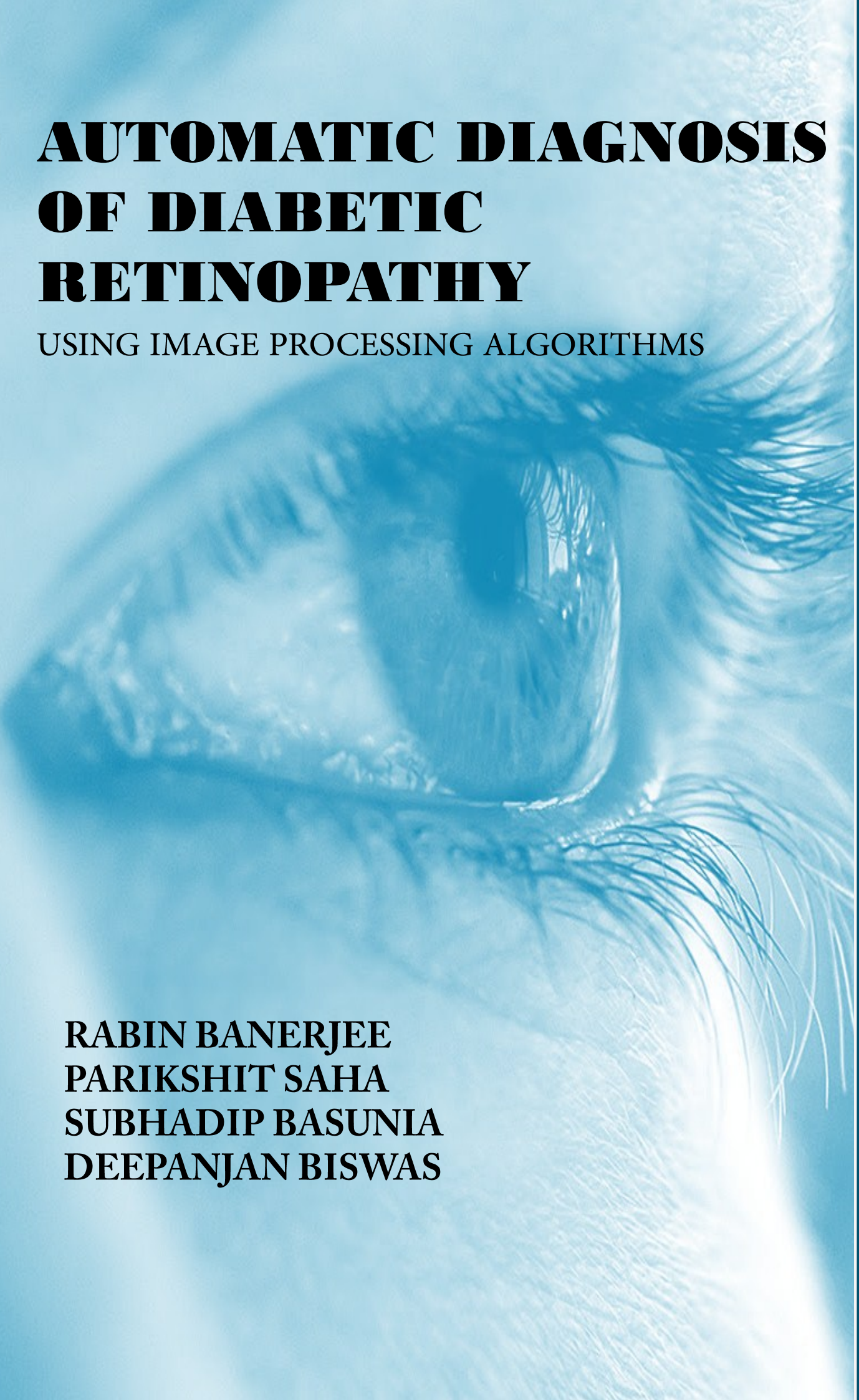
2012

AUTOMATIC DIAGNOSIS OF DIABETIC RETINOPATHY

USING IMAGE PROCESSING ALGORITHMS



RABIN BANERJEE
PARIKSHIT SAHA
SUBHADIP BASUNIA
DEEPANJAN BISWAS

PROJECT REPORT ON

AUTOMATIC DIAGNOSIS OF DIABETIC RETINOPATHY

USING IMAGE PROCESSING ALGORITHMS

UNDER THE GUIDANCE OF

Prof. H.S. DUTTA

4TH YEAR (8TH SEMESTER)

GROUP MEMBER:	ROLL NO:
RABIN BANERJEE	08102003049
PARIKSHIT SAHA	08102003015
SUBHADIP BASUNIA	08102003021
DEEPANJAN BISWAS	08102003020

ELECTRONICS & COMMUNICATION ENGINEERING DEPARTMENT

KALYANI GOVERNMENT ENGINEERING COLLEGE

KALYANI, NADIA, W.B., PIN –741235

কল্যানী- ৭৪১ ২৩৫
নদীয়া, পশ্চিমবঙ্গ

Kalyani– 741 235
Nadia, West Bengal

তাং/Date-

কল্যানী গভঃ ইঞ্জিনিয়ারিং কলেজ

Kalyani Government Engineering College

(Govt. Of West Bengal)

This is certify that this is the report of the B.Tech final year project on "AUTOMATIC DIAGNOSIS OF DIABETIC RETINOPATHY" performed by RABIN BANERJEE, PARIKSHIT SAHA, SUBHADIP BASUNIA, DEEPANJAN BISWAS under my supervision and guidance.

The report in present form is submitted for the fulfillment of all the requirements as specified by KALYANI GOVERNMENT ENGINEERING COLLEGE, KALYANI, NADIA, WB-741235 and as per regulation of the WEST BENGAL UNIVERSITY OF UNIVERSITY, KOLKATA, WEST BENGAL. The results in the report are original in nature to the best of my knowledge and belief and worth of incorporation in B.Tech curriculum of the Electronics & Communication Engineering Department.

……

Prof. H.S. DUTTA

ASSISTANT PROFESSOR
ELECTRONICS & COMMUNICATION ENGINEERING DEPT.
KALYANI GOVERNMENT ENGINEERING COLLEGE
KALYANI, NADIA,WB-741235

……

Dr. ACHINTYA DAS

HOD.
ELECTRONICS & COMMUNICATION ENGINEERING DEPT.
KALYANI GOVERNMENT ENGINEERING COLLEGE
KALYANI, NADIA,WB-741235

……

EXAMINERS SIGN:

ABSTRACT

This project applies the process and knowledge of digital signal processing and image processing to diagnose diabetic retinopathy from images of retina using DSP Processor. The Processing stage equalizes the uneven illumination associated with fundus images and also removes noise present in the image. Segmentation stage clusters the image into two distinct classes while the Disease Classifier stage was used to distinguish between candidate lesions and other information. Method of diagnosis of red spots, bleeding and detection of vein-artery crossover points were also developed in this work using the color information, shape, size, object length to breadth ration as contained in the digital fundus image in the detection of this disease.in our project involves this implementation using DSP Processor.

ACKNOWLEDGEMENT

The success of any project depends largely on the encouragement and guidelines of many others, apart from the group members. We take this opportunity to express our gratitude to the people who have been involved in the successful completion of this project.

We would like to show our greatest appreciation to Prof. H.S.Dutta. We can't say thank you enough for his tremendous support and help. Without his encouragement and guidance this project would not have materialized.

The guidance and support received from all the members who contributed and who are contributing to this project, was vital for the success of the project. We are grateful for their constant support and help.

Contents

MOTIVATION

Diabetic retinopathy (DR) can be defined as damage to microvascular system in the retina due to prolonged hyperglycemia. However, due to the large number of diabetic subjects, DR is likely to pose a public health burden in India. A huge increase of DR in people at risk for the worst complications of chronic illness with too few specialists to provide regular screening and identify people in imminent danger of permanent vision loss.

Despite physician recommendations that people with diabetes obtain annual dilated retinal examinations to detect sight-threatening lesions, only approximately half of all known diabetics currently receive this standard of care. For many in underserved areas of the country, and for many more who do not recognize the risks or who do not have the financial ability to pay for conventional care delivery, diabetic retinopathy threatens to deprive them of their vision and a productive and fulfilling life.

Digital detection of diabetic retinopathy is a technological means of bridging the gap between recommendations and actual access to healthcare. It will be able to reach patients who would not otherwise obtain dilated retinal examinations.

Automatic detection of diabetic retinopathy using digital imaging and telemedicine is a useful addition to conventional physician screening. It overcomes time, money, and logistical constraints, improves access, reduces disparities, provides equity of care, and improves outcomes, and may be beneficial even in the case of current unprecedented increases in newly diagnosed cases of diabetes.

So we are trying to develop a system which can take a picture of eye of a patient and automatically diagnosis whether a patient have DR or not. This may be beneficial for the current situation of India.

INTRODUCTION

All patients with diabetes mellitus will have a higher risk of being diagnosed with diabetic retinopathy. In Singapore & India like third world countries, 22% of patients are diagnosed with diabetic retinopathy and half of this affected patients had severe sight-threatening diseases.

Diabetic retinopathy happens when the tiny blood vessels are damaged. These blood vessels are responsible for providing nutrients and oxygen to the retina. As shown in Figure 1, the person with diabetic retino -pathy does not have a clear vision of the surround - ings. Thus, he will face numerous problems and difficulties in his daily life routine. When his conditions become more severe, he will gradually lose his vision. Hence, early diagnosis or treatment will be able to prevent blindness complicated by diabetic retinopathy. In order to prevent loss of vision, periodic eye-examination is necessary to check for any early warning signs. If diabetic retinopathy is detected at an early stage, laser treatment is usually performed to delay the progression of the disease.

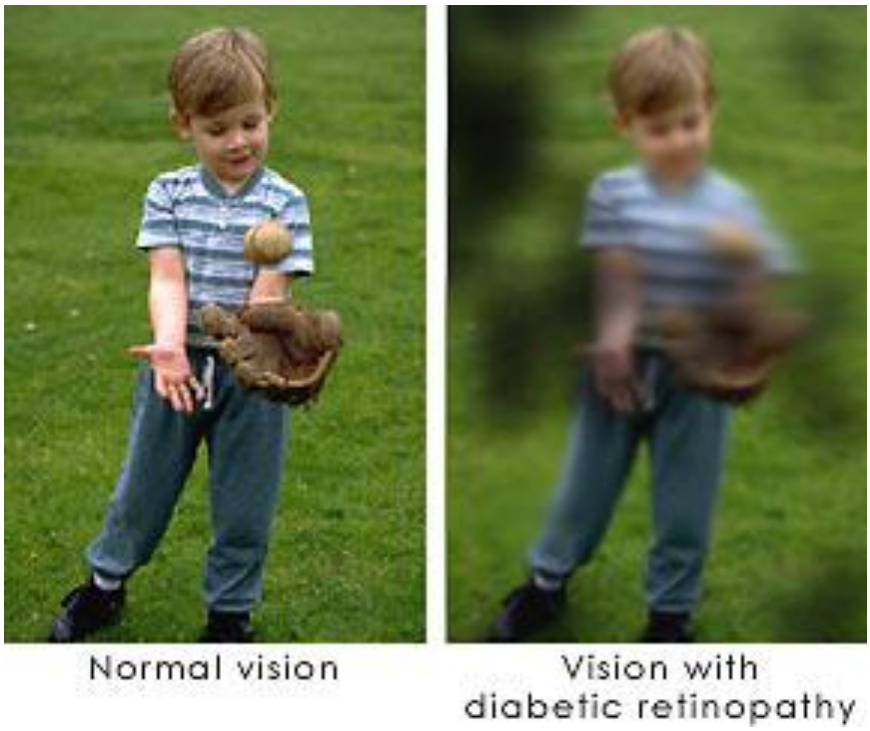


Figure 1: Normal vision and vision with diabetic retinopathy

EYE ANATOMY

The anatomy of the human eye is shown in side figure. When light enters the pupil, the lens will focus it and then detected by the retina. The amount of light entering the eye will be regulated by the iris. The movement of the iris is controlled by the contraction and relaxation of the ciliary muscle. The retina contains light-sensitive receptors consisting of rods and cones. As rods are more sensitive and require less light to function, they are used mostly in the night or dim places with poor visibility. Unlike the characteristic of rods, cones are less sensitive and are normally used in daylight for perception of colour. Upon detected by the retina, electrical signal will be delivered to the brain by the optic nerve. Finally, these impulses will be translated into images that we perceived. The macula is located in the centre of the retina and this is responsible for providing a central vision. Also, the fovea is located within the macula and this location provides the sharpest vision and colour perception.

