Steerable filters generated with the hypercomplex dual-tree wavelet transform

WEDEKIND, J., AMAVASAI, B. P. and DUTTON, K.

Published version

WEDEKIND, J., AMAVASAI, B. P. and DUTTON, K. (2007). Steerable filters generated with the hypercomplex dual-tree wavelet transform. In: IEEE International Conference on Signal Processing and Communications (ICSPC 2007), Dubai, United Arab Emirates, 24-27 November 2007. IEEE, 1291-1294.

2007 IEEE International Conference on Signal Processing and Communications (ICSPC 2007), 24-27 November 2007, Dubai, United Arab Emirates

STEERABLE FILTERS GENERATED WITH THE HYPERCOMPLEX DUAL-TREE WAVELET TRANSFORM

J. Wedekind, B. Amavasai, K. Dutton

MMVL, Materials and Engineering Research Institute
Sheffield Hallam University,
Pond Street,
Sheffield S1 1WB
{J.Wedekind,B.P.Amavasai,K.Dutton}@shu.ac.uk

ABSTRACT

The use of wavelets in the image processing domain is still in its infancy, and largely associated with image compression. With the advent of the dual-tree hypercomplex wavelet transform (D-HWT) and its improved shift invariance and directional selectivity, applications in other areas of image processing are more conceivable. This paper discusses the problems and solutions in developing the DHWT and its inverse. It also offers a practical implementation of the algorithms involved. The aim of this work is to apply the DHWT in machine vision.

Tentative work on a possible new way of feature extraction is presented. The paper shows that 2-D hypercomplex basis wavelets can be used to generate steerable filters which allow rotation as well as translation.

Index Terms— Image Processing, Wavelet transforms, Feature extraction, Algorithms, Linear systems

1. INTRODUCTION

Wavelets are of significant interest in signal processing. However in contrast to the discrete Fourier transform the discrete wavelet transform is not shift invariant. In the area of image processing this has restricted the use of the wavelet transform to areas such as image compression where shift invariance is not a requirement. Recent research in wavelet signal processing however has resulted in the dual-tree complex wavelet transform[1] which offers approximate shift invariance and amplitude-phase analysis.

Analogical to 1-D signals requiring a pair of complex wavelets, 2-D signals require a quadruple of hypercomplex wavelets for analysis[2]. This analogy extends to higher dimensions as well, and the hypercomplex wavelet transform can for example be used to filter 3-D data[3]. The hypercomplex wavelet transform has already been used for optic flow estimation, texture segmentation, and feature extraction.

This paper outlines a complete implementation of Selesnick's biorthogonal wavelet filter design technique and the dual-tree hypercomplex wavelet transform. The dual-tree hypercomplex wavelet is then used to generate three steerable filters which allow rotation as well as translation.

2. STATE OF THE ART

An outline of Selesnick's filter design technique for designing biorthogonal wavelets[4] is hereby given. For a more detailed introduction to the dual-tree wavelet transform see Selesnick's joint publication with Kingsbury[1].

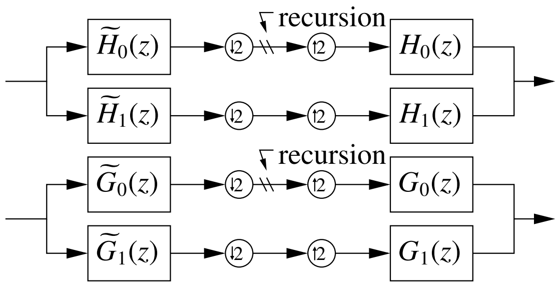


Figure 1. Biorthogonal wavelet filters

If h_0, h_1, $\widetilde{h}_0$, and $\widetilde{h}_1$ are odd-length real-valued filters and H_0, H_1, $\widetilde{H}_0$, and $\widetilde{H}_1$ their Z-transforms, the perfect reconstruction condition for the first filter bank in figure 1 is

$$\begin{gathered}\widetilde{H}_0(z)\,H_0(-z) + \widetilde{H}_1(z)\,H_1(-z) = 0 \text{ and}\\ \widetilde{H}_0(z)\,H_0(z) + \widetilde{H}_1(z)\,H_1(z) = 2\end{gathered} \tag{1}$$

