The APEX-CV Pro Library

UG-10328-02-05

Version	Details of Change	Author	Date
01	Initial Revision	J. Lalonde, C. Garrard	August 1, 2014
02	Update to add Image Pyramid, Interpolate, Accumulate, Accumulate Squared and Gaussian.	A. Saechao, J. Cairns	October 1, 2014
03	Update Harris Corner Detection documentation	A. Saechao	October 31, 2014
04	APEX-CV base update: abs, clz	G. Billig	January 29, 2015
05	APEX-CV pro update: BRIEF, Block Matching, Harris, Hough	A. Saechao	July 9, 2015

Contents

Chapter 1

APEX-CV Pro Library

The APEX-CV Pro library provides high-level functionality for developers to design their own computer vision applications while taking advantage of CogniVue's massively parallel APEX-CV architecture. The library contains the following modules:

1. APEX-CV Base Library - Basic image processing functionality.
2. APEX-CV Feature Detection Library
    - Block Matching
    - Binary Robust Independent Elementary Features
    - Harris Corner Detector
    - Hough Line Detector
3. APEX-CV Image Pyramid Library
    - Image Pyramid
4. APEX-CV Image Transform Library
    - Image Remapper
    - Image Resize

Chapter 2

APEX-CV Base Library

The APEX-CV Base library provides basic functionality for developers to design their own imaging-based applications while taking advantage of CogniVue's massively parallel APEX architecture. Currently various arithmetic operations, color conversions and image filters are provided as listed below. The maximum resolution image supported is 512 pixel width.

- Arithmetic Operations:
  - Absolute difference
  - Accumulate
  - Accumulate squared
  - Accumulate weighted
  - Addition
  - Bitwise AND, NOT, OR, XOR
  - Magnitude
  - Subtraction
  - Thresholding
- Interpolation:
  - Bilinear Grayscale
  - Linear Grayscale
- Color Conversions:
  - RGB565 to RGB888
  - RGB888 to RGB565
  - RGB888 to Y
  - RGB888 to YUV
- Image Filters:
  - Bilateral filter
  - Box filter
  - Convolve filter
  - Dilate filter

Chapter 3

Block Matching

The Block Matching algorithm is used to locate matching macroblocks between two images. This is done in APEX-CV by using the Sum of Absolute Differences approach. See apex::Blockmatching.

To perform the search, a number of search points and locations need to be specified. The the algorithm will then search within a 28x28 region of pixels, the search window, around those points for a matching macroblock. This is done by calculating the SAD score for all 16x16 region of pixels within the search window. A block is considered to be matching if the SAD score is below a user specified maximum SAD threshold.

Chapter 4

Binary Robust Independent Elementary Features

BRIEF is a fast method for the feature descriptor calculation. It finds the binary strings directly without calculating descriptors in floating point numbers. BRIEF takes smoothed image patch and selects a set of nd(x,y) location pairs in Gaussian distribution pattern. Then pixel intensity comparisons are done for each pair, and the results are stored in binary. This is applied for all the nd location pairs to get a nd-dimensional bitstring.

APEX-CV BRIEF is implemented based on OpenCV BRIEF implementation. First, the sum of 9x9 pixel patch is calculated. To reduce the computational cost, integral image is used. Then, the comparison of pixel intensity between selected pair is performed. This comparison pairs are selected from 48x48 regions around a keypoint, with Gaussian distribution pattern. The result of comparison is stored as binary string. For example, let one location pair be p and q. If I(p) $<$ I(q), then its result is 1, else it is 0. The size of descriptor is either 16 (default), 32, or 64 bytes.

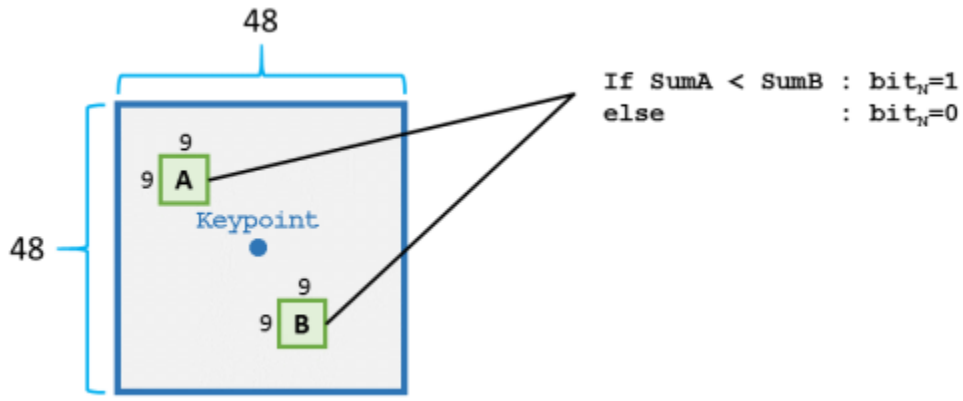

Figure 4.1: APEX-CV BRIEF

Chapter 5

Harris Corner Detector

5.1 Overview

The APEX-CV Harris Corner Detector computes a 16-bit corner response image from an 8-bit grayscale image. The algorithm is a fixed point implementation of [3] and is similar to the function `cornerHarris` in OpenCV.

5.2 Calculating Harris Corner Response

When calculating the corner response, a threshold and mask are used to filter out responses that do not meet the minimum score and points where a corner should not be returned. A mask value of 0xFF will allow that point to return a corner response. Points that are turned off by the mask or by the threshold will be set to INT16_MIN (-32768). For example, to enable every point as a possible corner and to only keep responses above 0:

```
ICP_ContigDataDesc lSrc;
ICP_ContigDataDesc lDst;
ICP_ContigDataDesc lThreshold;
ICP_ContigDataDesc lMask;
// Memory allocation
...

uint8_t* lpMask   = (uint8_t*)lMask.RetDataPtr();
int16_t* lpThresh = (int16_t*)lThreshold.RetDataPtr();
memset(lpMask, 0xFF, sizeof(uint8_t*)*width*height);
lpThresh[0] = 0;

apex::HarrisCornerDetector harris;
harris.initialize(lSrc, lMask, lThreshold, lDst);
harris.process();
```

The *initialize(...)* function can also take in the Sobel filter size (3 or 5), Box filter size (3 or 5), and the Harris coefficient (0-255). See member documentation for more information.

5.3 Sorting the Harris Corner Responses

The corner response can be sorted for the largest responses, with a minimum distance in between each corner using the Harris Sort Corners function. This function is an ARM based implementation which does not run on the APUs.

This function utilizes the ICP_FeatureDesc class to store the corners. This class is a container for the ICP_Feature structure which stores the position (x, y) and strength of a feature. Note: Memory allocation and deallocation is NOT handled by this class.

For example, to sort the corner responses obtained in the section Calculating Harris Corner Response :

```
ICP_FeatureDesc lCorners;

// Memory Allocation
void* lpCornersOal = OAL_MemoryAllocFlag(sizeof(ICP_Feature)*10, OAL_MEMORY_FLAG_ALIGN(
      ALIGN2_CACHELINE)|OAL_MEMORY_FLAG_CONTIGUOUS);

lCorners.Init(OAL_MemoryReturnAddress(lpCornersOal, ACCESS_NCH_NB),
              OAL_MemoryReturnAddress(lpCornersOal, ACCESS_PHY),
              10);

harris.sortCorners(lDst,      /* The output from harris.getScores */
                   lCorners,  /* The ICP_FeatureDesc object */
                   10,        /* The minimum distance between corners */
                   10         /* The maximum number of corners to return */
                  );

printf("Number of corners: %d\n", lCorners.RetCount();

for (int i = 0; i < lCorners.RetCount(); ++i) {
   printf("Corner (%d, %d) strength: %d\n", lCorners[i].position.x, lCorners[i].position.y, lCorners[i].
       strength);
}
```

This will then populate the ICP_FeatureDesc *lCorners* with the corners sorted from largest response to smallest response.

5.4 ICP Feature Descriptor

The ICP Feature Descriptor Class, ICP_FeatureDesc, is used to store in a contiguous region of memory data of type ICP_Feature.

The ICP_Feature structure contains the position of the feature, as a ICP_Point_16S type, and the strength of the feature as a int16_t type. The ICP_Point_16S type has two public members: int16_t x and int16_t y.

See class documentation for more information.

Chapter 6

Hough Line Detector

6.1 Overview

The APEX-CV Hough Line Detector detects lines from an 8-bit grayspcale image. The algorithm is based on [2] and is similar to the function `HoughLines` in OpenCV. A good overview of the Hough transform can be found on `Wikepedia`

6.2 Line Representation

The detected lines are expressed in polar coordinates (ρ, θ), where ρ is the nearest distance of the line to the image center (c_x, c_y) and θ is angle of the normal to the line. This is shown below.

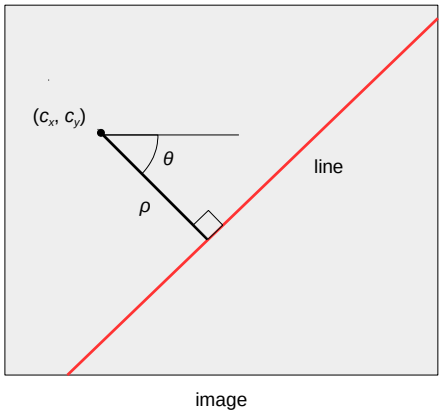

Figure 6.1: The geometric interpretation of the Hough line coordinates (rho, theta).

6.3 Supported Image Sizes

Currently four image widths are supported. Images passed to the detector must be exactly one of those widths. The image height must be a multiple of four. The maximum image height is determined by the image width and the available APU memory for the Hough accumulator. The supported widths and the corresponding maximum height are shown in the following table.

Supported Width	Maximum Height
192	1588
192	1568
640	1468
1280	960

Figure 6.2: Supported image sizes for the Hough Line Detector.

6.4 Specifying Angles for Detection

The APEX-CV Hough Line Detector is designed for maximum flexibility. The simplest way to specify the detection angles is by specifying the number of angles only. In this case the detector divides the range of angles $[0, \pi[$ evenly by the number of angles specified. A maximum of 256 angles can be specified. For example, calling

```
apex::HoughLineDetector hough;
ICP_ContigDataDesc image;
// ...setup image
hough.initialize(image);
hough.setTheta(180);
```

gives a detector with a line resolution of 1° (since $1^\circ = pi/180$ radians).

For more flexibility, the angle starting value and angle resolution can be specified. Angles must be expressed in radians. For those more familiar with degrees, the conversion factors apex::HoughLineDetector::rad2deg and apex::HoughLine-Detector::deg2rad are supplied to convert from radians to degrees and degrees to radians, respectively. For example

```
using namespace apex;
HoughLineDetector hough;
ICP_ContigDataDesc image;
// ...setup image
hough.initialize(image);
hough.setTheta(32, 15*HoughLineDetector::deg2rad, 5*
      HoughLineDetector::deg2rad);
```

gives a detector sensitive to 32 lines with angles starting from 15° and each separated by 5°.

