Medical Image Analysis, Ph.D course

Project : Find & track leucocytes

Institute : IMM

Teacher : Milan Sonka

**Authors : Michael Lund, s012499
Lars A. Conrad - Hansen, s022897**

Course period : 20/10/02 – 24/10/02

Introduction :

The project we were assigned to solve was based on five raw-data sequences, which displayed blood vessels and their circumferential tissue. Our task was to identify and track the leucocytes passing through the respective vessels.
In order to complicate things, each image was contaminated with two dark shadow-like blobs under which the leucocytes could hide; on top of that, the vessels shifted their size and shape within the individual sequences, which was especially apparent in sequence three (**figure 1**).

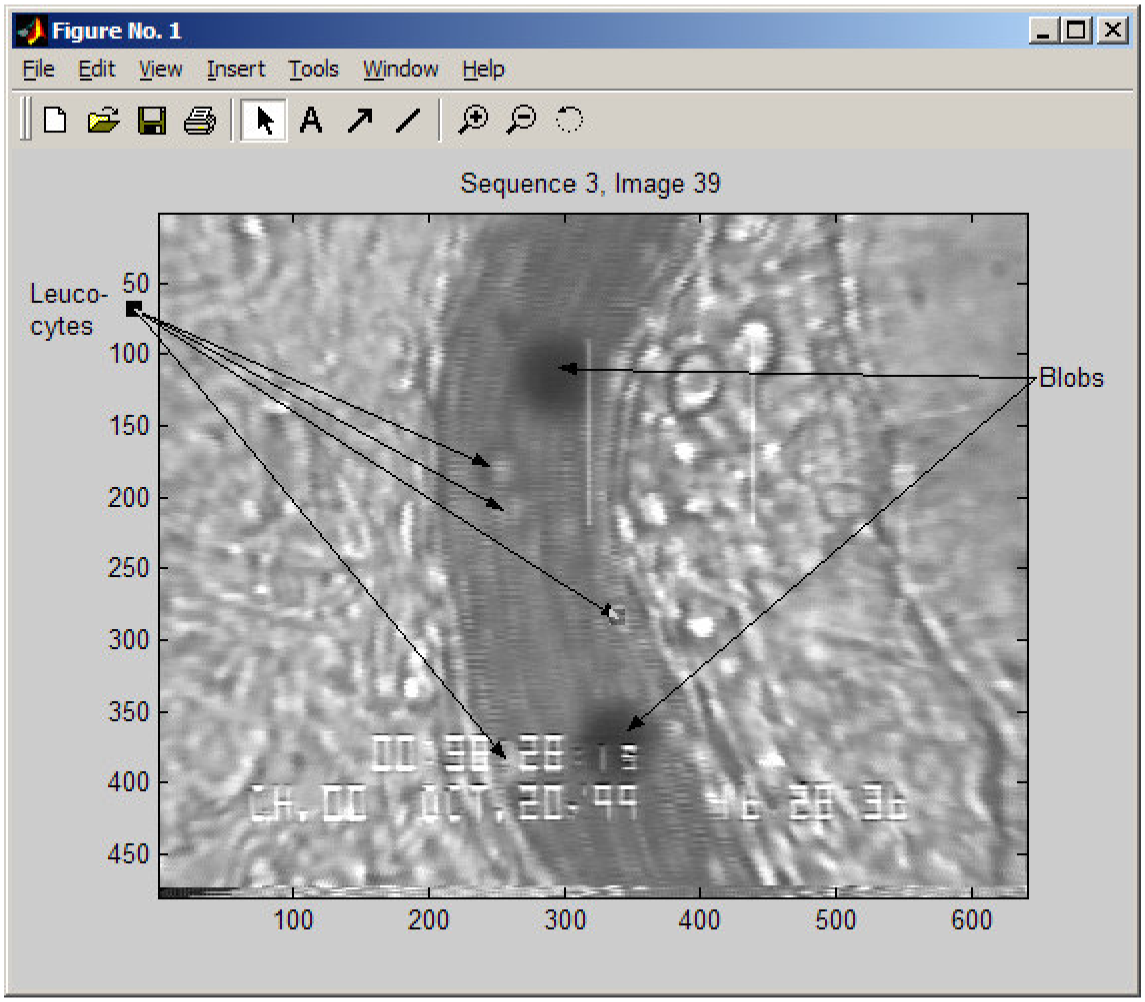


Figure 1 : Sample image

The writing on the bottom of each image didn't make things any easier either, so we didn't have any shortage of obstacles. Besides that, the images were noisy and the gray level of the leucocytes changed from frame to frame.

All relevant code can be inspected in the appendix.

Angle of Attack :

The first objective of the project was to extract the region of interest (the lumen) on which we then afterwards could focus on.
There are many ways to extract the blood vessel in the sequences given, manual, automated, or semiautomated methods. We decided to try to develop an automated method to extract the lumen. When developing an automated method, is it important that the method is robust. We therefore spent quite a lot of time examining different ways to extract the lumen. We looked into gray level threshold, variance threshold, morphological operators, filters and edge detectors. Several of the methods looked interesting, but the single most interesting image analysis tool seemed to be the good old threshold technique.
One of the most apparent features of the lumen is the lower gray level values and more homogenous distribution of pixel intensities. In general the lumen is 'darker' than the area outside the lumen. Especially the 2 dark blobs are significant darker than the surroundings and lay in the lumen or close to it, in all 5 sequences. It could therefore be tempting to these blobs as guides, but to our knowledge these blobs cannot in general be expected to lie next to the lumen. Our preliminary trials had shown us that a rough outline of the lumen could be achieved by thresholding the variance of the image (20*20 neighborhood), but we decided to use the gray level intensities instead.

The grainy multi-structural appearance of the tissue outside the lumen persuaded us to use a morphological filter, which, when we tested it later against other filtering methods, provided the best results, since it also enhanced the edges of the vessel by closing some of the gaps. The filter consisted of an erosion followed by two dilations and finally another erosion.

