Detection of Motor Vehicles and Humans on Ocean Shoreline

Seif Abu Bakr

Dec 14, 2009
Boston University
Department of Electrical and Computer Engineering
Technical report No.ECE-2009-05

BOSTON
UNIVERSITY

Detection of Motor Vehicles and Humans on Ocean Shoreline

SEIF ABU BAKR

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

Dec 14, 2009

Technical Report No.ECE-2009-05

Summary

The aim of this project is to present a method to identify objects within an image. The method is based on using a covariance matrix to describe the content of an image. It is a new approach that is based on non linear and non Euclidean quantities. This method will demonstrate the power of the covariance matrix as a region descriptor.

Another important part of this project is using integral images for fast covariance computation. It is similar to using a look-up table to store values that are frequently calculated and therefore reduce the computational complexity. The concept of integral images can be applied to many other calculations, not just covariance matrix computation. It can also be used in mean value computation. However, this report only describes how to use it in fast covariance computation.

The application of the covariance matrix approach is to detect cars and humans in an image, in fact, ideally, in video sequences. The report mentions the important aspects that have to be taken into consideration when choosing the feature vector and when scanning the image region by region.

Last but not least, the report discusses improvements and limitations of this approach.

Contents

List of Figures

1. Introduction

An important area of image processing research today is computer vision. It is a widely spread topic that has several applications, such as object and texture identification, motion detection and face recognition.

In this project, the aim is to identify objects in the foreground of an image, particularly on a beach. The main objects that I will focus on detecting are humans and cars.

In general, objects in an image have several properties or features. Properties of objects are shape, size, color, luminance (or reflectivity) and texture. Different objects have different properties, and inversely, different properties mold different objects. In image processing, these properties can be described by various factors; we call them features. An object, with specific properties, can be described by spatial features like primary colors, luminance and first order and second order luminance gradients.
One approach to identify an object is to calculate the color distribution of the object within an image, i.e make a histogram to describe the colors of that object. However, this approach would entail many problems, because of luminance variation from one image to another, and the histogram values will change from one picture to the other. For example, consider an image of a light blue car that has a particular color distribution in an image shot while the car has a certain luminance. Under different luminance conditions, the color saturation will change, resulting in different histogram values. Furthermore, under different resolutions, the picture of the same object will have a different window size and thus also a different histogram. Therefore, this approach is not efficient in this application.

The approach discussed in this paper is based on using the covariance matrix as a descriptor of a region in the image. This approach has computational advantages as well as higher accuracy. Calculating the covariance of a region yields a ‘d by d’ matrix, where d is the number of features used. The number of elements in the feature vector is usually between 4 to 9. This makes the range of covariance matrix size vary from 16 to 81 elements. Compared to the histogram approach mentioned above, the covariance approach provides a small d by d matrix to describe an M by N region, which significantly reduces the computational complexity.

2. Covariance matrix as an Image descriptor

Suppose we have an M by N color image with three values per pixel, i.e. Red, Green and Blue. In this case, every pixel will have a three dimensional feature vector. If we add to every pixel another dimension that carries additional information, we would have a four dimensional feature vector for every pixel, and so on… . Now consider adding two dimensions to our three dimensional RGB image, such that we have a five dimensional feature vector for every pixel. For example, if the feature vector *f* is composed of the features R, G, B, vertical luminance gradient and horizontal luminance gradient, then the feature vector for every pixel is

$$f = [\mathrm{R}, \mathrm{G}, \mathrm{B}, \frac{dL}{dx}, \frac{dL}{dy}]^{\mathrm{T}}$$

The covariance matrix will be calculated as follows

$$\mathrm{C}_R = \frac{1}{n-1}\sum_{k=1}^{n}(\mathrm{z}_k - \mu)(\mathrm{z}_k - \mu)^T$$

Where μ is the mean feature vector for a region in the image with *n* pixels. In this case, the covariance matrix will look like this:-

$$\begin{pmatrix}
\mathrm{cov}(R,R) & \mathrm{cov}(R,G) & \mathrm{cov}(R,B) & \mathrm{cov}(R,\frac{dI}{dx}) & \mathrm{cov}(R,\frac{dI}{dy}) \\
\mathrm{cov}(G,R) & \mathrm{cov}(G,G) & \mathrm{cov}(G,B) & \mathrm{cov}(G,\frac{dI}{dx}) & \mathrm{cov}(G,\frac{dI}{dy}) \\
\mathrm{cov}(B,R) & \mathrm{cov}(B,G) & \mathrm{cov}(B,B) & \mathrm{cov}(B,\frac{dI}{dx}) & \mathrm{cov}(B,\frac{dI}{dy}) \\
\mathrm{cov}(\frac{dI}{dx},R) & \mathrm{cov}(\frac{dI}{dx},G) & \mathrm{cov}(\frac{dI}{dx},B) & \mathrm{cov}(\frac{dI}{dx},\frac{dI}{dx}) & \mathrm{cov}(\frac{dI}{dx},\frac{dI}{dy}) \\
\mathrm{cov}(\frac{dI}{dy},R) & \mathrm{cov}(\frac{dI}{dy},G) & \mathrm{cov}(\frac{dI}{dy},B) & \mathrm{cov}(\frac{dI}{dy},\frac{dI}{dx}) & \mathrm{cov}(\frac{dI}{dy},\frac{dI}{dy})
\end{pmatrix}$$

Where *I* is the grayscale luminance.

2.1 Comparing two covariance matrices

From the previous page we know how two compute the covariance matrix of an image or a region within the image. In this part, we will show how to compare two covariance matrices and decide whether they describe the same object or not.

A covariance matrix has a nonlinear relation with its corresponding feature image. In general, variances have a square relationship with their corresponding observation. Similarly, covariance matrices have a square relationship with their corresponding feature image. Therefore, we cannot just subtract two covariance matrices from each other to check for their similarity. Minimum distance rule is valid for Euclidean quantities, but not for quantities like covariance matrices, which exist in Riemannian space.

The metric for comparison between two covariance matrices is

$$\rho(\mathbf{C}_1, \mathbf{C}_2) = \sqrt{\sum_{i=1}^{n} \ln^2 \lambda_i(\mathbf{C}_1, \mathbf{C}_2)}$$

Where $\lambda_i(C_1,C_2)$ are generalized Eigenvalues for the matrices C_1 & C_2. The metric is a scalar value, which indicates the dissimilarity between the two images with covariance matrices C_1 & C_2. In other words, the smaller this number is, the greater the similarity between the two images. If $C_1 = C_2$, then the result of the metric will be $\rho=0$.

2.2 Efficient scanning using Integral Images

Now we know how to compute a covariance matrix of an image and how to compare two covariance matrices to check for the similarity of two images. However, we have to be able to locate the object we want to identify within the image, in order to compute the covariance for it, and then compare it to another reference covariance matrix. For example, the covariance matrix describing the image in Figure 1 is different from that describing Figure 2, although the car in Figure 2 is in Figure 1. Therefore, we need to scan through region by region in Figure 1 and compare every region with a reference covariance matrix, which is the covariance matrix describing Figure 2 in this case.

