Motion Filtering

Jiaxu Fu, Virginia Ruiz Albacete

Advisor: Prof. Konrad

May 3, 2010

Boston University

Department of Electrical and Computer Engineering

EC-720 Technical report

BOSTON
UNIVERSITY

Summary

This project presents a successful algorithm to filter out moving objects, based on their orientation and speed, based on sliding block matching and dynamic thresholding. Different experiments were carried out in order to prove its efficiency and an objective measure of its efficiency was computed.

Contents

List of Figures

List of Tables

1 Introduction

Since its origins, video recording has become an essential technology in different fields, such as surveillance or data capturing, due to its efficiency in analyzing different environments and situations. These video sequences require human check, whether they are used for security purposes or information analysis. As a result of dramatic decreases in the cost of both cameras and storage devices, the amount of video data stored has significantly grown, and with this growth comes the need to for automatic data analysis. Motion detection is no longer the optimal solution for general examination of a video. Thus, it is necessary to develop a technique that allows for the interpretation of a video or the filtering out of noteworthy data from a sequence without having to sift through the entire thing. This can also be valuable for compression purposes and data storage, where only the data of interest would be extracted and used.

2 Problem Definition

The goal is to develop an algorithm to filter out certain motions (direction, speed) in a video sequence. Thus, only objects moving with certain velocities or orientation would be extracted and displayed. The algorithm sequence can be seen in Figure 1.

2.1 Literature Review

While performing our initial research, we discovered a number of ways that motion filtering can be achieved. The most important ones can be summarized as follows:

- Object tracking:

Its goal is to detect a certain known object and follow its trajectory by finding its shape in consecutive frames. However, this method is not appropriate for the purpose of this study because the goal is to filter out any object moving in certain directions.

- Frequency Domain:

One of the implementations that uses this approach of motion detection is described in Heeger's paper [2] based on Gabor Filters in the spatiotemporal-frequency domain. It shows how the power spectrum of a moving signal occupies a plane that can be filtered out to extract different motions. A discussion on this technique can be found in Loya Zorn and Amarnat's EC720 Final Project.

- Optical Flow:

Optical flow can be defined as the pattern of apparent motion of objects caused by the relative motion between an observer and the scene. Although this idea does not fully solve our problem, it will be the inspiration of our project [3].

2.2 Assumptions

In order to simplify the problem, some assumptions must be taken into consideration:

- The objective is to obtain two filters: one will segment the images by orientation and the other one by relative velocity. In this project, the filters will be applied separately.

- Only three different motion orientations will be taken into account. For each image sequence, the horizontal, vertical and diagonal movements will be extracted. This means that orientations are normalized between [0º, 90º]; for example, no distinction will be made between a car moving from left to right versus right to left; they will both be considered to be moving horizontally.

- When talking about speed segmentation, no absolute value will be computed. The velocity filtering will be relative to the different motions in the image. Therefore, objects will be extracted based on whether they are faster or slower than one another.

- The algorithm will be used to analyze mainly street scenes. This assumption will be used in order to improve the method, based on the previous information that these videos contain (straight simple movement). It can be useful to separate different flows of traffic that occur in an intersection in order to analyze a highway and a secondary road separately.

- Two different cameras will be used to test the algorithm:

1. ISS network cameras: Used for the horizontal and diagonal sequences. 250 frames will be captured and processed from these cameras. Each picture has a resolution of 240 x 320 pixels. These cameras present some inconveniences such as missed frames (which will cause the image to jump) or hard shakes when zooming to far-away locations.

2. Sony DSC-H1 camera: Camera used to record complex scenes (such as intersections where more than one movement is present). Since the resolution of this camera is of 480 x 640 pixels, a smaller number of frames were captured.

