iLab C++ Neuromorphic Vision Toolkit Overview

**Components:**
- Basic image processing and vision
- Attention-related neural components
- Object recognition-related neural components
- Scene gist/layout-related neural components
- Basic knowledge base / ontology
- Hardware interfacing
- Beowulf message passing
- Applications

**Implementation:**
- C++, somewhat Linux-specific
- Additional perl/matlab/shell scripts for batch processing
- Uniprocessor as well as Beowulf
The basic architecture

• The diagram on the next slide is an overview of this computational neuroscience model.
• Suggested readings: see http://iLab.usc.edu/publications/

• Start with Itti & Koch, Nature Reviews Neuroscience, 2001, for an overview.
• Then see Itti, Koch and Niebur, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1998, for the core algorithm.
• Then see Itti & Koch, Vision Research, 2000 and Itti & Koch, Journal of Electronic Imaging, 2001, for more advanced competition for salience.

• See papers by Christian Siagian, Christopher Ackerman & Nitin Dhavale for more on robotics applications, gist, and localization.
• See papers by Vidhya Navalpakkam, Rob Peters and Lior Elazary for more on scene understanding & top-down biasing.
• See papers by Nathan Mundhenk for more on contour integration, surprise
• Etc...
Welcome to the iLab Neuromorphic Vision C++ Toolkit (iNVT)!

The iLab Neuromorphic Vision C++ Toolkit (iNVT, pronounced “invent”) is a comprehensive set of C++ classes for the development of neuromorphic models of vision. Neuromorphic models are computational neuroscience algorithms whose architecture and function is closely inspired from biological brains. The iLab Neuromorphic Vision C++ Toolkit comprises not only base classes for images, neurons, and brain areas, but also fully-developed models such as our model of bottom-up visual attention and of Bayesian surprise.

Features at a glance:

- The source tree is maintained using the Subversion (SVN) revision control system.
- The main development platform is Linux. However, the core programs also compile under Windows (using cygwin) and MacOS X.
- All source code is distributed freely under the GNU General Public License. Registered users get access to our central SVN source code repository and hence receive updates in real-time, not only when we make major releases.
- Low-level helper classes, including Point2D, Rectangle, PixRGB<T>, Range, Timer, XWindow, etc.
- Template Image<T> and ImageSet<T> classes with hundreds of image processing functions and copy-on-write / ref-counting semantics.
Root: Image class

- Template class
  - e.g., Image<byte>, Image<PixRGB<float>>, Image<Neuron>

- Implemented using copy-on-write/ref-counting
  - Makes copying a light operation

- Many associated methods
  - Shape ops
  - Color ops
  - Mono only
  - Math ops
  - Matrix ops
  - I/O
  - Filter ops
  - Transforms
C++ Templates

- **The old way:** ByteImage, FloatImage, ColorImage, etc. yields lots of duplicated code that achieves essentially the same operations.

- **The C++ way:** write your algorithm only once, and make it operate on an unknown data type T. The compiler will then generate machine code corresponding to your algorithm and various data types for T, such as, T=byte, T=float, T=MyClass, etc.

```cpp
template <class T> class Image {
  public:
    Image();
    T getPixelValue(const int x, const int y) const;
    void setPixelValue(const T& value, const int x, const int y);
  private:
    T* data;
};

int main(const int argc, const char **argv) {
  Image<float> myImage; myImage.setPixelValue(1.23F, 10, 10);
  return 0;
}
```

See src/Image/Image.H
Operator overloads

- C++ allows you to define operators such as +, -, *, etc for your various classes.

Example:

```cpp
Image<byte> img1, img2;

img1 += 3;  // calls Image<T>::operator+=(const T& value)

img1 = img1*2 + img2/3;  // calls operator*(const T& value), operator/(const T& value), and operator+(const Image<T>& im)
```

See Image/Pixels.H, Image.H
Automatic type promotions

• Using type traits to determine at compile time whether the result of an arithmetic operation will fit in the same type as the operands.

• Extends the canonical C++ promotions to non-canonical types.

• Examples:

Image<byte> im;

im + im is an Image<int>
im * 2.0F is an Image<float>
im * 2.0 is an Image<double>

Automatic type demotion with clamping

• Assignment from a strong type into a weak type will ensure that no overflow occurs.

