FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Extracting Sensor Specific Noise Models

BACHELORARBEIT

zur Erlangung des akademischen Grades

**Bachelor of Science**

im Rahmen des Studiums

**Medieninformatik und Visual Computing**

eingereicht von

**Nicolas Grossmann**
Matrikelnummer 1325103

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dr. Michael Wimmer
Mitwirkung: Dr. Stefan Ohrhallinger

Wien, 28. August 2017

| Nicolas Grossmann | Michael Wimmer |
|---|---|

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Extracting Sensor Specific Noise Models

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Bachelor of Science**

in

**Media Informatics and Visual Computing**

by

**Nicolas Grossmann**
Registration Number 1325103

to the Faculty of Informatics

at the TU Wien

Advisor: Dr. Michael Wimmer
Assistance: Dr. Stefan Ohrhallinger

Vienna, 28th August, 2017

| Nicolas Grossmann | Michael Wimmer |
|---|---|

# Erklärung zur Verfassung der Arbeit

Nicolas Grossmann
Heiligenstädter Straße 167-171/1/6

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. August 2017

Nicolas Grossmann

# Danksagung

Ich möchte mich an dieser Stelle bei meinen Betreuern Dr. Stefan Ohrhallinger und Dr. Michael Wimmer für ihre Hilfe und ihr ausführliches Feedback während meiner Arbeit bedanken. Des Weiteren möchte ich mich bei dem Institut für Computergraphik und Algorithmen für die Bereitstellung von Materialien und Räumlichkeiten bedanken.

Allem voran gilt mein Dank meinem Kollegen Thomas Köppel, welcher mich während dieser Arbeit tatkräftig unterstützt hat, sei es nun tagelanges Messen, das Bauen von Versuchsteilen oder auch das Korrekturlesen dieser Arbeit.

# Acknowledgements

I would like to express my gratitude to my supervisors Dr. Stefan Ohrhallinger and Dr. Michael Wimmer for their great support and extensive feedback during the work on this thesis. Furthermore, I want to thank the Institute of Computer Graphics and Algorithms for providing us with the materials and locations for our measurements.

But most of all I want to thank my colleague Thomas Köppel, who worked together with me in many parts of the project, be it days of measuring, crafting setup parts or proofreading this thesis.

# Kurzfassung

Im Laufe der letzten Jahre kamen immer mehr Tiefensensoren für den privaten Gebrauch auf den Markt, wie die Kinect oder die seit kurzem erhältliche Phab2Pro. In dieser Arbeit analysieren wir den durchschnittlichen Messfehler dieser Sensoren in axialer und lateraler Richtung. Als Teil eines zweiteiligen Projektes werden wir in dieser Arbeit die Abhängigkeit des Messfehlers von der Distanz und dem Rotationswinkel zwischen Sensor und Objekt betrachten. In weiterer Folge werden wir zwei Modelle erstellen, das erste aus den in dieser Arbeit erhobenen Daten und das zweite, aus der Kombination des vorherigen Models mit dem eines Kollegen, die erstellten Modelle können in weiterer Folge für die Verbesserung von Tiefenbildern verwendet werden.

# Abstract

With the growing number of consumer-oriented depth sensors like the Kinect or the newly released Phab2Pro, the question of how precise these sensors are arises. In this thesis we want to evaluate the average noise in the generated depth measurements in both the axial direction and the lateral directions. As part of a two-part project this thesis will view the noise's development with varying distance and angle. Finally, we will present and evaluate two models describing the noise behavior, with the first being derived from solely this thesis' measurements and the second one being a combination of the previous model and a model of a colleague. This derived models can be used in a post-processing step to filter the generated depth images.

# Contents

CHAPTER 1

# Introduction

## 1.1 Background

Over the last years several customer-oriented depth sensors arrived on the market. Depth sensors allow taking images of their surrounding world, like normal cameras, but instead of recording color their images picture the distance between the camera and the sensor. This information is particularly useful in several different applications, as it allows easier segmentation and recognition of objects in scenes.

While many different cameras were already able to deliver depth information, their prices were pretty high and they were mostly used in industrial applications. It was not until around 2010 that the first consumer-affordable cameras came on the market. The most widely known being the KinectV1, which is a depth sensor combined with a color camera that was used as a new way to play on the Xbox360. This sensor allowed people to play games without the need of a typical controller, by just using their bodies to make certain movements that controlled their game actions. The good perception and capabilities of the KinectV1 resulted in an increased interest of business and science in this cheap depth sensor. All of this led Microsoft to releasing not only an SDK for the Kinect that allowed developers to create own programs for the depth sensor, but also a second, more advanced sensor version the KinectV2. This resulted in many papers dealing with the use of the Kinect in robotics [ElHY12], human- [XCA11], gesture- [RMYZ11] and object-detection [KE12], medical visualization [GPC11] and many more.

Just recently, another big development in the field of consumer depth sensors happened: the first smartphone utilizing depth perception was released. While some producers already developed small depth sensors that were built into laptops, for example, none of them were integrated in a hand-held device. The Phab2Pro, from Lenovo, uses a sensor similar to the one in the KinectV2, but in a much smaller size, which allows the sensor to be built into a usual smartphone. The depth data of the sensor is processed by a Google

(a) KinectV2 (image taken from [Cor14])

(b) Phab2Pro (image taken from [Ltd])

Figure 1.1: Images of the two sensors

product called Project Tango, which uses several techniques including machine learning to extract information out of the depth data and make the results usable for apps. Most of these apps are based on an augmented reality in which the depth information acts as an anchor, tying virtual and physical reality closer together, for example, by enabling occlusions through objects.

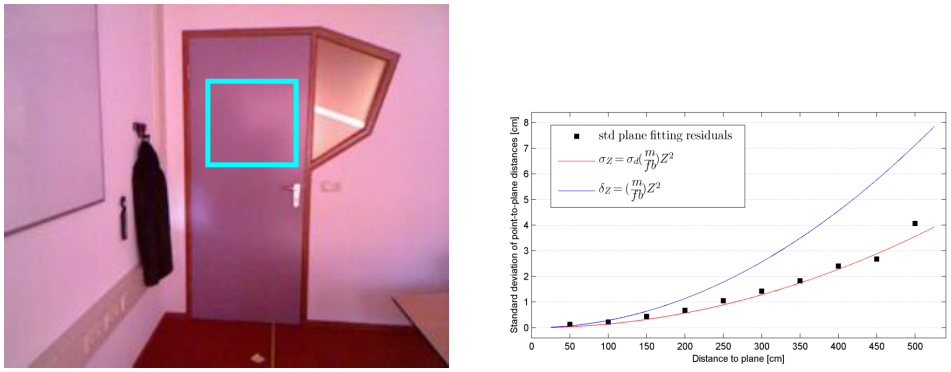
## 1.2 Previous Work

The problem with most of these affordable depth sensors is that they are prone to having a higher amount of noise (measurement errors) in their generated depth images than more expensive models. Errors in the depth measurements may lead to problems in the processing pipelines of different applications. Surfaces and objects reconstructed from measurements may have lots of small bumps or miss some details. To minimize the effects of noise, many papers have analyzed it and tried to come up with ways to describe its behavior.

### 1.2.1 Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications

One of the first papers that deals with this kind of problems of the KinectV1 was written by Khoshelham et al. [KE12]. The goal of the paper was to analyze the quality of the gathered depth data in indoor situations. Their two major contributions were firstly a calibration procedure for the sensor and secondly a statistical analysis of the random errors in depth measurements and the depth resolution. A laser scanner was used as a ground truth measure in the paper.

By comparing the differences between the KinectV1 point cloud to the laser scanner point cloud, they showed the necessity of a calibration procedure. As shown in the paper, a standard calibration procedure like it is used for color cameras can be used to calibrate the




(a) Image of the setup (b) Results of the noise measurements

Figure 1.2: Images taken from [KE12]

infrared camera in the sensor used for the depth perception. The statistical errors were calculated by measuring a flat surface (a door). A plane was fitted through the resulting point cloud and the difference between the plane positions and the measurements was calculated. The statistical parameters yielded the noise of the sensor.

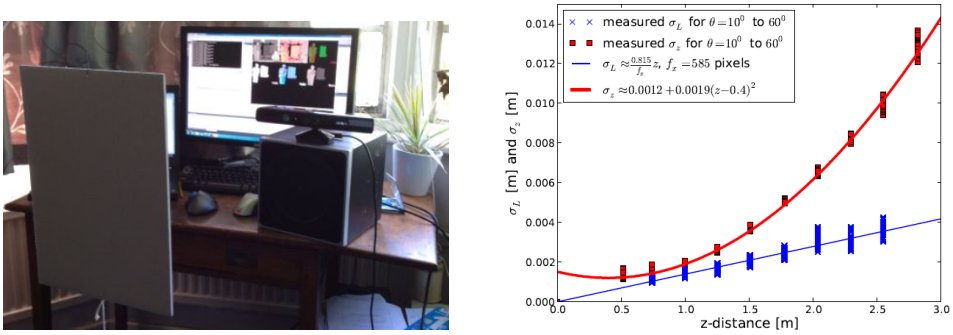
The result of the work was a calibration procedure combining the color images with the point cloud and a statistical model of the noise distribution for the KinectV1, modeling the circumstance that the depth noise grows with increasing distance. (images in Figure 1.2)

### 1.2.2 Modeling Kinect Sensor Noise for Improved 3D Reconstruction and Tracking

Based on the previous paper, Nguyen et al. [NIL12] developed a similar statistical model for the sensor noise of the KinectV1. Using this calculated model, they tried to show that 3D reconstruction can be improved by sensor specific noise models.

The model proposed by Khoshelham et al. 1.2.1 was extended in this paper to not only consider the distance between sensor and camera, but also the surface angle. Furthermore, apart from noise along the image axis, lateral noise orthogonal to the image axis was measured.

The statistical models, calculated from these measurements, were used to filter depth images produced by the KinectV1 and to enhance the standard KinectFusion pipeline, which is used for 3D reconstruction. Their enhanced reconstruction method was able to extract finer surface details than the standard one. They also showed that their extended KinectFusion version has improved tracking capabilities when compared to its original. (images in Figure 1.3)




(a) Image of the setup (b) Results of the noise measurements

Figure 1.3: Images taken from [NIL12]

### 1.2.3 KinectV2 for Mobile Robot Navigation: Evaluation and Modeling

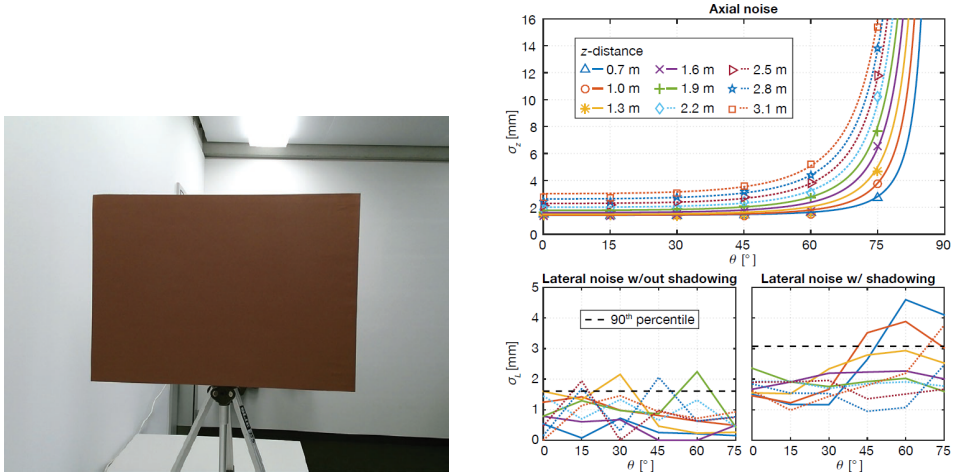
One of the first papers using the noise measurement techniques from the previously mentioned papers for the KinectV2 was written by Fankhauser et al. [FBR+15]. Their main goal was to evaluate the possibility of using the KinectV2 in mobile robot navigation in indoor and outdoor scenarios.

Because the KinectV2 was relatively new at this time, they also proposed a calibration procedure for improved depth measurements and evaluated the quality of the measurements for different distances. They also repeated the measurement procedure from Nguyen et al. [NIL12], both indoors and outdoors. From these measurement results, two noise models were derived for the axial noise. The indoor model was similar to the previous models, while the noise was generally lower for the KinectV2. The outdoor model used an additional parameter for its calculation, the angle of the sun. The lateral noise of the KinectV2 indicated no specific noise function and was only estimated by a constant factor. (images in Figure 1.4)

While this paper did not contain a practical component using the calculated models, they delivered a detailed basis for further projects using the KinectV2, by explaining their suitability for problems in the field robot navigation (e.g. close objects recognition, temperature problems or overcast illumination).

## 1.3 Scope of this Thesis

This thesis is one part of a two-part project, where two different types of models are calculated defining the measured sensor noise of the KinectV2 and the Phab2Pro. The focus of this thesis is mainly guided by the previous papers in this field, which measure noise on planar surfaces and statistically model the measured results. Our goal is to evaluate the depth sensor noise of the newly released Phab2Pro and repeat the previously executed measurements for the KinectV2. Additionally, our work extends the previous




(a) Image of the setup

(b) Results of the noise measurements

Figure 1.4: Images taken from [FBR+15]

setups – which only extracted lateral noise in the x-axis – to also extract noise from the y-axis. We intend to generate empirical models describing the noise for all three directions for both cameras: one model that is based on the target's distance and surface rotation and another one that combines the previous model with the one of a colleague, whose model uses the pixel coordinates and the distance as parameters. Finally, the calculated models are validated by measuring a well-defined object.

## 1.4 Outline

This thesis is structured in such a way that in Chapter 2 mainly the sensor technology behind both cameras is described, including their error sources. Furthermore, this chapter should give an overview of the theoretical basis on which this kind of sensor can be evaluated and how its noise can be described. In Chapter 3, the experimental setups we used are described in detail, as well as the processing steps we took to extract the valuable data from the measurements and how we calculated the noise models. After that, we present the results in Chapter 4. This includes a short analysis of the sensor error, the calculated noise measures with their corresponding models, as well as the axial and lateral noise models for both sensors. In this chapter we also define an extended noise model, which is created by combining the model from this thesis with the one of a colleague's thesis, which uses pixel coordinates and distance as model parameters. Furthermore, we try to evaluate the quality of our models by comparing them to preceding models in the literature and by evaluating them with a simple experiment. Finally, in Chapter 5, we discuss the results of our work and how it could be improved.

CHAPTER 2

# Theory

## 2.1 Sensors

Our work focuses on the evaluation of the measurement quality of two specific depth sensors, the KinectV2 and the Phab2Pro, both of which are time-of-flight sensors. As these sensors are commonly used, an understanding of their typical measurement problems is necessary to support applications using them. In this chapter, we will introduce each sensor, explain the principles of their underlying technology and discuss their advantages and disadvantages. It will also be explained, how the measurement errors can be statistically analyzed and modeled.

### 2.1.1 KinectV2

The Kinect is a series of depth sensors from Microsoft, which were released for consumers. Its primary purpose is to act as a control device for several games on the Xbox One console. Many of which are developed around concepts that require an active form of player involvement like dancing or sports games. The Kinect supports this by not only generating a depth map of the scene, but also by actively detecting persons and their poses. [Cor13]

Microsoft released two versions of the Kinect, the first one in 2010 and the second more advanced one in 2013. This work focuses on the latter one. While both systems deliver similar outputs, in terms of data, both deliver color and depth images with additional player recognition information, their inner workings are fundamentally different. The KinectV1 uses the structured-light principle, which illuminates the scene with an infrared dot pattern to measure the dot displacement and calculate the depth. The KinectV2 uses a time-of-flight system for depth estimation, which utilizes the travel time of light impulses to estimate distances. [SLK15]

Originally, the Kinect did not have an open software development kit for non-commercial use, but with the growing interest and popularity of the depth sensor, Microsoft released an SDK that allowed accessing KinectV1 and later V2 data with own applications. This allowed many different companies and scientists to develop uses for the Kinect apart from gaming like advanced user interfaces [BBW+11], security measures [For] or robotics [FBR+15].

### 2.1.2 Phab2Pro

The second device used in this work is the Phab2Pro. Unlike the KinectV2, the Phab2Pro is not only a simple sensor but a whole smartphone with all of its functionalities. The Phab2Pro, produced by Lenovo, is the first commercially available smartphone that is capable of advanced depth recognition. The reason for this is that the Phab2Pro comes with an integrated time-of-flight sensor alongside its camera. [Ltd]

The processing part of the depth information is done by Google Project Tango, which is a framework for the development of depth-aware apps on Android devices. The framework delivers processed information to various specially developed apps. Most of these apps are games that use the depth information to create a virtual reality in the real world. This project from Google intends to bringing depth sensing technology to smartphones. In contrast to the Kinect for Windows SDK that is only able to handle sensor input from two devices, Google's Project Tango tries to be universally usable on every smartphone that has certain features. For example, the basic depth sensor can be everything from a structured-light sensor, to a time-of-flight sensor or even a stereo-vision based system. By encapsulating the underlying technology, the framework offers depth data in a universally applicable way to develop apps for different smartphones.

