IKB 405 - Polarization in Spinnnaker

Applies to:

BFS-U3-51S5P-C (mono) & BFS-U3-51S5PC-C (color)

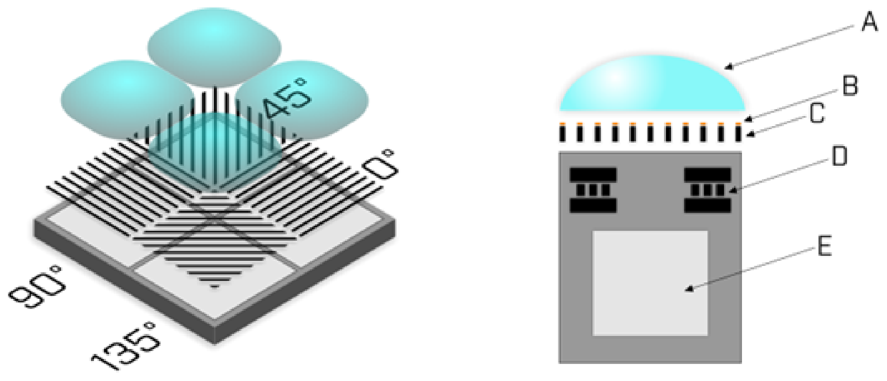
There are now Blackfly S models which feature Sony Polarsens on sensor polarization technology.

See here for information about Polarsens technology and some applications that it enables:

https://www.sony-semicon.co.jp/products_en/IS/sensor5/index.html

Here is a brief guide to getting started with BFS polarized cameras.

Mono Polarization Cameras



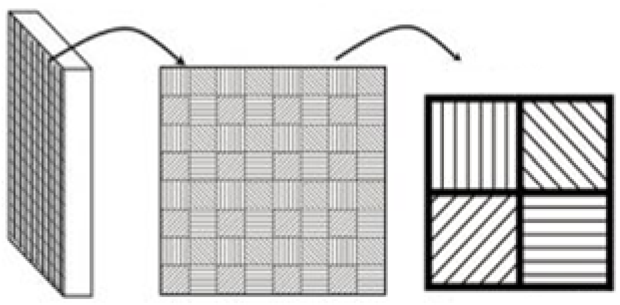
Each pixel's polarizing filter (C) is coated with an anti-reflective layer (B) and is positioned between the microlens (A) and the light sensitive photodiode (E).

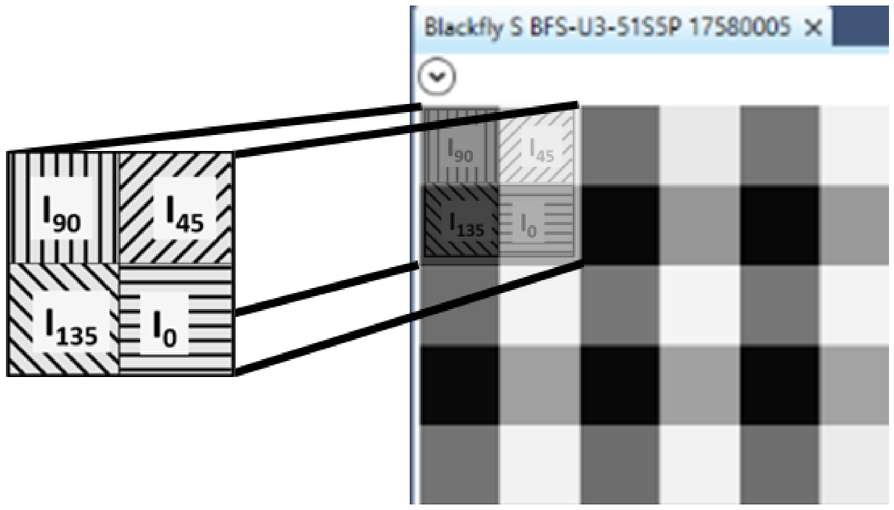
The physical layout of the polarized sensor is composed of an array of polarized quads. Each quad contains 4 pixels. Each of these four pixels has a wire-grid polarizing filter layer.

The four angles polarization are : 90°, 45°, 135° and 0°. In the image above the angle label refers to the transmission angle which is perpendicular to the wire grid.

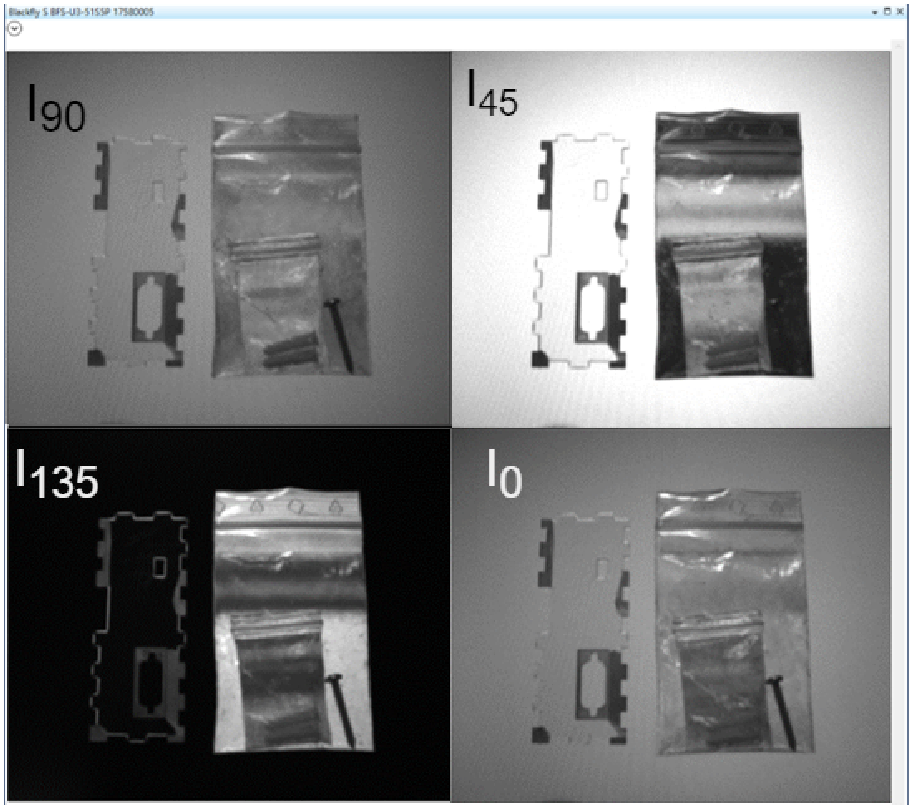
Note that:

- The **minimum** transmission of polarized light occurs **parallel** to the wire-grid axis.
- The **maximum** transmission of polarized light occurs **perpendicular** to the wire-grid axis.





