DEGREE PROJECT IN MECHANICAL ENGINEERING,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2021

3D Scanner

Scanning small objects and recreating them visually as a mesh in a computer

CARL EGENÄS

AXEL SACILOTTO

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

3D Scanner

Scanning small objects and recreating them visually as a mesh in a computer

CARL EGENÄS, AXEL SACILOTTO

Bachelor's Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2021:39

Abstract

The purpose of this project was to construct a 3D scanner capable of scanning smaller objects and visualize them in a computer with satisfying accuracy. The goal was then to generate an STL file able to be 3D printed in an attempt at reverse engineering. Components, materials and tools were provided by KTH to the best of their ability and a budget of 1000 SEK was given to purchase components not available at KTH.

The scanner was designed using Solid Edge and utilizes two stepper motors to scan objects. One motor is used to rotate a platform that the object is placed upon and the second stepper motor is used to move an elevator on which a distance sensor is mounted. By keeping track of the elevator's height in conjunction with the rotation of the object, the distance measured by the sensor can be converted into a point in a Cartesian coordinate system.

Several different methods were tested in order to see how results varied. Firstly, the density of scanned points was increased, meaning that the sensor returned values more often as the stepper motor was rotating. Secondly, multiple measurements were made for a single point to determine an average distance and in that way reduce noise and uncertainty.

Placing a single laser sensor perpendicular to the object-rotating plate proved to be the optimal arrangement in terms of accuracy with the limited budget provided for this project. The scans are very time consuming which makes it important to decide whether to prioritize speed or accuracy.

Keywords: Mechatronics, 3D Scanner, Arduino, Stepper Motor, Mesh

Referat

3D-Scanner

Syftet med detta projekt var att konstruera en 3D-scanner kapabel att scanna mindre objekt och visualisera dem i en dator med tillfredsställande resultat. Målet var sedan att generera en STL-fil som går att skriva ut i 3D-skrivare för att försöka använda sig av reverse engineering. Komponenter, material och verktyg försågs av KTH så gott det gick och en budget på 1000 kr var tillgänglig för att inhandla komponenter som inte fanns på KTH.

Skannern designades med hjälp av Solid Edge och använder sig av två stegmotorer för att skanna object. En motor användes för att rotera den plattform som objektet placerades på och den andra stegmotorn användes för att flytta en hiss varpå en avståndssensor monterades. Genom att hålla koll på hissens höjd i kombination med rotationen av objektet kan avståndet som sensorn uppmäter konverteras till en punkt i det kartesiska koordinatsystemet.

Ett flertal metoder testades för att undersöka hur resultaten varierade. För det första ökades densiteten av skannade punkter, det vill säga sensorn returnerade värden oftare än stegmotorn roterade. För det andra genomfördes ett flertal mätningar för varje enskild punkt för att bestämma ett medelavstånd och på så sätt minimera brus och osäkerhet.

Att placera en enstaka lasersensor vinkelrätt mot objektroterande plattan visade sig vara det optimala arrangemanget för noggrannhet med den begränsade budgeten för det här projektet. Inskanningarna är väldigt tidskrävande vilket gör det viktigt att bestämma sig för att prioritera snabbhet eller noggrannhet.

Nyckelord: Mekatronik, 3D-Scanner, Arduino, Stegmotor, Mesh

Acknowledgements

We would like to thank our supervisor and examiner Nihad Subasic for providing lectures and guidance for the full duration of this project. Furthermore, we thank the assistants Amir Avdic and Malin Lundvall for sharing their knowledge and helping us in our times of need. We would also like to thank Staffan Qvarnström för ordering and providing material and components to all projects. Lastly we thank the School of Industrial Engineering and Management for providing us with various materials and all the equipment needed to complete the project.

Carl Egenäs, Axel Sacilotto
Stockholm, May 2021

Contents

List of Figures

List of Tables

List of abbreviations

3D	Three-Dimensional
ADC	Analog to Digital Converter
CAD	Computer Aided Design
CPU	Central Processing Unit
DAC	Digital to Analog Converter
DC	Direct Current
F	Farad
FOV	Field Of View
GND	Ground (electrical)
KTH	Kungliga Tekniska Högskolan (Royal Institute of Technology)
LiDAR	Light Detection And Ranging
OLED	Organic Light-Emitting Diode
RAM	Random Access Memory
ROM	Read Only Memory
SEK	Swedish Krona
STL	Standard Triangle Language (file format)
TOF	Time-Of-Flight
V	Volt
VMOT	Voltage Motor

Chapter 1

Introduction

This chapter contains an introduction to the relevant subjects, the purpose and scope of the project and a brief explanation of the method.

1.1 Background

Echolocation, the use of sound to locate objects, is in theory a simple concept that animals, like the bat, use in order to navigate [1] and hunt for food. Despite being the basis for old, yet advanced technologies like radar and sonar, it is not that widely available for the average consumer. Although, this is beginning to change as the 2020 iPhone 12 Pro series was launched with a LiDAR sensor, making the technology more accessible to the masses.

The story is similar for 3D printing, a technology that can trace its roots far back in time when humans manually "3D printed" clay pots [2] layer by layer, as seen in Figure 1.1. Today the process is automated, see Figure 1.2, and at the forefront of modern technology. However, 3D printing is often a complicated story that requires expensive equipment which makes it rare for ordinary people to be exposed to the technology outside of big industries. 3D printing has many applications today [3] including medical, aerospace and automotive.

Figure 1.1: *The creation of a clay pot [2]*

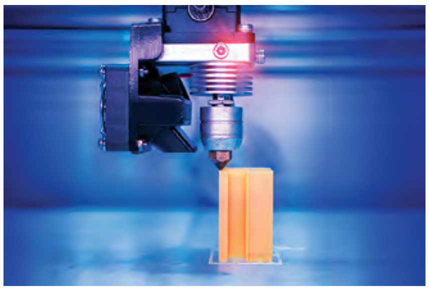


Figure 1.2: *A 3D printer in action [4]*

1.2 Purpose

The purpose of this project is to design and construct a device that, with the help of a distance sensor, can scan smaller objects and recreate them visually on a screen. That visual recreation will then be used to 3D print a copy of the object in an attempt at reverse engineering. The following three research questions will be explored in the project:

- With what degree of accuracy can the three-dimensional object be recreated in a computer?
- How long does it take to create the visualization of the object?
- What arrangement of sensors will yield the best results in terms of accuracy?

1.3 Scope

The 3D scanner created in this project will be a cheaper and simpler model than what is currently available on the market. It will contain resources supplied by KTH, including a microcontroller, cables and other components as well as access to tools, and a budget of 1000 SEK for additional purchases.

1.4 Method

The first step is to gather relevant information regarding sensors and other potential components for the project in order to understand the limitations and potential obstacles that might be encountered. The construction and coding of the scanner will be a process of trial and error apart from what can be planned in theory. To design the prototype, the 3D CAD program Solid Edge will be used, with some parts being 3D printed from there directly. Once the construction is completed, the code will be tested and altered based upon experiments in order to find the optimal solution or solutions for capturing the scanned object in detail.

Chapter 2

Theory

This chapter covers the relevant theory behind the project in general and specifically the theory relevant to our research questions.

2.1 An overview and important mechanisms

The 3D scanning device created in this project will be based on the mechanisms of the scanner depicted in Figure 2.1. It will consist of a rotating platform and an elevator mechanism, both of which will help scan the entire object. The object will sit on the platform and the distance sensor will be mounted on the elevator which will move up and down by rotating a threaded rod passing through a nut fastened in the middle of the elevator.

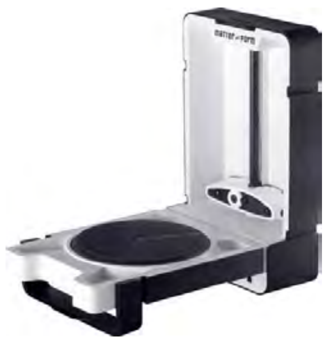


Figure 2.1: *The Matter and Form MFS1V1 3D Scanner [5]*

Both the rotation of the platform and the threaded rod will come from stepper motors. More about those can be found in Section 2.4. The motors and the distance sensor will be controlled by an Arduino UNO microcontroller, described in Section 2.3.

2.2 Distance sensor

A distance sensor is a component that measures the distance between itself and the nearest object straight ahead of the sensor by combining both a transmitter and receiver in one component. There are different ways of measuring distance and two common ones are explained in Section 2.2.1.

2.2.1 Scanning methods

Time-of-flight (TOF) is a method which uses a pulse [6] of laser or sound and calculates the distance by measuring the time it takes for the pulse to return. In the case of a laser sensor, the speed of light is used to calculate the distance travelled by the laser beam. It is calculated using Equation 2.1:

$$Distance = c \cdot \frac{t}{2}, \tag{2.1}$$

where c is the speed of light and t is the time it takes for the laser beam to return to the source.

Triangulation is a method optimized for short distance measurements [6] where great accuracy is required. It makes use of a laser diode and an image sensor [7] and calculates the angle at which the laser beam hits the sensor, as seen in Figure 2.2. It requires expensive image sensors to achieve such high accuracy.

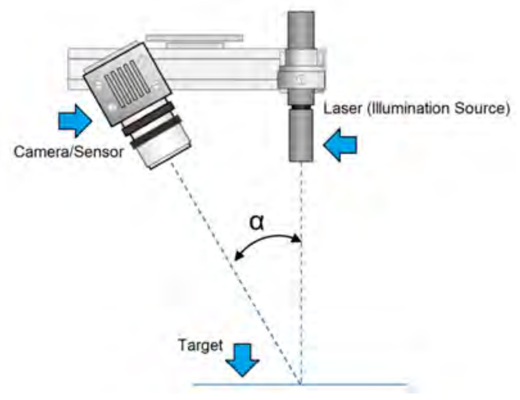


Figure 2.2: *The triangulation scanning method [8]*

2.3 Microcontroller

A microcontroller is a small computer optimized for certain simple tasks. This includes programmable analog and digital reading and writing and support for pulse width modulation. Functionality varies between models but microcontrollers generally consist of a central processing unit [9] (CPU), that can best be described as the brain of the controller, handling calculations and logic among other things. They also typically have input and output ports, memory in form of random access memory (RAM) and read only memory (ROM), analog to digital converters (ADC) and lastly digital to analog converters (DAC).

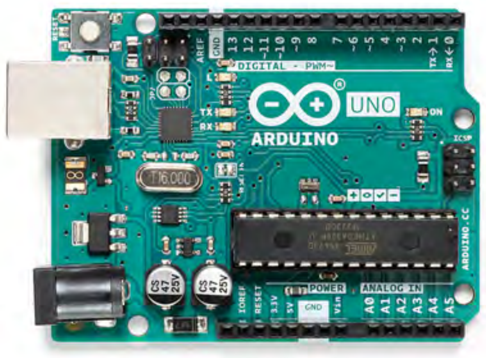


Figure 2.3: *Arduino UNO, the microcontroller board used in this project [10]*

2.4 Stepper motor

A stepper motor is a variant of the common DC motor and generally consists of a permanent magnet as a rotor surrounded by several electromagnets arranged in groups [11], or so called phases. By turning different phases on and off in a certain sequence a step is completed, hence the name step or stepper motor, see Figure 2.4. Since the rotation is done step by step, it is possible to accurately control

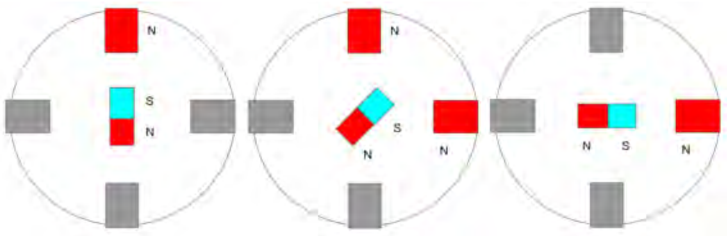


Figure 2.4: *Simplified version of a stepper motor. Motion left to right: Step, half step, next step. Made with Google Presentations*

the position of the motor without any feedback from sensors and achieve the same functionality [12] as the traditional servo motor. Furthermore, unlike the traditional servo motor, a stepper motor allows for an unlimited range of motion and is thus highly practical when accurate and continuous rotation is required. Lastly, it is also possible to have a partial step motion where the rotor is rotated a half, eighth or even smaller increments of a step, but in order to do so, a stepper motor driver is required.

2.5 Stepper motor driver

A stepper motor driver is a component that, unlike a microcontroller, can handle the relatively high amounts of current and voltage that a stepper motor requires and also allows for control of the motor with the use of a standard signal of 5 or 3,3 V from a microcontroller. Moreover, by giving signal to one or more of the *M* ports, see Figure 2.5, it is possible to divide the step resolution [13] into halves, eighths and all the way down to 1/32th thus giving even more control of the steps, see Figure 2.6. The driver has a potentiometer that can set a current limit to not exceed the current rating of the stepper motor. It needs a motor supply voltage of 8,2-45 V connected across VMOT and GND [14], but to avoid voltage spikes above 45 V, it is wise to put a large capacitor across VMOT and GND, see Figure 2.5.

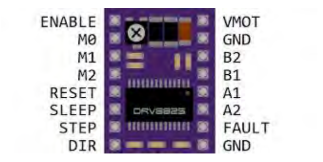
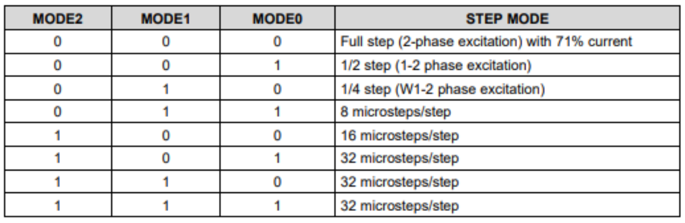


Figure 2.5: *DRV8825 stepper motor driver [15]*



MODE2	MODE1	MODE0	STEP MODE
0	0	0	Full step (2-phase excitation) with 71% current
0	0	1	1/2 step (1-2 phase excitation)
0	1	0	1/4 step (W1-2 phase excitation)
0	1	1	8 microsteps/step
1	0	0	16 microsteps/step
1	0	1	32 microsteps/step
1	1	0	32 microsteps/step
1	1	1	32 microsteps/step

Figure 2.6: *Table of how to achieve different levels of microstepping [13]*

2.6 OLED I2C SH1106 1.3" display

This display can be used do present different patterns, data or text sent from the Arduino microcontroller. It uses the SH1106 driver [16] and therefore the Adafruit_SH1106 library [17] is needed to connect the display to the arduino. The display is depicted in Figure 2.7.

