TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

3D reconstruction aims to reproduce the 3D profile of real objects as accurate as possible, which require accurate world coordinate $X/Y/Z$ (noted as $X^w/Y^w/Z^w$ henceforth) 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

Table 1.1: 3D profile acquisition Taxonomy

3D Shape Extraction			
Passive		Active	
Single Vantage Point	Multiple Vantage Points	Single Vantage Point	Multiple Vantage Points
• Shape from Texture • Shape from Occlusion • Time to Contact • Shape from Defocus • Shape from Contour	• Passive Stereo • Structure from Motion • Shape from Silhouettes	• Time of Flight • Shape from Shading	• Structured Light • Active Stereo • Photometric Stereo

Three dimensional (3D) profile measurement technologies have been developed by various means, as summarized by Theo Moons [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, as shown in the 3D profile acquisition taxonomy diagram [6] given in table 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 [4]. 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.
Figure 1.1 shows the an infrared speckle pattern that is projected onto a target by the infrared projector of PrimeSense 3D scanner,

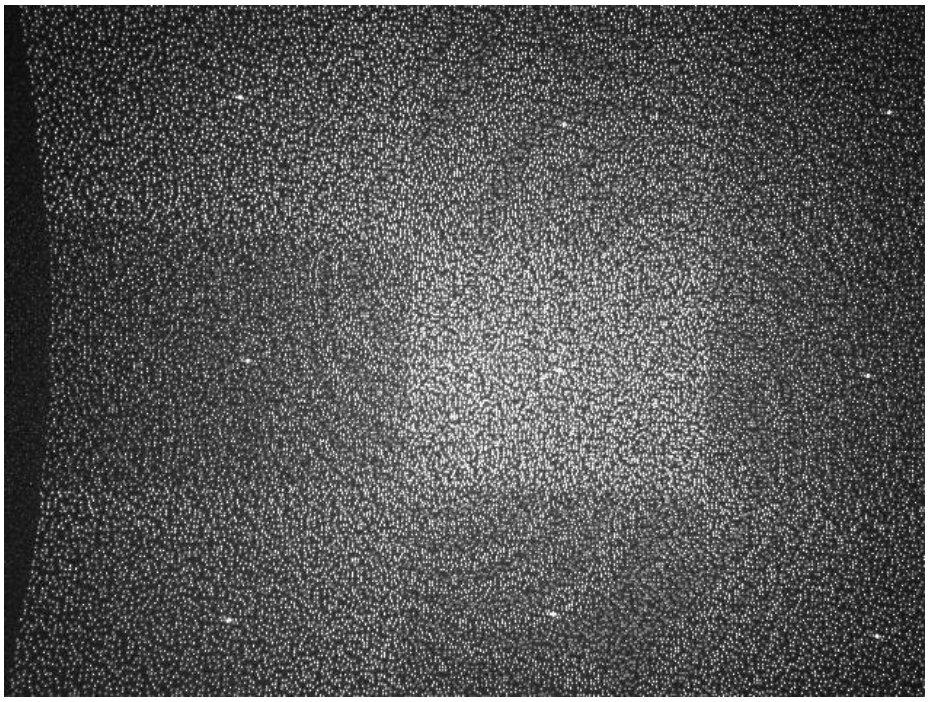


Figure 1.1: PrimeSense SL Infrared Pattern.
Source: `http://www.ros.org/wiki/kinect_calibration/technical`

and an infrared camera inside the scanner is employed 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.

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.

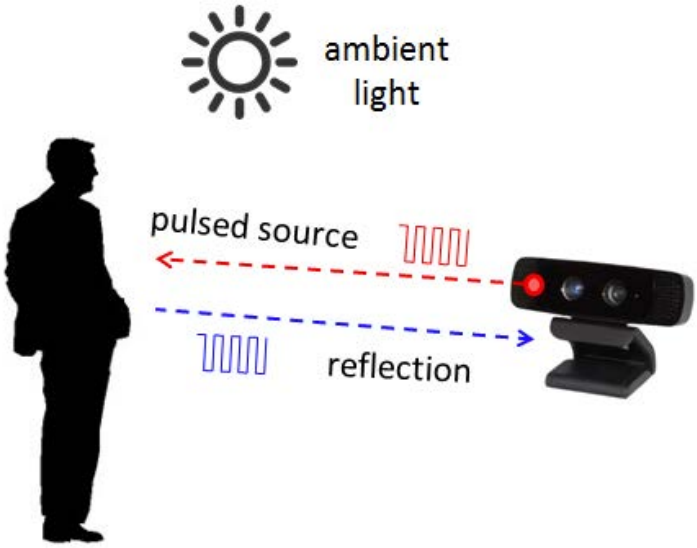


Figure 1.2: 3D time-of-flight camera operation [2]

The variable that ToF camera measures is the phase shift between the illumination and reflection, which will be translated to distance [2]. To detect the phase shifts, light source is pulsed or modulated by a continuous wave, typically a sinusoid or square wave. As figure 1.2 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 Structured Light 3D Scanner Calibration

Structured light (SL) technique is the fastest method for 3D reconstruction. By driving the projector / camera pair at very high frame rates, the object's motion then become small over the pattern set. However, at a high frame rate, the process speed of incoming video becomes an issue. Instead of recoding camera frames to memory and then applying off-line processing (like many video-based SL systems chose), Kai [4] made a good research on structured light 3D reconstruction in real-time, which brought in a pinhole-camera model based 3D scanner calibration method.

To be able to produce 3D point clouds in real-time, Kai proposed a look-up table based solution that is built on the derivation of pinhole camera model 3-by-4 transformation matrix, with a novel dual-frequency pattern. By his method, a 640 by 480 video steam can generate intermediate phase data at 1063.8 frames per second and full 3D coordinate point clouds at 228.3 frames per second, which is a satisfying speed. However, this method, being not able to remove lens distortions, is only one good lead-in 3D camera calibration method.

3D scanner calibration aims to determine the mathematical equation for every single pixel's sight, i.e., to calculate a beam equation. Concretely, with "depth" (Z^w) value given in RGB-D cameras, the beam equation calculation is to determine the linear translation coefficients from Z^w to X^w and Y^w. And how well the calibration is depends on how much distortions are removed in the beam equation.

Two kinds of distortions need to be taken care of through 3D camera calibration: perspective distortion and lens distortion. Perspective distortion is caused by the position of the camera relative to the subject, or by the position of the subject within the image frame, linear. And lens distortion is caused by optical design of lenses, non-linear. Traditionally and commonly, an ideal pinhole camera model is employed as an simple algorithm in 3D computer vision to describe a mapping from the 3D world coordinates to camera image row and column, by 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 to solve the non-linear radial and tangential lens distortion.

Totally based on the pinhole camera model (a 3-by-4 matrix), the 3D camera calibration

method from Kai's analysis can only take care of the linear perspective distortion, which not able to handle the non-linear radial dominated lens distortion. Details will soon be discussed in in chapter 2.

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, 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.

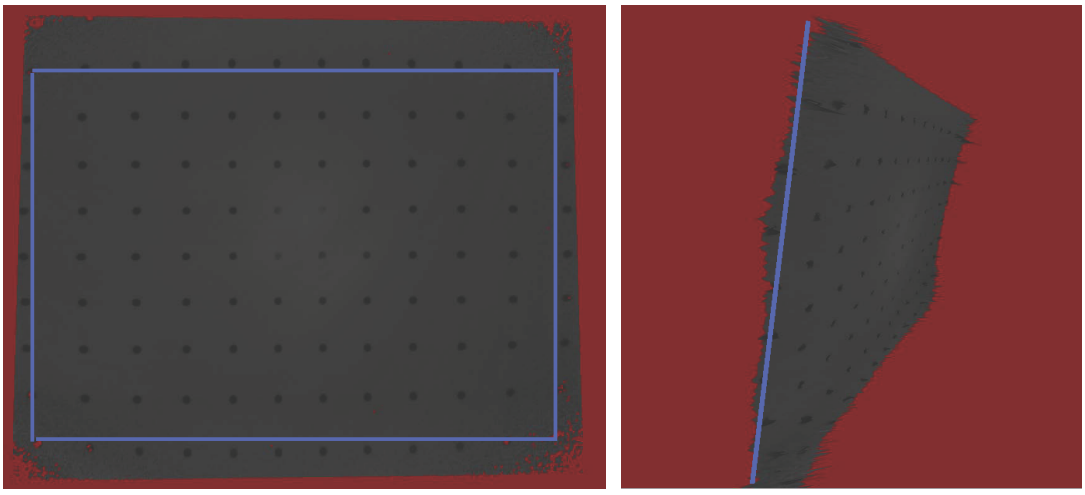
Thoroughly abandoned the pinhole camera model, the second method directly determines the beam equation for every single pixel by collecting distortions-removed X^w/Y^w and accurate Z^w values. X^w/Y^w values are calibrated by high order, concretely $4th$ order, polynomial surface mapping. And the accurate Z^w values are imported from external Z^w-aixs tracking module. In short, this is totally a data-based calibration method.

Figure 1.3: KinectV2 Calibration System

To collect enough data along Z^w-aixs, a rail is used in the calibration system. As shown in figure 1.3, 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.

1.4 Summation

RGB-D cameras' calibration cannot be easily handled by a pinhole camera model. First of all, as mentioned in section 1.2 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 [3], so such so that the $depth$ values are not able to guarantee an accurate Z^w.



(a) Front View (b) Left View

Figure 1.4: 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 1.4b, 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 From Structured Light to 3D Scanner Calibration based on Pinhole Model

Pinhole camera model is wildly used in 3D camera's reconstruction, however, it is not able to handle non-linear radial distortion. In this chapter, a pinhole model based 3D camera calibration method (derived from a 3D reconstruction example of structured light method) will be discussed in detail, and then extended into two newly proposed calibration methods.

