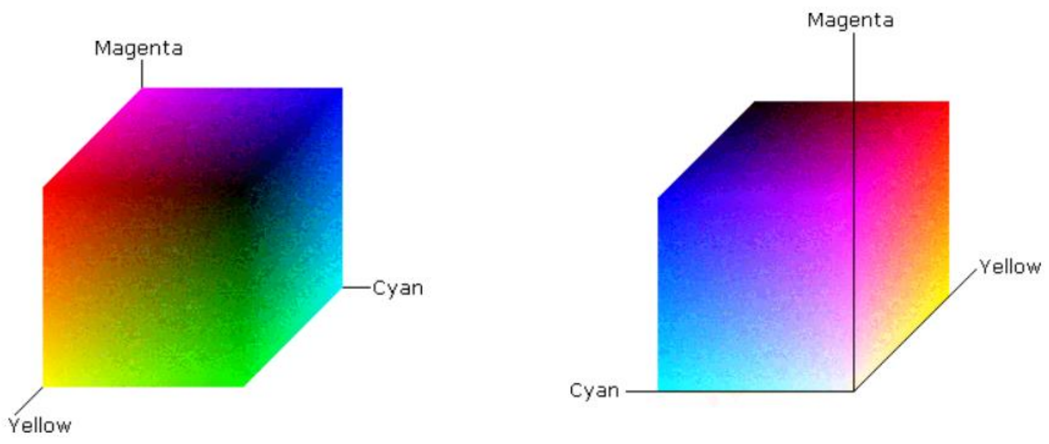


Fig.5.2.1 Colour representations of the CMY colour space

Contrary to RGB, the CMY model is a subtractive, it works by masking colours on a lighter surface, most commonly white. The ink reduces the light that would otherwise be reflected. It is defined as “subtractive” due to the fact that it “subtracts” the red/green/blue ink from white light: white light minus red, returns cyan, white light minus green returns magenta and white light minus blue returns yellow. Having this in mind, the formula that has been used in the code in the RGBtoCMY function is the following:

- Cyan = (1 – ( Red / 255 )) * 100
- Magenta = (1 – (Green / 255)) * 100
- Yellow = (1 – (Blue / 255)) * 100

The method that this formula is in, accepts an RGB colour model as a parameter, from which the red, green and blue values, for the CMY ones to be calculated, are taken.

There is also a qRound called in every line, as all of the colours are declared as doubles, and qRound helps loose less precision between double and int conversions. All of the values are multiplied by 100, so that the histogram visualization is easier.

Something extra that can be noticed at the end of the function is an equation:

- a = (a /100) * a

This equation has been done for all of the three colours – cyan, magenta and yellow. The purpose of this additional calculation is to take a fraction (/percentage) of the newly converted colour, so that later on when the conversion from CMY to RGB is done there is an actual difference in the colours and the change can be visually reflected onto the image. This equation does not change the state of the Histogram, since the values there are reflected as percentages, and the change that is applied is constant.

The RGB to CMY conversion is used to get the values that the Histogram will be based on. To change the colours of the loaded image, a reverse conversion from CMY to RGB is done – CMYtoRGB(CMY model). The function accepts a CMY colour model object as a parameter and returns an RGB object. This inversion is executed, because, as mentioned above the CMY colour model is used for printing, while all of the colours displayed by modern monitors utilize red, green and blue as primary colours. The formula used in this method is:

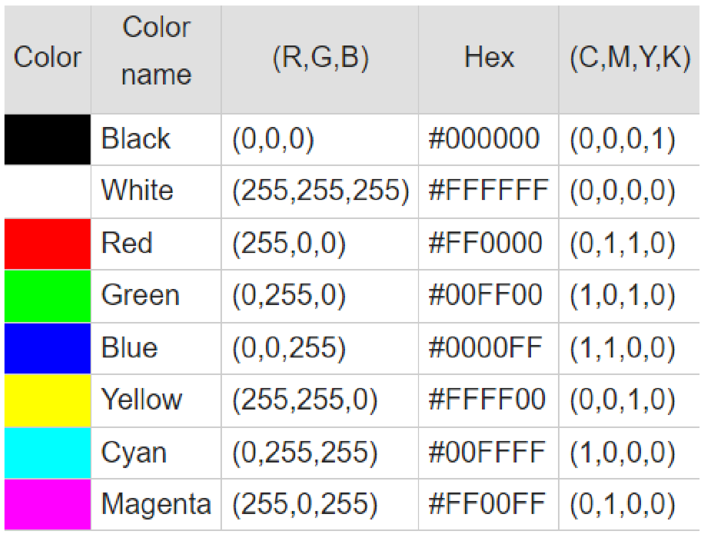
- Red =(255 * (100 – Cyan)) / 100
- Red =(255 * (100 –Magenta)) / 100
- Blue =(255 * (100 –Yellow)) / 100

