constant robotics

Polar Processor C++ library v1.0.0

programmer's manual

www.constantrobotics.com

CONTENTS

DOCUMENT VERSIONS ... 3
LIBRARY VERSIONS ... 3
DESCRIPTION ... 3
LIBRARY FILES ... 3
KEY FEATURES AND CAPABILITIES ... 4
LIBRARY PRINCIPLE ... 5
- GENERAL PRINCIPLE OF OPERATION ... 5
- FULL INTENSITY IMAGE ... 5
- GLARE REMOVAL ... 5
- VIRTUAL LINEAR POLARIZATION FILTER ... 6
- DEGREE OF LINEAR POLARIZATION (DOLP) ... 6

LIBRARY USE ... 7
- LIBRARY PRINCIPLES ... 7
- IMAGE PROCESSING CLASS DESCRIPTION ... 7
  - Declaration of the PolarProcessorApi class ... 7
  - removeGlare(…) method ... 8
  - totalIntensity(…) method ... 8
  - virtualPolarizer(…) method ... 9
  - DoLP(…) method ... 9
  - getVersion() method ... 10

EXAMPLES OF HOW TO USE THE LIBRARY ... 10
- GLARE REMOVAL EXAMPLE ... 10
- EXPLANATIONS TO THE GLARE REMOVAL EXAMPLE ... 12
- EXAMPLE OF OBTAINING AN IMAGE WITH AN ANY POLARIZATION ANGLE ... 13
- EXPLANATIONS TO THE VIRTUAL POLARIZER EXAMPLE ... 15
- EXAMPLE OF DEGREE OF LINEAR POLARIZATION (DOLP) ... 16
- EXPLANATIONS TO THE LINEAR POLARIZATION DEGREE EXAMPLE ... 17

DOCUMENT VERSIONS

Table 1 – Document versions.

Version	Release date	What is new?
1.0.0	23.06.2022	Programmer's manual for v1.0.0 of the Polar Processor C++ library.

LIBRARY VERSIONS

Table 2 – Library versions.

Version	Release date	What is new?
1.0.0	23.06.2022	The first version of the Polar Processor C++ library. Implemented the functions: 1. Automatic glare removal. 2. Virtual linear polarizing filter. 3. Calculation of the full intensity image. 4. Calculation of the linear polarization degree.

DESCRIPTION

C++ library **Polar Processor** version **1.0.0** (library) is intended to process images acquired by polarization sensors (for example, Sony IMX250MZR/MYR, Sony IMX264MZR/MYR, Sony IMX253MZR/MYR) and implements algorithms of automatic glare removal, virtual linear polarization filter, full intensity image calculation and calculation of linear polarization degree. The library allows to process video from polarization cameras in real time. The library is written in C++ (C++11 standard). The library does not use any third-party code and does not include other libraries. The library is compatible with any processors and operating systems that support the C++ compiler (C++11 standard). The library contains a description of the **PolarProcessorApi** class whose methods are used for image processing. The library does not perform any background calculations. All calculations are performed by calling the corresponding method of the PolarProcessorApi class and end by returning control to the main thread.

LIBRARY FILES

Table 3 – Source code files of the library.

File	Description
PolarProcessor.h	Header file containing a description of the PolarProcessor C++ class.
PolarProcessorApi.h	Header file containing a description of the PolarProcessorApi C++ class.
PolarProcessorVersion.h	A header file containing a description of the library version.
PolarProcessor.cpp	PolarProcessor C++ class methods implementation file.
PolarProcessorApi.cpp	PolarProcessorApi C++ class methods implementation file.

Table 4 – Library files when delivered to the user without source code.

File	Description
PolarProcessorApi.h	Header file containing a description of the PolarProcessorApi C++ class.
PolarProcessorVersion.h	A header file containing a description of the library version.
PolarProcessorApi.lib (Window OS) или PolarProcessorApi.a (Linux OS)	Static library file for the Windows or Linux operating systems.

KEY FEATURES AND CAPABILITIES

Table 5 – Key features and capabilities of the library.

