TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Data-Based Real-Time 3D Calibration

From Chapter **??** we know one dimension calibration method is suitable for calibrating multiple cameras together; three dimension object calibration has the highest accuracy but also cost more on system setup; two dimension plane calibration owns both of good accuracy and simple setup. However, those traditional calibration methods are not ideal enough while considering that researchers are chasing after accuracy. Either that, the static calibration pattern does not have enough points to offer distortions' information. Concretely only few limited points could be extracted in the three dimension calibration system as shown in Fig. **??**. Or, it is hard to control the extracted calibration points to cover enough area of the image space in the famous two dimension calibration methods. Fig. 1.1 shows the simulated multiple planes of checkerboard with respect to the camera space, with data from the two dimension calibration method that is employed in both of Matlab and OpenCV applications, to intuitively inspect the extrinsic parameters. Using this method, researchers need to manually keep changing their poses of holding the checkerboard, in order to get enough calibration points to cover all of the image space area. Besides, all of the traditional calibration methods are assuming that the depth sensor offers perfect accurate Z^C values for all of its pixels, *i.e.*, $Z^C_{[row,col]} - Depth_{[row,col]} = E_{\text{Constant}}$ such that for all image space range of $[row, col]$ pixels share one same error E_{Constant}. But in practical, depth sensors always have some defects in getting same depth accuracy for all of their pixels, which we will call as "Depth Distortion". As shown in Fig. **??**, even observing a flat wall there are still many bumps and hollows in the reconstructed 3D image.

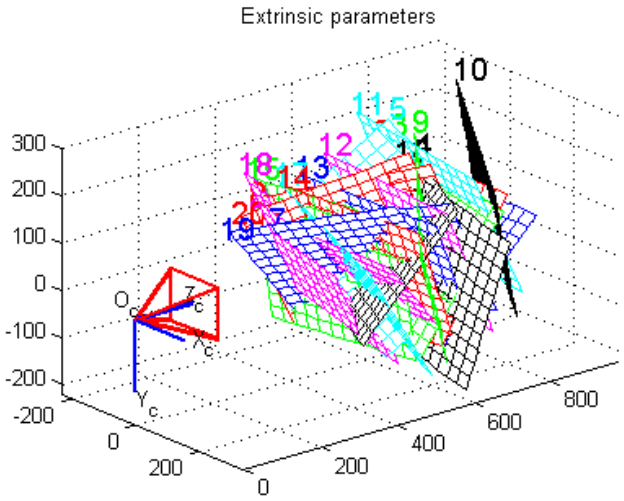


Figure 1.1: Simulated Planes of Checkerboards showing Extrinsic Parameters

Most of those defects in the methods discussed above are based on the losing control of the calibration points. Even though a flexible two dimension calibration method got its camera observing orientations easily changed by hand, it is almost impossible to numerically locate all desired poses that can make calibration points fill up all image space area. However, Tsai's old calibration system, which requires a known motion of the plane, can make them up. It has been obsoleted nowadays due to the fact that knowing the motion is not necessary for determining the intrinsic and extrinsic parameters. But as the resolution of camera sensor gets higher and higher, like a high definition sensor, researchers need a better calibration system. What's more, with the motion of the calibration plane controllable, it will not be a problem to determine the mapping from $Depth(D)$ to Z^W, thus equation **??** could be applied as one important step to simplify 3D reconstruction using a look-up table (lut) after calibration. Note that, both of the mapping from D to Z^W and the coefficients $c/d/e/f$ help map from Z^W to X^W/Y^W are with respect to per-pixel, such that even the "Depth Distortion" could be calibrated.

1.1 Novel Per-Pixel Calibration Method

In this thesis, we build a moving plane calibration system collecting tree dimensional data, with a rail that gets the RGB-D camera mounted on its slider observing a planar pattern. The 3D camera we used is KinectV2, but the calibration method could be applied on any RGB-D cameras. As shown in Fig. **??**, a uniform grid dots pattern is hung on the wall, and the rail is perpendicular to the wall. We will assign the wall, on which the canvas is hung and printed with uniform grid dots pattern, as the $X^W Y^W$ plane in world space, and Z^W-axis would be along the rail. The RGB-D camera waiting to calibrate is mounted on the slider. Note that, in this calibration system, the only unit that needs to be perpendicular to the wall is the rail, whereas the RGB-D camera has no need to require its observation orientation. Because all of the mappings we are going to determine are with respect to per-pixel, *i.e.*, the calculated parameters group for every single pixel, which will determine the pixel's view, are independent with that of the other pixels. As the slider moves along the rail, the planar dots pattern hung on the wall is moving further with respect to the RGB-D camera. This rail offers the possibility of taking infinite various frames of various camera working distances (or Z^C). Although the dots pattern hung on the wall for camera calibration is static itself, however, the dots distributions would be dynamic (covering every single pixel) in the image space when counted together in all of those various frames of various Z^C.

In this per-pixel calibration method, we directly focus on the view of every single pixel, the beam. Got inspired by Kai (eqn. **??**), our camera calibration method consists of two

big steps: frames ($X^{\mathrm{W}}Y^{\mathrm{W}}Z^{\mathrm{W}}+D$) data collection, plus per-pixel mapping parameters determination after frames collection; *i.e.*, to collect frames data of Z^{W} from external and (X^{W}, Y^{W}) by a transformation from (R,C), plus to calculate per-pixel parameters that help map from D to Z^{W} and $c/d/e/f$ in eqn. **??** that map from Z^{W} to $X^{\mathrm{W}}/Y^{\mathrm{W}}$. Note that, neither have we decided the mapping model from D to Z^{W}, nor from (R,C) to (X^{W}, Y^{W}). We save the flexibilities between accuracy and complexity for now, and will decide those two models based on data.

Assuming the total size of the depth sensor is as small as a point, and the Z^Ws for all of the pixels in one frame would share the same value, which could be measured by a laser distance measurer nearby the camera. To simplify the digital image processing (DIP) when extracting a desired point (R, C), which will soon be discussed in section 1.2, we assign the "2D origin" ($X^{\mathrm{W}}/Y^{\mathrm{W}}=0$) of world space coordinates at the center of the dot which is closest to the center of the camera's field of view. And the laser measurer spot nearby the camera to at Z^{W}=0, such that Z^W values are always negative. Note that, the final assigned world space origin ("3D origin") is not necessarily be where the camera sensor is. It depends on the camera's observation orientation, whose changing leads to the change of the "2D origin" (and finally change the 3D origin).

In our calibration system, Z^W values for all pixels of every frame will be supported in real-time by a calibrated BLE Optical-Flow tracking module, as will be discussed later in section **??**. The "Unit One" of Z^W value is assigned to be same with the side of unit-square of the uniform grid pattern, which can simplify the DIP processing of (C, R) extraction. Concretely, the distance between every two adjacent dots' centers in real-world is 228mm. Therefore, Z^W = -Z(mm) / 228(mm), where Z is the distance in reality from the camera to dots-pattern plane along the rail. Fig. 1.2 shows one sample frame of 3D reconstruction in the assigned world space, where both of the origin and Z-axis are high-lighted in blue.