3 Methodology and Implementation

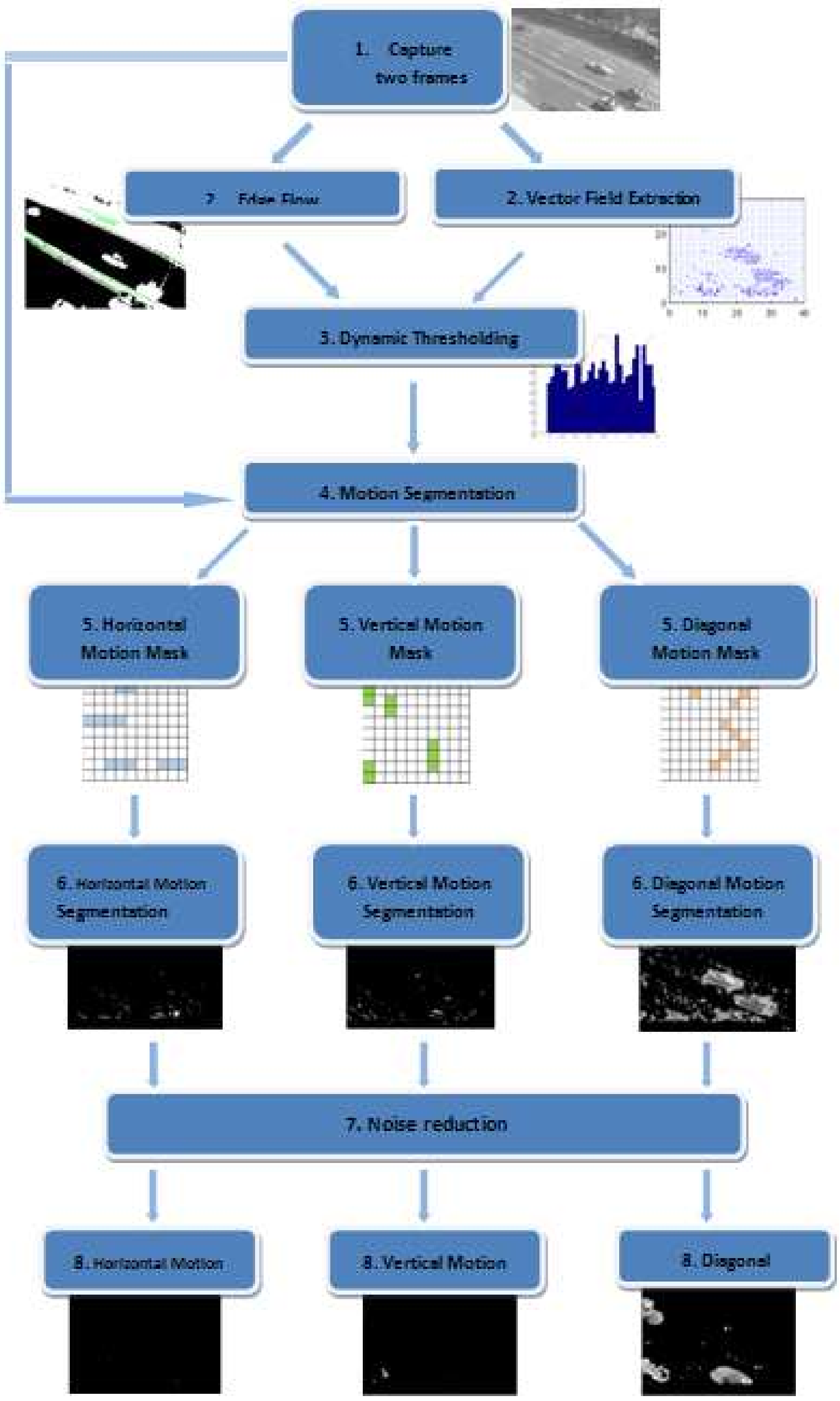
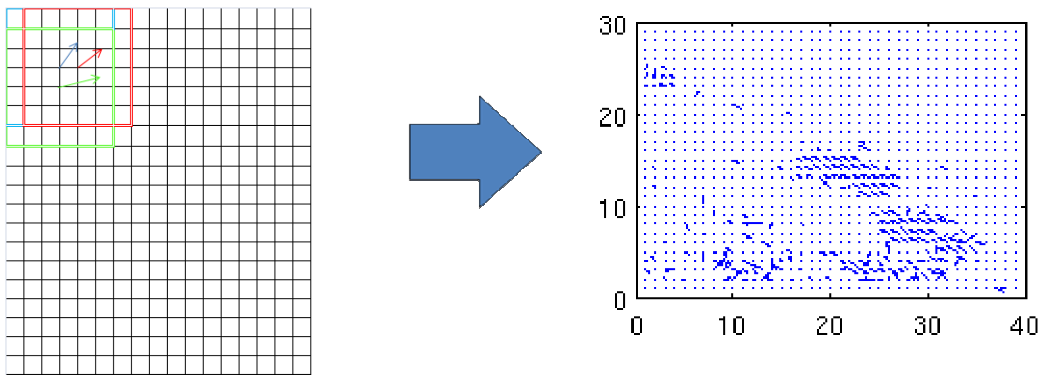


Figure 1: Algorithm Flow Chart

3.1 Vector Field Extraction

The method developed in this project is based on vector field motion estimation. To accurately describe the displacement of moving objects, it is necessary to assign a motion vector to every pixel in the image. This is achieved through the sliding block method (Figure 2), where the image is divided into blocks with fixed sizes and a matching backward block is applied to it. The computed displacement vector will be assigned to the center pixel of the block. Then this block will be moved over by one pixel to repeat the same process.



od

The main idea behind block matching is to create a block big enough to avoid large, flat, constant areas that would disrupt its tracking in successive frames, but small enough to avoid containing more than one object (such as two cars for example). The compromise between these two objectives has been performed by manual inspection. A different block size has been assigned for each video sequence. A similar process has been taken into consideration for the search window since some of the motions present more displacement than others (e.g. far away areas seem to move slower than closer movements).

3.2 Background Edge Flow

As previously mentioned, the main goal of this algorithm is to analyze street videos. These scenes contain some information that can improve the performance of the algorithm. For example, it would be possible for someone to tell the direction in which cars will move without seeing any object passing by simply by looking at the orientation of the lines of the lanes or the guard rails. To make use of this information, a background edge flow detection algorithm has been implemented. After converting a frame into a white and black image and applying some morphological operations to it, the intensity gradient is computed. This essentially captures the rate of change. Thus, in the ideal case, detection of zero-crossings in the second derivative captures local maxima in the gradient, as it can be seen in Figure 3.

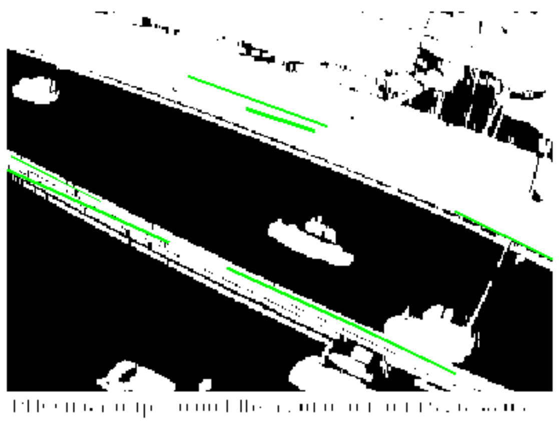


Figure 3: Background Edge Flow

This step will return an angle with the strongest orientation of the background edge flow. In the case that this orientation is not strong enough or doesn't exist (e.g. in an intersection), no specific angle will be obtained.

3.3 Dynamic Threshold

Based on the vector field and the background edge flow angle obtained from the previous steps, our goal is to separate the motions captured in a video in three different directions: vertical, horizontal and diagonal.

Since there is a vector defining each pixel in the image, it is possible to obtain the angle Θ of each moving pixel through basic trigonometry:

$$\Theta = \mathrm{arctg}\ (Y/X)$$

Where Y is the vertical displacement and X is the horizontal displacement.

In order to classify the motions, it is necessary to set a threshold angle that will decide which motion falls under which orientation. If a fixed threshold is used, a manual inspection of the images will be needed in order to adjust this parameter to different sequences. However, our goal is to create an automatic algorithm to filter out the motion. This can be achieved by using the previous information obtained through the background edge flow.

