TELEDYNE FLIR

APPLICATION NOTE

Lepton 3.1R Dewarping

Document Number: Rev 100
Publication Date: 08/22/2023

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Revision History

Version	Date	Comments
100	08/22/2023	Initial Release

1.2 Reference Files

Ref Number	File name	Comments
1	Lepton_Dewarp_example.py	Python sample code for Lepton Dewarping

1.3 Scope

The infrared (IR) Lepton 3.1R utilizes a 95° wide field of view (WFOV) lens. Barrel distortion is created by the WFOV lens, causing the center of the image to appear magnified slightly more than the edges. That makes straight lines appear to curve around the edge of the image. Barrel distortion is undesirable for many imaging applications. This application note describes how to apply distortion correction (dewarping) on the Lepton 3.1R output using OpenCV built-in functions.

2 IMAGE DISTORTION

2.1 Overview

A WFOV lens allows the imager to capture more in a scene, but it will cause distortion making objects in the scene appear deformed. Generally, two types of distortion can exist in a camera: radial and tangential. Radial distortion is when straight lines curve either inwards to or outwards from the center, while tangential distortion is when the lens is tilted with respect to the image plane and the image appears skewed. To fix lens distortion, calibrate the camera to compute the transform matrices, which correct the camera's mapping into the image plane.

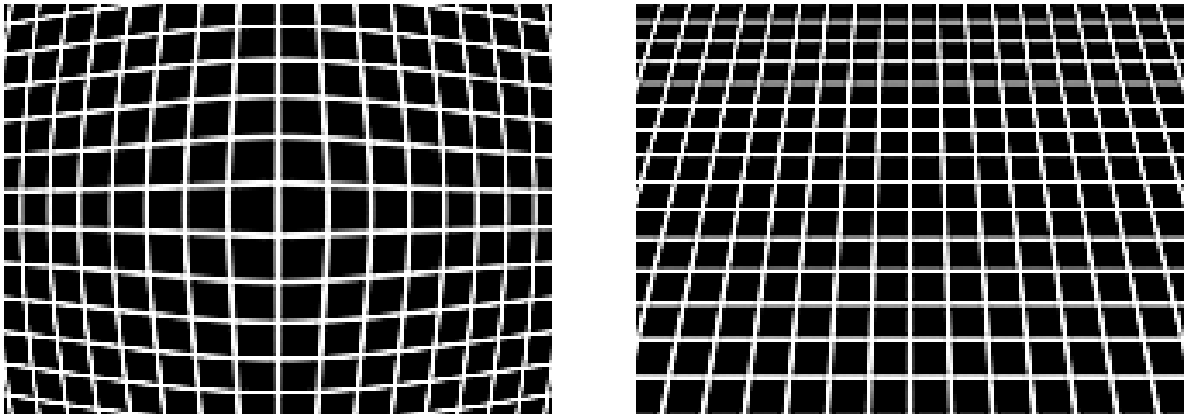

FIGURE 1. RADIAL DISTORTION (LEFT) AND TANGENTIAL DISTORTION (RIGHT)

2.2 Applying the Dewarp Transformation

To correct any lens distortion with OpenCV's method, cameras are calibrated by capturing test patterns to model the distortion. However, capturing calibration patterns using an IR camera is challenging, so using the default matrices characterizing the camera distortion below is recommended. A brief guide on how to take captures of the calibration pattern is described in Section 2.3.

```
camera_parameters = { 'camera matrix': [[104.65403680863373, 0.0, 79.12313258957062],
                                        [0.0, 104.48251047202757, 55.689070170705634],
                                        [0.0, 0.0, 1.0]],
                      'distortion coeff': [[-0.39758308581607127,
                                            0.18068641745671193,
                                            0.004626461618389028,
                                            0.004197358204037882,
                                            -0.03381399499591463]],
                      'new camera matrix: [[66.54581451416016, 0.0, 81.92717558174809],
                                           [0.0, 64.58526611328125, 56.23740168870427],
                                           [0.0, 0.0, 1.0]]
                    }
```

