A low-cost implementation of super-resolution based on a video encoder

Conference Paper · December 2002

DOI: 10.1109/IECON.2002.1185489 · Source: IEEE Xplore

5 authors, including:

Gustavo Marrero Callico
Universidad de Las Palmas de Gran Canaria
148 PUBLICATIONS **1,001** CITATIONS
SEE PROFILE

Antonio Nunez
Universidad de Las Palmas de Gran Canaria
141 PUBLICATIONS **462** CITATIONS
SEE PROFILE

A Low-Cost Implementation of Super-Resolution based on a Video Encoder

Gustavo M. Callicó, Antonio Núñez,
Applied Microelectronics Research Institute, ULPGC
Department of Electronic and Automatic Engineering
Canary Islands, Spain
gustavo@iuma.ulpgc.es

Rafael Peset Llopis, Ramanathan Sethuraman,
Marc Op de Beeck
Philips Research Laboratories Eindhoven,
Royal Philips Eindhoven, The Netherlands
rafael.peset.llopis@philips.com

Abstract – **This paper presents an approach to improve the quality of digital images over the sensor resolution using super-resolution techniques. In order to obtain a feasible low cost implementation, the resources have been restricted to those that can be found in a generic video encoder, i.e.: the motion estimator, the motion compensator, image loop memory, etc. The super-resolution system has been implemented over a co-design platform developed by the Philips Research Laboratories in Eindhoven, while performing minimal changes on the overall hardware architecture. Nevertheless, this methodology can easily be extended to any generic video encoder architecture. The results show important improvements in the image quality, assuming that sufficient sample data is available. Based on these results, some generalizations can be made about the impact of the sampling process on the quality of the super-resolution image.**

I. INTRODUCTION

The straightforward way to increase the resolution of an image is to use higher resolution sensors, at the expenses of higher cost. This however results in a decrease the size of the active pixel area where the light integration is performed. As less amount of light reaches the sensor it will be less sensitive to the shot noise. It has been estimated that the minimum photo-sensors size is around 50µm^2 [1], a limit that has already been reached by the CCD technology. One solution to this problem is to increase the resolution using algorithms such as the super-resolution (SR), wherein high-resolution images are obtained with low-resolution sensors at low costs. SR is also a smart way to perform image zooming without using mechanical parts to move the lenses.

This paper addresses a low-cost solution for the implementation of SR algorithms over SOC (System-On-Chip) platforms in order to achieve high-quality image improvements. Low-cost constrains are accomplished in the sense that SR is performed without developing a specific hardware, but re-using a video encoder. This encoder can be used either in compression mode or in SR mode.

Super-resolution can be defined as a technique to increase the image resolution of pictures by exploiting the spatio-temporal redundancy of data in several displaced images. Although the SR algorithm has been implemented on an encoder architecture developed by Philips Research, the same SR algorithm can adapted to other hybrid video encoder platforms.

A. Super-resolution algorithms

It must be pointed out that, in order to obtain significant improvements in the resulting SR image, some amount of aliasing in the input low-resolution images must be provided. In other words, if all high frequency information is removed from the input images (for instance by using an optical low-pass filter), it will be impossible to recover the edge details contained in the high frequencies.

From the first frequency domain based SR algorithm, initially proposed by Huang and Tsay in 1984 [2], several SR algorithms have been proposed with diverse results. Although unlike [2] this approach is not performed in the frequency domain, we maintain the general scheme of registration and restoration. This work is more related to [3] in the sense that it is performed in the spatial domain and starts from a set of low-resolution displaced images, but without a blur correction and without the need of a precise knowledge of the motion among images. As in [4] we take into account sub-pixel displacements and like [5] we use the approach of multiple image restoration. Nevertheless, all the previous methods assume global and uniform translation among the different pictures, whereas this work makes no assumption about the motion among scenes.

The algorithm exposed in this paper is a modified version of [6], adapted to be executed inside a real video encoder, i.e. restricting the operators to those ones that can be found in such kind of platforms. The new added operators can be implemented on a block basis and are easy to accommodate inside the existing co-processors in order to minimize the impact on the overall architecture.

B. The hybrid video encoder platform

This work is positioned in the frame of '*SOC platform oriented design*', in the sense that the hardware/software platform is already established and we seek to modify the existing platform as less as possible (low cost constrains) in order to map the SR algorithm. Although we have used an existing hybrid video encoder developed by Philips [7,8] the SR algorithm can probably fit in any other video encoder.

