CSE 5524 - Homework #3
9/12/2016
Rakshith Kunchum

1. Generate a 4-level Gaussian pyramid (original image is level-1) and the corresponding Laplacian pyramid of an image (select one from the web).Use the formula in the notes to first determine a viable image size, and crop the image (if needed) to test the pyramid code. Use a=0.4 for the Gaussian mask – use separable masks! Write/use functions for properly reducing and expanding an image.

Original Image used

Burt and Adelson Formula was used to crop the image to a viable size.
The size of the image considered was 513x513.

Gaussian Pyramid

For generating gaussian pyramids, there are two stages : applying gaussian filter(smoothing) and reducing the size by half along each dimension(subsampling).

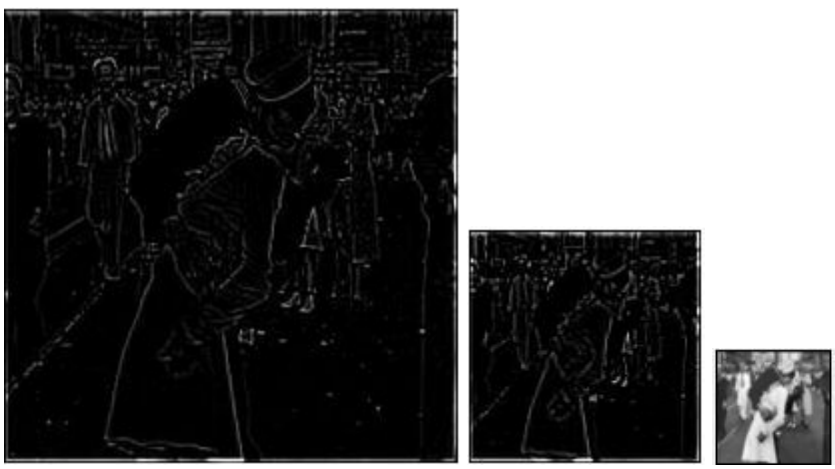
The gaussian filter used was [0.05 0.25 0.4 0.25 0.05]. (for a = 0.4)

Using the property of separability, the gaussian filter was applied separately along the different dimensions. This reduces process time and is easy to implement as well. For subsampling, alternate rows and columns of the image were filtered out and discarded.

Immediately after obtaining the gaussian image, it was expanded and interpolated, in order to obtain the approximate image at a lower level. The difference of the gaussian and the approximate image gives us the Laplacian image.