2.1 Pinhole Camera Model

Figure 2.1 shows the basic diagram of a pinhole camera model [5] [9] with a reflected image plane for friendly intuition. 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.

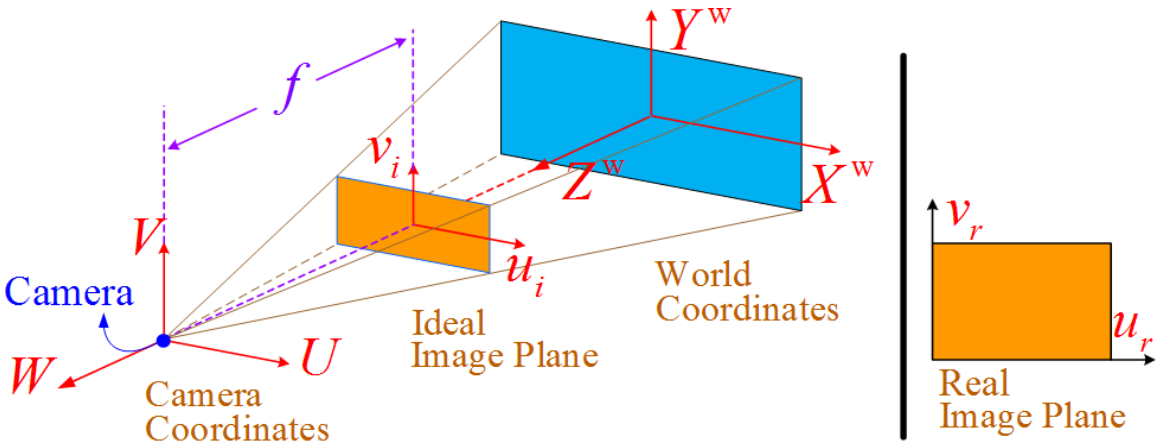


Figure 2.1: The pinhole camera model

Extrinsic Calibration

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

$$\begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} X^w \\ Y^w \\ Z^w \\ 1 \end{bmatrix} \tag{2.1}$$

or for simplicity,

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} R & T \end{bmatrix} \cdot \begin{bmatrix} X^w \\ Y^w \\ Z^w \end{bmatrix} \tag{2.2}$$

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

$$\begin{bmatrix} R & T \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{2.3}$$

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:

$$W \begin{bmatrix} u_i \\ v_i \end{bmatrix} = f \begin{bmatrix} U \\ V \end{bmatrix} \tag{2.4}$$

Homogeneous coordinates:

$$W \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} fU \\ fV \\ W \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} U \\ V \\ W \end{bmatrix} \tag{2.5}$$

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,

$$\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = \begin{bmatrix} s_u & s_\theta & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \tag{2.6}$$

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 2.5 and equation 2.6, we could get

$$W \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = \begin{bmatrix} s_u & s_\theta & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} U \\ V \\ W \end{bmatrix} \tag{2.7}$$

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 2.2 and 2.7, as

$$k \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & T \end{bmatrix} \cdot \begin{bmatrix} X^w \\ Y^w \\ Z^w \end{bmatrix} = C \cdot \begin{bmatrix} X^w \\ Y^w \\ Z^w \end{bmatrix} \tag{2.8}$$

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

$$C = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & T \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \tag{2.9}$$

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. To inspect the pinhole camera matrix, its effects focuses on the linear processes of rotation and translation, given a perspective view. In other words, this 3x4 transformation matrix is specially for perspective distortion removal. It is built on the

homogeneous coordinates, which is built on, and also limited by the linear system. And a mapping using this 3x4 transformation matrix between two coordinates can only be linear (handle linear perspective distortion).

2.2 Structured Light 3D Reconstruction in Real-Time

Using a 3D reconstruction method of structured light, applying Phase Measuring Profilometry (PMP) technique, a per-pixel 3D camera calibration method based on pinhole camera model is detailedly introduced in this section. Both of the advantage and disadvantage are discussed.

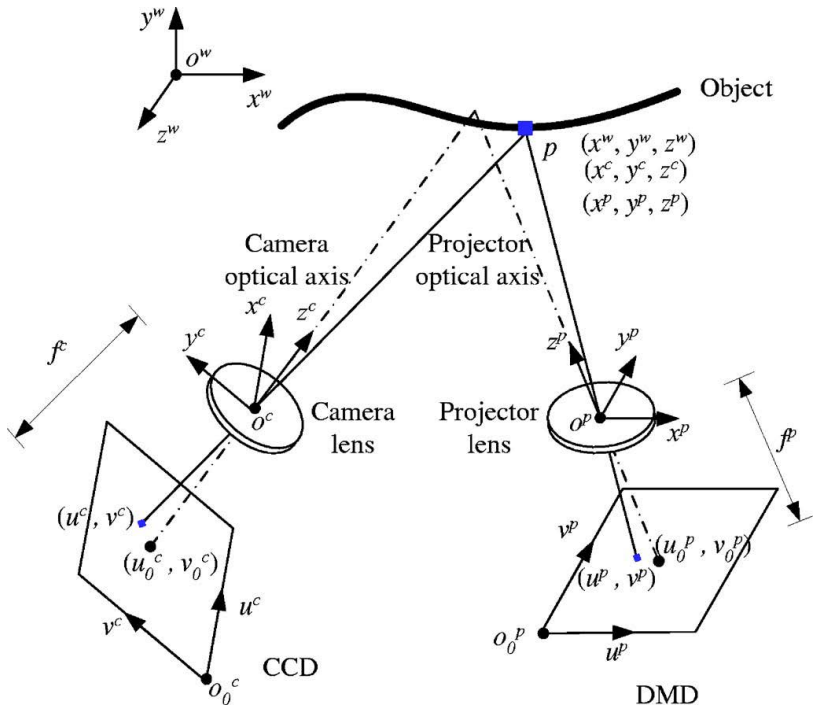


Figure 2.2: SL PMP Configuration Diagram [8]

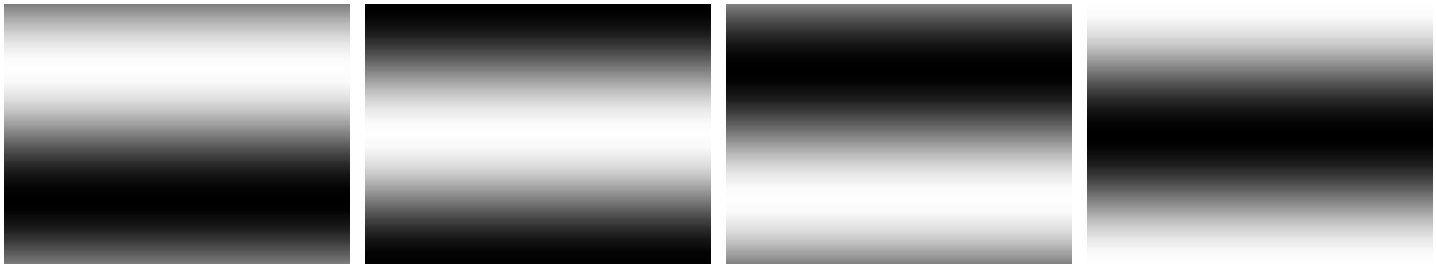
A 3D shape measurement system consists of a Charge-Coupled Device (CCD) camera and a Digital Micro-mirror Device (DMD) projector. 2D image pattern strategies are always preferred for fast scanning if a DMD projector is involved [**][8]. And the multi-shot pattern Phase Measuring Profilometry strategy 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 2.2.

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

$$I_n^p(x^p, y^p) = A^p(x^p, y^p) + B^p(x^p, y^p)Cos(2\pi f y^p - \frac{2\pi n}{N}) \tag{2.10}$$

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 2.3: PMP base frequency patterns

Figure 2.3 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

$$I_n^c(x^c, y^c) = A^c(x^c, y^c) + B^c(x^c, y^c)Cos[\phi(x^c, y^c) - \frac{2\pi n}{N}] \tag{2.11}$$

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]

$$\phi(x^c, y^c) = arctan\left[\frac{\sum_{n=1}^{N} I(x^c, y^c)Sin(2\pi n/N)}{\sum_{n=1}^{N} I(x^c, y^c)Cos(2\pi n/N)}\right] \tag{2.12}$$

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

$$y^p = \phi(x^c, y^c)/(2\pi f) \tag{2.13}$$

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 2.1 equation 2.9, are written as [7]

$$M^c = \begin{bmatrix} m^c_{11} & m^c_{12} & m^c_{13} & m^c_{14} \\ m^c_{21} & m^c_{22} & m^c_{23} & m^c_{24} \\ m^c_{31} & m^c_{32} & m^c_{33} & m^c_{34} \end{bmatrix} \tag{2.14}$$

and

$$M^p = \begin{bmatrix} m^p_{11} & m^p_{12} & m^p_{13} & m^p_{14} \\ m^p_{21} & m^p_{22} & m^p_{23} & m^p_{24} \\ m^p_{31} & m^p_{32} & m^p_{33} & m^p_{34} \end{bmatrix} \tag{2.15}$$

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

$$x^c = \frac{m^c_{11}X^w + m^c_{12}Y^w + m^c_{13}Z^w + m^c_{14}}{m^c_{31}X^w + m^c_{32}Y^w + m^c_{33}Z^w + m^c_{34}} \tag{2.16}$$

$$y^c = \frac{m^c_{21}X^w + m^c_{22}Y^w + m^c_{23}Z^w + m^c_{24}}{m^c_{31}X^w + m^c_{32}Y^w + m^c_{33}Z^w + m^c_{34}} \tag{2.17}$$

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

$$x^p = \frac{m^p_{11}X^w + m^p_{12}Y^w + m^p_{13}Z^w + m^p_{14}}{m^p_{31}X^w + m^p_{32}Y^w + m^p_{33}Z^w + m^p_{34}} \tag{2.18}$$

$$y^p = \frac{m^p_{21}X^w + m^p_{22}Y^w + m^p_{23}Z^w + m^p_{24}}{m^p_{31}X^w + m^p_{32}Y^w + m^p_{33}Z^w + m^p_{34}} \tag{2.19}$$

