Fisheye Lens Distortion Correction – Notes!

Note: I’ve added a separate doc to the Git repo with all the specifications of the camera, survey, calibration images, etc.

**OpenCV documentation:** <https://docs.opencv.org/3.4/db/d58/group__calib3d__fisheye.html> and (easier to understand I think): <https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html>

* From the OpenCV docs, I think I need to use the undistortimage() function, which ‘Transforms an image to compensate for fisheye lens distortion’.
* At least from the docs here, it seems like this function is a heck of a lot simpler than the others in the docs. I.e., there’s a whole lot less parameters.

**Dotboard note:**

* I’ve used a symmetrical circle pattern (7x7 grid) for my calibration images.
* Dotboard sizes:
  + Dot diameter = 1.5 cm
  + Distance between edges of dots = 1.6 cm
  + Distance between center of dots = 3.1 cm
* Pretty sure we’ll need to use the findCirclesGrid function which has the equations for the dot board I’ve used. You just need the dot board dimensions (7x7). <https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#findcirclesgrid>

**Tutorials:**

* This guy has a made a tutorial ‘not for mathematicians’, which I can appreciate: <https://medium.com/@kennethjiang/calibrate-fisheye-lens-using-opencv-333b05afa0b0>. Main difference I see here is he’s used the standard checkerboard, where I’ve used a dot board (i.e., black dots on silver background). I see that in his Part 1 tutorial, if you run the script as he has written, it crops out ~30% of pixels on the edges of the original images. He points to Part 2 if you don’t want that to happen (which we don’t – we need all them pixels!). Part 2: <https://medium.com/@kennethjiang/calibrate-fisheye-lens-using-opencv-part-2-13990f1b157f>
* Python tutorial: probably going to be really useful: <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html>

**Random note:**

* I figured out a really simple way to work out where the dot board was in my arrays. Because the silver background all has outlying temps, when I did conditional formatting in excel (highlight everything <0) the dotboard popped right up! Ahhh, not everything is that hard… yay.

**My current understanding of the steps we need to do:**

1. Convert calibration CSVs to black & white RGBs. Black dots should stay black. Silver background should be white.
2. Run opencv script(s) to remove distortion from initial TIR RGB images. Output to corrected RGB images. Note, I believe this script needs to run after all the scripts we’ve written so far to get the ‘distorted’ RGB files.

* Note: to keep things simple, let’s run this with the same test batch of ~10 temperature image files.
* Note: the calibration CSV files don’t have a header.
* Question/thought – I am not sure if the fact that we’ve cropped/masked the edges of the original images is going to affect this process, because the cropped array might be a different size/have NaN values on the edges? Is so, we could possibly do this masking after the distortion correction.
* At the end, we should try to save the results of the correction exercise so I can pass this along to others who want to use these results to correct other images taken with this lens. OpenCV suggests saving the results as an OpenCV style XML or YAML file, depending on the extension you give in the configuration file.