In the above figure you can observe that when we zoom in to the pixel level of an image of partially polarized light we can see the effect of the polarizing filter.

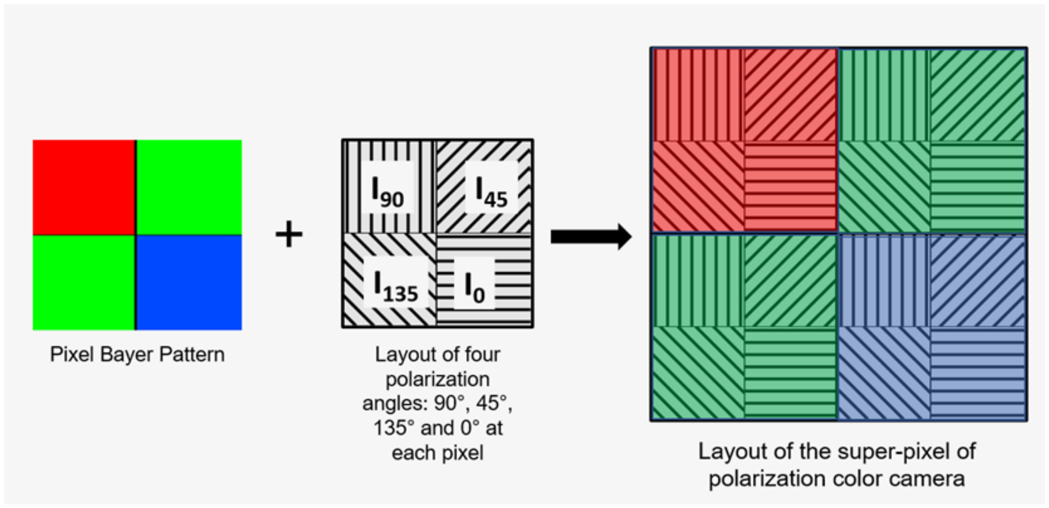


We can also extract and group all pixels from each of the four quadrants using the quad algorithm and combine them to create a composite image as shown above.

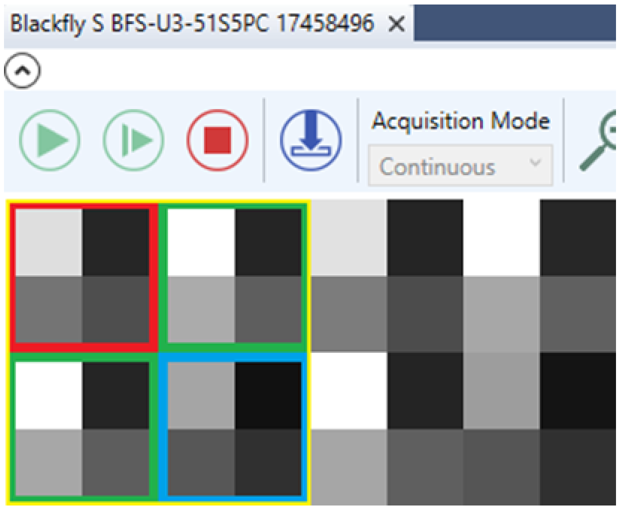
Color Polarization Cameras

Color polarization cameras also have an array of polarized quads but in addition they also have a Bayer filter.

Unlike traditional color cameras where the Bayer tile repeats every 2x2 pixels, color polarization Bayer tiles repeat every 4x4 pixels which creates a 4x4 "super pixel" as seen below.


Pixel Bayer Pattern
+
I_{90}
I_{45}
I_{135}
I_{0}
Layout of four polarization angles: 90°, 45°, 135° and 0° at each pixel
Layout of the super-pixel of polarization color camera

This "super pixel" is highlighted below in SpinView when the image display is zoomed in to the pixel level.



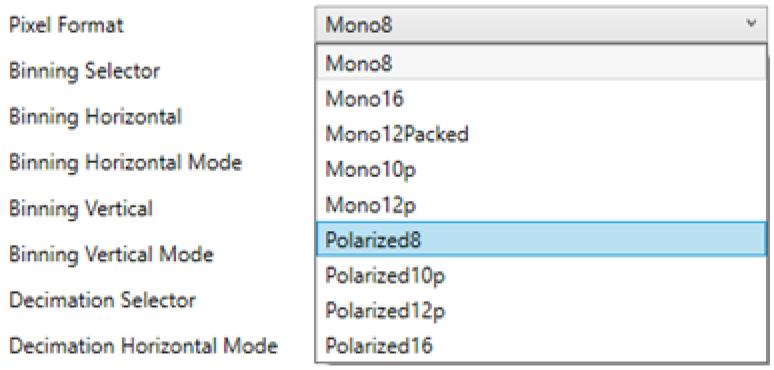
Polarization Algorithms Available in SpinView for Windows

Note: HeatMap processing is only applicable to monochrome cameras.

Selecting a Polarized Pixel Format

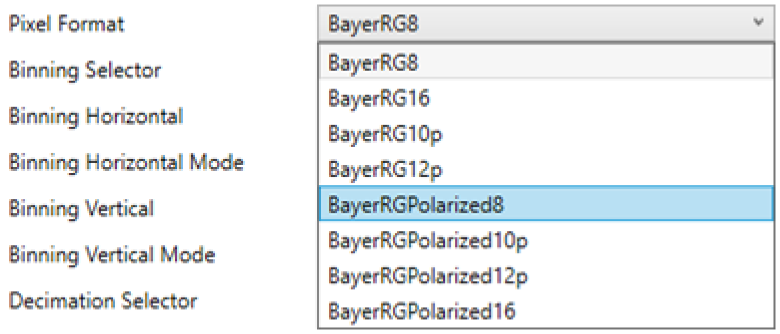
SpinView polarization features are only available for Polarized8 (for monochrome cameras) and BayerRGPolarized8 (on color cameras)

Mono



Pixel Format	Mono8
Binning Selector	
Binning Horizontal	
Binning Horizontal Mode	
Binning Vertical	
Binning Vertical Mode	
Decimation Selector	
Decimation Horizontal Mode	

