Developers Guide

Vision Systems Laboratory, Dublin City University
info@neatvision.com

1. Introduction

NeatVision was originally designed so that it could be easily extended by building on previously developed algorithms. This feature has been finalised with the release of version 2.x of the NeatVision visual programming environment. This document outlines how to:

- develop new NeatVision components that can ultimately be reused by other NeatVision developers
- reuse the core NeatVision components in new user defined components
- submit your component or library of components to wider NeatVision community.

The following sections assume a basic level of familiarity with the Java programming language from Sun Microsystems and the NeatVision developers plug-in. Additional details on the design concepts behind NeatVision along with detailed explanations of many of the its algorithms can be found in *P.F. Whelan and D. Molloy (2000),* ***Machine Vision Algorithms in Java: Techniques and Implementation,*** *Springer (London), 298 Pages [ISBN 1-85233-218-2]* .

This document relates to the NeatVision components `neatvision.jar` (version 2.1) and `developer.jar` (version 2.0) used in conjunction with SUNs `J2SDK` (version: 1.3.1.07) and `JAI` (version: 1.1.2.beta).

Note to developers:
The default set-up for NeatVision is to stop detailed messages been sent to the start-up DOS window (this speeds up the user interaction). When developing code it is useful to enable messaging to allow developers to view all internal messages. This requires the *–verbose* flag to be set. For example:

```
C:\...\java.exe -classpath d:\..\neatvision.jar;d\..\developer.jar NeatVision -verbose
```

To send messages to this window from your NeatVision program use the following:

```
int cc=10;
System.out.println("cc"+cc);
```

2. Developing for Reuse

The skeleton code for a new NeatVision component is generated using the component development wizard (see Example 1). In the previous version of NeatVision (version 1.0) the entry point for a component was the **`main()`** method and the programmer was responsible for interfacing directly with component inputs and outputs to read and write the associated data values. In NeatVision 2.x the **`main()`** method is replaced by the **`create()`** method. This revised

approach makes the development of new NeatVision components more straightforward and facilitates component reuse.

The programmer is no longer required to interface directly with the component inputs and outputs. Instead, when a component becomes active, NeatVision automatically reads the data values at the inputs to a component and passes the associated data to the **`create()`** method in the form of a populated **`DataBlock`** object. Each entry in the **`DataBlock`** object corresponds to the data at the input with the corresponding index (0, 1, 2, etc. 0 being the topmost input). After the input data has been processed the results must be stored in a new **`DataBlock`** object which is returned from the **`create()`** method upon completion[1]. Each entry in the returned **`DataBlock`** object is then passed to the output with the corresponding index (0, 1, 2, etc. 0 being the topmost output).

2.1 The `DataBlock` class

The **`DataBlock`** class is used to represent the input and output data values associated with a particular NeatVision component. The data associated with a **`DataBlock`** object is represented as an array of objects. This means that the **`DataBlock`** class is future proof and will deal with any type of data that may be supported either by the core NeatVision components or any custom components developed by NeatVision users. The specification for the DataBlock class is listed below in sections 2.1.1 and 2.2.2.

2.1.1 The *`get`* Methods of the `DataBlock` class

The methods provided by the **`DataBlock`** class for reading the data values that it stores. Support is provided for primitive data types and images. Any other data values are treated as objects.

- **`int getInteger(int index)`**
  Get the integer value at the input with the specified index.
- **`long getLong(int index)`**
  Get the long primitive value at the input with the specified index.
- **`float getFloat(int index)`**
  Get the float primitive value at the input with the specified index.
- **`double getDouble(int index)`**
  Get the double primitive value at the input with the specified index.
- **`boolean getBoolean(int index)`**
  Get the Boolean primitive value at the input with the specified index.
- **`String getString(int index)`**
  Get the **`String`** object at the input with the specified index.
- **`Image getImage(int index)`**
  Get the **`java.awt.Image`** object at the input with the specified index. This object can be used for operations that require an image of this class i.e AWT imaging operations.
- **`GrayImage getGrayImage(int index)`**
  Get the **`GrayImage`** object at the input with the specified index. This object can be used for getting direct access to the pixel data for a greyscale image
- **`RenderedOp getRenderedOp(int index)`**
  Get the **`javax.media.jai.RenderedOp`** object with the specified index. This object can be used in conjunction with Java Advanced Imaging (JAI) operators.
- **`public RGBImage getRGBImage(int index)`**
  Get the **`RGBImage`** object at the input with the specified index. This object can be used for getting direct access to the pixel data for a colour image.

[1] **Note:** If only one object is being returned from the **`create()`** method (i.e. if the block has only one output) then it is not necessary to encapsulate this within a `DataBlock` object. Instead, it can be returned directly and NeatVision will pass the returned object to the single output of the component.

- **`Object get(int index)`**
  Get the object at the input with the specified index. This method can be used for reading data not supported by any of the methods listed above.

2.1.1 The `set` Methods of the `DataBlock` class

The methods provided by the **`DataBlock`** class for adding new data. Support is provided for primitives. Any other data values are treated as objects.

- **`void add(int value)`**
  Add the integer primitive argument at the next available index.
- **`void add(long value)`**
  Add the long primitive argument at the next available index.
- **`void add(float value)`**
  Add the float primitive argument at the next available index.
- **`void add(double value)`**
  Add the double primitive argument at the next available index.
- **`void add(boolean value)`**
  Add the Boolean primitive argument at the next available index.
- **`void add(Object object)`**
  Add the object argument at the next available index. This method should be used for adding any data that is not a primitive to a **`DataBlock`** object.

Note that the values stored in a **`DataBlock`** object can be retrieved in any order, however they must be stored in the same order that they appear at the output of the relevant component.

3. How to Reuse

The functionality provided by any of the core NeatVision classes can be called from within custom user defined classes that are developed using the NeatVision developers plug-in. This is achieved by calling the static **`create()`** method of the **`NeatVision`** class.

- **`Object NeatVision.create(String class, DataBlock args)`**

The parameters of the **`create()`** method are a **`String`** object and **`DataBlock`** object. The **`String`** object represents the class name of the desired component and the **`DataBlock`** object represents the parameters that will be passed to an off-screen instantiation of the desired component. The **`DataBlock`** argument must have the same number of entries as the number of inputs connected to the desired component and each entry must represent the data required by the associated input (0, 1, 2, etc.). The **`create()`** method then returns a new **`DataBlock`** object that represents the outputs that were generated after the requested component processed the specified inputs. The **`create()`** method can also handle up to four arguments that are not encapsulated within a **`DataBlock`** object, for example:

- **`Object NeatVision.create(String class, Object arg0)`**
  Call the create method of the single input component with name ‘class’.
- **`Object NeatVision.create(String class, Object arg0, Object arg1)`**
  Call the create method of the dual input component with the name ‘class’.

All arguments must be represented as objects when using this approach. This means that any primitives must be wrapped before being passed to the **`create()`** method. Take the integer arguments for the dual threshold operation as an example:

```
int hiThresh = 100;
int loThresh = 100;

Integer hiThreshObj = new Integer(hiThresh);
Integer loThreshObj = new Integer(loThresh);

GreyImage output = (GreyImage)NeatVision.create("DualThreshold",input,hiThreshObj,loThreshObj);
```

There are special wrapper classes available for converting all primitive types (**`boolean`**, **`byte`**, **`short`**, **`int`**, **`long`**, **`double`** and **`float`**) into objects (**`Boolean`**, **`Byte`**, **`Short`**, **`Integer`**, **`Long`**, **`Double`** and **`Float`**). Objects of these classes can be constructed by simply passing the relevant primitive to the constructor of the relevant class (see **`int`** to **`Integer`** example above). The static **`create()`** method of the NeatVision class routes the specified **`DataBlock`** object to the **`create()`** method of the specified class and returns the resulting output **`DataBlock`** object.

```
import DataBlock;
import CoreInterface;

public class testComponent extends CoreInterface
{
	public testComponent()
	{
		name = "test";
		inputs = 2;
		outputs = 1;
		width = 30;
		height = 20;
	}

	public void setup()
	{
		Input[0].setConnectionType(UNDEFINED);
		Input[0].setConnectionMode(NORMAL);
		Input[0].shortDescription = "";
		Input[0].setConnectionDescription(new String[]{
			"No connection description available"});

		Input[1].setConnectionType(UNDEFINED);
		Input[1].setConnectionMode(NORMAL);
		Input[1].shortDescription = "";
		Input[1].setConnectionDescription(new String[]{
			"No connection description available"});

		Output[0].setConnectionType(UNDEFINED);
		Output[0].setConnectionMode(NORMAL);
		Output[0].shortDescription = "";
		Output[0].setConnectionDescription(new String[]{
			"No connection description available"});
	}

	public void doubleClick()
	{

	}

	public Object create(DataBlock args)
	{
		return null;
	}
}
```

Example 1: The skeleton code for a double input/single output component. Note that the entry point is the **`create()`** method. This is called whenever the block receives a full complement of input data.

```
import DataBlock;
import CoreInterface;
import NeatVision;

public class testComponent extends CoreInterface
{
	public testComponent()
	{
		name = "test";
		inputs = 1;
		outputs = 1;
		width = 30;
		height = 20;
	}

	public void setup()
	{
		Input[0].setConnectionType(UNDEFINED);
		Input[0].setConnectionMode(NORMAL);
		Input[0].shortDescription = "";
		Input[0].setConnectionDescription(new String[]{
			"No connection description available" });

		Output[0].setConnectionType(UNDEFINED);
		Output[0].setConnectionMode(NORMAL);
		Output[0].shortDescription = "";
		Output[0].setConnectionDescription(new String[]{
			"No connection description available" });
	}

	public void doubleClick()
	{

	}

	public Object create(DataBlock args)
	{
		GrayImage input = (GrayImage)args.get(0);
		GrayImage output = (GrayImage)NeatVision.create("Not",input);
		return output;
	}
}
```

Example 2: A simple example of reuse, calling the `Not` operation from inside a custom user defined class.