• Example:

```cpp
Image<byte> im1, im2;  Image<float> im3;

im1 = im3;  // will clamp values of im3 to 0..255 range and convert

im2 = im1 * 2.0;  // will create an Image<double> containing the
                 // result of im1 * 2.0, then clamp this image to
                 // 0..255 pixel range, then assign to im2.
```
Automatic type demotion with clamping

- Promotion rules (in Util/Promotions.H):
  Basically follow the C/C++ canonical promotions

  - byte, byte -> int; Byte, int16 -> int; int16, int16 -> int; etc...
  - int, int -> int
  - byte, float -> float; int, float -> float; float, float -> float, etc...
  - byte, double -> double; int, double -> double; float, double -> double, etc...
Copy-on-write / ref counting

- The standard way:

Image object contains an array of pixels:

```
int width, height;
T* data;
```

Problem: copy is expensive, need to copy the whole data array (can be large, e.g., a 16MP RGB image uses 48MB of memory).
Copy-on-write / ref counting

In particular, this makes it very expensive to return Image objects from functions, hence essentially forbidding the natural syntax:

```cpp
Image<float> source;
Image<float> result = filter(source);  // With a function:

Image<float> filter(const Image<float>& source) {
    Image<float> res;
    // fill-up pixel values of res, processing values from source
    return res;
}
```

Indeed what happens here is:
1) Inside `filter()`, allocate a new image `res` to hold the result
2) In the ‘return’ statement, copy that local image to some temporary
3) In the ‘=` statement, copy that temporary to Image ‘result’
Copy-on-write / ref counting

• The smart way: only keep a pointer to the actual pixel data in each Image object. When making copies of the Image object, keep track of how many are pointing to the same pixel data. When the last Image object is destroyed, free the pixel data. If the user attempts to modify the contents of one of the images that point to the same data, first make a copy of the data.

```cpp
Image<byte> img1, img2, img3;   img2 = img1; img3 = img1;
```

See Image/ArrayData.H, Image.H
Free functions rather than methods

• Given the copy-on-write mechanism, it is now very cheap to return Image objects. Thus, the more natural ‘free function’ syntax may be used for most image processing functions, instead of the ‘class method’ syntax.

• Example: let’s say I want to pass an image through 3 successive filters, filter1(), filter2() and filter3():

**Class method syntax:** the filterX() are methods of class Image

```cpp
const Image<float> source;
Image<float> result1, result2;
result1.filter1(source);
result2.filter2(result1);
result1.filter3(result2);
result2.freeMem();
```

**Free function syntax:** the filterX() are functions not attached to a class

```cpp
const Image<float> source;
Image<float> result = filter3(filter2(filter1(source)));
```

See Image/* .H
Iterators

- Accessing data via pointers is error-prone, use iterators instead. Our classes that hold some data that can be iterated on provide iterator support very similar to that of the STL classes.

- Example:

```c++
Image<byte> img;

Image<byte>::iterator itr = img.beginw(), stop = img.endw();
while (itr != stop) {
    *itr++ = 0;
}
```

See Image/Image.H
**Shared pointers**

- When objects communicate with lots of other objects, it is often difficult to know who will run out of scope first. When new memory is allocated for an object that will be passed around and used by several objects, we would like an automatic way of freeing the memory when everybody is done with it.
- Hence the class `shared_ptr<T>` which behaves like a pointer, except that when the last `shared_ptr` to an object runs out of scope, it will destroy/free the memory for that object.

**Example:**

In `obj1`: `SharedPtr<Message> mymsg(new Message());`

In `obj2`: `SharedPtr<Message> mymsg2(mymsg);`

`mymsg2->function();`

Message will be destroyed only when its `SharedPtr`’s have run out of scope in both `obj1` and `obj2`.  

