FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Extracting Noise Models – considering X / Y and Z Noise

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Thomas Köppel
Matrikelnummer 1327052

an der Fakultät für Informatik

der Technischen Universität Wien

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

Wien, 28. August 2017

Thomas Köppel	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Extracting Noise Models – considering X / Y and Z Noise

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Thomas Köppel
Registration Number 1327052

to the Faculty of Informatics

at the TU Wien

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

Vienna, 28th August, 2017

Thomas Köppel | Michael Wimmer

Erklärung zur Verfassung der Arbeit

Thomas Köppel
Hauptstraße 7a, 7344 Stoob

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

Thomas Köppel

Acknowledgements

I would like to express my gratitude to our supervisors, Dr. Stefan Ohrhallinger and Dr. Michael Wimmer, for the great support during our work. Their feedback and constructive contributions made the execution of this thesis possible. Furthermore, I would like to thank my dear colleague, Nicolas Grossmann, for the cooperation and the support during many months of intense work at the institute and beyond and for proofreading this thesis.

I would also like to thank the Institute of Computer Graphics and Algorithms of the Technical University of Vienna for the financial support and the education making this work possible.

In addition, I am very thankful for the support of my family.

Abstract

We have developed two different test setups allowing the characterization of noise in X, Y and Z direction for the KinectV2 and the Phab2Pro depth sensors. We have combined these two methods, generating a single noise model allowing a prediction of the amount of noise in specific areas of an image in the three respective directions at a certain distance and rotation. We have conducted two test setups and measured the noise from 900 mm to 3.100 mm for the generation of the noise models. The test setup of this thesis focused on determining the noise in X, Y and Z direction, covering the whole frustum of the respective depth sensor. In this thesis, Z noise was measured against a wall and X and Y noises were measured using a 3D chequerboard that was shifted through the room, allowing the above mentioned coverage of the whole frustum. Along the edges of the cells of the chequerboard, the X and Y noise was measured. The combined model was evaluated by using a solid cube to classify the quality of our noise model.

Contents

CHAPTER 1

Introduction

Depth sensors are used in various areas nowadays like computer vision, augmented reality, human computer interaction, the gaming industry and many more [NIL12]. Over the last few years, many affordable depth sensors like the KinectV1, KinectV2 and the Lenovo Phab2Pro arrived on the market, offering the technique of using depth sensors to a wider audience. But the resulting captures from depth sensors (depth maps or pointclouds) suffer from noise. The identification and estimation of this noise is the main task of this thesis.

In general, our project consists of two parts focusing on different test setups. These two setups got combined and evaluated. The setup of my colleague focused on axial and lateral noise measurement, using a plane placed in the centre of the captures with different rotations. My setup focused on the work by Choo et al. [CLDB14]. The main contribution of this paper was to identify the behaviour of noise when considering the whole frustum of the KinectV2.

1.1 Noise Model Calculation – Previous Work

Many different projects have identified the problem of noisy depth maps or point clouds and have come up with different models describing it. Especially, affordable depth sensors show a lower signal-to-noise ratio. Therefore, the extraction and description of this noise can be crucial in many applications in computer vision or robotics. Even surfaces can be reconstructed more appropriately, avoiding holes and noisy structures by using noise models [NIL12].

1.1.1 Modeling Kinect Sensor Noise for Improved 3D Reconstruction and Tracking

"Modeling Kinect Sensor Noise for Improved 3D Reconstruction and Tracking", published by Nguyen et al. [NIL12], was one of the first projects not only concentrating on axial noise in Z direction but also considering the lateral component, leading to better results especially in surface reconstruction. The main focus and contribution of their thesis was identifying axial as well as lateral noise in the centre of the depth maps by considering different rotation angles and distances.

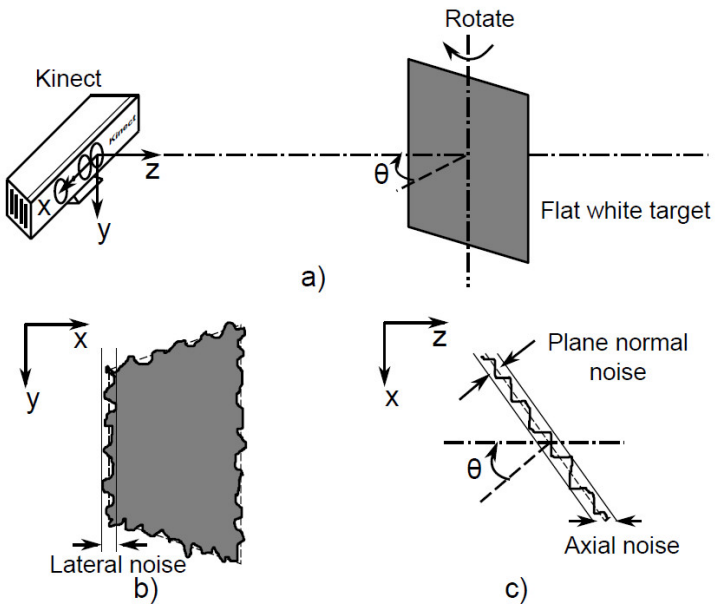

Figure 1.1: Test setup from Nguyen et al. [NIL12][p. 2]

Figure 1.1 shows their basic test setup. [NIL12] They used the KinectV1 sensor and measured the axial and lateral noise, using a plane, rotated to different orientations. The axial noise was measured along the plane's normal, while the lateral part was measured at the left and right edge of the plane.

The results of their measurements were included into the KinectFusion pipeline for 3D reconstruction, leading to better results, less holes and less noise – which increased the reconstruction accuracy. The main findings were the linear increase of lateral noise with distance and a quadratic increase of axial noise. [NIL12]

1.1.2 Kinect v2 for Mobile Robot Navigation: Evaluation and Modeling

"Kinect v2 for Mobile Robot Navigation: Evaluation and Modeling" [FBR+15], published by Fankhauser et al., focused on generating noise models allowing robots to navigate more appropriately in indoor and outdoor scenarios. They developed two different models allowing an estimation of axial and lateral noise, the second model took the influence of sunlight into account. The KinectV2 sensor was used for their measurements.

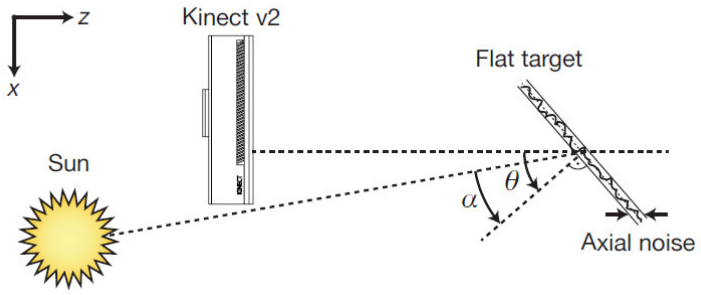

Figure 1.2: Test setup from Fankhauser et al. [FBR+15][p. 5]

Similar to Nyguen et al. [NIL12], the test setup consisted of a depth sensor centred at a plane at different distances and rotation angles (see Figure 1.2). Their main contribution was the introduction of the sunlight angle into their outdoor noise model, allowing robots to perceive their environment more appropriately. [FBR+15]

Using the KinectV2, the noise models showed a lower axial noise compared to the KinectV1, especially at higher distances [FBR+15]. The lateral noise was chaotic and described by a constant value, using the 0.9 percentile.

1.1.3 Statistical Analysis-Based Error Models for the Microsoft KinectTM Depth Sensor

"Statistical Analysis-Based Error Models for the Microsoft KinectTM Depth Sensor" [CLDB14], by Choo et al., proposed a new technique of acquiring noise models by considering the whole frustum of the KinectV2 depth sensor. Their main contribution was including the position of an object in the depth image to the noise model.

The axial noise was measured against a flat surface. The lateral part was extracted using a chequerboard made from LEGO which covered the whole frustum of the captures, allowing lateral noise calculation at the borders of the cells (see Figure 1.3) [CLDB14]. The generated noise model, which described their measurements, showed a larger lateral

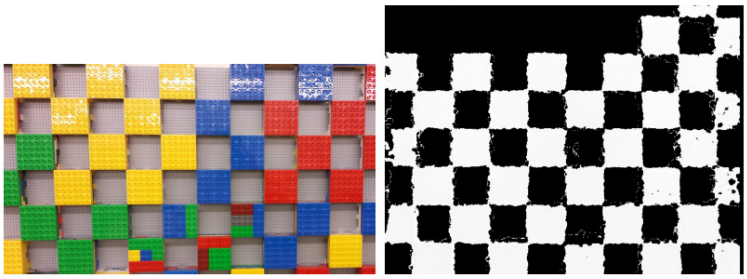

Figure 1.3: Test setup from Choo et al. [CLDB14][p. 17441]

noise in both X and Y direction compared to the model of Nyguen et al. [NIL12]. Choo et al [CLDB14] showed the importance of the pixel's location relative to the centre, potentially increasing the quality of applications using the KinectV2.

1.2 Content and Structure of this Thesis

This thesis focuses on one part of our two noise models, describing the noise extraction similar to the work by Choo et al. [CLDB14]. We used a flat surface for the axial noise and a chequerboard for the lateral noise and measured at different distances for both the KinectV2 and the Phab2Pro, covering the whole frustum of the respective depth sensor. Afterwards, we combined these models and evaluated the resulting noise model, using a real world scenario featuring a pressboard cube.

The next chapters focus on the detailed description of my setup and the generation and evaluation of the combined noise model. Chapter 2 explains the terminology and theoretical aspects needed for the understanding of depth sensors in general and the techniques used for the noise extraction and model generation. Chapter 3 describes the test setup and the procedure of capturing the axial noise in Z direction and the lateral noise in X and Y direction, using a wall and a 3D chequerboard. Afterwards, the image processing steps are explained in detail, allowing the image to be processed accordingly. The following subsections explain the preprocessing and extraction steps. Chapter 4 focuses on the description of the extracted noise model, using three improvements and combining this model with the model of my colleague and evaluating it using a pressboard cube.

CHAPTER 2

Theory

This chapter is going to give an introduction to different terms and techniques we used during the project in order to gain a better understanding of the process. The most important theoretical aspects are going to be explained briefly.

2.1 Depth Sensors

Depth sensors are used to measure distances in an observed scene. There are several different sensors from different companies, from low-budget to high-demand ones. Our project focused on two different depth sensors – the KinectV2 sensor, from Microsoft, and the depth sensor integrated into the Lenovo Phab2Pro.

There are two common techniques of how depth sensors acquire their measurements: the structured-light and the time-of-flight method [SLK15].

The **structured-light** method follows an active stereo-vision technique [SLK15]. A projector emits a pattern onto the scene that gets deformed based on the objects in it. A camera measures the scene from a slightly different position. Comparing the distorted, captured pattern to the original one, the corresponding depths can be calculated. Figure 2.1 shows an explanatory scene where a pattern is projected onto a statue.

Time-of-flight cameras [SLK15] measure the time it takes light to return to the camera for the calculation of the depth values. An illumination unit is used that emits infra-red light into the scene, which is then reflected back to a sensor array. Considering the time it takes to sense the reflected infra-red light, the distances of the objects in the scene can be calculated. Time-of-flight cameras have the advantages of being simple and compact, the illumination unit and the sensor are close together, unlike in stereo vision. Another

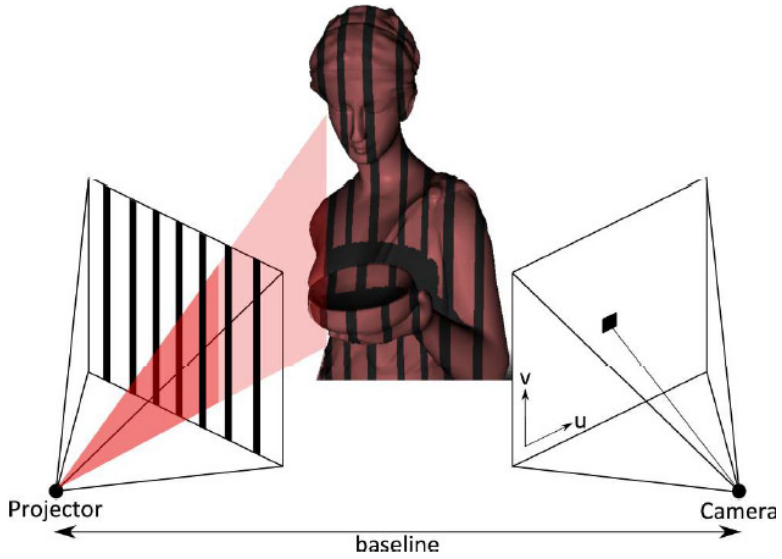

Figure 2.1: Principle of Structured Light Sensors [SLK15][p. 6]

advantage is the high speed – no complex algorithms are needed during the computation – and the lower noise. These cameras can easily be integrated into real-time applications because of the high speed.

2.1.1 Error Sources for Structured-Light and Time-of-Flight Sensors

When measuring distances in a scene, using a depth sensor, many factors can influence the result, which need to be taken into account.