Color


Pixel Format: BayerRG8

Binning Selector

Binning Horizontal

Binning Horizontal Mode

Binning Vertical

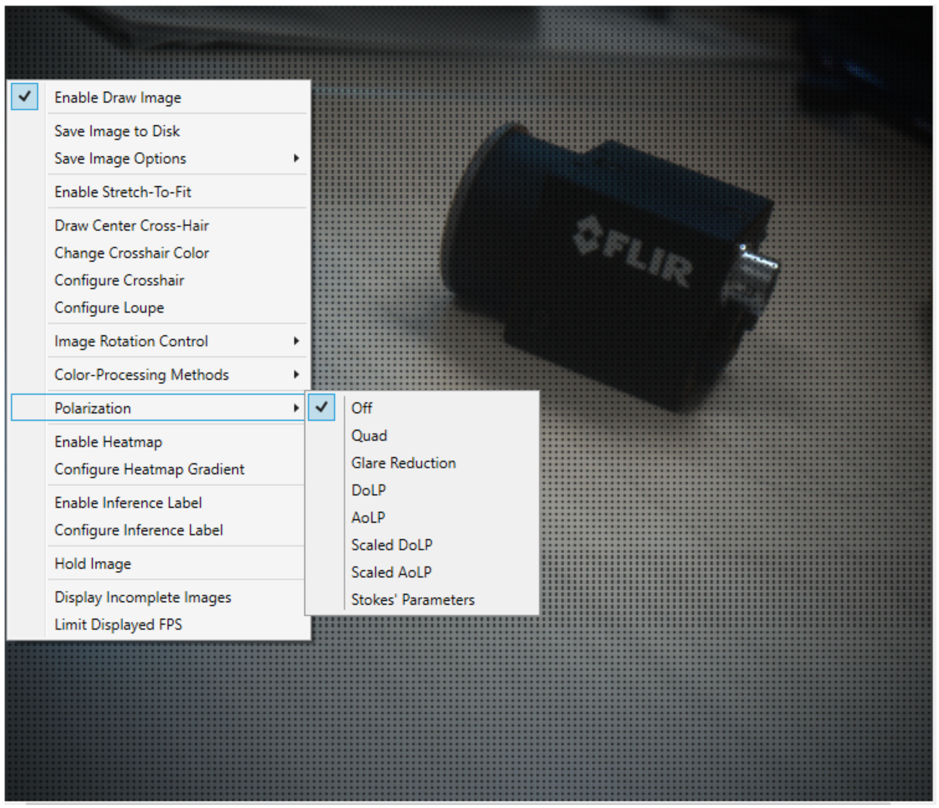
Binning Vertical Mode

Decimation Selector

BayerRG8
BayerRG16
BayerRG10p
BayerRG12p
BayerRGPolarized8
BayerRGPolarized10p
BayerRGPolarized12p
BayerRGPolarized16

Displaying Image Processed with Polarization Algorithms

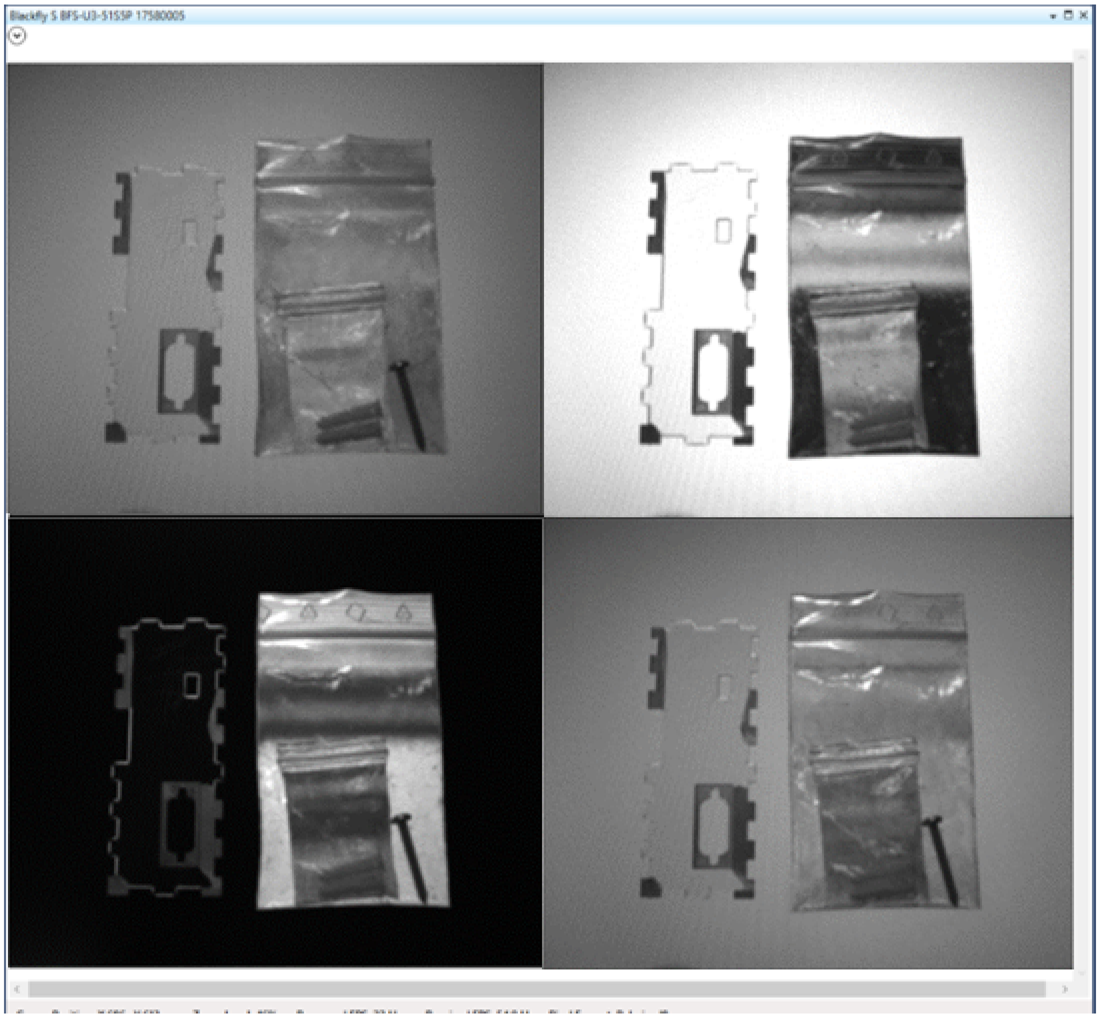
View polarization features by right clicking on the image display window and hovering over Polarization.



Quad

Side-by-side view of all extracted polarized quadrants

Clockwise starting at the top left: 90°, 45°, 0°, 135°



Stokes Parameters

S0

S0 = I0 + I90
The sum of the intensities of the horizontally and vertically polarized pixels
This represents the intensity of the light beam.