## 2.2 Errors

The general principle of time-of-flight cameras like the KinectV2 and the Phab2Pro is to utilize a complex sensor unit and a modulatable illumination unit, both normally operating in the infrared spectrum. The illumination unit sends out differently modulated light pulses into the scene, which are reflected from the contained objects. The reflected light is picked up by the sensor unit and based on its time-of-flight, the depth of the reflecting object can be estimated. Because of the extremely fast speed of light ($300\,000\,\mathrm{km\,s^{-1}}$) a camera that works in close range needs to work in nanoseconds.

This architecture has many advantages over other technologies: [SLK15]

- **Simple and Compact:** The camera itself does not have moving parts like the ones used in laser scanners. Furthermore, there is no need for a wide aperture because both its illumination and sensor unit can be close together, as opposed to stereo vision systems.

- **Speed:** Most time-of-flight cameras have high frame rates, some even working with more than 100 frames per second. The reason for this is that the depth measurements for all pixels are done at once. Furthermore, the processing of the measurements to calculate the depth is rather easy without the need of too complex algorithms. The rather high frame rates enable the use in real-time applications.

### 2.2.1 Error Sources

Like every technology, time-of-flight cameras have their own set of disadvantages caused by problematic lighting conditions: [SLK15]

- **Background Illumination:** Because most time-of-flight sensors use infrared light, background light, which is light not directly produced by the camera, has to be filtered before it interferes with the depth calculation. This problem is especially noticeable in outdoor situations. Different manufactures have developed use case specific ways to account for background illumination.
- **Reflections:** Surfaces reflecting the infrared light lead to wrong measurements. There are many reason for this problem like specular, highly reflective surfaces that may result in superimposition of measured values. In our work we observed that even diffuse surfaces turn into specular surfaces at extreme angles, leading to unwanted reflections (as seen in our experiments with surfaces at 70°).
- **Interference:** If multiple time-of-flight sensors are used in parallel, this may lead to interferences because the illumination unit of one camera might lead to false light measurements in the sensor of the other unit. There are several ways to face this problem like different modulations for each illumination unit.
- **Temperature:** As shown in the work from Breuer et al. [BBA14] the higher energy consumption and the resulting increase in temperature of some time-of-flight cameras like the KinectV2 can lead to a shifting in distance values during the device's warm up phase.
- **Systematic Error:** The depth images of the sensor are falsified by two types of systematic errors: calibration errors and wiggling errors. The first one is simply the result of an inadequate camera calibration like wrong assumptions about the principal point or the lens distortion. This type of error can be reduced by a standard camera calibration procedure. The second one is the result of small approximations in the measurement pipeline.

### 2.2.2 Error Model

This work mainly focuses on analyzing the systematic error (which remains after a calibration procedure) and its relation to the distance and rotation of an object. Our goal is to provide an error model describing this relation in mathematical terms. The

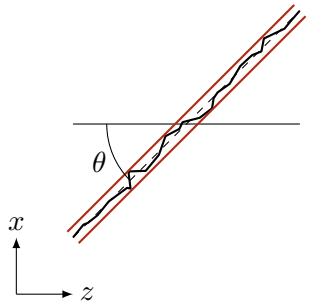



Figure 2.1: Axial noise along the face of the planar target

resulting model could in turn be used to enhance 3D reconstruction processes to resolve finer details with fewer frames.

To calculate such an error model we use a stochastic approach where the error is calculated based on the difference between measurement results and ground truth information. Other works have shown that this approach is suitable for time-of-flight sensors like the KinectV2 [FBR+15]. Based on the technical similarity of our two sensors, we assumed that this approach is also applicable for the Phab2Pro.

The systematic noise has two main components that need to be analyzed separately as previously shown [NIL12]:

### Axial Noise

The axial noise describes the depth measurement error along the z-axis. In general, it describes the average difference between the measured depth and the actual depth. This type of error is determined by assuming a flat surface that is perpendicular to the z-axis and measuring its distance. The modeled domain describing the ground truth is represented by a plane. The difference for each pixel to the plane would then in turn be the axial noise. [NIL12]

At this point it is important to consider the difference between the model domain and the real-world domain. The plane and its structure probably differ from the modeled plane because of tiny bumps in its structure and small setup errors in distance, rotation or orientation. The real-world domain is then measured with the noisy sensor. The resulting data can be described as lying in the measurement domain. The error is estimated from this domain. [CLDB14]

The real ground truth is unknown to a certain degree because obtaining it would require another more precise sensor that in turn would be prone to errors. [CLDB14] To evade this circular problem, we tried to generate a real-world domain whose errors are smaller than the tested sensors resolution and used a virtual ground truth model that is not too prone to setup distortions.

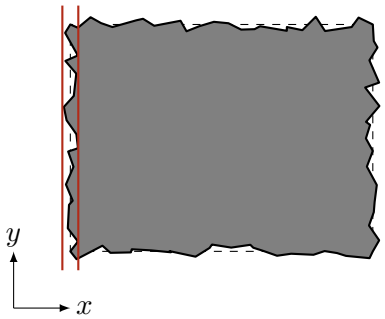



Figure 2.2: Lateral noise alongside the edge of the planar target

Furthermore, we assume that the sensor noise has a normal distribution. This is based on the previous works by Nguyen et al. [NIL12] and Fankhauser et al. [FBR+15] and our own measurements seen in Figure 4.3. The characterizing parameter of noise with this distribution is the standard deviation.

Considering all of this, the final procedure we used to extract the axial noise was as follows:

1. Placing a plane at a certain distance and rotation around the y-axis (verified by manual measuring)
2. Measuring the plane with the sensor
3. Extracting the point cloud of the plane
4. Fitting a plane based on a model equation to get the ground truth model
5. Determining the distance between each measured point and its modeled ground truth
6. Calculating the standard deviation of the measurements

### Lateral Noise

The lateral noise is a measure for the pixel deviation along the two image axes x and y. It describes how much the measured point is vertically or horizontally misplaced compared to the actual point on the object. To measure this, the plane is assumed to be perpendicular to the z-axis so that its border points are measured as straight lines. The difference between the presumed border line and the measured one is the lateral noise. This noise can be measured for the x- and the y-direction. [NIL12]

Similar to the axial noise, the real ground truth position of the point is uncertain and cannot be perfectly measured. To minimize small errors in the real-world domain, we tried to use a virtual ground truth model that fits the borders to the measured data.

Even though other works showed that the lateral noise for time-of-flight cameras is only partially described by a normal distribution, the standard deviation is still used as a measure because, as stated in Fankhauser et al. [FBR+15], many frameworks assume a Gaussian noise distribution.

CHAPTER 3

# Method

## 3.1 Experimental Setup

Because our work combines two separate statistical noise models we also need two separate test setups to measure noise. The first one is used to capture axial and lateral noise based on the distance and surface angle of the target, this test setup will be described in detail in this thesis. The second test setup is used to capture the axial and lateral noise based on the target distance and its position in the depth image. The latter setup can be read in detail in the work of my colleague. [Kö17]

### 3.1.1 Previous setups

As we mentioned before, previous papers measured the axial noise of depth cameras. Most of them used a setup consisting out of a moving depth sensor and a fixed planar target. One of the earliest works trying to quantify axial noise of a customer-oriented depth sensor is Khoshelham et al. [KE12]. Their experiment used a KinectV1 aimed at a door to measure its surface's distance. These measurements were then compared to a fitted plane perpendicular to the z-axis so that the standard deviation was the axial noise. Their measurements were repeated at increasing distances from the target. In doing so, they found out that the noise grows with increasing distance between sensor and target. Later works like Nguyen et al. [NIL12] also used a KinectV1 but with an improved version of this setup: a rotatable board. This allowed to not only extract the axial noise on its surface, but also the lateral noise along its edges. Furthermore, their noise model not only incorporated the target's distance but also the target's rotation. Both mentioned works showed that it is possible to measure the noise of a structured-light camera with this setup. The applicability of this setup to time-of-flight sensors like the one used in the KinectV2 or the Phab2Pro is shown by Fankhauser et al. [FBR+15]. One part of their work was to quantify the indoor and outdoor noise of a KinectV2 in axial and lateral direction. Their setup is largely similar to the one used by Nguyen

et al. [NIL12]. It consisted of a KinectV2 pointing towards a rotatable plane, taking measurements at different distances and angles. These measurements were used to create two noise models one for indoor and one for outdoor applications. This paper showed that the commonly used setup is able to measure axial and lateral noise not only for structured-light cameras, but also for time-of-flight cameras.

### 3.1.2 Our Setup

Our test setup was nearly identical to the ones used in Nguyen et al. [NIL12] and Fankhauser et al. [FBR+15], which were based on Khoshelham et al. [KE12]. As the target a wooden press board with a faint white surface was mounted onto a tripod. Although the rather high weight of the board had to be accounted for, lighter materials like card boards were no option because as our first tries showed, these lighter and thinner materials were too flexible and would bend under their own weight. This would have led to measurement errors. The surface color and faintness were chosen to minimize the error through reflection. Finally, all parts from the tripod to the plane were carefully adjusted to account for different error sources like an uneven floor. To extract the angular characteristics of the noise, the target plane was vertically rotated from 0° to 80° in intervals of 10°. To ensure a valid angle, a rotation device, which is used in photography to rotate a camera to a certain angle, was placed between the tripod and the board. Furthermore, two differently sized press boards were used to keep the measured area in the depth images roughly at the same size at different distances. Between 0.9 m and 1.9 m a board with the size of 0.4 m × 0.3 m was used and between 2 m and 3 m a board with 0.6 m × 0.4 m was used. A larger board with the size of 1.0 m × 0.8 m was not used due to it being too heavy for the tripod.

The camera in our setup was placed on a wooden construct that in turn was standing on a desk in front of the fixed target. Similar to the target, the camera and its desk were carefully adjusted to be evenly leveled and parallel to each other to minimize errors due to the test setup. To extract the distance dependent noise characteristics, the camera was moved from 0.9 m to 3.0 m in intervals of 0.1 m. The mentioned range does not cover the minimal and maximal sensing range perfectly (0.5 m to 4.5 m for the KinectV2) because the noise at this extreme ranges was too high to extract any information. To ensure that the distances were valid, a tape ruler was used for the initial positioning. From there on out, a fixed tape ruler between the desk and the wooden construct allowed us to make precise backward steps and to keep the camera in a parallel position to the target. The camera was positioned in such a way that the y-axis of the depth image overlapped with the rotation axis of the target and the x-axis overlapped with the lower border of the board. The latter one was used to keep the perspectively distorted border of the board parallel to the image border, which helps with the automatic board recognition (as described in 3.4.2).

Before the measurements were started, the camera was running for at least 30 minutes to reduce temperature dependent errors [BBA14]. After that, the test procedure started at 0.9 m at 0°. At each setting of distance $z$ and angle $\theta$ 200 images were taken. 100 images

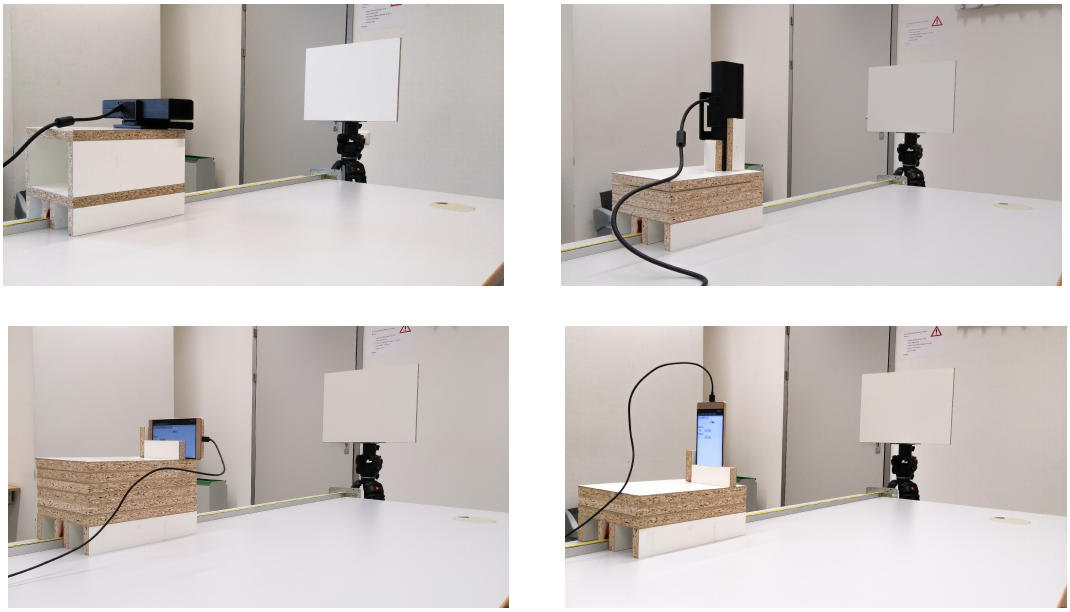


Figure 3.1: Images of the setups for the two different sensors in both horizontal and vertical position

with the camera in a horizontal position to extract the axial noise and the lateral noise in x-direction and 100 images with the camera in a vertical position to extract the lateral noise in y-direction. This process was repeated until a depth map or a point cloud was captured for all combinations of distances and angles.

## 3.2 Data Extraction

To ensure that our project can calculate the noise for any arbitrary depth sensor, we separated our measurement software into two parts: a device dependent program to save measured data in the common format `.csv` and a device independent evaluation toolkit programmed in MATLAB. For nearly all calculations in our project like the finding of a region of interest (see section 3.4.2) or the measuring of the axial and the lateral noise (see section 3.5), only two types of data were needed: a depth image and a point cloud. Depth images are also called depth maps and can be seen as images, but instead of saving color information at each position, they save the distance to the nearest object for each pixel. They can easily be analyzed with standard image processing techniques and filters. The objects in the scene are perspectively distorted and have to be reshaped and repositioned before they can be measured. Point clouds model the scene by a high number of points where each point corresponds to exactly one measurement of the camera. Each point has a fixed position in 3D space based on its real-world position. If the point cloud was created properly, the points of the positions of the objects corresponds to the real-world without any distortions. While this enables easy measurements for planes, the point

cloud can not be as easily processed and filtered as a 2D image. Because it is possible to calculate one from another (see section 3.4.1) the extraction of only one of the two types of data was needed in our measurements. As mentioned before, our project used two different sensors, the KinectV2 and the Phab2Pro. Because both of them used depth sensing technology from different companies, there was no universal way to extract data from the devices. For this reason, we developed two different tools to save the data into a readable format.

During the first setup, 200 measurements were taken at each plane orientation. This led to a total amount of 35.200 measurements for each depth sensor (22 distances $\times$ 8 rotations $\times$ 200 measurements = 35.200 measurements). Each measurement was saved in one consecutively numbered file. To organize all of these files we used a rather easy but effective directory structure. Each experimental setup had a directory containing sub-directories for each distance that in turn had sub-directories for each rotation, containing the measurements.

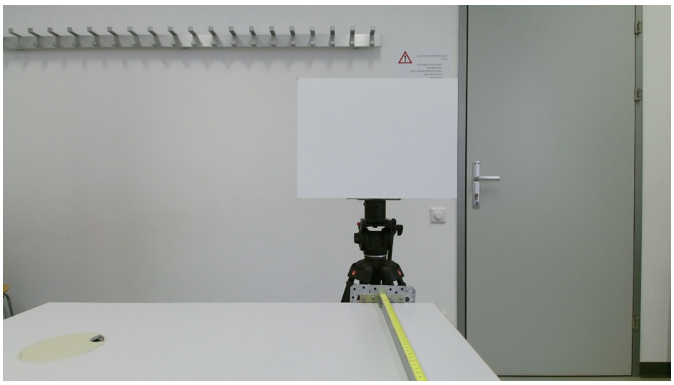
### 3.2.1 Kinect for Windows SDK

The KinectV2 sensor noise was extracted by using the Kinect for Windows SDK, provided by Microsoft. It contains not only drivers for both Kinect models but also an API to easily create programs using the sensor. Once it is installed and the KinectV2 is connected to a computer, it allows complete access to all functionalities and data provided by the sensor. This includes access to the raw data of the four different streams for color, depth, infrared and audio. The API offers an event based interface that notifies an application if a Kinect frame arrives. To allow this a function needs to be assigned to an event handler.

