

Spring 2020

3Dscanner

Thomas Heckman

Robert Kalb

Shreyes Nallan

Follow this and additional works at: https://scholarcommons.scu.edu/elec_senior



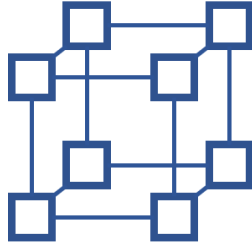
Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Heckman, Thomas; Kalb, Robert; and Nallan, Shreyes, "3Dscanner" (2020). *Electrical Engineering Senior Theses*. 50.

https://scholarcommons.scu.edu/elec_senior/50

This Thesis is brought to you for free and open access by the Engineering Senior Theses at Scholar Commons. It has been accepted for inclusion in Electrical Engineering Senior Theses by an authorized administrator of Scholar Commons. For more information, please contact rscroggin@scu.edu.



3Dscanner

By

Thomas Heckman, Robert Kalb, Shreyes Nallan

SENIOR DESIGN PROJECT REPORT

Submitted to

The Department of Electrical Engineering

of

SANTA CLARA UNIVERSITY

in Partial Fulfillment of the Requirements
for the degree of
Bachelor of Science in Electrical Engineering

Santa Clara, California

Spring 2020

SANTA CLARA UNIVERSITY
Department of Electrical Engineering

I HEREBY RECOMMEND THAT THE THESIS PREPARED
UNDER MY SUPERVISION BY

Thomas Heckman, Robert Kalb, Shreyes Nallan

3Dscanner

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF


BACHELOR OF SCIENCE
IN
ELECTRICAL ENGINEERING



Jun 11, 2020

Thesis Advisor: Dr. Andrew Wolfe

date


Shoba Krishnan (Jun 11, 2020 16:54 PDT)

Jun 11, 2020

Department Chair: Dr. Shoba Krishnan

date

Abstract

This project entails designing, prototyping, and testing a 3D scanner. The device that we are building uses LIDAR to take position data of a 3D object, then analyze and encode the sensor data as an STL file that can later be 3D printed out at the same resolution. We aim to build an affordable, high-performance 3D scanner that takes advantage of the falling cost of LIDAR in order to bring 3D scanning capabilities to individuals, the maker community, and even small businesses.

We begin this process by choosing a sensor, the YDLIDAR X4, as our primary method of taking position data of the object. We take calibration data with this sensor to ensure that it is suitable to our needs. In doing so, we find that we may need to incorporate certain statistical methods, like dithering, in order to increase the accuracy of the system. We determine an effective layout for scanning all sides of an object, overcoming obstacles like scanning objects with concave surfaces. The physical system is mocked up in Solidworks, enabling us to 3D print, laser cut, and buy all the necessary components of the system. The system is constructed while an interactive user interface is created. We develop an algorithm for turning individual data points from the raw data of the X4 sensor into 3D printable STL files, and interface it with a program that controls the motors to take consistent, comprehensive scans of any object on the platform.

In the end, we find two limitations of LIDAR in 3D scanning systems - high-gloss black surfaces and certain steep angles cannot be scanned adequately by LIDAR. However, once our system is constructed, we are able to take 3D scans of common objects, and even 3D print one of our scanned objects. The scan is compared to the original object, and the dimensional accuracy of our scanner is verified.

Acknowledgements

Our team has received a \$1500 grant from the Santa Clara University School of Engineering, which we are very grateful for. We have also received funding and a significant amount of mentorship and guidance from our advisor, Dr. Andrew Wolfe, which has been invaluable to our team's success this year. We greatly appreciate all contributions to our project and team!

Table of Contents

Abstract	3
Acknowledgements	4
Table of Contents	5
List of Figures	7
Chapter 1: Introduction	8
1.1 3D Scanner Definition	8
1.2 Background Information	8
Chapter 2: Objectives	9
2.1 Project Goals	9
2.2 Project Requirements	9
Chapter 3: Project Plan	10
3.1 Subsystems Overview	10
3.2 Functional Block Diagram	10
Chapter 4: Sensor and Calibration	11
4.1 YDLIDAR X4 Sensor	11
4.2 Calibration	12
4.3 Dithering	13
Chapter 5: Subsystems	14
5.1 Physical Build	14
5.2 Software: General Overview	17
5.3 Motor and Sensor Software	17
5.3.1 State Encoder	18
5.3.2 Motor Homing	19
5.3.3 Calibration Scan	19
5.3.4 Pre-Scan	19
5.3.5 Full Scan	20
5.4 Communication Between Systems	20
5.5 Data Processing and STL Generation	20
5.5.1 Preprocessing	20
5.5.2 Averaging	21
5.5.3 Contour Generation	22

5.5.4 Connecting Layers	23
5.5.5 Verification	24
5.6 User Interface	24
5.6.1 Menu Window UI	24
5.6.2 Progress Bar UI	25
Chapter 6: Schedule and Division of Labor	26
6.1 Project Schedule	26
6.2 Division of Labor	26
Chapter 7: Findings and Results	27
7.1 Findings	27
7.2 Results	30
7.3 Conclusion	30
References	32
Appendix A: Senior Design Conference Slides	33

List of Figures

Figure 1. Process of 3D Scanning a Cylinder

Figure 2. From Left to Right: Ciclop 3D Scanner, Matter and Form 3D Scanner, LIDAR Visualization

Figure 3. System Workflow and Incorporation of Subsystems

Figure 4. Functional Block Diagram

Figure 5. X4 LIDAR Sensor from YDLIDAR

Figure 6. A Solidworks model of our test bench is pictured on the left, with the actual implementation pictured on the right.

Figure 7. Recreation of a flat surface using the X4's normalized raw data

Figure 8. Images reprinted from "An Optimum Condition of the Dithering Signal against the Quantization Noise in ADC" by Liu et al. showing the effects of dithering in signal recreation

Figure 9. Original designs for the scanning tower (left) and rotating platform (right)

Figure 10. The C-clamp Problem: as the C-clamp rotates, a single data point would not capture the entire surface of the object

Figure 11. Final Solidworks render of the full system (left) and physical system (right)

Figure 12. Solidworks render of the tower base (left) and moving platform (right)

Figure 13. Solidworks render of the rotating platform

Figure 14. Diagram of software flow

Figure 15. State Machine for Full Scan Routine

Figure 16. Conceptual diagram of the raw data used to generate an STL.

Figure 17. Evolution of a point cloud through three algorithm stages. Left: points after the preprocessing step; middle: points after the averaging step; right: final contour generated.

Figure 18. Triangles generated when connecting two layers. Only the triangles originating from the bottom layer are shown.

Figure 19. Examples of 3D models created when noisy simulated data is passed through all stages of the STL generation algorithm.

Figure 20. Sample screens from the menu UI. Top left: the main page; top right: the scan settings page; bottom: setting an individual scan parameter.

Figure 21. Example screen for the progress bar UI.