Figure 2.7: *I2C OLED display [18]*

2.7 Visualization

To be able to create a three-dimensional visualization in a computer, software able to process the point data is needed. Each measurement from the distance sensor corresponds to a point in a Cartesian coordinate system. The point cloud, which is a large collection of points, will be converted to a mesh, as shown in Figure 2.8. The mesh is a collection of triangular connections [19] between the points in the point cloud. This creates a surface with the purpose of resembling the scanned object as closely as possible.

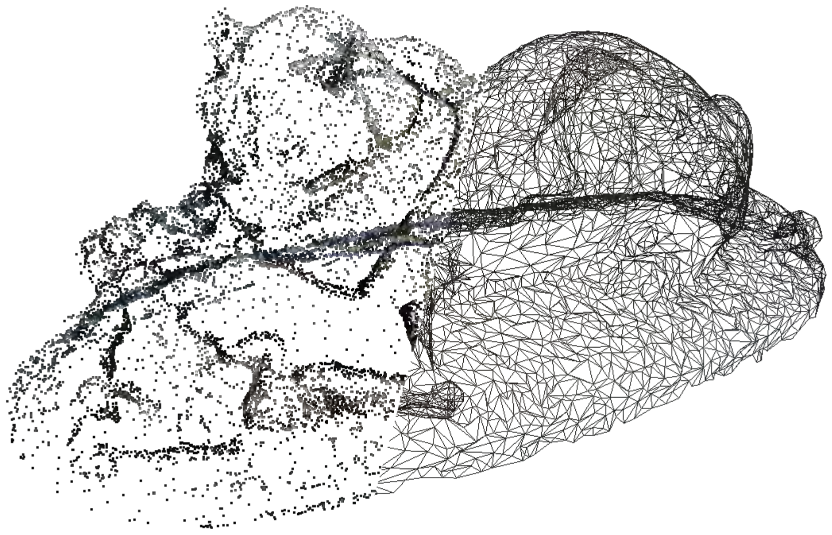


Figure 2.8: *An illustration of the conversion from a point cloud (left) to a mesh [20] (right)*

Chapter 3

Demonstrator

This chapter covers the design, simulation and construction of the demonstrator, the included circuitry as well as the software necessary to make everything work.

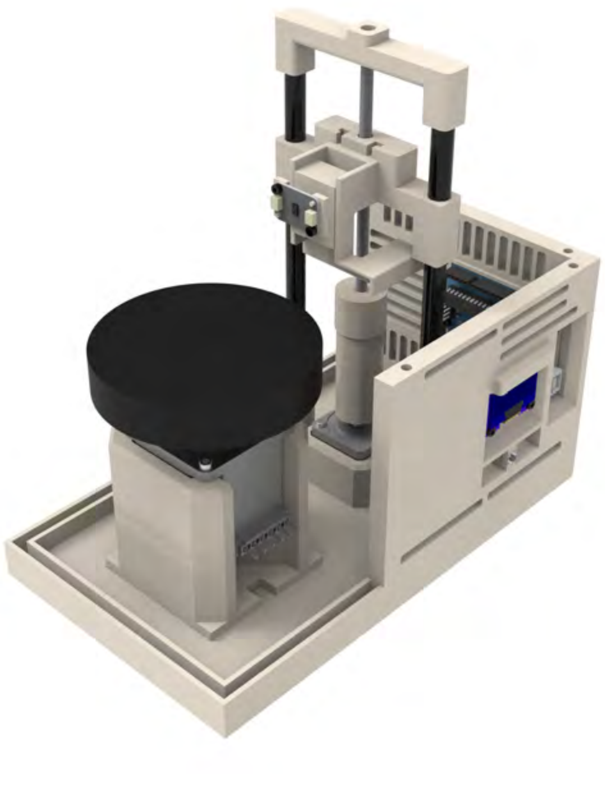


Figure 3.1: *The 3D Scanner designed in Solid Edge, image rendered in Keyshot*

3.1 Acumen Simulation

The movement of the elevator and the rotation of the object-rotating plate was simulated using Acumen [21] with the aim of visualizing the scanning process and confirm that everything moved and worked as intended. Acumen is a program based on the programming language Java and allows the user to simulate movements in an accessible environment that requires very little prior programming experience.

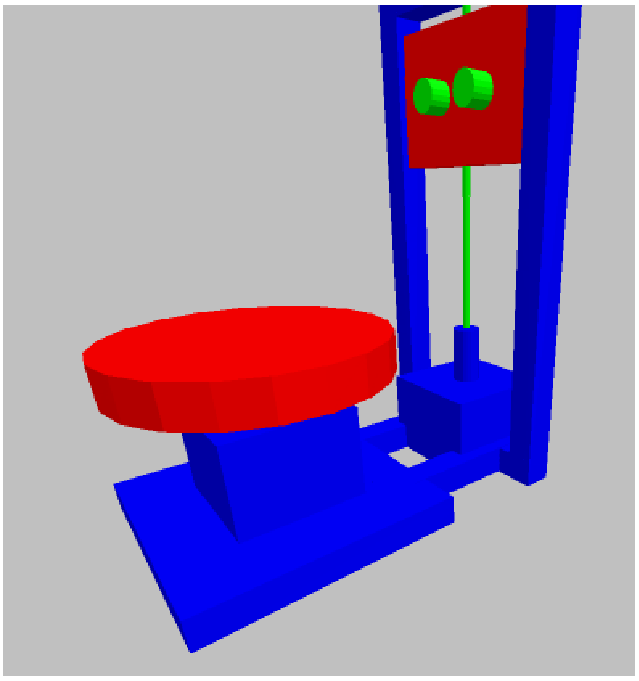


Figure 3.2: *A simulation of the object-rotating plate and the elevator mechanism. Picture taken in the Acumen environment*

Figure 3.2 shows an early simulation of the elevator (right) and object-rotating plate (left) based on a simplified version of the 3D-scanner with only the moving parts included.

3.2 Hardware

The hardware used to construct the prototype includes various parts designed in the 3D CAD program Solid Edge and those parts came to life through 3D printing. It also includes two stepper motors, an Arduino UNO microcontroller, a VL6180X distance sensor, three steel rods, one of which is threaded, a nut and screws of various sizes to fasten different components.

3.2.1 Base plate

The base plate was designed in Solid Edge and helps keep the whole structure in place. It is equipped with enclosing support for both stepper motors and the two steel rods on either side of the elevator (see Section 3.2.3). It is also surrounded by a slot to permit the installation of module-like walls, as seen in Figure 3.3. These walls are used to fasten, from left to right in Figure 3.3: the Arduino UNO, the OLED display and finally a breadboard.

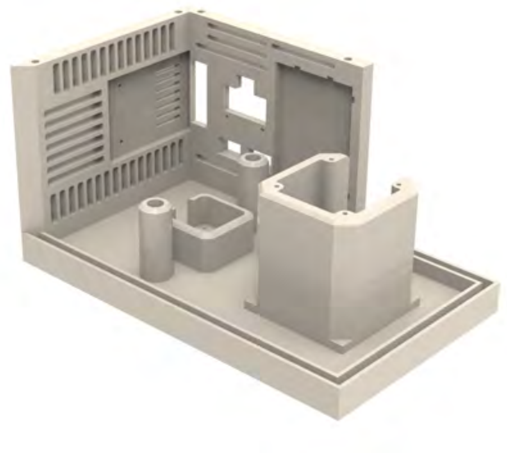


Figure 3.3: *The base plate with walls designed in Solid Edge, image rendered with Keyshot*

3.2.2 Spinning platform

The spinning platform is powered by a KH56KM2-801 stepper motor from Japan Servo which rotates 1,8° per step, or 200 steps per revolution. The object meant to be scanned will sit on the surface of the platform and rotate around the stepper motor's axis of revolution. The mechanism is portrayed in Figure 3.4.

The motor is larger and more powerful than necessary for most light objects, but this problem was counteracted in a couple of ways. Firstly, by adding friction to the spinning platform's surface and secondly, by reducing the motor's torque. The added friction to the surface helps the object stay in place and not slip when the motor takes very aggressive and distinct steps. Friction was added by using a worn XIOM Sigma I Pro table tennis rubber, as depicted in Figure 3.5. To maximize friction, the rubber was turned upside down, leaving the side with traces of glue facing up. The lateral area of the spinning platform was then wrapped with Butterly side tape, more commonly used to protect table tennis rubbers.

The motor's torque was reduced with the use of the stepper motor driver. Firsty, the potentiometer was turned to allow as little current as possible to pass to the motor while still being able to rotate normally. Secondly, the use of microsteps

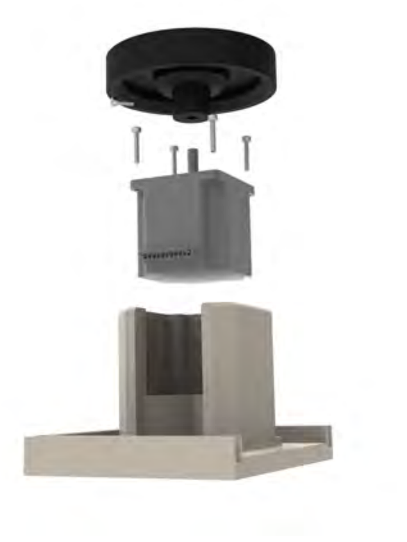


Figure 3.4: *The spinning mechanism, made in Solid Edge and rendered in Keyshot*

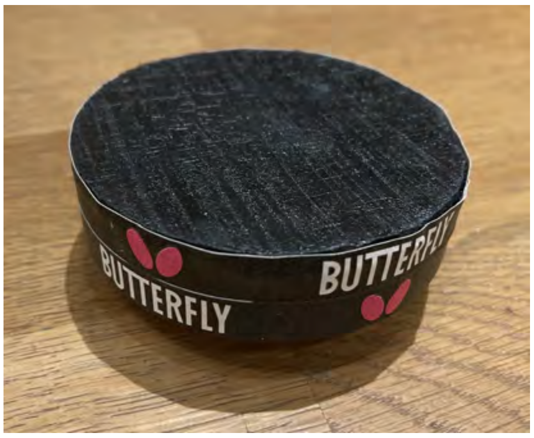


Figure 3.5: *A close-up image of the spinning platform and the rubber added to increase friction. Picture taken by Axel Sacilotto*

was implemented, forcing the motor to take 16 microsteps for every step, further reducing its torque.

The stepper motor needs 200 steps to complete a full revolution. This means that an object with an average radius perpendicular to the axis of rotation of r, will be measured in increments of 1/200th of its circumference. This means the average difference between measured points on the surface of the object can be calculated with the following formula:

$$d_{step} = \frac{2}{200}\pi r. \tag{3.1}$$

If r assumes the typical value of 3 cm, it will result in an average distance between points of 0,94 mm. If this does not yield a satisfying result, measurements can be taken after a certain amount of microsteps instead of a whole step.

3.2.3 Elevator mechanism and distance sensor

The elevator is powered by a TS3214N61 stepper motor from Tamagawa Seiki and its only purpose is to move the distance sensor up and down to collect all the necessary point data. On either side of the stepper motor there is a steel rod to guide the sensor and keep it in place. The vertical motion of the elevator is made possible by spinning a threaded rod through a nut resulting in a moved distance equal to the lead of the thread per full revolution. The transmission of the motor's torque to the threaded rod is achieved through the use of a shaft coupling designed in Solid Edge. All the components that make up the elevator are shown in Figure 3.6.

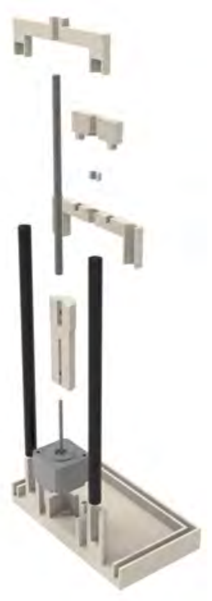


Figure 3.6: *The elevator design, made in Solid Edge and image rendered in Keyshot*

The threaded rod is of size M6 with a lead of 1 mm. This means that for every complete revolution of the stepper motor, the distance sensor will move vertically 1 mm in the desired direction. Thus it would be possible to rotate the stepper motor fewer steps and move the distance sensor an even shorter distance before making the next measurement if an even higher resolution is desired.

The sensor used in this project is the VL6180X sensor from Adafruit, shown in Figure 3.7. It is a laser TOF sensor with a range between 5-100 mm [22].

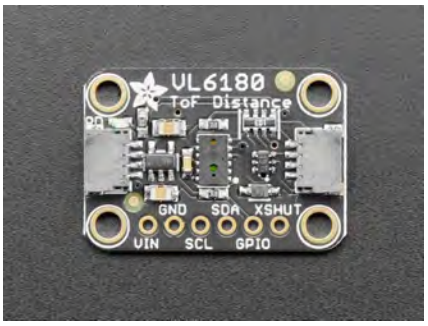


Figure 3.7: *VL6180X LiDAR distance sensor from Adafruit [23]*

The sensor is fastened as shown in Figure 3.8. It makes use of several small modules to easily remove and replace the sensor with a different one if circumstances change. It is fastened by sliding it into the slots on the elevator and also comes with a small tunnel to guide the wires. The design was first destined for the VL53L0X distance sensor from M5Stack, as seen in Figure 3.9. The reason was that it had a simple design for mounting with its Lego-compatible holes [24]. Though the measuring range of the VL53L0X is up to two meters and therefore the VL6180X is more suitable for small distances like this with its maximum range of 100 mm.