For the most flexibility, an arbitrary set of up to 256 angles my be specified. These angles are expressed in radians and are passed to the detector as an array of floats. For example,

```
const int thetaCount = 10;
float theta[thetaCount] = {-0.02f, -0.01f, 0.f, 0.01f, 0.02f, 1.55f, 1.56f, 1.57f, 1.58f, 1.59f};
apex::HoughLineDetector hough;
ICP_ContigDataDesc image;
// ...setup image
hough.initialize(image);
hough.setTheta(thetaCount, theta);
```

gives a detector sensitive only to vertical and horizontal lines (lines about 0 and $pi/2$ radians).

6.5 Non-Maxima Suppression

By default, non-maxima suppression (NMS) is performed on the Hough accumulator. NMS is simply a comparison of neighbouring accumulator values in both the rho and theta directions; If the accumulator value is greater than the previous value and greater or equal to the next value (in both directions), then that line is considered for detection (the remaining condition being that the accumulator value be above the specified threshold).

Control over NMS is provided by the flag apex::HoughLineDetector::NonMaxSupFlag. The possible states are

1. NMS_NONE: No NMS is performed.
2. NMS_RHO: NMS is performed in rho direction only.
3. NMS_THETA: NMS is performed in theta direction only.
4. NMS_RHO|NMS_THETA: NMS is performed in both directions (default state).

It is recommended to always use NMS in the rho direction (as the rho resolution is only 1 pixel). However, if a coarse angle resolution is used (i.e. the angle step is large), then NMS between angles should be turned off. For example

```
using namespace apex;
HoughLineDetector hough;
ICP_ContigDataDesc image;
// ...setup image
hough.initialize(image);
hough.setTheta(32, 15*HoughLineDetector::deg2rad, 5*
      HoughLineDetector::deg2rad, HoughLineDetector::NMS_RHO)
      ;
```

disables NMS between angles, which is appropriate since the angle resolution (5°) is coarse.

Chapter 7

Image Pyramid

There are two common kinds of image pyramids: Gaussian pyramids and Laplacian pyramids. Here, we present the APEX-CV Image Pyramid, an implementation of Gaussian image pyramid creation.

To upsample an image, the source image is upsized 2x in each dimension, with the new even-numbered rows and columns filled with zeros. Then, the expanded image is convolved with the 5x5 Gaussian kernel below. As a result, the area is increased to exactly four times the area of the source image.

$$\frac{1}{64}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Figure 7.1: The Gaussian Matrix

To downsample an image, first the source image is convolved with the above 5x5 Gaussian kernel (divided by 4), then every even-numbered row and column is removed, 2x downsampling in each dimension. As a result, the area is reduced to exactly one-quarter the area of the source image.

Chapter 8

Image Remapper

The APEX-CV Remapper maps one image to another from a floating point lookup table. The algorithm is similar to the function `remap` in OpenCV.

The current implementation of apex::Remapper exists as an example application, designed for a specific set of input image size, output image size, and float map. (Increasing the flexibility in supported block and image sizes will be addressed in future releases.) The map files supported by apex::Remapper are found in ICP_SDK/src/Features/ICP/ALGO/Remap/data, as follows:

- RGB remap: dewarp_320_240_RGB.map, source image size 320x240, destination image size 320x240, source block size 8x9, destination block size 10x5
- Greyscale : dewarp_640_480_RGB.map, source image size 640x480, destination image size 640x480, source block size 8x17, destination block size 20x5

Chapter 9

Image Resize

The Image Resize performs a vertical and horizontal resize on the input image, according to the required size of the output image. Note that vertical and horizontal scale factors (and directions) are independent. It is similar to the function `resize()` in OpenCV.

Currently, only integer scale factors are supported for both vertical and horizontal scaling. Upsampling and downsampling are both supported. The maximum output width currently supported is 1024 pixels. There is no current restriction on the input image height.

Currently, the output width must satisfy the equation 32n, where n$>$1. There may be issues with widths which are an odd multiple of 32, this matter is still under investigation.

The maximum supported vertical upscale factor is 32.

All of the above restrictions should be addressed in future development..

Chapter 10

Class Index

10.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 11

Class Documentation

11.1 apex::Arithmetic Class Reference

Arithmetic class containing various arithmetic functions.

`#include <ApexCV_base.hpp>`

Public Member Functions

- Arithmetic ()

  Default constructor.
- ~Arithmetic ()

  Destructor.
- int abs (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Absolute value function.
- int absdiff (const ICP_ContigDataDesc &src0, const ICP_ContigDataDesc &src1, ICP_ContigDataDesc &dst)

  Absolute difference function.
- int accumulate (const ICP_ContigDataDesc &src0, const ICP_ContigDataDesc &src1, ICP_ContigDataDesc &dst)

  Accumulate function.
- int accumulateSquared (const ICP_ContigDataDesc &src0, const ICP_ContigDataDesc &src1, ICP_ContigDataDesc &dst, unsigned char scale)

  Accumulate Squared function.
- int accumulateWeighted (const ICP_ContigDataDesc &src0, const ICP_ContigDataDesc &src1, ICP_ContigDataDesc &dst, unsigned char alpha)

  Accumulate Weighted function.
- int add (const ICP_ContigDataDesc &src0, const ICP_ContigDataDesc &src1, ICP_ContigDataDesc &dst)

  Addition function.
- int bitwiseAND (const ICP_ContigDataDesc &src0, const ICP_ContigDataDesc &src1, ICP_ContigDataDesc &dst)

  Bitwise AND function.
- int bitwiseNOT (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Bitwise NOT function.
- int bitwiseOR (const ICP_ContigDataDesc &src0, const ICP_ContigDataDesc &src1, ICP_ContigDataDesc &dst)

  Bitwise OR function.

- int bitwiseXOR (const ICP_ContigDataDesc &src0, const ICP_ContigDataDesc &src1, ICP_ContigDataDesc &dst)

  Bitwise XOR function.
- int clz (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Count leading zeros function.
- int magnitude (const ICP_ContigDataDesc &src0, const ICP_ContigDataDesc &src1, ICP_ContigDataDesc &dst)

  Magnitude function.
- int subtract (const ICP_ContigDataDesc &src0, const ICP_ContigDataDesc &src1, ICP_ContigDataDesc &dst)

  Subtraction function.
- int threshold (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst, unsigned int threshold)

  Threshold function.

11.1.1 Detailed Description

Arithmetic class containing various arithmetic functions.

This class is an interface for using arithmetic functions on the host.

11.1.2 Constructor & Destructor Documentation

11.1.2.1 apex::Arithmetic::Arithmetic ( )

Default constructor.

11.1.2.2 apex::Arithmetic::~Arithmetic ( )

Destructor.

11.1.3 Member Function Documentation

11.1.3.1 int apex::Arithmetic::abs (const ICP_ContigDataDesc & *src*, ICP_ContigDataDesc & *dst*)

Absolute value function.

Calculates the absolute value of *src0* buffer and stores the result in *dst* buffer.

Supported datatypes are:

- signed 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src	Source 0 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.2 int apex::Arithmetic::absdiff (const ICP_ContigDataDesc & *src0,* const ICP_ContigDataDesc & *src1,* ICP_ContigDataDesc & *dst*)

Absolute difference function.

Calculates the absolute difference between *src0* and *src1* buffers and stores the result in *dst* buffer.

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src0	Source 0 memory buffer. See ICP_ContigDataDesc [1]
src1	Source 1 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.3 int apex::Arithmetic::accumulate (const ICP_ContigDataDesc & *src0,* const ICP_ContigDataDesc & *src1,* ICP_ContigDataDesc & *dst*)

Accumulate function.

Calculates the addition of the *src0* and *src1* buffers and stores the result in *dst* buffer.

Supported datatypes are:

- unsigned 8 bit to signed 16 bit

Returns

Error code (zero on success).

Parameters

src0	Source 0 memory buffer. See ICP_ContigDataDesc [1]
src1	Source 1 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.4 int apex::Arithmetic::accumulateSquared (const ICP_ContigDataDesc & *src0,* const ICP_ContigDataDesc & *src1,* ICP_ContigDataDesc & *dst,* unsigned char *scale*)

Accumulate Squared function.

Calculates the addition of the *src0* and *src1* (squared) buffers and stores the result in *dst* buffer.

Supported datatypes are:

- unsigned 8 bit to signed 16 bit

Returns

Error code (zero on success).

Parameters

src0	Source 0 memory buffer. See ICP_ContigDataDesc [1]
src1	Source 1 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]
scale	Shift amount. Value of [0, 15]

11.1.3.5 int apex::Arithmetic::accumulateWeighted (const ICP_ContigDataDesc & *src0,* const ICP_ContigDataDesc & *src1,* ICP_ContigDataDesc & *dst,* unsigned char *alpha*)

Accumulate Weighted function.

Accumulate weighted uses the equation: $dst = src_0 * (1 - \frac{alpha}{256}) + src_1 * \frac{alpha}{256}$.

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src0	Source 0 memory buffer. See ICP_ContigDataDesc [1]
src1	Source 1 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]
alpha	Weight amount. 8 bit fixed point value of [0, 1)

11.1.3.6 int apex::Arithmetic::add (const ICP_ContigDataDesc & *src0,* const ICP_ContigDataDesc & *src1,* ICP_ContigDataDesc & *dst*)

Addition function.

Adds *src0* and *src1* buffers and stores the result in *dst* buffer. In case of overflow the result is saturated.

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit
- signed 16 bit to signed 16 bit

Returns

Error code (zero on success).

Parameters

src0	Source 0 memory buffer. See ICP_ContigDataDesc [1]
src1	Source 1 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.7 int apex::Arithmetic::bitwiseAND (const ICP_ContigDataDesc & *src0*, const ICP_ContigDataDesc & *src1*, ICP_ContigDataDesc & *dst*)

Bitwise AND function.

Bitwise ANDs *src0* and *src1* buffers and stores the result in *dst* buffer. See (http://en.wikipedia.org/wiki/Bitwise_operation) for more information on bitwise operations.

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit
- unsigned 16 bit to unsigned 16 bit
- unsigned 32 bit to unsigned 32 bit

Returns

Error code (zero on success).

Parameters

src0	Source 0 memory buffer. See ICP_ContigDataDesc [1]
src1	Source 1 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.8 int apex::Arithmetic::bitwiseNOT (const ICP_ContigDataDesc & *src*, ICP_ContigDataDesc & *dst*)

Bitwise NOT function.

Bitwise NOTs *src* buffer and stores the result in *dst* buffer. See (http://en.wikipedia.org/wiki/Bitwise_operation) for more information on bitwise operations.

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src	Source 0 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.9 int apex::Arithmetic::bitwiseOR (const ICP_ContigDataDesc & *src0*, const ICP_ContigDataDesc & *src1*, ICP_ContigDataDesc & *dst*)

Bitwise OR function.