Parameter	Value and description
Library language.	C++ (C++11 standard).
Compatibility with operating systems.	Compatible with any operating system that supports the C++ compiler (C++11 standard).
Allocated memory.	No more than 4 MB of statically allocated memory. The library does not use dynamically allocated memory.
Types of images to be processed.	Grayscale RAW images with **0°**, **45°**, **90°** and **135°** polarization filters (2x2 pixel group): macro pixel 90° 45° 90° 45° 90° 45° 90° 45° 135° 0° 135° 0° 135° 0° 135° 0° 90° 45° 90° 45° 90° 45° 90° 45° 135° 0° 135° 0° 135° 0° 135° 0° 90° 45° 90° 45° 90° 45° 90° 45° 135° 0° 135° 0° 135° 0° 135° 0° 90° 45° 90° 45° 90° 45° 90° 45° 135° 0° 135° 0° 135° 0° 135° 0° Bayer RGGB color RAW images with **0°**, **45°**, **90°** and **135°** polarization filters (4x4 pixel group): macro pixel 90° 45° 90° 45° 90° 45° 90° 45° 135° 0° 135° 0° 135° 0° 135° 0° 90° 45° 90° 45° 90° 45° 90° 45° 135° 0° 135° 0° 135° 0° 135° 0° 90° 45° 90° 45° 90° 45° 90° 45° 135° 0° 135° 0° 135° 0° 135° 0° 90° 45° 90° 45° 90° 45° 90° 45° 135° 0° 135° 0° 135° 0° 135° 0° Any number, arrangement and angle of polarization filters can be prepared on request. **Note:** Orientation of the filters does not indicate the transmitted polarization orientation.
Minimum and maximum image sizes.	The minimum image size is 2x2 pixels. There is no limit on the maximum size of the input images. The width and height of the input images must be a multiple of 2. Width and height of the output images are half the size of the input images.
Speed of calculations.	Calculation speed depends on the chosen method of image processing, as well as on the size of the image. Calculation time is directly proportional to the increase in image size. The library does not perform any background tasks. The library performs calculations only within the limits of the corresponding PolarProcessorApi C++ class method call. Test applications are

Parameter	Value and description
	provided to evaluate the performance on a particular hardware platform.

Note: the given parameter values are applied to the concept of video frame(s) and the concept of pixel.

LIBRARY PRINCIPLE

GENERAL PRINCIPLE OF OPERATION

Library is based on approximation of optical radiation polarization values for a group of pixels with known polarization filter angles (pixel polarizers). The standard version of the library is designed for matrices, with a group of pixel polarizers of 0°, 45°, 90° and 135°. In case conventional matrices with custom pixel polarizers with any polarization angle are used, the library version with the required configuration of pixel polarizers is available on request.

FULL INTENSITY IMAGE

A full-intensity image is an image in which the total intensity is calculated for each group of pixel polarizers, as if the group were a single pixel and no polarizing filters were present. Such an image shows what an image would look like with a conventional sensor. This image is intended for general visualization of polarization images, since viewing images of individual polarizations may not be informative or distorted (e.g., when light passes through transparent materials). The full intensity image is used for comparison when removing glare. Figure 1 shows the example of the full intensity image.

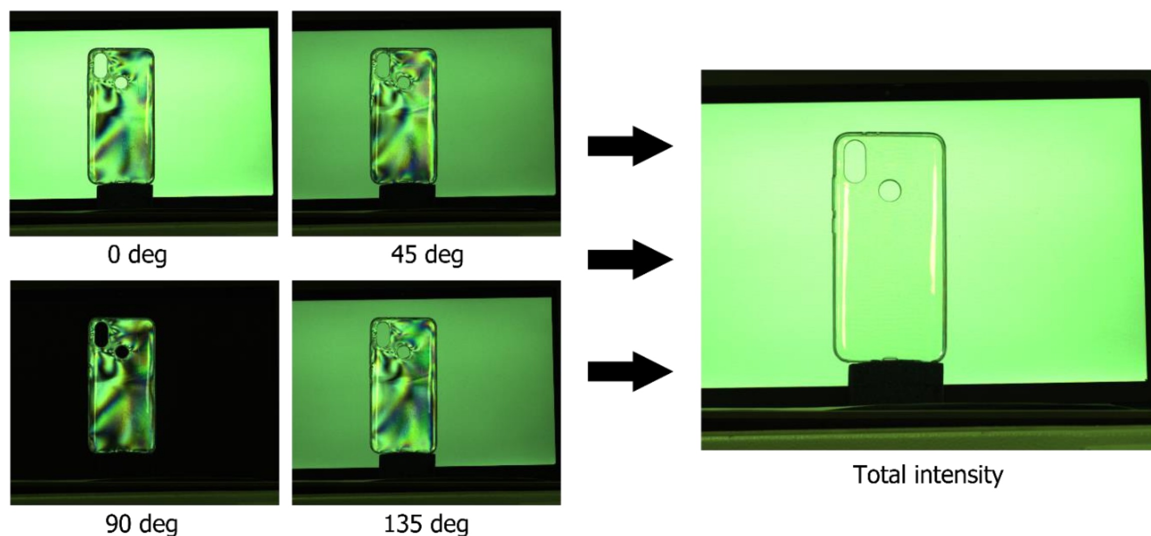


Figure 1 – Principle of making full intensity image.

GLARE REMOVAL

An image with removed glare is an image in which an optimal minimum of intensity is found for each group of pixels with polarization filters. The algorithm provides a deeper minimization compared to simply selecting the minimum value from the 4 polarization channels. After glare filtering, the image may be quite dark, so the library has the ability to perform gamma correction after glare removal. Figure 2 shows the image with the glare removed.