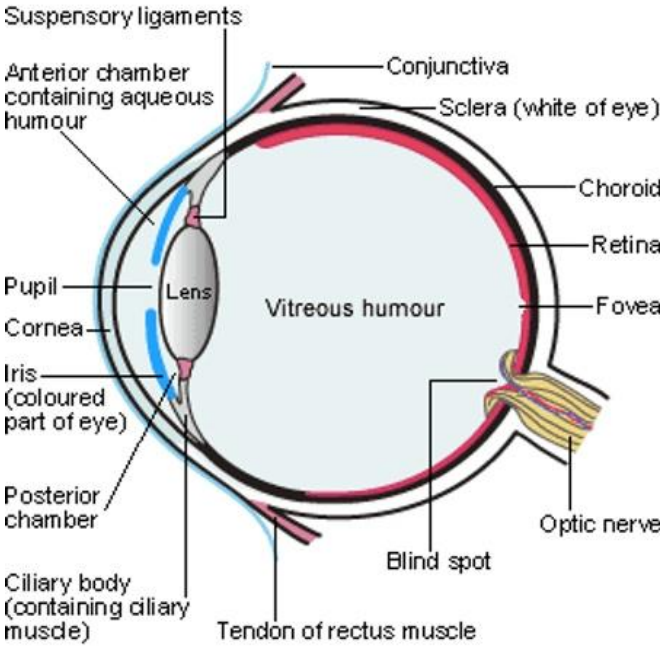


Figure 2:Human Eye Structure

DIABETIC RETINOPATHY

The effect of diabetes on the eye is called Diabetic Retinopathy (DR). It is known to damage the small blood vessel of the retina and this might lead to loss of vision. The disease is classified into three stages viz:

1. *Non-Proliferative or Background Diabetic Retinopathy*
2. *Proliferate Diabetic Retinopathy*

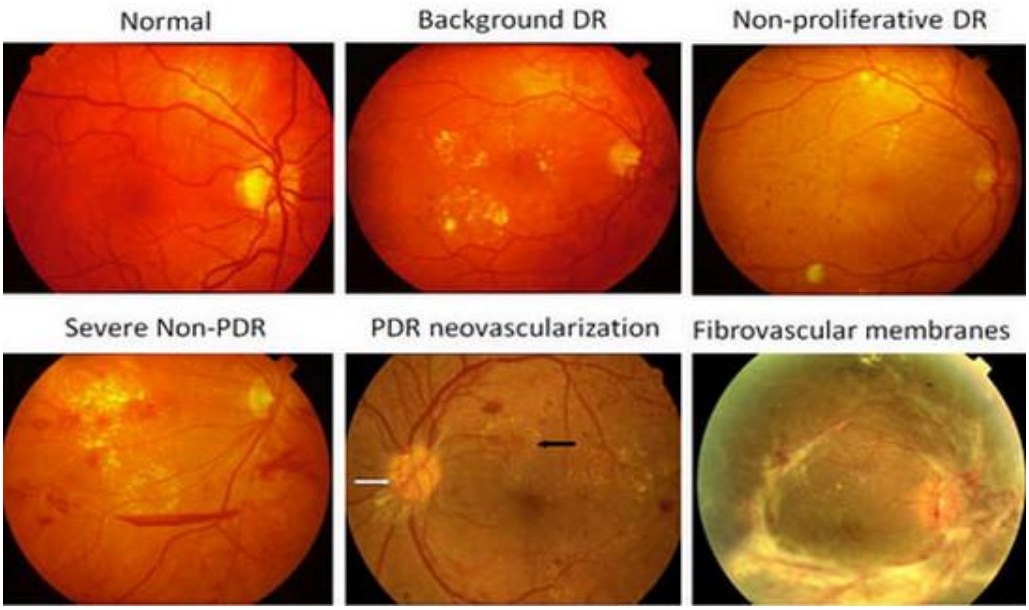


Figure 3: Stages of Diabetic Retinopathy

Nonproliferative diabetic retinopathy occurs when there are only intraretinal microvascular changes, such as altered retinal vascular permeability and eventual retinal vessel closure. Clinically, the hallmark of the nonproliferative phase is microaneurysms and intraretinal abnormalities. Neovascularization is not a component of the nonproliferative phase. However, in advanced NPDR, nonperfusion of the retina may develop and lead to the proliferative phase. Proliferative diabetic retinopathy is characterized by new vessels and sometimes fibrous bands proliferating on the retinal surface. In both nonproliferative and proliferative

diabetic retinopathy, macular edema can occur as increased retinal vascular permeability leads to accumulation of fluid in the retinal area serving central vision.

Proliferative diabetic retinopathy continues to be a major cause of blindness throughout the world. The natural history demonstrates that its development is primarily related to progressive retinal ischemia from diabetic retinopathy. The primary complications leading to vision loss, tractional retinal detachment and vitreous hemorrhage, are dependent upon the relationship between the neovascular tissue and the vitreous.

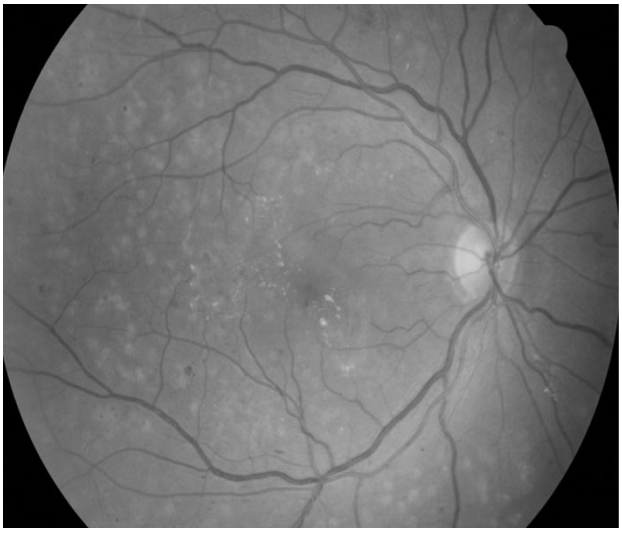


Figure 4:a. Non-Proliferative DR

Figure 4:b.Proliferative DR

ABNORMALITIES ASSOCIATED WITH THE EYE

Abnormalities associated with the eye due to Diabetic Retinopathy can be divided into five different forms. If not properly treated DR might eventually lead to loss of vision.

Ophthalmologists have come to agree that early detection and treatment is the best treatment for this disease. Diabetic Retinopathy can occur in any of the form described below as related to this research work.

Microaneurysms: These are the first clinical abnormality to be noticed in the eye. They may appear in isolation or in clusters as tiny, dark red spots or looking like tiny haemorrhages within the light sensitive retina. Their sizes ranges from 10-100 microns i.e. less than 1/12th the diameter of an average optics disc and are circular in shape, at this stage, the disease is not eye threatening.

Haemorrhages: Occurs in the deeper layers of the retina and are often called 'blot' haemorrhages because of their round shape.

Hard Exudates: These are one of the main characteristics of diabetic retinopathy and can vary in size from tiny specks to large patches with clear edges. As well as blood, fluid that is rich in fat and protein is contained in the eye and this is what leaks out to form the exudates. These can impair vision by preventing light from reaching the retina.

Soft Exudates: These are often called 'cotton wool spots' and are more often seen in advanced retinopathy.

Neovascularization: This can be describe as abnormal growth of blood vessels in areas of the eye including the retina and is associated with vision loss. This occurs in response to ischemia, or diminished blood flow to ocular tissues. If these abnormal blood vessels grow around the pupil, glaucoma can result from the increasing pressure within the eye.

These new blood vessels have weaker walls and may break and bleed, or cause scar tissue to grow that can pull the retina away from the back of the eye. When the retina is pulled away it is called a retinal detachment and if left untreated, a retinal detachment can cause severe vision loss, including blindness. Leaking blood can cloud the vitreous (the clear, jelly-like substance that fills the eye) and block the light passing through the pupil to the retina, causing blurred and distorted images. In more advanced proliferate retinopathy; diabetic fibrous or scar tissue can form on the retina.

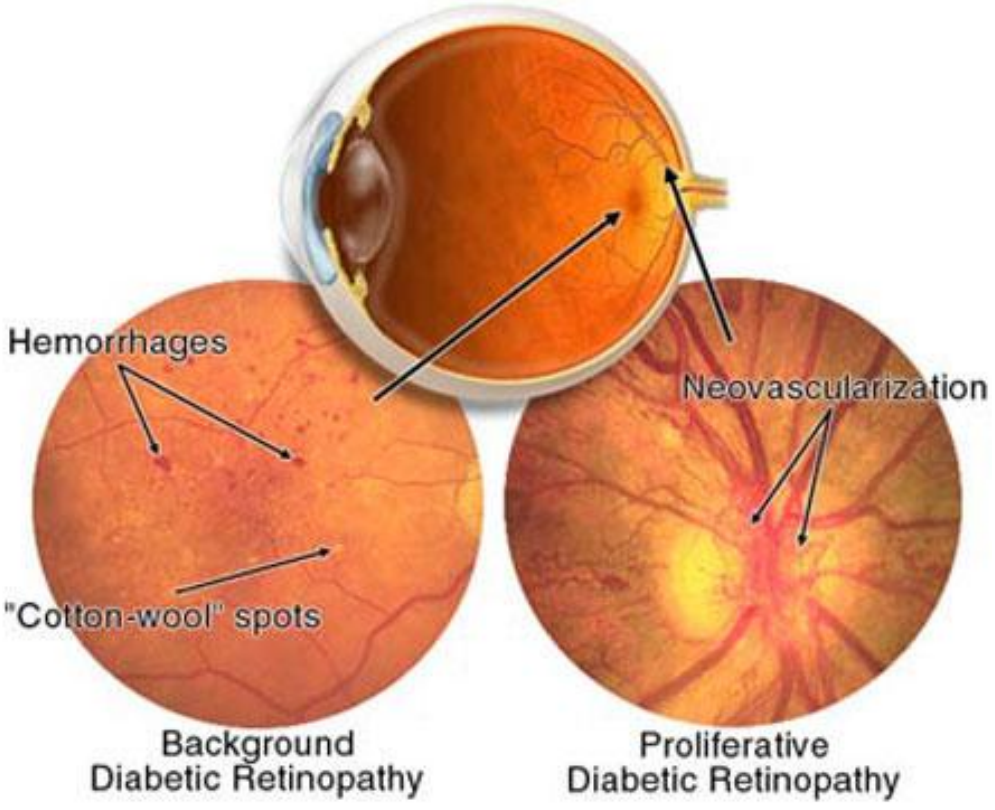


Figure 5:Abnormalities due to Diabatic retinopathy

REVIEW WORK AND PRE-PROCESSING STEPS

IMAGE PROCESSING: We can think of an image as a function, f, from R2 to R:

f(x, y) gives the intensity at position (x, y)
Realistically, we expect the image only to be defined over a rectangle, with a finite range:
f: [a,b]x[c,d] à [0,1]
A color image is just three functions pasted together. We can write this as a vector-valued function:

COLOR SPACE CONVERSION: A common strategy is to match the luminance of the grayscale image to the luminance of the color image. To convert any color to a grayscale representation of its luminance, first one must obtain the values of its red, green, and blue (RGB) primaries in linear intensity encoding, by gamma expansion. Then, 30% of the red value, 59% of the green value, and 11% of the blue value is added together. Colours in an image may be converted to a shade of gray by calculating the effective brightness or luminance of the colour and using this value to create a shade of gray that matches the desired brightness. The effective luminance of a pixel is calculated using the formula,

$$Y = 0.21\ R + 0.71\ G + 0.07\ B.$$

This luminance value can then be turned into a grayscale pixel. A grayscale digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information.

ZERO-PADDING: The result of this color space conversion section is fed to the edge padding section of PPS. In this subsection, the image is padded with zeros so as to remove unwanted noise that may be introduced during the intensity enhancement and Segmentation stage and also to be able to calculate the minimum and maximum intensity value of the whole image.

There are four steps associated with this section, viz. image intensity thresholding, image fillings, and minimum and maximum intensity detection.

HISTOGRAM EQUALIZATION: Histogram equalization is nothing but a finding of cumulative distribution function for a given probability density function. Modeling of the histogram is usually done by the use of continuous process functions rather than discrete process functions. Suppose for a given image the intensity levels are continuous quantities and is normalized to the range [0 1]. According to Gonzalez and Woods [2002], transformation can be performed on the probability density function of the intensity levels input image Pr(r) is to obtain S as shown below where ω is the dummy variable of integration.

After the transformation, the image will have an increased dynamic range, high contrast and the probability density function of the output will be uniform, which can be regarded as a Cumulative Distribution Function (CDF).
Ps (s) = 1 for $0 \leq s \leq 1$, else zero.