S1

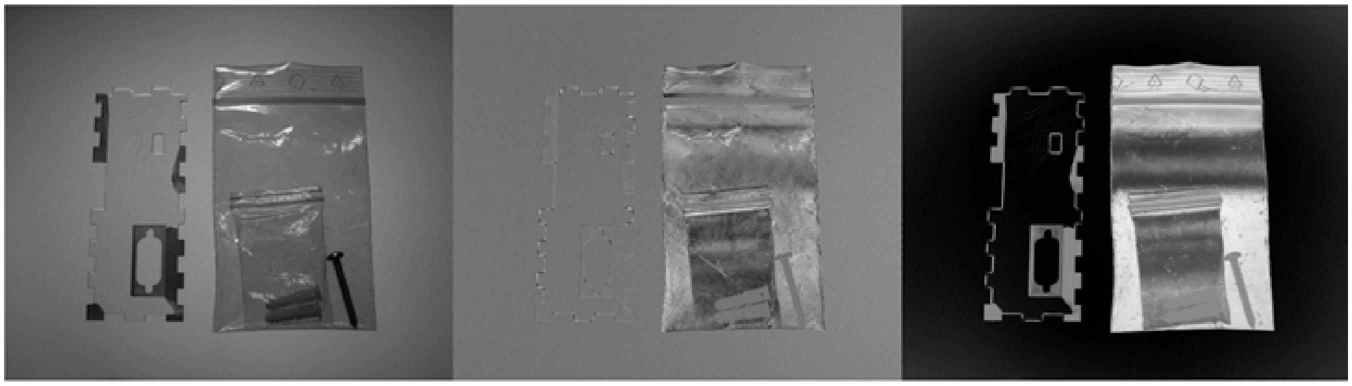
S0 = I0 - I90
The difference between the intensities of the horizontally and vertically polarized pixels

S2

S0 = I45 - I135
The difference between the intensities of the 45° and 135° polarized pixels.

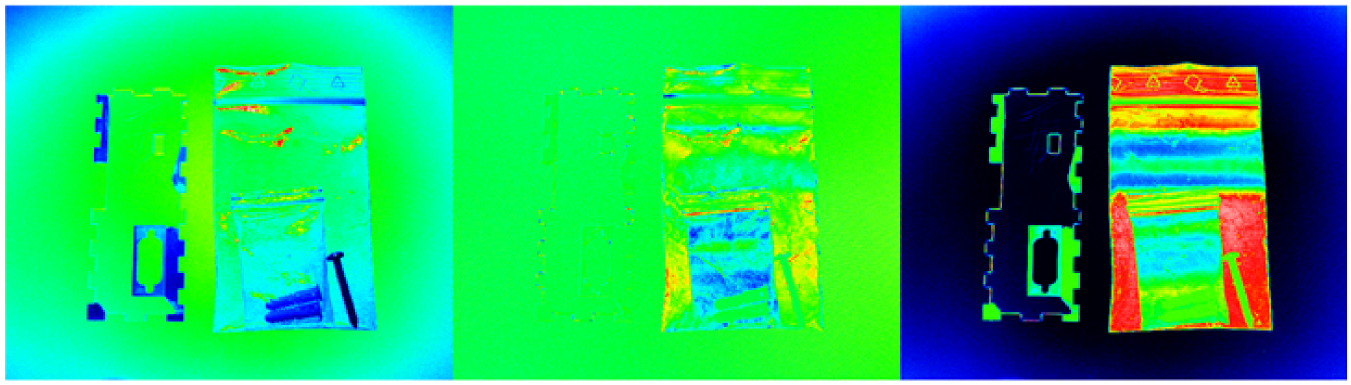
Greyscale

Left to right : S0, S1, S2



HeatMap

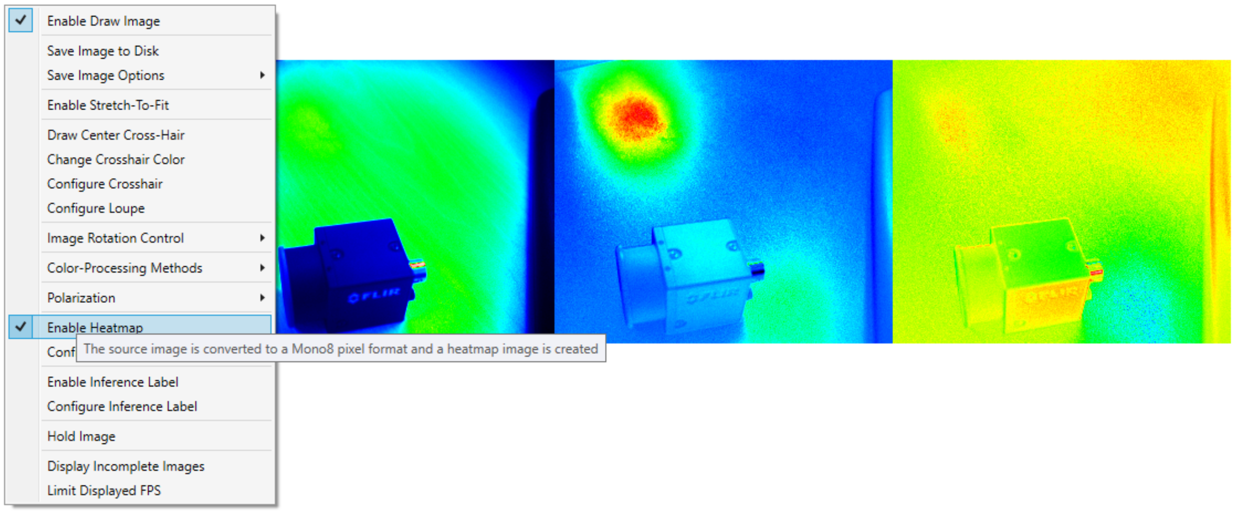
Left to right : S_0, S1, S2



In some cases it may be useful to view the displayed images as heatmaps. Where a color from a gradient is assigned to a pixel according to it's intensity.

For example in the case of Stokes values, colors from the upper end of the gradient represent high stokes values.

To enable heatmaps, right click on the image display window and select Enable Heatmap.

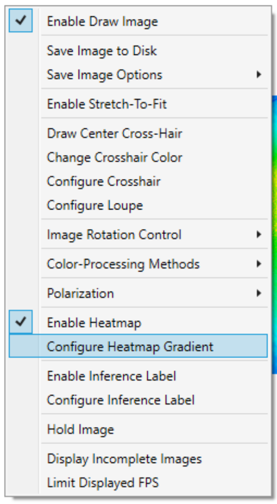


The source image is converted to a Mono8 pixel format and a heatmap image is created

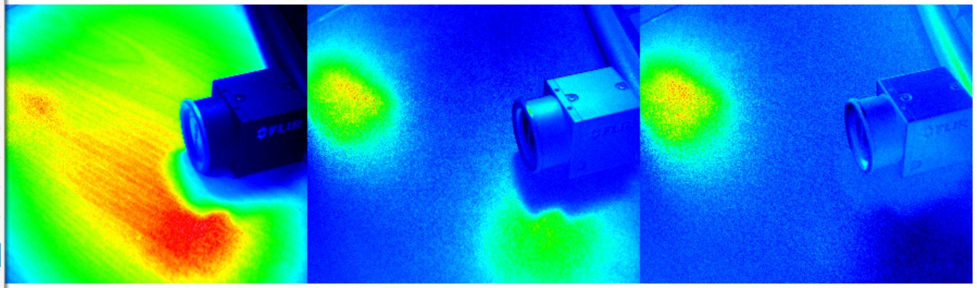
As of SpinView for Spinnaker 1.27 and above, heatmaps are available for all pixel formats.