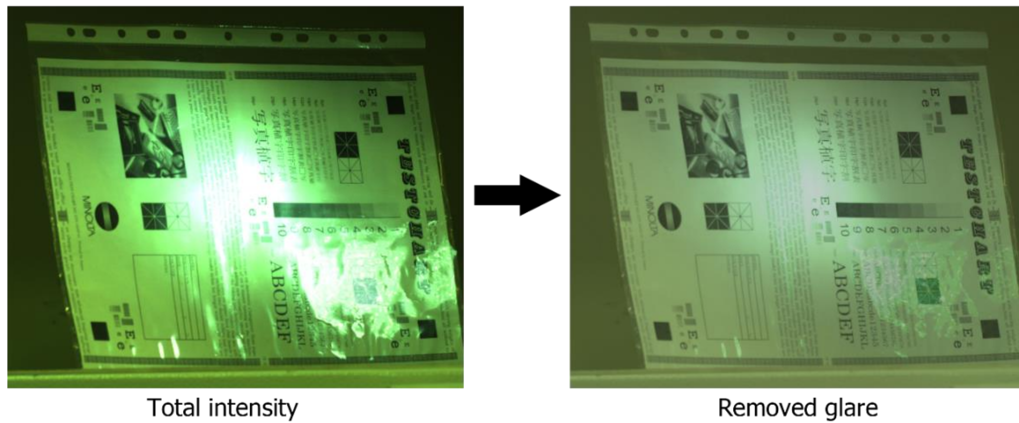


Figure 2 – Example of glare removal.

VIRTUAL LINEAR POLARIZATION FILTER

Virtual linear polarization filter is designed to obtain an image similar to the image with a physically installed linear polarization filter. Virtual polarizing filter is convenient to use in case it is necessary to dynamically change the installation angle or to obtain an image taking into account the polarizing filter when it is physically absent. The angle can be fine-tuned, as it can be set by a floating-point number. Figure 3 shows an example of a virtual linear polarizer.

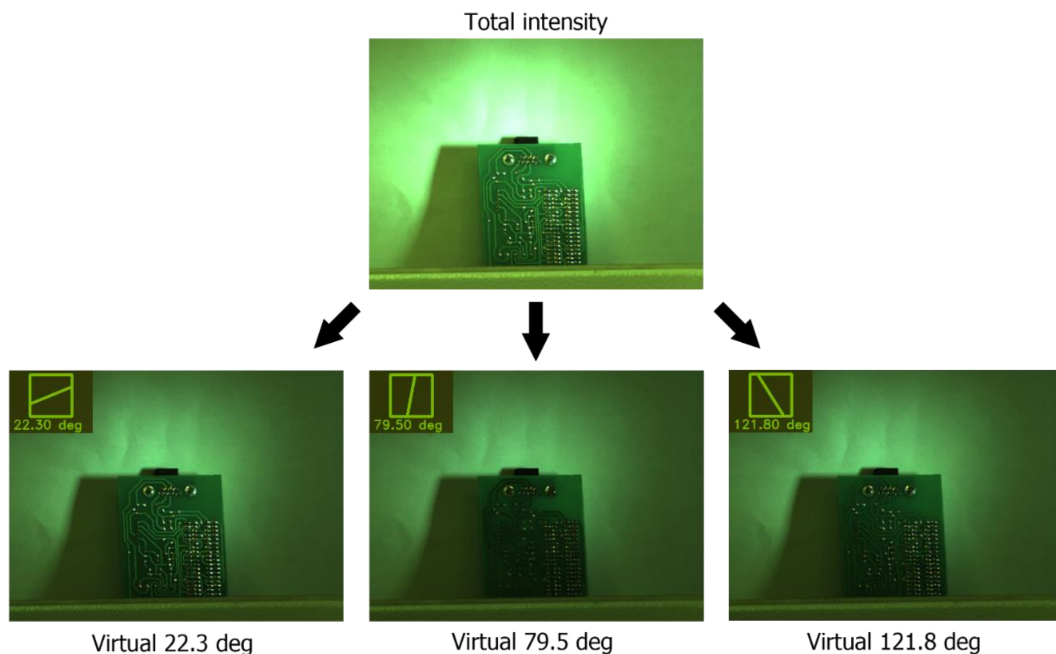


Figure 3 – Example of the virtual linear polarizer.

DEGREE OF LINEAR POLARIZATION (DOLP)

The Degree of Linear Polarization (DoLP) is an image in which the degree of linear polarization component is found for each group of pixels with polarization filters. DoLP reflects what proportion of the radiation has a linear polarization character with respect to the whole radiation. Figure 4 shows the example of DoLP image.

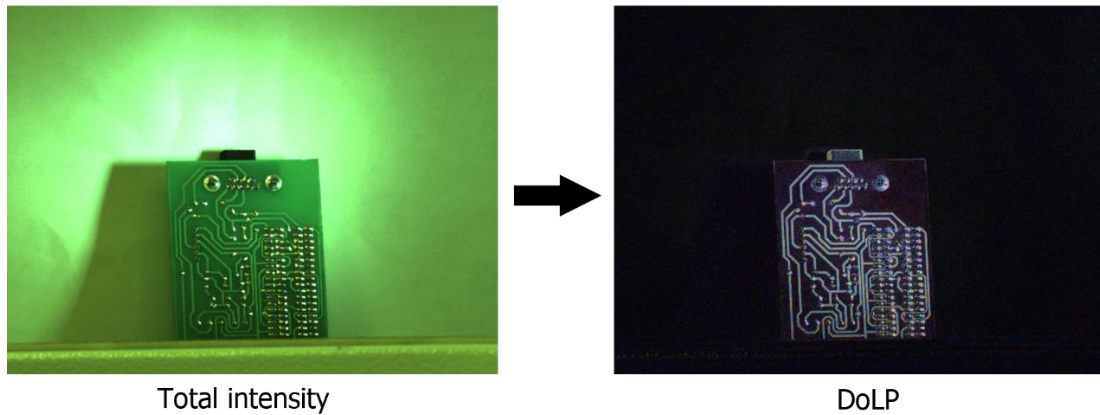


Figure 4 – Example of DoLP image.

