TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

1.1 RGBD Cameras

A RGB-D camera, also called depth camera, is a sensing system that capture RGB images along with per-pixel depth information. Usually it is simply a combination of a RGB sensor a depth sensor with certain alignment algorithm between them. 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. As the depth sensor technologies opens a new epoch for three-dimensional (3D) markets, the multimedia technologies development is changing from two-dimensional (2D) to 3D on games, movies, health care aid, military training, *etc*., all various areas that are in demand of a more direct reflection on the real world [7]. Simple 2D scenes are not able to meet the social demands any more. And with the fast development of multimedia technologies, three-dimensional scene display has become a hotspot in the display field. As an extension of classic 2D video, the 3D dynamic display technology can provide a more comprehensive immersive feeling to users than a 2D video. Gesture recognition, 3D modeling, 3D printing, augmented reality, virtual reality, *etc*., a lot of ongoing researches and applications on depth cameras are famous now, cooperated with Human Computer Interaction (HCI) technologies.

The PrimeSense's technology had been originally applied to gaming, whose user interfaces are based on gesture recognition instead of using a controller (also called Natural User Interface, NUI). It was best known for licensing the hardware design and chip used in Microsoft's first generation of Kinect motion-sensing system for the Xbox 360 in 2010. The PrimeSense sensor projects an infrared speckle pattern, which will then be captured by an infrared camera in the sensor. A special microchip is employed to compare the captured speckle pattern part-by-part to reference patterns stored in the device, which were captured previously at known depths. The final per-pixel depth will be estimated based on which reference patterns the captured pattern matches best. Other than the first generation of Kinect camera, Asus Xtion PRO sensor, another consumer NUI application product, has also applied the PrimeSense's technology.

As a competitor of PrimeSense Structured Light technology, time-of-flight technology had been applied into PMD[Vision] CamCube cameras and 3DV's ZCam cameras. 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 sin-

gle point. The "time cost" variable that ToF camera measures is the phase shift between the illumination and reflection, which will be translated to distance [27]. To detect the phase shifts, light source is pulsed or modulated by a continuous wave, typically a sinusoid or square wave. The ToF camera illumination is typically from a LED or a solid-state laser operating in the near-infrared range invisible to human eyes. Fabrizio *et al*.[5] compared the time-of-flight (PMD[Vision] CamCube) camera and PrimeSense (first generation Kinect) camera in 2011. He showed that the time-of-flight technology is more accurate and claimed that the time-of-flight technology will not only be extended to support colours and higher frame sizes, but also rapidly drop in price. In 2009, 3DV agreed to sell its ZCam assets to Microsoft. And in 2013, Microsoft released the Xbox One, whose NUI sensor KinectV2 features a wide-angle ToF camera.

Unlike the PrimeSense's speckle pattern or KinectV2's ToF, Intel RealSense camera utilizes stereo vision. Its sensor actually has three cameras: two IR cameras (left and right), and one RGB camera. Additionally, RealSense camera also has an IR laser projector to help the stereo vision recognize depth at unstructured surfaces. Compared with KinectV2 camera, RealSense camera is more like a desktop usage to capture faces or even finger gestures, whereas the KinectV2 could do better to capture the full body actions with all joints. The effective distances of KinectV2 and RealSense hardwares are different. The KnectV2 is optimized to 0.5m 4.5m, while RealSense are designed for 0.2m 1.2m depends on different devices.

1.2 Human Computer Interface

Pattern recognition and gesture recognition are of the hottest sustained research activities in the area of HCI. Being a significant part in non verbal communication hand gestures are playing vital role in our daily life. Hand Gesture recognition system provides us an innovative, natural, user friendly way of interaction with the computer which is more familiar to the human beings. Gesture Recognition has a wide area of application including human machine interaction, sign language, immersive game technology *etc*. By keeping in mind the similarities of human hand shape with four fingers and one thumb, [35] presents a real time system for hand gesture recognition on the basis of detection of some meaningful shape based features like orientation, center of mass (centroid), status of fingers, thumb in terms of raised or folded fingers of hand and their respective location in image. Since gestures based on hand and finger movements can be robustly understood by computers by using a special 3D IR camera, users are allowed to play games and interact with computer applications in natural and immersive ways that improve the user experience. In 2010,

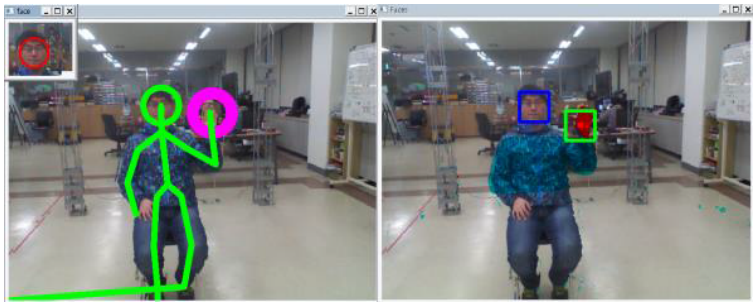


Figure 1.1: Calling Gesture Recognition Using Kinect [51]

Microsoft's first generation Kinect, using PrimeSenses technology, is based on projecting a structured light pattern on the object in order to build and track the skeleton of the users body to control a series of electronic games. [25] also discussed similar techniques for 3D object reconstruction. The appearance of RGBD cameras around revolutionized robotics applications, as the sensor is very easy to use and oers data of enough quality for most applications. By offering an alternative to time-cost procedures such as stereo-vision and laser scanning, RGB-D cameras open up the possibility of real-time robot interaction in human environments. Using this depth-map along with the 2D color images obtained via the Kinect camera, the computational unit of the Xbox 360 console was capable of providing a fairly robust gesture recognition solution. After [24] proposed two methods for gesture recognition, a plethora of applications have been produced around Kinect by various groups of image processing researchers.

A Kinect-based calling gesture recognition scenario is proposed by [51] for taking order service of an elderly care robot. Its proposed scenarios are designed mainly for helping non expert users like elderly to call service robot for their service request. In order to facilitate elderly service, natural calling gestures are designed to interact with the robot. Fig. 1.1 shows the evaluation of gesture recognition when sitting on chair. Individual people is segmented out from 3D point cloud acquired by Microsoft Kinect, skeleton is generated for each segment, face detection is applied to identify whether the segment is human or not, and specific natural calling gestures are designed based on skeleton joints. [18] proposed another smart and real-time depth camera based on a new depth generation principle. A monotonic increasing and decreasing function is used to control the frequency and duty-cycle of the NIR illumination pulses. The adjusted light pulses reflect off of the object of interest and are captured as a series of images. A reconfigurable hardware architecture calculates the depth-map of the visible face of the object in real-time from a number of images. The final depth map is then used for gesture detection, tracking and recognition. Fig. 1.2 shows an example extraction of hand skeleton. In 2013, Jaehong [26] *et al.* develop

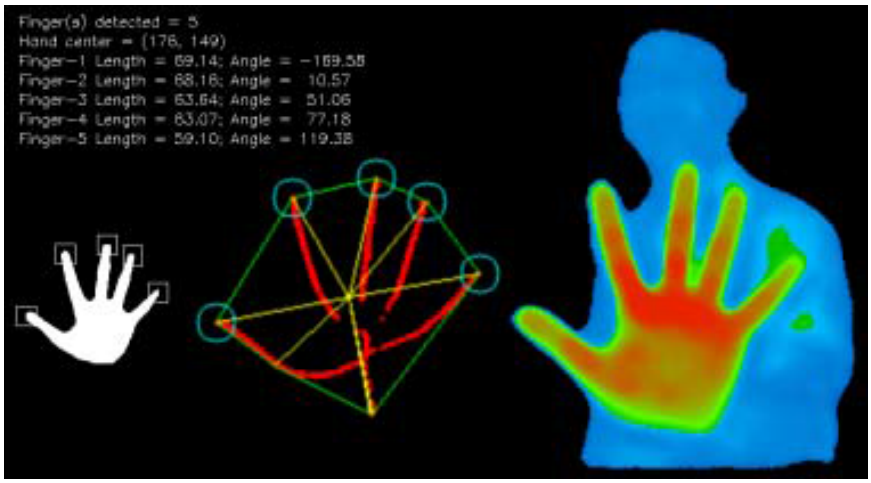


Figure 1.2: Finger Detection using Depth Data [18]

and implement a Kinect-based 3D gesture recognition system for interactive manipulation of 3D objects in educational visualization softwares.

RGB-D cameras own great credits in mobile robotics building dense 3D maps of indoor environments. Such maps have applications in robot navigation, manipulation, semantic mapping, and telepresence. [16] present a detailed RGB-D mapping system that utilizes a joint optimization algorithm combining visual features and shape-based alignment. Building on best practices in Simultaneous Localization And Mapping (SLAM) and computer graphics makes it possible to build and visualize accurate and extremely rich 3D maps with RGB-D cameras. Visual and depth information are also combined for view-based loop closure detection, followed by pose optimization to achieve globally consistent maps. SLAM is the process of generating a model of the environment around a robot or sensor, while simultaneously estimating the location of the robot or sensor relative to the environment. SLAM has been performed in many ways, which can be categorized generally by their focus on localization or environment mapping [44]. SLAM systems focused on localizing the sensor accurately, relative to the immediate environment, make use of sparse sensor data to locate the sensor. Using range sensors such as scanning laser range-finders [22], LiDAR and SONAR [10], many robot applications use SLAM systems only to compute the distance from the sensor to the environment. SLAM systems focused on mapping use dense sensor output to create a high-fidelity 3D map of the environment, while using those data to also compute relative location of the sensor [34] [46]. Many modern SLAM algorithms combine both approaches, usually by extracting sparse features from the sensor and using these for efficiently computing the location of the sensor. This position is then used to construct a map from dense sensor data.

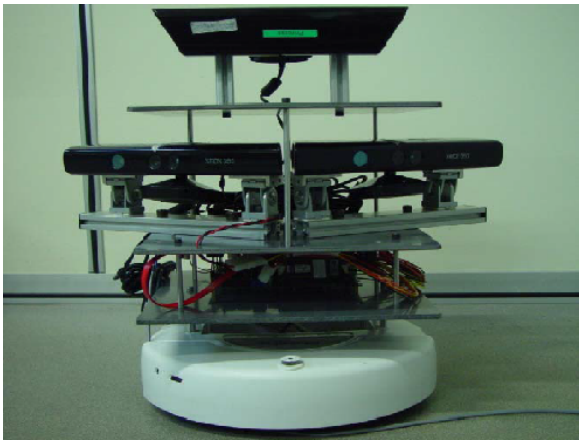


Figure 1.3: SLAM system with Only RGBD Cameras [28]

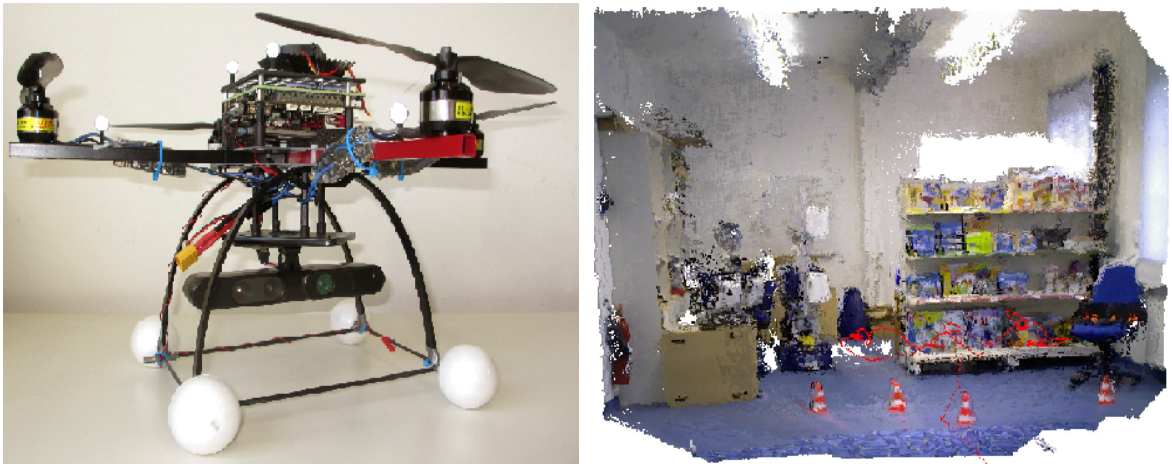
With a consumer RGB-D camera providing both color images and dense depth maps at full video frame rate, there appears a novel approach to SLAM that combines the scale information of 3D depth sensing with the strengths of visual features to create dense 3D environment representations, which is called RGB-D SLAM. [8] gives an open source approach to visual SLAM from RGB-D sensors, which extracts visual keypoints from the color images and uses the depth images to localize them in 3D. [28] builds an efficient SLAM system using three RGBD sensors. As shown in fig. 1.3, one Kinect looking up toward the ceiling can track the robots trajectory through visual odometry method, which provide more accurate motion estimation compared to wheel motion measurement without being disturbed under wheel slippage. And the other two contiguous horizontal Kinects can provide wide range scans, which ensure more robust scan matching in the RBPF-SLAM framework. Also using RGB-D sensor for SLAM, [32] presents a constraint bundle adjustment which allows to easily combine depth and visual data in cost function entirely expressed in pixel. In order to enhance the instantaneity of SLAM for indoor mobile robot, [45] proposed a RGBD SLAM method based on Kinect camera, which combined Oriented FAST and Rotated BRIEF (ORB) algorithm with Progressive Sample Consensus (PROSAC) algorithm to execute feature extracting and matching. ORB algorithm which has better property than many other feature descriptors was used for extracting feature. At the same time, ICP algorithm was adopted for coarse registration of the point clouds, and PROSAC algorithm which is superior than RANSAC in outlier removal was employed to eliminate incorrect matching. To make the result more accurate, pose-graph optimization was achieved based on General Graph Optimization (g2o) framework. Fig. 1.4 shows the 3D volumetric map of the lab, which can be directly used to navigate robots.