Figure 22. Full project Gantt chart

Figure 23: Scanning a computer mouse (top) and the resulting STL (bottom)

Figure 24. Scanning the blocks at originally (left) yielded no data because of the angle of incidence, but rotating them (right) gave the data we expected

Figure 25. Slide from a presentation by the Ford Motor Company confirming the poor results yielded when scanning glossy black surfaces with LIDAR

Figure 26. Slide from a presentation by the Ford Motor Company confirming the poor results yielded when scanning a surface with high angle of incidence using LIDAR

Figure 27. A can of Ice Cubes gum that was scanned (left), the STL output from our 3D scanner (middle), and the 3D print of the STL (right)

Chapter 1: Introduction

1.1 3D Scanner Definition

A 3D scanner is a device that creates digital, three-dimensional representations of real-world objects. These digital forms can then be edited, manipulated, and/or physically recreated by a 3D printer or CNC machine. Figure 1 below shows a representation of what a 3D scanner does: it starts with a real-world object like the cylinder on the left, it takes data that represents the object's size and shape, and the data is used to create a digital representation of the object like the one on the right. The cylinder pictured here is in an STL file format - one of the most commonly used formats on 3D printers today.

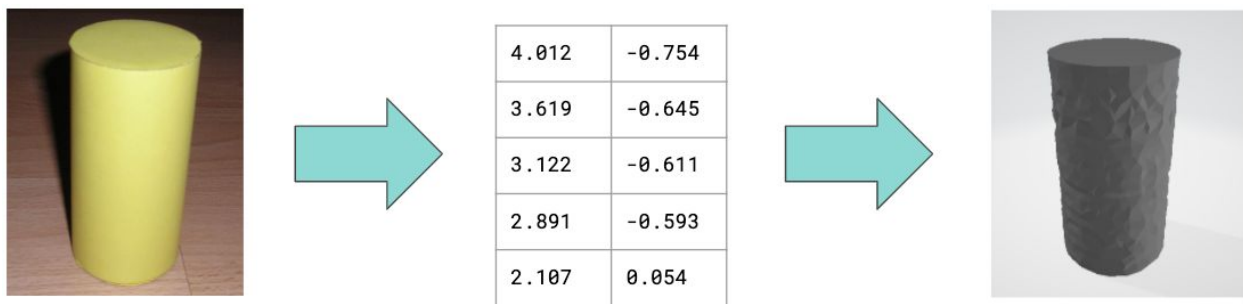


Figure 1: Process of 3D Scanning a Cylinder

STL files can be viewed and edited in programs like Blender, Meshlab, or Solidworks to make any changes to the object or otherwise improve upon it digitally. 3D scanners enable new, faster methods of rapid-iteration prototyping, since objects can be produced, tested, and re-scanned so that the model can be digitally altered and prepared for its next iteration.

1.2 Background Information

When we tell people that we're building a 3D scanner, the first question we always hear is: "Don't 3D scanners already exist?" And the answer is "yes," but the 3D scanners on the market all have disadvantages, and we're hoping to improve upon what's already available. For example, the scanner you see on the left in Figure 2 is a reasonably cheap one – you can buy it for about \$150 to \$200, but its scans are not particularly accurate. The scanner pictured in the middle of Figure 2 will give much higher resolution scans, but it costs closer to \$1,000. In addition, most 3D scanners on the market today use cameras to get position data. Nowadays, there's new technology available. With the advent of self-driving vehicles and their use of LIDAR, the cost of LIDAR sensors is starting to drop, and we wanted to see if we could take

advantage of this new technology in our project. We sought to build a 3D scanner whose cost would be comparable to the current low-end desktop scanners, but give performance similar to a high-end desktop scanner.



Figure 2: From Left to Right: Ciclop 3D Scanner, Matter and Form 3D Scanner, LIDAR Visualization

Chapter 2: Objectives

2.1 Project Goals

When we started this project in July 2019, we came up with the idea to design, prototype, and troubleshoot a low-cost, high performance 3D scanner by April 2020 for less than \$2000. By the end of the project, our goal was to have a fully functioning prototype of our 3D scanner, a developed scanning algorithm, a complete user-interface, and at least one object scanned and 3D printed.

2.2 Project Requirements

We wanted this scanner to have a price and performance comparable to that of hobby-level 3D printers. The Prusa i3-MK3 is priced around \$750, and has a nominal accuracy of 0.2 mm. Ideally, this 3D scanner could even be sold as a kit, similar to the Prusa printers. In addition, the user interface would have to be constructed in a way that is user-friendly, so that anybody would have the ability to take scans with our scanner.

Chapter 3: Project Plan

3.1 Subsystems Overview

The work we did in this project falls into three main categories. First is the physical build. This includes the scanning tower, rotating platform, and LIDAR sensor that come together to get scans of an entire object. The second is the software, which controls the motors and sensors using low-level C++ code on a Raspberry Pi. This starts the scans and reads out the LIDAR data. Third, the data is transferred to our data processing algorithms, which function as high-level Python code running on a PC. These algorithms analyze the data and turn it into an STL file. The scanner's user interface is incorporated into the algorithm subsystem and is also run on the PC.

Figure 3 below shows how these subsystems are incorporated and work together to create 3D scans.

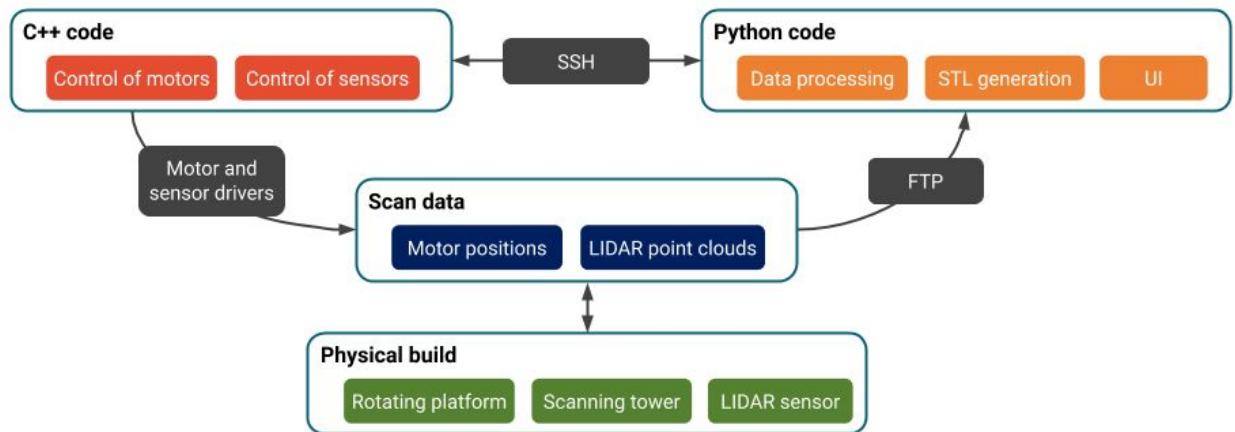


Figure 3: System Workflow and Incorporation of Subsystems

3.2 Functional Block Diagram

Figure 4 below shows a view of the physical system. The object being scanned rests on the rotating platform. For one 360 degree layer of the object, the LIDAR spins and takes a certain number of data points as a stepper motor, inside the highlighted yellow piece, rotates the platform with micrometer accuracy. During this process the dithering stepper, seen in green, moves the LIDAR sensor back and forth to create a distribution of points to be averaged. The purpose of this movement will be explained later. Then, the next layer is scanned after the tower stepper, in the highlighted blue piece, moves the LIDAR vertically upwards. This process is continued for the entire height of the object. The motors are controlled by a microcontroller

which communicates with a Raspberry Pi controlling the LIDAR. All scanning data is passed back to the host computer wirelessly through an SSH connection to be processed by the scanning program.