This equation makes it even more obvious why an extra step was needed when the first conversion from RGB to CMY was made, because if it was missing, the original RGB values that were initially put in the RGBtoCMY function, would be returned here, mostly without any change. A small difference: +-1, would occur with some colours due to the double data type and qRound() being used, however this would in no way visually affect the colours of the uploaded image.

## 5.2.2 RGB to CMYK to RGB

The CMYK colour model is not that different from the CMY one, the main difference being that CMYK, implements black as an additional primary colour. Although Cyan, Magenta and Yellow might be sufficient for printing, most printers use black, as well. This is due to the fact that a mixture of the three colours might not create a dark enough black. Moreover, using an additional black, spares the more expensive inks and speeds up the printing process overall.

This model is subtractive as well, the colours get darker as you mix them together. This reflects what happens when an image is being printed out on paper, the more ink/pigment is added, the darker the colours get. Contrary to this, when using the RGB colour model, the colour gets brighter with the increase of the red, green and blue values.



| Color | Color name | (R,G,B) | Hex | (C,M,Y,K) |
|---|---|---|---|---|
| ⬛ | Black | (0,0,0) | #000000 | (0,0,0,1) |
| ⬜ | White | (255,255,255) | #FFFFFF | (0,0,0,0) |
| 🟥 | Red | (255,0,0) | #FF0000 | (0,1,1,0) |
| 🟩 | Green | (0,255,0) | #00FF00 | (1,0,1,0) |
| 🟦 | Blue | (0,0,255) | #0000FF | (1,1,0,0) |
| 🟨 | Yellow | (255,255,0) | #FFFF00 | (0,0,1,0) |
| 🟦 | Cyan | (0,255,255) | #00FFFF | (1,0,0,0) |
| 🟪 | Magenta | (255,0,255) | #FF00FF | (0,1,0,0) |

Fig5.2.2 RGB to CMYK table

Similar to the CMY-RGB-CMY conversions, both methods RGBtoCMYK and CMYKtoRGB accept colour model objects, RGB and CMYK, respectively, and return the newly created ones, CMYK and RGB.

To execute the RGB to CMYK conversion, all colours – red, green and blue are divided by 255, as the CMYK values have a 0-1 range. After that, the 1 minus the max value of the three is assigned as the key/blacK value. This is followed by a condition which, checks whether the black value isn’t equal to one. In the event it is, all of the other colours are assigned 0 as a value. If it is not, however, the following formula is implemented:

- Cyan = (((1 – Red/255) - Black)/(1 - Black)) * 100;
- Magenta = (((1 – Green/255) - Black)/(1 - Black)) * 100;
- Yellow = (((1 – Blue / 255) - Black)/(1 - Black)) * 100;

Once again, the colours are normalized by multiplying them with a hundred and using qRound. After that, the same formula, which was present in CMY is used:

- q = (q / 100) * q

The reason for this is the same, so that a visual change can be noticed in the image when a reverse conversion is made. The CMYK to RGB process alters the colours using the following equations:

- Red = 255 * (1 - Cyan) * (1 - Black)
- Green = 255 * (1 - Magenta) * (1 - Black)
- Blue = 255 * (1 - Yellow) * (1 - Black)

With these calculations, the range of the colour values is restored to 0 – 255, and the corresponding RGB model is returned.

### 5.2.3 RGB to HSI to RGB

In the HSI model all of the colours are represented using three components: hue, saturation and intensity. Furthermore all three components have a different range of values.

