Image Editor in MATLAB

Aryan Agal
Department of Energy Science and Engineering
Indian Institute of Technology, Bombay
Mumbai, India
aryanagal@iitb.ac.in

***Abstract*—This project implements some basic image processing algorithms introduced in class. An interface to conveniently access all functionality was also designed.**

***Keywords*—image processing, algorithms, implementation, intensity transforms**

I. Introduction

Any algorithm is best understood if implemented and its results are well known. Image processing algorithms are one such category, which can conveniently be understood by coupling from-scratch implementation with results. This assignment implements an image editor using MATLAB 2018a, in order to improve my understanding of image-processing.

II. Preparation

A. Background Reading

Read the class notes on image processing (EE610, Prof. Amit Sethi, Autumn 2018). Went through a few MATLAB support pages to get the GUI up and running. Took occasional help from the DIP section on www.tutorialspoint.com

B. Design Approach

Used MATLAB GUIDE to build the GUI for the application. Chose MATLAB as GUI was much easier to build in it (WYSIWYG editor was provided). Added different buttons and used their callback functions to implement image processing algorithms, since it was more convenient this way. Used an extra button with each slider to use the slider value, since user could make a mistake in the first try. Implemented undo, undoAll and redo using

Note: starter code for MATLAB was unusable without the fig file which wasn't provided, and python starter code only listed libraries, rendering them both useless.

III. Challenges Faced

A. Replicating Python Lists

Python lists are a convenient way to create and store different renders of the image after application of Image Processing algorithms. Trying to figure out how to get a replica of python lists in MATLAB turned out to be much harder than expected. Browsed quite a few MATLAB support pages, with multiple different solutions, confusing due to each with seemingly inconsistent syntax. Solution: Used a "cell array" defined using "{ and }". Can append to end by directly using var_name{end+1}=value. Very useful for maintaining list after undo, for redo.

B. Inconsistent image value ranges

Debugging the problem where different functions used different ranges of output(0-255 and 0-1 both work with imshow, hence the problem remained undetected till I used multiple filters). Solution: Used 0-1 range consistently everywhere.

C. Histogram Equalization errors

The concept of histogram equalization was unclear to me. I could understand that we need to integrate (hence take a CDF of the intensity map), but could not figure out what to do with the CDF for a long time. Solution: Tried some things(almost trial and error), then realized the cdf map would lie between 0-1. Hence figured out how to map cdf values. Compared results with `histeq` function(inbuilt in MATLAB 2018a) to complete it. This was reimplemented multiple times due to comparison with `histeq` applied on all channels instead of only the intensity channel.

D. Using slider values

Hit a block when trying to use slider values in other functions. Solutions on support pages were not very clear to me. Solution: Looked at example code to figure out.

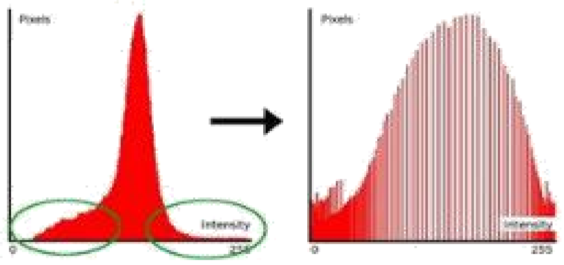
IV. Theory: Image Processing Algorithms

This project required implementation of many image processing algorithms, which are described below.

A. Equalizing Histogram

Histogram equalization is a technique for adjusting image intensities to enhance contrast. It is an intensity transform. Histogram equalization involves transforming the intensity values so that the histogram of the output image approximately matches a specified histogram(here we assume it to match an equally distributed pdf)[1].

Histograms of Intensity Maps[2]



B. Gamma Correction

Gamma correction, or often simply gamma, is a nonlinear operation used to encode and decode luminance or tristimulus values in video or still image systems[3]. Gamma correction is, in the simplest cases, defined by the following power-law expression:

$$V_{\text{out}} = AV_{\text{in}}^{\gamma}$$

where the non-negative real input value Vin is raised to the power gamma, multiplied by a constant A.

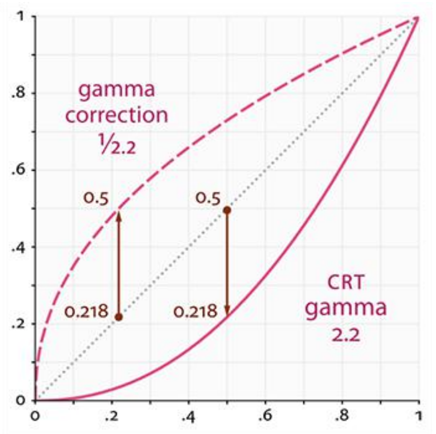


Figure: Gamma Correction curves[4]

C. Log Transformations

The log transformations can be defined by this formula[5]

$$s = c \log(r + 1)$$

Where s and r are the pixel values of the output and the input image and c is a constant. The value 1 is added to each of the pixel value of the input image because if there is a pixel intensity of 0 in the image, then log (0) is equal to infinity. So 1 is added, to make the minimum value at least 1.

