APPLICATION NOTE

Getting Started with pylon and OpenCV

Applicable to all Basler USB3 Vision, GigE Vision, and IEEE 1394 cameras

Document Number: AW001368
Version: 01 Language: 000 (English)
Release Date: 18 November 2015
Software Version (pylon): 5.x

BASLER
the power of sight

Contacting Basler Support Worldwide

Europe, Middle East, Africa

Basler AG
An der Strusbek 60–62
22926 Ahrensburg
Germany

Tel. +49 4102 463 515
Fax +49 4102 463 599

support.europe@baslerweb.com

The Americas

Basler, Inc.
855 Springdale Drive, Suite 203
Exton, PA 19341
USA

Tel. +1 610 280 0171
Fax +1 610 280 7608

support.usa@baslerweb.com

Asia-Pacific

Basler Asia Pte. Ltd.
35 Marsiling Industrial Estate Road 3
#05–06
Singapore 739257

Tel. +65 6367 1355
Fax +65 6367 1255

support.asia@baslerweb.com

www.baslerweb.com

Table of Contents

1 Introduction

OpenCV is an open-source computer vision library that allows you to perform image processing on Basler Machine Vision cameras. This application note provides information on how to install and use OpenCV in combination with Basler's pylon Camera Software Suite in Microsoft Visual Studio on Windows operating systems.

OpenCV does not support Machine Vision standards such as USB3 Vision, GigE Vision, or IEEE 1394 (FireWire). Therefore, it is not recommended to grab images using OpenCV API functions. Instead, Basler recommends using the pylon Camera Software Suite SDK to grab images and convert them to OpenCV images.

This document provides further information on the integration of OpenCV functions for image display, image saving, and video recording into your pylon source code.

2 Requirements

2.1 Basler pylon Camera Software Suite

This document assumes you are using the latest version of the pylon Camera Software Suite for Windows. The procedures described in this application note have been accomplished with the pylon Camera Software Suite version 5.0.0.6150 (64 bit).

You can download the pylon Camera Software Suite from the **Downloads** section of the Basler website: www.baslerweb.com.

For information about how to install the pylon Camera Software Suite, see the *Installation and Setup Guide for Cameras Used with pylon for Windows* (AW000611). You can download the document from the **Documents Download** section of the Basler website: www.baslerweb.com.

2.2 OpenCV

This document assumes you are using OpenCV version 3.0.0.

You can download OpenCV for Windows from www.opencv.org.

2.3 Microsoft Visual Studio and Microsoft Windows

The procedures described in this application note have been accomplished with Microsoft Visual Studio 2012 on a Windows 7 64-bit operating system.

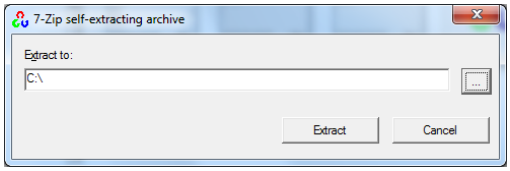
	The procedures described in this document can also be applied to Microsoft Visual Studio 2013 and Windows 7 or Windows 8 (32-bit or 64-bit operating systems).

3 OpenCV Installation

The following steps apply to OpenCV version 3.0.0, but may also apply to newer versions.

To set up the environment variables for OpenCV:

1. Extract the OpenCV files to C:\

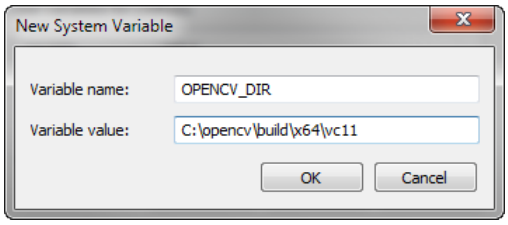


2. On the **Start** menu, right-click **Computer** and select **Properties**.
3. In the left pane of the **System** dialog box, click **Advanced system settings**.
4. On the **Advanced** tab of the **System Properties** dialog box, click **Environment Variables**.
5. In the **System Variables** box of the **Environment Variables** dialog box, scroll to **Path** and select it.
6. Click the lower of the two **Edit** buttons in the dialog box.
7. In the **Edit System Variable** dialog box, scroll to the end of the string in the **Variable value** box and add

   `;C:\opencv\build\x64\vc11\bin`

   For the 32-bit OpenCV version, replace `x64` with `x86`.

