GRAPHICAL USER INTERFACE FOR DATA PROCESSING IN MEDICAL IMAGING

A THESIS SUBMITTED TO
THE FACULTY OF ACHITECTURE AND ENGINEERING
OF
EPOKA UNIVERSITY

BY

THEMISTOKLI DUKOLI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

MAY, 2016

Approval of the thesis:

GRAPHICAL USER INTERFACE FOR DATA PROCESSING IN MEDICAL IMAGING

submitted by……………………… in partial fulfillment of the requirements for the degree of **Master of Science in Department of Computer Engineering, Epoka University** by,

Prof. Dr. ……………...____________________
Dean, Faculty of Architecture and Engineering

Prof. Dr. ………….. ____________________
Head of Department, **Computer Engineering, EPOKA University**

Prof. Dr. …………… ____________________
Supervisor, **……………….Dept., EPOKA University**

Examining Committee Members:

Prof. Dr. …………….. ____________________
………………. Dept., ………….. University

Prof. Dr. ……………. ____________________
………………. Dept., ………….. University

Assoc. Prof. Dr. ,,,,,,,,,,,,,,,,,,,, ____________________
………………. Dept., ………….. University

Date:____________

ABSTRACT

GRAPHICAL USER INTERFACE FOR DATA PROCESSING IN MEDICAL IMAGING

Dukoli, Themistokli

B.Sc., Department of Computer Engineering

Supervisor: Dr. Arban Uka

Nowadays there are a lot of image processing methods that offer a lot of different things. One of them is MATLAB, which is a software that provides a high level programming language combined with a lot of libraries and an easy Graphic User Interface.

Here we look at MATLAB’s Image Processing Toolbox and its capabilities and building GUI using the GUIDE tool. Using these two libraries we will build an application capable of processing images, our focus will be on medical imaging, more specifically finding and comparing the cells in images. The solutions and ideas presented here are open for interpretation and further development.

Keywords: MATLAB, GUIDE, Medical Imaging, Image Processing

ABSTRAKT

NDERFAQE GRAFIKE PER ANALIZE TE DHENASH NE IMAZHERI MJEKESORE

Dukoli, Themistokli
B.Sc., Departamenti i Inxhinierisë Kompjuterike
Udhëheqësi: Dr. Arban Uka

Ne ditet e sotme ekzistojne shume menyra per te procesuar imazhet. Nje nga keto eshte MATLAB, i cili eshte nje software qe na ofron nje gjuhe programimi te nivelit te larte te kombinuar me librari te shumta dhe nje nderfaqe grafike te lehte per t'u perdorur.

Ne kete material do te shofim Image Processing Toolbox dhe GUIDE te cialt jane dy nder libarite e shumta te MATLAB. Duke perdorur keto dy librari do te ndertojme nje aplikacion i cili do te jete i afte te perpunoj imazhe, fokusi jone do te jete fusha e mjekesise; me konkretisht gjetja dhe krahasimi i qelizave qe ndodhen ne imazhe. Zgjidhjet dhe idete e prezantuara ketu jane te hapura per interpretim dhe zhvillim te metejshem.

Fjalët kyçe: MATLAB, GUIDE, Imazhe Mjekesore, Procesim Imazhesh,

Dedicated to my brother

ACKNOWLEDGEMENTS

I would like to express my special thanks to my supervisor Dr. Arban Uka for his continuous guidance, encouragement, motivation and support during all the stages of my thesis. I sincerely appreciate the time and effort he has spent to improve my experience during my graduate years.

I am also deeply thankful to all the other professors for the help and support given to me throughout these 3 years.

My sincere acknowledgements go to my thesis progress committee members, for their comments and suggestions throughout the entire thesis.

I am especially grateful to my parents for being there for me all this time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The huge growth of the computer technology has made that a lot of fields of life to change drastically. A lot of the sciences have gotten a lot of positives from this. Computers have become a vital part of everyday life and are helping people to perform their tasks quicker and better. This made the programmers to focus on creating applications that are easy to use and effective. Computer graphics and image processing fields also benefited from this technology growth.

Initially MATLAB was used to help with the mathematical operations between matrices and vectors, which is way it became popular throughout the teaching facilities. Although it was mostly used for mathematical reasons very quickly a lot of libraries became available. This was made possible since MATLAB can work very well with morphological transformations, is good in illustrating mathematical equations. Also MATLAB can be considered part of the high level programming language group since a lot of the unnecessary details are not visible to the designers and it can work better than the other processing programs with linear and nonlinear filters. So in order decide if it is the right software for image processing we will dive deeper in the way this application works by testing it.

1.1 The study

The purpose of this study is to dive deeper into the image processing field and get as much information as possible. During this study we will use information from different fields like computer graphics, Image Processing Toolbox and Guide Tool from MATLAB. The main materials that will be covered are about adding and subtracting color models, color maps, image transformations, spatial transformations, filters and the way to use them, built axes and also we will look at the main controllers of Guide Tool. Since MATLAB is more used in the teaching facilities to solve mathematical problems and is a bit useful in medicine we will try to implement an image processing application so that we can prove that it also can be used like normal image processing software.

For this study a lot of materials have been consulted for information about image processing and how it can be achieved using MATLAB. The most important and useful information is found in the website of Mathworks which instructs how to use the MATLAB's libraries, but also some information can be found in previous studies about this topic. The following steps will be followed for the creation of the application:

1. We will draft the outline of our program

2. We will build the GUI and the main window using the Guide Tool from MATLAB
3. Build the menu bar
4. Start programming and debugging

After taking all these steps and be sure that the application has not bugs we will test it and present the results.

1.2 Medical imaging

Medical imaging is the technique and process of creating visual representations of the interior of a body for clinical analysis and medical intervention, as well as visual representation of the function of some organs or tissues (physiology). Medical imaging seeks to reveal internal structures hidden by the skin and bones, as well as to diagnose and treat disease. Medical imaging also establishes a database of normal anatomy and physiology to make it possible to identify abnormalities.

In modern medicine, medical imaging has undergone major advancements. Today, this ability to achieve information about the human body has many useful clinical applications. Over the years, different sorts of medical imaging have been developed, each with their own advantages and disadvantages.

1. Using X-rays; X-ray imaging uses an X-ray beam that is projected on the body. When passing through the body, parts of the x-ray beam are absorbed. On the opposite side of the body, the X-rays are detected, resulting in an image.
2. Molecular imaging; Molecular imaging provides detailed information of the biological processes taking place in the body at cellular and molecular levels and can indicate disease in its earliest stages.
3. Other types of medical imaging; Some types of medical imaging work without using ionizing radiation, for example magnetic resonance imaging (MRI) and ultrasound imaging, and have specific uses in the diagnosis of disease.

CHAPTER 2

COMPUTER IMAGES

2.1. Computer graphics

Computer generated images or also known as computer graphics is a very broad field since by definition it includes everything that has to do with images from the process of manipulation of images till the applications that manipulate and acquire the images. We will focus on the part of the processes an image goes through. Firstly the images are divided in two types: vectorial and raster images.

In order to understand the processes that happen to an image we should first understand the way an image is build and the differences between vector images and raster ones. Raster images are made by bitmaps, which is a small grid of rectangles that we call pixels. The pixel is the smallest building block of a raster image and the information it contains is its position in the in the image and the color it represents. On the other hand the characteristics of the bitmap are its size and the bits it takes to represent a pixel. Knowing this we can see that for an image to be of a good quality it needs to have a high number of pixels.

These pixels are arranged in columns and rows, which helps us to determine another characteristic of the raster images which is called image resolution. The image resolution has several ways to be defined for example it can be the number of rows and columns in an image or the most used one is the multiplication of the number of rows with the number of columns and then converted to megapixels. The second way it is more useful since from it we can derive the size of an image which is helpful when saving the images since we want the image of high resolution but low size.

The other type of graphic is the vector one, which differs a lot from the raster graphics. The first and more important difference is that the vector images are not made of pixels but made of paths. The path is a start point, an end point, middle points between these two and the curves and angles along this path. So vector graphics are mostly made of lines, squares and other geometrical shapes which can be formed by paths. This gives them one of the most distinct feature, which is the ability to be scaled without losing the quality of the image. Also since they are not made of pixels their size is smaller compared to raster ones.

Knowing all these we can see that either one of this type of graphics have their advantages and disadvantages. As mentioned above vector graphics keep their quality even when they are scaled on the other hand raster images if are scaled a lot become ‘pixelated’ and lose quality. Another difference is their size, since it is dependent on the number of pixels raster graphics are bigger then vector ones. Also raster images have more file formats that can be saved as while vector images do not have a big variety of formats. And lastly but not least vector graphics can be converted to raster images pretty easily but converting raster images to vector ones is not impossible but is a very difficult process. As seen above none of the types of images is better than the other but rather they complement each other.

2.2 Colors

Another feature that defines the types of images except the way they are build is the type of color system that it uses. Usually the color systems use three main colors; the values of these three colors are used to give a value to a specific color on a pixel. These three numbers are also used to divide the color systems in two groups: additive and subtractive.

The most known system is the RGB, which stands for red-green-blue, and is an additive one. This system works by adding together the three main colors and then mixing with black in order to get the color we need. This way we start from black and try to go to white, which makes it very good to be used in monitors, TVs etc. On the other hand the subtractive system uses three different colors as the main ones and instead of adding to black it subtracts from white to give the color needed. The main colors of this system are cyan, magenta and yellow. Since the way the colors are mixed is starting from white and trying to go to black, this system is mostly used on printers.