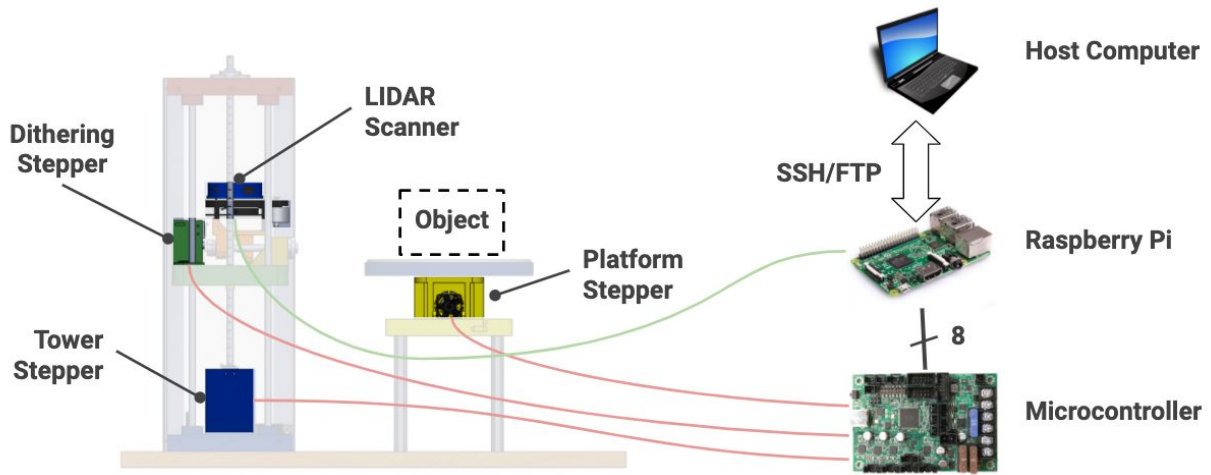


Figure 4: Functional Block Diagram

Chapter 4: Sensor and Calibration

4.1 YDLIDAR X4 Sensor

In the early stages of the project, we researched multiple LIDAR sensors to see if we could find one that was suitable for our needs. Ultimately, we chose a sensor from the company YDLIDAR, called the “X4”. The X4, pictured in Figure 5, is a 360 degree LIDAR scanner used primarily for adding computer vision to autonomous robots. It has a scanning rate of 5 kHz, operates at a wavelength of 785 nm (in the near-infrared spectrum) and a low price point at under \$80. In addition, its datasheet (from ydlidar.com/download) claims that its laser is “Class 1”, which, according to lasersafetyfacts.org, is “safe, even for long-term intentional viewing.”



Figure 5: X4 LIDAR Sensor from YDLIDAR

4.2 Calibration

YDLIDAR claimed that the X4 had the accuracy we needed, but to be sure, we created a testbench for calibrating the sensor and understanding its impressive high speed data capture. The testbench, shown in Figure 6, is simply the X4 sensor, a stepper motor, and a wall attached to a screw drive powered by the motor. The testbench was designed in Solidworks, and implemented primarily using 3D printed components. This testbench allowed us to take a variety of scans to understand the sensor's performance, including testing for the scanner's linearity, monotonicity, absolute accuracy, differential accuracy, angular accuracy, and more.

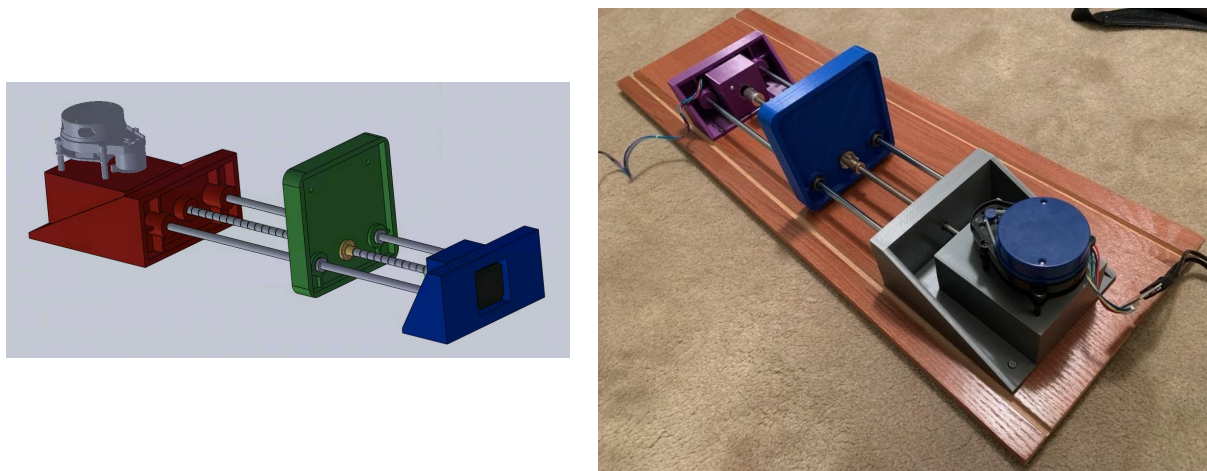


Figure 6: A Solidworks model of our test bench is pictured on the left, with the actual implementation pictured on the right.

We took a scan across the flat surface of the test bench, which, when we normalize the data to the angle associated with each distance value, yielded the graph shown in Figure 7. This graph is meant to be a reconstruction of a flat surface.

