LAPLACIAN IMAGE BLENDING

a) Problem statement: *(gPyr,lPyr) = ComputePyr(input_image,num_layers)*. Depending on the size of input_image, num_layers is to be checked if valid. If not, use the maximum value allowed in terms of the size of input_image.

Calculation of Gaussian and Laplacian pyramid:
The computation of gaussian and Laplacian pyramids is done as follows
Result:

```
Assume the base layer of the gaussian pyramid as the original size of the image.
While layer less than num_layers and largeSize greater than 1 do
| gPyr(next) = gPyr(Prev) – smooth(downscale(gpyr(prev)));
End
Lpyr(top) = gpyr(top);
While layer greater than 1 do
| lpyr(prev) = gpyr(prev) – smooth(upscale(gpyr(current)));
End
```

Algorithm 1: Gaussian/Laplacian calculation

The Gaussian Kernel is calculated using the fspecial function from Matlab. The function accepts sigma(variance) and k(kernel size) as arguments, which are calculated as follows.

$$\sigma = \frac{(2 * downscale)}{6}$$

$$K = int(truncate * \sigma + 0.5)$$

Truncate taken to be 4.0 by default.
We can tune the hyper-parameter downscale to get desired results. This can be taken as a heuristic approach to finding the optimum window size. As the transition while blending depends on the variance and kernel size of the Gaussian filter. It is important to tune these parameters to get a smooth transition. The Gaussian pyramid of the mask is taken to ensure a smooth transition between the 2 images. The optimum transition window for alpha depends on the maximum frequencies of the images A and B.

The above approach can be considered as a heuristic way to find out optimum kernel size by tuning hyper-parameters of the spatial domain gaussian kernel.

The resultant pyramids on the Lena image are as follows:

