Examensarbete utfört i datorseende
vid Tekniska högskolan vid Linköpings universitet
av
Felix Järemo Lawin

LiTH-ISY-EX–15/4884–SE
Linköping 2015
Depth data processing and 3D reconstruction using the Kinect v2

Examensarbete utfört i datorseende vid Tekniska högskolan vid Linköpings universitet av

Felix Järemo Lawin

LiTH-ISY-EX–15/4884–SE

Handledare: Hannes Ovrén
ISY, Linköpings universitet

Examinator: Per-Erik Forssén
ISY, Linköpings universitet

Linköping, 17 september 2015
Depth data processing and 3D reconstruction using the Kinect v2

The Kinect v2 is a RGB-D sensor manufactured as a gesture interaction tool for the entertainment console XBOX One. In this thesis we will use it to perform 3D reconstruction and investigate its ability to measure depth.

In order to sense both color and depth the Kinect v2 has two cameras: one RGB camera and one infrared camera used to produce depth and near infrared images. These cameras need to be calibrated if we want to use them for 3D reconstruction. We present a calibration procedure for simultaneously calibrating the cameras and extracting their relative pose. This enables us to construct colored meshes of the environment. When we know the camera parameters of the infrared camera, the depth images could be used to perform the Kinect fusion algorithm. This produces well-formed meshes of the environment by combining many depth frames taken from several camera poses.

The Kinect v2 uses a time-of-flight technology were the phase shifts are extracted from amplitude modulated infrared light signals produced by an emitter. The extracted phase shifts are then converted to depth values. However, the extraction of phase shifts includes a phase unwrapping procedure, which is sensitive to noise and can result in large depth errors.

By utilizing the ability to access the raw phase measurements from the device we managed to modify the phase unwrapping procedure. This new procedure includes an extraction of several hypotheses for the unwrapped phase and a spatial propagation to select amongst them. This proposed method has been compared with the available drivers in the open source library libfreenect2 and the Microsoft Kinect SDK v2. Our experiments show that the depth images of the two available drivers have similar quality and our proposed method improves over libfreenect2. The calculations in the proposed method are more expensive than those in libfreenect2 but it still runs at 2.5× real time. However, contrary to libfreenect2 the proposed method lacks a filter that removes outliers from the depth images. It turned out that this is an important feature when performing Kinect fusion and future work should thus be focused on adding an outlier filter.
Abstract

The Kinect v2 is a RGB-D sensor manufactured as a gesture interaction tool for the entertainment console XBOX One. In this thesis we will use it to perform 3D reconstruction and investigate its ability to measure depth.

In order to sense both color and depth the Kinect v2 has two cameras: one RGB camera and one infrared camera used to produce depth and near infrared images. These cameras need to be calibrated if we want to use them for 3D reconstruction. We present a calibration procedure for simultaneously calibrating the cameras and extracting their relative pose. This enables us to construct colored meshes of the environment. When we know the camera parameters of the infrared camera, the depth images could be used to perform the Kinect fusion algorithm. This produces well-formed meshes of the environment by combining many depth frames taken from several camera poses.

The Kinect v2 uses a time-of-flight technology were the phase shifts are extracted from amplitude modulated infrared light signals produced by an emitter. The extracted phase shifts are then converted to depth values. However, the extraction of phase shifts includes a phase unwrapping procedure, which is sensitive to noise and can result in large depth errors. By utilizing the ability to access the raw phase measurements from the device we managed to modify the phase unwrapping procedure. This new procedure includes an extraction of several hypotheses for the unwrapped phase and a spatial propagation to select amongst them. This proposed method has been compared with the available drivers in the open source library libfreenect2 and the Microsoft Kinect SDK v2. Our experiments show that the depth images of the two available drivers have similar quality and our proposed method improves over libfreenect2. The calculations in the proposed method are more expensive than those in libfreenect2 but it still runs at 2.5× real time. However, contrary to libfreenect2 the proposed method lacks a filter that removes outliers from the depth images. It turned out that this is an important feature when performing Kinect fusion and future work should thus be focused on adding an outlier filter.
Acknowledgments

Firstly, I would like to thank my supervisor Hannes Ovrén and my examiner Per-Erik Forssén for their work in the design of the extensions to the depth data processing algorithm presented in this thesis and all the guidance I received throughout the project. Secondly, I would like to thank Giulia Meneghetti and Andreas Robinson for setting up a workplace and a computer for me. I also want to thank everybody else at CVL for all the help and interesting discussions.

Felix Järemo Lawin
August 2015
# Contents

1 Introduction 1
   1.1 Aims ............................................. 1
   1.2 Motivation ................................. 2
   1.3 Method ...................................... 2

2 The Kinect v2 3
   2.1 Kinect v2 hardware ......................... 3
      2.1.1 Color camera ........................ 3
      2.1.2 Infrared camera ..................... 4
   2.2 Time-of-flight ............................. 5
      2.2.1 Reflection disturbances .......... 9
   2.3 Phase unwrapping .......................... 10
      2.3.1 Phase fusion ....................... 12
   2.4 Proposed extension ....................... 12
      2.4.1 Hypothesis extraction ......... 13
      2.4.2 Spatial Propagation ............. 13
   2.5 Pipeline description ...................... 14
      2.5.1 libfreenect2 with outlier rejection 15
      2.5.2 Hypothesis propagation pipeline 16
      2.5.3 libfreenect2 without outlier rejection 16
      2.5.4 Microsoft Kinect SDK v2 ...... 17
   2.6 Implementation .......................... 18

3 Calibration 19
   3.1 The pinhole camera model ............ 19
      3.1.1 Distortion models .......... 20
   3.2 Single camera calibration .......... 22
   3.3 Joint camera calibration ............ 23
   3.4 Rectify image .......................... 25
   3.5 Distance to depth ...................... 25
   3.6 Depth to color mapping .............. 27

4 3D Reconstruction 29
Introduction

The Kinect v2 is an RGB-D sensor, which means that it can sense both color and depth. It has one RGB camera and one infrared camera used to detect depth and is designed as tool for gesture interaction with the entertainment console XBOX One. Its predecessor, the Kinect v1, has been around commercially since 2010. The cheap prize and good performance has made the Kinect v1 a popular tool in robotics and other computer vision research fields. It uses a structured light projector in order to triangulate depth values. The newer Kinect v2 was released commercially in 2014 and is not as well documented as the Kinect v1 at this point. One of the most apparent distinctions from the Kinect v1 is that the Kinect v2 uses a time-of-flight technology to provide a depth image. Another distinction between the two sensors is that the Kinect v2 outputs raw sensor measurements, while the Kinect v1 outputs depth values that have been processed in firmware. For Kinect v2 users this allows for software modifications of the depth calculations.

1.1 Aims

In this thesis we will examine the data produced by the Kinect v2 to evaluate the ability to measure depth and perform 3D reconstruction. We will evaluate its available data processing algorithms and also suggest extensions in the depth calculations. Further on, the performance of the Kinect v2 will be evaluated through a set of experiments. We will visualize the results from the 3D reconstruction algorithms using 3D graphics. This requires the cameras to be calibrated. Therefore we will present a method for calibrating the cameras of the sensor. Additionally, we want to examine the ability to measure depth larger than the specified depth of 4.2 meters suitable for a living room[5].
1.2 Motivation

RGB-D sensors such as the Kinect v2 have the potential to become useful both in computer vision research and in various commercial applications. It is already used in the gaming industry, however it may also be useful for indoor robotics and 3D reconstruction. This thesis aims to provide an evaluation of the performance of the sensor and thus give an indication of its applicability for those fields of research. The quality of the depth images is an important property if we want to perform 3D reconstruction. Therefore, any improvements will be helpful.

1.3 Method

The thesis can be divided into three main sections:

1. Firstly, we need to understand the Kinect v2 and how the data is processed. This will include studies of related papers and other publications. The time-of-flight technology will be explained and how this is applied in the open source library *libfreenect2* [20] for the Kinect v2 to measure depth. This part will also present our new processing algorithm as an extension to the method in *libfreenect2*.

2. Secondly, in order to perform 3D reconstruction we need to calibrate the Kinect v2 cameras. We will present a method for calibrating two cameras simultaneously and in that process obtain the relative pose between the cameras. We will use some of the built in calibration functions in *OpenCV*[4] in the implementation. Our calibration will be compared with the already built in calibration in the Kinect v2 device.

3. In the third step we will perform 3D reconstruction. We will use the results from the calibration step to construct colored mesh representations of the environment. The implementation *Kinfu* in *The Point Cloud Library* (PCL)[22] will be integrated with *libfreenect2* to perform the Kinect fusion algorithm. This will produce a mesh representation of the environment constructed from a set of depth images taken from several camera poses. We will then examine the sensitivity to reflection disturbances, which is an unwanted property that comes with the time-of-flight technology. This is done by setting up different environments with different amount of exposure to these reflection disturbances.

By performing these steps we can evaluate performance of the Kinect v2 sensor as tool for 3D reconstruction.
In this chapter we will provide a short description of the Kinect v2 hardware and analyze the depth calculations. We will also utilize the ability to acquire raw sensor measurements by introducing an extension to the depth calculations. There are two types of error sources in the depth images produced using the time-of-flight principle. The first one is due to the measurement noise which causes small temporal deviations in the depth image, while the second one is due to falsely unwrapped phase shifts causing large errors in the depth. This thesis introduces a new method that attempts to reduce these large errors by improving the phase unwrapping of time-of-flight depth measurements.

### 2.1 Kinect v2 hardware

The Kinect v2 has two cameras: one color camera for producing RGB images of visible light and one infrared camera coupled with an infrared pulse modulated light source for producing depth images and near infrared images. Figure 2.1 shows the Kinect v2 device. In contrast to its predecessor, the Kinect v1, Kinect v2 outputs raw sensor measurements, instead of depth maps, and these are decoded by the host driver, i.e. either the Microsoft SDK [18], or the open source drivers in *libfreenect2*[20].

#### 2.1.1 Color camera

The RGB images have a size of 1920 × 1080 pixels and are produced at a rate of 30Hz. The data is compressed to JPEG on the device before it is transmitted and therefore it needs to be unpacked before viewing.
2.1.2 Infrared camera

The infrared camera of the Kinect v2 has a $512 \times 424$ CMOS array of pixels. Each pixel in the infrared camera has two photo diodes\cite{5}. When a photo diode is turned on it converts light into current which can be integrated into a electrical potential. In the Kinect v2 the diodes are switched on and off rapidly such that when the first diode is turned on the second diode is turned off. For each frame the light is measured using the difference between the output voltage produced by the diodes. In theory, this results in a correlation between the input light and the reference signal driving the pixel diodes. This technique is called quantum efficiency modulation\cite{5} and is used in the Kinect v2 to extract the phase of an amplitude modulated light signal. The amplitude modulated signal is produced by a light emitter, which is a part of the Kinect v2 device. The emitter is driven by the same reference signal driving the pixel diodes.

The Kinect v2 also uses a multi-shutter engine, which reduces the risk for saturation. The multi-shutter engine uses several different shutter times and the longest shutter time that does not result in a saturation is chosen as the output for the specific pixel. The value is normalized with respect to the shutter time \cite{5}.

We will describe exactly how the phase and amplitude is extracted from the infrared camera measurements and how this is used to produce a depth image in section 2.2.
2.2 Time-of-flight

The basic idea in the time-of-flight principle (ToF) is to emit light-pulses and measure the time difference between emission time, $t_e$, and received time $t_r$. The distance $d$ can be calculated as

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

where $c$ is the speed of light.

This can be accomplished using various technologies, however the Kinect v2 utilize amplitude modulated infrared light with a CMOS array receiver, where the phase difference between the emission source and the received light is measured. The received signal $S(t)$ is correlated with phase shifted versions of the reference signal, $R(t)$, driving the light emitter. This is achieved on the camera chip by using quantum efficiency modulation and integration resulting in a voltage value $V$ [1, 5]. The emitted signal can for simplicity be modeled as

$$R(t) = I_0(1 + \cos(2\pi f_m t)) \tag{2.2}$$

Where $I_0$ is the amplitude of the signal. Thus, due to the time of flight, the received signal will be

$$S(t) = I_r(1 + \cos(2\pi f_m t - \phi)) \tag{2.3}$$

where $I_r$ is the light intensity of the reflected light. The value of $I_r$ depends on many factors, among them the distance to the light reflecting object and the reflectance. The signals is illustrated in figure 2.2.

