A Method for Motion Adaptive Frame Rate Up–Conversion

R. Castagno*  P. Haavisto†  G. Ramponi‡

Abstract

A frame interpolation algorithm for frame rate up-conversion of progressive image sequences is proposed. The algorithm is based on simple motion compensation and linear interpolation. A motion vector is searched for each pixel in the interpolated image and the resulting motion field is median filtered to remove inconsistent vectors. Averaging along the motion trajectory is used to produce the interpolated pixel values.

The main novelty of the proposed method is the motion compensation algorithm which has been designed with low computational complexity as an important criterion. Subsampled blocks are used in block matching and the vector search range is constrained to the most likely motion vectors. Simulation results show that good visual quality has been obtained with moderate complexity.

The algorithm has been designed mainly for 50 Hz to 75 Hz frame rate up-conversion with applications in a multimedia environment, but it can also be used in advanced television receivers to remove artifacts due to low scan rate.

Corresponding author:
Roberto Castagno
Signal Processing Laboratory
Swiss Federal Institute of Technology
CH-1015 LAUSANNE, SWITZERLAND
Tel.: +41 21 693 3632
FAX: +41 21 693 7600
E-mail: castagno@ltssg4.epfl.ch

*D.E.E.I., University of Trieste, Italy.
Current affiliation:
Signal Processing Laboratory
Swiss Federal Institute of Technology
Lausanne, Switzerland
†Nokia Research Center, Tampere Finland
‡D.E.E.I., University of Trieste, Italy
1 Introduction

The development of advanced television and the use of multimedia information systems has caused a rapid growth in the number of image sources and display formats used today. This has resulted in a need for efficient and flexible algorithms to convert between these formats [1]. Furthermore, display format conversions are also used as an efficient means to reduce artifacts due to the sampling process, thus improving the visual image quality by better matching the display format with the human visual characteristics.

Current European television standards, as well as the future HDTV transmission standard, employ an interlaced scan format at a field rate of 50 Hz. This results in visible artifacts, such as line flicker, line crawl, and large area flicker. Increasing the scan rate of the display translates these artifacts to higher frequencies where the human eye is less sensitive to them, yielding better subjective image quality [2]. The conversion of the received interlaced signal to a progressive format (deinterlacing) achieves a significant reduction in line flickering and line crawl. The doubling of the field rate from 50 to 100 Hz (interlaced format) is another solution to the above mentioned problems and has already been implemented in commercial television sets. Some proposed algorithms perform a format conversion and other tasks at the same time, such as noise reduction (for example [3]) or cross color reduction [4].

Conversion methods have been proposed with a wide range of computational complexities depending on the final task: consumer electronics products will require easily implementable (and thus cheaper) solutions, professional equipment will achieve better results at the cost of a higher complexity and with small production volumes. Suggested approaches also vary: for example, in [5] sampling rate reduction and display up-conversion are performed using three-dimensional linear low-pass filtering assisted by motion adaptation both in pre- and post-filtering. Other authors suggest adopting non-linear techniques for deinterlacing tasks [6, 7, 8] or for field rate up-conversion [9, 10, 4].

Progressive image format for video is a natural choice in a multimedia environment where workstation monitors are used as display devices. 50 Hz frame rate is not adequate in this application because of large area flicker, and, therefore, monitors often use a frame rate closer to 75 Hz. Converting a 50 Hz interlaced video source to be displayed on a workstation monitor requires three steps: deinterlacing, frame rate up-conversion to about 75 Hz, and format conversion to adjust the spatial resolution to the display window [11, 8] (See Fig. 1).

This paper proposes an algorithm for frame rate up-conversion of progressive image sequences by a factor of 1.5, i.e., from 50 Hz to 75 Hz (or equivalently from 60 Hz to 90 Hz). In a multimedia environment this algorithm ought to be considered as a block of the above described larger transcoding system. However, the algorithm can also be used in advanced television receivers where
the 75 Hz progressive format can be feasible whereas the 100 Hz progressive rate could be technically too complex.

An important design constraint to the proposed algorithm was keeping the computational complexity as low as possible. Although the frame interpolation algorithm has been designed with the 1.5 conversion factor in mind, it is trivial to extend the algorithm to double the frame rate, say, from 50 Hz to 100 Hz. Moreover, with slight modifications the method is well suited also for other conversion factors, such as 1.25 (50 Hz to 62.5 Hz or 60 Hz to 75 Hz).

Despite the wide variety of conversion tasks, some basic considerations are common to most cases. It is well known that methods involving simple frame repetition are simple to implement, give “perfect” results where no motion is present, but fail badly in non-stationary areas, resulting in clearly visible motion jerkiness. Linear interpolation between pixels located in the same position in the previous and following original frames also performs well only in stationary areas, but results in blurring where movement is present. The need for some form of motion adaptiveness is clear (see Sec. 2.2 for references).

