**Chapter 6**

**Implementation**

**6.1 Algorithm**

Step 1: Stained and magnified human blood smear image is fed to the program as input.

Step 2: Input image is converted to grayscale and is enhanced by performing histogram stretching.

Step 3: The void spaces within the cells of the enhanced image are filled using the imfill() command. Use of imfill() would require the image to be complemented before and after the operation since the function only fills white into blacks spaces bounded by white boundaries and not vice-versa.

Step 4: The resulting image is then converted into a binary image.

Step 4: The image is complemented and small unwanted spots are discarded from the image using ‘bwareaopen’, which removes all the objects in the diagram containing fewer than the number of pixels mentioned in the threshold level (here, threshold level= 50). The image is then complemented back to its original form for further processing.

Step 5: The cells/objects intersecting the borders of the image would produce inaccurate results. Therefore, such cells are discarded using imclearborder().

Step 6: Individual cells are detected by using bwconncomp() command in Matlab. bwconncomp() returns a structure of connected components in the image passed to it. This structure is then passed to regionprops() to obtain the area and perimeter of every cell.

Step 7: The number of valid cells are computed by considering only the cells falling within the quartile range.

Step 8: The metric value for every valid cell is computed and stored. Metric value is computed using the formula 4\*PI\*area/(perimeter)2.

Step 9: The valid cells are further classified as normal and abnormal cells depending on the metric value corresponding to the cells. A threshold metric value is decided (usually above 0.75), above which a cell is classified as normal.

Step 10: A threshold is decided depending on which a decision is made whether the individual is diagnosed with the disease or not. If the number of abnormal cells is equal to or more than the threshold, a positive SCA result is displayed.

**6.2 Working of the project**

ASCAD.fig

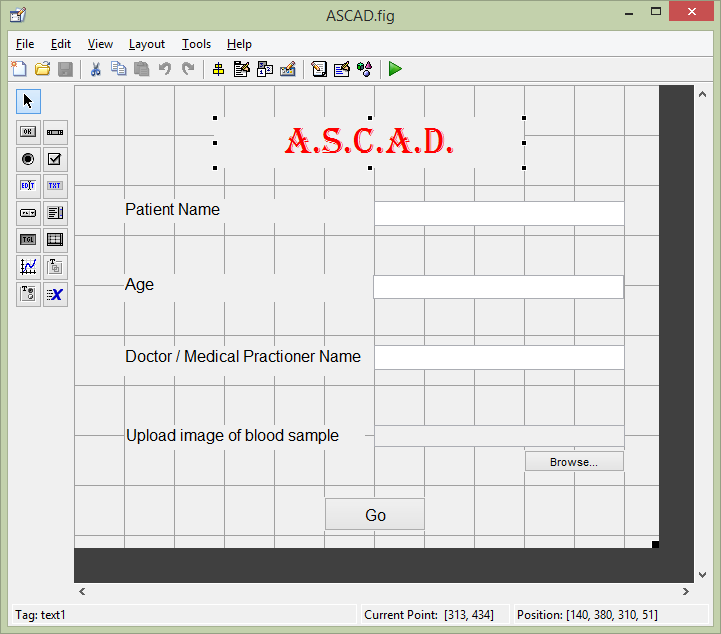


Fig. 6.2.1 GUI layout

ASCAD.m

function varargout = ASCAD(varargin)

% ASCAD MATLAB code for ASCAD.fig

% ASCAD, by itself, creates a new ASCAD or raises the existing

% singleton\*.

%

% H = ASCAD returns the handle to a new ASCAD or the handle to

% the existing singleton\*.

%

% ASCAD('CALLBACK',hObject,eventData,handles,...) calls the local

% function named CALLBACK in ASCAD.M with the given input arguments.

%

% ASCAD('Property','Value',...) creates a new ASCAD or raises the

% existing singleton\*. Starting from the left, property value pairs are

% applied to the GUI before ASCAD\_OpeningFcn gets called. An

% unrecognized property name or invalid value makes property application

% stop. All inputs are passed to ASCAD\_OpeningFcn via varargin.

% Last Modified by GUIDE v2.5 07-Apr-2015 16:20:01

% Begin initialization code - DO NOT EDIT

gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ...