![Diagram](image)

**Figure 2.2: Illustration of the time of flight principle**

The phase shift $\phi$ is a function of the time difference $t_r - t_e$ which gives us

$$I_r \cos(2\pi f_m t - \phi) = I_r \cos((2\pi f_m (t - (t_r - t_e)))) \tag{2.4}$$

By combining the expressions in (2.1) and (2.4) we get
The Kinect v2

\[ I_r \cos(2\pi f_m t - \phi) = I_r \cos\left(2\pi f_m \left(t - \frac{2d}{c}\right)\right) \]  \hspace{1cm} (2.5)

which implies

\[ d = \frac{c \cdot \phi}{4\pi f_m} \]  \hspace{1cm} (2.6)

Since there are two unknown parameters in \( S(t) \), \( I_r \) and \( \phi \), we need at least two different measurements. As suggested in the patent [2] \( S(t) \) could be mixed with two phase shifted versions of the reference signal for the emitter \( \cos(2\pi f_m t - \theta_0) \) and \( \cos(2\pi f_m t - \theta_1) \), where \( \theta_1 \) and \( \theta_0 \) are 90 degrees apart. The mixed output signal is then low-pass filtered such that the frequency components are removed.

\[ V = LP[S(t) \cos(2\pi f_m t - \theta)] = LP[(I_r + I_r \cos(2\pi f_m t - \phi)) \cdot \cos(2\pi f_m t - \theta)] = 0.5I_r \cos(\phi - \theta) \]  \hspace{1cm} (2.7)

Equation (2.7) shows the output value \( V \) after the mixing and low-pass filtering of \( \cos(2\pi f_m t - \theta_0) \) and \( \cos(2\pi f_m t - \theta_1) \).

In equation (2.7) we see that this produces the signals \( 0.5I_r \cos(\phi - \theta_0) \) and \( 0.5I_r \cos(\phi - \theta_1) \). By using these trigonometric identities

\[ \sin\left(\alpha + \frac{\pi}{2}\right) = \cos(\alpha) \]  \hspace{1cm} (2.8)

\[ \phi = \text{atan2}(-0.5I_r \sin(-\phi), 0.5I_r \cos(-\phi)) \]  \hspace{1cm} (2.9)

we can extract the phase shift. Here \( \text{atan2} \) denotes the arctangent function with two arguments that can be found in many programming languages.

This signal model is a rough simplification and has a bad resemblance with reality. It is hard to produce a perfect cosine-shaped amplitude modulation without a DC-component and higher order harmonics. In [10] this is taken into account for \( S(t) \) and \( R(t) \) as they are modeled as non-harmonic signals with an unknown DC component \( c \):

\[ S(t) = c_s + \sum_{k=1}^{K} S'_k \cos(2\pi k f_m t - k\phi) = c_s + \sum_{k=1}^{K} \frac{S'_k}{2}(e^{i2\pi k f_m t - ik\phi} + e^{-i2\pi k f_m t + ik\phi}) \]  \hspace{1cm} (2.10)

\[ R(t) = c_r + \sum_{k=1}^{K} R'_k \cos(2\pi k f_m t) = c_r + \sum_{k=1}^{K} \frac{R'_k}{2}(e^{i2\pi k f_m t} + e^{-i2\pi k f_m t}) \]  \hspace{1cm} (2.11)

Here \( R(t) \) is the reference signal for the \( n \)th measurement, where \( \{n\}^N_{0} \), which should have a phase offset \( \frac{2\pi n}{N} \) for \( N \) measurements. In [10] the extraction of \( \phi \)
in the general case is derived by using the Fourier series of $S(t)$ and the reference signal $R(t)$:

$$R(t) = \sum_{j=-\infty}^{\infty} R_j e^{i2\pi f_m t}$$  \hspace{1cm} (2.12)

$$S(t) = \sum_{k=-\infty}^{\infty} S_k e^{i k 2\pi f_m t - ik \phi}$$  \hspace{1cm} (2.13)

$S(t)$ is then correlated with $R(t + \frac{2\pi n}{2\pi f_m N})$:

$$V_n = \frac{1}{T} \int_{t}^{\infty} \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} R_j S_k e^{i j 2\pi f_m t + \frac{12\pi n}{N}} e^{i k 2\pi f_m t - ik \phi} dt =$$

$$\sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} R_j S_k e^{i k \frac{2\pi n}{N} - ik \phi} \frac{1}{T} \int_{t}^{\infty} e^{i j 2\pi t} e^{i k 2\pi t} dt$$  \hspace{1cm} (2.14)

Here $V_n$ is the output of the correlation and represents the $n$th measurement. If $j \neq -k$ the integral in equation (2.14) is negligible since the integration time $T$ is in practice very small. If $j = -k$ we get:

$$V_n \approx \sum_{k=-\infty}^{\infty} R_{-k} S_k e^{i k \frac{2\pi n}{N} - \phi}$$  \hspace{1cm} (2.15)

We can now use (2.10) and (2.11) and write (2.15) as:

$$V_n = c + \sum_{k=1}^{K} A_k \left( e^{i k \frac{2\pi n}{N} - \phi} + e^{-i k \frac{2\pi n}{N} - \phi} \right) = c + \sum_{k=1}^{K} A_k \left( e^{i k \frac{2\pi n}{N} - \phi} + e^{-i k \frac{2\pi n}{N} + \phi} \right)$$  \hspace{1cm} (2.16)

where $c = R_0 S_0$ and $A_k = S_k R_k'$

For $N$ measurements this could be written in matrix form as:

$$\begin{pmatrix} V_0 \\ \vdots \\ V_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 & 1 \\ w & \bar{w} & \cdots & w^K & \bar{w}^K & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w^{N-1} & \bar{w}^{N-1} & \cdots & \bar{w}^{K(N-1)} & w^{K(N-1)} & 1 \end{pmatrix} \begin{pmatrix} \frac{A_1}{2} z_1 \\ \frac{A_1}{2} \bar{z}_1 \\ \vdots \\ \frac{A_K}{2} z_K \\ \frac{A_K}{2} \bar{z}_K \\ c \end{pmatrix}$$  \hspace{1cm} (2.18)

where $z_k = e^{-ik\phi}$, $w = e^{i \frac{2\pi n}{N}}$ and $\bar{w} = e^{-i \frac{2\pi n}{N}}$  \hspace{1cm} (2.19)
The least-squares solution to the system of equations in (2.18) could be found using the pseudo inverse of $W$:

$$W = \begin{pmatrix} w_1 & \bar{w}_1 & \cdots & w_K & \bar{w}_K & 1 \end{pmatrix}$$

(2.20)

and

$$W^* = \begin{pmatrix} \bar{w}_1 & w_1 & \cdots & \bar{w}_K & w_K & 1 \end{pmatrix}^T$$

(2.21)

where $w_k = (1, w^k, \ldots, w^{k(N-1)})^T$

(2.22)

$$w_k^T w_l = \sum_{n=0}^{N-1} e^{i \frac{2\pi n}{N} (k+l)} = 0 \quad \forall k, l$$

(2.23)

$$\bar{w}_k^T w_l = \sum_{n=0}^{N-1} e^{i \frac{2\pi n}{N} (k-l)} = \begin{cases} N, & \text{if } k=l \\ 0, & \text{otherwise.} \end{cases}$$

(2.24)

$$\Rightarrow \left( \frac{1}{N} W^* \right) \cdot W = 1$$

(2.25)

If $N > 2K$, we have equally as many or more measurements than unknowns. This implies that the phase shifts, amplitudes and DC-component could be extracted accurately. Note that $K$ could be interpreted as the bandwidth of the signals $S(t)$ and $R(t)$ [10], thus $S(t)$ could be reconstructed without aliasing using significantly high sample frequency according to the Nyquist-Shannon sampling theorem. From the solution of $X$ in (2.18) the phase and amplitude could now be extracted using:

$$k\phi = -\arg \left[ \sum_{n=0}^{N-1} V_k e^{-i(2\pi kn/N)} \right].$$

(2.26)

and

$$A_k = \frac{2}{N} \left\| \sum_{n=0}^{N-1} V_k e^{-i(2\pi kn/N)} \right\|.$$ 

(2.27)

This introduces a trade-off between the sample rate and the accuracy of the output phase measurement. For a dynamic environment, the number of samples i.e., the number of images per frame for a time-of-flight camera, the sample rate needs to be kept low to reduce errors caused by motion blur from moving objects.

In the Kinect v2 three different phase shifted versions of the reference signals is used which are shifted $\frac{2\pi}{3}$ radians apart with respect to each other [2]. This results in three voltage values $V_0$, $V_1$, $V_2$, which are used to calculate the phase shift between the emitted and the received signals, using:

$$\phi = -\arg \left[ \sum_{n=0}^{2} V_n e^{-i(p_0+2\pi n/3)} \right].$$

(2.28)

And the amplitude:
\[ A = \frac{2}{3} \left\| \sum_{n=0}^{2} V_n e^{-i\left(p_0 + \frac{2\pi n}{3}\right)} \right\| . \] (2.29)

Here \( p_0 \) is a common phase offset, which is specified for every pixel in the receiver. The Kinect v2 approach is equivalent to setting \( K = 1 \) in the equations (2.10) and (2.11) resulting in exact solutions for harmonic signals. The patent [2] states that a large proportion of the errors caused by the higher order harmonics of the signals will be canceled out through averaging over the basis vectors \( e^{i2\pi/3} \) and \( e^{i4\pi/3} \). The good quality in the depth images produced by the sensor using this method shows that this is a good approach.

### 2.2.1 Reflection disturbances

A problematic property that comes with time-of-flight is disturbances caused by reflections from other objects than the observed one as illustrated in figure 2.3.

![Illustration of the time-of-flight principle with reflection disturbances.](image)

Such disturbances cause the measurement in the pixel to be contaminated by a second signal source which is delayed compared with the target signal \( S(t) \). Since the signal model in equation (2.10) assumes only one phase shifted signal, the resulting phase will be shifted such that the observed object is perceived to be further away than it actually is. If the signals are harmonic, the sum of the signals will simply become another harmonic signal with a different phase shift. An attempt to resolve this issue is provided in [7]. They modify the firmware of the Kinect v2 to emit and receive at 5 frequencies instead of the default 3. Using these new measurements, the reflections can be separated and the dominant reflection will be considered as the target signal. In this thesis, however, we will be using the unmodified firmware and thus we will not be protected from this kind of problem.
2.3 Phase unwrapping

Equation (2.28) will produce the same value $\phi$ if the true phase shift was $\phi + 2\pi n$, $\forall n \in \mathbb{N}$. Thus, $\phi$ is ambiguous in an environment where $d$ can be larger than $c/(2f_m)$. Finding the correct period, i.e. $n$ in the expression:

$$\phi = \phi_{\text{wrapped}} + 2\pi n,$$

(2.30)

is called phase unwrapping. If the wrong period is chosen it will result in large depth errors.

To reduce measurement noise, and increase the range in which $\phi$ is unambiguous, the Kinect v2 uses amplitude modulated light signals with three different frequencies. As mentioned in section 2.1.2 these modulation frequencies are set to 16, 80 and 120 MHz [23]. For each of the three frequencies, three phase shifts are used to calculate a phase according to equation (2.28) (and thus a total of nine measurements are used in each depth calculation).

Figure 2.4 shows the phase to distance relation for the three amplitude modulated signals. We see that if the phase shifts are combined, a common wraparound occurs at 18.75 meters. This is the maximum range in which the Kinect v2 can operate without depth ambiguity.

The unambiguous depth range increases as the modulation frequency decreases. From this one might conclude that the modulation frequency should be kept as low as possible to obtain a large depth range without ambiguity. However, as is stated in [11], the standard deviation of the noise in the phase shift measurements is inversely proportional to the size of modulation frequency. To achieve both a high depth resolution and a large unambiguous depth range the measurements from the different modulation frequencies are combined.

