Bachelor's thesis

NTNU
Norwegian University of Science and Technology

Joar Olai Røyset Lier

Automatic Creation of 3D Printable Brace from Scan of Broken Hand

Bachelor's thesis in Automatisering og robotikk
Supervisor: Øystein Bjelland
Co-supervisor: Paul Steffen Kleppe
May 2022

NTNU
Kunnskap for en bedre verden

Joar Olai Røyset Lier

Automatic Creation of 3D Printable Brace from Scan of Broken Hand

Bachelor's thesis in Automatisering og robotikk
Supervisor: Øystein Bjelland
Co-supervisor: Paul Steffen Kleppe
May 2022

Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Acknowledgements

I would like to thank my supervisors Paul Steffen Kleppe and Øystein Bjelland.

I would also like to tank two volunteers, Mari Haram and Espen Godø that helped me when I needed an extra pair of hands.

Abstract

The goal of this project was to make an easy-to-use automatic program to 3d scan the hand, with the intention of this later being used to create a patient-specific brace for hand fractures. The result was that the program made in this report was fully automatic in creating the model of the hand. Also, a test show that it works on different people. At the end of the project, the physical system still had two issues that should be resolved. The first was a slight slack in the system, that made the forearm of the brace too big. The second were the lighting conditions. Because of the variable lighting conditions there could be places on the hand that the 3d scanner can't read.

The scanning process is that the patient holds the hand on a support in front of a depth camera, then an operator starts the scanning program and rotates the depth camera around the hand. On average one out of three scans were good. After this, the automatic data processing program is run. If the scan result is good enough it will output the 3d model. If not for the slack in the physical system, this 3d model would have been good enough to be used for create a brace.

The automated data processing program follows this procedure.
1. Turn the depth image into a point cloud.
2. Merge the different point cloud into a single coordinate system
3. Smooth the point cloud.
4. Turn the point cloud into a 3d model.
5. Sample points from the 3d model and then make a new 3d model with these points.
6. Smooth the 3d model.
7. Finally enlarge the 3d model.

Figure 1: The 3d scanner

Contents

Chapter 1

Introduction

The goal of this bachelor is to make an easy-to-use automatic program to 3d scan the hand, with the intention of later being used to create a brace for broken hands. This chapter includes the why a 3d printed plastic brace is useful, previous work and the goal and priorities of this bachelor.

1.1 Motivation

In Norway about 15 000 adults break bones in their hand yearly [8]. Today those people need to manually be plastered by a nurse. 3d scanning and then 3d printing the hand support may save time for the nurse, and this method may need less training than plastering the hand. If the hand support is made from plastic instead of plaster, then the patient does not need to worry about the plaster getting wet. The patient can therefore shower and can keep better hygiene on the broken hand. It also may be possible to remove the plastic support without destroying it, and therefore easier to examine the broken hand.

1.2 Earlier Work

In 2017, a thesis was done on the same topic. But the content was not accessible. Also there was a 2018 paper that was not accessible, but it was about NTNUs work on the 3d scanner [6]. There were two theses from 2019, both worked on the same 3d scanner, but one work more on the physical system [10] and the other worked more on software [2]. The result was a 3d scanner based on RaspberryPi cameras and photogrammetry to create a 3d model of the hand. But its resulting 3d model was too low quality to generate a brace from it. There was another thesis in 2020, which had a great in-depth summary of the useful information of the earlier theses. Please refer to this thesis for earlier work. The 2020 thesis also made the 3d scanner that this thesis is based on. The 2020 thesis tried two different depth sensors, Intel RealSense L515 and the lidar in an iPhone. The result of the iPhone was very good but the result from the L515 was not good enough for creating a brace from. They also managed to create a 3d model of a brace

from the hand. It is unknown if they printed out the brace and tested it on the one that was 3d scanned. They used a tool to create the point cloud of the hand, which might not be easy to automate. They used Siemens NX to manually create the model of a brace from the point cloud of the hand. They concluded that Siemens NX was not the right tool for an automated process [9]. Later, there was a 7-page paper in 2020 that summarized the earlier work, which included the thesis from 2020 [7]. Finally, there was a thesis in 2021, this thesis made small modification to the 3d scanner. They added a mount for the Realsense L515 and made the 3d scanner a bit sturdier. They also made a python program that can create a point cloud with the Intel Realsense L515. The position of the camera needs to be manually typed into the program for it to work.

1.3 Objectives

The aim of this bachelor is to further develop a setup for scanning the hand, and then use the 3d scan to make a plastic brace for the hand. The main challenge is to make a program that automatic process data from a 3d scanner to make it into a 3d model of the hand. The 3d model needs to be accurate enough to be used to create a brace for the patient.

This thesis has the following objectives:

- Explore cameras for 3D-scanning of hands.
- Develop software for automatic processing of 3D-scan data.
- To automatically generate brace from processed 3D-scan data.

Chapter 2

Theory

2.1 Camera

2.1.1 Pin Hole Camera Model

The pin hole camera model describes where a point in 3d space will be in a 2d image taken by a camera. This model is a simplification of how light would travel from an object to the image sensor in the camera. If one draws an infinitely long line that intersects the pinhole and the point in 3d space. Then where that line intersects the image sensor plane, decides where that point is on the image(2.1) [11]. The position and size of the image plane is decided by the camera matrix. In reality the image sensor is not perfectly flat and there is a lens between the point and the image plane, this can be partly compensated for with the cameras distortion coefficients.

2.1.2 Sub-pixel Corner Detection

The camera can only see the amount of light hitting a pixel. Therefore, if a pixel receives photons from two different surfaces, the pixel will have a color which is a mix of two different surfaces (2.2). If the color of the two surfaces is known, then it can be calculated how much of each surface is in the pixel. If the lines are straight the position of the corner can be calculated with the help of multiple pixels.

2.2 Aruco

Aruco (2.3) is a type of pattern used to get relative position and rotation of the Aruco relative to the camera. To get the position and rotation of the Aruco, the size of the Aruco and (the camera matrix and distortion coefficients) need to be known. When calculating the distance to the Aruco, the position of the corner in the image is detected, then the pinhole model is used to find the position and rotation of the Aruco. OpenCV has a function that does this, it is therefore not necessary to know how to do this manually. The pattern of the Aruco does not

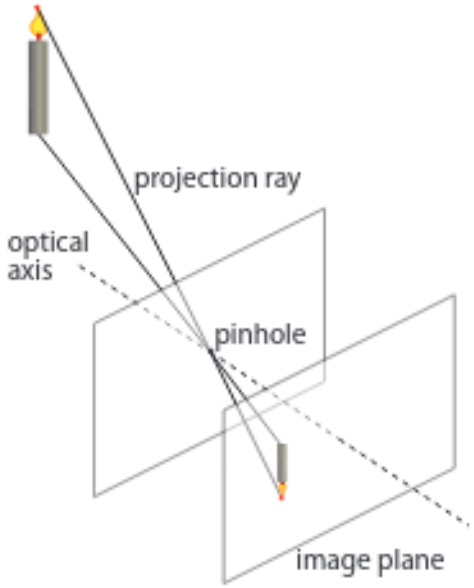
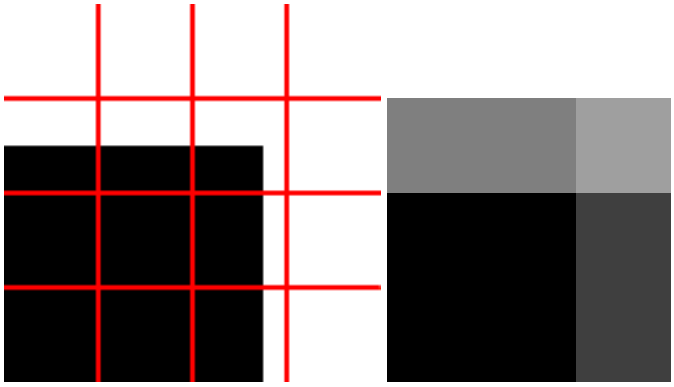


Figure 2.1: Shows the projection geometry from the pinhole model. Because the triangle between the image plane and pinhole is known, if the size or the distance to the candle is known, then the position of the candle can be calculated [11].



(a) (b)

Figure 2.2: Shows how a real corner would look like for a camera. If it is assumed that the edges are straight, this effect can be reversed. With this method the corner can be more precisely placed than on which pixel it is, and therefore it is called sub-pixel corner detection. (a) Shows what the camera tries to capture. The red line marks the cameras pixels. (b) Show what the image from the camera.

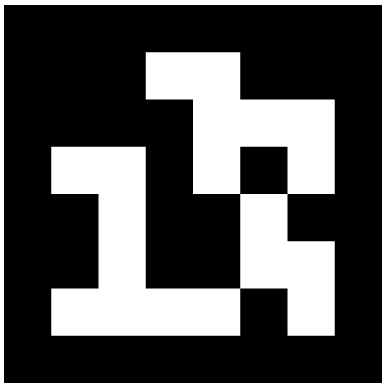


Figure 2.3: Example of an Aruco code.

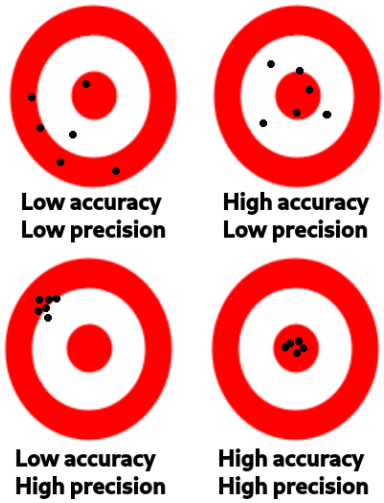


Figure 2.4: Difference between accuracy and precision.

store information. It is therefore necessary to compare a pattern of the Aruco to a list (this list is called a dictionary), and then used this list to give the Aruco an id, and to tell the software what part of the Aruco is by default upward.

2.3 Accuracy vs Precision

Accuracy is how close the average reading is to the true value of what it is reading.
Precision is how much different measurement differ, when the reading is taken in the same situation.

2.4 Other Terms

L515: short for Intel Realsense L515 which is a depth camera.
D435: short for Intel Realsense D435 which is a depth camera.

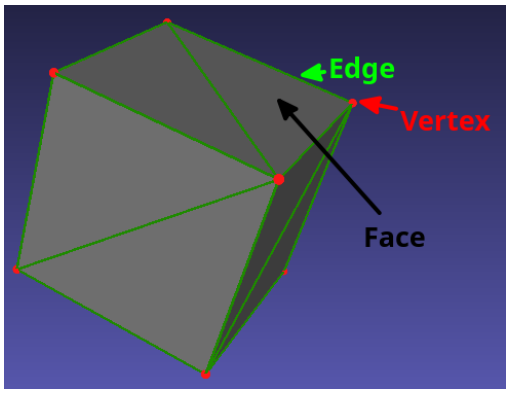


Figure 2.5: Explains face, edge and vertex. Gray triangles are the faces. Green lines are the edges. Red points are vertexes.

GiB: short for gibibyte. gibibyte is 2^{30} bytes. This is sometimes called GB, but GiB is more specific.
Point Cloud: a group of a lot of points
Stl-file: 3D mesh file format
Face: plane between the edges, often a triangle (Figure 2.5)
Edge: The end of a face/where two faces meet. (Figure 2.5)
Vertex: The end of an edge/where two edges meet. (Figure 2.5)

Chapter 3

Methods and Materials

This chapter is about the depth sensor, the method used, and the failed methods. This chapter also shows how different algorithms affects the 3d model. There were many different sequences of algorithms that were tried out on the 3d model. However, this is only shows the sequence that was used in the final product.

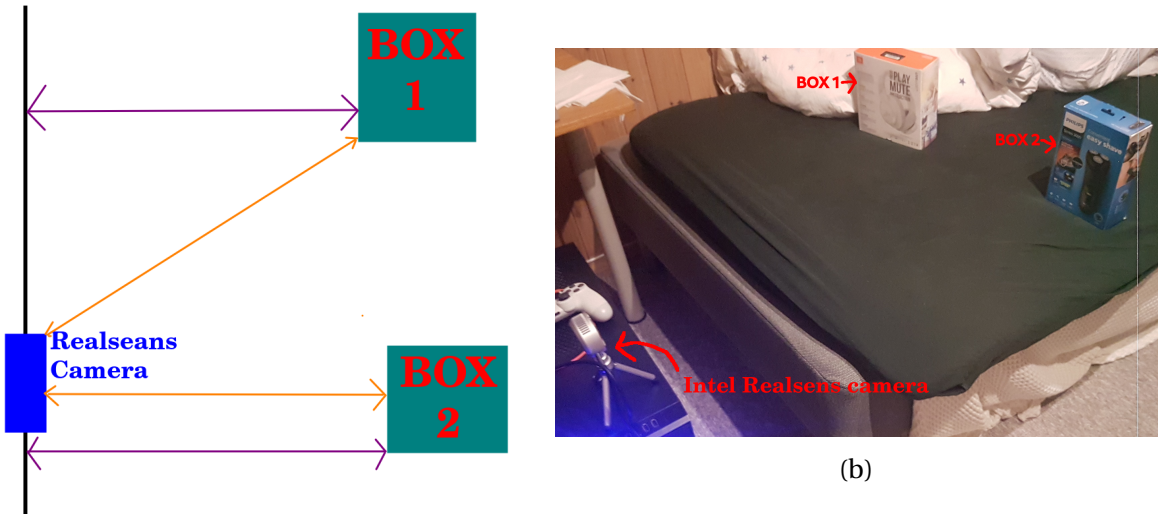
3.1 Intel Realsense

Intel Realsense is a family of products, where each product has a depth camera, rgb-camera and sometimes has an accelerometer and gyro. The depth camera seems to measure the distance between the object and an imaginary plane tangential on the camera and not the distance to the camera. Another way of looking at this is that the depth camera returns the z-coordinate and not the distance to the camera. Figure (3.1) shows this.

3.1.1 Realsense D435

Intel Realsense D435 can be used to measure the distance to objects that is very close (165 mm), and it has a 2% accuracy if the object is 2 m away[4]. D435 uses stereo vision to measure the distance, because those two cameras are not in the same space, it is possible that a surface is blocked by an object in front of one of the cameras but is visible for the other. Because one of the cameras cannot see the surface, the distance to the surface cannot be measured. This looks like a "shadow" of undefined distance from the object in front, as seen in figure (3.2), the closer the object is compared to the background, the more the shadows separate from the object.

D435 can create many spots of undefined distance, some of the undefined distance is because of "shadows" but other parts are affected by lighting condition and the angle of the surface compared to the camera. Black spots because of tilt are shown in figure (3.3), and black spots from lighting condition can be implied from the result chapter, when comparing good and bad lighting condition. D435 has many modes, but there are two modes used in this project. One is the default, this gives good visibility, but the data can be a bit noisy. The other mode is accuracy



(a) (b)

Figure 3.1: (a) Model of output of Realsense camera. The blur rectangle is the Realsense camera, and it is pointing toward box 2. (b) Shows the real test that was modeled in (a). Box 2 is closer to the camera than box 1, but from testing this scenario the output from the Realsens tells that box 1 is closer. The hypothesis is that the distance is the distance between the box and an imaginary plane tangential on the camera (black line in the picture). Another way of looking at this is that the output is the z coordinate in a Cartesian coordinate system, and not the distance to the camera. The result of the experiment fits this hypothesis. Conclusion: the output of the distance camera in Realsense is the length of the purple line and not the orange line.

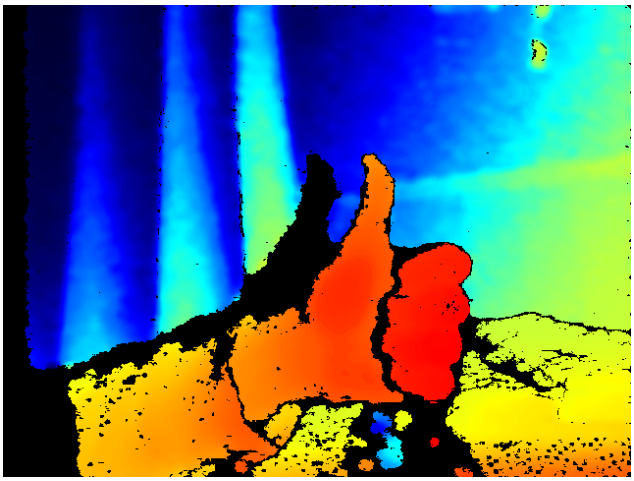


Figure 3.2: D435 distance image of a hand with thump up (black = undefined, red = close, blue = far) This image show that object creates an undefined "shadow", this shadow is more noticeable if the distance between an object and the object behind it is large (as seen from the thumb and the black “shadow” of the thumb, but the shadow of other finger (in the middle of the hand) is very small

mode. This gives low noise data, but it is extremely sensitive to lighting condition, and if the lighting condition is not good enough then it returns undefined distance.

3.1.2 Realsense L515

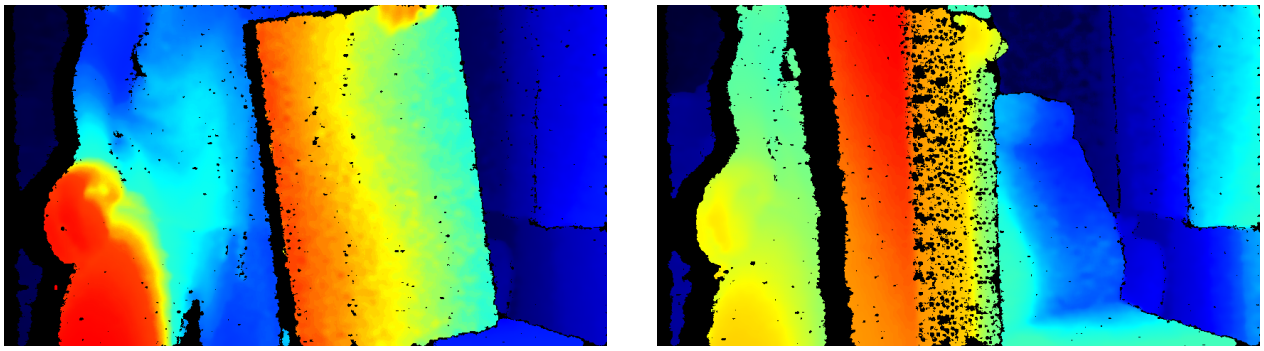
L515 is a lidar, it has a minimum distance of 0.25 m. It has an inaccuracy of 5 mm at 1 m distance [3]. This inaccuracy is quite flat, even if the distance increased to 9 times the length (9m) then the inaccuracy would only be increased to 3 times the size (14mm). If an object is almost in front of another object, then the object behind it will appear closer than it really is. Also, if there is an object close to the camera, then this can crate spots on the image that has an undefined distance, this is shown in figure (3.5). L515 creates a clean image that looks like it has little noise, but just because it is clean does not make it accurate. L515 creates an image that is more precise, but less accurate than D435.

3.2 Camera Calibration

When calibrating the camera, we want to find the camera matrix and the distortion coefficients of camera.