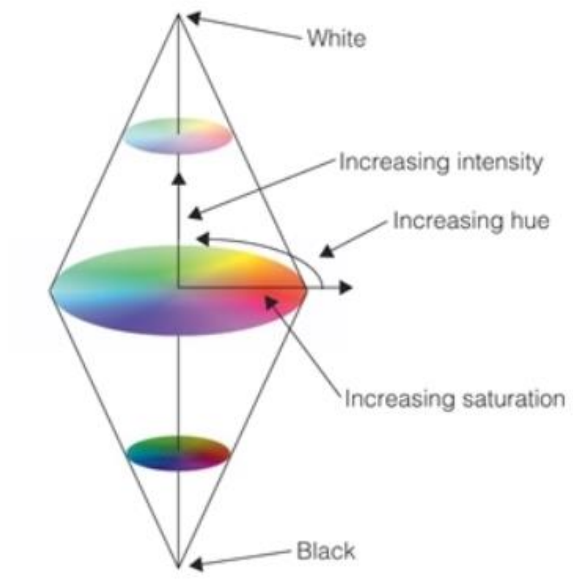


Fig5.2.3 HIS components visualized

Hue covers a 0 – 360 scope, being the biggest one of the three. This is due to the fact that this component describes the colour itself in the form of an angle. Zero degree equals red, while 120 means green and 240 is blue. Moreover, 60 degrees is yellow, whereas 300 degrees equals magenta.

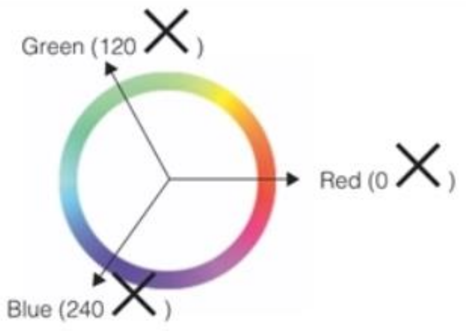


Fig5.2.3a Conversion from hue to numerical value is done with a colour wheel

Saturation evaluates how much a colour is diluted with white light. Different formulas offer different scopes for this value – either [0-1] or [0-255]. Nevertheless, the range, which is introduced in the code is from 0 to 1, where 0 represents grey. However, in the RGBtoHSI function, the end result for saturation is multiplied by 100, so that the values can be better seen on the histogram.

Less
saturated
More
saturated

Fig5.2.3b Range of saturation values for a single hue

Lastly, the third element – intensity can be referred to as the brightness of a colour. It is assigned values between 0 and 255, where 0 corresponds to black, while 255 is white. The calculations for the RGB to HSI conversion start with the intensity evaluation:

- Intensity = 1/3 * (Red + Green + Blue)

After that, the saturation can be summed up, using the following formula:

- Saturation = 1 – 3 * (MIN(Red, Green, Blue) / (Red + Green + Blue))

Following this, the hue can be calculated with the equation:

$$h = \cos^{-1}\left\{\frac{0.5\cdot\left[(r-g)+(r-b)\right]}{\left[(r-g)^2+(r-b)(g-b)\right]^{1/2}}\right\} \qquad h\in[0,\pi] \text{ for } b\le g$$

$$h = 2\pi-\cos^{-1}\left\{\frac{0.5\cdot\left[(r-g)+(r-b)\right]}{\left[(r-g)^2+(r-b)(g-b)\right]^{1/2}}\right\} \qquad h\in[\pi,2\pi] \text{ for } b> g$$

Fig5.2.3c Hue formula

For the cos and $cos^{-1}$ – qCos and qAcos from the QTMath library are used. For convenience, the components are then converted into their respective ranges. Hue is multiplied by $180/\pi$, the saturation is multiplied by 100 and the intensity by 255. All of these values are then put towards the histogram.

For the HSItoRGB, there is function, which follows a set of formulas in order to convert all of the HSI components to RGB colours.

$$h = H \cdot \pi / 180 ; \qquad s = S/100 ; \qquad i = I/255$$

$$x = i \cdot (1 - s)$$

$$y = i \cdot \left[ 1 + \frac{s \cdot \cos(h)}{\cos(\pi/3 - h)} \right]$$

$$z = 3i - (x + y);$$

when $h < 2\pi/3$, $b = x$; $r = y$ and $g = z$.

when $2\pi/3 \leq h < 4\pi/3$, $h = h - 2\pi/3$, and $r = x$; $g = y$ and $b = z$.

when $4\pi/3 \leq h < 2\pi$, $h = h - 4\pi/3$, and $g = x$; $b = y$ and $r = z$.

