Digital Image Processing using *Mathematica* Link for LabVIEW

A demo developed for **NI Week 2002**

August 14-16, 2002, Austin, TX

Concept and Programming: David J. Ritter – BetterVIEW Consulting

Digital Image Processing using *Mathematica* Link for LabVIEW

Objective:

To acquire images into LabVIEW and process them using built-in Mathematica functions and the Digital Image Processing Add-ons for Mathematica.

Components of the Demo System:

- LabVIEW 6.1
- Mathematica 4.1
- Mathematica Link for LabVIEW (version 2.0)
- Digital Image Processing Add-ons for Mathematica
- Apple PowerBook G4 running MacOS 9.2
- Orange Micro 'iBot' Firewire webcam

Abstract:

This demo was designed to illustrate how LabVIEW-acquired data can be processed, analyzed, manipulated, and plotted using a combination of "LabVIEW", "Mathematica", and the "Mathematica Link for LabVIEW". While this particular demo uses image data, the same approach could be extended to any type of data that can be acquired using LabVIEW and compatible DAQ hardware.

Overview of the Process:

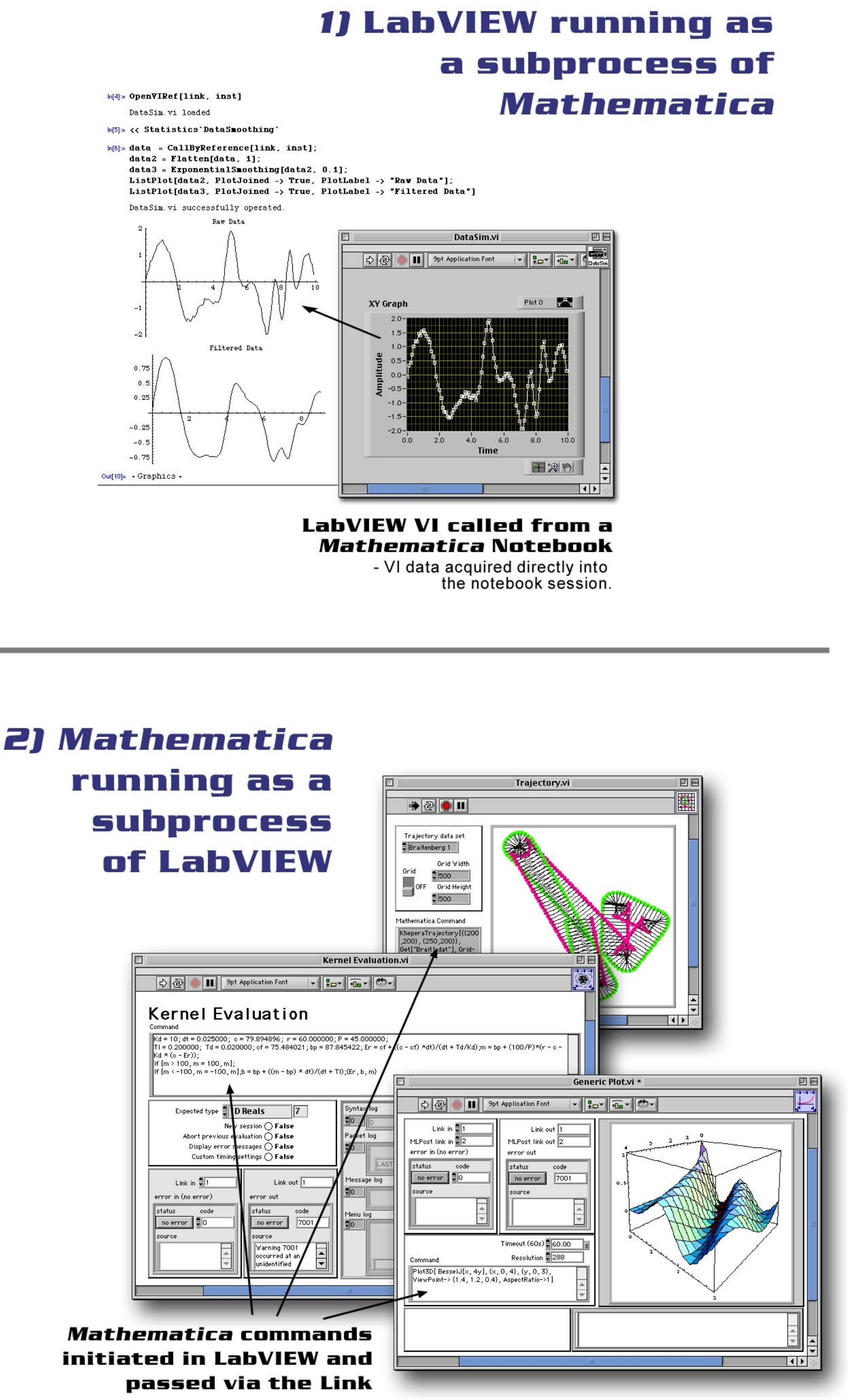
Because it is a flexible communications toolkit, Mathematica Link for LabVIEW projects can take many forms. However, the most common configurations will be one of the following:

1) Projects where a LabVIEW VI is called as a subprocess of Mathematica, and data is passed via the Link for further processing and analysis in a Mathematica notebook, and/or

2) Projects where the Mathematica Kernel is called as a subprocess of a LabVIEW VI.

Both configurations are illustrated in Figure 1 on the next page.

Figure 1. A bi-directional communications mechanism.


1) LabVIEW running as a subprocess of *Mathematica*
LabVIEW VI called from a *Mathematica* Notebook
- VI data acquired directly into the notebook session.
DataSim.vi
XY Graph
Plot 0
Amplitude
Time
Raw Data
Filtered Data
Out[10]= - Graphics -
2) *Mathematica* running as a subprocess of LabVIEW
Trajectory.vi
Kernel Evaluation.vi
Generic Plot.vi
Mathematica commands initiated in LabVIEW and passed via the Link

```
In[4]:= OpenVIRef[link, inst]

        DataSim.vi loaded

In[5]:= << Statistics`DataSmoothing`

In[6]:= data  = CallByReference[link, inst];
        data2 = Flatten[data, 1];
        data3 = ExponentialSmoothing[data2, 0.1];
        ListPlot[data2, PlotJoined -> True, PlotLabel -> "Raw Data"];
        ListPlot[data3, PlotJoined -> True, PlotLabel -> "Filtered Data"]

        DataSim.vi successfully operated.
```

Trajectory data set: Braitenberg 1
Grid: OFF
Grid Width: 500
Grid Height: 500

Mathematica Command

```
KheperaTrajectory[{{200
,200}, {250,200}},
Get["Brait1.dat"], Grid-
```

Kernel Evaluation

Command

```
Kd = 10; dt = 0.025000;  c = 79.894896;  r = 60.000000; P = 45.000000;
TI = 0.200000;  Td = 0.020000; cf = 75.484021; bp = 87.845422; Er = cf + ((c - cf) *dt)/(dt + Td/Kd);m = bp + (100/P)*(r - c -
Kd * (c - Er));
If [m > 100, m = 100, m];
If [m < -100, m = -100, m];b = bp + ((m - bp) * dt)/(dt + TI);{Er, b, m}
```

Expected type: D Reals 7
New session: False
Abort previous evaluation: False
Display error messages: False
Custom timing settings: False

Link in: 1
error in (no error): status no error, code 0
Link out: 1
error out: status no error, code 7001
source: Warning 7001 occurred at an unidentified

Link in: 1
MLPost link in: 2
error in (no error): status no error, code 0
Link out: 1
MLPost link out: 2
error out: status no error, code 7001
Timeout (60s): 60.00 s
Resolution: 288

