TECHNICAL SUPPORT

Embedded Image Processing

C-RED Cameras

Table

Glossary

- **HDR**: High Dynamic Range
- **ADU**: Analog Digital Unit
- **HG**: High Gain
- **LG**: Low Gain
- **ADC**: Analog to Digital Converter
- **NUC**: Non Uniformity Correction
- **IWR**: Integrate While Read
- **ITR**: Integrate Then Read
- **FPS**: Frames Per Second
- **CDS**: Correlated Double sample

1. Introduction

First Light Imaging has implemented an FPGA-embedded image processing workflow. This enables image correction and processing to be performed before sending data over CameraLink® or USB. The purpose is to optimize real-time applications.

The workflow which applies is described below. This document focuses on the "Convolution (variant 2) Bad pixel correction" (enabled by default), "Filtering" and "Thresholding" blocks. These functionalities can be toggled on or off. The other functionalities are described in the camera user manuals.

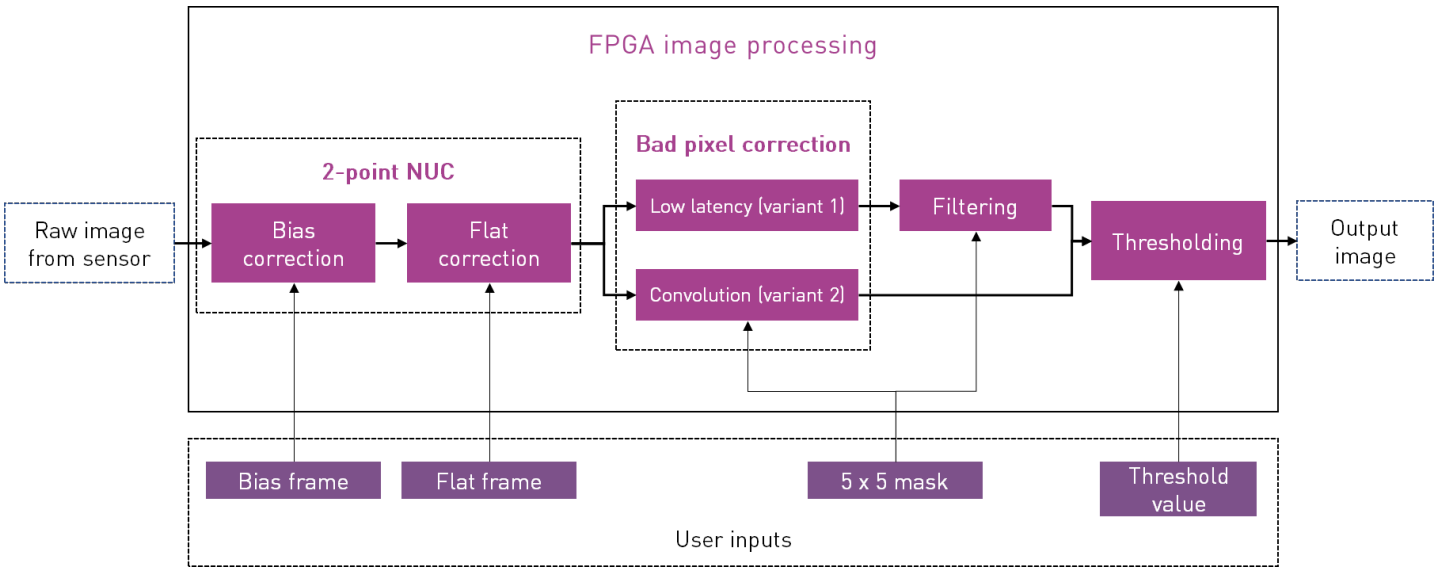


Fig. 1 : FPGA-embedded image processing workflow

The filtering capability is available for C-RED 3 and C-RED 2 Lite from Firmware 2.3.1 onward and for C-RED 2 and C-RED 2 ER from Firmware 5.3.0 onward. The Software Development Kit (SDK) supports it from Version 2.9.3 onward and First Light Vision software supports it from Version 2.5.1 onward.

To track the availability of all the mentioned features, a software feature has been added.
If the features are available, the following bit will be set to 1.
#define SW_FEATURES_FILTERING 0x004

Note: You can only use one of the two convolution corrections at the same time: either you enable filtering, and the second variant of the bad pixel correction will be unavailable, or you enable this convolution bad pixel correction, and filtering will be unavailable.

2. Image filtering

2.1. Introduction

Filtering is one of the fundamental operations in image processing. The general idea is that a pixel is adjusted by values in its surrounding neighborhood.

The "convolution matrix" defines which neighboring pixels to consider when filtering and how much to weight those pixels. Filtering corresponds to the mathematical convolution operation (noted "*") between the input image I and the convolution matrix C. This is illustrated below and can be essentially summed up as:

- Center a convolution matrix on a pixel
- Multiply the pixels *under* that kernel by the values *in* the kernel
- Sum all those results
- Replace the center pixel with the summed result

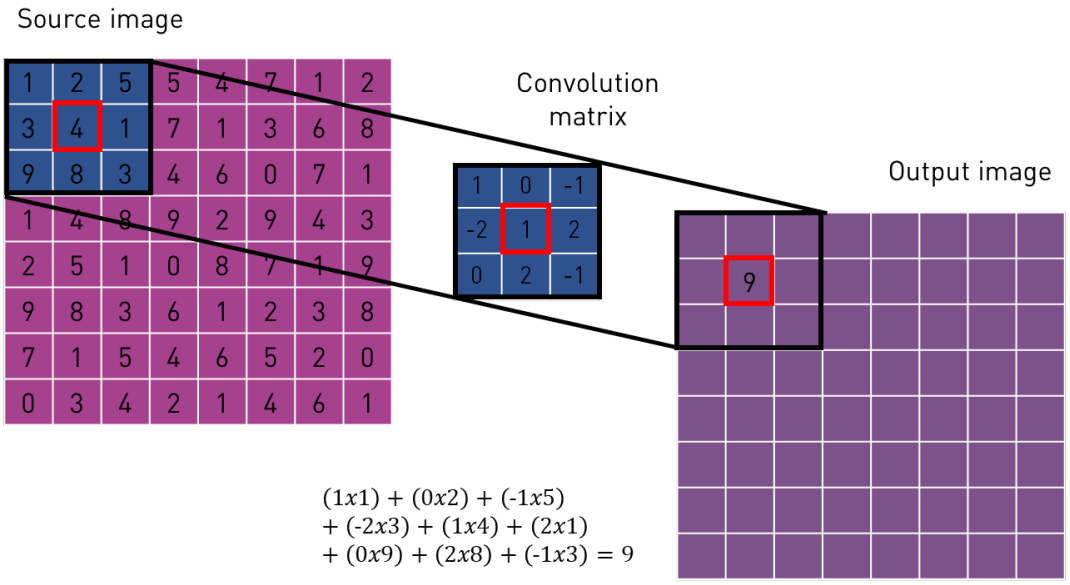


Fig. 2 : Illustration of the convolution principle

Different convolution matrix will have different effects on the source image, for example removing noise, enhancing features, detecting edges, improving sharpness, making the image brighter or darker. Filtering can also be used as a pre-processing step before operations such as thresholding.