Fig5.2.3d HIS to RGB

Unfortunately, this formula did not provide a satisfactory result and the image colours were not correctly changed. However, another way was found. Red was directly equated to hue. Since hue has a max of 360, the value was firstly divided by itself, to get into the 0-1 range and then multiplied by 255, which would give a new scope of 0-255, corresponding to that of the red colour. The same was done with the saturation. The intensity was left as it is, due to the fact its' primary range equates to the needed one.

## 5.3 Histogram widget

This class inherits QWidget and its' main purpose is to visualize the different histograms for the different colour models. For the realization of this task, QT charts was heavily utilized. In order to have access to the class, a declaration of: QT + = charts is made in the .pro file. The layout and gui are created dynamically, so a .ui form is non – existent. The histogramInit() function is the place, where all of the needed elements are initialized.

The first objects to be set up are four QBarSets. Each set corresponds to a separate category, in this case to a separate colour. Even though, not all models have four shades, four bars are declared, because of CMYK, which returns four values and all of them have to be presented in the diagram.

After that a QStackedBarSeries object is allocated, to which all of the previously created bars, are added. The goal of this entity is to present information as vertically stacked bars, with one bar per colour (category). Several QStrings, representing the names of the colours (categories) and several QColors, which have the role of filling the bars on the chart, are all set to default values.

The QChart is created next. Using the functions that the class provides, a title is set: QChart.setTitle(QString title), together with a way of displaying the data: histogramChart->setAnimationOptions(QChart::SeriesAnimations), which allows the bars to be gradually and smoothly shown on the screen, rather than just suddenly appear. The function addSeries is used to add the previously declared QStackedBarSeries to the chart. For QChart to be then displayed, a special widget is needed – QChartView. setChart(QChart ch) sets the current chart to ch. The chart view is then added to the layout of the histogram widget.

After finishing with the gui set up, four functions – set*ColourModel*: setRGB, setCMY, setCMYK and setHSI follow. All of the functions adhere to the same algorithm of employing the colour conversion formulas for the respective colour model in order to draw out all of the required values for the construction of the histogram. All of the methods have a QImage passed by reference. This image is used as the base for every color conversion and all of the required values are extracted from it. All, except RGB, which is of type void, return the newly constructed QImage.

Before the colour conversion is started several things have to be done. The first one being, to call the method – clearColourSets(), the aim of which is to remove all of the values in the sets before getting the new values. The remove range that has been set to all four set.clear()-s is 0 – 360, the maximum equal to hue in HSI as it has the biggest range out of all the colours.

In the header four QVectors of type double are declared. They will hold the values for the histogram. To make sure there are no inconsistencies with the visualization, all four are resized: for RGB – 256, for CMY/CMYK – 101, for HSI – 361, and filled with zeros at the beginning of each function.

Afterwards, a nested for-cycle follows, which is responsible for iterating through all of the pixels of the image, taking the values and passing them to the colour conversion class. The outer for loop has an end value of image.height(), in other words it takes care of rows, while the inner one is responsible for the width() – columns. In the first loop, just before the width one is called, the following code is called:

- QRgb *row = <reinterpret_cast<QRgb*>>(loadedImage.scanline(i))

The scanline function is available in the QImage class, and its objective is to return a pointer to the pixel data at the line with index i. To make the values useful to the code a reinterpret_cast is used, which by simple definition converts a pointer of some data into a pointer of another data, even if the datatypes before and after conversion are different. The needed type for this function is QRgb, which returns an ARGB quadruplet (255, r, g, b). After that, in the inner for-cycle, a QRgb object is constructed using the scanline and the current column, from which the red, green and blue values are extracted with qRed(QRgb _rgb), qGreen(QRgb _rgb) and qBlue(QRgb _rgb), all of which return the red, green or blue component, respectively, from an ARGB quadruplet – _rgb. Having acquired the needed values, the colour conversion – for CMY, CMYK and HSI, for the histogram values is done. In the header there five pointers to object declared, one for each colour model and one for the colour conversion class. For example for cmy the following line is called:

- Cmy_object = new Cmy_object(colorConversion_object(new Rgb_object(red, green, blue)));

Similar functions are called for CMYK and HSI, where CMY is replaced by the respective colour model. This function, utilizes the colour conversion functions and returns the CMY/CMYK/HSI values, corresponding to the RGB values that have been directly extracted from the loaded image. The converted values are then returned, using the getter functions and added to the vectors.

