CM2202 - Scientific Computing and Multimedia Applications

Individual project work – MATLAB Image Browser/Editor

Prof A D Marshall & Dr Yukun Lai

Geraint Harries - 1100682

Time Spent: 20 Hours

Overview of design and implementation

I decided to design the UI with a selection of buttons and sliders (see appendix 1). This, I decided was the friendliest way with regards to the user. Users instinctively know how to use buttons and sliders. Although this may not be scaleable however, given that the system is not going to be scaled up, it is appropriate.

I satisfied the requirement for “providing a means for organizing the media into appropriate folders or albums.” by using the folder manager built into GUIDE. (See appendix 2) through this you are able to create and name a folder, copy files and store pictures appropriately.

The workflow of the system is you open an image, are able to edit it by pressing a series of buttons and sliders and then are able to save the image and close the system.

<u>The total list of features:</u>

- Save
- Open
- Flip (Horizontal and Vertical)
- Rotate (90 degrees clockwise, anti-clockwise and bespoke*)
- 3D histogram
- Greyscale
- Black and White
- Contrast (Edit greyscale and increase colour)
- Sharpen
- Blur
- Crop
- Fish Eye
- Edge Detection
- Noise Reduction
- Instagram
- Just one colour (Red, Green, Blue)
- Undo/ Reset
- Brightness*

* Slider

Algorithmic Description of the main elements of the solution (Including Novel Features)

Rotate

There are several options when it comes to rotate. You can either click a button to rotate 90 degrees clockwise or anti clockwise or use the slider to rotate the image a different amount.

The rotations are done by using the MatLab function imrotate(), where the parameters are the image and the angle you wish to rotate by. For the 90 degree rotations the values 90 and -90 are used. For the bespoke angle, the slider value is entered.

Grey (B&W)

The greyscale function runs the function rgb2grey(). It only takes an image as parameter and formats the weighted sum of the RGB values (0.2989 * R + 0.5870 * G + 0.1140 * B).

The black and white function im2bw() converts a greyscale image into a black and white image. imc

Contrast

There are two contrast functions, a greyscale contrast and a colour contrast. The greyscale contrast works my using the built in function imcontrast() which opens a contrast panel (see appendix 6). This allows bespoke contrast adjustment of greyscale images.

The colour contrast works on a push button system where a click increases the contrast by an arbitrary amount. n clicks increases it n times. It uses a Matlab function called imadjust(). The parameters of the imadjust are the image, the current colour values and the doctored colour values.

Sharpen and Blur

For sharpen I used a the matlab function fspecial(). fspecial creates a predefined filter depending on the parameters entered in the function.

I used the same function as sharpen but added a gaussian filter as a parameter. This creates a blur effect on the image.

Brightness

For this function I used the value of a slider multiplied by 100 and added it to the current picture. The picture is then saved with the new brightness value.

Flips

To flip the image I used the Matlab function flipdim(). You enter the image and an integer to determine the axis in which it's being flipped (1 - Vertical, 2 - Horizontal).

Undo/ Reset

For the reset function, I declared a global variable called backspace which when a function was called stored the image before the function had affected it. This meant that at anytime

if you clicked backspace the image would be changed to the image before it was affected. This allows only one undo, however you are able to re-open the original image

Fish Eye

I downloaded a lens distort file from Matlab Central[1] which distorts the image through a barrel distort. A barrel distort image magnification decreases with distance from the optical axis. (See appendix 3)

Histogram

I downloaded a 3D histogram from Matlab Central[2]. When run, this codes opens another figure with a 3D histogram with colour spheres varying in size depending on frequency. This enables you to see a divided colour representation of the image and understand the main colours. (See appendix 4)

Instagram

I downloaded an instagram filter from an online blog[3] which, when run, applies an “instagram” affect on the current image. (See Appendix 5). It creates the effect by calculating the normalization matrix, calculating desaturation movements, adding yellow, desaturating and edge burning.

One Colour

I used some code from stack overflow[4] which turned a whole image greyscale apart from one colour. On the system I have three buttons, red, green and blue which when pressed greyscale the image apart from the chosen colour. (See appendix 7)

Open

I used Matlab function uigetfile() for opening an image and then set that image as the current image. When opening a file you have the option to open several file types (See Appendix 8). Including, jpeg, png and bmp

Save

I used the Matlab function uiputfile() and imwrite() for saving an edited image. Similar to above you have several options when saving the file in regards to file type.

Crop

I used the inbuilt matlab function imcrop() to crop images. The function requires one parameter, which is the image you’re cropping. All you have to simply do then is click the button, drag the cursor on the area you want cropping and double click. (See Appendix 9 and 10)

