FIRST Robotics Competition (FRC)

Application Programming Interface (API) Specification

For Machine Vision

Rev 0.7

December 20, 2008

Beth Finn

BAE SYSTEMS

Table of Contents

FRC Vision API Specification

Introduction

The FIRST Robotics Competition (FRC) Vision API provides FRC teams the capacity to create programs to run on the 2009 FRC Robot Controller for automated image processing.

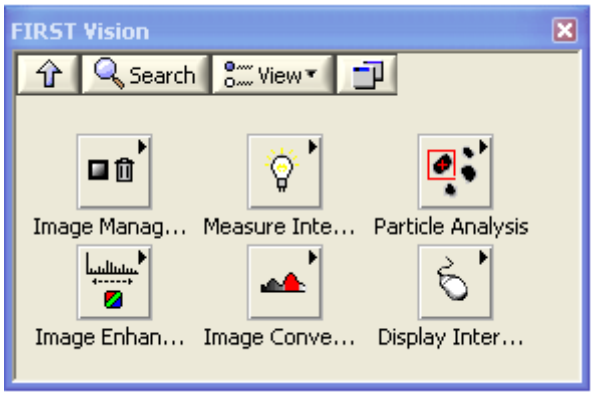
The FRC Vision API is comprised of two separate APIs, a LabVIEW palette of vision VIs and a C++ library providing the same capabilities. LabVIEW users may link in the simplified vision palette using the menu items described below. For C and C++ programmers, the WPILib.a library provides C wrappers for the more complex procedure calls provided with the National Instruments (NI) C library. These calls are coded in the open source file VisionAPI.cpp and are described in this document.

Programmers have the option with both LabVIEW and C++ to use the simplified vision palette or C wrappers, or to call the vision libraries directly. C++ programmers can inspect the nivision.h header file for specifications of all possible calls to the Vision library. LabVIEW users have access to the full set of palettes under the Vision and Motion menu. Appendix A to this document describes data structures used by the simplified library. The *NIVisionCVI.chm* help file and the *NI Vision for LabWindows/CVI User Manual* provide more information about data structures and usage.

This document will describe both the LabVIEW palette and the C API for each function. Data structures are described in a separate document provided as an Appendix A. More information for the LabVIEW VIs and their components is available using Context help, obtained under the "Help" menu or by using <ctrl-H>. More detail on the C API is available in the doxygen help file for FRC Vision and in the function headers.

FIRST Vision Palette Description

The LabVIEW vision palette will consist of 8 Subpalettes with the following high level descriptions below.



- Image Management
- Measure Intensity
- Particle Analysis
- Image Enhancement
- Image Conversion
- Display Interaction

C/C++ API Description

The C API includes similar functionality to the LabVIEW Vision Palette, with additional routines added for higher levels calls for color tracking. The FRC C API is included as part of the WPI Robotics Library, with the file names *VisionAPI.cpp and TrackAPI.cpp*. Camera control is provided in *AxisCamera.cpp*. These interfaces are all in C, however C++ wrappers for the camera interface is planned for the future.

The sections below describe the functions in each API. A screen shot of each LabVIEW function is provided and the image types supported by each function are specified. The C++ library function call is described, including the parameters, return value, and the function call that is made to the underlying LabWindows™/CVI™ vision library. Advanced programmers who wish to call directly into the vision library to override the simplified call defaults can use this information to formulate their calls. The *nivision.h* file specifies the prototypes for all calls into this library, which is a proprietary National Instruments product. The FRC API, support utilities, and vision demo programs are open source code under the BSD license.

Naming Conventions

C routines prefixed with “imaq” belong to NI’s LabVIEW/CVI vision library. Routines prefixed with “frc” are simplified interfaces to the vision library provided by BAE Systems for FIRST Robotics Competition use.

A Note on Error Handling in C

Most of the vision functions return an integer value, 1 if the call was successful, and 0 if unsuccessful. To obtain information on a failure, the GetLastError call can be used to get an error number. If debug is turned on, the debug error text is printed to the console or debug file. The error text may also be obtained in the C++ API by calling **frcGetErrorText(errorCode).** For LabVIEW, the LabWindows/CVI Function Reference Help includes the Error text under **Function Reference**.

Image Management Functions

The image management VIs are part of the Vision for LabVIEW Library.

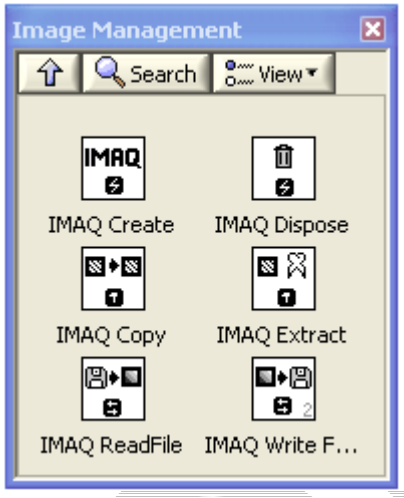


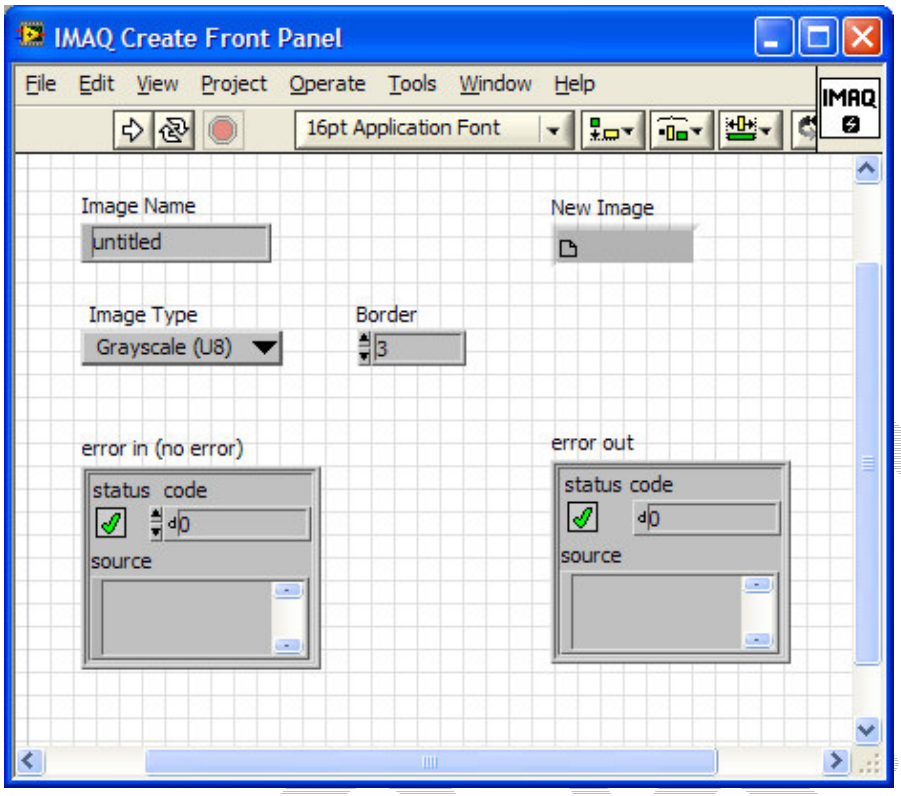
Image Creation

Functions used to create an image in memory. The created image will be 0 x 0 pixels in size. To change the image size, use imaqSetImageSize().

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL, IMAQ_IMAGE_RGB_U64

LabVIEW API: IMAQ Create



C API: frcCreateImage

Image* frcCreateImage(ImageType type)

Return value:

Image* - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

type - specifies the image type. Choose from the following ImageType values:

Grayscale (U8) (0)	8 bits per pixel (unsigned, standard monochrome)
Grayscale (16) (1)	16 bits per pixel (signed)
Grayscale (SGL) (2)	32 bits per pixel (floating point)
Complex (CSG) (3)	2 × 32 bits per pixel (floating point)
RGB (U32) (4)	32 bits per pixel (red, green, blue, alpha)

HSL (U32) (5) 32 bits per pixel (hue, saturation, luminance, alpha)

RGB (U64) (6) 64 bits per pixel (red, green, blue, alpha)

Library Function Call:
Image* imaqCreateImage(ImageType type, int borderSize)

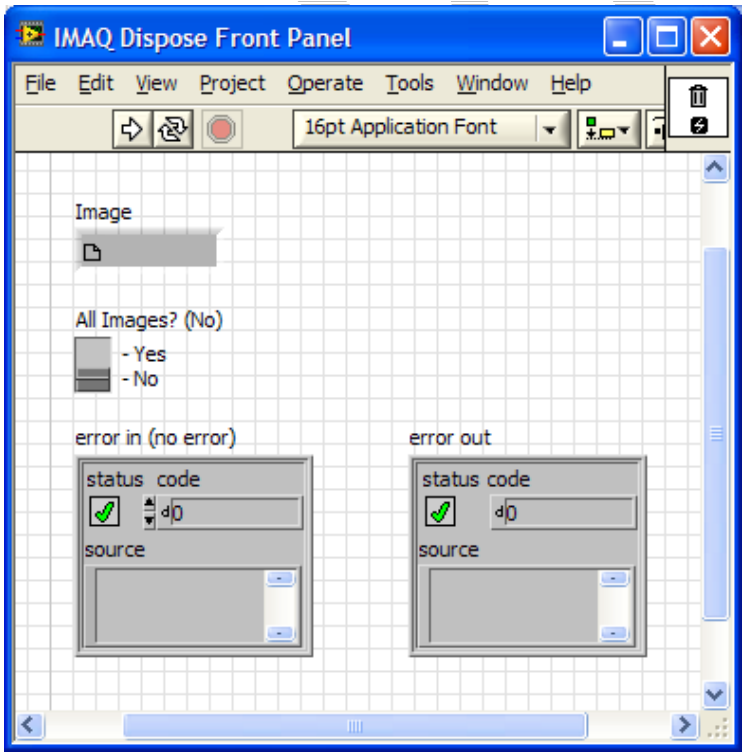
borderSize determines the width, in pixels, of the border to create around an image. FrcCreateImage will create an image with a border of 3 pixels.

Resource Disposal

Cleans up resources associated with images, regions of interest (ROIs), arrays, and reports that you no longer need. After you dispose of something, the space it occupied in memory is freed for future use, and the freed image or other data structure is no longer available for use. Execute this function only when the object is no longer needed in your application.

LabVIEW API: IMAQ Dispose

This VI is required for each image created in an application to free the memory allocated by the IMAQ Create function. You can use IMAQ Dispose for each call to IMAQ Create or just once for all images created with IMAQ Create.



C API: frcDispose

This function must be called to free memory for each image created when it is no longer needed. It may also be used to free memory for any other object. All objects should be disposed of when no longer used to prevent memory leaks.

int frcDispose(void* object)

int frcDispose(const char* functionName, void* object, ... , NULL)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

functionName – for the multi-item call only, the name of the function calling frcDispose().
object - specifies the object whose memory is to be freed. This may be any data structure with memory allocated to it. The second version takes a variable number of objects. The last item in the list of objects must be NULL.

Calls:
int imaqDispose(void* object)

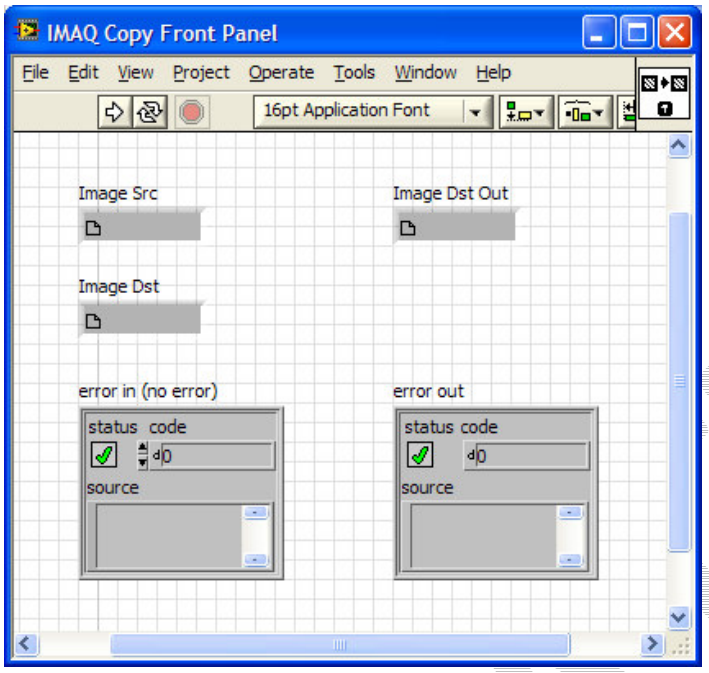
Image Copy Functions

Creates a copy of an image. Copies the specifications and pixels of one image into another image of the same type. You can use this function to keep an original copy of an image (for example, before processing an image). The full definition of the source image as well as the pixel data are copied to the destination image. The border size of the destination image also is modified to be equal to that of the source image. If the source image contains additional information, such as calibration information, overlay information, or information for pattern matching, this information is also copied to the destination image.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL, IMAQ_IMAGE_RGB_U64

LabVIEW API: IMAQ Copy



C API: frcCopyImage

Copies the entire source image to the destination image, including the border size and vision information. To copy an area of one image to an area of another image, use imaqCopyRect() instead. The image must be disposed of (frcDispose()) when no longer needed.

int frcCopyImage(Image* dest, const Image* source)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest – Image that holds the result. The image must have been previously created (frcCreateImage).

source – The source image to copy.

