UNIVERSITY OF KENTUCKY

MASTER THESIS

Universal Real-Time XYZ rectified Reconstruction for RGB-D Cameras

Author:
Sen Li

Supervisor:
Daniel L. Lau

June 27, 2016

Contents

Chapter 1

Introduction

3D reconstruction aims to reproduce the 3D profile of real objects as accurate as possible, which require accurate world $X/Y/Z$ (noted as X^w /Y^w /Z^w henceforth) coordinate values in three dimensional space for every single point of a profile. Ever since the Kinect brought low-cost depth cameras into consumer market, with PrimeSense 3D sensing technology as the core depth determination principle for its first generation, great interest has been invigorated into RGB-D sensors. Camera calibration is a necessary part in 3D reconstruction in order to extract metric information from 3D images, i.e., to determine a translation from Z^w to X^w /Y^w for every pixel based on its row and column. In the mean time, optical and perspective distortion become a problem that stops from getting a good view. On most wide angle prime lenses and many zoom lenses with relatively short focal lengths, especially cheap low quality lenses, barrel distortion would typically be present.

In this research, a more accurate novel method with precise calibration system is brought in for real-time rectification and 3D reconstruction of universal RGB-D cameras.

1.1 3D Reconstruction

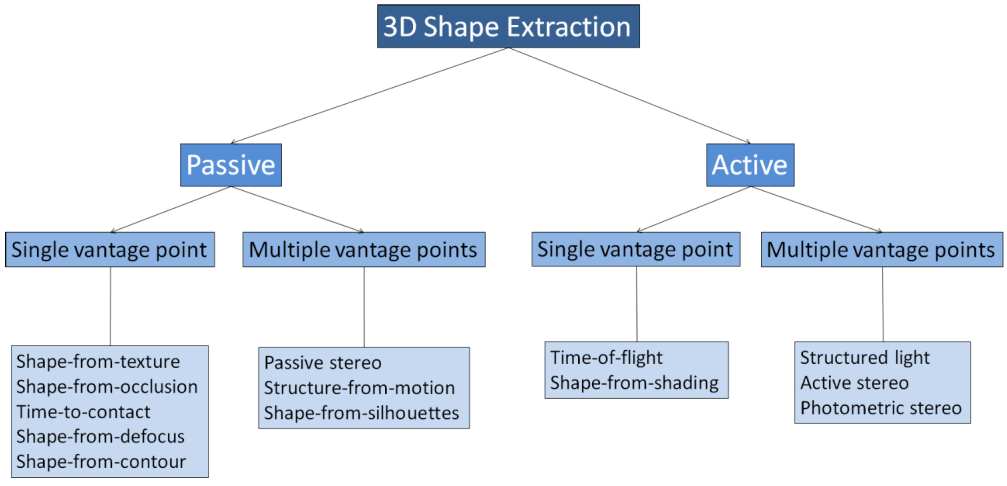


FIGURE 1.1: 3D profile acquisition Taxonomy

Three dimensional (3D) profile measurement technologies have been developed by various means, as summarized by Curless and Seitz [1], among which the non-contact optical methods are widely applied into reality as consumer RGB-D camera. Traditionally, with Pinhole camera model, as the basics of camera calibration, to supply the translation from Z^w to X^w /Y^w , the core procedure of 3D Reconstruction falls on the determination of per-pixel depth to serve as Z^w .

Within the non-contact optical category, as well as 3D reconstruction using multiple images, there are two levels of distinctions[2], as shown in the 3D profile acquisition taxonomy diagram is given in Figure 1.1.

The first distinction: active methods and passive methods. Their classifications are decided by the control of light sources. Active methods need special light sources control as part of the strategy to get 3D information, while on the other hand, passive techniques could work with whichever reasonable available ambient light. With a special known illumination offering more information to simplify some of the steps for 3D information acquiring process, active methods tend to be computationally less demanding. Both of the famous consumer PrimeSense and KinectV2 3D cameras, which are calibrated by the new proposed approach, are using active methods.

The second distinction: single-vantage points methods and multi-vantage points methods. The second distinction is determined by the number of vantage points. With a single vantage system, reconstruction is done based on single view point. In the case that there are multiple viewing or illumination components, all of them would be positioned very close to each other so that they could ideally coincide. For multi-vantage points methods, several viewpoints, with possible controlled illumination source positions, are involved. As contrast with the single-vantage points methods, the multi-vantage systems need the different components to be positioned far enough from each other.

Among the above non-contact optical methods, structured-light and time-of-flight methods are of the most practical importance. As will be discussed shortly, the PrimeSense technology and SeikowaveLCG camera use Stuctured light methods, and the KniectV2 camera uses Time-of Flight.

Structured Light

Structured light (SL) based techniques are famous for its fast speed. It is composed of one camera and one light pattern projector[3]. The projector projects a series of special known patterns onto a target, and the camera captures the corresponding images, which contain special information corresponded to the patterns from the projector. A decoding algorithm would be used to extract world coordinate information of the target object from the captured images, by analyzing the relationship among the camera, the projector and the target object using triangulation.

Being after accuracy, the most important issue for structured light method comes to the question of, how to design the projected patterns. In other words, how to design the coding algorithm and its corresponding decoding strategy will decide the final quality

of the reconstructed 3D profile. Various classified SL pattern strategies have been proposed, and are still being studied.

1.1.1 PrimeSense Structured Light

The PrimeSense 3D camera uses an infrared projector to project an infrared speckle pattern onto a target , as shown in figure 1.2,

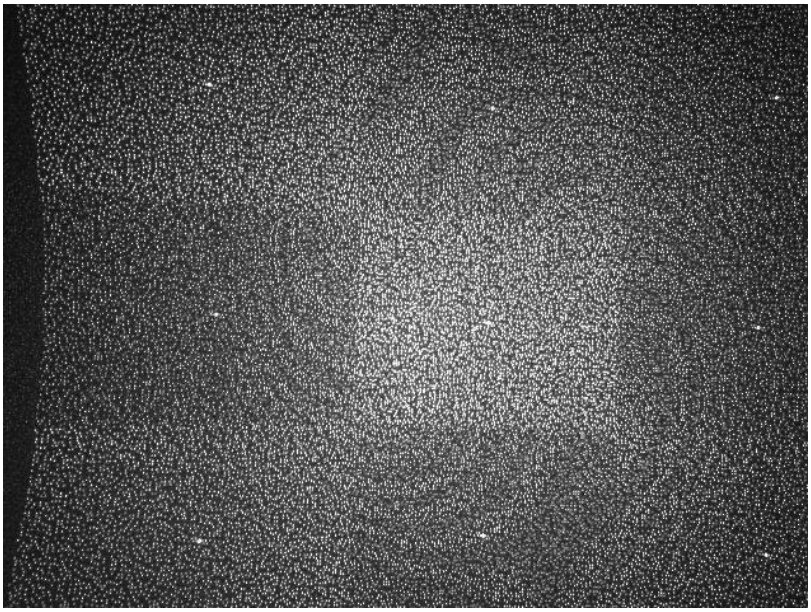


FIGURE 1.2: PrimeSense SL Infrared Pattern

and an infrared camera to capture images of the target. By comparing part by part to reference patterns, that were captured previously at known depths and stored in the device, the per-pixel depth could be looked up based on the reference pattern that the projected pattern matches best.

After the per-pixel depth data determined from the infrared sensor, the next step would be to correlate to a calibrated RGB data, which will generate a popular unified representation of target's profile: point cloud, a collection of points with XYZ 3D coordinates and RGB color data. What's more, the surface normals of the target's profile are also stored in every single point of the point cloud data.