- **Background Illumination** can lead to an over-saturation. Kinect sensors use a filter to remove too high background light in order to cope with that problem. [SLK15]
- **Interference Problems** can occur when using several depth sensors simultaneously. [SLK15]
- **Temperature Change**. Especially time-of-flight sensors need to be cooled actively in order to compensate the high temperature caused by the power consumption of the illumination unit. The warm up phase before the cooling system activates can lead to a slight shifting in distances. [SLK15]
- **Systematic Error**. The measurements can suffer from systematic errors, which can be reduced easily by a camera calibration. [SLK15]
- **Multi-Path Effects** occur if the light emitted by the sensor does not take the direct way to the object and back to the sensor due to scattering of reflective

objects. [SLK15] During our research we noticed that even quite diffuse objects led to a reflection of light at higher surface angles, especially at the setup from my colleague [Gro17].

2.1.2 Kinect

The Microsoft depth sensor KinectV1 was one of the first depth sensors being introduced to the gaming industry. It allows the control of avatars in real-time games without the need of a joystick or another controller. A person can simply interact with the game, using the own body. Later on, the KinectV2 got introduced combined with the XBoxOne.

Both of these sensors return the same type of data but their internal systems vary. The KinectV1 is based on the structured-light approach, while the KinectV2 uses the time-Of-flight method.

Due to the high success of these depth sensors, Microsoft published an SDK allowing programmers to create their own applications for the Kinect sensors. Using this SDK, we could easily obtain the required data.

We used the KinectV2 in our work. Figure 2.2 shows a picture of the KinectV2.

Figure 2.2: Picture of the KinectV2

2.1.3 Phab2Pro

The Phab2Pro, manufactured by Lenovo, was the first smartphone with the Google Tango project, which allows depth sensing. Their own apps have been published, utilizing the new feature of depth measuring on a smartphone. The Phab2Pro includes a time-of-flight depth sensor, which allows these measurements.

Using the Project Tango API, by Google, allowed us to access the point clouds generated by the device. We programmed an Android application allowing the capturing of several point clouds. Figure 2.3 shows the back side of a Phab2Pro with its depth sensor.

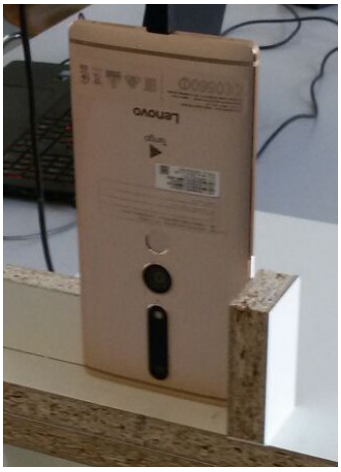

Figure 2.3: Picture of the Phab2Pro

2.2 Noise: X, Y, Z

The main focus of this thesis is the generation of a noise model that describes the noise of the KinectV2 and the Phab2Pro depth sensors considering the axial part in Z direction and the lateral part in X and Y direction. The noise is calculated based on differences between the ground truth and the measurements. Figure 2.4 shows the difference between axial and lateral noise.

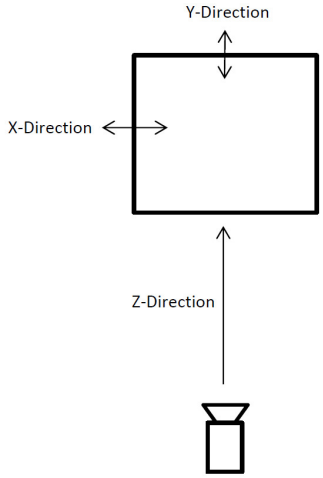

Figure 2.4: Explanation – Axial and Lateral Noise

Axial noise is measured along the viewing direction of the respective depth sensor. Deviations along this axis are considered being axial in Z direction. It describes the difference between the measured depth and the actual depth. We used a plane modelled according to the data and measured the deviations of each pixel to that plane for the

noise calculation.

Lateral noise is measured at the axes perpendicular to the Z axis. The lateral noise in X direction is measured at the left and right edges of a rectangle, while the lateral noise in Y direction is measured at the top and bottom edges.

2.3 Root-Mean-Square-Error (RMSE)

We used the Root-Mean-Square-Error as a quality measure of our resulting noise models. This error indicates how accurately the fitted model describes the data points. The Root-Mean-Square-Error is defined as the square root of the variance of an estimator.

In our case, the RMSE is the root of the squared difference between the estimated noise of the model and the actual measured noise. The lower the RMSE is, the better the model fits the data. The error tells how many mm – in our case – the noise model differs from the data.

$$RMSE = \sqrt{(\text{Noise of Model} - \text{Noise of Data})^2}$$

2.4 Scale Invariant Blob Detection – Laplacian Blob Detector

Scale Invariant Blob Detection is a scale-invariant way of detecting blobs in images, meaning that blobs can be detected independently of their respective size. The Laplacian Blob Detector was used in this thesis. It is a differential operator of second order. [Sze10]

When using Laplacian Blob Detection, two steps are performed when searching for blobs in an image [Sze10]: Firstly, a spatial selection of interest points and secondly, an assignment of a scale. Using a Laplacian of Gaussian kernel results in a maximum negative response at light blobs and a maximum positive response at dark blobs. The respective scale of the Laplacian is then "matched" to the scale of the blob. Figure 2.5 shows the maximum negative response of a blob. The radius used in this figure is 8, which means that blobs with a size of 8 get detected. If the Laplacian operator hits the respective blobs with this size, it produces a maximum response. The position of this response is the spatial location of the blob, while the radius $\sigma = 8$ represents the size.

The complete algorithm works as follows: [Sze10] It starts with an initial scale σ, which can be selected manually. Each iteration increases the scale by multiplying it with a constant factor k. A scale-normalized Laplacian of Gaussian kernel is used to convolve

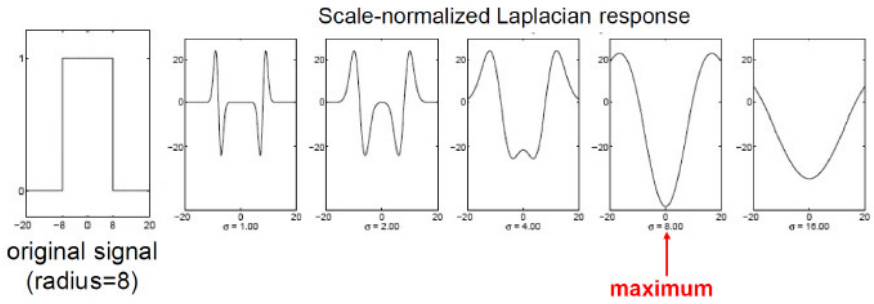

Figure 2.5: Blob Detection with the Laplacian Operator [Sze10]

the image with the respective scale σ. The absolute responses of the kernel are then saved for each level in order to detect both light and dark blobs. Depending on the number of iterations, a 3D scale space of different size results from this operation. Each slice of this space represents a convolution of the image with the Laplacian of Gaussian kernel using the respective scale σ. To identify the blobs in this 3D scale space, a 3x3x3 cube is marching through it. If the centre value is the largest value in the cube, a blob is detected at the size of the corresponding σ of this level at the location of the maximum.

We used the Laplacian Operator in our work to automatically detect each square of the chequerboard, allowing noise extraction at the borders without having to select the rectangles manually.

2.5 RANSAC

RANSAC (Random Sample Consensus) [FB81] is a paradigm of fitting a model to specific data. It is able to identify and remove so called "gross errors" in the data, allowing a better model generation. It is, for example, used after SIFT [Low99] when stitching images [Sze10], identifying false matches and removing them.

When applying RANSAC [FB81], random data points are chosen in the first step [FB81]. The number depends on the purpose. For example, when stitching images [BL07] considering the SIFT [Low99] features, 4 points are chosen. Based on the selected data points, a homography is calculated in the case of image stitching [BL07] or generally a model is fitted. The number of inliers is the number of data points that are below a certain distance to the model. This number is saved in each iteration. This procedure is applied many times (for example, 1000 times in our case). After the iterations, the inliers of the best model (the one with the most inliers) are used for the fitting of the final model.

"Gross errors" can be erased from the data points in order to allow a suitable model fitting. In our work we used RANSAC before the noise models were fitted through the

measured data points, allowing a more accurate model. About 3 - 5% of the datapoints were considered being "gross errors". Removing them improved the RMSE (cut it in half).

CHAPTER 3

Method

3.1 Test Setup

As mentioned beforehand, our noise model combines two different test setups. The first test setup measures axial and lateral noise considering the distance and the rotation of a plane placed in front of a depth sensor. The second one measures axial and lateral noise considering the distance and the location in the depth image. This thesis focuses on the second test setup. The thesis of my colleague describes the first setup in detail. [Gro17]

3.1.1 Previous Setup

The axial and lateral noise based on the distance and the location in the depth image was measured by Choo et al. [CLDB14]. Considering the axial noise, they measured against a flat surface and for the lateral noise in X and Y direction a 3D chequerboard was used and they obtained their values at the edges of the squares of the chequerboard. Both measurements covered the whole frustum of the KinectV2 sensor.

3.1.2 Our Setup – General Setup

Our test setup used the method explained by Choo et al. [CLDB14] using a flat plane and a chequerboard for the noise extraction. The depth sensors were placed onto a table in the respective distance, facing a wall. The chequerboard had a size of 76x76 cm. In order to allow a coverage of the whole frustum of the sensors, this board had to be moved along the wall. Therefore, a grid of masking tape was attached to the wall to avoid irregularities while measuring.

Figure 3.1 shows the chequerboard. It consists of 32 heightened squares allowing measurements along the edges. The basis consists of a plywood plane with 4 LEGO baseplates.

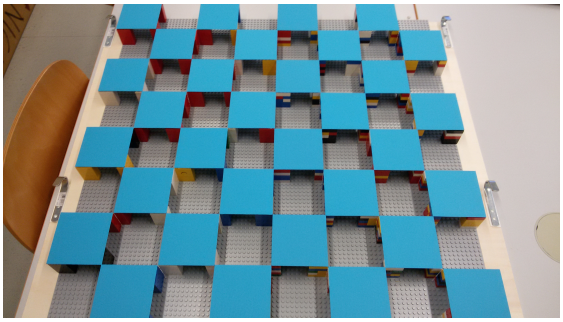

Figure 3.1: Chequerboard

The blue carton squares with a width of 9.2 cm were elevated using 5 4x2 Lego bricks plugged on top of each other. The squares were fixed using double faced adhesive tape (see Figure 3.2). Metallic handles were attached to the sides in order to allow proper holding of the board when moving it across the wall.

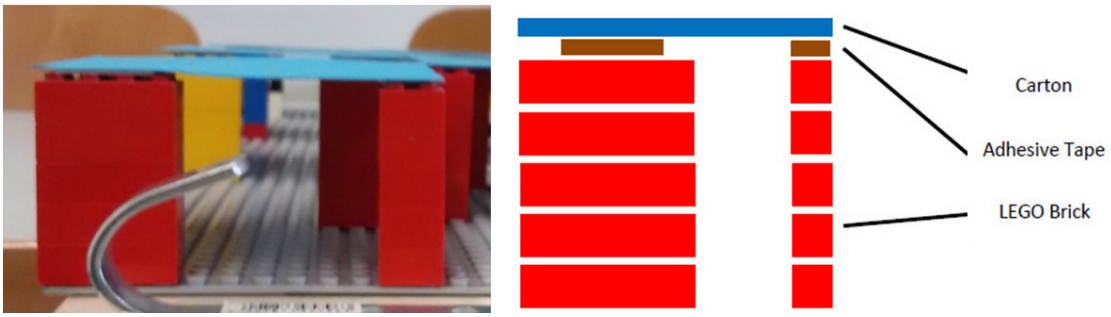

Figure 3.2: Single Square from the Side

3.2 Data Acquisition

Our test setup focused on noise calculation of the KinectV2 and the Lenovo Phab2Pro depth sensors. In order to allow our work to be able to handle different cameras, we split the application into two parts, a depth sensor specific application for the data extraction and a Matlab application for the noise calculation and noise model generation.

In order to calculate the axial and lateral noise of a scene, both depth maps and point clouds are needed. Depth maps are used to calculate the lateral noise by image recognition tools and a standard deviation calculation in Matlab. Point clouds are needed to calculate the axial noise, using plane fitting and again a calculation of the standard deviation.

The two sensors were produced by two different companies allowing data extraction over their own APIs. It is suitable to extract either a depth map or a point cloud. A depth

map can be transformed into a point cloud and vice versa.