In digital images, the intensity levels are discrete in nature, so the method above is often

$$S_k = T(r_k) = \sum_{j=1}^{k} P_r(r_j) = \sum_{j=1}^{k} \frac{n_j}{n}$$

referred to as histogram equalization method, though the output image histogram is not uniform due to the discrete nature of the variables. For discrete value data, the summations and equalization methods above become for j = 1,2,……..L, where s_k is the intensity value in the output (processed) image corresponding to the value r_k in the input image. Example of this is as shown below

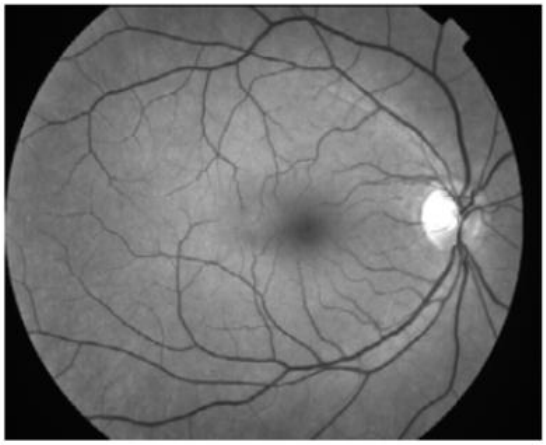


Figure 6:a. Input Image

Figure 6:b.Output Image

ADAPTIVE HISTOGRAM EQUILIZATION

The main objective of this method is to define a point transformation within a local fairly large window with the assumption that the intensity value within it is a stoical representation of local distribution of intensity value of the whole image. The local Window is assumed to be unaffected by the gradual variation of intensity between the image centres and edges. The point transformation distribution is localized around the mean intensity of the window and it covers the entire intensity range of the image. Consider a running sub image W of N X N pixels centered on a pixel P (i,j) , the image is filtered to produce another sub image P of (N X N) pixels according to the equation below

$$p_n = 255.\left(\frac{[\phi_w(p) - \phi_w(Min)]}{[\phi_w(Max) - \phi_w(Min)]} \right)$$

Where,

$$\phi_w(p) = \left[1 + \exp\left(\frac{\mu_w - p}{\sigma_w} \right) \right]^{-1}$$

and Max and Min are the maximum and minimum intensity values in the whole image, while μw and σw indicate the local window mean and standard deviation which are defined as:

$$\mu_w = \frac{1}{N^2} \sum_{(i,j)\in(k,l)} p(i,j)$$

$$\sigma_w = \sqrt{\frac{1}{N^2} \sum_{(i,j)\in(k,l)} (p(i,j) - \mu_w)^2}$$

MORPHOLOGICAL FILTERING

Morphological operations are a set of image processing operations that analyzes the shapes within the image. It applies a structuring element to the image and output the image of the same size. The output value of each pixel is determined by the neighboring pixels with its corresponding pixel of input image. The size and shape of the structuring element affects the number of pixels being added or removed from the object in the image.

The most basic morphological operations used are dilation and erosion.

Erosion: Erosion removes pixels on the object boundaries in the image by changing it to the background pixel. This shrinks the object and breaks up a single object.

Dilation: Dilation, on the other hand, adds pixels to the object boundaries by changing the background pixel surrounding it. This enlarges the object and multiple objects could merge together as one.

Opening or closing is a single function with the combination of dilation and erosion.

Opening: In opening, the image would undergo erosion followed by dilation. This removes the small object pixels before enlarging the remaining

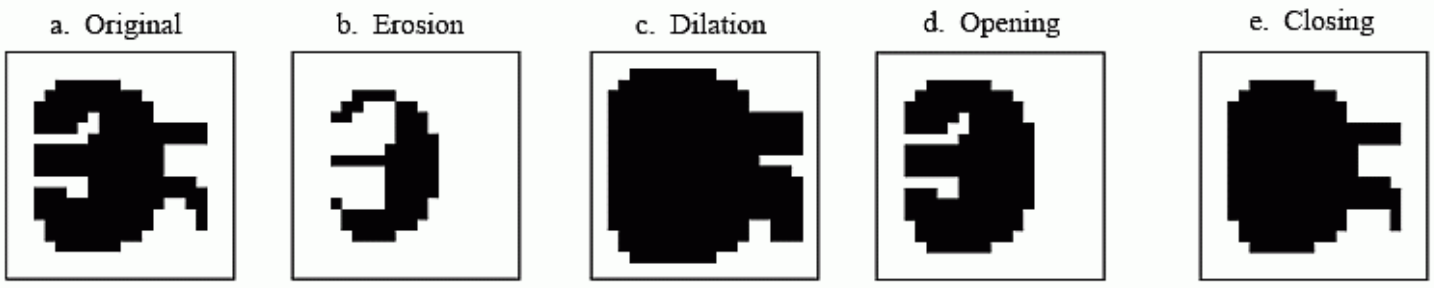


Figure 7:Example of Morphological operations

Closing: In Closing, the image would undergo dilation followed by erosion. This removes the small background pixels before enlarging the remaining. In this way, the contours of the object smoothen and small object gaps fused. These functions help to handle noise in the image or adjust it to "enclose" a certain desired object.

Segmentation: Image segmentation is used to locate the objects or boundaries in the image. In edge detection function, the contours of the objects are extracted from the image. Canny method is used for this project as it is better compared to the other similar Matlab functions by having two different thresholds to detect the edges.

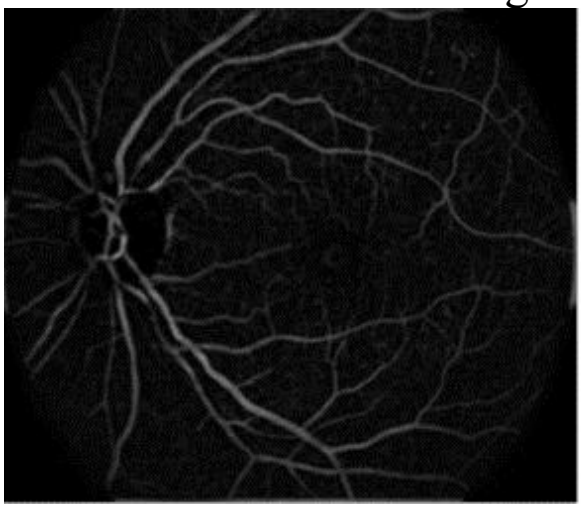


Figure 8:a.Input image

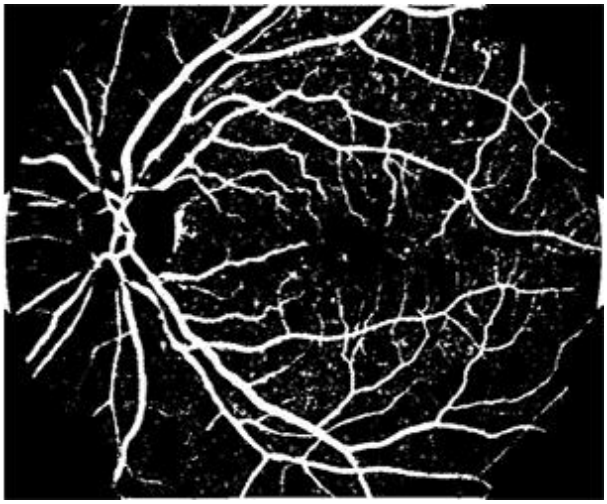


Figure 8:b.Output Image after Segmentation

BLOOD VESSEL DETECTION

Blood vessels are extracted in this project for the identification of diabetic retinopathy. The contrast of the fundus image tends to be bright in the centre and diminish at the side, hence preprocessing is essential to minimize this effect and have a more uniform image. After which, the green channel of the image is applied with morphological image processing to remove the optical disk. Image segmentation is then performed to adjust the contrast intensity and small pixels considered to be noise are removed. Another green channel image is processed with image segmentation and combined with the mask layer. These two images are compared and the differences are removed. The obtained image would represent the blood vessels of the original image.

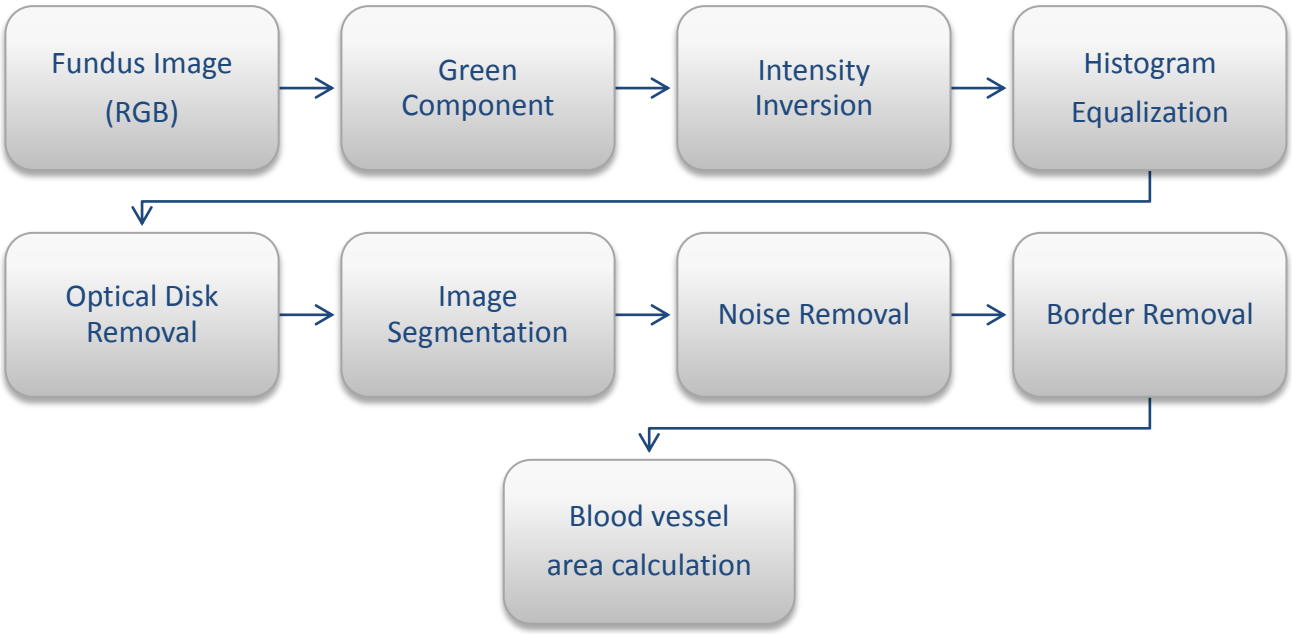


Figure 9: Block Diagram of Blood Vessel Detection

Step 1: Fundus Image acquisition:

Originally retinal images are obtained using fundus camera by opthalmologists by Fundus Camera.We have used DRIVE Database for Fundus Image testing purpose.

Step 2: Intensity Inversion of Green Channel:

The intensity of the green channel is then inversed before adaptive histogram equalization is applied. Grayscale image instead of the green channel is used as it is more efficient in the detection in later stages.

Step 3: Histogram Equalization:

The contrast of the fundus image tends to be bright in the centre and diminish at the side, hence preprocessing is essential to minimize this effect and have a more uniform image.Hence, the image's contrast is stretched by applying adaptive histogram equalization.

Step 4:Optical Disk Removal:

The optical disk is a black patch in the image as shown at Figure 11.d. Morphological opening which consisted of erode followed by dilate is applied. Erode function protects the small blood vessels by reducing their sizes while dilate function blows up the larger remaining details which are intended to be removed. The optical disk is then removed by subtracting Figure:10 with Figure: 11.d.

Figure 10: Image after Opening

Step 5: Image segmentation:

We used binary thres-holding method for image segmentation step.For this, the image (Figure:11.e) is then converted to a binary image. The pixels of the input image are converted to binary 1 (white) for values greater than the selected threshold and to binary 0 (black) if otherwise.

Step 6: Noise Removal:

After segmentation, the binary image becomes noisy. So, we calculate all connected clusters & delete pixel-areas smaller than a particular value, considering them as noise.

Step 7: Border Removal:

While working with binary image, the circular border creates some noise pixels. Grayscale image instead of the green channel is used as it is more efficient in border detection. We used canny method to detect the edges before enclosing the circular region with a top and bottom bar. Image filling is done to fill the region. The circular border is obtained after subtracting the dilated image with the eroded image. The border is then subtracted from Figure 11:g to get final image Figure 11:h.