By letting $H_1(z) = \widetilde{H}_0(-z)$ and $\widetilde{H}_1(z) = -H_0(-z)$ the first part of equation (1) is satisfied. The biorthogonality is established by

$$\begin{aligned} &H_0(z) = F(z)\,D(z), && \widetilde{H}_0(z) = \widetilde{F}(z)\,D(z^{-1})\,z^{1-L},\\ &G_0(z) = F(z)\,D(z^{-1})\,z^{1-L}, && \widetilde{G}_0(z) = \widetilde{F}(z)\,D(z)\\ &\text{with } F(z) = Q(z)\,(1+z^{-1})^K \text{ and } \widetilde{F}(z) = \widetilde{Q}(z)\,(1+z^{-1})^{\widetilde{K}} \end{aligned} \tag{2}$$

where K and $\widetilde{K}$ are the numbers of desired vanishing moments. D is a Thiran filter[4] to approximate a half sample delay

$$d(n) = \binom{L-1}{n}(-1)^n \prod_{k=0}^{n-1} \frac{\tau - L + 1 + k}{\tau + 1 + k}, \text{ here } \tau = 0.5 \tag{3}$$

To also fulfil the second part of equation (1) both filter-pairs have to meet the following condition

$$\begin{gathered}\widetilde{H}_0(z)\,H_0(z) = \widetilde{G}_0(z)\,G_0(z) = Q(z)\,\widetilde{Q}(z)\,S(z) = 2\\ \text{where } S(z) \coloneqq (1+z^{-1})^{K+\widetilde{K}}\,D(z)\,D(z^{-1})\,z^{1-L}\end{gathered} \tag{4}$$

Solving the following equation system yields $R(z) \coloneqq Q(z)\,\widetilde{Q}(z)$.

$$\begin{pmatrix} s_N & 0 & \cdots & & \\ s_{N-2} & s_{N-1} & s_N & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & 0 & s_1 & s_2 & s_3 \\ & & \cdots & 0 & s_1 \end{pmatrix} \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{N-1} \\ r_N \end{pmatrix} = \begin{pmatrix} \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix} \tag{5}$$

where $N = K + \widetilde{K} + 2\,L - 1$, and N is odd

This project was supported by the Nanorobotics EPSRC Basic Technology grant GR/S85696/01

Note that R is symmetric ($R(z) = R(z^{-1})\,z^{N-1}$) because of equation (5).

3. SPECTRAL FACTORISATION WITH LAGUERRE

To determine a pair of spectral factors Q and $\widetilde{Q}$, each root of R is assigned to either become a root of Q or a root of $\widetilde{Q}$. Since R is symmetric and odd-length, for every root o_i there is a related root at $1/o_i$. As R also is real-valued, each complex root therefore has related roots at o_i^*, $1/o_i$, and $1/o_i^*$. The roots of $R(z)\,z^{N-1}$ can be determined using Laguerre's iterative method[5] and polynomial division. Instead of reducing the polynomial by only a single root, first polynomial division with $(1-o_i)\,(1-o_i^*)\,(1-1/o_i)\,(1-1/o_i^*)$ is attempted. If the error is too large, the occurrence of a pair of real roots is assumed and reduction with $(1-o_i)\,(1-1/o_i)$ is performed. This approach allows to safely choose the roots in the next step. Polynomial division without remainders is formulated as a least squares problem as shown in [6].

As Q and $\widetilde{Q}$ need to be symmetric and real-valued, each root of a group of two or four related roots must be assigned to the same spectral factor. Furthermore the difference in size of Q and $\widetilde{Q}$ should be minimal. Applying these criteria can still leave a list of choices. At least for larger filters however there does not seem to be much difference between these.

After choosing a spectral factorisation the filters can be computed according to equation (2). Finally the filters are normalised. Note that equation (5) requires $r_1 = 0$ and $r_N = 0$. This is solved by performing spectral factorisation for $r_2\,z^{N-2} + r_3 + z^{N-3} + \ldots + r_{N-1}$ and later extending $\widetilde{Q}(z)$ with a zero coefficient at the beginning and the end.

