APPLICATION NOTE


```
// Create an instant camera object with the firs
Camera_t camera( CTlFactory::GetInstance().Creat

// Register an image event handler that accesses
camera.RegisterImageEventHandler( new CSampleIma
Ownership_TakeOwnership);

// Open the camera.
camera.Open();
```

Merging Color Data of Basler 2D Cameras with Basler blaze Depth Data

Applicable to 2D color cameras and blaze cameras only

Document Number: AW001665
Version: 01   Language: 000 (English)
Release Date: 29 January 2021

Álava Ingenieros
GRUPO ÁLAVA

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

This document describes how to use a Basler 2D color area scan camera and a Basler blaze 3D ToF camera to create a joint camera system that will allow you to combine the RGB color data of the 2D color camera with the depth data of the blaze camera. The result of such an operation, also known as data fusion, is shown in Figure 1. It shows a point cloud of the blaze camera where the right half has been colorized using the 2D camera's color data.
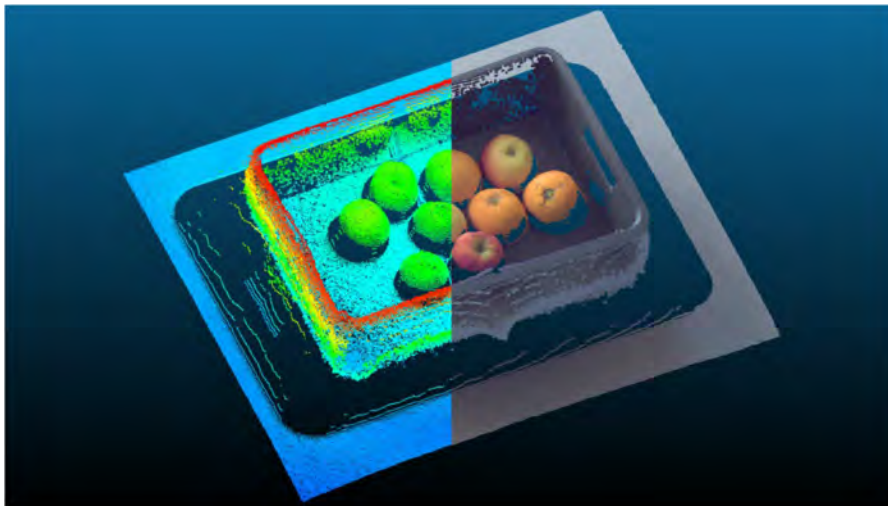

Figure 1: Point Cloud of the Basler blaze Colorized Using the Basler 2D Camera RGB Color Data

The Basler blaze provides 3D point clouds in real time. The point cloud represents the surface of an object as seen by the camera. In addition, the camera provides an infrared intensity image of the scene. For a large number of applications, this data is already sufficient. In some applications, however, it may be desirable to have additional high-resolution color data. For example, the appearance of objects under infrared lighting can differ from their appearance in visible light. It's also easier to analyze fine structures such as bar codes or labels in a higher-resolution color image. In addition, the neural networks used in machine learning are usually trained on color data in the visible range.

Use cases affected by these challenges may benefit from the data fusion techniques described in this document. Data fusion allows the merging of data from the point cloud of a Basler blaze camera with the RGB data from a 2D color camera with the aim of forming a single, combined dataset. Adding high-resolution 2D images to 3D applications can improve pattern recognition or object detection algorithms.

The blaze SDK contains sample programs that demonstrate how to calibrate the joint camera system, how to perform the data fusion, and how to display the data stream in real-time. The sample programs use the open-source libraries OpenCV and Point Cloud Library for calibration, data fusion, and display.

2 Technical Setup

2.1 Selection and Assembly of Components

There are several aspects to consider when selecting components for the system.

◾ Positioning of the cameras

The data from the color camera is transformed into the coordinate system of the blaze camera. This works best if the optical axes of the cameras are aligned and they are as close to each other as possible.

◾ Resolution of the 2D color camera

The resolution of the 2D color camera should be at least the same as that of the blaze camera, preferably even higher. Basler recommends choosing a camera with at least 4 times the resolution of the blaze, i.e., 1.2 MP, to benefit from the higher-resolution color image.