The thresholding was accomplished using the algorithm for optimal thresholding, which initially takes four 'almost' arbitrary pixel values as background, whereas the rest of the image is regarded as object (these points are only almost arbitrary, since they should be situated in the vicinity of the respective corners of the image in order to insure stable results). At the next step, the means for the background as well as the object gray-level values are computed using

$$\mu^t_B = \frac{\sum_{(i,j)\in background} f(i,j)}{\#background - pixels} \quad ; \qquad \mu^t_O = \frac{\sum_{(i,j)\in objects} f(i,j)}{\#object - pixels}$$

from which the threshold

$$T^{(t+1)} = \frac{\mu^t_B + \mu^t_O}{2}$$

is calculated. This procedure is iteratively repeated until $T^{(t+1)} = T^{(t)}$.

In MATLAB code the filter was constructed using the *imerode* and *imdilate* functions; applying the filter on top of the thresholding procedure, produced the best results

```
imseq = double(im(:,:,num));
imthresh = imseq < thresh;
mask = strel('disk',masksize);
trin1 = imerode(imthresh,mask);
trin2 = imdilate(trin1,mask);
trin3 = imdilate(trin2,mask);
trin4 = imerode(trin3,mask);
```

The following two boxes show the implementation of the thresholding algorithm. We searched for the optimal threshold of each individual image in the respective sequences and discovered that they only marginally differed from each other within each sequence (approx. ± 1 pixel !), so we decided to calculate the threshold average, which then was applied.

```
[r c z] = size(im);
for s = 1:z
   thresh(s) = optimalThresh1(im(:,:,s));
end;

thresh = (sum(thresh))/z
```

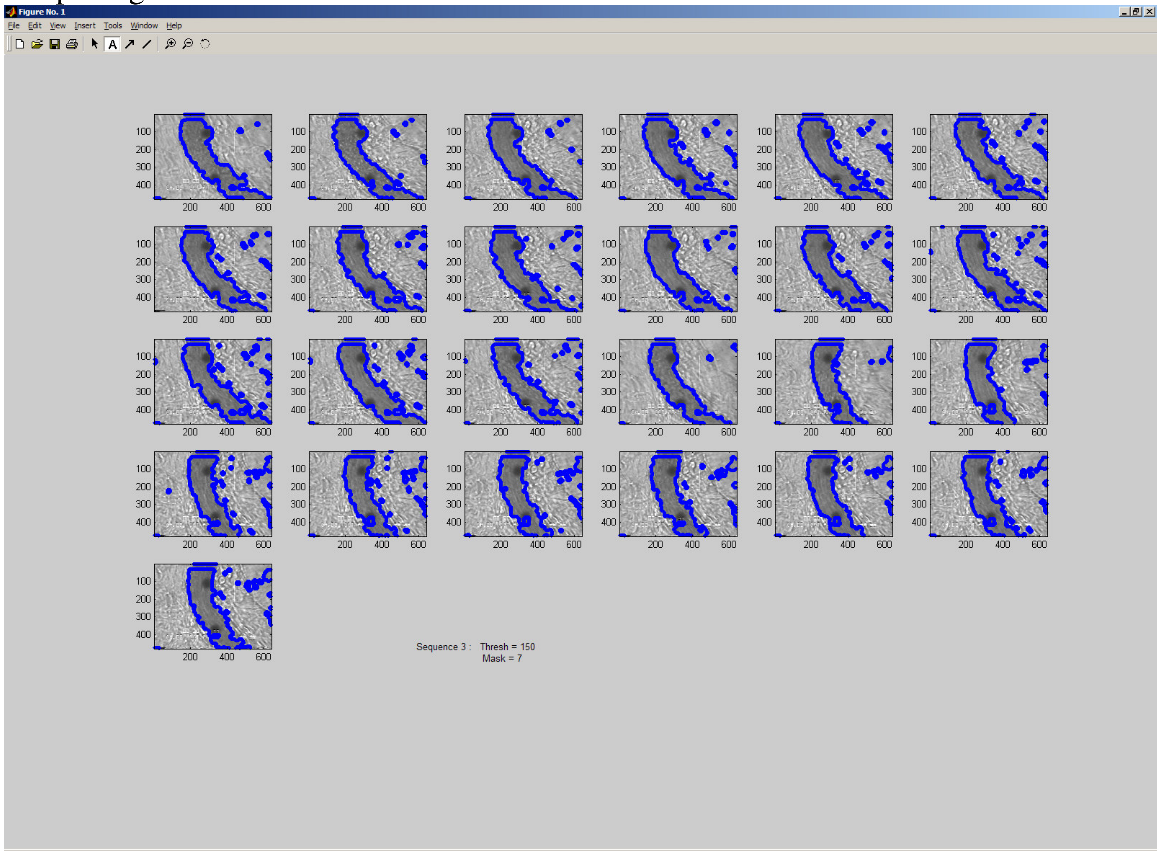
```
function T = optimalThresh1(im)

[r c]=size(im);
col_c=floor(c/50);
rows_c=floor(r/50);
corners=[im(1:rows_c,1:col_c); im(1:rows_c,(end-col_c+1):end);...
        im((end-rows_c+1):end,1:col_c);im((end-rows_c+1):end,(end-col_c+1):end)];
T=mean(mean(corners));
loop = 0;

while loop == 0

  mean_obj=sum(sum( (im > T).*im ))/length(find(im > T));
  mean_backgnd=sum(sum( (im <= T).*im ))/length(find(im <= T));
  new_T=(mean_obj+mean_backgnd)/2;
  if(new_T==T)
    loop = 1;
  else
    T=new_T;
  end;
end;
```

In order to assess the effectiveness of thresholding followed by filtering, we took the gradient magnitude of the resulting data and superimposed the results onto their respective originals.
Figure 2 in the upper half on the next page illustrates this. We didn't quite manage to remove all of the noise yet and there are indentations on some of the plots, due to the morphological removal of the text.