Code

Main

```
function varargout = guidepush(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @guidepush_OpeningFcn, ...
                   'gui_OutputFcn',  @guidepush_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


function guidepush_OpeningFcn(hObject, handles, varargin)

handles.output = hObject;

guidata(hObject, handles);

function varargout = guidepush_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

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

Backspace = Picture;

value = 0;
value = get(hObject,'Value')*100;
Picture = Picture + value;
imshow(Picture)
% 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 Brightness_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Brightness (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 BW.
function BW_Callback(hObject, eventdata, handles)
% hObject    handle to BW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Picture;
global Backspace;

Backspace = Picture;

try
    if size(Picture, 3) == 3
        Picture = im2bw(Picture);
    end
    imshow(Picture)
catch exception
    warndlg(exception,'Error')
end


% --- Executes on button press in Open.
function Open_Callback(hObject, eventdata, handles, varagin)
% hObject    handle to Open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Picture;
global filename;
global pathname;
global showPicture;
global Backspace;

Backspace = Picture;


try
    [filename, pathname] =
uigetfile({'*.jpg';'*.png';'*.jpeg';'*.ai';'*.bmp';'*.emf';'*.fig';'*.pbm';'*.pc
x';'*.pdf';'*.pgm';'*.png';'*.ppm';'*.tif'}, 'Pick a photo');
    A = strcat(pathname, filename);
    disp(filename)
    Picture = imread(A);
    showPicture = imshow(Picture);
catch exception
    warndlg(exception,'Error')
end

% --- Executes on button press in Save.
function Save_Callback(hObject, eventdata, handles)
% hObject    handle to Save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global filename;
global pathname;
global Picture;
global Backspace;
```

```
Backspace = Picture;

[filename, pathname] = uiputfile({'*.jpg'},'Save as');
dir = strcat(pathname, filename);
disp(dir);
imwrite(Picture, dir)


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

Backspace = Picture;

try
    if size(Picture, 3) == 3
        Picture = rgb2gray(Picture);
    end
    imshow(Picture)
catch exception
    warndlg(exception,'Error')
end

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

Backspace = Picture;

try
    Picture = imrotate(Picture, 90);
    imshow(Picture)
catch exception
    warndlg(exception,'Error')
end

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

Backspace = Picture;

try
    Picture = flipdim(Picture, 1);
    imshow(Picture)
catch exception
    warndlg(exception,'Error')
```

```
end

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

Backspace = Picture;

try
    Picture = flipdim(Picture, 2);
    imshow(Picture)
catch exception
    warndlg(exception,'Error')
end

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

Backspace = Picture;

try
    Picture = imcrop(Picture);
    imshow(Picture)
catch exception
    warndlg(exception,'Error')
end

% --- Executes on slider movement.
function slider6_Callback(hObject, eventdata, handles)
% hObject    handle to slider6 (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
global Picture;
global Backspace;

Backspace = Picture;

try
    value = get(hObject,'Value')*100;
    Picture = imrotate(Picture, value);
    imshow(Picture)
catch exception
    warndlg(exception,'Error')
end

% --- Executes during object creation, after setting all properties.
function slider6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider6 (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 FishEye.
function FishEye_Callback(hObject, eventdata, handles)
% hObject    handle to FishEye (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Picture;
global Backspace;

Backspace = Picture;

try
    Picture = lensdistort(Picture, 2);
    imshow(Picture)
catch exception
    warndlg(exception,'Error')
end

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns listbox1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from listbox1

% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


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

Backspace = Picture;
```

```
try
    if size(Picture, 3) == 3
        Picture = retro_filter_vectorized(Picture,0.1, 0.5, 0.3, 0.5, 0.05);
    else
        warndlg('You are unable to Instagram a binary image','Error')
    end
    imshow(Picture)
catch exception
    warndlg(exception,'Error');
end

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

Backspace = Picture;

try
    if size(Picture, 3) == 3
      Picture = rgb2gray(Picture);
    end
    Picture = edge(Picture, 'sobel');
    imshow(Picture);
catch exception
    warndlg(exception,'Error')
end

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

Backspace = Picture;

try
    h = fspecial('gaussian', 4, 4);
    Picture = imfilter(Picture, h);
    imshow(Picture);
catch exception
    warndlg(exception,'Error')
end

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

Backspace = Picture;

try
```

```
    h = fspecial('unsharp');
    Picture = imfilter(Picture,h);
    imshow(Picture);
catch exception
    warndlg(exception,'Error')
end

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

Backspace = Picture;


try
    if size(Picture, 3) == 1
        showPicture = imshow(Picture);
        Picture = imcontrast(showPicture);

        ImageHandle = findobj(handles, 'Type', 'image');

    else
        warndlg('You are only able to edit the contrast of a binary image.
Convert to greyscale before continuing','Error')
    end
catch exception
    warndlg(exception,'Error')
end

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

Backspace = Picture;


try
    %if size(Picture, 3) == 3
     %   Picture = imadjust(Picture,[.2 .3 0; .6 .7 1],[]);
    %else
     %   warndlg('Only Colour Images are Valid','Error')

    %end

    if size(Picture, 3) == 3
        Picture = imadjust(Picture, [.2 .3 0; .6 .7 1], []);
    else
        Picture = imadjust(Picture, [0.2 0.5],[]);
    end
    imshow(Picture);
catch exception
    warndlg(exception,'Error')
```

```
end

% --- Executes on button press in Histogram.
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)
global Picture;
global filename;
global Backspace;

Backspace = Picture;


disp(filename);

    [freq, freq_emph, freq_ly] = image_hist_RGB_3d(filename);
    show(Picture)


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

Backspace = Picture;

try
    if size(Picture, 3) == 1
        Picture = wiener2(Picture,[5 5]);
    else
        warndlg('Only Greyscale Images are Valid, please greyscale your
image','Error')
    end
    imshow(Picture);
catch exception
    warndlg(exception,'Error')
end


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

Backspace = Picture;

try
    Picture = imrotate(Picture, 270);
    imshow(Picture)
catch exception
    warndlg(exception,'Error')
end
```

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

Backspace = Picture;

for mm = 1:size(Picture,1)
    for nn = 1:size(Picture,2)
        if Picture(mm,nn,1) < 80 || Picture(mm,nn,2) > 80 || Picture(mm,nn,3) >
100
            gsc = 0.3*Picture(mm,nn,1) + 0.59*Picture(mm,nn,2) +
0.11*Picture(mm,nn,3);
            Picture(mm,nn,:) = [gsc gsc gsc];
        end
    end
end
imshow(Picture)


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

Backspace = Picture;

for mm = 1:size(Picture,1)
    for nn = 1:size(Picture,2)
        if Picture(mm,nn,2) < 80 || Picture(mm,nn,3) > 80 || Picture(mm,nn,1) >
100
            gsc = 0.3*Picture(mm,nn,1) + 0.59*Picture(mm,nn,2) +
0.11*Picture(mm,nn,3);
            Picture(mm,nn,:) = [gsc gsc gsc];
        end
    end
end
imshow(Picture)

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

Backspace = Picture;

for mm = 1:size(Picture,1)
    for nn = 1:size(Picture,2)
        if Picture(mm,nn,3) < 80 || Picture(mm,nn,1) > 80 || Picture(mm,nn,2) >
100
```

```
            gsc = 0.3*Picture(mm,nn,1) + 0.59*Picture(mm,nn,2) +
0.11*Picture(mm,nn,3);
            Picture(mm,nn,:) = [gsc gsc gsc];
        end
    end
end
imshow(Picture)


% --- Executes on button press in Reset.
function Reset_Callback(hObject, eventdata, handles)
% hObject    handle to Reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Backspace;
global Picture;
Picture = Backspace;

imshow(Picture);
```

Retro Filter

```
function [ img ] = retro_filter_vectorized( img, alpha, beta, gamma, delta,
epsilon )
%RETRO_FILTER_VECTORIZED Runs a retro filter on an image
%   Runs a retro filter on an image.
%
%   PARAMETERS
%   ==========
%
%   all parameters are in the range [0,1]
%   alpha       weight for yellow sepia tone
%   beta        weight for desaturation
%   gamma       weight for blue adjustment
%   delta       weight for burning the edges of the image
%   epsilon     used to normalize perturbations.
%
%   constants that worked well for me in order of their appearance:
%   alpha=0.1, beta=0.5, gamma=0.3, delta=0.5, epsilon=0.05
%   ...on most images, at least

    s = size(img);

    % calculate center
    c = s(1:2)/2;
    n = sum(c.^2); % normalization constant

    % calculate normalized intensity matrix
    i = (img(:,:,1) + img(:,:,2) + img(:,:,3))/3;

    % calculate desaturation movements
    mr = (img(:,:,2) + img(:,:,3))/2 - img(:,:,1);
    mg = (img(:,:,1) + img(:,:,3))/2 - img(:,:,2);
    mb = (img(:,:,1) + img(:,:,2))/2 - img(:,:,3);

    % make a meshgrid
    [X,Y] = meshgrid(1:s(2), 1:s(1));
```

```
    % add yellow for sepia
    img(:,:,1:2) = img(:,:,1:2) + alpha*255;

    % turn the blue channel into a weighted sum of the original
    % blue value and the intensity
    img(:,:,3) = (1-gamma)*img(:,:,3) + gamma*i;

    % desaturate
    img(:,:,1) = img(:,:,1) + mr*beta;
    img(:,:,2) = img(:,:,2) + mg*beta;
    img(:,:,3) = img(:,:,3) + mb*beta;

    % make a matrix (the size of the image) of normally distributed random
    % values
    rand = delta*ones(s(1:2)) - normrnd(0,1,s(1:2)).^2*epsilon;

    % burn the edges
    black = uint8(255 * rand .* ((X-c(2)).^2 + (Y-c(1)).^2)/n);
    img(:,:,1) = img(:,:,1) - black;
    img(:,:,2) = img(:,:,2) - black;
    img(:,:,3) = img(:,:,3) - black;

    % bind the image channels between 0 and 255 in case any overflow or
    % underflow occured
    img(img < 0) = 0;
    img(img > 255) = 255;
end
```

Lens Distort