The architecture used in Philips is shown in Fig. 1. The software tasks are executed on an ARM processor and the hardware tasks are executed on the VLIW processors (namely, pixel processor, motion estimator processor, texture processor and stream processor). The pixel processor (PP) communicates with the pixel-domain (image sensor or display) and performs line to stripe (16 lines) conversion. The motion estimator processor (MEP) evaluates a set of candidate vectors received from software and selects the best vector for full, half and quarter pixel refinements. The output of the MEP consists of motion vectors, sum-of-absolute-difference (SAD) values, and intra metrics. This information is used in software (running on ARM) to determine the encoding approach for the current macro-block (MB).

The texture processor (TP) performs the encoding of MBs and stores the decoded MBs in the loop memory. The output of the TP consists of VLE (variable length encode) codes for the DCT coefficients of the current MB. The stream processor (SP) packs the VLE codes generated by the TP for coefficients and VLE codes for headers generated by software on the ARM.

II. SUPER-RESOLUTION BASICS

Calling *f(x,y,t)* to the low resolution input image, all the input sub-system effects (lenses filtering, chromatic irregularities, sample distortions, information loss due to format conversions, system blur, etc.) will be included in *h(x,y)*. So, assuming linear effects in lens, sensors, and colour processing the input to the algorithm will be the two dimensional convolution expressed in (1).

$$g(x,y,t) = f(x,y,t) ** h(x,y) \qquad (1)$$

Calling $S(\breve{x},\breve{y},t)$ to the image obtained after applying the SR algorithm, and $SR(\breve{x},\breve{y})$ to the SR algorithm itself, they are related as indicated in (2).

$$S(\breve{x},\breve{y},t) = g(\breve{x},\breve{y},t) ** SR(\breve{x},\breve{y}) \qquad (2)$$

These relationships are summarized in Fig. 2.a concerning to the real system and are simplified in Fig. 2.b.

The algorithm starts supposing that a number '*p*' of low resolution images of size N×M are available as *g(x,y,t$_i$)*, where '*t$_i$*' denotes the sample time of the image. As the algorithm only refers the last '*p*' images, from now on the index '*l*' will be used, defined as *l=i mod p*, to refer the images inside the algorithm. For clearness, the memory image $g_l'(x,y)$ will be used to store the input image that the algorithm is going to use, and it is linked to *g(x,y,t$_i$)* through (3). If we call $\bar{g}_l'(x,y)$ to the average input image, as given in (4), the average error for the first iteration is obtained by computing the differences between this average image and every input image, as show in (5).

$$g_l'(x,y) = g(x,y,t_i) \quad / \quad l = i \bmod p \qquad (3)$$

$$\bar{g}'(x,y) = \frac{1}{p}\sum_{l=0}^{p-1}\left(\frac{1}{N\cdot M}\cdot\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} g_l'(i,j)\right) \quad , \quad \forall\, x,y \qquad (4)$$

$$e_l(x,y)^{(1)} = g_l'(x,y) - \bar{g}'(x,y) \quad , \quad l = 0\,..\,(p-1) \qquad (5)$$

This error must be transformed to high resolution coordinates through, by example, a nearest neighbour replication interpolator (6). As the missing pixels are going to be recovered by the SR algorithm it is not worth to use a higher quality interpolator, which will be slower and more expensive.

$$e_l(\breve{x},\breve{y})^{(1)} = \boldsymbol{upsample}\left(e_l(x,y)^{(1)}\right) \quad , \quad l = 0\,..\,(p-1) \qquad (6)$$

Once in high resolution the error must be adjusted to the reference frame, shifting the image $\Delta\delta_l(x,y)^{(1)}_{(fr2ref)}$ and $\Delta\lambda_l(x,y)^{(1)}_{(fr2ref)}$ amounts in the horizontal and vertical coordinates respectively. In principle, these displacements are applied to every pixel individually, but this will depend upon the motion estimation technique employed. When all the errors have been adjusted to the reference, they are averaged and this average will be taken as the first update of the SR image, as shown in (7).