1.1.2 SeikowaveLCG SL Phase Measuring Profilometry

SeikowaveLCG 3D camera consists of a Charge-Coupled Device (CCD) camera and a Digital Micro-mirror Device (DMD) projector. 2D image pattern strategies are always preferred for a fast scan if a is involved. [4][5] And the multi-shot pattern Phase Measuring Profilometry (PMP) strategy was used for its properties of robust and accuracy. With PMP information encoded in the structured light pattern projected onto the target,

the CCD camera could capture a series of images that contains PMP informatio. Triangulation analyzing could be used to extract the 3D world coordinates for each points of the target profile, by a determination of the relationships among CCD camera, DMD projector, and the target object. A system configuration of PMP application is given in figure 1.3.

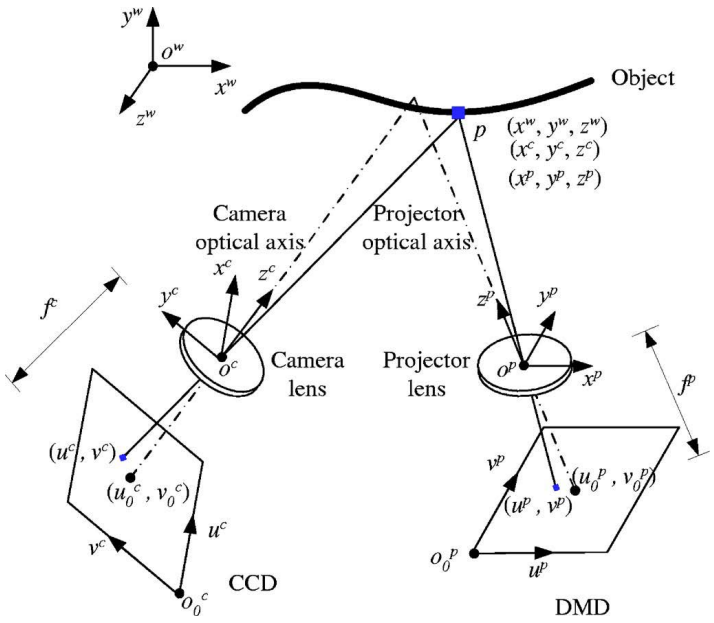
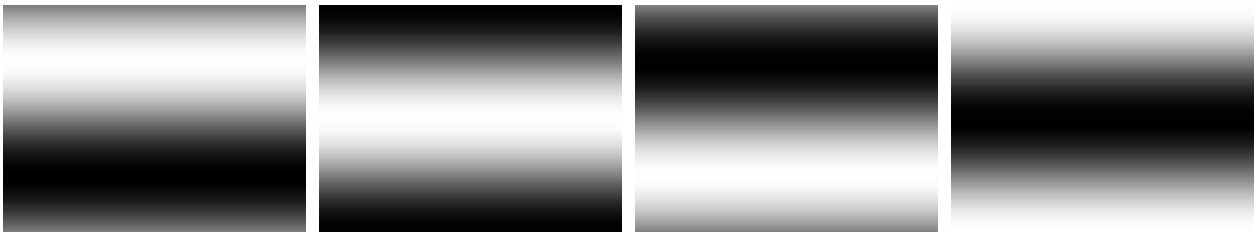


FIGURE 1.3: SL PMP based Configuration Diagram

PMP method uses either vertical or horizontal sinusoid patterns, which could be described as:

$$I_n^p(x^p, y^p) = A^p(x^p, y^p) + B^p(x^p, y^p)Cos(2\pi f y^p - \frac{2\pi n}{N}) \tag{1.1}$$

where (x^p, y^p) denotes the coordinates of every single pixel in the projector, I_n^p denotes the intensity of the corresponding pixels, A^p and B^p are constants, f is the frequency of sine wave. The subscript n represents the index of phase shift, while capital N is the total number of phase shift.



(A) 1 (B) 2 (C) 3 (D) 4

FIGURE 1.4: PMP base frequency patterns

Figure 1.4 shows a group of sine wave patterns, where the number of total phase shift $N = 4$ and frequency $f = 1$. From viewpoint of the camera, the sinusoid patterns is

distorted by the target surface topology, so that the captured images could be expressed as

$$I_n^c(x^c, y^c) = A^c(x^c, y^c) + B^c(x^c, y^c)Cos[\phi(x^c, y^c) - \frac{2\pi n}{N}] \tag{1.2}$$

where(x^c, y^c) denotes the coordinates of every single pixel in the camera, and the term $\phi(x^c, y^c)$ represents the corresponding phase value, which could be computed as follows [6]

$$\phi(x^c, y^c) = arctan\left[\frac{\sum_{n=1}^{N} I(x^c, y^c)Sin(2\pi n/N)}{\sum_{n=1}^{N} I(x^c, y^c)Cos(2\pi n/N)}\right] \tag{1.3}$$

After the camera term $\phi(x^c, y^c)$ for every single pixel is computed, the corresponding projector coordinate y^p could be derived through equation

$$y^p = \phi(x^c, y^c)/(2\pi f) \tag{1.4}$$

With the knowledge of y^p, the perspective information between camera and projector is the last step to go for applying triangulation analysis to extract world coordinates. Based on pinhole camera model, the perspective matrices for both of the CCD camera and DMD projector, as will be derived later in section 1.2.1 equation 1.20, are written as [7]

$$M^c = \begin{bmatrix} m_{11}^c & m_{12}^c & m_{13}^c & m_{14}^c \\ m_{21}^c & m_{22}^c & m_{23}^c & m_{24}^c \\ m_{31}^c & m_{32}^c & m_{33}^c & m_{34}^c \end{bmatrix} \tag{1.5}$$

and

$$M^p = \begin{bmatrix} m_{11}^p & m_{12}^p & m_{13}^p & m_{14}^p \\ m_{21}^p & m_{22}^p & m_{23}^p & m_{24}^p \\ m_{31}^p & m_{32}^p & m_{33}^p & m_{34}^p \end{bmatrix} \tag{1.6}$$

The mapping from 3D world coordinates to 2D camera coordinates are given by

$$x^c = \frac{m_{11}^c X^w + m_{12}^c Y^w + m_{13}^c Z^w + m_{14}^c}{m_{31}^c X^w + m_{32}^c Y^w + m_{33}^c Z^w + m_{34}^c} \tag{1.7}$$

$$y^c = \frac{m_{21}^c X^w + m_{22}^c Y^w + m_{23}^c Z^w + m_{24}^c}{m_{31}^c X^w + m_{32}^c Y^w + m_{33}^c Z^w + m_{34}^c} \tag{1.8}$$

Likewise, the translation from 3D world coordinates to 2D projector coordinates are given by

$$x^p = \frac{m_{11}^p X^w + m_{12}^p Y^w + m_{13}^p Z^w + m_{14}^p}{m_{31}^p X^w + m_{32}^p Y^w + m_{33}^p Z^w + m_{34}^p} \tag{1.9}$$

$$y^p = \frac{m_{21}^p X^w + m_{22}^p Y^w + m_{23}^p Z^w + m_{24}^p}{m_{31}^p X^w + m_{32}^p Y^w + m_{33}^p Z^w + m_{34}^p} \tag{1.10}$$

Since three out of four equations 1.7 ~ 1.10 are enough to solve X^w, Y^w, and Z^w, and y^p is already calculated, the 3D world coordinates $X^w/Y^w/Z^w$ could be derived from Eqs 1.7, 1.8, and 1.10

$$\begin{bmatrix} X^w \\ Y^w \\ Z^w \end{bmatrix} = \begin{bmatrix} m^c_{11} - m^c_{31}x^c, & m^c_{12} - m^c_{32}x^c, & m^c_{13} - m^c_{33}x^c \\ m^c_{21} - m^c_{31}y^c, & m^c_{22} - m^c_{32}y^c, & m^c_{23} - m^c_{33}y^c \\ m^p_{21} - m^p_{31}y^p, & m^p_{22} - m^p_{32}y^p, & m^p_{23} - m^p_{33}y^p \end{bmatrix}^{-1} \begin{bmatrix} m^c_{34}y^c - m^c_{14} \\ m^c_{34}y^c - m^c_{24} \\ m^p_{34}y^p - m^p_{24} \end{bmatrix} \quad (1.11)$$

1.1.3 Time of Flight (KinectV2)

Based on known speed of light, Time-of-Flight (ToF) camera resolves distance by measuring the time cost of a special light signal traveling between the camera and target for every single point. KinectV2 is one of the practical consumer 3D camera that applied the technology of ToF. Using the active modulated infrared source light together with a low-cost CMOS pixel array, KinectV2 realize its an attractive solution that owns compact construction, high accuracy and up to 30 fps frame-rate.