◾ Field of view

The field of view of the 2D color camera should match that of the blaze camera, which is 67° x 51°. However, Basler recommends choosing a lens for the 2D color camera with a field of view about 5 % larger than that of the blaze. This ensures that color information is available for all points of the point cloud even if the positioning of the cameras isn't perfect.

To achieve the best possible match of the cameras' fields of view and to reduce occlusion, Basler recommends placing the 2D color camera as close as possible to the blaze's lens. This means installing it on the right-hand side of the blaze. To join the two cameras, a mounting bracket is required. This will ensure that the positioning of the cameras won't change during use.

2.2   Sample System Setup

This application note is based on an sample system that meets the above requirements. The components used for this are listed in Table 1. Basler designed a mounting bracket to allow joining the ace camera to the blaze camera with optimum alignment of the optical axes and minimal distance between the cameras. The mounting bracket was printed on a 3D printer. Figure 2 shows an image of the assembled system.

If you want to build the system as described here, you can download a detailed description of the necessary components and their assembly as well as the 3D model of the Basler blaze/ace mounting bracket for 3D printing from the Basler website.

Description	Order Number
Camera, Basler blaze-101	107796
Camera, Basler acA1300-75gc	106757
Lens, Basler Lens C125-0418-5M-P f4mm	2000034830

Table 1: Materials Required for the Assembly of the Sample Camera System

Figure 2: Assembled Sample Camera System

2.3   Software Installation

To run the sample, a computer with Windows or Linux operating system is required. The camera assembly can be connected to the computer directly or via a switch. The following software components must be installed:

- Basler blaze SDK

  You can download the SDK from the Basler website.

- Basler pylon Camera Software Suite (version 6 or higher)

  You can download the software from the Basler website.

- OpenCV

  For more information, visit https://opencv.org/.

- PointCloudLibrary (PCL) (Optional: Required only for visualization)

  For more information, visit http://pointclouds.org/.

The Basler blaze SDK includes a readme file with instructions for compiling and running the samples. For calibration and data fusion, use the sample programs in the **Samples\cpp\ColorAndDepthFusion** folder.

> ⓘ The sample programs mentioned above use software triggers to synchronize the acquisition timing of the 2D color camera and the blaze camera. This synchronization option is easy to set up and should already be sufficient for many applications. In case greater synchronization accuracy is required, e.g., in environments with fast movements, you can increase the synchronization accuracy of the acquisition times by synchronizing the cameras via Precision Time Protocol (PTP) IEEE1588. To do this, the cameras are set to a dedicated PTP mode that synchronizes the cameras' clocks.
>
> A sample program that demonstrates the setup and operation of multiple cameras with PTP is available in the samples folder under **Samples\cpp\MultiCam\SynchronousFreeRun**.
>
> For more information about the PTP feature in Basler cameras, visit docs.baslerweb.com.

3   Displaying the Color Point Cloud

The fusion of depth and color information by means of the projective geometry is based on the pinhole camera model. Here, the depth data is projected into the coordinate system of the 2D color camera using the intrinsic and extrinsic parameters. The relationship obtained in this process can be used to merge the data. Because the 2D color camera is attached to the blaze camera in a rigid setup, the rotation and translation between the cameras are fixed and must only be determined once.

3.1   Calibration

The purpose of the calibration is to determine the relative transformation between the 2D color camera and the blaze camera. Furthermore, the intrinsic parameters, i.e., the focal length and distortion of the 2D color camera are determined.

To perform the calibration, a calibration target is required. The folder containing the sample programs in the SDK also includes a printable calibration target (**chessboard.pdf**, shown in Figure 3). The target has a chessboard pattern that defines its own coordinate system. The OpenCV library contains the `findChessboardCorners()` function, which is able to detect the corners of the chessboard in an image and to return the image coordinates of every corner point. If both cameras are "looking" at the target, corresponding points can be identified using the target coordinates, which are the same for both cameras.

To use the target, print it in its original size on a DIN A3 (297 x 420 mm) sheet. For the target to work correctly, i.e., the accuracy of the calibration, the printing accuracy should be at least 0.1 mm.