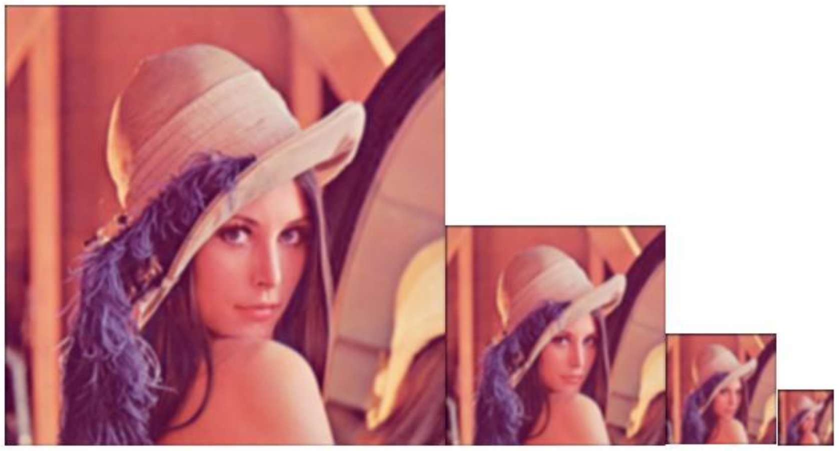


Figure 1 Gaussian pyramids with downsampling

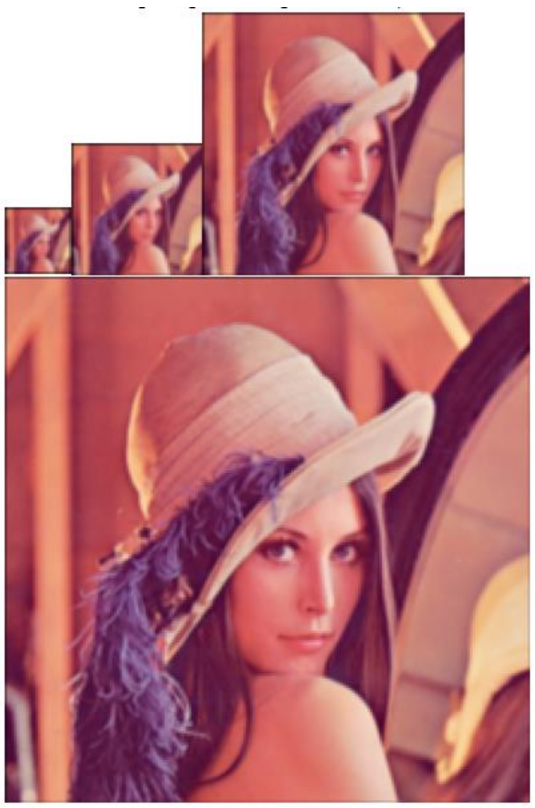


Figure 2 Laplacian pyramids with upsampling

MATLAB code to compute pyramids:

```
function [gPyr,lPyr] = ComputePyr(img1, num_layers)
    layer = 1;
    dscale = 2.5;
    sigm = (2*dscale)./6;
    kernS = floor(4*sigm+0.5);
    h = fspecial('gaussian', kernS ,sigm);
    gPyr{layer} = img1;
    while(sum(size(gPyr{layer}))~=2 && layer<=num_layers)
        gPyr{layer+1} = convfreq(gPyr{layer}, h);
        gPyr{layer+1} = downscale(gPyr{layer+1},0.5);
        layer = layer+1;
    end
    lPyr{layer}=gPyr{layer};
    while layer>1
        lPyr{layer-1} = gPyr{layer-1}-convfreq(upscale(gPyr{layer},size(gPyr{layer-1})), h);
%         figure;
%         imshow(lPyr{layer});
        layer = layer-1;
    end
end
```

Helper functions:

Nearest Neighbor interpolation performed to upscale and downscale images through the pyramids. The upscale function uses newSize as argument to avoid dimension mismatch while blending the images. (Happens Due to integer rounding).

Downsampling:

```
function [ResampledImage] = downscale(InputImage, ScalingFactor)
%   Resize the given image according to the given scaling factor, implement
%   nearest neighbor interpolation while upsampling to sample the images.

scale = [ScalingFactor, ScalingFactor];
oldSize = size(InputImage);
newSize = max(floor(scale.*oldSize(1:2)),1); %to avoid 0 index error
% nearest neighbor interpolation.
% calculate the new row and column indices for assignment of pixels
rowIndices = min(round(((1:newSize(1))-0.5)./scale(1)+0.5), oldSize(1));
colIndices = min(round(((1:newSize(2))-0.5)./scale(2)+0.5), oldSize(2));
ResampledImage = InputImage(rowIndices, colIndices,:);
end
```

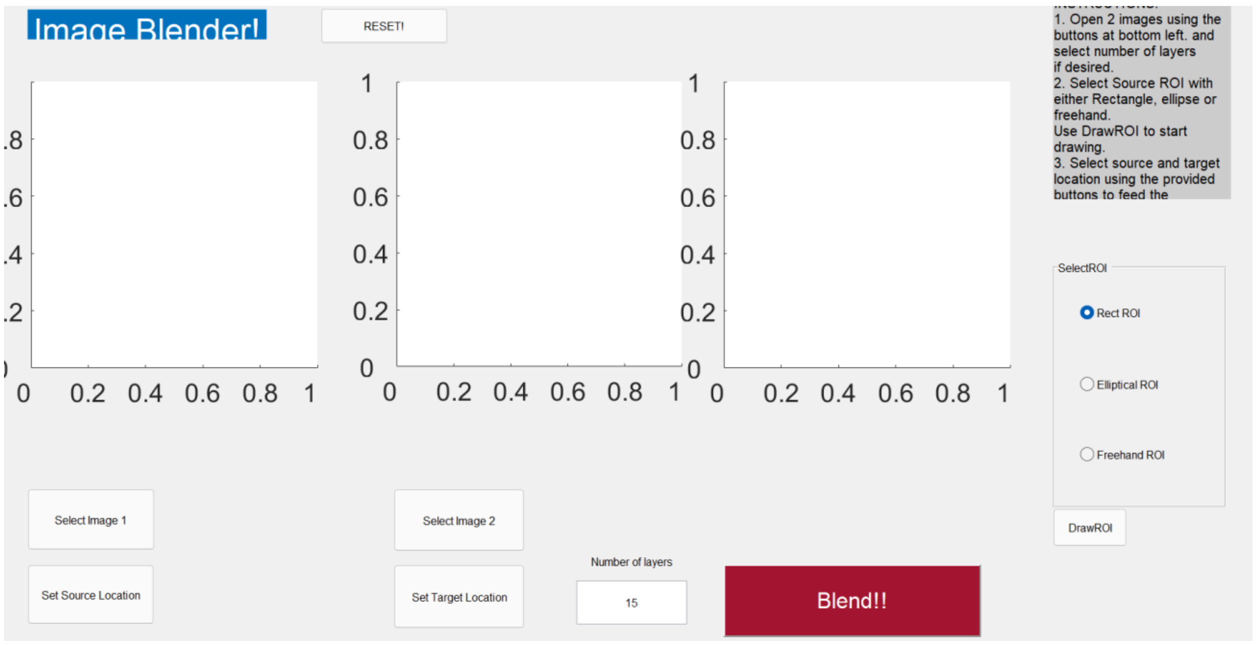
Upsampling:

```
function [ResampledImage] = upscale(InputImage,newSize)
oldSize = size(InputImage);
scale = newSize./oldSize;
%subtract -0.5 to consider center of pixel. Add 0.5 to negate subtraction
%min function used to avoid any overflows
rowIndices = min(round(((1:newSize(1))-0.5)./scale(1)+0.5), oldSize(1));
colIndices = min(round(((1:newSize(2))-0.5)./scale(2)+0.5), oldSize(2));
%assignment based on the newly calculated indices
```

```
ResampledImage = InputImage(rowIndices, colIndices,:);
End
```