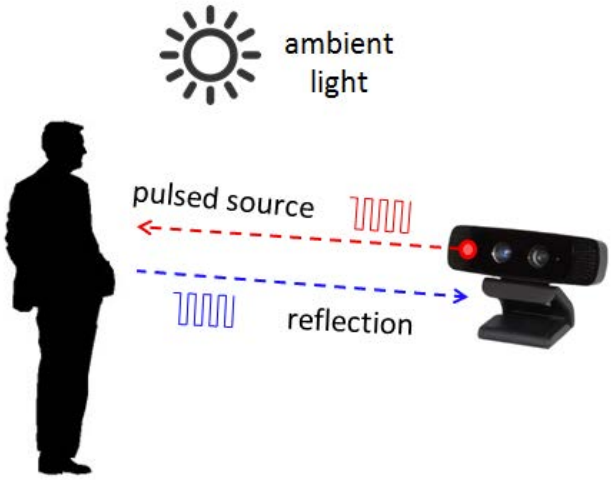


FIGURE 1.5: 3D time-of-flight camera operation

The variable that ToF camera measures is the phase shift between the illumination and reflection, which will be translated to distance [8]. To detect the phase shifts, light source is pulsed or modulated by a continuous wave, typically a sinusoid or square wave. As figure 1.5 shows, the ToF camera illumination is typically from a LED or a solid-state laser operating in the near-infrared range invisible to human eyes. A camera working in the same spectrum captures the reflected light and converts photonic energy to electrical signal, which contains distance (depth) information.

The distance measured for every single pixel is saved into a 2D addressable array, which results in a depth map. KinectV2 has a depth map of 512 * 424 unsigned short data collections, which could be finally rendered, together with corresponded RGB stream, into a tree dimensional space point cloud.

1.2 Traditional RGB-D camera calibration

Traditionally, camera calibration consists of two parts: ideal pinhole camera model calibration, and lens distortion correction. The pinhole camera model works as an simple algorithm in 3D computer vision to describe a mapping from the 3D world coordinates to camera image row and column, thereby giving a translation method from Z^w to X^w and Y^w for every single pixel. It works decently only for ideal pinhole cameras that have no lens, whereas real cameras need extra modifications and supplementations. In order to accurately solve the non-linear distortions problem for a real camera, the second part radial and tangential lens distortion must be considered.

1.2.1 Pinhole Camera Model

Figure 1.6 shows the basic diagram of a pinhole camera model with a reflected image plane for friendly intuition [9].From this model, the mapping between 3D space world coordinate and the image plane row and column could be separated into two parts of transformations. The first part is the transformation between world coordinates system $X/Y/Z$ and camera coordinate system $U/V/W$, which forms a 4x4 perspective transformation matrix (**extrinsic calibration**) that works for 3D rotation and translation. And the second part is the mapping between 3D camera coordinates system $U/V/W$ and 2D image plane coordinates u/v, which forms a 3x3 perspective transformation matrix (**intrinsic calibration**) that works for not only the rescaling between camera coordinates and virtual ideal image coordinates, but also for translating and skewing between the virtual ideal image plane and real image plane. **Extrinsic Calibration**

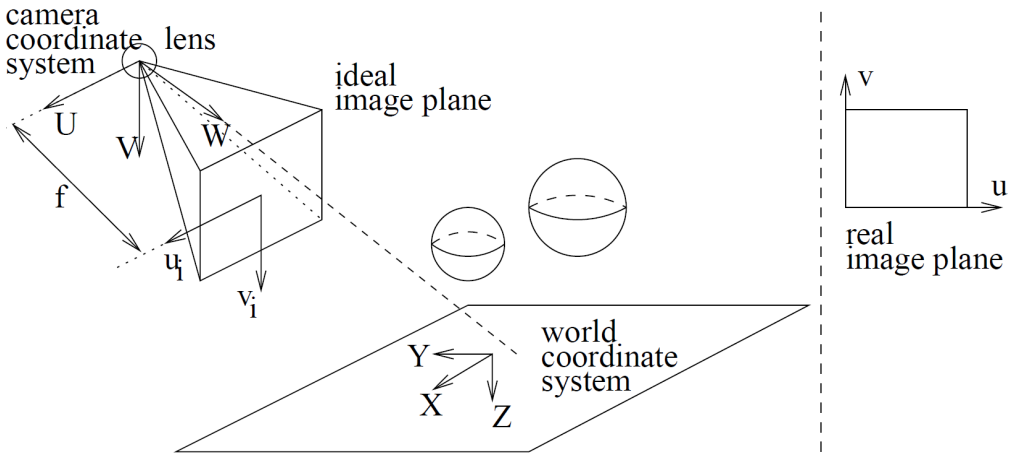


FIGURE 1.6: The pinhole camera model

Without any camera parameters, the extrinsic calibration formula could be written, through homogeneous coordinates, as

$$\begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} X^w \\ Y^w \\ Z^w \\ 1 \end{bmatrix} \tag{1.12}$$

or for simplicity,

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} R & T \end{bmatrix} \cdot \begin{bmatrix} X^w \\ Y^w \\ Z^w \end{bmatrix} \tag{1.13}$$

where $(U,V,W)^T$ are the camera coordinates, and the transformation matrix component $[R\ T]$, which is part of the 4x4 perspective matrix, is modelling rotation and translation , written as

$$\begin{bmatrix} R & T \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{1.14}$$

Intrinsic Calibration

Intrinsic Calibration could be separated into two sections. The first section is to rescale from camera coordinates to virtual ideal image coordinates. For a easier integration of two sections, the first section's formula is given through both of 2D coordinates to homogeneous coordinates.

2D coordinates:

$$W \begin{bmatrix} u_i \\ v_i \end{bmatrix} = f \begin{bmatrix} U \\ V \end{bmatrix} \tag{1.15}$$

Homogeneous coordinates:

$$W \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} fU \\ fV \\ W \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} U \\ V \\ W \end{bmatrix} \tag{1.16}$$

The second section is for translating and skewing between the virtual ideal image plane u_i/v_i and real image plane u_r/v_r,

$$\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = \begin{bmatrix} s_u & s_\theta & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \tag{1.17}$$

where (u_0, v_0) denotes the optical center (or principal point), $[s_u, s_v]$ are skew coefficients in pixels along u and v axes , and s_θ is an skewed angle generated by s_u and s_v. To combine equation 1.16 and equation 1.17, we could get

$$W \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = \begin{bmatrix} s_u & s_\theta & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} U \\ V \\ W \end{bmatrix} \tag{1.18}$$

where $[f_u, f_v]$ denote focal lengths in pixels along u and v after skewing, and s is a new skew coefficient after combination.

