# Developing the calibration package and configuration

## Learning the undistortion process

### Lens distortion

The initial problem to solve in the images from the camera is that of lens distortion. This can be seen in the raw image below as the straight lines of the dot grid bulge outwards from the center.

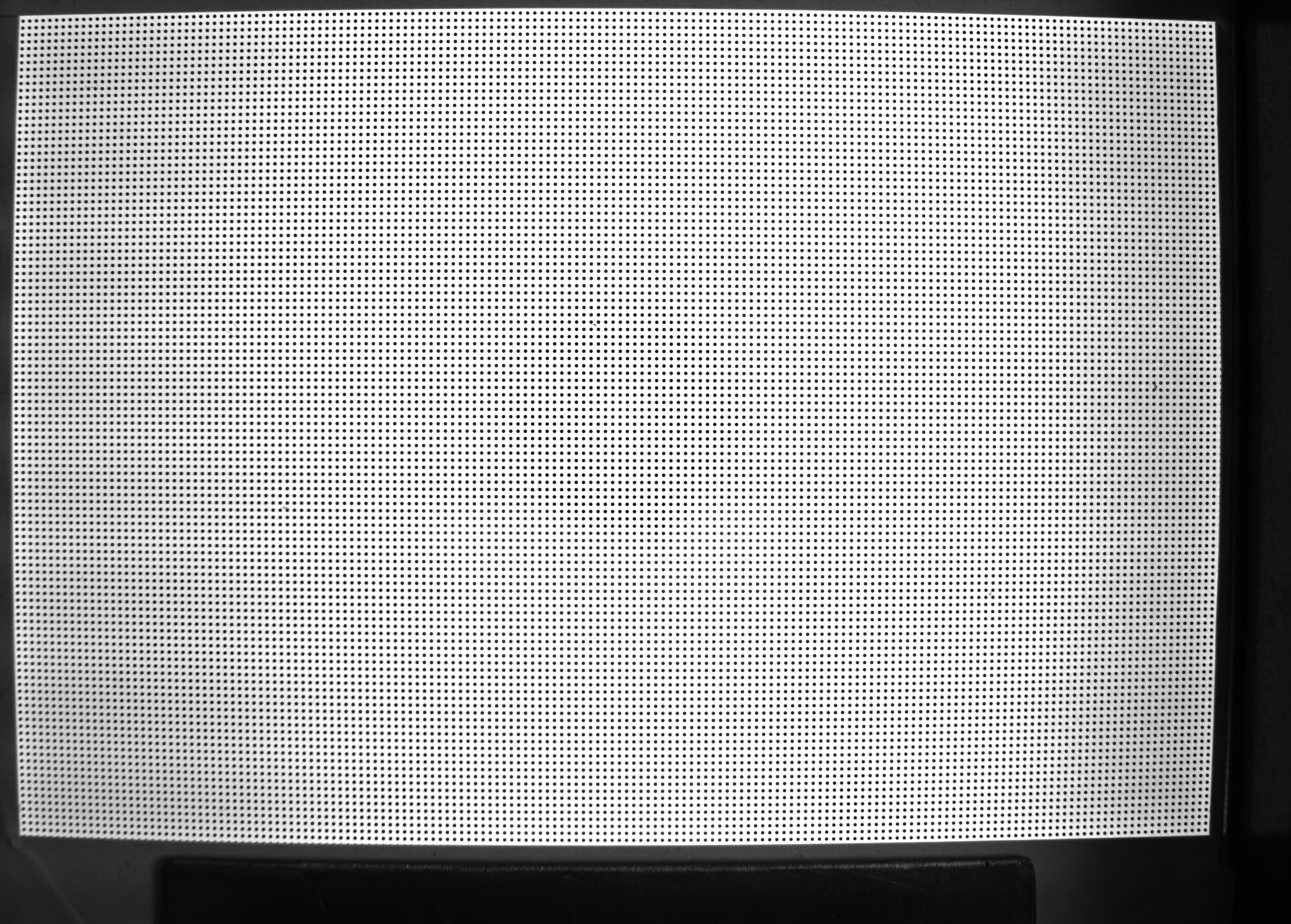


Figure 1: dot grid plate calibration image distcor\_01

OpenCV provides ‘calibrateCamera’ and ‘getOptimalNewCameraMatrix’ functions, which provide values that can be used in the ‘undistort’ function to fix the lens distortion.