Calls:

int imaqDuplicate(Image* dest, const Image* source)

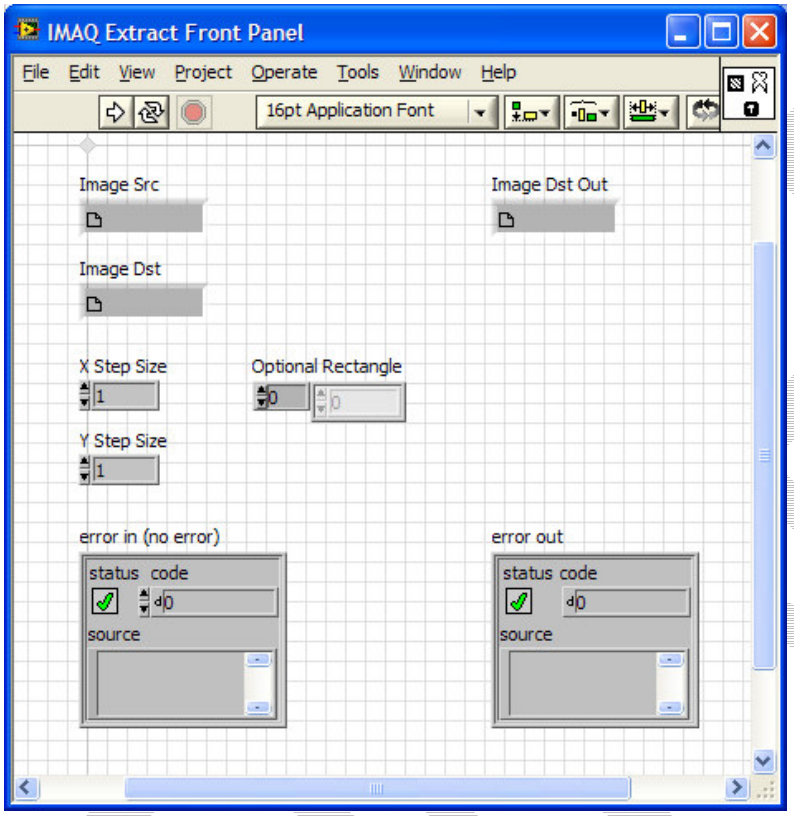
Extract Functions

Used to crop or scale down an image by adjusting the horizontal and vertical resolution. The source image and destination image must be the same image type.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

LabVIEW API: IMAQ Extract



C API: frcCrop

This function crops the image according to the rectangle passed in without changing the scale.

int frcCrop(Image* dest, const Image* source, Rect rect)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest – Image that holds the result. The image must have been previously created (frcCreateImage).
source – The source image to scale.
rect - defines a four-element array that contains the left, top, right, and bottom coordinates of the region to process. To apply the operation to the entire image, set **Rect** to IMAQ_NO_RECT.

Calls:

int imaqScale(Image* dest, const Image* source, int xScale, int yScale, ScalingMode scaleMode, Rect rect)

C API: frcScale

This function scales the entire image larger or smaller.

int frcScale(Image* dest, const Image* source, int xScale, int yScale, ScalingMode scaleMode) ;

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest – Image that holds the result. The image must have been previously created (frcCreateImage).
source – The source image to scale.
XScale - the horizontal reduction ratio. This is the vertical sampling step, which defines the columns to be extracted. For example, with an **X Step Size** equal to 3, one out of every three columns is extracted from the **source** into the **dest**. Each column is extracted if 1 is used (no change to scale).
yScale - the vertical reduction ratio. This is the horizontal sampling step, which defines the lines to be extracted. Each row is extracted if 1 is used (no change to scale).
scaleMode defines the type of scaling.

IMAQ_SCALE_LARGER = duplicate pixels to make the image larger
IMAQ_SCALE_SMALLER = subsample pixels to make the image smaller
If xScale or yScale are set to 1, either value may be entered here, and there is no change to scale for that dimension.

Calls:

int imaqScale(Image* dest, const Image* source, int xScale, int yScale, ScalingMode scaleMode, Rect rect)

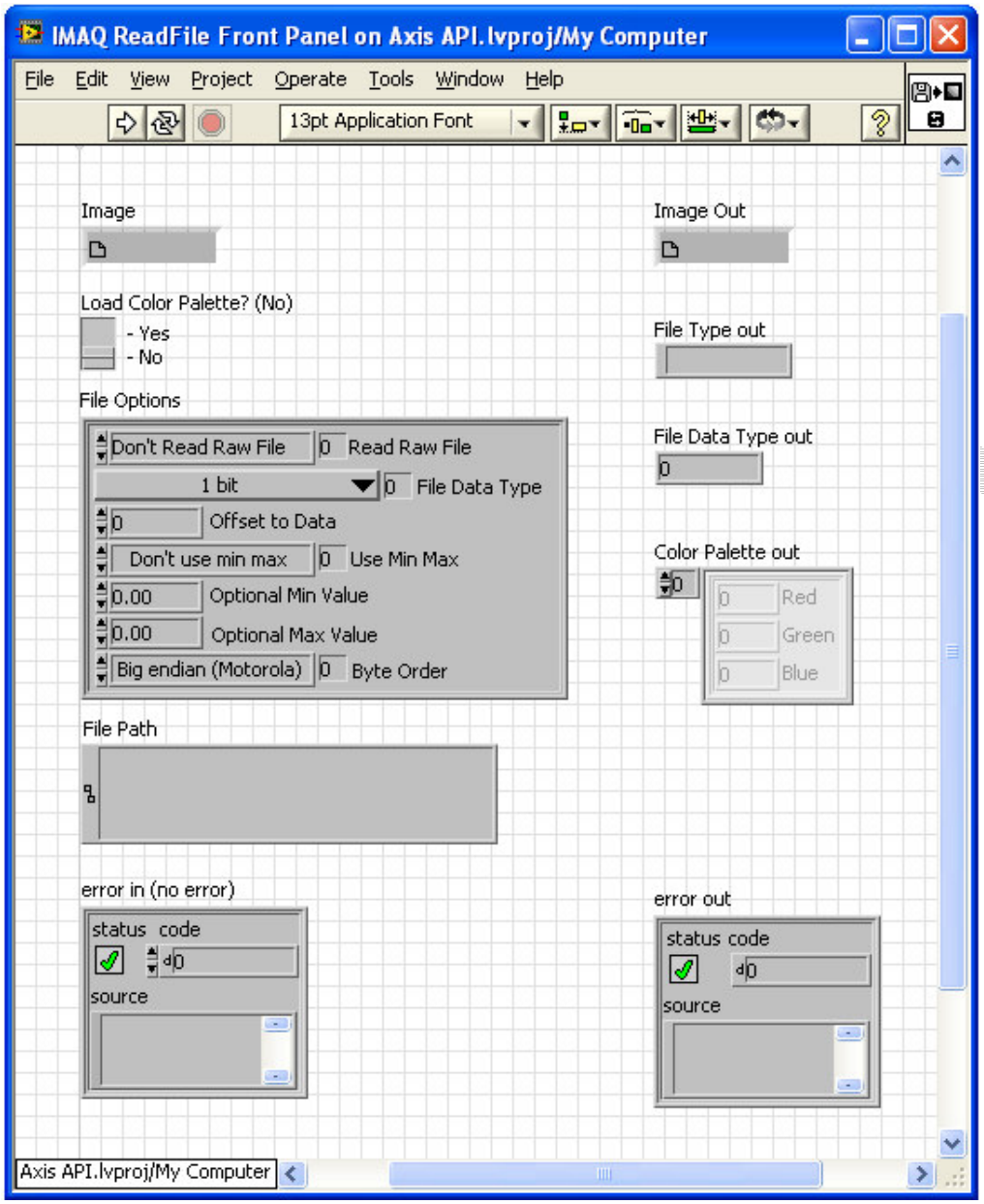
Vision File Functions

Reads and writes an image, along with extra vision information associated with the image, to/from a PNG file. This extra vision information includes overlay information, pattern matching template information, and calibration information.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL, IMAQ_IMAGE_RGB_U64

LabVIEW API: IMAQ Read Image and Vison Info



C API: frcReadVisionFile

This function reads .PNG files with image and vision data.

int frcReadVisionFile (Image* image, const char* filename)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

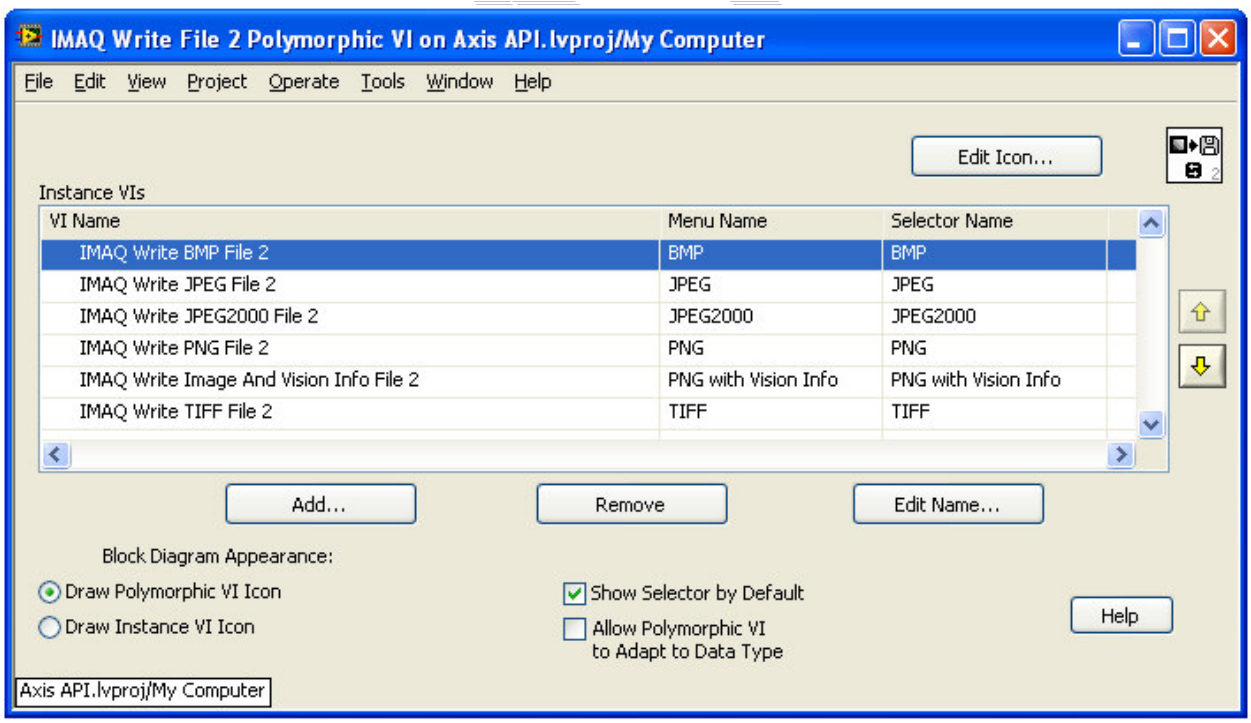
image - The image in which the function stores data it reads from the file.
filename - The name of the file to read. This parameter is required and cannot be NULL.

Discussion:

This call retrieves .png files only. For files in other formats (JPEG, JPEG2000, TIFF, AIPD, or BMP) use frcReadImage().

Calls:
int imaqReadVisionFile(Image* image, const char* fileName, RGBValue* colorTable, int* numColors)

LabVIEW API: IMAQ Write Image and Vision Info



C API: frcWriteVisionFile

This function writes .PNG files with image and vision data.

int frcWriteVisionFile(const Image* image, const char* fileName)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

image - The image to write to a file.
filename - The name of the file to write. This parameter is required and cannot be NULL.