Since three out of four equations 2.16 ~ 2.19 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 2.16, 2.17, and 2.19

$$\begin{bmatrix} X^w \\ Y^w \\ Z^w \end{bmatrix} = \begin{bmatrix} m^c_{11} - m^c_{31}x^c, & m^c_{12} - m^c_{32}x^c, & m^c_{13} - m^c_{33}x^c \\ m^c_{21} - m^c_{31}y^c, & m^c_{22} - m^c_{32}y^c, & m^c_{23} - m^c_{33}y^c \\ m^p_{21} - m^p_{31}y^p, & m^p_{22} - m^p_{32}y^p, & m^p_{23} - m^p_{33}y^p \end{bmatrix}^{-1} \begin{bmatrix} m^c_{34}y^c - m^c_{14} \\ m^c_{34}y^c - m^c_{24} \\ m^p_{34}y^p - m^p_{24} \end{bmatrix} \tag{2.20}$$

Traditionally, as derived above, it is the arctangent computation (equation 2.12) and the matrix inversion (equation 2.20) that prove to be the bottleneck, preventing real-time surface reconstruction. However, Kai proposed a LUT-based solution that solves the bottleneck by expanding those two equations into new forms, directly building accurate LUTs. Based on Kai's derivation [4], X^w and Y^w can be computed respectively as

$$X^w_{(col^c,row^c)} = c_{(col^c,row^c)} Z^w_{(col^c,row^c)} + d_{(col^c,row^c)} \tag{2.21}$$

and

$$Y^w_{(col^c,row^c)} = e_{(col^c,row^c)} Z^w_{(col^c,row^c)} + f_{(col^c,row^c)} \tag{2.22}$$

where

$$c_{(col^c,row^c)} = \frac{(m^c_{22}m^c_{33} - m^c_{23}m^c_{32})col^c + (m^c_{13}m^c_{32} - m^c_{12}m^c_{33})row^c + (m^c_{12}m^c_{23} - m^c_{13}m^c_{22})}{(m^c_{21}m^c_{32} - m^c_{22}m^c_{31})col^c + (m^c_{12}m^c_{31} - m^c_{11}m^c_{32})row^c + (m^c_{11}m^c_{22} - m^c_{12}m^c_{21})},$$

$$d_{(col^c,row^c)} = \frac{(m^c_{22}m^c_{34} - m^c_{24}m^c_{32})col^c + (m^c_{14}m^c_{32} - m^c_{12}m^c_{34})row^c + (m^c_{12}m^c_{24} - m^c_{14}m^c_{22})}{(m^c_{21}m^c_{32} - m^c_{22}m^c_{31})col^c + (m^c_{12}m^c_{31} - m^c_{11}m^c_{32})row^c + (m^c_{11}m^c_{22} - m^c_{12}m^c_{21})},$$

$$e_{(col^c,row^c)} = \frac{(m^c_{23}m^c_{31} - m^c_{21}m^c_{33})col^c + (m^c_{11}m^c_{33} - m^c_{13}m^c_{31})row^c + (m^c_{13}m^c_{21} - m^c_{11}m^c_{23})}{(m^c_{21}m^c_{32} - m^c_{22}m^c_{31})col^c + (m^c_{12}m^c_{31} - m^c_{11}m^c_{32})row^c + (m^c_{11}m^c_{22} - m^c_{12}m^c_{21})},$$

$$f_{(col^c,row^c)} = \frac{(m^c_{23}m^c_{32} - m^c_{22}m^c_{33})col^c + (m^c_{12}m^c_{33} - m^c_{13}m^c_{32})row^c + (m^c_{13}m^c_{22} - m^c_{12}m^c_{23})}{(m^c_{21}m^c_{32} - m^c_{22}m^c_{31})col^c + (m^c_{12}m^c_{31} - m^c_{11}m^c_{32})row^c + (m^c_{11}m^c_{22} - m^c_{12}m^c_{21})}$$

and (col^c, row^c) denote the address of a camera's pixel, $c/d/e/f$ are the corresponding linear coefficients that help map from Z^w to X^w and Y^w.

After Kai's expansion from equation equation 2.20 to equation 2.21 and 2.22, the parameters of the projector's pinhole camera matrix are gone, and everything left belongs to the camera (with superscript of c). It means that, for arbitrary RGB-D 3D camera, its calibration could be done by acquiring its 3-by-4 pinhole camera matrix, and then generating beam equations of 2.20 and 2.21 for every single pixel.

2.3 Shortcoming and Extension

Shortcoming

The camera calibration method discussed in section 2.2 is totally derived from the pinhole camera model, which gives the relationship from Z^w to X^w and Y^w, and the 2D mapping from u_r/v_r ($column/row$) to X^w/Y^w, as figure 2.1 shows. And equation 2.9 and 2.8 tell us that, the pinhole matrix can only handle linear transformation, i.e., the calibration exclu-

sively based on the pinhole camera model can only handle perspective distortions, whereas the most important radial dominated non-linear lens distortions still exist inside X^w and Y^w.

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.

Figure 1.4a 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 $r = \sqrt{x^2 + y^2}$:

$$\begin{aligned} x_{distorted} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + [p_1(r^2 + 2x^2) + 2p_2 xy] \\ y_{distorted} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + [p_2(r^2 + 2y^2) + 2p_1 xy] \end{aligned} \tag{2.23}$$

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.Equation 2.23 tells us that, high order polynomial mapping is needed in order to remove the lens distortions. However, the 3-by-4 camera matrix is fixed by order 1*st* (linear) that, it is impossible to make X^w/Y^w be mapped from *column* / *row* by a higher order polynomial transformation. We have to relinquish the pinhole model's benefit of 2D mapping from *column* / *row* to X^w/Y^w, and have to relinquish equation 2.20, 2.21 and 2.22.

Extended Method One

Although the 3-by-4 pinhole camera matrix is not helpful for removing lens distortion, the pinhole camera model is still a good model that can help determining the relationship from Z^w to X^w and Y^w, and that is exactly what a 3D camera calibration needs for determining the beam equation for every single pixel.

Figure 2.4 shows the lens distortion removed pinhole camera model. In this model, the coordinates-pairs (*column* / *row* and X^w/Y^w), which was used to train the 3-by-4 pinhole matrix, are now used to train a high order polynomial transformation matrix. The orange pillow shape shows the lens distortion removed image, which was a rectangle in blue.

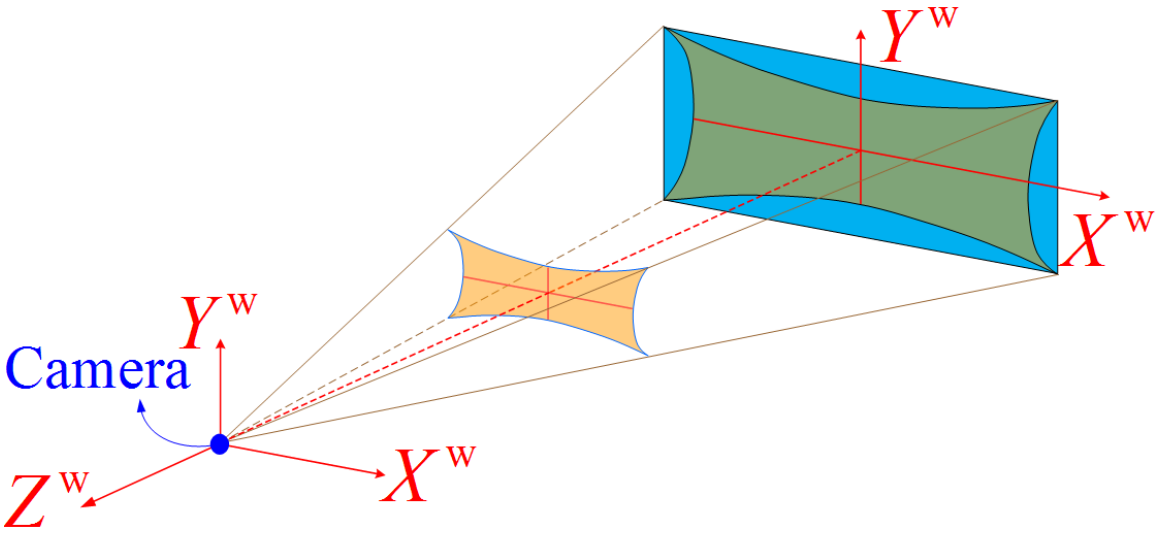


Figure 2.4: Lens Distortion Removed Pinhole Camera model

With X^w/Y^w's lens distortions removed, the last thing to go for is to calculate beam equations for every single pixel. It is straight forward to determine a line equation (beam equation) given two know points. The first point is the origin, and the second point could be calculated using the high order transformation matrix.

Extended Method Two

The extended method one, which is introduced above, is still not ideal. The focal point and focal length of a camera practically will change as its lens changes, so that light rays should not converge at the origin, and probably not even at one point. In order to get more accurate LUT for a better 3D view, the second point of the beam for every single pixel should also be a datum retrieved from the camera.

The second method, chasing after accuracy, is to add a rail along Z^w-axis for multiple points data acquisition. Not only real points (in contrast to assuming the origin as the second point) will be used for beam equation determination, but the Z^w values could also be calibrated with external accurate data support in case *depth* distortion (as shown in figure 1.4b).