```
function I2 = lensdistort(I, k, varargin)
%LENSDISTORT corrects for barrel and pincusion lens abberations
%   I = LENSDISTORT(I, k)corrects for radially symmetric distortions, where
%   I is the input image and k is the distortion parameter. lens distortion
%   can be one of two types: barrel distortion and pincushion distortion.
%   In "barrel distortion", image magnification decreases with
%   distance from the optical axis. The apparent effect is that of an image
%   which has been mapped around a sphere (or barrel). In "pincushion
%   distortion", image magnification increases with the distance from the
%   optical axis. The visible effect is that lines that do not go through the
%   centre of the image are bowed inwards, towards the centre of the image,
%   like a pincushion [1].
%
%   I = LENSDISTORT(...,PARAM1,VAL1,PARAM2,VAL2,...) creates a new image image,
%   specifying parameters and corresponding values that control various aspects
%   of the image distortion correction. Parameter names case does not matter.
%
%   Parameters include:
%
%   'bordertype'            String that controls the treatment of the image
%                           edges. Valid strings are 'fit' and 'crop'. By
%                           default, 'bordertype' is set to 'crop'.
%
%   'interpolation'         String that specifies the interpolating kernel
%                           that the separable resampler uses. Valid
```

```
%                           strings are 'cubic', 'linear' and 'nearest'. By
%                           default, the 'interpolation' is set to 'cubic'
%
%   'padmethod'             string that controls how the resampler
%                           interpolates or assigns values to output elements
%                           that map close to or outside the edge of the input
%                           array. Valid strings are 'bound', circular',
%                           'fill', 'replicate', and symmetric'. By
%                           default, the 'padmethod' is set to 'fill'
%
%   'ftype'                 Integer between 1 and 4 that specifies the
%                           distortion model to be used. The models
%                           available are
%
%                           'ftype' = 1:    s = r.*(1./(1+k.*r));
%
%                           'ftype' = 2:    s = r.*(1./(1+k.*(r.^2)));
%
%                           'ftype' = 3:    s = r.*(1+k.*r);
%
%                           'ftype' = 4:    s = r.*(1+k.*(r.^2));
%
%                           By default, the 'ftype' is set to 4.
%
%   Class Support
%   -------------
%   An input intensity image can be uint8, int8, uint16, int16, uint32,
%   int32, single, double, or logical. An input indexed image can be uint8,
%   uint16, single, double, or logical.
%
%   Examples
%   --------
%       % read image
%       I = imread('cameraman.tif');
%
%       % Distort Image
%       I2 = lensdistort(I, 0.1);
%
%       % Display both images
%       imshow(I), figure, imshow(I2)
%
%   References
%   --------------
%   [1] http://en.wikipedia.org/wiki/Distortion_(optics), August 2012.
%
%   [2] Harri Ojanen, "Automatic Correction of Lens Distortion by Using
%       Digital Image Processing," July 10, 1999.
%
%   [3] G.Vassy and T.Perlaki, "Applying and removing lens distortion in post
%       production," year???
%
%   [4] http://www.mathworks.com/products/demos/image/...
%       create_gallery/tform.html#34594, August 2012.
%
%   Created by Jaap de Vries, 8/31/2012
%   jpdvrs@yahoo.com
%
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------
```

```
% This part of the codes creates variable input parameters using the input
% parser object
p = inputParser;
%   Make input string case independant
p.CaseSensitive = false;

%   Specifies the required inputs
addRequired(p,'I',@isnumeric);
addRequired(p,'k',@isnumeric);

%   Sets the default values for the optional parameters
defaultFtype = 4;
defaultBorder = 'crop';
defaultInterpolation = 'cubic';
defaultPadmethod = 'fill';

%   Specifies valid strings for the optional parameters
validBorder = {'fit','crop'};
validInterpolation = {'cubic','linear', 'nearest'};
validPadmethod = {'bound','circular', 'fill', 'replicate', 'symmetric'};

%   Funtion handles to determine wheter a proper input string has been used
checkBorder = @(x) any(validatestring(x,validBorder));
checkInterpolation = @(x) any(validatestring(x,validInterpolation));
checkPadmethod = @(x) any(validatestring(x,validPadmethod));

%   Create optional inputs
addParamValue(p,'bordertype',defaultBorder,checkBorder);
addParamValue(p,'interpolation',defaultInterpolation,checkInterpolation);
addParamValue(p,'padmethod',defaultPadmethod,checkPadmethod);
addParamValue(p,'ftype',defaultFtype,@isnumeric);

%   Pass all parameters and input to the parse method
parse(p,I,k,varargin{:});

%--------------------------------------------------------------------------
% This determines wether its a color (M,N,3) or gray scale (M,N,1) image
if ndims(I) == 3
     for i=1:3
        I2(:,:,i) = imdistcorrect(I(:,:,i),k);
     end
elseif ismatrix(I)
    I2 = imdistcorrect(I,k);
else
    error('Unknown image dimensions')
end

%--------------------------------------------------------------------------
% Nested function that perfoms the transformation
    function I3 = imdistcorrect(I,k)
    % Determine the size of the image to be distorted
    [M N]=size(I);
    center = [round(N/2) round(M/2)];
    % Creates N x M (#pixels) x-y points
    [xi,yi] = meshgrid(1:N,1:M);
    % Creates converst the mesh into a colum vector of coordiantes relative to
    % the center
    xt = xi(:) - center(1);
    yt = yi(:) - center(2);
    % Converts the x-y coordinates to polar coordinates
```

```
    [theta,r] = cart2pol(xt,yt);
    % Calculate the maximum vector (image center to image corner) to be used
    % for normalization
    R = sqrt(center(1)^2 + center(2)^2);
    % Normalize the polar coordinate r to range between 0 and 1
    r = r/R;
    % Aply the r-based transformation
    s = distortfun(r,k,p.Results.ftype);
    % un-normalize s
    s2 = s * R;
    % Find a scaling parameter based on selected border type
    brcor = bordercorrect(r,s,k, center, R);

    s2 = s2 * brcor;


    % Convert back to cartesian coordinates
    [ut,vt] = pol2cart(theta,s2);

    u = reshape(ut,size(xi)) + center(1);
    v = reshape(vt,size(yi)) + center(2);
    tmap_B = cat(3,u,v);
    resamp = makeresampler(p.Results.interpolation, p.Results.padmethod);
    I3 = tformarray(I,[],resamp,[2 1],[1 2],[],tmap_B,255);
    end

%--------------------------------------------------------------------------
% Nested function that creates a scaling parameter based on the
% 'bordertype' selected
    function x = bordercorrect(r,s,k,center, R)
        if k < 0
            if strcmp(p.Results.bordertype, 'fit')
               x = r(1)/s(1);
            end
            if strcmp(p.Results.bordertype,'crop')
               x = 1/(1 + k*(min(center)/R)^2);
            end
        elseif k > 0
            if strcmp(p.Results.bordertype, 'fit')
               x = 1/(1 + k*(min(center)/R)^2);
            end
            if strcmp(p.Results.bordertype, 'crop')
               x = r(1)/s(1);
            end
        end
    end

%--------------------------------------------------------------------------
% Nested function that pics the model type to be used
    function s = distortfun(r,k,fcnum)
        switch fcnum
        case(1)
            s = r.*(1./(1+k.*r));
        case(2)
            s = r.*(1./(1+k.*(r.^2)));
        case(3)
            s = r.*(1+k.*r);
        case(4)
            s = r.*(1+k.*(r.^2));
        end
```

