Sofware Engineering

Project

2014/2015

Students: Bangău Alexandru, Boţa Alina,

Buleandră Cristian,

Constantinescu Adrian,

Dan Ana-Veronica,

Georgescu Andrei

Project Title: Image Analyzer and Converter

Supervisor: Prof. dr. ing Brezovan Marius

1 PROBLEM DESCRIPTION

The Image Analyzer and Converter project analyzes images and then converts them by applying basic image editing functions on them.

- User may open any image from the file system as needed.
- It can detect and display various image file formats including JPG, BMP, PNG, TIFF and more.
- It detects and displays all these image types and even converts them to jpg and YUV format.
- The system can also filter images based on various filters.
- Contrast and brightness functions are also available here.
- To display an image with respect to its RGB components, a histogram utility is provided.
- The original and filtered version of an image can also be viewed in parallel for comparison.
- User may get filtered and converted images as desired using this image converter project.

2 SUMMARY

The application's main objective is to correctly display an image saved in multiple file formats and offer the user the possibility to edit, adjust and save it.

3 CONTENTS

4 CHAPTER 1: INTRODUCTION

This document is intended to inform the user about the application's functionality as well as the user and system requirements and the architecture of the project.

5 CHAPTER 2: REQUIREMENT ANALYSIS

The application's requirements are divided as follows:

5.1 USER REQUIREMENTS

The user shall be able to:

- Open and save any image in various format files including JPG, BMP, PNG, TIFF and more;
- Preview and adjust the image's brightness and contrast;
- Apply greyscale and sepia filters;
- View the image's RGB histogram;
- View the original image next to the edited one;
- Rotate the image clockwise and counter clockwise;
- Flip the image vertically and horizontally.

5.2 SYSTEM REQUIREMENTS

The system requirements define what will be implemented in the project. They are composed of:

5.2.1 Functional Requirements

5.2.1.1 Description

The functional requirements describe the functionality or the services provided by the system:

1. The system will have an intuitive Graphical User Interface that will guide the user through the options without any effort.
2. The GUI will have a menu that will provide a front end to all the implemented functions.
3. The Application will provide the possibility to choose a file from the personal computer and open it.
4. There will be implemented conversion methods and added filters to the file chooser in order to save an image in different file formats.
5. The application will have filter functions for greyscale and sepia, possibility to adjust brightness and contrast and functions to blur and sharpen the image.

6. The back end will also provide a family of functions that will rotate and flip the current image.
7. The application will contain a way to save the original image and display it as well as go back one step (Undo).
8. The application will compute and display the RGB histogram of the selected image.
9. The application will have a dialog based information system for particular cases such as: The user wants to edit, but no image was opened.

5.2.1.2 Constraints:

a. Data constraints:

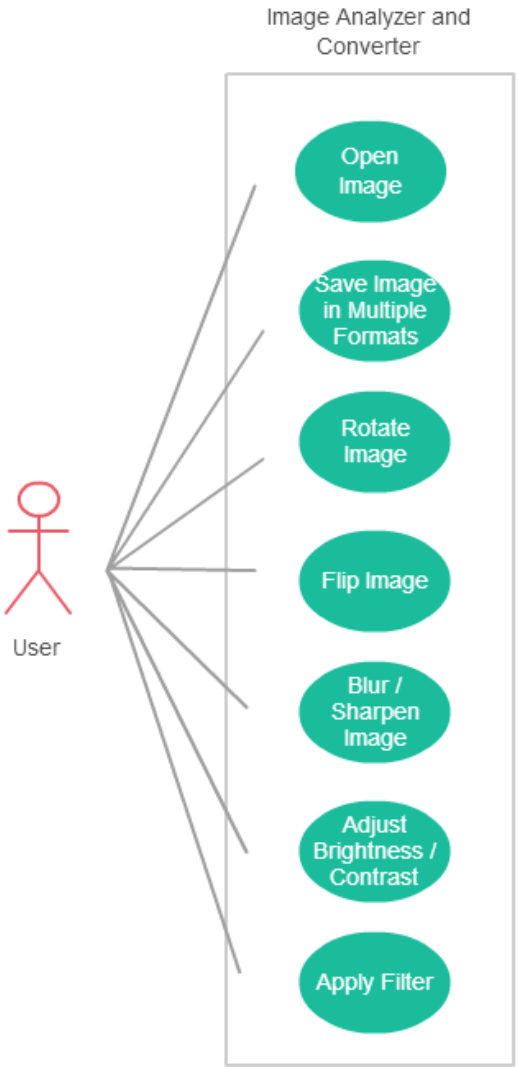
The file types that the application will work with are: JPG, BMP, PNG, TIFF, YUV.

b. Functions and processes constraints:

The implementation will be done in an object oriented language (Java) and the development technique is based on protyping.

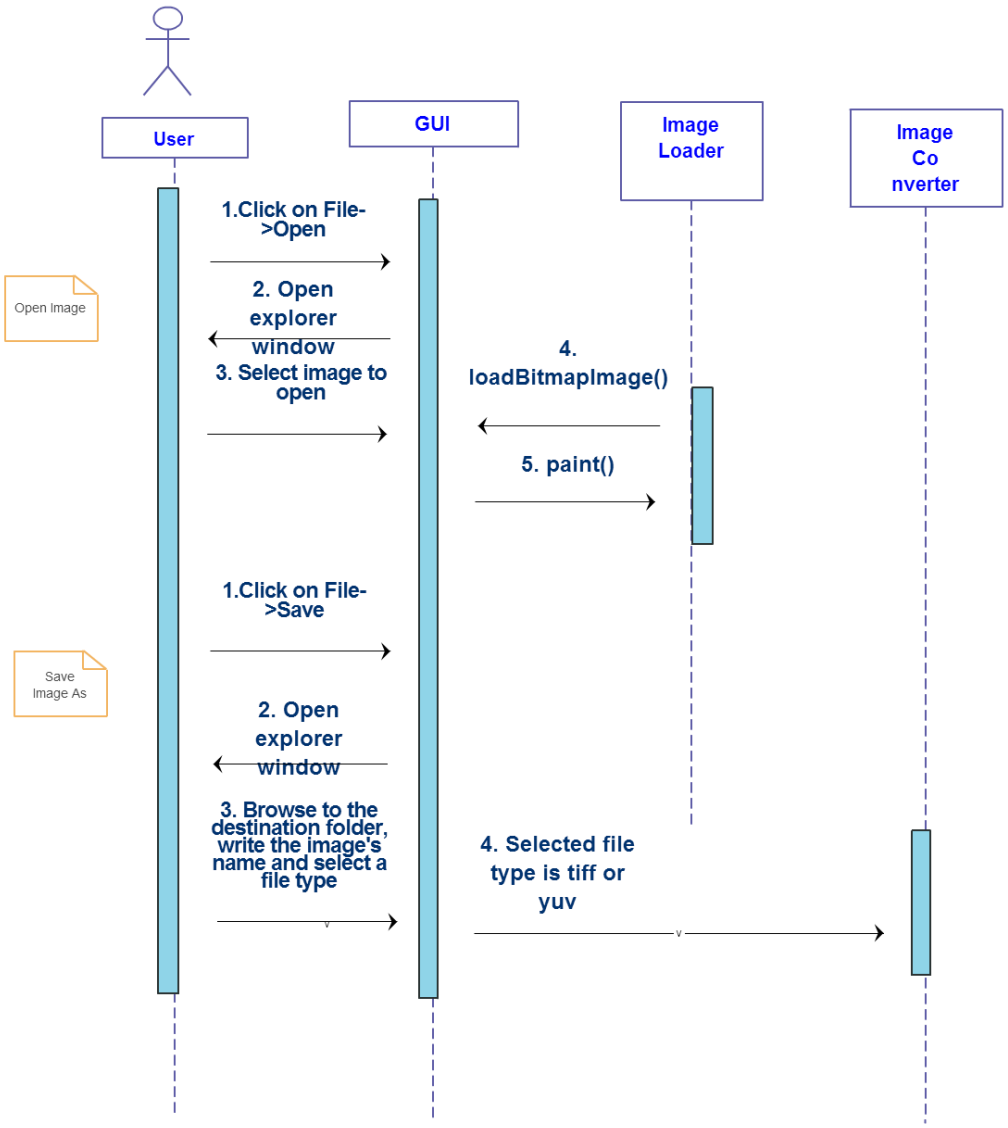
The requirements are better illustrated using the following UML diagrams:

5.2.1.3 Use Case Diagram

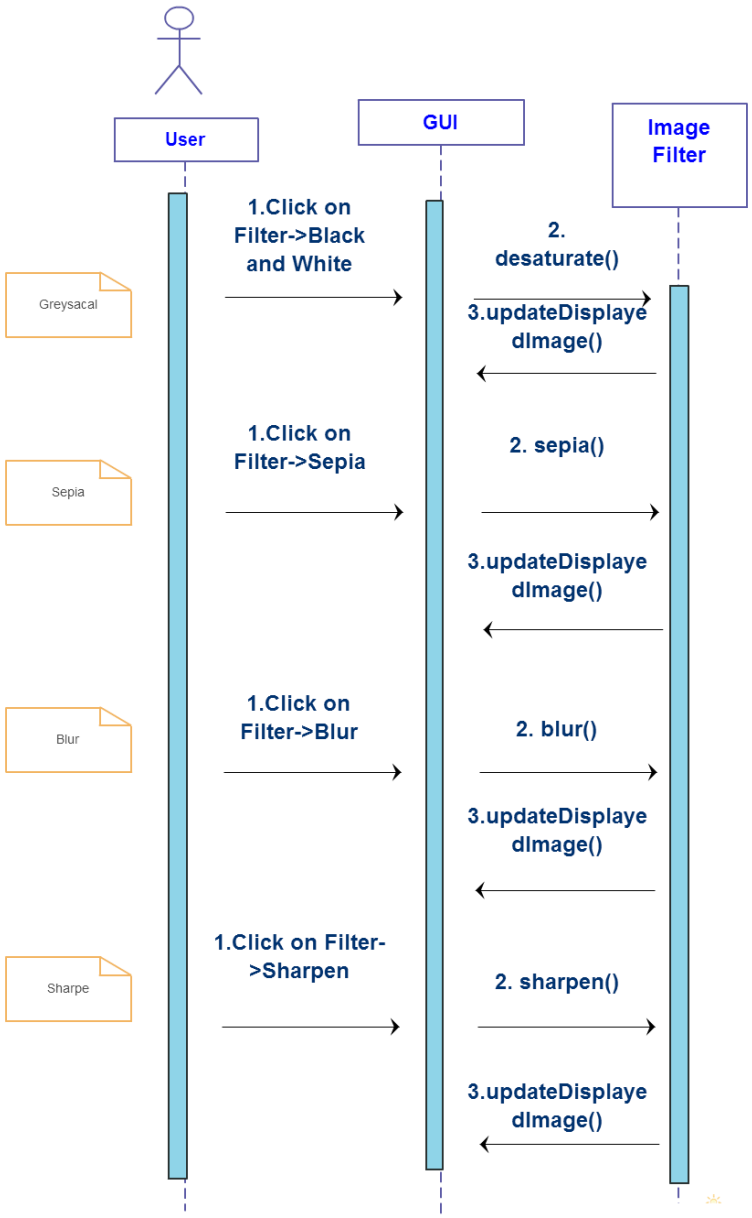


5.2.1.4 Sequence Diagrams

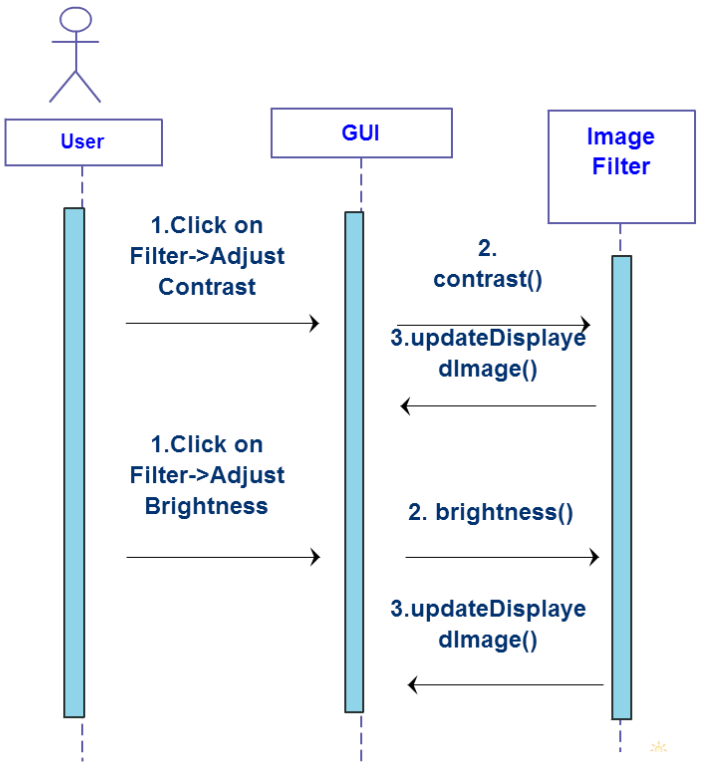
1. Open and Save functions



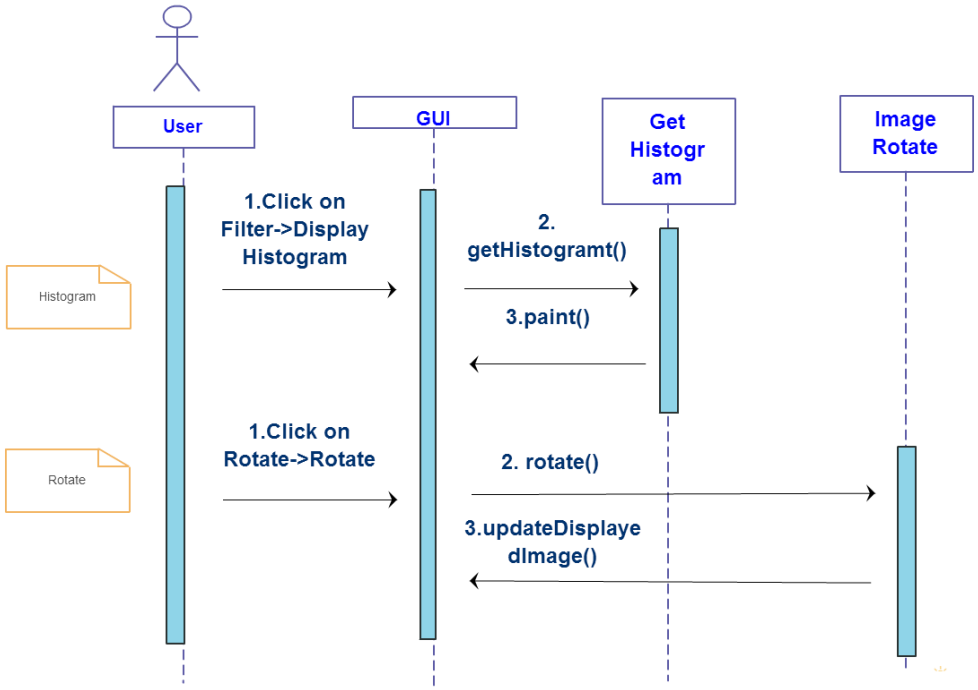
2. Filters


User
GUI
Image Filter
1.Click on Filter->Black and White
2. desaturate()
3.updateDisplayedImage()
Greysacal
1.Click on Filter->Sepia
2. sepia()
3.updateDisplayedImage()
Sepia
1.Click on Filter->Blur
2. blur()
3.updateDisplayedImage()
Blur
1.Click on Filter->Sharpen
2. sharpen()
3.updateDisplayedImage()
Sharpe