We know that for the Kinect v2, the three calculated phase shifts $\phi_0$, $\phi_1$, $\phi_2$ correspond to the same distance, $d$, see (2.1). This gives us the following relation:

$$d = \frac{c(\phi_0 + 2\pi n_0)}{4\pi f_0} = \frac{c(\phi_1 + 2\pi n_1)}{4\pi f_1} = \frac{c(\phi_2 + 2\pi n_2)}{4\pi f_2},$$

(2.31)

where $n_0, n_1, n_2$ are the sought unwrapping coefficients. If we insert the frequencies $f_0 = 80$MHz, $f_1 = 16$MHz and $f_2 = 120$MHz, their ratios are $f_0 : f_1 : f_2 = 10 : 2 : 15$, which gives us:

$$\frac{\phi_0 + 2\pi n_0}{4\pi \cdot 10} = \frac{\phi_1 + 2\pi n_1}{4\pi \cdot 2} = \frac{\phi_2 + 2\pi n_2}{4\pi \cdot 15} \iff (2.32)$$

$$\frac{3(\phi_0 + 2\pi n_0)}{2\pi} = \frac{15(\phi_1 + 2\pi n_1)}{2\pi} = \frac{2(\phi_2 + 2\pi n_2)}{2\pi} \iff (2.33)$$

$$\frac{3\phi_0}{2\pi} + 3n_0 = \frac{15\phi_1}{2\pi} + 15n_1 = \frac{2\phi_2}{2\pi} + 2n_2 \quad (2.34)$$

Here $\phi_0$, $\phi_1$ and $\phi_2$ are computed using (2.28) and in general contain noise with standard deviations $\sigma_{\phi_0}$, $\sigma_{\phi_1}$ and $\sigma_{\phi_2}$. For simplicity of notation we set
2.3 Phase unwrapping

![Graphs of 80MHz, 16MHz, and 120MHz phases](image)

**Figure 2.4:** Wrapped phases in the range 0 to 25 meters. Top to bottom: \( f_0, f_1, f_2 \). The dashed line at 18.75 meters indicates the common wrap-around point for all three phases.

\[ t_0 = \frac{3\phi_0}{2\pi}, \quad t_1 = \frac{15\phi_1}{2\pi} \quad \text{and} \quad t_2 = \frac{2\phi_2}{2\pi}, \] which implies \( \sigma_{t_0} = \frac{3\sigma_\phi_0}{2\pi}, \quad \sigma_{t_1} = \frac{15\sigma_\phi_1}{2\pi} \quad \text{and} \quad \sigma_{t_2} = \frac{2\sigma_\phi_2}{2\pi}. \] We obtain the following system of equations:

\[
\begin{align*}
3n_0 - 15n_1 &= t_1 - t_0 \quad \text{(2.35)} \\
3n_0 - 2n_2 &= t_2 - t_0 \quad \text{(2.36)} \\
15n_1 - 2n_2 &= t_2 - t_1 \quad \text{(2.37)}
\end{align*}
\]

At the correct unwrapping \((n_0, n_1, n_2)\), the residual noise in the phase measurements (2.35) to (2.37) is given by (2.38) to (2.40):

\[
\begin{align*}
3n_0 - 15n_1 - (t_1 - t_0) &= \epsilon_1 \quad \text{(2.38)} \\
3n_0 - 2n_2 - (t_2 - t_0) &= \epsilon_2 \quad \text{(2.39)} \\
15n_1 - 2n_2 - (t_2 - t_1) &= \epsilon_3 \quad \text{(2.40)}
\end{align*}
\]

We thus want to find a triplet \((n_0, n_1, n_2)\) that minimizes
\[ J(n_0, n_1, n_2) = \frac{\epsilon_1^2}{\sigma_{\epsilon_1}^2} + \frac{\epsilon_2^2}{\sigma_{\epsilon_2}^2} + \frac{\epsilon_3^2}{\sigma_{\epsilon_3}^2} \]  

(2.41)

with an integer constraint on \((n_0, n_1, n_2)\). Assuming independence of \(\epsilon_1, \epsilon_2\) and \(\epsilon_3\), this cost function corresponds to the negative log-likelihood of the parameters. For normally distributed residuals, (2.38) to (2.40) imply:

\[ \sigma_{\epsilon_1}^2 = \sigma_{t_1}^2 + \sigma_{t_0}^2 \]  

(2.42)

\[ \sigma_{\epsilon_2}^2 = \sigma_{t_2}^2 + \sigma_{t_0}^2 \]  

(2.43)

\[ \sigma_{\epsilon_3}^2 = \sigma_{t_2}^2 + \sigma_{t_1}^2 \]  

(2.44)

which provides the scaling in the cost function (2.41).

The open source library libfreenect2 \[20\] does not use (2.41), instead it finds \((n_0, n_1, n_2)\) using a greedy approach: First \(n_0 - 5n_1\) is solved for in (2.35) and used to extract an unwrapped \(t_0\) such that it has the same ambiguity range as \(t_1\). This value is then subtracted from \(t_2\) to form the right hand of (2.37), from which \(n_2\) is found, assuming either \(n_1 = 0\) or \(n_1 = 1\). The best choices of \(n_1\) and \(n_2\) are then used to unwrap \(t_0\) and \(t_1\). This method is fast but it is sensitive to noise, as it chooses values of \(n_0, n_1\), and \(n_2\) separately.

### 2.3.1 Phase fusion

After unwrapping, the scaled phase measurements \(t_0, t_1\) and \(t_2\) are combined using a weighted average:

\[ t = \frac{1}{\sum_{m=0}^{2} 1/\sigma_{t_m}} \sum_{m=0}^{2} \frac{t_m}{\sigma_{t_m}}. \]  

(2.45)

Such a normalization by the standard deviation minimizes the expected variance of the fused phase.

The standard deviations \(\sigma_{t_0}, \sigma_{t_1}\) and \(\sigma_{t_2}\) could be estimated, but according to \[11\], they should be inversely proportional to the modulation frequency, and this assumption is also used in libfreenect2. The \(t\) estimate is later scaled to a proper distance, and finally converted to a depth (i.e. distance in the forward direction).

### 2.4 Proposed extension

It is of critical importance that the phase is correctly unwrapped, as choosing the wrong period will result in large depth errors. There has been, several attempts to reduce the noise produced by Tof-cameras. Median and weighted Gaussian filtering of the depth image was attempted in [9]. This resulted in less noisy depth images but on the downside it caused blurriness and lost definition around edges, which is a common artifact for such regularizers. In [13] however, processing is made on the raw phase measurements of a one frequency system. They assume two candidates to the unwrapped output phase if the phase measurement is close
2.4 Proposed extension

to theToF ambiguous range. The candidate that minimizes the weighted distance
to neighbours both in a spatial and the temporal domain becomes the output
phase. This method may work well for salt and pepper noise as the authors
demonstrate, but it will face troubles when large regions are wrongly unwrapped
both spatially and temporally. It is not applicable in the system described here
since three frequencies introduces potentially wrongly unwrapped phase values
all over the unambiguous range. Temporal filters will also cause artifacts in dy-
namic environments.

In this thesis we will perform filtering on the phase measurements. We pro-
pose a method for phase unwrapping, different from the one in \textit{libfreenect2}. We
then perform a spatial propagation in order to determine the correct unwrapped
phase. The idea is that this will result in a regularization of the depth image
without smoothing artifacts.

2.4.1 Hypothesis extraction

Contrary to the \textit{libfreenect} phase unwrapping procedure all possible combina-
tions of values \((n_0, n_1, n_2)\) are considered, for \(n_1 \in [0,1]\). For a given value of
\(n_1\), only a limited number of values for \(n_0\) and \(n_2\) are reasonable for each dis-
tance. For example, looking at figure 2.4, if \(n_0 = n_1 = 0\), \(n_2\) could either be 0 or 1.
In total 30 different hypotheses for \((n_0, n_1, n_2)\) are constructed in this way. These
can then be ranked by (2.41).

Compared with the \textit{libfreenect2} approach, that only considers one hypothe-
sis, the testing of 30 hypotheses is more expensive. On the other hand, the true
minimum of (2.41) is guaranteed to be checked.

Under the assumption of independent Gaussian noise, the minimum of (2.41)
is a maximum likelihood estimate of the true phase shift. In the low noise case,
we can thus expect it to be correct. This is however not necessarily the case in
general. Therefore the \(K\) best hypotheses are saved for further consideration, as
explained below.

2.4.2 Spatial Propagation

The next step is to determine which of the hypotheses to use as the final phase
shift output. This is done by assuming that neighbouring pixels tend to have
similar phase values. The noise tends to be smaller for pixels close to the center of
the image than for pixels near the edges. This is due to a vignetting effects in the
camera that causes the central pixels in the pixel array to be exposed to a larger
amount of light. This means that the hypothesis that minimizes the expression
in (2.41) is the correct one with high probability for pixels near the image center.
Using this notion all within \(N\) columns or \(M\) rows from the image center are
initialized with the first hypothesis. We now add the assumption that smooth
changes in phase are more likely than discontinuities. Using this assumption the
The following cost function is constructed:

\[ c_i(x) = \frac{J_i(x)}{\sum_{x' \in \mathcal{N}(x)} w(x')} \sum_{x' \in \mathcal{N}(x)} w(x') |\phi_i(x) - \phi(x')|^2 \]  

(2.46)

where \( w(x') = \begin{cases} 1 & \text{if } \phi(x') \text{ is valid}, \\ 0 & \text{otherwise}. \end{cases} \)  

(2.47)

Here \( \phi_i(x) \) is the phase value and \( J_i(x) \) is the value of the cost function (2.41) for hypothesis \( i \) in pixel \( x \). The cost function is propagated from the middle to the edge of the image vertically and horizontally, as illustrated in figure 2.5.

Figure 2.5: Illustration of hypothesis propagation seen from image and pixel level. Left: Propagation in horizontal direction. Right: Propagation in vertical direction.

The set \( \mathcal{N}(x) \) contains the 3 neighbouring pixels from the previous row or column, as shown in figure 2.5. The hypothesis that minimizes the cost \( c_i(x) \) is chosen. The propagation is step-wise synchronized meaning that all pixels within the current column or row must be completed before the next propagation step is ready to go. However, there is no synchronization between the vertical and the horizontal propagation resulting in two solutions per pixels. These are \( \phi_v(x) \) and \( \phi_h(x) \) for the vertical and horizontal propagation respectively. The final chosen phase is the one with the smallest corresponding costs \( c_v(x) \) and \( c_h(x) \) (note that \( v \) and \( h \) can correspond to the same hypothesis and thus \( \phi_v(x) \) and \( \phi_h(x) \) will be equal).

2.5 Pipeline description

The objective is to present a phase unwrapping algorithm that finds the correct phase with as few errors over the depth image as possible. The algorithm will be compared to the one already implemented in libfreenect2 and the one in Microsoft Kinect SDK v2. In addition to the phase unwrapping procedure libfreenect2 also provides a set of outlier rejection steps to remove depth values that are considered defective. These pixels will hereby be called undetected pixels. However,
this processing removes some of the pixels with valid depth as well. Saturated pixels are also treated as undetected.

In order to interpret the performances of the different algorithms, we now describe the pipelines used to calculate the depth.

### 2.5.1 libfreenect2 with outlier rejection

The voltage measurements produced by the Kinect v2 sensor are first processed by a bilateral filter, which results in a spatial smoothing without affecting edges. The amplitude is calculated using (2.29). As stated in [14], the depth estimate error increases as the amplitude decreases, introducing less reliability on the phase unwrapping outputs. In *libfreenect2*, this is handled by thresholding the amplitude measurements, resulting in an undetected phase value for amplitudes below the threshold.