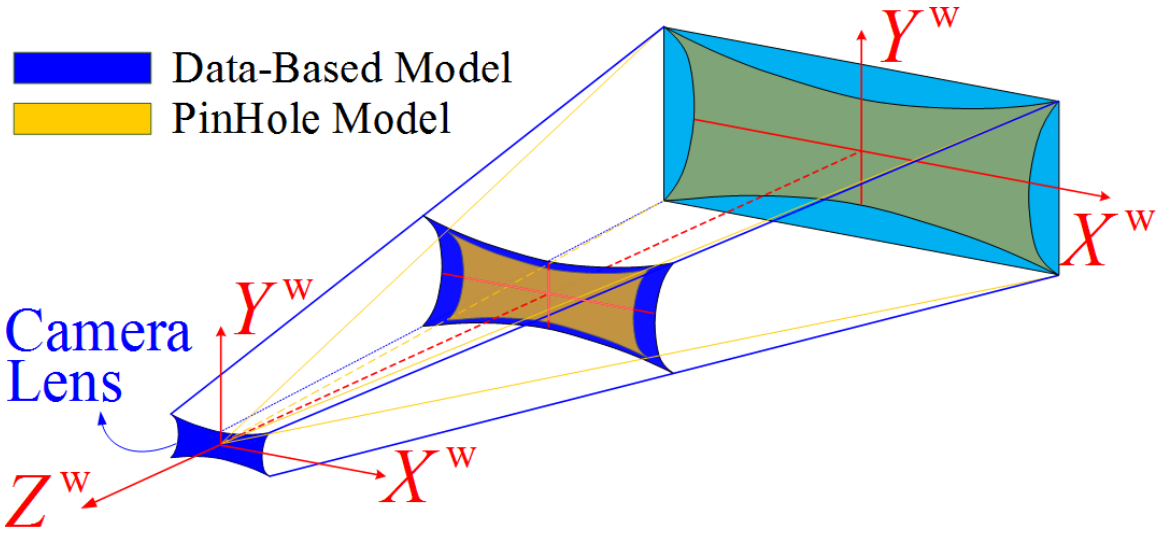


Figure 2.5: Comparison: Data-Based Calibration and Pinhole Model Calibration

Figure 2.5 gives the data-based model in dark blue, in which the light rays will converge in front of the origin (the lens prolongs the focal length from in front of the origin to the origin) with also a pillow shape. This model is proved practically on KinectV2 in section 3.2 figure 3.10.

Chapter 3 Data-Based Real-Time 3D Calibration

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.

3.1 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 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.

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

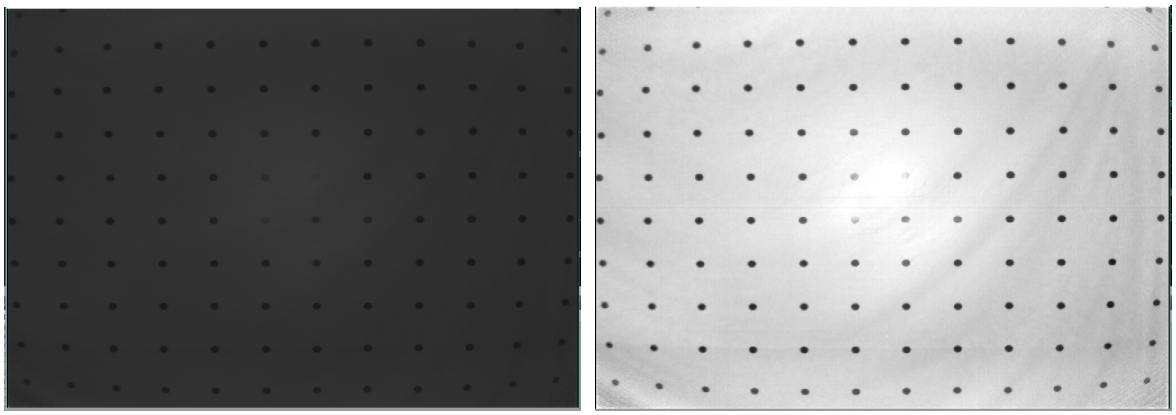
$$Intensity_{\text{\_gray}} = 0.21Red + 0.72Green + 0.07Blue \tag{3.1}$$

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 3.1a (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)



(a) Raw (b) Histogram Equalized

Figure 3.1: NearIR Streams before / after Histogram Equalization

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

$$Intensity = n/N * (1-0) + 0 = n/N \tag{3.2}$$

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

$$CDF(n) = \frac{sum}{N_{\text{\_Total Pixels/Image}}} \tag{3.3}$$

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

$$\begin{aligned} floor &= n_{\text{\_floor}}/N \\ ceiling &= n_{\text{\_ceiling}}/N \end{aligned} \tag{3.4}$$

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

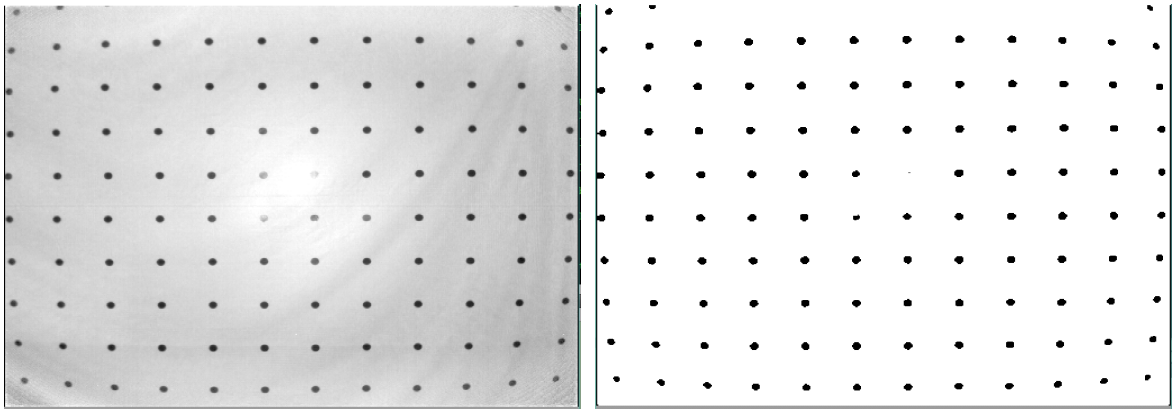
$$Intensity_{\text{\_new}} = \frac{Intensity_{\text{\_original}} - floor}{ceiling - floor} \tag{3.5}$$

After this final rescaling step of Histogram Equalization, the new image gets better contrast effect, as shown in figure 3.2a

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 3.2: 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

$$Intensity_{\text{\_binarized}} = \begin{cases} 1, & I_{\text{\_textured}} - I_{\text{\_background}} - C_{\text{\_offset}} > 0 \\ 0, & else \end{cases} \tag{3.6}$$

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 3.2, 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.

$$I_{\text{\_Path Marked}} = I_{\text{\_Original}} * \frac{(128I_{\text{A}} + 64I_{\text{B}} + 32I_{\text{C}} + 16I_{\text{D}} + 8I_{\text{E}} + 4I_{\text{F}} + 2I_{\text{G}} + I_{\text{H}})}{255} \tag{3.7}$$

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.

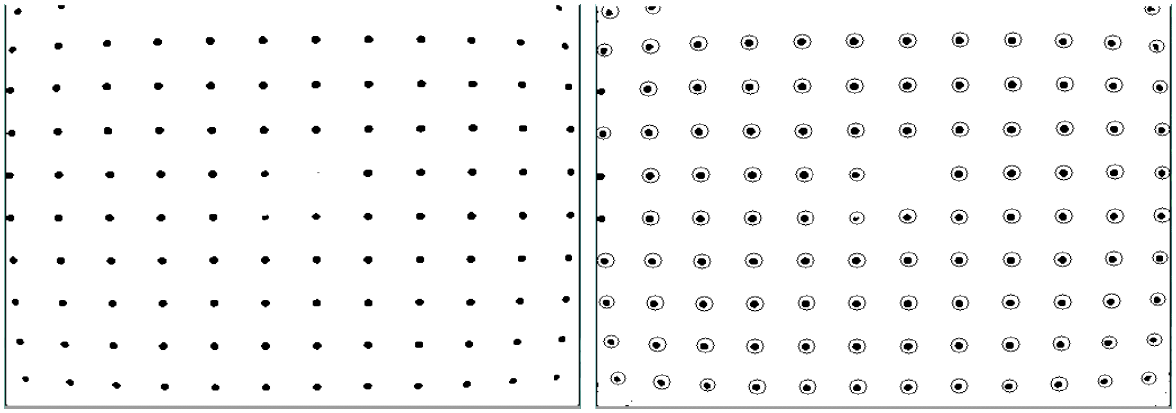
As for the sniffing step, base on the distribution table of *A*~*H* that has been discussed above and their corresponding weight given by equation 3.7, the markers *A*/*B*/*C*/*D* are valid (non-zero) as long as the intensity value of pixel *O* satisfies the following conditions shown as below.

$$\begin{aligned} &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}80 == 1), \quad then, \quad \text{marker } A \text{ is valid} \quad (\text{ go Up }) \\ &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}40 == 1), \quad then, \quad \text{marker } B \text{ is valid} \quad (\text{ go Left}) \\ &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}20 == 1), \quad then, \quad \text{marker } C \text{ is valid} \quad (\text{ go Right }) \\ &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}10 == 1), \quad then, \quad \text{marker } D \text{ is valid} \quad (\text{ go Down }) \end{aligned} \tag{3.8}$$

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.

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 3.3b, the centers of



(a) before Extraction (b) Extracted and Marked

Figure 3.3: Valid Dot-Clusters Extracted in NearIR

valid dot-clusters are marked within their corresponding homocentric circles.