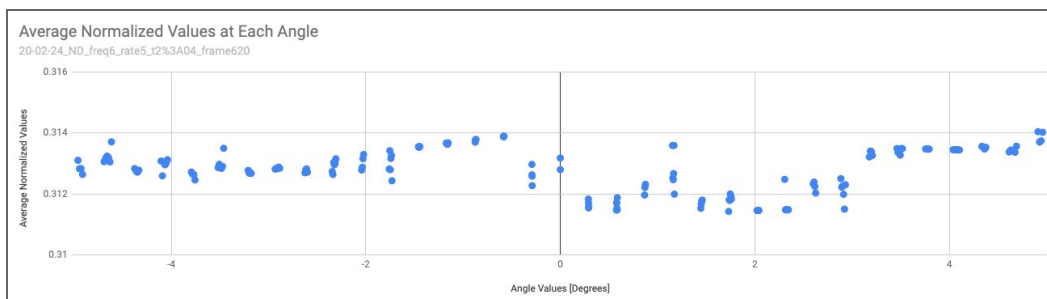


Figure 7: Recreation of a flat surface using the X4's normalized raw data

The graph's x-axis is the range of data we chose to use from the sensor - spanning from -5 degrees to 5 degrees, where each data point represents a spot on the flat surface. The y-axis is the distance measurement from the scanner to the flat surface, normalized to the angle of each point. The resulting graph should have been a flat line, but instead, the data has an undesirable distribution of values varying by as much as $\pm 1.5\text{mm}$ (which is in stark contrast to the claimed accuracy of the X4 sensor). For this reason, we chose to implement dithering in our routine.

4.3 Dithering

Dithering is a data sampling technique that aids in a more accurate recreation of captured data, especially in an application where quantization noise exists. To implement dithering, a controlled amount of noise must be introduced into the system such that the level of noise is equal to the range of results for a given known value. When the spread of data for a single point is averaged, the mean will more accurately represent the actual value. Figure 8 shows three graphs from a study titled "An Optimum Condition of the Dithering Signal against the Quantization Noise in ADC", by Liu et al, showing how they were able to more accurately recreate signals using dithering to eliminate quantization noise.

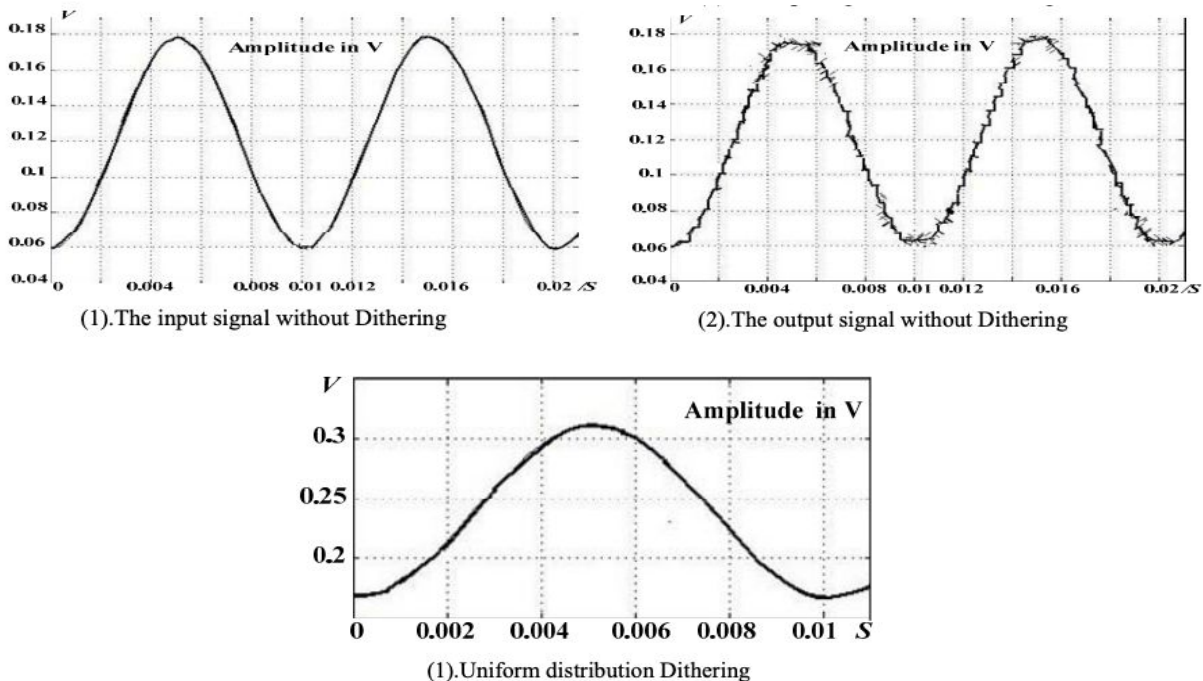


Figure 8: Images reprinted from "An Optimum Condition of the Dithering Signal against the Quantization Noise in ADC" by Liu et al. showing the effects of dithering in signal recreation

In our system, we chose to implement dithering by moving the X4 sensor closer to and further away from the object that we were scanning. Theoretically, this would allow us to obtain more accurate measurements for each data point.

Chapter 5: Subsystems

5.1 Physical Build

At the beginning of the project, we decided that we wanted to have a scanning tower that housed the LIDAR sensor and enabled it to move vertically, and a separate rotating platform where we could place the objects to be scanned. By rotating the objects and moving the LIDAR vertically, we could capture data of the full exterior of any object. We created some initial designs for the scanning tower and rotating platform in Solidworks, which are shown in Figure 9.

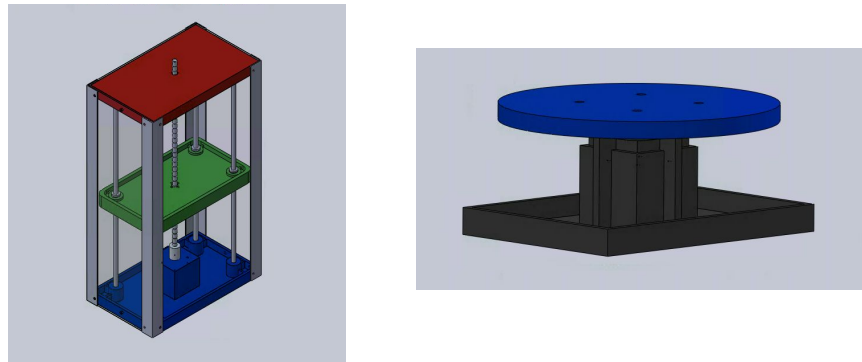


Figure 9: Original designs for the scanning tower (left) and rotating platform (right)

This layout left us with one problem that we referred to as the “C-clamp problem”. The issue is that if we tried to scan a C-clamp (or any object with multiple contours on a single layer) using a single datapoint from the LIDAR, we wouldn’t be able to capture data on the “inside” surfaces of it. Figure 10 shows this problem - the three pictures represent the C-clamp being rotated, and the red circle represents a single point where a LIDAR sensor would take data.

