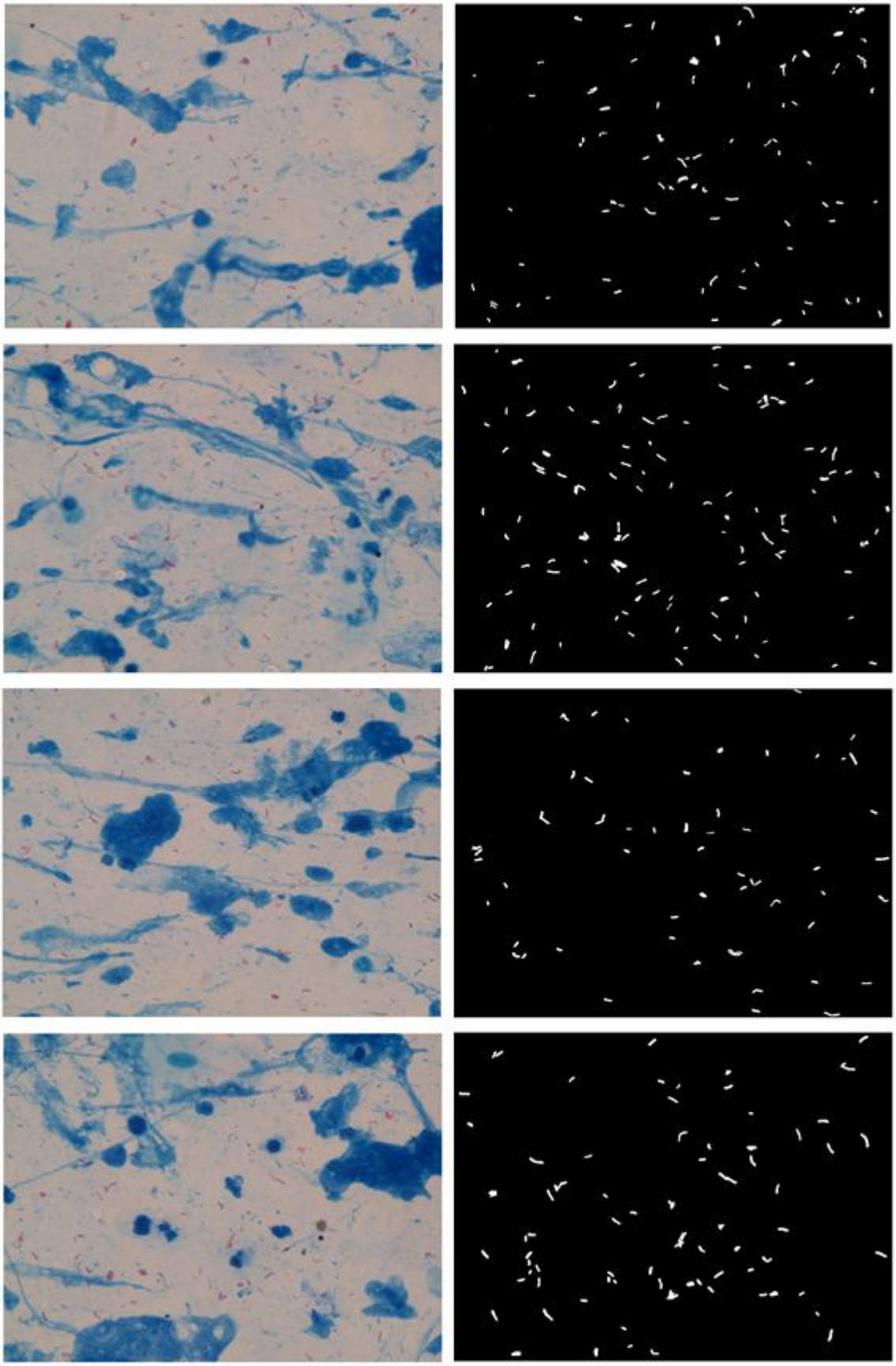
ANEXOS

ANEXO A: EJEMPLOS DE SEGMENTACIÓN MANUAL



ANEXO B: ANÁLISIS DE ESPACIOS DE COLOR

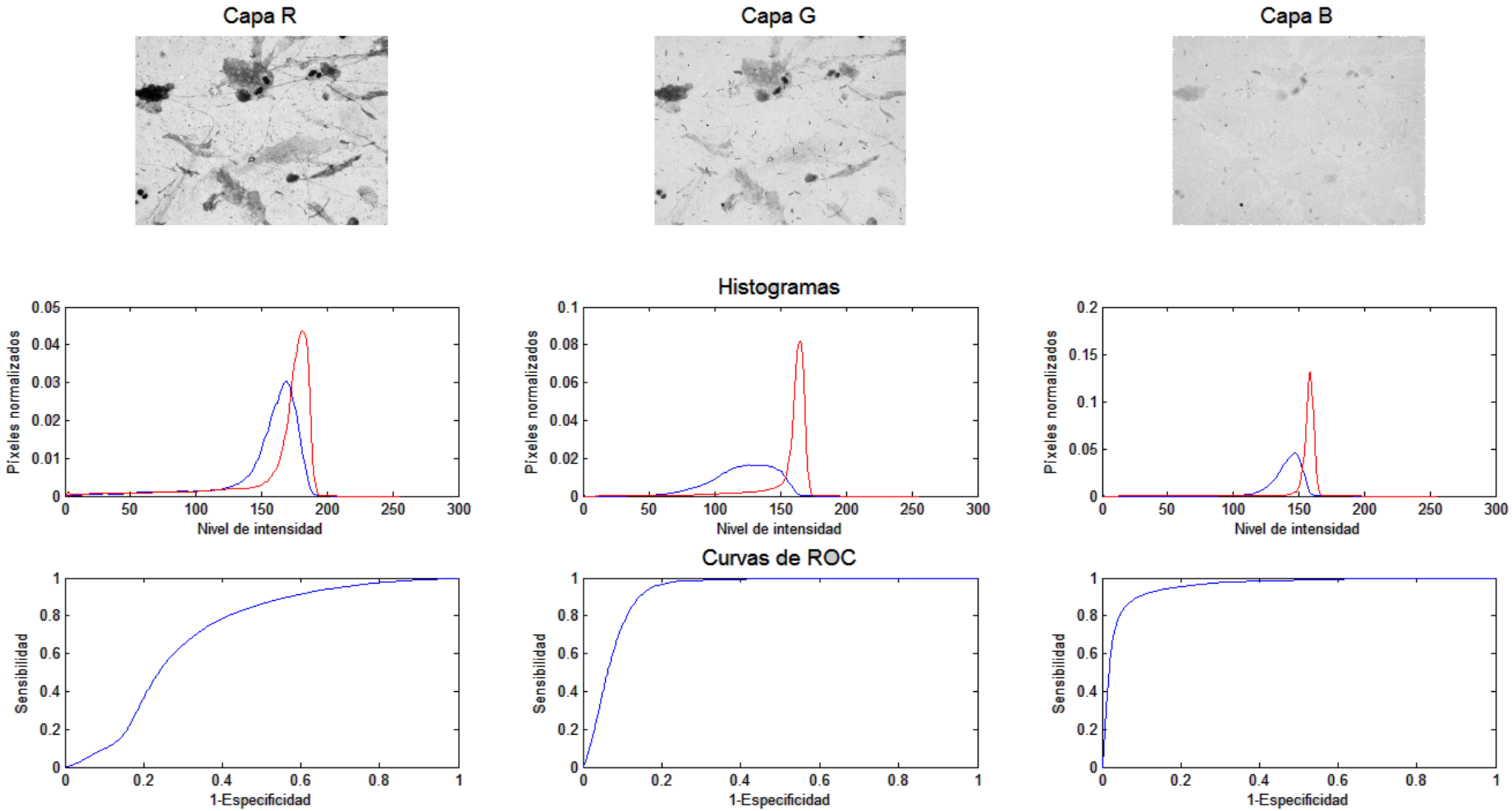


Figura 2.1: Análisis del espacio de color RGB.

