

Justyna Inglot

Advanced Image Processing with Matlab

Bachelor's Thesis
Information Technology


May 2012



MIKKELIN AMMATTIKORKEAKOULU

Mikkeli University of Applied Sciences

DESCRIPTION

 MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences		Date of the bachelor's thesis 07.05.2010
Author Justyna Inglot	Degree programme and option Information Technology	
Name of the bachelor's thesis Advanced Image Processing with Matlab		
Abstract <p>The world of the computer graphics is evolving rapidly every day. The market is full of image processing applications. But how many of them really contribute to the growth of the science and technology? Is there any program that combines image processing functionalities with programming techniques ? The answer to these questions is Matlab.</p> <p>Matlab is a software that provides a high level programming language, many thematic libraries and easy implementable Graphic User Interface mechanisms.</p> <p>This paper presents information on wide aspects of the computer graphics, introduction to Matlab and its Image Processing Toolbox. Later, the thesis focuses on the methods of creating a GUI using built-in GUIDE tool. All theoretical studies are followed by an implementation of an image processing application. A very important step of software production is testing. Few examples of test scenarios along with short descriptions are listed in the final part of this paper.</p> <p>Solutions presented in this thesis leaves an open door for the future development. Possibilities and ideas are illustrated in the last chapter.</p>		
Subject headings, (keywords) Matlab, image processing, graphics, gui, graphical user interface, transformation, digital filters, colormap, color models, rgb, cmyk, guide, image processing toolbox		
Pages 57 p.+ app.8	Language English	URN
Remarks, notes on appendices		
Tutor Matti Koivisto	Employer of the bachelor's thesis Mikkeli University of Applied Sciences	

CONTENTS

1	INTRODUCTION.....	1
1.1	First look over the topic	1
1.2	Main objectives of the study	2
1.3	Realization methods and techniques	3
2	COMPUTER GRAPHICS	4
2.1	Computer graphics in details.....	4
2.2	Color systems	5
2.3	Color maps	7
2.4	File formats	9
2.5	Image transformations.....	10
3	MATLAB ENVIRONMENT.....	12
3.1	What exactly is Matlab?.....	12
3.2	Data representation	13
3.3	Endless possibilities	15
4	IMAGE PROCESSING TOOLBOX	18
4.1	Color transformation functions	18
4.2	Spatial transformation functions	20
4.3	Open, Save, Display functions.....	21
4.4	Other functions.....	22
5	MATLAB "GUIDE" TOOL	25
5.1	User friendly graphical interface.....	25
5.2	Main components of GUI	27
5.2.1	Common knowledge	27
5.2.2	Buttons and Sliders	29
5.2.3	Axes	30
5.3	Creating menu	31
6	DESIGN AND IMPLEMENTATION OF AN IMAGE PROCESSING APPLICATION.....	34
6.1	Where to start ?	34
6.2	Designing the window	34
6.2.1	Menu File	36
6.2.2	Menu Transformations.....	37
6.2.3	Menu Filters and Help	38

6.3 General rules of coding	38
6.4 Testing new application	42
7 CONCLUSION	47
7.1 Evaluation of the final outcome and benefits of the study.....	47
7.2 Future development.....	48
BIBLIOGRAPHY	50
APPENDICES	

LIST OF FIGURES

- Figure 1.** Cube of RGB color model
- Figure 2.** Comparison of RGB and CMYK color systems
- Figure 3.** The example of three dimensional matrix, built in Matlab
- Figure 4.** Ready-built colormaps from Matlab's Image Processing Toolbox
- Figure 5.** Comparison of photographs processed with Image Processing Toolbox
- Figure 6.** Example of graphical user interface with some of the components
- Figure 7.** Property Inspector
- Figure 8.** An example of Property Inspector for slider bar
- Figure 9.** An example of Property Inspector for axes
- Figure 10.** An exemplary menu created in Menu Editor
- Figure 11.** Simple, GUI with ready – built menu
- Figure 12.** GUIDE Quick start window
- Figure 13.** First step of designing the application window
- Figure 14.** First part of menu bar designing
- Figure 15.** Second part of menu bar designing
- Figure 16.** Finished menu bar, some options presented
- Figure 17.** Inactive menu options
- Figure 18.** Informative message box
- Figure 19.** An example of *crop* function
- Figure 20.** Procedures of changing the colormap
- Figure 21.** Removal of the noise for the grayscale picture
- Figure 22.** Illustration of the lightness options and their correction
- Figure 23.** Comparison of a high resolution image before and after compression
- Figure 24.** Comparison of a low resolution and small size image before and after compression

LIST OF TABLES

- Table 1.** Examples of image enhancing filters
- Table 2.** Exemplary hypothetical matches between GUI elements and image processing functions

1 INTRODUCTION

1.1 First look over the topic

Every single day world is evolving very fast. Rapid development of the computer technology has affected all the scientific areas. Medicine, automation, data analysis, finances, biology, chemistry, economics and many, many more have benefited from the technology expansion. People got interested in the possibilities of information technology and they have noticed that computer can help them with daily tasks. This need has motivated software programmers to create new systems, that would incorporate the ease of use with the effectiveness of work. Those big changes have influenced also computer related fields, including computer graphics as well as an image processing. Soon all artists were about to experience a huge step towards the simplification of working with graphics. The Internet was flooded with new applications used to process pictures. Some of them are free and some others are commercial.

In the same time programmers were working on designing a system, that would perform operations on vectors and matrixes in a simple, interactive way. Not long time after creating Matlab, it became very popular, especially among teaching facilities. Many libraries has been developed, among them Image Processing Toolbox (Moler [referred 16.04.2012]). In comparison to other image processing programmes it doesn't score many points. First of all, it is not free. Secondly, it does not provide an easy learnable environment – some programming skills are needed. On the other hand it gives multiple opportunities of illustrating mathematical equations. 'Normal' applications lose with Matlab in the area of an image recognition and filters adaptation. No program works better with using for example morphological transformation than Matlab. It also gives a lot of possibilities for creating linear and nonlinear filters. High level programming language that hides unnecessary details from designers can definitely be considered as an asset too. In order to decide if Matlab is the right tool to implement a software with, the future programmer has to take a closer look on the main purpose of the application.

1.2 Main objectives of the study

There are few goals resultant from this thesis. Main purpose is to learn new information from the topic of an image processing area. In additional, dissertation presented here will cover material from following fields:

1. Computer graphics
 - Raster and vector graphics;
 - Additive and subtractive color models;
 - Various colormaps and their attributes;
 - Selected file formats;
 - Commonly used image transformations;

2. Matlab
 - Quick glance on what Matlab is;
 - Data representation;
 - Exemplary toolboxes and libraries;

3. Image Processing Toolbox
 - Image enhancement functions;
 - Spatial transformation commands;
 - Selected filters usage;

4. Matlab's Guide Tool
 - Main controllers of Graphic User Interface;
 - Property Inspector appliance;
 - Creation of the menu;

Matlab's library Image Processing Toolbox has mostly found usefulness in medical purposes and mathematical problems. This thesis has been created to demonstrate the ability of Matlab to have a 'regular' image processing functionality as well.

In order to achieve that I will design and implement an image processing application. Methods of realization are described in the following subchapter.

1.3 Realization methods and techniques

There are many ways to support the learning process. Finding information might be a hard task if it is not well structured. The most helpful tools are written materials. Studies of previously mentioned topics are based on few books concerning image processing and Matlab. Some of the information may be withdrawn from Cracow's University of Technology lectures. Other valuable pieces of data might be found on the Internet. The website of Matlab's producer – Mathworks is also a wide source of the toolboxes, as well as their functions and parameters.

After a broad study of all the topics of interest, the project of an application will be created. In order to achieve that, a sketch draft will be drawn in the beginning. Then the main window will be built, along with all the elements of the Graphical User Interface. This process will be supported with Matlab's GUIDE tool, described later. The next step of application production is designing the menu bar. Before programming the actual code, it is very important to define possible constraints and errors. Specification of these problems in the early stage of the project management may have crucial influence on the program structure. Whole process of creating the application will be followed by few tests along with results and comments. Finally, I will present conclusions and point out the areas of the future development.

2 COMPUTER GRAPHICS

2.1 Computer graphics in details

To understand better the idea of computer graphics, the definition should be taken into consideration. Everything that refers to the representation and manipulation of image data by a computer stands for term computer graphics. What also can be called with that term are various technologies used to create and manipulate images. There are two types of computer graphics – raster and vector (Ozimek, Lectures from Computer Graphics on Cracow University of Technology, 2009a).

Raster graphics is a way of presenting images as a grid of small rectangles – pixels. Those grids, also known as matrixes are often called bitmaps. Each matrix is built out of certain amount of rows and columns. Looking deeper, each row contains some quantity of pixels and each pixel is assigned its position and color in the image. More pixels image contains – more accurate picture will be. Important information is that every bitmap is characterized by size given in pixels and by number of bits per pixel. The second feature is called a color depth, which represents the number of colors that can be used to present the image (Ozimek, Lectures from Computer Graphics on Cracow University of Technology, 2009a). It can be easily calculated with proper amount of bits used to encode all the colors (1 bit = 2 colors, 2 bits = 4 colors, 3 bits = 8 colors, etc.).

Raster graphics has also one additional trait: image resolution. It is strictly connected with size of an image, the amount of rows (height) and columns (width) mentioned before define resolution. There is also a different way to describe that term – the resolution is given as one number, which is the quantity of all the pixels that image contains divided per one million. This is a product of multiplying number of rows per number of columns and it is presented in megapixels. That outcome is also the amount of pixels that influences on size of an image, while saved on the hard drive. More pixels will need more space to store them. Also known fact is that more pixels give more detailed image. The conclusion would be to save pictures with high resolution but skillfully optimize the size while saving.

Vector graphics differs from raster graphics in many ways. The biggest difference is that images, described with vector graphics are not made from grid of pixels. They are composed of paths, which are defined by the start and the end point, together with curves, angles and other points along the path. Lines, squares, triangles can also be used to represent path (TechTerms.com [referred 23.03.2010]). One of the main assets of a vector graphics is that it can be scaled without losing the quality. File size of vector image is smaller and not dependable on the dimensions of the picture. Taking into consideration both types of graphics, their advantages and disadvantages can be presented. Firstly, the main difference is the size of image file. Because raster graphic files are depending on the amount of pixels, they consume much more disk space. When it comes to ability of scaling, vector graphics seems to be better choice because it is easily scalable without losing the quality.

Raster graphic images lose the quality while zooming in and it becomes “pixelated”. Secondly, saving images as a raster graphic gives the user bigger variety of file formats. Vector graphics images do not provide many exterior file formats for saving them separately from the vector graphic processing programs. Other dissimilarity refers to transforming one type of graphics into another. Vector graphics can be easily saved as bitmaps, which requires giving the resolution. On the other hand changing raster graphics into vector image is a very difficult process. In conclusion, both types complement one another, giving the users great amount of possibilities.

2.2 Color systems

Thinking about computer graphics, more features should be taken under the investigation. In order to understand their meaning, color systems (or often called color models) need to be presented. Color system used in computer graphics is usually described as a three color system. It means that each color on the image is depicted with three numerical values. Those values describe the color used in the picture. Because of those three important numbers, all color systems have been divided into two groups: additive and subtractive. For each one different algorithm is applied.

Additive color systems are based on adding three primary colors – red, green and blue to black and mixing them towards getting new colors. More colors are mixed, more close to white new result is. This color system is mostly used in computer graphics.

Subtractive color model operates on rules opposite to additive systems. Three numerical values are subtracted from white in order to create new color tones. Primary colors used here are cyan, magenta and yellow. More colors are mixed, more close to black new color is. Subtractive model is commonly used while printing. Both systems are complementary to each other (Ozimek, Lectures from Computer Graphics on Cracow University of Technology, 2009b).

There are three well known, common color systems: RGB, CMYK and HSV. First one, additive system RGB stands for red (R), green (G) and blue (B). It is widely used for any image file formats. Graphics that is using RGB model represents each pixel as a three numerical values in brackets. First value is the amount of red, second stands for green and the third one is blue. Those values are used to create color presented on the screen. All values can't outreach 255 and can't be lower than 0. For example triplet (0,0,0) stands for black when (255,255,255) goes as white color. Every other mix of values stands for different color. For better understanding how colors are changing, Figure 1 presents the cube of RGB color. Axe X stands for red, Y for green and Z for blue. With moving along all axes different colors can be created.

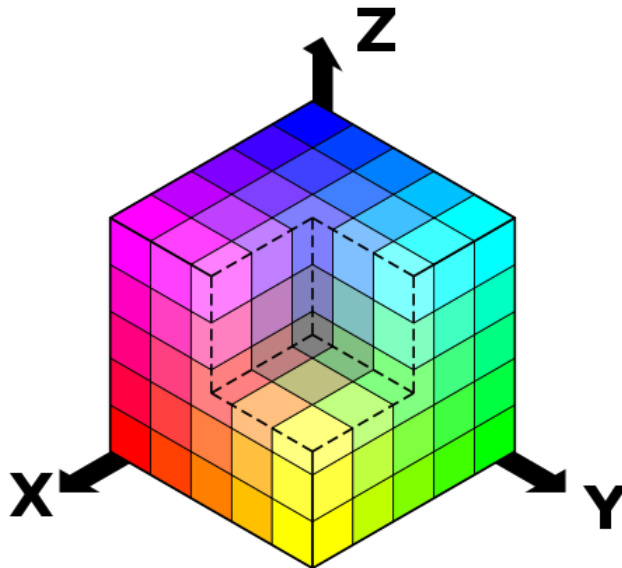


Figure 1. Cube of RGB color model (Wikimedia Commons [referred 25.03.2010])

What is worth mentioning, RGB systems are directed at device. It means that on different device colors can be seen differently. To make printing world easier other color

system is used. CMYK is commonly known as system using four color tones to create new colors. Cyan (C), magenta (M), yellow (Y) and black (K – not to confuse with B like blue) are the main components of this system. Algorithm of formation new colors is subtractive. CMYK system can use three or four values to represent percentage of primary colors used. Opposite to RGB system, white stands as (0,0,0,0). It is important that values have to range between 0 and 100. From combination (100,100,100) black should be created but in reality it is muddy brown. This is why fourth value is added to steer amount of K value to get real black color – (100,100,100,100). The difference between RGB and CMYK system is shown in Figure 2.