The solution implemented in this project uses Gaussian distributions as thresholds. Gaussian functions (Equation 1) can be defined by 2 parameters: the mean and the variance. Three different Gaussian distributions are created to represent a horizontal, diagonal and vertical threshold. These distributions will be centered at 0°, 45° and 90°, respectively (through their means), and their variances will depend on the background edge flow, where the previous knowledge will be applied.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Equation 1: Gaussian Function

If we represented all the angles of the vector field in a histogram like the one showed in Figure 4, when applying the three Gaussian functions, the angles will be scaled by each of them, producing three weighted values. The highest of these values will determine which orientation the angle falls under (if the highest value is the result of the scaling by the horizontal Gaussian, then the angle will be classified as horizontal, and so on). Assume that no previous information is obtained, that is, either there is not a strong edge flow orientation or no background analysis method has been applied. In this case, the three Gaussians will have the same variance, and they will act as fixed thresholds, separating the angles at 30º and 60º.

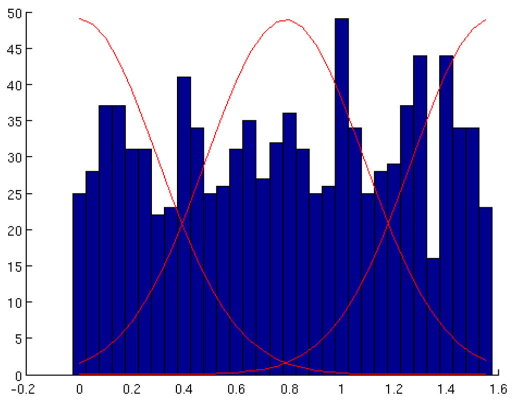


Figure 4: Dynamic Thresholding without Edge Flow Information

However, if the background edge has a significant value for a specific angle, for example, it has a higher horizontal component, it seems reasonable to allow the variation of the Gaussian distribution centered at 0º (horizontal) to be greater than the variations of

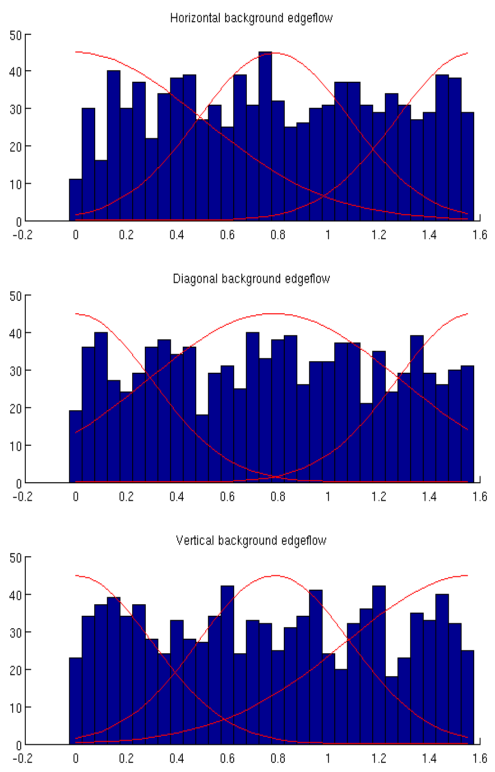
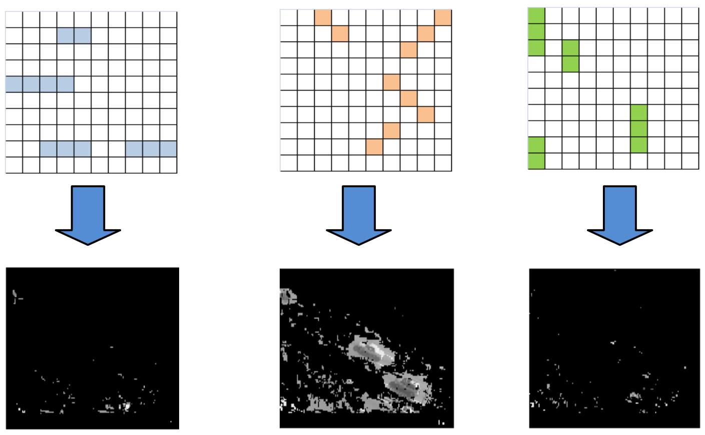


Figure 5: Dynamic Thresholding using Background Edge Flow

the other two Gaussians. This will have no impact for the pixels close to the Gaussian means, since their highest value will be given when being scaled by the closer Gaussian. However, more of the intermediate angles will fall under the Gaussian with a larger variance. In Figure 5 this can be seen as raising the threshold of one of the functions. For the first graph, the horizontal Gaussian (centered at 0º) has a larger variance than the other two. Thus, the crossing point with the diagonal Gaussian (centered at 45º) is greater than the crossing point of the two functions for fixed threshold. This means that all the pixels that fall under this function before the crossing point will be considered as horizontal, because their greater value will be obtained when scaling them by this Gaussian. This remains true for the other orientations as well.

3.4 Motion Filtering

Once the moving pixels have been assigned a certain motion, they must be displayed separately. To do so, three different masks are created: horizontal, diagonal and vertical. By applying these masks to the original frame, one can successfully extract the moving pixels of the image. As it can be seen in Figure 6, for each frame, three different images will be obtained containing the horizontally, diagonally and vertically moving pixels. Successive motion-extracted frames will form a video of the filtered objects.



ion Filtering

3.5 Noise Reduction

The use of sliding block matching methods to extract the vector field can lead to mismatches and other errors that produce noise at the output of the algorithm. However, these incorrect vector fields can be considered as uncorrelated noise (noise and pepper) and can therefore be reduced by applying a median filter. Although the filtered image will be smoothed out, the results will be improved, as it can be seen in Figure 7.