Sequence 3 : Thresh = 150
Mask = 7

Figure 2 : Superimposed gradient of the thresholded and filtered images

The following table displays the values for thresholds and mask sizes that produced the best results when applied on the respective sequences :

Sequence	Threshold	Morphological mask
1	190	9
2	140	7
3	150	7
5	90	11
6	90	15

The various filter sizes were assessed visually using the function *morphfilter()*, which is included in the appendix.

Our next attempt was to isolate the region of interest from the remaining noise. The initial idea to solve this problem was to employ a graph-searching algorithm, but all the algorithms we devised were dependent on some sort of initial value to start it. We tried the different implementations out on the easiest sequence (sequence 1) , and found the methods too cumbersome to work with, due to the necessity of initializing each independent graph. **Figure 3** displays the resulting regions of interest, where it is apparent that the results are not exactly stable (especially if we keep in mind that the easiest sequence was used).

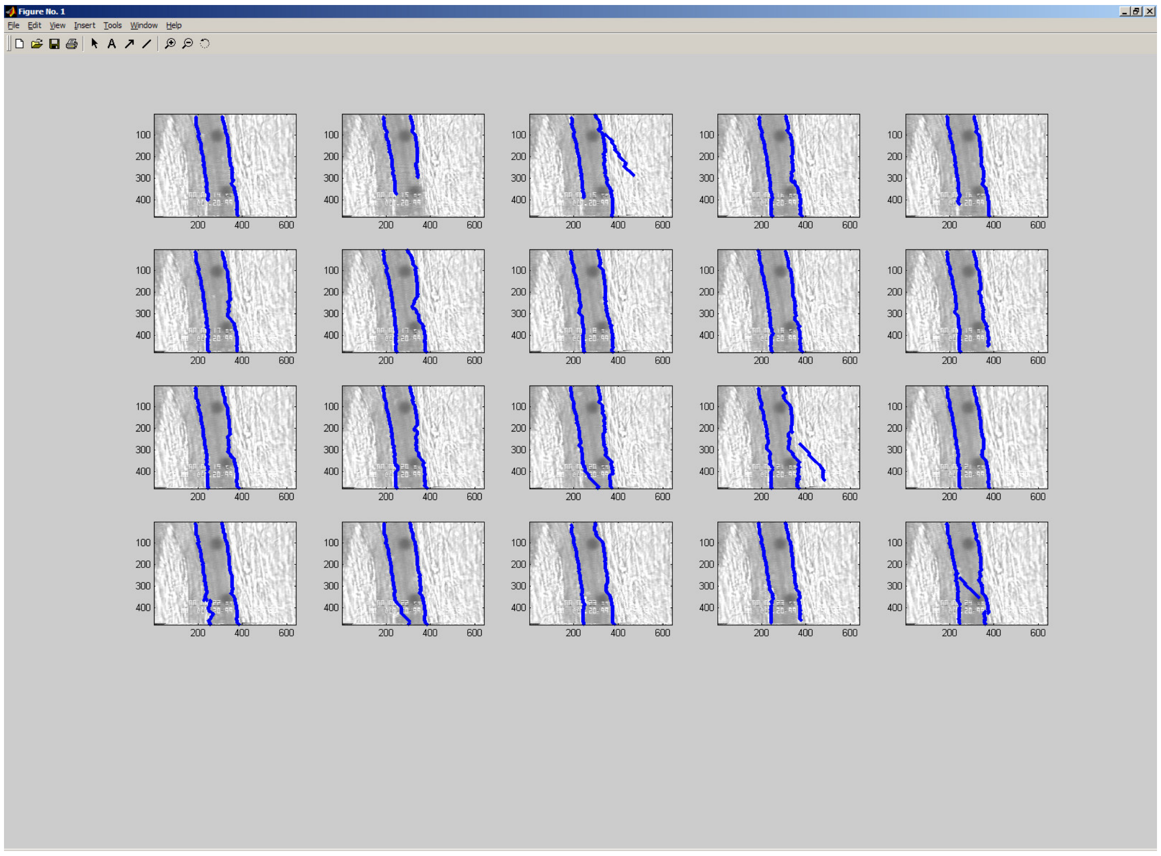


Figure 3 : Extracting the region of interest using graph searching on sequence 1

Some of the graphs were simply cut off or bent inwards where the writing in the original images was located, and on three occasions the graphs strayed off where they were pulled into different directions, due to the underlying noise.

We then thought of a much more efficient method by classifying everything that resulted from the thresholding and filtering as regions, from which then the largest region was extracted, knowing that the largest region corresponded to our region of interest. This procedure was also applied on each individual image of the respective sequences, using the function *bwlabel()*.

```
[L,t]=bwlabel(trin4);
   [lx1,ly1] = find(L==1);
   maxreg = size(lx1);
   counter = 0;
   for reg = 2:t
      [lx1,ly1] = find(L==reg);
      [a,b] = size(lx1);
      if maxreg(:,1) < a;
         maxreg = size(lx1);
         counter = reg;
      end;
   end;
   if counter == 0
      counter = 1;
   end;
   [lx1,ly1] = find(L==counter);
```

The advantages of this method were that no initial values were needed to start of the search, thus improving the speed, and that only the truly connected components resulted as output, ridding us of all the stray values.