See rutz/shared_ptr.h,  
Also nub/ref.h

iLab C++ Neuromorphic Toolkit
Elementary core classes

- **Dims**: for 2D (width, height) dimensions
  - Dims.H
- **Point2D<T>**: An (i, j) 2D point
  - Point2D.H
- **PixRGB<T>**: a (red, green, blue) triplet
  - Pixels.H
- **Timer**: to count time with arbitrary accuracy
  - Timer.H
- **CpuTimer**: to measure time and CPU load
  - CpuTimer.H
- **Range**: specifies a numeric range of values
  - Range.H
- **LevelSpec**: specifies scales for feature/saliency map
  - LevelSpec.H
- **Rectangle**: a rectangle
  - Rectangle.H
- **Angle**: an angle
  - Angle.H
- **shared_ptr<T>**: a shared pointer
  - shared_ptr.h
- **VisualEvent**
- **VisualObject**
- **VisualFeature**
- **...**
Core definitions

- **Promotions.H**: the automatic type promotion rules
- **atomic.H**: atomic (one-CPU-instruction) operations
- **Mathfunctions.H**: basic math functions
- **JobServer.H / WorkThreadServer.H**: multithreading support
- **Log.H**: comprehensive logging facility
- **StringConversions.H**: convert various datatypes to/from string
- **StringUtil.H**: various string manipulation utilities (e.g., tokenize)
- **sformat.H**: like sprintf for std::string
- **TypeTraits.H**: compile-time information about types
- ...
Logs

• Provide a unified, convenient mechanism for text message output.
• 4 levels: LDEBUG, LINFO, LERROR, LFATAL
• printf()-like syntax
• Automatically adds class/function name, system error messages (use prefix ‘P’), a user id (use prefix ‘ID’), a line number (compile-time option)
• Can print to stderr or syslog

The hard way:
fprintf(stderr, “In myFunction(), could not open file ‘%s’ (error: %s)\n”, filename, strerror(errno));

>>> In myFunction(), could not open file `test’ (error: file not found)

The easy way:
PLERROR(“Could not open file ‘%s ’, filename);

>>> MyClass::myFunction: Could not open file `test’ (file not found)
**Helper classes**

- **Raster**: to read/write/display Images in various formats
- **V4Lgrabber**: to grab images from video source (PCI/USB)
- **XWindow**: to display image collections & interact
- **FrameIstream, FrameOstream, FrameSeries**: easily read images from image files, movies, cameras, etc
- **Etc...**
## src/ directories

<table>
<thead>
<tr>
<th>AppDevices/</th>
<th>Channels/</th>
<th>GUI/</th>
<th>nub/</th>
<th>SceneUnderstanding/</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppEye/</td>
<td>CINNIC/</td>
<td>HMAX/</td>
<td>ObjRec/</td>
<td>Script/</td>
</tr>
<tr>
<td>AppGUI/</td>
<td>CMapDemo/</td>
<td>Ice/</td>
<td>Parallel/</td>
<td>SeaBee/</td>
</tr>
<tr>
<td>AppMedia/</td>
<td>CmuCam/</td>
<td>Image/</td>
<td>plugins/</td>
<td>SIFT/</td>
</tr>
<tr>
<td>AppNeuro/</td>
<td>Component/</td>
<td>inst@</td>
<td>Simulation/</td>
<td></td>
</tr>
<tr>
<td>AppPsycho/</td>
<td>Controllers/</td>
<td>INVT/</td>
<td>PointCloud/</td>
<td>Surprise/</td>
</tr>
<tr>
<td>Apps/</td>
<td>Corba/</td>
<td>Landmark/</td>
<td>Psycho/</td>
<td>tcl/</td>
</tr>
<tr>
<td>ArmControl/</td>
<td>Demo/</td>
<td>Learn/</td>
<td>Qt/</td>
<td>TestSuite/</td>
</tr>
<tr>
<td>Audio/</td>
<td>Devices/</td>
<td>Matlab/</td>
<td>QtUtil/</td>
<td>TIGS/</td>
</tr>
<tr>
<td>Beobot/</td>
<td>Envision/</td>
<td>MBARI/</td>
<td>Raster/</td>
<td>Transport/</td>
</tr>
<tr>
<td>BeoSub/</td>
<td>extra/</td>
<td>Media/</td>
<td>RCBot/</td>
<td>Util/</td>
</tr>
<tr>
<td>Beowulf/</td>
<td>Foveator/</td>
<td>ModelNeuron/</td>
<td>Robots/</td>
<td>VFAT/</td>
</tr>
<tr>
<td>BO/</td>
<td>GA/</td>
<td>NeovisionII/</td>
<td>rutz/</td>
<td>Vgames/</td>
</tr>
<tr>
<td>BPnnet/</td>
<td>GameBoard/</td>
<td>Nerdcam/</td>
<td>SalGlasses/</td>
<td>Video/</td>
</tr>
<tr>
<td>Cell/</td>
<td>Gist/</td>
<td>Neuro/</td>
<td>Sbqa/</td>
<td></td>
</tr>
</tbody>
</table>