'gui\_OpeningFcn', @ASCAD\_OpeningFcn, ...

'gui\_OutputFcn', @ASCAD\_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 ASCAD is made visible.

function ASCAD\_OpeningFcn(hObject, eventdata, handles, varargin)

% 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 ASCAD (see VARARGIN)

% Choose default command line output for ASCAD

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

% UIWAIT makes ASCAD wait for user response (see UIRESUME)

% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.

function varargout = ASCAD\_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure

varargout{1} = handles.output;

function P\_name\_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.

function P\_name\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function P\_age\_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.

function P\_age\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function D\_name\_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.

function D\_name\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function img\_path\_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.

function img\_path\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in pushbutton1.

function pushbutton1\_Callback(hObject, eventdata, handles)

handles.output = hObject;

[fn pn] = uigetfile('\*.jpg','select dicom file');

complete = strcat(pn,fn);

set(handles.img\_path,'string',complete);

% --- Executes on button press in Go.

function Go\_Callback(hObject, eventdata, handles)

P= get(handles.P\_name, 'String');

A= str2num(get(handles.P\_age, 'String'));

D= get(handles.D\_name, 'String');

IP= get(handles.img\_path, 'String');

if isempty(P) % checking that the patient name field is not left blank

errordlg('Please enter the Patient Name.', 'Error');

else if isempty(A) % checking that the patient age is not left blank

errordlg('Please enter the age of the patient.', 'Error');

else if A<0 % checking if the user has entered a negative age

errordlg('Please enter a valid age of the patient.', 'Error');

else if isempty(D);

errordlg('Please enter the name of the doctor or medical practioner who advised the test. Enter N.A. in case of self test.', 'Error');

else if isempty(IP)

errordlg('Image path empty. Please browse to the input image.','Error');

else

Code(get(handles.P\_name, 'String'), get(handles.P\_age, 'String'), get(handles.D\_name, 'String'), get(handles.img\_path, 'String'));

end

end

end

end

end

Code.m

function Code(P, A, D, IP)

warning off;

P\_name= P; %input('Enter Patient name : ','s');

P\_age= A; %input('Enter Patient age : ');

D\_name= D; %input('Enter Doctor (medical practioner) name : ','s');

%img\_input = input('Enter the image index : ');

img\_path= IP; %strcat('C:\AAA\Img\',int2str(img\_input),'.jpg');

img= imread(img\_path);

%figure, imshow(img);

img\_grayS = rgb2gray(img);

%figure, imshow(img\_grayS);

%histogram stretching

bins = linspace(0,255,256);

H = hist(img\_grayS(:), bins);

H(H==0) = eps(sum(H));

cdf = [0,cumsum(H)/sum(H)]; %cumulative distribution function

pct= 0.05; %percent of pixel values to ignore

h\_low = interp1(cdf, [0,bins], pct);

h\_high = interp1(cdf, [0,bins], 1-pct);

stretchedImg= uint8((double(img\_grayS)-h\_low)/(h\_high-h\_low) \* 255);

%img\_grayS= uint8(imadjust(img\_grayS, stretchlim(img\_grayS), [])); %histogram stretching using inbuilt function

%figure, imshow(stretchedImg);

BW\_c= imcomplement(stretchedImg);

BW\_filled= imfill(BW\_c,4, 'holes');

BW\_refined= imcomplement(BW\_filled);

%figure, imshow(BW\_refined);

%Gaussian filter

myfilter = fspecial('gaussian',[3 3],16);

filteredImg = imfilter(BW\_refined, myfilter, 'replicate');

img\_binary= im2bw(BW\_refined,graythresh(filteredImg)); % converting the enhanced image to binary

%figure, imshow(img\_binary);

img\_binary\_c= imcomplement(img\_binary); %bwareaopen and bwconncomp works only on bright pixels

refined\_img= bwareaopen(img\_binary\_c,200, 4); %removing unwanted spots from the image having a maximum pixel density of 200 pixels

%figure, imshow(refined\_img);

%EROSION (EXPERIMENTAL)

se = strel('disk',2);

erodedBW = imerode(refined\_img,se); %Shrinks each 'disk' shaped object in the image

