Investigating the Motion Estimation by Block Algorithm Technique

Harshdeep Singh Mann¹, Kanishk Bansal², Niriksha S.³ and C.Gururaj⁴
123Student, 4Assistant Professor, Dept. of Telecommunication Engg.
1,2,3,4 BMS College of Engineering, Bangalore, Karnataka, India
Email: harshdeep.mann@yahoo.com; kanishkbansal@yahoo.com; niriksha72@gmail.com, gururaj.tce@bmsce.ac.in

Abstract— This paper is a review of the block matching algorithms used for motion estimation in video compression. It implements and compares 7 different types of block matching algorithms. The algorithms that are evaluated in this paper are widely accepted by the video compressing community and have been used in implementing various standards, ranging from MPEG1 / H.261 to MPEG4 / H.263. The paper also presents a very brief introduction to the entire flow of video compression.

I. INTRODUCTION

With the advent of the multimedia age and the spread of Internet, video storage on CD/DVD and streaming video has been gaining a lot of popularity. The ISO Moving Picture Experts Group (MPEG) video coding standards pertain towards compressed video storage on physical media like CD/DVD, while the International Telecommunications Union (ITU) addresses real-time point-to-point or multi-point communications over a network. The former has the advantage of having higher bandwidth for data transmission. Interframe predictive coding is used to eliminate the large amount of temporal and spatial redundancy that exists in video sequences and helps in compressing them. The use of the knowledge of the displacement of an object in successive frames is called Motion Compensation. There are a large number of motion compensation algorithms for interframe predictive coding. In this study, however, we have focused only on one class of such algorithms, called the Block Matching Algorithm. These algorithms estimate the amount of motion on a block by block basis, i.e. for each block in the current frame, a block from the previous frame is found, that is said to match this block based on a certain criterion. There are a number of criteria to evaluate the "goodness" of a match and some of them are:

1. Cross Correlation Function
2. Pel Difference Classification (PDC)
3. Mean Absolute Difference
4. Mean Squared Difference
5. Integral Projection

The whole idea behind motion estimation based video compression is to save on bits by sending JPEG encoded difference images which inherently have less energy and can be highly compressed as compared to sending a full frame that is JPEG Motion JPEG, where all frames are JPEG encoded, achieves anything between 10:1 to 15:1 compression ratio, where as MPEG can achieve a compression ratio of 30:1 and is also useful at 100:1 ratio.

On the whole there are a very large number of algorithms for block based motion compensation. This report includes a study of some of the main algorithms. We have tried to implement some of these and the results are discussed.

DOI: 03.AETS.2013.4.102
© Association of Computer Electronics and Electrical Engineers, 2013
II. BLOCK MATCHING ALGORITHMS

The idea behind block matching is to divide the current frame into a matrix of ‘macro blocks’ that are then compared with corresponding block and its adjacent neighbors in the previous frame to create a vector that stipulates the movement of a macro block from one location to another in the previous frame. This movement calculated for all the macro blocks comprising a frame, constitutes the motion estimated in the current frame. The search area for a good macro block match is constrained up to p pixels on all fours sides of the corresponding macro block in previous frame. This ‘p’ is called as the search parameter. Larger motions require a larger p, and the larger the search parameter the more computationally expensive the process of motion estimation becomes. Usually the macro block is taken as a square of side 16 pixels, and the search parameter p is 7 pixels. The matching of one macro block with another is based on the output of a cost function. The macro block that results in the least cost is the one that matches the closest to current block.

There are various cost functions, of which the most popular and less computationally expensive is Mean Absolute Difference (MAD) given by equation (i). Another cost function is Mean Squared Error (MSE) given by equation (ii).

\[
MAD = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |C_{ij} - R_{ij}|
\]

(i)

\[
MSE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (C_{ij} - R_{ij})^2
\]

(ii)

Peak-Signal-to-Noise-Ratio (PSNR) given by equation (iii) characterizes the motion compensated image that is created by using motion vectors and macro clocks from the reference frame.

