Boundary Conditions for Young–van Vliet Recursive Filtering

Bill Triggs, *Member, IEEE,* and Michaël Sdika

***Abstract*—Young and van Vliet have designed computationally efficient methods for approximating Gaussian-based convolutions by running a recursive infinite-impulse-response (IIR) filter forward over the input signal, then running a second IIR filter backward over the first filter's output. To transition between the two filters, they use a suboptimal heuristic that produces significant amplitude and phase distortion for all points within about three standard deviations of the right-hand boundary. In this correspondence, a simple linear transition rule that eliminates this distortion is derived.**

***Index Terms*—Bidirectional recursive filtering, boundary conditions, Gaussian smoothing.**

I. Introduction

Young and van Vliet (YvV) have developed computationally efficient forward–backward infinite-impulse-response (IIR) recursions for Gaussian filters [1], Gaussian derivatives [2], and Gabor filters [3]. (See [3] for their most recent design rules for Gaussians, and [4] for space-variant extensions and a performance comparison with other IIR Gaussian methods, including Deriche's original method [5], [6].) Our approach also applies to the analogous recursion [7], [8] for B-spline-based signal processing. All of the YvV filters work forward, recursively calculating a running sum u_t as a linear combination of the input signal i_t and the k previous u values, then work backwards calculating a running sum v_t as a linear combination of u_t and the l previously calculated v values:

$$u_t = i_t + \sum_{j=1}^{k} a_j u_{t-j}, \qquad t = 1, \ldots, n \tag{1}$$

$$v_t = u_t + \sum_{j=1}^{l} b_j v_{t+j}, \qquad t = n, \ldots 1. \tag{2}$$

The final output is a scaled version of v_t, and $\{a_{j,j=1\ldots k}\}$ and $\{b_{j,j=1\ldots l}\}$ are suitably chosen filter coefficients. For Gaussians, YvV chose $k = l$ and $\boldsymbol{a} = \boldsymbol{b}$ [1], [3]. For other filters, i_t may be a linear transformation of the original input signal, e.g., a discrete derivative for derivative filters [1].

II. Problem With Heuristic Boundary Conditions

To complete the specification (1), (2), we must fix initial conditions for u near $t = 1$ and for v near $t = n$. For u, we can pretend that the signal existed and took some nominal constant value i_- (typically either 0 or i_1) for all $t < 1$. The correct initialization at $t = 1$ is then to set all $u_{1-j,j=1\ldots k}$ to $i_-/(1 - \sum_{j=1}^{k} a_j)$, the steady-state response to an infinite stream of i_-'s. Similarly, if we could suppose that for all $t > n$, u_t took some constant value u_+, the correct condition at $t = n$ would be to set $v_{n+j,j=1\ldots l}$ to $u_+/(1-\sum_{j=1}^{l} b_j)$, the steady-state response to an infinite stream of u_+'s. YvV apparently do exactly this, with $i_- = i_1$ and $u_+ = u_n$ (cf. [3, eq. (20) and (21)]). Another plausible choice for u_+ would be $i_+/(1 - \sum_{j=1}^{k} a_j)$, the steady state u resulting from an infinite stream of constant input values i_+ above $t = n$ (typically, i_+ would be either i_n or 0).

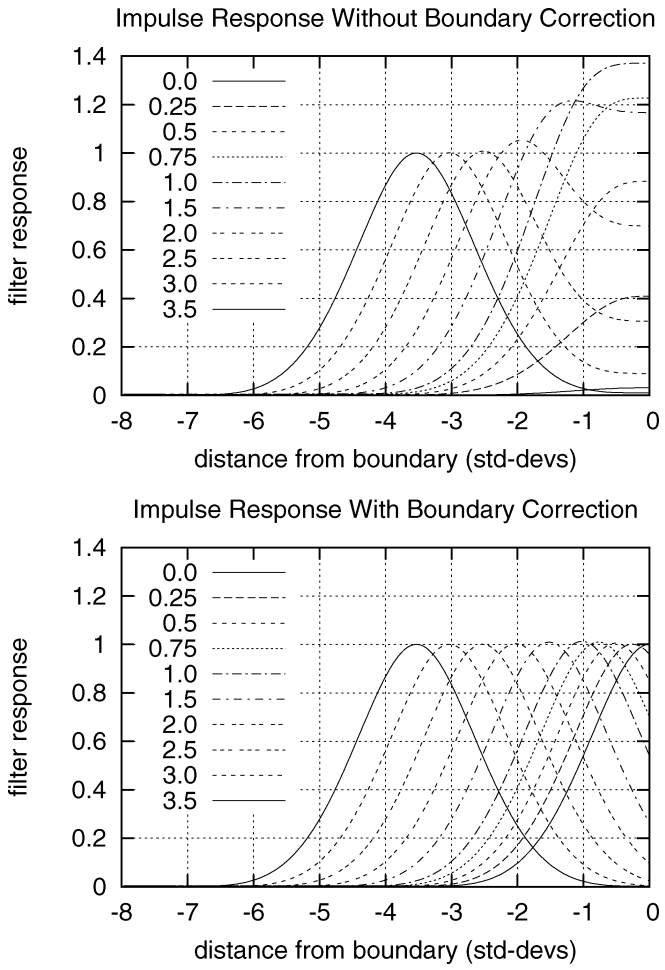

Fig. 1. Impulse responses for points at various numbers of standard deviations from the right boundary, for a YvV Gaussian filter, with the standard YvV boundary heuristic $u_+ = u_n$ (top) and with our new boundary correction (14) (bottom). The corrected responses are much closer to the desired (truncated Gaussian) form.

Unfortunately, neither choice for u_+ is correct. If the forward filter were continued to $t \gg n$ with input i_+, its output would decay smoothly from u_n to $i_+/(1 - \sum_{j=1}^{k} a_j)$ within a few standard deviations, and the corresponding backward filter would take all elements of this "advance warning" signal into account when calculating its response. In fact, the forward–backward process *only* gives the correct overall impulse response when the full double recursion is run without truncation. Incorrect truncation causes significant amplitude and phase (geometric position) distortion for all points within about three standard deviations of the boundary. Fig. 1 illustrates the extent of the problem.

III. Derivation of Linear Boundary Correction

To correct for the effects of truncation, we notionally extend the forward–backward pass to $t \to \infty$ assuming a constant input value i_+ above $t = n$ and calculate the coefficients $\{v_{n+j,j=1\ldots l}\}$ that would result from this infinite extension, given i_+ and the final forward filter state $\{u_{n-j,j=0\ldots k-1}\}$. The whole process is linear so the v's must be linear functions of the u's and i_+. First suppose that $i_+ = 0$. Gathering

Manuscript received July 19, 2004; revised June 13, 2005. This work was partly supported by European Union research projects LAVA and aceMedia. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Trac D. Tran.

The authors are with GRAVIR-CNRS-INRIA, 38330 Montbonnot, France (e-mail: Bill.Triggs@inrialpes.fr; Michael.Sdika@inrialpes.fr; website: http://lear.inrialpes.fr/people/triggs).

Digital Object Identifier 10.1109/TSP.2006.871980

the u's, v's into running k, l vectors $\boldsymbol{u}$, $\boldsymbol{v}$, the forward and backward passes become

$$\boldsymbol{u}_t = \boldsymbol{A}\boldsymbol{u}_{t-1} = \boldsymbol{A}^{t-n}\boldsymbol{u}_n, \qquad t > n, i_+ \equiv 0 \tag{3}$$

$$\boldsymbol{v}_t = \boldsymbol{I}_1\boldsymbol{u}_t + \boldsymbol{B}\boldsymbol{v}_{t+1}, \qquad t \geq n \tag{4}$$

where $\boldsymbol{A}^k = \boldsymbol{A}\cdot\boldsymbol{A}\cdot\ldots\cdot\boldsymbol{A}$ (k terms) is the k^{th} power of $\boldsymbol{A}$ and see (5), shown at the bottom of the page

$$\boldsymbol{u}_t \equiv \begin{pmatrix} u_t \\ \vdots \\ u_{t-k+1} \end{pmatrix} \quad \boldsymbol{A} \equiv \begin{pmatrix} a_1 & \cdots & a_{k-1} & a_k \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \\ 0 & \cdots & 1 & 0 \end{pmatrix} \tag{6}$$

$$\boldsymbol{v}_t \equiv \begin{pmatrix} v_t \\ \vdots \\ v_{t+l-1} \end{pmatrix} \quad \boldsymbol{B} \equiv \begin{pmatrix} b_1 & \cdots & b_{l-1} & b_l \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \\ 0 & \cdots & 1 & 0 \end{pmatrix} \tag{7}$$

and $\boldsymbol{I}_1 = (1\ 0\ \ldots\ 0)^\top(1\ 0\ \ldots\ 0)$ is an $l \times k$ matrix with a 1 in the top-left corner and zeros elsewhere. Combining these equations for all $t \geq n$, we have $\boldsymbol{v}_n = (\sum_{i=0}^{\infty}\boldsymbol{B}^i\boldsymbol{I}_1\boldsymbol{A}^i)\boldsymbol{u}_n$. We need to calculate the $l \times k$ matrix $\boldsymbol{M} \equiv \sum_{i=0}^{\infty}\boldsymbol{B}^i\boldsymbol{I}_1\boldsymbol{A}^i$ that links the initial backward state $\boldsymbol{v}_n$ to the final forward one $\boldsymbol{u}_n$. By $\boldsymbol{M}$'s recursive definition

$$\boldsymbol{M} = \boldsymbol{I}_1 + \boldsymbol{B}\boldsymbol{M}\boldsymbol{A}. \tag{8}$$

Writing $\boldsymbol{M} = \begin{pmatrix} \boldsymbol{m}^1 \\ \vdots \\ \boldsymbol{m}^l \end{pmatrix}$ by rows as a kl-element row vector $\overleftrightarrow{\boldsymbol{M}} = (\boldsymbol{m}^1, \ldots, \boldsymbol{m}^l)$ and, similarly for $\boldsymbol{I}_1$, converts (8) to

$$\overleftrightarrow{\boldsymbol{M}} = \overleftrightarrow{\boldsymbol{I}}_1 + \overleftrightarrow{\boldsymbol{M}}\begin{pmatrix} \boldsymbol{b}_1\boldsymbol{A} & \boldsymbol{A} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{b}_{l-1}\boldsymbol{A} & \boldsymbol{0} & \cdots & \boldsymbol{A} \\ \boldsymbol{b}_l\boldsymbol{A} & \boldsymbol{0} & \cdots & \boldsymbol{0} \end{pmatrix}. \tag{9}$$

This sparse system is easily solved to give ($\boldsymbol{e}^1 = (1, 0, \ldots, 0)$):

$$\boldsymbol{m}^1 = \boldsymbol{e}^1\left(\boldsymbol{I} - \sum_{j=1}^{l}\boldsymbol{b}_j\boldsymbol{A}^j\right)^{-1}, \ \boldsymbol{m}^i = \boldsymbol{m}^1\boldsymbol{A}^{i-1}, \qquad i = 2, \ldots, l \tag{10}$$

Alternatively, if $l \ll k$, it may be preferable to write $\boldsymbol{M} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_k)$ by columns as a kl-element column vector $\boldsymbol{M}^\updownarrow$, so that (8) becomes

$$\boldsymbol{M}^\updownarrow = \boldsymbol{I}_1^\updownarrow + \begin{pmatrix} \boldsymbol{a}_1\boldsymbol{B} & \boldsymbol{B} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{a}_{k-1}\boldsymbol{B} & \boldsymbol{0} & \cdots & \boldsymbol{B} \\ \boldsymbol{a}_k\boldsymbol{B} & \boldsymbol{0} & \cdots & \boldsymbol{0} \end{pmatrix}\boldsymbol{M}^\updownarrow \tag{11}$$

with solution ($\boldsymbol{e}_1 \equiv (1\ 0\ \ldots\ 0)^\top$):

$$\boldsymbol{m}_1 = \left(\boldsymbol{I} - \sum_{j=1}^{k}\boldsymbol{a}_j\boldsymbol{B}^j\right)^{-1}\boldsymbol{e}_1 \tag{12}$$

$$\boldsymbol{m}_i = \left(\sum_{j=i}^{k}\boldsymbol{a}_j\boldsymbol{B}^{j-i+1}\right)\boldsymbol{m}_1, \qquad i = 2, \ldots, k. \tag{13}$$

As an example, the $\boldsymbol{M}$ of the ($k = l = 3$, $\boldsymbol{a} = \boldsymbol{b}$) Gaussian filter recommended by YvV is given in (5), shown at the bottom of the page.

Finally, to handle nonzero i_+, we can simply reduce to the $i_+ = 0$ case by subtracting the constant-u response $u_+ = i_+/(1 - \sum_{j=1}^{k} a_j)$ from each component of $\boldsymbol{u}_n$, apply $\boldsymbol{M}$, then add back the corresponding constant-v response $u_+/(1 - \sum_{j=1}^{l} b_j)$ to get $\boldsymbol{v}_n$.

IV. Summary of Method

In summary, Young and van Vliet recursive filters suffer from severe amplitude and phase distortion at the right boundary unless the backward running coefficients are initialized from the forward ones as follows, where $\boldsymbol{M}$ is given by (5), (8), (10) or (12) and (13):

$$\begin{pmatrix} v_n \\ \vdots \\ v_{n+l-1} \end{pmatrix} = \boldsymbol{M}\begin{pmatrix} u_n - u_+ \\ \vdots \\ u_{n-k} - u_+ \end{pmatrix} + \begin{pmatrix} v_+ \\ \vdots \\ v_+ \end{pmatrix} \tag{14}$$

$$u_+ = \frac{i_+}{1 - \sum_{j=1}^{k} a_j} \quad v_+ = \frac{u_+}{1 - \sum_{j=1}^{l} b_j}. \tag{15}$$

An implementation for two-dimensional Gaussian image filtering is available on the author's website. J.-M. Geusebroek has also incorporated the technique in his IIR filtering package, available from his website http://www.science.uva.nl/~mark/downloads.html

Acknowledgment

The authors would like to thank J.-M. Geusebroek for pointing out an error in the submitted version of (14).

$$\boldsymbol{M} = \frac{1}{(1 + a_1 - a_2 + a_3)(1 - a_1 - a_2 - a_3)\cdot(1 + a_2 + (a_1 - a_3)a_3)} \times \begin{pmatrix} -a_3a_1 + 1 - a_3^2 - a_2 & (a_3 + a_1)(a_2 + a_3a_1) & a_3(a_1 + a_3a_2) \\ a_1 + a_3a_2 & -(a_2 - 1)(a_2 + a_3a_1) & -(a_3a_1 + a_3^2 + a_2 - 1)a_3 \\ a_3a_1 + a_2 + a_1^2 - a_2^2 & a_1a_2 + a_3a_2^2 - a_1a_3^2 - a_3^3 - a_3a_2 + a_3 & a_3(a_1 + a_3a_2) \end{pmatrix} \tag{5}$$

REFERENCES

[1] I. Young and L. van Vliet, "Recursive implementation of the Gaussian filter," *Signal Process.*, vol. 44, pp. 139–151, 1995.

[2] L. van Vliet, I. Young, and P. Verbeek, "Recursive Gaussian derivative filters," in *Proc. Int. Conf. Pattern Recognition*, Brisbane, 1998, pp. 509–514.

[3] I. Young, L. van Vliet, and M. van Ginkel, "Recursive Gabor filtering," *IEEE Trans. Signal Process.*, vol. 50, no. 11, pp. 2799–2805, Nov. 2002.

[4] S. Tan, J. Dale, and A. Johnston, "Performance of three recursive algorithms for fast space-variant Gaussian filtering," *Real-Time Imag.*, vol. 9, pp. 215–228, 2003.

[5] R. Deriche, "Recursively implementing the Gaussian and its derivatives," in *Int. Conf. Image Processing*, Singapore, Sep. 1992, pp. 263–267.

[6] ——, "Recursively implementing the Gaussian and its derivatives," INRIA Sophia-Antipolis, Montbonnot, France, Research Rep. 1893, 1993.

[7] M. Unser, A. Aldroubi, and M. Eden, "B-spline signal processing: Part I—Theory," *IEEE Trans. Signal Process.*, vol. 41, no. 2, pp. 821–833, Feb. 1993.

[8] M. Unser, A. Aldroubi, and M. Eden, "B-spline signal processing: Part II—Efficient design and applications," *IEEE Trans. Signal Process.*, vol. 41, no. 2, pp. 834–848, Feb. 1993.