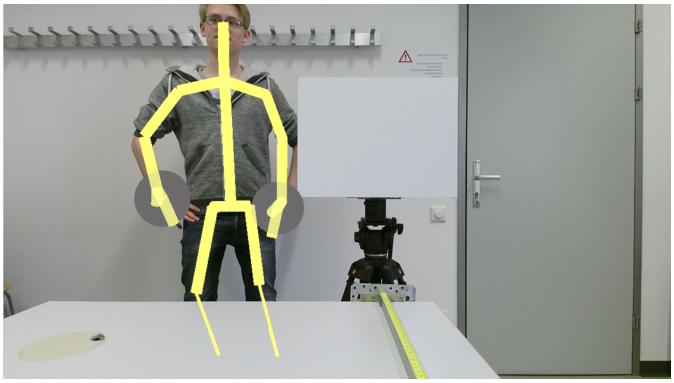
```
sensor = KinectSensor.GetDefault();

if (sensor != null)
{
        sensor.Open();
        reader = sensor.
                OpenMultiSourceFrameReader(
                        FrameSourceTypes.Depth);
        reader.MultiSourceFrameArrived += readFrame;
}
```

We used Visual Studio 2013 to develop a simple Visual C# WPF application that connected to the KinectV2 and saved the arriving frames of the depth stream on demand. As seen in Figure 3.3 the GUI consists of an image of the depth stream and a number of elements to specify the setup parameters. The depth stream was visualized by mapping it between black for close objects and white for distant objects. This live image was useful during the early stages of calibration and alignment, as it gives a good impression of the generated depth images. As described by the labels, the two text boxes define the current distance and rotation of the plane, which in turn specifies the target directories of the generated measurements. To keep the time between measurements as short as possible,

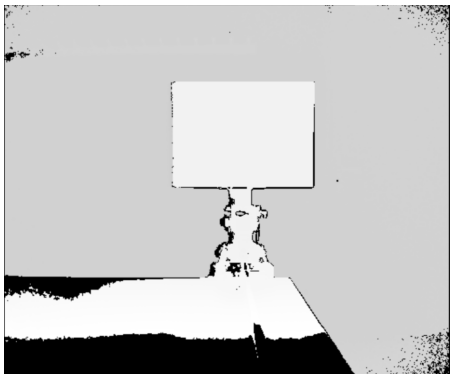


(a) Color stream



(b) Body data stream

(c) Infrared stream



(d) Depth stream

Figure 3.2: Images of the four visual streams provided by the Kinect (taken in Kinect Studio v2.0)


Figure 3.3: Screenshot of the C# WPF application used in our experiments

we used two tricks: The first one was using the plus and minus buttons beside the text boxes to increment or decrement the current alignment values by the usual step sizes (100 mm for the distance and 10° for the rotation), which allowed a quicker parameter change. The second and more important one was the asynchronous event handling for measurements. Because frames usually arrive faster than they can be written into `.csv` files, having only one thread handle one frame after another takes around 20 s. This process can be sped up by letting the event-handling thread start a separate thread to save the received frame. In doing so, the measurements were finished in around 4 s seconds. While the threads were still busy with saving the frames, we were able to rotate or move to the next alignment. To keep track of these processes, we implemented two signals an acoustic beeping sound to inform us that all necessary frames were recorded and a progress bar tracking the number of saved files.

### 3.2.2 Project Tango API

To extract depth information from the Phab2Pro we used the Project Tango API. The API uses (similar to the Kinect) an event-based system that notifies the app when a certain type of information is available. The depth data from the sensor is a good example of the abstraction done by the framework. While the Kinect SDK allowed direct access to the depth stream, Project Tango delivers an already preprocessed point cloud. The automatic preprocessing also transforms the point cloud to fit the motion of the sensor. While this allows to get correctly aligned point clouds when moving around in a room this transformation was unwanted for our evaluation as it may lead to incorrect measurements. To handle this problem we simply disabled the motion tracking feature in our app.

```
mTango.connectListener(framePairs, new OnTangoUpdateListener() {
            @Override
            public void onPoseAvailable(final TangoPoseData pose) {
                // ...
            }

            @Override
            public void onXyzIjAvailable(TangoXyzIjData xyzIj) {
                // Unsupported
            }

            @Override
            public void onPointCloudAvailable(final
                TangoPointCloudData pointCloudData) {
                // Save point cloud to file
            }

            @Override
            public void onTangoEvent(final TangoEvent event) {
                // ...
            }

            @Override
            public void onFrameAvailable(int cameraId) {
                // Only works for the two color cameras
            }
        });
```

We developed an app for the Phab2Pro, using Android Studio and the Java API for Project Tango (alternatively Unity, C and C++ are also supported). As seen in Figure 3.4, the layout of the app was similar to the WPF application for the Kinect. It has two text labels describing the current distance and rotation. Their values can be quickly changed between alignments, using the plus and minus buttons to increase or decrease the current value by the usual step sizes. A big button was used to start the measurements. To keep the measuring time as low as possible, the recording of a point cloud and its saving have been programmed to be asynchronous. The arriving point clouds were copied into a buffer and then handed to a separate thread to write them into a locally stored `.csv` file. Again, the reaching of the end of the point cloud recording was marked by an acoustic sound, while the number of saved files was tracked by a progress bar. Generally, the layout was designed in such a way that our app was still easily usable, while being mounted horizontally or vertically on our apparatus.

## 3.3 Calibration

The depth sensing in both cameras uses the images of an infrared camera to calculate the depth of a point. Like with nearly all optical cameras, the resulting images are partially

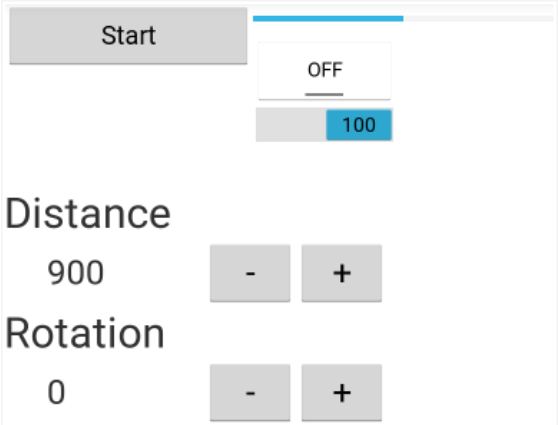



Figure 3.4: Screenshot of the app for the Phab2Pro used in our experiments

| | KinectV2 | Phab2Pro |
|---|---|---|
| Synchronous Time | 20 s | 80 s |
| Asynchronous Time - Measuring | 4 s | 25 s |
| Asynchronous Time - Saving | 16 s | 55 s |

Table 3.1: Comparison of synchronous and asynchronous measurement times for both programs

distorted because the lens bends the light that passes through before it is measured at the photo sensor. This leads to the unwanted effect that straight lines in the real world get bent in the image. [LMM+15] The two most typical distortion patterns are the barrel distortion – where the image appears to bend outward like a round barrel – and the pincushion distortion – where the lines bend toward the center like a pincushion.

This effect would have a negative impact on our noise calculations because, for example, the lateral noise was defined as the deviations along a straight edge. If an edge would be bent through lens distortions, the measured noise would be much higher than it actually was. The problem can be solved by transforming the images based on the parameters of the used lenses. The taken images can be simply bent into the opposing direction of the distortion until lines that should be straight are straight again. [FBR+15]

As previous works showed, a standard calibration procedure, which is used for most cameras, suffices for depth sensors using infrared cameras. Removing the distortion requires exact knowledge of the camera intrinsics and extrinsics like focal length, optical center, skew coefficients and real-world rotation and translation. Because most of these were not known, a procedure was needed to calculate them. [LMM+15] Most commonly, this is done by comparing known point coordinates with their corresponding image coordinates and estimating the intrinsics and extrinsics from the differences between them.

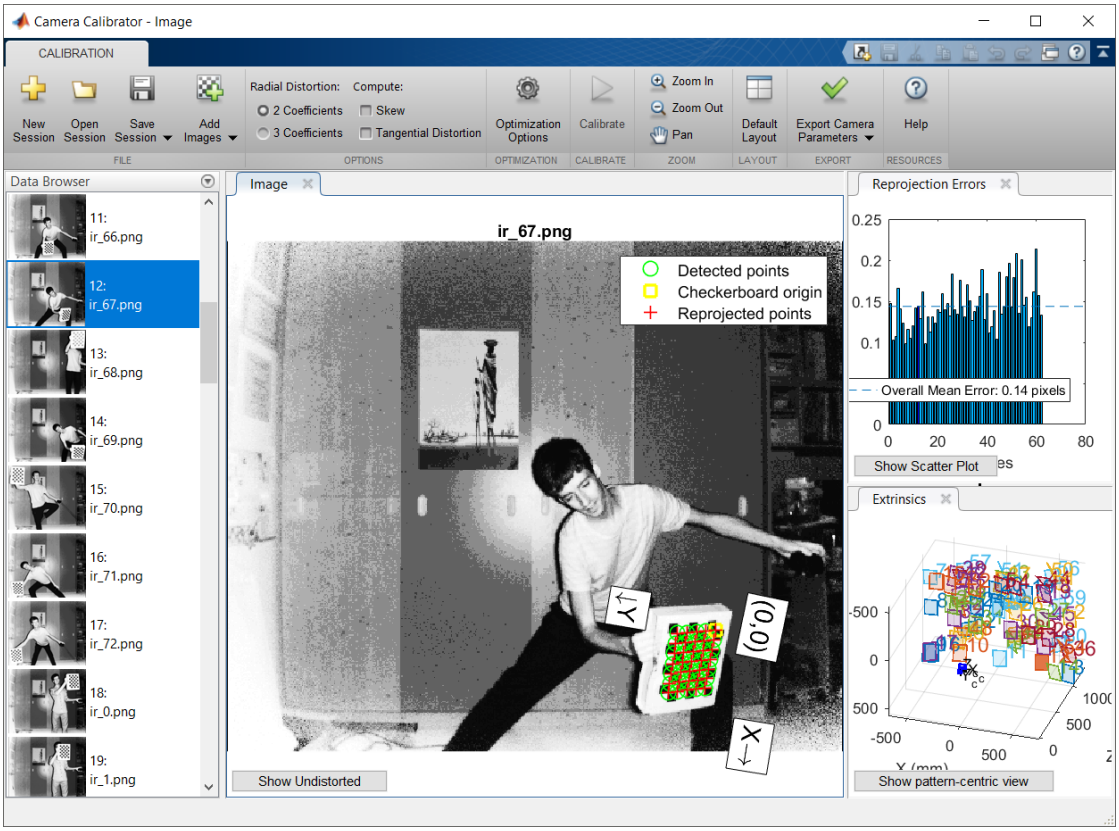
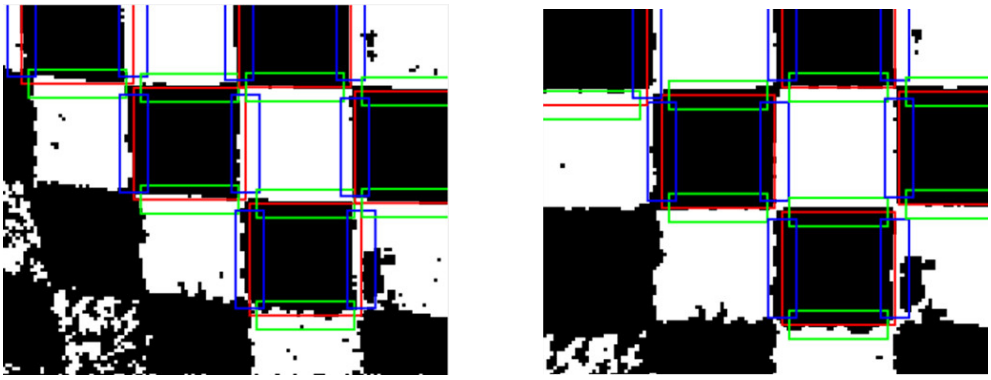



Figure 3.5: Screenshot of the MATLAB `cameraCalibration` tool. The left side shows the input images, the center contains the currently selected image with the detected pattern highlighted and the right side shows the fitting results (errors and pattern positions are displayed). [Kö17]

### 3.3.1 KinectV2 Calibration

For the KinectV2 we used a checkerboard pattern printed on an A4 sized paper as a reference [Wie17] and took several infrared images with the pattern at different positions and rotations, covering most of the image area. The resulting images were then used as an input for the MATLAB `cameraCalibrator` that detects the position of the pattern in the images. For each pattern its position and orientation were calculated based on the joints between the black and white fields. Using this information, the tool estimates the concrete camera parameters. [Zha00]

The resulting information was then used in the MATLAB function `undistortImage` that deformed the input depth images, according to the camera parameters. The improvements can be seen by looking at the outer regions of the depth maps 3.6 [Kö17]. [TM17b] [Bou04]



(a) Before Calibration: Distorted 3D checkerboard pattern

(b) After Calibration: Undistorted 3D checkerboard pattern

Figure 3.6: Comparison between two depth images of our second setup using a 3D checkerboard pattern. [Kö17]

### 3.3.2 Phab2Pro Calibration

The calibration procedure used for the KinectV2 cannot be a applied to the Phab2Pro because the Project Tango framework does not offer direct access to the images provided by the infrared sensor. Luckily, the Phab2Pro removes lens distortions by default [Goo] in their reconstructed point clouds, as seen in Figure 3.8. The calculated depth images bend inwards, indicating that the original images had a barrel distortion.

## 3.4 Preprocessing

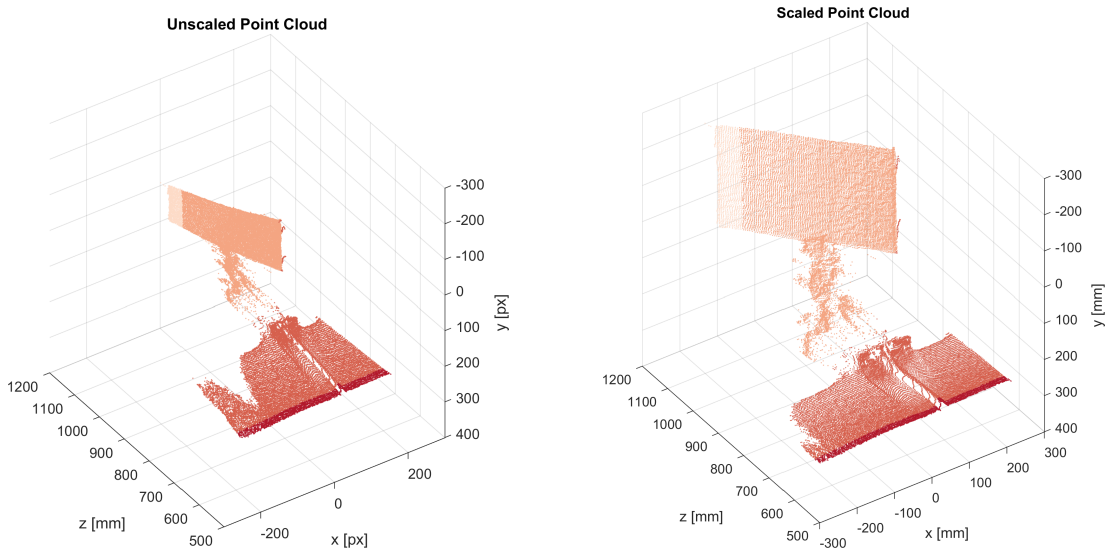
The gathered raw data cannot be directly used to extract the specific sensor noise because there were several problems that had to be accounted for. The fundamental problem was that the two different sensors deliver different types of data, the KinectV2 returns a depth image, while the Phab2Pro returns a point cloud. Furthermore, in both of these data sets the plane does not fill the whole sensor area, which means that the exact position of the plane was unknown and had to be found. This had do be done, while also ignoring wrong data from other objects than the plane.

### 3.4.1 Data Conversions

As mentioned before, both a depth image and a point cloud were needed in the calculation of the lateral and the axial noise. Because each of the sensors only returned one of them, a conversion from one type of data to the other one was needed.

#### Depth Image to Point Cloud Conversion

Starting with a depth image (like the one returned from the KinectV2), a simple conversion of a pixel position ($x$, $y$) and a depth value ($z$) to a point in space would lead to a




(a) Perspectively distorted point cloud generated by directly using the depth image coordinates as x- and y-positions

(b) Undistorted point cloud resulting of scaling the previous point cloud according to 3.1 and 3.2

Figure 3.7: Comparison of an unscaled and a scaled KinectV2 point cloud

| | KinectV2 | | Phab2Pro |
|---|---|---|---|
| | Manufacturer | Calibrated | |
| $width_{[px]}$ | 512 | 512 | 224 |
| $height[px]$ | 424 | 424 | 172 |
| $FoV_H$ | 70.6 | 73.78 | 65.45 |
| $FoV_V$ | 60.0 | 63.73 | 52.52 |

Table 3.2: Camera Intrinsics

rectangular cuboid view frustum with the objects inside it being skewed as seen in Figure 3.7. Objects closer to the camera appear bigger than they are and objects farther away (like the wall) are displayed smaller. To restore the original proportions, as well as convert the pixel distances into metric distances, the exact camera intrinsics needed to be known. Namely, the resolution of the depth image and the field of view were used in our calculation. We used our own specifications of these parameters, which we obtained through the calibration. (see Table 3.2) for the KinectV2. [NIL12]

$$x_{[mm]} = z * \frac{2\tan(\frac{FoV_H}{2})}{width_{[px]}} * (x_{[px]} - \frac{width_{[px]}}{2}) \tag{3.1}$$

$$y_{[mm]} = z * \frac{2\tan(\frac{FoV_V}{2})}{height_{[px]}} * (y_{[px]} - \frac{height_{[px]}}{2}) \tag{3.2}$$

Applying these equations to all points in the depth image allowed the calculation of their corresponding real-world $x_{[mm]}$ and $y_{[mm]}$ coordinates. To speed up the calculation process, the equation parts that are independent of the concrete points can be extracted and precalculated, as they are constant for all depth images. [NIL12]

$$\rho_x = \frac{2\tan(\frac{FoV_H}{2})}{width_{[px]}} \tag{3.3}$$

$$\rho_y = \frac{2\tan(\frac{FoV_V}{2})}{height_{[px]}} \tag{3.4}$$

$$x_{[mm]} = z * \rho_x * (x_{[px]} - \frac{width_{[px]}}{2}) \tag{3.5}$$

$$y_{[mm]} = z * \rho_y * (y_{[px]} - \frac{height_{[px]}}{2}) \tag{3.6}$$

The complete set of transformed points of one depth image is a correctly scaled point cloud, which was used to extract axial noise.