3. Adjustments



4. RGB Histogram and Rotate



5.2.2 Non – functional Requirements

5.2.2.1 Description

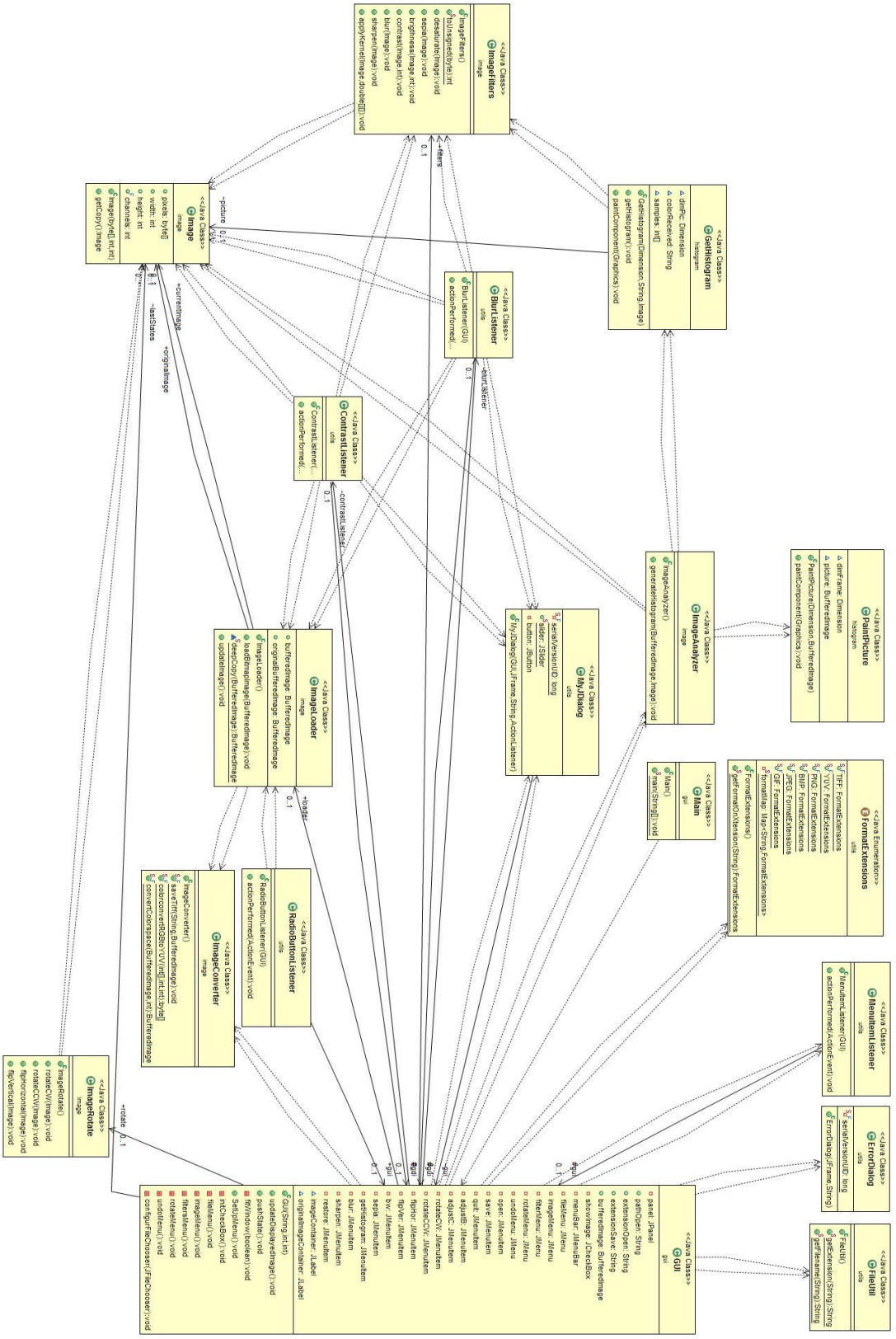
The non - functional requirements are grouped as follows:

1. Product Requirements
   The graphical user interface should be intuitive and implemented with Java Swing.
2. Organization Requirements
   The project's development is composed of 4 phases:
   - Inception
     Deadline: 10.04.2015
     Contents: initial requirements, initial architecture, initial work items
   - First Prototype
     Deadline: 8.05.2015
     Contents: release plan, prototype
   - Second Prototype
     Deadline: 19.05.2015
     Contents: release plan, prototype
   - Final Delivery
     Deadline: 8.06.2015
     Contents: user documentation, final binary
3. External Requirements
   The images uploaded by users should not be saved anywhere else except where the user wants them to and should not be distributed or used by anyone except the user.

6 CHAPTER 3: DESIGN AND ARCHITECTURE

6.1 STRUCTURE

The application's structure and modules are described using the following class diagram:



7 CHAPTER 4: IMPLEMENTATION

7.1 MODULES' DESCRIPTION

1. **GUI**
   The GUI is created in Java Swing and is composed of: a JFrame that contains a JMenu and each menu element has an associated listener and icon. The most complex listeners are the open and save ones that involve a FileChooser. For the most important commands, key strokes were assigned.

2. **Filters**

Filters were manually implemented to work directly on a pixels array (not on a BufferedImage), this way the performance of the filters is greatly improved.

We have created a custom **Image** class that only contains a pixels byte array and the width and height of the image.

For simplified filters implementation, every **Image** loaded is converted to the **RGB** format.

The filters can be split into three categories:

- **Byte specific filters** - which only change the value of one **byte** (one channel) at each iteration step:
  - Brightness
    - This filter increases the value of each **byte** in the pixels array with a given **intensity**. The brightness intensity can be between -255 and 255, but the user interface includes **-100%** > **100%** slider range that gets mapped to this interval.
  - Contrast
    - This filter makes light pixels lighter and dark pixels darker. This is done by iterating through each **byte** in the pixels array and if the value is larger than 127 it gets increased by the **contrast intensity**, otherwise it gets decreased. The contrast intensity can be between -100 and 100.

- **Pixel specific filters** - which only change, at each iteration step, the value of one **pixel** considering only the value of that pixel (three channels: R,G,B) :
  - Black & White (desaturate)
    - Each **pixel** gets its **RGB** channels changed to the same value, which is the average of those three channels: **R = G = B = avg(R,G,B)**. An improvement is that instead of directly using the avg we use an weighted average to compensate for the different intensities that the human eye sees the colors at. (eg: **Blue** channel is less visible to the human eye, so we use a smaller weight for it).
  - Sepia
    - Each **pixel** gets its **RGB** channels changed to distinct values, based on some weights for each channels, weights that mostly increase the **Green** and **Red** channels.