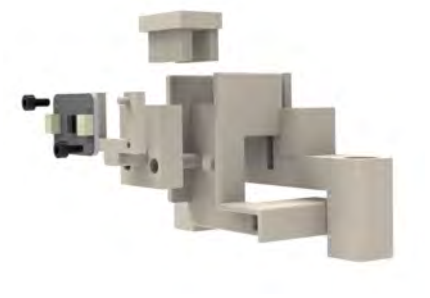


Figure 3.8: *How the sensor is mounted on the elevator, made in Solid Edge, image rendered in Keyshot*

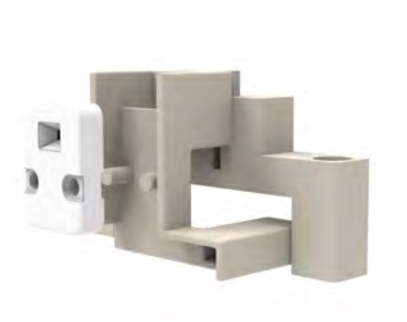


Figure 3.9: *How the VL53L0X sensor from M5Stack could be mounted on the same elevator, designed in Solid Edge, image rendered in Keyshot*

3.2.4 Circuitry

The circuitry consists of one Arduino uno, two DVR8825 stepper motor drivers, one 470μF capacitor, a TS3214N61 stepper motor, a KH56KM2-801 stepper motor, one I2C OLED display, one 12V battery and a VL6180X LiDAR distance sensor. Figure 3.10 shows how the components are connected to each other. Most wiring was soldered using multistranded wires and the stepper motor drivers were connected using a breadboard.

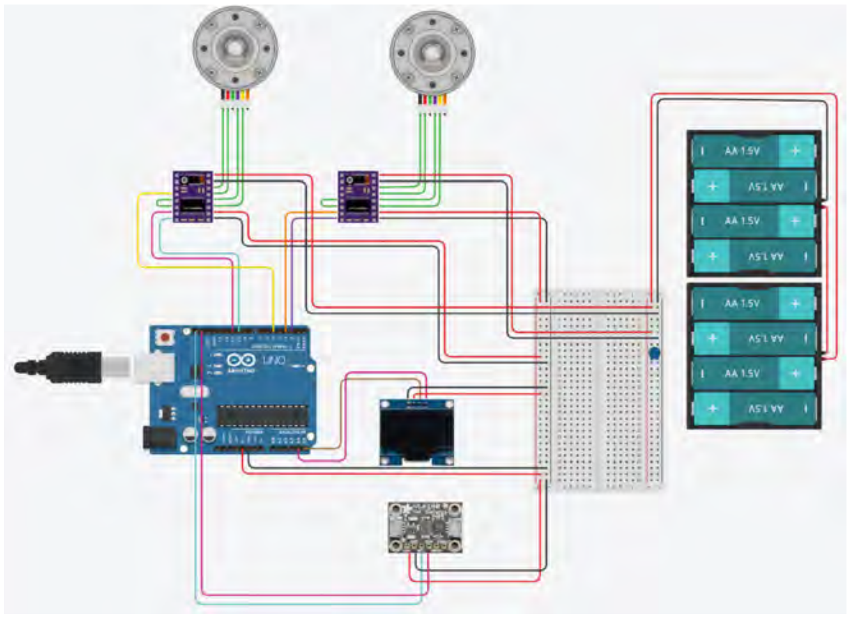


Figure 3.10: *The circuitry, made with Tinkercad and Google Presentations*

3.3 Software

This section will cover the Arduino code and the software used to import the point data and convert it to a mesh ready to be 3D printed.

3.3.1 Collecting data points

The data collecting software is represented by the flow chart in Figure 3.12 and is based around a Cartesian coordinate system that is centered in the middle of the object-rotating plate. Since the height, angle and distance are all known or measurable parameters but are not known in terms of x, y and z, a translation is needed. As shown in Figure 3.11, a point on the rotating plate can be interpreted as a point on a cylinder and therefore cylindrical coordinates are used to determine x, y and z:

$$\begin{cases} x = r \cdot \cos\theta & (3.2) \\ y = r \cdot \sin\theta & (3.3) \\ z = z & (3.4) \end{cases}$$

where θ is the known rotation of the object, z is the known height of the elevator/sensor and the radius r is calculated by:

$$r = \textit{distance to plate center} - \textit{measured distance}. \qquad (3.5)$$

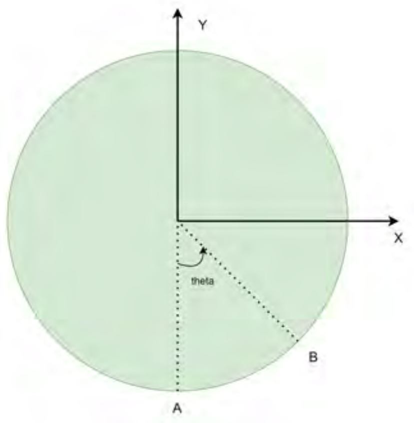


Figure 3.11: *Object-rotating plate (green). Point A is the measured point and point B is the coordinate calculated adjusted for the rotation, made with draw.io*

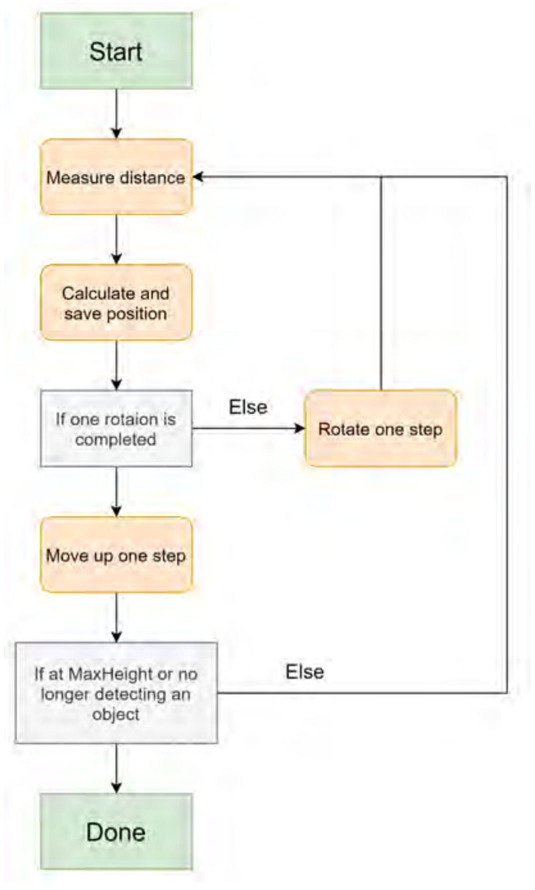

Start
Measure distance
Calculate and save position
If one rotaion is completed
Else
Rotate one step
Move up one step
If at MaxHeight or no longer detecting an object
Else
Done

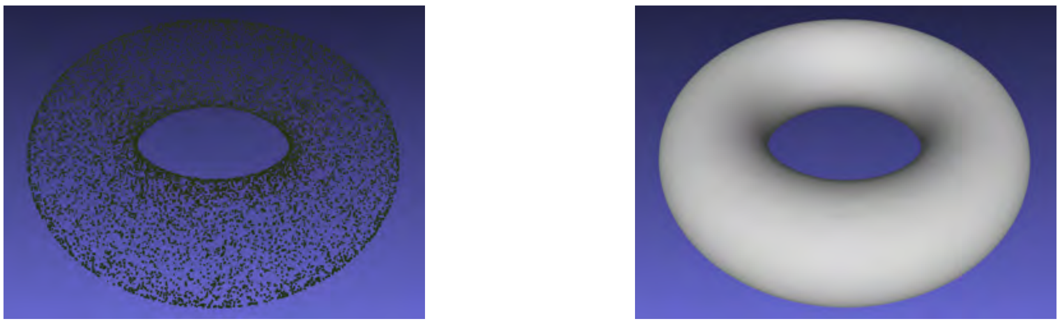
Figure 3.12: *Flow chart of the program that collects and saves data points, made with draw.io*

3.3.2 Saving data points

In order to save the collected data points to a readable document, a text-file for example, CoolTerm by Roger Mier's Freeware [25] is used. CoolTerm allows the user to save all the information that is printed to the Arduino's serial monitor and is thus very useful in this project.

3.3.3 Visualization

In order to visualize the object and initiate the reverse engineering process in the computer, the points are imported as a text file with each row containing an x-, y-, and z-coordinate. This project made use of Meshlab [26] to convert the points to a mesh and export it as an STL-file able to be 3D printed. A generic example of this process can be seen in Figure 3.13 where 10,000 points were randomly generated on the boundaries of a parameterization of a torus.



(a) *Points* ***(b)*** *Mesh*

Figure 3.13: *An example of the conversion from points to a mesh using Meshlab, pictures taken in Meshlab*

Chapter 4

Experiments

This chapter covers experiments regarding accuracy of the distance sensor and time requirements to complete scans of different accuracy.

4.1 Distance sensor accuracy

The accuracy of the sensor was tested by placing an object at a fixed distance and taking 1000 measurements. This was in order to get a large enough sample size for the experiment. Table 4.1 shows the experiment where the 1000 measurements were first evaluated on their own and then put into groups of five, ten, 50 and 200 to calculate averages. All results of these five methods were produced by a program written in the Python language and every measurement was plotted using Matplotlib [27]. The graphs and the python code can be found in Appendix A.

Measurements	Average [mm]	Max [mm]	Min [mm]	Spread (Max-Min) [mm]
One	43,966	48	40	8
Five	43,966	46	42,2	3,8
Ten	43,966	45,1	42,9	2,2
Fifty	43,966	44,48	43,54	0,94
Two hundred	43,966	44,29	43,645	0,645

Table 4.1: *Experiment of the sensor's accuracy and how to reduce uncertainty by using an average of multiple measurements.*

The experiment shows that in order to get a good rendition of the scanned object, using an average of several measurements for each point might be required. The spreads of the different methods were plotted to visualize the trend of how the spread decreases with an increased amount of measurements. This is depicted in Figure 4.1 where the trend shows that the spread decreases by about a factor of four for every tenfold increase in the amount of measurements, although it slows down a bit when approaching 200.

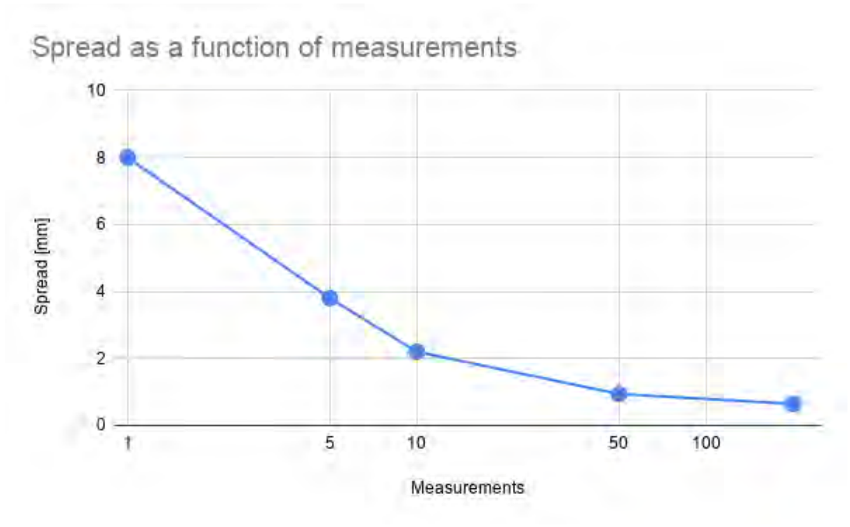


Figure 4.1: *The spread as a function of the amount of measurements per average, from Google Sheets*

However, more measurements require more time to complete and in order to determine if the increased accuracy justifies the added time, an additional experiment was conducted, described in Section 4.2.

4.2 Scan duration

The scan duration was measured by scanning a single layer of an object, meaning a full rotation of the object on the same z-coordinate. This measurement was made for the same categories as in Table 4.1 and the results are listed in Table 4.2.

Measurements per point	Time per rotation [s]
One	6,865
Five	16,963
Ten	27,457
Fifty	111,925
Two hundred	426,864

Table 4.2: *Time necessary to complete a full rotation with different accuracy.*

The total time to complete a full scan will be roughly equal to the duration of a single rotation multiplied with the amount of layers. The times from Table 4.2 can be seen plotted in Figure 4.2 where a clear linear trend can be seen. This is to be expected since the time required to take a measurement is independent from the rest of the program and more or less constant at similar distances.

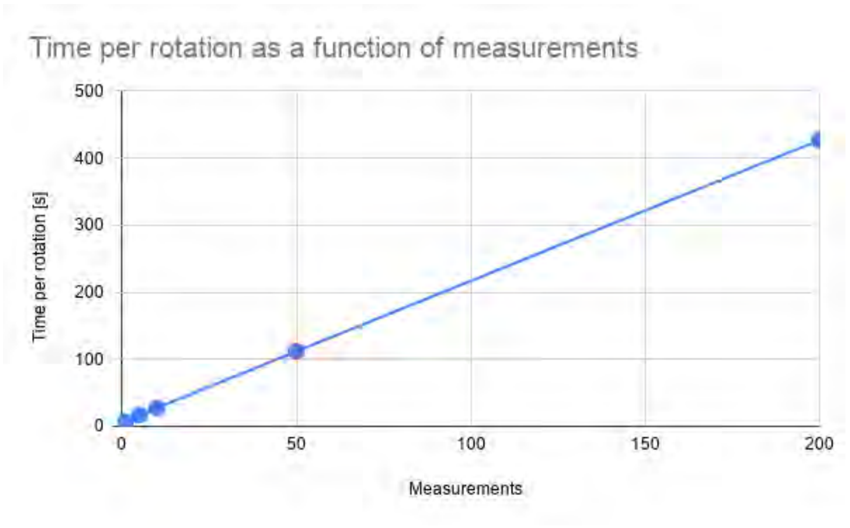


Figure 4.2: *The time required to complete a full rotation plotted against the amount of measurements used to get a final value for each point, from Google Sheets*

Chapter 5

Results

This chapter covers the results of scanning a 32 mm tall french madeleine cake, as shown in 5.1, with varying layer heights and numbers of measurements per step. Figures 5.2 through 5.5 contain images of scanning results, all of which were taken in the Meshlab environment.