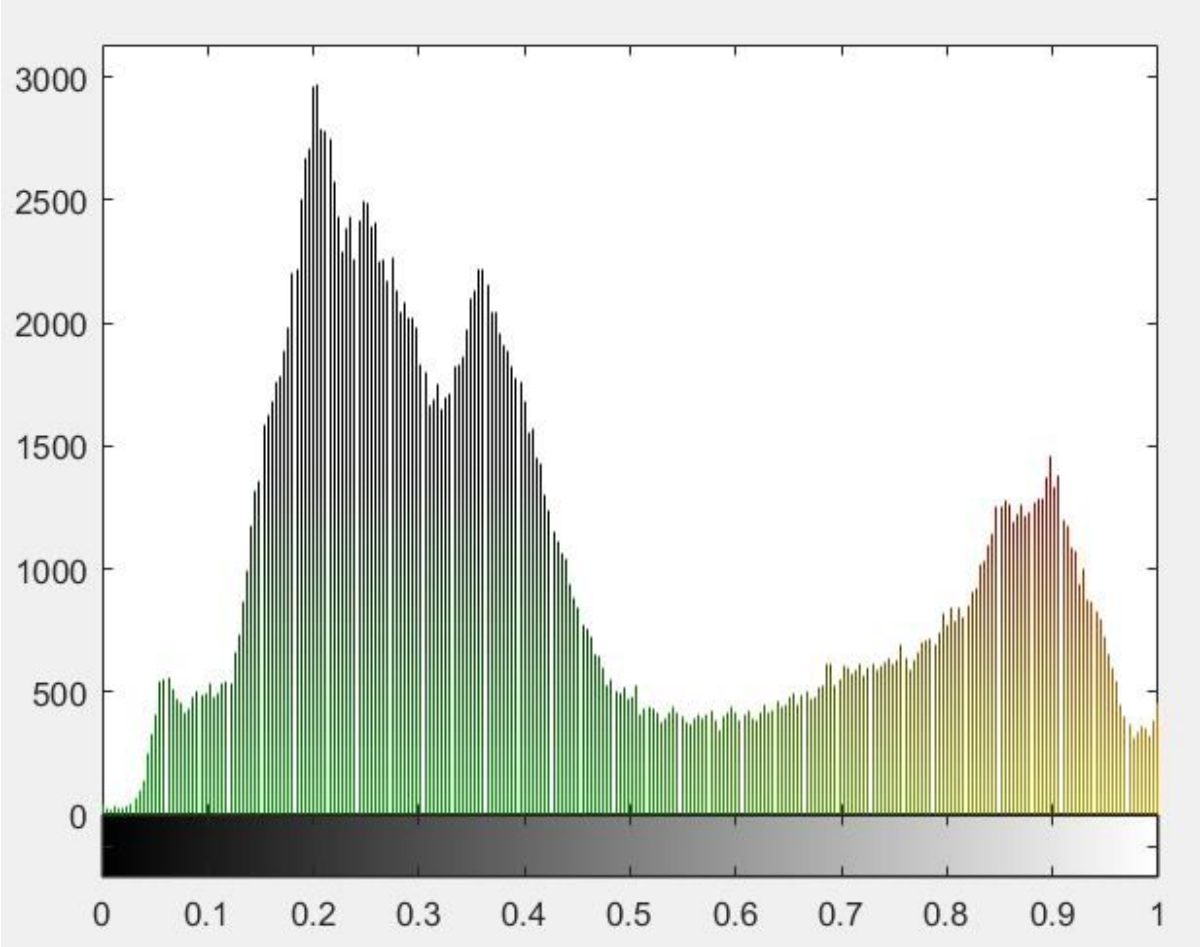
While implementing the gaussian smoothing filter, the pixels on the border of the image was neglected. Because of this we can observe a distinct border appearing in the Gaussian images as we go up the pyramid.

Laplacian Pyramid



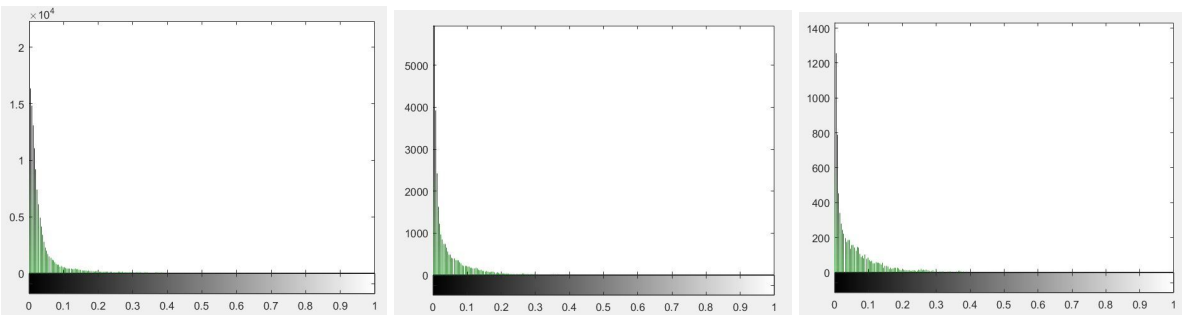
The Laplacian images are the edge detected images except for the last level. This is because Laplacian image is nothing but difference between an image and an interpolated version of the reduced-gaussian-smoothed image.
This is done in order to perform lossless compress the image. (Even lossy compression can also be done.)

Following is the histogram of the original image :



Since pixels are distributed everywhere, lossless compression of the image is difficult.

Following is the histogram of L1, L2, L3



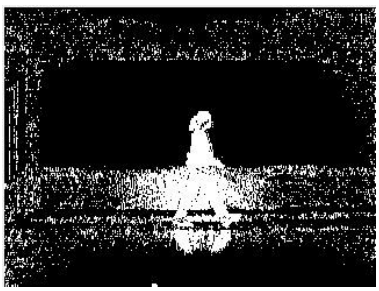
Since the distribution is skewed we can employ some sort of Huffman coding to store the image in a compressed fashion.

2. Using the grayscale images (walk.bmp, bg000.bmp) provided on the WWW site, perform background subtraction 1 using simple image subtraction to identify the object. Experiment with thresholds.

