TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

1.1 Contributions of this thesis

For RGB-D cameras, RGB steam and Depth steam are two steams that independent but correlated with each other. With respect to every X^w/Y^w correlated single pixel-pair, Depth steam offers the additional voxel world coordinates Z^w, while RGB steam offers the additive color property. As described in section **??**, even though a pinhole camera model (3-by-4 transformation matrix) could help do 3D scanner calibration, it is only for ideal camera without lens. That is to say, in practical the lens distortions correction is separated from pinhole camera model calibration. Even though same pixel coordinate-pairs (world coordinates and image plane coordinates) could be re-utilized to solve radial dominated lens distortions, as a second step after the determination of a 3-by-4 pinhole camera model, the calculation of the separated step brings a second-time translation cost for every single pixel of every frame. This is not a good way to do real-time reconstruction.

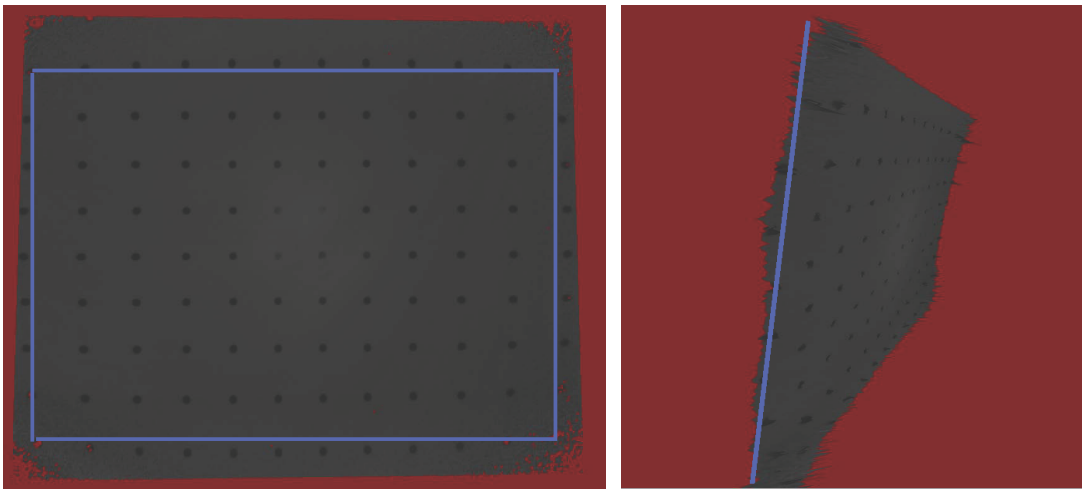
In order to remove the radial dominated lens distortions, two 3D camera calibration methods, got inspired from Kai's method, are proposed and discussed. The first method inherits the advantages given by the pinhole camera model, which can offer the relationship between Z^w and X^w/Y^w for every single pixel as long as its field of view is pre-calculated. This method is only discussed theoretically, because the second method is simpler, more accurate, and is finally applied into practical calibration.Thoroughly abandoned the pinhole camera model, the second method directly determines the beam equation for every single pixel by collecting distortions-removed X^w/Y^w and accurate Z^w values. X^w/Y^w values are calibrated by high order, concretely $4th$ order, polynomial surface mapping. And the accurate Z^w values are imported from external Z^w-aixs tracking module. In short, this is totally a data-based calibration method.

Figure 1.1: KinectV2 Calibration System

To collect enough data along Z^w-aixs, a rail is used in the calibration system. As shown in figure 1.1, the rail is perpendicular to the uniform round dot pattern as Z^w axis. A RGB-D camera KinectV2 is mounted on the top of the slider, while a BLE OF tracking module is specially designed and mounted at the bottom of the slider, observing the rail and supporting accurate Z^w values. With this calibration system that helps collect data easily, a distortion-removed XYZ-D Look-Up Table (LUT) will be generated for real-time 3D reconstruction.

1.2 Summation

RGB-D cameras' calibration cannot be easily handled by a pinhole camera model. First of all, as mentioned in section **??** and detailed in section 2.1, a pinhole camera model is not able to handle the non-linear radial dominated lens distortion. What's worse, the depth resolution deteriorates notably with depth in practical [7], so such so that the $depth$ values are not able to guarantee an accurate Z^w.



(a) Front View (b) Left View

Figure 1.2: Raw NearIR 3D Reconstruction based on Pinhole Camera Model

Noises among depth data vary randomly, camera by camera and pixel by pixel; which means a rough point-cloud plane full of bumps and hollows will be reconstructed even though the camera is observing a wall. As shown in figure 1.2b, the blue straight line should be the left side of the 3D reconstruction, whereas most pixels on the left side border are apparently not sitting on a straight line.Got inspired and extended from Kai's 3D reconstruction research, a data-based XYZ-D LUT calibration method is proposed and applied into practical application, with the help of a specially designed BLE OF tracking model supporting external accurate Z^w values. In applying this method, not only lens distortion, but also depth distortion can be removed from the 3D coordinates for reconstruction.

In Chapter 2, a pinhole camera model based calibration method is discussed in detail, from which two extended calibration methods are proposed and discussed. The second proposed method is well explained in Chapter 3. X^w and Y^w are mapped separately through a fourth order surface fitting translation from image plane row and column to directly solve the lens distortion problem. Then, Z^w values are totally supported from external BLE optical-flow sensor, which accurately tracks camera movements along Z-axis. Chapter 4 will introduce the whole calibration system, with the individually designed BLE OF tracking module. Finally, a data-based XYZWRGB-D look-up table will be generated for real-time reconstruction.

Chapter 2 Traditional RGB-D Cameras Calibration

2.1 Pinhole Camera

A pinhole camera is a simple optical imaging device in the shape of a closed box or chamber. A pinhole camera is completely dark on all the other sides of the box including the side where the pin-hole is created. Fig. 2.1 shows an inspection of a pinhole camera. In its front is a pin-hole that help create an image of the outside space on the back side of the box. When the shutter is opened, the light shines through the pin-hole and imprint an image onto a sensor (or photographic paper, or film) placed at the back side of the box. In order to analyze parameters like focal distance, field of view, etc., pinhole camera has its own three dimensional space (noted as X^c, Y^c, and Z^c). Note that, according to Cartesian Coordinates "right hand" principle, the camera is looking down the negative of Z^c-axis, given X^cY^c directions as shown in the figure. Its focal length of the pinhole camera is the distance on the Z^c-axis, between the pinhole at the front of the camera and the paper or film at the back of the camera.

Pinhole cameras are characterized by the fact that they do not have a lens. It rely on the fact that light travels in straight lines, which is a principle called the rectilinear theory of light. This makes the image appear upside down in the camera, as shown in fig. 2.2. Tracing the corners of the camera sensor through the pin hole, those dark green lines show the limits of the field of view in 3D coordinate space. The back side plane of the pinhole camera, which is behind the origin at a positive Z^c-axis and also where our sensor sits, is also called the focal plane. It is not intuitive, nor convenient for mathematical analysis that the images on the focal plane are always upside down. So a virtual focal plane is defined

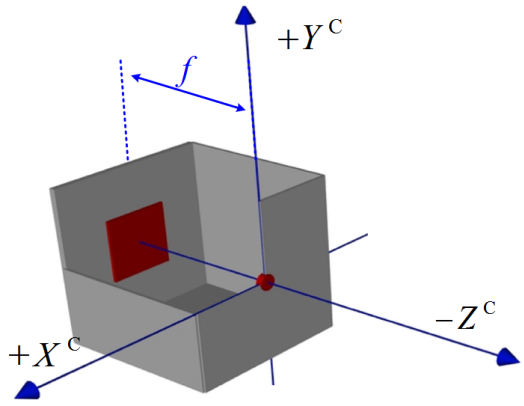


Figure 2.1: The Pinhole Camera Inspection

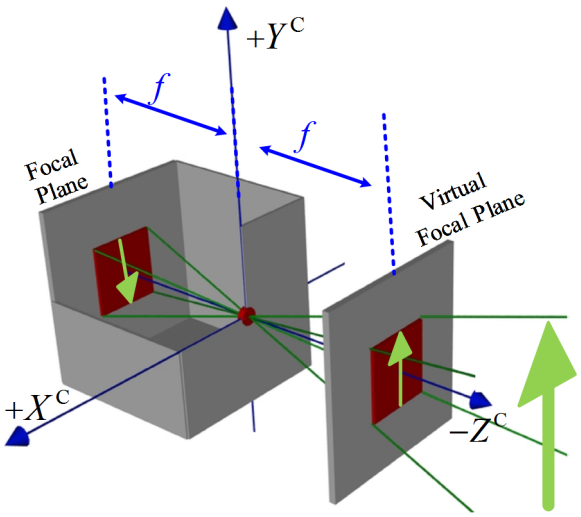


Figure 2.2: Virtual Focal Plane of a Pinhole Camera

created in front of the pinhole on the negative Z^c-axis, which is equal distant from the focal point (pin hole) as the actual focal plane is behind. Notice that the limits of the field of view intersect with the virtual focal plane at the four corners of the up-right image just as they disseminate from the four corners of the sensor at the real focal plane.

With the virtual focal plane, the camera body with the real focal plane could be removed. And the rest parts in front of the the camera body, the focal point and the virtual focal plane together, form the most common pinhole camera model. In order to employ this model for 3D analyzing points inside the camera's field of view based on 2D image, the most prior step is to define the relationship between points in 3D camera space and the 2D image coordinates row and column of the corresponding captured image. As shown in fig. 2.3, only the sensor (in color red) is visible at the virtual focal plane. The focal point is right at the origin of the camera 3D space coordinates, from where to the sensor has

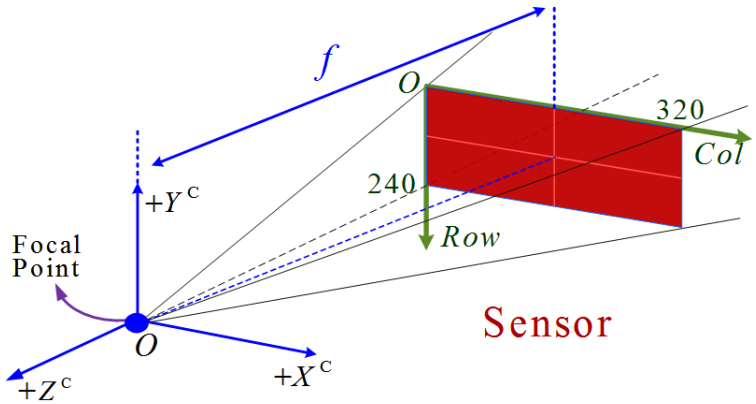


Figure 2.3: Common Pinhole Camera Model

the vertical distance of f, the focal distance. The 2D image coordinates are in dark green, where its origin is sitting at the up-left corner of the sensor. Taking the PrimeSense camera, whose RGB and Depth sensor share the pixel height and width 240*320. As long as both of the camera 3D space and image 2D space are defined, the next step is the determine the determine their relationship. Note that, the range of image should be either ([0:239], [0:319]) or ([1:240], [1:320]).