Step 8: Blood vessel area calculation:

Finally the area of the blood vessels is obtained by using two loops to count the number of pixels with binary 1 (white) in the final blood vessel image(Figure 11:h)

Stages of blood vessel detection

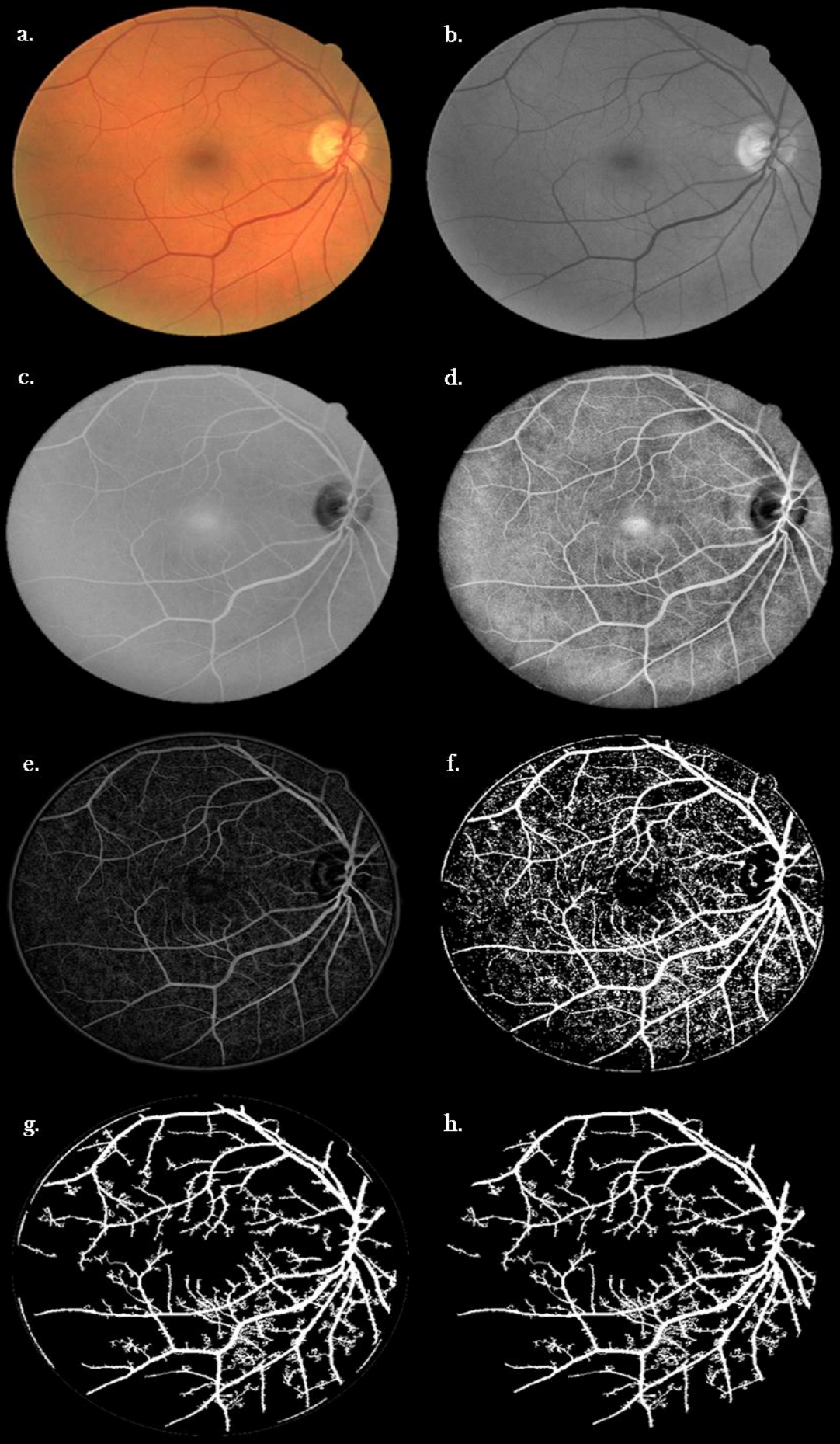


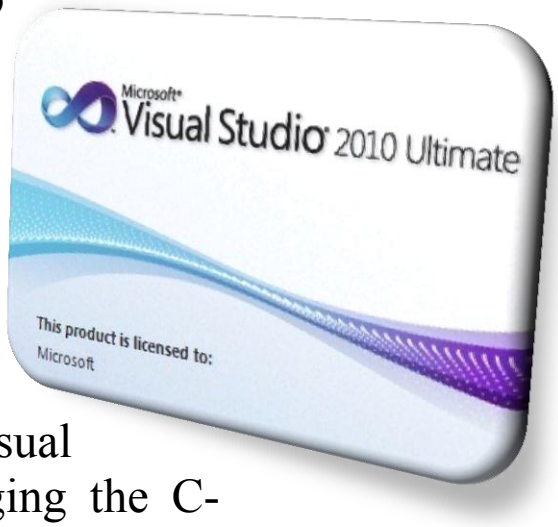
Figure 10: a.Original Image ; b. Green component ; c. Intensity Inversion ; d.Histogram Equalization ; e. Optical Disk Removal ; f. Image Segmentation ; g. Noise Removal ; h. Border Removal

SOFTWARE IMPLEMENTION

Software implementation of Blood vessel detection & area calculation in our project is done in three different phase.

Development Phase: Initially we implemented the basic pre-processing stages by matlab programming on MATLAB vR2012a.MATLAB is chosen for development phase software implementation as some of the basic building blocks are already available in Matlab Image Processing Toolbox.

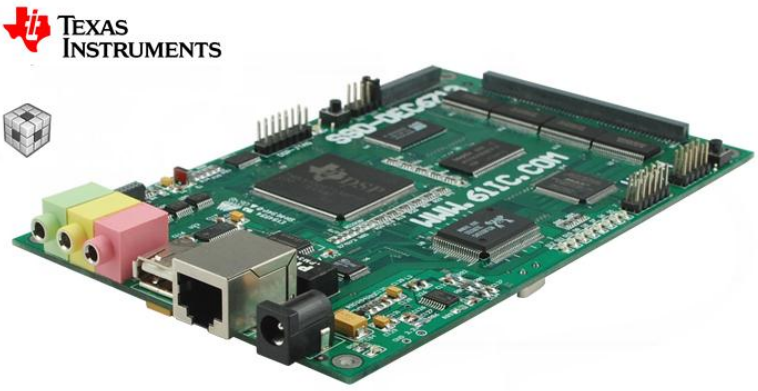
Testing Phase: Later we used to convert the MATLAB programms in C programming language as we can't be able to implement those matlab programs for embedded application.We wrote C-functions of all the MATLAB Library function needed for our project. At first we have used Dev-C++ & Microsoft Visual Studio 2010 for compiling & debugging the C-codes, for writing embedded-C version of those C codes.



Implementation Phase: Finally the embedded C code was executed in the DSP processor kit, Using Code Composer Studio as the interfacing software.This software is also used to emulate the DSK Processor in PC.

HARDWARE IMPLEMENTION

Digital Signal Processing is distinguished from other areas in computer science by the unique type of data it uses: signals. In most cases, these signals originate as sensory data from the real world: seismic vibrations, visual images, sound waves, etc.



DSP is the mathematics, the algorithms, and the techniques used to manipulate these signals after they have been converted into a digital form. This includes a wide variety of goals, such as:
enhancement of visual images, recognition and generation of speech, compression of data for storage and transmission, etc.

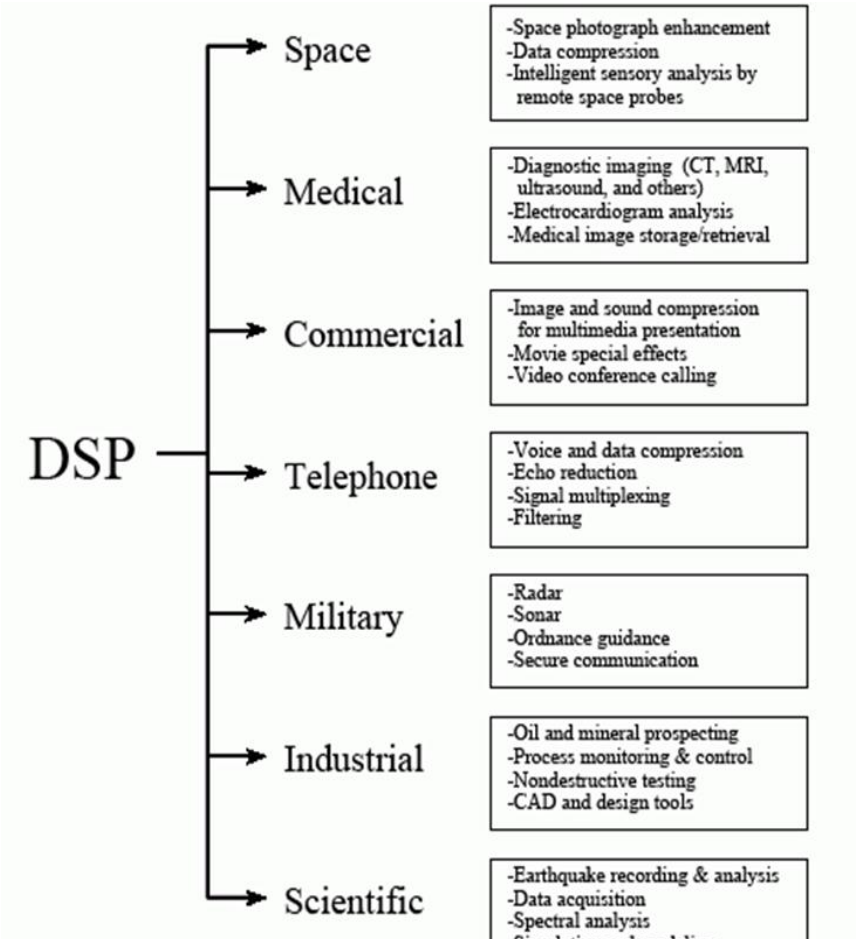


Figure 12: Different Application field of Digital Signal Processing

Suppose we attach an A/D converter to a computer and use it to acquire a chunk of real world data. These special characteristics have made image processing a distinct subgroup within DSP.

HARDWARE IMPLEMENTION-TMS320C6713 DSK

The Texas Instruments TMS320C6713 Digital Signal Processing Starter Kit is low cost development platforms for real-time digital signal processing applications. Each comprises a small circuit board containing a TMS320C6713 floating-point digital signal processor and a TLV320AIC23 analog interface circuit (codec) and connects to a host PC via a USB port. PC software in the form of Code Composer Studio (CCS) is provided in order to enable software written in C or assembly language to be compiled and/or assembled, linked, and downloaded to run on the DSK.

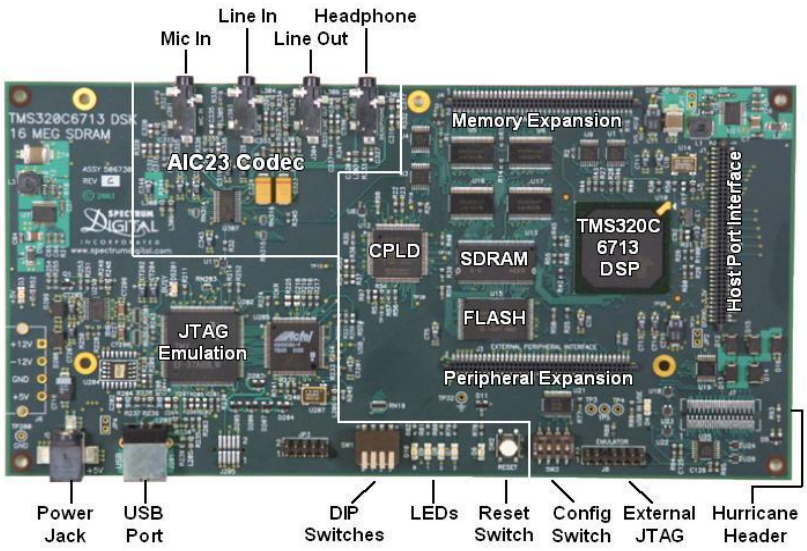


Figure 13: Architecture of TMS320C6713 DSP Kit

A digital signal processor (DSP) is a specialized form of microprocessor. The architecture and instruction set of a DSP are optimized for real-time digital signal processing. Typical optimizations include hardware multiply - accumulate (MAC) provision, hardware circular and bit-reversed addressing capabilities (for efficient implementation of data buffers and fast Fourier transform computation), and Harvard architecture (independent program and data memory systems). In many cases, DSPs resemble microcontrollers in so far as they provide single chip computer solutions incorporating onboard volatile and nonvolatile memory and a range of peripheral interfaces and have a small footprint, making them ideal for embedded applications.

In addition, DSPs tend to have low power consumption requirements. This attribute has been extremely important in establishing the use of DSPs in cellular handsets. As may be apparent from the foregoing, the distinctions between DSPs and other, more general purpose, microprocessors are blurred. No strict definition of a DSP exists. Semiconductor manufacturers bestow the name DSP on products exhibiting some, but not necessarily all, of the above characteristics as they see fit. The C6x notation is used to designate a member of the Texas Instruments (TI) TMS320C6000 family of digital signal processors. The architecture of the C6x digital signal processor is very well suited to numerically intensive calculations. Based on a very - long - instruction - word (VLIW) architecture, the C6x is considered to be TI's most powerful processor family. Digital signal processors are used for a wide range of applications, from communications and control to speech and image processing. They are found in cellular phones, fax/modems, disk drives, radios, printers, hearing aids, MP3 players, HDTV, digital cameras, and so on. Specialized (particularly in terms of their onboard peripherals) DSPs are used in electric motor drives and a range of associated automotive and industrial applications. Overall, DSPs are concerned with real-time signal processing. Real-time processing means that the processing must keep pace with some external event; whereas non real-time processing has no such timing constraint. The external event to keep pace with is usually the analog input. While analog-based systems with discrete electronic components including resistors and capacitors are sensitive to temperature changes, DSP - based systems are less affected by environmental conditions such as temperature. DSPs enjoy the major advantages of micro - processors. They are easy to use, flexible, and economical.

CODE COMPOSER STUDIO

Code Composer Studio (CCS) provides an integrated development environment IDE) for real-time digital signal processing applications based on the C programming language. It incorporates a C compiler, an assembler, and a linker. It has graphical capabilities and supports real-time debugging. The C compiler compiles a C source program with extension .c to produce an assembly source file with extension .asm . The assembler assembles an .asm source file to produce a machine language object file with extension .obj . The linker combines object files and object libraries as input to produce an executable file with extension .out . This executable file represents a linked common object file format (COFF), popular in Unix - based systems and adopted by several makers of digital signal processors. This executable file can be loaded and run directly on the digital signal processor. A Code Composer Studio project comprises all of the files (or links to all of the files) required in order to generate an executable file. A variety of options enabling files of different types to be added to or removed from a project are provided.