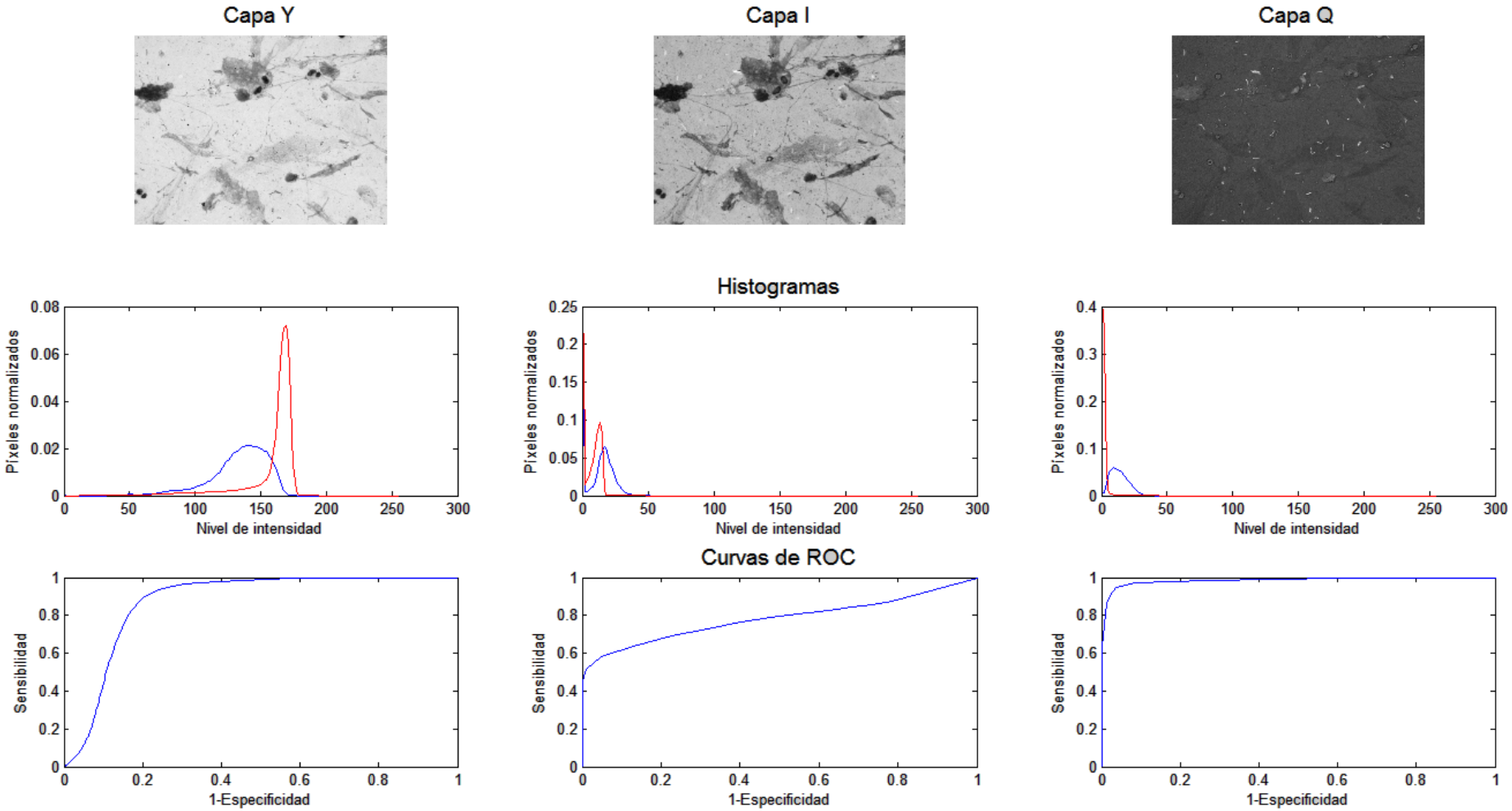


Figura 2.2: Análisis del espacio de color NTSC (YIQ).

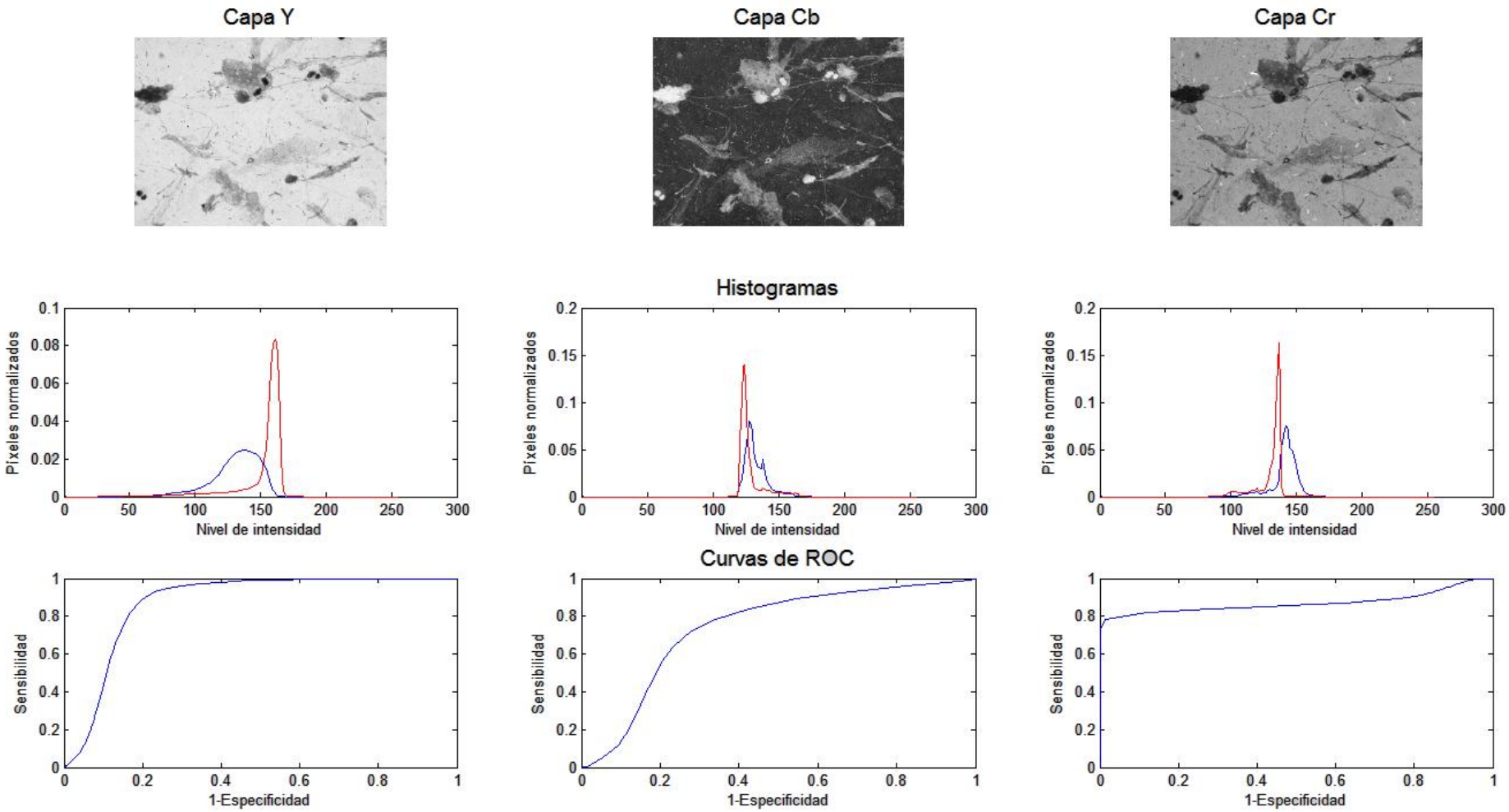


Figura 2.3: Análisis del espacio de color YCbCr.