$$S_0(\breve{x},\breve{y})^{(1)} = \frac{1}{p}\cdot\sum_{l=0}^{p-1} e_l\left(\breve{x} + \Delta\delta_l(x,y)^{(1)}_{(fr2ref)}, \breve{y} + \Delta\lambda_l(x,y)^{(1)}_{(fr2ref)}\right)^{(1)} \qquad (7)$$

Equation (7) reflects the result of the first iteration. For instance, $S_0(\breve{x},\breve{y})^{(1)}$ is the first version of the SR image, corresponding to *t=t$_0$*, and it will be upgraded in every iteration. The nth iteration starts from that image and begins obtaining by decimation a low resolution version followed by the computation of the displacements between every one of these inputs images and this decimated image and vice versa, i.e. between the decimated image and the input images. In that manner the displacements of the nth iteration will be available: $\Delta\delta_l(x,y)^{(n)}_{(fr2ref)}$, $\Delta\lambda_l(x,y)^{(n)}_{(fr2ref)}$, $\Delta\delta_l(x,y)^{(n)}_{(ref2ref)}$ and $\Delta\lambda_l(x,y)^{(n)}_{(ref2fr)}$. The low resolution version of the image obtained in high resolution is given by (8).

$$S_0(x,y)^{(n)} = \boldsymbol{downsample}\left(S_0(\breve{x},\breve{y})^{(n-1)}\right) \qquad (8)$$

The next step is to compensate the motion of the high-resolution image towards the input frames using the displacements $\Delta\delta_l(x,y)^{(n)}_{(ref2ref)}$ and $\Delta\lambda_l(x,y)^{(n)}_{(ref2fr)}$, converting then to low-resolution and getting the error respecting to every input image, as shown in (9).

$$e_l(x,y)^{(n)} = g_l'(x,y) - S_0\left(x + \Delta\delta_l(x,y)^{(n)}_{(ref2fr)}, y + \Delta\lambda_l(x,y)^{(n)}_{ref2fr}\right)^{(n)} \qquad (9)$$
$$l = 0\,..\,(p-1)$$

This low-resolution error must be taken again to high resolution through interpolation and compensate it motion again towards the reference. The average of these *'p'* errors constitutes the nth incremental update of the high-resolution image, as shown in (10).

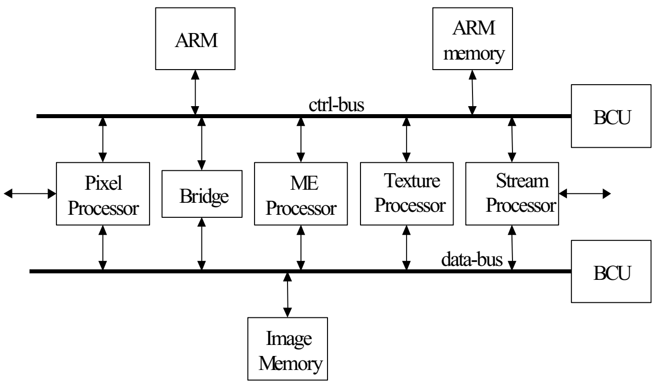

Fig. 1. Architecture for the Multi-Standard Video/Image Codec developed in Philips Research.

$$S_0(\breve{x},\breve{y})^{(n)} = S_0(\breve{x},\breve{y})^{(n-1)} + \frac{1}{p}\cdot\sum_{l=0}^{p-1} e_l\left(\breve{x}+\Delta\delta_l(x,y)^{(n)}_{(fr2ref)},\breve{y}+\Delta\lambda_l(x,y)^{(n)}_{(fr2ref)}\right)^{(n)} \qquad (10)$$

The convergence is reached when the changes in the average error are negligible, i.e. when the variance of the average error is below of a certain threshold determined in an empirical way.

Once the SR image is obtained for time t_0 with the first *'p'* images, the process must be repeated with the next *'p'* images to obtain the next SR image using a previously established number of iterations or iterating up to convergence. Acquire a SR image implies the use of 'p' low resolution images, and so, at the instant t_i, the SR image *k=integer(i/p)* will be generated. In such case, equation (10) must be generalized in (11).

$$S_k(\breve{x},\breve{y})^{(n)} = S_k(\breve{x},\breve{y})^{(n-1)} + \frac{1}{p}\cdot\sum_{l=0}^{p-1} e_l\left(\breve{x}+\Delta\delta_l(x,y)^{(n)}_{(fr2ref)},\breve{y}+\Delta\lambda_l(x,y)^{(n)}_{(fr2ref)}\right)^{(n)} \qquad (11)$$

This last and more general equation (11) reflects the SR image at instant *k* as a combination of *'p'* low resolution images after *'n'* iterations.