The paper is organized as follows: in Sec. 2 the frame interpolation algorithm is presented. Detailed explanation of the most important procedures is given in Sec. 3. Simulation results are discussed in Sec. 4.

2 50 Hz to 75 Hz frame rate up-conversion

2.1 Temporal distribution of interpolated frames

Most proposed methods for frame (or field) rate up-conversion obtain the up-converted sequence by repeating some or all of the incoming frames and interpolating the remaining ones. For 50 Hz to 100 Hz conversion, all the original frames are included in the new sequence, and frame rate doubling is achieved by interpolating a new frame between every two consecutive original frames. This makes it possible to design symmetric algorithms in which all incoming fields are treated equally. This is not the case in 50 Hz to 75 Hz conversion where a 1.5 conversion factor is needed and the contribution of the original frames to the interpolated ones varies.

Let us consider the temporal distances between interpolated and original frames in the 50 Hz and 75 Hz sequences. A simple example of the temporal distribution of interpolated frames is shown in Fig. 2. One out of three new frames can be obtained by repeating original frames, while the other two are interpolated. Since the quality of the original frames will invariably be better than that of the interpolated frames, it is a disadvantage that only half of the originally available frames are displayed. In addition to this, the temporal location of the interpolated frames is not symmetric with respect to the original frames from which they originate, and thus the contribution of such frames should be not equal. To interpolate linearly a frame which is located
at distances ($d_1, d_2$) from input frames ($F_1, F_2$), the weights to be used are
$d_2/(d_1 + d_2)$ and $d_1/(d_1 + d_2)$ respectively. In the situation of Fig. 2 the
weights would be $1/3$ and $2/3$, requiring divisions by 3 which are not as easily
implementable on hardware as division by 2 or 4. This problem has to be faced
whenever a non-integer conversion factor is involved [9, 12, 13]. In these cases,
the importance of a correct choice of the original fields to be used in order to
obtain judder-free interpolation must be underlined.

The proposed algorithm uses a time shifted sequence of interpolated frames
in which one out of every three new frames (we shall call them “z-frames”, see
Fig. 3) is located half way between two original frames, and the remaining two
are located at 3.3 ms from an original frame (x-frames and y-frames). The
z-frames are then interpolated from two original frames as described in the
following. The x- and y-frames are replicas of the original frames.

This approach allows us to:

- use all the original frames in the new sequence, retaining a higher amount
  of original information.

- interpolate only one out of three frames in the new sequence, reducing
  by half the amount of computation needed if compared with the method
  of Fig. 2.

- avoid divisions by 3, simplifying the hardware implementation.

As a drawback, this choice could introduce a certain amount of motion
jerkiness since the x- and y-frames undergo a small temporal displacement.
However, subjective testing has shown that when viewed at 75 Hz this artifact
is not disturbing even in the most critical areas of slow uniform motion.

When implemented using these principles, the 50 to 75 Hz conversion re-
duces to frame interpolation. The interpolation algorithm proposed in this
paper can be used for other applications, e.g. for 50 to 100 Hz or 50 to 62.5 Hz
conversion. While in the former case the extension is straightforward, in the
latter case small temporal displacements should again be introduced: a sample
scheme in which 2 out of 5 frames are interpolated and the remaining 3 are
replicas of original ones is depicted in Fig.4. Frame A is copied, frame B is
copied with an anticipation of 4 ms, frame C is interpolated ($C'$) and delayed
by 2 ms, frame D is interpolated ($D'$) and anticipated by 2 ms and eventually
frame E is copied and delayed by 4 ms.

2.2 Motion compensation

The use of motion compensation is necessary in frame interpolation where no
spatial information is available for the interpolated frame. In deinterlacing,
for example, half of the lines of the interpolated frame are available giving us
an idea of “what is happening in the sequence at that moment.” In frame
interpolation, we do not have such help: all the information available belongs either to the previous or the following frame, but is not simultaneous with the frame to be inserted in the sequence. Hence the importance of finding and exploiting spatial correlation between the two available frames.

Motion estimation and compensation typically is the computationally most complex part of a frame interpolation algorithm, and often represents its most distinctive part.

As a low level approach, we can name the simple detection of motion. Even the simplest conversion methods consist in fact of at least two different algorithms: one for stationary areas, one for moving picture elements [14]. It is typical of most deinterlacing methods (for example [10, 13]), to switch between inter-field interpolation for stationary areas and intra-field interpolation for moving areas. A more sophisticated solution consists of using different parameter settings according to different amounts of detected motion [9].