LIBRARY USE

LIBRARY PRINCIPLES

The library consists of only 5 source code files: PolarProcessor.h, PolarProcessorApi.h, PolarProcessorVersion.h, PolarProcessor.cpp, PolarProcessorApi.cpp. Compiled version of the library includes only 3 files: PolarProcessorApi.h, PolarProcessorVersion.h, PolarProcessorApi.lib (Windows OS) or PolarProcessorApi.a (Linux OS). To use the library, the developer must include these files in his C++ project. The PolarProcessorApi.h file contains the declaration of the PolarProcessorApi C++ class, which implements the image processing algorithms. The sequence of using the library is as follows:

1. Connect the library files to your project.
2. Create an instance of the PolarProcessorApi class.
3. To process the image call the method corresponding to the required processing algorithm.

IMAGE PROCESSING CLASS DESCRIPTION

Declaration of the PolarProcessorApi class

The PolarProcessorApi class is declared in the PolarProcessorApi.h file. The declaration of the PolarProcessorApi class is shown below.

```
namespace plib
{
   class PolarProcessorApi
   {
   public:

      PolarProcessorApi();

      ~PolarProcessorApi();

      static std::string getVersion();

      bool removeGlare(uint8_t* src, uint8_t* dst, int width, int height, float gamma = -1);
```

```
        bool totalIntensity(uint8_t* src, uint8_t* dst, int width, int height);

        bool virtualPolarizer(uint8_t* src, uint8_t* dst, int width, int height, float angle);

        bool DoLP(uint8_t* src, uint8_t* dst, int width, int height);
    };
}
```

Table 6 – PolarProcessorApi class methods.

Method	Description
PolarProcessorApi()	Class constructor.
~PolarProcessorApi()	Class destructor.
getVersion()	Static method for getting the library version string.
removeGlare(…)	Glare removal method.
totalIntensity(…)	Method to get full intensity image.
virtualPolarizer(…)	Virtual polarizer method.
DoLP(…)	Method to get image of linear polarization degree.

removeGlare(…) method

The removeGlare(...) method is designed to get an image with glare removed. The declaration of the method is given below.

```
bool removeGlare(uint8_t* src, uint8_t* dst, int width, int height, float gamma = -1)
```

Parameters:

src	The pointer to the input image data is a one-dimensional array of bytes sized **width * height**. RAW grayscale or RAW Bayer RGGB data format.
dst	The pointer to the resulting image data is a one-dimensional byte array of size **(width * height)/4** (the size of the resulting image is half the size of the input image). RAW grayscale or RAW Bayer RGGB data format.
width	The width of the input image. Must be both greater than 2 and a multiple of 2.
height	The height of the input image. Must be both greater than 2 and a multiple of 2.
gamma	Gamma correction coefficient. The default value is -1, which means no gamma correction. If the value is >=0, the method will perform gamma correction. The gamma correction coefficient determines the degree of non-linearity of the color correction. A value of 1 does not affect the change in brightness. Values less than 1 lighten the image (with a value of 0, the image is completely white). Values greater than 1 make the image darker (no upper limit, but values greater than 10 make the image almost completely black). Typical value is 0.45 (1/2.2).

Return value:
The method returns **TRUE** if the input image satisfies the required conditions (image size is greater than 2x2 and both width and height are multiples of 2) and the calculations are successful. The method returns **FALSE** if the size requirements of the input image are violated or there are problems in the process of calculation.

totalIntensity(…) method

The totalIntensity(...) method is intended for getting an image of total intensity. The declaration of the method is given below.

```
bool totalIntensity(uint8_t* src, uint8_t* dst, int width, int height)
```

Parameters:

src	The pointer to the input image data is a one-dimensional array of bytes sized **width * height**. RAW grayscale or RAW Bayer RGGB data format.
dst	The pointer to the resulting image data is a one-dimensional byte array of size **(width * height)/4** (the size of the resulting image is half the size of the input image). RAW grayscale or RAW Bayer RGGB data format.
width	The width of the input image. Must be both greater than 2 and a multiple of 2.
height	The height of the input image. Must be both greater than 2 and a multiple of 2.

