University of Tartu
Faculty of Science and Technology
Institute of Technology
Computer Engineering Curriculum

Lembit Valgma

3D reconstruction using Kinect v2 camera

Bachelor's thesis (12 ECTP)

Supervisor: Assoc. Prof. Gholamreza Anbarjafari
Morteza Daneshmand, MSc

Tartu 2016

3D mudeli koostamine Kinect v2 kaamera abil

Lühikokkuvõte:

Kinect on kergesti kasutatav ning suhteliselt odav RGB-D kaamera, mis võimaldab lisaks värvipildile salvestada ka infot, kui kaugel vaadeldav objekt kaamerast on. Tänu sellisele funktsionaalsusele on Kinect huvipakkuvaks alternatiiviks tavalistele 3D skanneritele, mille hinnad on enamasti palju kõrgemad. Kinecti uus versioon, Kinect v2, lubab nii värvi kui kauguse infot salvestada oluliselt parema kvaliteediga kui originaal.

Käesolevas bakalaureusetöös kirjeldatakse ning realiseeritakse meetod, mille abil on võimalik Kinect v2 abil salvestatud videost rekonstrueerida eseme 3D mudel. Autor testis meetodi rakendatavust kasutades erineva kuju ning pinnakattega esemeid. Enamikel juhtudel töötas rekonstruktsioon piisavalt hästi ning võib öelda, et Kinect v2 sobib lihtsamate 3D mudelite loomiseks hästi. Kinecti kauguse mõõtmise tehnoloogia seab piirangud rekonstrueeritava eseme pinnakattele, väga peegeldavate või läbipaistvate pindade korral ei suuda Kinect v2 kaugusi korrektselt hinnata. Sümmeetriliste esemete korral ei suuda kirjeldatud meetod leida korrektset teisendust kahe vaatepunkti vahel ning rekonstruktsioon ei ole võimalik.

Võtmesõnad: 3D rekonstruktsioon, ICP, Kinect
CERCS: T111 Pilditehnika

3D reconstruction using Kinect v2 camera

Abstract:

Kinect is an easy to use and affordable RGB-D acquisition device that provides both spatial and color information for captured pixels. That makes it an attractive alternative to regular 3D scanning devices that usually cost significantly more and do not provide color info. Second generation of Kinect (v2) provides even better quality depth and color images to user.

This thesis describes and implements method for 3D reconstruction using Kinect v2. Method suitability for various objects is tested and analyzed. In most circumstances the method provided satisfactory reconstructions unless very high resolution is desired. However some limitation were observed. Reflective and transparent surfaces cause failure due to depth capturing technology in Kinect v2, symmetric objects cause problems for described frame registration algorithm. For better understanding, Kinect v2 depth measuring process is described.

Keywords: 3D reconstruction, ICP, Kinect
CERCS: T111 Imaging, image processing

Contents

List of Figures

Acronyms

ADC	Analog to digital converter
CMOS	Complementary metal–oxide–semiconductor
FOV	Field of view
ICP	Iterative closest point
IR	Infrared
RGB-D camera	Camera cabable of capturing color and depht information
SDK	Software development kit
TOF	Time of flight

1 Introduction

RGB-D cameras combine a regular color video camera with an active depth capturing sensor. For each pixel, they provide both the color information and the measurement of the distance from the camera to the object that is represented by the corresponding pixel. Additionally, they commonly provide quite high frame rates, making it possible to capture moving objects with negligible distortion.

In the recent years, RGB-D cameras have seen significant rise in popularity, mainly due to the emergence of the Microsoft Kinect series [4], which has been a major accessory for their Xbox game consoles. In 2010, Kinect v1 was released, which used the structured light technology to capture the depth information. The second version of Kinect was released in 2014 along with the new Xbox version. The new Kinect uses a modulated light technology for depth measurements. It enables to obtain the depth info with much better resolution and quality while also limiting the interference from outside sources [5, 6].

Compared to most 3D scanners, Kinect is very affordable, which makes it an attractive tool for many researchers and companies interested in 3D modelling. However, Kinect was originally designed for tracking human body movements [7]. Since the tracking has to be done in real time, Kinect has been designed such that it could capture the depth frames with high frequency, being maximum 30 Hz. However, the latter means that there is very little time for accurate measuring to be done. As a result, the Kinect depth info can be considerably noisy or even incomplete [8]. Nevertheless, still researchers have obtained decent results using Kinect as a 3D modelling tool [9, 10, 11].

The aim of this thesis is to devise a practically efficient 3D reconstruction algorithm with a built-in texturing module, which demands evaluating the capability of the Kinect v2 to perform the associated task, i.e. to provide sufficient and reliable depth and color information, as well as understanding and overcoming its limitations. In order to do so, acquiring the input data and combining it into an object representation, must be studied.

The thesis is divided into following topics:

- In Chapter 2, a literature review on the description of the Kinect depth sensor and its working principles are provided, and the modulated wave time of flight and the distance to depth conversion principles are discussed;
- In Chapter 3, the theoretical background to rigid transformation in the 3D space is presented, and the ICP method for registering the frames is described;
- In Chapter 4, a description of all the steps needed for 3D reconstruction with the Kinect is presented, where the output point cloud maintains the original texture of

the object without any computational or algorithmic cost, and the settings associated with the practical implementation are specified;

- In Chapter 5, the experimental 3D reconstruction results obtained using the process described in the previous chapter are provided, and the limitations of the proposed system are discussed;
- Finally, the thesis is concluded through summarizing the main topics covered by the content.

2 Kinect

2.1 General overview

Kinect v2 is a RGB-D acquisition device designed by Microsoft as contact free controller for Xbox One. In addition to color camera it also has depth camera which allows to find out both color and spatial information about filmed scene. Kinect can be connected to computer using USB 3 connection and used as input for 3D modelling task.

Detailed description about Kinect history, its applications and how to obtain usable RGB-D data from Kinect is given in Sandra Demitševa bachelor thesesis [12]. The main focus of current thesis is manipulation of 3D point clouds. Kinect depth sensor is used to obtain the initial point cloud. Following is the description how it works.

2.2 Depth sensor

Kinect v2 has three infrared light sources each generating a modulated wave with different amplitude. In order to capture reflected waves, Kinect also has infrared camera. Location of lasers and sensors is shown in figure 2.1.

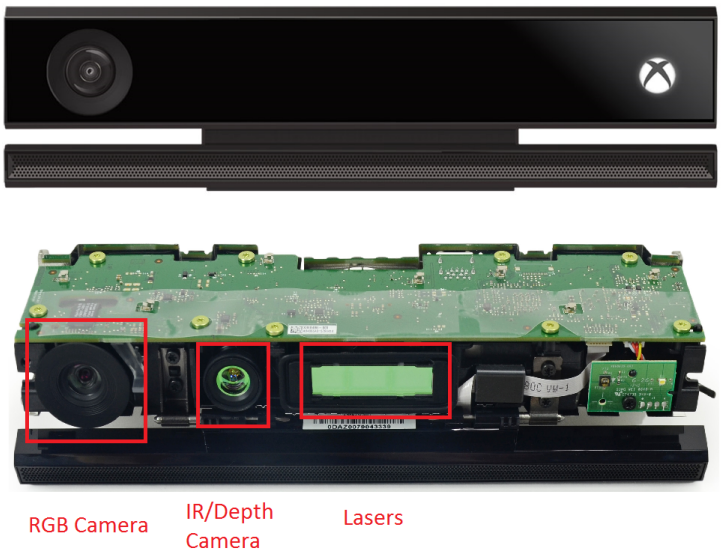


Figure 2.1: Location of Kinect sensors.

2.2.1 Sensor specifications

The infrared sensor in Kinect v2 is state of the art 512×424 CMOS array of differential pixels. Sensor performance parameters are listed in figure 2.2. Each pixel has two photo diodes (A, B) being controlled by the same clock signal that controls wave modulation. Photo diodes convert captured light into current which can be measured. The diodes are driven by the clock signal such that if $A = [a_i]$ is turned on, $B = [b_i]$ is turned off, and *vice versa*. Then, according to [7]

- $([a_i] - [b_i])$ shows correlation between retrieved light and the clock signal and can be used to obtain phase information ("depth image");
- $([a_i] + [b_i])$ gives regular grayscale image illuminated by normal ambient lighting ("ambient image");
- $\sqrt{\sum_i ([a_i] - [b_i])^2}$ gives grayscale image that is independent of ambient lighting ("active image").

Process Technology	TSMC 0.13 1P5M
Pixel Pitch	10u*10u
Pixel Array	512*424Pixels
Chip size	8.2mm*14.2mm
System Dynamic Range	> 2500 = 68db
Modulation Contrast	68% @ 860nm @50Mhz
Modulation Frequency	10-130Mhz
Average Modulation Frequency	80Mhz
FOV	70 (H) X 60 (V) degrees
Depth Uncertainty	< 0.5% of range
Distance Range	0.8-4.2m
Operating Wavelength	860nm
Frame Rate	max 60fps (typical 30fps)
ADC	2GS/s
Effective Fill Factor	60%
Reflectivity	15%-95%
Chip Power	2.1W
Responsivity @ 860nm	0.144 A/W
Readout Noise	320 uV differential
F#	1.07
ADC Resolution	10

Figure 2.2: Depth sensor performance parameters [1].

Narrowband-pass filter is used to block all light except in the 860 nm wavelength range that corresponds to infrared illumination system wavelength.

Kinect uses also multi-shutter engine that merges data from multiple shutters and chooses the best shutter value for each pixel. Longest shutter time that does not cause saturation is used. Engine also normalizes all values relative to the longest shutter time.