Calls:
int imaqWriteVisionFile(const Image* image, const char* fileName, const RGBValue* colorTable)

Image File Functions

Read and write images in several standard formats.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL, IMAQ_IMAGE_RGB_U64

C API: frcReadImage

Creates image object from the information in a file. The file can be in one of the following formats: PNG, JPEG, JPEG2000, TIFF, AIPD, or BMP.

int frcReadImage (Image* image, const char* filename)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

image - The image in which the function stores data it reads from the file.
filename - The name of the file to read. This parameter is required and cannot be NULL.

C API: frcWriteImage

Write an image to a file. The file can be in one of the following formats: PNG, JPEG, JPEG2000, TIFF, AIPD, or BMP.

int frcWriteImage (Image* image, const char* filename)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

image - The image to write to a file.

filename - The name of the file to write. This parameter is required and cannot be NULL.

Parameter Discussion:

The file type is determined by the extension of the filename, as follows:

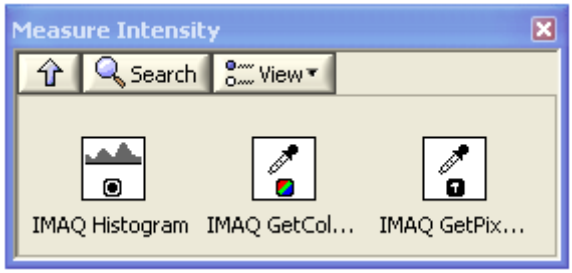
Extension	File Type
.aipd or .apd	AIPD
.bmp	BMP
.jpg or .jpeg	JPEG
.jp2	JPEG2000
.png	PNG
.tif or .tiff	TIFF

The following are the supported image types for each file type:

File Types	Image Types
AIPD	all image types
BMP, JPEG	8-bit, RGB
PNG, TIFF, JPEG2000	8-bit, 16-bit, RGB, RGBU64

Intensity Measurement Functions

These VIs are part of the Vision for LabVIEW Library.



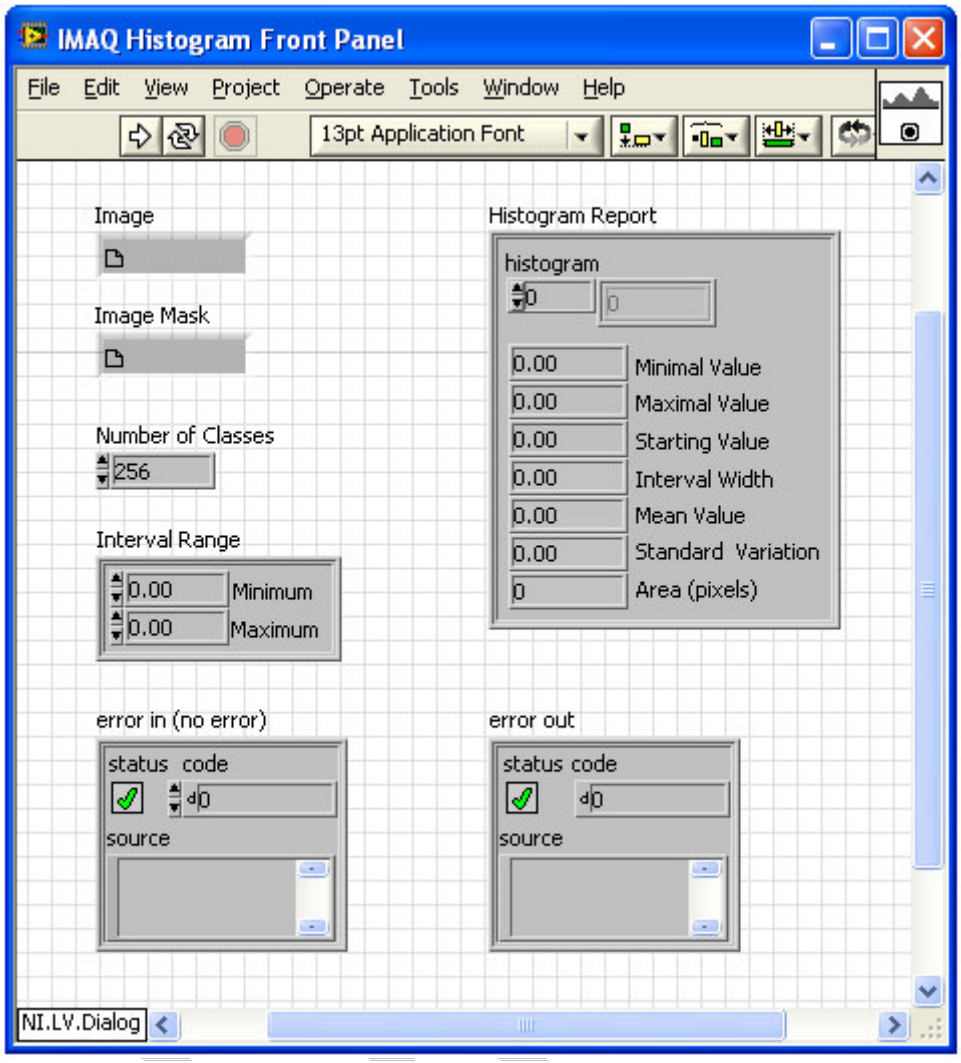
Histogram Function

Measures the pixel intensities in a rectangle of an image. Outputs intensity based statistics about an image such as Max, Min, Mean and Standard Deviation of pixel value.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

LabVIEW API: IMAQ Histogram



C API: frcHistogram

HistogramReport* frcHistogram(const Image* image, int numClasses, float min, float max)

HistogramReport* frcHistogram(const Image* image, int numClasses, float min, float max, Rect rect)

Return value:

HistogramReport - On success, this function returns a report describing the pixel value classification. On failure, this function returns NULL. To get extended error information, call GetLastError(). When you are finished with the report, dispose of it by calling frcDispose().

Parameters:

image - The image whose histogram the function calculates.
numClasses :The number of classes into which the function separates the pixels. Determines the number of elements in the histogram array returned
min: The minimum pixel value to consider for the histogram. The function does not count pixels with values less than **min**
max: The maximum pixel value to consider for the histogram. The function does not count pixels with values greater than **max**
rect: Region of interest in the image. If not used, the entire image is used.

Parameter Discussion

min—Setting both **min** and **max** to 0 causes the function to set **min** to 0 for 8-bit images and to the actual minimum value of the image for all other image types.
max—Setting both **min** and **max** to 0 causes the function to set **max** to 255 for 8-bit images and to the actual maximum value of the image for all other image types.

Calls:
HistogramReport* imaqHistogram(const Image* image, int numClasses, float min, float max, const Image* mask)

C API: frcColorHistogram

Calculates the histogram, or pixel distribution, of a color image.

ColorHistogramReport* frcHistogram(const Image* image, int numClasses, ColorMode mode)
ColorHistogramReport* frcHistogram(const Image* image, int numClasses, ColorMode mode, Image* mask)

Return value:

ColorHistogramReport* - On success, this function returns a report describing the classification of each plane in a HistogramReport. On failure, this function returns NULL. To get extended error information, call GetLastError(). When you are finished with the report, dispose of it by calling frcDispose().

Parameters:

image - The image whose histogram the function calculates.
numClasses :The number of classes into which the function separates the pixels. Determines the number of elements in the histogram array returned
mode - The color space in which to perform the histogram. Possible values include IMAQ_RGB and IMAQ_HSL.
mask - An optional mask image. This image must be an IMAQ_IMAGE_U8 image. The function calculates the histogram using only those pixels in the image whose corresponding pixels in the mask are non-zero. Set this parameter to NULL to calculate the histogram of the entire image, or use the simplified call.

Calls:

ColorHistogramReport* imaqColorHistogram2(Image* image, int numClasses, ColorMode mode, const CIEXYZValue* whiteReference, Image* mask);

Pixel Value Data Access

GetColorPixelValue is used to get the Red, Green and Blue values of a pixel. GetPixelValue is used to get the intensity value of a pixel in a grayscale image. The LabVIEW implements two different VIs for this while the C API returns a data structure containing either grayscale, RGB, HSL, Complex or RGBU64Value depending on the type of image.

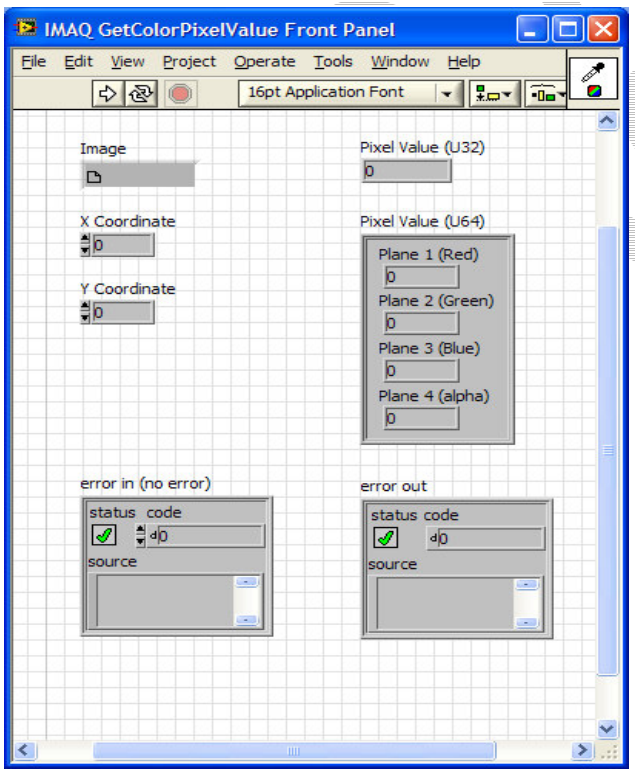
Image Types Supported – GetColorPixelValue

IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL, IMAQ_IMAGE_RGB_U64

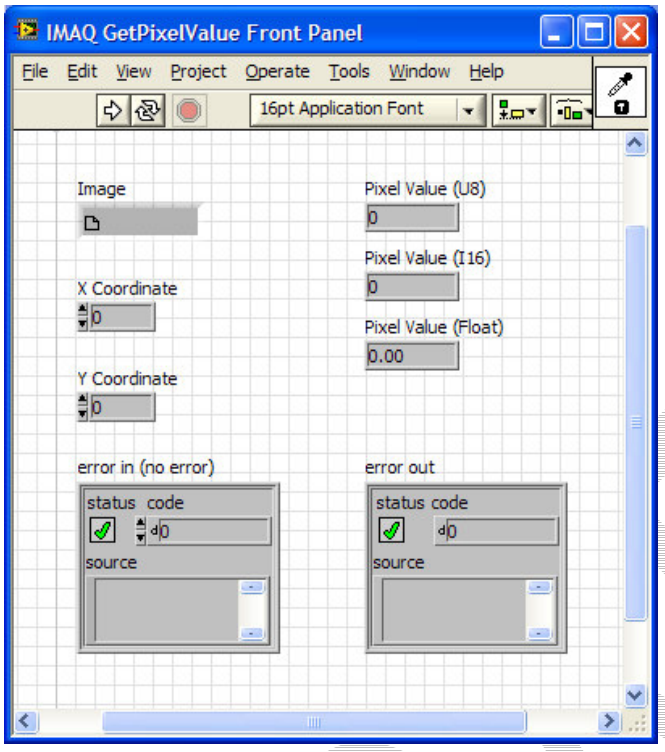
Image Types Supported – GetPixelValue

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

LabVIEW API: IMAQ GetColorPixelValue



LabVIEW API: IMAQ GetPixelValue



C API: frcGetPixelValue

The C API call works for both grayscale and color images. The data returned in the PixelValue is determined by the image type.

int frcGetPixelValue (const Image* image, Point pixel, PixelValue* value)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

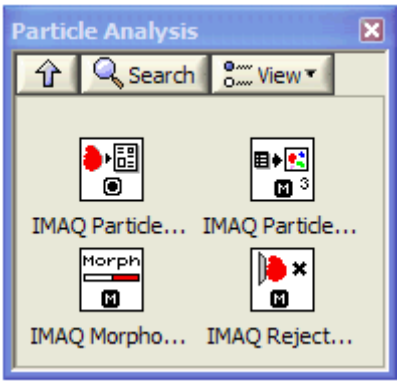
image - The image whose pixel value the function queries.
pixel - The coordinates of the pixel that the function queries.
value - On return, the value of the specified image pixel. The float format can accept values from all supported image types (8-bit, 16-bit, or 32-bit floating point. This parameter is required and cannot be NULL.

