ME 59911 – Experimental Methods in Fluid Mechanics and Combustion

Image Processessing

Yiannis Andreopoulos, PhD

andre@ccny.cuny.edu

February 28, 2020

ME 59911 – 4ST

Student name: Gurpreet Singh

ID: 23403184

The City College of New York
Mechanical Engineering Department

My phone: iPhone 11 Pro Max

Overview

The objective of this assignment is to be able to take an image and use it to determine the calibration matrix of your cell phone. As many people know, cameras have identifying features which makes some better than others in terms of taking pictures of quality. The higher the megapixels are on a camera, the better it will look and more pixels it will display on the screen. For example, a 3 megapixel camera, MP for short, will only capture half of the detail and pixels of that of a 6 MP camera. This is because of the resolution of which images are taken and the pixels per inch. Using what I know about my camera phone, I will calculate the calibration matrix of my phone by designing a calibration plate and importing these files onto MATLAB.

Procedure with Results/Discussion

I had difficulty designing my first 2D calibration plate as I had no printer to use at home. This forced me to hand draw my plate using copy paper. While I wanted to be as accurate as possible, I knew that I could not be as accurate as a printed copy. I decided to use graphing paper and copy paper for this step. I know that the boxes in the graphing paper are all 1 cm x 1 cm, and this allowed for me to space out my dots evenly. Using the graphing paper, I placed it on top of the copy paper and decided to press hard against the graphing paper and get the imprint onto the copy paper to make an accurate plate of dots. Before deciding how many dots to do, I recalled that it's better to use more dots to get a more accurate representation of the calibration matrix even though a small amount of dots would be enough. I decided to draw a matrix of 11 x 11. The results are seen below.

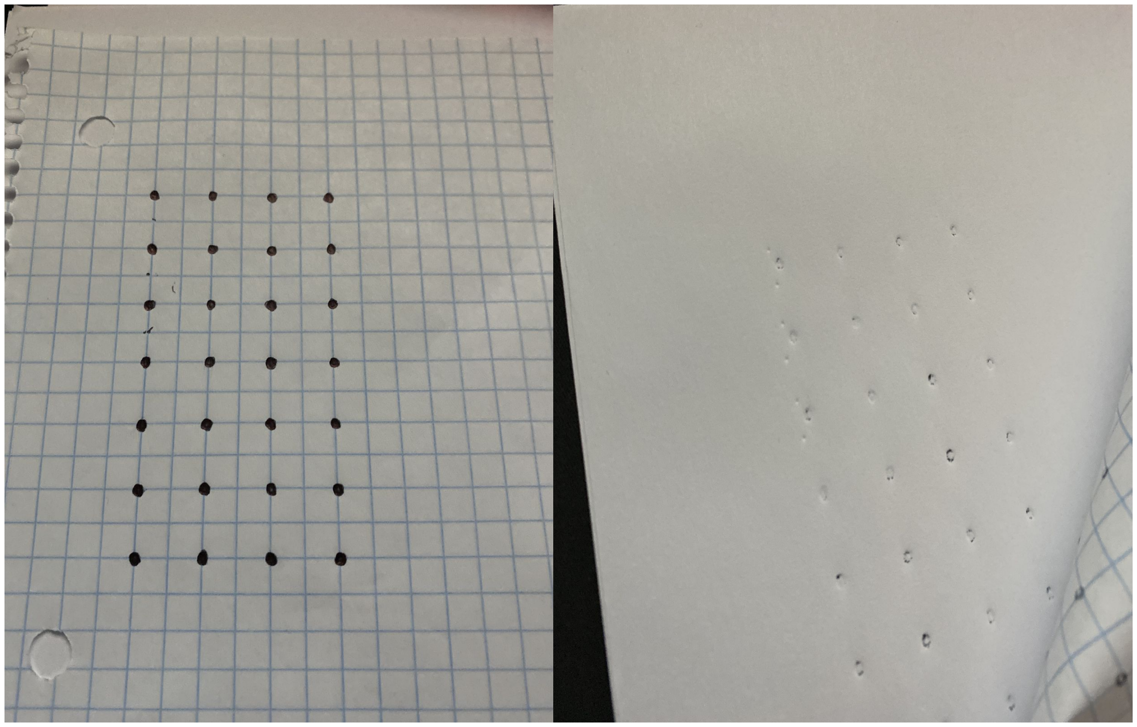


Figure 1: Design of Calibration Plate

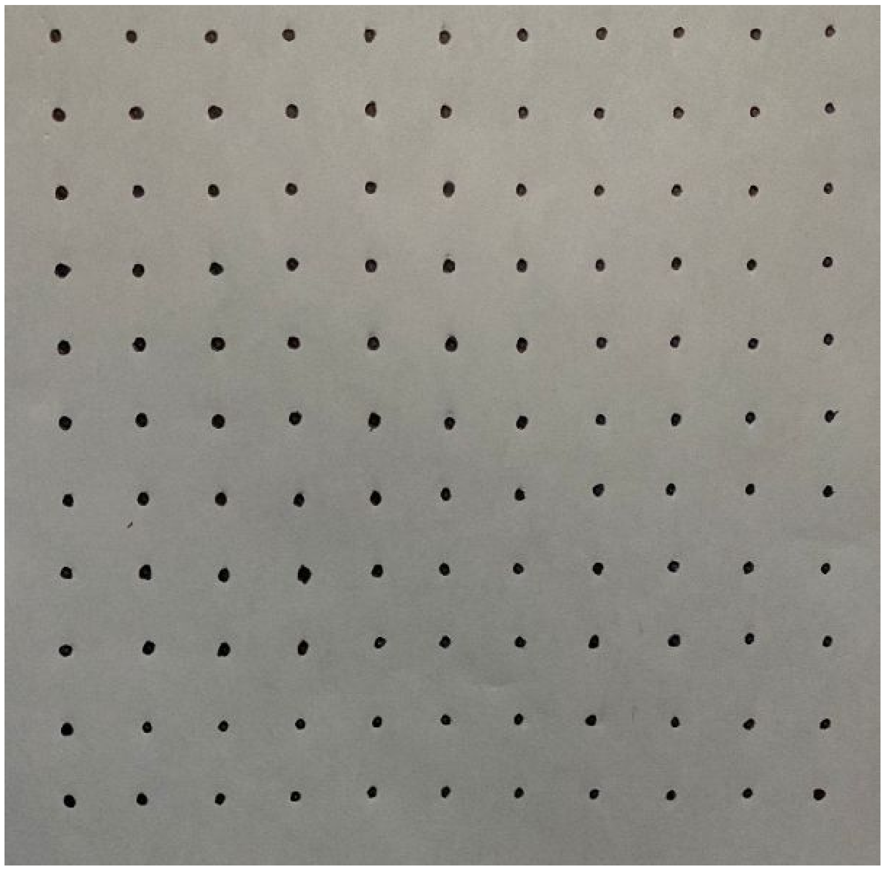
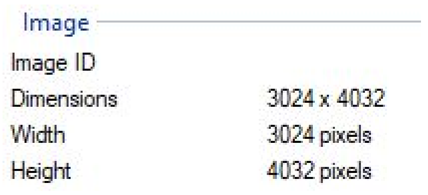


Figure 2: 11 x 11 2D Calibration Sheet

Now that I have taken a picture with my camera, I have to import the image onto my computer and to make sure that I don't reduce the size of the image, I decided to transfer it directly with a USB cable rather than sending it over an application. I then used the properties tool and verified the quality of the image so that I can move onto the next step. I had seen that my photo was 3024 x 4032 resolution and knew that my camera is 12 MP. I decided to verify this by multiplying these numbers.



Image

Image ID	
Dimensions	3024 x 4032
Width	3024 pixels
Height	4032 pixels

Figure 3: Photo dimensions of Calibration Plate picture

$$3024(4032) = \frac{12192768}{10^6} \approx 12\,\mathrm{MP}$$

I have the **iPhone 11 Max Pro** and was confused since my phone has 3 cameras. After further online research I had noticed that only one of the cameras is in use if you're taking a non-zoomed in image. Therefore I was satisfied with the result above. Using the same image, I had to convert it to grayscale in order to make sure that MATLAB would be able to detect all of the black dots on the paper and disregard all of the other streaks or non-binary colors which may appear on the sheet. I did this by using an online converter which directly allows me to put the image and redownload it as a png file. Afterwards, I decided to put this grayscale png image into another converter which allowed for me to convert the extension from .png to .tiff. This is an important step because .tiff files do not lose quality or clarity when they are edited many times.