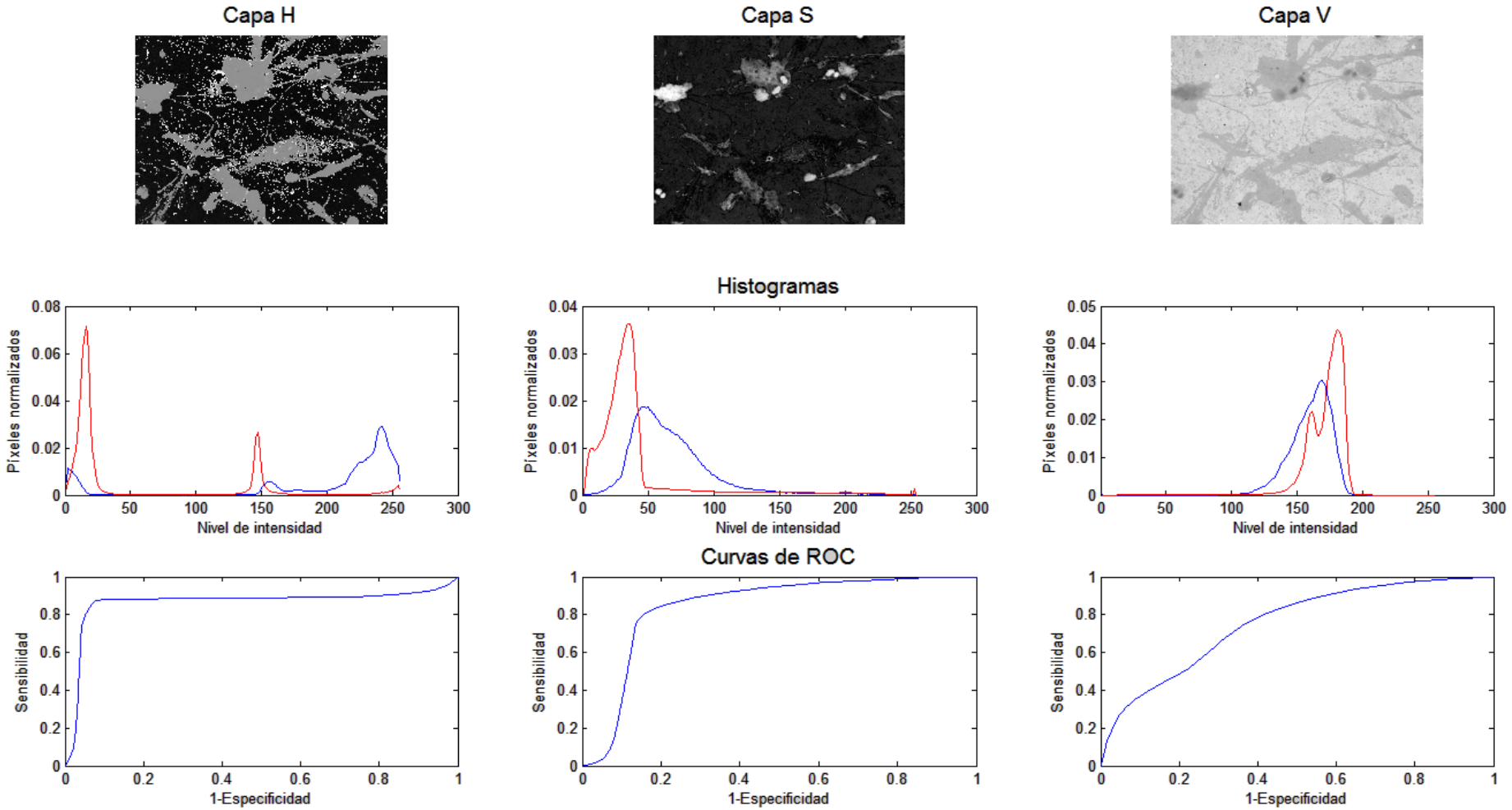


Figura 2.4: Análisis del espacio de color HSV.

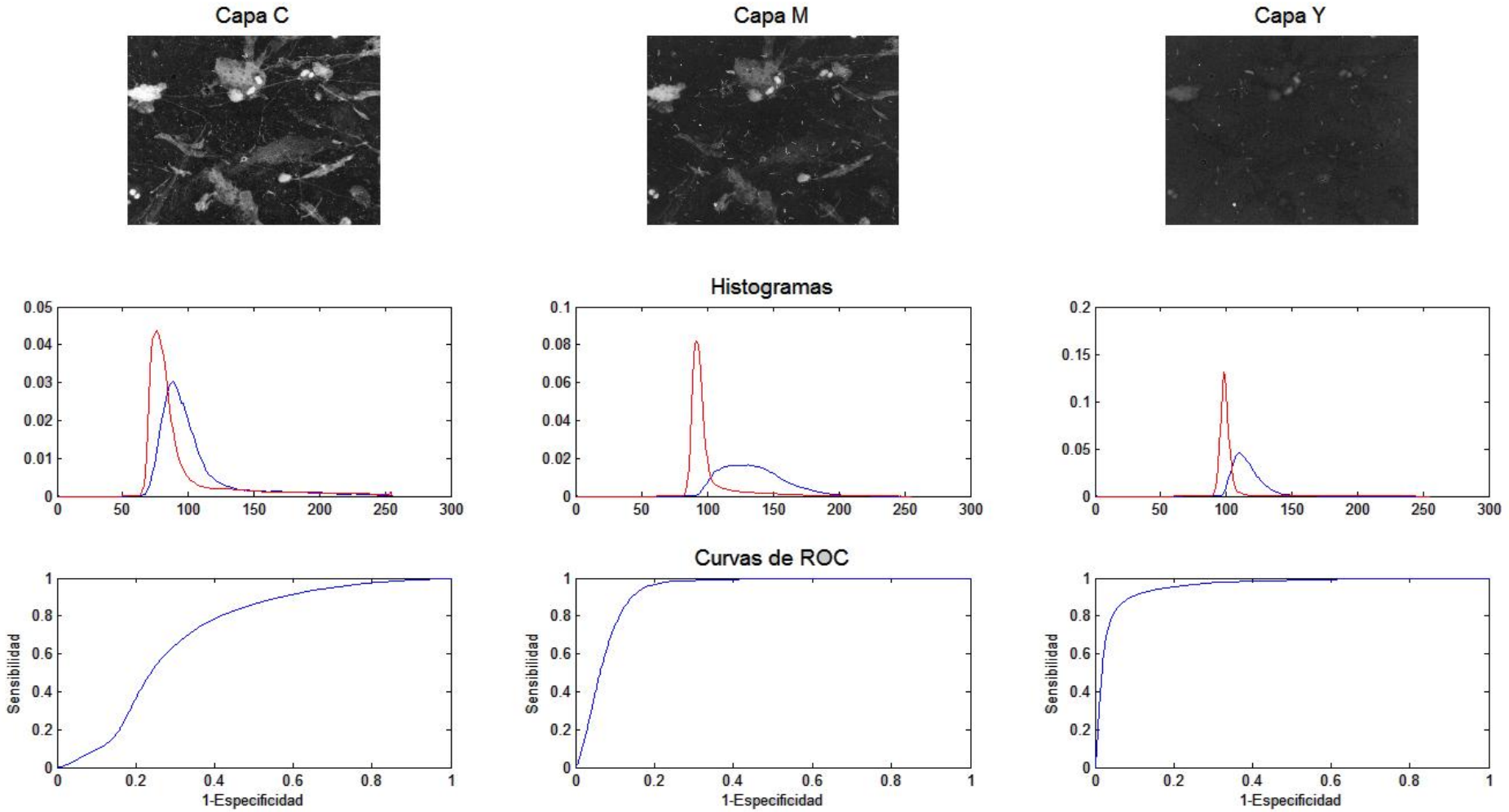


Figura 2.5: Análisis del espacio de color CMY.

