AwFrameProcessing

Revision History:

Version	*Date*	*Modifications*	*Author*
1.0.0	01-03-12	Document creation	João Santos
1.0.1	11-04-12	Minor changes in creating the dll	João Santos
1.0.2	10-07-12	Added Frames Mean to the dll	João Santos
1.0.3	12-07-12	Updated for processingCore.dll v2.0.0	João Santos
2.0.0	20-12-12	Changed name to AwFrameProcessing.dll v2.0.0	João Santos
2.1.0	14-04-14	Updated informations regarding C++/CLi and C# language	João Santos
2.2.0	24-04-14	Updated minor changes	João Santos
2.3.0	28-08-14	Updated Image Data class	João Santos
2.4.0	16-09-14	Added ProcessingWrapper, Snapshot and AwVideo	João Santos
2.5.0	09-01-15	Updated Snapshot and AwVideo	João Santos

Table of Contents

Index of Tables

Index of Figures

1 Introduction

This document explains how to use and to reference the AwFrameProcessing library in either C++/CLI and C#.
It also shows all the frame processing algorithms that are used, and how to setup all their parameters.
Then it shows how to create a Bitmap with all those algorithms applied.

2 AwFrameProcessing

2.1 Add AwFrameProcessing to your project

In this section will be described how to use this library. For that it will be used **Visual Studio 2012** to show the step-by-step of how to use this image processing library.

2.1.1 Using in C#

2.1.1.1 Adding the reference to the DLL

In C#, to add the reference, the user should click with the right-click above **References** and then choose the option **Add Reference**, as in Figure 1:

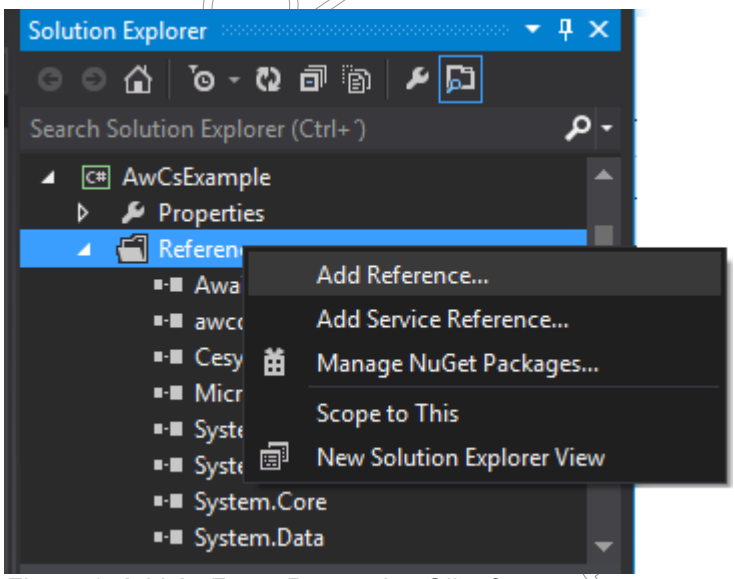

Figure 1: Add AwFrameProcessing C# reference

After this, you need to choose where the file is, as is explained in Figure 4 and 5.

2.1.2 Using in C++

When using it in c++, the user needs to choose the "Common Language Runtime Support (/clr)" option, as we can see in Figure 2. With this option active, the user is able to load .NET dll's into their project.