Additional detailed examples of component reuse can be found in Appendix A. The list of core NeatVision classes available for reuse are listed in Appendix B.

3.1 Practical programming issues:

One typical problem that can occur with other versions of the JDK involves the failure of the NeatVision block to be updated once the new component has compiled correctly. This can be overcome by deleting it from your visual programme and reloaded prior to use (this does not effect the other block elements in you visual workspace). Failure to do so may cause the class loader to ignore your changes and in turn may lock the system. Also avoid reusing variable names automatically generated by the wizard e.g. width/height as this may cause conflicts within your own code segment declarations.

4. Component/Library Submission

If you feel that you would like to make your components or libraries to be available to the wider NeatVision community then please submit:

- The relevant component/library class files and package information for all components
- Sample images / data for evaluating your component/library
- Appropriate documentation
- User details and licence information for your submission

This information to be sent to tech@neatvision.com using the subject header “Submission”. Your libraries and support information will placed on the NeatVision web site for general access.

5. Conditions of Use

If you have found this software useful and/or used it as part of your research work, you are requested to cite the following

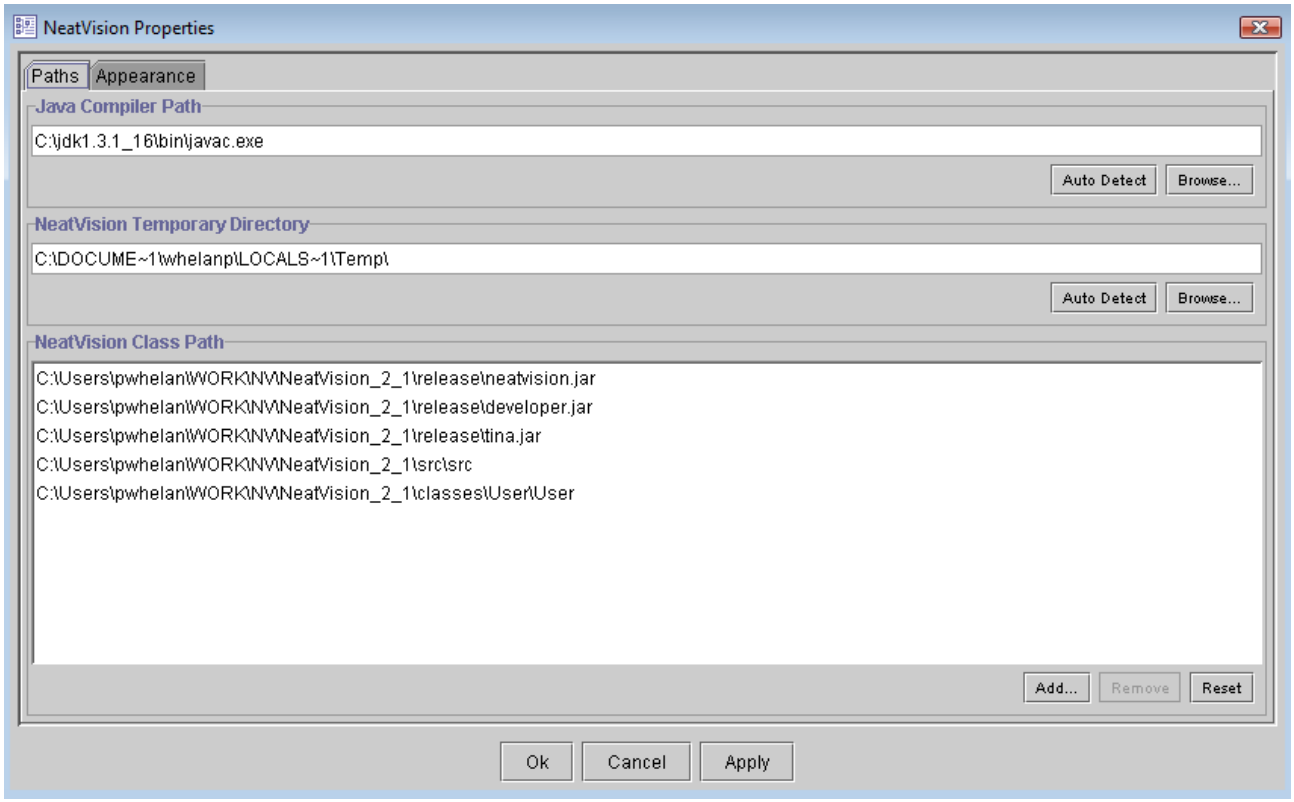
- P.F. Whelan and D. Molloy (2000), **Machine Vision Algorithms in Java: Techniques and Implementation,** Springer (London)

6. Terms and Conditions

All downloads are subject to the following NeatVision License And Terms and Conditions.

Vision Systems Group
16 September 2003

Notes:



Delete “package User;” from wizard generated file

Appendix A

Appendix A.1: `TestDevPixelLevel.java`

```
// TestDevPixelLevel.java

// Project Name:      NeatVision (Ver 2.0)
// Written by:        Paul Whelan
// Initial Version:   03/03/03
// Latest Revision:
// Description:       Sample file to illustrate NeatVision pixel manipulation
//*************************************************************************************
// Copyright (C) 2003, Paul F Whelan, Vision Systems Group, DCU
// This library is distributed in WITHOUT ANY WARRANTY; without even the implied warranty
// of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

import DataBlock;
import CoreInterface;

public class TestDevPixelLevel extends CoreInterface
{
       public TestDevPixelLevel()
       {
              name = "TestDevPixelLevel";
              inputs = 2;
              outputs = 3;
              width = 30;
              height = 30;
       }

       public void setup()
       {
              Input[0].setConnectionType(IMAGE);
              Input[0].setConnectionMode(NORMAL);
              Input[0].shortDescription = "";
              Input[0].setConnectionDescription(new String[]{
                     "Input GS image"      });

              Input[1].setConnectionType(INTEGER);
              Input[1].setConnectionMode(NORMAL);
              Input[1].setDefaultValue(new Integer(100));                          // Setup an input default value
              Input[1].shortDescription = "[Integer] [Default = 100]";
              Input[1].setConnectionDescription(new String[]{
                     "Grey scale offset"   });

              Output[0].setConnectionType(IMAGE);
              Output[0].setConnectionMode(NORMAL);
              Output[0].shortDescription = "";
              Output[0].setConnectionDescription(new String[]{
                     "Processed Image"     });
```

```
                Output[1].setConnectionType(INTEGER);
                Output[1].setConnectionMode(NORMAL);
                Output[1].shortDescription = "";
                Output[1].setConnectionDescription(new String[]{
                        "Processed Image Width"       });

                Output[2].setConnectionType(INTEGER);
                Output[2].setConnectionMode(NORMAL);
                Output[2].shortDescription = "";
                Output[2].setConnectionDescription(new String[]{
                        "Processed Image Height"      });
        }

        public void doubleClick()
        {
        }

        public Object create(DataBlock arguments)
        {
                GrayImage argument0 = arguments.getGrayImage(0);
                int argument1 = arguments.getInteger(1);

                DataBlock return0 = pixel_offset(argument0,argument1);
                return(return0);
        }

        private DataBlock pixel_offset(GrayImage inp1,int offset)
        {
                // do not use "width" "height" variables as this conflicts
                // with the block "width" "height" definitions
                int image_width = inp1.getWidth();
                int image_height= inp1.getHeight();

                GrayImage output = new GrayImage(image_width,image_height);

                // wrap variables
                Integer widthw = new Integer(image_width);
                Integer heightw = new Integer(image_height);

                // add offset and divide by two to keep in range.
                for (int y=0; y<image_height; y++)
                        for (int x=0; x<image_width; x++)
                                {output.setxy(x,y,(offset+inp1.getxy(x,y))>>1);}

        // Make the data available to other library functions.
                DataBlock returns = new DataBlock();
                returns.add(output);
                returns.add(widthw);
                returns.add(heightw);
                return(returns);
        }
}
```