As for (X^{W}, Y^{W}) values' collection, a transformation from (R,C) is needed, during which the lens distortions correction must be considered. In the traditional calibration method, world space $X^{\mathrm{W}}/Y^{\mathrm{W}}/Z^{\mathrm{W}}$ are mapped to undistorted (R',C') by linear pinhole camera matrix M. And then the undistortion step from undistorted (R',C') to distorted (R,C) is done by eqn. **??**, which uses a high order (higher than 2nd order) polynomial equation. Assuming that there is a high order mapping relationship directly from the distorted image space (R,C) to world space (X^{W}, Y^{W}), we will do different orders of two-dimensional polynomial prototypes in Matlab using its application of "Curve Fitting Toolbox", and then decide a best-fit mapping model with a high accuracy and a relative small number of parameters.

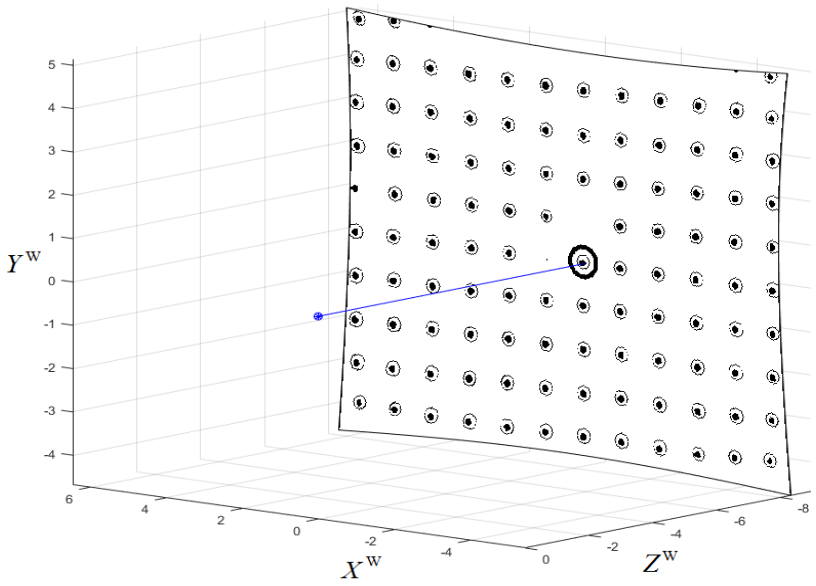


Figure 1.2: NearIR $X^W Y^W Z^W$ 3D Reconstruction

A 3x3 linear transformation matrix A (as 1^{st} order polynomial) is usually used for perspective distortion correction, give by eqn. 1.1. We will try the perspective distortion correction to check if it will also work on the lens distortion correction.

$$\begin{bmatrix} zX^W \\ zY^W \\ z \end{bmatrix} = A \cdot \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} \tag{1.1}$$

A two-dimensional polynomial model means surface mapping between two different spaces. The second order polynomial mapping has 2x6=12 parameters, written as

$$\begin{aligned} X^W &= a_{11}C^2 + a_{12}CR + a_{13}R^2 + a_{14}C + a_{15}R + a_{16} \\ Y^W &= a_{21}C^2 + a_{22}CR + a_{23}R^2 + a_{24}C + a_{25}R + a_{26}\,, \end{aligned} \tag{1.2}$$

and similarly, the fourth order polynomial mapping has 2x15=30 parameters, given by eqn. 1.3.

$$\begin{aligned} X^W &= a_{11}C^4 + a_{12}C^3R + a_{13}C^2R^2 + a_{14}CR^3 + a_{15}R^4 + a_{16}C^3 + a_{17}C^2R \\ &\quad + a_{18}CR^2 + a_{19}R^3 + a_{110}C^2 + a_{111}CR + a_{112}R^2 + a_{113}C + a_{114}R + a_{115} \\ Y^W &= a_{21}C^4 + a_{22}C^3R + a_{23}C^2R^2 + a_{24}CR^3 + a_{25}R^4 + a_{26}C^3 + a_{27}C^2R \\ &\quad + a_{28}CR^2 + a_{29}R^3 + a_{210}C^2 + a_{211}CR + a_{212}R^2 + a_{213}C + a_{214}R + a_{215} \end{aligned} \tag{1.3}$$

To prototype equation 1.2 and 1.3 in Matlab, "Curve Fitting Toolbox" is used to obtain the 2x6 and 2x15 parameters, using 107 points' coordinates pairs of image space R/C and world coordinates X^W/Y^W. Based on the "Goodness of fit" of transformation parameters from Matlab, the Root-Mean-Square Error (RMSE) of (X^W, Y^W) is (0.06796, 0.05638) for the 2^{nd} order polynomial, and (0.02854, 0.02343) for the 4^{th} order polynomial. As the order of polynomial goes higher, the RMSE gets smaller, and the cost of parameters' calculation gets much higher as well. After the prototyping in Matlab, the deferent orders of polynomial models will be applied into real-time streams transformation to get live transformed world space reconstruction. Eventually, the 4^{th} order polynomial transformation model is selected as the distortion removal mapping model, which can give an intuitive undistorted world space image while costing relative less calculations. Detailed results and comparisons will be shown soon in section 2.1.

Till now, the mapping model from (R,C) to $(X^{\mathrm{W}}, Y^{\mathrm{W}})$ has been decided, with which the data collection of $X^{\mathrm{W}}Y^{\mathrm{W}}Z^{\mathrm{W}} + D$ is ready to be completed. With enough data, the four per-pixel parameters in the beam equation. **??** are now able to be calculated. In the meanwhile, a best-fit mapping model from D to Z^{W} could also be determined. Both of D and Z^{W} are continuous data, so that their function could written as a polynomial expression, based on Taylor series. Fig. 1.3 shows the polynomial fitting result in Matlab "Curve Fitting Tool" toolbox, with 32 points of DZ^{W} values (at pixel $column$=256 and row=212) from 32 frames. It is apparent that Z^{W} is linear with D, which is also reasonable. Therefore, for