See the *NIVisionCVI.chm* help file for descriptions of parameter data structures.

Calls:
int imaqGetPixel(const Image* image, Point pixel, PixelValue* value)

Particle Analysis Functions

These VIs are part of the Vision for LabVIEW Library.



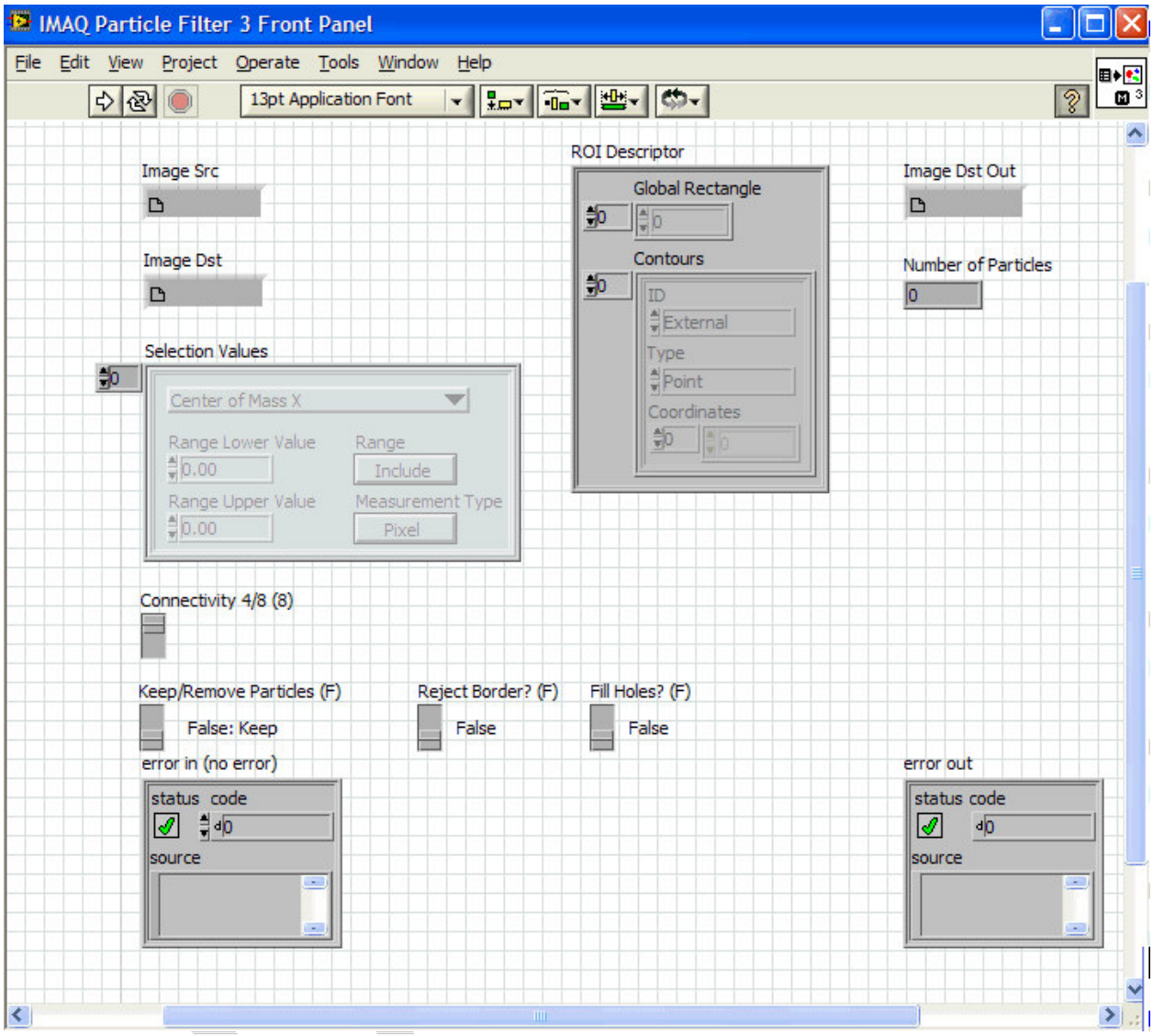
Filtering Particles

Filters particles out of an image based on their measurements.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

LabVIEW API: IMAQ Particle Filter 3



C API: frcParticleFilter

int frcParticleFilter(Image* dest, Image* source, const ParticleFilterCriteria2* criteria, int criteriaCount, const ParticleFilterOptions* options, int* numParticles)

int frcParticleFilter(Image* dest, Image* source, const ParticleFilterCriteria2* criteria, int criteriaCount, const ParticleFilterOptions* options, Rect rect, int* numParticles)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest - the destination image. If **dest** is used, it must be the same size as the **Source** image. It **will** contain only the filtered particles.
source - The image containing the particles to filter.
criteria - An array of criteria to apply to the particles in the source image. This array cannot be NULL. See the *NIVisionCVI.chm* help file for definitions of criteria.
criteriaCount - The number of elements in the **criteria** array.
options – Binary filter options, including rejectMatches, rejectBorder, and connectivity8.
rect – Area to filter. If called without this parameter, the entire image is used.
numParticles - On return, the number of particles left in the image.

See the *NIVisionCVI.chm* help file for descriptions of parameter data structures.

Calls:
int imaqParticleFilter3(Image* dest, Image* source, const ParticleFilterCriteria2* criteria, int criteriaCount, const ParticleFilterOptions* options, const ROI* roi, int* numParticles)

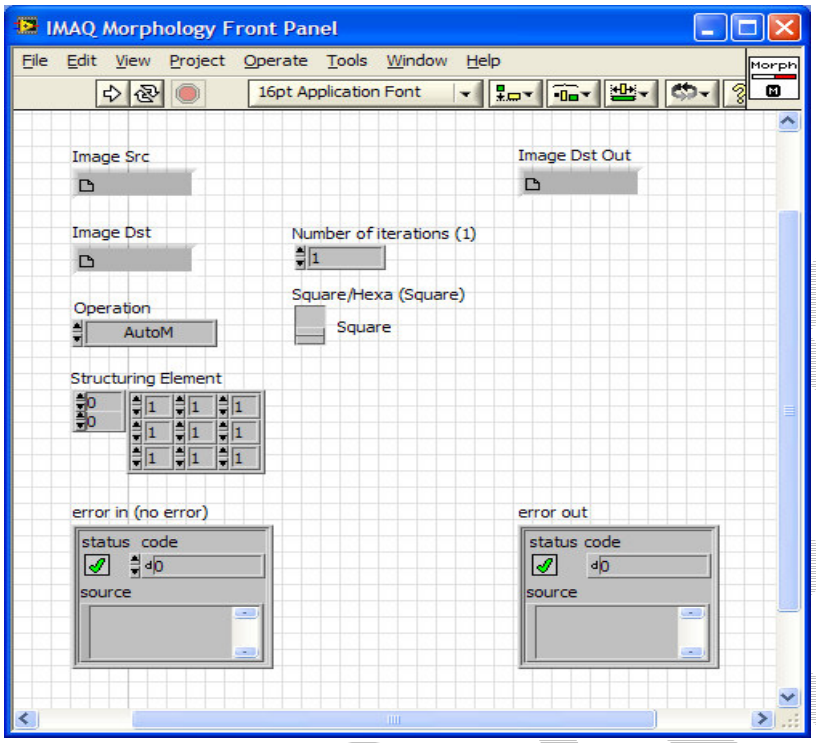
Morphological Transformation

Performs morphological transformations on binary images. These transformations can adjust the image by filling holes, removing small particles, smoothing boundaries, eliminating isolated pixels and other actions. For a description of types of transforms possible, see the *LabWindows/CVI Function Reference Help* and the *NI Vision Concepts Manual.*

Image Types Supported

IMAQ_IMAGE_U8
The connected source image for a morphological transformation must have been created with a border capable of supporting the size of the structuring element. A 3 × 3 structuring element requires a minimal border of 1, a 5 × 5 structuring element requires a minimal border of 2, and so on. The border size of the destination image is not important.

LabVIEW API: IMAQ Morphology



C API: frcMorphology

int frcMorphology(Image* dest, Image* source , MorphologyMethod method)
int frcMorphology(Image* dest, Image* source , MorphologyMethod method, const StructuringElement* structuringElement)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest - the destination image

source - The image on which the function performs the morphological operations. The calculation modifies the border of the source image. The border must be at least half as large as the larger dimension of the structuring element.

method - The morphological transform to apply. See *MorphologyMethod* in the *NIVisionCVI.chm* help file .

structuringElement - The structuring element used in the operation. Set this parameter to NULL if you do not want a custom structuring element (or use the simplified call).

See the *NIVisionCVI.chm* help file for descriptions of parameter data structures.

Calls:

int imaqMorphology(Image* dest, Image* source, MorphologyMethod method, const StructuringElement* structuringElement)

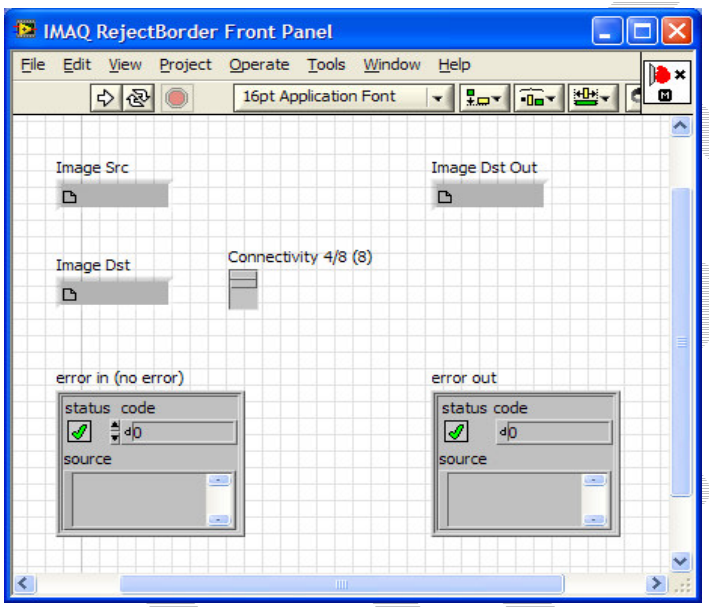
Reject Border Particles

Filters out any particles that are touching the border of an image. The source image must be an 8-bit binary image.

Image Types Supported

IMAQ_IMAGE_U8

LabVIEW API: IMAQ RejectBorder



C API: frcRejectBorder

This function eliminates particles that touch the border of the image. All border pixel values are set to 0.

int frcRejectBorder(Image* dest, Image* source)

int frcRejectBorder(Image* dest, Image* source, int connectivity8)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest - the destination image
source - The source image. If the image has a border, the function sets all border pixel values to 0.
connectivity8 - specifies the type of connectivity used by the algorithm for particle detection. The connectivity mode directly determines whether an adjacent pixel belongs to the same particle or a different particle. Set this parameter to TRUE to use connectivity-8 to determine whether particles are touching (This is the default setting for the simplified call). Set this parameter to FALSE to use connectivity-4 to determine whether particles are touching.