Select a random object point P^C in the camera space located at camera 3D coordinates (X_C, Y_C, Z_C). A line passing both of the point P^C and the focal point intersects with the virtual focal plane at P^I, with its image 2D coordinates (R, C). To determine the mapping function, we can start a the proportional relationship. As shown in fig. 2.4, the center point in the image coordinates, which is usually called "principle point", could be determine by column of half-width and row of half-height. Concretely, the principle point (R_h, C_h) is either (119.5, 159.5) if range is ([0:239], [0:319]), or (120, 320) if range is ([1:240], [1:320]). So, we could get the relative row and column distance of R_r and C_r) by eqn. 2.1.

$$\begin{aligned} R_r &= R - R_h \\ C_r &= C - C_h \end{aligned} \tag{2.1}$$

Based on by triangulation, it is straight forward to tell the proportional relationship between f/Z_C and C_r/X_C, R_r/Y_C. Thus we get eqn. 2.2.

$$\begin{bmatrix} C_r \\ R_r \end{bmatrix} = f \begin{bmatrix} X_C/Z_C \\ Y_C/Z_C \end{bmatrix} \tag{2.2}$$

And by changing the relative distance R_rC_r back to the 2D image coordinates (R, C), then

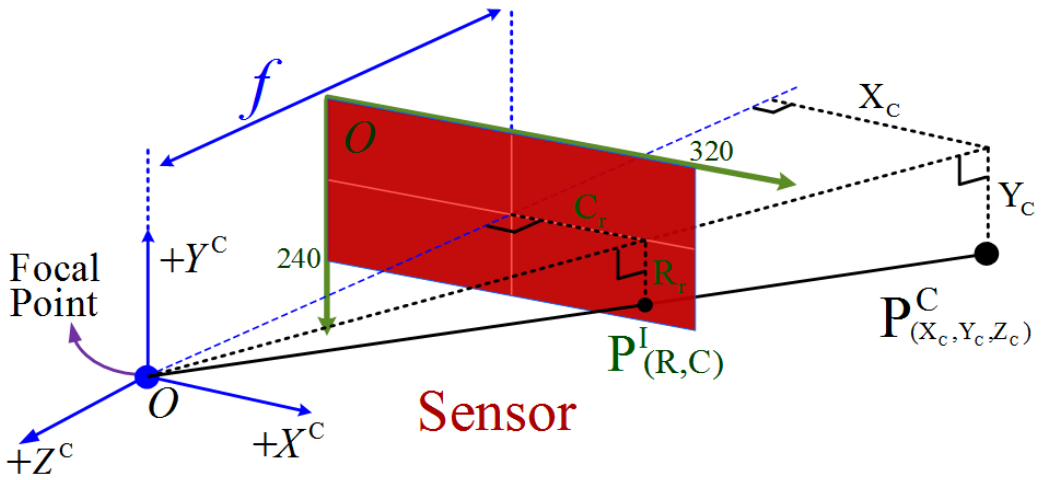


Figure 2.4: Mapping from Camera Space to Image Space

eqn. 2.2 will be written as

$$\begin{bmatrix} \mathrm{C} \\ \mathrm{R} \end{bmatrix} = f \begin{bmatrix} \mathrm{X_C}/\mathrm{Z_C} \\ \mathrm{Y_C}/\mathrm{Z_C} \end{bmatrix} + \begin{bmatrix} \mathrm{C_h} \\ \mathrm{R_h} \end{bmatrix}, \tag{2.3}$$

if written in homogeneous coordinates, we will get eqn. 2.4.

$$\mathrm{Z_C} \begin{bmatrix} \mathrm{C} \\ \mathrm{R} \\ 1 \end{bmatrix} = \begin{bmatrix} f\mathrm{X_C} \\ f\mathrm{Y_C} \\ \mathrm{Z_C} \end{bmatrix} + \begin{bmatrix} \mathrm{Z_C C_h} \\ \mathrm{Z_C R_h} \\ 0 \end{bmatrix} = \begin{bmatrix} f & 0 & \mathrm{C_h} \\ 0 & f & \mathrm{R_h} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{X_C} \\ \mathrm{Y_C} \\ \mathrm{Z_C} \end{bmatrix} \tag{2.4}$$

Till Now, we haven't consider the units translation between the camera 3D space the image 2D space. The random object point $\mathrm{P^C}$'s mapping point $\mathrm{P^I}$ (R, C) on the image space is expressed in millimeters (or inches). Since it is necessary to express the image space coordinates (R, C) in pixels, we need to find out the resolution of the sensor in pixels/millimeter. Considering that, the pixels are not necessarily be square-shaped, we assume they are rectangle-shaped with resolution α_c and α_r pixels/millimeter in the *Col* and *Row* direction respectively. Therefore, to express $\mathrm{P^I}$ in pixels, its C and R coordinates should be multiplied by α_c and α_r respectively.

$$\begin{bmatrix} \mathrm{Z_C C} \\ \mathrm{Z_C R} \\ \mathrm{Z_C} \end{bmatrix} = \begin{bmatrix} f\alpha_c\mathrm{X_C} \\ f\alpha_r\mathrm{Y_C} \\ \mathrm{Z_C} \end{bmatrix} + \begin{bmatrix} \mathrm{Z_C}\alpha_c\mathrm{C_h} \\ \mathrm{Z_C}\alpha_r\mathrm{R_h} \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha_c f & 0 & \alpha_c\mathrm{C_h} \\ 0 & \alpha_r f & \alpha_r\mathrm{R_h} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{X_C} \\ \mathrm{Y_C} \\ \mathrm{Z_C} \end{bmatrix} = K\mathrm{P^C} \tag{2.5}$$

Note that K only depends on the intrinsic camera parameters like its focal length, resolution in pixels, and sensor's width and height. Thus, the mapping matrix K is also called a camera's intrinsic matrix. Considering that the pixels might be parallelogram-shaped instead of rigid rectangle-shaped (when the image coordinate axis *Row* and *Col* are not orthogonal to each other), usually K has a skew parameter s, given by

$$K = \begin{bmatrix} f_c & s & t_c \\ 0 & f_r & t_r \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

, where $f_c = \alpha_c f$ and $f_r = \alpha_r f$ are the focal length in pixels on the *Col* and *Row* directions respectively, $t_c = \alpha_c \mathrm{R_h}$ and $t_r = \alpha_r \mathrm{R_h}$ are the translation parameters that help move the origin of image coordinate to the principle point.

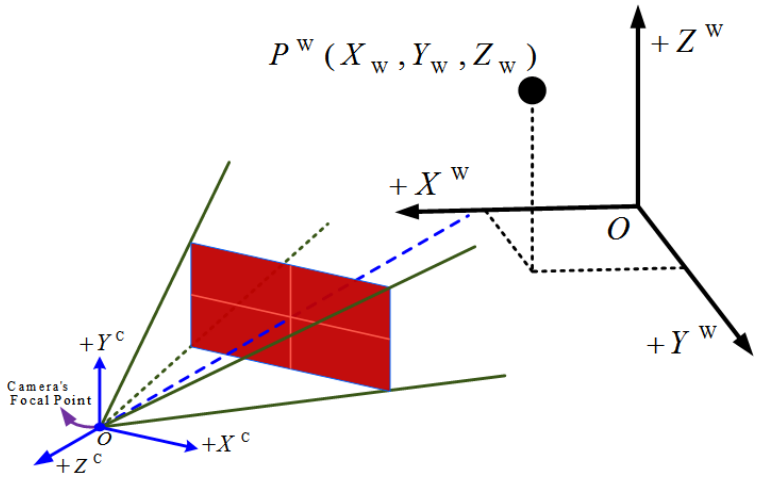


Figure 2.5: Pinhole Camera in World Space

Now we have K, which helps map between camera 3D space and image 2D space. But we are still not able to employ it yet. The camera 3D space is respect to the camera sensor only. Neither can we directly tell the camera 3D coordinates of an object point, nor can we assign it. All we can do is to use the camera space as an intermediate between the image coordinates and world coordinates, which we could assign by ourselves.

Fig. 2.5 shows a pinhole camera observing an arbitrary object point P in the world space. We assign the world coordinates so that the object point has world space coordinates $P^W(X_w, Y_w, Z_w)$. Although the world space and camera space are two different spaces, we could easily transform between each other through rotation and translation, as long as both of the spaces are using rigid Cartesian Coordinates. With a standard rotation matrix R_{3*3} and a translation matrix T_{3*1}

$$R_{3*3} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \; T_{3*1} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}, \tag{2.7}$$

we can get the transformation matrix $[R_{3*3}\ T_{3*3}]$ from the world space to camera space, as shown in eqn. 2.8.

$$\begin{bmatrix} X^C \\ Y^C \\ Z^C \end{bmatrix} = R \begin{bmatrix} X^W \\ Y^W \\ Z^W \end{bmatrix} + T = \begin{bmatrix} R_{3*3} & T_{3*1} \end{bmatrix} \begin{bmatrix} X^W \\ Y^W \\ Z^W \\ 1 \end{bmatrix} \tag{2.8}$$

The parameters that help map from world space to camera space depend on how we assign

the world coordinates. Since none of them are from the camera even though they are belongs to an important part of camera calibration, usually the matrix $[R_{3*3}\ T_{3*3}]$ is called extrinsic camera matrix. With both of the extrinsic camera matrix (help map from world space to camera space) and the intrinsic camera matrix (help map from camera space to image space), we are now able to build the connection between the world space coordinates, which could be assigned by ourselves, and the image space *Row* and *Column*, which are the streams we retrieved from the camera. To combine the intrinsic camera matrix and extrinsic camera matrix (combine eqn. 2.5 and eqn. 2.8), we get

$$Z^C \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} = K \begin{bmatrix} X^C \\ Y^C \\ Z^C \end{bmatrix} = K \begin{bmatrix} R_{3*3} & T_{3*1} \end{bmatrix} \begin{bmatrix} X^w \\ Y^w \\ Z^w \\ 1 \end{bmatrix} = M \begin{bmatrix} X^w \\ Y^w \\ Z^w \\ 1 \end{bmatrix} \tag{2.9}$$

$$M = K \begin{bmatrix} R_{3*3} & T_{3*1} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix}, \tag{2.10}$$

where $M = K[R_{3*3}\ T_{3*1}]$, as written in eqn. 2.10.

Note that, although Z^C values can be retrieved from the depth sensor streams, they will be employed during the calculation of M, because they will be expressed by the third row parameters in matrix M. Z^C will only be used in the step of 3D reconstruction after the pinhole camera matrix M is determined, as will be discussed in details in section 2.2. Thus, Z^C in eqn. 2.9 is commonly substituted as an intermediate parameter k. We did not change Z^C for the consistency of derivations. To inspect the pinhole camera matrix M, it is composed of rotation/translation matrix for 3D space transforming and intrinsic perspective matrix for handling both of perspective view mapping and shape-skewing, all of which belong to linear processing. In other words, this 3x4 transformation matrix is specially for handling perspective view, or perspective distortion. The pinhole camera model is based on the homogeneous coordinates, which means its matrix M is also limited by linear processing.