3.2.1 Acquisition with the KinectV2

Considering the KinectV2 depth sensor, we used the Kinect for Windows SDK by Microsoft. It contains all the drivers needed to work with the sensor on a computer. Using their API, the data by the sensor can easily be used in a visual studio application. The API allows access to four different channels of the sensor for colour, depth, infra-red and audio. We created a C# WPF application allowing the data extraction using the depth channel. The Kinect for Windows API allows an event-based interaction of our application with the sensor and gets notified if frames arrive.

```
sensor = KinectSensor.GetDefault();

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

[Gro17]

We created a GUI-application (see Figure 3.3) for convenient execution of the depth measurements. The distance can be set over the respective buttons and 10 depth images were made when pressing the "Start" Button. These 10 captures were taken at each of the 9 positions (if a 3x3 grid is enough to cover the whole frustum) of the chequerboard in the respective cell. Like mentioned beforehand, the chequerboard structure needs to cover the whole frustum of the sensor. Therefore, several depth maps needed to be considered. The parameter "number" specifies the location of the chequerboard in each frame – 1 meaning that the board is located in the top left corner. If the first row consists of 3 measurements of the board, the second row starts with the number 4. These depth maps were saved in a csv file and got a unique ID per distance and location of the board.

3.2.2 Acquisition with the Lenovo Phab2Pro

Considering the Lenovo Pahab2Pro sensor, we used the Project Tango API by Google. It works similar to the Kinect API, allowing an event-based notification at frame arrival. In contrast to the Kinect API allowing access to the direct streams – like the depth stream – the Project Tango API returns a preprocessed point cloud and access to the depth stream is not possible. Usually, these returned point clouds are aligned automatically in order to fit the motion of the Phab2Pro. When moving around in the scene, this automatic alignment is wanted but not at our data acquisition because it may lead to incorrect measurements. The motion tracking can simply be disabled to remove the alignment.

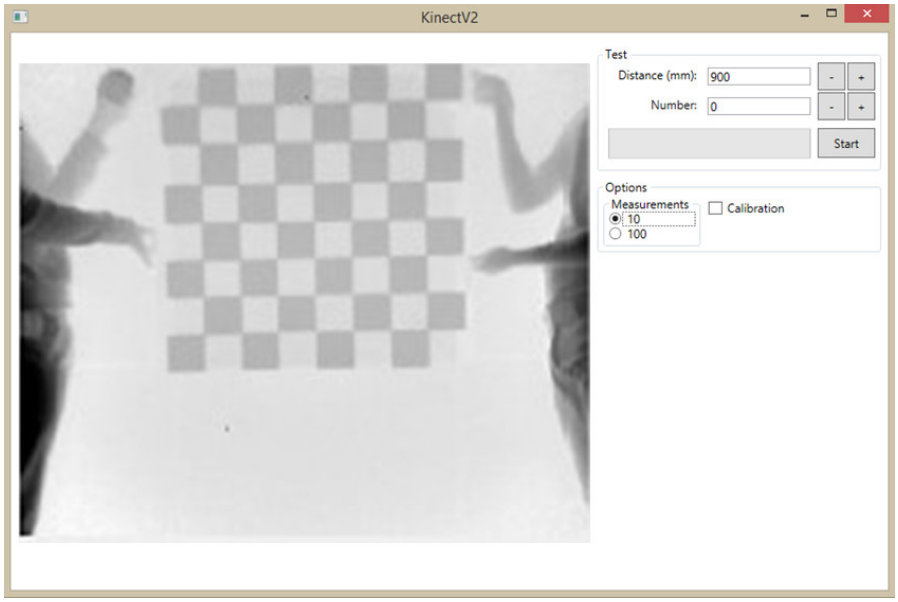

Figure 3.3: Kinect Application for Data Extraction

```
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 d) {
        // 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
```