However, calibrateCamera requires a grid of points in the image, so first we must detect this grid from the image. Two functions exist in OpenCV for this purpose, one for detecting grids of dots, like the one shown above, and one for grids of corner intersection points in a chessboard pattern.

The simplest of these is the ‘findChessboardCorners’ which takes the image and the numbers of rows and columns in the grid. ‘findCirclesGrid’ also takes the image and grid size, but also flags describing the type of grid search to perform, a blob detector, and a configuration object. This need for the blob detector presents certain difficulties:

* For it to run at a useable speed, it must have filter parameters set to reduce the search space from all possible dots to just the type we are looking for. These parameters must be set up manually.
* The blob detector should be capable of detecting every dot in the grid, excluding any false positives.

After a few tries of fiddling with the blob detectors parameters, a good setup was found that would detect most of the 19,720 dots in the image. By saving a copy of the image with the selected dots highlighted, the issue became clear. To the right of the image, two dots are joined together, and just above the center is an extra blob that was also being selected.

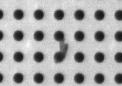


Figure 2: Joined blobs in discor\_01



Figure 3: Extra dirt blob in distcor\_01

To rectify this, I edited these errors out of the images by hand, separating the conjoined dots to leave as round looking remnants as I could.

While these changes meant the blob detector was able to locate every dot in the image, OpenCV’s grid detection algorithm was still not able to detect the grid, presumably due to some combination of the high number of dots, their density, and the severity of the lens distortion, as a point near the center of an edge may lie directly between the ends of the next row out.

To work around this, I considered editing a sparser grid of dots, but decided against that due to the amount of precise drawing it would have taken to ensure certain rows and columns were left untouched. Instead, I decided to manufacture a chessboard image based on the dot grid, using the Inkscape vector graphics editor to draw quadrilaterals with corners that lined up with the approximated center of every tenth dot:

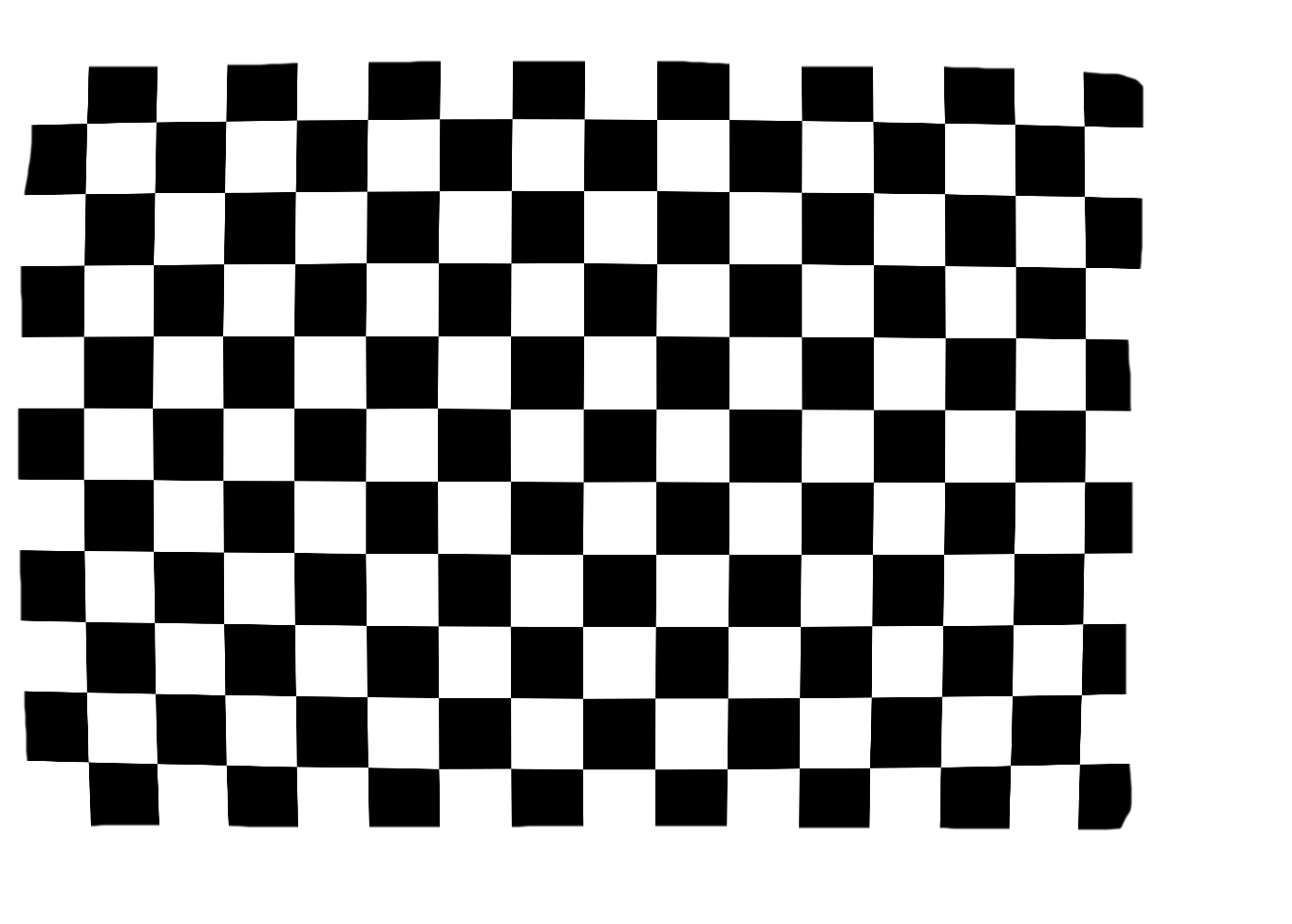


Figure 4: Mocked chessboard image based on distcor\_01

Using this image in the ‘findChessboardCorners’ method was a success, allowing the lens distortion parameters to be calculated. As a test of the blob detector, the blob detector and ‘findCirclesGrid’ method were able to locate the grid in the original dots image if it was undistorted with the lens correction parameters.

### Keystone correction

The correction for lens distortion fixes images so that straight lines in the real world appear again as straight lines. However, if the camera is off axis, then one side of the calibration image will be closer to the camera than the other side, and rather than being a rectangle in the undistorted image, it will appear as a parallelogram.

This can be corrected with a homography transformation, which can also correct for any rotation of the grid, ensuring that the corrected grid is both rectangular and aligned with the x/y axes of the image.