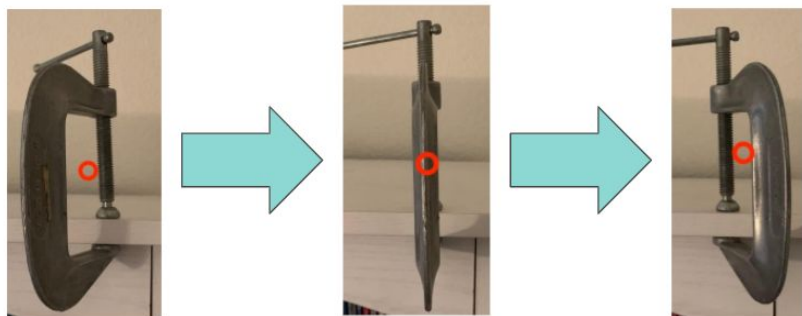


Figure 10: The C-clamp Problem: as the C-clamp rotates, a single data point would not capture the entire surface of the object

Fortunately, our chosen sensor (YDLIDAR's X4) takes full rotational scans with data points at 0.5 degree increments. By using multiple data points on every scan, we were able to stitch together a set of points from each viewing angle on the object, and capture the inside surface of the C-clamp and other object concavities. This feature enabled us to keep a similar design for our final version of the scanning tower.

After going through five iterations of both the scanning tower and rotating platform, we arrived at our final design. Figure 11 shows the final Solidworks render on the left, and the actual physical system on the right.



Figure 11: Final Solidworks render of the full system (left) and physical system (right)

One feature that we consider to be particularly important is that all of the colored parts can be 3D printed on hobby-level 3D printers, and in fact, were printed on a Prusa i3-MK3 in our final build. Most of the other pieces in the system – the motors, fans, lead screws, shaft collars, etc – are commercially available parts. Almost none of it had to be customized. That means that if we were to turn this project into a commercially available system for customers, it could be sold as a kit, with a similar cost and difficulty of assembly as a hobby-level 3D printer (like a Prusa). Ultimately, it could be marketed towards the DIY and maker communities for those reasons, enabling small businesses and even individuals to have 3D scanning capabilities aiding them in rapid-iteration design processes.

In Figure 12, the render on the left shows the base of the scanning tower, which houses two NEMA-17 stepper motors. These motors drive lead screws that enable the raising and lowering of the middle platform. Two limit switches above the motors provide a homing position for the platform, and two fans provide active cooling for the motors. In the center of the base, we have a mount for our motor controller board. The render on the right of Figure 12 shows the moving platform and the dithering mount for the X4 sensor. The green platform has

four linear bearings mounted in it so that it can slide up and down along linear guide rods, which give smooth, repeatable motion and prevent racking. The dithering platform itself, the orange piece, is driven by another NEMA-17 stepper motor, with two more linear bearings that travel horizontally on shorter guide rods. The dithering platform also has two limit switches to give the sensor a “home” position.

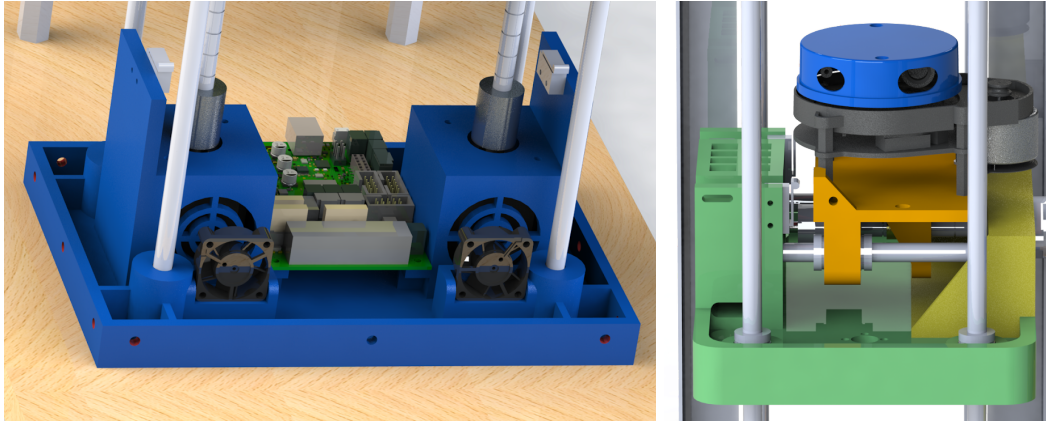


Figure 12: Solidworks render of the tower base (left) and moving platform (right)

Finally from the hardware side is the rotating platform, shown in Figure 13. The yellow piece of this platform is fixed at a set distance away from the scanning tower. The object to be scanned will be placed on top of this platform. The white piece on the very top is an acrylic disk that is press-fitted into the 3D printed blue mount. We wanted the top to be as flat as possible, which meant having no screw heads sticking up from the surface that could block parts of the object from being scanned, and no counterbores to sink screw heads into the surface, because that the object to be scanned could tip into those holes. That’s why we chose to have a press-fitted acrylic disk. The surface is rotated by another actively cooled NEMA-17 stepper motor, which is connected to the blue rotating platform through a motor hub. The blue piece is also supported from below by a turntable bearing concentric to the motor shaft to provide extra stability.

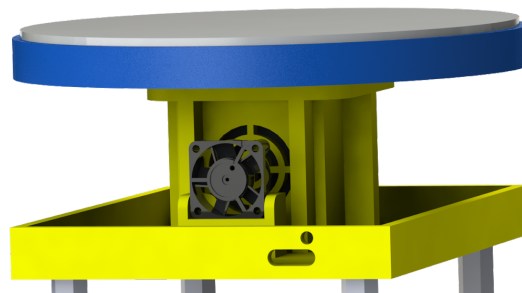


Figure 13: Solidworks render of the rotating platform

5.2 Software: General Overview

The scanning process involves many interconnected algorithms run on three different platforms: a host PC running Windows, a Raspberry Pi, and an Arduino-style microcontroller (Figure 14). The main program begins with a menu UI window on the host PC, where the user picks scan parameters and starts the scan. The scan parameters are sent through an SSH connection to the Raspberry Pi. In order to begin the scan, the main Python algorithm creates an STL file where the digital model will be built, while a C++ algorithm on the Raspberry Pi starts a calibration and prescan routine. Once this is finished, the system enters the main scan routine. In this process, the LIDAR is swept, taking a certain number of datapoints; then the platform is rotated, and the LIDAR is swept again. This continues until the platform finishes a full rotation. At this point, the scanning tower moves up and takes the next layer of data. At the same time, the current layer's scan data is file-transferred to the Windows machine. There, a set of Python algorithms process the raw data, generate a contour for the layer, connect that contour to the contours above and below, and update the STL file. The scan process ends when the top of the object is reached; at that point, the top surface is filled in and the STL file is closed.