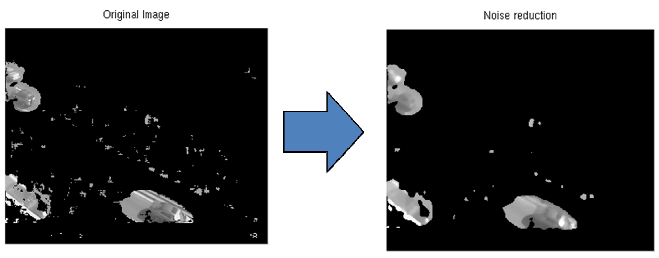


Figure 7: Results after Noise Reduction

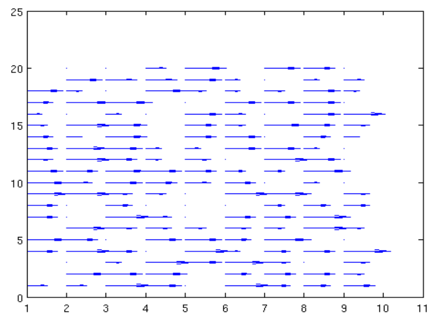
4 Experimental Results

In order to show our method works, three experiments are conducted: apply the method to a simulated ideal vector field; test it on frame pairs captured by cameras; and apply the algorithm to different video sequences.

4.1 Synthetic Results: Experiments on Simulated Ideal Vector Field

Five ideal vector fields are generated to prove whether our method can extract motions along horizontal, vertical and diagonal directions, respectively. The results will be displayed in three windows, each showing the extracted horizontal, diagonal and vertical extracted pixels. A moving pixel will create a white dot in the corresponding output. Black pixels correspond to non-moving areas for that orientation. Results are shown from Figures 8 – 12.

(a)



(b)

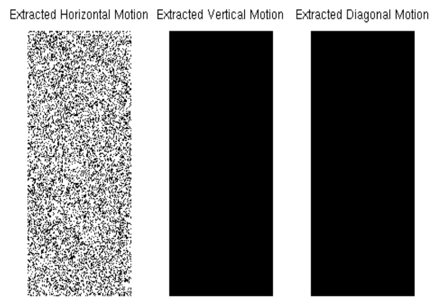
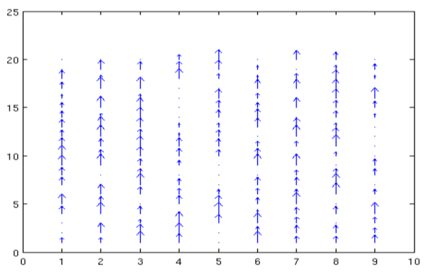

Extracted Horizontal Motion
Extracted Vertical Motion
Extracted Diagonal Motion

Figure 8: (a): Ideal Horizontal Vector Field (b): Extracted Horizontal, Vertical and Diagonal Motion

(a)



(b)

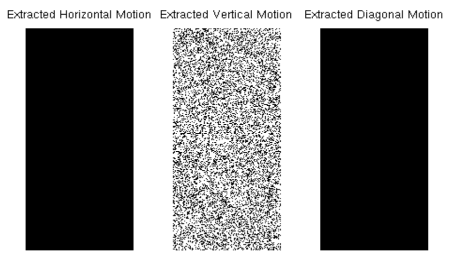
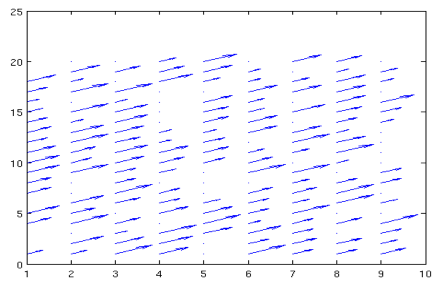
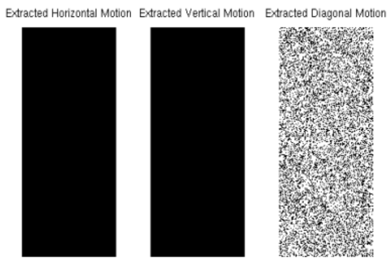

Extracted Horizontal Motion
Extracted Vertical Motion
Extracted Diagonal Motion

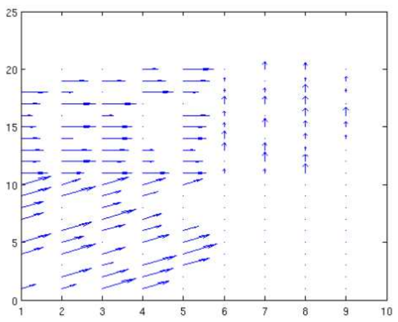
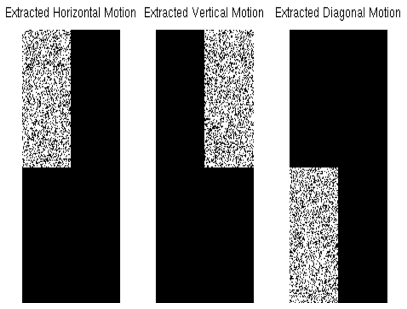
Figure 9: (a): Ideal Vertical Vector Field (b): Extracted Horizontal, Vertical and Diagonal Motion

(a) (b)

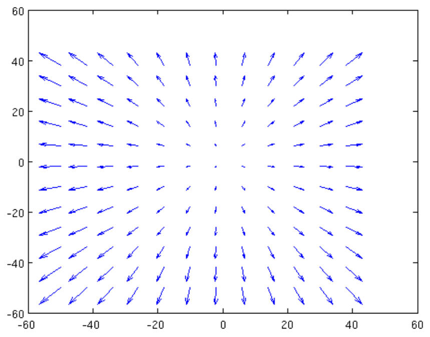
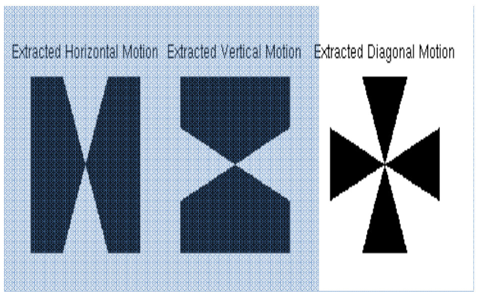
and Diagonal Motion

(a) (b)

al, Vertical and Diagonal

(a) (b)

nd Diagonal Motion

4.2 Image comparison: Experiment on image pairs captured by cameras.

The procedure explained above has been implemented and tested in two different pairs of images: one with mainly diagonal components and another with mainly horizontal components. When trying to explore the vertical motion, we realized that a helicopter view would be required to record it; frontally captured image sequences would involve an object size change when moving toward or away from the camera that would make the block matching method fail. To prove the efficiency of this method, the results obtained by applying the background information are compared to those obtained without its application.