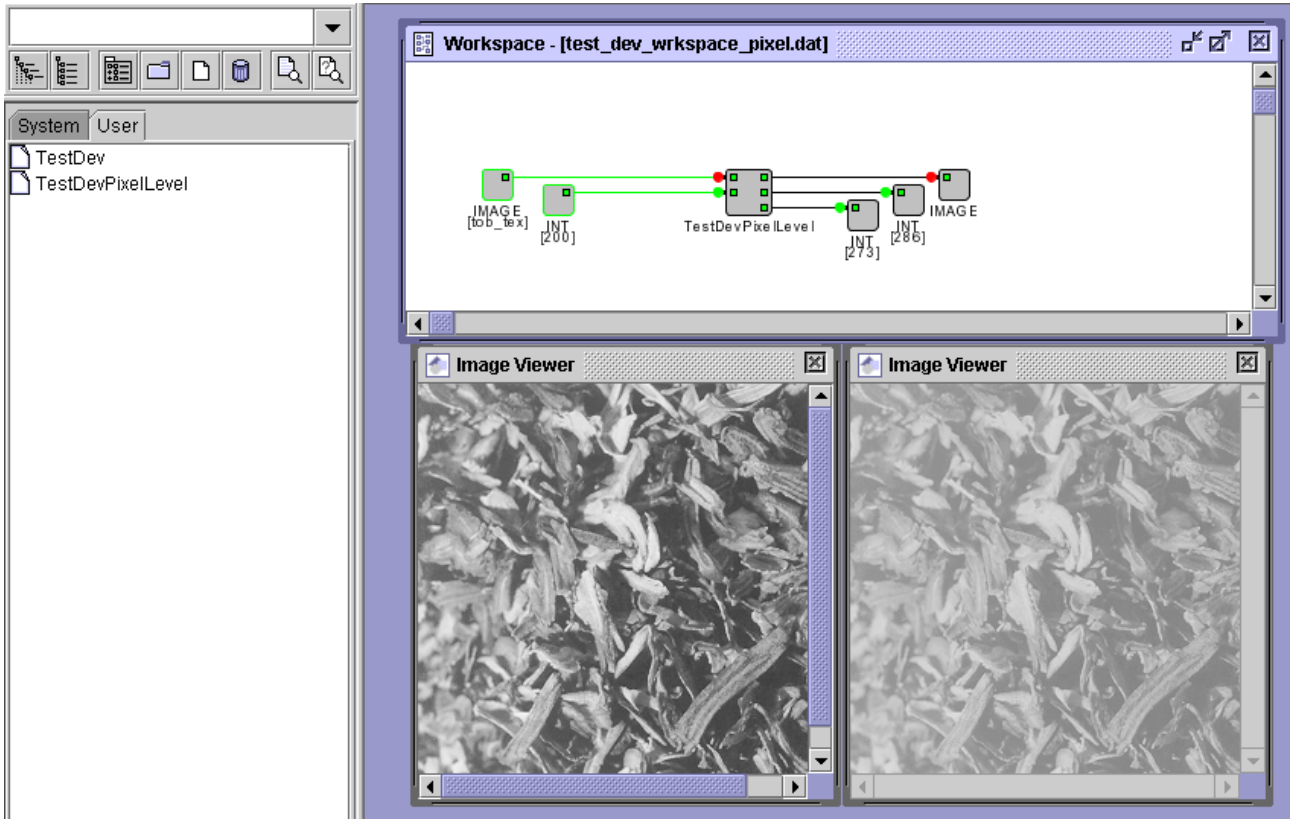


Figure 1. Development of the `TestDevPixelLevel` class. This example illustrates how to manipulate images at the pixel level within the NeatVision environment. The greyscale offset input value has a default setting of 100. This can be overwritten by the user input as illustrated above.

Appendix A.2: conv_test.java

```
// conv_test.java

// Project Name:      NeatVision (Ver 2.0)
// Written by:        Paul Whelan
// Initial Version:   03/03/03
// Latest Revision:
// Description:              Convolution example
//*************************************************************************
// Copyright (C) 2003, Paul F Whelan, Vision Systems Group, DCU
// This library is distributed in WITHOUT ANY WARRANTY; without even the implied warranty
// of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

import DataBlock;
import CoreInterface;

public class conv_test extends CoreInterface
{
       public conv_test()
       {
              name = "conv_test";
              inputs = 1;
              outputs = 1;
              width = 30;
              height = 20;
       }

       public void setup()
       {
              Input[0].setConnectionType(IMAGE);
              Input[0].setConnectionMode(NORMAL);
              Input[0].shortDescription = "";
              Input[0].setConnectionDescription(new String[]{
                     "Input image"         });

              Output[0].setConnectionType(IMAGE);
              Output[0].setConnectionMode(NORMAL);
              Output[0].shortDescription = "";
              Output[0].setConnectionDescription(new String[]{
                     " Output image"       });
       }

       public void doubleClick()
       {

       }
```

```
        public Object create(DataBlock arguments)
        {
        // setup input variables
                GrayImage argument0 = arguments.getGrayImage(0);

        // setup return variables
                DataBlock return0 = conv_test_dev(argument0);
                return(return0);
        }

        private DataBlock conv_test_dev(GrayImage inp1)
        {
        // We must wrap all primitive classes e.g. wrap the integer class
                GrayImage output_a = new GrayImage(width,height);

                Integer [] mask = new Integer[9];

                // mask enteries default to null
                mask[0]= null;                                  // dont care = null
                mask[1]= new Integer(1);
                mask[2]= new Integer(1);
                mask[3]= new Integer(1);
                mask[4]= new Integer(1);
                mask[5]= new Integer(1);
                mask[6]= new Integer(1);
                mask[7]= new Integer(1);
                mask[8]= null;                                  // dont care = null

        // Apply convolution
                output_a = (GrayImage)NeatVision.create("Convolution",inp1, mask);

        // Make the data available to other library functions.
                DataBlock returns = new DataBlock();
                returns.add(output_a);
                return(returns);
        }
}
```

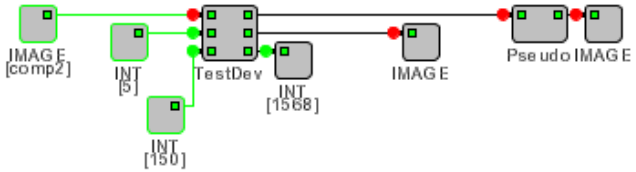
Appendix A.3: `TestDev.java`

```
// TestDev.java

// Project Name:      NeatVision (Ver 2.0)
// Written by:        Paul Whelan
// Initial Version:   03/03/03
// Latest Revision:
// Description:       Sample file to illustrate the NeatVision Development environment
//*************************************************************************
// Copyright (C) 2003, Paul F Whelan, Vision Systems Group, DCU
// This library is distributed in WITHOUT ANY WARRANTY; without even the implied warranty
// of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

import DataBlock;
import CoreInterface;

public class TestDev extends CoreInterface
{
// This is the constructor for the block, the fields below must be filled in.
// This will create a container into which the functionallity of the block can be built.
        public TestDev()
        {
                name = "TestDev";
                inputs = 3;
                outputs = 3;
                width = 30;
                height = 30;
        }

// As the constructor above was used to generate the physical skeleton of the
// block, we must add substance this is done through the addition of three
// methods, the first of which setup() specifies the data type of the input
// and output connectors which were generated in the constructor

        public void setup()
        {
                Input[0].setConnectionType(IMAGE);
                Input[0].setConnectionMode(NORMAL);
                Input[0].shortDescription = "";
                Input[0].setConnectionDescription(new String[]{
                        "Input GS image"      });

                Input[1].setConnectionType(INTEGER);
                Input[1].setConnectionMode(NORMAL);
                Input[1].setDefaultValue(new Integer(4));
                Input[1].shortDescription = "[Integer] Clusters [Default = 4]";
                Input[1].setConnectionDescription(new String[]{
                        "Number of clusters required" });
```

