Homework Assignment #3

Computer Vision for HCI - CSE 5524 Shailesh Tripathi (tripathi.52)

Q1.

4-level pyramid was constructed. The linear interpolation function looks like this

2 0 4	interpolate along cols	2 3 4	along rows	2 3 4
0 0 0	=>	0 0 0	=>	4 5 6
6 0 8		6 7 8		6 7 8

```
% Read the input image and convert to grayscale
rgbImage = imread('umaid.jpg');
grayImage = double(rgb2gray(rgbImage));

% Number of levels in the pyramid
levelCount = 4;
% Calculate appropriate size using Burt & Adelson formula
% R = Mr * 2^N + 1
% C = Mc * 2^N + 1
% If N >= levelCount, the pyramid can be generated. For simplicity, let
% N = leveCount
rows = floor((size(grayImage,1) - 1) / (2^levelCount)) * (2^levelCount) +
1;
cols = floor((size(grayImage,2) - 1) / (2^levelCount)) * (2^levelCount) +
1;
truncatedImage = grayImage(1:rows, 1:cols);

% Display Gaussian and Laplacian Pyramid
GenerateGaussianAndLaplacian(truncatedImage, levelCount);

% Function to generate the pyramids
function [] = GenerateGaussianAndLaplacian(truncatedImage, levelCount)

      a = 0.4;
      window1D = [0.25 - 0.5 * a, 0.25, a, 0.25, .25 - 0.5 * a];

      GaussianImageList = cell(1, levelCount);
      LaplacianImageList = cell(1, levelCount);

      % Level-1 of Gaussian pyramid is the original image.
      GaussianImageList{1} = truncatedImage;

      for level = 2: levelCount

      % Compute corresponding level images of the Gaussian pyramid
```

```
        % Level-i Gaussian
        GaussianImageList{level} = BlurAndSample(GaussianImageList{level-1},
window1D);

        % Now that the Gaussian image for this level is available, compute
        % Laplacian of previous level.
        LaplacianImageList{level - 1} = GaussianImageList{level-1} -
GetInterpolatedImage(GaussianImageList{level});
        end

        % The last level of Laplacian pyramid is same as last level of
        % Gaussian pyramid
        LaplacianImageList{levelCount}  = GaussianImageList{level};

        % Display each level of both the pyramids.
        for level = 1: levelCount
        imshowpair(GaussianImageList{level}/255,
LaplacianImageList{level}/255, 'montage');
        title(strcat('Gaussian-Laplacian pyramid. Level-',int2str(level)))
        pause;
        end
end

% Function to perform Gaussian blur and sample the image.
function [sampledImage] = BlurAndSample(img, window1D)
        % Gaussian blur
        % Gaussian filter applied along X axis.
        gXIm = imfilter(img, window1D, 'replicate');

        % Gaussian filter applied along Y axis.
        gIm = imfilter(gXIm, window1D', 'replicate');

        % Sample the image by selecting every 1 out of 2 pixels.
        sampledImage = gIm(1:2:end, 1:2:end);

end

% Interpolation function
function [interpolatedImage] = GetInterpolatedImage(image)
        [rows, cols] = size(image);
        interpolatedImage = zeros(2*rows-1, 2*cols-1);
```

```
        % Copy original values to appropriate indices
        interpolatedImage(1:2:end, 1:2:end) = image(:, :);

        % Interpolate along the columns
        interpolatedImage(1:2:end, 2:2:end) = 1/2*(image(:, 1:end-1) +
image(:, 2:end));

        % Interpolate along the rows.
        % The interpolated values from the previous
        % step is used to further interpolate the middle pixel value(Eg.
center
        % pixel in a 5x5 matrix, where only the 4 corners are available.
First,
        % we complete the top and bottom row. Then for the use these two rows
        % to interpolate the middle row.
        interpolatedImage(2:2:end, :) = 1/2*(interpolatedImage(1:2:end-2, :)
+ interpolatedImage(3:2:end, :));
end
```

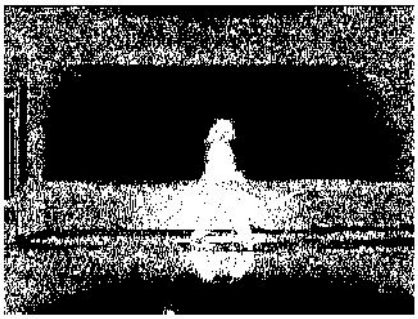
As mentioned in the code, the appropriate size of the image is computed using Burt & Adelson formula. For simplicity, the value of N is kept same as the number of the layers in the pyramid.

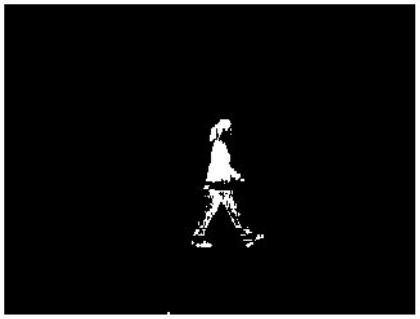
Q2.

As an object appears in the background scene, the pixel values change drastically corresponding to the image location. This difference is compared with a threshold to classify the change as a noise(discarded) or object(considered).

When the threshold was small, the noise also gets recorded. In case when threshold is very high, only very strong changes are captured which might not be very helpful in case the change in the color is very minute(eg, person's cloth color is very similar to background).