   If you are using Microsoft Visual Studio 2013, replace `vc11` with `vc12`.
8. Click **OK**.
   You should now set an OpenCV environment variable to facilitate your work. The variable will hold the path to the build directory of the OpenCV library. To do so:
9. Click the lower of the two **New** buttons in the dialog box.
10. In the **New System Variable** dialog box, select **Variable name** and add `OPENCV_DIR`.
11. Select **Variable value** and add `C:\opencv\build\x64\vc11`:



Depending on your Visual Studio and Windows operating system version, you may need to adapt the **Variable value** entry accordingly. Possible entries for OpenCV 3.0.0 are:

`C:\opencv\build\x86\vc11` (Visual Studio 2012 – 32-bit Windows)

`C:\opencv\build\x64\vc11` (Visual Studio 2012 – 64-bit Windows)

`C:\opencv\build\x86\vc12` (Visual Studio 2013 – 32-bit Windows)

`C:\opencv\build\x64\vc12` (Visual Studio 2013 – 64-bit Windows)

4 Common Settings for Building Applications with pylon and OpenCV in Visual Studio

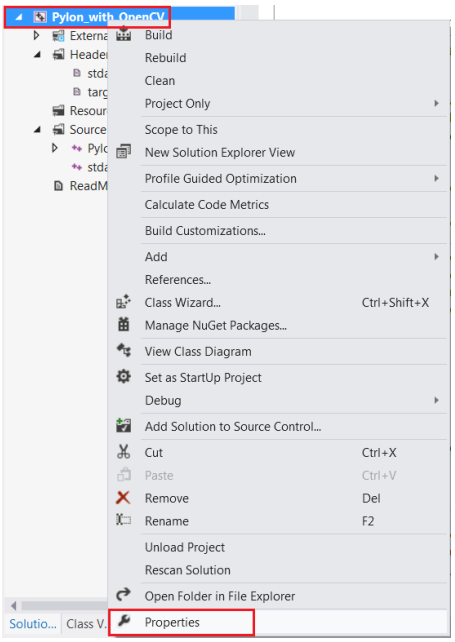
The steps below outline how to prepare the Visual Studio settings in order to use OpenCV with Basler's pylon Camera Software Suite SDK.

For more information about how to set up your Microsoft Visual Studio project for building applications with pylon, refer to the *Programmer's Guide and API Reference for pylon for Windows*. The guide is part of the pylon Camera Software Suite installation (**Start** > **All Programs** > **Basler** > **pylon 5 Camera Software Suite** > **Documentation** > **C++ Programmer's Guide and API Reference Documentation**).

This application note assumes that you have already read the *Programmer's Guide and API Reference for pylon for Windows* (especially the topic *Programmer's Guide* and its section *Common Settings for Building Applications with pylon*) and have set up the PYLON_DEV_DIR environment variable for building a 64-bit application in your project. The project used in this document is named "Pylon_with_OpenCV".