%figure, imshow(erodedBW);

img\_binary1 = imclearborder(erodedBW); %remove border objects

%figure, imshow(img\_binary1);

CC= bwconncomp(img\_binary1, 4); % keeping connectivity 4 to ignore diagonal connection

CA= regionprops(CC, 'Area'); %returns the areas of all objects in the image identified by bwconncomp()

CP= regionprops(CC, 'Perimeter'); %returns the perimeters of all objects in the image identified by bwconncomp()

img\_binary2= imcomplement(img\_binary1);

areas= cell(CC.NumObjects,1); %storing the area of individual components from CA.Area

%CA.Area is not feasible to use for computations

for i=1: CC.NumObjects

areas{i,1}= CA(i,1).Area;

end

areas= cell2mat(areas); %converting the cell to array

iqr= quantile(areas,0.75)-quantile(areas,0.25); %inter quartile range

average=iqr\*1.5; %going by the definition of an outlier

%counting the number of valid and invalid cels in the image (only considering the objects with areas within the inter quartile range)

validCells=0; invalidCells=0;

for i=1:CC.NumObjects

if CA(i,1).Area >= quantile(areas,0.25)-average && CA(i,1).Area <= quantile(areas,0.75)+ average

validCells= validCells+1;

else

invalidCells= invalidCells+1;

end

end

%calculating the metric value for each valid cell

metric= cell(validCells ,1);

k=1;

for i=1:CC.NumObjects

if CA(i,1).Area >= quantile(areas,0.25)-average && CA(i,1).Area <= quantile(areas,0.75)+ average

metric{k,1} = (4\*pi\*CA(i,1).Area)/(CP(i,1).Perimeter \* CP(i,1).Perimeter);

k=k+1;

end

end

normal=0; abnormal=0;

for i=1:validCells

if metric{i,1}>=0.75 %all cells with a metric value above 0.75 are considered normal, as they prove to be more circular

normal=normal+1;

else

abnormal=abnormal+1;

end

end

threshold=0.1\*validCells; % 10 percent of the number of valid cells (Experimental)

% Reference - http://goo.gl/E4NCam

%printing the report

disp(sprintf('\n\nReport :-'));

disp(sprintf('Patient name : %s', P\_name));

disp(sprintf('Patient age : %s', P\_age));

disp(sprintf('Doctor / Medical practioner name : %s', D\_name));

disp(sprintf('Image Path : %s', img\_path));

disp(sprintf('\nTotal number of objects detected\t= %d',CC.NumObjects));

disp(sprintf('Number of valid cells\t\t\t\t= %d',validCells));

disp(sprintf('Number of normal cells\t\t\t\t= %d',normal));

disp(sprintf('Number of abnormal cells\t\t\t= %d',abnormal));

disp(sprintf('Threshold \t\t\t\t\t\t\t= %f',threshold));

if abnormal>threshold

disp(sprintf('\nResult : \t\tYou are diagnosed with Sickle Cell Anaemia.\n'));

msgbox(sprintf(' Patient name : %s\n Patient age : %s\n Doctor / Medical practioner name : %s\n Image Path : %s\n\n Total number of objects detected\t= %d\n Number of valid cells\t\t\t\t= %d\n Number of normal cells\t\t\t\t= %d\n Number of abnormal cells\t\t\t= %d\n Threshold \t\t\t\t\t\t\t= %f\n\n Result : \t\tYou are diagnosed with Sickle Cell Anaemia.\n', P\_name, P\_age, D\_name, img\_path, CC.NumObjects, validCells, normal, abnormal, threshold),'Report- Positive');

else

disp(sprintf('\nResult : \t\tÝou are safe.\n'));

msgbox(sprintf(' Patient name : %s\n Patient age : %s\n Doctor / Medical practioner name : %s\n Image Path : %s\n\n Total number of objects detected\t= %d\n Number of valid cells\t\t\t\t= %d\n Number of normal cells\t\t\t\t= %d\n Number of abnormal cells\t\t\t= %d\n Threshold \t\t\t\t\t\t\t= %f\n\n Result : \t\tYou are safe.\n', P\_name, P\_age, D\_name, img\_path, CC.NumObjects, validCells, normal, abnormal, threshold),'Report- Negative');

end

end %function Code end