An MRF-Based DeInterlacing Algorithm with Exemplar-Based Refinement

Shengyang Dai* *(IEEE Student Member)*, Simon Baker , and
Sing Bing Kang *(IEEE Senior Member)*

***Abstract*—In this paper, we propose an MRF-based deinterlacing algorithm that combines the benefits of rule-based algorithms such as motion-adaptation, edge-directed interpolation, and motion compensation, with those of an MRF formulation. MRF-based interpolation and enhancement algorithms are typically formulated as an optimization over pixel intensities or colors, which can make them relatively slow. In comparison, our MRF-based deinterlacing algorithm uses interpolation functions as labels. We use 7 interpolants (3 spatial, 3 temporal, and 1 for motion compensation). The core dynamic programming algorithm is therefore sped up greatly over the direct use of intensity as labels. We also show how an exemplar-based learning algorithm can be used to refine the output of our MRF-based algorithm. The training set can be augmented with exemplars from static regions of the same video, as a form of "self-learning."**

***Index Terms*—Video DeInterlacing, Markov-Random Fields (MRF), Exemplar-Based Learning, Self-Learning.**

I. Introduction

INTERLACING was invented in the 1930's by Randall Ballard as a way to increase the resolution of Cathode Ray Tubes (CRTs). Standards authorities subsequently incorporated interlacing into NTSC and PAL. While CRT resolution is no longer an issue, and a variety of other display technologies such as LCD, plasma, and micromirror (DLP) are commonplace, interlaced video still has the benefit of requiring half the bandwidth. The two main broadcast formats for high-definition television (HDTV), 1080i (1920$\times$1080 interlaced) and 720p (1280$\times$720 progressive scan), require roughly the same bandwidth. Some broadcasters use the non-interlaced 720p, whereas others prefer the higher resolution of 1080i. In summary, the capture of interlaced video (NTSC, PAL, DV, 1080i, etc.) will remain commonplace for the foreseeable future because a lower bandwidth is required for a given resolution.

While the adverse effects of interlacing are largely imperceptible on a CRT because of the low persistence of the phosphor screen and high frame-rates, when interlaced video is viewed on LCD, DLP, or plasma displays, "mouse teeth" artifacts can be visible around the boundary of moving objects. The effect becomes more visible when interlaced video is displayed progressively at lower frame-rates or when single frames are viewed, since two fields captured at different times are presented at the same time. To display interlaced video on modern LCD, DLP, and plasma displays, a deinterlacing algorithm is required.

Many deinterlacing algorithms operate by first deciding how to interpolate the video locally at each missing pixel and then performing the interpolation [1]. For example, motion adaptation algorithms decide whether to interpolate spatially or temporally based on the results of a motion detection algorithm. If the scene is moving, spatial interpolation is used to avoid mouse-teeth artifacts. If the scene is stationary, temporal interpolation is used to obtain the highest possible resolution. Edge-directed algorithms such as [2], [3] estimate the dominant (spatio-temporal) edge direction and then avoid interpolating across the edge. Motion-compensation algorithms [4], [5], [6], [7], [8], [9], [10] first compute (global or local) motion and then use the motion to interpolate the missing pixels. Because motion estimation can be very unreliable (local motion can be very noisy, and global motion may only be appropriate in parts of the scene), an important component in motion-compensation algorithms is to estimate how reliable the motion is. A decision is then made whether to trust the motion-compensated pixel, or to fall back on a simpler algorithm. A criticism of all of these algorithms is that they are essentially a set of rules to determine how to interpolate the video at each missing pixel. These rules have associated parameters which must be tuned carefully to obtain the best performance. More sophisticated algorithms that combine several techniques can lead to a complex sets of rules with a sizeable number of parameters to be tuned.

An alternative approach to deinterlacing is to pose the problem as a Markov Random Field (MRF) [11]. In such a formulation, a *data cost* is used to encourage the missing pixels to be close to their neighbors in the rows above, below, before and after. In [11], the exact cost function depends on an estimate of the local edge structure; this is a form of edge-based interpolation. A *regularization term* is also used to encourage the pixels in each row to vary smoothly. The main limitation of the MRF algorithm in [11] is that it is computationally demanding as it requires a discrete optimization using simulated annealing over a large number of possible intensity values (around 17 in actual implementation in [11], while ideally 256 per color band for the the full intensity-based MRF formulation). We also found that our algorithm gives better results. See Table II in Section II-G4.

In the first part of this paper (Section II), we present an MRF-based deinterlacing algorithm that combines the advantages of the "rule-based" approaches (high efficiency and

Shengyang Dai is with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, Tel: (847) 491-4466, Fax: (847) 491-4455, Email: s-dai@northwestern.edu.

Simon Baker and Sing Bing Kang are with Microsoft Research, Redmond, WA, 98052, Tel: (425) 421-7748, Fax: (425) 936-7329, Email: {sbaker, sbkang}@microsoft.com.

sharper results) with the principled approach of using an MRF (which reduces the need for parameter tuning in a complex set of rules). We assume that the deinterlacing algorithm can choose between a number of different interpolants that could be applied at each pixel in the video. We consider 7 different interpolants in this paper. The first 2 are simple spatial (vertical) and temporal interpolants, analogous to motion adaptation. The next 2 consist of interpolation at $45°$ and $135°$, examples of edge-directed interpolants. We also include 2 interpolants to model occlusion and disocclusion to reduce ghosting caused around motion discontinuities. Finally, we include 1 interpolant for optical flow based motion compensation. Other interpolants could also be used, for example, a global motion based motion compensation algorithm. Our contribution is the general framework, not the specific interpolants.

We choose the interpolant at each pixel using a Markov Random Field (MRF); however, the labels in our MRF correspond to the interpolants, rather than to the grey-levels for the intensity-based MRF formulation of [11]. After the labels have been chosen, the output image is generated by interpolating with the chosen interpolant. This leads to a huge computational speed-up.

The key component in our MRF formulation is defining a data cost for the interpolants. We propose subsampling the input progressive video by a factor of 2 in both space and time. The result is 4 progressive videos with small offsets. We apply the interpolants to these videos and compare with the actual pixel values. The data cost is RMS error between the actual and interpolated pixels. The primary benefits of this data cost is that all interpolants are evaluated in the same manner and the units of the data cost are the same for all interpolants. This makes deciding which interpolant is the best one locally at any given pixel a straightforward comparison with no tuning parameters. In particular, there is no need to explicitly estimate the reliability of the optical flow. The reliability is implicitly contained in the data cost of the optical flow interpolant.

We do regularize the decision of the best interpolant to use with the MRF regularization term. This introduces the algorithm's only parameters, which empirically we found to be very easy to set. Empirically, we compared 1D regularization, 2D regularization, and 3D regularization and found their performance to be very similar. This allows us to use 1D regularization for solving the MRF efficiently via dynamic programming.

In the second part of this paper (Section III), we present an exemplar-based learning algorithm in the spirit of [12], [13] to refine the output of our MRF-based algorithm. We first train our algorithm on generic images in the traditional manner and show how it can reduce artifacts such as jagged edges. The performance of exemplar-based algorithms such as [12], [13] improves the closer the exemplars match the input images. The ideal case is to use the input images as training exemplars, but this is not possible in most applications. Interlaced video has the interesting property that interlacing artifacts disappear in static regions. In Section III-A, we present an algorithm to take advantage of this insight by augmenting the training data with exemplars *from static regions of the same video.* While this "self learning" can only help for videos containing static or slowly moving regions, we show qualitatively that when it does apply, the performance can be substantially improved. Note that in [14], a related technique was used to fill in holes in a video by copying pixels from other parts of the same video.

II. MRF-Based Deinterlacing Algorithm

In this section, we present an MRF-based deinterlacing algorithm where the state labels represent interpolation functions rather than pixel colors. This substantially reduces the search space and allows an efficient solution. It also eliminates the need for data costs and regularity functions between neighboring pixel intensities which inevitably lead to over smoothing of the output.

A. Problem Statement and Notation

We assume that the input is an interlaced video stream $I(x, y, t)$, where x is the horizontal coordinate or column index and y is the vertical coordinate or scan line index. We use the terminology that each interlaced video frame is made up of 2 fields and the temporal coordinate $t = 1, 2, \ldots$ denotes the field. We assume that the fields are captured at equally spaced time intervals. We follow the NTSC convention and assume the even numbered lines ($y = 2, 4, \ldots$) compose the first of a pair of fields and the odd lines ($y = 1, 3, \ldots$) compose the second. With this convention, only the pixels $I(x, y, t)$ where $(y+t) \bmod 2 = 1$ are present in the interlaced signal. The task of deinterlacing consists of estimating $I(x, y, t)$ at the missing pixels $M = \{(x, y, t) : (y + t) \bmod 2 = 0\}$.

B. MRF Formulation

Assume we have L interpolation functions $F_1, \ldots, F_L$. For example, F_1 could be vertical spatial interpolation:

$$F_1(I, x, y, t) \; = \; \frac{1}{2}\left(I(x, y-1, t) + I(x, y+1, t)\right). \qquad (1)$$

Interpolating with F_1 is very similar to *Bob* deinterlacing, which strictly consists of replicating lines rather than interpolating. F_2 may be temporal interpolation:

$$F_2(I, x, y, t) \; = \; \frac{1}{2}\left(I(x, y, t-1) + I(x, y, t+1)\right). \qquad (2)$$

Interpolating with F_2 is very similar to *Weave* deinterlacing, which strictly consists of using only either the field before or after. Note that if $I(x, y, t)$ is a missing pixel, $F_1(I, x, y, t)$ and $F_2(I, x, y, t)$ can both be computed from the pixels actually present in the interlaced video I.

We then pose deinterlacing as choosing the best interpolation function at each missing pixel in M. This interpolant is then applied to generate the output video. We choose the best interpolation function by minimizing the following Markov Random Field (MRF) optimization criterion:

$$\sum_{s \in M} \left[D(l_s, x, y, t) + \sum_{s' \in \mathcal{N}_s} R(l_s, l_{s'}, s, s') \right] \qquad (3)$$

over all of the interpolation labels $l_s \in \{1, 2, \ldots, L\}$ of the missing pixels in M, where $s = \{x, y, t\}$ indexes pixels. In

Equation (3), $D(l_s, x, y, t)$ is the data cost of label l_s at s. $D(l_s, x, y, t)$ measures how good an interpolant l_s is likely to be (Section II-D). $R(l_s, l_{s'}, s, s')$ is the regularization term which encourages consistency between the labels l_s and $l_{s'}$ in a small neighborhood $\mathcal{N}_s$ (Section II-E).

Our algorithm differs in two main ways from rule-based algorithms such as motion adaptation [1], [15], extensions to model edge direction [2], [3], or motion compensation [4], [5], [6], [16], [7], [8], [9], [10], [17]. First, a global decision is made by integrating information over a neighborhood. This increases the robustness to noise and aliasing effects. Second, our data cost (Section II-D) is a direct measure of how good the various interpolants are, rather than an ad-hoc rule such as whether the scene is moving or not, or whether motion estimation is reliable.

Li and Nguyen [11] proposed an MRF-based algorithm for deinterlacing. They set up the MRF in the usual manner over pixel intensities rather than interpolation functions. While setting up the MRF over intensities may at first glance seem appealing, the state space is significantly larger and the process much more computationally demanding. In particular, Li and Nguyen [11] used a pruning procedure to reduce the number of candidates to 17 and then used simulated annealing to optimize the MRF. In contrast, the proposed interpolant-based MRF is inherently faster because the state space is lower dimensional. We also found regularizing over the interpolants to perform better in terms of the quality of the deinterlaced video. See Table II in Section II-G4.

C. Additional Interpolation Functions

In Section II-B we defined two interpolation functions as illustrative examples. We use up to $L = 7$ interpolants in this paper. We now describe the other 5. Vertical spatial interpolation using F_1 can lead to jagged edges when the dominant edge direction is not horizontal or vertical [2], [3]. To alleviate these effects, we add spatial interpolations at 45° and 135°:

$$F_3(I, x, y, t) = \frac{1}{2}\left(I(x-1, y-1, t) + I(x+1, y+1, t)\right) \quad (4)$$

$$F_4(I, x, y, t) = \frac{1}{2}\left(I(x+1, y-1, t) + I(x-1, y+1, t)\right) \quad (5)$$

These interpolants allow our MRF algorithm to perform edge-directed interpolation [2], [3]. Temporal interpolation using F_2 can lead to ghosting around the edges of objects because $I(x, y, t-1)$ may be a pixel in the background and $I(x, y, t+1)$ may be a pixel in the foreground (or vice versa). To reduce this ghosting, we include functions that only interpolate forwards and backwards in time. They are suitable for regions which are just about to appear or be occluded respectively:

$$F_5(I, x, y, t) = I(x, y, t+1) \quad (6)$$

$$F_6(I, x, y, t) = I(x, y, t-1). \quad (7)$$

Finally, we add a motion compensation based interpolation function [4], [5], [6], [16], [7], [8], [9], [10], [17]. We compute optical flow using a pyramid-based flow algorithm based on [18]. Suppose the computed flow is $u(x, y, t), v(x, y, t)$, i.e., pixel $I(x, y, t)$ moves to pixel $I(x+u(x, y, t), y+v(x, y, t), t+1)$ in the next field. We then interpolate by following the flow forwards and backwards:

$$F_7(I, x, y, t) = \frac{1}{2}\left(I(x+u(x, y, t), y+v(x, y, t), t+1) + I(x-u(x, y, t), y-v(x, y, t), t-1)\right). \quad (8)$$

Since the flow may take non-integral values, and the destination may be a missing pixel, we compute $I(x+u(x, y, t), y+v(x, y, t), t+1)$ and $I(x-u(x, y, t), y-v(x, y, t), t-1)$ by interpolating the known pixels at $t+1$ and $t-1$ respectively.

Note that our MRF-based formulation is independent of the exact choice of interpolation functions. Other interpolants may be used in addition to or instead of the functions described in this paper. For example, a global motion based motion compensation algorithm could be used. Fewer interpolants may also be used.

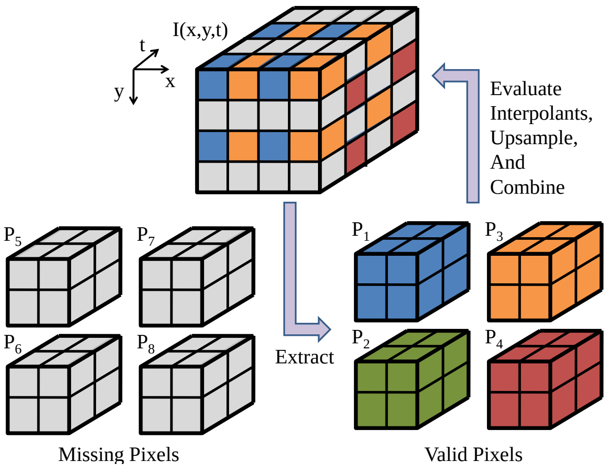


Fig. 1. An illustration of the data cost used in our algorithm. We subsample the input video $I(x, y, t)$ by a factor of 2 with all possible offsets to generate 4 progressive videos P_1, P_2, P_3, and P_4. We then evaluate the various interpolants on these progressive videos, upsample the results, and average using Equation (11).

D. Data Cost

We now describe the data cost $D(l_s, x, y, t)$ for a label $l_s \in \{1, 2, \ldots, L\}$ and missing pixel $s = (x, y, t) \in M$. Consider the following 8 sub-videos extracted from the interlaced video $I(x, y, t)$ by subsampling by a factor of 2 with all 8 possible offsets (see Figure 1 for an illustration):

$$\begin{aligned}
P_1(x, y, t) &= I(2x, 2y, 2t+1) \\
P_2(x, y, t) &= I(2x, 2y+1, 2t) \\
P_3(x, y, t) &= I(2x+1, 2y, 2t+1) \\
P_4(x, y, t) &= I(2x+1, 2y+1, 2t) \\
P_5(x, y, t) &= I(2x, 2y, 2t) \\
P_6(x, y, t) &= I(2x, 2y+1, 2t+1) \\
P_7(x, y, t) &= I(2x+1, 2y, 2t) \\
P_8(x, y, t) &= I(2x+1, 2y+1, 2t+1). \quad (9)
\end{aligned}$$

In Equations (9), x, y, and t run over half their usual ranges. Also, note that in practice we pre-filter each field $I(x, y, t)$

with a Gaussian blur with $\sigma = 0.4$ before we subsample (to reduce noise). The first four of these subsequences (P_1, P_2, P_3, P_4) contain all the known pixels; they are now subsampled progressive videos. The second four videos (P_5, P_6, P_7, P_8) contain all the missing pixels.

The interpolants F_{l_s} can be evaluated on the progressive videos P_i, $i = 1, \ldots, 4$, because all the data is present. For each pixel (x, y, t), we can compare the actual value $P_i(x, y, t)$ to the interpolated value $F_{l_s}(P_i, x, y, t)$ to yield

$$C_i(l_s, x, y, t) \;=\; \|P_i(x, y, t) - F_{l_s}(P_i, x, y, t)\|^2, \qquad (10)$$

where the norm $\|\cdot\|$ is needed for color. (Note that the flow does not need to be scaled for F_7 because the subsampling is performed for both space and time.)

If we wanted to evaluate the interpolants at a pixel present in the interlaced video, we could find the unique corresponding pixel in the subsampled videos and use the appropriate value of $C_i(l_s, x, y, t)$. To evaluate at a missing pixel $(x, y, t) \in M$, we set the data cost $D(l_s, x, y, t)$ to be the mean of $C_i(l_s, x, y, t)$ at the 4 pixels corresponding to the 4 nearest neighbors of s (two vertical and two temporal) which are present in the interlaced video:

$$D(l_s, x, y, t) \;=\; \frac{1}{4} \sum_{(i,j) \in Q} C_k\left(l_s, \left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y+i}{2} \right\rfloor, \left\lfloor \frac{t+j}{2} \right\rfloor\right) \qquad (11)$$

where $Q = \{(-1, 0), (1, 0), (0, -1), (0, 1)\}$, $k = 2 * (x \bmod 2) + (y + i) \bmod 2$ and $\lfloor\cdot\rfloor$ is the floor function which returns the integral part. This expression always finds the four pixels in the $C_i(l_s, x, y, t)$ that correspond to the four nearest neighbors of $(x, y, t) \in M$ because if $y + i$ is even, $t + j$ is odd, and vice versa.

Note that the data cost in Equations (10) and (11) is a direct measure of how well the interpolants perform on the downsampled videos P_i. This measure should be compared with indirect methods of predicting how well interpolants may perform, for example the use of motion detection in motion adaption and motion confidence estimation in motion compensation. All possible interpolants are compared in the same units (RMS pixel difference) and no tuning parameters are needed. There is no need to estimate the reliability of optical flow. This information is implicitly encoded in the data cost of the optical flow interpolant F_7.

Any deinterlacing algorithm must make assumptions to estimate the missing data. The major assumption in our algorithm is that if an interpolant performs well in the subsampled videos P_i, it will also perform well in the interlaced video itself. In a sense, our empirical results are largely an evaluation of how well this assumption holds. Intuitively, however, we argue that our assumption is reasonable. If there is an edge at 45° in I, there will also be one in P_i. If there is an occlusion in I, there will also be one in P_i. If the flow is accurate and interpolates well in P_i, it should also be accurate and interpolate well in I.

E. Regularization Cost and Optimization

No data cost can be totally reliable at all pixels. Spatial and temporal aliasing is always possible because of the lack of the missing pixels that we are trying to estimate. Noise in the original video $I(x, y, t)$ can also propagate into the data cost. The purpose of the regularization term $R(l_s, l_{s'}, s, s')$ is to reduce this noise by integrating information over a neighborhood $\mathcal{N}_s$.

One benefit of our algorithm is that the choice of the regularization neighborhood allows a trade-off between computational cost and the quality of the results. If we wish to obtain the fastest performance, we can set the neighborhood to be 1D (i.e., $\mathcal{N}_s = \{(x-1, y, t), (x+1, y, t)\}$) and optimize the MRF in Equation (3) by dynamic programming. To obtain better quality results, we can set the neighborhood to be 2D (i.e., $\mathcal{N}_s = \{(x-1, y, t), (x+1, y, t), (x, y-2, t), (x, y+2, t)\}$) or 3D (i.e., $\mathcal{N}_s = \{(x-1, y, t), (x+1, y, t), (x, y-2, t), (x, y+2, t), (x, y, t-2), (x, y, t+2)\}$). In the 2D and 3D cases, we optimize Equation (3) using belief propagation. Optimizing over the entire 3D space-time is not practical, so in the 3D case we perform the optimization over a sliding window of the previous K frames. The results in this paper were obtained with $K = 3$. We experimented with larger values of K but found little benefit.

F. Setting the Regularization Parameters

We chose the regularization weight $R(l_s, l_{s'}, s, s') = \mu(l_s, l_{s'}, n(s, s'))$ to be homogeneous, depending only on the relative position between s and s'. Here, $n(s, s')$ is an indicator function: $n(s, s') = 1$ if s and s' are horizontal neighbors, 2 for vertical neighbors, and 3 for temporal neighbors. We then set $\mu(l_s, l_{s'}, 1) = 1.2\mu_0(l_s, l_{s'})$, $\mu(l_s, l_{s'}, 2) = 0.6\mu_0(l_s, l_{s'})$, and $\mu(l_s, l_{s'}, 3) = 0.12\mu_0(l_s, l_{s'})$, where:

$$\mu_0 = \begin{pmatrix} 0.0 & 5.0 & 0.3 & 0.3 & 5.0 & 5.0 & 1.0 \\ 5.0 & 0.0 & 5.0 & 5.0 & 4.0 & 4.0 & 3.0 \\ 0.3 & 5.0 & 0.0 & 0.6 & 5.0 & 5.0 & 1.0 \\ 0.3 & 5.0 & 0.6 & 0.0 & 5.0 & 5.0 & 1.0 \\ 5.0 & 4.0 & 5.0 & 5.0 & 0.0 & 3.0 & 1.0 \\ 5.0 & 4.0 & 5.0 & 5.0 & 3.0 & 0.0 & 1.0 \\ 1.0 & 3.0 & 1.0 & 1.0 & 1.0 & 1.0 & 0.0 \end{pmatrix}. \qquad (12)$$

The diagonal elements of μ_0 are set to 0.0 to encourage local label smoothness. The other values were chosen based on simple reasoning. For example, $\mu_0(1, 2)$ is large to discourage transitions from spatial to temporal interpolation. On the other hand, $\mu_0(1, 3)$ and $\mu_0(1, 4)$ between vertical interpolation and spatial interpolation at 45° and 135° is much smaller, thus discouraging transitions between them less.

The results in this paper are all obtained using the regularization parameters provided above and in Equation (12). The values were set using common sense and a little bit of trial and error on the "Wave" sequence (Figure 2). Empirically we found it very easy to set reasonable parameters and the performance to be largely independent of the exact values. Note, however, that algorithms have been proposed to learn MRF regularization parameters. In particular, algorithms such as the pseudo-likelihood method [19], could be used. For the 1D case, iterative scaling or gradient-based methods could be applied, since the derivative of likelihood can be efficiently computed based on estimating the expectation of each feature

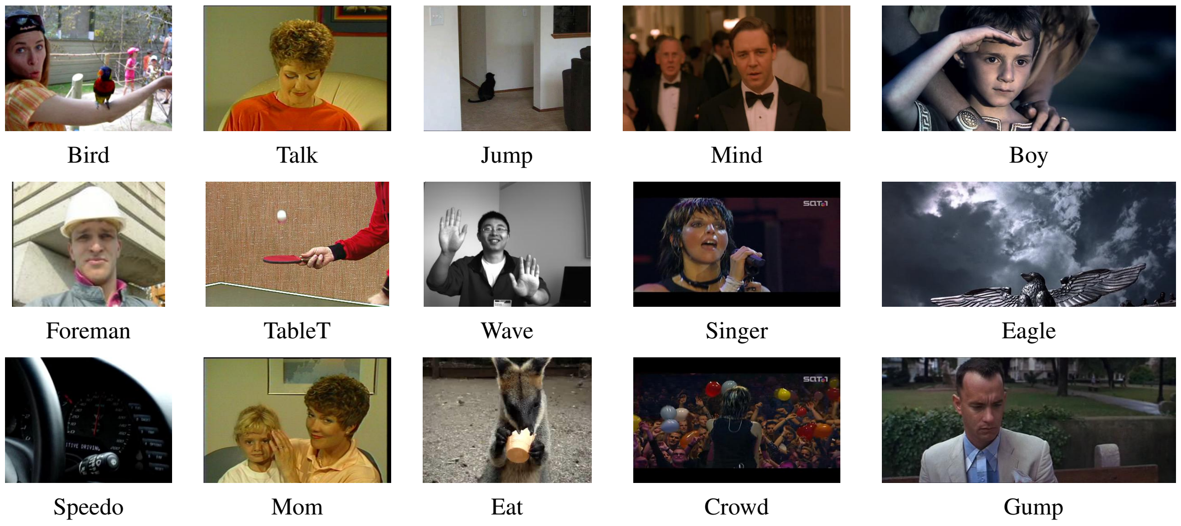


Fig. 2. The first frame of the 15 videos used to evaluate our algorithm. The videos are of varying size, aspect ratio, and quality. The sequences range from 77 frames (Eagle) to 452 frames (Gump).

with respect to the model distribution by dynamic programming [20].

G. *Experimental Results*

We evaluated our algorithm on 15 videos of varying size, aspect ratio, and quality (SNR). Figure 2 contains the first image in each sequence. The sequences range from 77 frames (Eagle) to 452 frames (Gump). To obtain quantitative results, we withhold every other line from the videos as ground-truth, pass the remaining interlaced video to the algorithms for processing, and then compare the predicted scan lines to the withheld ground-truth.

The remainder of this section is organized as follows. We first show how our algorithm performed with varying numbers of labels (Section II-G1). In Section II-G2, we compare the performance across the dimensionality of the regularization. In Section II-G3, we present timing results. Finally, in Section II-G4, we compare the performance of our algorithm with a number of other algorithms.

1) Contribution of the Extra Interpolants: In Figure 3, we present quantitative results obtained by varying the number of interpolants (labels) used in our algorithm. We include results for 2-labels (F_1 and F_2, roughly corresponding to motion-adaptation), 4-labels (F_1–F_4, an edge-directed algorithm), 6-labels (F_1–F_6, an algorithm with occlusion and disocclusion modeling), and all 7-labels (F_1–F_7, an algorithm with optical flow based motion compensation.) In the figure, we show the RMS pixel error for each sequence and each number of labels. 1D regularization using dynamic programming is used in all cases.

The results in Figure 3 show that on almost all sequences a substantial improvement is obtained by adding the edge-directed interpolants F_3 and F_4. A qualitative illustration of this is shown for one frame of the Mind sequence in Figure 4. In the left column we show the results without the edge-directed labels F_3 and F_4. In the right column we show the results with the edge-directed labels F_3 and F_4. On the top

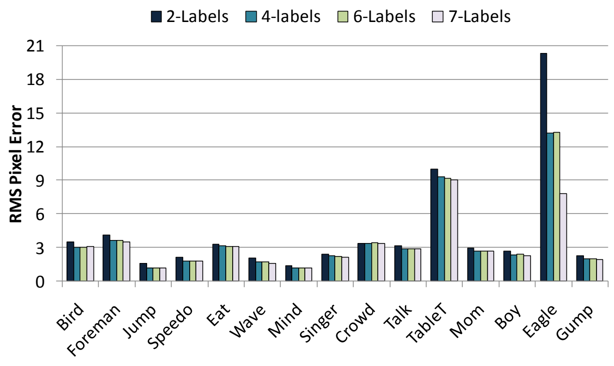
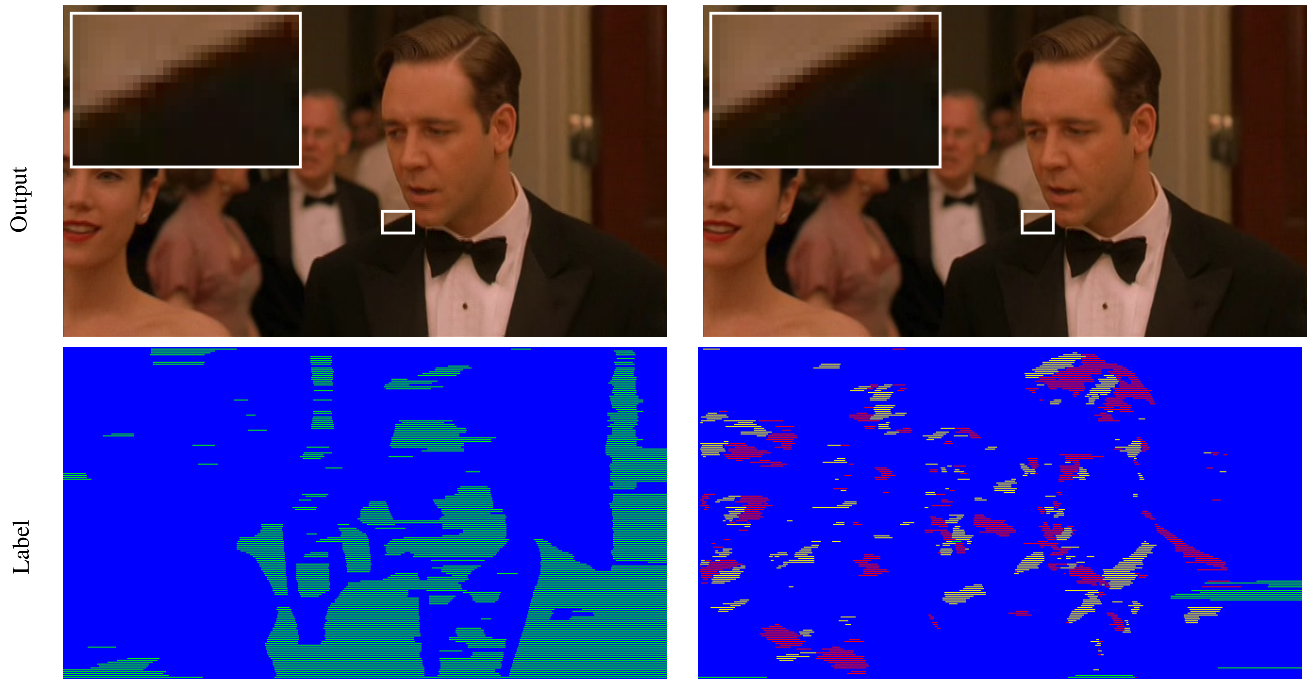


Fig. 3. The RMS pixel error for 4 different variants of our algorithm: 2-labels (F_1 and F_2, roughly corresponding to motion-adaptation), 4-labels (F_1–F_4, a edge-directed algorithm), 6-labels (F_1–F_6, an algorithm including occlusion and disocclusion modeling), and all 7-labels (F_1–F_7, an algorithm with optical flow based motion compensation.)

we show the output image, together with a cropped close up region. The results show a clear visual improvement using the extra labels. On the bottom we show the labels chosen by our algorithm: blue = F_1, green = F_2, yellow = F_3, and red = F_4. The edge directed labels appear as one would expect in the regions of strong diagonal edges.

The results in Figure 3 show that there is very little quantitative improvement obtained by adding interpolants F_5 and F_6 to model occlusions and disocclusions. Very few pixels in each video are about to be occluded or disoccluded and so it is not surprising that there is little quantitative difference. A qualitative illustration of the utility of these interpolants is shown for one frame of the Wave sequence in Figure 5. In Figure 5(a) we show the results without labels F_5 and F_6. In Figure 5(b) we show the results with labels F_5 and F_6. Without the extra interpolants, the eyebrow is not as dark as it should be due to ghosting. With F_5 and F_6, the eyebrow is interpolated using the next frame only, and so is as dark



(a) Without F_3 and F_4 (b)With F_3 and F_4

Fig. 4. A comparison with (b) and without (a) the edge-directed interpolants F_3 and F_4. With the extra interpolants, the edges are less jagged. See cropped close up region in the top row. The labels 45° (white) and 135° (red) are chosen in the appropriate places.

as it should be. Figure 5(c) shows the difference between Figure 5(a) and Figure 5(b) (without the closeups). Only a very small percentage of the pixels are changed by the addition of the new labels. Figure 5(d) shows the labels chosen. Red corresponds to label F_6. As expected, the new label was only chosen in pixels about to be occluded. Note that the moving part (hand) is mostly spatially interpolated (F_1) while the static part (face) is mostly temporally interpolated (F_2).

The results in Figure 3 show that for some sequences there is a substantial quantitative improvement by using the optical flow based motion compensation interpolant F_7. The Eagle sequence benefits significantly from its consistent global motion pattern (which ensures relatively high accuracy of the flow estimation technique) and highly textured appearance with almost horizontal edge structure (which is hard for other interpolants). For other sequences there is no improvement. Note that on none of the sequences is there any significant degradation caused by the addition of F_7. The optical flow algorithm we use [18], while reasonable, is by no means perfect. It is important to note that we perform no direct assessment of the quality of the optical flow. This is done implicitly in the computation of the data cost (see Section II-D). A qualitative illustration of the utility of F_7 is shown for one frame of the Eagle sequence in Figure 6. The cropped region shows that the use of F_7 can help substantially.

2) Regularization Dimension: In Figure 7, we present a quantitative comparison of the effect of regularization. We used the 7-label version of our algorithm and showed results for no regularization, 1D regularization, 2D regularization, and 3D regularization. We plot the results as percentage reduction in RMS error over using no regularization. The results show

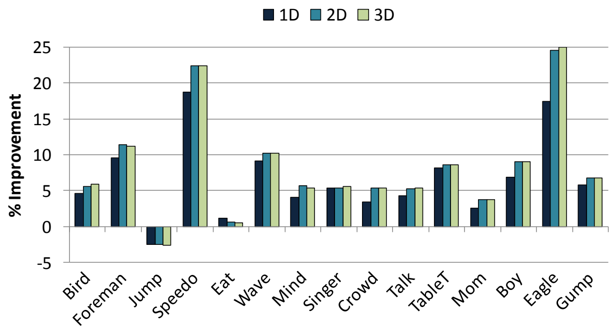
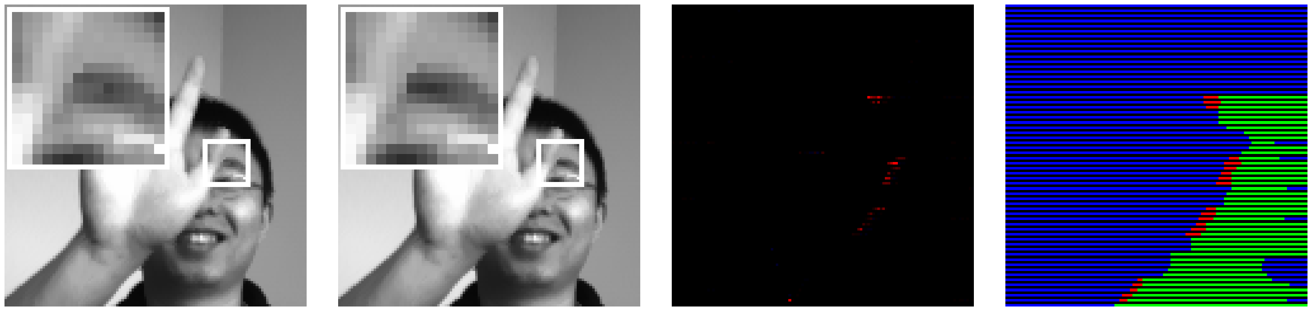


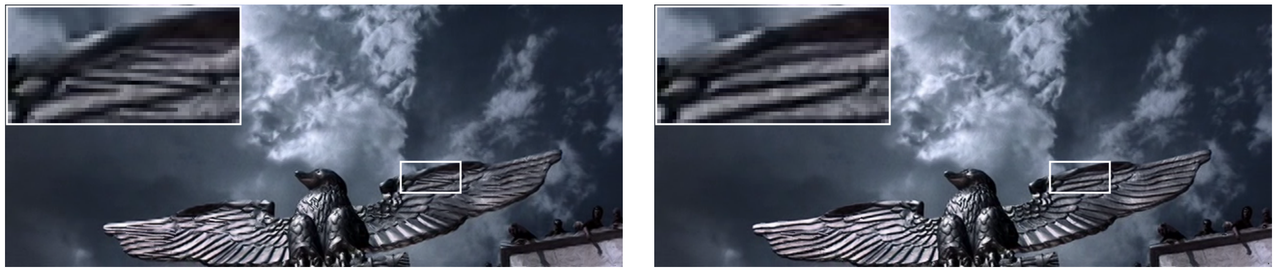
Fig. 7. A quantitative comparison between 1D, 2D, and 3D regularization for the 7-label version of our algorithm. We plot the percentage reduction in RMS error over no regularization. 1D regularization improved the results substantially, 2D helped slightly more for all except two of the videos, but 3D regularization produced no significant improvement over 2D.

that using regularization helps in almost all cases. However, 2D regularization only helps a little more than 1D, and 3D regularization never makes a significant difference over using 2D regularization. Among all the sequences we tested, the Speedo and Eagle sequences benefit the most from the regularization term. The reason is that these sequences contain a lot of details, and the data term tends to be less reliable in this case, while the regularization term is of great help to correct those errors. On the other hand, the Jump sequence performs a little worse due to the extreme lack of complex image details and texture. In this case, the data term itself is already pretty reliable, and the regularity term sometimes oversmooths the labels.



(a) Without F_5 and F_6 (b) With F_5 and F_6 (c) Difference (d) Label

Fig. 5. A comparison with (b) and without (a) the occlusion and disocclusion interpolants F_5 and F_6. Without the extra interpolants, the eyebrow is not as dark as it should be due to the ghosting, and as dark as it should be with the extra interpolants. (c) Shows that relatively few pixels are affected by the new interpolants because relatively few pixels are about to be occluded or disoccluded. The quantitative benefit of F_5 and F_6 is therefore minimal. (d) Shows that the the new occlusion label F_6 (red) is chosen where one would expect it.



(a) Without F_7 (b) With F_7

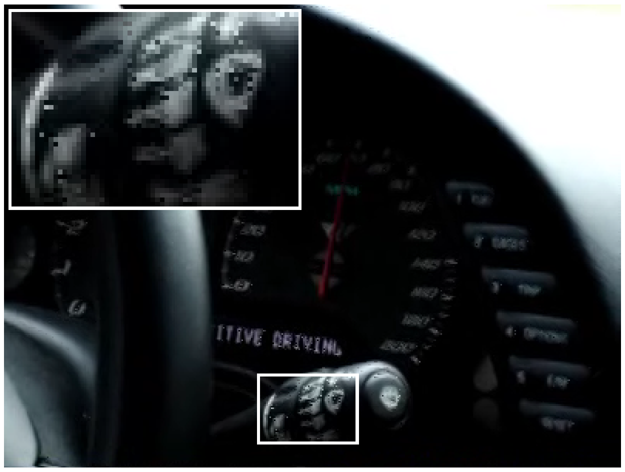
Fig. 6. A comparison with (b) and without (a) the optical flow interpolant F_7. With F_7, the appearance of the wing of the eagle is substantially improved. It is important to note that we perform no direct assessment of the quality of the optical flow. This quality assessment is done implicitly in the computation of the data cost in Section II-D. Hence, even though the optical flow can often be erroneous, the quality of the deinterlaced image is never significantly affected in an adverse manner.

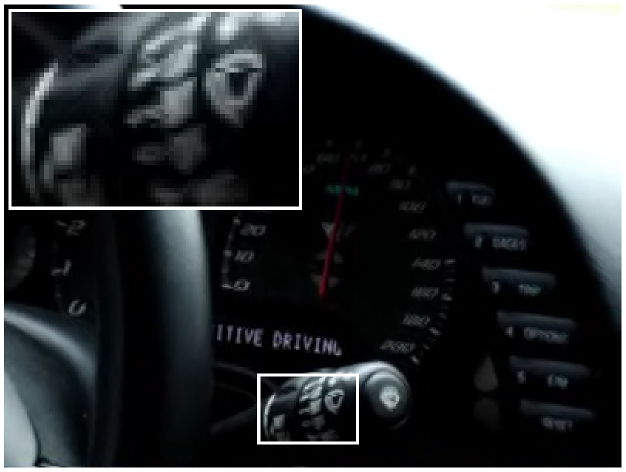
In Figure 8, we present qualitative results for the Speedo sequence. We include the results for one frame for (a) no regularization, (b) 1D regularization, (c) 2D regularization, and (d) 3D regularization. For this particular frame, using 1D regularization yielded a substantial improvement. There are, however, still some errors in (b). These errors are caused by the fast irregular motion that results in three consequent observable fields coincidentally showing similar patterns (and thus tagged with wrong interpolation functions). 2D regularization helped to correct those errors, while 3D regularization yielded no further visual improvement. The labels for 2D (g) and 3D (h) look far smoother than for 1D (f), however this only seems to result in a fairly small improvement in visual quality over 1D regularization. The labels with no regularization (e) are very noisy. There may be ways to approximate 2D regularization without the computational overhead of full belief propagation. One possibility is to filter the data cost in the vertical direction. An evaluation of exactly how well such approximations perform is left as future work.

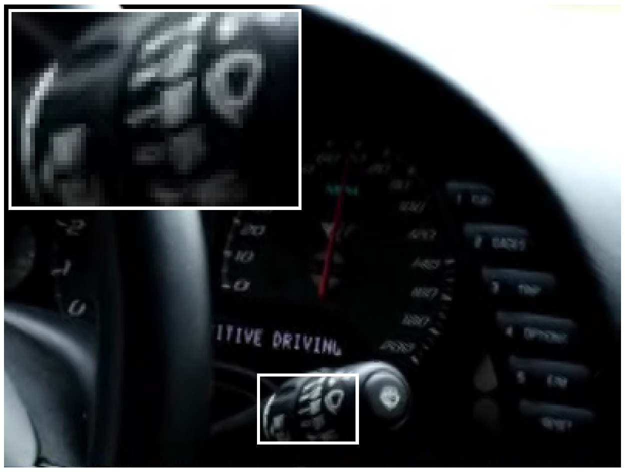
3) Timing Results: In Table I, we present timing results obtained on an HP nc8430 2.0GHz Core Duo laptop (2GB RAM, 4MB L2 cache, 667MHz front side bus) for a 320×240 pixel interlaced video (field size is 320×120) using an unoptimized 7-label version of our algorithm. Without optical flow, 2D or 3D regularization our algorithm runs in around 29 ms per frame. With a real-time flow algorithm, our algorithm could also be made to run in real-time. 2D or 3D regularization is currently only possible for offline operation or single-frame processing (for example to view or print a high quality still). Note that with 256 labels, the 1D dynamic programming (DP) alone would take approximately 9 ms $\times(256/7)^2 \approx 12$ sec.

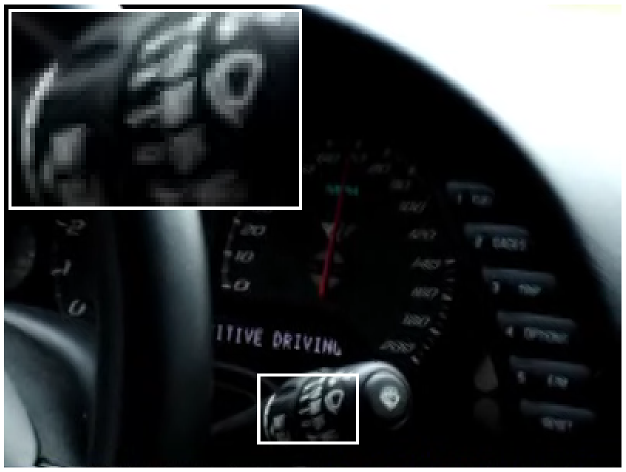
We also implemented a more optimized (no parallel instructions such as MMX or SSE are used) 4-label version $(F_1, \ldots, F_4)$ of our algorithm. This implementation operates at 288Hz for 320×240, 72Hz for 720×480, and 12Hz for 1920×1080 on an HP xw8200 3.60GHz workstation (2GB RAM, 2MB L2 cache, 800MHz front side bus). Much of the algorithm involves simple image filtering operations and can be significantly sped up by converting the code to SSE (Streaming SIMD Extensions) for Intel chipsets.

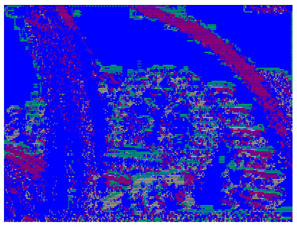
4) Comparison with Other Algorithms: Based on the quantitative results in Figure 3 and the timing results in Table I, there are 2 variants of our algorithm that offer different trade-offs in terms of speed vs. quality. The 4-label version is real-time, the 7-label version is not quite real-time without a faster optical flow algorithm. We now empirically compare these variants with two Virtual Dub [21] plugins, one from Gunnar Thalin [22] (deinterlace - smooth), the other the Smart Deinterlacer Filter v2.8 [23]. We also compared with the Alparysoft Deinterlacing Filter v2.0 [24]. Unfortunately, this last filter was unable to process seven of the sequences (Bird, Jump, Speedo, Singer, Crowd, Talk, and Mom) because

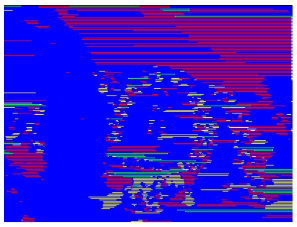

(a) No Regularization

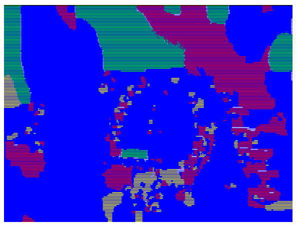

(b) 1D Regularization


(c) 2D Regularization


(b) 3D Regularization


(e) No Regularization


(f) 1D Regularization


(g) 2D Regularization

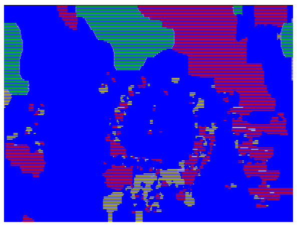

(h) 3D Regularization

Fig. 8. A qualitative comparison of varying the dimension of regularization on one frame from the Speedo sequence. 1D regularization yielded a substantial improvement. 2D regularization helped even more, however 3D regularization yielded no further visual improvement. The labels for 2D (g) and 3D (h) look far smoother than for 1D (g), however this only seems to result in a fairly small improvement in visual quality over 1D regularization. The labels with no regularization (e) are very noisy.

TABLE I

Timing results in milliseconds (ms) per frame on an HP nc8430 2.0GHz Core Duo laptop (2GB RAM, 4MB L2 cache, 667MHz front side bus) for a 320×240 pixel interlaced video (field size is 320×120) using the 7-label version of our algorithm. Without flow or 2D/3D regularization our algorithm runs runs in real-time. A more optimized 4-label version $(F_1, \ldots, F_4)$ of our algorithm operates at 288Hz for 320×240, 72Hz for 720×480, and 12Hz for 1920×1080 on an HP xw8200 3.60GHz workstation (2GB RAM, 2MB L2 cache, 800MHz front side bus.)

	Filter	Flow	Compute F_i	1D DP	2D BP	3D BP	Apply
Time	1 ms	232 ms	9 ms	17 ms	7801 ms	76624 ms	2 ms

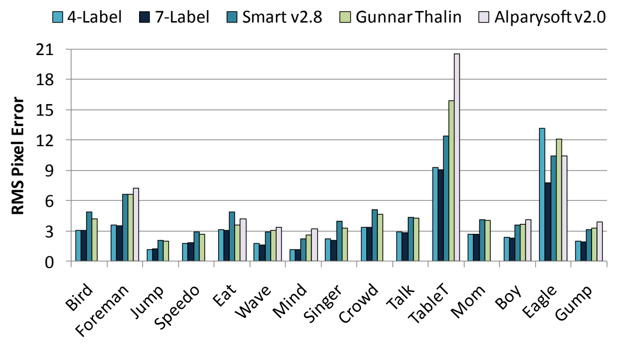


Fig. 9. A quantitative comparison between the 4-label and 7-label versions of our algorithm with two Virtual Dub [21] plugins (one from Gunnar Thalin [22], the other the Smart Deinterlacer Filter v2.8 [23]) and the Alparysoft Deinterlacing Filter v2.0 [24] (which was unable to process seven of the sequences). We computed the RMS pixel error averaged over all frames. We found that both versions of our algorithm outperformed the other 3 algorithms on all sequences except the Eagle sequence, where the 7-label algorithm outperformed the others, while the 4-label version is slightly worse.

of the unusual frame dimensions. The trial version of this filter also adds an icon which we mask out before computing the quantitative results. Figure 9 shows the RMS pixel error averaged over all the frames for each video. Both the 4-label and 7-label versions of our algorithm use 1D regularization with dynamic programming. Both versions of our algorithm outperformed the other three filters in all the videos except the Eagle sequence, sometimes quite dramatically. For the Eagle sequence, the 7-label version outperformed the 3 other algorithms, while the 4-label version was slightly worse. The Eagle sequence contains a highly textured object moving with a very simple motion, the ideal case for motion compensation-based algorithms, but relatively untypical of real-world videos where the motion is generally more complex. We also computed the same RMS error measure over just edge pixels (where the magnitude of the gradient was above a threshold, subsequently dilated). The results, which are omitted to avoid unnecessary redundancy, are very similar.

In Figure 10, we show one frame from the TableT sequence. Figure 10(a) shows the result of applying Weave on the interlaced input, Figure 10(b) the 4-label version of our algorithm, Figure 10(c) the 7-label version of our algorithm, Figure 10(d) Smart v2.8 [23], Figure 10(e) Gunnar Thalin [22], and finally in Figure 10(f) Alparysoft v2.0 [24]. In the bottom right of each image is a closeup view of the hand region. The results show that both the 4-label and 7-label versions of our algorithm produced more natural looking folds on the shirt. The boundary of the arm is also smoother. Also, the net and lines on the table look better.

In Figure 11, we include similar results for the Foreman sequence. We show closeup views of the lip region, which looks sharper and more natural with our algorithm. The diagonal lines in the background are also less jagged with both the 4-label and 7-label versions of our algorithm than with the other 3 algorithms.

In Table II we present quantitative results obtained by using ground truth data, comparing the performance of the 4-label and 7-label versions of our algorithm with a number of recently published algorithms on the Foreman sequence. In particular, we compared with Wang *et al.* [8], Chang *et al.* [9], Ouyang *et al.* [10], and Lee *et al.* [3]. The best performing of these algorithms obtained a PSNR of about 35.50 dB. In comparison, our 4-label algorithm has a PSNR of 37.77 dB and our 7-label algorithm a PSNR of 38.03 dB.

III. Refinement by Exemplar Based Learning

There are limits on how well interpolation algorithms such as deinterlacing and super-resolution can perform without strong priors on the statistics of natural images [13]. In this section, we present an exemplar-based learning algorithm in the spirit of [12], [13] to refine the output of our MRF-based algorithm. Learning a correction rather that starting from scratch is a form of normalization and allows us to remove any exemplars that may lead to gross artifacts. In Section III-A, we describe how the training set can be augmented with exemplars taken from static regions of the same video in an online fashion to improve the performance further.

We first describe the offline training phase used to collect the set of exemplar pairs $\{(T_i, O_i)\,|i = 1, \ldots, N\}$, where T_i is a vector of features of the input and O_i is the output correction. We begin with a progressive video $I_p(x, y, t)$. We generate an interlaced input video $I_i(x, y, t)$ by discarding the appropriate pixels. We then run our MRF-based deinterlacing algorithm. (To date we have only used training images rather than video. We ran a reduced version of our MRF-based deinterlacing algorithm that can only use spatial interpolants; i.e., we remove F_2, F_5, F_6, and F_7 from the list of possible interpolants in the training phase. The learnt correction is then only applied when one of the spatial labels is chosen.) Let the output be $I_o(x, y, t)$. We compute the high frequency correction:

$$I_h(x, y, t) \;=\; I_p(x, y, t) - I_o(x, y, t) \tag{13}$$

and learn this correction as a function of the mid frequency:

$$I_m(x, y, t) \;=\; I_o - G_\sigma * I_o, \tag{14}$$

where G_σ is a Gaussian blur kernel, with $\sigma = 2.0$, and $*$ denotes convolution. We compute a normalization factor:

$$I_n(x, y, t) \;=\; \sqrt{G_\sigma * I_m^2 + \epsilon}, \tag{15}$$

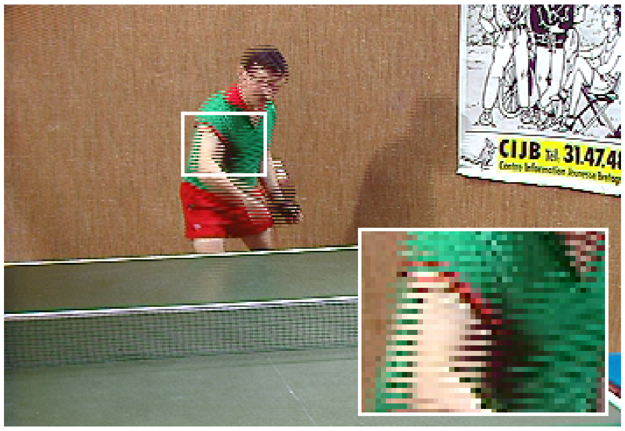
where ϵ is a small positive regularizing constant. We then normalize $\overline{I}_h(x, y, t) = I_h(x, y, t)/I_n(x, y, t)$ and $\overline{I}_m(x, y, t) = I_m(x, y, t)/I_n(x, y, t)$. For each missing pixel (x, y, t) in the interlaced video $I_i(x, y, t)$, we then generate an exemplar with a 20 dimensional feature vector:

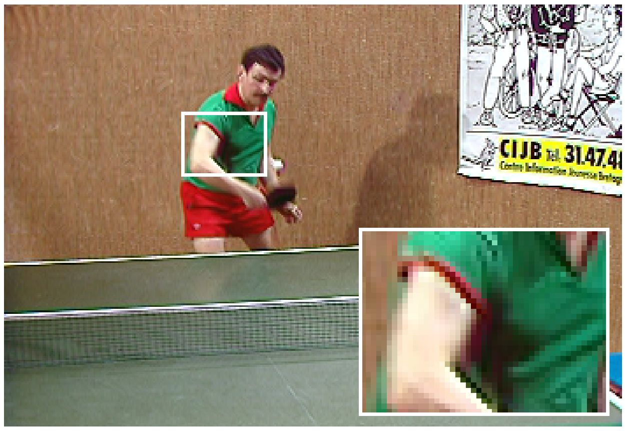
$$T \;=\; (\overline{I}_m(x + j, y + k, t)), \tag{16}$$

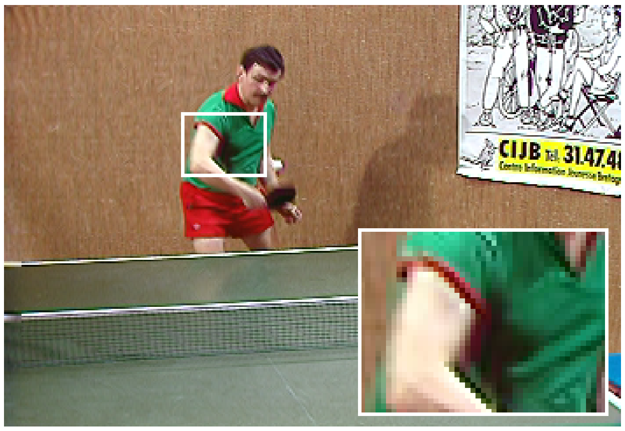
where $j = -2, -1, 0, 1, 2$ and $k = -3, -1, 1, 3$. We also generate an output:

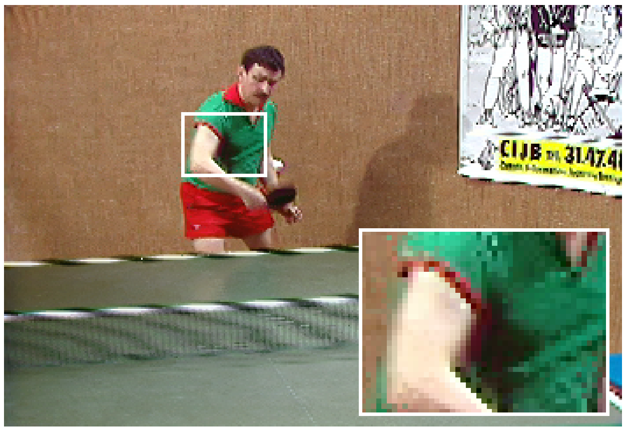
$$\begin{aligned} O \;=\; & (\overline{I}_h(x - 2, y, t), \overline{I}_h(x - 1, y, t), \overline{I}_h(x, y, t), \\ & \overline{I}_h(x + 1, y, t), \overline{I}_h(x + 2, y, t)). \end{aligned} \tag{17}$$

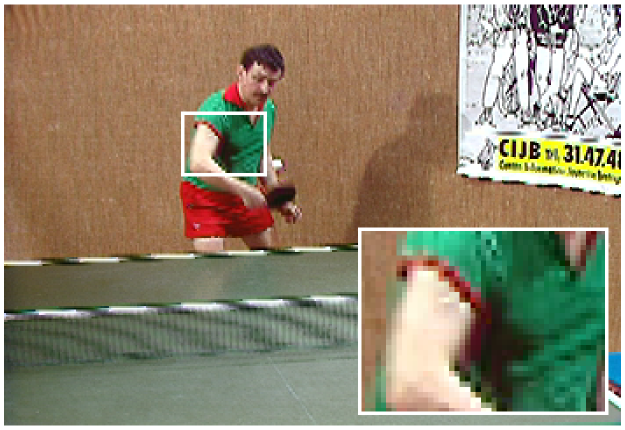
We add the exemplar (T, O) to the training set if: (1) it is interesting enough (we discard exemplars where the magnitude


(a) Input (Weave)


(b) 4-Label


(c) 7-Label


(d) Smart v2.8


(e) Gunnar Thalin

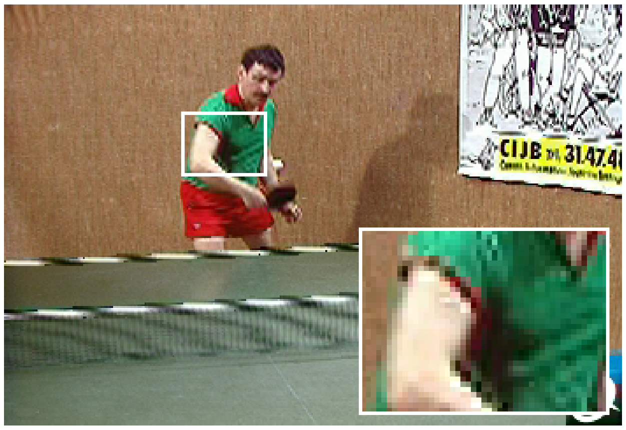
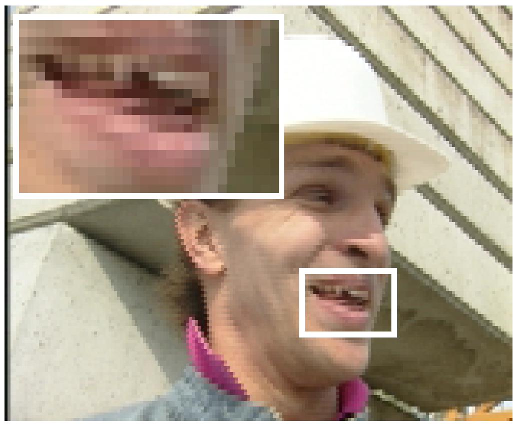

(f) Alparysoft v2.0

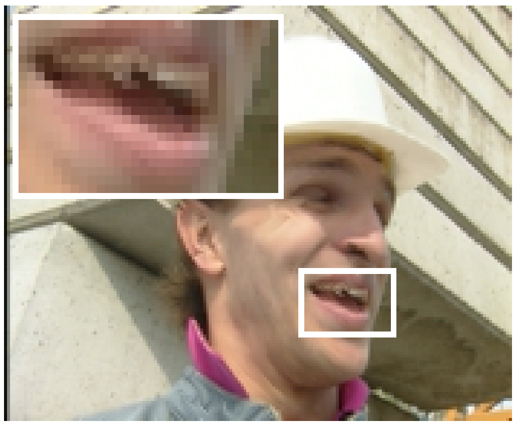
Fig. 10. One frame from the TableT sequence comparing the 4-label and 7-label versions of our algorithm with Smart v2.8 [23], Gunnar Thalin [22], and Alparysoft v2.0 [24]. The cropped hand region shows both versions of our algorithm to yield more naturally looking folds on the shirt, and a smoother boundary to the arm.

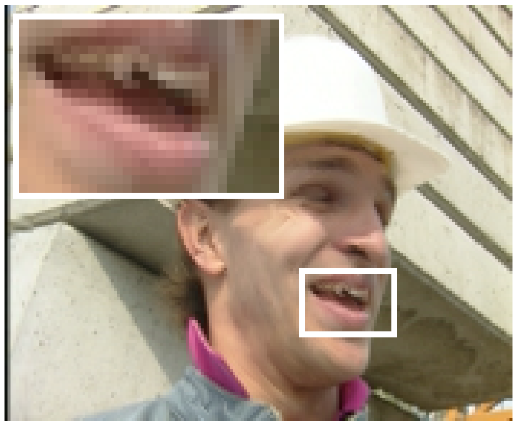
TABLE II

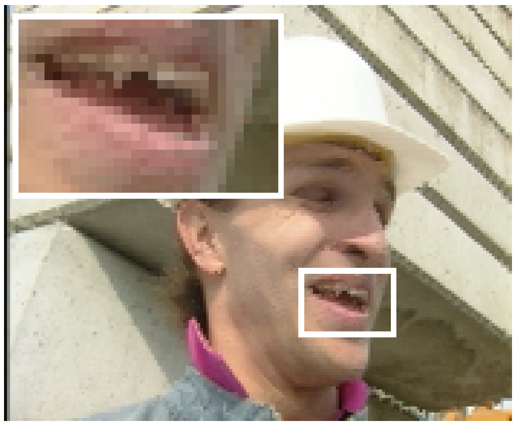
A COMPARISON OF THE 4-LABEL AND 7-LABEL VERSIONS OF OUR ALGORITHM WITH A NUMBER OF RECENTLY PUBLISHED ALGORITHMS ON THE FOREMAN SEQUENCE. BOTH THE 4-LABEL AND 7-LABEL VERSIONS OF OUR ALGORITHM OUTPERFORM ALL OF THESE RECENTLY PROPOSED ALGORITHMS.

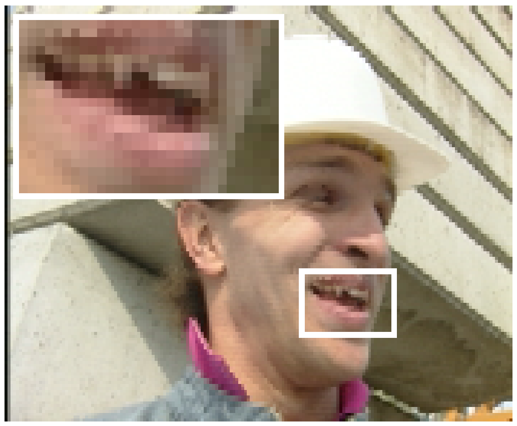
	4-Label	7-Label	Wang *et al.* [8]	Chang *et al.* [9]	Ouyang *et al.* [10]	Li *et al.* [11]	Lee *et al.* [3]
PSNR	37.77dB	38.03dB	31.50dB	34.77dB	34.92dB	$\approx$ 35.50dB	34.00dB


(a) Input (Weave)


(b) 4-Label


(c) 7-Label


(d) Smart v2.8


(e) Gunnar Thalin

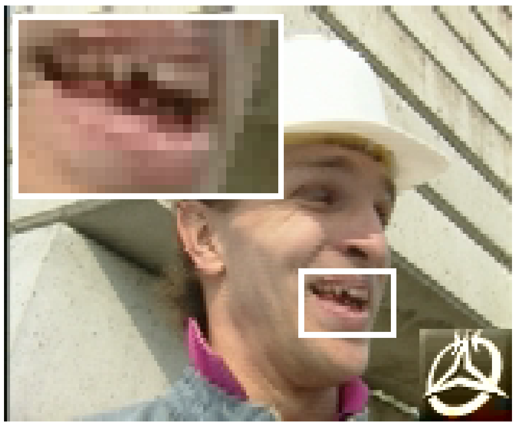

(f) Alparysoft v2.0

Fig. 11. One frame from the Foreman sequence comparing the 4-label and 7-label versions of our algorithm with Smart v2.8 [23], Gunnar Thalin [22], and Alparysoft v2.0 [24]. Closeups of the lip region show that both versions of our algorithm yielded a sharper mouth region. In particular, note that the lower lip looks more natural and less blocky with our algorithms. The diagonal lines on the wall behind the foreman also look less jagged with our algorithm.

of the gradient is too low), and (2) the output is not extreme (we discard exemplars where the magnitude of the correction O is greater than that of both of its neighbors on the same row).

In the online phase of the algorithm, we compute I_o, I_m, I_n, and $\overline{I}_m$ for the interlaced video input. For each missing pixel (x, y, t), we compute the mid-frequency feature vector T using Equation (16). We then perform an approximate nearest neighbor search in the exemplar set to find the top 20 corrections $O_i : i = 1, \ldots, 20$, and undo the normalization. We considered 3 different methods to choose the final correction to apply based on these candidates: (1) Mean: We compute the mean of the correction vectors O_i and take the third component as the correction. (2) Median: We compute the median of the third components of the correction vectors O_i. (3) MRF-Based: Following the approach of [12], we set up a 1D MRF along each scan-line using the match cost as the data term and the distance between overlapping elements of the correction vectors as the regularity term. We found the results of taking the mean and the median to be very similar, with the mean slightly better (less than 1% difference in RMS error.) The MRF-based approach is a little worse, with RMS errors typically $3-4\%$ higher. The reason appears to be that the MRF tends to introduce additional smoothing along each scanline. We used the mean to generate results in this paper.

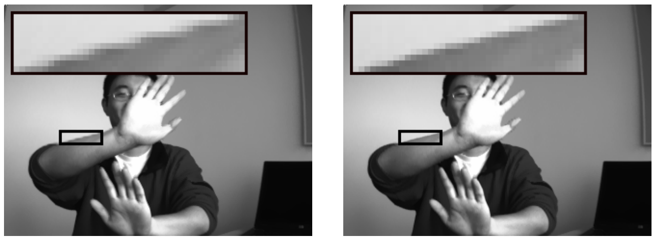
A. *Online Learning*

As described in the introduction, deinterlacing provides the opportunity to use exemplars extracted from the video actually being processed. We now describe an online algorithm to learn video-specific exemplars as the video is processed. A variety of extensions of this algorithm are possible (see Section IV), including batch processing where exemplars are taken from everywhere in the video.

Our algorithm has 2 exemplar sets, a set of generic exemplars learnt before the algorithm is run, and a fixed size set of video specific exemplars, initialized to be empty. As the video is processed, any regions for which the temporal interpolant is chosen are processed in the same way as in the offline training phase in Section III. Exemplars are added to the video specific set as above: (1) if they are interesting enough, and (2) if they are not extreme. When the video specific exemplar set becomes full, exemplars are discarded using a FIFO (first-in first-out) queue. Both the generic and video-specific exemplar sets are then used. Note that errors cannot propagate because there is no feedback.

B. *Experimental Results*

We evaluated our learning-based deinterlacing algorithms on the same 15 videos used in Section II-G. See Figure 2 for the first frame in each sequence. Our generic training data was extracted from 11 images. The images are included in Figure 12. We first present quantitative results for the generic offline learning algorithm (Section III-B1). In Section III-B2, we present qualitative results for the generic offline learning algorithm. In Section III-B3, we present qualitative results for the online learning algorithm. Finally, in Section III-B4, we present timing results for the learning algorithm.



(a) Without Learning (b) Generic Offline Learning

Fig. 13. One frame from the Wave sequence comparing our algorithm with and without generic offline exemplar-based learning. With exemplar-based learning the edges are sharper and almost no jagged effects remain.

1) Quantitative Results for Generic Offline Learning: We first performed a quantitative evaluation of how much the learning-based refinement helps. We performed two comparisons. The first was on the 6-label version of our algorithm, without the optical flow interpolant F_7. We found that the generic offline learning algorithm described in Section III resulted in a 3.0% reduction in the RMS intensity error, computed on average over all 15 sequences. Of the 15 sequences, 14 sequences showed no change or a moderate improvement. Only one sequence (Talking) showed worse results, the most likely reason for which is that the training images are less well matched to this particular video. A larger and more diverse collection of training data might remedy this situation. We also compared with the 7-label version of our algorithm. For this version of the algorithm, our generic offline algorithm resulted in a reduction in the RMS intensity error of 2.0%, again computed on average over all 15 sequences.

2) Qualitative Results for Generic Offline Learning: In Figure 13, we show one frame from the Wave sequence comparing our algorithm with and without the generic offline exemplar-based learning refinement. The results show that with exemplar-based learning the edges are sharper and less jagged.

3) Qualitative Results for Online Learning: In Figure 14 we show one frame from the Mom sequence comparing our exemplar-based learning algorithm with and without exemplars added in an online fashion from the same video, as described in Section III-A. With the online addition of exemplars from the same video, the results are far sharper with fewer gross artifacts. Note that vertically moving, thin horizontal structures such as the eyebrows and eyelids are one of the hardest cases for deinterlacing algorithms.

4) Timing Results: On average across the 15 sequences, the exemplar-based refinement takes 3.9 secs per frame. The addition of the additional training samples takes 12.6 secs per frame in the online learning algorithm. Both of these timing results were computed on an HP nc8430 2.0GHz Core Duo laptop (2GB RAM, 4MB L2 cache, 667MHz front side bus). Neither of these techniques is fast enough for real-time processing, but could be used either for offline enhancement or to view or print a single frame.

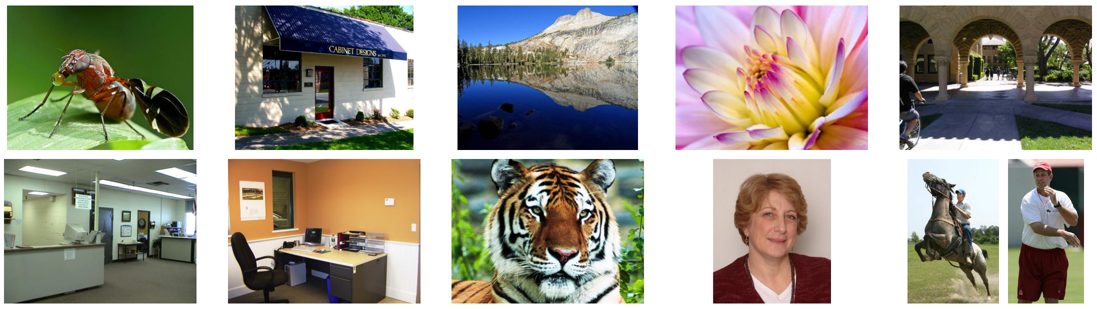
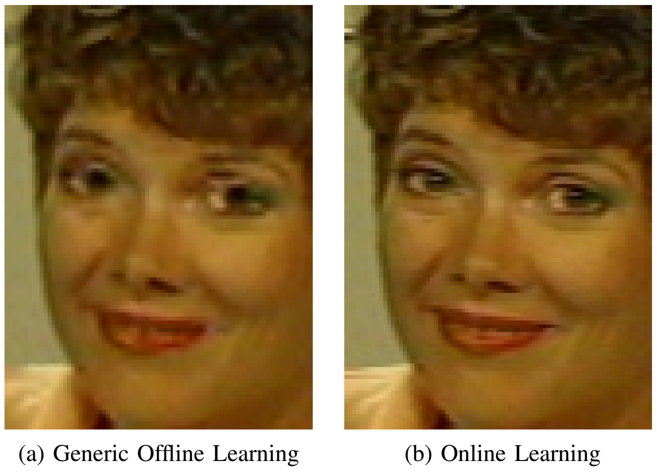


Fig. 12. The 11 training images used by our learning-based refinement algorithms.



(a) Generic Offline Learning (b) Online Learning

Fig. 14. One frame from the Mom sequence comparing the use of offline generic learning with online learning by automatically adding exemplars from static regions of the same video. With the online learning, the results are far sharper and there are fewer gross artifacts.

IV. Conclusion

In the first part of this paper, we described an efficient MRF-based deinterlacing algorithm. The main contribution is to formulate the MRF over interpolation functions rather than pixel intensities. This yields significant speedup over intensity-based MRF. The proposed algorithm also compares favorably with recently reported approaches and several commercial systems in terms of output quality.

The key component in our algorithm is the data cost. The data cost is evaluated by downsampling the video in both space and time and evaluating the interpolants on the resulting progressive videos. This data cost embeds the key assumption made by our algorithm that if an interpolant performs well on the downsampled video, then it can also be expected to perform well in the original video. In general, the downsampling could hide texture at the original resolution. But if there is an edge at one resolution, there is likely to be one at the lower resolution. Similarly, the motion at both the high and low resolutions should be the same. So if there is motion at one resolution, there will be at the other, and optical flow will be correspondingly accurate.

In the second part of the paper, we described an exemplar-based learning algorithm to refine the solution. Here, we feel the most interesting part is the use of exemplars extracted from stationary or near stationary parts of the same video. Our algorithm was based on a simple FIFO queue. Two possible areas for future work are: (1) better algorithms for choosing which exemplars to discard, and (2) the use of spatio-temporal locality or tracking to match exemplars better.

References

[1] G. de Haan and E. B. Bellers, “Deinterlacing - an overview,” *Proceedings of the IEEE*, vol. 86, no. 9, pp. 1839 – 1857, 1998.

[2] H. Y. Lee, J. W. Park, T. M. Bae, S. U. Choi, and Y. H. Ha, “Adaptive scan rate up-conversion system based on human visualcharacteristics,” *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 999–1006, 2000.

[3] G. Lee, H. T. Li, M.-J. Wang, and H.-Y. Lin, “Motion adaptive deinterlacing via edge pattern recognition,” in *International Symposium on Circuits and Systems*, 2007.

[4] P. Delogne, L. Cuvelier, B. Maison, B. V. Caillie, and L. Vandendorpe, “Improved interpolation, motion estimation, and compensation for interlaced pictures,” *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 482–491, 1994.

[5] L. Vandendorpe, L. Cuvelier, B. Maison, P. Queluz, and P. Delogne, “Motion-compensated conversion from interlaced to progressive formats,” *Signal Processing: Image Communication*, vol. 6, no. 3, pp. 193–211, 1994.

[6] A. J. Patti, M. I. Sezan, and A. M. Tekalp, “Robust methods for high quality stills from interlaced video in the presence of dominant motion,” *IEEE Circuits and Systems for Video Technology*, vol. 7, no. 2, pp. 328–342, 1997.

[7] S. Yang, Y.-Y. Jung, Y. H. Lee, and R.-H. Park, “Motion compensation assisted motion adaptive interlaced-to-progressive conversion,” *IEEE Circuits and Systems for Video Technology*, vol. 14, no. 9, pp. 1138–1148, 2004.

[8] D. Wang, A. Vincent, and P. Blanchfield, “Hybrid de-interlacing algorithm based on motion vector reliability,” *IEEE Circuits and Systems for Video Technology*, vol. 15, no. 8, pp. 1019–1025, 2005.

[9] Y.-L. Chang, S.-F. Lin, C.-Y. Chen, and L.-G. Chen, “Video de-interlacing by adaptive 4-field global/local motion compensated approach,” *IEEE Circuits and Systems for Video Tech.*, vol. 15, no. 12, pp. 1569 – 1582, 2005.

[10] K. Ouyang, G. Shen, S. Li, and M. Gu, “Advanced motion search and adaptation techniques for deinterlacing,” in *International Conference on Multimedia and Expo*, 2005.

[11] M. Li and T. Nguyen, “A de-interlacing algorithm using markov random field model,” *IEEE Transactions on Image Processing*, vol. 16, no. 11, pp. 2633 – 2648, 2007.

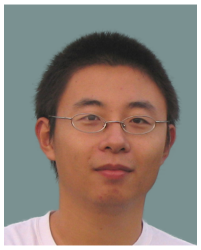
[12] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, “Learning low-level vision,” *International Journal on Computer Vision*, vol. 40, no. 1, pp. 25–47, 2000.

[13] S. Baker and T. Kanade, “Limits on super-resolution and how to break them,” *IEEE Transactions on Pattern Recognition and Machine Intelligence*, vol. 24, pp. 1167–1183, 2002.

[14] Y. Wexler, E. Schechtman, and M. Irani, “Space-time video completion,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

[15] M. Zhao, M. Bosma, and G. de Haan, “Making the best of legacy video on modern displays,” *Journal of the Society for Information Display*, vol. 15, no. 1, pp. 49 – 60, 2007.

[16] G. de Haan, “IC for motion-compensated de-interlacing, noise reduction, and picture-rate conversion,” *IEEE Transactions on Consumer Electronics*, vol. 45, no. 3, pp. 617 – 624, 1999.
[17] M. Biswas, S. Kumar, and T. Nguyen, “Performance analysis of motion-compensated de-interlacing systems,” *IEEE Transactions on Image Processing*, vol. 15, no. 9, pp. 2596 – 2609, 2006.
[18] J. Bouguet, “Pyramidal implementation of the Lucas-Kanade feature tracker: description of the algorithm,” OpenCV Document, Intel Microprocessor Research Labs, Tech. Rep., 2000.
[19] S. Z. Li, *Markov Random Field Modeling in Image Analysis*. Tokyo: Springer-Verlag, 2001.
[20] H. M. Wallach, “Conditional random fields: An introduction,” Department of Computer and Information Science, University of Pennsylvania, Tech. Rep., 2004.
[21] Virtual Dub, http://www.virtualdub.org/index.html.
[22] G. Thalin, http://www.guthspot.se/video/.
[23] Smart v2.8, http://neuron2.net/smart/smart.html.
[24] Alparysoft v2.0, http://www.alparysoft.com/.



Shengyang Dai (S’05) is currently a Ph.D. candidate in the Electrical Engineering and Computer Science department of Northwestern University. His research interests include image/video processing, computer vision, and machine learning. He did summer interns at NEC Laboratories America (Cupertino, CA), Microsoft Research (Redmond, WA), and Google Research (Mountain View, CA) in 2006, 2007, and 2008, respectively. He received his B.S. and M.S. degrees from the Electrical Engineering department of Tsinghua Univresity, Beijing, China, in 2001 and 2004 respectively. He received the Outstanding Graduate Student Fellowship at Tsinghua University in 2004, and the Everly Fellowship at Northwestern University in 2008. He is a Student member of the IEEE.

Simon Baker is a Senior Researcher in the Interactive Visual Media Group at Microsoft Research (MSR). Before joining MSR, he was an Associate Research Professor in the Robotics Institute at Carnegie Mellon University. He obtained a Ph.D. in the Department of Computer Science at Columbia University in 1998, an M.A. in Mathematics from Trinity College, Cambridge in 1995, an M.Sc. in Computer Science from the University of Edinburgh in 1992 and a B.A. in Mathematics from Trinity College, Cambridge in 1991. Simon’s research interests include face recognition, modeling and tracking, human body modeling and tracking, super-resolution, 3D reconstruction, vision for safe driving, projector-camera systems, and all aspects of video processing. Simon was an Associate Editor of the IEEE Transactions on Pattern Analysis and Machine Intelligence from 2004–2008 and was Program Chair for the IEEE Conference on Computer Vision and Pattern Recognition in 2007.

Sing Bing Kang received his Ph.D. in robotics from Carnegie Mellon University, Pittsburgh in 1994. He is currently Principal Researcher at Microsoft Corporation working on image-based modeling as well as image and video enhancement. Sing Bing has co-edited two books in computer vision (”Panoramic Vision” and ”Emerging Topics in Computer Vision”), and coauthored a book on image-based rendering. He has served as area chair and member of technical committee for the three major computer vision conferences (ICCV, CVPR, and ECCV). He has also served as papers committee member for SIGGRAPH’07 and SIGGRAPH Asia’08. Sing Bing is currently CVPR’09 program co-chair and IEEE TPAMI Associate Editor.