2.2. Convolution matrix

2.2.1. Definition

In the camera, a 5-by-5 coefficient matrix can be defined by the user to adjust the value of a pixel according to its surrounding pixels. A generic convolution matrix is shown below. The highlighted coefficient $c_{3,3}$ is the one that will be centered on the pixel which will be modified.

$$M = \frac{1}{p} x \begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} & c_{4,5} \\ c_{5,1} & c_{5,2} & c_{5,3} & c_{5,4} & c_{5,5} \end{pmatrix}$$

Fig. 3 : Convolution matrix used for image processing

The p coefficient is called the $divisor$.
The matrix $total\ weight$ is the sum of all coefficients in the matrix : $\sum_{i,j} c_{i,j}$.
And the matrix $gain$ can be defined as : $\frac{\sum_{i,j} c_{i,j}}{p}$.

The purpose of the gain feature is to enhance (if $gain > 1$) or diminish (if $gain < 1$) image brightness.
When $p = \sum_{i,j} c_{i,j}$ then $gain = 1$ and the image brightness is preserved.

Note: The user will be able to define the $c_{i,j}$ coefficients and the divisor p. However, the user should keep in mind that the $gain$ is computed from $\sum_{i,j} c_{i,j}$ and p.

Image pixels within the 5-by-5 matrix are used to compute the central pixel (the one centered on $c_{3,3}$) only if:

- The pixel has a non-null associated $c_{i,j}$ coefficient
- The pixel is within the frame
- It is not part of the excluded frame border

2.2.2. Convolution matrix file format

Each convolution matrix file is 7 lines long and are derived from the .csv file format.

- The first line is used only for file description. It can be a title, or any other relevant information.
- The second line holds the divisor coefficient p. This coefficient is read after the first colon.
- The last 5 lines are used to describe the coefficients of the convolution matrix C.

Data is separated by semicolons, and lines are marked using carriage returns ("\n").

An example of a convolution matrix file is as follows:

```
Description:This is a default 5*5 gaussian convolution matrix
Divisor:22639.9
1;29.2;90;29.2;1;
29.2;854.1;2630.7;854.1;29.2;
90;2630.7;8103.1;2630.7;90;
29.2;854.1;2630.7;854.1;29.2;
1;29.2;90;29.2;1;
```

Fig. 4 : Example of a convolution matrix file

Each $\frac{c_{i,j}}{p}$ coefficient has an 8-bit fractional precision and has a 10-bit whole part. This translates into:

- The $\frac{c_{i,j}}{p}$ coefficients must be set between -512 and +512 (which provide 2^{10} whole part values)
- The $c_{i,j}$ coefficients of the convolution matrix can take large values, as long as the divisor p is large enough.
- If the matrix total weight divided by the divisor ($|\frac{1}{p}\ x\ \sum_{i,j} c_{i,j}|$) is less than $0.003\ (=\ 2^{-8})$, it will be considered zero by the firmware.
- If the matrix total weight $\sum_{i,j} c_{i,j}$ is equal to zero, the divisor p should be set considering the matrix total weight as equal to one.

Additionally, each $|\frac{c_{i,j}}{\max(c_{i,j})}|$ is defined on 14-bits in the firmware. Hence, every absolute value of $c_{i,j}$ coefficient divided by the matrix largest coefficient $\max(c_{i,j})$, must be greater than 0.00006 $(=\ 2^{-14})$,. Else, it will be rounded to 0 by the firmware.

It is possible to set coefficients to 0, thus ignoring the pixel associated to the coefficient set to 0. For example, to define a 3-by-3 convolution matrix :

$$M = \frac{1}{p} x \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & c_{2,2} & c_{2,3} & c_{2,4} & 0 \\ 0 & c_{3,2} & c_{3,3} & c_{3,4} & 0 \\ 0 & c_{4,2} & c_{4,3} & c_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Fig. 5 : Definition of a 3-by-3 convolution matrix

Note: If the format is not carefully respected, in particular regarding the divisor coefficient, the camera behavior will be unknown. No specific warning will be issued to the user by the firmware. It may be easier to configure using First Light Vision software.

Calculated pixel values are rounded to the closest smaller or equal unit value. Ex: 2.3 = 2. 2.99 = 2. These pixel values are defined on 16-bits.

2.2.3. Customization and storage of convolution matrix files

Values are stored in camera presets. For each of the 10 camera presets, there are 10 convolution matrix files: one pre-defined and 9 that can be customized. Hence, up to 100 convolution matrix files can be saved in the camera.

A stored convolution matrix file can be pointed at using the "`convolutionMatrixIndex`" index.

- To get the current convolution matrix file index, use the command "`convolutionMatrixIndex`".
- To set the current convolution matrix file index, use the command "`set convolutionMatrixIndex X`" where X is an integer between 0 and 9 included. Setting a matrix file index using this command updates the current convolution matrix.

To retrieve a convolution matrix file from the camera, the following commands are to be used:

- "`recvfile convolutionMatrix size`": returns the size of the convolution matrix file stored in the camera.
- "`recvfile convolutionMatrix md5sum`": returns the md5sum of the convolution matrix file
- "`recvfile convolutionMatrix content`": returns the content of the convolution matrix file

To send a file to the camera, use the command "`sendfile convolutionMatrix`".

- Files with a convolutionMatrixIndex equal to 0 are hardcoded and cannot be customized.
- All files with convolutionMatrixIndex not 0 can be customized by the user.

Note: If the file that is changed is currently in use, the correction will not be updated as the file is overwritten. To update the current correction, the correction must be enabled again.

Note: The "`restoreFactory`" command also restores all the convolution matrix.

2.2.4. Borders

It is possible to ignore the border of the image when filtering.
It means that the first and last line and column will be set to a specific value. When using filtering mode, it will set to 0 the excluded pixels. This mode is off by default.

The commands are :

- "`set excludeBorders on`"
- "`set excludeBorders off`"

2.2.5. Limitations

To use the filtering functionality, the low latency (variant 1) is the only method which can be used to perform bad pixel correction. The default (variant 2) method cannot be used simultaneously with filtering.

Using the filtering mode introduces an additional 3-lines long latency. Refer to the camera user manual for more information on latency.

If the user switches between the bad pixel correction variant 2 method and the filtering feature, he should keep in mind that the same convolution matrix are used. Hence, if the current matrix in use is modified when in bad pixel variant 2 mode, this same matrix index will be used when switching to filtering.

2.2.6. Convolution matrix examples

Depending on the coefficients, a convolution matrix can have a wide range of effects on the image. Some of the common operations that can be achieved with single-step convolution using a 3-by-3 or 5-by-5 matrix are detailed below. Of course, other matrix can be defined for custom operation.

2.2.6.1. Box blurring using the mean convolution matrix

The mean convolution is the most straightforward convolution operation. It can be defined as a 3-by-3 matrix or larger. Its effect is "smoothing", or blurring, on the image. This is useful to remove noise. The dimension of the matrix defines the spatial dimension of the features which will be filtered out.

$$M_3 = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad M_5 = \frac{1}{25}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Note : In this example the $c_{i,j}$ coefficients are set to 1. The matrix $total\ weight$ is 9 for M_3 and 25 for M_5. The p coefficient is tuned to achieve a $gain$ of 1. Hence, the brightness of the image will not be modified.

