WROCLAW UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS

FIELD:	Computer Science
SPECIALITY:	Internet Enginering

Multimdia & Computer Visualisation.

Histogram equalization algorithm implementation.

AUTHOR:
Jarosław Szumega
Ilona Brzozowska

COURSE SUPERVISOR:
Marek Woda, Ph.D.

GRADE:

WROCŁAW 10.01.2016

Contents

Chapter 1

Project description

1.1 Subject

The main goal of following project is to choose and to implement an algorithm that will perform histogram equalization.
Such an operation is required e.g. when photo/picture was over- or underexposed by light. As a result it is too bright or too dark to be properly percept by human eye.

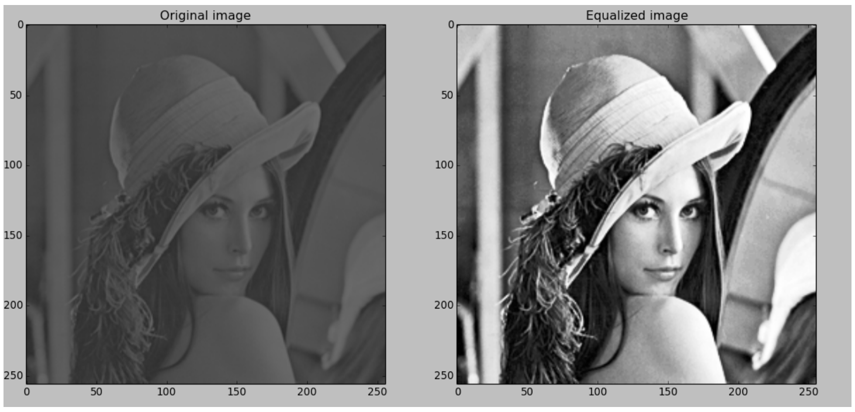


Figure 1.1 'Lena image' - before and after equalizer applied.

1.2 Scope

The work may be divided into parts:

- research related to image processing and histogram normalization,
- implementation selected algorithm and all necessary tools,
- quality comparison.

The result should be useful tool to perform image improvements - histogram equalization - at a level that in fact will improv image general look.

When working solution will be present, th simple tests with built-in Octave/Matlab functions will be performed.

1.3 Chosen technologies

During project classes, the following technologies had been agreed with supervisor:

- Unix systems,
- using Octave engine for image processing,
- Python as programming environment,
- PyQt for user-friendly GUI.

The result will be simple program that wraps essential functions into graphical interface - just to simplify usage of histogram correction.

Chapter 2

Implementation

2.1 Performance issues

The essential part of project are own-written Octave scripts, that perform image processing. As it turned out during developing matrix operations - it is much faster than using Python-defined algorithms.
To achieved cooperation between python interpreter and Octave (native and own functions) there was used **oct2py** module. It simply allows to set up octave session, configure it and call either built-in functions or defined by user.
After that they can be called as a engine object methods.

Listing 2.1 Using octave engine.

```
        octave = oct2py.Oct2Py()
        octave.addpath('./scripts')
        octave.pkg('load', 'image')
```

2.2 Functions in Octave

The most essential function is the one that combines performing equalization with proper pre- and postprocessing. The image type shall be detected. It is necessary because:

- grayscale images need histogram equalization just to be applied on matrix that represents them,
- RGB images at first should be convert to YUV. Then on Y-part the qualization is performed. YUV is agai combained togther and converted into RGB.

As it was stated before - the project consist also of user-defined methods written specifically for image processing.The following functions are own work of author:

- higher-level histogram performer,
- histogram equalization algorithm (uniform),
- converter between RGB and YUV (both directions),

Listing 2.2 Histogram correction with post- and preprocessing

```
function img = hist_processings(I)
[rows,columns,channels] =  size(I);

if channels == 1
  img = histcor(I);

else
  yuv = rgb2yuv(I);
  ui = yuv(:,:,1);
  ui = ui.*255;
  ui = uint8(ui);
  ui = histcor(ui);
  yuv(:,:,1) = double(ui)./255;
  img = yuv2rgb(yuv);
end
imwrite(img,'tmp.png');
end
```

Listing 2.3 Histogram equalization algorithm
(stretching histogram in 'uniform' way)

```
function cor = histcor(img)

for i=1:256
h(i) = sum(sum(img==i-1));
end

cor = img;
s = sum(h);

for i = 1:256
I = find(img==i-1);
cor(I) = sum(h(1:i))/s*255;
end
```

2.3 GUI in Python

Really important part is designing user interface. It should be simple, but also designed with care to be practical.