Bitwise ORs *src0* and *src1* buffers and stores the result in *dst* buffer. See (http://en.wikipedia.org/wiki/-Bitwise_operation) for more information on bitwise operations.

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit
- unsigned 16 bit to unsigned 16 bit
- unsigned 32 bit to unsigned 32 bit

Returns

Error code (zero on success).

Parameters

src0	Source 0 memory buffer. See ICP_ContigDataDesc [1]
src1	Source 1 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.10 int apex::Arithmetic::bitwiseXOR (const ICP_ContigDataDesc & *src0*, const ICP_ContigDataDesc & *src1*, ICP_ContigDataDesc & *dst*)

Bitwise XOR function.

Bitwise XORs *src0* and *src1* buffers and stores the result in *dst* buffer. See (http://en.wikipedia.-org/wiki/Bitwise_operation) for more information on bitwise operations.

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit
- unsigned 16 bit to unsigned 16 bit
- unsigned 32 bit to unsigned 32 bit

Returns

Error code (zero on success).

Parameters

src0	Source 0 memory buffer. See ICP_ContigDataDesc [1]
src1	Source 1 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.11 int apex::Arithmetic::clz (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst*)

Count leading zeros function.

Counts the leading zeros of *src0* buffer and stores the result in *dst* buffer.

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit
- signed 8 bit to unsigned 8 bit
- unsigned 16 bit to unsigned 8 bit
- signed 16 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src	Source 0 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.12 int apex::Arithmetic::magnitude (const ICP_ContigDataDesc & *src0,* const ICP_ContigDataDesc & *src1,* ICP_ContigDataDesc & *dst*)

Magnitude function.

Calculates the magnitude between two sources following the format $dst = \sqrt{src_0^2 + src_1^2}$. The result is floored to the nearest integer. Note if you want to calculate the gradient magnitude you must calculate the image gradients and pass them in to this function.

Supported datatypes are:

- signed 16 bit to unsigned 16 bit

Returns

Error code (zero on success).

Parameters

src0	Source 0 memory buffer. See ICP_ContigDataDesc [1]
src1	Source 1 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.13 int apex::Arithmetic::subtract (const ICP_ContigDataDesc & *src0*, const ICP_ContigDataDesc & *src1*, ICP_ContigDataDesc & *dst*)

Subtraction function.

Subtracts *src0* from *src1* buffers and stores the result in *dst*. In case of over/underflow the result is saturated.

Supported datatypes are:

- unsigned 8 bit to signed 16 bit
- signed 16 bit to signed 16 bit

Returns

Error code (zero on success).

Parameters

src0	Source 0 memory buffer. See ICP_ContigDataDesc [1]
src1	Source 1 memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.1.3.14 int apex::Arithmetic::threshold (const ICP_ContigDataDesc & *src*, ICP_ContigDataDesc & *dst*, unsigned int *threshold*)

Threshold function.

Returns an 8 bit boolean image based on the equation: $dst = \begin{cases} 255 & \text{if } src > threshold \\ 0 & \text{otherwise} \end{cases}$

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit
- unsigned 16 bit to unsigned 8 bit
- unsigned 32 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src	Source 0 memory buffer. See ICP_ContigDataDesc [1]

dst	Destination memory buffer. See ICP_ContigDataDesc [1]
threshold	Threshold value.

11.2 apex::Blockmatching Class Reference

Blockmatching class.

`#include <ApexCV_blockmatching.hpp>`

Public Member Functions

- Blockmatching ()

  Default constructor.
- ~Blockmatching ()

  Destructor.
- int initialize (ICP_ContigDataDesc &lOutput0, ICP_ContigDataDesc &lOutput0Status, const ICP_ContigDataDesc &lInput0, const ICP_ContigDataDesc &lInput1, const ICP_ContigDataDesc &lPointsArray, int sadThreshold)

  Initialize the block matching.
- int process ()

  Match the blocks.
- void release ()

  Release Resources.
- int reconnectIO (ICP_ContigDataDesc &lOutput0Points, ICP_ContigDataDesc &lOutput0Status, const ICP_ContigDataDesc &lInput0, const ICP_ContigDataDesc &lInput1, const ICP_ContigDataDesc &lPointsArray)

  Reconnect IO.
- void setSadThreshold (int sadThreshold)

  Set SAD Threshold.

11.2.1 Detailed Description

Blockmatching class.

This class is an interface for using the block matching algorithm on the APEX.

11.2.2 Constructor & Destructor Documentation

11.2.2.1 apex::Blockmatching::Blockmatching ( )

Default constructor.

Allocate resources for the ACF processes.

11.2.2.2 apex::Blockmatching::~Blockmatching ( )

Destructor.

Release resources for the ACF processes.

11.2.3 Member Function Documentation

11.2.3.1 int apex::Blockmatching::initialize (ICP_ContigDataDesc & *IOutput0,* ICP_ContigDataDesc & *IOutput0Status,* const ICP_ContigDataDesc & *IInput0,* const ICP_ContigDataDesc & *IInput1,* const ICP_ContigDataDesc & *IPointsArray,* int *sadThreshold*)

Initialize the block matching.

Initializes the internal buffers for the process.

Parameters

IOutput0	Matched points on the new image
IOutput0Status	Indicates if the points SAD score is below the threshold
IInput0	"Previous" image
IInput1	"New" Image
IPointsArray	Set of points on the previous image and will be used to specify the locations to be searched on the new image
sadThreshold	Maximum SAD score

11.2.3.2 int apex::Blockmatching::process ( )

Match the blocks.

The block matching process uses the Sum of Absolute Differences algorithm to perform block matching. The process will search in the "new" image within a window around the search points in the "previous" image. The template size determines how many pixels are used in the SAD calculation. The window size determines the area in which the template is used to perform SAD calculations.

Supported Template Size: 16x16 Supported Window Size: 28x28

11.2.3.3 int apex::Blockmatching::reconnectIO (ICP_ContigDataDesc & *IOutput0Points,* ICP_ContigDataDesc & *IOutput0Status,* const ICP_ContigDataDesc & *IInput0,* const ICP_ContigDataDesc & *IInput1,* const ICP_ContigDataDesc & *IPointsArray*)

Reconnect IO.

Reconnects the input and outputs to the blockmatching process. This only needs to be done if the connected Input/-Outputs are changed. If only the data within (no size, or type changes), then this does not need to be called.

Parameters

IOutput0Points	Matched points on the new image
IOutput0Status	Indicates if the points SAD score is below the threshold
IInput0	"Previous" image
IInput1	"New" Image
IPointsArray	Set of points on the "previous" image and will be used to specify the locations to be searched on the new image

11.2.3.4 void apex::Blockmatching::release ( )

Release Resources.

Releases the internal buffers and resets the class state to initial.

11.2.3.5 void apex::Blockmatching::setSadThreshold (int *sadThreshold*)

Set SAD Threshold.

Change the SAD threshold

Parameters

sadThreshold	Maximum SAD score

11.3 apex::Brief Class Reference

BRIEF class.

`#include <ApexCV_brief.hpp>`

Public Types

- enum BRIEF_DESCRIPTOR_SIZE { **BRIEF_DESCSIZE_NONE** = -1, **BRIEF_DESCSIZE_16** = 0, **BRIEF_DESCSIZE_32**, **BRIEF_DESCSIZE_64** }

  Descriptor size enum.

Public Member Functions

- Brief ()

  Default constructor.
- ~Brief ()

  Default destructor.
- int initialize (const ICP_ContigDataDesc &src, const ICP_ContigDataDesc &keypointIn, const ICP_ContigDataDesc &keypointInCnt, ICP_ContigDataDesc &keypointOut, ICP_ContigDataDesc &keypointOutCnt, ICP_ContigDataDesc &descriptor)

  Initializes the class.
- int setDescriptorSize (int descriptorSize)

  Set descriptor size.
- int reconnectIO (const ICP_ContigDataDesc &src, const ICP_ContigDataDesc &keypointIn, const ICP_ContigDataDesc &keypointInCnt, ICP_ContigDataDesc &keypointOut, ICP_ContigDataDesc &keypointOutCnt, ICP_ContigDataDesc &descriptor)

  Connects the input/outputs to the process.
- int process ()

  Run BRIEF process.

11.3.1 Detailed Description

BRIEF class.

This class is an interface for using the BRIEF algorithm.

11.3.2 Member Enumeration Documentation

11.3.2.1 enum apex::Brief::BRIEF_DESCRIPTOR_SIZE

Descriptor size enum.

These enums are used to specify the descriptor size of the brief descriptor. Default is BRIEF_DESCSIZE_16. Available:

- BRIEF_DESCSIZE_NONE
- BRIEF_DESCSIZE_16 - 16 byte descriptors
- BRIEF_DESCSIZE_32 - 32 byte descriptors
- BRIEF_DESCSIZE_64 - 64 byte descriptors

11.3.3 Constructor & Destructor Documentation

11.3.3.1 apex::Brief::Brief ( )

Default constructor.

11.3.3.2 apex::Brief::∼Brief ( )

Default destructor.

11.3.4 Member Function Documentation

11.3.4.1 int apex::Brief::initialize (const ICP_ContigDataDesc & *src,* const ICP_ContigDataDesc & *keypointIn,* const ICP_ContigDataDesc & *keypointInCnt,* ICP_ContigDataDesc & *keypointOut,* ICP_ContigDataDesc & *keypointOutCnt,* ICP_ContigDataDesc & *descriptor*)

Initializes the class.

Connects the buffers, and allocates any internal buffers.

Parameters

src	Source memory buffer. Datatype is ICP_DATATYPE_08U.
keypointIn	Input keypoint (x,y) data. Datatype is ICP_DATATYPE_16S. Data has to be stored in a chunk with size 32∗X, where X can be 1 through 31.
keypointInCnt	Input keypoint count. Datatype is ICP_DATATYPE_16U.
keypointOut	Output keypoint (x,y) data. Datatype is ICP_DATATYPE_16S.
keypointOutCnt	Output keypoint count. Datatype is ICP_DATATYPE_16U.
descriptor	Destination memory buffer. Datatype is ICP_DATATYPE_08U.

11.3.4.2 int apex::Brief::process ( )

Run BRIEF process.

BRIEF descriptor is extracted for given *src* image and *keypoint* data, with *descriptor* size.

Supported datatypes are:

- unsigned 8 bit image, signed 16 bit input keypoint, unsigned 16 bit input keypoint count to signed 16 bit output keypoint, unsigned 16 bit output keypoint count and unsigned 8 bit descriptor

Returns

Error code (zero on success).

11.3.4.3 int apex::Brief::reconnectIO (const ICP_ContigDataDesc & *src*, const ICP_ContigDataDesc & *keypointIn*, const ICP_ContigDataDesc & *keypointInCnt*, ICP_ContigDataDesc & *keypointOut*, ICP_ContigDataDesc & *keypointOutCnt*, ICP_ContigDataDesc & *descriptor*)

Connects the input/outputs to the process.