---

iLab C++ Neuromorphic Toolkit

Tuesday, July 20, 2010
ImageSets, a.k.a. Image Pyramids

- Collection of images
- Dyadic image reduction from one level to next
- Various filters applied before reduction

Gaussian Pyramid

Idea: Represent NxN image as a “pyramid” of 1x1, 2x2, 4x4,..., $2^k \times 2^k$ images (assuming $N=2^k$)
Channels

- Implement a pyramid or collection of pyramids plus some I/O functions and additional processing

- Various derived instances can be identified by name

- SingleChannel: contains one pyramid
- ComplexChannel: contains a collection of SingleChannels
Single Channels

ChannelBase

SingleChannel

BlueYellowChannel  FlickerChannel  GaborChannel  IntensityChannel  RedGreenChannel  TemplateMatchChannel

PedestrianChannel

ChannelBase

type_with_N_bits<T, N>

TAKE_ABS
NORMALIZED_OUTPUT
NO_OPTS

SingleChannel
Complex channels
VisualCortex

- Run-time configurable collection of channels, plus additional I/O and access methods
Brain

VisualCortex plugged-in at run-time

iLab C++ Neuromorphic
Brain: basic operation

Data flow is controlled by a **blackboard architecture**: Brain modules can post messages and can register **callbacks** which will be called when some messages are posted by other modules.

Processing flow is driven by reading new input images (stream oriented):

- Get an input image
- Process it through VisualCortex, get saliency map input
- Feed saliency map
- Let saliency map evolve
- Let task-relevance map evolve
- Combine saliency map and task-relevance map outputs to feed attention-guidance map
- Let attention-guidance map evolve
- Feed output of attention-guidance map to winner-take-all
- Get winner-take-all output, if any
- Feed that to saccade controller
- Also feed it to shape estimator
- Activate inhibition of return
- ...

iLab C++ Neuromorphic Toolkit
Welcome to iLab at the University of Southern California!

Research

People

Classes

Publications

Facilities

Opportunities

Events + Links
Welcome to the Beobot Project!

Beobots are autonomous robots whose brains are standard Linux clusters of computers which run real-time neuromorphic vision algorithms.

Just like Beowulf Clusters have revolutionized the world of high-performance computing, replacing costly and slowly-evolving custom supercomputer hardware by assemblies of inexpensive, mass-produced personal computers, we hope that Beobots (a Beowulf cluster on a mobile robot) will lead the way towards a new generation of robotics systems that are inexpensive, rapidly evolving, built from standard mass-produced components, and armed with sufficient computational power to run real-time neuromorphic vision algorithms.

- So what exactly is a Beobot?
- What hardware is it made of?
- What software does it run?
- Who are the people working on it?
## Recent CVS / Forum Activity

Ordered by last CVS commit date/time.

<table>
<thead>
<tr>
<th>User</th>
<th>Last CVS Commit</th>
<th>Last iLab Forum Post</th>
</tr>
</thead>
<tbody>
<tr>
<td>daesu</td>
<td>2003-05-13 at 14:40 saliency/src3/test-roadShape.C</td>
<td>---</td>
</tr>
<tr>
<td>beobot</td>
<td>2003-02-20 at 21:56 saliency/bin/bbsync</td>
<td>---</td>
</tr>
<tr>
<td>rhirata</td>
<td>2002-10-23 at 14:54 beobots/software/gyro/Gyro2.C</td>
<td>---</td>
</tr>
<tr>
<td>jsn</td>
<td>2001-12-10 at 13:16 beobots/software/lcd/lcd.C</td>
<td>---</td>
</tr>
<tr>
<td>juliet</td>
<td>---</td>
<td>Sun Jun 1 23:25:23 2003</td>
</tr>
<tr>
<td>aprilla</td>
<td>---</td>
<td>Mon Jun 3 01:53:33 2002</td>
</tr>
</tbody>
</table>

## Latest CVS commits

Ordered by commit date/time.