Qt Designer was used to prepare window and its controls in drag'n'drop way 2.1.

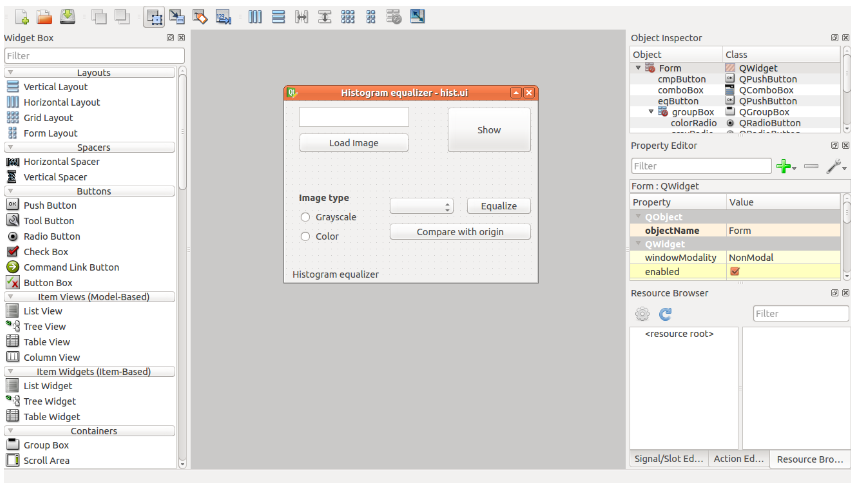


Figure 2.1 QtDesigner tool for designing user interface.

After creation of *.ui file, the intrface can be converted to a form that allow to use it from Python (import as a module).

```
\$ pyuic4 hist.ui > hist_gui.py
```

That creates Python code responisble for Qt window and element drawing. But still, the functionality does not exist!

To connect gui events with python code, the signal/slot 2.3 mechanism was applied.

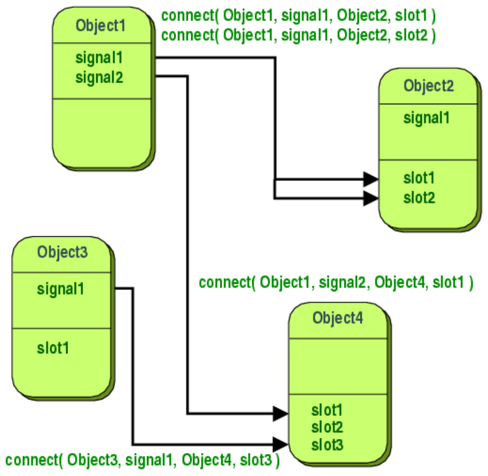

Object1
signal1
signal2
connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect(Object1, signal2, Object4, slot1)
Object4
slot1
slot2
slot3
connect(Object3, signal1, Object4, slot3)

Figure 2.2

Figure 2.3 Illustration of signal/slot mechanism.

Defining connection between signal and slot can be done in following way:

Listing 2.4 Adding action for buttons click event.

```
        self.ui.loadButton.clicked.connect(self.openFile)
        self.ui.showButton.clicked.connect(self.imShow)
        self.ui.eqButton.clicked.connect(self.equalize)
        self.ui.cmpButton.clicked.connect(self.compare)
```

Fixing issues
One last remark is about cooperation between Octave engine and Python. Not always the entire data was properly transfered between engine and interpreter (especially in case of multidimensional arrays like in RGB images).
To solve it, the solution from graphics editors was mimicked - the engine saves *temporary file* to be read by Python-based program.

Chapter 3

Results

3.1 System at work

A few images was tested to evaluat if th program works correctly. The result are really promising (pictures 3.1, 3.2). Especially taking dark, grayscale images.

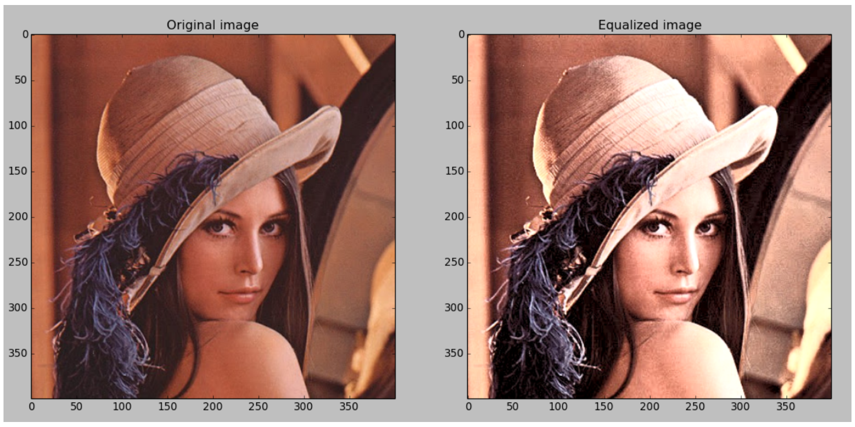


Figure 3.1 RGB image with slightly underexposure.