2.2.6.2. Gaussian blurring

Gaussian convolution matrix can be defined using approximations of the Gaussian 2D distribution. As with box blurring, a larger convolution matrix will result in a greater degree of filtering.
The main parameter of the Gaussian filter is its standard deviation. To obtain weights close to 0 one the edges and optimize discretization, the matrix dimension should be $1 + 2\,x\,3\sigma$. Meaning the standard deviation of a 5-by-5 mask should be 2/3 and that of a 3-by-3 mask should be 1/3.

$$M_3 = \frac{1}{512}\begin{pmatrix} 0.0605 & 5.4432 & 0.0605 \\ 5.4432 & 489.9852 & 5.4432 \\ 0.0605 & 5.4432 & 0.0605 \end{pmatrix}$$

$$M_5 = \frac{1}{512}\begin{pmatrix} 0.0226 & 0.6609 & 2.0357 & 0.6609 & 0.0226 \\ 0.6609 & 19.3145 & 59.4927 & 19.3145 & 0.6609 \\ 2.0357 & 59.4927 & 183.2505 & 59.4927 & 2.0357 \\ 0.6609 & 19.3145 & 59.4927 & 19.3145 & 0.6609 \\ 0.0226 & 0.6609 & 2.0357 & 0.6609 & 0.0226 \end{pmatrix}$$

2.2.6.3. Image gradient computation

The convolution operation can be used for image gradient computation, *ie.* oriented-edge detection.

The Prewitt matrix is a simple, discrete differentiation operator which computes an approximation of the gradient of the image intensity. The M_x matrix will detect gradient in the horizontal orientation, while the M_y matrix will detect gradient along the vertical orientation.

$$M_{x3} = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \ and\ M_{y3} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

The Sobel convolution matrix performs a gradient operation along with smoothing.

$$M_{x3} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \ and\ M_{y3} = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Larger convolution matrix can be used for the same purpose with enhanced filtering performances. An example of a large Prewit and a large Sobel convolution matrix, respectively, are provided below. Note that other variants can be found in the literature.

$$M_{x5} = \begin{pmatrix} -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \end{pmatrix} \qquad M_{x5} = \begin{pmatrix} -5 & -4 & 0 & 4 & 5 \\ -8 & -10 & 0 & 10 & 10 \\ -10 & -20 & 0 & 20 & 20 \\ -8 & -10 & 0 & 10 & 10 \\ -5 & -4 & 0 & 4 & 5 \end{pmatrix}$$

Note: For these convolution matrix, the sum of the matrix coefficient is 0. Hence, in homogeneous regions of the image, the convolution will output a value of 0, meaning no edge is detected.

2.2.6.4. Laplacian edge detection

Laplacian convolution matrixes are used for edge detection. Mathematically, they compute the second derivative of an image. The resulting image will yield a positive value on the darker side and a negative value on the lighter side of the edge.
There are different methods to for computing the discrete approximation of the Laplacian operator, hence various matrix can be found in the literature. Commonly used convolution matrix are shown below:

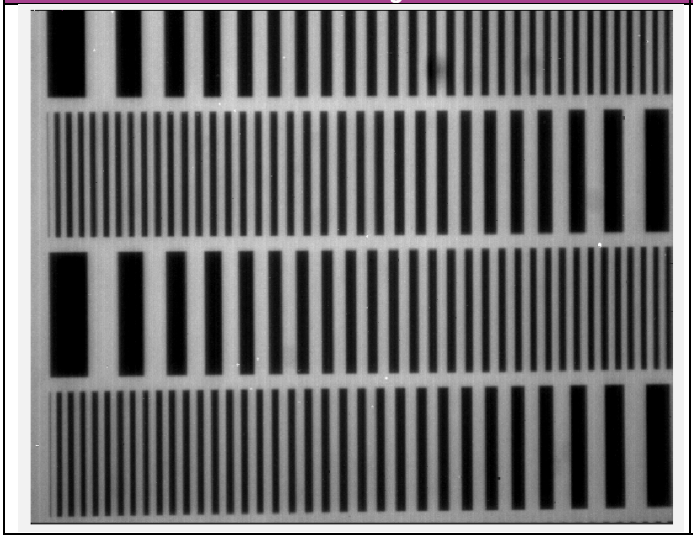
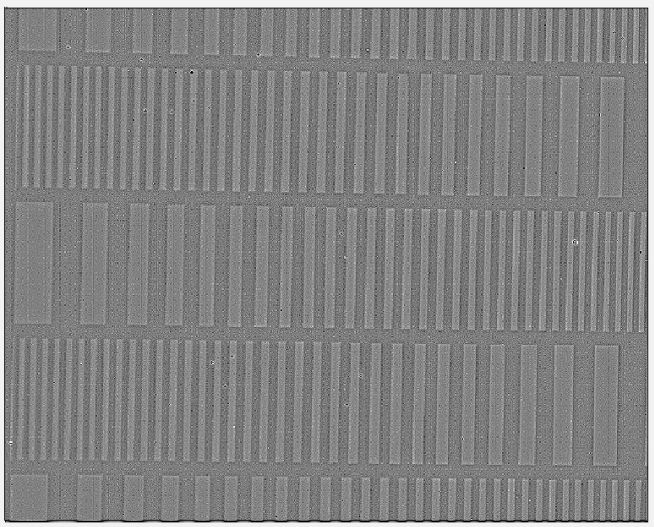
$$M_{xy3} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad M_{xy3} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

The Laplacian edge detection being very sensitive to noise, it is common to combine it to a Gaussian filtering in order to include noise smoothing. This can be achieved as a two-step convolution or by combining the Laplacian and the Gaussian functions in a single convolution matrix. This operation is called Laplacian of Gaussian (LoG).

A possible convolution matrix, with a standard deviation of 2/3 would be:

$$M_5 = \begin{pmatrix} 0 & 3 & 6 & 3 & 0 \\ 3 & 21 & 7 & 21 & 3 \\ 6 & 7 & -161 & 7 & 6 \\ 3 & 21 & 7 & 21 & 3 \\ 0 & 3 & 6 & 3 & 0 \end{pmatrix}$$

A typical result, obtained with a C-RED 3 camera is shown below. Bad pixel and bias corrections are applied.