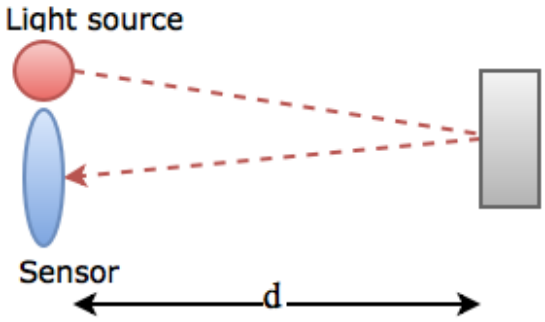


Figure 2.3: Time-of-flight operation principle.

2.2.2 Measuring distance

Kinect v2 uses optical time-of-flight (TOF) technology for measuring distances [13], [7]. The operation principle in TOF device is based on measuring the time it takes for light wave to travel from emitter to object and back to sensor. Let d be distance from the sensor, then the simplest case can be expressed as

$$d = \frac{t_r - t_e}{2} \cdot c, \tag{2.1}$$

where t_e and t_r represent time for light pulse emitting and receiving, c is speed of light in air. Measuring single light pulses is not very practical for scene capturing devices like Kinect. Hence, the amplitude modulated infrared light is used with CMOS array receiver. Distance is calculated based on the phase difference of emitted light wave and the detected light wave reflected from the object. Let the transmitted wave

$$T(t) = \sin(\omega t) \tag{2.2}$$

have modulation frequency $\omega = 2\pi f$. Then, distance traversed by modulated wave is $2d$ which produces phase shift φ. Received wave

$$R(t) = \beta \sin(\omega t - \varphi) \tag{2.3}$$

also has modified amplitude which depends on many factors. However, amplitude is not needed for measuring distance, so it can be discarded. Phase shift depends on time difference

$$\varphi = \omega t_r - \omega t_e = \omega(t_r - t_e). \tag{2.4}$$

Subsidising 2.1 results in

$$\varphi = \frac{2d}{c}\omega, \tag{2.5}$$

which implies

$$d = \frac{\varphi c}{2\omega}. \tag{2.6}$$

In order to obtain the phase shift, at least two measurements are needed (as there are two unknowns in 2.3). For that, received signal $\beta \sin(\omega t - \varphi)$ is mixed with a phase-delayed version of the reference signal

$$R_o(t) = \sin(\omega t - \varphi_{offset}). \tag{2.7}$$

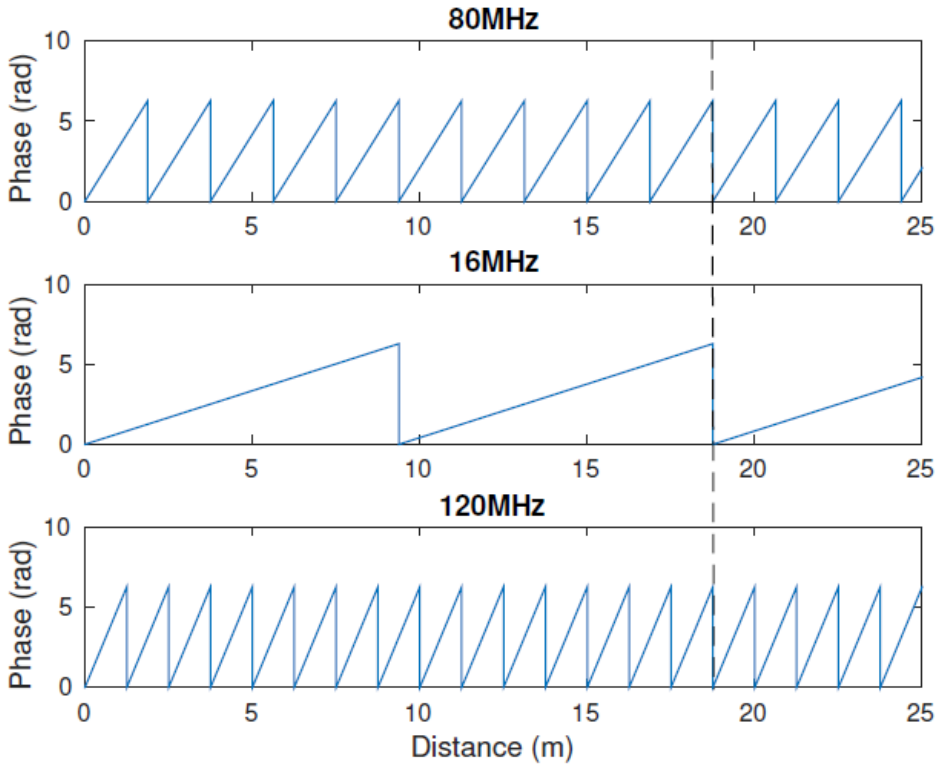


Figure 2.4: Wrapped phases and distances for 80MHz, 16MHz and 120MHz modulated waves used in Kinect distance calcuation [2]. Dashed line shows the common wrap around at 18.75 meters.

This results in

$$\begin{aligned} R(t) \cdot R_o(t) &= \beta \sin(\omega t - \varphi) \cdot \sin(\omega t - \varphi_{offset}) \\ &= 0.5\beta \left(\cos(\omega t - \varphi - \omega t + \varphi_{offset}) - \cos(\omega t - \varphi + \omega t - \varphi_{offset})\right) \\ &= 0.5\beta \cos(\varphi - \varphi_{offset}) - 0.5\beta \cos(2\omega t - \varphi - \varphi_{offset}). \end{aligned} \tag{2.8}$$

The result is processed by low-pass filter, which removes the second addend, yielding

$$LP\left[R(t) \cdot R_o(t)\right] = 0.5\beta \cos(\varphi - \varphi_{offset}). \tag{2.9}$$

Using different values for φ_{offset}, the phase-shift φ can be estimated independent of signal amplitude β. Kinect sensor uses three different phase-shifts of 0°, 120°and 240° [14].

Since measuring the distance is based on phase shift of the modulated wave, the maximum uniquely measurable distance depends on the wavelength of the modulated wave. Phase wraps around at 360°(2π). This would imply that using longer wavelengths would allow for measuring longer distances. Using shorter wavelengths gives better resolution [13]. In order to enable good resolution and also measuring longer distances, Kinect uses three different frequencies of 120 MHz, 80 MHz and 16 MHz [7]. Common wrap around for these frequencies occurs at 18.75 meters, which is the also the maximum distance where depth can be uniquely identified. Figure 2.4 shows the relation between phase and distance for used frequencies.

Given three frequencies $f_0 = 80\text{MHz}$, $f_1 = 16\text{MHz}$, $f_2 = 120\text{MHz}$ and respective phase

shifts φ_0, φ_1, φ_2, according to 2.6, the distance d must satisfy

$$d = \frac{c}{4\pi f_0}(\varphi_0 + 2\pi n_0) = \frac{c}{4\pi f_1}(\varphi_1 + 2\pi n_1) = \frac{c}{4\pi f_2}(\varphi_2 + 2\pi n_2), \tag{2.10}$$

where $0 \leq n_0 \leq 9$, $0 \leq n_1 \leq 1$ and $0 \leq n_2 \leq 14$ are unknown wrapping coefficients. Substituting frequencies gives

$$\begin{aligned} \frac{\varphi_0 + 2\pi n_0}{4\pi \cdot 10} = \frac{\varphi_1 + 2\pi n_1}{4\pi \cdot 2} = \frac{\varphi_2 + 2\pi n_2}{4\pi \cdot 15} \Leftrightarrow \\ \frac{3(\varphi_0 + 2\pi n_0)}{2\pi} = \frac{15(\varphi_1 + 2\pi n_1)}{2\pi} = \frac{2(\varphi_2 + 2\pi n_2)}{2\pi} \Leftrightarrow \\ \frac{3\varphi_0}{2\pi} + 3n_0 = \frac{15\varphi_1}{2\pi} + 15n_1 = \frac{2\varphi_2}{2\pi} + 2n_2. \end{aligned} \tag{2.11}$$

By denoting

$$t_0 = \tfrac{3\varphi_0}{2\pi}, \quad t_1 = \tfrac{15\varphi_1}{2\pi}, \quad t_2 = \tfrac{2\varphi_2}{2\pi} \tag{2.12}$$

the following system of equations for finding n_0, n_1 and n_2 can be obtained

$$\begin{aligned} 3n_0 - 15n_1 &= t_1 - t_0, \\ 3n_0 - 2n_2 &= t_2 - t_0, \\ 15n_1 - 2n_2 &= t_2 - t1. \end{aligned} \tag{2.13}$$

Kinect v2 sensor actually outputs the phase shift values, giving researchers the opportunity to use various methods for actual depth map estimation. In current thesis, the default implementation from Microsoft SDK [15] is used that produces depth in millimeters for each pixel.

2.3 Transformation from depth to 3D coordinates

Time-of-flight sensor described in the previous section provides for each sensor pixel a distance measure from the center of the camera. In order to model the real space and objects, true 3D coordinates for the points are desired. This can be obtained using the standard pinhole camera model [3] and doing the calculation in reverse order. First, the depth map calculation from distance map is presented, then regular pinhole camera model is presented.