2.2 3D Camera Calibration

The calibration of a 3D camera aims to be able to generate the world coordinates (X^w, Y^w, Z^w) and corresponding *RGB* values for every single pixel, given the depth steams and RGB streams retrieved from the 3D camera. From section 2.1, we know that the pinhole camera matrix M (eqn. 2.10) could help map from the world space to image space, however

not able to directly transform image space data to world space coordinates. In order to determine $X^w/Y^w/Z^w$ (based on eqn. 2.5 and eqn. 2.8), both of the intrinsic camera matrix and extrinsic camera matrix are needed, both of which are intermediate parameters and practically can only be determined through matrix M. Thus, the first job for 3D camera calibration is to solve the pinhole camera matrix M. To solve the pinhole camera matrix, we can use least squares fit with known 3D points (X^w, Y^w, Z^w) and their corresponding image points (R,C). With one point, based on eqn. 2.10 and 2.9, we can get two equations.

$$\begin{aligned} m_{11}X^w + m_{12}Y^w + m_{13}Z^w + m_{14} - m_{31}X^wC - m_{32}Y^wC - m_{33}Z^wC - m_{34}C = 0 \\ m_{21}X^w + m_{22}Y^w + m_{23}Z^w + m_{24} - m_{31}X^wR - m_{32}Y^wR - m_{33}Z^wR - m_{34}R = 0 \end{aligned} \tag{2.11}$$

There are totally 12 unknowns to solve, thus we need at least six points to solve the 3x4 pinhole camera matrix M. Using n-points least squares to solve the best fit, we can build a 2n equations matrix, given by eqn. 2.12.

$$\begin{bmatrix} X_1^w & Y_1^w & Z_1^w & 1 & 0 & 0 & 0 & 0 & -X_1^wC_1 & -Y_1^wC_1 & -Z_1^wC_1 & -C_1 \\ 0 & 0 & 0 & 0 & X_1^w & Y_1^w & Z_1^w & 1 & -X_1^wR_1 & -Y_1^wR_1 & -Z_1^wR_1 & -R_1 \\ X_2^w & Y_2^w & Z_2^w & 1 & 0 & 0 & 0 & 0 & -X_2^wC_2 & -Y_2^wC_2 & -Z_2^wC_2 & -C_2 \\ 0 & 0 & 0 & 0 & X_2^w & Y_2^w & Z_2^w & 1 & -X_2^wR_2 & -Y_2^wR_2 & -Z_2^wR_2 & -R_2 \\ & & & & & & & \vdots & & & & \\ X_n^w & Y_n^w & Z_2^n & 1 & 0 & 0 & 0 & 0 & -X_n^wC_n & -Y_n^wC_n & -Z_n^wC_n & -C_n \\ 0 & 0 & 0 & 0 & X_n^w & Y_n^w & Z_n^w & 1 & -X_n^wR_n & -Y_n^wR_n & -Z_n^wR_n & -R_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{2.12}$$

Considering that this matrix is build on homogeneous system, there is no unique solution. There can always be a total-zeros solution. To make the solution unique, we select $m_{34} = 1$, so that the homogeneous eqn. 2.12 could be changed into an inhomogeneous format like $AX = B$, where the known matrix A is a 2n*11 matrix and known matrix B is a 2n

vector, as eqn. 2.13 shows below.

$$\begin{bmatrix} X_1^w & Y_1^w & Z_1^w & 1 & 0 & 0 & 0 & 0 & -X_1^wC_1 & -Y_1^wC_1 & -Z_1^wC_1 \\ 0 & 0 & 0 & 0 & X_1^w & Y_1^w & Z_1^w & 1 & -X_1^wR_1 & -Y_1^wR_1 & -Z_1^wR_1 \\ X_2^w & Y_2^w & Z_2^w & 1 & 0 & 0 & 0 & 0 & -X_2^wC_2 & -Y_2^wC_2 & -Z_2^wC_2 \\ 0 & 0 & 0 & 0 & X_2^w & Y_2^w & Z_2^w & 1 & -X_2^wR_2 & -Y_2^wR_2 & -Z_2^wR_2 \\ & & & & & & & \vdots & & & \\ X_n^w & Y_n^w & Z_2^n & 1 & 0 & 0 & 0 & 0 & -X_n^wC_n & -Y_n^wC_n & -Z_n^wC_n \\ 0 & 0 & 0 & 0 & X_n^w & Y_n^w & Z_n^w & 1 & -X_n^wR_n & -Y_n^wR_n & -Z_n^wR_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} C_1 \\ R_1 \\ C_2 \\ R_2 \\ \vdots \\ C_n \\ R_n \end{bmatrix} \tag{2.13}$$

Using pseudo inverse, eqn. 2.13 can be solved by $X = (A^TA)^{-1}A^TB$, where X is an 11-elements vector and $X(1) \sim X(11)$ correspond to $m_{11} \sim m_{33}$. And the 3x4 pinhole camera matrix (eqn. 2.10) will be solved as eqn. 2.14.

$$M = \begin{bmatrix} X(1) & X(2) & X(3) & X(4) \\ X(5) & X(6) & X(7) & X(8) \\ X(9) & X(10) & X(11) & 1 \end{bmatrix} \tag{2.14}$$

After we get the perspective projection matrix M, the next step is to recover the intrinsic and extrinsic camera matrix K and $[R_{3*3}, T_{3*1}]$, with which we could generate the world coordinates $X^w/Y^w/Z^w$. Starting from the decomposition of eqn. 2.10 step by step, and we could get

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & \\ m_{21} & m_{22} & m_{23} & O_{3*1} \\ m_{31} & m_{32} & m_{33} & \end{bmatrix} + \begin{bmatrix} & & & m_{14} \\ & O_{3*3} & & m_{24} \\ & & & m_{34} \end{bmatrix}, \tag{2.15}$$

$$K[R_{3*3}\ T_{3*1}] = \begin{bmatrix} KR_{3*3} & O_{3*1} \end{bmatrix} + \begin{bmatrix} O_{3*3} & KT_{3*1} \end{bmatrix} \tag{2.16}$$

and

$$M_{3*3} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = KR_{3*3} \tag{2.17}$$

where O denotes zero matrices with their sizes noted by subscripts. From eqn. 2.7, we know that R_{3*3} is a standard rotation matrix, which has its property of orthogonal. Also from eqn. 2.6, we know that K is an upper triangular matrix. Thus, all of the above fit in the prerequisites of RQ decomposition, which is a technique that could help us decompose the M_{3*3} into the upper triangular intrinsic matrix K and rotation matrix R_{3*3}. After we got R_{3*3}, the translation matrix T_{3*1} could be determined with eqn. 2.16.

Now we find the way to determine both of the intrinsic camera matrix and the extrinsic camera matrix. With depth streams measuring Z^C, we are able to transform the 2D image data retrieved from the camera into 3D camera space point cloud by eqn. 2.5, and then generate the world space point cloud by eqn. 2.8. The basic pinhole camera model calculation is widely used in various camera calibration techniques. Based on different calibration systems, Zhengyou [17] classified those calibration techniques into four categories: unknown scene points in the environment (self-calibration), 1D objects (wand with dots), 2D objects (planar patterns undergoing unknown motions) and 3D apparatus (two or three planes orthogonal to each other).

Self-calibration technique do not use any calibration object, and can be considered as zero-dimension approach because only image point correspondences are required. Just by moving a camera in a static scene, the rigidity of the scene provides in general two constraints [10] on the cameras internal parameters from one camera displacement by using image information alone. Therefore, if images are taken by the same camera with fixed internal parameters, correspondences between three images are sufficient to recover both the internal and external parameters which allow us to reconstruct 3-D structure up to a similarity [9] [5].

One Dimension points-line calibration employs one dimension objects composed of a set of collinear points. With much lower cost than two dimensional or even three dimensional calibration system, using one dimension objects in camera calibration is not only a theoretical aspect, but is also very important in practice especially when multi-cameras are involved in the environment. To calibrate the relative geometry between multiple cameras, it is necessary for all involving cameras to simultaneously observe a number of points. It is hardly possible to achieve this with 3D or 2D calibration apparatus1 if one camera is mounted in the front of a room while another in the back. This is not a problem for 1D ob-

jects. Xiangjian [6] shows how to estimate the internal and external parameters using one dimensional pattern in the camera calibration. And Zijian [19] employed one dimensional objects as virtual environments in practical multiple cameras calibration.

2D and 3D objects calibration systems usually give better calibrations. P. F. Sturm *et al*. [12] presented a general algorithm for plane-based calibration that can deal with arbitrary numbers of views that observe a planar pattern shown at different orientations, so that almost anyone can make such a calibration pattern by him/her-self, and the setup is very easy. Both of Matlab and OpenCV have applied this two dimension plane calibration method in their applications. Zhengdong [15] compared this two dimension plane camera calibration method and self-calibration method. Hamid [1] applied this method into practical calibration and employed the calibrated camera into camera pose estimation and distance estimation application.

In 3D object calibration technique, camera calibration is performed by observing a calibration object whose geometry in 3D space is known for very good precision. Calibration can be done very efficiently [4]. The calibration objects usually consist of two or three planes orthogonal to each other. Paul [3] applied the three dimension object calibration in his PHD project. Mattia [11] wrote a detailed tutorial from building the 3D object (fig. 2.6) for calibration, to scanning using the calibrated camera. Fig. 2.7a shows how six points are selected for calibration and fig. 2.7b shows the 3D reconstruction after calibration.

