Tampereen yliopisto

Roope Korkee

3D SCANNING WITH A KINECT DEPTH SENSOR

Bachelor's thesis
Faculty of Information Technology and Communication Sciences
Examiner: Prof. Joni Kämäräinen
May 2022

ABSTRACT

Roope Korkee: 3D scanning with a Kinect depth sensor
Bachelor's Thesis
Tampere University
Information Technology
May 2022

Real-life objects can be transferred to the virtual 3D world with 3D scanning techniques. Scanning can be performed either with a stationary camera while rotating the object or by moving the camera around the scanned object. This work investigates how a two-dimensional depth image of a depth sensor can be transformed into a three-dimensional point cloud and what filtering methods can be used to separate an object from its background. Point clouds taken from different directions are combined, and the quality of the obtained point clouds is studied.

Microsoft Kinect V2 depth sensor was used to take depth images. The device calculates the distance to the destination by measuring the flight time of the infrared pulse it emits. Color information for the image was available, but this work focused only on using distance information. The device was used to scan objects from several different viewpoints. Background points were gradually filtered out of each image until only the scanned object point cloud remained. Each image taken from a different angle is initially in its own coordinate system. Point clouds are aligned and connected to the same global coordinate system. Alignment is performed using an iterative closest point (ICP) algorithm.

The quality of the resulting point clouds for small objects is not good. The overall shapes of the resulting point cloud is correct, but the more accurate surface shapes disappear in the error noise. With a larger object, the error noise did not have as much effect on the result as the surface area was larger. This results a decent quality point cloud. A good scan result requires special features of the object and the environment. For example, reflective surfaces cause incorrect locations for points. These types of surfaces should be avoided or considered when handling data. Important point information may be lost in filtering, and errors may accumulate when points are combined using the ICP algorithm. The methods used are well suited for use if the goal is to obtain only a rough model of the described object.

Keywords: 3D modeling, ICP, point cloud, depth image, Kinect

TIIVISTELMÄ

Roope Korkee: 3D-skannaus Kinect-syvyyssensorilla
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Toukokuu 2022

Tosielämän esineitä voidaan siirtää virtuaaliseen 3D-maailmaan 3D-skannaustekniikoilla. Skannaus voidaan suorittaa joko paikallaan olevalla kameralla kohdetta pyörittämällä tai liikuttamalla kameraa skannattavan kohteen ympärillä. Tässä työssä tutkitaan, kuinka syvyyssensorin kaksiulotteinen syvyyskuva voidaan muuntaa kolmiulotteiseksi pistepilveksi ja millä suodatusmenetelmillä kohde voidaan erottaa taustastaan. Eri suunnista otettuja pistepilviä yhdistetään ja saatujen pistepilvien laatua tutkitaan.

Microsoft Kinect V2 -syvyysanturia käytettiin syvyyskuvien ottamiseen. Laite laskee etäisyyden kohteeseen mittaamalla lähettämänsä infrapunapulssin lentoajan keston. Kuvan väritiedot olivat saatavilla, mutta tässä työssä keskityttiin vain etäisyystietojen käyttöön. Laitteen avulla esineitä skannattiin useista eri näkökulmista. Taustapisteitä suodatettiin vähitellen pois jokaisesta kuvasta, kunnes jäljelle jäi vain skannatun esineen pistepilvi. Jokainen eri kulmasta otettu kuva on aluksi omassa koordinaatistossaan. Pistepilvet kohdistetaan ja yhdistetään samaan globaaliin koordinaattijärjestelmään. Kohdistus suoritetaan käyttämällä iteratiivista lähimmän pisteen algoritmia, ICP:tä (engl. iterative closest point).

Tuloksena saatujen pistepilvien laatu pienille kohteille ei ole hyvä. Pistepilvien ulkomuodot ovat oikeat, mutta tarkemmat pinnanmuodot katoavat virhekohinassa. Suuremmalla kuvattavalla kohteella virhekohinalla ei ollut kovin suurta vaikutusta tulokseen, kun kohteen pinta-ala oli suurempi. Tämä tuottaa melko hyvänlaatuisen pistepilven. Hyvä skannaustulos vaatii kohteen ja ympäristön erityispiirteitä. Esimerkiksi heijastavat pinnat aiheuttavat virheellisiä sijainteja pisteille. Tämäntyyppisiä pintoja tulee välttää tai ottaa huomioon datan käsittelyssä. Tärkeät pistetiedot voivat kadota suodatuksessa ja virheitä voi kertyä, kun pisteitä yhdistetään ICP-algoritmilla. Käytetyt menetelmät sopivat hyvin käyttötarkoitukseen, jos tavoitteena on saada vain karkea malli kuvatusta kohteesta.

Avainsanat: 3D-mallinnus, ICP, pistepilvi, syvyyskuva, Kinect

PREFACE

This thesis concludes 3 years of bachelor's studies in information technology at the University of Tampere. The motivation began with an interest in the Kinect sensor that I had never been able to try. This topic created an excellent opportunity to gain experience in how Kinect and 3D scanning work. A significant amount of information has been achieved on the subject with this work.

I want to thank the Department of Signal Processing at the University of Tampere for lending me a Microsoft Kinect sensor. This study has provided an excellent opportunity to study the matter in practice and learn more about it. Nowadays, it would be difficult to get a Kinect sensor, as Microsoft's support for this device has long since ceased to exist, and such devices are no longer being manufactured. I would also like to thank my family members who have helped with the work by proofreading this document and suggesting improvements. I would also like to thank my supervisor, Professor Joni Kämäräinen, for providing a good research topic and support as the work progressed.

Tampere, May 16, 2022

Roope Korkee

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

3D	Three dimensional
ICP	Iterative Closest Point
IR	Infrared
PCL	Point Cloud Library; Open-source code library for modifying point clouds
RANSAC	Random Sample Consensus
RGB-D	Red, Green, Blue, and Depth; color image with pixel depth values
ROI	Region Of Interest
TOF	Time Of Flight

1. INTRODUCTION

3D modeling is a widely studied field. For example, Lai and Su [1] proposed a method to map the environment using a depth sensor-equipped mobile robot. The mobile robot could localize itself using neural networks. Surroundings can be created for augmented reality or virtual reality applications where the user can interact in the virtual environment with the 3D objects. Creating a virtual object usually requires direct 3D modeling, for example, with 3D-modelling software. With 3D scanning technology, an object or even an entire environment can be brought into the virtual world, and realistic-looking virtual objects can be created.

Figure 1. *3D scanned sofa and the resulting complete 3D point cloud*

However, 3D scanning as a process is not simple. The process usually requires a camera that gets distance information from the image it takes or a similar method where the distance values can be computed. The captured depth image is initially a two-dimensional image, with each pixel containing distance information from the camera to that point. This 2D depth image is converted to a 3D world coordinate system. The point clouds taken from different directions should already be sufficient to be combined using the iterative closest point (ICP) algorithm, but the data acquired is not perfect and includes errors. ICP is a computationally demanding process when there are many points to be aligned. Different filtering methods should be used to remove the extra points from the point cloud. Reducing the points improves the efficiency of the used algorithms dramatically and helps with the point cloud alignment as outlier points can cause the alignment to misalign.

This work focuses on the 3D scanning of single objects, a game controller, and a bigger object, a sofa, as seen in Figure 1. The focus is on testing the methods on the game controllers and seeing how the used methods can be utilized with the large object.

The overall goal of this work is to get acquainted with 3D modeling using Microsoft Kinect V2. For simplicity, only the depth data is used in this work, although RGB color data for the image is also available. The motivation is to get acquainted with the 3D modeling step by step from the depth information of the picture. The work aims to determine how good results are obtained by utilizing only the acquired point clouds and existing code libraries. The work examines sources of errors and problems, and seeks possible solutions for the found problems.

This document is structured as follows. Chapter 2 provides background information on the studies on the RGB-D sensor related to 3D modeling using depth data. The methods used in this work are discussed in chapter 3. Chapter 4 describes the use of the methods in the conducted experiments. Chapter 5 presents the results obtained from the experiments, and chapter 6 provides suggestions for improvement for further research regarding this paper.

2. LITERATURE BACKGROUND

2.1 3D scanning technologies

Song et al. [2] discussed different methods for building a 3D model on different 3D scanning technologies and used Kinect to create a 3D view of a scene. The 3D model can be made with a stereo camera system, a laser scanner, or a time-of-flight sensor (TOF).