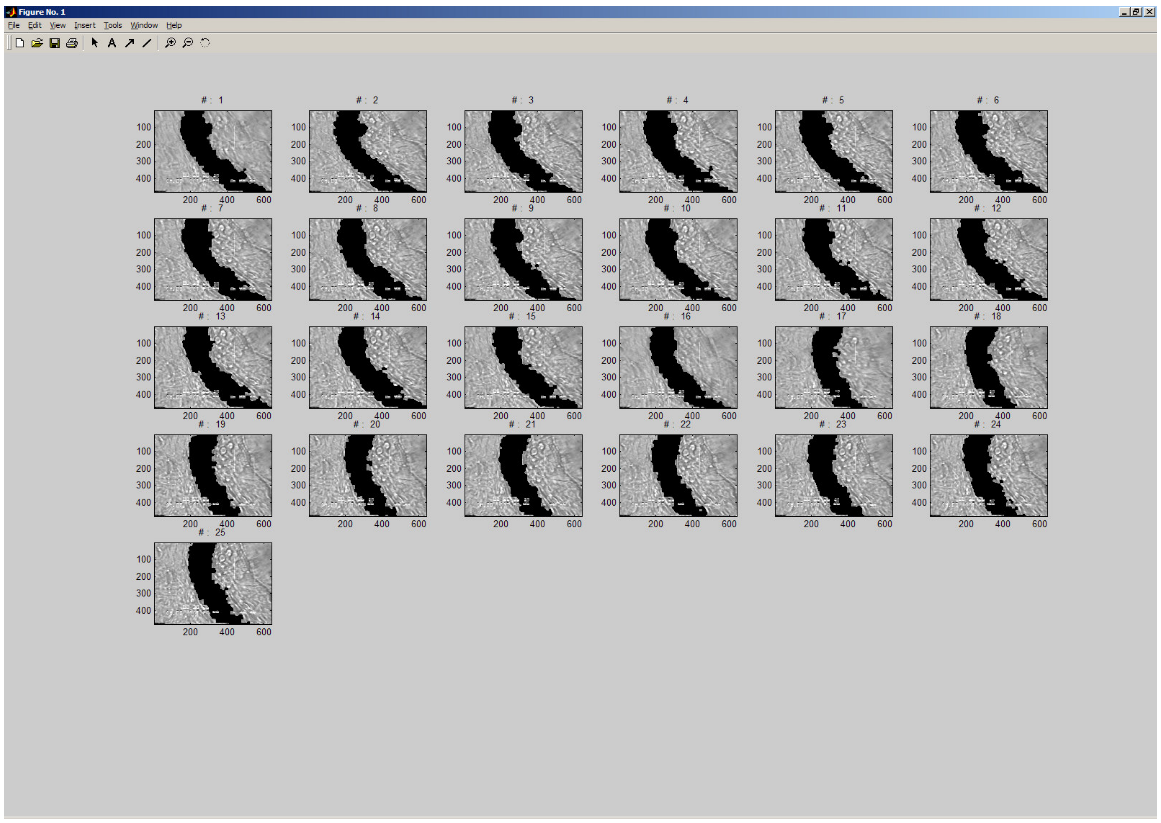


Figure 4 : Superimposed extracted regions of sequence 3

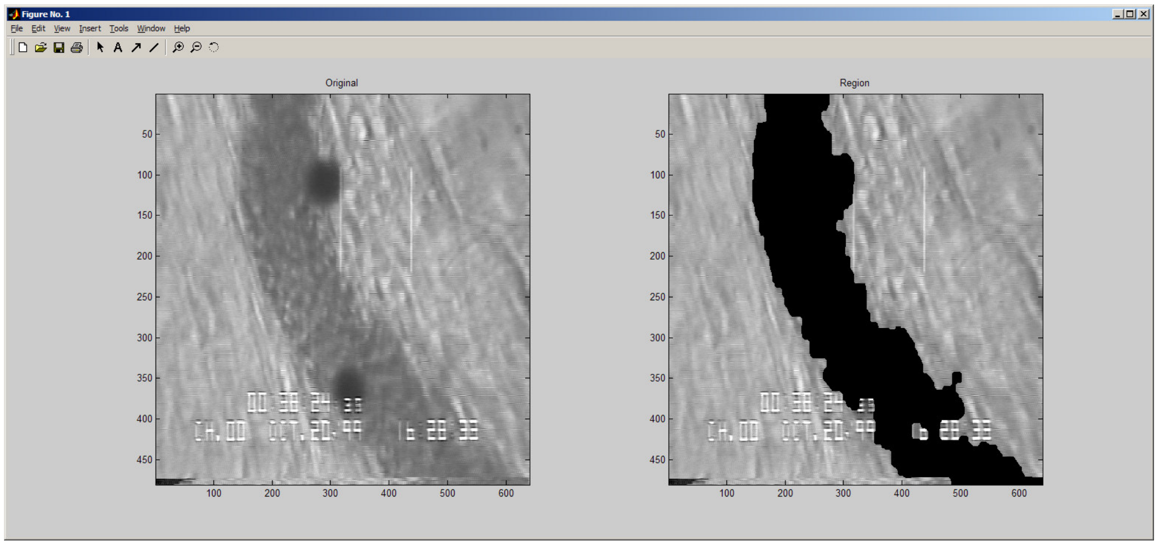


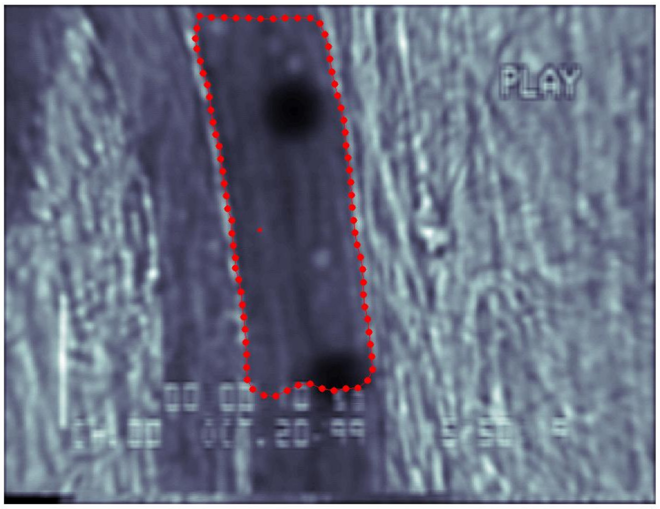
Figure 5 : First image of sequence 3; left the original, right the extracted region

Figure 4 shows the results of our region extraction on the first 25 images of sequence 3 superimposed on their original images and **figure 5** focuses on an individual image and its region of interest.
The extracted regions still had small holes where the text used to be, and the shadow-like blobs are not filtered out, but since the leucocytes that we eventually wanted to track, couldn't really be seen underneath those structures anyway, we decided that the results were good enough and could be used in the next phase of the project.

