Rich Juszkiewicz
EEN 538 – Digital Image Processing
10/25/05

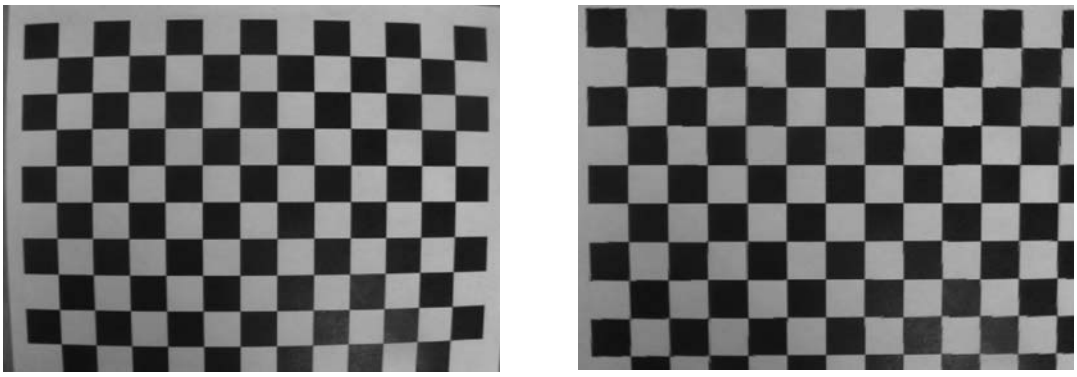
Project 3 – Image Distortion Correction

Abstract

The images captured from low end digital cameras are often geometrically inaccurate due to the low quality of their lenses. This distortion can either be of the barrel effect or of the pincushion effect. The barrel effect gives the image an inflated appearance—the image looks as if it is on the edge of a blown up balloon. Vertical lines that are supposed to be straight have an outward curvature to them. An image suffers from pincushion distortion if the opposite is true: vertical lines have an inward curvature to them. Both of these types of distortion, while they do affect the entire image to some extent, are more apparent at the extreme edges of the image. This paper examines barrel distortion, in particular, by devising and implementing an algorithm in MATLAB to rectify it.

Introduction

In order to correct barrel distortion two images of a checkerboard grid are needed. One image, of a perfect grid, is used as a reference image and the other is of a grid exhibiting barrel distortion as outputted by a camera with lens distortion. In this particular experiment the perfect grid is not entirely perfect but its imperfections are slight enough to allow for accurate testing of the proposed image distortion correction algorithm. The two images used are shown in figure 1. The slight off centeredness of the perfect image is visible on the right edge of this image. Please note that the images in figure 1 are not shown at full scale. Their full resolution of 990x730 was used throughout the experiment but is not practical for display purposes.



(a) The distorted checkerboard image

Figure 1

(b) The reference checkerboard image

Algorithm:

Since the distortion is greatest at the edges of the image and lesser towards the center applying a global mathematical model to repair the non-uniformly distorted image would be very complicated. It may be possible to design a global model using some sort of high order prediction algorithm and/or a differential equation to account for the change in distortion from edge to center and then back to edge again but such methods are beyond the scope of this paper.

Instead, a relatively simple bilinear model was applied locally at the four corners of every square in the grid. This model would not work globally because the barrel distortion is not constant throughout the image. Each corner has a pair of four variable bilinear equations associated with it that link corresponding points from the reference image to the distorted image. One equation is for the x-value of the point and the other is for the y-value of the point. These equations are shown below in equation 1. They are bilinear because of the last term which multiplies x_i and y_i, thus making the equation dependent on both x and y.

$$\hat{x}_i = p_0 + p_1 * x_i + p_2 * y_i + p_3 * x_i * y_i$$
$$\hat{y}i = q_0 + q_1 * x_i + q_2 * y_i + q_3 * x_i * y_i$$

Equation 1: The bilinear mapping functions in which the p's and q's are the unknown scaling coefficients and the x's and y's are the known coordinate values of the corners being mapped.

In these equations $\hat{x}_i$ and $\hat{y}i$ represent points on the distorted image while x_i and y_i represent points on the reference image. The subscript i varies from one to four based on which of the four corners is being addressed. The numbering scheme used in this paper is shown in figure 2.

Figure 2: Square illustrating the numbering scheme used

Having four corners on each square of the grid yields four $\hat{x}$ equations and four unknown variables (p_0, p_1, p_2, p_3). This is all that is needed to solve the system of equations. In order to solve this system of equations efficiently linear algebra

is used. A coefficient matrix is setup for the four equations (i=1,2,3,4) and is multiplied by a column vector of length four representing the unknowns all of which is set equal to the corresponding values on the distorted image. The matrix equation used to compute the p values is shown below in equation 2.

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1 * y_1 \\ 1 & x_2 & y_2 & x_2 * y_2 \\ 1 & x_3 & y_3 & x_3 * y_3 \\ 1 & x_4 & y_4 & x_4 * y_4 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \end{bmatrix}$$

Equation 2: Matrix equation used to solve for p's.

Since most of the underlying computation in MATLAB exploits various aspects of linear algebra this equation is solved rapidly because it is already in matrix form. An analogous equation is used to solve for the unknown q values. The column vector containing the p's is swapped out for a column vector containing the q's and it is all set equal to the y values of the distorted image. This is shown below in equation 3.

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1 * y_1 \\ 1 & x_2 & y_2 & x_2 * y_2 \\ 1 & x_3 & y_3 & x_3 * y_3 \\ 1 & x_4 & y_4 & x_4 * y_4 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{bmatrix}$$

Equation 3: Matrix equation used to solve for q's.

Since the equations are set equal to the points in the distorted image ($\hat{x}_i$ and $\hat{y}i$) the mapping function generated from solving this system will model the distortion rather than correct it, which is interesting to examine so this process is carried out first and then the variables are later switched so that the mapping function corrects the distortion.