To verify whether your printed target is accurate enough, check whether the dimensions of the squares in the chessboard are 40 mm by 40 mm each.

Targets printed on 6-mm aluminum Dibond (an economical choice available through many online outlets) provide good results. For initial experiments, inkjet or laser prints may suffice if carefully glued to a flat surface, such as an aluminum plate or a piece of thick cardboard.

If the dimensions of the squares in your print aren't 40 mm by 40 mm, you can modify the sample program accordingly to ensure accurate calibration. To do this, change the value of the `m_fieldSize` parameter from 40.0 to the value of the squares in your print:

```
float m_fieldSize = 40.0; // mm corner size
```
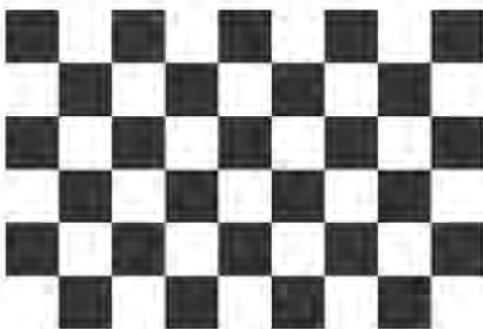

Figure 3: Calibration Target

To perform the calibration, start the **ColorAndDepthCalibration** sample program and take several images of the chessboard by pressing the **S** key. For the calibration, take 10 to 15 images of the chessboard at different positions covering the entire field of view of both cameras. The chessboard must always be completely visible in both cameras. If the chessboard has been detected correctly, it is highlighted in the images (see Figure 4). Figure 5 shows a variety of target positions that are suitable for performing the calibration.
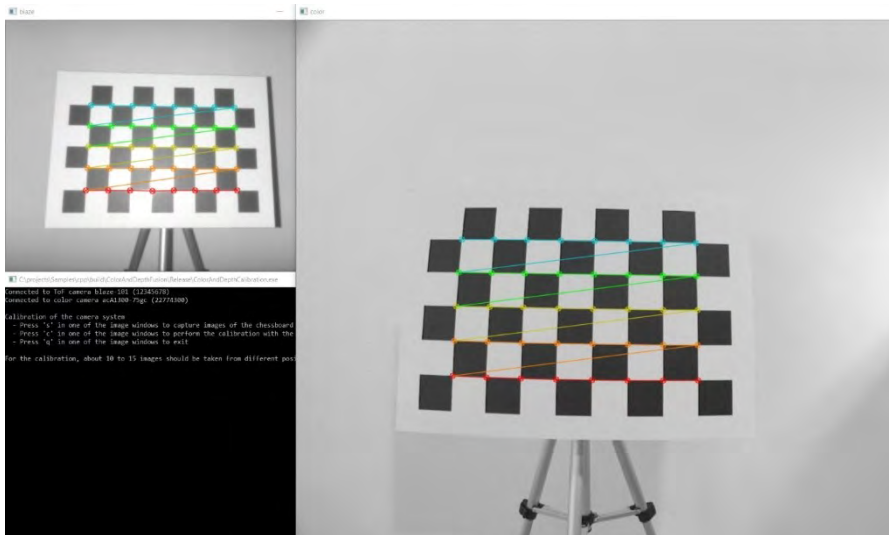

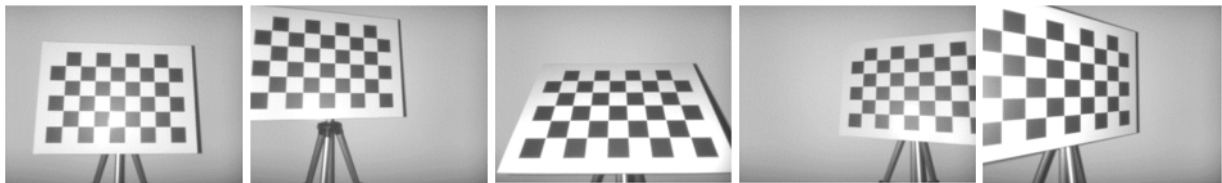Figure 4: Performing the Calibration of the Camera System

Figure 5: Examples of Chessboard Positions Suitable for Performing the Calibration