During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values. The higher pixel values are kind of compressed in log transformation. This result in following image enhancement.

The value of c in the log transform adjust the kind of enhancement you are looking for.e.

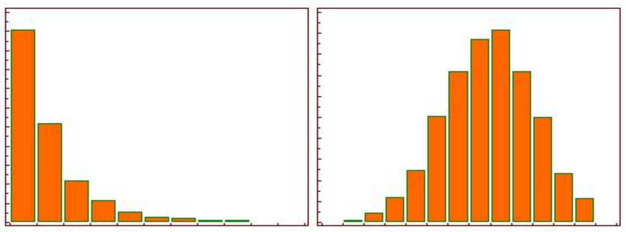
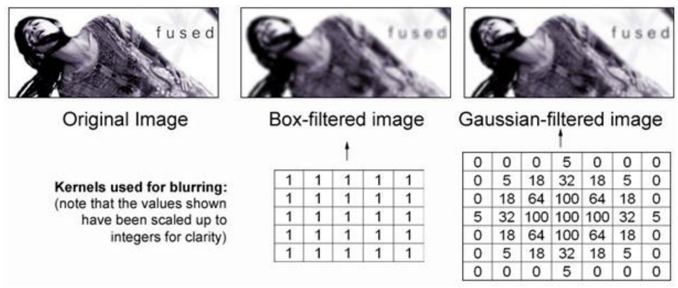


Figure: Log Transformations

D. Blur Transform

Blurring is the application of a filter to make objects within an image less distinct. There are multiple kinds of blurring,

Blurring can be achieved by many ways. The common type of filters that are used to perform blurring are 1)Mean filter 2) Weighted average filter 3)Gaussian filter



Kernels used for blurring:
(note that the values shown have been scaled up to integers for clarity)

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Figure: Blurring

E. Negative Transform[5]

In negative transformation, each value of the input image is subtracted from the L-1 and mapped onto the output image.

So each value is subtracted by 255(in our case). The lighter pixels become dark and the darker picture becomes light. And it results in image negative.

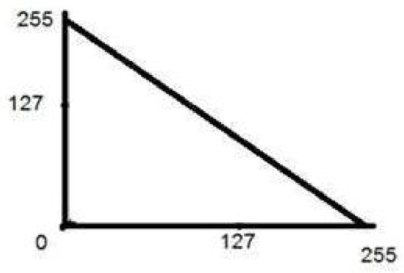


Figure: Negative Image Transform

F. Image Sharpening[6]

Sharpening an image increases the contrast between bright and dark regions to bring out features. The sharpening process is basically the application of a high pass filter to an image. It is similar to blurring, except typically the filter used is centrally weighted, with negative values on all other cells. It works roughly by adding a similar, blurred image.

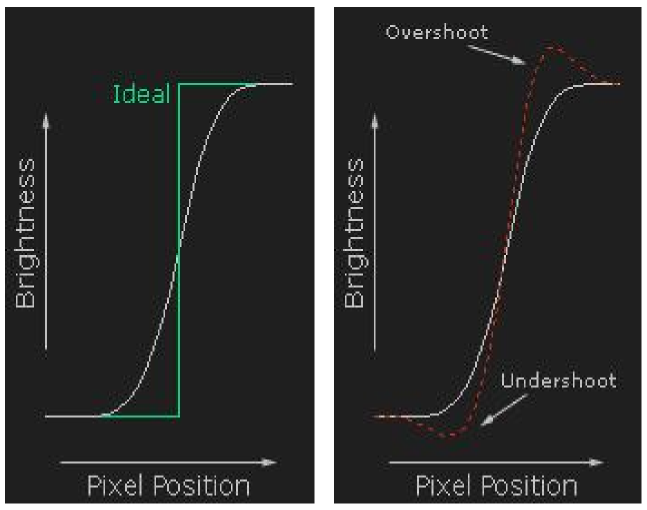


Figure: Image Sharpening

V. TESTING

Testing was done using multiple transforms on each image.