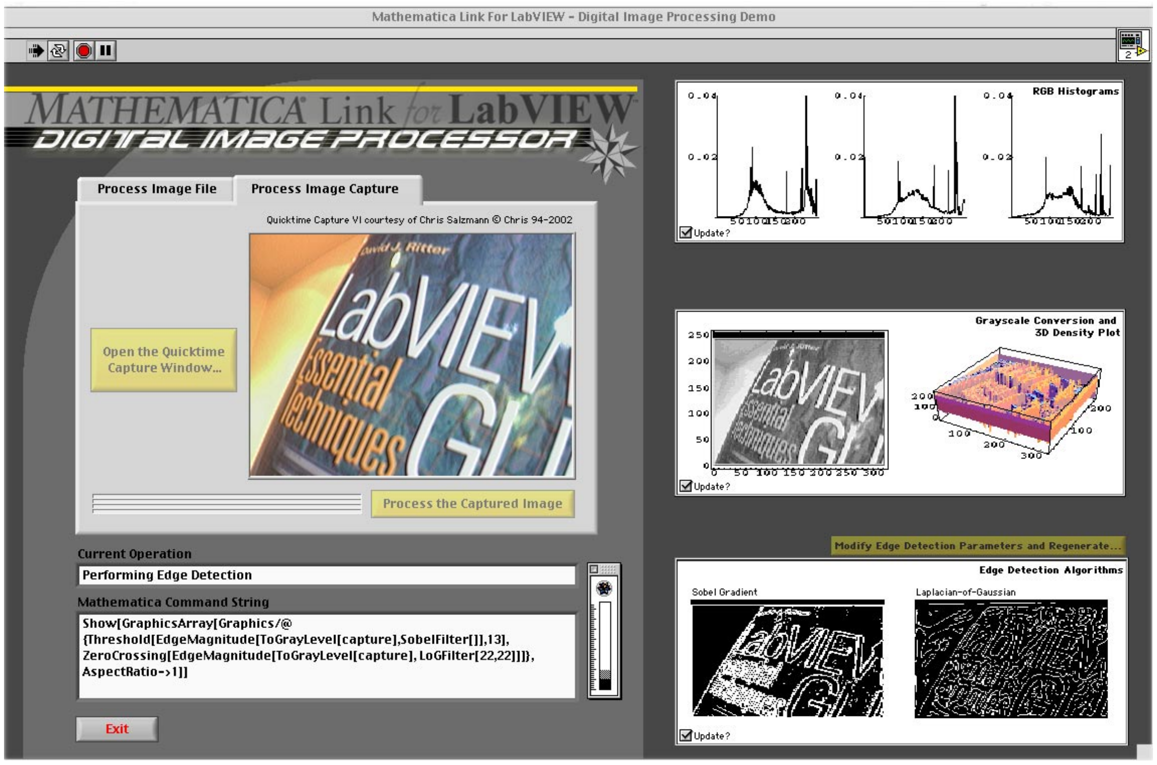
Command

```
Plot3D[ BesselJ[x, 4y], {x, 0, 4}, {y, 0, 3},
ViewPoint-> {1.4, 1.2, 0.4}, AspectRatio->1]
```

The most efficient way to proceed to the second configuration (in the *lower half* of Figure 1) often involves some experimentation using the first, or *upper* configuration.

With this in mind, we will now define our eventual target: a LabVIEW Image Processing application that calls Mathematica as a subprocess of LabVIEW. The front panel for the target application is featured in Figure 2. (More details about the operation of this VI will follow later in this document.)

Figure 2. The final image processing VI – front panel view.


Mathematica Link For LabVIEW - Digital Image Processing Demo
MATHEMATICA Link for LabVIEW
DIGITAL IMAGE PROCESSOR
Process Image File
Process Image Capture
Quicktime Capture VI courtesy of Chris Salzmann © Chris 94-2002
Open the Quicktime Capture Window...
Process the Captured Image
RGB Histograms
Update?
Grayscale Conversion and 3D Density Plot
Update?
Modify Edge Detection Parameters and Regenerate...
Edge Detection Algorithms
Sobel Gradient
Laplacian-of-Gaussian
Update?
Exit

Current Operation

Performing Edge Detection

Mathematica Command String

```
Show[GraphicsArray[Graphics/@
{Threshold[EdgeMagnitude[ToGrayLevel[capture],SobelFilter[]],13],
ZeroCrossing[EdgeMagnitude[ToGrayLevel[capture], LoGFilter[22,22]]]},
AspectRatio->1]]
```

Development Process:

The first step on the path to our final destination involved some experimentation within a Mathematica notebook. In the initial planning phase, a LabVIEW VI was called as a subprocess of Mathematica. For preliminary testing, we used 'MathLink VI Server.vi' and the 'VIClient.m' package (both included with Mathematica Link for LabVIEW) to call a Quicktime-based image capture VI. The Quicktime VI was supplied by Christophe Salzmann of EPFL. (Interested readers should note that this Quicktime capture VI has been featured in LTR articles, and can also be found on the CD-ROM distributed with "LabVIEW GUI - Essential Techniques", published by McGraw-Hill.)

The Mathematica-based investigations employed 2 steps:

1) Acquiring the image using the Quicktime VI, and

2) Processing the image in Mathematica.