Now that I have the .tiff file ready, I was able to use the MATLAB code (Cal1_1FEPThin2.m) provided and rename my file and the code to where it would read into the script. There was a portion of the code of which I had to adjust where I had realized it was set for a 9 x 18 matrix and had to adjust it so that it was set for a 11 x 11 matrix. Additionally, I had to change the dX and dY values with respect to the spacing between my plots. Since my dots were 20 millimeters apart, I used the value 20 for both of these variables. Afterwards, I was able to run the code and received the following results:

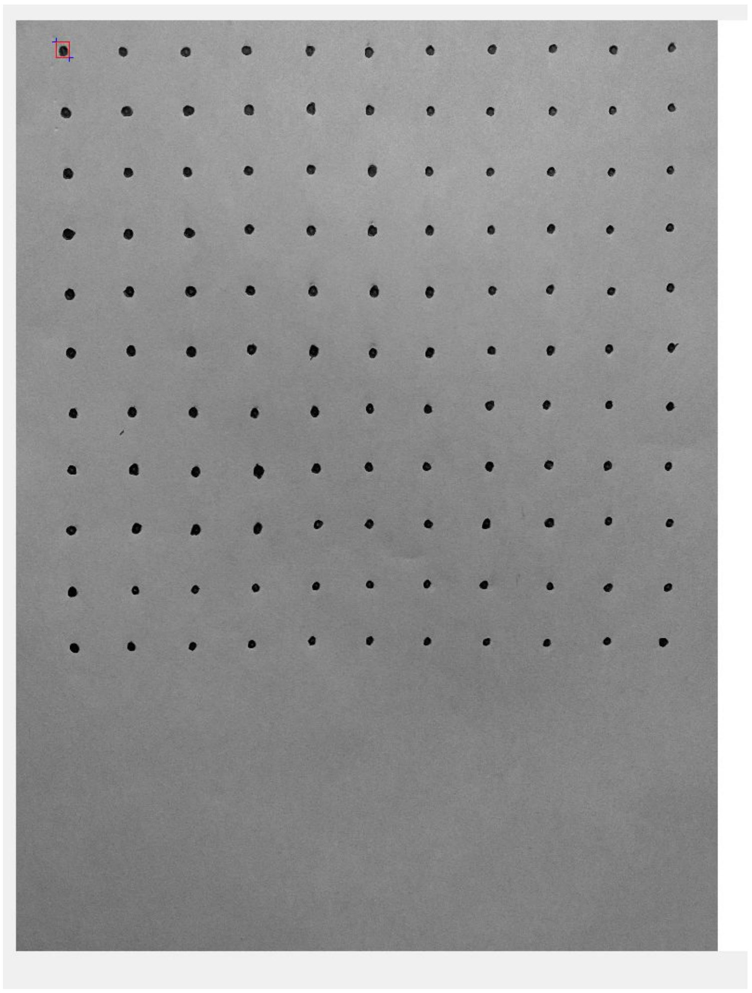


Figure 4: Selecting portion to scan for along the .tiff file

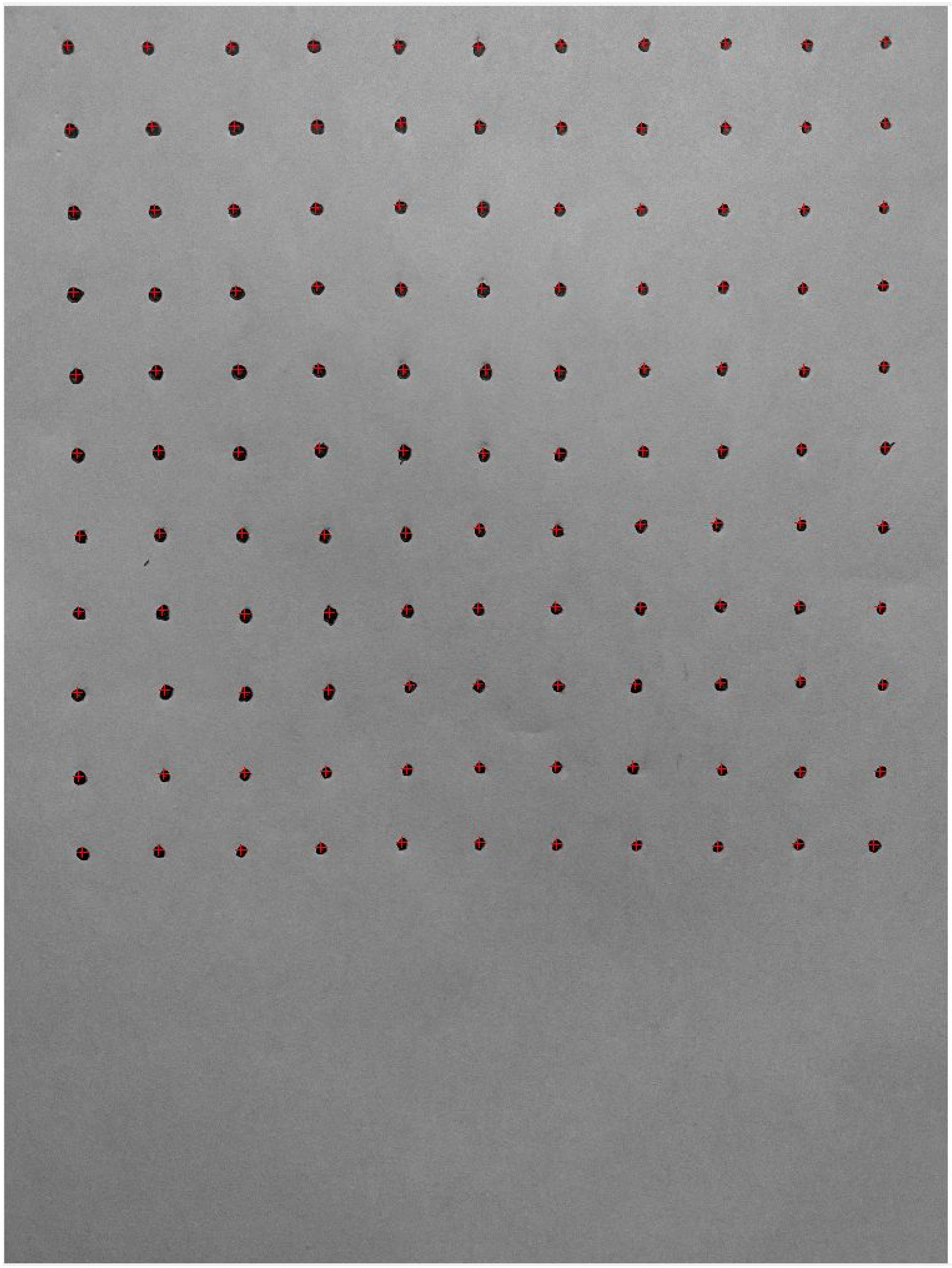


Figure 5: All points on the .tiff file recognized

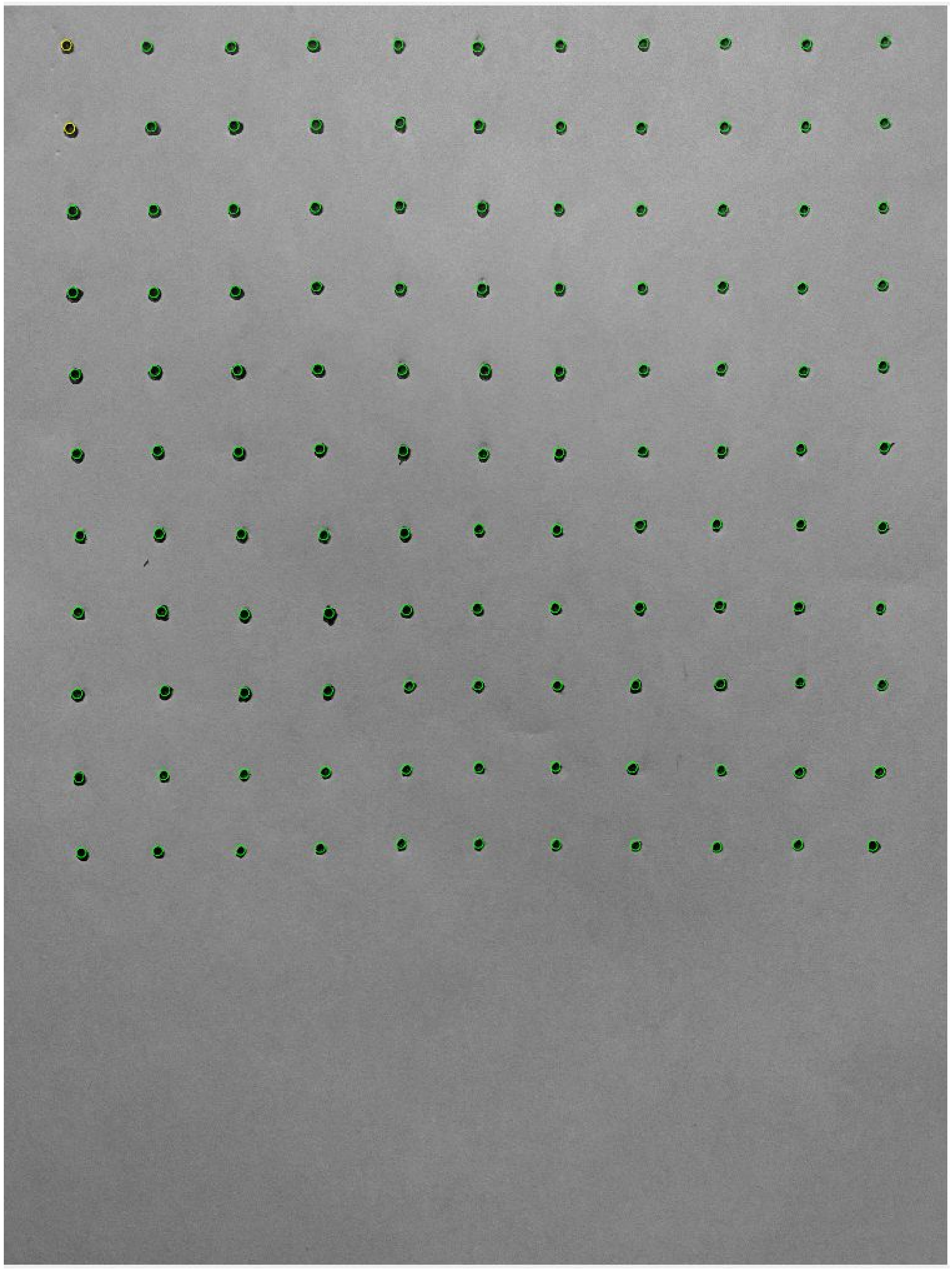


Figure 6: Successfully captured all points on the .tiff file and established origin

After running these commands, the MATLAB code was able to generate a text file which had saved all the x and y coordinates of the points that were captured in Figure 6. These points will be used in the next step when applying a different depth to the plot. The application of a different depth was done by creating another sheet of paper with same diameter circles with the same amount of spacing. This was then cut out to cover a portion of the original sheet by applying a material of depth. I decided to use a few erasers to increase the depth of the cut plot. These erasers have a flat surface and provide easy coverage for the plot to go on top of. They also had a decent height which allowed for a good depth which will help when analyzing the image in MATLAB. I measured the height of the erasers to understand the depth that it will provide using a caliper. This can be seen below.

