CompE565, Fall 2021
Homework 3: Motion Estimation for Video Compression

Prepared by

Elena Pérez-Ródenas Martínez

E-mail: eperezrodenasm3836@sdsu.edu

Electrical and Computer Engineering

San Diego State University

Contents

Page

List of Figures **1**

1 Introduction **2**

2 Procedural Section **2**

3 Results **6**

4 Conclusion **11**

5 References **11**

List of Figures

1 I-Frame 6
2 Motion vector for image 7 and 8 7
3 Y component of the original video frame 7, predicted video frame 8 and error frame 7
4 Motion vector for image 8 and 9 8
5 Y component of the original video frame 8, predicted video frame 9 and error frame 8
6 Motion vector for image 9 and 10 9
7 Y component of the original video frame 9, predicted video frame 10 and error frame 9
8 Motion vector for image 10 and 11 10
9 Y component of the original video frame 10, predicted video frame 11 and error frame 10

1 Introduction

In this assignment we will learn the Motion Estimation technique for reducing the temporal redundancy in video sequences. The goal of Motion Estimation is to estimate motion in a moving image sequence.

Just as we saw in the previous assignment, JPEG compression exploits spatial redundancy. Video compression does that as well, but it can also exploit temporal redundancy. When the motion is slow there is a lot of temporal correlation and using prediction techniques can be very useful in order to save space for either storage and transmission. In short videos this is not as important but when the videos are long we could compress them a lot and make them more efficient.

The process consists of finding corresponding points between two images or video frames creating a motion vector. The points that correspond to each other in two images or frames of a real scene or object are usually the same point in that scene or on that object. So with the reference frame and the motion vectors we can recreate the new frame with no need to store it.

In the procedural section we will use Matlab to implement the full technique with a search window of 32x32 pixels, five frames from #6 to #10 and 4:2:0 format. We will use SAD method and for each frame we will generate the difference frame and the reconstructed one.

2 Procedural Section

First of all we will read the video using the correct path and save its width and height.

```
video = VideoReader('C:\Users\Elena Pérez-Ródenas\Desktop\SDSU\MULTIMEDIA COMMUNICATION
SYSTEMS\HW3\walk_qcif.avi');

width = video.Width;
height = video.Height;
```

Then we will convert every frame in the video to YCbCr format 4:2:0. In order to do that we will read every frame, transform it and then apply the subsampling. The variable i will count the number of frames and j will be the current one so the third coordinate will show the number of the current frame in each component. [1]

```
i=0;
j=1;
while hasFrame(video)
    videoFrame(j).cdata = readFrame(video);
    Frame_YCbCr = rgb2ycbcr(videoFrame(j).cdata(:,:,:));
    Y_Comp(:,:,j) = Frame_YCbCr(:,:,1);
    Cb_Comp(:,:,j) = Frame_YCbCr(1:2:end,1:2:end,2);
    Cr_Comp(:,:,j) = Frame_YCbCr(1:2:end,1:2:end,3);
    i = i+1;
    j = j+1;
end
```

The GOP (Group of Pictures) is a collection of successive pictures within a coded video stream. [2] Each one has a I-frame that is coded independently of all other pictures. We will access the I-frame of the GoP(6:10) and plot it.

```
I_Frame = Y_Comp(:,:,6);
imshow(I_Frame)
title('I-Frame')
```

We will calculate the total number of blocks in an image to later initialize the search window and show the motion vectors.

```
total_MB = (width/16)*(height/16)
fprintf("The total number of blocks in the image is:%d\n",total_MB);
```

We will do a loop for going through the 4 following frames and get the predicted and error frame as well as the MSE between them. First, we will save #6.

```
for i = 7:10
    Y_Comp(:,:,i-1) = Frame_YCbCr(:,:,1);
    Cb_Comp(:,:,i-1) = Frame_YCbCr(1:2:end,1:2:end,2);
    Cr_Comp(:,:,i-1) = Frame_YCbCr(1:2:end,1:2:end,3);
```