There are also other different systems to create colors but they all are alternations of the two main ones mentioned above RGB and CMYK. As mentioned above RGB stands for the initials of the three main colors used by this system red, green and blue. In this system a color on a pixel is represented by three numerical values which start from 0 and can go till 255. So knowing this we get that black is represented by (0, 0, 0) while white is (255, 255, 255).

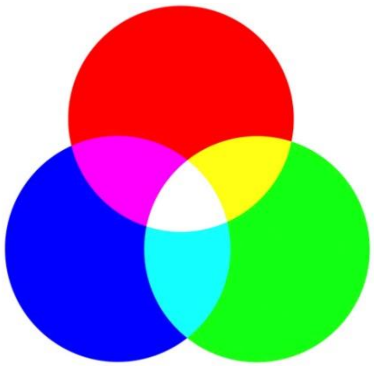


Figure 1 RGB color scheme

On the other hand the CMYK system uses four colors instead of three, which are cyan, magenta, yellow and black. In this system the value of the color of a pixel is represented by four numerical values, which stand for the percentage of each color in the mix, so it starts from 0 till 100. Since this method subtracts from white till we go to black it means that the code (0, 0, 0, 0) represents white while (100, 100, 100, 100) represents black which is the opposite of the RGB method.

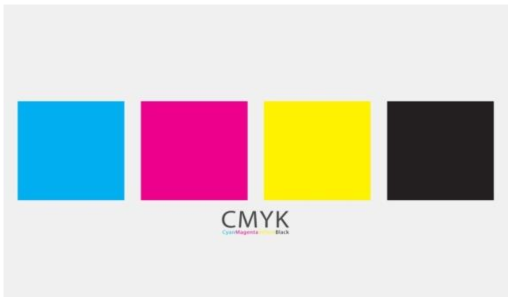


Figure 2 CMYK color scheme

2.3 Colormap

Another important characteristic that defines images is the color map used by the image. A color map is defined as a lookup table specifying the colors to be used in rendering a palettized image. There are a lot of different types of color maps; we will look at some of them down below.

The most common known of the types of color maps is the bitmap. The bitmap works by storing a value between one and zero, a bit, for every pixel in the image. The meaning of the bit is that if it has value 0 doesn't have a color and if it has value 1 it has a color. So it means that bitmap is very good for storing information about black and white images. A slightly change bitmap called grayscale uses, instead of one bit per pixel, eight bits per pixel. Grayscale differs from bit map in that way that instead of representing only black and white; it uses different shades of gray to represent the colors. This is made possible from the use of the eight bits since this allows us to use a scale of 256 values. The grayscale is often created by measuring the intensity of the light at each pixel.

Another very used color map is the indexed color map. This method works by storing the color information not in the pixel of the image but in a separate array called palette and the pixel points to the position of the color in this array, this palette has a maximum storing capacity of 256 distinct colors. This kind of method is very used because it can save disk space and speed up the process of file transferring.

There are also a lot of indexed color map systems that are more powerful, one of them is the true color system. In this system each of the main colors are represented by 8 bits, so we have 24 bits for all the colors in the image, this way we get 256 different shades of red, green and blue. Also the number of available colors becomes 2^{24}, which is a very large number, more than the human eye can discriminate. Usually this system is used in high-quality photographic images. For each pixel, generally one byte is used for each channel of RGB while if a fourth byte is present is usually used as an alpha channel. There also exist systems that use more than 8 bits for each of the main colors. But since the human eye can only see ten million different colors, these deep color systems are not very commercially used. These systems help a lot with the processing of the image since they enable us to separate the image into three different ones and working on them separately.

2.4 Compression

File formats are very important if we want to process images since different formats are represented in distinct ways by the computer. The most important factor to be considered in the file format is the type of algorithm each format uses to compress the image. There are two types of algorithms used lossless and lossy algorithm. These two differ because the lossless algorithm, as its name suggests, tries to compress the image without losing in quality but this means that the size of the image file will bit a bit high. On the other hand the lossy algorithm compresses the image so that its size can be as small as possible but in doing so it loses in quality of the image.

Another way file formats differ from one another is the type of image they try to compress. Raster images are the most popular and some of the more used file formats are .jpeg, .png. While the vector graphics are mostly used by image processing programs so the file formats depend from them, for example .dwg that AutoCAD uses. Except all these there is also a file format .eps that can be used to save either a raster image or a vector one and is used a lot by the Adobe image processing programs.

We will look a bit more on the above mentioned formats in order to fully understand how they work and how important are when processing an image. First is .jpeg (Joint Photographic Experts Group) which uses a lossy type algorithm but it tries to discard information that is not very noticeable and it supports an image size of 65535x65535 pixels (2^{16}=65536). The .png (Portable Network Graphics) as the name tells is a format that is mostly used on the internet, mostly by the browsers. It uses a true color mapping and has a pretty good compressing ratio since it uses a lossless algorithm. Also since was made to be used on the internet it only uses the RGB method. And lastly the .eps (Encapsulated PostScript) file format which is mostly used for vector images but can also can save raster images. This format works by keeping the characteristics of the graphic and it has a low resolution preview encapsulated for some programs to display it, making the file to take a big disk space. This type of file is inserted within another PostScript document and used by it.

All the above mentioned features are mostly used to describe an image to the computer and are influenced by the image features like:

1) Brightness that decides if the image is too dark or too light and can alternate it by adding or subtracting by the RGB value in order to make the image lighter or darker respectively
2) Hue and saturation which are used to represent the color and the perception of it

3) Contrast that is used to make the objects in the picture more emphasized from the background

CHAPTER 3

MATLAB USE IN IMAGE PROCESSING

3.1 MATLAB

MATLAB stands as an acronym for '*matrix laboratory*' and is a powerful numerical computing environment. It is developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions, implementation of algorithms and also interfacing with programs written in other programming languages. An image and a video (which is a series of images appearing at a frequency of approximately 30 frames per second) can be directly represented by a matrix. The simplest case is a black and white image where each cell in the matrix records the pixel intensity. Although its main intent is to be an environment for numerical computing it has a lot of libraries that can be used to analyze data, simulation or even statistics, these toolboxes are what make MATLAB so versatile and important for a lot of science fields.

3.2 The data

MATLAB's basic data element is the array which most of the time it does not need dimensioning. However the main feature of MATLAB is its extensibility, since it can incorporate as part of it new applications that are created based on it. These toolboxes are ready to use functions used to solve particular problems by using specialized technologies. These libraries can be available for a large number of scientific areas as: signal processing, control systems, neural networks, fuzzy logic, simulation and a lot others. But in order to understand MATLAB better we must look at the five main parts that is based on. First is the **MATLAB language** that is a high level array/matrix language with control flow statements, functions data structures, input/output and object-oriented programming features. It allows both programming in the small to rapidly create quick throw away programs and also if needed to create complete large and complex applications. Second part of MATLAB is its **environment and desktop tools**. This is the set of tools and facilities that you work with as the user. It includes facilities for managing the variables in your workspace and importing and exporting data. It also includes tools for developing, managing and debugging M-files, which are MATLAB's applications. Third piece of the **MATLAB System is its graphics**. It includes high level commands for two and three dimensional data visualization, image processing, animation and graphics presentation. It also includes low level commands that allow to customize the appearance of graphics as well as to build complete GUI on your MATLAB applications. Fourth is MATLAB's **mathematical function library**, which it the vast collection of computational algorithms from the elementary ones like sum, sine etc to more sophisticated functions like matrix inverses, Bessel functions etc.

Last but not least is MATLAB's **Application Program Interface (API)**, which allows the user to write in a different language for example C and still interact with MATLAB. It includes facilities for calling routines from MATLAB called dynamic linking and also calling MATLAB as a computational engine.

As mentioned above MATLAB's main difference from the other programming environments is that it represents its data as vectors or matrices. There are a lot of different ways of declaring the matrices; the most used is to just type the values. But to do so there are some rules that should be followed: firstly the elements of a row should be separated by spaces or commas, secondly the end of each row is assigned by using the semicolon and thirdly the whole list of values must be inside square brackets. Except this way there are also ready built in commands that create matrices or vectors according to the user's needs. To access an element in the matrix round brackets are used, inside of them are the number of row and column the element is. The variables in MATLAB are declared as in any other programming language but with the difference that in MATLAB the variable type is not declared but is implicitly assigned. Also if a result variable is not declared MATLAB creates a temporary variable that stores the latest operation's value.