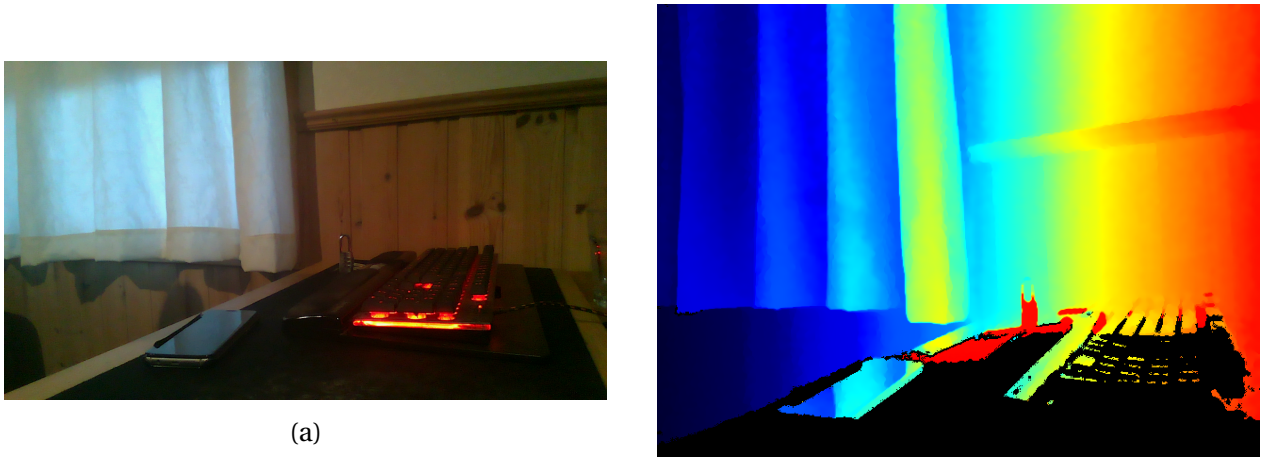
3.2.1 Potential Depth Camera Calibration

In the end the depth camera was not calibrated. This affected the reasoning for later decisions. When calibrating an RGB camera, one usually does it by showing the camera a known image



(a) (b)

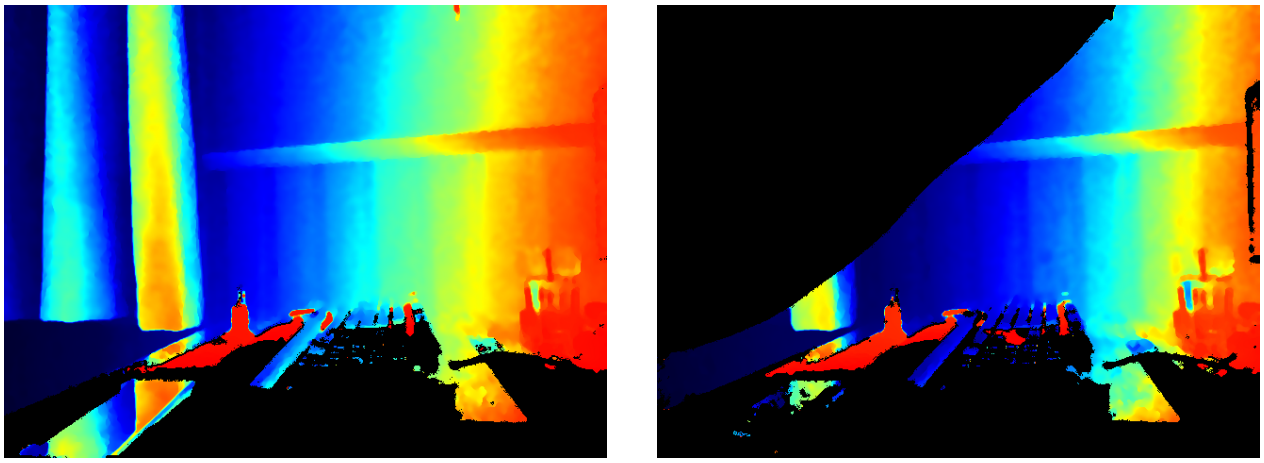
Figure 3.3: D435 distance image of a plate (black = undefined, red = close, blue = far)
(a) The plate is facing the camera, and therefore the image is good. (b) The plate is facing away from the camera, and therefore there are a lot of undefined spaces on the plane



(a)

(b)

Figure 3.4: (a) is a picture of the room, (b) is the distance camera from L515 (black = undefined, red = close, blue=far), (a) was not taken by L515 in the same place as (b). This shows that glossy surfaces that are facing away form the camera, act as a mirror for L515, (as can be seen in the phone and the keyboard



(a) (b)

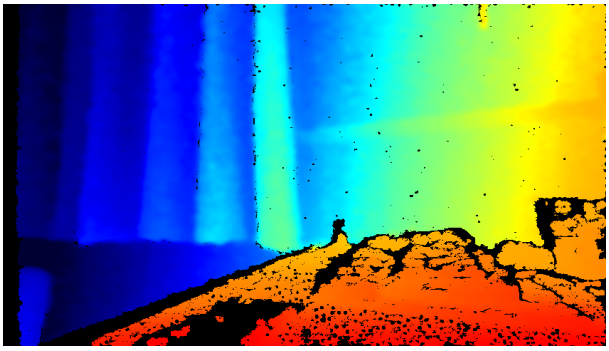
Figure 3.5: (a) and (b) are distance images of L515 (black = undefined, red = near, blue = far). Both pictures are taken from the exact same position, the only difference is that in (b) an object was placed in the top left corner, very close to the sensor. This shows that having an object very close in front of L515 will create undefined places other places in the image. This effect is strongest on glossy surfaces, but it can also happen to other surfaces.

with black and with pattern. This method is hard to do with a depth camera because the depth camera only sees depth. The image would look like a plane, and it is hard to calibrate a camera with a plane. One way to solve this is by creating holes in a plane where the image is black, then the depth camera can see the image. The problem with this method is that if the plane is not infinitely thin then the holes is going to seem smaller to the depth camera than they really are (because the camera is not going to believe the wall of the hole is part of the hole). Those tiny changes can make the calibration so bad that it becomes unusable. One way to get an extremely thin plane is to cut holes in a paper, and then fasten the paper in a frame. If the paper needs more support to be flat, the paper can be glued to plastic foil, this is because Depth camera can see though plastic foil. (At least L515, but D435 is vision based, and therefore should be able to see through the plastic foil, but this has not been tested.)

3.2.2 RGB Camera Calibration

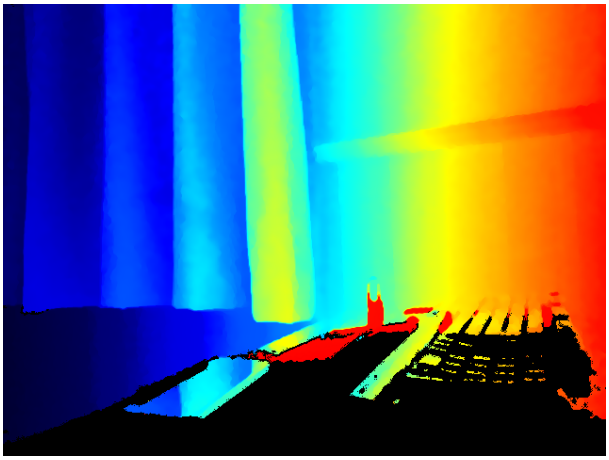
At first circle patterns were used to calibrate the camera. This was because if the circle was cut out of the paper, then the paper would be one piece and could therefore been used to calibrate the depth camera. After testing it was concluded that this pattern gave such bad calibration that it was unusable. Therefore, a chessboard pattern was used instead. The calibration patter is shown in figure (3.7b).

(a)



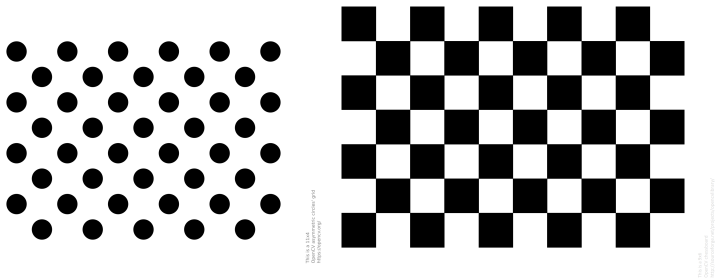
(b)

(c)



(d)

Figure 3.6: Comparison between D435 and L515. (a) and (b) are Realsense D435 and were taken from the same position, (c) and (d) is realsense L515 and were also taken from the same position. D435 and L515 were placed approximately the same distance away from the keyboard. The left images are using the RGB cameras and on the right the colormap of the distance (black = undefined, red = close, blue=far). This shows that the distance measurement has a lot bigger field of view than the RGB camera in both D435 and L515. L515 gets a clearer distance image, that is better at detecting shapes.



(a) (b)

Figure 3.7: Two images that can be used to calibrate the camera, (a) did not work, but (b) did work well.

3.3 Software

Intel realsense SDK 2 was necessary to use realsense camera. Also, OpenCV was necessary to get information needed out of the pictures.

3.3.1 Programs and Libraries

The things that are needed by the PC's are

- The PC that is connected to Realsense camera needs a USB port with speed as high or higher than USB 3.0.
- The PC that runs the code created in this bachelor has only been tested on Linux, but if the paths in the program change, then it might work on windows.
- Meshmixer is only supported on windows.

Processor	OS	Usecase
Intel Pentium 4417U	Arch Linux 5.16.15	Used to take the depth images of librealsense. It was used because it was the only laptop with at least USB 3.0, because the realsense camera needs at least that much bandwidth.
AMD Ryzen 9 3900X	Arch Linux 5.17.5	Processes the data.
Intel Pentium 4417U	Windows 10 21H1	Using Meshmixer to finalize the brace and using PrucaSliser to go from STL to step for the 3d printer.

Table 3.1: Those PC was used in this project

Programe	Versjon	Usecase
MeshLab	MeshLab 2022.02	Visualize point cloud and 3d models and merge the 2 planes of the brace into one .stl file
Meshmixer	3.5.474	Cut up the brace and make the 3d model easier for Prusasliser to work with
PrusaSlicer	2.4.2	Turn .stl (a 3d file) into .step file with 3d printers uses to 3d print an object

Table 3.2: 3d party program used in this project

Language	Version	Use case
c++	c++17	The programming language mainly used to program in this bachelor
Python	3.10.4	Made a program in python to visualize point cloud before I started using MeshLab for this task

Table 3.3: Programming language used in this project

Library	Version	Use case
OpenCV	4.4.5	Used to read, manipulate, and display images and videos, and therefore helped with calculate position of the camera, and getting from depth image to point cloud. It is also a dependency of librealsense. It was also used for Aruco and camera calibration.
librealsense	2.50.0	Get inputs from intel realsense cameras
CGAL	5.4	Remove outlier points, point smoothing, finding normals, point cloud to mesh, smooth mesh.
Boost	1.78.0	Dependensies of cgal
Eigen	3.4.0	Dependensies of cgal
mpft	4.1.0.p13	Dependensies of cgal
VCGlib	VCGLib 2022.02	Was used to sample points from 3d mesh
Eigen*	3.3.9**	Dependensies of VCGlib*

Table 3.4: Programming libraries used in this project
* When downloading VCGlib, the download includes this Eigen library. ** 3.4.0 seems to work when tested it. But just to be sure the version included with the library was used.

3.4 Get Camera Position

If the global position and rotation of the camera is always known it is easy to merge multiple depth scans in the same coordinate system, because all that is needed is to rotate the scan equal to the cameras rotation, and then set an offset equal to the camera position, and then the 3d scan can be inserted into a global coordinate system.

Because of how easy it is to merge multiple 3d scans in a single coordinate system if the position and rotation is known, it was first attempted to get the global position of the camera.

3.4.1 Get Position with Aruco

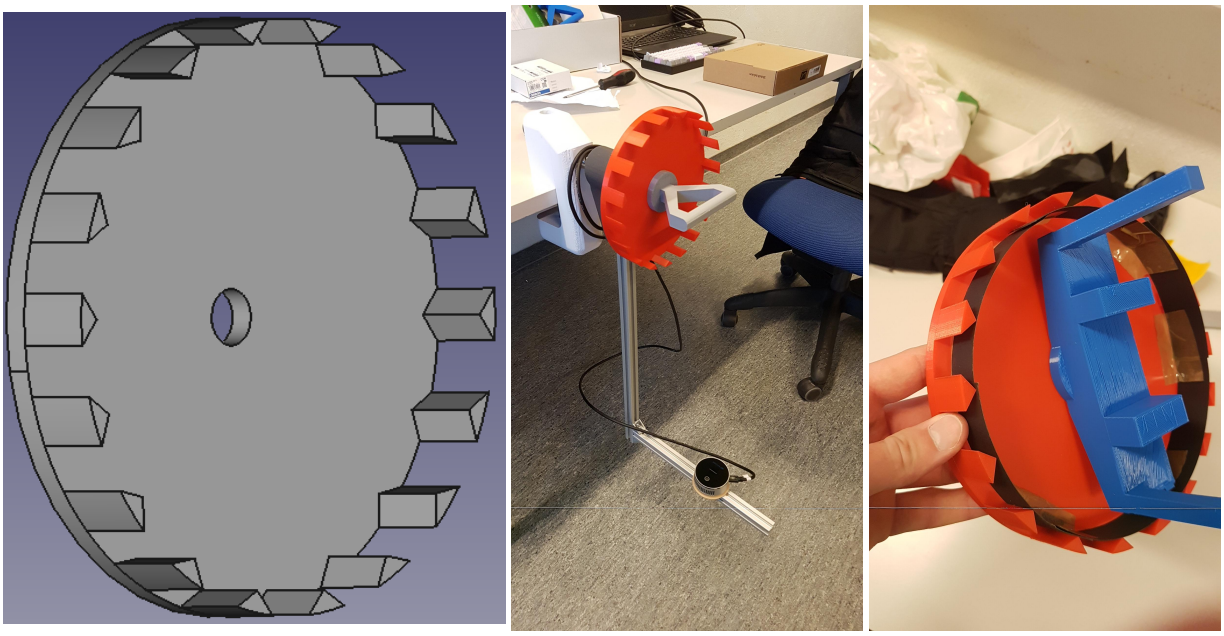
The position from Aruco is obtained with the two OpenCV functions cv::aruco::detectMarkers, which detects the Aruco code, and cv::aruco::estimatePoseSingleMarkers, which calculates the distance the position and rotation of the Aruco code. The position vector (called translation vector) returned from OpenCV, is the cartesian coordinate, where high x means left in the image, high y means low in the image and high Z means far away from the xy-plane that intersects the camera. The rotation vector returned from OpenCV is a Rodrigues rotation vector, where the length of the vector is the amount of rotation(radian) around the unit vector.

It is important that the aruco is glued into a solid plate, and it is also important to have a big margin between the end of the aruco and the end of the paper. This is because a slight curve in the aruco, will make the software calculate a slightly wrong position and a very wrong rotation of the aruco.

3.4.2 Calculate Position with Gyro

A few of the Realsenses sensors have an IMU (E.G. L515), the IMU includes both a gyroscope and an accelerometer. One way is to use the angle of the camera and the radius of the circular path that the camera takes to calculate the position of the camera. The angle was mainly going to be obtained from the gyroscope, however over time the gyroscope can start to drift. To help prevent the drift the accelerometer could be used to find a general direction of gravity and therefore the general direction of down.

I was unable to read the gyro and accelerometer data from the L515, and therefore unable to use that data to calculate the position of the sensor. The reason that I was unable could be because of the operative system that was used, $Linux \rightarrow Arch \rightarrow KDE$ was used. A few forums from the internet gave the impression that the API for accessing the IMU data was not complete, and it might be complete when other continue this project. Forums also showed that people (with unknow operative system) managed to get the IMU data, it is therefore possible to get the IMU data.

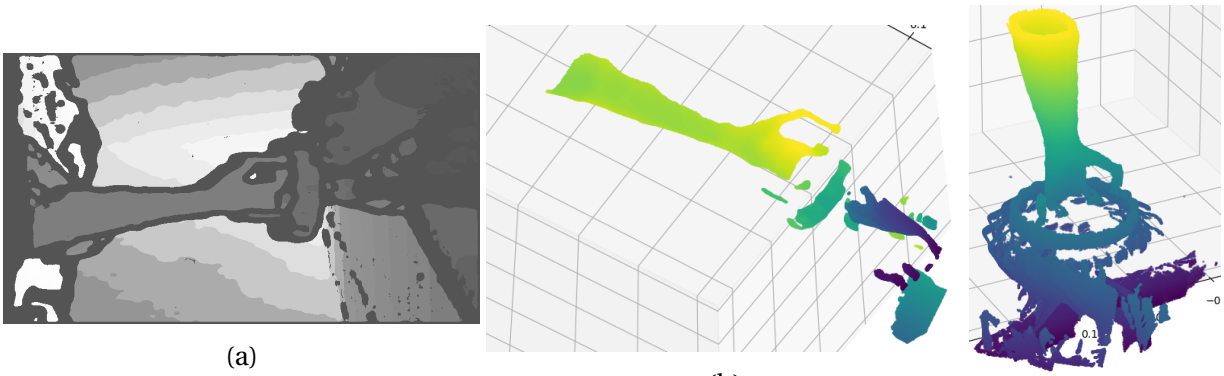


(a) (b) (c)

Figure 3.8: This shows the object used to get to get the position of the camera. The camera counts the pillar and uses that instead of a potentiometer, to find the position of the camera. (a) is the 3d model of the object (b) the object mounted in a real system (c) the object with a black piece of paper, so that it is easier for rgb-camera to count the pillars

3.4.3 Get Position Relative to a Known Object

If one manages to get the camera position relative to a stationary object it is also a way to get the position of the camera. The stationary object that was used was a disk with a pillar sticking up every 20 degrees around the disc, which is shown in figure (3.8). Then the program counted pillars instead of using a potentiometer. The program got the position by checking if a pixel sees the pillar, and when it changes from seeing the pillar to not seeing the pillar, it can be assumed that the camera has moved 20 degrees. And because the camera rotates around this object with a known radius, the position and direction of the camera can be known. If the camera sometimes moves back a short distance before going forward again, then the software might count a pillar two times and therefore believe that the camera is 20 degrees farther ahead than it really is. With L515 the depth camera was used to check if a pixel was on the pillar or not. The D435 depth camera is bad at seeing edges, and therefore the RGB camera was used and a black ring was placed on the inside of the pillars, see figure (3.8). The RGB camera output is highly affected by the lighting condition, and it is therefore important to have even lighting condition around the disk, to eliminating false positives/negatives. This method was used.



(a) (b) (c)

Figure 3.9: shows before the result of the different steps in creating a point cloud (a) a grayscale depth image, normally the image is too dark for human to see, and in order to make it visible a lot of the data has disappeared. The drakes gray part means undefined distance. (b) is the point cloud created from (a) (c) is all the point cloud meshed into a single file

3.4.4 Hybrid

Mutiple methods of getting position might be used together to fill in each other weaknesses. A combination that would have worked well would be combining the plate with pilar that was discussed in the last section and aruco code. Because then the program would be able to distinguish between pillars and remove a lot of false positives. And it might be more accurate than just using aruco code.

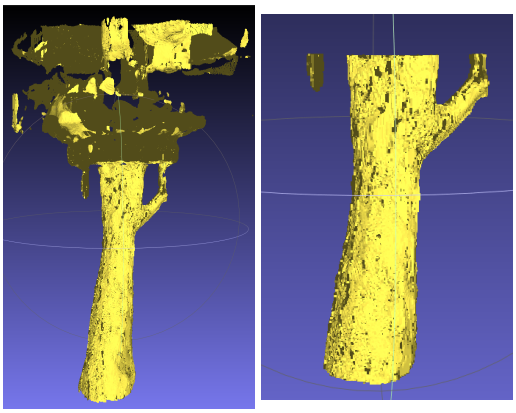
3.5 Depth Images to Point Cloud

The depth camera delivers a depth image. With image can be made into a point cloud using the pinhole model, the result of which can be seen in figure (3.9). The realsense library includes a function to do this for you, but because it was believed that the L515 (which was used under the development of this part of the code) that was used had a bit higher FOV than the average L515. And because it was believed that the function included in the realsense library assumed that the FOV was the average FOV of L515, and therefore the homemade function would be more accurate than the realsense version.

To stitch the local point cloud into a global point cloud, there was needed a way to translate local coordinate system into the global coordinate system. To do this, three functions were created: xGlob(xLocal, yLocal, zLocal), yGlob(xLocal, yLocal, zLocal). zGlob(xLocal, yLocal, zLocal)

$$xGlob(x,y,z) = a \cdot [x,y,z] + camPos$$

Where a is a vector where each of the 3 elements is [1,0,0] $\cdot$ normal vector of x, y or z in the old coordinate system into the new one. and $camPos$ is the position of the depth camera (aka, (0,0,0) in the local coordinate system) in the global coordinate system.



(a) (b)

Figure 3.10: Shows before and after removing unwanted points. (a) before (b) after

3.5.1 Point Cloud to Brace

Out of the sequences explored, this sequence seems to work best. But it is possible to do things in different sequences.