Now we will save the original i Frame and the reference as the i+1 frame. We will set the macroblock size to 16 pixels as it is asked and initialize the search window.

```
    originalFrame = Y_Comp(:,:,i);
    referenceFrame = Y_Comp(:,:,i+1);
    MBSize = 16;
    searchWindow = zeros(total_MB,2,2);
```

Now it's time to call the function MotionEsimation which takes as parameters the original and reference pictures and the size of the macroblock and returns the search window used, the new reconstructed image, the MSE between the frames and the total number of comparisions.

```
    [searchWindow,reconstructedImage, MSE,totalComparision] =
        MotionEstimation(originalFrame,referenceFrame,MBSize);
```

Making use of the search window we will plot the motion vectors which represent the displacement of the first pixel of the current macroblock with respect to the first pixel of the macroblock in the reference frame. The first coordinate of the search window represents which block are we refering to (first, second...), the second one is the coordinate and the third one if its the position or the velocity (x or mvx). [3]

```
    figure()
    quiver(searchWindow(:,2,1),searchWindow(:,1,1),searchWindow(:,2,2),
        searchWindow(:,1,2));
    title(['Motion vector for image ',num2str(i),' and ',num2str(i+1)]);
```

We will print the MSE between every pair of consecutive frames.

```
fprintf("MSE between frame %d and %d is %d\n",i,i+1,MSE);
```

Finally we will plot the original frame, the reconstructed one and the error frame. It is important to bear in mind that for the reconstructed image we will need to use the command uint8 to convert the image to a uint8 matrix and be able to plot it.

```
figure()
subplot(2,2,1),imshow(originalFrame),title(['Y Component of Original Video Frame:'
    ,num2str(i)])

reconstructedImage = uint8(reconstructedImage);
subplot(2,2,2),imshow(reconstructedImage),title(['Predicted video frame'
    ,num2str(i+1)])

errorFrame = originalFrame - reconstructedImage;
subplot(2,2,3),imshow(errorFrame),title('Error between original frame and
    predicted frame')

end
```

In addition to this we will compute the total additions and comparisons. This will be explained later in the section *Computational load*.

```
additionsSAD = 2*(MBSize*MBSize)-1;
w = 1;
posValues = (2*w+1)^2;
add = posValues*additionsSAD;
fprintf("\nTotal comparisons:%d\n",totalComparisons);
fprintf("\nTotal additions:%d",add);
```

Motion Estimation function

The function receives as a parameter the original and reference frames and the size of the MacroBlock (16). It will return the search window, the reconstructed image, the MSE and the total number of comparisons.

First we will initialize the variables and start a double loop going through each MacroBlock. We will get the current one taking it from the original frame and then go through smaller blocks applying SAD algorithm to get the most similar one.

We will set min_loss to a really big number and there we will save the minimum difference between the reference frame and the current one. We will go through each block starting in different possitions every 1 pixel and make sure we are inside the margins of the frame. In that case we will get the block in the reference frame starting in that possition, calculate the difference with the current one and take the absolute value of the sum. If this number is less than the

minimum we will update the variables and calculate the new MSE.