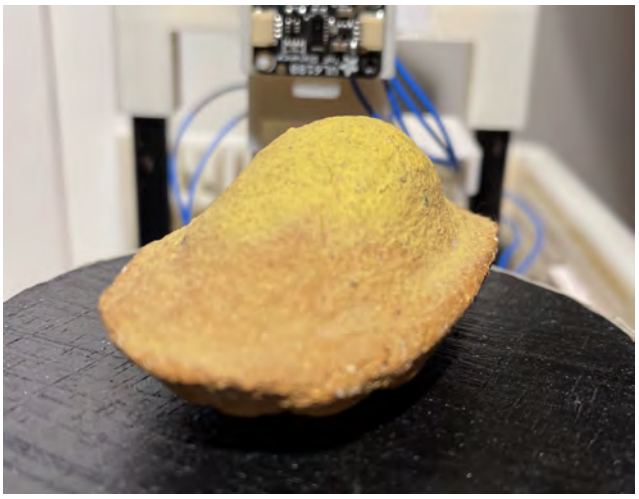
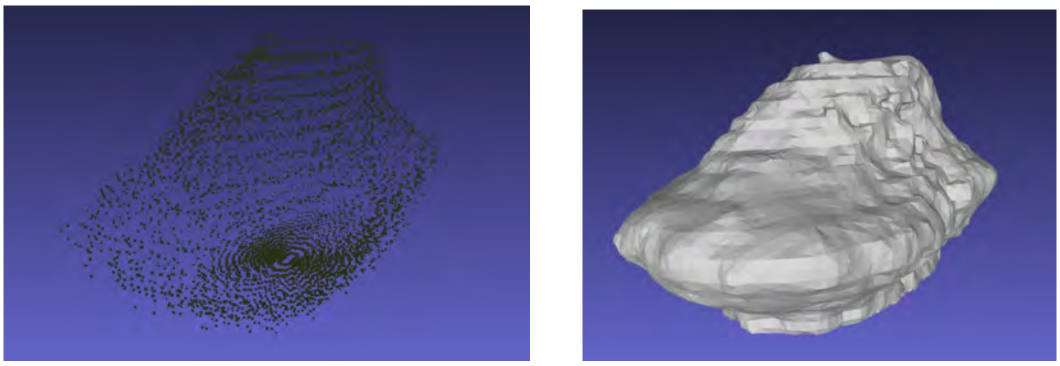
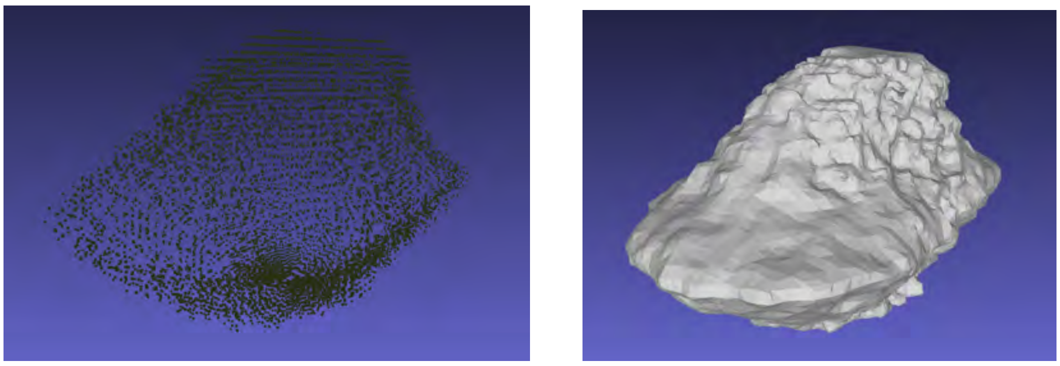


Figure 5.1: *A stone model of a madeleine cake, the object used to compare different levels of accuracy visually. Picture taken by Axel Sacilotto*



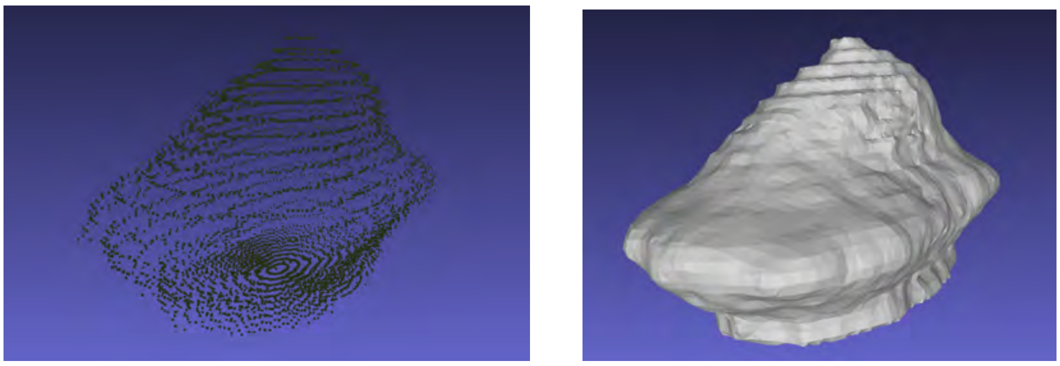
(a) Points *(b) Mesh*

Figure 5.2: *Scan 1: 1 measurement per point and 2 mm between horizontal layers*



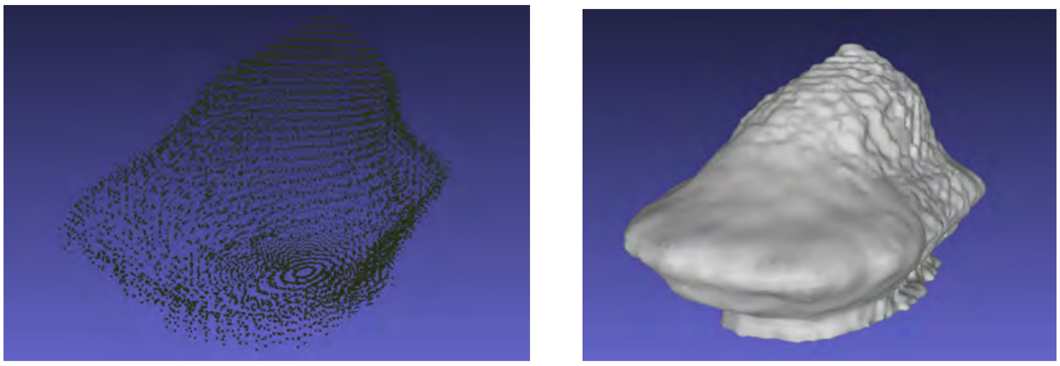
(a) Points *(b) Mesh*

Figure 5.3: *Scan 2: 1 measurement per point and 1 mm between horizontal layers*



(a) Points *(b) Mesh*

Figure 5.4: *Scan 3: 5 measurements per point and 2 mm between horizontal layers*



(a) Points *(b) Mesh*

Figure 5.5: *Scan 4: 5 measurements per point and 1 mm between horizontal layers*

	Measurements per point	Layer height [mm]	Time to complete scan
Scan 1	1	2	3 min 56 s
Scan 2	1	1	6 min 41 s
Scan 3	5	2	6 min 58 s
Scan 4	5	1	12 min 11 s

Table 5.1: *Time necessary to complete 3D scan of the madeleine model with different levels of accuracy*

Since the madeleine cake is 32 mm tall while laying on the platform, as in Figure 5.1, the device needs to scan 16 layers for scan 1 & 2 and 32 layers for scan 3 & 4.

Chapter 6

Discussion

6.1 Accuracy of recreation

The results showed that in order to display a good rendition of the object, it is crucial to collect a lot of data points and also do so accurately. Figures 5.2 and 5.3 showed that a single measurement per point gave reasonable accuracy but at the cost of a rough and almost mountain-like surface. Thicker layers were preferred in this case since the noise-like nature of the surface became amplified when layers were close together, as shown in Figure 5.3.

The most accurate recreations were achieved when taking the average of several measurements per point as shown in Figure 5.5, in combination with a thin layer height. Using a five measurement average captured the largest amount of detail, and if time had permitted would have been the version to be 3D printed, but still showed some noise-like tendencies on the surface. This phenomenon emerges because of the spread of measurements, but as shown in Table 4.1, the spread could be reduced by increasing the number of measurements taken per point. Reducing the spread should result in even better accuracy but the scanning time would be greatly increased as well.

Lastly, the accuracy could also be increased by lowering the layer thickness in combination with a large number of measurements per point. There is also the possibility of scanning more than 200 points per revolution of the platform by taking advantage of the microsteps beyond just decreasing the torque. To conclude, there are numerous ways to increase the accuracy of the scan but they all come at the cost of increasing scanning time, perhaps to an unsustainable level.

6.2 Sensors and sensor arrangement

During the initial stages of the project, ultrasonic distance sensors or combinations of several sensors in order to get the most accurate measurements possible were considered, but in the end the decision was made to use a single LiDAR sensor instead. The reason for not going with an ultrasonic distance sensor was mainly because of the large field of view that a normal ultrasonic sensor possesses, varying between models but up to 15° in some [28] cases. A large FOV would not allow the sensor to distinguish between points situated closely together on the object, resulting in a distorted recreation.

The results showed that a single LiDAR sensor gave decent accuracy, and that turning away from an ultrasonic sensor was a good choice. The results also confirmed the suspicion that using the average of a number of measurements improved accuracy greatly, as suggested in experiment 4.1.

Chapter 7

Conclusion and future work

7.1 Conclusion

To conclude the project, the research questions stated in section 1.2 are answered:

- *With what degree of accuracy can the three-dimensional object be recreated in a computer?*

There are many ways to tweak different parameters to increase accuracy and the results can range from a bit noisy to a fair degree of accuracy. With only one measurement per point, one can expect an error of about ±4 mm and with an average of five measurements per point, the error is reduced to ±1,9 mm. The way to deal with the error of the sensor is simply by taking the average of a large enough sample of measurements for every point. This is the most important thing since unevenness due to low density of points can be smoothed out in post processing with the right knowledge. If 200 measurements were to be used to get a good average for every point, the error would be reduced to ± 0,3225 mm.

- *How long does it take to create the visualization of the object?*

The simplest scan tested in this project took 3 min 56 s for a height of 32 mm, while the longest took 12 min 11 s. Depending on the height of the object and the expected accuracy, the duration of the scan could land somewhere between these two times but it would take significantly longer to scan a tall object with a high degree of accuracy, say with an error below ± 1 mm. There are many parameters in the Arduino code that can be changed to impact the scanning time but to get a sufficiently accurate scan, one can expect to scan for 10 minutes at the very least.

- *What arrangement of sensors will yield the best results in terms of accuracy?*

In terms of using the approach of a single TOF sensor, using a short distance LiDAR sensor was the preferable method over, for example, an ultrasonic sensor. It is hard to compare to other systems since none were tested, but in theory it would seem that the biggest obstacle was money and that the approach of this project was the superior one.

7.2 Future work

There are a couple of things that could be changed in order to improve the scanner in the future. First and foremost, the long scanning time is arguably the biggest problem and methods to shorten it by reducing the time needed to collect all the data points would be a welcomed addition to the project. The argument could be made that arranging several sensors around the object would speed up the scanning process while retaining a good level of accuracy since each sensor would only need to cover a small part of the object. However, if such an alteration is made, the elevator could require additional support to keep it stable by incorporating at least two stepper motors in the elevator design, placing them on opposite sides of the object-rotating plate.

Another alteration that could be made is to replace the large stepper motor used to rotate the object with a less powerful version since the motor was too powerful in most cases and required both tuning and a sticky surface in order to keep objects from falling of the edge. Using a smaller motor would also allow for the design to be more compact and user friendly. This is something that could be greatly improved since this design was not very practical but rather was only focused on the goal of being able to scan objects successfully.

Lastly, some quality of life improvement could be made adding buttons for starting, stopping and resetting the scanning process as well as upgrading the OLED display and the programming in order to visualize the object being scanned in real time on the display.

Bibliography

[1] Bat Conservation Trust. *Flight, food and echolocation.* URL: https://www.bats.org.uk/about-bats/flight-food-and-echolocation. (accessed: 2021-05-07).

[2] Tomorrow's world today. *A Brief History of Pottery.* URL: https://www.tomorrowsworldtoday.com/2018/08/01/a-brief-history-of-pottery/. (accessed: 2021-02-15).

[3] 3D Printing Industry. *The Free Beginners Guide.* URL: https://3dprintingindustry.com/3d-printing-basics-free-beginners-guide/#01-basics. (accessed: 2021-04-01).

[4] Paul Godfrey. *3D Printing: The shape of things to come.* URL: https://satelliteprome.com/interviews/3d-printing-the-shape-of-things-to-come/. (accessed: 2021-02-15).

[5] Conrad.se. *Matter and Form MFS1V1 3D-scanner.* URL: https://www.conrad.se/p/matter-and-form-mfs1v1-3d-scanner-1555134. (accessed: 2021-04-01).

[6] Johan Moberg. "3d scanner: Accuracy, performance and challenges with a low cost 3d scanning platform". Degree project, first cycle, 15 credits. KTH Royal Institute of Technology, 2017, pp. 2–34. URL: https://www.diva-portal.org/smash/get/diva2:1200549/FULLTEXT01.pdf.

[7] João Guilherme DM França et al. "A 3D scanning system based on laser triangulation and variable field of view". In: *IEEE International Conference on Image Processing 2005.* Vol. 1. IEEE. 2005, pp. I–425. DOI: https://doi.org/10.1109/ICIP.2005.1529778.

[8] Movimed. *What is Laser Triangulation?* URL: https://www.movimed.com/knowledgebase/what-is-laser-triangulation/. (accessed: 2021-04-09).

[9] Electronics Hub. *Basics of Microcontrollers – History, Structure and Applications.* URL: https://www.electronicshub.org/microcontrollers-basics-structure-applications/. (accessed: 2021-05-05).

[10] Antratek. *Arduino UNO.* URL: https://www.antratek.com/arduino-uno. (accessed: 2021-04-01).