Threshold = 5 **threshold = 100**





Threshold = 150

```
backgroundImage = double(imread('bg000.bmp'));
objectImage = double(imread('walk.bmp'));

% set the threshold
T = 150;
% Compute the difference and compare with threshold
diffImage = abs(objectImage - backgroundImage) > T;

imshow(diffImage);
pause;
```

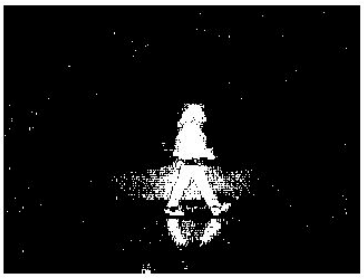
Q3.

Total of 30 images were able to produce statistical data (mean and standard deviation) for each pixel. This helps in capturing noise in the image. Since all camera have some amount of noise associated with it, it is not fair to perform background subtraction from a single image. Hence, for each pixel, some

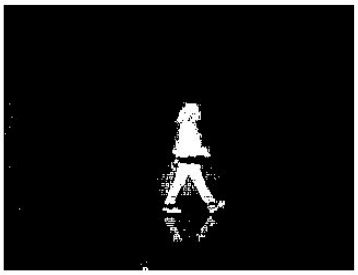
statistics are collected which can generalize that some amount of flickering is allowed. But if a pixel deviates strongly from the mean value of the pixel, it is definitely a new object in the view.
The threshold value of 15 for comparing statistical distance gave best results.

```
numBg = 30;
% Read all images from file
for i = 1:numBg
      filename = strcat('bg', num2str(i-1, '%03d'), '.bmp');
      backgroundImageArr(:,:,i) = double(imread(filename));
end

% Compute pixel-wise mean of all background images.
meanImage = mean(backgroundImageArr,3);

% Compute standard deviation
sigmaImage = std(backgroundImageArr, 1, 3);%zeros(size(objectImage));

% Compute statistical distance of object pixels from background image.
T = 15;
bsIm = (abs(objectImage - meanImage) ./ sigmaImage) > T;
imshow(bsIm);
```

Threshold = 5

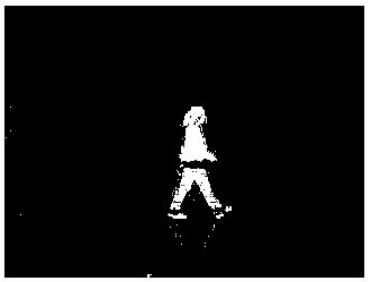


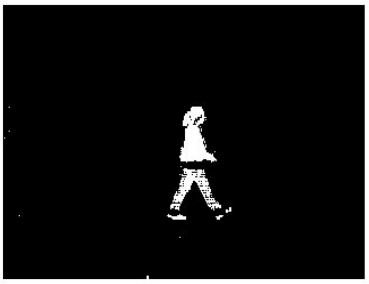
Threshold = 10



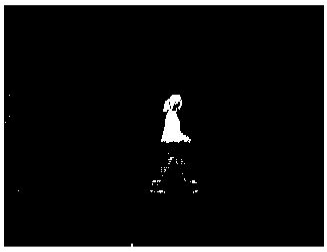
Threshold = 15

Threshold = 20





Threshold = 50



The threshold is for the statistical distance in the units of standard deviation. The higher this distance, more the probability that, it is a new object in the scene and not just noise. But if the value is set too high, some part of the object is left out as the difference is not huge.

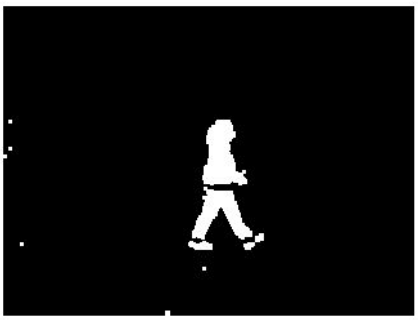
This method generally performs better than the previous method because it consider multiple images to get the feature of the background(even if some part is flickering, it won't show as a new object but will take this case in consideration).

Q4. Dilate the image to enlarge each region(and possible two regions very close to each other as can be seen in the given image).

```
% Dilate the image
d_bsIm = bwmorph(bsIm, 'dilate');
imshow(d_bsIm);
pause;
```

Threshold = 15

Threshold = 20



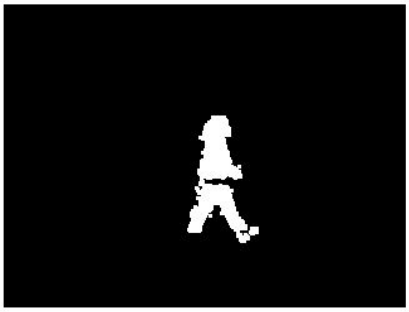
Q5.

Apply Connected component algorithm with 8 neighbors to get the labels of all the regions.

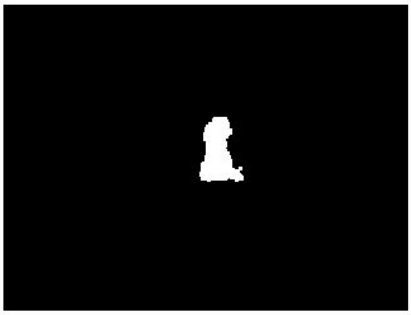
Regionprops function is use to extract the area of each and every label and that is used to largest region.

```
% Label the boxes using connected components algorithm.
[L, num] = bwlabel(d_bsIm, 8);

% Find the label with largest area using regionprops tool
stat= regionprops(L, 'Area');
[maxArea, maxLabel] =max([stat.Area]);

% Display the largest block only.
maxLabelImage = (L==maxLabel);
imshow(maxLabelImage);
```

Threshold = 15



Threshold = 20



As can be clearly seen from the results, the small variation in threshold value can make huge difference in the results. Before this step, the threshold=20 looked as a better candidate(less noise) for the image, but after this operation, it was realized that 15 might give better results(easily identifiable object).