RGB-D camera is also famous in application of doing visual odometry on autonomous

Figure 1.4: 3D Map of RGBD-SLAM with ORB and PROSAC [45]

flight of a micro air vehicle (MAV), helping acquire 3D models of the environment and estimate the camera pose with respect to the environment model. Visual odometry generally has unbounded global drift while estimating local motion. To bound estimation error, it can be integrated with SLAM algorithms, which employ loop closing techniques to detect when a vehicle revisits a previous location. A computationally inexpensive RGBD-SLAM solution is discussed by [37] tailored to the application on autonomous MAVs, which enables our MAV to fly in an unknown environment and create a map of its surroundings completely autonomously, with all computations running on its onboard computer. Fig. 1.5a shows the MAC with an RGB-D sensor (the first generation of Kinect) mounted. And fig. 1.5b shows the reconstruction based on the full point clouds, with the estimated trajectory shown in red dots.

RGB-D sensors can be used on a much smaller scale than SLAM to create more detailed, volumetric reconstructions of objects and smaller environments, which opens a new world to the fast 3D printing. 3D printing is an additive technology in which 3D objects are created using layering techniques of different materials, such as plastic, metal, *etc*. It has been around for decades, but only recently is available and famous among the general public. The first 3D printing technology developed in the 1980s was stereolithography (SLA) [17]. This technique uses an ultraviolet (UV) curable polymer resin and an UV laser to build each layer one by one. Since then other 3D printing technologies have been introduced. Nowadays, some companies like iMaterialise or Shapeways offer 3D printing services where you can simply upload your CAD model on-line, choose a material and in a few weeks your 3D printed object will be delivered to your address. This procedure is quite straight-forward when you got your CAD model. However, 3D shape design tends to



(a) RGBD UAV (b) Reconstruction and Estimated Trajectory

Figure 1.5: RGBD-SLAM for Autonomous MAVs [37]

be a long and tedious process, with the design of a detailed 3D part usually requiring multiple revisions. Fabricating physical prototypes using low cost 3D fabrication technologies at intermediate stages of the design process is now a common practice, which helps the designer discover errors, and to incrementally rene the design. Most often, implementing the required changes directly in the computer model, within the 3D modeling software, is more difficult and time consuming than modifying the physical model directly using hand cutting, caving and sculpting tools, power tools, or machine tools. When one of the two models is modified, the changes need to be transferred to the other model, a process we refer to as synchronization.

KinectFusion, a framework that allows a user to create a detailed 3D reconstruction of an object or a small environment in real-time using Microsoft Kinect sensor, has garnered a lot of attention in the reconstruction and modeling field. It enables a user holding and moving a standard Kinect camera to rapidly create detailed 3D reconstructions of an indoor scene [19]. Not only an entire scene, a specific smaller physical object could also be cleanly segmented from the background model simply by moving the object directly. Fig. 1.6 shows how the interested object (a teapot) is accurately segmented from the background by physically removed. The sub-figure (A) shows surface normals, and sub-figure (B) is the texture mapped model. [12] proposed and introduced a from-Sense-to-Print system that can automatically generate ready-to-print 3D CAD models of objects or humans from 3D reconstructions using the low-cost Kinect sensor. Further, [14] addresses the problem of synchronizing the computer model to changes made in the physical model by 3D scanning the modified physical model, automatically detecting the changes, and updating

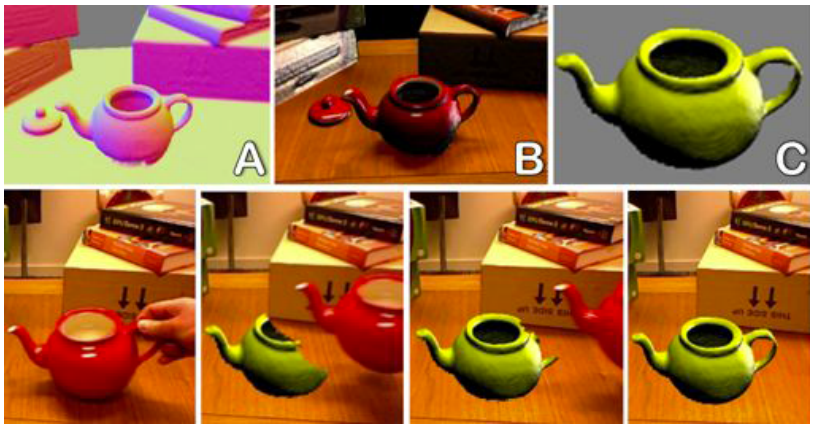


Figure 1.6: Object Segmentation in KinectFusion [19]

the computer model. A new method is proposed that allows the designer to move fluidly from the physical model (for example his 3D printed object, or his carved object) to the computer model. In the proposed process the physical modification applied by the designer to the physical model are detected by 3D scanning the physical model and comparing the scan to the computer model. Then the changes are reflected in the computer model. The designer can apply further changes either to the computer model or to the physical model. Changes made to the computer model can be synchronized to the physical model by 3D printing a new physical model.

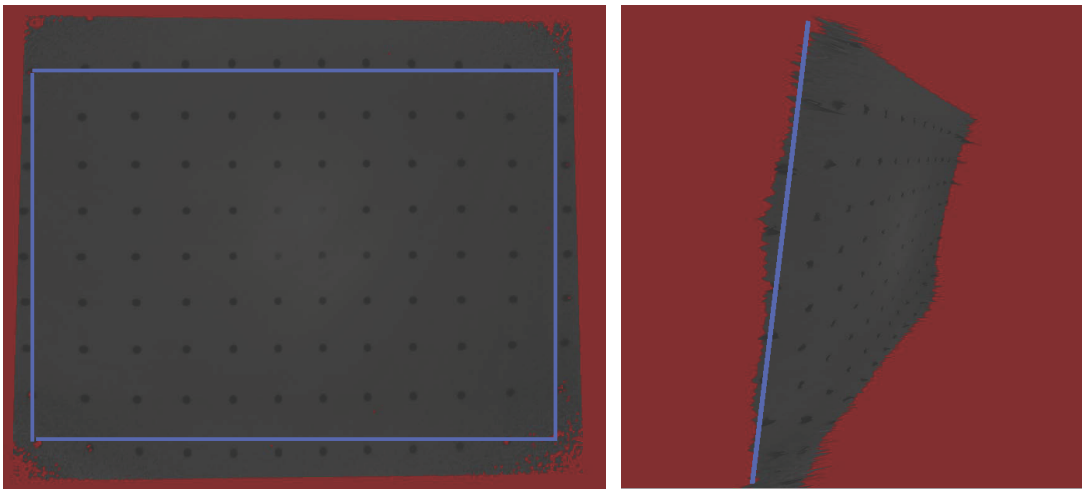
1.3 Calibration of RGB-D Cameras

Camera calibration is a necessary step in 3D computer vision in order to extract metric information from 2D images. The calibration of a RGB-D camera aims to be able to generate the world coordinates (X^W, Y^W, Z^W) with corresponding RGB values for every single pixel, given the depth steams and RGB streams retrieved from the camera. During the camera calibration, the removal of lens distortions must be involved in order to get the undistorted 3D reconstruction, especially for consumer cameras that utilize a low-cost lens. 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. With a wide-angle lens where the field of view of the lens is much wider than the size of the image sensor, barrel distortions will commonly happen. For example, fig. 1.7a shows KinectV2's raw (without distortion removal) NearIR 3D image observing a flat wall, on which a canvas printed with uniform grid dots pattern is hung. A blue rectangle is dropped onto the image, which helps show the "barrel-shape" deformation of the uniform grid pattern.

For decades, much work on camera calibration has been done, starting from the pho-

togrammetry community [3] [9], to computer vision ([42] [11] [49] to cite a few). After the release of the low-cost Microsoft Kinect, a lot of researchers focus their calibrations specially on RGB-D cameras (*e.g.*, Kinect). Jan *et al.* [38] analyze Kinect as a 3D measuring device, experimentally investigate depth measurement resolution and error properties and make a quantitative comparison of Kinect accuracy with stereo reconstruction from SLR cameras and a 3D-TOF camera. Using small data sets of Kinect, Carolina *et al.* [36] present a new method for calibrating a color-depth camera pair, which prevents the calibration from suffering a drift in scale. Aaron *et al.* [39] presented a novel multi-Kinect calibration algorithm that only requires a user to move single spherical object in front of the Kinects observed from multiple viewpoints.

However, those camera calibration methods, using limited number of points to train the non-linear distortion-removal model, are not able to cover all pixels of a sensor. Besides, with one pinhole camera model (3-by-4 transformation matrix) for raw (deformed) 3D reconstruction and another model to remove lens distortion, the final 3D reconstruction cannot be displayed efficiently in real-time. What's worse, those calibration methods are assuming that the depth sensor offers perfectly same precision depth value for all pixels. Whereas in practical depth sensors always have some defects in getting same depth accuracy for all of their pixels, which we will call as "Depth Distortion". Kourosh and Sander [21] provide an analysis of the accuracy and resolution of Kinect's depth data. They show that the random error of depth measurement increases with increasing distance to the sensor, and ranges from a few millimeters up to about 4 cm at the maximum range of the sensor. Fig. 1.7b shows the side view of KinectV2's raw 3D NearIR image observing the



(a) Front View (b) Left View

Figure 1.7: Raw Distorted KinectV2 NearIR 3D Reconstruction

Figure 1.8: KinectV2 Calibration System

wall, and the blue straight line is added on the left side of the 3D reconstruction. We can tell that most pixels on the left side border are apparently not sitting on a straight line.

In order to get enough calibrating points that can cover all pixels of the sensor, as well as solving the "Depth Distortion", a moving plane with camera on rail calibration system is introduced in this thesis. As shown in figure 1.8, the rail is perpendicular to the uniform round dots pattern, along with the Z^w axis. A RGB-D camera KinectV2 is mounted on the top of the slider, whose moving along the rail generates same effect as a moving pattern plane. This calibration system makes it possible to do per-pixel "Depth Distortion" calibration.

1.4 Summation

The depth sensor technologies opens a new epoch for three-dimensional (3D) markets. Ever since the Microsoft brought the low-cost depth camera Kinect into consumer market, RGB-D cameras are famous in in research areas of HCI, and an accurate RGB-D camera calibration is now important in computer vision. However, the traditional camera calibra-

tion methods are not ideal on accuracy and efficiency. They do not cover all pixels of a sensor, and did not consider the problem of depth "Depth Distortion" either. Besides, those methods need the distortion removal to be a second step after raw 3D reconstruction (from a pinhole camera matrix), which is not an efficient way displaying the 3D reconstruction in real-time.

In this thesis, a novel per-pixel calibration method with look-up table based 3D reconstruction in real-time is introduced, using a rail calibration system shown in fig. 1.8. With the pinhole camera model thoroughly abandoned, this proposed calibration method directly determines a beam equation for every single pixel by collecting undistorted X^w/Y^w with external accurate Z^w values. X^w/Y^w values will be generated from high order polynomial transformation, concretely $4th$ order, directly from image space *row* and $column$. And the accurate Z^w values are imported from external Z^w-aixs tracking module. In short, the proposed calibration method will be totally based on image streams data.

In Chapter 2, a pinhole-camera-model based calibration method is discussed in detail, including the lens distortions removal as the second step. The proposed data-based calibration method is well explained in Chapter 3, where two polynomial models will be introduced. One lens-distortions removal model helps map directly from image plane *row* and $column$ to X^w and Y^w separately, during calibration data collection. And another "depth distortion" removal model helps map per-pixel depth (D) to Z^W when generating look-up table for per-pixel beam equation. Z^w values will be totally supported from external BLE optical-flow sensor, will be discussed in Chapter 4. Finally, a *row* by $column$ by 6, three dimensional look-up table will be generated for real-time 3D reconstruction. Chapter 5 will talk the future work of RGB-D cameras calibration.