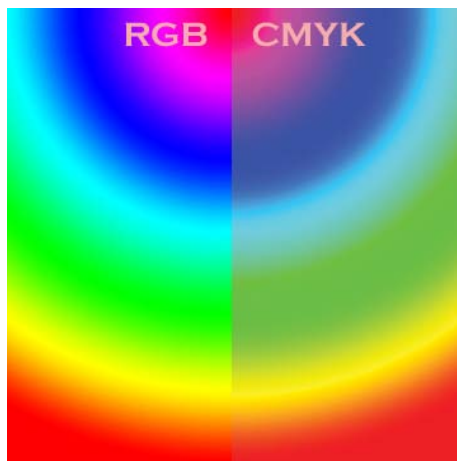


Figure 2. Comparison of RGB and CMYK color systems (Wikimedia Commons [referred 25.03.2010])

2.3 Color maps

Basic knowledge about image properties and colormaps is important for image processing. Colormap is a set of limited colors used in displaying image. There are few different colormaps such as bitmap, grayscale, index color, highcolor and truecolor. This chapter describes each one of them briefly.

Bitmap can be defined in many various ways. Usual meaning of bitmap is map of bits. Considering image processing, bitmap gives the user possibility to store one bit per pixel. Information included in that bit defines if the pixel has color (bit = 1) or doesn't have a color (bit = 0). White color is often used as option of "having the color".

Therefore bitmap is commonly used to store black-white images. Other definition says that each pixel may contain more bits to store the information. For example eight bits per pixel can be used to code 256 tones of grayscale (www.wisegeek.com [referred 25.03.2010]).

Grayscale is a colormap that usually stores information about the image in eight bits per pixel. Like mentioned before, with eight bits it is possible to create scale of 256 shades of gray. Sometimes grayscale is described as carrying only one pixel which stores information about brightness of black. Result is a scale of different tones of gray. All range of gray may be represented as RGB triple value. All three numbers have to be equal, for example (123,123,123) or (3,3,3).

Another useful colormap is called indexed colormap. Each pixel of image stores information about index to the place in colors array, for example set by the user. The biggest palette can contain 256 colors. Indexed color is very useful. It saves computer's memory and disk space. Smaller color palettes can be used to represent icons and pictures with small range of colors. Indexed colormap also supports setting transparent color. Picking transparent pixels is very helpful when it comes to images with irregular shapes. Thanks to that feature object on image can be put on a background without necessity of deleting rectangular shape (Vanderburg et al. 1996, chapter 12).

More advanced colormap is called truecolor. Information about the image is stored in 24 bits, which equals 3 bytes. Each color from RGB model gets 8 bits for storage of information about shade and channel of color. Truecolor system uses at least 256 tones of each color (red, green and blue), which gives huge range of usable colors - 16.777.216 possible variations. This system is often used in high quality photographs and complex images (Ozimek, lectures from Digital image processing, 2010a).

There are plenty of colormaps but one more worth describing is called highcolor. It offers very wide range of colors, bigger than in truecolor system. The description of point on the image contains 16 bits of information. This amount allows to code from 0 to 65535 different values for one RGB color, which gives $65535 \times 65535 \times 65535$ color variations. The precision in highcolor system is twice much better than in truecolor. Mostly because that highcolor is used while photographing more prevalent colors, for example skin tones or skies (Ozimek, lectures from Digital image processing, 2010a).

In truecolor and highcolor system images, picture can be separated into three RGB channels. Each color channel might be useful while processing computer graphics. Having the knowledge about colormaps and color systems is very important when it comes to image processing.

2.4 File formats

For people working with advanced graphics it is quintessential to understand similarities and differences between image files formats. One of the biggest dissimilarities is the algorithm used to compress the size of a file. There is lossless and lossy algorithm. Lossless way reduces size without losing quality of picture but also it cannot compress file to a very small size. Lossy method allows decreasing image size but not without consequences – quality of picture becomes poor (Ozimek, lectures from Digital image processing, 2010b).

File formats can also be divided with regard to type of graphics. From raster graphics, among others, the most commonly used formats are JPEG, GIF and PNG. When it comes to vector graphic there are not many formats because most of them are formats specified by an image processing program, like for example AutoCAD (.dwg) or CorelDraw (.cdr). EPS format is worth noticing. It can be implemented to save both types of graphics. First group compared will be raster graphics formats and it is based on lectures from Digital image processing, provided by Cracow University of Technology.

Abbreviation JPEG stands for Joint Photographic Experts Group. The compression for this file type is lossy. Information that is discarded while compression cannot be noticed with human eye though. JPEG supports about 16 millions of colors.

One less demanding way of saving files is GIF format. Acronym GIF translates as Graphic Interchange Format. It is limited to 256 colors, therefore may be used for simple web images, logos, etc. Despite lossless compression, GIF has worse quality of picture mostly due to small range of colors.

Another format is PNG which stands for Portable Network Graphics. This format is supported by web browsers, although not by all of them. It was made as an answer to GIF but PNG supports truecolor mode and can compress file from 5-25% more. From similarities between those two formats, transparency plays a big role. Both formats allow transparent pixels to be present in the picture. The main difference between PNG and GIF is, that the second one is able to be saved as animations, when first one does not.

When it comes to vector graphics EPS – Encapsulated PostScript, is mostly known format. What is interesting, it can save raster graphics as well as vector images. It contains information needed for printing and very often small-sized preview of the image. EPS format files usually take bigger disk space than other graphic files formats. Most often users save vector graphics in their “natural environment” in order to keep ability of processing them later.

2.5 Image transformations

There is a group of transformations that are about to improve the properties of an image. Brightness, contrast, hue, saturation and threshold are the most widely used.

Brightness, called also luminance helps when image is too dark or too light. Some value is added to RGB colors in order to make picture lighter and some value is subtracted from those components to make image darker. Changing contrast is making difference in properties of the picture which makes the object more distinguishable from the background (Ozimek, lectures from Digital image processing, 2010c). Higher contrast underlines shadows and highlights of the image. Properties hue and saturation are connected with each other. They both refer to color manipulation. Hue changes the way picture is perceived. It can tinge picture with any color, so it will be seen like it was behind a color filter. Saturation of color means that the color can be fully saturated or faded. The last value from possible saturation decrease results in grayscale.

Thresholding is a bit more complicated operation. Input image is usually in grayscale but color picture also works. Output image is a bitmap, where black stands for background pixels and white for foreground objects. The only parameter of thresholding

process is intensity. Each pixel on the image is compared with intensity threshold. If intensity of pixel is higher than the parameter, pixel is set to be white on the output image. Otherwise, pixel becomes black. Thanks to that method density of threshold can be set by the user.

The other group of transformations applies to changes in size or shape of the image. Extending width and height, rotation and twisting image against the axes are just small piece of the whole group containing coordinates transformations. When talking about rotation the angle is the parameter. Most common are 90 degrees clockwise and counterclockwise rotations. They apply to change picture from horizontal to vertical and vice-versa. Other operations are not described because of their simple nature.

3 MATLAB ENVIRONMENT

3.1 What exactly is Matlab?

The name 'Matlab' comes from two words: matrix and laboratory. According to The MathWorks (producer of Matlab), Matlab is a technical computing language used mostly for high-performance numeric calculations and visualization. It integrates computing, programming, signal processing and graphics in easy to use environment, in which problems and solutions can be expressed with mathematical notation. Basic data element is an array, which allows for computing difficult mathematical formulas, which can be found mostly in linear algebra. But Matlab is not only about math problems. It can be widely used to analyze data, modeling, simulation and statistics. Matlab high-level programming language finds implementation in other fields of science like biology, chemistry, economics, medicine and many more.

In the following paragraph which is fully based on the MarthWorks, 'Getting started with Matlab', I introduce the main features of the Matlab.

Most important feature of Matlab is easy extensibility. This environment allows creating new applications and becoming contributing author. It has evolved over many years and became a tool for research, development and analysis. Matlab also features set of specific libraries, called toolboxes. They are collecting ready to use functions, used to solve particular areas of problems. Matlab System consist five main parts. First, Desktop Tools and Development Environment are set of tools helpful while working with functions and files. Examples of this part can be command window, the workspace, notepad editor and very extensive help mechanism. Second part is The Matlab Mathematical Function Library. This is a wide collection of elementary functions like sum, multiplication, sine, cosine, tangent, etc. Besides simple operations, more complex arithmetic can be calculated, including matrix inverses, Fourier transformations and approximation functions. Third part is the Matlab language, which is high-level array language with functions, data structures and object-oriented programming features. It allows programming small applications as well as large and complex programs. Fourth piece of Matlab System is its graphics. It has wide tools for displaying graphs and functions. It contains two and three-dimensional visualization, image processing, building graphic user interface and even animation. Fifth and last

part is Matlab's External Interfaces. This library gives a green light for writing C and Fortran programs, which can be read and connected with Matlab.

3.2 Data representation

Data representation in Matlab is the feature that distinguishes this environment from others. Everything is presented with matrixes. The definition of matrix by MathWorks is a rectangular array of numbers. Matlab recognizes binary and text files. There is couple of file extensions that are commonly used, for example *.m stands for M-file. There are two kinds of it: script and function M-file. Script file contains sequence of mathematical expressions and commands. Function type file starts with word *function* and includes functions created by the user. Different example of extension is *.mat. Files *.mat are binary and include work saved with command File/Save or Save as (Mrozek & Mrozek, 2001, 64-65).

Since Matlab stores all data in matrixes, program offers many ways to create them. The easiest one is just to type values. There are three general rules:

- the elements of a row should be separated with spaces or commas;
- to mark the end of each row a semicolon ';' should be used;
- square brackets must surround whole list of elements.

After entering the values matrix is automatically stored in the workspace (MathWorks, 2002, chapter 3.3). To take out specific row, round brackets are required. In the 3x3 matrix, pointing out second row would be (2,:) and third column (:,3). In order to recall one precise element bracket need to contain two values. For example (2,3) stands for third element in the second row. Variables are declared as in every other programming language. Also arithmetic operators are represented in the same way – certain value is assigned to variable. When the result variable is not defined, Matlab creates one, named Ans, placed in the workspace. Ans stores the result of last operation. One command worth mentioning is *plot* command. It is responsible for drawing two dimensional graphs. Although this command belongs to the group liable for graphics, it is command from basic Matlab instructions, not from Image Processing toolbox. It is not suitable for processing images, therefore it will not be described.

Last paragraph considers matrixes as two-dimensional structures. For better understanding how Matlab stores images, three-dimensional matrixes have to be explained. In three dimensional matrixes there are three values in the brackets. First value stands for number of row, second value means column and third one is the extra dimension. Similarly, fourth number would go as fourth dimension, etc. The best way to understand it, is to look at Figure 3, which presents the method of pointing each element in this three dimensional matrix.

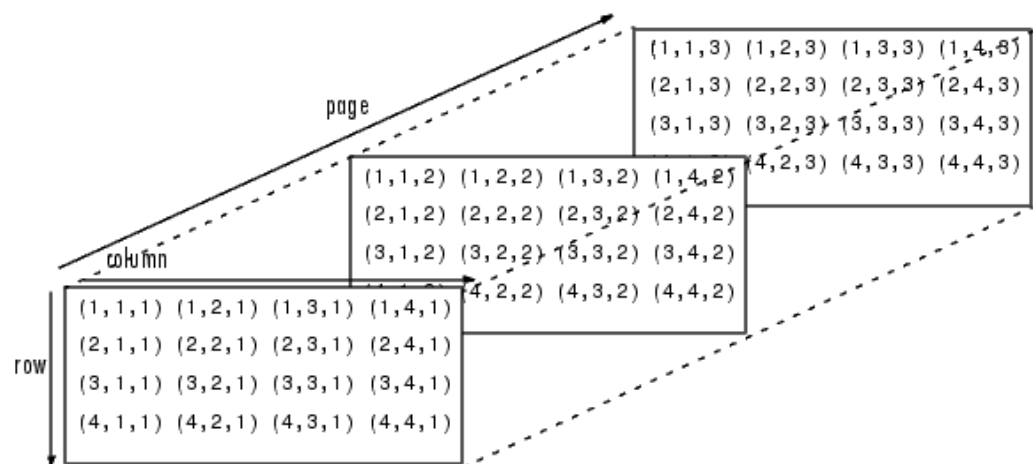


Figure 3. The example of three dimensional matrix, built in Matlab (Ozimek, lectures from Digital image processing, 2010a)

As mentioned before, Matlab stores images in arrays, which naturally suit to the representation of images. Most pictures are kept in two-dimensional matrices. Each element corresponds to one pixel in the image. For example image of 600 pixels height and 800 pixels width would be stored in Matlab as a matrix in size 600 rows and 800 columns. More complicated images are stored in three-dimensional matrices. Truecolor pictures require the third dimension, to keep their information about intensities of RGB colors. They vary between 0 and 1 value (MathWorks, 2009, 2.12).

