

About These Slides

These slides were assembled for the Seventh Annual Student Research Symposium in Cumnock Hall on April 25, 2006. This is the second iteration of the SRS slides with updates since the previous 2004 SRS. The details contained within these slides were meant to fit on a three-fold posterboard.

The information regarding Phission was current at the time but have changed slightly since the symposium was held. The differences are in the “Future Work” section since development work continues, pushing Phission's capabilities beyond it's previous features.

Some of the text in the slides assumes some knowledge of C/C++ and the related topics therein.

Thank you for your interest in Phission.

What is Phission?

Phission is a software toolkit developed with the following design attributes in mind:

- ◆ Simple,
- ◆ Modular,
- ◆ Flexible,
- ◆ Extensible,
- ◆ Portable

Phission is used to quickly construct concurrent data processing and analysis systems on mobile robots or general purpose computers. The parallel architecture and internal synchronization make use of the underlying operating system threading and synchronization constructs for the best performance and implements constructs that aren't natively supported. Phission abstracts those lower level implementation details from the developer. Native operating system support includes Windows, Linux, Blackfin/VDK & Blackfin/uClinux.

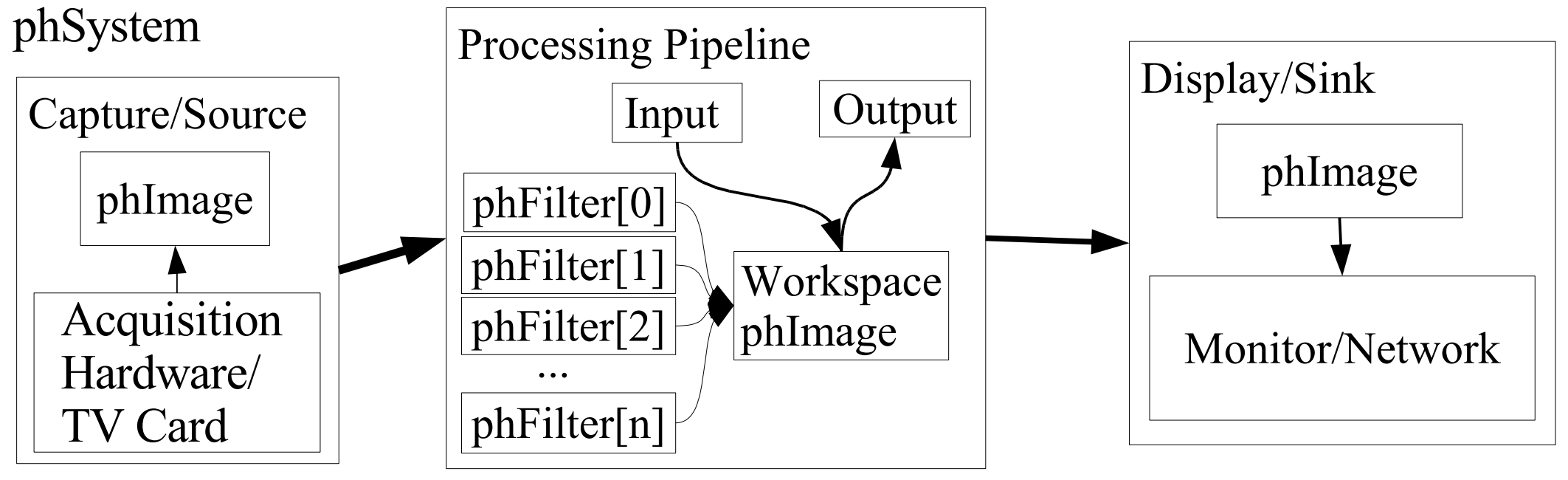
The high level concurrent system is made of three main components: *Capture/Source*, *Processing* and *Display/Sink* units. Image data enters the system at the *Capture/Source* and is copied to the *Processing* unit when the *Processing* unit is ready for the data. Data leaving the *Processing* unit, or the *Source*, can be viewed with a *Display/Sink* object.