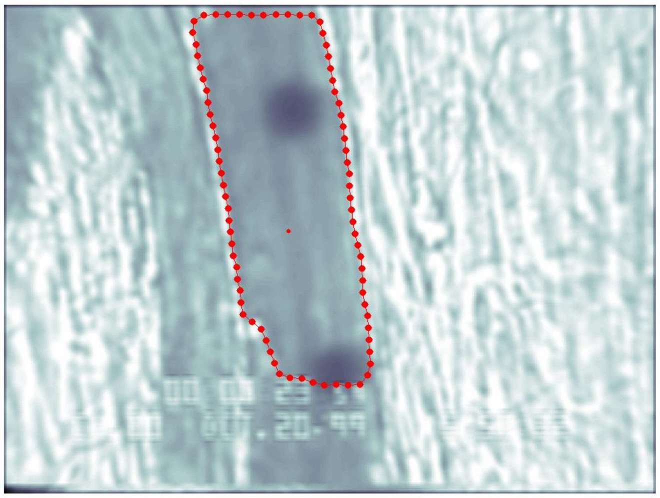
The next objective was to locate the leucocytes. The leucocytes have the property that they have pixel values close to the threshold value. When applying the threshold it came soon apparent that we had a problem with our lumen extraction. With our used method the border bumps inward when leucocytes lie next to the border. Our method was to imprecise. We decided therefore to improve our method. We decided to use a snake to improve our border finding. The snake should be started in the center of the previous extracted region and expand until it reach the border. An ordinary snake with balloon force and gradient image energy was therefore used.

The new result was a lot better. Visual inspection showed results close to the true border, there were however some problems, most of then minor. One of the problems were the digits in the bottom of the picture, the snake couldn't go past the digits. It was therefore decided to exclude the lower part of the image. In a few of the images there were also problems with the corners of the snake. There only major problem were with sequence 3, the snake didn't behave good enough to be used in a automated system. In some regions did the border seem to be to weak.

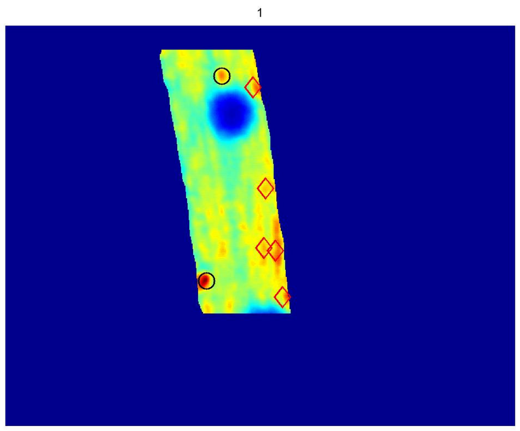
The next step in the process was to filter out the region selected by the snake. This was easily done.

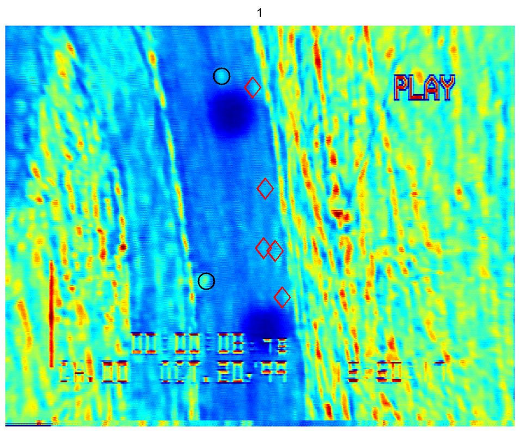


The “only” remaining item is to locate the leucocytes. Again it was decided to found the procedure on thresholding with the previous calculated values.
However this information is not sufficient, some of the images are contaminated with long “bright” stripes that to our knowledge has nothing to do with leucocytes. The segmented regions were separated into leucocytes or garbage based on size. Objects with a large or small (in x or y direction) size were classified as garbage. Objects with small area were likewise considered as garbage.
The last measure is based on the gradient of the objects. Objects not yet classified as garbage and having large gradients are rewarded (classified as leucocytes).

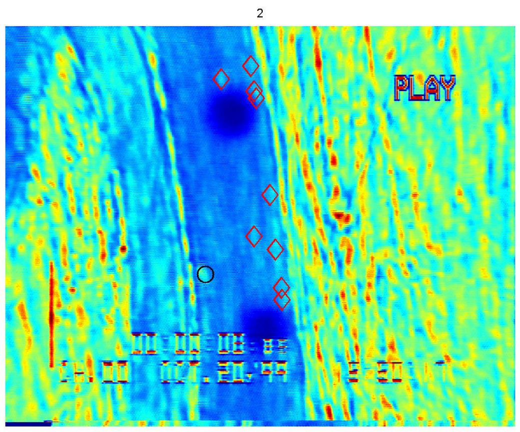


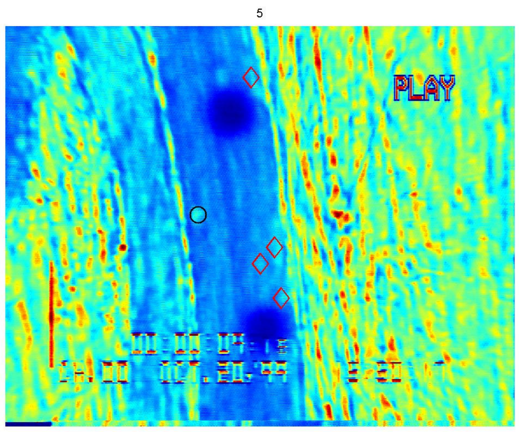
Below are seen some images showing some of our results. The first 2 images are from the same frame. The circles are identified leucocytes and the diamonds are rejected objects.