[11] Additional information regarding stepper motors. *Stepper motors.* URL: `https://www.explainthatstuff.com/how-stepper-motors-work.html`. (accessed: 2021-04-01).

[12] Information regarding the inner workings of a stepper motor. *How a stepper motor works.* URL: `https://howtomechatronics.com/how-it-works/electrical-engineering/stepper-motor/`. (accessed: 01.04.2021).

[13] Texas Instruments. *DRV8825 Stepper Motor Controller IC datasheet.* URL: `https://www.electrokit.com/uploads/productfile/41016/drv8825.pdf`. (accessed: 2021-05-08).

[14] Pololu. *DRV8825 Stepper Motor Driver Carrier, High Current.* URL: `https://www.pololu.com/product/2133`. (accessed: 2021-04-01).

[15] Makerguides. *Arduino UNO.* URL: `https://www.makerguides.com/drv8825-stepper-motor-driver-arduino-tutorial/`. (accessed: 2021-05-08).

[16] Learn Robotics. *OLED Display with Arduino Tutorial.* URL: `https://www.learnrobotics.org/blog/oled-display-arduino/`. (accessed: 2021-04-20).

[17] wonho-maker. *Adafruit_SH1106.* URL: `https://www.learnrobotics.org/blog/oled-display-arduino/`. (accessed: 2021-04-20).

[18] makerlab-electronics. *I2C OLED display.* URL: `https://www.makerlab-electronics.com/product/1-3-i2c-oled-display/`. (accessed: 2021-05-07).

[19] A Kumar, PK Jain, and PM Pathak. “Industrial application of point cloud/STL data for reverse engineering”. In: *DAAAM International Scientific Book* (2012), pp. 445–462. DOI: `https://doi.org/10.2507/daaam.scibook.2012.38`.

[20] Ph.D. Florent Poux. *5-Step Guide to generate 3D meshes from point clouds with Python.* URL: `https://towardsdatascience.com/5-step-guide-to-generate-3d-meshes-from-point-clouds-with-python-36bad397d8ba`. (accessed: 2021-02-15).

[21] Acumen. URL: `http://www.acumen-language.org/p/download.html`. (accessed: 2021-04-01).

[22] ST. *Proximity and ambient light sensing (ALS) module.* URL: `https://cdn-learn.adafruit.com/assets/assets/000/037/608/original/VL6180X_datasheet.pdf`. (accessed: 2021-04-15).

[23] Adafruit. *Adafruit VL6180X Time of Flight Distance Ranging Sensor (VL6180) - STEMMA QT.* URL: `https://www.adafruit.com/product/3316`. (accessed: 2021-05-03).

[24] M5Stack. *M5Stack VL53L0X datasheet.* URL: `https://www.elfa.se/Web/Downloads/_t/ds/U010_eng_tds.pdf`. (accessed: 2021-04-01).

[25] Roger Meier. *CoolTerm.* URL: `https://freeware.the-meiers.org/`.

[26] Meshlab. URL: `meshlab.net`. (accessed: 2021-04-01).

[27] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: https://doi.org/10.1109/MCSE.2007.55.

[28] cdon.se. *Ultrasonic distance sensor HC-SR04*. URL: https://cdon.se/bygg-verktyg/ultraljuds-distans-matare-hc-sr04-passar-till-arduino-p49319705?gclid=CjwKCAjwkN6EBhBNEiwADVfya4G7AWFAKbk3dVSdIcweU9DnmykmRZcda-CgWXJ5Y28-zaUlv0Ql0RoC-YYQAvD_BwE&gclsrc=aw.ds#fo_c=1753&fo_k=140e53412bd846816291b3b8fe4b9fc7&fo_s=gplase. (accessed: 2021-04-25).

Appendix A

Measurement Accuracy Experiment

This appendix presents graphs of all measurements from the experiment in Table 4.1 as well as the python code to generate the graphs and the key figures presented in the aforementioned table.

A.1 Python code

```
""" Testing the accuracy of the VL6180X sensor

    By: Carl Egenäs, Axel Sacilotto
    Course: MF133X - Bachelor's thesis in mechatronics
    Date: 2021-05-09

    This code takes a text file of measurements and examines how the
    measurement error can be reduced by putting values in groups of
    different sizes and using the average of values in each group.
    This simulates taking multiple measurements for each point and
    then using the average of those measurements to determine the
    distance to that point.

 """
f = open("staticTestCoords.txt", "r") # File with 1000 measurements

#All measurements in one array:
one = f.readlines()

#Arrays with values after an average has been calculated from every
group of five, ten, 50 and 200 measurements:
five = []
ten = []
fifty = []
```

```
twohun = []

Measurements are entered into an array:
for i in range(0, len(one)):
    one[i] = int(one[i])

counter = 0

#Groups of five
sum_5 = 0
for x in one:
    sum_5 += x
    counter += 1
    if(counter==5):
        five.append(sum_5/5)
        counter = 0
        sum_5 = 0

#Groups of ten:
sum_10 = 0
for y in five:
    sum_10 += y
    counter += 1
    if(counter==2):
        ten.append(sum_10/2)
        counter = 0
        sum_10 = 0

#Groups of 50:
sum_50 = 0
for z in ten:
    sum_50 += z
    counter += 1
    if(counter==5):
        fifty.append(sum_50/5)
        counter = 0
        sum_50 = 0

#Groups of 200:
sum_200 = 0
for w in fifty:
    sum_200 += w
    counter += 1
    if(counter==4):
        twohun.append(sum_200/4)
        counter = 0
        sum_200 = 0

#This function returns the average of ALL measurements in the
selected array, the smallest and largest value, and the spread
between the largest and smallest:
def avMinMax(numbers):
    return sum(numbers)/len(numbers), min(numbers), max(numbers),
    max(numbers)-min(numbers)

av_one, min_one, max_one, spread_one = avMinMax(one)
av_five, min_five, max_five, spread_five = avMinMax(five)
av_ten, min_ten, max_ten, spread_ten = avMinMax(ten)
av_fifty, min_fifty, max_fifty, spread_fifty = avMinMax(fifty)
av_twohun, min_twohun, max_twohun, spread_twohun = avMinMax(twohun)

#Prints the results of the above function for all arrays:
print('av_one: ', av_one)
print('min_one: ', min_one)
print('max_one: ', max_one)
print('spread_one: ', spread_one)
print('')
print('av_five: ', av_five)
print('min_five: ', min_five)
print('max_five: ', max_five)
print('spread_five: ', spread_five)
print('')
print('av_ten: ', av_ten)
print('min_ten: ', min_ten)
print('max_ten: ', max_ten)
print('spread_ten: ', spread_ten)
print('')
print('av_fifty: ', av_fifty)
print('min_fifty: ', min_fifty)
print('max_fifty: ', max_fifty)
print('spread_fifty: ', spread_fifty)
print('')
print('av_twohun: ', av_twohun)
print('min_twohun: ', min_twohun)
print('max_twohun: ', max_twohun)
print('spread_twohun: ', spread_twohun)

#Plots all values of the selected array:
```

```
import matplotlib.pyplot as plt
plt.plot(one) #Switch for each graph
plt.ylabel('Measurement value [mm]')
plt.xlabel('Measurements in chronological order')
plt.suptitle('No average') #Switch for each graph
plt.show()
```

A.2 Graphs

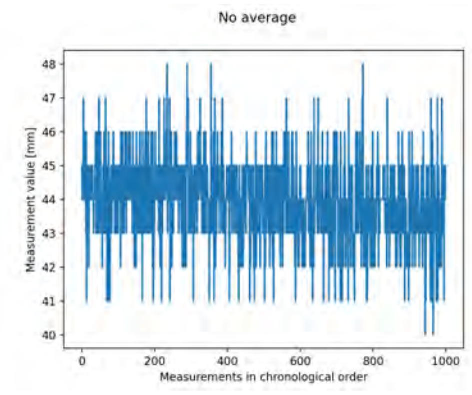


Figure A.1: *Counting every measurement*

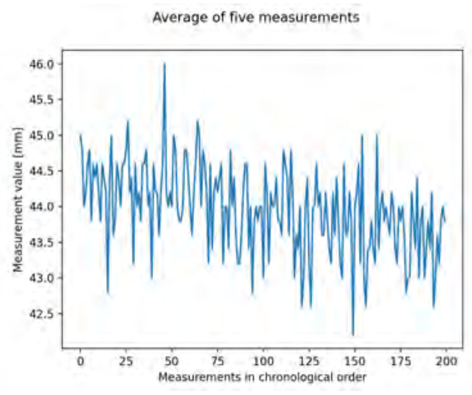


Figure A.2: *Average of five measurements*

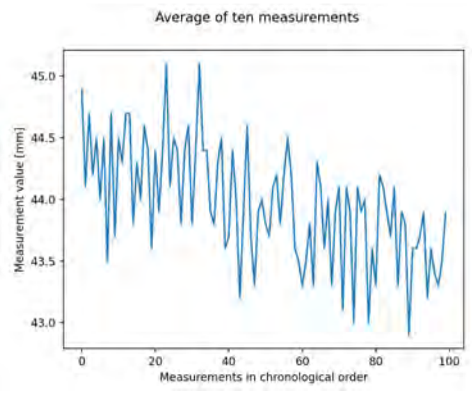


Figure A.3: *Average of ten measurements*

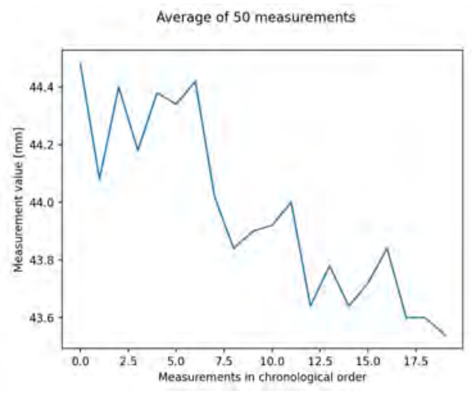


Figure A.4: *Average of 50 measurements*

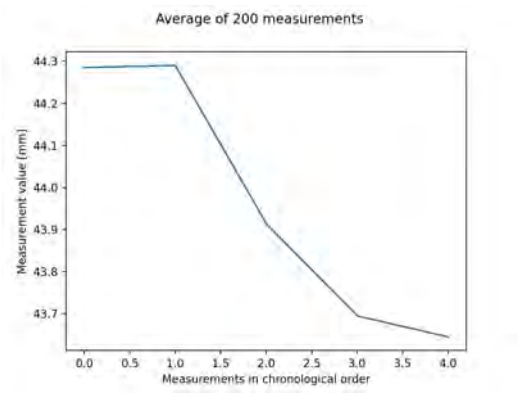


Figure A.5: *Average of 200 measurements*

Appendix B

Arduino code

```
/*
Degree project in mechanical engineering, first cycle, 15 credits
Course: MF133X - Bachelor's thesis in mechatronics
By: Carl Egenäs, Axel Sacilotto
Date: 2021-05-09

This code controls the entire process of the scanner.
It starts by finding the surface of the spinning platform
to begin scanning and then scans the object layer by layer
while storing all the scanned points. When the sensor no
longer detects an object or the elevator has reached its
highest point, the scanning ends and the OLED display shows
the duration of the scan.

The Wire.h and VL6180X.h libraries were downloaded from the
Arduino library manager.
The Adafruit_SH1106.h library was downloaded from Github via:
https://github.com/wonho-maker/Adafruit_SH1106
 */
// Distance sensor
#include <Wire.h>
#include <VL6180X.h>
VL6180X sensor;

//Stepper for object
#define dirPin_object 11
#define stepPin_object 10
#define microPin 5

// Stepper for elevator
```

```
#define dirPin_Z 2
#define stepPin_Z 3

//OLED display
#include <Adafruit_SH1106.h>
#define OLED_RESET 4
Adafruit_SH1106 display(OLED_RESET);

//Steps per revolution for the stepper motors
const int StepsPerRevolution = 200;

//Number of measurements to take the average of when scanning each point
const int MeasNum = 5;
const float RadPerStep = 2*PI/StepsPerRevolution; //Radians per step
const int microSteps = 16;
const int layerHeight = 1; //Levels the elevator moves up every time
unsigned long startTime;
unsigned long endTime;
unsigned long scanTime;

float CurrentRad_object = 0; //Keeps track of the angle the object has rotated
int CurrentZ = 0; //Keeps track of the location of the elevator
double MaxHeight = 120; // measured in mm as the maximum heigh of the scanned object
int lead = 1; //lead of the threaded rod in the elevator
int minDist = 17; //Smaller than this means the sensor is below the platform
int dis2Origo = 53; //Distance to origo (measured with the distance sensor)
double x; //For the coordinates
double y;
int z;

double Distance = 0; //Measured distance
boolean Done = false; //If the scan is done
boolean start = true; // If we are in the starting phase and haven't started scanning
boolean tooFarDown = false;
boolean tooFarUp = false;
boolean firstLayer = true;
boolean endScreen = false;

void setup() {
  Serial.begin(9600);

  Wire.begin();
  sensor.init();
  sensor.configureDefault();
  sensor.setTimeout(500);

  pinMode(dirPin_object, OUTPUT);
  pinMode(stepPin_object, OUTPUT);
  pinMode(dirPin_Z, OUTPUT);
  pinMode(stepPin_Z, OUTPUT);

  display.begin(SH1106_SWITCHCAPVCC, 0x3C);

  display.display();
  delay(2000);
  display.clearDisplay();

}

void loop() {
  if(start == true){
    Start(); //Find start position before starting the scan
  }
  if(Done != true){ //When at start position, begin scanning
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(0,15);
    display.println("Scanning");
    display.println("object");
    display.display();
    Done = true;
    digitalWrite(dirPin_object, LOW);

    //Rotate the object a full revolution and measure the distance at every step
    oneRev();
    digitalWrite(dirPin_Z, HIGH);
    UpOneStep(); //Then move on to the next layer
    firstLayer = false;
    //UpOneStep();
  }
  else if(endScreen != true){ //If Done, display the time it took and stop scanning
    endScreen = true;
    endTime = millis();
    scanTime = (endTime-startTime)/1000;
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(0,15);
    display.println("Done!");
    display.print("Duration: ");
    display.print(scanTime);
    display.println("s");
    display.display();
    //Serial.println(" DONE!!! ");
  }
}
void Start(){
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0,15);
  display.println("Finding");
  display.println("start");
  display.println("position");
  display.display();

  findHome();

  start = false;
  CurrentZ = 0;
  UpOneStep();
  startTime = millis();
}
void findHome(){ //Find the edge of the platform to start the scan
  for(int a = 0; a<5; a++){
    Distance += sensor.readRangeSingleMillimeters();
  }
  Distance = Distance / 5;
  //Serial.println(Distance);
  if(Distance < minDist){
    Distance = 0;
    tooFarDown = true;
    digitalWrite(dirPin_Z, HIGH);
    UpOneStep();
    if(tooFarUp == false){
      findHome();
    }
  }
  else{
    Distance = 0;
    tooFarUp = true;
    if(tooFarDown == false){
      digitalWrite(dirPin_Z, LOW);
      UpOneStep();
      findHome();
    }
  }
}

//Measures the desired amount of points and gives the global
// variable Distance the value to be used
void measure(){
  for(int k=0; k<MeasNum; k++){
        Distance += dis2Origo - sensor.readRangeSingleMillimeters();
      }
      Distance = Distance/MeasNum; // Calculate average
}
void oneRev(){
  for(int j=0; j<StepsPerRevolution; j++){
    measure();
    //Serial.println(Distance);
    if(Distance<dis2Origo+5){
      Done = false;
      CoordCalculator(Distance);
      Distance = 0;
    }
    RotOneStep();
  }
}

void RotOneStep(){ //This goes through the microsteps for every step
  digitalWrite(microPin, HIGH);
  for(int m = 0; m < microSteps; m++){
    digitalWrite(stepPin_object, HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPin_object, LOW);
    delayMicroseconds(1000);
  }
    CurrentRad_object +=RadPerStep;
}

void UpOneStep(){ // Moves the elevator up one level
  for(int j = 0; j<layerHeight*StepsPerRevolution; j++){
    digitalWrite(stepPin_Z,HIGH);
    delay(4);
    digitalWrite(stepPin_Z,LOW);
    delay(4);
  }
  delay(200);
  CurrentZ += layerHeight*lead;
}

//Takes the radius measured and converts it to x-, y-, and z-coordinates
void CoordCalculator(int dist){
  x = dist * cos(CurrentRad_object);
  y = dist * sin(CurrentRad_object);
  z = CurrentZ;
  Serial.print(x);
  Serial.print("; ");
  Serial.print(y);
  Serial.print("; ");
  Serial.println(z);
  if(firstLayer){

    //Draws a bottom layer to get a stable foundation for the object
    for(double c = 1; c>0; c-=0.1){
      Serial.print(c*x);
      Serial.print("; ");
      Serial.print(c*y);
      Serial.print("; ");
      Serial.println(0);
    }
  }
}
```