The most convenient way of pointing locations in the image, is pixel coordinate system. To refer to one specific pixel, Matlab requires number of row and column that stand for sought point. Values of coordinates range between one and the length of the row or column. Images can also be expressed in spatial system coordinates. In that case positions of pixel are described as x and y. By default, spatial coordinates corres-

pond with pixel coordinates. For example pixel (2,3) would be translated to $x=3$ and $y=2$. The order of coordinates is reversed (Koprowski & Wróbel, 2008, 20-21).

3.3 Endless possibilities

As mentioned earlier, Matlab offers very wide selection of toolboxes. Most of them are created by Mathworks but some are made by advanced users. There is a long list of possibilities that this program gives. Starting from automation, through electrical engineering, mechanics, robotics, measurements, modeling and simulation, medicine, music and all kinds of calculations. Next couple of paragraphs will shortly present some toolboxes available in Matlab. The descriptions are based on the theory from Mrozek&Mrozek (2001, 387 – 395) about toolboxes and Mathworks.com.

Very important group of toolboxes are handling with digital signal processing.

Communication Toolbox provides mechanisms for modeling, simulation, designing and analysis of functions for the physical layer of communication systems. This toolbox includes algorithms that help with coding channels, modulation, demodulation and multiplexing of digital signals. Communication toolbox also contains graphical user interface and *plot* function for better understanding the signal processing. Similarly, Signal Processing Toolbox, deals with signals. Possibilities of this Matlab library are speech and audio processing, wireless and wired communications and analog filter designing.

Another group is math and optimization toolboxes. Two most common are Optimization and Symbolic Math toolboxes. The first one handles large-scale optimization problems. It contains functions responsible for performing nonlinear equations and methods for solving quadratic and linear problems. More used library is the second one. Symbolic Math toolbox contains hundreds of functions ready to use when it comes to differentiation, integration, simplification, transforms and solving of equations. It helps with all algebra and calculus calculations.

Small group of Matlab toolbox handles statistics and data analysis. Statistics toolbox features are data management and organization, statistical drawing, probability computing and visualization. It also allows designing experiments connected with statistic

data. Financial Toolbox is an extension to previously mentioned library. Like the name states, this addition to Matlab handles finances. It is widely used to estimate economical risk, analyze interest rate and creating financial charts. It can also work with evaluation and interpretation of stock exchange actions. Neural Networks Toolbox can be considered as one of the data analyzing library. It has set of functions that create, visualize and simulate neural networks. It is helpful when data change nonlinearly. Moreover, it provides graphical user interface equipped with trainings and examples for better understanding the way neural network works.

Some toolboxes do not belong to any specific group but they are worth mentioning. For example Fuzzy Logic Toolbox offers wide range of functions responsible for fuzzy calculations. It allows user to look through the results of fuzzy computations. Matlab provides also very useful connection to databases through Database Toolbox. It allows analyzing and processing the information stored in the tables. It supports SQL (Structured Query Language) commands to read and write data, and to create simple queries to search through the information. This specific toolbox interacts with Oracle and other database processing programs. And what is most important, Database Toolbox allows beginner users, not familiar with SQL, to access and query databases.

Last but not least, very important set of libraries – image processing toolboxes. Mapping Toolbox is one of them, which is responsible for analyzing geographic data and creating maps. It provides compatibility for raster and vector graphics which can be imported. Additionally, as well two-dimensional and three-dimensional maps can be displayed and customized. It also helps with navigation problems and digital terrain analysis.

Image Acquisition Toolbox is a very valuable collection of functions that handles receiving image and video signal directly from computer to the Matlab environment. This toolbox recognizes video cameras from multiple hardware vendors. Specially designed interface leads through possible transformations of images and videos, acquired thanks to mechanisms of Image Acquisition Toolbox.

Image Processing Toolbox is a wide set of functions and algorithms that deal with graphics. It supports almost any type of image file. It gives the user unlimited options

for pre- and post- processing of pictures. There are functions responsible for image enhancement, deblurring, filtering, noise reduction, spatial transformations, creating histograms, changing the threshold, hue and saturation, also for adjustment of color balance, contrast, detection of objects and analysis of shapes. Some of those and more functions will be described in details in the next chapter.

4 IMAGE PROCESSING TOOLBOX

4.1 Color transformation functions

As mentioned previously, this chapter will describe in details some of the Image Processing Toolbox functions. For ability to distinguish them from the text, they will be written *in italics*. For better understanding parameters that all functions take, they will be shown in the brackets next to the name of given function. ‘A’ will mean exemplary input image. All descriptions are based on the website www.mathworks.com, which provides wide compendium of knowledge about all Matlab functions, including those from Image Processing Toolbox. First group of operations is responsible for changes and information concerning color transformation of images.

Couples of functions do not change anything in the picture but they are crucial when it comes to gain information about it, without need of opening the actual object of interests. *Isbw(A)* returns value 1 if the image is black&white, and value 0 otherwise. Some operations have sense only when executed on binary graphic files. For example adjusting contrast, brightness or other changes, usually made on colorful pictures, would not work with black&white images. Function *isgray(A)*, similarly to previous one, checks colormap of the image. As the name suggests, this time function returns value 1 if the picture is grayscale and value 0 otherwise. It may also become useful while deciding if some operations can be performed on the file. *Isrgb(A)* informs if examined file is the RGB image. These three functions are essential when it comes to deciding about changing the colormap or color system. Knowing if the picture is black&white, grayscale or RGB determines what transformations can be done to the file. There would be no point in trying to make some changes to the image, if they are inoperative for some color models or maps.

Command *colormap(map)* is connected with the previously mentioned, however, it is not Image Processing Toolbox function. It exists in Matlab main library. It sets current image colormap to one that stands in the brackets as a parameter. There is about twenty ready-built colormaps in Matlab. Some interesting examples of them are: hsv, jet, gray, hot and bone. Hsv stands for hue-saturation-value colormap. It starts from red and goes through yellow, green, cyan, blue, magenta and comes back to red. It is very often used to display periodic functions in Matlab. Jet is a variation of hsv but it starts

with dark blue and goes through cyan, green, yellow and red. Parameter gray changes colormap to shades of gray. Next one, called 'hot' ranges colors from black, red, orange, yellow to white. Bone parameter is similar to grayscale but it contains tinge of blue, for more 'electronic look-like' effect. Both bone and hot color tables are used in medical diagnostics (Cracow University of Technology,[07.04.2010]). Other color-maps can be seen in Figure 4, below.



Figure 4. Ready-built colormaps from Matlab's Image Processing Toolbox (www.mathworks.se [referred 14.04.2012])

There are three more functions connected with changing colormap of image. *Im2bw* produces black&white picture from grayscale, indexed or RGB file. There is couple of possible ways of defying parameters. To convert grayscale to binary graphics it is enough to put in the brackets name of file as a parameter. Optionally, there is a place after the comma for level of threshold that will be used while conversion. Default value is 0.5 which stands for average density of threshold. This constant must range between zero and one. If the input picture is RGB, *im2bw* converts it first to gray shades and then to black&white. When dealing with indexed image, user can also put colormap of it in the brackets, just after name of the file. Similar operation existing in the toolbox is *rgb2gray*. It converts RGB image to grayscale by eliminating the hue and saturation from it. Two of standard Matlab library function can be also considered as image processing operation. *Rgb2hsv* and *hsv2rgb* are responsible for changing colormap from RGB to hsv and vice-versa. Each map is a matrix with some number of rows and three columns. In the RGB image those columns represent intensity of red, blue and green color and respectively in hsv image, hue, saturation and color value.

Another smaller group of functions are those responsible for picture enhancements. *Imadjust* adjusts image intensity values. As an additional parameter user is allowed to specify two squared brackets ranges. Pixels that do not belong to those ranges are clipped. That is how this procedure increases contrast of the input image. Other function that is responsible for contrast changes is *imcontrast*, which creates ready-built contrast adjustment tool. It takes opened picture as an object of contrast customization. Unfortunately, this tool works only with grayscale images.

A very useful function of the main Matlab library is *brighten* command. It brightens or darkens image opened in the axes. Parameter of this procedure has to differ from between minus one and one. For all values from -1 to 0, it tones down the colormap and correspondingly, from 0 to 1, lightens it. Since zero makes no changes, it is excluded from both ranges. *Brighten* command works for all types of colormaps.

Last function described here will be *roicolor*. It selects specific region of interest, based on a one or more colors. It takes three values as parameters. First is the name of file, then after the comma lower value of color and after that, higher value of color. The result will be a bitmap, containing white and black areas. If color on original picture was between two values given as parameters, area will be white. Two the same values will mark only one color area. It is very important to know colormap of processed image. Parameters will vary from 0 to 255. Each number points different color, which is dependable from colormap.

4.2 Spatial transformation functions

Spatial transformation functions are separate group that is responsible for all changes concerning size, rotating and cropping an image. A simple and effective command is *imresize*. It takes two arguments in the round brackets – the name of the picture and after the comma, a value that stands for multiplier. If this number is between zero and one, then the result image is smaller. Respectively, if this constant is greater than one, therefore the output picture is bigger than the original.

Image Processing Toolbox offers a function that rotates pictures. It is called *imrotate* and it usually takes two arguments inside the brackets. First is the name of the file in the apostrophes and the second, is the angle of rotation. Positive values stand for

counterclockwise direction of rotation, therefore negative numbers go as clockwise oriented turning. There is third additional parameter, used to determine if the modified picture should stay the same size or should it be cropped to the size of original one. First option can be gained with putting a word 'loose' on the third place in the brackets. Text 'crop' will make the output image the exact size as the input one. If this parameter is not specified by the user, default value leaves picture non-cropped.

There is an independent function responsible for cropping operation. *Imcrop* cuts the image to the selected rectangle. A user defines the area with mouse and as a result, cropped image is displayed in a new figure, if not specified differently. Holding keyboard button Shift down instead of rectangle, picked area will be a square. Matlab's Image Processing Toolbox offers wide range of functions, designed to deal with spatial transformations but the most important ones are described above.

Matlab's main library offers two additional functions, which are worth mentioning. *Fliplr* flips the image along vertical axis. The same way, *flipud* flips picture along horizontal axis. The only disadvantage is, that both commands are defined to work with two-dimensional matrixes. Therefore only bitmaps and grayscale pictures can be processed by this procedure.

4.3 Open, Save, Display functions

This group of Image Processing Toolbox handles basic operations like opening, closing, displaying and saving the image file. In addition Matlab's library contains couple of useful commands. *Imread* deals with reading image from graphics file. As a parameter in the brackets it takes the name of the file and its extension. Among supported formats are bmp, gif, jpeg, png and tiff. *Imread* returns a two-dimensional array if the image is grayscale and a three-dimensional one, if the picture is color. The function mentioned above, allows also reading an indexed image and an associating colormap with it. In order to do that, instead of giving one variable as a result, user needs to put second variable that will stand for the map, just after the comma in square brackets.

Complementary, function *imwrite* writes the image to the graphics file. It supports the file formats as mentioned before, with *imread* command. Each file extension has its own syntax but there is one simple that works for most of them. It takes three parame-

ters in the brackets: first - the array with the image, second – the name of the new file, third – the file format. Interesting part is the specification of this function, which depends from the file extension. Usually after all obligatory parameters, there is a place for extra ones, after the comma. For example for a gif file format optional criterions can be ‘TransparentColor’, which specifies the color that will be treated as the transparent one. ‘DelayTime’ sets the delay in seconds between images in case of gif animation. ‘LoopCount’ defines the number of times to repeat the animation. All additional parameters are followed by the values. For a jpeg file possible attributes are ‘Mode’ and ‘Quality’. In the first one values are either ‘lossy’ or ‘lossless’, which indicates the method of compression. As it comes to quality, it’s a number between 0 and 100 which saves the image in specific size – higher the number is – higher the quality and the size of the file. All optional parameters, for other file formats can be found in the Matlab documentation.

A very useful function exists in Image Processing Toolbox. *imshow* is responsible for displaying an image. It works with black&white, grayscale and color pictures. Simply, matrix that includes a graphic file or just a filename can be treated as a parameter for this procedure. Alternatively, the image can be displayed with its colormap. Map should be given after the comma in the brackets, along the filename. The shown picture has to be in the current directory or specified by the path to the file.

4.4 Other functions

Last paragraph of this chapter will describe miscellaneous functions from Image Processing Toolbox and Matlab’s main library. Often used *imfinfo* displays various information about the image. Among all the data fields, returned by this procedure there are nine of them that are the same with every file format. Those are:

- Filename – contains name of the image;
- FileModDate – last date of modification;
- FileSize – an integer indicating the size of the file, in bytes;
- Format – graphic file extension format;
- FormatVersion – number or string describing the file format version;
- Width – width of the image in pixels;
- Height – height of the image in pixels;
- BitDepth – number of bits per pixel;

- `ColorType` – indicates type of the image, either ‘truecolor’ for RGB image, ‘grayscale’ for grayscale image or ‘indexed’ for an indexed image.

Image Processing Toolbox function *impixel* may become helpful when pixel color values (red, green and blue) are required. Normal syntax of this procedure displays the image and waits for the user to specify the pixels with the mouse. Pixels can be determined also non-interactively. *Impixel* takes three parameters in that case – first one is a matrix containing the image, second and third one are numbers of coordinates of the selected pixel. Its colors are returned to the workspace variable `Ans`.