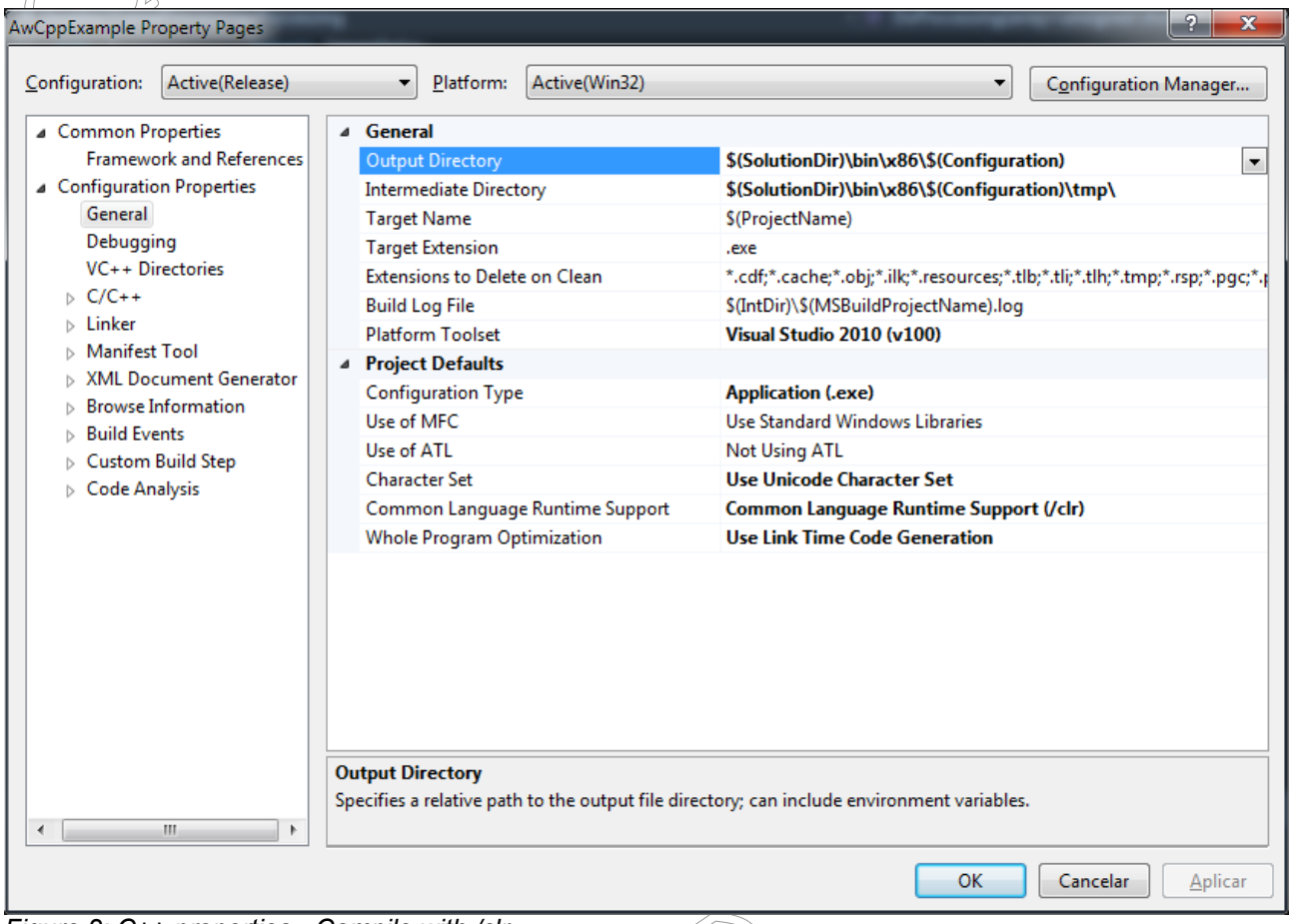

Figure 2: C++ properties - Compile with /clr

2.1.2.1 Adding the reference to the DLL

To add the AwFrameProcessing.dll reference, first you need to click on the **Framework and References** option, below the *Common Properties* option. This is marked with a "1" in Figure 3.