After you have taken the images, start the calibration by pressing the **C** key. During calibration, the intrinsic parameters of the 2D color camera as well as the relative transformation between it and the blaze camera are determined. The intrinsic parameters of the blaze have already been determined during manufacturing and can be read out. All the resulting data is stored in an XML file. The following section shows an example of the result of a camera system calibration.

```xml
<?xml version="1.0"?>
<opencv_storage>
<colorCameraMatrix type_id="opencv-matrix">
  <rows>3</rows><cols>3</cols><dt>d</dt>
  <data>
    8.53e+02 0.        6.62e+02
    0.       8.53e+02 5.00e+02
    0.       0.        1.
</data>
</colorCameraMatrix>
<colorDistortion type_id="opencv-matrix">
  <rows>1</rows><cols>5</cols><dt>d</dt>
  <data>-2.43e-01 9.85e-02 0. 0. 0.</data>
</colorDistortion>
<blazeCameraMatrix type_id="opencv-matrix">
  <rows>3</rows><cols>3</cols><dt>d</dt>
  <data>
    5.0887e+02 0.         3.1844e+02
    0.         5.0887e+02 2.3244e+02
    0.         0.         1.
  </data>
</blazeCameraMatrix>
<blazeDistortion type_id="opencv-matrix">
  <rows>1</rows><cols>5</cols><dt>d</dt>
  <data>0. 0. 0. 0. 0.</data>
</blazeDistortion>
<rotation type_id="opencv-matrix">
  <rows>3</rows><cols>3</cols><dt>d</dt>
  <data>
    9.9997e-01 -3.8126e-03 -5.1768e-03
    3.8790e-03  9.9990e-01  1.2873e-02
    5.1273e-03 -1.2893e-02  9.9990e-01
  </data>
</rotation>
<translation type_id="opencv-matrix">
  <rows>3</rows><cols>1</cols><dt>d</dt>
  <data>-4.1266e+01 3.8105e-01 -1.6245e+00</data>
</translation>
</opencv_storage>
```

During the first step of the calibration, the intrinsic camera parameters of the 2D color camera are determined. In the sample program, this is done using the `calibrateCamera()` function from the OpenCV software library. The result is a 3 x 3 matrix of the intrinsic camera parameters. This matrix is called **colorCameraMatrix** in the result file. Furthermore, a 5-element vector is determined that contains the distortion coefficients of the 2D color camera's lens. This is called **colorDistortion** in the result file.

The corresponding parameters of the blaze camera, i.e., **blazeCameraMatrix** and **blazeDistortion**, don't have to be calculated, but can be read directly from the camera. The distortion coefficients are zero for the blaze since the distortion is already corrected internally in the camera.

During the second step of the calibration, the relative transformation between 2D color camera and blaze camera is determined. This information is required to project the 3D points of the blaze into the 2D color image to determine the corresponding color information of the 3D points. To determine the transformation, the `stereoCalibrate()` function from the OpenCV software library is used. This function returns the rotation and translation between the two cameras. In the result file, these are a 3 x 3 rotation matrix and a 3-element vector describing the translation.

With the help of the rotation matrix and the translation vector, points in the coordinate system of the blaze camera can be transformed to points in the coordinate system of the 2D color camera. In technical terms, a change of basis is performed. Due to its duality, the tuple of rotation matrix and translation vector is equivalent to the position of the blaze camera with respect to the coordinate system of the 2D color camera.

Note that the calibration must be repeated every time the relative position or the optical setup of the cameras changes, e.g., when changing the focus of the cameras.

3.2   Data Fusion: Overlaying 3D Points with Color Data

Merging the 3D data from a Basler blaze camera with the color image from a 2D color camera adds color information to the points of the point cloud. This is achieved by matching a given 3D point from the blaze camera with the pixel of the 2D color camera onto which this point is projected. The sample program needs the **calibration.xml** file that was created during calibration for this because it contains the necessary intrinsic camera parameters as well as the relative transformation between the cameras.

For the projection of the points, the `projectPoints()` function from the OpenCV software library is used. The red, green, and blue (RGB) color information of this pixel is then set as the color of the point in the point cloud. The result is a color point cloud.