These two steps are summarized in two Mathematica notebooks: "CaptureImage.nb" and "Process Capture.nb". The contents of both notebooks are combined in Figure 3 on the next page . Both of the notebooks depicted can be found in "LV_Capture_MM_Process.zip". (NOTE: Mathematica 41. or Wolfram's free Mathematica notebook viewer application are required to view these files. The viewer and more information can be found online at wolfram.com.)

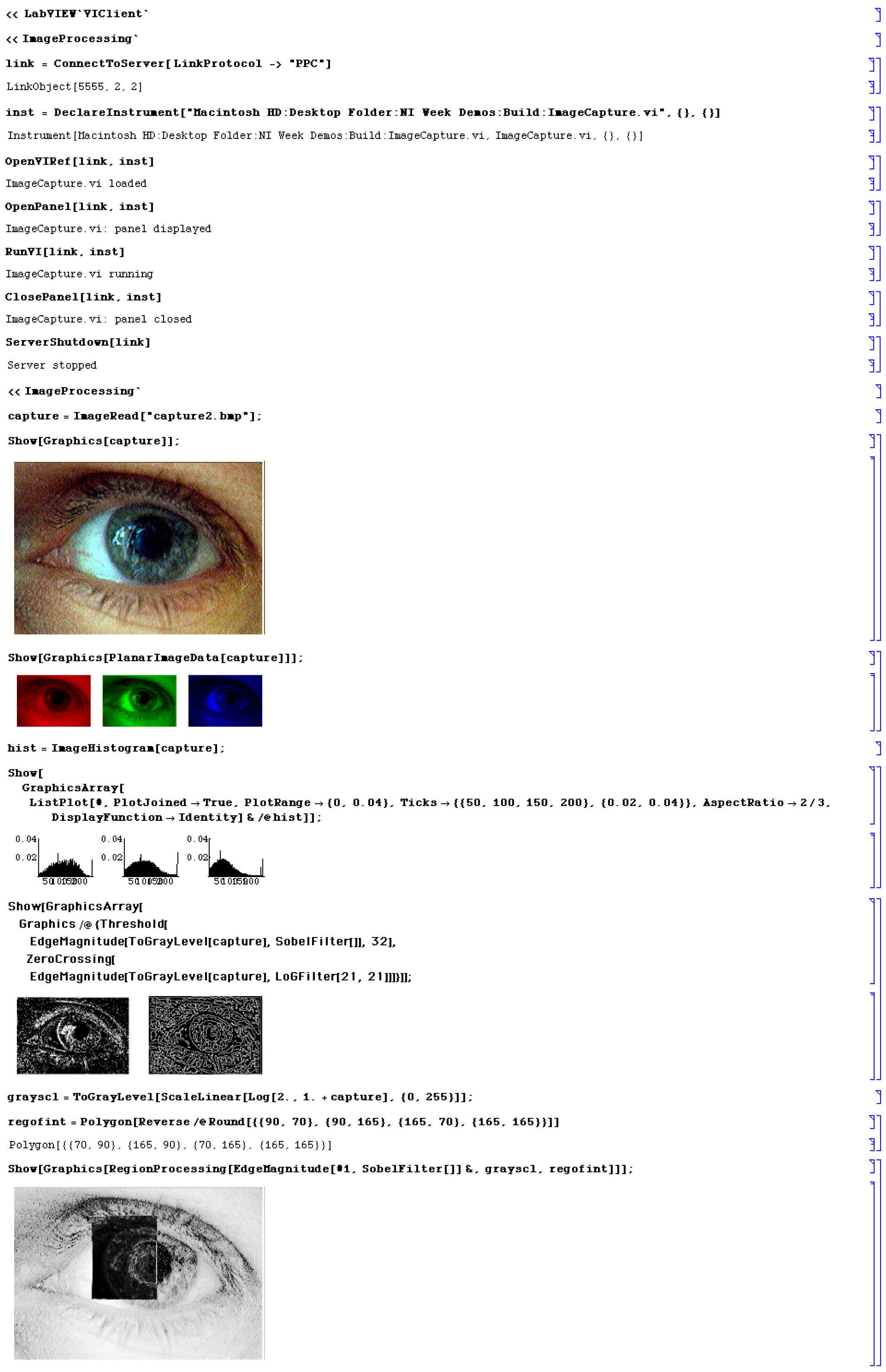
Developing the Final VI:

After defining the parameters of the experiment using VI calls inside a Mathematica notebook, the next step was to develop the MathLink-enabled LabVIEW VI. We incorporated the Mathematica commands from the initial Mathematica investigations into string constants on the LabVIEW VI diagrams. Integration of the Mathematica commands into the LabVIEW VI enabled us to simplify user interaction and ensure repeatability of our experiments. The final LabVIEW GUI replaces the command-line Mathematica user interface with a much easier to use LabVIEW-based user interface.