Use this to reconnect the input and output buffers. This only needs to be done if the connected Input/Outputs are changed. If only the data within (no size, or type changes), then this does not need to be called.

Parameters

src	Source memory buffer. Datatype is ICP_DATATYPE_08U.
keypointIn	Input keypoint (x,y) data. Datatype is ICP_DATATYPE_16S. Data has to be stored in a chunk with size 32∗X, where X can be 1 thru 31.
keypointInCnt	Input keypoint count. Datatype is ICP_DATATYPE_16U.
keypointOut	Output keypoint (x,y) data. Datatype is ICP_DATATYPE_16S.
keypointOutCnt	Output keypoint count. Datatype is ICP_DATATYPE_16U.
descriptor	Destination memory buffer. Datatype is ICP_DATATYPE_08U.

11.3.4.4 int apex::Brief::setDescriptorSize (int *descriptorSize*)

Set descriptor size.

Set output descriptor size. Size should be either 16, 32 or 64.

Parameters

descriptorSize	BRIEF descriptor size, either 16, 32 or 64.

11.4 apex::ColorConverter Class Reference

Color converter class containing support for converting image color types.

`#include <ApexCV_base.hpp>`

Public Types

- enum ConversionType { RGB565_TO_RGB888, RGB888_TO_RGB565, RGB888_TO_Y, RGB888_TO_YUV }

  List of conversion types.

Public Member Functions

- ColorConverter ()

Default constructor.

- ~ColorConverter ()

  Destructor.

- int convert (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst, ConversionType ct, int R2YFactor=0, int G2YFactor=0, int B2YFactor=0)

  Convert function.

11.4.1 Detailed Description

Color converter class containing support for converting image color types.

This class is an interface for using color conversion functions on the host.

11.4.2 Member Enumeration Documentation

11.4.2.1 enum apex::ColorConverter::ConversionType

List of conversion types.

Enumerator

RGB565_TO_RGB888 16 bit RGB565 to 32 bit representation of RGB888X

RGB888_TO_RGB565 32 bit representation of RGB888X to 16 bit RGB565

RGB888_TO_Y 4-tuple 8 bit R, G, B, X to 8 bit Y

RGB888_TO_YUV 4-tuple 8 bit R, G, B, X to 4-tuple 8 bit Y, U, V, X

11.4.3 Constructor & Destructor Documentation

11.4.3.1 apex::ColorConverter::ColorConverter ( )

Default constructor.

11.4.3.2 apex::ColorConverter::~ColorConverter ( )

Destructor.

11.4.4 Member Function Documentation

11.4.4.1 int apex::ColorConverter::convert (const ICP_ContigDataDesc & *src*, ICP_ContigDataDesc & *dst*, ConversionType *ct*, int *R2YFactor* = 0, int *G2YFactor* = 0, int *B2YFactor* = 0)

Convert function.

Converts an image from one type to another based on ConversionType.

R2YFactor, G2YFactor and B2YFactor are Q0.8 fixed point values used with RGB888_TO_Y following the formula:

$$Y = \left\lfloor \frac{R2YFactor}{256} * R + \frac{G2YFactor}{256} * G + \frac{B2YFactor}{256} * B + 0.5 \right\rfloor$$

For example, conversion following Recommendation ITU-R BT.601-7 (http://www.itu.int/rec/R-REC-B-T.601-7-201103-I/en) would use factor values of 77(0.299), 150(0.587) and 29(0.114).

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]
ct	Color conversion type. See ConversionType
R2YFactor	Conversion factor for red used with RGB888_TO_Y
G2YFactor	Conversion factor for green used with RGB888_TO_Y
B2YFactor	Conversion factor for blue used with RGB888_TO_Y

11.5 apex::HarrisCornerDetector Class Reference

Apex Harris Corner Detector.

`#include <ApexCV_harris.hpp>`

Public Member Functions

- HarrisCornerDetector ()

  Default constructor. Allocate resources for the ACF processes.
- ~HarrisCornerDetector ()

  Destructor. Release resources for the ACF processes.
- int initialize (const ICP_ContigDataDesc &src, const ICP_ContigDataDesc &mask, const ICP_ContigDataDesc &thresh, ICP_ContigDataDesc &dst, int boxSize=3, int sobelSize=3, int cornerCoef=0)

  Initialize the corner detector.
- int reconnectIO (const ICP_ContigDataDesc &src, const ICP_ContigDataDesc &mask, const ICP_ContigDataDesc &thresh, ICP_ContigDataDesc &dst)

  Reconnect the Input and Output Buffers.
- int setSobelSize (int size)

  Set Sobel filter Size.
- int setBoxSize (int size)

  Set Box filter Size.
- int setCornerCoefficient (int cornerCoef)

  Set the corner coefficient.
- int setParameters (int boxSize, int sobelSize, int cornerCoef)

  Set Box filter, Sobel filter, and corner coefficient.
- int process ()

  Get the Harris corner scores.
- int retBlockWidth ()

  Returns Block Width.
- int retBlockHeight ()

  Returns Block Height.
- int sortCorners (const ICP_ContigDataDesc &src, ICP_FeatureDesc &corners, const int minDist, const int maxCorners)

  Sort the Harris corner scores.

11.5.1 Detailed Description

Apex Harris Corner Detector.

apex::HarrisCornerDetector is the host-ACF interface for creating, initializing, executing and releasing the Harris Corner Detector on Apex.

11.5.2 Constructor & Destructor Documentation

11.5.2.1 apex::HarrisCornerDetector::HarrisCornerDetector ( )

Default constructor. Allocate resources for the ACF processes.

11.5.2.2 apex::HarrisCornerDetector::~HarrisCornerDetector ( )

Destructor. Release resources for the ACF processes.

11.5.3 Member Function Documentation

11.5.3.1 int apex::HarrisCornerDetector::initialize (const ICP_ContigDataDesc & *src*, const ICP_ContigDataDesc & *mask*, const ICP_ContigDataDesc & *thresh*, ICP_ContigDataDesc & *dst*, int *boxSize* = 3, int *sobelSize* = 3, int *cornerCoef* = 0)

Initialize the corner detector.

Specify the Box and Sobel filter sizes and the corner coefficient used in getScores. The available filter sizes are 3 and 5 for both filters.

Parameters

src	8-bit grayscale source image.
mask	8-bit source mask image. 0xFF turns on a pixel, anything else turns it off
thresh	A 16-bit signed integer value used for the minimum score threshold
dst	16-bit signed integer corner score destination image.
boxSize	Box filter size. Size 3 and 5 are supported.
sobelSize	Sobel filter size. Size 3 and 5 are supported.
cornerCoef	Corner coefficient. Range 0-255. (*k* in [3]).

11.5.3.2 int apex::HarrisCornerDetector::process ( )

Get the Harris corner scores.

Returns

Error code for the ACF execution (zero on success).

For each pixel the corner score (referred to as "corner response" in [3]) is computed. The output is an image of 16-bit signed integer corner scores.

11.5.3.3 int apex::HarrisCornerDetector::reconnectIO (const ICP_ContigDataDesc & *src*, const ICP_ContigDataDesc & *mask*, const ICP_ContigDataDesc & *thresh*, ICP_ContigDataDesc & *dst*)

Reconnect the Input and Output Buffers.

Use this to reconnect the input and output buffers. This only needs to be done if the connected Input/Outputs are changed. If only the data within (no size, or type changes), then this does not need to be called.

Parameters

src	8-bit grayscale source image.
mask	8-bit source mask image. 0xFF turns on a pixel, anything else turns it off
thresh	A 16-bit signed integer value used for the minimum score threshold
dst	16-bit signed integer corner score destination image.

11.5.3.4 int apex::HarrisCornerDetector::retBlockHeight ( )

Returns Block Height.

Returns the block width used in the process.

11.5.3.5 int apex::HarrisCornerDetector::retBlockWidth ( )

Returns Block Width.

Returns the block width used in the process

11.5.3.6 int apex::HarrisCornerDetector::setBoxSize (int *size*)

Set Box filter Size.

Change the Box filter size used. Valid values are either 3 or 5.

Parameters

size	Box filter size. Size 3 and 5 are supported.

11.5.3.7 int apex::HarrisCornerDetector::setCornerCoefficient (int *cornerCoef*)

Set the corner coefficient.

Change the corner coefficient

Parameters

cornerCoef	Corner coefficientRange 0-255.

11.5.3.8 int apex::HarrisCornerDetector::setParameters (int *boxSize,* int *sobelSize,* int *cornerCoef*)

Set Box filter, Sobel filter, and corner coefficient.

Change the Box filter size (3 or 5), Sobel filter size (3 or 5), and the corner coefficient.

Parameters

boxSize	Box filter siz. Size 3 and 5 are supported.
sobelSize	Sobel filter size. Size 3 and 5 are supported.
cornerCoef	Corner coefficient. Range 0-255.

11.5.3.9 int apex::HarrisCornerDetector::setSobelSize (int *size*)

Set Sobel filter Size.

Change the Sobel filter size used. Valid values are either 3 or 5.

Parameters

size	Sobel filter size. Size 3 and 5 are supported.

11.5.3.10 int apex::HarrisCornerDetector::sortCorners (const ICP_ContigDataDesc & *src,* ICP_FeatureDesc & *corners,* const int *minDist,* const int *maxCorners*)

Sort the Harris corner scores.

Returns

Error code for the ACF initialization and execution(zero on success).

Sort the Harris corner scores from largest to smallest, choose the top *maxCorners*, and remove corners that are within a *minDist* of each other. The output is a list of sorted corners.

Parameters

src	Source corner score buffer.
corners	Destination corner feature description buffer.
minDist	The minimum distance between corners.
maxCorners	Maximum number of corners to return.

11.6 apex::Histogram Class Reference

Histogram class containing histogram support.

`#include <ApexCV_base.hpp>`

Public Member Functions

- Histogram ()

  Default constructor.
- ~Histogram ()

  Destructor.
- int exec (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Histogram execution.

11.6.1 Detailed Description

Histogram class containing histogram support.

This class is an interface for using the histogram kernel on the host.

11.6.2 Constructor & Destructor Documentation

11.6.2.1 apex::Histogram::Histogram ( )

Default constructor.

11.6.2.2 apex::Histogram::~Histogram ( )

Destructor.

11.6.3 Member Function Documentation

11.6.3.1 int apex::Histogram::exec (const ICP_ContigDataDesc & *src*, ICP_ContigDataDesc & *dst*)

Histogram execution.

Calculates an image histogram (http://en.wikipedia.org/wiki/Image_histogram) from the *src* buffer and returns the result in *dst* buffer. The resulting histogram has 256 bins each representing the respective number of pixel values found in the *src* image.