III. IMPLEMENTATION ON A VIDEO ENCODER

A. Algorithmic modifications

Instead of starting with an average image, as indicated in equation (4) it is faster and easier to start with an up-sampled version of the first low-resolution input image. Therefore, the final SR image will be aligned with the first image, whose motion is well known.

The next modifications are intended to adapt the algorithm in terms of basic actions that can be easily implemented on a video encoder. The straightforward way in a video encoder to determine the displacements between pictures is by using the motion estimator, which is normally used to code pictures of types P or B in MPEG. Further, as the motion estimation is one of the most sensitive steps of the SR algorithms it has been decided to use a quarter-pixel precision for it. Consequently, for displacing a picture, the motion compensator is also prepare to work with the same precision. The main problem is that usually, SR algorithms are intended to work on a pixel basis and the motion blocks of the compressor work on a block basis. This mismatch will produce a quality degradation when the motion does not match the block sizes, i.e. when the object is smaller that the block size or when more than one moving objects exists inside the block.

It is a basic principle of binary arithmetic that the addition of two N-bit numbers produce an N+1 bit number. This has been an important problem source for adapting the algorithm. Inside the encoder architecture, every pixel is represented as an eight-bit number. Inside the co-processors, the image values can be processed using more bits (block level processing), but the result must be stored again in an eight-bit image memory. For video compression, this is not a significant problem, but when reusing the same architecture for arithmetic image processing, we face the limitation that any intermediate result must be stored in 8 bit. Because of this problem, we have adopted the following solutions:

- Perform as many arithmetic operations as possible at the block-level, with higher precision.
- Rearrange the arithmetic operations in such a way that, when storing the intermediate results, these are bounded, as close as possible, to an 8 bits number.

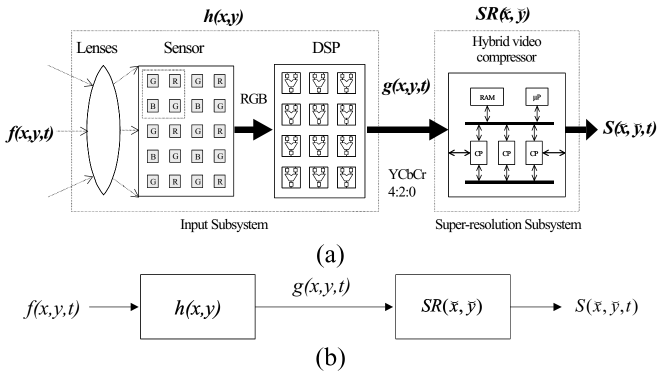

Fig. 2. Scheme of the real overall system (a) and the simplified model (b) together with the used terminology.

All memories are eight bits wide, except HR_A, which must be nine bits wide (it has been highlighted in Fig. 3 by showing HR_A in boldface). This memory must be wider because it must store arithmetical results that can overflow 8 bits, especially in the beginning of the iterations.

The implemented algorithm is shown in Fig.3 in pseudo-code. LR_I[] are the low-resolution input frames, HR_B is the SR image result, LR_B is a low-resolution version of HR_B; HR_T is a temporal high-resolution image to avoid overlaps while performing the motion compensation; HR_A accumulates the average error that will be used as an update for the SR image; HR_S stores the result of displacing the SR image and the errors between that shifted image and the up-sample input image, and finally, MV_ref2fr[] and MV_fr2ref[] are motion vectors memories to store the motion between the reference and the input frames and between the input frames and the reference respectively. The number of frames to be combined in a high-resolution image is 'nr_frames' and 'nr_iterations' is the maximum number of pre-established iterations. The iterative process can be aborted when convergence is achieved, i.e. when the variance of HR_A is below a certain threshold.

As the motion estimation is the most expensive operation in terms of time and power consumption, we have performed the optimization of assuming the motion between the reference and the frame as the inverse of the motion between the frame and the reference.

It is interesting to highlight that the presence of aliasing in the low-resolution input images largely decreases the accuracy of the motion vectors. In that sense, better results are obtained by filtering the images before performing the motion estimation. A spatial low-pass filter has been used.