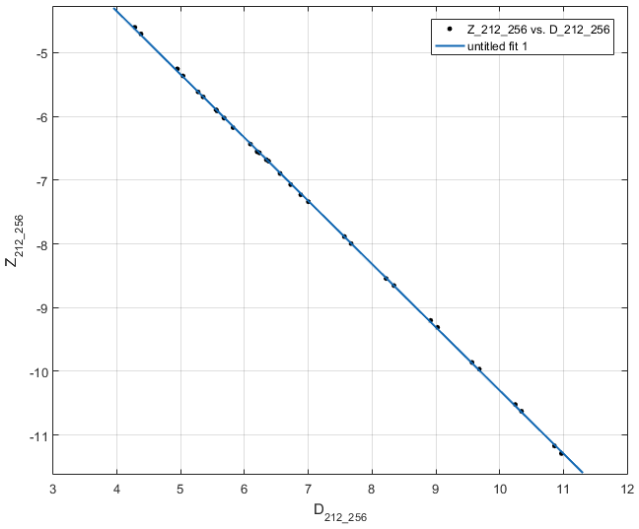


Figure 1.3: Polynomial Fitting between D and Z

every single pixel, Z^{W} could be mapped from D through

$$Z^{W}_{[row,col]} = a_{[row,col]} D_{[row,col]} + b_{[row,col]}, \tag{1.4}$$

where $[row, col]$ denotes the address of a pixel, a/b are the corresponding linear coefficients that help map from D to Z^{W}. With eqn. 1.4 supporting the per-pixel Z^{W} values, eqn. **??** help generating the per-pixel $X^{W}Y^{W}$ values, all of the 3D camera's per-pixel calibration is now done, and we are all ready to show an undistorted 3D $X^{W}Y^{W}Z^{W}$ reconstruction image.

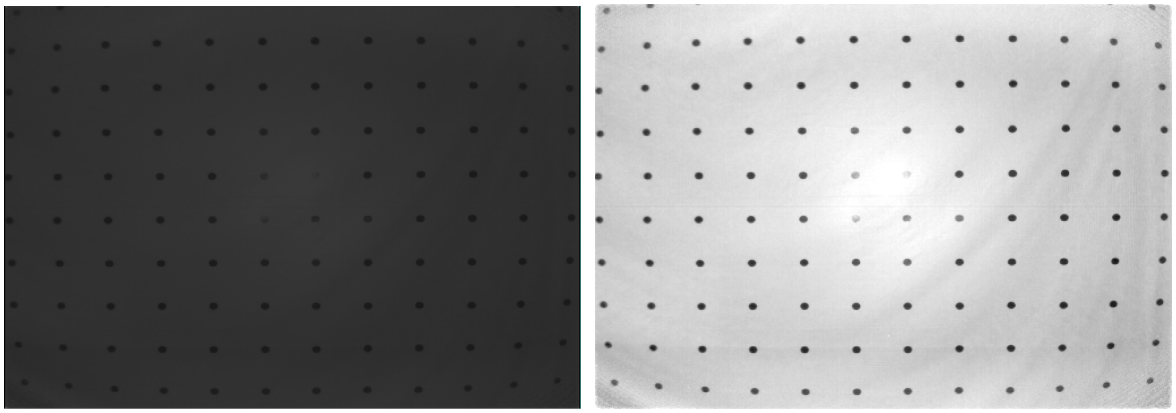
1.2 DIP Techniques on (R, C) Extraction

In order to train the 4^{th} order mapping model that can map from (R, C) to (X^{W}, Y^{W}), we need to obtain at least 15 points' coordinate-pairs of both image space coordinates and world space coordinates. Therefore, a robust DIP process to extract the points' addresses (R, C)s is of the top priority. Concretely in this project, we are going to extract the dot-clusters' centers from KinectV2 NearIR steams using DIP techniques. To guarantee a robust processing, the extraction steps consist of gray-scaling, histogram equalization, adaptive thresholding and a little trick on black pixels counting. OpenGL is selected as the CPU image processing language. The default data type of steams saved on GPU during processing is single-floating, with a range from 0 (balck) to 1 (white).

Gray-scaling is done in order to suit for both of the RGB and the NearIR steams. For NearIR steam, its data contains only color gray. There is no need to consider gray-scale problem, and data will be saved on GPU as single-floating automatically. Whereas for RGB steam, a conversion from RGB to gray value is needed. Typically, there are three converting methods: lightness, average, and luminosity. The luminosity method is finally chosen as a human-friendly way for gray-scaling, because it uses a weighted average value to account for human perception, which is give by eqn. 1.5.

$$Intensity_{\_\mathrm{gray}} = 0.21Red + 0.72Green + 0.07Blue \tag{1.5}$$

As values saved on GPU, all of the pixel intensity values are within the range of [0, 1], where "0" means 100% black and "1" means 100% white. In practical, NearIR steam image is always very dark, as shown in Fig. 1.4a (with their intensity values every close to zero). In order to enhance the contrast of NearIR image for a better binarizing, rescaling is necessary. In this section, histogram equalization technique is used maximize the range of valid pixel intensity distributions. Same process is also compatible on the RGB steam. Commonly, Probability Mass Function (PMF) and Cumulative Distributive Func-



(a) Raw (b) Histogram Equalized

Figure 1.4: NearIR Streams before / after Histogram Equalization

tion (CDF) will be calculated to determine the minimum valid intensity value ($floor$) and maximum valid value ($ceiling$) for rescaling, whereas tricks could be used by taking advantage of the GPU drawing properties.

PMF means the frequency of every valid intensity value for all of the pixels in an image. Dividing all of the pixels in terms of their intensity values into N levels, every pixel belongs to one level of them, which is called gray level. With an proper selection of N to make sure a good accuracy, the intensity value of a pixel could be expressed based on its gray level n

$$Intensity = n/N * (1-0) + 0 = n/N, \tag{1.6}$$

where n and N are integers and $1 \leqslant n \leqslant N$.PMF calculation is very similar with the points-drawing process in terms of GPU that, both of them share the properties of pixel-by-pixel calculation. For the GPU points-drawing process onto a customer framebuffer, the single-floating "color" value could go beyond the normal range [0, 1], with a maximum value of a signed 32-bit integer (2^{31} - 1). And different "color" values will be added together to form a "summational-color" in the case that some pixels are drawn onto the same position coordinates. "Taking" the range of pixel intensity values [0, 1] "as" a segment on x-axis waiting to be drawn, the intensity frequency "as" the "summational-color" of multiple pixels with different intensity drawn at the same position, and the counting process of intensity frequency "as" a points-drawing process, PMF could be calculated by drawing all of the pixels onto the x-axis within the normal intensity range [0, 1], with every single pixel's position coordinates re-assigned as ($pixel\_intensity$, 0) and its "color" value constantly being equal to one. Given the width (range of x-axis) of customer framebuffer being [-1, 1] in OpenGL, which is twice the range of pixel intensity [0, 1], the half-width of the customer

framebuffer is same with the total number N of gray levels, which determines the precision of $floor$ / $ceiling$ intensity selection.