In addition, a Code Composer Studio project contains information about exactly how files are to be used in order to generate an executable file. Compiler/linker options can be specified. A number of debugging features are available, including setting breakpoints and watching variables, viewing memory, registers, and mixed C and assembly code, graphing results, and monitoring execution time. One can step through a program in different ways (step into, or over, or out). Real-time analysis can be performed using CCS's real-time data exchange (RTDX) facility. This allows for data exchange between the host PC and the target DSK as well as analysis in real-time without halting the target.

MATLAB Simulink Model

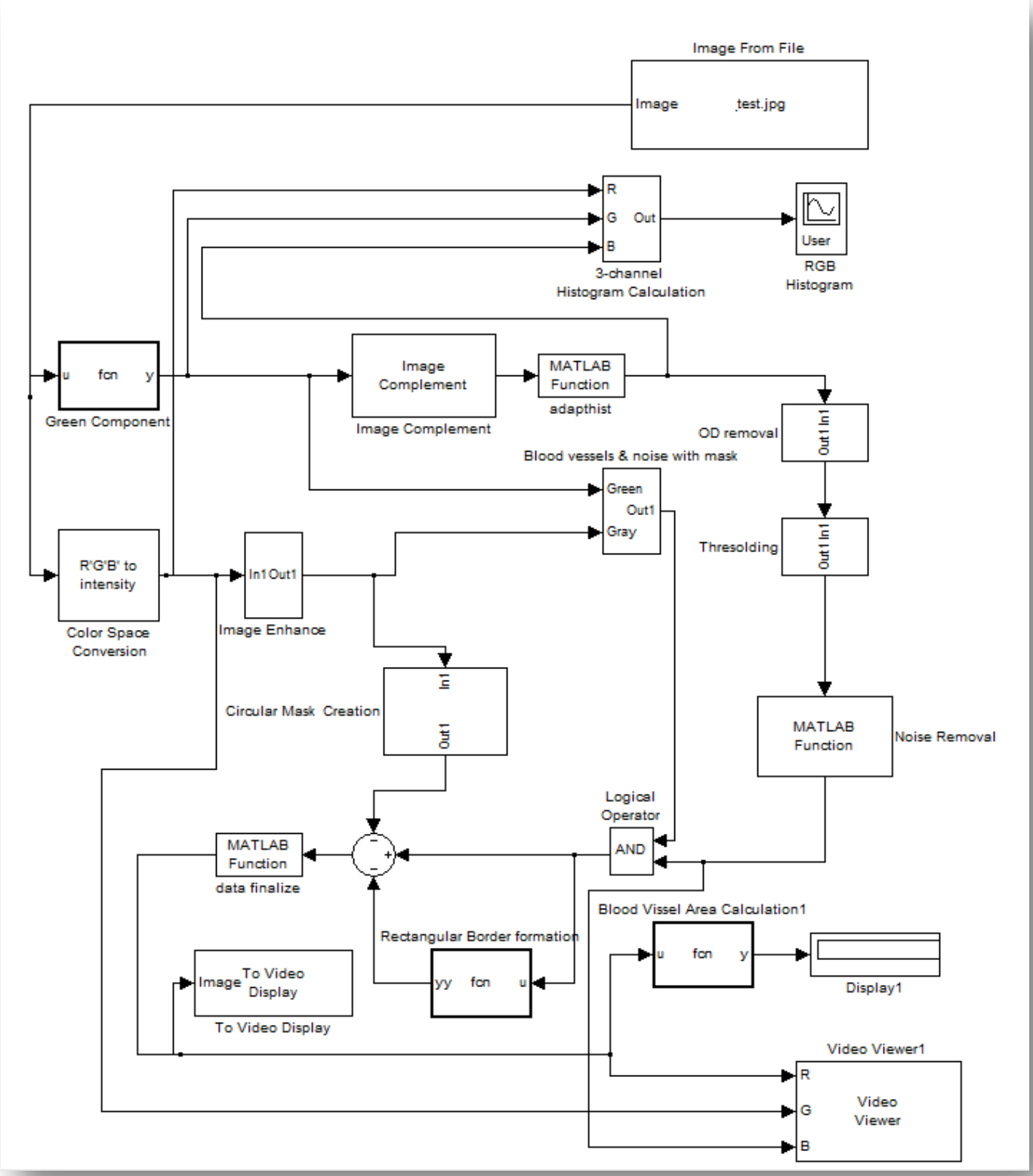


Figure 14:Simulink Model

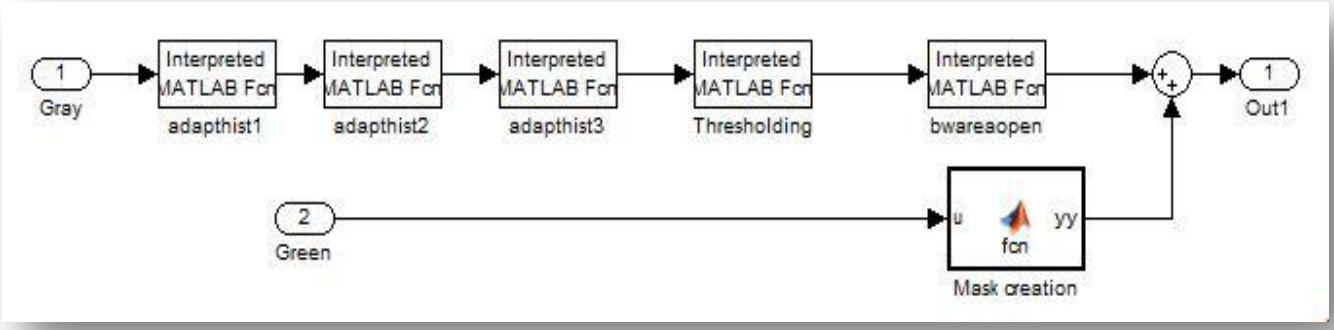


Figure 15: Detailed Blood vessels & noise with mask block

```
%function mask creation
function yy = fcn(u)
[w,h]=size(u);
max_GB_column=max(u);
max_GB_single=max(max_GB_column);
[row,col] =
find(u==max_GB_single);
med_row = floor(median(row));
med_col = floor(median(col));
r = 90;
[x,y]=meshgrid(1:h, 1:w);
yy = sqrt((x-med_col).^2+(y-med_row).^2)<= r;
```

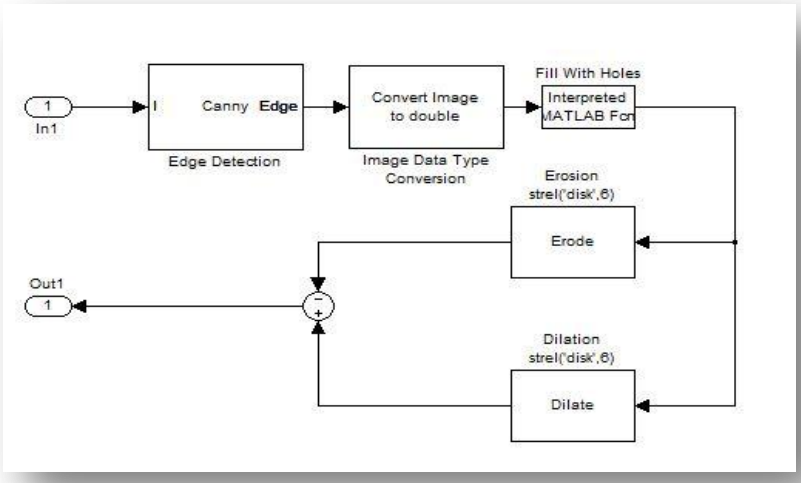


Figure 166:Detailed Circular Disk Creation Block

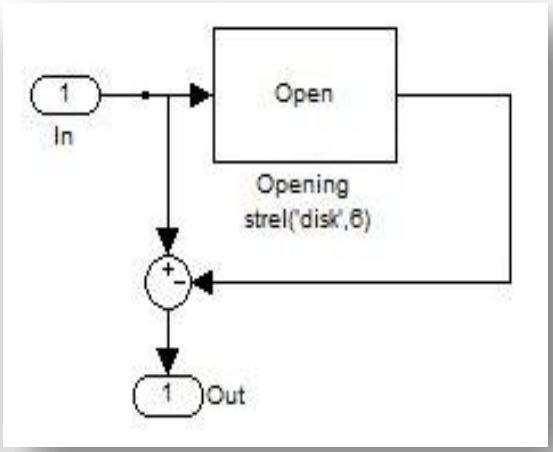


Figure 177:Detailed OD removal Block

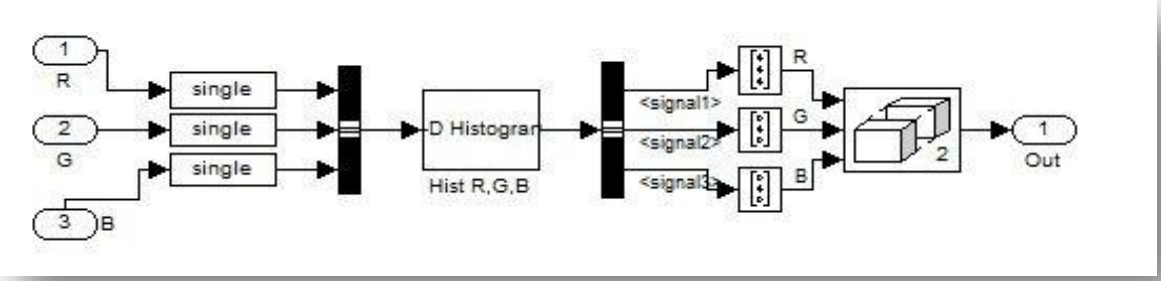


Figure 18: 3-channel histogram calculation

```
%Rectangular border formation
function yy = fcn(u)
[w,h]=size(u);
box_5pix=u*0;
%box_5pix=zeros(576,720);
for x=1:5 for y=1:h %for top bar
box_5pix(x,y)=1; %1->white
end
end
for x=w-5:w for y=1:h %for bottom bar
box_5pix(x,y)=1; %1->white
    end
end
for x=1:w for y=1:5 %for left bar
box_5pix(x,y)=1; %1->white
end
end
for x=1:w for y=h-5:h %for right bar
box_5pix(x,y)=1; %1->white
end
end
yy = box_5pix*255;
```

```
%blood vessel area calculation
function y = fcn(u)
[w,h]=size(u);
area_vessels = 0;
for x = 1:w for y = 1:h
if u(x,y) == 255
area_vessels= area_vessels+1;
end
end
end
y = area_vessels;
```