If heatmap is enabled and the current pixel format is not Mono8, the images are converted to Mono8 before converting to the heatmap image that is displayed.

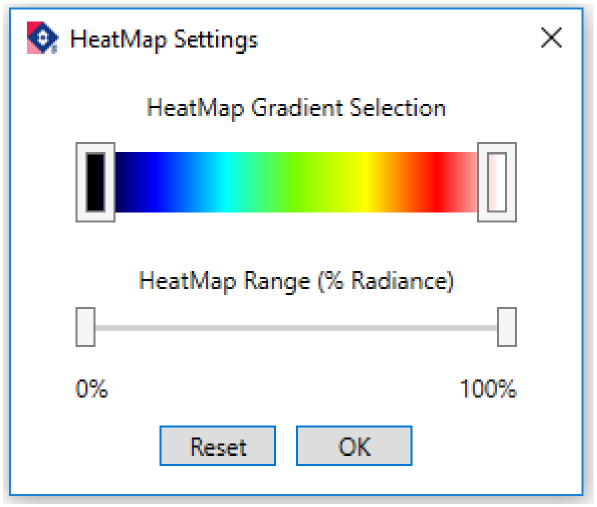
The heatmap settings are configurable by selecting Configure Heatmap Gradient in the display window settings.



- ✓ Enable Draw Image
- Save Image to Disk
- Save Image Options
- Enable Stretch-To-Fit
- Draw Center Cross-Hair
- Change Crosshair Color
- Configure Crosshair
- Configure Loupe
- Image Rotation Control
- Color-Processing Methods
- Polarization
- ✓ Enable Heatmap
- Configure Heatmap Gradient
- Enable Inference Label
- Configure Inference Label
- Hold Image
- Display Incomplete Images
- Limit Displayed FPS



There are options to configure the the heatmap gradient as well as what range of pixel intensities (expressed as % Radiance) to apply the color gradient to.



HeatMap Settings

HeatMap Gradient Selection

HeatMap Range (% Radiance)

0% — 100%

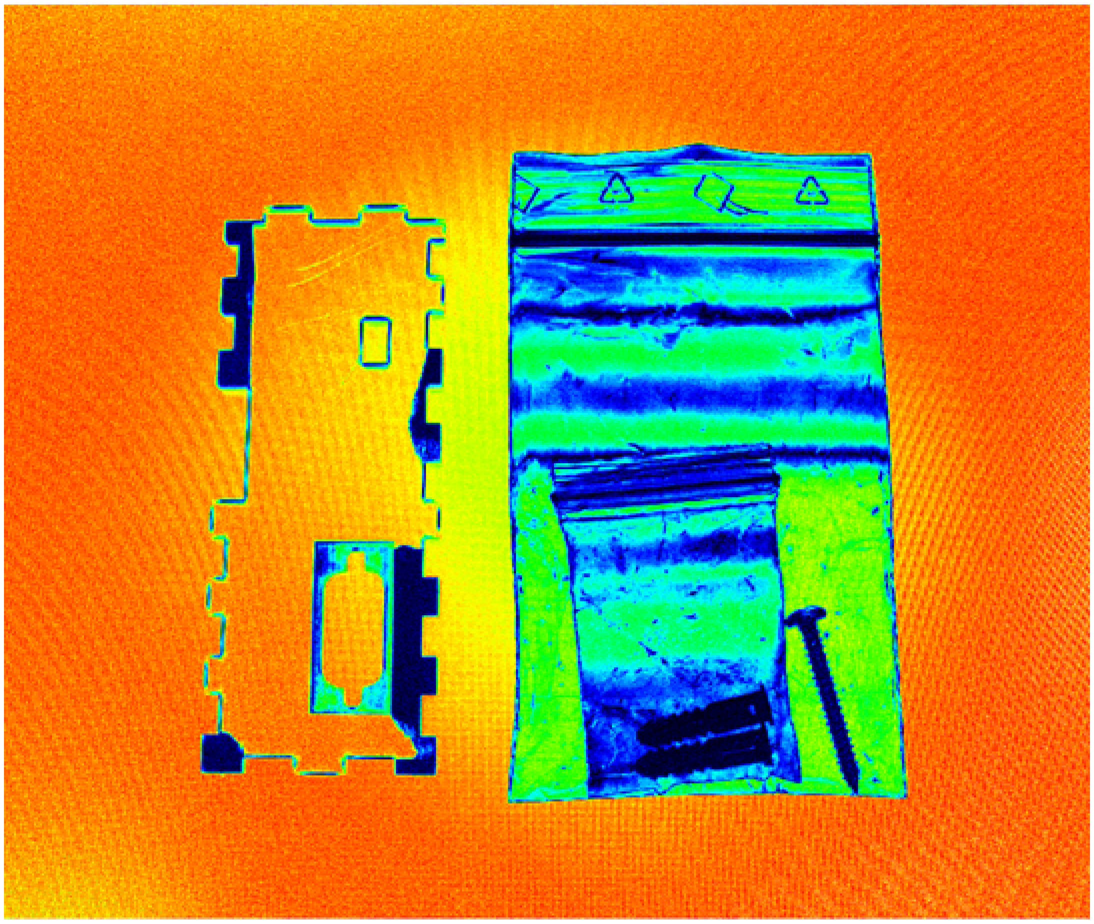
Reset | OK

Degree of Linear Polarization (DoLP)

The proportion of light that is linearly polarized for a given pixel quadrant.

$$DoLP = \frac{\sqrt{S1^2 + S2^2}}{S0}$$

DoLP sample image with heatmap applied:

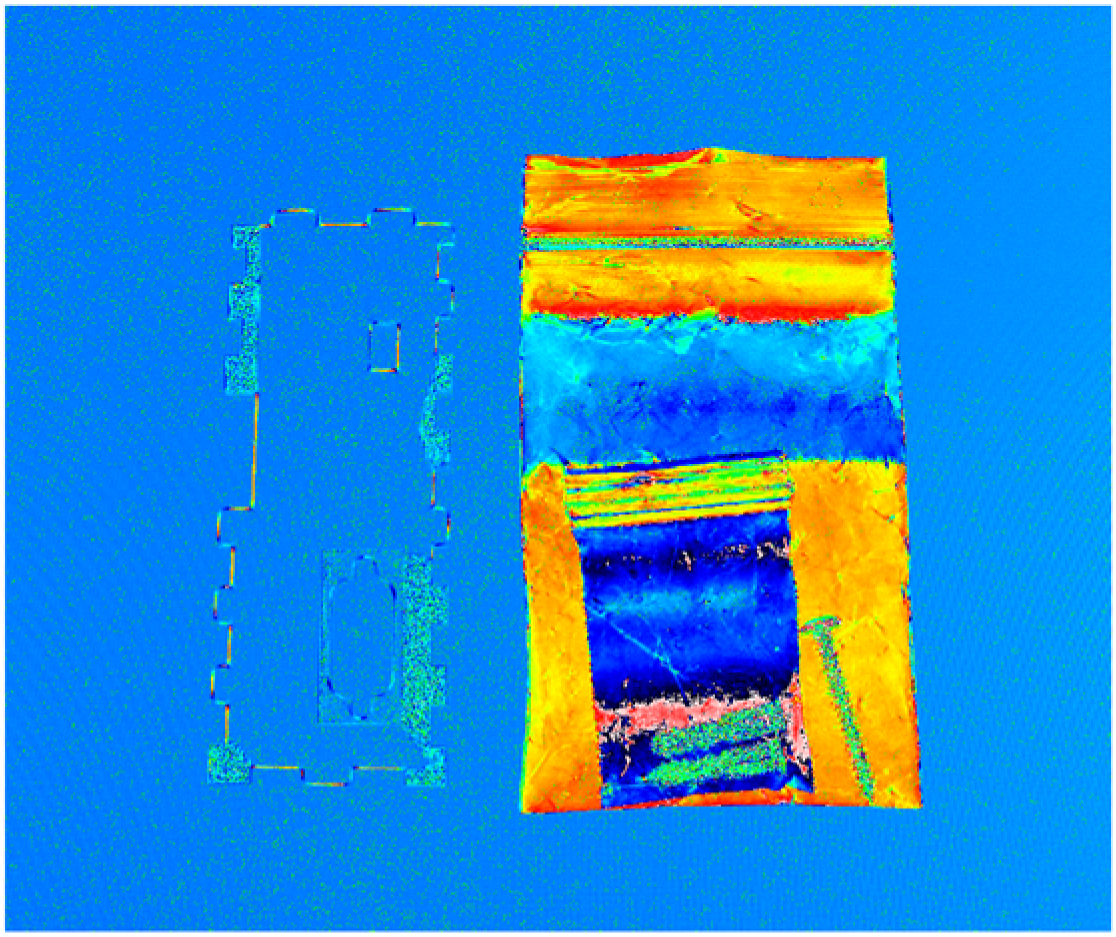


Angle of Linear Polarization (AoLP)

The angle at which linearly polarized light oscillates for a given pixel quadrant.

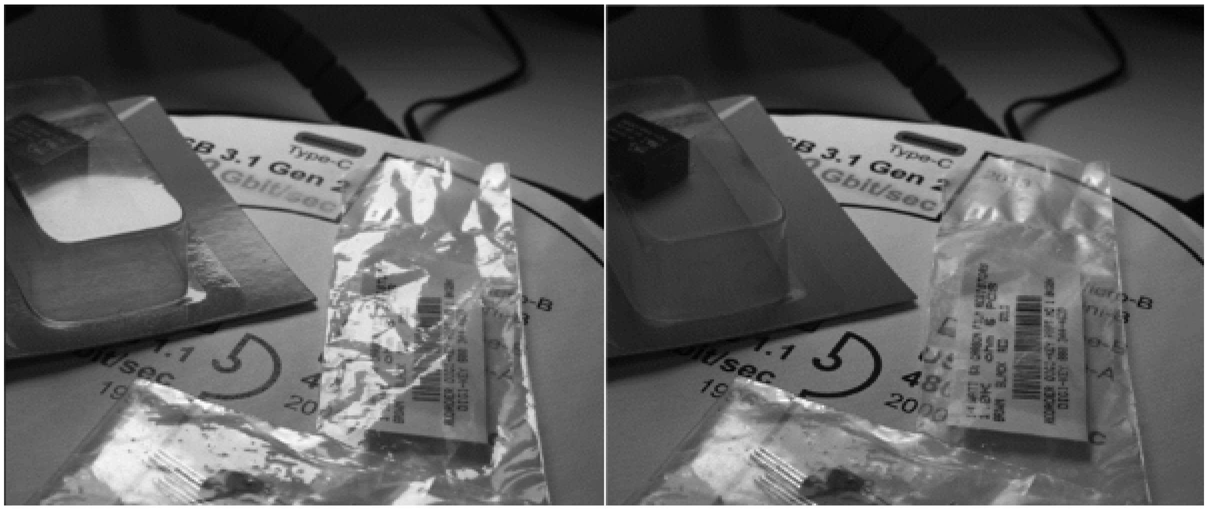
$$AoP = \frac{1}{2}\arctan\left(\frac{S2}{S1}\right)$$

AoLP sample image with heatmap applied:



Glare Reduction

When un-polarized light is incident upon a dielectric surface, the reflected portion of the light is partially polarized according to Brewster's law. Selecting the filtered pixel that most effectively blocks this polarized light in each pixel quadrant reduces glare in the overall image. Since one pixel is selected from each 2x2 polarized pixel quadrant the resulting image will be a quarter of the raw image's resolution.

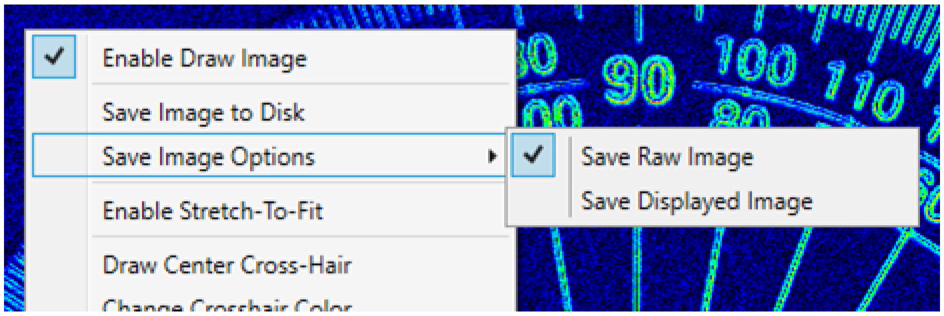


Left: View from a non-polarized camera

Right: View from a camera with on sensor polarization running Glare Reduction in SpinView

Saving the Displayed Images

Spinnaker 1.19.0.22 and above support saving any displayed image in SpinView.
This can be accessed through the same right-click menu on the image display window



Note: This does not apply to the image recording window.

Polarization API Example

Note: There is a complete Polarization C++ example that comes packaged with Spinnaker

Configure a Polarized Pixel Format

As with SpinView, to access the Spinnaker SDK Polarization options you will need to use an 8 bit polarization format.