With PMF calculated and each intensity frequency that mapped to its corresponding gray level saved in the customer framebuffer, CDF could be easily calculated as

$$CDF(n) = \frac{sum}{N_{\text{\_Total Pixels/Image}}}, \tag{1.7}$$

where the gray level n is counted from the middle of the framebuffer's width to the end (1 ~ N). And sum is the summation of customer framebuffer's values added up consecutively from 1 till n. Then, at appropriate CDFs, e.g., $CDF(n_{\text{\_floor}}) = 0.01$ and $CDF(n_{\text{\_ceiling}}) = 0.99$, the intensities $floor$ and $ceiling$ could be written as

$$\begin{aligned} floor &= n_{\text{\_floor}}/N \\ ceiling &= n_{\text{\_ceiling}}/N \end{aligned} \tag{1.8}$$

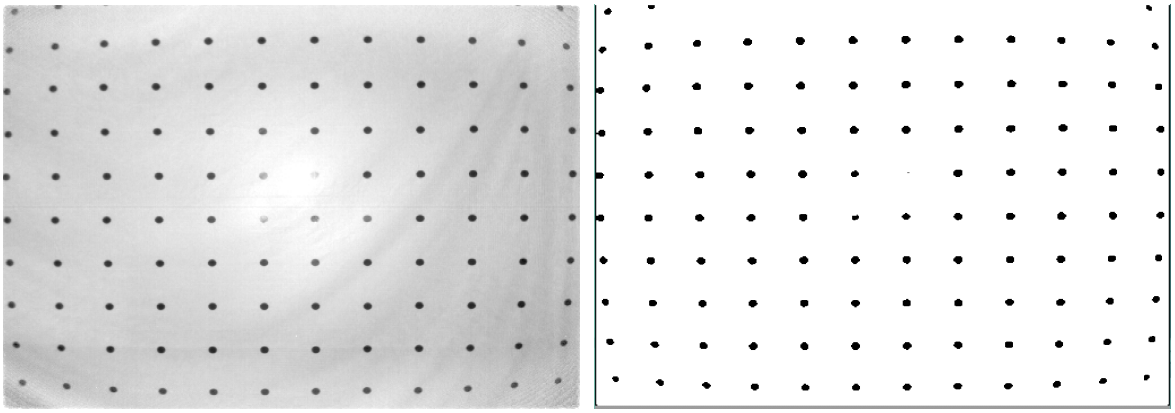
Finally, a new intensity value of every single pixel in an image could be rescaled as

$$Intensity_{\text{\_new}} = \frac{Intensity_{\text{\_original}} - floor}{ceiling - floor} \tag{1.9}$$

After this final rescaling step of Histogram Equalization, the new image gets better contrast effect, as shown in Fig. 1.5a

Affected by radial dominated lens distortions, the intensity value tend to decrease as the position of a pixel moves from the center of an image to the borders, in the case of observing a singular color view. Therefore, an adaptive thresholding process is needed, whereas using fixed thresholding will generate too much noise around borders. To segment the black dots from white background, we could simply subtract an image's background from an textured image, where the background comes from a blurring process of that image.There are three common types of blurring filters: mean filter, weighted average filter, and gaussian filter. Mean filter is selected for this background-aimed blurring process, because it has the smallest calculation and also a better effect of averaging than the others. After the blurred image containing background information is obtained, the binarizing (subtraction) process for every single pixel could be written as

$$Intensity_{\text{\_binarized}} = \begin{cases} 1, & I_{\text{\_textured}} - I_{\text{\_background}} - C_{\text{\_offset}} > 0 \\ 0, & else \end{cases}, \tag{1.10}$$



(a) Histogram Equalized (b) Binarized

Figure 1.5: NearIR Streams before / after Adaptive Thresholding

where I is short for $Intensity$ of every single pixel, and $C_{\text{-offset}}$ is a small constant that could be adjusted depending on various thresholding situations. In this project, $C_{\text{-offset}}$ is around 0.1.To sharpen the edge of the binarized image for a better "circle" shape detection, a median filter could be added as the last step of adaptive thresholding. As shown in Fig. 1.7, background is removed in the binarized image after adaptive thresholding.

After the adaptive thresholding, image data saved on GPU is now composed of circle-shaped "0"s within a background of "1"s. In order to locate the center of those "0"s circle, which is the center of captured round dot, it is necessary to know the edge of those circles. A trick is used to turn all of the edge data into markers that could lead a pathfinder to retrieve circle information.The idea that helps to mark edge data is to reassign pixels' values (intensity values) based on their surroundings. Using letter O to represent one single pixel in the center of a 3x3 pixels environment, and letters from A~H to represent surroundings, a mask of 9 cells for pixel value reassignment could be expressed as below.

E	A	F
B	O	C
G	D	H

To turn the surroundings A~H into marks, different weights will be assigned to them. Those markers with different weights have to be non-zero data, and should be counted as the edge-part of circles. Therefore, the first step is to inverse the binary image, generating an image that consists of circle-shaped "1"s distributed in a background of "0"s.After reversing, the next step is to assign weight to the surroundings. OpenGL offers convenient automatic data type conversion, which means the intensity values from "0" to "1" of single-floating data type save on GPU could be retrieved to CPU as unsigned-byte data type from "0" to

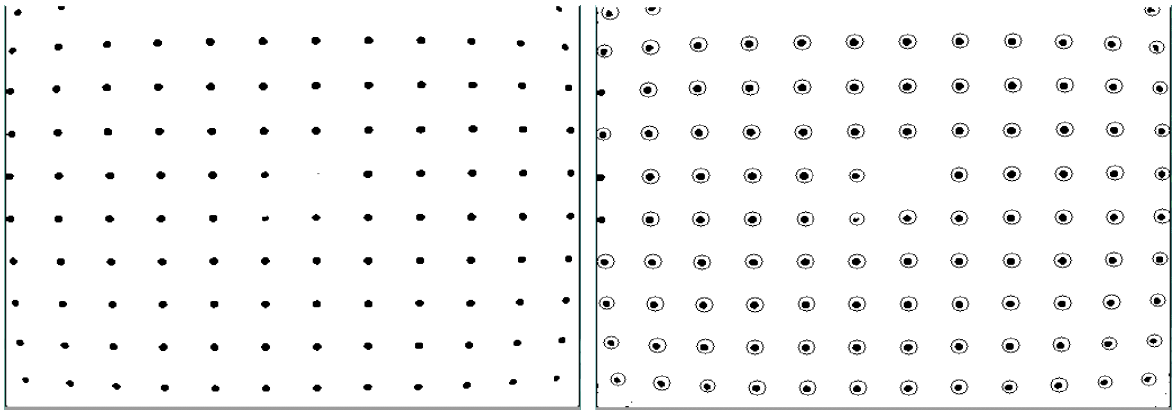
"255". Considering a bitwise employment of markers, a binary calculation related weight assignment is used in the shader process for pixels. The intensity reassignment for every single pixel is expressed as the equation below.

$$I_{\text{\_Path Marked}} = I_{\text{\_Original}} * \frac{(128I_{\text{A}} + 64I_{\text{B}} + 32I_{\text{C}} + 16I_{\text{D}} + 8I_{\text{E}} + 4I_{\text{F}} + 2I_{\text{G}} + I_{\text{H}})}{255} \tag{1.11}$$