Supported datatypes are:

- unsigned 8 bit to unsigned 32 bit bins

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.7 apex::HoughLineDetector Class Reference

Apex Hough Line Detector.

`#include <ApexCV_hough.hpp>`

Classes

- struct Line

  Line data structure associated with the Hough Line Detector.

Public Types

- enum NonMaxSupFlag { NMS_NONE = 0, NMS_RHO = 1, NMS_THETA = 2 }

  Non-maxima suppression flag. To disable non-maxima suppression (nms), use NMS_NONE. To use nms on rho only, use NMS_RHO. To use nms on theta only, use NMS_THETA. To use nms on both, use NMS_RHO | NMS_THETA.
- typedef uint32_t PackedLine

  Packed line format for the Hough Line Detector.

Public Member Functions

- HoughLineDetector ()

  Default constructor. Allocate resources for the Hough transform ACF process.
- ~HoughLineDetector ()

  Destructor. Release resources for the Hough transform ACF process.
- int initialize (const ICP_ContigDataDesc &image, int pixelThreshold=127, int accumThreshold=100, int thetaCount=180, float *theta=0, int nonMaxSuppressionFlag=(NMS_RHO|NMS_THETA))

  Initialize parameters and allocate resources for the Hough transform ACF process.
- int initialize (const ICP_ContigDataDesc &image, int pixelThreshold, int accumThreshold, int thetaCount, double thetaStart, double thetaStep, int nonMaxSuppressionFlag=(NMS_RHO|NMS_THETA))

  Initialize parameters and allocate resources for the Hough transform ACF process.
- int reconnectIO (const ICP_ContigDataDesc &image, int pixelThreshold, int accumThreshold, int thetaCount, float *theta, int nonMaxSuppressionFlag=(NMS_RHO|NMS_THETA))

  Reconnect IO.
- int reconnectIO (const ICP_ContigDataDesc &image, int pixelThreshold, int accumThreshold, int thetaCount, double thetaStart, double thetaStep, int nonMaxSuppressionFlag=(NMS_RHO|NMS_THETA))

  Reconnect IO.
- int release ()

  Release resources and resets parameters for the Hough transform ACF process.
- int setPixelThreshold (int pixelThreshold)

  Set the pixel threshold for line detection. Only pixels greater than this value are used in the Hough transform.
- int setAccumThreshold (int accumThreshold)

  Set the Hough accumulator threshold for line detection.
- int setTheta (int thetaCount, float *theta=0, int nonMaxSuppressionFlag=(NMS_RHO|NMS_THETA))

  Specify the number of angles to be detected and their floating point values.
- int setTheta (int thetaCount, double thetaStart, double thetaStep, int nonMaxSuppressionFlag=(NMS_RHO|NMS_THETA))

  Specify the number of angles to be detected and their floating point values.
- int process ()

  Detect lines on the image.
- int rhoCount () const

  Get the number of rho values.
- int rhoStart () const

  Get the starting rho value for the Hough accumulator.
- int thetaCount () const

  Get the number of angles to detect.
- const float * thetaData () const

  Get a pointer to the angles to detect.
- int nonMaximaSuppressionFlag () const

Get the non-maxima suppression flag.

- int lineCount () const

  Get the number of detected lines.

- const PackedLine * packedLineData () const

  Get a pointer to the packed line data.

- Line line (int index) const

  Get the line coordinates for a detected line.

- Line line (PackedLine packedLine) const

  Get the line coordinates for a detected line.

Static Public Member Functions

- static int checkParameters (int imageCols, int imageRows, int pixelThreshold, int accumThreshold, int thetaCount)

  Check the validity of Hough initialization parameters.

- static int accumulator (PackedLine line)

  Get the Hough accumulator value from the PackedLine.

- static int rhoID (PackedLine line)

  Get the rho index from the PackedLine.

- static int thetaID (PackedLine line)

  Get the angle index from the PackedLine.

Static Public Attributes

- static const int cuCount

  The number of computation units (i.e. the number of angles computed in parallel).

- static const int maxRhoCount

  Maximum range of rho values.

- static const int maxLinesPerIteration

  Maximum number of detected lines per ACF process iteration.

- static const double deg2rad

  Conversion factor from degrees to radians.

- static const double rad2deg

  Conversion factor from radians to degrees.

- static const double pi

  The ratio of a circle's circumference to its diameter.

11.7.1 Detailed Description

Apex Hough Line Detector.

apex::HoughLineDetector is the host-ACF interface for creating, initializing, executing and releasing the Hough Line Detector on Apex.

11.7.2 Member Typedef Documentation

11.7.2.1 typedef uint32_t apex::HoughLineDetector::PackedLine

Packed line format for the Hough Line Detector.

Detected lines are stored in 32 bits. The first 8 bits are the angle index. The next 12 bits are the rho index. The last 12 bits are the Hough accumulator value.

11.7.3 Member Enumeration Documentation

11.7.3.1 enum apex::HoughLineDetector::NonMaxSupFlag

Non-maxima suppression flag. To disable non-maxima suppression (nms), use NMS_NONE. To use nms on rho only, use NMS_RHO. To use nms on theta only, use NMS_THETA. To use nms on both, use NMS_RHO | NMS_THETA.

Enumerator

NMS_NONE No non-maxima suppression.

NMS_RHO Non-maxima suppression on rho. Since the rho step is 1 pixel, this flag should always be used.

NMS_THETA Non-maxima suppression on theta. This flag should be used when the angle resolution small.

11.7.4 Constructor & Destructor Documentation

11.7.4.1 apex::HoughLineDetector::HoughLineDetector ( )

Default constructor. Allocate resources for the Hough transform ACF process.

11.7.4.2 apex::HoughLineDetector::~HoughLineDetector ( )

Destructor. Release resources for the Hough transform ACF process.

11.7.5 Member Function Documentation

11.7.5.1 static int apex::HoughLineDetector::accumulator (PackedLine *line*) [static]

Get the Hough accumulator value from the PackedLine.

Returns

The Hough accumulator value. The accumulator value is the number of pixels collinear to a given line.

11.7.5.2 static int apex::HoughLineDetector::checkParameters (int *imageCols*, int *imageRows*, int *pixelThreshold*, int *accumThreshold*, int *thetaCount*) [static]

Check the validity of Hough initialization parameters.

Returns

Error code for initialization (zero on success).

Parameters

imageCols	Image columns.
imageRows	Image rows.
pixelThreshold	Threshold to qualify a pixel for the Hough accumulator.
accumThreshold	Hough accumulator threshold. This is the minimum number of collinear pixels that form a line.
thetaCount	Number of angles to detect.

11.7.5.3 int apex::HoughLineDetector::initialize (const ICP_ContigDataDesc & *image*, int *pixelThreshold* = 127, int *accumThreshold* = 100, int *thetaCount* = 180, float * *theta* = 0, int *nonMaxSuppressionFlag* = (NMS_RHO|NMS_THETA))

Initialize parameters and allocate resources for the Hough transform ACF process.

Returns

Error code for initialization (zero on success).

Parameters

image	Image input.
pixelThreshold	Threshold to qualify a pixel for the Hough accumulator.
accumThreshold	Hough accumulator threshold. This is the minimum number of collinear pixels that form a line.
thetaCount	Number of angles to detect. The maximum supported value is 256.
theta	Angles to detect. If the pointer is null, the full range of angles is equally partitioned by thetaCount.
nonMax-SuppressionFlag	Non-maxima suppression flag.

11.7.5.4 int apex::HoughLineDetector::initialize (const ICP_ContigDataDesc & *image*, int *pixelThreshold*, int *accumThreshold*, int *thetaCount*, double *thetaStart*, double *thetaStep*, int *nonMaxSuppressionFlag* = (NMS_RHO|NMS_THETA))

Initialize parameters and allocate resources for the Hough transform ACF process.

Returns

Error code for initialization (zero on success).

Parameters

image	Image input.
pixelThreshold	Threshold to qualify a pixel for the Hough accumulator.
accumThreshold	Hough accumulator threshold. This is the minimum number of collinear pixels that form a line.
thetaCount	Number of angles to detect. The resulting angle resolution is pi/thetaCount.
thetaStart	Starting angle to be detected. The maximum supported value is 256.
thetaStep	Step between angles (angle resolution).
nonMax-SuppressionFlag	Non-maxima suppression flag.

11.7.5.5 Line apex::HoughLineDetector::line (int *index*) const

Get the line coordinates for a detected line.

Returns

The line coordinates for a detected line.

Parameters

index	Index of the detected line. This value must be within [0, lineCount].

11.7.5.6 Line apex::HoughLineDetector::line (PackedLine *packedLine*) const

Get the line coordinates for a detected line.

Returns

The line coordinates for a detected line.

Parameters

packedLine	The packed line data of the detected line.

11.7.5.7 int apex::HoughLineDetector::lineCount ( ) const

Get the number of detected lines.

Returns

The number of detected lines.

11.7.5.8 int apex::HoughLineDetector::nonMaximaSuppressionFlag ( ) const

Get the non-maxima suppression flag.

Returns

The non-maxima suppression flag.

11.7.5.9 const PackedLine* apex::HoughLineDetector::packedLineData ( ) const

Get a pointer to the packed line data.

Returns

A pointer to the packed line data.

11.7.5.10 int apex::HoughLineDetector::process ( )

Detect lines on the image.

Returns

Error code for the ACF execution (zero on success).

Lines are detected based on the parameters specified by initialize, setPixelThreshold, setAccumThreshold and setTheta.

The number of detected lines is accessed by lineCount. The packed line data is accessed by packedLineData. Line coordinates are accessed by line .

Returns

Error code for detection (zero on success).

11.7.5.11 int apex::HoughLineDetector::reconnectIO (const ICP_ContigDataDesc & *image,* int *pixelThreshold,* int *accumThreshold,* int *thetaCount,* float * *theta,* int *nonMaxSuppressionFlag* = (NMS_RHO|NMS_THETA))

Reconnect IO.

Reconnects the input and outputs. This only needs to be done if the connected Input/Outputs are changed. If only the data within (no size, or type changes), then this does not need to be called.

Parameters

image	Image input.
pixelThreshold	Threshold to qualify a pixel for the Hough accumulator.
accumThreshold	Hough accumulator threshold. This is the minimum number of collinear pixels that form a line.
thetaCount	Number of angles to detect.
theta	Angles to detect.
nonMax-SuppressionFlag	Non-maxima suppression flag.

11.7.5.12 int apex::HoughLineDetector::reconnectIO (const ICP_ContigDataDesc & *image,* int *pixelThreshold,* int *accumThreshold,* int *thetaCount,* double *thetaStart,* double *thetaStep,* int *nonMaxSuppressionFlag* = (NMS_RHO|NMS_THETA))

Reconnect IO.

Reconnects the input and outputs. This only needs to be done if the connected Input/Outputs are changed. If only the data within (no size, or type changes), then this does not need to be called.