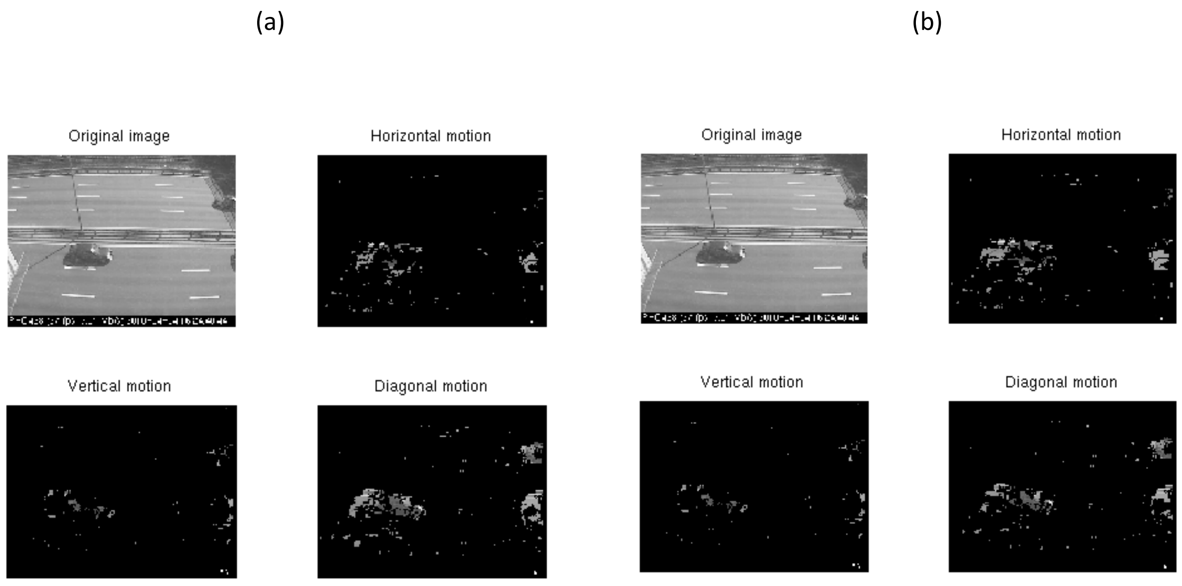


Figure 13: Horizontal video filtered with (a) fixed thresholding (b) dynamic thresholding

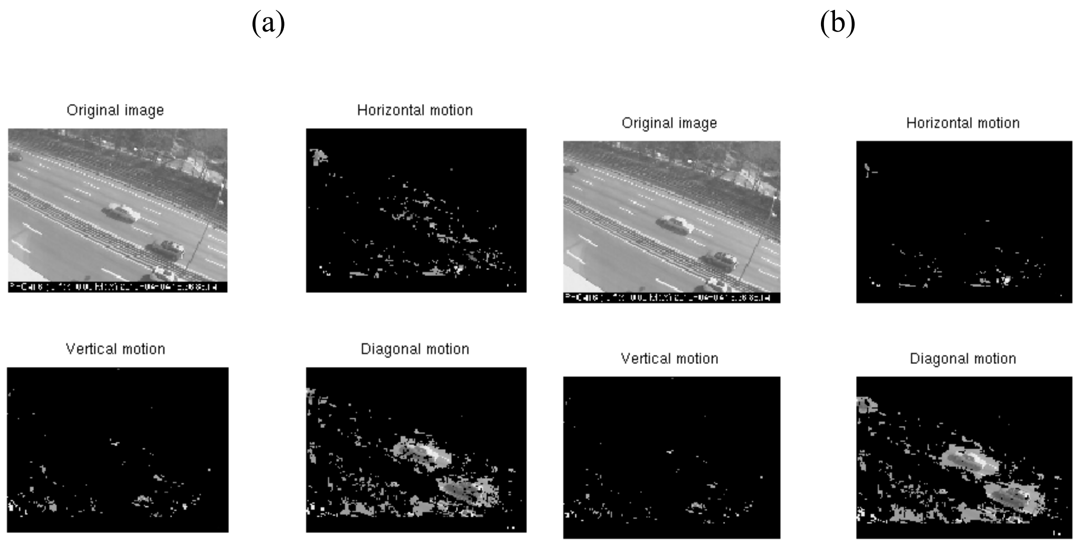


Figure 14: Diagonal video filtered with (a) fixed thresholding (b) dynamic thresholding

As we can see from the results in Figures 13 and 14, the background orientation implementation affects to the results positively. The incorrect assignation of moving pixels is mainly due to erroneous vector field extraction, and not dubious vectors.

4.3 Video Results: Experiment on image sequences

In the stage of experiments, our motion filter is applied for three image sequences which contain horizontal motion, diagonal motion and complex motion respectively. And the results show that our algorithm can successfully extract motions alone in three directions.

4.4 Objective Measure

When dealing with images, the subjective impression of the results is the main concern. However, an objective measure is also needed in order to analyze and compare the performance of this algorithm. Since to obtain a true answer for this problem, a manual adjustment of all the image sequences would be needed, another measure is developed. Assuming a horizontal image sequence, the percentage of pixels that are filtered out as horizontally moving is computed. A comparison with the other percentages (% vertical and % diagonal moving pixels) will estimate the behavior of this method.

