$

NeatVision: A Development Environment for Machine Vision Engineers

By

Paul F. Whelan, Robert Sadleir & Ovidiu Ghita[1]

Abstract: This Chapter will detail a free image analysis development environment for machine vision engineers. The environment provides high-level access to a wide range of image manipulation, processing and analysis algorithms (over 300 to date) through a well-defined and easy to use graphical interface. Users can extend the core library using the developer's interface, a plug-in, which features, automatic source code generation, compilation with full error feedback and dynamic algorithm updates. The Chapter will also discuss key issues associated with the environment and outline the advantages in adopting such a system for machine vision application development.

$.1 Introduction

For many novices to the world of machine vision, the development of automated vision solutions may seem a relatively easy task, as it only requires a computer to understand basic elements such as shape, colour and texture? Of course this is not the case. Extracting useful information

[1] *http://www.cipa.dcu.ie/*

from images is a difficult task and as such requires a flexible machine vision application development environment. The design of machine vision systems requires a broad spectrum of techniques and disciplines. These include electronic engineering (hardware and software design), engineering mathematics, physics (optics and lighting), mechanical engineering (since industrial vision systems deal with a mainly mechanical world) as well as the system engineering aspects of developing reliable industrial systems. In this chapter will focus on one aspect of the machine vision design cycle, namely the algorithm development environment (refereed to as NeatVision (1)). It aims to provide novice and experienced machine vision engineer's with access to a multi-platform (realised through the use of Java) visual programming development system.

Java is an interpreted programming language and as such applications written in Java are not executed directly on the host computer, instead these applications are interpreted by the Java Virtual Machine. As a results Java programs generally run slower than native compiled programs written in languages such as C or C++. This performance issue is constantly being addressed by Sun Microsystems. Just-in-time compliers significantly improved the performance of Java applications by removing the need to reinterpret already executed code. Sun further improved the performance of Java by introducing HotSpot technology. This technology enhances application performance by optimising garbage collection and improving memory management. With the recent release of the Java 2 Platform Standard Edition 1.5.0 the performance of Java is approaching that of native programming languages.

The NeatVision environment provides an intuitive interface which is achieved using a drag and drop block diagram approach. Each image processing operation is represented by a graphical block with inputs and outputs that can be interconnected, edited and deleted as required. NeatVision (Version 2.1) is available free of charge and can be downloaded directly via the Internet[2].

$.1.1 Standard Installation

The requirements for the standard NeatVision installation are:

- **JRE 1.4.X -** The J2SE Java Runtime Environment (JRE) allows end-users to run Java applications. (e.g. *j2re-1_4_2_09-windows-i586-p.exe*)
- **JAI 1.X -** Java Advanced Imaging (JAI) API. (e.g. *jai-1_1_X-lib-windows-i586-jre.exe*)
- **NeatVision Standard Edition** (*neatvision.jar*). NeatVision is distributed as a ".jar" file. The contents of this file should not be extracted; any attempt to do this will cause NeatVision to cease functioning.

Please insure that the JAI is placed in the same path as the JRE. This only becomes an issue if you have multiple versions of Java on your machine

[2] www.NeatVision.com

(the default is that JAI will place itself in the most recent version of the Java). Assuming the JRE has been installed in `C:\Program Files\Java\j2re1.4.2_09` and the NeatVision jar file is in `D:\NV`, then the following single line command[3] will enable NeatVision to run from `D:\NV`.

```
"C:\Program Files\Java\j2re1.4.2_09\bin\java.exe" -classpath
              D:\NV\neatvision.jar NeatVision
```

$.2 NeatVision: An Interactive Development Environment

NeatVision is just one example of a visual programming development environment for machine vision (2), other notable examples include commercial programmes such as Khoros (3) and WiT (4). Visual programming involves defining variables, specifying operations, which are to be performed on these variables and their derivatives in order to perform a specific task. This is achieved by creating a structured flow of data using branching, looping and conditional processing. Traditionally computer programs have been written using textual programming languages. These programs can process data in a complex fashion; unfortunately the data paths and the overall structure of the program cannot be easily identified from the textual description. This can make it very difficult to appreciate the relationship between the source code and the functionality, which it represents. Although the programmer specifies the data flow in a visual program, the order in which the components execute is defined by the availability of data. Conditional processing concepts are supported in the visual domain by using dedicated flow control components. The main disadvantages of existing visual programming environments includes their cost, lack of cross platform support and the fact that they tend to be focused on image processing rather than image analysis applications (the latter must be considered a key element of any practical machine vision application).

Text based programming languages such as MATLAB (5) can be a powerful alternative to visual programming. In addition to the disadvantages outlined with respect to the visual programming languages, text based approaches require the user to have a higher level of programming skills when compared to visual programming environments. Text based interactive environments are generally more suitable to experienced users, in fact experienced users can become frustrated by the visual programming environment as complex programmes can take longer to develop (Note: we have recently developed a MATLAB compatible *VSG Image Processing & Analysis Toolbox* toolbox (12) that replicates NeatVisions functionality to allow MATLAB users build machine vision

[3] The NeatVision argument **must have** a capital 'N' and 'V'.

solutions). Hence our aim is to produce a suitable environment for those new to machine vision while retaining the flexibility of program design for the more experienced users. Based on our review of exiting text and visual programming based machine vision development environments, the key criteria necessary are outlined below:

- **Multi-platform**: The development environment must be able to run on a wide range of computer platforms.
- **Focused on machine vision engineering**: The environment should contain a wide range of image processing and analysis techniques necessary to implement practical machine vision engineering applications.
- **Easy to use**: It should allow users to concentrate on the design of machine vision solutions, as opposed to emphasizing the programming task.
- **Upgradeable**: The environment must contain a mechanism to allow users to develop custom vision modules.

