Mathematical Approaches to Linear Vector Filtering of Color Images

S. J. Sangwine
Department of Electronic Systems Engineering
University of Essex, Colchester, United Kingdom

T. A. Ell
Savage, Minnesota

Abstract

Very few linear vector filters are known for color images and this may be due to the lack of a mathematical framework. This paper explores some possible mathematical approaches to linear vector filtering, including hypercomplex numbers, and dot and cross products. It also considers the fundamental pixel level operations that must exist to make linear vector filtering possible.

Introduction

Linear filtering is well-established in greyscale image processing, and of course, in digital signal processing generally. In color image processing, however, there have been very few developments of *linear* filters, excluding those that operate separately on the color image components *as if they were greyscale images*. It is believed that the first linear vector color filter is that published by Sangwine in Ref. 1 in which convolution with quaternion (hypercomplex) masks was used to define a novel color edge detector based on rotation of pixel values about the 'gray line' of RGB space. In effect, the filter was a vector version of the classic Prewitt edge detector. Later, in Refs. 2 and 3, Evans, Sangwine and Ell published two more filters based on convolution with quaternion-valued masks. These papers showed the need for some systematic understanding of the fundamental pixel-level operations from which a linear vector filter may be composed, and for an algebraic approach to the design of linear filters.

The situation is very different with non-linear filters, where there are many known non-linear vector filters,[4] including generalizations of order-statistics filters to vector pixels (the classic example being the vector median filter). In this paper, we explore possible mathematical approaches to linear vector filtering, and we list some of the possible fundamental pixel-level operations that might be used to compose linear vector filters.

In a vector colour image filter, the pixel values within an image are considered as vector quantities, and vector operations are applied to the pixels. In the vector median filter, for example, the *norm* of the vector difference between two pixel values is used (a scalar quantity which measures some 'length' or magnitude of a vector) to rank the pixels within a window according to centrality. We also consider whether it is worthwhile to try to develop linear vector filters, and whether there are new types of filter that may exist, waiting to be discovered.

Linearity

A linear filter obeys the principle of superposition. In the context of image processing, this may be stated as follows. Given two images, x_1 and x_2 which are processed by a linear filter f to yield output images y_1 and y_2:

$$y_1 = f(x_1) \text{ and } y_2 = f(x_2)$$

then an input image obtained by adding x_1 and x_2 pixel by pixel, when processed by f will yield an output image identical to the pixel by pixel sum of y_1 and y_2:

$$y(m, n) = y_1(m, n) + y_2(m, n),\ \forall\ (m, n)$$

and

$$x(m, n) = x_1(m, n) + x_2(m, n),\ \forall\ (m, n)$$

then we must have $y = f(x)$. A corollary is that an arbitrary input image may be decomposed and the components processed by f to yield the components of the output image. When these components are added pixel by pixel, the result will be identical to that obtained if f had been applied to the original image without decomposition.

A second corollary is that the input image may be scaled by a constant (that is all pixels are multiplied by the constant) and the result of filtering with f will be a scaled version of the output image: $f(kx) = kf(x)$, where k is constant. Usually, k is a real constant, but in the case of vector filters, there may be other possibilities.

Linear Spaces

The concept of a 'linear space' is also relevant to the issue of linear vector filters. In a linear space the following conditions will hold for any vectors x and constants k where $\cdot$ denotes multiplication:

$$x_1 + x_2 = x_2 + x_1$$
$$x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3$$
$$x + 0 = x$$
$$x + (-x) = 0$$
$$(k_1 \cdot k_2) \cdot x = k_1 \cdot (k_2 \cdot x)$$
$$1 \cdot x = x$$
$$0 \cdot x = 0$$
$$k \cdot (x_1 + x_2) = k \cdot x_1 + k \cdot x_2$$
$$(k_1 + k_2) \cdot x = k_1 \cdot x + k_2 \cdot x$$

These equations are not logically independent. Note that commutative multiplication is not a requirement. The significance of this will be discussed later in the paper. Of course, there are practical ways in which a filter may depart from linearity, the most significant of which is due to the finite range of the pixel values. This is well known in digital signal processing, where sample values must remain within a finite range, but it does not invalidate the theory of linear filtering – it simply means that implementations of linear filters must be carefully designed to avoid sample or pixel overflow, for example, by computing intermediate arithmetic results with a greater number of bits. In the context of colour image processing, we must be aware that addition of pixel values (vectors) may easily yield vectors outside the finite range of the colour space. This is no different to the problem of sample overflow in digital filtering (*e.g.* of audio) but it is sometimes not clearly understood by those without a signal processing background working in image processing.