1. Remove Points

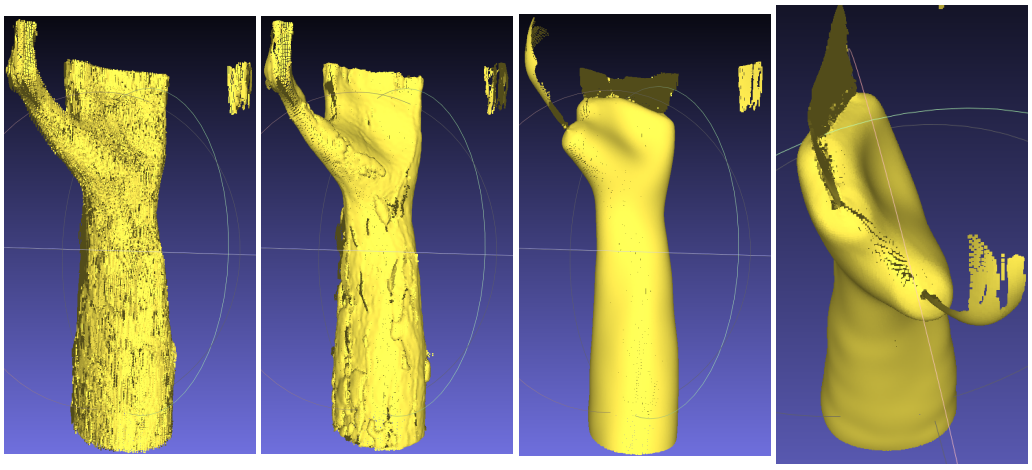
First it is important to remove the points that are not part of the hand, because those points may affect the algorithms that are later used. It also reduces computation time. This is done by removing every point that is not in the place we would expect the hand to be. This is done by removing every point outside of 2 cylinder, a small cylinder around the hand and a big cylinder around the arm. The result of which can be seen in figure (3.10).

If the point cloud is too noisy, then some of the misreading by the camera can be removed by removing points where the concentration of points is not dense enough. Because the point cloud was good enough, this was not done in the final solution.

2. Smooth Point Cloud.

Jet smooth was used to smooth the point cloud. When merging the different layer point cloud in the previous section, it is normal that the different scan slightly disagrees where a part of the hand is. The result of this is that the merged point cloud gets multiple layers. Smoothing the point cloud usually does not make the figure shrink, (in contrast to smoothing the mesh). Therefore, this is a good place to try to merge different layers.

When using the jet smooth algorithm there is mainly one variable that needs to be tuned to get the best result. This variable is how many different neighboring points are going to be considered when smoothing a point. When deciding how many points should be considered when smoothing a point, there are two things that should be considered. One is that it should be enough to merge the different layers. The other thing is that if too many points are considered, then it will merge the top and the bottom of the hand into a single plane, which is bad. This is



(a) (b) (c) (d)

Figure 3.11: Shows before and after smoothing, and what happens if over-smoothing. (a) no smoothing (b) the smoothing used in final product (c) and (d) are the same over-smoothing point cloud from two different viewpoints, one can see that the hand has become a plane.

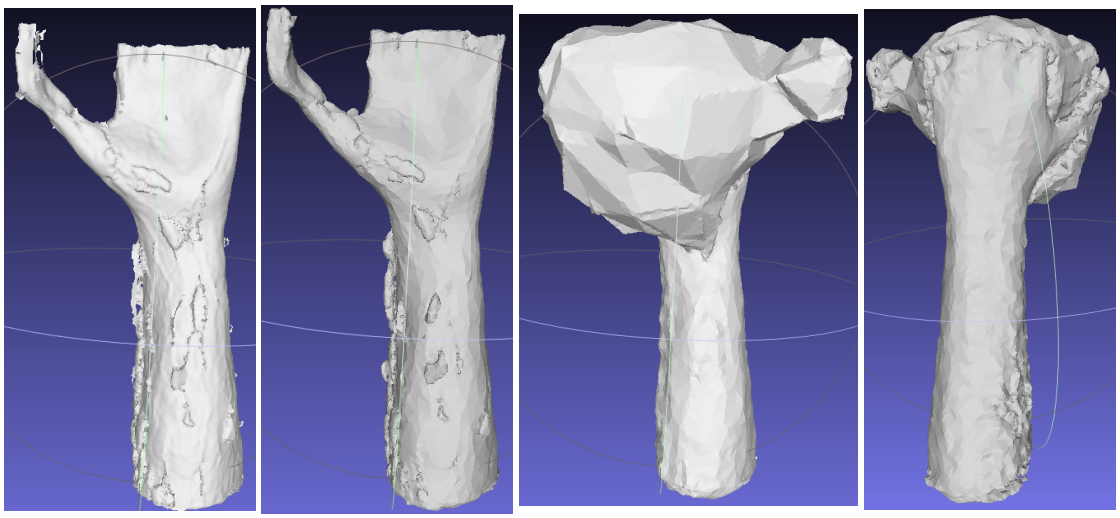
shown (c) and (d) in figure(3.11. If the point cloud is not smooth enough after a single smooth, then the algorithm can be run multiple times on the same point cloud. An example result of this algorithm is shown in figure (3.11).

3. Point to Mesh

There are many ways to transform a point cloud into a mesh. There were two successful ways that was attempted in this project. The two ways were Scale Space Surface Reconstruction[12] and Poisson reconstruct[5]. Poisson reconstruct was used in the final product.

Scale Space Surface Reconstruction is a algorithm that first smoothens the point cloud and then uses the points as vertexes. The advantages with this is that the shape created is very similar to the point cloud, and it is simple to use. The disadvantage is that even if it looks like a single 3d model it is actually multiple smaller 3d models, which can be problematic in some algorithms.

Poisson reconstruction uses a combination of the normal and position of the point to create the 3d model. The advantage with this is that the 3d model is often smooth and there becomes only one 3d model. The disadvantage is that it is more complex and if it is not rightly tuned then it can create models that do not look like the point cloud at all. There also exist shapes that if it is included, will deform the 3d model and make it different than the point cloud. One such shape is the small indent from the end of the shirt to the skin. This indent will make the model of the hand too small and model of the shirt too big. When tuning this algorithm, the most important step is calculating the normals. When calculating normals, X number of neighbors of a point is used to calculate its normal. The higher X is the smoother the 3d model, but it has the same limitation as when smoothing the point cloud, which is if X is too large, it is going to believe that the thumb or the hand is a plane, and therefore create a 3d model that does not look like the hand at all. If it believes that the thumb is a plane, it might create a 3d model with a realistic



(a) (b) (c) (d)

Figure 3.12: Shows result of turning point cloud into mesh. (a) Scale space surface reconstruction. (b) The Poisson reconstruct used in the final product. (c) and (d) the same Poisson reconstruction, but it used too many points to calculate normals.

looking thumb, but the thumb will not be in the right position.

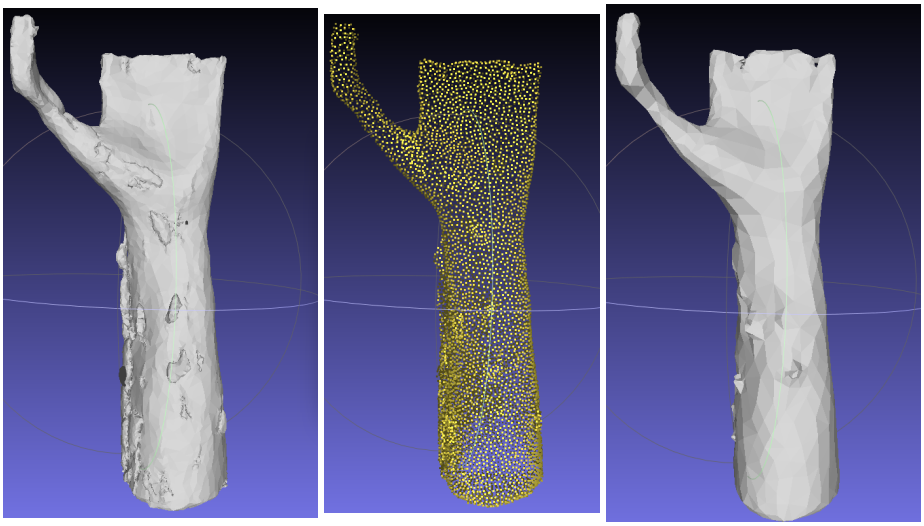
Examples of the algorithm described in this section are shown in figure (3.12).

4. Sampling and Reconstruction.

The 3d-model can have vertical pieces sticking up, or small indented areas. Sampling and then re-meshing sometime remove those pieces as shown in figure (3.13). Smoothing the mesh can also be used for this, but could deform the mesh more. Sampling and reconstruction can also help with filling holes in mesh and merging multiple surfaces lying on top of each other, but if the 3d scan is good, the 3d model should not have those problems at this point. Poisson sampling was used to sample the 3d model, and then normals were calculated and then re-meshed with Poisson reconstruction. When reconstructing it is important to use a reconstruction algorithm that only creates one object because the algorithms used later work better with one object.

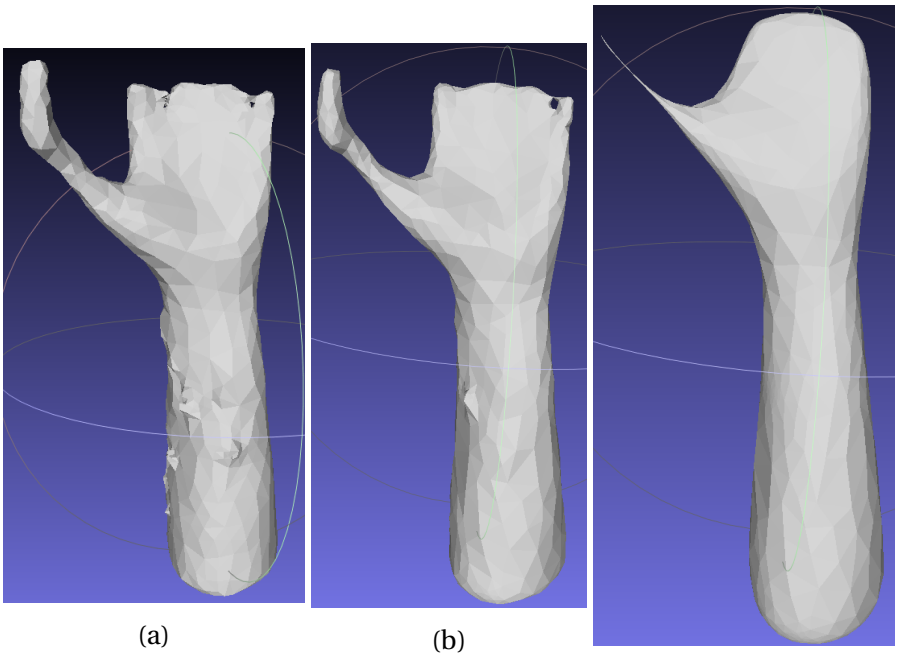
5. Shape Smooth

Mesh smoothing also removes vertical pieces sticking up, but it is more consistent than sampling and reconstruction. Unfortunately, it also shrinks the 3d scan. It also makes the 3d model smoother, which is important in the next section. Mesh smoothing has one main parameter the smoothing constant controls how big defects it can smooth out, but if this value is high the 3d model will shrink a lot. This is especially visible for the thumb in figure (3.14). If one wants to get the model smoother without the 3d model shrinking too much, one can run the algorithm multiple times with a lower smoothing constant.



(a) (b) (c)

Figure 3.13: Shows sampling and reconstruction. (a) before sampling (b) sampling (c) reconstruction.



(a) (b) (c)

Figure 3.14: Shows mesh smoothing (a) before smoothing. (b) the smoothing used in the final product. (c) over-smoothed model.

6. Surface Offset

For an unknown reason the initial point cloud from the 3d scan is smaller than the hand. The too small point cloud shrinks even further when the data is processed. Not all the shrinking is gotten in "5. shape smooth" and it seems that it shrinks more on the side of the arm, than on the top and bottom of the arm. Because of all this the 3d model needs to get bigger to fit the hand that got 3d scanned. A way to do this is to create an offset on the surface. The 3d model needs to grow more on the side of the hand than the top and bottom.

The way the offset works is by calculating the normal of a vertex, and then moving that point in the direction of the normal. The length the point moves is dependent on the direction the normal is pointing.

$$distance = a + b \cdot y^2_{normal}$$

Where a is the minimum distance a point should move, b is the extra distance points with vector in y direction should move, and the result of this is shown in figure (3.16).
The reason that we need to move points further if they are pointing in the y-direction is because the y-direction is the direction of the side of the hand. And the side of the hand has shrunk more than the rest of the hand. The reasons that y^2 was used was because, it was known that using $abs(y)$ would make the normal that pushed the point furthest in the x direction was not [1,0,0]. Figure (3.15) shows that this is also the case for y^2, but this was not known before it was too late.

7. Create Shell Around the Hand

To create the support for the broken hand, it is needed to create a shell around the hand. This is done by making a new surface which is offset a distance from the hands surface. Then invert the faces of the original surface, and then merge the new surface with the inverted hand surfaces.

Creating the two surfaces is done in the same program that makes the hand bigger (see section “6. surface offset”). But inverting the faces of the inner surface and merging the two layers has not been automated jet.

The two not automated steps are done manually in MeshLab, the exact way is shown in Appendix (1).

3.5.2 Cutting and Printing the Brace

This part is outside the scope of this thesis, but it was done to test how well the brace fits the hand. First the unwanted part of the 3d model was cut out, and then the 3d model was cut vertically in half so that it would be possible to place the brace around the hand. First the brace 3d model had to be sent to a windows machine, and then with Meshmixer cut away unwanted parts of the 3d model, and then cut the model in two so that it can be equipped. Those steps are shown in Appendix (2).

Finally, the .stl file was turned into a 3d printable file format and then 3d print the brace. This was done with PrusaSlicer.

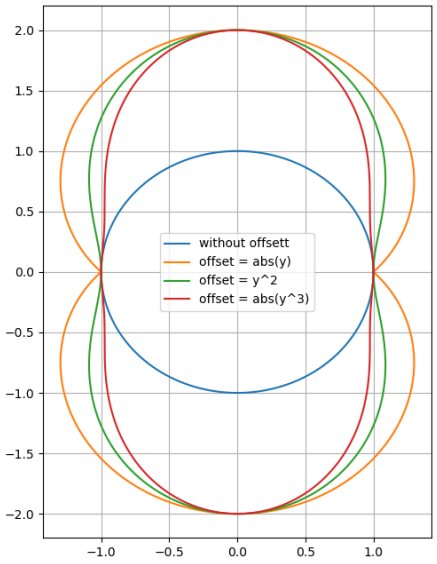
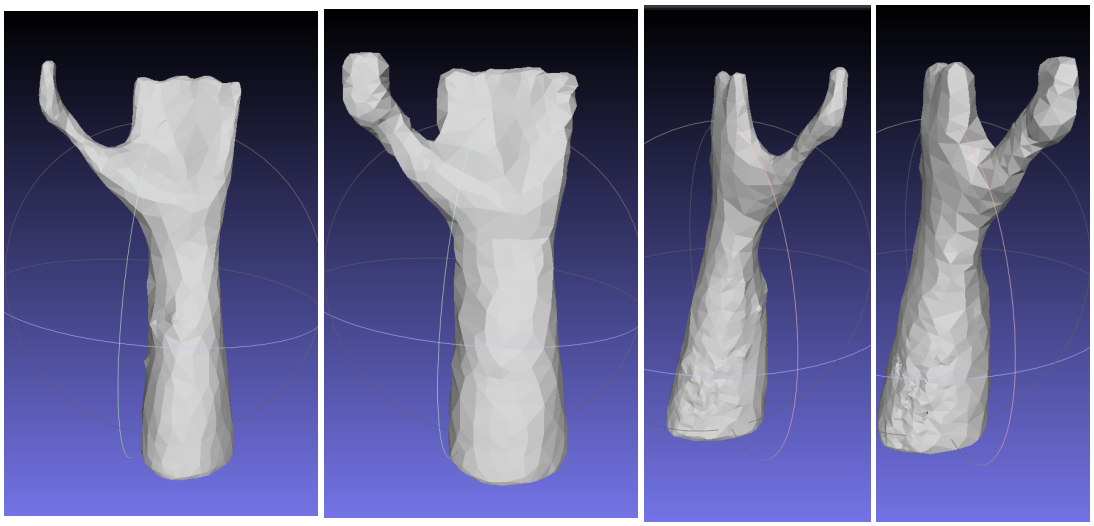


Figure 3.15: This shows that if it wanted to make a shape taller but not wider, and it is desired to do this by moving each point in the direction of its normal. Then moving the point equal to $|y_{part\_of\_normal}|^3$. This is useful to know because the hand needs to expand more in its width than height.
Blue: is without offset, orange is with offset = y value of its normal, green is offset y^2, red is offset y^3.
This graph shows that offset y and y^2 makes the furthest left/right part of the circle no longer the furthest left/right of the circle. but this does not happen when using y^3



(a) (b) (c) (d)

Figure 3.16: Shows mesh smoothing (a) before smoothing. (b) the smoothing used in the final product. (c) over-smoothed model.

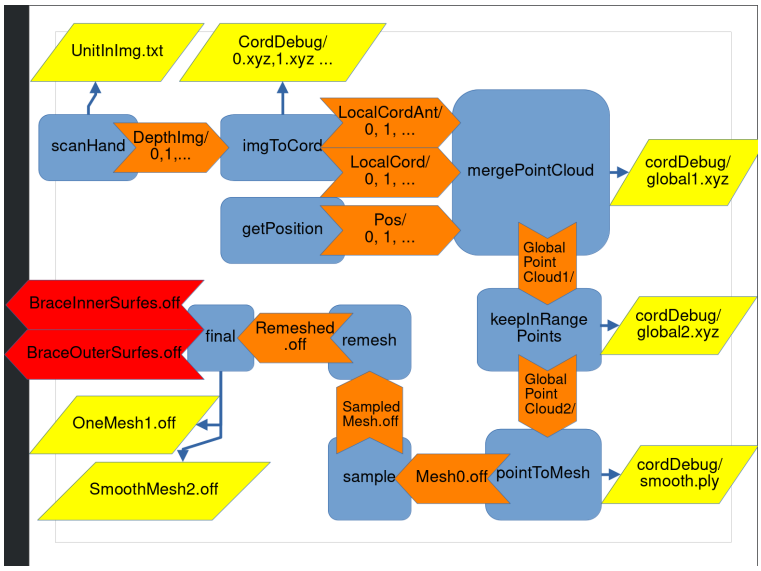


Figure 3.17: Show the automated software pipeline. The blue square is the different programs. The orange arrows are the file, that is written by the program at the start of the arrow and read by the program at the end of the arrow. The red arrow is the end product. And the yellow parallelograms are files saved for debugging purposes. What each script does is shown in the table 3.5.

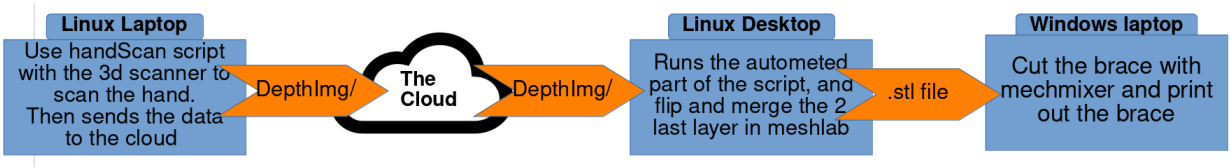


Figure 3.18: Show the workflow from the 3d scan to brace.

3.5.3 System Implementation

The automated part of the software is a bundle of 9 smaller scripts, the diagram is shown in figure (3.17). Each script writes its output to a file. Then the next uses that file as an input. This method has the advantage that, if it is discovered that there is a mistake in one of the scripts. Then only the faulty script and the subsequent scripts need to run, and not the entire program. Another advantage is that not all the scripts need to be run on the same computer. This is used in this project to 3d scan on one computer and process the data on another, this is shown in figure(3.18). It also allows for easier changing the sequence the scripts are run at. This is because only the name for input/output file needs to change. The files are stored in the "/tmp" folder in Linux. Everything in this directory is only stored in ram. Therefore, it takes a very short time to read/write the files and they does not wear out the SSD/Hard drive. The entire program took 1 minute and 12 seconds on an AMD Ryzen 9 3900X processor.