A. Three Step Search (TSS) [1,2]

This algorithm was introduced by Koga et al in 1981. It searches for the best motion vectors in a coarse to fine search pattern. The algorithm may be described as:
Step 1: An initial step size is picked. Eight blocks at a distance of step size from the centre (around the centre block) are picked for comparison.
Step 2: The step size is halved. The centre is moved to the point with the minimum distortion.
Steps 1 and 2 are repeated till the step size becomes smaller than 1. A particular path for the convergence of this algorithm is shown below:

![Three Step Search Diagram]

Fig 2.1 : Example path for convergence of Three Step Search

One problem that occurs with the Three Step Search is that it uses a uniformly allocated checking point pattern in the first step, which becomes inefficient for small motion estimation.
B. Two Dimensional Logarithmic Search (TDL) [1]

This algorithm was introduced by Jain & Jain around the same time that the Three Step Search was introduced and is closely related to it. Although this algorithm requires more steps than the Three Step Search, it can be more accurate, especially when the search window is large. The algorithm may be described as:

Step 1: Pick an initial step size. Look at the block at the center of the search area and the four blocks at a distance of \( s \) from this on the X and Y axes. (the five positions form a + sign)

Step 2: If the position of best match is at the center, halve the step size. If however, one of the other four points is the best match, then it becomes the center and step 1 is repeated.

Step 3: When the step size becomes 1, all the nine blocks around the center are chosen for the search and the best among them is picked as the required block. A particular path for the convergence of the algorithm is shown in the following figure:

![Fig 2.2: Example path for convergence of Two Dimensional Logarithmic Search](image)

A lot of variations of this algorithm exist and they differ mainly in the way in which the step size is changed.

C. Binary Search (BS) [3]

The basic idea behind this algorithm is to divide the search window into a number of regions and do a full search only in one of these regions. It may be described as:

Step 1: The MAD is evaluated on a grid of 9 pixels that include the centre, the four corners of the search window and four pixels at the boundaries. The search window is divided into regions based on these points.

Step 2: A full search is performed in the region corresponding to the point with the smallest MAD.

The convergence of the algorithm may be viewed in figure 4:

![Fig 2.3: Example path for convergence of Binary Search](image)
D. Four Step Search (FSS) [3]

The algorithm starts with a nine point comparison and then the other points for comparison are selected based on the following algorithm:

Step 1: Start with a step size of 2. Pick nine points around the search window centre. Calculate the distortion and find the point with the smallest distortion. If this point is found to be the centre of the searching area go to step 4, otherwise go to step 2.

Step 2: Move the centre to the point with the smallest distortion. The step size is maintained at 2. The search pattern, however depends on the position of the previous minimum distortion.

a) If the previous minimum point is located at the corner of the previous search area, five points are picked (as shown in the figure).

b) If the previous minimum distortion point is located at the middle of the horizontal or vertical axis of the previous search window, three additional checking points are picked. (as shown in the figure)

Locate the point with the minimum distortion. If this is at the centre, go to step 4 otherwise go to step 3.

Step 3: The search pattern strategy is the same, however it will finally go to step 4.

Step 4: The step size is reduced to 1 and all nine points around the centre of the search are examined.

E. Orthogonal Search Algorithm (OSA) [1]

This algorithm was introduced by Puri in 1987 and it is a hybrid of the Three Step Search and the Two Dimensional Logarithmic Search.

It has a vertical stage followed by a horizontal stage for the search for the optimal block. The algorithm may be described as follows:

Step 1: Pick a step size (usually half the maximum displacement in the search window).

Take two points at a distance of step size in the horizontal direction from the centre of the search window and locate (among these) the point of minimum distortion. Move the centre to this point.

Step 2: Take two points at a distance step size from the centre in the vertical direction and find the point with the minimum distortion.

Step 3: Halve the step size, if it is greater than one, else halt.
A particular path for the convergence of the algorithm may be shown in the following figure.