A visual program can be created by defining input data using the input components, then implementing the desired algorithm using the processing and flow control components. The data flow is specified by creating interconnections between the components. The program can be completed by adding output components to view the data resulting from the algorithm execution. Details on the design of the NeatVision development environment appear elsewhere (6).

$.3 NeatVision's Graphical User Interface (GUI)

The NeatVision GUI (Figure $.1) consists primarily of a workspace where the processing blocks reside. The processing blocks represent the functionality available to the user. Support is provided for the positioning of blocks around the workspace, the creation and registration of interconnections between blocks. The lines connecting each block represent the path of the data through the system. Some of the blocks can generate child windows, which can be used for viewing outputs, setting parameters or selecting areas of interest from an image. If each block is thought of as a function, then the application can be thought of as a visual programming language. The inputs to each block correspond to the arguments of a function and the outputs from a block correspond to the return values. The advantage of this approach is that a block can return more than one value without the added complexity of using C-style pointers. In addition, the path of data through a visual program can be dictated using special flow control components. A visual program can range in complexity from three components upwards and is limited only by the availability of free memory.

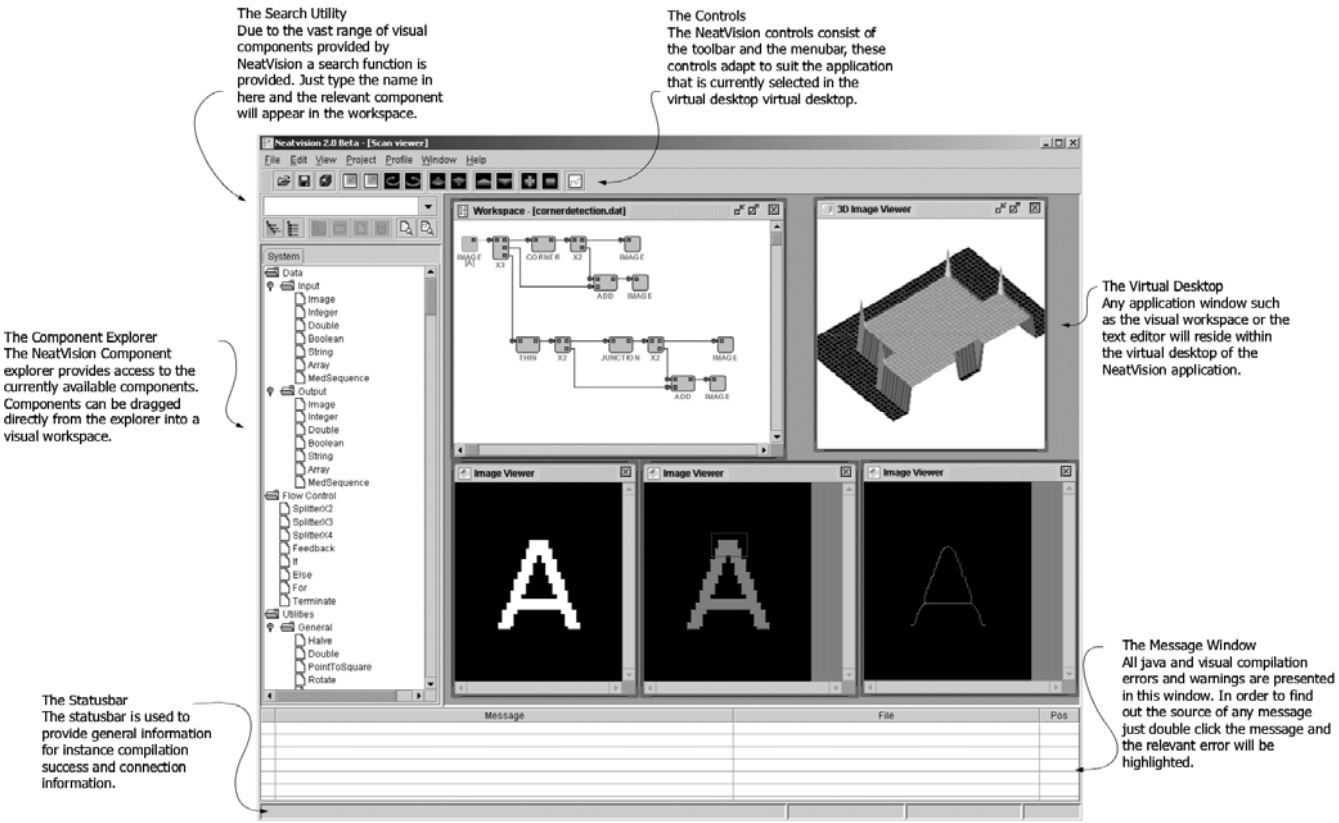


Figure $.1: Key features of the NeatVision environment.