Once all of the p and q values are known two mapping functions are generated- one for the x values and one of the y values, respectively. Knowing these p and q values allows for the mapping of all of the points located inside of the four corners from the reference image to the distorted image. For example, the location of any pixel inside of the four boundary points (the corners) is plugged into equation 1 along with the newly acquired p (or q, if it is the y-value of the pixel being calculated) values and the location of that pixel on the distorted image is found.

However, this new location will more than likely not be an integer because the p and q values are real. Simply rounding the value to the nearest integer is not a proper way to account for this so interpolation must be used to find that pixel's

corresponding location and intensity on the reconstructed image. (See the results section to see the effects of rounding rather than interpolating).

The interpolation method used is also bilinear. The equation that this interpolation is based off of is the same as equation 1 and is solved in a similar way so to avoid redundancy it will not be discussed here. The four integer values nearest to the real valued point are used in calculating the interpolated value.

Implementation:

Corner Detection:

A method for automatically finding corner points was designed after the mapping and interpolation functions were tested and verified to be functional on a few squares of the image. Having a corner detection algorithm prevents the user from having to tediously select every corner point manually and it is also more accurate.

The ideal end result of this algorithm is to have a binary image that is black every where (binary 0) with white dots (binary 1) the size of one pixel at the locations of every corner of the image. Erosion is used in conjunction with some morphological operations to get the desired result.

Implementing this algorithm requires converting the image to binary; this is the first operation done on the image. MATLAB's *graythresh* and *im2bw* functions were used to do this. Once the image was reduced to only two colors the edges of it were then detected using the *zerocross* method of MATLAB's *edge* function. The *zerocross* method yielded the cleanest and most usable results. All of the other methods available were tested before it was determined that *zerocross* was the best. It produced lines one pixel wide that outlined the boundaries of every square on the grid.

The erosion process is carried out by creating a structural element in the shape of a cross. Two nested for-loops were used to generate a cross that consists of two intersecting three pixel wide lines. The image to be eroded, after edge detection, has intersections of only one pixel wide so the edges must be enlarged before the erosion process will be successful. (Note that erosion with a cross of one pixel wide everywhere was tried with the output from the edge detection filter and it was not successful mainly because not all of the intersections are well defined crosses when every line in the image is only one pixel wide.)

To get the lines thickened, and in useable form, various operations in MATLAB's *bwmorph* function were used. The order in which these methods were applied proved to be critical. The first method used was *thicken*, it was given a parameter of three which increased the thickness of every line in the image from one to three. It may seem that this is all that is necessary to set up the image for

the erosion process but there are some small gaps in the thickened lines--especially at corners, which are the points of interest. To fill in these gaps three methods of *bwmorph* are used, the first two used three as the input parameter while the third doesn't require an input parameter. The first of these three was the *bridge* method, the second was *fill* and the last was *dilate.*

Finally, after all of this the erosion process was successful; it left dots at all of the interior corners in the image. These dots were larger than one pixel in size but their shapes were perfect circles so the *shrink* method of *bwmorph* was used with inf as the input parameter. This shrunk all of the dots down infinitely until they couldn't get any smaller which left one white pixel at every interior corner of the image. Detecting only interior corners was not quite the desired result. This process only detected interior corners because the corners on the edges of the image don't form a cross so the erosion process failed at those points. Perhaps if the shape used in erosion was an "L" it may have yielded better results but instead the border corner locations were manually programmed in. This entire corner detection process is carried out on both the reference grid and the distorted grid because the corners of both images are necessary for the solving of equation 2.

Store Corners in Arrays:

Once the two images containing all of the corner locations are created the values of the corner points must be stored orderly for easy access. To do this a series of independent for-loops are used. There is one for loop for each of the thirteen columns in each image. The for-loops detect every point in each column from top to bottom in the order that they appear and store their locations into an array. The x-value is stored in column one of the array and the y-value is stored in column two of the array. At the end of the thirteen loops all of these arrays are concatenated to form one large array for each image. Each of these arrays contains all of the locations of the corner points in the order that they appear in the image. This makes accessing the corner points when they are needed by the program quick and intuitive.

While loop:

The rest of the implementation of this algorithm is inside of a while-loop. The contents of this while-loop include the calculation of the mapping functions, the calculation and interpolation of the locations and their intensities inside of each square and the constructing of the new image from all of the aforementioned calculations. A while-loop is used to iterate through each of the squares in the images by using the arrays that the corner points are stored in. For every square in the image a mapping function is calculated from its four corners and then using a for-loop all of the points inside of the four corners are plugged into the mapping function and their mapped locations are obtained after interpolation. At the end of this for-loop all of these points are mapped onto a new image. The while loop

terminates at the last four corners of the image (the bottom right of the image) thus completely reconstructing a new, altered, image. MATLAB's *ceil* and *floor* functions are used in the interpolation calculations to find the integer values both above and below the real value calculated by the mapping function.

Results:

The barrel distortion program described above was first run by mapping the reference grid to a distorted grid. The output is shown in figure 3.