Generic Perspective Matrix of the Pinhole Camera Model

After both of the extrinsic and intrinsic transformation matrices have been derived, the generalization formula of the pinhole camera model could be derived, combining equation 1.13 and 1.18, as

$$k\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & T \end{bmatrix} \cdot \begin{bmatrix} X^w \\ Y^w \\ Z^w \end{bmatrix} = C \cdot \begin{bmatrix} X^w \\ Y^w \\ Z^w \end{bmatrix} \tag{1.19}$$

where W on the left side has been replaced by k to be a more general proportion coefficient, and a combined matrix can be expressed as

$$C = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & T \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \tag{1.20}$$

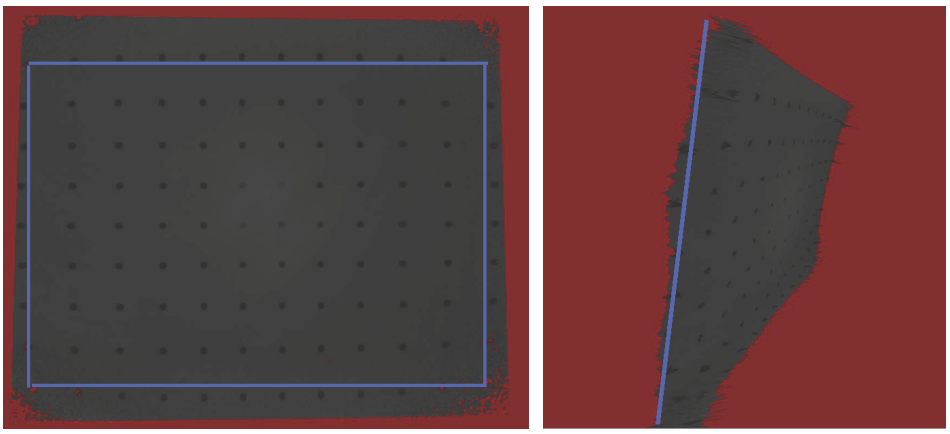
The final 3x4 matrix C is considered as the generic perspective transformation matrix of a pinhole camera model, which gives a mapping between the 3D world coordinates and 2D real image coordinates.

1.2.2 Lens distortion

Lens distortion could be classified into two groups [10] : 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.



(A) Front View (B) Left View

FIGURE 1.7: Raw NearIR 3D Reconstruction based on Depth Stream

Figure 1.7a shows the front view of raw NearIR steam 3D reconstruction, with a blue rectangle along the outermost layer of dot-clusters near the border. Due to the radial dominated distortions, most of the dot-clusters on the outermost layer are not

sitting on the four sides (straight lines) of the rectangle.

The lens distortion can be expressed as power series in radial distance $r = \sqrt{x^2+y^2}$:

$$\begin{aligned} x_{distorted} &= x(1+k_1r^2+k_2r^4+k_3r^6)+[p_1(r^2+2x^2)+2p_2xy] \\ y_{distorted} &= y(1+k_1r^2+k_2r^4+k_3r^6)+[p_2(r^2+2y^2)+2p_1xy] \end{aligned} \tag{1.21}$$

where higher order parameters are omitted for being negligible; $(x_{distorted}, y_{distorted})$ denote the distorted points, (x, y) denote the undistorted pixel locations, k_i's are coefficients of radial distortion, and p_j's are coefficients of tangential distortion.

1.3 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 1.2, the traditional way to reconstruct 3D point cloud not only has the time-cost problem, but also makes Z^w accuracy unguaranteed. First of all, the lens distortions correction is separated from pinhole camera model calibration. Even though same pixel-pair's of world coordinates and image plane coordinates could be re-utilized to solve radial dominated lens distortions, 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. What's worse, the depth resolution deteriorates notably with depth in practical [11], and 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 1.7b, 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.

In General, a rectification of 3D coordinates $X^w/Y^w/Z^w$ for every single pixel is needed to get a better view of a target's 3D profile. In this thesis, instead of using model based traditional method, an accurate data based $X^w/Y^w/Z^w$ rectified real-time reconstruction method is proposed.

As shown in figure 1.8, the RGB-D camera KinectV2 is mounted on the slider of the rail, which is perpendicular to the uniform round dot pattern as Z^w axis. With the round dot patten plane being a 2D plane of X^wY^w where Z^w being constant for all of the pixels, the center of each round dot maps to a fixed 2D world coordinates (X^w, Y^w). In Chapter 2, X^w and Y^w are rectified separately through a fourth order surface fitting translation from image plane row and column to directly solve the lens distortion problem in equation 1.21, considering that the parameters in that equation with power level larger than 4 are negligible. Then, Z^w values are totally supported from external BLE optical-flow sensor, which accurately tracks camera movements

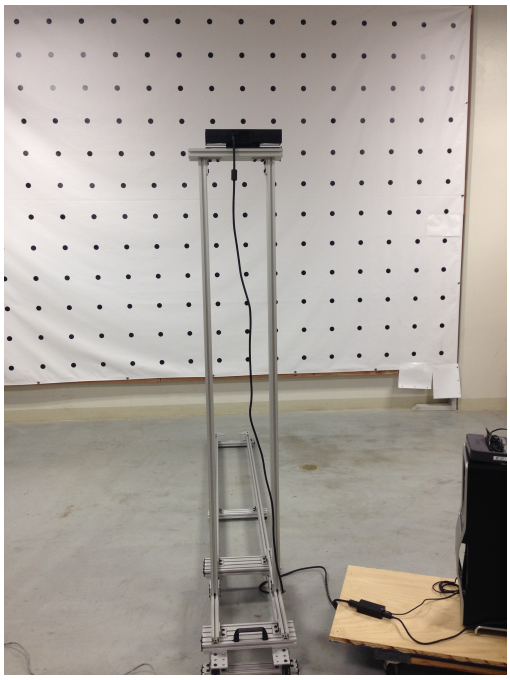


FIGURE 1.8: KinectV2 Calibration System

along Z-axis. Finally, a data-based XYZWRGB-D look-up table will be generated for real-time reconstruction. Whole calibration system will be introduced in detail in Chapter 3. Results will be shown in Chapter 4.

1.4 Summation

Chapter 2

Real-Time 3D Rectification

As analyzed in chapter 1, 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, 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.

2.1 X^w/Y^w Rectifications

The uniform round grid captured by RGB/NearIR steams contains radial dominated distortions information, which will be used for X^w/Y^w rectifications, 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.

2.1.1 Round Dot Center $(column, row)$ extraction

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

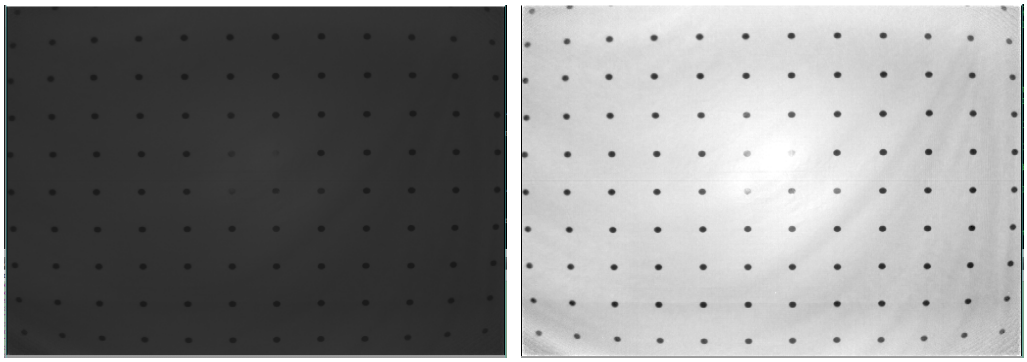
In order to suit for both of the RGB and the NearIR steams, the first step of binarizing is to do gray-scaling. For NearIR steam, its data contains only color gray. There is no need to consider gray-scale problem, and data will be saved on GPU as single-floating automatically. Whereas for RGB steam, 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

$$Intensity_{\text{\_gray}} = 0.21Red + 0.72Green + 0.07Blue \tag{2.1}$$

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 figure 2.1a (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.