For monochrome polarization cameras, this is Polarized8. For color polarization cameras this is BayerRGPolarized8.

Using the ImageUtilityPolarization class to apply polarization algorithms

Algorithms that are available through the Spinnaker C++ SDK are as follows:

Quad

The extract polar quadrant algorithm allows you to extract an image comprised of one of the four polarization angles.

```
ImagePtr ImageUtilityPolarization::ExtractPolarQuadrant(const ImagePtr& srcImage,const PolarizationQuadrant
desiredQuadrant);
```

Where PolarizationQuadrant is defined as follows:

```
enum PolarizationQuadrant
{
/** The 0 degree of polarization. */
QUADRANT_I0,
/** The 45 degree of polarization. */
QUADRANT_I45,
/** The 90 degree of polarization. */
QUADRANT_I90,
/** The 135 degree of polarization. */
QUADRANT_I135
};
```

The source image pixel format must be Polarized8 or BayerRGPolarized8.
The destination image pixel format must be Mono8 or BayerRG8 respectively.
The destination image height and width will be half that of the source image.

Glare Reduction

The glare reduction algorithm will produce a glare reduced image by selecting the darkest pixel value from each 2x2 pixel polarization quadrant.

```
ImagePtr ImageUtilityPolarization::CreateGlareReduced(const ImagePtr& srcImage);
```

The source image pixel format must be Polarized8 or BayerRGPolarized8.
The destination image pixel format will be Mono8 or BayerRG8 respectively.
The destination image height and width will be half that of the source image.

Stokes Parameters

The create stokes algorithms allows you to compute the stokes parameters from a raw polarization image.

```
ImagePtr ImageUtilityPolarization::CreateStokesS0(const ImagePtr& srcImage, const ColorProcessingAlgorithm
colorProcessingAlg = DEFAULT);

ImagePtr ImageUtilityPolarization::CreateStokesS1(const ImagePtr& srcImage, const ColorProcessingAlgorithm
colorProcessingAlg = DEFAULT);

ImagePtr ImageUtilityPolarization::CreateStokesS2(const ImagePtr& srcImage, const ColorProcessingAlgorithm
colorProcessingAlg = DEFAULT);
```

The source image pixel format must be Polarized8 or BayerRGPolarized8.
The destination image pixel format will be Mono16s or RGB16s respectively.
The destination image height and width will be half of the source image.

The bounds on the stokes parameters are as follows:

0 <= S0 <= 510

-255 <= S1 <= 255

-255 <= S2 <= 255

Since these bounds are outside of the range of what can be held in an 8 bit format, the stokes parameter images will be stored in a 16 bit format.

To access the raw stokes values you can do the following:

```
const auto* stokesImageData = static_cast<short*>(stokesImagePtr->GetData());
```

Degree of Linear Polarization (DoLP)

```
ImagePtr ImageUtilityPolarization::CreateDolp(const ImagePtr& srcImage, const ColorProcessingAlgorithm
colorProcessingAlg = DEFAULT);
```

The source image pixel format must be Polarized8 or BayerRGPolarized8.
The destination image pixel format will be Mono32f or RGB32f respectively.
The destination image height and width will be half of the source image.

The theoretical bounds on the DoLP image are:

0 <= DoLP <= 1

These values are computed as float and as such the image data is stored in a 32 bit format.

To access the raw DoLP data for an image you can do the following:

```
const auto* dolpImageData = static_cast<float*>(dolpImagePtr->GetData());
```

Angle of Linear Polarization (AoLP)

```
ImagePtr ImageUtilityPolarization::CreateAolp(const ImagePtr& srcImage, const ColorProcessingAlgorithm
colorProcessingAlg = DEFAULT);
```

The source image pixel format must be Polarized8 or BayerRGPolarized8.
The destination image pixel format will be Mono32f or RGB32f respectively.
The destination image height and width will be half of the source image.

The angle of linear polarization in stored in radians and theoretical bounds on the AoLP image are:

-pi/2 <= AoLP <= pi/2

This is around:

-1.57 <= AoLP <= 1.57

These values are computed as float and as such the image data is stored in a 32 bit format.

To access the raw AoLP data for an image you can do the following:

```
const auto* aolpImageData = static_cast<float*>(aolpImagePtr->GetData());
```

Normalizing the Raw Image Data

Available in Spinnaker 1.25+

Image normalization can be used to map image data from one range of values (Source) to another range of values (Destination).

Each pixel in the source image will be mapped to a pixel in the destination image according to the following equation:

DestPixelValue= ((maxDest - minDest) * (SourcePixelValue - minSrc) / (maxSrc - minSrc)) + minDest

(Normalization Formula)

One instance where this is especially useful is when you need to convert image data from floating point computed values to integer pixel values that can be displayed as an image.

For example using the ImageUtilityPolarization you can compute the angle of linear polarization (AoLP) for each 2x2 pixel quadrant of a raw polarized image:

```
const auto aolpImage = ImageUtilityPolarization::CreateAolp(pRawPolarizedImage);
```

The pixel values will be in radians in the range of -pi/2 to pi/2. The aolpImage's image data ("pixels") will now be composed of an array of 32 bit floating point values (pixel format = Mono32f for mono or RGB32f for color), of size one quarter of the original pRawPolarizedImage. One value for each polarized quadrant in pRawPolarizedImage.

If we want to display the AoLP image as a mono8 image for example then we need to map each floating point value in the range of [-1.5708, 1.5708] to an integer value in the range [0, 255].

To accomplish this we will use the ImageUtility's CreateNormalized function as follows:

```
const auto normalizedImage = ImageUtility::CreateNormalized(aolpImage, PixelFormat_Mono8, ImageUtility::
ABSOLUTE_DATA_RANGE);
```

Properties that are important for us in image normalization according to the formula **(Normalization Formula)** are:

Destination bounds(minDest, maxDest)

Source bounds: (minSrc, maxSrc)

The options for creating a normalized images are:

```
static ImagePtr CreateNormalized( const ImagePtr& srcImage, const PixelFormatEnums destPixelFormat,
SourceDataRange srcDataRange = IMAGE_DATA_RANGE);
static ImagePtr CreateNormalized(const ImagePtr& srcImage, const double min, const double max, SourceDataRange
srcDataRange = IMAGE_DATA_RANGE);
static ImagePtr CreateNormalized(const ImagePtr& srcImage, const double min, const double max, const
PixelFormatEnums destPixelFormat, SourceDataRange srcDataRange = IMAGE_DATA_RANGE);
```

For the destination bounds:

We can choose to supply a destination pixel format, destination max & min values, or both. If we do not supply a destPixelFormat then the source pixel format will be used for the destination image. If we do not supply a min and max then the entire range of the destination pixel format will be used.