Script	Ssecase
scanHand	Save a depth image of the hand when a pillar(see figure 3.8) enter or leaves a pixel.
imgToCord	Turns the depth images into point clouds.
getPosition	Calculate the position of the images.
mergePointCloud	Transform the coordinate system of each local coordinate system into a global one, and then merge the point clouds into a single point cloud.
keepInRangePoints	Removes all the points that are not in range.
pointToMesh	Smoothen the point cloud, then calculate normals and then turn the point cloud into a mesh with Poisson surface reconstruction.
sample	Sampling the mesh. A lot of this code is a copy paste from [1].
remesh	Calculate normals and then turn the point cloud into a mesh with Poisson surface reconstruction.
final	First it removes all but the largest mesh*. Then it smoothens the mesh. Then tries to detect and fill holes* and then it makes two new meshes with an offset of the earlier mesh.

Table 3.5: Shows what the different scripts does.
* Removes all but the larges mesh and hole fill are no longer necessary, because the mesh does not have holes and Poisson reconstruction only create a single mesh. It was useful in earlier version of the code. The code had yet to be removed before the latest test. Did not want to have untested code and therefore it is still in the code.

Workflow and Hardware

When worked on this project I had only access to a slow laptop with an Intel Pentium 4417U, and a fast desktop at home with AMD Ryzen 9 3900X. Needed to do the 3d scanning in the school. Therefore, the laptop was used for the 3d scanning, but the laptop was too slow to process the data. Therefore, the 3d scanning data was moved to the desktop to be processed. Meshmixer is only supported on windows. The laptop used to 3d scan can dual-boot between Windows and Linux and was therefore used to cut the brace with Meshmixer and print out the brace. A diagram of this is shown in figure (3.18).

Chapter 4

Result

This chapter includes the results of the project. The final result before major method changes was shown. However, this chapter does not show the result of minor changes or tuning of the process. It also does not show the result of the different steps from point cloud to smooth model of the hand (that is in "material and method" chapter). This chapter shows how good the physical system is, how good it is to get an accurate point cloud of the hand, how good it is to create a smooth hand model from the point cloud, and how good the brace fits the hand that was 3d scanned.

There were a lot of failed attempts and a lot of tuning before getting the final result. There were 59 saved 3d scans, and probably 3 times as many unsaved scans as saved scans. The 59 saved scans include 6.2 GiB of data in total. From those scans a lot of time was used processed the data, and many 3d models were created. From those 3d models a small percentage was printed out, and figure (4.1) shows almost all of them. There were at least 36 different scripts made, and some of those scripts have been modified many times in tuning.

4.1 Physical System

4.1.1 Physical System Passed Down from Earlier Projects

The system gotten from earlier bachelor had an unwanted slack in structure from loose tolerances resulted in a misalignment, referred to as unwanted tilt. The system could be tilted 6 degrees with a small force, see figure (4.3). This tilt is unwanted. The system did not have a handle to rest the hand on. This made it more difficult to hold the hand still. The disc(4.2) could rotate independently from the rest of the device. This rotation made it more difficult to hold the hand still when resting the hand on the plate. This rotation could be removed if there were small pieces of paper placed in the joint that connect the plate with the rest of the system.

The arm(4.2) is the same that is used in the final design, except that the arm has been shortened to reduce weight of the arm, this might have helped reduce tilt.

Figure 4.1: Shows almost all the different braces tested.

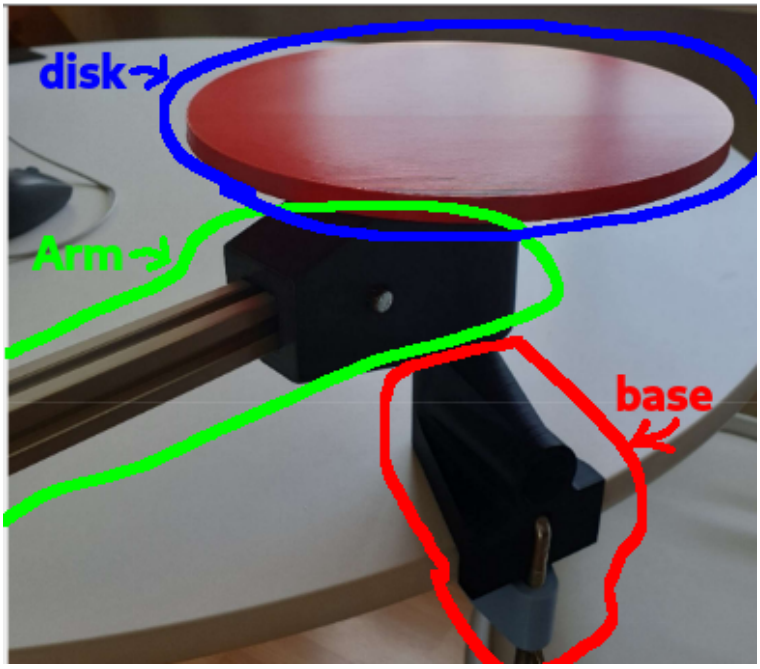
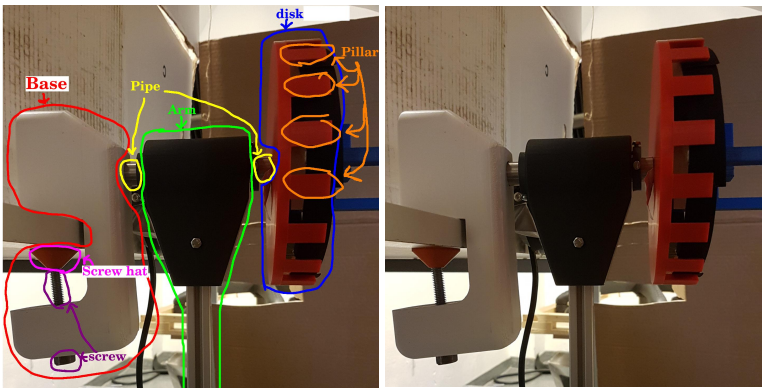


Figure 4.2: Shows the nomenclature used for different part in this thesis. This is the system that got passed down from earlier bachelor [9].

(a) (b)

Figure 4.3: Show how the cameras tilt in direction of elbow was measured, and the result is a 6-degree tilt. The yellow beam is a level that is placed vertically. The left image shows a 2-degree tilt when someone pushes the camera, with a weak force, towards what is being 3d-scanned. The right image shows a 4-degree tilt when someone pushes the camera, with a weak force, away from what is being 3d-scanned.



(a) (b)

Figure 4.4: Show the new base, and the name used for the different parts in this thesis. (a) shows name, (b) same as (a) without the drawings

4.1.2 Redesigned Base of System

The base is shown in figure (4.4), and it does not move in unwanted direction. The two holes that the pipe(4.4) is connected to deformed when the system is 3d printed. Because of this the two holes needs to be sanded down. The result of this sanding is that the top hole fits the beam. But the beam cannot enter all the way on the side hole. Even though the beam cannot be inserted all the way, the beam is still stuck and would not move in normal used, which is a good thing.

The screw hat could collapse from the pressure from the screw, as shown in figure (4.5). It was not visible from the outside that the hat had collapsed. But if it does, it prevents the screw from being tightening into the table. This makes the base able to move, which is not desirable. Now the hat is printed with 100% infill. The hat has never collapsed after this change was made.

The final system still used the arm(4.4) from the original system. This arm could tilt two degrees toward the elbow when a small force was applied on the arm, as shown in figure (4.6). The force needed for this is so small that the weight of the hand can make the hand tilt 2 degrees. When a medium force was applied, the part of the arm which is a metal beam could start bending, this could make the camera tilt even further. It is believed that the force needed to do this is so large that if the one 3d scanning is careful, the beam will not bend when 3d scanning. The only tilt of the new system is the arm, which can tilt two degrees, which is an improvement from the old system 6-degree tilt.

4.1.3 Handle

The different handles can be seen in figure (4.7). The first handle was a small disc. This was awkward to rest the hand on, and there was no place to rest the thumb. Therefore, a proper handle was made, but this blocked the view of the hand from the 3d scanner. It also encourages the placement of the thumb which made it hard to 3d scan. After this the final handle was made, this handle creates a place to rest all the fingers and it encourages a hand placement that is ideal

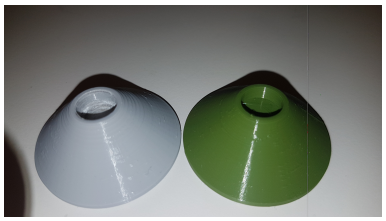
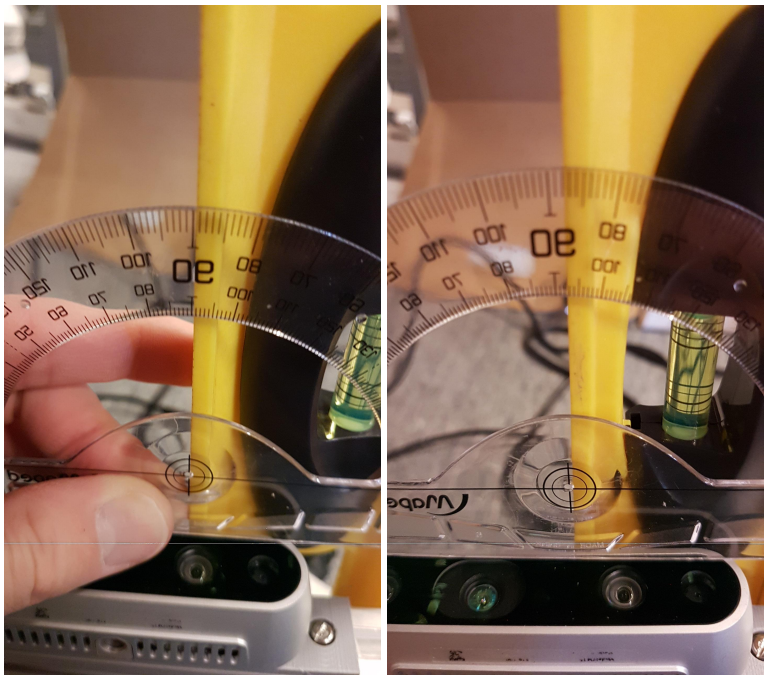


Figure 4.5: shows an undamaged screw hat (right) and a screw hat that has collapsed from the pressure of the screw(left). If you look closely, it can be seen that the hole in the gray hat is deeper than the hole in the green hat. This is what was meant by describing the hat as collapsed



(a) (b)

Figure 4.6: Show how the cameras tilt in the direction of elbow was measured. The yellow beam is a level that is vertical. The left image shows a 0-degree tilt when gravity pushes the camera away from the arm getting 3d-scanned. The right image shows a 2-degree tilt when someone pushes the camera, with a weak force, towards what is being 3d-scanned

(a) (b) (c)

Figure 4.7: Shows the different handles and how the hand is rested on it. (a) first generation handle, (b) second generation handle, (c) third and final handle.

for the 3d scanner. However, this handle has gotten a complaint that it was too small for people with big hands.

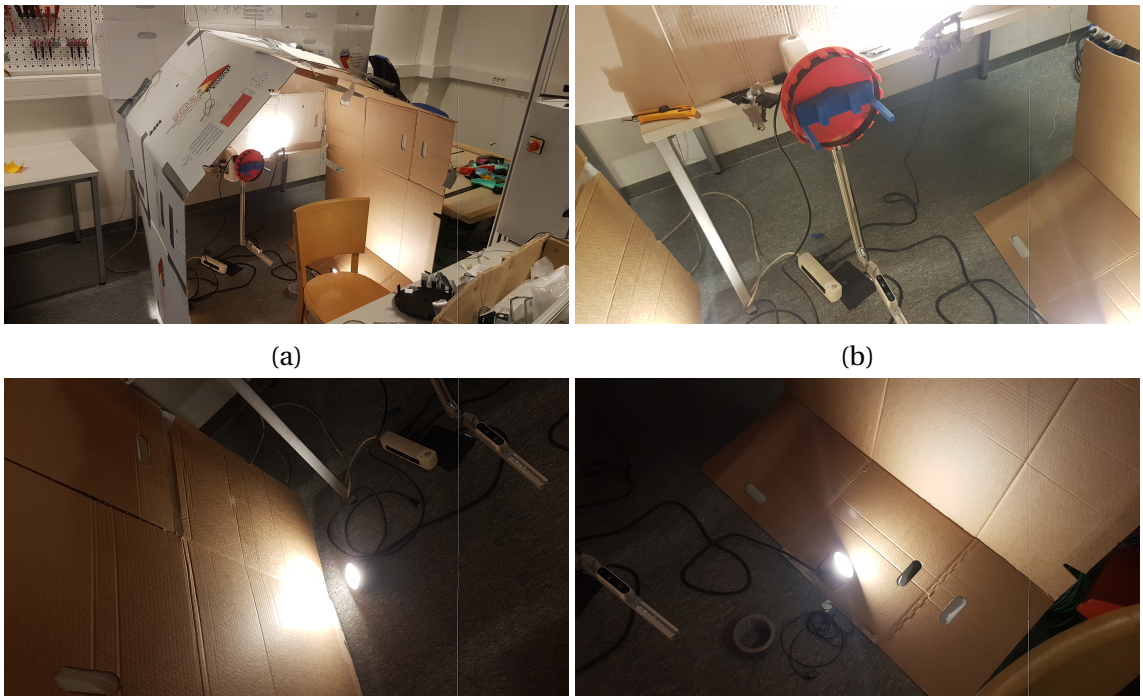
4.1.4 Light Conditions

In the beginning the lighting condition was not controlled. The 3d scan was taken in a room with light above the 3d scanner, but not below. Toward the end of the bachelor the lighting condition were more controlled. There was built a tent of cardboard boxes, and the inside was lit up with three small floodlights pointing towards the wall of the tent, shown in figure(4.8). This made the entire hand approximately illuminated the same amount, but this can still be improved.

4.2 Get Camera Position

4.2.1 Aruco

The camera calibration with circular pattern, shown in figure (3.7), gave an unusable result. The camera calibrating with the chessboard pattern gave a better result, but most of the calibration was not good enough to get an accurate position and rotation of an Aruco. In the end distortion coefficients were set to 0, and the camera matrix was manually tuned. At first the Aruco was not close to detecting the cameras position, but after a long time it was discovered that the Aruco codes were not perfectly flat. When this was discovered the Aruco codes were glued to a wooden plate. After all this, the method of using Aruco was still not able to fit two different 3d scans with the L515 into a single coordination system, which is shown in figure (4.9). At the time it was



(a) (b)

(c) (d)

Figure 4.8: Shows the cardboard tent with lights that was used to control the lighting condition

concluded that it was not possible for the L515 to get an accurate enough position from Aruco. But after more experience with L515, it was realistic that it might have been inaccuracies in the depth sensor that made the two 3d scan not align up, and not inaccuracy in estimating the position of L515. The experiment is therefore inconclusive. And using Aruco to estimate the position of the camera might have been a good way of estimating position.

Because it was first believed that estimating position with Aruco code was too inaccurate this method was not pursued further.

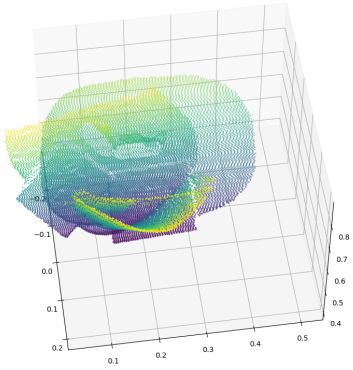
4.2.2 Angle Finding with Pillar Counting

This method is basically counting how many pillars have passed the camera and use that to get the same result as a potentiometer. There is more detail about this method in section 3.4.3.
How good method was at detecting the position of the camera was tested in 3 ways:

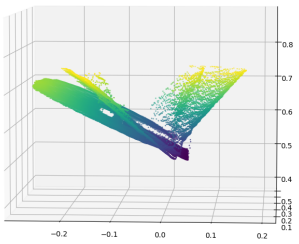
1. Have a person look at the image from the camera with the mouse pointing at the same pixel that the program uses to detect if there is a pillar there or not. Then the persons check if the program detects that the pillar has passed/entered the program at the same time that the beginning/end of the pillar passed the mouse cursor.

2. Move the arm 360 degrees around the plate(4.4), and then check how many pillars the program counted. The plate includes 18 pillars, and therefore the program should detect 18 pillars.

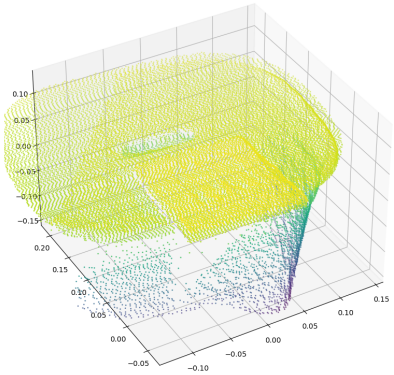
3. Check if the point cloud generated by the 3d scanner seems right.



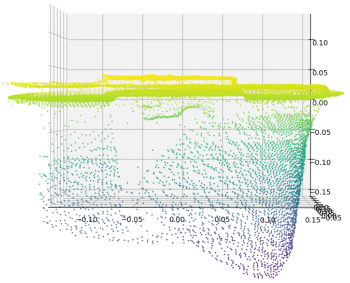
(a)



(b)



(c)



(d)

Figure 4.9: Those pictures are two 3d scan of a laundry basket with L515, and both 3d scans are plotted into the same coordinate system.
(a) and (b) is if it is assumed that the two scans were taken from the same position which they were not. (Those picture shows that the two 3d scans did not align before Arucos was used)
(c) and (d) is if the two position of the camera is estimated with the help of an Aruco code.
(d) shows that the two 3d scan does not perfectly align. This might be because the estimated position of the camera was not good enough or it might be because of inaccuracies in L515.

L515

For L515 the system seemed to work great to find the position of the camera. The L515 used its depth camera to detect if the camera had passed a pillar.

From test 1 it detected a pillar as good as a human. From test 2 it often detected 18 pillars, but sometimes detected more than 18 pillars, but this might be because it was hard to not move the arm backwards when rotating the arm around. From test 3 it seems right, but the data from L515 is so bad (see figure (4.10) that it is hard to tell if the position data is correct.

D435 without controlled lighting condition

This setup was unreliable, sometimes it worked and sometimes it did not. D435 depth camera was bad at detecting edges, and therefore was not suited at detecting the edge of the pillar, therefore its RGB-camera was used.

From test 1 it sometimes counted one pillar twice, or it might count multiple pillars as one pillar. From test 2 it did not detect 18 pillars most of the time, but sometimes it did detect 18 pillars. From test 3 only the point cloud of the 3d scans that detected 18 pillars was used. From those point clouds, some of them the position used to create the point cloud was clearly wrong. But for most it was hard to know if the position was wrong or if the depth sensor read the wrong depth.

D435 with controlled lighting condition

This setup seemed to work well. There seems to be a slight tilt (see section 4.7). This method used RGB-camera instead of depth sensor, because D435 is not good at detecting edges.

This setup was not tested with test 1. From test 2 it detected 18 pillars most of the time, but sometimes it detected more or fewer pillars. From test 3 most of the 3d scans were mostly in the right position, but one surface from 2 different images did not always match perfectly. This might be because of inaccuracy in the camera, or it might be because the estimated position is wrong.

4.3 3d Scanning

4.3.1 L515

This did not create a good enough 3d scan as shown in figure(4.10). Because it had high precision and low accuracy it was difficult to distinguish the real shape of the hand. After removing 30% of the points that were in the least dense areas, and after a lot of smoothing. The result was something that could barely resemble an arm, the model of can be seen in figure(4.11). The surface had many hills and valleys, and the hand part did not look accurate. Both of those things make it almost unusable for creating a brace out of. The 3d model is also too small. It had a

diameter of only 37mm, compared to 46mm which was the diameter at the same place, of the mesh that was used in the final brace.

4.3.2 D435 without controlled lighting condition

D435 has different modes. There were two modes that were used, it was the default mode and the high accuracy mode. The default mode shown in figure (4.12), was not as accurate as the high accuracy mode, but it was a lot more accurate than L515. After a lot of data manipulation, the result was something that looked like a hand, as shown in figure (4.13). But after printing it out it was shown that it was way too small, and that it had shrunk a lot more at the side of the hand than at the top.