If a vector filter is linear, then it should have all the usual characteristics of linear filters, such as an impulse response (the output of the filter when applied to an 'impulse' image containing a single non-zero pixel). There are some problems with this idea, as evidenced by one of the few particular examples available so far of linear vector filters. The filter published in Ref. 1 depended on convolution with two quaternion masks, so that the filter 'coefficients' are in fact pairs of quaternions multiplied on the left and on the right of the pixels. (Quaternion multiplication is noncommutative, and a rotation, as used in this filter, requires left and right values, which are in fact hypercomplex conjugates.) There is therefore no simple definition of the impulse response of this filter, since neither quaternion mask is sufficient on its own to characterize the filter, and the two masks cannot be combined without loss of information about the filter. It appears therefore that the notion of 'impulse response' has to be extended in some way for linear vector filters in the general case. It is not yet known what this extension to the concept will be.

Fundamental Pixel-Level Operations

In greyscale filtering (and classical digital signal processing), the fundamental operations that may be applied to pixels (signal samples) are: scaling by a constant; addition; and spatial shift (time delay in the case of digital filters processing signals which vary over time, spatial shift in the case of images). These three operations are sufficient to allow implementation of any linear filter. Clearly, these operations may be applied to colour image pixels, and the resulting filters will be linear, but they will not be *vector* filters because these three fundamental operations cannot exploit the vector nature of the pixel values, since the results of scaling, adding, and shifting, do not permit any interaction between the vector components of the pixel. More precisely, the outputs of these operations *in any combination* in any one of the three components of a colour pixel do not depend on the inputs in the other two components. Thus the set of fundamental pixel-level operations required to define linear greyscale filters, are not sufficient for linear vector filters.

So, a significant question addressed in what follows is this: what additional fundamental operations are required to implement linear vector filters? Given some candidate operations, what constitutes a minimum set of additional operations (some operations may be defined in terms of others and may not therefore be fundamental)? Possible candidates are:

- the scalar (dot) product between two vectors, also called the *inner product*,
- the outer product between two vectors,
- the vector (cross) product between two vectors,
- projection of a vector onto a direction,
- reflection,
- rotation.

The scalar product yields a scalar, that is, the result of the scalar product applied to two pixels would not be a (vector) pixel but a scalar quantity. It is therefore not very useful for defining a vector filter, although it does yield a result dependent on the angle between the two vectors and it can be used in implementing projection.

The outer product is nothing more than a matrix product [Ref. 5, p3] and it therefore yields a matrix (3 × 3 in the case of 3-space vectors). It does not appear to be useful as a linear filtering operation, unless some further operation is applied to the result to yield a vector, and it is not clear to the authors that there is any advantage in doing this.

The vector (or cross) product yields a vector result perpendicular to the plane of the two vectors in question, and therefore is potentially useful as a fundamental vector operation. It can be defined in terms of unit vectors as a determinant, and it is therefore easy to implement numerically.[6] Given two vectors $u = a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k}$ and $v = d\boldsymbol{i} + e\boldsymbol{j} + f\boldsymbol{k}$ the vector product is given by:

$$\mathbf{u} \times \mathbf{v} = (bf-ce)\mathbf{i} + (cd-af)\mathrm{j} + (ae-bd)\mathbf{k} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a & b & c \\ d & e & f \end{vmatrix}$$

The cross product is not commutative, but as was seen above, this is not a problem.

Rotation is not a fundamental operation because any rotation can be composed as the sum of two reflections [Ref. 7, p41], although it has been used directly in a published linear vector filter already cited.[1]

The quaternion algebra provides a concise algebraic representation for all of the above fundamental operations apart from the outer product: the scalar (dot) product, the vector (cross) product, and projection, rotation and reflection. The basic definitions and properties of quaternions are reviewed briefly in the Appendix. Here we show how each of the fundamental operations is expressed algebraically, without proof (the proofs are not difficult).

