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
Basic functionality

Find the most interesting location in the image (next slide)
The model’s prediction

Here is what our model of bottom-up, saliency-based attention found
(next slide)
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 Vidhya Navalpakkam for more on scene understanding
  • See papers by Nathan Mundhenk for more on contour integration
  • See papers by Nitin Dhavale for more on eye movements
  • See papers by Chris Ackerman for more on gist
  • Etc…
iLab C++ Neuromorphic Toolkit

Itti & Koch, Nature Reviews Neurosci 2001

Input image

Multiscale low-level feature extraction

Colours
Red, green, blue, yellow, etc.

Intensity
On, off, etc.

Orientations
0°, 45°, 90°, 135°, etc.

Other
Motion, junctions and terminators, stereo disparity, shape from shading, etc.

Attended location

Inhibition of return
Winner-take-all

Saliency map

Centre-surround differences and spatial competition

Feature maps

Feature combinations

Top-down attentional bias and training

Itti & Koch, Nature Reviews Neurosci 2001
Architecture
The 2002 E3 came and went, but here is a full report on the future trends set by the gaming industry. The gaming community, with its various salient interfaces and its much vaunted multi-million dollar booths that exhibited the current reigning consoles were captured in our exclusive footage. Presented through iLab, complete with predictions from our visual attention model (captured by yellow and green circles) and saliency maps (shown to the right of each video clip).

The purpose of this footage is to test the visual saliency algorithm on video games. iLab is investigating possible applications of the saliency software to the development of more sophisticated opponents. These opponents will use intelligent agents that would employ the iLab algorithms for visual perception. Testing is also done on the Beobot robotics platform to be released this year, with external components designed by yours truly.

At left are MPEG movie clips following line-up. Legend: Hawk’s Pro Skater 4. Sup visual saliency software attuned to even more specific characteristics (a specific color, for example), it is not the case here, so as to give each game the same evaluation. The saliency software also considers the ‘inhibition of return,’ which means that once a specific location has been visited, it will not return to the location before 1 to 2 seconds. Each movie clip is in real-time, and the software updates at 30 frames per second.
Visual Attention: Movies

In the first short MPEG video, we show both the original input image and the corresponding saliency map in dynamical evolution. The saliency map begins with charging up from the input. Then the most salient location is found and transiently inhibited. The yellow circle on top of the original image represents the current focus of attention.

The first image shows simple objects in noise; objects are selected by order of decreasing saliency (here luminance contrast). After all objects have been attended, less salient locations in the background noise are attended. The previously attended and inhibited locations progressively charge-up again and may be again attended.

Then, in the examples of pop-out targets, we show how significant the salience contrast is between the salient target and itself and the number of distractors. In the example of conjunctive targets, which are competing, the target does not pop out. In these examples, the attentional features are distributed and generated by a serial search model, which perform a serial search.

The Batman™ poster demonstrates how, generally, the attentional trajectories generated by the model seem to agree with our reading of the image. Note how a good coverage of the image is obtained (attention does not only go to the 3 faces, but looks around about everywhere), with frequent checking on the most salient objects. Good performance was also obtained with a database of traffic sign images from Daimler Benz, Inc. Since road signs have been designed to be salient (the car is also salient here), they are found before the system looks around in the trees.

We conclude this demonstration with an example of robustness to noise.

In this second video, the spatial competition among conspicuous locations within each feature map is demonstrated. Details about the implementation of such spatial competition can be found in our 2001 Journal of Electronic Imaging paper.
See here for an interactive demo
The big picture...
Low-level features:
oriented edges, color opponencies,
intensity contrast, motion energy,
stereo disparity, etc.

Proto-objects:
corners, T-junctions, simple
geometric shapes, etc.

Saliency map:
potentially interesting
objects, actors and
actions

Attention:
most interesting
locations

Localized object
recognition:
walking couple

Behavioral goal specification:
e.g., "look for people"

Cognitive scene understanding:
"a couple walks down the beach"

http://iLab.usc.edu/bu/
Eye/head Movement generation
How does task influence attention?

Low level features:
Oriented edges, Color opponencies, Intensity contrast, motion energy, Stereo disparity etc.