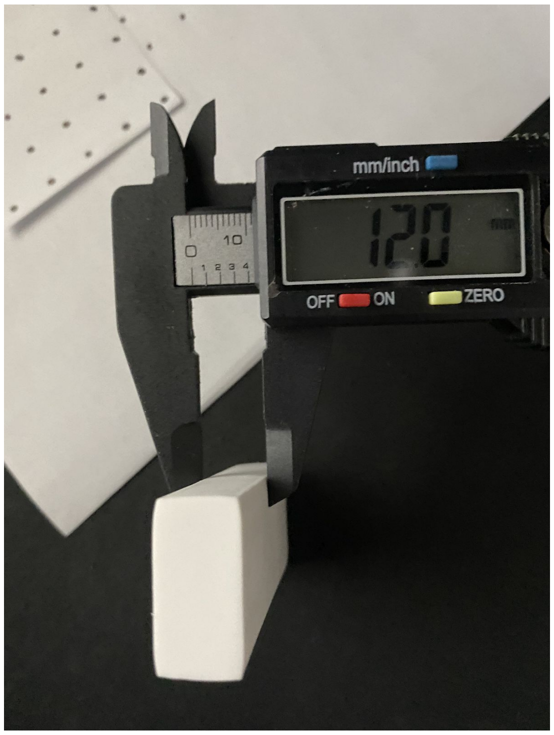


Figure 7: Measuring the depth of the object used to raise depth with a caliper

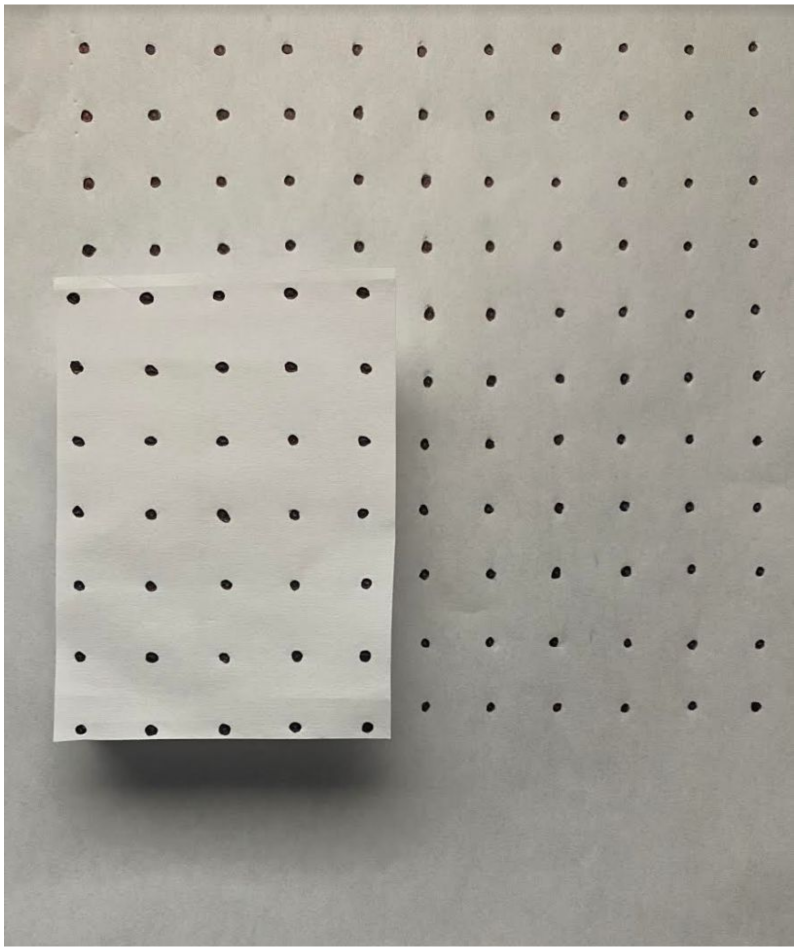


Figure 8: New 3D Calibration plate with raised depth

After this step, I had to convert the image to grayscale and convert it to a .tiff again using the same method as the 2D Calibration plate. Afterwards I was able to import it into MATLAB and modify the code again (Cal1_Left1_1_NewPlate.m). I noticed that the previous code, which was supplied with the image, had used integer values corresponding with how many "+" signs were in each column and numbered the columns. I decided to follow this method, however, my approach would have to be different because I did not have varying depths per column, rather I had different depths in the same column. This seemed like a difficult task but when I had began modifying the code I noticed that the first 5 columns of my plate were different and had to state in the code that the first 4 dots in these 5 columns were at Z = 0 and the rest were elevated at Z = 12, since this is the height of the object that I had used to provide depth to my plate. Additionally, I had to modify the code and say how many dots were in each of the columns. This was easy since I had done a simple matrix and knew that all the columns would consist of 11 dots. Now, I would have to incorporate the x and y values of the points that I have obtained in the previous script and load them into this code. I had done this by using the 'load' command and read the values into MATLAB and assigned each column to its respective variable, X and Y. Afterwards, I was able to run the script and received the following results:

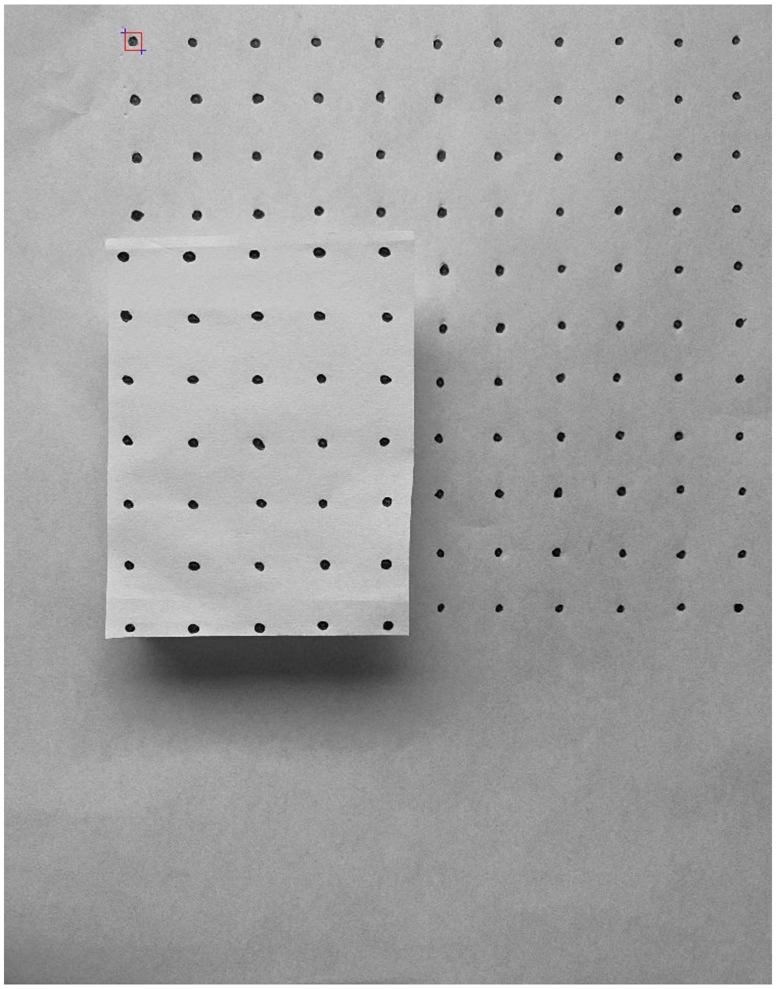


Figure 9: Selecting portion to scan for along the 3D .tiff file

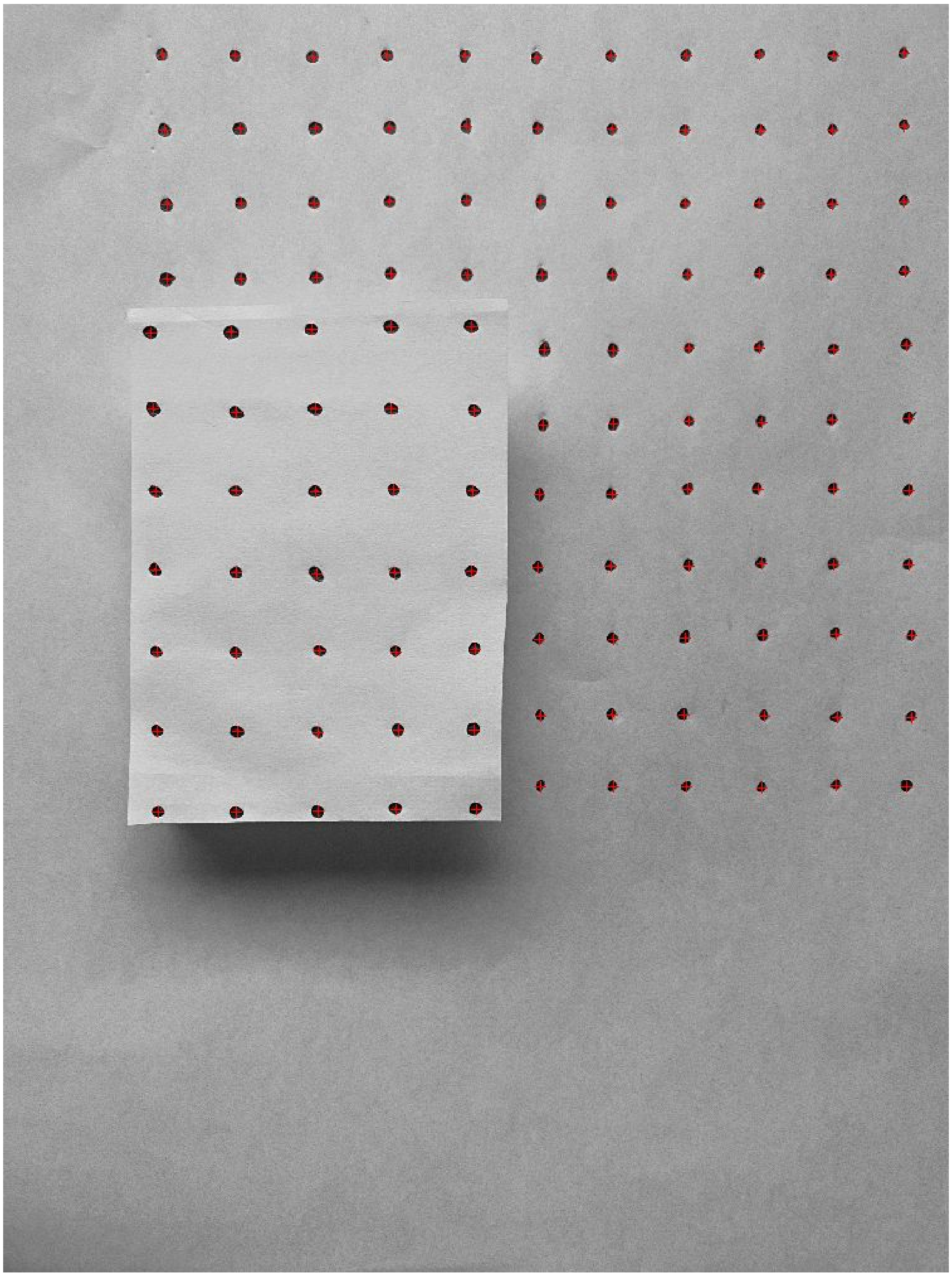


Figure 10: All points on the 3D .tiff file recognized

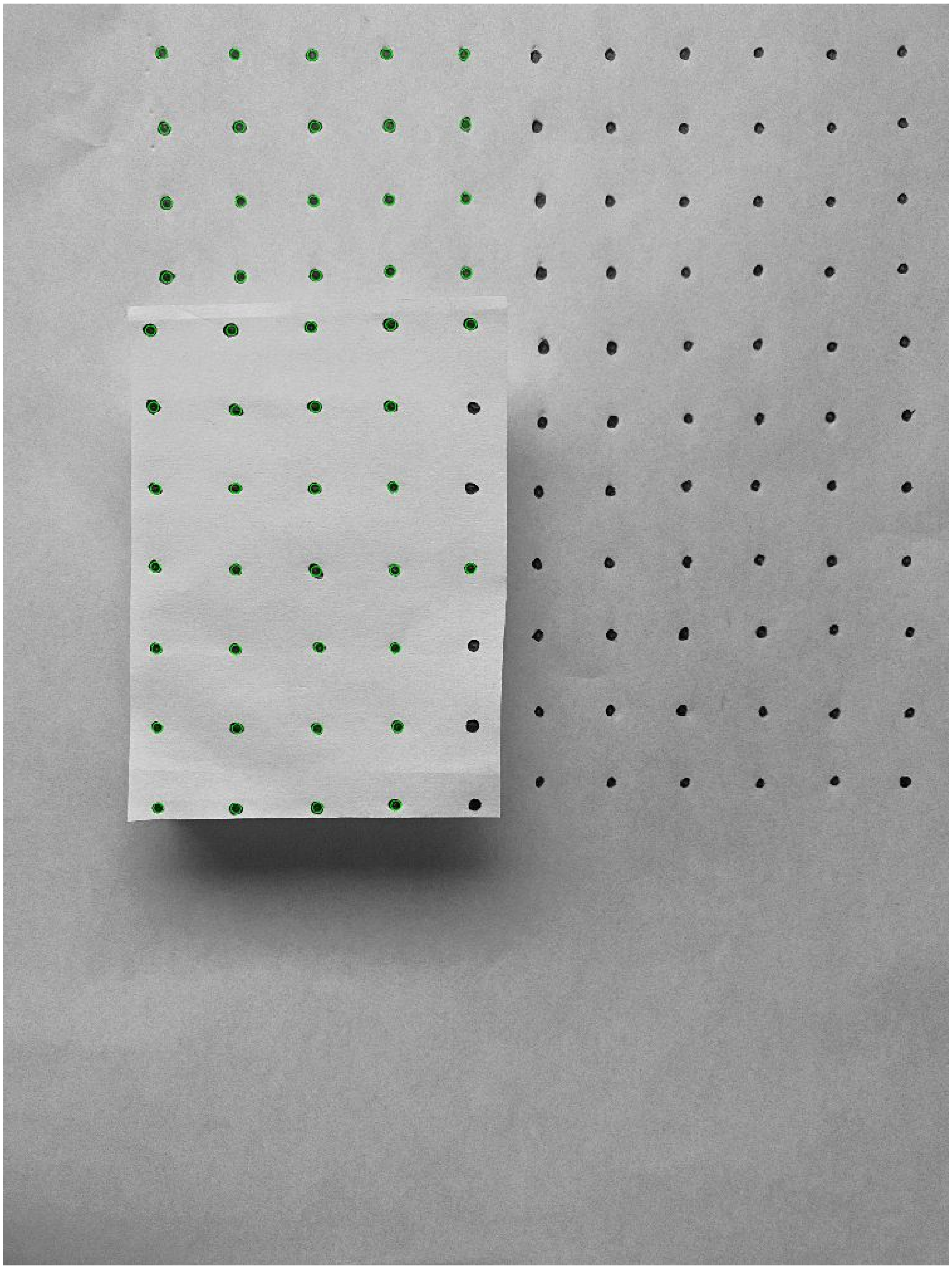


Figure 11: Successfully captured majority of points on left side on the 3D .tiff file

As seen in Figure 11 above, the purpose of the code was to detect the left hand portion of the 3D .tiff file. This was done successfully as the majority of the points were detected and the points on a different depth were also detected. Before running this script, I had uncommented a save command which exports a text file and saves different values of integers which we can now use to read into the next MATLAB script. Additionally, the script was unable to run since there was a variable which was not being used known as ‘Kdata’. I had to comment out this variable in order for the script to run. I had also created another save file which would store the Z values that were calculated in this portion of the script so that they can be used later.