The 2D picture taken with a stereo camera can be converted to 3D when the cameras' positions, viewing directions, and intrinsic parameters are known. By comparing similarities in the images, regions of interest (ROI), such as edges, can be found. The images are combined, and geometric methods can determine the corresponding points in the three-dimensional coordinate system. Y. Huang and M. Chen [3] used these methods to create 3D models of simple geometric objects with a stereo camera. This article provides a more detailed geometric overview of how a camera system works and how images are combined. Moro et al. [4] used a stereo camera system to create an entirely virtual three-dimensional environment utilizing ROI and object recognition. The result of this work is beneficial as an application for mobile robots that need to localize themselves on a local map. Jiajun Zhu et al. [5] developed a prototype 8-camera stereo camera system that can capture 360-degree panoramic images and quickly form a 3D view of the environment based on these images. The paper also compared the features and results of the laser scanner in this paper and the manufactured prototype.

2.2 3D reconstruction with Kinect

There are different versions of the Kinect depth sensor. The first version of the sensor is Kinect v1, released in 2010. Kinect v2 was released in 2014, which improved many features compared to the previous version. Both sensors have a color camera, an infrared (IR) camera, and an IR projector. The most significant difference between the versions is how they measure distances. Using geometric methods, Kinect v1 uses structured light to determine how far the target is from the sensor. Kinect v2 uses the time-of-flight (TOF) to estimate the distance to the object, resulting in less erroneous depth images and more accurate distance information.

TOF-device emits a light pulse, the flight time of which is measured. The distance can be calculated by the flight time and the speed of light. TOF cameras are also relatively inexpensive compared to accurate but also more expensive laser scanners. With the

release of Kinect, the number of studies related to 3D reconstruction increased massively.

Azure Kinect was released after Kinect v2. Kurillo et al. [6] compared Azure Kinect to Kinect v2 sensor by examining the imaging of a flat plane from different distances and comparing the results. Data taken from the same wall with a laser scanner were used as a reference. Lachat et al. [7] compared the versions of Kinect with each other and performed a reconstruction of the 3D model, and analyzed the sources of the error.

In [8], Yang et al. scanned objects on a rotating platform with Kinect and proposed new methods for connecting the point clouds. Combining point clouds is much easier in this case because the camera position is known to be stationary.

2.3 Calibration and error sources

Camera calibration is also an essential step in reducing the effect of lens distortion on the result. The calibration aims to determine the values of the camera's intrinsic parameters. Smisek et al. [9] examined the camera parameters and distortions. Knowing the correct calibration values also allows for converting 2D image depth value pixels to 3D points without the distortion caused by the camera system. A chessboard pattern is usually used for calibration. In [10], Hansard et al. explored how this pattern can be detected automatically, making the calibration easier.

Among the features of the camera, the focal length, as well as the focal point, must be utilized to make the depth data three-dimensional. In the paper [11], Lachat et al. present a formula for converting the depth image coordinates (*u*, *v*) to the XYZ coordinate system as

$$\begin{cases} X &= \frac{u-c_x}{f_x} \cdot Z \\ Y &= \frac{v-c_y}{f_y} \cdot Z, \\ Z &= Z \end{cases} \tag{1}$$

where u and v are the indices of the image matrix horizontally and vertically. Variables c_x and c_y are the coordinates of the principal points of the image in the image matrix. Variables f_x and f_y are the camera's focal point coordinates. The Z coordinate is obtained from the depth image, the same z coordinate as the Z in the XYZ coordinate system.

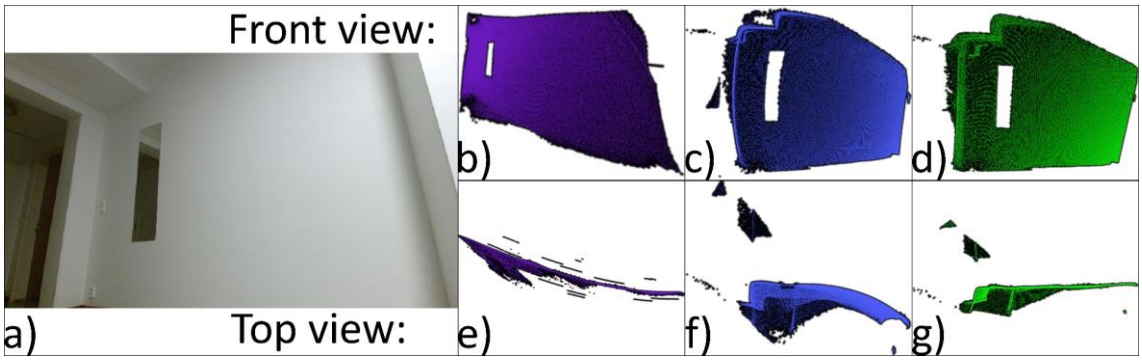

Figure 2. *Front (b, c, d) and top (e, f, g) point cloud views of a planar wall. (a) Original image. (b, e) Kinect raw depth data. (c, f) Point cloud in XYZ coordinates with lens distortions. (d, g) Point cloud in XYZ coordinate system without lens distortions*

A three-dimensional point cloud is obtained from the depth image, as in Figure 2, pictures *c* and *f* visualize the result if only the equation (1) was used. In the picture, the planar wall is round instead of flat. The Kinect IR camera's intrinsic parameters can be used to correct radial and tangential distortions from the depth data. Figure 2, pictures *d* and *g* also show a distortion-corrected version of the same depth data. Kurillo et al. [6] provide more detailed information about the correction of the distortions in their paper. The Libfreenect2[1] code library provides a method to convert the raw depth data into a distortion corrected 3D point cloud to convert the point directly, considering the camera parameters. A 2D depth image is transformed into a 3D point cloud with no noticeable distortion using the camera's default intrinsic parameters.

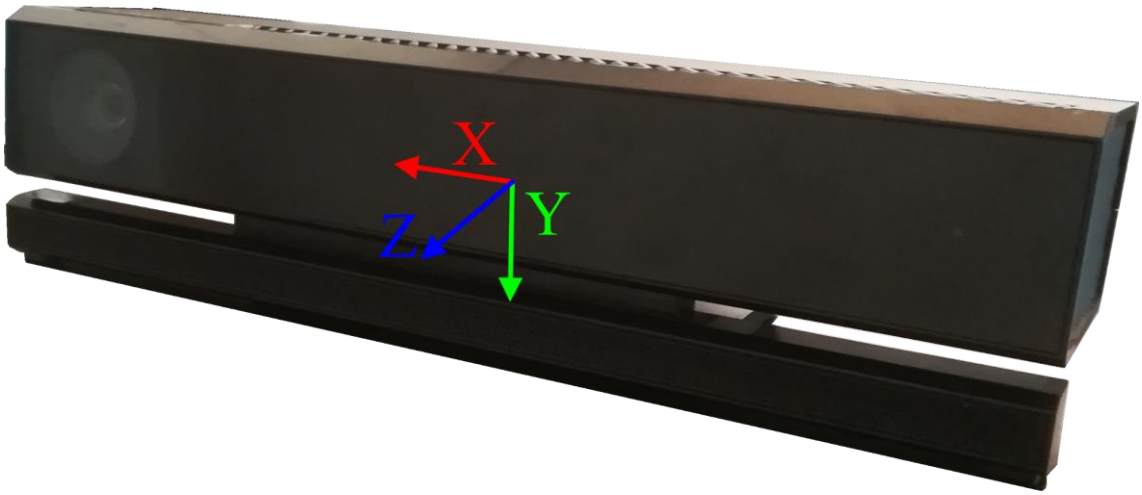

Figure 3. *Kinect coordinate system*

Equation (1) centers *u* and *v* image coordinates and scales them to the appropriate distance from the given axis. The Kinect sensor can get distance measurements between

[1] https://github.com/OpenKinect/libfreenect2

0.5 and 4.5 meters [7]. The distance values are the same as the Z-values in the XYZ coordinate system. Therefore, the camera must be located at the origin of each depth image. The coordinate axis directions are shown in Figure 3.