Calls:
int imaqRejectBorder(Image* dest, Image* source, int connectivity8)

Particle Analysis

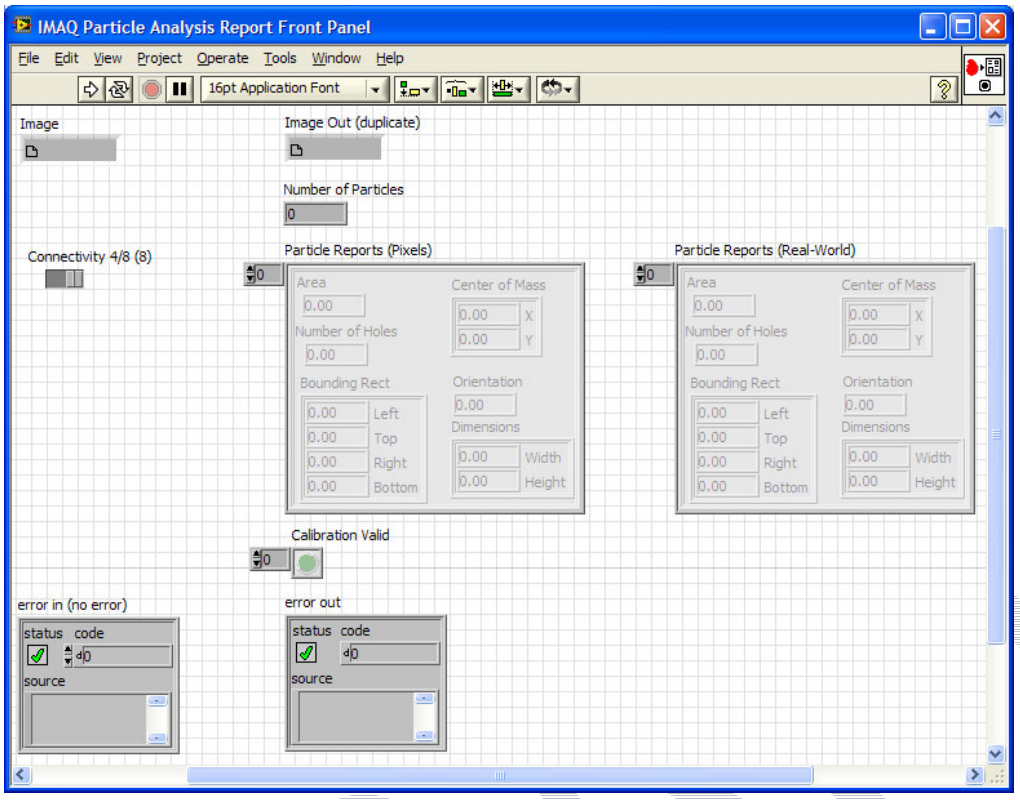
Outputs a report with various measurements a single particle found in an image. Returns a report containing the most commonly used particle measurements.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

LabVIEW API: IMAQ ParticleAnalysisReport

LabVIEW outputs a report with various measurements for every particle found in an image. Returns the number of particles detected in a binary image and an array of reports containing the most commonly used particle measurements.



C API: frcCountParticles

This function counts the number of particles in a binary (thresholded) image.

int frcCountParticles(Image* image, int* numParticles)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

image - The image containing the particles to count.
particleNumber – On return, the number of particles in the image.

Calls:
int imaqCountParticles(Image* image, int connectivity8, int* numParticles)

C API: frcParticleAnalysis

This conducts analysis on a single particle. If analysis on more than one particle is desired, call frcCountParticles and call frcParticleAnalysis on each particle desired.

Int frcParticleAnalysis (Image* image, int particleNumber, ParticleAnalysisReport* par)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

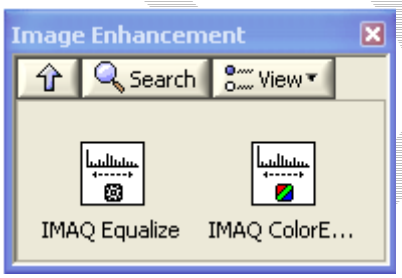
Parameters:

image - The image containing the particle to get information about.
particleNumber - The number of the particle to get information about.
ParticleAnalysisReport* - On success, this function returns a structure containing information about each particle. On failure, this function returns NULL. To get extended error information, call GetLastError(). When you are finished with this structure, dispose of it by calling frcDispose().

See Appendix A for description of the ParticleAnalysisReport data structure.

Image Enhancement Functions

These VIs are part of the Vision for LabVIEW Library.



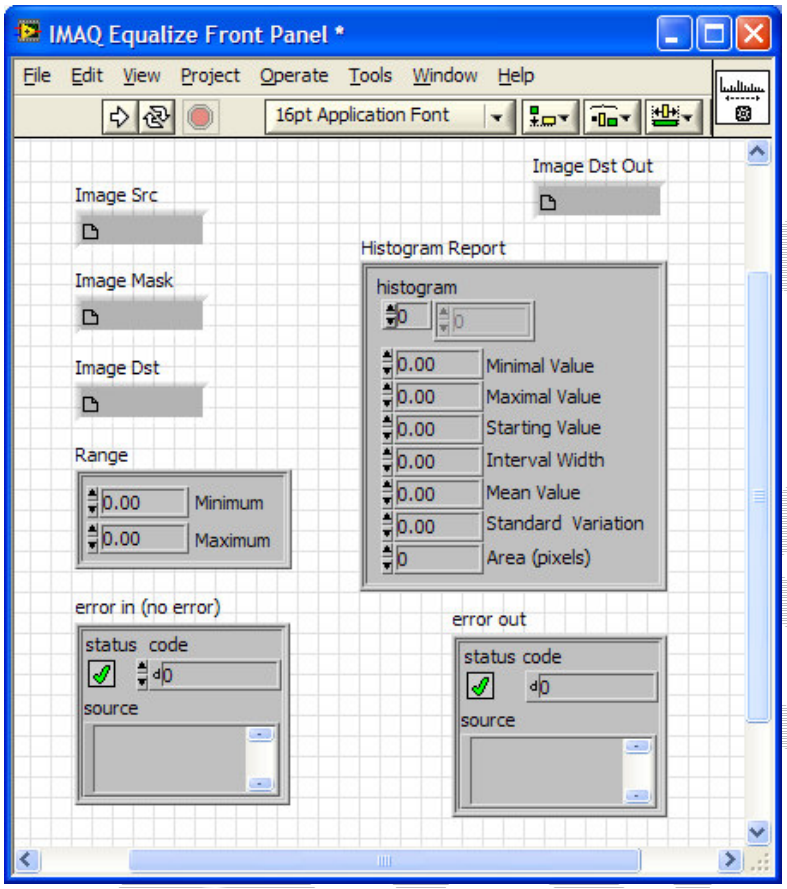
Equalization

Improves contrast on a grayscale image. Calculates the histogram of an image and redistributes pixel values across the desired range to maintain the same pixel value distribution. Pixels whose values are the same before the redistribution also have common pixel values after the redistribution.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16

LabVIEW API: IMAQ Equalize



C API: frcEqualize

int frcEqualize(Image* dest, const Image* source, float min, float max)

int frcEqualize(Image* dest, const Image* source, float min, float max, const Image* mask)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest - the destination image

source - The source image.

min is the smallest value used for processing. After processing, all pixel values that are less than or equal to the **Minimum** in the original image are set to 0 for an 8-bit image. In 16-bit and floating-point images, these pixel values are set to the smallest pixel value found in the original image.

max is the largest value used for processing. After processing, all pixel values that are greater than or equal to the **Maximum** in the original image are set to 255 for an 8-bit image. In 16-bit and floating-point images, these pixel values are set to the largest pixel value found in the original image.
mask is an 8-bit image that specifies the region of the small image that will be copied. Only those pixels in the **Image Src (Small)** image that correspond to an equivalent non-zero pixel in the mask image are copied. All other pixels keep their original values. The entire image is processed if **Image Mask** is NULL or if this parameter is omitted.

Calls:
int imaqEqualize(Image* dest, const Image* source, float min, float max, const Image* mask)

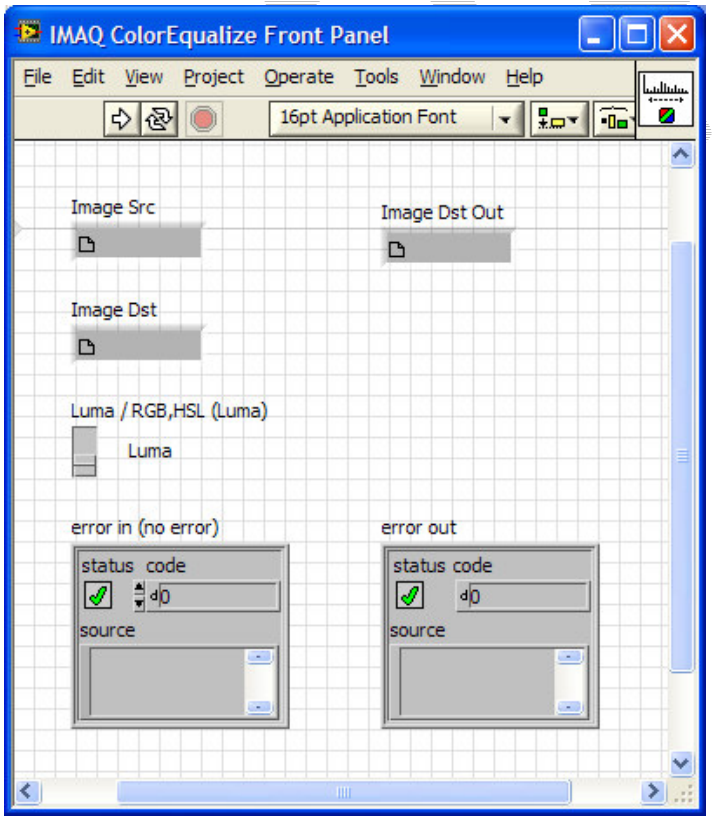
Color Equalization

Improves contrast on a color image. Calculates the histogram of each plane of a color image and redistributes pixel values across the desired range while maintaining pixel value groupings.

Image Types Supported

IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

LabVIEW API: IMAQ ColorEqualize



C API: frcColorEqualize

int frcColorEqualize(Image* dest, const Image* source)
int frcColorEqualize(Image* dest, const Image* source, int colorEqualization)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

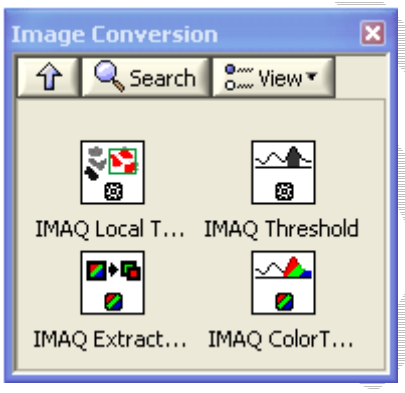
dest - The destination image
source - The image to equalize.
colorEqualization - Set this parameter to TRUE to equalize all three planes of the image (the default). Set this parameter to FALSE to equalize only the luminance plane

Calls:

int imaqColorEqualize(Image* dest, const Image* source, int colorEqualization)

Image Thresholding and Conversion

These VIs are part of the Vision for LabVIEW Library.



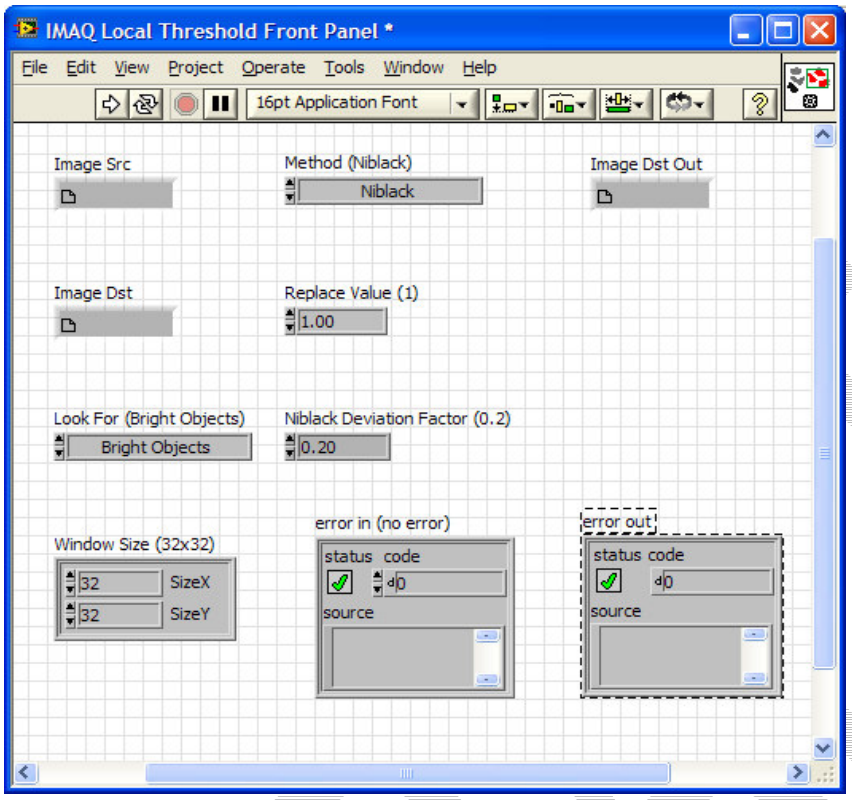
Smart Thresholding

Automatically thresholds a grayscale image into a binary image for Particle Analysis based on a smart threshold. The function sets the kept object pixels with a value that defaults to 1.0.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16

LabVIEW API: IMAQ Local Threshold



C API: frcSmartThreshold

int frcSmartThreshold(Image* dest, const Image* source, unsigned int windowWidth, unsigned int windowHeight, LocalThresholdMethod method, double deviationWeight, ObjectType type)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest - The destination image

source - The image to threshold.

windowWidth - The width of the rectangular window around the pixel on which the function performs the local threshold. This number must be at least 3 and cannot be larger than the width of **source**.

windowHeight - The height of the rectangular window around the pixel on which the function performs the local threshold. This number must be at least 3 and cannot be larger than the height of **source**.

method - Specifies the local thresholding method the function uses. Value can be IMAQ_NIBLACK (which computes thresholds for each pixel based on its local statistics using the Niblack local thresholding algorithm.), or IMAQ_BACKGROUND_CORRECTION (which does background correction first to eliminate non-uniform lighting effects, then performs thresholding using the Otsu thresholding algorithm)

deviationWeight - Specifies the k constant used in the Niblack local thresholding algorithm, which determines the weight applied to the variance calculation. Valid k constants range from 0 to 1. Settings this value to 0 will increase the performance of the function because the function will not calculate the variance for any of the pixels. The function ignores this value if **method** is not set to IMAQ_NIBLACK
type - Specifies the type of objects for which you want to look. Values can be IMAQ_BRIGHT_OBJECTS or IMAQ_DARK_OBJECTS.