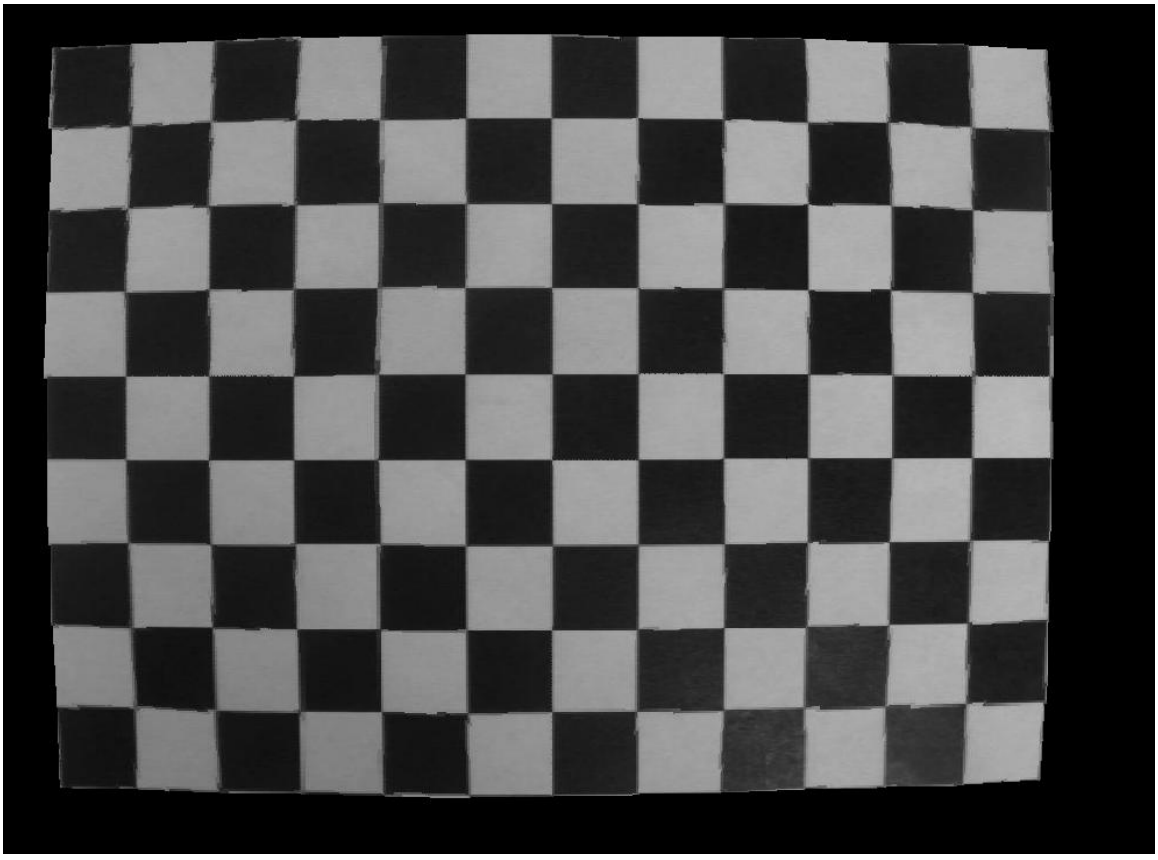


Figure 3: The output obtained by distorting the reference grid.

Figure 3 illustrates that the program correctly models barrel distortion because it outputted a distorted version of the perfect grid. The purpose of this paper is to correct barrel distortion but it must first be correctly modeled before it can be corrected.

To test how well the program corrects barrel distortion it was run in inverse mode with the image in figure three as the input to the program. The output of this is shown in figure 4. To illustrate the necessity for interpolation figure 5 shows what figure 4 would look like with simple rounding used instead of interpolation.