Processing units contain *Filter* units that are given the image data to perform any basic or complex image processing and analysis. Multiple color spaces and conversion between all those color spaces is supported internally to Phission which the *Filter* units can utilize.

Possible Applications

- ◆ Object tracking and recognition for mobile robotic applications
- ◆ Acquire a colored ball to place in a colored trash can
- ◆ People following: train on the color of a persons pants and follow them
- ◆ Car tracking: Assist in robot navigation to cross the street
- ◆ Teleoperation: The NetDisplay / phNetSource combination allows transferring and receiving of images over the network for web cam type monitoring of what the mobile robot sees.

Example Application Design



1.) A Source class continually capture and buffer the most recent image. It is always ready for anyone to copy it in a thread safe manner.

2.) `phPipeline` copies the input image to the workspace and runs each filter process serially on the image data. A workspace image is used to reduce copying of image data.

3.) Take the image data and display it to the monitor. The display/sink can be anything such as a network or a movie file. JPEG compression is also supported for network transmission.

Example Application Code

```
// Declare all the Phission objects being used for object tracking
phSystem                system;                phPipeline                pipeline;
V4LCapture              capture;              X11Display                processed_display;
X11Display              live_display;          gaussianBlur_Filter       gauss;
meanBlur_Filter         mean;                 blob_Filter               blob;
phBlobData              blobdata;

// Add the threaded objects to the phSystem for easy system startup and shutdown
system.addCapture(capture);                    system.addPipeline(pipeline);
system.addDisplay(processed_display); system.addDisplay(live_display);
// Initialize the blob filter to look for red objects
blob.addColor(new_phColorRGB24(255,0,0),new_phColorRGB24(40,40,40));
// Add the Filters to the pipeline; They're executed in the same order they were added
pipeline.add(mean); pipeline.add(gauss); pipeline.add(blob);

// Connect the inputs/outputs for the Capture, Pipeline and Displays
live_display.setLiveSourceInput(capture.getLiveSourceOutput());
pipeline.setLiveSourceInput(capture.getLiveSourceOutput());
processed_display.setLiveSourceInput(pipeline.getLiveSourceOutput());

// Connect the blobdata to the output of the blob filter; The phBlobData object
// facilitates thread safe operations
blobdata.connect(blob.getLiveBlobOutput());

system.start(); // Start the Phission concurrent processing sub-system in
while (!done) {
    if (blobdata.update() == phLiveObjectUPDATED){
        /* Robot Control code using data from blob; move left/right; go forward/backward */
    }
};
system.stop(); // Cleanup/Wakeup all Phission threads and stop the processing sub-system
```

Why does Phission exist?

- ◆ To put emphasis on the development of filtering, processing, and analysis of the data by supplying ready made capture and display facilities
- ◆ To encapsulate framework and operating system details away from the researcher so the majority of time will be applied to writing the actual algorithms and analysis routines
- ◆ The system has comparatively little CPU overhead versus processing routines to maximize the performance on low end general purpose computers
- ◆ During initial research, no comparable open source toolkit with the desired support, design characteristics, portability and extensibility was available
- ◆ In addition to being used for the application of student thesis research, Phission provides a much more powerful vision processing system for the Pyrobot environment than the default system.

Platform Objects

- ◆ *phMutex*: Mutual Exclusion locks
- ◆ *phSemaphore*: Semaphores
- ◆ *phCondition*: Condition Variables
- ◆ *phRWLock*: Reader Writer Locks
- ◆ *phTime/phTimeStamp*: Timing routines
- ◆ *phSocket/phServerSocket*: General Socket and Server related functionality
- ◆ *phThread*: C++ Thread object

Has extended functionality for robust application control using special setup, cleanup, wakeup, and error methods. Also supplies synchronous thread spawning to detect errors during the initiation of a new thread such as failure to open capture hardware or videos.