Raw image	After Laplacian edge detection
	

2.2.6.5. Image embossing

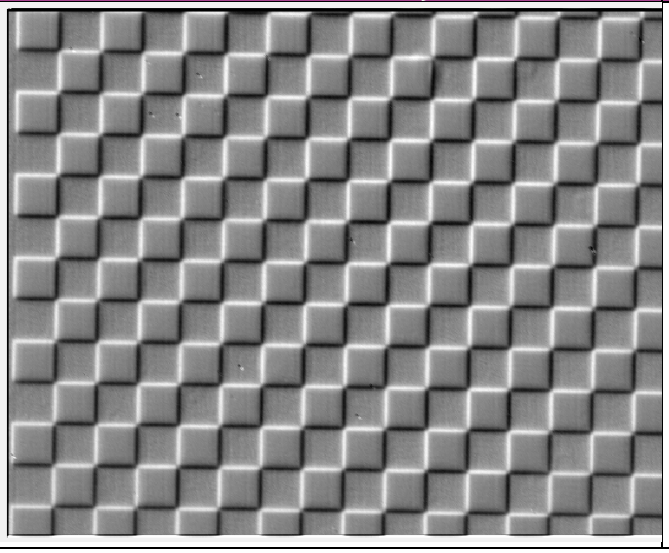
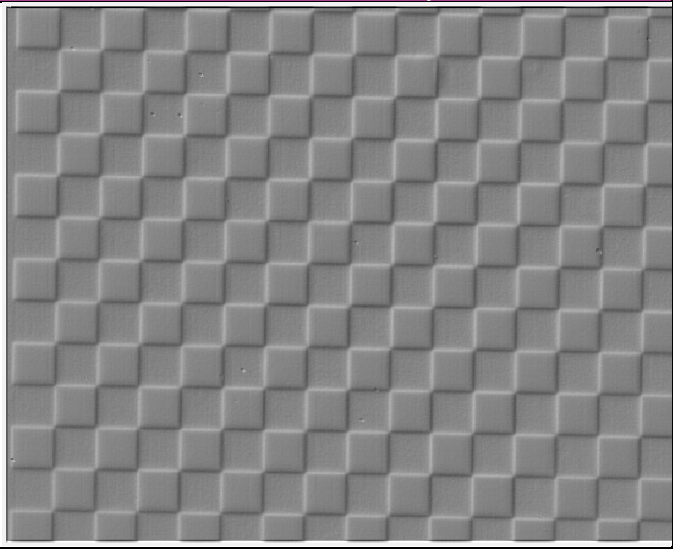
The emboss convolution matrix is a directional difference filter which will enhance the edges in a specific direction (vertical, horizontal or diagonals). The four primary emboss convolution matrix shown below can be rotated.

$$\begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

In order to tune the depth of the edges, the emboss convolution matrix can be enlarged.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

The effect of the convolution matrix dimension is illustrated bellow with an emboss filter.

Emboss filter 5-by-5	Emboss filter 3-by-3
	

2.2.6.6. Sharpness enhancement

Numerous sharpening methods are available with convolution matrix. One technique is to subtract a blurred image (for example using a Gaussian operator) to the image. In a single convolution matrix this can be expressed as :

$$L_{y3} = -\frac{1}{512}\begin{pmatrix} 0.0226 & 0.6609 & 2.0357 & 0.6609 & 0.0226 \\ 0.6609 & 19.3145 & 59.4927 & 19.3145 & 0.6609 \\ 2.0357 & 59.4927 & -328.7495 & 59.4927 & 2.0357 \\ 0.6609 & 19.3145 & 59.4927 & 19.3145 & 0.6609 \\ 0.0226 & 0.6609 & 2.0357 & 0.6609 & 0.0226 \end{pmatrix}$$

2.3. First Light Imaging software tutorial

The filtering capability of the camera is fully integrated into the First Light Vision software.

2.3.1. Filtering feature manager window

The filtering feature window can be opened using the specific tab on the top left of the software interface.

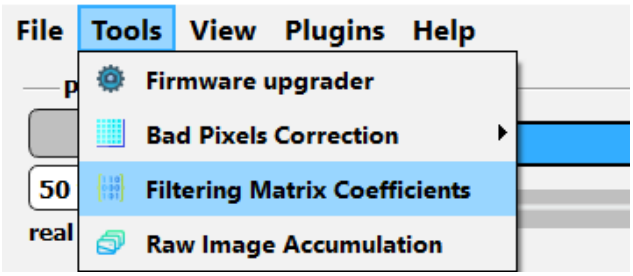


Fig. 6 : Menu to customize the convolution matrix coefficients

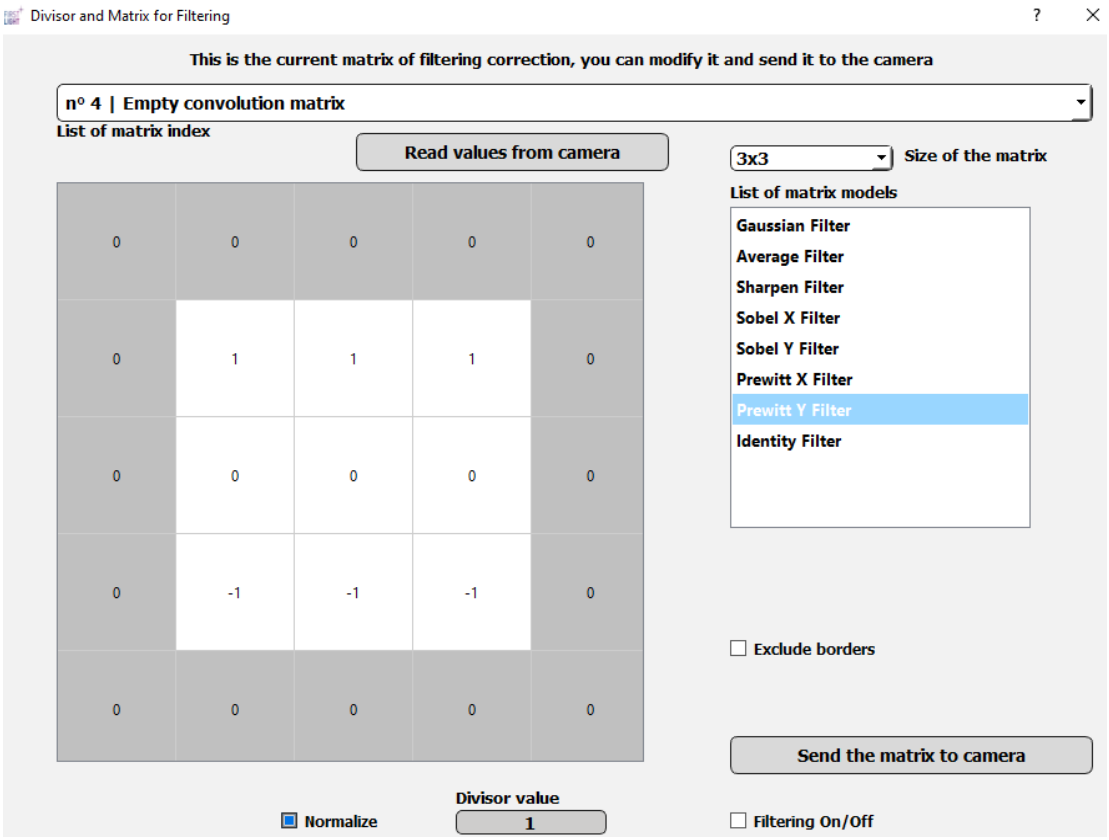


Fig. 7 : Interface of the filtering feature

The workflow for defining and applying a new matrix for a convolution operation is the following:

(1) **Choose the matrix index**. Up to 10 matrixes can be defined and saved in each camera preset. The first matrix (n°0) is a default matrix and cannot be modified by the user. The other 9 matrixes can be customized and their description can be modified.

(2) **Define the size of the matrix**, either 3-by-3 or 5-by-5. For the definition of a 3-by-3 matrix there is still a 5-by-5 matrix, but the borders are set to 0 and cannot be modified, these cells have a gray background.

(3) **Define the matrix coefficients**. This can be done by choosing a predefined matrix among the listed models or by customizing the values directly on the box representation of the matrix.

(4) **Set the divisor value**. By default, the divisor value p is the sum of all the cells, and 1 if the sum is equal to 0. This enables to have a $gain$ of 1 and preserve image brightness.

If the "normalize" check box is unchecked the user can enter another value. The effect of the divisor value p is described in paragraph 2.2.1.