How it Works:

The front panel for the VI was presented previously in Figure 2. You may want to refer to this figure again throughout the following examinations.

Interaction with this VI begins in the upper-left area of the panel. For demonstration purposes, the VI offers two operational modes: "Process Image File", and "Process Image Capture". The user selects between these modes by changing pages of the Tab Control. The first mode, "Process Image File" applies the Mathematica processing steps to .TIF images stored on disk (see Figure 4). The second mode, "Process Image Capture" uses the same Quicktime image capture VI used in the initial Mathematica experiments, and captures a 'live' Firewire image for subsequent processing. This mode will be discussed in the next section (see Figure 5).



```
<< LabVIEW`VIClient`

<< ImageProcessing`

link = ConnectToServer[LinkProtocol -> "PPC"]

LinkObject[5555, 2, 2]

inst = DeclareInstrument["Macintosh HD:Desktop Folder:NI Week Demos:Build:ImageCapture.vi", {}, {}]

Instrument[Macintosh HD:Desktop Folder:NI Week Demos:Build:ImageCapture.vi, ImageCapture.vi, {}, {}]

OpenVIRef[link, inst]

ImageCapture.vi loaded

OpenPanel[link, inst]

ImageCapture.vi: panel displayed

RunVI[link, inst]

ImageCapture.vi running

ClosePanel[link, inst]

ImageCapture.vi: panel closed

ServerShutdown[link]

Server stopped

<< ImageProcessing`

capture = ImageRead["capture2.bmp"];

Show[Graphics[capture]];

Show[Graphics[PlanarImageData[capture]]];

hist = ImageHistogram[capture];

Show[
  GraphicsArray[
   ListPlot[#, PlotJoined → True, PlotRange → {0, 0.04}, Ticks → {{50, 100, 150, 200}, {0.02, 0.04}}, AspectRatio → 2/3,
      DisplayFunction → Identity] & /@ hist]];

Show[GraphicsArray[
  Graphics /@ {Threshold[
    EdgeMagnitude[ToGrayLevel[capture], SobelFilter[]], 32],
   ZeroCrossing[
    EdgeMagnitude[ToGrayLevel[capture], LoGFilter[21, 21]]]}]];

grayscl = ToGrayLevel[ScaleLinear[Log[2., 1. + capture], {0, 255}]];

regofint = Polygon[Reverse /@ Round[{{90, 70}, {90, 165}, {165, 70}, {165, 165}}]]

Polygon[{{70, 90}, {165, 90}, {70, 165}, {165, 165}}]

Show[Graphics[RegionProcessing[EdgeMagnitude[#1, SobelFilter[]] &, grayscl, regofint]]];
```

Figure 3. Initial investigations undertaken in a Mathematica Notebook.