Small function, *imcontour* handles only grayscale graphic files. It draws a contour plot of the image data. The least complex syntax of this procedure requires a two-dimensional input matrix, which contains the grayscale image. Different command that handles only black&white or grayscale files – *imfill*, fills holes in the picture. The construction of this function allows user to select regions to flood-fill interactively, with a mouse. Optionally, useful parameters may be string ‘holes’, placed after matrix containing a black&white image. As a definition of a hole Matlab defines small dark areas, surrounded by light pixels.

Very complex function – *fspecial*, creates predefined filters that can be used while processing the image. As a parameter it takes one value, which states the type of the filter. More interesting possibilities of them might be ‘disk’, ‘motion’ and ‘unsharp’. Value ‘disk’ returns a circular averaging filter with a radius specified by the user. The default radius is 5. The result of using this filter is the picture becoming blurred. ‘Motion’ filter answers with linear motion of a camera. User can determine the amount of pixels moved and the angle of motion. Default values returns the picture as it was blurred by 9 pixels of horizontal camera movement. Complementary, parameter ‘unsharp’ is responsible for sharpening blurred image. Although the name states differently, this filter is mostly used to acuminate smudged picture. Figure 5 shows comparison of original photography with the ones processed in Matlab, by function *fspecial* (first picture is original, second one motion blurred and third one sharpened).



Figure 5. Comparison of photographs processed with Image Processing Toolbox

Every filter specified by function *fspecial* can be used only with help of *imfilter* command. This particular operation applies change to the file, resulting with the same-sized output image. It is important to remember that those two commands works together and are commonly used while filtering graphic files.

5 MATLAB "GUIDE" TOOL

5.1 User friendly graphical interface

According to Galitz (2002, 15, 41 - 51), a graphical user interface can be defined as set of techniques and mechanisms, used to create interactive communication between a program and a user. The author of the book underlines the importance of designing process by presenting essential rules. Proper visual composition is a must. The aim is to give the user aesthetically pleasant working environment. Colors, alignment and simplicity of look should be considered carefully. Every function, button or any other object should have its meaning, simple and understandable by an average program user. Similar components should have analogous looks and usage. Functions ought to perform quickly and result with wanted outcome. Flexibility can be perceived in this topic as being sensitive to each user's knowledge, skills, experience, personal performance and other differences that may occur. A good interface is simple, limits the number of actions and do what it is expected to do. It is not an easy task to design an efficient and user-friendly graphical interface.

Luckily, Matlab provides a helpful tool called 'GUIDE'. After typing *guide* into Matlab's command line, a quick start window appears. From the choice of exemplary positions it is recommended to pick 'Blank GUI'. In the new window it is possible to drag and drop each object into the area of the program. On the left side of the created figure there is a list of possible components. The list includes a push button, slider, axes, static and edit texts – which will be described in details in the next paragraph. It also contains objects that will be briefly explained below (solely based on Mathworks.com):

- Toggle Button – once pressed stays depressed and executes an action, after the second click it returns to the raised state and performs the action again;
- Check Box – generates an action when checked and indicates its state (checked or not checked), many options might be ticked in the same time;
- Radio Button – similar to the check box, but only one option can be selected at any given time, function starts working after the radio button is clicked;
- Listbox – displays a list of items and enables user to select one or more from them;
- Pop-up Menu – open a list of choices when the arrow is pressed;

- Panel – groups all components what makes interface easy and understandable, positions of all objects are relative to the panel and do not change while moving the whole panel;
- Button Group – similar to the panel but able to manage specific behavior of radio and toggle buttons that are logically grouped;
- ActiveX Component – allows displaying ActiveX controls that are interactive technology extensions of html. They enable sound, Java applets and animations to be integrated in a Web page.

An example of GUI with random components is presented in Figure 6.

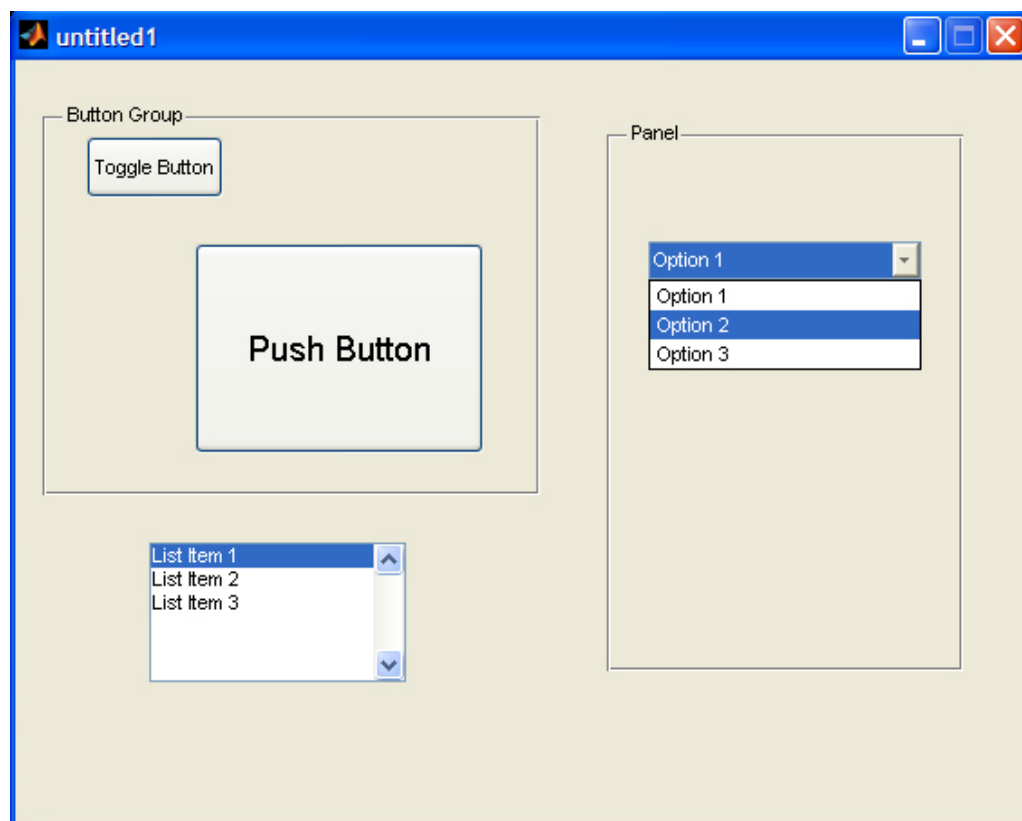


Figure 6. Example of graphical user interface with some of the components

After the first time saving, GUIDE stores the interface in two files- .fig file, where the description of whole graphic part is placed and .m file, where the code that controls the actions can be found. Each object properties are kept in the .fig file and can be set directly from GUIDE tool, thanks to ready-built Property Inspector. All actions, usually called 'callbacks' can be modified and changed in the .m file. Every single component has 'Tag' property, which is used while creating the name of the callback refer-

ence. To get access to each attribute, Matlab offers command *set*. It requires reference to the object that is about to be changed and the name of the property, followed by its value. Among other characteristics, there is an action trigger - callback operation. It is important to know, that any element can have its own specific implementation of this function. Besides operations responsible for actions of objects, there are two additional functions implemented in *.m* file:

- Opening function – executes tasks before the interface becomes visible to the user;
- Output function – if needed, it returns variables to the command line.

There is much more behind mechanisms and techniques of programming GUI but this topic will be explained closely in the next chapter.

5.2 Main components of GUI

5.2.1 Common knowledge

All operative user interface components of Matlab GUI are called ‘uicontrols’. They all contain various selections of properties to be set. After a programmer double-clicks an object created in GUIDE, a window of Property Inspector appears. It is a list of all changeable traits of the component, represented by Figure 7, below.

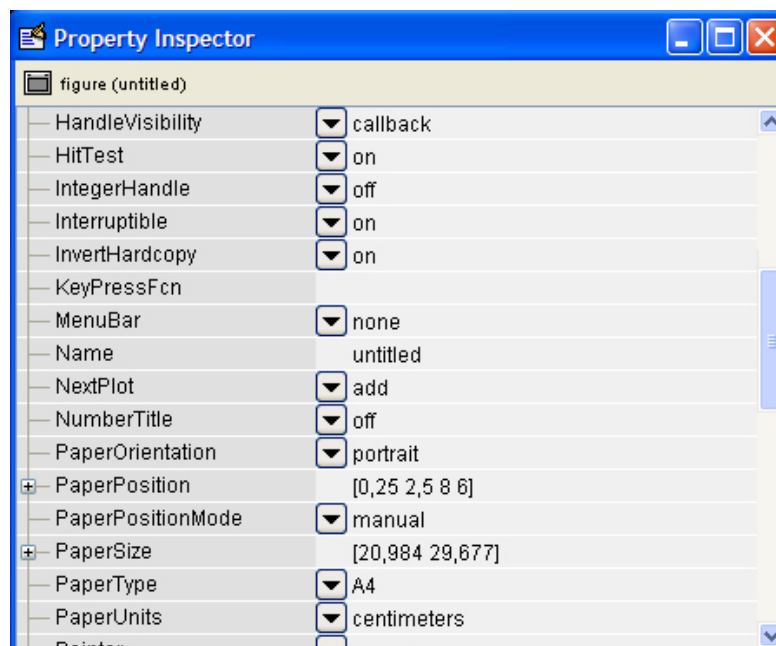


Figure 7. Property Inspector

Most of GUIDE controls have common properties, responsible for the same characteristics of a component. In addition every object has several supplementary features. Each attribute can be queried with command *get* and changed by command *set*, as mentioned before.

First group of attributes is responsible for control of visual style and appearance. ‘Backgroundcolor’ defines color of the rectangle of the uicontrol. Similarly, ‘ForegroundColor’ sets tinge of the string that figures on the button. Important field ‘CData’ allows to put a truecolor image on the button instead of the text. Parameter ‘String’ places given word on the button. Line ‘Visible’ can take either on or off value, the object can be visible or not. Even not seen, it still exists and allows getting all the information about it.

Next collection of properties concerns information about the object. ‘Enable’ defines if the button is on, off or inactive. Option ON states that uicontrol is operational. Respectively, alternative OFF, states disability of proceeding any action on the button. In this case label is grayed out. Selecting inactive value allows showing component as enabled, but in real, it is not working. The kind of uicontrol is decided by ‘Style’ field. Possible values of this parameter are: pushbutton, togglebutton, radiobutton, checkbox, edit, text, slider, listbox and popupmenu. Every created object has its name, stored in ‘Tag’ property. It assists in maintaining the application and navigates among the components. Another useful attribute is ‘TooltipString’. Every time a user rolls a mouse over the uicontrol and leaves it there, a text set in this place is shown. Those small hints can be helpful in case object is not completely understandable. Last feature from this group is ‘UserData’. It allows connecting any data with the component and can be reached with *get* function.

Third category deals with positioning, fonts and labels. ‘Position’ parameter is responsible for placement of the object. It requires four values which are: the lower left corner of the component (distance from the corner of the figure) and its height and width. ‘Units’ field is used by Matlab for measurements and interpretation of distance. Attainable values can be inches, centimeters, points, pixels and characters. Pixels are default setting. There is couple of font properties. With them a programmer can decide ‘FontAngle’ (normal, italics or oblique), ‘FontName’ (font family), ‘FontSize’ and

'FontWeight' (light, normal, demi or bold). Parameter 'HorizontalAlignment' determines the justification of the text of the 'String' property. Possibilities to set are left, right and center.

Last group of properties considers all actions performed by the application. Attribute 'ButtonDownFcn' executes callback function whenever a user presses the mouse button while the pointer is near or in five pixel-wide border around the component. There is a field named 'Callback' containing a reference to either M-file or valid Matlab expression. Whenever an object is activated, a callback function will be executed. Two next features – 'CreateFcn' and 'DeleteFcn' work in the way opposite to each other. First one specifies a callback routine that performs action when Matlab creates an uicontrol. Respectively, second trait starts an operation every time uicontrol object is destroyed. This characteristic is definitely an asset, because a programmer can set some actions just before a component will be removed from the application. A more complex field, called 'Interruptible', contains information concerning actions triggered by the user, during executing of one of callback functions. This property can take on or off value. In the first case, Matlab will allow second operation to interrupt first one. Accordingly, if off is the selected option, the main callback will not be interfered.

There are properties important only for particular uicontrols. Next four paragraphs will briefly describe some of the components and their additional features.

5.2.2 Buttons and Sliders

Push buttons are important components because they allow a user to interact with the program on a visual and simple level. Usually buttons are suggestive and they convey their main purpose. When it comes to sliders, they are not less valuable than buttons. Thanks to sliders, users can change for example brightness or contrast of the image, with some certain steps. Field 'Style' takes argument pushbutton or slider, dependable from the type of uicontrol. There are four parameters, connected together. 'Min' and 'Max' specify the minimum and maximum slider values. Defaults are 0 for minimum and 1 for maximum. Matlab will not allow defining the lowest number bigger than expected utmost numeral. Using both properties, 'SliderStep' attribute can be determined. As the name suggest, this characteristic calculates the size of the step which

a user may modify, by clicking arrows on this component. The step of the slider is a two element vector. By default it equals the bracket [0,01 0,1], which sets one percent change for clicks on the arrow button and ten percent modification for clicks in the middle. Also feature ‘Value’ relies on previous numbers. It is set to the point, indicated by the slider bar and a programmer can access it with *get* function.