(5) **Save the matrix and divisor value**. Once the matrix is fully defined with matrix cell coefficients and a divisor value, the user can send it to the camera with the button "Send the matrix to the camera".

(6) **Enable filtering**. Check the "filtering on/off" checkbox.

The "Exclude borders" checkbox will ignore the borders of the image (first line, last line, first column and last column) during the convolution operation. The exclusion is made by setting an arbitrary 0 value (black) for these pixels. The purpose of excluding the border is to avoid abnormal pixel values from disturbing pixels around them. **This parameter is saved independently for each matrix index.**

The button "Read values from camera" the user can get the values saved inside the camera for the selected matrix index.

2.4. Software Development Kit

Here are the relevant commands to use to enable and configure the filtering feature with FliSDK. Please refer to the SDK's documentation (FirstLight_SDK_2_9_x_API_User_Manual_20230529.pdf) for more information.

In the FliSerialCamera, we have the following command:

- sendCommand(), to be used with the "convolutionMatrixIndex" and "set convolutionMatrixIndex X" commands to get the index of the currently loaded convolution matrix and change it, respectively.

In the FliCred class, we have the following methods:

- getUserConvolutionMatrix() and setUserConvolutionMatrix() are used to get the current convolution matrix and divisor coefficient and configure them, respectively.
- getFilteringModeOnOff() and setFilteringModeOnOff() are used to get whether the filtering is currently on or off and change it, respectively.

2.5. Command Line Interpreter commands

To enable the filtering mode, use the command "`set filtering on`".
To disable this mode, use the command "`set filtering off`".
To get this mode's status, issue the command "`filtering`".

Setting the filtering mode to "`on`" updates the current convolution matrix file in use.

As described earlier on, the other commands associated to the convolution matrix are:

- "`convolutionMatrixIndex`" : index.
- "`convolutionMatrixIndex`" : to get the current convolution matrix file index, use the command
- "set `convolutionMatrixIndex X`": to set the current convolution matrix file index, use the command where X is an integer between 0 and 9 included. Setting a matrix file index using this command updates the current convolution matrix.

- "`recvfile convolutionMatrix size`": returns the size of the convolution matrix file stored in the camera.
- "`recvfile convolutionMatrix md5sum`": returns the md5sum of the convolution matrix file
- "`recvfile convolutionMatrix content`": returns the content of the convolution matrix file
- "`sendfile convolutionMatrix`". To send a file to the camera
- "`restoreFactory`" restores all the convolution matrixes (in addition to other parameters, check the camera user manual for a full description)

3. Bad pixel correction

3.1. Changes with regards to filtering

The method used in the second variant of the bad pixel correction and the filtering feature is the same: correction using a user-defined convolution matrix. The previous section focused on how it applies to filtering. For bad pixel correction, only a few things change:

- The coefficient at the center of the matrix will be fixed to 0 (because it corresponds to a bad pixel, whose value we do not want).
- The correction is only applied to certain pixels, listed in the bad pixel map in the camera (see relevant sections of the camera's User Manual).
- Other bad pixels are ignored by the correction (meaning their coefficients, if they are within a convolution matrix for a bad pixel, will be fixed to 0).
- First Light Vision has a specific tool to manage the bad pixel correction. It works the same as the filtering tool.

Note: You can only use one of the two convolution corrections at the same time: either you enable filtering, and the second variant of the bad pixel correction will be unavailable, or you enable this convolution bad pixel correction, and filtering will be unavailable.

3.2. Software Development Kit

Here are the relevant commands to use to enable and configure the bad pixel convolution correction feature with FliSDK.
Please refer to the SDK's documentation (FirstLight_SDK_2_9_x_API_User_Manual_20230529.pdf) for more information.

In the FliSerialCamera, we have the following command:

- sendCommand(), to be used with the "convolutionMatrixIndex" and "set convolutionMatrixIndex X" commands to get the index of the currently loaded convolution matrix and change it, respectively.

In the FliCred class, we have the following methods:

- getUserConvolutionMatrix() and setUserConvolutionMatrix() are used to get the current convolution matrix and divisor coefficient and configure them, respectively.
- getBadPixelModeOnOff() and setBadPixelModeOnOff() are used to get whether the bad pixel correction is currently on or off and change it, respectively.
- getKindOfBadPixelCorrection() and setKindOfBadPixelCorrection() are used to get the current variant of bad pixel correction used and change it, respectively.

In the FliCredTwo and FliCredThree classes, we have the following methods:

- getBadPixelState() and enableBadPixel() to get the current state of the bad pixel correction and change it, respectively.

3.3. Command Line Interpreter commands

To change the bad pixel correction mode, use the command "`set badpixelcorrectionvariant <mode>`", where "`<mode>`" can be 1 (for the legacy, low latency correction) or 2 (for the convolution correction).
To get the current mode of the correction, use the command "`badpixelcorrectionvariant`".
To enable the correction, use the command "`set badpixel on`".
To disable the correction, use the command "`set badpixel off`".
To get the correction's status, issue the command "`badpixel`".

4. Thresholding

4.1. Introduction

The thresholding feature can be used alone or after a bad pixels and/or filtering operations.

This mode allows the camera to change the value p of pixels according to two thresholds (the low level and the high level). For each pixel:

- If its value is lower or equal to the low threshold, the pixel is set to the value "low value"
- If the pixel value is greater or equal than the high threshold, the pixel is set to the value "high value"
- If the pixel value is between the two thresholds, it is set to the value "middle value"

$$if\ p \leq Threshold\_low\_level \quad p = Low\_value$$
$$if\ p \geq Threshold\_high\_level \quad p = High\_value$$
$$if\ Threshold\_low\_level < p < Threshold\_high\_level \quad p = Middle\_value$$

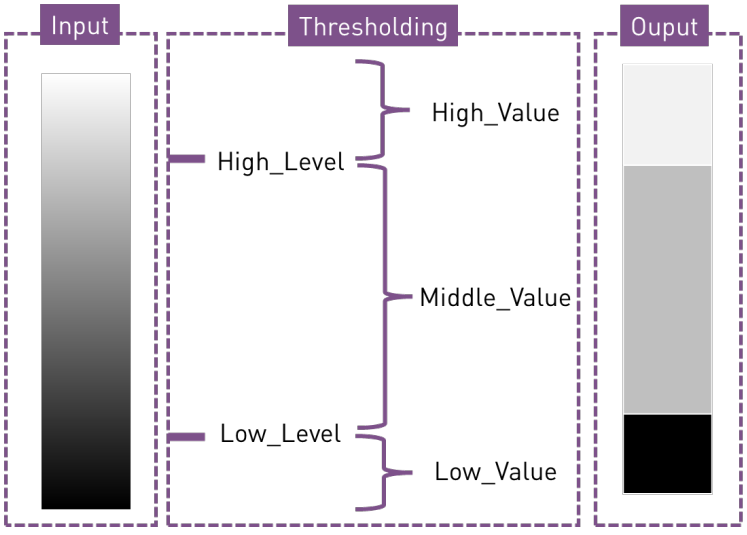


Fig. 8 : Visual representation of the thresholding operation

The thresholding operation is an interesting image preprocessing step. For instance, some applications require the automatic detection of an event, such as a fire or electric discharge for example.