```
    }
});
```

[Gro17]

We created an Android application in Android Studio using the Java API provided by Project Tango for the data acquisition with the Lenovo Phab2Pro sensor. This application includes the same functionalities as the previous one for the Kinect. The distance and the number can easily be changed using the "+" and "-" buttons. In order to measure time-efficiently, our app takes and stores the measurements asynchronously, distributed over several threads. If all threads have completed the recording, a beep sound is noticeable and if they completed saving the point clouds, the progress bar shows 100%. (see Figure 3.4)

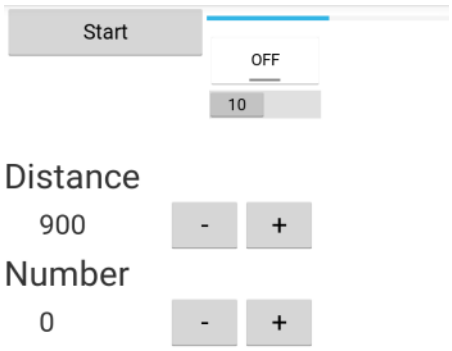

Figure 3.4: Android Application for Data Extraction [Gro17]

3.2.3 Execution of the Data Acquisition

Using our programmes for extracting the data of the KinectV2 and the Lenovo Phab2Pro, we conducted our acquisition. Figure 3.5 shows the basic structure of the setup.

The respective depth sensor was placed onto a table and was aligned orthogonally to the wall. The table was then moved back and forth to the appropriate distances. We started our measurements at a distance of 900 mm and measured every 200 mm up to the final distance of 3100 mm. These distances are based on the work by Choo et al. [CLDB14], oriented at the manufacturer-recommended sensing range. The frustum of the respective sensor needs to be covered by the wall to be able to obtain noise in the whole image region. At each distance we firstly took 10 captures of the plain wall for the calculation of the axial noise. These were taken and saved by our applications and labelled with an ID from 1 to 10. The folder containing these was labelled with the number 0 at each distance. The other folders up from 1 contain the data for the lateral noise in X and Y direction. We attached a grid of masking tape on the wall in order to

Figure 3.5: The basic test setup

allow proper adjusting and to avoid irregularities in the results of the measurements of the chequerboard (see Figure 3.6).

Afterwards, we moved the chequerboard along the grid and took 10 captures in each cell it was shifted to. When measuring with the KinectV2, I held a mouse to start the process because two people were needed to hold the chequerboard. When using the Phab2Pro, a colleague helped us because another person was needed to press the buttons on the smartphone. The folder ID – starting with 1 for the chequerboard measurements – depends on the cells needed to cover the whole frustum. Nearer positions require less cells. The cell where the chequerboard was located top left in each distance got labelled with 1, the next one in that row to the right with 2 and so on. Figure 3.7 shows an example of the labelling of each cell.

In this example, 9 measurements with 10 captures each were needed to cover the whole frustum. At each distance in each of the cells, 10 captures were taken and got stored in the respective folder. A chequerboard cell taken at 900 mm distance at the top left corner got stored in the files 900/1/1-10.csv.

We moved the table back orthogonally to the wall and repeated the same procedure for each distance. The farther away the sensor was located to the wall, the more cells were needed. Afterwards, these csv files were used for the noise extraction of the respective sensor.

Figure 3.6: The actual measurement with the chequerboard in one position of the grid

1	2	3
4	5	6
7	8	9

Figure 3.7: Labelling of the position of the chequerboard in the respective grid. In this example, 3x3 measurements are enough to cover the whole frustum.

3.3 Preprocessing

3.3.1 Kinect Distortion

Due to the lenses of the sensors, straight lines in the captured scene look like curved lines leading to a non-rectilinear projection. Figure 3.8 shows the possible distortions.

Either a negative radial distortion or a positive radial distortion can appear. [TM] The images taken by the KinectV2 sensor show a positive radial distortion – or a so called "barrel" distortion – which looks like the image is mapped around a sphere. To correct that distortion, the Matlab tool "Camera Calibrator" was used. Hereby, several snapshots of a 2D black and white chequerboard structure are needed for the calibrator to calculate lens and camera specific parameters to correct this distortion. Figure 3.9 shows the camera calibrator of Matlab. In the centre, the currently selected image with the detected chequerboard structure is visible and on the right side, the detected error and the camera

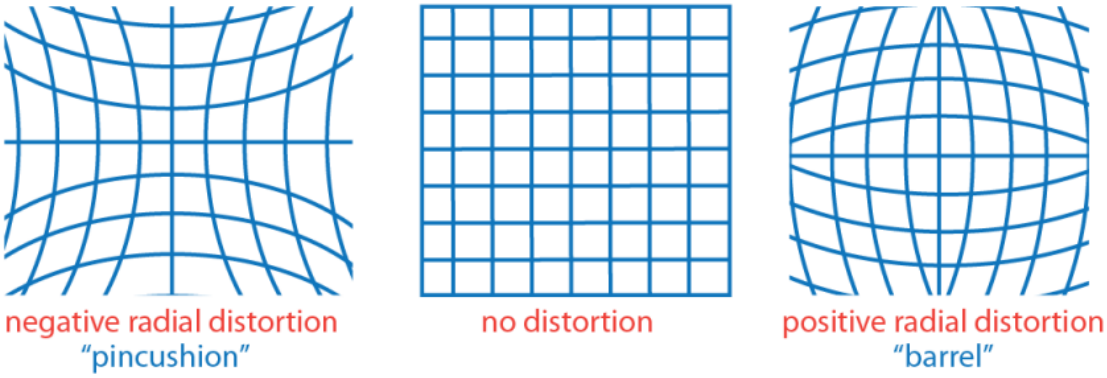

Figure 3.8: Possible lens distortions [TM]

extrinsics.

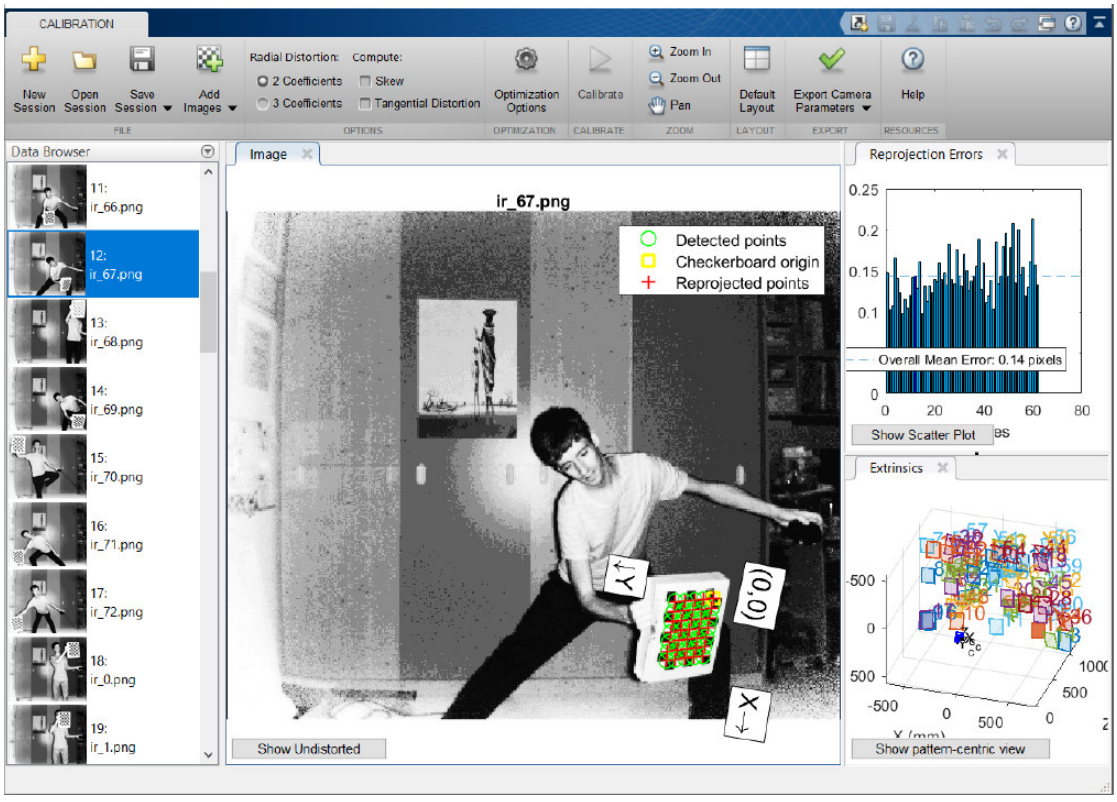

Figure 3.9: Matlab Camera Calibrator

Figure 3.10 shows an image from the calibration before and after the correction. On the left side, the barrel distortion is visible, especially at the left border of the cupboard. In the right image, the barrel distortion got corrected, which is again especially visible at

the same location.

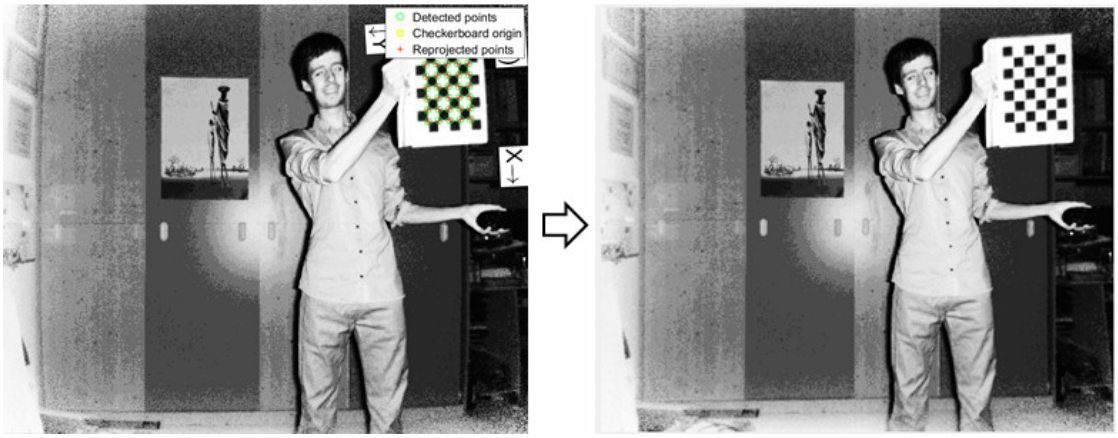

Figure 3.10: Image with and without barrel distortion after the calibration

Figure 3.11 shows a binary image of the chequerboard in the left lower corner of the frustum. The barrel distortion got removed after the calibration, which is displayed in the image on the right.

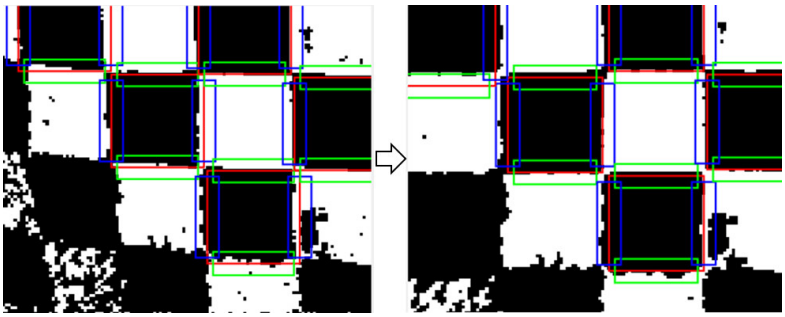

Figure 3.11: Binary Images of the chequerboard before and after calibration

3.3.2 Data Converting

As mentioned above, both point clouds and depth maps are necessary for axial and lateral noise calculation. The KinectV2 sensor returns depth maps, while the Phab2Pro sensor returns point clouds. Therefore, converting needed to be done as a preprocessing step. The Table 3.1 shows the intrinsic camera parameters of the sensor that were needed for the transformations explained beneath.

Depth Map to Point Cloud

The returned depth maps of the KinectV2 sensor needed to be transformed to point clouds allowing proper axial noise extraction. Simply mapping the pixel positions x and

	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 3.1: Camera Intrinsics [Gro17]

y and the depth values z to a point cloud would result in a rectangular cuboid view frustum where objects inside of the frustum are skewed. Distant objects appear smaller, while near object seem to be larger. To avoid this distortion, the intrinsic parameters of the camera needed to be known. We used the resolution of the depth image and the field of view in our transformation that got published by the manufacturer. Applying the formulas beneath allows the calculation of the respective world coordinates and avoids the distortions mentioned above. After converting all data points of the depth map, a correctly scaled point cloud allowing axial noise extraction is returned.

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

[Gro17]

Point Cloud to Depth Map

When transforming points clouds into depth maps, the inverse procedure has to be applied. The point clouds returned by the Phab2Pro sensor need to be transformed to two dimensional depth maps. During this transformation, objects nearer to the sensor need to be increased in size while farther away objects need to be decreased in order to achieve the inverse effect to the transformation before. Again, the intrinsic camera parameters are needed. The formulas above can simply be inverted to grant a transformation from real-world coordinates to pixel coordinates.

$$x_{[px]} = \frac{x_{[mm]}}{z * \rho_x} + \frac{width_{[px]}}{2}$$
$$y_{[px]} = \frac{y_{[mm]}}{z * \rho_y} + \frac{height_{[px]}}{2}$$

[Gro17]

3.4 Axial Noise Calculation

The axial noise is the noise in the direction of the Z axis. To achieve the extraction, point clouds were used. In our setup, the point clouds showing only the wall, covering the whole frustum were used. These point clouds were extracted at the distances from 900 mm to 3100 mm each 200 mm – captured 10 times. For the actual calculation of the axial noise, a plane got fitted through each of the point clouds. This plane was used as a ground truth of the wall. Afterwards, the deviation from the distance of the plane to the distance of the respecting points of the point clouds was calculated for each of the 10 captures. Considering these 10 point clouds, the standard deviation of each of these 10 captures was calculated for each pixel coordinate (X, Y). The standard deviation for each pixel is the axial noise in Z direction.

Figure 3.12 shows a point cloud of the KinectV2 sensor where the sensor was placed at a distance of 900 mm away from the wall. The black surface is the fitted plane – the ground truth of the wall.

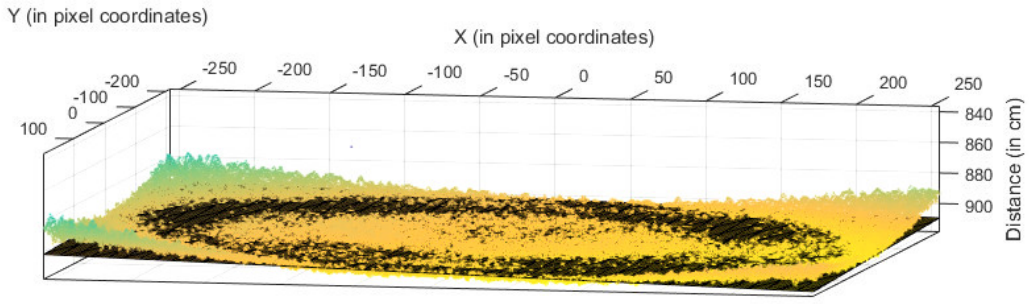

Figure 3.12: Point cloud of the KinectV2 at 900 mm distance with the fitted plane

3.4.1 Ground Truth Estimation using Plane Fitting

Our implementation used the Matlab function `fit` for the plane fitting estimating the ground truth of the wall. This function fits a plane trough the point cloud with minimal distances over all points. The result of the `fit` function is an equation that calculates Z dependent on the X and Y values.