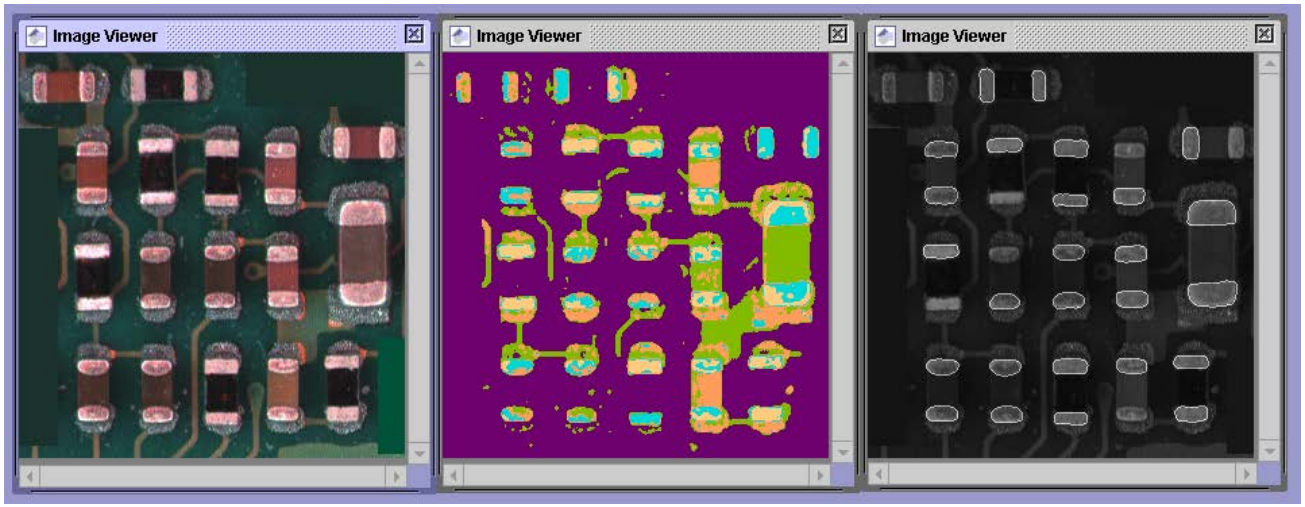
```
                Input[2].setConnectionType(INTEGER);
                Input[2].setConnectionMode(NORMAL);
                Input[2].setDefaultValue(new Integer(150));
                Input[2].shortDescription = "[Integer] Loop count [Default = 150]";
                Input[2].setConnectionDescription(new String[]{
                        "Reconstruction Loop Count"   });

                Output[0].setConnectionType(IMAGE);
                Output[0].setConnectionMode(NORMAL);
                Output[0].shortDescription = "";
                Output[0].setConnectionDescription(new String[]{
                        "Boundary removal and K-Means clustering"    });

                Output[1].setConnectionType(IMAGE);
                Output[1].setConnectionMode(NORMAL);
                Output[1].shortDescription = "";
                Output[1].setConnectionDescription(new String[]{
                        "Closed structures"   });

                Output[2].setConnectionType(INTEGER);
                Output[2].setConnectionMode(NORMAL);
                Output[2].shortDescription = "";
                Output[2].setConnectionDescription(new String[]{
                        "Approx perimeter of the strong edges"       });
        }

// The second method is called whenever a double click event occurs over this block,
// double clicks are useful for generating frames or dialogs for viewing or
// altering images or for entering processing parameters, e.g. thresholding
        public void doubleClick()
        {
        }

// The next method is called whenever the block has new inputs which need to be
// processed. Basically what you need to do inside this method is read the new
// data in from the sockets, process images using the java image processing libraries
// then setting the output image and finally but most importantly  signal the new
// state of the block as being WAITING_TO_SEND, i.e. the image has been sent to
// the plug and awaits transmission to the next block.

        public Object create(DataBlock arguments)
        {
        // setup input variables
                GrayImage argument0 = arguments.getGrayImage(0);
                int argument1 = arguments.getInteger(1);
                int argument2 = arguments.getInteger(2);

        // setup return variables
                DataBlock return0 = blob_test_dev(argument0,argument1,argument2);
                return(return0);
        }
```

```
// The sample program reads in a greyscale image and two integer variables representing
// number of clusters required and the loop count for the reconstruction by dilation
// operation. The block return the k-means clusters after all edge data has been removed
// from the image. It also returns the fitting of closed curves to the strong features
// in the input image. The pixel count of this edge data is also returned.

        private DataBlock blob_test_dev(GrayImage inp1,int clus, int loop_count)
        {
                int inp1_width = inp1.getWidth();
                int inp1_height = inp1.getHeight();

        // We must wrap all primitive classes e.g. wrap the integer class
                Integer clus_wrap = new Integer(clus);
                GrayImage output_a = new GrayImage(inp1_width,inp1_height);
                GrayImage output_b = new GrayImage(inp1_width,inp1_height);

        // remove boundary regions using reconstruction by dilation
                output_a = (GrayImage)NeatVision.create("SingleThreshold",inp1, new Integer(0));
                output_a = (GrayImage)NeatVision.create("Mask",output_a, new Integer(3));
                output_a = (GrayImage)NeatVision.create("Not",output_a);
                output_a = (GrayImage)NeatVision.create("Minimum",inp1, output_a);
                for(int lc=0;lc<loop_count;lc++)
                        {
                        output_a = (GrayImage)NeatVision.create("Dilation",output_a, new Integer(8));
                        output_a = (GrayImage)NeatVision.create("Minimum",inp1, output_a);
                        }
                output_a = (GrayImage)NeatVision.create("Subtract",inp1,output_a);
                output_a = (GrayImage)NeatVision.create("Median",output_a);

        // Apply K-Means clustering
                output_a = (GrayImage)NeatVision.create("GrayScaleCluster",output_a,clus_wrap);

        // Load the canny magnitude image
                DataBlock temp = (DataBlock)NeatVision.create("Canny", inp1, new Double(1.9),new Integer(20),new Integer(230));
                output_b = (GrayImage)temp.getGrayImage(0);
                output_b = (GrayImage)NeatVision.create("SingleThreshold",output_b, new Integer(127));

        // keep only closed structures and find approx perimeter of the strong edges
                output_b = (GrayImage)NeatVision.create("EdgeLabel",output_b,new Boolean(true));
                output_b = (GrayImage)NeatVision.create("SingleThreshold",output_b, new Integer(1));
                Integer area_1 = (Integer)NeatVision.create("WhitePixelCounter",output_b);
                output_b = (GrayImage)NeatVision.create("Add",output_b,inp1);

        // Make the data available to other library functions.
                DataBlock returns = new DataBlock();
                returns.add(output_a);
                returns.add(output_b);
                returns.add(area_1);
                return(returns);
        }
}
```

(a)



(b)



(c)

Figure 2. Development of the `TestDev` class. (a) The associated class file tag in the user area. (b) A sample program illustrating the operation of this block. (c) Sample images (left to right): Input image, K-Mean clustered image and the closed structure overlay image.

Appendix B

Appendix B: Main NeatVision 2.1 Methods

The following list summarises some of the main NeatVision methods users may wish to interface too. Many of these are fairly self-explanatory, but if the method you require is not listed or does not have enough information to enable you to use it drop us an email at *tech@neatvision.com* with "NeatVision Methods" in the subject line. Additional help can be found in the input/output tags for each block in the NeatVision visual programming interface.. Also refer to *P.F. Whelan and D. Molloy (2000),* ***Machine Vision Algorithms in Java: Techniques and Implementation****, Springer (London), 298 Pages [ISBN 1-85233-218-2]* for additional details.

Normalization of greyscale image operations occurs to keep the output image within greyscale range 0-255.

Method	Description	Inputs (Index #: data type [descriptor])	Outputs (Index #: data type [descriptor])
DATA	Image, Integer, Double, Boolean, String, Array (of integers) and 3D (DICOM, Analyze, Vol, Sequence)		
FLOW CONTROL	SplitterX2,SplitterX3, SplitterX4, Feedback, If, Else, For and Terminate		
UTILITIES			
HalveImageSize	A grey-scale image whose size is halved	0:GrayImage	0:GrayImage
DoubleImageSize	A grey-scale image whose size is doubled	0:GrayImage	0:GrayImage
PointToSquare	A grey-scale image whose white pixels are represented by white squares.	0:GrayImage	0:GrayImage
PointToCross	A grey-scale image whose white pixels are represented by white crosses.	0:GrayImage	0:GrayImage
Rotate	A grey-scale image is rotated in a clockwise direction by a user specified amount	0:GrayImage 1: Integer [user specified rotation (degrees)]	0:GrayImage
RotatePlus90	A grey-scale image is rotated in a clockwise direction by 90 degrees	0:GrayImage	0:GrayImage
RotateMinus90	A grey-scale image is rotated in an anticlockwise direction by 90 degrees.	0:GrayImage	0:GrayImage
ROI[2]	A grey-scale image from which a rectangular region of interest is extracted by the user via the GUI.	0:GrayImage	0:GrayImage
PolyROI[3]	A grey-scale image from which a polygon region of interest is extracted by the user via the GUI.	0:GrayImage [User interaction]	0:GrayImage

[2] **Left click and hold** to draw the ROI, then release when complete.

[3] The user inputs a polygon by **left-clicking** a series of points (marked in red). When the user clicks a point within 4 pixels of the start point or alternatively **right-click** to finalize and close the polygon. Once closed the polygon will be displayed in green. To begin a new polygon use **shift-click**.

EnhancePolyROI[2]	A grey-scale image from which a polygon region of interest shall be emphasised. User defined input region.	0:GrayImage [User interaction]	0:GrayImage
Measure_Line	An image from which the Euclidean distance between two user-selected points is calculated. Must rerun programme to generate new line length.	0:GrayImage [User interaction]	0:Double [Euclidean distance]
Scale	A grey-scale image is scaled by user defined dimensions	0:GrayImage 1: Integer [width of the scaled image] 2: Integer [height of the scaled image]	0:GrayImage
Mask	A grey-scale image whose border is masked by a user specified amount.	0:GrayImage 1: Integer [Mask size in pixels, Default =3]	0:GrayImage
Centroid	Replace the greyscale shapes (Range 0-255) in the original image by their respective centroids (commonly used after the 8-bit labelling operators)	0:GrayImage	0:GrayImage [Binary]
Centroid_16	Replace the greyscale shapes (Range 0-65535) in the original image by their respective centroids (commonly used after the Label_16 operators)	0:GrayImage	0:GrayImage [Binary]
BinaryToGreyscale	Convert WHITE pixels in a binary image to a given greyscale.	0:GrayImage [Binary] 1:Integer [greyscale (0-255)]	0:GrayImage
GreyScalePixelSum	Generates an integer which is the sum of all pixels contained in the input image	0:GrayImage	0:Integer
FirstWhitePixelLocator	Coordinate point representing the location of the first white pixel in the image input image.	0:GrayImage	0:Coordinate
RemoveIsolatedWhitePixels	Remove isolate white pixels (3x3) region)	0:GrayImage	0:GrayImage
SaltnPepperGenerator	Add salt and pepper noise to the input image	0:GrayImage 1:Double (0-1.0)	0:GrayImage
AdditiveWhiteNoiseGenerator	Add a user defined level of white noise to the input image	0:GrayImage 1:Integer (0-1024)	0:GrayImage
GaussianNoiseGenerator	Add a user defined quantity of Gaussian noise to the input image	0:GrayImage 1:Double (0-255.0)	0:GrayImage
RayleighNoiseGenerator	Add a user defined quantity of Rayleigh noise to the input image	0:GrayImage 1:Double (1.0-255.0)	0:GrayImage
PoissonNoiseGenerator	Add a user defined quantity of Poisson noise to the input image	0:GrayImage 1:Double (0-511.0)	0:GrayImage