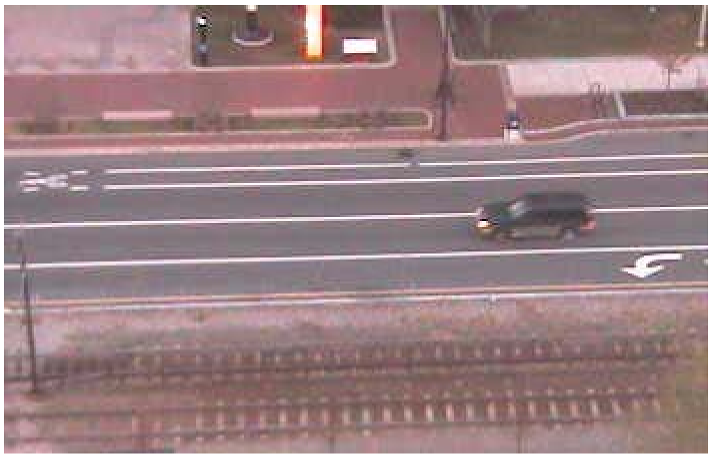

Figure 1: demonstration.

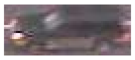

Figure 2: demonstration.

If we divide the image into small rectangular regions and compute the covariance matrix for every region, it will be an inefficient scanning algorithm. We would have to go through a nested for loop to compute the covariance matrix for every pixel in the rectangular region and then sum them up and divide by the number of pixels. Doing that for every rectangular area in the image would take too much time.

A better way to do this is to use integral images. To get the integral image, we first compute the covariance matrix for every pixel in the image. This will give us a four dimensional block (Let the four dimensional block be named *CI)*; for every pixel in the M by N image we have a d by d matrix describing it. The next step is to sum up all the covariance matrices up to a point *(x,y)* in the four dimensional block. The point *(x,y)* on the integral image is the summation of all covariance matrices up to that point in *CI*. For example, the covariance matrix at point (2,2) in the integral image is the sum of the covariance matrices at the points (1,1), (1,2), (2,1) and (2,2) in *CI.* Moreover, the covariance matrix at point (M,N) in the integral image is the sum of all covariance matrices in *CI*.

To illustrate the significance of integral images, consider figure 3. The point (x', y') is described by a covariance matrix that is the sum of all covariance matrices of pixels in the yellow region. Similarly, the point (x'', y'') is described by a covariance matrix that is the sum of all covariance matrices of pixels in the green, yellow, blue and red regions. So in order to calculate the covariance matrix describing the yellow region, we would have to divide the matrix at point (x', y') by the number of pixels in the yellow region. Also, to calculate the covariance matrix describing all the colored regions, we divide the matrix in at point (x'', y'') by the number of pixels in the colored regions.

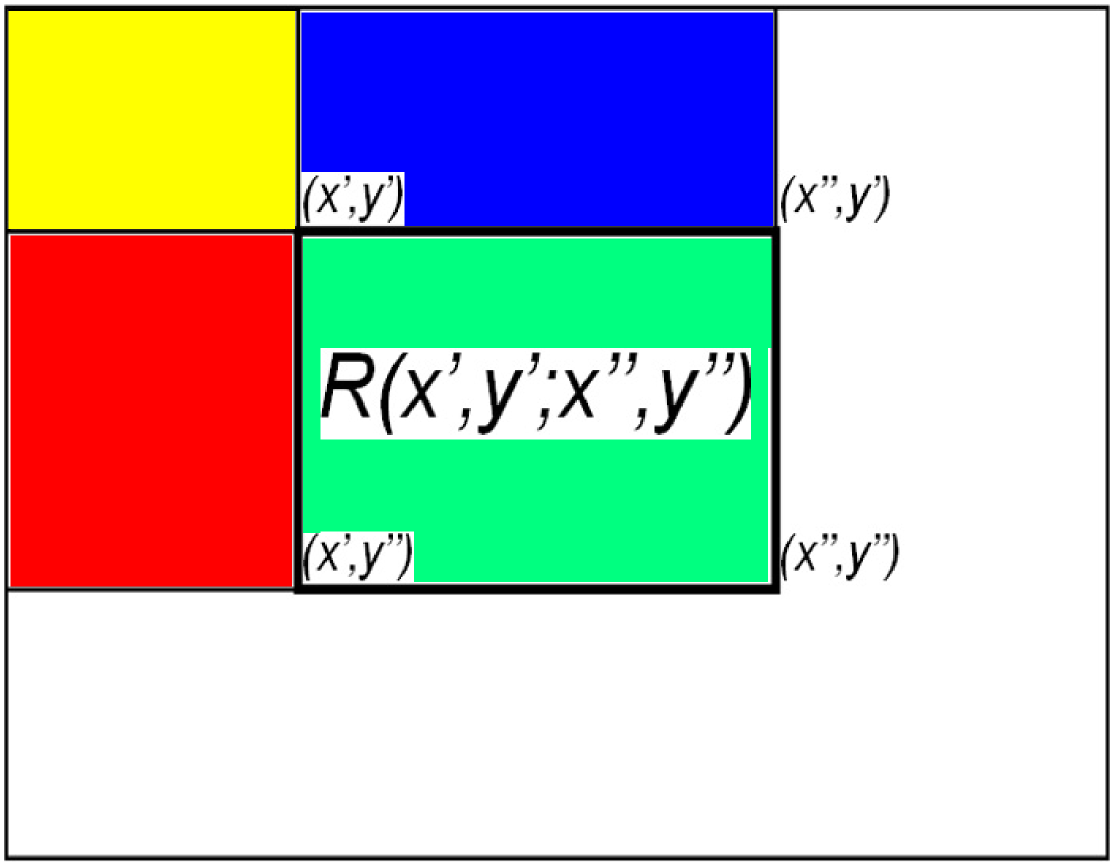

Figure 3: Integral Image.

Now suppose we want to compute the covariance of the green region alone. The point (x'', y''), as mentioned before, includes the sum of all the regions' covariance matrices. Thus, we have to subtract the points (x'', y') and (x', y'') from (x'', y'') to remove the summed values of the blue and red regions from the matrix in the point (x'', y''). However, by subtracting the points (x'', y') and (x', y'') from (x'', y'') we would be removing the sum of the yellow region twice, so we need to compensate for that by adding the point (x', y').

The equation for calculating the covariance matrix of the green region is

$$covmatrix = [INTIM(x'',y'') - INTIM(x',y'') - INTIM(x'',y') + INTIM(x',y')]/(\#\ of\ pixels)$$

Where *INTIM* is the integral image and *covmatrix* is the covariance matrix of the green region.

As a result, using integral images we can calculate the covariance of a region by two subtractions, one addition and one division. Computationally, this is much more efficient than taking every region and calculating the covariance matrix for it using the covariance equation.

3. Identifying objects

To identify objects, we need to choose a feature vector that carry essential information about the target objects. For example, to identify oranges in an image, we need to choose a feature vector that includes Red, Green, Blue, gradients and Laplacians. Including Red, Green and blue is important because oranges have a unique color. Gradients and Laplacians are important to indicate the round shape of the orange.

Besides choosing the right feature vector, having a rich database that has covariance matrices of the object's different orientations increases the accuracy significantly. This is even more important when target objects are non-rigid objects that can appear deformed in an image.

Scanning through the image is also a factor that can be optimized. For example, in the case of identifying objects on the beach, we should not compute covariance matrices for the upper regions of the picture, which most likely is the sky. So we should only check regions in the image that are in the area of interest.

3.1 Identifying Motor vehicles

To identify Motor vehicles, the feature vector was chosen as follows

$$f = \left[\frac{dI}{dx}, \frac{dI}{dy}, \frac{d^2 I}{dx^2}, \frac{d^2 I}{dy^2} \right]^T$$

where I is the grayscale luminance of the image.

Color features are not used here because cars are not distinguished by colors. The same shaped car can appear in two different colors. On the other hand, vertical and horizontal gradients and Laplacians are good descriptors of cars because they carry important information to distinguish cars from other objects.