$$z = a * x + b * y + c \tag{3.1}$$

The function above is a linear function in dependence of the variables X and Y. The Matlab function `fit` calculates the best coefficients for the variables a, b and c. This fitted plane was used as a ground truth for the noise calculation.

3.4.2 Noise Calculation

For calculating the axial noise, we used the Matlab function `feval`. This function used the fitted plane that was extracted before. For each data point with X and Y pixel coordinates, the distance from the plane was calculated (see Figure 3.13), resulting in a distance value for each pixel coordinate (X, Y) for each distance for each of the 10 captures. The axial noise was the calculated standard deviation of all the distance values.

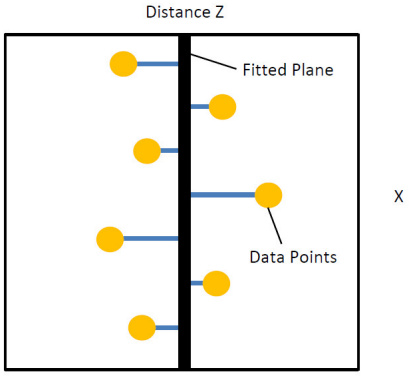

Figure 3.13: Data Points (yellow), Fitted Plane (black) and distance to each other (blue)

3.5 Lateral Noise Calculation

Like mentioned beforehand, our test setup also calculated the lateral noise in both X and Y direction. To achieve this, a 3D chequerboard was used. The chequerboard was measured, covering the whole frustum of the sensor at the respective distance. The board had to be moved across the wall and several captures had to be taken in each cell of the grid. Noise was extracted at the edges of each chequerboard field.

3.5.1 Thresholding and Area Selection

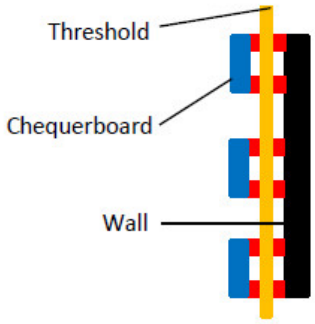

Figure 3.14: Explanatory Figure of the Depth Threshold

Firstly, the depth maps needed to be transformed into a binary image to allow proper noise calculation along the edges. Figure 3.15 shows the obtained depth maps at a distance of 1100 mm normalized between 800 mm and 1200 mm. All of these depth maps cover the whole frustum of the sensor. In order to transform the depth maps into binary images, a threshold needed to be selected leading through the LEGO pillars (see Figure). Using this threshold, objects closer to the camera – especially the blue carton on top of the LEGO pillars – were assigned 0 and all values behind – like the wall – 1. This threshold needed to be slightly adjusted for each distance and capture. Figure 3.15 shows the resulting binary images after thresholding at a distance of 1100 mm.

In order to improve the automatic detection of the rectangles of the chequerboard, a preprocessing step needed to be done because the structures of my colleague's and my body were disturbing the detection, leading to false positives. Therefore, a manual rectangle selection via a Matlab script was added. For each image, a rectangle was selected covering the position of the chequerboard in the respective image.

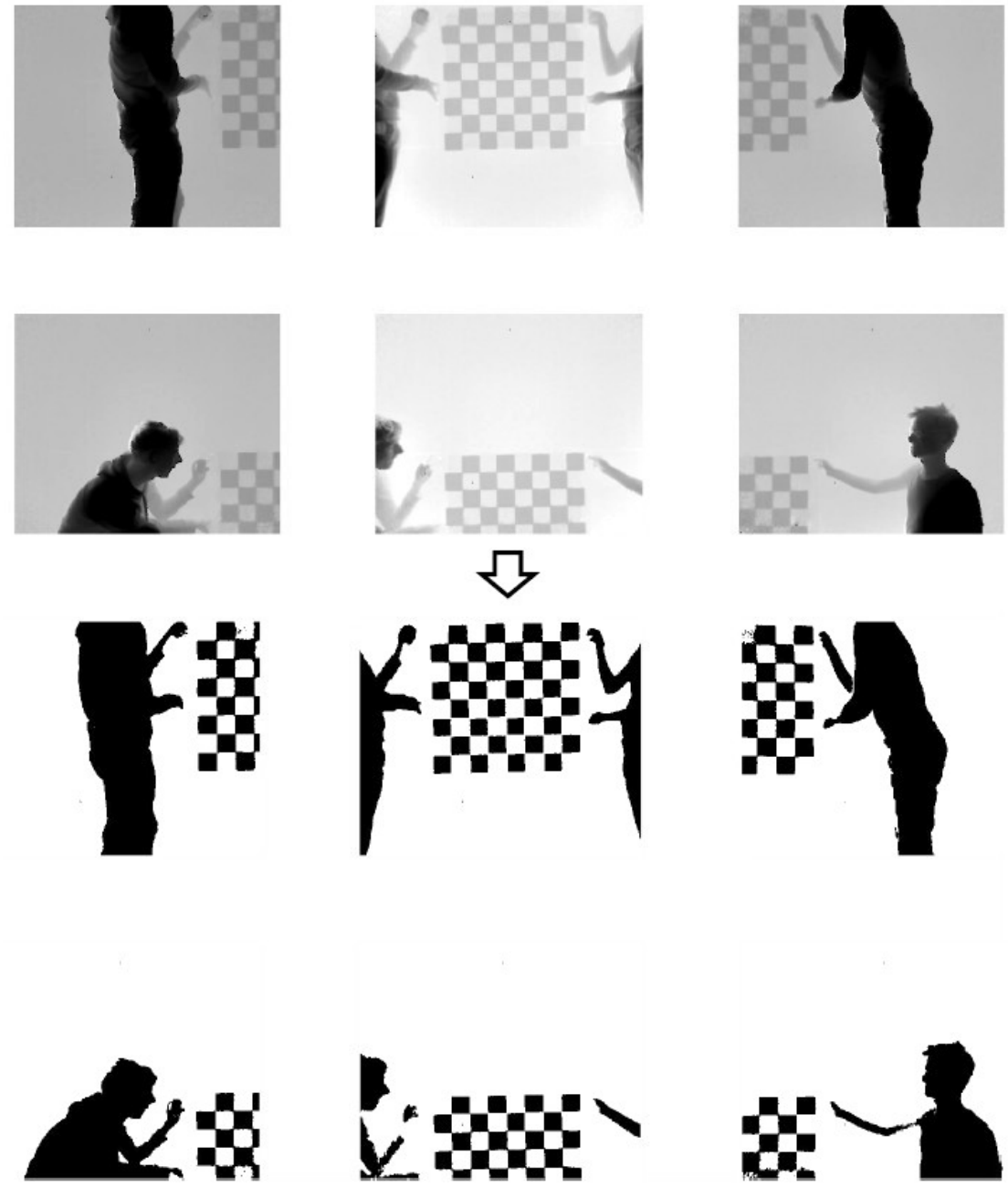

Figure 3.15: Depth Maps of the KinectV2, Distance = 1100 mm, Binary Images after Thresholding is beneath

3.5.2 Square Detection

In order to detect the squares for the lateral noise calculation, we used the previously selected rectangular areas of the chequerboard and applied a Scale-Invariant Blob Detection in form of a Laplacian blob detector. These detected blobs could be transformed into the desired squares.

The Laplacian blob detector performs two steps when searching for blobs: a spatial selection of interest points and an assignment of a scale. The Laplacian operater is a differential operator of second order. [Sze10] It will achieve maximum response at the centre of the blobs. The scale of the Laplacian is then "matched" to the scale of the blob. Figure 3.16 shows an example of a signal with the radius of 8. The algorithm now iterates over different scales starting with 1 multiplied by 2 each step (σ). At $\sigma = 8$, the radius of the original signal matches the current scale and a maximum negative response is achieved. The blob got detected with the respective scale. The centre of the detected blob shows the position of the maximum response.

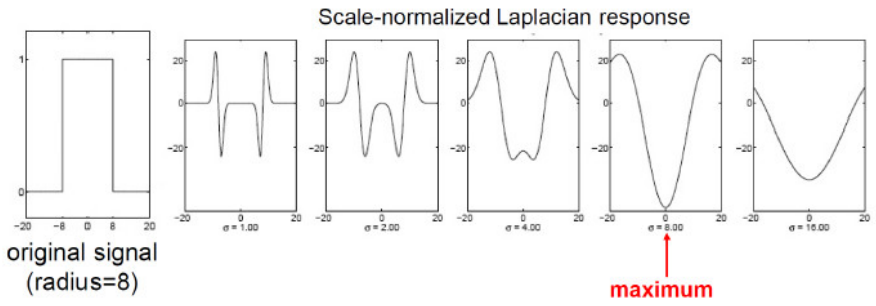

Figure 3.16: Blob Detection with the Laplacian Operator [Sze10]

The algorithm works as follows [Sze10]: It starts with an initial scale (σ). This scale is multiplied by a constant factor k each round. In each iteration, a scale normalized Laplacian of Gaussian filter is built with the current scale σ. The image is now convolved with the respective Laplacian of Gaussian filter and the absolute responses are saved for each level. This procedure leads to a 3D scale space. The maximum responses are searched in this 3D scale space. A response has to be maximal in the 3x3 neighbourhood on the same level and in the respecting neighbourhood one level below and one level above. These found maxima represent blobs with the location of their centres and the scale σ.

Our implementation started with the selected rectangular areas of the chequerboard in the respective images. The initial scale had to be found manually. The selected part of the depth map got convolved with the Laplacian of Gaussian kernel at different levels. Figure 3.17 shows an example of 4 different levels of the chequerboard.

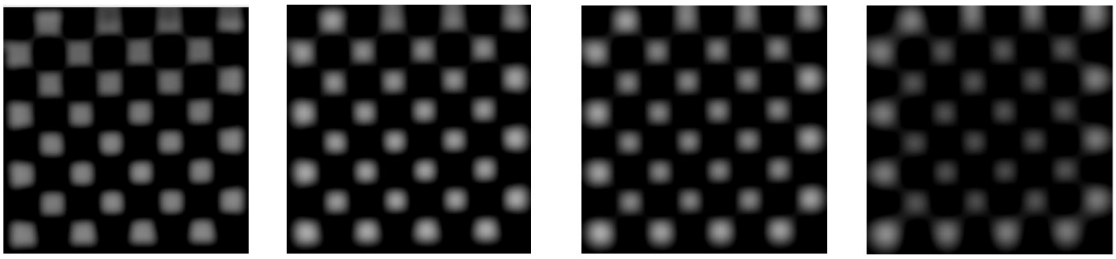

Figure 3.17: Responses of the Laplacian of Gaussian at different levels

Dark blobs in the image correspond to strong positive responses while bright blobs correspond to strong negative responses in the Laplacian of Gaussian scale space. Because we only needed black rectangles for the noise extraction, we only considered maximum positive responses. Most of the blobs in Figure 3.17 were found at the scale space on the second left. Hereby, the white areas seem to be having specular highlights, meaning that the response is very high in the centres of them. Comparing a white area of the second subfigure with the first and third shows that the maxima in the 3x3x3 neighbourhood are likely to be contained in the second subfigure. Most of the blobs got detected there.

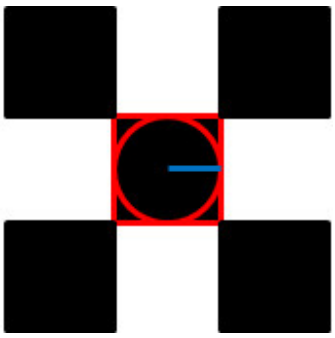

Figure 3.18: Transformation of the detected blobs to squares

The scale (σ) – the level where the blobs were found – represents the radius of the blob, which was considered half the width of the square. The left upper corner of the squares was calculated via subtraction of the radius from the centre in x and y direction. From this left upper point, a square with a side length of 2*radius was constructed to represent a detected square of the chequerboard. Figure 3.19 shows the detected squares at a distance of 1100 mm of the respective location of the chequerboard. These squares were used for further calculations.

3.5.3 Region of Interest Selection for X and Y Noise

The previously detected squares were used to select the Regions of Interest where the lateral noise can be calculated for the X and Y direction. Lateral noise in X direction was

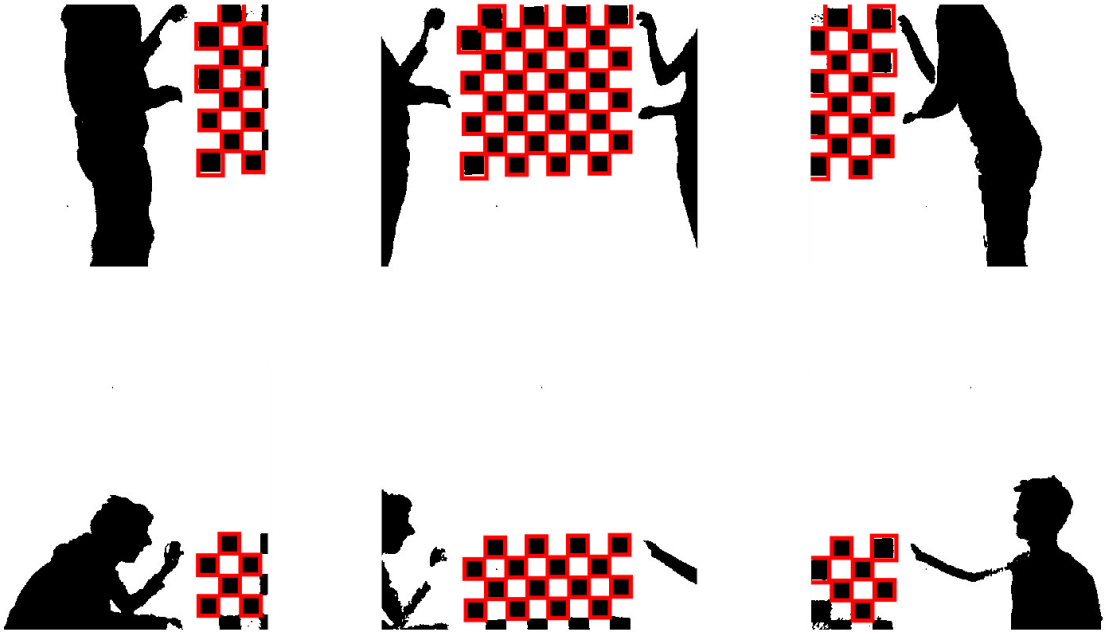

Figure 3.19: Detected Squares after the Laplacian Operator at a distance of 1100 mm

measured at the vertical edges of the detected squares and lateral noise in Y direction at the horizontal ones. Along these edges, rectangles were constructed with an extended width or height in both directions orthogonal to the detected edges, surrounding the edges. Figure 3.20 shows the constructed rectangles along the edges. They covered a part of the black coloured chequerboard's square and a part outside in the white background region. Blue rectangles mark the area for X noise while green rectangles mark the area for the Y noise calculation. Blue rectangles had been decreased in height and green rectangles in width to minimize overflowing to other parts of the image. Along the edges in the blue and green rectangles, a standard deviation was calculated and noise extracted.

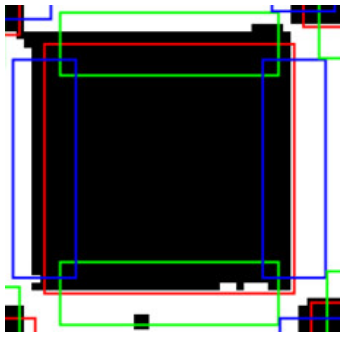

Figure 3.20: Detected Square with Rectangles for X and Y Lateral Noise Calculation

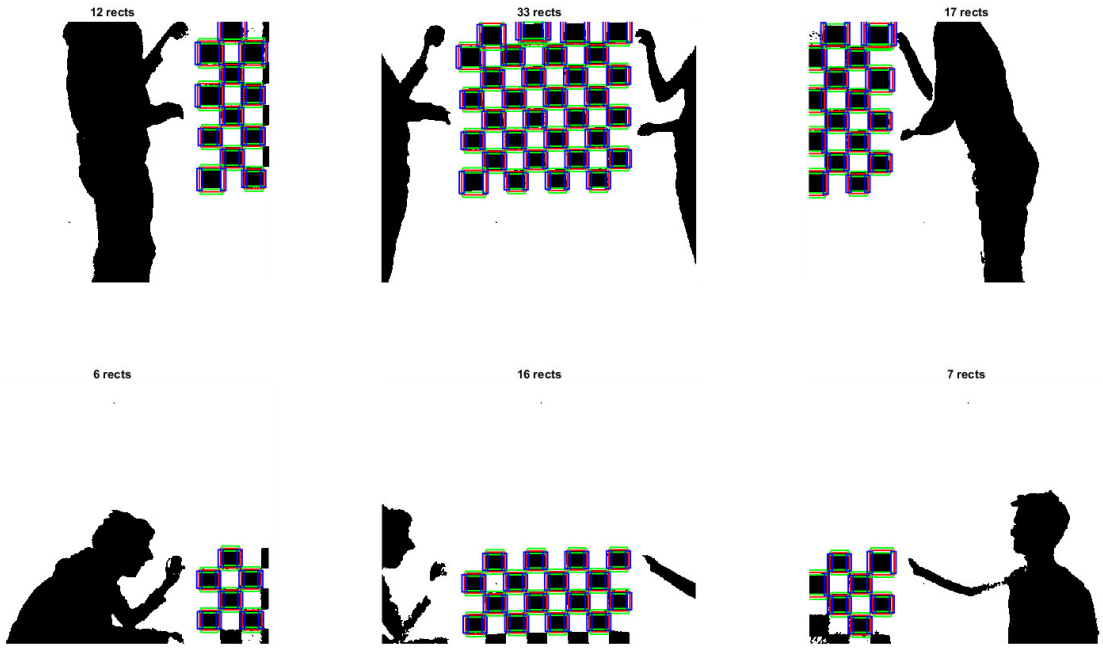

Figure 3.21: Detected Squares with Rectangles for X and Y Lateral Noise Calculation

3.5.4 Noise Calculation for X and Y Noise

Noise could be calculated in the before mentioned rectangles along the edges of the detected areas. The basic idea was to reduce the selected area to a white line with a width of one pixel. Using this row, the standard deviation was calculated and the noise in X and Y direction extracted.

Before the actual reduction, the rectangles needed to be transformed to an equal layout. The basic layout followed the structure of the green rectangle at the top of the squares. The matrix of the green rectangle on the bottom simply needed to be flipped, the matrix of the blue rectangle on the right needed to be transposed and the blue rectangle on the left needed to be flipped and transposed.

The next step was the above-mentioned reduction to a 1 px line of white pixels (value = 1) for the calculation of the standard deviation. This algorithm works after following

scheme:

Algorithm 3.1: Reduction to a 1 px row

Data: Binary values of a rectangular area

Result: Reduced 1 px line