Image Sequence	*Horizontal moving pixels (%)*	*Vertical moving pixels (%)*	*Diagonal moving pixels (%)*
Horizontal Road	63.2047	20.4748	16.3205
Diagonal Road	2.1460	5.1495	92.4345
Intersection Road	9.4373	30.0656	60.4917

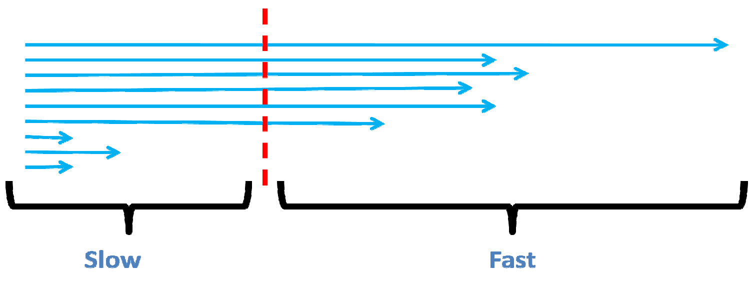
Table 1: Percentage of moving pixels per category

As it can be seen in Table 1, the best performance is given for the diagonal road where over 92% of the pixels are correctly classified. The horizontal road has a higher error due to optical effects of the recorded data (the camera deviates the image slightly on the edge, creating a diagonal effect). The intersection road simulates a more complex scenario, where two different motions can be seen: horizontal and diagonal. The most abundant one is the diagonal direction. However, the high percentage of vertically moving pixels is due to the shaking of the camera and the zoom.

5 Speed Filtering

5.1 Methodology

The second part of this project focuses on filtering the images in a video according to different speeds. The process followed for this task is basically the same as the one used to filter out orientations, but this time the length of the vectors will be the main discriminator of the process.



·esholding

In order to generalize the method for any set of images, the filtered speed is going to be relative to the other speeds in the image avoiding once more a fixed threshold. Therefore, the threshold is computed as half the length of the average vector obtained from the 4 longer vectors of the vector field (Figure 15), without taking into account the largest one. With this measure we aim to obtain some resiliency against errors, since a long mismatched vector could derive to wrong speed filtering.

5.2 Experimental Results

As explained for orientation filtering, a previous analysis of our method was carried out using ideal vector fields.

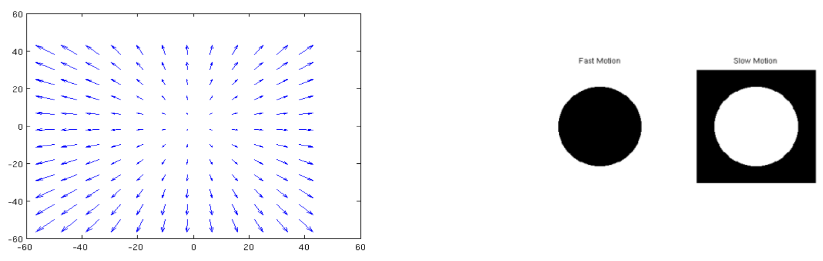


Figure 16: (a) Ideal Flower-Like Vector Field (b): Extracted Fast and Slow Motion

In Figure 16, the successful results of this algorithm can be seen: when extracting fast motions, the vectors with vector of length greater than the threshold will have an output of one (white) and vice versa for slow motions. However, when applying this method to real video sequences, the outcome is acceptable only for videos with much differentiated speed objects. This is due to the fact that the block matching method used to obtain the vector field performs poorly for uniform moving areas. While the orientation of the displacement vectors may be more or less accurate, they can point at any pixel in the uniform area. Therefore, for some video sequences the fast filter will show the outline of the cars, while the slow filter will obtain the inside of the objects.

6 Conclusions and Future Work

This project presents a successful algorithm to filter out moving objects, based on their orientation and speed, based on sliding block matching and dynamic thresholding. Different experiments were carried out in order to prove its efficiency and an objective measure of its efficiency was computed.

After these experiments, the sliding block matching method has been found to perform acceptably well in obtaining the pixels' orientation, but poorly in uniform moving areas. Another source of outcome noise is due to mismatching and luminance and focal lens changes.

For future work, it would be necessary to improve video segmentation based on speed, with morphological techniques or object tracking methods that would mitigate the mismatching produced through sliding block matching.

Also, a simultaneous orientation and speed filtering would be desirable, in order to simplify security tasks and others.

Apply simultaneous [orientation and speed] segmentation.

7 Appendix: Matlab Codes

```
% Project EC720
%main

close all
clear all
clc

%load video_hor.mat
%load video_diag.mat
%load video_road36.mat % Vertical
load video_road34.mat % intersection

%video = video_hor;
%video = video_diag;
%video = video_road36;
video = video_road34;


%% Sizes
% Diagonal: BS = 22; WS = 20; NF = 251;
% Horizontal: BS = 30; WS = 22; NF = 251;
% Vertical: BS = 45; WS = 15; NF = 143
% Intersection: BS = 22; WS = 15; NF = 170

BS = 22; % Block Size
WS = 15; % Window size (#pixels we are going to move up&down)
%INC = BS; % Disjoint block matching
INC = 1; % Sliding block matching

NF = 169; %Number frames


%% Compute the Vector field of the frame
```

```
a = double(rgb2gray(video.frames(1).cdata));
[ma,na]= size(a);


% Structure to save results
Frames = struct('VF', zeros(ma,na,2),'Horizontal',zeros(ma,na), 'Vertical',zeros(ma,na), 'Diagonal',
zeros(ma,na));
Frames(1,1:NF) = Frames;


for i = 1:NF
   b = double(rgb2gray(video.frames(i).cdata));

   [d_sq, e_sq] = VectorField2 (a, b, BS, WS, INC);


   % Obtain frame's motion
   [H_frame, V_frame, D_frame] = MotionSepPlot (b, d_sq, BS);

   % Save values in a struct
   Frames(1,i).VF = d_sq;
   Frames(1,i).Horizontal = H_frame;
   Frames(1,i).Vertical = V_frame;
   Frames(1,i).Diagonal = D_frame;

   a = b;
end

%save 'FramesHorizontal.mat' Framessave 'FramesIntersection.mat' Frames
```