The accuracy mode gave better accuracy, but it did not manage to get a reading of the entire hand, as shown in figure (4.14). After less processing of the data than used in the default mode, it got a 3d model which looked like a hand (expect for the fingers and thumb), which is shown in figure (4.15). Because it used less pre-possessing, it was believed that it was more accurate. However, it was way too small at the side of the hand.

4.3.3 D435 with controlled lighting condition

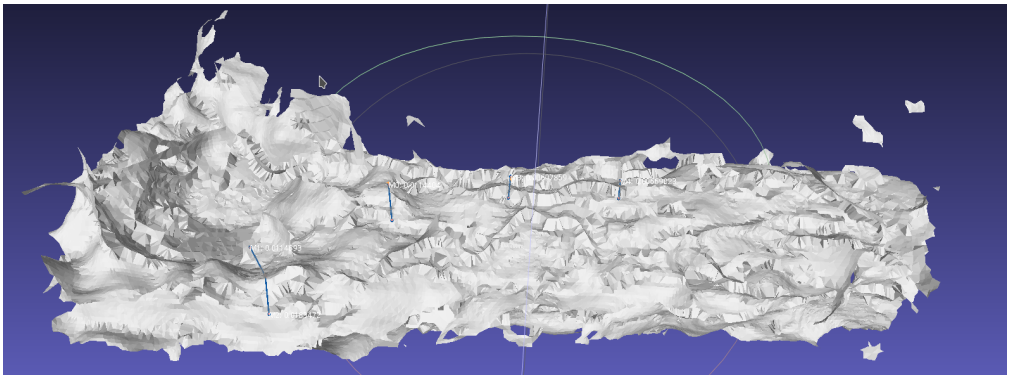
With the better lighting condition, the accuracy mode of D435 was able to measure the distance to more of the surface of the hand. The lighting condition was tuned in order to get the accuracy mode to be able to measure as much of the hand as possible. The result of this was that the scan was quite good. There was a little bit of noise but a lot less than earlier attempts, and it managed to pick up the entire hand, as shown in figure (4.16). The result of smoothing was also good, but it was a bit smaller than before smoothing, as seen in figure (4.17).

4.4 Creating Offset

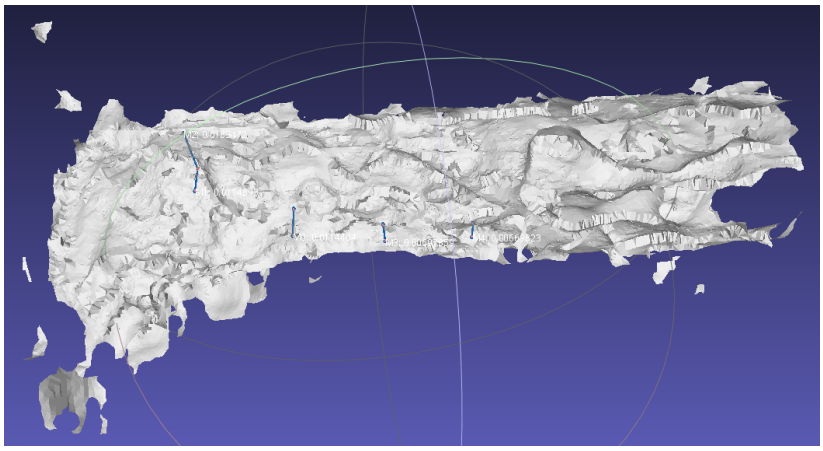
There was no problem with enlarging the latest 3d model with creating an offset of the surface, the result of which is shown in figure (4.18). After testing it was discovered that 3 mm offset in the x-direction and 8 mm offset in the y-direction fitted the best. Enlarging can create spikes that points inward, but because the 3d model was so smooth, this did not happen in this case.

4.5 Creating Brace

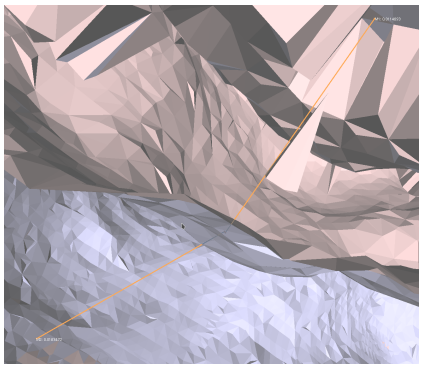
Did not encounter any problems with cutting up and printing the model. The result of which can be seen in figure (4.19). The brace fit well in the hand but was too big at the forearm.



(a)

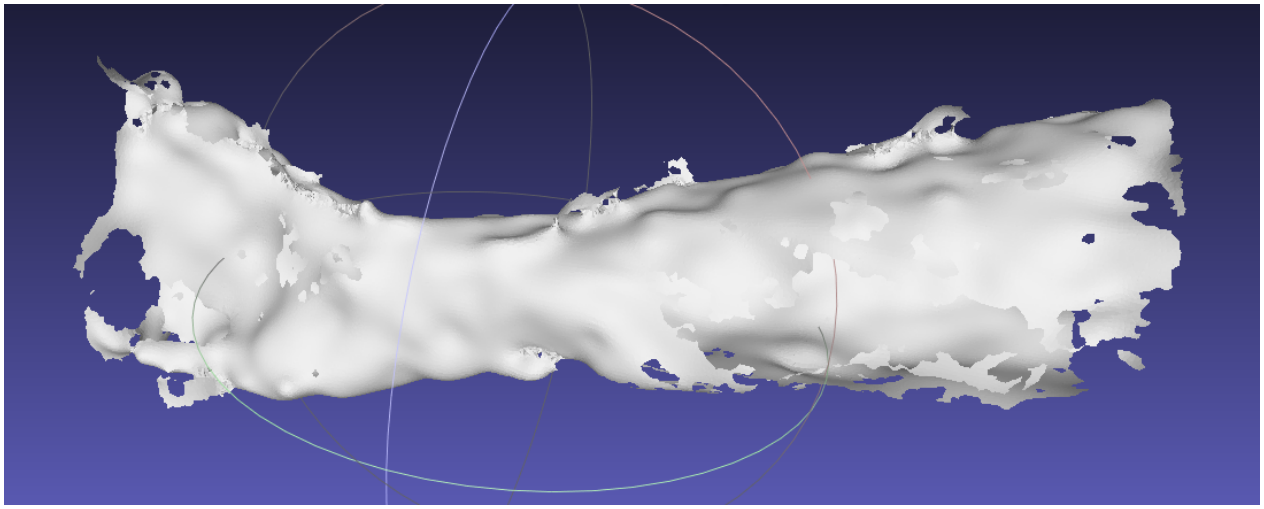


(b)

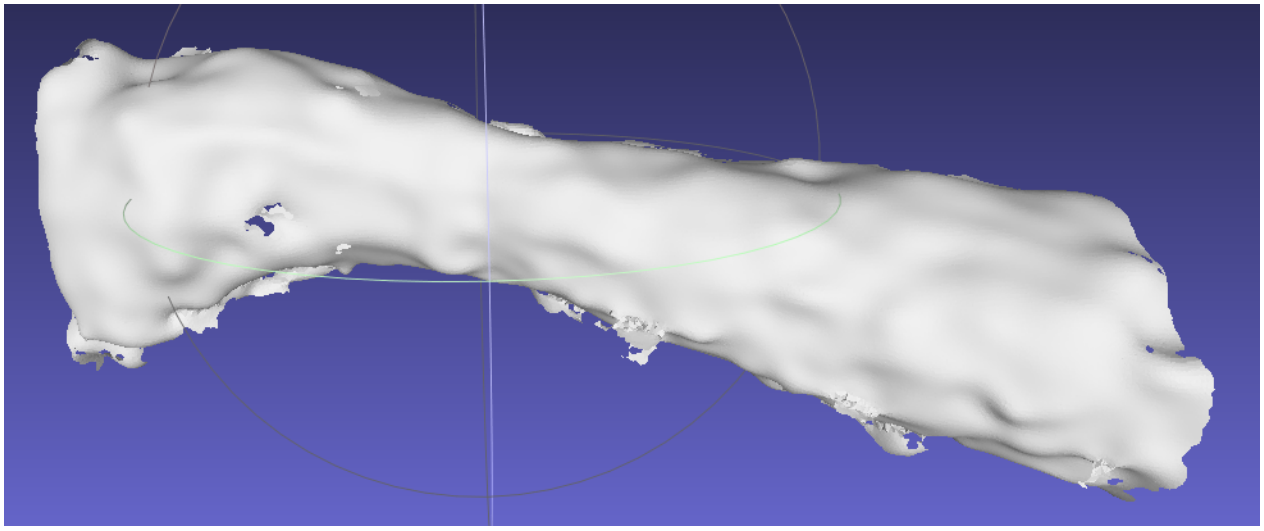


(c)

Figure 4.10: L515: shows a surface created with minimum manipulation of the point cloud. The reason that 3d model was shown instead of the point cloud is because it is hard to see the shape of a point cloud from images. The hand in this image was using handle v2(4.7) which make it harder to get a good scan of the thumb. One surface of the real hand has multiple surfaces in the scan, the blue line goes from the inner layer to the outer layer. One exception is the blue line most to the left in (a) and (b) which are the same lines as the orange in (c).
The points in the point cloud are often evenly distributed between the inner and outer layer of the 3d model. (a) shows the hand from 30 degrees from the front, the left side of the image is the hand. (b) shows the opposite side of the hand from (a). The left side of the image is the hand. (c) shows the thickness of the layers. The top line is 11 mm long and goes between the inner layer and out most layer of the hand. The bottom orange line is 18 mm long and shows the distance between the inner layer from top of the hand to the inner layer from the bottom of the hand. This will make it difficult to merge the two surfaces into a single surface without merging the both sides of the hand. The points in the point cloud are often evenly distributed between the inner and outer layer of the 3d model.

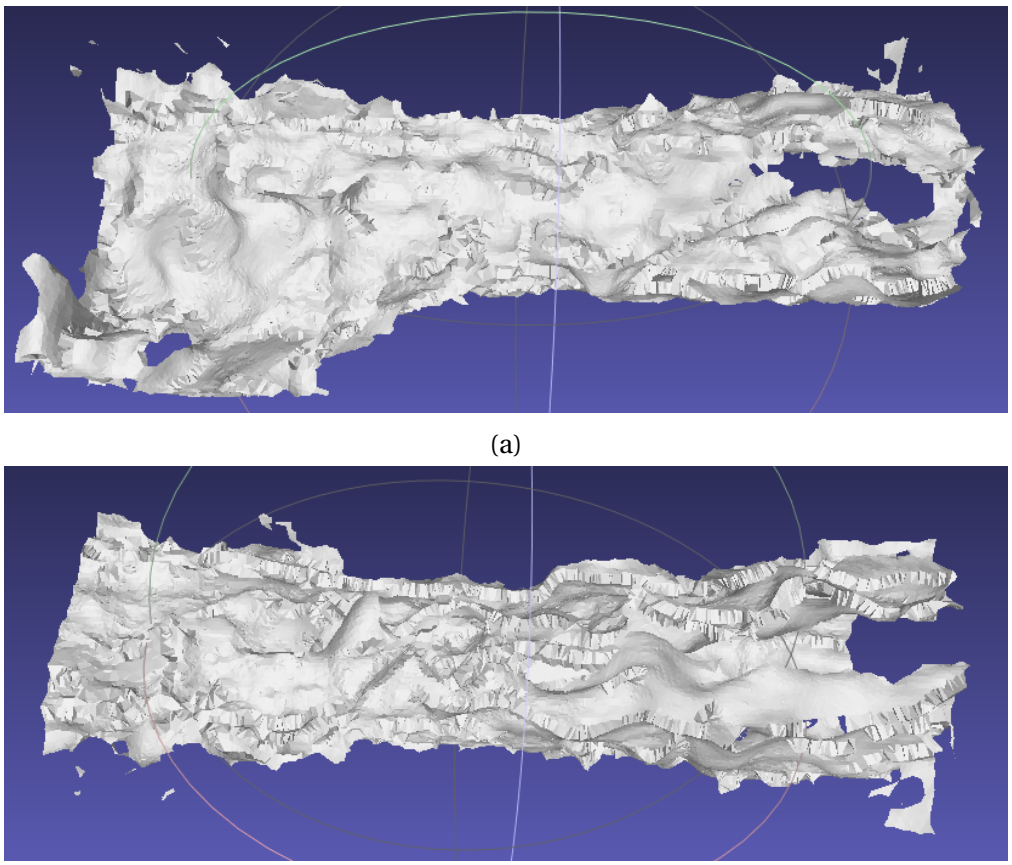


(a)



(b)

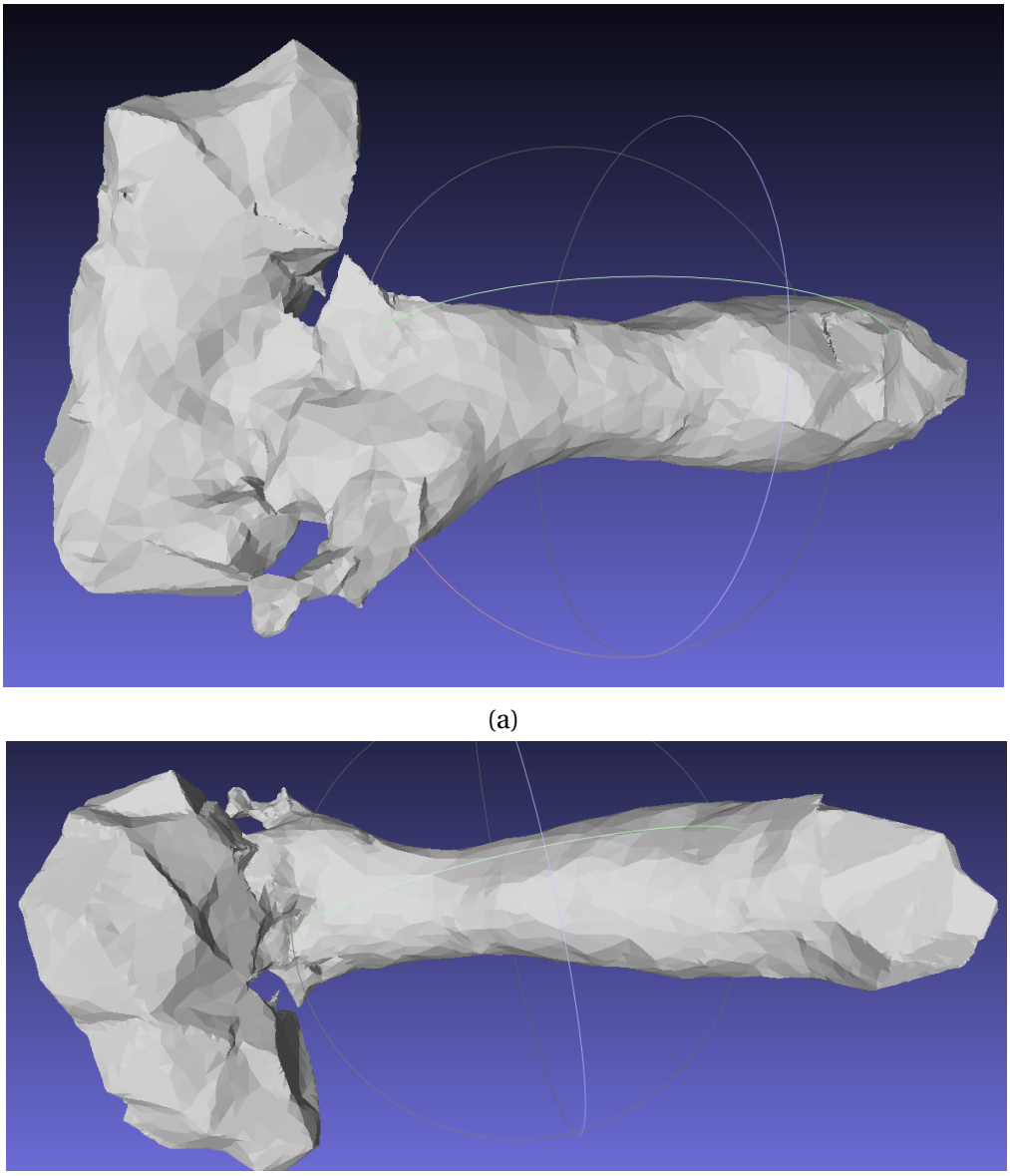
Figure 4.11: Shows the 3d model gotten from L515 after a lot smoothing. The smoothing process was first, remove 30% of the outlier. Then smoothed the point cloud 20 times using 200 neighbor points to smooth each point. Then Poisson reconstruction was used to turn the point cloud into a 3d model. The point cloud is the same used in figure (4.10).
(a) is the smoothed model from the front (b) is the smoothed model from the back.



(a)

(b)

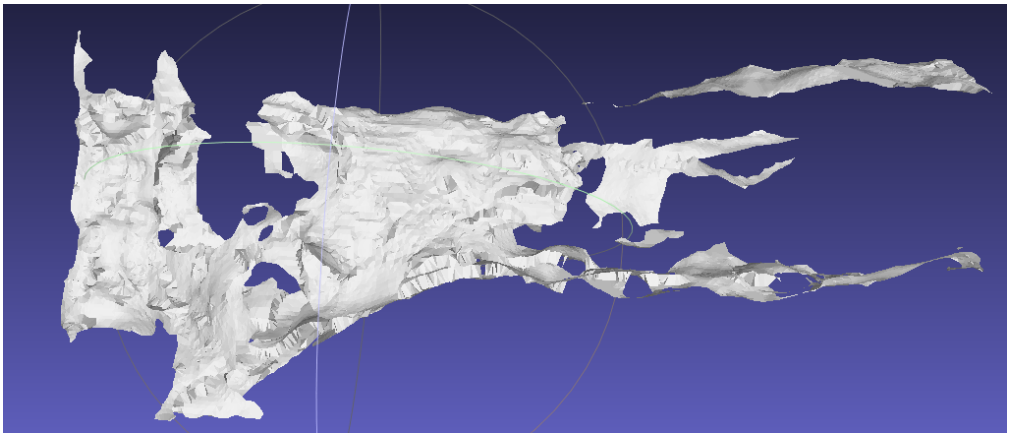
Figure 4.12: D435: default mode: shows a surface created with minimum manipulation of the point cloud, to help visualize the point cloud.
This shows that the point cloud is very noisy (a) shows the hand from the front, the left side of the image is the hand. (b) shows the opposite side of the hand from (a).



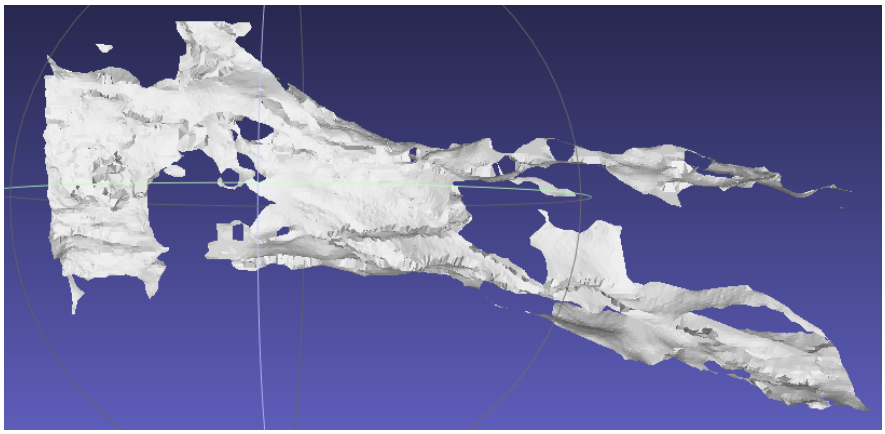
(a)

(b)

Figure 4.13: Shows 3d model of D435 with default setting after a lot of processing. The setting used to get this 3d model was not saved.
The person getting 3d-scanned had a swelling from an injury under the arm that day.
The 3d scan is a lot better than L515 (4.11), but the hand is still quite rough.
At the top of the hand, it looks like there is a big "balloon". A likely reason why it looks like this is because too many points were used to calculate the normals, more detail at section 3.5.1.