```
1  start at first column;
2  while currentColumn != lastColumn do
3  |  traverse from top to bottom;
4  |  if Value from current position == 1 then
5  |  |  if Value[row + 1] == 1 then
6  |  |  |  set value at current position = 0;
7  |  |  end
8  |  |  if Value[row + 1] == 0 and Value[row + 2] == 1 then
9  |  |  |  set value at current position = 0;
10 |  |  end
11 |  end
12 |  switch to next column;
13 end
```

The function traverses each column of the detected rectangles around the edges. If the value in the next row is 1 and the current value is 1, then the current value gets assigned 0. This removes the white areas above the black ones and leaves a 1 px – white line. The second `if` is necessary for eliminating black interfering pixels. Figure 3.22 shows an example of the reduction algorithm.

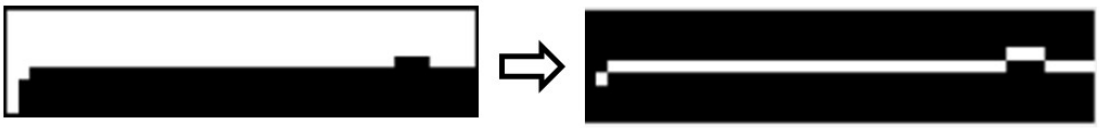

Figure 3.22: Reduction of the binary values to a 1 px line

After the reduction, the standard deviation was calculated using the Matlab function `std` over the positions of the white pixels. The returned standard deviation got multiplied by the distance and a constant, sensor specific factor.

$$\sigma_{[mm]} = \sigma_{[pixel]} * \text{distance} * 0.0028...\text{KinectV2} \tag{3.2}$$

$$\sigma_{[mm]} = \sigma_{[pixel]} * \text{distance} * 0.0057...\text{Phab2Pro} \tag{3.3}$$

The returned noise is the actual lateral noise in either X or Y direction, depending on the selected rectangle. The noise corresponds to the real-world noise in mm. This noise got saved with the current X and Y coordinates of the centre of the selected rectangle.

CHAPTER 4

Noise Models and Evaluation

During our noise model calculation, we started with fitting the data and added three improvements afterwards in order to improve the accuracy of the resulting noise model. This was done for the results of the test setup measurements in X, Y and Z direction separately. We improved our results by using RANSAC [FB81], a splitting into different regions and a cubic function, which will be explained in the sections below. The final noise model for the KinectV2 and the Phab2Pro consists of Improvement 3 4.4 for the axial noise and Improvement 2 4.3 for the lateral noise, leading to the most accurate coverage of the measured data while achieving the best results at our evaluation.

4.1 Initial Noise Models

The before-measured noise for the X, Y and Z direction was used as the input for the fitting of the formula $\sigma_{x,y,z}$ (Equation 4.1). The initial noise model calculation considered all data points from the measurements and tried to fit a model through the whole dataset.

We used the deviation of the noise model from the measured data as a quality measure (RMSE). Using the initial method, the KinectV2 model resulted in a 2.8923 mm RMSE in X direction, a 2.8496 mm RMSE in Y direction and a 1.7698 mm RMSE in Z direction from the measured data. These high deviations result from gross outliers that disturb the fitting process.

The X and Y noises of the Phab2Pro could not be modelled appropriately using the initial measurements due to chaotic noise and especially high noise at the borders. At distances farther away from the sensor – up from 2500 mm – the detection of the rectangles of the chequerboard was difficult due to the lower resolution and quality of the point clouds.

4.2 Improvement 1: Noise Models with RANSAC

To achieve better results we needed to get rid of interfering values – so called gross outliers. We used RANSAC [FB81]. Figure 4.1 shows the resulting noise models after using the RANSAC improvement. For the Z direction, the RMSE for the KinectV2 measurements is 0.687 mm, while for the Phab2Pro it is 1.1903 mm, leading to more appropriate results when fitting the noise model according to the measured data.

The X and Y noise models for the KinectV2 show an RMSE of 2.1 mm and 2.0 mm, which is still too high to be used as a qualitative noise model.

The X and Y noise models for the Phab2Pro are still to chaotic to be modelled appropriately. The RMSE for the X and Y noise is about 7 mm, which is definitely too high to result in a specific noise model.

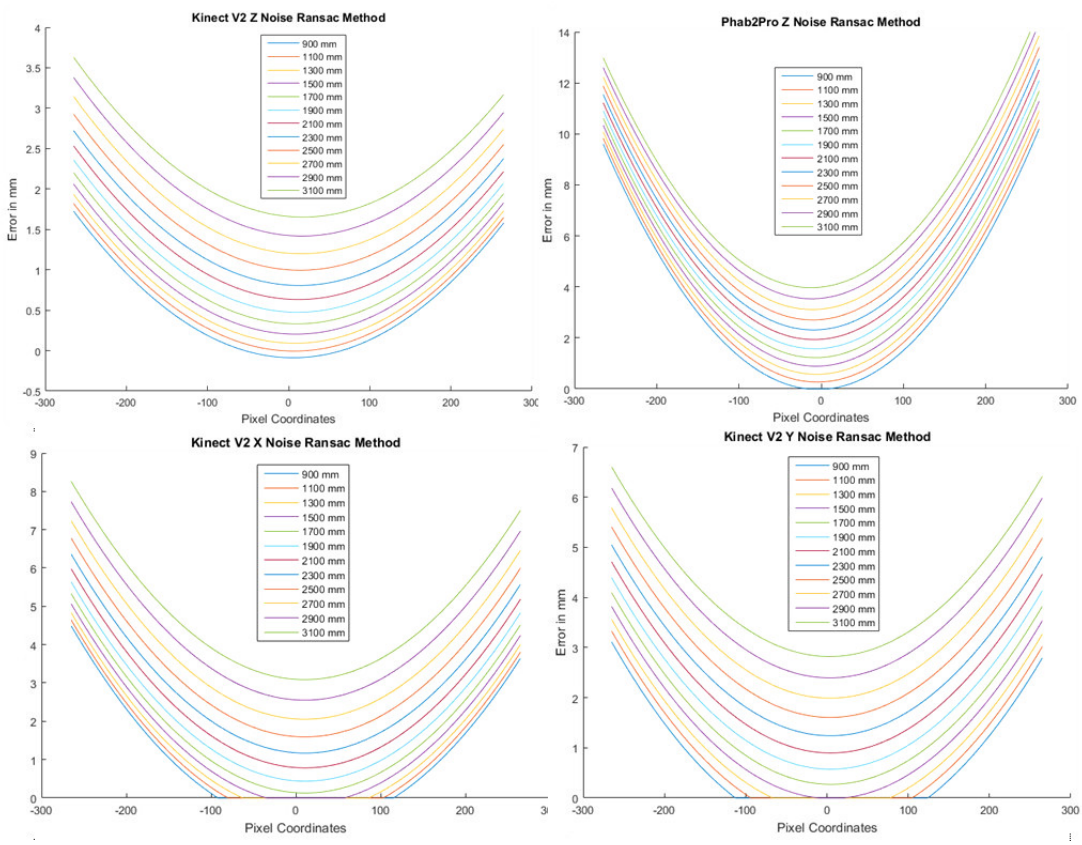

Figure 4.1: Noise Models using the RANSAC Improvement – X and Y noise for the Phab2Pro is missing – could not be modelled appropriately – RMSE of 7 mm

4.3 Improvement 2: Noise Models with Regions and RANSAC

Like Choo et al. [CLDB14] suggested, we used a splitting into regions in order to lower the RMSE. The noises of the X, Y and Z direction were handled independently. The previously calculated noise over the whole frustum was used as input data for this improvement. This area got split up into 8 rows and 8 columns (like Choo et al. [CLDB14] suggested). In each of these cells, a mean value of the deviations was calculated. This average noise was then taken as a representative value for the respective cell. Figure 4.2 shows an example of the region splitting for the KinectV2 sensor in Z direction (axial noise), starting at a distance of 900 mm at the top left and ending at 3100 mm at the bottom right. The colouring emphasizes the average noise in the respective area. In the dark blue cells, the average error is near 0 mm and at the yellow areas about 12 mm. The farther away the measurements were conducted, the higher the average error got in these areas. Especially at the values measured above 2500 mm, the high noise produced by the depth sensor in the corners is visible.

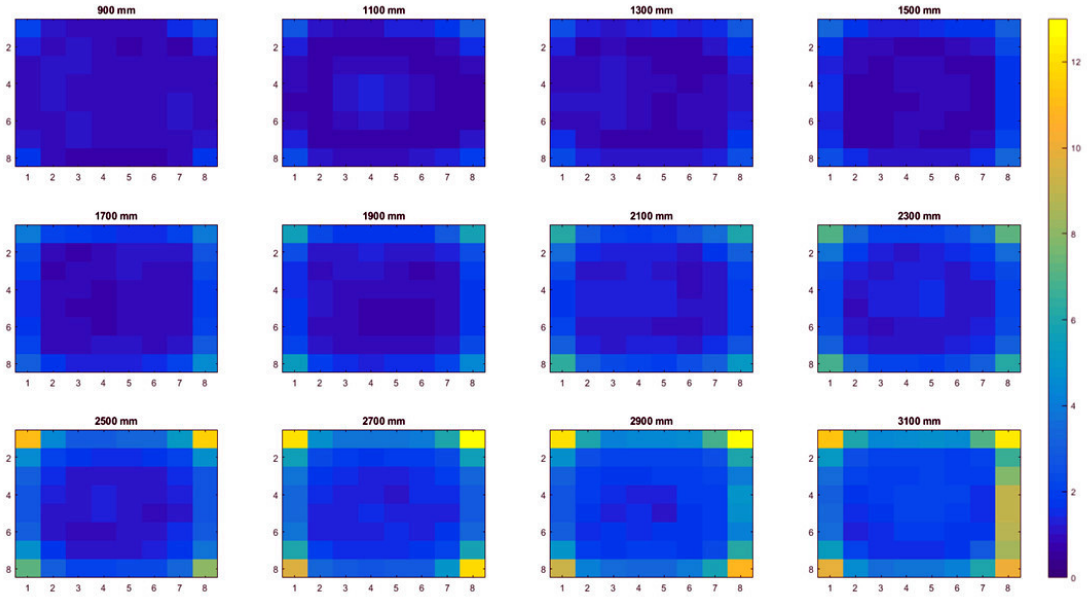

Figure 4.2: 8 x 8 regions of Z Noise in mm

These average noise values for each region/cell are input values for RANSAC [FB81]. The noise model is fitted in the first step. RANSAC [FB81] was then executed 1000 times to remove gross outliers. Considering the KinectV2 in Z direction, 3.77% of the data values got classified as gross outliers, leading to an average estimation error of 0.6131 mm – meaning that the noise model in average deviates with 0.6131 mm – which was another improvement compared to the initial fitting and the fitting using only RANSAC

[FB81]. Figure 4.3 shows the resulting noise model for the KinectV2 in Z direction at the bottom left. The growth of the noise, the farther away the measurements were conducted, is especially visible in the figure as well as the much higher noise at the borders of the frustum. The respective noise models for the X and Y noise for the KinectV2 can be seen in Figure 4.3 at the top.

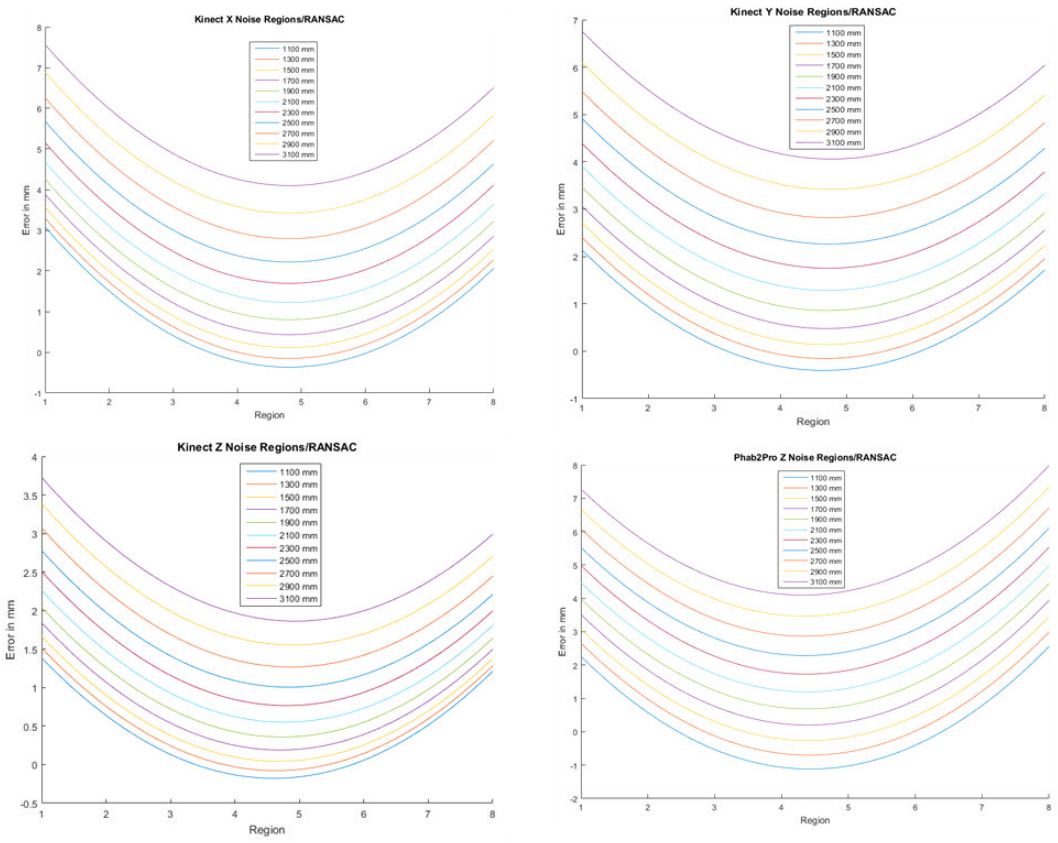

Figure 4.3: Noise Models X, Y, Z for the KinectV2, Z for the Phab2Pro with Regions/RANSAC – X and Y noise for the Phab2Pro is missing – could not be modelled appropriately

$$\sigma(x,y,z) = \alpha_{[1]}x^2 + \alpha_{[2]}y^2 + \alpha_{[3]}z^2 + \alpha_{[4]}xy + \alpha_{[5]}yz + \alpha_{[6]}xz + \alpha_{[7]}x + \alpha_{[8]}y + \alpha_{[9]}z + \alpha_{[10]} \tag{4.1}$$

Table 4.1 shows the coefficients needed for the noise model formula for the KinectV2 and the Phab2Pro sensor. The splitting into regions works well for the KinectV2 in general. RANSAC [FB81] removed only from 3 to 5% gross outliers and improved the result of the noise model. The average deviation of the noise model to the measured data is 0.61 mm

	KinectV2 X	KinectV2 Y	KinectV2 Z	Phab2Pro Z
$\alpha[1]$	0.2379	0.1901	0.1179	0.2877
$\alpha[2]$	0.1578	0.1795	0.1160	0.2317
$\alpha[3]$	$6.4360 * 10^{-7}$	$5.4320 * 10^{-7}$	$2.8563 * 10^{-7}$	$2.9479 * 10^{-7}$
$\alpha[4]$	0.0271	0.0269	0.0050	0.0123