- **Kernel based filters** - filters that change the value of each **pixel** based on the pixel values around it and the **kernel (convolution matrix)** given. The convolution matrix contains the width and height of the neighbouring area to consider and also the weights for each neighbour. Any filter that uses a kernel can be implemented by simply calling the **applyKernel** function with a different kernel. The ones implemented in this project are:
    - Blur
        - The kernel of size **3x3** contains weights so that, when applied, it averages the pixel values around the center pixel, but keeping the center pixel the dominant one (largest weight).
    - Sharpen
        - The kernel of size **3x3** contains weights so that the value of the center pixel is increased more if the pixels' values around it are higher. So, by using this weights, it makes sure that each pixel intensity is increased in order to make it stand out more.

3. Rotations

The rotations are done using the original image's pixel array and rotating or flipping it accordingly. There are implemented only the CW rotations and horizontal flip, since the CCW rotation is done by rotating the image CW three times and vertical flip is done by two CW rotations and a horizontal flip.

4. Histogram

The histogram is generated using ImageFilters and sampling into a graph every color (red, green and blue) from the image.

5. Convertor

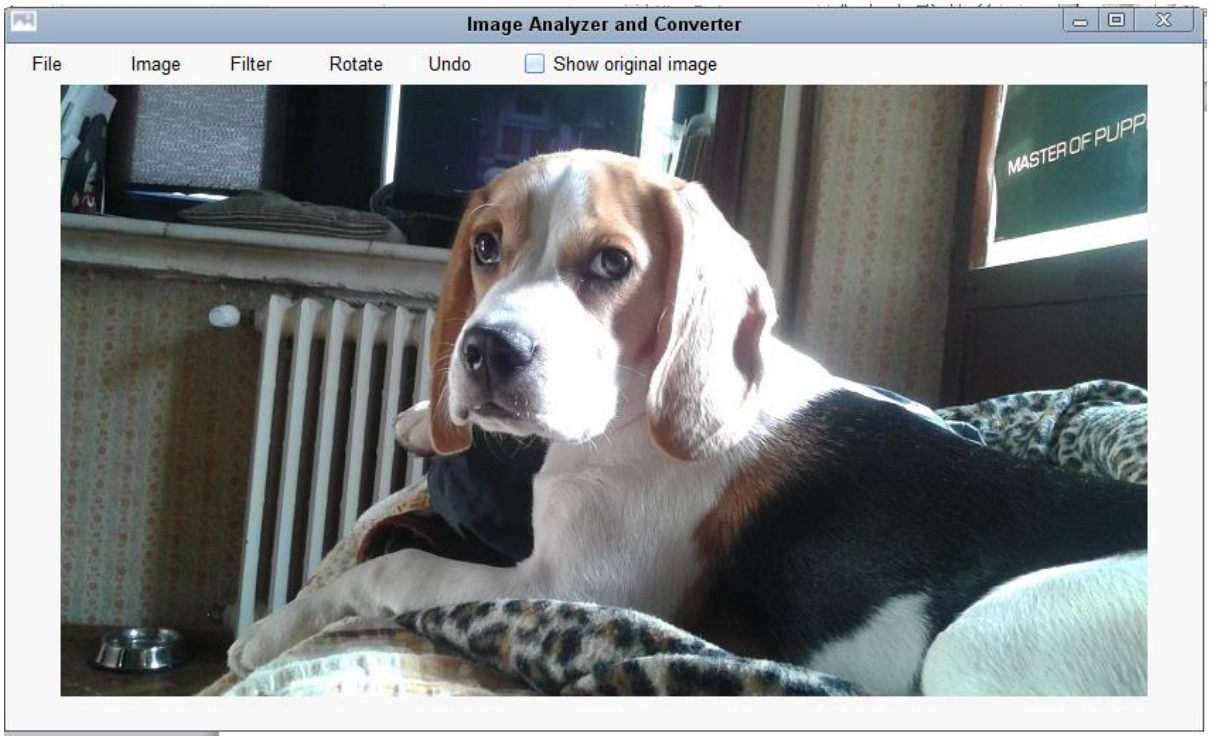
The conversions are done for TIFF and YUV file formats:

- TIFF – there is used TIFFImageWriteParam ;
- YUV – the file is converted using the following formulas for obtaining the necessary channels:

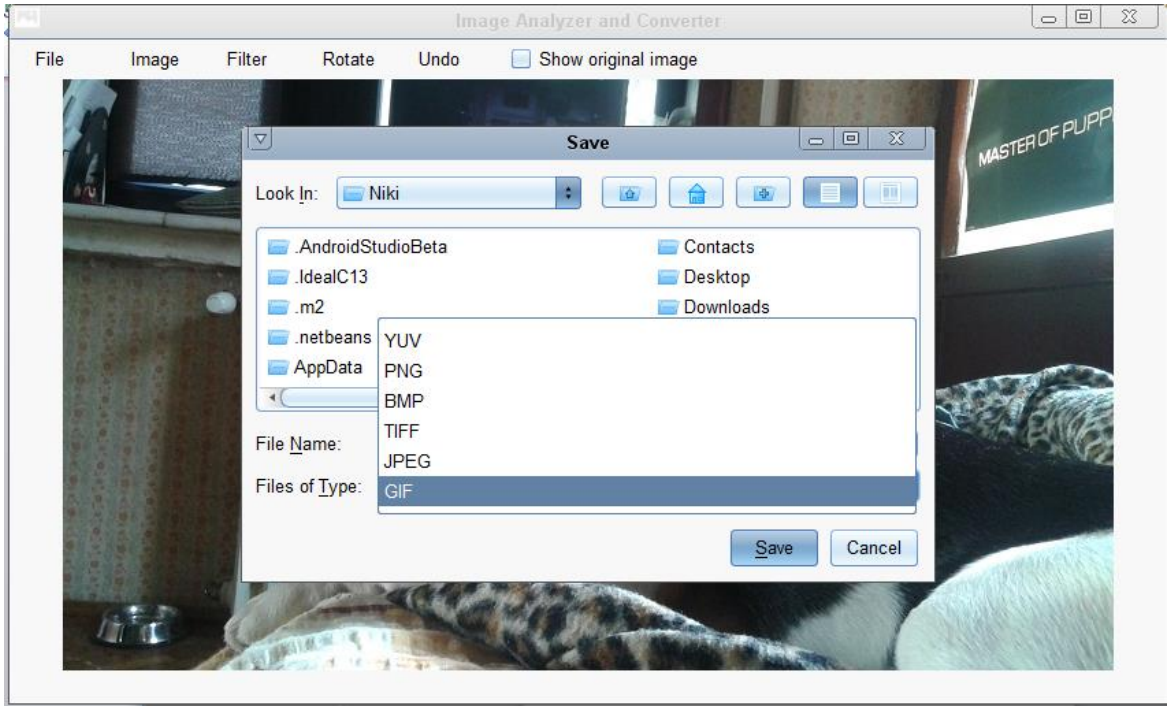
```
R = (aRGB[index] & 0xff0000) >> 16;
G = (aRGB[index] & 0xff00) >> 8;
B = (aRGB[index] & 0xff) >> 0;

Y = ((66 * R + 129 * G +  25 * B + 128) >> 8) +  16;
U = (( -38 * R -  74 * G + 112 * B + 128) >> 8) + 128;
V = (( 112 * R -  94 * G -  18 * B + 128) >> 8) + 128;
```