Khoshelham [12] investigated the effect of distance on depth image quality. As the distance increases, so does the uncertainty of the point location in the point cloud. If the Kinect is too close to the object, the transmitted light pulse may arrive earlier than the device can receive, resulting in invalid results. The error variation of one pixel is initially within a radius of about 5 mm from the correct position. After about 30 minutes of the device running, the error variation drops to about 1 mm, significantly improving accuracy.

2.4 KinectFusion

The primary motivator for this work is KinectFusion [13], made by Newcombe et al. The Kinect camera can be held in the user's hand and moved freely. At the same time, a 3D model is created in real-time based on the captured views. The captured frames are put on a global coordinate system, and the camera position is tracked by the needed rotation and translation calculated by the ICP algorithm. On the consecutive frames, the movement is minimal, and the alignment is successful as the point clouds are nearly aligned already. Only the depth information is used in the paper, so there is no need to consider lighting or errors caused by the color data. KinectFusion utilizes the CUDA cores of the graphics card to enable parallel computing and real-time operation.

Many other papers also use KinectFusion in their work. In [14], Yue et al. developed an algorithm that is even faster than KinectFusion. In [15], Qin Te et al. corrected errors in KinectFusion and improved the efficiency of the ICP algorithm. Doan et al. [16] selected an object from the view to be deleted and filled the resulting holes with new points of the correct color.

Teng et al. [17] proposed an interactive method for removing the item from the background when the user marks which part of the points belong to the scene and which to the object. Izadi et al. [18] modified the original KinectFusion to allow the user to interact with the created 3D environment where 3D objects that follow the physics can be added to the 3D view. User intervention is also a central part of this thesis, as the aim is to analyze the content of the various stages in more detail.

3. METHODS

Bernardini and Rushmeier [19] described what should be considered in 3D modeling in general. The acquisition of the 3D model is covered only superficially as there was no need for texturizing as the color data were not used, and the object surface mesh was not constructed in this work. The focus is on capturing only one object at a time, so there is no need to include the points in the background. Point filtering is utilized, resulting in only a point cloud of the object. Filtered point clouds are combined using the ICP algorithm.

These filtering methods are unnecessary as the point clouds could be combined straight with the ICP algorithm. The problem is that the ICP method slowed down significantly as the number of points grew. Also, the ICP is sensitive to the erroneous points that do not belong to the object causing incorrect alignment.

3.1 Depth data acquisition

Kinect V2 has a resolution of a single depth image horizontally 512 pixels and vertically 424 pixels. The sensor is capable of a framerate of 30 frames per second. [7]

The Libfreenect2 code library provides the drivers that allow to connect to a Kinect V2 device and obtain and use the camera image data directly. This library also provides a function to convert the depth image pixel to a point in the *XYZ* coordinate system with minimal distortion.

The disadvantage of the conversion process is that each pixel in the depth image must be individually converted into an *XYZ* point in a for-loop. The conversation slows down the program significantly when taking a picture. Taking a single image took approximately 350 to 450 milliseconds. The hardware used is relatively old; see section 4.1 for more information. The used hardware is approximate as old hardware as the Kinect V2 is. Newer hardware would perform much better with this task. Still, a much more efficient method would suffice, but this was the only method that worked.

3.2 Filtering

The filtering methods used in this thesis follow the same techniques used in Santala's [20] thesis. The filtering methods presented can all be found in the PointCloudLibrary[2]

[2] https://github.com/PointCloudLibrary/pcl

(PCL). The filtering methods are easy to implement as all the methods can be compiled from the same source. The idea of filtering is to obtain only those points in the point cloud that belong to the scanned object. Filtering also provides better performance in processing point clouds when redundant points are not processed. Also, handling a smaller point cloud is easier in the following filtering and ICP-aligning methods.

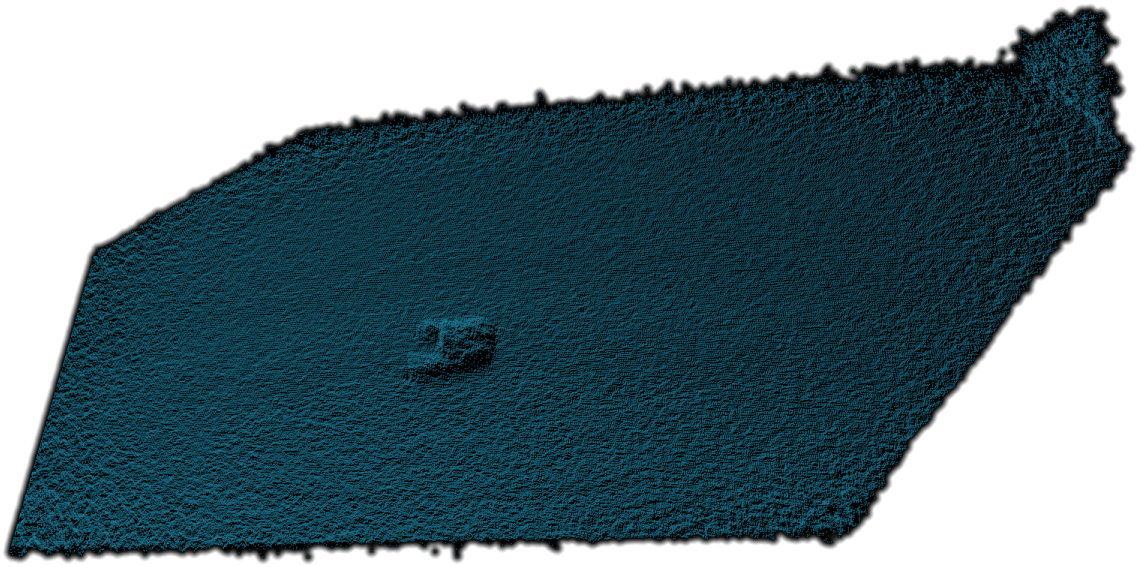

Figure 4. *Point cloud of a game controller on a flat floor*

Figure 4 shows one view of a game controller on a flat empty floor. The number of points in the background is massive compared to the points of the game controller. Filtering is needed to get rid of redundant background points.

3.2.1 Distance filter

Distance filter is a passthrough filter that is given a distance range. The points outside that range will get filtered out. The remaining is a point cloud that contains the wanted object and the platform on which the scanned object is located. In the experiments, this filter reduces the redundant points significantly.

This filter is unsuitable when the camera is moved, resulting in the required distance band to be changed. This filter could be better suited for a stationary camera where the distance bands are known. However, this filter could keep the point distance uncertainty at certain thresholds as the uncertainty for the point location grows with the distance. Alternatively, if the goal is to scan the object with a moving camera, the distance band for the filter can be measured.

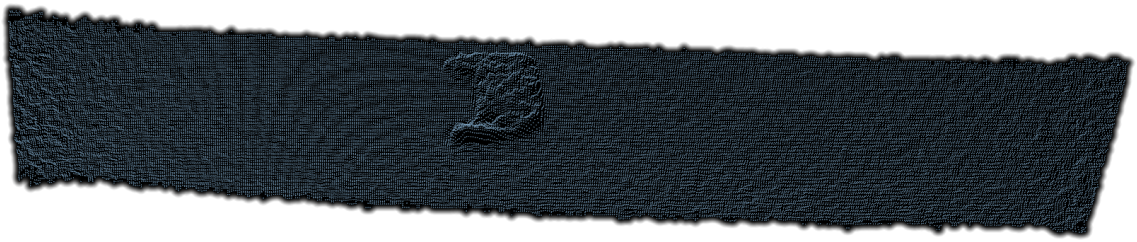

Figure 5. *Distance filtered point cloud*

The distance filter is used in Figure 5. The points with a too high or too low Z-value are filtered out. The resulting point cloud is a wide band of points where in the middle lies the scanned object.

3.2.2 Downsampling

Downsampling is a filtering method to reduce the number of points in the point cloud by averaging the point locations. The 3D space is filled with 3D cubes, called voxels. Depending on the size of the voxel, one voxel in the voxel grid can have multiple points inside. The center of mass, the centroid, is calculated from these points within the voxel, and only the calculated centroids remain in the point cloud. This method improves efficiency by significantly reducing the number of points. At the same time, this causes inaccuracies in the locations of the points because the filtered points are only averages of the original points.

This filtering method was used to test what effects this filter has on the resulting point cloud after the ICP alignment. The filter was used with the ICP alignment step to limit the number of points from growing after each merging.