HTTPSendScript	Send arguments to a URL	0:String [URL] 1:String [Arguments]	0:String [Return values]
HTTPGetImage	Retrieve image from a URL	0:String [URL] 1:String [Arguments]	0:GrayImage [Retrieved Image]
ARITHIMETIC			
Add	Image addition	0:GrayImage [A] 1:GrayImage [B]	0:GrayImage [C = A+B]
Subtract	Image subtraction	0:GrayImage [A] 1:GrayImage [B]	0:GrayImage [C = A-B]
Multiply	Image multiply	0:GrayImage [A] 1:GrayImage [B]	0:GrayImage [C = A*B]
Divide	Image division	0:GrayImage [A] 1:GrayImage [B]	0:GrayImage [C = A/B]
And	Boolean AND operation	0:GrayImage [A] 1:GrayImage [B]	0:GrayImage [C = AND(A,B)]
Or	Boolean OR operation	0:GrayImage [A] 1:GrayImage [B]	0:GrayImage [C = OR(A,B)]
Not	Boolean NOT operation	0:GrayImage [A]	0:GrayImage [C = NOT(A)]
Xor	Boolean Exclusive OR operation	0:GrayImage [A] 1:GrayImage [B]	0:GrayImage [C = XOR(A,B)]
Minimum	Minimum of two images	0:GrayImage [A] 1:GrayImage [B]	0:GrayImage [C = Min(A,B)]
Maximum	Maximum of two images	0:GrayImage [A] 1:GrayImage [B]	0:GrayImage [C = Max(A,B)]
HISTOGRAM			
HighestGreyLevelCalculator	Compute the highest grey level from the input image	0:GrayImage	0:Integer [highest grey level]
LowestGreyLevelCalculator	Compute the lowest grey level from the input image	0:GrayImage	0:Integer [lowest grey level]
MeanSquareError	Compare the input images using the mean square error operation	0:GrayImage 1:GrayImage	0:Double [mean square error]
AverageIntensityCalculator	Compute the average intensity of the input image	0:GrayImage	0:Double [average intensity]
EntropyCalculator	Compute the entropy of the input image	0:GrayImage	0:Double [entropy]
VarienceCalculator	Compute the variance of the input image	0:GrayImage	0:Double [varience]
KurtosisCalculator	Compute the kurtosis of the input image	0:GrayImage	0:Double [kurtosis]
StandardDeviationCalculator	Compute the standard deviation of the input image	0:GrayImage	0:Double [standard deviation]
SkewnessCalculator	Compute the skewness deviation of the input image	0:GrayImage	0:Double [skewness]

LocalEqualisation3x3	Local histogram equalisation using a 3x3 region	0:GrayImage	0:GrayImage
LocalEqualisation5x5	Local histogram equalisation using a 5x5 region	0:GrayImage	0:GrayImage
PROCESSING			
Inverse	Inverse the LUT of the input image	0:GrayImage	0:GrayImage
Logarithm	Transform the linear LUT into logarithmic	0:GrayImage	0:GrayImage
Exponential	Transform the linear LUT into exponential	0:GrayImage	0:GrayImage
Power	The linear LUT is raised to a user specified double value	0:GrayImage 1:Integer [power, default=3.0]	0:GrayImage
Square	The linear LUT is raised to power of 2.	0:GrayImage	0:GrayImage
SingleThreshold	Single threshold operation	0:GrayImage 1:Integer [(1-255): Default = 128]	0:GrayImage [Binary]
MidlevelThreshold	Single threshold operation: threshold level = MIDGREY (127)	0:GrayImage	0:GrayImage [Binary]
DualThreshold	Dual threshold operation. All pixels between the upper and lower thresholds are marked in WHITE.	0:GrayImage 1:Integer [upper value, default =128] 2:Integer [lower value, default =1]	0:GrayImage [Binary]
TripleThreshold	This operation produces an LUT in which all pixels below the user specified lower level appear black, all pixels between the user specified lower level and the user specified upper level inclusively appear grey and all pixels above the user specified upper level appear white.	0:GrayImage 1:Integer [upper value, default =128] 2:Integer [lower value, default =1]	0:GrayImage
EntropicThreshold	Compute the entropy based threshold. Relies on maximising the total entropy of both the object and background regions to find the appropriate threshold	0:GrayImage	0:Integer
Threshold3x3	Adaptive threshold in a 3x3 region.	0:GrayImage 1:Integer [constant offset, default=0]]	0:GrayImage
Threshold5x5	Adaptive threshold in a 5x5 region.	0:GrayImage 1:Integer [constant offset, default=0]]	0:GrayImage
IntensityRangeEnhancer	Stretch the LUT in order to occupy the entire range between BLACK (0) and WHITE (255)	0:GrayImage	0:GrayImage
HistorgramEqualiser	Histogram equalisation	0:GrayImage	0:GrayImage

IntensityRangeStrecher	Stretch the LUT between the lower and upper threshold to occupy the entire range between BLACK (0) and WHITE (255)	0:GrayImage 1:Integer [lower grey level, default=0] 2:Integer [upper grey level, default=255]	0:GrayImage
IntegrateImageRows	Integrate image rows	0:GrayImage	0:GrayImage
IntegrateImageColumns	Integrate Image columns	0:GrayImage	0:GrayImage
LeftToRightSum	Pixel summation along the line	0:GrayImage	0:GrayImage
LeftToRightWashFunction	Left To Right wash function (once a white pixel is found, all pixels to its right are also set to white)	0:GrayImage	0:GrayImage
RightToLeftWashFunction	Right To Left wash function (once a white pixel is found, all pixels to its left are also set to white)	0:GrayImage	0:GrayImage
TopToBottomWashFunction	Top To Bottom wash function (once a white pixel is found, all pixels to its below are also set to white)	0:GrayImage	0:GrayImage
BottomToTopWashFunction	Bottom To Top wash function (once a white pixel is found, all pixels to its above are also set to white)	0:GrayImage	0:GrayImage
FILTER			
Convolution	Convolution. This operation requires coefficients to be specified in the form of a square, odd sized integer array, “null” represents “don’t cares”. See Appendix A.2 for an example.	0:GrayImage 1:Integer [] [Array of mask values. No entry default to null. “Don’t Care” = null statement]	0:GrayImage
DOLPS	DOLPS – Difference of low pass 3x3 filters. Image A results from applying 3 iterations of the low pass filter. Image B results from applying 6 iterations of the low pass filter. DOLPS = A-B.	0:GrayImage	0:GrayImage
LowPass	Low pass 3x3 filter	0:GrayImage	0:GrayImage
Sharpen	High pass 3x3 filter	0:GrayImage	0:GrayImage
Median	Median 3x3 filter	0:GrayImage	0:GrayImage
Midpoint	Midpoint 3x3 filter	0:GrayImage	0:GrayImage
RectangularAverageFilter	Rectangular Average Filter operation. Size of filter is user defined	0:GrayImage 1:Integer [filter size, default = 5]	0:GrayImage
SmallestIntensityNeighbour	Replace the central pixel of the 3x3 mask with the minimum value	0:GrayImage	0:GrayImage
LargestIntensityNeighbour	Replace the central pixel of the 3x3 mask with the maximum value	0:GrayImage	0:GrayImage

AdaptiveSmooth	Adaptive smoothing of grey scale images. In order to apply it to colour images, the input image has to be split into RGB components and adaptive smooth has to be applied to each channel. If the colour image is applied directly the algorithm will smooth the average intensity image. (Slow process)	0:GrayImage 1:Integer [number of iterations: possible values: 1 to 10, default = 5] 2:Double [variance strength: possible values: 0.1 -> 0.9, default = 0.2] 3:Double [Diffusion parameter: possible values: 1.0 -> 20.0, default = 10.0]	0:GrayImage
EDGES			
Roberts	Roberts edge detector	0:GrayImage	0:GrayImage
Sobel	Sobel edge detector	0:GrayImage	0:GrayImage
Laplacian	Laplacian edge detector. User defined 4-connected or 8-connected neighbourhood	0:GrayImage 1:Integer [possible values: 4 or 8, default = 8]	0:GrayImage
Prewitt	Prewitt edge detector	0:GrayImage	0:GrayImage
FreiChen	FreiChen edge detector	0:GrayImage	0:GrayImage
BinaryBorder	Binary Border edge detector	0:GrayImage [Binary]	0:GrayImage [Binary]
NonMaxima	Edge detection using non maxima suppression	0:GrayImage	0:GrayImage
IntensityGradientDirection	Compute the 3x3 intensity gradient direction. Gradients range from 1 to 8.	0:GrayImage	0:GrayImage [pixel values from 1-8]
ZeroCrossingsDetector	Zero crossings edge detector	0:GrayImage	0:GrayImage
Canny	Canny edge detector	0:GrayImage 1:Double [standard deviation or spread parameter, possible values: 0.2 -> 20.0, default = 1.0] 2:Integer [lower threshold, default = 1] 3:Integer [upper threshold, default = 255]	0:GrayImage [edge magnitudes] 1:GrayImage [edge directions]
EdgeLabel	Edge labelling operation. Expects a binary image resulting from the application of the Canny edge detector.	0:GrayImage 1:Boolean [Set True if you want closed structures]	0:GrayImage [A binary image whose edge pixels are grouped into polygonal shapes]
LineFitting	Line fitting in the edge structure. Expects a binary image resulting from the application of the Canny edge detector.	0:GrayImage 1:Boolean [Set True if you want closed structures]	0:GrayImage [A binary image whose edge pixels are grouped into polygonal shapes]