2.3.1 Distance to depth

Given point P, its distance from the camera center d, focal length f and distance from principal point to point P projection on image plane x, distance z from camera center C to point P_c must be calculated. First distance from C to P projection on image plane can be calculated

$$l = \sqrt{f^2 + x^2}. \tag{2.14}$$

Then, based on the similar triangles property

$$\frac{z}{d} = \frac{f}{l}, \tag{2.15}$$

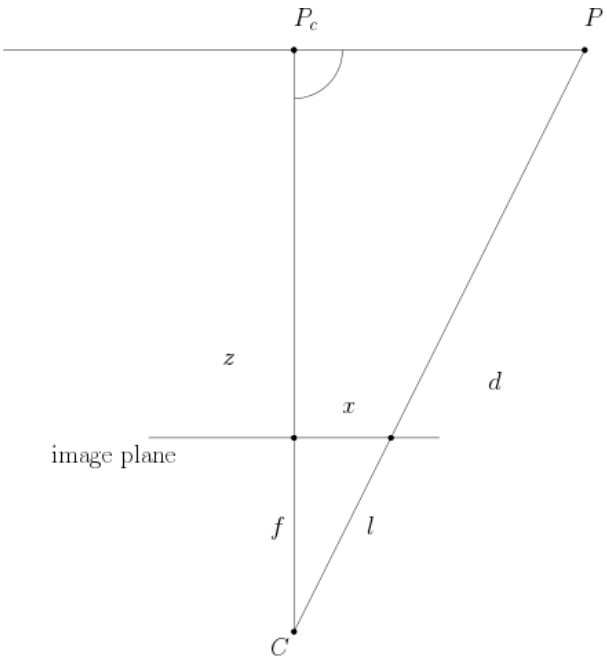


Figure 2.5: Calculation of depth from distance information

from which it can be written that

$$z = d\frac{f}{l} = d\frac{f}{\sqrt{f^2 + x^2}}. \tag{2.16}$$

Kinect SDK outputs the depth values already in the depth map format. It means that, instead of the distance from camera center to point, each pixel depth value represents the distance to the plane that has the corresponding point and is perpendicular to camera principal axis.

2.3.2 Pinhole camera model

Let the center of the coordinate system be at the center of camera, f is the distance between center of projection and the image plane (focal length of camera), line starting from center of projection and perpendicular to the image plane is called principal axis, principal point is the intersection of principal axis and image plane. Let $P(X, Y, Z)$ be 3D coordinates of a model point, $p(x, y)$ corresponding coordinates on the camera image plane (in same units as 3D coordinates). By the similar triangles property, image plane coordinates can be written as

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}. \tag{2.17}$$

Now x and y are described in real world coordinates, on the image the coordinates are however in pixels. Transforming the coordinates requires knowing the column-wise and row-wise density of the pixels (pixels per millimeter), let them be k_u and k_v respectively. The principal point distance from the pixel coordinate origin is also required, let its coordinates be $(-x_0, -y_0)$. Then point p coordinates in pixel coordinate system can be

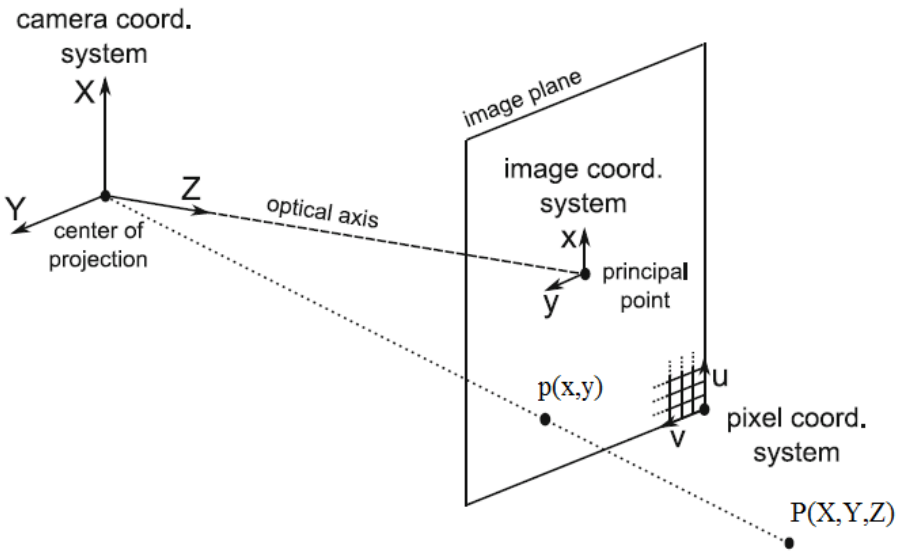


Figure 2.6: Pinhole camera model [3].

written as

$$\begin{aligned} u &= k_u(x + x_0) = k_u f\frac{X}{Z} + k_u x_0, \\ v &= k_v(y + y_0) = k_v f\frac{Y}{Z} + k_v y_0. \end{aligned} \tag{2.18}$$

After normalising point P coordinates by dividing it with its Z coordinate

$$P' = \frac{P}{Z} = \begin{pmatrix} X/Z \\ Y/Z \\ 1 \end{pmatrix} = \begin{pmatrix} X' \\ Y' \\ 1 \end{pmatrix},$$

formula 2.18 can be written in matrix form

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} k_u f & 0 & k_u x_0 \\ 0 & k_v f & k_v y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X' \\ Y' \\ 1 \end{pmatrix} = KP'. \tag{2.19}$$

The four values in the matrix are called *camera intrinsic parameters* and usu denoted as

$$\alpha_u = k_u f,\ \alpha_v = k_v f,\ u_0 = k_u x_0,\ v_0 = k_v y_0. \tag{2.20}$$

In pixels these represent focal length (α_u, α_v) and coordinates of the principal point (u_0, v_0). Camera intrinsics matrix K can be represented as

$$\begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.21}$$

In general, camera intrinsics for given camera are known or they can be found out by calibration. For the Kinect, they can be obtained from the SDK [16]. Considering that

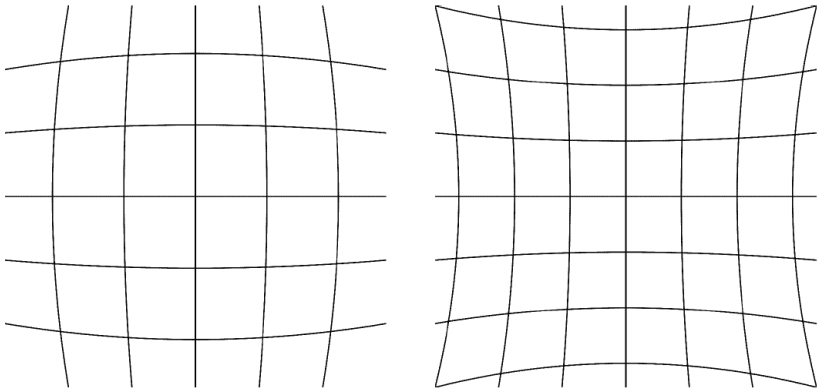


Figure 2.7: Common lens distortions: barrel (left), pincushion (right).

u, v, K and Z are known for point P, based on the previous discussion, the x- and $y-$coordinates can be obtained

$$K^{-1}\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = P' = \frac{P}{Z}, \tag{2.22}$$

from which

$$ZK^{-1}\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}. \tag{2.23}$$

Since Z is known, X and Y can be calculated for all u and v. Kinect SDK provides a convenient table of (γ_u, γ_v) values for each pixel (u, v), such that

$$\begin{aligned} X &= \gamma_u \cdot Z, \\ Y &= \gamma_v \cdot Z. \end{aligned} \tag{2.24}$$

2.3.3 Radial distortion

The discussion in the previous section was about the perfect pinhole camera model. In real life the lens of the camera causes some distortion of the points being modeled on image plane (Figure 2.7).

There are many types of distortions but it has been found that the most common and significant is radial distortion. Most of the distortion can be therefore be described by relative simple model [17]. Let (x, y) be the ideal (distortion free) coordinates on image plane, and (x_d, y_d) the corresponding real observed coordinates. The principal point, $(0, 0)$ is assumed to be distortion free under this model. Then

$$\begin{aligned} x_d &= x \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \\ y_d &= y \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \end{aligned} \tag{2.25}$$

where

$$r = \sqrt{x^2 + y^2} \tag{2.26}$$

and k_1, k_2, k_3 are radial distortion coefficients. The number of addends in the formula 2.25 is not limited in general model, however Zhang [17] claims that two is enough in most cases Kinect SDK provides access to first three coefficients [16]. There seems to be little evidence if SDK uses distortion correction or not.

2.4 Alignment of depth and color sensor

Since depth and color information is captured by different sensors, the mapping between RGB and depth sensor is required. The RGB camera can be modelled as pinhole camera, similarly to depth camera described previously. Let K_c be camera intrinsics matrix for RGB camera, then similarly to 2.23, it can be written

$$Z_c K_c^{-1} \begin{pmatrix} u_c \\ v_c \\ 1 \end{pmatrix} = \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}, \tag{2.27}$$

where X_c, Y_c and Z_c are point P coordinates in RGB camera coordinate system. Since cameras are fixed, there exists a rigid transformation T from depth coordinates to RGB coordinates such that

$$T \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}. \tag{2.28}$$

Substituting values from 2.23 and 2.27 this can be written as

$$TZK^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = Z_c K_c^{-1} \begin{pmatrix} u_c \\ v_c \\ 1 \end{pmatrix}. \tag{2.29}$$

Here, all parameters are known or can be obtained by calibration [18]. Formula 2.29 describes mapping between depth and color camera. Kinect SDK provides convenient mapping functions in both directions. Important to note is that the mapping relation relies on the fact that both sensors can see the point being represented by image. Due to different locations of the sensors and also different field-of-view, there are regions where the direct mapping is not possible, as can be seen from Figure 2.8.

2.5 Challenges

Theory described above works in ideal conditions. In real operation there are several factors that can cause errors in depth measurement. Sarbolandi et al. [14] analyzed several error sources and compared effect on two generations of Kinects. Following are sources that have larger effect on Kinect v2 that uses time of flight technology.