Appendix C

Acumen simulation

```
/*
#### Simulating the 3D Scanner ####

By: Carl Egenäs, Axel Sacilotto
Course: MF133X - Bachelor’s thesis in mechatronics
Date: 2021-05-09

Models the 3D scanner and simulates the important motion.
*/

model Support(pos) =
initially
_3D = ()
always
_3D = ( // Rods + elevator stepper motor
        Box center = pos + (0,0,7) // blue rods
        color = blue
        size = (4,0.5,0.5)
        rotation = (pi/2,0,0),

        Box center = pos + (0,0,-0.5)
        color = blue
        size = (4,0.5,0.5)
        rotation = (pi/2,0,0),

        Box center = pos + (1.75,0,3.25)
        color = blue
        size = (8,0.5,0.5)
        rotation = (0,pi/2,0),
        Box center = pos + (-1.75,0,3.25)
        color = blue
        size = (8,0.5,0.5)
        rotation = (0,pi/2,0),

        // Rods between stepper motor with the red hat and the elevator
        Box center = (1,1.25,-0.5) + pos
        color = blue
        size = (0.5,3,0.5)

        Box center = (-1,1.25,-0.5) + pos
        color = blue
        size = (0.5,3,0.5),

        // Stepper Motor for elevator
        Box center = pos
        color = blue
        size = (2,2,1.5),
        Cylinder center = pos + (0,0,1)
        color = blue
        size = (1.5,0.25)
        rotation = (pi/2,0,0),

        // Long green rod that connects to the stepper motor
        Cylinder center = pos + (0,0,3.5)
        color = green
        size = (7.5,0.0625)
        rotation = (pi/2, 0,0)
        )

model Hiss(pos, a, D) =
initially
_3D = ()
always
_3D = ( // Red elevator
        Box center = pos + (0,0,3) + D // +D for movement upwards
        color = red
        size = (3,0.25,2),

        // Distance sensor
        Cylinder center = pos + (0.5,0.25,3) + D // Moving the entire thing
        color = green
        size = (0.5,0.25),
        Cylinder center = pos + (-0.5,0.25,3) + D
        color = green
        size = (0.5,0.25)
        )

model Servo(pos,a) =
initially
_3D = (), _Plot=()
always
_3D = (Box center = pos +(0,0,1) // "Stepper motor"
       color = blue
       size = (2,2,1.5),

       Box center = pos // Baseplate
       color = blue
       size = (4,4,0.5),

       // Attachment between stepper motor and the red plate
       Cylinder center = pos +(0,0,2)
       color = blue
       size = (1.5,0.25)
       rotation = (pi/2,0,0),

       Cylinder center = pos +(0,0,2.5) // Red plate
       color = red
       size = (0.6,2)
       rotation = (pi/2,0,a)
       )

model Main(simulator) =
initially
// Creating objects and defining movement

c1 = create Support((0,0,0.5)),
c2 = create Hiss((0,0,0.5),0,(0,0,0)),
c3 = create Servo((0,4,0),0),

p = 0 , p' = 0, // prim ( ' ) equals velocity.
a = 0, a' = 0
always

a' = 0.1,
p' =0.25,
c2.D = (0,0,p),  // Movement of elevator
```

```
c3.a = a // Rotation of the red plate
```

Appendix D

Steps taken in Meshlab to generate an STL file

This series of images shows how to turn a text file of coordinates into an STL file using Meshlab. All images are taken in the Meshlab environment.

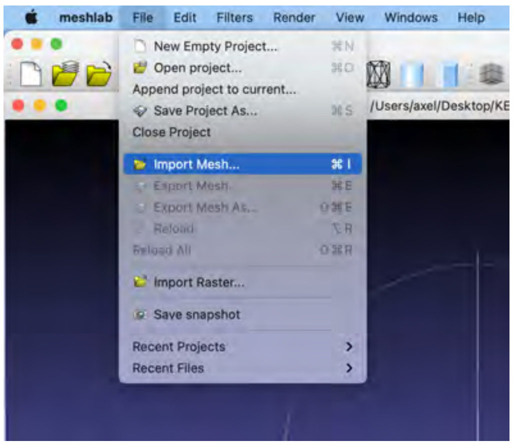
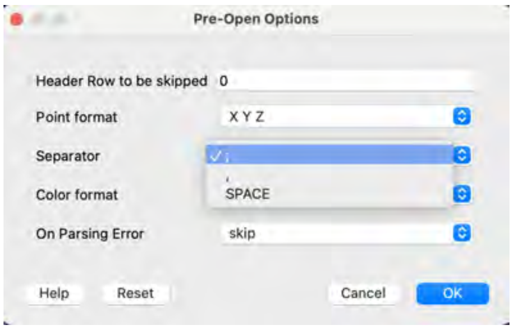


Figure D.1: *Step 1: Import a mesh*


Pre-Open Options

Header Row to be skipped: 0

Point format: X Y Z

Separator: ✓ ;

,

SPACE

Color format

On Parsing Error: skip

Figure D.2: *Step 2: Choose a text document with each row containing x- y and z-coordinates separated by either semi-colon, comma or space. Choose the appropriate separator for your file*

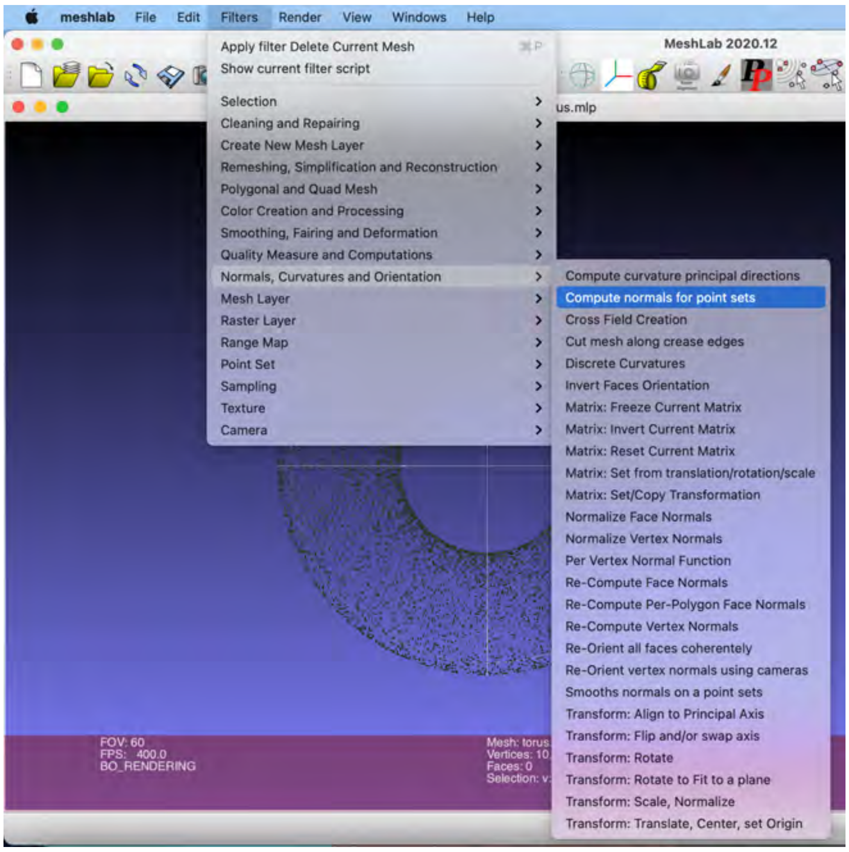
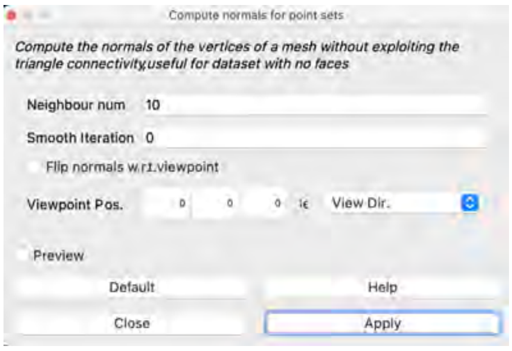

meshlab
File
Edit
Filters
Render
View
Windows
Help
MeshLab 2020.12
Apply filter Delete Current Mesh
Show current filter script
Selection
Cleaning and Repairing
Create New Mesh Layer
Remeshing, Simplification and Reconstruction
Polygonal and Quad Mesh
Color Creation and Processing
Smoothing, Fairing and Deformation
Quality Measure and Computations
Normals, Curvatures and Orientation
Mesh Layer
Raster Layer
Range Map
Point Set
Sampling
Texture
Camera
Compute curvature principal directions
Compute normals for point sets
Cross Field Creation
Cut mesh along crease edges
Discrete Curvatures
Invert Faces Orientation
Matrix: Freeze Current Matrix
Matrix: Invert Current Matrix
Matrix: Reset Current Matrix
Matrix: Set from translation/rotation/scale
Matrix: Set/Copy Transformation
Normalize Face Normals
Normalize Vertex Normals
Per Vertex Normal Function
Re-Compute Face Normals
Re-Compute Per-Polygon Face Normals
Re-Compute Vertex Normals
Re-Orient all faces coherentely
Re-Orient vertex normals using cameras
Smooths normals on a point sets
Transform: Align to Principal Axis
Transform: Flip and/or swap axis
Transform: Rotate
Transform: Rotate to Fit to a plane
Transform: Scale, Normalize
Transform: Translate, Center, set Origin
FOV: 60
FPS: 400.0
BO_RENDERING

Figure D.3: *Step 3: When you see the points on the screen it is time to compute normals as shown here*



Compute the normals of the vertices of a mesh without exploiting the triangle connectivity,useful for dataset with no faces

Neighbour num 10

Smooth Iteration 0

Flip normals w.r.t.viewpoint

Viewpoint Pos. 0 0 0 View Dir.