A dramatic improvement of image quality can be achieved if the algorithm for moving areas can not only adapt to the presence of motion, but also find and exploit information about the direction of this motion. The most commonly used tool for this task is block matching, but other solutions have also been adopted, such as phase correlation [15].

### 2.2.1 Block matching

Block matching methods themselves can have a wide range of performance and complexity level. For interpolation tasks, accuracy of motion estimation should go down to the single picture elements to avoid annoying artifacts, such as the typical blocking effect and errors on edges, to which the human eye is particularly sensitive.

Some authors suggest a one-pass procedure, e.g. [16], in which for each interpolated pixel a motion vector is estimated by matching a block of pixels with analogous displaced blocks. Other methods explore the motion characteristics of the sequence in different stages. Motion vectors are evaluated for larger areas first, and this information is then iteratively used as a starting point for more detailed analysis, involving smaller picture areas [15, 17, 18, 19]. It is common practice in the above cited literature to postprocess the estimated motion vector fields in order to increase their consistency and remove incorrectly estimated block vectors.

Different block sizes and pixel distribution have been suggested by several authors for a block matching procedure. In particular, once the size of the block of pixels has been chosen (e.g., $3 \times 3$ in Fig. 5(b) and $5 \times 3$ in Fig. 5(c) and (d)), it is not necessary to include all the pixels in the block, but subsampled blocks can be matched without degrading the results and with less computation, as in [20]. A $1:2$ sampling ratio, resulting in the checker-pattern of Fig. 5 (d), has been adopted for this work, and the sum of the absolute pixel differences
calculated over the 8 pixels in the block (block matching error, BME) has been used as the correlation index.

### 2.2.2 Assumption for linear motion

The approach followed in this paper is based on the assumption that any motion between two consecutive original frames is rectilinear and uniform. The task of motion estimation is to evaluate the motion vector whose components best approximate the horizontal and vertical displacement, between the previous and the following frame, of the object to which pixel P belongs. To evaluate the motion of the pixel in position \((x, y, t_n)\) in the interpolated frame, we consider a number of candidate motion vectors, we associate a trajectory to each of them and we compute the BME between pixel blocks located at the two ends of the trajectory. The vector with the highest correlation (i.e. the lowest BME) is eventually selected. In the following, positive motion vector coordinates refer to motion advancing from left to right and downwards.

When the motion vector has even components, the choice of the associated trajectory (and thus of the pixel blocks to be matched) is straightforward. In Fig. 6, let \((2i, 2j)\) be one of the vectors to be tested for the motion of pixel P with coordinates \((x, y, t_n)\): the corresponding trajectory connects pixels A and B, whose coordinates are respectively \((x - i, y - j, t_{n-1})\) and \((x + i, y + j, t_{n+1})\).

The case of motion vectors with one or both odd components needs to be briefly discussed, instead. While interpolating the pixel in position \((x, y, t_n)\), consider an object moving e.g. three pixels to the right between the two original frames. In particular let us follow a detail moving from position \((x - 1, y, t_{n-1})\) in the previous frame to position \((x + 2, y, t_{n+1})\) in the following one. There is no trajectory, either in the set centered in pixel \((x, y, t_n)\) or in the set centered in pixel \((x + 1, y, t_n)\) that connects these two pixels. The choice of the most correlated motion vector depends on noise and can quite likely lead to an error in the interpolation. The same happens for any movement involving an odd number of pixels horizontally and/or vertically.

A possible solution to this problem is to perform a spatial interpolation by a factor 2, vertically and horizontally, of the two input frames, and to use blocks of interpolated pixels for motion estimation when the vector under test has an odd component. The interpolated pixels should then also be used for the actual evaluation of the grey levels of the pixel in the new frame, after the best motion vector has been evaluated.

Our tests showed that the use of interpolated pixels in the block matching procedure degrades areas with tiny detail. Here spatial interpolation smooths the abrupt luminance changes, and hence the BME's corresponding to different trajectories are often quite near to each other. The choice of the best correlation can therefore be affected by noise and lead to interpolation errors,
which are randomly distributed in time and space and are quite noticeable, with flicker appearing in detailed zones.

A different solution is here proposed, using only original pixels in the motion estimation procedure, while spatially interpolated pixels are still used in the actual interpolation, after a motion vector with one or both odd components has been estimated.