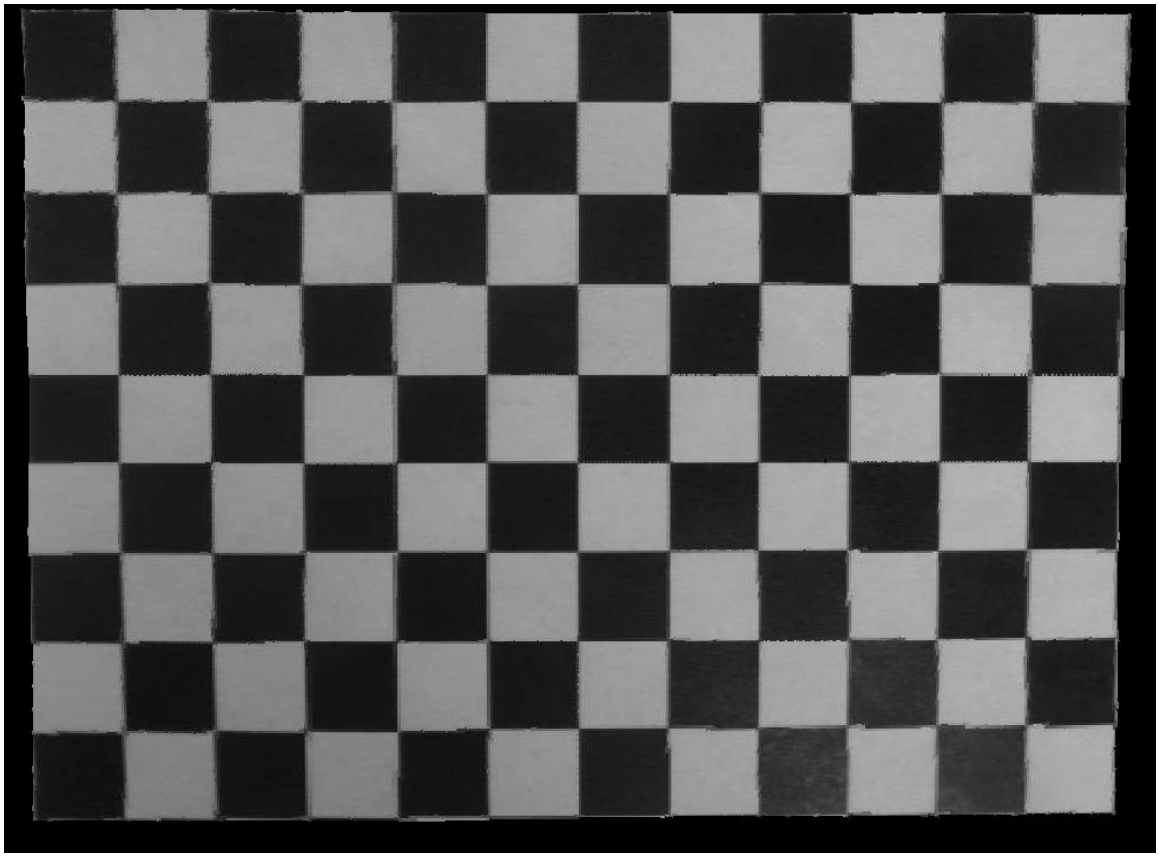
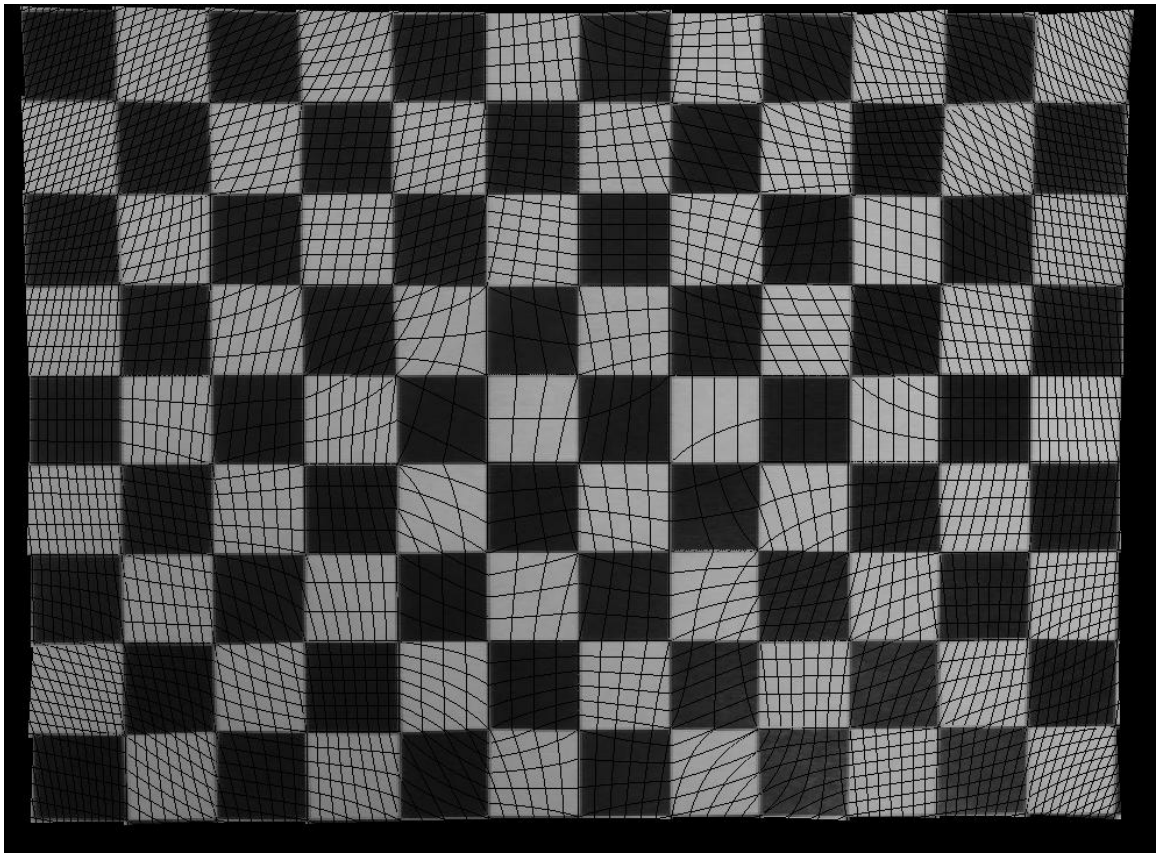


Figure 4 (above): The output obtained by correcting the barrel effect generated in figure 3.
Figure 5 (below): The output of the program with rounding implemented in place of interpolation



To further test the robustness of the geometric distortion aspects of this program an image generated in Photoshop was used. The image's size is 990x730, the same as every other image used in this experiment, and it consists of a perfect white rectangle drawn on a black background. This image is not very complex but if it distorts correctly it will be visible. Keep in mind that barrel distortion is more prevalent on the edges of the image and since the rectangle is located in the center of the image it won't distort too much but it should still be noticeable. The original rectangle image is shown in figure 6 and the rectangle image after running through the programs distortion model is shown in figure 7.