See the *NIVisionCVI.chm* help file for descriptions of parameter data structures.

Calls:
int imaqLocalThreshold(Image* dest, const Image* source, unsigned int windowWidth, unsigned int windowHeight, LocalThresholdMethod method, double deviationWeight, ObjectType type, float replaceValue)

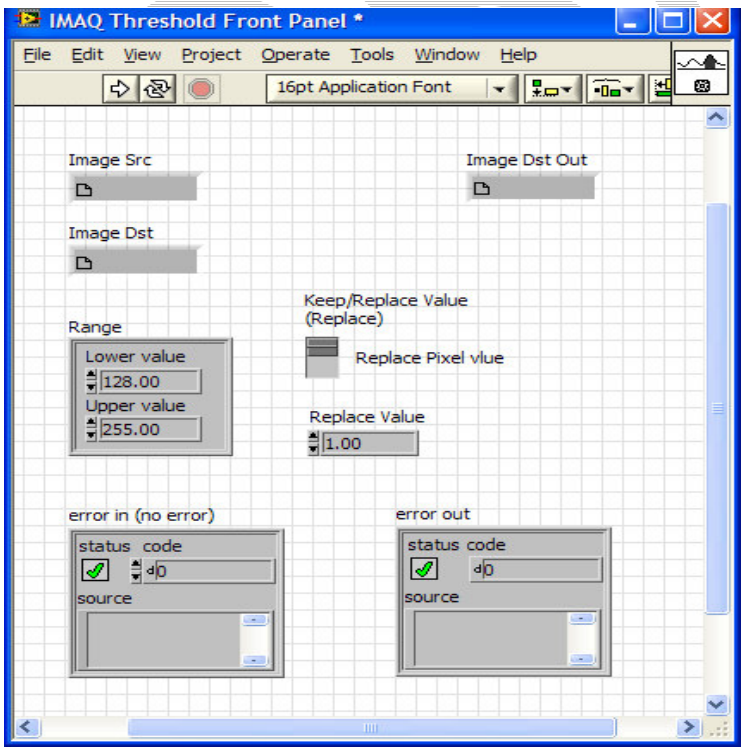
Simple Thresholding

Converts a grayscale image to a binary image for Particle Analysis based on a fixed threshold. Thresholds an image. The function sets pixels values outside of the given range to 0. The function sets pixel values within the range to a given value or leaves the values unchanged.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16

LabVIEW API: IMAQ Threshold



C API: frcSimpleThreshold

int frcSimpleThreshold(Image* dest, const Image* source, float rangeMin, float rangeMax)

int frcSimpleThreshold(Image* dest, const Image* source, float rangeMin, float rangeMax, float newValue)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest - The destination image.
source - The image to threshold.
rangeMin – The lower boundary of the range of pixel values to keep.
rangeMax - The upper boundary of the range of pixel values to keep.
newValue - The replacement value for pixels within the range. Use the simplified call to leave the pixel values unchanged. Defaults to 1.0 if simplified call is used.

Calls:

int imaqThreshold(Image* dest, const Image* source, float rangeMin, float rangeMax, int useNewValue, float newValue)

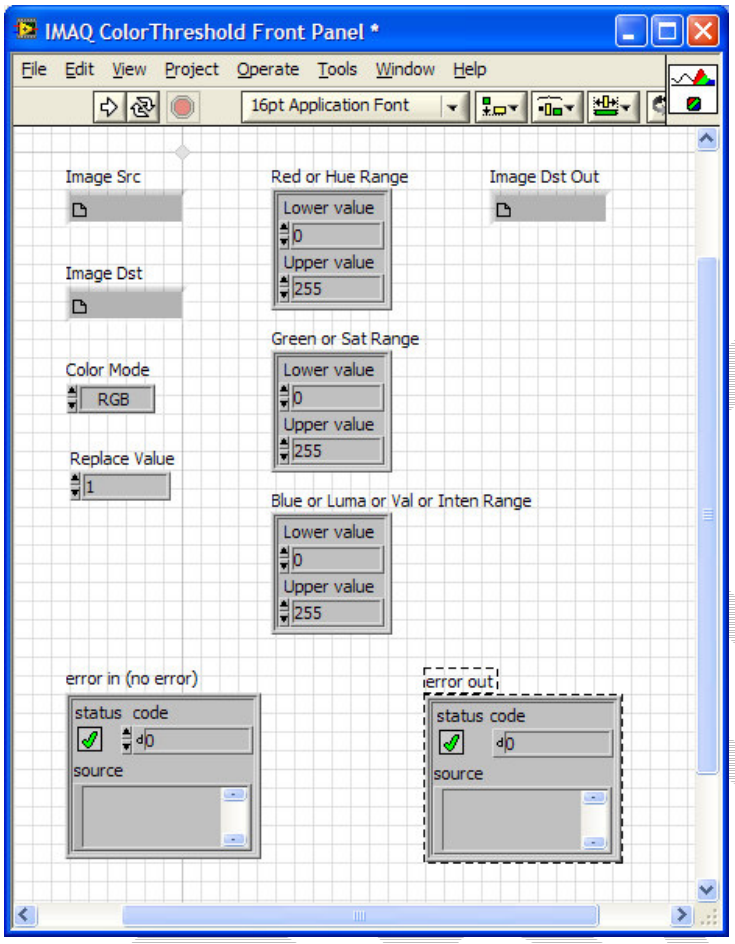
Color Thresholding

Applies a threshold to the Red, Green, and Blue values of a color image. Result is a grayscale image. The function selects a pixel if all three color components fall within the specified range. The function replaces the value of selected pixels with the given replacement value and sets the value of unselected pixels to 0.

Image Types Supported

IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

LabVIEW API: IMAQ Color Threshold



C API: frcColorThreshold

int frcColorThreshold(Image* dest, const Image* source, ColorMode mode, const Range* plane1Range, const Range* plane2Range, const Range* plane3Range)

int frcColorThreshold(Image* dest, const Image* source, int replaceValue, ColorMode mode, const Range* plane1Range, const Range* plane2Range, const Range* plane3Range)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest - The destination image. This image must be an IMAQ_IMAGE_U8 image.
source - The image to threshold.

replaceValue - The value the function assigns to selected pixels Defaults to 1 if simplified call is used.
mode - The color space to perform the threshold in (IMAQ_RGB, IMAQ_HSL, IMAQ_HSV, IMAQ_HSV).
plane1Range - The selection range for the first plane of the image. Set this parameter to NULL to use a selection range from 0 to 255.
plane2Range - The selection range for the second plane of the image. Set this parameter to NULL to use a selection range from 0 to 255.
plane3Range - The selection range for the third plane of the image. Set this parameter to NULL to use a selection range from 0 to 255.

Calls:
IMAQ_FUNC int IMAQ_STDCALL imaqColorThreshold(Image* dest, const Image* source, int replaceValue, ColorMode mode, const Range* plane1Range, const Range* plane2Range, const Range* plane3Range)

Parameter Discussion

plane1Range—The color plane depends on the mode, as follows:

IMAQ_RGB red plane

IMAQ_HSL hue plane

IMAQ_HSV hue plane

IMAQ_HSI hue plane

plane2Range—The color plane depends on mode, as follows:

IMAQ_RGB green plane

IMAQ_HSL saturation plane

IMAQ_HSV saturation plane

IMAQ_HSI saturation plane

plane3Range—The color plane depends on mode, as follows:

IMAQ_RGB blue plane

IMAQ_HSL luminance plane

IMAQ_HSV value plane

IMAQ_HSI intensity plane

Calls:
IMAQ_FUNC int IMAQ_STDCALL imaqColorThreshold(Image* dest, const Image* source, int replaceValue, ColorMode mode, const Range* plane1Range, const Range* plane2Range, const Range* plane3Range)

C API: frcHueThreshold

This version thresholds based on hue range in the HSL mode.

int frcHueThreshold(Image* dest, const Image* source, const Range* hueRange)

int frcHueThreshold(Image* dest, const Image* source, const Range* hueRange, int minSaturation)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

dest - The destination image. This image must be an IMAQ_IMAGE_U8 image.
source - The image to threshold.
hueRange - The selection range for the hue plane of the image. Set this parameter to NULL to use a selection range from 0 to 255.
minSaturation – The minimum saturation level. If called without this parameter a default minimum saturation of 40 value used.

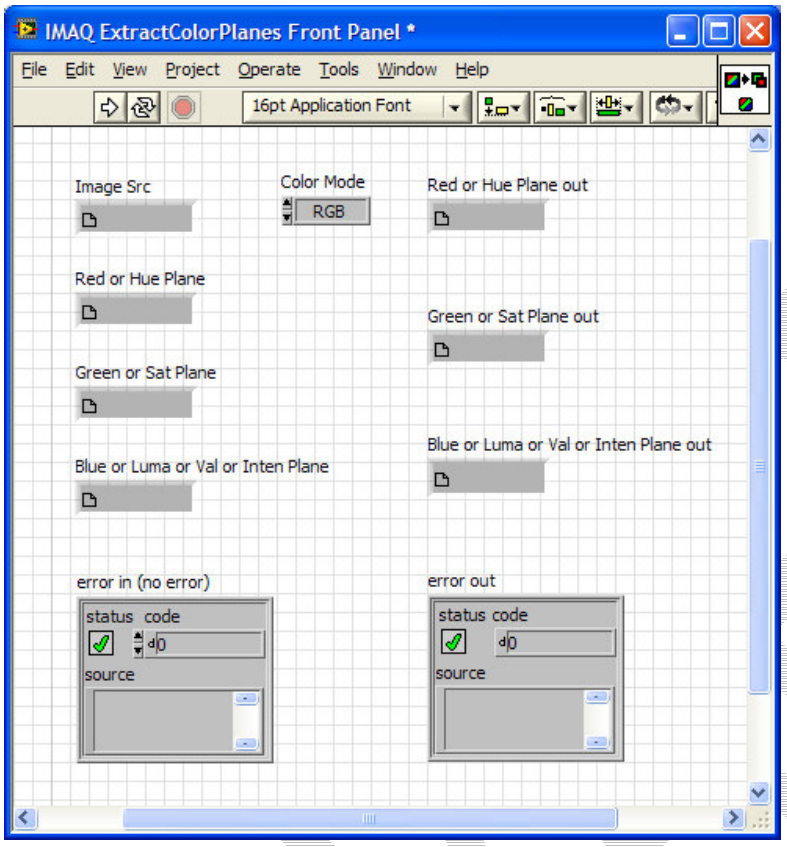
See the *NIVisionCVI.chm* help file for descriptions of parameter data structures.

Color Plane Extraction

These functions extract the Red, Green, Blue, Hue, Saturation or Luminance information from a color image. Result is a grayscale image comprised only of the component selected. The plane you extract may be independent from the type of the image. For example, you can extract the hue plane from a 32-bit RGB image or the green plane from an HSL image.
Image Types Supported

IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL, IMAQ_IMAGE_RGB_U64

LabVIEW API: IMAQ Extract Color Planes



C API: frcExtractColorPlanes

int frcExtractColorPlanes(const Image* image, ColorMode mode, Image* plane1, Image* plane2, Image* plane3)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

image - The source image that the function extracts the planes from.

mode - The color space that the function extracts the planes from (IMAQ_RGB, IMAQ_HSL, IMAQ_HSV, IMAQ_HSV)

plane1 - On return, the first extracted plane. Set this parameter to NULL if you do not need this information.