Chapter 2 Traditional RGB-D Cameras Calibration

2.1 Pinhole Camera

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

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

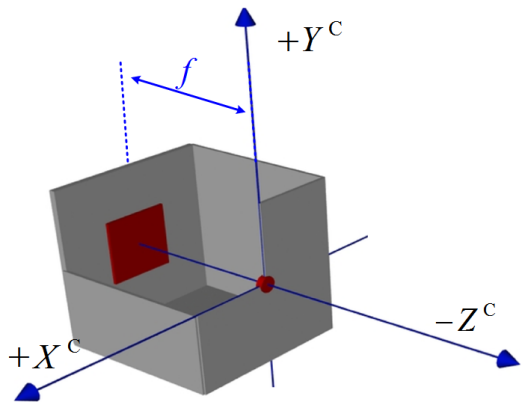


Figure 2.1: The Pinhole Camera Inspection

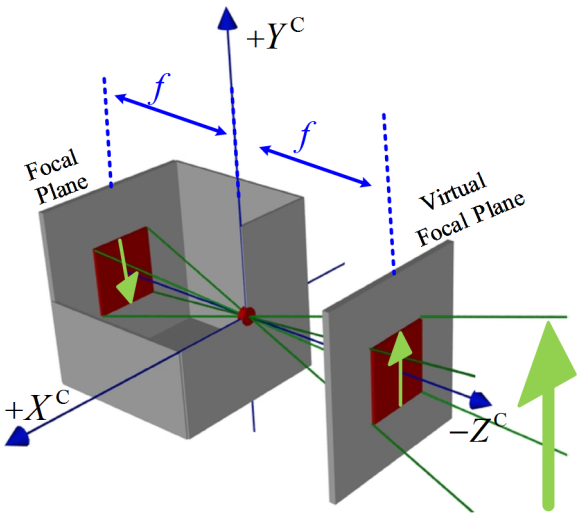


Figure 2.2: Virtual Focal Plane of a Pinhole Camera

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

With the virtual focal plane, the camera body with the real focal plane could be removed. And the rest parts in front of the the camera body, the focal point and the virtual focal plane together, form the most common pinhole camera model. In order to employ this model to analyze arbitrary 3D object points inside the camera's field of view in 2D image space, the prior step is to define the relationship between points in 3D camera space and the 2D image space (*row* and *column*). As shown in fig. 2.3, The focal point is right at the origin of the camera 3D space coordinates, from where to the sensor is the vertical distance of f, the focal distance. The 2D image coordinates are in dark green, and its origin is sitting

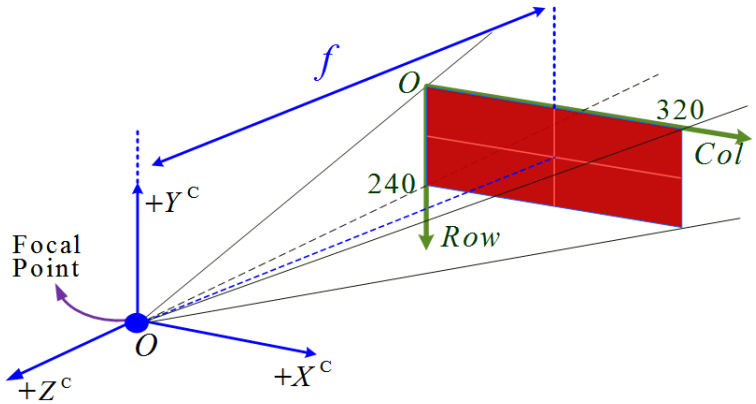


Figure 2.3: Common Pinhole Camera Model

at the up-left corner of the sensor. Only the a virtual sensor (in color red) is visible on the virtual focal plane, whose size in 2D image space is noted as 240*320 (using the size of PrimeSense camera). As long as both of the camera 3D space coordinates and image 2D space coordinates are defined, the next job is to build a mapping between them. Note that, the range of image should be either ([0:239], [0:319]) or ([1:240], [1:320]).

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

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

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

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

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

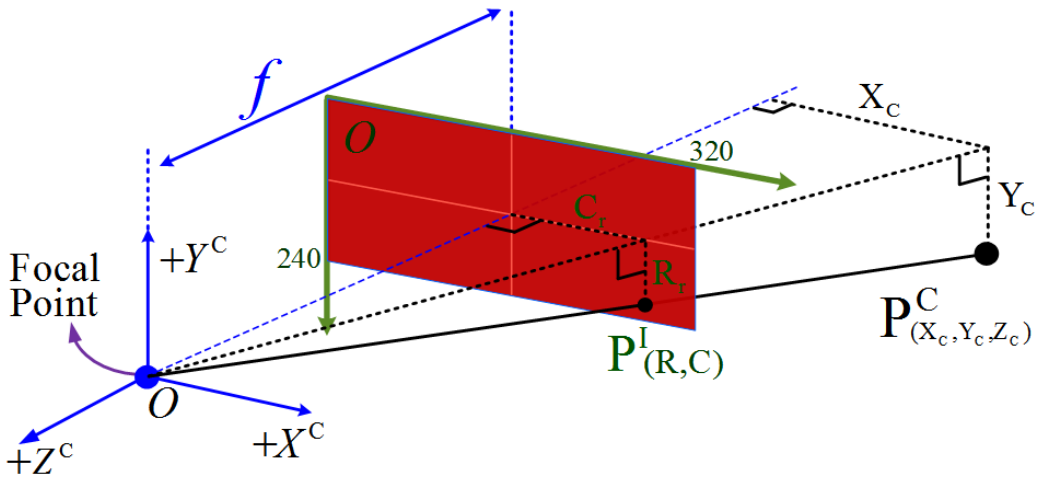


Figure 2.4: Mapping from Camera Space to Image Space

eqn. 2.2 will be written as

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

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

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

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

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

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

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

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

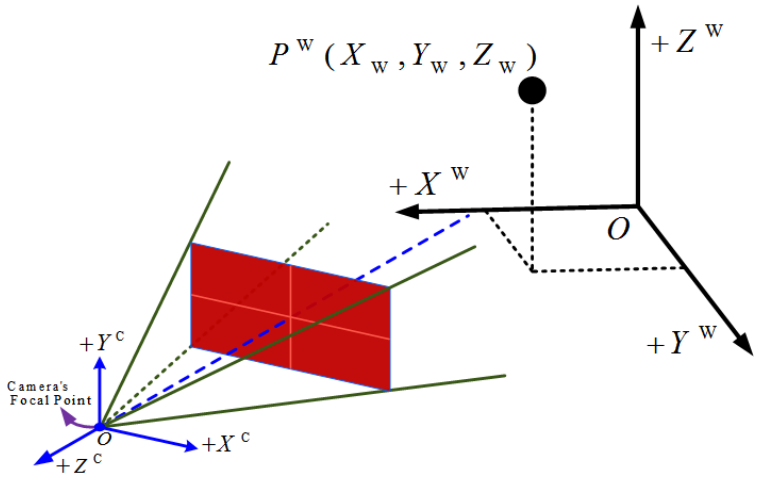


Figure 2.5: Pinhole Camera in World Space

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

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

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

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

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

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

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

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

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

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

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

2.2 3D Camera Calibration

The calibration of a 3D camera aims to be able to generate the world coordinates (X^W, Y^W, Z^W) and corresponding *RGB* values for every single pixel, given the depth steams and RGB streams retrieved from the 3D camera. From section 2.1, we know that the pinhole camera matrix M (eqn. 2.10) could help map from the world space to image space, however not

able to directly transform image space data to world space coordinates. In order to determine $X^W/Y^W/Z^W$ (based on eqn. 2.5 and eqn. 2.8), both of the intrinsic camera matrix and extrinsic camera matrix are needed, both of which are intermediate parameters and practically can only be determined through matrix M. Thus, the first job for 3D camera calibration is to solve the pinhole camera matrix M. To solve the pinhole camera matrix, we can use least squares fit with known 3D points (X^W, Y^W, Z^W) and their corresponding image points (R,C). With one point, based on eqn. 2.10 and 2.9, we can get two equations.

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

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

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

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

vector, as eqn. 2.13 shows below.

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

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

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

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

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

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

and

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

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

Now we find the way to determine both of the intrinsic camera matrix and the extrinsic camera matrix. With depth streams measuring Z^C, we are able to transform the 2D image data retrieved from the camera into 3D camera space point cloud by eqn. 2.5, and then generate the world space point cloud by eqn. 2.8. The basic pinhole camera model calculation is widely used in various camera calibration techniques. Based on different calibration systems, Zhengyou [49] classified those calibration techniques into four categories: unknown scene points in the environment (self-calibration), 1D objects (wand with dots), 2D objects (planar patterns undergoing unknown motions) and 3D apparatus (two or three planes orthogonal to each other).

Self-calibration technique do not use any calibration object, and can be considered as zero-dimension approach because only image point correspondences are required. Just by moving a camera in a static scene, the rigidity of the scene provides in general two constraints [31] on the cameras internal parameters from one camera displacement by using image information alone. Therefore, if images are taken by the same camera with fixed internal parameters, correspondences between three images are sufficient to recover both the internal and external parameters which allow us to reconstruct 3-D structure up to a similarity [30] [13].

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

jects. Xiangjian [15] shows how to estimate the internal and external parameters using one dimensional pattern in the camera calibration. And Zijian [52] employed one dimensional objects as virtual environments in practical multiple cameras calibration.

2D and 3D objects calibration systems usually give better calibrations. P. F. Sturm *et al*. [40] presented a general algorithm for plane-based calibration that can deal with arbitrary numbers of views that observe a planar pattern shown at different orientations, so that almost anyone can make such a calibration pattern by him/her-self, and the setup is very easy. Both of Matlab and OpenCV have applied this two dimension plane calibration method in their applications. Zhengdong [47] compared this two dimension plane camera calibration method and self-calibration method. Hamid [1] applied this method into practical calibration and employed the calibrated camera into camera pose estimation and distance estimation application.

In 3D object calibration technique, camera calibration is performed by observing a calibration object whose geometry in 3D space is known for very good precision. Calibration can be done very efficiently [11]. The calibration objects usually consist of two or three planes orthogonal to each other. Paul [6] applied the three dimension object calibration in his PHD project. Mattia [33] wrote a detailed tutorial from building the 3D object (fig. 2.6) for calibration, to scanning using the calibrated camera. Fig. 2.7a shows how six points are selected for calibration and fig. 2.7b shows the 3D reconstruction after calibration.

Zhengyou Zhang, a Chinese professor of computer science, IEEE and ACM Fellow and a specialist in computer vision and graphics, has deep studies on camera calibration from one-dimension calibration to tree-dimension calibration [50] [48] [49]. The accuracy of calibration from 1D to 3D is getting better, but the calibration system setting-up needs more and more work and cost as well. One dimension object is suitable for calibrating multiple cameras at once. Two dimension planer pattern approaches seems to be a good compromise, with good accuracy and simple setup. Also using the three dimension method