function [d_sq, e_sq] = VectorField2 (a, b, BS, WS, INC)

```
% load video_hor.mat
% a = double(rgb2gray(video_hor.frames(1).cdata));
% b = double(rgb2gray(video_hor.frames(3).cdata));
% BS = 16; % Block Size
% WS = 20; % Window size (#pixels we are going to move up&down)


[ma,na]= size(a);


% To avoid boundary problems, we create a boundary of "infinite" (255)
% values
%brep = padarray(b,[1 1], 'replicate','post');
bp = padarray(b,[WS WS],inf);
% [mbp nbp]= size(bp);

% displacement matrix: [y,x,i], where x & y are coord of the block being
% evaluated and i=1 is the amount of displacemente in y direction and i=2
% the displacement on x direction
d_sq = zeros(ma,na,2);


for k = 1:INC:ma-(BS-1)
  for q = 1:INC:na-(BS-1)
    % Coord for reference image
    error_sq1 = inf;
    af = a(k:k+(BS-1),q:q+(BS-1)); % selection of block

    for i = k:k+(2*WS-1) % We move a WS to both sides through the PADDED image
      for j = q:q+(2*WS-1)
        % Coord for search image
        bf=bp(i:i+(BS-1), j:j+(BS-1));
        error_sq2 = sum(sum((af-bf).^2));
        if error_sq2 < error_sq1
          % Gives DISPLACEMENT no new coords
          d_sq(k,q,1) = i-(k+WS); % The displacement characterizes the middle point of the
block
          d_sq(k,q,2) = j-(q+WS);
          error_sq1 = error_sq2;
        end
```

```
            end
        end
    end
end
%save 'd_sqHor.mat' d_sq


% Create the displaced image by the sq error
Im_sq = zeros(ma,na);
for i = 1:BS:(ma-BS+1)
    for j = 1:BS:(na-BS+1)
        Im_sq(i:(i+BS-1),j:(j+BS-1)) = b((i+d_sq(i,j,1)):(i+d_sq(i,j,1)+BS-
1),(j+d_sq(i,j,2)):(j+d_sq(i,j,2)+BS-1));
    end
end
e_sq = b-Im_sq;
```

%function OrExtration

```
% EC720 PROJECT
% orientation.m
% Input: Original image, OrMatrix
% Output:


a = double(rgb2gray(video.frames(1).cdata));
b = double(rgb2gray(video.frames(3).cdata));

d = d_sq;

OrMatrix = orientation (d);
[mOM,nOM] = size(OrMatrix);

% Masks for different motions
```

```
Hor = zeros(mOM, nOM);
Hor(find(OrMatrix) == 0) = 1;

Ver = zeros(mOM, nOM);
Ver(find(OrMatrix) == 1) = 1;


Diag = zeros(mOM, nOM);
Diag(find(OrMatrix) == 2) = 1;

% Obtaining the moving regions of the first frame
HorMotion = b.*Hor;
VerMotion = b.*Ver;
DiagMotion = b.*Diag;

figure
imshow(HorMotion,[]);


% Display the moving "blocks" by the displacement vector
```

function OrMatrix = FrameOr (d,im)

```
% EC720 PROJECT
% orientation.m
% Input: d= displacement
% Output: orientation Matrix

% displacement matrix: [y,x,i], where x & y are coord of the block being
% evaluated and i=1 is the amount of displacemente in y direction and i=2
% the displacement on x direction
%d = d_sq;
[md, nd, kd] = size(d);


% Multiplication by a very small number to avoid NaN
s = ones(md,nd);
```

```
s = s.*eps;
d(:,:,2) = d(:,:,2)+s;
theta = atan(d(:,:,1)./d(:,:,2));

[mz,nz] = size(theta);

% Reshape: Matrix -> vector
vectorTheta = abs(reshape(theta,1,mz*nz));

angle = BackOr(im);

 [g_hor, g_ver, g_diag] = GaussFilter(vectorTheta, angle);
% Gaussian functions at each point

% Scaling
horMult = g_hor.*vectorTheta;
verMult = g_ver.*vectorTheta;
diagMult = g_diag.*vectorTheta;

OrientationTheta = zeros(1,mz*nz);

for i = 1:mz*nz
        temp = max(horMult(i), verMult(i));
        maximum = max(temp, diagMult(i));

        if maximum ~= 0 && maximum == diagMult(i)
            OrientationTheta(i) = 3; % 3 = flag for diagonal movements
        elseif maximum ~= 0 && maximum == verMult(i)
            OrientationTheta(i) = 2; % 2 = flag for vertical movements
        elseif  maximum ~= 0
            OrientationTheta(i) = 1; % 1 = flag for horizontal movements
        end
end

OrMatrix = reshape(OrientationTheta,mz,nz);
```

```
function [g_hor, g_ver, g_diag] = GaussFilter(Theta, bck_Or)
% EC720 PROJECT
% gauss_funct.m
% Input: Th = Theta
% Output: 3 gaussian functions

Mhor = 0;
Mver = pi/2;
Mdiag = pi/4;
sigma1 = 0.6;
sigma2 = 0.3;

if bck_Or < 30 % horizontal
   sigmaX = sigma1;
   sigmaY = sigma2;
   sigmaD = sigma2;

elseif bck_Or > 60 % vertical
   sigmaX = sigma2;
   sigmaY = sigma1;
   sigmaD = sigma2;

else  % diagonal
   sigmaX = sigma2;
   sigmaY = sigma2;
   sigmaD = sigma1;
end



%ang = (0:0.01:pi);


g_hor = (1/(sqrt(2*pi*0.5)))*exp(-((Theta-Mhor).^2)./(2*sigmaX^2));
g_ver = (1/(sqrt(2*pi*0.5)))*exp(-((Theta-Mver).^2)./(2*sigmaY^2));
g_diag = (1/(sqrt(2*pi*0.5)))*exp(-((Theta-Mdiag).^2)./(2*sigmaD^2));
```

```
function [Fast_frame, Slow_frame] = SpeedSeg (b, d, BS)
% EC720 PROJECT
% Speed Segmentation
% Input: b = input frame; d = vector field; BS= Block Size
% Output: [Fast_frame, Slow_frame, D_frame] = motion from diferent directions


%b = double(rgb2gray(video_hor.frames(1).cdata));
[ma,na] = size(b);


%d = d_sq;
%BS = 16;

% number of samples used to compute the threshold
samp = 4;

length = sqrt(d(:,:,2).^2+ d(:,:,1).^2);

% Find the average "long" length
a = reshape(length, 1, ma*na);
a = sort(a,2,'descend');

average = sum(a(2:samp+1))/samp;
average = average/2;

% Create a mask on the image
maskFast = zeros(ma,na);
temp1 = zeros(ma,na);
temp2 = zeros(ma,na);
%maskSlow = zeros(ma,na);

maskFast((length>average)==1) = 1;
temp1((length <= average)==1) = 1;
temp2((length > 0)==1) = 1;
maskSlow = temp1.*temp2;
```

```
% Extract the fast/slow moving areas
Fast_frame = maskFast.*b;
Slow_frame = maskSlow.*b;
```