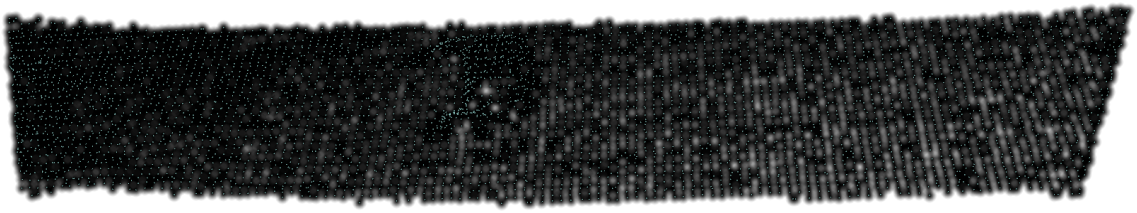

Figure 6. *Downsampled point cloud*

Figure 6 shows the point cloud after distance filtering and downsampling. When comparing Figures 5 and 6, the resulting point cloud is not as dense, and the points are in a grid-like pattern.

If the voxel size is too tiny, points will not get filtered as each voxel contains at most 1 point. Instead, if each voxel contains at least 2 points, the points get filtered. Small and detailed shape information will disappear from the point cloud if the selected voxel size is too large.

Large voxel size can be selected if the object's physical size is large and does not contain small, exact surface details. It depends entirely on the application for which the filtering is used and needed. The size of the proposed minimum filter voxel size should be chosen as it should filter the points at least half the original amount. When each voxel contains an average of 2 points to be averaged, the resulting point cloud keeps a decent quality.

3.2.3 Planar removal

Planar filter finds a plane that maximizes the number of points nearby. Either the plane or the rest of the points can be removed. In this work, the scanned object was placed on a flat floor. The planar filter was used to separate the obstacle from the scanned object. As a result, the resulting point cloud contains only the desired object.

Planar filtering uses the Random Sample Consensus (RANSAC) [21] algorithm to search for a plane and its associated points efficiently. The points of the described point cloud already deviate from the plane due to measurement errors and remaining lens distortions. So, it is necessary to determine the maximum distance to the found plane included in the filtering. These planar points are removed from the point cloud, leaving only some groups of points, the clusters, depending on how many objects are in the acquired point cloud.

With this filter, it should be noted that the object's surface must not have flat surfaces. Otherwise, it may be filtered out during the filtering process. For example, if the captured view has a table and a large display screen on the table, the screen could be selected as the plane to be filtered. The incorrect plane found can also cause some points in the table to be deleted as they are within the found plane. For this reason, it would be good not to have other objects in the vicinity of the main object, especially objects with a large planar surface.

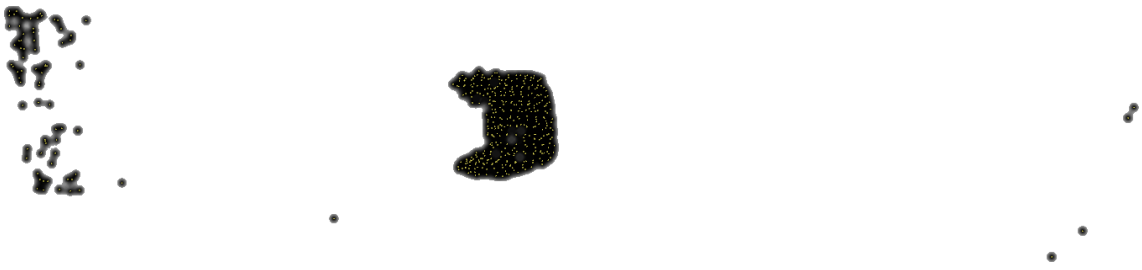

Figure 7. *The point cloud after planar removal*

Figure 7 shows the point cloud after removing the planar component from the point cloud. The image shows that the floor component is removed. The resulting point cloud has some points remaining from the floor caused by the camera lens distortions making the floor slightly curve.

This filter can cause undesired results. As such, the use of this filter should be carefully considered. Instead, the following filtering method, cluster extraction, could be used to separate the scanned object from the background.

3.2.4 Clustering

In *cluster extraction*, the separate groups of points at most a certain distance from their nearest point neighbor can be found. The Euclidean cluster separation algorithm is utilized in clustering. A k-dimensional tree [22] search is used to group the points. Clustering was used to extract the game controller point cloud from the remaining background points.

The distance parameter must be chosen carefully because too small a value causes too many smaller clusters. If the value is too large, the clusters can join one bigger cluster, defeating this filtering method's purpose. The minimum and maximum sizes of clusters can also be selected. If the incoming cluster does not have at least the minimum number of points, that cluster is excluded from the output. The maximum cluster size constraint is not used because having too small a value would limit the original larger cluster to only a smaller portion of the larger cluster, which is not desirable.

Figure 8. *Controller cluster extracted*

Clustering is unnecessary if the scanning location is optimal, smooth, and free of other objects. The resulting point cloud might already be filtered to include only the desired object. However, the filtering removes the erroneous individual points from the remaining point cloud that have not yet been filtered. In Figure 8, the point cloud includes only the points in the controller cluster from Figure 7. Any other points in the background have been discarded, resulting in only the points from the scanned object.

When scanning a single object, it can be assumed that the object's point cloud is the largest of all the clusters. If there were more objects in the view, filtering would result in all clusters being in their own sets, and the correct ones should be selected for further processing, as the output may contain invalid clusters. The selection could use an identification method to identify the point clouds according to their shape. However, in the code, all cluster point clouds are outputted separately. For this reason, the object must be the only or at least the largest in the environment. The filtering parameters must be carefully selected to acquire the correct object cluster.

3.2.5 Inverse filtering

Inverse filtering is a method to select points from the original unfiltered point cloud based on another point cloud. This method created a denser object cluster from the original point cloud using another less dense point cloud cluster.

In this filtering method, the goal is to acquire the point locations from the original depth data. The denser cluster was created using the filtered cluster as an input cloud. All the points in the original point cloud that are a certain distance from the input point cloud are included in the outputted point cloud. This method gives a new object cluster that is more accurate than the usually filtered cluster. This filtering method also brings previously filtered out erroneous points back.

Figure 9. *Inverse filtered controller cluster*

An example of the inverse filter can be seen in Figure 9. The cluster was extracted from the original unfiltered point cloud in Figure 4 using a source point cloud from Figure 8. The point cloud is a lot denser and has finer surface details.

3.3 Iterative closest point

Bernardini and Rushmeier presented in the paper [19] the working principles of the ICP algorithm. Rusinkiewicz and Levoy [23] showed different algorithm variations and compared the performance of the variations of the ICP. The general principle of operation of the ICP algorithm is as follows:

1. Find the nearest point from the target point cloud to each point in the source point cloud.
2. Find the required rotation and translation for the point cloud that minimizes the mean square error between the points.
3. Transform the point cloud to the rotation and translation estimated in the previous step.
4. Return to step 1 and continue the iterations until the resulting mean square error is zero or is less than the specific error limit.

Points can be selected from the source and destination point clouds that should be targeted. External invalid points for which no immediate neighbor can be found are discarded. Other error methods can also be used, such as point-to-level error metrics. [23]

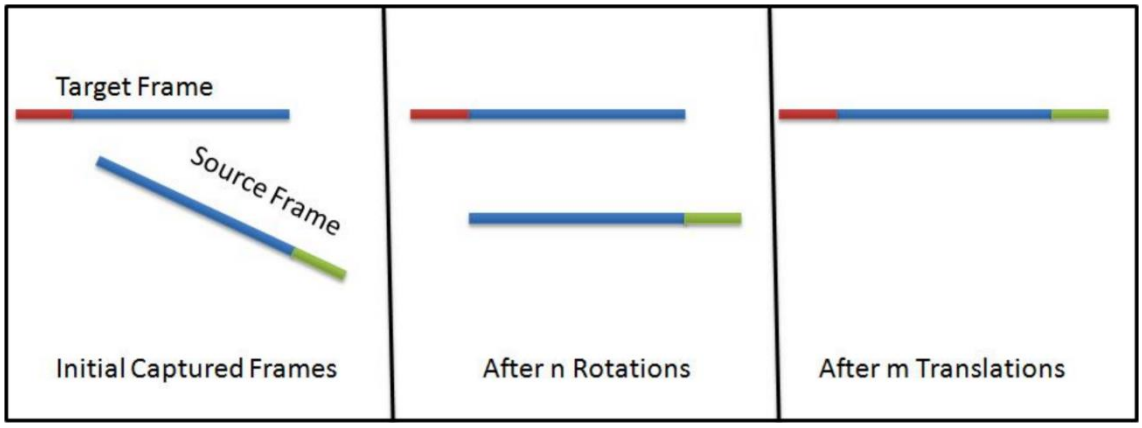

Figure 10. *Example of ICP algorithm. Manojkumar and Reddy [24]*

Figure 10 shows a simplified version of the ICP algorithm. The 2 lines can be considered the point clouds to be aligned. The source frame will be aligned with the target frame. The blue part is the overlapping part of the point clouds, the red and the green parts are unique parts of the point clouds. In this example, the rotations and translations are done separately, but rotation and translation are done in the same iteration in ICP. In the next iteration, the source frame is at a new location to which a new rotation and translation are calculated. The point clouds will converge as a new transformation is calculated to minimize the error metric. In the case of Figure 10, the blue parts will be aligned. After

converging, the point clouds are merged, and the resulting point cloud gets updated with new unique points from the source frame.