Using (2.31) one can derive that the unwrapped phase values should follow the relation:

\[
\phi_0 + 2\pi n_0 : \phi_1 + 2\pi n_1 : \phi_2 + 2\pi n_2 = \frac{1}{3} : \frac{1}{15} : \frac{1}{2}
\]

(2.48)

If noise is added the phase values will deviate from this relation. We now define vectors from the right hand side and the left hand side of (2.48):

\[
v_{\phi} = (\phi_0 + 2\pi n_0, \phi_1 + 2\pi n_1, \phi_2 + 2\pi n_2)
\]

(2.49)

\[
v_{\text{relation}} = \left(\frac{1}{3}, \frac{1}{15}, \frac{1}{2}\right)
\]

(2.50)

The rate at which the phase value deviates is measured by the angle \(\theta\) between such vectors. The angle is calculated from the cross product:

\[
\|v_{\phi}\| \cdot \sin(\theta) = \frac{\|v_{\phi} \times v_{\text{relation}}\|}{\|v_{\text{relation}}\|}
\]

(2.51)

In *libfreenect2* pixels are marked as undetected when the following constraint is fulfilled:

\[
\|v_{\phi}\| \cdot \sin(\theta) > 12.8866 \cdot A_{\text{max}}^{0.533}
\]

(2.52)

where \(A_{\text{max}}\) is the largest amplitude (as defined in (2.29)) among the three modulation frequencies.

In the next step the unwrapped phase is transformed to a depth value. The user could customize the allowed range for depth values. If the depth value is outside the range the pixel is marked as undetected. In this thesis though, the range used is 0.5 to 18.75 meters. The resulting depth values are filtered spatially. This filter marks pixels as undetected if their \(3 \times 3\) neighbourhood has a large depth or amplitude variance. It also sets pixels as undetected if their depth value deviates from its neighbours. The filter is also combined with an edge detector applied to the voltage measurements, where pixels on edges are marked as undetected. The whole pipeline is illustrated in figure 2.6.
2.5.2 Hypothesis propagation pipeline

This is the proposed pipeline. Just like the libfreenect2 outlier rejection pipeline it uses the bilateral filter but instead of the libfreenect2 phase unwrapping method it uses the one proposed in section 2.4. There is no outlier rejection, thus there will be no undetected pixels except for those that are saturated.

In [11] it is claimed that the standard deviation of the depth measurements is inversely proportional to the modulation frequency of the light. Using this and expressions (2.42) to (2.44) the weights applied in the cost function (2.41) become:

\[
\frac{1}{\sigma_{t_1}^2} = 0.7007 \\
\frac{1}{\sigma_{t_2}^2} = 366.2946 \\
\frac{1}{\sigma_{t_3}^2} = 0.7016
\]

The relation between the weights favor small errors in equation (2.39). This is reasonable since this equation evokes most of the hypothesis and the measurements \(t_0\) and \(t_2\) introduces much less noise than \(t_1\) due to modulation frequency.

The number of hypotheses used in the propagation was \(K = 2\). Experiments has shown that setting \(K = 3\) did not improve the results. The propagation offset parameters \(N\) and \(M\) were set to 1. This means that there are only four pixels in the center of the image that are not affected by the hypothesis propagation. Figure 2.7 summarize the pipeline.

2.5.3 libfreenect2 without outlier rejection

This pipeline uses the same phase unwrapping method as the libfreenect2 outlier rejection pipeline. However, there is no outlier rejection and as a result there will
be no undetected pixels except for the saturated ones. It will be used to compare the phase unwrapping methods without losing detected pixels during processing. The pipeline is illustrated in figure 2.8.

### 2.5.4 Microsoft Kinect SDK v2

The algorithms used in *Microsoft Kinect SDK v2* to produce the depth values are unknown. However, by looking at the output depth values one can conclude that this pipeline also contains an outlier rejection scheme.
2.6 Implementation

The proposed pipeline was implemented by modifying the *libfreenect2*\textsuperscript{1} code for depth calculations using OpenCL [15] for GPU acceleration. Running the proposed pipeline on a NVIDIA GeForce GTX 760 GPU, the frame rate for the depth calculations was above 80 Hz, which was roughly 3 times slower than the original *libfreenect2* code, although well over the frame rate of the Kinect v2 device at 30 fps [5]. Almost all of the extra time is spent in the propagation step of the algorithm. Further optimization in this part of the implementation could speed up the process significantly.

The *libfreenect2* with outlier rejection is just the original *libfreenect2* code, while the one without outlier rejection is slightly modified by removing the outlier rejection steps from the code.

\footnotesize{\textsuperscript{1}As in *opencl_depth_packet_processor.cl* Mar 25 2015 commit: baa692910a09}
If we want a well performing 3D reconstruction from a set of 2 dimensional image points we need to calibrate the cameras that produce the images. In order to map points in the depth image to color values in the RGB image we need to find the relative pose between the IR camera and the RGB camera. This can be done by performing an additional calibration step that involves both cameras simultaneously. The Kinect v2 already provides calibration parameters. These parameters could be compared with the parameters extracted from this calibration step. In this chapter we will explain how this can be done using well known image processing algorithms. The *OpenCV* library [4] was used in the implementations.

### 3.1 The pinhole camera model

The pinhole camera model implies that points in a 3 dimensional world are mapped onto the image plane of the camera. This mapping consists of a rigid transformation, consisting of a rotation $\mathbf{R}$ and a translation $\mathbf{t}$, into a camera centered coordinate system. The transformed point is then projected onto the image plane. The points on the image plane are then mapped to pixels by a linear transformation. This transformation is often referred to the intrinsic camera matrix, here noted $\mathbf{A}$, containing the intrinsic camera parameters. Since $\mathbf{A}$ is camera specific it needs to be extracted through a calibration procedure, for instance the Zhang calibration method [24]. Additionally, the lens of the camera distorts the image before it enters pixel plane. This distortion can be modeled as a nonlinear function $f(\mathbf{y}, \mathbf{k})$, where $\mathbf{y}$ is a point in the image plane and $\mathbf{k}$ is the set of distortion parameters, which maps image coordinates to other image coordinates. The function $f(\mathbf{y}, \mathbf{k})$ can be constructed in various ways. The parameters in $\mathbf{k}$ are unknown and can be found through a calibration procedure along with the intrinsic camera parameters $\mathbf{A}$. The mapping of a point $\mathbf{X}_w$ in the world, to a point $\mathbf{y}$ in the
image in image, or pixel coordinates can be modeled in the following way using homogeneous coordinates:

\[ X = [R|t]X_w \]  

(3.1)

The point \( X_w \) is transformed to the camera coordinate system and projected onto the image plane. The vector \( X \) is \( 3 \times 1 \) and contains the homogeneous coordinates of the projected point.

\[ x_p = X/X(3) \]  

(3.2)

Here \( x_p \) is normalized to canonical form meaning that the whole vector \( x \) is divided by its third element. The first two elements of \( x_p \) now represent the coordinates of the projected point in the image plane.

\[ x_{dist} = f(x_p, k) \]  

(3.3)

Distortion is applied and \( x_{dist} \) now contains the distorted image coordinates. In the end the distorted points \( x_{dist} \) is mapped to the pixel plane.

\[ y = Ax_{dist} \]  

(3.4)

\[ A = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \]  

(3.5)

The normalized xy-coordinates of \( y \) contain the pixel coordinates.

\[ y_{pixel} = \frac{y(1,2)}{y(3)} \]  

(3.6)

Here, the two first elements of \( y \) is divided with the third element and will correspond to the pixel coordinates.

If we want to map \( y \) to \( X_w \), we basically need to do the above sequence of mappings backwards. The objective is to find \( A, f \) and \( k \) which are unknown camera specific properties.

Without the distortion \( f \), the mapping is linear and we can construct the camera matrix \( C \):

\[ C = A[R|t]. \]  

(3.7)

### 3.1.1 Distortion models

There are several distortion models that handle different kinds distortions. The most common ones handle radial and tangential distortion. Here, the models available in the OpenCV library [4] and the atan model [6] were tested and evaluated. The models in OpenCV can be expressed in the following way:
3.2 Single camera calibration

\[ \mathbf{x}_p = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \]  

(3.8)

and

\[ \mathbf{x}_{dist} = \begin{pmatrix} u_{dist} \\ v_{dist} \end{pmatrix} \]  

(3.9)

\[ u_{dist} = u \cdot \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + 2 k_7 \cdot u \cdot v + k_8 \cdot (r^2 + 2 u^2)}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} \]  

(3.10)

\[ v_{dist} = v \cdot \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + k_7 \cdot (r^2 + 2 v^2) + 2 k_8 \cdot u \cdot v}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} \]  

(3.11)

where

\[ r = \sqrt{u^2 + v^2} \]  

(3.12)

The parameters \( \{k_i\} \) are unknown. The quotient part of the model describes the radial distortion and the rest decides the tangential part of the model. In OpenCV it is possible to set each of the parameters to zero and in that way neglect parts of the model.

The atan model [6] describes a fish-eye distortion and can be expressed in the following way:

\[ r_n = \sqrt{(u - c_1)^2 + (v - c_2)^2} \]  

(3.13)

\[ \phi = \text{atan2}(v - c_2, u - c_1) \]  

(3.14)

\[ r = \frac{\arctan(r_n \cdot \gamma)}{\gamma} \]  

(3.15)

\[ u_{dist} = c_1 + r \cdot \cos(\phi) \]  

(3.16)

\[ v_{dist} = c_2 + r \cdot \sin(\phi) \]  

(3.17)

Here \( c = [c_1, c_2] \) is the distortion center and is a free unknown parameter as is \( \gamma \). The advantage of the atan model is that it only has 3 unknown parameters and that it is analytically invertable. This makes it fast to use when an image is undistorted. The OpenCV models are in general not analytically invertible and must be approximately inverted using iterative methods.
3.2 Single camera calibration

The simplest and most commonly used way of calibrating single cameras is using a planar chessboard pattern. The world coordinates system is fixed in the pattern and correspondences in the image can be found automatically by using corner detection and pattern recognition algorithms. Snapshots from the camera are taken while the pattern is moved and rotated. From these images the unknown parameters can be constrained. Initially no distortion is assumed. The Zhang calibration [24] method is used to initialize the $A$ matrix. Now the iterative Perspective-n-Point algorithm (PnP) can be used to find the camera poses for each image. This gives an initial guess for $R$ and $t$ for each camera pose defined in the chessboard pattern fixed coordinate system. Each observed projection $y_i$ of the points $X_i$ can be estimated using equations 3.1-3.4 for one of the distortion models in section 3.1.1. From this the following cost function can be defined:

$$x_{i,j} = A f ([R_j|t_j]X_i, k)$$ (3.18)
$$\hat{y}_{i,j} = \frac{x_{i,j} \cdot xy}{x_{i,j} \cdot z}$$ (3.19)
$$\epsilon_{i,j} = y_{i,j} - \hat{y}_{i,j}$$ (3.20)

where $\epsilon_{i,j}$ is the residual error of the projection of point $i$ in image $j$. 

*Figure 3.1: OpenCV chessboard pattern[3].*
3.3 Joint camera calibration

The two cameras in the Kinect can be calibrated separately using the algorithm explained in section 3.2. Those parameters can be used to initialize the combined calibration with both cameras. If we make sure that all chessboard points are visible in both cameras we can transform points from one camera to the other. Figure 3.2 shows one image pair were the chessboard is visible in both cameras.