Return value:
The method returns **TRUE** if the input image satisfies the required conditions (image size is greater than 2x2 and both width and height are multiples of 2) and the calculations are successful. The method returns **FALSE** if the size requirements of the input image are violated or there are problems in the process of calculation.

virtualPolarizer(…) method

The virtualPolirizer(...) method is designed to produce an image with an any polarization angle. The declaration of the method is given below.

```
bool virtualPolarizer(uint8_t* src, uint8_t* dst, int width, int height, float angle)
```

Parameters:

src	The pointer to the input image data is a one-dimensional array of bytes sized **width * height**. RAW grayscale or RAW Bayer RGGB data format.
dst	The pointer to the resulting image data is a one-dimensional byte array of size **(width * height)/4** (the size of the resulting image is half the size of the input image). RAW grayscale or RAW Bayer RGGB data format.
width	The width of the input image. Must be both greater than 2 and a multiple of 2.
height	The height of the input image. Must be both greater than 2 and a multiple of 2.
angle	The dedicated angle of the virtual polarizer, relative to the pixel with 0° polarization. Angle in radians. The angle can be specified by any value (including negative values), during calculations the angle value is automatically reduced to the range [0, 2π].

Return value:
The method returns **TRUE** if the input image satisfies the required conditions (image size is greater than 2x2 and both width and height are multiples of 2) and the calculations are successful. The method returns **FALSE** if the size requirements of the input image are violated or there are problems in the process of calculation.

DoLP(…) method

The DoLP(...) method is designed to produce an image of the degree of linear polarization. The declaration of the method is given below.

```
bool DoLP(uint8_t* src, uint8_t* dst, int width, int height)
```

Parameters:

src	The pointer to the input image data is a one-dimensional array of bytes sized **width * height**. RAW grayscale or RAW Bayer RGGB data format.
dst	The pointer to the resulting image data is a one-dimensional byte array of size **(width * height)/4** (the size of the resulting image is half the size of the input image). RAW grayscale or RAW Bayer RGGB data format.
width	The width of the input image. Must be both greater than 2 and a multiple of 2.
height	The height of the input image. Must be both greater than 2 and a multiple of 2.

Return value:
The method returns **TRUE** if the input image satisfies the required conditions (image size is greater than 2x2 and both width and height are multiples of 2) and the calculations are successful. The method returns **FALSE** if the size requirements of the input image are violated or there are problems in the process of calculation.

getVersion() method

The static getVersion() method is designed to get the string of the current library version. The declaration of the method is given below.

```
static std::string getVersion()
```

Return value:
The method returns the string of the current software library version in the format "**1.0.0**".

The method can be called without creating an object of the PolarProcessorApi class, as shown below.

```
std::cout << plib::PolarProcessorApi::getVersion();
```

EXAMPLES OF HOW TO USE THE LIBRARY

This section contains the source code of simple demo applications for testing the library. The applications use the open source library OpenCV to read images from a file and visualize the results.

GLARE REMOVAL EXAMPLE

The example opens the image file and then successively invokes the full intensity image, glare removal, and gamma-correction. The full intensity image demonstrates what an image with glare looks like.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include "PolarProcessorApi.h"
```

```
int main(void)
{
    std::cout << "=====================================" << std::endl;
    std::cout << "PolarProcessor v" << plib::PolarProcessorApi::getVersion() << std::endl;
    std::cout << "Glare removal example" << std::endl;
    std::cout << "=====================================" << std::endl;

    // Haight of polirized RGGB image.
    int height = 2048;
    // Width of polarized RGGB image.
    int width = 2448;
    // Polar processor object.
    plib::PolarProcessorApi pProcessor;

    // Read polarized RGGB image.
    cv::Mat inputImg = cv::imread(
        "img\\example_silicon_case.bmp", cv::IMREAD_GRAYSCALE);

    // Make total intencity image (size / 2 of source image).
    cv::Mat totalIntensityImg =
        cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
    pProcessor.totalIntensity(
        inputImg.data, totalIntensityImg.data, width, height);

    // Converting from RAW (RGGB) to RGB.
    cv::Mat totalIntensityRGB(height / 2, width / 2, CV_8UC3);
    cv::cvtColor(totalIntensityImg, totalIntensityRGB, cv::COLOR_BayerRG2RGB);

    // Show image
    namedWindow("Total intensity image RGB", cv::WINDOW_KEEPRATIO);
    cv::imshow("Total intensity image RGB", totalIntensityRGB);

    // Make image with removed glare (size / 2 of source image).
    cv::Mat removedGlareImg = cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
    pProcessor.removeGlare(inputImg.data, removedGlareImg.data, width, height);

    // Converting from RAW (RGGB) to RGB
    cv::Mat removedGlareRGB(height / 2, width / 2, CV_8UC3);
    cvtColor(removedGlareImg, removedGlareRGB, cv::COLOR_BayerRG2RGB);

    // Show image
    namedWindow("Removed glare RGB", cv::WINDOW_KEEPRATIO);
    cv::imshow("Removed glare RGB", removedGlareRGB);

    // Make image with removed glare ang gamma correction (size / 2 of source image)
    cv::Mat removedGlareGammaImg =
        cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
    pProcessor.removeGlare(inputImg.data,
        removedGlareGammaImg.data, width, height, 0.45);