Note: The output image after a thresholding operation is still encoded on 16 bits. Thresholding does not enable to achieve higher framerates.

4.2. Threshold levels and values

The thresholding operation can be activated / deactivated.
It requires three five user inputs:

- Thresholding high level
- Thresholding low level
- Thresholding high value
- Thresholding middle value
- Thresholding low value

The two threshold levels (low and high) and the three values (high, middle and low) are encoded over 16 bits. They depend on the signed/ unsigned state of the camera:

- If the unsigned state is *off*, the threshold level and values can be set anywhere between 0 and 65536.
- If the unsigned state is *on*, the most significant bit becomes a sign bit. The threshold level can then be set between -32767 included and 32767.

Note: If the unsigned state of the camera were to change when thresholding is enabled, and if current threshold levels and values are not valid values according to the unsigned state, threshold levels and values will be automatically adjusted to the closest correct value. For instance:

- A negative threshold will be changed to 0 when the unsigned state switches from *false* to *true*.
- A threshold greater than 32767 will be changed to 32767 when the unsigned state switches from *true* to *false*.

4.3. First Light Vision tutorial

The thresholding can be entirely configured using the First Light Vision interface. The parameters are available in the Control panel, as shown below.

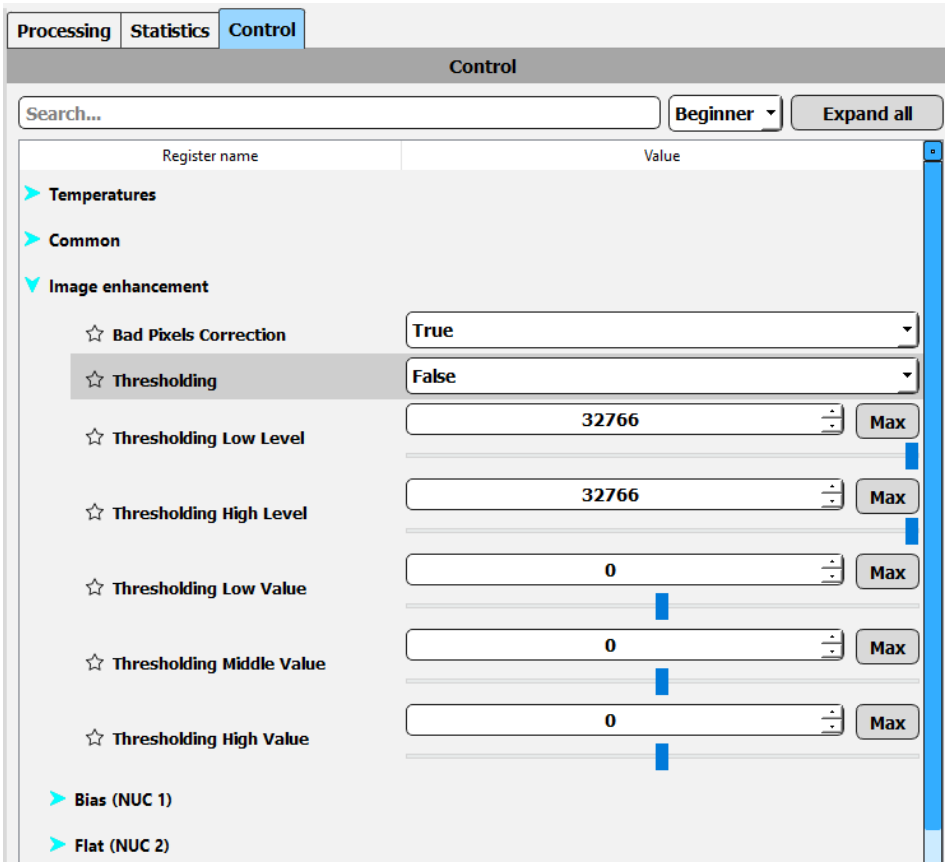


Fig. 9 : Interface for setting the thresholding parameters

4.4. Command Line Interpreter (CLI) commands

To set the various values and levels of the thresholding mode:

- "`set thresholding on`" : enable the thresholding mode.
- "`set thresholding off`" : disable the thresholding mode.
- "`set thresholding lowlevel <value>`" : sets the low threshold of the thresholding mode to <value>.
- "`set thresholding highlevel <value>`" : sets the high threshold of the thresholding mode to <value>.
- "`set thresholding lowvalue <value>`" : sets the low value of pixels in thresholding mode to <value>.
- "`set thresholding middlevalue <value>`" : sets the middle value of pixels in thresholding mode to <value>.
- "`set thresholding highvalue level <value>`" : sets the high value of pixels in thresholding mode to <value>.

To get the various values and levels of the thresholding mode:

- "`thresholding`" : returns the state of the thresholding mode, either on or off.
- "`thresholding lowlevel`" : returns the ADU value of the thresholding low level used by the camera. Can be automatically changed to fit the state of the unsigned mode.
- "`thresholding highlevel`" : returns the ADU value of the thresholding high level used by the camera. Can be automatically changed to fit the state of the unsigned mode.
- "`thresholding lowvalue`" : returns the ADU value of the thresholding low value used by the camera. Can be automatically changed to fit the state of the unsigned mode.

- "`thresholding middlevalue`" : returns the ADU value of the thresholding middle value used by the camera. Can be automatically changed to fit the state of the unsigned mode.
- "`thresholding highvalue`" : returns the ADU value of the thresholding high value used by the camera. Can be automatically changed to fit the state of the unsigned mode.

4.5. Typical workflows for image thresholding

Some typical workflow using the thresholding feature are illustrated below.

4.5.1. Two-level thresholding

It is possible to perform a simple two-level thresholding operation on the image.
Here the parameters of the thresholding are as follows:

- Thresholding high level = 6000 ADU
- Thresholding low level = 6000 ADU
- Thresholding high value = 160000 ADU
- Thresholding middle value = 0 ADU
- Thresholding low value = 0 ADU

Which translates into:

$$if\ pixelValue \leq 6000 \quad pixelValue = 0$$
$$if\ pixelValue \geq 6000 \quad pixelValue = 16000$$
$$if\ 6000 < pixelValue < 6000 \quad pixelValue = 0$$

The image below illustrates the process and the result.

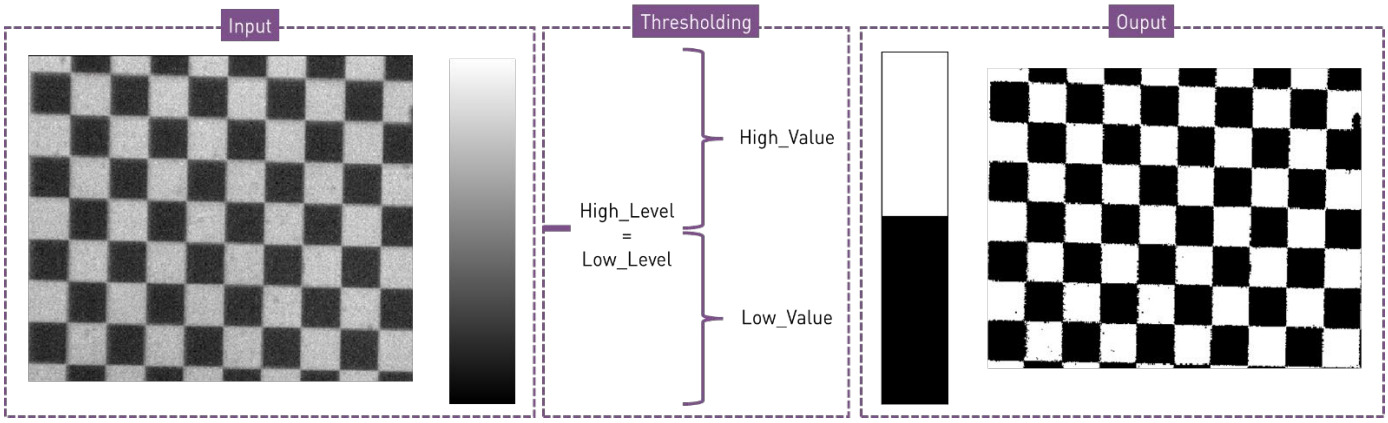


Fig. 10 : Single-step thresholding example on a checkboard

4.5.2. Three-level thresholding

It is possible to perform a three-level thresholding operation on the image. In this example a more complex target is imaged.