We wish to identify cars, and distinguish SUVs from Sedans. Therefore, there should be two different databases, one for Sedans and the other for SUVs. The Sedan cars and the SUVs that were used in the database are shown in Figures 4 & 5, respectively. We store the covariance matrices of these cars in a database for cars.

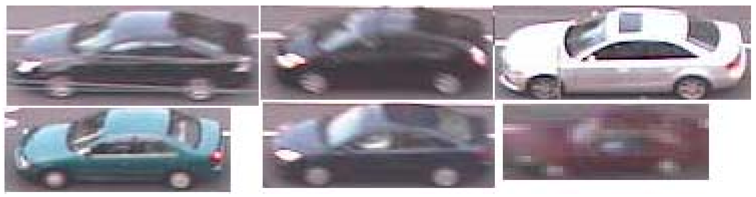
Figure 4: Sedan cars in database.

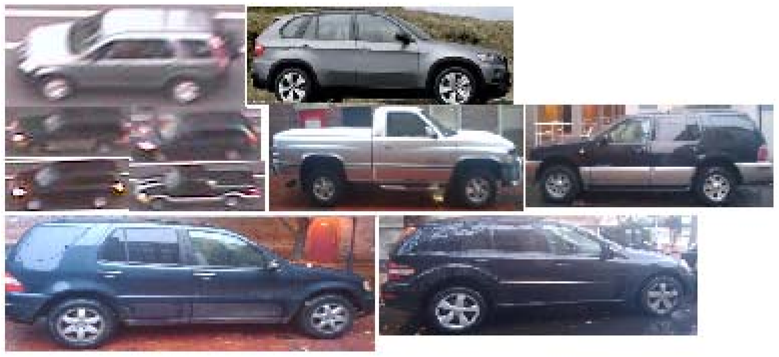
Figure 5: SUVs in database.

The algorithm starts by computing the grayscale luminance of the image and hence the gradients and Laplacians of the image. The next step is to calculate the integral image. Then we start scanning the image region by region. However, we only scan regions of interest, i.e. we scan the street because we are looking for cars and that is where they are. For every region we scan through we compute its covariance matrix using the calculated integral image. Then we compare it to the cars in the database. First we compare it to the SUVs; if the number from the comparison metric is smaller than a certain threshold, we store this number in an array dedicated for the SUVs comparison numbers. We also store the corresponding coordinates of the rectangular region. When we have scanned all the regions in the image, we take the minimum comparison number in the SUV array and we decide that this is the region where the SUV is most likely located. It is also recommended to scan the whole image more than once with changing sizes of the rectangular region, in order to account for the different sizes of the cars that can appear. We do the same steps for Sedans. The MATLAB algorithm for this is in the Appendix.

3.2 Identifying Humans

Similar to the case of identifying cars, we use the same feature vector to identify humans:-

$$f = \left[\frac{dI}{dx}, \frac{dI}{dy}, \frac{d^2 I}{dx^2}, \frac{d^2 I}{dy^2} \right]^T$$

Again, we did not use colors as features because colors do not distinguish humans. Humans have different skin colors and can appear in clothes with different colors. Therefore, we also use gradients and Laplacians because they carry important information that can distinguish humans.

Like the case of cars in 3.1, we have a database for humans. The database stores the covariance matrices of reference images that we use in comparison. Figure 6 shows the pictures used in the database.

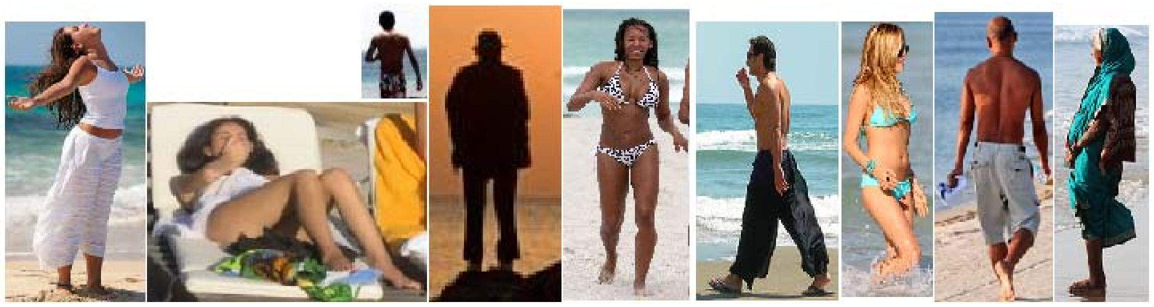

Figure 6: Human database.

When identifying one single human, the algorithm is exactly the same as the one used to identify cars. However, we can also identify multiple humans by taking the least comparison number for different rectangle sizes. The code for that is in the Appendix.

4. Experimental Results

4.1 Experimental Results for Motor vehicles

For testing the algorithm on cars, snapshots of Commonwealth Avenue were used. The results were good for a small database like the one used (see Figure 7)

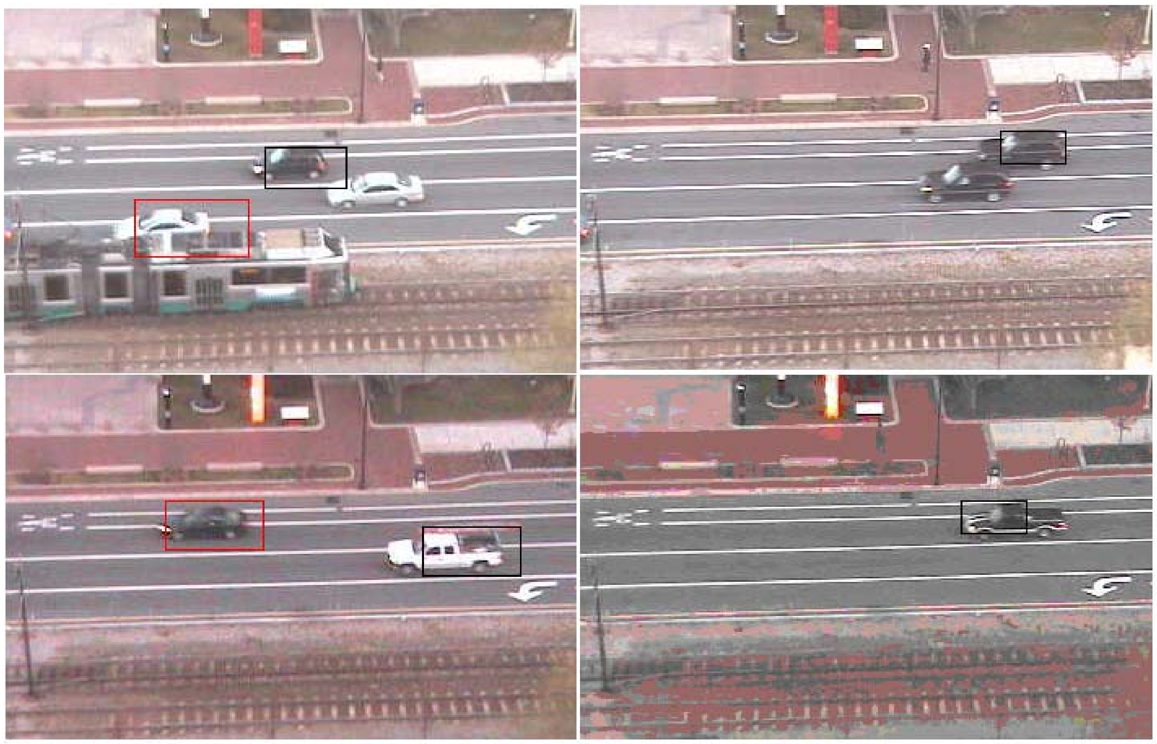

Figure 7: Successful Detection: Red Box for Sedan, Black Box for SUV.