As each block is processed it is highlighted (in green) to illustrate that it is active. This allows users to see the relevant speeds of parallel data streams within a visual program. This can help identify potential processing bottlenecks within the workspace allowing for a more efficient balanced design. The colour coding of the blocks data connection type and its status also aids in the design process. To aid user operation each data connection has two colour coded properties, namely the block *data type* and *connection status*. NeatVision currently supports eight data types, i.e. Image (red), Integer / Array data (green), Double precision Floating point data (blue), Boolean data (orange), String data (pink), Fourier data (light blue), Coordinate data (purple) and Undefined data (black). The other connection property relates to its status. There are three main states for a connection, *connected (green), disconnected (red)* and *disconnected but using default value (orange)*.

This approach provides a fast and simple alternative to conventional text based programming, while still providing much of the power and flexibility. The visual workspace can be compiled and executed as with a conventional programming language. Errors and warnings are generated depending on the situation. There is currently support for 15 input graphics file formats and 13 output formats. Some of the main formats are listed below (*R* indicates a read-only file format and *RW* indicates a read/write file format).

- **BMP** (*RW*) Microsoft Windows Bitmap.
- **BYT** (*RW*) Raw image data, grey scale only with a maximum pixel depth of 8 bits.
- **FPX** (*R*) Kodak FlashPix is a multiple-resolution, tiled image file format based on JPEG compression.
- **GIF** (*R*) Graphics Interchange Format (GIF), is a bitmap file format which utilises Lemple-Zev-Welch (LZW) compression.
- **JPEG (JPG)** (*RW*) Joint Photographic Experts Group (JPEG) file interchange format is a bitmap file utilising JPEG compression.
- **PCX** (*RW*) PC Paintbrush (PCX), a bitmap file using either no compression or Run Length Encoding (RLE).
- **PNG** (*RW*) PNG is a lossless data compression method for images.
- **PBM** (*RW*) Portable BitMap. The portable bitmap format is a lowest common denominator monochrome file format.
- **PGM** (*RW*) Portable Greymap Utilities. This is a non-compressed bitmap format, hence allowing image data to be left intact.
- **PPM** (*RW*) Portable PixelMap. The portable pixelmap format is a lowest common denominator colour image file format.
- **RAS** (*RW*) Sun Raster Image (RAS), a bitmap file format using either no compression or RLE.

- **RAW** (*RW*) Raw image data. This is similar to the BYT format described earlier except in this case colour image data is also supported.
- **TIFF (TIF)** (*RW*) Tagged Image File Format is a bitmapped file format using a wide range of compression techniques.

System parameters can be adjusted and the system may be reset and executed again until the desired response is obtained. At any stage blocks may be added or removed from the system. NeatVision also contains a built in web browser to allow easy access to online notes and support tools.

$.4 Design Details

NeatVision is designed to work at two levels. The *user* level allows the design of imaging solutions within the visual programming environment using NeatVisions core set of functions. NeatVision (Version 2.1) contains 300 image manipulation, processing and analysis functions, ranging from pixel manipulation to colour image analysis to data visualisation. To aid novice users, a full introductory tutorial and some sample programmes can be found on the NeatVision website. A brief description of the main system[4] components (7) is given below; see Appendix I for additional details:

- **Data types**: Image, integer, double, string, Boolean, array, medical image sequences.
- **Flow control**: Path splitting, feedback, if (else), for loop. (See Figure $.2).
- **Utilities**: Rotation, pixel manipulation, resize, URL control, additive noise generators, region of interest, masking operations.
- **Arithmetic operators**: Add, subtract, multiply, divide, logical operators.
- **Histogram**: General histogram analysis algorithms, local equalization.
- **Image Processing**: Look-up tables (LUT), threshold, contrast manipulation.
- **Neighbourhood based filtering**: Lowpass, median, sharpen, DOLPS, convolution, adaptive smoothing (or filtering).
- **Edge detection**: Roberts, Laplacian, Sobel, zero crossing, Canny
- **Edge features**: Line/arc fitting, edge labelling and linking.
- **Analysis**: Thinning, binary detection, blob analysis, labelling, shape feature measures, bounding regions, grey scale corner detectors.

[4] Items in italics are only included in the NeatVision advanced edition.

- **Clustering**: K-means (grey scale and colour), unsupervised colour clustering.
- **Image transforms**: Hough (line and circle), Medial Axis, DCT, Cooccurrance, Fourier, distance transforms.
- **Morphology**: Several 2D morphological operators, including erosion, dilation, opening, closing, top-hat, hit-and-miss, watershed.
- **Colour**: Colour space conversion algorithms, RGB, HSI, XYZ, YIQ, Lab.
- **3D Volume**: 3D Operators (thinning, Sobel, threshold, labelling), maximum and average intensity projections, rendering engine (Java and Intel native: wire frame, flat, Gouraud, Phong), *3D to 2D conversion, data scaling, 3D windowing, 3D arithmetic, 3D image processing, 3D labelling, 3D morphological operators, 3D reconstruction, 3D clustering*
- **Low Level**: Pixel level operators; get pixel value, set pixel value and basic shape generation.
- **String**: String operators, object addition, to upper case and to lower case.
- **Maths**: An extensive range of numerical operators and utilities, including constants and random number generation.
- **JAI Colour**: Colour algorithms implemented using JAI (Java Advance Imaging (8)), operators, processing, filters and edge detectors.
- **OSMIA functions (Wintel native only):** NEMA and AIFF image reader, ejection fraction measurement, 2D optical flow (Lucas & Kanasde and Horn & Shcunck courtesy of Barron (9) via the European Union funded OSMIA project (10)), XY normalization

At the more advanced *developers* level (11), NeatVision allows experienced application designers to develop and integrate their own functionality through the development and integration of new image processing/analysis modules.