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

The list of round dot centers $(column, row)$s is extracted through section 3.1. The following is to map every dot center's $(column, row)$ to its corresponding world coordinates (X^w, Y^w). As shown in figure 1.3, 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

$$\begin{bmatrix} zX^w \\ zY^w \\ z \end{bmatrix} = A \cdot \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} \tag{3.9}$$

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

$$A_{\text{\_refined}} = T \cdot A = \begin{bmatrix} 1 & 0 & -X_{\text{\_Zero\_A}} \\ 0 & 1 & -Y_{\text{\_Zero\_A}} \\ 0 & 0 & 1 \end{bmatrix} \cdot A \tag{3.10}$$

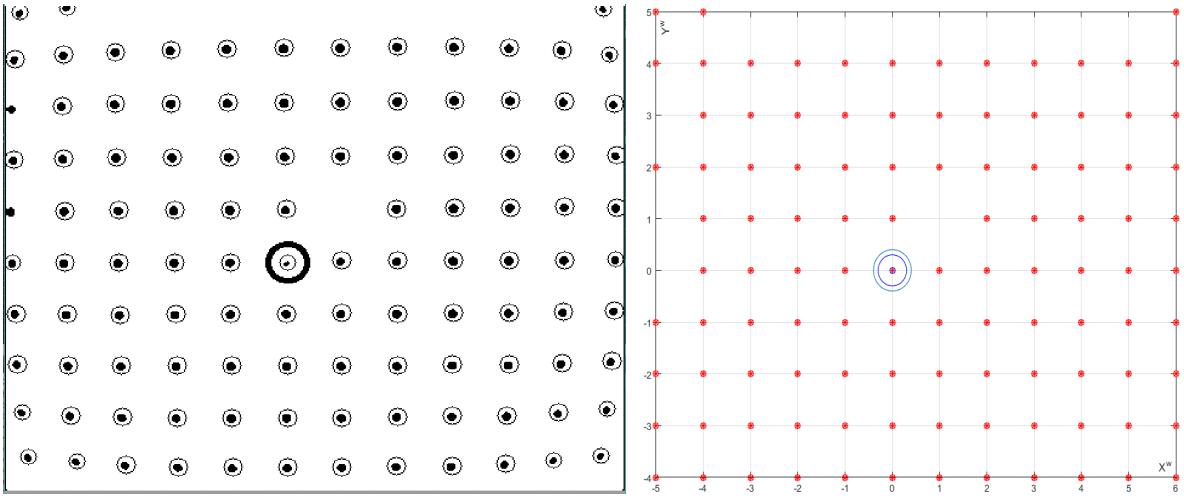
where the integer world coordinate point $(X_{\text{\_Zero\_A}}, Y_{\text{\_Zero\_A}})$ are mapped from the center point $(C_{\text{\_center}}, R_{\text{\_center}})$ of FoV in image plane by the so-called "best" transformation matrix A, written as

$$\begin{bmatrix} zX_{\text{\_Zero\_A}} \\ zY_{\text{\_Zero\_A}} \\ z \end{bmatrix} = A \cdot \begin{bmatrix} C_{\text{\_center}} \\ R_{\text{\_center}} \\ 1 \end{bmatrix} \tag{3.11}$$

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 3.4b, world coordinates are integers and the origin (blue circle) is chosen as where the center-closest dot-cluster is sitting.

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-



(a) ImagePlane Coordinate (b) World Coordinate

Figure 3.4: Coordinates-Pairs: (Row, Column)s and (X^w, Y^w)s

pairs, a perspective distortion correction (1^{st} order polynomial with z bound) could be applied using equation 3.9. As shown in figure 3.6b, the 1^{st} order polynomial correction is not able to handle non-linear distorted mapping. Based on the lens distortion equation 2.23, 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

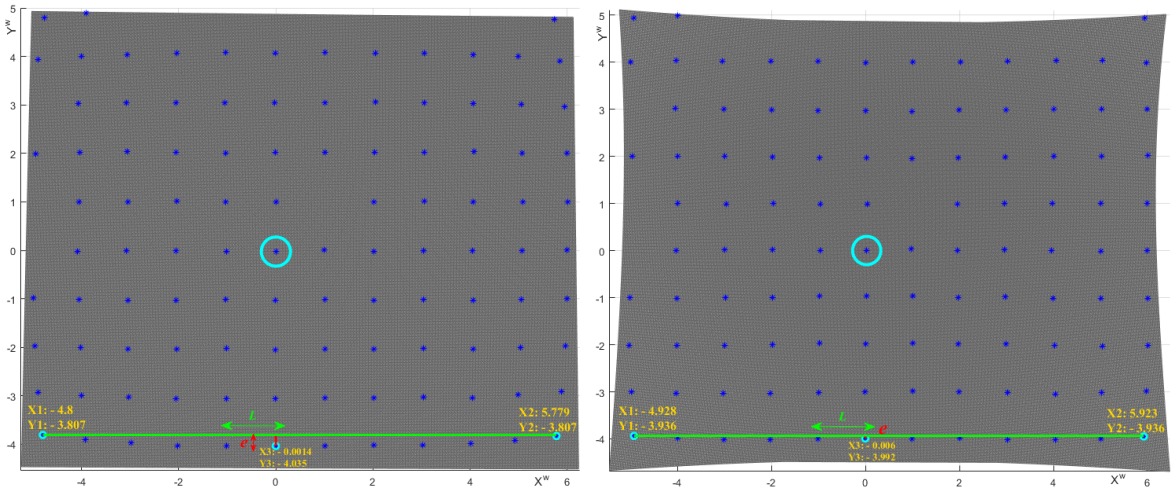
$$\begin{aligned} X^w &= a_{11}C^2 + a_{12}CR + a_{13}R^2 + a_{14}C + a_{15}R + a_{16} \\ Y^w &= a_{21}C^2 + a_{22}CR + a_{23}R^2 + a_{24}C + a_{25}R + a_{26} \end{aligned} \tag{3.12}$$

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

$$\begin{aligned} X^w = {} & a_{11}C^4 + a_{12}C^3R + a_{13}C^2R^2 + a_{14}CR^3 + a_{15}R^4 + a_{16}C^3 + a_{17}C^2R \\ & + a_{18}CR^2 + a_{19}R^3 + a_{110}C^2 + a_{111}CR + a_{112}R^2 + a_{113}C + a_{114}R + a_{115} \\ Y^w = {} & a_{21}C^4 + a_{22}C^3R + a_{23}C^2R^2 + a_{24}CR^3 + a_{25}R^4 + a_{26}C^3 + a_{27}C^2R \\ & + a_{28}CR^2 + a_{29}R^3 + a_{210}C^2 + a_{211}CR + a_{212}R^2 + a_{213}C + a_{214}R + a_{215} \end{aligned} \tag{3.13}$$

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

To prototype equation 3.12 and 3.13 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 3.5,



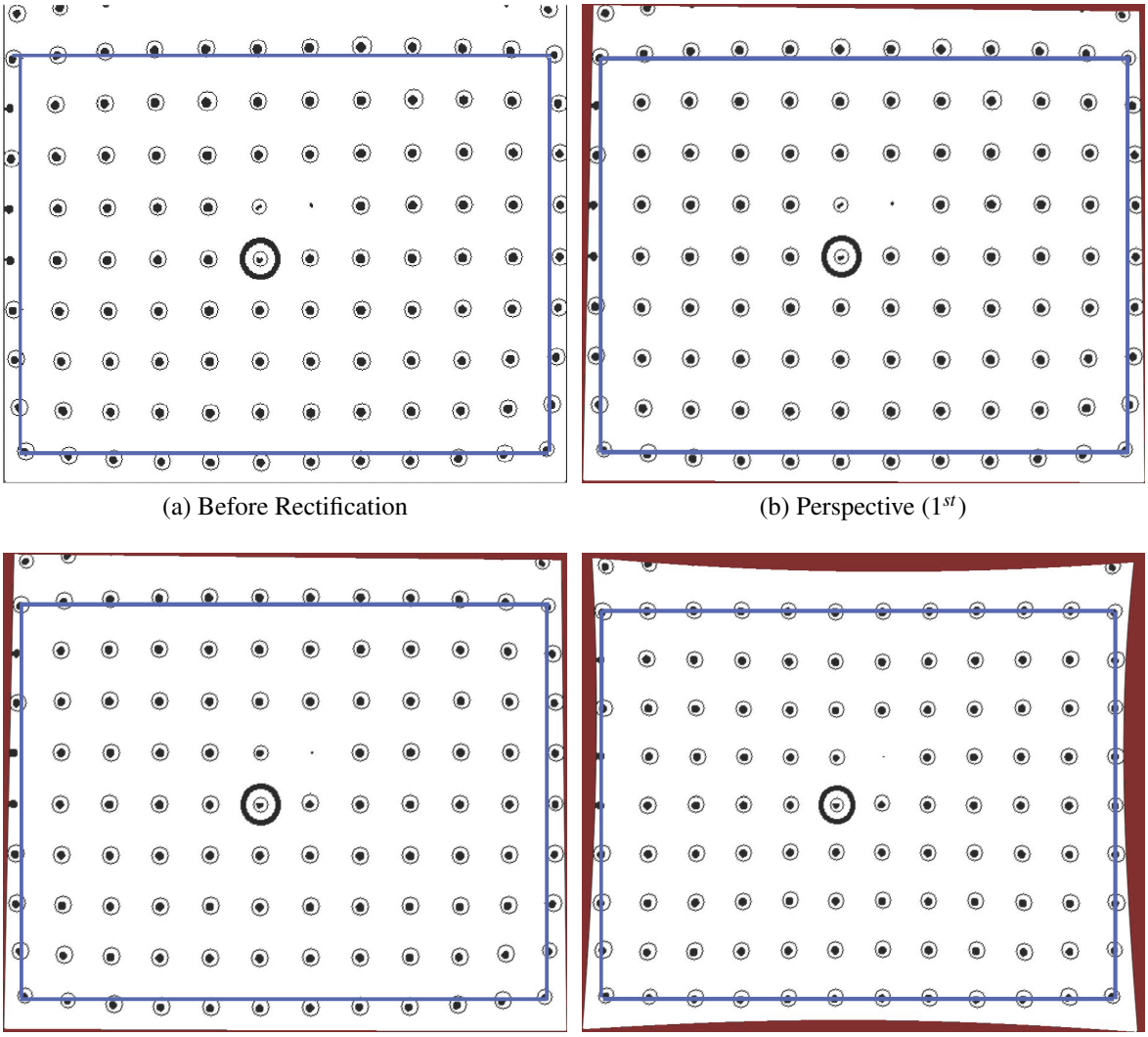
(a) 2^{nd} Order (b) 4^{th} Order

Figure 3.5: 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 3.6c and figure 3.6d), whose image-outline are same with Matlab prototypes (figure 3.5a and figure 3.5b). Comparing among the four results shown in figure 3.6, the more dot-clusters sitting on the blue rectangle, the less distortion an image has.