Let \((x, y, t_n)\) be the pixel we want to interpolate, and let \(\tilde{V} = (2i + 1, 2j + 1)\) with \(i, j \in \mathbb{Z}\), the motion vector whose correlation index we want to evaluate. The two pixel blocks to be matched are then centered in pixels \((x-i, y-j, t_{n-1})\) and \((x+i+1, y+j+1, t_{n+1})\). This choice clearly introduces an inaccuracy in the motion estimation, since the interpolated pixel can be no longer located at the center of the block for which the motion is estimated (in particular the displacement can be 0, 0.5 or \(\sqrt{2}/2\) pixels, depending on the number of odd components). However, this displacement very seldom affects the overall quality of the interpolated sequence, since in the large majority of cases, i.e. inside picture areas with the same movement, the correct motion vector is chosen anyway. The only zones where errors might occur are those where objects with different motion trajectories overlap or are disclosed: in these cases, motion estimation is slightly biased. It should be noted, anyway, that all boundary pixels undergo the same displacement (so that the image maintains a good amount of coherence), and the resulting jerkiness is often very well masked by the general blurring that affects these areas, even in the original sequences.

In addition to this, the BME corresponding to the best correlated trajectory is generally quite different from those corresponding to other directions: this makes the method less sensitive to noise and increases the performance of the weighting procedure which will be described in Sec. 2.2.4.

### 2.2.3 Selection of candidate motion vectors

The choice of the candidate motion vectors directly affects the range of motion speeds that can be estimated with the algorithm, as well as the size and the shape of the blocks of pixels used in the block matching procedure. Extending the search for correlation to trajectories corresponding to higher speeds also increases the risk of erroneously finding motion in stationary picture areas, and vice versa. This must be prevented by using larger blocks in block matching, which unfortunately results in a higher amount of computation.

Several candidate motion vector sets have been tested with real world as well as test sequences, and a speed limit of 4 pixels per frame horizontally and 2 pixels per frame vertically has proved to be a good compromise. This choice gives more importance to horizontal motion which is more common in natural sequences than vertical motion. Although the range of motion speeds that can be compensated for is reduced, the performance degradation caused
by this limitation is usually masked by the very same motion. When the above mentioned speed limit is set, a $5 \times 3$ subsampled block with 8 pixels in a quincunx distribution turned out to achieve a good compromise between performance and complexity of the block matching procedure.

### 2.2.4 Restricted motion vector search

For some applications, the above described algorithm can be quite complex: a thorough search in the speed range $(-4 \ldots +4)$ horizontally and $(-2 \ldots +2)$ vertically would require the test of 45 motion vector candidates for each interpolated pixel. Each test involves computing 8 absolute pixel differences, resulting in a block matching complexity of about 360 absolute value calculations and additions per pixel.

By assuming a smoothly varying motion vector field, a large number of motion vector candidates can be skipped with little loss of estimation accuracy.

As well as setting a limit to the fastest motion speeds we want to detect by sizing the support window, we can set a similar limit to the speed differences between adjacent pixels. A simple way to achieve this goal is to restrict the search for a motion vector to a limited number of vectors which are, according to some criterion, not very different from those belonging to the neighboring pixels. This can also be seen as a kind of preliminary consistency check on the motion vector field. In our algorithm, when interpolating the pixel in position $(x, y)$, we use the motion vector found for the previous pixel $(x - 1, y)$ as a reference and we only search its 9 neighboring vectors. Denote the vector of the pixel $(x - 1, y)$ as $\mathbf{V}$ and its components as $V_x$ and $V_y$. We perform the block matching procedure only on the trajectories corresponding to vectors whose components differ by 0 or 1 from $V_x$ and $V_y$. In this way only 9 block matchings have to be performed instead of 45 for each interpolated pixel. Similar use of previously estimated motion vectors as a starting attempt for further search has been suggested in [17] and [22].

It is clear that the described restriction on the number of trajectories to be tested slows down the reaction of the algorithm to changes in the motion vector field. Where an edge in the motion vector field appears (e.g., for overlapping objects with different trajectories), the restricted search tends to produce a streaking effect where several horizontally consecutive pixels may take on wrong values until a correct motion estimate is again found (see Fig. 10). A similar problem affects a quite common pattern: a narrow vertical line moving on (or together with) a uniform background. In the uniform area, the motion vectors tend to assume a random distribution. When the algorithm reaches the vertical line in its scanning, the prediction motion vector $\mathbf{V}$ can therefore have any of the possible orientations, and in particular a very different one from that of the real motion of the detail. The restricted search procedure will then fail in the detection of the detail and an error will occur (see Fig. 13(a)).
To reduce this problem, we propose to weight the correlation indexes in a way that favours vectors which are closer to \((0, 0)\). A similar approach has been applied also in commercial motion-estimation chips [23]. The aim is to let motion vector \(\hat{V}\) move to extreme positions only when it is necessary, and try to bring it back to a more neutral configuration as often as possible. This will make it easier for the algorithm to react on a new motion direction and to lock on the motion of a new object or a detail. Let BME be the block matching error associated with the vector \(\hat{U}\), whose correlation index is being weighted. The weighted correlation index (WCI) is then given by the formula:

\[
WCI = BME \cdot (1 + K \cdot (U_x^2 + U_y^2))
\]

(1)

Where \(K\) is a parameter that can be thought of as an elastic constant. The simulations suggest a value in the range 0.05–0.2.