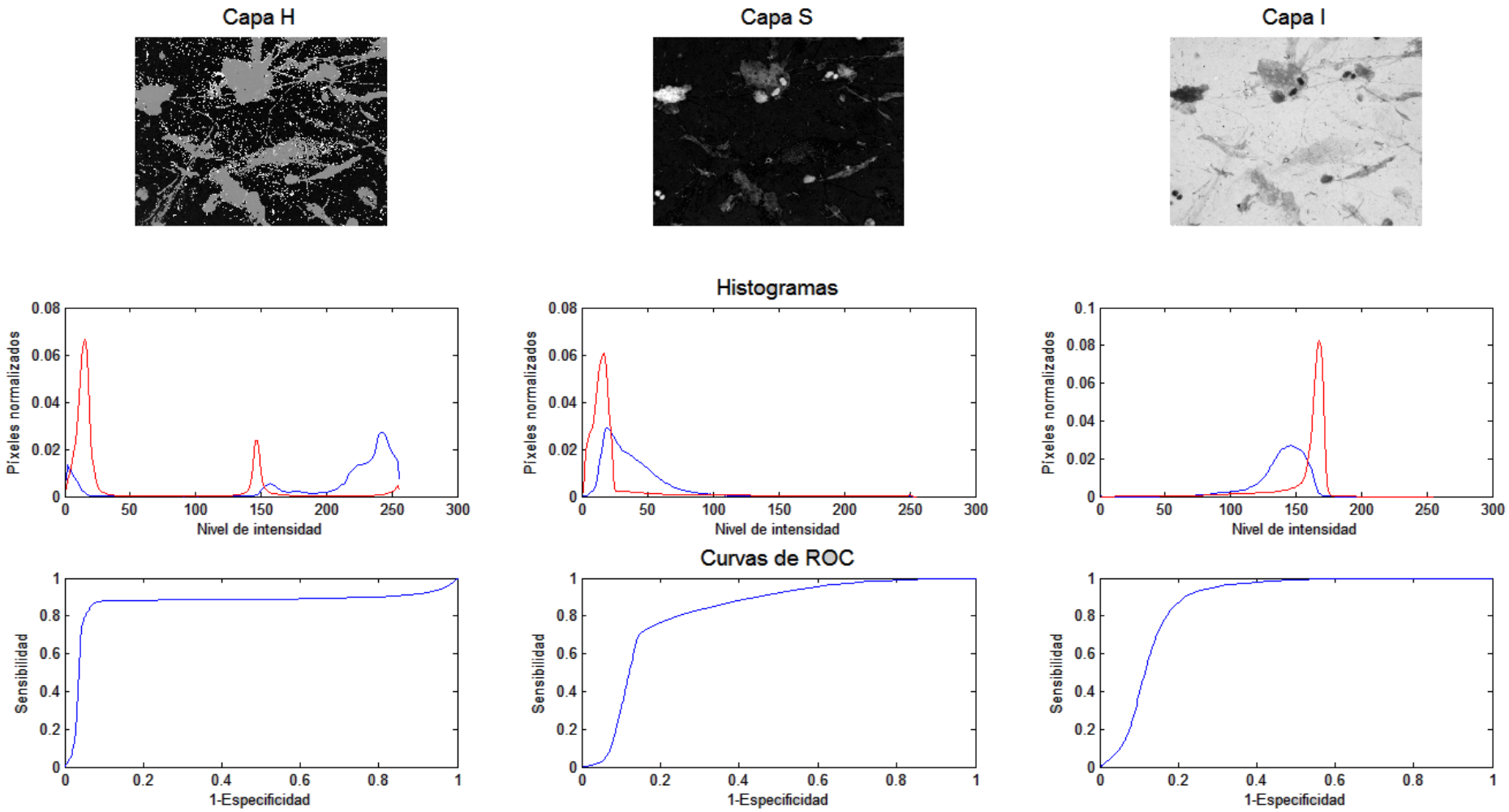


Figura 2.6: Análisis del espacio de color HSI.

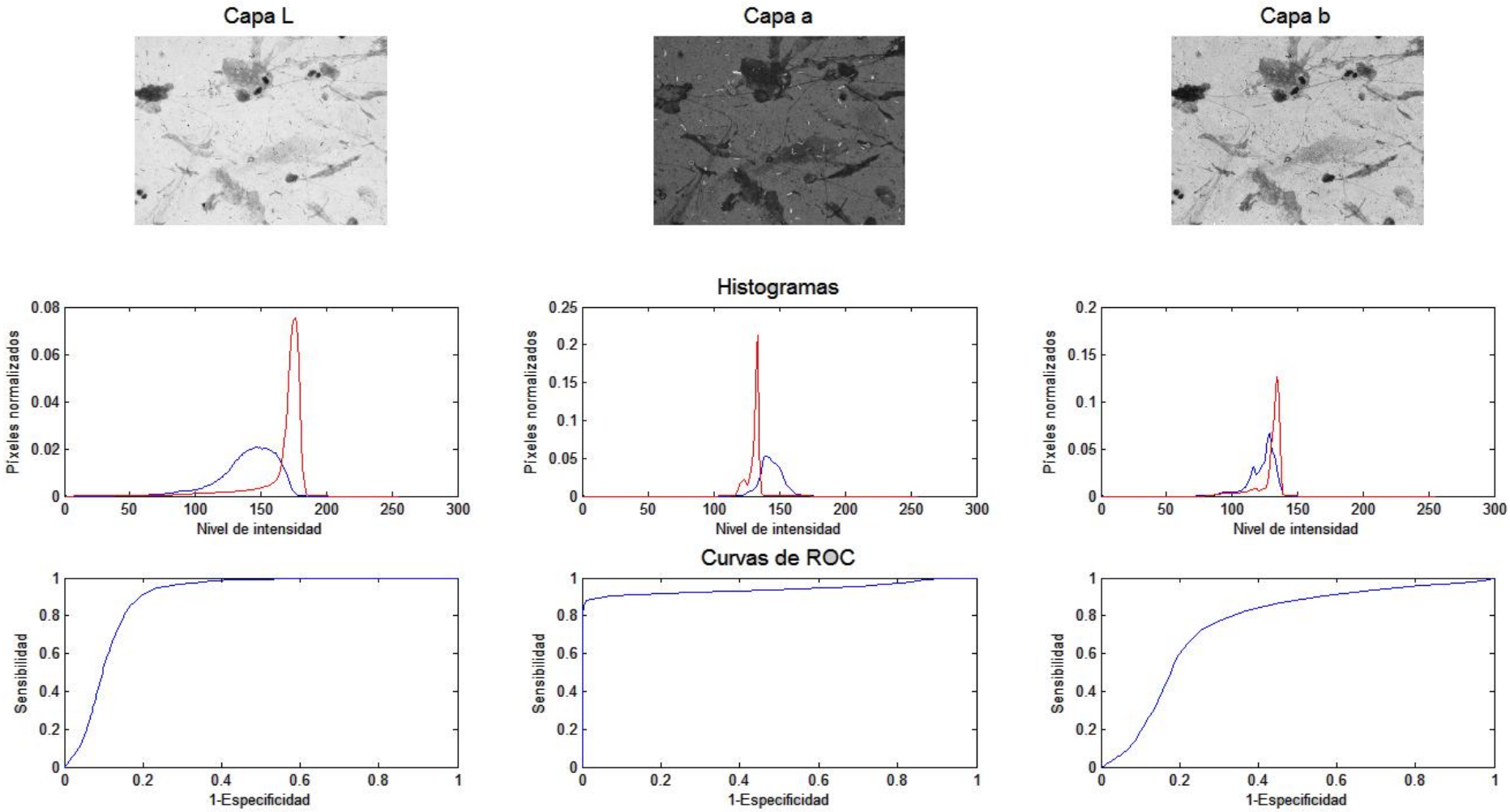


Figura 2.7: Análisis del espacio de color CIE Lab.