For the source bounds:

We have the choice to supply an optional enum value called SourceDataRange. If we do not supply it then the default will be IMAGE_DATA_RANGE.

Here is an excerpt from the ImageUtility.h file that explains the SourceDataRange options:

```
/**
* Image normalization source data options.
* Options to normalize the source data based on the max and min values present in the specific
* image (image data) or the theoretical abosolute max and min image data values for the image type (absolute
data).
* By default the abosolute max and min values for an image are the max and min values allowable for
* the image's pixel format. An exception to this is for some computed image data formats such as
* AoLP, DoLP and Stokes, where the absolute max and min are dependant on the algorithm used.
*
* For a given pixel, normalization is done by:
* NormalizedValue = ((maxDest - minDest) * (PixelValue - minSrc) / (maxSrc - minSrc)) + minDest
*/
enum SourceDataRange
{
    /** Normalize based on the actual max and min values for the source image. */
    IMAGE_DATA_RANGE,
    /** Normalize based on the theoretical max and min values for the source image. */
    ABSOLUTE_DATA_RANGE,
    /** Normalize based on the actual min and theoretical max values for the source image. */
    IMAGE_MIN_ABSOLUTE_MAX,
    /** Normalize based on the theoretical min and actual max values for the source image. */
    ABSOLUTE_MIN_IMAGE_MAX
};
```

So in the case of an AoLP image we will choose to normalize the image based on the ABSOLUTE_DATA_RANGE, and use the default destination image bounds so that the minimum and maximum theoretical angles [-pi/2, pi/2] will be mapped to the bounds [0, 255] of the destination image. This is important for if we want to turn the image into a heat map to get a better visualization of the angle of polarization in different parts of the image. For a heat map a color is assigned to a pixel value and so we want to know what color corresponds to what angle. For example the lowest color corresponds to the lowest possible angle (not the lowest angle in the image) so we can accurately determine the angle at any location based on the color of the pixel. If we normalized based of the IMAGE_DATA_RANGE then different colors could correspond to different angles in different images.

Note on absolute data range:

For some computed image types whose data bounds are not the pixel format bounds, the absolute data range will be assigned when the image is created. In the case of AoLP images the computed data bounds are [-pi/2, pi/2] or equivalently [-1.5708, 1.5708] approximately. Whereas since the pixel data type is 32 bit float, the bounds for the pixel format would be [-3.40282e+38, 3.40282e+38] and if we were to normalize this data range into [0, 255] when all the actual values were bound by [-1.5708, 1.5708] then we would not get any usable data.

For images produced with the AoLP, DoLP and Stokes algorithms, the absolute data range is as follows:

ImageUtilityPolarization::CreateAolp -> [-1.5708, 1.5708] approximately

ImageUtilityPolarization::CreateDolp ->[0.0, 1.0]

ImageUtilityPolarization::CreateStokesS0 -> [0, 510]

ImageUtilityPolarization::CreateStokesS1 -> [-255, 255]

ImageUtilityPolarization::CreateStokesS2 -> [-255, 255]

In all other cases if ABSOLUTE_DATA_RANGE is used then the pixel format theoretical max and min will be used as the source min/max

For SourceDataRange =IMAGE_DATA_RANGE, prior to normalization the max and min pixel values for the specific image will be determined and used as the srcMin and srcMax in the calculation. The benefit of this is that the entire range of the destination image's pixel format will be utilized. The actual minimum/maximum pixel value in the source will become the theoretical minimum/maximum pixel value in the destination. For example if we have a mono8 image and the minimum pixel value in the image is 20 (pixel A) and the maximum pixel value is 100 (pixel B). If we are normalizing to a mono16 image then pixel A' (normalization of pixel A) will be 0 and pixel B' (normalization of pixel B) will be 65535. All values in between 20 and 100 in the source image will be normalized accordingly to be between 0 and 65535.

Creating a Heat Map from an Image

It may be useful to visualize some images by mapping their pixel intensities to a gradient of colors.

For example with AoLP and DoLP images it can be easier to visualize the angle or degree of polarization when it is displayed as a heat map.

This way the range of angles/degrees present in the AoLP/DoLP images can be mapped to a gradient of colors which can help to visualize these values.

To do this we will use the Spinnaker SDK ImageUtilityHeatmap:

```
static ImagePtr CreateHeatmap(const ImagePtr& srcImage);
```

Mono Example
Here is an example of how we can create an AoLP image, normalize the image into the mono8 pixel format, create a heat map image and then save it.

```
const auto pAolpImage = ImageUtilityPolarization::CreateAolp(pRawPolarizedImage);
const auto pAolpNormalizedImage = ImageUtility::CreateNormalized(pAolpImage, PixelFormat_Mono8, ImageUtility::
SourceDataRange::ABSOLUTE_DATA_RANGE);
const auto pAolpHeatmapImage = ImageUtilityHeatmap::CreateHeatmap(pAolpNormalizedImage);
pAolpHeatmapImage->Save("AolpHeatMap.jpg");
```

Color Example
For color polarization cameras we have to normalize to PixelFormat_RGB8, then convert the normalized image to mono8 before using it to create a heatmap:

```
const auto pAolpImage = ImageUtilityPolarization::CreateAolp(pRawPolarizedImage);
const auto pAolpNormalizedImage = ImageUtility::CreateNormalized(pAolpImage, PixelFormat_RGB8, ImageUtility::
SourceDataRange::ABSOLUTE_DATA_RANGE);
const auto pAolpHeatmapImage = ImageUtilityHeatmap::CreateHeatmap(pAolpNormalizedImage->Convert
(PixelFormat_Mono8));
pAolpHeatmapImage->Save("AolpHeatMap.jpg");
```

It is also possible to set the low and high colors used for the heat map gradient. By default it is set from HEATMAP_BLACK to HEATMAP_WHITE.

As an example here is how to configure the default setting:

```
ImageUtilityHeatmap::SetHeatmapColorGradient(ImageUtilityHeatmap::HEATMAP_BLACK, ImageUtilityHeatmap::
HEATMAP_WHITE);
```

The heat map can also be manipulated to focus on a certain range of values. Using this you can have the heat map color gradient span a certain percentage of the input image's intensity values.

By default the color gradient will span the whole pixel format range of the source image data. From 0% to 100%. If the srcImage image was mono8 this would mean 0 to 255. The SetHeatmapRange takes in values in percent.

Here is how to set the full range:

```
ImageUtilityHeatmap::SetHeatmapRange(0, 100);
```

To focus on a certain range of AoLP angles for example you will have to map a range of angles [-pi/2 , pi/2], to a range of percent [0, 100]. You will then input the percent values into ImageUtilityHeatmap::SetHeatmapRange

Permissions: Public

Related tickets

Related bugs

Other resource