The next MATLAB script to be used was to analyze the sides of the plate from taking an image from 45 degrees from the left and 45 degrees from the right. This was done by taking these images and merging them together. After merging these images together, I had done the same process to convert these images to .tiff files. I noticed that the MATLAB code which I needed to use (Cal1_Left_1_1_0.m) needed to be modified to fit the method of which I was approaching the problem. After analyzing this code and the next (ViewOnly1.m) I had noticed that the next step for this assignment required an array which consisted of xc, yc, x, y and z values. In order to do this, I would need to create a save file which would consist of these values. The results of this merged 45 degree plates can be seen below.

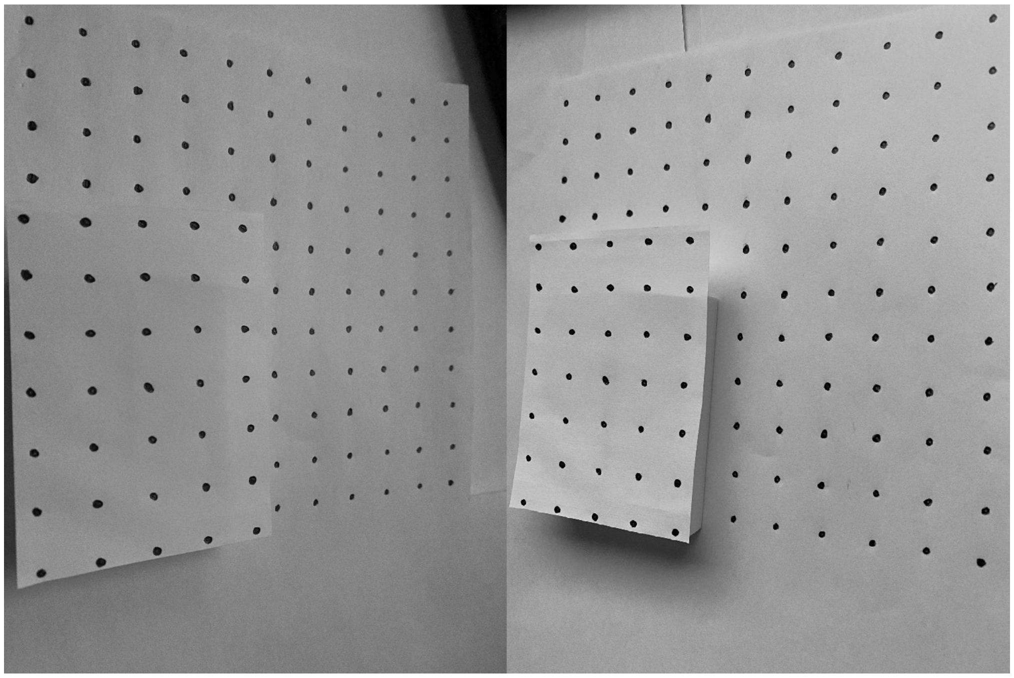


Figure 12: 45 degree plates calibration

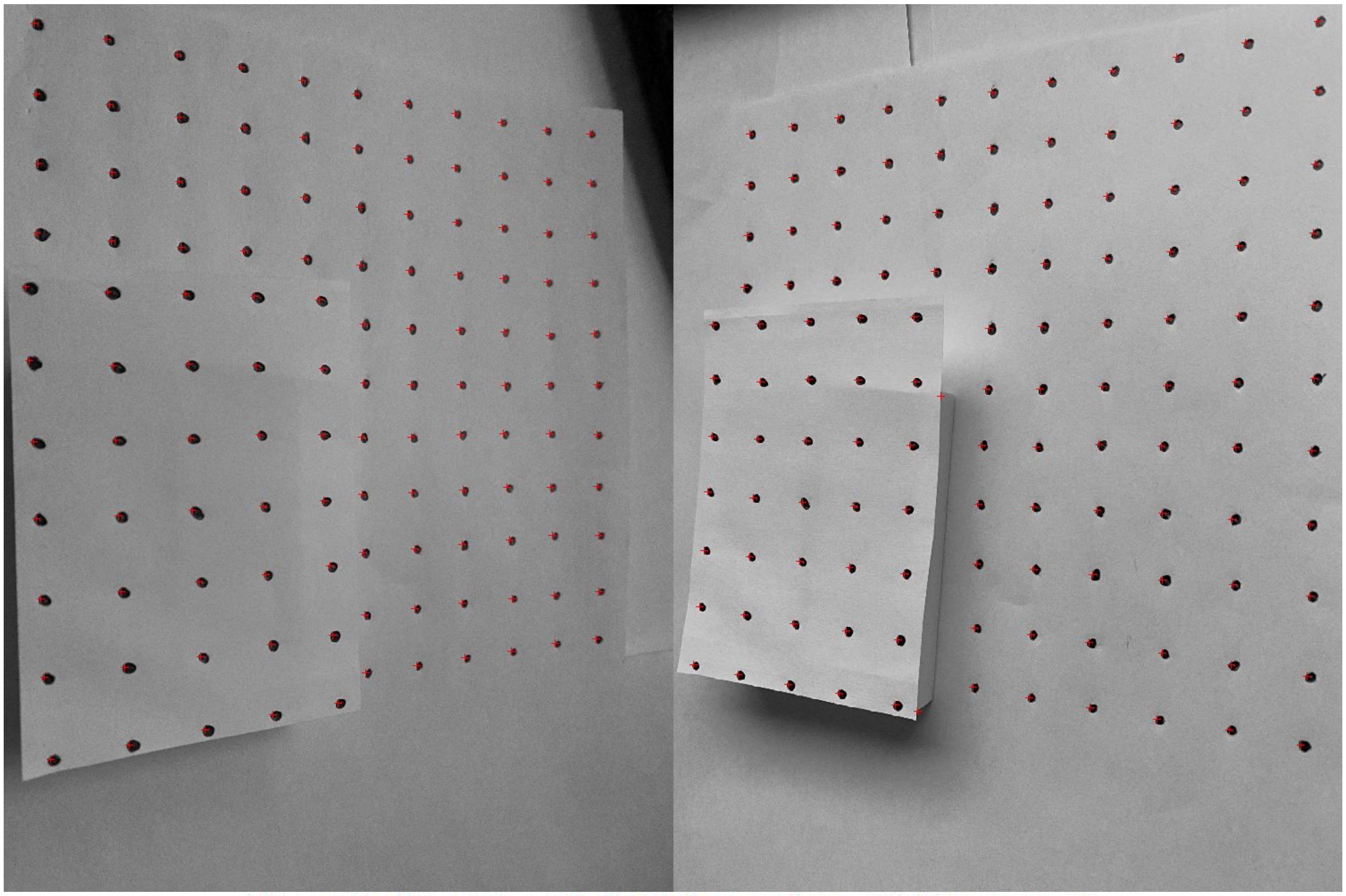


Figure 13: All points on the 3D 45 degree planes .tiff file recognized

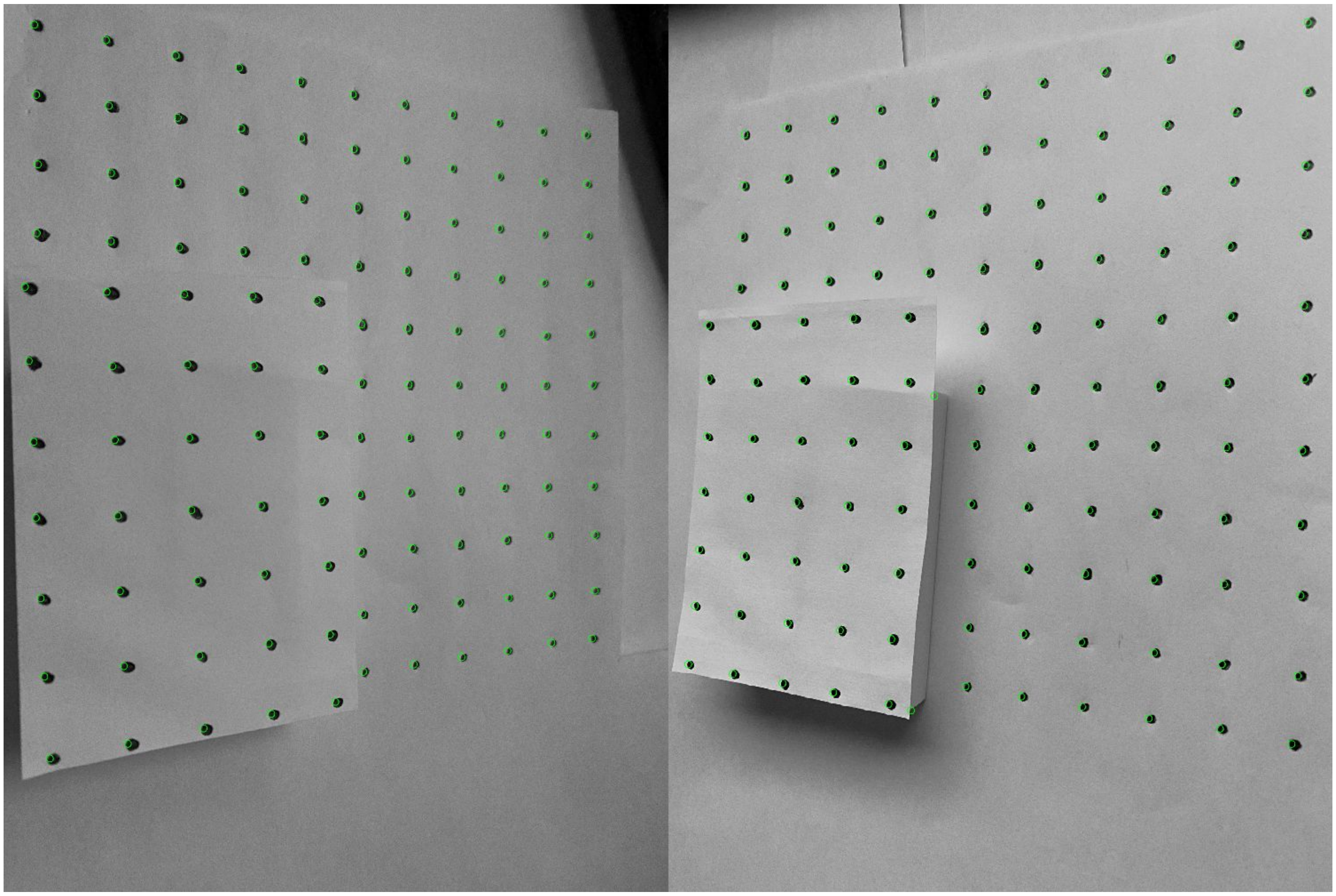


Figure 14: Successfully captured majority of points on the 3D 45 degree plates .tiff file

Using my previous script, I had already saved the Z values and was able to import and transpose the matrix so that it would be a single column. Additionally with the values obtained from the 2D calibration, I had already obtained the X and Y values and using MATLAB commands, I was able to split the matrix into different column vectors. Finally, I know that the length of the xc and yc may vary so I decided to limit the length of these vectors to the length of the original X and Y values obtained from the OUTA_Left.txt file. After finishing, I labeled a text file ‘Done.txt’ and imported all of the required values into this file. I then read this file into the next script. For this MATLAB code (viewonly.m) I did not have to modify anything. Rather, when I was running the results for the code, I had noticed that some of the commands were not working and I did not understand why. I had looked online for more information but noticed that these commands don't exist and they were functions. After more research I was able to locate the decomposecamera.m and rq3.m function which made my script run. After running the script, I received the following results:

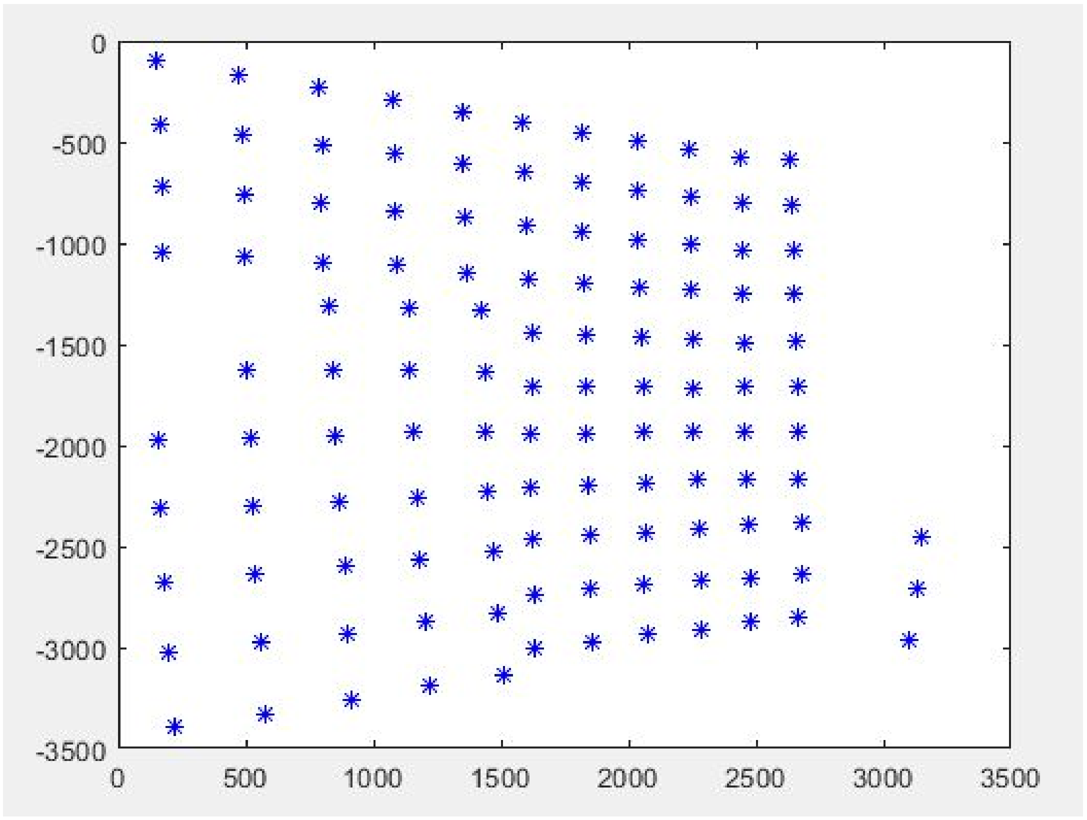


Figure 15: ysA vs xsA plot

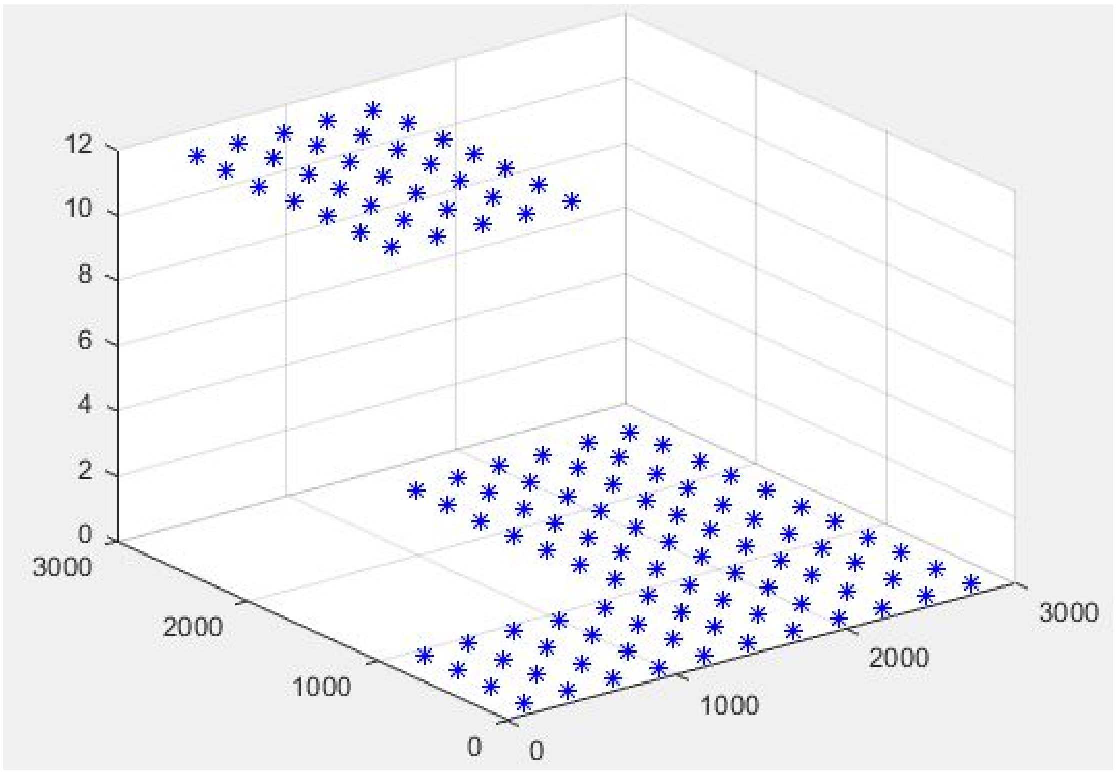


Figure 16: World Coordinates on Specimen from Loaded Data

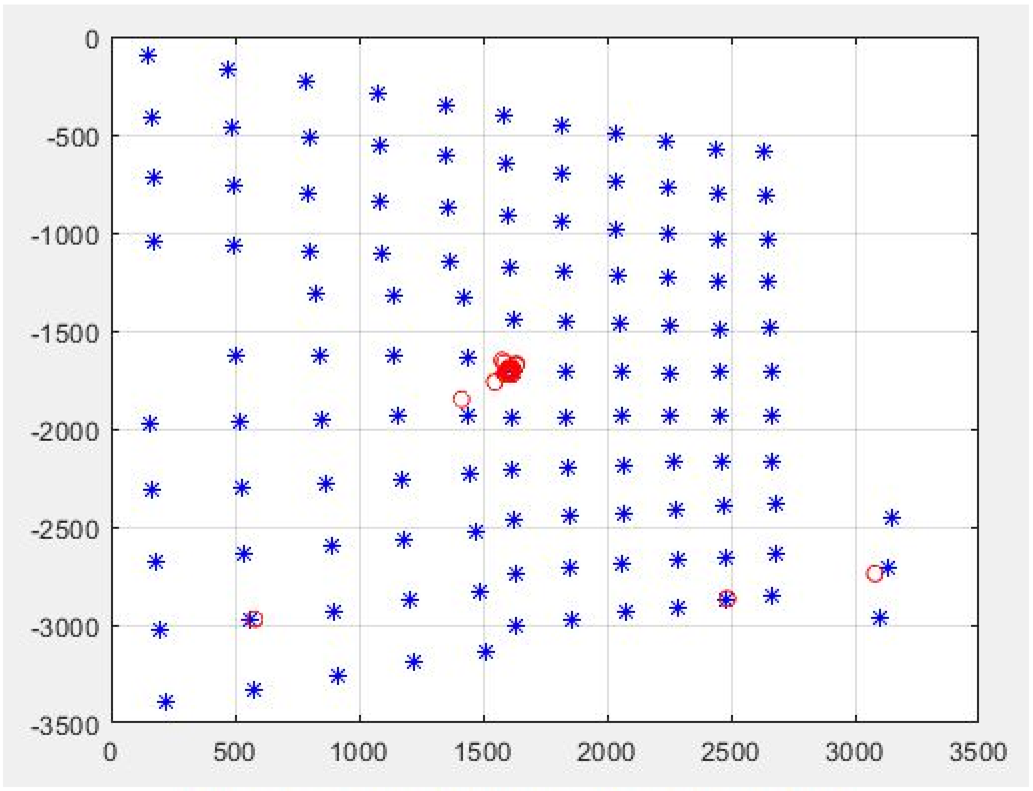


Figure 17: ysA vs xsA plot with xsa3D and ysa3D

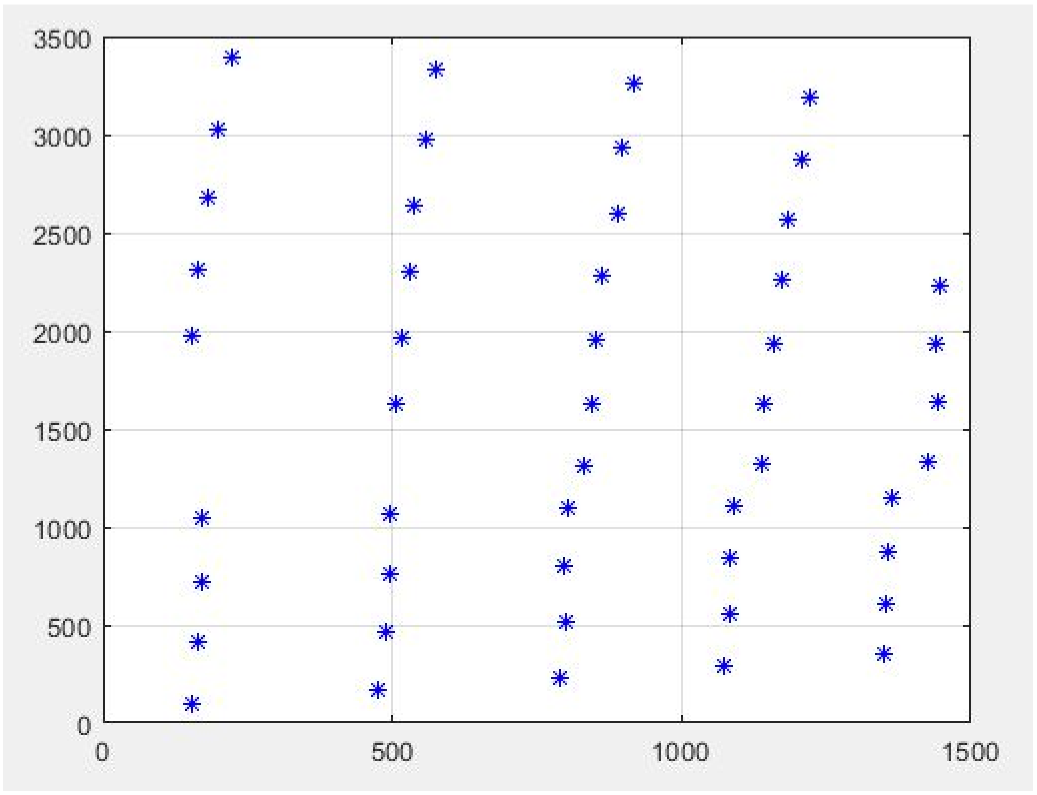


Figure 18: ysRA vs xsRA plot

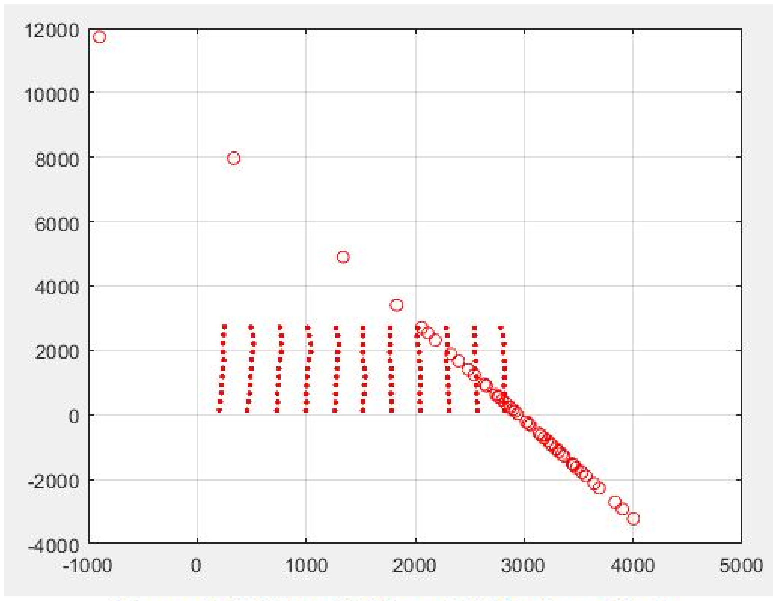


Figure 19: Plot of XA, with YwA vs XwA

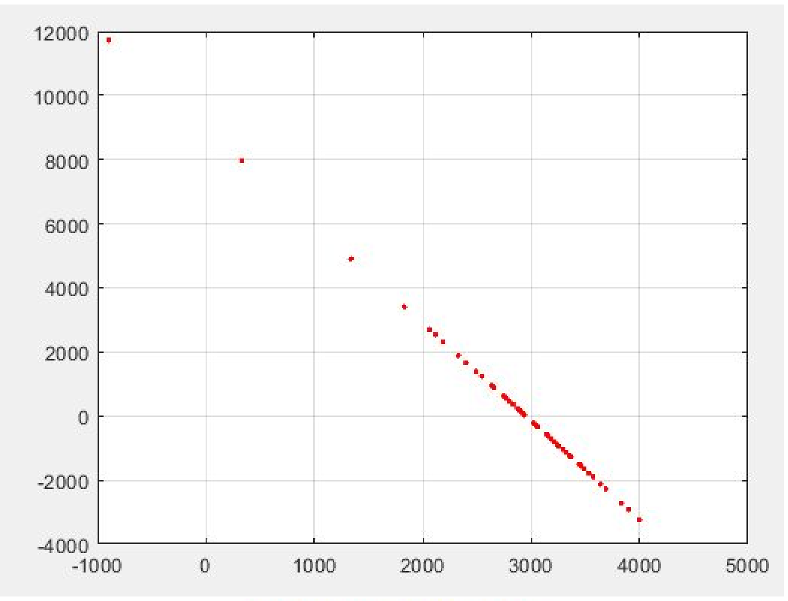


Figure 20: Plot of XA

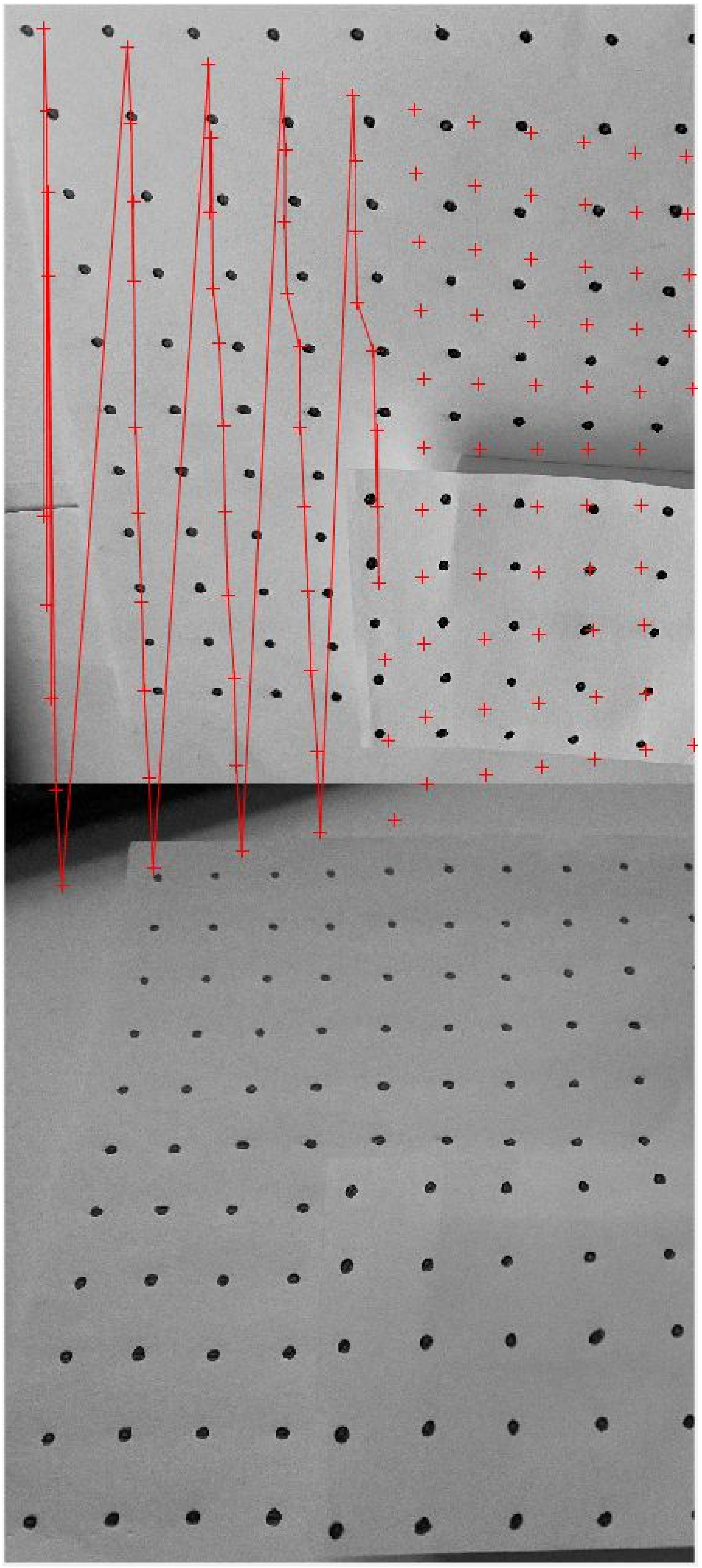


Figure 21: Calibrated plate using imshow and xsRA, ysRA, xsA and ysA

We can now describe the projective matrix using the workspace variables calculated through MATLAB. Before doing so, when using MATLAB to run my script, I receive errors due to the detection of the dots on the plot. Sometimes the dots would all be detected and the correction solution would be displayed above, however, the fx value would be very low which would then turn my f value, or the distance from the picture, very low. But when I receive the right answer, my plate above is not correct. I will be using the situation where my fx value is possible and happens to show in the Workspace after multiple attempts. This might be due to the fact that when I made my second sheet to add depth to my plate, the dots towards the end of the sheet have no blank space and this may cause some errors in the MATLAB code. I have attempted to fix this several times but the issue persists and my fx remains very low.

Cx	Cy	fx	fy	scale	Tx	Ty	Tz