ArcFitting	Arc fitting in the edge structure. Expects a binary image resulting from the application of the Canny edge detector.	0:GrayImage 1:Boolean [Set True if you want closed structures] 2:Boolean [Set True if you want display the circles associated with detected arcs] 3:Boolean [Set True if you want display the lines that are not grouped into arcs segments]	0:GrayImage [A binary image whose edge pixels are grouped into polygonal shapes]
EdgeLinking[4]	Edge linking (scanning window is user defined). Expects a binary image resulting from the application of the Canny edge detector.	0:GrayImage 1:Integer [The size of scanning window. (5-11)]	0:GrayImage [Edge linked image]
ANALYSIS			
ThinOnce	Full application of the thinning algorithm. Thin *till completion* resulting in a skeleton image.	0:GrayImage [Binary]	0:GrayImage [Binary]
Thin	The input binary image is thinned N times as specified by the user	0:GrayImage [Binary] 1:Integer [N – number of iterations]	0:GrayImage [Binary]
CornerPointDetector	Skeleton corner detection from a binary image based on a 3x3 region	0:GrayImage [Binary]	0:GrayImage [Binary]
JunctionDetector	Skeleton junction detection from a binary image based on a 3x3 region	0:GrayImage [Binary]	0:GrayImage [Binary]
LimbEndDetector	Skeleton limb end detection from a binary image based on a 3x3 region	0:GrayImage [Binary]	0:GrayImage [Binary]
BiggestBlob	Extract the biggest white blob from a binary image	0:GrayImage [Binary]	0:GrayImage [Binary]
SmallestBlob	Extract the smallest white blob from a binary image	0:GrayImage [Binary]	0:GrayImage [Binary]
BlobFill	Fill the holes in a binary image	0:GrayImage [Binary]	0:GrayImage [Binary]
Labeller	Label by location unconnected shapes in a binary image (Range 0-255)	0:GrayImage [Binary]	0:GrayImage
LabelByArea	Label the unconnected shapes in a binary image in relation to their size (Range 0-255)	0:GrayImage [Binary]	0:GrayImage
MeasureLabelledObjects	Measure the N (user specified) largest objects in a binary image (Range 0-255)	0:GrayImage [Binary] 1:Integer [limit on the number of labelled objects measured, default=5]	0:String [contains data describing the measured objects: (Grey Scale, Area, Centroid)]

[4] O. Ghita and P.F. Whelan (2002), “A computationally efficient method for edge thinning and linking using endpoints”, **Journal of Electronic Imaging**, 11(4), Oct. 2002, pp 479-485.

WhiteBlobCount	Count the number of white bobs in a binary image (Range 0-255)	0:GrayImage [Binary]	0:Integer [Range 0-255] 1:GrayImage [A white cross is overlaid on each blob found.]
Label_16	Label by location the unconnected shapes in a binary image (Range 0-65535). Note: This is outside the 8-bit display range. Slow process.	0:GrayImage [Binary]	0:GrayImage
WhiteBlobCount_16	Count the number of white bobs in a binary image (Range 0-65535). Slow process.	0:GrayImage [Binary]	0:Integer [Range 0-65535] 1:GrayImage [A white cross is overlaid on each blob found.]
ConvexHull	Compute the convex hull boundary	0:GrayImage [Binary]	0:GrayImage [Binary]
FilledConvexHull	Compute the filled convex hull	0:GrayImage [Binary]	0:GrayImage [Binary]
CrackDetector	Highlight cracks in the input image	0:GrayImage	0:GrayImage
EulerNumberCalculator	Compute the Euler number from a binary image	0:GrayImage [Binary]	0:Integer [Euler number]
WhitePixelCounter	Compute the number of white pixels	0:GrayImage	0:Integer [pixel count]
IsolateHoles	Isolate holes in a binary image	0:GrayImage [Binary]	0:GrayImage [Binary]
IsolateBays	Isolate bays in a binary image	0:GrayImage [Binary]	0:GrayImage [Binary]
ConnectivityDetector	Connectivity detection in a thinned skeleton binary image. Mark points critical for connectivity in a 3x3 region.	0:GrayImage [Binary]	0:GrayImage
BoundingBox	Minimum area bounding rectangle	0:GrayImage	0:GrayImage
FilledBoundingBox	Filled minimum area bounding rectangle	0:GrayImage	0:GrayImage
BoundingBoxTopCoordinate	Compute the top left coordinate of the minimum area bounding rectangle	0:GrayImage	0:Coordinate [top left]
BoundingBoxBottomCoordinate	Compute the bottom right coordinate of the minimum area bounding rectangle	0:GrayImage	0:Coordinate [bottom right]
CornerDetector	Grey Scale (SUSAN) corner detector	0:GrayImage 1:Integer [Brightness threshold] 2:Integer [Geometric threshold]	0:GrayImage [Corner points]
K-MEANS CLUSTERING			
GrayScaleCluster	Cluster a grey scale image (number of clusters are user defined) using the k-means algorithm.	0:GrayImage 1:Integer [Number of clusters]	0:GrayImage [Gray-scale]
ColorCluster	Cluster a colour image (number of clusters are user defined) using the k-means algorithm.	0:Image [Color Image Input] 1:Integer [Number of clusters]	0:GrayImage [Gray-scale]

Un_ColorCluster	Unsupervised colour clustering using the k-means algorithm.	0:Image 1:Double [Low threshold (possible values 0.5-1.0), default=0.6] 2:Double [High threshold (possible values 1.0-1.5), default=1.2]	0:GrayImage [Gray-scale] 1:Image [Colour] 2:Integer [Number of clusters]
PseudoColor	Pseudo-colour operation	0:Image [grey-scale or colour image]	0:Image [false colour image]
TRANSFORM[#]			
MedialAxisTransform	Medial axis transform operation. Binary image showing the simple skeleton	0:Image [binary]	0:Image [binary]
MedialAxisTransform_GS	Medial axis transform operation. GS image where each point on the skeleton has an intensity which represents its distance to a boundary in the original object	0:Image [binary]	0:GrayImage [grey scale]
FFT	Fast Fourier Transform: FFT	0:GrayImage [Input image dimension must be a power of 2]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]
IFFT	Inverse Fourier Transform	0:File [A Fourier data file which shall be interpreted as an image.]	0:GrayImage [The resulting gray-scale image which represents the interpreted Fourier data]
FFTLowpass	Low pass frequency filter	0:File [Fourier Data File] 1:Double [cut-off value (0-1.0)]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]
FFTHighpass	High pass frequency filter	0:File 1:Double [cut-off value (0-1.0)]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]
FFTAdaptiveLowpass	FFT adaptive lowpass filter	0:File 1:Double [limit (0-1.0)]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]

[#] Some of these functions use data types / variables that are for internal NeatVision use **only**. Access to such data (e.g. pixel access) is can be done directly in Java, see example in Appendix A.1

FFTBandpass	FFT band-pass filter	0:File [Fourier Data File] 1:Double [inner limit (0-1.0)] 2:Double [outer limit (0-1.0)]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]
FFTBandstop	FFT band-stop filter	0:File [Fourier Data File] 1:Double [inner limit (0-1.0)] 2:Double [outer limit (0-1.0)]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]
FFTMultiply	Multiply two Fourier data files	0:File [Fourier Data File] 1:File [Fourier Data File]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]
FFTDivide	Divide one Fourier data file by another	0:File [Fourier Data File] 1:File [Fourier Data File]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]
FFTGaussian	FFT Gaussian filter. Input 0 requires an integer value that = 2^n where n is a +ve integer. Note: size = width = height	0:Integer [size of a new Fourier data file which contains Gaussian coefficients] 1:Double [Standard deviation of the Gaussian coefficients (0.1-5.0)]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]
FFTSelectivePass	FFT selective frequency filter	0:File [Fourier Data File] 1:Double [The cutoff value of the filter (0-1.0)] 2:Double [The x-offset of the symmetric selective filter (0-1.0)] 3:Double [The y-offset of the symmetric selective filter (0-1.0)]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]
FFTSymmetricSelectivePass	FFT selective symmetric frequency filter	0:File [Fourier Data File] 1:Double [The cutoff value of the filter (0-1.0)] 2:Double [The x-offset of the symmetric selective filter (0-1.0)] 3:Double [The y-offset of the symmetric selective filter (0-1.0)]	0:File [Fourier Data File] 1:GrayImage [Grey-scale image transformed to its Fourier coefficients]

DCT2D	Direct Cosine Transform operation	0:GrayImage [Input image dimension must be a power of 2]	0:GrayImage [Real Part] 1:GrayImage [DCT Magnitude]
IDCT2D	Inverse DCT (filtering factor is user defined)	0:GrayImage 1:Double [DCT quality coefficient (0-2.0)]	0:GrayImage [IDCT image]
Hough	Line Hough Transform	0:GrayImage [Binary]	0:GrayImage
InverseHough	Inverse Hough Transform. The integer input specifies how many of the brightest pixels shall be taken into account when performing the Inverse Hough operation.	0:GrayImage 1:Integer [Number of bright points to be considered, default=10]	0:GrayImage
CircHough	Circular Hough Transform	0:GrayImage [binary image to be subjected to the circular Hough transform]	0:GrayImage [Image] 1:GrayImage [Transform space]
CooccurrenceMatrixGenerator	Compute the co-occurrence matrix	0:GrayImage	0:GrayImage
CooccurrenceMatrixEnergyCalculator	Compute the energy of the co-occurrence matrix	0:GrayImage	0:Double
CooccurrenceMatrixEntropyCalculator	Compute the entropy of the co-occurrence matrix	0:GrayImage	0:Double
CooccurrenceMatrixContrastCalculator	Compute the contrast of the co-occurrence matrix	0:GrayImage	0:Double
CooccurrenceMatrixHomogenityCalculator	Compute the homogeneity of the co-occurrence matrix	0:GrayImage	0:Double
DistanceTransform3x3	Compute the distance transform in a 3x3 window (input binary image)	0:GrayImage [Binary]	0:GrayImage
DistanceTransform5x5	Compute the distance transform in a 5x5 window (input binary image)	0:GrayImage [Binary]	0:GrayImage
LeftToRightDistanceTransform	Left to right distance transform (input binary image)	0:GrayImage [Binary]	0:GrayImage
RightToLeftDistanceTransform	Right to left distance transform (input binary image)	0:GrayImage [Binary]	0:GrayImage
TopToBottomDistanceTransform	Top to bottom distance transform (input binary image)	0:GrayImage [Binary]	0:GrayImage
BottomToTopDistanceTransform	Bottom to top distance transform (input binary image)	0:GrayImage [Binary]	0:GrayImage
GrassFireTransform	Grass fire transform (input binary image) [8-connected]	0:Image [Binary]	0:Image [grey-scale]
MORPHOLOGY			
Dilation	Dilation operation (user specify connectivity of the structured element 4 or 8)	0:GrayImage 1:Integer [(4 or 8), default=8]	0:GrayImage