b) GUI to make a mask

GUI through which the user can interact with the program to select images, Region of interest and the locations for blending. The readme file has been attached and the GUI also has instructions written on it. Select Source and Target location buttons are used to calculate the vector used to align the source and target images. This can be treated as a user input where user points out the location of target and source if he desires specific areas to be blended. Number of layers can be entered in the text box and the Select ROI button group can be used to select the desired Region of interest the user desires to make for mask creation.

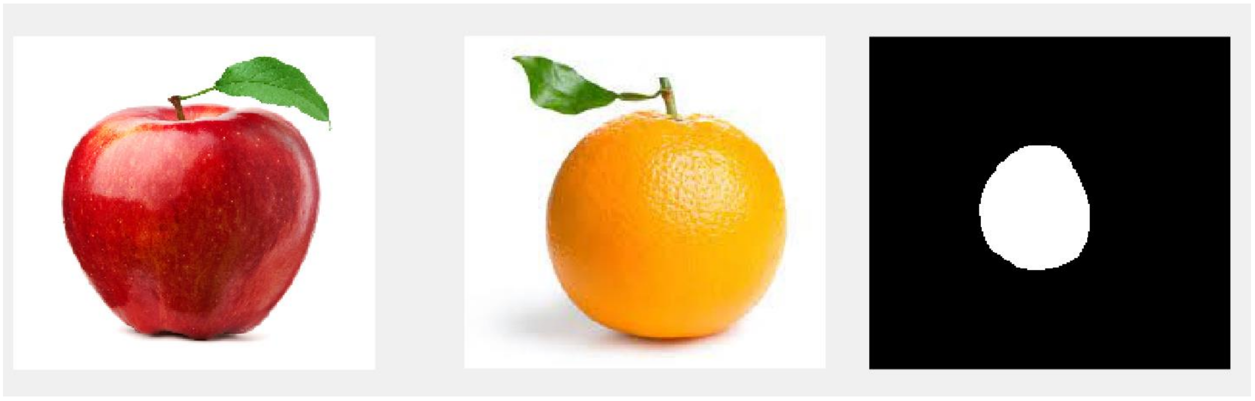


1. Open 2 images using the buttons at bottom left. and select number of layers if desired.
2. Select Source ROI with either Rectangle, ellipse or freehand.
Use DrawROI to start drawing.
3. Select source and target location using the provided buttons to feed the

Code:

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

% Last Modified by GUIDE v2.5 13-Nov-2019 11:25:45

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

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

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using Blend.
if strcmp(get(hObject,'Visible'),'off')
%     plot(rand(5));
end

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


% --- Outputs from this function are returned to the command line.
function varargout = Blend_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;