A. Selection of Images

The first image used was Lena, the standard test image for Image processing.[7]

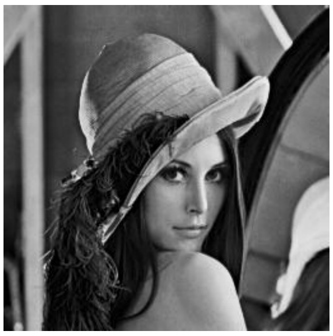
It has bright varied colours, which make it useful for testing colour transforms like the negative transform as well as grayscale.

The second image used is the cameraman image, again quite familiar to the image processing world. It was selected, due to high variation in its intensity.

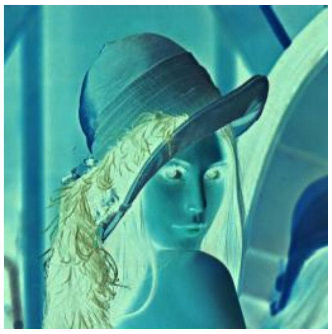
B. Results obtained

Different tests on the images were done. Here are the results:

Original

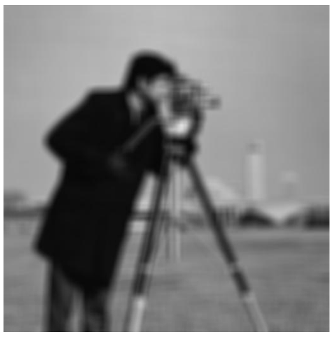


Grayscale

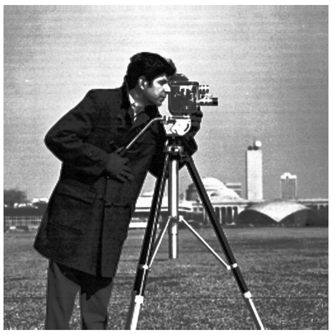


Negative

Original

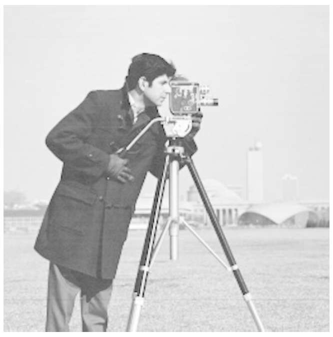


Blurred

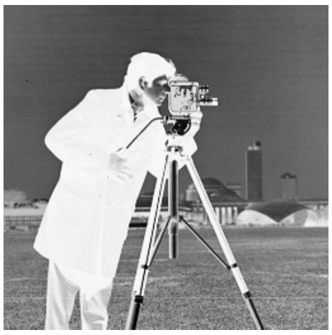


Hist. Equalized

Original

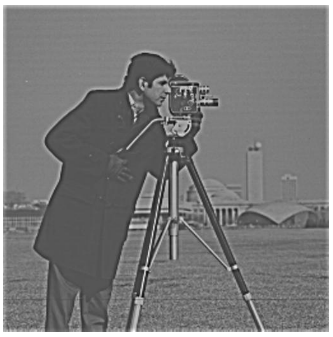


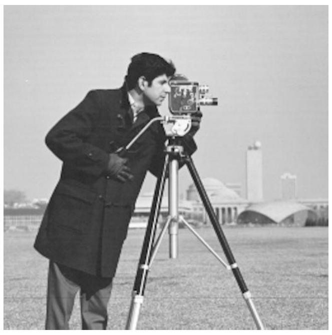
Log-Transformed



Negative

Original


Sharpened


Gamma Corrected

VI. Acknowledgment

I would like to thank Prof. Amit Sethi of the Electrical Engineering Department, for giving me an opportunity to work on this project. I would also like to thank Satyaprajna Sarthak Sahoo, for his cooperation during my ill-health.

VII. References

[1] MathWorks MATLAB Documentation, "Histogram Equalization" https://in.mathworks.com/help/images/histogram-equalization.html

[2] OpenCV documentation, "Histogram Equalization". https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html

[3] Wikipedia entry on "Gamma Correction", https://en.wikipedia.org/wiki/Gamma_correction

[4] Learn OpenGL, "Gamma Correction", https://learnopengl.com/Advanced-Lighting/Gamma-Correction

[5] Log Transformation, Negative Transformation, "Gray Level Transformations", the DIP guide, Tutorialspoint. https://www.tutorialspoint.com/dip/gray_level_transformations.htm

[6] Sharpening: Unsharp Mask, Cambridge in Colour Tutorials https://www.cambridgeincolour.com/tutorials/unsharp-mask.htm

[7] Standard test images, Image Databases of ImageProcessingPlace.com http://www.imageprocessingplace.com/root_files_V3/image_databases.htm