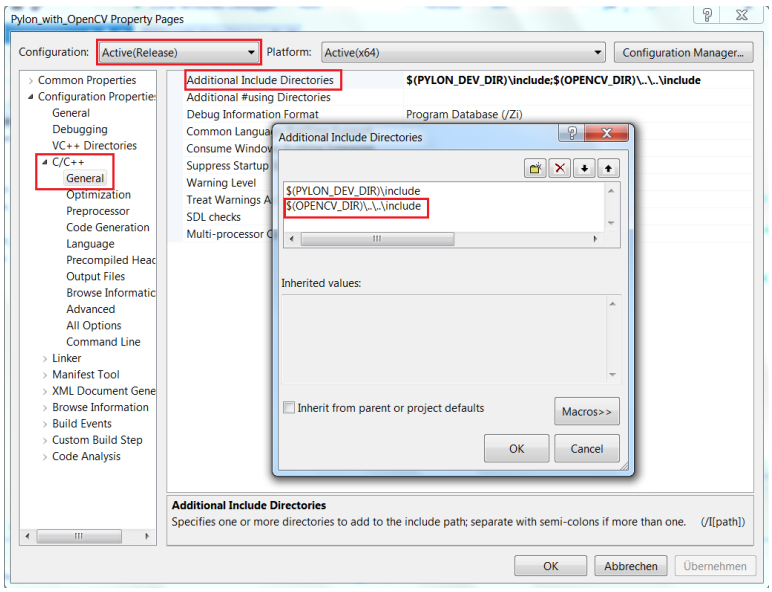
To set up the Visual Studio project properties for OpenCV:

1. Open your project in Visual Studio 2012, right-click on the project, and select **Properties**:

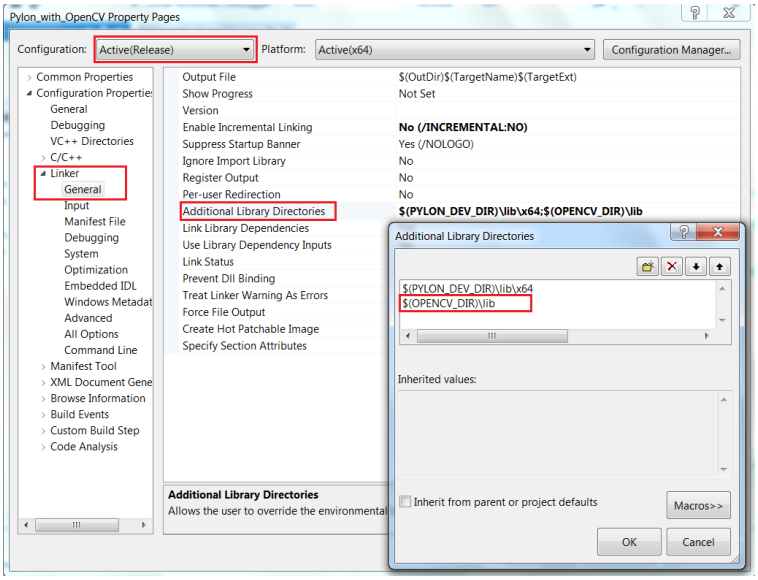


2. In the **Property Pages** window, select **C/C++** > **General**.

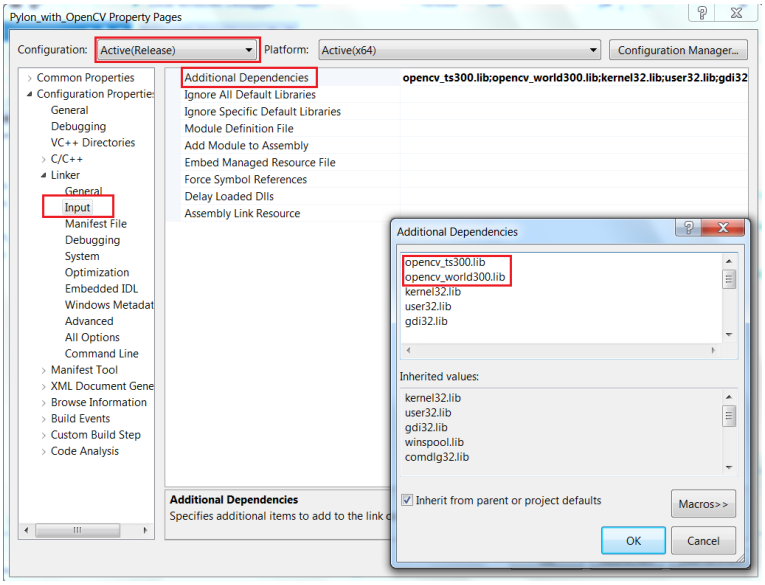
3. From the **Additional Include Directories** drop down, select **Edit** and add the OpenCV include directory: `$(OPENCV_DIR)\..\..\include`