Temperature drift

Since time of flight technology requires quite high illumination power to cover the whole scene, it produces quite a lot of heat and even require active cooling. Depth sensor output

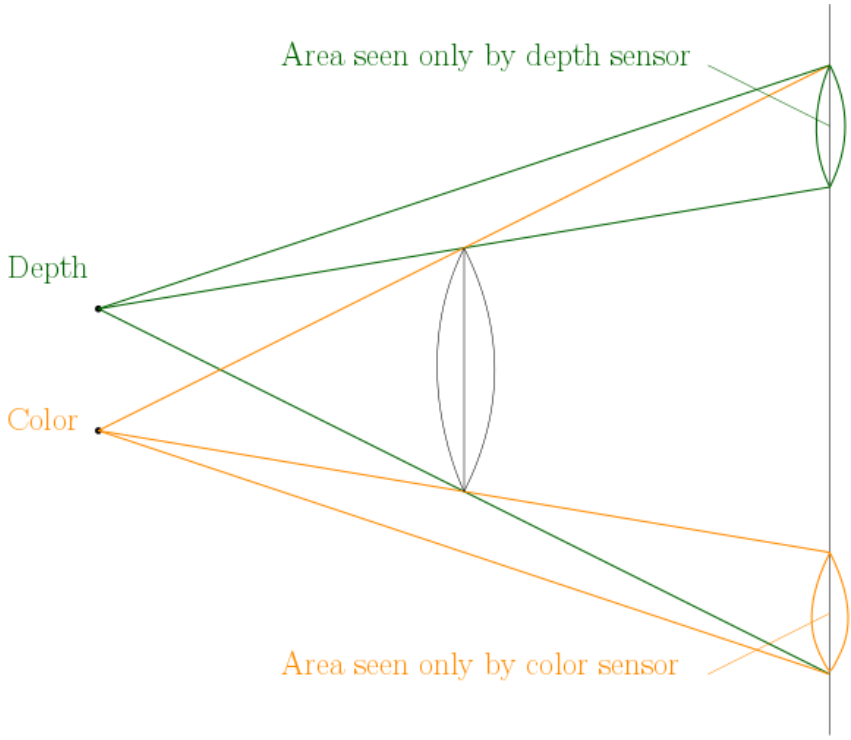


Figure 2.8: Depth and color camera "blind spots".

change somewhat during the warming process for some time after it has been turned on. Output stabilises in about 30 minutes.

Systematic distance error

Approximations in the sinusoidal signal shape lead to some systematic depth error measurements in Kinect sensor. The magnitude of the systematic error has been show to be relatively small, in the order of 1-2 mm [5].

Depth inhomogeneity

At object boundaries pixels can have inconsistent depth values. This is because some of the light reflected is obtained from object but some from the background, mixing that information together can produce so called *flying pixels* which show object boundaries at incorrect coordinates.

Multi-path effects

Since time of flight measuring principle relies on capturing light reflected back from the object surface, it can happen that light does not travel directly from illumination unit to object and back to sensor but instead takes indirect path being reflected from several surfaces. When such info is captured by sensor the depth measurement combines several waves and gives incorrect results. For very reflective surfaces, if it is positioned

at relatively flat angle towards camera, no light might be reflected back and there is no depth info about that surface.

Semitransparent and scattering media

When surface, such as glass, only reflects part of the light directly and some other part is reflected from within the object, there is additional phase shift and the depth measurement is incorrect.

Dynamic scenery

Important assumption in Kinect depth measuring system is that each pixel observes a single object point during the whole acquisition process. When objects are moving, that assumption is violated. Since Kinect v2 takes several measurements to produce depth value for pixel, the movement can cause incorrect depth measurements, specially at the object borders.

3 Point cloud registration

Kinect captures sequence on snapshots (set of points in three dimensional space, point clouds) of area within the camera frame. Moving Kinect around the stationary object or rotating the object on the turntable in front of Kinect, allows capturing the entire surface of the object. In order to reconstruct the entire surface, individual views must be combined. Each view of the object is slightly transformed compared to previous and for combining the point clouds, relative transformation between two views needs to be found. Once the transformation is found, point clouds can be aligned with each other. The process of finding proper transformation is called point cloud registration.
Repeating this process for all the viewpoints allows to align all the point clouds and thus recreate the entire object surface.

There are various algorithms for point cloud registration. They can differ in terms of available input data, if rough estimation of the transformation is known or not, if color information is given. Methods also vary by the kind of correspondences used, motion calculation method, robustness and registration strategy. Salvi et al. describe more common methods [19] and analyse the performance.

Since Kinect provides high frequency snapshots, the relative motion between frames is very small, allowing to roughly estimate the transformation. In such situation, the most commonly used algorithm is Iterative Closest Point (ICP), which finds the registration by minimizing the the distance between point-correspondences, known as closest point.

There are also registration methods that use color information only or in addition to range information. They usually use some key-points in the frame and various point descriptors, most commonly SIFT [11] and FAST [10]. These key-points are highly distinctive and provide good initial estimate for transformation. Because of that they are good for scene reconstruction but not as needed for simpler object reconstruction described in the current thesis. Since these methods are computationally more costly and don't provide a lot of benefit, color based methods are not used in this work.

In the following, point cloud and transformation representation is described. After that, ICP process is described.

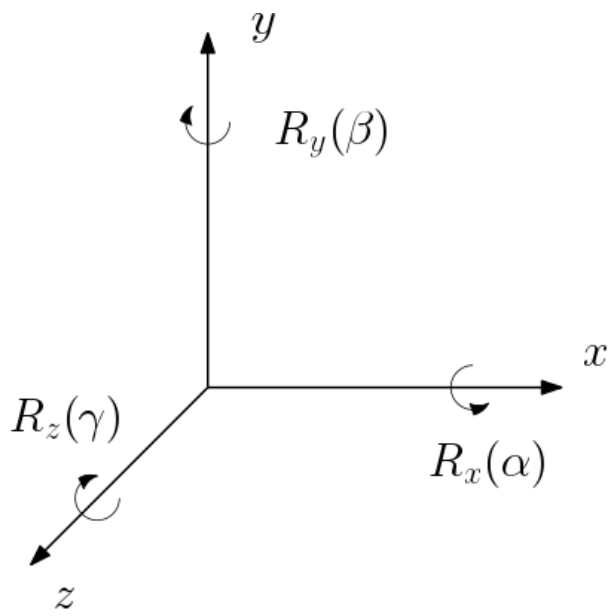


Figure 3.1: Rotation around x-axis (α), y-axis (β) and z-axis (γ).

3.1 Transformation

3.1.1 Point representation

Single point in three dimensional space is represented by its coordinates

$$p_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}. \tag{3.1}$$

Set of related points is called point cloud

$$P = \{p_i \mid i = 1, ..., N\}, \tag{3.2}$$

where N is the number of points.

3.1.2 Rotation representation

Rotation matrices and order

Rotation can be expressed by three independent rotations around fixed axis using rotation matrices

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix},$$

$$R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}, \tag{3.3}$$

$$R_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

In order to acquire full rotation matrix, individual rotations must be applied one at the time. Since the angles depend on the order of applying the rotations, the order should be fixed. In current thesis, the following notation is used

$$R = R(\alpha, \beta, \gamma) = R_z(\gamma)R_y(\beta)R_x(\alpha). \tag{3.4}$$

$$R(\alpha, \beta, \gamma) = \begin{pmatrix} \cos\beta\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \cos\beta\sin\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{pmatrix} \tag{3.5}$$

For small angles, linearisation can be done using $\cos\theta \approx 1$, $\sin^2\theta \approx 0$ and $\sin\theta \approx \theta$. Then rotation matrix can be approximated by

$$R(\alpha, \beta, \gamma) = \begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix}. \tag{3.6}$$

Axis angle

Rotation can be expressed also by determining the rotation axis and the rotation amount around that axis. The axis is usually defined by unit vector, $\vec{e}$, and the rotation amount by angle θ. Then rotation can be written as a pair

$$\langle \vec{e}, \theta \rangle = \left((e_x, e_y, e_z)^T, \theta \right). \tag{3.7}$$

Quaternions

Quaternion is defined as four dimensional complex number

$$q = q_0 + q_1 i + q_2 j + q_3 k, \tag{3.8}$$

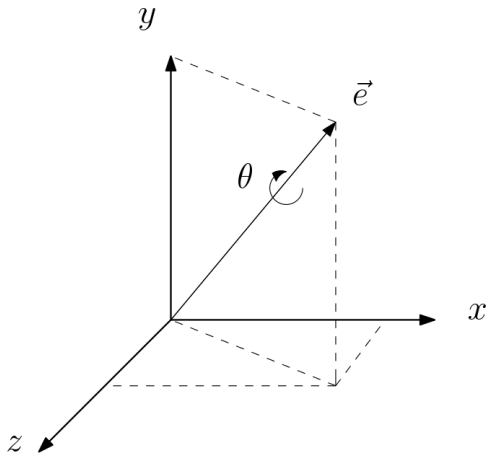


Figure 3.2: Rotation representation using axis $\vec{e}$ and angle θ.

where $q_0, q_1, q_2, q_3 \in \mathbb{R}$ and

$$i^2 = j^2 = k^2 = ijk = -1. \tag{3.9}$$

Quaternion conjugate is defined as

$$q^* = q_0 - q_1 i - q_2 j - q_3 k. \tag{3.10}$$

Quaternion magnitude is defined as

$$||q|| = \sqrt{qq^*} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}. \tag{3.11}$$

Unit quaternion is a quaternion with magnitude 1

$$||q|| = 1. \tag{3.12}$$

Reciprocal or inverse of quaternion is defined as

$$q^{-1} = \frac{q^*}{||q||^2}, \tag{3.13}$$

which for unit quaternions implies

$$q^{-1} = q^*. \tag{3.14}$$

Unit quaternions can be used to represent rotation. Given axis angle representation $\left((e_x, e_y, e_z)^T, \theta\right)$, the corresponding unit quaternion is expressed as

$$q = \cos\frac{\theta}{2} + e_x \sin\frac{\theta}{2} i + e_y \sin\frac{\theta}{2} j + e_z \sin\frac{\theta}{2} k. \tag{3.15}$$