<table>
<thead>
<tr>
<th>Date</th>
<th>Time (PST)</th>
<th>User</th>
<th>Version</th>
<th>File</th>
<th>Log Message</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Done**
Welcome to the iLab Neuromorphic Vision C++ Toolkit. This toolkit is being developed at USC and Caltech to provide a comprehensive suite of C++ classes for image processing, biologically-inspired computational vision, hardware interfacing, and parallel processing on Linux beowulf clusters. Among the executables included in the distribution are our bottom-up, saliency-based visual attention model (single-CPU and beowulf versions), models of contour integration in primary visual cortex, models of object recognition in inferotemporal cortex, and our beebots quad-CPU autonomous neuromorphic robot code. Please see the README for installation and compilation instructions.

The main program to use is called evision in the saliency/bin/ directory.

See the README and programmer notes for detailed information on building and using the programs in the toolkit.

Here are links to several key classes in the toolkit:

- Image
- ModelComponent
- ModelManager
- ChannelBase
- VisualCortex
- Brain
- InputFrameSeries
- OutputFrameSeries

Other pages of interest:
Hey, Laurent Itti, you have 7 messages.
Sep 2nd, 2003, 1:20am

Forum name | Topics | Posts | Last post
-----------|--------|-------|-----------
**General** |        |       |           
News        | 17     | 25    | Apr 25th, 2003, 1:46pm by Laurent Itti
Openings    | 2      | 3     | Dec 3rd, 2002, 8:46am by Laurent Itti

**C++ Neuromorphic Vision Toolkit** |        |       |           
General Discussion | 35     | 258   | Aug 30th, 2003, 2:19am by lynnxx
Bugs          | 32     | 207   | Aug 29th, 2003, 12:18pm by yamin
Feature Requests | 22     | 155   | Aug 31st, 2003, 7:44pm by zhanshi
Neuroscience Issues | 4      | 59    | Oct 31st, 2002, 2:44pm by Dirk Walther
Architecture Issues | 4      | 72    | Jun 28th, 2003, 12:00pm by zhanshi
<table>
<thead>
<tr>
<th>Subject</th>
<th>Started by</th>
<th>Replies</th>
<th>Views</th>
<th>Last post</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Windows as command-line option</td>
<td>zhanshi</td>
<td>14</td>
<td>142</td>
<td>Aug 31(^{st}), 2003, 7:44pm by zhanshi</td>
</tr>
<tr>
<td>IEEE1394 update</td>
<td>laurent itti</td>
<td>6</td>
<td>76</td>
<td>Aug 30(^{th}), 2003, 9:27pm by zhanshi</td>
</tr>
<tr>
<td>STL, doxygen, and graphviz</td>
<td>zhanshi</td>
<td>4</td>
<td>53</td>
<td>Aug 29(^{th}), 2003, 10:43am by laurent itti</td>
</tr>
<tr>
<td>Added PNG write capability</td>
<td>Rob Peters</td>
<td>1</td>
<td>16</td>
<td>Jul 23(^{rd}), 2003, 6:34pm by laurent itti</td>
</tr>
<tr>
<td>Methods for computing orientations</td>
<td>Dirk Walther</td>
<td>7</td>
<td>71</td>
<td>Apr 28(^{th}), 2003, 11:12am by laurent itti</td>
</tr>
<tr>
<td>Dust off the Raster interface?</td>
<td>Rob Peters</td>
<td>11</td>
<td>88</td>
<td>Mar 21(^{st}), 2003, 5:49pm by Rob Peters</td>
</tr>
<tr>
<td>Pyramids; LOGVERB+FULLTRACE</td>
<td>Rob Peters</td>
<td>22</td>
<td>185</td>
<td>Mar 8(^{th}), 2003, 4:53pm by laurent itti</td>
</tr>
<tr>
<td>tests that take a long time</td>
<td>laurent itti</td>
<td>11</td>
<td>96</td>
<td>Mar 6(^{th}), 2003, 4:29pm by laurent itti</td>
</tr>
<tr>
<td>threadsafe refcounting</td>
<td>laurent itti</td>
<td>4</td>
<td>54</td>
<td>Jan 25(^{th}), 2003, 11:20am by laurent itti</td>
</tr>
<tr>
<td>Detection of targets by biasing features</td>
<td>vidhya</td>
<td>1</td>
<td>57</td>
<td>Jan 13(^{th}), 2003, 5:20pm by laurent itti</td>
</tr>
<tr>
<td>itsLevels in PyramidBase</td>
<td>Dirk Walther</td>
<td>4</td>
<td>67</td>
<td>Dec 3(^{rd}), 2002, 11:44am by Dirk Walther</td>
</tr>
</tbody>
</table>
Welcome to the iLab Publication Server!