ANEXO C: ALGORITMOS EN MATLAB

```
%*************************************************************************************************
%HALLA EL HISTOGRAMA NORMALIZADO DE BACILOS Y DE FONDO PARA
EL ESPACIO DE COLOR RGB Y LOS GUARDA COMO ARCHIVOS .mat
%*************************************************************************************************
clear all
close all
clc

Ab = 0; %area total de bacilos
Af = 0; %area total de fondo
HRt1 = 0; %histograma acumulado de bacilo capa roja
HRt2 = 0; %histograma acumulado de fondo capa roja
HGt1 = 0; %histograma acumulado de bacilo capa verde
HGt2 = 0; %histograma acumulado de fondo capa verde
HBt1 = 0; %histograma acumulado de bacilo capa azul
HBt2 = 0; %histograma acumulado de fondo capa azul

ruta = 'C:\Users\Juan Sato\Desktop\';
ruta1 = fullfile(ruta,'fotos a segmentar\');
filelist1 = dir([ruta1 '*.tif']);
filenames1 = {filelist1.name}';

ruta2 = fullfile(ruta,'Segmentacion manual\');
filelist2 = dir([ruta2 '*.tif']);
filenames2 = {filelist2.name}';

for k=1:15
    file1 = fullfile(ruta1,filenames1{k}); %imagen original
    I1 = imread(file1);
    I1 = im2double(I1);

    file2 = fullfile(ruta2,filenames2{k}); %imagen segmentada
    I2 = imread(file2);
    I2 = im2double(I2);
    I3=~I2;

    PR1 = regionprops(I2);
    A1 = (PR1.Area);       %area de bacilos
    PR2=regionprops(I3);
    A2 = (PR2.Area);       %area de fondo

    Ab = Ab + A1;    %area total de bacilos
    Af = Af + A2;    %area total de fondo

    R = I1(:,:,1);
    R1 = R.*I2; %bacilos
    R2 = R.*I3; %fondo
    HR1 = imhist(R1);
    HRt1 = HRt1 + HR1;
    HR2 = imhist(R2);
    HRt2 = HRt2 + HR2;
```

```
        G = I1(:,:,2);
        G1 = G.*I2;
        G2 = G.*I3;
        HG1 = imhist(G1);
        HGt1 = HGt1 + HG1;
        HG2 = imhist(G2);
        HGt2 = HGt2 + HG2;


        B = I1(:,:,3);
        B1 = B.*I2;
        B2 = B.*I3;
        HB1=imhist(B1);
        HBt1 = HBt1 + HB1;
        HB2=imhist(B2);
        HBt2 = HBt2 + HB2;
end

%Eliminamos los pixeles negros de la imagen segmentada manualmente de los
histogramas
HRt1(1) = HRt1(1)-Af;
HRt2(1) = HRt2(1)-Ab;
HGt1(1) = HGt1(1)-Af;
HGt2(1) = HGt2(1)-Ab;
HBt1(1) = HBt1(1)-Af;
HBt2(1) = HBt2(1)-Ab;

%Normalizamos los histogramas
HR1 = HRt1./Ab;       %histograma de bacilos para la capa R
HR2 = HRt2./Af;       %histograma del fondo para la capa R

HG1 = HGt1./Ab;       %histograma de bacilos para la capa G
HG2 = HGt2./Af;       %histograma del fondo para la capa G

HB1 = HBt1./Ab;       %histograma de bacilos para la capa B
HB2 = HBt2./Af;       %histograma del fondo para la capa B

figure(1);plot(HR1,'b'); hold; plot(HR2,'r');title('Histograma Capa R');
figure(2);plot(HG1,'b'); hold; plot(HG2,'r');title('Histograma Capa G');
figure(3);plot(HB1,'b'); hold; plot(HB2,'r');title('Histograma Capa B');

%Grabamos los histogramas normalizados en un archivo mat
nombre = 'RGB.mat';
save(nombre, 'HR1', 'HR2', 'HG1', 'HG2', 'HB1', 'HB2');


%%
%*************************************************************************************************
%LEE HISTOGRAMAS (ARCHIVOS .MAT), HALLA LAS CURVAS DE ROC Y SUS
RESPECTIVAS ÁREAS BAJO LA CURVA PARA CADA ESPACIO DE COLOR.
%*************************************************************************************************
clear all
close all
clc
```

```
ruta = 'C:\Users\Juan Sato\Desktop\PruebasNI\histogramas\';
filelist = dir([ruta '*.mat']);
filenames = {filelist.name}';

Color = {'CMY'; 'HSI'; 'HSV'; 'Lab'; 'RGB';  'YCbCR'; 'YIQ'};
Capa = {'Curva de ROC C'; 'Curva de ROC M'; 'Curva de ROC Y';
        'Curva de ROC H'; 'Curva de ROC S'; 'Curva de ROC I';
        'Curva de ROC H'; 'Curva de ROC S'; 'Curva de ROC V';
        'Curva de ROC L'; 'Curva de ROC a'; 'Curva de ROC b';
        'Curva de ROC R'; 'Curva de ROC G'; 'Curva de ROC B';
        'Curva de ROC Y'; 'Curva de ROC Cb'; 'Curva de ROC Cr';
        'Curva de ROC Y'; 'Curva de ROC I'; 'Curva de ROC Q'};

c = 1;

for h = 1:7
    HR1=0; HG1=0; HB1=0;
    HR2=0; HG2=0; HB2=0;
    file = fullfile(ruta,filenames{h});
    load(file);

    for t = 0:256
        VP1 = 0;
        FN1 = 0;
        VN1 = 0;
        FP1 = 0;

        VP2 = 0;
        FN2 = 0;
        VN2 = 0;
        FP2 = 0;

        VP3 = 0;
        FN3 = 0;
        VN3 = 0;
        FP3 = 0;

        if t == 0
          FN1=sum(HR1);
          VN1=sum(HR2);

          FN2=sum(HG1);
          VN2=sum(HG2);

          FN3=sum(HB1);
          VN3=sum(HB2);

        else
          VP1=sum(HR1(1:t));
          FP1=sum(HR2(1:t));

          VP2=sum(HG1(1:t));
          FP2=sum(HG2(1:t));

          VP3=sum(HB1(1:t));
```

```
            FP3=sum(HB2(1:t));

            if t~=256
               FN1=sum(HR1(t+1:256));
               VN1=sum(HR2(t+1:256));

               FN2=sum(HG1(t+1:256));
               VN2=sum(HG2(t+1:256));

               FN3=sum(HB1(t+1:256));
               VN3=sum(HB2(t+1:256));
            end

         end

         Sensibilidad1(t+1) = VP1/(VP1+FN1);
         Sensibilidad2(t+1) = VP2/(VP2+FN2);
         Sensibilidad3(t+1) = VP3/(VP3+FN3);

         NoEspecificidad1(t+1) = FP1/(FP1+VN1);
         NoEspecificidad2(t+1) = FP2/(FP2+VN2);
         NoEspecificidad3(t+1) = FP3/(FP3+VN3);

      end

      Area1 = trapz(NoEspecificidad1,Sensibilidad1);
      Area2 = trapz(NoEspecificidad2,Sensibilidad2);
      Area3 = trapz(NoEspecificidad3,Sensibilidad3);

      if Area1<0.5
       Sensibilidad1 = 1 - Sensibilidad1;
       NoEspecificidad1 = 1 - NoEspecificidad1;
       Area1 = -trapz(NoEspecificidad1,Sensibilidad1);
      end

      if Area2<0.5
       Sensibilidad2 = 1 - Sensibilidad2;
       NoEspecificidad2 = 1 - NoEspecificidad2;
       Area2 = -trapz(NoEspecificidad2,Sensibilidad2);
      end

      if Area3<0.5
       Sensibilidad3 = 1 - Sensibilidad3;
       NoEspecificidad3 = 1 - NoEspecificidad3;
       Area3 = -trapz(NoEspecificidad3,Sensibilidad3);
      end


figure(h);title(Color{h});

subplot(1,3,1);plot(NoEspecificidad1,Sensibilidad1);title(Capa{c});ylabel('Sensibilida
d');xlabel('1-Especificidad');legend(['Area=',num2str(Area1)],'Location','NorthEast');

subplot(1,3,2);plot(NoEspecificidad2,Sensibilidad2);title(Capa{c+1});ylabel('Sensibili
```

```
dad');xlabel('1-
Especificidad');legend(['Area=',num2str(Area2)],'Location','NorthEast');

subplot(1,3,3);plot(NoEspecificidad3,Sensibilidad3);title(Capa{c+2});ylabel('Sensibili
dad');xlabel('1-
Especificidad');legend(['Area=',num2str(Area3)],'Location','NorthEast');

c=c+3;
end


%%
%*****************************************************************************************
%CONTEO DE BACILOS DE TUBERCULOSIS EN UNA IMAGEN
%*****************************************************************************************
clear all
close all
clc

%LECTURA DE LA IMAGEN
ruta = 'C:\Users\Juan Sato\Desktop\fotos a segmentar\';
filelist = dir([ruta '*.tif']);
filenames = {filelist.name}';
file = fullfile(ruta,filenames{1});
I1 = imread(file);
figure(1); imshow(I1);title('Imagen Original');


%EXTRACCIÓN DE LA CAPA Q
I2 = im2double(I1);
I3 = rgb2ntsc(I2);
I4 = I3(:,:,3);
figure(2);imshow(I4,[]);title('Capa Q');


%APLICACIÓN DE FILTRO GAUSSIANO
N = fspecial('gaussian',[3 3],0.65);
I5 = imfilter(I4,N);
figure(3);imshow(I5);title('Aplicacion de Filtro Gaussiano');


%DETECCIÓN DE BORDES
gx = fspecial('sobel');
gy = gx';
Gx = imfilter(I5,gx);
Gy = imfilter(I5,gy);
G = sqrt(Gx.^2+Gy.^2);
figure(4);imshow(G); title('Magnitud de la gradiente');
Mas = G>max(G(:))*.095;
figure(7);imshow(Mas,[]);
Mas = bwareaopen(Mas,300);
figure(5);imshow(Mas);title('Deteccion de bordes');
```

```
%PRODUCTO DE LA DETECCIÓN DE BORDES Y LA CAPA Q
B = Mas.*I4;
figure(6);imshow(B,[]);title('Producto de los bordes con la capa Q');


%UMBRALIZACIÓN CON MÉTODO DE OTSU
T = graythresh(B(B~=0));
BW = im2bw(I4,T);
figure(7);imshow(BW);title('Imangen umbralizada');


%ELIMINACION DE OBJETOS PARTIDOS POR EL BORDE DE LA IMAGEN
[eti, neti] = bwlabel(BW);
[F,C] = size(I4);
Borde = zeros(F,C);
borde(1,:) = 1;
borde(F,:) = 1;
borde(:,1) = 1;
borde(:,C) = 1;
for j = 1:neti
   I9 = eti==j;
   I10 = I9.*borde;
   Suma = sum(I10(:));
   if suma>0
      I11 = ~I9;
      BW = BW.*I11;
   end
end


%ELIMINACIÓN DE OBJETOS QUE NO SE ENCUENTRAN EN EL RANGO DE
AREAS
I6 = bwareaopen(BW,200);
I7 = bwareaopen(BW,2000);
I7 = ~I7;
I8 = I6.*I7;


%ELIMINACIÓN DE OBJETOS QUE NO SE ENCUENTRAN EN EL RANGO DE
EXCENTRICIDADES
[eti, neti] = bwlabel(I8);
for i = 1:neti
   I9 = eti==i;
   e = regionprops(I9,'Eccentricity');
   E =(e.Eccentricity);

   if E<0.892942 || E>0.9925
      I10 = ~I9;
      I8 = I8.*I10;
   end
end
figure(9);imshow(I8);title('Resultado final');
%%
```