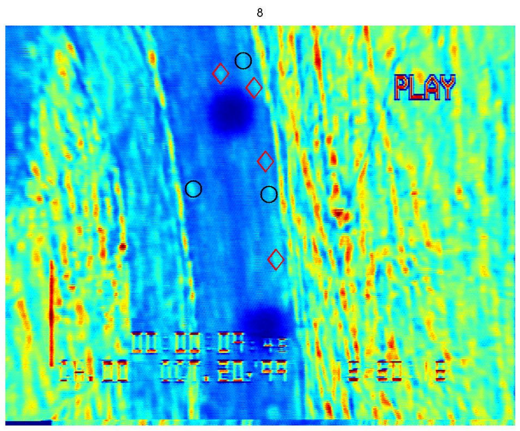


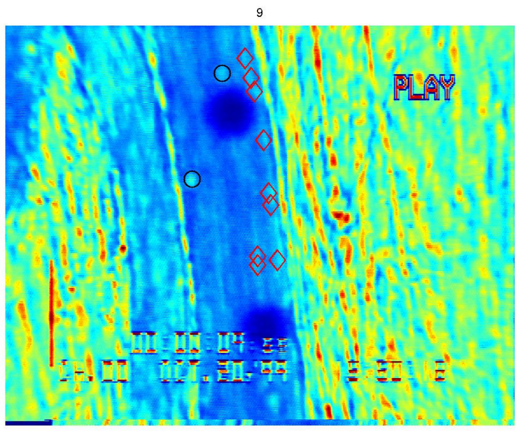


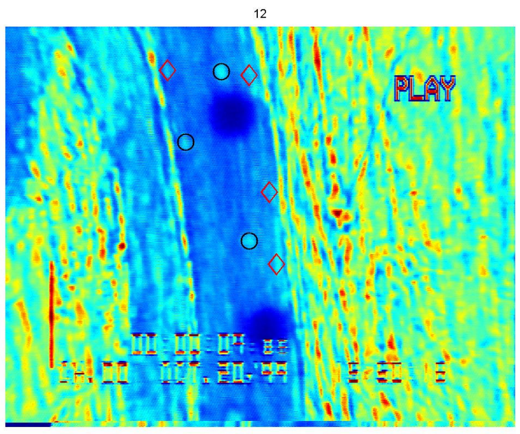
The first of the two images are filtered with an average filter. Not only gave this filter better result, but it made it also easier for us to visual track the leucocytes. The above result seems okay; the "objects" in the left part of the image is to our knowledge not leucocytes. The following results are not as good.

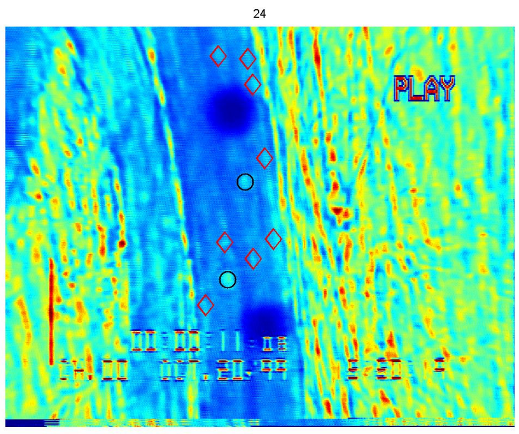




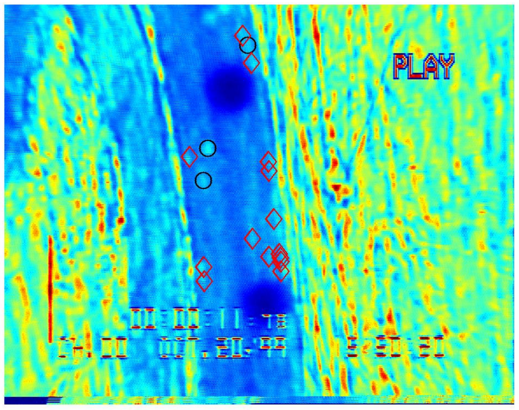




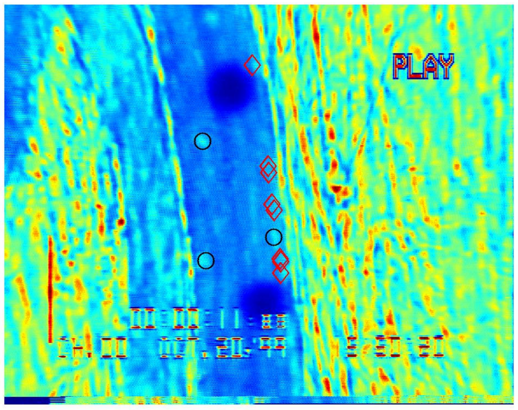




31



32



It is apparent from the above results that our method is not good enough. Especially the object just above the upper dark spot, has a problem, it sort of flicks on and off. Actually are we not ourselves sure whether the object is a leucocyte or not.

Conclusion

When we started our project we expected that it shouldn't take to long time, but we must we have use more time than expected and there is still room for improvements. One of the major problems with the sequences is the many artifacts and poor playback quality. It should however still be possible to achieve much better results.
The procedure for localization of the lumen border can still be improved and should properly be based on the image gradient (like the snake) either in the form of a modified snake, level sets or similar methods.

However, the major improvement required is with the leucocyte classification. By using the gradient information more active and by analyzing the shape of the objects and their gradients it should be possible to improve the classification. If the classification method also considers the 3D information, a pretty robust method should derive.

The best result could properly be achieved with a quite different method; a pixel classification approach were maybe the best solution.