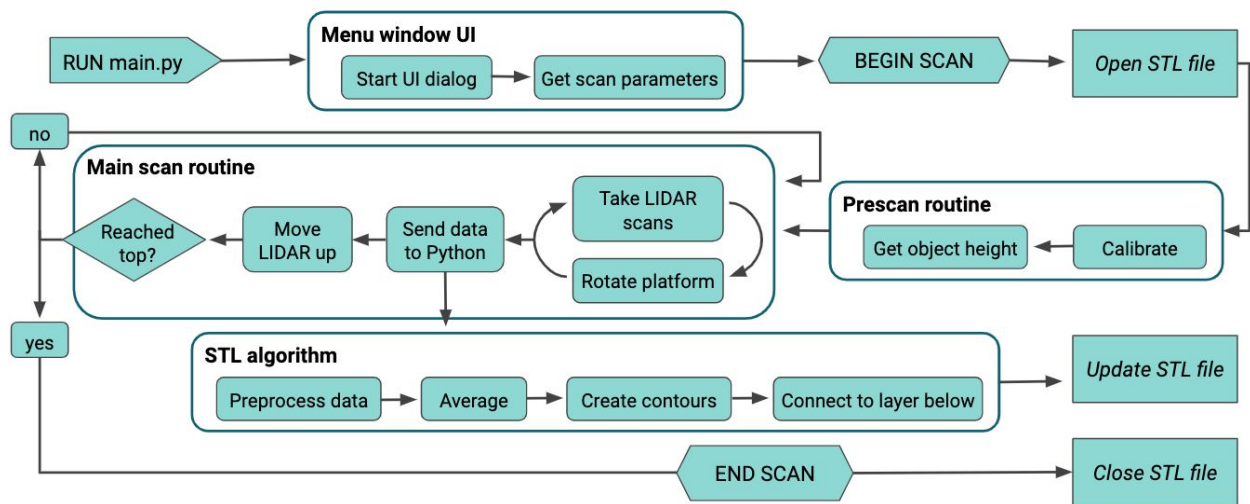


Figure 14: Diagram of software flow

5.3 Motor and Sensor Software

In order to achieve an accurate and efficient scan, it is necessary to conduct two preliminary tests before taking a full scan of the entire object. For simplicity, the state machine for a full scan routine (detailed in Figure 15) is modified to accommodate three different types of scans as described in the following sections. Before explaining the nuances of each scan type, it is necessary to understand how communication is established between the Raspberry Pi and the Arduino, and how the motors undergo homing calibration.

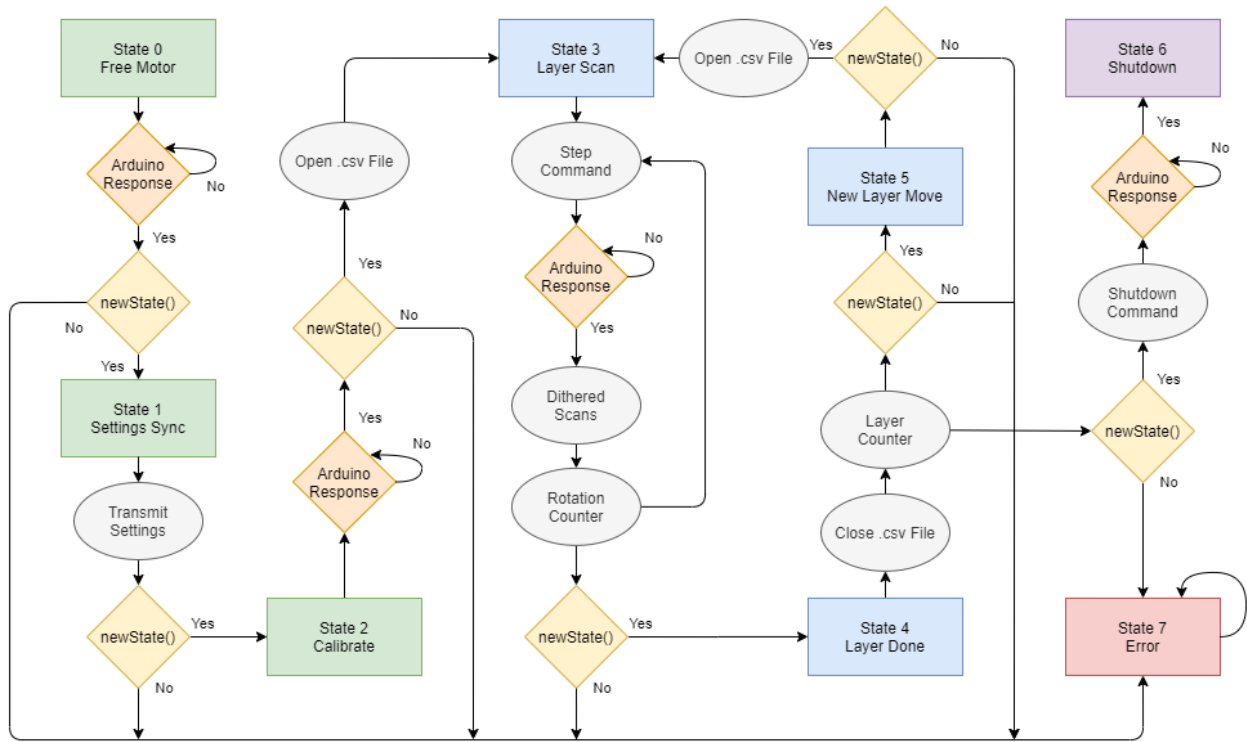


Figure 15: State Machine for Full Scan Routine

5.3.1 State Encoder

Three digital pins are set aside from both intelligent systems to write or read state changes. In this instance the Raspberry Pi assumes the role of master and is therefore in charge of writing to these fully encoded digital pins. The Arduino, as the slave, will react to the state published by the Raspberry Pi. The following actions in bullet form are what is described by the yellow `newState` functions in Figure 15.

- Changes in state begin when the Raspberry Pi raises a separate new state pin and waits for the Arduino to raise a response pin.
- The Arduino raises the response pin once it finishes its procedure for the previous state and detects a new state request from the Raspberry Pi
- When the Raspberry Pi sees a raised response pin, the new state will be published, and the Raspberry Pi will lower the new state pin. A time delay is additionally added to allow the Arduino to enter the next state and receive additional commands.
- A lowering of the new state pin will prompt the Arduino to read the encoded state and lower its response pin.

Overall this simple method has worked quickly and reliably and is well suited for the system given the abundance of digital pins available.

5.3.2 Motor Homing

During State 2, the vertical motors and dithering motors are driven to find their home using endstops as precise position indicators. When an endstop is found, the motors are driven very slowly in the opposite direction until the endstop is released. A preset distance from the negative edge of the endstop is used as a precise home for the motors. This calibration is invaluable for ensuring reliable motor positions across all three scan types that will be described in the following sections.

5.3.3 Calibration Scan

The STL generation process requires a reference for how far the scanner sits from the object platform. By scanning the outside edge of the rotating platform, an accurate reference point can be achieved by selecting the shortest distance value between the X4 and the rotating platform. This datapoint also contains an angle to normalize all of our data to, in case the X4's intrinsic 0° angle is off-axis. To negate the impact of any small physical changes to the system, a calibration scan is run at the start of each scan routine. In terms of the procedure, the system will follow the state diagram in Figure 15 but only conduct one loop through the blue highlighted states for a single layer of data taken at the platform edge. The resulting data is fed back to the python program in a CSV file. The python will process the calibration data so it is ready to be fed to the full scan.