phLiveObject

- ◆ The *phLiveObject* encapsulates the thread-safe mechanisms and synchronization for copying data
- ◆ A *phLiveObject* is capable of either swapping data or copying data during an update call
- ◆ A *phLiveObject* derived class must overload the 'swap' and 'copy' methods

phImage

- ◆ This is an abstract class that provides methods for:
 - ◆ Resizing using Bilinear Interpolation or Nearest Neighbor algorithms
 - ◆ Conversion between many formats including: RGB24, RGBA32, BGR24, ABGR32, HSV24, YUV9, GRAY8
 - ◆ Writing images to PPM or JPEG image files
 - ◆ Compress/zip images using ZLIB for lossless transmission
- ◆ *phImage* is derived from *phLiveObject* to allow for “Live Source” connections made internally to the Display classes, the Pipeline and Capture classes

Capture

- ◆ Supported capture facilities include: OmniVision 7620 for the Blackfin HandyBoard, Video Files(mpeg/avi/etc.), Video4Linux, Video4Windows and the *NetDisplay/phNetSource* combination
- ◆ The *phCaptureInterface* and *phImageCapture* classes supply the common overloadable interface for the capture facilities.
- ◆ While active, the capture class continually captures images and makes the image available for other threads to copy
- ◆ Allows changing of brightness, contrast, color, hue, and whiteness
- ◆ If there are multiple video devices, any number of capture class objects can exist to provide many video stream sources
- ◆ Supports multiple hardware/kernel buffers for initiating multiple captures in parallel
- ◆ Permits stopping and restarting of capture thread within the normal flow of the program

Display

- ◆ The Display classes display data as fast as possible and won't keep the application from capturing or processing more data
- ◆ A Display class is derived from the *phDisplayInterface* class to provide the same interface between all display classes
- ◆ The Researcher doesn't have to know low level details of how to get images displayed on a system

Filter

- ◆ Provides a common interface for executing a filter
- ◆ Provides image format verification for image filters that don't support all color spaces. For example, The C optimized filters are optimized for images with a RGB24 byte layout
- ◆ Allows “monitoring” of filtered image result so each stage of a pipeline can be monitored without any need for hacking the *phPipeline*
- ◆ Each filter in a pipeline can be enabled or disabled to allow for more flexibility in the pipeline system

Pipeline

- ◆ Provides an encapsulated interface for processing and analyzing data in parallel with the normal application
- ◆ Accepts *phFilter* derived classes to allow multiple stages in a pipeline. For example, Gaussian blurring before color matching or edge detecting
- ◆ *phPipeline* output can be linked to either a *Display* class or another *phPipeline*

phSimpleVision

- ◆ Provides a simple C++ object that encapsulates a basic system of one capture input, one pipeline, one network display and one native display
- ◆ Simple methods allow the execution of specific filters & train and track methods for the RGB / HSV color spaces

Available Displays

- ◆ *SDLDisplay* Uses the SDL (Simple Direct-media Layer) API for displaying images
- ◆ *NetDisplay* A server which clients can connect to and receive image data over the network
- ◆ *X11Display* Uses Xlib API for displaying and allows multiple displays per process
- ◆ *FLDisplay* Uses the FLTK (Fast Light Tool Kit) framework for displaying images
- ◆ *GDIDisplay* Uses the Win32 GDI API to display natively in windows

Available Capture APIs

- ◆ *V4LCapture* Video4Linux v1 API supplied by the Linux kernel for many common tv cards and other image acquisition hardware
- ◆ *VFWSource* VideoForWindows API supplied by the Win32 API for basic image acquisition on windows. (This doesn't support DirectShow capturing)
- ◆ *phNetSource* Network capture object to be used with its *NetDisplay* companion class. This makes sending and receiving images much simpler and supports data throttling.
- ◆ *phAvcodecSource* This uses the FFMpeg Avcodec/Avformat libraries to support many common video file formats which include MPEG and AVI formatted files.