    // Converting from RAW (RGGB) to RGB
    cv::Mat removedGlareGammaRGB(height / 2, width / 2, CV_8UC3);
    cvtColor(removedGlareGammaImg, removedGlareGammaRGB, cv::COLOR_BayerRG2RGB);

    // Show image
    namedWindow("Removed glare RGB with gamma correction", cv::WINDOW_KEEPRATIO);
```

```
    imshow("Removed glare RGB with gamma correction", removedGlareGammaRGB);
    cv::waitKey();

    return 0;
}
```

Figure 5 – The result of the application (left full intensity image, middle image with glare removed, right image with glare removed and gamma correction).

EXPLANATIONS TO THE GLARE REMOVAL EXAMPLE

The application requires the OpenCV library header files and the Polar Processor library to be connected.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include "PolarProcessorApi.h"
```

At the beginning of the application the library version and information about the demonstration program are displayed.

```
std::cout << "====================================" << std::endl;
std::cout << "PolarProcessor v" << plib::PolarProcessorApi::getVersion() << std::endl;
std::cout << "Glare removal example" << std::endl;
std::cout << "====================================" << std::endl;
```

Next, the variables for the dimensions of the demo image are declared, and an instance of the PolarProcessorApi object is created:

```
int height = 2048;
int width = 2448;
plib::PolarProcessorApi pProcessor;
```

After that the source image file is read.

```
cv::Mat inputImg = cv::imread("img\\example_silicon_case.bmp", cv::IMREAD_GRAYSCALE);
```

Then an empty image is created to store the result image of full intensity. The size of the result image is 2 times smaller than the original image. The totalIntensity(...) method is called to get the full intensity image.

```
cv::Mat totalIntensityImg = cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
pProcessor.totalIntensity(inputImg.data, totalIntensityImg.data, width, height);
```

The resulting image is converted from RGGB to RGB and displayed.

```
cv::Mat totalIntensityRGB(height / 2, width / 2, CV_8UC3);
cv::cvtColor(totalIntensityImg, totalIntensityRGB, cv::COLOR_BayerRG2RGB);

namedWindow("Total intensity image RGB", cv::WINDOW_KEEPRATIO);
cv::imshow("Total intensity image RGB", totalIntensityRGB);
```

Then the same steps are performed to obtain an image with the glare removed. An empty image is created to store the image with the glare removed. The size of the processed image is half the size of the original image. The removeGlare(...) method is called to get the image with the glare removed.

```
cv::Mat removedGlareImg = cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
pProcessor.removeGlare(inputImg.data, removedGlareImg.data, width, height);

cv::Mat removedGlareRGB(height / 2, width / 2, CV_8UC3);
cvtColor(removedGlareImg, removedGlareRGB, cv::COLOR_BayerRG2RGB);

namedWindow("Removed glare RGB", cv::WINDOW_KEEPRATIO);
cv::imshow("Removed glare RGB", removedGlareRGB);
```

The last step demonstrates removing glare with gamma correction. An empty image is created to store the image with glare removal and gamma correction. The size of the processed image is twice less than the original image. The removeGlare(...) method with specifying the gamma correction coefficient is called to obtain an image with removed glare.

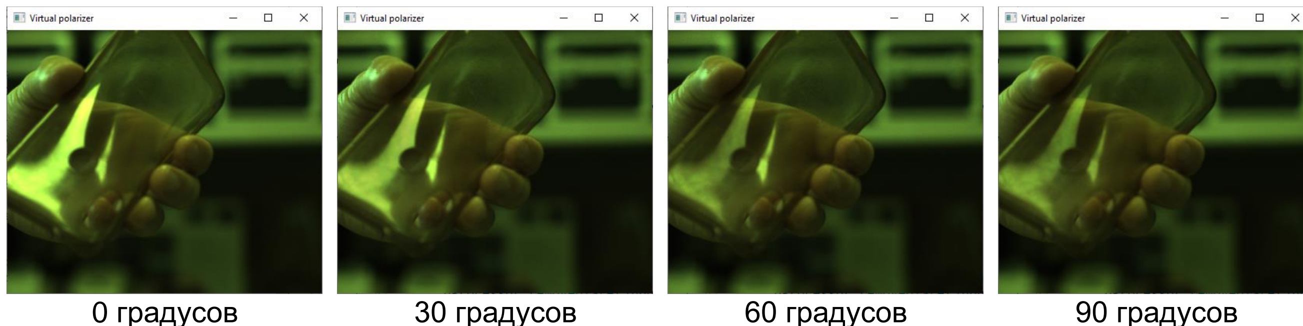
```
cv::Mat removedGlareGammaImg = cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
pProcessor.removeGlare(inputImg.data, removedGlareGammaImg.data, width, height, 0.45);

cv::Mat removedGlareGammaRGB(height / 2, width / 2, CV_8UC3);
cvtColor(removedGlareGammaImg, removedGlareGammaRGB, cv::COLOR_BayerRG2RGB);

namedWindow("Removed glare RGB with gamma correction", cv::WINDOW_KEEPRATIO);
imshow("Removed glare RGB with gamma correction", removedGlareGammaRGB);
cv::waitKey();
```