7.2 DETAILED RUNNING EXAMPLE

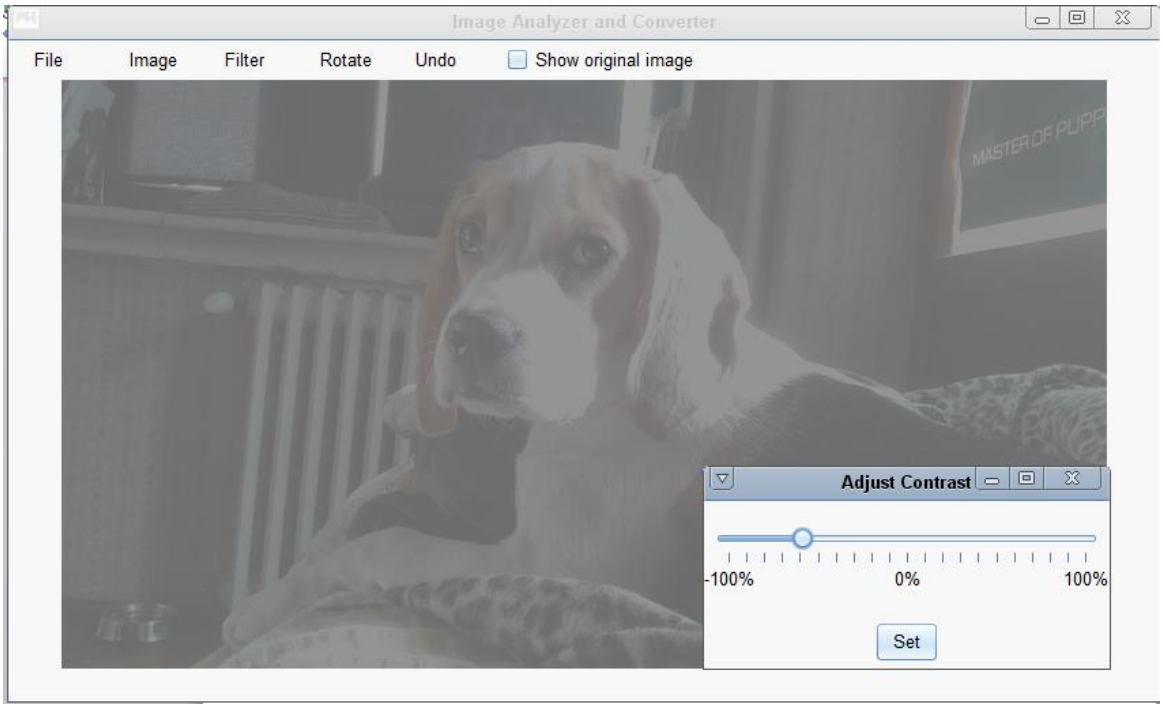
Open image



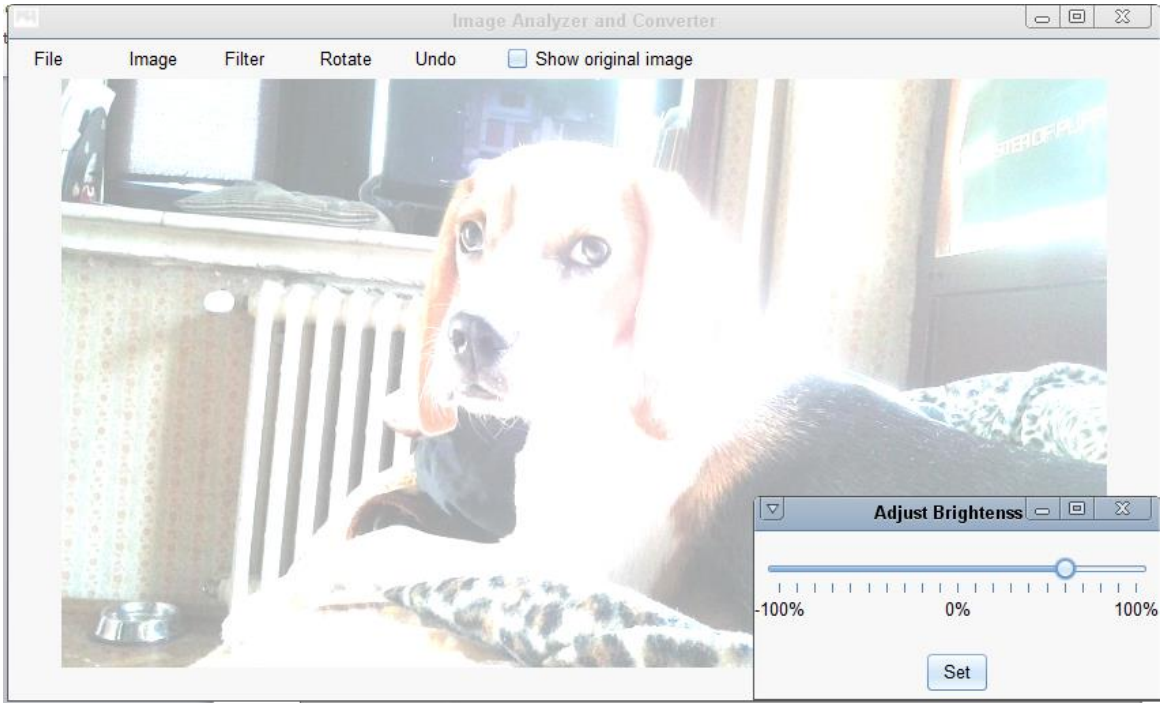
Save Image



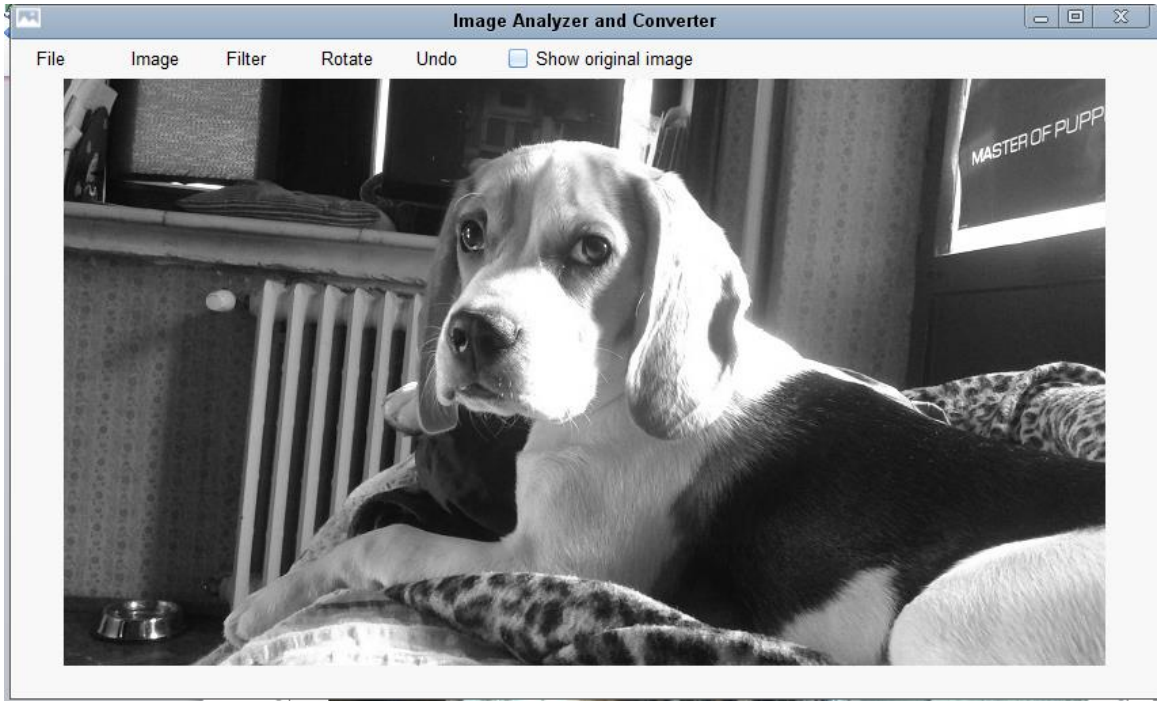
Adjust Contrast



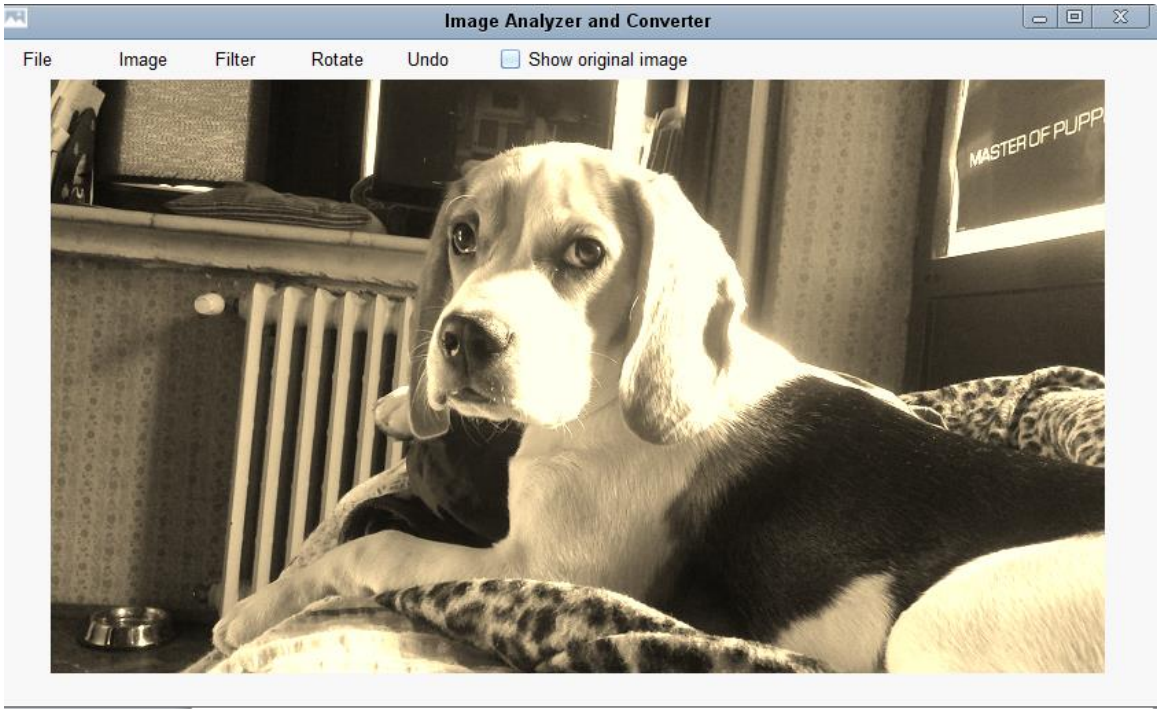
Adjust Brightness