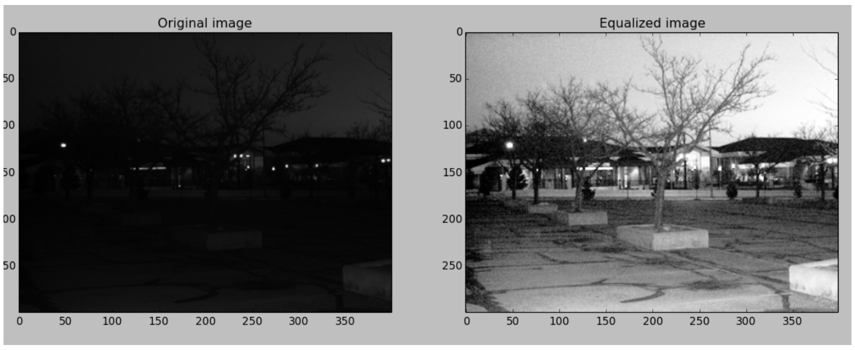


Figure 3.2 Photo taken by night - almost impossible to catch details.

3.2 Comparison to native functions

To compare own histogram correction with Octave's **histeq()** function, the proper script was prepared.

In every case, the Octave algorithm is much faster than hand-written one. But during testing there was an example where author's implementation is better: As it can be

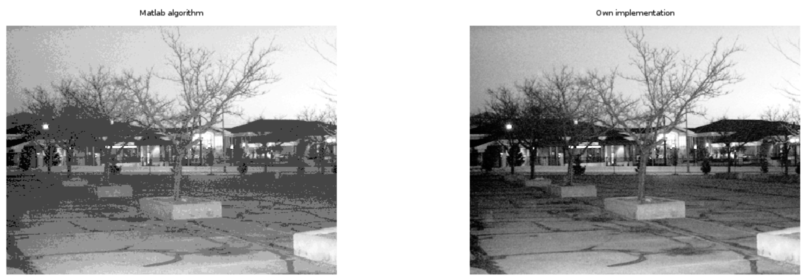


Figure 3.3 Native function confronts user-defined.

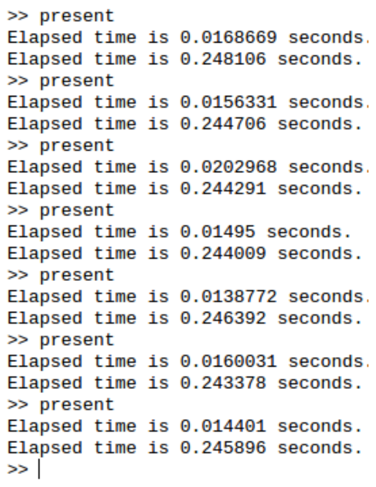
seen with naked eye -the Octave native produces some kind of artifacts. The transition between pixels is not smooth.

In case of author function - there is no such a problem. The image is definitely more 'readable' and it is with no disturbing artifacts.

3.3 Summary

The histogram equalization may help not only fixing not well exposured files, but also to see the previously not seen details (like in case of night scenario).

The intersting issue is also fact, that user-defined algorithm can beat specialist's work and perform better visual effect. Although in case of time execution, the native solution is a lot faster.



```
>> present
Elapsed time is 0.0168669 seconds.
Elapsed time is 0.248106 seconds.
>> present
Elapsed time is 0.0156331 seconds.
Elapsed time is 0.244706 seconds.
>> present
Elapsed time is 0.0202968 seconds.
Elapsed time is 0.244291 seconds.
>> present
Elapsed time is 0.01495 seconds.
Elapsed time is 0.244009 seconds.
>> present
Elapsed time is 0.0138772 seconds.
Elapsed time is 0.246392 seconds.
>> present
Elapsed time is 0.0160031 seconds.
Elapsed time is 0.243378 seconds.
>> present
Elapsed time is 0.014401 seconds.
Elapsed time is 0.245896 seconds.
>> |
```

Figure 3.4 Values are in order Octave time, author's time.