B. Implementation issues

Table 1 shows the memory requirements of the algorithm for different input sizes. It must be noticed that the output image will be four times larger (2× in horizontal and vertical direction) thatn the input image.

These requirements include four input memories and a buffer of three stripes each of sixteen pixels high, for reading the input images. They also take into account the chrominance and the extra bit of HR_A. The total memory requirements, as a function of macro-blocks is $MB\_y \cdot (6724 \cdot MB\_x+4608)$ expressed in bytes, where MB_x and MB_y are respectively the number of MBs in the horizontal and vertical directions.

To perform the up-sample and down-sample operations it is necessary to include in hardware an upsampling and downsampling blocks, where the operations are performed on a block basics. Upsampling is performed by nearest neighbour replication from an 8×8 block to a 16×16 MB. Downsampling is achieved by averaging four neighbouring pixels to a single one, passing from a 16×16 MB to an 8×8 block.

The motion estimation and motion compensations tasks are performed using the motion estimator and the motion compensator blocks, but have been modified to work in quarter pixel precision. The arithmetic operations such as additions, subtractions and arithmetic shifts are implemented on the texture processor. The overall control is done in software, on the ARM.

IV. RESULTS

A. Experimental setup

Some synthetic sequences have been generated with the objective of assessing the algorithm itself, independently of the image peculiarities, and to enable the measure of reliable metrics. Those sequences share the following characteristics: Firstly, in order to isolate the metrics from the image peculiarities, the same frame has been replicated all over the sequence. Thus, any change in the quality will only be due to the algorithm processing. Secondly, the displacements have been randomly generated, except for the first image of the low-resolution input set, which always will be the zero vector, because that frame will be used as the reference in the peak signal to noise ratio (PSNR) computation. Thirdly, in order to avoid the border effect when moving the frame, we have opted for using a large image format and removing the borders in the metrics computation. Fig. 4 depicts the experimental setup to generate the test sequences. A set of 40 input frames from 40 random motion vectors has been generated. This synthetic sequence has been used as the input for the SR process. Initially the algorithm performed 80 iterations over every four input frames set. The result was a ten frame sequence.

```
LR_B = LR_I[0]
HR_B = Upsample(LR_I[0])

FOR it = 0 .. nr_iterations-1
    IF (it $\neq$ 0)   LR_B = Downsample(HR_B)
    MV_fr2ref[0] = 0
    MV_ref2fr[0] = 0
    FOR fr = 1 .. nr_frames-1
        MV_fr2ref[fr] = Calc_Motion_Estimation (LR_I[fr], LR_B)
        MV_ref2fr[fr] = - MV_fr2ref[fr]
        MV_fr2ref[fr] = 2 $\times$ MV_fr2ref[fr]
        MV_ref2fr[fr] = 2 $\times$ MV_ref2fr[fr]
    END FOR
    HR_A = 0
    FOR fr = 0 .. nr_frames-1
        HR_S = Motion_Compensation (HR_B, MV_ref2fr[fr])
        HR_S = Upsample(LR_I[fr])/2 – HR_S/2
        HR_T = Motion_Compensation (HR_S, MV_fr2ref[fr])
        HR_A = HR_A + HR_T/2
    END FOR
    HR_B = HR_B + HR_A
    variance = variance(HR_A)
    If (variance < variance_threshold) Then break
END FOR
```

Fig. 3. Pseudo-code of the SR algorithm implemented on the video encoder.

TABLE I
MEMORY REQUIREMENTS FOR DIFFERENT SIZES OF THE INPUT IMAGE

Size	MB_x	MB_y	Memory (Kbytes)
SQCIF (128×96)	8	6	315.19
QCIF (176×144)	11	9	650.07
CIF (352×288)	22	18	2,600.30
VGA (640×480)	40	30	7,879.69
4CIF (704×576)	44	36	10,401.19

The displacements introduced in the VGA images in pixel units are reflected in the low-resolution input pictures divided by four, i.e. in quarter pixel units. This being the precision of the motion estimator, we can compare the real (artificially introduced) displacements with the ones delivered by the motion estimator.

B. Quality measures

Fig. 5 (a) shows the reference picture 'KANTOOR' together with the sub-sampled sequences that constitute the input low-resolution sequence (b) and the nearest neighbour (c) and bilinear interpolations (d) obtained from the first low-resolution frame.