Zhengyou Zhang, a Chinese professor of computer science, IEEE and ACM Fellow and a specialist in computer vision and graphics, has deep studies on camera calibration from one-dimension calibration to tree-dimension calibration [18] [16] [17]. The accuracy of calibration from 1D to 3D is getting better, but the calibration system setting-up needs more and more work and cost as well. One dimension object is suitable for calibrating multiple cameras at once. Two dimension planer pattern approaches seems to be a good compromise, with good accuracy and simple setup. Also using the three dimension method

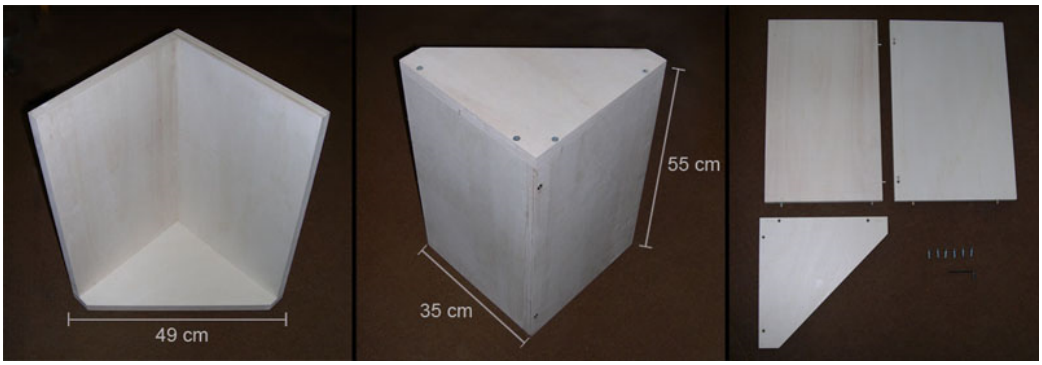
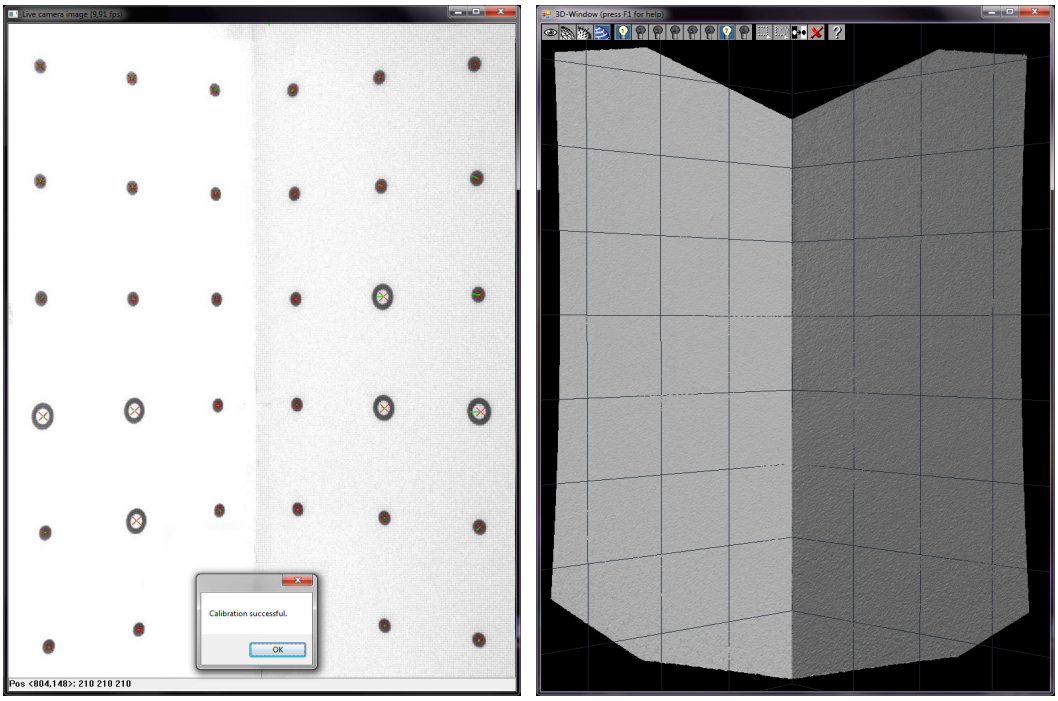


Figure 2.6: Building 3D Calibration Object [11]



(a) Six Points to Calibrate (b) Reconstruction After Calibration

Figure 2.7: Three Dimension Object Camera Calibration [11]

for calibration, Kai [8] derived the per-pixel beam equation, the linear relationship that could map to X^w/Y^w from Z^w as eqn. 2.18 shows, directly from pinhole camera matrix M for every single pixel. That is to say, we could easily look up X^w/Y^w after calibration once found the way to get Z^w.

$$\begin{aligned} X^w_{[row,col]} &= c_{[row,col]} Z^w_{[row,col]} + d_{[row,col]} \\ Y^w_{[row,col]} &= e_{[row,col]} Z^w_{[row,col]} + f_{[row,col]} \end{aligned} \tag{2.18}$$

where $c/d/e/f$ are per-pixel coefficients for the linear beam equations, and the subscripts $[row, col]$ are corresponding per-pixel image coordinates.

2.3 Lens Distortion

All above in Chapter 2 are talking about the ideal pinhole camera, without lenses. Whereas in practical, as a result of several types of imperfections in the design and assembly of lenses composing the camera optical system, there are always lens distortions for a camera and the expressions in eqn. 2.2 are not valid any more. Lens distortion could be classified into two groups [14] : radial distortion, and tangential distortion. Imperfect lens shape causes light rays bending more near the edges of a lens than they do at its optical center.

The smaller the lens, the greater the distortion. Barrel distortions happen commonly on wide angle lenses, where the field of view of the lens is much wider than the size of the image sensor. Improper lens assembly will lead to tangential distortion, which occurs when the lens and the image plane are not parallel. fig. 2.8 shows how radial distortion d_r and tangential distortion d_t affect the object point position in the image. Note that both of radial distortion and tangential distortion are with respect to image space row and column, and what we will take later is negative distortion instead of positive. Distortions are present because the field of view (FoV) in camera space has been affected by the lens. For most consumer RGB-D cameras with cheap lens, their distortions are usually barrel distortions (negative distortion) resulted by the enlarged field of view in the camera space, because the larger view was squeezed into the sensor. Fig. 2.9 intuitively shows how the lens enlarged the field of view of in the camera space and then generates the barrel distortions.

There are (a)(b)(c) three parts shown in fig. 2.9. Each part has the pinhole camera only on the top, in contrast to the camera-with-lens situation at the bottom. To understand how the barrel distortion happens, we should go through from part (c) to part (a). In part (c), the gray background uniform grid is the "object" our that the camera is going to observe, and the blue frames shows the FoV of the camera in the camera space. Due to the fact that, there will be worse and worse distortions as one pixel goes from the center to the edge, the enlarged FoV of a camera with lens in the camera space is in pincushion (or star) shape. With the enlarged Fov is mapped to the "Virtual Focal Plane", as defined in fig. 2.2, the pincushion shape doesn't change because rays from the camera space have not gone through the lens yet. Note that we quoted the "Virtual Focal Plane" because the image on

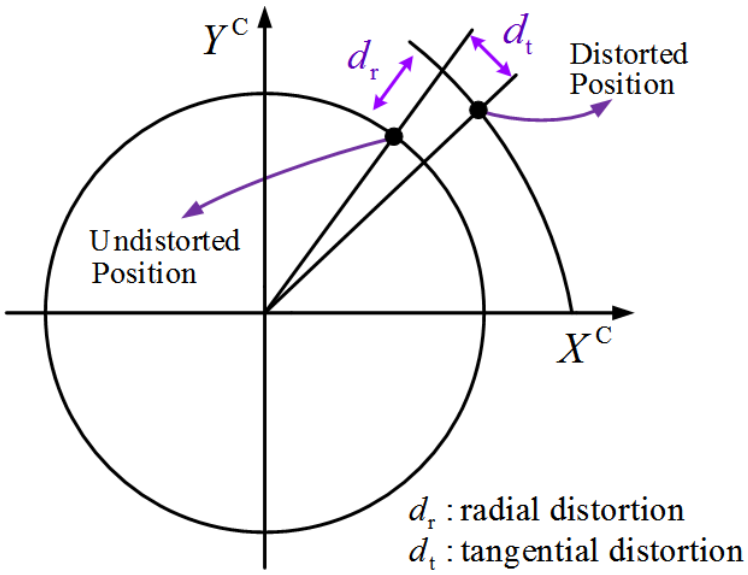


Figure 2.8: Radial and Tangential Distortion Affection In Image Space

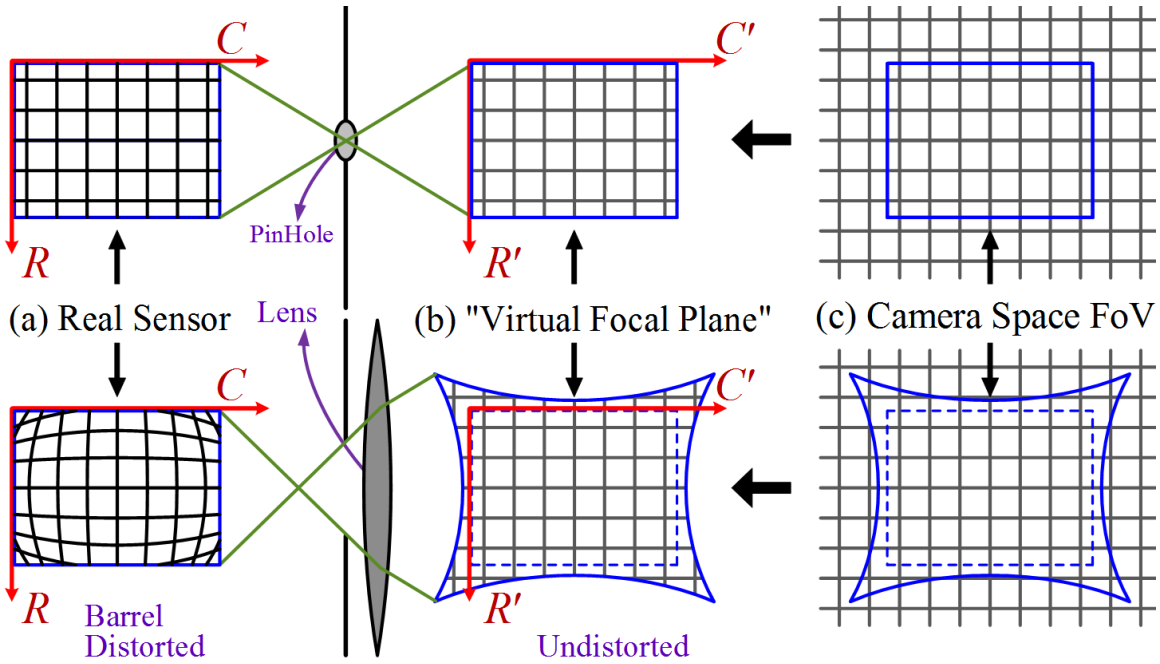


Figure 2.9: From Camera Space to Image Space with Lens Distortions

this virtual plane, when considering lens distortion, does not equal to the real focal plane (where the sensor is) any more. We can tell from part (b) that, even though the image space coordinates still are composed of *Col* and *Row*, their ranges have changed from positive integers only to the whole real integers that include negative ones. But the sensor never changes, and so the image space in part (a) still has its range of positive integers. With rays going through the lens, the pincushion-shape FoV (the frame in blue) will be squeezed into a small rectangle, and thus we get the image in the real focal plane with its background grid showing a barrel distorted shape.

With lens distortions counted, eqn. 2.2 now needs to be changed into

$$\begin{bmatrix} \mathrm{C'_r} \\ \mathrm{R'_r} \end{bmatrix} = f \begin{bmatrix} \mathrm{X_C}/\mathrm{Z_C} \\ \mathrm{Y_C}/\mathrm{Z_C} \end{bmatrix} \tag{2.19}$$

where $\mathrm{C'_r}$ and $\mathrm{R'_r}$ denote the relative pixel distance on the undistorted "Virtual Focal Plane", whose FoV is pincushion-shape and image coordinates' ranges include negative integers.