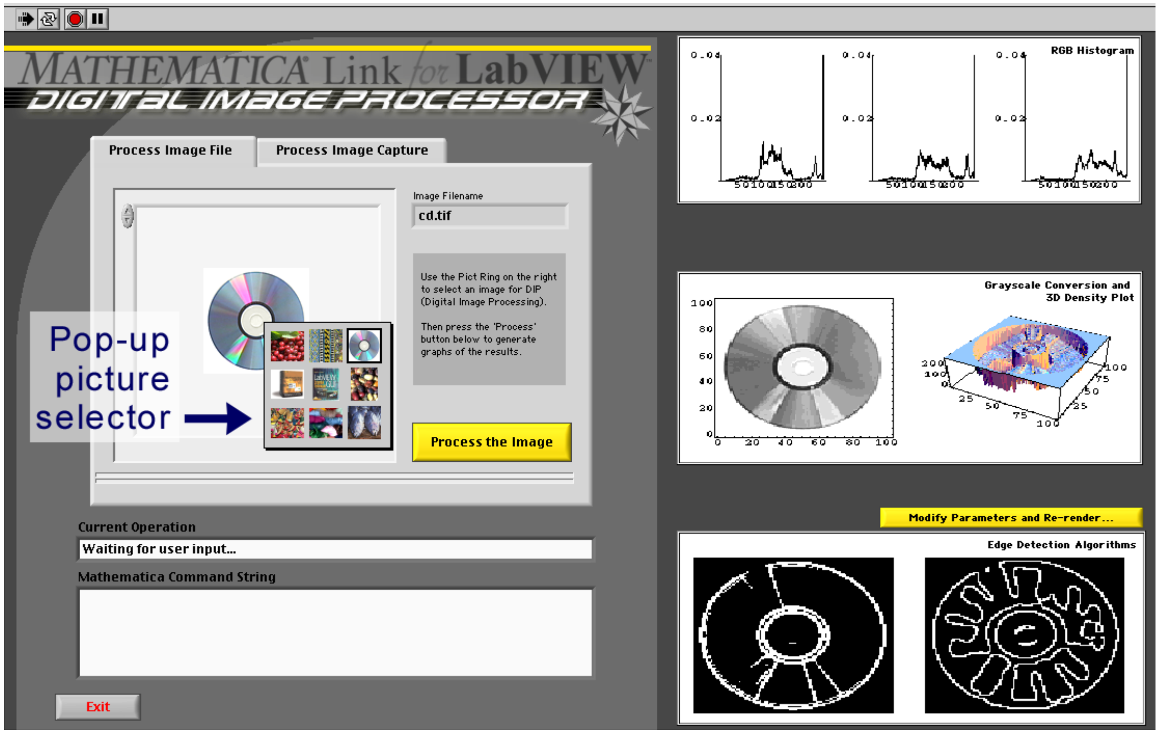

MATHEMATICA Link for LabVIEW
DIGITAL IMAGE PROCESSOR
Process Image File
Process Image Capture
Pop-up picture selector
Process the Image
Exit
RGB Histogram
Grayscale Conversion and 3D Density Plot
Modify Parameters and Re-render...
Edge Detection Algorithms

Image Filename
cd.tif

Use the Pict Ring on the right to select an image for DIP (Digital Image Processing).

Then press the 'Process' button below to generate graphs of the results.

Current Operation
Waiting for user input...

Mathematica Command String

Figure 4. Running the VI in ‘Process Image File’ mode.

“Process Image File” mode:

When user selects the first mode, "Process Image File", he or she is presented with a Pict Ring control with representations of the image files available for processing. After selecting an image, the user simply presses the Process button. The analysis and plotting sequence is initiated. The demo performs four operations on the target image:

1) It generates three histograms, one each for the R, G, and B (red, green, and blue) channels in the image.
2) It converts the color image to grayscale and displays this grayscale representation.
3) It uses the grayscale data to generate a 3D density plot of the image.
4) It applies two edge detection filters to the image -- a Sobel gradient filter, and a Laplacian-of-Gaussian filter , then plots the results.

As reflected in the figure, the results of each step are displayed on the right-hand side of the panel as they are completed.