For point $p = (x, y, z)^T$ construct respective quaternion by

$$p_q = 0 + xi + yj + zk. \tag{3.16}$$

Then rotation result $p' = (x', y', z')^T$ can be computed by

$$p'_q = q p_q q^{-1}, \tag{3.17}$$

where $p'_q = 0 + x'i + y'j + z'k$.
Alternatively, the rotation represented by unit quaternion $q = q_0 + q_1 i + q_2 j + q_3 k$ can be expressed as rotation matrix by

$$R(q) = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} \tag{3.18}$$

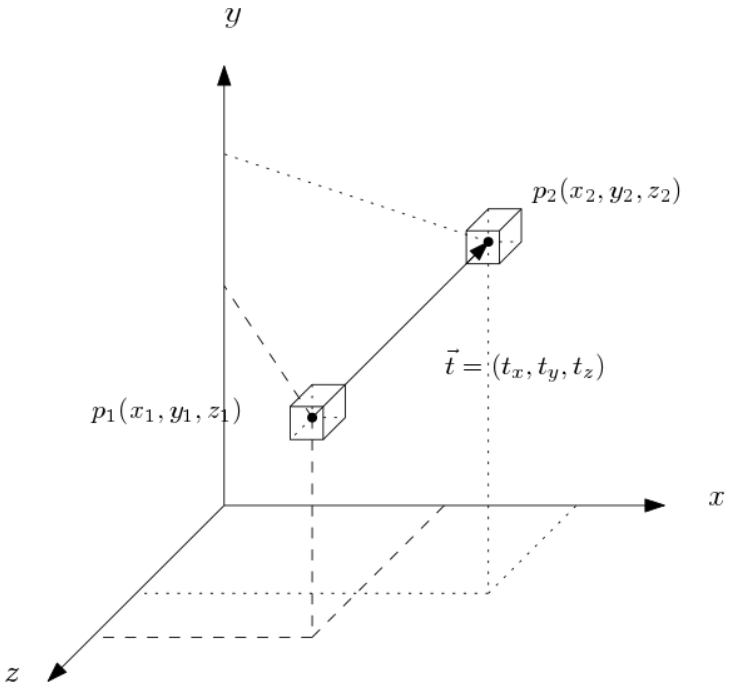


Figure 3.3: Translation from P_1 to P_2.

3.1.3 Translation

Translation in three dimensional space can be expressed as vector

$$\vec{t} = (t_x, t_y, t_z), \tag{3.19}$$

where t_x, t_y, t_z show the movement along x-, y- and z-axis respectively. Then coordinates after translation can be written as

$$p_2 = p_1 + \vec{t}. \tag{3.20}$$

3.1.4 Rigid transformation

If all the points in the object have the same distance between each other before and after the transformation then the transformation is called rigid. In this thesis only rigid transformations consisting of rotation and translation are considered. Let R be rotation and $\vec{t}$ translation applied to point p_i, then the resulting point m_i can be expressed as

$$m_i = Rp_i + \vec{t}. \tag{3.21}$$

3.1.5 Homogeneous coordinates and transformation matrix

Homogeneous coordinates can be used to express point coordinates

$$p_i = \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \tag{3.22}$$

and augmented transformation matrix

$$T = \begin{pmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_x \\ R_{2,1} & R_{2,2} & R_{2,3} & t_y \\ R_{3,1} & R_{3,2} & R_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.23}$$

Using the above notation, formula 3.21 can be written as

$$m_i = Tp_i. \tag{3.24}$$

3.2 Iterative closest point

The task of the registration algorithm is to find transformation T that would be applied to one of the point clouds in order to bring two point clouds as close to each other as possible. Usually the points that transformation is applied to, are called data points, and the other point cloud is called model points. In the following, let data points be denoted by $P = \{p_i \mid i = 1, ..., N\}$ and model points $M = \{m_i \mid i = 1, ..., N\}$. The goal can then be mathematically described as minimizing some metric e, where

$$e = f(M, TP). \tag{3.25}$$

The most common metric to be used is sum of squares. Given point correspondence between p_i and m_i, e can be defined as

$$e = \frac{1}{N} \sum_{i=1}^{N} \|m_i - Tp_i\|^2. \tag{3.26}$$

Since in the current setup, the shape of the object does not change, only rotation and translation must be considered. This allows to express the minimization metric by rotation and translation

$$e = \frac{1}{N} \sum_{i=1}^{N} \left\|m_i - Rp_i - \vec{t}\right\|^2. \tag{3.27}$$

Rotation has three parameters and translation has another three, in total there are six parameters to be estimated. Given at least three corresponding points, the parameters can be estimated. Horn [20] has shown a closed form solution for the minimization problem in 3.27.

Approach described above is usable if there is known correspondence between points, i.e. for each $i = 1, ..., N$ it is known which p_i is transformed to which m_i. Point clouds acquired by Kinect do not give such correspondence.

Iterative closest point (ICP) algorithm proposed by Besl and McKay [21] solves this problem by using closest points before the transformation as initial approximation, finding transformation and applying found transformation on data points. Process is iterated until convergence, using transformed data points at each step as input for calculating closest points.

3.2.1 ICP algorithm steps

1. Data points P and model points M are given.
2. Initialize iteration by setting $P_0 = P$, $R = I$, $\vec{t} = (0, 0, 0)$ and $k = 0$. If there is some initial estimation for rotation and translation from some other source, use those for initialization. Steps 3-7 are repeated until convergence within some tolerance τ.
3. Compute closest points Y_k for P_k in M.
4. Find optimal transformation T_k between P_0 and Y_k.
5. Apply the transformation $P_{k+1} = T_k P_0$.
6. Find error e_k between Y_k and M. Terminate iteration if change in error falls below threshold, i.e. if $e_k - e_{k+1} < \tau$.

Besl and McKay proved that ICP algorithm always converges monotonically to a local minimum with respect to the mean-square distance objective function. If the transformation is small, local minimum is very likely also global minimum.

3.2.2 Corresponding point set registration

Given corresponding point clouds $P = p_i$ and $M = m_i$ the goal is to find rotation R and translation $\vec{t}$ such that metric 3.27 would be minimized.
Denote center of mass μ_p for data points P and μ_m for model points M, which can be calculated as

$$\mu_p = \frac{1}{N}\sum_{i=1}^{N} p_i, \quad \mu_m = \frac{1}{N}\sum_{i=1}^{N} m_i. \tag{3.28}$$

The cross-covariance matrix for P and M is defined as

$$\Sigma_{pm} = \frac{1}{N}\sum_{i=1}^{N}\left[(p_i - \mu_p)\,(m_i - \mu_m)^T\right] = \frac{1}{N}\sum_{i=1}^{N}\left[p_i m_i^T\right] - \mu_p \mu_m^T. \tag{3.29}$$

Symmetric 4 x 4 matrix is formed

$$Q(\Sigma_{pm}) = \begin{pmatrix} \mathrm{trace}(\Sigma_{pm}) & \Delta^T \\ \Delta & \Sigma_{pm} + \Sigma_{pm}^T - \mathrm{trace}(\Sigma_{pm}) I_3 \end{pmatrix}, \tag{3.30}$$

where

$$\Delta = \begin{pmatrix} A_{2,3} \\ A_{3,1} \\ A_{1,2} \end{pmatrix}, \quad A = \Sigma_{pm} - \Sigma_{pm}^T. \tag{3.31}$$

Unit eigenvector $q = (q_0, q_1, q_2, q_3)^T$ corresponding to the maximum eigenvalue in matrix $Q(\Sigma_{pm})$ is selected as quaternion representation of the optimal rotation. The rotation matrix $R(q)$ can be constructed using 3.18. Optimal translation vector can be found using rotation matrix and mass centers

$$\vec{t} = \mu_m - R(q)\mu_p. \tag{3.32}$$

3.2.3 Distance to tangent plane

Although ICP algorithm described above is shown to converge, its result is sensitive to noise present in point measurements and also to non-overlapping regions in point clouds. At the same time as Besl and McKay presented their algorithm, Chen and Medioni introduced their own variant of ICP [22]. They used distance from point to tangent plane in other point cloud for point matching and minimization target. In general the process is same as described in 3.2.1 with metric e being minimized defined as

$$e = \sum_{i=1}^{N} \left[\left(m_i - Rp_i - \vec{t} \right) \cdot \vec{n_i} \right]^2, \tag{3.33}$$

where n_i is the estimated tangent normal for the i-th model point. In order to find matching points at each iteration, distance to tangent planes in model point cloud must be minimized.
In general, there is no closed form solution for finding the optimal transformation that would lead to convergence. Hence a least squares method must be used in order to solve it. Analytical solution exists if rotation matrix is linearized as described in 3.6. This can be done for small rotation angles. Closed form solution is described by Low [23].

3.2.4 ICP variants

Due to its usefulness and popularity there has been a lot of research regarding ICP and its suitability in different scenarios. Many improvements and modifications to the various steps of the algorithm have been proposed. Rusinkiewicz and Levoy [24] classified ICP modifications as affecting six stages of the algorithm.

1. Selection of points from input data

Instead of using all the data and model points, some sub-section of points might be used. It might be useful to remove some outliers and noise from point clouds to improve correct registration. In order to speed up calculations a random or uniform sampling might be used. Point selection might also be restricted to points that are more distinctive, either using higher intensity gradient and some color information, if available.