With reference to the last paragraph of Sec. 2.2.2, if the non-weighted BME corresponding to the most correlated direction is generally quite smaller than the others, a higher value of \(K\) can be used in the weighting procedure, thus achieving a better behaviour of the method in the above described critical areas.

### 2.2.5 Median filtering of the motion vector field

The above described procedure produces a motion vector field where each of the interpolated pixels has its own motion vector. There may occur errors in the estimation of motion vectors, mainly due to noise, especially in areas where small detail is present. In order to increase the consistency of the motion vector field componentwise median filtering is applied to the vector field. A \(3 \times 3\) window is used; the horizontal and vertical components of the 9 input vectors are filtered separately and the outputs are chosen as components of the output vector.

It should be pointed out that the output vector does not necessarily belong to the set of 9 input vectors, as we would expect from a standard median filter: however the other characteristics of median filtering – such as impulsive noise removal and edge preservation – are maintained by componentwise filtering, which on the other hand greatly reduces the amount of computation needed, if compared with other suggested approaches to vector median filtering. [21]

### 3 A detailed description of the algorithm

A formal description of the algorithm is presented here.

Let \((x, y, t_n)\) be the pixel to be interpolated, and let \(\hat{V}\) be the estimated motion vector for pixel \((x-1, y, t_n)\): the 9 vectors \((U^{(1)}, \ldots, U^{(9)})\) along which the block matching procedure is performed are listed in Tab. 1.
\[ \tilde{U}^{(1)} = (V_x - 1, V_y - 1) \quad \tilde{U}^{(2)} = (V_x, V_y - 1) \quad \tilde{U}^{(3)} = (V_x + 1, V_y - 1) \]
\[ \tilde{U}^{(4)} = (V_x - 1, V_y) \quad \tilde{U}^{(5)} = (V_x, V_y) \quad \tilde{U}^{(6)} = (V_x + 1, V_y) \]
\[ \tilde{U}^{(7)} = (V_x - 1, V_y + 1) \quad \tilde{U}^{(8)} = (V_x, V_y + 1) \quad \tilde{U}^{(9)} = (V_x + 1, V_y + 1) \]

Table 1: The nine motion vectors to be investigated for the interpolation of pixel in position \((x, y)\) when \(\tilde{V} = (V_x, V_y)\) is the estimated motion vector for pixel in position \((x - 1, y)\).

We must evaluate the coordinates of the pixels where each of the corresponding trajectory intersects the previous and the following frame, taking into account the proper displacements for vectors with odd components. Denote with \(\Delta X^{(h)}_{n-1}, \Delta Y^{(h)}_{n-1}, \Delta X^{(h)}_{n+1}, \Delta Y^{(h)}_{n+1}\) the horizontal and vertical offsets of the end-points of the trajectory from pixels \((x, y, t_{n-1})\) e \((x, y, t_{n+1})\) respectively.

\[
\begin{align*}
\Delta X^{(h)}_{n-1} &= \frac{U^{(h)}_x - [U^{(h)}_x]_2}{2} \\
\Delta Y^{(h)}_{n-1} &= \frac{U^{(h)}_y - [U^{(h)}_y]_2}{2} \\
\Delta X^{(h)}_{n+1} &= \frac{U^{(h)}_x + [U^{(h)}_x]_2}{2} \\
\Delta Y^{(h)}_{n+1} &= \frac{U^{(h)}_y + [U^{(h)}_y]_2}{2}
\end{align*}
\]

(2)

where the expression \([\ldots]_m\) denotes “modulo \(m\)”.

Correlation indexes, i.e. the Block Matching Errors must now be evaluated along each trajectory. The two pixel blocks (shaped as in Fig. 5(d)) to be matched are located at the two ends of the trajectory, whose spatial-temporal coordinates are, respectively:

\[
\begin{align*}
(x - \Delta X^{(h)}_{n-1}, y - \Delta Y^{(h)}_{n-1}, t_{n-1}) \\
(x + \Delta X^{(h)}_{n+1}, y + \Delta Y^{(h)}_{n+1}, t_{n+1})
\end{align*}
\]

(3)  (4)

The 9 correlation indexes \(BM E^{(1)}, \ldots, BM E^{(9)}\) are evaluated as the sum of the absolute grey level differences of the 8 pairs of corresponding pixels. They are then weighted, as in Eq. 1 to yield the set of the 9 weighted correlation indexes \(WC I^{(1)}, \ldots, WC I^{(9)}\):

\[
WC I^{(h)} = BM E^{(h)} \times (1 + K \times ((U^{(h)}_x)^2 + (U^{(h)}_y)^2))
\]

(5)

We set then:

\[
\tilde{U}_{best} = \tilde{U}^{(h)} : WC I^{(h)} = \min_h \{WC I^{(h)}\}
\]

(6)
\( \vec{U}_{best} \) is the optimal estimated motion vector for the pixel in position \((x, y, t_n)\).