```
    end


end
```

3D Histogram

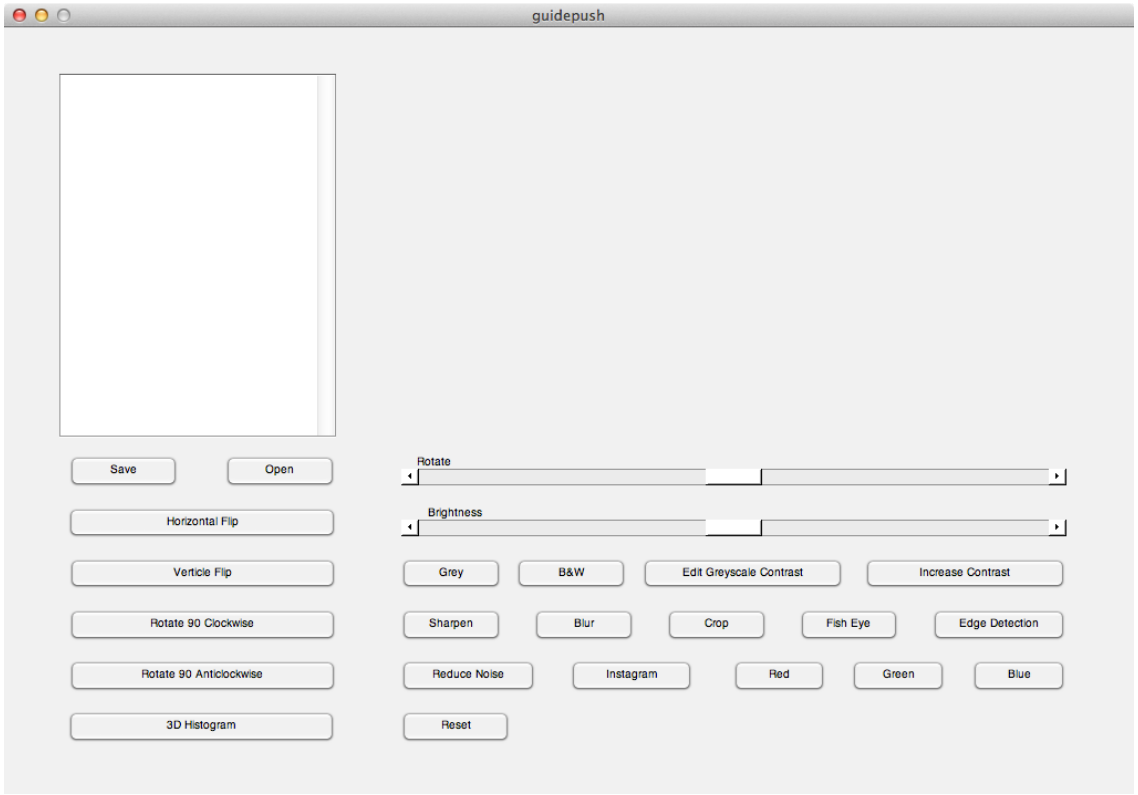
```
function [freq, freq_emph, freq_app] = image_hist_RGB_3d(imname,n,gamma)

% Creates 3D-histogram from an RGB image
% in the form of balls in the RGB cube
%
% Input:
%  IMNAME......Name of the image file in RGB 24bpp
%  N...........Number of histogram bins within each axis;
%              if undefined, the default value is taken N = 6
%  GAMMA.......Power value used for emphasizing small frequencies:
%              out = out.^GAMMA;
%              GAMMA = 1 means no change.
%              If undefined, the default value is GAMMA = 0.5.
%              If GAMMA is set to 0, the "emphasizing" is done so that
%              empty cells will remain at zero and the rest will contain 1.
%
% Output:
%  FREQ.........Frequencies arranged in the 3D matrix. The resultant values
%               can be also non-integral (due to a compensation which is
%               done if the cells ranges differ).
%  FREQ_EMPH....Emphasized values of FREQ, dependent on GAMMA
%  FREQ_APP.....Frequencies (without emphasizing) arranged in the 3D matrix,
%               layers arranged according to the initial appearance of the
histogram
%               FREQ_APP(i,j,k) :
%               (R increases with j, G increases with k, B increases with i)
%
% Example:
%  [freq, freq_emph, freq_app] = image_hist_RGB_3d('image.jpg',5,0.6);

% (c) 2012 Jan Zatyik, Pavel Rajmic,
% Brno University of Technology, Czech Republic
% version 1.25
% www.splab.cz

% Last revision: October 19, 2012

% Credits: inspired by
% http://ij-plugins.sourceforge.net/ij-vtk/color-space/index.html



%% Checking the input arguments
```

```
if nargin <= 2
    gamma = 0.5;
end
if nargin == 1
    n = 6;
end
if n == 1
    error('it is nonsense to use n=1');
end

%% Reading the image
disp('Starting...')
disp('Reading the image...');

im = imread(imname);    % loading the image
s = size(im);
im = double(im);

%% Assigning the pixels to the histogram cells
disp('Assigning the pixels to the histogram cells...');
step = 255/n;
cell_no = ceil(im/step); % numbers of cells for all pixels
% cell_no can be computed as 0, so we assing them to the first cell
cell_no(cell_no==0) = 1;


%% Computing the frequencies
disp('Computing the frequencies...');
% initialization of vector, which will be used for saving the frequencies
freq = zeros(n^3,1);

% computing to which index belong the data
a = cell_no(:,:,1) - 1;
b = cell_no(:,:,2) - 1;
c = cell_no(:,:,3) - 1;
index = n * a +n^2 *b + c + 1;

% computation of frequencies
for col = 1:s(2)
    for row = 1:s(1)
        freq(index(row,col)) = freq(index(row,col)) + 1;
    end
end
% Reshaping into 3D-array
freq = reshape(freq,n,n,n);


%% Compensation due to (possible) non-uniform length of cells
bound = linspace(0,255,n+1); %where are the cell boundaries
bound_int = floor(bound); %integers
% averages (used later during plotting)
cell_avrg = cumsum(diff(bound)) - (255/n)/2;
cell_avrg_int = round(cell_avrg);

% how many values can the cells contain
% (these values can differ by 1 or will be the same)
cell_ranges = diff(bound_int);
maxmin_ratio = max(cell_ranges) / min(cell_ranges);
% determining which cells are to be compensated
cells_to_compensate = find(cell_ranges - min(cell_ranges)); %nonzero elements
```

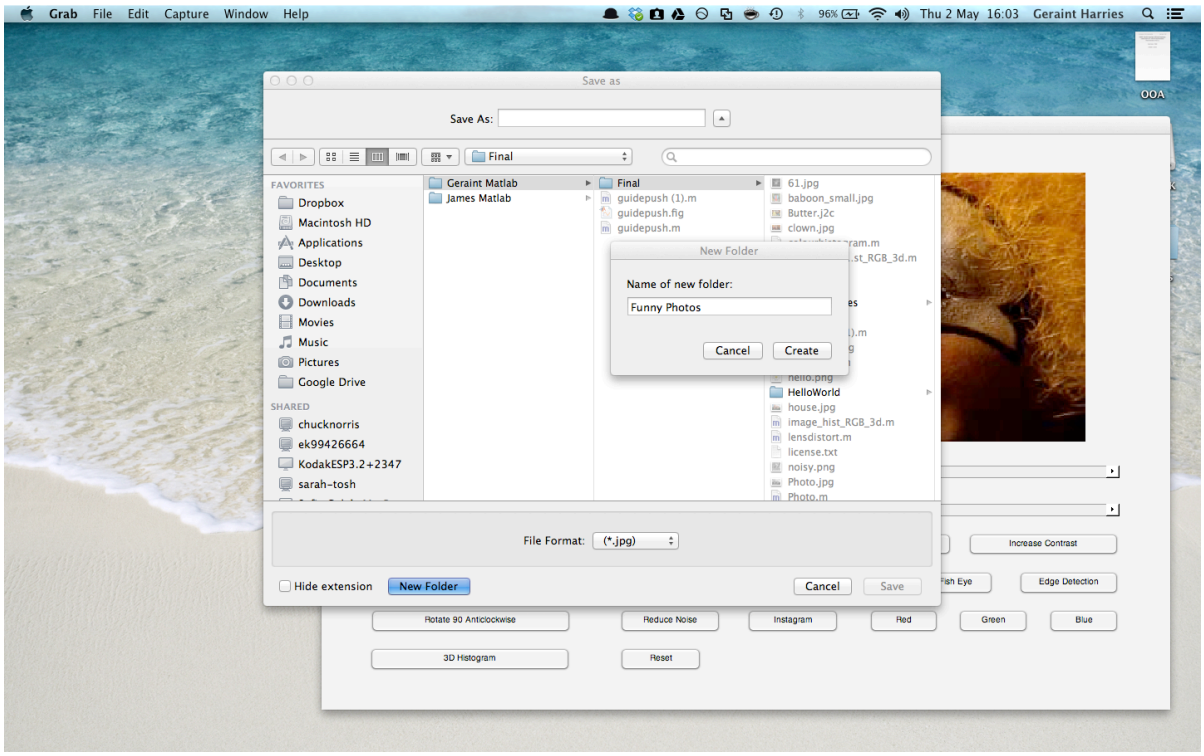
```
% calculate the compensated frequencies (rows(R), columns(G), slides(B))
freq(cells_to_compensate,:,:) = freq(cells_to_compensate,:,:) / maxmin_ratio;
freq(:,cells_to_compensate,:) = freq(:,cells_to_compensate,:) / maxmin_ratio;
freq(:,:,cells_to_compensate) = freq(:,:,cells_to_compensate) / maxmin_ratio;