4. 2-D HYPERCOMPLEX WAVELET TRANSFORM

The two-dimensional wavelet tree shown in [7], which already uses four-element vectors, can be represented using hypercomplex numbers as follows. First the real-valued image is multiplied with $(1+i+j+k)$ so that all four components of the resulting hypercomplex number equal each other.

$$prepare(X)(\vec{z}) := X(\vec{z})\,(1+i+j+k) \tag{6}$$

$1, i, j, k \in HCA_2$ are the units of the commutative hypercomplex algebra HCA_2[2]. The layers of the wavelet pyramid are computed by recursively applying the following function to the lower frequency band

$$\begin{aligned} &decompose^{(1)}_{a,b}(W)(\vec{z}) := \\ &\quad [\downarrow_{z_2} 2]\Big([\downarrow_{z_1} 2](\mathcal{R}W(\vec{z})\,\widetilde{H}_a(z_1))\,\widetilde{H}_b(z_2)\Big) + \\ &\quad [\downarrow_{z_2} 2]\Big([\downarrow_{z_1} 2](\mathcal{I}W(\vec{z})\,\widetilde{G}_a(z_1))\,\widetilde{H}_b(z_2)\Big) + \\ &\quad [\downarrow_{z_2} 2]\Big([\downarrow_{z_1} 2](\mathcal{J}W(\vec{z})\,\widetilde{H}_a(z_1))\,\widetilde{G}_b(z_2)\Big) + \\ &\quad [\downarrow_{z_2} 2]\Big([\downarrow_{z_1} 2](\mathcal{K}W(\vec{z})\,\widetilde{G}_a(z_1))\,\widetilde{G}_b(z_2)\Big) \\ &\quad \text{where } a,b \in \{0,1\} \end{aligned} \tag{7}$$

The operators $\mathcal{R}$, $\mathcal{I}$, $\mathcal{J}$, and $\mathcal{K}$ are for accessing the different components of the hypercomplex number.

For the inverse wavelet transform the values are recursively composed using the following function

$$\begin{aligned} &compose(W_{0,0}, W_{1,0}, W_{0,1}, W_{1,1})(\vec{z}) := \\ &\sum_{a,b\in\{0,1\}} \Big([\uparrow_{z_2} 2]\Big([\uparrow_{z_1} 2](\mathcal{R}W_{a,b}(\vec{z})\,H_a(z_1))\,H_b(z_2)\Big) + \\ &\quad [\uparrow_{z_2} 2]\Big([\uparrow_{z_1} 2](\mathcal{I}W_{a,b}(\vec{z})\,G_a(z_1))\,H_b(z_2)\Big)\,i + \\ &\quad [\uparrow_{z_2} 2]\Big([\uparrow_{z_1} 2](\mathcal{J}W_{a,b}(\vec{z})\,H_a(z_1))\,G_b(z_2)\Big)\,j + \\ &\quad [\uparrow_{z_2} 2]\Big([\uparrow_{z_1} 2](\mathcal{K}W_{a,b}(\vec{z})\,G_a(z_1))\,G_b(z_2)\Big)\,k \Big) \end{aligned} \tag{8}$$

The real-valued image is reconstructed by applying the inverse of

$$finalise(W)(\vec{z}) := \frac{1}{4}\left(\mathcal{R}W(\vec{z}) + \mathcal{I}W(\vec{z}) + \mathcal{J}W(\vec{z}) + \mathcal{K}W(\vec{z})\right) \tag{9}$$

5. IMPLEMENTATION

We have implemented the DHWT in Y. Matsumoto's programming language Ruby. Since Ruby is an interpreted language, the code can be used in an interactive Ruby session. T. Hunter's image processing extension was used to load and save images.

While the datatypes for representing 2-D arrays of hypercomplex numbers can be implemented in Ruby easily, the performance is insufficient to process images in real-time. As a solution M. Tanaka has implemented *NArray* which is a static datatype for Ruby to manipulate large arrays in real-time. Unfortunately the code is static and cannot be easily extended. Therefore an array datatype was implemented which allows definition of custom element-types.

Ruby offers methods to pack numerical data into a platform-dependent binary representation. *E.g.* integers can be converted to bytes and later on be retrieved as follows

```
[1,2].pack("cc")           => "\001\002"
"\001\002".unpack("cc") => [1, 2]
```

This allows the implementation of an array datatype in Ruby which operates on binary data. A custom element-type can be created by implementing a corresponding mapping to and from binary data. Similar as in the *NArray* implementation, array elements are only temporarily represented as Ruby objects.