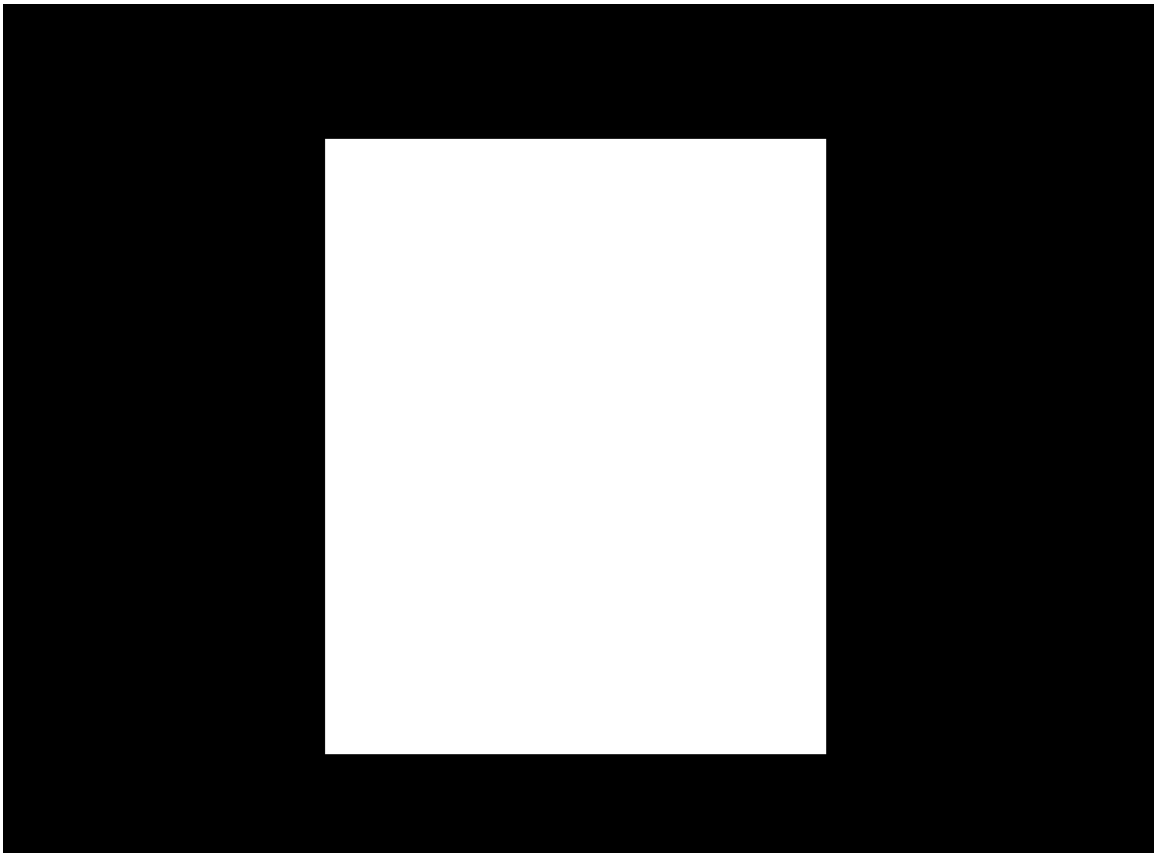


Figure 6: Rectangular image generated in Photoshop

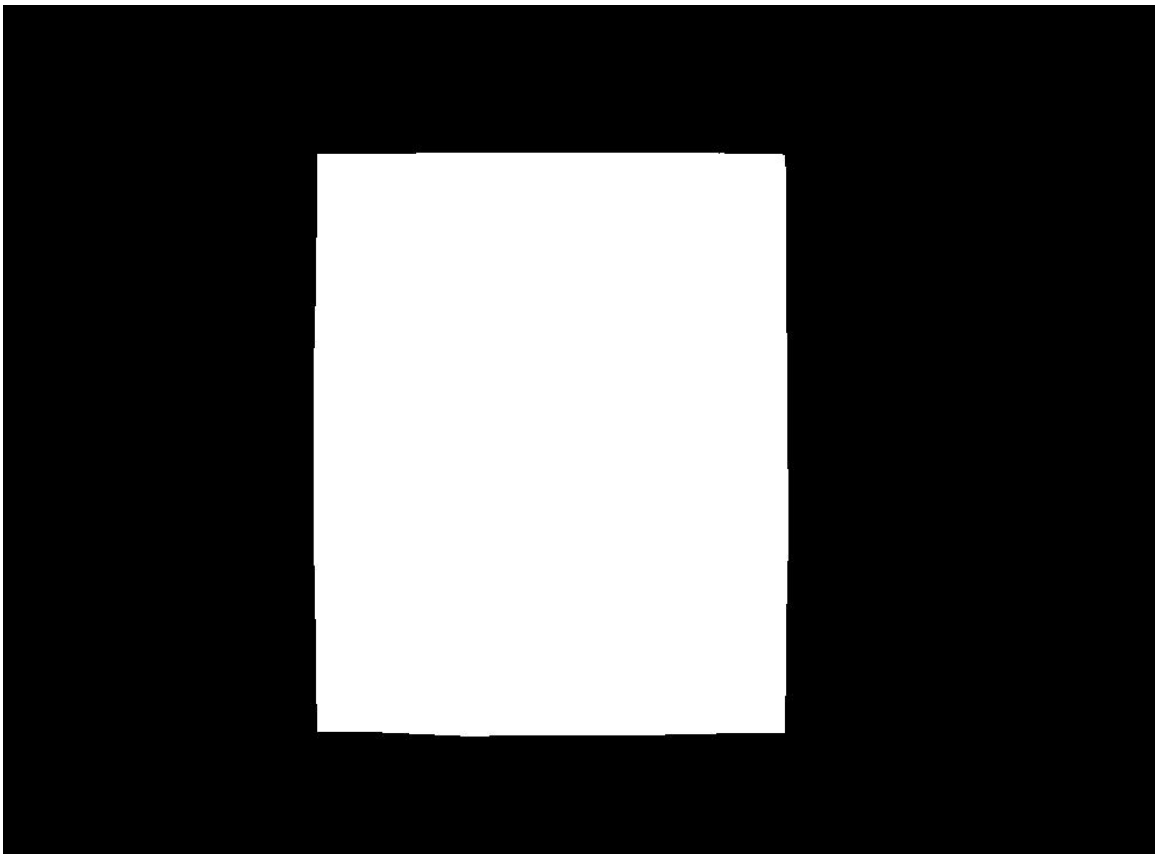


Figure 7: Rectangular image from figure 6 after being distorted by the program

It is evident from comparing figures 6 and 7 that the program accurately models barrel distortion. Since the reference grid used to establish the mapping functions and points is somewhat off center the rectangle's location is shifted a bit. This is one of the imperfections mentioned in the introduction section. Nonetheless a slight bulge is visible in figure 7 which illustrates the effects of barrel distortion. The image in figure 7 was run through the barrel distortion correction program and the output is shown in figure 8.

Note that aside from the small amount of aliasing and minor off-centeredness (due to the imperfections in the reference image in figure 1) the image of the original rectangle was accurately restored. This further proves that the barrel distortion correction program operates correctly.