The same procedure is then repeated for pixel in position \((x + 1, y, t_n)\), setting \( \vec{V} = \vec{U}_{best} \) in Tab. 1.

Applying this mechanism for each pixel of the interpolated frame, we to build a motion vectors field, which will be eventually post-processed by a vector median filter.

To this purpose, a \( 3 \times 3 \) mask scans the vector field. For each pixel \((x, y, t_n)\) we consider the 9 vectors belonging to the mask, which have been estimated for those pixels whose coordinates are:

\[
(x - p, y - q, t_n) \quad \left\{ \begin{array}{l}
p = -1, 0, 1 \\
q = -1, 0, 1
\end{array} \right.
\]

(7)

Denote these vectors with: \((\vec{W}^{(1)}, \ldots, \vec{W}^{(9)})\). The output vector of the median filter is:

\[
\vec{W} = (W_x^{(m)}, W_y^{(m)})
\]

(8)

where

\[
W_x^{(m)} = med [W_x^{(1)}, \ldots, W_x^{(9)}]
\]

(9)

\[
W_y^{(m)} = med [W_y^{(1)}, \ldots, W_y^{(9)}]
\]

(10)

The filtered motion vectors field will be used during the actual interpolation of the luminance levels of the pixels in the new frame \( L(x, y, t_n) \).

The interpolation formulas for the three possible cases when the motion vector is \( \vec{W} = (2i, 2j) \) (no odd components), \( \vec{W} = (2i + 1, 2j) \) (one odd component, e.g. the horizontal one) or \( \vec{W} = (2i + 1, 2j + 1) \) (both components odd) are, respectively:

\[
L(x, y, t_n) = \frac{1}{3}[(L(x - i, y - j, t_{n-1}) + L(x + i, y + j, t_{n+1}))]
\]

(11)

\[
L(x, y, t_n) = \frac{1}{3}[(L(x - i, y - j, t_{n-1}) + L(x + i, y + j, t_{n+1}) +
L(x - i - 1, y - j, t_{n-1}) +
L(x + i + 1, y + j, t_{n+1})]
\]

(12)

\[
L(x, y, t_n) = \frac{1}{4}[(L(x - i, y - j, t_{n-1}) + L(x - i - 1, y - j - 1, t_{n-1}) +
L(x - i - 1, y - j, t_{n-1}) +
L(x + i + 1, y + j, t_{n+1}) +
L(x + i, y + j, t_{n+1}) +
L(x + i + 1, y + j + 1, t_{n+1}) +
L(x + i, y + j + 1, t_{n+1}) +
L(x + i + 1, y + j + 1, t_{n+1})]
\]

(13)
Table 2: Motion compensation example. The table entries show the errors (before and after weighting) associated with the candidate motion vectors at the different horizontal spatial positions of Fig. 7.

### 3.1 An example

The following example shows how the algorithm deals with a simple test pattern. We consider a black object moving three pixels to the right (with a 1 pixel wide shadow, to make the case more realistic) and an overlapping white object, moving two pixels to the left (Fig. 7). In order to make the example simpler, we shall consider only horizontal motion. The matching blocks are shaped as in Fig. 5(d).

The restricted search will then be performed on 3 trajectories only since vertical motion is not considered, but the extension to the full case is straightforward. The results of the matchings are summarized in Table 2: each row in the table corresponds to a column of the interpolated frame in Fig. 7. For each position (numbered from 0 to 16), we calculate the mean absolute error and the weighted correlation index (boldface) along the three candidate motion vectors, assuming that the black object has a grey level of 0, the shade has a level of 10, the white background and the white object have a level of 20. The value of the constant \( K \) (see Eq. 1) has been set to 0.05. It can be seen that the three tested motion vectors are centered around the most correlated direction found for the previous pixel (marked with a brace in Table 2), as explained in Sec. 2.2.4.