Erosion	Erosion operation (user specify connectivity of the structured element 4 or 8)	0:GrayImage 1:Integer [(4 or 8), default=8]	0:GrayImage
Open	Opening operation (user specify connectivity of the structured element 4 or 8)	0:GrayImage 1:Integer [(4 or 8), default=8]	0:GrayImage
Close	Closing operation (user specify connectivity of the structured element 4 or 8)	0:GrayImage 1:Integer [(4 or 8), default=8]	0:GrayImage
ErodeNxN	Erosion operation with a user defined NxN structuring element (X or null = don't cares)	0:GrayImage 1:Integer [Array]	0:GrayImage
DilateNxN	Dilation operation with a user defined NxN structuring element (X or null = don't cares)	0:GrayImage 1:Integer [Array]	0:GrayImage
MorphologicalValley	Morphological valley operation (user specify connectivity of the structured element 4 or 8) [Default=8]	0:GrayImage 1:Integer (4 or 8)	0:GrayImage
MorphologicalTophat	Morphological top hat operation (user specify connectivity of the structured element 4 or 8) [Default=8]	0:GrayImage 1:Integer (4 or 8)	0:GrayImage
HitAndMiss	Hit and miss transformation. Hit and miss array masks must not overlap.	0:GrayImage 1:Integer [user defined hit array, blanks correspond to DON'T CARE)] 2:Integer [user defined miss array]	0:GrayImage
MorphGradient	Morphological Gradient (user specify connectivity of the structured element 4 or 8) [Default=8]	0:GrayImage 1:Integer	0:GrayImage
ReconByDil	Reconstruction by dilation	0:GrayImage 1:GrayImage [Seed] 2:Integer [SE size]	0:GrayImage [Reconstructed] 1:GrayImage [Elements removed]
ReconByDil_UI	Reconstruction by dilation via a user selected seed point (8-connected).	0:GrayImage [User interaction]	0:GrayImage [Reconstructed] 1:GrayImage [Elements removed]
DBLT	Double [Hysteresis] Threshold based reconstruction. Binary Outputs. Seed threshold to reduce noise Mask threshold to maximise signal	0:GrayImage 1:Integer [seed threshold] 2:Integer [mask threshold]	0:GrayImage [Reconstructed] 1:GrayImage [Seed Image] 2:GrayImage [Seed Image]
Watershed	Watershed transform (return the watershed image and the region boundaries image)	0:GrayImage	0:GrayImage [Watershed Image] 1:GrayImage [Binary, Watershed boundaries]
COLOUR			
GreyScaler	Average three colour planes	0:Image [colour]	0:GrayImage

ColourToRGB	Extract the RGB color planes	0:Image [colour]	0:GrayImage [R] 1:GrayImage [G] 2:GrayImage [B]
RGBToColour	Create an image from individual RGB channels	0:GrayImage [R] 1:GrayImage [G] 2:GrayImage [B]	0:Image [colour]
ColourToHSI	Extract the HSI colour planes	0:Image [colour]	0:GrayImage [H] 1:GrayImage [S] 2:GrayImage [I]
HSIToColour	Create an image from individual HSI planes	0:GrayImage [H] 1:GrayImage [S] 2:GrayImage [I]	0:Image [colour]
ColourToOpponent	Extract the opponent process colour representation	0:Image [colour]	0:GrayImage [Red_Green] 1:GrayImage [Blue_Yellow] 2:GrayImage [White_Black]
ViewOpponent	Normalize (0-255) opponent process colour channels. Used to view the normalized colour (unsaturated) channels	0:GrayImage [Red_Green] 1:GrayImage [Blue_Yellow] 2:GrayImage [White_Black]	0:GrayImage [Red_Green] 1:GrayImage [Blue_Yellow] 2:GrayImage [White_Black]
ColourToCMY	Extract the CMY (Cyan, Magenta, Yellow) colour planes	0:Image [colour]	0:GrayImage [C] 1:GrayImage [M] 2:GrayImage [Y]
CMYToColour	Create an image from individual CMY (Cyan, Magenta, Yellow) planes	0:GrayImage [C] 1:GrayImage [M] 2:GrayImage [Y]	0:Image [colour]
ViewCMY	Normalize (0-255) CMY channels. Used to view the normalized colour (unsaturated) channels	0:GrayImage [C] 1:GrayImage [M] 2:GrayImage [Y]	0:GrayImage [C] 1:GrayImage [M] 2:GrayImage [Y]
ColourToYUV	Extract the YUV colour planes	0:Image [colour]	0:GrayImage [Y] 1:GrayImage [U] 2:GrayImage [V]
YUVToColour	Create an image from individual YUV planes	0:GrayImage [Y] 1:GrayImage [U] 2:GrayImage [V]	0:Image [colour]
ViewYUV	Normalize (0-255) YUV channels. Used to view the normalized colour (unsaturated) channels	0:GrayImage [Y] 1:GrayImage [U] 2:GrayImage [V]	0:GrayImage [Y] 1:GrayImage [U] 2:GrayImage [V]
ColourToYIQ	Extract the YIQ colour planes.	0:Image [colour]	0:GrayImage [Y] 1:GrayImage [I] 2:GrayImage [Q]

YIQToColour	Create an image from individual YIQ planes	0:GrayImage [Y] 1:GrayImage [I] 2:GrayImage [Q]	0:Image [colour]
ViewYIQ	Normalize (0-255) YIQ channels. Used to view the normalized colour (unsaturated) channels	0:GrayImage [Y] 1:GrayImage [I] 2:GrayImage [Q]	0:GrayImage [Y] 1:GrayImage [I] 2:GrayImage [Q]
ColourToXYZ	Extract the XYZ colour planes	0:Image [colour]	0:GrayImage [X] 1:GrayImage [Y] 2:GrayImage [Z]
XYZToColour	Create an image from individual XYZ planes	0:GrayImage [X] 1:GrayImage [Y] 2:GrayImage [Z]	0:Image [colour]
ViewXYZ	Normalize (0-255) XYZ channels. Used to view the normalized colour (unsaturated) channels	0:GrayImage [X] 1:GrayImage [Y] 2:GrayImage [Z]	0:GrayImage [X] 1:GrayImage [Y] 2:GrayImage [Z]
ColourToLAB	Extract the Lab colour planes.	0:Image [colour]	0:GrayImage [L] 1:GrayImage [a] 2:GrayImage [b]
LABToColour	Create an image from individual Lab planes	0:GrayImage [L] 1:GrayImage [a] 2:GrayImage [b]	0:Image [colour]
ViewLAB	Normalize (0-255) Lab channels. Used to view the normalized colour (unsaturated) channels.	0:GrayImage [L] 1:GrayImage [a] 2:GrayImage [b]	0:GrayImage [L] 1:GrayImage [a] 2:GrayImage [b]
3D VOLUME			
DicomSave	A grey-scale volume image whose pixels shall be saved into DICOM format (*.dcm) (Double-click to activate). Requires the DICOM header file generated by *DicomRead*.	0:VolumeImage 1:String [Path of the original DICOM header file]	
DicomRead	Extract the grey-scale volume image data from a DICOM image. The header information is also made available to be passed to *DicomSave*		0:VolumeImage 1:String [Path of the original DICOM header file] 2:String [DICOM header]
XYZviewer	Slices in a grey-scale 3D image are viewed from their X, Y and Z directions	0:VolumeImage	
IMGfrom3D	Get a slice from a 3D data set (slice is specified by user). Returns the min max pixel values from within the slice.	0:VolumeImage 1:Integer [Range: 1 to the number of slices in the volume, default=1]	0:GrayImage 1:Integer [minimum pixel value within slice] 2:Integer [maximum value within slice]

Scale3dData	Scale pixel values in a 3D grey-scale image to the range 0 – integer input	0:VolumeImage 1:Integer [Scale range required, (default=255)]	0:VolumeImage
Thres3D	Threshold the 3D data	0:VolumeImage [Grey-scale] 1:Integer [Threshold value, default=200]	0:VolumeImage [Binary]
Mask3D	Generate a 3D mask. Zeros a user-defined number of rows, columns and slices.	0:VolumeImage [Grey-scale] 1:Integer [size of 3D mask, default=1]	0:VolumeImage [Grey-scale]
Sobel3D	3D Sobel 3x3x1 (18-neighbourhood) edge detector	0:VolumeImage [Grey-scale]	0:VolumeImage [Grey-scale]
Blob3D	Extract the 3D blobs from binary 3D image. Each blob is assigned a grey scale value.	0:VolumeImage [Binary]	0:VolumeImage [Binary]
BigestBlob3D	Extract the N (user defined) biggest 3D blobs from 3D binary image.	0:VolumeImage [Binary] 1:Integer [Number of large blobs required, range 0-255, default=1]	0:VolumeImage [Binary]
Thinning3D	3D thinning operation of a binary 3D data set	0:VolumeImage [Binary]	0:VolumeImage [Binary]
MIP	Maximum intensity projection transform	0:VolumeImage	0:GrayImage
AIP	Average intensity projection transform	0:VolumeImage	0:GrayImage
PushSlice	Push (insert) an image slice into the 3D data set	0:VolumeImage 1: GrayImage [Image to be inserted] 2:Integer [slice number - between 1 and depth (default=1)] 3:Integer [minimum pixel value within slice (default=1)] 4:Integer [maximum pixel value within slice (default=255)]	0:VolumeImage