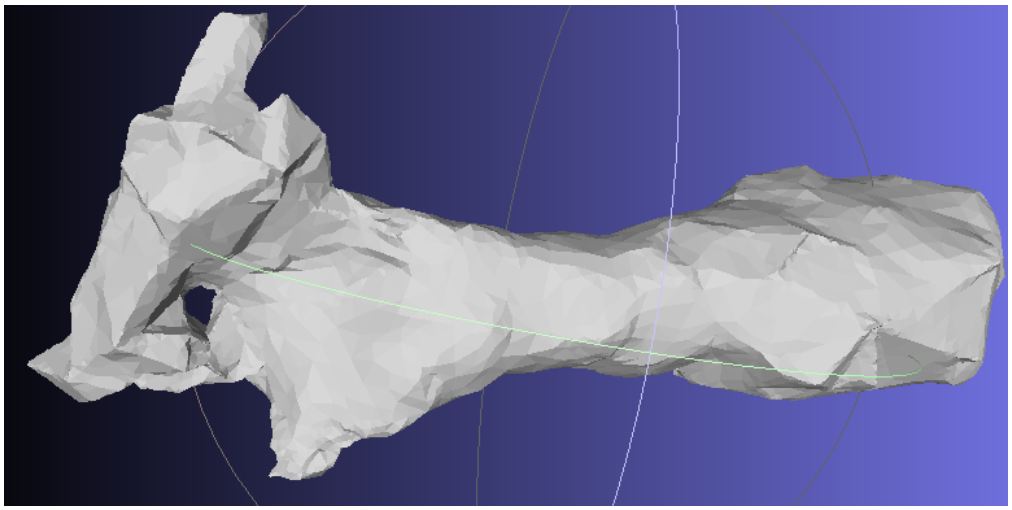


(a)

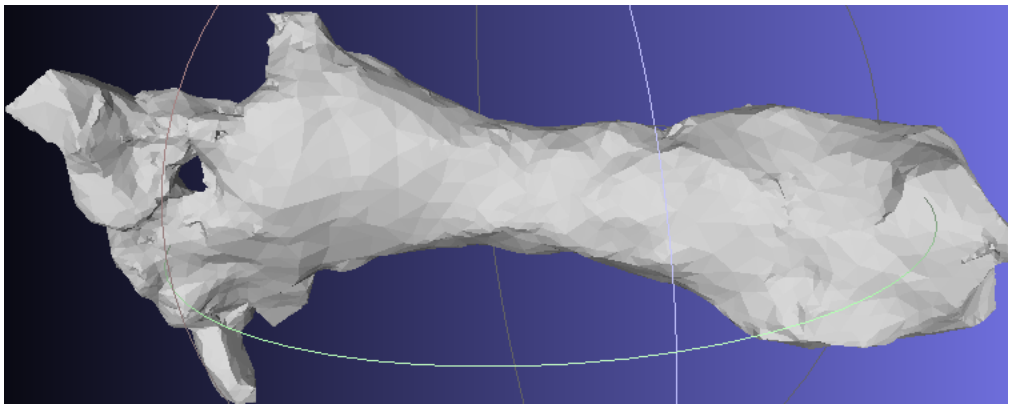


(b)

Figure 4.14: D435: accuracy mode: Shows a surface created from with minimum manipulation of the point cloud, to help visualize the point cloud.
Those images show that the result is more accurate than D435 without the default setting, but there are large areas where the camera does not pick up any points. Most of the surface was just a plane but at the bottom of the hand had a 2 mm thickness
(a) shows the hand from the front, the left side of the image is the hand. (b) shows the opposite side of the hand from (a).

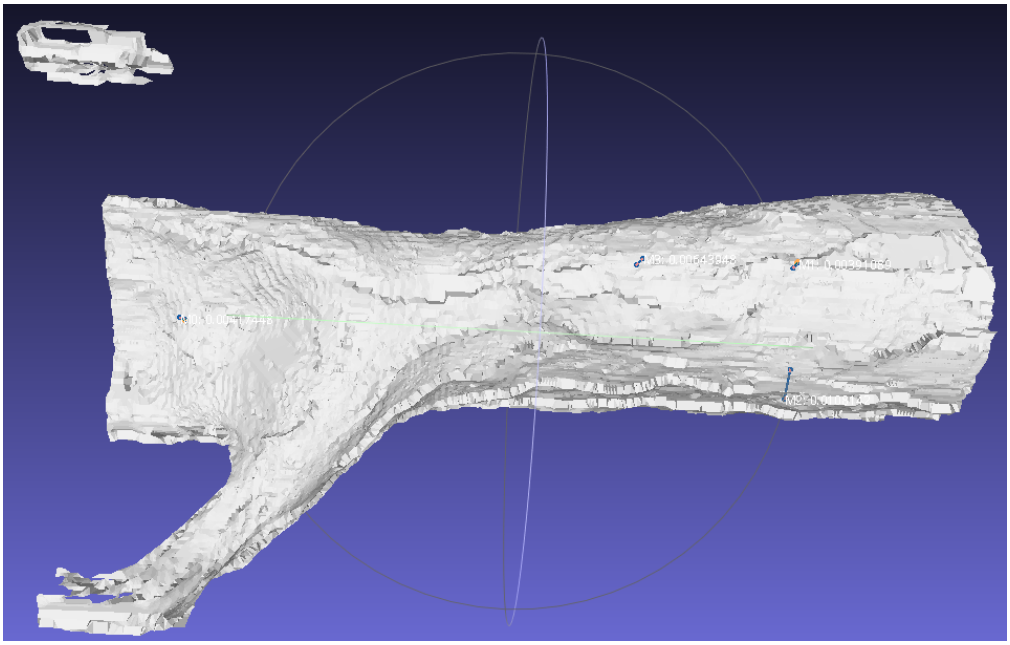


(a)

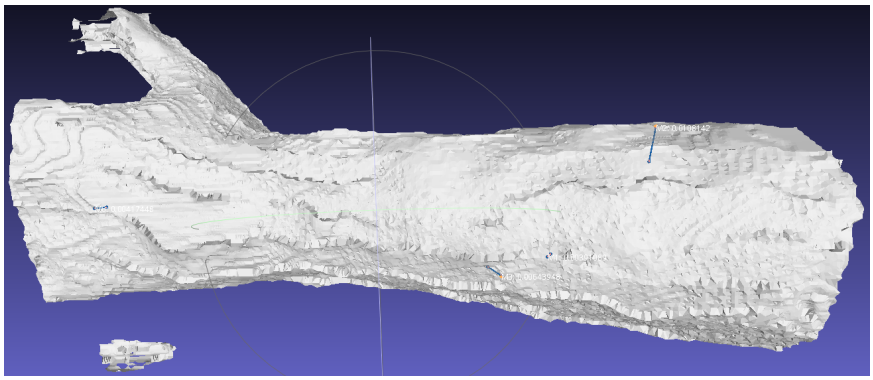


(b)

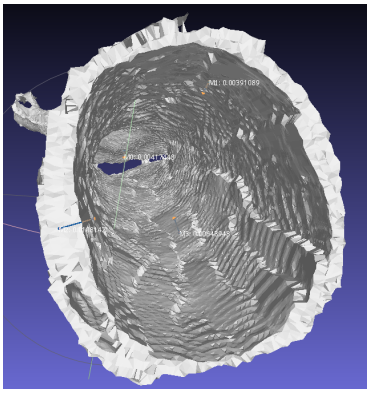
Figure 4.15: Shows 3d model of D435 with accuracy setting after a lot of processing. The setting used to get this 3d model was not saved.
The person getting 3d-scanned had a swelling from an injury under the arm that day.



(a)

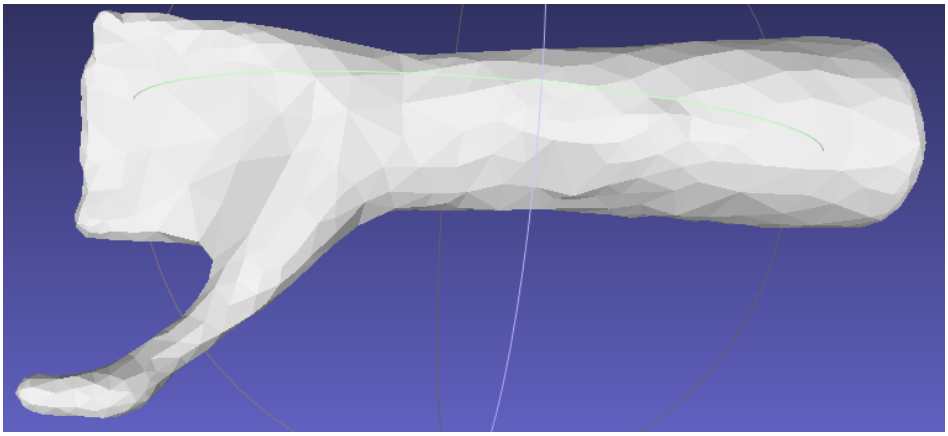


(b)

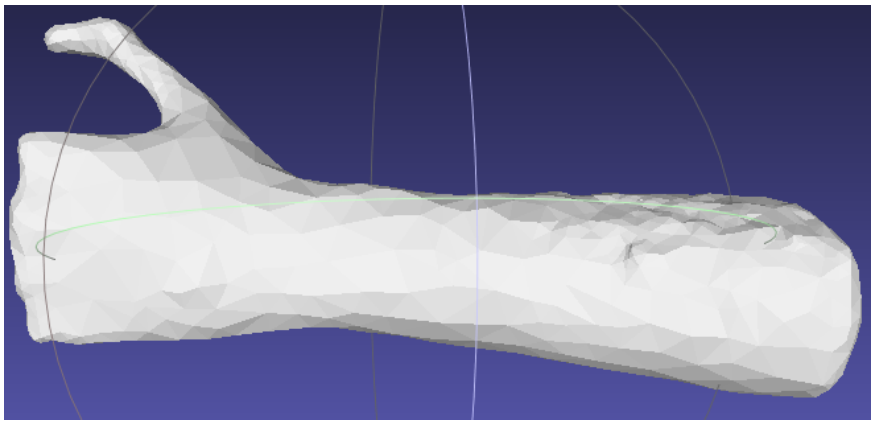


(c)

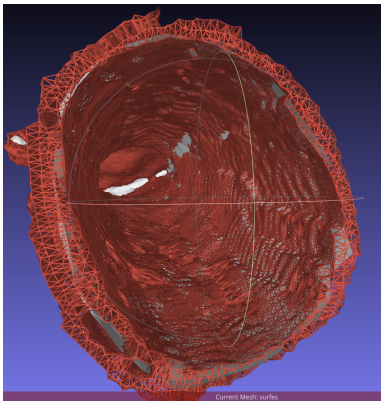
Figure 4.16: D435 accuracy mode with controlled lighting condition: Shows a surface created from with minimum manipulation of the point cloud, to help visualize the point cloud.
The images show that the point cloud is a lot better then earlier attempts. (a) shows the hand from the front, The floating beam in the top corner is part of the hand rest, and the reason the "thumb" is so long, is because it connects with the hand rest. (b) shows the hand from the back. (c) shows the inside of the model, this shows how the thickness of the surface is in range 4 to 10 mm. The points in the point cloud are often evenly distributed between the inner and outer layer of the 3d model.



(a)

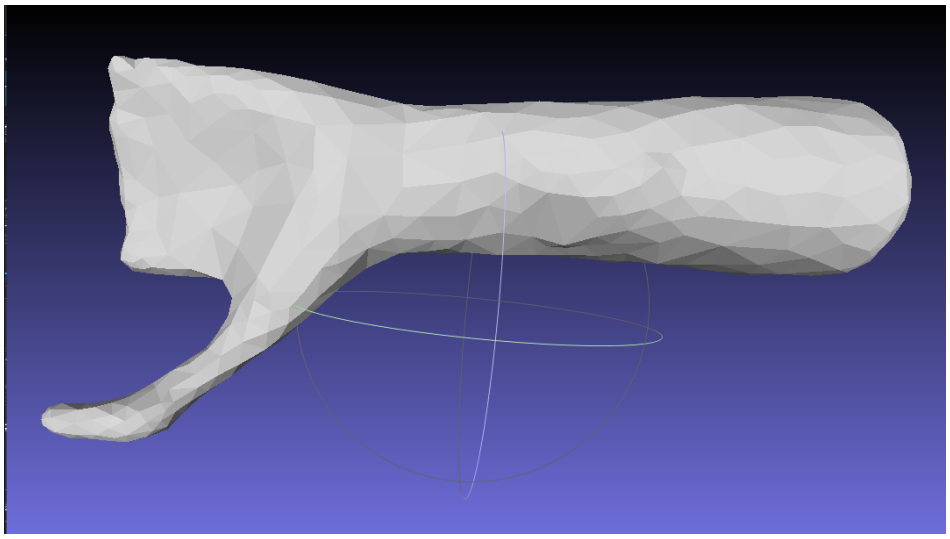


(b)

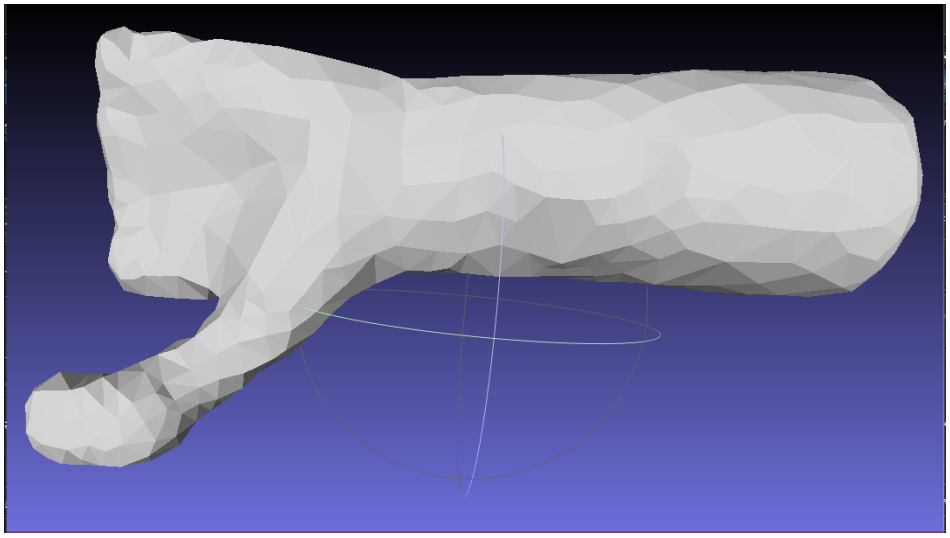


(c)

Figure 4.17: D435 accuracy mode with controlled lighting condition: Shows a 3d model after the automatic manipulation of the point cloud and 3d model, but before it becomes enlarged. The images show that the 3d model is a lot better than the earlier attempts. The images show that the point cloud is smooth and looks like a hand. It also shows that the hand shrunk when it was processed. (a) shows the hand from the front. (b) shows the hand from the back. (c) shows the inside of the model, the red mesh is the mesh gotten with minimum manipulation (4.16).

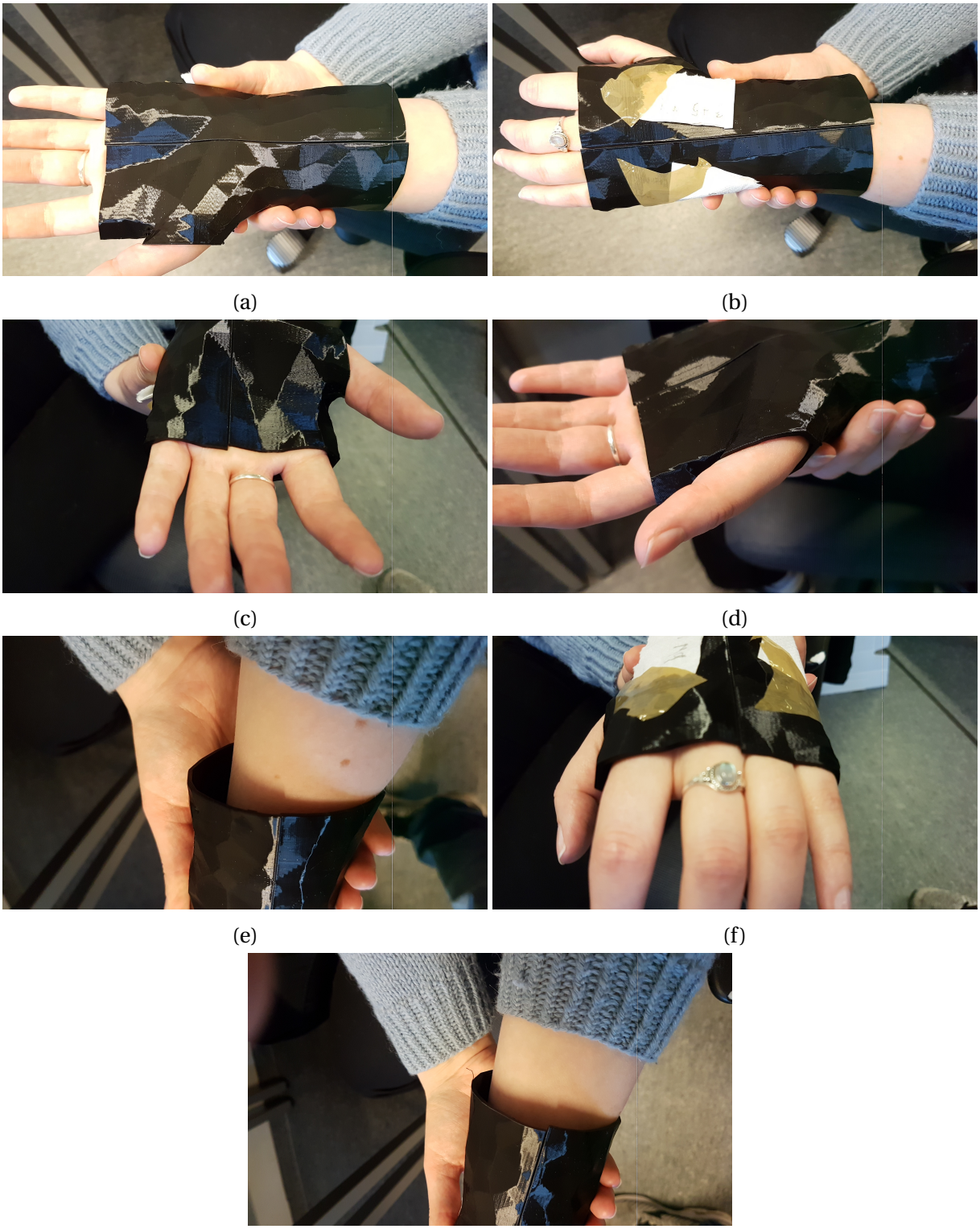


(a)



(b)

Figure 4.18: Shows before and after enlargement with a surface offset of 3 mm in x-direction (towards the camera) and 8 mm in y-direction (side of the arm).
(a) shows the model from figure (4.16).
(b) shows the model after enlargement, this model is the inside of the brace in figure (4.19), before the model gets cut up. The pictures were taken from identical position as in (a). Therefore, the increase in size seen is the real increase in size.