% --------------------------------------------------------------------
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to OpenMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end
% --- Executes on button press in Img1.
% --- read images selected by the user
function Img1_Callback(hObject, eventdata, handles)
% hObject    handle to Img1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    [img1, path1, ~] = uigetfile('*.jpg; *.png; *.jpeg', 'Pick a source image');
    if ~isequal(img1, 0)
        fg = im2double(imread(horzcat(path1,img1)));
    end
    axes(handles.Source);
    cla;
    fg1 = fg;
    hObject.UserData = fg;
    imshow(fg)

% --- Executes on button press in Img2.
% --- read images selected by the user
function Img2_Callback(hObject, eventdata, handles)
% hObject    handle to Img2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    [img2, path2, ~] = uigetfile('*.jpg; *.png; *.jpeg', 'Pick a source image');
    if ~isequal(img2, 0)
        bg = im2double(imread(horzcat(path2,img2)));
    end
    axes(handles.Target);
    cla;
    imshow(bg)
    hObject.UserData = bg;

% --- Executes during object creation, after setting all properties.
function Img2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Img2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in Rect.
% --- select rectangular ROI
function Rect_Callback(hObject, eventdata, handles)
% hObject    handle to Rect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    el = findobj('Tag', 'Ellipse');
    fr = findobj('Tag', 'FreeHnd');
    crt = findobj('Tag', 'DrawROI');
    axes(handles.Source);
    try
        roi = crt.UserData{3};
        delete(roi);
    catch
    end
    el.Value=0;
    fr.Value=0;


% --- Executes on button press in Ellipse.
% --- select elliptical ROI
function Ellipse_Callback(hObject, eventdata, handles)
% hObject    handle to Ellipse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    re = findobj('Tag', 'Rect');
    fr = findobj('Tag', 'FreeHnd');
    crt = findobj('Tag', 'DrawROI');
    axes(handles.Source);
    try
        roi = crt.UserData{3};
        delete(roi);
    catch
    end
    re.Value=0;
    fr.Value=0;

% --- Executes on button press in FreeHnd.
% --- select freehand ROI
function FreeHnd_Callback(hObject, eventdata, handles)
% hObject    handle to FreeHnd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    re = findobj('Tag', 'Rect');
    el = findobj('Tag', 'Ellipse');
    crt = findobj('Tag', 'DrawROI');
    axes(handles.Source);
    try
        roi = crt.UserData{3};
        delete(roi);
    catch
    end
    re.Value=0;
    el.Value=0;

% --- Executes on button press in Blnd.
% --- Blend the selected images based on selected ROI
function Blnd_Callback(hObject, eventdata, handles)
% hObject    handle to Blnd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    h1 = findobj('Tag', 'Img1');
    h2 = findobj('Tag', 'Img2');
    nl = findobj('Tag', 'nlayers');
    msk = findobj('Tag', 'DrawROI');
    trg = findobj('Tag', 'SetTrgtLocn');
    bg = h2.UserData;
    try
        fg1 = trg.UserData{1};
        bw_mask = trg.UserData{2};
    catch
        fg1 = msk.UserData{1};
        bw_mask = msk.UserData{2};
    end
    %handle RGB images.
    if size(bg,3)>1
        blendedImg(:,:,1) = blendPyramid(fg1(:,:,1),bg(:,:,1),bw_mask,str2double(nl.String));
        blendedImg(:,:,2) = blendPyramid(fg1(:,:,2),bg(:,:,2),bw_mask,str2double(nl.String));
        blendedImg(:,:,3) = blendPyramid(fg1(:,:,3),bg(:,:,3),bw_mask,str2double(nl.String));
    else
        blendedImg(:,:,1) = blendPyramid(fg1(:,:,1),bg(:,:,1),bw_mask,str2double(nl.String));
    end
%     blendedImg = ScaleRGBValues(blendedImg);
    axes(handles.Source);
    imshow(h1.UserData);
    axes(handles.Target);
    imshow(bg);
    axes(handles.Outputs);
    imshow(blendedImg);
    imwrite(blendedImg, 'blend.png');