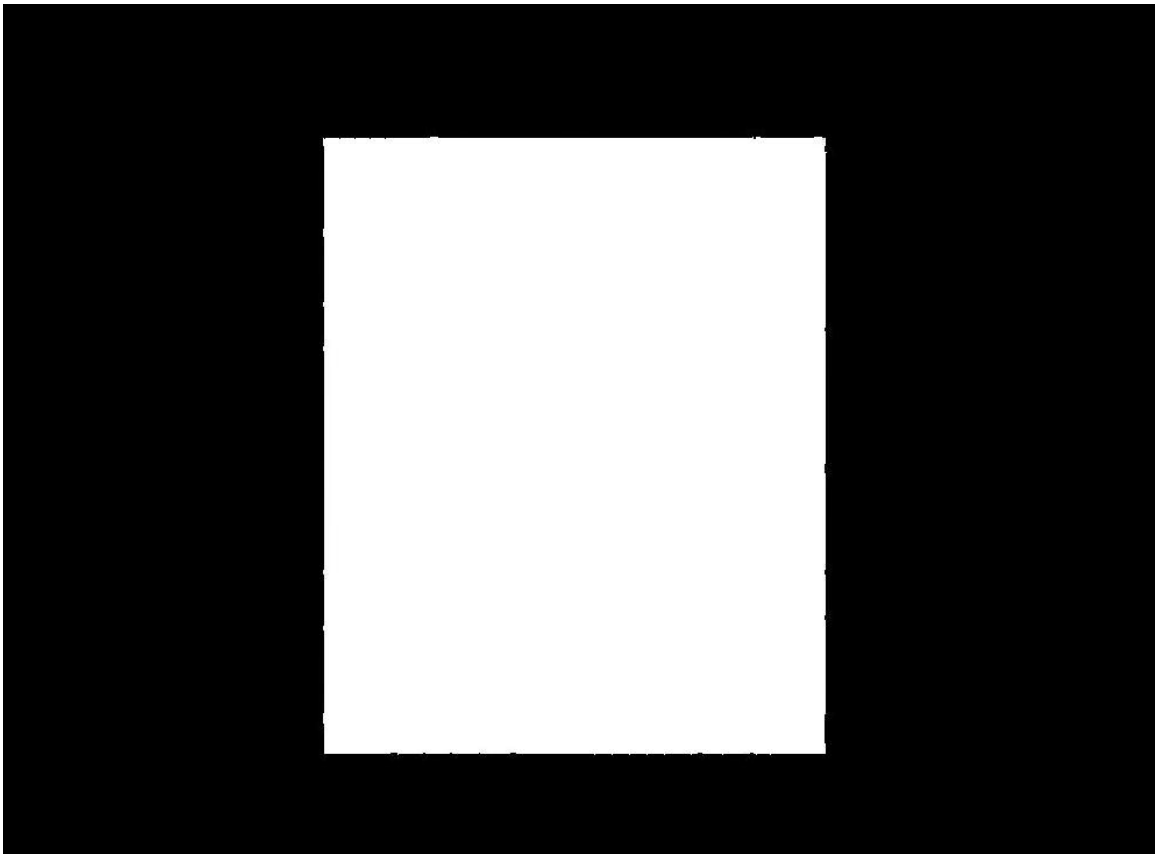


Figure 8: Rectangle image from figure 7 after being corrected.

To test the effectiveness and accuracy of the interpolation method used histograms of various images are used. The histogram of the reference image is shown in figure 9, the corrected image in figure 10 and the difference image (figure 9-figure 10, or reference image – corrected image) in figure 11. In figure 10 there is a line at zero (along the y axis) that extends above 16000 that cannot be is not visible in the figure. This abundance of values at zero is because of the black border around the image that is outputted by the program. Since there aren't full squares at the edges of both the reference image and the original distorted image the program cannot compute the data that belongs in the black space. An extrapolation process could be developed to account for this but that was not the focus of this paper.

It can be seen from comparing figures 9 and 10 that the histograms of the images have very similar shapes. The difference in the peaks of the populated regions can be accounted for due to the lack of data around the edges in the new image and because of slight interpolation discrepancies around the edges of each square (see below and figure 12).

The difference image that figure 11 is constructed from is show in figure 12. The first and obvious difference in the two images is once again the data near the edge of the image. The gray values around the edges that are present in the reference image and not the reconstructed image vary from 125 to 175 thus

accounting for the presence of data in this range in figure 11. The discrepancy of border data is also responsible for the large amounts of zero valued pixels in figure 11.

If the reference image and the image created by the program were both cropped in the same exact ways at the same pixels to eliminate the border that is not calculated within the program then the difference histogram would only show the interpolation discrepancies thus eliminating most of the non-zero values present in figure 11.

Aside from the border the only other differences are around the edges of where squares are present in the reference image. This is because the interpolation of bordering squares often overlaps and the one that is calculated second is the one that remains even though it may not be the correct value. Other than these small differences the new image is very close to identical (in all areas that were computed) to the reference image.