Preview

Figure D.4: *Step 4: Use the default settings and choose "Apply"*

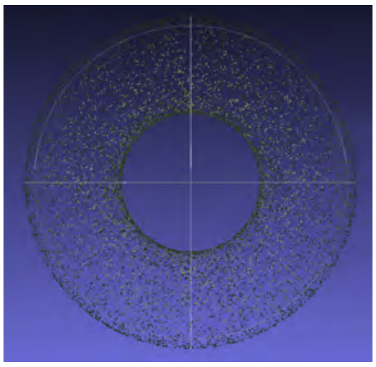


Figure D.5: *Step 5: Now it should look something like this*

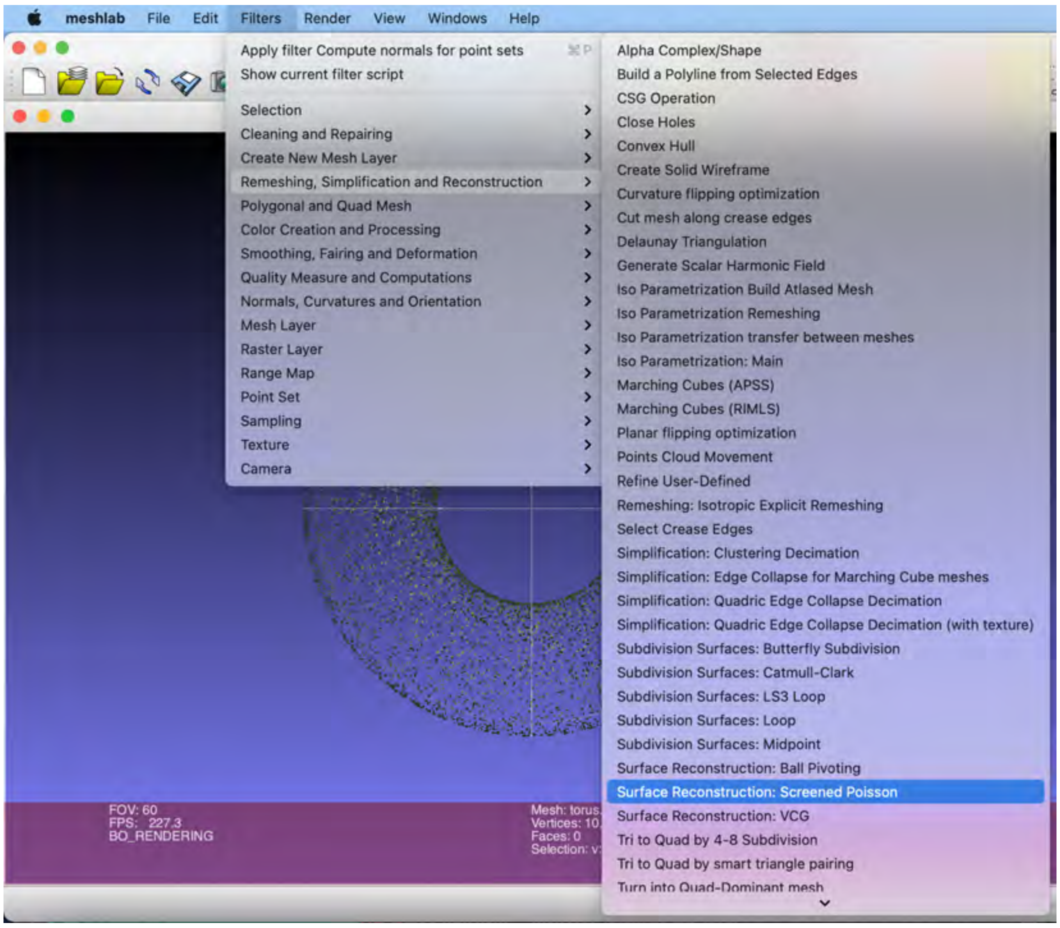
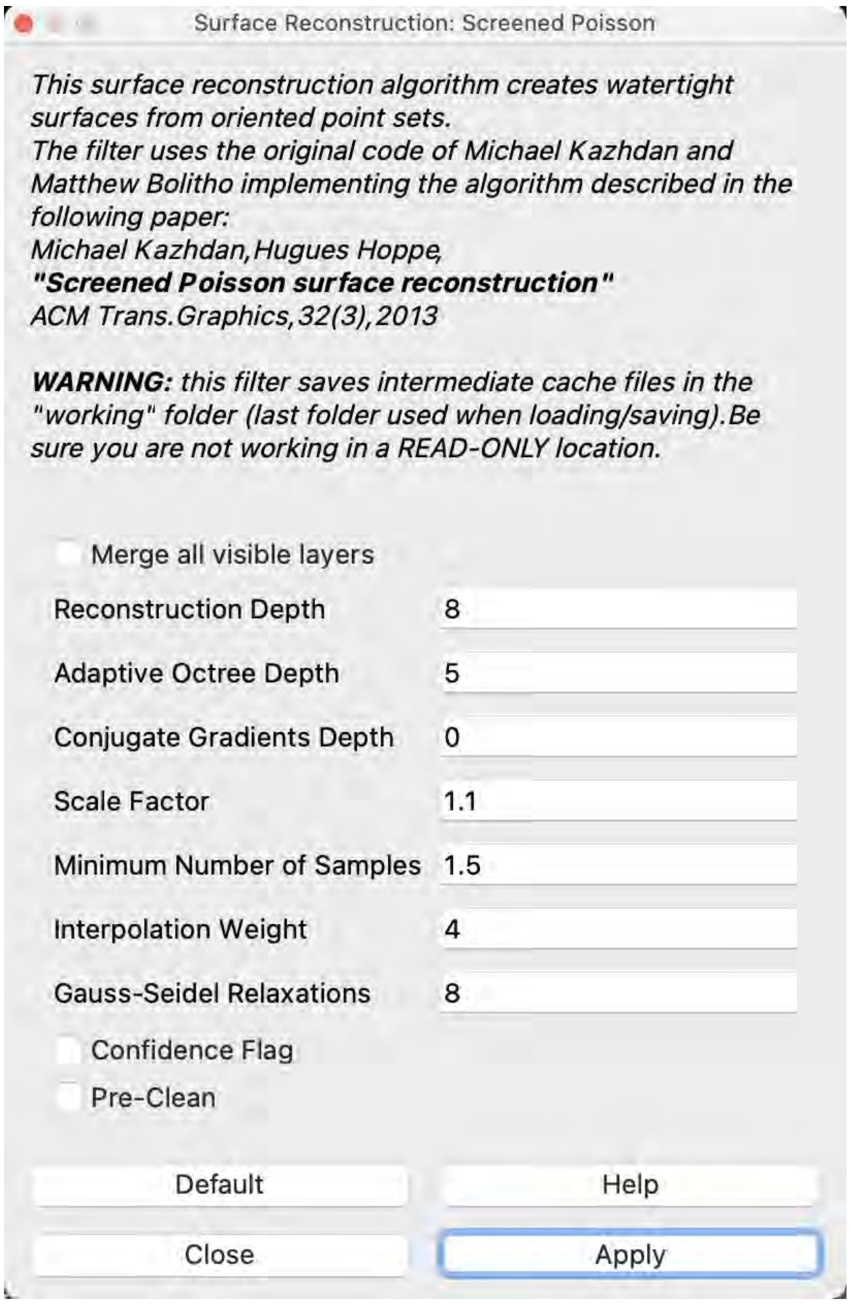

meshlab
File
Edit
Filters
Render
View
Windows
Help
Apply filter Compute normals for point sets
Show current filter script
Selection
Cleaning and Repairing
Create New Mesh Layer
Remeshing, Simplification and Reconstruction
Polygonal and Quad Mesh
Color Creation and Processing
Smoothing, Fairing and Deformation
Quality Measure and Computations
Normals, Curvatures and Orientation
Mesh Layer
Raster Layer
Range Map
Point Set
Sampling
Texture
Camera
Alpha Complex/Shape
Build a Polyline from Selected Edges
CSG Operation
Close Holes
Convex Hull
Create Solid Wireframe
Curvature flipping optimization
Cut mesh along crease edges
Delaunay Triangulation
Generate Scalar Harmonic Field
Iso Parametrization Build Atlased Mesh
Iso Parametrization Remeshing
Iso Parametrization transfer between meshes
Iso Parametrization: Main
Marching Cubes (APSS)
Marching Cubes (RIMLS)
Planar flipping optimization
Points Cloud Movement
Refine User-Defined
Remeshing: Isotropic Explicit Remeshing
Select Crease Edges
Simplification: Clustering Decimation
Simplification: Edge Collapse for Marching Cube meshes
Simplification: Quadric Edge Collapse Decimation
Simplification: Quadric Edge Collapse Decimation (with texture)
Subdivision Surfaces: Butterfly Subdivision
Subdivision Surfaces: Catmull-Clark
Subdivision Surfaces: LS3 Loop
Subdivision Surfaces: Loop
Subdivision Surfaces: Midpoint
Surface Reconstruction: Ball Pivoting
Surface Reconstruction: Screened Poisson
Surface Reconstruction: VCG
Tri to Quad by 4-8 Subdivision
Tri to Quad by smart triangle pairing
Turn into Quad-Dominant mesh
FOV: 60
FPS: 227.3
BO_RENDERING
Mesh: torus
Vertices: 10
Faces: 0
Selection: v

Figure D.6: *Step 6: The next step is to create the mesh, which is done as shown here*


Surface Reconstruction: Screened Poisson

This surface reconstruction algorithm creates watertight surfaces from oriented point sets.
The filter uses the original code of Michael Kazhdan and Matthew Bolitho implementing the algorithm described in the following paper:
Michael Kazhdan, Hugues Hoppe,
"Screened Poisson surface reconstruction"
ACM Trans.Graphics,32(3),2013

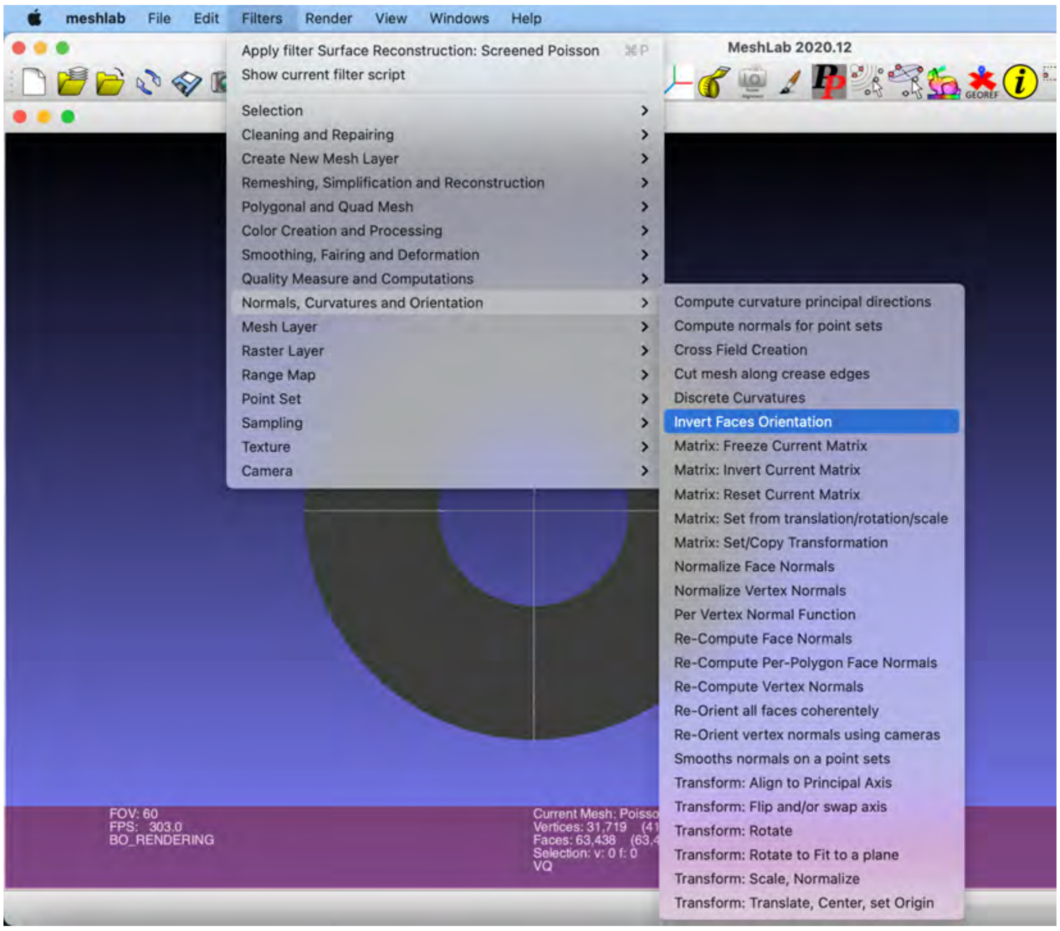
***WARNING:** this filter saves intermediate cache files in the "working" folder (last folder used when loading/saving). Be sure you are not working in a READ-ONLY location.*

☐ Merge all visible layers

Reconstruction Depth	8
Adaptive Octree Depth	5
Conjugate Gradients Depth	0
Scale Factor	1.1
Minimum Number of Samples	1.5
Interpolation Weight	4
Gauss-Seidel Relaxations	8

☐ Confidence Flag
☐ Pre-Clean

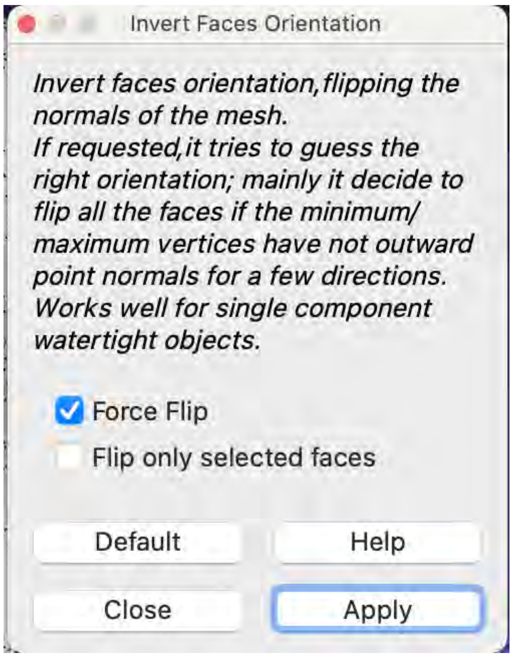
Figure D.7: *Step 7: Once again, use the default settings and hit "Apply"*


meshlab
File
Edit
Filters
Render
View
Windows
Help
MeshLab 2020.12
Apply filter Surface Reconstruction: Screened Poisson ⌘P
Show current filter script
Selection
Cleaning and Repairing
Create New Mesh Layer
Remeshing, Simplification and Reconstruction
Polygonal and Quad Mesh
Color Creation and Processing
Smoothing, Fairing and Deformation
Quality Measure and Computations
Normals, Curvatures and Orientation
Mesh Layer
Raster Layer
Range Map
Point Set
Sampling
Texture
Camera
Compute curvature principal directions
Compute normals for point sets
Cross Field Creation
Cut mesh along crease edges
Discrete Curvatures
Invert Faces Orientation
Matrix: Freeze Current Matrix
Matrix: Invert Current Matrix
Matrix: Reset Current Matrix
Matrix: Set from translation/rotation/scale
Matrix: Set/Copy Transformation
Normalize Face Normals
Normalize Vertex Normals
Per Vertex Normal Function
Re-Compute Face Normals
Re-Compute Per-Polygon Face Normals
Re-Compute Vertex Normals
Re-Orient all faces coherently
Re-Orient vertex normals using cameras
Smooths normals on a point sets
Transform: Align to Principal Axis
Transform: Flip and/or swap axis
Transform: Rotate
Transform: Rotate to Fit to a plane
Transform: Scale, Normalize
Transform: Translate, Center, set Origin
GEOREF

```
FOV: 60
FPS:  303.0
BO_RENDERING
```

```
Current Mesh: Poisso
Vertices: 31,719   (41
Faces: 63,438   (63,4
Selection: v: 0 f: 0
VQ
```

Figure D.8: *Step 8: If the surface of the mesh is inside out (it looks dark) simply invert faces orientation as shown here*


Invert Faces Orientation

Invert faces orientation, flipping the normals of the mesh.
If requested, it tries to guess the right orientation; mainly it decide to flip all the faces if the minimum/ maximum vertices have not outward point normals for a few directions.
Works well for single component watertight objects.