MATLAB RESULTS

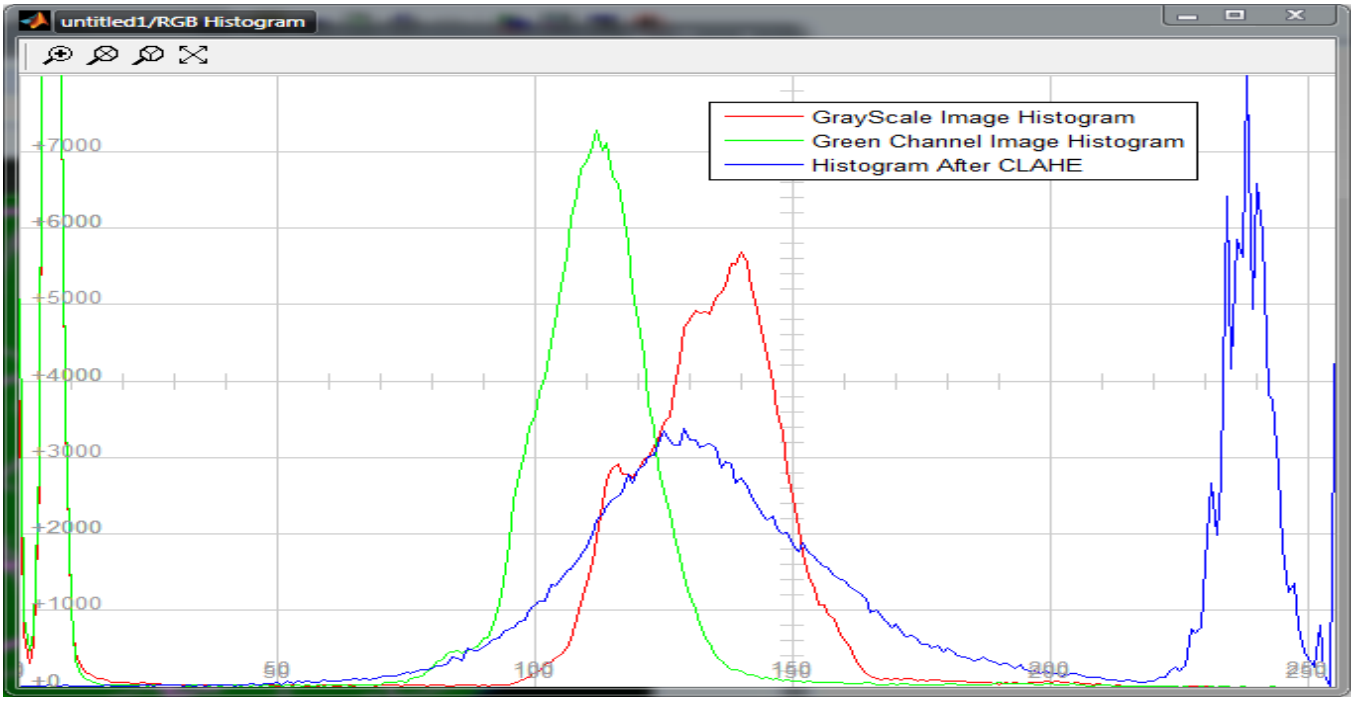


Figure 19:Histogram Analysis

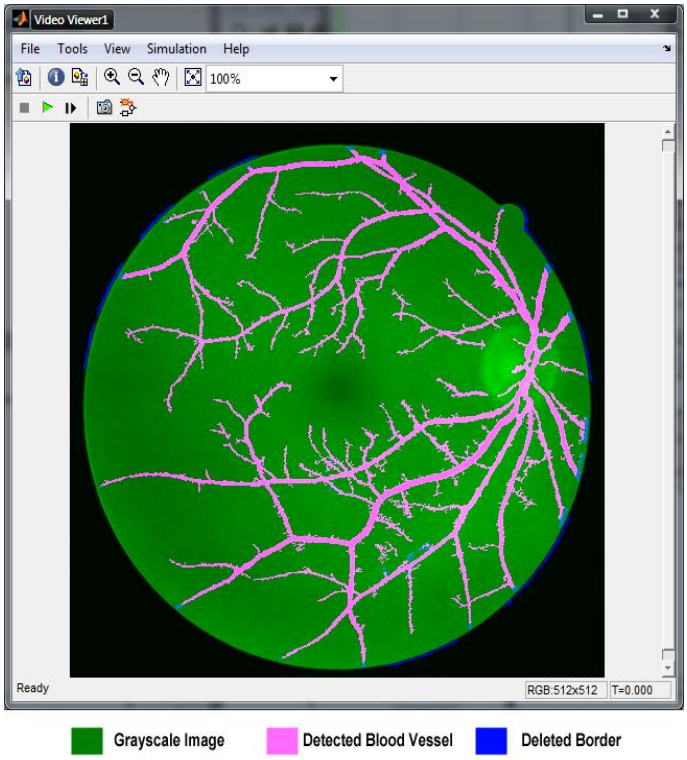


Figure 20: Detected Blood vessel image

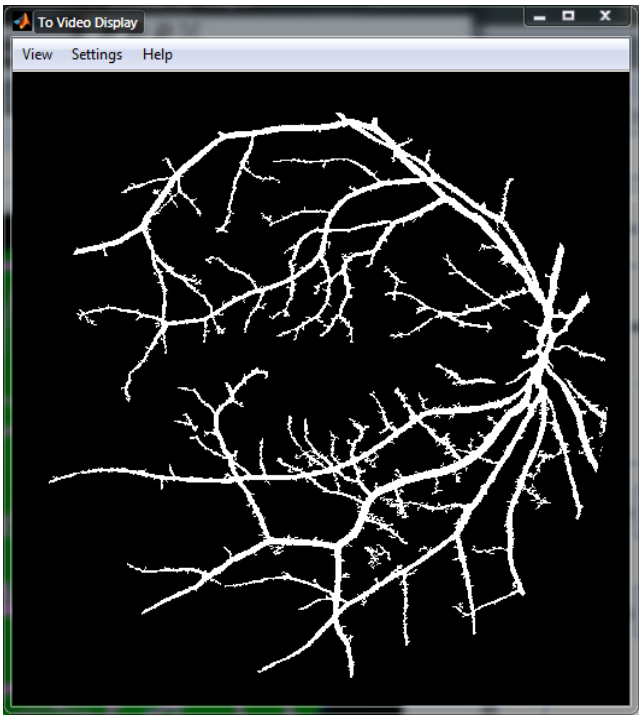


Figure 19: Input-Output Interlaced Image

C CODE

```
********************* image_operation.C  *********************

void im2bw(unsigned char *bwim,unsigned char ths,int X,int Y)
{
     int i;
     for(i=0;i<X*Y;i++)
     bwim[i]=bwim[i]>ths?255:0;
}
void imMinus(unsigned char *im1,unsigned char *im2,int row,int
col)
{
     short i,j;
     for(i=0;i<row;i++)
     for(j=0;j<col;j++)
     if(im1[i*col+j]>=im2[i*col+j])
     im1[i*col+j]=0;
     else
     im1[i*col+j]=(im2[i*col+j]-im1[i*col+j]);
}
void whitePixelCount(unsigned char *bwim,int X,int Y)
{
     int i,j;
     int NosOfWhitePixel, NosOfBlackPixel;
     unsigned long n=1;
     NosOfBlackPixel=0;
     NosOfWhitePixel=0;
for(i=0;i<X;i++)
     for(j=0;j<Y;j++)
     {
          if(bwim[i*Y+j]==255)
          NosOfWhitePixel++;
          else NosOfBlackPixel++;
     }
     printf("\nNos Of White Pixels are %d\n",NosOfWhitePixel);
     printf("Nos Of Black Pixels are %d \n",NosOfBlackPixel);
}
void imAnd(unsigned char *im1,unsigned char *im2,int row,int
col)
{
     short i,j;
     for(i=0;i<row;i++)
          for(j=0;j<col;j++)
          {
               im1[i*col+j]=im1[i*col+j] | im2[i*col+j];
          }
}
```

```
************************  Morph.H  ************************
#define NN 15
short mask[NN][NN] = {          0,0,0,0,1,1,1,1,1,1,1,0,0,0,0
                               ,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0
                               ,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0
                               ,0,1,1,1,1,1,1,1,1,1,1,1,1,1,0
                               ,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                               ,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                               ,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                               ,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                               ,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                               ,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                               ,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                               ,0,1,1,1,1,1,1,1,1,1,1,1,1,1,0
                               ,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0
                               ,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0
                               ,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0
                     };


void mask_erosion(unsigned char *,unsigned char *,int row,int
col);

void mask_dilation(unsigned char *,unsigned char *,int row,int
col);

void morph_mask_Opening(unsigned char *,unsigned char *,int
row,int col);

void morph_mask_Closing(unsigned char *,unsigned char *,int
row,int col);

************************  Morph.C  ************************

#include "morph.h"
#include "stdlib.h"
#include <memory.h>
void mask_dilation(unsigned char *the_image,unsigned char
*out_image,int row,int col)
{
int a, b, count, i, j, k,Y;
   short  max;
   Y=col;
     for(i=NN/2; i<row-NN/2; i++){
      for(j=NN/2; j<col-NN/2; j++){
         max = 0;
         for(a=-NN/2; a<=NN/2; a++){
             for(b=-NN/2; b<=NN/2; b++){
                if(mask[a+NN/2][b+NN/2] == 1){
                   if(the_image[(i+a)*Y+(j+b)] > max)
                      max = the_image[(i+a)*Y+(j+b)];
                }}}
```

```
         out_image[i*Y+j] = max;
      }}}

void mask_erosion(unsigned char *the_image,unsigned char
*out_image,int row,int col)
{
int a, b, count, i, j, k,Y;
   short   min;
   Y=col;
   for(i=NN/2; i<row-NN/2; i++){
      for(j=NN/2; j<col-NN/2; j++){
         min = 255;
         for(a=-NN/2; a<=NN/2; a++){
             for(b=-NN/2; b<=NN/2; b++){
                if(mask[a+NN/2][b+NN/2] == 1){
                   if(the_image[(i+a)*Y+(j+b)] < min)
                      min = the_image[(i+a)*Y+(j+b)];
                }}}
         out_image[i*Y+j] = min;
      }}}

void morph_mask_Opening(unsigned char *the_image,unsigned char
*out_image,int row,int col)
{
     short i,j;
     unsigned char *temp;//[row][col];
     temp=(unsigned char *)malloc(sizeof(char)*row*col);
     mask_erosion(the_image,out_image,row,col);
     memcpy (temp, out_image, row*col*sizeof(unsigned char));
     mask_dilation(temp,out_image,row,col);
     free(temp);
}


void morph_mask_Closing(unsigned char *the_image,unsigned char
*out_image,int row,int col)
{
     short i,j;
     unsigned char *temp;//[row][col];
     temp=(unsigned char *)malloc(sizeof(char)*row*col);
     mask_dilation(the_image,out_image,row,col);
     memcpy (temp, out_image, row*col*sizeof(unsigned char));
     mask_erosion(temp,out_image,row,col);
     free(temp);
}
```

```
********** bwareaopen.c **********


#define TH 95 //65
static int COUNT=0;
static int DELREG[TH][2]={{0},{0}};
static unsigned char Flag;
int X,Y;
int indexing(unsigned char *in_img,unsigned char *out_img,int
m,int n)
{
if(Flag==0)
return(0);
if(Flag==1){
if(out_img[m*Y+n]==0)
 {
 out_img[m*Y+n]=1;
 if(in_img[m*Y+n]==255)
   {

   //printf("%d  %d \n",m,n);
   indexing(in_img,out_img,(m-1),(n-1));
   indexing(in_img,out_img,(m-1),n);
   indexing(in_img,out_img,(m-1),(n+1));
   indexing(in_img,out_img,m,(n-1));
   indexing(in_img,out_img,m,(n+1));
   indexing(in_img,out_img,(m+1),(n-1));
   indexing(in_img,out_img,(m+1),n);
   indexing(in_img,out_img,(m+1),(n+1));
   if(COUNT<=TH){
   DELREG[COUNT][0]=m;
   DELREG[COUNT][1]=n;
}
   COUNT++;
     if(COUNT==TH){
     Flag=0;
     }
   }
  }
 }
 return(0);
 }


 void bwareaopen(unsigned char *in_img,unsigned char
*out_img,int row,int col)
 {

 int i,j,xx;
X=row;
Y=col;
```

```
for(i=1;i<X;i++)
{
for(j=1;j<Y;j++)
    {
    if(in_img[i*Y+j]==255 && out_img[i*Y+j]==0 )
    {
     Flag=1;
    indexing(in_img,out_img,i,j);
    //printf("end of cluster :::size=%d \n",COUNT);
//area delete if below threshold amount
       if(COUNT<TH)
       {
       for(xx=0;xx<COUNT;xx++)
       {
       in_img[(DELREG[xx][0]) *Y +(DELREG[xx][1])]=0;
       }
       }
    COUNT=0;
    }
   }
}}

*****************************************************************

**************************  CLAHE.C  ************************
/* ANSI C code from the article"Contrast Limited Adaptive
Histogram Equalization" by Karel Zuiderveld, karel@cv.ruu.nl
in "Graphics Gems IV", Academic Press, 1994

*Author: Karel Zuiderveld, Computer Vision Research Group,
Utrecht, The Netherlands (karel@cv.ruu.nl)*/



#include <stdlib.h>

const unsigned int uiMAX_REG_X = 16;
const unsigned int uiMAX_REG_Y = 16;



int CLAHE (unsigned char* pImage, unsigned int uiXRes,
unsigned int uiYRes,
      unsigned char Min, unsigned char Max, unsigned int
uiNrX, unsigned int uiNrY,
           unsigned int uiNrBins, float fCliplimit)
{
    unsigned int uiX, uiY;
    unsigned int uiXSize, uiYSize, uiSubX, uiSubY;
    unsigned int uiXL, uiXR, uiYU, uiYB;
```

```
    unsigned long ulClipLimit, ulNrPixels;
    unsigned char* pImPointer;
    unsigned char aLUT[uiNR_OF_GREY];
    unsigned long* pulHist, *pulMapArray;
    unsigned long* pulLU, *pulLB, *pulRU, *pulRB;

    if (uiNrX > uiMAX_REG_X) return -1;
    if (uiNrY > uiMAX_REG_Y) return -2;
    if (uiXRes % uiNrX) return -3;
    if (uiYRes & uiNrY) return -4;
  //  if (Max >= 256) return -5;
    if (Min >= Max) return -6;
    if (uiNrX < 2 || uiNrY < 2) return -7;
    if (fCliplimit == 1.0) return 9;
    if (uiNrBins == 0) uiNrBins = 128;

    pulMapArray=(unsigned long *)malloc(sizeof(unsigned
long)*uiNrX*uiNrY*uiNrBins);
    if (pulMapArray == 0) return -8;

    uiXSize = uiXRes/uiNrX; uiYSize = uiYRes/uiNrY;
    ulNrPixels = (unsigned long)uiXSize * (unsigned
long)uiYSize;

    if(fCliplimit > 0.0) {