Since the cameras are fixed with respect to each other we know that this transformation is constant. From here on we denote the IR camera as $camera_I$ and the color camera as $camera_C$. The transformation of points in the world to $camera_I$...

$$\varepsilon = [\varepsilon_{1,1}, \varepsilon_{2,1}, ..., \varepsilon_{M,N}]$$ (3.21)

Here $\varepsilon$ contains the residual errors for all M points in all N images. The objective is to minimize $\|\varepsilon\|^2$ with respect to all camera poses $R_i$ and $t_i$, the intrinsic camera matrix $A$ and the distortion parameters $k$ of the selected distortion model. The norm $\|\varepsilon\|^2$ can be minimized iteratively using the Levenberg-Marquardt algorithm over all the free parameters. The output of the iterations will be a refined set of parameters from which we can extract $A$ and $k$.
is $T_I$ and to $camera_C$ is $T_C$. The transformation from $camera_I$ to $camera_C$ is the transformation $T_{ItoC}$ and it transforms points from the $camera_I$ centered coordinate system to the $camera_C$ centered coordinate system.

$$T_CX = T_{ItoC}T_I X \quad (3.22)$$

this gives,

$$T_{ItoC} = T_CT_I^{-1} \quad (3.23)$$

A rigid transformation in a homogenous coordinate system can be expressed as

$$T = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \quad (3.24)$$

and

$$T^{-1} = \begin{pmatrix} R^T & -R^T t \\ 0 & 1 \end{pmatrix} \quad (3.25)$$

which gives

$$T_{ItoC} = \begin{pmatrix} R_{ItoC} & t_{ItoC} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_C R_I^T & -R_C R_I^T t_I + t_C \\ 0 & 1 \end{pmatrix} \quad (3.26)$$

We can then construct a similar cost function to the equations (3.20) and (3.21). $R_{ItoC}$ and $t_{ItoC}$ can be initialized by using equation (3.26) and known values on $R_I$, $t_I$, $R_C$ and $t_C$ from the previous initialization.

$$x_{I_{i,j}} = A_I f_I([R_{Ij}|t_{Ij}|X_i,k_I]) \quad (3.27)$$

$$\hat{y}_{I_{i,j}} = \frac{x_{I_{i,j}.xy}}{x_{I_{i,j}.z}} \quad (3.28)$$

$$\epsilon_{I_{i,j}} = y_{I_{i,j}} - \hat{y}_{I_{i,j}} \quad (3.29)$$

$$\epsilon_I = [\epsilon_{I1,1}, \epsilon_{I2,1}, ..., \epsilon_{IM,N}] \quad (3.30)$$

$$x_{C_{i,j}} = A_C f_C([R_{ItoC_{j}|t_{ItoC_{j}}|T_I X_i,k_C}) \quad (3.31)$$

$$\hat{y}_{C_{i,j}} = \frac{x_{C_{i,j}.xy}}{x_{C_{i,j}.z}} \quad (3.32)$$

$$\epsilon_{i,j} = y_{C_{i,j}} - \hat{y}_{C_{i,j}} \quad (3.33)$$

$$\epsilon_C = [\epsilon_{C1,1}, \epsilon_{C2,1}, ..., \epsilon_{CM,N}] \quad (3.34)$$
\[ \epsilon = [\epsilon_C, \epsilon_I] \quad (3.35) \]

Like in section 3.2 we minimize \( \|\epsilon\|^2 \), here with respect to \( R_{ij}, t_{ij}, R_{itoC} \) and \( t_{itoC} \), the intrinsic camera matrix \( A_I \) and \( A_C \) and the distortion parameters \( k_I \) and \( k_C \). The output is a refined set of camera parameters and also the transformation from \textit{camera}_I to \textit{camera}_C. Note that this calibration process is analogous to transforming points from \textit{camera}_C to \textit{camera}_I.

### 3.4 Rectify image

The parameters from the calibration could now be used to rectify the images. This is done by mapping the undistorted image plane to the distorted image plane.

\[ y = A f (A^{-1} y_{undist}, k) \quad (3.36) \]

The coordinates represented by \( y \) are in general not integers so in order to extract the value at the location \( y \) bi-linear interpolation is used.

### 3.5 Distance to depth

The time-of-flight technology used in the Kinect v2 provides a distance map, meaning that every pixel measures the distance from the camera center to the perceived object. More convenient would be to have a depth map instead i.e, the distance along the z-axis.

\[ n, x_s, x_p, z, d \]

\[ x^2 + y^2 + z^2 = 1 \]

\textbf{Figure 3.3:} Illustration of point \( x \) projected into the plane \( z=1 \) and the unit sphere \( x^2 + y^2 + z^2 = 1 \).
Figure 3.3 illustrates a 3D point $\mathbf{x}$ that is projected into a camera at pixel $\mathbf{y}$. The distance $d$ to the point is measured. This distance $d$ is converted to the depth $z$, which is the distance between the camera center to the plane that contains $\mathbf{x}$ and is parallel to the image plane. The projection line from $\mathbf{x}$ to the camera center $\mathbf{n}$ travels through the point $\mathbf{x}_p$ on the image plane and the point $\mathbf{x}_s$ on the unit sphere.

The measured pixel $\mathbf{y}$ is transformed to image plane coordinates by solving the system of equations for $\mathbf{x}_p$

$$\mathbf{y} = \mathbf{A} f(\mathbf{x}_p, \mathbf{k}) \quad (3.37)$$

where

$$\mathbf{A} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (3.38)$$

$$\mathbf{x}_p = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \quad (3.39)$$

and $f$ is the function that models the lens distortion with distortion parameters $\mathbf{k}$. Since $f$ is non-linear in general it is not analytically solvable.

In the end we want to find the relation between $d$ and $z$. There are several solutions to this problem. I will present one that uses the following relations.

$$\mathbf{x} = d \cdot \mathbf{x}_s \quad (3.40)$$

$$\mathbf{x} = z \cdot \mathbf{x}_p \quad (3.41)$$

which implies

$$z \cdot \mathbf{x}_p = d \cdot \mathbf{x}_s \quad (3.42)$$

Since $\mathbf{x}_s$ and $\mathbf{x}_p$ lie on the same line through the camera center $\mathbf{n}$ we get $\mathbf{x}_s$ if we normalize $\mathbf{x}_p$. This implies

$$\mathbf{x}_s = \frac{\mathbf{x}_p}{||\mathbf{x}_p||} \quad (3.43)$$

Using this and equation 3.42 we get

$$z = \frac{d}{||\mathbf{x}_p||} \quad (3.44)$$

where

$$||\mathbf{x}_p|| = \sqrt{u^2 + v^2 + 1} \quad (3.45)$$
3.6 Depth to color mapping

From the undistorted depth points in the depth image its corresponding 3D points could be reconstructed using:

\[ X = z \cdot A^{-1} y_{\text{undist}} \quad (3.46) \]

These points could now be mapped into the RGB camera frame using relative pose transformation \( T_{ItoC} \). The transformed point could now be projected into the RGB image and a color value could be extracted.
In computer vision research, one of the most interesting field of application for the Kinect v2 would be 3D reconstruction. By using the camera calibration described in chapter 3 and the depth images constructed using the algorithms described in chapter 2 we can now perform the 3D reconstruction and visualization methods we will discuss in this chapter.

4.1 Single frame 3D Reconstruction

A way to illustrate the data produced by the Kinect v2 is to create an RGB-D mesh from the depth image and colors provided by the RGB camera. The RGB data are sampled by computing the 3D coordinates of the depth pixels and then projecting them onto the RGB image plane as explained in section 3.6. This requires the intrinsic parameters of the device to be known, and for this we use the calibration setup from section 5.1.2 with the IR camera parameters provided by the Kinect v2 device fixed. Before generating the 3D point cloud, the depth image is pre-processed by a bilateral filter applied to the inverse depth. This allows larger variations in distant points than in points close to the camera, which is reasonable since more uncertainty is introduced at larger depths. Triangles are then created by connecting neighbouring depth pixels as illustrated in figure 4.1.

These triangles could then be rendered using OpenGL [12]. In order to clean up the mesh, triangles with points where the depths $z_0$, $z_1$, $z_2$ are not fulfilling the following constraint:

$$\frac{\|z_i - z_j\|}{\text{median}(z_0, z_1, z_2)} < T, i, j \in [0, 1, 2], i \neq j$$

(4.1)

are suppressed. In this thesis we set $T = 0.05$. Consequently only triangles with vertices close to each other will be rendered.
4.2 Multi-frame 3D reconstruction

The depth images produced by the Kinect can be used to produce 3D models. However, the images are noisy and the models are incomplete since only the surfaces that are visible from the camera can be 3D reconstructed. In order to obtain complete 3D models of observed objects information from several viewpoints can be fused into a single representation. The algorithm Kinect fusion, described in [19], can produce such 3D models with very high accuracy. In this section a description of this algorithm will be provided.

4.2.1 Kinect fusion

Kinect fusion is a simultaneous localization and mapping (SLAM) algorithm. This means that the observed model could be updated while simultaneously finding the pose of the camera. The algorithm is designed such that it can be performed in parallel on a GPU. How this is achieved is explained in detail in [19] and will here be summarized.

Using the mapping explained in section 3.6 image points $x$ in the depth image are 3D reconstructed and transformed into world coordinates using $T_w$. The matrix $T_w$ is a rigid transformation and can be defined, using homogeneous coordinate representation, as:

$$T_w = \begin{pmatrix} R_{w,k} & t_w \\ 0 & 1 \end{pmatrix}$$  

The 3D reconstructed points constitute a list of vertices $V$. Together with the corresponding vertices of two neighboring pixels, the vertex $V(x)$ span a plane
with the following normal:

\[
N(x) = (V(u + 1, v) - V(u, v)) \times (V(u, v + 1) - V(u, v))
\]

where \( x = [u, v] \).

The model is stored as a truncated signed distance function (TSDF) defined as:

\[
F(x) = \begin{cases} 
d(x, \Omega), & \text{if } x \in \Omega \\
-d(x, \Omega), & \text{if } x \in \Omega^c
\end{cases}
\]

where \( d(x, \Omega) = \inf_{y \in \Omega} (x, y) \) (4.5)

The TSDF is represented on the GPU as a voxel grid. Each voxel represents a position \( x \) in 3D space and contains the signed distance \( F(x) \) to the closest surface point of the model and a weight \( W(x) \) that corresponds to its certainty. Measurements detected outside the voxel grid are ignored. By using ray casting \[21\] from a virtual camera pose, the surface could be predicted and converted to estimated vertices, \( \hat{V}(x) \), and surface normals, \( \hat{N}(x) \).

For each new depth frame \( k \), the vertices \( V_k(x) \) and the normals \( N_k(x) \) are constructed. They are aligned against the predicted vertices and normals from the previous camera pose \( \hat{V}_{k-1}(x) \) and \( \hat{N}_{k-1}(x) \).

The vertices are matched using the following constraints:

\[
\Omega(x) \neq 0 \text{ iff } \begin{cases} 
\|T_w,k V_k(x) - \hat{V}_{k-1}(x)\| < \epsilon_d \\
\langle R_w N_k(x), \hat{N}_{k-1}(x) \rangle < \epsilon_\theta
\end{cases}
\]

where \( \epsilon_d \) and \( \epsilon_\theta \) are designed thresholds. The objective is to find a \( T_{w,k} \) such that

\[
E(T_{w,k}) = \sum_{\Omega(x) \neq 0} \|T_{w,k} V_k(x) - \hat{V}_{k-1}(x)\|^2
\]

is minimized. This implies that the new depth map is aligned with the model. However, \( E(T_{w,k}) \) is a non-linear function and \( \arg\min\limits_{T_{w,k}} E(T_{w,k}) \) cannot in general be found analytically. In \[19\] \( \arg\min\limits_{T_{w,k}} E(T_{w,k}) \) is found iteratively were \( T_{w,k} \) is initialized with \( T_{w,k-1} \).

\[
T_{w,k}^{i+1} = T_{inc} T_{w,k}^i
\]

where \( T_{w,k}^0 = T_{w,k-1} \) and

\[
T_{inc} = \begin{pmatrix} R_{inc} & t_{inc} \\ 0 & 1 \end{pmatrix},
\]

\[
t_{inc} = (t_x, t_y, t_z)^T
\]
The rotation is assumed to be small between iterations, thus the rotation matrix $R_{inc}$ could be approximated as:

$$R_{inc} = \begin{pmatrix}
1 & \alpha & -\gamma \\
-\alpha & 1 & \beta \\
\gamma & -\beta & 1
\end{pmatrix} \quad (4.14)$$

The parameters to be found are put into the vector $p = (\beta, \gamma, \alpha, t_x, t_y, t_z)^T$. Expression (4.9) can now be written as:

$$E(p) = \sum_{\Omega(x)\neq 0} \|\hat{N}_{k-1}(x)^T (G(x)p + T_{w,k}^i V_k(x) - \hat{V}_{k-1}(x))\|^2_2 \quad (4.15)$$

where $G(x) = \left[-T_{w,k}^i V_k(x) \right]_x I_{3 \times 3} \quad (4.16)$

By taking the derivative of $E(p)$ w.r.t $p$ and set to zero the following system of equations can be constructed:

$$\sum_{\Omega(x)\neq 0} A^T A p = A^T \hat{N}_{k-1}(x)^T (T_{w,k}^i V_k(x) - \hat{V}_{k-1}(x)) \quad (4.17)$$

where $A^T = G(x)^T \hat{N}_{k-1}(x) \quad (4.18)$

The transformation $T_{inc}$ can be derived from least squares solution of (4.17) for $p$. The procedure is then repeated for $T_{w,k}^{i+1}$ until the alignment is completed.