This work utilizes the interactive ICP code of PCL, which has been slightly modified to obtain better alignment results for point clouds. ICP may not always find the correct target location for the point cloud, and a new iteration will not correct this. For this reason, the user can move the point cloud itself to align it to the correct location better. The user can move the point cloud in the *X*, *Y*, and *Z* directions and rotate the point cloud around the *X*, *Y*, and *Z* axes around the centroid of the point cloud.

A 4x4-sized transformation matrix transforms the points to a new location with rotation and translation. The 4x4 matrix is defined as follows

$$M = \begin{bmatrix} R_{3X3} & T_{3X1} \\ 0 & 1 \end{bmatrix}, \tag{2}$$

where R_{3X3} corresponds to the rotation matrix used to rotate the point cloud, and T_{3X1} corresponds to the translation matrix that moves the points in the direction of the translation vector.

Let a point in the point cloud be defined as $\mathbf{p} = (p_x, p_y, p_z, 1)^T$, R_{3X3} is an identity matrix and $\mathbf{T} = (t_x, t_y, t_z, 1)^T$. Therefore to translate the point $\mathbf{p}$ in the direction of $\mathbf{T}$, the new location $\mathbf{p}'$ is defined as

$$\mathbf{p}' = M\mathbf{p} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{bmatrix} = \mathbf{p} + \mathbf{T}. \tag{3}$$

Thus, in translation, it is sufficient to add the position coordinates of the transfer vector to the coordinates of the point cloud points.

In this work, the point cloud is set to rotate around its centroid point in the selected direction of rotation. The rotation matrices in the rotation of the point about the axis are

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{4}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{5}$$

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

where θ corresponds to rotation angle in the positive direction of rotation. [25] Equation (4) corresponds to rotation around the x-axis, equation (5) to rotation around the y-axis, and equation (6) to rotation around the z-axis. For a point cloud to rotate around its centroid, it must first move to the origin in the negative direction of the centroid location vector according to equation (3). Rotate it around the desired axis and then move back to the original position in the positive direction of the centroid location vector.

For example, let us rotate the point $\mathbf{p}$ around the x-axis with an angle of θ. Let the centroid of the point cloud be $\mathbf{c} = (c_x, c_y, c_z, 1)^T$ and $M = \begin{bmatrix} R_x(\theta) & 0 \\ 0 & 1 \end{bmatrix}$. The new position $\mathbf{p}'$ of the position vector $\mathbf{p}$ is thus obtained from the equation

$$\mathbf{p}' = M(\mathbf{p} - \mathbf{c}) + \mathbf{c}. \tag{7}$$

The alignment performed by the user does not have to be very precise. The ICP algorithm can proceed with the new point cloud position and improve accuracy as the points align better.

4. EXPERIMENTS

The methods were studied by selecting the Steam controller as the main object to be scanned. As seen in Figure 11, the controller is sufficiently asymmetric, so the alignment of point clouds should be easy to perform correctly. In addition, the size of the object is small enough to make point cloud editing fast. A flat surface with no other objects was chosen as the scanning location. A total of 8 images were taken of the controller from about 1 meter in each direction. No more pictures were taken to make point cloud management easy and quick.

Figure 11. *Steam controller*

Another larger object, a sofa, was also scanned to see how the methods could be generalized. The texture is not reflective, so the results should be more accurate at the surface level than the game controller. Also, the processing time will be longer as the object includes more surface area because there are more points in the point cloud to be handled. The sofa point clouds did not align properly together for the most part, even with a manual adjustment. The resulting point cloud was mainly combined by manually controlling the target point cloud.

4.1 Hardware

The experiments were run on Ubuntu 20.04 operating system. These tests could not be performed on Windows due to compatibility issues with various code libraries and drivers for the Kinect sensor. Kinect V2 (Kinect for Windows) was used as the depth camera. The central processing unit used was an Intel i5 2500, and the graphics card used was

Nvidia GeForce GTX 970. The graphics card was needed to visualize the Kinect camera view and the point clouds at various stages of the study.

4.2 Filtering

The distances at which the described game controller remained in the point cloud were tested for depth filtering for each point cloud. The filtering distance limits varied, although the scanning distance remained approximately the same. The distance to the controller was about 1 m, and the filtering distances were 0.8–1.1 m. It is recommended to determine the object's estimated nearest and farthest distance at the scanning time. This data gives good, estimated values for the required filter limits.

Target point cloud filtering was evaluated with voxel boxes of varied sizes in downsampling. Values were assessed between 1 mm and 20 mm. At 1 mm, the points were not filtered out. With slightly bigger voxels, the points began to filter out. At about 5 mm, the number of points was half the original. A fair value of the voxel size was found to be 10 mm. The points were filtered to about 1/8 of the original point cloud size, and the point cloud shape resembled the original controller shape.

In planar filtering, the game controller and the floor level were separated. The error distance can be selected from the found plane in the filter. Because there are still lens distortions in the point cloud, the plane is slightly curved, at least at its farthest edges. The distance error was evaluated from small distances (1 mm) to larger sizes until a good result was obtained by removing most of the floor planar points. The error distance should not be considered exceptionally large, as points near the bottom of the game controller will also be filtered out. The maximum length at which most of the plane got filtered without removing the bottom of the controller was 7 mm.

After planar filtering, there were still individual points left. A cluster extraction was performed to get rid of the extra points. The optimal distance tested 11–18 mm was so that the closest erroneous points were not included in the game controller cluster. At distances less than 11 mm, many essential points began to be filtered from the cluster. The game controller's point cloud cluster consisted of 370 points. The lower limit for cluster size was set at 300 points. The upper limit was not placed because it would be too small to reduce the size of the larger cluster and would not benefit from this need for filtering.

The inverse filtering was also performed to compare the quality of the resulting point clouds. With a search distance of 1 mm, the outputted point cloud had less of the points than in the input cloud cluster. With a too high search distance, the points on the floor

were included in the filtered output. A search distance of 10 mm was used, bringing all the original points around the game controller cluster to a new point cloud.

The sofa point clouds were filtered in almost the same methods as the game controller. Downsampling was not used as the goal was to keep the quality as high as possible. Inverse filtering was not needed as the quality stayed the same. The largest distance to the found plane in planar removing was 80 mm. Changing this parameter also changed the found plane. Too high a value and another planar surface were found from the object. The cluster extraction was used with a search distance of 60 mm, and a minimum size of the cluster was set to 1 000 points to filter the smaller cluster off.

In general, the filtering parameters do not generalize. The values of the parameters depend highly on the scanned object and the surrounding environment, which makes selecting the filtering parameter values hard. If the scanning process was made automatic, the use of the filters should be minimized.

4.3 Point cloud alignment

As mentioned in section 3.1, the conversion from a 2D image to the 3D point cloud took a lot of time, and the idea of real-time or timed image capturing was discarded. The point cloud alignment was used on point clouds where the camera positions are all far from each other and with a different viewing angle. The point clouds from different directions on the game controller are not aligned. Each point cloud can be thought to be its own coordinate system, all of which have an origin corresponding to the camera's position at the scanning time.