APPENDIX

MATLAB Functions

```
function morphfilter()

close all;
load Test1;
im = Test1;     num = 1;
imseq = double(im(:,:,num));
thresh = 190;
imthresh = imseq < thresh;
endloop = 17;   %    i.e. the max size of the mask
figure
colormap(gray),imagesc(imseq),title('Original Image');
for t = 3:2:endloop;
   figure,
   colormap(gray),
   mask = strel('disk',t);
   trin1 = imerode(imthresh,mask);
   subplot(2,2,1),imagesc(trin1),title('A-B');
   trin2 = imdilate(trin1,mask);
   subplot(2,2,2),imagesc(trin2),title('(A-B)+B');
   trin3 = imdilate(trin2,mask);
   subplot(2,2,3),imagesc(trin3),title('((A-B)+B)+B');
   trin4 = imerode(trin3,mask);
   subplot(2,2,4),imagesc(trin4),title('(((A-B)+B)+B)-B');
end;
```

```
function RegionFinder2(filname)

close all;
if nargin == 0
   filname = 'Test1';
end;

if filname == 'Test1'
   load Test1;
   im = Test1;
   thresh = 190;
   masksize = 9;
elseif filname == 'Test2'
   load Test2;
   im = Test2;
   thresh = 140;
   masksize = 7;
elseif filname == 'Test3'
   load Test3;
   im = Test3;
   thresh = 152;
   masksize = 7;
elseif filname == 'Test4'
   load Test4;
   im = Test4;
   thresh = 90;
   masksize = 11;
else
   load Test5;
   im = Test5;
   thresh = 90;
   masksize = 15;
end;

[r c z] = size(im);
for s = 1:z
   thresh(s) = optimalThresh1(im(:,:,s));
end;

thresh = (sum(thresh))/z



grad = 1;
%autosearch(im,grad,thresh,masksize);
%displaySeq(im);
image_holder = regfind(im,thresh,masksize);
```

```
function dummy = regfind(im,thresh,masksize)

[x y z] = size(im);
dummy = zeros(x,y,z);
d = 39;
figure
zsqrt = round(sqrt(z))+1;
for num = 1:z

   %**********     morphfilter     *****************************

   imseq = double(im(:,:,num));
   imthresh = imseq < thresh;
   mask = strel('disk',masksize);
   trin1 = imerode(imthresh,mask);
   trin2 = imdilate(trin1,mask);
   trin3 = imdilate(trin2,mask);
   trin4 = imerode(trin3,mask);

   %*************     extracting the largest region     *********

   [L,t]=bwlabel(trin4);
   [lx1,ly1] = find(L==1);
   maxreg = size(lx1);
   counter = 0;
   for reg = 2:t
      [lx1,ly1] = find(L==reg);
      [a,b] = size(lx1);
      if maxreg(:,1) < a;
         maxreg = size(lx1);
         counter = reg;
      end;
   end;
   if counter == 0
      counter = 1;
   end;
   [lx1,ly1] = find(L==counter);
   size([lx1,ly1]);
   for k = 1:size(lx1)
      dummy(lx1(k),ly1(k),num) = im(lx1(k),ly1(k),num);
   end;
   colormap(gray),subplot(zsqrt,zsqrt,num),imagesc(dummy(:,:,num)),title(sprintf('Region #: %i',num));

end;

return;
```

```
function T = optimalThresh1(im)

[r c]=size(im);
col_c=floor(c/50);
rows_c=floor(r/50);
corners=[im(1:rows_c,1:col_c); im(1:rows_c,(end-col_c+1):end);...
        im((end-rows_c+1):end,1:col_c);im((end-rows_c+1):end,(end-col_c+1):end)];
T=mean(mean(corners));
loop = 0;

while loop == 0
  mean_obj=sum(sum( (im > T).*im ))/length(find(im > T));
  mean_backgnd=sum(sum( (im <= T).*im ))/length(find(im <= T));
  new_T=(mean_obj+mean_backgnd)/2;
  if(new_T==T)
    loop = 1;
  else
    T=new_T;
  end;
end;

return;
```

```
function displaySeq(im)

close all;

[x y z] = size(im);
if z < 30
   fin = z;
   elseif z > 30 & z < 60
      fin = z/2;
   else
      fin = z/4;
end;
fin = 30;
count = 1;
zsqrt = round(sqrt(fin))+1;
figure
for num = 30:60
   subplot(zsqrt,zsqrt,count),colormap(gray),imagesc(im(:,:,num)),title(sprintf('Im #: %i',num));
   count = count + 1;
end;

return;
```

```
function autosearch(im,gradient,thresh,masksize)

drawgradient = gradient;
[x y z] = size(im);
dummy = zeros(x,y,2);
if z < 30
   fin = z;
   elseif z > 30 & z < 60
      fin = z/2;
   else
      fin = z/4;
end;
zsqrt = round(sqrt(fin))+1;
figure
for num = 1:fin

   %*********     morphfilter     *****************************

   imseq = double(im(:,:,num));
   imthresh = imseq < thresh;
   mask = strel('disk',masksize);
   trin1 = imerode(imthresh,mask);
   trin2 = imdilate(trin1,mask);
   trin3 = imdilate(trin2,mask);
   trin4 = imerode(trin3,mask);

   %****************     Finding the different regions     ***

   [L,t]=bwlabel(trin4);

   %************     extracting the largest region     *********

   [lx1,ly1] = find(L==1);
   maxreg = size(lx1);
   counter = 0;
   for reg = 2:t
      [lx1,ly1] = find(L==reg);
      [a,b] = size(lx1);
      if maxreg(:,1) < a;
         maxreg = size(lx1);
         counter = reg;
      end;
   end;
   if counter == 0
      counter = 1;
   end;
   [lx1,ly1] = find(L==counter);
   size([lx1,ly1]);
   dummy(:,:,2) = imseq;
   for k = 1:size(lx1)
     dummy(lx1(k),ly1(k),2) = L(lx1(k),ly1(k));
   end;
   subplot(zsqrt,zsqrt,num),colormap(gray),imagesc(dummy(:,:,2));
   if drawgradient == 0
      [dx dy] = gradient(double(trin4));
      gradmag = sqrt(dx.^2 +dy.^2);
      mat1 = gradmag;
      [x1 y1] = find(mat1);
      hold on;
      subplot(zsqrt,zsqrt,num),plot(y1,x1,'b.');
      hold off;
   end;

end;