For CMY and CMYK a reverse conversion is made: from CMY/CMYK to RGB, following the same model as the one above. After the new three - red, green and blue values are received, they are passed to a new QRrb, which is equated to the current one. This takes care visual representation of the model on the picture. For HSI, no conversion is made. The hue, saturation and intensity are directly used as RGB values, however due to the fact that the hue has a far wider range than 255, and the saturation has a max of 100, a normalization takes place:

- Red = (Hue / 360) * 255
- Green = (Saturation / 100) * 255
- Blue = Intensity

After the pixel colours have been changed as well deallocation of the memory is executed at the end of the inner for – cycle, the allocated models (using the “new” keyword) are deleted. This makes the program faster as no excess memory is taken up and everything is freed up before the closing of the program.

When the cycles have finished their iterations and all of the data is acquired, the preparation for the QChart set up is started. The four QString-s declared in the Init function are set, to the names of the colours of the current model, for RGB/CMY and HSI, the fourth QString is set to an empty one. The four QColours are being set to represent the colours of the model, as well, with the fourth one being assigned QT::transparent for all models except CMYK, where it is equated to QT::Black. All of these parameters are used in the setColourSets() function, where the now initialized colour names become labels to the QBarSets and the Qcolours, the colours of the bars.

Following this a function either setChart or setChartCMYK is called, the only difference being that the CMYK one iterates through all four vectors, while the other one goes only through the first three. Since the values in the vectors are too large, using std::max_element, the biggest number is returned from each vector, and before adding a number from the vector to a set for the chart, it is first divided by the max and then multiplied by 100, this normalizes the number and displays the histogram using percentages, which promotes a cleaner look and an easier understanding of the data.

The last function, which is called form within the setChart/setChartCMYK is the setHistogramChartXYAxis, which is passed an axisXMaxNumber and an axisYMaxNumber. Since the representation uses percentages, the y axis is always the same and the max of it is 100, however, as mentioned several times, the different colour models have different value ranges and the x axis is changing with every new model, so for it, the length of the first vector is passed, as vectors are resized according to the model with every call to a set model function. Both of the functions used for the axes set up – setRange and setTitleText are a part of QChart class.

## 5.4 Mainwindow

Last is the main window construction. Similar to the Hsitogram Widget, the gui is made dynamically, rather than using a .ui form, although one here is indeed present. All of the widgets are declared and initialized in the initWidget function. Since this is a mainwindow class a central widget has to be set in order for all of the elements to be visualized. This is achieved by creating an empty QWidget, to which all of the other components are assigned. After everything is initialized, the QWidget in question is set with the function setCentralWidget(QWidget w), to the current object.

In the initWidget function, everything is sorted into Layouts smaller layouts, which are then added to the main one. All of the QLineEdit, which are expected to receive red, green or blue values are assigned QIntValidators, so that numbers less than 0 and higher than 255, are not accepted:

- QLineEdit edit->setValidator(new QIntValidator(0, 255, this));

Furthermore, on all of the line edits a placeholder text is set, which will give information about the text box, when the user hovers his/her mouse above it.

In this class there are several functions, taking care of the image modification: changed red/green/blue channel and pixel swap, all of which are hidden from the user, so they are declared with the method widget->setVisible(false), so that the layout does not become overcrowded. These elements can be revealed with the different buttons added to the gui. In the event that the user wants to hide them again, rather than adding an additional button a series of QShortcuts have been added, which allow the user to execute functions with a dedicated key combination.