Plane2 - On return, the second extracted plane. Set this parameter to NULL if you do not need this information.

Plane3 - On return, the third plane. Set this parameter to NULL if you do not need this information

Parameter Discussion

plane1—The data contained in **plane1** depends on the mode, as follows:

Mode	Plane
IMAQ_RGB	Red
IMAQ_HSL	Hue
IMAQ_HSV	Hue
IMAQ_HSI	Hue

plane2—The data contained in **plane2** depends on the mode, as follows:

Mode	Plane
IMAQ_RGB	Green
IMAQ_HSL	Saturation
IMAQ_HSV	Saturation
IMAQ_HSI	Saturation

plane3—The data contained in **plane3** depends on the mode, as follows:

Mode	Plane
IMAQ_RGB	Blue
IMAQ_HSL	Luminance
IMAQ_HSV	Value
IMAQ_HSI	Intensity

Calls:
int imaqExtractColorPlanes(const Image* image, ColorMode mode, Image* plane1, Image* plane2, Image* plane3)

C API: frcExtractHuePlane

This is a simplified version of frcExtractColorPlanes that obtains the Hue plane for a HSL image. It assumes a saturation range of minSaturation – 255, and a luminance range of 0-255.

int frcExtractHuePlane(const Image* image, Image* huePlane, int minSaturation)

Return value:

int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

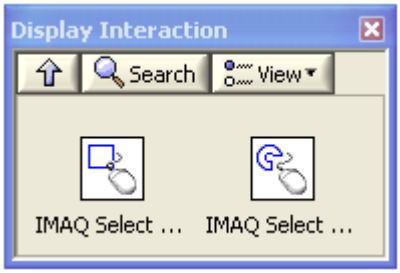
Parameters:

image - The source image that the function extracts the planes from.
huePlane - On return, the extracted hue plane.
minSaturation - The minimum saturation level required (default 40).

Calls:
int imaqExtractColorPlanes(const Image* image, ColorMode mode, Image* plane1, Image* plane2, Image* plane3)

LabVIEW API: Display Interaction

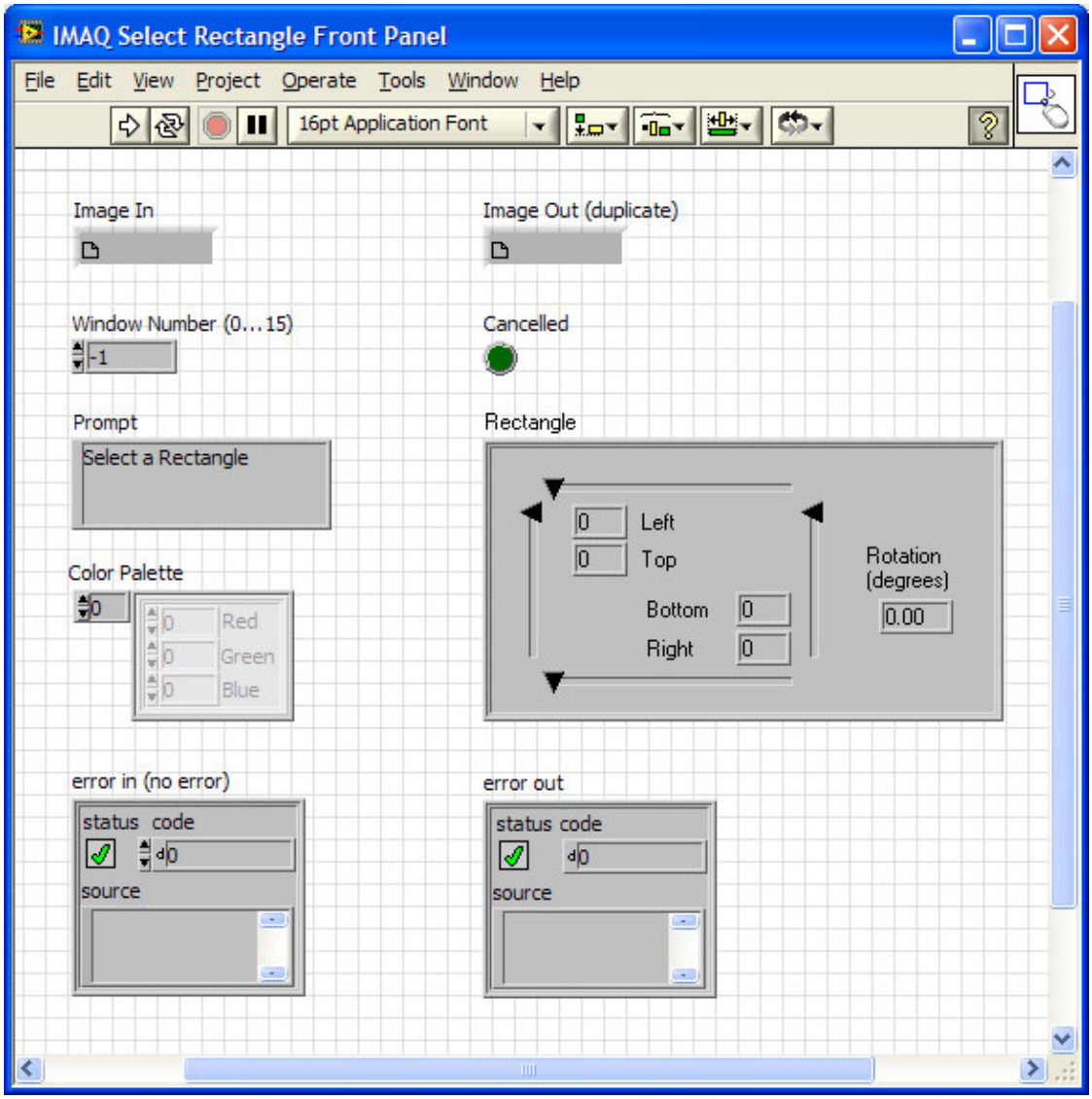
These VIs are part of the Vision for LabVIEW Library. As user interactive VIs, they are not for use in robot programs, but are useful image utilities for viewing images on your PC. They are not visible in the Real Time Palette for Robot Programming, but will be accessible when the target is the PC.



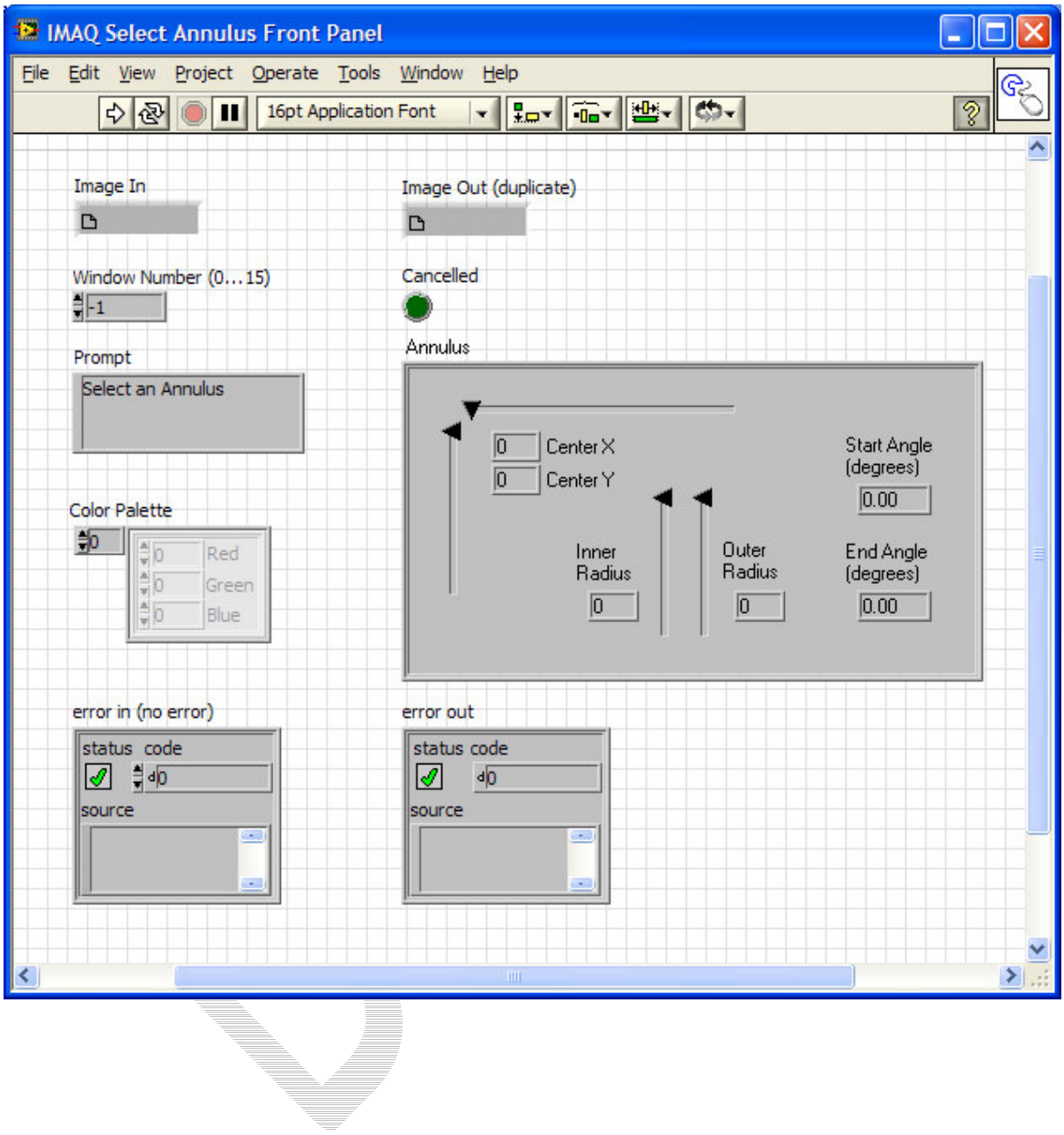
Select Shapes

On the PC, these VIs generate a window that prompts the user to select a shape, either a rectangle or annulus (circle).

LabVIEW API: IMAQ Select Rectangle



LabVIEW API: IMAQ Select Annulus



C API: Camera and Image Acquisition Functions

Functions to support camera management and acquisition of an image from the Axis Camera.

Camera Control

Starts and stops the camera task. This task runs independently, retrieving images from the camera and making them available to tracking and image processing functions in raw and decoded form.

StartCameraTask

Start the camera task. If a camera task is currently running, it is terminated and a new one is started.

int StartCameraTask()

int StartCameraTask(int frames, int compression, ImageSize resolution, ImageRotation rotation)

int StartCameraTask(int frames, int compression, ImageSize resolution, ImageRotation rotation, int decode)

Return value:

int - On success, this function returns the task ID. On failure, this function returns -1. To get extended error information, call GetLastError().

Parameters:

frames – Camera frame rate (1 – 30)
compression – The amount of image compression **(**0-100)
resolution – Determines number of pixels in the image. Possible values are k640x480, k320x240, k160x120
rotation – If the camera is not mounted in the normal position, the image returned may be rotated to compensate. Possible values for ImageRotation are: ROT_0 , ROT_90 , ROT_180, ROT_270
decode – 0: do not decode Images, 1:decode JPEG images into HSL format (default)

StopCameraTask

Stops the camera task.

int StopCameraTask()

Return value:

int – If a task was stopped, this function returns the task ID. If no camera task was running, this function returns –1.

Camera Server Management

Controls whether the server is obtaining images from the camera. Image acquisition is started automatically when the camera task is started. If desired, image acquisition may be stopped and restarted using these functions.

StartImageAcquisition

Start the camera server acquisition of images.

void StartImageAcquisition(void)

StopImageAcquisition

Stop the camera server acquisition of images. The camera server can start up again by running StartImageAcquisition().

void StartImageAcquisition(void)

StartImageSignal

Start serving images. The task will be signaled when a new image is ready.

void StartImageSignal(int taskId)

Parameters:

taskId – The task to be signaled of information. If called with a 0, it turns off signaling.

Axis Image Acquisition

Acquires a single image from the image buffer.

Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_RGB

GetImage

int GetImage(Image* image, double *timestamp)

Return value:

int* - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

image – On return, an image from the camera. Before calling, this image must be created (use frcCreateImage). The image must be disposed of (frcDispose()) when no longer needed.

timestamp – On return, the time that the image was stored.

GetImageBlocking

Gets the latest image, ensuring that it has been updated from the previous image retrieved by comparing timestamps. This routine will block for MAX_BLOCKING_TIME_SEC before returning an error.

int GetImageBlocking(Image* image, double *timestamp, double lastImageTimestamp)

Return value:

Image* - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError().