To create the color point cloud, a transformation of the P_{blaze} 3D points of the blaze camera into the camera coordinate system of the 2D color camera using the $R_{blaze \rightarrow ace}$ rotation and $T_{blaze \rightarrow ace}$ translation is performed first.

$$P_{ace}=[R_{blaze \rightarrow ace} \mid T_{blaze \rightarrow ace]} \cdot P_{blaze}$$

Next, the projection into the 2D pixel coordinate system of the 2D camera is done using the K_{ace} intrinsic camera parameters.

$$p=K_{ace} \cdot [I \mid 0] \cdot P_{ace}$$

The color point cloud generated in this way is then displayed using the PCLVisualizer from the PCL software library. Figure 6 shows the input images of the blaze camera and the 2D color camera as well as the color 3D point cloud generated from them. The additional color information allows the red and green boxes to be clearly recognized. In the depth image alone, they are almost indistinguishable.
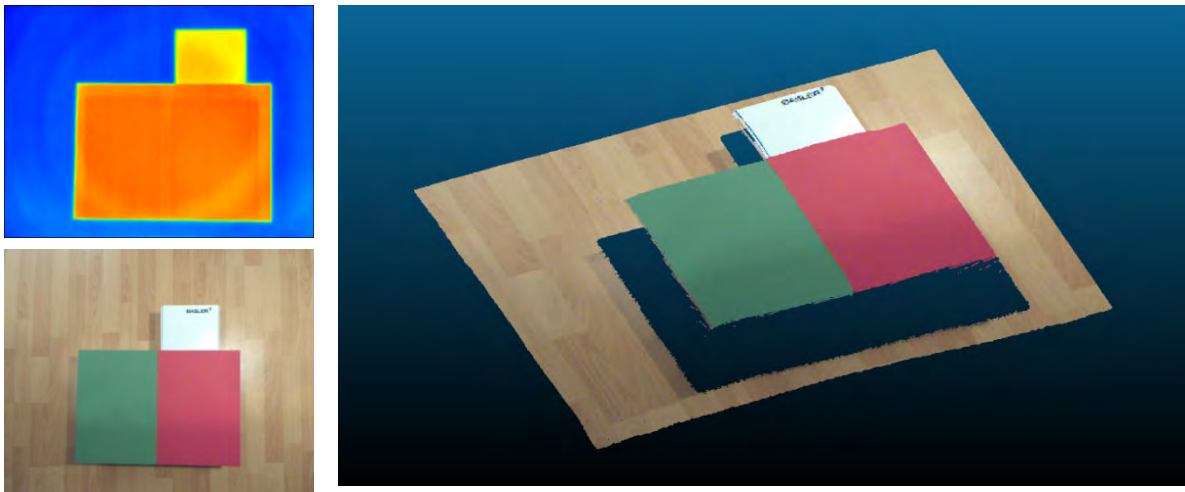

Figure 6: Depth Image of the blaze and Color Image of the 2D Camera as Well as the Resulting Color 3D Point Cloud

4   Conclusion

The data fusion process detailed in this document allows you to combine the depth data of a blaze camera with the color data of a 2D color camera. This way you can generate a high-resolution color image of the scene in addition to the depth image. Merging these datasets can help you with image processing and interpretation. Also, the color image can be used to generate a color point cloud that can make understanding the 3D scene easier, especially for human users.

5   Further Reading

A thorough discussion of homogeneous transforms, camera calibration, and stereo techniques is beyond the scope of this document. Comprehensive overviews of the principles of computer vision can be found in the following publications:

- Richard Hartley, Andrew Zisserman: *Multiple View Geometry in Computer Vision*, Cambridge University Press 2004
- Richard Szeliski: *Computer Vision: Algorithms and Applications*, Springer 2011

For further information about the library functions mentioned in this text, please refer to the OpenCV online documentation of the **Camera Calibration** and **3D Reconstruction** modules (https://opencv.org) and the documentation of the Point Cloud Library (https://pointclouds.org).

For comprehensive information about Basler cameras, their features, and other Basler products, visit docs.baslerweb.com.

Revision History

Document Number	Date	Changes
AW00166501000	28 January 2021	Initial release version of this document.