When running the ICP algorithm, it was found that point clouds do not consistently settle in the right place. In addition, there is no preliminary information about the location of the point cloud in the case of separate point clouds. An approximating preliminary guess cannot be made to the desired location so that the ICP iterates to the correct location.

The needed rotation and translation were too big for the algorithm to converge correctly. Two different methods were used for incorrectly aligned point clouds, point cloud inverse filtering and allowing the user to move the point cloud to a better location between ICP iterations.

The assumption was that there were so few points that ICP could not detect the correct targeting points. Inverse filtering was developed, which takes from the original point cloud all the points at a certain distance from the points in the controller cluster. The result is a denser point cloud that is more accurate than the filtered. ICP still produced incorrect positions in the alignments.

The testing also included modifying the PCL interactive ICP code so that the user can move the point cloud along the *x*, *y*, and *z* axes. In addition, the point cloud can be rotated about the *x*, *y*, and *z* axes in small increments, allowing the user to move the point cloud at 6 degrees of freedom.

Manually correcting a moving point cloud was challenging because there were no coordinate axes in the view that would indicate which direction the rotation or displacement would occur. However, after experimenting with the movement commands, the correct orientation of the adjustment was found. There was no need to be precise here because as the iteration process continued, the algorithm found a better optimum for the point cloud and aligned the point clouds better. With the proper alignment, the point clouds were merged.

In this test case for the game controller, the direction of rotation direction *z* was the best to move the point cloud. The camera took pictures around the game controller, almost rotating around the z-axis of the camera.

The merged point cloud was filtered using the downsampling method to mitigate the error caused by erroneous and too many points in the merged point cloud. The objective was to average the points in the merged point cloud so the successive merging could align better. This interactive ICP method was also used to combine the inversely filtered point clouds of the controller. The merging of the denser point clouds was done to compare the quality of both point clouds.

It was tedious to align the sofa point clouds. Each point cloud to be aligned contained 51 000 to 94 000 points of the object. For this large number of points, the ICP algorithm was slow. One iteration took approximately 10 seconds at most. After each alignment, downsampling was performed with a size of 3 mm. With this size, the original point clouds do not get filtered, but the overlapping points after merging got averaged with the filter, limiting the size of the resulting point cloud a little.

Also, the ICP could not align the point cloud, even if the source point cloud was set manually on top of the target point cloud. For this behavior of the ICP, most of the sofa point clouds were moved in the right direction with 1 iteration and moved manually to the correct location. The reason might be for the systematic misalignment that new points in the source point cloud are tried to be aligned to the existing target points, causing incorrect converging location.

5. RESULTS

The Kinect sensor does not require much for the operating system. The minimum requirements are easily met with current computers. As the Kinect v2 is old technology, the related research and software were done some time ago. It was difficult to find working programs or guides to help when compatibility issues arose.

The experiment results showed that the object's surface should not be reflective. The surface of the game controller is mainly matte, but it is also slightly glossy. The reflectivity caused the points of the controller to be too far from the correct position due to reflection. As a result, the shape of the point cloud is slightly distorted. This distortion is most noticeable in the z-direction, in the separate point cloud views of the object.

It should be noted that Kinect can produce point clouds with varying accuracy. At the nearest distance of 0.5 m, the distance between adjacent points is about 1.4 mm, and at 4.5 m, it is about 12 mm [7]. Because the depth data cannot be acquired closer than 0.5 m to the object, a more accurate point cloud cannot be obtained from the object's surface shapes. The game controller described is too small for the camera's field of view, so there were extra points left in the background that had to be filtered out.

There are also errors in the distance data from other sources. The image has lens distortions, and calibration may not give a complete correction. The distance values of the device may vary due to the device's systematic measurement error.

As the study found, ICP does not always find the correct location in the target point cloud. An initial guess for the estimated transformation would have helped in the general case, but such approximating guess was hard to make with this experimenting setup. The ICP did not align because too few or too many points were in the point clouds. The wrong optimum is too easy to find in a dense point cloud when there are not as many points in the movable point cloud. The solution to this could be to keep the number of points in the target point cloud unchanged, for example, by downsampling. The accuracy of the point cloud shape does not improve, but the points move in better positions on average.

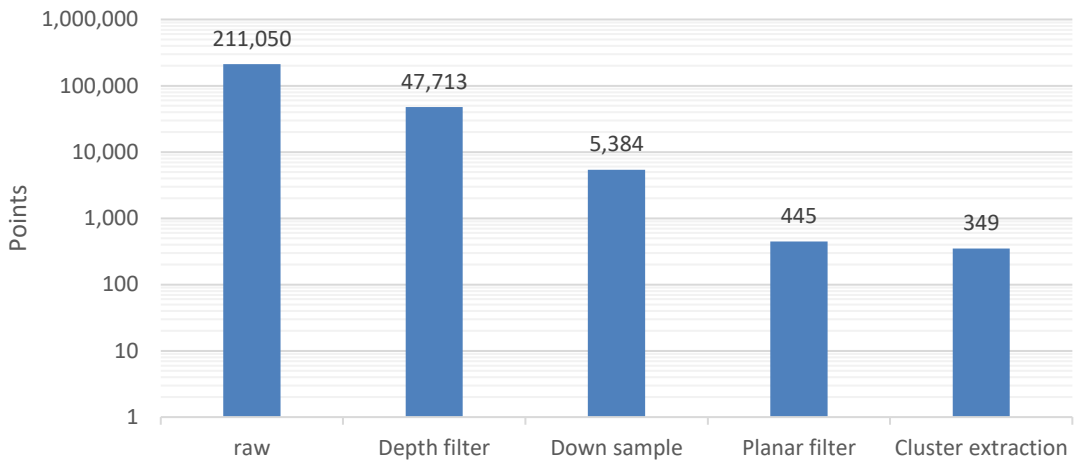

Figure 12. *Bar chart of the point cloud sizes after each filtering step*

Figure 12 shows how essential filtering functions are to maintain efficiency. The number of points in the remaining point cloud is significantly lower than that in the original. Note that the points axis is logarithmic, so each filtering method removes a vast portion of the points from the previous step.

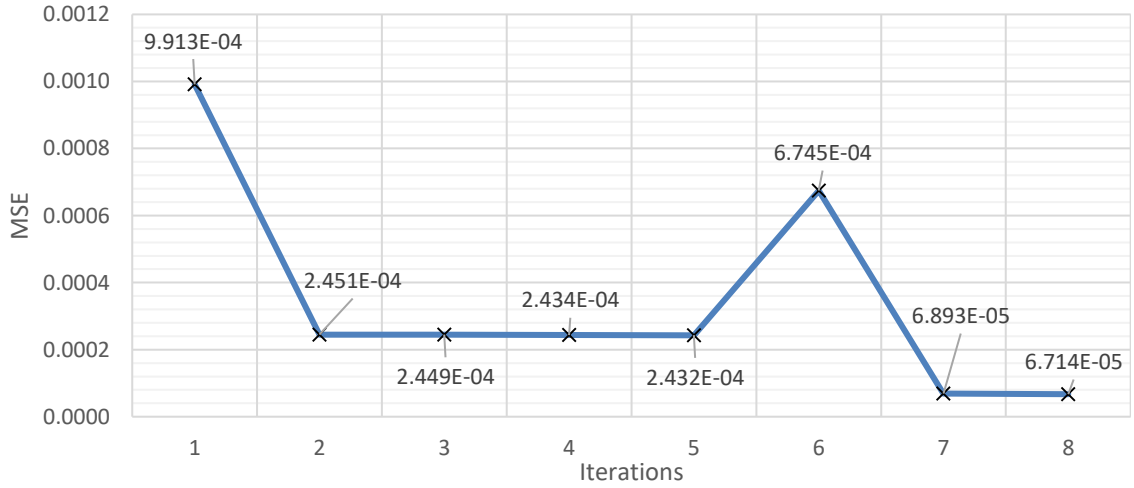

Figure 13. *Line chart of one of the ICP point cloud alignment mean-squared error*

Figure 13 shows the mean square error (MSE) obtained by combining the point clouds in each iteration. The point cloud settles at the first optimal location almost immediately after the first iteration. The error does not change much in the fifth iteration. However, the point cloud settled in the wrong position, so it had to be rotated manually. In the sixth iteration, the error increased slightly with correction, but in the following iteration rounds, the point cloud settled in place as the MSE error reached a lower level.