Figure 8 shown below, represents exemplary Property Inspector for a slider bar.

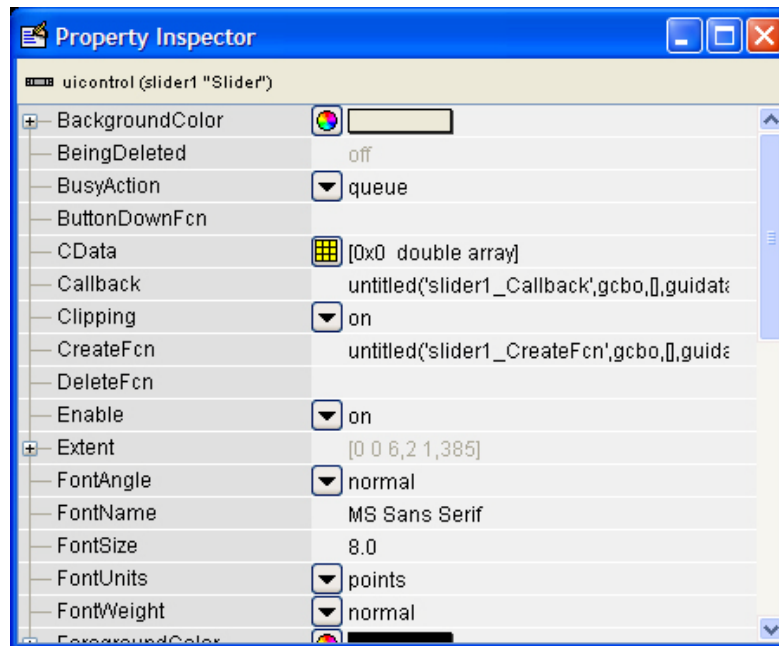


Figure 8. An example of Property Inspector for a slider bar

5.2.3 Axes

Axes component contains several additional attributes. ‘Box’ property defines whether the region of the axes will be enclosed in two – dimensional or three – dimensional area. Options ‘XTick’, ‘XTickLabel’ and ‘YTick’, ‘YTickLabel’ allow a programmer to define what values will be displayed along the horizontal and vertical axis. As a separator, the easiest way is to use this line ‘|’. Also the location of both lines can be set with help of ‘XAxisLocation’ and ‘YAxisLocation’ features. ‘XGrid’ and ‘YGrid’ creates the grid that might be useful while cropping or resizing processed image (Marchand&Holland, 2003, 248-283).

Besides all graphical attributes responsible for outer look of the axes, this object contains also all features common for different components.

A lot of properties will not be described here because they refer to appearance of graphs, drawn with *plot* command, while this paper treats about image processing. Therefore, axes will be used as an area of picture input and display.

Figure 9 illustrates Property Inspector for an interface component - axes.

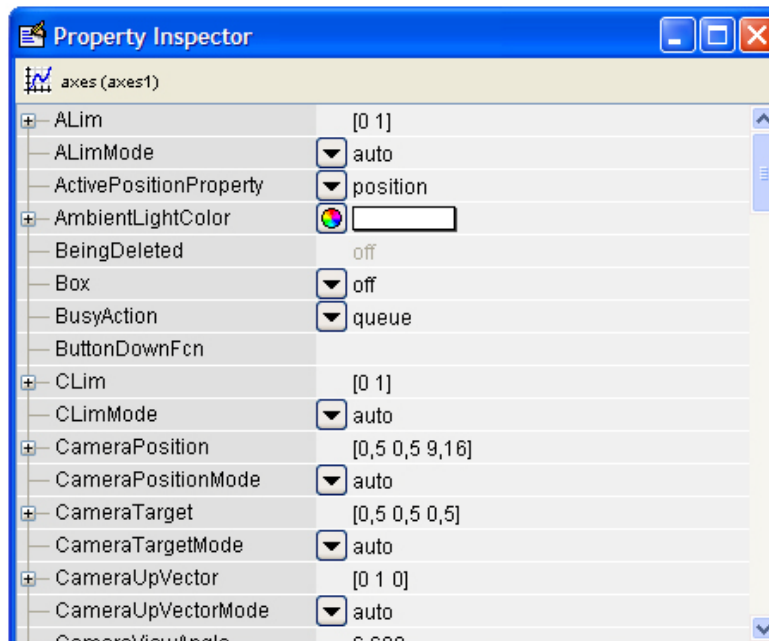


Figure 9. An example of Property Inspector for axes

5.3 Creating menu

Every decent application should have the menu bar. An average computer user is accustomed to possibility of getting most things done with the help of the menu. That is why Matlab enables programmers to create two kinds of menus:

- Menu bar objects – drop-down menus whose titles are situated on the top of the figure;
- Context menu objects – pop-down menus that appear after a user right – click one of the component.

To create both of them, GUIDE offers Menu Editor. They are implemented with two objects – *uimenu* and *uicontextmenu*.

After entering GUIDE Menu Editor it is possible to create a hierarchical menu, without any limitations of items amount. This tool helps programmers on many levels. Process of making menu becomes intuitive and simple. It enables setting of menu properties with Property Inspector, for every menu and submenu element. Creating context menu requires changing the tab into 'Context Menus'. Then the process goes similarly to the menu bar building. There are several properties that can be set just after new menu is generated. 'Label' defines the name of the item that will be displayed to the user. 'Tag' value determines the name, needed to identify the callback function. 'Separator above this item' is responsible for a slim line between logically divided menu elements. Another attribute 'Check mark this item' displays a check next to the menu item and indicates the current state of this item. To ensure that users can select any option, property 'Enable this item' has to be marked. (Marchand&Holland, 2003, 432-440).

Menu Editor is presented in Figure 10, below.

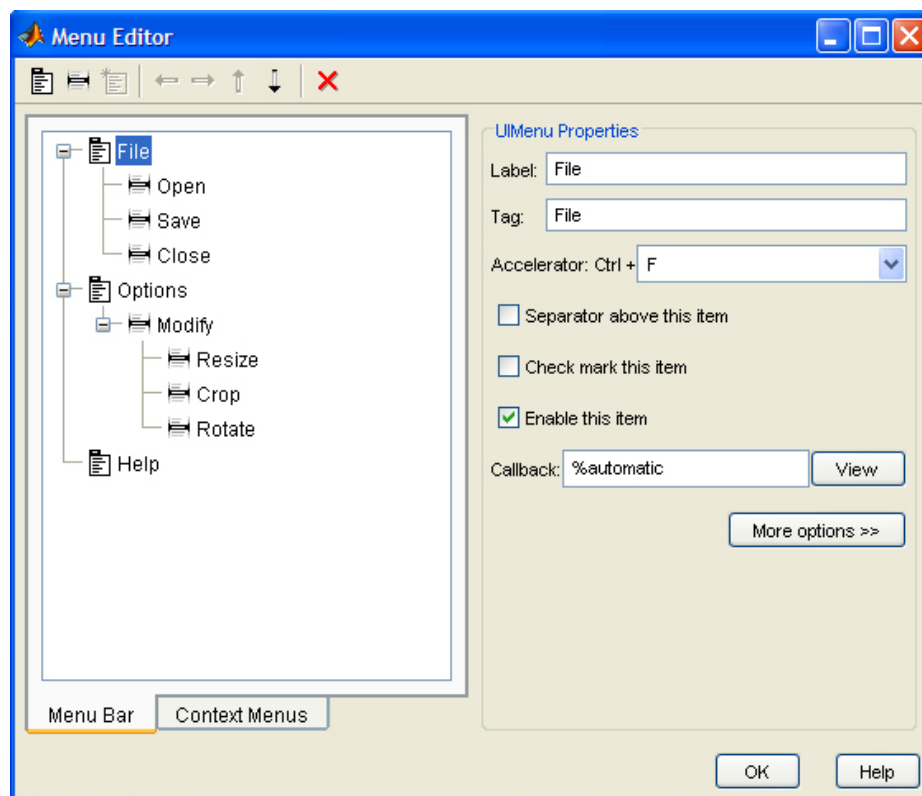


Figure 10. An exemplary menu created in Menu Editor

Next I will describe the properties of the menu. These descriptions are solely based on Marchand&Holland's (2003, 434 – 440) book, chapter 10th.

The ‘Accelerator’ field defines the keyboard equivalent that a user can press to activate particular uimenu object. Presence of the shortcuts is valuable addition to the GUI. Thanks to them the time and effort of action is reduced. Sequence Ctrl + Accelerator selects the menu item. Only items that do not have a submenu can be connected with some shortcut. ‘Callback’ is previously explained reference to the function that performs an action. Whenever a menu item has a submenu, all elements from there are called ‘children’ of the mentioned item. Parameter ‘Children’ lists all submenu elements in a column vector. If there is no ‘children’, the field becomes an empty matrix. Another feature decides if an option is available to the user. If it is not then ‘Enable’ value is set to off. In that case, the name of the menu item is dimmed and indicates that it is not possible to select it. For nicer visual effect, a programmer can change the font color of the menu labels with ‘ForegroundColor’ attribute.

When it comes to the context menu, only one option is responsible for it. ‘UIContextMenu’ as a default, takes ‘none’ parameter. If the context menu was created before, its name should appear in the list of options. After selecting it, a user can enjoy right-click menu for the given component. Figure 11 presents ready- built menu.

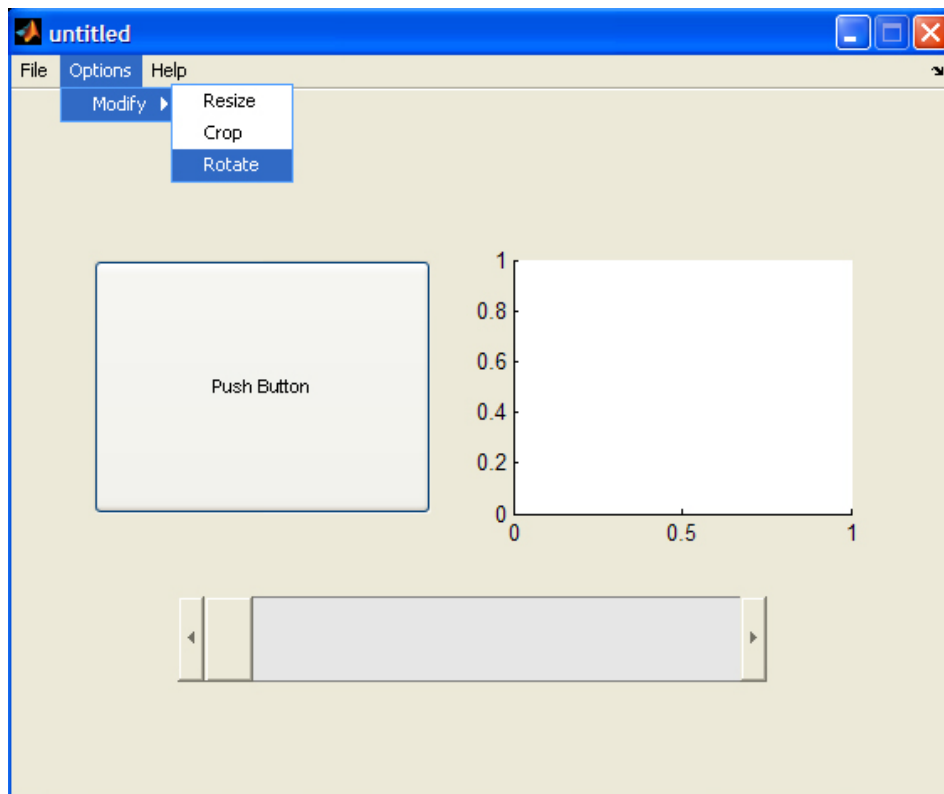


Figure 11. Simple, GUI with ready – built menu

6 DESIGN AND IMPLEMENTATION OF AN IMAGE PROCESSING APPLICATION

6.1 Where to start ?

Every process of creating a computer program should be preceded with careful consideration of the task. In this chapter I will describe how to implement an image processing application with a Graphical User Interface using Matlab. As mentioned in earlier stages of the thesis, the program should be able to perform basic operations on various images. Rotation, cropping, changing the colormap, blurring, sharpening are only couple of examples of program's functionality.

Particular steps taken towards achieving the goal are in order: design of the window together with the procedure of placing the elements of GUI, construction of the menu bar. Ready prototype of the application will be tested against some scenarios with regard to the common usage examples. So how to begin this interesting journey of software implementation ?

6.2 Designing the window

It is a good practice to start the window design with sketching a draft. With the project on paper is it easier to imagine and plan functions, buttons and all other elements of the application. Matlab will help in creating the program on the computer. After typing *guide* command, the choice window appears. There are two tabs, one stands for creating new a GUI and the other one for opening an application that already exists. From the first tabs is it possible to pick either some template or a blank project. To start designing from the beginning, a blank GUI should be selected. Figure 12, shown below, presents the quick start window.