Starting from position 3, the algorithm begins to react to the presence of
the moving edge: motion vector \((1, 0)\) is selected and vector \(\vec{V}\) begins to move towards the correct position. This process continues in positions 4 and 5, and interpolation along the selected directions give the following results: grey level for position 3 is 17.5, for position 4 it is 15, for position 5 it is 5, and eventually for position 6 it is 0. At position 6 the algorithm begins to detect the presence of the white object, and a transition phase begins: in position 7 and 8 the algorithm cannot find any direction that clearly shows better correlation, therefore the weighting procedure tends to bring the search back to a more “central” position. For example, in position 7 all the three tested vectors give 60 as a BME; the weighting procedure favours the trajectory nearest to direction \((0, 0)\), namely vector \((1, 0)\). The weighting of correlation indexes therefore makes the algorithm capable of locking on the white object starting on position 9, and in position 10 the detected motion is \((-2, 0)\). In the following positions, since the white object has a uniform color, the weighting procedure moves the searched directions back to around the position \((0, 0)\). If there were details moving in the same direction, the algorithm would stay locked on \((-2, 0)\) instead.

4 Results

The proposed method has been tested on real-world and test sequences, both using single interpolated frames and viewing full converted sequences. The classification and the comparison of performance of different versions of the algorithm is not a trivial task and subjective testing was mainly used due to the structure of the problem itself. It is difficult to find a testing procedure yielding a meaningful numerical performance index, such as, e.g., the mean square error, since the output of the filter has to be different from any of the available original frames. In addition to that, the output of objective testing methods often does not match the subjective impression. More detailed discussion on the above mentioned problem can be found e.g. in [24].

An objective numerical performance figure has been used for the setting of some parameters or the testing of different solutions during the development of the algorithm. The method consisted of considering a frame belonging to a progressive sequence and interpolating it using the two adjacent frames. A mean square error could then be evaluated between the original and the interpolated frame. However, the results are not robust enough to be used as true performance indexes of the algorithm. In many cases, indeed, variations of the MSE due to changes of the parameter affected only the least significant digits, or were not detectable at all, even if the improvement of the picture quality was clear to the viewer. Moreover, the assumption for linear motion reduces the reliability of MSE as a performance index: a sub-pixel displacement error can result in a large MSE, even if the overall quality of the image is good.
Finally, it should be observed that skipping one original frame means doubling the movement speeds to be detected, bringing the displacement of some objects in the sequence out of reach for the motion detection procedure.

Results from the analysis of single frames must then be merged with those of the full sequence viewing. A video sequencer has been used extensively during our tests, enabling us to view interpolated sequences with a scan rate of 75 Hz in real time.

Due to the structure of the problem, progressive sequences were needed as input for our experiments: most tests were performed on sequence “RENATA” (courtesy of RAI), which was originally produced using progressive scan: the scene consists of a model walking in front of a background, followed by the panning of the camera. Other sequences were digitized in interlaced format instead, and were first deinterlaced and then interpolated with our algorithm. This double processing allowed us to evaluate the overall effect of the first two blocks of the transcoding system mentioned in the Introduction (see Fig. 1).

As stated before, it is not possible to compare the interpolated frames with anything “original”. Nevertheless, the overall result of the interpolation can be appreciated in Fig. 8 and Fig. 9. Figure 8 represents an original frame, copied as it was into the interpolated sequence (“y” type frame in Fig. 3), in Fig. 9 the following interpolated “z type” frame is shown: it can be seen that the detail is preserved satisfactorily and almost no blurring is introduced, so that detail flickering is avoided in the interpolated sequence. A critical part of the picture is the strip of background area to the right of the model’s head: this zone corresponds to a step in the motion vector field, since the head is moving to the left and the background to the right. A thin zone of blurring can be observed in the detail of the calendar. The motion of the head itself masks quite well this artifact when the sequence is viewed in real time.

In this zone the weighting of the correlation indexes gave the most satisfactory results, as can be seen in Figures 9 to 12.

Picture 10 shows the result when no weighting of the indexes is performed: the area of blurring to the right of the head is much wider than in Fig. 9. The reduction of the streaking effect obtained by the weighting procedure can be easily spotted by comparing the two test frames represented in Figs 11(a) and 11(b).

In both cases we should concentrate on the background, which moves—in the frames being examined—two pixels per frame to the right. A white pixel indicates that the correct motion has been estimated for the background, and the optimal result would be having full white in zones corresponding to details. The streaking effect is clearly visible in Fig. 11(a), in the form of black horizontal lines, the number of which is greatly reduced if weighting is performed on the correlation indexes (Fig. 11(b)). In both images, some isolated black pixels indicate the presence of impulsive errors in the motion detection, mainly due to the presence of noise in the original frames.
Such test frames can be used also to view the effect of the vector median filtering on the field: Figs. 12(a) and 12(b) show the correctness of the motion detection after median filtering has been performed on the fields to which Figs. 11(a) and 11(b) respectively refer.

It can be seen from Fig. 12(b) that the combined action of the weighting procedure and the vector median filter eliminates both the streaking effect and the impulsive errors, giving a quite reliable motion estimation. The remaining black pixels correspond mostly to areas where no detail is present, and the weighting of indexes results in (0, 0) being chosen as the interpolation direction.