(A) Raw (B) Histogram Equalized

FIGURE 2.1: NearIR Streams before / after Histogram Equalization

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

$$Intensity = n/N * (1-0) + 0 = n/N \tag{2.2}$$

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

$$CDF(n) = \frac{sum}{N_{\text{\_Total Pixels/Image}}} \tag{2.3}$$

where the gray level n is counted from the middle of the framebuffer's width to the end ($1 \sim N$). 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

$$\begin{aligned} floor &= n\_\text{floor}/N \\ ceiling &= n\_\text{ceiling}/N \end{aligned} \tag{2.4}$$

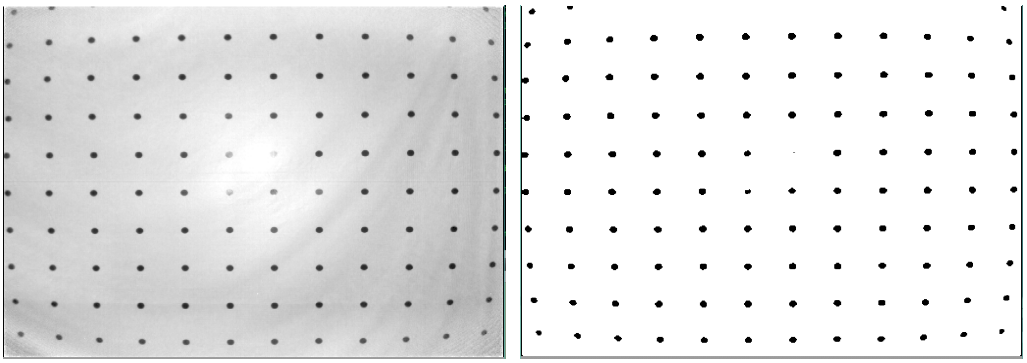
Finally, a new intensity value of every single pixel in an image could be rescaled as

$$Intensity\_\text{new} = \frac{Intensity\_\text{original} - floor}{ceiling - floor} \tag{2.5}$$

After this final rescaling step of Histogram Equalization, the new image gets better contrast effect, as shown in figure 2.2a

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.



(A) Histogram Equalized (B) Binarized

FIGURE 2.2: NearIR Streams before / after Adaptive Thresholding

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

$$Intensity\_\text{binarized} = \begin{cases} 1, & I\_\text{textured} - I\_\text{background} - C\_\text{offset} > 0 \\ 0, & else \end{cases} \tag{2.6}$$

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 figure 2.2, 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.

$$I_{\_\text{Path Marked}} = I_{\_\text{Original}} * \frac{(128I_\text{A} + 64I_\text{B} + 32I_\text{C} + 16I_\text{D} + 8I_\text{E} + 4I_\text{F} + 2I_\text{G} + I_\text{H})}{255} \tag{2.7}$$

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

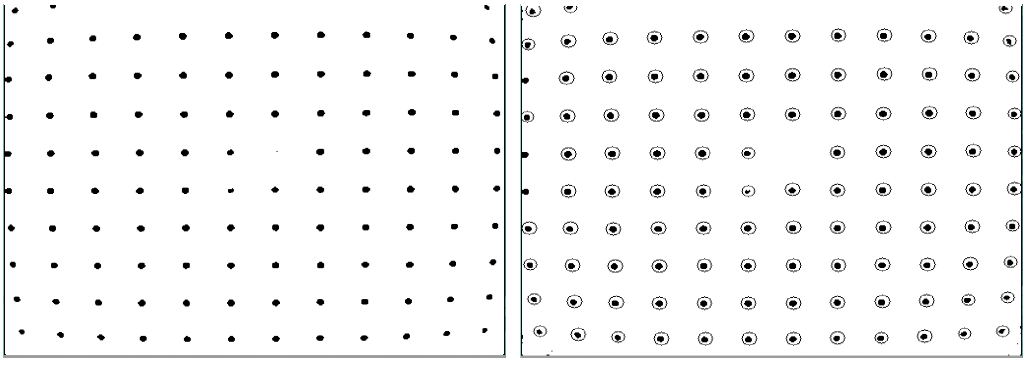
in this step may affect the processes of the other pixels (which cannot be a parallel processing), it is necessary to do it on CPU. The single-floating image data will be retrieved from GPU to a buffer on CPU as unsigned-byte data, waiting for inspection. And correspondingly the new CPU image will have 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 2.7, 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.

$$\begin{aligned} &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}80 == 1), \quad then, \quad \text{marker } A \text{ is valid} \ \ (\text{ go Up }) \\ &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}40 == 1), \quad then, \quad \text{marker } B \text{ is valid} \ \ (\text{ go Left}) \\ &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}20 == 1), \quad then, \quad \text{marker } C \text{ is valid} \ \ (\text{ go Right }) \\ &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}10 == 1), \quad then, \quad \text{marker } D \text{ is valid} \ \ (\text{ go Down }) \end{aligned} \tag{2.8}$$

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.



(A) before Extraction (B) Extracted and Marked

FIGURE 2.3: Valid Dot-Clusters Extracted in NearIR

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 figure 2.3b, the centers of valid dot-clusters are marked within their corresponding homocentric circles.

2.1.2 (X^w, Y^w) Fitting based on Uniform Grid

The list of round dot centers $(column, row)$s is extracted through section 2.1.1. The following is to map every dot center's $(column, row)$ to its corresponding world coordinates (X^w, Y^w). As shown in figure 1.8, 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^wY^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

$$\begin{bmatrix} zX^w \\ zY^w \\ z \end{bmatrix} = A \cdot \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} \tag{2.9}$$

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

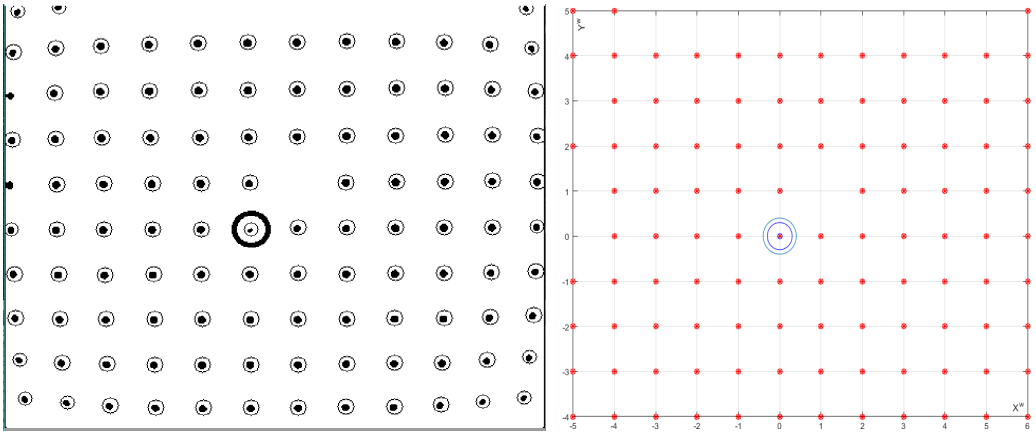
cluster that is closest to the center of FoV. The refined transformation matrix $A_{\_\text{refined}}$ is written as

$$A_{\_\text{refined}} = T \cdot A = \begin{bmatrix} 1 & 0 & -X_{\_\text{Zero\_A}} \\ 0 & 1 & -Y_{\_\text{Zero\_A}} \\ 0 & 0 & 1 \end{bmatrix} \cdot A \tag{2.10}$$

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

$$\begin{bmatrix} zX_{\_\text{Zero\_A}} \\ zY_{\_\text{Zero\_A}} \\ z \end{bmatrix} = A \cdot \begin{bmatrix} C_{\_\text{center}} \\ R_{\_\text{center}} \\ 1 \end{bmatrix} \tag{2.11}$$

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 figure 2.4b, 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 2.4: Coordinates-Pairs: (Row, Column)s and (X^w, Y^w)s

2.1.3 High Order Polynomial Surface Mapping

Using four squared-shape distributed dots, the 3x3 transformation matrix can help to find the integer value list of world coordinate X^w/Y^w. Further with all extracted dots' coordinate-pairs, a perspective distortion correction (1^{st} order polynomial with z bound) could be applied using equation 2.9. As shown in figure 2.6b, the 1^{st} order polynomial correction is not able to handle non-linear distorted mapping. Based on the lens distortion equation 1.21, a higher order two-dimensional polynomial transformation matrix is needed for both of x and y rectification. Considering that the parameters in that equation, which means a surface mapping polynomial function, with power level larger than four are practically negligible, both of the second order and fourth order polynomial mapping are discussed below (prototyped in Matlab and realized in real-time image plane rectification).