In a first scheme the parameters of the thresholding are as follows:

- Thresholding high level = 6000 ADU
- Thresholding low level = 2000 ADU
- Thresholding high value = 1000 ADU
- Thresholding middle value = 500 ADU
- Thresholding low value = 0 ADU

Which translates into:

$$if\ pixelValue \leq 2000 \quad pixelValue = 0$$
$$if\ pixelValue \geq 6000 \quad pixelValue = 1000$$
$$if\ 2000 < pixelValue < 6000 \quad pixelValue = 500$$

The image below illustrates the process and the result.

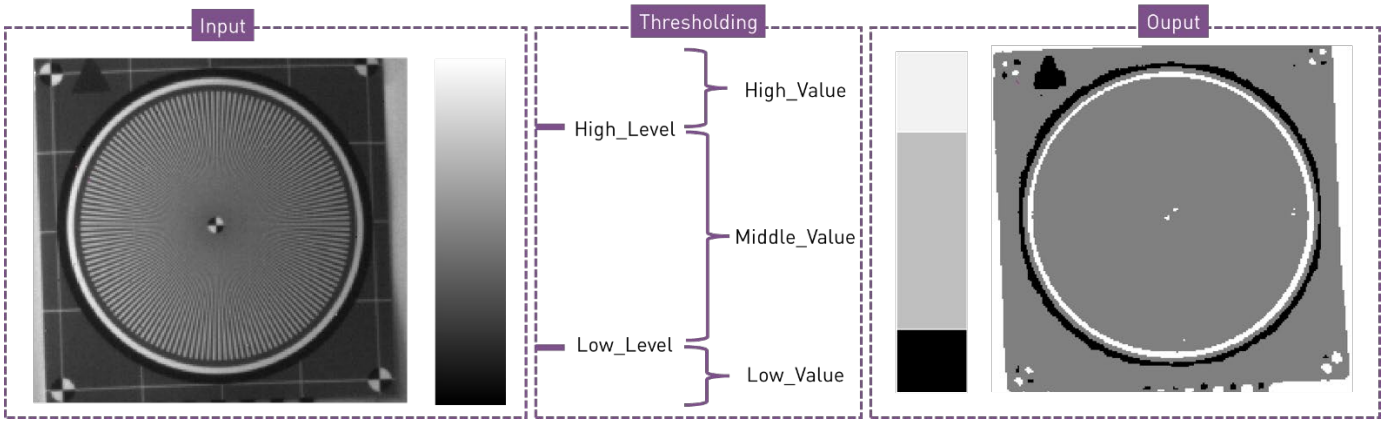


Fig. 11 : Example of a 3-level threshold

In a second scheme the high and low values are set to the same ADU number.
Here the parameters of the thresholding are as follows:

- Thresholding high level = 6000 ADU
- Thresholding low level = 2000 ADU
- Thresholding high value = 1000 ADU
- Thresholding middle value = 0 ADU
- Thresholding low value = 1000 ADU

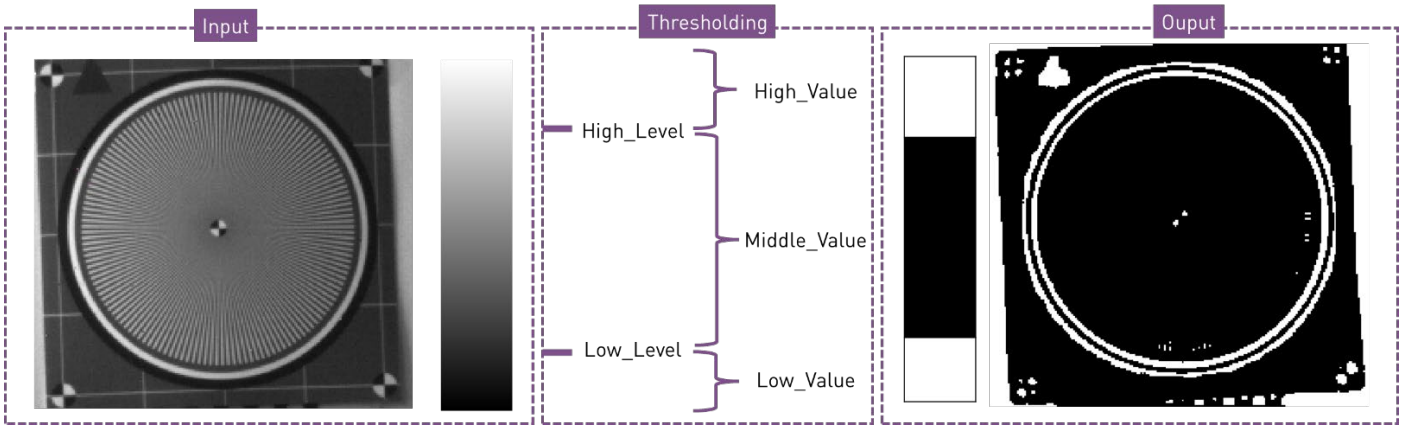


Fig. 12 : Example of a 3-level threshold

4.5.3. Thresholding combined to filtering for edge detection

The thresholding operation can be performed after image filtering.
This workflow can, for example, be used for advanced edge detection.
In the example below, the camera is imaging a resolution target. A Prewitt filter is applied to detect the horizontal edges, resulting in an image with a grey homogeneous background (no edge), white lines (where dark to light edges are detected) and dark lines (where light to dark edges are detected).

A two-level thresholding step can be used to isolate dark to light edges, as shown below.