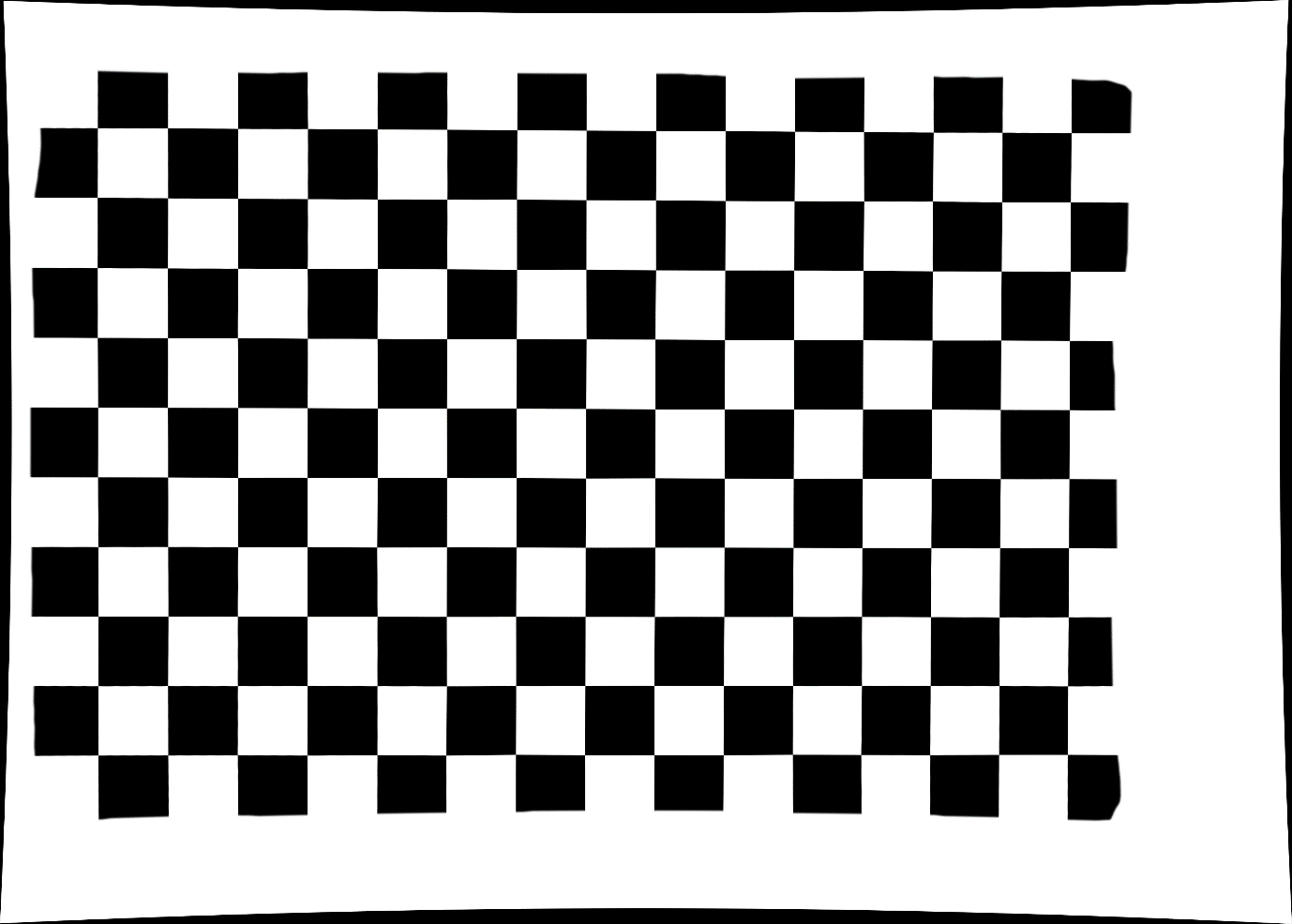


Figure 5: The mocked Chessboard image with lens correction

OpenCV’s ‘findHomography’ method requires a set of points in the image being provided to it, and their corresponding locations in the homography corrected image. The points in the image can simply be the corners of the grid. To prevent any loss of scale or information, I determine which side of the parallellogram has the most pixels per column or row, and use that to calculate the size of the rectangle in the corrected image, with an arbitrarily chosen 50 pixel border.