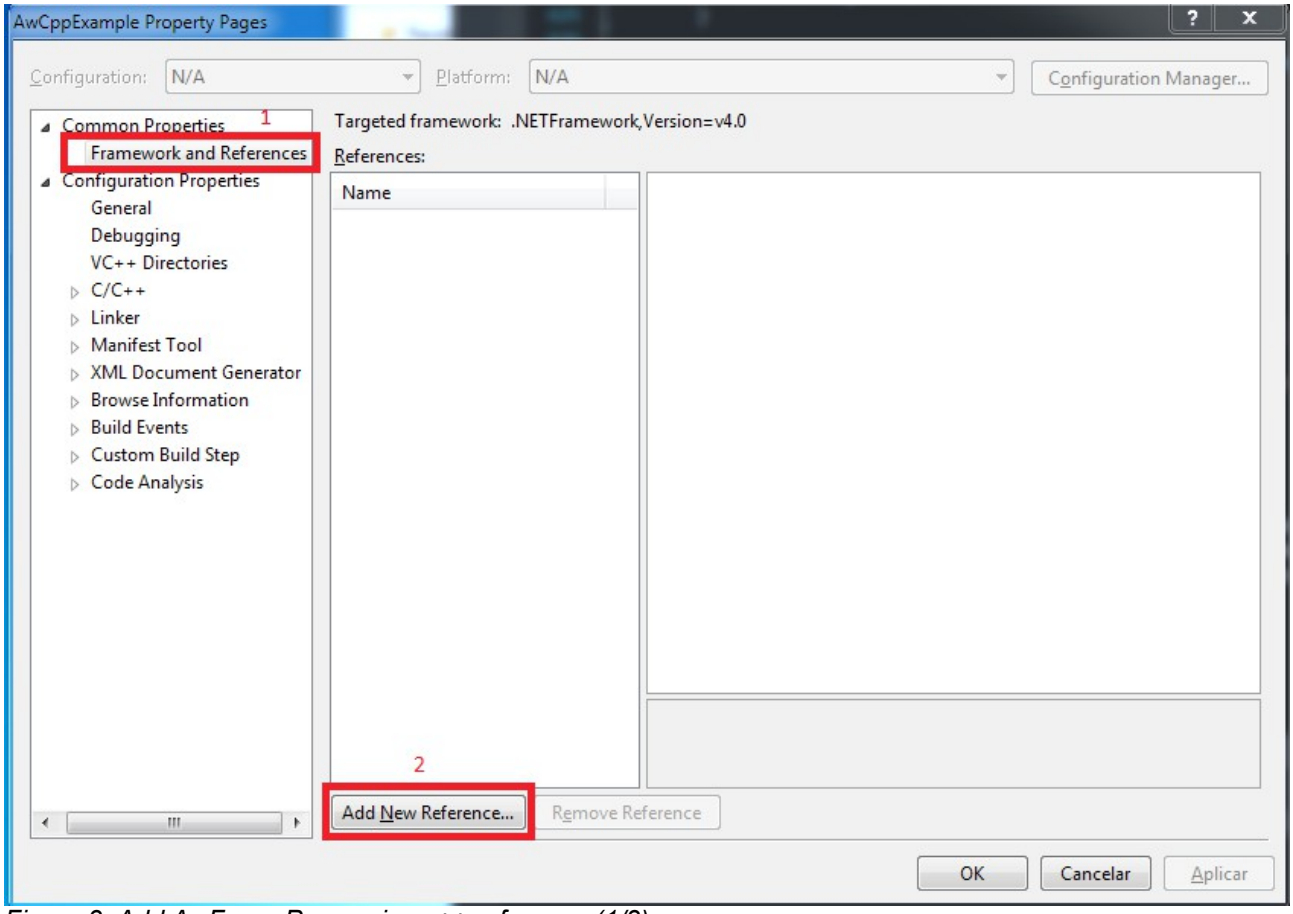

Figure 3: Add AwFrameProcessing c++ reference (1/3)

After clicking it, Figure 3 will appear. Then, you should click on **Add New Reference** (marked with a "2", in the image).

Then, Figure 4 will appear. On this image the user should click on the **browse** option (marked with a "3", allowing you to choose the **AwFrameProcessing.dll**, as in Figure 5, marked with a "5".