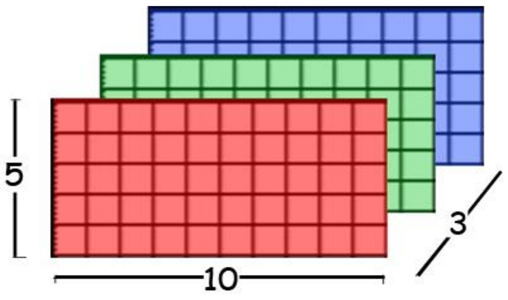
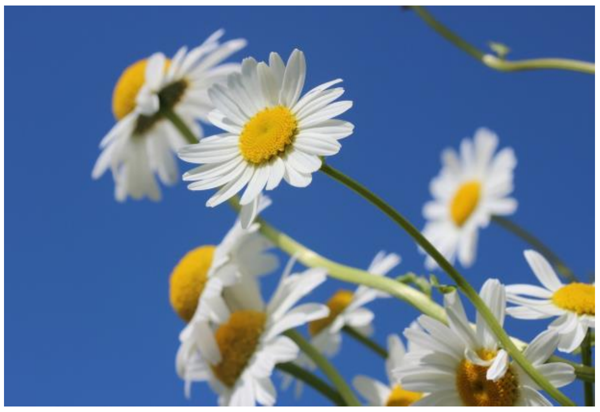
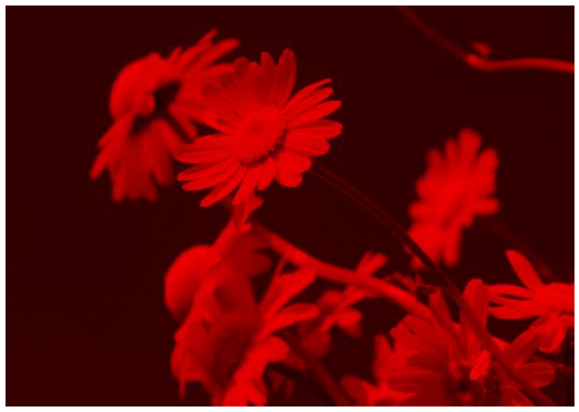


Figure 3 MATLAB RGB color matrix

For example:





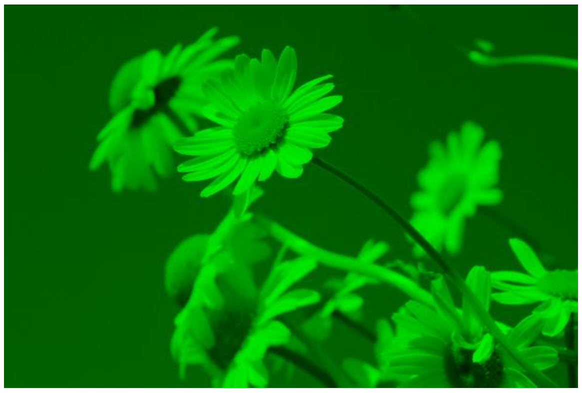
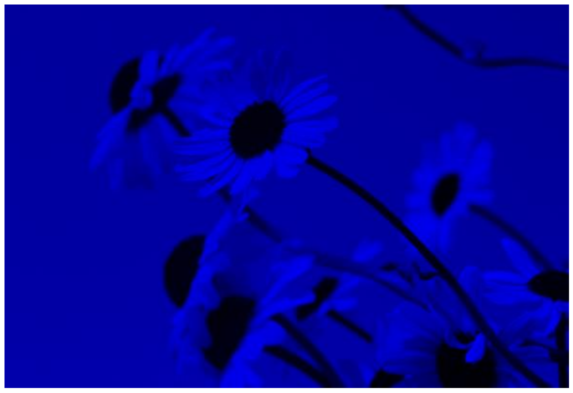



Figure 4 The three RGB channels in an image, (a) The original image, (b) The red channel image, (c) The green channel image, (d) The blue channel image

Understanding the way the variables and matrices are used by MATLAB is important since the images that we will process will also be stored as a vector or a matrix. Most pictures are stored in standard two dimensional matrices, where each element corresponds to a pixel in the image, which is the case with bitmap images. But most of the images are encoded using indexed maps which records the value of the color the pixel has so this type of images are represented as a three dimensional matrix, where the third value keeps the information about the intensity of the RGB color. The default way of location pointing in an image is using the pixel coordinate, for example pixel (2,3) corresponds to x=3 and y=2. As we can see the order of the coordinates is reversed.

3.3 The toolboxes

However what makes MATLAB different and very good for scientific studies is the big variety of toolboxes that are integrated, starting from automation, electrical engineering, robotics, modeling and simulation, medicine, music and a lot more available calculations. We will look at some of the most important toolboxes and their purpose and then will focus on the image processing one.

One of the most used toolbox is the group used for signal processing purposes. Two of the most important toolboxes of this group are **Communication Toolbox** that handles all the problems that arise on the physical layer of communication systems; it provides mechanisms for simulation, designing modulation and demodulation and also multiplexing of signals. The other on is the **Signal Processing Toolbox** that is mostly used for audio and wireless communications and it offers filter designing. There is also a toolbox for non engineering fields like statistics or data management. This toolbox enables users to manage data, compute probabilities and also the designing of experiments that use statistic data. There is also another group of toolboxes that is more focused on mathematical operations and optimization. The two main toolboxes of this group are the **Optimization Toolbox** which contains functions that handle non linear equations and solves quadratic problems, the other one is the **Symbolic Math Toolbox** which it is mainly used for integration, differentiation and transformation of equations. The last toolbox mentioned here is the group of image processing ones. In this group are worth mentioning the **Image Acquisition Toolbox** which has a big number of functions that handle receiving signal, image or video, directly to MATLAB; this is made possible by the ability of its interface to recognize the different types of video cameras used today. Also this toolbox has a lot of functions that can transform the signal according to the user's needs. The second toolbox of this group is the **Image Processing Toolbox** that has functions that deal with image processing and alternation and it supports all the above mentioned types of graphics. The number of functions and image alternation that are possible with this toolbox is very big; we can reduce noise, filter the image, adjust the saturation, enhance the image etc, this are some of the functions, but we will look at this toolbox more deeply in the next chapter.

CHAPTER 4

IMAGE PROCESSING TOOLBOX

Here in this chapter we will focus on the Image Processing Toolbox and its functions since it will be the cornerstone where we will focus our image processing program. All the functions and their descriptions are taken from MATLAB's official website and different documentations available online.

4.1 The color functions

The first group it comprises of functions that have to do with the colors and the colormap of an image. First are the functions which tell us the type of colors that are in the image; **isbw(A)** returns the value 1 if the image is black and white, following these trend there are functions that check if the image is graymap **isgray(A)** and if the image has the RGB colormap with the function **isrgb(A)**. Knowing the colormap of the image is important since it tells us what kind of other functions we can use to transform it depending by the colormap. Know that we know the colormap we have functions that we can change it so that we can apply more transformations to the image. The most trivial function and that can be used for all the colormaps is **colormap(map)** which sets the colormap of an image to the value put in the brackets. Except this functions there are also other functions specific to a colormap like **rgb2gray** which converts RGB images to grayscale ones, **im2bw** on the other hand converts all types of images to blackand white ones. Black and White images have pixel intensity of 'zero' and 'one'. This can be shown in the images below:

Figure 5 Image transformations, (a) The original image, (b) The grayscale image, (c) The black and white image

These functions are important because we can get a lot of information about the image and then apply the proper transformation functions according to the type of colormap.

Another group of functions is the one that enhances the image attributes. In this group there are functions like **imcontrast** which as the name suggests changes the contrast of an image, **imadjust** is another function that changes the contrast of an image by clipping the pixels that are not on the specified intensity range. Last but not least is the function **brighten** which depending on the value given it darkens or lightens up the image, if the value is between 0 and -1 the image gets darkened while if the value is between 0 and 1 the image gets lightened.

4.2 The shape and operational functions

The most important functions of this toolbox are the spatial transformation ones. These functions are used to change the shape and size of an image according to the user needs. The first one is **imresize** and as the name suggests it changes the size of the image according to the parameter given, if the number is between 0 and 1 the image gets smaller while if it is bigger than 1 it gets bigger. The other function of this group is the **imrotate** which rotates and image according to the parameter it gets in the brackets, clockwise if the number is negative and counterclockwise if the number is positive. There are also some not very used functions but worth mentioning like **fliplr** and **flipud** which respectively flip the image along the vertical and horizontal axis. The issue with these functions is that they can only be used with grayscale graphics that’s why they are not used very much.

Along with all these functions the toolbox also has the standard functions of opening, saving and showing the image that the user is working. The functions are **imread** for getting the image, **imshow** for displaying the image and **imwrite** for saving the image. The first function gets as a parameter the name of the image and converts it to a two dimensional matrix if it is grayscale or black and white or three dimensional one it the image has RGB colormap, the second function gets as a parameter the name of the graphic file and if needed a second one for the type of the colormap if we want to display it differently from the original one. Lastly the **imwrite** function gets as a parameter the matrix the image is converted to, the name we want to save it and thirdly the type we want to save it, according to the type if needed a fourth parameter is added to specify specific attributes for the image format. The difference from all the other environments is that MATLAB does all this based on the matrix system.

Another function that we are going to use for testing is the **imfindcircles**, which is an already build in function that analyzes the image given and tries to find the circles that exist there given some parameters. Also another function that we will use for testing is the **viscircles** which will color the perimeter of the circles that **imfindcircles** will find.

4.3 Minor functions

Also there are some minor functions worth mentioning that help in one way or another to the processing of the image. First is the **imfinfo** which takes as a parameter the name of the image and shows useful information about the image, this may include: the date of the last modification, the size, the format, width and height of the image in pixels and a lot more. Another one is the **impixel** function that it is usually used on images with RGB colormap since it returns the value of three main colors of a specified pixel, given as an input parameter. Another worth mentioning function is the **fspecial** which is used to create a two dimensional filter according to the type specified as an input, some of these types are disk, log, motion, Gaussian etc. This function cannot be used directly on the image but it is always used as a parameter for the function **imfilter** that is used to apply a filter to a desired graphic image, as an input this function takes the type of the filter and the name of the image we want to apply the filter on. All this is possible since MATLAB sees an image as a matrix and all this are just matrix operations.

CHAPTER 5

GUIDE

5.1 Graphical User Interface (GUI)

Now lastly before we start developing the image processing application we will look the GUIDE (Graphical User Interface Development Environment) tool that MATLAB has. We will use this tool to create the GUI of the application. But first we must understand what GUI (Graphical User Interface) is.

GUI is used so that users can interact with electronic devices through graphical icons and visuals indicators; the term was created in order to distinguish this type of interaction from the text based ones. The actions in a GUI are usually performed through direct manipulation of the graphical elements. The goal of GUI is to enhance the efficiency and ease of use of the program, in order to ensure this structure is kept the method of user-centered design is applied. This method has some essential rules that help with the design like: proper visual composition in order that the environment will be aesthetically pleasant from the user's perspective, every button and menu should have a very understandable name and its function should be very clear to the average user, also the interface should be flexible and adjustable for every type of user. So the main guidelines make the GUI to be simple, intuitive and the result should be available with as much less effort as possible.