2. Matching these points to samples in the other point cloud

Various metrics can be used to find matching points. Square distance and distance to tangent plane were already described. Other possible variants include projection of data points onto model point surface and then searching within smaller range. Finding matching points is the most computationally costly step in ICP, however it can be speeded up by using efficient algorithms, like kD trees.

3. Weighting the corresponding pairs appropriately

Different weights could be given to point pairs depending on some metric. Lower weight could be applied to points with higher point-to-point distances. Normals compatibility could be used or metric based on the uncertainty of imaging device.

4. Rejecting certain pairs based on looking at each pair individually or considering the entire set of pairs

This step allows to effectively remove outliers, where matching hasn't worked or couldn't work. Some percentage of worst pairs could be rejected based on some metric, 10 % is suggested. Other methods include rejection of point pairs including point that lies on point cloud boundary. However, that requires creating mesh from the point cloud. Also, preserving the distance between points within point clouds could be used as criteria, i.e. if distance between two points in data points differs significantly from matching points in model points, these points are discarded.

5. Assigning an error metric based on the point pairs

Most commonly used are the two metrics described earlier. Sum of squared distances between corresponding points is also known as *point-to-point* metric. Sum of squared distance from point to corresponding tangent plane is known as *point-to-plane* distance. Generally, *point-to-plane* requires more computation for one iteration but converges in less iterations and is less sensitive to noise and non-overlapping regions.

6. Minimizing the error metric

Point-to-point metric has closed form solution, as has linearized *point-to-plane*. In general case some non-linear method (e.g. Levenberg–Marquardt) must be used for solving the problem.

3.3 Global alignment

ICP allows to find pairwise registration of captured point clouds. The goal however is to align all the point clouds in the sequence. This can be done using cumulative transformations. First, a reference point cloud must be chosen, let it be P_0. Then, given sequence of point clouds P_i, ICP finds transformations T_i such that

$$P_{i-1} \approx T_i P_i, \quad i = 1, ..., N. \tag{3.34}$$

Applying transformations iteratively, the cumulative transformations TC_i can be found

$$TC_i = T_i \cdot TC_{i-1}, \quad i = 1, ..., N, \tag{3.35}$$

where $TC_0 = I$. Cumulative transformation matrices can be used to bring all point clouds into the same coordinate reference

$$Q_i = TC_i \cdot P_i, \quad i = 0, ..., N. \tag{3.36}$$

Finally reconstructed point cloud Q is a union of all aligned point clouds

$$Q = \bigcup_{i=0}^{N} Q_i. \tag{3.37}$$

3.3.1 Accumulation error

Since registration is done pairwise and transformations calculated cumulatively, small errors in the individual transformations can cause significant errors in the global alignment. The problem of adjusting global reconstruction to produce jointly optimal structure and parameter adjustment, is known as bundle adjustment problem. Thorough and comprehensive treatment was given by Triggs et al. [25].
The setup in current thesis generally consists of too short sequences for global adjustment to be necessary. However, if required, simple error distribution is done based on assumption that full rotation of the object should result in identity transformation.

4 Implementation

This chapter describes the setup and implementation steps for 3D reconstruction with Kinect v2.

4.1 Setup

Kinect can capture info about the object from one direction at once and in order to reconstruct the full object, different views around the object need to be captured. It can be achieved by moving the camera around the object or rotation the object in front of a fixed camera. The former method is somewhat inconvenient due to Kinect requiring connection to outlet. Also, moving the camera causes it to shake and that can create some artifacts when capturing depth information. Because of these reasons, fixed camera setup was chosen.

Kinect v2 camera was fixed on a tripod with approximate distance from the centre of turntable of 0.8 meters. Turntable itself was white and its diameter was 0.6 meters. Object was rotated for full rotation while Kinect captured frames with frequency from 10 Hz to 30 Hz. Turntable made full rotation in about 30 seconds, which means that maximum rotation between consecutive images was less than two degrees. That frequency proved to be more than enough for reconstruction and for avoiding incorrect local minimums. During testing, it seemed that using one frame after every 10 degree rotation did not cause any problems for reconstruction while providing good speed-up for the process.

Objects chosen were of small to medium size, not exceeding 1 meter in any dimension. Camera was placed on the optimal distance from turntable in order to obtain good depth information. Since Kinect depth measuring range starts from 0.5 meters, camera distance from closest point of the object had to be at least 0.5 meters. Since Kinect has a relatively wide field of view, such distance meant that smaller objects represented a relatively low percentage of capture frame, causing number of object points acquired to be too small for reconstruction. Objects with dimensions less than 0.05 meters were discarded.

Final aspect to consider was camera orientation towards object. It was important that, after rotation, most of the object surface would be captured. Secondly, the possibility of segmenting out the object from background was required. Based on these criteria, camera was placed perpendicular to rotation axis while trying to align the center of object and camera. For obvious reasons, object surface resting against the turntable could not be captured.

4.2 Method implementation

After filming the object the task is to combine consecutive views into single point cloud. For that, first frame is fixed as reference which coordinate system is used. All the other frames are transformed to that coordinate system. Transformation can be calculated iteratively using ICP and alignment process described in 3.3. Initially, first frame is loaded and global transformation matrix T_{accum} created. In the beginning $T_{accum} = I$, where I is identity matrix.
For each consecutive frame the following steps are performed

1. Read in depth and color frame and apply Kinect transformation to obtain point cloud of the entire frame. Using Kinect SDK, it is simple to map both color and depth information.

2. Apply pre-processing to extract object point cloud $P_{current}$ and point cloud representing data points P_{data} for ICP.

3. For point cloud representing model points P_{model} use data points from previous frame.

4. Use ICP algorithm to obtain approximated transformation matrix T that describes how P_{data} is transformed to P_{model}.

5. Obtain absolute transformation matrix for current point cloud $P_{current}$ by multiplying transformation matrix T from previous point with all other previously obtained transformation matrices $T_{accum} = TT_{accum}$.

6. Apply cumulative transformation matrix T_{accum} to current point cloud $P_{current}$ in order to align it to reference coordinate system $P_{align} = T_{accum}P_{current}$

7. Merge the new aligned point cloud with already aligned points.

Finally, some post-processing is applied on the obtained point cloud. Steps are summarised in flow chart displayed in Fig. 4.1.

In the following, some of the steps and choices made are discussed in detail.

4.2.1 Pre-processing and selection of points

Object segmentation

The first step is to segment out points representing the object from the background. It is relatively simple to do using depth information. 3D bounding box was fitted around the object and all points located outside were removed. Since camera was perpendicular to rotation axis, it was usually possible to remove turntable also that way. In some cases where needed, a plane was fitted on points and everything beneath it was discarded. For bounding box, choosing the height and width (Y- and X-axis) was relatively straightforward. Choosing the depth (Z-axis) required some experiments since depth information

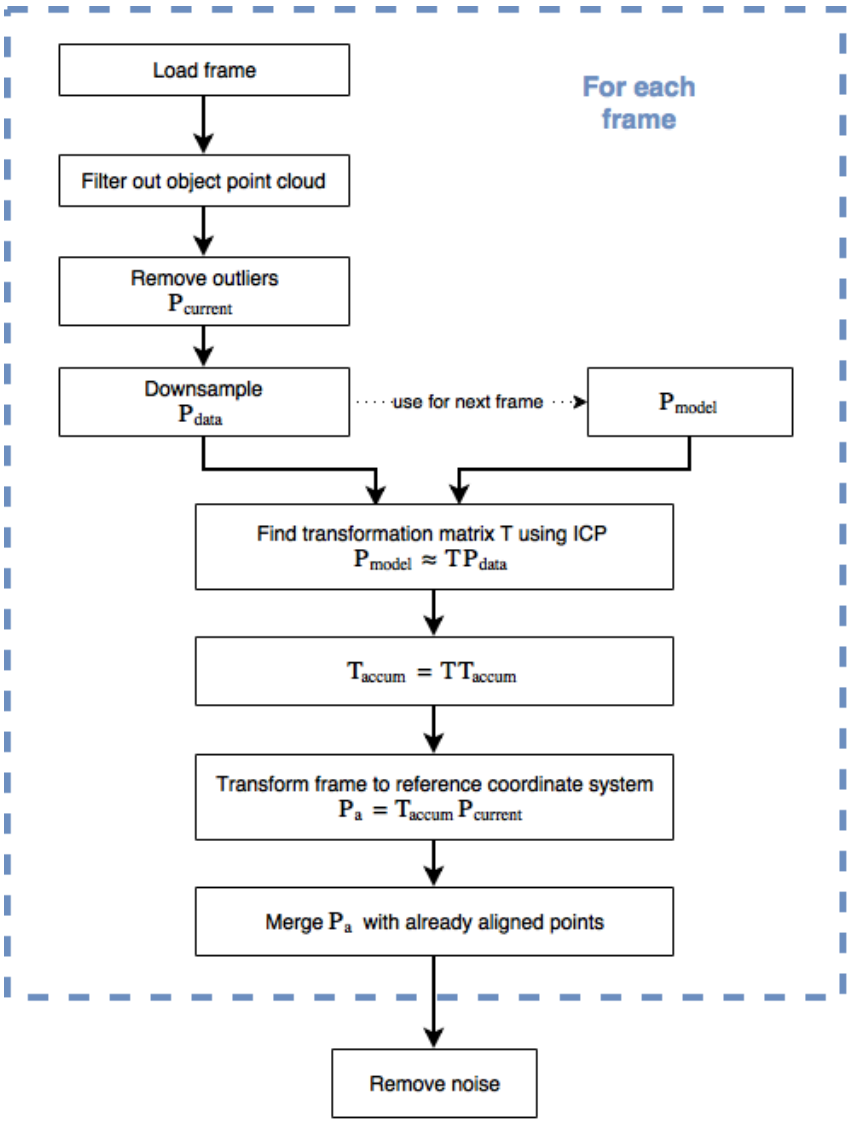


Figure 4.1: Steps for constructing point cloud from Kinect image sequences.

from Kinect can be distorted on the edges of the object. Since frames were available with high frequency, even setting a very conservative threshold for depth, didn't cause consecutive frames to have small number of overlapping points, and allowed ICP to converge quite well.