(a) (b) (c) (d) (e) (f) (g)

Figure 4.19: The final brace, it fits well in the hand, but it is too large around the forearm

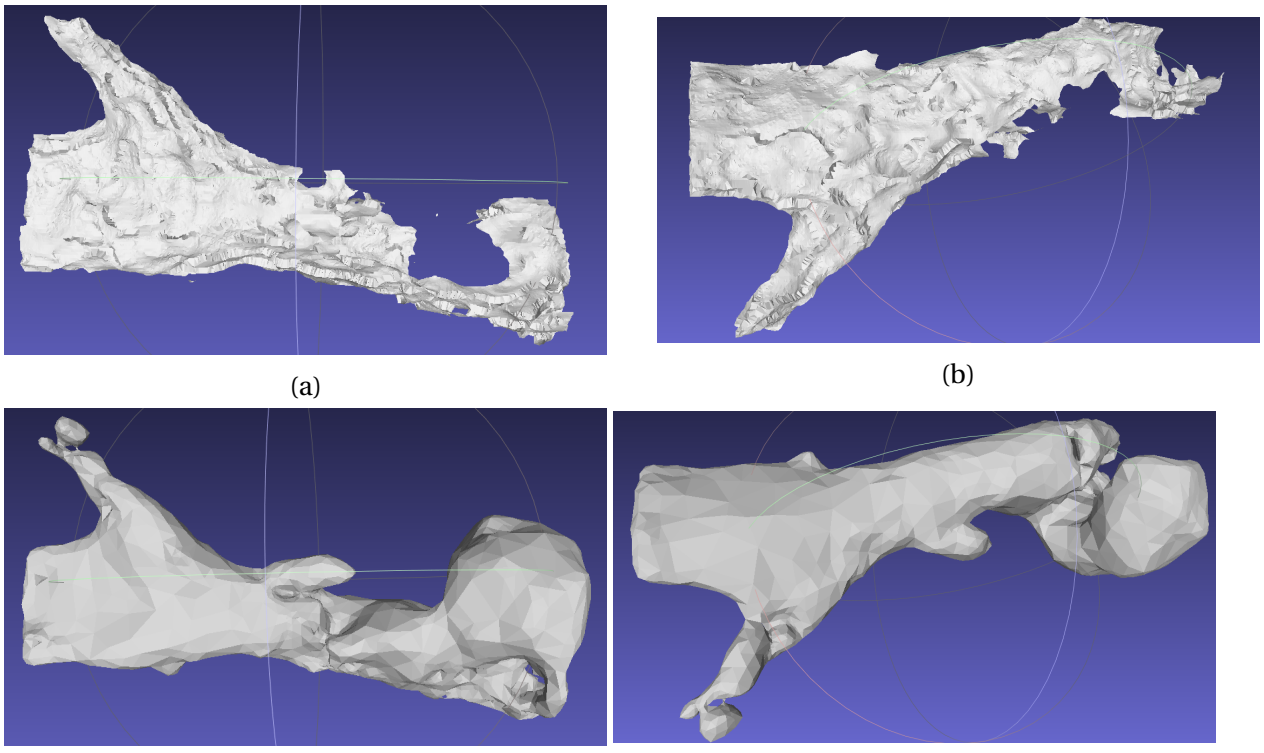
Participant 1	
Scan Result	Got a good point cloud after 3 scans. The point cloud did not have holes and it was dense.
Brace Result	Generally good brace but it was too large around the forearm.
Changes in Automated process	The automated process was built for this person.
Participant 2	
Scan Result	Very bad, did try 15 times without getting a good point cloud. One of the better point clouds, did not include most of the forearm, and the point cloud was less dense than the point cloud from participant 1.
Brace Result	No brace was created, because the point cloud was too bad.
Changes in Automated process	—||—
Participant 3	
Scan Result	2 out of 6 3d scans were good. The best point cloud was not as good as the point cloud from participant 1. It had a hole near the thumb and a big hole in the forearm. But the hole in the forearm was so far up the arm that it did not matter for the brace used for testing.
Brace Result	It was too little around the same place that there was a hole in the point cloud. Just as for participant 1 the forearm was too big, expect for this it fits good.
Changes in Automated process	The only change was that it needed 2 mm bigger offset when the brace was enlarged.

Table 4.1: Shows the result when checked if what the different scripts did.

4.6 Test Automation of the System

It was a possibility that the automatic system used to create the brace in figure (4.19) is a system that is only tuned to work on that one person (participant 1). To test if the system can be generalized to more people, the system was tested on two new participants (participant 2 and 3). The result of this is shown in figure (4.20), (4.21) and (4.22). The result is summarized in table (4.1). For unknown reasons, the 3d scanner did not get a good point cloud from participant 2. There were more than 15 attempts at getting a good scan of the hand from this person. Because the 3d scanning part did not succeed, there was not created a brace.

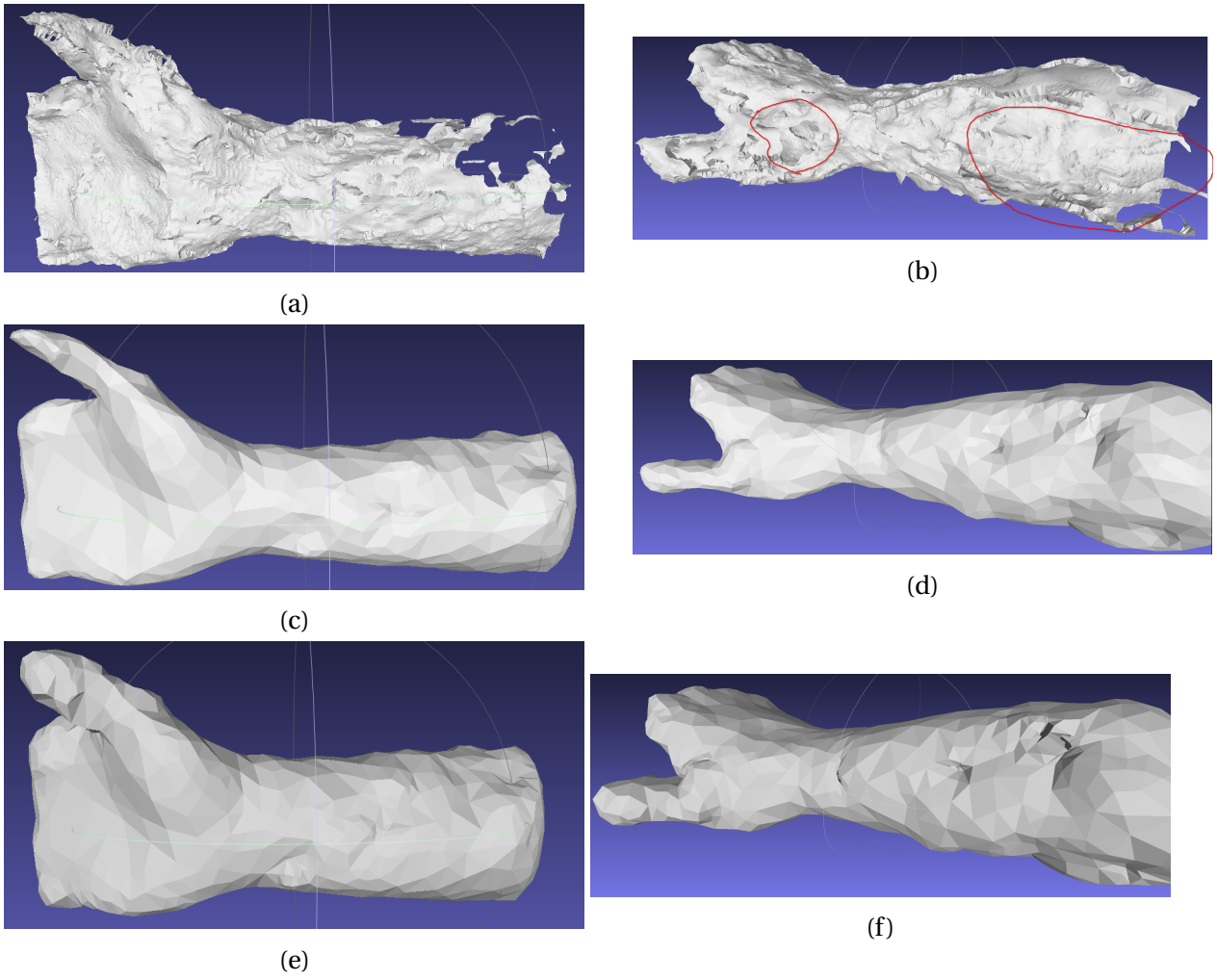
There was better luck with participant 3. participant 3 got a good enough point cloud from 3d scan in 2 out of 6 cases. But participant 1 got a better point cloud. The automated part of the process did not need to be changed except for the enlargement step. Here it needed to add 2 mm offset. The brace fit well, except that the area for the thumb that the depth sensor did not read was too small, and it was too wide around the wrist and the forearm, as shown in figure (4.22). But it was also too wide in the same area for the first participant.



(a) (b)

(c) (d)

Figure 4.20: Shows the result from 3d scanning participant 2.
The scan used the same settings as used for participant 1, but it was stopped before enlargement. There were over 15 attempts at getting a god scan, and this scan is one of the better.
(a) and (b) are the 3d model created with minimum manipulation of the point cloud. Those images show that there was a lot of the surface the depth camera did not get. It also shows that the part of surface that the 3d scanner got is noisy. (c) and (d) are the 3d model after going through the same algorithm with the same constants as participant 1 used, but the model was stopped before enlargement.



(a) (b) (c) (d) (e) (f)

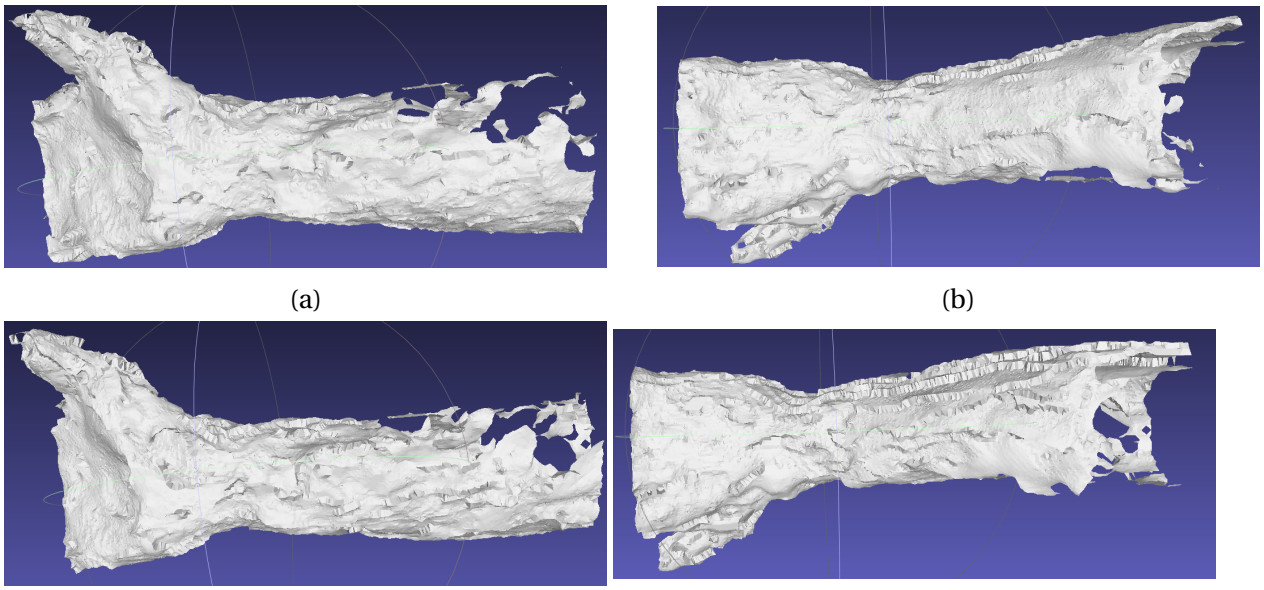
Figure 4.21: Shows the result from 3d scanning participant 3.
The scan used the same settings as used for participant 1. There were 6 scans, 2 of them were good, this is one of them.
(a) and (b) are the 3d model created with minimum manipulation of the point cloud. Inside the red circle there are a lot of surfaces missing. The model is noisier than participant 1 but less than participant 2. (c) and (d) are the 3d model after going through the same algorithm with the same constants as participant 1 used, but the model was stopped before enlargement. (e) and (f) is the inside of the brace.

(a)

(b)

(c)

Figure 4.22: The final brace for participant 3. It is too small around the thumb, around the area that did not have a surface in the point cloud (see figure (4.21)). Except for that, it fit good in the hand, and it was too large around the wrist and arm.



(a) (b)

(c) (d)

Figure 4.23: Compares point cloud from 3d scan if assumed no or 2-degree tilt.
Shows that the 2-degree tilt gets noisier. Shows the result from 3d scanning participant 3.
The scan used the same settings as used for participant 1. (a) and (b) are the 3d model created with minimum manipulation of the point cloud. Inside the red circle there are a lot of surfaces missing. (c) and (d) are the 3d model after going through the same algorithm with the same constants as participant 1 used, but the model was stopped before enlargement.

4.7 Checking for Tilt

The arm that holds the 3d scanner can tilt 2 degrees (see figure (4.6)), and the 3d scan creates a 3d model that gets too small toward the finger, and too large towards the elbow. It was therefore thought that it is the 2-degree tilt that might make the 3d model so small in the finger and so big towards the elbow.

To test this hypothesis, a 3d scan was processed twice. One time it was assumed that there was no tilt to the camera, the other time there was assumed that the camera tilted 2 degrees towards the elbow. The results are shown in figure (4.24) and (4.23). The point cloud of the with 2-degree tilt was noisier than the point cloud without tilt. But the final 2-degree brace was a lot tighter towards the elbow than the 0-degree brace. The 2-degree brace was ca. 3 mm to big 7 cm from the hand, and 0-degree brace was ca. 12 mm to big at the same place. The 2-degree brace was tight in most of the hand part of the brace, but the 0-degree brace was only tight right below the fingers.

(a)

(b)

Figure 4.24: (a) is the brace created if it is assumed that the camera does not tilt toward the elbow Those images show that (b) fits the hand much better than (a) and therefore the camera might be tilting when 3d scanning.

Chapter 5

Discussion

In this chapter, how well the objectives were achieved is discussed, how much is left before it can be used in practice, how beneficial it will be for the hospitals that use it and if it might have some other use cases.

5.1 Performance of Developed System

The objectives in the bachelor was to make a 3d scanner and a program that automatically processed data from the 3d scanner to make it into a 3d model of a hand, which later is going to be used to 3d print a brace.

The task that this bachelor tried to solve was to make a system that 3d scans the hand and then uses that 3d scan to create a model of the hand that is accurate enough to be used to create a brace from. This process should be as automatic as possible.

The 3d models created in this bachelor have been so wide up the forearm, that it does not support those parts of the forearm. The 3d model therefore needs improvements before it can be used as the basis for a brace. The oversized forearm is most likely caused by slack in the construction of the frame. Measures to overcome this issue will be discussed in the next section. Another problem with the way the task was solved was that the 3d scan was a little unreliable, 1 out of 3 people that got scanned did not got a good scan, even after a lot of scans. Most of the process is automatic. The only part that is not automatic is rotating the 3d scanner around the hand and creating a brace out of the 3d model. The latter of those 2 steps is outside the scope of the task, but it is still partly automated. There is one last problem, and that is that one of the models needed a 2 mm bigger offset when the 3d model was enlarged.

5.1.1 How to Solve Oversize Forearm

The experiment that showed that with a 2-degree tilt most of the extra width around the forearm is removed (see figure 4.24). But the brace was still a little bit looser around the forearm than it was in the hand. This may suggest that the camera can tilt slightly farther than 2 degrees.

The way the tilt was measured was not accurate enough to measure less than 1 degree. It also might be the case that the data processing shrinks the hand more than the forearm. But when comparing the 3d model before enlargement to the point cloud, the hand or the forearm of the 3d model was not noticeably bigger/smaller than the corresponding part of the point cloud. Therefore, it is not the main reason why the forearm on the 3d model is so large. There is strong evidence that the tilt is the main reason the forearm of the 3d model is too big, it is therefore concluded that the tilt is the major or only reason for that the forearm of the 3d model is so big.

There are three ways to solve this problem. One is to make the construction more stable, as told in results, most if not all the tilt can be removed by making the 3d scanning arm more tightly connected to the beam it rotates around. Another way is to use a different way to get the position of the camera. If the position of the camera is gotten in a way that can also detect the tilt. If the program knows the tilt on each frame, then the tilt is not a problem. Aruco is one example that can detect tilt, there is also possible to use a hybrid between different ways to detect position. One example of this is that the main position is gotten the same way it is now, but the way it decides the tilt is by choosing the tilt, such that the floor and walls of the room line up in the different frames. The reason that the tilt of the camera now makes the forearm big is because the program wrongly guesses the tilt in at least some frame. Therefore, using a method to find the tilt of the camera only changes the problem from "guessing the position wrong because of tilt in the system" to "guessing the position wrong because of uncertainties of position reading algorithm". It is therefore important that the finding position algorithm is extremely accurate and precise, which might be hard to accomplish. The final way to solve this problem is to make the tilt in each frame repeatable and known. This might be done by letting a motor rotate the arm around the hand instead of a human. The motor might be able to push the arm with the same force each time, and therefore it might have the same tilt on the same frame each time the system 3d scans. If the system has the same tilt each time it 3d scan, then the tilt could be found by 3d scanning an object with known shape. The problem with this method is that even if the system has the same tilt when it was tested, it might not have the same tilt after wear and tear or in a different environment. The good thing about this method is that a motor is desirable for other reasons than detecting tilt, and therefore it might not be that much more work to implement.

The safest option to reduce the negative effect of tilt seems to be to make the construction more stable. It seems easy to improve the construction stability, but it might be hard to make the system so stable that tilt no longer is a problem.

5.1.2 How to make the system more reliable

The results show a correlation between good point cloud after scan and accurate 3d model of hand. D435 seems to be greatly affected by lighting condition, and the lighting condition of the 3d scanner can be greatly improved. This probably will improve the 3d model, but it will decrease the number of scans needed to get a good scan. Other 3d scanners than D435 might work better and give a better point cloud.

The reason why the two new participants that tested if the algorithm was general did not get as good of a result as the first participant might be. Because the shape of their hand is different, and D435 has problem picking up the shape. This seems unlikely because the shape of the hand is quite similar. It is also unlikely that it is because of skin color because their skin color was quite similar, and the one that got the worst scan did not have the lightest or darkest skin color. Another reason can be because the light condition changed, the tent can easily deform, the lights are not glued to the floor, and the lights from the window might have changed the lighting of the hand. This seems to be the most likely reason. This might be solved by improving the lighting condition or by using a different 3d scanner that is not as affected by the lighting condition. It might also be because the way the 3d scanner was manually turned around the hand.

Making the 3d scanner more stable would probably remove some of the noise of the point cloud which would make the system more reliable, but it probably would not remove the holes in the point cloud which is the main reason for its unreliability.

5.1.3 How to solve offset problem

There was one step in the automated process that needed to be changed between the two models. That one thing was the offset used to enlarge the hand. The offset difference was not big, only 2mm. Two attempts do not give a good enough data to conclude anything about the range, but if it is only 2 mm, then it is 5 times more accurate than the depth camera used, which has an inaccuracy of about 10 mm[4].

$$distance \cdot \%Accuracy \approx 476mm \cdot 0.02 = 9.5mm$$

This problem might be mitigated by right design of the brace. Flesh is flexible, therefore if the brace does not get supported on the bones, and it has breathing holes to let the flesh move out of the way. Then having a 2 mm range in offset might not be a problem. However, this might not work in practice. The Orthopedic surgeon ,Terje Vagstad, explained that having a 2 mm to big brace would not be a problem. He told that if the brace is to big, a sleeve can be placed under the brace, and that this can be more comfortable than having the brace directly on the skin.

The brace can therefore be made slightly larger then it is now, and therefore the 2 mm is not a problem.

The reason for the 2 mm difference in offset is unknown. But it is reasonable to assume that it is because of the tilt problem, because of the quality of the point cloud or it is because of how that data is manipulated. How to improve the quality of the point cloud is discussed in "How to make the system more reliable". If the quality of the point cloud increases, then it is not necessary to manipulate the data as much. Therefore, solving the two other problems and eventually reducing the then not needed data manipulation will most likely reduce or remove the range of the difference in offset.

5.2 Usefulness of Automated Creation of Brace

5.2.1 Usefulness for the Hospital

This product can save time for the nurses because they would not need to plaster the hand. It takes a long time to 3d print a brace, therefore the patient might not save time, but the nurse can do other things while they are waiting. The time between 3d scan and mounting the brace will probably not be as productive for the nurse, as the same span of time usually is. This is because they will not be able to take on tasks that take too long time, without being interrupted when the brace should be mounted. The nurse has the option to delay the mounting of the brace, but this will waste the patient's time. If the machine needs multiple attempts to make a good enough brace, then the less productive time between scanning and mounting might not be worth the time saved from plastering the hand manually. One way to reduce the time wasted by multiple attempts, is to use multiply 3d printer to print multiple attempts of the brace at the same time. But this might be expensive and take up a lot of space.

This process will probably take longer time for the patient, but it is probably worth it, because of the utility of a brace that can withstand water. Plaster gets damaged by water, but plastic brace does not. Because of this you can shower the broken hand, and the brace will include breathing holes. This will increase hygiene and comfort.