Ruby allows introspection, *i.e.* the existence of a method with a certain name can be checked during run-time using the method `Object::respond_to?`. This can be used to develop a method which tries to invoke an efficient native implementation before falling back to using a slower generic implementation.

A large number of native implementations is required to cover all possible operations. There are 12 element datatypes (integer, complex, ...), 3 unary operations (negation, square root, absolute value), 3 accumulating operations (minimum, maximum, sum), and 6 binary operations (minus, plus, multiply, ...). Furthermore native implementations for down-, and upsampling, correlation, type-conversions, and extraction of sub-arrays are required. Optimising binary operations is especially hard because in each case there is an array-array-operation, a scalar-array-operation, and an array-scalar-operation to be supported. $12\cdot 12\cdot 3\cdot 6 = 2592$ different native methods are required to provide for all possible binary operations.

Instead of implementing a code-generator as in the *NArray* project, the problem was addressed by nesting C++ templates. The major obstacles to this approach can be overcome by using template meta-programming techniques which were developed within the Boost project[8]. For example an entry of the compile-time look-up table for return-types of binary operations is implemented as follows

```
template<>
struct _coercion< complex< double >,
                  hypercomplex< float > >
{
  typedef hypercomplex< double > type;
};
```

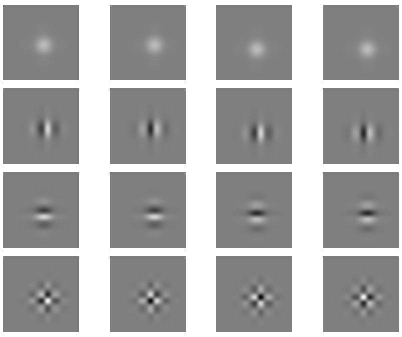
In a similar way function objects are selected and method names are computed. Also the conversion from a C++ datatype to a Ruby class requires the use of templates.

6. STEERABLE FILTERS

Table (1) shows all components of the four hypercomplex basis wavelets of the third level of the wavelet pyramid. The images were generated by composing a wavelet pyramid of zeros with a single hypercomplex impulse (for example $W^{(3)}_{a,b}(\vec{z}) = z_1^{-4}\,z_2^{-4}\,j$) where $(-4,-4)$ is next to the centre of the pyramid).

Table 1. Hypercomplex basis wavelets

		$W^{(3)}_{a,b}(\vec{z}) = z_1^{-4}\,z_2^{-4}\,h$			
b	a	$h = 1$	$h = i$	$h = j$	$h = k$
0	0				
0	1				
1	0				
1	1				



We can pool the four hypercomplex coefficients for a linear combination of the four basis wavelets shown in table 1 as a 2×2 matrix so that $W^{(3)}_{a,b} = z_1^{-4}\,z_2^{-4}\,v_{a,b}$

$$\mathcal{V} = \begin{pmatrix} v_{0,0} & v_{0,1} \\ v_{1,0} & v_{1,1} \end{pmatrix}, \mathcal{V} \in HCA_2^{2\times 2} \tag{10}$$

One can see in table 1 that the low-frequency wavelets have a pattern which has half the frequency of its high-frequency sibling. If we take this into account, we can model small translations of the texture defined by $\mathcal{V}$ as follows

$$e^{\begin{pmatrix} 2\pi\,\Delta y\, j/2 & 0 \\ 0 & 2\pi\,\Delta y\, j \end{pmatrix}} \cdot \mathcal{V} \cdot e^{\begin{pmatrix} 2\pi\,\Delta x\, i/2 & 0 \\ 0 & 2\pi\,\Delta x\, i \end{pmatrix}} \tag{11}$$

Note that this model requires a commutative algebra, *i.e.* this forbids the use of quaternions. The pattern is centred if $\mathcal{V} = \mathcal{A}(1+i+j+k)$ where $\mathcal{A}$ is real-valued ($\mathcal{A} \in \mathbb{R}^{2\times 2}$). Table (2) shows the result.