ANEXO D: ALGORITMO EN C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRGBPixel.h"
#include <iostream>
#include <iomanip>
using namespace std;

int main( int , char * argv[])
{
  float CapaR, CapaG, CapaB, CapaQ;

  int fil, filas=2736, col, columnas=3648, j=0, i=0;

  float **MatrizQ;
  MatrizQ = new float* [filas];
  for(i=0;i<filas;i++)
  {
     MatrizQ[i] = new float [columnas];
  }

  int **Labeled;
  Labeled = new int* [filas];
  for(i=0;i<filas;i++)
  {
     Labeled[i] = new int [columnas];
  }

  float **Matriz;
  Matriz = new float* [filas+2];
  for(i=0;i<(filas+2);i++)
  {
     Matriz[i] = new float [columnas+2];
  }

  float **Zeropad;
  Zeropad = new float* [filas+2];
  for(i=0;i<filas+2;i++)
  {
     Zeropad[i] = new float [columnas+2];
  }

  float **Zeropad2;
  Zeropad2 = new float* [filas+2];
  for(i=0;i<filas+2;i++)
  {
     Zeropad2[i] = new float [columnas+2];
  }

  float **filter1;
  filter1 = new float* [filas];
  for(i=0;i<filas;i++)
  {
```

```
    filter1[i] = new float [columnas];
  }

  float **G;
  G = new float* [filas];
  for(i=0;i<filas;i++)
  {
     G[i] = new float [columnas];
  }

  float **Mas;
  Mas = new float* [filas];
  for(i=0;i<filas;i++)
  {
     Mas[i] = new float [columnas];
  }

  float **Mas2;
  Mas2 = new float* [filas+2];
  for(i=0;i<filas+2;i++)
  {
     Mas2[i] = new float [columnas+2];
  }

  float **Mas3;
  Mas3 = new float* [filas];
  for(i=0;i<filas;i++)
  {
     Mas3[i] = new float [columnas];
  }

  int **binario;
  binario = new int* [filas];
  for(i=0;i<filas;i++)
  {
     binario[i] = new int [columnas];
  }

//*************************************************************************************
// CARGAMOS LA IMAGEN
//*************************************************************************************
  typedef itk::RGBPixel< float > PixelType;
  typedef itk::Image< PixelType, 2 > ImageType;
  typedef itk::ImageFileReader< ImageType >  ReaderType;

  ReaderType::Pointer reader = ReaderType::New();

  reader->SetFileName("C:\\Users\\Juan Sato\\Desktop\\Fotos en
jpg\\Foto001_1.jpg");

  reader->Update();

  std::cout<<"Imagen leida \n"<<endl;

  ImageType::Pointer image = reader->GetOutput();
```

```
 ImageType::IndexType pixelIndex;

 std::cout<<"Antes del for loop\n"<<endl;

 for (fil=0;fil<filas;fil++)
 {
          for (col=0;col<columnas;col++)
          {
                    pixelIndex[0] =  col; //#columna
                    pixelIndex[1] =  fil; //#fila

                    PixelType onePixel = image->GetPixel( pixelIndex );

                    PixelType::ValueType red   = onePixel.GetRed();
                    PixelType::ValueType green = onePixel.GetGreen();
                    PixelType::ValueType blue  = onePixel.GetBlue();

                    red   = onePixel[0];  // extract Red   component
                    green = onePixel[1];  // extract Green component
                    blue  = onePixel[2];  // extract Blue  component

                    CapaR = red;
                    CapaG = green;
                    CapaB = blue;

                    CapaQ=((0.211497340306828)*(CapaR) + (-
0.522910690302974)*(CapaG) + (0.311413349996145)*(CapaB))/255;

          MatrizQ[fil][col] = CapaQ; //MATRIZ DE LA CAPA Q

          filter1[fil][col] = 0; //LLENO DE CEROS FILTER1

          Labeled[fil][col] = 0; //LLENO DE CEROS LABELED

          binario[fil][col] = 0; //LLENO DE CEROS BINARIO

           }
 }
 std::cout << "Se extrajo la Capa Q\n";
 std::cout << "CALCULO DEL VALOR UMBRAL\n";



//*********************************************************************************************
// APLICAMOS FILTRO GAUSSIANO Y DETECCION DE BORDES
//*********************************************************************************************

//Zeropadding para filtro gaussiano
 std::cout << "    Aplicando filtro Gaussiano... \n";

 for (fil=0;fil<(filas+2);fil++)
 {
    for (col=0;col<(columnas+2);col++)
          {
```

```
                Zeropad[fil][col] = 0; //LLENO DE CEROS ZEROPAD
                Zeropad2[fil][col] = 0; //LLENO DE CEROS ZEROPAD2
            }
  }

  for (fil=1;fil<(filas+1);fil++)
  {
            for (col=1;col<(columnas+1);col++)
            {
                Zeropad[fil][col] = MatrizQ[fil-1][col-1];
            }
  }

//Se aplica filtro gaussiano
  for (fil=1;fil<filas+1;fil++)
  {
     for (col=1;col<columnas+1;col++)
     {
        filter1[fil-1][col-1] = 0.036067039662004*(Zeropad[fil-1][col-1]+Zeropad[fil-
1][col+1]+Zeropad[fil+1][col-1]+Zeropad[fil+1][col+1]) +
0.117779163136869*(Zeropad[fil-1][col]+Zeropad[fil][col-
1]+Zeropad[fil][col+1]+Zeropad[fil+1][col]) +  0.384615188804507*Zeropad[fil][col];
     }
  }

//Zeropadding para deteccion de bordes
  std::cout << "    Deteccion de bordes... \n";
  for (fil=1;fil<(filas+1);fil++)
  {
            for (col=1;col<(columnas+1);col++)
            {
                Zeropad2[fil][col] = filter1[fil-1][col-1];
            }
  }

//Se aplica deteccion de bordes
  float maximo = 0;
  float Gx,Gy;

  for (fil=1;fil<filas+1;fil++)
  {
     for (col=1;col<columnas+1;col++)
     {
        Gx = Zeropad2[fil-1][col-1] + 2*Zeropad2[fil-1][col] + Zeropad2[fil-1][col+1] -
Zeropad2[fil+1][col-1] - 2*Zeropad2[fil+1][col] - Zeropad2[fil+1][col+1];
        Gy = Zeropad2[fil-1][col-1] - Zeropad2[fil-1][col+1] + 2*Zeropad2[fil][col-1] -
2*Zeropad2[fil][col+1] + Zeropad2[fil+1][col-1] - Zeropad2[fil+1][col+1];
        G[fil-1][col-1] = pow((Gx*Gx + Gy*Gy),0.5);

        if (fil==1 && col==1)
        {
           maximo = G[fil-1][col-1];
        }
        else
        {
```

```
            if (G[fil-1][col-1] >= maximo)
            {
               maximo = G[fil-1][col-1];
            }
         }
      }
   }

   for (fil=0;fil<filas;fil++)
   {
      for (col=0;col<columnas;col++)
      {
         if (G[fil][col] > (maximo*0.095))
         {
            Mas[fil][col] = 1;
         }
         else
         {
            Mas[fil][col] = 0;
         }
      }
   }