After trying with every position we will get the reconstructed image which will be most similar block found to the original one and we will also save the parameters in the search window in order to plot the motion vectors afterwards.

```
function [searchWindow,reconstructedImage,MSE,totalComparisons] =
    MotionEstimation(originalFrame,referenceFrame,MBSize)
totalComparisons = 0;
[row,col] = size(originalFrame);
c = 1;
for RowMB = 1:MBSize:row
    for ColMB = 1:MBSize:col
        currentMb = originalFrame([RowMB:(RowMB+MBSize-1)],
            [ColMB:(ColMB+MBSize-1)]);
        min_loss = 999999999;
        elements = zeros(1,2);
        for RowBlock = -1:1
            for ColBlock = -1:1
                % pad the search window with extra rows and columns
                paddedRow = RowMB+RowBlock;
                paddedCol = ColMB+ColBlock;
                if ((paddedRow + MBSize - 1) <= row) && ((paddedCol +MBSize - 1)<= col)
                    && (paddedRow > 0) && (paddedCol > 0)
                    RefMb = referenceFrame(paddedRow:(paddedRow+MBSize-1),
                        paddedCol:(paddedCol+MBSize-1));
                    difference_matrix = RefMb - currentMb;
                    %Computing SAD
                    diff = abs(sum(difference_matrix(:)));
                    if (diff<min_loss)
                        min_loss = diff;
                        elements = [paddedRow,paddedCol];
                        % Calculate the MSE
                        MSE = sum(sum(difference_matrix.^2));
                        MSE = MSE./256;
                        totalComparisons = totalComparisons +1;
                    end
                end
            end
        end
        VectDisplay = [elements(1) - RowMB,elements(2) - ColMB];
        %Get the reconstructed image
        reconstructedImage(RowMB:(RowMB+MBSize - 1),ColMB:(ColMB+MBSize -1)) =
            referenceFrame(elements(1):(elements(1)+MBSize - 1),elements(2):
            (elements(2)+MBSize -1));
        % Update the searching window
        searchWindow(c,:,1) = [RowMB,ColMB];
```

```
        searchWindow(c,:,2) = VectDisplay;
        c = c+1;
    end
end
end
```

[4]

Computational load

For a macroblock size of 16x16 pixels, SAD requires 2x256-1=511 additions/subtractions. Since the formula of the Sum of Absolute Difference for a macroblock $B_k(n)$ is given by:
$Distortion(B_k(n), d) = \sum_{\forall(x,y)\in B_k(n)} |U_k(x, y) - U_r(x + d_x, y + d_y)|$[5]

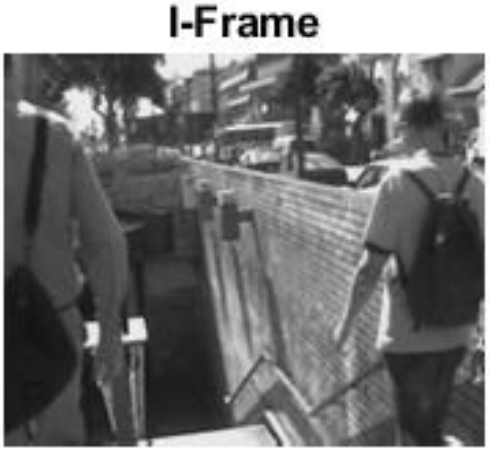
For each motion vector there are $(2w + 1)^2$ possible values. The distortion will be calculated then $(2w+1)^2$ times. But then we have to minimize that distortion so if $w = 1$ then $(2w+1)^2 = 9$ and the SAD will need 9x511=4599 additions.

On the other hand if we compute the program we get a total of 775 comparisons and 9 in each macroblock.

3 Results

After obtaining the frames of the video this is the Y component of the 6th frame.

Figure 1: I-Frame



Now for the following 4 frames we will get the motion vector, the Y component of the original frame, the predicted video frame and the error frame between the correlative frames:

Figure 2: Motion vector for image 7 and 8

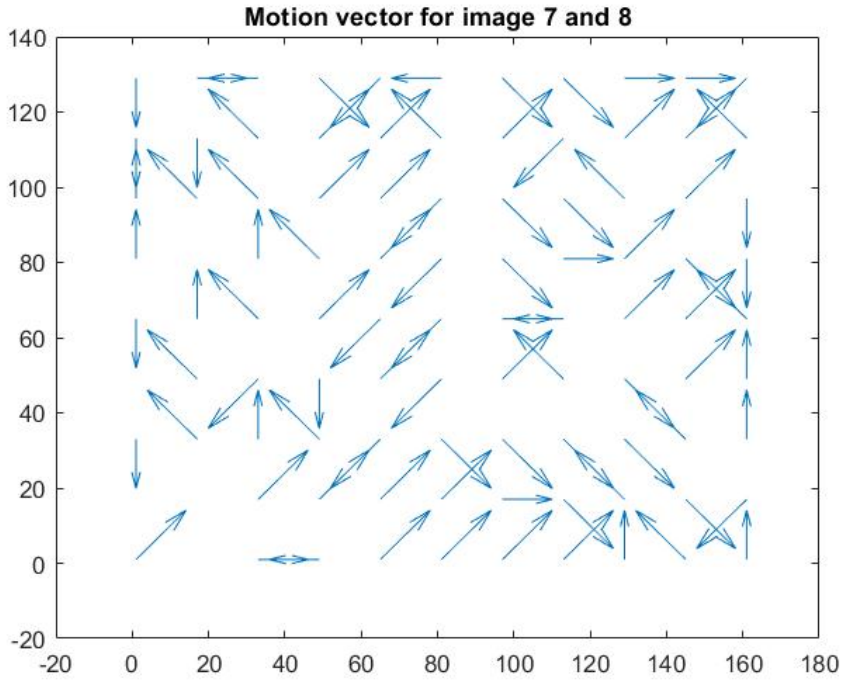


Figure 3: Y component of the original video frame 7, predicted video frame 8 and error frame