One can see in table 1 that the low frequency wavelets can be used to generate a steerable gradient-like shape. Term (12) yields the rotating pattern shown in table (3)).

$$(1-k)\,\cos(\alpha)\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + (i-j)\,\sin(\alpha)\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{12}$$

Figure 2 shows how the frequency domain is covered by the basis wavelets. As can be observed, modelling rotations in general is much more difficult, because signal energy is transferred

Table 2. translation in x-direction ($\Delta y = 0$)

$\Delta x = 0$	$\frac{1}{6}$	$\frac{2}{6}$	$\frac{3}{6}$	$\frac{4}{6}$	$\frac{5}{6}$

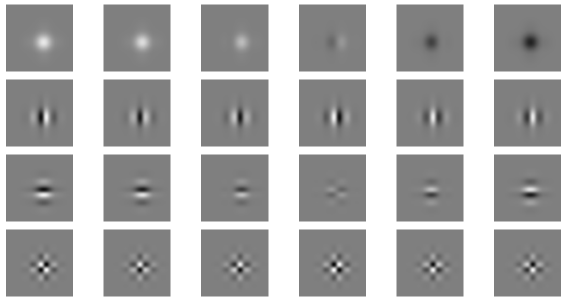
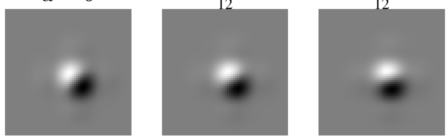
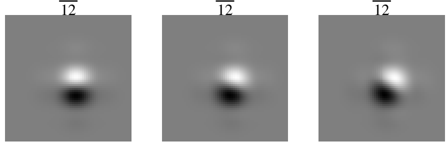


Table 3. rotating gradient shape

$\alpha = 0$	$\frac{\pi}{12}$	$\frac{2\pi}{12}$



$\frac{3\pi}{12}$	$\frac{4\pi}{12}$	$\frac{5\pi}{12}$



between different basis wavelets. *E.g.* if all signal energy is concentrated in $v_{0,1}$ and in the first quadrant (see figure 2), a rotation of $\frac{\pi}{2} - \rho$ (where $\rho := \sin^{-1}(\frac{1}{3})$, see figure 2) will transfer all energy to $v_{1,0}$. A solution to this problem is to use polar separable

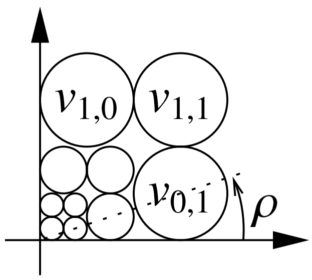


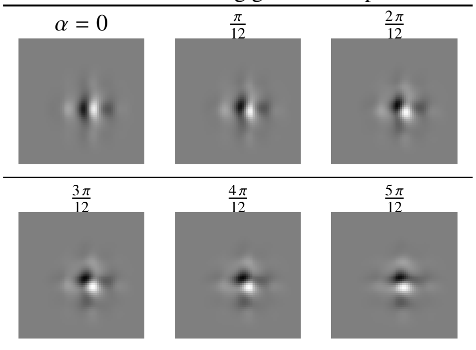
Figure 2. Basis wavelets of different scale covering the first quadrant of the frequency domain

filters as in [9]. However this approach does not allow to model the translations as shown above.

However using linear combinations of the basis wavelets (see table 1) one can approximate rotating patterns. Using the term (13), table (4) was generated.

$$\begin{aligned} &(1+k)\,\cos(\alpha+\rho)\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + (i+j)\,\cos(\alpha-\rho)\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + \\ &(1+k)\,\sin(\alpha+\rho)\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + (i+j)\,\sin(\alpha-\rho)\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + \\ &\frac{1}{2}\,(1-i)\,\sin(\alpha)\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} + \frac{1}{2}\,(1-j)\,\cos(\alpha)\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned} \tag{13}$$

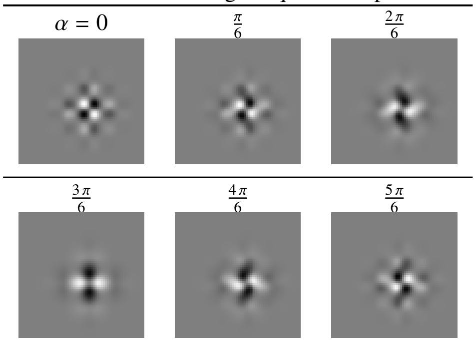
Table 4. rotating gradient shape



Term (14) generates a rotating chequered shape (see table (5). One can see that the outer fringes of the filter are not participating in the rotation of the pattern.