Duane [2] gave the lens distortion equation, and the undistorted *Col* and *Row* (C'/R' in our notation) can be expressed as power series in radial distance $r = \sqrt{C^2 + R^2}$:

$$\begin{aligned} C' &= C(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + [p_1(r^2 + 2C^2) + 2p_2 CR] \\ R' &= R(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + [p_2(r^2 + 2R^2) + 2p_1 CR] \end{aligned} \tag{2.20}$$

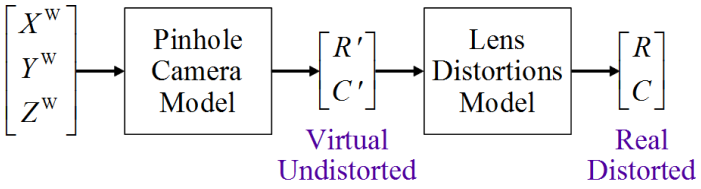


Figure 2.10: Traditional Camera Calibration Flow Char

where higher order parameters are omitted for being negligible; (C', R') denote the undistorted pixels in the "Virtual Focal Plane", (C, R) denote the distorted pixel in real sensor image, k_i's are coefficients of radial distortion, and p_j's are coefficients of tangential distortion. The five parameters $k_1/k_2/k_3/p_1/p_2$ are usually called distortion parameters. With the distortion parameters calculated, the distorted (C, R) could be undistorted into (C', R'), and then (C', R') could be used to generate the world space $X^w/Y^w/Z^w$ with intrinsic and extrinsic parameters.

2.4 Summation

In this chapter, a pinhole camera model is introduced in detail, which is expressed as a 3x4 matrix M that could help map from the world space 3D coordinate to image space 2D coordinates. The 3x4 matrix M can be decomposed into two separate matrix, the intrinsic matrix K and extrinsic matrix $[R_{3*3}\ T_{3*1}]$, each of them corresponds to one of the two parts of the pinhole camera model. The extrinsic matrix $[R_{3*3}\ T_{3*1}]$ consists of parameters outside the camera, which help map from 3D world space to 3D camera space. While the intrinsic matrix K consists of parameters all from the camera itself, which help map from the 3D camera space to 2D image space. After the pinhole camera model, various camera calibration techniques are discussed, all of which are based on the determination of the pinhole camera matrix M and then the recovery of the intrinsic matrix K and extrinsic matrix $[R_{3*3}\ T_{3*1}]$. Considering the lens distortions in practical, a lens distortions correction model (with $k_1/k_2/k_3/p_1/p_2$ five parameters) is given to undistort the lens distortions.

Fig. 2.10 shows the flow chart of the whole traditional camera calibration method based on the pinhole camera model. Considering the lens distortions, both of the pinhole camera model (matrix M) and the lens distortions model (five parameters for undistortion) need to be determined. The pinhole camera model can help map from the world space (X^W, Y^W, Z^W) to the undistorted image space (R', C'), which are on the "Virtual Focal Plane" as noted in fig. 2.9. And the lens distortion model help remove the lens distortions by mapping from (R', C') to (R, C). The pinhole camera model can be determined by eqn. 2.14, and the lens distortion model could be determined by eqn. 2.20.

Chapter 3 Data-Based Real-Time 3D Calibration

From Chapter 2 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. 2.7a. 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. 3.1 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. 1.2b, even observing a flat wall there are still many bumps and hollows in the reconstructed 3D image.

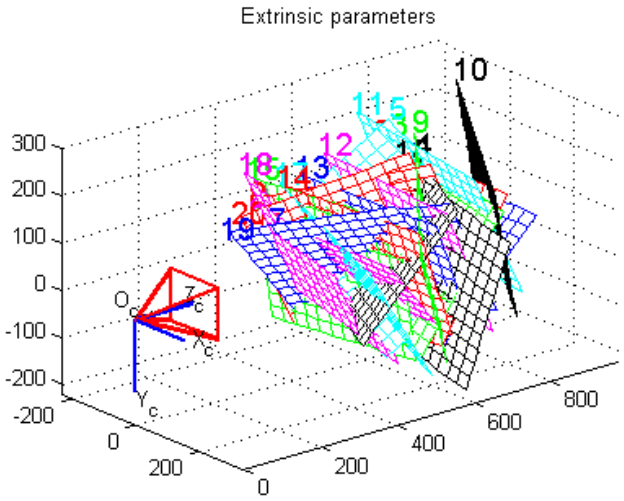


Figure 3.1: Simulated Planes of Checkerboards showing Extrinsic Parameters

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 2.18 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.

3.1 Novel Per-Pixel Calibration and LUT Reconstruction

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. 1.1, 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. 2.18), 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. 2.18 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 3.2, 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 distance in reality from the camera to dots-pattern plane along the rail. Fig. 3.2 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 $X^{\mathrm{W}}/Y^{\mathrm{W}}/Z^{\mathrm{W}}$ is 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. 2.20, which is 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 and then decide a best-fit mapping model.

With squared-shape distributed points (C, R) extracted from image space, we can re-

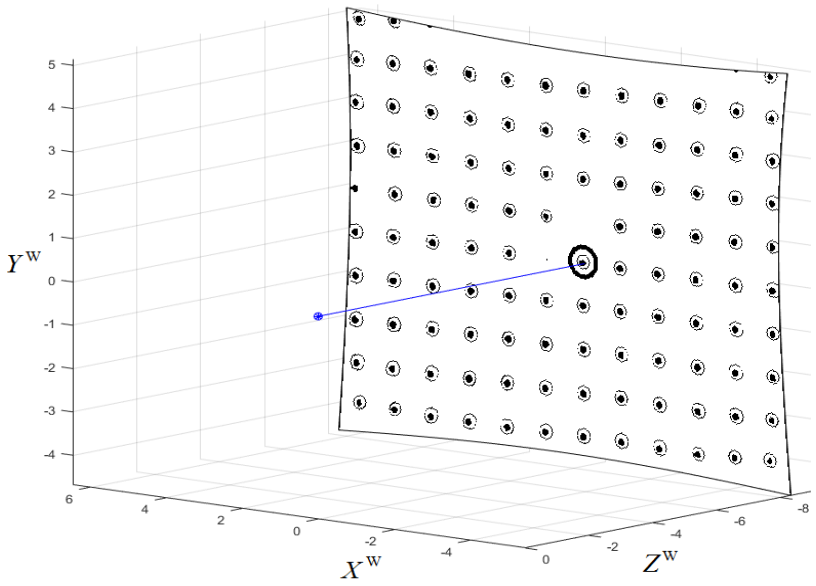


Figure 3.2: NearIR $X^W Y^W Z^W$ 3D Reconstruction

cover a distorted image in Matlab, as shown in fig.3.3a. Using a mathematical distortion (d) measurement [13]

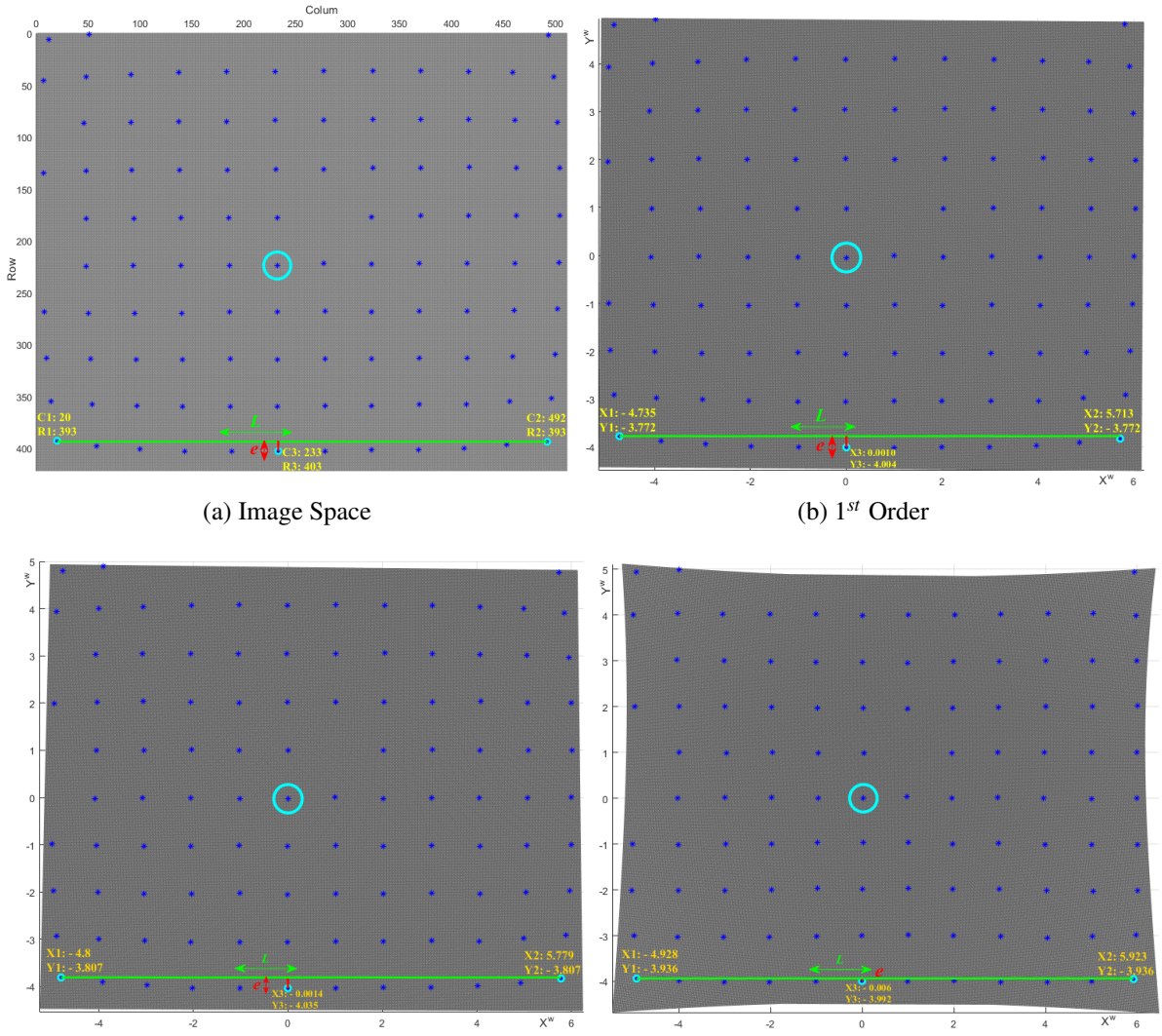
$$d(\%) = e * 100/L, \tag{3.1}$$

we can get the original distortion $d_0 = (R3-R1)/(C2-C1) = (403-393)/(492-20) = 2.1\%$. A 3x3 linear transformation matrix A (1^{st} order polynomial) is usually used for perspective distortion correction, give by eqn. 3.2.

$$\begin{bmatrix} zX^W \\ zY^W \\ z \end{bmatrix} = A \cdot \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} \tag{3.2}$$