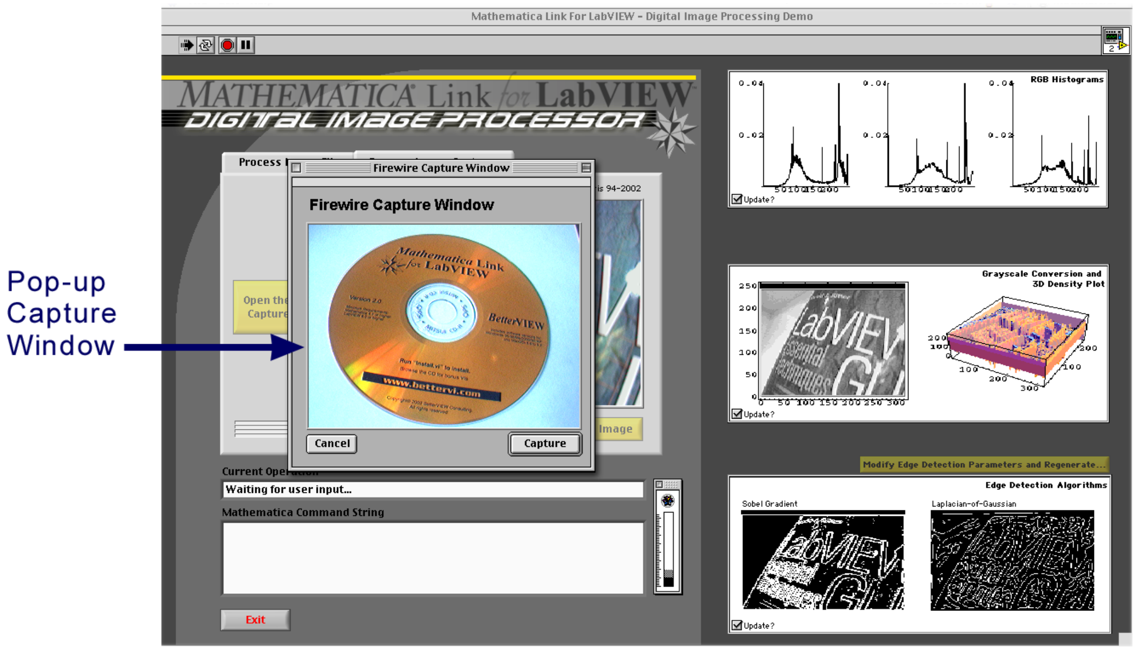

Pop-up Capture Window
Mathematica Link For LabVIEW - Digital Image Processing Demo
MATHEMATICA Link for LabVIEW
DIGITAL IMAGE PROCESSOR
Firewire Capture Window
Cancel
Capture
RGB Histograms
Grayscale Conversion and 3D Density Plot
Modify Edge Detection Parameters and Regenerate...
Edge Detection Algorithms
Sobel Gradient
Laplacian-of-Gaussian
Update?
Current Operation
Waiting for user input...
Mathematica Command String
Exit

Figure 5. Running the VI in 'Process Image Capture' mode.

"Process Image Capture" mode:

If the user selects the second mode, "Process Image Capture", the Quicktime capture window is presented. When a Firewire camera is present, this window remains open displaying a 'live' camera image. When the user clicks the 'Capture' button, the image is captured for subsequent processing. After capturing a suitable image, the user simply clicks the 'Process' button to initiate the processing step, just as in the "Process Image File" mode outlined previously.

Additional Comments:

The Mathematica image processing and plotting commands are hard-wired into the LabVIEW diagrams. However, by simply changing the commands, the operation of this VI can be modified. In other words, no rewiring is necessary to completely alter the behavior of this VI!

The basic structure for this VI is a state machine. User interaction takes place in a single case, as do image processing, initialization, and error handling. The high-level "Generic Plot.vi" passes the plot requests to Mathematica and converts Mathematica's Postscript graphics output into bitmaps that can be displayed in the LabVIEW Intensity Graphs on the right-hand side of the main VI panel. As noted in the Mathematica Link for LabVIEW User's Guide, conversion of Mathematica graphics to bitmaps is a multi-stage process. First, Mathematica processes the plot request and generates a Postscript graphic. This Postscript graphic is temporarily saved to disk as a graphic file to enable the conversion.

Next, an executable called 'MLPost' is called to convert the Postscript file into bitmap data and save it to a temporary bitmap file. Finally, this bitmap must be rendered in the LabVIEW panel. While this multi-step process may seem tedious to LabVIEW programmers that are accustomed to working entirely in the bitmap domain, the advantage of Mathematica's Postscript output is that it offers superior hard-copy output from Postscript-compatible printers. (Naturally, Mathematica Link for LabVIEW offers tools for outputting the Postscript data directly in printer-friendly formats.)

Feel free to examine the VI diagrams to see how various Mathematica Link for LabVIEW components were combined to realize this demo.

MacOS-specific Elements:

This demo was developed to run on MacOS. As a result there are a couple of Mac-specific features and components worth mentioning.

- The Quicktime Image Capture subVI, based on Chris Salzmann's Quicktime VIs, can capture images using any Quicktime-compatible video source, including Firewire camcorders and USB webcams.

- Because LabVIEW is not yet available on MacOSX, this demo was constructed to run under MacOS 9.2. Unfortunately, MacOS 9.2 doesn't employ a preemptive multitasking scheme, and the Mac's cooperative multitasking implementation assigns the highest execution priority to the foreground application—background processes have substantially less time to execute than the foreground application. When running Mathematica Link for LabVIEW, switching the top-most application based on the current processing step can noticeably improve performance. Therefore, Applescript was used in various places to switch between LabVIEW, the Mathematica Kernel, and MLPost during processing. This achieved the desired objective of optimizing Link performance under MacOS 9.2.