$.5 Developers Environment

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 2.1 of the NeatVision visual programming environment. This allows users 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.

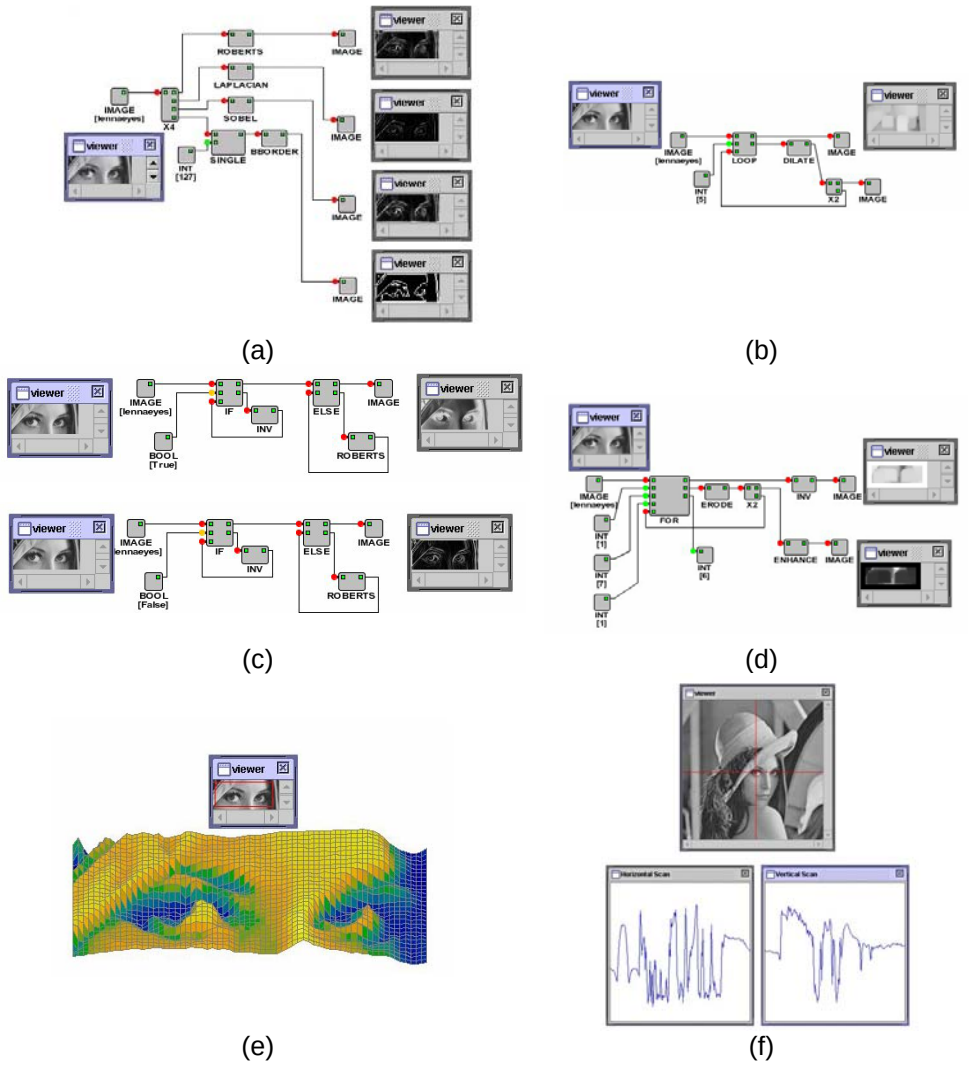


Figure $.2 Flow control / graphic utility examples: (a) Path splitting, (b) looping feedback, (c) if (else), (d) for loop, (e) 3D Viewer, (f) Image profiling.

NeatVision development assumes a basic level of familiarity with the Java programming language from Sun Microsystems and the NeatVision

developers plug-in. Additional details on developing for NeatVision (11) and the design concepts behind NeatVision along with detailed explanations of many of the its algorithms can be found are also available (6).