The second order polynomial mapping has 2x6=12 parameters, written as

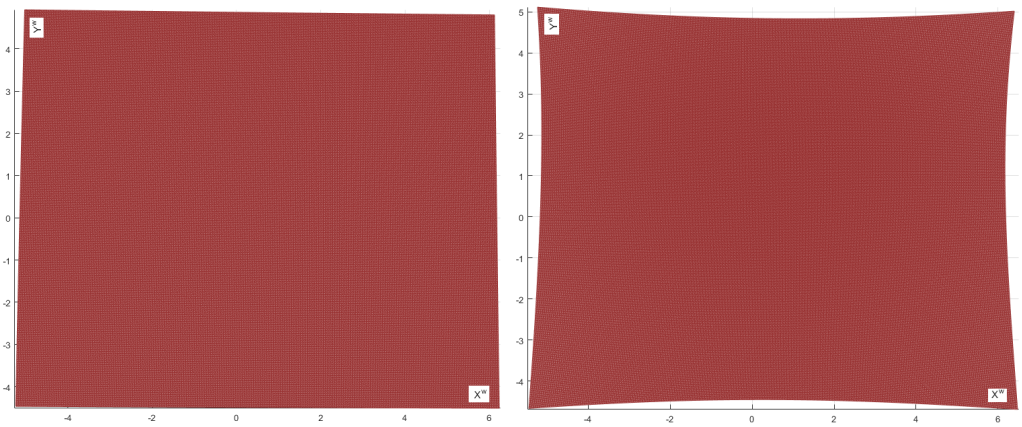
$$\begin{aligned} X^w &= a_{11}C^2 + a_{12}CR + a_{13}R^2 + a_{14}C + a_{15}R + a_{16} \\ Y^w &= a_{21}C^2 + a_{22}CR + a_{23}R^2 + a_{24}C + a_{25}R + a_{26} \end{aligned} \tag{2.12}$$

similarly, the fourth order polynomial mapping has 2x15=30 parameters,

$$\begin{aligned} X^w &= a_{11}C^4 + a_{12}C^3R + a_{13}C^2R^2 + a_{14}CR^3 + a_{15}R^4 + a_{16}C^3 + a_{17}C^2R \\ &\quad + a_{18}CR^2 + a_{19}R^3 + a_{110}C^2 + a_{111}CR + a_{112}R^2 + a_{113}C + a_{114}R + a_{115} \\ Y^w &= a_{21}C^4 + a_{22}C^3R + a_{23}C^2R^2 + a_{24}CR^3 + a_{25}R^4 + a_{26}C^3 + a_{27}C^2R \\ &\quad + a_{28}CR^2 + a_{29}R^3 + a_{210}C^2 + a_{211}CR + a_{212}R^2 + a_{213}C + a_{214}R + a_{215} \end{aligned} \tag{2.13}$$

where C and R are shorted for *Column* and *Row*.

To prototype equation 2.12 and 2.13 in Matlab, "Curve Fitting Toolbox" is used to obtain the 2x6 and 2x15 parameters. Once those two set of parameters are obtained, the rectified image could also be determined, as shown in figure 2.5,

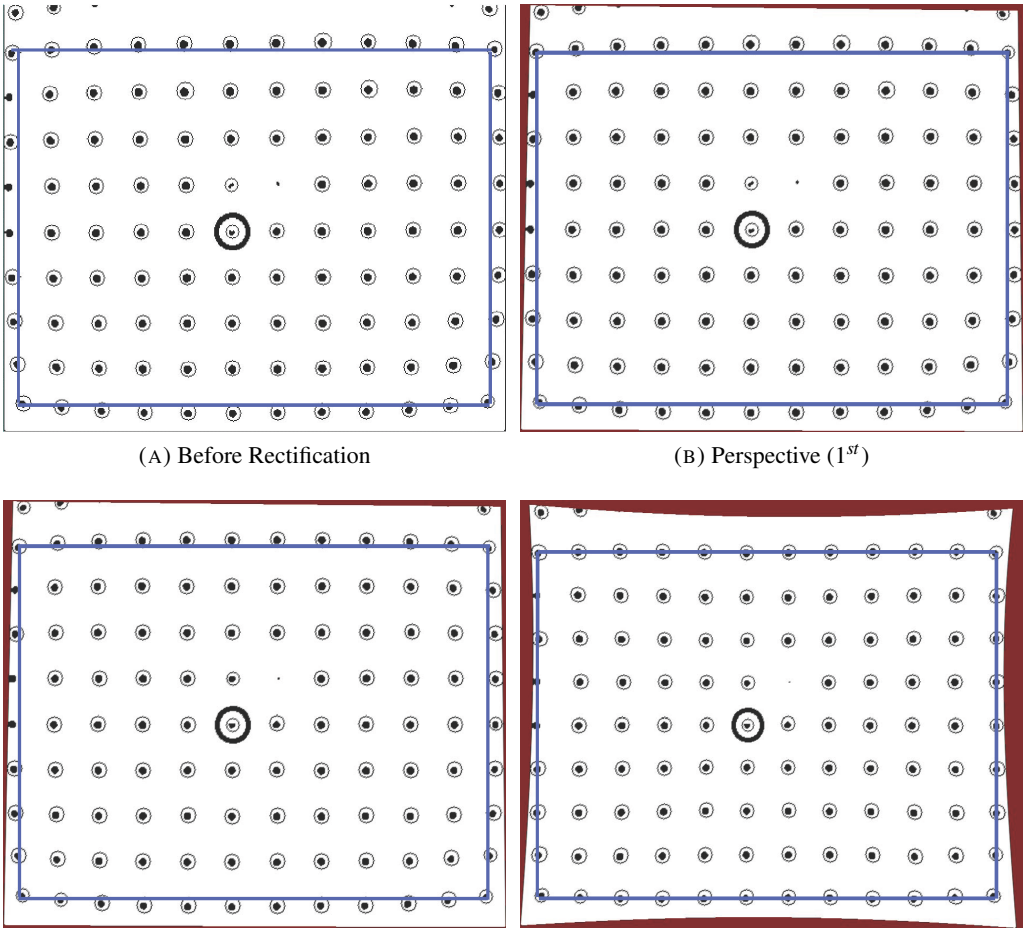


(A) 2^{nd} Order (B) 4^{th} Order

FIGURE 2.5: X^w/Y^w Matlab Prototype, 2^{nd} / 4^{th} Order Polynomial

where based on "Goodness of fit" 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.

Then, apply those polynomial surface mappings into real-time steams rectification. We can see the rectified steam image (figure 2.6c and figure 2.6d), whose image-outline are same with Matlab prototypes (figure 2.5a and figure 2.5b). Comparing among the four results shown in figure 2.6, the more dot-clusters sitting on the blue rectangle, the less distortion an image has.