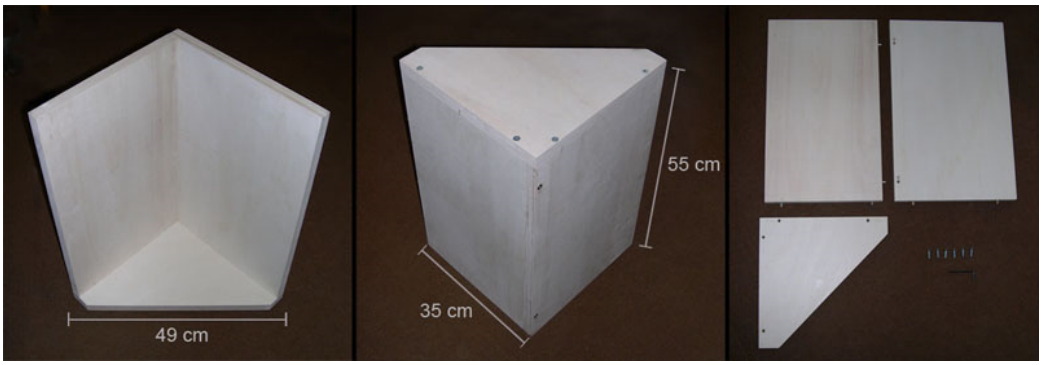
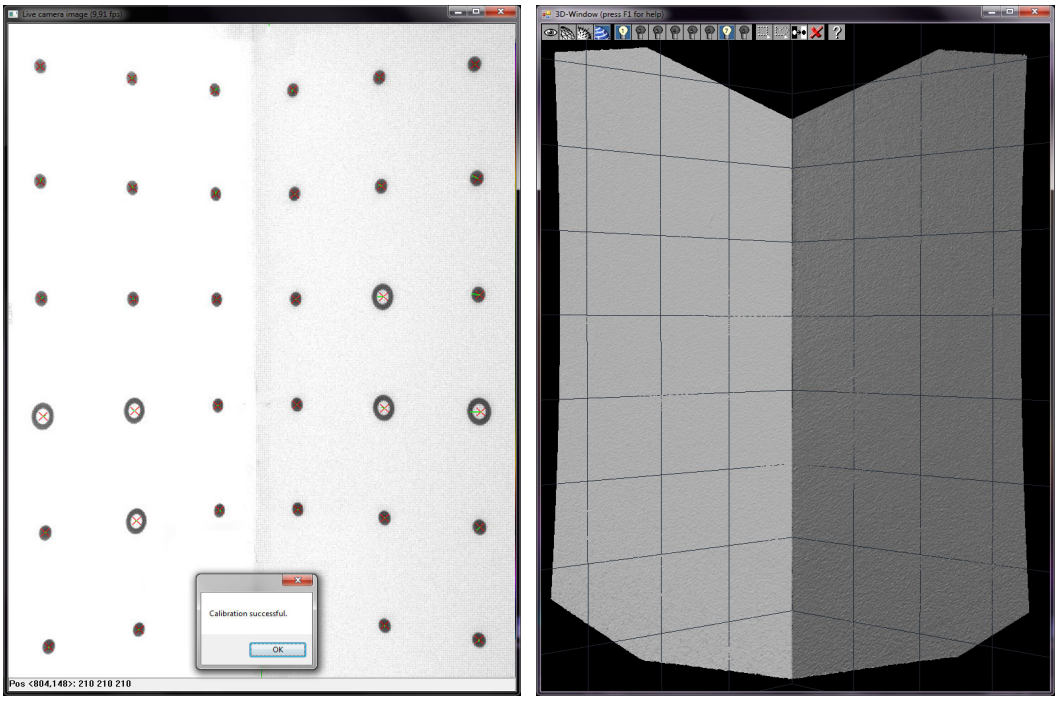


Figure 2.6: Building 3D Calibration Object [33]



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

Figure 2.7: Three Dimension Object Camera Calibration [33]

for calibration, Kai [29] derived the per-pixel beam equation, the linear relationship that could map to X^W/Y^W from Z^W as eqn. 2.18 shows, directly from pinhole camera matrix M for every single pixel. That is to say, we could easily look up X^W/Y^W after calibration once found the way to get Z^W.

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

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

2.3 Lens Distortion

All above in Chapter 2 are talking about the ideal pinhole camera, without lenses. Whereas in practical, as a result of several types of imperfections in the design and assembly of lenses composing the camera optical system, there are always lens distortions for a camera and the expressions in eqn. 2.2 are not valid any more. Lens distortion could be classified into two groups [43] : radial distortion, and tangential distortion. Imperfect lens shape causes light rays bending more near the edges of a lens than they do at its optical center.

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

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

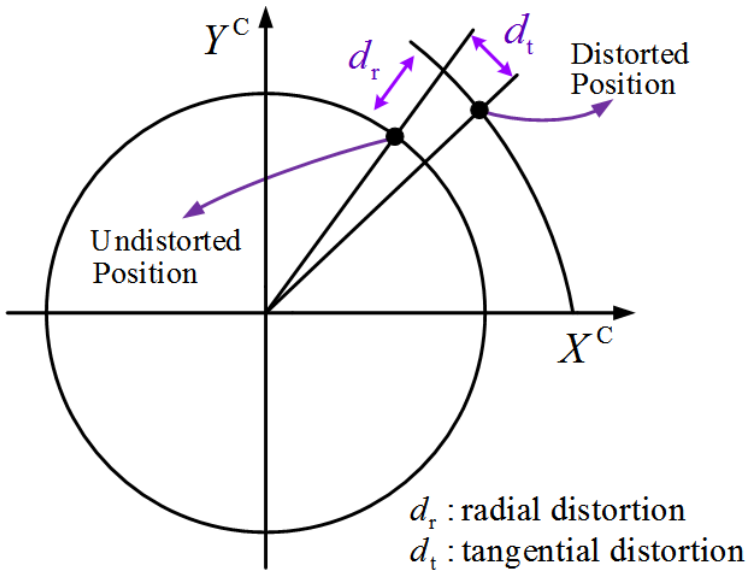


Figure 2.8: Radial and Tangential Distortion Affection In Image Space

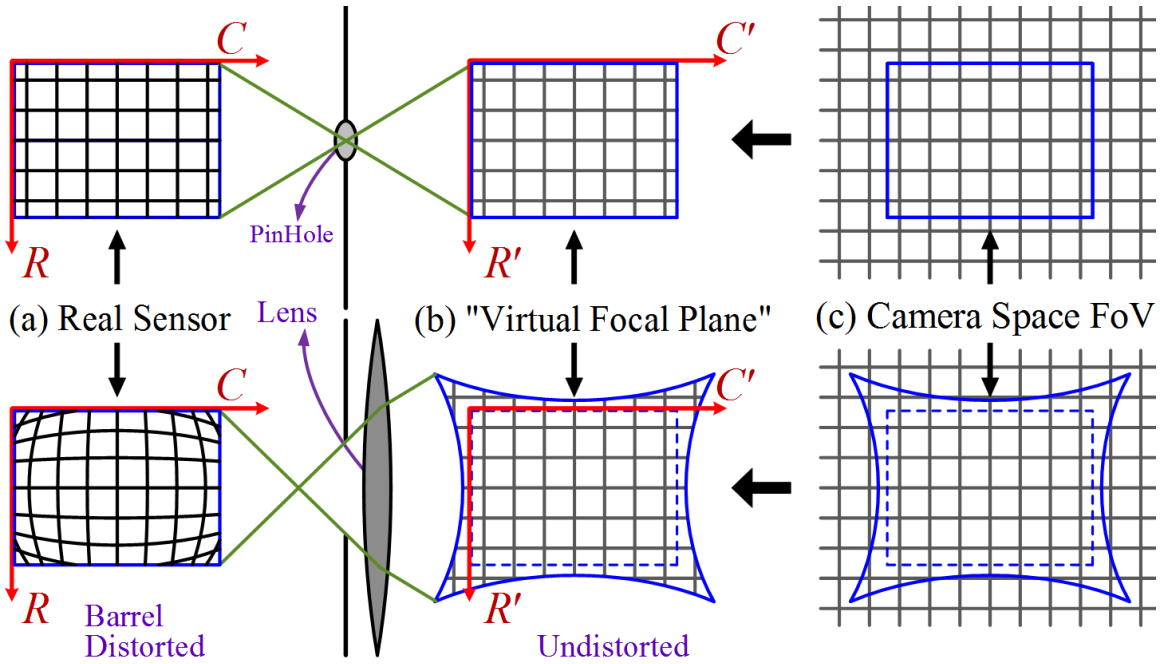


Figure 2.9: From Camera Space to Image Space with Lens Distortions

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

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

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

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

Duane [4] gave the lens distortion equation, and the undistorted *Col* and *Row* (C'/R' in our notation) can be expressed as power series in radial distance $r = \sqrt{C^2 + R^2}$:

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

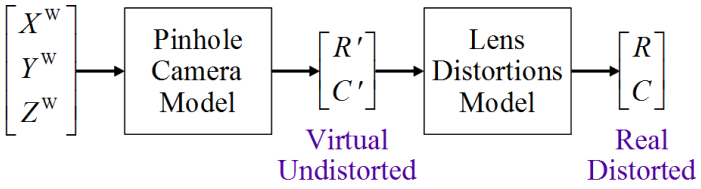


Figure 2.10: Traditional Camera Calibration Flow Char

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

2.4 Summation

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

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

Chapter 3 Data-Based Real-Time 3D Calibration

From Chapter 2 we know one dimension calibration method is suitable for calibrating multiple cameras together; three dimension object calibration has the highest accuracy but also cost more on system setup; two dimension plane calibration owns both of good accuracy and simple setup. However, those traditional calibration methods are not ideal enough while considering that researchers are chasing after accuracy. Either that, the static calibration pattern does not have enough points to offer distortions' information. Concretely only few limited points could be extracted in the three dimension calibration system as shown in Fig. 2.7a. Or, it is hard to control the extracted calibration points to cover enough area of the image space in the famous two dimension calibration methods. Fig. 3.1 shows the simulated multiple planes of checkerboard with respect to the camera space, with data from the two dimension calibration method that is employed in both of Matlab and OpenCV applications, to intuitively inspect the extrinsic parameters. Using this method, researchers need to manually keep changing their poses of holding the checkerboard, in order to get enough calibration points to cover all of the image space area. Besides, all of the traditional calibration methods are assuming that the depth sensor offers perfect accurate Z^C values for all of its pixels, *i.e.*, $Z^C_{[row,col]} - Depth_{[row,col]} = E_{\text{Constant}}$ such that for all image space range of $[row, col]$ pixels share one same error E_{Constant}. But in practical, depth sensors always have some defects in getting same depth accuracy for all of their pixels, which we will call as "Depth Distortion". As shown in Fig. 1.7b, even observing a flat wall there are still many bumps and hollows in the reconstructed 3D image.

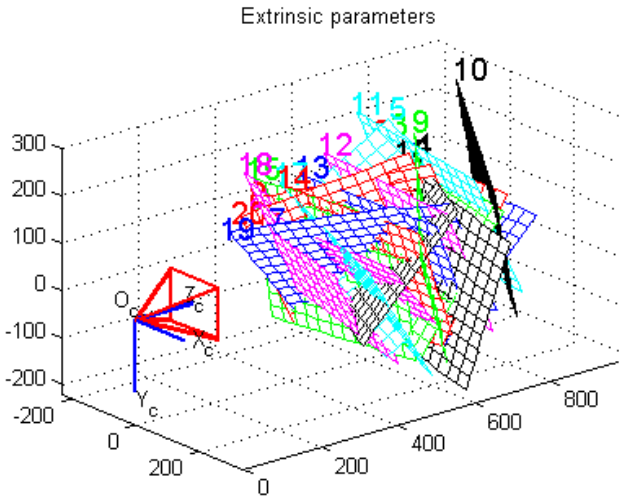


Figure 3.1: Simulated Planes of Checkerboards showing Extrinsic Parameters

Most of those defects in the methods discussed above are based on the losing control of the calibration points. Even though a flexible two dimension calibration method got its camera observing orientations easily changed by hand, it is almost impossible to numerically locate all desired poses that can make calibration points fill up all image space area. However, Tsai's old calibration system, which requires a known motion of the plane, can make them up. It has been obsoleted nowadays due to the fact that knowing the motion is not necessary for determining the intrinsic and extrinsic parameters. But as the resolution of camera sensor gets higher and higher, like a high definition sensor, researchers need a better calibration system. What's more, with the motion of the calibration plane controllable, it will not be a problem to determine the mapping from $Depth(D)$ to Z^W, thus equation 2.18 could be applied as one important step to simplify 3D reconstruction using a look-up table (lut) after calibration. Note that, both of the mapping from D to Z^W and the coefficients $c/d/e/f$ help map from Z^W to X^W/Y^W are with respect to per-pixel, such that even the "Depth Distortion" could be calibrated.

3.1 Novel Per-Pixel Calibration and LUT Reconstruction

In this thesis, we build a moving plane calibration system collecting tree dimensional data, with a rail that gets the RGB-D camera mounted on its slider observing a planar pattern. The 3D camera we used is KinectV2, but the calibration method could be applied on any RGB-D cameras. As shown in Fig. 1.8, a uniform grid dots pattern is hung on the wall, and the rail is perpendicular to the wall. We will assign the wall, on which the canvas is hung and printed with uniform grid dots pattern, as the $X^W Y^W$ plane in world space, and Z^W-axis would be along the rail. The RGB-D camera waiting to calibrate is mounted on the slider. Note that, in this calibration system, the only unit that needs to be perpendicular to the wall is the rail, whereas the RGB-D camera has no need to require its observation orientation. Because all of the mappings we are going to determine are with respect to per-pixel, *i.e.*, the calculated parameters group for every single pixel, which will determine the pixel's view, are independent with that of the other pixels. As the slider moves along the rail, the planar dots pattern hung on the wall is moving further with respect to the RGB-D camera. This rail offers the possibility of taking infinite various frames of various camera working distances (or Z^C). Although the dots pattern hung on the wall for camera calibration is static itself, however, the dots distributions would be dynamic (covering every single pixel) in the image space when counted together in all of those various frames of various Z^C.