Parameters:

image – On return, an image from the camera. Before calling, this image must be created (use frcCreateImage). The image must be disposed of (frcDispose()) when no longer needed.

timestamp – On return, the time that the image was stored.

lastImageTimestamp – A timestamp passed in to verify that the image returned is new.

PC Image Server

Manages the task that sends images to the PC. This is a C++ class that when instantiated sends images to the PC to be displayed on the Dashboard. The Camera must be started before creating a PCVideoServer, and the PC must be at IP address 10.x.x.6, where x.x = team number.

Example:

```
StartCameraTask(();          //start camera
PCVideoServer pc;            //start serving images to PC
pc.Stop();                   //stop serving images
pc.Start();                  //start serving images again
```

Camera Metrics

Obtains camera metrics.

GetCameraMetric

Get camera instrumentation counters.

int GetCameraMetric(FrcvCameraMetric metric)

Return value:

int – the value of the counter requested.

Parameters:

metric – The counter queried. This can be:

CAM_STARTS, CAM_STOPS, CAM_NUM_IMAGE, CAM_BUFFERS_WRITTEN,
CAM_BLOCKING_COUNT, CAM_SOCKET_OPEN, CAM_SOCKET_INIT_ATTEMPTS,
CAM_BLOCKING_TIMEOUT, CAM_GETIMAGE_SUCCESS, CAM_GETIMAGE_FAILURE,
CAM_STALE_IMAGE, CAM_GETIMAGE_BEFORE_INIT,
CAM_GETIMAGE_BEFORE_AVAILABLE,CAM_READ_JPEG_FAILURE, CAM_PC_SOCKET_OPEN,
CAM_PC_SENDIMGAGE_SUCCESS, CAM_PC_SENDIMAGE_FAILURE,
CAM_PID_SIGNAL_ERR, CAM_BAD_IMAGE_SIZE, CAM_HEADER_ERROR

C API: High Level Functions for Tracking

These C++ functions are provided to provide a higher level functionality, similar to the tracking data returned by the CMU camera. These functions provide the capability to acquire images and return data based on a color specified. To use these calls, include *TrackAPI.h.*

Tracking Data

This function returns default tracking parameters for the specified color and lighting conditions. The tracking data is returned for hue, saturation and luminance, and may be used as input for FindColor(). These defaults may suffice for quick testing, but it is recommended that the user inspect the desired image with the NI Vision Assistant to obtain the best values.

TrackingThreshold GetTrackingData (FrcHue hue, FrcLight light)

Return value:

TrackingThreshold – This function returns a structure containing HSL ranges.

Parameters:

hue – A predefined hue. Valid values are RED, GREEN, BLUE, YELLOW, ORANGE, PURPLE, WHITE, and PINK.

Light - A predefined type of light. Valid values are PASSIVE_LIGHT, BRIGHT_LIGHT, ACTIVE_LIGHT, WHITE_LIGHT, and FLUORESCENT.

See Appendix A for description of the TrackingThreshold data structure.

Color Tracking

This function obtains a tracking report based on color. It optionally also returns a color report with detailed color statistics. The reports are based on the largest detected particle of the targeted color.

int FindColor(frcHue color, ParticleAnalysisReport *trackReport)

int FindColor(const Range* hueRange, ParticleAnalysisReport *trackReport)

int FindColor(const Range* hueRange, int minSaturation, ParticleAnalysisReport *trackReport)

int FindColor(ColorMode mode, const Range* plane1Range,
const Range* plane2Range, const Range* plane3Range, ParticleAnalysisReport *trackReport)

int FindColor(ColorMode mode, const Range* plane1Range,
const Range* plane2Range, const Range* plane3Range, ParticleAnalysisReport *trackReport, ColorReport *colorReport)

Return value:

Int - On success, this function returns 1. On failure, this function returns 0. To get extended error information, call GetLastError()..

Parameters:

color – One of several predefined color ranges (see Appendix).
mode – Color mode: IMAQ_RGB or IMAQ_HSL. On the simplified calls, HSL is used.
plane1Range - The range for the first plane (hue/Red).
plane2 Range - The range for the second plane (saturation/Green).
plane3 Range - The range for the third plane (luminance/Blue).
hueRange - The range for hue.
minSaturation - The minimum value for saturation (maximum will be 255).
trackReport – On return, pointer to structure containing values concerning the color area found.
colorReport – On return, pointer to structure containing values concerning the color parameters found.

See Appendix A for description of the FrcHue, ParticleAnalysisReport and ColorReport data structures.

C API: Utility Functions

These C++ functions are provided to support testing and error handling. To use these calls, include *TrackAPI.h* or *BaeUtilites.h.*

Debug utilities

dprintf

Optionally prints a debug string to the terminal display, a debug (.dbg) file, or both, depending on the setting of the dprintfFlag (0:OFF, 1:MOSTLY OFF, 2:SCREEN ONLY, 3:FILE ONLY, 4:BOTH). The files are created on the home directory of the cRIO as <sourcefilename>.debug. These files should be deleted from the cRIO when the debug activity is complete to prevent eventual disk overflow.

void dprintf (loggingLevel, <format string>, <optional substitution data> ...)

Parameters:

loggingLevel – Logging level. This can be LOG_DEBUG, LOG_INFO, LOG_ERROR, LOG_CRITICAL, or LOG_FATAL.
Following parameters – Standard C/C++ format string and optional substitution characters.

Example:

char funcName[] = "someFunctionName";
…
dprintf(LOG_DEBUG, "Test number %i", testNum);

SetDebugFlag

Sets the debug flag to write to a file or console output. The debug files are written to the root directory and are called “filename.debug” where filename = name of the calling routine. The files may be ftp’d to the PC for inspection. Be careful to remove these periodically so that they do not get too large and fill up the disk. It is recommended to use file debug levels only during active debugging, and turn off debug (or screen only) most of the time.

void SetDebugFlag (FrcvDebugOutputType flag)

Parameters:

flag – Debug output type. This can be:

```
DEBUG_OFF,                  // no debug
DEBUG_MOSTLY_OFF,   // only critical/fatal errors logged
DEBUG_SCREEN_ONLY, // no file, only console
DEBUG_FILE_ONLY, or   // no console, only file
DEBUG_SCREEN_AND_FILE.      // both console & file
```

PrintReport

Displays the chosen report to the dprintf output.

void PrintReport(ParticleAnalysisReport* myReport)

void PrintReport(ColorReport* myReport)

void PrintReport(TrackingThreshold* myReport)

Parameters:

myReport – The report to print.

ShowActivity

Displays an activity indicator on the console output.

void ShowActivity (char *fmt, ...)

Parameters

printf format – Variable length input to display on the line after the activity indicator.

Error Handling Utilities

GetLastError

Get the code for an error returned from the NI Vision library. This function calls dprintf to write optional debug information to display or debug file. If no debug is desired, either set dprintf to OFF or call imaq_GetErrorCode() instead.

int GetLastError()

Return value:

int – The integer error code

GetErrorText

Gets the text for an error code obtained by GetErrorCode or imaqGetErrorCode. This function calls dprintf to write optional debug information to display or debug file. If no debug is desired, either set dprintf to OFF or call imaq_GetErrorCode() instead.

char* GetErrorText(int errorCode)

Return value:

char* – The text for the error.

Parameters:

errorCode – error code to get the text for

Example:

```
char* errorText;
errorText = GetErrorText(errorCode)
```

Range Conversion Utilitites

NormalizeToRange

Normalizes the input position in the range given to the corresponding position in the range –1 to +1. Note that the particle analysis report provides x and y values that have already been normalized for convenience.

float NormalizeToRange(float normalizedValue)
float NormalizeToRange(float normalizedValue, float minRange, float maxRange)

Return value:

float – The new value representing position in the range

Parameters:

normalizedValue – value in the normalized (-1.0 to 1.0) range

minRange - minimum value of range (if not used in call this is 0.0).
maxRange – maximum value of range (if not used in call this is 1.0).

RangeToNormalized

Normalizes the input position in the range given to the corresponding position in the range –1 to +1. The particle analysis report provides x and y values that have already been normalized for convenience.

double RangeToNormalized(double position, int range)

Return value:
double – The new value within the range –1.0 to 1.0.

Parameters:
position – position in the range
range – size of range (beginning at 0)

Example:
Range originalRange;
originalRange.min = 0; originalRange.max=255;
double myposition = center_x_value_returned_from_ imaqMeasureParticle;
double validDriveInput = RangeToNormalized (myposition, originalRange);

Timing Utilities

GetTime

Returns system time to nanosecond resolution

double GetTime (void)

Return value:
double - system time

ElapsedTime

Returns time elapsed (in seconds) from the input start time.

double ElapsedTime (double startTime)

Parameters:
double - start time

Servo Panning Utilities

Functions to support panning the servo.

SinPosition

Calculates Sine wave increments (-1.0 to +1.0). The first time this is called it sets up the time increment. Subsequent calls give values along the sine wave depending on current time. If the wave is stopped and restarted, it must be reinitialized with a new "first call".

double SinPosition (double *period, double sinStart)

Parameters:

period – length of time in seconds to complete a full sine wave (pan)
sinStart – where the sine wave begins (0.0 = 2 pi; pi/2 = 1.0, etc.)

panInit

Initializes the pan function

void panInit ()
void panInit (double period)

Parameters:

Period – number of seconds for one complete pan

panForTarget

Moves the servo incrementally. panInit() must be called first.

void panForTarget (Servo *panServo, double sinStart)

Parameters:

panServo – pointer to the horizontal servo
sinStart – where in the sine wave to begin the pan; i.e. 0 is center of the servo range, -1 and +1 are opposite ends of the servo range.

Appendix A: FRC Vision API Data Structures

Data structures provided by the LabWindows™/CVI™ library are documented in the *NI Vision for LabWindows/CVI Function Reference Help* file. The following are unique data structures for the FRC C++ API.

ColorReport

Contains information about hue, saturation, and luminance of a particle in an image.

Name	Type	Description
numberParticlesFound	int	Number of particles found for this color
largestParticleNumber	int	Particle index of the largest particle
particleIndex	int	Particle number
particleHueMax	float	Maximum hue of the particle (0-255)
particleHueMin	float	Minimum hue of the particle (0-255)
particleHueMean	float	Mean hue of the particle (0-255)
particleSatMax	float	Maximum saturation of the particle (0-255)
particleSatMin	float	Minimum saturation of the particle (0-255)
particleSatMean	float	Mean saturation of the particle (0-255)
particleLumMax	float	Maximum luminance of the particle (0-255)
particleLumMin	float	Minimum luminance of the particle (0-255)
particleLumMean	float	Mean luminance of the particle (0-255)

ParticleAnalysisReport

Contains information about size, location and quality of a particle in an image.

Name	Type	Description
imageHeight	int	Height in pixels of image
imageWidth	int	Width in pixels of image
imageTimestamp	double	Time that image was retrieved from the camera
particleIndex	int	Particle number
center_mass_x	double	X-coordinate of the point representing the average position of the total particle mass, assuming every point in the particle has a constant density
center_mass_y	double	Y-coordinate of the point representing the average position of the total particle mass, assuming every point in the particle has a constant density.
center_mass_x_normalized	double	Center of mass x value normalized to -1.0 to +1.0 range
center_mass_y_normalized	double	Center of mass y value normalized to -1.0 to +1.0 range
particleArea	double	Area of the particle
boundingRect	Rect	Identifies left, right, top, bottom pixels
particleToImagePercent	double	Percentage of the particle Area covering the Image Area.
particleQuality	double	Percentage of the particle Area in relation to its Particle and Holes Area

Range

A NI data structure used to specify ranges, re-documented here for clarity.

Name	Type	Description
minValue	int	Minimum value of the range
maxValue	int	Maximum value of the range

TrackingThreshold

Contains specification of targeted hue, saturation and luminance.

Name	Type	Description
name	char[64]	Name of the color
hue	Range	Contains minimum and maximum hue range (0-255)
saturation	Range	Contains minimum and maximum saturation range (0-255)
luminance	Range	Contains minimum and maximum luminance range (0-255)

FrcHue

An enumerated type with sample values for colors with pre-specified values for Hue. Valid values are RED, GREEN, BLUE, YELLOW, ORANGE, PURPLE, WHITE, and PINK.

FrcLight

An enumerated type with pre-specified values for Saturation and Luminance. Valid values are PASSIVE_LIGHT, BRIGHT_LIGHT, ACTIVE_LIGHT, WHITE_LIGHT, and FLUORESCENT.