(A) Before Rectification

(B) Perspective (1^{st})

(C) 2^{nd} Order

(D) 4^{th} Order

FIGURE 2.6: NearIR Stream High Order Polynomial Rectification

Not only from the comparison of RMSE, but more intuitively from figure 2.6, we can tell that the 4^{th} order polynomial surface mapping is much better than the second order (which is similar to the perspective correction). However, that more accurate process also cost more calculations, and requires more data (coordinate-pairs) to train the transformation model. 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 rectification is replaced by 2^{nd} order when the number of dot-clusters is too few to train its transformation model.

2.2 Z^w Rectification

Instead of from Depth steam, Z^w values will be determined directly from external real-world data. With the help of BLE Optical-Flow tracking module, as will be discussed later in section 3.2, Z^w values will be supported for every frame in real-time based on the camera's distance to the pattern. All of the pixels in one frame share the same Z^w

in world coordinate.

As well as the determination of X^w/Y^w, which has been discussed in section 2.1.2, Z^w value is also based on the uniform-grid's "Unit One", the side of unit-square of the grid pattern. 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 camera's working distance from the camera to dots pattern in reality offered by the BEL Optical-Flow tracking module. Choosing the origin to be where the camera is sitting, and the positives of X^wY^w to be right and up, the Z^w values are always negative. Figure 2.7 shows one frame of 3D reconstruction in world coordinates based on rectified X^w/Y^w and Z^w, where both of the origin and Z-axis are high-lighted as blue.

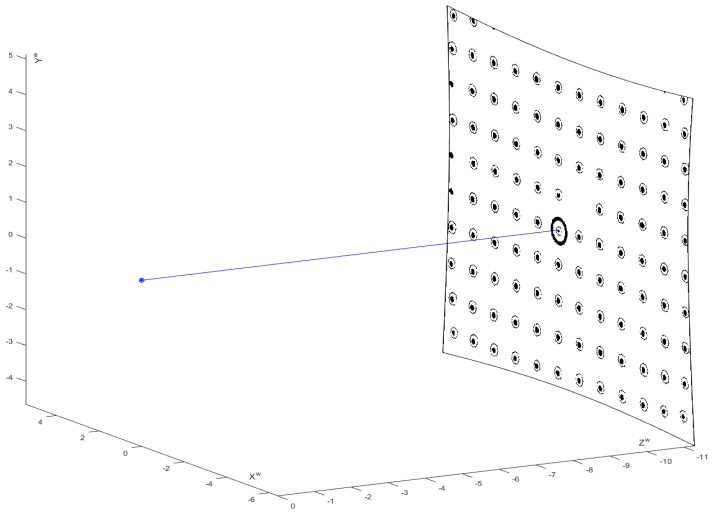


FIGURE 2.7: NearIR Rectified $X^w/Y^w/Z^w$ 3D Re-construction

2.3 Data-Based XYZWRGB-D Look-Up Table

In 3D re-construction, $R/G/B$ and D (short for $Depth$) values are given for every pixel in one frame, while the rectified $X/Y/Z$ (short for $X^w/Y^w/Z^w$ in this section) values are determined by processes of looking up and calculating, which is based on D. Therefore, the mission in this section is to find mappings from D to $X/Y/Z$. Considering that, both of D and Z denote the distance from the camera to the object (dots pattern), it is straight forward to start from the relationship between D and Z. As long as Z is obtained, X and Y for one single pixel could be determined through a linear relationship, due to the fact that the ray propagation is along a straight line.

2.3.1 From D to Z

Both of D and Z are continuous data, so that their function could written as a polynomial expression, based on Taylor series. Figure 2.8 shows the polynomial fitting result,

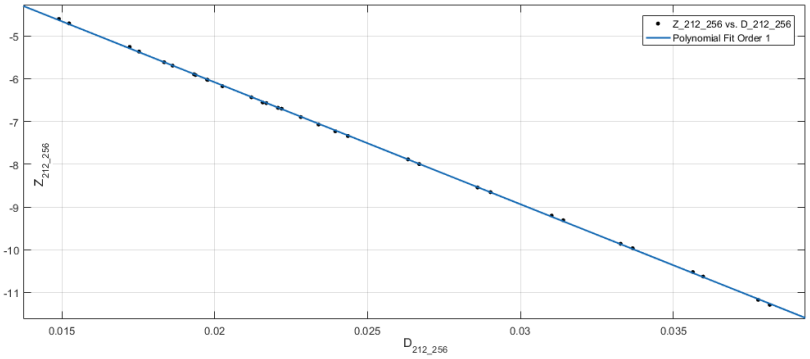


FIGURE 2.8: Polynomial Fitting between D and Z

with 32 *D*/*Z* values (at pixel *column*=256 and *row*=212) from 32 frames, from Matlab "Curve Fitting Tool" toolbox. It is apparent that Z is linear with D, which is also reasonable. Therefore, for every single pixel, Z could be retrieved from D through

$$Z(col, row) = a_{(col,row)}D(col, row) + b_{(col,row)} \tag{2.14}$$

where (col, row) denote the address of a pixels, $a_{(col,row)}$ and $b_{(col,row)}$ are the corresponding linear coefficients that help map from D to Z.

2.3.2 From Z to X/Y

For every possible Z, within the slider's range on the rail, X/Y values for all of the pixels are rectified (as discussed in section 2.1). As for every single pixel, its view, a beam in 3D space, determines the linear relationships for both of Z-X and Z-Y. Figure 2.9

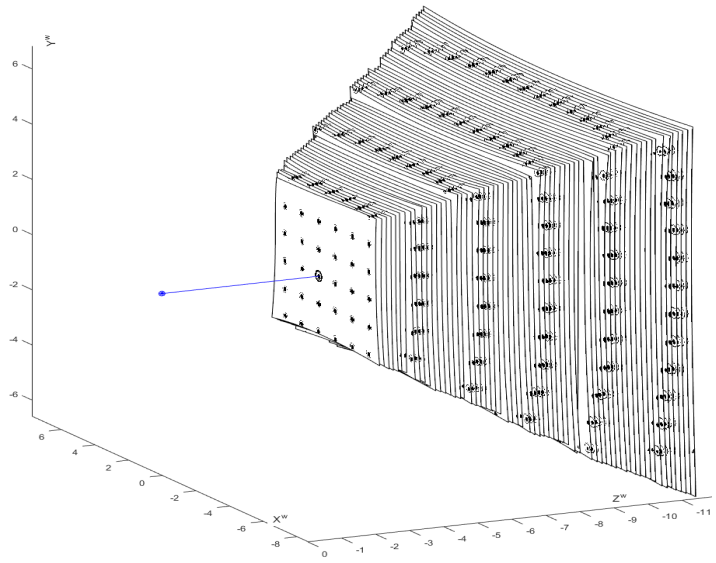


FIGURE 2.9: 63 Frames NearIR Rectified 3D Reconstruction

shows 63 frames of NearIR rectified 3D reconstruction, which gives an intuitionally pyramid shape of a camera sensor's field of view. The pyramid is composed of all of

the pixel-beams, every single one of which could be expressed as

$$\begin{aligned} X(col, row) &= c_{(col,row)}Z(col, row) + d_{(col,row)} \\ Y(col, row) &= e_{(col,row)}Z(col, row) + f_{(col,row)} \end{aligned} \tag{2.15}$$

where (col, row) denote the address of a pixels, $c/d/e/f$ are the corresponding linear coefficients that help map from Z to X and Y.

Figure 2.10 shows some sample beams composed of coefficients $c/d/e/f$, which gives a rectified field of view. It explains the lens distortions affections, that those beams are converged in front of the origin.