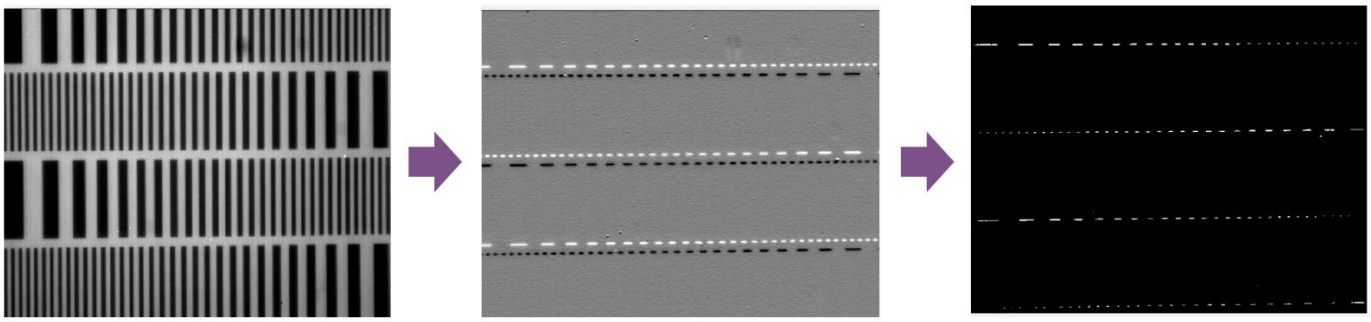


Fig. 13 : Full two-step process for oriented-edge detection : Prewitt filtering followed by thresholding

A three-level thresholding operation could have enabled the detection of all horizontal edges.

4.5.4. Large filters to improve edge detection

Because sensors are intrinsically noisy, using small 3-by-3 filters tends to result in noisy images which can lead to unproper thresholding.
Larger filters will add a filtering performance, hence reducing the noise in the image. The thresholding step will be more accurate.
Below is an illustration on a workflow designed to detect the edges of squares on a checkboard.

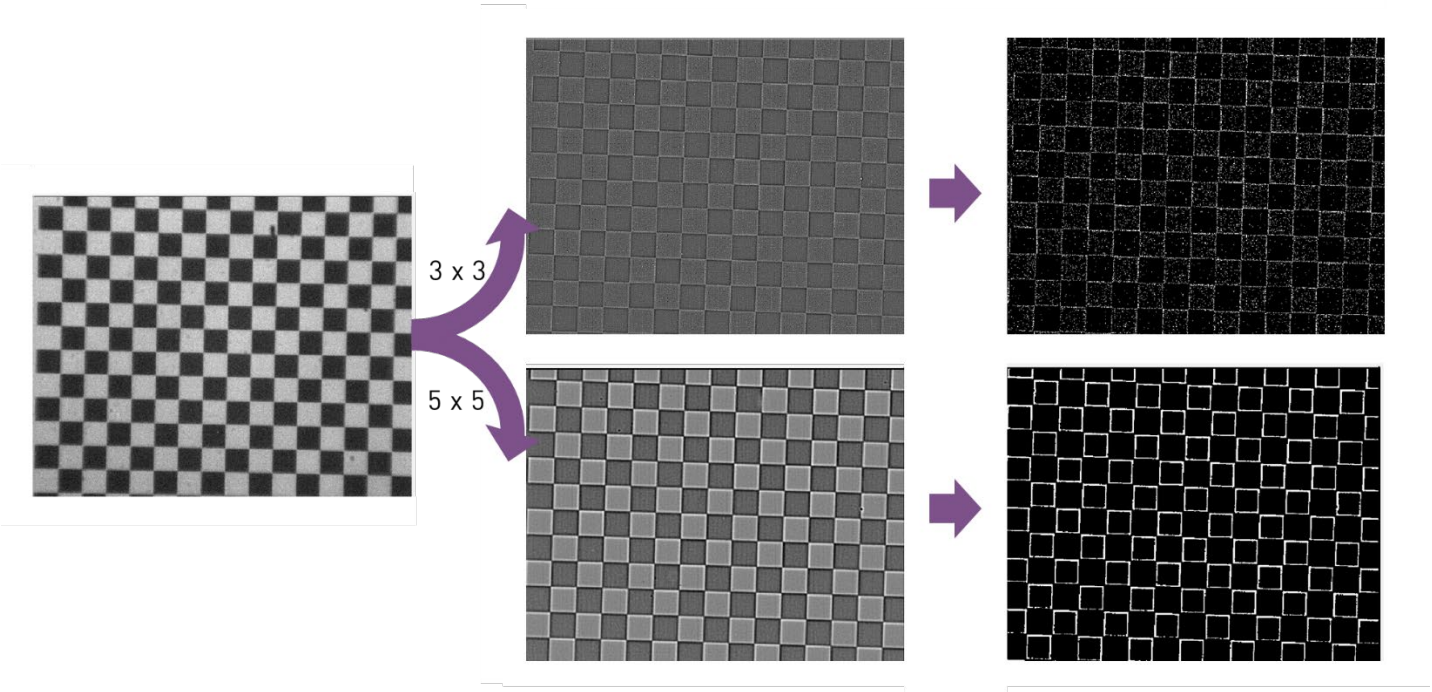


Fig. 14 : Full two-step process (convolution followed by thresholding) for edge detection using small and large filtering convolution matrix.

5. List of commands

The list of available commands of the camera can be displayed using 'help' command in the terminal of the FliVision software. The available commands for the embedded filtering and thresholding features are listed below.

COMMANDS	DESCRIPTION
`set badpixelcorrectionvariant 1`	Sets the bad pixel correction variant to the Legacy, low latency mode.
`set badpixelcorrectionvariant 2`	Sets the bad pixel correction variant to the convolution correction mode.
`badpixelcorrectionvariant`	Gets the current variant loaded in the camera.
`badpixel`	Gets the current state of the bad pixel correction, on or off.
`set badpixel on`	Sets the bad pixel correction on.
`set badpixel off`	Sets the bad pixel correction off.
`filtering`	Gets the current state of the filtering, on or off.
`set filtering on`	Sets the filtering on.
`set filtering off`	Sets the filtering off.
`excludeBorders`	Gets whether the borders are to be excluded of the convolution.
`set excludeBorders on`	Sets the border exclusion on.
`set excludeBorders off`	Sets the border exclusion off.
`recvfile convolutionMatrix size`	Gets the size of the convolution matrix file currently loaded in the camera.
`recvfile convolutionMatrix md5sum`	Gets the md5sum of the convolution matrix file currently loaded in the camera.
`recvfile convolutionMatrix content`	Gets the content of the convolution matrix file currently loaded in the camera.
`convolutionMatrixIndex`	Gets the index of the convolution matrix file currently loaded in the camera.
`set convolutionMatrixIndex X`	Sets the index of the convolution matrix file to be loaded in the camera.
`thresholding`	Gets the current state of the thresholding, on or off.
`set thresholding on`	Sets the thresholding on.
`set thresholding off`	Sets the thresholding off.
`thresholding lowlevel`	Gets the current low thresholding level
`thresholding highlevel`	Gets the current high thresholding level
`thresholding lowvalue`	Gets the current value that replaces pixels under the low thresholding level
`thresholding middlevalue`	Gets the current value that replaces pixels over the low thresholding level and under the high thresholding level
`thresholding highvalue`	Gets the current value that replaces pixels over the high thresholding level
`set thresholding lowlevel X`	Sets the low thresholding level
`set thresholding highlevel X`	Sets the high thresholding level

`set thresholding lowvalue X`	Sets the value that replaces pixels under the low thresholding level
`set thresholding middlevalue X`	Sets the value that replaces pixels over the low thresholding level and under the high thresholding level
`set thresholding highvalue X`	Sets the value that replaces pixels over the high thresholding level

To summarize, there are three kinds of commands: 'get', 'set' and 'exec'.

The 'get' commands can be used to retrieve some values from the camera, 'set' to set parameters and 'exec' to ask the camera to do a task.

For any further information, please contact First Light Imaging's support team (support@first-light.fr).