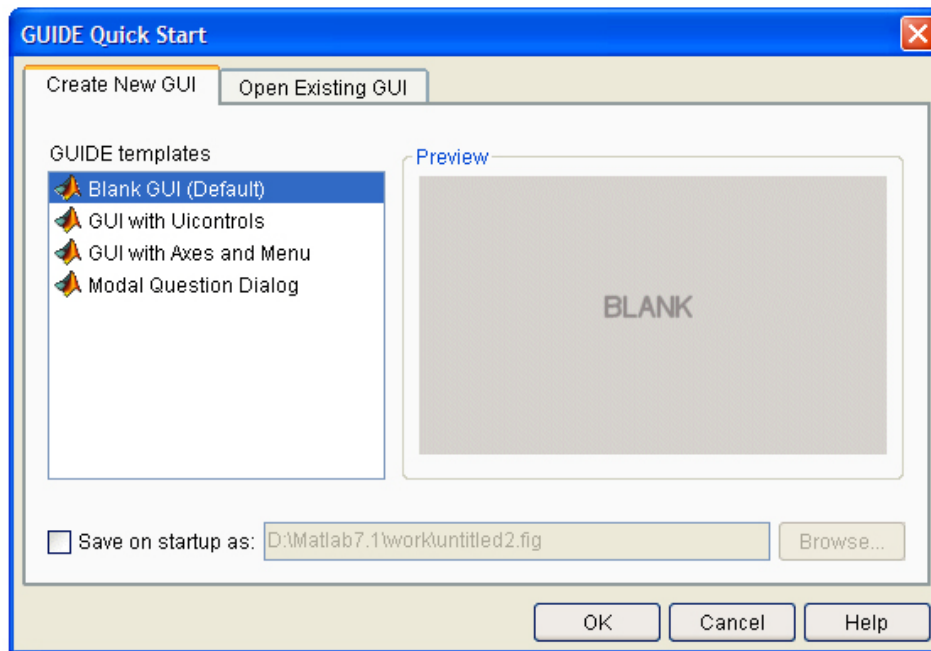


Figure 12. GUIDE Quick start window

Creating a new application window requires well thought draft. All components need to be positioned in logical and functional way. Menu items have to be understandable for the user. Chaos and mess should not be present in the layout of the program.

First thing is adjusting the size of the window. In the Property Inspector, in the field ‘position’, two last values stand for width and height, respectively. Two reminding numbers set position of the figure on the screen. Next thing determines if users are allowed to resize the window. For the safety reasons, resizing is not an expected action.

Thinking about the purpose of the program, axes in which image will be displayed are second created component. Background color was changed to the same color as the figure background, so if there is no picture loaded, white unused area is not seen. Axes ticks were deleted because they are mainly used for various kinds of plots – not for photographs processing. Thin, black border shows the area, where image will be loaded.

Brightness of the image can be regulated with next element that is placed on the right side of the axes. Buttons ‘+’ and ‘-’ will allow quick and convenient method of changing luminosity of the picture. Static Textbox positioned above will tell the user what is the purpose of the buttons. Second static textbox was designed to show the user in-

formation, which he/she would like to withdraw from the image. At the end of designing GUI one button was created: restoring function. It will be responsible for loading the original picture to the axes in case the user did not like the effect of transformations. In all components the font was changed to the Arial type, size of 12 and dark blue color. The outlook of the application window after the first step of designing process is presented in Figure 13.

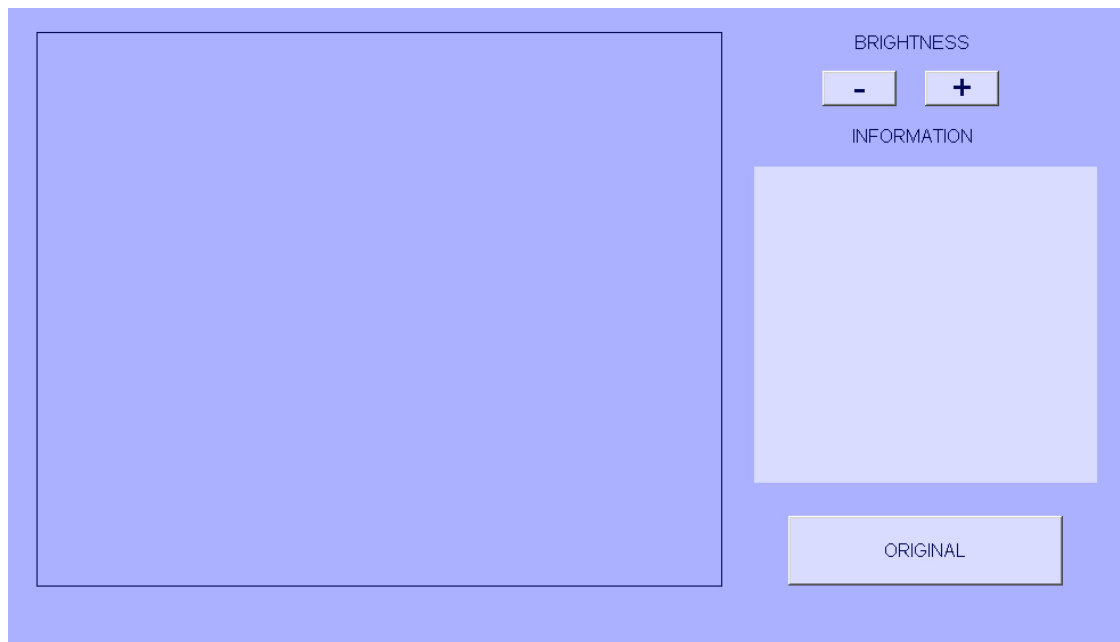


Figure 13. First step of designing the application window

Very important part of designing an application is creating a menu. Menu bar for this program will contain: File, Transformations, Filters and Help headings. Each heading will be briefly described in next paragraphs.

6.2.1 Menu File

Menu item 'File' contains five elements, which are: Open, Save, Save with compression, Info about the file and Exit. It is necessary to change each element's label and tag property. The reason is enabling easier maintenance within the components. Also setting the 'Accelerator' field will make the application more familiar to users. Accelerator parameter is responsible for a shortcut to the function. Well known combinations are ctrl+o for opening a file, ctrl+s for saving the picture, ctrl+I for image information and ctrl+q for quitting the application. More of them may be created to make

the program easier to use for an experienced user. Shortcut keys are displayed next to the menu options. Subheadings have the same names as labels or better recognition. For example subheadings for File heading have 'Open', 'Save', 'Exit' tags. Those tags menu items will be connected with a proper function that will execute an expected action. Figure 14 presents menu item 'File' creation.

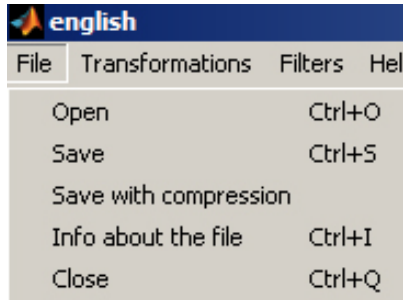


Figure 14. First part of menu bar designing

6.2.2 Menu Transformations

The outlook of the Transformations menu heading is presented on the diagram below.

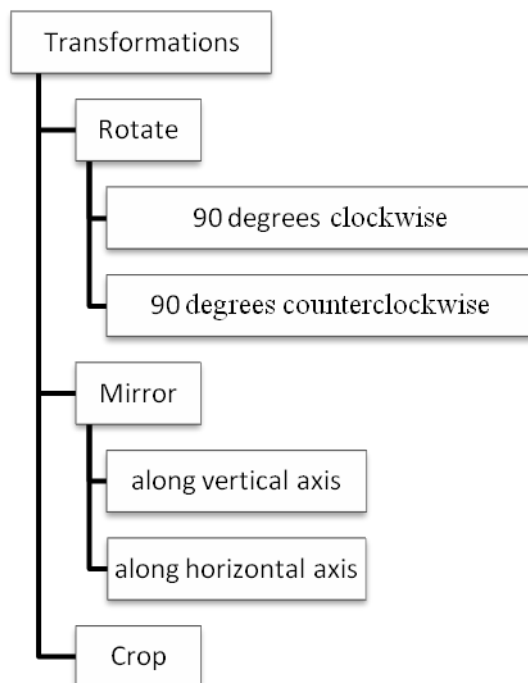


Figure 15. Second part of menu bar designing

As shown in Figure 15, ‘Transformations’ menu heading will allow a user to rotate, flip and crop the image. There are two options for the rotation angle – 90 degrees clockwise and 90 degrees counterclockwise. It will be available to flip the picture either vertically or horizontally, however, this option will become enabled only if the picture is black&white or in grayscale. As explained in previous section, functions *fliplr* and *flipud* are operable only on those kinds of pictures.

6.2.3 Menu Filters and Help

Two headings considered in this paragraph are ‘Filters’ and ‘Help’. First one includes three different ways to blur an image, sharpening, adding and removing noise along with circulating effect of the picture. All those operations are based on *imfilter* and *fspecial* functions, explained previously. It is worth mentioning that option ‘add noise’ is created mostly to show the functionality of Matlab and function that removes the noise from the image. Next option is a histogram equalization and correction of contrast. It is also possible to convert a color picture into the grayscale or make it a bit-map.

Last main menu item is ‘Help’. It is not that important for the application to work but often users want to know facts about the program, the author and how it actually works. Figure 16 shows fully designed menu bar.

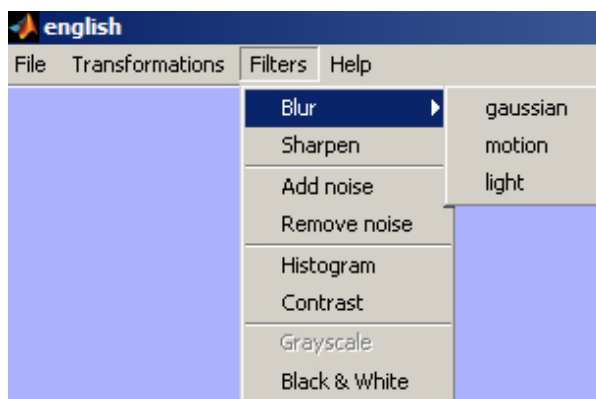


Figure 16. Finished menu bar, some options presented

6.3 General rules of coding

After creating the graphical user interface it is time to connect it with Matlab functions. There are two ways of doing it: the first one demands from a programmer cod-

ing in .m file, where he has to create all the components. What is more, properties for each element must be set using only text. This is not the fastest method. Thankfully, GUIDE mechanism comes in handy here. It is enough to drag and drop the objects into the workspace and set their parameters with Property Inspector, mentioned earlier. With this option the application will need an additional .fig file that contains the interface. The technique used in this thesis mixes both – visual creation of the GUI along with programming inside the .m file.

Every application needs to be very carefully thought through. The programmer has to remember about user mistakes as well as unexpected and unusual order of actions. Next paragraph will present how the problem of limits and constraints was handled.

Just after opening the application, the buffer of the image is empty therefore no operations can be performed. First adjustment was to disable menu options until the picture is loaded. Of course options of opening, closing and getting help stay active. During the opening of the image, the type of the input file is checked. Respectively, some of the functions get activated only for grayscale pictures. Next thing is the message box that shows up to inform the user about the type of the image and available processing options. Figures 17 and 18 present both adjustments.

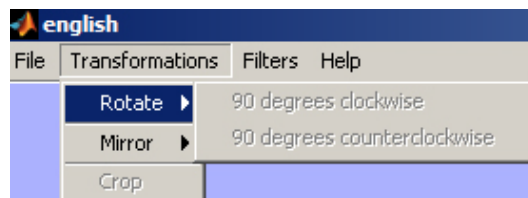


Figure 17. Inactive menu options

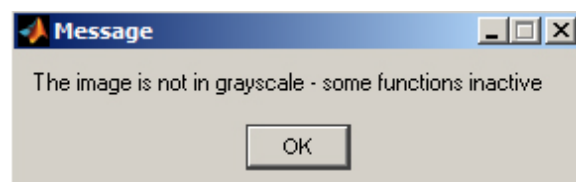


Figure 18. Informative message box

Next I will describe how the functions explained above are actually implemented. In the couple of next paragraphs I will point out the most important parts of the code. Full listing can be found in Appendix 1.