The first two fundamental operations are based on the product of two pure quaternions (vectors). The full quaternion product is defined in the Appendix, and the case of two vectors is a special case of the full quaternion product. If we denote the two vectors by $\mathbf{u}$ and $\mathbf{v}$, then their product $\mathbf{uv}$ is $-\mathbf{u}\cdot\mathbf{v}+\mathbf{u}\times\mathbf{v}$, which is a full quaternion with scalar part $-\mathbf{u}\cdot\mathbf{v}$ and vector part $\mathbf{u}\times\mathbf{v}$. (The dot indicates the scalar product.) Changing the order of the product to $\mathbf{vu}$ does not affect the scalar part, but it negates the vector part (the cross product reverses direction when its operands are interchanged). Thus we have:

$$\mathbf{u}\cdot\mathbf{v} = -\tfrac{1}{2}(\mathbf{uv}+\mathbf{vu}) \text{ and } \mathbf{u}\times\mathbf{v} = \tfrac{1}{2}(\mathbf{uv}-\mathbf{vu})$$

Projection can be expressed algebraically in terms of a dot product and multiplication with a unit vector. From this, we can derive a direct algebraic expression for projection. Assume that we have a *unit* vector $\mathbf{u}$, and an arbitrary vector $\mathbf{v}$, and that we want the projection of $\mathbf{v}$ onto $\mathbf{u}$. In conventional mathematical notation, this can be expressed as $(\mathbf{u}\cdot\mathbf{v})\mathbf{u}$, since the dot product is a scalar quantity dependent only on the magnitude of $\mathbf{v}$ (the magnitude of $\mathbf{u}$ is unity), and multiplying the unit vector $\mathbf{u}$ by this magnitude achieves our aim. Using our expression above for the dot product we have:

$$(\mathbf{u}\cdot\mathbf{v})\mathbf{u} = -\tfrac{1}{2}(\mathbf{uv}+\mathbf{vu})\mathbf{u}$$

which simplifies to

$$(\mathbf{u}\cdot\mathbf{v})\mathbf{u} = -\tfrac{1}{2}(\mathbf{uvu}+\mathbf{vu}^2)$$

The square of any unit pure quaternion (vector) is –1 (this follows from the dot product expression above) and hence we obtain for projection:

$$(\mathbf{u}\cdot\mathbf{v})\mathbf{u} = \tfrac{1}{2}(\mathbf{v}-\mathbf{uvu})$$

As an aside, this may also be derived in a different way, by recognising the term $\mathbf{uvu}$ as a reflection (below).

Reflection of an arbitrary vector $\mathbf{v}$ in a plane normal to a unit vector $\mathbf{u}$ is expressed algebraically as $\mathbf{uvu}$.[8]

Rotation of an arbitrary vector $\mathbf{v}$ about a direction represented by a unit vector $\mathbf{u}$ through an angle θ is expressed algebraically as $\exp(\tfrac{1}{2}\theta\mathbf{u})\mathbf{v}\exp(-\tfrac{1}{2}\theta\mathbf{u})$.[8] The two exponentials are full quaternions, and they are also conjugates.

It should be readily apparent from the above that the quaternion product is sufficient to implement all of the operations proposed as fundamental pixel level vector operations and that no other product is needed. In contrast, an approach based on dot and cross products requires more than one product operation. It follows from the sufficiency of the quaternion product that the scalar (dot) and vector (cross) products are also a sufficient set.

Implementation

There are at least two possible approaches to implementing the operations suggested in the previous section.

Rotations and reflections may be implemented as matrix products between a rotation/reflection matrix (3×3) and a column vector (3×1) holding the pixel value. The dot and cross products between two vectors require a different implementation, but they are both straightforward to implement in software, given the two vector pixels as inputs. Projection may be directly coded in terms of the dot product scaling a unit vector.

An alternative which has been used by two groups[9-15] is to use hypercomplex numbers or quaternions, and to implement the operations that were expressed algebraically in the previous section directly in quaternion arithmetic. This requires a library of code to be written since no standard programming language to date has a built-in library for quaternion arithmetic, but the complexity of this library is not great. This approach has the advantage that the implementation is closely linked to the algrebraic derivation of any filter that is developed.