**Point Cloud to Depth Image Conversion**

The inverse of the previously mentioned method was used to transform a point cloud (like the one returned from the Phab2Pro) into a 2D depth image. To achieve this, the cone shaped view frustum needed to be rescaled to represent a cuboid, meaning that objects closer to the camera needed to be increased in size, while objects farther away had to be reduced in size. The final result is a depth image with its pixels containing the depth values similar to the sensor output of the KinectV2. To transform a point cloud, we needed to know the same camera intrinsics as in the depth image to point cloud transformation. For the Phab2Pro these were obtained through the API (see Table 3.2). The previously used equation can be inverted to represent the inverse transformation, mapping the real-world coordinates ($x_{[mm]}$ and $y_{[mm]}$) to image coordinates ($x_{[px]}$ and $y_{[px]}$).

$$x_{[px]} = \frac{x_{[mm]}}{z * \rho_x} + \frac{width_{[px]}}{2} \tag{3.7}$$

$$y_{[px]} = \frac{y_{[mm]}}{z * \rho_y} + \frac{height_{[px]}}{2} \tag{3.8}$$

The resulting reshaped point cloud was then saved into a 2D depth image. This was achieved by "painting" each transformed point's depth into the corresponding image position – similar to the painter's algorithm. Points sharing the same $x_{[px]}$ and $y_{[px]}$ coordinates were painted back to front. This means that the closest point covers the points behind it. Normally this case would not happen if the unprocessed point cloud

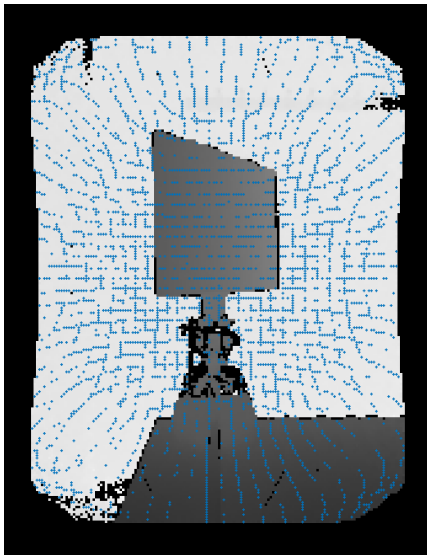


Figure 3.8: Overlapping points in the point cloud to depth image transformation because of preprocessing

of a time-of-flight or a structured-light camera was used. But because the point cloud returned by Project Tango was already preprocessed to account for effects like lens distortions, it resulted in around 10% of the points being occluded (as seen in Figure 3.8). To minimize the negative influence of this lossy transformation to our axial noise calculation, the generated depth image was only used to find the region of interest, while the original point cloud was used for the actual calculation. As for the lateral noise measured alongside the edges, the painter's algorithm prioritizes the closer points of the plane, which prohibits the generation of additional noise through the transformation.

### 3.4.2 Region of Interest Detection

As previously mentioned, the plane we used as a measurement target was neither filling the whole screen nor did it have the same size at different distances. Furthermore, the calculations to extract axial and lateral noise not only needed the concrete borders of the plane, but they were also very sensible to outside noise factors like foreign objects (wall, tripod, ...). By detecting the plane's boundaries as close as possible, we were able to effectively eliminate points not belonging to the plane.

#### Create Binary Image

As a basis for the further region of interest calculations, a binary image was used. The white pixels were used to indicate regions of interest, while black pixels were ignored. The initial binary image was created by applying a threshold to the depth image. The threshold consists of an upper and a lower boundary that were calculated based on the plane's width $w$, distance from the sensor $d$ and rotation $\theta$. The lower boundary was

based on the depth of the plane's side that got rotated away from the sensor while the upper boundary was based on the side of the plane closer to the sensor. Because the axial noise along the plane might have led to some points inside of it falling out of this range, we extended the threshold region by a small percentage $p$ to be more tolerant to outliers.

$$l = \sin\theta * \frac{w}{2} * p \tag{3.9}$$

$$b_{upper} = d + l \tag{3.10}$$

$$b_{lower} = d - l \tag{3.11}$$

Each depth pixel inside this range got classified as white, every pixel too far away or too close got classified as black.

$$c(z) = \begin{cases} 1 & \text{if } z \geq b_{lower} \text{ and } z \leq b_{upper} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

While this threshold calculation was able to correctly classify the plane, it also detected a lot of noise of objects being at the same distance as the plane like the tripod or parts of the table, the apparatus was standing on. (see Figure 3.10b)

#### Mask Out Unused Regions

To improve the detection rate, the amount of wrongly classified pixels not belonging to the plane needed to be reduced to a minimum. An easy way of getting rid of many outliers like the surrounding tables and chairs was masking out certain areas. To do this, the smallest region containing the complete plane for all different distances was found manually for each camera. All points outside of this region got classified as black because none of them could be part of the plane. (see Figure 3.10c and 3.10d)

#### Remove Noise

As previous works have shown, the sensor noise rises with growing distance and angle. This may lead to some points inside the plane exceeding the calculated limits and getting classified as black. But because of the next steps needed for the plane to be as continuously labeled as possible, noise inside the plane had to be removed. This was done by applying a hole filling algorithm that set black pixels completely surrounded by white ones to white. In doing so, we effectively reduced the noise inside the plane. To remove the noise alongside its borders, we used a square shaped structure element to first dilate and than erode the image. In this way, small pixel gaps alongside the borders were closed. (see Figure 3.10e)

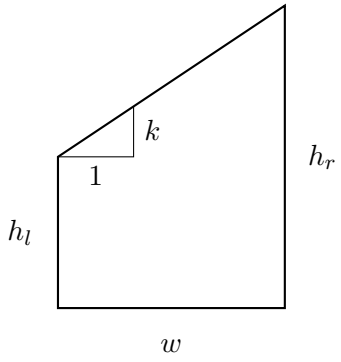



Figure 3.9: Scheme of the plane structure element

**Erode and Dilate Plane**

While the previous step smoothed the appearance of the plane, it also smoothed the appearance of the remaining noise components, some of which (like the tripod) may even have gotten more dominant. To finally remove all other objects except for the plane in our binary image, we used a specially calculated structure element in combination with erosion and dilation. In the previous step, we saw that by first dilating and then eroding we effectively removed black noise smaller than the structure element. In this step we wanted to achieve the opposite effect: removing white parts that stood out and which were smaller than our plane.

To achieve this, a special structure element was created that closely resembled the shape of the plane in the current distance and rotation. This was possible due to the positioning of the plane in the image. Its lower edge laid directly on the image's x-axis and the whole plane was directly centered around the image's y-axis. The effect of this positioning was that the shape could be calculated by knowing the camera intrinsics, the plane's size (width $w$ and height $h$), its distance and rotation.

$$h_l = \left\lfloor \frac{h}{b_{upper} * \rho_y} \right\rfloor \tag{3.13}$$

$$h_r = \left\lfloor \frac{h}{b_{lower} * \rho_y} \right\rfloor \tag{3.14}$$

$$w = \left\lfloor \frac{\cos(\theta) * w}{d * \rho_x} \right\rfloor \tag{3.15}$$

$$k = \frac{(h_r - h_l)}{w} \tag{3.16}$$

With this calculations, a binary structure element was created. The final result resembled the plane in the image, while being a little bit smaller to be more resistant to the

remaining noise. After that, the image was eroded with the structure element, removing objects not having the same size or form as the plane. This was followed by a dilation to restore the remaining elements to their correct size. Applying this procedure effectively removed every object in the scene that was not the plane itself in nearly all images. (see Figure 3.10f)

#### Calculate Bounding Box

The final step, after everything except the plane pixels was removed from the binary image, was to get the bounding box of the remaining white part. This was done by using the MATLAB function `regionprops` that measures different properties of the white parts in an image. The result contained the exact bounding box of the remaining white region, our plane. The region was then returned as the region of interest for further noise calculations. (see Figure 3.10g)

## 3.5 Noise Extraction

By combining the preprocessed point cloud and depth images with their corresponding regions of interest determining the plane's position, the axial and lateral noise were calculated. In general, this was done by taking the standard deviation of the difference between the data and the calculated ground truth. [NIL12] [FBR+15]

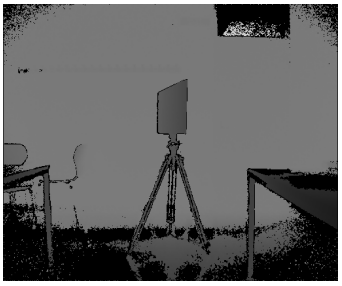
### 3.5.1 Lateral Noise Extraction

The lateral noise was calculated for both the x- and the y-direction. The noise was extracted based on the pixel positions alongside the plane's edges in the depth image. The idea was that the plane was normal to the image's x-axis, which should result in a perfect line in the resulting depth images. The so called wiggling error of time-of-flight cameras tampers with our results, leading to misplaced pixels along the edges. By measuring the horizontal pixel positions this error could be estimated.

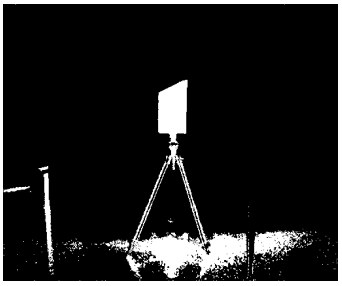
Because this calculation required depth images, the Phab2Pro point cloud data needed to be projected onto a depth image, as mentioned during the preprocessing. As for the direction of the error, the extraction of noise with the sensor in its horizontal position led to the lateral noise in x-direction and the extraction of noise with the 90° rotated sensor to the lateral noise in y-direction.

#### Border Region Calculation

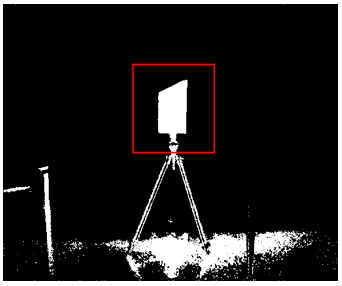
Based on the previously calculated regions of interest, the regions containing each of the vertical plane borders could be estimated. At first, the border region's width was manually defined. The region of interest's x-coordinates were moved away from the plane by half the border width. The height of the region was taken from the detected region of interest. To only measure the horizontal lateral noise component, the top and bottom of the border region were trimmed by a scale factor.



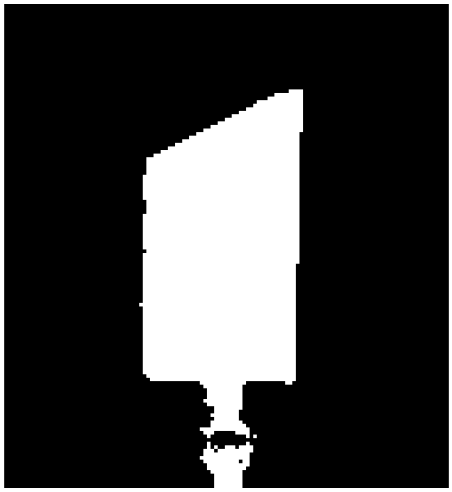
(a) Depth map for $z = 2100\ \text{mm}, 60°$



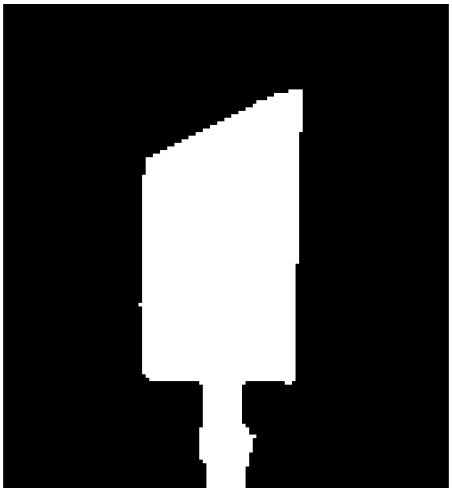
(b) Binary image from depth threshold



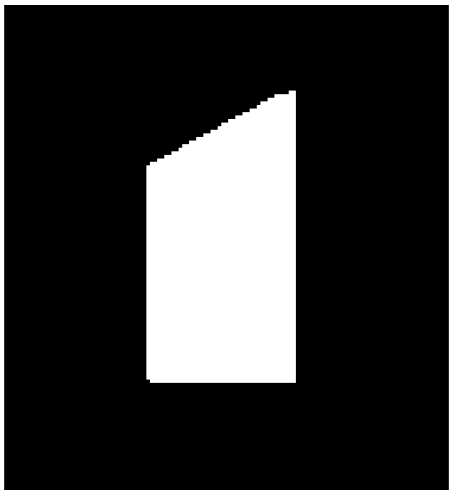
(c) Manually selected mask region



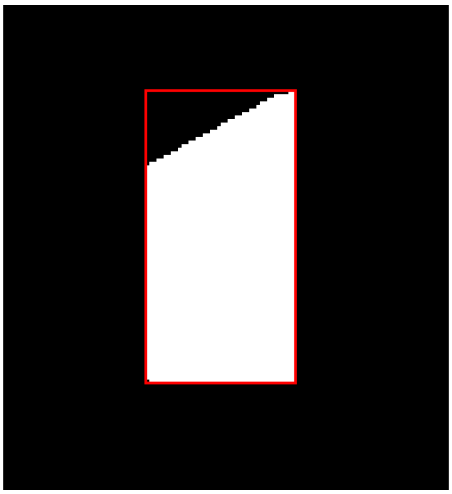
(d) Remaining binary image with unused regions removed



(e) Noise reduced image after filling holes and smoothing edges



(f) Binary image after the application of the plane shaped structure element



(g) Bounding box of the remaining white area

Figure 3.10: Processing steps of the plane recognition

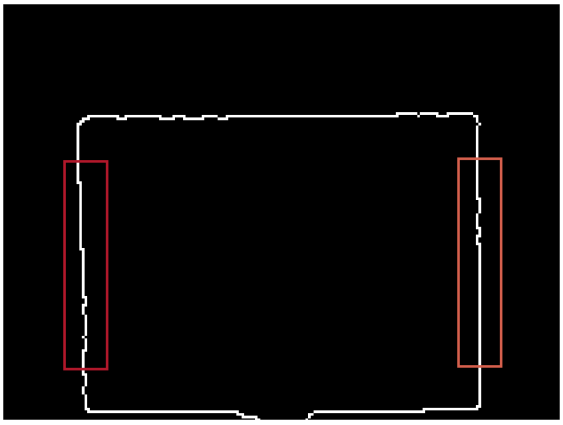


Figure 3.11: KinectV2 edge image for the lateral noise extraction with highlighted border regions. ($z = 1000$ mm, $\theta = 0°$)

#### Edge Image Creation

Because the lateral noise components were measured around the edges of the plane, they had to be made visible. The first step was to transform the depth image into a binary image by applying a threshold that marks objects in the depth range of the plane as white and everything else black. After that, a Canny edge detector was applied to the image. The special property of this edge detector is that it not only detects edges (changes from black to white) but also that these edges are only one pixel in their width. The filtered image was scanned for horizontal lines (rows with more than one white pixel). From this lines only the outermost pixels were kept white while the others were set to black. This was used to count the bumpy parts of the edge only once.

#### Lateral Noise Quantification

After this preprocessing step, each line contained at most one white pixel. The column indices of these pixels were determined by applying the MATLAB function `find`. The lateral noise was the standard deviation of these values. The results were two $\sigma_L[px]$ describing the horizontal lateral noise for each side of the plane. Both the left and right side noise were handled and saved separately because as Fankhauser et al. [FBR+15] showed, their values differ greatly. The reason for this is that sensors like the KinectV2 have an asymmetric setup with the infrared sensor and emitter being slightly apart, leading to one side of the plane being better illuminated and measured than the other.