After knowing this we take a look at what GUIDE offers us, to access the GUIDE tool we can either type "guide" in the command line or at the menu tab New->GUI, after that the GUIDE menu pops up. The menu has some predefined layouts as well as a blank one for us to choose. For our purpose we will look at the blank one and the options if offers. After selecting it we get the main window where we have the layout on the right and the different components that can be drag and drop on the left. The components available are:

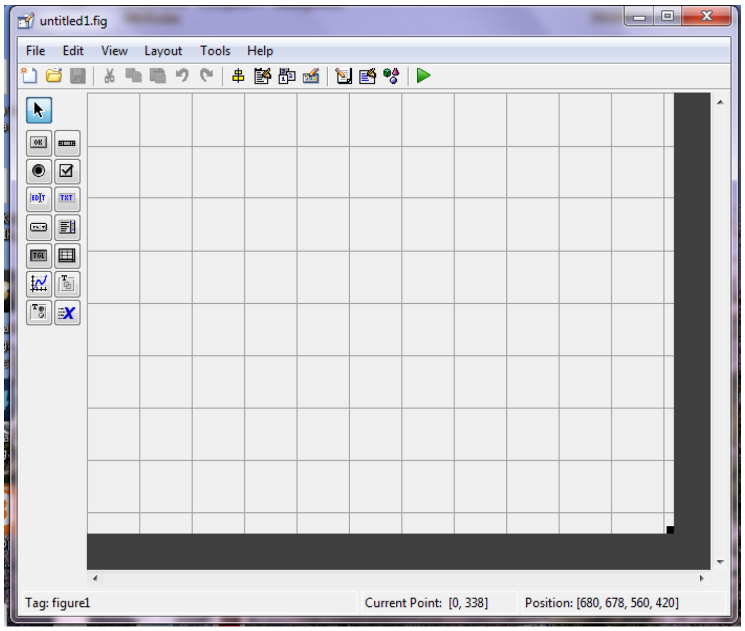

untitled1.fig
File Edit View Layout Tools Help
Tag: figure1
Current Point: [0, 338]
Position: [680, 678, 560, 420]

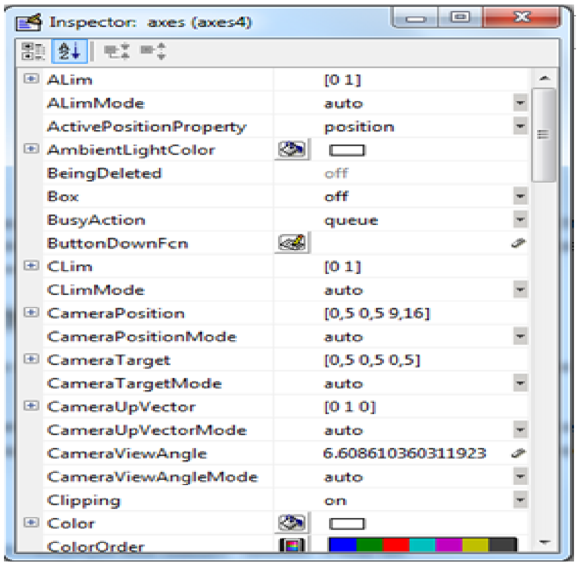
Figure 6 The main interface of GUIDE

- Push Button - it is a component that every time is pressed it performs an action.
- Slider – accepts numeric input within a specific range by enabling the user to move it.
- Radio Button and Check Box – Check Boxes and Radio Buttons are similar, that is when an option is selected an action is generated but the difference is that the Radio Buttons are mutually exclusive.
- Edit Text and Static Text – Edit Texts are fields that enable the user to enter or modify text strings while Static Texts are used for labeling and cannot be changes interactively by the user.
- List Box – Display a list of items and enable users to select one or more items.
- Table and Axes – Used to create tables and graphs.
- Panel and Button Group – The two arrange components into groups making the interface easier to understand, the difference is that Button Groups are used only on selection behavior buttons like radio ones.

Also we get a lot more options on the menu bar in order to make our GUI more user friendly, some of these options allow us to resize the layout if needed, align the components in the layout etc. Once the sample layout is saved we see that it generates two different files, a .fig one and a .m. The .fig file contains all the description and object properties of the objects while the .m file contains all the code that controls the actions also known as callbacks. The callbacks is created using the tag name that a component has, this property can be accessed and changed by using the Property Inspector. In the .m file there are also two important functions except callbacks which are: Opening function that executes tasks before the GUI is shown and the output function that is mostly situational and it returns values if needed.

5.2 The components of GUIDE

All these components have different attributes and actions that should be specified, this is done by double clicking the component and the Property Inspector pops up for the specific component, another way to get the dialog box is by going to the menu bar View->Property Inspector and there select the component that we want to inspect. The menu is comprised of a list of changeable attributes which are grouped depending on their function. Most of the components have similar attributes but also they all have some features unique to them. These attributes can also be changed using the command *set*, after each is called by the command *get*.



Inspector: axes (axes4)

ALim	[0 1]
ALimMode	auto
ActivePositionProperty	position
AmbientLightColor	⬜
BeingDeleted	off
Box	off
BusyAction	queue
ButtonDownFcn	
CLim	[0 1]
CLimMode	auto
CameraPosition	[0,5 0,5 9,16]
CameraPositionMode	auto
CameraTarget	[0,5 0,5 0,5]
CameraTargetMode	auto
CameraUpVector	[0 1 0]
CameraUpVectorMode	auto
CameraViewAngle	6.608610360311923
CameraViewAngleMode	auto
Clipping	on
Color	⬜
ColorOrder	🟦🟩🟥🟦🟪🟨⬛

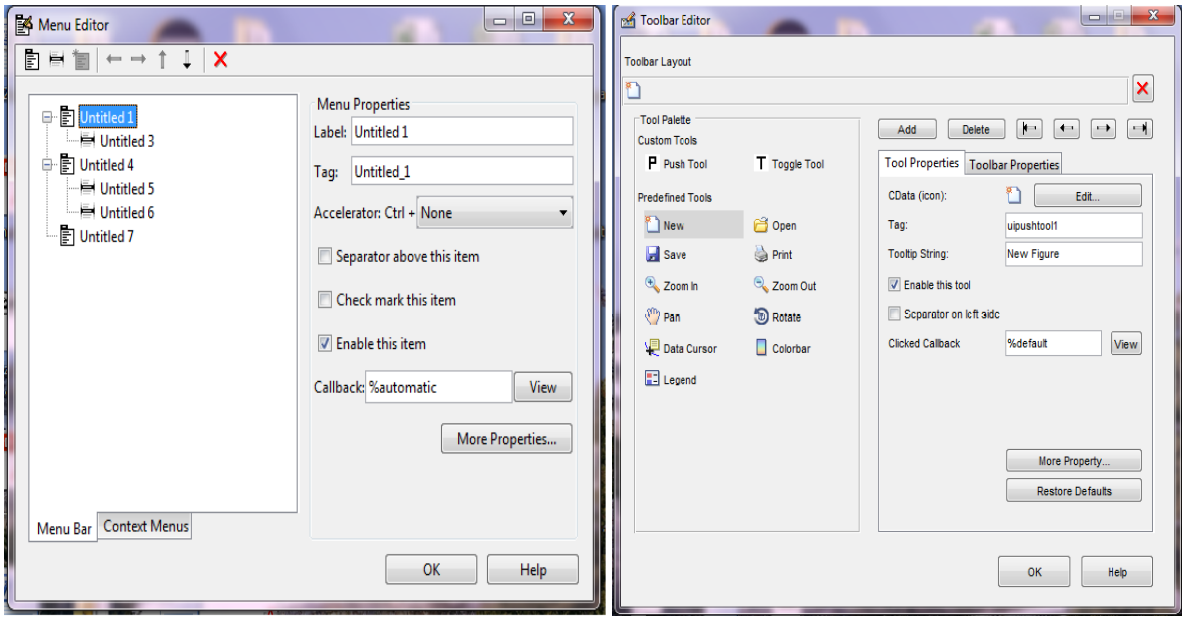
Figure 7 Property inspector of a GUI element

In order for the user to be able to find the attribute quickly MATLAB has grouped them according to their responsibilities. The first group is responsible for the appearance traits, here we can find properties like *background color* that as the name suggests defines the color of the background of a component, *string* is used to put the name on a button, *visible* which makes the component to be visible to the user or not etc. The other group is the one that manages the functionality of the component. Part of this group are properties like *enable* which according to the selected setting it defines if the component will be operational, here also is the *tag* property mentioned on the previous paragraphs and as mentioned there it is used to create unique names for each component so that it can be specified better in the programming file, also another attribute mentioned before is found here, which is the *get* command that is used to get the user's data. Another group manages the positioning and labels of the components, here we find attributes that affect the position of the component object, the font of the text in the component and also the alignment of this text inside the component. Last but not least is the group of traits that manages the callback actions of these components. In this group there are properties like *callback* which takes as an input the name of an .m file and whenever the object is activated the referenced function will execute, *interruptible* is an attribute that when activated it allows a second operation to happen and interrupt the main callback etc. But there are also some attributes that are specific to each component and we will look them at the next paragraph.