- QShortcut(QKeySequence(Qt::Key_A + Qt::Key_B), this, SLOT(function()));

In the main window there is also a function called ConnectSlots(), the sole aim of which is to connect all of the needed signals that the elements emit to a corresponding function. Most of the signals that are being initialized are those of buttons – clicked(), and there is one for the QComboBox, which informs us, whenever a different colour model is picked – currentIndexChanged(int):

- QObject::connect(sender, SIGNAL(signal), receiver, SLOT(method))

On_Open_Clicked is a slot, activated when the open button is clicked. It opens a QFileDialog, which lets the user choose an image to load from the local files on his/her computer. After a picture is chosen, the path to it is set in the QLineEdit next to the button, while the image is converted into an RGB32 format, in order to make everything standard. If the image is not empty, then a copy of it is made in a folder tempFiles – with the saveTempImage() method. This is done, so that the changes made by the user, to colour channels, are persistent through the life of the application and affect the histograms of the different colour models. The image is then displayed and by default its RGB histogram is shown, unless the model was changed beforehand in the combo box: colourModelComboBox_CurrentIndexChanged(colourModelComboBox->currentIndex());

When another model is chosen from the drop down menu on the right - colourModelComboBox_CurrentIndexChanged is executed. This method takes the image that has been previously saved to the tempFiles folder, including any changes to the colour channels, if such have occurred, and depending on the chosen colour model, a corresponding function from the histogram widget is called, which sets and visualizes the histogram and returns a new image, which displays the changes in the colours, that appear with the separate models.

The on_Save_clicked function is another slot, invoke by a clicked signal. Since the image is originally loaded into a QLabel, so that it can visualized, this function takes the label in question, converts it into a QPixmap and then to QImage. This procedure allows the user to save the image that is currently being displayed in a program to a folder of his/her choice.

In the event that the user wants to change the colour channel, a button on the left has to be clicked, which will activate the function - on_ShowRedChannelChangeBox_clicked, for example. This will reveal a text box with a button next to it. As mentioned above the line edit can only hold values between 0 and 255, and anything outside of this range will be equated either to the min or the max. For a change to take place, the button "Change Red Channel" has to be clicked. Following, the on_ChangeRedChannelButton_clicked() method will be called.

The algorithm that is used for the colour change is the same one used in the Histogram widget. A nested loop with a scanline in it is constructed, after which the original QRgb is equated to a new one, with the green and blue values kept the same, and the red being replaced with the value, that was put in the text box by the user. The modifications are then displayed to the QLabel that holds the image, furthermore, the copy that was saved when the image was open is updated and the current histogram is reloaded to reflect the transformation. Corresponding functions are made for the green and blue channels, as well.

Another functionality that the user can deploy is the pixel colour swap. Again hidden, the elements for it can be revealed by clicking on the button on the left, calling the on_ShowSwapPixelColourBox_clicked method. This will reveal six text boxes and a button, prompting the user to input a red, green and blue values, to be found and another three, which they will be swapped with. Rather than working on the temp file here, the currently displayed images is taken, as, firstly, it is unknown which colour model is chosen and secondly, due to the fact there might be a difference between the values of the pixels in the temp file and the version that is shown.

To achieve the swap two QColors are constructed in a nested loop. The first one has the values of the current pixel and the second one holds the values that the user is searching for. In order to get the elements for the current colour – the already mentioned and described scanline method is used. In the event that the shade that is being looked for is equal to the current one, the colour of the present pixel is overwritten with the new values the user has chosen. These changes are temporary and are not saved, to the temp image.

On the left side there are two more buttons – revert image and information. By clicking on the revert image button, on_RevertImageButton_clicked() is invoked. This method calls saveTempImage(), which overwrites the already saved one - temp, with the original one, by taking its' original path from the QLineEdit at the top, after which colourModelComboBox_CurrentIndexChanged(int ind) is called, as well, so that the "new" original picture can be loaded and the histogram adjusted to it. This process deletes all of the previously made changes.

The only thing the information button does is execute on_Information_clicked(), which opens a QMessageBox, containing all of the key combinations that can be used in the application and their purpose.

The last function is deleteTempImage(). This function goes into the tempFiles folder and deletes the image that was created at the beginning of the program. To make sure the image is not deleted beforehand, this method is called in the destuctor of mainwindow, which is invoked upon closing the application.