However, there were also errors in detection. Sometimes, SUVs were identified as Sedans and vice versa. Other times, the stripes on the street were identified as cars. These errors occur because we use gradients as features, and the stripes in the streets can create shapes that can have a covariance matrix similar to one of the stored covariance matrices. If we have a bigger database, these kinds of errors would rarely happen. Some examples of the errors are shown in figure 8.

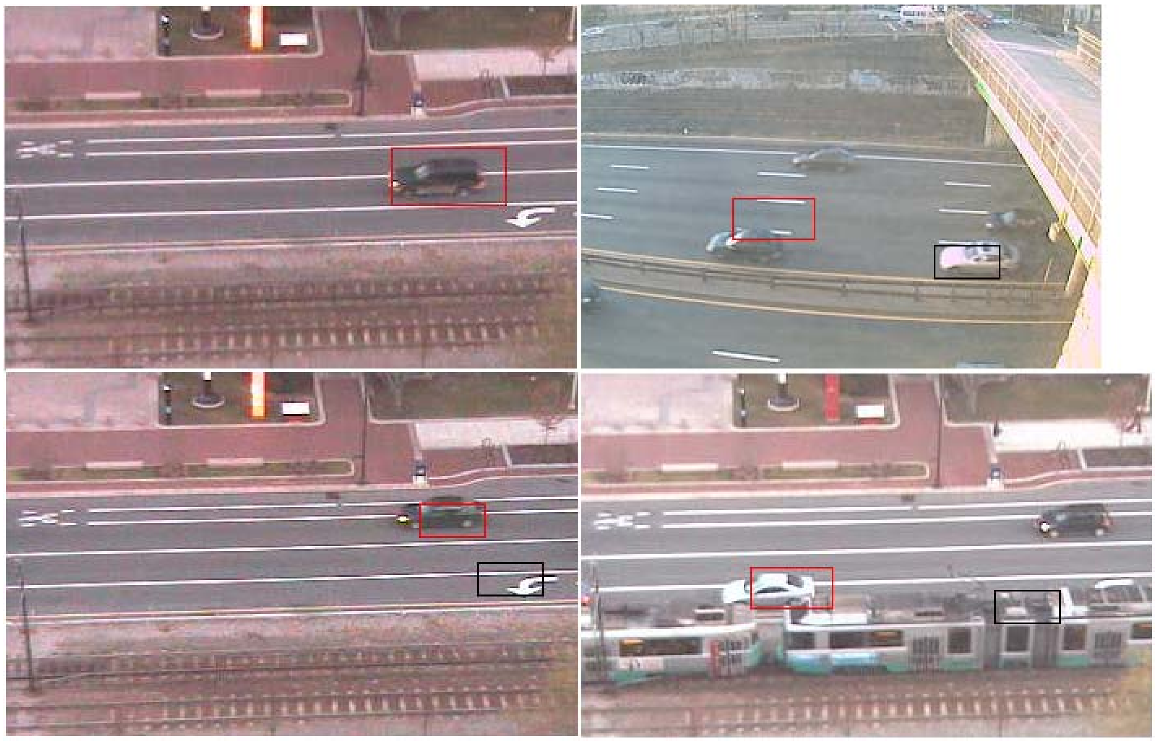

Figure 8: Errors in detection: Red Box for Sedan, Black Box for SUV.

4.2 Experimental Results for Humans

The images that were used in testing the algorithm on humans were pictures of humans at the beach. Therefore, it was easier to identify humans because there are no surrounding details like the case of cars on the street. The background was sand and sky, so the gradients did not carry misleading information.

Two algorithms were used in testing images with humans. The first was a single object detection algorithm. The code for it can be found in the Appendix and the results are shown in Figure 9.

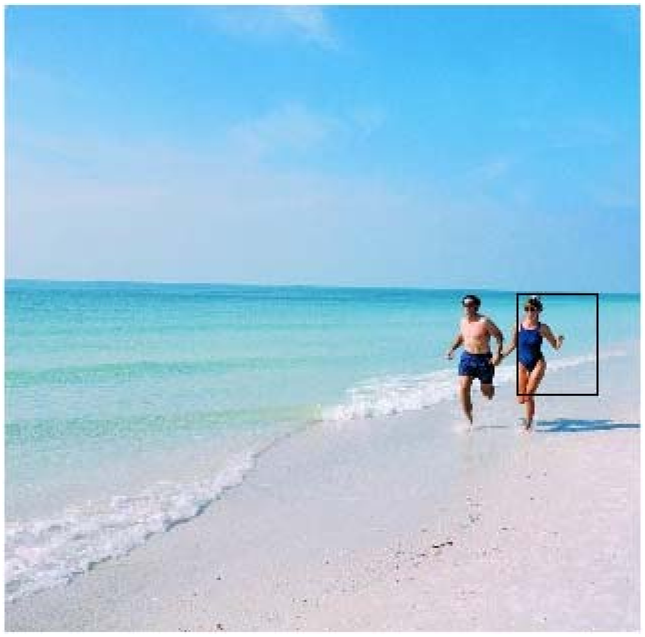

Figure 9: single detection.

The second algorithm used was aimed at detecting multiple humans in the image. It had some successful results and some errors as well. Figure 10 shows a successful triple detection.

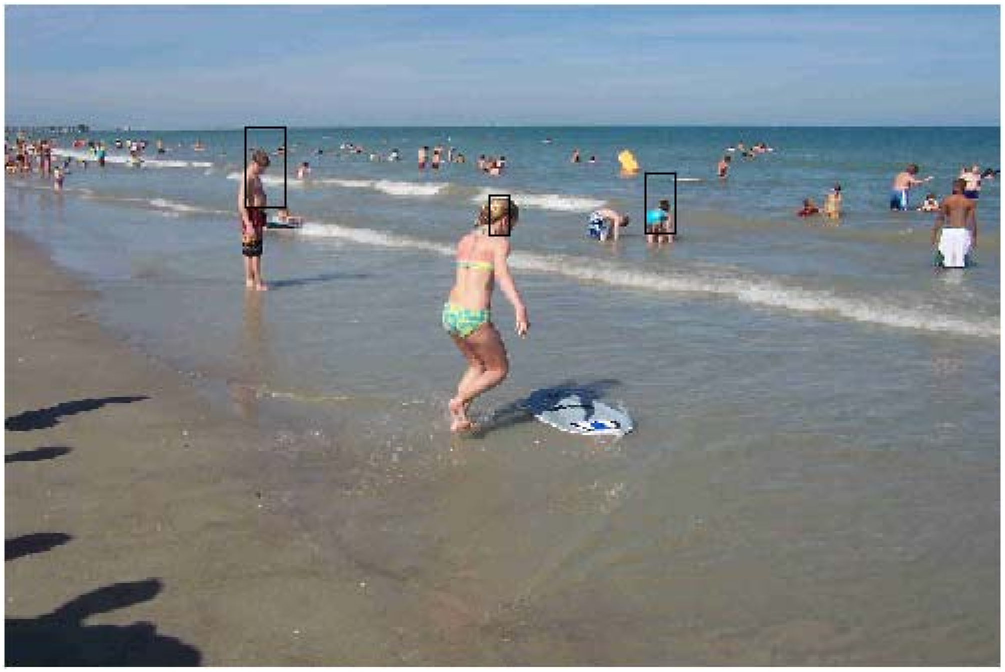

Figure 10: Multiple detection (three targets).