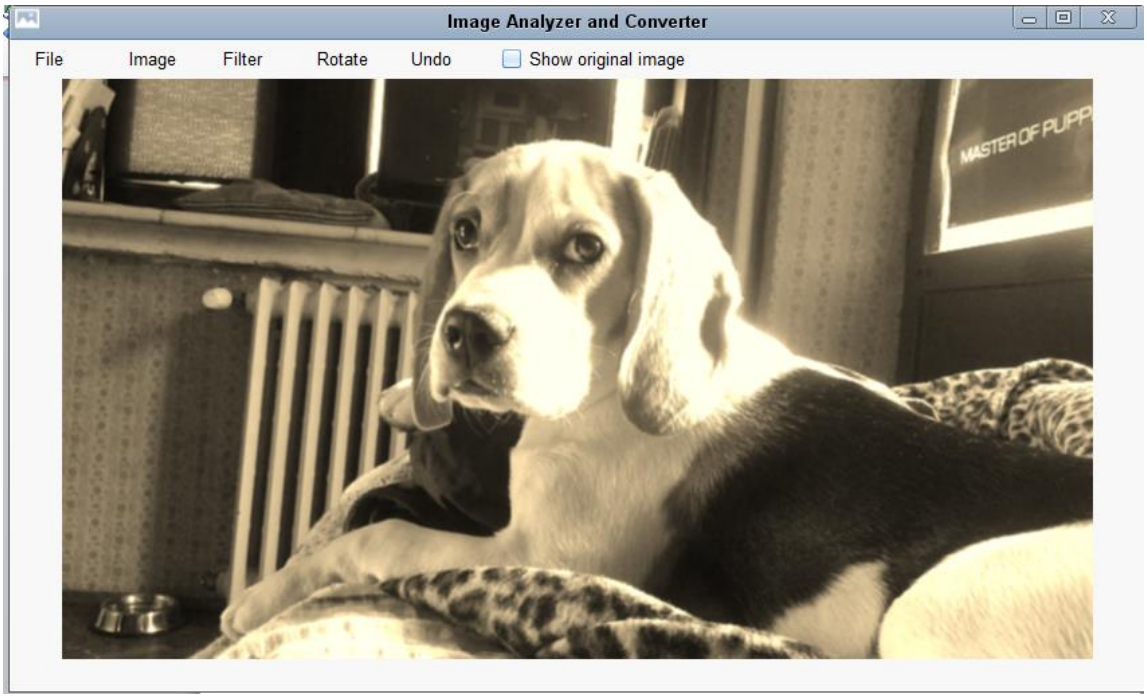
Greyscale



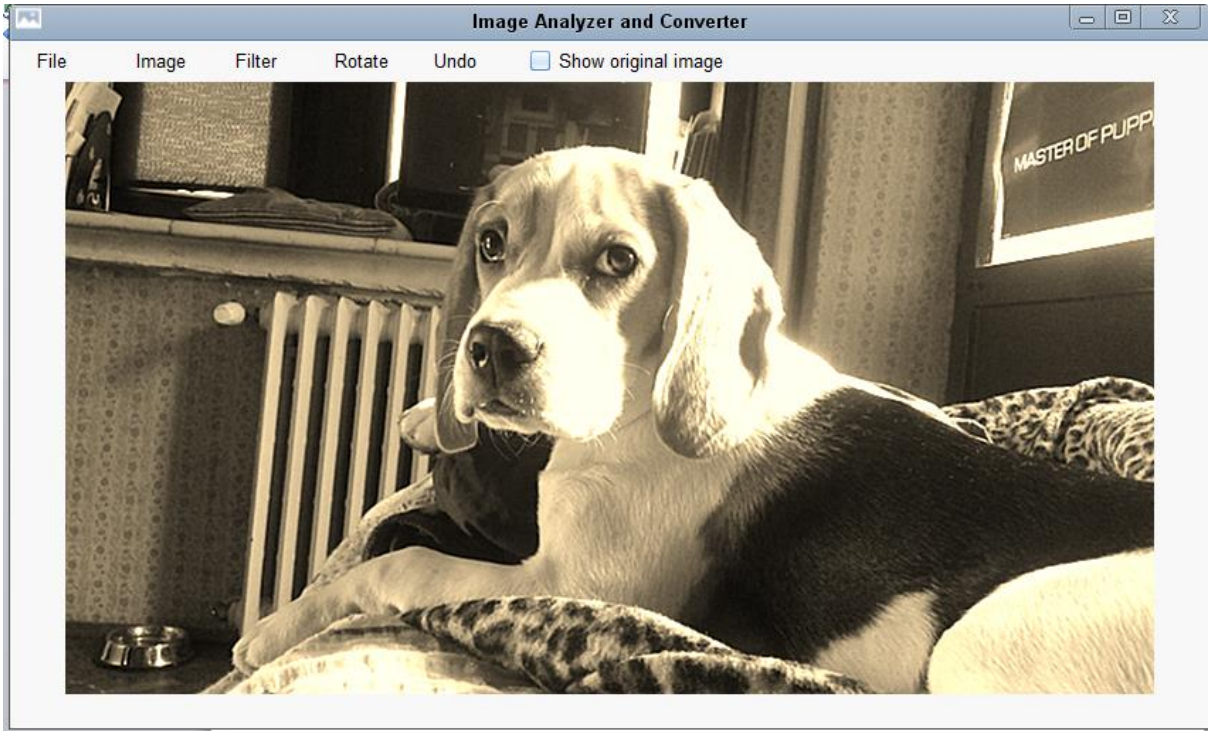
Sepia



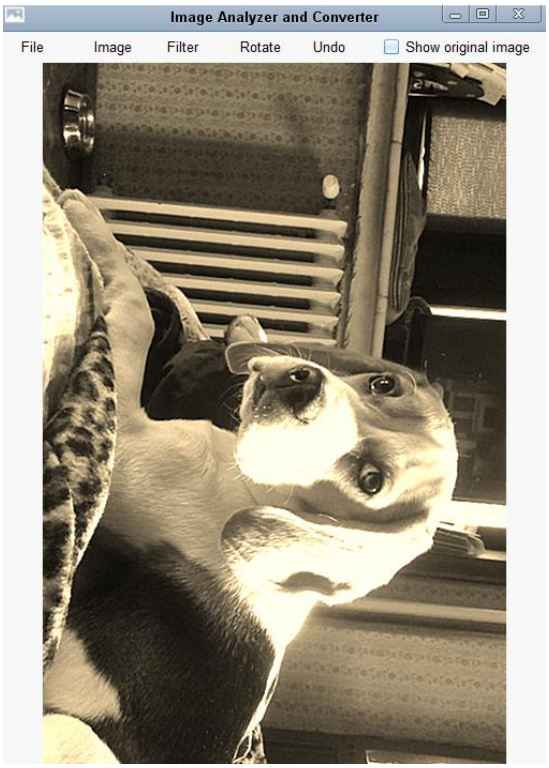
Blur



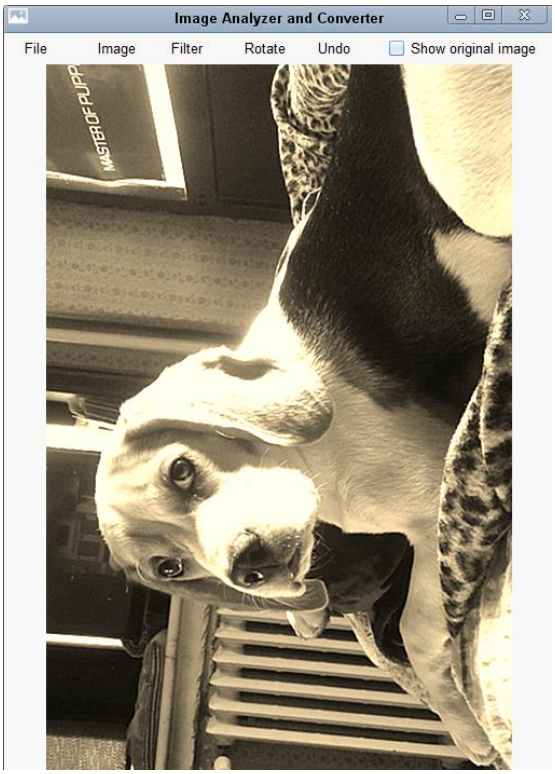
Sharpen



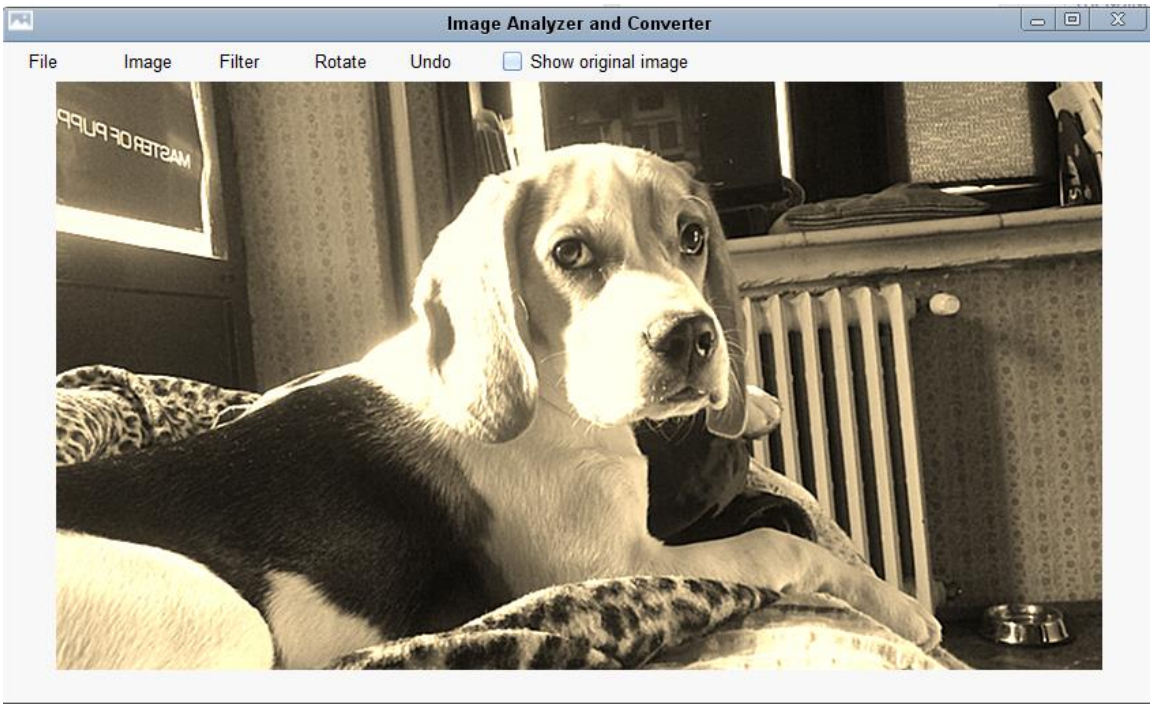