These matrices, along with the input image, are passed in as arguments image into the following OpenCV built-in function to correct the camera-to-image plane mapping. A sample code in Python for demonstration is provided in Section 2.4.

```
undistorted_img = cv2.undistort(img, camera_matrix, distortion_coeff, None, new_camera_matrix)
```

Notice that there is a camera matrix and a new camera matrix. Both represent intrinsic camera parameters; the only difference is that the camera matrix defines the original intrinsic camera parameters whereas the new camera matrix scales and shifts the original camera matrix. Without specifying the new camera matrix in the undistort function, it uses the camera matrix by default, which outputs a rectilinear image having the same IFOV as its input. The tradeoff for obtaining straight lines is that some pixels in the corners of the input image are lost. Applying the new, all pixels version camera matrix shown above provides the option to retain all pixels from the input, but it introduces black pixels around the borders. Figure 2 shows the sample input and the possible outputs.

FIGURE 2. ORIGINAL (LEFT), RECTILINEAR OUTPUT (CENTER), AND ALL PIXELS VERSION OUTPUT (RIGHT)

2.3 Camera Calibration

Since the OpenCV website provides a thorough explanation and sample application code of the calibration process, this application note does not cover the details. The documentation can be found at https://docs.opencv.org/3.4/d4/d94/tutorial_camera_calibration.html.

As introduced in section 2.2, taking calibration pattern captures in IR is not straightforward. Please follow the guide below to take effective captures for IR camera calibration.

1. Setting Up
    - Use a circle grid calibration pattern, as it is not ideal to display a thermal checkerboard. Asymmetrical or symmetrical circle grids work.
    - An 8x8 circle grid is recommended.
    - A paper printout of the calibration pattern glued on cardboard and briefly illuminated with high energy output lights, or the sun can create thermal contrast.

2. Taking the Captures
    - Have at least a 6x6 circle grid visible in the capture.
    - Some circles may be blurred near the image's borders due to the WFOV.
    - Each capture of the circle grid should occupy only portions of the FOV.
    - The collection of captures should span most areas of the FOV, i.e., one or two captures of the circle grid near the center and eight to ten captures around the border of the FOV.
    - Ten to twelve total captures will provide optimized calibration, and additional captures are not advantageous.

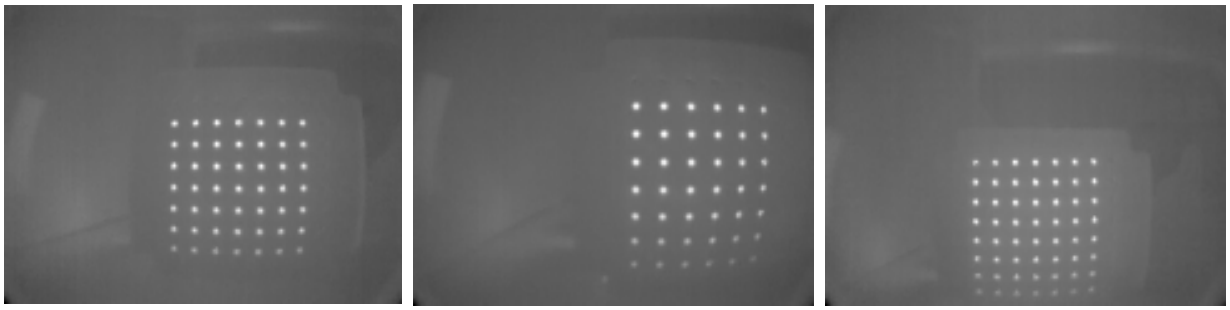

FIGURE 3. CENTER (LEFT), BOTTOM RIGHT (CENTER), AND BOTTOM CENTER (RIGHT) EXAMPLE LEPTON 3.1R CALIBRATION PATTERN CAPTURES

2.4 Support

Sample script *Lepton_Dewarp_example.py*[1] provided at www.flir.com/lepton.

For technical support, visit the FLIR Support Center at http://support.flir.com.