Parameters

image	Image input.
pixelThreshold	Threshold to qualify a pixel for the Hough accumulator.
accumThreshold	Hough accumulator threshold. This is the minimum number of collinear pixels that form a line.
thetaCount	Number of angles to detect. The resulting angle resolution is pi/thetaCount.
thetaStart	Starting angle to be detected. The maximum supported value is 256.
thetaStep	Step between angles (angle resolution).
nonMax-SuppressionFlag	Non-maxima suppression flag.

11.7.5.13 int apex::HoughLineDetector::release ( )

Release resources and resets parameters for the Hough transform ACF process.

Returns

Error code for release (zero on success).

11.7.5.14 int apex::HoughLineDetector::rhoCount ( ) const

Get the number of rho values.

This is the size of the Hough accumulator in CMEM. It is given by $\text{round}\left(\sqrt{w^2+h^2}\right)+1$, where (w,h) is the image width and height.

Returns

The number of rho values.

11.7.5.15 static int apex::HoughLineDetector::rhoID (PackedLine *line*) [static]

Get the rho index from the PackedLine.

Returns

The rho index. The rho index $r >= 0$ is the index of the Hough transform for the line.

11.7.5.16 int apex::HoughLineDetector::rhoStart ( ) const

Get the starting rho value for the Hough accumulator.

This is the offset that must be applied to the rho index to obtain the true rho value. That is, rho = rhoID + rhoStart.

Returns

The starting rho value for the Hough accumulator.

11.7.5.17 int apex::HoughLineDetector::setAccumThreshold (int *accumThreshold*)

Set the Hough accumulator threshold for line detection.

Returns

Error code for initialization (zero on success).

Parameters

accumThreshold	Hough accumulator threshold. This is the minimum number of collinear pixels needed for line detection.

11.7.5.18 int apex::HoughLineDetector::setPixelThreshold (int *pixelThreshold*)

Set the pixel threshold for line detection. Only pixels greater than this value are used in the Hough transform.

Returns

Error code for initialization (zero on success).

Parameters

pixelThreshold	Threshold to qualify a pixel for the Hough accumulator.

11.7.5.19 int apex::HoughLineDetector::setTheta (int *thetaCount,* float * *theta* = 0, int *nonMaxSuppressionFlag* = (**NMS_RHO**|**NMS_THETA**))

Specify the number of angles to be detected and their floating point values.

Returns

Error code for initialization (zero on success).

Parameters

thetaCount	Number of angles to detect. The maximum supported value is 256.
theta	Angles to detect. If the pointer is null, the full range of angles is equally partitioned by thetaCount.
nonMax-SuppressionFlag	Non-maxima suppression flag.

11.7.5.20 int apex::HoughLineDetector::setTheta (int *thetaCount,* double *thetaStart,* double *thetaStep,* int *nonMaxSuppressionFlag* = (**NMS_RHO**|**NMS_THETA**))

Specify the number of angles to be detected and their floating point values.

Returns

Error code for initialization (zero on success).

Parameters

thetaCount	Number of angles to detect. The maximum supported value is 256.
thetaStart	Starting angle.
thetaStep	Step between successive angles.
nonMax-SuppressionFlag	Non-maxima suppression flag.

11.7.5.21 int apex::HoughLineDetector::thetaCount ( ) const

Get the number of angles to detect.

Returns

The number of angles to detect.

11.7.5.22 const float* apex::HoughLineDetector::thetaData ( ) const

Get a pointer to the angles to detect.

Returns

A pointer to the angles to detect.

11.7.5.23 static int apex::HoughLineDetector::thetaID (PackedLine *line*) [static]

Get the angle index from the PackedLine.

Returns

The angle index. The angle index $i \in [0, 255]$ corresponds to i^{th} angle specified by setTheta.

11.8 ICP_Feature Struct Reference

ICP Feature structure. This structure is used by the ICP_FeatureDesc class to store the position (x and y), and the strength of a feature.

`#include <ICP_FeatureDesc.hpp>`

Public Attributes

- ICP_Point_16S position
- int16_t strength
- int16_t reserve

11.8.1 Detailed Description

ICP Feature structure. This structure is used by the ICP_FeatureDesc class to store the position (x and y), and the strength of a feature.

11.8.2 Member Data Documentation

11.8.2.1 ICP_Point_16S ICP_Feature::position

The x and y location of the feature.

11.8.2.2 int16_t ICP_Feature::reserve

Used for memory alignment.

11.8.2.3 int16_t ICP_Feature::strength

The strength of the feature.

11.9 ICP_FeatureDesc Class Reference

ICP Feature Descriptor.

`#include <ICP_FeatureDesc.hpp>`

Public Member Functions

- ICP_FeatureDesc ()

  Default constructor.
- ICP_FeatureDesc (void ∗const lpData, void ∗const lpDataPhys, int32_t maxElements)

  Constructor.
- void ∗ RetDataPtr ()

  Return Data Pointer.
- void ∗ RetDataPtrPhys ()

  Returns the 'physical' Data Pointer.
- int32_t RetSpan ()

  Returns the span of a single feature.
- int32_t RetSize ()

  Returns the maximum number of features.
- int32_t RetCount ()

  Return the number of features available.
- int32_t SetCount (int32_t count)

  Set the number of features available.
- int32_t Add (int16_t x, int16_t y, int16_t strength=0)

  Add a feature.
- int32_t Remove (int32_t ind)

  Remove a feature at an index.
- ICP_Feature & GetFeature (int32_t ind)

  Get a feature at a specified index.
- int32_t Set (int32_t ind, int16_t x, int16_t y, int16_t strength=0)

  Set a feature.
- ICP_Feature & operator[] (int32_t ind)

  Operator [].
- void Init (void ∗const lpData, void ∗const lpDataPhys, int32_t maxElements)

  Initialize the descriptor.

11.9.1 Detailed Description

ICP Feature Descriptor.

ICP_FeatureDesc is a container class designed to encapsulate a contiguous region of data of type ICP_Feature. It does not allocate or deallocate memory; it simply standardizes the representation of a contiguous memory region so it can be used by framework level services. ICP_Feature is a structure that contains the position of a feature and its strength.

The memory must be allocated using OAL for the number of features you want to store. i.e. If you want to be able to store 30 features:

```
void *lBuffOal = OAL_MemoryAllocFlag(sizeof(ICP_Feature)*30,
                 OAL_MEMORY_FLAG_ALIGN(ALIGN2_CACHELINE)|OAL_MEMORY_FLAG_CONTIGUOUS);
ICP_FeatureDesc Desc(OAL_MemoryReturnAddress(lBuffOal, ACCESS_NCH_NB),
                     OAL_MemoryReturnAddress(lBuffOal, ACCESS_PHY),
                     30);
```

11.9.2 Constructor & Destructor Documentation

11.9.2.1 ICP_FeatureDesc::ICP_FeatureDesc ( )

Default constructor.

11.9.2.2 ICP_FeatureDesc::ICP_FeatureDesc (void ∗const *lpData,* void ∗const *lpDataPhys,* int32_t *maxElements*)

Constructor.

Constructor that initializes a contiguous data region. Note that the data region must be physically contiguous in memory (not just contiguous from the OS point of view).

Parameters

lpData	Pointer to contiguous data region.
lpDataPhys	Physical pointer to contiguous data region (for HW use).
maxElements	The maximum number of features.

11.9.3 Member Function Documentation

11.9.3.1 int32_t ICP_FeatureDesc::Add (int16_t *x,* int16_t *y,* int16_t *strength* = 0)

Add a feature.

Returns

The result of the operation (zero on success).

This adds a feature to the descriptor. This should only be used if the count of the number of features available is kept accurate. This will check the number of features against the maximum number of features, if it is not at full capacity, the new feature will be added and the count is incremented.

11.9.3.2 ICP_Feature& ICP_FeatureDesc::GetFeature (int32_t *ind*)

Get a feature at a specified index.

Returns

The ICP_Feature at the specified index

This will return a feature at index *ind*. If the index is greater than the maximum size of the descriptor, the feature at index 0 is returned.

11.9.3.3 void ICP_FeatureDesc::Init (void ∗const *lpData,* void ∗const *lpDataPhys,* int32_t *maxElements*)

Initialize the descriptor.

This will initialize the descriptor with a contiguous data region. Note that the data region must be physically contiguous in memory (not just contiguous from the OS point of view).

Parameters

lpData	Pointer to contiguous data region.
lpDataPhys	Physical pointer to contiguous data region (for HW use).
maxElements	The maximum number of features.

11.9.3.4 ICP_Feature& ICP_FeatureDesc::operator[] (int32_t *ind*)

Operator [].

Returns

ICP_Feature at index

This will return a feature at the specified index. If the index is greater than the maximum size of the descriptor, the feature at index 0 is returned.

11.9.3.5 int32_t ICP_FeatureDesc::Remove (int32_t *ind*)

Remove a feature at an index.

Returns

The result of the operation (zero on success).

This removes a feature from the descriptor at the specified index. The remaining features will be shifted to fill the space. This should only be used if the count is kept updated.

11.9.3.6 int32_t ICP_FeatureDesc::RetCount ( )

Return the number of features available.

Returns

The number of features currently stored

Returns the number of features currently stored in the descriptor. This value is incremented each time a feature is added with the ICP_FeatureDesc::Add function. If the features are manually added/removed, the count can be updated using ICP_FeatureDesc::SetCount. Only accurate if class functions are used to add/remove features, or if kept accurate by updating count.

11.9.3.7 void* ICP_FeatureDesc::RetDataPtr ( )

Return Data Pointer.

Returns

A void data pointer to the contiguous data region

Returns a void data pointer to the contiguous data region.

11.9.3.8 void* ICP_FeatureDesc::RetDataPtrPhys ( )

Returns the 'physical' Data Pointer.

Returns

A void 'physical' data pointer to the contiguous data region.

Returns a void 'physical' data pointer to the contiguous data region.

11.9.3.9 int32_t ICP_FeatureDesc::RetSize ( )

Returns the maximum number of features.

Returns

The number of features the descriptor can hold

Returns the maximum number of features the descriptor can hold.

11.9.3.10 int32_t ICP_FeatureDesc::RetSpan ( )

Returns the span of a single feature.

Returns

The span of a feature

Returns the number of bytes a single ICP_Feature occupies

11.9.3.11 int32_t ICP_FeatureDesc::Set (int32_t *ind,* int16_t *x,* int16_t *y,* int16_t *strength* = 0)

Set a feature.

Returns

The result of the operation (zero on success).

This will modify the feature at index *ind* to contain the specified position and strength. The strength will default to 0 if not specified.

11.9.3.12 int32_t ICP_FeatureDesc::SetCount (int32_t *count*)

Set the number of features available.

Returns

The result of the operation (zero on success).

This sets the number of features available in the descriptor. The count is limited to the range [0, Max Elements];

11.10 apex::ImageFilter Class Reference

Image filter class containing various image filters.