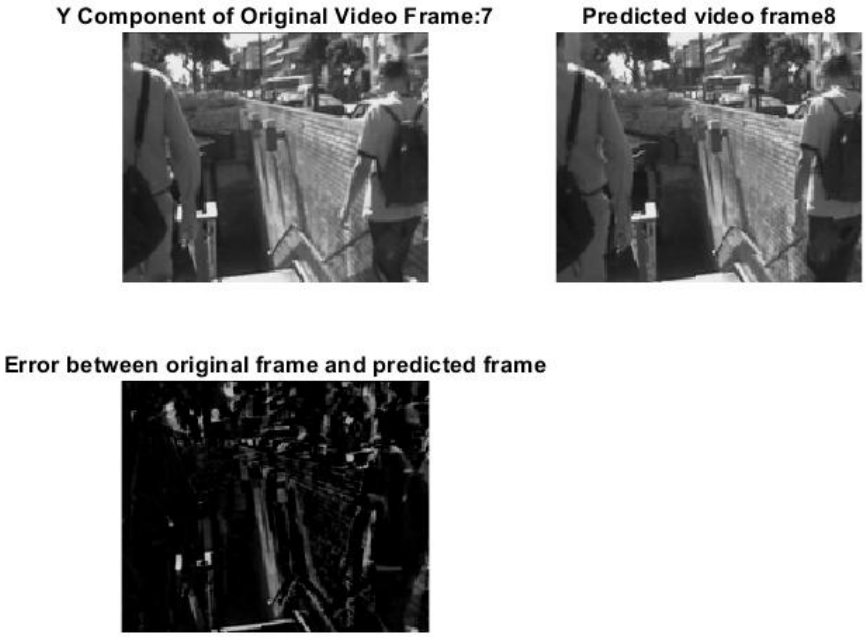


Figure 4: Motion vector for image 8 and 9

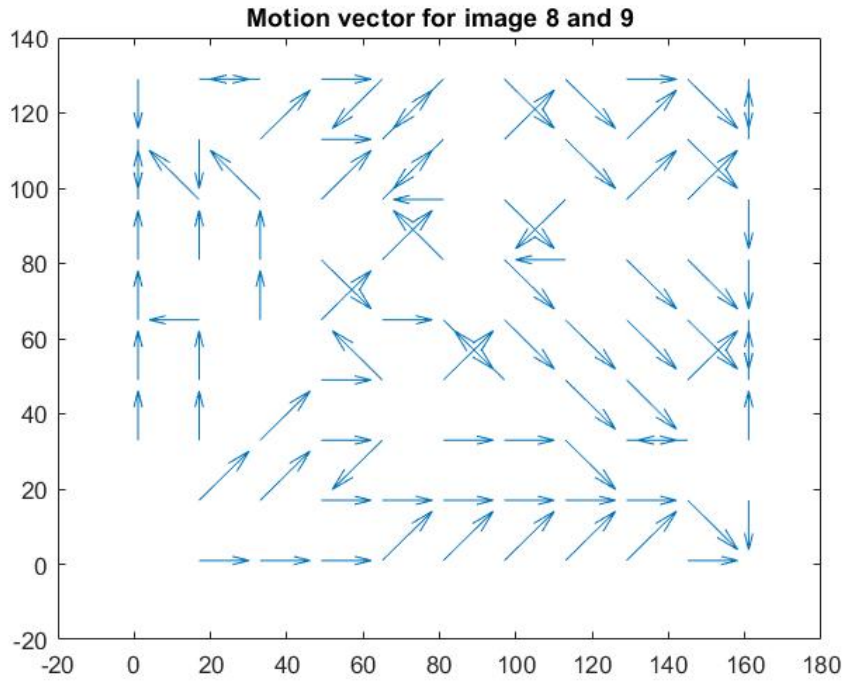


Figure 5: Y component of the original video frame 8, predicted video frame 9 and error frame