Removing outliers

Depending on the object surface, Kinect depth data can be relatively noisy, specially on the edges of the object. In order to reduce the errors in registration process, obvious outliers are removed by using a low-pass filter. All points whose average distance from neighboring points is higher than expected threshold (one standard deviation is used) are thrown away. Removing of outliers must be done carefully at this point in order to not throw away too many points as the density of points in one frame is not very high. Removing geometrically important points might cause ICP to converge to wrong transformation.

Downsampling

Depending on the number of points representing the object, point clouds can be reduced to increase the speed of registration calculations. Since most of the objects used were relatively small and reconstruction time was not critical, downsampling was not used in experiments performed for current thesis.

4.2.2 Error metric and point matching

Both *point-to-point* and *point-to-plane* metrics were tested. Probably due to the relative noisiness of Kinect depth data, *point-to-plane* minimization gave much better results and was used for all reconstructions. Since the rotation from frame to frame was small, identity transformation was provided as initial guess for ICP. Theoretically a better assumption could be provided as input but it seemed to have very little effect on speed of convergence in real experiments.

4.2.3 Global alignment and noise removal

After registering individual frames and merging them, there can be two types of problems. Firstly, the individual registrations might contain some errors which appears in the final point cloud as misalignment (f.e. "double nose problem"). It didn't appear often with experimental objects but when needed, it was fixed using error distribution method, which is not described in current work. The other problem is accumulated noise from individual frames. Removing outliers from initial frames only helped against single outliers. However, often bigger areas had distorted coordinates from some particular view. That could not be removed from single frame but can be removed from the merged point cloud. The filtering method was chosen similar to single frame outlier removal but number of neighbours used for distance calculation and distance threshold were chosen much stronger. Heavy noise removal also caused some problems since more horizontal surfaces can have lower point densities and using strong filtering parameters, could be removed from final result.

4.2.4 Color mapping

Registration and merging steps have been performed using only spatial information. Since Kinect can provide one-to-one mapping between spatial and color information, the color information can be simply carried along when performing transformations and merged into final point cloud. If two points are very close together, the spatial and color information from different points can be combined to obtain smaller data set. In current thesis, 1 mm x 1 mm x 1 mm box grid was used and points within one cube were averaged (both spatial coordinates and color values). Result is a colorized point cloud representing the entire object.

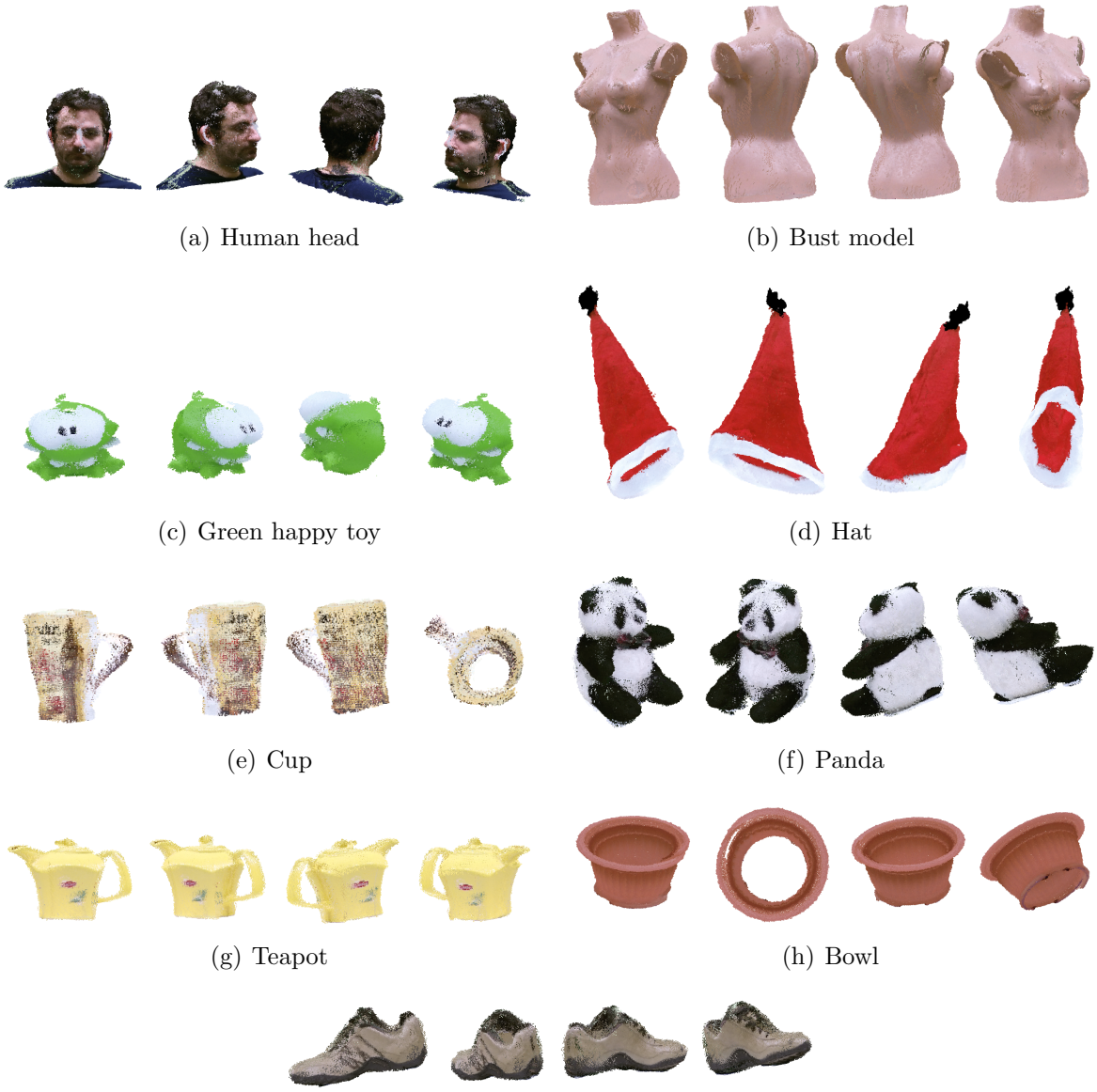
5 Experimental results and discussions

Objects with various shapes, sizes and surfaces were tested to evaluate the viability and quality of 3D reconstruction with Kinect. Sequences were captured with Kinect v2 using Kinect SDK 2.0. Reconstruction code was written in Matlab using native ICP implementation. Processing time for the reconstruction varies a lot depending on the number of points representing the object in frame. Using Intel i7 3.3 GHz processor and 60 input frames, it took between 30 (approximately 1000 points per frame) to 90 seconds (approximately 15 000 points per frame) to reconstruct the object. Final runtime depends also a lot of the noise removal parameters used for the last step, calculating average distances from neighbours is computationally very expensive.

5.1 Results

Point clouds obtained from reconstructions are presented in figures 5.1 and 5.2. Since high frequency input data was available, ICP had to register relative small rotations. Unless objects had very specific features, the correct registration could be achieved. Most important factor affecting the quality of the reconstruction was noise and distortions in the input frames. Since Kinect calculates depth based on reflecting light, the reflective properties of object surface severely affect the result. As can be seen from reconstructions, best results are achieved from objects covered by fabric, best represented by green pouf and pillow. Fabric surfaces of various color give good results, as can be observed from red hat and white and black panda. Even almost completely black objects, such as black bag, provide relatively good reconstructions. For fabric surfaces, object shape plays very little role in the result since the depth measurements are precise. Detail size still must be considered, as Kinect cannot accurately capture very thin details, such as shoelaces on the shoe image.

Objects with more reflective surfaces start to have some problems. The depth information is still good on surfaces facing Kinect but surfaces at larger angle develop distortions. It can be clearly seen on depth data captured from cylindrical object such as a cup in figure 5.3 or surface at big angle such as black box in figure 5.5. Depending on the magnitude of the problem, it might still be possible to reconstruct the object. Such examples can be seen from bust and teapot in figure 5.1, the point clouds contain more noise but can still be aligned. Green toy shows the limitation of Kinect, it is relatively small (approximately 10 cm diameter) with many round surfaces.



(a) Human head (b) Bust model

(c) Green happy toy (d) Hat

(e) Cup (f) Panda

(g) Teapot (h) Bowl

(i) Shoe

Figure 5.1: Reconstruction results displayed from four different orientations.

In addition to creating more noise, object shape can also play role in the result if ICP convergence. Some surface detail is required from symmetrical objects for ICP to give correct results. Even relatively small detail is enough as can be seen from bowl reconstruction in figure 5.1.

Noise removal

As mentioned earlier, once distortions from individual frames get merged into final point cloud, they can be removed by filtering. Effect of noise removal can be seen from figure 5.4. Noise removal is a great tool for producing better results but its use can be limited if object has fine details. In that case important object information can also be removed.

(a) Green pouf

(b) Pillow

(c) Backpack

(d) Chair

(e) Black bag

Figure 5.2: Reconstruction results displayed from four different orientations.

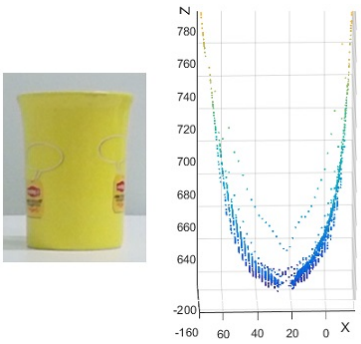


Figure 5.3: Depth info of cup viewed from above. Incorrect shape can be seen clearly on the edges of the cup.

