

High Frame Rate Up Conversion

Ana Bertran

3/13/2004

Project Description and Motivation

The following project evaluates different methods for frame rate up conversion in the case where we want to go from a low frame rate to a high one. We have assumed simple translational motion in the x,y directions in order to be able to complete the project within the time-frame allocated. For dealing with occlusions, it has been further assumed that the motion is in x but it will be explained how that can be changed for including y motion.

Frame rate up conversion has applications from converting between standards (ex, PAL to NTSC) to low bit rate compression for video-conferencing, video-phone and video games. In the first set of applications one can use simple up-converting techniques such as linear interpolation between frames or sample and hold, without the user noticing many artifacts due to the small differences between the frame rates. However, for the second set of applications, going from low bit rates to high ones, the simple interpolation techniques will introduce significant artifacts. In the case of sample and hold normally smooth motion will appear jerky, choppy in the places where motion has occurred and we will be able to distinguish a beat in between the frames. For example if we are going from 10 fps to 30 fps we will see frame 1 at position (x_0, y_0) for 1/10th of a second and all of a sudden it will jump to position $(x_0 + d_x, y_0 + d_y)$, where d_x and d_y are the amount of displacement between the original frames 1 and 4. In the case of simple linear interpolation the image will consist of a linearly interpolated combination of frames 1 and 4 so if we have motion this will result in the image containing part of the object at the location it was in 1 and part of it at the location in 4, thus the image will appear blurry where there has been motion and we will see ghost artifacts.

These two situations can be clearly appreciated in the following examples, where the original image contained a rectangle moving according to:

$$x = v_x * t + \frac{1}{2} * a_x * t^2 + x_{offset} \quad (1)$$

$$y = v_y * t + \frac{1}{2} * a_y * t^2 + y_{offset} \quad (2)$$

under a fixed background.

Figure 1 shows the up sampled sequence (30 frames per second) using the linear interpolation method:

Linear Interpolation

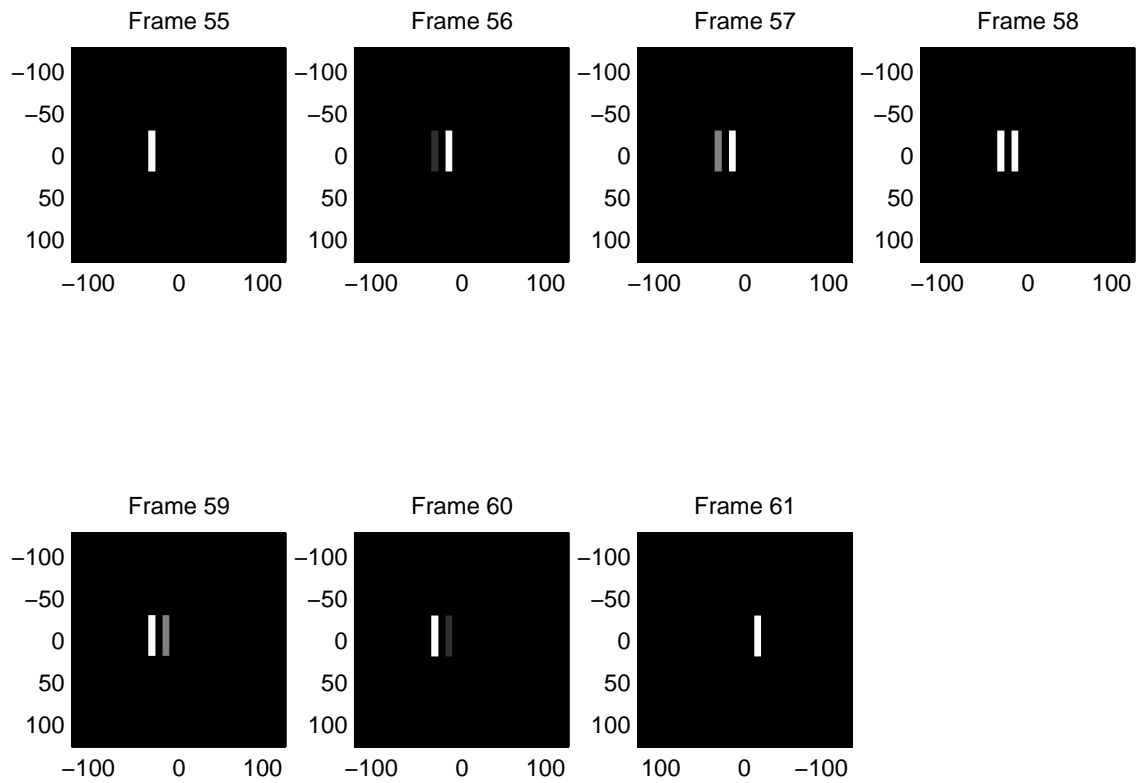


Figure 1: Results of applying linear interpolation across frames.

In the figure we can see ghost artifacts appearing because we are averaging between frames that have the rectangle in different positions.

As shown in Fig. 2 sample and hold results in jumps in the motion of the rectangle. Thus the motion is no longer smooth.

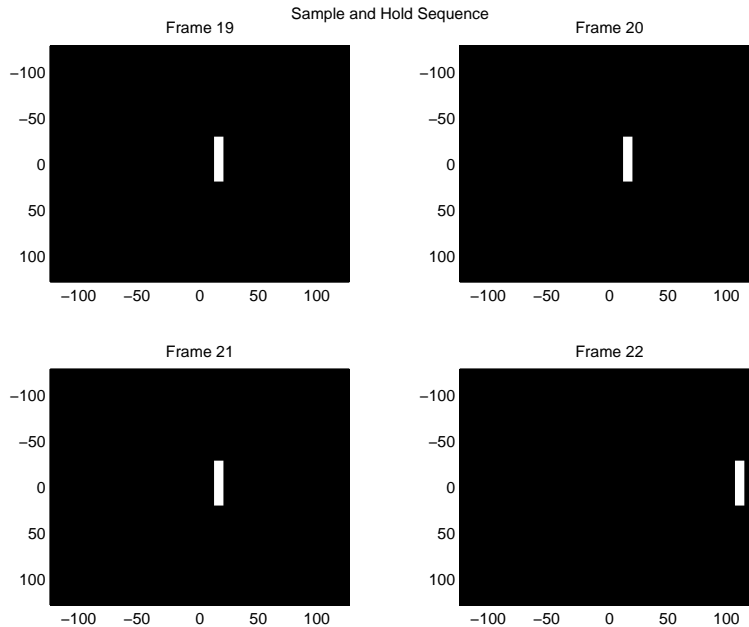


Figure 2: Results of applying sample and hold.

To account for these problems I will describe an algorithm that accounts for the motion through motion-compensated interpolation. This method is not without errors because it can only capture linear motion between frames and thus will fail to capture any acceleration or other higher order terms between frames. Also our motion could fall in between pixels creating additional errors. Errors can also be introduced due to the difficult problem of occlusions. If pixels are occluded (covered or uncovered) between frames it becomes difficult to interpolate between the frames. The problem of occlusions will be covered in this report. In order to account for acceleration between frames a fourth method for frame rate up conversion will be studied, we will call it motion compensation with acceleration. It basically looks at the motion vectors for the next and previous sets of frames to trace a motion trajectory curve and interpolate along it.

To compare the different models I will use a PSNR calculations of the interpolated sequence verses the original (before down sampling, on an image which has an object moving according to a sine wave equation against a fixed background). The motion of the object is described in Fig. 3 if we use a down sampling factor of 6:

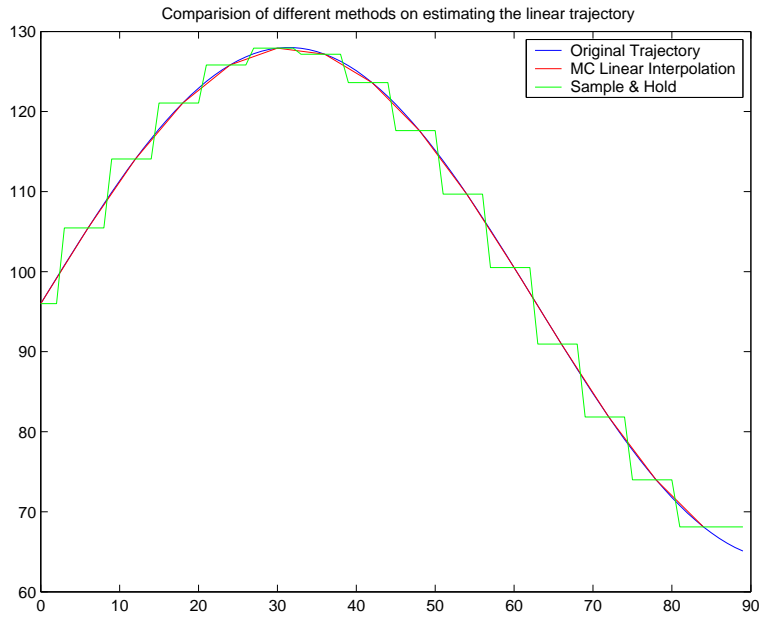


Figure 3: The motion curve.

Figure 3 shows how motion compensated interpolation should be better at capturing the motion than sample and hold.

The equation for such a motion is:

$$x = a + b * \sin(\alpha t) \tag{3}$$

and thus we will have $v_x = b * \alpha * \cos(\alpha t)$ and $a_x = -b * (\alpha)^2 * \sin(\alpha t)$.

The zoom out view in Fig. 4 shows how motion compensated interpolation will not completely capture the motion since it is missing higher order terms than the velocity.

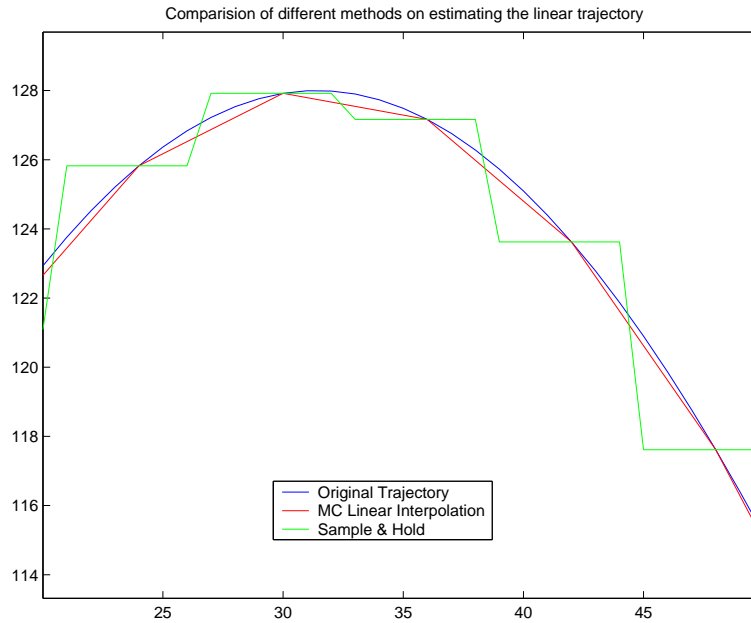


Figure 4: Zoomed version of the motion curve.

Methodology

Two different methods will be used to compensate for the motion: linear motion compensation and accelerated motion compensation. These methods share the first stages of the algorithm. The system diagram in Fig. 5 shows the steps in the algorithm:

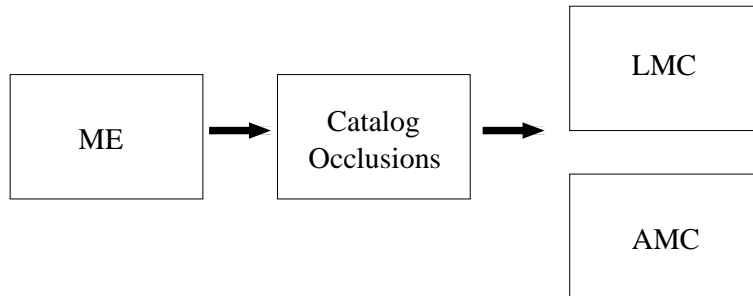


Figure 5: Block diagram of the algorithm.

Due to using overlapped block matching for the motion estimation stage, I had to use a reduced resolution image in order to avoid the computational intensity at the higher resolution. Alternatives will be discussed in the motion estimation part.

Motion Estimation

To reduce block artifacts with respect to using exhaustive BM, I used an overlapped exhaustive block-matching algorithm. This method is described in [B]. It consists of performing an exhaustive block-matching algorithm using 16x16 blocks and jumping every 8 pixels. As shown on Fig. 6 you end up having multiple motion vector candidates for each 8x8 block.

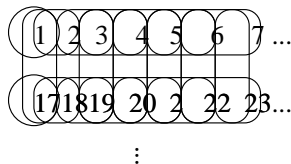


Figure 6: Overlapping block matching.

The candidate selection is performed by assigning the first block its only motion vector. Then blocks in the first row choose between the left motion vector, the current one or the one from the next frame according to a minimum SAD criteria. Those in the first column chose to min SAD between the MV from the block on top, the current one or the one from the next frame. For other blocks the MV is selected according to minimizing SAD between the MV from the top, left, current and next frame block. This is a sort of MV neighborhood smoothing operation. During the motion vector prediction the SADs are captured along with the MVs in order to use them during the occlusion categorization part.

We carry out motion estimation both in a forward way (the k^{th} frame as the anchor and the $k^{th} - 1$ frame as the predicted one) and in a backward way (ie reversing the roles just described). The reason for this will become apparent when we discuss classification of occlusions.

Finally the MVs are cleaned out to avoid illegal MVs, that is those that will result in motion outside our legal bounds.

One should note that block matching does not capture the true motion vectors since the resulting smoothness of the algorithm is insufficient leading to inconsistencies in the velocity field[C]. This is specially the case at boundaries of moving objects where the algorithm can not cope with the discontinuities in the velocity plane[C].As can be seen in Fig. 7 while my moving object is a rectangle, some of the block that contains its border and background show non-zero MVs while others don't and so the block MV field no longer looks like a rectangle. Thus ME has failed for those blocks since they contain a mixture of moving object and static

background.

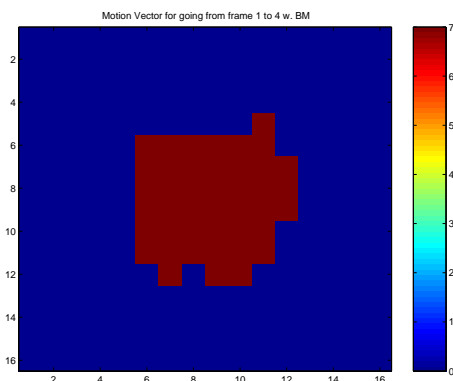


Figure 7: Errors in ME along a moving object's borders.

To try to capture the true vector field I started implementing a hierarchical block matching algorithm as described in [A] where I had three levels first level being a 128 by 128 image followed by a 256 by 256 and finishing with a 512 by 512 image. Due to the resolution of the last level it was necessary to implement the three step search method. The hierarchical block search idea is to first start with the first level in order to capture large displacements, then use the resulting estimate to search through an effectively smaller area in the next higher resolution level and so on. The total displacement is the sum of the displacement vectors of each stage. I ran out of time before being able to implement it fully so it is left as future work. It would be interesting to compare the result of this method to the one obtained with overlapped exhaustive block-matching.

Cataloging Occlusions

Between the two frames that we have, pixels can be covered or uncovered and it is a true challenge to capture this effect in the in-between frames.

The following picture captures the problem:

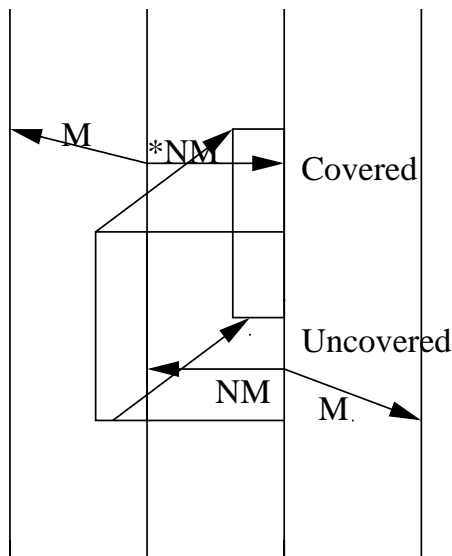


Figure 8: Cataloging Occlusions. M=match, NM=non-match

Only those blocks whose SAD is greater than a value need to be considered. In our case that value is zero since we don't have noise in our image, however in real life there should be a threshold that accounts for the noise.

As can be seen from this figure in order to classify a block as covered in between the available frames (2 and 3 in this example), one has to always use frame 2 (in this example) as the predicted frame (that is the frame whose blocks we search for). We need to search for frame 2's blocks in frame 1 and frame 3, and the idea is that we will find them in frame 1 but not in 3. Thus the comparison is between the SAD of the backward MV from 3 to 2 and the SAD of the forward MV from 1 to 2. If the latter is smaller then we classify it as a covered block.

A similar procedure can be used for uncovered pixels: in order to classify a block as uncovered in between the available frames (2 and 3 in this example), one has to always use frame 3 (in this example) as the predicted frame (that is the frame whose blocks we search for). We need to search for frame 3's blocks in frame 2 and frame 4, and the idea is that we will find them in frame 4 but not in 2. Thus the comparison is between the SAD of the forward MV from 2 to 3 and the SAD of the backward MV from 4 to 3. If the latter is smaller then we classify it as an uncovered block. Some of the results can be seen in Fig. 9 and 10:

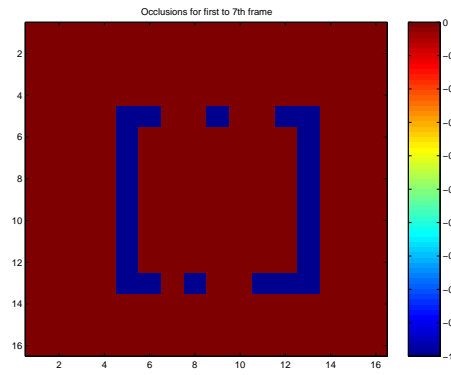


Figure 9: Occlusions found between Frames 1 and 7.Red=unoccluded, Blue=uncovered

As can be seen from Fig. 9, one can not classify covered pixels between frames 1 and 2 since one does not have a previous frame to compare it against. A similar effect happens between the end-1 frame and the end frame but this time with uncovered pixels.

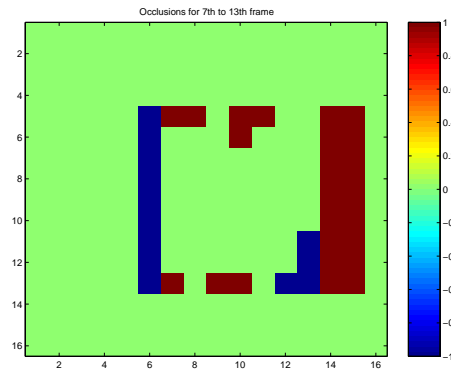


Figure 10: Occlusions found between Frames 7 and 13.Blue=uncov., Red=cov, Green=neither

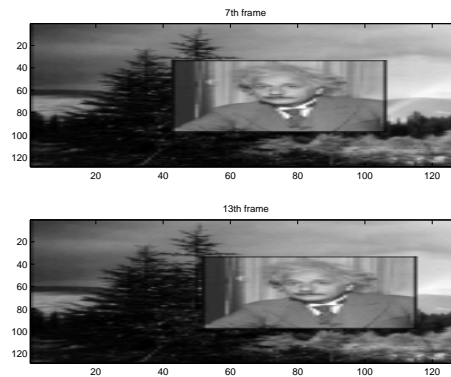


Figure 11: Frames 7 and 13 for reference to interpret Fig. 10.

As can be seen not all the pixels are classified correctly due to some areas having similar consistencies across blocks.

Linear Motion Compensation

In the final block of our system diagram, we have three distinct cases: pixels with no occlusions, uncovered pixels and covered pixels. Each case has to be treated appropriately.

For pixels with no occlusions, we first backtrack all the MVs. For example if we have down sampled by 3 so that we only have frame 1 and 4, then the MV between frame 1 and 2 will be $1/3$ of MV_{14} , while being careful to make sure we haven't moved to a new block during the motion. If we have, we will have to assign this MV to the new block. Thus some blocks will be left without a MV. For those without a MV assigned one needs to interpolate using for example the nearest available vectors. Once we have the backtracked MVs for the new frame, then we will look for the corresponding blocks in frame 1 and fill in frame 2 accordingly. However, some of our MVs will be non integer values, thus we need to interpolate and get the appropriate part in frame 1 to frame 2 that corresponds to the floor of the MV and the rest to the ceiling of the MV.

Fig.12 shows how the motion trajectory works in the case of LMC:

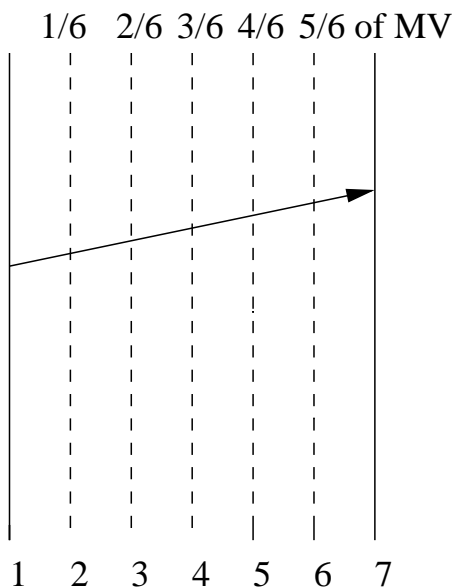


Figure 12: Motion trajectories for LMC, unoccluded pixels. 10.

For the case of occluded pixels, this is where we have to make the assumption that the pixels are moving in x only to simplify the situation. We first figure out the number of pixels

that are occluded and where they are located (ie at the beginning of the block or at the end) according to the following rules:

If uncovered and moving in +ve x direction the uncovered pixels will occupy the first indexes of my blocks. If moving in the -ve x direction the uncovered pixels will occupy the last indexes of my blocks For all motions the number of uncovered pixels will be:

$$\text{round}\left(\frac{d-1}{\text{upsmplngfctr}} * \frac{\text{size}_{\text{blocks}}}{2}\right) \quad (4)$$

where d is the frame number we are trying to interpolate.

If covered and moving in +ve x direction the covered pixels will occupy the last indexes of my blocks. If moving in the -ve x direction the covered pixels will occupy the first indexes of my blocks.

For all motions the number of uncovered pixels will be:

$$\text{round}\left(\frac{d-1}{\text{upsmplngfctr}} * \frac{\text{size}_{\text{blocks}}}{2}\right) \quad (5)$$

where d is the frame number we are trying to interpolate.

$$\text{round}\left(\frac{\text{upsmplngfctr} - (d-1)}{\text{upsmplngfctr}} * \frac{\text{size}_{\text{blocks}}}{2}\right) \quad (6)$$

For the pixels in the occluded block that are not occluded we use the appropriate neighboring blocks to estimate their MVs and we interpolate along their motion trajectory as we did for the unoccluded blocks. Fig.13 describes how we compensate during occlusions:

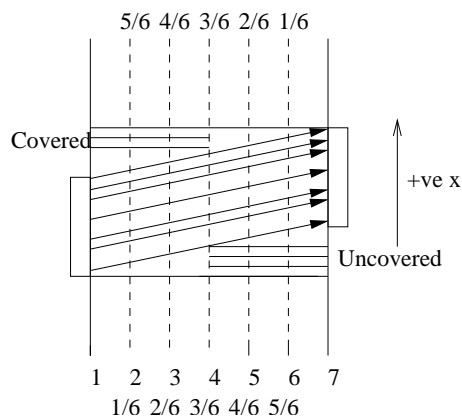


Figure 13: Trajectories for LMC, occluded pixels.

Accelerated Motion Compensation

The final block can also be implemented using accelerated motion compensation where we estimate the curve of our motion and assign MVs in between each of the frames that we want to interpolate along their motion trajectory. In order to solve the following equation:

$$d_x = v_x t + 0.5 a_x t^2 \quad (7)$$

for a_x and v_x one needs two MVs, that is two degrees of freedom, however in order to get a more accurate estimation one can solve a Least Mean Square problem using three motion vectors. One from the set of frames before and one from the set of frames that follow. We need to solve the following equation:

$$\begin{pmatrix} d1_x \\ d2_x \\ d3_x \end{pmatrix} = \begin{pmatrix} t1 & t1^2 \\ t2 & t2^2 \\ t3 & t3^2 \end{pmatrix} \begin{pmatrix} v_x \\ 0.5 * a_x \end{pmatrix}$$

to obtain:

$$\begin{pmatrix} v^* \\ 0.5 * a_x^* \end{pmatrix} = (A^T A)^{-1} A^T b$$

where:

$$A = \begin{pmatrix} t1 & t1^2 \\ t2 & t2^2 \\ t3 & t3^2 \end{pmatrix}$$

$$B = \begin{pmatrix} d1_x \\ d2_x \\ d3_x \end{pmatrix}$$

Fig.14 shows an example of how the MVs will end up looking in this case:

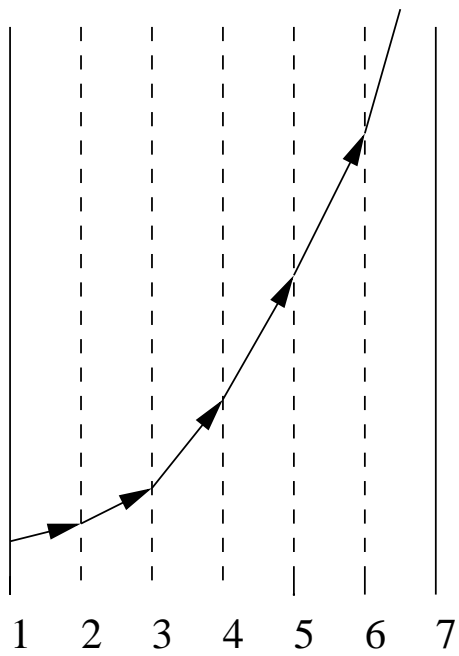


Figure 14: Motion trajectories for AMC.

The difficulty with this method arises in that one needs to track blocks across frames and it becomes difficult to have 3 accurate MVs from frame to frame. The best way to tackle this problem is to segment the image into different moving objects each with its MV and estimate the acceleration curve.

Since implementing object segmentation is another report in itself. I manually estimated the curve by looking at the resulting MVs from the center of the moving object. Fig 15 shows the result obtained.

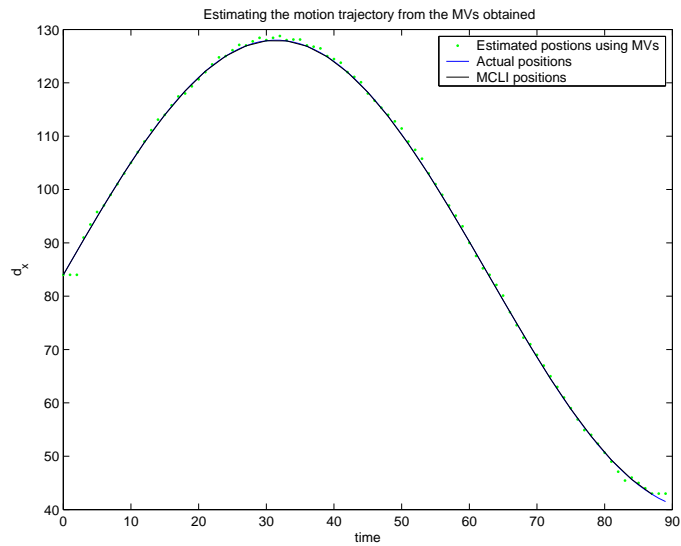


Figure 15: Results obtained when estimating the motion trajectory using LMS.

A zoomed out view in Fig 16 shows that the motion trajectory tracked will be less accurate in this case than using MCLI because the MVs are not the true MVs and because to get a more precise estimate we should use more MVs. Thus to turly compare the two one should create motion with more acceleration and one should use true motion vectors estimated with for example hierarchical block matching.

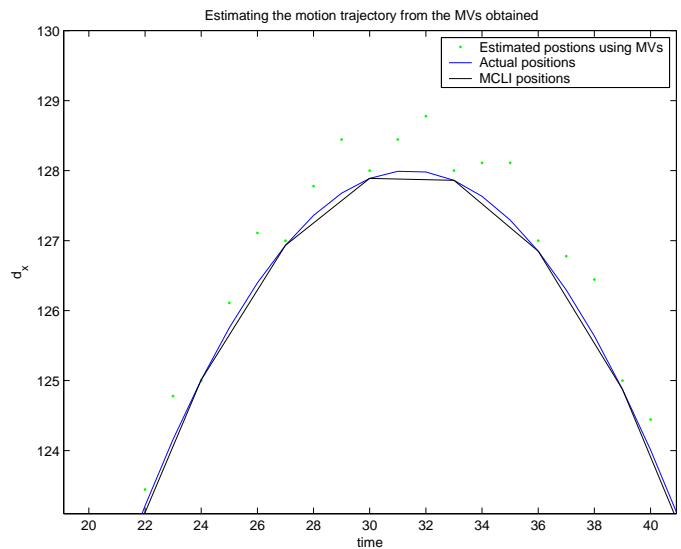


Figure 16: Zoom in of Fig 15.

Conclusion and Future Work

Fig 17 shows that linear motion compensation works better than simple linear interpolation and much better than sample and hold since it has higher PSNR. There were two frames where our signal is at the peak of the sine wave and thus does not move much between frames where simple linear interpolation worked better but in general LMC was better by a few dBs.

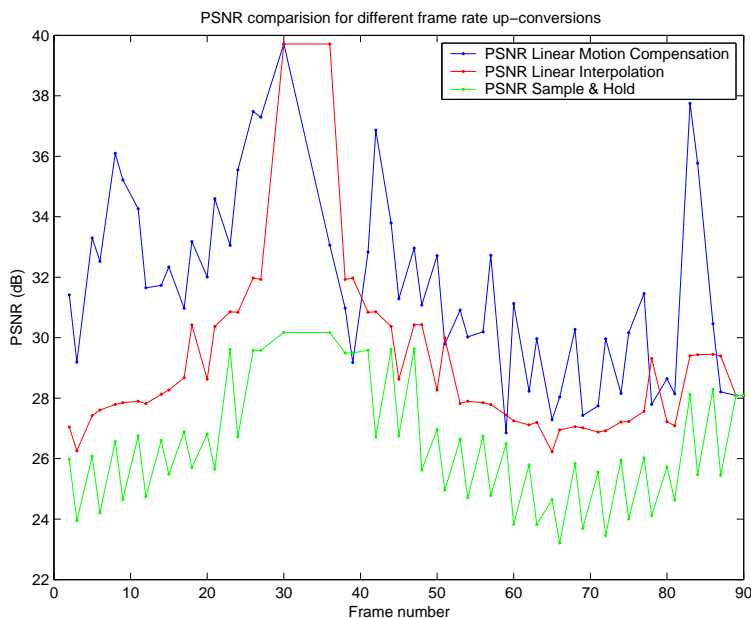


Figure 17: PSNR results obtained.

The improved performance can also be seen in the attached videos where I show einstein moving in a static background with sample and hold, linear motion and LMC.

As can be appreciated frame rate up conversion is a difficult problem as there are many details one has to look into.

It is crucial to have good motion estimation and use true motion vector fields which can be obtained using the hierarchical approach explained above and which is left for future work. Additionally some of the motion can fall in between pixels and thus we have to interpolate. As described, dealing with occlusions is a difficult problem with many details and things that can go wrong especially if our SAD measurements are not too accurate. Additionally left to future work is dealing with both x and y movement when compensating the occluded parts. In our case since we only have x movement then we can say that certain columns will be

occluded and the rest won't. In the case of y and x movement then we have to take a diagonal across the block to separate occluded from unoccluded pixels.

Also, motion can change drastically between frames and we won't be able to capture these drastic changes since they can not be described by the first and second order terms. We could also have hit a critical velocity during down sampling and we are not able to recover our motion, for example looking at the frames that are left it could seem as though there was no motion when in fact there is.

Another problem is how to deal with scene changes. For example the MC with acceleration relies on previous and past MVs that will be lost during scene changes. A solution to this is to realize that there has been a scene change by looking at the difference from one frame to another and act as though it was the first frame.

References

- [A] Motion Compensating Interpolation Considering Covered and Uncovered Background; Robert Thoma and Matthias Bierling
- [B] Using Motion-Compensated Frame-Rate Conversion for the Correction of 3:2 Pulldown Artifacts in lman, Video Sequences; Kevin Hilman, Hyuon Wook Park, and Yongmin Kim
- [C] True-Motion Estimation with 3-D Recursive Search Block Matching; Gerard de Haan, Paul W.A.C. Biezen, Henk Huijgen and Olukayode A. Ojo
- [D] Digital Video Standards Conversion in the presence of accelerated motion; Amdrew J. Patti, M. Ibrahim Sezan and M.Murat Tekalp
- [E] Frame Rate Up- Conversion using transmitted true motion vectors; Yen-Kuang Chen, Anthony Vetro, Huifang Sun, and S.Y. Kung
- [F] Video Processing and Communications; Yao Wang, Jorn Ostermann, Ya-Qin Zhang
- [G] John Apostolopoulos and Susie Wee