$\alpha[5]$	$9.7023 * 10^{-5}$	$3.1201 * 10^{-5}$	$2.1864 * 10^{-5}$	$1.6039 * 10^{-4}$
$\alpha[6]$	$-3.1194 * 10^{-6}$	$-2.1397 * 10^{-5}$	$-3.6410 * 10^{-5}$	$2.9375 * 10^{-5}$
$\alpha[7]$	-2.3915	-1.8550	-1.0682	-2.6284
$\alpha[8]$	-1.7107	-1.7529	-1.0805	-2.3220
$\alpha[9]$	$-8.4266 * 10^{-4}$	$-7.2166 * 10^{-5}$	$-1.0438 * 10^{-4}$	$5.9948 * 10^{-4}$
$\alpha[10]$	9.1643	7.1424	4.4954	8.3749
$RMSE(mm)$	1.1410	1.0950	0.6131	1.2362
$Rem.Outliers(\%)$	5.338	4.9479	3.0601	6.1198

Table 4.1: Results of the Noise Model (Equation 4.1) using the Regions/RANSAC Approach

for the Z direction and about 1 mm for the X and Y direction (RMSE), leading to the most accurate results compared with the initial noise model and the first improvement using only RANSAC [FB81]. The splitting method can also be applied to the Phab2Pro measurements in Z direction, leading to an RMSE of 1.2 mm. Using the splitting into 8 x 8 regions for the X and Y noise of the Phab2Pro led to many NaN values due to the smaller resolution and the higher noise in the images, therefore, accurate results could not be achieved. Splitting the data into fewer than 8 x 8 regions did not represent the measured data well. For the Phab2Pro, only the Z noise region splitting is appropriate, leading to better results in describing the Z noise. Figure 4.3 shows the noise model for the Z noise for the Phab2Pro using the Regions/RANSAC approach on the bottom right.

Fankhauser et al. [FBR+15] were not able to extract a specific noise model for the lateral noise in their test setup and used a 0.9 percentile overestimation instead. The thesis of my colleague [Gro17] used the same approach when characterizing the lateral noise. Due to consistency, we decided to use the quantile based approach in this thesis, too. Only the lateral noise of the KinectV2 could be described by a noise model in this thesis. Therefore, we considered each slice of measurements – like seen in Figure 4.2 – and computed the 0.9 percentile. Due to the lower resolution and the higher artefacts of the Phab2Pro captures, we removed the outer regions at the border for the calculation of the percentiles because of the low detection rate at the borders, especially when using the Phab2Pro. Following formula was used for the fitting, taking the distance as an input value and calculating the X and Y noise accordingly. Table 4.2 shows the resulting coefficients for the formula for the X and Y noise of the KinectV2 and the Phab2Pro.

$$\sigma(z) = \alpha_{[1]}z^3 + \alpha_{[2]}z^2 + \alpha_{[3]}z + \alpha_{[4]} \tag{4.2}$$

	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
$RMSE(mm)$	0.1465	0.0959	0.3031	0.5689

Table 4.2: Results of the X and Y Noise Models (Equation 4.2) using the Regions Approach with Quantiles

4.4 Improvement 3: Noise Models with Regions and RANSAC – Cubic Function

While modelling the axial noise, we decided to use a cubic function allowing a better coverage of the measured noise. For lower distances, the second improvement shows negative values leading to our decision to use a cubic function. We also wanted to achieve a lower RMSE for the axial noise. The KinectV2 Z noise model using a cubic function achieved an RMSE of 0.4852 mm and the Phab2Pro 1.0668 mm, even enhancing the results compared to the second improvement. Using a cubic function also led to the best results at our evaluation. Figure 4.5 shows the results of the noise models for the KinectV2 and the Phab2Pro, using a cubic function for the axial noise in Z direction. Table 4.3 shows the 30 coefficients for the functions for both depth sensors.

$$\begin{aligned}\sigma(x,y,z) =& \alpha_{[1]} + \alpha_{[2]}z^1 + \alpha_{[3]}z^2 + \alpha_{[4]}y^1 + \alpha_{[5]}y^1z^1 + \\ & \alpha_{[6]}y^1z^2 + \alpha_{[7]}y^2 + \alpha_{[8]}y^2z^1 + \alpha_{[9]}y^2z^2 + \alpha_{[10]}x^1 + \\ & \alpha_{[11]}x^1z^1 + \alpha_{[12]}x^1z^2 + \alpha_{[13]}x^1y^1 + \alpha_{[14]}x^1y^1z^1 + \alpha_{[15]}x^1y^1z^2 + \\ & \alpha_{[16]}x^1y^2 + \alpha_{[17]}x^1y^2z^1 + \alpha_{[18]}x^1y^2z^2 + \alpha_{[19]}x^2 + \alpha_{[20]}x^2z^1 + \\ & \alpha_{[21]}x^2z^2 + \alpha_{[22]}x^2y^1 + \alpha_{[23]}x^2y^1z^1 + \alpha_{[24]}x^2y^1z^2 + \alpha_{[25]}x^2y^2 + \\ & \alpha_{[26]}x^2y^2z^1 + \alpha_{[27]}x^2y^2z^2 + \alpha_{[28]}x^3 + \alpha_{[29]}y^3 + \alpha_{[30]}z^3\end{aligned} \tag{4.3}$$

4.5 Combined Noise Model

The two noise models were combined into a single noise model describing the axial and lateral noise for both the KinectV2 and the Phab2Pro, taking the X and Y pixel coordinates, the distance and the rotation into account. For the axial noise calculation, we decided to use a weight function for the combination of our two noise models. The model described in this thesis does not take rotation into account, therefore, the more rotated the surface is, the less influence does the noise model of this thesis have. The model of my colleague [Gro17] is entitled Model 1 and the model of this thesis Model 2 in the following section.

	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
$RMSE(mm)$	0.4852	1.0668

Table 4.3: Resulting coefficients for the Noise Model (Equation 4.3) using the Regions/Ransac Approach – Cubic Function

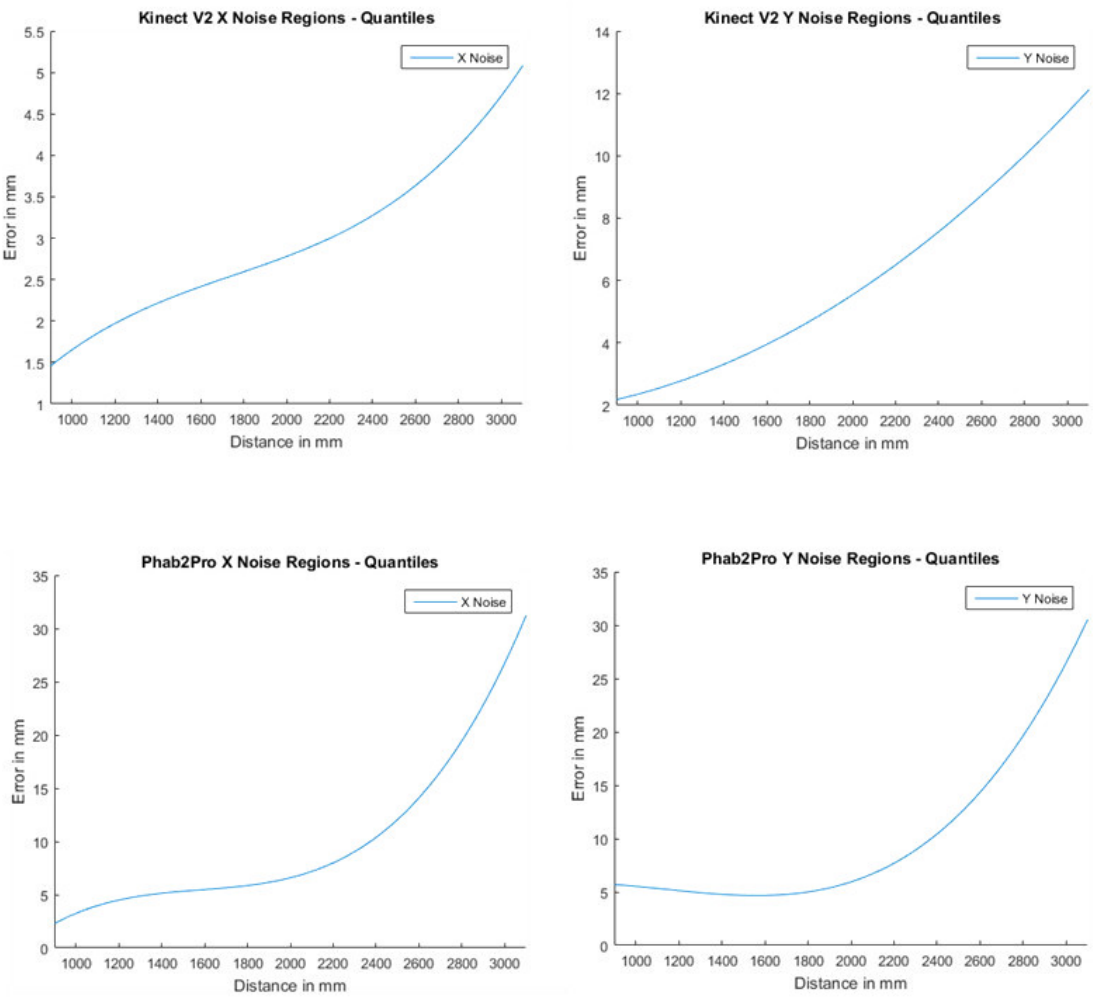

Figure 4.4: Noise Models using Regions/Quantiles

Due to consistency, we used the 0.9 percentile for both depth sensors for the two noise models, overestimating the lateral noise so that 90% of the measured noise should be below the estimated lateral noise for the X and Y direction from our combined noise model. This procedure of taking the 0.9 percentile is oriented at the work by Fankhauser et al. [FBR+15].

$$\sigma(x, y, z) = CombinedModel(X, Y, Z, R)$$

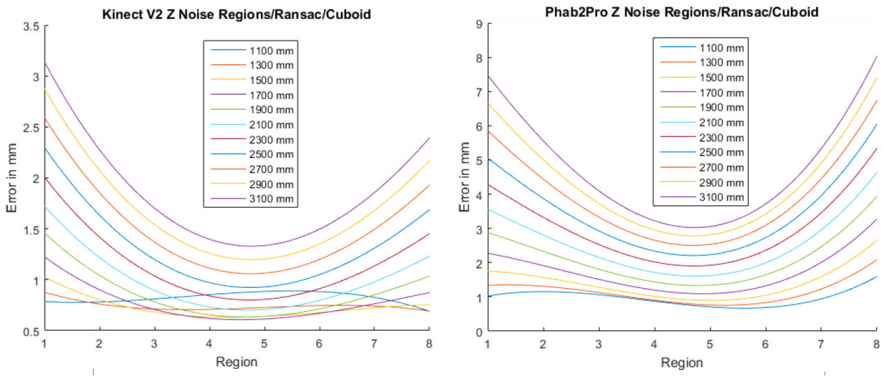

Figure 4.5: Noise Models for the Z Direction using a Cubic Function

4.6 Evaluation

4.6.1 Evaluation Setup

The combined noise model was evaluated through a real-world scenario using a solid pressboard cube measured with the KinectV2 and the Phab2Pro sensors. The cube size is 300 x 300 x 300 mm. The cube got placed at different positions covering the whole frustum in a 3 x 3 grid. At each of these positions, the distance and the rotation of the cube was altered. At a distance of approximately 1 m and 2 m, measurements were conducted, representing a near and a far distance. A 0 and 45 degree rotation was used in order to measure the cube, showing one and two faces. Through tilting the depth sensors, depth images highlighting 3 faces were measured. The noise calculation of the X, Y and Z direction is similar to the described method, but only 1 image was used per distance/rotation/position. These measured noises got compared to the combined noise model afterwards. The rotation and distance could be extracted manually due to the simplicity of the evaluation setup.

The **axial noise** was measured at the captured surfaces of the cube. At different orientations either 1, 2 or 3 faces were visible that were selected manually during the evaluation. In the next step, a plane was fitted through the captured data points and the noise was extracted – similar to the test setup – by calculating the standard deviation from the plane to the data points. The average distance value of each plane was taken as the Z parameter for the evaluation. Considering the rotation, the normal vectors between the plane and the camera were used, allowing the calculation of the angle.