In this per-pixel calibration method, we directly focus on the view of every single pixel,

the beam. Got inspired by Kai (eqn. 2.18), our camera calibration method consists of two big steps: frames ($X^{\mathrm{W}}Y^{\mathrm{W}}Z^{\mathrm{W}} + D$) data collection, plus per-pixel mapping parameters determination after frames collection; *i.e.*, to collect frames data of Z^{W} from external and (X^{W}, Y^{W}) by a transformation from (R,C), plus to calculate per-pixel parameters that help map from D to Z^{W} and $c/d/e/f$ in eqn. 2.18 that map from Z^{W} to $X^{\mathrm{W}}/Y^{\mathrm{W}}$. Note that, neither have we decided the mapping model from D to Z^{W}, nor from (R,C) to (X^{W}, Y^{W}). We save the flexibilities between accuracy and complexity for now, and will decide those two models based on data.

Assuming the total size of the depth sensor is as small as a point, and the Z^Ws for all of the pixels in one frame would share the same value, which could be measured by a laser distance measurer nearby the camera. To simplify the digital image processing (DIP) when extracting a desired point (R, C), which will soon be discussed in section 3.2, we assign the "2D origin" ($X^{\mathrm{W}}/Y^{\mathrm{W}} = 0$) of world space coordinates at the center of the dot which is closest to the center of the camera's field of view. And the laser measurer spot nearby the camera to at Z^{W}=0, such that Z^W values are always negative. Note that, the final assigned world space origin ("3D origin") is not necessarily be where the camera sensor is. It depends on the camera's observation orientation, whose changing leads to the change of the "2D origin" (and finally change the 3D origin).

In our calibration system, Z^W values for all pixels of every frame will be supported in real-time by a calibrated BLE Optical-Flow tracking module, as will be discussed later in section 4.2. The "Unit One" of Z^W value is assigned to be same with the side of unit-square of the uniform grid pattern, which can simplify the DIP processing of (C, R) extraction. Concretely, the distance between every two adjacent dots' centers in real-world is 228mm. Therefore, Z^W = -Z(mm) / 228(mm), where Z is distance in reality from the camera to dots-pattern plane along the rail. Fig. 3.2 shows one sample frame of 3D reconstruction in the assigned world space, where both of the origin and Z-axis are high-lighted in blue.

As for (X^{W}, Y^{W}) values' collection, a transformation from (R,C) is needed, during which the lens distortions correction must be considered. In the traditional calibration method, world space $X^{\mathrm{W}}/Y^{\mathrm{W}}/Z^{\mathrm{W}}$ are mapped to undistorted (R',C') by linear pinhole camera matrix M. And then the undistortion step from undistorted (R',C') to distorted (R,C) is done by eqn. 2.20, which uses a high order (higher than 2nd order) polynomial equation. Assuming that there is a high order mapping relationship directly from the distorted image space (R,C) to world space (X^{W}, Y^{W}), we will do different orders of two-dimensional polynomial prototypes in Matlab and then decide a best-fit mapping model.

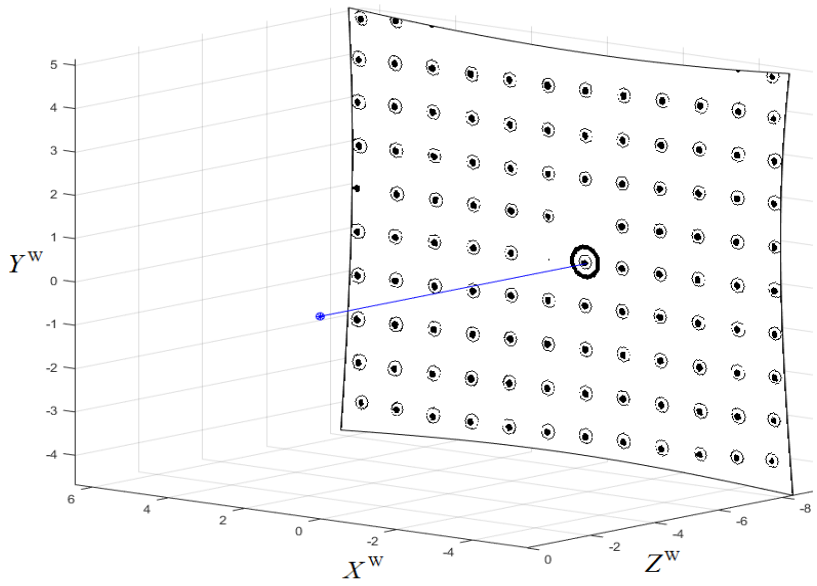


Figure 3.2: NearIR $X^W Y^W Z^W$ 3D Reconstruction

With squared-shape distributed points (C, R) extracted from image space, we can recover a distorted image in Matlab, as shown in fig.3.3a. Using a mathematical distortion (d) measurement [41]

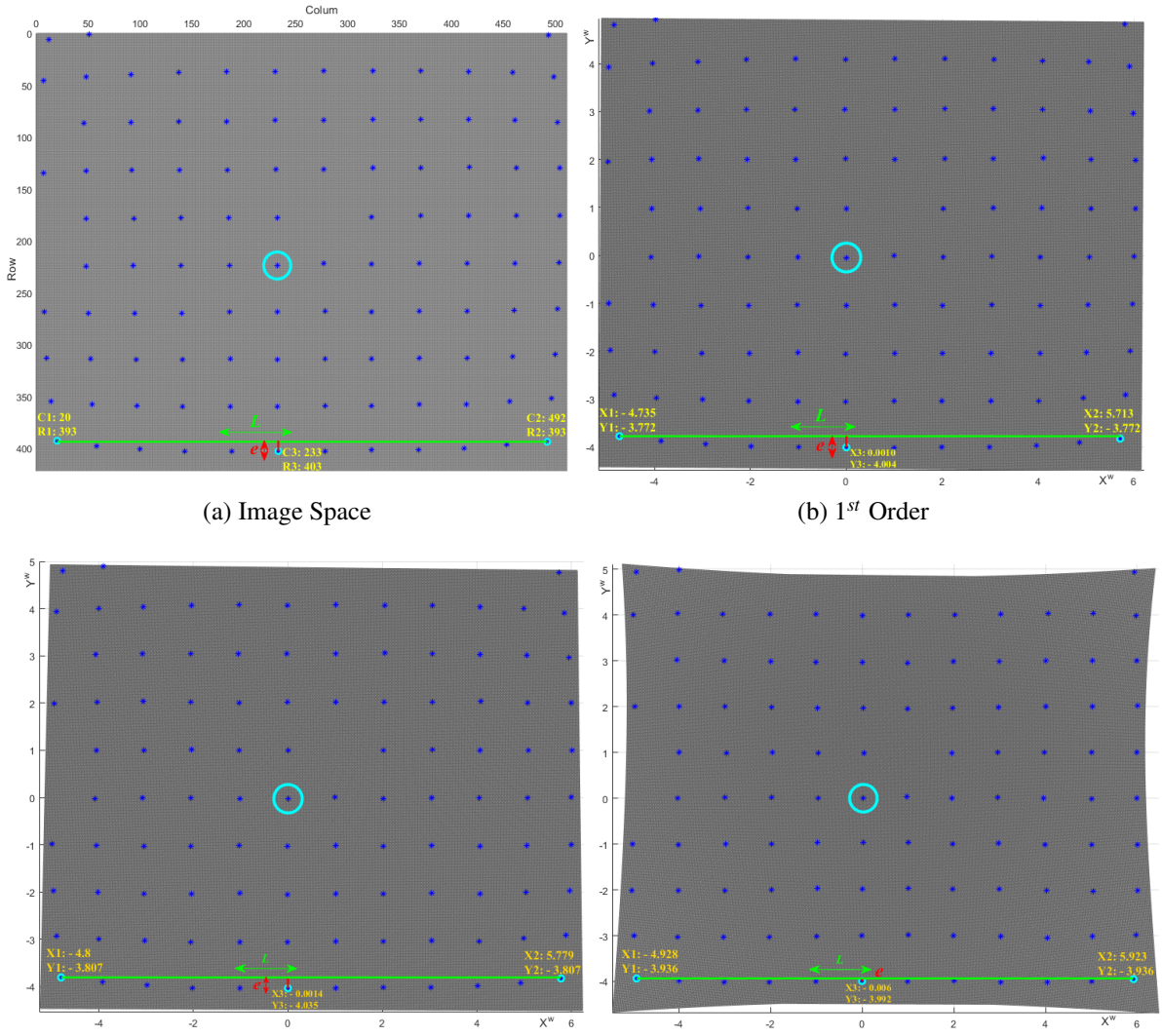
$$d(\%) = e * 100/L, \tag{3.1}$$

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

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

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

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



(a) Image Space

(b) 1^{st} Order

(c) 2^{nd} Order

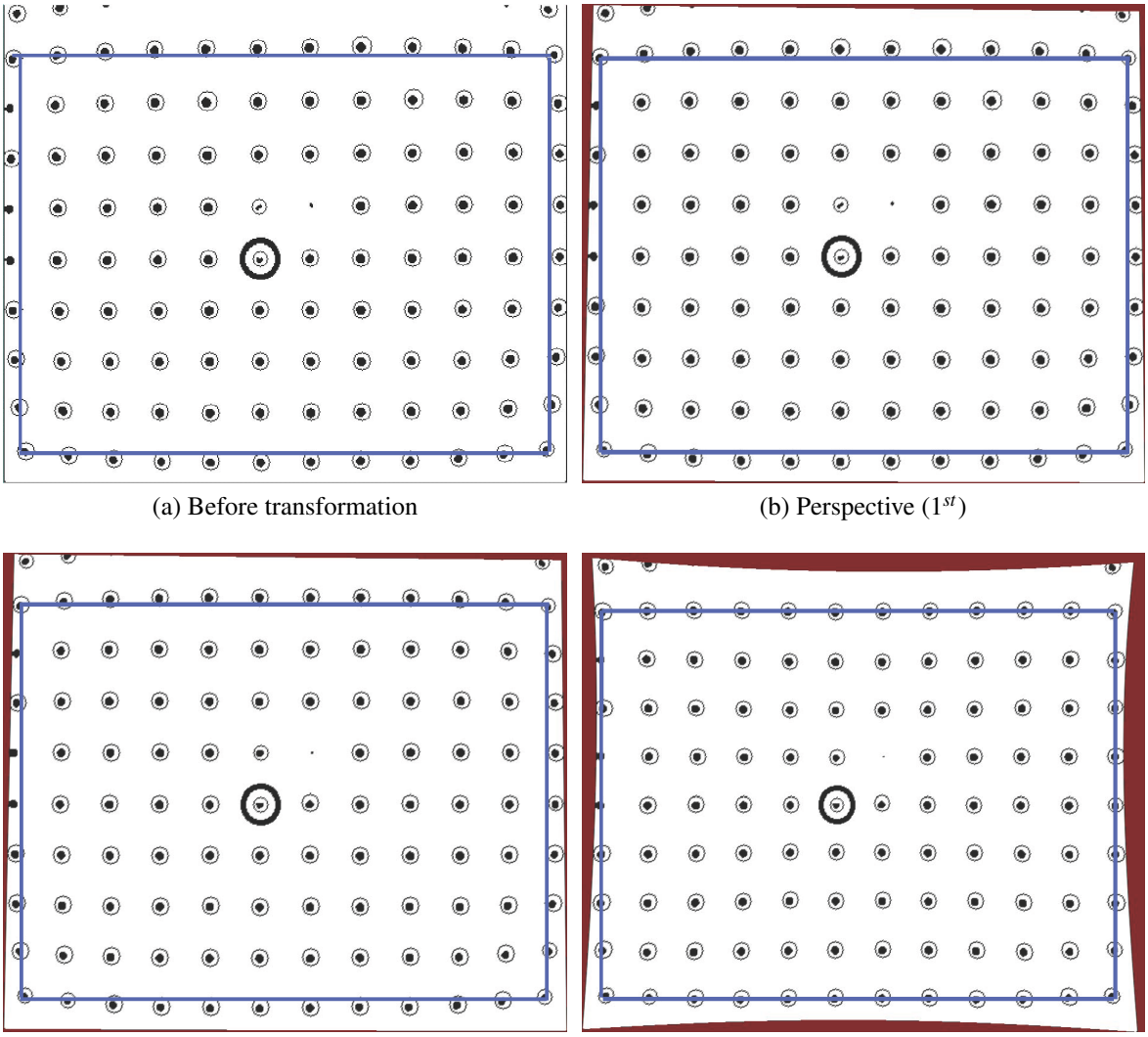
(d) 4^{th} Order

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

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

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

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



(a) Before transformation

(b) Perspective (1^{st})