When using a 5 target detection algorithm, some targets are detected more than once. Multiple target detection is about changing the rectangular region size and scanning the image more than once with the different rectangle sizes. It happens that for two rectangle sizes, the same region is the most likely region to be the target. If we have a bigger database with more orientations inside, this error (figure 11) would happen rarely.

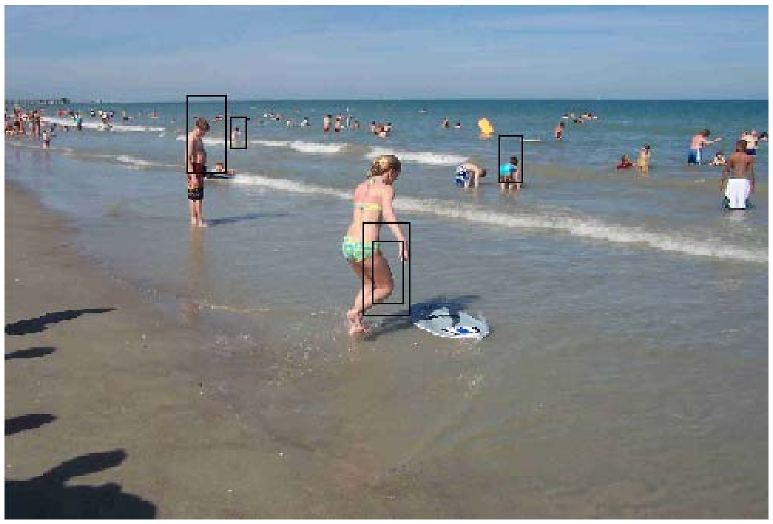
Figure 11: Multiple detection (five targets) with error.

Another error occurs when testing images like figure 12. In this case, the shape behind the man walking in the water created a curvy shape that distorted the detection. This can be avoided if we have a bigger database.

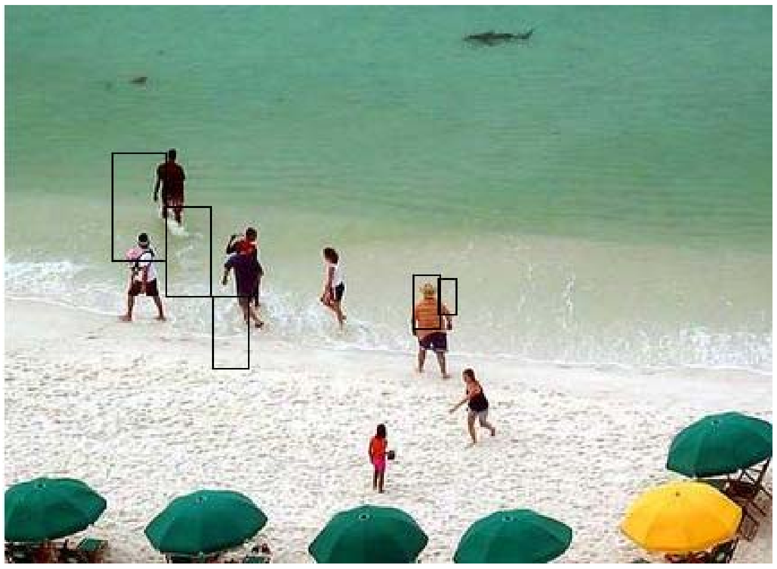
Figure 12: Five target detection, multiple errors.

5. Conclusions and possible improvements

The method is very efficient, although we used only a small database. However, we need to do some improvements and set some guidelines before we can implement a real time system that uses this algorithm. The following steps should be done:

1. Build a bigger Database for every desired target object. The database should contain at least 50 entries for each object, i.e. 50 covariance matrices for each Sedans, SUVs and humans.
2. Every Camera should have its own database that is specifically designed for it. If the camera is likely to capture images of cars from the side, then the database should include covariance matrices that describe side views of cars.
3. Combine this method with motion detection for video sequences. This will help to locate vehicles within the image efficiently. Also, it would locate vehicle faster and decrease the processing time of the algorithm.

Appendix

1. MATLAB code for detection of Sedan and SUV:-

```
clear
clc

%loading the database for Cars:-
load database

%___________________________________________________________
%main code
stepsize=2;                           %stepsize in scanning the image
barb=(double(imread('commav31','tif')))/255;
% barb=barb(200:224,300:352,:);
s=size(barb);
i=rgb2gray(barb);
dery=i(2:s(1),:)-i(1:s(1)-1,:);       %//first order vertical gradient
derx=i(:,2:s(2))-i(:,1:s(2)-1);       %//first order horizontal gradient
x(:,:,1)=[zeros(1,s(2)); dery];   %//setting the first row to zeros to
adjust matrix size
x(:,:,2)=[zeros(s(1),1) derx];    %//setting the first column to zeros
to adjust matrix size
dery2=x(2:s(1),:,1)-x(1:s(1)-1,:,1);  %//second order vertical gradient
derx2=x(:,2:s(2),2)-x(:,1:s(2)-1,2);   %//second order horizontal
gradient
x(:,:,3)=[zeros(1,s(2)); dery2];
x(:,:,4)=[zeros(s(1),1) derx2];
mew=sum(sum(x))/(s(1)*s(2));
minim=1;
for u=1:s(1);
    for v=1:s(2);
        c1=(x(u,v,:)-mew);
        covariance(u,v,:,:)=[c1(1); c1(2); c1(3); c1(4)]*[c1(1) c1(2)
c1(3) c1(4)];
    end
end
%computing integral image____________________________
integralim(1,1,:,:)=covariance(1,1,:,:);
for u=2:s(1);
    integralim(u,1,:,:)=sum(covariance(1:u,1,:,:));
end
for u=2:s(2);
    integralim(1,u,:,:)=sum(covariance(1,1:u,:,:));
end
for u=2:s(1);
    for v=2:s(2)
        integralim(u,v,:,:)=covariance(u,v,:,:)+integralim(u,v-
1,:,:)+integralim(u-1,v,:,:)-integralim(u-1,v-1,:,:);
    end
end
%finished computing integral image____________________________
for kk=1:4;
count=1;
count2=1;
```

```
rowsize(kk)=15+5*kk;
colsize(kk)=30+10*kk;
for u=70:stepsize:155-rowsize(kk);
    for v=1:stepsize:s(2)-colsize(kk);

c=integralim(u,v,:,:)+integralim(u+rowsize(kk),v+colsize(kk),:,:)-
integralim(u,v+colsize(kk),:,:)-integralim(u+rowsize(kk),v,:,:);
        covmatr(:,:)=c(:,:,1:4,1:4)/(rowsize(kk)*colsize(kk));
%calculating the covariance of a region
        rho(1)=sqrt(sum((log(eig(covmatr,covmatr1))).^2));
%comparing with database
        rho(2)=sqrt(sum((log(eig(covmatr,covmatr2))).^2));
%|
        rho(3)=sqrt(sum((log(eig(covmatr,covmatr3))).^2));
%|
        rho(4)=sqrt(sum((log(eig(covmatr,covmatr4))).^2));
%|
        rho(5)=sqrt(sum((log(eig(covmatr,covmatr5))).^2));
%|
        rho(6)=sqrt(sum((log(eig(covmatr,covmatr6))).^2));
%|
        rho(7)=sqrt(sum((log(eig(covmatr,covmatr7))).^2));
%|
        rho(8)=sqrt(sum((log(eig(covmatr,covmatr8))).^2));
%|
        rho(9)=sqrt(sum((log(eig(covmatr,covmatr9))).^2));
%|
        rho(10)=sqrt(sum((log(eig(covmatr,covmatr10))).^2));
%|
        rho(11)=sqrt(sum((log(eig(covmatr,covmatr11))).^2));
%|
        minv=min(rho);
        if(minv<=minim)
            covnum(count)=find(rho==min(rho));
            rho2(count)=minv;
            m(count)=u;
            n(count)=v;
            count=count+1;
        end
        rho1(1)=sqrt(sum((log(eig(covmatr,covmatrs1))).^2));
%sedan
        rho1(2)=sqrt(sum((log(eig(covmatr,covmatrs2))).^2));
%sedan
        rho1(3)=sqrt(sum((log(eig(covmatr,covmatrs3))).^2));
%sedan
        rho1(4)=sqrt(sum((log(eig(covmatr,covmatrs4))).^2));
%sedan
        rho1(5)=sqrt(sum((log(eig(covmatr,covmatrs5))).^2));
%sedan
        rho1(6)=sqrt(sum((log(eig(covmatr,covmatrs6))).^2));
%sedan
        rho1(7)=sqrt(sum((log(eig(covmatr,covmatrs7))).^2));
%sedan
        minv2=min(rho1);
        if(minv2<=minim)
%sedan
            covnum12(count2)=find(rho1==min(rho1));
            rho12(count2)=minv2;
            m12(count2)=u;
            n12(count2)=v;
            count2=count2+1;
        end
    end
end
if(count>1)
rhores(kk)=min(rho2);
coords1=find(rho2==min(rho2));
mm1(kk)=m(coords1);
nn1(kk)=n(coords1);
end
clear rho2
if(count2>1)                                %sedan
rhores1(kk)=min(rho12);
coords11=find(rho12==min(rho12));
mm12(kk)=m12(coords11);
nn12(kk)=n12(coords11);
end
clear rho12
end
if(count>1)
coords=find(rhores==min(rhores));
mm=mm1(coords);
nn=nn1(coords);
barb(mm:mm+rowsize(coords),nn,:)=0;                     %labeling SUV
in black box
barb(mm:mm+rowsize(coords),nn+colsize(coords),:)=0;
barb(mm,nn:nn+colsize(coords),:)=0;
barb(mm+rowsize(coords),nn:nn+colsize(coords),:)=0;
end
if(count2>1)
coords2=find(rhores1==min(rhores1));                %sedan RED
mm2=mm12(coords2);
nn2=nn12(coords2);
barb(mm2:mm2+rowsize(coords2),nn2,1)=255;               %labeling sedan
in red box
barb(mm2:mm2+rowsize(coords2),nn2,2:3)=0;
barb(mm2:mm2+rowsize(coords2),nn2+colsize(coords2),1)=255;
barb(mm2:mm2+rowsize(coords2),nn2+colsize(coords2),2:3)=0;
barb(mm2,nn2:nn2+colsize(coords2),1)=255;
barb(mm2,nn2:nn2+colsize(coords2),2:3)=0;
barb(mm2+rowsize(coords2),nn2:nn2+colsize(coords2),1)=255;
barb(mm2+rowsize(coords2),nn2:nn2+colsize(coords2),2:3)=0;
end
imshow(barb)
```