The metrics of the pictures are always compared to the corresponding metrics of the bilinear and nearest neighbor replication interpolations. The interpolation is another way to increase the image size, and thus it must be taken into account as an alternative and faster way to SR. The main difference between interpolation and SR is that the later adds new information from other images while the former only uses information from the same picture. The interpolation PSNR will be the lower bound: a PSNR above the interpolation level implies SR improvements. Fig. 7 shows the luminance PSNR evolution of each frame during the iterations.

From the chart, it is noticeable that for certain frames the quality rises up to a maximum value as the number of iteration increases. This is the expected result, but for some frames, the quality starts to rise and after a few iterations it drastically drops. The reason of this unexpected result is that the displacements were randomly generated and so, the samples presented in each frame are randomly distributed. If the samples contain all the original information (fragmented over the four input frames) then the SR process will be able to properly reconstruct the image. If some data is missing, then the SR process will try to adapt the SR image to the available input set, including the missing data that will be taken as zeroes.

Depending on the missing data, a higher or lower PSNR will be obtained, decreasing below the interpolation level (frames 7 and 9) when the available data are clearly insufficient. In Fig. 8 we have made a classification of the SR frame depending on the available input samples. The best cases are frames of type 'a' where all the samples are present and the worst cases are frames of type 'd' where four equivalent motion vectors were generated picking up the same sample positions four times. In Fig. 7 the type of each frame is shown in bold face the type of each frame. At the light of the possibility of not having all the necessary data available, it is better to stop the iterative process after a few initial iterations. After examining several sequences, it seems that eight iterations is a good trade-off between quality and computation effort for the average case. The rest of the charts will only show the final metrics reached after eight iterations.

If all input data is available, then a PSNR of 34.56 dB for frame number four can be reached. Fig. 6 shows the resulting frame number 4 (of type 'a') in spatial and frequency domains together with the associated errors w.r.t. the reference. It is clearly appreciated that the low-frequencies exhibit lower errors than the higher frequencies. The reconstruction process tries to recover as much high resolution frequencies as possible, but the low-frequency information is easier to recover, mainly because it was almost all present prior to the SR reconstruction process. After eight iterations almost all the frames exhibit PSNR above the interpolation levels (Fig. 9).

V. CONCLUSIONS

We have shown that it is possible to obtain SR improvements using a generic video encoder, performing minimal changes on the architecture such as using sub-pixel motion estimation, enough loop memory, some additional block-level operator and a 9 bits accumulator. If all the arithmetic operations can be performed at the block level, then a 9 bits wide memory will be unnecessary.

It is not always possible to achieve SR improvements. If all the sampled data is not present (this can occur in non-type 'a' frames), then the quality will decrease with the number of iterations. In that sense, it is preferable to limit the number of iterations, as shown in Fig. 7. With this approach, the quality of the SRA always increases with iterations. This principle is related with the real acquisition process in the following way: to obtain SR improvement, sufficient degree of freedom in the sensor must be provided to guarantee a random number of samples to fit in the new high-resolution grid. If the movement of the sensor is limited (for instance, by the use of a tripod or some other clamping system) some pixel positions will never be sampled.

Although low-cost constraints have been accomplished, in the sense that we have reused a hybrid video encoder instead of developing a specific SR system, it will be desirable to achieve real time constraints and to reduce the memory requirements. In this sense, although iterative algorithms show a good behaviour and robustness in presence of noise and/or inaccurate motion knowledge, they are not probably the best option for real time performance.

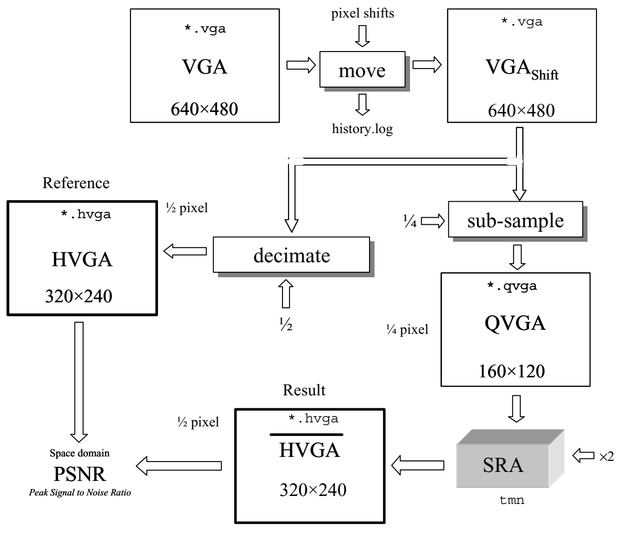

Fig. 4. Experimental setup for the test sequences generation.