//*********************************************************************************************
//ELIMINAMOS OBJETOS PEQUEÑOS Y MULTIPLICAMOS LAS MATRICES
//*********************************************************************************************
   std::cout << "    Eliminando objetos pequeños... \n";
   for (fil=0;fil<(filas+2);fil++)
   {
            for (col=0;col<(columnas+2);col++)
            {
               Mas2[fil][col] = 0;
            }
   }

   for (fil=1;fil<(filas+1);fil++)
   {
            for (col=1;col<(columnas+1);col++)
            {
               Mas2[fil][col] = Mas[fil-1][col-1];
            }
   }

   int fcola = 0;

   int *P1;
   P1 = (int *) malloc(sizeof(int)*2);
   int *P2;
   P2 = (int *) malloc(sizeof(int)*2);

   int  fil2, col2, pos, pos2, x, y, f3, c3;

   float **cola;
   cola = new float* [9980928];
```

```
for(i=0;i<9980928;i++)
{
    cola[i] = new float [2];
}

for (fil2=1;fil2<(filas+1);fil2++)
{
        for (col2=1;col2<(columnas+1);col2++)
        {
            fil = fil2;
            col = col2;
            if (Mas2[fil][col] == 1)
            {
                Mas2[fil][col] = 0;
                fcola = 0;
                cola[fcola][0] = fil;
                cola[fcola][1] = col;
                P2[0] = 0;
                P2[1] = 0;
                P1[0] = cola[fcola][0];
                P1[1] = cola[fcola][1];
                pos = 1;
                pos2 = 1;

                while (pos2 <= pos)
                {
                    for (x=-1;x<2;x++)
                    {
                        for (y=-1;y<2;y++)
                        {
                            if (Mas2[fil+x][col+y] == 1)
                            {
                                pos++;
                                Mas2[fil+x][col+y] = 0;
                                fcola++;
                                cola[fcola][0] = fil+x;
                                cola[fcola][1] = col+y;
                            }
                        }
                    }
                    pos2++;
                    P1[0] = cola[pos-1][0];
                P1[1] = cola[pos-1][1];
                    if (pos2 <= pos)
                    {
                        P2[0] = cola[pos2-1][0];
                        P2[1] = cola[pos2-1][1];
                    }
                    else
                    {
                        P2[0] = cola[pos-1][0];
                        P2[1] = cola[pos-1][1];
                }
                    fil = P2[0];
                    col = P2[1];
```

```
                }
                if (fcola < 399)
                {
                    for (j=0;j<fcola+1;j++)
                    {
                        f3 = cola[j][0];
                        c3 = cola[j][1];
                        Mas[f3-1][c3-1] = 0;
                    }
                }
            }
        }
    }

//Multiplicamos las matrices
    std::cout << "    Multiplicando mascara de gradiente por la imagen... \n";
    float limite = 0.522910690302973;
    for (fil=0;fil<filas;fil++)
    {
        for (col=0;col<columnas;col++)
        {
            if (Mas[fil][col] == 0)
            {
                Mas3[fil][col] = 0;
            }

            if (Mas[fil][col] == 1)
            {
                Mas3[fil][col] = (MatrizQ[fil][col] + limite)*(255/(2*limite));
                Mas3[fil][col] = floor(Mas3[fil][col] + 0.5);
            }
        }
    }


//*********************************************************************************************
//APLICACION DEL METODO DE OTSU Y UMBRALIZACION
//*********************************************************************************************
    std::cout << "    Umbralizacion por Metodo de Otsu... \n";

    //Hallamos el histograma
    int N;
    float ceros=0;
    float *hist;
    hist = (float *) malloc(sizeof(float)*256);
    for (i=0;i<256;i++)
    {
        hist[i] = 0;
    }
    for (fil=0;fil<filas;fil++)
    {
        for (col=0;col<columnas;col++)
        {
            if (Mas3[fil][col] != 0)
            {
```

```
            N = Mas3[fil][col];
            hist[N] = hist[N] + 1;
        }
        else
        {
            ceros++;
        }
    }
}


//Algoritmo de otsu
float *histnormalizado;
histnormalizado = (float *) malloc(sizeof(float)*256);
float w = 0, u = 0, uT = 0, work1, maximo1 = 0, T, T1;
int k = 256;

for (i=1;i<=k;i++)
{
    histnormalizado[i-1] = hist[i-1]/(9980928-ceros);
}

for (i=1;i<=k;i++)
{
    uT = uT + i*histnormalizado[i-1];
}

for (j=1;j<=k;j++)
{
    w = 0;
    u = 0;
    for (i=1;i<=j;i++)
    {
        w = w + histnormalizado[i-1];
        u = u + i*histnormalizado[i-1];
    }
    work1 = (uT*w -u)*(uT*w -u)/(w*(1-w));

    if (maximo1 < work1)
    {
        maximo1 = work1;
        T=j;
    }
}

T1 = (T*(2*limite)/255) - limite + 0.0003;  //VALOR UMBRAL


//Umbralizacion
for (fil=0;fil<(filas);fil++)
{
        for (col=0;col<(columnas);col++)
        {
            if (MatrizQ[fil][col]>T1)
      {
```

```
                MatrizQ[fil][col] = 1;
            }
            else
            {
                MatrizQ[fil][col] = 0;
            }
                }
    }
    //std::cout<<"    Fin de la umbralizacion. Valor umbral calculado: "<< T1 << "\n";


//*********************************************************************************************
//ELIMINACION DE OBJETOS QUE ESTEN FUERA DEL RANGO DE AREAS
//*********************************************************************************************
    //std::cout << "    Eliminando objetos pequeños... \n";
    for (fil=0;fil<(filas+2);fil++)
    {
            for (col=0;col<(columnas+2);col++)
            {
                Mas2[fil][col] = 0;
            }
    }

    for (fil=1;fil<(filas+1);fil++)
    {
            for (col=1;col<(columnas+1);col++)
            {
                Mas2[fil][col] = MatrizQ[fil-1][col-1];
            }
    }

    fcola = 0;

    for (fil2=1;fil2<(filas+1);fil2++)
    {
            for (col2=1;col2<(columnas+1);col2++)
            {
                fil = fil2;
                col = col2;
                if (Mas2[fil][col] == 1)
                {
                    Mas2[fil][col] = 0;
                    fcola = 0;
                    cola[fcola][0] = fil;
                    cola[fcola][1] = col;
                    P2[0] = 0;
                    P2[1] = 0;
                    P1[0] = cola[fcola][0];
                    P1[1] = cola[fcola][1];
                    pos = 1;
                    pos2 = 1;

                    while (pos2 <= pos)
                    {
                        for (x=-1;x<2;x++)
```

```
                    {
                        for (y=-1;y<2;y++)
                        {
                            if (Mas2[fil+x][col+y] == 1)
                            {
                                pos++;
                                Mas2[fil+x][col+y] = 0;
                                fcola++;
                                cola[fcola][0] = fil+x;
                                cola[fcola][1] = col+y;
                            }
                        }
                    }
                    pos2++;
                    P1[0] = cola[pos-1][0];
                P1[1] = cola[pos-1][1];
                    if (pos2 <= pos)
                    {
                        P2[0] = cola[pos2-1][0];
                        P2[1] = cola[pos2-1][1];
                    }
                    else
                    {
                        P2[0] = cola[pos-1][0];
                        P2[1] = cola[pos-1][1];
                }
                    fil = P2[0];
                    col = P2[1];
                }
                if (fcola < 200 || fcola > 2000)
                {
                    for (j=0;j<fcola+1;j++)
                    {
                        f3 = cola[j][0];
                        c3 = cola[j][1];
                        MatrizQ[f3-1][c3-1] = 0;
                    }
                }
            }
        }
    }


//*********************************************************************************************
//ELIMINACION DE BACILOS PARTIDOS POR EL BORDE DE LA IMAGEN
//*********************************************************************************************
  for (fil=0;fil<(filas+2);fil++)
  {
            for (col=0;col<(columnas+2);col++)
            {
                Matriz[fil][col] = 0;
            }
  }

  for (fil=1;fil<(filas+1);fil++)
```

```
{
        for (col=1;col<(columnas+1);col++)
        {
           Matriz[fil][col] = MatrizQ[fil-1][col-1];
        }
}

int eti = 0;


for (fil2=1;fil2<filas+1;fil2=fil2+filas-1)
{
   for (col2=1;col2<columnas+1;col2++)
   {
      fil = fil2;
      col = col2;
      if (Matriz[fil][col] == 1)
      {
         Matriz[fil][col] = 0;
         fcola = 0;
               cola[fcola][0] = fil;
               cola[fcola][1] = col;
               P2[0] = 0;
               P2[1] = 0;
               P1[0] = cola[fcola][0];
               P1[1] = cola[fcola][1];
               pos = 1;
               pos2 = 1;

               while (pos2 <= pos)
               {
                  for (x=-1;x<2;x++)
                  {
                     for (y=-1;y<2;y++)
                     {
                        if (Matriz[fil+x][col+y] == 1)
                        {
                           pos++;
                           Matriz[fil+x][col+y] = 0;
                           fcola++;
                           cola[fcola][0] = fil+x;
                           cola[fcola][1] = col+y;
                        }
                     }
                  }
                  pos2++;
                  P1[0] = cola[pos-1][0];
            P1[1] = cola[pos-1][1];
                  if (pos2 <= pos)
                  {
                     P2[0] = cola[pos2-1][0];
                     P2[1] = cola[pos2-1][1];
                  }
                  else
                  {
```