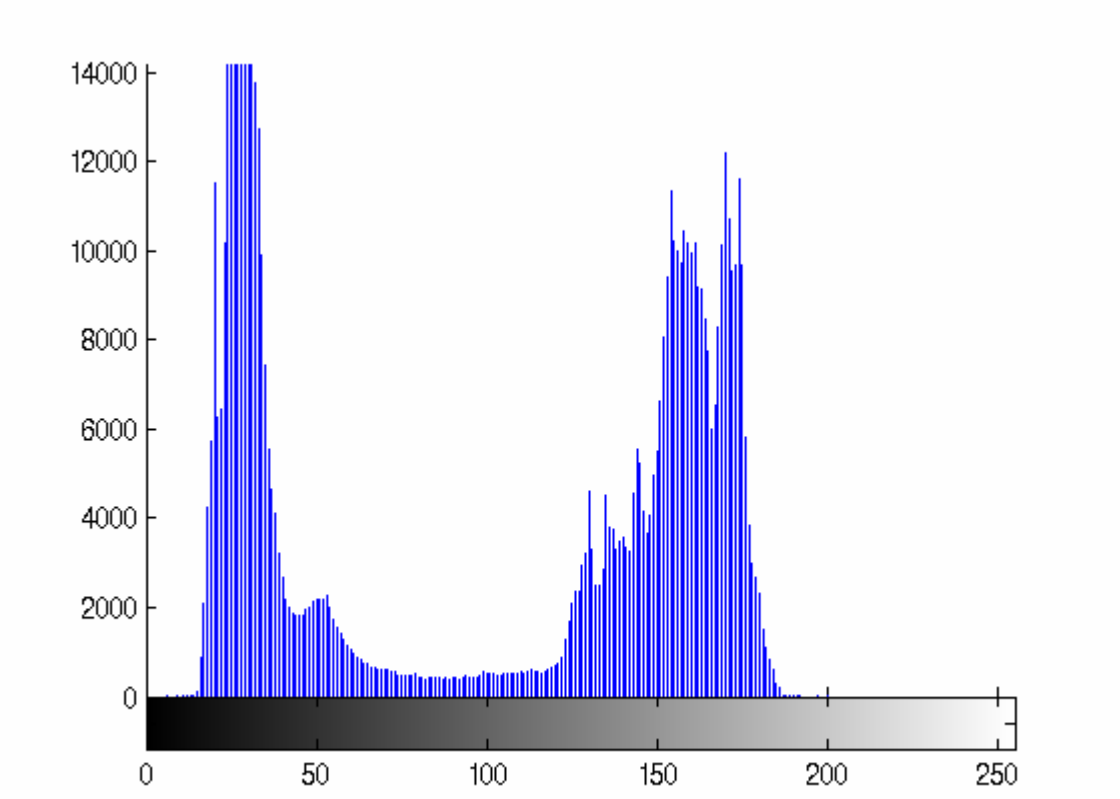


Figure 9: Histogram of the reference image.

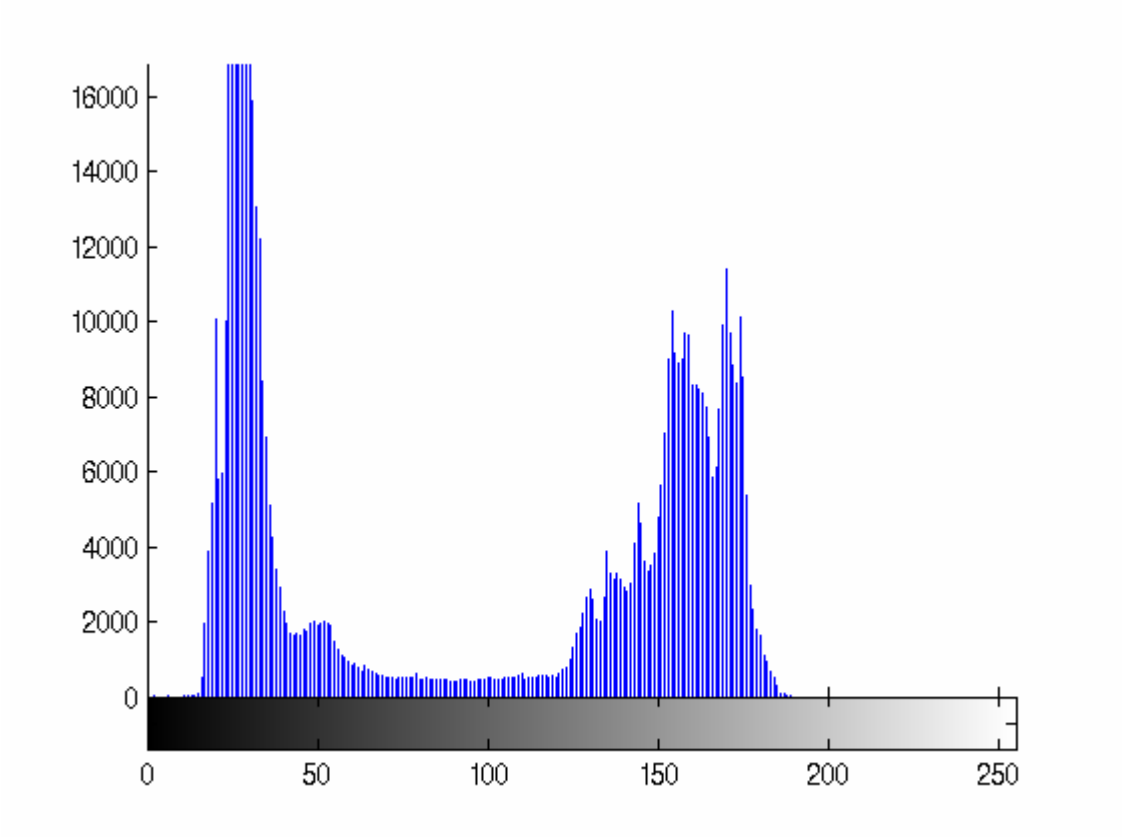
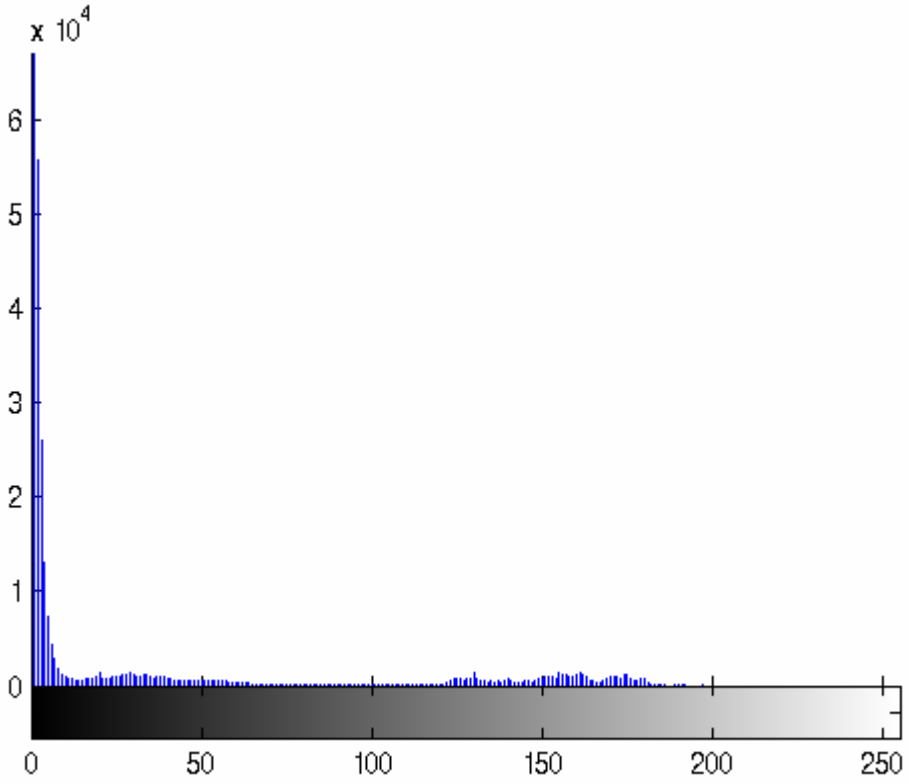