`#include <ApexCV_base.hpp>`

Public Types

- enum SobelType { SOBEL_X, SOBEL_Y, SOBEL_BOTH }

  List of sobel filter types.
- enum PrewittType { PREWITT_X, PREWITT_Y }

  List of prewitt filter types.

Public Member Functions

- ImageFilter ()

  Default constructor.
- ~ImageFilter ()

  Destructor.
- int bilateralFilter (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst, int sigmaColor, int sigmaSpace)

  Bilateral image filter.
- int boxFilter (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Box image filter.
- int dilateFilter (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Dilate image filter.
- int erodeFilter (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Erode image filter.
- int medianFilter (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst, int windowSize)

  Median filter.
- int gaussianFilter (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Gaussian image filter.
- int sobelFilter (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst, int windowSize, SobelType st=SOBEL_BOTH)

  Sobel filter.
- int convolveFilter (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst, const signed char *filterCoeff, int windowSize=3, int filterScale=0)

  Convolve filter.
- int prewittFilter (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst, PrewittType pt=PREWITT_X)

  Prewitt filter.

11.10.1 Detailed Description

Image filter class containing various image filters.

This class is an interface for using image filters on the host. Filter windows that exceed the boundaries of the image use replicated pixels for the padding.

11.10.2 Member Enumeration Documentation

11.10.2.1 enum apex::ImageFilter::PrewittType

List of prewitt filter types.

Enumerator

PREWITT_X Apply prewitt filter in x direction

PREWITT_Y Apply prewitt filter in y direction

11.10.2.2 enum apex::ImageFilter::SobelType

List of sobel filter types.

Enumerator

SOBEL_X Apply sobel filter in x direction

SOBEL_Y Apply sobel filter in y direction

SOBEL_BOTH Apply sobel filter in both x and y direction then sum their absolute values

11.10.3 Constructor & Destructor Documentation

11.10.3.1 apex::ImageFilter::ImageFilter ( )

Default constructor.

11.10.3.2 apex::ImageFilter::∼ImageFilter ( )

Destructor.

11.10.4 Member Function Documentation

11.10.4.1 int apex::ImageFilter::bilateralFilter (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst,* int *sigmaColor,* int *sigmaSpace*)

Bilateral image filter.

Applies a bilateral filter to *src*. *sigmaColor* represents the weight of color/intensity difference and *sigmaSpace* represents the weight of spacial difference. See: [4] for more information.

Supported window size is:

- 5 x 5

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]
sigmaColor	Sigma value for color space.
sigmaSpace	Sigma value for distance space.

11.10.4.2 int apex::ImageFilter::boxFilter (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst*)

Box image filter.

Applies a box filter to *src* buffer. Each *dst* buffer pixel is calculated as follows: $dst = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * src$

Supported window size is:

- 3 x 3

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit
- signed 16 bit to signed 16 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1].
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.10.4.3 int apex::ImageFilter::convolveFilter (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst,* const signed char * *filterCoeff,* int *windowSize* = 3, int *filterScale* = 0)

Convolve filter.

Applies a user defined filter with coefficients *filterCoeff* to *src* buffer and returns the result in *dst* buffer. *filterScale* can be used to divide each filtered pixel by a value of $2^{filterScale}$. If the resulting pixel would overflow the *dst* datatype its value is saturated.

Supported window sizes are:

- 3 x 3
- 5 x 5

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit
- unsigned 8 bit to signed 16 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]
filterCoeff	Array containing the filter coefficients to be applied.
windowSize	Defines a filter window with dimensions *windowSize* x *windowSize*. Default is 3. See supported window sizes.
filterScale	Optional: used to scale the resulting filtered pixel by $2^{filterScale}$. Supported range of [0, 16].

11.10.4.4 int apex::ImageFilter::dilateFilter (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst*)

Dilate image filter.

Applies a dilate filter to *src* buffer. Each *dst* buffer pixel is the maximum pixel value contained in the window of the respective *src* buffer pixel.

Supported window size is:

- 3 x 3

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit
- signed 16 bit to signed 16 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.10.4.5 int apex::ImageFilter::erodeFilter (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst*)

Erode image filter.

Applies an erode filter to *src* buffer. Each *dst* buffer pixel is the minimum pixel value contained in the window of the respective *src* buffer pixel.

Supported window size is:

- 3 x 3

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.10.4.6 int apex::ImageFilter::gaussianFilter (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst*)

Gaussian image filter.

Applies a gaussian filter to *src* buffer. Each *dst* buffer pixel is calculated as follows: $dst = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * src$

Supported window size is:

- 3 x 3

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1].
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.10.4.7 int apex::ImageFilter::medianFilter (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst,* int *windowSize*)

Median filter.

Applies a median filter to *src* buffer. Each *dst* buffer pixel is the median pixel value contained in the window of the respective *src* buffer pixel.

Supported window sizes are:

- 3 x 3
- 5 x 5

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]
windowSize	Defines a filter window with dimensions *windowSize* x *windowSize*. Default is 3. See supported window sizes.

11.10.4.8 int apex::ImageFilter::prewittFilter (const ICP_ContigDataDesc & *src*, ICP_ContigDataDesc & *dst*, PrewittType *pt* = PREWITT_X)

Prewitt filter.

Applies a prewitt filter based on PrewittType to *src* buffer and returns the result in *dst* buffer. Similar to the sobelFilter, prewitt uses the following filter coefficients:

$$filter_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, filter_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$$

Supported formats are:

- x direction
- y direction

Supported window size is:

- 3 x 3

Supported datatypes are:

- unsigned 8 bit to signed 16 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]
pt	Specify which prewitt filter type to use. Default is PREWITT_X. See PrewittType.

11.10.4.9 int apex::ImageFilter::sobelFilter (const ICP_ContigDataDesc & *src*, ICP_ContigDataDesc & *dst*, int *windowSize*, SobelType *st* = SOBEL_BOTH)

Sobel filter.

Applies a sobel filter based on SobelType to *src* buffer and returns the result in *dst* buffer. SOBEL_BOTH calculates both SOBEL_X and SOBEL_Y images and then sums their absolute values following the formula: $dst(i,j) = |SOBEL_X(i,j)| + |SOBEL_Y(i,j)|$

The 3x3 filter coefficients for x and y directions are:

$$filter_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, filter_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

The 5x5 filter coefficients for x and y directions are:

$$filter_x = \begin{bmatrix} -1 & -2 & 0 & +2 & +1 \\ -4 & -8 & 0 & +8 & +4 \\ -6 & -12 & 0 & +12 & +6 \\ -4 & -8 & 0 & +8 & +4 \\ -1 & -2 & 0 & +2 & +1 \end{bmatrix}, filter_y = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ +2 & +8 & +12 & +8 & +2 \\ +1 & +4 & +6 & +4 & +1 \end{bmatrix}$$

Supported formats are:

- x direction
- y direction
- both directions

Supported window sizes are:

- 3 x 3
- 5 x 5

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit
- unsigned 8 bit to signed 8 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]
windowSize	Defines a filter window with dimensions *windowSize* x *windowSize*. Default is 3. See supported window sizes.
st	Specify which sobel filter type to use. Default is SOBEL_BOTH. See SobelType.

11.11 apex::IntegralImage Class Reference

Integral Image class containing integral image support.

`#include <ApexCV_base.hpp>`

Public Member Functions

- IntegralImage ()

  Default constructor.

- ~IntegralImage ()

  Destructor.
- int exec (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Integral Image execution.

11.11.1 Detailed Description

Integral Image class containing integral image support.

This class is an interface for using the integral image kernel on the host.

11.11.2 Constructor & Destructor Documentation

11.11.2.1 apex::IntegralImage::IntegralImage ( )

Default constructor.

11.11.2.2 apex::IntegralImage::~IntegralImage ( )

Destructor.

11.11.3 Member Function Documentation

11.11.3.1 int apex::IntegralImage::exec (const ICP_ContigDataDesc & *src*, ICP_ContigDataDesc & *dst*)

Integral Image execution.

Calculates an integral image (http://en.wikipedia.org/wiki/Summed_area_table) from the *src* buffer and returns the result in *dst* buffer. Each *dst* pixel respresents the sum of all *src* pixels left of and above the *dst* pixel including the *src* pixel

Supported datatypes are:

- unsigned 8 bit to unsigned 32 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer. See ICP_ContigDataDesc [1]
dst	Destination memory buffer. See ICP_ContigDataDesc [1]

11.12 apex::Interpolation Class Reference

Image interpolation class containing various interpolation methods.

```
#include <ApexCV_base.hpp>
```

Public Member Functions

- Interpolation ()

  Default constructor.
- ∼Interpolation ()

  Destructor.
- int linearGrayscale (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst, ICP_ContigDataDesc &deltaX)

  Linear Grayscale interpolation.
- int bilinearGrayscale (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst, ICP_ContigDataDesc &delta)

  Bilinear Grayscale interpolation.

11.12.1 Detailed Description

Image interpolation class containing various interpolation methods.

This class is an interface for using image interpolations on the host. Interpolation windows that exceed the boundaries of the image use replicated pixels for the padding.

11.12.2 Constructor & Destructor Documentation

11.12.2.1 apex::Interpolation::Interpolation ( )

Default constructor.

11.12.2.2 apex::Interpolation::∼Interpolation ( )

Destructor.

11.12.3 Member Function Documentation

11.12.3.1 int apex::Interpolation::bilinearGrayscale (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst,* ICP_ContigDataDesc & *delta*)

Bilinear Grayscale interpolation.

Applies bilinear interpolation to *src*. *delta* represents the distance the resultant pixel was from the left adjacent pixels and from the top adjacent pixels <x, y>, normalized between 0-255.

Supported window size is:

- 2 x 2

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer.
dst	Destination memory buffer.
delta	Delta XY values for interpolation.

11.12.3.2 int apex::Interpolation::linearGrayscale (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst,* ICP_ContigDataDesc & *deltaX*)

Linear Grayscale interpolation.

Applies linear interpolation to *src*. *deltaX* represents the distance the resultant pixel was from the left adjacent pixel, normalized between 0-255.

Supported window size is:

- 2 x 1

Supported image size is:

- Width: multiple of 2
- Height: any

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

src	Source memory buffer.
dst	Destination memory buffer.
deltaX	Delta X values for interpolation.

11.13 apex::HoughLineDetector::Line Struct Reference

Line data structure associated with the Hough Line Detector.

`#include <ApexCV_hough.hpp>`

Public Member Functions

- Line (int rho_=0, float theta_=0.f)

  Default constructor.

Public Attributes

- int rho

  Nearest distance of the line to the center of the image.
- float theta

  Angle of the line's normal.

11.13.1 Detailed Description

Line data structure associated with the Hough Line Detector.

11.13.2 Constructor & Destructor Documentation

11.13.2.1 apex::HoughLineDetector::Line::Line (int *rho_* = `0`, float *theta_* = `0.f`) `[inline]`

Default constructor.