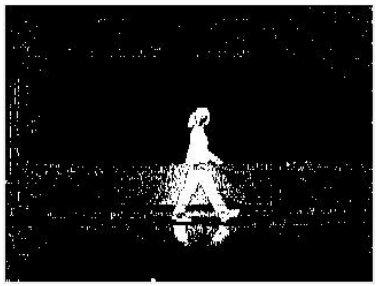
The walker image was extracted using simple subtraction.

walkerIm = abs((bgIm(:,:,1) - walkIm)) > Threshold;

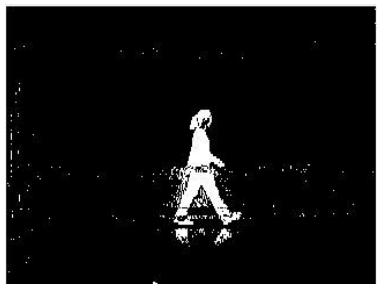
Threshold = 10



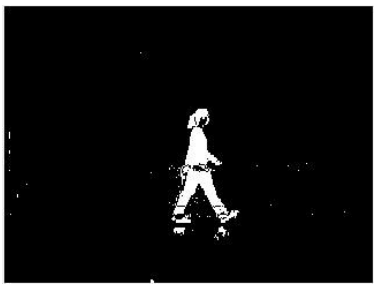
Threshold = 20



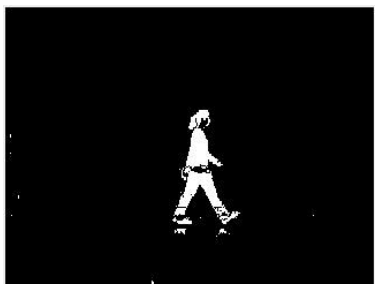
Threshold = 30



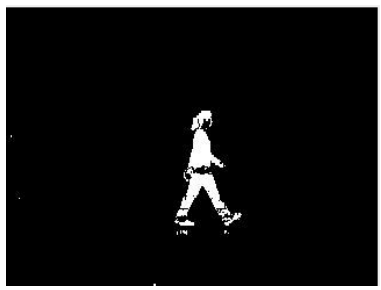
Threshold = 30



Threshold = 40



Threshold = 60



The original grayscale image was converted into a binary image with only the foreground selected. As we increase the threshold the noise in the foreground image reduces. The threshold is sort of a tuning parameter. For initial values of threshold, we can observe that the foreground is ridden with salt and pepper noise.

At Threshold =60, the image of the walker is very distinct.
But if we increase the threshold to a very higher value, we might be missing a significant part of the original foreground. This fact is evident from the below image extracted for Threshold value of 150.

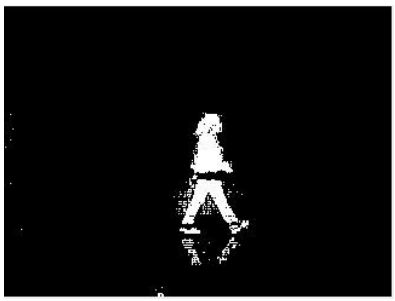
For very high Threshold of 150

3. Using the grayscale images (walk.bmp, bg[000-029].bmp) provided on the WWW site, perform background subtraction 2 using statistical distances. Experiment with Thresholds.

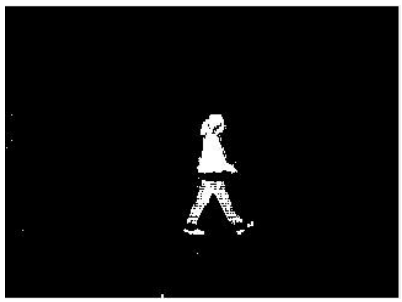
Following formula was used:

walkerIm2 = abs((walkIm - bgImAvg).^2./bgImStd.^2) > Threshold^2;

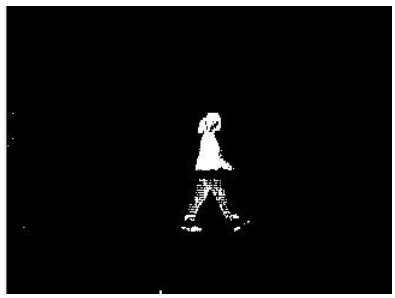
Threshold = 10



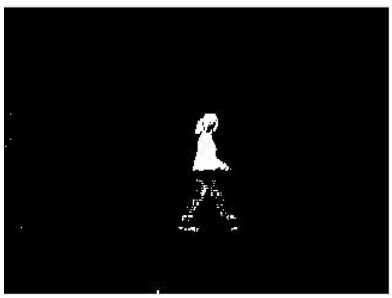
Threshold = 20



Threshold = 30



Threshold = 30



Threshold = 40

Threshold = 60

As compared to simple image subtraction, the statistical subtraction has the advantage that it removes the salt and pepper noise. It can adapt well even if there is minor changes in the pixels belonging to the background. In this way it is robust.