Sliders are for example are a component that has some specific properties, the *style* trait is the one that takes as a parameter the maximum and the minimum value that are shown in the slider and after defining them the *slider step* attribute can be used to indicate the step that the slider should take in case of the user moves it. Another component that has a lot of specific attributes is the axes. If we open the Property Inspector of this object we can see properties like *box* that defines the area where the axis will be, between two or three dimensional. Other properties that axes component uses are *xgrid* or *ygrid* that defines the size of the grid that the axes will be shown, also there is the *xtick* or *xticklabel* which are used to put the respectively the numbers that will be shown along the x axis and the label it will have, these properties also are available for the y axis. Other components that have specific traits are tables and list boxes which will not be looked more deeply since they are not very useful in image processing.

5.3 Menus and Toolbars

For our GUI to be practical and easy to use we must have a toolbar and a menu bar. This is possible by going to the Tools menu and there we see the Menu Editor and the Toolbar Editor options. GUIDE allows us to create pull-down menus, standard drop-down menus situated on the top, as well as context menus which appear when the user right-clicks on the component. After the Menu Editor is opened the menu bar option is by default selected. Here by clicking New Menu option we get the pop up box that we must fill with the needed information like *label* which is the name that will be shown to the user, *tag* is the name that will be used to identify the object in the functions. Also we get the *accelerator* property where if we want we specify a shortcut for the menu option, this can be done only to items that do not have a submenu and last is the *callback* is where the name of the .m file or the function is put. Here also we can add submenus or even create cascade menus, the properties available for either the submenus or the cascade ones are the same as for the menu object. To create a context menu we select the toolbar and after that we go to the New Context Menu option on the toolbar. Here unlike the drop down menu we get only the *callback* and the *tag* properties; in this case the *tag* property will be used to associate the proper context menu with the proper component, this is done in the Property Editor's trait *uicontextmenu*. After this we click the New Menu Item as in the previous example and we get the same attributes to be specified.



Untitled 1
Untitled 3
Untitled 4
Untitled 5
Untitled 6
Untitled 7

Menu Properties

Label: Untitled 1

Tag: Untitled_1

Accelerator: Ctrl + None

☐ Separator above this item

☐ Check mark this item

☑ Enable this item

Callback: %automatic

CData (icon):

Tag: uipushtool1

Tooltip String: New Figure

☑ Enable this tool

☐ Separator on left side

Clicked Callback: %default

Figure 8 The toolbar configuration interface

By default the GUIDE does not create a menu or a toolbar, in order for us to display our created menu we have to put the value of the *menu bar* option to none and if we want so that together with the custom crated menu to include the MATLAB's standard menu than we set the value to figure.

For creating a toolbar, firstly we have to activate it like we did with the menu bar by setting the *toolbar* property to figure after that we go to the Tools-> Toolbar Editor we get a popup menu. The window is divided into three parts on the left we have the tool palette from where we get the tools, on the right we have the tool properties and the toolbar properties where different attributes are defined and on top of them we have the toolbar layout. The making of the toolbar is much easier from that of the menu one since here we only have to drag and drop the desired tool from the palette to the layout and since the tools are predefined the *callback* attribute and some other attributes are already assigned except for the *label* one that is blank and is filled by the user.

CHAPTER 6

IMPLEMENTATION OF THE APPLICATION

Before starting the implementation of the application we should outline the steps that we will take in order to finish the task properly. First of all we must have a clear idea of what we want our program to be able to do, in our case the application should be able to perform image processing operations, like cropping, rotation, altering the colormap, histogram analyze etc. on the images it takes as an input by the user.

Second we have to order the work that should be done and what part of the application should be created at every step. Designing of the main application window is the first task that should be done, after that we build the menu bar and the toolbar for our GUI and once the layout is completed we go to the code of the application and alter the parameters and the callbacks of the components so that they will function according to our needs. When a prototype is finished the next thing to do is to test it and fix any bug or error.

6.1 The design

As said above the first thing to do is to design the main application, a good practice is before everything to do a sketch of how we want our window to look and based on that to start creating it. To open the GUIDE tool we can just type guide in the command window or go to the menu New-> GUI, after doing this we get a pop up window from where we can select to create a blank layout or load an existing one, in our case we select to create a blank GUI. The hardest part now is to maintain a proper level of functionality for the components and their position should have a logical flow. First things first we have to set our window size, this can be done by going to Property Inspector set the height and width of the window to the position attribute. Knowing what our program will do we start positioning the elements, on the left and right of the window we will set an axes component where the ticks will be removed and a black border will be set since here it will be the area where the images will be shown. In order for the window to have consistency the background of the axes will be set the same color as the rest of the window. Also on the top center will be two buttons which will change the brightness of the image and static text element above to name the buttons. Below these buttons will be a reset button so that the user if needed will load the original image discarding all the changes, another that will compare the histograms of the circles found on the images and two static text components one will be used to name the component while the other will show needed information to the user about the image that was loaded. Also below the image axes we will put the components for defining the parameters needed for the *imfindcircles* function; there will be two static texts with the name of the of the parameters, two edit texts for the radius input and one slider for defining the sensitivity. After the main layout is done, the next thing to do is to create the menu bar which in our case will have File, Transformations and Filters as menu items.

The first menu option is File which is pretty common and very familiar to users since all of the modern applications have it. Our File menu has 4 submenus Open, Save, Info and Close. In order for the program to be as much intuitive and as much familiar to the users we will use the known shortcuts at the accelerator attribute for each submenu. At the Transformations menu we have all the alternations we can make an image go through like crop, rotate etc. This is all possible thanks to the available MATLAB functions mentioned on the previous chapters. These functions will also be used for the Filters menu where we have more transformations that an image can take like noise removal, color map conversion etc, most of them have to do with the *fspecial* and *imfilter*.

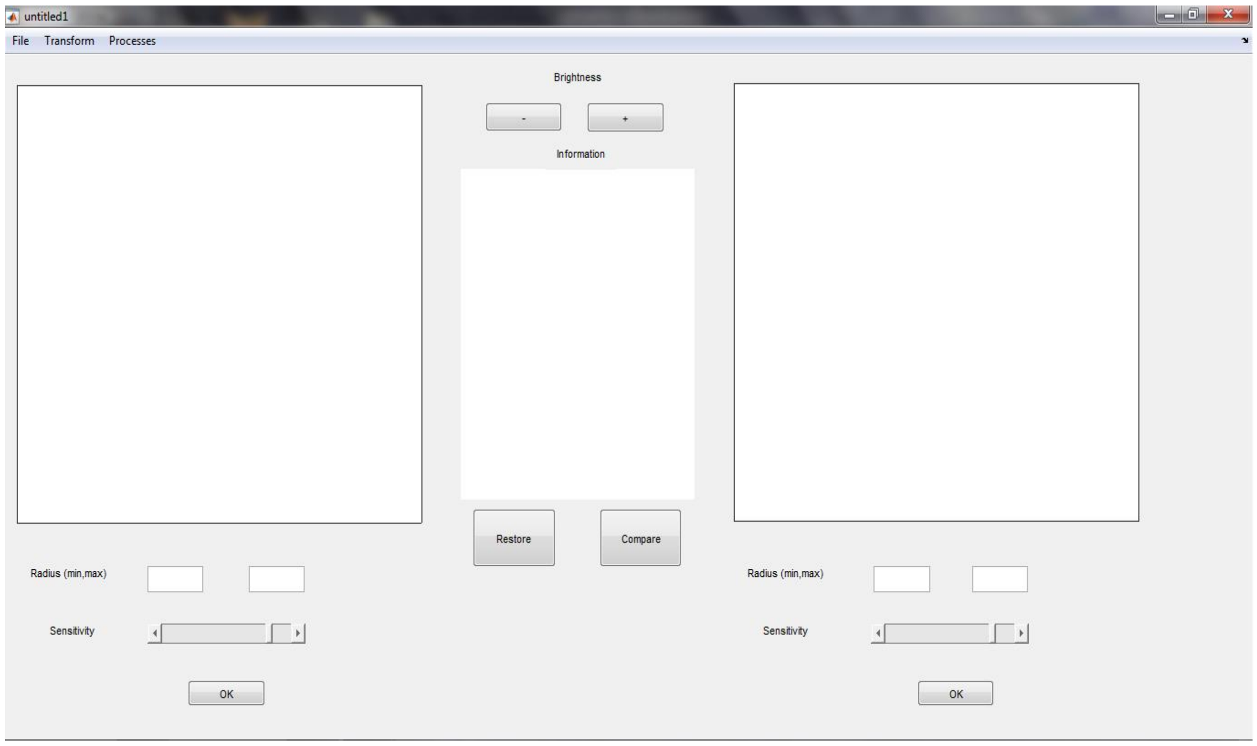


Figure 9 The view of our application

6.2 The coding

Now that the graphical part of our layout is finished we have to look at the operational part of it. We have to make sure that all the menus and components function properly; this can be done either by hard coding the functions (callbacks) in an .m file or drag and dropping the components to the workspace bar and set their parameters using the Property Inspector. Here we will try to use both methods interchangeable but will focus more on the hard coding part.

When doing the programming part of the application we must have into consideration the way the user will interact with it and the kind of problems he may stumble upon. Firstly the user should not be able to make a selection from the Transformations or the Filters menu if the image is not loaded by the user, the moment the image is loaded the menus should be available to be selected. Another thing to keep in mind is the restore button, in order to not have problems while using this option when the image is loaded it will be assigned to 2 different variables. The first variable will be used as the main image that will be processed and when the restore option is selected the original image assigned to the second variable will be loaded. The function *uigetfile* will be used to get the image from the local storage, the function *imread* will be used to load the image to the application and the function *imshow* will be used to print the image to the layout. Also another important option on the menu bar is Save, this will be done by using the functions *uiputfile* and *imwrite*.