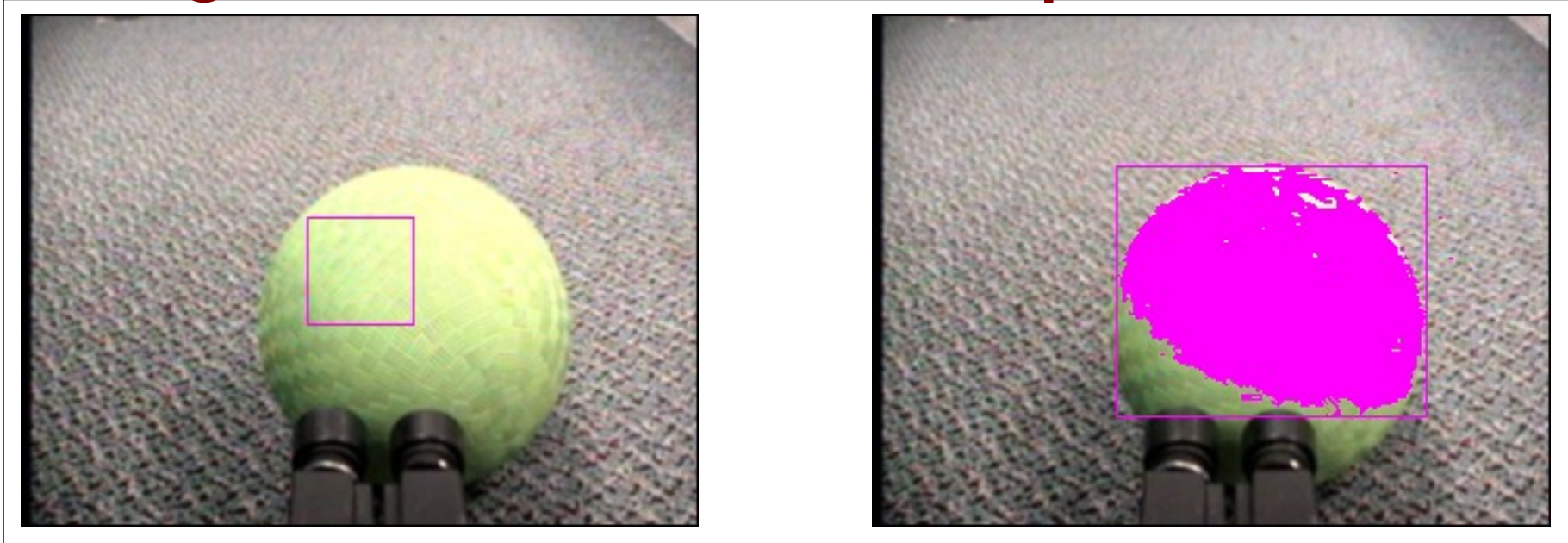
Available Image Filters

<i>add</i>	add frames together
<i>ave</i>	average N frames
<i>blob</i>	match, blob colors and output blob data
<i>blobify</i>	blob color channel which is output from <i>match</i>
<i>brightness</i>	software brightness alteration
<i>canny</i>	Canny edge detection
<i>convert</i>	convert between image formats
<i>ddimage</i>	Double Difference frames
<i>gaussian3x3</i>	RGB24 Gaussian blur, kernel size 3x3
<i>gaussianBlur</i>	phColor supported Gaussian blur, kernel size 5x5
<i>grayScale</i>	convert to gray scale format
<i>histogram</i>	histogram an area of the image
<i>inverse</i>	mirror color values: 255 to 0; 0 to 255; 254 to 1
<i>mask</i>	Mask off a frame given an image mask
<i>match</i>	match a color and output to color channel