After this reassignment, the image is not binary any more. Every non-zero intensity value contains marked information of its surroundings, data at the edge of circles are now turned into fractions. In other words, the image data saved on GPU at the moment is composed of "0"s as background and "non-zero"s circles, which contains fractions at the edge and "1"s in the center.Now, it is time to discover dots through an inspection over the whole path-marked image, row by row and pixel by pixel. Considering that, a process of one single pixel in this step may affect the processes of the other pixels (which cannot be a parallel processing), it is necessary to do it on CPU. The single-floating image data will be retrieved from GPU to a buffer on CPU as unsigned-byte data, waiting for inspection. And correspondingly the new CPU image will have its "non-zero"s circles composed of fractions at the edge and "1"s in the center. Whenever a non-zero value is traced, a dot-circle is discovered and a singular-dot analysis could start. The first non-zero pixel will be called as an anchor, which means the beginning of a singular-dot analysis. During the singular-dot analysis beginning from the anchor, very connected valid (non-zero) pixel will be a stop, and a "stops-address" queue buffer is used to save addresses of both visited pixels and the following pixels waiting to be visited. On very visit of a pixel, there is a checking procedure to find out valid (non-zero) or not. Once valid, the following two steps are waiting to go. The first step is to sniff, looking for possible non-zero pixels around as the following stops. And the second step is to colonize this pixel, concretely, changing the non-zero intensity value to zero. Every non-zero pixel might be checked 1~4 times, but will be used to sniff for only once.

As for the sniffing step, base on the distribution table of *A*~*H* that has been discussed above and their corresponding weight given by equation 1.11, the markers *A*/*B*/*C*/*D* are valid (non-zero) as long as the intensity value of pixel *O* satisfies the following conditions shown as below.

$$\begin{aligned}
&if\ (I_{\text{O}}\ \&\ 0\text{x}80 == 1), \quad then, \quad \text{marker } A \text{ is valid} \ \ (\text{ go Up }) \\
&if\ (I_{\text{O}}\ \&\ 0\text{x}40 == 1), \quad then, \quad \text{marker } B \text{ is valid} \ \ (\text{ go Left}) \\
&if\ (I_{\text{O}}\ \&\ 0\text{x}20 == 1), \quad then, \quad \text{marker } C \text{ is valid} \ \ (\text{ go Right }) \\
&if\ (I_{\text{O}}\ \&\ 0\text{x}10 == 1), \quad then, \quad \text{marker } D \text{ is valid} \ \ (\text{ go Down })
\end{aligned} \tag{1.12}$$



(a) before Extraction (b) Extracted and Marked

Figure 1.6: Valid Dot-Clusters Extracted in NearIR

Once a valid marker is found, its address $(column, row)$ will be saved into the "stops-address" queue. One pixel's address might be saved for up to 4 times, but "colonizing" procedure will only happen once at the first time, so that the sniffing will stop once all of the connected valid pixels in a singular dot-cluster are colonized as zeros. In the second step "colonizing", I_O is changed to zero, variable *area* of this dot-cluster pluses one, and bounding data $RowMax$ / $RoxMin$ / $ColumnMax$ / $ColumnMin$ are also updated.Finally, the Round Dot Centers $(column, row)$ could be determined as the center of bounding boxes with their borders $RowMax$ / $RoxMin$ / $ColumnMax$ / $ColumnMin$. After potential noises being removed based on their corresponding *area* and shape (ratio of width and height), the data left are taken as valid dot-clusters. As shown in Fig. 1.6b, the centers of valid dot-clusters are marked within their corresponding homocentric circles.

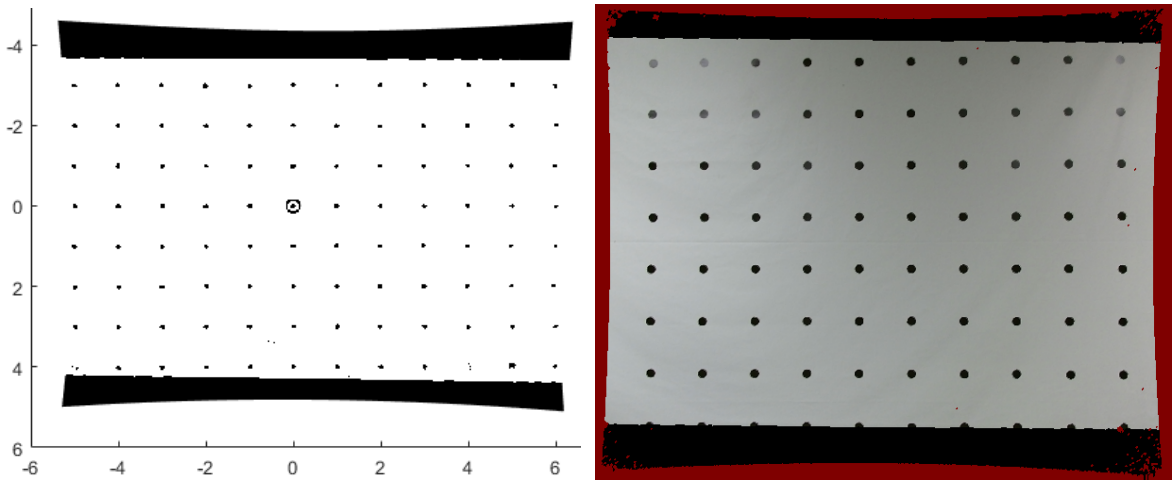
1.3 Alignment of RGB Pixel to Depth Pixel

A undistorted 3D reconstruction could be displayed with the help of a LUT generated by the per-pixel calibration method, which has been discussed in section 1.1. However, we have not figured out yet what the color of each pixel is. To generate a colored 3D reconstruction with a combination of a random depth sensor and a random RGB sensor, we need to align the RGB pixels to depth pixels. The intermediate between the depth sensor image space and RGB sensor image space is the world space. As long as we figure out the mapping from world space to RGB sensor image space, the color of pixel with known $X^W Y^W Z^W$ could be looked up from the RGB image space. The pinhole camera matrix M is used

to map from world space to RGB image space. Using the frame data from Kinect RGB steams and Kinect NearIR streams, a Matlab prototype of RGB pixels alignment is shown in fig. 1.7a, where the RGB textured is mapped onto its corresponding NearIR image, who has same pixels with depth sensor. The total black area on the top edge and bottom edge is where the depth sensor's view goes beyond the RGB sensor's view. Fig. 1.7b shows the screen-shot of live video after calibration with the RGB pixels aligned by a pinhole camera model M.