(c) 2^{nd} Order

(d) 4^{th} Order

Figure 3.4: NearIR Stream High Order Polynomial Transformation

To prototype equation 3.3 and 3.4 in Matlab, "Curve Fitting Toolbox" is used to obtain the 2x6 and 2x15 parameters. Based on the "Goodness of fit" of transformation parameters from Matlab, the Root-Mean-Square Error (RMSE) of (X^W, Y^W) is (0.06796, 0.05638) for the 2^{nd} order polynomial, and (0.02854, 0.02343) for the 4^{th} order polynomial. Fig. 3.3c and fig. 3.3d show the transformed images in world space respectively by the 2^{nd} order and 4^{th} order polynomial parameters, from which we can get $d_2 = [-3.807 - (-4.035)]/[5.779 - (-4.8)] = 2.1\%$ and $d_4 = [-3.936 - (-3.992)]/[5.923 - (-4.928)] = 0.516\%$. It is straightforward to tell that, d_4 is much smaller than d_0 and fig. 3.3d intuitively

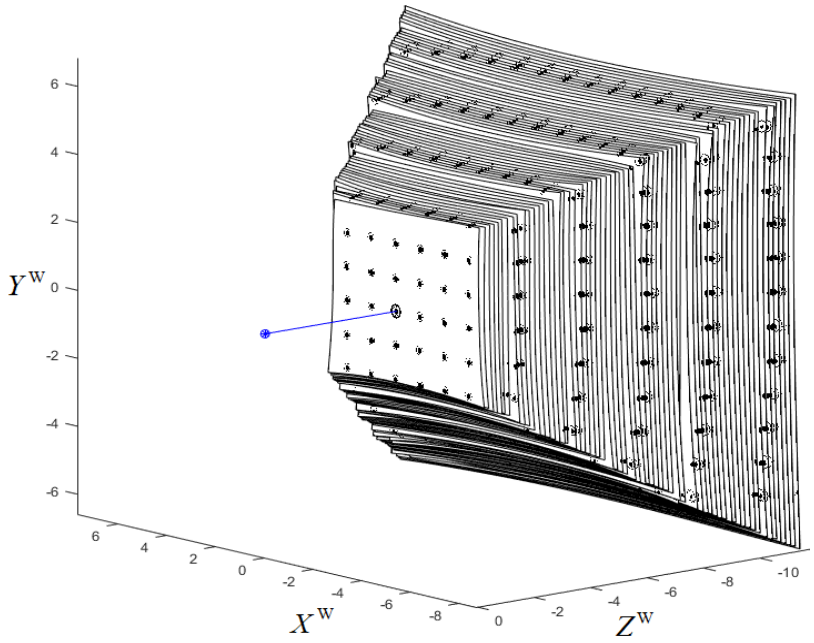


Figure 3.5: 63 Frames NearIR Calibrated 3D Reconstruction

shows a satisfying calibrated image.

Then, by applying those polynomial mappings into real-time streams transformation, we can get the transformed stream images. As shown in fig. 3.4, the outlines of the transformed steam images are same with Matlab prototypes in fig. 3.3. It is easy to tell that the 4^{th} order polynomial surface mapping is much better than the second order, and a higher order than 4^{th} could be more accurate. However, as the order of the polynomial mapping goes higher, the number of parameters also get larger and larger, which costs more calculations and requires more data (coordinate-pairs) for training the transformation model. Considering that a 5^{th} order polynomial mapping will have much more (2x21=42) parameters to calculate while may not enhance much accuracy, we choose the 4^{th} order polynomial as the main mapping model to get $X^W Y^W$ values from RC. Limited by the static dot pattern, fewer and fewer dot-clusters could be observed by the camera as the camera getting closer to the dot pattern. Practically, 4^{th} order calibration is replaced by 2^{nd} order to guarantee a robust software when the observed dot-clusters are too few to train the transformation model.

Till now, the mapping model from (R,C) to (X^W, Y^W) has been decided, with which the data collection of $X^W Y^W Z^W + D$ is ready to be completed. Fig. 3.5 shows 63 frames of collected data, which gives an pyramid shape of a camera sensor's field of view. With enough data, a best-fit mapping model from D to Z^W could be determined. Both of D and

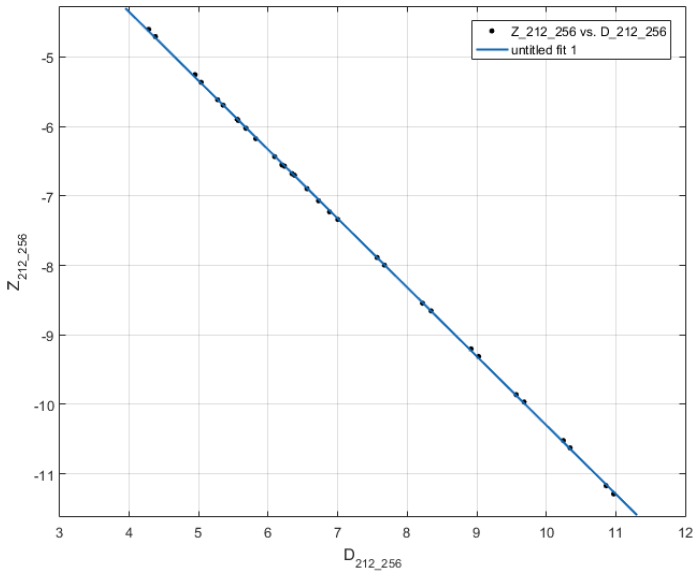


Figure 3.6: Polynomial Fitting between D and Z

Z^{W} are continuous data, so that their function could written as a polynomial expression, based on Taylor series. Fig. 3.6 shows the polynomial fitting result in Matlab "Curve Fitting Tool" toolbox, with 32 points of DZ^{W} values (at pixel *column*=256 and *row*=212) from 32 frames. It is apparent that Z^{W} is linear with D, which is also reasonable. Therefore, for every single pixel, Z^{W} could be mapped from D through

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

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

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

3.2 DIP Techniques on (R, C) Extraction

In order to train the 4^{th} order mapping model that can map from (R, C) to (X^W, Y^W), we need to obtain at least 15 points' coordinate-pairs of both image space coordinates and world space coordinates. Therefore, a robust DIP process to extract the points' addresses

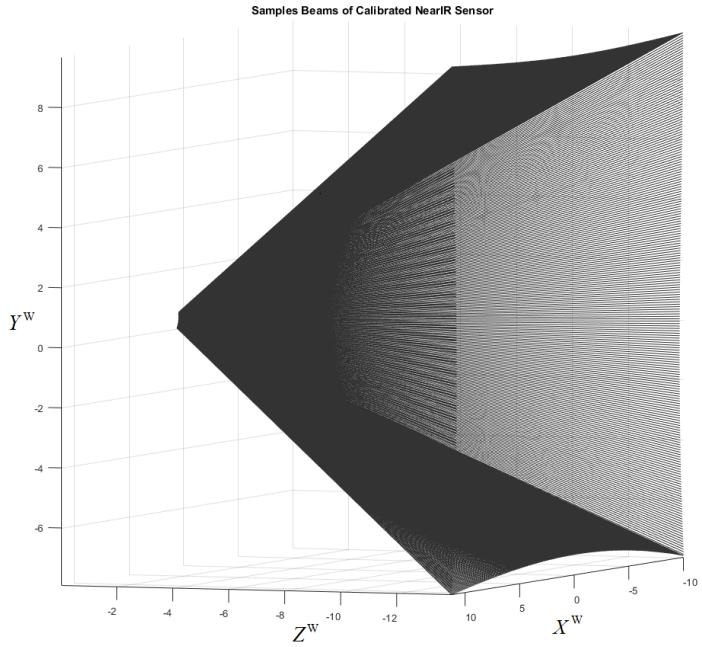


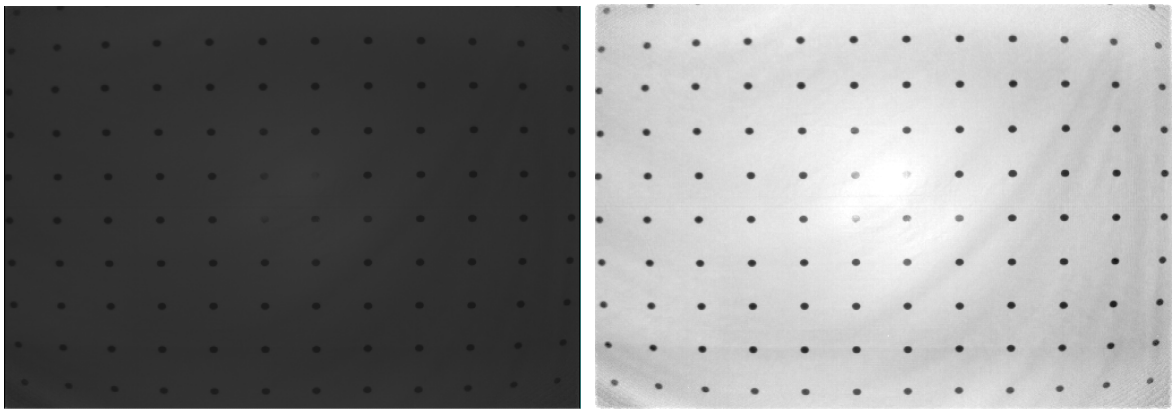
Figure 3.7: Sample Beams of Calibrated NearIR Field of View

(R, C)s is of the top priority. Concretely in this project, we are going to extract the dot-clusters' centers from KinectV2 NearIR steams using DIP techniques. To guarantee a robust processing, the extraction steps consist of gray-scaling, histogram equalization, adaptive thresholding and a little trick on black pixels counting. OpenGL is selected as the CPU image processing language. The default data type of steams saved on GPU during processing is single-floating, with a range from 0 (balck) to 1 (white).

Gray-scaling is done in order to suit for both of the RGB and the NearIR steams. For NearIR steam, its data contains only color gray. There is no need to consider gray-scale problem, and data will be saved on GPU as single-floating automatically. Whereas for RGB steam, a conversion from RGB to gray value is needed. Typically, there are three converting methods: lightness, average, and luminosity. The luminosity method is finally chosen as a human-friendly way for gray-scaling, because it uses a weighted average value to account for human perception, which is give by eqn. 3.6.

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

As values saved on GPU, all of the pixel intensity values are within the range of [0, 1], where "0" means 100% black and "1" means 100% white. In practical, NearIR steam image is always very dark, as shown in Fig. 3.8a (with their intensity values every close to zero). In order to enhance the contrast of NearIR image for a better binarizing, rescaling is necessary. In this section, histogram equalization technique is used maximize the



(a) Raw (b) Histogram Equalized

Figure 3.8: NearIR Streams before / after Histogram Equalization

range of valid pixel intensity distributions. Same process is also compatible on the RGB steam. Commonly, Probability Mass Function (PMF) and Cumulative Distributive Function (CDF) will be calculated to determine the minimum valid intensity value ($floor$) and maximum valid value ($ceiling$) for rescaling, whereas tricks could be used by taking advantage of the GPU drawing properties.

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

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

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

equal to one. Given the width (range of x-axis) of customer framebuffer being [-1, 1] in OpenGL, which is twice the range of pixel intensity [0, 1], the half-width of the customer framebuffer is same with the total number N of gray levels, which determines the precision of $floor$ / $ceiling$ intensity selection.

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

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

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

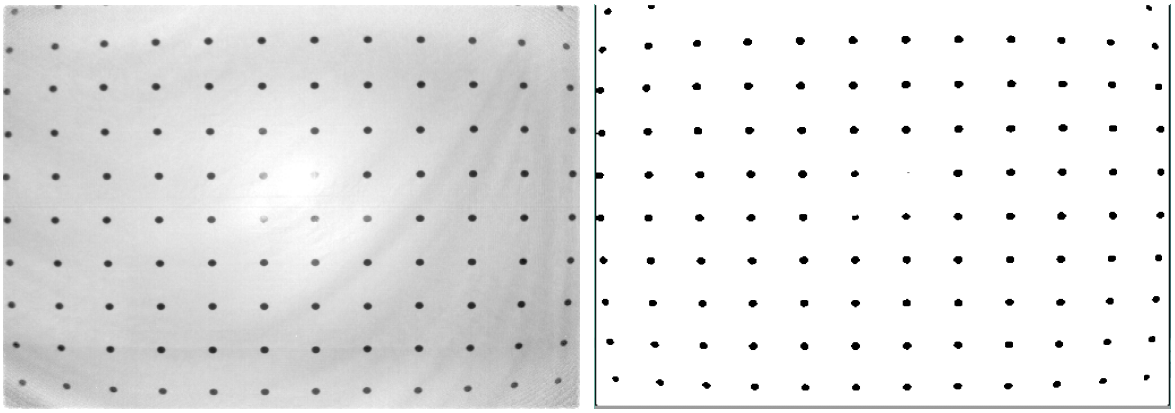
$$\begin{aligned} floor &= n_{\text{\_floor}}/N \\ ceiling &= n_{\text{\_ceiling}}/N \end{aligned} \tag{3.9}$$

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

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

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

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



(a) Histogram Equalized (b) Binarized

Figure 3.9: NearIR Streams before / after Adaptive Thresholding

(subtraction) process for every single pixel could be written as

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

where I is short for $Intensity$ of every single pixel, and $C_{\text{\_offset}}$ is a small constant that could be adjusted depending on various thresholding situations. In this project, $C_{\text{\_offset}}$ is around 0.1.To sharpen the edge of the binarized image for a better "circle" shape detection, a median filter could be added as the last step of adaptive thresholding. As shown in Fig. 3.11, background is removed in the binarized image after adaptive thresholding.