But just like simple subtraction, statistical subtraction also eliminates significant part of the foreground image if the threshold level is increased to a very high value. This fact is demonstrated below.

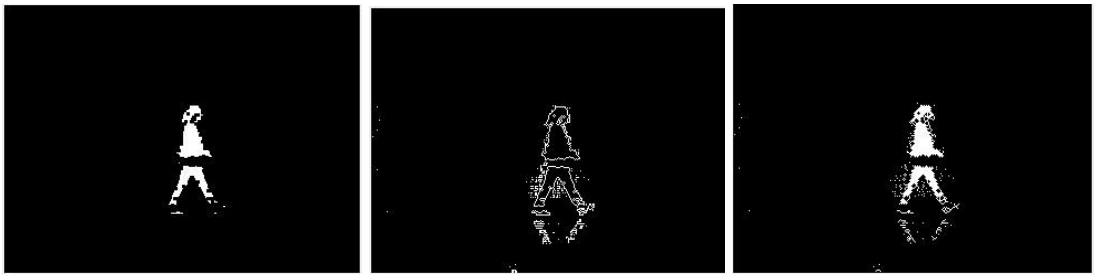
For very high Threshold of 150

4. Dilate the best binary image resulting from problem 3.

Best image was seen for Threshold = 20. Dilate operation was performed on it.

Dilate operation tries to smoothen the edges of the foreground image. It also connects the adjacent components in the foreground image. It increases the span of the foreground image.

Other operations supported by Matlab was also explored.

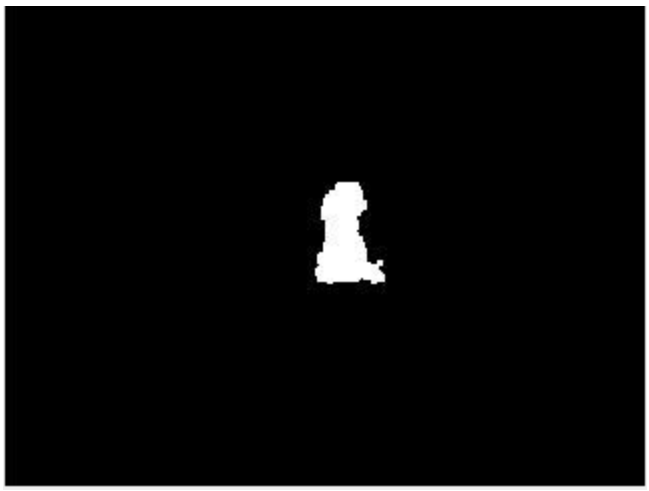


Erode Remove Shrink

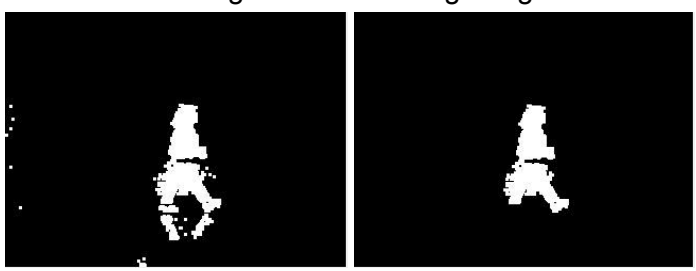
'Remove' seems to identify the outline of the foreground image. Different operations can be applied in series in order to obtain a 'good' noiseless foreground image. Of-course 'good' here is subjective.

5. Next perform a connected components algorithm, and keep only the largest region in L (save/display as an image).

With the minimum threshold as 8, totally 10 regions were identified.Out of them the single largest region has been printed below.



The same experiment mentioned above was performed for Threshold value of 10. It was observed that It gave a better single region for the walker.