Fig. 3.3b shows the corresponding 1^{st} order polynomial prototype, whose distortion $d_1 = (Y1-Y3)/(X2-X1) = [-3.772-(-4.004)]/[5.713-(-4.735)] = 2.2\%$, as expected, is not getting smaller at all. After the 1^{st} order polynomial, both of the second order and fourth order polynomial mappings are discussed. The second order polynomial mapping has 2x6=12 parameters, written as

$$\begin{aligned} X^W &= a_{11}C^2 + a_{12}CR + a_{13}R^2 + a_{14}C + a_{15}R + a_{16} \\ Y^W &= a_{21}C^2 + a_{22}CR + a_{23}R^2 + a_{24}C + a_{25}R + a_{26}\,, \end{aligned} \tag{3.3}$$



(a) Image Space

(b) 1^{st} Order

(c) 2^{nd} Order

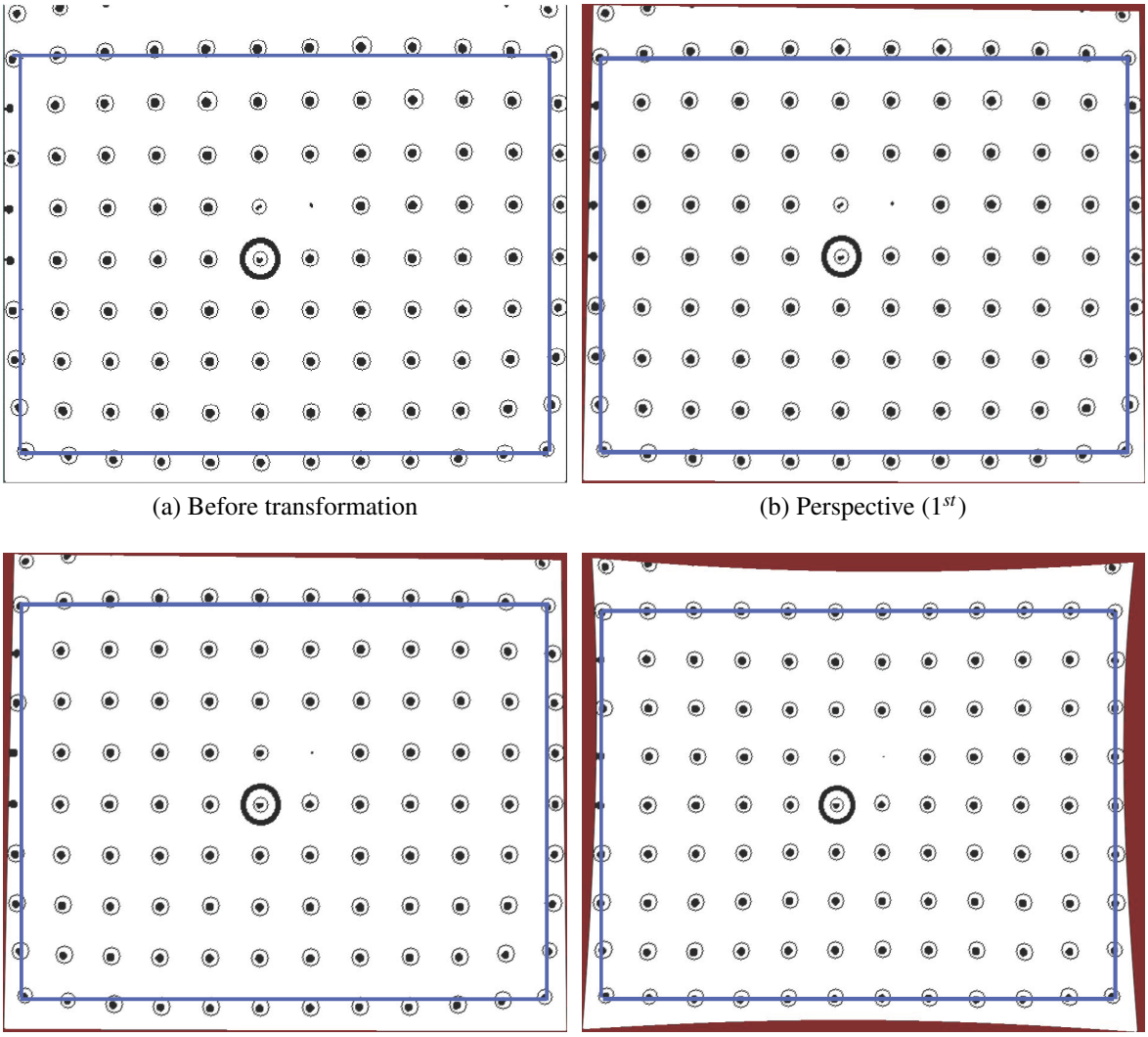
(d) 4^{th} Order

Figure 3.3: $X^W Y^W$ Matlab Polynomial Prototype

and similarly, the fourth order polynomial mapping has 2x15=30 parameters, given by eqn. 3.4.

$$\begin{aligned} X^W = {} & a_{11}C^4 + a_{12}C^3R + a_{13}C^2R^2 + a_{14}CR^3 + a_{15}R^4 + a_{16}C^3 + a_{17}C^2R \\ & + a_{18}CR^2 + a_{19}R^3 + a_{110}C^2 + a_{111}CR + a_{112}R^2 + a_{113}C + a_{114}R + a_{115} \end{aligned} \tag{3.4}$$

$$\begin{aligned} Y^W = {} & a_{21}C^4 + a_{22}C^3R + a_{23}C^2R^2 + a_{24}CR^3 + a_{25}R^4 + a_{26}C^3 + a_{27}C^2R \\ & + a_{28}CR^2 + a_{29}R^3 + a_{210}C^2 + a_{211}CR + a_{212}R^2 + a_{213}C + a_{214}R + a_{215} \end{aligned}$$



(a) Before transformation

(b) Perspective (1^{st})

(c) 2^{nd} Order

(d) 4^{th} Order

Figure 3.4: NearIR Stream High Order Polynomial Transformation

To prototype equation 3.3 and 3.4 in Matlab, "Curve Fitting Toolbox" is used to obtain the 2x6 and 2x15 parameters. 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. Fig. 3.3c and fig. 3.3d show the transformed images in world space respectively by the 2^{nd} order and 4^{th} order polynomial parameters, 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. 3.3d intuitively

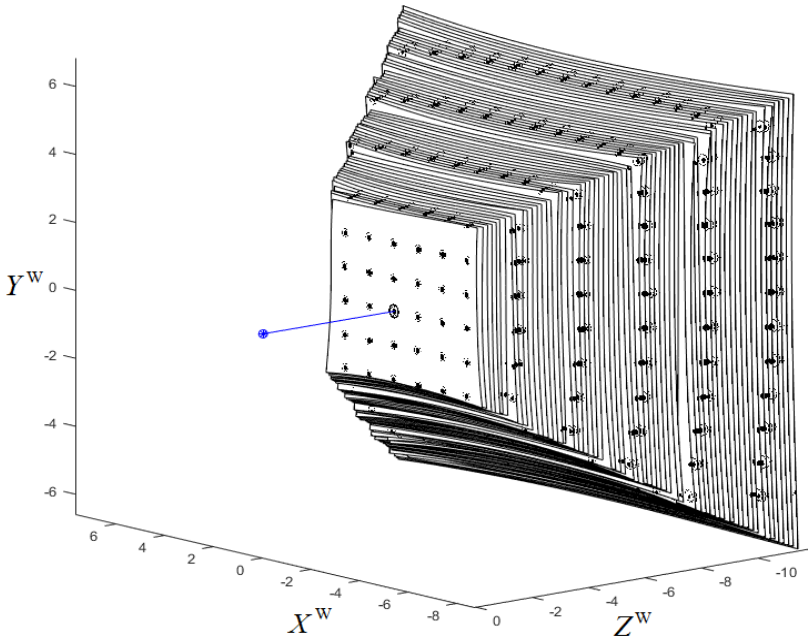


Figure 3.5: 63 Frames NearIR Calibrated 3D Reconstruction

shows a satisfying calibrated image.

Then, by applying those polynomial mappings into real-time streams transformation, we can get the transformed stream images. As shown in fig. 3.4, the outlines of the transformed steam images are same with Matlab prototypes in fig. 3.3. 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} could 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 (2x21=42) parameters 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 when the number of dot-clusters is too few to train its transformation model.

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. Fig. 3.5 shows 63 frames of collected data, which gives an pyramid shape of a camera sensor's field of view. With enough data, a best-fit mapping model from D to Z^{W} could be determined. Both of D and Z^{W} are continuous data, so that their function could written as a polynomial expression,

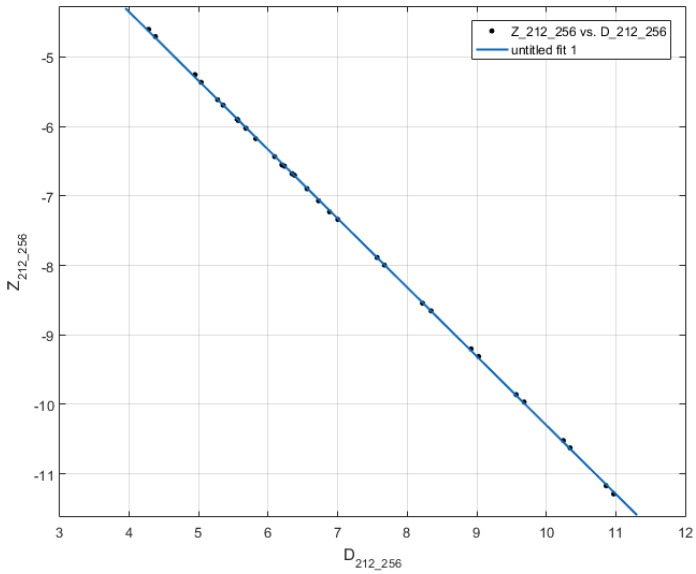


Figure 3.6: Polynomial Fitting between D and Z

based on Taylor series. Fig. 3.6 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 every single pixel, Z^{W} could be mapped from D through

$$Z^{W}_{[row,col]} = a_{[row,col]} D_{[row,col]} + b_{[row,col]}, \tag{3.5}$$

where $[row, col]$ denotes the address of a pixels, a/b are the corresponding linear coefficients that help map from D to Z^{W}.

The per-pixel X^{W} is determined by eqn. 3.5. And per-pixel $X^{W}Y^{W}$ will be determined by beam equation. 2.18. Fig. 3.7 shows some sample beams composed of coefficients $c/d/e/f$, which gives a undistorted field of view. To combine equation 3.5 and 2.18, 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. With enough data generating a $column$-by-row-by-6 look-up table that contains 6 coefficients $(a/b/c/d/e/f)$ for every single pixel, a calibrated real-time 3D reconstruction could be displayed.