Parameters

*rho_*	Nearest distance of the line to the center of the image.
*theta_*	Angle of the line's normal.

11.14 apex::PyramidCreation Class Reference

Pyramid creation class.

`#include <ApexCV_pyramid.hpp>`

Public Member Functions

- PyramidCreation ()

  Default constructor.
- ~PyramidCreation ()

  Default destructor.
- int initialize (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Initializes the class.
- int reconnectIO (const ICP_ContigDataDesc &src, ICP_ContigDataDesc &dst)

  Connects the input/outputs to the process.
- int process (bool isPyramidUp)

  Run the pyramid creation process.

11.14.1 Detailed Description

Pyramid creation class.

This class is an interface for using the pyramid creation algorithm.

11.14.2 Constructor & Destructor Documentation

11.14.2.1 apex::PyramidCreation::PyramidCreation ( )

Default constructor.

11.14.2.2 apex::PyramidCreation::~PyramidCreation ( )

Default destructor.

11.14.3 Member Function Documentation

11.14.3.1 int apex::PyramidCreation::initialize (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst*)

Initializes the class.

Connects the buffers to the process.

Parameters

src	Source memory buffer. Datatype is ICP_DATATYPE_08U.
dst	Destination memory buffer. Datatype is ICP_DATATYPE_08U.

11.14.3.2 int apex::PyramidCreation::process (bool *isPyramidUp*)

Run the pyramid creation process.

Upscale or downscale *src* buffer and stores the result in *dst* buffer. The process will upscale *src* buffer if *isPyramidUp* = true. The process will downscale *src* buffer if *isPyramidUp* = false. Default is *isPyramidUp* = true.

Supported datatypes are:

- unsigned 8 bit to unsigned 8 bit

Returns

Error code (zero on success).

Parameters

isPyramidUp	Set true if execute pyramid up, or false if execute pyramid down.

11.14.3.3 int apex::PyramidCreation::reconnectIO (const ICP_ContigDataDesc & *src,* ICP_ContigDataDesc & *dst*)

Connects the input/outputs to the process.

Use this to reconnect the Input and Output Buffers. This only needs to be done if the connected Input/Outputs are changed. If only the data within (no size, or type changes), then this does not need to be called.

Parameters

src	Source memory buffer. Datatype is ICP_DATATYPE_08U.
dst	Destination memory buffer. Datatype is ICP_DATATYPE_08U.

11.15 apex::Remapper Class Reference

Apex Remapper.

`#include <ApexCV_remap.hpp>`

Public Member Functions

- Remapper ()

  Default constructor. Allocate resources for the ACF process.
- ~Remapper ()

  Destructor. Release resources for the ACF process.
- int initialize (ICP_ContigDataDesc &lSrcImage, ICP_ContigDataDesc &lDestImage, float *mapX, float *mapY, int32_t destBlockWidth, int32_t destBlockHeight, apex::INTER_TYPE interpolation, apex::BORDER_TYPE borderMode, int32_t borderValue)

  Initialize the remap ACF process.
- int release ()

  Release Resources.
- int reconnectIO (ICP_ContigDataDesc &lSrcImage, ICP_ContigDataDesc &lDestImage, float *mapX, float *mapY, int32_t destBlockWidth, int32_t destBlockHeight, apex::INTER_TYPE interpolation, apex::BORDER_TYPE borderMode, int32_t borderValue)

  Reconnect IO.
- int getBlockSize (float *mapX, float *mapY, int32_t mapWidth, int32_t mapHeight, int32_t destBlockWidth, int32_t destBlockHeight, apex::INTER_TYPE interpolation, int32_t srcStride, int32_t &srcBlockWidth, int32_t &srcBlockHeight)

  Calculate the required source block size.
- int process ()

  Execute the remap ACF process.
- int RetLUTs (ICP_ContigDataDesc &rlDeltaDesc, ICP_ContigDataDesc &rlLocalOffsetDesc, ICP_ContigDataDesc &rlBlockOffsetDesc)

  Returns the LUTs needed for the remap ACF process.

11.15.1 Detailed Description

Apex Remapper.

apex::Remapper is the host-ACF interface for creating, initializing, executing and releasing image remap on Apex.

11.15.2 Constructor & Destructor Documentation

11.15.2.1 apex::Remapper::Remapper ( )

Default constructor. Allocate resources for the ACF process.

11.15.2.2 apex::Remapper::∼Remapper ( )

Destructor. Release resources for the ACF process.

11.15.3 Member Function Documentation

11.15.3.1 int apex::Remapper::getBlockSize (float * *mapX*, float * *mapY*, int32_t *mapWidth*, int32_t *mapHeight*, int32_t *destBlockWidth*, int32_t *destBlockHeight*, apex::INTER_TYPE *interpolation*, int32_t *srcStride*, int32_t & *srcBlockWidth*, int32_t & *srcBlockHeight*)

Calculate the required source block size.

This method is used to determine the required source block side, given a map and destination block size. The internal state of the apex::Remapper class is not affected by this call.

The primary purpose of this method is to allow developers to choose a set of block sizes for a specific remap application.

Returns

Error code for the execution (zero on success).

Parameters

mapX	Input. Interleaved x/y float map.
mapY	Input. Reserved for future use.
mapWidth	Input. The width of the destination image. This is also the width of the map
mapHeight	Input. The height of the destination image. This is also the height of the map
destBlockWidth	Input. APU block size for destination image. Used in SetInputChunkSize() call for local offset and delta.
destBlockHeight	Input. APU block size for destination image. Used in SetInputChunkSize() call for local offset and delta.
interpolation	Input. Method of interpolation to be used.
srcStride	Input. Stride (width plus padding) of the source image the map is based on.
srcBlockWidth	Output. APU block size for source image. Used in metadata for remap kernel, source image ek size.
srcBlockHeight	Output. APU block size for source image. Used in metadata for remap kernel, source image ek size.

11.15.3.2 int apex::Remapper::initialize (ICP_ContigDataDesc & *ISrcImage*, ICP_ContigDataDesc & *IDestImage*, float * *mapX*, float * *mapY*, int32_t *destBlockWidth*, int32_t *destBlockHeight*, apex::INTER_TYPE *interpolation*, apex::BORDER_TYPE *borderMode*, int32_t *borderValue*)

Initialize the remap ACF process.

This method is used to initialize the remap processing. This processing will calculate the required source block size, and generate the internal offset and delta tables required for the APU implementation of resize. Note that the source block size is calculated based on the map and the destination block size. This source block size is required to match the ek value given in the ACF kernel metadata (currently 10x5 for RGB, and 20x5 for greyscale). This requirement will be revisited in future releases.

This method can be called once before the image processing loop. It shall be called before remap() is invoked.

Returns

Error code for the execution (zero on success).

Parameters

lSrcImage	Source memory buffer.
lDestImage	Destination memory buffer.
mapX	Floating point x map.
mapY	Floating point y map.
destBlockWidth	APU block size
destBlockHeight	APU block size
interpolation	Method of interpolation to use
borderMode	Method of border padding to use
borderValue	Value used for constant border

11.15.3.3 int apex::Remapper::process ( )

Execute the remap ACF process.

This method executes the image remap which was configured by initialize().

Returns

Error code for the execution (zero on success).

11.15.3.4 int apex::Remapper::reconnectIO (ICP_ContigDataDesc & *lSrcImage,* ICP_ContigDataDesc & *lDestImage,* float * *mapX,* float * *mapY,* int32_t *destBlockWidth,* int32_t *destBlockHeight,* apex::INTER_TYPE *interpolation,* apex::BORDER_TYPE *borderMode,* int32_t *borderValue*)

Reconnect IO.

Reconnects the input and outputs to the process. This only needs to be done if the connected Input/Outputs are changed. If only the data within (no size, or type changes), then this does not need to be called. This just calls release and then initialize.

Parameters

lSrcImage	Source memory buffer.
lDestImage	Destination memory buffer.
mapX	Floating point x map.
mapY	Floating point y map.
destBlockWidth	APU block size
destBlockHeight	APU block size
interpolation	Method of interpolation to use
borderMode	Method of border padding to use
borderValue	Value used for constant border

11.15.3.5 int apex::Remapper::release ( )

Release Resources.

Releases the internal buffers and resets the class state to initial.

11.15.3.6 int apex::Remapper::RetLUTs (ICP_ContigDataDesc & *rlDeltaDesc,* ICP_ContigDataDesc & *rlLocalOffsetDesc,* ICP_ContigDataDesc & *rlBlockOffsetDesc*)

Returns the LUTs needed for the remap ACF process.

This method returns the LUTs which was configured by initialize().

Returns

Error code for the execution (zero on success).

11.16 apex::Resize Class Reference

Apex Resize.

`#include <ApexCV_resize.hpp>`

Public Member Functions

- Resize ()
  Default constructor. Allocate resources for the ACF process.
- ∼Resize ()
  Destructor. Release resources for the ACF process.
- int initialize (ICP_ContigDataDesc &lSrcImage, ICP_ContigDataDesc &lDestImage)
  Initialize the resize ACF process.
- int reconnectIO (ICP_ContigDataDesc &lSrcImage, ICP_ContigDataDesc &lDestImage)
  Reconnect IO.
- int process ()
  Execute the resize ACF process".

11.16.1 Detailed Description

Apex Resize.

apex::Resize is the host-ACF interface for creating, initializing, executing and releasing image resize on Apex.

11.16.2 Constructor & Destructor Documentation

11.16.2.1 apex::Resize::Resize ( )

Default constructor. Allocate resources for the ACF process.

11.16.2.2 apex::Resize::∼Resize ( )

Destructor. Release resources for the ACF process.

11.16.3 Member Function Documentation

11.16.3.1 int apex::Resize::initialize (ICP_ContigDataDesc & *lSrcImage,* ICP_ContigDataDesc & *lDestImage*)

Initialize the resize ACF process.

Returns

Error code for the execution (zero on success).

Parameters

lSrcImage	Source memory buffer.
lDestImage	Destination memory buffer.

11.16.3.2 int apex::Resize::process ( )

Execute the resize ACF process".

Returns

Error code for the initialization (zero on success).

11.16.3.3 int apex::Resize::reconnectIO (ICP_ContigDataDesc & *lSrcImage,* ICP_ContigDataDesc & *lDestImage*)

Reconnect IO.

Reconnects the input and outputs to the process. This only needs to be done if the connected Input/Outputs are changed. If only the data within (no size, or type changes), then this does not need to be called. This just calls release and then initialize.

Parameters

lSrcImage	Source memory buffer.
lDestImage	Destination memory buffer.

Bibliography

[1] Acf.chm. Technical report, CogniVue, 2014. 17, 18, 19, 20, 21, 22, 23, 29, 33, 49, 50, 51, 52, 53, 54

[2] Richard O. Duda and Peter E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, January 1972. 8

[3] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988. 6, 30

[4] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846, Jan 1998. 48

Index