Gist:
Outdoor beach scene

Layout:
1. Grass
2. Sand
3. Sea
4. Sky

Bottom-up salience of locations

Top down task-relevance of locations

Task specification
"look for humans"

iLab C++ Neuromorphic Toolkit
Towards modeling the influence of task on relevance

**Gist:**
Outdoor beach scene

**Layout:**
1. Grass
2. Sand
3. Sea
4. Sky

**Task specification**
“look for humans”

**Knowledge of task-relevant entities and their spatial relations**

**Working Memory**
(Frontal cortex)

**Top down task-relevance of locations**

**Visual scene**

Torralba et al, JOSA-A 2003
Task specification
“look for legs”

Long Term Memory
(Ontology)

Working Memory
• Creates, maintains task graph
• Computes relevance of fixation
• Predicts location of other relevant entities

Agent
(relay information)

Low level features

Gist:
Outdoor Sports scene

Layout:
1. Sky
2. Trees
3. track

Saliency map

Visual Scene

Task Relevance Map

Inhibition of return

Localized object recognition
“head"

“head is relevant”
“look down to find legs”

iLab C++ Neuromorphic Toolkit
Attention Guidance Map

“look down to find legs”

• Creates, maintains task graph
• Computes relevance of fixation
• Predicts location of other relevant entities

“head is relevant”
“look down to find legs”

iLab C++ Neuromorphic Toolkit
Attention Guidance Map

“look down to find legs”

Agent
(relay information)
**Task specification**

“look for legs”

**Long Term Memory (Ontology)**

**Working Memory**
- Creates, maintains task graph
- Computes relevance of fixation
- Predicts location of other relevant entities

**Agent**

- Saliency map
- Task Relevance Map
- Attention Guidance Map
- Localized object recognition
- “legs”

**Visual Scene**

-Gist:
  - Outdoor Sports scene

- Layout:
  1. Sky
  2. Trees
  3. track

**Low level features**

- Not implemented
- Implemented

**Inhibition of return**

“target found”

**iLab C++ Neuromorphic Toolkit**

**Attention Guidance Map**
Object recognition
Top-down biasing to guide attention towards Known objects

Unbiased

Biased for coke cans
Real-time processing
On Beowulf clusters

http://iLab.usc.edu/beo/
Beowulf + robot = “Beobot”

- Two ROCKY-3742EYFG motherboards (dual-P-III, Gigabit Ethernet, 10/100 Ethernet, IEEE-1394 (FireWire), on-board video and sound, many additional integrated peripherals)
- 512MB RAM memory
- Two color video cameras
- 10/100 Ethernet connection for programming & testing
- Custom backplane with battery power management
- IBM 1GB microdrive or flash hard-drive
- Lithium battery packs
- Two Pentium-III 1GHz CPUs + coolers
- IBM 1GB microdrive or flash hard-drive
- Lithium battery packs
- Serial servo controller
- Traxxas E-Maxx 4WD chassis (stiffened suspension, lowered gear ratio, foam-filled tires)
http://iLab.usc.edu/beobots/
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
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
  - 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 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 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.
```

iLab C++ Neuromorphic Toolkit
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 array.
Copy-on-write / ref counting

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

```
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 `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:

```cpp
Image<byte> img;

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

See 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 `SharedPtr<T>` which behaves like a pointer, except that when the last `SharedPtr` 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();`

  See `SharedPtr.H`

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

- **Dims**: for 2D (width, height) dimensions  
  Dims.H
- **Point2D**: An \((i, j)\) 2D point  
  Point2D.H
- **Point2DT**: A Point2D plus a time  
  Point2DT.H
- **PixRGB\(\langle T\rangle\)**: a (red, green, blue) triplet  
  Pix.H
- **BitObject**: object defined by connected pixels  
  BitObject.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
- **SharedPtr\(\langle T\rangle\)**: a shared pointer  
  SharedPtr.H
- **VisualEvent**
- **VisualObject**
- **VisualFeature**
- ...
Core definitions

- **Promotions.H**: the automatic type promotion rules
- **atomic.H**: atomic (one-CPU-instruction) operations
- **Saliency.H**: a few generic helper functions like MAX, MIN, etc and basic type definitions like byte, int32, uint64, etc
- **colorDefs.H**: various default color definitions
- **Log.H**: comprehensive logging facility
- **StringConversions.H**: convert various datatypes to/from 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:

```c
fprintf(stderr, "In myFunction(), could not open file ‘%s’ (error: %s)\n", filename, strerror(errno));
```

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

The easy way:

```c
PLERROR("Could not open file ‘%s ’, filename);
```

```c
>>> MyClass::myFunction: Could not open file `test’ (file not found)
```

See log.H
Helper classes

- **Raster**: to read/write/display Images in various formats

- **V4Lgrabber**: to grab images from video source (PCI/USB)

- **IEEE1394grabber**: idem for FireWire cameras

- **XWindow**: to display image collections & interact

- **VCC4**: to control pan/tilt/zoom camera

- **SSC**: to control pan/tilt on beobot camera

- Etc…
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 $N \times N$ image as a "pyramid" of $1 \times 1$, $2 \times 2$, $4 \times 4$, ..., $2^k \times 2^k$ images (assuming $N = 2^k$)

levels: 0, 1, 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
Complex channels
VisualCortex

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

VisualCortex plugged-in at run-time
Brain: basic operation

In Brain::input(), called for every new input image
  • Get an input image
  • Process it through VisualCortex, get saliency map input

In Brain::evolve(), called every 0.1ms of simulated time
  • 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
  • …
Beowulf

- Multi-threaded class

- Handles transparent passing of TCP messages
  - TCP messages are run-time collections of objects

- TCP messages implemented using COW

- Uses TCP communications for distant nodes

- Uses shared memory for local nodes
Beobot

iLab C++ Neuromorphic Toolkit
FrameGrabber, etc, etc
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>rjpeters</td>
<td>2003-08-02 at 11:06</td>
<td>Wed Jul 23 18:03:45 2003</td>
</tr>
<tr>
<td>vidhya</td>
<td>2003-06-25 at 01:03</td>
<td>Sun Jan 12 11:43:12 2003</td>
</tr>
<tr>
<td>daesu</td>
<td>2003-05-13 at 14:40</td>
<td>---</td>
</tr>
<tr>
<td>beobot</td>
<td>2003-02-20 at 21:56</td>
<td>---</td>
</tr>
<tr>
<td>rhirata</td>
<td>2002-10-23 at 14:54</td>
<td>---</td>
</tr>
<tr>
<td>jsn</td>
<td>2001-12-10 at 13:16</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>
Hey, Laurent Itti, you have 7 messages.
Sep 2nd, 2003, 1:20am

Forum name | Topics | Posts | Last post
---|---|---|---
**General**
News
Read about the latest happenings of iLab
Moderators: Forum Admin, Laurent Itti
17 | 25 | Apr 25th, 2003, 1:46pm by Laurent Itti
Openings
Find openings for positions at iLab
Moderators: Forum Admin, Laurent Itti
2 | 3 | Dec 3rd, 2002, 8:46am by Laurent Itti

**C++ Neuromorphic Vision Toolkit**
General Discussion
General discussion around the iLab C++ Neuromorphic Vision Toolkit
Moderators: Forum Admin, Laurent Itti
35 | 258 | Aug 30th, 2003, 2:19am by lynix
Bugs
Bugs and other problems
Moderators: Forum Admin, Laurent Itti
32 | 207 | Aug 29th, 2003, 12:18pm by yamini
Feature Requests
Feature Requests
Moderators: Forum Admin, Laurent Itti
22 | 155 | Aug 31st, 2003, 7:14pm by shenshi
Neuroscience Issues
Discussion of neuroscience issues and their implementation in the toolkit
Moderators: Forum Admin, Laurent Itti
4 | 59 | Oct 31st, 2002, 2:44pm by Dirk Walther
Architecture Issues
Discussion of general architecture issues, in particular regarding the abstraction of brain operating system into CPR devices
4 | 72 | Jun 28th, 2003, 12:00pm by yamini
<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 31st, 2003, 7:14pm by zhanshi</td>
</tr>
<tr>
<td>IEEE1394 update</td>
<td>Laurent Itti</td>
<td>6</td>
<td>75</td>
<td>Aug 30th, 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 29th, 2003, 10:43am by Laurent Itti</td>
</tr>
<tr>
<td>Added PNG write capability</td>
<td>Rob Peters</td>
<td>1</td>
<td>15</td>
<td>Jul 23rd, 2003, 6:34pm by Laurent Itti</td>
</tr>
<tr>
<td>Methods for computing orientations</td>
<td>Dirk Walkher</td>
<td>7</td>
<td>71</td>
<td>Apr 28th, 2003, 11:12am by Laurent Itti</td>
</tr>
<tr>
<td>Dust off the Raster interface?</td>
<td>Rob Peters</td>
<td>11</td>
<td>68</td>
<td>Mar 21st, 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 6th, 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>95</td>
<td>Mar 6th, 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 25th, 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 13th, 2003, 5:20pm by Laurent Itti</td>
</tr>
<tr>
<td>itsLevels in PyramidBase</td>
<td>Dirk Walkher</td>
<td>4</td>
<td>67</td>
<td>Dec 3rd, 2002, 11:44am</td>
</tr>
</tbody>
</table>
Welcome to the iLab Publication Server!

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

Publications by Year

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
Now! 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 **bzip2** 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><img src="image" alt="STIMart.tar" /></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><img src="image" alt="STIMautobahn.tar" /></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><img src="image" alt="STIMcoke.tar" /></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>
<tr>
<td><img src="image" alt="STIMcolor.tar" /></td>
<td>STIMcolor.tar</td>
<td>180 + 180</td>
<td>2.1 MB</td>
<td>Color popout search arrays + target masks</td>
<td>Itti &amp; Koch, Vis. Res., 2000</td>
</tr>
<tr>
<td><img src="image" alt="STIMoricol.tar" /></td>
<td>STIMoricol.tar</td>
<td>180 + 180</td>
<td>2.1 MB</td>
<td>Orientation/color conjunctive search arrays + target masks</td>
<td>Itti &amp; Koch, Vis. Res., 2000</td>
</tr>
</tbody>
</table>