✓ Force Flip
Flip only selected faces

Figure D.9: *Step 9: Hit "Apply"*

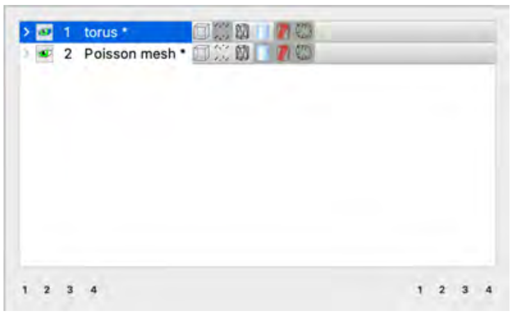


Figure D.10: *Step 10: On the right side of the screen, click the eye next to your file of points to hide the points*

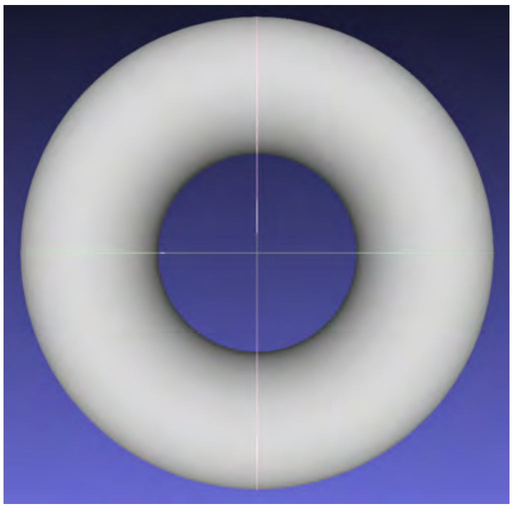


Figure D.11: *Step 11: Your mesh is now done and should look something like this*

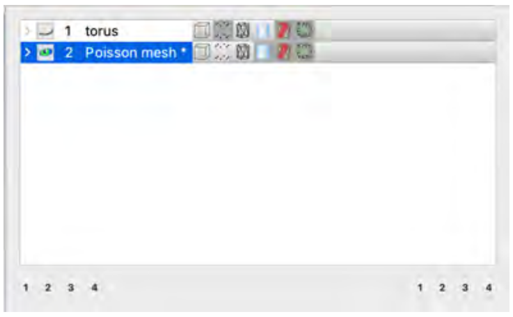


Figure D.12: *Step 12: To export the file, first select your Poisson mesh*

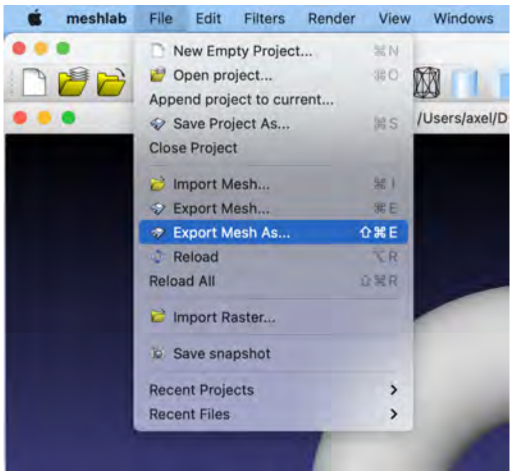
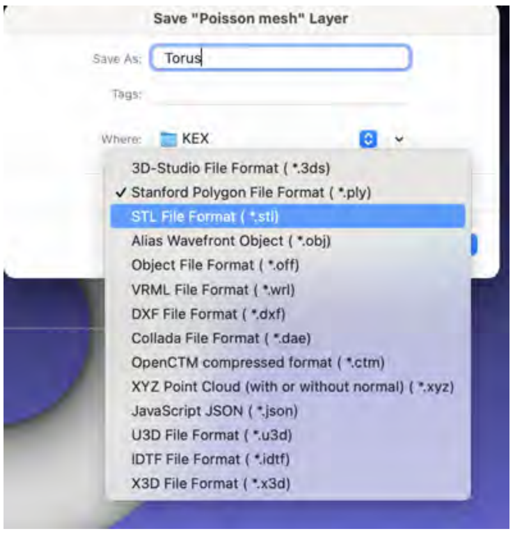


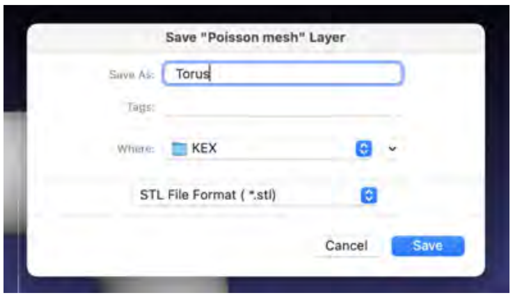
Figure D.13: *Step 13: Then choose "Export Mesh As..."*


Save "Poisson mesh" Layer
Tags:
3D-Studio File Format (*.3ds)
✓ Stanford Polygon File Format (*.ply)
STL File Format (*.stl)
Alias Wavefront Object (*.obj)
Object File Format (*.off)
VRML File Format (*.wrl)
DXF File Format (*.dxf)
Collada File Format (*.dae)
OpenCTM compressed format (*.ctm)
XYZ Point Cloud (with or without normal) (*.xyz)
JavaScript JSON (*.json)
U3D File Format (*.u3d)
IDTF File Format (*.idtf)
X3D File Format (*.x3d)

Save As: Torus

Where: KEX

Figure D.14: *Step 14: Choose a name for your file and select STL File Format*



Save As: Torus

Where: KEX

STL File Format (*.stl)

Figure D.15: *Step 15: Now the STL file can be saved and opened in other 3D environments or be 3D printed*

Appendix E

Stepper motor datasheet

2-Phase Hybrid Stepping Motor 1.8°

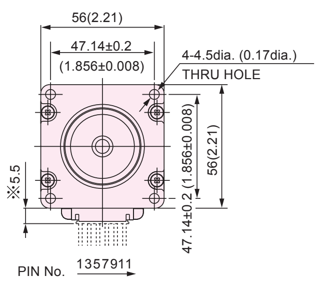
KH56 series 800 type

HIGH TORQUE, LOW VIBRATION AND LOW NOISE

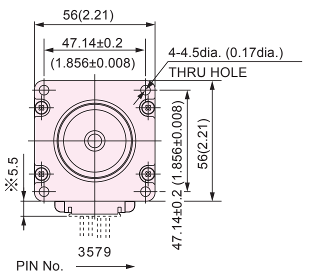
STANDARD SPECIFICATIONS

MODEL		KH56KM2			
		-801	-802	-803	-851
DRIVE METHOD	————	UNI-POLAR			BI-POLAR
NUMBER OF PHASES	————	2			2
STEP ANGLE	deg./step	1.8			1.8
VOLTAGE	V	2.4	3.7	6.8	4.35
CURRENT	A/PHASE	3.0	2.0	1.0	1.5
WINDING RESISTANCE	Ω/PHASE	0.8	1.85	6.8	2.9
INDUCTANCE	mH/PHASE	1.1	3.3	13.5	10.7
HOLDING TORQUE	mN • m	833	833	833	981
	oz • in	118	118	118	132
DETENT TORQUE	mN • m	37	37	37	37
	oz • in	5.6	5.6	5.6	5.6
ROTOR INERTIA	g • cm2	270	270	270	270
	oz • in2	1.48	1.48	1.48	1.48
WEIGHTS	g	650	650	650	650
	lb	1.4	1.4	1.4	1.4
INSULATION CLASS	————	JIS Class E (120°C 248°F) (UL VALUE : CLASS B 130°C 266°F)			
INSULATION RESISTANCE	————	500VDC 100MΩmin.			
DIELECTRIC STRENGTH	————	500VAC 50HZ 1min.			
OPERATING TEMP. RANGE	°C	0 to 50			
ALLOWABLE TEMP. RISE	deg.	70			

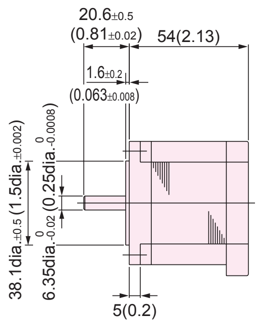
DIMENSIONS unit = mm (inch)



UNI-POLLAR



Bi-POLAR


20.6±0.5
(0.81±0.02)
54(2.13)
1.6±0.2
(0.063±0.008)
38.1dia.±0.5 (1.5dia.±0.002)
6.35dia.-0.02 0 (0.25dia.-0.0008 0)
5(0.2)
SINGLE SHAFT

Antriebstechnik GmbH
Starkenburgstr. 6 * 64546 Mörfelden
Tel.:06105 24044 * Fax:06105 25593
info@color-technik.net
www.color-technik.net

Features
- Stronger torque generated in higher speed zone
- Lowered Vibration by increased stiffness of body construction
- Improved Efficiency

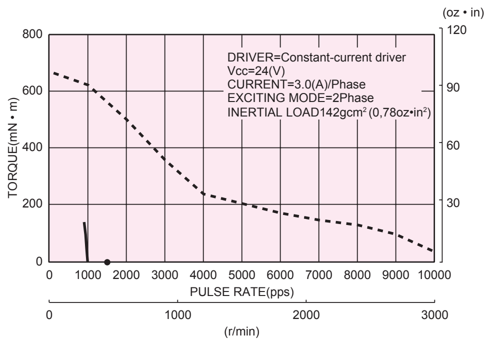
LOAD OF SHAFT:
30N THRUST LOAD
40N RADIAL LOAD 20mm FROM FRONTPLATE

——	PULL-OUT
—●	PULL-IN

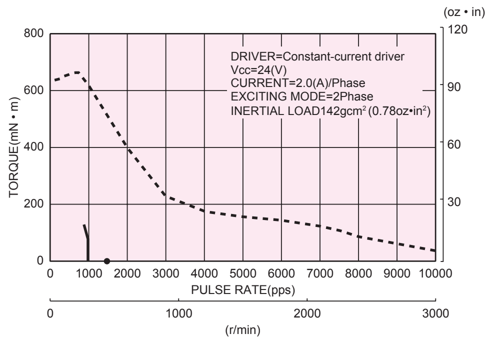
TORQUE CHARACTERISTICS vs. PULSE RATE

UNI-POLAR

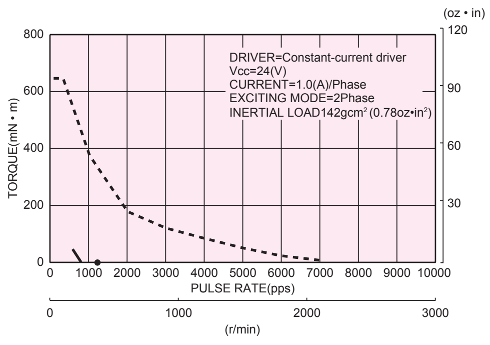
KH56KM2-801



KH56KM2-802,

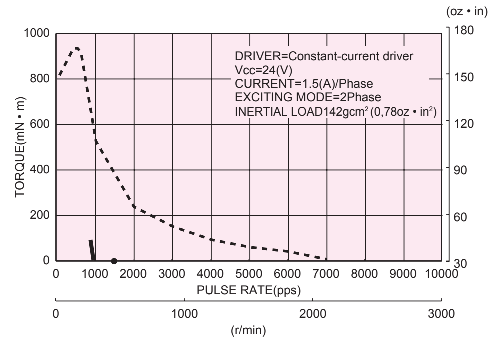


KH56KM2-803



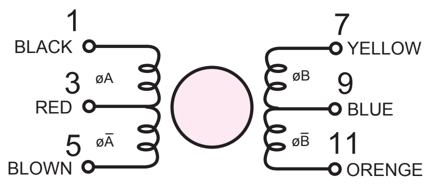
BI-POLAR

KH56KM2-851



CONNECTION DIAGRAMS

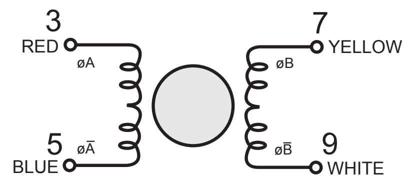
UNI-POLAR



EXCITATION SEQUENCE

STEP	1	2	3	4
BLACK	–			–
YELLOW	–	–		
BLOWN		–	–	
ORENGE			–	–
RED	+	+	+	+
BLUE	+	+	+	+

BI-POLAR

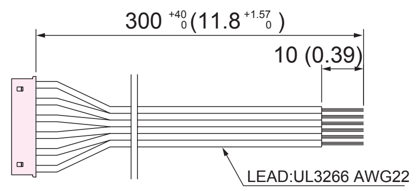


EXCITATION SEQUENCE

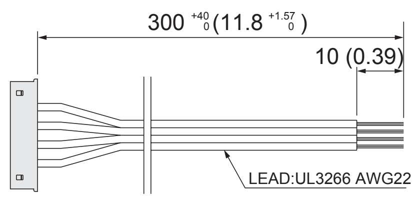
STEP	1	2	3	4
RED	+	+	–	–
YELLOW	–	+	+	–
BLUE	–	–	+	+
WHITE	+	–	–	+

CONNECTION CABLE TO MOTOR unit = mm (inch)

UNI-POLAR



BI-POLAR



JST XHP-11

TRITA ITM-EX 2021:39

www.kth.se