1.4 Summation

A per-pixel calibration method, using a moving plane calibration system, is proposed in this chapter. The main idea of this calibration method is to make it available to generate accurate (undistorted) per-pixel world space position (X^W, Y^W, Z^W) directly in real-time with the least calculation. The per-pixel Z^W will be mapped from per-pixel depth value D, and then per-pixel X^W/Y^W will be determined by per-pixel linear beam equation from its corresponding Z^W. Note that the per-pixel beam equation represents the lens-distortion removed field of view of a single pixel. And the "depth distortion" could be removed during the per-pixel mapping from D to Z^W. In short, this per-pixel calibration method consists of two big steps: $X^W Y^W Z^W + D$ data collection and mapping parameters determination. D is simply from depth streams. Z^W is from external based on the camera's position on the rail. And the undistorted (X^W, Y^W) are from the transformation of R/C by a 4^{th} order polynomial mapping model, during which lens-distortions could be removed. With the frames



(a) (b)

Figure 1.7: Alignment of RGB Texture onto NearIR Image

data of $X^W Y^W Z^W + D$ collected, the per-pixel mapping parameters $a/b/c/d/e/f$ could be determined by eqn. 1.4 and eqn. **??**.

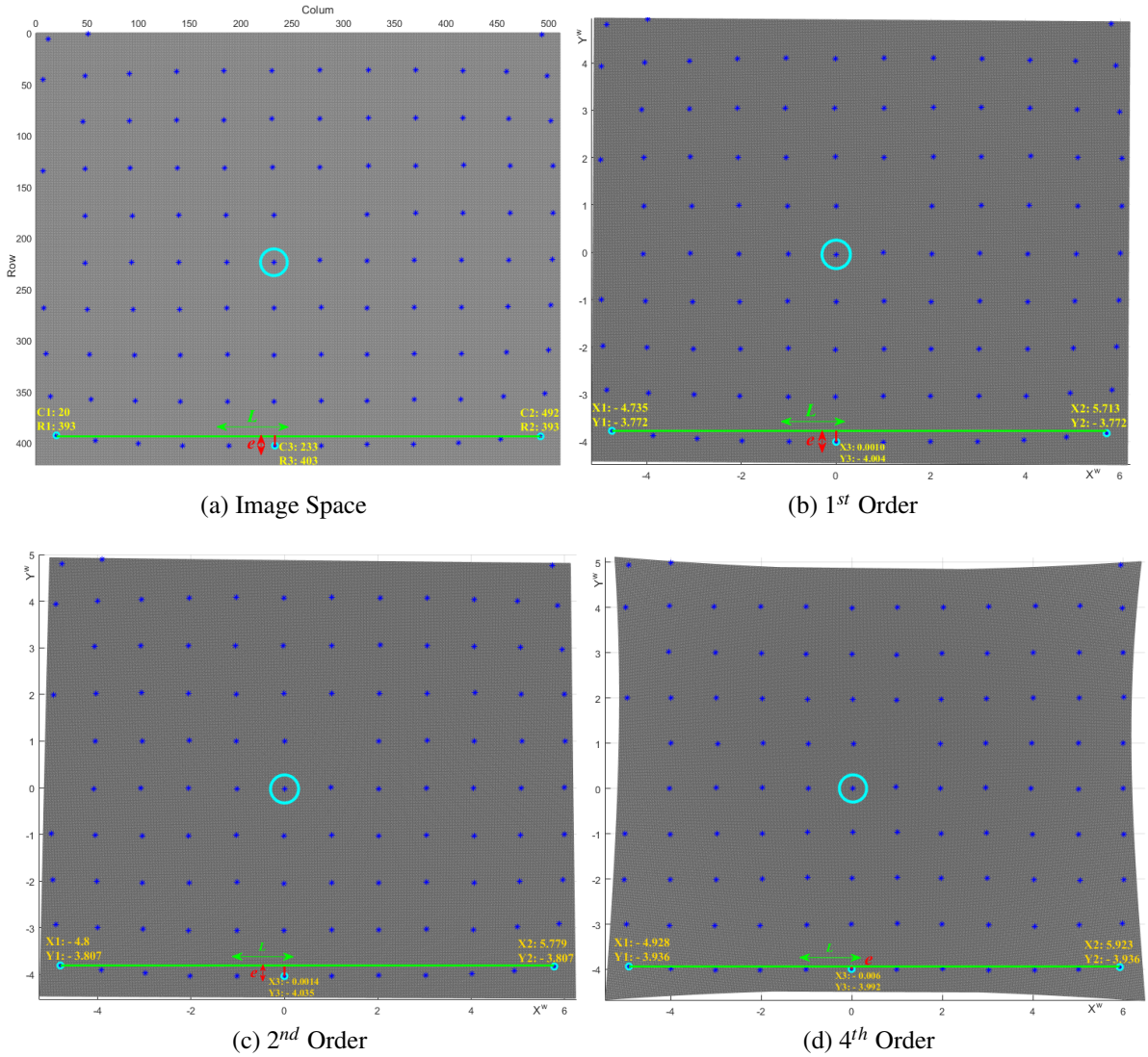
Without using the traditional pinhole camera model, two polynomial mapping models are employed in this calibration method. The first model is the two-dimensional 4^{th} order polynomial mapping from R/C to X^W/Y^W during the frames data collection, which takes care of the removal of lens distortions; and the second model is the linear mapping from D to Z^W, which can handle "depth distortion". Both of the two mapping models are determined and calculated by real streams data from the camera, so that we claim this per-pixel calibration method a "data-based" calibration method.

Besides the data-based calibration method, a robust DIP process during calibration is also discussed, which guarantees that the live video during undistorted frames collection could be in real-time. Note that "real-time" here means being able to show an undistorted frame before the start of the second frame processing. After the undistorted 3D reconstruction, the alignment of the RGB pixel to Depth pixel is also discussed. The data-based calibration method could be applied universally on any RGB-D cameras. With the alignment of RGB pixels, it could even work on the combined 3D camera of a random Depth sensor and a random RGB sensor.

Chapter 2 Results of Calibration and 3D Reconstruction

2.1 Calibration Results

As discussed in section 1.1, the per-pixel calibration method requires a two-dimensional high-order polynomial model to remove lens distortions and generate estimated (X^W, Y^W)s from image space (R, C)s. In this section, we will show detailed comparisons among different order polynomial results of both Matlab prototypes and real-time live applications. Fig. 2.1 shows the Matlab prototypes of the simulated original image, world space $X^W Y^W$