The positive effects of the weighting procedure can be observed also in Figs. 13(a) and 13(b). In these pictures, a part of a car door is represented (the handle and the lock are visible). A thin detail is moving on uniform background: in the former image, obtained with no weighting, the interpolation fails quite often and the vertical bar is damaged; in the latter the artifact is no longer present.

5 Conclusions

In this paper a frame interpolation algorithm for frame rate up-conversion is proposed. Although the algorithm has been designed for the 1.5 conversion factor (50 Hz to 75 Hz), minor changes can make the method suitable for other conversions, such as 50 to 100 Hz, 50 to 62.5 Hz or 60 to 75 Hz. The task of keeping computational complexity as low as possible has been given the highest priority during the development of this work and simulations have shown that the performance of this algorithm is an acceptable compromise between simplicity and image quality. Since the motion compensation procedure has proved to be the most requiring in terms of computational weight, the compromise mainly affects this part of the method, resulting in a limitation in the range of movements that can be detected and correctly compensated for. The artifacts introduced in fast motion areas are – at least partially – masked by the motion itself. Errors which can occur even when the speed is within the measurable range, are mainly due to noise and critical luminance patterns moving at certain speeds. In the latter case, phenomena of “spatio-temporal aliasing” may occur: such problems, on the other side, often affect also more sophisticated methods.

The most obvious development of the presented study is to extend the method to the processing of the chrominance components of the signal. A first step in this direction has already been made by interpolating the chrominance data according to the motion vector field obtained from luminance data. The result are quite encouraging: the presence of colours partially masks artifacts due to the above mentioned errors, and in general increases the overall image quality. A further development will consist of exploiting the knowledge of
chrominance data in order to obtain more reliable information about motion: in particular, a check on the consistency of the data in the three signal components (Y,U,V or R,G,B) is likely to reduce the occurrence of errors due to noise in luminance data.

A hardware implementation of this algorithm will be dealt with in the future: modifications of the presented method might therefore be suggested during the optimization process required in the hardware design.

Acknowledgments

The authors would like to thank Dr. Marino Balanza for his personal contribution to this research, in particular on the issues related to the computational complexity of the method.
The authors would also like to thank the reviewers for their valuable comments and suggestions.

This work has been partially supported by the ESPRIT project NAT 7130.

References


[23] SGS–Thompson Microelectronics, STI3220

Figure captions

Figure 1: Block scheme of a suggested transcoding system for TV image sequences.

Figure 2: An example of the temporal distribution of frames in 50 to 75 Hz frame rate conversion.

Figure 3: The “time-shifted” frame distribution. The x- and y-frames are replicas of the original frames with a small temporal displacement and the z-frames are interpolated from two original frames.

Figure 4: The temporal distribution of frames for conversion from 50 to 62.5 Hz.

Figure 5: Examples of tested block shapes.

Figure 6: Motion compensated linear interpolation.

Figure 7: An example: two overlapping objects.

Figure 8: An original frame (number 94) of sequence RENA T A

Figure 9: An interpolated frame (number 95) of sequence RENA T A

Figure 10: A detail of frame 95 as obtained with no weighting of the correlation indexes: the “streaking” effect is clearly visible to the right of the model’s head.

Figure 11: A white pixel indicates correct motion detection on the background. The “streaking” effect is visible in picture (a) to the right of the model’s head. The weighting of the correlation indexes improves the correctness of the motion detection: in picture (b) “streaking” effect is less noticeable.

Figure 12: (a) Effect of the vector median filtering on the field obtained without weighting of the correlation indexes (Fig. 11(a)). (b) Effect of the vector median filtering on the field obtained with weighting of the correlation indexes (Fig. 11(b)): in zones where detail is present the motion detection is fully reliable (white pixels)

Figure 13: (a) Errors due to a failure in the motion detection procedure without weighting of correlation indexes (see Section 3.2.2). (b) The same detail of Fig. 13(a) processed with weighted correlation indexes: artifacts have
been eliminated.

**Table 1:** The nine motion vectors to be investigated for the interpolation of pixel in position \((x, y)\) when \(\hat{V} = (V_x, V_y)\) is the estimated motion vector for pixel in position \((x - 1, y)\).

**Table 2:** Motion compensation example. The table entries show the errors (before and after weighting) associated with the candidate motion vectors at the different horizontal spatial positions of Fig. 7.
Figure 4:

Figure 5:

(a) single pixel  (b) 5 point cross  (c) 7 point cross  (d) 8 point quincunx

trajectory end point

Figure 6:
Figure 7:

Figure 8:
Figure 9:

Figure 10:

Figure 11:

Figure 12:

Figure 13: