Efficient Implementation of Volterra Filters For De-interlacing TV Images – Snell and Wilcox

1 Introduction

A standard TV image is transmitted as a series of horizontal lines. To reduce band-width effects in transmission, half of a picture is transmitted in each frame, *i.e.*, information is only given about the picture on alternate lines, a process called interlacing. A difficulty with this process is that other images (for example computer images) do not have alternate lines omitted. Thus it is desirable to be able to interpolate an existing TV image to obtain information on the image between alternate lines; this is the de-interlacing process.

Interpolation is a well known procedure covered in many numerical analysis textbooks, however special features of the TV image require a special approach to the interpolation process. Firstly, a standard monochrome TV image contains a very large amount of information, typically 576×720 pixels, which have gray scales varying between -128 and 128. Thus any interpolation procedure needs to be fast. Secondly, the procedure should be effective at resolving features with large changes in gray scale - for example a light object placed in a dark background. A problem with many interpolation procedures is that because of a low resolution caused by a low density of sampling points, sharp features can be interpolated by a staircase function causing considerable image distortion.

The objective of this study was to find an interpolation procedure which was fast and resolved edges well. There is a literature on several such methods; for example, the use of wavelets, radial basis functions and neural nets to interpolate images. However, for the purpose of the study group we restricted our attention to the use of cubic Volterra filters which had already been used with some success by Snell and Wilcox. The result of the work is a decomposition of a nonlinear Volterra filter into a product of linear filters for which the coefficients can be optimised efficiently for a given set of training data. Such decompositions had a much smaller RMS error than the comparable linear filters and almost as small an RMS error as the optimal Volterra filter. Although the filters were optimised for a one dimensional aperture, it was also shown how they could be extended to fully two dimensional apertures.

2 Volterra Filters

Suppose that (x_i) is a (large) set of values of gray scale at the pixels $i = 1, \dots, N$ and that (r_i) is the desired interpolant. Each value of r_i will be made up by combining N values of x_i which are arranged in an *aperture.* The basic Volterra filter is a cubic map from (x_i) to (r_i) given by

$$\begin{aligned} r_i = &\sum_{j=0}^{N-1} a_j\, x_{i-j} + \sum_{j,k=0}^{N-1} b_{jk}\, x_{i-j}\, x_{i-k} \\ &+ \sum_{j,k,\ell=0}^{N-1} c_{jk\ell}\, x_{i-j}\, x_{i-k}\, x_{i-\ell}. \end{aligned} \tag{1}$$

Here N is the number of terms in the aperture and also the order of the filter, which comprises linear, quadratic and cubic sections. The complexity of this filter, (*i.e.*, the total number of operations needed to calculate each r_i) increases in proportion to N^3, and this can be very high for large N. Thus, although the Volterra filter can be effective in giving improved image quality, it is costly to implement. We note that by exploiting symmetry (for example replacing a sum over all values of j, k, l by a sum over $0 \le j \le k \le l$ the cost of implementation can be reduced although it is still proportional to N^3. Observe, however, that r_i is a linear function of the coefficients a_j, b_{jk}, c_{jkl}. Thus, if r_i and x_i are both known, for example in a set of training data, then the coefficients can be estimated easily by minimising the RMS error between the predicted and observed interpolants or, equivalently, by solving the normal equations for the resulting over determined system relating x_i to r_i. Thus, although implementing an efficient Volterra filter is hard, finding an optimal Volterra filter is comparatively easy.

We considered two procedures for reducing the cost of implementing the filter in (1):

1. Decomposing the Volterra filter into the product of linear filters, optimising these designs using the training data and testing them against further data;
2. Considering constraints on the coefficient desirable for the detection of features such as edges and hence reducing the number of terms which need to be calculated.

3 Linear Decompositions of Volterra Filters

3.1 One-dimensional apertures

The work in this section was motivated by the papers [1],[2] on MMD (Multi-Memory Decompositions) decompositions of Volterra filters.

Initially we consider the data from the pixels to be a one-dimensional data set, for example taking an aperture to be N pixels arranged in a vertical line, (x_i), producing a one-dimensional interpolant (r_i). This is not strictly the case for two-dimensional apertures, nor does it consider the effects of the boundary of the image. We return to this in the following subsection.

When implementing a Volterra filter certain symmetry restrictions can be applied. For example if we transform x_i to $-x_i$ we would expect r_i to transform to $-r_i$. In Appendix 1 we show that this means that quadratic terms in the filter should be excluded and hence we consider decomposing a linear plus cubic volterra filter of the form

$$r_i = \sum_{j=0}^{N-1} a_j \, x_{i-j} + \sum_{j,k,l=0}^{N-1} c_{jkl} \, x_{i-j} \, x_{i-k} \, x_{i-l}. \tag{2}$$

Now, consider a set of linear filter banks of the form shown in Figure 1. Here we have

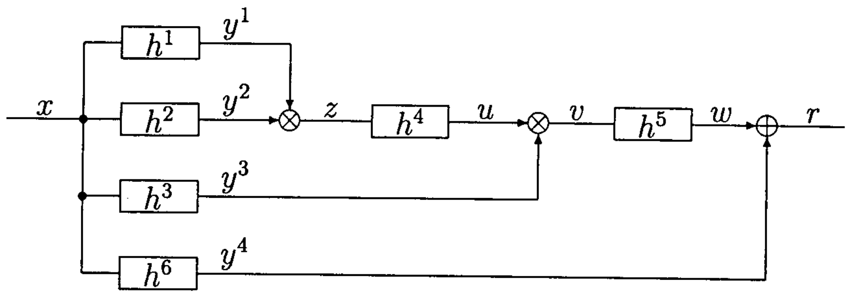

Figure 1: Filter structure approximating the cubic Volterra filter.

$$
\begin{aligned}
y_i^1 &= \sum_{j=0}^{NA-1} h_j^1\, x_{i-j}, \\
y_i^2 &= \sum_{j=0}^{NA-1} h_j^2\, x_{i-j}, \\
z_i &= y_i^1 y_i^2, \\
u_i &= \sum_{j=0}^{NB-1} h_j^4\, z_{i-j}, \\
y_i^3 &= \sum_{j=0}^{NA+NB-1} h_j^3\, x_{i-j}, \\
v_i &= u_i y_i^3, \\
w_i &= \sum_{j=0}^{NC-1} h_j^5 v_{i-j}, \\
y_i^4 &= \sum_{j=0}^{N} h_j^6 x_{i-j}, \\
r_i &= y_i^4 + w_i.
\end{aligned}
\tag{3}
$$

Observe that r_i is a cubic function of x_i so this filter is a special example of a Volterra filter. This architecture can be easily implemented in MATLAB.

The filter architecture is characterised by the orders NA, NB, NC, N of the *linear* filters and by the resulting vectors of coefficients $h^1, h^2, \ldots, h^6$ of the filter banks. Simple counting arguments show that

$$NA + NB + NC - 2 = N. \tag{4}$$

Furthermore, we observe that for a filter described by the vectors

$$\mathbf{h^1}, \mathbf{h^2}, \mathbf{h^3}, \mathbf{h^4} \text{ and } \mathbf{h^5}$$

the same output will be given by a filter described by the vectors

$$\lambda_1\mathbf{h^1}, \lambda_2\mathbf{h^2}, \lambda_3\mathbf{h^3}, \lambda_4\mathbf{h^4} \text{ and } \lambda_5\mathbf{h^5},$$

provided that $\lambda_1\lambda_2\lambda_3\lambda_4\lambda_5 = 1$. Hence, we can (without significant loss of generality) take

$$h_1^1 = h_1^2 = h_1^3 = h_1^4 = 1. \tag{5}$$

For perfect resolution of constants, (*i.e.*, if $x_i = C$ for all i then we should also have $r_i = C$) this implies that

$$\left(\sum_{j=0}^{NA-1} h_j^1\right)\left(\sum_{j=0}^{NA-1} h_j^2\right)\left(\sum_{j=0}^{NA+NB-1} h_j^3\right)\left(\sum_{j=0}^{NB-1} h_j^4\right)\left(\sum_{j=0}^{NC-1} h_j^5\right) = 0 \tag{6}$$

and that

$$\sum_{j=0}^{N-1} h_j^6 = 1. \tag{7}$$

We found it useful not to **impose** (6) and (7) but rather to monitor these conditions as a measure of the accuracy of the solution.

To calculate each new data point, the number of multiplications is

$$3NA + 2NB + NC + N + 1, \tag{8}$$

which is much smaller than N^3 when N is large. Thus if such filters can be optimised and behave similarly to a Volterra filter, they offer a significant speed up in terms of cost.

To determine the optimal such filter requires the calculation of the coefficients of the vectors $\mathbf{h}^1, \ldots, \mathbf{h}^6$ for which there are

$$3NA + 2NB + NC + N - 6 \tag{9}$$

degrees of freedom. We do this as follows. Given the filter as implemented we set

$$d_{jkl} = \sum_{m=0}^{NC-1} h_m^5 h_{j-m}^3 \left(\sum_{n=0}^{NB-1} h_n^4 h_{k-m-n}^1 h_{l-m-n}^2\right). \tag{10}$$

To ensure symmetry of the Volterra filter (and also to reduce the cost of implementing the filter by exploiting symmetry) we then set

$$c_{jkl} = \frac{1}{6}\left(d_{jkl} + d_{jlk} + d_{kjl} + d_{klj} + d_{lkj} + d_{ljk}\right). \tag{11}$$

The values of the coefficients h are then optimised by comparing the interpolants against a set of *training data,* in which the values of x_i are given together with the correct interpolant, R_i. Suppose that the calculated values of the interpolants for a given set of values for the coefficients h are given by r_i then

$$r_i = \sum_{j=0}^{N-1} h_j^6 x_{i-j} + \sum_{j,k,l=0}^{N-1} c_{jkl}\, x_{i-j}\, x_{i-l}\, x_{i-k}. \tag{12}$$

Now construct the vectors

$$\mathbf{z} = [h_1^6, \dots, h_{N-1}^6, 0, \dots, 0, c_{000}, c_{001}, \dots, c_{N-1,N-1,N-1}]^T, \tag{13}$$

where the zero elements correspond to the unused quadratic terms in the filter and

$$\mathbf{r} = (r_0, r_1, \dots, r_{N-1})$$

We then have

$$\mathbf{r} = M\mathbf{z} \tag{14}$$

where M is a (large) matrix with the same number of rows as the number of interpolated data points and is such that the i-th row has the form

$$[x_i,\ x_{i-1},\ \dots,\ x_{i-N+1}, x_i^2,\ \dots,\ x_{i-N}^2,\ x_i^3,\ x_i x_i x_{i-1}\ \dots] \tag{15}$$

To minimise the RMS error between the predicted and the actual interpolants we seek to minimise

$$(\mathbf{R} - \mathbf{r})^T(\mathbf{R} - \mathbf{r}) \tag{16}$$

where $\mathbf{R}$ is the vector of actual interpolants. That is, we aim to minimize the function

$$f = \mathbf{R}^T\mathbf{R} - 2\mathbf{z}^T M^T \mathbf{R} + \mathbf{z}^T M^T M \mathbf{z}. \tag{17}$$

Observe that, whereas M and $\mathbf{R}$ are large, the matrix $M^T M$ and the vector $M^T\mathbf{R}$ are relatively small.

Our procedure to calculate the vectors $\mathbf{h}^1, \dots, \mathbf{h}^6$ is then as follows:

1. For $\mathbf{h}^1, \mathbf{h}^2, \ldots, \mathbf{h}^6$ calculate c_{jkl} using (10) and (11);
2. Form the vector $\mathbf{z}$ in (13);
3. Calculate f from (17);
4. Minimise f over all vectors $\mathbf{h}^1, \ldots, \mathbf{h}^6$.

This is an unconstrained nonlinear minimisation process.

It was initially implemented using the MATLAB minimisation routine. This worked, but took a considerable time to find the minimum (typically about 20 minutes). Subsequently, it was implemented using the NAG routine E04JAF and then took only a few seconds to find the minimum. A FORTRAN code to implement this algorithm is given in Appendix 2. To test for the problem of finding false minima, the routine was started from a variety of initial guesses for the vectors $\mathbf{h}_i$.[1] In practice, it was found that setting all initial values of $\mathbf{h}_i$ to zero, apart from those in (5), resulted in good convergence of the minimization. The algorithm was implemented for the set of training data labeled **girl** with an aperture of 4 vertical pixels so that $N = 4$. In this case there are ten possible architectures (*i.e.*, there are 10 ways of choosing NA, NB, NC so that $NA + NB + NC = 6$). For each case we calculated the optimal set of filter coefficients the amount of work per calculation of each r_i, and the resulting RMS error. The results were as follows:

NA	NB	NC	Workload	RMS error
1	1	4	16	18.4
1	2	3	17	17.3
1	3	2	18	16.9
1	4	1	19	17.4
2	1	3	18	17.3
2	2	2	19	16.9
2	3	1	20	17.3
3	2	1	21	17.1
3	1	2	20	16.5
4	1	1	22	16.8

[1]It is worth noting that a feature of the MMD decomposition is that several different vectors of coefficients can combine to give almost identical Volterra filters. For example, interchanging $\mathbf{h}^1$ and $\mathbf{h}^2$ does not change the resulting Volterra filter.

For the same data the optimal *linear* filter (*i.e.*, the filter with $c_{j,k,\ell} = 0$ $\forall j, k, \ell$) had an RMS error of 18.45 and the optimal *Volterra* filter (in which the coefficients c_{jkl} were unconstrained), had an RMS error of 16.019.

Observe from this table, that the simplest representation of the Volterra filter given by $NA = NB = 1$, or

$$r_i = \sum_{j=0}^{N-1} h_j^6 \, x_{i-j} + \sum_{j=0}^{N-1} h_j^4 \, x_{i-j}^3,$$

and called the Hammerstein filter, performs rather poorly. Indeed it is only slightly better than the linear filter.

In contrast, the best architecture seems to be given by $(NA, NB, NC) = (3, 1, 2)$, resulting in an RMS error of 16.5—which is not too different from that of the optimal Volterra filter. From a zero initial guess (subject to (5)), the resulting optimal coefficients are:

$\mathbf{h}_1 = [1, 80.5411, -58.7979]$;
$\mathbf{h}_2 = [1, -1.2917, -0.07452]$;
$\mathbf{h}_3 = [1, -0.1313, -0.9131]$;
$\mathbf{h}_4 = [1]$;
$\mathbf{h}_5 = [-3.5163 \times 10^{-7}, 3.6224 \times 10^{-7}]$;
$\mathbf{h}_6 = [-0.1034, 0.61278, 0.5851, -0.0952]$.

Observe that the values of $\mathbf{h}_5$ are small as an effect of x_i^3 having a value of the order of 10^6.

Given a different initial guess it is possible that different filter coefficients will be obtained, but we observed that the resulting Volterra filters all had almost identical RMS errors - indicating strongly that we have different decompositions of essentially the same filter.
We also optimised for the case $(NA, NB, NC) = (2, 2, 2)$ which gave an RMS error of 16.9—rather larger. In this case the optimal filter coefficients are given by:

$\mathbf{h}_1 = [1, -1.172]$;
$\mathbf{h}_2 = [1, 1.604]$;
$\mathbf{h}_3 = [1, -0.123, 0.867]$;
$\mathbf{h}_4 = [1, -0.827]$;
$\mathbf{h}_5 = [-6.024 \times 10^{-6}, 6.9776 \times 10^{-6}]$;
$\mathbf{h}_6 = [-0.089, 0.593, 0.581, -0.086]$.

Note that in both filters $h_1^5 + h_2^5 \sim 0$ and $\sum_{j=0}^{3} h_j^6 \sim 1$, agreeing with (6,7). In neither case is the linear filter symmetric, although it does closely resemble the 'optimal' sinc filter. This may be the effect of not correctly including the boundary conditions in the covariance matrix.

To compare the performance of the resulting filters we try them on some data with sharp changes, and also for smoothly varying sinusoid data. In particular (using Matlab notation)
$\mathbf{X}_1 = [100 * ones(100,1); -100 * ones(100,1)]$,
$\mathbf{X}_2 = [100 * ones(100,1); 0 * ones(100,1)]$,
$\mathbf{X}_3 = 100 * \cos([0 : 0.1 : 10])$.
When using the $(3,1,2)$ architecture the results are given in Figures 2, 3 and 4.

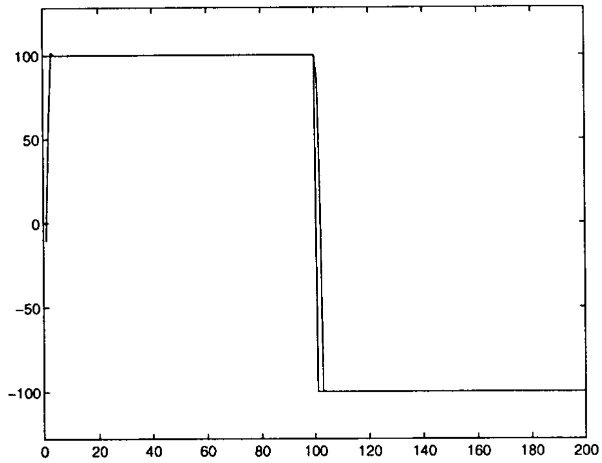

Figure 2: Interpolation of X_1 by the (3,1,2) filter

We see that in Figure 2 and Figure 4 the filter introduces a small delay but otherwise reproduces the function well. In Figure 2 the filter correctly reproduces the leading edge of the discontinuity but gives an overshoot at the trailing edge.

The corresponding results for the $(2,2,2)$ architecture are given in Figures 5, 6 and 7. The quality of the interpolation in these cases is clearly poorer than in the previous figures, with greater overshoots.

Both of these filters show improvements over an optimal linear filter, which introduces greater overshoot at the discontinuity, illustrated in Figure 8.

We conclude from this that the $(3,1,2)$ architecture performs very nearly as well (judging on RMS error) as the optimal Volterra filter and certainly very much better than the optimal linear filter for a four pixel vertical aper-

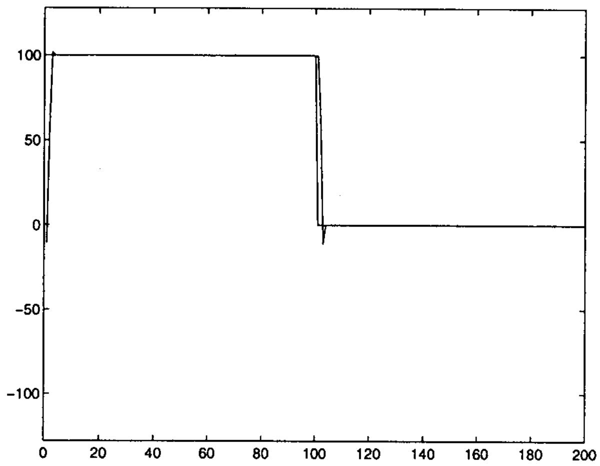

Figure 3: Interpolation of X_2 by the (3,1,2) filter

ture.

We now consider generalising this approach to data sets from apertures extending both vertically and horizontally.

The filter structure described in [1],[2] (see Figure 1), acts on a stream of *scalars.* That means the data is one-dimensional, which is rarely the case when we consider television images, or rather the only apertures we can have with the scalar filter are one-dimensional, as to the left in Figure 9.
If we use an aperture as shown in the middle of Figure 9, we do not have a stream of scalars but rather a stream of vectors. In Figure 1 all the variables should be vectors except the final output r and its components w and y^4.

A linear filter h is characterized by its length N and it acts by $y_n = \sum_{i=0}^{N-1} h_i x_{n-i}$, where $h_i \in \mathbb{R}$. If we want x_n to be a vector of length K and y_n to be a vector of length L then we just have to replace the scalar h_i with a $K \times L$ matrix $\underline{\underline{h}}_i$, *i.e.*, we have

$$\mathbf{y}_n = \sum_{i=0}^{N-1} \underline{\underline{h}}_i \mathbf{x}_{n-i}.$$

Thus, a linear vector filter is characterized by three numbers, the length N, the size of the input vector K and the size of the output vector L. The multiplication operator $\otimes$ takes two vectors and multiply the entries with each other

$$(x_1, \ldots, x_K) \otimes (y_1, \ldots, y_K) = (x_1 y_1, \ldots, x_K y_K).$$

All in all we have the structure in Figure 10. Let us return to the linear

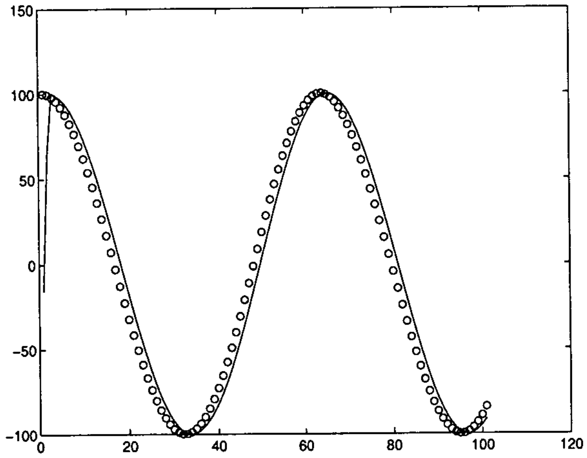

Figure 4: Interpolation of X_3 by the (3,1,2) filter

vector filter $\mathbf{y}_n = \sum_{i=0}^{N-1} \underline{\underline{h}}_i \mathbf{x}_{n-i}$ where $\underline{\underline{h}}_i \in \mathbb{R}^{K \times L}$. There is no reason for L to be large than the highest rank of $\underline{\underline{h}}_i$, $i = 0, \ldots, N-1$, so we may assume that $L \leq K$.

The number of multiplications of two variables in the vector filter in Figure 10 is $K + M \leq 2K$. The exact numbers for additions and multiplications by constants are rather complicated but, if we replace L and M by K, then they both are of the order $(N_a + N_4 + N_5)K^2$.

Just as in the scalar case, multiplying each of the linear components of the cubic filter by a scalar λ_1, λ_2, λ_3, λ_4, and λ_5, corresponds to multiplying the final result w by the product of the scalars $\lambda_1\lambda_2\lambda_3\lambda_4\lambda_5$, but in the case of vector-streams we can do more. We may replace λ_1 and λ_2 by $L \times L$-matrices and λ_3 and λ_4 by $M \times M$-matrices, but then it is no longer clear what the effect on w is.

3.2 A restricted class of Volterra filters

Suppose we want an aperture as to the right in Figure 9, this can not be achieved by considering the images as a stream of vectors or scalars. We need to forget about the streams and just have a certain aperture of say N pixels. One way of looking at a cubic Volterra filter is to consider it as a cubic approximation to some unknown (and probably nonexistent) perfect interpolation function. Instead of looking at an arbitrary cubic functions we can look at cubic functions of the form

$$\sum_{k=1}^{M} (a_1^k x_1 + \cdots + a_N^k x_N)(b_1^k x_1 + \cdots + b_N^k x_N)(c_1^k x_1 + \cdots + c_N^k x_N).$$

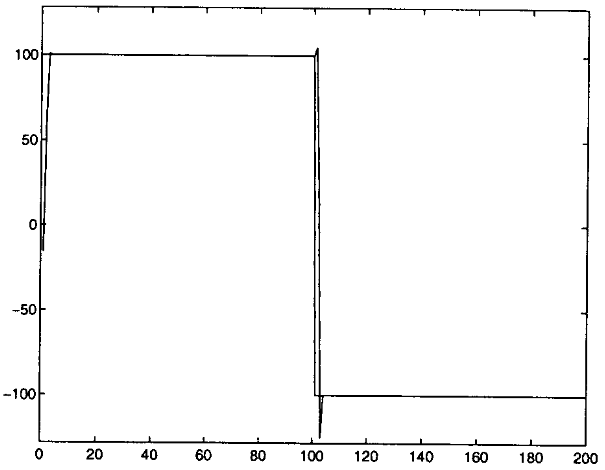

Figure 5: Interpolation of X_1 by the (2,2,2) filter

We should note that if M is sufficiently large then we can get any Volterra filter but the cost is then just as high. The cost of this filter is:

additions	$3M(N-1)$;
multiplication by constant	$3MN$;
multiplication of two variables	$2M$;

Just as in the two previous cases we should add a linear term.

4 Subclasses of Volterra filters with good interpolation properties

We remarked in Section 2 that there is no simple algorithm for finding an efficient implementation of a general, high-order, multivariate Volterra filter. Therefore, it makes sense to consider *subclasses* of Volterra filters, for which a general implementation algorithm might be reasonably expected to be found.

Earlier in this report, attention has focussed on a subclasses of filters which are given by a combination of simple filter components, and which are thus, by construction, efficient to implement.

However, in this section we take a different approach. We consider what constitute desirable and sensible properties for a de-interlacing filter. Then, by formulating these properties mathematically, and solving the constraints which thus arise, we are able to describe subclasses of Volterra filters which have good interpolation properties built-in. Typically, there are still some degrees of freedom left in the filter, and these may be trained by test data.

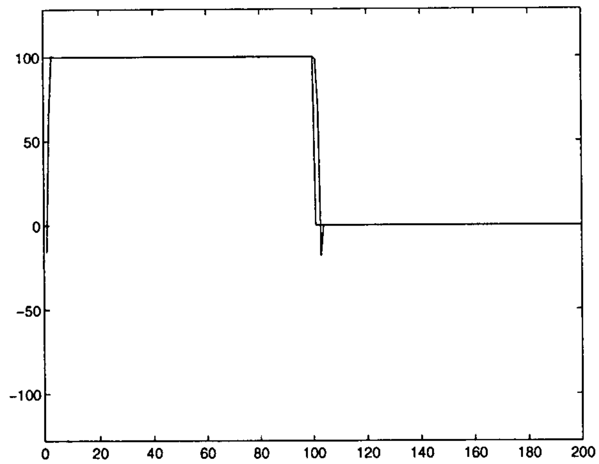

Figure 6: Interpolation of X_2 by the (2,2,2) filter

Because these filters have fewer degrees of freedom than the most general cases, it should be easier to see how they might be implemented efficiently.[2]

After some discussion, we decided that the following were minimum requirements for a sensible filter:

1. Symmetry: if the television image is reflected in either a horizontal or a vertical axis, the actual interpolated values should not change; rather the interpolated field should just be reflected in the same way. This assumption greatly reduces the degrees of freedom by forcing (*e.g.*, pairs of, quads of) coefficients to equal each other. (Of course, we also assume translational symmetry by integer pixel quantities, which means that the same filter is used to find each interpolated point.)

2. We would like the interpolation to be very accurate (if not exact) in regions where the image intensity or voltage varies smoothly (*e.g.*, linearly). Perfect interpolation for a linear profile produces a set of linear constraints on the filter coefficients, all but one of which are homogeneous.

3. Edge resolution: the filter must behave well in the neighbourhood of edges (which are sharp spatial transitions in the 'voltage' describing colour/brightness). In particular, we wish to design filters without Gibbs' phenomenon, where sharp edges induce spurious oscillation in

[2]An interesting and open question is whether the subclasses of filters described in this section overlap with the subclasses of filters discussed elsewhere in this report. We would expect those filters, after they have been trained, to satisfy approximately the properties described here.

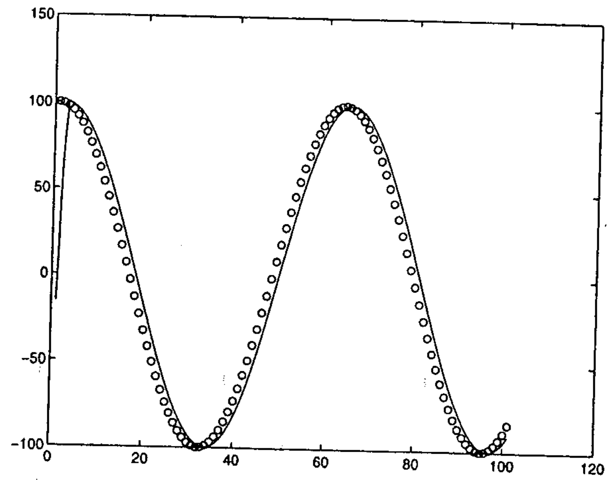

Figure 7: Interpolation of X_3 by the (2,2,2) filter

the voltage levels of nearby interpolated points. Eliminating Gibbs' phenomenon gives an extra set of homogeneous linear constraints on the filter coefficients.

There is an extra reason for considering filters with good built-in edge resolution: currently Snell and Wilcox choose their filter by minimising the two-norm of the interpolation error with respect to some test data. However, the two-norm will not penalise bad interpolation errors in localised regions (*e.g.*, near edges) heavily — hence the optimal filter with respect to the two-norm may give bad edge errors, which are highly visible to the human eye. There are two solutions to this problem:

- Optimise with respect to a higher p-norm (which is less computationally convenient);
-) Continue to optimise with the two-norm, but only over a set of filters (*e.g.*, the class satisfying property 3 above). which already have good edge resolution properties.

To illustrate what might be achieved by our approach, we work through two examples, showing how to construct:

1. Good cubic four point filters in one space dimension (see Figure 11(a));
2. Food six point filters in two space dimensions (see Figure 11(b)).

Higher order filters and larger apertures are amenable to our procedure, but the calculations are much more involved, and we do not consider them here.

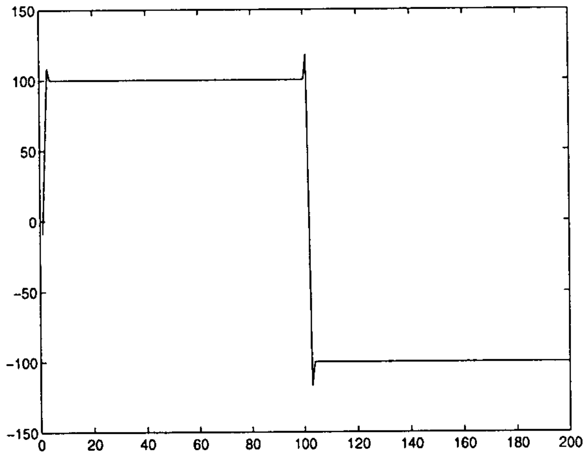

Figure 8: Interpolation of X_1 by the linear filter

$\times$	x_{n-3}	$\times$	$\times$	$\times$	$\mathbf{x}_{n-3}$		$\times$		x_1
$\times$	x_{n-2}	$\times$	$\times$	$\times$	$\mathbf{x}_{n-2}$	$\times$	$\times$	$\times$	x_2, x_3, x_4
$\circ$	y_n		$\circ$		y_n		$\circ$		y
$\times$	x_{n-1}	$\times$	$\times$	$\times$	$\mathbf{x}_{n-1}$	$\times$	$\times$	$\times$	x_5, x_6, x_7
$\times$	x_n	$\times$	$\times$	$\times$	$\mathbf{x}_n$		$\times$		x_8

Figure 9: Example of an aperture for a scalar stream, an aperture for a vector stream and a general aperture.

4.1 Example 1.

We consider the four point filter depicted in Figure 11(a). We suppose that the unknown point, whose voltage V_0 is to be determined by the Volterra filter, is located at $x = 0$, and the four input data points are at $x = x_{-2}$, x_{-1}, x_{+1}, x_{+2}, and have known voltages V_{-2}, V_{-1}, V_{+1}, and V_{+2} respectively. (If $h > 0$ is the inter-pixel spacing, then $-x_{-1} = x_{+1} = h$ and $-x_{-2} = x_{+2} = 3h$.) Our four point cubic Volterra filter calculates the unknown voltage by

$$V_0 = \alpha_0 + \sum_i \alpha_i V_i + \sum_{i \le j} \alpha_{ij} V_i V_j + \sum_{i \le j \le k} \alpha_{ijk} V_i V_j V_k, \qquad i, j \in \{-2, -1, +1, +2\}. \tag{18}$$

Here α_0, α_i, α_{ij}, α_{ijk} are constants which determine the properties of the filter. In this case there are 4 α_i, 10 α_{ij}, and 20 α_{ijk}, and hence altogether

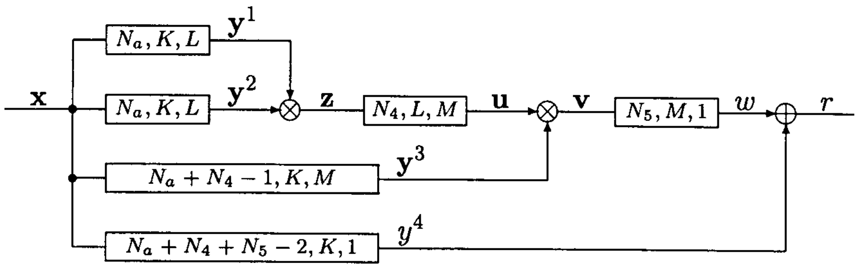

Figure 10: Vector filter structure.

35 degrees of freedom in the filter—we now show how requiring good filter properties reduces the degrees of freedom.

First of all, we impose reflectional symmetry about $x = 0$, *i.e.,* swapping (V_{-2} and V_{+2}) and (V_{-1} and V_{+1}) should not change the interpolated value V_0. This forces $\alpha_{-i} = \alpha_i$, and $\alpha_{-i\,,\,-j} = \alpha_{+i\,,\,+j}$, $\alpha_{-i\,,\,-j\,,\,-k} = \alpha_{+i\,,\,+j\,,\,+k}$ (up to re-ordering of indices). In full, we obtain $\alpha_{-2} = \alpha_{+2}$, $\alpha_{-1} = \alpha_{+1}$ at the linear level of the filter, $\alpha_{-2\,,\,-2} = \alpha_{+2\,,\,+2}$, $\alpha_{-1\,,\,-1} = \alpha_{+1\,,\,+1}$, $\alpha_{-2\,,\,-1} = \alpha_{+1\,,\,+2}$, $\alpha_{-2\,,\,+1} = \alpha_{-1\,,\,+2}$ at the quadratic level of the filter, and $\alpha_{-2\,,\,-2\,,\,-2} = \alpha_{+2\,,\,+2\,,\,+2}$, $\alpha_{-1\,,\,-1\,,\,-1} = \alpha_{+1\,,\,+1\,,\,+1}$, $\alpha_{-2\,,\,-2\,,\,-1} = \alpha_{+1\,,\,+2\,,\,+2}$, $\alpha_{-2\,,\,-2\,,\,+1} = \alpha_{-1\,,\,+2\,,\,+2}$, $\alpha_{-2\,,\,-2\,,\,+2} = \alpha_{-2\,,\,+2\,,\,+2}$, $\alpha_{-2\,,\,-1\,,\,-1} = \alpha_{+1\,,\,+1\,,\,+2}$, $\alpha_{-1\,,\,-1\,,\,+1} = \alpha_{-1\,,\,+1\,,\,+1}$, $\alpha_{-1\,,\,-1\,,\,+2} = \alpha_{-2\,,\,+1\,,\,+1}$, $\alpha_{-2\,,\,-1\,,\,+1} = \alpha_{-1\,,\,+1\,,\,+2}$, $\alpha_{-2\,,\,-1\,,\,+2} = \alpha_{-2\,,\,+1\,,\,+2}$ at the cubic level of the filter.

Together these constraints reduce the degrees of freedom to 19, which may be represented by α_0 together with the vectors

$$\mathbf{a}^1 = (\alpha_{+1},\, \alpha_{+2})^T, \tag{19}$$

$$\mathbf{a}^2 = (\alpha_{+1\,,\,+1},\, \alpha_{+2\,,\,+2},\, \alpha_{-1\,,\,+2},\, \alpha_{+1\,,\,+2},\, \alpha_{-2\,,\,+2},\, \alpha_{-1\,,\,+1})^T, \tag{20}$$

and

$$\mathbf{a}^3 = (\alpha_{+1\,,\,+1\,,\,+1}, \alpha_{+2\,,\,+2\,,\,+2},\, \alpha_{-2\,,\,+1\,,\,+1},\, \alpha_{-1\,,\,+1\,,\,+1},\, \alpha_{+1\,,\,+1\,,\,+2},\, \alpha_{-2\,,\,+2\,,\,+2}, \\ \alpha_{-1\,,\,+2\,,\,+2},\, \alpha_{+1\,,\,+2\,,\,+2},\, \alpha_{-2\,,\,+1\,,\,+2},\, \alpha_{-1\,,\,+1\,,\,+2})^T, \tag{21}$$

which represent the free coefficients at the linear, quadratic, and cubic levels of the filter respectively.

Next, we consider what extra constraints are provided by supposing that a linear variation in voltage is interpolated exactly, *i.e.,* , we set $V_i = (Ah)l_i + B$ (where $-l_{-2} = l_{+2} = 3$, and $-l_{-1} = l_{+1} = 1$), and require that (18) returns $V_0 = B$.

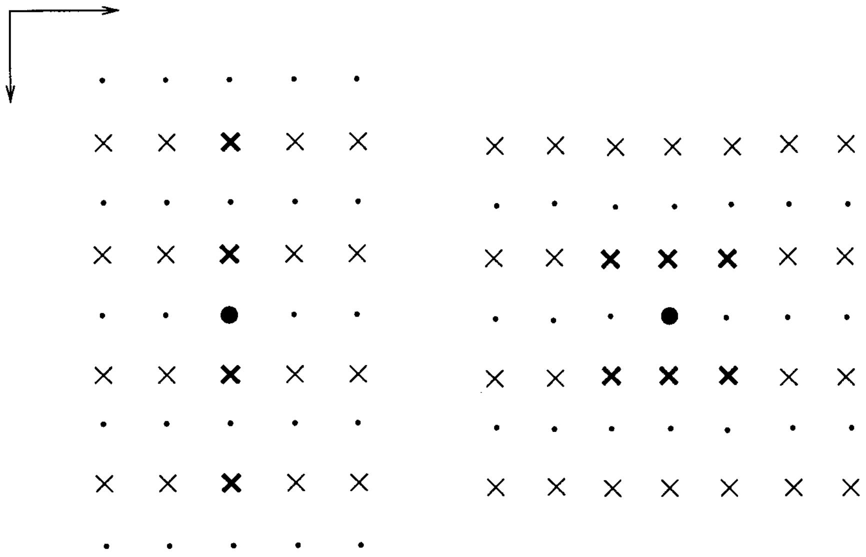

Figure 11: (a) The four point aperture for the one dimensional filter of Example 1; and (b) the six point aperture for the filter of Example 2. In each case, the known field is plotted with '$\times$', and the interpolated field with '$\bullet$'. The bold characters denote exactly which points of the known field are used to determine the value of a particular point in the interpolated field.

We equate polynomial terms in A and B in (18) At the $O(1)$ level we obtain $\alpha_0 = 0$, and also

$$\begin{aligned} O(B): \sum_i \alpha_i = 1, \quad O(B^2): \sum_{i \leq j} \alpha_{ij} = 0, \\ O(B^3): \sum_{i \leq j \leq k} \alpha_{ijk} = 0, \end{aligned} \tag{22}$$

which give

$$(1,1)\mathbf{a}^1 = 1/2, \quad (2,2,2,2,1,1)\mathbf{a}^2 = 0$$

and

$$(2,2,2,2,2,2,2,2,2,2)\mathbf{a}^3 = 0,$$

respectively. Most of the other polynomial terms vanish by the symmetry which has already been imposed, but we do obtain

$$O(A^2): \ (2,18,-6,6,-9,-1)\mathbf{a}^2 = 0, \tag{23}$$

and

$$O(A^2B):\ (2,54,-6,-2,6,-54,-18,18,-18,-6)\mathbf{a}^3 = 0. \qquad (24)$$

Now, let us consider what extra constraints are put on the filter coefficients by requiring that edges are interpolated in an accurate manner. With reference to Figure 12, we take a perfectly sharp edge, *i.e.*, the voltage is a step function with value f_- to the left of the edge and f_+ to the right. Two different edge positions need to be considered:

(a) between x_{+1} and x_{+2}, in which case the filter should give f_- for the interpolated value V_0. (N.B.: because of the symmetry relations which have already been imposed on the coefficients, taking the edge between x_{-2} and x_{-1} gives the same constraints, so we do not consider that position. Further, note that the calculations which follow do not depend on whether $f_- < f_+$ or $f_- > f_+$.);

(b) Also, we have edge position between x_{-1} and x_{+1}, where it is not possible to say what value V_0 should take, because we do not know whether the edge is to the left or right of $x = 0$.

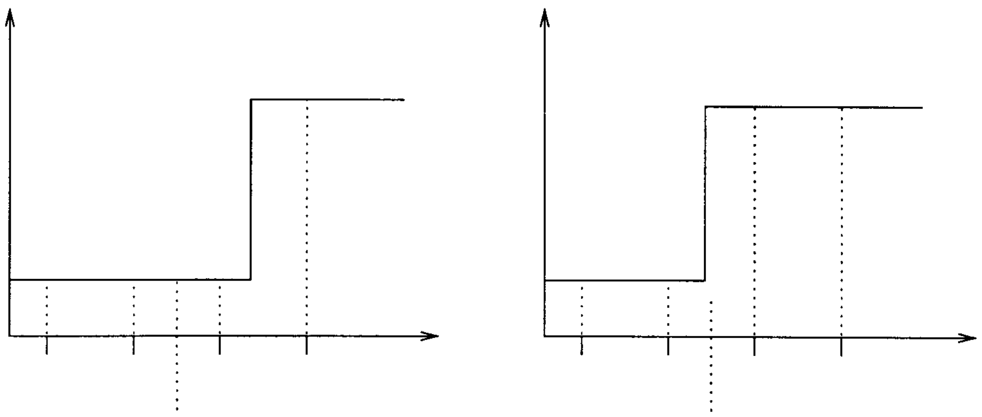

Figure 12: Edge positions for the 1D filter of Example 1. (N.B., here the x-axis runs horizontally, $x = 0$ denotes the interpolated point and $x = x_{-2}$, x_{-1}, x_{+1}, x_{+2} denote the known points.) There are two edge positions up to symmetry: (a) the asymmetric position, where the interpolated point should take the value f_-, and (b) the symmetric position where the value that the interpolated point should take is not clear.

First, consider the edge position of Figure 12(a). For (18) to give $V_0 = f_-$, the linear level $\sum \alpha_i V_i$ alone must give f_-, and the quadratic $\sum \alpha_{ij} V_i V_j$ and cubic $\sum \alpha_{ijk} V_i V_j V_k$ levels must vanish. (To see this, equate polynomial terms in f_- and f_+.) Taking into account the symmetry in the coefficients, the linear level of (18) gives

$$f_- = 2\alpha_{+1} f_- + \alpha_{+2} f_+, \tag{25}$$

which we want to hold *for all* values of f_- and f_+. Consequently,

$$\alpha_{+1} = 1/2 \quad \text{and} \quad \alpha_{+2} = 0, \tag{26}$$

and the linear level of the filter is wholly determined.

Observe that this result is rather different from the optimal sinc function linear filter used for interpolating smooth functions.

Note that for the edge position of Figure 12(b), if V_0 is to be independent of quadratic and higher terms in f_- andf_+, then it must now take the average value $(f_- + f_+)/2$.

We now consider what constraints on $\mathbf{a}^2$ and $\mathbf{a}^3$ are produced by requiring the quadratic and cubic sums of (18) to vanish for each of the two edge positions shown in Figure 12.

First consider the quadratic sum $\sum \alpha_{ij} V_i V_j$, and the edge position of Figure 12(a). Substituting $V_{-2} = V_{-1} = V_{+1} = f_-$ and $V_{+2} = f_+$ in the sum, and setting to zero, yields 3 homogeneous linear constraints on the α_{ij}. However, these constraints add up to give $\sum \alpha_{ij} = 0$, which has already been obtained by assuming the perfect fit of a linear profile. Thus, at most 2 of these constraints are linearly independent with those that have been obtained before — hence we need only note that $O(f_+^2)$ terms give $(0,1,0,0,0,0)\mathbf{a}^2 = 0$, and that $O(f_- f_+)$ terms give $(0,0,1,1,1,0)\mathbf{a}^2 = 0$ (taking into account the symmetry relations between coefficients).

For the quadratic sum and the symmetric edge position of Figure 12(b), the three constraints arising from $O(f_-^2), O(f_- f_+), O(f_+^2)$ also add to $\sum \alpha_{ij} = 0$. Further, because of the symmetry relations that have been imposed on the coefficients, the $O(f_-^2)$ and $O(f_+^2)$ equations are identical; hence there is only one genuinely new constraint which can be linearly independent with those that have been derived previously — so, we need only note that $O(f_+^2)$ terms give $(0,0,2,0,1,1)\mathbf{a}^2 = 0$.

At this stage we have found all the constraints on the coefficient vector $\mathbf{a}^2$ for the quadratic level of the filter. Together they take the form

$$\begin{pmatrix} +2 & +2 & +2 & +2 & +1 & +1 \\ +2 & +18 & -6 & +6 & -9 & -1 \\ 0 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & +1 & +1 & 0 \\ 0 & 0 & +2 & 0 & +1 & +1 \end{pmatrix} \mathbf{a}^2 = \mathbf{0}. \tag{27}$$

We have five constraints on six unknowns, and (27) may be row-reduced to yield the general solution

$$\mathbf{a}^2 = \upsilon\,(+1, 0, +3, -1, -2, -4)\,, \tag{28}$$

with one degree of freedom υ, which yields

$$\begin{aligned} \upsilon\left\{\left(V_{-1}^2 + V_{+1}^2\right) + 3\,(V_{-1}V_{+2} + V_{-2}V_{+1}) - (V_{-2}V_{-1} + V_{+1}V_{+2})\right. \\ \left. -2V_{-2}V_{+2} - 4V_{-1}V_{+1}\right\}, \end{aligned} \tag{29}$$

for the quadratic level of the filter.

Now we consider the coefficients $\mathbf{a}^3$ of the cubic level of the filter. For each edge position (rather like the calculations at the quadratic level), adding the equations arising from equating the $O(f_-^3)$, $O(f_-^2 f_+)$, $O(f_- f_+^2)$, $O(f_+^3)$ terms gives $\sum \alpha_{ijk} = 0$, which has already been obtained by requiring the perfect fit of a linear profile. Hence at most three of the four equations are linearly independent of those that have been obtained before. For example, for the edge position of Figure 12(a), we need only note the $O(f_-^2 f_+)$, $O(f_- f_+^2)$, and $O(f_+^3)$ equations.

However, for the symmetric edge position of Figure 12(b), the ($O(f_-^3)$ and $O(f_+^3)$) and ($O(f_-^2 f_+)$ and $O(f_- f_+^2)$) equations are identical — hence there is only one genuinely new equation, *e.g.,* that from $O(f_+^2)$, which can be linearly independent of those obtained before.

Altogether we have found 6 homogeneous constraints on the 10-vector $\mathbf{a}^3$ (consisting of 2 from the perfect linear fit, 3 from the asymmetric edge position, and 1 from the symmetric edge position). These may be solved to give the general solution

$$\begin{aligned} \mathbf{a}^3 = \phi(0, 0, -1, 0, 0, 0, 0, 0, 0, 1) + \psi(-1, 0, -3, 2, 1, 0, 0, 0, 1, 0) \\ + \chi(7, 0, 9, -8, -8, -1, 0, 1, 0, 0) + \omega(4, 0, 5, -5, -4, -1, 1, 0, 0, 0), \end{aligned} \tag{30}$$

with 4 degrees of freedom, ϕ, ψ, χ, and ω. The solution (30) can be used to write down the general form of the cubic level of a filter with our sensible properties. Our properties have reduced the 35 degrees of freedom in filter (18) to just five—one at the quadratic level and four at the cubic level. This is a substantial improvement, for the reasons we discussed earlier.

4.2 Example 2

We now consider the filter of Figure 11(b), where the voltage V_0 at a point, *e.g.*, the origin, of the unknown field is calculated by a polynomial function

$$\begin{aligned} V_0 = \alpha_0 &+ \sum \alpha_{(i,j)} V_{(i,j)} \\ &+ \sum \alpha_{(i,j)(k,l)} V_{(i,j)} V_{(k,l)} \\ &+ \sum \alpha_{(i,j)(k,l)(m,n)} V_{(i,j)} V_{(k,l)} V_{(m,n)}, \end{aligned} \tag{31}$$

of the voltages at six surrounding points $(i,j) = (\pm 1, 0)$, $(\pm 1, \pm 1)$ of the known field. (The index (i,j) denotes the point with coordinates $x_1 = ih$ and $x_2 = jh$, where h is the inter-pixel spacing.)

In formula (31), there are 6, 21, and 56 coefficients at the linear, quadratic, and cubic levels respectively. (These numbers take into account the permutational symmetry of indices, *e.g.*, $\alpha_{(i,j)(k,l)}$ and $\alpha_{(k,l)(i,j)}$ are considered identical, as are, *e.g.*, $\alpha_{(i,j)(k,l)(m,n)}$ and $\alpha_{(k,l)(i,j)(m,n)}$ and the four other similar permutations.)

Let us concentrate on the linear and quadratic levels of the filter, and what constraints are implied by assuming symmetry (*i.e.*, that the interpolated value V_0 should be independent of the data being reflected about $x_1 = 0$ or $x_2 = 0$.) At the linear level, such symmetry implies

$$\alpha_{(-1,-1)} = \alpha_{(-1,+1)} = \alpha_{(+1,-1)} = \alpha_{(+1,+1)} \text{ and } \alpha_{(-1,0)} = \alpha_{(+1,0)}, \tag{32}$$

so that there are only 2 degrees of freedom, which may be represented by the vector $\mathbf{a}^1 = (\alpha_{(+1,0)}, \alpha_{(+1,+1)})$. Note (in contrast to the one dimensional filter of Example 1) that we have square symmetry, so that quads (as well as pairs) of coefficients may be forced equal.

At the quadratic level of the filter, it can be shown that symmetry reduces the degrees of freedom to 8, which may be represented by the vector

$$\begin{aligned} \mathbf{a}^2 = (&\alpha_{(+1,0)(+1,0)}, \alpha_{(+1,+1)(+1,+1)}, \alpha_{(-1,0)(+1,0)}, \\ &\alpha_{(+1,0)(+1,+1)}, \alpha_{(-1,0)(+1,+1)}, \alpha_{(-1,+1)(+1,+1)}, \\ &\alpha_{(+1,-1)(+1,+1)}, \alpha_{(-1,-1)(+1,+1)})^T. \end{aligned} \tag{33}$$

Next we consider what constraints on $\mathbf{a}^1$ and $\mathbf{a}^2$ are provided by supposing that a linear voltage profile,

$$V = Ax_1 + Bx_2 + C, \tag{34}$$

is fitted exactly. In a similar fashion to Example 1, we substitute $V_0 = C$ and (34) in the RHS of (31), and equate polynomial terms in A, B, and C.

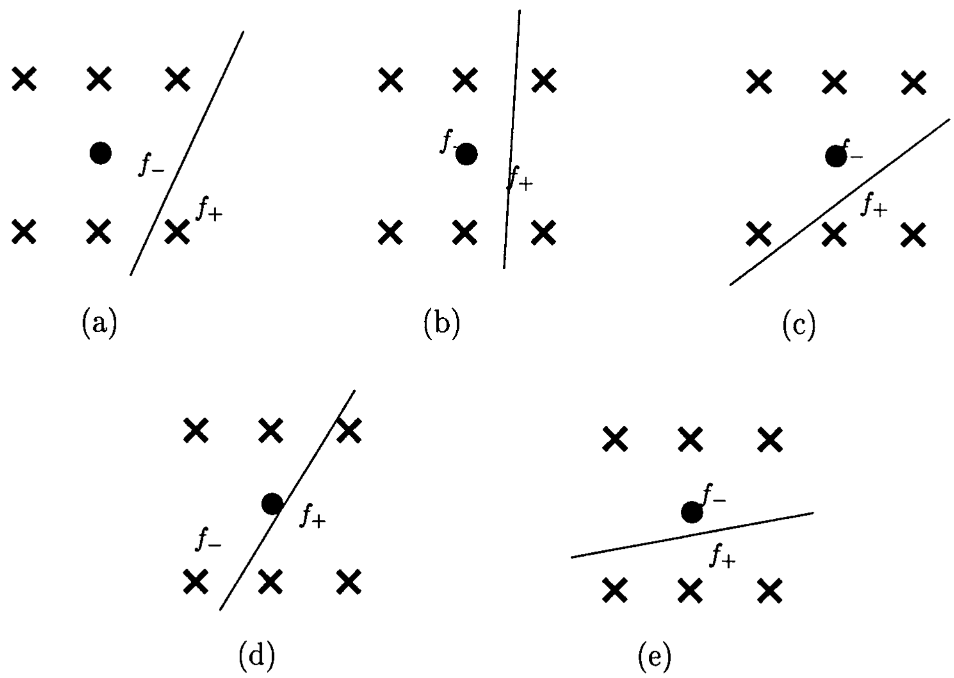

Figure 13: The five possible edge positions (up to symmetry) for the 2D filter of Example 2. '$\times$' denote the known points, and '$\bullet$' denotes the interpolated point.

We immediately obtain $\alpha_0 = 0$ and $4\alpha_{(+1,+1)} + 2\alpha_{(+1,0)} = 1$ (from $O(C)$), together with

$$\sum \alpha_{(i,j)(k,l)} = 0 \qquad \text{(from } O(C^2)), \tag{35}$$

and two other constraints at the quadratic level from $O(A^2)$ and $O(B^2)$. (The $O(AB)$ term vanishes automatically by the symmetry constraints already imposed.)

Next we consider what constraints are imposed on coefficients by supposing that edges are interpolated in an accurate manner — we follow the approach of Example 1, by supposing that the filter interpolates step functions exactly.

Figure 13 depicts the five possible ways (a)–(e) in which the edge may cut through the six point aperture. Position (a) (for which we require $V_0 = f_-$) forces $\alpha_{(+1,+1)} = 0$, and together with the linear fit constraint, this implies $\alpha_{(+1,0)} = 1/2$, so that the linear level of the filter is wholly determined.

Next, consider the effect of the edge constraints on the quadratic level of the filter: each of the positions (a) to (e) yields 3 homogeneous linear constraints on $\mathbf{a}^2$, corresponding to equating $O(f_-^2)$, $O(f_-f_+)$ and $O(f_+^2)$

terms. However, for each edge position, the three constraints add together to give (35), so that at most two, *e.g.,* those from $O(f_+^2)$ and $O(f_- f_+)$, are linearly independent of the linear fit constraints obtained previously. Further, for each of the symmetric edge positions (d) and (e), the $O(f_-^2)$ and $O(f_+^2)$ equations are identical, so that at most one genuinely new equation (*i.e.,* linearly independent of those found before) is obtained. Thus, although one may expect $5 \times 3 = 15$ constraints on $\mathbf{a}^2$ to arise from positions (a)–(e), one obtains only $3 \times 2 + 2 = 8$ constraints.

Altogether, one has 11 homogeneous linear constraints (3 from the perfect linear fit, and 8 from edge constraints) on the 8-vector $\mathbf{a}^2$ of quadratic level coefficients. Unfortunately, these constraints are not degenerate and have rank 8 — *hence the quadratic level of our filter must be identically zero.*

However, if one advances to the cubic level, symmetry constraints reduce the number of degrees of freedom from 56 to 16. The perfect linear fit conditions provide 3 constraints (arising from $O(C^3)$, $O(A^2C)$, and $O(B^2C)$ terms), edge positions (a)–(c) provide 3 constraints each (*e.g.,* arising from $O(f_-^2 f_+)$, $O(f_- f_+^2)$, and $O(f_+^3)$ terms), and symmetric edge positions (d) and (e) provide 1 constraint (*e.g.,* that arising from the $O(f_+^3)$ term) each — in total, we have 14 homogeneous linear constraints on 16 coefficients.

Hence, we can guarantee that there is a nonzero family (with at least 2 degrees of freedom) of cubic terms with our sensible interpolation properties.

5 Conclusion

We have shown how to build Volterra filters with sensible interpolation properties built-in. These properties greatly reduce the number of free parameters and thus it should be easier to see how the filters can be efficiently implemented.

In Example 2, we saw that there were more constraints on the quadratic level of the filter than degrees of freedom, and hence that there were no nonzero terms which have our sensible interpolation properties. As filters become bigger (Snell And Wilcox often consider 20 point apertures), the number of coefficients increases; but, the number of constraints arising from edge interpolation also rises (because the number of different ways of cutting the aperture increases). An interesting question is which number grows fastest, *i.e.,* do we have more and more degrees of freedom as apertures grow, or does it become impossible to find nonzero filters with our sensible properties?

Work is now under way to develop software and counting theorems to find the number of coefficients and constraints for arbitrary, large apertures, using the sorts of techniques which have been outlined in our two examples. PA,CJB,DB,JG,RAG,AMH,JL,MoG,AR,REW,AZ.

References

[1] Walter A. Frank, *Low complexity 3rd order nonlinear filtering,* IEEE Workshop on Nonlinear Signal Processing, Neos Marmaras, Greece,(1995), 384–387.

[2] Walter A. Frank, *An efficient approximation to the quadratic Volterra filter and its application in real time loudspeaker linearization,* Signal Processing, **45**, (1995), 97–113 384–387.

Report submitted by Chris Budd, Jens Gravesen & R. Eddie Wilson

Contributions from: Rebecca Gower, James Lawry, Alan Zinober, Philip Aston, Dana Bedivan, Andrew Hogg.

A Why only uneven terms?

We are looking for a polynomial approximation to an unknown (and probably nonexistent) "perfect interpolation function" $\mathbb{R}^N \to \mathbb{R}$. Assuming some simple symmetry properties simplifies the this task considerably.

Theorem 1. *If $p : \mathbb{R}^N \to \mathbb{R}$ is a polynomial that satisfies $p(-\mathbf{x}) = -p(\mathbf{x})$, then p only has uneven terms.*

Proof. Let

$$p(x_1, \ldots, x_N) = \sum_{k=0}^{n} \prod_{1 \leq i_1 \leq \cdots \leq i_k \leq N} \alpha_{i_1 \ldots i_k} x_{i_1} \cdots x_{i_k},$$

then

$$p(-x_1, \ldots, -x_N) = \sum_{k=0}^{n} (-1)^k \prod_{1 \leq i_1 \leq \cdots \leq i_k \leq N} \alpha_{i_1 \ldots i_k} x_{i_1} \cdots x_{i_k},$$

adding the two equations gives us:

$$0 = 2 \sum_{0 \leq 2l \leq n} \prod_{1 \leq i_1 \leq \cdots \leq i_{2l} \leq N} \alpha_{i_1 \ldots i_{2l}} x_{i_1} \cdots x_{i_{2l}} \quad \text{for all } (x_1, \ldots, x_N),$$

from which it follows that $\alpha_{i_1 \ldots i_k} = 0$ if k is even. □

B Fortran code

The following is a Fortran code to calculate the optimal MMD filter for a 4 pixel aperture. It prompts the user for the filter architecture which are the three numbers NA, NB, NC which must add to six. To run the code you must have access to the NAG library (although any unconstrained optimisation package should do). You will need to have a data file

`cor.dat`

giving the correlation matrix $M^T M$, the vector $M^T \mathbf{R}$ and the scalar $\mathbf{R}^T \mathbf{R}$. You will also need to create the file

`guess.dat`

which contains the initial guess for the coefficients. In practice a file full of zeroes will do. The output is given in the file

`opt.dat`.

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                            c
c  Code to solve the Snell and Wilcox optimisation problem   c
c  =======================================================   c
c                                                            c
c                                                            c
c  Chris Budd, University of Bath,   March 1998              c
c  ==============================================            c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c ============================================================
c
c  Data
c  ====

c  h    : unknowns in the filter coefficients
c  hi   : i = 1,6 filter banks
c  hh   : coefficients in the Volterra filter
c  xtx  : etc. correlation input

c
c ============================================================
c
   implicit double precision(a-h,o-z)


   common/a1/ic(20,3)
   common/a2/h1(10),h2(10),h3(10),h4(10),h5(10),h6(10),hh(34)
   common/a3/xtx(34,34),xty(34),yty
   common/a4/na,nb,nc,num

   dimension bl(20),bu(20),iw(30)
   dimension h(20),w(500)

   open(unit=10,file='opt.dat',status='unknown')
c
c       Read in the initial guess of coefficients
c

   call init(h)
```

```
c
c  Read in the correlation matrix
c
   call xtxset


c
c     Set up pointers
c

    call ipoint

c
c  Optimise the system using the NAG subroutine e04jaf
c  which performs an unconstrained optimisation.
c


   n = num
   ibound = 1
   liw = 30
   lw = 500
   ifail = 1

    call e04jaf(n,ibound,bl,bu,h,f,iw,liw,w,lw,ifail)


c
c  write out the solution
c

   write(10,*) 'MMD Volterra filter'
   write(10,*) '-------------------'
   write(10,*) ' '
   write(10,*) 'Solution architecture ',na,nb,nc
   write(10,*) ' '


   write(10,*) 'Converged values for each filter bank'
   write(10,*) ' '
```

```
      call filter(h)
      write(10,*) 'h1: ',(h1(i),i=1,na)
      write(10,*) 'h2: ',(h2(i),i=1,na)
      write(10,*) 'h3: ',(h3(i),i=1,na+nb-1)
      write(10,*) 'h4: ',(h4(i),i=1,nb)
      write(10,*) 'h5: ',(h5(i),i=1,nc)
      write(10,*) 'h6: ',(h6(i),i=1,4)
      write(10,*) ' '

      call hhset
      write(10,*) 'Coefficients of the Volterra filter'
      write(10,*) '-----------------------------------'
      write(10,*) ' '

      do 110 i=1,34
      write(10,*) i,hh(i)
110continue

      write(10,*) ' '
      write(10,*) 'RMS error of optimal MMD filter is: ',f

      write(6,*) ' '
      write(6,*) 'RMS error of optimal MMD filter is: ',f

      close(10)

      stop
      end

c
c ============================================================
c

     subroutine funct1(n,hc,fc)

c
c Calculates the penalty function of the
c optimisation routine
c

     implicit double precision (a-h,o-z)
```

```
      common/a2/h1(10),h2(10),h3(10),h4(10),h5(10),h6(10),hh(34)
      common/a3/xtx(34,34),xty(34),yty

      dimension hc(20)

c
c  Set up the filters using constraints where necessary
c

        call filter(hc)

c
c     Set up vector hh
c

        call hhset

        sum=yty

        do 20 i=1,34
        sum1=0.
            do 10 j=1,34
            sum1=sum1+xtx(i,j)*hh(j)
 10         continue
        sum = sum + hh(i)*sum1 - 2.*hh(i)*xty(i)
 20     continue

c         write(6,*) sum,hc(1)

        fc=sum

        return

 1000   format(1x,5(f10.5,1x))

        end

c
c ===========================================================
c
```

```
      subroutine init(h)
c
c         Inputs the initial guess
c
      implicit double precision(a-h,o-z)
      common/a4/na,nb,nc,num
      dimension h(20)

      write(6,*) 'Give the filter architecture'
      read(5,*) na,nb,nc

      num=5+2*na+nb

      open(unit=2,file='guess.dat',status='unknown')

      write(6,*) 'Initial guess'
      write(6,*) ' '

       do 10 i=1,num
       read(2,*) h(i)
       write(6,*) h(i)
 10    continue

      write(6,*) ' '
      close(2)
      return
      end

c
c ==========================================================
c

      subroutine xtxset
      implicit double precision(a-h,o-z)
      common/a3/xtx(34,34),xty(34),yty
      dimension xtxn(34,34),xtyn(34),c(34),wk1(34),wk2(34)

      open(unit=3,file='cor.dat',status='unknown')

      n=34
```

```
      ifail=0
      ia=34

      do 20 i=1,34
         do 10 j=1,34
         read(3,*) xtx(i,j)
         xtxn(i,j)=xtx(i,j)
 10     continue
      read(3,*) xty(i)
      xtyn(i)=xty(i)
 20continue


      do 25 i=1,34
      read(3,*) dummy
 25continue

      read(3,*) yty

      close(3)

c
c        Estimate optimal volterra filter
c

c
c        The following is a NAG routine to invert a matrix
c

      call f04asf(xtxn,ia,xtyn,n,c,wk1,wk2,ifail)

      write(10,*) ' '
      write(10,*) ' '
      write(10,*) 'Volterra filter calculation'
      write(10,*) '==========================='
      write(10,*) '                           '
      write(10,*) ' Optimal Volterra filter coefficients'
      write(10,*) ' ------------------------------------'
      write(10,*) '                        '

      do 50 i=1,34
```

```
   write(10,*) i,c(i)
50continue
   write(10,*) '  '

   sum=yty

      do 70 i=1,34
      sum=sum-2.*c(i)*xty(i)
      psum=0.
         do 60 j=1,34
         psum=psum+xtx(i,j)*c(j)
60       continue
      sum=sum+c(i)*psum
70    continue

   write(10,*) 'RMS error of the optimal filter is   ', sum
   write(10,*) ' '
   return
   end

c
c ============================================================
c

   subroutine hhset
c
c     Sets up the hh vector
c

   implicit double precision(a-h,o-z)

   common/a1/ic(20,3)
   common/a2/h1(10),h2(10),h3(10),h4(10),h5(10),h6(10),hh(34)


    do 100 i=1,4
    hh(i) = h6(i)
100continue

    do 110 i=5,14
    hh(i) = 0.
```

```
 110continue

     do 120 i=15,34
     j =  i-14

     ii = ic(j,1)
     jj = ic(j,2)
     kk = ic(j,3)

c
c    Calculate intermediate coefficients
c
    aa = ff(ii,jj,kk)+ff(ii,kk,jj)+ff(jj,ii,kk)+ff(jj,kk,ii)
    hh(i) = (aa+ff(kk,ii,jj)+ff(kk,jj,ii))/6.

    if ((ii.ne.jj).and.(jj.ne.kk).and.(kk.ne.ii)) then
    hh(i) = hh(i)*6.
    endif

    if ((ii.eq.jj).and.(ii.ne.kk)) hh(i) = hh(i)*3.
    if ((ii.eq.kk).and.(ii.ne.jj)) hh(i) = hh(i)*3.
    if ((jj.eq.kk).and.(ii.ne.jj)) hh(i) = hh(i)*3.

 120  continue

    return
    end



c
c ============================================================
c

   subroutine filter(h)

   implicit double precision(a-h,o-z)
   common/a2/h1(10),h2(10),h3(10),h4(10),h5(10),h6(10),hh(34)
   common/a4/na,nb,nc,num

   dimension h(20)
```

```
c
c     Sets up the filter banks and applies the constraints
c

  h1(1) = 1.
  h2(1) = 1.

    if (na.gt.1) then

      do 10 i=1,na-1
      h1(i+1) = h(i)
      h2(i+1) = h(na+i-1)
 10  continue

    endif


    h3(1) = 1.

    if (na+nb-1.gt.1) then

      do 20 i=1,na+nb-2
      h3(i+1) = h(2*na+i-2)
 20  continue

    endif

    h4(1) = 1.

    if (nb.gt.1) then

      do 30 i=1,nb-1
      h4(i+1) = h(nb+3*na-4+i)
 30  continue

    endif

    do 40 i=1,nc
    h5(i) = h(2*nb+3*na-5+i)
 40continue
```

```
      h6(1) = h(num-3)
      h6(2) = h(num-2)
      h6(3) = h(num-1)
      h6(4) = h(num)

      return
      end

c
c ============================================================
c


      subroutine ipoint

      common/a1/ic(20,3)
c
c     Sets up pointers to the data indices
c

      ic(1,1) = 0
      ic(1,2) = 0
      ic(1,3) = 0

      ic(2,1) = 0
      ic(2,2) = 0
      ic(2,3) = 1

      ic(3,1) = 0
      ic(3,2) = 0
      ic(3,3) = 2

      ic(4,1) = 0
      ic(4,2) = 0
      ic(4,3) = 3

      ic(5,1) = 0
      ic(5,2) = 1
```

```
ic(5,3) = 1

ic(6,1) = 0
ic(6,2) = 1
ic(6,3) = 2

ic(7,1) = 0
ic(7,2) = 1
ic(7,3) = 3

ic(8,1) = 0
ic(8,2) = 2
ic(8,3) = 2

ic(9,1) = 0
ic(9,2) = 2
ic(9,3) = 3

ic(10,1) = 0
ic(10,2) = 3
ic(10,3) = 3

ic(11,1) = 1
ic(11,2) = 1
ic(11,3) = 1

ic(12,1) = 1
ic(12,2) = 1
ic(12,3) = 2

ic(13,1) = 1
ic(13,2) = 1
ic(13,3) = 3

ic(14,1) = 1
ic(14,2) = 2
ic(14,3) = 2

ic(15,1) = 1
ic(15,2) = 2
ic(15,3) = 3
```

```
      ic(16,1) = 1
      ic(16,2) = 3
      ic(16,3) = 3

      ic(17,1) = 2
      ic(17,2) = 2
      ic(17,3) = 2

      ic(18,1) = 2
      ic(18,2) = 2
      ic(18,3) = 3

      ic(19,1) = 2
      ic(19,2) = 3
      ic(19,3) = 3

      ic(20,1) = 3
      ic(20,2) = 3
      ic(20,3) = 3


      return
      end

c
c ==========================================================
c

      double precision function ff(ii,jj,kk)

c
c     calculates the intermediate coefficients of the Volterra filter
c

      implicit double precision(a-h,o-z)

      common/a2/h1(10),h2(10),h3(10),h4(10),h5(10),h6(10),hh(34)
      common/a4/na,nb,nc,num

      sum = 0.
```

```
      do 20 ll = 0,nc-1

      sum1=0.

       do 10 mm = 0,nb-1

       j1 = jj-mm-ll
       j2 = kk-mm-ll
       n1 = na-1

       if ((j1.ge.0).and.(j1.le.n1).and.(j2.ge.0).and.(j2.le.n1)) then
       sum1=sum1+h4(mm+1)*h1(j1+1)*h2(j2+1)
       endif

   10 continue

       n3=na+nb-1

       if (((ii-ll).le.n3).and.((ii-ll).ge.0)) then
       sum = sum+sum1*h5(ll+1)*h3(ii-ll+1)
       endif

   20 continue

      ff = sum

      return
      end
```

Appendix three: Output

The following is the file

`opt.dat`

created when optmising the $(3, 1, 2)$ filter using the *girl* test image.

```
Volterra filter calculation
===========================

 Optimal Volterra filter coefficients
 ------------------------------------

        1 -8.9130844719457786E-02
        2  0.5987875705312924
        3  0.5790294448848945
        4 -8.6293237312759341E-02
        5  2.8136854929962395E-04
        6 -7.2505629753418356E-04
        7  1.7481162214815838E-04
        8  4.6859261785271791E-06
        9 -1.3571065960541544E-04
       10  1.6766713332701235E-03
       11 -7.2758968850160171E-04
       12 -9.9065351052699977E-04
       13  1.9106195475938193E-04
       14  2.4146613505981082E-04
       15 -3.4652998087846348E-06
       16 -2.6204276920334582E-05
       17  2.6557574758950933E-05
       18 -5.3441497961841746E-06
       19  4.0155916617402962E-05
       20  2.0278951865025677E-06
       21 -1.8192339248831110E-06
       22 -3.0208140141996826E-05
       23  1.7460292172708630E-05
       24 -9.2670139148104249E-06
       25 -1.5527872018860456E-06
       26 -1.2672580315562128E-05
       27 -1.6568897991731025E-05
       28 -1.5843718750425863E-05
       29 -4.2878359786767087E-06
```

```
      30  2.9108348569319611E-05
      31  1.1268675383362890E-05
      32  2.6080079410444757E-05
      33 -2.0873545454280818E-05
      34 -4.9643830327691209E-06

RMS error of the optimal filter is     16.01913180067564

MMD Volterra filter
-------------------

Solution architecture            3            1            2

Converged values for each filter bank

h1:  1.000000000000000   80.54113650908003   -58.79791819006203
h2:  1.000000000000000   -1.290744618049413   -7.452512436174E-02
h3:  1.000000000000000   -0.1313062155505152  -0.9131080341948733
h4:  1.000000000000000
h5:-3.5162819360725E-07   3.6224119601860692E-07
h6:-0.1034556961938496    0.6125895835435735      0.5851376076171718
   -9.5231028171078E-02

Coefficients of the Volterra filter
-----------------------------------

       1 -0.1034556961938496
       2  0.6125895835435735
       3  0.5851376076171718
       4 -9.5231028171078841E-02
       5  0.0000000000000000E+00
       6  0.0000000000000000E+00
       7  0.0000000000000000E+00
       8  0.0000000000000000E+00
       9  0.0000000000000000E+00
      10  0.0000000000000000E+00
      11  0.0000000000000000E+00
      12  0.0000000000000000E+00
      13  0.0000000000000000E+00
      14  0.0000000000000000E+00
      15 -3.5162819360725633E-07
```

```
16 -2.7820501175926814E-05
17  2.1022285424528633E-05
18  0.0000000000000000E+00
19  4.0213644541039521E-05
20 -1.8484765101127162E-06
21  0.0000000000000000E+00
22 -2.0443249362126583E-05
23  0.0000000000000000E+00
24  0.0000000000000000E+00
25 -4.4376020079184224E-06
26 -1.4911620605454200E-06
27 -2.1656789625155458E-05
28 -1.8784931246337220E-05
29  1.9042680706753014E-06
30  2.1060276832393731E-05
31  6.3516378530186342E-06
32  3.1061396201482659E-05
33 -2.3325864887009759E-05
34 -1.4493880029159903E-06

RMS error of optimal MMD filter is:    16.49908224025793
```