5.2 Limitations

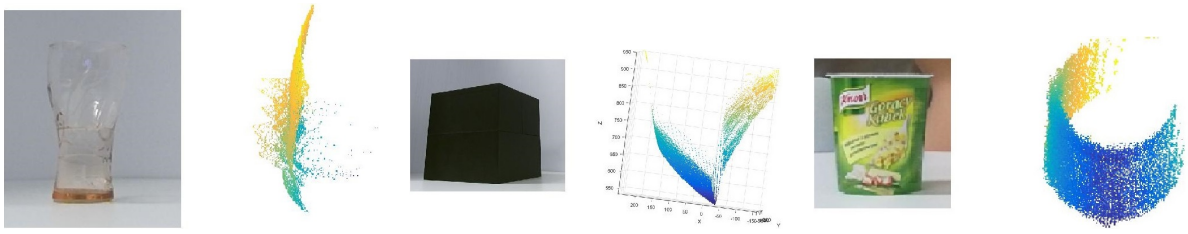
There are some scenarios where reconstruction method described in the current thesis does not work.

5.2.1 Kinect sensor limitations

Due to the technology Kinect uses to calculate depth, it cannot handle reflections well. If object is transparent and the same wave gets partially reflected from object surface and partially reflected from background, the resulting depth image does not correspond to the true situation. It can be observed from figure 5.5 where drinking glass is almost not captured and the main effect is distortion of background depth information.

(a) Panda (b) Boot

Figure 5.4: Merged point clouds before (left) and after noise removal.



(a) Transparent glass (b) Black box (c) Symmetrical object

Figure 5.5: Point clouds of various problematic surfaces and objects.

Second problem related with reflections are surfaces that are very reflection, like mirrors and similar objects. Since light has to reflect back to sensor for Kinect to register it these objects might not appear in depth image at all or have very distorted measurements. Even stainless steel utensils have severe problems which can be observed from figure 5.6.

5.2.2 Symmetrical objects

Additionally, the method doesn't work with fully symmetrical objects. If the depth info is identical in two frames then ICP algorithm converges already with initial transformation guess (identity transformation) and the frames are not aligned correctly. Situation can be seem in figure 5.5 where applying the method didn't give point cloud of the full object. The symmetrical scenario can be fixed with using additional colour information (f.e. placing markers on round table) to calculate the rotation parameters.

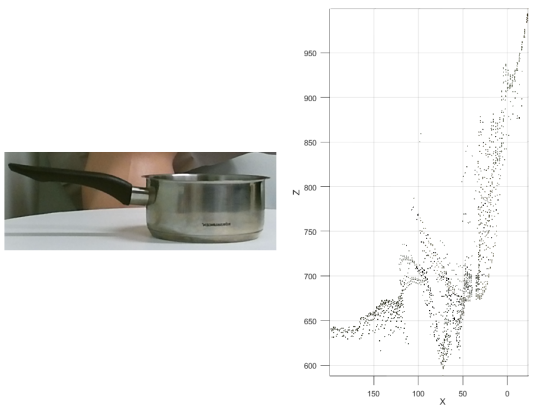


Figure 5.6: Depth info of an object with very reflective surface viewed from above.

Summary

This thesis proposed, implemented and verified the applicability of a system for fast and efficient 3D reconstruction of objects using the Microsoft Kinect v2, and investigated its suitability, as well as its strengths and weaknesses, in terms of achieving the foregoing goal. First, the depth measuring technology utilized by the Kinect v2 sensor was described, and its theoretical limitations were analyzed. Next, the proposed method was introduced through a step-by-step analysis of all the modules involved in the system, consisting of the stages starting from image acquisition and proceeding towards formation and post-processing of a global point cloud representing the object. One of the advantages of the proposed system is, that the created point cloud already represents the object with its original color texture. The algorithm was applied to various objects, where in the most cases, the results were satisfactory. Quality of the reproduction depends significantly on the surface properties of the object. The more reflective the object is, the lower is the quality of its reconstruction. In the case of failure, which mostly happens when dealing with very reflective or transparent surfaces, the reason was the inability of the Kinect sensor to provide the correct depth information. Examples included stainless steel and mirrors, were the reconstruction could not be completed. Since the frame registration method described in this thesis used only the range data, the proposed method also failed to reconstruct the object if it did not have enough distinctive geometric features. If the surface range data stays the same in a pair of consecutive images, as is the case for symmetric objects, the current method cannot be used. For most of the objects, these limitations do not affect the result significantly, which makes the described system, along with the Kinect camera, suitable approach for 3D modelling in settings were high resolution is not demanded. Future works shall concentrate on improving the quality of the created point clouds, especially in the sense of global alignment.

Acknowledgements

I would like to thank my supervisors Gholamreza Anbarjafari and Morteza Daneshmand for guiding and supporting me throughout the thesis process.

I would also like to thank other members of ICV for lending a helping hand when asked.

Bibliography

[1] A. Payne, A. Daniel, A. Mehta, B. Thompson, C. S. Bamji, D. Snow, H. Oshima, L. Prather, M. Fenton, L. Kordus, *et al.*, "7.6 A 512× 424 CMOS 3D Time-of-Flight image sensor with multi-frequency photo-demodulation up to 130MHz and 2GS/s ADC," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 134–135, IEEE, 2014.

[2] F. Järemo Lawin, "Depth data processing and 3D reconstruction using the Kinect v2," Master's thesis, Linköping University, 2015.

[3] P. Sturm, "Pinhole camera model," in *Computer Vision*, pp. 610–613, Springer, 2014.

[4] Microsoft, "Microsoft kinect." `http://www.xbox.com/en-US/xbox-one/accessories/kinect-for-xbox-one` [Accessed: 2016-01-06].

[5] D. Pagliari and L. Pinto, "Calibration of Kinect for Xbox One and Comparison between the Two Generations of Microsoft Sensors," *Sensors*, vol. 15, no. 11, pp. 27569–27589, 2015.

[6] E. Lachat, H. Macher, T. Landes, and P. Grussenmeyer, "Assessment and Calibration of a RGB-D Camera (Kinect v2 Sensor) Towards a Potential Use for Close-Range 3D Modeling," *Remote Sensing*, vol. 7, no. 10, pp. 13070–13097, 2015.

[7] J. Sell and P. O'Connor, "The Xbox One system on a chip and Kinect sensor," *IEEE Micro*, no. 2, pp. 44–53, 2014.

[8] L. Yang, L. Zhang, H. Dong, A. Alelaiwi, and A. El Saddik, "Evaluating and improving the depth accuracy of Kinect for Windows v2," *Sensors Journal, IEEE*, vol. 15, no. 8, pp. 4275–4285, 2015.

[9] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, *et al.*, "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*, pp. 559–568, ACM, 2011.

[10] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[11] J. Xiao, A. Owens, and A. Torralba, "SUN3D: A database of big spaces reconstructed using SfM and object labels," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1625–1632, 2013.

[12] S. Demitševa, "Gesture based computer controlling using Kinect camera." Bachelor thesis, Tartu Ülikool, 2015.

[13] C. S. Bamji, P. O'Connor, T. Elkhatib, S. Mehta, B. Thompson, L. A. Prather, D. Snow, O. C. Akkaya, A. Daniel, A. D. Payne, *et al.*, "A 0.13 μm CMOS system-on-chip for a 512× 424 time-of-flight image sensor with multi-frequency photo-demodulation up to 130 MHz and 2 GS/s ADC," *Solid-State Circuits, IEEE Journal of*, vol. 50, no. 1, pp. 303–319, 2015.

[14] H. Sarbolandi, D. Lefloch, and A. Kolb, "Kinect range sensing: Structured-light versus Time-of-Flight Kinect," *Computer Vision and Image Understanding*, vol. 139, pp. 1–20, 2015.

[15] Microsoft, "Kinect sdk." `https://developer.microsoft.com/en-us/windows/kinect` [Accessed: 2016-04-09].

[16] Microsoft, "Camera intrinsics structure." `https://github.com/Kinect/Docs/blob/master/Kinect4Windows2.0/k4w2/Reference/Kinect_for_Windows_v2/Kinect/CameraIntrinsics_Structure.md` [Accessed: 2016-04-15].

[17] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.

[18] C. Zhang and Z. Zhang, "Calibration between depth and color sensors for commodity depth cameras," in *Computer Vision and Machine Learning with RGB-D Sensors*, pp. 47–64, Springer, 2014.

[19] J. Salvi, C. Matabosch, D. Fofi, and J. Forest, "A review of recent range image registration methods with accuracy evaluation," *Image and Vision computing*, vol. 25, no. 5, pp. 578–596, 2007.

[20] B. K. Horn, "Closed-form solution of absolute orientation using unit quaternions," *JOSA A*, vol. 4, no. 4, pp. 629–642, 1987.

[21] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Robotics-DL tentative*, pp. 586–606, International Society for Optics and Photonics, 1992.

[22] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992.

[23] K.-L. Low, "Linear least-squares optimization for point-to-plane ICP surface registration," *Chapel Hill, University of North Carolina*, vol. 4, 2004.

[24] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pp. 145–152, IEEE, 2001.

[25] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *Vision algorithms: theory and practice*, pp. 298–372, Springer, 1999.

License

Non-exclusive license to reproduce thesis and make thesis public

I, Lembit Valgma,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   "3D reconstruction using Kinect v2 camera", supervised by Assoc. Prof. Gholamreza Anbarjafari and Morteza Daneshmand,

2. am aware of the fact that the author retains these rights.

3. certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 20.05.2016