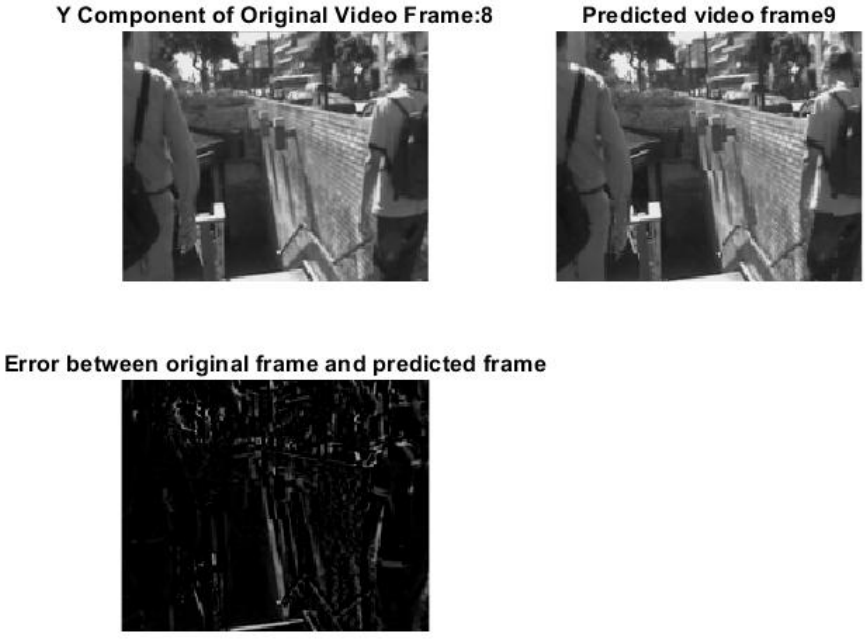


Figure 6: Motion vector for image 9 and 10

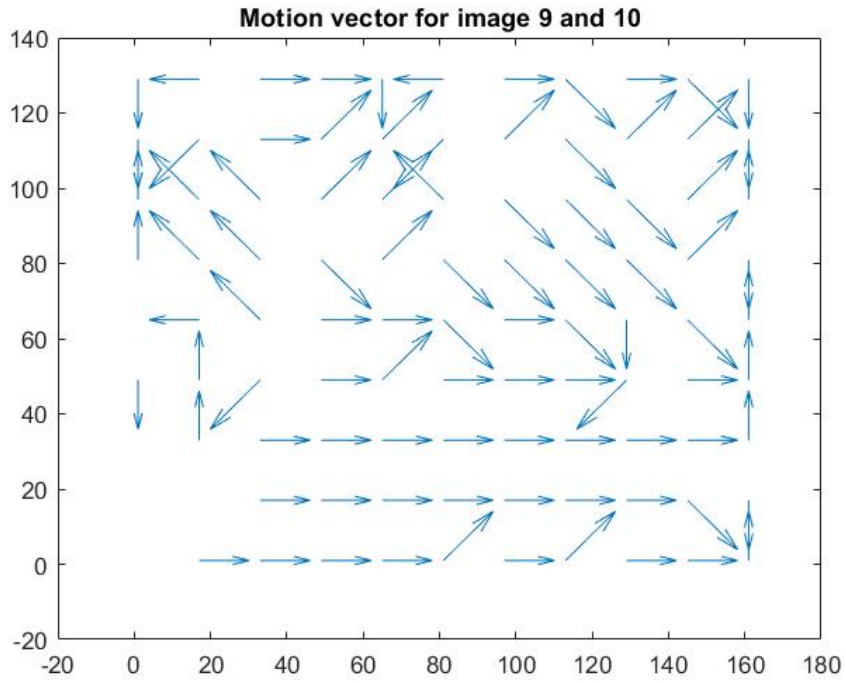


Figure 7: Y component of the original video frame 9, predicted video frame 10 and error frame