115 publications, 73 with abstract, 55 available as PDF.

Publications by Year

- in-press
- 2003
- 2002
- 2001
- 2000
- 1999
- 1998
- 1997
- 1996
- 1995
- 1994
- 1992
- 1991
- 1990
- 1989

Publications by Type and by Theme

- All Publications
- Journal Articles
- Publications in Press
- Book Chapters
- Proceedings from International Conferences
- Master Theses
- Patents and Copyrights
- Ph.D. Theses
- Beobots
- Model of Bottom-Up Saliency-Based Visual Attention
- Computer Vision
- Human Eye-Tracking Research
- Functional Neuroimaging
- Medical Research
- Medical Image Processing
- Computational Modeling
- Press Coverage
- Human Psychophysics
- Review Articles and Chapters
This project was started at Caltech with Prof. Christof Koch. It is actively being pursued both here and at Caltech (both jointly and in different directions).

The Theory
Details about the trainable model of bottom-up, task-independent visual attention under development in our laboratory.

The Images
A short overview of example images and the corresponding attentional trajectories. Test images, psychophysical stimuli, target detection images, natural scenes, artwork, etc.

The Movies
Several MPEG movies showing attentional trajectories and the temporal dynamics of the Saliency Map for test, psychophysical, artistic and natural images. Also shown are 3D warping of the original image onto the evolving saliency map.

The Interactive Demo
An interactive demonstration of the dynamic behavior of our attentional model, for a variety of complete image databases. Most recent Java™-aware Web browser required.

The Publications
Some pre-versions of our papers describing this research are available in HTML, Postscript and PDF format.

The Ongoing Projects
New! Previews of a few of our ongoing projects and preliminary screenshots. These include our SaliencyVehicle off-road muscle car, our real-time SaliencyCam which computes attentional deployment on live video feeds (15 frames/s), our SaliencyAgent which detects salient pedestrians in natural color scenes, and other exciting projects.

The C++ Source Code
The C++ source code and associated doxygen documentation are available through our CVS server. You will need the latest version of g++ (3.x) and several non-standard packages installed on your Linux distribution (e.g., IEEE1394
# iLab Image Databases

These image databases are provided for testing and evaluation only. Some of the images in the databases have been grabbed from the web, and may be subject to copyright. So, do not use these images in any commercial application!

All images are in **PPM** (24-bit color) or **PGM** (8-bit greyscale) format, compressed with **bzlib2** and compiled in **tar** archives.

**Note:** We have put a lot of effort into making these databases available to you. By downloading any of the databases below, you agree to properly cite the associated master reference, which typically is the paper where we first described the database and used it with our model, and to provide a link to the present web page.

<table>
<thead>
<tr>
<th>Samples</th>
<th>Database</th>
<th># Images</th>
<th>Size</th>
<th>Description</th>
<th>Master Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STIMart.tar</td>
<td>20</td>
<td>4.0 MB</td>
<td>Miscellaneous artwork, posters and portraits</td>
<td>Itti et al., IEEE PAMI, 1998</td>
</tr>
<tr>
<td></td>
<td>STIMautobahn.tar</td>
<td>90 + 90</td>
<td>56 MB</td>
<td>Color images with German traffic signs + target masks</td>
<td>Itti &amp; Koch, J. Elec. Imag., 2001</td>
</tr>
<tr>
<td></td>
<td>STIMcoke.tar</td>
<td>104 + 104</td>
<td>53 MB</td>
<td>Color images with a red can + target masks</td>
<td>Itti &amp; Koch, J. Elec. Imag., 2001</td>
</tr>
</tbody>
</table>