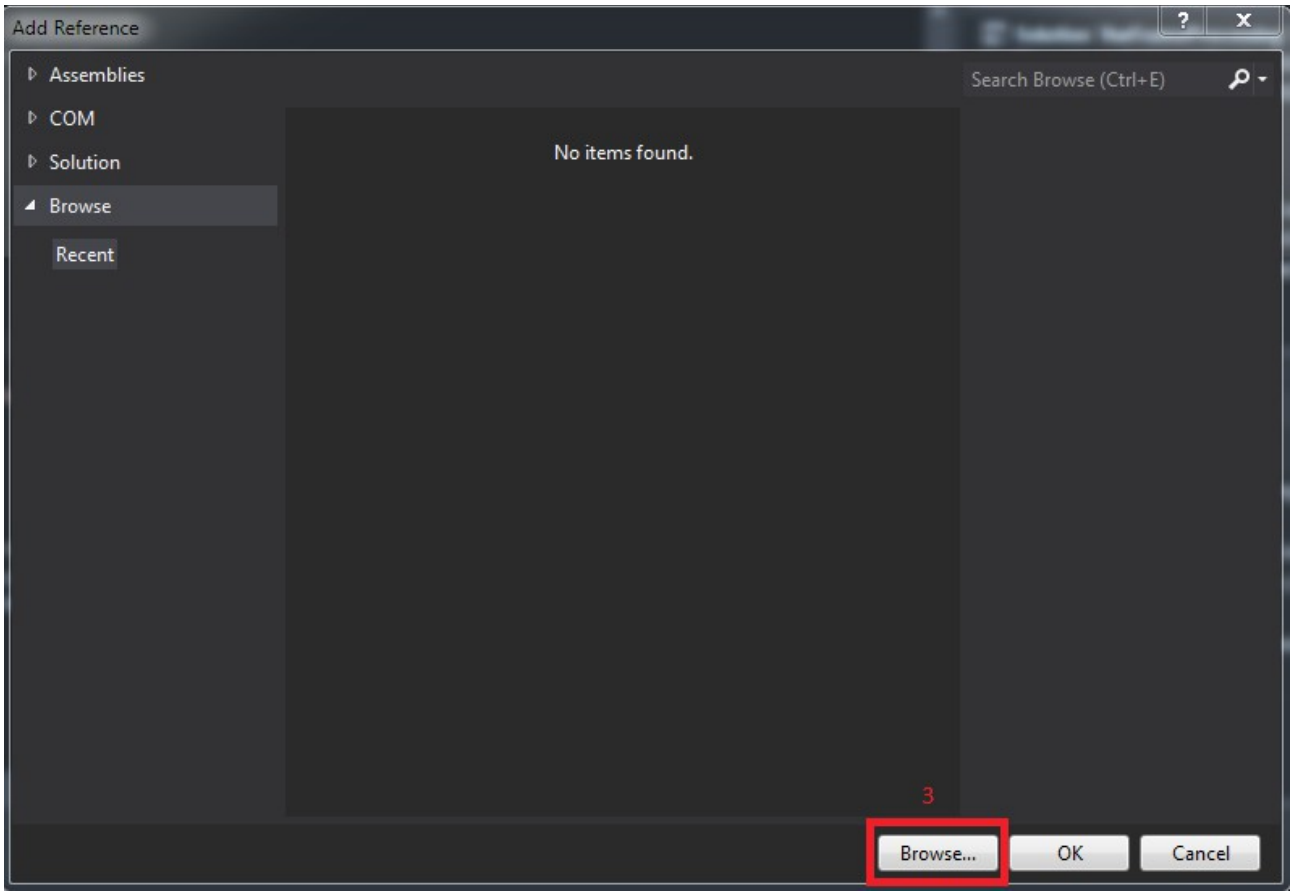

Figure 4: Add AwFrameProcessing c++ reference (2/3)

Note: If the image you are seeing, doesn't allow you to choose the **browse** option, please check that you have the /**clr** option chosen in the **Common Language Runtime Support** option, as in Figure 2.