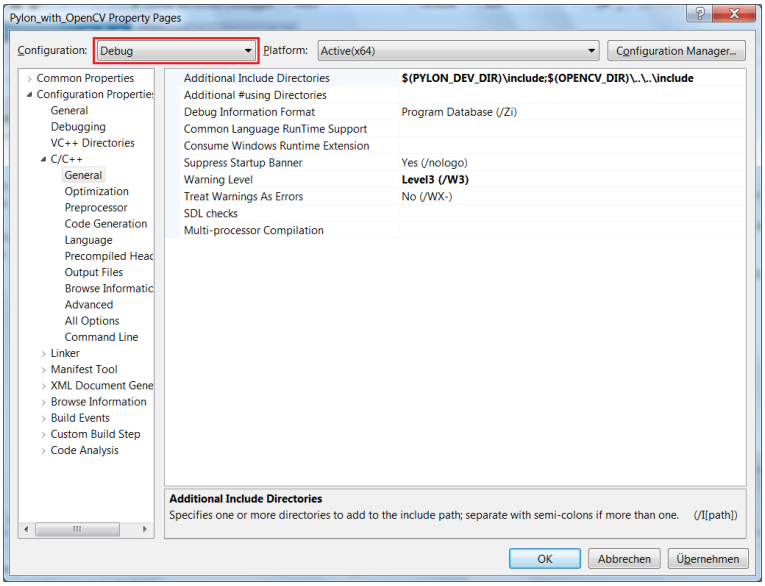
4. Click **OK**.
5. In the **Property Pages** window, select **Linker** > **General**.
6. From the **Additional Library Directories** drop down, select **Edit** and add the OpenCV library directory: `$(OPENCV_DIR)\lib`



7. Click **OK**.
8. In the **Property Pages** window, select **Linker** > **Input**.
9. From the **Additional Dependencies** drop down, select **Edit** and add the following OpenCV lib files:
   - `opencv_ts300.lib`
   - `opencv_world300.lib`



10. Click **OK**.
11. Click **Apply**.
12. To configure your project for **Debug** mode, do the following:
    a. In the **Configuration** drop down box, select **Debug**.



    b. Repeat steps 2 to 8.
    c. In step 9, add the following OpenCV debug lib files:
       - `opencv_ts300d.lib`
       - `opencv_world300d.lib`
    d. Click **OK**.
    e. Click **Apply**.

5 Sample Code to Run pylon and OpenCV

The following sample code demonstrates how to accomplish these tasks:

- Acquire images with pylon API functions.
- Convert the acquired pylon images to OpenCV images.
- Display and save images with OpenCV functions.
- Record video files with OpenCV functions.

1. First, we include all necessary OpenCV and pylon API header files, declare required namespaces, and define some global variables needed throughout our sample.

Pylon_with_OpenCV.cpp

(Global Scope)

```
#include "stdafx.h"

// Define if images are to be saved.
// '0'- no; '1'- yes.
#define saveImages 1
// Define if video is to be recorded.
// '0'- no; '1'- yes.
#define recordVideo 1

// Include files to use OpenCV API.
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/video/video.hpp>

// Include files to use the PYLON API.
#include <pylon/PylonIncludes.h>
#ifdef PYLON_WIN_BUILD
#    include <pylon/PylonGUI.h>
#endif

// Namespace for using pylon objects.
using namespace Pylon;

// Namespace for using OpenCV objects.
using namespace cv;

// Namespace for using cout.
using namespace std;

// Number of images to be grabbed.
static const uint32_t c_countOfImagesToGrab = 10;
```