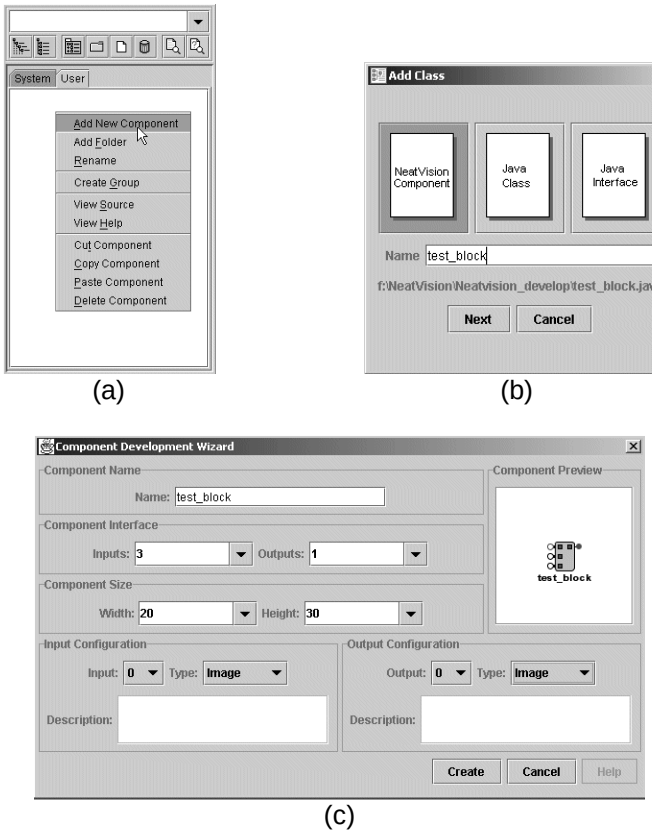


Figure $.3: NeatVision component development wizard. (a) Select *Add New Component* from the popup menu of the *user* area of the component explorer. (b) Select component to be added. (c) Define the component skeleton (i.e. the number and type of inputs and outputs for the associate block). This generates the necessary component wrapping code.

When the developer's interface and the Java Developers Kit (JDK)[5] are present a 'user' tab appears alongside the 'system' tab in the component

[5] Installation of the developer plug-in will require you to upgrade the JRE to the JDK. As the JDK compiler is not backward compatible and is frequently modified, we have restricted developer's interface to a specific version of the Java - i.e. JDK 1.3.X. To activate the developers version, you will need the **developers update file** (*developer.jar*). To install this please close down NeatVision and place *developer.jar* in the classpath. It should be activated the next time you start NeatVision. (Also be sure to download the latest version of

explorer. The developer can add a new component by right clicking anywhere within the ‘user’ view of the component explorer. After right clicking, a popup menu will appear. The ‘Add New Component’ option must be selected from this menu in order to create a new file (Figure $.3). The user is then queried as to whether they would like to create a NeatVision Component, a Java Class or a Java Interface (Note: a NeatVision Component is just a special type of Java class). A filename for the class or interface must be specified at this point. If a Java Interface or standard Java class is specified at then a text editor window is displayed with the relevant skeleton source code. The developer may edit and compile this code as desired. If a NeatVision component is specified then a component development wizard is launched.

The wizard allows the developer to specify the visual appearance of the component including width and height in pixels, component name and number of inputs and outputs. The wizard also allows the developer to specify the data type and data description associated with each of the inputs and outputs. Once all of the required information has been entered the developer need only press the ‘Create’ button to generate to skeleton source code for the desired NeatVision component. The developer can then edit the skeleton source code (Example $.1) as required in order to implement the desired component functionality. At any stage the source code for the component can be compiled. This is achieved by selecting the relevant compile option from the ‘Project’ menu. Selecting the compile option launches the Java compiler distributed with the JDK and any errors in the specified file(s) are subsequently listed in the message window at the bottom of the main NeatVision window. For each error a description, filename and line number are provided. The user need only click on an error message to highlight the relevant error in the source code. Once all errors have been corrected the message ‘compilation succeeded’ is printed in the status bar. Following successful compilation the block is available for use and can be included in a workspace like any core NeatVision component.

The NeatVision developers interface extends and complements the visual programming interface by allowing users to develop custom components that can encapsulate the functionality of core NeatVision components, thus extending the already vast NeatVision library of components.

---

neatvision.jar). e.g. `..\java.exe -classpath ..\neatvision\neatvision.jar; ..\neatvision\developer.jar NeatVision`

$.5.1 Developing for Reuse

As mentioned previously, the skeleton code for a new NeatVision component is generated using the component development wizard (see Example $.1). In previous versions of NeatVision 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 2.1 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[6]. 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).

$.5.1.1 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`** can be found in the NeatVision Developers guide (11).

$.5.2 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)
```

---

[6] **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.

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.

Example $.2 illustrates a simple example of reuse. This involves calling the `Not` operation from inside a custom user defined class. Example $.3 illustrates the development of the `TestDev` class. This sample program

removes boundary regions prior to K-Means clustering. The Canny edge detector is then applied to the original image and keeping only closed structures we find the approximate perimeter of the strong edges. The K-Means and the closed structure overlay images along with the approximate perimeter values are the final block outputs.

```
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.

$.6 Sample Applications

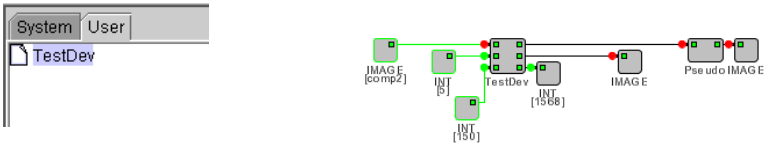
NeatVision provides an image analysis software development environment that can work at several levels. For example, at a relatively low level individual pixels can be manipulated. Alternatively, NeatVisions built in functionality can be used to generate solutions to complex machine vision problems. Figure $.5 illustrates how the *reconstruction by dilation* morphological operator can be used to remove or detect objects that are touching the binary image border. Figure $.6 illustrates how NeatVision can be used to isolate defects in the centre panel of a crown bottle top.

```
// 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);
        }
```