3.2 DIP Techniques on (Col, Row) Extraction

As analyzed in chapter **??**, traditional camera calibration based on pinhole camera model is not able to supply RGB-D cameras a satisfying real-time solution for lens and depth distortions. In this chapter, instead of using any models, a data based look-up table (LUT) method is used, which will not only suit for both of NearIR steams and RGB steams,

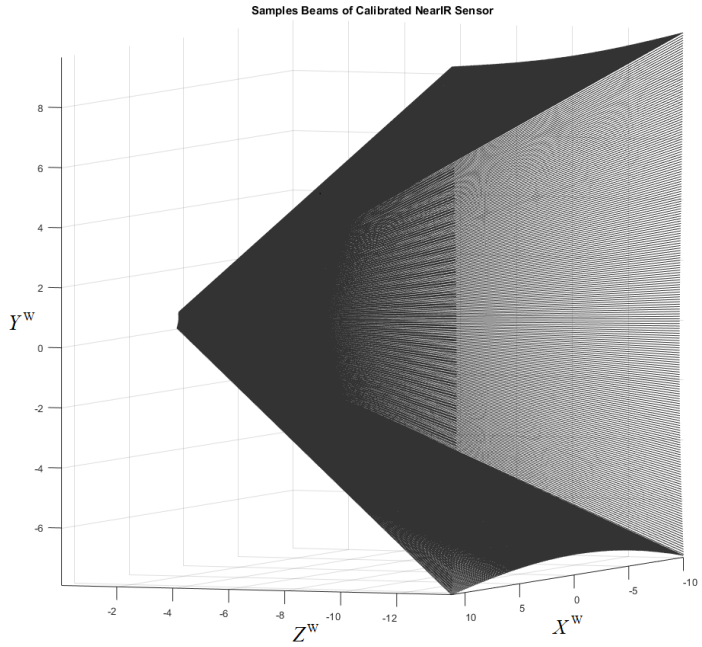


Figure 3.7: Sample Beams of Calibrated NearIR Field of View

but also for all of the universal RGB-D cameras. Rectifications of 3D world coordinates $X^W/Y^W/Z^W$ will be separated into two parts.

Above all, X^W and Y^W will be separately translated though two different transformation matrices that contains radial dominated lens distortions information. The transformation matrices are determined by pixels' coordinates-pairs (image plan coordinates *row* / *column* and corresponding world coordinates X^W/Y^W) that are extracted from captured uniform round dot grid pattern. Then, for each frame, external calibrated data from optical-flow sensor will be added as fixed Z^W for every pixel in the frame. After both of RGB stream and NearIR stream pixels find a match with world coordinate $X^W/Y^W/Z^W$, depth data will be finally added for each pixel to generate a table of XYZWRGB-D.

3.3 X^W/Y^W Calibration

The uniform round grid captured by RGB/NearIR steams contains radial dominated distortions information, which will be used for X^W/Y^W calibrations, to generate transform matrices to translate image plane rows and columns to world coordinates X^W/Y^W. In order to appropriately make use of the distortions information, the core mission is to locate the coordinates-pairs of each round dot.

The whole process of transformation matrices generation could be separated into three

steps. The first step is to track the $(column, row)$ of each round dot cluster's center captured by RGB/NearIR steams, and the second step is to determine the corresponding world coordinates (X^W, Y^W) for every $(column, row)$. After the coordinates-pairs are determined for every round dot cluster's center captured by RGB/NearIR steams, the last step is to use them to train high-order polynomial surface fitting models to generate transformation matrices, which could map from $(column, row)$ to (X^W, Y^W) for every single pixel of a frame. The round dot pattern consists of black dots and white background. As the simplest way to segment black dots from background, which is not 100% white in the captured image, thresholding (binarizing) is applied as pre-processing of the uniform grid's tracking, using Digital Image Processing (DIP) technologies. After an adaptive binarizing of a frame, a "sniffer" (edge modification) will be applied for captured clusters' centers determination, *i.e.*, the extraction of (column, row) as clusters' centers.

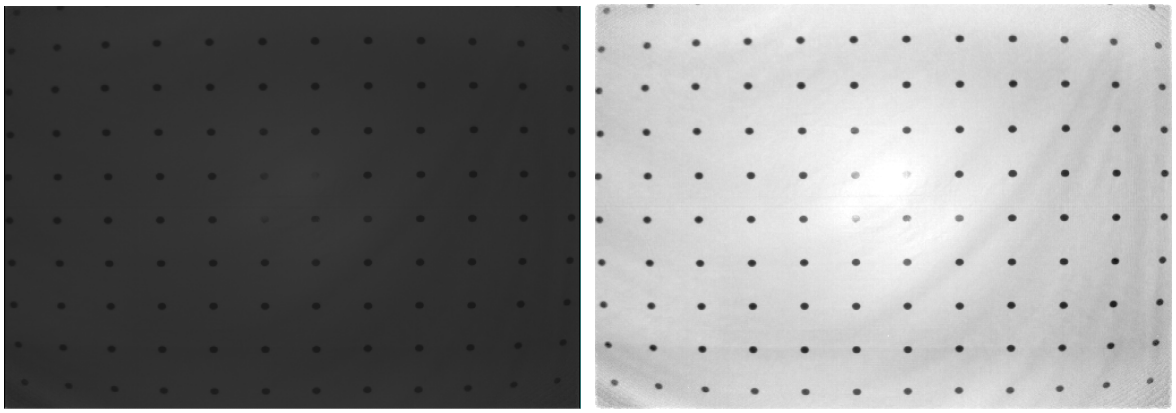
Digital Image Processings

In this section, DIP technologies are used for the goal of RGB/NearIR steams' binarizing. Considering that the many steps of processing will be applied on every single pixel of every frame, using GPU (parallel processing) has more advantages on handling this kind of mission than using CPU. OpenGL is selected as image processing language, and the default data type of steams saved on GPU during processing will be Single-Floating type, with a range from 0 (balck) to 1 (white). For both of RGB steam and NearIR steam, steam data need to be saved onto GPU first, during which progress gray-scale converting is also done.

"Converting RGB/NearIR to Gray-Scale"

In order to suit for both of the RGB and the NearIR steams, the first step of binarizing is to do gray-scaling. 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 could be written as

$$Intensity_{\_\text{gray}} = 0.21Red + 0.72Green + 0.07Blue \tag{3.6}$$



(a) Raw (b) Histogram Equalized

Figure 3.8: NearIR Streams before / after Histogram Equalization

"Histogram Equalization"

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. 3.8a (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 Function (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, as

$$Intensity = n/N * (1-0) + 0 = n/N \tag{3.7}$$

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

$$CDF(n) = \frac{sum}{N_{\text{\_Total Pixels/Image}}} \tag{3.8}$$

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.

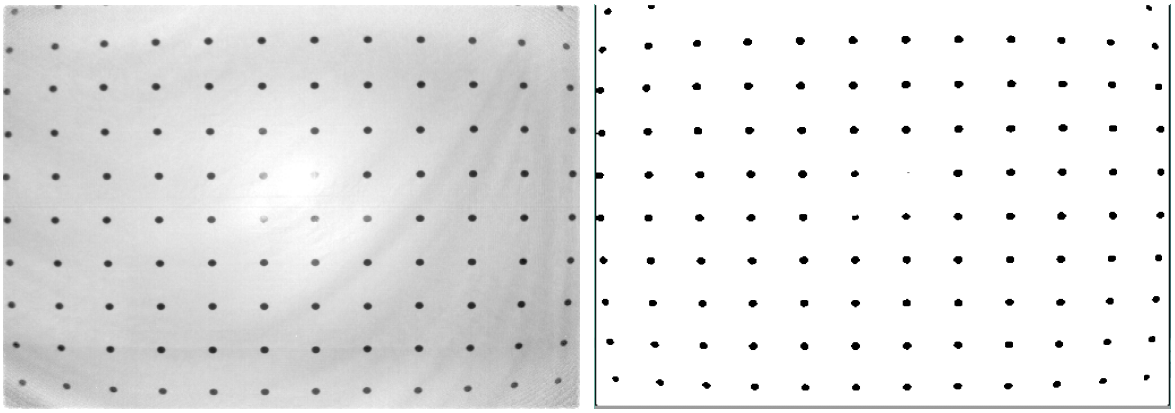
Then, at appropriate CDFs, e.g., $CDF(n_{\text{\_floor}}) = 0.01$ and $CDF(n_{\text{\_ceiling}}) = 0.99$, the intensities $floor$ and $ceiling$ could be written as

$$\begin{aligned} floor &= n_{\text{\_floor}}/N \\ ceiling &= n_{\text{\_ceiling}}/N \end{aligned} \tag{3.9}$$

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

$$Intensity_{\text{\_new}} = \frac{Intensity_{\text{\_original}} - floor}{ceiling - floor} \tag{3.10}$$

After this final rescaling step of Histogram Equalization, the new image gets better contrast effect, as shown in Fig. 3.9a



(a) Histogram Equalized (b) Binarized

Figure 3.9: NearIR Streams before / after Adaptive Thresholding

"Adaptive Thresholding"

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, using fixed thresholding will generate too much noise around borders, and an adaptive thresholding process is needed. To segment the black dots from white background, we could simply subtract an image's background from an textured image. And an image's background comes from a blurring process on 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

$$Intensity_{\text{\_binarized}} = \begin{cases} 1, & I_{\text{\_textured}} - I_{\text{\_background}} - C_{\text{\_offset}} > 0 \\ 0, & else \end{cases} \tag{3.11}$$

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. 3.9,

background is removed in the binarized image after adaptive thresholding.

Sniffer for Round Dot Center

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.

$$I_{\text{\_Path Marked}} = I_{\text{\_Original}} * \frac{(128I_{\text{A}} + 64I_{\text{B}} + 32I_{\text{C}} + 16I_{\text{D}} + 8I_{\text{E}} + 4I_{\text{F}} + 2I_{\text{G}} + I_{\text{H}})}{255} \tag{3.12}$$

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.