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

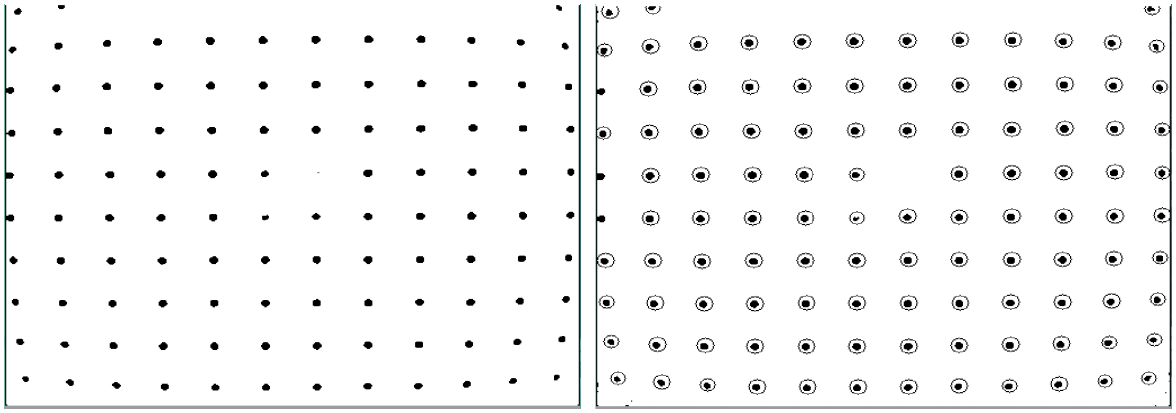
E	A	F
B	O	C
G	D	H

To turn the surroundings *A*~*H* into marks, different weights will be assigned to them. Those markers with different weights have to be non-zero data, and should be counted as the edge-part of circles. Therefore, the first step is to inverse the binary image, generating an image that consists of circle-shaped "1"s distributed in a background of "0"s.After reversing, the next step is to assign weight to the surroundings. OpenGL offers convenient automatic data type conversion, which means the intensity values from "0" to "1" of single-floating data type save on GPU could be retrieved to CPU as unsigned-byte data type from "0" to "255". Considering a bitwise employment of markers, a binary calculation related weight assignment is used in the shader process for pixels. The intensity reassignment for every single pixel is expressed as the equation below.

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

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

As for the sniffing step, base on the distribution table of *A*~*H* that has been discussed above and their corresponding weight given by equation 3.12, the markers *A*/*B*/*C*/*D* are



(a) before Extraction (b) Extracted and Marked

Figure 3.10: Valid Dot-Clusters Extracted in NearIR

valid (non-zero) as long as the intensity value of pixel O satisfies the following conditions shown as below.

$$\begin{aligned} &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}80 == 1), \quad then, \quad \text{marker } A \text{ is valid} \ \text{(go Up)} \\ &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}40 == 1), \quad then, \quad \text{marker } B \text{ is valid} \ \text{(go Left)} \\ &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}20 == 1), \quad then, \quad \text{marker } C \text{ is valid} \ \text{(go Right)} \\ &if\ (I_{\mathrm{O}}\ \&\ 0\mathrm{x}10 == 1), \quad then, \quad \text{marker } D \text{ is valid} \ \text{(go Down)} \end{aligned} \tag{3.13}$$

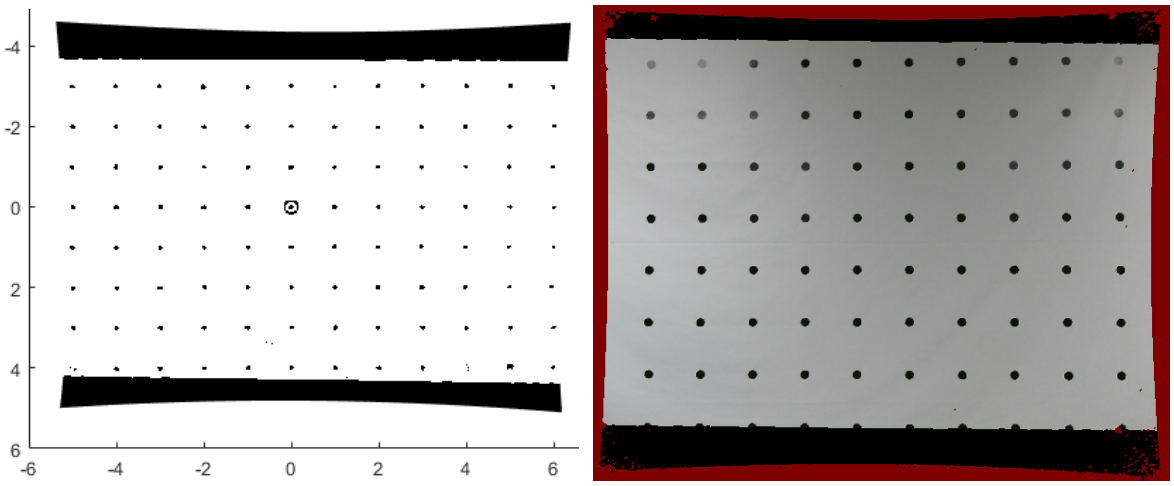
Once a valid marker is found, its address $(column, row)$ will be saved into the "stops-address" queue. One pixel's address might be saved for up to 4 times, but "colonizing" procedure will only happen once at the first time, so that the sniffing will stop once all of the connected valid pixels in a singular dot-cluster are colonized as zeros. In the second step "colonizing", I_{O} is changed to zero, variable *area* of this dot-cluster pluses one, and bounding data $RowMax$ / $RoxMin$ / $ColumnMax$ / $ColumnMin$ are also updated.Finally, the Round Dot Centers $(column, row)$ could be determined as the center of bounding boxes with their borders $RowMax$ / $RoxMin$ / $ColumnMax$ / $ColumnMin$. After potential noises being removed based on their corresponding *area* and shape (ratio of width and height), the data left are taken as valid dot-clusters. As shown in Fig. 3.10b, the centers of valid dot-clusters are marked within their corresponding homocentric circles.

3.3 Alignment of RGB Pixel to Depth Pixel

A undistorted 3D reconstruction could be displayed with the help of a LUT generated by the per-pixel calibration method, which has been discussed in section 3.1. However, we have not figured out yet what the color of each pixel is. To generate a colored 3D reconstruction with a combination of a random depth sensor and a random RGB sensor, we need to align the RGB pixels to depth pixels. The intermediate between the depth sensor image space and RGB sensor image space is the world space. As long as we figure out the mapping from world space to RGB sensor image space, the color of pixel with known $X^W Y^W Z^W$ could be looked up from the RGB image space. The pinhole camera matrix M is used to map from world space to RGB image space. Using the frame data from Kinect RGB steams and Kinect NearIR streams, a Matlab prototype of RGB pixels alignment is shown in fig. 3.11a, where the RGB textured is mapped onto its corresponding NearIR image, who has same pixels with depth sensor. The total black area on the top edge and bottom edge is where the depth sensor's view goes beyond the RGB sensor's view. Fig. 3.11b shows the screen-shot of live video after calibration with the RGB pixels aligned by a pinhole camera model M.

3.4 Summation

A per-pixel calibration method, using a moving plane calibration system, is proposed in this chapter. The main idea of this method is to directly determine the per-pixel mapping from D to Z^W, and then to X^W/Y^W. Therefore, this per-pixel calibration method con-



(a) (b)

Figure 3.11: Alignment of RGB Texture onto NearIR Image

sists of two big steps: $X^W Y^W Z^W + D$ data collection and mapping determination. D is directly from depth streams. Z^W is from external based on the camera's position on the rail. And (X^W, Y^W) are from the transformation of R/C by a 4^{th} order polynomial mapping model. With the frames data of $X^W Y^W Z^W + D$ collected, the per-pixel mapping parameters $a/b/c/d/e/f$ could be determined by eqn. 3.5 and eqn. 2.18. And finally a *column*-by-*row*-by-6 look-up table that contains 6 coefficients $(a/b/c/d/e/f)$ for every single pixel will be generated for the display of a calibrated real-time 3D reconstruction. Without using the traditional pinhole camera model, two polynomial mapping models are employed in this calibration method. The first model is the two-dimensional 4^{th} order polynomial mapping from R/C to X^W/Y^W during the frames data collection, which takes care of the removal of lens distortions; and the second model is the linear mapping from D to Z^W, which can handle "depth distortion". Both of the two mapping models are determined and calculated by real streams data from the camera, so that we claim this per-pixel calibration method a "data-based" calibration method.

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

4.1 Rail System

As already shown in chapter **??** 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 4.1 shows.

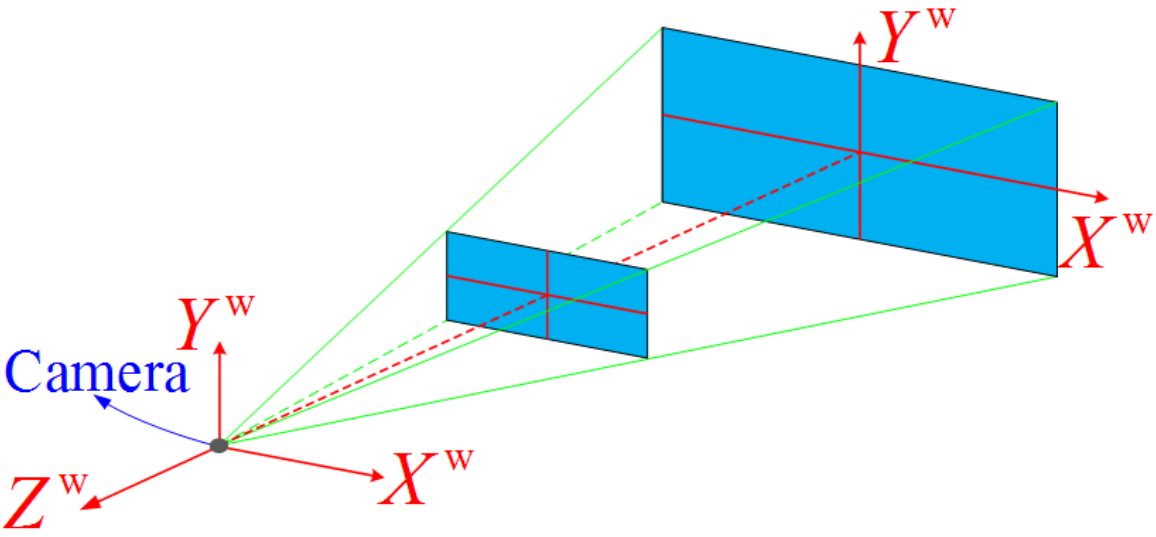


Figure 4.1: World Coordinate Frame

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.