Not only from the comparison of RMSE, but more intuitively from figure 3.6, 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



(a) Before Rectification

(b) Perspective (1^{st})

(c) 2^{nd} Order

(d) 4^{th} Order

Figure 3.6: NearIR Stream High Order Polynomial Rectification

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.

3.2 Z^w Calibration

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 4.2, 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 3.1, 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 3.7 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.

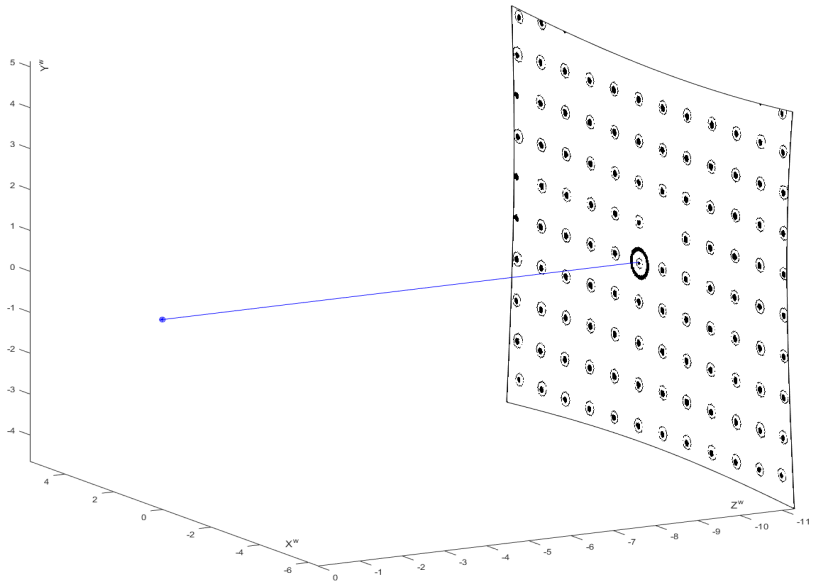


Figure 3.7: NearIR Rectified $X^w/Y^w/Z^w$ 3D Re-construction

3.3 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.

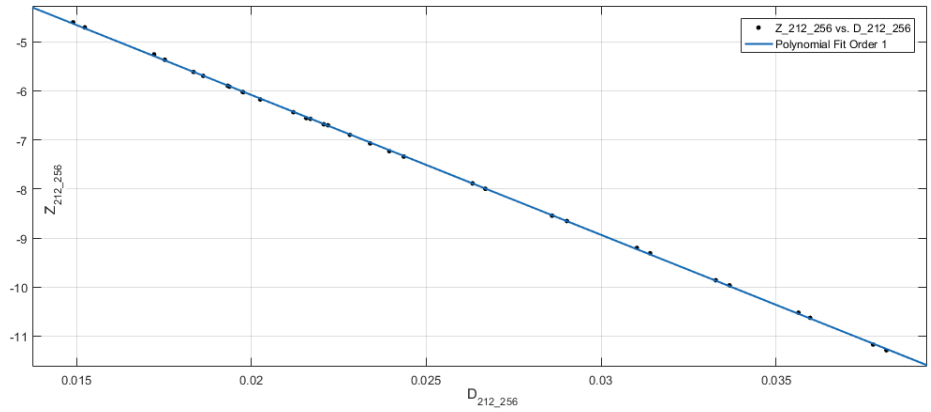


Figure 3.8: Polynomial Fitting between D and Z

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 3.8 shows the polynomial fitting result, 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

$$Z(col, row) = a_{(col,row)}D(col, row) + b_{(col,row)} \tag{3.14}$$

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.

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 3.1). 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 3.9 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

$$\begin{aligned} X(col, row) &= c_{(col,row)}Z(col, row) + d_{(col,row)} \\ Y(col, row) &= e_{(col,row)}Z(col, row) + f_{(col,row)} \end{aligned} \tag{3.15}$$

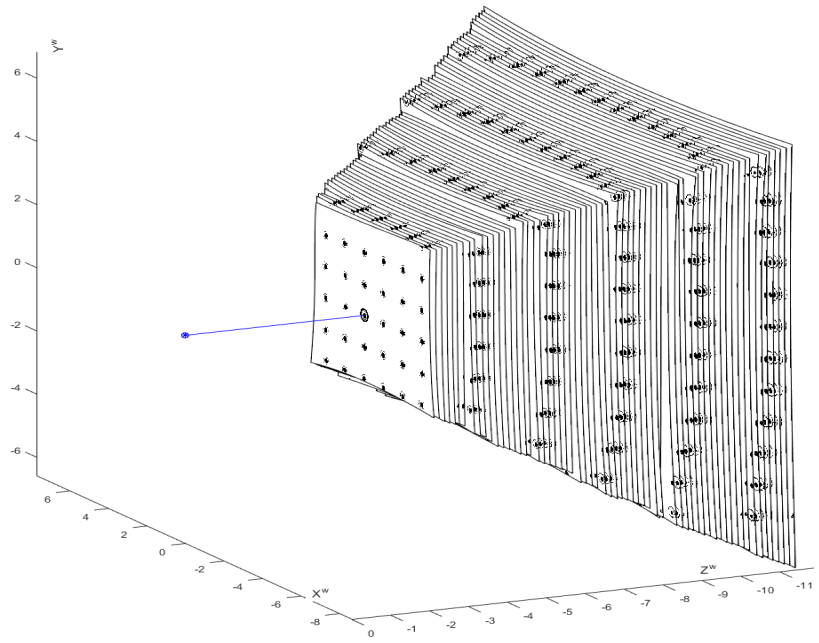


Figure 3.9: 63 Frames NearIR Rectified 3D Reconstruction

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 3.10 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.

To combine equation 3.14 and 3.15, 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.

3.4 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.

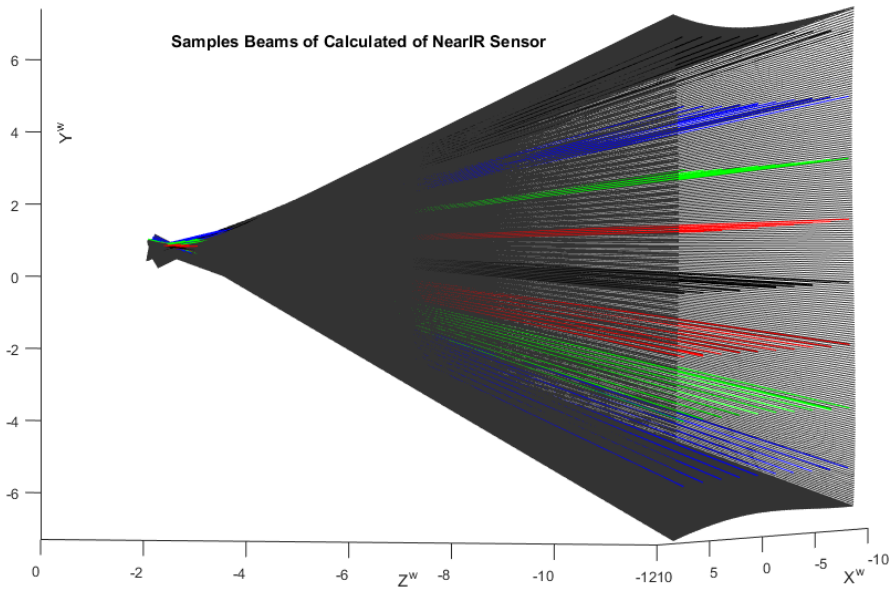


Figure 3.10: Sample Beams of Rectified NearIR Field of View

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 4.2, 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 3.9.

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).

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 3.14 and equation 3.15.

Chapter 4 Calibration System for RGBD Cameras

4.1 Rail System

As already shown in chapter 1 figure 1.3, 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 4.1 shows.

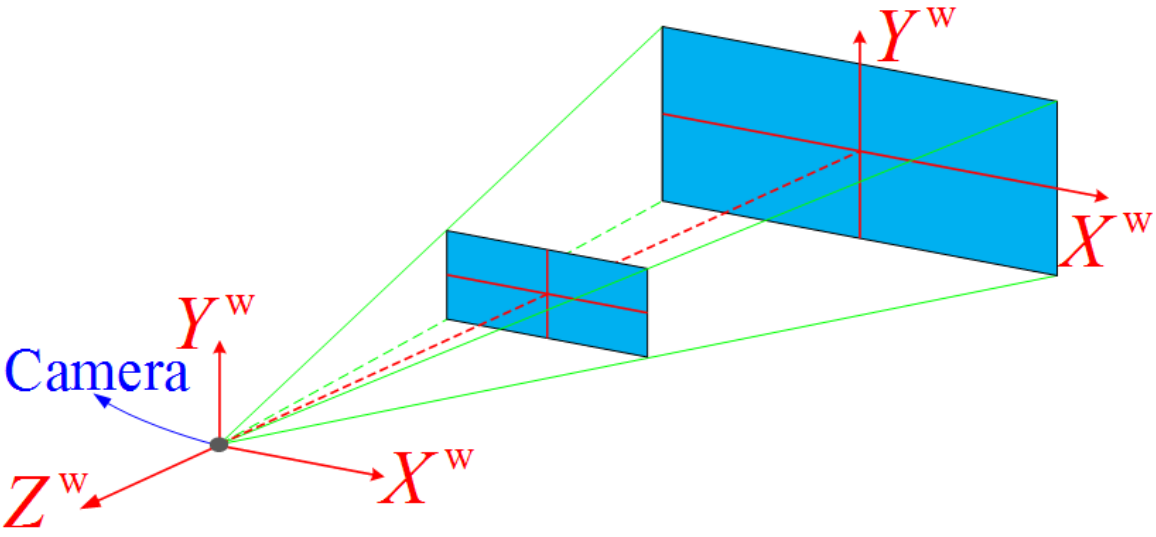


Figure 4.1: World Coordinate Frame

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 4.2. 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.