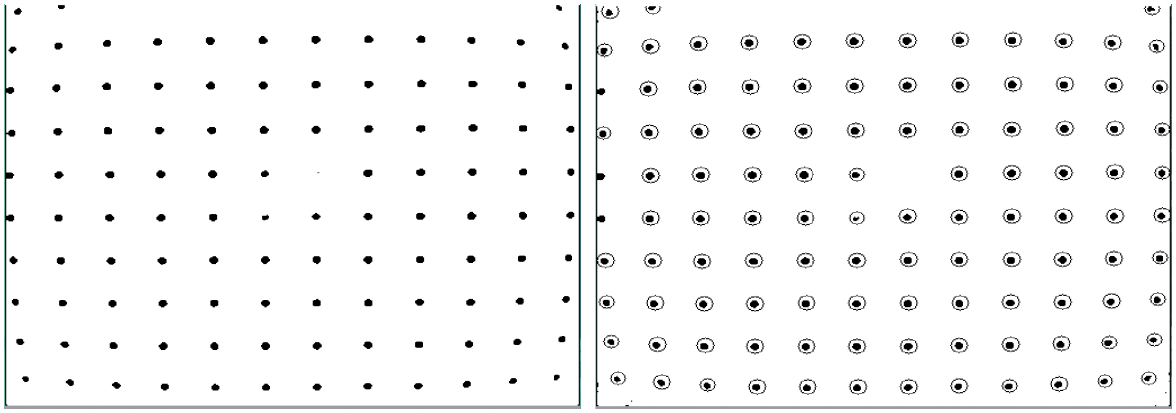
During the singular-dot analysis beginning from the anchor, very connected valid (non-zero) pixel will be a stop, and a "stops-address" queue buffer is used to save addresses of both visited pixels and the following pixels waiting to be visited. On very visit of a pixel, there is a checking procedure 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 3.12, 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.

$$\begin{aligned}
&if\ (I_{\mathrm{O}}\ \&\ \mathrm{0x80} == 1), \quad then, \quad \text{marker } A \text{ is valid} \quad (\text{ go Up }) \\
&if\ (I_{\mathrm{O}}\ \&\ \mathrm{0x40} == 1), \quad then, \quad \text{marker } B \text{ is valid} \quad (\text{ go Left}) \\
&if\ (I_{\mathrm{O}}\ \&\ \mathrm{0x20} == 1), \quad then, \quad \text{marker } C \text{ is valid} \quad (\text{ go Right }) \\
&if\ (I_{\mathrm{O}}\ \&\ \mathrm{0x10} == 1), \quad then, \quad \text{marker } D \text{ is valid} \quad (\text{ go Down })
\end{aligned} \tag{3.13}$$

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



(a) before Extraction (b) Extracted and Marked

Figure 3.10: Valid Dot-Clusters Extracted in NearIR

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. 3.10b, the centers of valid dot-clusters are marked within their corresponding homocentric circles. The list of round dot centers $(column, row)$s is extracted through section 3.3. The following is to map every dot center's $(column, row)$ to its corresponding world coordinates (X^W, Y^W). As shown in Fig. 1.1, world coordinates are from the uniform grid. Taking the side of unit-square (distance between two adjacent dots) as "Unit One" in the world coordinates and one dot as the origin of plan $X^W Y^W$, every dot cluster's center *column* / *row* will be mapped to integer values X^W/Y^W.

Ideally, a 3x3 perspective transformation matrix could help set a linear mapping between two plane coordinates, and 3 dot centers with know coordinates pair of $(column, row)$ and (X^W, Y^W) are enough to determine the transformation matrix. Once four points with a squared-shape *column*/*row* distribution is found, a 3x3 perspective transformation matrix A could be determined by solving eqn. 3.2, where C and R are vectors consist of $(column, row)$s of four squared-shape distributed points; X^W and Y^W are vectors consist of four points $(0,0)$, $(0,1)$, $(1,1)$, and $(1,0)$; z denotes the third axis in the homogenous system connecting two planes.

Due to the distortions, cluster centers in image plane are not uniformly distributed, and this 3x3 transformation matrix can only generate corresponding decimal X^W/Y^W values that are close integers. But in practical, the correct integer values X^W/Y^W could still be calculated through *Rounding*. The list of cluster centers' image coordinate $(column, row)$s can give many groups of four squared-shape distributed points, while each of them gives a different image coordinate distance that maps to the "Unit One" in world coordinates. Taking those points with generated X^W/Y^W values that are within an appropriate range close to integers as valid points, the "best" 3x3 transformation matrix could be determined by going through all of the possible groups of four squared-shape distributed points and picking out the group that leads to the most valid points.

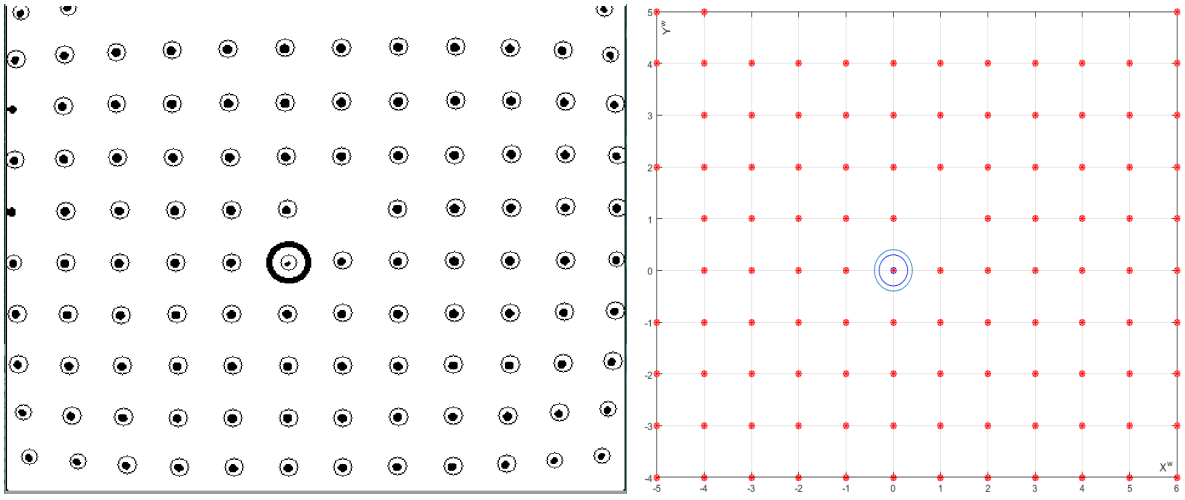
In this way, the so-called "best" transformation matrix can give a best "Unit One" distance in image coordinate, however its corresponding origin point $(0,0)$, one of the four points that are used to calculate a transformation matrix, is usually not close to the center of cameras' Field of View (FoV). A translation matrix T could be used to refine the "best" transformation matrix, and help to translate the origin point to be a dot cluster that is closest to the center of FoV. The refined transformation matrix $A_{\text{\_refined}}$ is written as

$$A_{\text{\_refined}} = T \cdot A = \begin{bmatrix} 1 & 0 & -X_{\text{\_Zero\_A}} \\ 0 & 1 & -Y_{\text{\_Zero\_A}} \\ 0 & 0 & 1 \end{bmatrix} \cdot A \tag{3.14}$$

where the integer world coordinate point $(X_{\text{\_Zero\_A}}, Y_{\text{\_Zero\_A}})$ are mapped from the center point $(C_{\text{\_center}}, R_{\text{\_center}})$ of FoV in image plane by the so-called "best" transformation matrix A, written as

$$\begin{bmatrix} zX_{\text{\_Zero\_A}} \\ zY_{\text{\_Zero\_A}} \\ z \end{bmatrix} = A \cdot \begin{bmatrix} C_{\text{\_center}} \\ R_{\text{\_center}} \\ 1 \end{bmatrix} \tag{3.15}$$

Eventually, the refined transformation matrix eventually generates a list of world coordinate points (X^W, Y^W)s that correspond to image coordinates $(column, row)$s. As shown in Fig. 3.11b, world coordinates are integers and the origin (blue circle) is chosen as where the center-closest dot-cluster is sitting.



(a) ImagePlane Coordinate (b) World Coordinate

Figure 3.11: Coordinates-Pairs: (Row, Column)s and (X^W, Y^W)s

3.4 Alignment from Depth Sensor to RGB Sensor

3.5 Real-Time analysis

Real-time analysis is in contrast with off-line process. In terms of camera steams process, real-time means to display the first processed frame before the second frame processing starts. In this thesis, "real-time" could be used to describe both of the 3D calibration procedure for Look-Up Table (LUT) generation and the live show of calibrated observation using LUT. In the process of 3D (X^W, Y^W, Z^W) calibration using an uniform grid dots pattern, "real-time" means being able to show a calibrated frame, as well as to record its calibrated $XYZWRGBD$ steams data, before the start of second frame processing. The recored frames ($XYZWRGBD$s) will be used to generating LUT.

With the help of the BLE Optical-Flow tracking module, which will be discussed in section **??**, this calibration step could really be called "real-time": not only X^W and Y^W are calibrated before the coming of the next frame, but also Z^W is updated (at a rate of 100Hz, maximum 5KHz) as the slider moving along the rail (along Z-axis). After an appropriate setting, a group (concretely, 63 frames) of calibrated $XYZWRGBD$ steams data could be recorded automatically as the slider moving (record one calibrated frame at every 25mm) from the head of the rail to the tail, as already shown in Fig. 3.5.

The processing speed depends on the number of detected dot-clusters. The further where

the camera is observing the dots pattern, the more dot-clusters are detected, the slower processing speed will be. In practical, the real-time calibration speed 4 fps at the head of the rail (working distance 1.05m), and 0.4 fps at the end of the rail (working distance 2.7m). For the live-show of observed steams after calibration, the camera's view is no longer limited by the uniform grid pattern. At that time, "real-time" means to look up the calibrated (X^W, Y^W, Z^W) based on D and read its corresponding RGB for every single pixel, and then show the reconstructed 3D image, before the next frame.

Concretely, for every single pixel, 6 coefficients (a,b,c,d,e,f) will be looked up, and their corresponding Z^W and X^W/Y^W will be determined by equation 3.5 and equation **??**.

Bibliography

[1] Hamid Bazargani. Camera calibration and pose estimation from planes. *IEEE Instrumentation & Measurement Magazine*, 18(6), Dec 2015.

[2] Duane C Brown. Decentering Distortion of Lenses. *Photogrammetric Engineering*, 32(3), May 1966.

[3] Paul Ernest Debevec. *Modeling and Rendering Architecture from Photographs*. PhD thesis, University of California at Berkeley, 1996.

[4] Olivier Faugeras. Three-Dimensional Computer Vision. *MIT Press*, Nov 1993.

[5] R. I. Hartley. An algorithm for self calibration from several views. Jun 1994.

[6] Xiangjian He. Estimation of Internal and External Parameters for Camera Calibration Using 1D Pattern. Nov 2006.

[7] Krystof Litomisky. Consumer RGB-D Cameras and their Applications. University of California, Riverside. 2012.

[8] Kai Liu. *Real-time 3-D Reconstruction by Means of Structured Light Illumination*. PhD thesis, University of Kentucky, 2010.

[9] Q.-T. Luong and O.D. Faugeras. Self-Calibration of a Moving Camera from Point Correspondences and Fundamental Matrices. *International Journal of Computer Vision*, 22(3), Mar 1997.

[10] Stephen J. Maybank and Olivier D. Faugeras. A theory of self-calibration of a moving camera. *International Journal of Computer Vision*, 8(2), Aug 1992.

[11] Mattia Mercante. Tutorial: Scanning Without Panels. In *DAVID-Wiki*, Jun 2014.

[12] P. F. Sturm and S. J. Maybank. On plane-based camera calibration: A general algorithm, singularities, applications. Jun 1999.

[13] 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.

[14] J. Weng, P. Cohen, and M. Herniou. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10), Oct 1992.

[15] Zhengdong Zhang. Camera calibration with lens distortion from low-rank textures. Jun 2011.

[16] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11), Nov 2000.

[17] Zhengyou Zhang. Camera Calibration. (Chapter 2). *Emergin Topics in Computer Vision*, 2004.

[18] Zhengyou Zhang. Camera calibration with one-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7), Jul 2004.

[19] Zijian Zhao. Practical multi-camera calibration algorithm with 1D objects for virtual environments. Apr 2008.