5.3.4 Pre-Scan

To reduce the number of empty layers being sent to the STL generator, a pre-scan is used to find the maximum height of the object. This will set an upper bound for the number of layers in the full scan to reduce the total scan time and prevent any phantom data that may occur in a non-existent layer. This process also gives us the ability to display a progress bar to the user (as detailed in section 5.5.2).

The procedure for finding height is much more complex than either the calibration scan or full scan. The most efficient solution, given the nature of the system, involves repurposing the state encoder (section 5.3.1) as a means to inform the Arduino of where the highest point of the object lies. The method starts by dividing up the entire vertical range of the scanner into eight even sections. A single scan at each increment will allow the Raspberry Pi to determine which section contains the object's peak by finding the highest layer with valid data and choosing the section above. The chosen section is relayed to the Arduino via the three state bits and promptly moves the motors to the bottom of this section. The process is continued with the new section divided into eight smaller sub-sections until an accurate height is determined with sub-millimeter accuracy.

As in the case of the calibration scan, the final height (calculated in millimeters) is fed to the python program in an CSV file.

5.3.5 Full Scan

With both calibration data and object height information from the preliminary tests, the python program is able to pass this data to the Raspberry Pi in conjunction with desired vertical and angular resolutions from the user interface (as described in section 5.5.1). The most difficult design aspect of the full scan was relaying resolution information from the Raspberry Pi to the Arduino. The key to this communication was reusing the state encoder (section 5.3.1), similar to the pre-scan routine, to select preset resolutions available to the user; essentially leaving a maximum of 8 possible settings for both vertical and angular resolution. Relaying this information occurs in State 1 of Figure 15.

The remainder of the scan follows the flow of the state diagram with an appropriate amount of layers determined by the height multiplied by the vertical resolution; both in millimeters.

5.4 Communication Between Systems

Over the course of the scan, we have to transfer data between the Windows PC, where scan parameters are set and scan data is analyzed, and the Raspberry Pi, where the scan is run. To accomplish this task, we used an SSH connection between the main Python program and the command line of the Raspberry Pi. This connection was implemented programmatically through the paramiko Python module. To start the scan, the following command was executed remotely on the Pi command line:

```
roslaunch ydlidar lidar.launch scanArgs:="-flag -levels -vertRes -angRes" &
```

To transfer data, the program used the `ls` command to search for a created CSV file in a certain directory. Once each layer's CSV file was found, it was transferred through SFTP (the SSH File Transfer Protocol) to a local directory on the Windows computer. The algorithms in section 5.5 accessed the data for each layer through this local directory.

5.5 Data Processing and STL Generation

The following algorithms are run on the Windows PC. They are all coded in Python, and dependent on the numpy library.

5.5.1 Preprocessing

When raw data from the LIDAR sensor comes in, the first step in incorporating that data into the STL file is to transform it into a consistent set of coordinates. Each raw datapoint

contains three values: the radial distance to the point r , the angle to the point θ , and the rotation angle of the platform α . We also know, from measurements of the full system and the calibration scan (detailed in section 5.3.3), the distance from the LIDAR sensor to the center of the rotating platform, denoted by A . Our goal is a set of coordinates x and y in the object's proper frame. For a full diagram, see Figure 16.

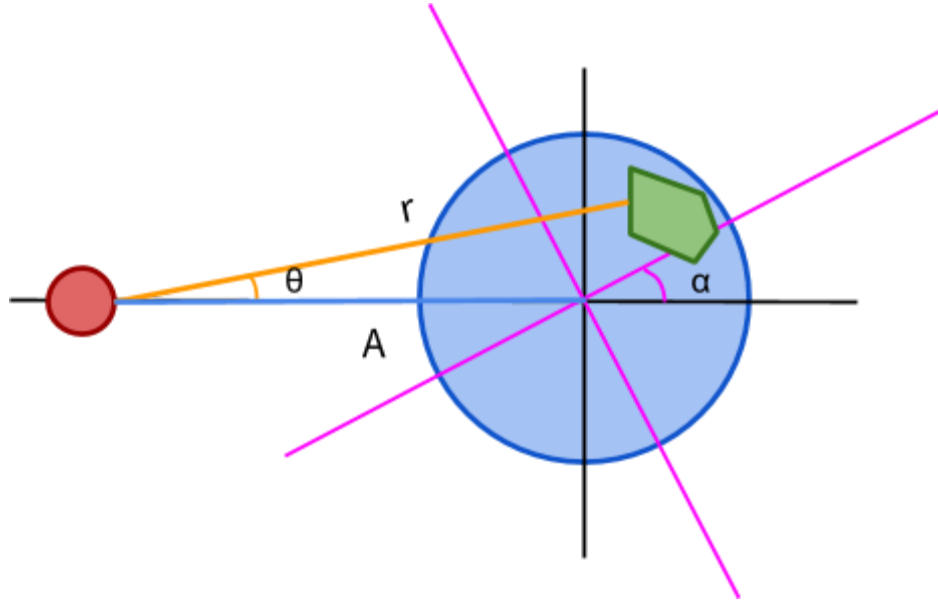


Figure 16: Conceptual diagram of the raw data used to generate an STL.

In order to get x and y , we start by getting x_0 and y_0 , which are the coordinates uncorrected for rotational angle:

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} r \cos(\theta) - A \\ r \sin(\theta) \end{bmatrix}$$

Then we use a rotation matrix to compensate for the rotation angle:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

These are our final coordinates. They will be used for the rest of the algorithm.

5.5.2 Averaging

We then slice the entire grid of points into uniform squares. The size of the squares is determined by the “Planar resolution” parameter, which is set by the user in the menu UI (see subsection 5.5.1 for details). All points within a certain square are averaged, and the squares that have only one point are ignored. This process helps filter out outliers, average out random noise, and reduce the number of points inputted to the contour generation algorithm.

5.5.3 Contour Generation

The purpose of contour generation is to take a set of points in two-dimensional space and create one or more closed shapes. The end goal is a list of connections -- the sequence of points that make up the outer ring of those closed shapes. Generating contours is one of the central problems of the field of computational geometry, and we were able to use a computational geometry Python package called `alphashape` to do most of the work for us.

The `alphashape` module takes a set of points and generates a shape object, which can be further manipulated, using the `shapely` module, to find a list of connections. Unfortunately, the output shape depends heavily on a parameter called `alpha` -- in fact, if we sweep `alpha`, we get a diverse range of possible shapes. These range from dozens of fractured triangles (`alpha = 1`) to a big blob covering all points (`alpha = 0`). There is no way to know, *a priori*, which `alpha` results in the optimal shape; therefore, we had to build another algorithm to do this.