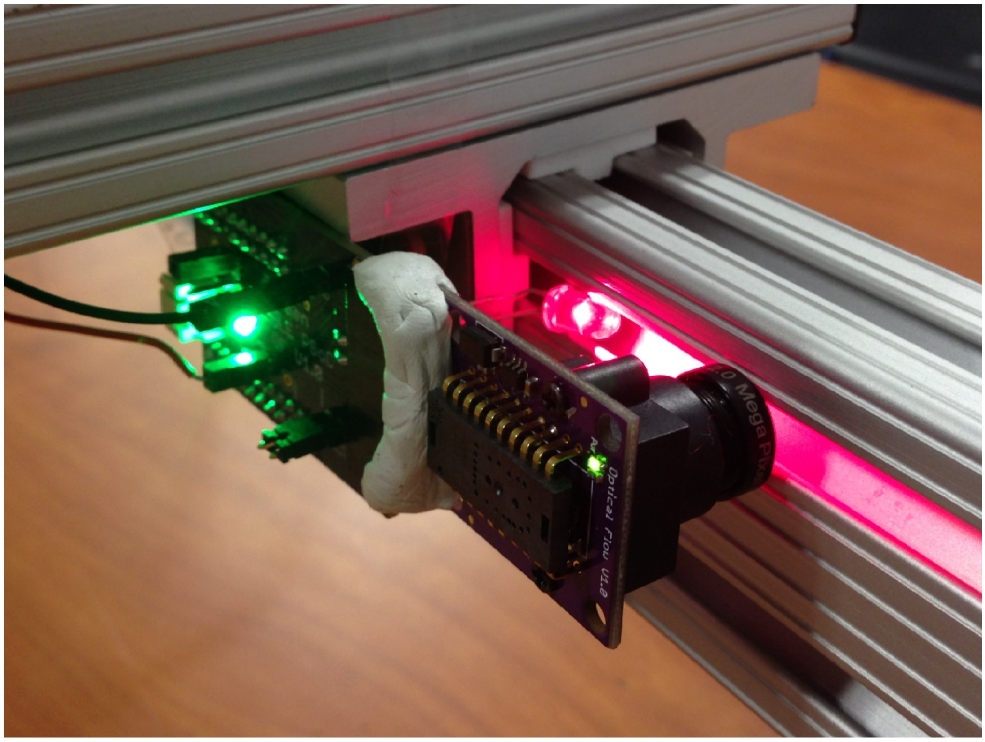


Figure 4.2: Mounted BLE Optical-Flow Tracking Module

4.2 BLE Optical-Flow Tracking Module

The BLE Optical-Flow tracking module consists of an Optical-Flow sensor for movement detection, a LED for illumination, a PSoC BLE module for wireless data communication, and a self-designed PCB as a joint.

Optical-Flow Sensor for Z Tracking

"Optical-Flow" depicts the motion of brightness patterns. In daily life, optical flow is continually processed in our visual system, offering the estimations informations of self-motion, relative depths, object speed, etc. Whereas in computer vision, optical flow is digitally saved as the motion vector for every voxel position, telling about the relative distances of objects in a given sequence of images.

For the rail system tracking on the slider along Z-axis, using optical flow for motion detection is one the best choice in the non-contact motion tracking methods.

Optical-Flow Motion Determination

The optical flow methods calculate the motion between every two image frames which are taken at times t and $t+\Delta t$ at every voxel position. Even though there are various algorithms using optical flow to determine motion, they are all based on the Brightness Constancy Constraint, which in $2D+t$ dimensional camera case can be written as

$$I(x, y, t) = I(x+\Delta x, y+\Delta y, t+\Delta t) \tag{4.1}$$

where (x, y, t) denotes the location of a voxel, $I(x, y, t)$ is its corresponding intensity; Δx and Δy are the changes between two frames taken at time t and time $t+\Delta t$.

An optical flow sensor is a vision sensor capable of measuring optical flow or visual motion and outputting a measurement based on optical flow. In this project, we use a optical mouse sensor, whose vision chip is integrated circuit that have both an image sensor for vision and a processor running one programmed optical flow algorithm in one compact implementation. As shown in figure 4.3, optical mouse sensor ADNS-3080 is used practically.

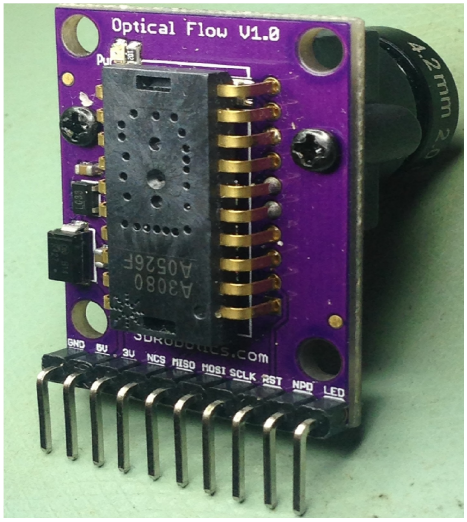


Figure 4.3: Optical Mouse Sensor

Accumulated Z^w-axis data calibration

Sensor ANDS-3080 has a programmable frame rate of over 6400 fps. With a lens mounted, this optical flow sensor can theoretically adjust its working distance from zero to one meter. Both of the raw 30-by-30 image frame and the calculated motion results could be retrieved from the optical flow sensor in two different working modes. In the “image burst” mode,

the raw 30-by-30 image frames form a 10 fps live video, helping adjust the lens for a better focus. Whereas in the "motion burst" mode, the calculated motion results offers accumulation data for both of the sensor's x-axis and y-axis.

Concretely as shown in figure 4.2, only the y-axis accumulated data is used as the accumulated Z^w-axis data in the world coordinate, whereas the x-axis accumulation is always zero (no motion on the x-axis. Employing the "motion burst" mode, on ever update, the world coordinate Z^w could be written as

$$Z^w = Z_0^w + ratio * y^{acc} = Z_0^w + ratio * (y_{\text{\_all\_previous}} + y_{\text{\_new\_updated}}) \tag{4.2}$$

where Z_0^w denotes the beginning point of accumulating (concretely at the closest end of the rail to dots pattern), $ratio$ maps the "unit one" from y-axis to the Z^w-axis, y^{acc} denotes the accumulated y-axis movement by data retrieved from the optical flow sensor, which equals to the sum of all previous values and the new updated value.

Bluetooth Low Energy for wireless communication

ZigBee, Bluetooth, and Bluetooth Low Energy (BLE) are three most commonly used Personal Area Network (PAN) wireless standards to choose from when wireless communication is needed. ZigBee runs on a mesh topology network (star and point-to-point are the other two basic topologies), which means that the information travels on a web of multiple nodes. Whereas Bluetooth and BLE are famous as point-to-point networking standard, which suits better for this project, from Optical-Flow sensor to PC.

There are huge differences between BLE and "classic" Bluetooth; despite falling under the same name, they are entirely different technologies. Bluetooth consumes more power and transmits farther and with more data. It is suited for streaming media such as playing music on Bluetooth speakers or taking a call through a Bluetooth headset.

Figure 4.4: Cypress PSoC BLE Module

Whereas BLE transmits less data over shorter distances using much less power than Bluetooth. Both of the transmission frequency and the amount of data to transfer per time are controllable by developer for BLE communication.

As for the wireless data transmission in this project, with the *x*/*y* accumulation data being to transfer, the amount of data to transfer per time is only two bytes. Moreover, the maximum frame rate of optical flow sensor ANDS-3080 is 6400 fps, which means the transmission frequency cannot be faster than 6400 Hz (we do not need that fast in the meantime). In short, BLE is finally chosen as the wireless networking to transfer data from the optical flow sensor to PC. Concretely, the Cypress PSoC BLE Module is used, with a 10KB maximum throughput that is enough for the demand in this project, as shown in figure 4.4.

BLE Programming based on Protocol

BLE is a low-power, short-range, low-data-rate wireless communication protocol. Unlike Bluetooth application that only the high level connection needs to be concerned, the BLE application developing needs low level programming, more complex but offering more control space for developer. Figure 4.5 shows the BLE Protocol Stack that BLE developers need to refer to.