The aligned vertex set $T_{w,k} V_{w,k}(x)$ can be converted to a TSDF $F_k(x)$. The global TSDF holding the model is then updated using a running weighted average. For visualization, a mesh is constructed from the TSDF using the marching cubes algorithm [16].

### 4.2.2 Implementation

In this thesis The Point Cloud Library (PCL) [22] implementation Kinfu was used to perform Kinect fusion. The communication with the Kinect v2 sensor as well as the depth image calculations were performed using our modified version of the libfreenect2 [20] as explained in section 2.6. This enabled us to compare the different pipelines described in section 2.5.
In this chapter the results from the calibration, the depth data processing, and 3D reconstruction will be presented. To evaluate the performance of the four pipelines described in section 2.5, they were tested in a set of experiments. Firstly, the outputs from the pipelines will be presented and discussed qualitatively and secondly their performances will be compared quantitatively.

5.1 Calibration Results

The single camera calibration is done by taking snapshots of a chessboard pattern in different poses. The chessboard pattern was put close to the camera so that all parts of the image was taken into account when the parameters were optimized. The models tested were the OpenCV models, see equation 3.10 to 3.11, and the atan-model, see equation 3.13 to 3.17. The OpenCV models were tested using different subsets of \( \{k_i\} \). The optimizations of parameters were performed by using the implementation of Levenberg-Marquart in [17]. One model for the IR camera and one model for the color camera can be chosen for performing the joint camera calibration.

There are also camera parameters that could be downloaded from the Kinect device. These parameters are then compared with our models on a set of images that are not used in the calibration.

The quality of the calibration is presented as the root mean square (RMS) of the vector \( \epsilon \), see 3.21 and 3.35.

\[
RMS = \sqrt{\frac{\|\epsilon\|^2}{N}}
\]

(5.1)

where \( N \) is the length of \( \epsilon \).
In the calibration steps the RMS only presents how well the model performs on the training data set. The evaluation step shows how well the models perform on data set not used in the calibration. The exact parameter values of the calibration models are presented in appendix C.

### 5.1.1 Single camera calibration results

The calibration procedure was setup with a $7 \times 10$ chessboard pattern and 42 different camera poses.

<table>
<thead>
<tr>
<th>model</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$, $k_1 - k_2$</td>
<td>0.21500</td>
</tr>
<tr>
<td>$A$, $k_1 - k_3$</td>
<td>0.2086</td>
</tr>
<tr>
<td>$A$, $k_1 - k_2, k_7 - k_8$</td>
<td>0.2087</td>
</tr>
<tr>
<td>$A$, $k_1 - k_3, k_7 - k_8$</td>
<td>0.2026</td>
</tr>
<tr>
<td>$A$, $k_1 - k_6$</td>
<td>0.2085</td>
</tr>
<tr>
<td>$A$, $k_1 - k_8$</td>
<td>0.2026</td>
</tr>
<tr>
<td>$A$, atan model</td>
<td>0.3964</td>
</tr>
</tbody>
</table>

**Table 5.1:** Ir camera calibration result

The results shows that the RMS is much smaller for the OpenCV models compared to the atan model. From this result we can reject the atan model as a suitable distortion model for the Kinect v2 cameras. There are small differences between the different OpenCV in terms of RMS. From that we can conclude that there is little to gain using a large set of distortion parameters.

### 5.1.2 Joint camera calibration results

Here the calibration is set up so that the transformation from the IR camera to the Color camera is found. The data set consists of 29 different camera poses with images visible in both IR and color.

<table>
<thead>
<tr>
<th>IR model</th>
<th>Color model</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$, $k_1 - k_3, k_7 - k_8$</td>
<td>$A$, $k_1 - k_3, k_7 - k_8$</td>
<td>0.275769</td>
</tr>
</tbody>
</table>

**Table 5.3:** Two camera calibration variable IR model parameters
5.1 Calibration Results

Transformation from IR to color:

\[
R = \begin{bmatrix}
0.9999 & 0.01050 & -0.002012 \\
-0.01050 & 0.9999 & 0.002074 \\
0.002034 & -0.002053 & 1.000
\end{bmatrix}
\] (5.2)

\[
t = \begin{bmatrix}
5.203 \\
0.03391 \\
-0.01520
\end{bmatrix}
\] (5.3)

This was also tried using the default IR camera parameters read from the device and fixating them during optimization. This gave the following result:

<table>
<thead>
<tr>
<th>IR model</th>
<th>Color model</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A, k_1 - k_3 ) (kinect v2 default)</td>
<td>( A, k_1 - k_3 )</td>
<td>0.363075</td>
</tr>
</tbody>
</table>

Table 5.4: Two camera calibration fixed IR model parameters

Transformation from IR to color:

\[
R = \begin{bmatrix}
0.9999 & 0.01074 & -0.004110 \\
-0.01073 & 0.9999 & 0.0007936 \\
0.004118 & -0.0007494 & 1.000
\end{bmatrix}
\] (5.4)

\[
t = \begin{bmatrix}
5.203 \\
0.01852 \\
-0.09242
\end{bmatrix}
\] (5.5)

We see that the relative rotation between the cameras is very small and that the translation is approximately horizontal. Since the coordinates of the chessboard used in the calibration is measured in centimeters the translation vector tells us that the color camera is placed 5.2 cm to the right seen from the IR camera center. This is a feasible result that corresponds with measurements done by hand.

5.1.3 Calibration Evaluation

The models found in the previous sections as well as the Kinect v2 default parameters is compared on a dataset containing images that were not used during the calibration step. These images contain chessboard patterns which are detected. Together with the model parameters the camera pose is calculated for each image using an iterative PnP method. The RMS values are calculated from the deviations of the projected chessboard points to the measured ones.
Table 5.5: Calibration evaluation result. * means that the parameters after the calibration step with two cameras were used.

From the evaluation RMS values we see that the errors are small, which means that the calibration is well performed. We see a slight improvement for the parameters after the combined cameras calibration step, which was expected since each point is measured from two views. The default Kinect v2 parameters perform only slightly worse than our parameters for the IR camera. Thus, these can be used in the confidence that they perform well.

The relative pose was evaluating by mapping the chessboard pattern into the IR camera frame and then to the color camera image using the transformation between the cameras obtained from the calibration step in section 5.1.2. The transformed chessboard point is then projected into the rectified RGB image.

\[
\begin{align*}
X_{Cj} &= A_C[R_{ItoC}|t_{ItoC}]T_I X_i \\
x_{C_{i,j}} &= X_{Cj}.xy \\
\epsilon_{i,j} &= y_{C_{i,j}} - x_{C_{i,j}}
\end{align*}
\]

The pose of the IR camera, \(T_I\), was estimated by using the iterative PnP algorithm. The residual \(\epsilon_{i,j}\) is the deviations of the estimated projection of the chessboard point \(X_i\) to the observed chessboard point \(y_{C_{i,j}}\) in image \(j\).

Figures 5.1 to 5.2 shows the regularized histograms of \(\epsilon_{i,j}\) using the parameters and relative pose from the combined camera calibration with fixed default IR parameters. The experiments were performed on a dataset with 31 infrared and RGB image pairs containing a chessboard pattern with \(7 \times 5\) corners. We see that the error is less than 1 pixel in the RGB image. This relatively small error indicates that the calibration is well performed.

5.1.4 Depth to color mapping evaluation

Using a similar experiment as the ones in the calibration step the depth to color mapping could be evaluated. The chessboard pattern is placed such that it is visible in both the IR and the color camera. Both the IR image and the color image are rectified, using the method described in section 3.4. Corresponding chessboard corners are recognized in the images. Then the depth values \(z\) from
5.1 Calibration Results

Figure 5.1: Norm of the projection error in the color image of chessboard points mapped from the IR image.

Figure 5.2: Error distribution of the projection in the color image of chessboard points mapped from the IR image in the Left: x direction Right: y direction.

the located chessboard point in the IR camera is extracted. The corresponding point in the RGB image could now be extracted using the method described in section 3.6. The same dataset and camera parameters were used as in the relative pose evaluation described in section 5.1.3, including the rotation in (5.4) and the translation in (5.5). In this dataset the depth of the chessboard points varied within the interval 80 to 180 centimeters from the Kinect device. We had to keep the chessboard quite close to the camera in order detect the pattern in the infrared image.

Figures 5.3 to 5.4 show that the errors in depth to color mapping are in general small on the distances that the chessboard was held. The errors have the same magnitude as the error in the relative pose evaluation test in section 5.1.3 so it is not possible to tell whether it is the errors in the depth that is the main cause.
Looking at 5.4 there seems to be a bias in the deviations in the x direction. From where these biases have their origin is difficult to conclude. One bias effect could come from the higher frequencies components discussed in section 2.2.

**Figure 5.3:** Norm of the projection error in the color image of chessboard points mapped from the depth image.

**Figure 5.4:** Error distribution of the projection in the color image of chessboard points mapped from the depth image in the Left: x direction Right: y direction.

### 5.2 Depth images

The depth images produced by the Kinect v2 are often noisy. When the scene contains surfaces with low reflectance the impact of the noise becomes stronger due to the lower signal strength. This can cause the phase unwrapping procedure to select the wrong phase which often leads to a large depth error in the end. Low
5.2 Depth images

Figure 5.5: RGB image of an office room taken by the Kinect v2.

signal strength can for instance be caused by the reflectance of the material, the distance to the sensor or if the surface has a steep angle towards the camera.

A typical result of the depth decoding is shown in figure 5.6. The depth images is taken in the room shown in figure 5.5. Dark pixels are near and bright pixels are far from the camera.

Each image has been produced by the output depth from the previously described pipelines in section 2.5. The images produced by the proposed pipeline and the libfreenect2 pipelines with and without outlier rejection are generated from the same input data. The image produced by the Microsoft Kinect SDK v2 pipeline had to be generated from different input data, as the API does not support externally logged raw sensor data. As all methods were run under the same conditions (the same scene, view, and sensor, and shortly after each other) this is still a fair comparison.

In the depth images the undetected pixels are green and pixels where the phase unwrapping is incorrect are usually very bright compared to their neighbors. As can be seen, the depth image processed by the proposed method has fewer erroneous pixels, especially in areas with low reflectance, such as the upper left image corner, and the computer screen to the left. However, it should be noted that a large quantity of the pixels that are seemingly correct in the lower right image of figure 5.6 are in fact only unwrapped to a depth closer to its surroundings. This may trick the eye of the beholder that the spatial propagation method produces less noisy images than it actually does.

For the spatial propagation method, the most relevant contender is libfreenect2 without outlier rejection (the two lower images in figure 5.6). Clearly the proposed method has a higher number of pixels with correct depth values. On the downside, the spatial propagation occasionally generates clusters of bad measurements as can be seen as dense white areas in the lower right image in figure 5.6. The clustering artifacts can be removed by limiting the number of hypotheses evaluated in (2.41), thus reducing the allowed range. This is demonstrated in figure 5.7, where only 13 hypotheses are evaluated, instead of 30, reducing the
maximal distance from 18.75m to 8.75m.

Interestingly the results from the *Microsoft Kinect SDK v2* are very similar to the *libfreenect2* method with outlier rejection in terms of number of correctly phase unwrapped pixels (see the two upper images in figure 5.6). These two methods mainly differ by the fact that *Microsoft Kinect SDK v2* does not have any wrongly unwrapped pixels but only undetected ones. This is probably a result of a different maximal distance threshold. Due to vignetting effects in the infrared camera of the Kinect v2 device, the amplitude is lower near the image corner than in the image centre. This introduces more noise and therefore more uncertainty in the distance values.
5.3 Single frame 3D reconstruction results

Figure 5.7: Spatial propagation with varied depth range. Left: Output depth for spatial propagation using 13 hypotheses only (i.e. maximal distance 8.75 m). Right: using all 30 hypotheses (i.e. maximal distance is 18.75m). The input data is the same as in figure 5.6.