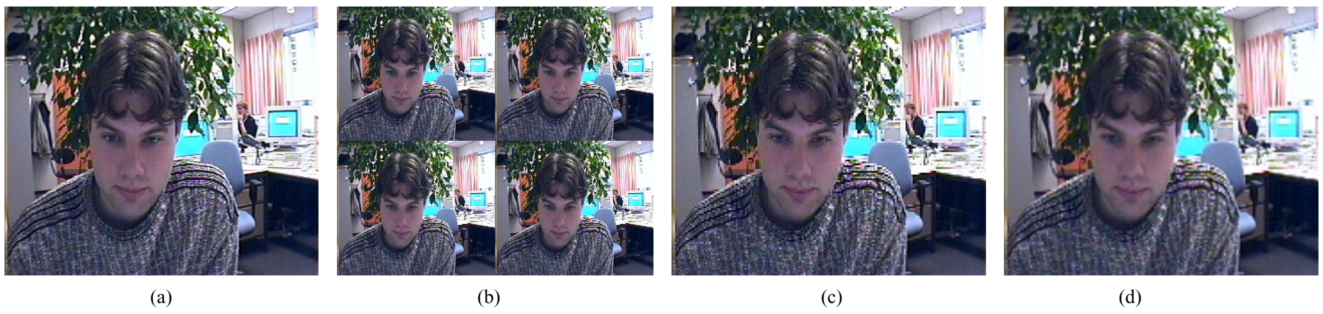

Fig. 5. Reference 'KANTOOR' picture in HVGA (a), the low-resolution input sequence derived from it (b) and the nearest neighbour (c) and bilinear interpolations (d).

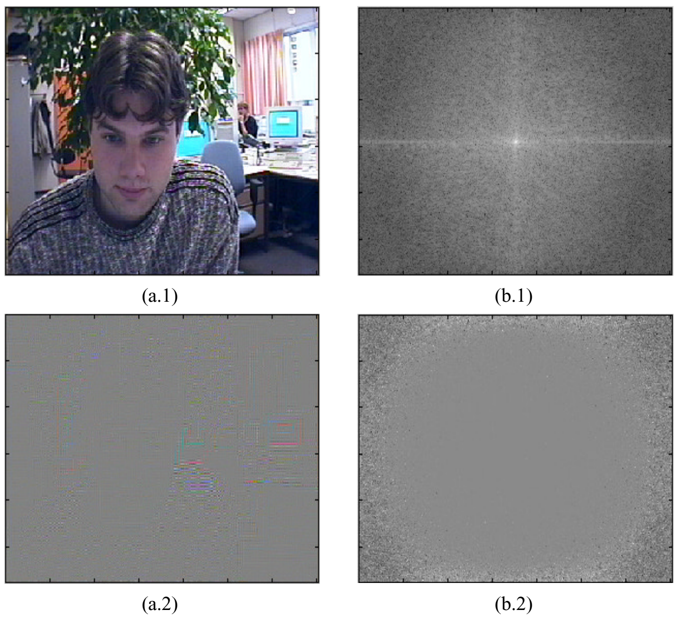

Fig. 6. Frame number 4 in the spatial domain (a) and in the frequency domain in magnitude (b), with them associated error (2).

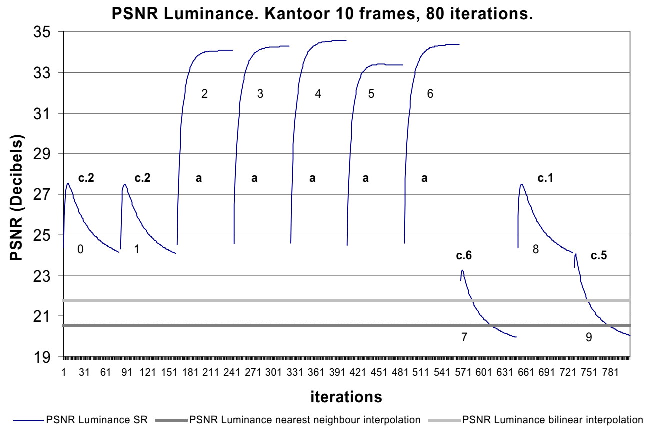

Fig. 7. Luminance PSNR for 80 iterations of the KANTOOR sequence.