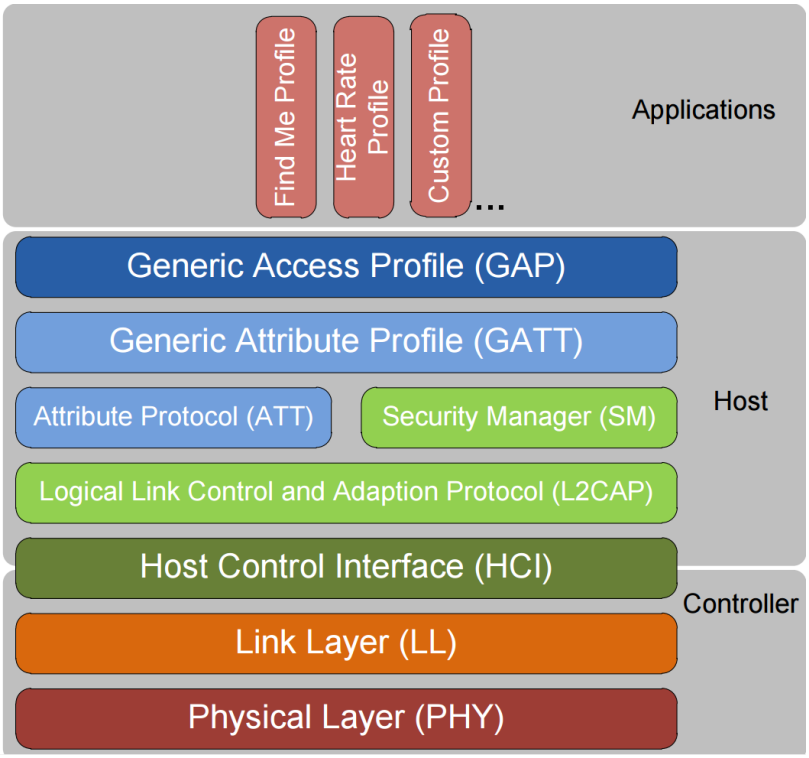


Figure 4.5: BLE Protocol Stack

Cypress BLE Module helps handle the lowest hardware controller section (PHY, LL, HCI) and part of the firmware host section (L2CAP, ATT, SM), so that developers only need to take care of the GAP and GATT layers. GAP provides an application-oriented interface that determines whether the device acts as a BLE link "Central" or "Peripheral", and configures the underlying layers accordingly. Whereas GATT defines methods to access data defined by ATT layer ("Client" to receive or "Sever" to send).

Concretely for the BLE programming in this project, on the GPA layer, the BLE module continuously retrieving data from Optical-Flow sensor is the "Peripheral" that advertises to a Central, while a BLE dongle (receiver connected on PC) is the "Central" that scans for advertisements from GAP Peripherals and is going to establish the connection with the BLE module. Whereas on the GATT layer, the BLE module is the "Sever" that contains data and the dongle on PC works as a "Client" to receive data.

To program on the PC side, both of the command sending and data receiving are through the serial port which connects the BLE dongle. Both of the output command to send and received the input data are packaged in a center format, which means for every command about BLE control, there must be a method to package it and a corresponding decoding

method to extract valid data. All of the commands that need to be managed are to handle GATT data transmission and GAP connection problem. After GATT and GAP setting, concretely, the optical flow sensor works at a sampling rate of 2000Hz and the BLE communication speed is 100 updates per second, i.e., Z^w is updated at 100Hz.

PCB Joint, Power Supply and Illumination

Figure 4.6 shows the PCB combining and powering (3.3 ~ 16V flexible voltage input) for a PSoC BLE module, an OF sensor, and a LED for OF sensor illumination. Figure 4.7 shows the overview of the BLE Optical-Flow Tracking Module.

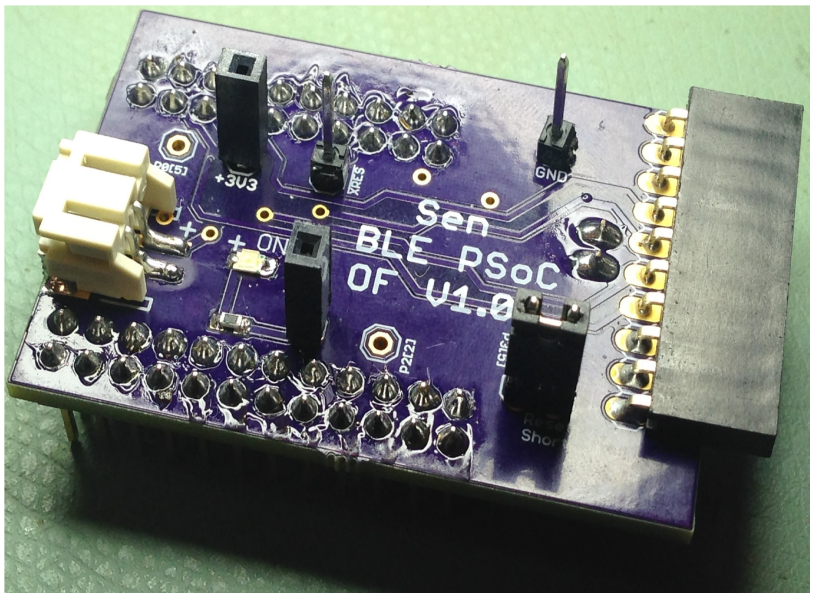
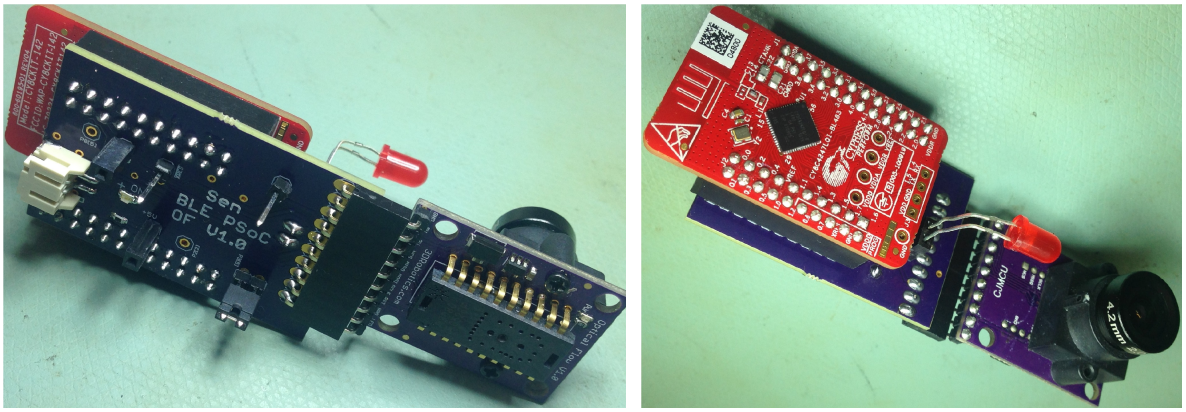


Figure 4.6: PCB: joint & power supply



(a) Front (b) Back

Figure 4.7: Overview of BLE OF Tracking Module

Chapter 5 Conclusion and Future Work

RGB-D cameras have both of lens distortions and depth distortion problem. Distortions correction is wildly discussed in image processing area. In this thesis, a data-based LUT software calibration method for general RGB-D cameras is proposed, in contrast with the pinhole-camera-model based traditional method. In this section, both of the advantages and and disadvantages are concluded, and the possible avenues of future study are discussed.

5.1 Conclusion

In traditional 3D cameras' calibration, 3D reconstruction (based on pinhole camera model) and lens distortion are separated, which depresses the efficiency of image processing. Moreover, it does not include the depth distortion correction, assuming that the depth values are accurate and employing them directly as Z^w in 3D reconstruction.

In stead of using a pinhole camera model, which in practical is not ideally accurate, our new proposed calibration method is based on real data. Not only X^w/Y^w values are rectified directly through one step transformation, but Z^w is also guaranteed accurate by importing externally measured data.

During this data-based calibration method, high-order polynomial surface mapping is employed for X^w/Y^w rectification, as discussed in section 3.1. And an IoT technology, an individual BLE OF tracking module is introduced for the support of external Z^w data, offering a way for the depth distortion correction.

In contrast with the traditional pinhole-camera-model based method, the new proposed data-based method offers more efficient rectified 3D reconstruction coordinates, with rectified Z^w after the depth distortion correction. One disadvantage is the memory cost. The data-based method brings in better accuracy at the expense of more data collection. With the help of the BLE OF tracking module, data could be collected automatically during one-way of sliding the slider on the rail from one end to the other, whereas the memories cost cannot be skipped. However, with the fast development of semiconductor technologies, memories cost should not be a problem.

5.2 Future Work

The new proposed data-based method applies universally to all of the RGB-D cameras. With a better calibration system and corresponding DIP technologies, there could be a huge improvement space for calibration accuracy.

An apparent limit during the X^w/Y^w rectifications is the static patten, which offers only uncontrollable dead distortion information. As the working distance changes, the dot-clusters, which contains distortion information, observed by a camera also changes, resulting an inconsistent resolution of rectification. What's worse, for different RGB-D cameras with different resolution, the pattern (size) needs to be changed for a better match.

In the future calibration system, in a lab with high-performance light control to reduce possible noises, the uniform pattern could be displayed by a large LCD screen. Managed thus, the uniform pattern is totally controllable, and will be part of the calibration system to form a new close-loop system. Data extracted from the camera stream supplying distortion information will be feedback for the best live pattern generating, helping adjust both size and distribution of the displayed pattern in real-time. In this way, we can control a fixed pattern distribution (offering uniformly distortion information) for cameras with various resolution at any working distance all along the rail.

The dots pattern could also be changed to uniform grid pattern (like a checker board) for more precise coordinates extraction. And corresponding DIP methods' improvement can also improve the accuracy for camera calibration.

[7]

Bibliography

[1] B. Curless and S. Seitz. 3d photography. *ACM Siggraph*, Course Notes(19):page 23, 2000.

[2] Larry Li. Time-of-Flight Camera – An Introduction. *Texas Instruments Technical White Paper*, SLOA190B, 2012.

[3] Krystof Litomisky. Consumer rgb-d cameras and their applications. university of california, riverside. 2012.

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

[5] Maria Magnusson. Short on camera geometry and camera calibration. linkping university, sweden. Report No: LiTH-ISY-R-3070, November, 2010.

[6] Theo Moons, Luc Van Gool, and Maarten Vergauwen. 3D Reconstruction from Multiple Images. *Foundations and Trends*, 4(4):287 – 404, Apr 2010.

[7] name. title. *J. Phys. Conf. Ser.*, 67:23, 2007.

[8] Song Zhang and Peisen S. Huang. Novel method for structured light system calibration. *Optical Engineering*, 8(45), Aug 2006.

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