function angle = BackOr (I)

```
% EC720 PROJECT
% or_background.m
% Input: w = image
% Output: angle of main orientation

% read the image into MATLAB and convert it to grayscale
%I = video_diag.frames(1).cdata;
%Igray = rgb2gray(I);
Igray = uint8(I);
%figure, imshow(I,[]);

% We can see that the image is noisy. We will clean it up with a few
% morphological operations
Ibw = im2bw(Igray,graythresh(Igray));
se = strel('line',3,90);
cleanI = imdilate(~Ibw,se);
%figure, imshow(cleanI,[]);

% Perform a Hough Transform on the image
% The Hough Transform identifies lines in an image
[H,theta,rho] = hough(cleanI);
peaks = houghpeaks(H,10);
lines = houghlines(Ibw,theta,rho,peaks);
%figure, imshow(cleanI,[])

% Transform
result = 0;
for k = 1:numel(lines)
```

```
    result = result + abs(lines(k).theta);
end

angle = 90 - abs(result/numel(lines));


function [H_frame, V_frame, D_frame] = MotionSepPlot (b, d, BS)
% EC720 PROJECT
% orientation.m
% Input: a = input frame; d = vector field; BS= Block Size
% Output: [H_frame, V_frame, D_frame] = motion from diferent directions

% load video_hor.mat
% load d_sqHor.mat
%
% a = double(rgb2gray(video_hor.frames(1).cdata));
% b = double(rgb2gray(video_hor.frames(3).cdata));
%
% d = d_sq;
% BS = 16;

OrMatrix = FrameOr (d,b);
[mOM,nOM] = size(OrMatrix);

% Masks for different motions
Hor = zeros(mOM, nOM);
Hor(find((OrMatrix) == 1))= 1;

Ver = zeros(mOM, nOM);
Ver(find((OrMatrix) == 2)) = 1;



Diag = zeros(mOM, nOM);
Diag(find((OrMatrix) == 3)) = 1;

% Obtaining the moving regions of the second (moved) frame
HorMotion = b.*Hor;
```

```
VerMotion = b.*Ver;
DiagMotion = b.*Diag;

[mh,nh] = find(HorMotion ~= 0);
[mv,nv] = find(VerMotion ~= 0);
[md,nd] = find(DiagMotion ~= 0);


% Project EC720
% Median Filter

close all
%clear all
clc

% %load video_hor.mat
%load video_diag.mat
% %load video_road36.mat % Vertical
% load video_road34.mat % intersection
video = video_hor;



%load FramesHor.mat
f = FramesHor;



% %% Sizes
% % Diagonal: BS = 22; WS = 20; NF = 251;
% % Horizontal: BS = 30; WS = 22; NF = 251;
% % Vertical: BS = 45; WS = 15; NF = 143
% % Intersection: BS = 22; WS = 15; NF = 170

NF = 250; %Number frames
%
%
%% Compute the Vector field of the frame
a = double(rgb2gray(video.frames(1).cdata));
```

```
[ma,na]= size(a);


%load FramesDiag.mat

% Structure to save results
FramesClean = struct('VF', zeros(ma,na,2),'Horizontal',zeros(ma,na), 'Vertical',zeros(ma,na),
'Diagonal', zeros(ma,na));
FramesClean(1,1:NF) = FramesClean;


for i = 1:NF

  % Save values in a struct
  FramesClean(1,i).VF = f(i).VF;
  FramesClean(1,i).Horizontal = medfilt2(f(i).Horizontal, [3 3]);
  FramesClean(1,i).Vertical = medfilt2(f(i).Vertical, [4 4]);
  FramesClean(1,i).Diagonal = medfilt2(f(i).Diagonal, [5 5]);

  %a = b;
end

%save 'FramesHorizontal.mat' Frames
save 'FramesCleanDiagonal.mat' FramesClean
%save 'FramesVertical.mat' Frames
%save 'FramesIntersection.mat' FramesClean



for i = 1:NF

  imshow(FramesClean(1,i).Horizontal,[]); title('Clean Diagonal motion');

  MovDiagClean(i)=getframe;

end
```

```
for i = 1:NF

   imshow(FramesHor(1,i).Horizontal,[]); title('Diagonal motion');

   MovDiag(i)=getframe;

end
```

8 References

[1] Al Bovik, “The Essential Guide to Video Processing”, *Elsevier*, 2009 pp. 141-220

[2] David J. Heeger, “Optical Flow Using Spatio-temporal Filters”, *International Journal of Computer Vision,* vol. 1, Number4, pp.279-302, 1998.

[3] Andés Bruhn & Joachim Weickert, “Lucas/Kanade Meets Horn/Schunck: Combining Local and Optic Flow Methods”, *International Journal of Computer Vision,* vol. 62, Number3, pp.211-231, 2005.

[4] http://www.alphatecltd.com/video/pdf/MotionEstimation.pdf

[5]
http://lcni.uoregon.edu/~mark/Geek_software/Video_motion/Video_motion_estimation.html#video_velocity_distribution_estimation