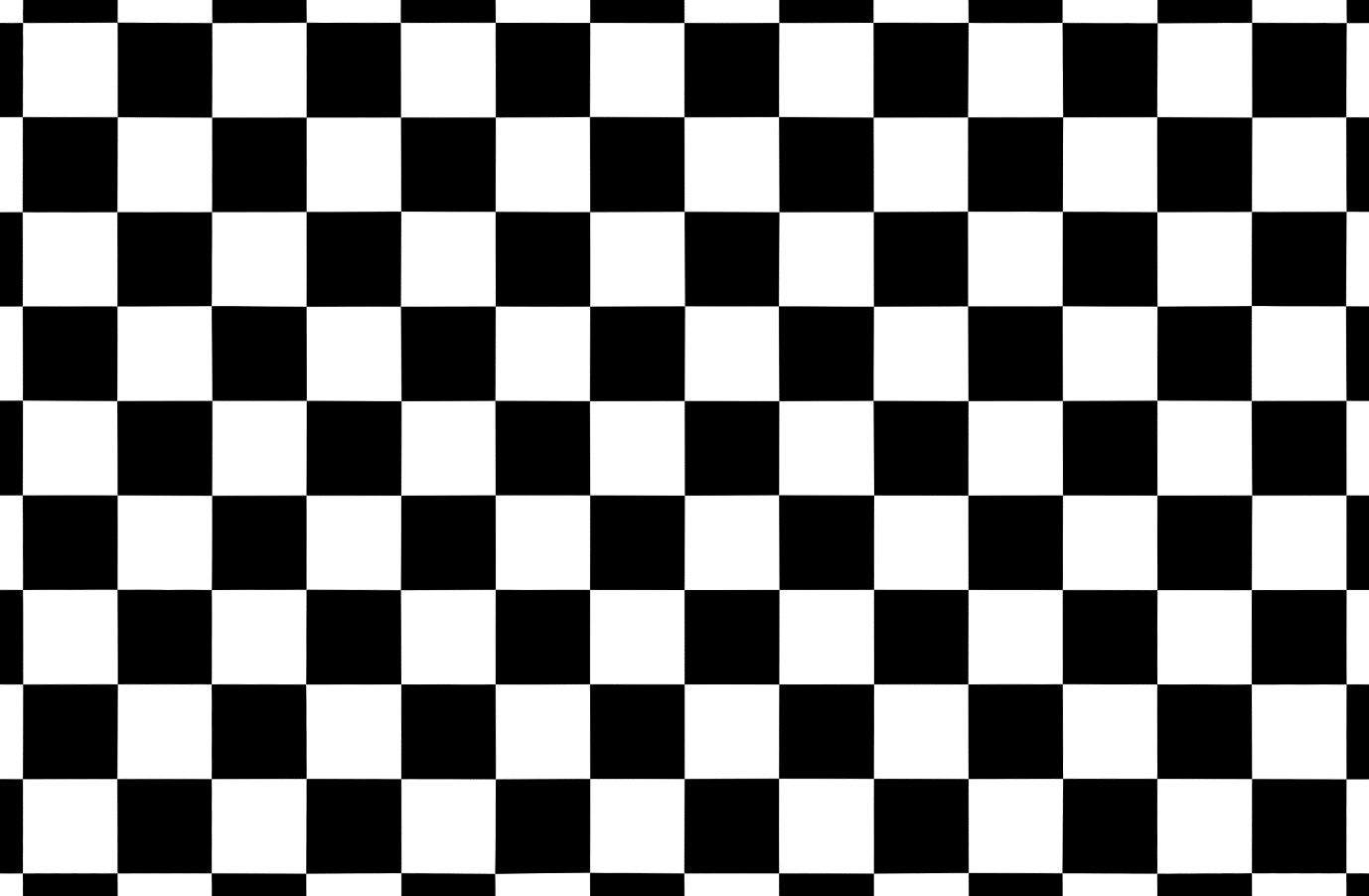


Figure 6: Mocked chessboard with lens and homography corrections applied

### Real world coordinates

Having corrections to transform the image to a regular grid, it then becomes simple to determine a set of real world coordinates for a point in the image. If a pixel’s x coordinate is a third of the way between the left and right sides of the grid, then its position must be a third of the distance between the sides.

## Creating the camera\_calibration package

With the ability to perform the required processing steps, the next task was to wrap this up into a usable product.

Due to the number of parameters:

* Camera matrix
* Distortion coefficients
* Ideal camera matrix
* Homography matrix
* The four grid corners in the undistorted image
* The four grid corners in real world coordinates

These were wrapped up into two classes, Corners and Config, which would be populated by a call to Config.generate() providing a chessboard image to setup lens correction, and a dot image for homography correction. The classes were provided in a package along with two methods, correct\_point and correct\_points, which would accept one or many points from the distorted image along with a Config object, and convert them to coordinates in the calibration grid.

To make the package usable in a wider variety of setups, the generate() method was replaced with a set of methods that could be used to populate the lens distortion and homography parameters separately, and using either a dots image, chessboard image, or an array of the grids coordinates. A correct\_image method was also added, along with an enum for indicating to the correction methods which of the correction steps should be performed.

## Determining the best configuration

To determine the accuracy of a configuration we need a set of points in distorted image space and their expected calibration plane coordinates. The initial configuration I tested used the mocked chessboard to calibrate lens distortion, and the distcor\_01 dot grid to calibrate the homography.

The points being tested were the grid corners, points at the approximate middles of each grid edge, and the center. To determine these accurately, I ran a dot detector on distcor\_01 and filtered the points by the approximate coordinates obtained by manual inspection. As the dot grid has 0.5mm spacing between rows and columns, it was a simple process to determine their positions on the grid in real world units (even if it took me far too long to realise the off-by-one error in my calculations that threw off the accuracy for a long time).

Once the off by one error was fixed, across those 9 points, the average deviation from expectation was 8 microns, with a maximum of 25 microns.

While quite accurate, the range for acceptable maximum deviation was set at 20 microns, and ideally significantly less than that. It also seemed inelegant to be using the mocked chessboard image to produce our calibrations, due to the required time it would take to produce such an image.