%% Emphasizing small frequencies by gamma
maxfreq = max(max(max(freq))); %maximum frequency
if gamma ~=1
    disp(['Recalculating frequencies by means of gamma=' num2str(gamma) ' ...'])
    if gamma == 0
        disp('!!! Warning: GAMMA is zero!');
        freq_emph = zeros(size(freq)); %zeros everywhere
        freq_emph(freq~=0) = 1; %ones; the same result as if .^0 was computed
        maxfreq = 1;
    else
        freq_emph = freq / maxfreq;   %first, normalize to [0,1]
        freq_emph = freq_emph.^gamma; %second, emphasize
        freq_emph = freq_emph * maxfreq; %finally, un-normalize
        %maximum frequency remains unchanged
    end
else
    freq_emph = freq; %no change
end


%% Drawing the histogram
disp('Drawing the histogram...')
figure
% whitebg([0.9 0.9 0.9])
maxradius = 255/n;
[Rss Gss Bss] = sphere(16); % mesh for unit sphere
% resizing the sphere to maximum
Rss = Rss * maxradius/2;
Gss = Gss * maxradius/2;
Bss = Bss * maxradius/2;

% loop over all histogram cells and plot the balls
for cnt_B = 1:n
    for cnt_G = 1:n
        for cnt_R = 1:n
            RGBfreq = freq_emph(cnt_B, cnt_R, cnt_G); %scalar
            if RGBfreq ~= 0 % if a sphere has to appear
                % begin with the initial sphere
                Rs = Rss;
                Gs = Gss;
                Bs = Bss;
                % size of the sphere according to the frequency
                ratio = RGBfreq / maxfreq;
                Rs = Rs * ratio;
                Gs = Gs * ratio;
                Bs = Bs * ratio;
                % translation the sphere to the right place
                modR = mod(cnt_R-1,n);
                modG = mod(cnt_G-1,n);
                modB = mod(cnt_B-1,n);
                Rs = Rs + (modR+0.5) * maxradius;
                Gs = Gs + (modG+0.5) * maxradius;
                Bs = Bs + (modB+0.5) * maxradius;
                % drawing
```

```
                h = surf(Rs,Gs,Bs);
                % coloring the sphere by the color taken from the center of the
respective cube
                colorR = cell_avrg_int(modR+1);
                colorG = cell_avrg_int(modG+1);
                colorB = cell_avrg_int(modB+1);
                set(h,'EdgeColor','none', ...
                    'FaceColor',[ colorR colorG colorB ]/255, ...
                    'FaceLighting','phong', ...
                    'AmbientStrength',0.7, ...
                    'DiffuseStrength',0.4, ...
                    'SpecularStrength',0.4, ...
                    'SpecularExponent',500, ...
                    'BackFaceLighting','reverselit');
                hold on
                hidden off
            end
        end
    end
end

% visualization parameters
set(gca, 'XColor', 'r','YColor', 'g', 'ZColor', 'b');
%set(gca, 'XColor', 'r','YColor', [0 0.7 0], 'ZColor', 'b');
set(gcf, 'color', 'none');
set(gca, 'color', 'none');
axis([ 0 255 0 255 0 255]);
xlabel('R');
ylabel('G');
zlabel('B');
camlight(14,36);
rotate3d on
view(14,36)
axis square

%% Compute FREQ_APP
if nargout>2
    disp('Rearranging frequencies into the structure as appeared in the
figure...')
    freq_app = zeros(n,n,n);
    for w = 1:n
        freq_app(:,:,w) = (flipud((freq(:,:,w))));
    end
end