VIII. Code

```
function varargout = ImageEditor(varargin)
% IMAGEEDITOR MATLAB code for ImageEditor.fig
%      IMAGEEDITOR, by itself, creates a new IMAGEEDITOR or raises the
%      existing singleton*.
%
%      H = IMAGEEDITOR returns the handle to a new IMAGEEDITOR or the
%      handle to the existing singleton*.
%
%      IMAGEEDITOR('CALLBACK',hObject,eventData,handles,...) calls the
%      local function named CALLBACK in IMAGEEDITOR.M with the given input
%      arguments.
%
%      IMAGEEDITOR('Property','Value',...) creates a new IMAGEEDITOR or
%      raises the existing singleton*.  Starting from the left, property
%      value pairs are applied to the GUI before ImageEditor_OpeningFcn
%      gets called.  An unrecognized property name or invalid value makes
%      property application stop.  All inputs are passed to
%      ImageEditor_OpeningFcn via varargin.
%
%      *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 ImageEditor

% Last Modified by GUIDE v2.5 03-Sep-2018 01:23:17

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ImageEditor_OpeningFcn, ...
                   'gui_OutputFcn',  @ImageEditor_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 ImageEditor is made visible.
function ImageEditor_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 ImageEditor (see VARARGIN)

% Choose default command line output for ImageEditor
handles.output = hObject;
axes(handles.axes1); % set imshow output to axis1

% Update handles structure
guidata(hObject, handles);

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

% define default values of global variables
global curr_im images_list list_end filename;
images_list = {};
list_end = 1;
images_list{list_end} = zeros(800,800,3); %blank image
curr_im = images_list{list_end}; %set current image to end of list
imshow(curr_im);
filename = ""; % set default value of filename to "" in order to check later from
unloaded state

% --- Outputs from this function are returned to the command line.
function varargout = ImageEditor_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 eqHistButton.
function eqHistButton_Callback(hObject, eventdata, handles)
% hObject    handle to eqHistButton (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
global curr_im images_list list_end;

hsv = rgb2hsv(curr_im);
I = round(255*hsv(:,:,3));

[rows,cols] = size(I);
tot = rows*cols;

%create a list out of intensity values, initialize pdf and cdf arrays
k = I(:);
pdf_var = zeros(256);
cdf_var = zeros(256);

%calculate pmf
for i=1:tot
    pdf_var(k(i)+1)=pdf_var(k(i)+1)+1;
end

%calculate "cmf"
sum=0;
for i=1:256
    sum = sum + pdf_var(i);
    cdf_var(i) = sum;
end

%calculate actual cdf
cdf_var = cdf_var/tot;

%map intensity values to image's intensity map
for i=1:rows
    for j=1:cols
        hsv(i,j,3) = cdf_var(I(i,j)+1);
    end
end

%save image and display
curr_im = hsv2rgb(hsv);

list_end = list_end + 1;
images_list{list_end} = curr_im;
imshow(curr_im);

% --- Executes on button press in gammaButton.
function gammaButton_Callback(hObject, eventdata, handles)
% hObject    handle to gammaButton (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
global curr_im images_list list_end;

gamma = get(handles.gammaSlider, 'Value');
hsv = rgb2hsv(curr_im);
I = hsv(:,:,3);
hsv(:,:,3) = im2double(I.^(10^gamma));

%save image and display
curr_im = hsv2rgb(hsv);
```

```
list_end = list_end + 1;
images_list{list_end} = curr_im;
imshow(curr_im);

% --- Executes on slider movement.
function gammaSlider_Callback(hObject, eventdata, handles)
% hObject    handle to gammaSlider (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
%        slider


% --- Executes during object creation, after setting all properties.
function gammaSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gammaSlider (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in logButton.
function logButton_Callback(hObject, eventdata, handles)
% hObject    handle to logButton (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
global curr_im images_list list_end;

% Take logarithm of intensity values(in 0-255), and normalize.
hsv = rgb2hsv(curr_im);
I = round(hsv(:,:,3).*255);
hsv(:,:,3) = log(I+1)/log(256); % added one to start log from log(1) = 0
disp(hsv(:,:,3))

%save image and display
curr_im = hsv2rgb(hsv);

list_end = list_end + 1;
images_list{list_end} = curr_im;
imshow(curr_im);


% --- Executes on button press in blurButton.
function blurButton_Callback(hObject, eventdata, handles)
% hObject    handle to blurButton (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
global curr_im images_list list_end;

hsv = rgb2hsv(curr_im);

%send to blur function, using the slider-extracted value. Set intensity
%map using the function result.
filterlengthby2 = round(get(handles.blurSlider, 'Value'));
k = applyfilter(filterlengthby2,"");
hsv(:,:,3) = k;
```

```matlab
%save image and display
disp(hsv(:,:,3))
curr_im = hsv2rgb(hsv);

list_end = list_end + 1;
images_list{list_end} = curr_im;
imshow(curr_im);