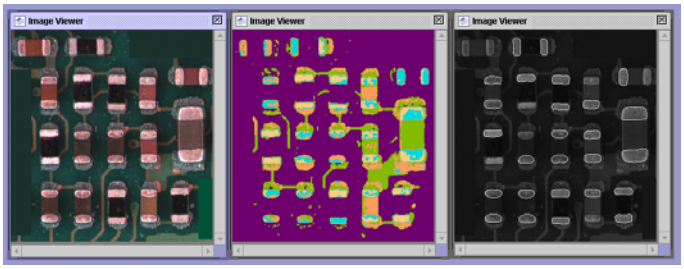
Example $.3: Development of the `TestDev` class.

```
// 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);
        }
}
```

Example $.3: Development of the `TestDev` class (Cont'd)

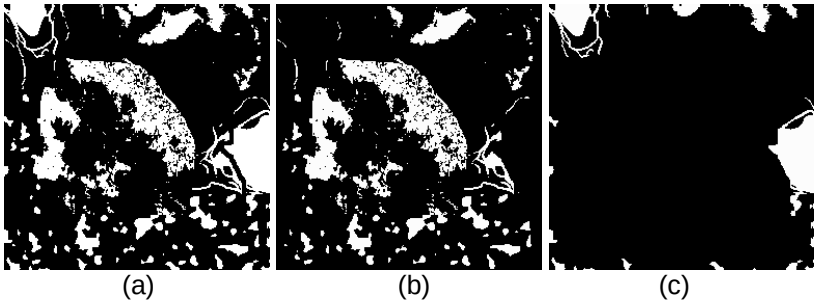


(a) (b)

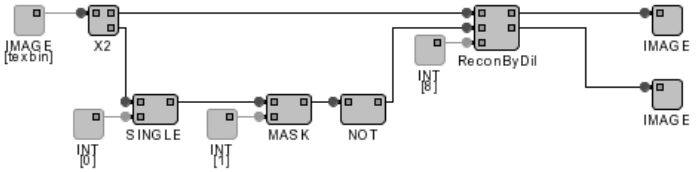


(c)

Figure $.4. Implementation of the `TestDev` class illustrated in Example $.3. (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-Means clustered image and the closed structure overlay image.

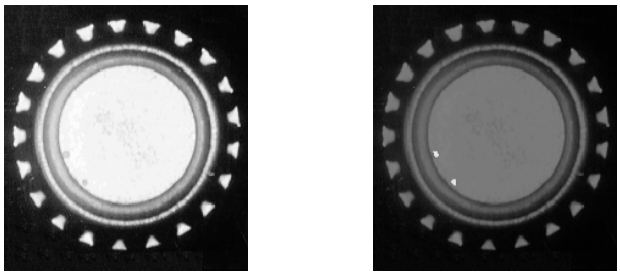


(a) (b) (c)

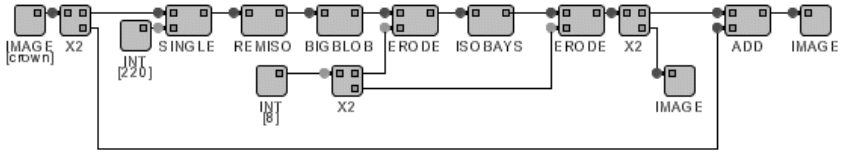


(d)

Figure $.5: Removal of boundary objects using reconstruction by dilation. (a) Original image. (b) Boundary object removal. (c) Detected boundary objects. (d) Associated visual workspace.



(a) (b)



(c)

Figure $.6: Bottle top inspection. (a) Input image. (b) Output image in which the defects are highlighted in white. (c) The NeatVision visual program.

$.7 Application Development Case Study

In this section we will outline the application of NeatVision to a typical machine vision application, namely the characterization of surface mount components. We aim to develop a robust *NeatVision* program capable of:

- Automatically counting all components fully within the field of view in the image illustrated in Figure $.7 (i.e. all elements touching the image boundaries must be removed prior to image analysis).
- Isolate, and highlight, the largest component within the image.
- Find the approximate area of the largest component within the image

The system must be robust, and with this in mind it should be capable of automatically determining threshold values from the image data.

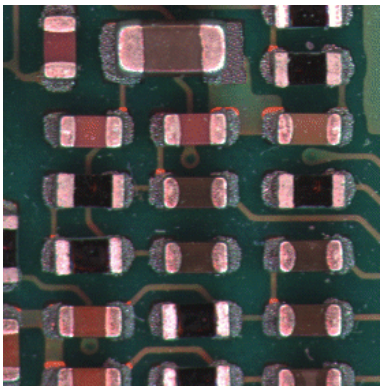


Figure $.7: Image segment (pcb_3) containing 15 surface mounted components fully with in its field of view. (Image supplied courtesy of Agilent Technologies (Ireland), acquired using their *sequential colour* technique at 470, 524 and 660 nm (BGR))

$.7.1 Outline Solution

The proposed solution, as summarized in the NeatVision program illustrated in Figure $.8 consists of a number of distinct stages. This solution is used solely to illustrate to power and flexibility of NeatVision and does not represent the optimal solution for such an application.

Stage 1: The input colour image (pcb_3.gif) is loaded and converted to greyscale. A binary version of this image is then automatically produced by examining the grey scale histogram upper and lower values. Due to the high contrast the automated threshold selection consists of the mid point between the upper and lower grey scales in the original image, Figure $.9.