2. MATLAB Code for single detection of Humans:-

```
clear
clc

%Database:-

load databaseh


%_______________________________________________________________________
____

%main code
magnification=1;
stepsize=2;
barb=(double(imread('beach2','tif')))/255;
% barb=barb(200:224,300:352,:);
s=size(barb);
i=rgb2gray(barb);
dery=i(2:s(1),:)-i(1:s(1)-1,:);      %//first order vertical gradient
derx=i(:,2:s(2))-i(:,1:s(2)-1);      %//first order horizontal gradient
x(:,:,1)=[zeros(1,s(2)); dery];   %//setting the first row to zeros to
adjust matrix size
x(:,:,2)=[zeros(s(1),1) derx];   %//setting the first column to zeros
to adjust matrix size
dery2=x(2:s(1),:,1)-x(1:s(1)-1,:,1);  %//second order vertical gradient
derx2=x(:,2:s(2),2)-x(:,1:s(2)-1,2);   %//second order horizontal
gradient
x(:,:,3)=[zeros(1,s(2)); dery2];
x(:,:,4)=[zeros(s(1),1) derx2];
mew=sum(sum(x))/(s(1)*s(2));
minim=1;
for u=1:s(1);
    for v=1:s(2);
        c1=(x(u,v,:)-mew);
        covariance(u,v,:,:)=[c1(1); c1(2); c1(3); c1(4)]*[c1(1) c1(2)
c1(3) c1(4)];
    end
end
%computing integral image________________
integralim(1,1,:,:)=covariance(1,1,:,:);
for u=2:s(1);
    integralim(u,1,:,:)=sum(covariance(1:u,1,:,:));
end
for u=2:s(2);
    integralim(1,u,:,:)=sum(covariance(1,1:u,:,:));
end
for u=2:s(1);
    for v=2:s(2)
        integralim(u,v,:,:)=covariance(u,v,:,:)+integralim(u,v-
1,:,:)+integralim(u-1,v,:,:)-integralim(u-1,v-1,:,:);
    end
end
%finished computing integral image____________________________
for kk=1:5;
count=1;
```

```
rowsize(kk)=magnification*(5+5*(kk-1));
colsize(kk)=magnification*(4+2*(kk-1));
for u=1:stepsize:s(1)-rowsize(kk);
    for v=1:stepsize:s(2)-colsize(kk);

c=integralim(u,v,:,:)+integralim(u+rowsize(kk),v+colsize(kk),:,:)-
integralim(u,v+colsize(kk),:,:)-integralim(u+rowsize(kk),v,:,:);
        covmatr(:,:)=c(:,:,1:4,1:4)/(rowsize(kk)*colsize(kk));
%calculating the covariance of a region
        rho(1)=sqrt(sum((log(eig(covmatr,covmatrh1))).^2));
%comparing with database
        rho(2)=sqrt(sum((log(eig(covmatr,covmatrh2))).^2));
%|
        rho(3)=sqrt(sum((log(eig(covmatr,covmatrh3))).^2));
%|
        rho(4)=sqrt(sum((log(eig(covmatr,covmatrh4))).^2));
%|
        rho(5)=sqrt(sum((log(eig(covmatr,covmatrh5))).^2));
%|
        rho(6)=sqrt(sum((log(eig(covmatr,covmatrh6))).^2));
%|
        rho(7)=sqrt(sum((log(eig(covmatr,covmatrh7))).^2));
%|
        rho(8)=sqrt(sum((log(eig(covmatr,covmatrh8))).^2));
%|
        rho(9)=sqrt(sum((log(eig(covmatr,covmatrh9))).^2));
%|
        rho(10)=sqrt(sum((log(eig(covmatr,covmatrh10))).^2));
%|
        minv=min(rho);
        if(minv<=minim)
            covnum(count)=find(rho==min(rho));
            rho2(count)=minv;
            m(count)=u;
            n(count)=v;
            count=count+1;
        end
    end
end
if(count>1)
rhores(kk)=min(rho2);
coords1=find(rho2==min(rho2));
mm1(kk)=m(coords1);
nn1(kk)=n(coords1);
end
clear rho2
end
coords=find(rhores==min(rhores));
mm=mm1(coords);
nn=nn1(coords);
barb(mm:mm+rowsize(coords),nn,:)=0;                          %labeling
human in black
barb(mm:mm+rowsize(coords),nn+colsize(coords),:)=0;
barb(mm,nn:nn+colsize(coords),:)=0;
barb(mm+rowsize(coords),nn:nn+colsize(coords),:)=0;

imshow(barb)
```