### 3.5.2 Axial Noise Extraction

The axial noise was calculated by comparing the measured plane points to the estimated plane. The difference between the plane and the measurements was the noise, which was a direct result of the unsteady depth measurements. By looking at the point cloud data,

| Equation | Independent Var. | Coefficients | Problem Par. |
|---|---|---|---|
| (3.17) | $x$ | | $\theta$, $d$ |
| (3.18) | $x$ | $a$ | $\theta$ |
| (3.19) | $x$ | $a$, $b$ | |
| (3.20) | $x$, $y$ | $a$, $b$, $c$ | |

Table 3.3: Plane Equation Parameters

the plane's surface seemed to have lots of small bumps and dells, which were a result of those measurement errors.

### Central Region Calculation

The region of interest determined by the preprocessing steps contained the whole plane up to its outer borders. The regions alongside the borders were prone to not only suffer from axial noise but also from lateral noise, as described in 3.5.1. To prevent a falsification of the axial noise, a border margin had to be added to the region of interest. This was done by decreasing the region's sides by 20%, the factor was manually chosen to eliminate any lateral noise interference, while still remaining an as large as possible region to calculate the axial noise.

### Plane Fitting

The points lying inside the resulting region were then selected, discarding the other points of the point cloud. For these plane points a plane with minimal distance to all points was calculated. This was done by using the MATLAB function `fit` that calculates the best coefficients for an equation with given points. The equation calculates $z$ based on the values of $x$ and $y$. We experimented with three different plane equations with different grades of restrictiveness in regard to the physical ground truth.

$$z = \frac{\sin(2\pi - \theta) * x + d}{\cos(2\pi - \theta)} \tag{3.17}$$

$$z = \frac{\sin(2\pi - \theta) * x + a}{\cos(2\pi - \theta)} \tag{3.18}$$

$$z = \frac{\sin(b) * x + a}{\cos(b)} \tag{3.19}$$

$$z = a * x + b * y + c \tag{3.20}$$

The equations are ordered from top to bottom by their flexibility to adapt to a certain set of points. This can be seen at the growing number of fitted coefficients in Table 3.3. Before each of the equations was examined, it had to be remarked that the first three equations ((3.17), (3.18), (3.19)) only need the $x$ variable to calculate $z$ because

these models assume that the plane was perfectly normal to the x-axis without a tilt. This means that every point with the same y-coordinate has the same depth $z$. The first equation (3.17) was entirely based on the known ground truth by using the known distance $d$ and rotation angle $\theta$ as fixed parameters. The second equation (3.18) neglects the distance constraint. The third equation (3.19) neglects the distance and rotation constraint of the plane but still assumes that it was not tilted. The ideal case would be using one of these three equations to get precise noise information like the standard deviation or the depth over- or underestimation. But the problem with these equations was that even with a lot of preparation, calibration and adjustment, we were not able to achieve the exact positions with no error at all in regard to distance, rotation or adjustment. By estimating the error with one of these strict equations, the noise of the sensor would be overshadowed by positioning errors. To prevent this we decided to use the standard and most flexible plane equation (3.20). Without any ground truth constraints the fitted plane allowed an estimation of the noise without being susceptible to positioning errors.

#### Axial Error Quantification

Given the fitted plane equation from the previous step, the ground truth depths could be estimated by using the measured point's x- and y-coordinates as input variables for the equation. This was done by the MATLAB function `feval` that uses a fitted model and various input parameters to calculate the resulting values. Each point's calculated ground truth depth was subtracted from its actually measured depth. This difference is the measurement error. By calculating the standard deviation of these errors, we got the axial noise $\sigma_A[mm]$ for a certain position. The axial noise was unlike the lateral noise already calculated in $mm$ and therefore did not need to be additionally transformed.

### 3.5.3 Noise Saving

Because for every combination of position, rotation and depth sensor, 200 images were made, the axial and lateral noise calculation for each data set can take multiple hours. For this reason, the results of each measurement were saved as a line in a `.csv`-file consisting out of the experiment's parameters (distance, rotation and measurement number) and the calculation results for the axial and lateral noise. For each distance a separate file named after the distance was created in a subdirectory of the dataset called `Results`. This allowed us to re-evaluate the results of all 200 images without having to re-calculate everything.

## 3.6 Model Generation

The empirically derived noise data was used as a basis for three mathematical models describing $\sigma_z$, $\sigma_x$ and $\sigma_y$ for a certain distance $d$ and surface rotation $\theta$. The goal was to find a model that characterizes our measured data as close as possible and allows a valid prediction of the positions lying in-between our measurements. The choices to make for

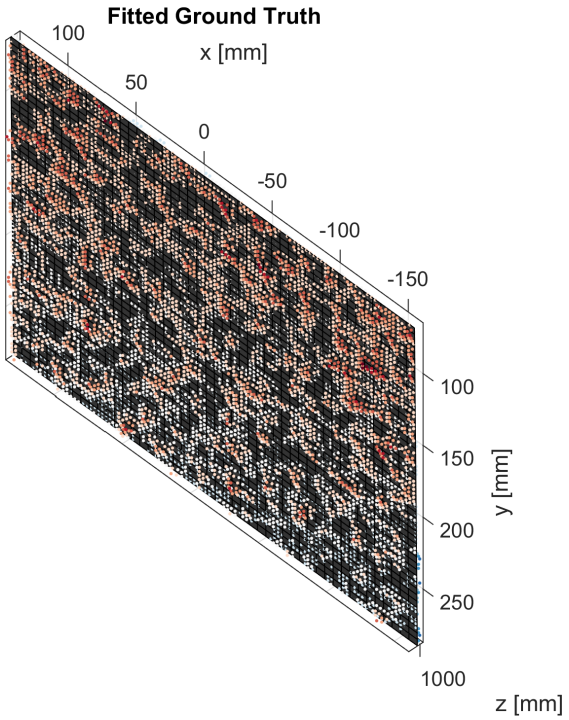



Figure 3.12: Fitted ground truth plane for KinectV2 point cloud ($z = 1000$ mm, $\theta = 0°$)

this were at first finding a mathematical formula that has the same shape as the noise and then fitting its coefficients to be as close as possible to the noise data points.

### 3.6.1 Data Preparation

The raw noise data, calculated in the previous steps, had to be transformed before it was usable for the generation of an appropriate model. The axial noise data has two measurements for each position: one from the horizontal and one from the vertical camera setup. Instead of using only one or another for our axial noise model, we decided do combine the gathered information to have an even better noise estimation. To do this, we calculated the average of the two standard deviations measured at each position and used this as our axial noise. The lateral noise was handled in a similar fashion because each position also had two measurement values: one of the left and of the right plane side. As mentioned before we kept both values apart to analyze the effects of different sensor illumination for the left and the right side. But because our model cannot differentiate between the both sides, we decided to combine both lateral noise values to one. This was again done by taking their average. The last step was the transformation of the pixel based lateral noise to a metric one $\sigma_L[mm]$, using the formula described in 3.4.1 and the known distance of the plane.

### 3.6.2 Model Fitting

After the noise data was prepared, an appropriate model representing the measurements had to be found. Such a model should fulfill two requirements:

1. Model the measured data distribution as close as possible
2. Remain general enough to be used to plausibly predict unmeasured values

Both of these requirements needed to be verified in a way. The first was checked by calculating the root-mean-square error (RMSE):

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{t=1}^{n} (\hat{z}_t - z_t)^2}{n}} \tag{3.21}$$

Additionally, we tried to maximize the value of $R^2$ that describes the proportion of data variance described through the chosen model.

The second constraint could not be calculated because it was hard to verify mathematically and was therefore visually determined by plotting the function and analyzing its behavior with different input values. A good model should have an as low as possible RMSE value and its function should have an as low as possible degree and avoid too high jumps between measured values. The last two constraints could be verified by plotting the surface of the calculated model.

To find the best fitting model for each axial and lateral data set, we split the fitting process into two parts: a manual model selection and an automatic fitting of the model coefficients. To dynamically propose a model and check how well it suits the gathered data, we used the MATLAB curve fitting toolbox, which allows a quick and interactive way to change the data, the underlying model and the fitting constraints. The output of the toolbox is both a mathematically calculated goodness consisting of many different key figures (RMSE, $R^2$, ...) and a visual plot of model and data points. 3.13

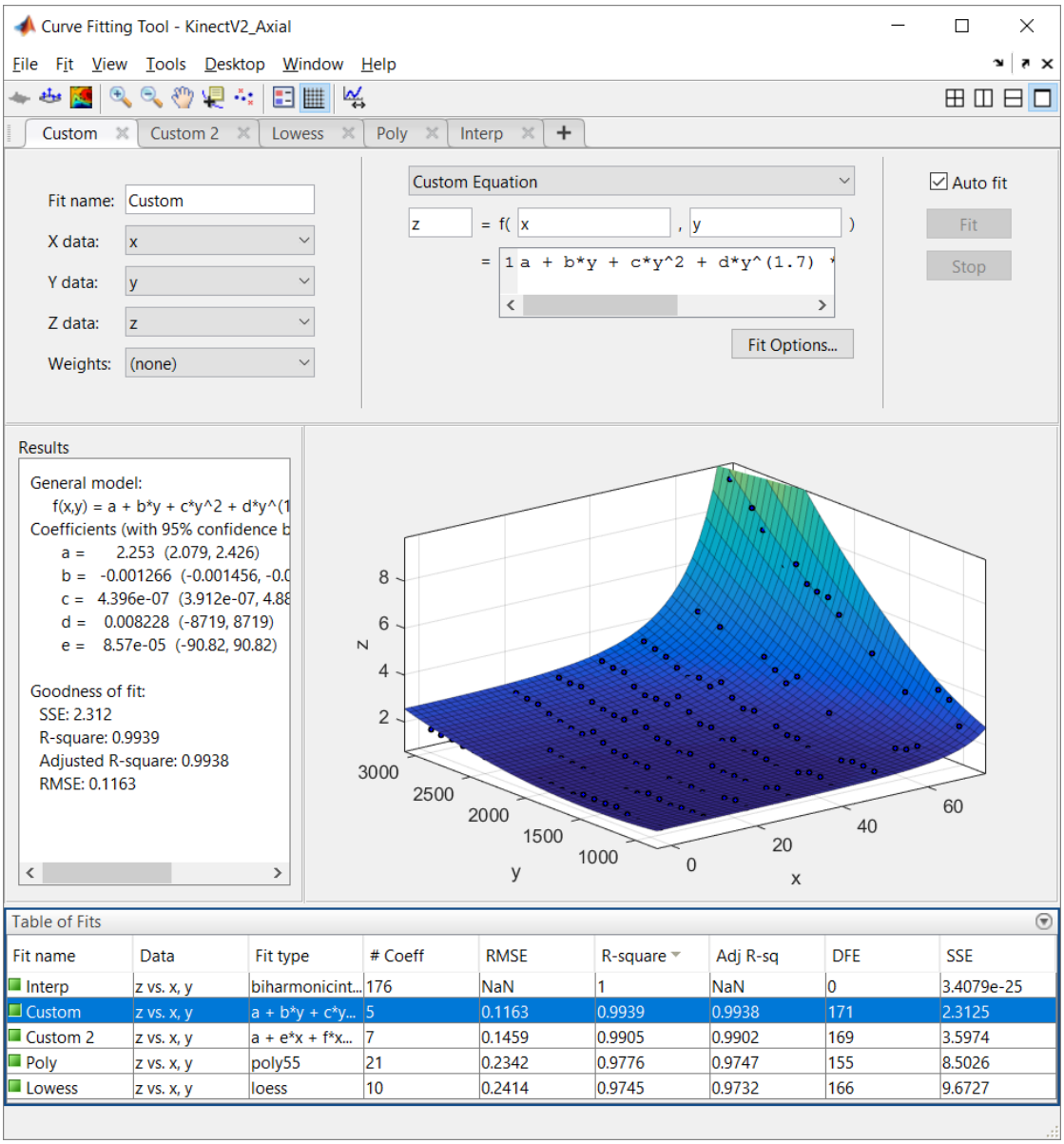



Figure 3.13: Screenshot of the MATLAB curve fitting toolbox

CHAPTER 4

# Results

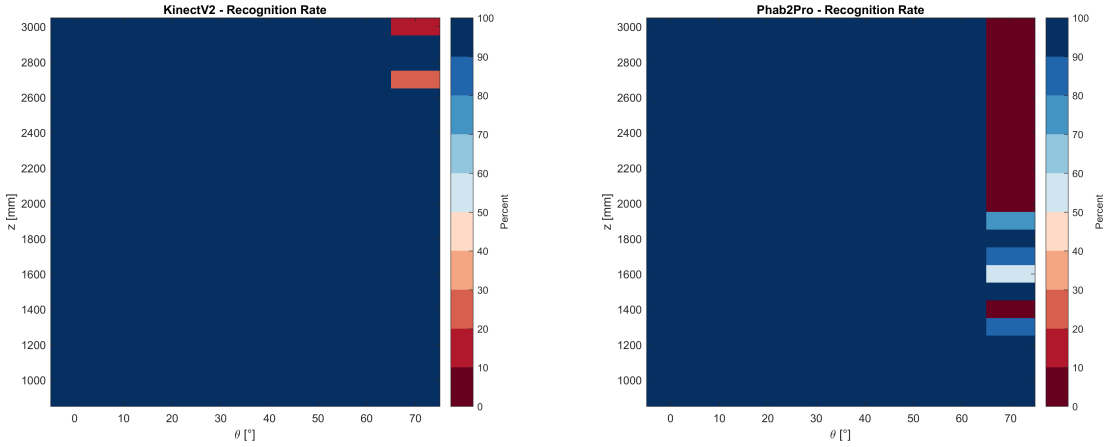
## 4.1 Recognition

As seen in Figure 4.1, the recognition rate for both cameras is 100% for angles between 0° and 60°. For 70°, the KinectV2 recognition has some small errors in larger distances. To compensate for this errors, we detected as many regions for the same distance and rotation as possible and calculated their average. The resulting region could then be used as input for the noise calculation, so that even images without a directly detected region of interest can be used. The Phab2Pro images for 70° are completely unrecognizable after a distance of 2 m, meaning that not even an average region can be used for the noise estimation, as not even a single region was found in the 100 images.

Looking at the raw data, this is not the fault of our algorithm but more a result of missing points in the point cloud due to too high noise. Both cameras deal differently with these high noise regions, while the Phab2Pro simply discards points in this regions, leaving them empty, the KinectV2 sets the distance of these points to 0 m. The plane in the resulting measurements (as seen in Figure 4.2) could not be detected using our morphological approach. The low detection rate did not tamper with the later calculations, as the noise for this angles could not be reliably calculated by plane fitting. The noise for this unmeasurable regions was assumed to be approaching infinity.

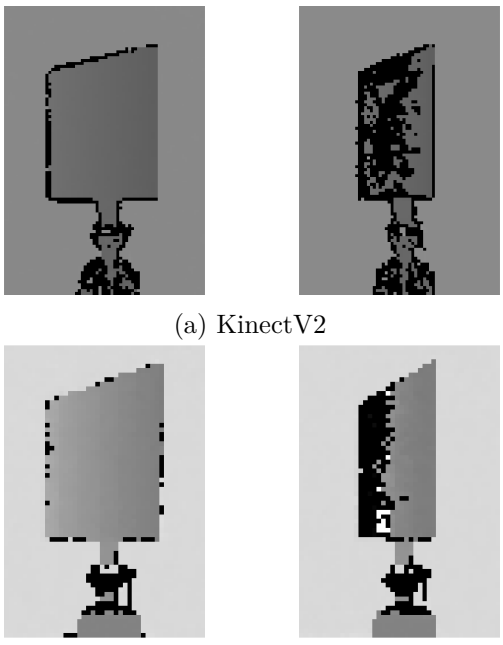
## 4.2 Noise Distribution

Because all following measures used in this work, to describe the sensor noise like the standard deviation and the noise models calculated from them, are based on the assumption that the systematic error is normally distributed, one of our next steps was to validate this claim. This was done by both looking at the histograms of the calculated axial and lateral noise and calculating the number of measurements, whose noise distributions correspond to a normal distribution based on the Kolmogorow-Smirnow-Test [TM17a].




(a) KinectV2 (b) Phab2Pro

Figure 4.1: Comparison of the recognition rate of our algorithm for the KinectV2 and Phab2Pro on the complete dataset



(a) KinectV2

(b) Phab2Pro

Figure 4.2: High noise in the depth images of both sensors for high surface angles (60° vs. 70°)

| | 0° - 70° | 0° - 60° |
|---|---|---|
| KinectV2 | 78.01 | 88.38 |
| KinectV2 rotated | 83.52 | 86.10 |
| Phab2Pro | 82.05 | 92.99 |
| Phab2Pro rotated | 84.15 | 95.45 |

Table 4.1: Percent of all measured axial noise distributions that are describable through a normal distribution (based on the results of the KS-Test)

### 4.2.1 Axial Noise Distribution

The plots of the axial noise for the KinectV2 and the Phab2Pro indicate a normal distribution (as seen in Figure 4.3) that seems to have a growing standard deviation the farther away the measured plane is and the more rotated it is. Both of these observations were already made for the KinectV2, but are now also shown for the Phab2Pro.

The results of the Kolmogorow-Smirnow-Test [TM17a] in Table 4.1 show that around 80% of the data is describable through a normal distribution at a significance level of 5%. The highest portion of measurements whose noise distribution is counted as not normally distributed are the ones taken at high angles like 70°. If those are taken out of the calculation, the test results go up to around 90%.