Starting with the procedure of opening the file there are couple of methods used in order to prevent errors. Firstly, the image is loaded into the two variables, one for using and one for restoring. Thanks to the command *handles* followed with a ‘dot’, variable becomes global. Function *uigetfile* allows selecting the file from the computer. Instruction *if* prevents from receiving an error when the user decides to cancel opening. Command *strcat* connects the name of the file with its path. This brings the possibility of using *imread* to load the picture. Later on *imshow* presents the image and all the changes in GUI are updated with *guidata* command. Below are selected fragments of the code.

```
[filename, pathname]=uigetfile(
{'*.jpg';*.gif';*.png';*.bmp';*.tif';*.eps'}, 'Select file');

if filename ==0
    return;
end

handles.var=strcat(pathname,filename);
handles.plik=imread(handles.var);
handles.oryginal = imread(handles.var);

axes(handles.axes);
imshow(handles.plik);

if isgray(handles.plik) == 1

set(handles.histogram, 'Enable', 'On');
guidata(hObject, handles);

msgbox('The image is in grayscale - all fuctions active', 'Message');
```

Another function from the menu bar is saving the file. As discussed previously, it can happen with or without the compression. Statement *uiputfile* along with *imwrite* are a fairly good mechanism of writing files. Compression allows to decrease the size of an image which is presented in the chosen piece of code below.

```
[FileName, PathName]=uiputfile({'*.jpg';*.tif';*.png';*.gif'}, 'Save
image');
imwrite(handles.plik, [PathName FileName], 'Quality', 50);
```

Next functionality worth mentioning is getting some information about the processed picture. *Imfinfo* helps to obtain that goal. Since metadata is stored in a structure type, it is necessary to use ‘dot’ construction to access it. All information are connected together with *strvcat* command, which not only joins the text but also separates them

with one empty line. Later set function embeds that string text into the information field.

```
i = imfinfo(fullfile(handles.pathname, handles.filename));
A1 = i.FileName;
A = strvcat('File Name: ',A1);
C1 = num2str(i.FileSize);
C2 = strcat(C1,'Bajts');
C = strvcat('File Size: ',C2);
I = strvcat(A,C,D,E,F,G,H);
set(handles.text3,'String',I);
```

An example of spatial transformation is presented in the lines below.

```
handles.plik = imrotate(handles.plik,-90,'bilinear','loose');
handles.plik = fliplr(handles.plik);
handles.plik = imcrop(handles.plik);
```

Very important part of the application are filters. Blurring can be acquired with three different methods, described in the previous section. Each filter is based on connecting two powerful commands – *fspecial* and *imfilter*. Adding noise takes a parameter such as ‘salt&pepper’, while removing noise is done with a median filter. Some of the functions changes state of others, meaning their activation mode. There would not be much sense in trying to change grayscale picture into a grayscale, for example. Additionally, histogram equalization option was created in menu. Also a procedure that corrects contrast. Examples of code are illustrated below.

```
H = fspecial('gaussian');
handles.plik = imfilter(handles.plik,H);
```

```
H = fspecial('unsharp',0.05);
handles.plik = imfilter(handles.plik,H,'same');
```

```
handles.plik = imnoise(handles.plik, 'salt & pepper',0.02);
handles.plik = medfilt2(handles.plik);
```

```
handles.plik = histeq(handles.plik);
handles.plik = imadjust(handles.plik);
```

Different type of functions, used in the application are those converting color to grayscale and grayscale to bitmap. Since after transforming image to the black and white it is not possible to regain the color data, most of the options are switched off.

```
handles.plik = rgb2gray(handles.plik);
```

```
set(handles.skala_szarosci, 'Enable', 'Off');  
  
handles.plik = im2bw(handles.plik);  
set(handles.kontrast, 'Enable', 'Off');  
set(handles.skala_szarosci, 'Enable', 'Off');
```

After creating the first prototype of the program, an extra starting window was designed. It allows selecting a language from English and Polish. To achieve expected results *openfig* and *close* commands were used. Parameter for *close* is *gcf*, which stands for handle to the current figure.

```
close(gcf);  
openfig(english);
```

When the user decided that he wants to end image processing, he can close the program from 'File' menu or with a shortcut ctrl+q.

6.4 Testing new application

The next and final step on the way of creating a new application is testing. This stage is very important. It helps to find mistakes or malfunctions of the program. What counts is the amount of tests and also the performed scenarios. Users from a different groups, variety of file formats, individual needs, distinct sequences of actions – all those factors are influential. Thanks to Polish and English language version, more tests from different users could be performed. Every comment was carefully thought out and suggestions for changes appeared. They are discussed in one of the last paragraphs concerning future possibilities of development. For the purpose of the thesis couple of scenarios were selected and presented on the next few figures, along with short descriptions. All images used for testing come from author's own materials.

The firstly tested scenario was a very simple operation of cropping selected part of an image. As presented in Figure 19, this procedure can be successfully used to zoom in a specifically chosen piece of the picture.



Figure 19. An example of *crop* function

Transformation from color to grayscale and then to the bitmap is shown in Figure 20.



Figure 20. Procedures of changing the colormap

The next tested process was adding an artificial noise and then removing it with the median filter. The results is presented in the next figure (Figure 21).



Figure 21. Removal of the noise for the grayscale picture

The last scenario that will be presented in this thesis is mostly about changing the brightness of the image. Firstly, the photography was loaded, then converted to the grayscale. The next two steps demonstrate operations of darkening and brightening. Finally, histogram was equalized, contrast corrected and the picture was saved with compressing option. The size of the compacted file was approximately two times

smaller than the original. In the end, master copy was restored. Results of all steps in this scenario are shown in Figure 22.



Figure 22. Illustration of the lightness options and their correction

There were more small tests performed but only couple of them are presented above. All those tests resulted with couple of conclusions. The main observation is that the program works well, although the functionality is simple and it could be wider. Functions from the group of opening, saving and closing the picture handled all the situations correctly. Even when a user tried to load an invalid file format it resolved with an error but did not affect the structure of the program. The message explaining that some of the operations are not working with color images appeared in correct moments. Additionally, after loading a colorful picture, specific parts of the menu bar stayed inactive. That prevents users from trying to operate incompatible procedures on wrong input image. The 'Open file' window selects the file formats properly. When it comes to saving the picture with the compression, quality difference is present. In order to significantly decrease the size of the saved file it was necessary to compromise the quality. Figures 23 and 24 illustrate the comparison of an original picture and the compressed one.

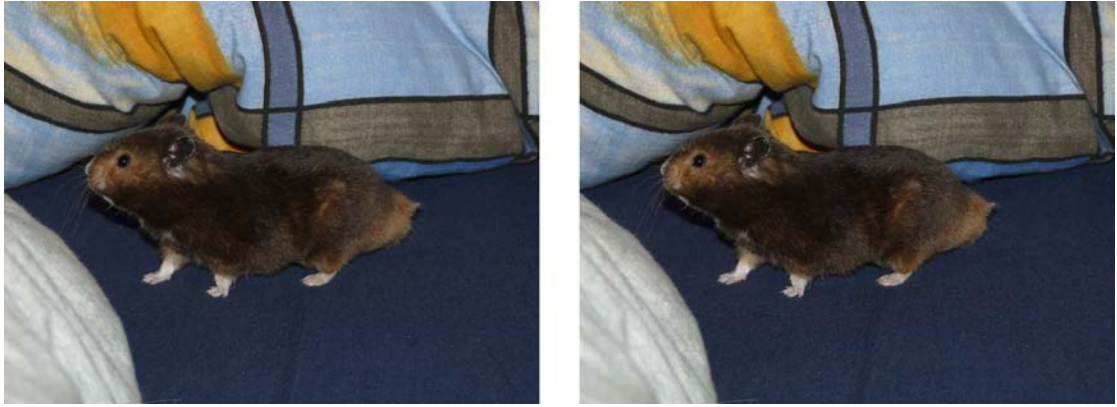


Figure 23. Comparison of a high resolution image before (on the left) and after (on the right) compression

Figure 23 proves that if the input image has high resolution and big size, the difference after compression is almost invisible. Figure 24 represents the results of compaction of a small size picture.



Figure 24. Comparison of a low resolution and small size image before (on the left) and after (on the right) compression

As shown in Figure 24, the difference between compressed and original file is irrelevant, while the size decreases 8 times.

All possible options regarding spatial transformations are working correctly. The ‘crop’ function allows a user to drag the rectangle around the fragment of the picture. As shown in Figure 19, this command can be successfully used for zooming as well. Scenario test demonstrated in Figure 20 checks the ability of the program to convert a color image into a grayscale and then into a bitmap. In my opinion the outcome of both commands is good, especially the bitmap. It is worth mentioning that the default level of threshold, calculated by Matlab produces a satisfactory effect.

Another test was performed only to show how median digital filter deals with noise reduction. It is not perfect but the result is still fair.

Figure 22 illustrates a spectrum of the brightness alterations. The buttons responsible for adjusting the lightness subtract small numerical value from the picture every time a user wants to decrease the brightness and respectively adds the same value to lighten an image. The same test checks if the 'restore' button works correctly. The ability to reconstruct the original picture is very important for users. It gives them the feeling that if something goes wrong during the transformations, they can always go back to the original copy.

Overall evaluation of my image processing application is positive. The only withdraws are the limited functionality towards some specific colormaps and narrow amount of existing functions and filters. Those conclusions leaves the application open for future development.

7 CONCLUSION

7.1 Evaluation of the final outcome and benefits of the study

After wide studies of topic-related areas the prototype of an image processing application was designed and implemented. During the first stage of the development the Graphical User Interface was created. It required careful consideration of the purpose of the program, in order to place elements correctly. The actual programming part introduced me to some challenges concerning mostly connecting functionality with the GUI objects. The most difficult part was to realize that not all of the functions can be implemented for any type of the graphic file. Just to call out an example – operations responsible for flipping the picture vertically and horizontally work only for grayscale images. Learning about the constraints and possibilities of errors helped me to understand how important it is to plan and think through all the details of the new software in the beginning of the work. To sum up, the outcome of this thesis is a simple program that presents possibilities of Matlab and the GUIDE tool.

In addition to the practical part of the thesis, the other goal was to learn new information from various areas related to the computer graphics and Matlab in general. Theory presented in this thesis covers quite precisely the most important topics within the computer graphics, such as colormaps, file formats, vector and raster graphics together with color systems. Great amount of time was given into getting to know the Image Processing Toolbox – one of the libraries of Matlab suite. Finding out what is the range of possible image transformations that can be done with this toolbox helped me to realize that Matlab is a great piece of software and may be widely used in many purposes.

Overall idea behind my thesis was to show that Matlab is not only good for complicated and complex mathematical drawings but also provides a broad collection of regular image processing functions. Of course it is great for medical image transformations and recognition but that is not all. Many programmers do not realize the full potential of Matlab and the Graphical User Interface tool that it provides.

Despite the outcome of my thesis, including the image processing application, the topic stays open to the future modifications and development.

7.2 Future development

Even though the aim of the study was completed, there is still a lot of possibilities for future development. Matlab is a powerful tool, which provides a lot of opportunities. Wide selection of the libraries, filled with multiple choice of available functions gives the programmer almost unlimited chance of growth. A simple image processing application can become a complex piece of a new software. Image Processing Toolbox implements many functions that were not used in my application. Among them there is a group of morphological operations such as dilation, erosion, morphological opening and closing, filling certain areas and more. IPT also provides couple of methods of a thresholding. A big collection of functions implement different types of filters. Some of them are needed in the process of image adjustments and others are responsible for object recognition. Examples of linear and nonlinear filter including their effects are presented in Table 1.

Table 1. Examples of image enhancing filters (Ozimek, lectures from Digital image processing, 2010d)

Category	Filter	Effect
LINEAR	High-pass filter	Sharpening, underlining contours
	Low-pass filter	Reduction of the noise, smoothing out
	Laplace's filter	sharpening
	Edge detection filter	Detection of all directions edges and corners
	Corner detection filter	
NON-LINEAR	Median filter	Noise reduction without blur effect
	Minimum filter	Decrease the brightness of the edge objects
	Maximum filter	Increase the brightness of the edge objects

Additionally, the GUIDE tool described in Chapter 5 gives the programmer multiple selection of elements, which can be used to create a friendly user interface. Those ob-

jects can be connected with different functionalities, resulting with plenty combinations. Table 2 illustrates few examples of possible solutions.

Table 2. Exemplary hypothetical matches between GUI elements and image processing functions

GUI element	Idea for implementation
Button	Finding round objects with one click
Edit text	Entering text and saving it in metadata structure
Edit text	Input RGB data to find and select area of that color
Slider	Three sliders for RGB channels control Four sliders for CMYK channels control
ActiveX - Flash	Flash animation
ActiveX - SQL	Database with images, selectable with SQL query

There are many possibilities for the future development of the application created within this thesis. Matlab is a powerful tool that provides with multiple methods and techniques required for building even very complex programs. The functionality and the design are limited only by the programmer's imagination. But is there a place for a Matlab-created image processing application among many free software that already exist ? It is hard to answer that question one way. As any other program, Matlab has its advantages and disadvantages. It depends strictly on the purpose for the new software. In my opinion basic image processing – maybe not, but anything more than simple transformations is eligible for Matlab environment. Especially when it is about complex modifications including using digital filters in the image enhancements. Overall, Matlab is developing very fast and it is becoming more and more popular among researchers. Maybe someday it will prevail in the computer software world? There is nothing left but wait and see what future will bring.

BIBLIOGRAPHY

Books

Galitz, Wilbert O. 2002. The Essential Guide to User Interface Design – An Introduction to GUI Design Principles and Techniques. United States of America. John Wiley & Sons.

Gonzales, Rafael C. – Woods, Richard E. – Eddins, Steven L 2008. Digital Image Processing using Matlab. Second Edition. United States of America. Gatesmark Publishing.

Koprowski, Robert – Wróbel, Zygmunt 2008. Praktyka przetwarzania obrazów z zadaniami w programie Matlab. Warszawa. Akademicka Oficyna Wydawnicza EXIT.

Marchand, Patrick - Holland, Thomas O. 2003. Graphics and GUIs with Matlab. United States of America. Chapman&Hall/CRC.

Mrozek, Bogumiła – Mrozek, Zbigniew 2001. Matlab 6; Poradnik użytkownika. Warszawa. Wydawnictwo PLJ.

The MarthWorks 2004. Getting started with Matlab. Version 7. United States of America. The MathWorks.

The MarthWorks 2009. Image Processing Toolbox 6 User's Guide. United States of America. The MathWorks.

The MarthWorks 2007. Matlab 7. Creating Graphical User Interfaces. United States of America. The MathWorks.

Vanderburg, Glenn L. 1996. Tricks of the Java programming gurus. United States of America. Sams Publishing.

Unpublished sources

Ozimek, Agnieszka 2009a. Computer Graphics. Materials from Lecture no. 1. Cracow. Cracow University of Technology.

Ozimek, Agnieszka 2009b. Computer Graphics. Materials from Lecture no. 4. Cracow. Cracow University of Technology.

Ozimek, Agnieszka 2010a. Digital image processing. Materials from Lecture no. 4. Cracow. Cracow University of Technology.