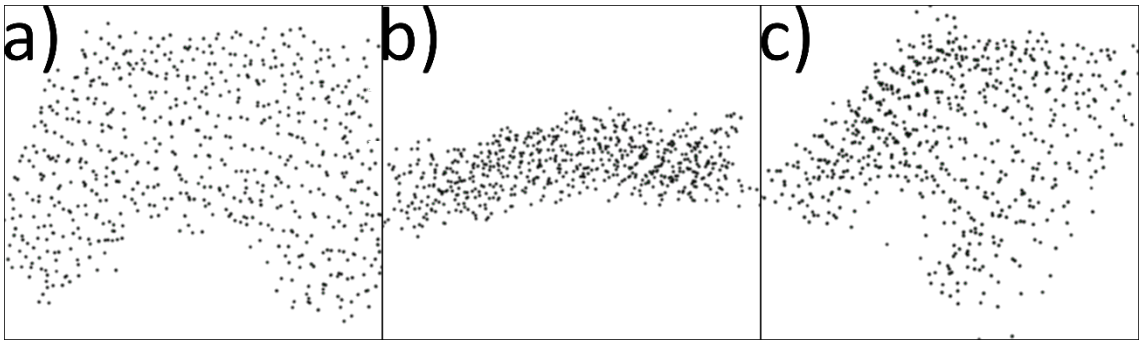

Figure 14. *Low-quality resulting point cloud from the controller. Top view (a), side view (b), perspective view (c)*

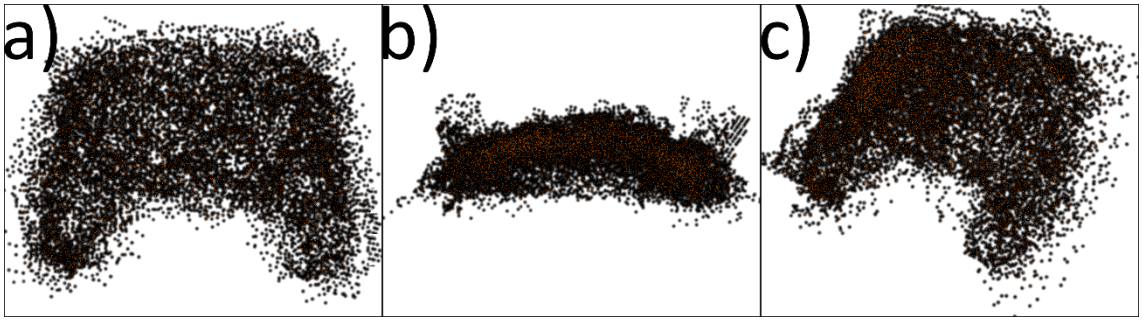

Figure 15. *High-quality resulting point cloud from the controller. Top view (a), side view (b), perspective view (c)*

The 3D scanned game controllers merged points are shown in Figures 14 and 15, showing the results from different viewpoints. In Figure 14, the standard mentioned filtering methods were used. The overall shape of the controllers can be determined but not much more. In Figure 15, the original unfiltered point data were combined. The resulting point cloud is much denser than in Figure 14. The denser point cloud has a lot of noise which scatters the points around the average surface, making the surface coarse and hard to distinguish.

Neither of the point clouds is precise. Precise surface details such as the buttons on the controller are lost among the sources of error, leaving the surface inaccurate. The errors caused by a reflective surface of the controllers can be seen in Figures 14 and 15, picture b, as outlier points near the controller's handles that do not belong to the average surface of the real object. The number of points in Figure 14 is nearly 700, and in Figure 15, almost 13 000.

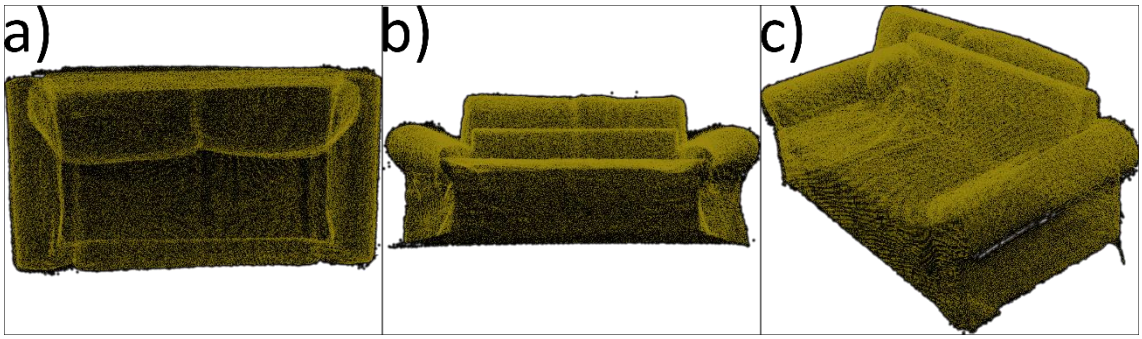

Figure 16. *Resulting point clouds from the scanned sofa. Top view (a), side view (b), perspective view (c)*

The resulting merged point cloud is shown in Figure 16 with different viewpoints. The quality is phenomenal compared to the game controller. The object's overall shape is correct, from the surface modalities to the sofa fabric's wrinkles. As the point clouds were aligned and merged by controlling the point cloud, the result has some ghosting edges, edges that did not get appropriately aligned and are in multiple distinct places in the point cloud. The ghosting cannot be seen in the pictures, as the double edges blend with the rest of the points. Another artifact is the distinct outlying points near the edges of the object remaining after the filtering. The total number of points in the resulting point cloud is about 450 000. Fewer points would have sufficed, but this goal was to keep the quality as high as possible, so the filtering was minimal.

The test result shows that these methods cannot provide an accurate, detailed point cloud as an output for smaller objects to be scanned. As with scanning a larger object, such as the sofa, the resulting point cloud, the average surface is more detailed. With the smaller object, filtering comes with lower quality but faster computing. Also, with the larger object, the quality is better with a cost of more computing time.

6. DISCUSSION

The resulting game controller point clouds do not have decent quality. There is a substantial variation in the location of the points around the average surface. The shape of the point clouds does not closely follow the variations in the form of the game controller surface, and the exact details of both point clouds are lost with the error noise. However, both point clouds are in the correct shape, so this type of 3D acquisition is better suited for larger objects or objects whose appearance needs to be modeled only roughly, and the accuracy of the surface does not matter.

Another 3D scanned object was a sofa. The quality of the resulting point cloud is good mainly because each captured view contained a lot of points that were not reduced by downsampling. Compared to the game controller, it is uncertain what causes the differences in the quality and outlier points in the point clouds. One significant factor could be the material, as the game controller is a bit reflective, and the sofa is not. Another factor might be the object's surface area. As the game controller surface area is small, the errors cumulate more clearly on combining the point clouds.

Many sources of error should be considered when an object needs to be 3D modeled with a depth sensor. If the focus is on modeling only one object, as in this work, the object must not have shiny surfaces. Otherwise, the depth sensor may not receive the light pulse transmitted at the right time. Figure 2 shows well what happens to the reflective surfaces. There is a hole in the point cloud at the point where the mirror should be. The points of the mirror itself are much farther away than they should be. It is also essential to consider the shape of the object. The object must not be symmetrical. Otherwise, point cloud alignment is unreliable when there is no reference to where the point cloud should be correctly aligned.

The object's distance from the camera is also an important source of error. The farther the object is from the camera, the fewer points in the object's point cloud, so the surface shapes are inaccurate. As the distance increases, the uncertainty of the value of the point distance also increases. Figure 2 also illustrates the distortion of the camera lens. Calibration would reduce the errors in distortion better. The camera was not calibrated separately, but the factory settings of the camera's intrinsic parameters were used. Therefore, it is unknown how much the calibration errors affect the results.

A filtering method was used that at the same time reduces the erroneous data but also increases it. The downsample filter calculates the average of a given point cloud and

reduces the number of points in the point cloud. Downsampling increases the uncertainty of the point location. This filter should be used under consideration.

The most significant error source is in ICP alignment. Based on the shape of the point cloud alone, the alignment accuracy is highly uncertain because the locations of the points taken from different angles are not in the same places.