### 4.2.2 Lateral Noise Distribution

The lateral noise distributions for both the x- and the y-direction do not show clear signs of a normal distribution in their fitted histogram plots (as seen in Figure 4.4 and 4.5). Furthermore, the noise does not seem to grow clearly with distance or angle.

This observation is validated by the Kolmogorow-Smirnow-Test [TM17a] that clearly states that no lateral noise measurement is describable by a normal distribution at a significance level of 5%. This indicates that either the measurement method based on pixels or the statistical model are probably not an appropriate measure for the lateral noise. In abundance of a better way to analyze the lateral noise components, we still used the standard deviation as a rough estimation for the following calculations.

## 4.3 Noise Model

For each position the noise is calculated based on all measurements. These noise values describe the fluctuation for a measured depth point on a surface. To analyze how the intensity of noise changes with the two independent variables $z$ and $\theta$, a simple line plot was used (as seen in Figure 4.6). Based on this data, the models were calculated. The results can be seen in Figure 4.7.

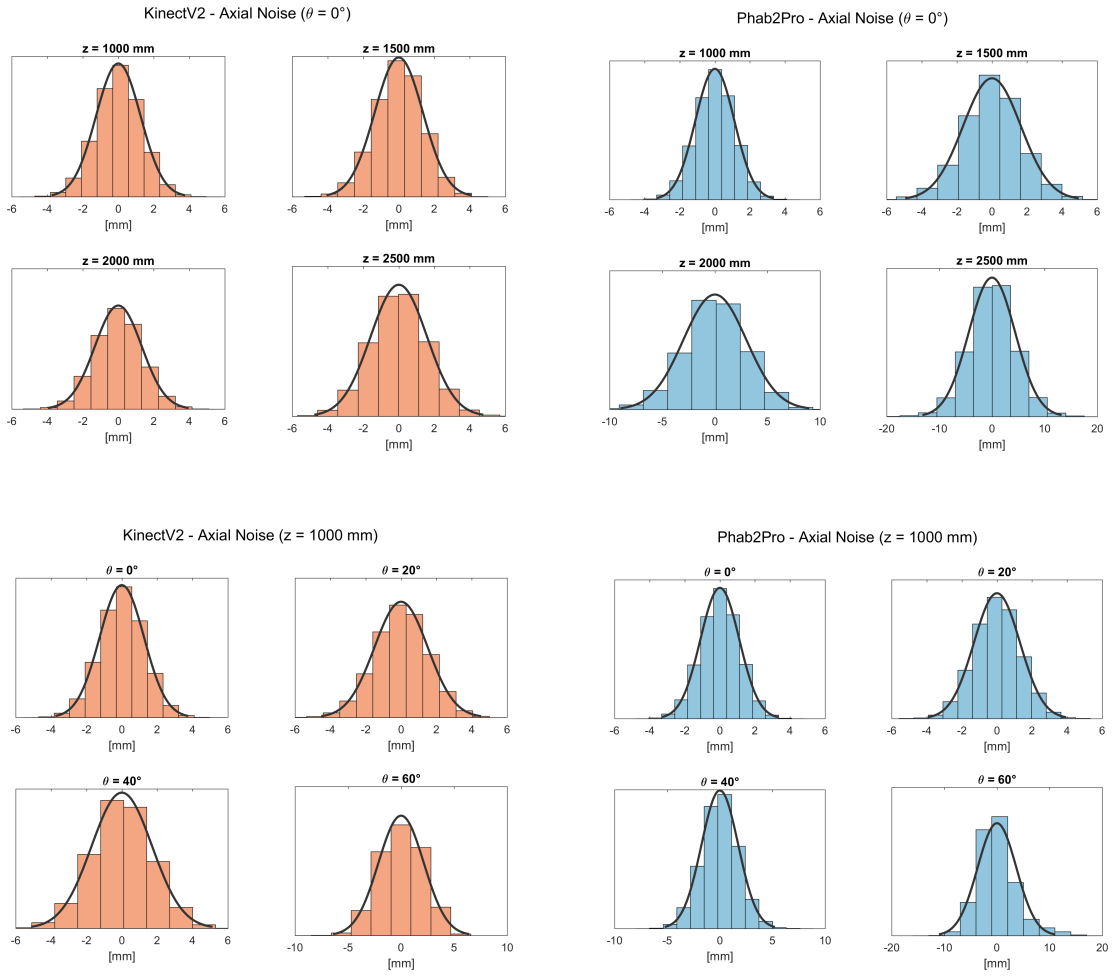



Figure 4.3: Comparison of the axial noise distributions for both sensors with different distances and rotations

### 4.3.1 Axial Noise Model

The plots of the axial noise indicate a low linear growth between 0° and 40° for both sensors. After that, the noise seems to grow rapidly with higher degrees approaching infinity. This behavior was already shown for the KinectV2 by Fankhauser et al. [FBR+15], whose model was adapted for our measurements and additionally applied to the Phab2Pro. Their function is a combination of a quadratic part for the distance $z$ and a hyperbolic part for the rotation $\theta$. While the most coefficients were fitted, as previously explained, the values for $e$ were found manually.

$$\sigma_{z_1} = a + b * z + c * z^2 + d * z^e * \frac{\theta^2}{(\frac{\pi}{2} - \theta)^2} \tag{4.1}$$

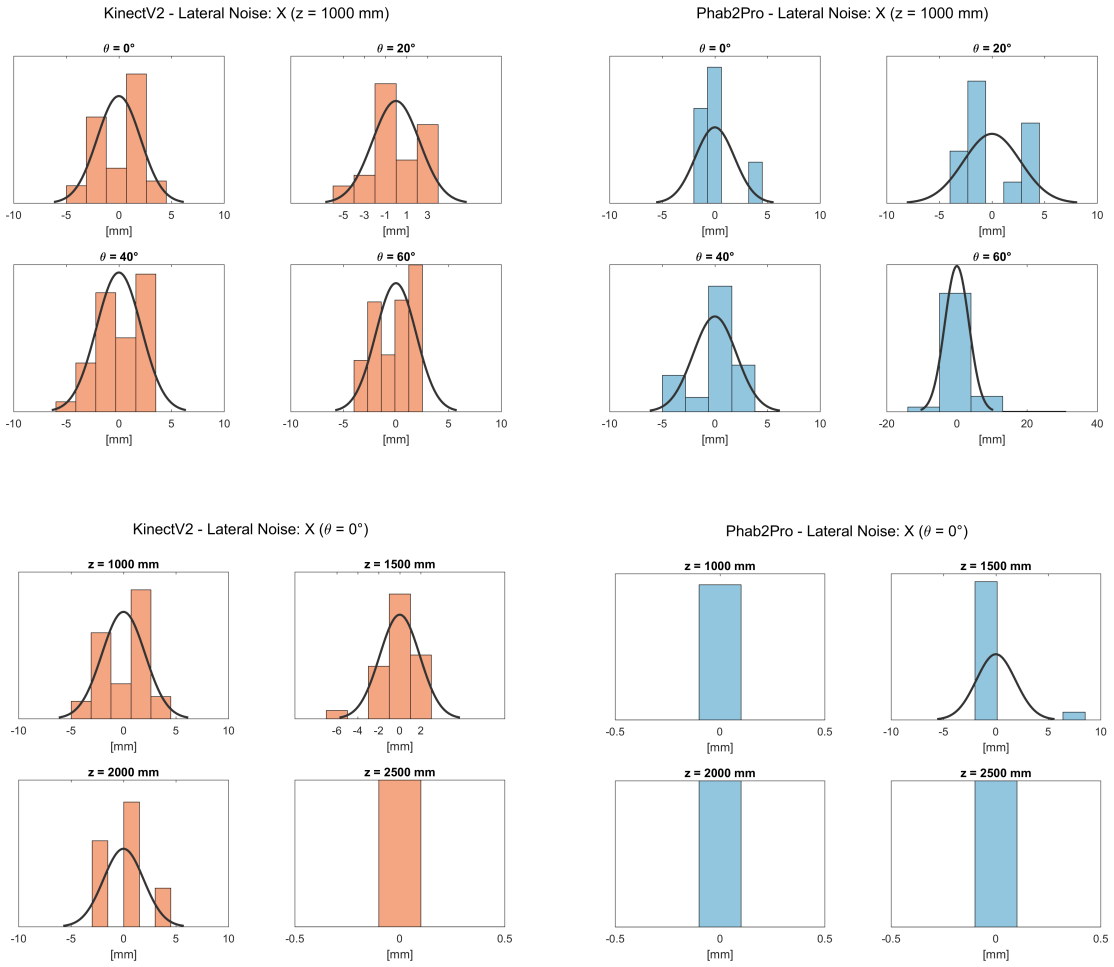



Figure 4.4: Comparison of the lateral noise distributions in x-direction for both sensors with different distances and rotations. [Kö17]

| | KinectV2 | Phab2Pro |
|---|---|---|
| $a$ | 2.094 | 0.3019 |
| $b$ | $-1.099 * 10^{-3}$ | $5.712 * 10^{-4}$ |
| $c$ | $4.048 * 10^{-7}$ | $6.183 * 10^{-7}$ |
| $d$ | $6.846 * 10^{-7}$ | $2.386 * 10^{-5}$ |
| $e$ | 1.7 | 1.47 |

Table 4.2: Axial Model Coefficients (Equation 4.1)

### 4.3.2 Lateral Noise Model

Because of the seemingly random nature of the lateral noise in our measurements, a mathematical model could not be fitted to the data. The reason for this could be

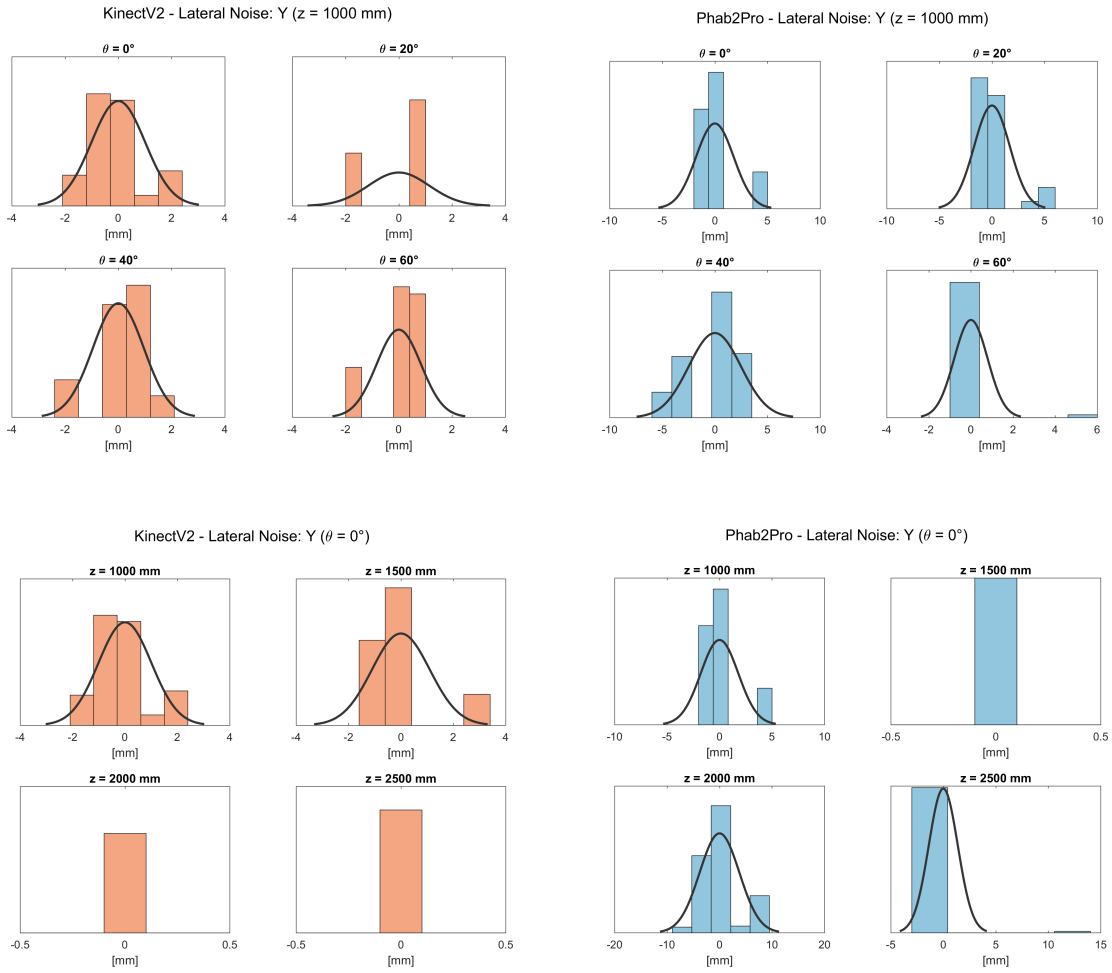



Figure 4.5: Comparison of the lateral noise distributions in y-direction for both sensors with different distances and rotations. [Kö17]

that either our noise extraction method was not sufficient enough to detect the noise appropriately, or that the noise simply does not follow a statistically significant pattern described by the standard deviation.

To circumvent this problem, we decided to calculate the 90-percentile to be used as a plausible upper boundary for the lateral noise in both directions. This value is constant for all distances and rotation angles, as no trend for any variable is visible in the data. It is also notable that the noise in y-direction is lower for both sensors.

### 4.3.3 Combined Model

As the previously mentioned model only used distance and rotation to estimate the noise, we decided to add additional parameters in order to enhance the quality of the model.

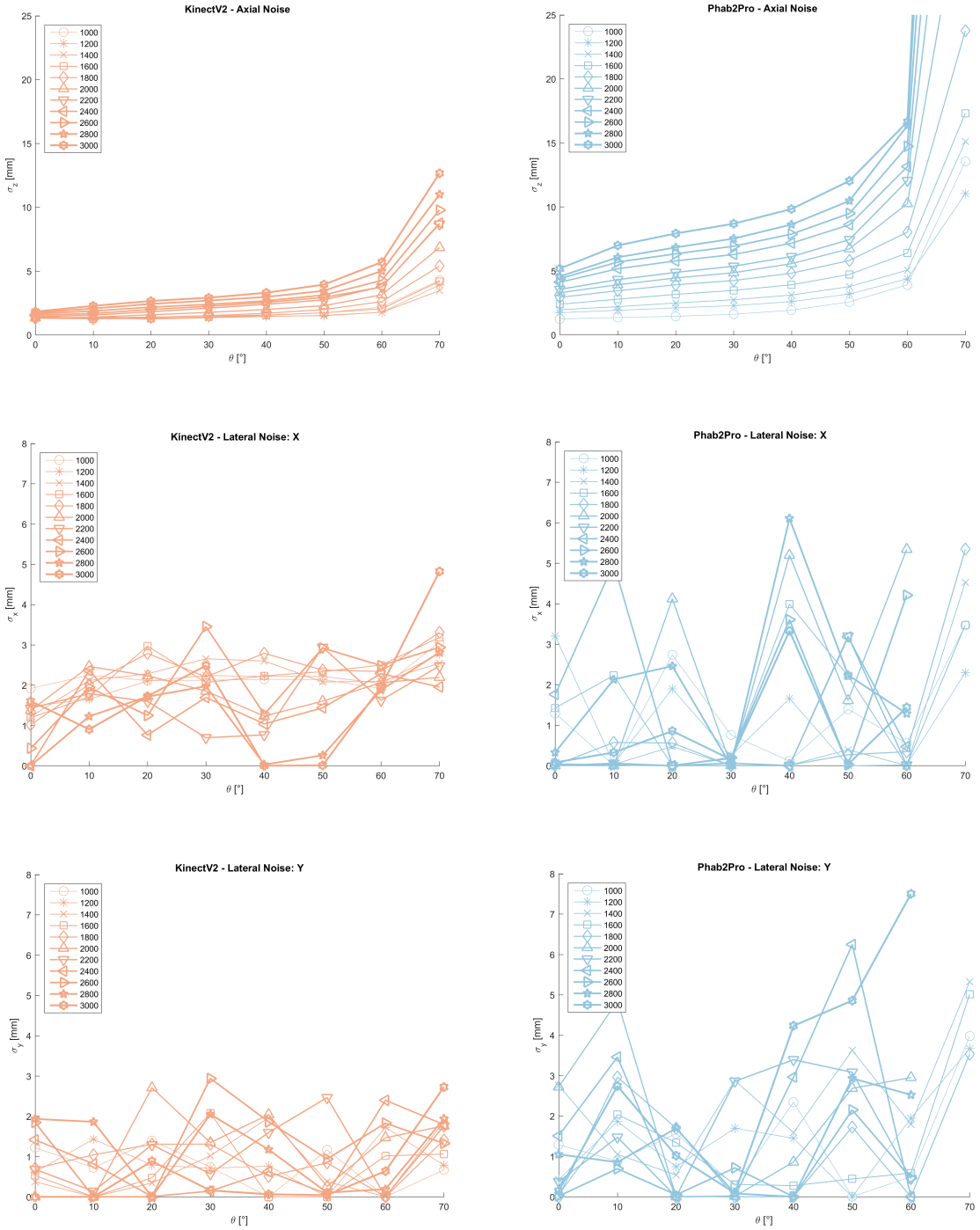



Figure 4.6: Calculated standard deviations of the fitted normal distributions for the sensor noise