EXAMPLE OF OBTAINING AN IMAGE WITH AN ANY POLARIZATION ANGLE

The application opens the image file and creates the image with the virtual polarizer from 0 to 360 degrees in 10-degree increments. In fact, this is analogous to the linear polarizer in front of the sensor.

```
#include <iostream>
```

```cpp
#include <opencv2/opencv.hpp>
#include "PolarProcessorApi.h"

int main(void)
{
    std::cout << "===================================" << std::endl;
    std::cout << "PolarProcessor v" <<
        plib::PolarProcessor::getVersion() << std::endl;
    std::cout << "Virtual polarizer filter example" << std::endl;
    std::cout << "===================================" << std::endl;

    // Haight of polirized RGGB image.
    int height = 2048;
    // Width of polarized RGGB image.
    int width = 2448;
    // Polar processor object.
    plib::PolarProcessorApi pProcessor;

    // Read polarized RGGB image.
    cv::Mat inputImg = imread("img\\example_virtual_polarizer.bmp", cv::IMREAD_GRAYSCALE);

    // Make RAW (RGGB) image with virtual poarizer (size / 2 of source image).
    cv::Mat virtualPolarizerImg = cv::Mat::zeros(height / 2, width / 2, CV_8UC1);

    // Make RAW (RGB) image with virtual poarizer (size / 2 of source image).
    cv::Mat virtualPolarizerRGB(height / 2, width / 2, CV_8UC3);

    for (uint32_t i = 0; i < 360; i = i + 10)
    {
        // Virtual polarizer
        pProcessor.virtualPolarizer(
            inputImg.data, virtualPolarizerImg.data,
            width, height, (float)i / 180 * 3.14159);

        // Converting from RAW (RGGB) to RGB.
        cvtColor(virtualPolarizerImg, virtualPolarizerRGB, cv::COLOR_BayerRG2RGB);

        // Show image
        namedWindow("Virtual polarizer", cv::WINDOW_KEEPRATIO);
        imshow("Virtual polarizer", virtualPolarizerRGB);
        cv::waitKey(30);
    }

    // Wait press key
    cv::waitKey();

    return 0;
}
```



0 градусов 30 градусов 60 градусов 90 градусов

Figure 6 – The application result.

EXPLANATIONS TO THE VIRTUAL POLARIZER EXAMPLE

The program only requires connecting the OpenCV library header files and the Polar Processor.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include "PolarProcessorApi.h"
```

At the beginning the library version and information about the demonstration program are displayed.

```
std::cout << "====================================" << std::endl;
std::cout << "PolarProcessor v" << plib::PolarProcessor::getVersion() << std::endl;
std::cout << "Virtual polarizer filter example" << std::endl;
std::cout << "====================================" << std::endl;
```

Next, the variables for the dimensions of the demo image are declared, and an instance of the PolarProcessorApi program class is created.

```
int height = 2048;
int width = 2448;
plib::PolarProcessorApi pProcessor;
```

After that the source image file is read.

```
cv::Mat inputImg = imread("img\\example_virtual_polarizer.bmp",cv::IMREAD_GRAYSCALE);
```

Then empty images are created to store the image resulting from the processing by the virtual polarization filter, as well as to store the converted image from RGGB to RGB. The size of the processed image is 2 times less than the original image.

```
cv::Mat virtualPolarizerImg = cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
cv::Mat virtualPolarizerRGB(height / 2, width / 2, CV_8UC3);
```

Then there is a cycle in which the angle of the virtual polarizer is determined in the range from 0 to 360 degrees in increments of 10 degrees.

```
for (uint32_t i = 0; i < 360; i = i + 10)
```

Inside the loop, the image is formed according to required angle of the virtual polarizer using the virtualPolarizer(...) method. The angle is specified in radians.

```
pProcessor.virtualPolarizer(inputImg.data, virtualPolarizerImg.data,
                            width, height, (float)i / 180 * 3.14159);
```

The resulting image is converted from RGGB to RGB. After that it is displayed in the named window and a delay of 30 milliseconds is started.

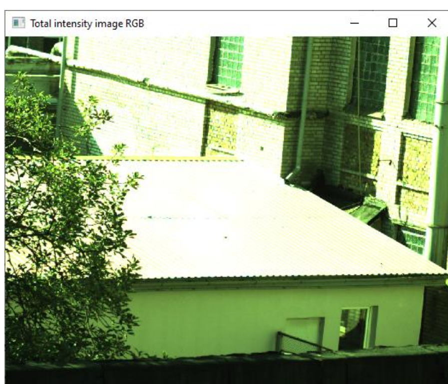
```
cvtColor(virtualPolarizerImg, virtualPolarizerRGB, cv::COLOR_BayerRG2RGB);