So I tried Config’s made with various other combinations of images, including a new set of chessboard calibration images that were taken, with various image adjustments. For these, I used the points from the chessboard grid image as inputs and calculated the expectations based on the grid size. Unfortunately, none of these combinations had accuracy as good as the mocked chessboard and distcor\_01 combo.

Following a discussion with Steven, we came up with another approach. If we undo the lens distortion on discor\_01 using the lens distortion from a chessboard image, we can then detect the grid in the dots image. We can also detect the points in the undistorted distcor\_01, and apply the lens correction on them. Then, for each point in the original image, we can find the location in the grid data structure that it should occupy by finding the location that most closely matches its distortion corrected value. We could then try to determine a more accurate distortion correction using the grid in the original image.

While the process of searching 19,720 times through 19,720 points is quite a slow one, taking 6 minutes to run on my laptop, it works, producing the grid data structure for the original distorted dot grid image. Unfortunately, once OpenCV’s calibrateCamera function was called with this grid it would consume all available memory and still not have completed after 20 minutes (16GB RAM, i7-6700HQ processor running at 3.1GHz). However, by creating a sparse grid that only contained every other row and column, it can complete in less than a minute consuming only 2GB of RAM.

Using this sparse grid derived lens distortion, and the grid corners for the homography transform, an average distortion of 7 microns, with maximum of 19 microns was achieved. By switching the homography to be determined with all points in the dot grid, this would be improved to a 3 micron average and 18 micron maximum. It should be noted though that despite its improvements in this situation, not all configurations are improved by using all points for the homography, so it was made into an argument for the homography property population methods.

To determine the best images to use in this setup, I set up a script to try calibrations using all available dot grid images, in both full grid and corner only homography modes. The best configuration produced used image distcor\_04, with an average deviation of 1.1 microns, and maximum deviation of 10.2 microns.

## Appendix A: Calibration Configuration Accuracies

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Lens correction image | Keystone correction image | Homography points used | Points assessed | Average deviation (mm) | Maximum deviation (mm) |
| Mocked chessboard | Distcor\_01 dot grid | Corners only | Corners, edge middles & center | 0.007744171885450431 | 0.024592731619686663 |
|  | | | | | |
| Brightness/contrast enhanced 002 chessboard | Distcor\_01 dot grid | Corners only | Corners, edge middles & center | 0.014642425080844464 | 0.03434263582352467 |
| Brightness/contrast enhanced 002 chessboard | | Corners only | Full 8x6 chess grid | 0.03930753162714686 | 0.10803360641805183 |
| Mocked chessboard | | Corners only | Full 15x10 chess grid | 0.013251843213749092 | 0.032943701291307234 |
| Otsu thresholded 002 chessboard | | Corners only | Full 8x6 chess grid | 0.033487211566522394 | 0.1295513021319049 |
|  | | | | | |
| Brightness/contrast enhanced 002 chessboard | Distcor\_01 dot grid | Full grid | Corners, edge middles & center | 0.01480831903284257 | 0.06258862664693551 |
| Brightness/contrast enhanced 002 chessboard | | Full grid | Full 8x6 chess grid | 0.029803304046409337 | 0.06034524091309046 |
| Mocked chessboard | | Full grid | Full 15x10 chess grid | 0.012010950182420236 | 0.035468910144781 |
| Otsu thresholded 002 chessboard | | Full grid | Full 8x6 chess grid | 0.02650891977079113 | 0.09703288680574501 |
|  | | | | | |
| Distcor\_01 dot grid | | Corners only | Full 170x116 dot grid | 0.0068365532033461155 | 0.018673387313718398 |
| Distcor\_01 dot grid | | Full grid | Full 170x116 dot grid | 0.0022701536138096307 | 0.017721708969434244 |
| Distcor\_02 dot grid | | Corners only | Full 170x116 dot grid | 0.004218925580149891 | 0.012243381261439354 |
| Distcor\_02 dot grid | | Full grid | Full 170x116 dot grid | 0.0017728766168080838 | 0.01416300054462748 |
| Distcor\_03 dot grid | | Corners only | Full 170x116 dot grid | 0.004020683715625367 | 0.013397027196077464 |
| Distcor\_03 dot grid | | Full grid | Full 170x116 dot grid | 0.001639502352099444 | 0.016362264638251155 |
| Distcor\_04 dot grid | | Corners only | Full 170x116 dot grid | 0.00287922279757079 | 0.011354858920247096 |
| Distcor\_04 dot grid | | Full grid | Full 170x116 dot grid | 0.0010981354562486117 | 0.01018928316974898 |
| Distcor\_05 dot grid | | Corners only | Full 170x116 dot grid | 0.00317002779156954 | 0.010845371042456366 |
| Distcor\_05 dot grid | | Full grid | Full 170x116 dot grid | 0.001215071633446034 | 0.01147723711604149 |
| Distcor\_06 dot grid | | Corners only | Full 170x116 dot grid | 0.005261551785365835 | 0.015087019176068832 |
| Distcor\_06 dot grid | | Full grid | Full 170x116 dot grid | 0.0020601171450275956 | 0.016513938577371812 |
| Distcor\_07 dot grid | | Corners only | Full 170x116 dot grid | 0.004083109193400039 | 0.01278590298858148 |
| Distcor\_07 dot grid | | Full grid | Full 170x116 dot grid | 0.0018219986170447994 | 0.01690539348426784 |
| Distcor\_08 dot grid | | Corners only | Full 170x116 dot grid | 0.00269622930308711 | 0.011706794411879575 |
| Distcor\_08 dot grid | | Full grid | Full 170x116 dot grid | 0.0010995301500154018 | 0.011350326425007346 |
| Distcor\_09 dot grid | | Corners only | Full 170x116 dot grid | 0.0024908190244764318 | 0.011224473643648016 |
| Distcor\_09 dot grid | | Full grid | Full 170x116 dot grid | 0.0011715860677561096 | 0.010530386642667236 |
| Distcor\_10 dot grid | | Corners only | Full 170x116 dot grid | 0.003967508597548494 | 0.014405578262619428 |
| Distcor\_10 dot grid | | Full grid | Full 170x116 dot grid | 0.0016008545701445454 | 0.01743122229457938 |
| Distcor\_11 dot grid | | Corners only | Full 170x116 dot grid | 0.0025916096989949374 | 0.010309196599181789 |
| Distcor\_11 dot grid | | Full grid | Full 170x116 dot grid | 0.0013611496063765836 | 0.010269859369846665 |