The **lateral noise** was measured at the borders of the cube similar to the extraction process in the test setup described in this thesis. Due to the evaluation setup using a rotated cube, the Y noise had to be extracted with an addition because of the tilted borders in the captures. Figure 4.6 shows an example. The red square in the left image

shows an explanatory area for the Y noise extraction of the cube. In this area the reduction algorithm – explained in 3.1 – was used, leading to a line with a width of 1 px, as seen on the right side of Figure 4.6. In contrast to the test setup, this line is tilted. Simply calculating the standard deviation of the white pixels did not lead to the correct result. Therefore, we fitted a line (red) through the diagonal line and calculated the standard deviation from that line. The borders of the square were selected manually during the evaluation process. The X and Y pixel coordinates of the centres of the red squares and the distance of the cube were used for the evaluation.

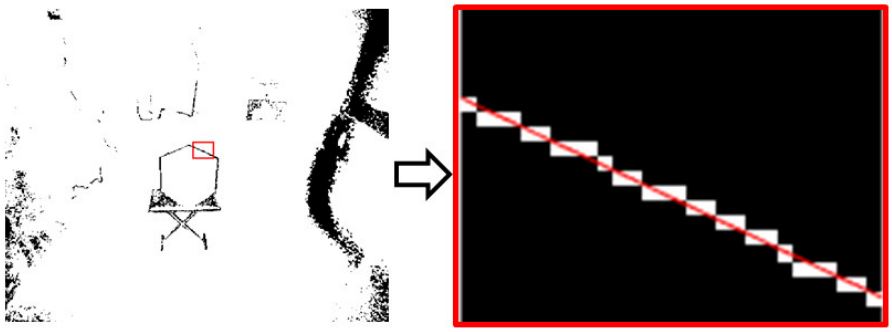

Figure 4.6: Y Noise Extraction during the Evaluation

4.6.2 Results

We decided to use the RMSE as a quality measure showing the deviation from the measured noise of the evaluation setup compared to the estimation of our combined noise mode. Because we used the 0.9 percentile, overestimating the lateral noise component, we checked, how many of the measured lateral noises of the evaluation setup were under the specified threshold of the combined noise model.

Figure 4.8 shows the results of the axial evaluation on the left side of the KinectV2 and on the right side of the Phab2Pro. Each point in this scatter-plot represents a measured surface of the cube at the rotation given by the X axis and the distance given by the Y axis. Each point was coloured in a divergent colour scheme showing colours from red to blue. The bars on the right side of the scatter-plots show the different deviations from the estimated model, highlighted by a colour. If a point in the scatter-plot is white, the RMSE is between -0.7 mm and 0.7 mm. The red colour shows a higher positive deviation and the blue colour shows a higher negative deviation.

Model 2 generally underestimated the axial noise when the object got rotated because rotation was not measured during this test setup. Table 4.4 shows the RMSE values of the different models. The general underestimation of Model 2 is visible in the scatter-plots of Figure 4.8. The RMSE values of Model 2 are, therefore, generally higher than Model 1. Simply averaging the two models lead to worse results than taking only Model 2.

	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 4.4: RMSE for the Axial Evaluation [Gro17]

	KinectV2	Phab2Pro
X – Combined (averaged)	94.4	93.4
Y – Combined (averaged)	94.4	94.8

Table 4.5: Resulting percentages of our estimated 0.9 percentiles

Therefore, we decided to introduce a weight function to our combined noise model. The more rotated the object is, the less influence does Model 2 have. In case of the KinectV2, the resulting combined noise model shows a slightly smaller RMSE, while the RMSE of the combined model for the Phab2Pro shows a significantly smaller error.

Figure 4.9 shows the evaluation of the lateral components X and Y, while Table 4.5 shows the percentage values of how many values at the borders of the cube were under our estimated threshold. As mentioned above, we used the 0.9 percentile for the lateral noises. The circles in the scatter-plots 4.9 show the X-deviations while the rhombi show the Y-deviations of the KinectV2 on the left and the Phab2Pro on the right side. The X and Y axes describe the pixel coordinates of the measured border of the cube. The distances of the measurements are not shown in the scatter-plots, but they all lie between 1 m and 2 m. The percentage values of 4.5 show a sophisticating result, therefore, simply averaging the two noise models was sufficient for the lateral case.

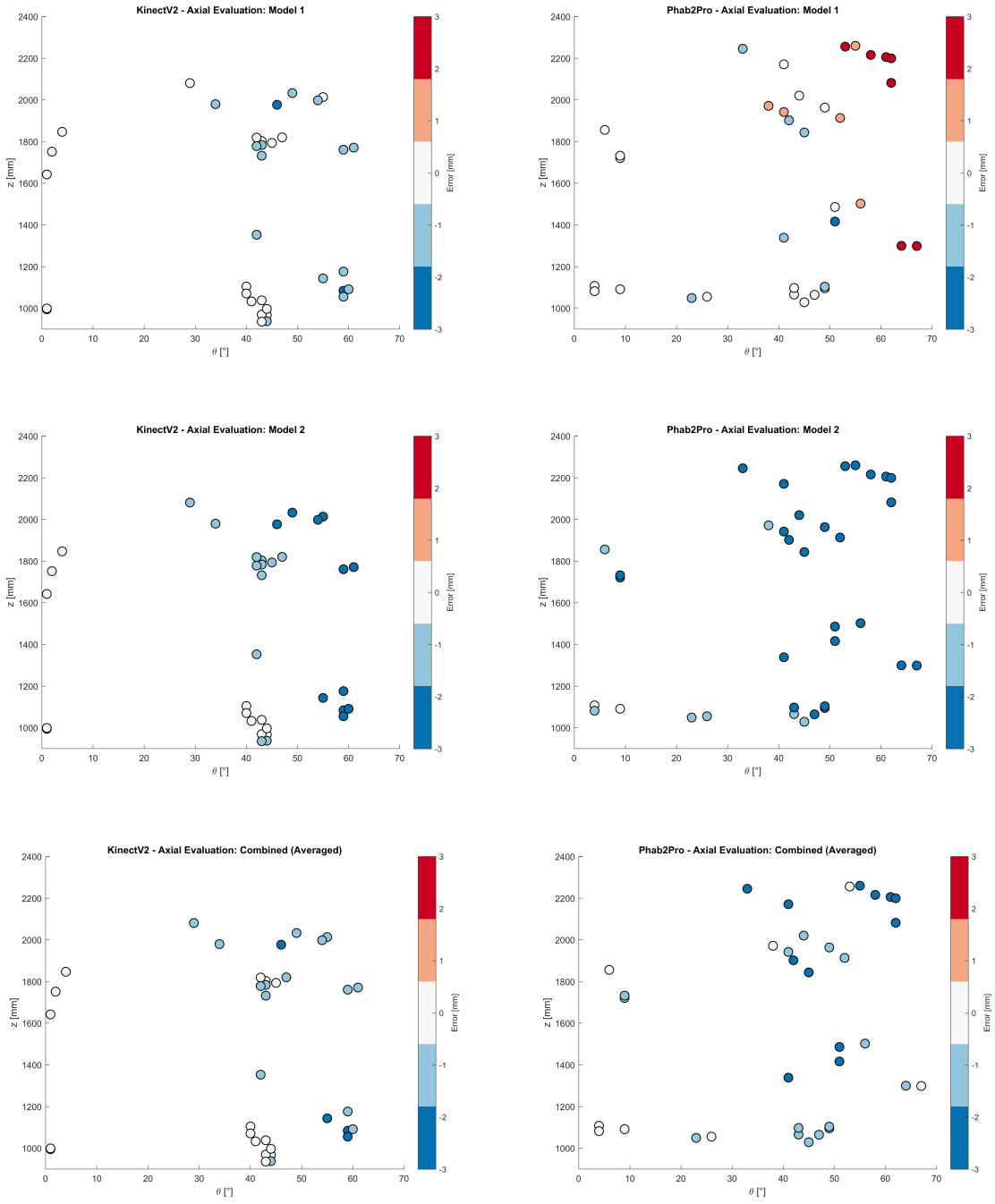

Figure 4.7: Results of the evaluation of both models and an averaged one [Gro17]

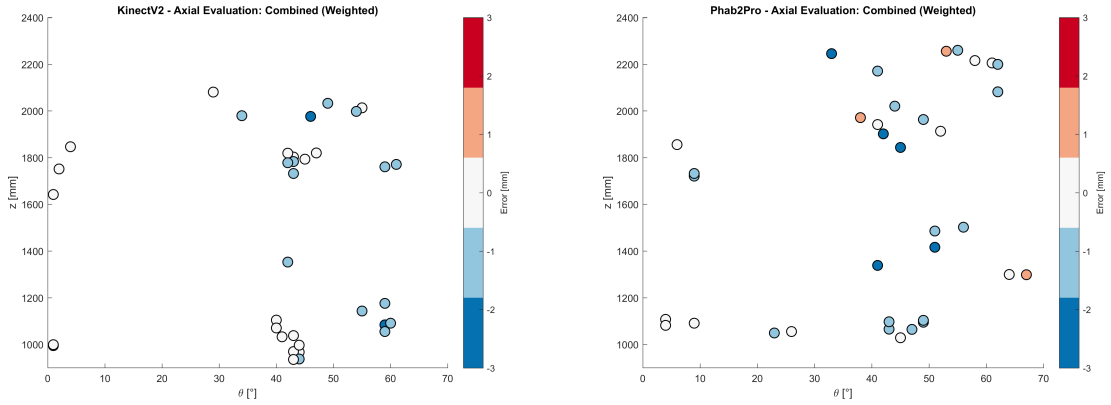

Figure 4.8: Results of the final axial model for both sensors [Gro17]

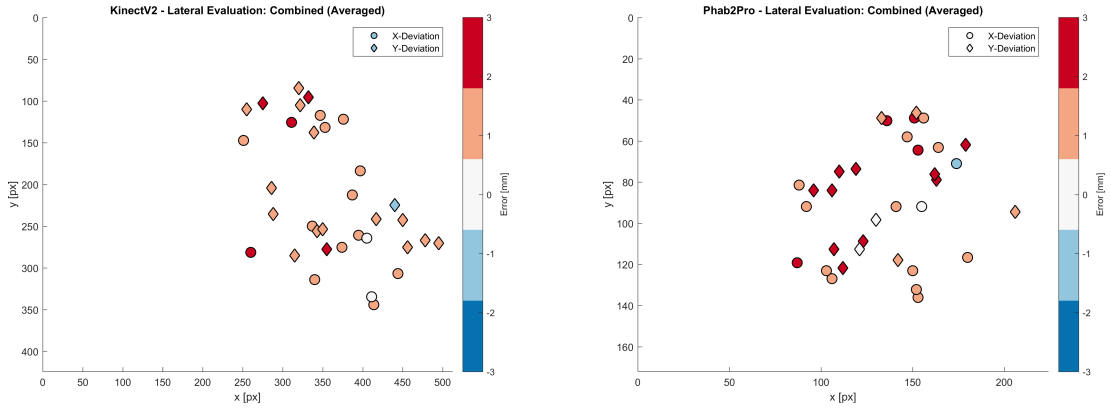

Figure 4.9: Results of the Lateral Evaluation

CHAPTER 5

Conclusion

We have created a combined noise model consisting of two different test setups, estimating axial and lateral noise for the KinectV2 and the Phab2Pro. Others have already developed noise models for the KinectV2, but we investigated the noise of the Phab2Pro for the first time.

The first setup was described by the thesis of my colleague – Nicolas Grossmann [Gro17] – measuring noise, using a plane placed at different distances, rotated to different orientations. The axial noise was measured at the surface of the plane, while the sensor was aligned horizontally and rotated by 90 degrees, in order to extract both the lateral noises in X and Y direction.

The second setup, described in this thesis, covers the whole frustum, taking the position in pixel coordinates and the distance into account. A 3D chequerboard was used to extract both lateral noises in X and Y direction at the edges of each field of the chequerboard.

Using the measurements of the two test setups, two empirically derived noise models were created and combined into a single one by using a weight function. The combined model was evaluated using a solid cube placed at different locations and orientations, leading to sophisticating results underlining the quality of our resulting noise model.

The model could further be improved by experimenting with different fitting functions and developing other improvements and techniques specifically aimed at a better coverage of the data, resulting in a smaller RMSE.

5.1 Resulting Models

	KinectV2	Phab2Pro
σ_{x_1}	2.9110	4.1207
σ_{y_1}	1.9617	3.6665

Table 5.1: Model 1: Lateral Results (0.9-percentile) [Gro17]

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

[Gro17]

	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 Coefficient Results (Equation 5.1) [Gro17]

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

	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 Coefficients for the X and Y Noise (Equation 5.2)

	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: Coefficients for the Z Noise (Equation 5.3)

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

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

[Gro17]

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

[Gro17]

	KinectV2	Phab2Pro
w_a	29	8
w_b	0.3	0.2
w_c	0	0.3

Table 5.5: Weight function coefficients for both sensors [Gro17]

List of Figures

List of Tables

List of Algorithms

Bibliography

[BL07] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.

[CLDB14] Benjamin Choo, Michael J. Landau, Michael D. DeVore, and Peter A. Beling. Statistical analysis-based error models for the microsoft kinect^{TM} depth sensor. In *Sensors*, 2014.

[FB81] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[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.

[Gro17] Nicolas Grossmann. Extracting Sensor Specific Noise Models. Bachelor's thesis, TU Wien, Austria, 2017.

[Low99] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[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.

[SLK15] Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb. Kinect range sensing: Structured-light versus time-of-flight kinect. *Computer Vision and Image Understanding*, 139:1–20, 2015.

[Sze10] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

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