3. MATLAB Code for Multiple detection of Humans:-

```
clear
clc

%Database:-

load databaseh

%______________________________________________________________________
____

%main code
stepsize=2;
magnification=2;
barb=(double(imread('beach1','tif')))/255;
s=size(barb);
i=rgb2gray(barb);
dery=i(2:s(1),:)-i(1:s(1)-1,:);     %//first order vertical gradient
derx=i(:,2:s(2))-i(:,1:s(2)-1);     %//first order horizontal gradient
x(:,:,1)=[zeros(1,s(2)); dery];  %//setting the first row to zeros to
adjust matrix size
x(:,:,2)=[zeros(s(1),1) derx];   %//setting the first column to zeros
to adjust matrix size
dery2=x(2:s(1),:,1)-x(1:s(1)-1,:,1);  %//second order vertical gradient
derx2=x(:,2:s(2),2)-x(:,1:s(2)-1,2);   %//second order horizontal
gradient
x(:,:,3)=[zeros(1,s(2)); dery2];
x(:,:,4)=[zeros(s(1),1) derx2];
mew=sum(sum(x))/(s(1)*s(2));
minim=0.55;               %threshold value
for u=1:s(1);
    for v=1:s(2);
        c1=(x(u,v,:)-mew);
        covariance(u,v,:,:)=[c1(1); c1(2); c1(3); c1(4)]*[c1(1) c1(2)
c1(3) c1(4)];
    end
end
%computing integral image____________________________
integralim(1,1,:,:)=covariance(1,1,:,:);
for u=2:s(1);
    integralim(u,1,:,:)=sum(covariance(1:u,1,:,:));
end
for u=2:s(2);
    integralim(1,u,:,:)=sum(covariance(1,1:u,:,:));
end
for u=2:s(1);
    for v=2:s(2)
        integralim(u,v,:,:)=covariance(u,v,:,:)+integralim(u,v-
1,:,:)+integralim(u-1,v,:,:)-integralim(u-1,v-1,:,:);
    end
end
%finished computing integral image_________________________________
for kk=1:5;
count=1;
```

```
rowsize(kk)=magnification*(20+10*(kk-1));
colsize(kk)=magnification*(10+5*(kk-1));
for u=1:stepsize:s(1)-rowsize(kk);
    for v=1:stepsize:s(2)-colsize(kk);

c=integralim(u,v,:,:)+integralim(u+rowsize(kk),v+colsize(kk),:,:)-
integralim(u,v+colsize(kk),:,:)-integralim(u+rowsize(kk),v,:,:);
        covmatr(:,:)=c(:,:,1:4,1:4)/(rowsize(kk)*colsize(kk));
%calculating the covariance of a region
        rho(1)=sqrt(sum((log(eig(covmatr,covmatrh1))).^2));
%comparing with database
        rho(2)=sqrt(sum((log(eig(covmatr,covmatrh2))).^2));
%|
        rho(3)=sqrt(sum((log(eig(covmatr,covmatrh3))).^2));
%|
        rho(4)=sqrt(sum((log(eig(covmatr,covmatrh4))).^2));
%|
        rho(5)=sqrt(sum((log(eig(covmatr,covmatrh5))).^2));
%|
        rho(6)=sqrt(sum((log(eig(covmatr,covmatrh6))).^2));
%|
        rho(7)=sqrt(sum((log(eig(covmatr,covmatrh7))).^2));
%|
        rho(8)=sqrt(sum((log(eig(covmatr,covmatrh8))).^2));
%|
        rho(9)=sqrt(sum((log(eig(covmatr,covmatrh9))).^2));
%|
        rho(10)=sqrt(sum((log(eig(covmatr,covmatrh10))).^2));
%|
        minv=min(rho);
        if(minv<=minim)
            covnum(count)=find(rho==min(rho));
            rho2(count)=minv;
            m(count)=u;
            n(count)=v;
            count=count+1;
        end
    end
end
if(count>1)
rhores(kk)=min(rho2);
coords1=find(rho2==min(rho2));
mm1(kk)=m(coords1);
nn1(kk)=n(coords1);
end
clear rho2
end
coords=find(rhores==min(rhores));
mm=mm1(coords);
nn=nn1(coords);
barb(mm1(1):mm1(1)+rowsize(1),nn1(1),:)=0;                  %labeling
target 1 with a black box
barb(mm1(1):mm1(1)+rowsize(1),nn1(1)+colsize(1),:)=0;
barb(mm1(1),nn1(1):nn1(1)+colsize(1),:)=0;
barb(mm1(1)+rowsize(1),nn1(1):nn1(1)+colsize(1),:)=0;
barb(mm1(2):mm1(2)+rowsize(2),nn1(2),:)=0;                    %labeling
target 2 with a black box
barb(mm1(2):mm1(2)+rowsize(2),nn1(2)+colsize(2),:)=0;
barb(mm1(2),nn1(2):nn1(2)+colsize(2),:)=0;
barb(mm1(2)+rowsize(2),nn1(2):nn1(2)+colsize(2),:)=0;

barb(mm1(3):mm1(3)+rowsize(3),nn1(3),:)=0;                    %labeling
target 3 with a black box
barb(mm1(3):mm1(3)+rowsize(3),nn1(3)+colsize(3),:)=0;
barb(mm1(3),nn1(3):nn1(3)+colsize(3),:)=0;
barb(mm1(3)+rowsize(3),nn1(3):nn1(3)+colsize(3),:)=0;

barb(mm1(4):mm1(4)+rowsize(4),nn1(4),:)=0;                    %labeling
target 4 with a black box
barb(mm1(4):mm1(4)+rowsize(4),nn1(4)+colsize(4),:)=0;
barb(mm1(4),nn1(4):nn1(4)+colsize(4),:)=0;
barb(mm1(4)+rowsize(4),nn1(4):nn1(4)+colsize(4),:)=0;

barb(mm1(5):mm1(5)+rowsize(5),nn1(5),:)=0;                    %labeling
target 5 with a black box
barb(mm1(5):mm1(5)+rowsize(5),nn1(5)+colsize(5),:)=0;
barb(mm1(5),nn1(5):nn1(5)+colsize(5),:)=0;
barb(mm1(5)+rowsize(5),nn1(5):nn1(5)+colsize(5),:)=0;
figure
imshow(barb)
```

4. MATLAB code for calculating the covariance matrix of an image read from directory

```
clear
clc

barb=(double(imread('test5','tif')))/255;
s=size(barb);
i=rgb2gray(barb);
dery=i(2:s(1),:)-i(1:s(1)-1,:);        %//first order vertical gradient
derx=i(:,2:s(2))-i(:,1:s(2)-1);        %//first order horizontal gradient
x(:,:,1)=[zeros(1,s(2)); dery];  %//setting the first row to zeros to
adjust matrix size
x(:,:,2)=[zeros(s(1),1) derx];   %//setting the first column to zeros
to adjust matrix size
dery2=x(2:s(1),:,1)-x(1:s(1)-1,:,1);  %//second order vertical
gradient
derx2=x(:,2:s(2),2)-x(:,1:s(2)-1,2);   %//second order horizontal
gradient
x(:,:,3)=[zeros(1,s(2)); dery2];
x(:,:,4)=[zeros(s(1),1) derx2];
x=x+0.5;
count=1;
mew=sum(sum(x))/(s(1)*s(2));
for u=1:s(1);
   for v=1:s(2);
      c1=(x(u,v,:)-mew);
      covariance(u,v,:,:)=[c1(1); c1(2); c1(3); c1(4)]*[c1(1) c1(2)
c1(3) c1(4)];
   end
end
covmatrs(:,:)=sum(sum(covariance))/(s(1)*s(2));
```