%% End
disp('Finished.')
```

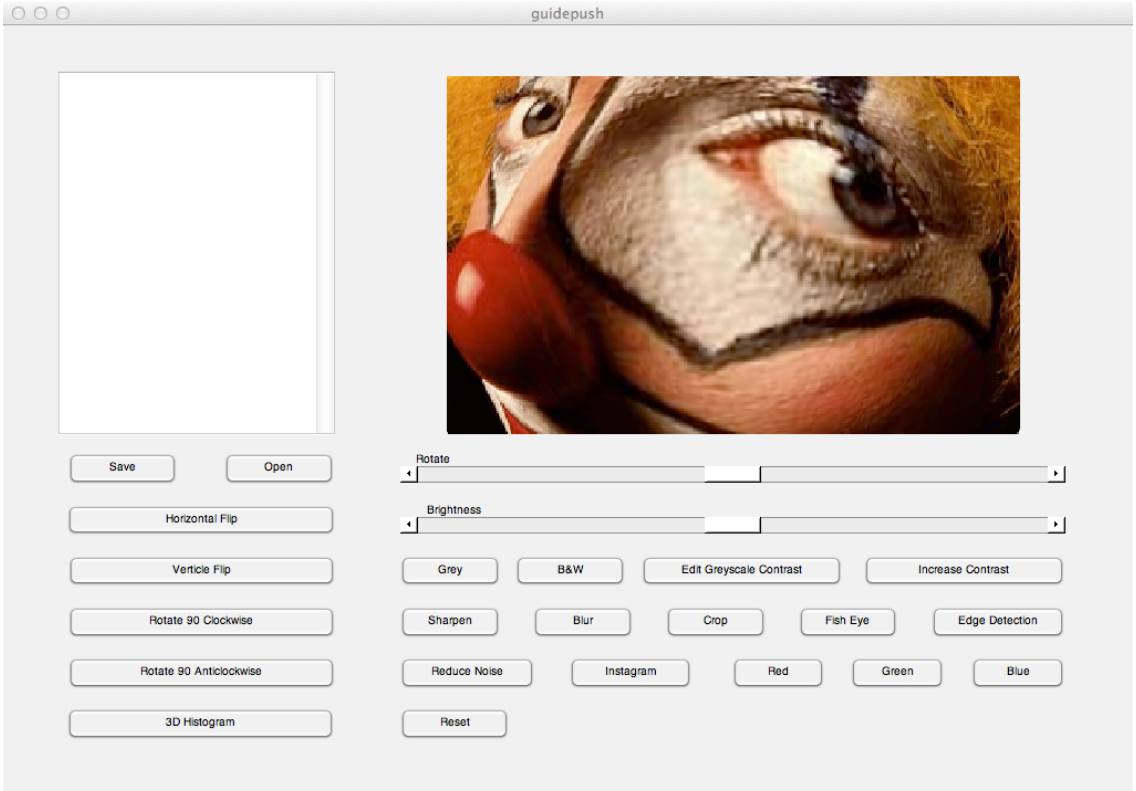
Appendix



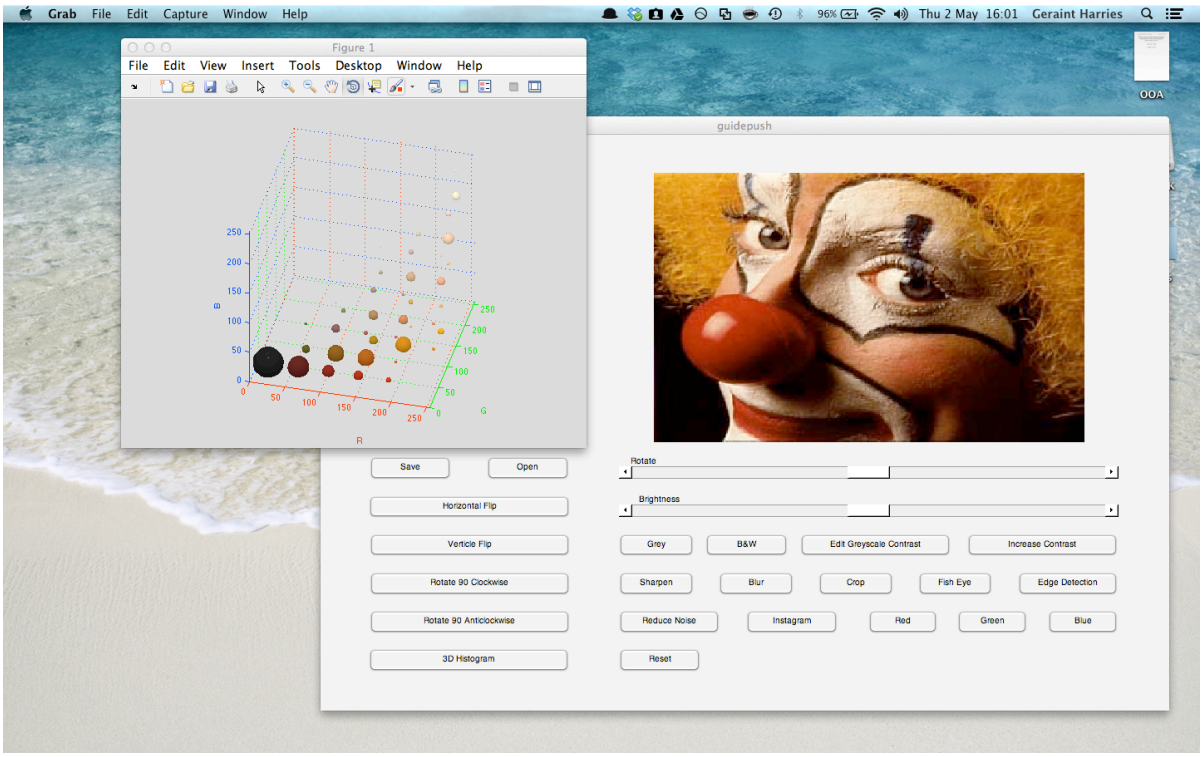
Appendix 1: Overview of system.



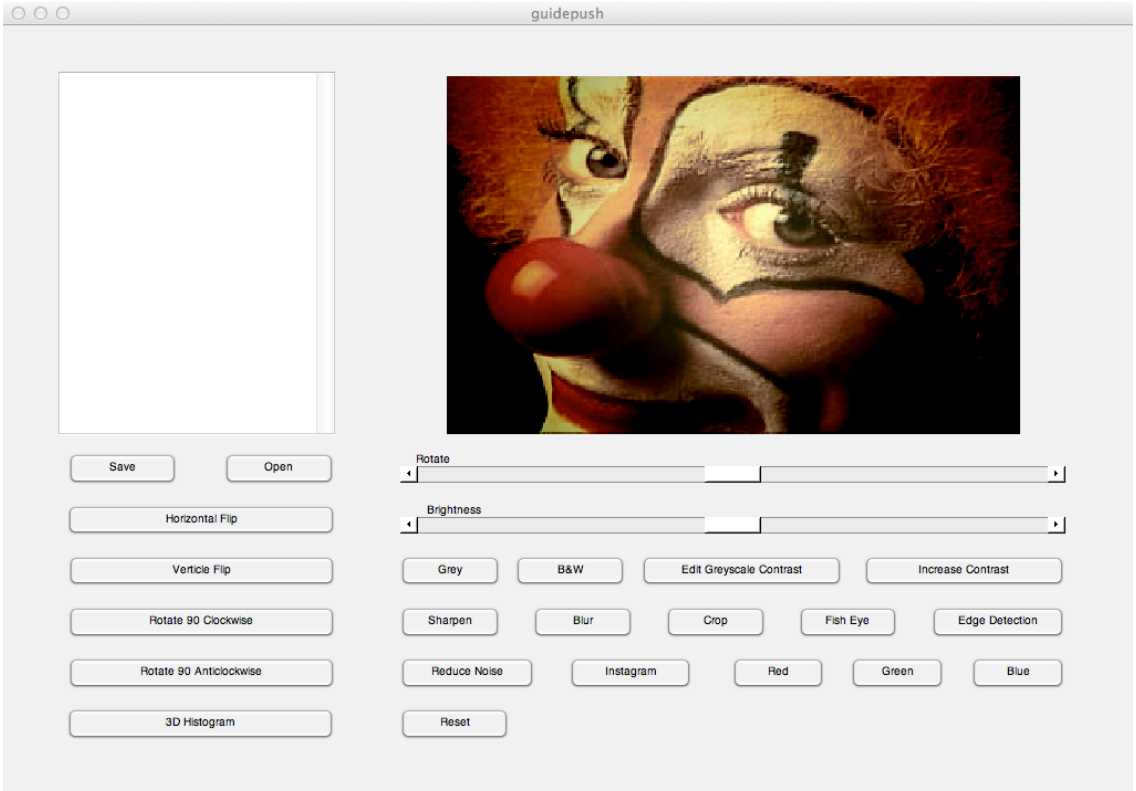
Appendix 2: Folder/ File Manager



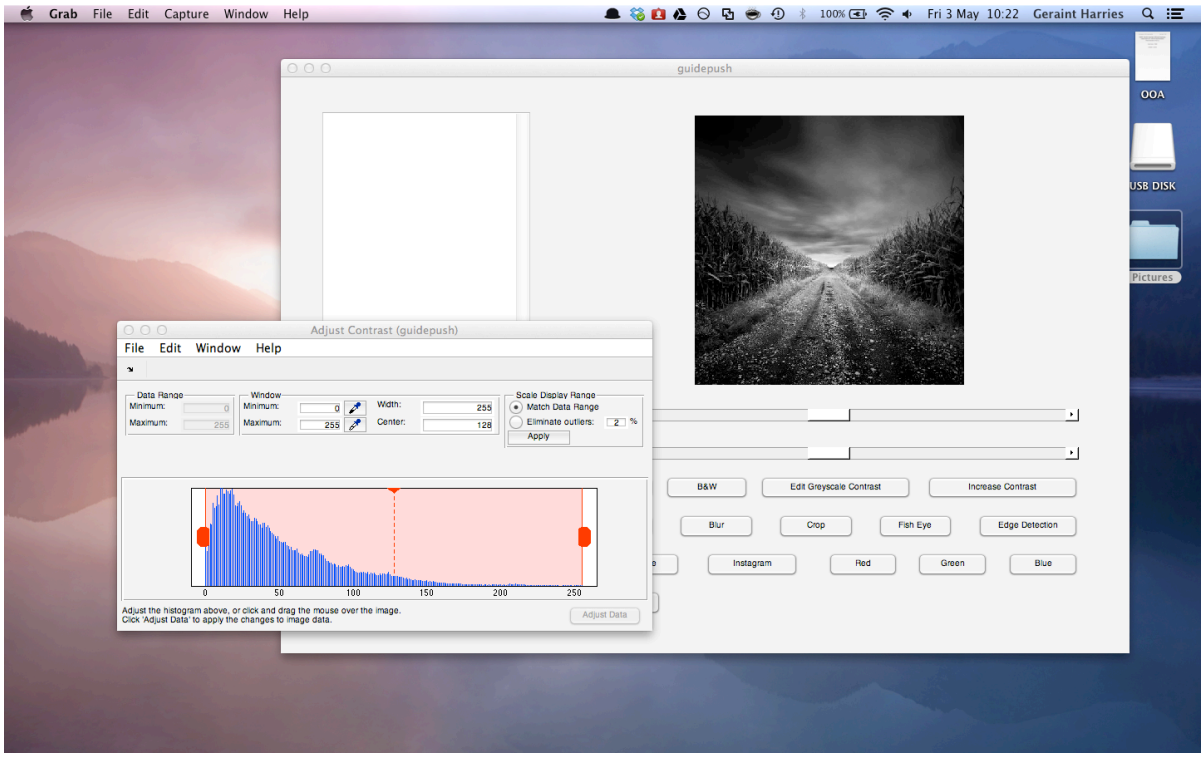
Appendix 3: Fish eye



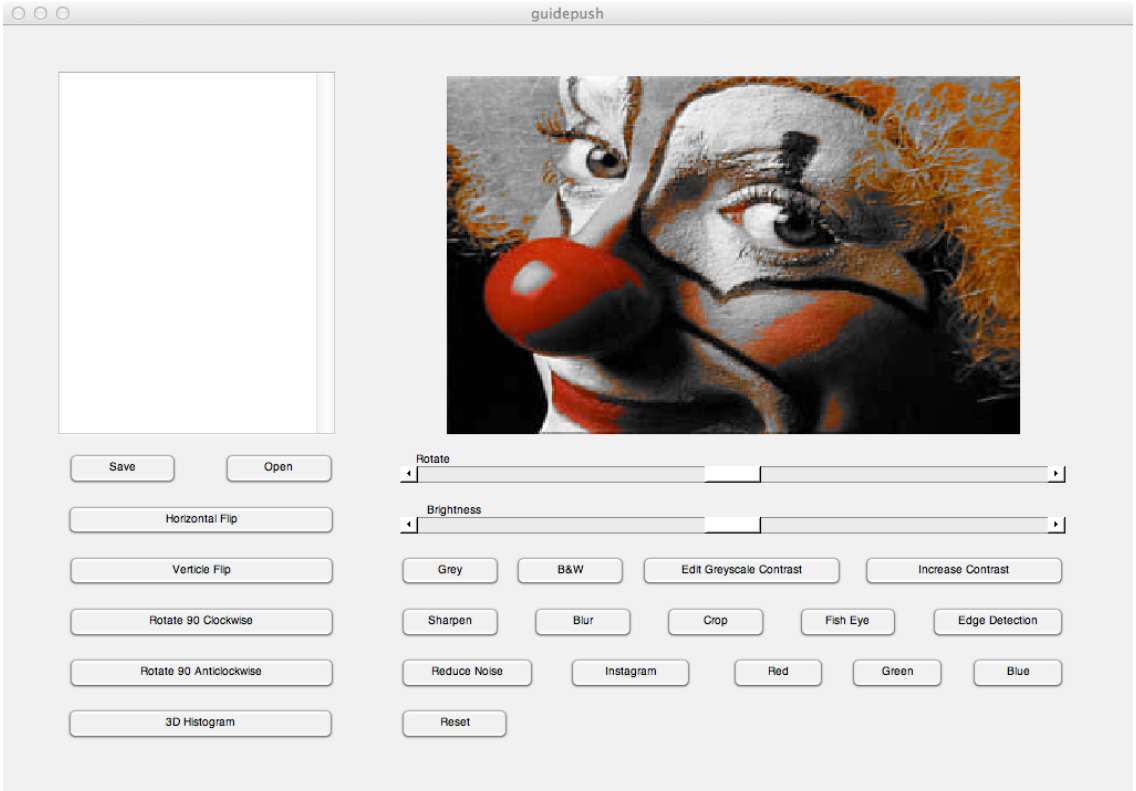
Appendix 4: 3D Histogram



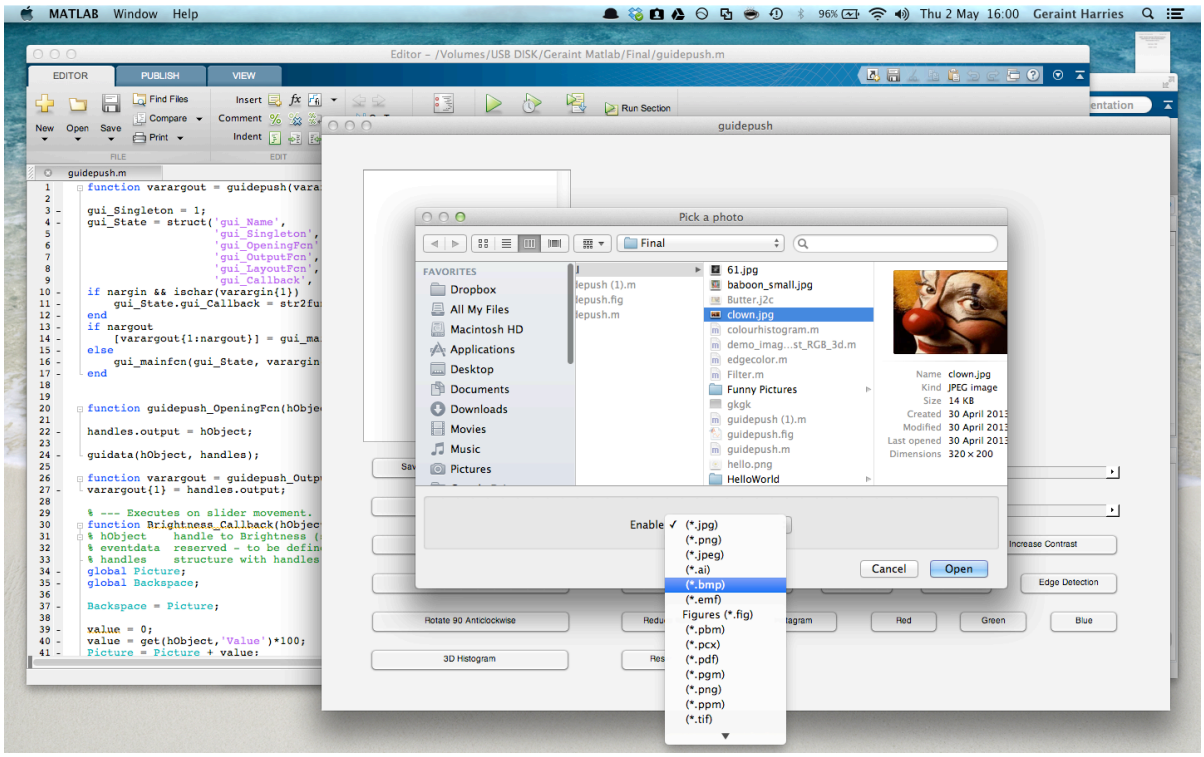
Appendix 5: Instagram



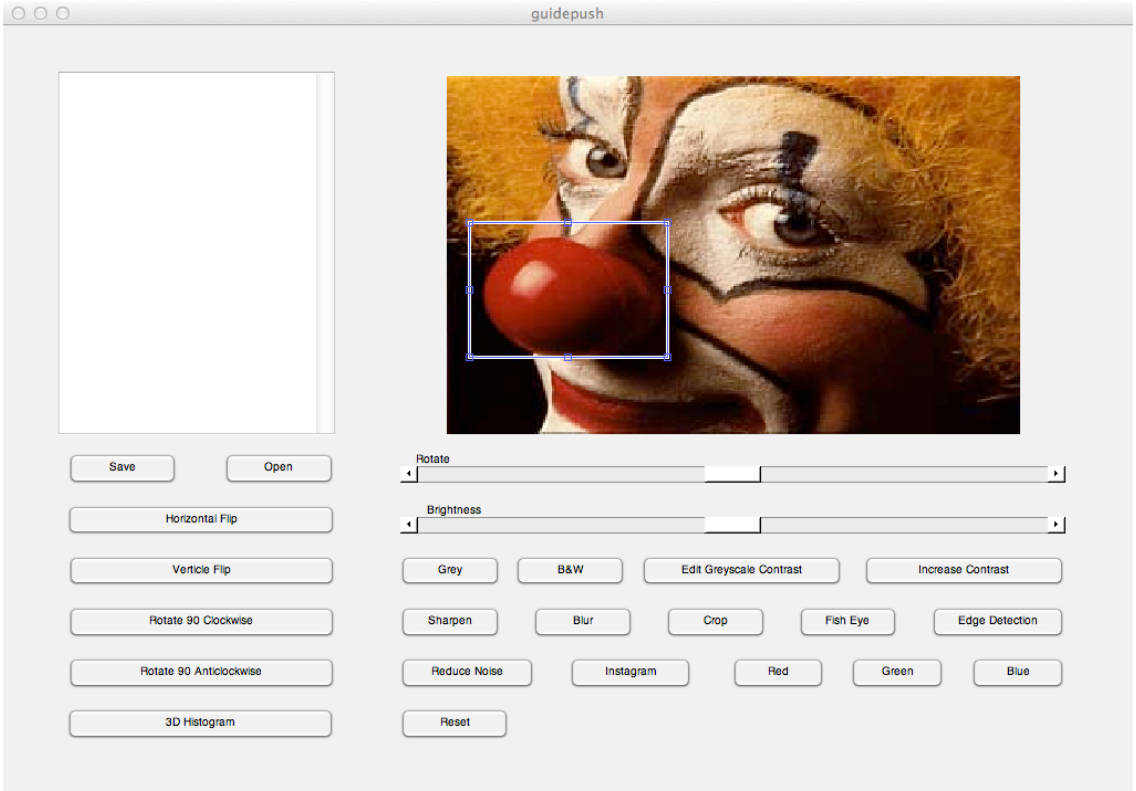
Appendix 6: Greyscale Contrast Panel



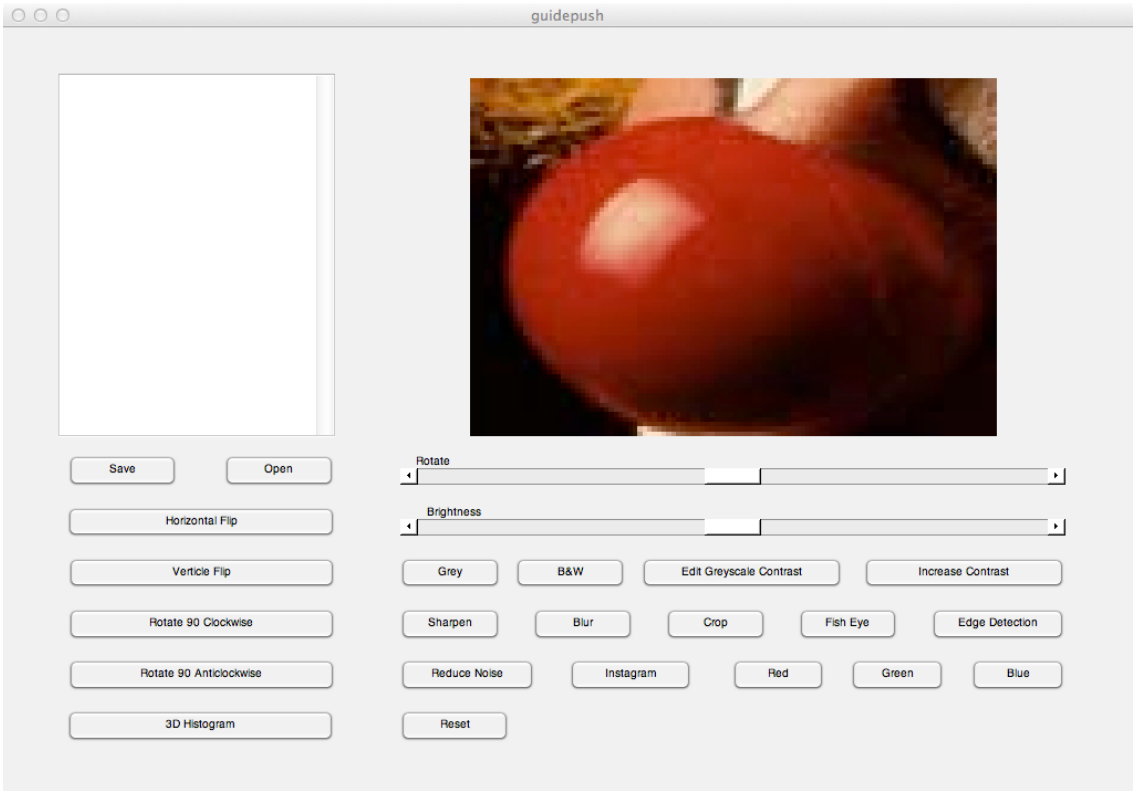