Stage 2: Using the morphological technique *Reconstruction by Dilation*, we isolate any part of the image touching the boundary, Figure $.10

Stage 3: Once the incomplete objects touching the image boundary have been removed we use a 5x5 RAF (Rectangular Averaging Filter) to remove any remaining noise, Figure $.11. Take care to threshold the filtered image at mid grey as the RAF filter produces a grey scale image.

Stage 4: Each blob region is now assigned a grey scale value using labelling by location. These grey patches are then count by finding and marking each grey patch by its centroid. The centroids are then counted and divided by 2 to give the final component count. Figure $.12

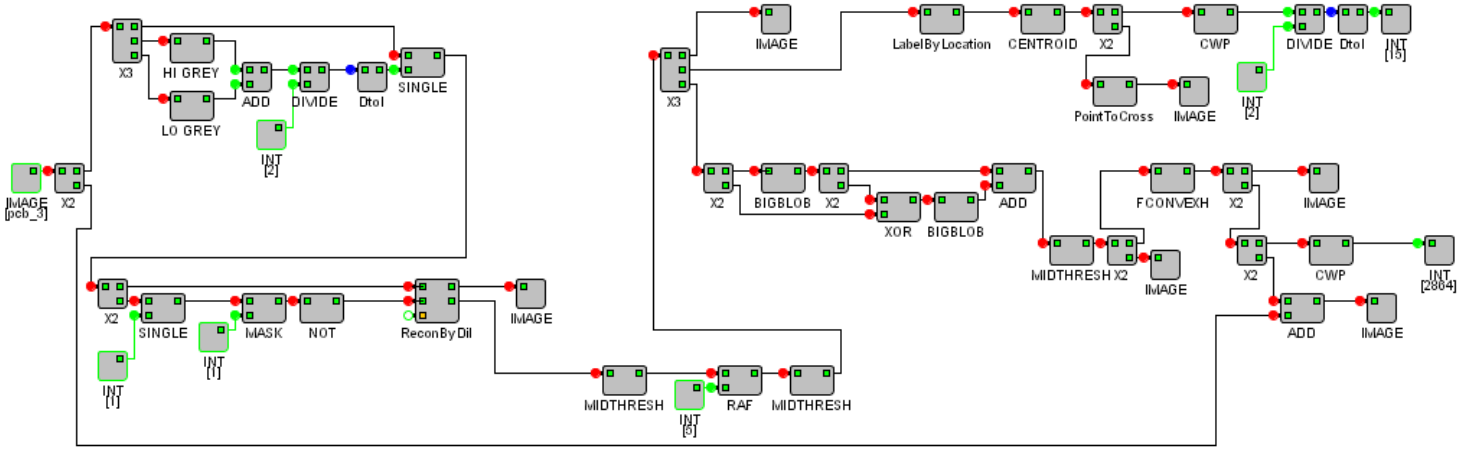


Figure $.8: Proposed NeatVision solution.

Stage 5: Using the fact that the largest surface mounted component is bounded by the two largest blobs in the image, we can identify and extract these blob regions, Figure $.13

Stage 6: The final step involves finding the convex hull of the output from the previous stage. This now approximates the largest surface mounted component. The area of this region is then calculates to determine its size. This convex hull image is then combined with the original image to highlight its location, Figure $.14.

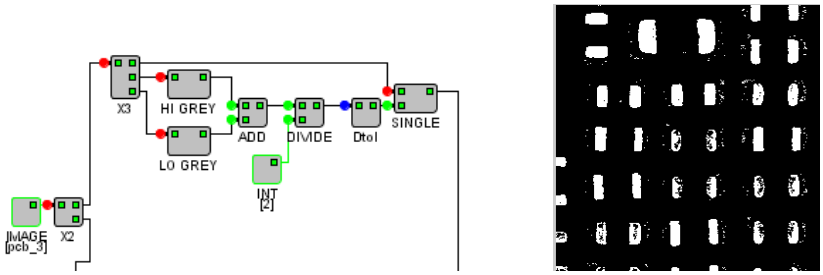


Figure $.9: Automated threshold selection stage and the resultant binary image.

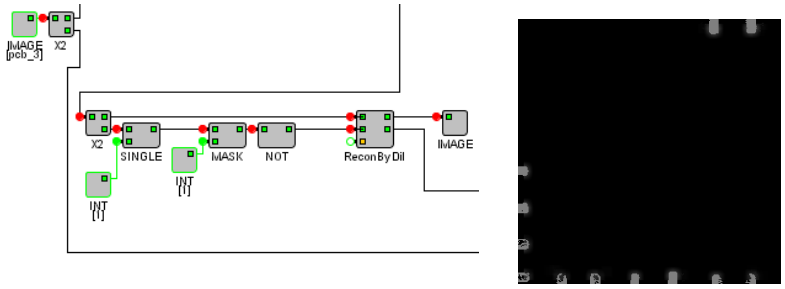


Figure $.10: Using reconstruction by dilation to identify the incomplete objects touching the image boundary.

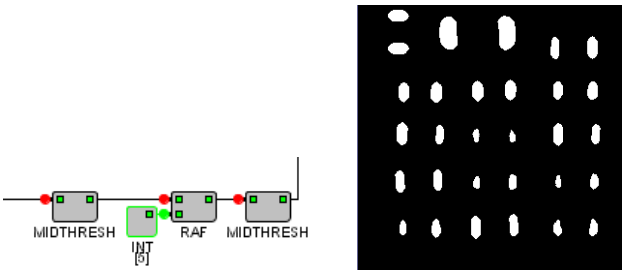


Figure $.11: Isolating the components of interest and noise removal.