The use of the 3d scanner might need less training than plastering the hand. But there would be more training in order to maintain the system. The system has many parts and complex sub systems, such as the 3d scanner, computer, software and 3d printer. When any of those parts get broken it might take a long time before the system can create more braces. Creating redundancies will help, but there are situations where redundancy will not help, such as malware and that Intel stops selling the 3d scanner used. Because of this the hospital still needs to keep the competence and tools needed to plaster the hand manually.

When breaking bones in the hand, the hand usually starts swelling. When the swelling decreases, the hand needs to be re-plastered. The plastic brace can be designed to allow the brace to be unmounted without destroying the brace. Because of this, if the 3d scanner is cheap enough and easy enough to use, then the 3d scanner can be borrowed by the person with the broken hand after the first brace. Then when he needs a new brace, he unmounts the old brace, 3d scans his own hand, remount the old brace, sends the data to the hospital. And when the brace is printed, he drives down to mount the new brace and for a quick checkup. If this works, it can save a lot of time for both the nurse and the patient.

5.2.2 Other Advantages

If this method becomes mass adopted, then there is going to be a lot of 3d scans of broken hands, and their recovery from when the hand gets re-scanned. This data might be valuable for researchers. One example of use case for this data is to use artificial neural network (AI) to predict the change in swellings. And with this AI, there might be possible to print all the braces

needed for the hand to recover with only the first scan. And there for saves a lot of time for the hospital and the patient. The data collection for this might be problematic for privacy, and there might be legal challenges to do this. It would be problematic if the patient would have to sign the 3d scanners privacy policy in order to treat their broken hand.

The system can also be used as a general purpose 3d scanner, but there is probably going to be other cheaper and better 3d scanner. Therefore, it is probably not going to be used for general purpose 3d scanning.

Chapter 6

Conclusions

The task in the bachelor was to make a 3d scanner and a program that automatically processes data from the 3d scanner to make it into a 3d model which later is going to be used to 3d print a brace. The software was a lot better than what could have been expected. It managed to fully automatically take the data from the 3d scanner and create a 3d model of the hand which was good enough to create a brace from. This program managed to create a good brace for a person it was not tuned for, the only thing that needed to change was a 2mm bigger offset from the surface. This 2mm is within the margin that it is okay for the brace to have. The hardware could be improved, the 3d scanner could tilt two degrees in an unwanted direction, this made the forearm of the 3d model too big and therefore the forearm of the brace was also too big. The 3d scanner also did not manage to read the position of every place on the arm, this is likely because of the lighting condition, which can improve.

The task given was very ambitious, and it was not solved perfectly. But the result was good, it greatly improved the development of the automatic brace system. And with a bit more time it is likely that the last two problems of the project could be fixed.

6.1 Further work

6.1.1 To Improve the current product

The things lacking in this bachelor to make it solve the task good enough so that it does not need to be worked on in the future is, removing the negative effect of tilt and to improving the reliability of the 3d scanner. Both of those seem to be able to be solved by improving the physical part of the system. This is a good thing, because the code used is more complex than the physical system. And therefore, the next bachelor only needs to learn the easier part of this bachelor. Unfortunately, there are a few constants that probably need to change in the code, if the physical system changes. The easiest way to solve those problems seems to be to redesign the arm of the 3d scanner and to improve the lighting condition. It is unknown if new problems will be discovered when those two things are done. But if there is no more problem, and these two

solutions would solve the existing problems, then it would not take a lot more time to solve the task of this bachelor.

6.1.2 Clinical Implementation

After what was discussed in the last section is done, then there needs to be created an automated program that creates a good 3d model of a brace out of the 3d model of the hand. These two things combined probably would takes an entire new bachelor. After this the product needs to be tested in the hospital, and it needs to be tested against people with different skin color and deformed hands. The edge cases need to be fixed and then there needs to be developed a way to "mass" produce the final product, this needs to include quality control. It also needs a way to prevent long down time when parts get broken. And an intuitive user interface and user manual needs to be created. And after this the product can start being mass adopted.

There is still a long way to go before the product is ready to be mass adopted. But there is not a long way before a prototype can be tested in the hospital.

Bibliography

[1] Paolo Cignoni. trimesh_sampling.cpp. URL https://github.com/cnr-isti-vclab/vcglib/blob/main/apps/sample/trimesh_sampling/trimesh_sampling.cpp.

[2] Torjus Aarsæther Dale, Astrid Nerhus; Hoseth. 3d-tilpasset støtteskinne ved håndleddsbrudd, 2019. URL http://hdl.handle.net/11250/2610888.

[3] Intel. *Intel® RealSenseTM LiDAR Camera L515*. Intel, 3 edition, 1 2021. URL https://www.intelrealsense.com/download/7691/.

[4] Intel. *Intel® RealSenseTM Product Family D400 Series*. Intel, 12 edition, 3 2022. URL https://www.intelrealsense.com/wp-content/uploads/2022/03/Intel-RealSense-D400-Series-Datasheet-March-2022.pdf.

[5] Michael Kazhdan1, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction, 2006. URL https://www.cs.jhu.edu/~misha/MyPapers/SGP06.pdf.

[6] Paul Steffen Kleppe, Andreas Fagerhaug Dalen, and Webjrn Rekdalsbakken. A novel way of efficient adaption of orthopaedic braces using 3d technology. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 345–350, 2018. doi: 10.1109/ICPHYS.2018.8387683.

[7] Webjørn Kleppe, Paul Steffen; Rekdalsbakken. Automatic production of patient adapted orthopaedic braces using 3d -modelling technology, 2020. URL https://hdl.handle.net/11250/2734372.

[8] Haakon Eduard Meyer. Fakta om beinskjørhet og brudd (osteoporose og osteoporotiske brudd). URL https://www.fhi.no/fp/folkesykdommer/beinskjorhet/beinskjorhet-og-brudd---fakta-om-os/.

[9] Stine Sønåsen Romundstad. Integrering av 3d teknologi for brukertilpasset gips ved Ålesund sykehus, 2020. URL https://hdl.handle.net/11250/2672393.

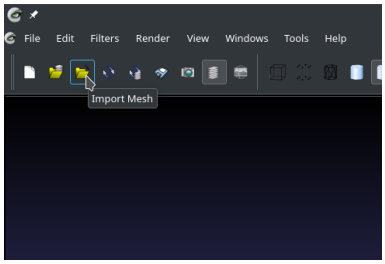
[10] Vegard Justsen André; Nedrelid Oscar Hatlo Sjåstad, Denis; Alvestad. Brukertilpasset gips, 2019. URL http://hdl.handle.net/11250/2613439.

[11] Carlo Tomasi. A simple camera model. URL https://courses.cs.duke.edu//fall16/compsci527/notes/camera-model.pdf.

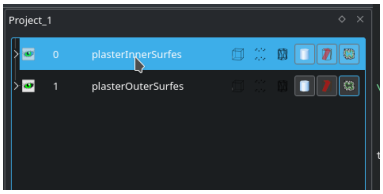
[12] Thijs van Lankveld. Scale-space surface reconstruction). URL https://doc.cgal.org/latest/Scale_space_reconstruction_3/index.html.

Appendices

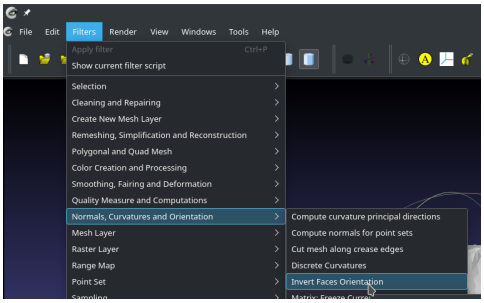
C Usermanual meshlab: flipping and merging mesh



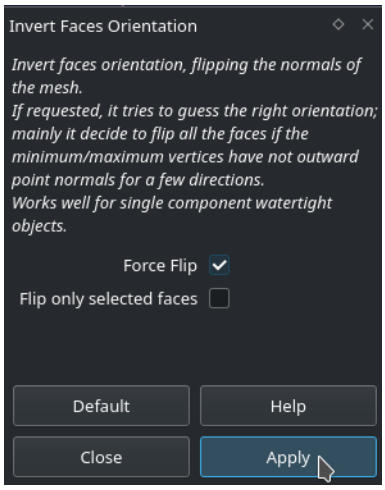
(a) : import plasterInnerSurfes.off and plasterOuterSurfes.off



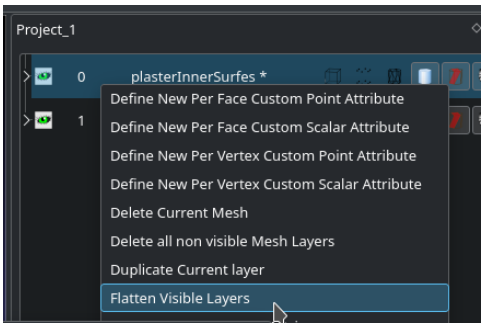
(b) : select plasterInnerSurfaces



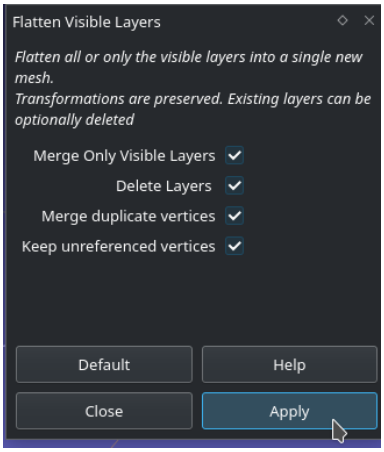
(c) : select invert Faces Orientation



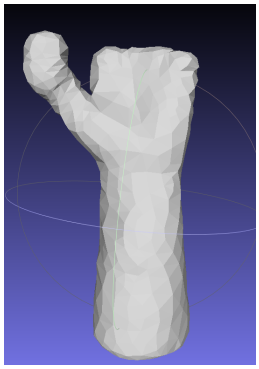
(d) : press apply once, it should turn the inside of the hand white



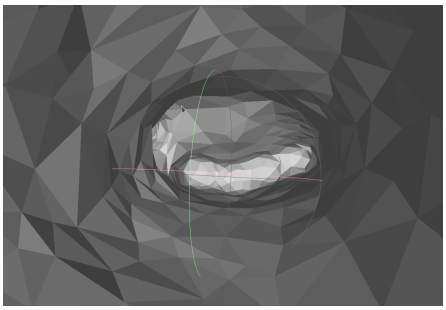
(e) : leftclick in select menu and flatten visible layers



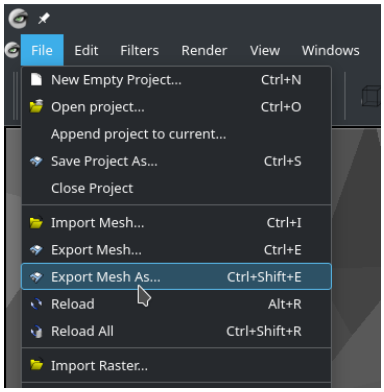
(f) : apply



(g) : check that the outside is white



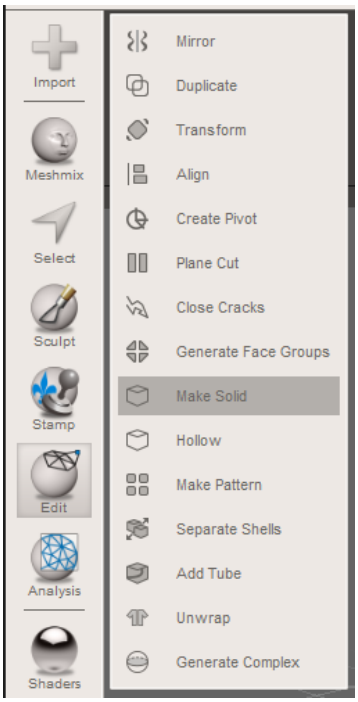
(h) : check that innside is white



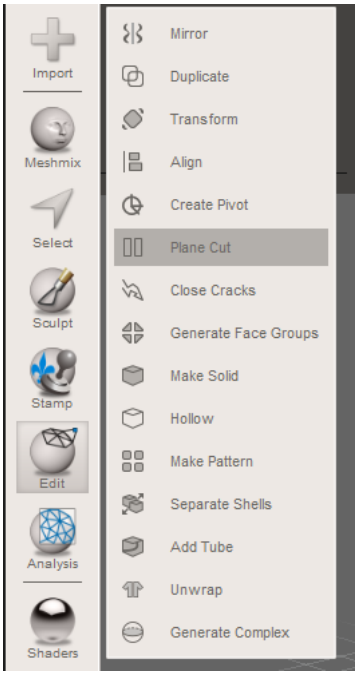
(i) : export mesh

Figure 1: Shows how to do the manual step in creating a shell around the hand. This shell is later going to be cut into pieces and then used as support for the broken hand

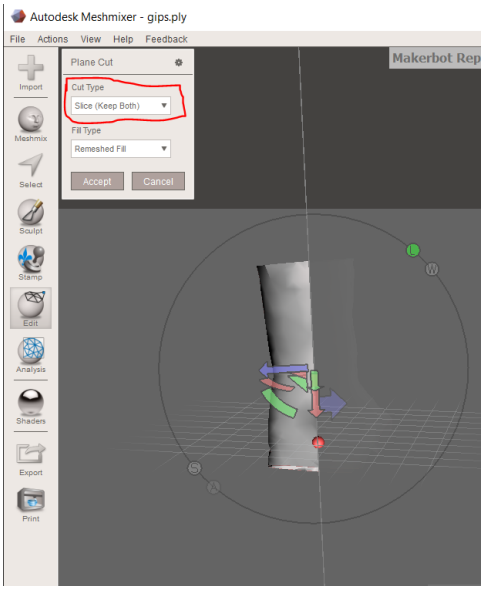
D Usermanual meshmixer: cutting the brace, and preparing it for printing



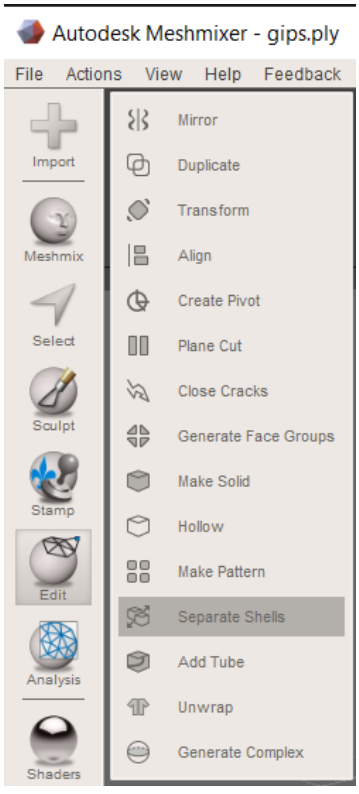
(a) : make solid, this increase the chanse that Prusasliser succsessfulli makes it into a step file



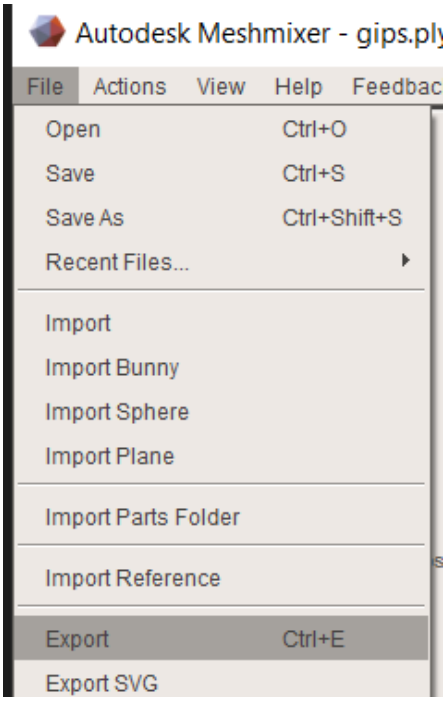
(b) : use "Plane Cut" to cut the mesh into the desired size



(c) : after cuting it, cut it one last time though the middel, but change the setting to "Slice (Keep Bouth)"



(d) : Seperate Shells



(e) : select one of the 2 parts, and then export it. Do the same with the other part

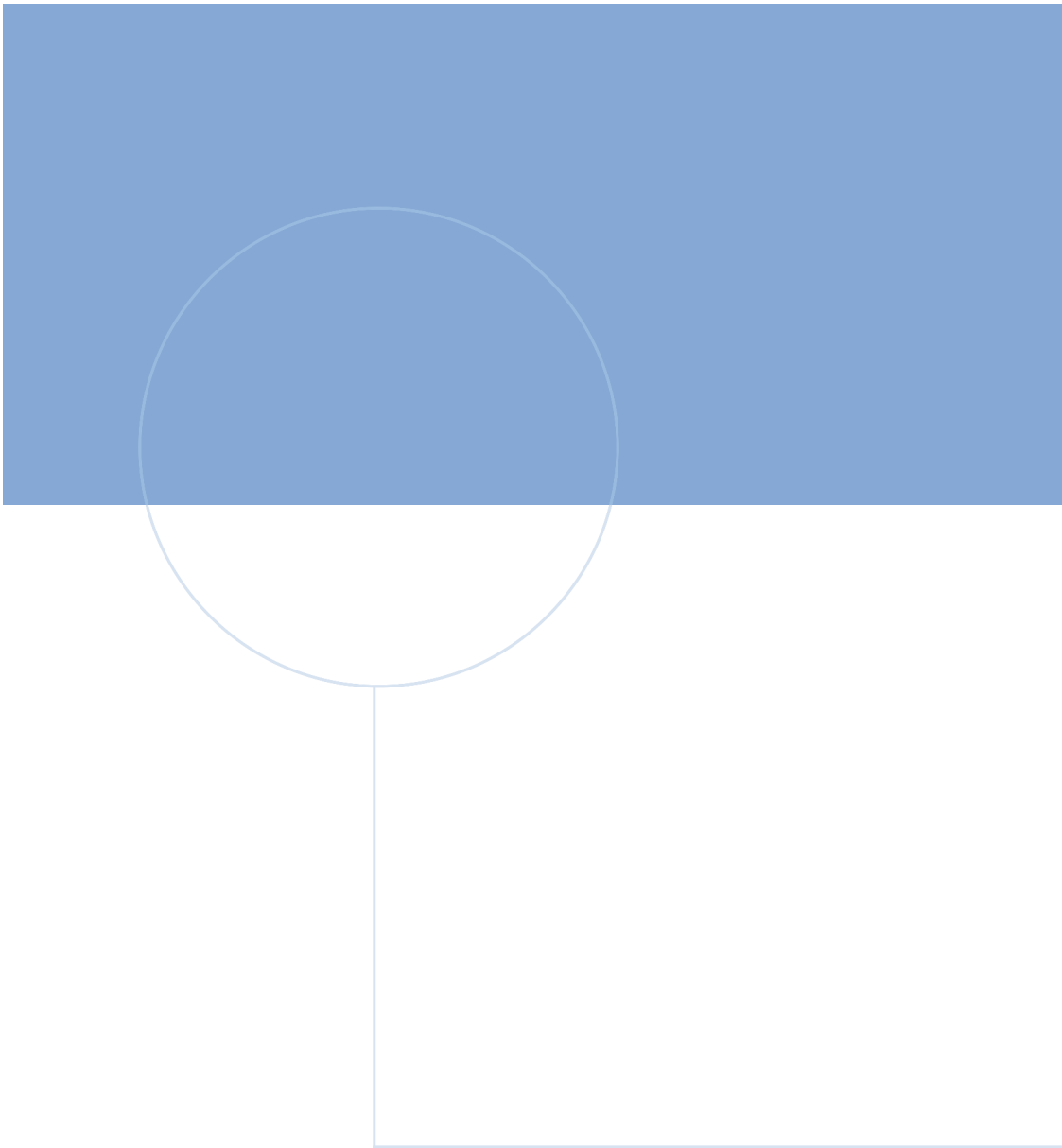
Figure 2: After this is done, convert the 2 parts into .step files, prusaSlicer was used to do this. After that just 3d print the step file

E code

The code used in this project is in a separate .zip folder.

F 3d models

The 3d model that was created from the automated system of participant 1 and 3 is in a separate .zip folder.



NTNU

Kunnskap for en bedre verden