The *imfinfo* function will be used to get needed information about the image and make it visible to the user at the information section in the layout. For the filters we will be using the Gaussian filter which has a build in function in MATLAB and together with this we will be using the color map conversion functions explained in the previous chapters. Using the color map conversion functions we have to keep in mind that not all the types of conversion should be active at one time, but depending on the type of color map the image has the proper conversions should be made available to the user for selection. Also the *imfindcircles* function is important so we have to carefully connect it with the GUI elements from which it will get the parameters needed for it to function properly. The return values of the function will be taken and used for the *viscircles* function as well as for the building of the comparing histograms which will be shown on a new pop-up window. This window is a default window created by the *histogram* function and it contains an axes where the plot is created.

6.3 The testing

After we finish the polishing of the code and the way the layout elements will work, we have to test the application so that we will be sure that it gives us the result we need and that there are no bugs that might cause a not so pleasant experience to the user. For this reason I tested the application several times using different types of image files and with different colormaps. Firstly colored images with high contrast and brightness were used for testing

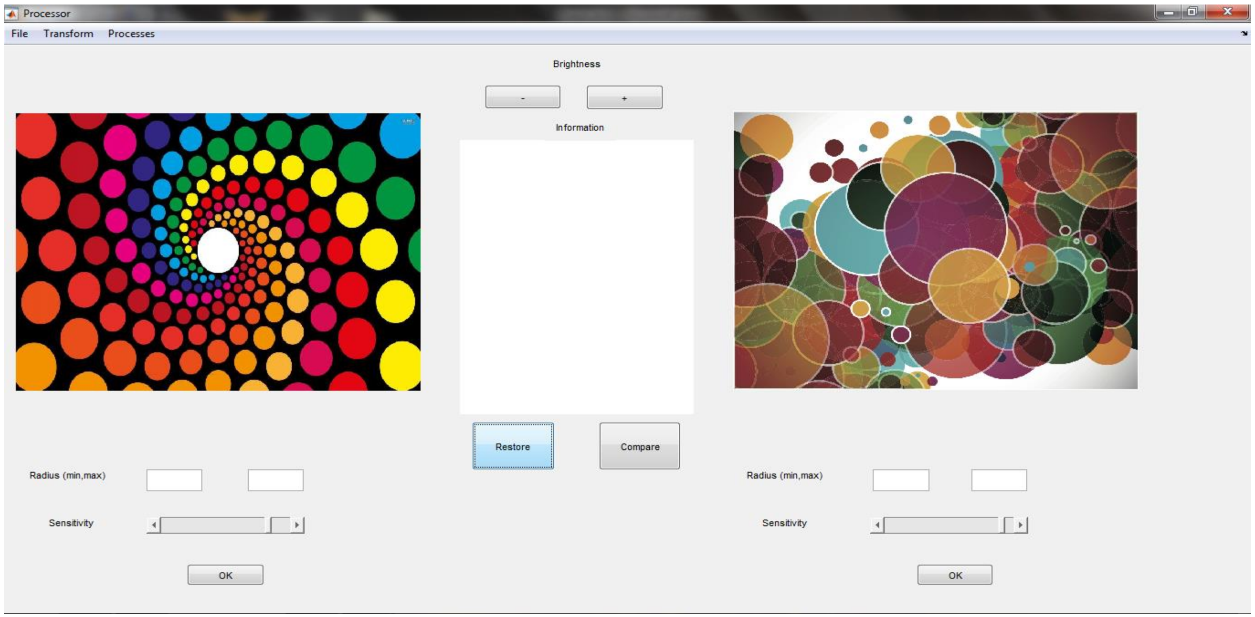


Figure 10 The first test of our application

As we can see the images are colored which means they use the colormap technology. In order for us to test the *imfindcircles* function we will have to convert them to grayscale ones. This will change the way that the images will be represented as matrices since that is the type of matrices the function uses and will make the perimeter drawing easier to spot to the user. After all the image changes are done we put the max and min radius of the circles we want the function to find and define the sensitivity we like using the slider. These parameters will be given to the function and after the circles are found the application will draw their perimeter at the image in order for the user to identify them. The final stage is this:

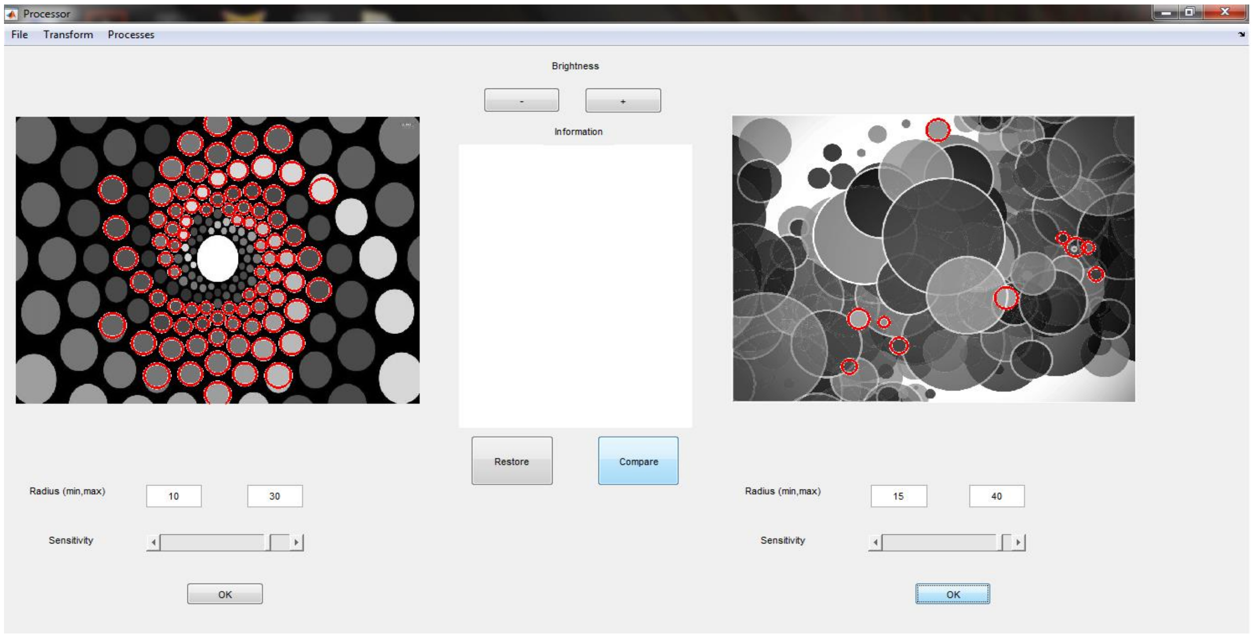


Figure 11 The *imfindcircles* functionality of the application

If the user is not satisfied with the circles found he can change the parameters until the output satisfies him. If needed he can compare the results by getting a histogram of the images and the results they gave for example:

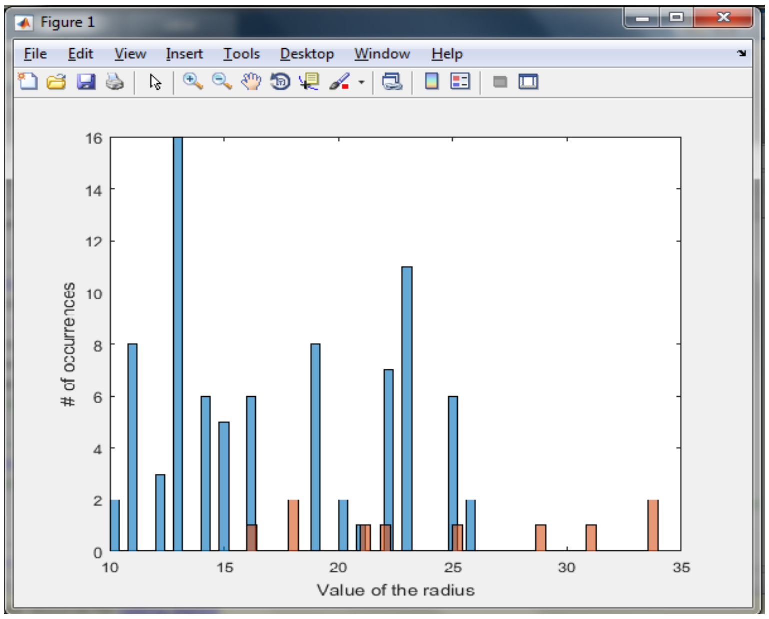


Figure 12 Comparing the radiuses of the circles in the images

The second test done is more practical one since the images used are taken from medical sites and are images of cells. In the first set the cells are in a normal environment and on the second set the cells are put on a solution and are photographed after 6 days in that solution. These images are not very clear and have a low contrast and brightness, they already are grayscale so they can be processed immediately but in order for the images to be more clear to the user we will increase the brightness. This is the first screenshot:

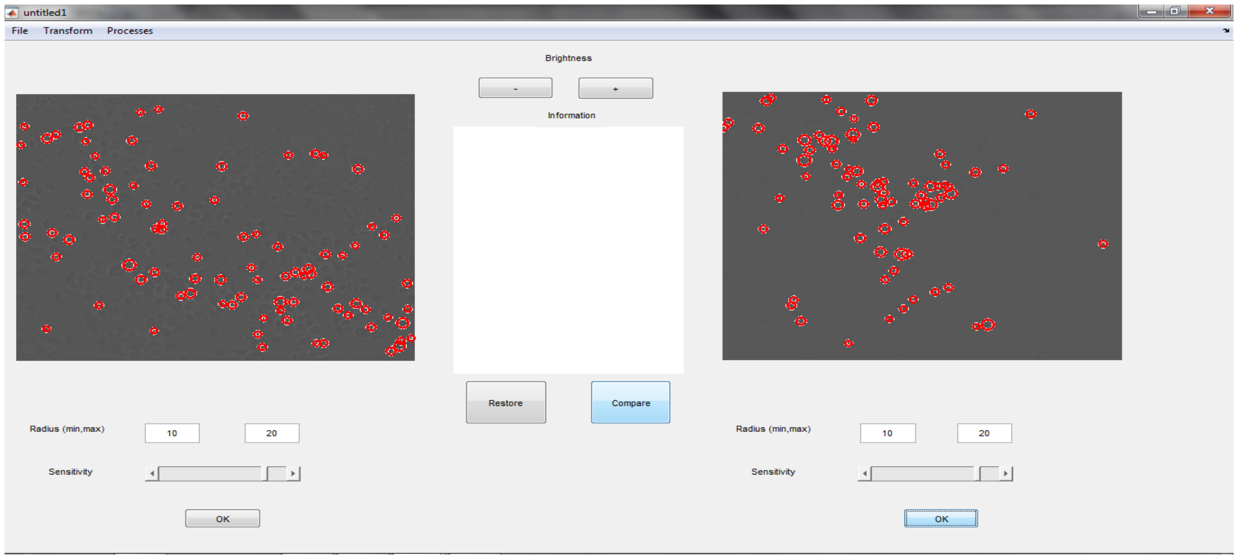


Figure 13 The second test using medical images (cells)

Another way we can make the images more clear is using the histogram equalization function that we have implemented to the application. Although this transformation will change the way the images are processed by the application in order to make the images easier to be seen for the user. The resulting view is something like this:

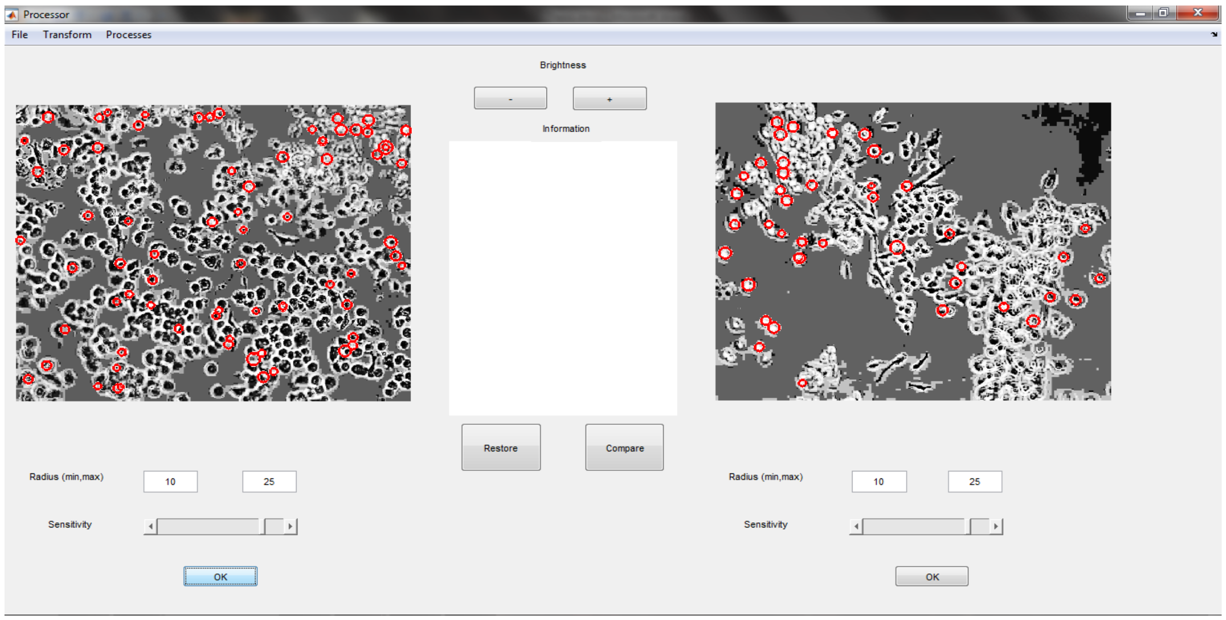


Figure 14 The images after they are processed with histogram equalization

We will take the same steps as we took at the first test of the application, firstly we will give the parameters for the function. After setting the desired parameters we can see the functions found some cells in the images. Now in order to compare them we click the compare button which will show the histogram of the two images comparing the number of the radiuses at each image.

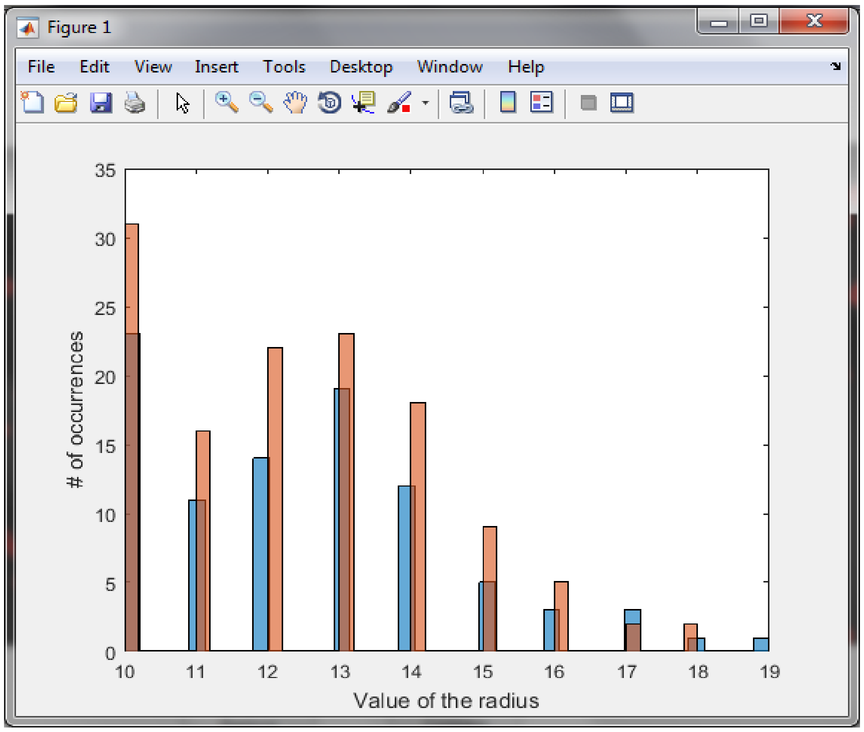


Figure 15 The histogram comparing the radiuses of the cells

If we see the images carefully we can see that the function missed a lot of cells also it found a lot of cells wrong, this shows the limits of the built in function for finding circles in MATLAB. This problem can be the focus on future studies.

The tests done above are a good way to watch the functionality of the application but in order for a proper and deep insight it is needed to make as many tests as possible with as many different testers as possible, this will help that all the possible scenarios and all the possible file formats will be tested and if needed the proper changes will be done to please all the different users. The further development and improvement of the application will be discussed on the final chapter of this paper.

The tests we did with the cells are focused on three metrics the number of cells, the mean of their radiuses in pixels and their standard deviation in pixels. As mentioned above we have cells photographed the first day and the sixth day. Also the cells are divided in two groups depending by their status active or suspended. Furthermore, these groups are divided in three types 2µm, 20µm and unpatterned. The segment of the radiuses given to the function is from 10 to 20 pixels.

The results of the test done with the cell images are gathered and shown in the tables below while the analyze of the results is done in the next chapter:

	Day 1			Day 6		
Metrics for the active cells	2μm	20μm	Unpatterned	2μm	20μm	Unpatterned
Nr of cells	42	279	92	133	253	75
Mean (radius)	12,6	13,9	12,4	14,1	14,2	13,36
Standard deviation (radius)	2,2	2,7	2,1	2,9	3,1	2,8

Table 1. The data taken from the application for the active cells (the values are in pixels)

	Day 1			Day 6		
Metrics for the suspended cells	2μm	20μm	Unpatterned	2μm	20μm	Unpatterned
Nr of cells	81	50	237	27	155	141
Mean (radius)	13,4	13,9	11,9	12,9	13,3	12,4
Standard deviation (radius)	2,3	2,5	1,9	1,9	2,5	2,1

Table 2. The data taken from the application for the suspended cells (the values are in pixels)

CHAPTER 7

CONCLUSIONS

After finishing all the steps that were needed for the application to be complete we can see that it works pretty well, has a simply functionality and all the components behave as we planned. The only drawback is that it is a simple application with a limited functionality due to the use of default built in MATLAB functions.

Our goal was to build an image processing application that I think is was achieved. We started by looking at the way images work and what we wanted out application to do. Later on the development stage we gave a high emphasis to the GUI of our application. The coding part of the application was a bit difficult but MATLAB's environment was a big help with the already built-in functions, so our focus was directed towards the proper way the components should interact.

Also analyzing the tables with the results of the cell images, we can see that in the case of the active cells the 20µ and the unpatterned ones have dropped in numbers unlike the 2µm ones. By looking at the mean of the radiuses we see that in all the cases the number has gotten bigger which tells us that the small cells have dissolved by the solution. In the case of the suspended cells we see that we have the opposite effect where the 2µm cells number has gotten smaller while the 20µm and the unpatterned ones have risen in numbers. Looking at the mean metric we see that the value difference is different in each case, this is not helping us to give and conclusive idea about the behavior of the cells in this case.

During the making of this paper except from the coding part we got to look at topics related to computer graphics and MATLAB libraries which helped us to figure out how to create the application and gave us a good understanding of the processes an image goes through. After we finish the application, we can see that the work was done properly but the possibilities the Image Processing Toolbox gives us are immense. Also GUIDE is a very good tool when it comes to creating a very user friendly interface and it can be used to a greater extend. This tells us that MATLAB is a very powerful environment that can be used to create even more elegant and complex applications, not only for image processing but for a wide range of different fields of study.