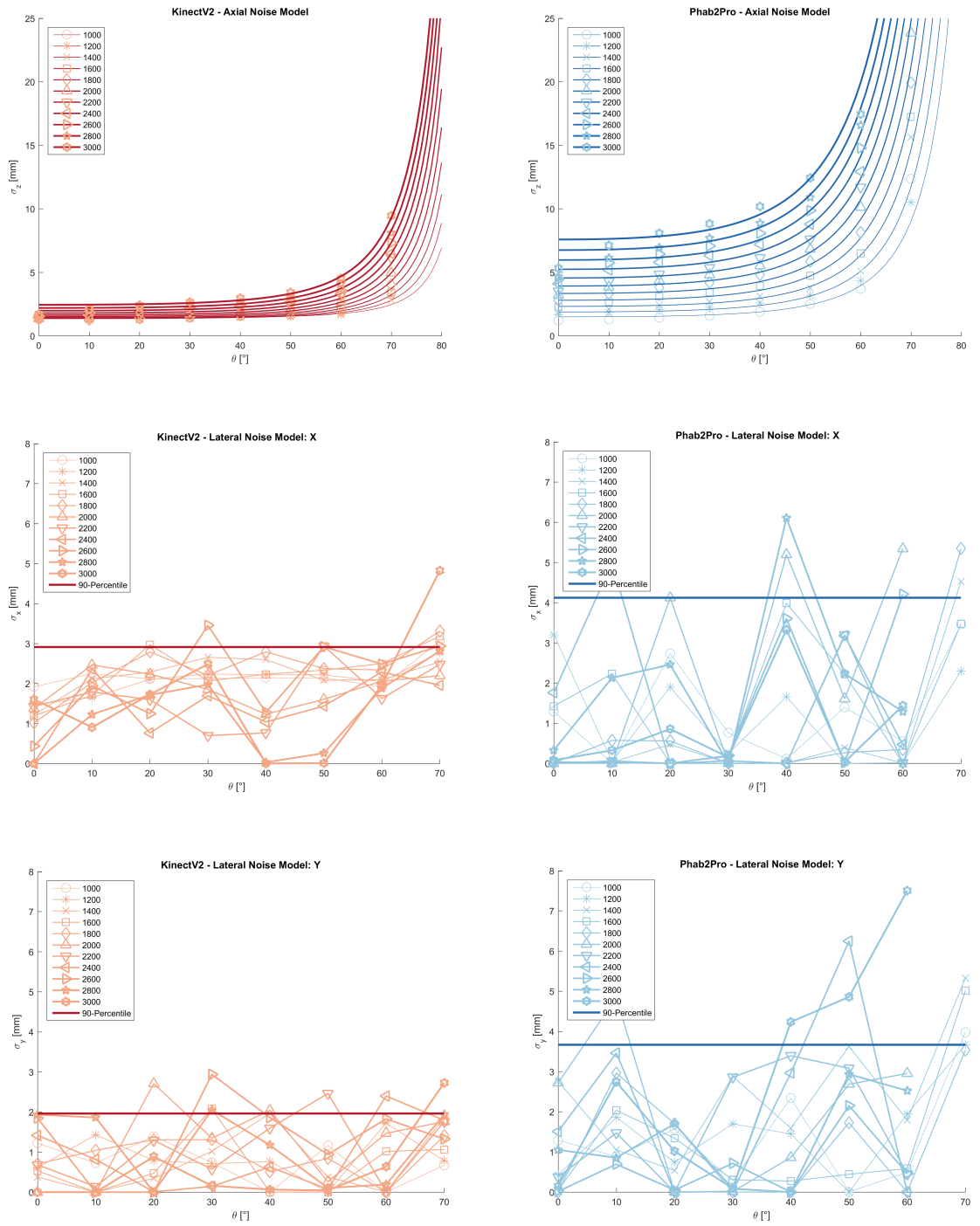



Figure 4.7: Results of the noise model estimations for the standard deviations of the sensor noise

| | KinectV2 | Phab2Pro |
|---|---|---|
| $\sigma_{x_1}$ | 2.9110 | 4.1207 |
| $\sigma_{y_1}$ | 1.9617 | 3.6665 |

Table 4.3: Lateral Models (90-percentile)

| | KinectV2 | Phab2Pro |
|---|---|---|
| $w_a$ | 29 | 8 |
| $w_b$ | 0.3 | 0.2 |
| $w_c$ | 0 | 0.3 |

Table 4.4: Weight Function - Coefficients

This was done by taking another axial and lateral model from the thesis of my colleague Thomas Köppel [Kö17] and joining it with the model presented in this thesis. The added model uses the distance and pixel coordinates of a point to determine the noise. This is based on the idea that the noise in a depth image is not uniform across the whole image. Each model represents a different component of the axial and lateral noise, the first estimates it based on the object's surface distance and rotation, while the second is based on the object's relative image position.

The model of this work (which will be called Model 1 - $\sigma_{x_1}$ $\sigma_{y_1}$ $\sigma_{z_1}$) is combined with the model from my colleague Thomas Köppel [Kö17] (which will be called Model 2 - $\sigma_{x_2}$ $\sigma_{y_2}$ $\sigma_{z_2}$) by multiplying them with a simple weight function $w$ and adding them up. Although we planed to only average between both models, our evaluation 4.4 showed that such a model would lead to worse results than Model 1. The reason for this seems to be that the additional noise from a rotated surface outweighs the image position dependent noise component. Because of this, our manually determined weight functions assign a higher weight to Model 1 at higher angles.

$$w = \max\left(\frac{w_a - \theta}{w_a} * w_b + w_c, w_c\right) \tag{4.2}$$

$$\sigma_x = (1 - w) * \sigma_{x_1} + w * \sigma_{x_2} \tag{4.3}$$
$$\sigma_y = (1 - w) * \sigma_{y_1} + w * \sigma_{y_2} \tag{4.4}$$
$$\sigma_z = (1 - w) * \sigma_{z_1} + w * \sigma_{z_2} \tag{4.5}$$

## 4.4 Evaluation

As a final step in our work, we tried two evaluate the quality of the produced models and compare them to not only one another, but also to the models from previous works.

| | KinectV2 | Phab2Pro |
|---|---|---|
| RMSERMSE [mm] | 0.0633 | 0.2328 |
| R-square | 0.9982 | 0.9992 |

Table 4.5: Axial Model Fit Key Figures

The quality of our fitted models was estimated in two ways: statistically, by comparing key figures of the fitted models and experimentally, by measuring a simple object in a real-world situation.

### 4.4.1 Statistical Evaluation

The first method is used to evaluate the distance- and rotation-based model. It was applied all throughout the model finding process, as the key figures used to judge a models quality are displayed in the MATLAB curve fitting toolbox (as seen in Table 4.5). With our model, we were able to achieve R-square values close to 1. This means that almost all of the variance in the data is described by our model. The values of the root-mean-square error (RMSE) for both models are also significantly below 1 mm. Although the RMSE for the Phab2Pro is noticeably higher than the value for the KinectV2, which is probably due to the lower signal-to-noise ratio, it is still rather small. Overall one can assume, based on this values, that the resulting models are appropriate for estimating the axial noise for both sensors.

For comparison, the RMSE of [FBR+15] was stated as 0.002 mm, which is approximately 30 times lower than ours. The reason for this is not easily identifiable, but might be a result of different test conditions. It might also be based on the fact that our axial model included measurements from two different rotations, whilst theirs only used one. For the Phab2Pro no comparable papers exist up to date.

### 4.4.2 Experimental Evaluation

The second method of evaluation used a simple setup in a real-world situation and was used for our combined model. The setup consisted of a cube of the dimensions 300 mm × 300 mm × 300 mm that was measured with both sensors. The cube was placed at different positions, distances and rotations (to show more or less faces) and the sensor was hand held. Specifically, the cube was measured with one face, two faces or three faces turned towards to camera. For the first two positions, the camera looked straight at the cube, while for the three faced position the sensor was above it. Distance wise, the cube was placed in one measurement series close to the sensor at 1 m and for another series farther away at 2 m. For each frame, the lateral and axial noise got roughly estimated by an approach, similar to the one explained in 3.5.2. But instead of taking 100 measurements, only a single one was made. The measured noise was then compared to the estimations made by our combined model. Because of the simplicity of the cube's

| | KinectV2 | Phab2Pro |
|---|---|---|
| Model 1 | 0.8947 | 2.5146 |
| Model 2 | 1.5197 | 4.1739 |
| Combined (Averaged) | 1.1780 | 2.0438 |
| Combined (Weighted) | 0.8926 | 1.7856 |

Table 4.6: Axial evaluation results (RMSE [mm]) for both sensors

| | KinectV2 | Phab2Pro |
|---|---|---|
| X - Combined (Averaged) | 94.4 | 93.4 |
| Y - Combined (Averaged) | 94.4 | 94.8 |

Table 4.7: Percent of all values below our estimations of the lateral noise calculated by our final lateral models

surface and the fixed positions, the model parameters like rotation and distance could be automatically determined.

For the axial model we calculated the RMSE of the difference between the measured values and the predicted ones. As for the lateral noise components, we chose to verify, if our model could serve as an upper border by checking, if the noise values are under the predicted values 90% of the cases. This way of verifying our lateral model was chosen, because the seemingly random structure of the noise was not predictable with any of our models and was only defined as the 90-percentile of the measurements.

The axial results, seen in Figure 4.8, show the previously mentioned circumstance of Model 1 being better at higher angles and Model 2 being better at lower angles. By applying a weight function to both models, we could adapt to this and generated a weighted combined model that is better than its initial components.

The lateral results, seen in Figure 4.10, show that both directions of our combined lateral models are most of the time above the measured values, thus they can be seen as a good upper border for the lateral noise. More concretely, the percentage of values that are correctly under our border is around 90% in all instances. In contrast to the axial model, the lateral models do not seem to need an additional weight function and could be simply averaged.

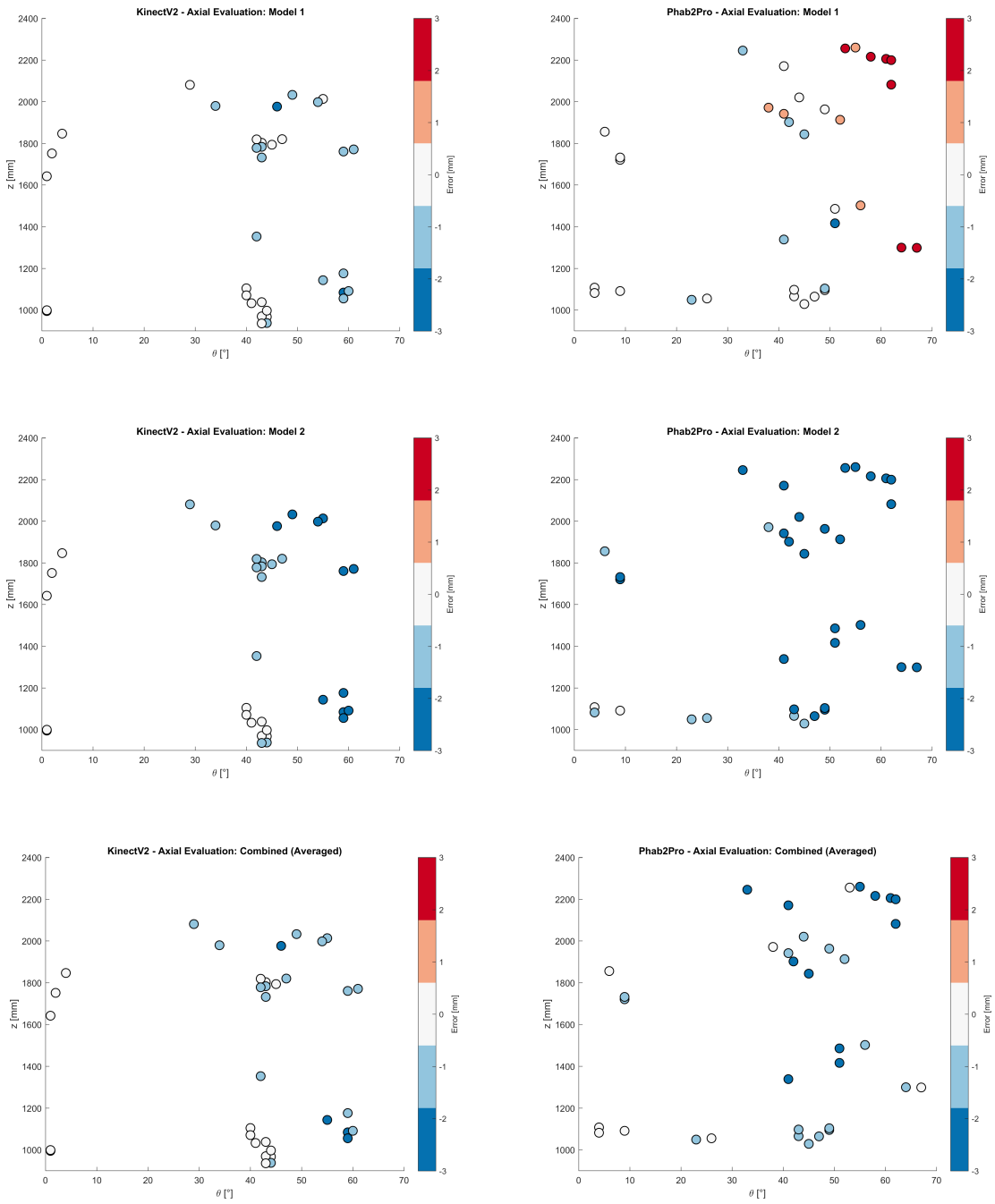



Figure 4.8: Results of the experimental evaluation of our separate models and the average model for both sensors. The dot position represents its detected distance and rotation, while its color represents how much the predicted value differs from the measured one.