1.5328e+03	1.7055e+03	1.13e+02	1.05e+02	2.6e-03	1.46e+03	1.11e+03	1

Using fx, we are able to find f by the following formula:

$$f = fx/20$$

$$f = 113/20 = 5.65$$

After analyzing the MATLAB code further, we can see that all the calculations provided in the Matrix Transformation notes have been incorporated and solved. This includes the super matrix which was determined to be LA from the workspace. This is clearly shown in Figure 19 below.

$$[M] = \begin{bmatrix} 1 & 0 & X_{w1} & Y_{w1} & Z_{w1} & 0 & 0 & 0 & -x_{s1}X_{w1} & -x_{s1}Y_{w1} & -x_{s1}Z_{w1} \\ 0 & 1 & 0 & 0 & 0 & X_{w1} & Y_{w1} & Z_{w1} & -y_{s1}X_{w1} & -y_{s1}Y_{w1} & -y_{s1}Z_{w1} \\ 1 & 0 & X_{w2} & Y_{w2} & Z_{w2} & 0 & 0 & 0 & -x_{s2}X_{w2} & -x_{s2}Y_{w2} & -x_{s2}Z_{w2} \\ 0 & 1 & 0 & 0 & 0 & X_{w2} & Y_{w2} & Z_{w2} & -y_{s2}X_{w2} & -y_{s2}Y_{w2} & -y_{s2}Z_{w2} \\ & & & & & & & & & & \\ & & & & & & & & & & \\ 1 & 0 & X_{wN} & Y_{wN} & Z_{wN} & 0 & 0 & 0 & -x_{sN}X_{wN} & -x_{sN}X_{wN} & -x_{sN}Z_{wN} \\ 0 & 1 & 0 & 0 & 0 & X_{wN} & Y_{wN} & Z_{wN} & -y_{sN}X_{wN} & -y_{sN}Y_{wN} & -y_{s1}Z_{wN} \end{bmatrix} \begin{Bmatrix} L_{A14} \\ L_{A24} \\ L_{A11} \\ L_{A12} \\ L_{A13} \\ L_{A21} \\ L_{A22} \\ L_{A23} \\ L_{A31} \\ L_{A32} \\ L_{A33} \end{Bmatrix} = \begin{Bmatrix} x_{s1} \\ y_{s1} \\ \\ \\ \\ \\ \\ \\ \\ x_{sN} \\ y_{sN} \end{Bmatrix}$$

$$[M]L_A = b$$

$$[M^T][M]L_A = [M^T]b$$

$$[M^TM]L_A = [M^T]b$$

$$L_A = [M^TM]^{-1}\,[M^T]b$$

Figure 22: Super matrix calculation and LA verification

The save files generated by the script give us the following results:

LA Values from Workspace			
-5.4444874e-01	-1.7826364e-01	-7.0935755e+01	1.6047665e+03
-5.7790809e-01	-1.8936787e-01	-7.5385877e+01	1.7035118e+03
-3.3923193e-04	-1.1106053e-04	-4.4240948e-02	1.0000000e+00

Conclusion

Using the steps provided to me, I realized that finding the calibration matrix of a camera is a difficult task. This analysis of understanding calibration and how cameras work has shown me what goes on behind the scenes when you have a digital device, such as a cell phone, and it’s able to measure the distance from one area to the next by using only the photo as reference. I found this interesting since it uses depth as a perception and performs many calculations based on only reference from the image. Overall, while this task did prove to be difficult, I had enjoyed being able to apply my knowledge to a real world situation where people all around the world don’t understand how much goes into taking a picture.

<u>**Works Cited**</u>

- Andreopoulos, Yiannis "Matrix Transformation 4." Experimental Methods in Fluid Mechanics and Combustion, 17 Apr. 2020, CUNY: City College of New York. PDF document.
- Andreopoulos, Yiannis "January 12 2012." Experimental Methods in Fluid Mechanics and Combustion, 17 Apr. 2020, CUNY: City College of New York. PDF document.
- https://www.atiz.com/resources/DPI-PPI-Megapixels-and-Resolution.pdf
- https://www.mathworks.com/help/vision/ug/camera-calibration.html
- https://www.peterkovesi.com/matlabfns/
- https://wolfcrow.com/understanding-terminology-pixels-megapixels-and-resolution/