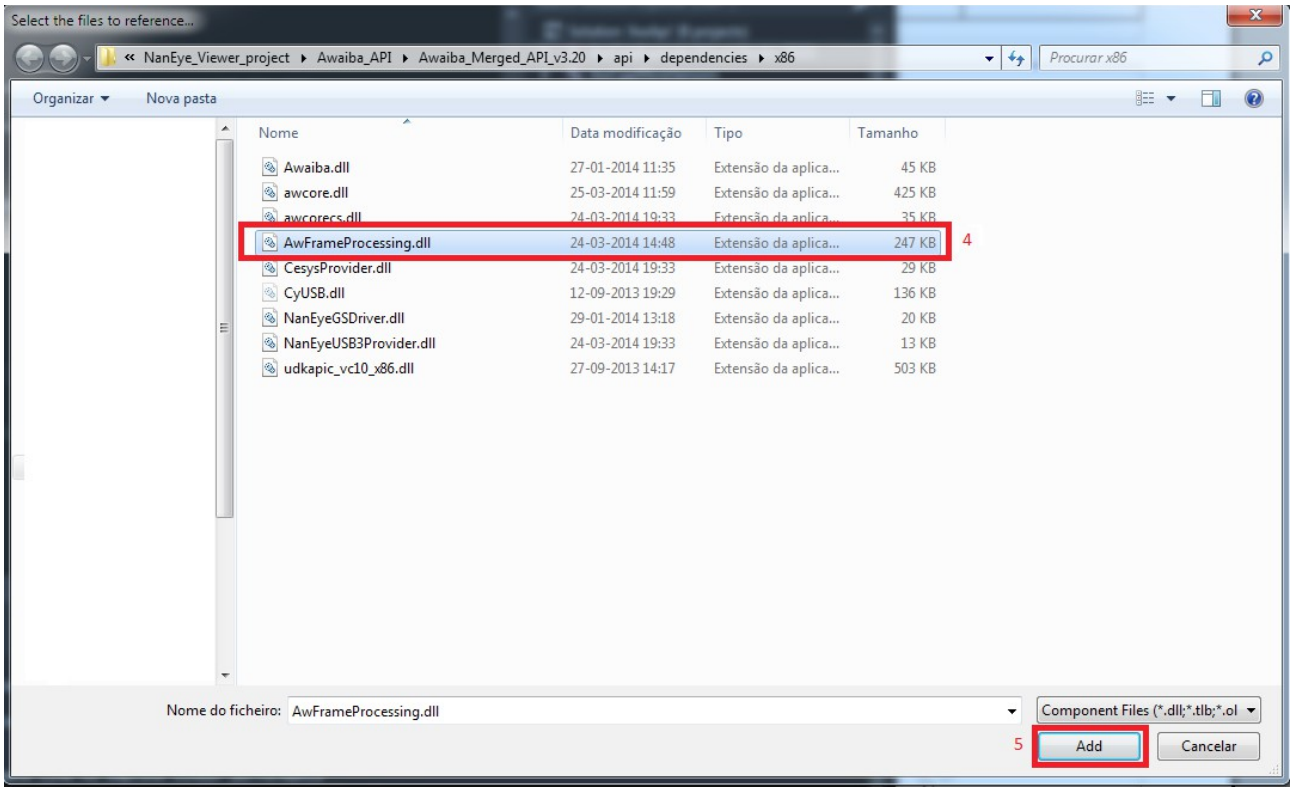

Figure 5: Add AwFrameProcessing c++ reference (3/3)

2.2 Using the AwFrameProcessing

After creating the references, in 2.1, to help in using the library, it is easier to include the reference in the your code, as in Table 1:

Language	Code
C++/CLI	`using namespace Awaiba::FrameProcessing;`
C#	`using Awaiba.FrameProcessing;`

Table 1: Using AwFrameProcessing

Note: The difference between C++/CLI and C# is that in C# we use the "." and in C++/CLI we use the "::". In the rest of the document, I will only document in C# (using ".").