return;
```

```
function points = makeSnakeMask (im,tres)

numPoints = 80; radius = 15;
filtersize = 15; sigma = 3;

alfa = 0.1; beta = 0.2; imW = 0.6; gamma = 0.1; balloon = 1.2; numIter = 2500;

[a b z] = size(im);

points = zeros(numPoints,2,z);

for imagenr = 1:z
    imagenr

    avimage = convolve2(im(:,:,imagenr), fspecial('Average',7) ,'same');
    center = calcCenter(avimage,tres,11);

    hpimage = highpad(im(:,:,imagenr));

    SnakePoints = CircleSnake(center,numPoints,radius);

    points(:,:,imagenr) = My_IM_Snake(hpimage,sigma,filtersize,SnakePoints,numIter, alfa, beta, gamma, imW,
                          balloon,100);

end


function im = highpad(im)

im(1:10,:)= 255;
im(end-10:end,:) = 255;
im(:,1:10) = 255;
im(:,end-10:end) = 255;
```

```
function newPoint = My_IM_Snake (bwimage,sigma,filtersize,SnakePoints, numIter, alfa, beta, gamma, imW, balloon, restep)

% Get gauss filterede image and gradient
[gaussImage, imageGrad, dfx, dfy] = gaussFilter(bwimage,sigma,filtersize);


newPoint = SnakePoints;

% The recalculation of the snake
for i = 1:numIter

    [newPoint c a b im ba] = IterateOnce(dfx, dfy, newPoint, gamma, alfa, beta, imW, balloon );
    if (mod(i,restep)==0)
        newPoint = reparameterize (newPoint);
    end
end

end

% Iteration function
% --------------------------------------------------------------
function [newPoint, change, alfaTerm, betaTerm, imageTerm, balloonTerm]  = IterateOnce(dfx, dfy, Old, timestep, alfa, beta, imW, balloon )


n = size(Old,1);

temp = [Old(end-1,1) Old(end,1) Old(:,1)' Old(1,1) Old(2,1);
        Old(end-1,2) Old(end,2) Old(:,2)' Old(1,2) Old(2,2)]';

imageTerm = zeros(n,2);
alfaTerm = alfa*(temp(2:end-3,:)+temp(4:end-1,:)-2*temp(3:end-2,:));
betaTerm = beta*(-temp(1:end-4,:)-temp(5:end,:)+4*temp(2:end-3,:)+4*temp(4:end-1,:)-6*temp(3:end-2,:));

for i = 1:n
    x = round(Old(i,1));
    y = round(Old(i,2));
    imageTerm(i,:) = [dfx(y,x),dfy(y,x)];
end

imageTerm = imW * imageTerm;

Nxy = ones(n,1)*mean(Old)-Old;
Nxy = Nxy./(sqrt(sum((Nxy.*Nxy),2))*[1 1]);
balloonTerm = balloon*Nxy;

change = (timestep*(alfaTerm+betaTerm+imageTerm-balloonTerm));
newPoint = max(Old+change,10);

end

% Gaussian filtering
function [gauss, imageGrad, dfx, dfy] = gaussFilter(image,sigma,filtersize)

gauss = conv2(image, fspecial('Gaussian',filtersize,sigma),'same');
[dx dy] = gradient(gauss);
imageGrad = dx.*dx +dy.*dy;
[dfx dfy] = gradient (imageGrad);

end

% reorganize points
function XY = reparameterize (XY)

N = size(XY,1);
XYp1 = XY([2:end 1],:);

dXY = sqrt(sum((XY-XYp1).*(XY-XYp1), 2));
L = sum(dXY);
arc = [0 cumsum(dXY)'];
arcNew = (L/N) * (0:(N-1)) ;
xxxNew = interp1(arc, [XY(:,1)' XY(1,1)], arcNew,'linear');
yyyNew = interp1(arc, [XY(:,2)' XY(1,2)], arcNew,'linear');
XY = [xxxNew' yyyNew'];
end
```

```
function LeucoTrack2(maskim,orgIm,tres)

[a b z] = size(maskim);

figure(1); movegui('northwest');colormap('bone')
figure(2); movegui('north');colormap('bone')

for i = 1:z

    % opsætning
    fcIm = convolve2(double(maskim(:,:,i)), fspecial('Average',5) ,'same');
    fcIm = imerode(fcIm>0,strel('disk',7)).*fcIm;
    corgIm = double(orgIm(:,:,i));

    % baggrund
    tIm = fcIm > 0; tIm2 = (1 - tIm);
    mask = imerode(tIm,strel('disk',7));
    grad = convolve2(my_grad(corgIm).*mask, fspecial('Average',5) ,'same');

    % threshold til de 2 pletter
    tres2 = (min(min(fcIm+tIm2*256))+tres)/2;
    mask3 = imerode(fcIm > tres2,strel('disk',5));

    % isolering af leucocytes
    tIm3 = (fcIm>tres);
    ctIm2 = imdilate(tIm3,strel('disk',3));

    % masker baggrund + pletter ud
    mctIm2 = ctIm2.*mask3.*mask;
    ixIm = (mask.*ctIm2);

    [lim n] = bwlabel(ixIm);

    for w = 1:n
        lw = (lim==w);
        [x y] = find(lw);

        difx=max(max(x))-min(min(x));
        dify=max(max(y))-min(min(y));
        sx = size(x,1);
        if (difx<30 & difx>4 & dify<30 & dify>4 & sx > 100 & sum(sum(lw.*grad))*size(x,1)>10000)
            center = [center;mean(x) mean(y)];
        else
            reject = [reject;mean(x) mean(y)];
        end
    end


    figure(1), imagesc(fcIm);title(i);axis off % masked image
    figure(2), imagesc(corgIm);title(i);axis off; % original image

    if n > 0;

        if size(center,1)>0
            figure(1);
            hold on; plot(center(:,2),center(:,1),'wo','LineWidth',1.5,'MarkerSize',15);hold off;
            figure(2);
            hold on; plot(center(:,2),center(:,1),'wo','LineWidth',1.5,'MarkerSize',15);hold off;
        end
        if size(reject,1)>0
            figure(1);
            hold on;plot(reject(:,2),reject(:,1),'rd','LineWidth',1.5,'MarkerSize',15);hold off;
            figure(2);
            hold on;plot(reject(:,2),reject(:,1),'rd','LineWidth',1.5,'MarkerSize',15);hold off;
        end
    end

    drawnow

    pause

end


function res = my_grad(im)

[dx1 dy1] = gradient(im);
res = sqrt(dx1.*dx1 +dy1.*dy1);
```