(a) Image Space

(b) 1^{st} Order

(c) 2^{nd} Order

(d) 4^{th} Order

Figure 2.1: $X^W Y^W$ Matlab Polynomial Prototype

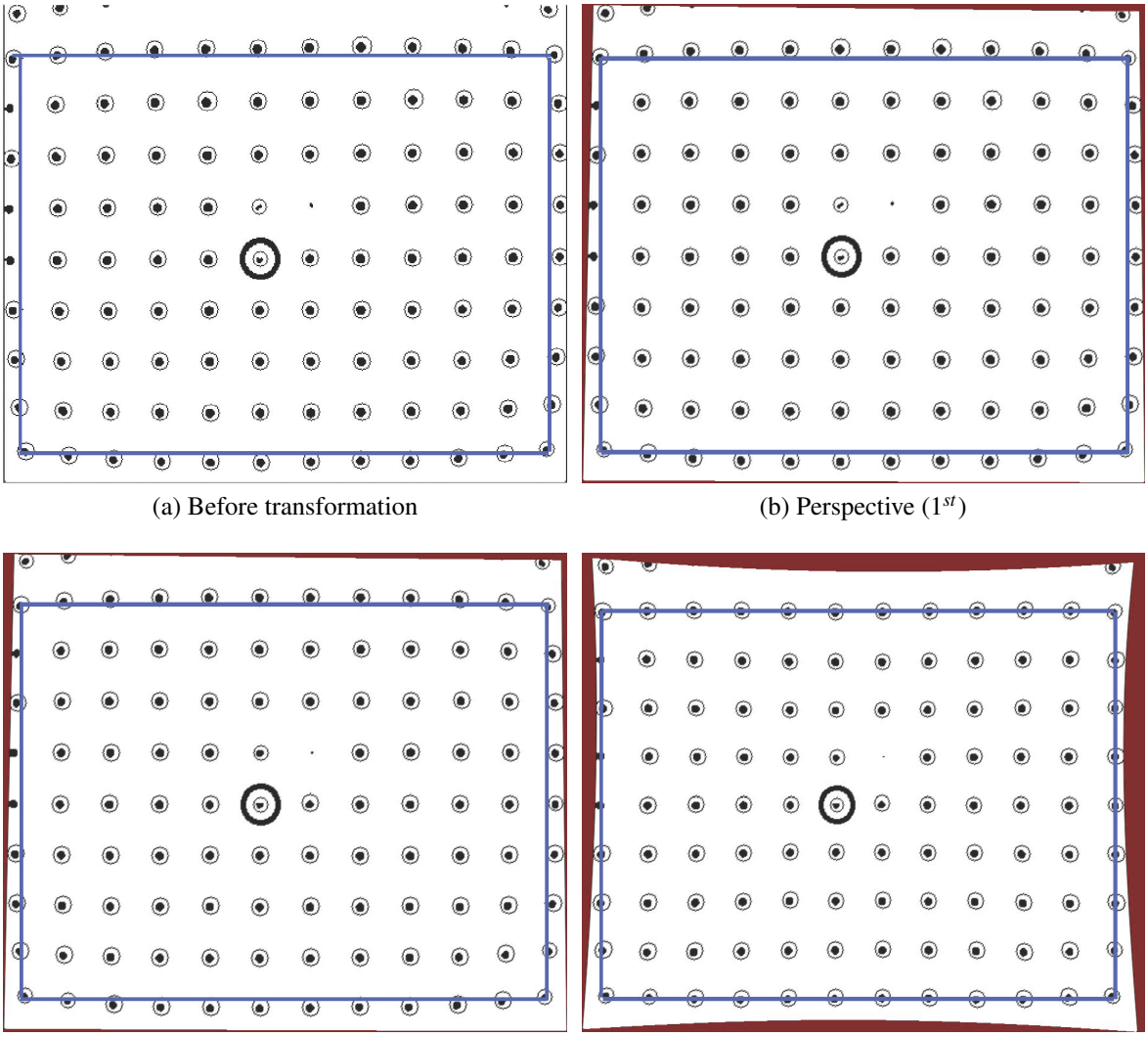
plane result after a two-dimensional 1^{st} order polynomial transformation, 2^{nd} order polynomial and 4^{th} order polynomial transformation.

With squared-shaped distributed points (C, R)s extracted from image streams, we can recover the original distorted image in Matlab, as shown in fig.2.1a. Using a mathematical distortion (d) measurement [1]

$$d(\%) = e * 100/L, \tag{2.1}$$

we can get the original distortion $d_0 = (R3 - R1)/(C2 - C1) = (403 - 393)/(492 - 20) = 2.1\%$. Fig. 2.1b shows estimated world space $X^W Y^W$ plane after a two-dimensional 1^st order polynomial transformation, whose distortion $d_1 = (Y1 - Y3)/(X2 - X1) = [-3.772 - (-4.004)]/[5.713 - (-4.735)] = 2.2\%$. As we may have expected, the distortion d_1 is not getting smaller at all. Fig. 2.1c and fig. 2.1d show the transformed world space $X^W Y^W$ plane images after the 2^{nd} order and 4^{th} order polynomial transformation respectively, from which we can get $d_2 = [-3.807 - (-4.035)]/[5.779 - (-4.8)] = 2.1\%$ and $d_4 = [-3.936 - (-3.992)]/[5.923 - (-4.928)] = 0.516\%$. It is straightforward to tell that, d_4 is much smaller than d_0 and fig. 2.1d intuitively shows a satisfying undistorted image. From eqn. 1.2 and eqn. 1.3, we know that the second order polynomial mapping has 2x6=12 parameters, and the fourth order polynomial mapping has 2x15=30 parameters. The higher order polynomial we use, the better radial distortion we are able to correct. In the meantime, the distortion removal model will have more parameters to calculate, and need more calibrating points to train the model.

By applying those two-dimensional polynomial models into real-time streams transformation, we can get the transformed stream images. As shown in fig. 2.2, the outlines of the transformed steam images are same with Matlab prototypes in fig. 2.1. It is easy to tell that the 4^{th} order polynomial surface mapping is much better than the second order, and a higher order than 4^{th} should be more accurate. However, as the order of the polynomial mapping goes higher, the number of parameters also get larger and larger, which costs more calculations and requires more data (coordinate-pairs) for training the transformation model. Considering that a 5^{th} order polynomial mapping will have much more parameters (2x21=42) to calculate while may not enhance much accuracy, we choose the 4^{th} order polynomial as the main mapping model to get $X^W Y^W$ values from RC. Limited by the static dot pattern, fewer and fewer dot-clusters could be observed by the camera as the camera getting closer to the dot pattern. Practically, 4^{th} order calibration is replaced by 2^{nd} order to guarantee a robust software when the observed dot-clusters are too few to train the transformation model.



(a) Before transformation

(b) Perspective (1^{st})

(c) 2^{nd} Order

(d) 4^{th} Order

Figure 2.2: NearIR Stream High Order Polynomial Transformation

The two-dimensional high-order polynomial mapping model will be applied in the first calibration step of $X^{\mathrm{W}}Y^{\mathrm{W}}Z^{\mathrm{W}} + D$ frames data collection. Fig. 2.3 shows 63 frames of collected $X^{\mathrm{W}}Y^{\mathrm{W}}Z^{\mathrm{W}}$, which gives an pyramid shape of a camera sensor's undistorted world space field of view. For each single pixel, its field of view is a beam, which could be mathematically expressed as equation. **??**. Some sample beams are shown in fig. 2.4, whose beam equation parameters $c/d/e/f$ are determined as the best-fit totally by the collected undistorted data.