Ozimek, Agnieszka 2010b. Digital image processing. Materials from Lecture no. 7. Cracow. Cracow University of Technology.

Ozimek, Agnieszka 2010c. Digital image processing. Materials from Lecture no. 3. Cracow. Cracow University of Technology.

Electronic sources

Moler, Cleve 2004. The origins of Matlab. Matlab News & Notes. Cleve's Corner. 12.2004. Available in www-format:

[URL:www.mathworks.se/company/newsletters/news_notes/clevescorner/dec04.html?s_cid=wiki_matlab_3.](http://www.mathworks.se/company/newsletters/news_notes/clevescorner/dec04.html?s_cid=wiki_matlab_3)

Vector Graphic 2012. The Tech Terms Computer Dictionary. Referred 23.03.2010. Available in www-format:

[URL:http://www.techterms.com/definition/vectorgraphic.](http://www.techterms.com/definition/vectorgraphic)

Cube of RGB color model 2010. Wikimedia Commons. Referred 25.03.2010. Available in www-format:

[URL:http://commons.wikimedia.org/wiki/File:RGBCube_d.svg.](http://commons.wikimedia.org/wiki/File:RGBCube_d.svg)

Comparison of RGB and CMYK color systems 2010. Wikimedia Commons. Referred 25.03.2010. Available in www-format:

[URL:http://commons.wikimedia.org/wiki/File:RGB_and_CMYK_comparison.png.](http://commons.wikimedia.org/wiki/File:RGB_and_CMYK_comparison.png)

Bitmap 2010. WiseGEEK clear answers for common questions. Referred 25.03.2010. Available in www-format:

[URL:http://www.wisegeek.com/what-is-a-bitmap-image.htm.](http://www.wisegeek.com/what-is-a-bitmap-image.htm)

Ready-built colormaps from Matlab's Image Processing Toolbox 2012. Math-Works Documentation. Available in www-format:

[URL:http://www.mathworks.se/help/techdoc/ref/colormap.html.](http://www.mathworks.se/help/techdoc/ref/colormap.html)

APPENDICES

Appendix 1: Full code listing.

```

function varargout = english(varargin)
% ENGLISH M-file for english.fig
%     ENGLISH, by itself, creates a new ENGLISH or raises the exist-
ing
%     singleton*.
%
%     H = ENGLISH returns the handle to a new ENGLISH or the handle
to
%     the existing singleton*.
%
%     ENGLISH('CALLBACK',hObject,eventData,handles,...) calls the
local
%     function named CALLBACK in ENGLISH.M with the given input ar-
guments.
%
%     ENGLISH('Property','Value',...) creates a new ENGLISH or rais-
es the
%     existing singleton*. Starting from the left, property value
pairs are
%     applied to the GUI before english_OpeningFunction gets called.
An
%     unrecognized property name or invalid value makes property
application
%     stop. All inputs are passed to english_OpeningFcn via varar-
gin.
%
%     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
only one
%     instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help english

% Last Modified by GUIDE v2.5 12-Jun-2010 15:29:43

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @english_OpeningFcn, ...
                  'gui_OutputFcn',  @english_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before english is made visible.

```

```

function english_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to english (see VARARGIN)

% Choose default command line output for english
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes english wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = english_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in button1.
function button1_Callback(hObject, eventdata, handles)
% hObject    handle to button1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = handles.oryginal;
axes(handles.axes);
imshow(handles.plik);
set(handles.w_pionie, 'Enable', 'Off');
    set(handles.w_poziomie, 'Enable', 'Off');
    set(handles.kontrast, 'Enable', 'Off');
    set(handles.dodaj_szum, 'Enable', 'Off');
    set(handles.usun_szum, 'Enable', 'Off');
    set(handles.kontrast, 'Enable', 'Off');
    set(handles.histogram, 'Enable', 'Off');
    set(handles.skala_szarosci, 'Enable', 'On');
guidata(hObject, handles);

% --- Executes on button press in button2.
function button2_Callback(hObject, eventdata, handles)
% hObject    handle to button2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.plik = handles.plik-5;
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

```

```

% --- Executes on button press in button3.
function button3_Callback(hObject, eventdata, handles)
% hObject      handle to button3 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
handles.plik = handles.plik+5;
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);
% -----
-
function plik_Callback(hObject, eventdata, handles)
% hObject      handle to plik (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% -----
-
function przekształcenia_Callback(hObject, eventdata, handles)
% hObject      handle to Przekształcenia (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% -----
-
function filtry_Callback(hObject, eventdata, handles)
% hObject      handle to Filtry (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% -----
-
function pomoc_Callback(hObject, eventdata, handles)
% hObject      handle to Pomoc (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% -----
-
function otworz_Callback(hObject, eventdata, handles)
% hObject      handle to Otworz (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
% Loading the Image
[filename, pathname]=uigetfile(
{'*.jpg'; '*.gif'; '*.png'; '*.bmp'; '*.tif'; '*.eps'}, 'Select file');
if filename ==0
    return;
end
handles.pathname = pathname;
handles.filename = filename;
handles.var=strcat(pathname,filename);
handles.plik=imread(handles.var);
handles.oryginal = imread(handles.var);
%Wczytywanie obrazu do axes
axes(handles.axes);
imshow(handles.plik);

```

```

%Ustawianie funkcji aktywnych w zależności od pliku
set(handles.zapisz,'Enable','On');
set(handles.zapisz_jako,'Enable','On');
set(handles.informacja_o_pliku,'Enable','On');
set(handles.w_lewo,'Enable','On');
set(handles.w_prawo,'Enable','On');
set(handles.wytnij,'Enable','On');
set(handles.gaussa,'Enable','On');
set(handles.w_ruchu,'Enable','On');
set(handles.lekkie,'Enable','On');
set(handles.wyostrzenie,'Enable','On');
set(handles.skala_szarosci,'Enable','On');
set(handles.czarno_bialy,'Enable','On');
set(handles.button1,'Enable','On');
set(handles.button2,'Enable','On');
set(handles.button3,'Enable','On');

if isgray(handles.plik) == 1
    set(handles.w_pionie,'Enable','On');
    set(handles.w_poziomie,'Enable','On');
    set(handles.kontrast,'Enable','On');
    set(handles.dodaj_szum,'Enable','On');
    set(handles.usun_szum,'Enable','On');
    set(handles.kontrast,'Enable','On');
    set(handles.histogram,'Enable','On');
    set(handles.skala_szarosci,'Enable','Off');
    msgbox('The image is in grayscale - all fuctions active','Message');
else
    msgbox('The image is not in grayscale - some functions inactive','Message');
end

guidata(hObject, handles);

% -----
-
function zapisz_Callback(hObject, eventdata, handles)
% hObject    handle to Zapisz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

[FileName,PathName]=uinputfile({'*.jpg'; '*.tif'; '*.png'; '*.gif'}, 'Save image');
imwrite(handles.plik,[PathName FileName]);

% -----
-
function zapisz_jako_Callback(hObject, eventdata, handles)
% hObject    handle to Zapisz_jako (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[FileName,PathName]=uinputfile({'*.jpg'; '*.tif'; '*.png'; '*.gif'}, 'Save image');
imwrite(handles.plik,[PathName FileName], 'Quality', 50);

```

```

% -----
-
function informacja_o_pliku_Callback(hObject, eventdata, handles)
% hObject    handle to Informacja_o_pliku (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
i = imfinfo(fullfile(handles.pathname, handles.filename));
A1 = i.Filename;
A = strvcat('File Name: ',A1);
C1 = num2str(i.FileSize);
C2 = strcat(C1,'Bajts');
C = strvcat('File Size: ',C2);
D1 = i.Format;
D = strvcat('File Format: ',D1);
E1 = num2str(i.Width);
E = strcat('Width: ',E1,' pixels');
F1 = num2str(i.Height);
F = strcat('Height: ',F1,' pixels');
G1 = i.ColorType;
G = strvcat('Color Type: ',G1);
H1 = num2str(i.BitDepth);
H2 = strcat(H1,'bits per pixel');
H = strvcat('Bit Depth: ',H2);
I = strvcat(A,C,D,E,F,G,H);
set(handles.text3,'String',I);
guidata(hObject, handles);

% -----
-
function zamknij_Callback(hObject, eventdata, handles)
% hObject    handle to Zamknij (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(gcf);

% -----
-
function obroc_Callback(hObject, eventdata, handles)
% hObject    handle to Obroc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
-
function w_prawo_Callback(hObject, eventdata, handles)
% hObject    handle to w_prawo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = imrotate(handles.plik,-90,'bilinear','loose');
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

% -----
-
function w_lewo_Callback(hObject, eventdata, handles)
% hObject    handle to w_lewo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = imrotate(handles.plik,90,'bilinear','loose');
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

```

```

% -----
-
function odbij_Callback(hObject, eventdata, handles)
% hObject    handle to Odbij (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
-
function w_pionie_Callback(hObject, eventdata, handles)
% hObject    handle to w_pionie (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = fliplr(handles.plik);
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);
% -----
-
function w_poziomie_Callback(hObject, eventdata, handles)
% hObject    handle to w_poziomie (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = flipud(handles.plik);
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

% -----
-
function wytnij_Callback(hObject, eventdata, handles)
% hObject    handle to Wytnij (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = imcrop(handles.plik);
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

% -----
-
function rozmazanie_Callback(hObject, eventdata, handles)
% hObject    handle to Rozmaz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
-
function lekkie_Callback(hObject, eventdata, handles)
% hObject    handle to lekkie (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

H = fspecial('disk',2);
handles.plik = imfilter(handles.plik,H,'replicate');
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

```

```

% -----
-
function gaussa_Callback(hObject, eventdata, handles)
% hObject    handle to gaussa (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
H = fspecial('gaussian');
handles.plik = imfilter(handles.plik,H);
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

% -----
-
function w_ruchu_Callback(hObject, eventdata, handles)
% hObject    handle to gaussa (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
H = fspecial('motion',5,3);
handles.plik = imfilter(handles.plik,H);
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

% -----
-
function wyostrzenie_Callback(hObject, eventdata, handles)
% hObject    handle to wyostrzenie (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
H = fspecial('unsharp',0.05);
handles.plik = imfilter(handles.plik,H,'same');
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

% -----
-
function dodaj_szum_Callback(hObject, eventdata, handles)
% hObject    handle to dodaj_szum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = imnoise(handles.plik, 'salt & pepper',0.02);
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

% -----
-
function usun_szum_Callback(hObject, eventdata, handles)
% hObject    handle to usun_szum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = medfilt2(handles.plik);
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

```

```

% -----
-
function histogram_Callback(hObject, eventdata, handles)
% hObject    handle to histogram (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = histeq(handles.plik);
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

% -----
-
function kontrast_Callback(hObject, eventdata, handles)
% hObject    handle to kontrast (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = imadjust(handles.plik);
axes(handles.axes);
imshow(handles.plik);
guidata(hObject, handles);

% -----
-
function skala_szarosci_Callback(hObject, eventdata, handles)
% hObject    handle to skala_szarosci (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = rgb2gray(handles.plik);
axes(handles.axes);
imshow(handles.plik);
if isgray(handles.plik) == 1
    set(handles.w_pionie, 'Enable', 'On');
    set(handles.w_poziomie, 'Enable', 'On');
    set(handles.kontrast, 'Enable', 'On');
    set(handles.dodaj_szum, 'Enable', 'On');
    set(handles.usun_szum, 'Enable', 'On');
    set(handles.kontrast, 'Enable', 'On');
    set(handles.histogram, 'Enable', 'On');
    set(handles.skala_szarosci, 'Enable', 'Off');

else
end
guidata(hObject, handles);

% -----
-
function czarno_bialy_Callback(hObject, eventdata, handles)
% hObject    handle to czarno_bialy (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.plik = im2bw(handles.plik);
axes(handles.axes);
imshow(handles.plik);
set(handles.w_pionie, 'Enable', 'Off');
    set(handles.w_poziomie, 'Enable', 'Off');
    set(handles.kontrast, 'Enable', 'Off');
    set(handles.dodaj_szum, 'Enable', 'Off');
    set(handles.usun_szum, 'Enable', 'Off');
    set(handles.kontrast, 'Enable', 'Off');
    set(handles.histogram, 'Enable', 'Off');
    set(handles.skala_szarosci, 'Enable', 'Off');

```



```
guidata(hObject, handles);
```

```
% -----
-
function instrukcje_Callback(hObject, eventdata, handles)
% hObject    handle to instrukcje (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
I = strvcat('INSTRUCTION', ' ', 'Functions will be activated after
loading graphic file.', 'Select ~File~ and ~Open~ from menubar, then
select the file and click ~OK~.');
```

```
set(handles.text3, 'String', I);
guidata(hObject, handles);
```

```
% -----
-
function o_programie_Callback(hObject, eventdata, handles)
% hObject    handle to o_programie (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
I = strvcat('Application for image processing with Matlab', 'Created
as a project of Bachelors', 'thesis.', ' ', ' ', ' ', 'Release version 1.0.',
'Version 1.1 planned for 2012.');
```

```
set(handles.text3, 'String', I);
guidata(hObject, handles);
```

```
% -----
-
function o_autorze_Callback(hObject, eventdata, handles)
% hObject    handle to o_autorze (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
I = strvcat('Justyna Inglot is the author of the application .', '
', 'In case of any questions, do not hesitate to contact me:', '
', 'jiii@wp.pl');
```

```
set(handles.text3, 'String', I);
guidata(hObject, handles);
```