5.2.1 Propagation evaluation

As explained in section 2.4, the proposed spatial propagation method ranks a set of phase unwrapping hypothesis. It then propagates vertically and horizontally through the depth image to select which of hypothesis to be used in the output image. The left image of figure 5.8 shows the resulting depth if only the highest ranked hypothesis in terms of minimizing the cost function in (2.41), is used in the output image. The result is actually very similar to the libfreenect2 pipeline without outlier rejection, see the lower left image of figure 5.6. Comparison with the right image of figure 5.8, which is the output of the full spatial propagation method, shows that the propagation reduces the number of erroneous pixels. The effect of the propagation step is that the corresponding pixels in the left and right image that are not equal have changed to the hypothesis with the second highest rank.

5.3 Single frame 3D reconstruction results

Another qualitative result can be produced using a colored mesh accordingly with the method described in section 4.1. Figure 5.9 shows an example of such a colored mesh taken in an office room. Gray areas represent depth points invisible for the RGB camera. We see that the colors are mapped onto corresponding depth points and it shows that the calibration performs well.

Results from environments with larger depths are shown in figure 5.10. We can see that the proposed method produces a larger coverage of triangles than libfreenect2 without outlier rejection. We can also see that the Kinect v2 can sense detailed depths far beyond the 4.2 m limit that was defined in the specifications
5.4 Multi-frame 3D reconstruction results

The Kinect fusion algorithm, as it is described in section 4.2.1, is designed to run in real-time. Figure 5.11 shows the resulting mesh of a room after a realtime scan using the spatial propagation method to produce the depth images. We can see that the mesh contains fine details and the coverage is almost complete except for occluded areas, for instance behind the chair and the computer screen. Scan
5.4 Multi-frame 3D reconstruction results

Figure 5.10: Top: RGB-D visualization of a climbing wall, bottom: RGB-D visualization of a room with the back wall at 18 meters from the sensor. Left: using libfreenect2 without outlier rejection, Right: using proposed phase unwrapping. Frustum ends at $d = 18.75$ m, and is shown as black lines. The plane at $d = 4.2$ m is also shown for reference (this is the Kinect v2 limit according to the specification [5]).

sessions like these were in general simple to perform, meaning that the we do not lose track of the camera pose.

To compare the different pipelines from section 2.5 Kinect fusion was run using logged data. Figures 5.13-5.15 shows the resulting mesh renderings produced by the different pipelines with and without rectified depth images. The data set is a scan of the floor with 3 boxes a chair and some other objects, see figure 5.12.

We see that there is not a big difference in the quality of the models between the pipelines. The libfreenect2 pipeline with outlier rejection has the best coverage as can be seen in areas in the lower right corner of the lower images. The proposed pipeline has a slightly better coverage compared with the libfreenect2 without outlier projection. Common for all pipelines is that the performance is better with the rectified depth images, as can be seen on the chair.
In contrast to the room in figure 5.11, this data set was a bit harder to scan. In many of the sessions the camera pose was falsely estimated leading to drifting and destruction of the mesh. The big box in the middle of the meshes is very distorted for all pipelines regardless of rectification of depth images. A plausible explanation to these problems is that the received signal from the objects is disturbed by reflections on the floor. This issue was discussed previously in section 2.2.1.

To examine this we modified the scene slightly by placing a carpet, which has less reflectivity than the floor, underneath the big box in the center. We also placed the big box on top of another box to reduce the reflections even more, see figure 5.16.

The resulting meshes using libfreenect2 pipeline with outlier rejection and rectified depth images from similar scans as the one in figures 5.13-5.15 are shown in figure 5.17.

We see a definite improvement in the sense that the big box is less distorted. Also the unwanted drifting of the camera pose did not occur during the Kinect fusion sessions on these scenes. The corresponding meshes using the spatial propagation pipeline and the libfreenect2 without outlier rejection are shown in figures A.1-A.4 in appendix A.

**Figure 5.11:** Kinect fusion mesh created in real time using the spatial propagation pipeline seen from two views.

**Figure 5.12:** RGB image of boxes on a floor, used in a Kinect fusion scan experiment.
5.5 Plane experiment

For a quantitative assessment we set the Kinect to record a planar floor at different angles and distances. We used in total 6 different poses, see figure 5.18, and table 5.6.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>d (mm)</td>
<td>2045</td>
<td>2100</td>
<td>2228</td>
<td>2323</td>
<td>2484</td>
<td>3024</td>
</tr>
<tr>
<td>α(°)</td>
<td>54.2</td>
<td>51.0</td>
<td>50.7</td>
<td>46.7</td>
<td>41.7</td>
<td>31.3</td>
</tr>
</tbody>
</table>

**Table 5.6:** Plane experiment camera pose properties. See figure 5.18 for explanation of d and α.

For each pose, the raw measurements from 100 depth frames, were logged. The four different pipelines along with the proposed pipeline without propagation were then used to create depth images from the logged data. As previously noted, the Microsoft SDK used a separate data log, but the same sensor, pose, and scene. From the depth images 3D points could be reconstructed using the intrinsic camera parameters provided by the Kinect v2 device. The lens distortion was neglected since it is small and did not affect the results in this experiment notably. In each frame a plane was then robustly fitted to the data using RANSAC [8]. The plane was then used to classify all depth values as either inliers or outliers, based
on the distance to the plane. The mean inlier percentage over the hundred images for each dataset is presented in figure 5.19. The maximum and minimum deviation from the mean is shown as error bars but is nearly invisible since these values were very small. The camera plane properties $d$ and $\alpha$ were also extracted from the estimated plane, see table 5.6.

The results from the plane estimations are shown in figure 5.19. As can be seen, spatial propagation consistently improves the coverage. The improvement is between 2-4%, depending on the angle to the plane $\alpha$, (see figure 5.18). These improvements appear to be larger for more complex scenes as in figure 5.6 and 5.10. Also, the results from the proposed pipeline without propagation seem to be consistent or even slightly improved compared with the libfreenect without outlier rejection. This is in accordance with the qualitative results displayed in figure 5.6 and it shows that using only the highest ranked hypothesis in the proposed pipeline is a strong contender to the libfreenect pipeline.
Figure 5.15: Kinect fusion mesh: Top: The spatial propagation pipeline. Bottom: with rectified depth images.

Figure 5.16: RGB image of a box Left: with a carpet Right: top of another box.
Figure 5.17: Kinect fusion mesh using libfreenect2 pipeline with outlier rejection and rectified depth images of boxes on a floor. Top: with a carpet Bottom: with the big box on top of another box.

Figure 5.18: Plane experiment setup where the camera pose is aimed with the angle $\alpha$ against the floor. The field-of-view of the camera is within the dashed lines. The distance along the camera view vector from the camera center to the floor is $d$. 
Figure 5.19: Inlier rate in plane estimate experiments. Inliers are defined as points within a distance of 30 cm from the plane.
Conclusion

In this thesis we have provided a description of how the Kinect v2 sensor works. This has included the theory behind the time-of-flight principle and some of the drawbacks such as reflection disturbances and higher order harmonics of the emitter signal. We also provided a framework of how to calibrate a system of two cameras such as the Kinect v2 sensor in order to map depth point into the colors image. From this the observed environment could be visualized as a colored mesh. The experiments in section 5.3 showed that the Kinect v2 indeed can be useful for 3D reconstruction of scenes with depths up to 18.75 meters. This is a much larger depth than the specifications indicated, which was 4.2 meters. Properties like this will be interesting for 3D reconstruction applications and other computer vision fields.

We have also shown that the depth processing methods available have room for improvement and we have in some aspects succeeded in doing just that by utilizing the raw time-of-flight measurements. The spatial propagation method that was proposed in this thesis has shown potential in improving the depth data processing procedure compared with the ones in libfreenect2 and Microsoft Kinect SDK v2. However, it has some negative properties like the clustering artifacts discussed in section 5.2. This is due to the fact that the propagation procedure for each iteration only performs regularization locally. For future work, an improvement to the proposed extension would be to select the hypothesis for each pixel that regularizes the depth image globally. The challenge would be to formulate such a solution to this problem and implement it to run in real time.

The Kinect fusion results indicate that the reflection disturbances is a problematic property introduced by the time-of-flight principle. For future work it would be interesting to add extensions to the depth calculation framework to remove these disturbances as the one in [7] discussed in section 2.2.1. Additionally, the Kinect fusion results show that an outlier rejection method is important for
such applications. For future work it would be interesting to integrate an outlier rejector with the spatial propagation pipeline. This outlier rejection method could for instance be similar to the ones used in libfreenect2. However, it might not be as strict since the experiments have shown that there seems to be a higher tolerance to noise after the propagation step.
Appendix
This appendix contains the results from the additional Kinect fusion sessions to illustrate the issues caused by reflections from the floor.

**Figure A.1:** Kinect fusion mesh of boxes on a floor with a carpet using *libfreenect2* pipeline without outlier rejection and with rectified depth images.
Figure A.2: Kinect fusion mesh of boxes on a floor with a carpet using the spatial propagation pipeline with rectified depth images.

Figure A.3: Kinect fusion mesh of boxes on a floor with the big box on top of another box using libfreenect2 pipeline without outlier rejection and with rectified depth images.

Figure A.4: Kinect fusion mesh of boxes on a floor with the big box on top of another box using the spatial propagation pipeline with rectified depth images.
The open source library *libfreenect2* attempts to process the data stream produced by the Kinect v2 device. Each data packet in the data stream is interpreted either as an JPEG-compressed RGB image or an 10 images sized infrared frame. The RGB image is unpacked using the library *turbojpeg* and could then viewed. The infrared images are processed using a number of steps explained below.

Here follows a overview of what the code in *libfreenect2* is doing to calculate the depth for each pixel:

1. Establish communication with the Kinect device and download $p_0$ table.

2. Wait for frame containing the 9 images.

3. Decode frame data and produce 9 voltage values, $V_{m,n}$ for each pixel from the images. Each pixel produces a 11 bit value and the frame is transferred as a byte array with $298496 (=512*424*11/8)$ elements for each of the 9 images. This is converted to 16 bits via a look-up table that contains all possible values for a 11 bit signed integer.

4. Calculate one phase $\phi_m$ for each frequency $f_m$ in accordance with equation 2.28 and include the $p_0$ table. This gives:

   $$\phi_m = \text{atan2}\left( -V_{m,1} \cdot \sin(p_0) - V_{m,2} \cdot \sin\left(p_0 + \frac{2\pi}{3}\right) - V_{m,3} \cdot \sin\left(p_0 + \frac{4\pi}{3}\right), \\
   V_{m,1} \cdot \cos(p_0) + V_{m,2} \cdot \cos\left(p_0 + \frac{2\pi}{3}\right) + V_{m,3} \cdot \cos\left(p_0 + \frac{4\pi}{3}\right) \right)$$

   (B.1)

5. Combine the three phases to produce a true unambiguous phase as explained in section 2.3.
6. Calculate the distance value from the phase. This is done by is using two tables $Z(x)$ and $X(x)$ according to equations (B.2)-(B.5). These tables are taken from a memory dump from one of the libfreenect2 developers experiments with the Kinect v2 SDK and probably need to be calibrated for each camera.

$$d_z = Z(x) \cdot \phi; \quad \text{(B.2)}$$
$$d_{max} = \phi \cdot 2083.333 \quad \text{(B.3)}$$
$$x_{mult} = \frac{X(x) \cdot 90}{d_{max}^2 \cdot 2 \cdot 8192} \quad \text{(B.4)}$$
$$d_{final} = \frac{d_z}{-d_z \cdot x_{mult} + 1} \quad \text{(B.5)}$$


In addition libfreenect2 also provides some filtering steps, removing outliers and using a bilateral filtering.