       ulClipLimit = (unsigned long) (fCliplimit * (uiXSize *
uiYSize) / uiNrBins);
       ulClipLimit = (ulClipLimit < 1UL) ? 1UL : ulClipLimit;
    }
    else ulClipLimit = 1UL<<14;
    MakeLut(aLUT, Min, Max, uiNrBins);

    for (uiY = 0, pImPointer = pImage; uiY < uiNrY; uiY++) {
     for (uiX = 0; uiX < uiNrX; uiX++, pImPointer += uiXSize)
{
         pulHist = &pulMapArray[uiNrBins * (uiY * uiNrX +
uiX)];

MakeHistogram(pImPointer,uiXRes,uiXSize,uiYSize,pulHist,uiNrBi
ns,aLUT);
         ClipHistogram(pulHist, uiNrBins, ulClipLimit);
         MapHistogram(pulHist, Min, Max, uiNrBins,
ulNrPixels);
     }
     pImPointer += (uiYSize - 1) * uiXRes;
    }


    for (pImPointer = pImage, uiY = 0; uiY <= uiNrY; uiY++) {
     if (uiY == 0) {
         uiSubY = uiYSize >> 1;  uiYU = 0; uiYB = 0;
     }
```

```
     else {
         if (uiY == uiNrY) {
          uiSubY = uiYSize >> 1;   uiYU = uiNrY-1;      uiYB =
uiYU;
         }
         else {
          uiSubY = uiYSize; uiYU = uiY - 1; uiYB = uiYU + 1;
         }
     }
     for (uiX = 0; uiX <= uiNrX; uiX++) {
         if (uiX == 0) {
          uiSubX = uiXSize >> 1; uiXL = 0; uiXR = 0;
         }
         else {
          if (uiX == uiNrX) {
              uiSubX = uiXSize >> 1;  uiXL = uiNrX - 1; uiXR =
uiXL;
          }
          else {
              uiSubX = uiXSize; uiXL = uiX - 1; uiXR = uiXL +
1;
          }
         }

         pulLU = &pulMapArray[uiNrBins * (uiYU * uiNrX +
uiXL)];
         pulRU = &pulMapArray[uiNrBins * (uiYU * uiNrX +
uiXR)];
         pulLB = &pulMapArray[uiNrBins * (uiYB * uiNrX +
uiXL)];
         pulRB = &pulMapArray[uiNrBins * (uiYB * uiNrX +
uiXR)];

Interpolate(pImPointer,uiXRes,pulLU,pulRU,pulLB,pulRB,uiSubX,u
iSubY,aLUT);
         pImPointer += uiSubX;
     }
     pImPointer += (uiSubY - 1) * uiXRes;
    }
    free(pulMapArray);
    return 0;
}
void ClipHistogram (unsigned long* pulHistogram, unsigned int
               uiNrGreylevels, unsigned long ulClipLimit)
{
    unsigned long* pulBinPointer, *pulEndPointer, *pulHisto;
    unsigned long ulNrExcess, ulUpper, ulBinIncr, ulStepSize,
i;
    long lBinExcess;

    ulNrExcess = 0;  pulBinPointer = pulHistogram;
    for (i = 0; i < uiNrGreylevels; i++) {
```

```
     lBinExcess = (long) pulBinPointer[i] - (long)
ulClipLimit;
     if (lBinExcess > 0) ulNrExcess += lBinExcess;
    };


    ulBinIncr = ulNrExcess / uiNrGreylevels;
    ulUpper =  ulClipLimit - ulBinIncr;

    for (i = 0; i < uiNrGreylevels; i++) {
      if (pulHistogram[i] > ulClipLimit) pulHistogram[i] =
ulClipLimit;
      else {
       if (pulHistogram[i] > ulUpper) {
           ulNrExcess -= pulHistogram[i] - ulUpper;
pulHistogram[i]=ulClipLimit;
       }
       else {
           ulNrExcess -= ulBinIncr; pulHistogram[i] +=
ulBinIncr;
       }
       }
    }

    while (ulNrExcess) {
     pulEndPointer = &pulHistogram[uiNrGreylevels]; pulHisto =
pulHistogram;

     while (ulNrExcess && pulHisto < pulEndPointer) {
         ulStepSize = uiNrGreylevels / ulNrExcess;
         if (ulStepSize < 1) ulStepSize = 1;
         for (pulBinPointer=pulHisto; pulBinPointer <
pulEndPointer && ulNrExcess;
           pulBinPointer += ulStepSize) {
          if (*pulBinPointer < ulClipLimit) {
              (*pulBinPointer)++;   ulNrExcess--;
          }
         }
         pulHisto++;
     }
    }
}
void MakeHistogram (unsigned char* pImage, unsigned int
uiXRes,
          unsigned int uiSizeX, unsigned int uiSizeY,
          unsigned long* pulHistogram,
          unsigned int uiNrGreylevels, unsigned char*
pLookupTable)
{
    unsigned char* pImagePointer;
    unsigned int i;
```

```
    for (i = 0; i < uiNrGreylevels; i++) pulHistogram[i] = 0L;

    for (i = 0; i < uiSizeY; i++) {
     pImagePointer = &pImage[uiSizeX];
     while (pImage < pImagePointer)
pulHistogram[pLookupTable[*pImage++]]++;
     pImagePointer += uiXRes;
     pImage = &pImagePointer[-uiSizeX];
    }
}

void MapHistogram (unsigned long* pulHistogram, unsigned char
Min, unsigned char Max,
            unsigned int uiNrGreylevels, unsigned long
ulNrOfPixels)
{
    unsigned int i;  unsigned long ulSum = 0;
    const float fScale = ((float)(Max - Min)) / ulNrOfPixels;
    const unsigned long ulMin = (unsigned long) Min;

    for (i = 0; i < uiNrGreylevels; i++) {
     ulSum += pulHistogram[i]; pulHistogram[i]=(unsigned
long)(ulMin+ulSum*fScale);
     if (pulHistogram[i] > Max) pulHistogram[i] = Max;
    }
}

void MakeLut (unsigned char * pLUT, unsigned char Min,
unsigned char Max, unsigned int uiNrBins)
{
    int i;
    const unsigned char BinSize = (unsigned char) (1 + (Max -
Min) / uiNrBins);

    for (i = Min; i <= Max; i++)  pLUT[i] = (i - Min) /
BinSize;
}

void Interpolate (unsigned char * pImage, int uiXRes, unsigned
long * pulMapLU,
     unsigned long * pulMapRU, unsigned long * pulMapLB,
unsigned long * pulMapRB,
     unsigned int uiXSize, unsigned int uiYSize, unsigned char
* pLUT)
{
    const unsigned int uiIncr = uiXRes-uiXSize;
    unsigned char GreyValue; unsigned int uiNum =
uiXSize*uiYSize;

    unsigned int uiXCoef, uiYCoef, uiXInvCoef, uiYInvCoef,
uiShift = 0;
```

```
    if (uiNum & (uiNum - 1))
    for (uiYCoef = 0, uiYInvCoef = uiYSize; uiYCoef < uiYSize;
      uiYCoef++, uiYInvCoef--,pImage+=uiIncr) {
     for (uiXCoef = 0, uiXInvCoef = uiXSize; uiXCoef <
uiXSize;
          uiXCoef++, uiXInvCoef--) {
         GreyValue = pLUT[*pImage];
         *pImage++ = (unsigned char ) ((uiYInvCoef *
(uiXInvCoef*pulMapLU[GreyValue]
                          + uiXCoef * pulMapRU[GreyValue])
                    + uiYCoef * (uiXInvCoef *
pulMapLB[GreyValue]
                          + uiXCoef * pulMapRB[GreyValue])) /
uiNum);
     }
    }
    else {
     while (uiNum >>= 1) uiShift++;
     for (uiYCoef = 0, uiYInvCoef = uiYSize; uiYCoef <
uiYSize;
          uiYCoef++, uiYInvCoef--,pImage+=uiIncr) {
          for (uiXCoef = 0, uiXInvCoef = uiXSize; uiXCoef <
uiXSize;
            uiXCoef++, uiXInvCoef--) {
            GreyValue = pLUT[*pImage];
            *pImage++ = (unsigned char)((uiYInvCoef*
(uiXInvCoef * pulMapLU[GreyValue]
                          + uiXCoef * pulMapRU[GreyValue])
                    + uiYCoef * (uiXInvCoef *
pulMapLB[GreyValue]
                          + uiXCoef * pulMapRB[GreyValue])) >>
uiShift);
         }
     }
    }
}
**********ColSpaceConv.c**********
//x,y should defined
void rgb2gray(unsigned char *bw,const unsigned char im[][Y*for_RGB])
{
     int i,j;
     for(i=0;i<X;i++)
     {
     for(j=0;j<Y;j++)
          bw[i*Y+j]=((0.21*(*(*(im+i)+j))+
          0.71*(*(*(im+i)+(j+Y)))+
           0.07*(*(*(im+i)+(j+2*Y))))+0.5);
     }
}

*****************************************************************
```

```
**************************data.h*****************************
#define X 256
#define Y 256
#define for_RGB 3
unsigned far char im[X*Y]={255,...//image data
255};
**************************main.c*****************************
#include <stdio.h>
#include <memory.h>
#include <time.h>
#define _TI_ENHANCED_MATH_H 1
#include <math.h>
#include <img_histogram.h>//c6000 IMGLIB
#define UINT8_MIN 0
#define UINT8_MAX 255
#define X image_width
#define Y image_height
#include "ColSpaceConv.c"
#include "CLAHE.c"
#include "image_operation.C"
#include "bwareaopen.c"
#include "MORPH.c"
#pragma DATA_ALIGN (hist, 4)
unsigned short hist[256];

/* scratch buffer needed by IMGLIB */
unsigned short t_hist[1024];
int main()
{
int i,j;
unsigned far char *bwim,temp[256*256],*temp1;
time_t start, stop;
long count;
DSK6713_init();
time(&start);
//rgb2gray(bwim,im);
IMG_histogram(im, X*Y, 1, t_hist, hist);
printf("%d ---
\n",CLAHE(im,X,Y,UINT8_MIN,UINT8_MAX,8,4,128,5.21));
memcpy (temp, im, X*Y*sizeof(unsigned char));
morph_mask_Opening(im,temp,X,Y);
imMinus(im,temp,X,Y);
im2bw(im);
memset(temp,0,X*Y*sizeof(unsigned char));
bwareaopen(bwim,temp,X,Y);
time(&stop);
printf(
"Finished in about %.0f seconds. \n", difftime(stop, start)
);
     return 0;
}
```