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rakshith Kunchum
% HW3
% 9/12/2016


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 1
I = cell(1,4);
L = cell(1,4);
I{1} = getViableSize(double(imread('sailor.jpg')));
figure, imshow(I{1}/255);
pause;
for i = 2:4
   I{i} = reduceIm(I,i-1);
   L{i-1} = I{i-1} - expandIm(I,i);
   figure, imshow(I{i-1}/255);
   figure, imshow(L{i-1}/255);
   pause;
end
L{4} = I{4};
figure, imshow(I{4}/255);
figure, imshow(L{4}/255);
pause;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 2
walkIm = double(imread('images/walk.bmp'));
bgIm = zeros(size(walkIm,1),size(walkIm,2),30);
for i=0:29
   filename = sprintf('images/bg%03d.bmp', i);
   bgIm(:,:,i+1) = double(imread(filename));
end
for Threshold = 10:10:60
   walkerIm = abs((bgIm(:,:,1) - walkIm)) > Threshold;
   figure, imshow(walkerIm)
end
walkerIm = abs((bgIm(:,:,1) - walkIm)) > 150;
figure, imshow(walkerIm)
pause;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 3
bgImAvg = mean(bgIm,3);
bgImStd = std(bgIm,[],3);
for Threshold = 60:-10:10
   walkerIm2 = abs((walkIm - bgImAvg).^2./bgImStd.^2) > Threshold^2;
   imshow(walkerIm2)
   pause;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 4
walkerIm2_dl = bwmorph(walkerIm2, 'dilate');
imshow(walkerIm2_dl)
pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 5
[walkerLabels, num] = bwlabel(walkerIm2_dl, 8);
figure, imshow(returnLargestLabelIm(walkerLabels, num));
figure, imshow(walkerIm2_dl);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Functions used
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ Im ] = getViableSize( Im )
%GETVIABLESIZE returns image of right size.
%   Burt and Adelson Formula
   Im = Im(1:2:1024,1:2:1024,1);
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ temp ] = reduceIm( I, i )
%REDUCE smoothen and reduce the image
% smoothen using gaussian filter with a = 0.4
   temp = smoothenIm(I{i}, 0.4);
   temp = temp(1:2:end,1:2:end);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ temp ] = smoothenIm( Im, a )
%SMOOTHENIM Smoothen the given image
   w = [.25-.5*a .25 a .25 .25-.5*a];
   temp2 = zeros(size(Im,1),size(Im,2));
   temp = zeros(size(Im,1),size(Im,2));
   for i=3:(size(Im,1)-2)
      for j=3:2:(size(Im,2)-2)
         temp2(i,j) = Im(i,j-2)*w(1,1) + Im(i,j-1)*w(1,2) + Im(i,j)*w(1,3) + Im(i,j+1)*w(1,4) +
Im(i,j+2)*w(1,5);
      end
   end

   for i=3:2:(size(Im,1)-2)
      for j=3:2:(size(Im,2)-2)
         temp(i,j) = temp2(i-2,j)*w(1,1) + temp2(i-1,j)*w(1,2) + temp2(i,j)*w(1,3) +
temp2(i+1,j)*w(1,4) + Im(i+2,j)*w(1,5);
      end
   end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ temp ] = expandIm( I, k )
%EXPANDIM Expand the image by doubling the size
   temp = zeros(size(I{k},1)*2, size(I{k},1)*2);
   for i=1:2:size(temp,1)
      for j=1:2:size(temp,2)
         temp(i,j) = I{k}((i+1)/2,(j+1)/2);
      end
   end
   temp = interpolateIm(temp);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ temp ] = interpolateIm( temp )
%INTRAPOLATEIM Interpolate the image

  for i=1:size(temp,1)
    for j=1:size(temp,2)
      if mod(i,2)==1 && mod(j,2)==1
        continue;
      elseif mod(i,2)==1 && mod(j,2)==0
        if j ~= size(temp,2)
          temp(i,j) = (temp(i,j-1)+temp(i,j+1))/2;
        end
      elseif mod(i,2)==0 && mod(j,2)==1
        if i ~= size(temp,1)
          temp(i,j) = (temp(i-1,j)+temp(i+1,j))/2;
        end
      else
        if i ~= size(temp,1) && j ~= size(temp,2)
          temp(i,j) = (temp(i-1,j-1)+temp(i+1,j-1)+temp(i-1,j+1)+temp(i+1,j+1))/4;
        end
      end
    end
  end
end
```