% --- Executes on button press in SetTrgtLocn.
function SetTrgtLocn_Callback(hObject, eventdata, handles)
% hObject    handle to SetTrgtLocn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    droi = findobj('Tag', 'DrawROI');
    fgp = droi.UserData{1};
    bwm = droi.UserData{2};
    src = findobj('Tag', 'SetSrcLocn');
    axes(handles.Target);
    trgtLoc = drawrectangle();
    trgtPos = trgtLoc.Position;
    try
        srcPos = src.UserData;
        transVec = trgtPos(1:2)-srcPos(1:2);
    catch
        transVec = [0,0];
    end
    %translate image to location selected by the user.
    fgp=imtranslate(fgp, transVec);
    bwm = imtranslate(bwm, transVec);
    axes(handles.Source);
    imshow(fgp);
    axes(handles.Outputs);
    imshow(bwm);
    hObject.UserData{1} = fgp;
    hObject.UserData{2} = bwm;
    imwrite(bwm, 'mask.png');
    imwrite(fgp, 'fgPadded.png');
    delete(trgtLoc);

% --- Executes on button press in SetSrcLocn.
function SetSrcLocn_Callback(hObject, eventdata, handles)
% hObject    handle to SetSrcLocn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    axes(handles.Source);
    srcLoc = drawrectangle();
    srcPos = srcLoc.Position;
    hObject.UserData = srcPos;
    delete(srcLoc);

function nlayers_Callback(hObject, eventdata, handles)
% hObject    handle to nlayers (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,'String') returns contents of nlayers as text
%        str2double(get(hObject,'String')) returns contents of nlayers as a double

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

% Hint: edit 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 DrawROI.
% --- draw ROI based on selected region type

function DrawROI_Callback(hObject, eventdata, handles)
% hObject    handle to DrawROI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    h1 = findobj('Tag', 'Img1');
    h2 = findobj('Tag', 'Img2');
    re = findobj('Tag', 'Rect');
    el = findobj('Tag', 'Ellipse');
    fr = findobj('Tag', 'FreeHnd');
    src = findobj('Tag', 'SetSrcLocn');
    trg = findobj('Tag', 'SetTrgtLocn');
    fg1 = trg.UserData;
    if isempty(fg1)
        fg1 = h1.UserData;
    end
    bg = h2.UserData;
    axes(handles.Source);
    if re.Value == 1 && el.Value == 0 && fr.Value == 0
        [fg1, roi, bw_mask] = createROIMask(fg1, bg, 1);
    elseif re.Value == 0 && el.Value == 1 && fr.Value == 0
        [fg1, roi, bw_mask] = createROIMask(fg1, bg, 2);
    else
        [fg1, roi, bw_mask] = createROIMask(fg1, bg, 3);
    end
    axes(handles.Outputs);
    imshow(bw_mask);
    axes(handles.Source);
    imshow(fg1);
    imwrite(bw_mask, 'mask.png');
    imwrite(fg1, 'fgPadded.png');
    hObject.UserData{1} = fg1;
    hObject.UserData{2} = bw_mask;
    hObject.UserData{3} = roi;


% --- 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)
    close(Blend);
    run('Blend');
```

Results of mask creation:



c)

Convolution and Padding functions from project 2 used. The convolution can be performed as a separable 1D convolution using 2 loops instead of nested loops which can improve the performance by a tad bit. In this case the convolution is with 3x3 gaussian filters generated with the fspecial command, hence the same function can be used to perform the convolution. Moreover, a faster approach can be moving to frequency domain, and multiplying and again moving back to spatial domain. Convfreq function has been used for convolution in frequency domain.

Convolution in frequency domain:

```
function [respo] = convfreq(img,kern)
```

```
% multplication in freq domain = conv in time domain
% using that for simplicity
    [rk,ck,chk]=size(kern);
    [r,c,ch] = size(img);
    zpr = r+2*floor(rk/2); zpc = c+2*floor(ck/2);
    imgs = zeros(zpr,zpc);
    imgs(floor(zpr/2)+(1:r)-floor(r/2), floor(zpc/2)+(1:c)-floor ...
        (c/2),:) = img;
    imgfreq = dft2(fftshift(imgs));
    kernfreq = zeros(zpr,zpc);
    kernfreq(floor(zpr/2)+(1:rk)-floor(rk/2), floor(zpc/2)+(1:ck)-floor ...
        (ck/2),:) = kern;
    kernfreq = dft2(kernfreq);
    opfreq = imgfreq.*kernfreq;
    resp = real(idft2(opfreq));
    respo = resp(1+floor(rk/2):end-floor(rk/2),1+floor(ck/2):end-floor(ck/2));
end
```

As with any convolution this requires padding of the image to handle edges and corners. The padding code from project 2 has been modified to handle all odd kernel sizes and exceptions as 1x1 and 2x1 image sizes. The padding code SetPadding.m has been attached in the .zip which can handle the all 4 types of padding. b) Create binary mask - Rectangular, Elliptical or Freeform. The image resizing and padding has been handled. The user needs to point out source and target location wherever required to align the faces/ source-target pair properly. The following code has been used to create a mask of desired size with required resize and reshape of the fore ground image as and when it was required.

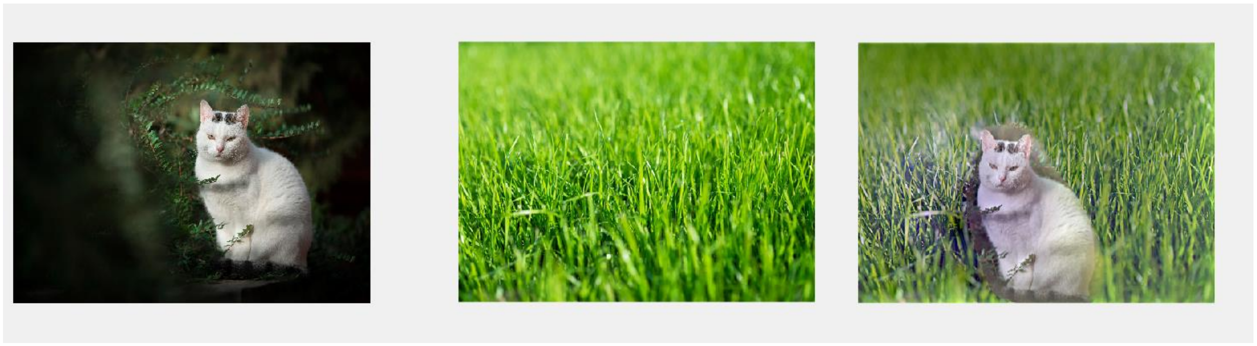
```
function [fgo, roi, bw_mask1] = createROIMask(fg, bg, roitype)
%   has 3 options - rectangular, elliptical and freeform.
    %check roitype
    if(roitype == 1)
        roi = drawrectangle('Label', 'ROI', 'color', [1 0 0]);
    elseif(roitype == 2)
        roi = drawellipse('Label', 'ROI', 'color', [1 0 0]);
    else
        roi = drawfreehand('Label', 'ROI', 'color', [1 0 0]);