Appendix 7: Greyscale besides red



Appendix 8: Valid types to open a file



Appendix 9: Drag on crop



Appendix 10: Result of cropping

References/ Used Code

[1] Mathworks.co.uk (2013) barrel and pincushion lens distortion correction - File Exchange - MATLAB Central. [online] Available at: http://www.mathworks.co.uk/matlabcentral/fileexchange/37980-barrel-and-pincushion-lens-distortion-correction [Accessed: 28 April 2013].

[2] Mathworks.co.uk (2013) 3D histogram of RGB image - File Exchange - MATLAB Central. [online] Available at: http://www.mathworks.co.uk/matlabcentral/fileexchange/38685-3d-histogram-of-rgb-image [Accessed: 28 April 2013].

[3] Scspc006.cs.uwaterloo.ca (n.d.) Untitled. [online] Available at: http://scspc006.cs.uwaterloo.ca/files/code/retro_filter/retro_filter_vectorized.m [Accessed: 28 April 2013].

[4] Stackoverflow.com (n.d.) matlab - How can I convert an RGB image to grayscale but keep one color? - Stack Overflow. [online] Available at: http://stackoverflow.com/questions/4063965/how-can-i-convert-an-rgb-image-to-grayscale-but-keep-one-color/4064240#4064240 [Accessed: 28 April 2013].