5. MATLAB code for car database

```
clear
clc


%SUV4
covmatr1 =[0.0106    0.0004    0.0108    0.0001
    0.0004    0.0031    0.0001    0.0028
    0.0108    0.0001    0.0175    0.0001
    0.0001    0.0028    0.0001    0.0048];

%SUV3
covmatr2 = [0.0082    0.0004    0.0073    0.0002
    0.0004    0.0031    0.0002    0.0027
    0.0073    0.0002    0.0105    0.0003
    0.0002    0.0027    0.0003    0.0047];

%SUV5
covmatr3 = [0.0070    0.0002    0.0069         0
    0.0002    0.0018         0    0.0014
    0.0069         0    0.0125    0.0001
         0    0.0014    0.0001    0.0029];

%SUV7
covmatr4 = [0.0054   -0.0001    0.0040   -0.0001
   -0.0001    0.0018   -0.0001    0.0014
    0.0040   -0.0001    0.0053         0
   -0.0001    0.0014         0    0.0019];

%SUV11
covmatr5 = [0.0045    0.0007    0.0039   -0.0001
    0.0007    0.0019    0.0003    0.0016
    0.0039    0.0003    0.0077    0.0002
   -0.0001    0.0016    0.0002    0.0031];

%test1
covmatr6 = [0.0701    0.0045    0.1027   -0.0029
    0.0045    0.0068    0.0076    0.0035
    0.1027    0.0076    0.2053   -0.0019
   -0.0029    0.0035   -0.0019    0.0069];

%test2
covmatr7 = [0.0124    0.0015    0.0155   -0.0008
    0.0015    0.0031    0.0017    0.0013
    0.0155    0.0017    0.0310   -0.0005
   -0.0008    0.0013   -0.0005    0.0026];
```

```
%test6
covmatr8 = [0.0333    0.0027    0.0437   -0.0001
    0.0027    0.0084    0.0021    0.0080
    0.0437    0.0021    0.0874   -0.0006
   -0.0001    0.0080   -0.0006    0.0159];

%test7
covmatr9 = [0.0116    0.0005    0.0106   -0.0007
    0.0005    0.0033   -0.0001    0.0017
    0.0106   -0.0001    0.0210   -0.0001
   -0.0007    0.0017   -0.0001    0.0033];

%test10
covmatr10 = [0.0107    0.0016    0.0080    0.0002
    0.0016    0.0024    0.0015    0.0010
    0.0080    0.0015    0.0160    0.0010
    0.0002    0.0010    0.0010    0.0020];

covmatr11 = [0.0039    0.0003    0.0039         0
    0.0003    0.0016         0    0.0016
    0.0039         0    0.0078         0
         0    0.0016         0    0.0033 ];

%sedan1
covmatrs1 = [0.0212    0.0019    0.0202   -0.0003
    0.0019    0.0069    0.0009    0.0053
    0.0202    0.0009    0.0402    0.0002
   -0.0003    0.0053    0.0002    0.0106 ];

%sedan2
covmatrs2 = [0.0188   -0.0001    0.0177         0
   -0.0001    0.0059   -0.0002    0.0054
    0.0177   -0.0002    0.0189    0.0001
         0    0.0054    0.0001    0.0108 ];

%sedan3
covmatrs3 = [0.0090    0.0002    0.0091   -0.0002
    0.0002    0.0010   -0.0001    0.0003
    0.0091   -0.0001    0.0180   -0.0001
   -0.0002    0.0003   -0.0001    0.0007 ];



%sedan4
covmatrs4 = [0.0117    0.0008    0.0084   -0.0002
    0.0008    0.0042    0.0001    0.0030
    0.0084    0.0001    0.0169    0.0001
   -0.0002    0.0030    0.0001    0.0060 ];
```

```
%sedan5
covmatrs5 =[0.0121         0    0.0107   -0.0004
        0    0.0032   -0.0002    0.0024
   0.0107   -0.0002    0.0161   -0.0002
  -0.0004    0.0024   -0.0002    0.0029];

%sedan6
covmatrs6 = [0.0145    0.0010    0.0124         0
   0.0010    0.0029    0.0005    0.0019
   0.0124    0.0005    0.0174    0.0004
        0    0.0019    0.0004    0.0037 ];

%sedan7
covmatrs7 =[0.0088    0.0005    0.0086   -0.0001
   0.0005    0.0024    0.0003    0.0018
   0.0086    0.0003    0.0172    0.0001
  -0.0001    0.0018    0.0001    0.0021];



save database
```

6. MATLAB code for human database

```
%human1
covmatrh1 = [0.0098    0.0050    0.0102    0.0052
   0.0050    0.0177    0.0025    0.0199
   0.0102    0.0025    0.0204    0.0040
   0.0052    0.0199    0.0040    0.0398 ];

%human2
covmatrh2 = [0.0042    0.0004    0.0040   -0.0003
   0.0004    0.0058   -0.0003    0.0049
   0.0040   -0.0003    0.0079   -0.0003
  -0.0003    0.0049   -0.0003    0.0097 ];

%human3
covmatrh3 = [0.0048    0.0001    0.0033   -0.0003
   0.0001    0.0042   -0.0003    0.0026
   0.0033   -0.0003    0.0065   -0.0001
  -0.0003    0.0026   -0.0001    0.0052];

%human4
covmatrh4 = [0.0016    0.0002    0.0015     0
   0.0002    0.0079    0    0.0075
   0.0015         0    0.0029         0
        0    0.0075         0    0.0114 ];

%human6
covmatrh6 = [0.0025    0.0006    0.0019    0.0002
   0.0006    0.0028         0    0.0024
   0.0019         0    0.0038    0.0001
   0.0002    0.0024    0.0001    0.0047 ];

%human7
covmatrh7 = [0.0033    0.0006    0.0025   -0.0001
   0.0006    0.0052    0.0001    0.0034
   0.0025    0.0001    0.0050    0.0002
  -0.0001    0.0034    0.0002    0.0067 ];

%human8
covmatrh8 = [0.0049    0.0017    0.0053    0.0014
   0.0017    0.0079    0.0007    0.0090
   0.0053    0.0007    0.0105    0.0008
   0.0014    0.0090    0.0008    0.0180 ];
```

```
%human9
covmatrh9 = [0.0092    0.0030    0.0113    0.0021
    0.0030    0.0128    0.0018    0.0145
    0.0113    0.0018    0.0226    0.0015
    0.0021    0.0145    0.0015    0.0291 ];

%human10
covmatrh10 = [0.0146    0.0010    0.0122    0.0003
    0.0010    0.0189   -0.0006    0.0146
    0.0122   -0.0006    0.0180    0.0002
    0.0003    0.0146    0.0002    0.0274 ];
```

References

O. Tuzel, F. Porikli, and P. Meer. Region covariance: A fast descriptor for detection and classification. In *Proc. 9th European Conf. on Computer Vision,* Graz, Austria, volume 2, pages 589–600, 2006.

O. Tuzel, F. Porikli, and P. Meer. "Covariance Tracking using Model Update Based on Means on Riemannian Manifolds.." *European Conf. on Computer Vision*. (2007)