The amplitude is calculated in this way

$$A^2_m = \left( -V_{m,1} \cdot \sin(p_0) - V_{m,2} \cdot \sin\left( p_0 + \frac{2\pi}{3} \right) - V_{m,3} \cdot \sin\left( p_0 + \frac{4\pi}{3} \right) \right)^2 +$$
$$\left( V_{m,1} \cdot \cos(p_0) + V_{m,2} \cdot \cos\left( p_0 + \frac{2\pi}{3} \right) + V_{m,3} \cdot \cos\left( p_0 + \frac{4\pi}{3} \right) \right)^2 \quad \text{(B.6)}$$

and from that an grayscale infrared image is calculated as

$$IR = \frac{A_0 + A_1 + A_2}{3} \quad \text{(B.7)}$$
The vector $\mathbf{k} = (k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8)$ holds the distortion parameters for the OpenCV models.

### C.1 IR camera

#### C.1.1 OpenCV Models

\[
\mathbf{k} = (0.08445, -0.1970, 0, 0, 0, 0.0008168, 0.001248) \quad (C.1)
\]

\[
\begin{pmatrix}
363.225 & 0 & 256.624 \\
0 & 363.338 & 206.25 \\
0 & 0 & 1
\end{pmatrix}
\]

\[\text{RMS} = 0.2087\]

\[
\mathbf{k} = (0.08410, -0.1963, 0, 0, 0, 0, 0, 0) \quad (C.3)
\]

\[
\begin{pmatrix}
363.291 & 0 & 255.327 \\
0 & 363.396 & 205.551 \\
0 & 0 & 1
\end{pmatrix}
\]

\[\text{RMS} = 0.2150\]

\[
\mathbf{k} = (0.1100, -0.3068, 0.1299, 0, 0, 0, 0.0008858, 0.001153) \quad (C.5)
\]

\[
\begin{pmatrix}
362.759 & 0 & 256.54 \\
0 & 362.875 & 206.296 \\
0 & 0 & 1
\end{pmatrix}
\]

\[\text{RMS} = 0.2150\]
\[ \text{RMS} = 0.2026 \]

\[ \mathbf{k} = (0.1106, -0.3097, 0.1336, 0, 0, 0, 0, 0) \quad (C.7) \]

\[ A = \begin{pmatrix} 362.793 & 0 & 255.329 \\ 0 & 362.901 & 205.531 \\ 0 & 0 & 1 \end{pmatrix} \quad (C.8) \]

\[ \text{RMS} = 0.2086 \]

\[ \mathbf{k} = (0.05672, -0.1499, -0.02608, -0.05283, 0.1563, -0.1571, 0.0008946, 0.001151) \quad (C.9) \]

\[ A = \begin{pmatrix} 362.764 & 0 & 256.538 \\ 0 & 362.88 & 206.31 \\ 0 & 0 & 1 \end{pmatrix} \quad (C.10) \]

\[ \text{RMS} = 0.2026 \]

\[ \mathbf{k} = (0.07648, -0.03294, -0.4045, -0.03571, 0.2852, -0.5503, 0, 0) \quad (C.11) \]

\[ A = \begin{pmatrix} 362.764 & 0 & 255.329 \\ 0 & 362.87 & 205.53 \\ 0 & 0 & 1 \end{pmatrix} \quad (C.12) \]

\[ \text{RMS} = 0.2085 \]

**Atan Model**

\[ c = [255.999, 212.003] \]

\[ \gamma = 1.871 \times 10^{-11} \]

\[ A = \begin{pmatrix} 363.403 & 0 & 256.531 \\ 0 & 363.194 & 206.316 \\ 0 & 0 & 1 \end{pmatrix} \quad (C.13) \]

\[ \text{RMS} = 0.3964 \]

### C.2 Color camera

**OpenCV Models**

\[ \mathbf{k} = (0.04640, -0.05070, 0, 0, 0.0003281, 8.683 \times 10^{-5}, 0, 0) \quad (C.14) \]
C.2 Color camera

\[
A = \begin{pmatrix}
1052.89 & 0 & 976.379 \\
0 & 1053.51 & 536.343 \\
0 & 0 & 1
\end{pmatrix}
\] (C.15)

\[RMS = 0.5670\]

\[
k = (0.04644, -0.0509, 0, 0, 0, 0, 0, 0)\] (C.16)

\[
A = \begin{pmatrix}
1052.89 & 0 & 976.379 \\
0 & 1053.51 & 536.343 \\
0 & 0 & 1
\end{pmatrix}
\] (C.17)

\[RMS = 0.5677\]

\[
k = (0.04670, -0.05190, 0.001253, 0, 0, 0, 0.0003283, 8.9764 e - 05)\] (C.18)

\[
A = \begin{pmatrix}
1052.89 & 0 & 976.646 \\
0 & 1053.53 & 537.144 \\
0 & 0 & 1
\end{pmatrix}
\] (C.19)

\[RMS = 0.5670\]

\[
k = (0.04666, -0.05177, 0.0008985, 0, 0, 0, 0, 0)\] (C.20)

\[
A = \begin{pmatrix}
1052.87 & 0 & 976.379 \\
0 & 1053.49 & 536.343 \\
0 & 0 & 1
\end{pmatrix}
\] (C.21)

\[RMS = 0.5677\]

\[
k = (0.02330, 0.02846, 0.2780, 0.07365, 0.2814, -0.02185, 0.0003266, 9.586 e - 05)\] (C.22)

\[
A = \begin{pmatrix}
1052.97 & 0 & 976.662 \\
0 & 1053.62 & 537.146 \\
0 & 0 & 1
\end{pmatrix}
\] (C.23)

\[RMS = 0.5666\]

\[
k = (0.02302, 0.03235, 0.2849, -0.02206, 0.07715, 0.2890, 0, 0)\] (C.24)

\[
A = \begin{pmatrix}
1052.95 & 0 & 976.378 \\
0 & 1053.58 & 536.35 \\
0 & 0 & 1
\end{pmatrix}
\] (C.25)

\[RMS = 0.5674\]
Atan Model

\( c = [960, 540] \)
\( \gamma = -1.31612e-07 \)

\[
A = \begin{pmatrix}
1052.89 & 0 & 976.646 \\
0 & 1053.53 & 537.144 \\
0 & 0 & 1
\end{pmatrix}
\]  \hspace{1cm} (C.26)

\( \text{RMS} = 2.042 \)

C.2.1 Joint cameras calibration Results

Here the calibration is set up so that the transformation from the IR camera to the color camera is found. The data set consists of 29 different camera poses with images visible in both IR and color.

Initializing IR camera model:

\[
k = (0.1100, -0.3068, 0.0008858, 0.001153, 0.1299, 0, 0, 0) \]  \hspace{1cm} (C.27)

\[
A = \begin{pmatrix}
362.759 & 0 & 256.54 \\
0 & 362.875 & 206.296 \\
0 & 0 & 1
\end{pmatrix}
\]  \hspace{1cm} (C.28)

Initializing color camera model:

\[
k = (0.04670, -0.05190, 0.0003283, 8.976e-05, 0.001253, 0, 0, 0) \]  \hspace{1cm} (C.29)

\[
A = \begin{pmatrix}
1052.89 & 0 & 976.646 \\
0 & 1053.53 & 537.144 \\
0 & 0 & 1
\end{pmatrix}
\]  \hspace{1cm} (C.30)

IR camera model after calibration:

\[
k = (0.1102, -0.3183, -0.0002991, 0.0009301, 0.1691, 0, 0, 0) \]  \hspace{1cm} (C.31)

\[
A = \begin{pmatrix}
363.751 & 0 & 256.19 \\
0 & 363.734 & 204.853 \\
0 & 0 & 1
\end{pmatrix}
\]  \hspace{1cm} (C.32)

Color camera model after calibration:

\[
k = (0.04212, -0.03566, -0.0007891, -0.0001752, -0.01292, 0, 0, 0) \]  \hspace{1cm} (C.33)
\[ A = \begin{bmatrix} 1054.3 & 0 & 975.082 \\ 0 & 1055.76 & 531.547 \\ 0 & 0 & 1 \end{bmatrix} \] (C.34)

\[ \text{RMS} = 0.2758 \]

Transformation from IR to color:

\[ R = \begin{bmatrix} 0.9999 & 0.01049 & -0.002012 \\ -0.01049 & 0.9999 & 0.002074 \\ 0.002034 & -0.002053 & 1.000 \end{bmatrix} \] (C.35)

\[ \bar{t} = \begin{bmatrix} 5.203 \\ 0.03392 \\ -0.01520 \end{bmatrix} \] (C.36)

This was also tried with using the default IR parameters and fixate them during optimization. This gave the following result:

IR camera model:

\[ k = (0.08996, -0.2655, 0, 0, 0.09011, 0, 0, 0) \] (C.37)

\[ A = \begin{bmatrix} 365.5653 & 0 & 255.5502 \\ 0 & 365.5653 & 204.9938 \\ 0 & 0 & 1 \end{bmatrix} \] (C.38)

Color camera model after calibration:

\[ k = (0.04179, -0.035052, 0, 0, -0.0120636, 0, 0, 0) \] (C.39)

\[ A = \begin{bmatrix} 1055.44 & 0 & 975.436 \\ 0 & 1057.12 & 533.595 \\ 0 & 0 & 1 \end{bmatrix} \] (C.40)

err after optimization: 0.363075

\[ R = \begin{bmatrix} 0.9999 & 0.01073 & -0.004110 \\ -0.01073 & 0.9999 & 0.0007936 \\ 0.004118 & -0.0007494 & 1.000 \end{bmatrix} \] (C.41)

\[ t = \begin{bmatrix} 5.203 \\ 0.01852 \\ -0.09242 \end{bmatrix} \] (C.42)
C.2.2 Evaluation

Initializing IR camera model:

\[ k = (0.1100, -0.3068, 0.0008858, 0.001153, 0.1299, 0, 0, 0) \]  \hspace{1cm} (C.43)

\[ A = \begin{pmatrix} 362.759 & 0 & 256.54 \\ 0 & 362.875 & 206.296 \\ 0 & 0 & 1 \end{pmatrix} \]  \hspace{1cm} (C.44)

\[ RMS = 0.3340 \]

IR camera model after combined calibration:

\[ k = (0.1102, -0.3183, -0.0002991, 0.0009301, 0.1691, 0, 0, 0) \]  \hspace{1cm} (C.45)

\[ A = \begin{pmatrix} 363.751 & 0 & 256.19 \\ 0 & 363.734 & 204.853 \\ 0 & 0 & 1 \end{pmatrix} \]  \hspace{1cm} (C.46)

\[ RMS = 0.3336 \]

Kinect default IR camera model:

\[ k = (0.08997, -0.2655, 0, 0, 0.09011, 0, 0, 0) \]  \hspace{1cm} (C.47)

\[ A = \begin{pmatrix} 365.5653 & 0 & 255.5502 \\ 0 & 365.5653 & 204.9938 \\ 0 & 0 & 1 \end{pmatrix} \]  \hspace{1cm} (C.48)

\[ RMS = 0.3430 \]

Initializing Color camera model:

\[ k = (0.04670, -0.05189, 0.0003283, 8.976e^{-5}, 0.001253, 0, 0, 0) \]  \hspace{1cm} (C.49)

\[ A = \begin{pmatrix} 1052.89 & 0 & 976.646 \\ 0 & 1053.53 & 537.144 \\ 0 & 0 & 1 \end{pmatrix} \]  \hspace{1cm} (C.50)

\[ RMS = 0.2512 \]

Color camera model after combined calibration:
\[ \mathbf{k} = (0.04212, -0.03566, -0.0007891, -0.0001754, -0.01292, 0, 0, 0) \]  
\[ \mathbf{A} = \begin{pmatrix} 1054.3 & 0 & 975.082 \\ 0 & 1055.76 & 531.547 \\ 0 & 0 & 1 \end{pmatrix} \]  

\( RMS = 0.2276 \)

Kinect default color camera model:

\[ \mathbf{k} = (0, 0, 0, 0, 0, 0, 0, 0) \]
\[ \mathbf{A} = \begin{pmatrix} 1081.3721 & 0 & 959.5 \\ 0 & 1081.3721 & 539.5 \\ 0 & 0 & 1 \end{pmatrix} \]  

\( RMS = 0.9187 \)


Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Felix Järemo Lawin