namedWindow("Virtual polarizer", cv::WINDOW_KEEPRATIO);
imshow("Virtual polarizer", virtualPolarizerRGB);
cv::waitKey(30);
```

EXAMPLE OF DEGREE OF LINEAR POLARIZATION (DOLP)

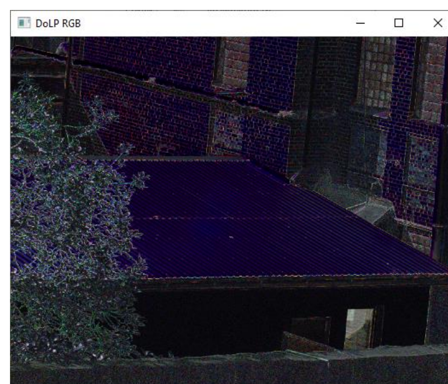
The demo application opens the image file and generates an image of the degree of linear polarization. The full intensity image is given for comparison.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include "PolarProcessorApi.h"

int main(void)
{
    std::cout << "=====================================" << std::endl;
    std::cout << "PolarProcessor v" <<
        plib::PolarProcessor::getVersion() << std::endl;
    std::cout << "Degree of Linear Polarization example" << std::endl;
    std::cout << "=====================================" << std::endl;

    // Init variables.
    int height = 2048;                  // Haight of polirized RGGB image.
    int width = 2448;                   // Width of polarized RGGB image.
    plib::PolarProcessorApi pProcessor;     // Polar processor object.

    // Read polarized RGGB image.
    cv::Mat inputImg = imread("img\\example_building.bmp", cv::IMREAD_GRAYSCALE);

    // Make total intencity image (size / 2 of source image).
    cv::Mat totalIntensityImg = cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
    pProcessor.totalIntensity(inputImg.data, totalIntensityImg.data, width, height);

    // Converting from RAW (RGGB) to RGB
    cv::Mat totalIntensityRGB(height / 2, width / 2, CV_8UC3);
    cvtColor(totalIntensityImg, totalIntensityRGB, cv::COLOR_BayerRG2RGB);

    // Show image
```

```
    namedWindow("Total intensity image RGB", cv::WINDOW_KEEPRATIO);
    imshow("Total intensity image RGB", totalIntensityRGB);

    // Make DoLP
    cv::Mat DoLPGrayImg = cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
    pProcessor.DoLP(inputImg.data, DoLPGrayImg.data, width, height);

    // Converting from RAW (RGGB) to RGB
    cv::Mat DoLPGrayRGB(height / 2, width / 2, CV_8UC3);
    cvtColor(DoLPGrayImg, DoLPGrayRGB, cv::COLOR_BayerRG2RGB);

    // Show image
    namedWindow("DoLP RGB", cv::WINDOW_KEEPRATIO);
    imshow("DoLP RGB", DoLPGrayRGB);
    cv::waitKey();

    return 0;
}
```



Total intensity



DoLP

Figure 7 – The application result.

EXPLANATIONS TO THE LINEAR POLARIZATION DEGREE EXAMPLE

The example only requires connecting the OpenCV library header files and the Polar Processor.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include "PolarProcessorApi.h"
```

At the beginning the library version and information about the example are displayed.

```
std::cout << "=======================================" << std::endl;
std::cout << "PolarProcessor v" << plib::PolarProcessor::getVersion() << std::endl;
std::cout << "Degree of Linear Polarization example" << std::endl;
std::cout << "=======================================" << std::endl;
```

Next, the variables for the dimensions of the demo image are declared, and an instance of the PolarProcessorApi program class is created.

```
int height = 2048;
int width = 2448;
plib::PolarProcessorApi pProcessor;
```

After that the source image file is read.

```
cv::Mat inputImg = imread("img\\example_building.bmp", cv::IMREAD_GRAYSCALE);
```

Then an empty image is created to store the resulting full-intensity image. The size of the processed image is 2 times less than the original one. The totalIntensity(...) method is called to get the full intensity image.

```
cv::Mat totalIntensityImg = cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
pProcessor.totalIntensity(inputImg.data, totalIntensityImg.data, width, height);
```

The resulting image is converted from RGGB to RGB and displayed.

```
cv::Mat totalIntensityRGB(height / 2, width / 2, CV_8UC3);
cvtColor(totalIntensityImg, totalIntensityRGB, cv::COLOR_BayerRG2RGB);

namedWindow("Total intensity image RGB", cv::WINDOW_KEEPRATIO);
imshow("Total intensity image RGB", totalIntensityRGB);
```

Then similar actions are performed to obtain an image of the degree of linear polarization. The size of the processed image is 2 times less than the original image. The DoLP(...) method is called to obtain an image of the degree of linear polarization.

```
cv::Mat DoLPGrayImg = cv::Mat::zeros(height / 2, width / 2, CV_8UC1);
pProcessor.DoLP(inputImg.data, DoLPGrayImg.data, width, height);
```

The resulting image is converted from RGGB to RGB and displayed.

```
cv::Mat DoLPGrayRGB(height / 2, width / 2, CV_8UC3);
cvtColor(DoLPGrayImg, DoLPGrayRGB, cv::COLOR_BayerRG2RGB);

namedWindow("DoLP RGB", cv::WINDOW_KEEPRATIO);
imshow("DoLP RGB", DoLPGrayRGB);
cv::waitKey();
```