![Figure 2.5: Example path for convergence of Orthogonal Search](image)

**F. One at a Time Algorithm (OTA) [1]**

This is a simple, but effective way of trying to find a point with the optimal block. During the horizontal stage, the point on the horizontal direction with the minimum distortion is found. Then, starting with this point, the minimum distortion in the vertical direction is found. The algorithm may be described as follows:

1. Pick three points about the centre of the search window (horizontal).
2. If the smallest distortion is for the centre point, start the vertical stage, otherwise look at the next point in the horizontal direction closer to the point with the smallest distortion (from the previous stage). Continue looking in that direction till you find the point with the smallest distortion.
3. Repeat the above, but taking points in the vertical direction about the point that has the smallest distortion in the horizontal direction.

One particular search path for the algorithm is shown in figure 8. This search algorithm requires very little time, however the quality of match is not very good.

![Figure 2.6: Example path for convergence of One at a Time Algorithm](image)
III. RESULTS

In this work, three video sequences Car, idea & plumber are used for compression. Car & plumber are in QCIF format (176 X 144) at 25 FPS frame rate & idea sequence is in CIF format (320X240) at 25 FPS frame rate. The PSNR comparison of the compensated images are generated using the motion estimation algorithms for car, idea and plumber respectively. Also performance comparison of computations for all video sequences is given.

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>AMAD</th>
<th>CPU TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSS</td>
<td>6.824</td>
<td>47.32</td>
</tr>
<tr>
<td>TDL</td>
<td>8.414</td>
<td>1.73</td>
</tr>
<tr>
<td>BS</td>
<td>9.272</td>
<td>0.6</td>
</tr>
<tr>
<td>FSS</td>
<td>8.638</td>
<td>1.04</td>
</tr>
<tr>
<td>OTA</td>
<td>9.035</td>
<td>0.49</td>
</tr>
<tr>
<td>OSA</td>
<td>8.714</td>
<td>0.99</td>
</tr>
</tbody>
</table>

TABLE II: IMAGE STREAM IDEA IMAGE FORMAT QCIF NUMBER OF FRAMES 10

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>AMAD</th>
<th>CPU TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSS</td>
<td>2.5</td>
<td>0.69</td>
</tr>
<tr>
<td>TDL</td>
<td>2.776</td>
<td>1.8</td>
</tr>
<tr>
<td>BS</td>
<td>3.005</td>
<td>0.56</td>
</tr>
<tr>
<td>FSS</td>
<td>2.827</td>
<td>1.0</td>
</tr>
<tr>
<td>OTA</td>
<td>2.895</td>
<td>0.39</td>
</tr>
<tr>
<td>OSA</td>
<td>2.808</td>
<td>1.00</td>
</tr>
</tbody>
</table>

TABLE III: IMAGE STREAM PLUMBER IMAGE FORMAT QCIF NUMBER OF FRAMES 10

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>AMAD</th>
<th>CPU TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSS</td>
<td>3.511</td>
<td>N/A</td>
</tr>
<tr>
<td>TDL</td>
<td>6.500</td>
<td>20.99</td>
</tr>
<tr>
<td>3.758</td>
<td>62.5</td>
<td>BS</td>
</tr>
<tr>
<td>3.675</td>
<td>41.83</td>
<td>FSS</td>
</tr>
<tr>
<td>3.751</td>
<td>17.07</td>
<td>OTA</td>
</tr>
<tr>
<td>3.7957</td>
<td>36.59</td>
<td>OSA</td>
</tr>
</tbody>
</table>

IV. CONCLUSION

In this report, different algorithms were presented and discussed. We implemented and compared most of them based on some test streams. Our evaluation was based on two measures, CPU time for computational complexity and AMAD for quality. As it is mentioned in the last section, it's believed that the hierarchical algorithms succeeded on maintaining the same quality of the Full Search and reducing the computational complexity. However, we are not sure about the AMAD as a reasonable measure. We are also not sure as whether a difference in AMAD represented by a difference of, say, 0.2 really translated to a large difference in perceptual quality. Another important measure that we need to consider in our future work is the bit stream. This measure will help us evaluate the performance of our algorithms, because ultimately these are the things that are being sent over the communication line. We also intend to continue looking at different
types of algorithms and working toward exploring new directions. We might try to combine the hierarchical approach with other fast search methods such as FSS and OSA to further reduce computations with little degradation on the quality.

REFERENCES