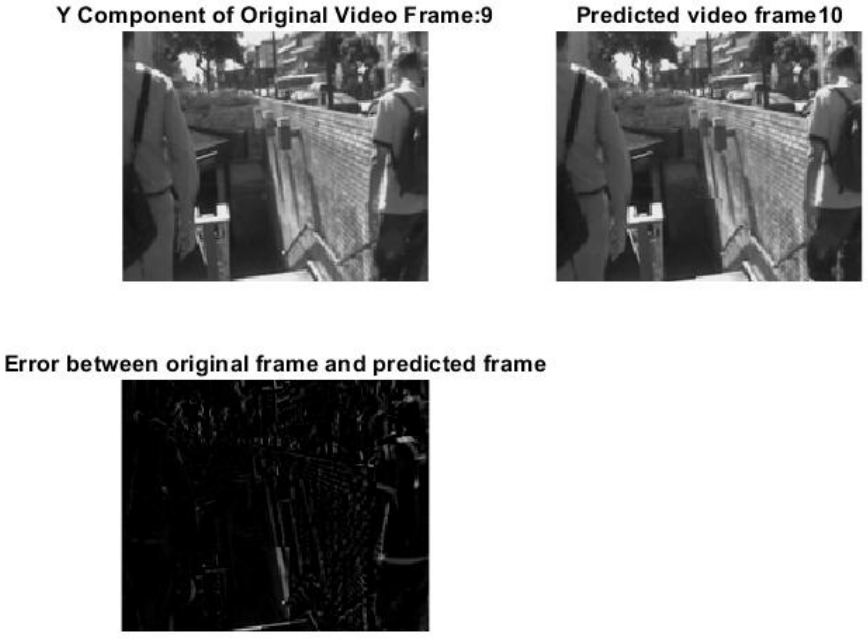


Figure 8: Motion vector for image 10 and 11

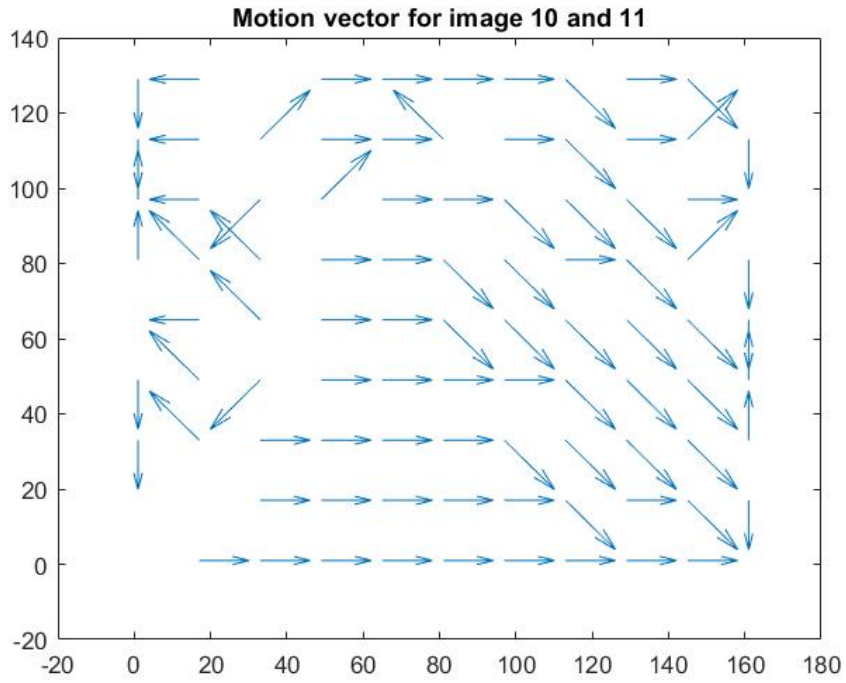
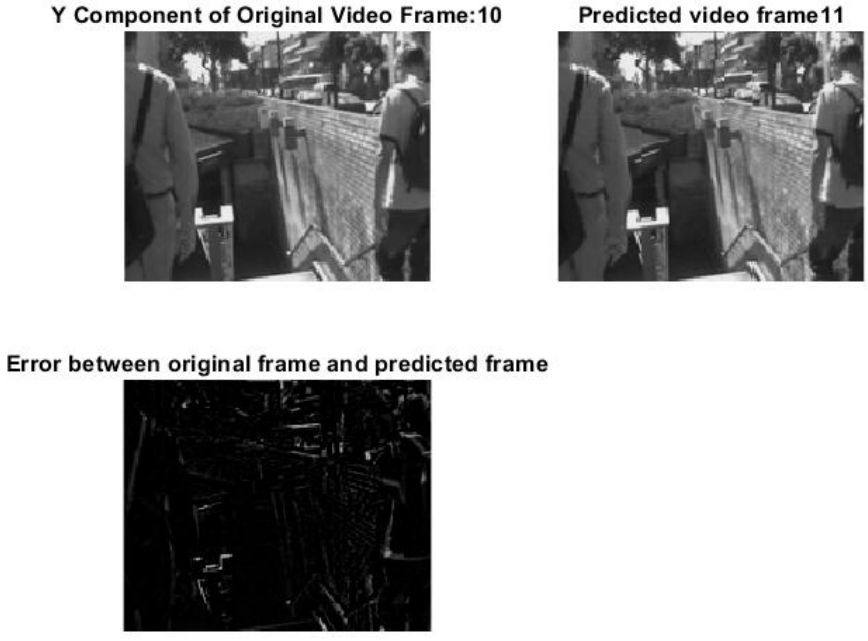


Figure 9: Y component of the original video frame 10, predicted video frame 11 and error frame



As we can see, the motion vectors indicate the movement of the video between the frames and it seems to follow some directions. Also, the predicted video frames are pretty clear although

they may have little imperfections. This shows that our function is working properly because the reconstructed image looks really similar to the following one. For example, between frames 7 and 8 we can see that the arm moves to the right and the direction of the vectors indicates the same. The MSE is always between 7 and 85 and indicates how different the two frames are.

```
The total number of blocks in the image is:99
MSE between frame 7 and 8 is 7.371094e+00
MSE between frame 8 and 9 is 2.058984e+01
MSE between frame 9 and 10 is 6.912109e+01
MSE between frame 10 and 11 is 8.099609e+01
```

Here we can see the MSE for each two correlative frames and it is quite small.

4 Conclusion

With this assignment I have learned how to read and display videos, split them into different frames and implement a motion estimation technique making use of the sum of absolute differences.

The goal of this assignment was to reduce temporal redundancy in a video thanks to the motion estimation technique finding the most similar frame from the original one. Also, learning how to determine and show motion vectors and the reconstructed frame.

For me the most difficult part was to code the motion estimation function and make it work because there are a lot of nested loops and it could get a bit confusing. Also, studying the SAD algorithm and apply it into the function was challenging because I hadn't seen anything similar before.

In my opinion this assignment was a really good way to learn this technique and understand the importance of reducing temporal redundancy in videos.

5 References

[1] https://www.mathworks.com/help/matlab/ref/videoreader.hasframe.html.

[2] https://en.wikipedia.org/wiki/Group_of_pictures.

[3] https://www.mathworks.com/help/matlab/ref/quiver.html.

[4] COMPE 565 Cheolhong An. *Introduction to Motion Estimation Motion compensation.*

[5] CompE 565: Multimedia Communication Systems. *HW 3: Motion Estimation for Video Compression (project description).*