Transforms

Linear greyscale filters can be implemented in the spatial (image) domain, usually by convolution with the impulse response of the filter (usually called a *mask* in image processing). This is because finite impulse response filters are the norm in image processing. However, any linear filter may also be implemented in the Fourier domain, with the advantage of speed over direct implementation in the spatial domain when the mask size is greater than some threshold level. The implementation in the spatial frequency domain is a point operation, and is therefore fast compared to the convolution required in the image domain, but the gain in speed is obtained at a cost of computing the Fourier transform of the image and an inverse Fourier transform to obtain the filtered image. For masks beyond a certain size, there is a net gain in time.

Fourier transforms of color images have already been demonstrated,[11-15] and since the Fourier transform is a linear operation, it should be possible to implement a linear vector filter in the same way (in the spatial frequency domain using a vector Fourier transform). The general case of a quaternion convolution requires left and right quaternion masks and to date no direct method has been published for implementing such a convolution in the frequency domain. Pei *et al.*[11] have described mathematically a method for implementing such convolutions as *four* separate single-sided convolutions that may be implemented in the frequency domain, but they have not demonstrated any results with an actual filter.

Several authors have reported results with hypercomplex Fourier transforms and methods have been published for implementing correlation using hypercomplex Fourier transforms.[16,17]

Conclusions

Linear vector filtering of colour images is a subject still in its infancy, and there is as yet no certainty that it will develop into a significant field. This paper has shown some of the concepts and has highlighted some of the questions that are still in need of answers.

Acknowledgement

The research presented in this paper is partly funded by the UK Engineering and Physical Sciences Research Council under grant number GR/M 45764.

Appendix: Quaternions

The quaternions were discovered by Hamilton in 1843. They combine by the normal rules of algebra with the exception that multiplication is not commutative. A quaternion has four components, one real and three imaginary. The usual notation, extended from that of the complex numbers is $q = w+x\mathbf{i}+y\mathbf{j}+z\mathbf{k}$ where w, x, y and z are real, and $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$ are complex operators which obey the following rules:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

$$\begin{array}{ccc} \mathbf{ij} = \mathbf{k} & \mathbf{jk} = \mathbf{i} & \mathbf{ki} = \mathbf{j} \\ \mathbf{ji} = -\mathbf{k} & \mathbf{kj} = -\mathbf{i} & \mathbf{ik} = -\mathbf{j} \end{array}$$

The conjugate of q is $w - x\mathbf{i} - y\mathbf{j} - z\mathbf{k}$ and its modulus is $\sqrt{(w^2+x^2+y^2+z^2)}$. A quaternion with zero real part is called a *pure* quaternion, and a quaternion with unit modulus is called a *unit* quaternion. The imaginary part of a quaternion has three components and may be associated with a 3-space vector. For this reason, it is sometimes useful to consider the quaternion as composed of a vector part and a scalar part, thus: $q = S(q) + \mathbf{V}(q)$, where the scalar part, $S(q)$, is the real part (w in our notation above), and the vector part is a composite of the three imaginary components, $\mathbf{V}(q) = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$.

Given two quaternions $q_1 = w_1+x_1\mathbf{i}+y_1\mathbf{j}+z_1\mathbf{k}$ and similarly for q_2, their product expressed in Cartesian terms (in 4 dimensions of course) is:

$$\begin{aligned} q_1q_2 = \; & (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2) \\ & + (w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2)\mathbf{i} \\ & + (w_1y_2 - x_1z_2 + y_1w_2 + z_1x_2)\mathbf{j} \\ & + (w_1z_2 + x_1y_2 - y_1x_2 + z_1w_2)\mathbf{k} \end{aligned}$$

It is more useful to express this in terms of the scalar and vector parts:

$$\begin{aligned} q_1q_2 = \; & (S(q_1)S(q_2) - \mathbf{V}(q_1) \cdot \mathbf{V}(q_2)) \\ & + (S(q_1)\mathbf{V}(q_2) + S(q_2)\mathbf{V}(q_1) + \mathbf{V}(q_1) \times \mathbf{V}(q_2)) \end{aligned}$$