We create a **ProcessingWrapper** instance, as shown in Table 2, for every sensor that is used:

Language	Code
C++/CLI	`ProcessingWrapper^ imgProc = gcnew ProcessingWrapper (NumBits, Width, Height, NumSen);`
C#	`ProcessingWrapper imgProc = new ProcessingWrapper (NumBits, Width, Height, NumSen);`

Table 2: Creating the Processing instance

When calling the constructor you need to say how many **bits** each pixel has (**NumBits**), and then the **Width** and the **Height** of the images. The **NumSen** is the number of Processing instances that will be created, one for each sensor.

Note: In C++/CLI, every class that is created, needs to use the "^" and to be created using the "**gcnew**" instead of the "new" keyword.

In case of **Nan Eye** and **NanEyeGS**, the number of **bits** is **10**.

2.3 DLLs Dependencies

In **version 1.3.2.1,** there following **DLLs** need to be in the **same folder as your application executable**:

- opencv_core249.dll
- opencv_imgproc249.dll
- aviFile.dll

3 Frame Processing

This dll allows to do apply some image processing algorithms:

- Masks – B&W
- Bad Pixel Removal
- Color Reconstruction
- Gamma Correction
- Brightness
- Frames Mean
- Skip Frames

Note: In the examples, first there are shown in both C++/CLI and C#, but after the first examples, it is either shown in C++/CLI or in C#. The difference between the two is only in using the "->" instead of the ".", the use of "gcnew" instead of "new" in the classes creation, and the use of "^" when creating a new class.

In the chapter below, every information will be done for sensor 0, using **ProcessingWrapper.Pr[0]** to access to the methods and properties.
Then you can change the sensorId according to the sensor instance that is used.

3.1 Masks – B&W

The user can use the Black & White masks to improve the image quality.

This masks are loaded using the path of the file, as shown in Table 3:

Language	Code
C++/CLI	`ProcessingWrapper::pr[0]->BlackCorrectionMask = gcnew Mask("N2CRGB4-000249.awblc");` `ProcessingWrapper::pr[0]->WhiteCorrectionMask = gcnew Mask("N2CRGB4-000249.awwlc");`
C#	`ProcessingWrapper.pr[0].BlackCorrectionMask = new Mask("N2CRGB4-000249.awblc");` `ProcessingWrapper.pr[0].WhiteCorrectionMask = new Mask("N2CRGB4-000249.awwlc");`

Table 3: Masks loading

Then, to this masks to be used, the **Apply** value has to be at **true,** as is shown in Table 4:

Language	Code
C++/CLI	`ProcessingWrapper::pr[0]->BlackCorrectionMask->Apply = true;` `ProcessingWrapper::pr[0]->WhiteCorrectionMask->Apply = true;`
C#	`ProcessingWrapper.pr[0].BlackCorrectionMask.Apply = true;` `ProcessingWrapper.pr[0].WhiteCorrectionMask.Apply = true;`

Table 4: Apply the masks

Note: On **version 1.3.2.1** of the AwFrameProcessing, the **LUT (LookUp Table) correction is not available**.

3.2 Bad Pixel Removal

This algorithm is applied when **`frameProcessing.RemoveBadPixel.Apply`** is set to **true**. The higher the value, the more homegeneous the image will look, i.e., it will be more difficult to detect the edges. Typically, this value is set to 50, as shown in Table :

Language	Code
C++/CLI	`ProcessingWrapper::pr[0]->RemoveBadPixel->Value = 50;` `ProcessingWrapper::pr[0]->RemoveBadPixel->Apply = true;`
C#	`ProcessingWrapper.pr[0].RemoveBadPixel.Apply = true;` `ProcessingWrapper.pr[0].RemoveBadPixel.Value = 50;`

Table 5: Bad Pixel Removal

3.3 Color Reconstruction

3.3.1 Color Pre-Gain:

This color pre-gain algorithm will always be applied. The user can set an array of four positions, representing the different pixels:

- `ProcessingWrapper.pr[0].colorReconstruction.pixelGain[0]` → The gain of the green1 pixel;
- `ProcessingWrapper.pr[0].colorReconstruction.pixelGain[1]` → The gain of the red pixel;
- `ProcessingWrapper.pr[0].colorReconstruction.pixelGain[2]` → The gain of the blue pixel;
- `ProcessingWrapper.pr[0].colorReconstruction.pixelGain[3]` → The gain of the green2 pixel;

The pixel's gain usually is between 0.5 and 2.

redPixelGain = 1.50;

```
ProcessingWrapper.pr[0].colorReconstruction.pixelGain[1] = 1.50f
```

3.3.2 Color

If the **Apply** option is set to true, then the color reconstruction algorithm will be applied:

```
ProcessingWrapper.pr[0].colorReconstruction.Apply = true;
```

3.3.3 Pixel Order

This variable allows to change the pixel order between:

- GBRG
- BGGR
- RGGB
- GRBG

The default value is **GBRG**, but it can be changed using the following method:

```
ProcessingWrapper.pr[0].colorReconstruction.SetBayerGrid(int id);
```

There is no need to call this function, unless a row or a column is being lost.

For Nan Eye, if the colors aren't correct, this function should be called with:

```
ProcessingWrapper.pr[0].colorReconstruction.SetBayerGrid(2);
```

3.3.4 Color Adjustment Matrix

This function allows to give a little more saturation to the colors. It is used as following:

```
ProcessingWrapper.pr[0].colorReconstruction.colorAdjustmentMatrix =
                new float[]{
                    1.2,-0.1,-0.1,
                    -0.15,1.3,-0.15,
                    -0.1,-0.1,1.2};
```

Important: The sum of each line has to be 1.0 to maintain the image's brightness. If the value is below 1.0, the image will become darker, and if it is higher than 1.0, will be brighter.

3.4 Gamma Correction

To variable **`imgProc.GammaCorrection.Apply`** can be set to **true** or **false**, applying, or not, the gamma correction algorithm.

This algorithm is applied with a certain gamma correction parameter. This parameter is set using the **`frameProcessing.GammaCorrection.Value`** variable. This value is set between 1 and 4.

```
ProcessingWrapper.pr[0].colorReconstruction.GammaCorrection.Apply=true;
ProcessingWrapper.pr[0].colorReconstruction.GammaCorrection.Value = 1.2f;
```

3.5 Brightness

The brightness value is always applied.Tipically, its value can change between 1 and 3:

```
ProcessingWrapper.pr[0].brightness.Apply = true;
ProcessingWrapper.pr[0].brightness.Value = 1.1;
```

3.6 Frames Mean

This feature allows to remove some of the image noise.
To use this, you need to configure the **ImagesToDoMean** and the **ThresholdValue** variables, and then choose the **Apply** to true.
This is shown below:

```
ProcessingWrapper.pr[0].framesMean.ImagesToDoMean = 10;
ProcessingWrapper.pr[0].framesMean.ThresholdValue = 50;
ProcessingWrapper.pr[0].framesMean.Apply = true;
```

3.7 Skip Frames

The skip frames is used in slower systems to prevent the high memory usage of the machine, and to assure that we don't have image delay.
When setting up the following line, you choose how many frames you grab, as is explained in Table 6:

Code	Explanation
`ProcessingWrapper.pr[0.skipFrames.FramesNumber=1;`	All the frames are processed and shown
`ProcessingWrapper.pr[0.skipFrames.FramesNumber=3;`	One in every Three frames is processed and shown
`ProcessingWrapper.pr[0.skipFrames.FramesNumber=5;`	One in every Five frames is processed and shown

Table 6: Skip Frames

The higher the value, the less frames per second will be processed and shown.

4 AwVideo and Snapshots

On version 1.3.2.1 of Awaiba Image Processing, it is possible to create AVI videos and take snapshots.

4.1 AwVideo

The user can create a Raw Video (with the raw data from the sensor) or a processed video (with the image after all the processing is done.