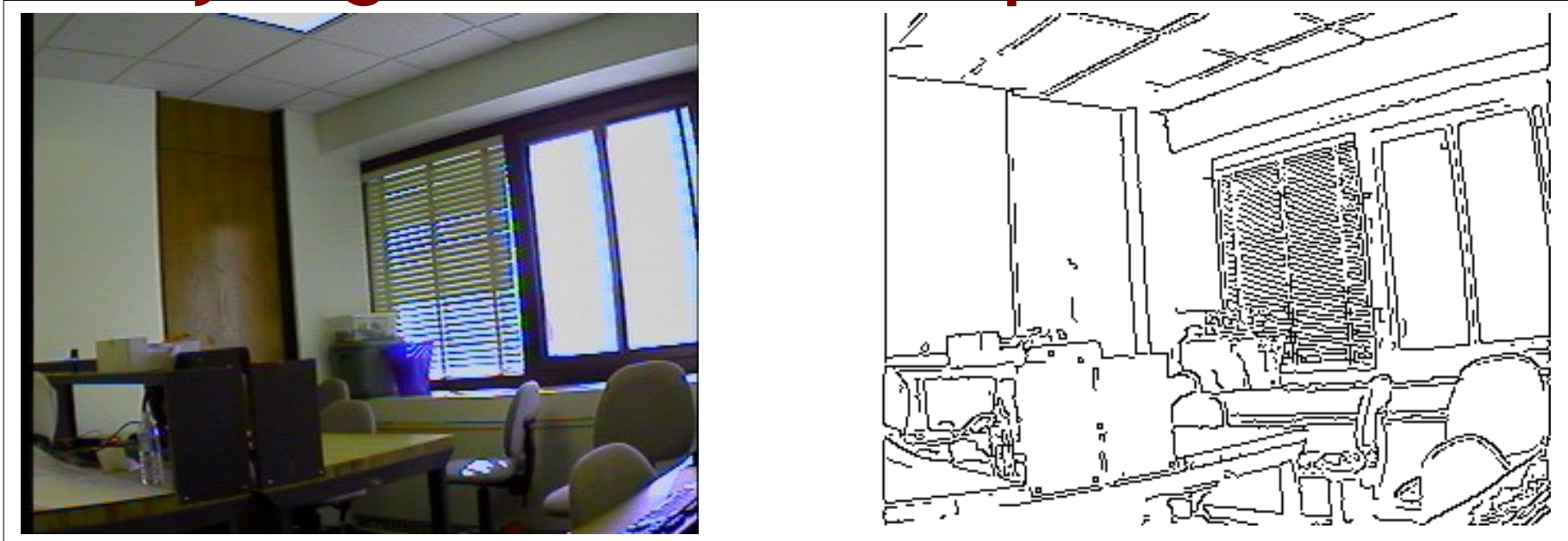
Available Image Filters

<i>meanBlur</i>	phColor supported mean blur, kernel size 3x3
<i>meanNxN</i>	RGB mean blur, kernel size 3x3, 5x5, 7x7, ...
<i>medianBlur</i>	phColor supported median blur, kernel $\geq 3 \times 3$
<i>medianNxN</i>	C optimized RGB24 median blur, kernel 3x3, 5x5, 7x7, 11x11
<i>motion</i>	Detects motion between frames
<i>original</i>	Replace workspace image with original pipeline input
<i>resize</i>	Resize the workspace image
<i>sobel3x3</i>	RGB24 Sobel edge detection, kernel 3x3
<i>sobel</i>	phColor supported Sobel edge detection 3x3
<i>subtract</i>	Subtracts N frames from each other
<i>superRGB</i>	Pulls out color channel information for <i>blobify</i>
<i>threshold</i>	Thresholds a color channel based on a limit. < <i>limit</i> goes to 0 > <i>limit</i> goes to 255

Histogram & Blob Filter Example



Canny Edge Detection Example



The Future of Phission

- ◆ phPipeline “Resource Table” for intra-pipeline filter data sharing
 - ◆ This will also allow multiple inputs to the phPipeline
- ◆ Data buffering: Never miss a piece of information
- ◆ Synchronous connections
- ◆ Audio Capture and Processing support
- ◆ Fixed-Point optimizations to all floating point operations
- ◆ Improved Nearest-Neighbor blob segmentation
- ◆ 16-bit, 32-bit, float data types for image channels
- ◆ Spherical Coordinate Transform(SCT)/LAB image format support
- ◆ Firewire(ieee1394) capturing support
- ◆ Complete tutorials on extending Phission's capture and filter

**Learn more about Phission
go to <http://www.phission.org>**

Research conducted under the auspices of Dr. Holly A. Yanco, the Robotics Lab, and the Computer Science Department of UMass Lowell.

Contact holly@cs.uml.edu for additional general information

Or contact pthoren@cs.uml.edu for information regarding Phission

Phission

Modular

Concurrent Image

Processing and

Analysis Toolkit

by Philip D.S. Thoren