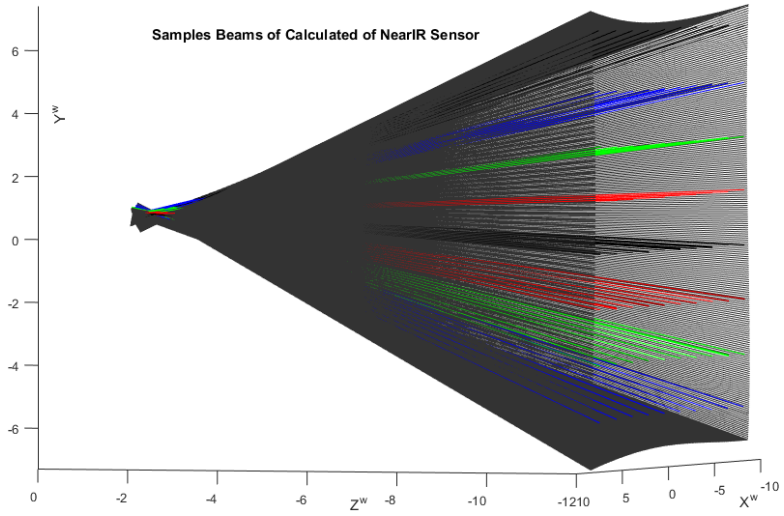


FIGURE 2.10: Sample Beams of Rectified NearIR Field of View

To combine equation 2.14 and 2.15, a rectified 3D coordinate (X, Y, Z) 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 rectified real-time 3D reconstruction could be displayed.

2.4 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 rectification procedure for Look-Up Table (LUT) generation and the live show of calibrated observation using LUT.

2.4.1 Real-Time in Rectification

In the process of 3D (X^w, Y^w, Z^w) rectification using an uniform grid dots pattern, "real-time" means being able to show a rectified frame, as well as to record its rectified $XYZWRGBD$ 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 3.2, this rectification step could really be called "real-time": not only X^w and Y^w are rectified 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 rectified $XYZWRGBD$ steams data could be recorded automatically as the slider moving (record one rectified frame at every 25mm) from the head of the rail to the tail, as already shown in figure 2.9.

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 rectification 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).

2.4.2 Real-Time Observing after Rectification

For the live-show of observed steams after rectification, the camera's view is no longer limited by the uniform grid pattern. At that time, "real-time" means to look up the rectified (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 2.14 and equation 2.15.

2.5 3D Reconstruction differences for PrimeSense, KinectV2, and Prosilica

pixels, steams, and shader comparison

Chapter 3

Calibration System for RGBD Cameras

3.1 Rail System

As already shown in chapter 1 figure 1.8, the whole calibration system consists of an RGB-D camera, a plane of round dot pattern, rail, and a BLE Optical-Flow tracking module. Other than the round dot pattern standing in front of the 3D camera for offering distortion information, all of the rest parts of the whole calibration system are centered on the rail, which is made with 80/20s. Taking the round dot pattern plane as plane X^wY^w, the rail is placed right perpendicular to the dot pattern along the direction of Z^w axis. Both of the RGB-D camera and the BLE Optical-Flow tracking module are mounted on the slider of the rail.

Sitting on the top of the elevated carriage of the slider, the RGB-D camera's vision keeps being horizontal, parallel with Z^w axis. And the origin is chosen right at the center of the camera's field of view, like figure 3.1 shows.

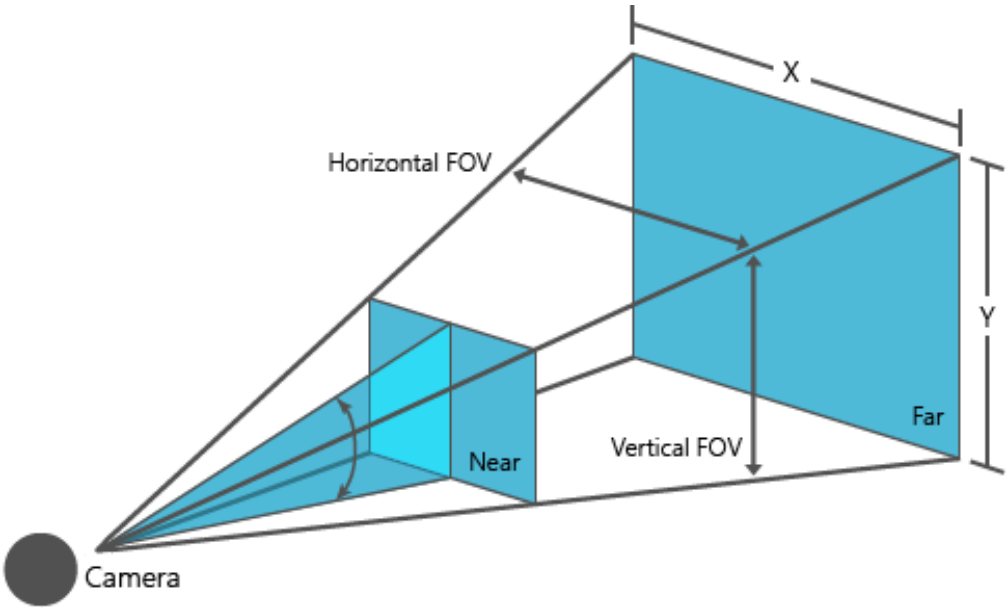


FIGURE 3.1: World Coordinate Frame

The BLE Optical-Flow tracking module is mounted at the bottom of the rail to

tracking the movements of the slider, as shown in figure 3.2. With infrared LED projecting injective rays onto the inner surface of the 80/20 groove, Optical-Flow sensor could generate accumulated X/Y value based on its observed optical flow changes (the changes of diffuse reflection rays from inner surface, generated by LED). The white re-stickable strip covering on the joint between PCB and OF sensor is for shocking absorption. Sliding along the rail, the accumulated Y value is always zero, and the accumulated X value records the movements of the slider, and will be sent into PC over the air by the BLE module.

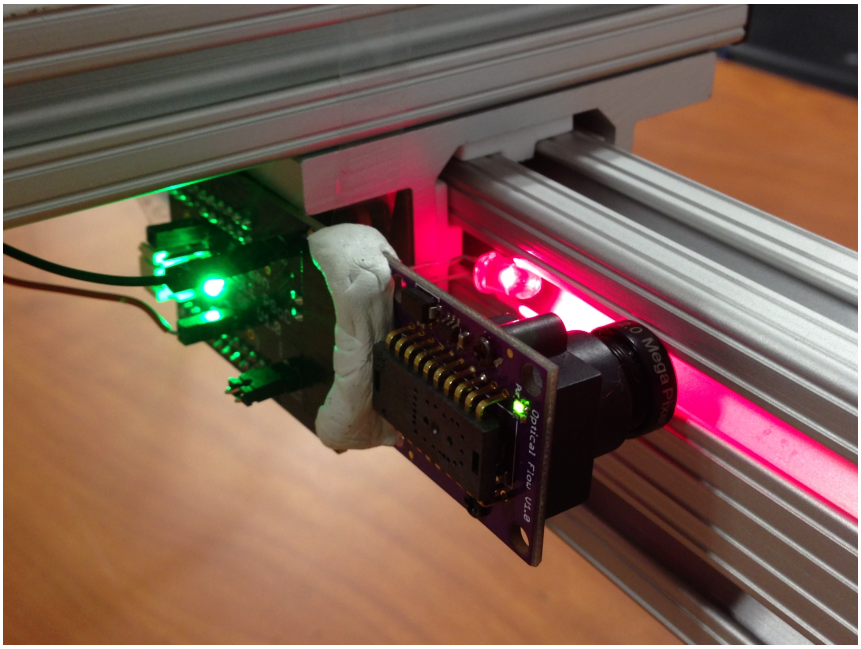


FIGURE 3.2: Mounted BLE Optical-Flow Tracking Module

3.2 BLE Optical-Flow Tracking Module

3.2.1 Optical-Flow Sensor for Z Tracking

2.2.1 Accumulated Z-axis data calibration

3.2.2 Bluetooth Low Energy for wireless communication

Chapter 4: results for 3 types of RGB-D cameras.
Chapter 5: conclusion