The only drawback in this case is the use of the already built in functions that MATLAB provides which limits the application, but this problem can be fixed by creating our own functions for finding different types of elements in the images. This very well can be a topic which can be studied and developed more on future works. Also the new tool developed for the newer version of MATLAB R2016a called App Designer can be integrated. This tool can be helpful in order to make the graphical view of the application more intuitive and easier since it has a set of interactive controls and plots.

REFERENCES

www.mathworks.com

www.wikipedia.org

‘ Digital Image Processing Using MATLAB ’ 2nd Ed. by Gonzalez, Woods, and Eddins (2009)

http://www.tutorialspoint.com/matlab/index.htm

‘ Graphics and GUI with Matlab ’ 3rd Ed. by Patrick Marchand and O . Thomas Holland (2002)

Digital Multimedia, Lecture Materials, Epoka University

http://www.medicalradiation.com/types-of-medical-imaging/

Introduction to Medical Imaging, Michal Strzelecki, Biomedical Engineering (2013)

‘ Medical Image Processing in a Clinical Environment ’, Wolfgang Birkfeneller, Medical University Vienna (2011)

https://www.dartmouth.edu/~library/biomed/guides/research/medimages.html

http://www.sciencesourcemedicalimages.com/

APPENDIX A

Appendix A.1 The code that gets the images

```
function Load1_Callback(hObject, eventdata, handles)
[filename1,pathname1]=uigetfile({'*.jpg';'*.gif';'*.png';'*.bmp';'*.tif'},'Select File');
if filename1==0
    return;
end
handles.pathname1=pathname1;
handles.filename1=filename1;
handles.var1=strcat(pathname1,filename1);
handles.image1=imread(handles.var1);
handles.original1=imread(handles.var1);
axes(handles.axes);
imshow(handles.image1);
set(handles.Load1,'Enable','On');
set(handles.Load2,'Enable','On');
set(handles.Info,'Enable','On');
set(handles.CounterClockwise,'Enable','On');
set(handles.Clockwise,'Enable','On');
set(handles.Crop,'Enable','On');
set(handles.Gaussian,'Enable','On');
set(handles.Motion,'Enable','On');
set(handles.Sharpen,'Enable','On');
set(handles.Gray,'Enable','On');
set(handles.Bw,'Enable','On');
set(handles.pushbutton1,'Enable','On');
set(handles.Vertically,'Enable','On');
set(handles.Horizontally,'Enable','On');
set(handles.Add,'Enable','On');
set(handles.Remove,'Enable','On');
set(handles.Histeq,'Enable','On');
set(handles.Gray,'Enable','On');
guidata(hObject,handles);
```

Appendix A.2 This code resets all the changes to the original image loaded

```
function pushbutton1_Callback(hObject, eventdata, handles)
handles.image1=handles.original1;
axes(handles.axes);
imshow(handles.image1);
handles.image2=handles.original2;
axes(handles.axes1)
imshow(handles.image2);
```

Appendix A.3 The code that sharpens the image

```
function Sharpen_Callback(hObject, eventdata, handles)
H=fspecial('unsharp',0.05);
handles.image1=imfilter(handles.image1,H,'same');
axes(handles.axes);
imshow(handles.image1);
habdles.image2=imfilter(handles.image2,H,'same');
axes(handles.axes1);
imshow(handles.image2);
guidata(hObject,handles);
```

Appendix A.4 This part of the code applies filters to remove the noise

```
function Remove_Callback(hObject, eventdata, handles)
handles.image1=imgaussfilt(handles.image1);
axes(handles.axes);
imshow(handles.image1);
handles.image2=imgaussfilt(handles.image2);
axes(handles.axes1);
imshow(handles.image2);
guidata(hObject,handles);
```

Appendix A.5 The function that applies the histogram equalization transformation

```
function Remove_Callback(hObject, eventdata, handles)
handles.image1=imgaussfilt(handles.image1);
axes(handles.axes);
imshow(handles.image1);
handles.image2=imgaussfilt(handles.image2);
axes(handles.axes1);
imshow(handles.image2);
guidata(hObject,handles);
```

Appendix A.6 The code that changes the colormap of the image to grayscale and black and white

```
function Gray_Callback(hObject, eventdata, handles)
handles.image1=rgb2gray(handles.image1);
axes(handles.axes);
imshow(handles.image1);
handles.image2=rgb2gray(handles.image2);
axes(handles.axes1);
imshow(handles.image2);
set(handles.Vertically,'Enable','On');
set(handles.Horizontally,'Enable','On');
set(handles.Add,'Enable','On');
set(handles.Remove,'Enable','On');
set(handles.Histeq,'Enable','On');
set(handles.Gray,'Enable','Off');
```

```
guidata(hObject,handles);

% ------------------------------------------------------------------
function Bw_Callback(hObject, eventdata, handles)
handles.image1=im2bw(handles.image1);
axes(handles.axes);
imshow(handles.image1);
handles.image2=im2bw(handles.image2);
axes(handles.axes1);
imshow(handles.image2);
set(handles.Vertically,'Enable','Off');
set(handles.Horizontally,'Enable','Off');
set(handles.Add,'Enable','Off');
set(handles.Remove,'Enable','Off');
set(handles.Histeq,'Enable','Off');
set(handles.Gray,'Enable','Off');
guidata(hObject,handles);
```

Appendix A.7 This code crops the image if needed

```
function Crop_Callback(hObject, eventdata, handles)
handles.image1=imcrop(handles.image1);
axes(handles.axes);
imshow(handles.image1);
handles.image2=imcrop(handles.image2);
axes(handles.axes1);
imshow(handles.image2);
guidata(hObject,handles);
```

Appendix A.8 The part of the code that flips the image vertically or horizontally

```
function Vertically_Callback(hObject, eventdata, handles)
handles.image1=fliplr(handles.image1);
axes(handles.axes);
imshow(handles.image1);
handles.image2=fliplr(handles.image2);
axes(handles.axes1);
imshow(handles.image2);
guidata(hObject,handles);

% ------------------------------------------------------------------
function Horizontally_Callback(hObject, eventdata, handles)
handles.image1=flipud(handles.image1);
axes(handles.axes);
imshow(handles.image1);
handles.image2=flipud(handles.image2);
axes(handles.axes1);
imshow(handles.image2);
guidata(hObject,handles);
```

Appendix A.9 The code that rotates the image

```
function Clockwise_Callback(hObject, eventdata, handles)
handles.image1=imrotate(handles.image1,-90,'bilinear','loose');
axes(handles.axes);
imshow(handles.image1);
handles.image2=imrotate(handles.image2,-90,'bilinear','loose');
axes(handles.axes1);
imshow(handles.image2);
guidata(hObject,handles);

% --------------------------------------------------------------------
function CounterClockwise_Callback(hObject, eventdata, handles)
handles.image1=imrotate(handles.image1,90,'bilinear','loose');
axes(handles.axes);
imshow(handles.image1);
handles.image2=imrotate(handles.image2,90,'bilinear','loose');
axes(handles.axes1);
imshow(handles.image2);
guidata(hObject,handles);
```

Appendix A.10 The functions that rise or lower the brightness of the images

```
function pushbutton2_Callback(hObject, eventdata, handles)
handles.image1=handles.image1-5;
axes(handles.axes);
imshow(handles.image1);
handles.image2=handles.image2-5;
axes(handles.axes1);
imshow(handles.image2);
guidata(hObject,handles);

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
handles.image1=handles.image1+5;
axes(handles.axes);
imshow(handles.image1);
handles.image2=handles.image2+5;
axes(handles.axes1);
imshow(handles.image2);
guidata(hObject,handles);
```

Appendix A.11 The code that executes when the compare button is pressed (showing the histogram)

```
function Compare_Callback(hObject, eventdata, handles)
figure(1),histogram(handles.r1,40);
hold on
histogram(handles.r2,40);
xlabel('Value of the radius');
ylabel('# of occurrences');
```

Appendix A.12 The code that takes all the parameters and tries to find and show the circles

```
function gatheredData1=gatherData1(handles)
gatheredData1.min1=get(handles.edit1,'string');
gatheredData1.min1=str2num(gatheredData1.min1);
gatheredData1.max1=get(handles.edit3,'string');
gatheredData1.max1=str2num(gatheredData1.max1);
gatheredData1.a=get(handles.sens1,'Value');

function gatheredData2=gatherData2(handles)
gatheredData2.min2=get(handles.edit2,'string');
gatheredData2.min2=str2num(gatheredData2.min2);
gatheredData2.max2=get(handles.edit4,'string');
gatheredData2.max2=str2num(gatheredData2.max2);
gatheredData2.b=get(handles.sens2,'Value');


% --- Executes on button press in ok1.
function ok1_Callback(hObject, eventdata, handles)
gatheredData1=gatherData1(handles);
[cen1,rad1]=imfindcircles(handles.image1,[gatheredData1.min1
gatheredData1.max1],'Method','twostage','Sensitivity',gatheredData1.a);
viscircles(handles.axes,cen1,rad1);
handles.r1=rad1;
guidata(hObject,handles);

% --- Executes on button press in ok2.
function ok2_Callback(hObject, eventdata, handles)
gatheredData2=gatherData2(handles);
[cen2,rad2]=imfindcircles(handles.image2,[gatheredData2.min2
gatheredData2.max2],'Method','twostage','Sensitivity',gatheredData2.b);
viscircles(handles.axes1,cen2,rad2);
handles.r2=rad2;
guidata(hObject,handles);
```