Figure 10: Histogram of the image created by the program

Figure 11: Histogram of the difference image



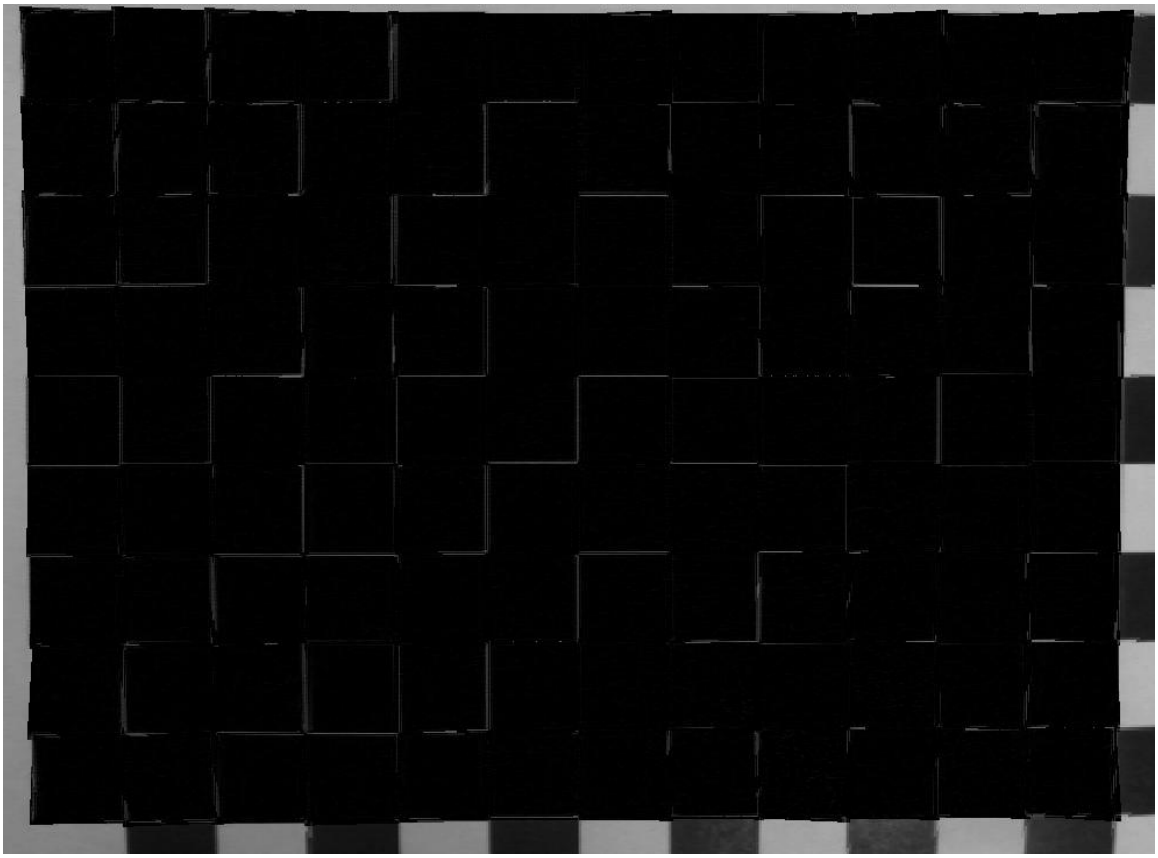


Figure 12: Difference image used to create figure 11. The intensity values of the lines inside the image range between 20 and 50.

Conclusion:

While this method for image distortion correction was proven to be mostly successful there is still some supplementary work that can be done to further improve this procedure. The first, and most obvious of these things, would be to get two images (distorted and reference) that are better centered and more corresponding than the two in figure 1. Another improvement to this procedure would be to come up with an extrapolation algorithm to account for the pixels near the edges of the image that are not a part of the calculation. A better corner detection algorithm could also be designed that eliminates the need to enter the outermost corner points into the program manually.

As for the discrepancies around the squares due to interpolation an algorithm could be written to check for the correct value before the pixel is falsely overwritten. Aside from the minor corrections discussed above this procedure is very successful.