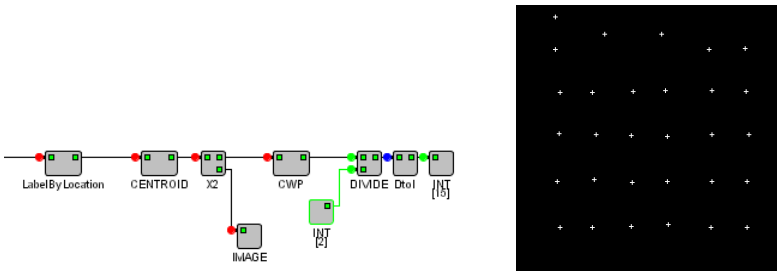


Figure $.12: Component counting by isolating and counting the grey patch centroid values (a cross indicates each centroid value).

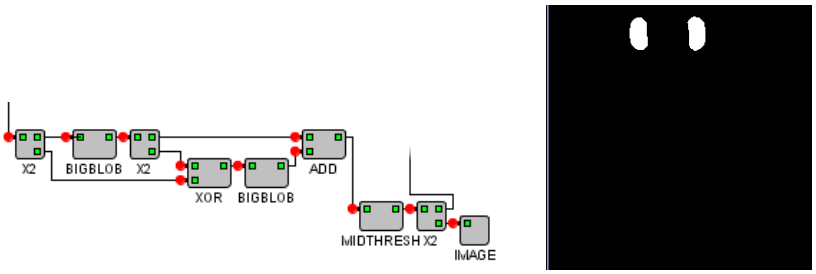


Figure $.13: Identifying the bounds of the largest component.

Figure $.14: Identifying calculating the area of the largest component. Overlay on original image to highlight its location.

$.8 Conclusions

NeatVision was designed to allow novice and experienced users to focus on the machine vision design task rather than concerns about the subtlety of a given programming language. It allows machine vision engineers to implement their ideas in a dynamic and straightforward manner. NeatVision standard and developers versions are freely available via the Internet and are capable of running on a wide range of computer platforms (e.g. Windows, Solaris, Linux).

Acknowledgements: While NeatVision is primarily a collaboration of researchers within the Vision Systems Group in Dublin City University (Ireland), we would also like to acknowledge all those who have supported the NeatVision project through the submission of algorithms for inclusion or for their constructive and useful feedback. Development of the MATLAB compatible *VSG Image Processing & Analysis Toolbox* which offers MATLAB users NeatVision functionality was funded in part by HEA PRTLI IV *National Biophotonics and Imaging Platform Ireland* (NBIPI) (13).

References

1. NeatVision: Image Analysis and Software Development Environment. Available at *http://www.neatvision.com* Accessed July 2010
2. Sage D and Unser M, Teaching Image Processing Programming in Java. IEEE Signal Processing Magazine 2003: Nov:43-52
3. Khoros: Khoral Research, Inc Available at *http://www.khoral.com* Accessed July 2010
4. WiT: Logical Vision, Available at *http://www.logicalvision.com* Accessed July 2010

5. MathWorks: Matlab, Available at *http://www.mathworks.com* Accessed July 2010
6. Whelan PF and Molloy D (2000), "Machine Vision Algorithms in Java: Techniques and Implementation", Springer-Verlag, London
7. NeatVision: Users Guide, Available at *http://neatvision.eeng.dcu.ie/user.html* Accessed July 2010
8. JAI: The Java Advanced Imaging (API), Available at *http://java.sun.com/products/java-media/jai* 2005; Accessed Oct.25
9. Barron JL, Fleet DJ and Beauchemin S. Performance of optical flow techniques. International Journal of Computer Vision 1994; 12(1):43-77.
10. OSMIA - Open Source Medical Image Analysis, EU Fifth Framework Programme (IST: Accompanying Measures). Available at *http://www.eeng.dcu.ie/~whelanp/osmia/* Accessed July 2010
11. NeatVision: Developers Guide, Available at *http://neatvision.eeng.dcu.ie/developer.html* Accessed July 2010
12. VSG Image Processing & Analysis Toolbox (VSG IPA TOOLBOX beta) Available at *http://www.cipa.dcu.ie/code.html* Accessed July 2010
13. National Biophotonics and Imaging Platform Ireland (NBIPI). *http://www.nbipireland.ie/* Accessed July 2010

APPENDIX I:

Users Summary

Vision Systems Laboratory, Centre for Image Processing and Analysis (CIPA)
Dublin City University
info@neatvision.com

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[7]	A grey-scale image from which a rectangular region of interest is extracted by the user via the GUI.	0:GrayImage	0:GrayImage
PolyROI[8]	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
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

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

[8] 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**.

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	Any white pixels with less than one white pixel (3x3) neighbour are set to black. This can be used to remove noise from a binary image.	0:GrayImage	0:GrayImage [Binary]
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 (1-1024)	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]	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[9]	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)]

[9] 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]

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.

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]
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.	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

[#] 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.

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
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/*		
OSMIA – Tina 5 Interface	See http://www.eeng.dcu.ie/~whelanp/osmia/ for details on interfacing NeatVision with Tina 5.0		

[#] 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.