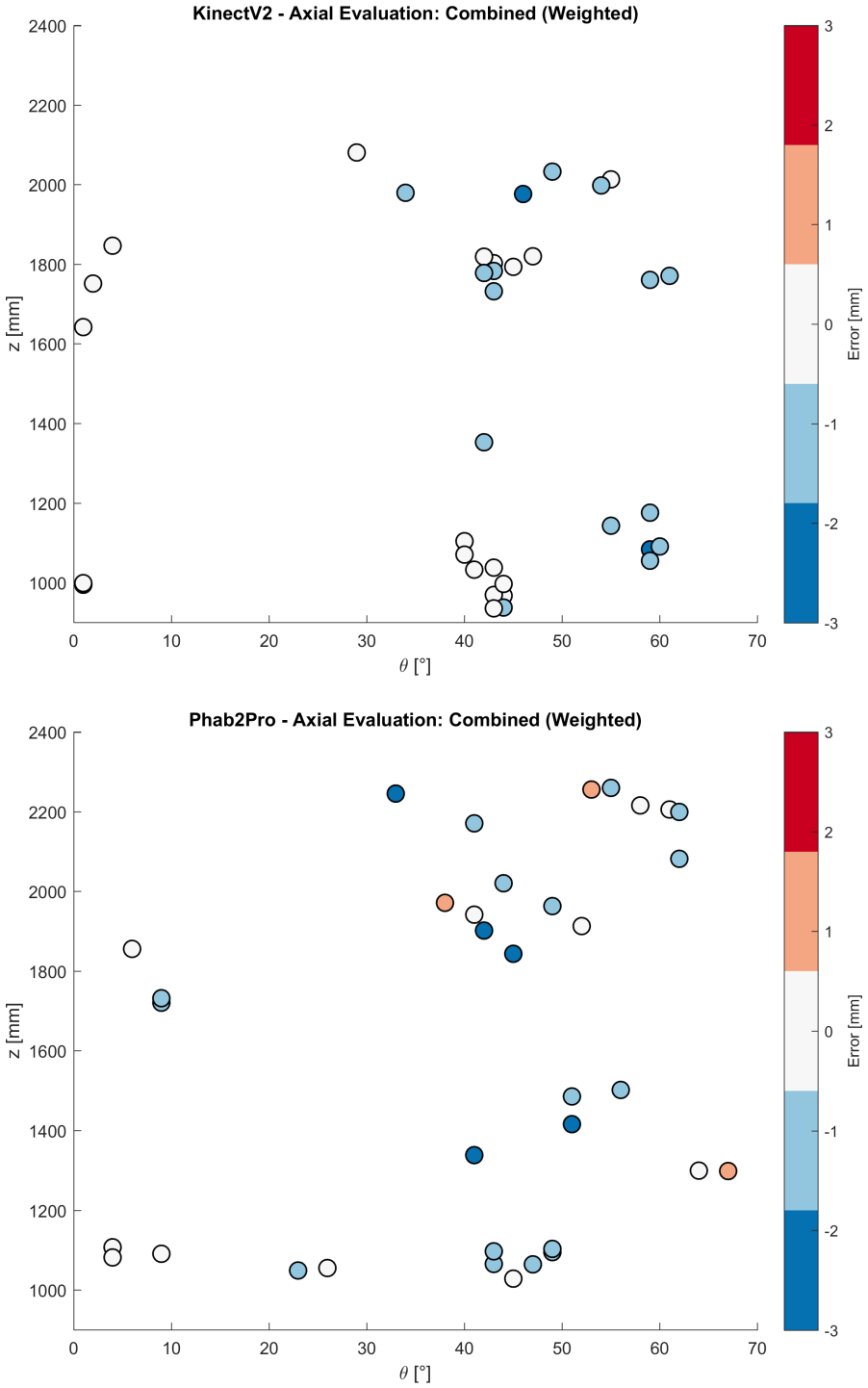



Figure 4.9: Results of the final axial model for both sensors

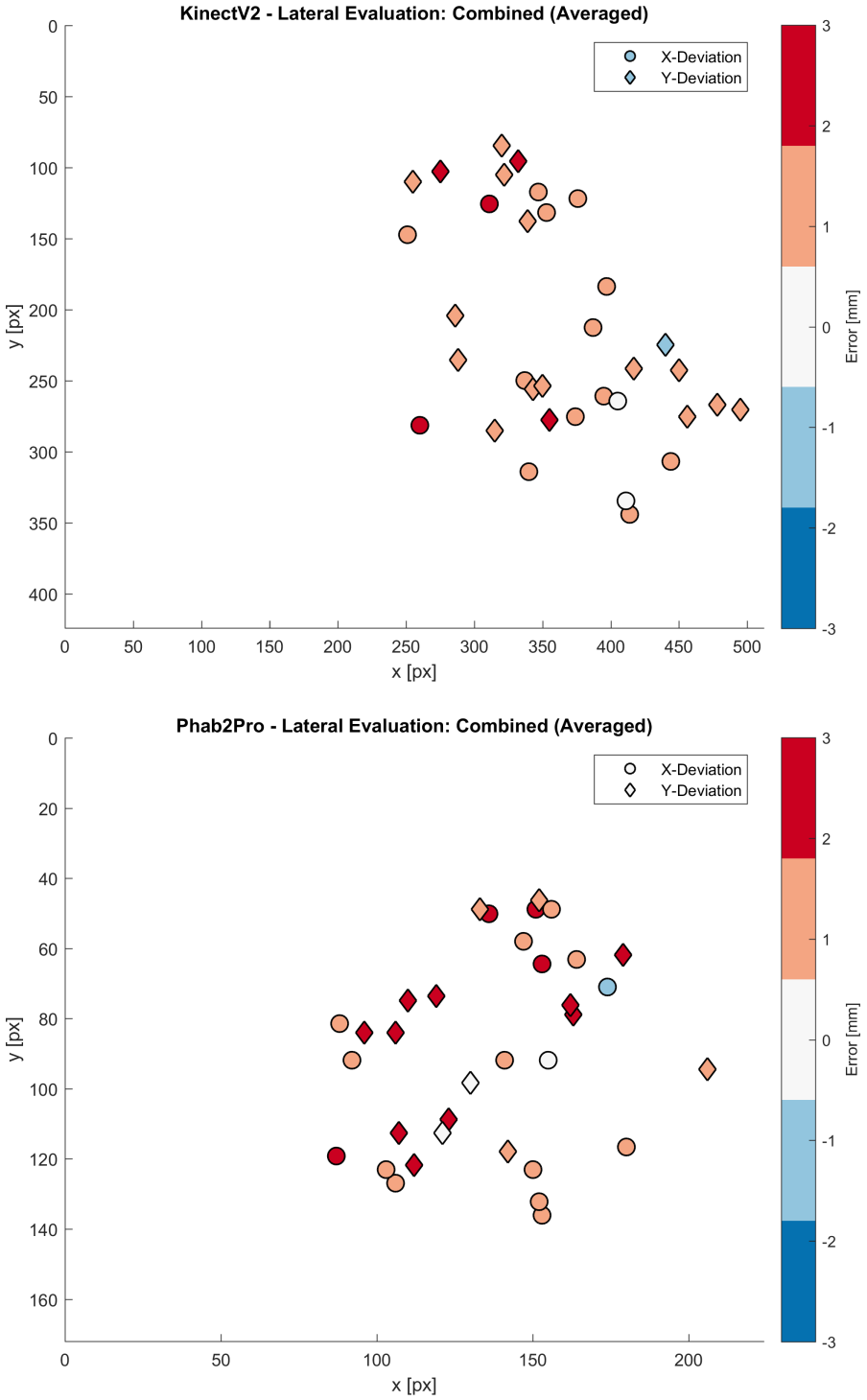



Figure 4.10: Results of the final lateral models for both sensors

CHAPTER 5

# Conclusion

In this thesis we presented two different models for estimating the sensor noise in the axial and both lateral directions. Similar models were already calculated for the KinectV2, but as of now we are the first to do this kind of experiment with the Phab2Pro.

We used a test setup consisting of a rotating planar target that was measured at different distances and rotations to estimate the noise. At each position two measurement series were made, one with the camera in a normal (horizontal) position and one with the camera (vertically) rotated 90° to calculate eventual differences between both lateral directions. Furthermore, we showed an automated pipeline to extract the sensor noise.

From these noise measures we derived an empirical model that used the object's distance and rotation as parameters. To further improve this model we combined it with the one of my colleague Thomas Köppel [Kö17], whose model used the image coordinates and the distance of the object as parameters. For the merging of the two separate models, we showed the need for a weight function that decides when to use which model. We used a simple linear function based on the object's rotation.

Finally, our empirical models were validated, using a simple setup, where a cube is placed in a scene with different distances, positions and rotations. For each face, the local noise is roughly estimated and compared to the predictions of our model. The results of the evaluation indicate that the combined axial model using a weight function is superior to its component models. A similar conclusion can be drawn for both lateral models.

Future work could include attempts to improve the combined model like using a more refined weight function joining the component models. The lateral noise component could be further analyzed to come up with a more concrete model. Furthermore, our generated noise models could be used to enhance KinectV2 and Phab2Pro depth data, possibly leading to improved reconstruction results.

## 5.1 Resulting Models

| | KinectV2 | Phab2Pro |
|---|---|---|
| $\sigma_{x_1}$ | 2.9110 | 4.1207 |
| $\sigma_{y_1}$ | 1.9617 | 3.6665 |

Table 5.1: Model 1 - Lateral Models (90-percentile)

$$\sigma_{z_1}(z,\theta) = a + b * z + c * z^2 + d * z^e * \frac{\theta^2}{(\frac{\pi}{2} - \theta)^2} \tag{5.1}$$

| | KinectV2 | Phab2Pro |
|---|---|---|
| $a$ | 2.094 | 0.3019 |
| $b$ | $-1.099 * 10^{-3}$ | $5.712 * 10^{-4}$ |
| $c$ | $4.048 * 10^{-7}$ | $6.183 * 10^{-7}$ |
| $d$ | $6.846 * 10^{-7}$ | $2.386 * 10^{-5}$ |
| $e$ | 1.7 | 1.47 |

Table 5.2: Model 1 - Axial Model Coefficients (Equation 5.1)

$$\sigma_{x_2/y_2}(z) = \alpha_{[1]}z^3 + \alpha_{[2]}z^2 + \alpha_{[3]}z + \alpha_{[4]} \tag{5.2}$$

[Kö17]

| | KinectV2 X | KinectV2 Y | Phab2Pro X | Phab2Pro Y |
|---|---|---|---|---|
| $\alpha[1]$ | 6.6987e-09 | 3.9018e-09 | $4.3031 * 10^{-9}$ | $1.9045 * 10^{-9}$ |
| $\alpha[2]$ | -3.1781e-05 | -1.3349e-05 | $-1.8169 * 10^{-5}$ | $-5.0756 * 10^{-6}$ |
| $\alpha[3]$ | 0.0518 | 0.013148 | 0.0284 | 0.0066 |
| $\alpha[4]$ | -23.4839 | 1.8268 | -11.4981 | -0.6622 |

Table 5.3: Model 2 - Lateral Model Coefficients (Equation 5.2) [Kö17]

| | KinectV2 Z | Phab2Pro Z |
|---|---|---|
| $\alpha[1]$ | 6.569 | -20.2165 |
| $\alpha[2]$ | -0.0063664 | 0.025415 |
| $\alpha[3]$ | 4.5605e-06 | -1.7253e-06 |
| $\alpha[4]$ | -3.2304 | 11.28 |
| $\alpha[5]$ | 0.0029717 | -0.01511 |
| $\alpha[6]$ | -1.6241e-06 | 3.0272e-06 |
| $\alpha[7]$ | 0.46318 | -1.5463 |
| $\alpha[8]$ | -0.00042755 | 0.0019859 |
| $\alpha[9]$ | 2.088e-07 | -4.2756e-07 |
| $\alpha[10]$ | -2.9137 | 11.0018 |
| $\alpha[11]$ | 0.0027723 | -0.01431 |
| $\alpha[12]$ | -1.6192e-06 | 2.6006e-06 |
| $\alpha[13]$ | 2.0513 | -4.9918 |
| $\alpha[14]$ | -0.0021142 | 0.0070606 |
| $\alpha[15]$ | 8.908e-07 | -1.5391e-06 |
| $\alpha[16]$ | -0.27216 | 0.6874 |
| $\alpha[17]$ | 0.00028939 | -0.00097463 |
| $\alpha[18]$ | -1.1505e-07 | 2.2922e-07 |
| $\alpha[19]$ | 0.30466 | -1.2972 |
| $\alpha[20]$ | -0.00025628 | 0.0015642 |
| $\alpha[21]$ | 1.6642e-07 | -2.8225e-07 |
| $\alpha[22]$ | -0.20004 | 0.54637 |
| $\alpha[23]$ | 0.00020537 | -0.00078213 |
| $\alpha[24]$ | -9.2035e-08 | 1.7218e-07 |
| $\alpha[25]$ | 0.026884 | -0.076423 |
| $\alpha[26]$ | -2.8628e-05 | 0.00010955 |
| $\alpha[27]$ | 1.1963e-08 | -2.6047e-08 |
| $\alpha[28]$ | -0.0037752 | 0.013358 |
| $\alpha[29]$ | -0.0029981 | 0.0101 |
| $\alpha[30]$ | -1.9996e-10 | -5.6216e-10 |

Table 5.4: Model 2 - Axial Model Coefficients (Equation 5.3) [Kö17]

$$
\begin{aligned}
\sigma_{z_2}(x,y,z) =& \alpha_{[1]} + \alpha_{[2]}z^1 + \alpha_{[3]}z^2 + \alpha_{[4]}y^1 + \alpha_{[5]}y^1z^1 + \\
& \alpha_{[6]}y^1z^2 + \alpha_{[7]}y^2 + \alpha_{[8]}y^2z^1 + \alpha_{[9]}y^2z^2 + \alpha_{[10]}x^1 + \\
& \alpha_{[11]}x^1z^1 + \alpha_{[12]}x^1z^2 + \alpha_{[13]}x^1y^1 + \alpha_{[14]}x^1y^1z^1 + \alpha_{[15]}x^1y^1z^2 + \\
& \alpha_{[16]}x^1y^2 + \alpha_{[17]}x^1y^2z^1 + \alpha_{[18]}x^1y^2z^2 + \alpha_{[19]}x^2 + \alpha_{[20]}x^2z^1 + \\
& \alpha_{[21]}x^2z^2 + \alpha_{[22]}x^2y^1 + \alpha_{[23]}x^2y^1z^1 + \alpha_{[24]}x^2y^1z^2 + \alpha_{[25]}x^2y^2 + \\
& \alpha_{[26]}x^2y^2z^1 + \alpha_{[27]}x^2y^2z^2 + \alpha_{[28]}x^3 + \alpha_{[29]}y^3 + \alpha_{[30]}z^3
\end{aligned}
\tag{5.3}
$$

[Kö17]

$$w = \max\left(\frac{w_a - \theta}{w_a} * w_b + w_c, w_c\right) \tag{5.4}$$

$$\sigma_x = (1 - w)\sigma_{x_1} + w\sigma_{x_2} \tag{5.5}$$
$$\sigma_y = (1 - w)\sigma_{y_1} + w\sigma_{y_2} \tag{5.6}$$
$$\sigma_z = (1 - w)\sigma_{z_1} + w\sigma_{z_2} \tag{5.7}$$

| | KinectV2 | Phab2Pro |
|---|---|---|
| $w_a$ | 29 | 8 |
| $w_b$ | 0.3 | 0.2 |
| $w_c$ | 0 | 0.3 |

Table 5.5: Weight Function - Coefficients

# List of Figures

# List of Tables

# Bibliography


[BBA14] Timo Breuer, Christoph Bodensteiner, and Michael Arens. Low-cost commodity depth sensor comparison and accuracy analysis. In *SPIE Security+ Defence*, pages 92500G–92500G. International Society for Optics and Photonics, 2014.

[BBW+11] Maged N Kamel Boulos, Bryan J Blanchard, Cory Walker, Julio Montero, Aalap Tripathy, and Ricardo Gutierrez-Osuna. Web gis in practice x: a microsoft kinect natural user interface for google earth navigation. *International journal of health geographics*, 10(1):45, 2011.

[Bou04] Jean-Yves Bouguet. Camera calibration toolbox for matlab. *Caltech Technical Report*, 2004.

[CLDB14] Benjamin Choo, Michael Landau, Michael DeVore, and Peter A Beling. Statistical analysis-based error models for the microsoft kinecttm depth sensor. *Sensors*, 14(9):17430–17450, 2014.

[Cor13] Microsoft Corporation. Kinect hardware. `https://developer.microsoft.com/en-us/windows/kinect/hardware`, 2013. Accessed: 2017-07-07.

[Cor14] Microsoft Corporation. Kinect for windows v2 sensor. `https://news.microsoft.com/kinect-for-windows-v2-sensor-2`, 2014. Accessed: 2017-07-07.

[ElHY12] Riyad A El-laithy, Jidong Huang, and Michael Yeh. Study on the use of microsoft kinect for robotics applications. In *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, pages 1280–1288. IEEE, 2012.

[FBR+15] Péter Fankhauser, Michael Bloesch, Diego Rodriguez, Ralf Kaestner, Marco Hutter, and Roland Siegwart. Kinect v2 for mobile robot navigation: evaluation and modeling. In *Advanced Robotics (ICAR), 2015 International Conference on*, pages 388–394. IEEE, 2015.

[For] Inc Forbes. South korea is using kinect to patrol the dmz. `https://www.forbes.com/sites/erikkain/2014/02/03/`

south-korea-is-using-kinect-to-patrol-the-dmz. Accessed: 2017-06-07.

[Goo] Inc Google. Calibrating your tango device. https://developers.google.com/tango/hardware/calibration. Accessed: 2017-04-17.

[GPC11] Luigi Gallo, Alessio Pierluigi Placitelli, and Mario Ciampi. Controller-free exploration of medical image data: Experiencing the kinect. In *Computer-based medical systems (CBMS), 2011 24th international symposium on*, pages 1–6. IEEE, 2011.

[Kö17] Thomas Köppel. Extracting Noise Models considering X Y and Z Noise. Bachelor's thesis, TU Wien, Austria, 2017.

[KE12] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[LMM+15] E Lachat, H Macher, MA Mittet, T Landes, and P Grussenmeyer. First experiences with kinect v2 sensor for close range 3d modelling. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(5):93, 2015.

[Ltd] Lenovo Group Ltd. Phab 2 pro smartphone. http://www3.lenovo.com/us/en/smart-devices/-lenovo-smartphones/phab-series/Lenovo-Phab-2-Pro/p/WMD00000220. Accessed: 2017-07-07.

[NIL12] Chuong V Nguyen, Shahram Izadi, and David Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 524–530. IEEE, 2012.

[RMYZ11] Zhou Ren, Jingjing Meng, Junsong Yuan, and Zhengyou Zhang. Robust hand gesture recognition with kinect sensor. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 759–760. ACM, 2011.

[SLK15] Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb. Kinect range sensing: Structured-light versus time-of-flight kinect. *CoRR*, abs/1505.05459, 2015.

[TM17a] Inc The MathWorks. kstest. https://de.mathworks.com/help/stats/kstest.html, 2017. Accessed: 2017-08-07.

[TM17b] Inc The MathWorks. What is camera calibration? https://de.mathworks.com/help/vision/ug/camera-calibration.html, 2017. Accessed: 2017-04-17.

[Wie17] Thiemo Wiedemeyer. Tools for using the kinect one (kinect v2) in ros. `https://github.com/code-iai/iai_kinect2`, 2017. Accessed: 2017-04-17.

[XCA11] Lu Xia, Chia-Chih Chen, and Jake K Aggarwal. Human detection using depth information by kinect. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 15–22. IEEE, 2011.

[Zha00] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.