We can see therefore that the scalar part of the product consists of the sum of the product of the scalar parts, minus the dot product of the vector parts; and that the vector part of the product consists of each of the vector parts scaled by the other scalar part, plus the cross product of the two vector parts. Any quaternion may be represented in polar form, thus: $q = |q|\exp(\mu\theta)$ where μ is a unit pure quaternion, and $0 < \theta < \pi$. The two values μ and θ are known respectively as the *eigenaxis* and *eigenangle* of the quaternion. Euler's formula generalizes to quaternions:

$$\exp(\mu\theta) = \cos\theta + \mu\sin\theta$$

where the cosine component is the scalar part, and the sine component is the vector part. Note that, because μ is a unit pure quaternion, its square is -1. It is thus a generalization of the complex operator to three dimensions.

The eigenaxis μ is very simply computed as:

$$\mu = \mathbf{V}(q)/|\mathbf{V}(q)|$$

with the only exceptional case when $\mathbf{V}(q) = 0$, in which case μ is undefined.

The eigenangle, or phase is easily computed as

$$\theta = \tan^{-1} |\mathbf{V}(q)|/S(q)$$

and is always positive. It is undefined if q is zero, (as is the argument of a complex number).

References

1. S. J. Sangwine, Electronics Letters, 34, 10, (1998).
2. C. J. Evans, S. J. Sangwine, and T. A. Ell, Colour sensitive edge detection using hypercomplex filters, Proc. Eusipco 2000, pp. 107–110, (2000).
3. C. J. Evans, T. A. Ell, and S. J. Sangwine, Hypercomplex color-sensitive smoothing filters, Proc. ICIP2000, pp. 541–544, (2000)
4. K. N. Plataniotis and A. N. Venetsanopoulos, Vector filtering, in The Colour Image Processing Handbook, S. J. Sangwine and R. E. N. Horne, Eds., chapter 10, Chapman and Hall, London, 1998, pp. 188–209.
5. Gene H. Golub and Charles F. Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore and London, second edition, 1989.
6. David Nelson, Ed., The Penguin Dictionary of Mathematics, Penguin Books, London, second edition, 1998.
7. P. M. Cohn, Solid Geometry, Routledge and Kegan Paul, London, 1961.
8. H. S. M. Coxeter, American Mathematical Monthly, **53**, Mar. 1946.
9. S.-C. Pei and C.-M. Cheng, IEEE Transactions on Communications, **45**, 5, (1997).
10. S. C. Pei and C. M. Cheng, IEEE Transactions on Image Processing, **8**, 5, (1999).
11. Soo-Chang Pei, Jian-Jiun Ding, and Ja-Han Chang, IEEE Transactions on Signal Processing, **49**, 11, (2001).
12. S. J. Sangwine, Electronics Letters, **32**, 21, (1996).
13. S. J. Sangwine and T. A. Ell, The discrete Fourier transform of a colour image, Proc. Second IMA Conference on Image Processing, pp. 430–441, (1998).
14. T. A. Ell and S. J. Sangwine, Decomposition of 2D hypercomplex Fourier transforms into pairs of complex Fourier transforms, Proc. Eusipco 2000, pp. 1061–1064, (2000).
15. S. J. Sangwine and T. A. Ell, Hypercomplex Fourier transforms of color images, Proc. ICIP 2001, **I**, pp. 137–140 (2001).
16. T. A. Ell and S. J. Sangwine, Hypercomplex Wiener-Khintchine theorem with application to color image correlation, Proc. ICIP2000, pp. 792–795 (2000).
17. S. J. Sangwine, T. A. Ell, and C. E. Moxey, Electronics Letters, **37**, 25, (2001).

Biography

Stephen Sangwine is a Senior Lecturer in the Department of Electronic Systems Engineering at the University of Essex, UK. He received a BSc in Electronic Engineering from the University of Southampton, UK in 1979 and a PhD from the University of Reading in 1991. His interests include linear vector filtering and transforms of colour images; non-linear vector image filtering; and digital hardware design. He is an IEE Member, and a Senior Member of the IEEE.