## Appendix B: Highest accuracy configuration values

The following code will create a new configuration object that with the computed parameters from the discor\_01 full grid homography configuration. It requires imports of the Config and Corners classes from the camera\_calibration package, as well as numpy’s array and float32.

Config(

distorted\_camera\_matrix=array(

[

[5.4757654619649838e03, 0.0000000000000000e00, 1.8977786697421388e03],

[0.0000000000000000e00, 5.4740252291335883e03, 1.2989798726569404e03],

[0.0000000000000000e00, 0.0000000000000000e00, 1.0000000000000000e00],

]

),

distortion\_coefficients=array(

[

[

-0.2703967509743592,

0.26348057588236573,

-0.00114882690035892,

0.0008420969428869703,

-0.3254696368352247,

]

]

),

undistorted\_camera\_matrix=array(

[

[5.2042651367187500e03, 0.0000000000000000e00, 1.8987904794491624e03],

[0.0000000000000000e00, 5.2145927734375000e03, 1.2942836798625649e03],

[0.0000000000000000e00, 0.0000000000000000e00, 1.0000000000000000e00],

]

),

homography\_matrix=array(

[

[1.0039925372428886e00, -1.3890548470602269e-03, -2.1259054931649462e02],

[-6.2817760571344554e-03, 1.0047590476566779e00, -1.0791976944207451e02],

[-3.2798224330415689e-06, -2.3172919100429813e-06, 1.0000000000000000e00],

]

),

grid\_image\_corners=Corners(

top\_left=array([50.0, 50.0], dtype=float32),

top\_right=array([3599.0, 50.0], dtype=float32),

bottom\_left=array([50.0, 2465.0], dtype=float32),

bottom\_right=array([3599.0, 2465.0], dtype=float32),

),

grid\_space\_corners=Corners(

top\_left=array([0.0, 0.0], dtype=float32),

top\_right=array([84.5, 0.0], dtype=float32),

bottom\_left=array([0.0, 57.5], dtype=float32),

bottom\_right=array([84.5, 57.5], dtype=float32),

),

)