$$\begin{aligned}&(1+i+j+k)\cos(\alpha)\begin{pmatrix}0&0\\0&1\end{pmatrix}+\\&(1+i-j-k)\sin(\alpha)\begin{pmatrix}0&0\\1&0\end{pmatrix}+(-1+i-j+k)\sin(\alpha)\begin{pmatrix}0&1\\0&0\end{pmatrix}\end{aligned}\tag{14}$$

Table 5. rotating chequered shape



Using the addition theorems, terms (12), (13), and (14) each can be brought into the following form which represents a steerable filter

$$\cos(\alpha)\mathcal{H}_1+\sin(\alpha)\mathcal{H}_2,\ \text{where}\ \mathcal{H}_1,\mathcal{H}_2\in HCA^{2\times 2}\tag{15}$$

7. CONCLUSION

A complete implementation of Kingsbury's dual-tree hypercomplex wavelet transform including Selesnick's filter design has been given. A fully functional program for manipulating arrays of hypercomplex numbers in Ruby was implemented. The program then was optimised by adding native methods for element-wise operations into this framework. This concept allows real-time performance to be achieved without sacrificing the flexibility of the datastructures in use. The implementation is available for free on the Nanorobotics website[1] under the terms and conditions of the GPL. Our implementation does not rely on proprietary software and therefore can potentially be integrated into an embedded platform.

It has been shown, how the basis wavelets can be used to model translation of patterns. Furthermore three patterns have been presented which allow approximate rotations as well. Future work will attempt to model rotating patterns more accurately. The motivation is to be able to represent arbitrary texture patches as a linear combination of steerable wavelets which can be steered both in rotation as well as translation. If such a wavelet basis exists, it would be possible to model arbitrary translations and rotations as operations in the hypercomplex domain. A feature extraction method based on this model would then be able to pick out salient features (*e.g.* edges, corners, and joints) and recover them regardless of rotation, translation, and scale.

Furthermore we would like to point out that the redundancy of the dual-tree complex wavelet transform can be overcome by using the softy-space projection[10] which relieves the redundancy by projecting the real-valued image on a hypercomplex image of lower resolution.

[1] http://vision.eng.shu.ac.uk/mmvlwiki/index.php/Nanorobotics

8. REFERENCES

[1] I. W. Selesnick, R. G. Baraniuk, and N. C. Kingsbury, "The dual-tree complex wavelet transform," *IEEE Signal Processing Magazine*, vol. 22, no. 6, pp. 123–51, Nov. 2005.

[2] T. Bülow, *Hypercomplex Spectral Signal Representations for Image Processing and Analysis*, Ph.D. thesis, Christian-Albrechts-Universität, Kiel, Germany, 1999.

[3] I. W. Selesnick and K. Y. Li, "Video denoising using 2d and 3d dual-tree complex wavelet transforms," in *Wavelets: application in Signal and Image Processing X*, Nov. 2003, vol. 5207, pp. 607–18.

[4] I. W. Selesnick, "The design of approximate Hilbert transform pairs of wavelet bases," *IEEE Transactions on Signal Processing*, vol. 50, no. 5, pp. 1144–52, May 2002.

[5] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: the Art of Scientific Computing*, Cambridge University Press, 1992.

[6] R. M. Corless, M. W. Giesbrecht, M. van Hoeij, I. S. Kotsireas, and S. M. Watt, "Towards factoring bivariate approximate polynomials," in *Proceedings of 2001 International Symposium on Symbolic and Algebraic Computation*, July 2001, pp. 85–92.

[7] N. Kingsbury, "Complex wavelets for shift invariant analysis and filtering of signals," *Applied and Computational Harmonic Analysis*, vol. 10, no. 3, pp. 234–253, 2001, Journal.

[8] A. Gurtovoy and D. Abrahams, "The boost c++ metaprogramming library," Tech. Rep., Mar. 2002, http://www.boost.org/libs/mpl/doc/paper/mpl_paper.pdf.

[9] Anil Anthony Bharath and Jeffrey Ng, "A steerable complex wavelet construction and its application to image denoising," *IEEE Transactions on Image Processing*, vol. 14, pp. 948–959, 2005.

[10] F. C. A. Fernandes, I. W. Selesnick, R. L. C. van Spaendonck, and C. S. Burrus, "Complex wavelet transforms with allpass filters," *Signal Processing*, vol. 83, pp. 1689–706, Aug. 2003, Elsevier International Journal.