% --- Executes on slider movement.
function blurSlider_Callback(hObject, eventdata, handles)
% hObject    handle to blurSlider (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
%        slider


% --- Executes during object creation, after setting all properties.
function blurSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to blurSlider (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes on button press in sharpenButton.
function sharpenButton_Callback(hObject, eventdata, handles)
% hObject    handle to sharpenButton (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
global curr_im images_list list_end;

hsv = rgb2hsv(curr_im);

%Take sharpen size from slider, send to the "blur" function to sharpen.
%Normalize to 0-1 range after the process
sharpen_size = round(get(handles.sharpenSlider, 'Value'));
k = applyfilter(sharpen_size,"sharpen");
hsv(:,:,3) = mat2gray(mat2gray(k)+hsv(:,:,3));
disp(hsv(:,:,3))
curr_im = hsv2rgb(hsv);

list_end = list_end + 1;
images_list{list_end} = curr_im;
imshow(curr_im);

% --- Executes on slider movement.
function sharpenSlider_Callback(hObject, eventdata, handles)
% hObject    handle to sharpenSlider (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
%       slider


% --- Executes during object creation, after setting all properties.
function sharpenSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sharpenSlider (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in negateButton.
function negateButton_Callback(hObject, eventdata, handles)
% hObject    handle to negateButton (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
global curr_im images_list list_end;

%negate image
curr_im = 1 - curr_im;

%save image and display
list_end = list_end + 1;
images_list{list_end+1} = curr_im;
imshow(curr_im);


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO) eventdata  reserved - to be defined
% in a future version of MATLAB handles    empty - handles not created
% until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1


% --- Executes on button press in loadButton.
function loadButton_Callback(hObject, eventdata, handles)
% hObject    handle to loadButton (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
global curr_im images_list list_end filename;

[filename,user_canceled] = imgetfile();
if user_canceled == 0
    %if file is loaded, then display
    list_end=1; %set as first image on load
    read = im2double(imread(filename));
    if length(size(read))==2
        read = repmat(read, [1, 1, 3]);
    end
    images_list{list_end} = read;
    curr_im = images_list{list_end};

    imshow(curr_im);
end

% --- Executes on button press in undoButton.
function undoButton_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to undoButton (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
global curr_im images_list list_end;

%roll back to previous image and if start reached, stop rolling back
if list_end ~= 1
    list_end = list_end - 1;
    curr_im = images_list{list_end};
    imshow(curr_im);
end

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

%roll back to first image in list
global curr_im images_list list_end;
list_end = 1;
curr_im = images_list{list_end};
imshow(curr_im);


% --- Executes on button press in saveButton.
function saveButton_Callback(hObject, eventdata, handles)
% hObject    handle to saveButton (see GCBO) eventdata  reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
global curr_im filename;
if filename~=""
    % if a file has been loaded, write the image to the source
    imwrite(curr_im, filename);
end

function res =  applyfilter(filterlengthby2,a)
global curr_im;

%get grayscale image with dims
hsv = rgb2hsv(curr_im);
I = hsv(:,:,3);
[rows,cols] = size(I);

for i=1:rows
    for j=1:cols
        %make area of application of filter
        A      =      I(max(1,i-filterlengthby2):min(rows,i+filterlengthby2),max(1,j-
filterlengthby2):min(cols,j+filterlengthby2));

        %if sharpen, then use a suitable filter
        if a=="sharpen"
            p = (2*filterlengthby2+1)^2;
            hsv(i,j,3) = (p*I(i,j)-sum(A(:)))/p;
        else
        %else blur with mean filter
            hsv(i,j,3) = mean(A(:));
        end
    end
end

res = hsv(:,:,3);
```

```
% --- Executes on button press in graybutton.
function graybutton_Callback(hObject, eventdata, handles)
% hObject    handle to graybutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global curr_im images_list list_end;

%select intensity map from image
curr_im = repmat(rgb2gray(curr_im), [1, 1, 3]);

list_end = list_end + 1;
images_list{list_end} = curr_im;
imshow(curr_im);


% --- Executes on button press in redoButton.
function redoButton_Callback(hObject, eventdata, handles)
% hObject    handle to redoButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global curr_im images_list list_end;
k = "";
try
    k = images_list{list_end+1};
catch
    msgbox("Can't redo, no further images stored.");
    return
end

list_end = list_end + 1;
curr_im = images_list{list_end};
imshow(curr_im);
```