Rotate Clockwise



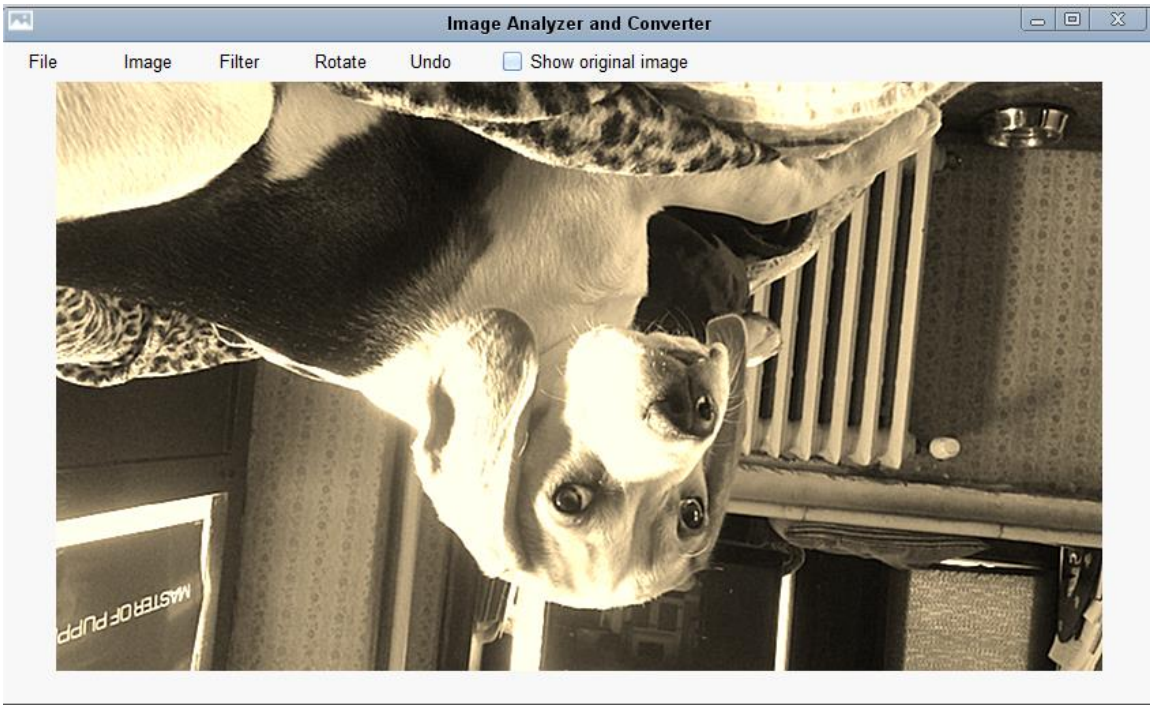
Rotate Counter Clockwise



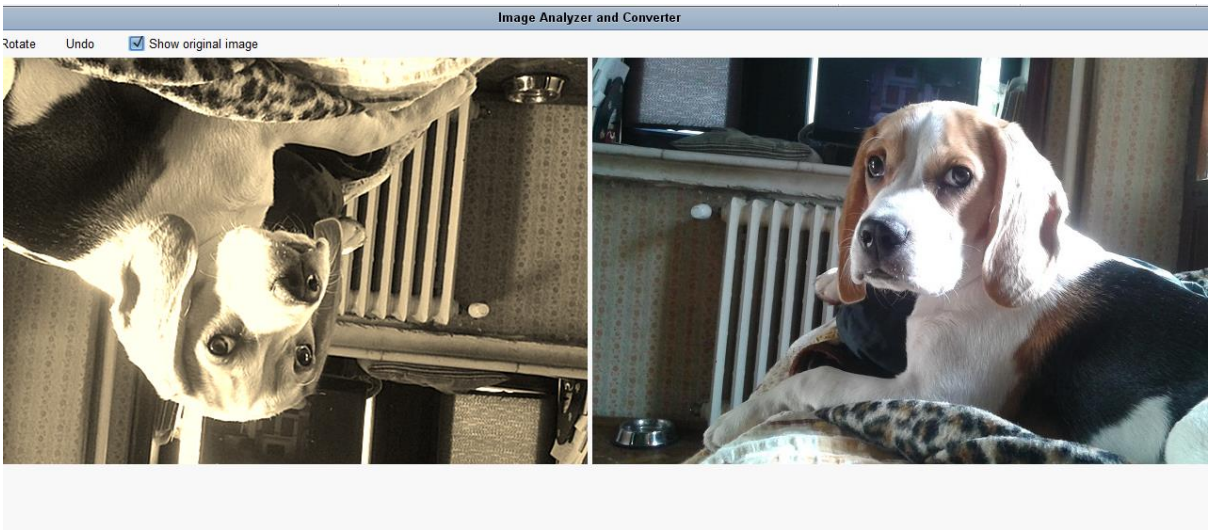
Flip Horizontally



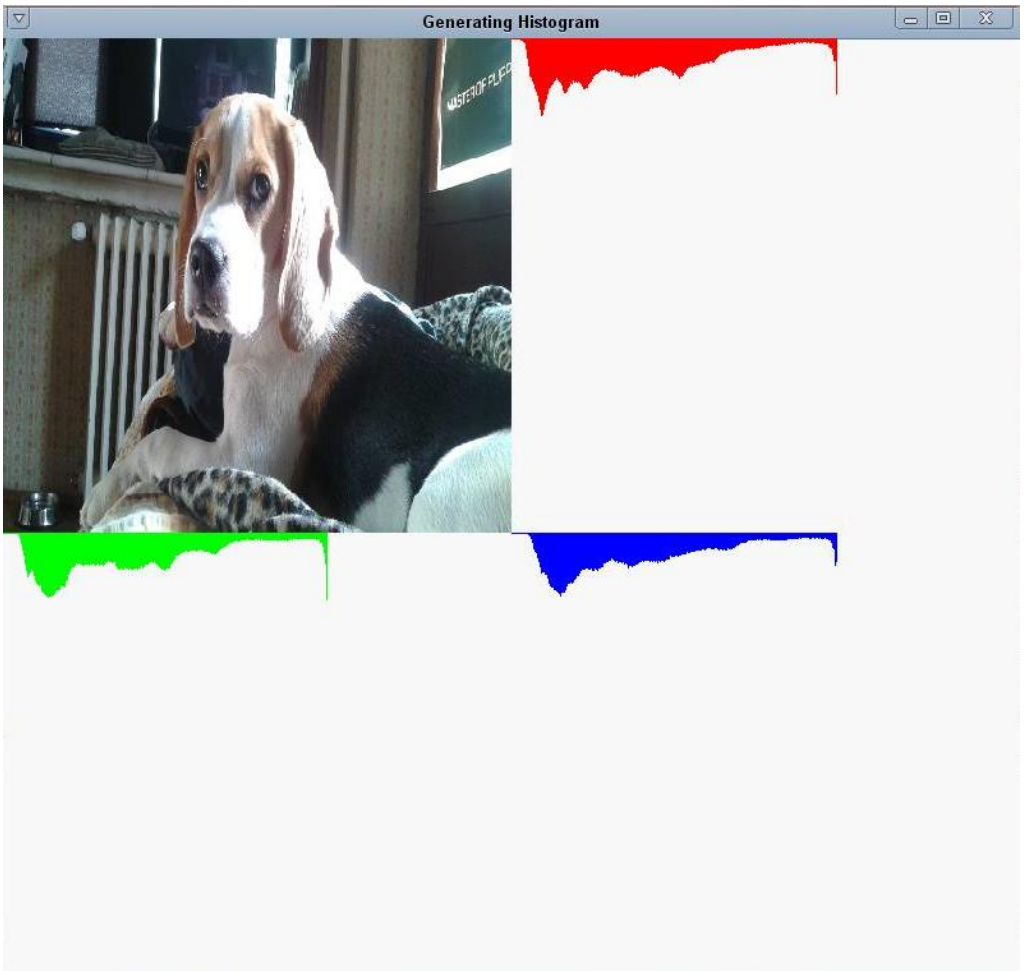
Flip Vertically



Display original image



Display Image RGB Histogram



8 CONCLUSIONS

The application successfully meets the problem's requirements and efficiently operates with multiple file formats.

9 BIBLIOGRAPHY

1. Wikipedia, www.wikipedia.com
2. Sommerville I., Software Engineering 10th Edition