    end
    %to handle difference in sizes of images
    %create mask from the selected ROI
    bw_mask = createMask(roi);
    [rb, cb, bch] = size(bg);
    [rf, cf, fch] = size(fg);
    %resize/ pad fg based on its original size.
    if(rf*cf>rb*cb)
        fgo=upscale(fg, [rb,cb,bch]);
        bw_mask1 = upscale(bw_mask, [rb,cb]);
    elseif(rf*cf<rb*cb)
        fgo = zeros(rb,cb,bch);
        bw_mask1 = zeros(rb,cb);
        if rf<rb && cf<cb
            % center the smaller fg and its mask by zero padding to bg size
            fgo(floor(rb/2)+(1:rf)-floor(rf/2), floor(cb/2)+(1:cf)-floor (cf/2),:) = fg;
            bw_mask1(floor(rb/2)+(1:rf)-floor(rf/2), floor(cb/2)+(1:cf)-floor(cf/2)) = bw_mask;
        elseif rf<rb && cf>cb
            fac = 1-((cf-cb)/cb);
            fg = downscale(fg,fac);
            bw_mask = downscale(bw_mask,fac);
            [rf, cf, fch] = size(fg);
            % center the smaller fg and its mask by zero padding to bg size
            fgo(floor(rb/2)+(1:rf)-floor(rf/2), floor(cb/2)+(1:cf)-floor (cf/2),:) = fg;
            bw_mask1(floor(rb/2)+(1:rf)-floor(rf/2), floor(cb/2)+(1:cf)-  floor(cf/2)) = bw_mask;
        elseif rf>rb && cf<cb
            fac = 1-((rf-rb)/rb);
            fg = downscale(fg,fac);
            bw_mask = downscale(bw_mask,fac);
            [rf, cf, fch] = size(fg);
            % center the smaller fg and its mask by zero padding to bg size
            fgo(floor(rb/2)+(1:rf)-floor(rf/2), floor(cb/2)+(1:cf)-floor (cf/2),:) = fg;
            bw_mask1(floor(rb/2)+(1:rf)-floor(rf/2), floor(cb/2)+(1:cf)- floor(cf/2)) = bw_mask;
        end
    else
        fgo = fg;
        bw_mask1 = bw_mask;
    end
end
```

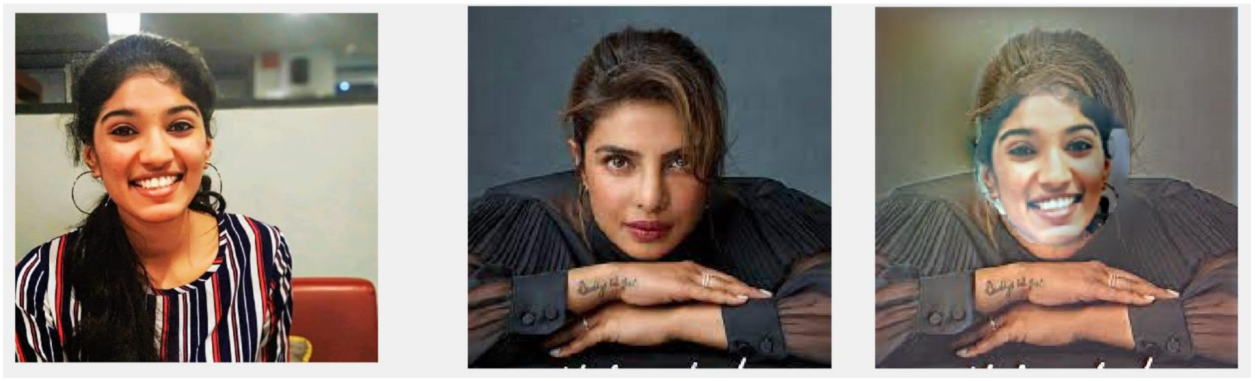
Image blending was done using the following code using the following algorithm:
Blending of images and alignment. The blending can be treated as α blending where the α can be taken as the Gaussian pyramid of the mask. The formula for blending can be taken as follows
$blendedImg(i) = lP\,yrA(i) * gP\,yr(i) + (1 - gP\,yr(i)) * lP\,yrB(i)$
We do the operations layer by layer and then collapse the pyramid.

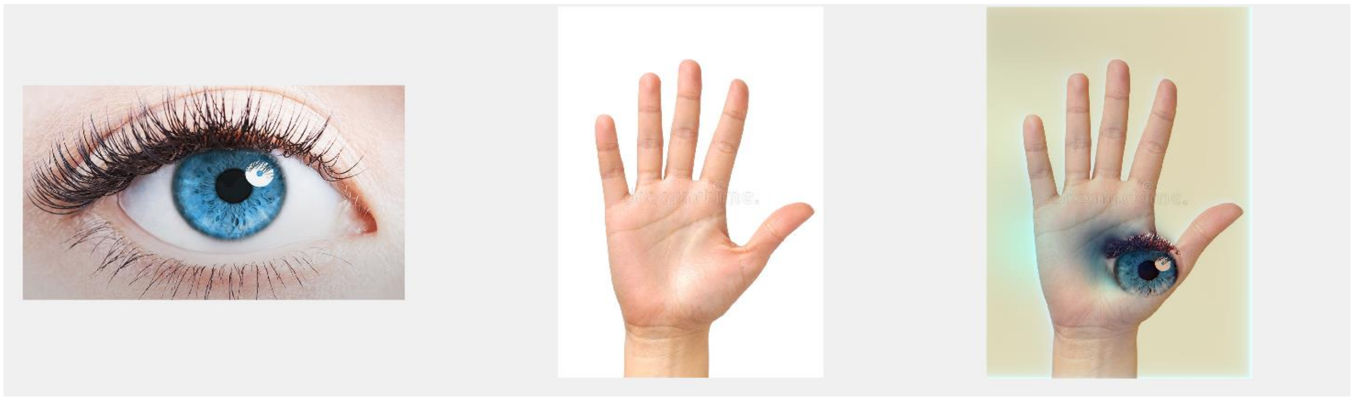
```
function [blendedImg] = blendPyramid(bg, fg, bw_mask, nlayers)
    [~, lPyrBG] = ComputePyr(bg, nlayers); %calculate laplacian pyr of 1st img
    [~, lPyrFG] = ComputePyr(fg, nlayers); %calculate laplacian pyr of 2nd img
    [gPyrMask,~] = ComputePyr(double(bw_mask), nlayers); %gaussian pyr of mask
    for i = 1:length(gPyrMask)
        %calculate blended pyramid based on alpha blending with mask gpyr
        lblend{i} = (gPyrMask{i}) .* lPyrBG{i} + (1-(gPyrMask{i})).*lPyrFG{i};
    end
    layer = length(lblend);
    %generate gaussian kernel
    dscale = 2.5;
    sigm = (2*dscale)./6;
    kernS = floor(4*sigm+0.5);
    h = fspecial('gaussian', kernS ,sigm);
    %collapse pyramid
    while(layer>1)
        lblend{layer-1} = convfreq(upscale(lblend{layer}, size(lblend{layer-1})),h)+lblend{layer-1};
        layer = layer-1;
    end
    blendedImg = lblend{1};
end
```

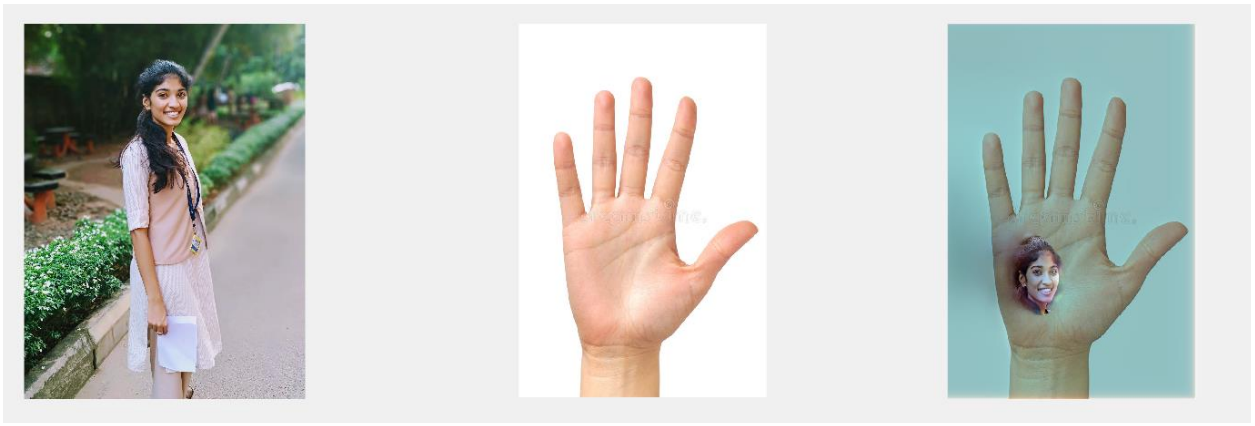
Results:

When I tried blending a 1024x 576 image of a cat with 1024 x 1024 image of grass, the resultant image has a 1024x 1024 size same as that of the background.









Conclusion:

This project delivers an end to end image blending system which can be operated through a GUI as discussed in the second section of the report and the third section of the report discusses on how the image blending is performed using the gaussian and Laplacian pyramids from the first section of the report.