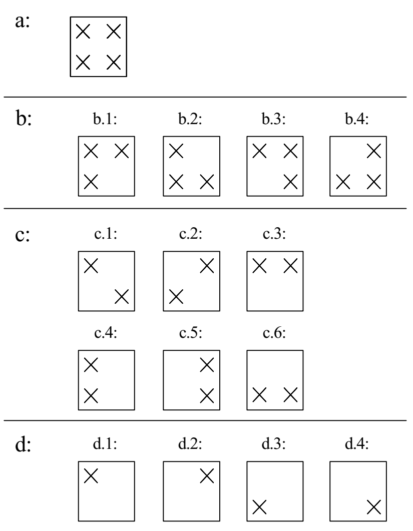

Fig. 8. SR frame classification depending on the available samples.

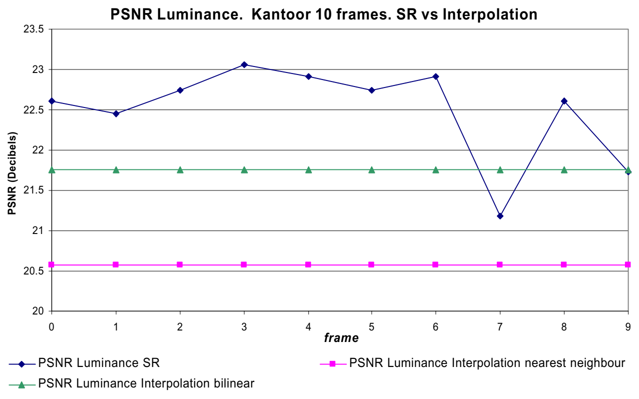

Fig. 9. PSNR of the luminance for the frame 4 after 8 iterations.

VI. ACKNOWLEDGEMENTS

The authors want to thank Richard Kleihorst for the contribution to this work.

VII. REFERENCES

[1] T. Komatsu, T. Igarashi, K. Aizawa, T. Saito, "Very high resolution imaging scheme with multiple different-aperture cameras," *Signal Processing: Image Communication* vol. 5, pp. 511-526, Dec. 1993.

[2] T. S. Huang and R. Y. Tsay, "Multiple Frame Image Restoration and Registration," *Advances In Computer Vision and Image Processing* (Ed. -T. S. Huang), vol. 1, JAI Press Inc., Greenwich, CT, 1984, pp. 317-339.

[3] H. Ur and D. Gross, "Improved Resolution from Sub-pixel Shifted Pictures," *CVGIP: Graphical Models and image Processing*, vol. 54, pp. 181-186, March 1992.

[4] Lucas J. Van Vliet and Cris L. Luengo Hendriks, "Improving spatial resolution in exchange of temporal resolution in aliased image sequences," in proc. of *11th Scandinavian Conf. On Image Analysis*, Kaugerlussauaq, Greenland, pp. 493-499, 1999.

[5] C. Srinivas and M. D. Srinath, "A Stochastic Model-Based Approach for Simultaneous Restoration I-Multiple Miss-registered Images," *SPIE*, vol. 1360, pp. 1416-1427, 1990.

[6] Marc J. Op De Beeck and Richard P. Kleihorst, "Super-Resolution of Regions of Interest in a Hybrid Video Encoder," *Philips conference on DSP*, 1999.

[7] R. Peset Llopis, M. Oosterhuis, S. Ramanathan, P. Lippens, R. Kleihorst, R. van der Vleuten and J. Lin, "A Low-Cost Low-Power H.263 Video Encoder for Mobile Applications," *Second International Symposium on Mobile Multimedia Systems & Applications*, Delf, The Netherlands, Nov. 2000.

[8] R. Peset Llopis, M. Oosterhuis, S. Ramanathan, P. Lippens, A. van der Werf, S. Maul and J. Lin, "HW-SW Codesign and Verification of a Multi-Standard Video and Image Codec," *IEEE ISQED*, San Jose, California, March 2001, pp. 393-398.