RenderEngine	Surface rendering of a binary image. Image can be displayed as a cloud of points, wire frame, flat shading, Gouraund shading and Phong shading. (Double-click to activate). Allows user to translate, scale and rotate image.	0:VolumeImage	0:VolumeImage
LOW LEVEL[#]			
GetPixel	A grey-scale image from which a pixel intensity at a certain coordinate is obtained. **NB:** See Appendix A.1 for a more efficient way to directly manipulate pixel data. NeatVision low level methods are not recommend for such low level pixel operations.	0:GrayImage 1: Coordinate [coordinate of the pixel in question]	0: Integer [intensity of the pixel at the specified coordinate]
SetPixel	A grey-scale image from which a pixel at a certain coordinate is replaced with one of a user defined intensity.	0:GrayImage 1: Integer [grey-scale intensity of the replacement pixel] 2: Coordinate [coordinate of the pixel in question]	0:GrayImage
RemovePixel	A grey-scale image from which a pixel at a certain coordinate is removed (removing a pixels sets that pixel to black).	0:GrayImage 1: Coordinate [coordinate of the pixel in question]	0:GrayImage
DrawLine	Draw a line in the grey-scale image	0:GrayImage 1: Coordinate [starting coordinate of the line] 2: Coordinate [finishing coordinate of the line] 3: Integer [gray-scale intensity of the line]	0:GrayImage
DrawBox	Draw a hollow box in the grey-scale image	0:GrayImage 1: Coordinate [upper top left] 2: Coordinate [lower bottom right] 3: Integer [grey-scale intensity]	0:GrayImage

[#] Some of these functions use data types / variables that are for internal NeatVision use **only**. Access to such data (e.g. pixel access) is can be done directly in Java, see example in Appendix A.1

FillBox	Draw a filled box in the grey-scale image	0:GrayImage 1: Coordinate [upper top left] 2: Coordinate [lower bottom right] 3: Integer [fill grey-scale intensity]	0:GrayImage
DrawCircle	Draw a white hollow circle in the grey-scale image	0:GrayImage 1: Coordinate [coordinate of the centre of the circle] 2: Integer [radius]	0:GrayImage
FillCircle	Draw a white filled circle in the grey-scale image	0:GrayImage 1: Coordinate [coordinate of the centre of the circle] 2: Integer [radius]	0:GrayImage
GetImageWidth	Width of the input grey-scale image	0:GrayImage	0: Integer [width of the input grey-scale image]
GetImageHeight	Height of the input grey-scale image	0:GrayImage	0: Integer [height of the input grey-scale image]
GenerateCoordinate	Generate the coordinate value from the (x,y) components.	0: Integer [x] 1: Integer [y]	0: Coordinate
GeneratePoints	Generate the (x,y) components of a given coordinate.	0: Coordinate	0: Integer [x] 1: Integer [y]
STRING			
StringAdd	Combine two strings (objects)	0: Undefined [first of two strings (objects) which are to be added] 1: Undefined [second of two strings (objects) which are to be added] 2:	0: String [The resulting string which is made up from the two input strings]
StringToLowerCase	A string which shall be converted to lower case	0: String	0: String
StringToUpperCase	A string which shall be converted to upper case	0: String	0: String
MATH[#]	Library of standard mathematical operators.		
JAIColour	See the Java™ Advanced Imaging website: *http://java.sun.com/products/java-media/jai/*		

[#] Some of these functions use data types / variables that are for internal NeatVision use **only**. Access to such data (e.g. pixel access) is can be done directly in Java, see example in Appendix A.1

OSMIA – Tina 5 Interface[5]			
NemaReader	Read a *Nema* image	Path of the file through the graphical interface	0: GrayImage 1: Double [minimum pixel value] 2: Double [maximum pixel value]
AiffReader	Read an *Aiff* image	Path of the file through the graphical interface	0: GrayImage 1: Double [minimum pixel value] 2: Double [maximum pixel value]
RenderEngineN	Render a binary volume image	0: VolumeImage [binary] 1: Integer [Thickness factor: Values: 1 to n]	
Ej_Frac	Compute the ejection fraction from two volume images.	0: VolumeImage [systole] 1: VolumeImage [diastole]	0: VolumeImage [binary] 1: VolumeImage [binary] 2: Double [Ejection fraction value]
Flow2D	Compute the 2D optical flow (Horn-Schunck or Lucas-Kanade). Original code by Prof. John Barron, UWO, Canada	0: String [Directory path for optical flow RAS sequence] 1: String [Stem name of the sequence] 2: Boolean [Swap data: PC should be TRUE] 3: Boolean [Method: TRUE : Horn-Schunck. FALSE: Lucas-Kanade] 4: Integer [Flow number: middle index of the sequence] 5: Double [Tau parameter, Default value: 0.5] 6: Double [Alpha parameter, Default value: 1.0] 7: Integer [Number of iterations, Default value: 50] 8: Integer [Offset parameter, Default value: 6] 9: Double [Scale parameter, Default value 12.0]	0: GrayImage [Output image illustrating the flow vectors]

[5] See http://www.eeng.dcu.ie/~whelanp/osmia/ for details on interfacing NeatVision with Tina 5.0

Aorta_n	Detect the aorta outline in a greyscale image.	0: GrayImage [Input image] 1: String [Path of the model data] 2: String [Path of the pca_model data] 3: Double [minimum pixel value] 4: Double [maximum pixel value]	0: Image [RGB image highlighting the aorta outline]
TSmooth	Tangential smooth operator	0: GrayImage [Input image] 1: Integer [Number of iterations]	0: GrayImage [Output image]
XY_Norm	Coil correction algorithm	0: GrayImage [Input image] 1: Double [Standard deviation]	0: GrayImage [Output image]
st_rec	Stereo rectification algorithm	0: String [Path of left image – aiff format] 1: String [Path of right image – aiff format] 2: String [Path of left cam file] 3: String [Path of right cam file]	0: GrayImage [Input left image] 1: GrayImage [Input right image] 2: GrayImage [Output left image] 3: GrayImage [Output right image]
Pairwise	Pairwise geometric histograms 2D object recognition	0: String [Directory path] 1: String [Filename – scene data] 2: String [Filename – model data]	0: Image [RGB image: Scene data] 1: Image [RGB image: Model data] 2: Image [RGB image: Recognised model superimposed on the input scene data]

References:

1. Paul F. Whelan, Robert J. T. Sadleir, and Ovidiu Ghita, (2004) "Informatics in Radiology (infoRAD): NeatVision: Visual Programming for Computer-aided Diagnostic Applications", Radiographics; 24(6):1779-1789

2. Robert J. T. Sadleir, Paul F. Whelan, Padraic MacMathuna, and Helen M. Fenlon (2004) "Informatics in Radiology (infoRAD): A Portable Toolkit for Providing Straightforward Access to Medical Image Data" Radiographics. 2004 Jul-Aug:24(4):1193-1202

3. Paul F. Whelan and R.J.T. Sadleir (2004), "A Visual Programming Environment for Machine Vision Engineers", Sensor Review, 24(3):265-270

4. Paul F. Whelan (2004), Neatvision 2.1 Users Guide.

5. Paul F. Whelan (2003), "Automated cutting of natural products: A practical packing strategy", Chapter 11 in Machine Vision for the Inspection of Natural Products. Mark Graves and Bruce Batchelor (Eds.), Springer (London). , ISBN: 1-85233-525-4, pp 307-329

6. Paul F. Whelan (2001), "Visual programming for machine vision", Chapter 8 in Intelligent Machine Vision: Techniques, Implementation & Interfacing B.G. Batchelor and F. Waltz, Springer-Verlag UK, 448 pages, ISBN: 3-540-762248, pp 229-320

7. Paul F. Whelan and D. Molloy (2000), Machine Vision Algorithms in Java: Techniques and Implementation, Springer (London), 298 Pages. ISBN 1-85233-218-2.

8. B.G. Batchelor and Paul F. Whelan (1997), Intelligent Vision Systems for Industry, Springer-Verlag (London), 457 pages, ISBN 3-540-19969-1.

9. Paul F. Whelan (1997), "Remote access to continuing engineering education - RACeE", IEE Engineering Science and Education Journal, 6(5), pp 205-211. Also published in the IEE Computer Forum.

10. Paul F. Whelan, B.G. Batchelor, M.R.F. Lewis and R. Hack (1997), "Machine vision and the World Wide Web: Design and training aids", Proceedings of the SPIE - The International Society for Optical Engineering, Vol. 3205 - Machine Vision Applications, Architectures, and Systems Integration VI, Pittsburgh (USA), pp 284-294.

11. B.G. Batchelor and Paul F. Whelan (1995), "Real-time colour recognition in symbolic programming for machine vision systems", Machine Vision and Applications; 8(6):385-398

12. B.G. Batchelor and Paul F. Whelan (Eds) (1994), Selected Papers on Industrial Machine Vision Systems, SPIE Milestone Series MS 97, SPIE Optical Engineering Press, 629 pages