4.1.1 Start RawVideo

```
ProcessingWrapper.pr[0].awVideo.StartRawVideo(frameRate, videoFile.avi);
```

The default frame rate is **45** (for NanEye) and then the **videoFile** is where the video will be stored.

4.1.2 Start ProcessedVideo

```
ProcessingWrapper.pr[0].awVideo.StartVideo( frameRate, videoFile.avi, true);
```

The default frame rate is **45** (for NanEye) and then the **videoFile** is where the video will be stored.

4.1.3 Stop Video

```
ProcessingWrapper.pr[0].awVideo.StopVideo();
```

This command will stop and store the video.

4.2 Snapshots

As for the snapshots, the user can also save the raw image (in PGM format, that saves the whole data range of the sensor) or the processed image.

4.2.1 RawImage

```
ProcessingWrapper.pr[0].TakeSnapshost().rawImage.Save("image.pgm");
```

4.2.2 ProcessedImage

```
ProcessingWrapper.pr[0].TakeSnapshost().processedImage.Save("image.png");
```

5 NanEye - Automatic Exposure Control

For more information regarding the Automatic Exposure Control, please check the Automatic Exposure Control document, that is available in Awaiba's website.

5.1 How to use - Inside NanEyeProvider project

Assuming that you are using one of the classes that are available in the **CesysProvider.dll** (more information in the AwAPI documentation, that can be found in Awaiba's website).

```
NanEye2DNanoUSB2Provider provider = new NanEye2DNanoUSB2Provider();
```

Then, the Automatic Exposure Control can be accessed like this:

```
provider.AutomaticExpControl()
```

On section 5.3 is a list of all the settings that can be adjusted in the AEC algorithm.

5.2 How to use – Outside the NanEyeProvider project

First, you need to have the **awcorecs.dll** in your project's references.

Then, after putting the required **using** (Awaiba.Drivers.Grabbers.NanEye), you can use the constructor as the following:

```
AutomaticExposureControlHardware AEC = new AutomaticExposureControlHardware(sensorId);
```

On section 5.3 is a list of all the settings that can be adjusted in the AEC algorithm.

5.3 Automatic Exposure Control settings

The user can change some parameters to adapt the algorithms to its needs. It can increase the grey target value, the step that it uses to change the registers to get to the desired grey value, among other options.

In Figure 6, there is the Awaiba Viewer interface that allows to change the most important settings in the “General” tab, and then other settings in the “Expert Registers”.

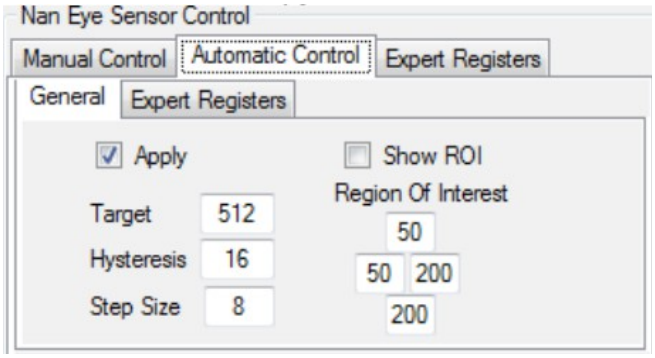

Figure 6: NanEye Automatic Exposure Control

Below is a description of the most important commands that can be used in the automatic exposure control algorithm:

```
AEC.Enabled = 1; //1->Will enable the AEC algorithm; 0->Disable
AEC.ShowROI = 1; //Will show the interest region where the AEC is used
AEC.TargetGreyValue = 500; //Value that the algorithm will try to get, by changing the
sensor's registers
AEC.TopROI = 50; //Used to define the region of intereset (Top)
AEC.BottomROI = 200; //Used to define the region of intereset (Bottom)
AEC.LeftROI = 50; //Used to define the region of intereset (Left)
AEC.RightROI = 200; //Used to define the region of intereset (Right)
```