```
                    P2[0] = cola[pos-1][0];
                    P2[1] = cola[pos-1][1];
                }
                    fil = P2[0];
                    col = P2[1];
                }
            }
        }
    }

for (fil2=1;fil2<filas+1;fil2++)
{
    for (col2=1;col2<columnas+1;col2=col2+columnas-1)
    {
        fil = fil2;
        col = col2;
        if (Matriz[fil][col] == 1)
        {
            Matriz[fil][col] = 0;
            fcola = 0;
                cola[fcola][0] = fil;
                cola[fcola][1] = col;
                P2[0] = 0;
                P2[1] = 0;
                P1[0] = cola[fcola][0];
                P1[1] = cola[fcola][1];
                pos = 1;
                pos2 = 1;

                while (pos2 <= pos)
                {
                    for (x=-1;x<2;x++)
                    {
                        for (y=-1;y<2;y++)
                        {
                            if (Matriz[fil+x][col+y] == 1)
                            {
                                pos++;
                                Matriz[fil+x][col+y] = 0;
                                fcola++;
                                cola[fcola][0] = fil+x;
                                cola[fcola][1] = col+y;
                            }
                        }
                    }
                    pos2++;
                    P1[0] = cola[pos-1][0];
                P1[1] = cola[pos-1][1];
                    if (pos2 <= pos)
                    {
                        P2[0] = cola[pos2-1][0];
                        P2[1] = cola[pos2-1][1];
                    }
                    else
                    {
```

```
                    P2[0] = cola[pos-1][0];
                    P2[1] = cola[pos-1][1];
                }
                    fil = P2[0];
                    col = P2[1];
                }
            }
        }
    }

//*********************************************************************************************
//ALGORITMO DE ETIQUETADO
//*********************************************************************************************
  for (fil2=1;fil2<(filas+1);fil2++)
  {
          for (col2=1;col2<(columnas+1);col2++)
          {
             fil = fil2;
             col = col2;
             if (Matriz[fil][col] == 1)
             {
                eti++;
                Matriz[fil][col] = 0;
                Labeled[fil-1][col-1] = eti;
                fcola = 0;
                cola[fcola][0] = fil;
                cola[fcola][1] = col;
                P2[0] = 0;
                P2[1] = 0;
                P1[0] = cola[fcola][0];
                P1[1] = cola[fcola][1];
                pos = 1;
                pos2 = 1;

                while (pos2 <= pos)
                {
                   for (x=-1;x<2;x++)
                   {
                      for (y=-1;y<2;y++)
                      {
                         if (Matriz[fil+x][col+y] == 1)
                         {
                            pos++;
                            Matriz[fil+x][col+y] = 0;
                            Labeled[fil+x-1][col+y-1] = eti;
                            fcola++;
                            cola[fcola][0] = fil+x;
                            cola[fcola][1] = col+y;
                         }
                      }
                   }
                   pos2++;
                   P1[0] = cola[pos-1][0];
                P1[1] = cola[pos-1][1];
```

```
                    if (pos2 <= pos)
                    {
                        P2[0] = cola[pos2-1][0];
                        P2[1] = cola[pos2-1][1];
                    }
                    else
                    {
                        P2[0] = cola[pos-1][0];
                        P2[1] = cola[pos-1][1];
                }
                        fil = P2[0];
                        col = P2[1];
                    }
                }
            }
  }

  std::cout << "Fin del Etiquetado\n";
  std::cout<<" Numero de etiquetas: "<< eti << "\n"<<endl;


//*****************************************************************************************
//ALGORITMO DE CONTEO (ANALISIS DE EXCENTRICIDAD)
//*****************************************************************************************
  int **Objeto;
  Objeto = new int* [filas];
  for(i=0;i<(filas);i++)
  {
     Objeto[i] = new int [columnas];
  }

  int count = 0;

  float e, mc00, mc20, mc02, mc11, mcn20, mcn02, mcn11, xc, yc, hu1, hu2, rhu2,
qmc00;
  int f, c, label;
  int filmin, filmax, colmin, colmax;
  int primerpixel;

  float suma00, suma01, suma10;

  for (label=1;label<=eti;label++)
  {
     suma00 = 0;
     primerpixel = 0;

     for (fil=0;fil<filas;fil++)
     {
        for (col=0;col<columnas;col++)
        {
           if (Labeled[fil][col] == label)
           {
              Objeto[fil][col] = 1;
              suma00 = suma00 + 1;
```

```
            if (primerpixel==0)
            {
               filmin=fil; filmax=fil; colmin=col; colmax=col;
               primerpixel = 1;
            }
            else
            {
               if (fil>filmax)
               {
                  filmax=fil;
               }

               if (col<colmin)
               {
                  colmin=col;
               }

               if (col>colmax)
               {
                  colmax=col;
               }
            }
         }
         else
         {
            Objeto[fil][col] = 0;
         }
      }
   }

   //Halla centroide
      suma01 = 0;
      suma10 = 0;
      for (c=colmin;c<=colmax;c++)
      {
         for (f=filmin;f<=filmax;f++)
         {
            if (Objeto[f][c] == 1)
            {
               suma01 = suma01 + f;
               suma10 = suma10 + c;
            }
         }
      }
      xc = suma10/suma00;
      yc = suma01/suma00;

     //Halla momentos centrales
      mc00 = suma00;
      mc20 = 0;
      mc02 = 0;
      mc11 = 0;

      for (f=filmin;f<=filmax;f++)
      {
```

```
            for (c=colmin;c<=colmax;c++)
            {
              if (Objeto[f][c]==1)
              {
                 mc20 = mc20 + (c-xc)*(c-xc);
                 mc02 = mc02 + (f-yc)*(f-yc);
                 mc11 = mc11 + (c-xc)*(f-yc);
              }
            }
          }

        //Halla momentos centrales normalizados
          qmc00=(mc00*mc00);
          mcn20 = mc20/qmc00;
          mcn02 = mc02/qmc00;
          mcn11 = mc11/qmc00;

        //Halla excentricidad
          hu1 = mcn20 + mcn02;
          hu2 = (mcn20 - mcn02)*(mcn20 - mcn02) + 4*(mcn11*mcn11);
          rhu2=pow(hu2,0.5);
          e = ((2*rhu2)/(hu1+rhu2));

          if (e>=0.797345415364 && e<=0.98505625)
          {
             count++;

             for (f=0;f<filas;f++)
             {
                for (c=0;c<columnas;c++)
                {
                   if (Labeled[f][c]==label)
                   {
                      binario[f][c]=1;
                   }
                }
             }
          }
       std::cout<<" Etiqueta numero: "<< label << "\n"<<endl;
       std::cout<<" Cuenta: "<< count << "\n"<<endl;
    }

    std::cout<<" Numero de etiquetas: "<< eti << "\n"<<endl;
    std::cout<<" Numero de bacilos: "<< count << "\n"<<endl;



//*********************************************************************************************
//GUARDA IMAGEN BINARIA RESULTANTE
//*********************************************************************************************
    typedef unsigned char InputPixelType;
    typedef itk::Image< InputPixelType, 2 > InputImageType;
    typedef itk::ImageFileWriter< InputImageType > WriterType;
    ImageType::SizeType size = {{(long unsigned int)columnas,(long unsigned
int)filas}};
```

```
  ImageType::RegionType region;
  region.SetSize(size);

  InputImageType::Pointer img = InputImageType::New();
  img->SetRegions(region);
  img->Allocate();

  for (fil=0;fil<filas;fil++)
  {
          for (col=0;col<columnas;col++)
          {
                  pixelIndex[0] =  col; //#columna
                  pixelIndex[1] =  fil; //#fila

                  if (binario[fil][col]==1)
                  {
                     img->SetPixel( pixelIndex, (unsigned char)( 255 ) );
                  }
                  else
                  {
                     img->SetPixel( pixelIndex, (unsigned char)( 0 ) );
                  }
          }
  }

  WriterType::Pointer writerImg = WriterType::New();
  writerImg->SetFileName("C:\\Users\\Juan Sato\\Desktop\\Resultado.jpg");
  writerImg->SetInput( img );
  writerImg->Update();

  return 0;
}
```