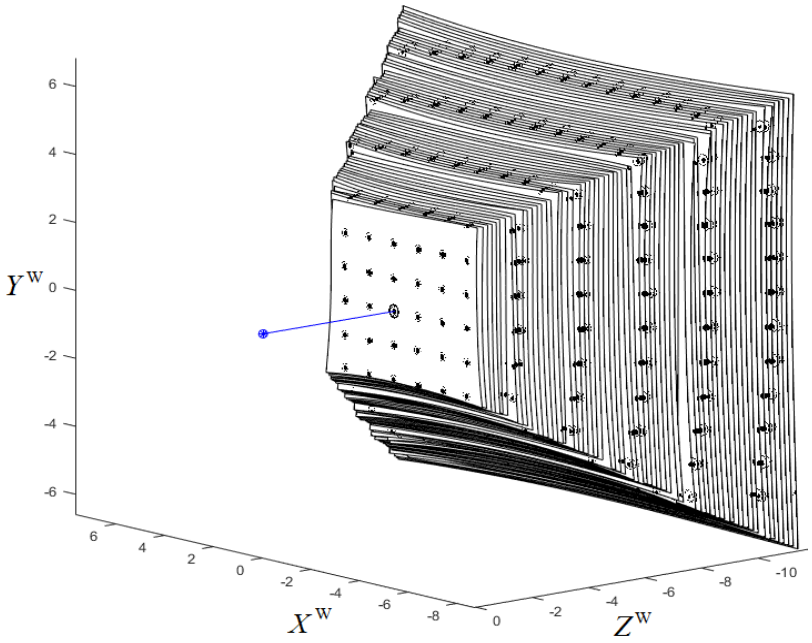


Figure 2.3: 63 Frames NearIR Calibrated 3D Reconstruction

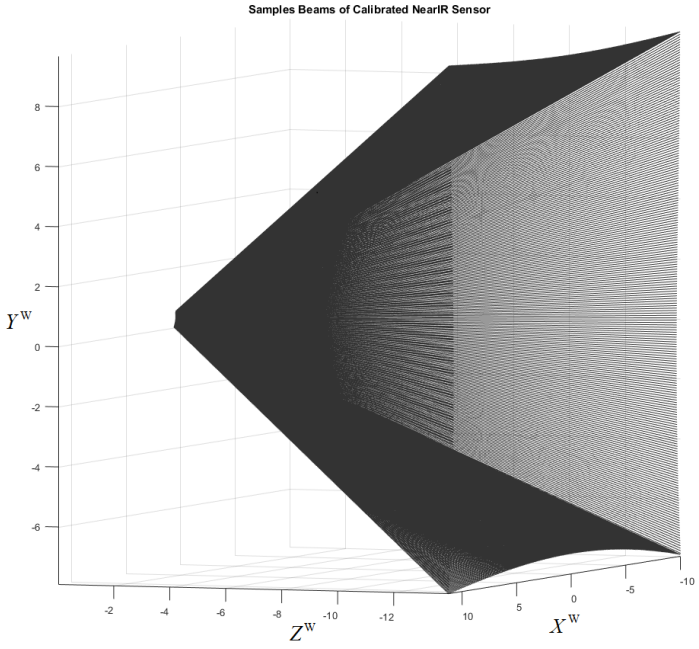
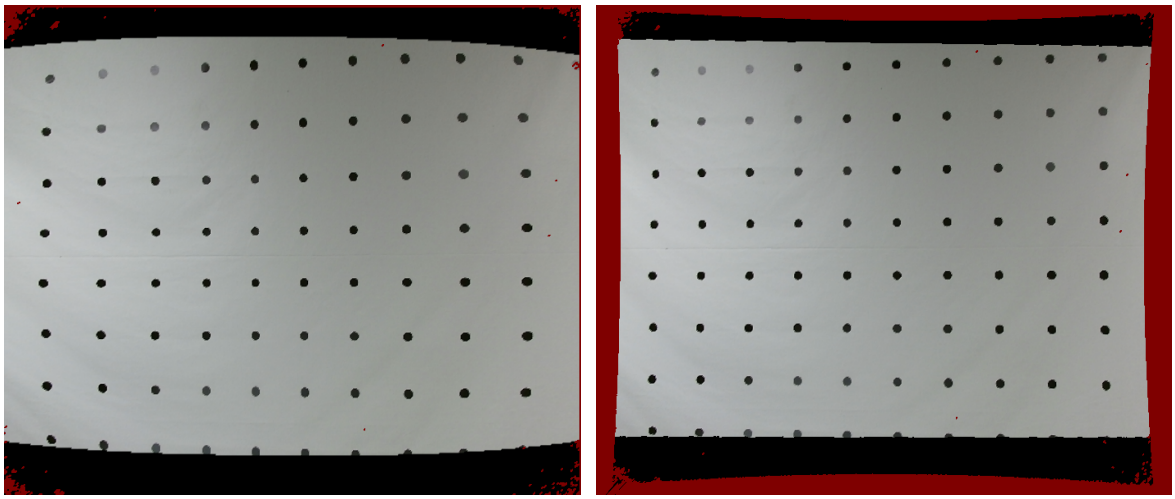


Figure 2.4: Sample Beams of Calibrated NearIR Field of View

2.2 3D Reconstruction in Real-Time

The 3D Reconstruction of undistorted $X^W/Y^W/Z^W$ in real-time is the final aim of a 3D camera's calibration. In the traditional camera calibration method, which consist of one pinhole-camera matrix to generate raw world space 3D coordinates and another model for



(a) Raw (Distorted) (b) Calibrated

Figure 2.5: Calibrated by Per-Pixel Calibration Method

lens distortion removal, three big transformations are needed to generate the world space coordinates: from 2D distorted image space to 2D undistorted image space, then to 3D camera space, and finally to 3D world space. For every single pixel's processing, it needs 5 parameters from distortion removal model for the first step non-linear calculation, and then a 3x3 intrinsic matrix to get its camera space coordinates, and a 3x4 extrinsic matrix to finally acquire the world space coordinates. The 3D reconstruction after the traditional calibration requires a lot of calculations for every single pixel.

Using the proposed data-based per-pixel calibration method, only three linear calculations with six parameters are needed to determine the world space coordinates for every single pixel. Two parameters a/b are utilized to generate world space Z^W, as expressed in eqn. 1.4. And the other four parameters $c/d/e/f$ are applied to get X^W/Y^W respectively based on eqn. **??**. In this way, there is no need to calculate any non-linear equation for distortions removal, and the camera space is totally left aside. Combining two equations together, the undistorted 3D world coordinates $(X^{\mathrm{W}}, Y^{\mathrm{W}}, Z^{\mathrm{W}})$ for every single pixel could be looked up based on D from a *column*-by-*row*-by-6 look-up table. Fig. 2.5 shows the comparison between raw (distorted) 3D reconstruction and calibrated 3D reconstruction after the per-pixel calibration.

Bibliography

[1] Z. Tang, R. Grompone von Gioi, P. Monasse, and J.M. Morel. High-precision Camera Distortion Measurements with a "Calibration Harp". *Computer Vision and Pattern Recognition*, 29(10), Dec 2012.