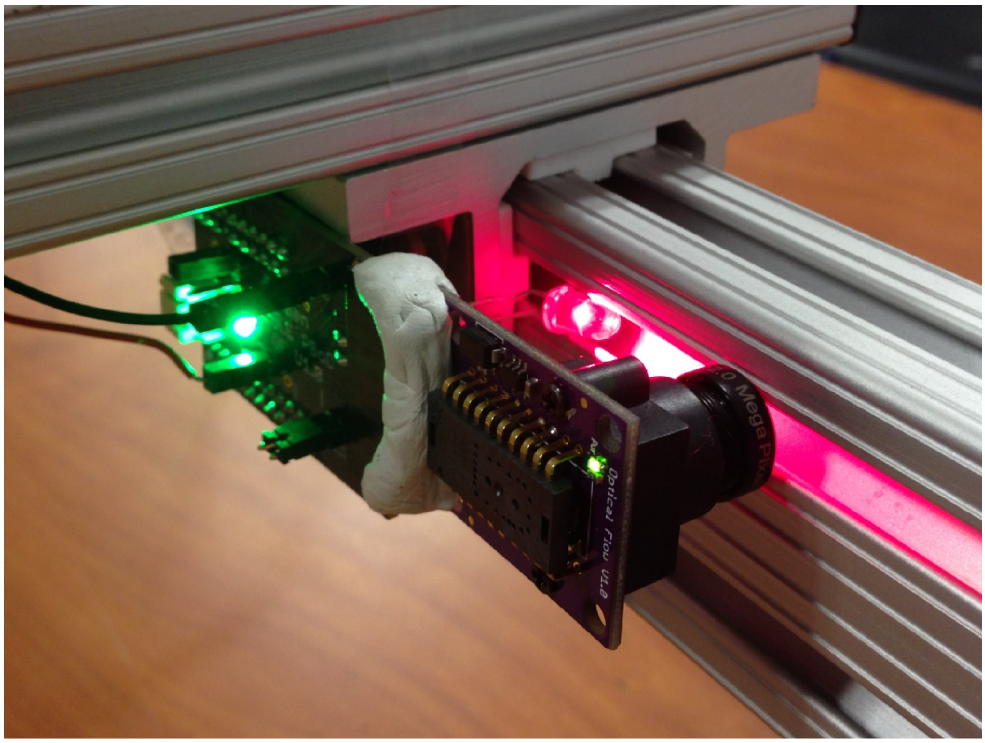


Figure 4.2: Mounted BLE Optical-Flow Tracking Module

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" 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 [20]. 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. 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 [2]. 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. Sensor ANDS-3080 has a programmable frame rate of over 6400 fps. With a lens

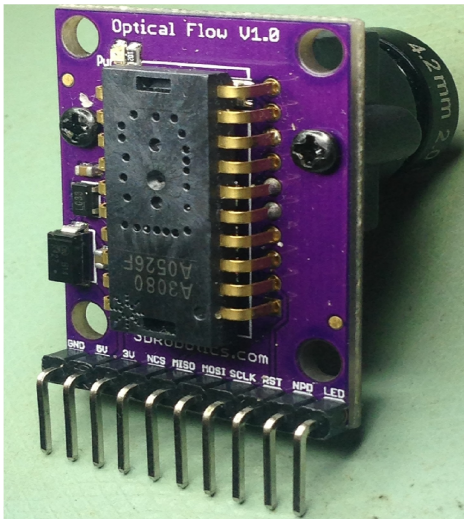


Figure 4.3: Optical Mouse Sensor

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. 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 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.

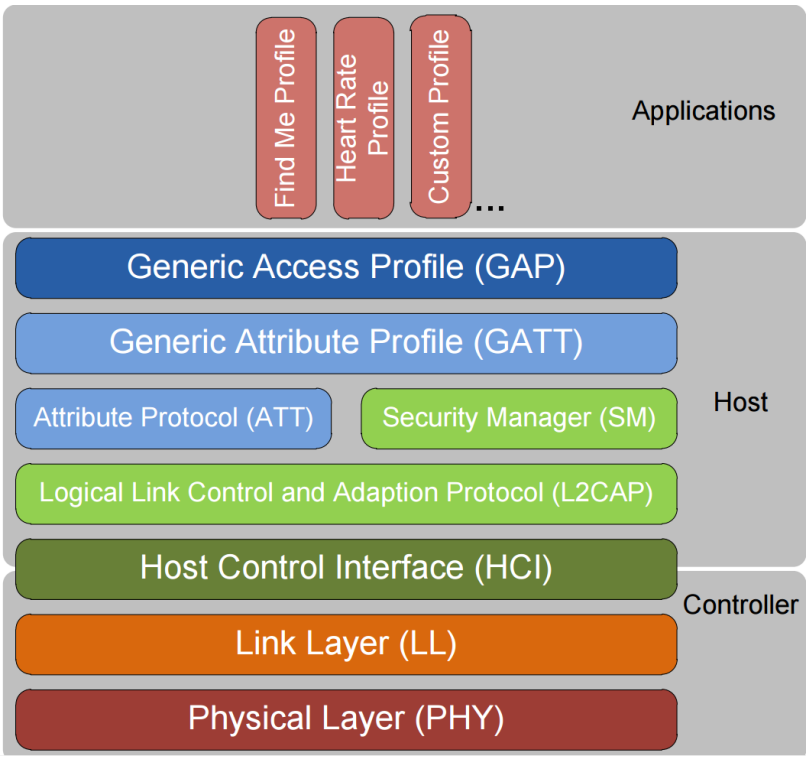


Figure 4.5: BLE Protocol Stack [23]

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.

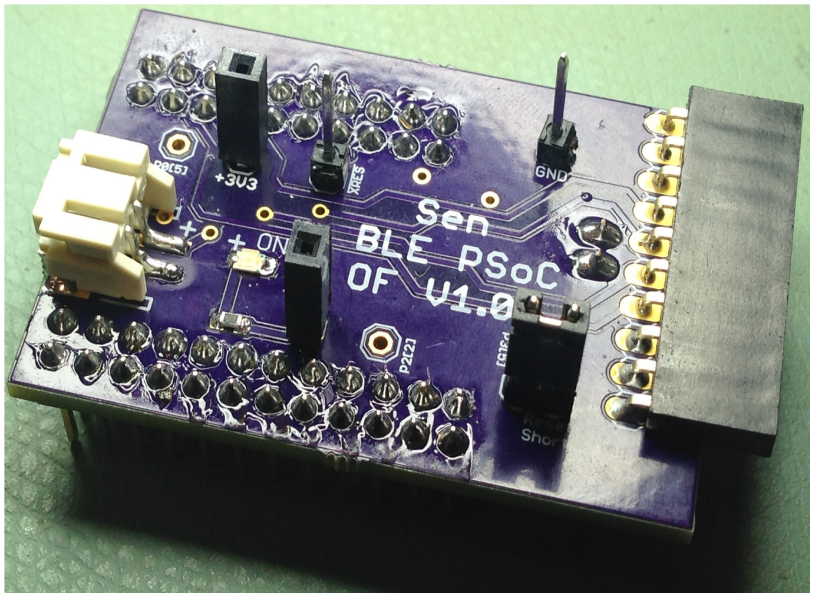
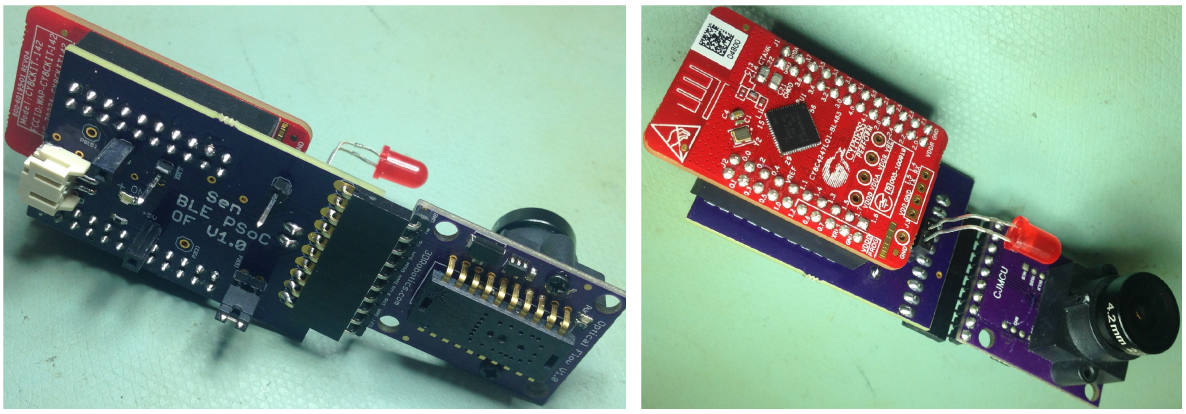


Figure 4.6: PCB: joint & power supply



(a) Front (b) Back

Figure 4.7: Overview of BLE OF Tracking Module

Bibliography

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

[2] S. S. Beauchemin and J. L. Barron. The Computation of Optical Flow. *ACM Computing Surveys (CSUR)*, 27(3), Sep 1995.

[3] D. C. Brown. Close-range camera calibration. *Photogrammetric Engineering*, 37(8), 1971.

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

[5] Three Depth-Camera Technologies Compared. F. Pece and J. Kautz and T. Weyrich". *First BEAMING Workshop*, June 2011.

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

[7] Yunxin Duan, Hui Deng, and Feng Wang. Depth Camera in Human-Computer Interaction: An Overview. In *Intelligent Networks and Intelligent Systems (ICINIS), 2012 Fifth International*, 2012.

[8] Felix Endres, Jurgen Hess, Nikolas Engelhard, Jurgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2012.

[9] W. Faig. Calibration of close-range photogrammetry systems: Mathematical formulation. *Photogrammetric Engineering and Remote Sensing*, 41(12), 1975.

[10] Maurice F. Fallon, John Folkesson, Hunter McClelland, and John J. Leonard. Relocating Underwater Features Autonomously Using Sonar-Based SLAM. *IEEE Journal of Oceanic Engineering*, 38(3), Jul 2013.

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

[12] Nadia Figueroa, Haiwei Dong, and Abdulmotaleb El Saddik. From Sense to Print: Towards Automatic 3D Printing from 3D Sensing Devices. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, Oct 2013.

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

[14] Ammar Hattab and Gabriel Taubin. 3D Modeling by Scanning Physical Modifications. In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, Aug 2015.

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

[16] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. In *Experimental Robotics*, 2014.

[17] C. W. Hull. Apparatus for production of three-dimensional objects by stereolithography,. *US Patent US4575330 A*, Mar 1986.

[18] Dan Ionescu, Viorel Suse, Cristian Gadea, and Bogdan Solomon. A single sensor NIR depth camera for gesture control. In *2014 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*, May 2014.

[19] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, and Richard Newcombe. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011.

[20] J.L.Barron and N.A.Thacker. Tutorial: Computing 2D and 3D Optical Flow. *TINA Memos*, (2004-012), Jan 2005.

[21] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. *Sensors*, 12(2), Feb 2012.

[22] Henrik Kretzschmar, Cyrill Stachniss, and Giorgio Grisetti. Efficient information-theoretic graph pruning for graph-based SLAM with laser range finders. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep 2011.

[23] Krishnaprasad M V (KRIS). BLE Protocol. *Cypress Application*, (AN91267), Oct 2015.

[24] Kam Lai, Janusz Konrad, and Prakash Ishwar. A gesture-driven computer interface using Kinect. In *Image Analysis and Interpretation (SSIAI)*, Apr 2012.

[25] P. Lavoie, D. Ionescu, and E. M. Petriu. 3-D object model recovery from 2-D images using structured light. In *Instrumentation and Measurement Technology Conference*, Jun 1996.

[26] Jaehong Lee, Heon Gul, Hyungchan Kim, Jungmin Kim, Hyoungrae Kim, and Hakil Kim. Interactive manipulation of 3D objects using Kinect for visualization tools in education. In *Control, Automation and Systems (ICCAS)*, Oct 2013.

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

[28] Maohai Li, Rui Lin, Han Wang, and Hui Xu. An efficient SLAM system only using RGBD sensors. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec 2013.

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

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

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

[32] Kathia Melbouci, Sylvie Naudet Collette, Vincent Gay-Bellile, Omar Ait-Aider, Mathieu Carrier, and Michel Dhome. Bundle adjustment revisited for SLAM with RGBD sensors. In *IAPR International Conference on Machine Vision Applications (MVA)*, May 2015.

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

[34] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, and Andrew J. Davison. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, Oct 2011.

[35] Meenakshi Panwar. Hand gesture recognition based on shape parameters. In *2012 International Conference on Computing, Communication and Applications*, Feb 2012.

[36] Carolina Raposo, Joao Pedro Barreto, and Urbano Nunes. Fast and Accurate Calibration of a Kinect Sensor. In *International Conference on 3D Vision*, Jun. 2013.

[37] Sebastian A. Scherer and Andreas Zell. Efficient onbard RGBD-SLAM for autonomous MAVs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013.

[38] Jan Smisek, Michal Jancosek, and Tomas Pajdla. 3D with Kinect. In *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Nov. 2011.

[39] Aaron N. Staranowicz, Christopher Ray, and Gian-Luca Mariottini. Easy-to-use, general, and accurate multi-Kinect calibration and its application to gait monitoring for fall prediction. In *Engineering in Medicine and Biology Society (EMBC)*, Aug. 2015.

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

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

[42] R. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4), Aug 1987.

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

[44] Jodie Wetherall, Matthew Taylor, and Darren Hurley-Smith. Investigation into the Effects of Transmission-channel Fidelity Loss in RGBD Sensor Data for SLAM. In *International Conference on Systems, Signals and Image Processing (IWSSIP)*, Sep 2015.

[45] Guan xi Xin, Xu tang Zhang, Xi Wang, and Jin Song. A RGBD SLAM algorithm combining ORB with PROSAC for indoor mobile robot. In *International Conference on Computer Science and Network Technology (ICCSNT)*, Dec 2015.

[46] Cheng-Kai Yang, Chen-Chien Hsu, and Yin-Tien Wang. Computationally efficient algorithm for simultaneous localization and mapping (SLAM). In *IEEE International Conference on Networking, Sensing and Control (ICNSC)*, Apr 2013.

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

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

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

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

[51] Xinshuang Zhao, Ahmed M. Naguib, and Sukhan Lee. Kinect based calling gesture recognition for taking order service of elderly care robot. In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, Aug 2014.

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