2. We then create the first camera we found, regardless of its interface, i.e. GigE Vision, USB3 Vision or IEEE 1394 (FireWire). We get the camera's GenICam *nodemap* to access the camera's *Width* and *Height* parameters which are needed for the initialization of OpenCV video creator. We set up a pylon image format converter object and define the desired output pixel format, which is needed to convert the grabbed pylon image buffer to a pylon image. After that, we convert the pylon image to an OpenCV image.

Pylon_with_OpenCV.cpp

(Global Scope)

```
// Automagically call PylonInitialize and PylonTerminate to ensure the pylon runtime system
// is initialized during the lifetime of this object.
Pylon::PylonAutoInitTerm autoInitTerm;

try
{
    // Create an instant camera object with the camera device found first.
    CInstantCamera camera( CTlFactory::GetInstance().CreateFirstDevice());

    // Print the model name of the camera.
    cout << "Using device " << camera.GetDeviceInfo().GetModelName() << endl;

    // Get a camera nodemap in order to access camera parameters.
    GenApi::INodeMap& nodemap= camera.GetNodeMap();
    // Open the camera before accessing any parameters.
    camera.Open();
    // Create pointers to access the camera Width and Height parameters.
    GenApi::CIntegerPtr width= nodemap.GetNode("Width");
    GenApi::CIntegerPtr height= nodemap.GetNode("Height");

    // The parameter MaxNumBuffer can be used to control the count of buffers
    // allocated for grabbing. The default value of this parameter is 10.
    camera.MaxNumBuffer = 5;

    // Create a pylon ImageFormatConverter object.
    CImageFormatConverter formatConverter;
    // Specify the output pixel format.
    formatConverter.OutputPixelFormat= PixelType_BGR8packed;
    // Create a PylonImage that will be used to create OpenCV images later.
    CPylonImage pylonImage;
    // Declare an integer variable to count the number of grabbed images
    // and create image file names with ascending number.
    int grabbedImages= 0;
```

3. The OpenCV video creator and OpenCV image objects are declared and initialized. Then, we use pylon API functions to start image acquisition and retrieve the grabbed images:

Pylon_with_OpenCV.cpp

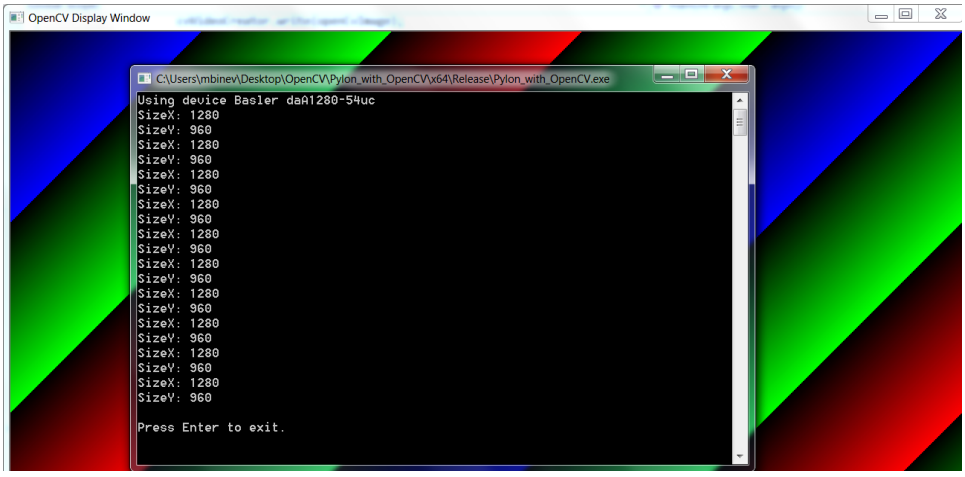
(Global Scope)

```
    // Create an OpenCV video creator.
    VideoWriter cvVideoCreator;
    // Create an OpenCV image.
    Mat openCvImage;

    // Define the video file name.
    std::string videoFileName= "openCvVideo.avi";

    // Define the video frame size.
    cv::Size frameSize= Size((int)width->GetValue(), (int)height->GetValue());

    // Set the codec type and the frame rate. You have 3 codec options here.
    // The frame rate should match or be lower than the camera acquisition frame rate.
    cvVideoCreator.open(videoFileName, CV_FOURCC('D','I','V','X'), 20, frameSize, true);
    //cvVideoCreator.open(videoFileName, CV_FOURCC('M','P','4','2'), 20, frameSize, true);
    //cvVideoCreator.open(videoFileName, CV_FOURCC('M','J','P','G'), 20, frameSize, true);

    // Start the grabbing of c_countOfImagesToGrab images.
    // The camera device is parameterized with a default configuration which
    // sets up free-running continuous acquisition.
    camera.StartGrabbing( c_countOfImagesToGrab, GrabStrategy_LatestImageOnly);

    // This smart pointer will receive the grab result data.
    CGrabResultPtr ptrGrabResult;

    // Camera.StopGrabbing() is called automatically by the RetrieveResult() method
    // when c_countOfImagesToGrab images have been retrieved.
    while ( camera.IsGrabbing())
    {
        // Wait for an image and then retrieve it. A timeout of 5000 ms is used.
        camera.RetrieveResult( 5000, ptrGrabResult, TimeoutHandling_ThrowException);
```

4. Finally, we convert the grabbed image buffer to a pylon image, which in turn is converted to an OpenCV image. The OpenCV image is used for image display, image saving, and video file recording using OpenCV functions:

```
Pylon_with_OpenCV.cpp*
(Global Scope)                                   main(int argc, char * argv[])
            // Image grabbed successfully?
            if (ptrGrabResult->GrabSucceeded())
            {
                // Access the image data.
                cout << "SizeX: " << ptrGrabResult->GetWidth() << endl;
                cout << "SizeY: " << ptrGrabResult->GetHeight() << endl;

                // Convert the grabbed buffer to a pylon image.
                formatConverter.Convert(pylonImage, ptrGrabResult);

                // Create an OpenCV image from a pylon image.
                openCvImage= cv::Mat(ptrGrabResult->GetHeight(), ptrGrabResult->GetWidth(), CV_8UC3, (uint8_t *) pylonImage.GetBuffer());

                // Set saveImages to '1' to save images.
                if (saveImages) {
                    // Create the current image name for saving.
                    std::ostringstream s;
                    // Create image name files with ascending grabbed image numbers.
                    s<< "image_" << grabbedImages << ".jpg";
                    std::string imageName(s.str());
                    // Save an OpenCV image.
                    imwrite(imageName, openCvImage);
                    grabbedImages++;
                }

                // Set recordVideo to '1' to record AVI video file.
                if (recordVideo)
                    cvVideoCreator.write(openCvImage);

                // Create an OpenCV display window.
                namedWindow( "OpenCV Display Window", CV_WINDOW_NORMAL); // other options: CV_AUTOSIZE, CV_FREERATIO

                // Display the current image in the OpenCV display window.
                imshow( "OpenCV Display Window", openCvImage);
                // Define a timeout for customer's input in ms.
                // '0' means indefinite, i.e. the next image will be displayed after closing the window.
                // '1' means live stream
                waitKey(1);
```

5. The image below shows the running application with a test image grabbed from a Basler dart USB3 Vision camera in an OpenCV display window:



If you want to get the complete sample source code, get in contact with your local Basler support team as listed at the beginning of this document.

Revision History

Document Number	Date	Changes
AW001368**01**000	18 Nov 2015	Initial release version of this document.