The alignment of the point clouds would be significantly improved if the point clouds had anchor points or an anchor object. These could give the ICP an accurate preliminary guess about the target's location. The suggestion is to modify the program to correctly find the direction in which the point cloud should be moved and rotated based on the shape of the point cloud object. The tricky thing about this method is aligning a perfectly symmetric object with several correct point cloud orientations. Alignment could be improved by using external geometric objects to align the point cloud to the global coordinate system when the alignment object is in the point cloud. On the other hand, if such a change were made, ICP alignment would no longer be needed because point clouds can be moved and connected directly to the same coordinate system with a few transformations. This method could also model the entire environment as in KinectFusion, but with a different approach.

It was learned how to get from depth data to a 3D model of an object. In addition, the importance of filtering was understood. Depth data contains many useless data points when modeling only a tiny object. In addition, it was helpful to learn how to move and rotate 3D point clouds in the *XYZ* coordinate system using matrix operations.

Depth sensors can be found in modern phones that could be utilized in 3D modeling. This could be one area of research on what kind of results are obtained with phone depth data in 3D modeling. The methods used would be the same as in this work.

The methods of this work are very user-specific. Further development would be to modify the implementation so that no user intervention is required and 3D modeling could be performed automatically and even in real-time. Before that, however, it would be essential to figure out how to get more accurate point clouds by reducing the impact of errors.

7. CONCLUSION

This work investigates the steps of 3D modeling of a Steam controller and a sofa when the depth data was collected with a Kinect v2 depth sensor. Point clouds were formed from the depth image of the camera, which was filtered to contain only the points of the scanned objects. Point clouds taken from different directions were combined using an Iterative Closest Point (ICP) algorithm. The ICP program was modified so that the user could change the position of the moving point cloud by moving and rotating it. Implementation prevents the point cloud from getting stuck in the wrong place with user intervention.

The resulting point clouds did not meet expectations for the scanned game controller as the quality of the point clouds was not good. On the contrary, the quality of the scanned sofa was excellent, but a lot of time and effort was put into achieving the results to get that good quality. The most significant source of error was in the implementation of ICP in both scans. Mostly, the point clouds did not align correctly automatically, and even with manual correction, the point clouds still aligned occasionally incorrectly. The problems with the ICP could be corrected by improving point alignment with anchor points, or other anchor objects in the point cloud or a better alignment algorithm.

The code used in this work is available on GitHub[3]. The code can be modified and used freely. This work provides a reasonable basis for possible further developments, such as solving the identified problems or enhancing the current implementation. A promising idea for further development would be to get rid of user-dependent implementation and proceed to automatic and real-time modeling of the object or the entire environment.

[3] https://github.com/roopekoo/Kinect3D

REFERENCES

[1] C. C. Lai and K. L. Su, “Development of an intelligent mobile robot localization system using Kinect RGB-D mapping and neural network,” *Comput. Electr. Eng.*, vol. 67, pp. 620–628, 2018, doi: 10.1016/j.compeleceng.2016.04.018.

[2] T. Song, L. Zhou, X. Ding, and W. Yi, “3D Surface Reconstruction Based on Kinect Sensor,” *Int. J. Comput. Theory Eng.*, pp. 567–573, 2013, doi: 10.7763/IJCTE.2013.V5.751.

[3] Ying-Yuan Huang and Mei-Yung Chen, “3D object model recovery from 2D images utilizing corner detection,” 2011, pp. 76–81. doi: 10.1109/ICSSE.2011.5961877.

[4] A. Moro, E. Mumolo, and M. Nolich, “Building virtual worlds by 3d object mapping,” 2010, pp. 31–36. doi: 10.1145/1878083.1878092.

[5] Jiajun Zhu, G. Humphreys, D. Koller, S. Steuart, and Rui Wang, “Fast Omnidirectional 3D Scene Acquisition with an Array of Stereo Cameras,” 2007, pp. 217–224. doi: 10.1109/3DIM.2007.25.

[6] G. Kurillo, E. Hemingway, M.-L. Cheng, and L. Cheng, “Evaluating the Accuracy of the Azure Kinect and Kinect v2,” *Sensors*, vol. 22, no. 7, Art. no. 7, Jan. 2022, doi: 10.3390/s22072469.

[7] E. Lachat, H. Macher, T. Landes, and P. Grussenmeyer, “Assessment and Calibration of a RGB-D Camera (Kinect v2 Sensor) Towards a Potential Use for Close-Range 3D Modeling,” *Remote Sens. Basel Switz.*, vol. 7, no. 10, pp. 13070–13097, 2015, doi: 10.3390/rs71013070.

[8] F. Yang, J. Chen, L. Zhang, J. Ge, and J. Ding, “A Triangular Texture Mapping for 3D Modeling with Kinect,” 2018, pp. 55–60. doi: 10.1145/3232829.3232838.

[9] J. Smisek, M. Jancosek, and T. Pajdla, “3D with Kinect,” 2011, pp. 1154–1160. doi: 10.1109/ICCVW.2011.6130380.

[10] M. Hansard, R. Horaud, M. Amat, and G. Evangelidis, “Automatic detection of calibration grids in time-of-flight images,” *Comput. Vis. Image Underst.*, vol. 121, pp. 108–118, 2014, doi: 10.1016/j.cviu.2014.01.007.

[11] E. Lachat, H. Macher, M.-A. Mittet, T. Landes, and P. Grussenmeyer, “FIRST EXPERIENCES WITH KINECT V2 SENSOR FOR CLOSE RANGE 3D MODELLING,” *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. XL-5/W4, no. 5, pp. 93–100, 2015, doi: 10.5194/isprsarchives-XL-5-W4-93-2015.

[12] K. Khoshelham, “ACCURACY ANALYSIS OF KINECT DEPTH DATA,” *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. XXXVIII-5/W12, pp. 133–138, 2012, doi: 10.5194/isprsarchives-XXXVIII-5-W12-133-2011.

[13] R. A. Newcombe *et al.*, “KinectFusion: Real-time dense surface mapping and tracking,” 2011, pp. 127–136. doi: 10.1109/ISMAR.2011.6092378.

[14] H. Yue, W. Chen, X. Wu, and J. Liu, “Fast 3D modeling in complex environments using a single Kinect sensor,” *Opt. Lasers Eng.*, vol. 53, pp. 104–111, 2014, doi: 10.1016/j.optlaseng.2013.08.009.

[15] Qin Ye, Yahui Yao, Popo Gui, and Yi Lin, “An improved ICP algorithm for kinect point cloud registration,” 2016, pp. 2109–2114. doi: 10.1109/FSKD.2016.7603507.

[16] N. Doan, D. Pham, T. Dinh, and T. Dinh, “Restoring surfaces after removing objects in indoor 3D point clouds,” 2013, pp. 189–197. doi: 10.1145/2542050.2542088.

[17] C.-H. Teng, K.-Y. Chuo, and C.-Y. Hsieh, “Reconstructing three-dimensional models of objects using a Kinect sensor,” *Vis. Comput.*, vol. 34, no. 11, pp. 1507–1523, 2017, doi: 10.1007/s00371-017-1425-2.

[18] S. Izadi *et al.*, “KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera,” 2011, pp. 559–568. doi: 10.1145/2047196.2047270.

[19] F. Bernardini and H. Rushmeier, “The 3D Model Acquisition Pipeline,” *Comput. Graph. Forum*, vol. 21, no. 2, pp. 149–172, 2002, doi: 10.1111/1467-8659.00574.

[20] M. Santala, “3D Content Capturing and Reconstruction Using Microsoft Kinect Depth Camera,” 2012. http://www.theseus.fi/handle/10024/41621 (accessed Apr. 06, 2022).
[21] M. Fischler and R. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981, doi: 10.1145/358669.358692.
[22] J. Bentley, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975, doi: 10.1145/361002.361007.
[23] S. Rusinkiewicz and M. Levoy, “Efficient variants of the ICP algorithm,” 2001, pp. 145–152. doi: 10.1109/IM.2001.924423.
[24] P. Manojkumar and G. R. M. Reddy, “Parallel implementation of 3D modelling of indoor environment using Microsoft Kinect sensor,” 2013, pp. 1–6. doi: 10.1109/ICCCNT.2013.6726784.
[25] P. R. Evans, “Rotations and rotation matrices,” *Acta Crystallogr. D Biol. Crystallogr.*, vol. 57, no. 10, pp. 1355–1359, 2001, doi: 10.1107/S0907444901012410.