LINKER CMD FILE

```
/* Linker command file

-l D:\install\CCStudio_v3.3\C6000\dsk6713\lib\dsk6713bsl.lib
-l D:\install\CCStudio_v3.3\c6200\imglib\lib\img62x.lib
-l D:\install\CCStudio_v3.3\C6000\csl\lib\csl6713.lib
-l D:\install\CCStudio_v3.3\C6000\cgtools\lib\rts6700.lib
-stack 0x00000400
-heap 0x00006000
MEMORY
{
IRAM: origin = 0h len = 016000h
SDRAM: origin = 80000000h len= 400000h
}
SECTIONS
{
.vectors > IRAM
.text > IRAM
.bss > IRAM
.cinit > SDRAM
.const > SDRAM
.stack > SDRAM
.cio > IRAM
.sysmem > IRAM
.data > IRAM
.far > SDRAM
.tables > IRAM
.offchip > SDRAM
mi_SDRAM > SDRAM
}
```

CCS v3.3 OUTPUT SCREENSHOTS

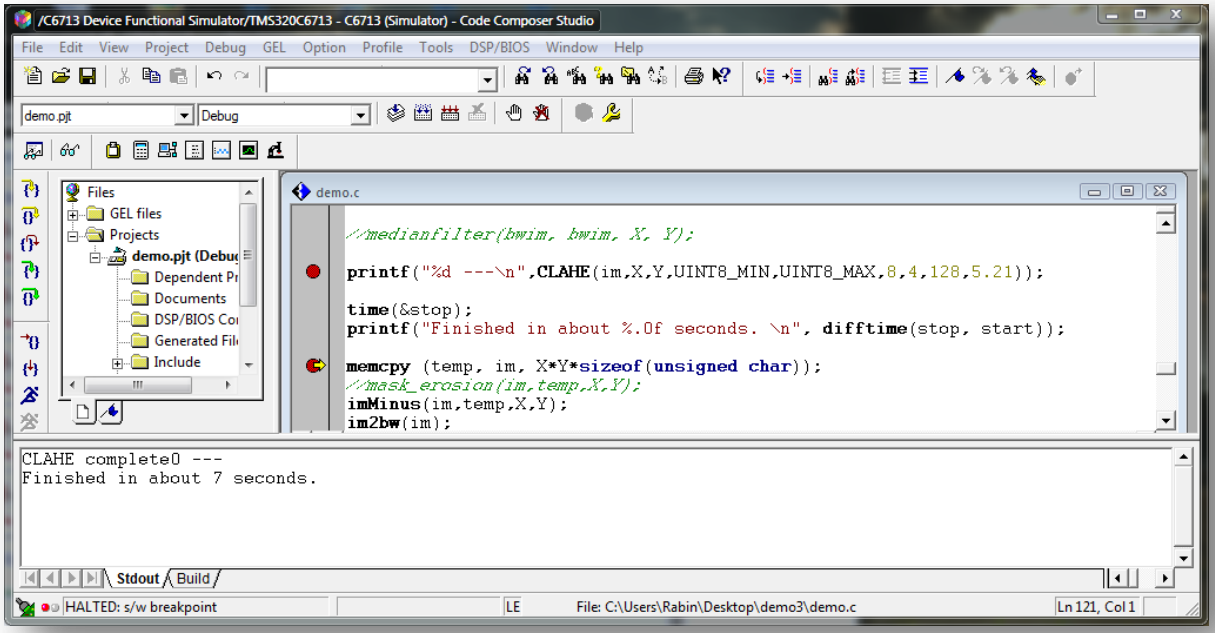


Figure 20 : CLAHE Computation in CCS 3.3

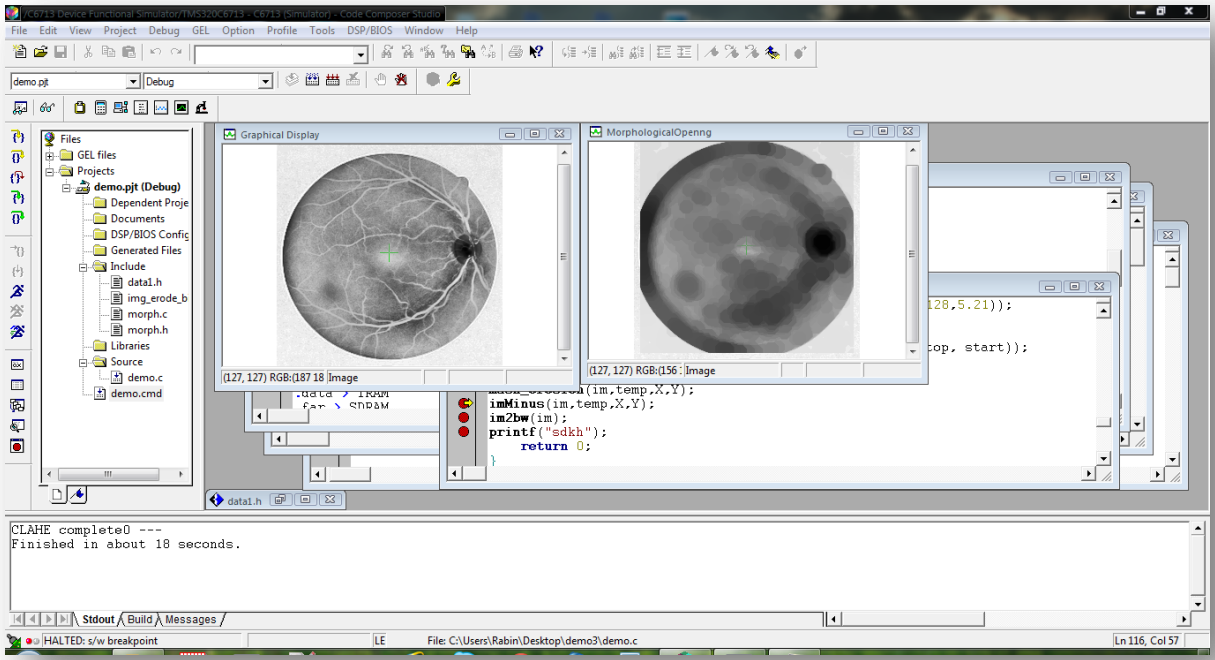


Figure 21: Morphological Filtering in CCS 3.3

SOFTWARE IMPLEMENTATION SCREEN SHOT

Figure 22 : Blood Vessel Detection in Visual Studio 2010 & OpenCV

Figure 23 :Total ANSI C code running under Windows Platform

RESULTS AND DISCUSSION

FUNDAS IMAGE NO	BLOOD VESSEL AREA	DR STAGE	FUNDAS IMAGE NO	BLOOD VESSEL AREA	DR STAGE
Image-1	24807	Mild NPDR	Image-11	23740	Mild NPDR
Image-2	24467	Mild NPDR	Image-12	19467	Normal Eye
Image-3	19491	Normal Eye	Image-13	21006	Normal Eye
Image-4	21144	Normal Eye	Image-14	21586	Normal Eye
Image-5	16954	Normal Eye	Image-15	22519	Mild NPDR
Image-6	25004	Mild NPDR	Image-16	21057	Normal Eye
Image-7	19707	Normal Eye	Image-17	28412	Moderate NPDR
Image-8	14787	Normal Eye	Image-18	35611	Moderate NPDR
Image-9	13753	Normal Eye	Image-19	22867	Mild NPDR
Image-10	17555	Normal Eye	Image-20	51182	PDR

We take 20 images of eye which is basic fundus image from a website. In which some eye images are affected from different types of DR and some are normal eye images. So we take the images as an input into our program and get the output as a no. of blood vessel area. So by knowing the types of DR of the images previously we can easily understand the corresponding Blood vessel area for that types of DR.So next time when we take an eye images to detect whether it is affected from DR or not, we can easily say it by seeing the no. of blood vessel area. Here is a database table for 20 images.

CONCLUSION

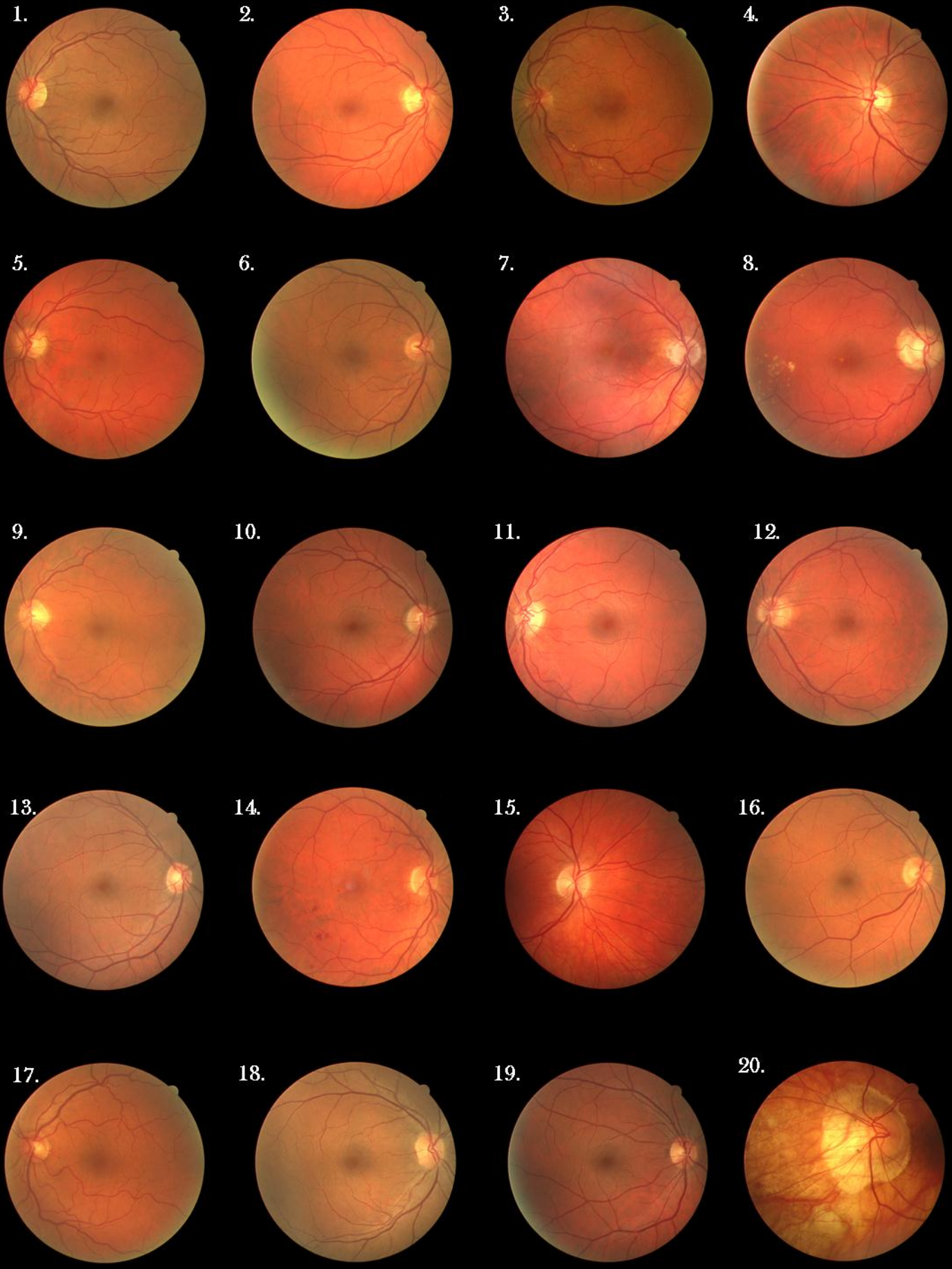
The primary objective of the work is to develop a hardware based system that will be able to identify patients with Diabetic Retinopathy either from color or gray level fundus image.

In the first phase of the work we have mainly highlighted the software based approach with some earlier renowned work on same field and observe that for large scale diagnosis we need to develop a hardware based approach for the ease of detection.

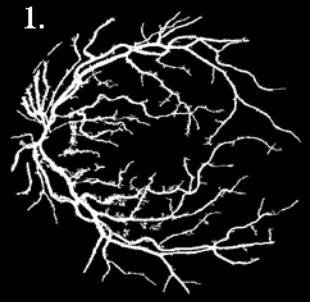
So in the second phase of our work we are mainly concentrated in the hardware based system. So we have done our project up to a stage which is successfully meeting hardware based system requirements. Though more work still need to be done in terms of hardware detection and to reduce the error due to over enhancement of noise and misdetection in this work.

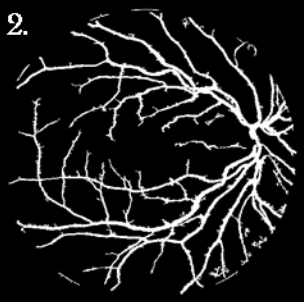
The achievements of this work include very good result in the diagnosis process and it shows how far the use of image processing can replace the tedious and strenuous work at our various hospitals. The results itself reflects the effectiveness of the hardware based implementation technique.

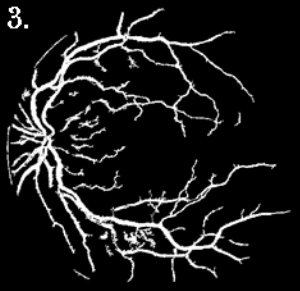
Annexture I: DRIVE Image Database



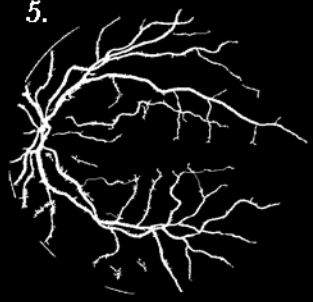
Annexture 2: Detected Blood vessel Images

1. 

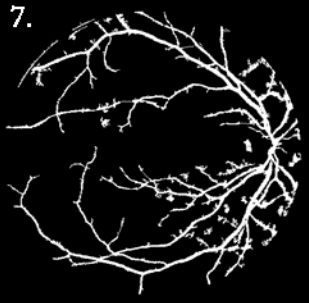
2. 

3. 

4. 

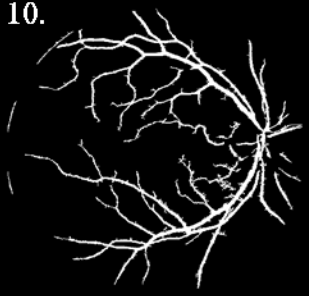
5. 

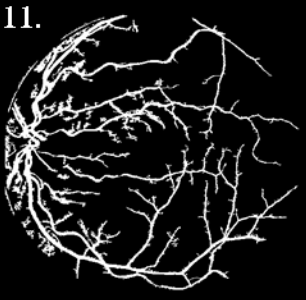
6. 

7. 

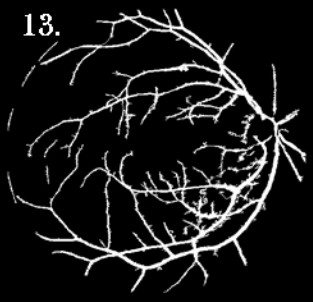
8. 

9. 

10. 

11. 

12. 

13. 

14. 

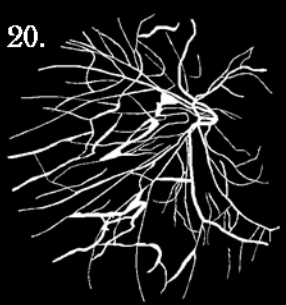
15. 

16. 

17. 

18. 

19. 

20. 

REFFERENCE

[1]Banerjee,R. et al ‘Hardware Based Analysis On Automated Early Detection Of Diabetic Retinopathy’,2nd International Conference On Computer, Communication, Control And Information Technology

[2] Banumathi A, Karthika, R., Kumar.A, “Performance analysis of matched filter techniques for automated detection of blood vessels in retinal images”, Conference on Convergent Technologies for Asia Pacific Region (2003), 2, pp 543–546.

[3] Iqbal,M.I. et al, “Automatic Diagnosis Of Diabetic Retinopathy Using Fundus Images”: By Blekinge Institute Of Technology Referenced October 2006.

[4] Vallabha, D., Dorairaj,et al. "Automated Detection and Classification of Vascular Abnormalities in Diabetic Retinopathy", 38th Asilomar Conference on Signals, Systems and Computers, November 2004.

[5] Wong Li Yun , U. Rajendra Acharya, Y.V. Venkatesh , Caroline Cheec,Lim Choo Min, “Identification of different stages of diabetic retinopathy using retinal optical images”, E.Y.K. Ng / Information Sciences(2008) 178 , pp 106–121.

[6] Gonzalez, Rafael C. and Woods, Richard E. ‘Digital Image Processing using MATLAB’, 2nd edition. Prentice Hall, 2002. ISBN 0-201-18075

[7] IMAGE DATABASE USED :- http://www.isi.uu.nl/Research/Databases/DRIVE/commence_download.php

[8] www.mathworks.com/products/image/functionlist.html