Our algorithm was based on two central principles: (1) the final shape should be as *convex* as possible; and (2) the final shape should include as many *points* as possible. For a consistent measure of property (1), we use the *isoperimetric ratio*, defined as the area of the shape divided by the perimeter squared. If we have N separate shapes, the area increases N times, while the perimeter squared increases by N^2 , so the isoperimetric ratio is multiplied by N to compensate.

$$\xi \equiv 4\pi N \frac{A}{p^2}$$

According to the *isoperimetric inequality*, ξ reaches its maximum value of 1 when the shape is a perfect circle, or a set of perfect circles.

The algorithm sweeps through `alpha` values from 1 to 0 in increments of 0.01. For each value, it finds the shape and calculates the number of points included and the value of ξ . All shapes with too few points or too many regions are ignored. When a big jump in ξ is detected -- pushing it above a certain threshold, defined as 0.8 -- the algorithm ends and returns the associated value of `alpha`. If no clear transitions are detected, the algorithm takes all shapes with $\xi > 0.8$, and picks the shape that includes the most points.

After these three steps, noisy raw data from a single layer has been averaged and turned into a set of contours (Figure 17).

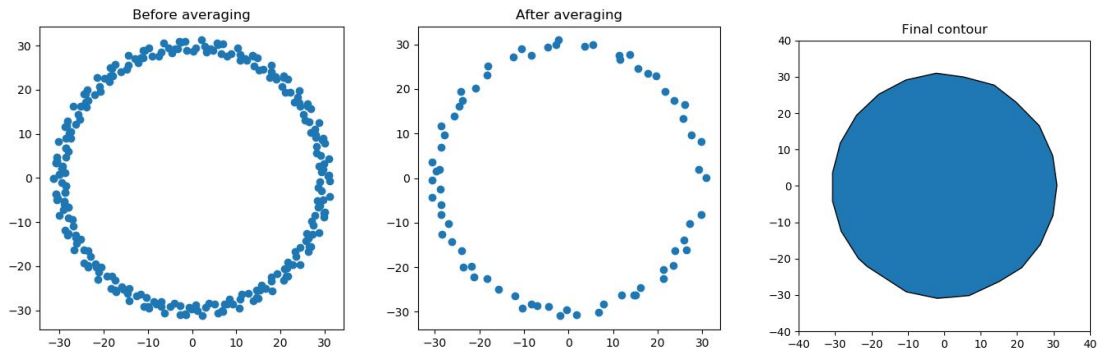


Figure 17: Evolution of a point cloud through three algorithm stages. Left: points after the preprocessing step; middle: points after the averaging step; right: final contour generated.

5.5.4 Connecting Layers

Once a contour is generated, it must be connected to the layer below it. Due to the stipulations of STL files, these connections must be in the form of triangles (Figure 18), with two points on the bottom layer and one point on the top (or vice versa).

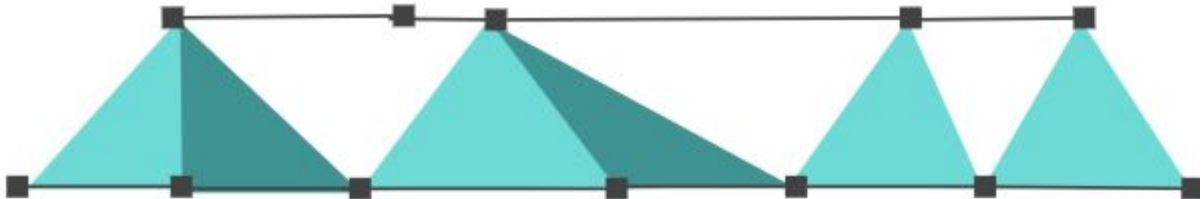


Figure 18: Triangles generated when connecting two layers. Only the triangles originating from the bottom layer are shown.

The connections occur in two steps: first, all line segments from the lower level are connected to the closest point on the higher level. Second, the line segments from the top are connected to the bottom. If there is a natural connection, bounded on both sides by already-created triangles, that connection is made. Otherwise, the algorithm runs through a loop to find the most likely point to connect to, making sure that the final triangle does not leave holes in the contour or overlap with existing triangles. Once a connection is made, the resulting triangle is printed out to the STL file.

The STL file is closed off on the top and bottom by a full triangulation of the top and bottom layers. This is achieved by using the `triangulate` function in the Python `shapely` module.

5.5.5 Verification

In order to verify the set of algorithms discussed here, simulated data was generated, then run through the entire suite of programs. The simulated layers were circles with uniform random noise added in; they were manipulated and arranged to form cones, cylinders, and rounded rectangles. All of the shapes tested were recreated faithfully by the algorithm (Figure 19). All models shown are sliceable in the PrusaSlicer software, which means that they can be successfully printed by a Prusa 3D printer.

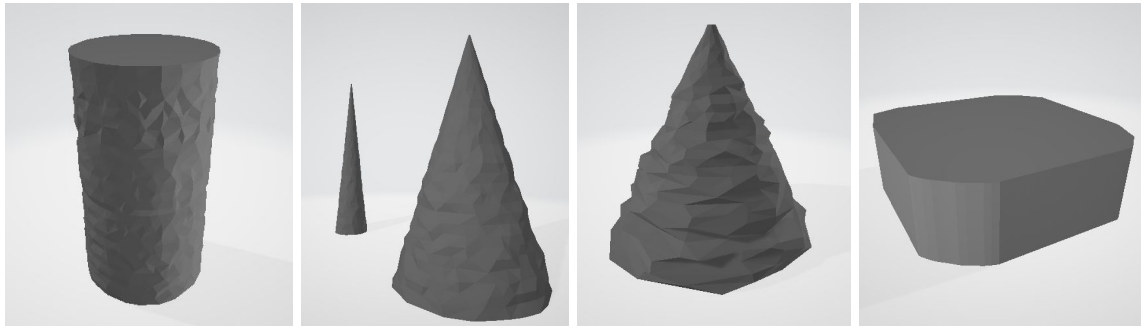


Figure 19: Examples of 3D models created when noisy simulated data is passed through all stages of the STL generation algorithm.

5.6 User Interface

The following user interface (UI) windows are run on the Windows PC. The color and font choices shown here were picked to emulate the UI of the Prusa family of 3D printers. They are synchronized throughout the system, including in the project logo.

5.6.1 Menu Window UI

The menu UI is the starting point of the scan process. In this GUI window, the user selects their scan parameters and begins the scan (Figure 20). These parameters are: the planar resolution, which determines the size of the averaging grid (section 5.5.2); the vertical resolution, which determines the z -step of the scanning tower; and the number of rotations of the platform on each layer. The menuWindow class allows defaults to be set for all of these parameters; it also allows for easy customization of the options provided. Once the scan is started, the parameters are converted to the appropriate units and sent to the Raspberry Pi through a shell command (section 5.4).

The menu UI is built in Tkinter, the Python GUI library for Windows and Unix machines. It is accessed as a class called menuWindow.

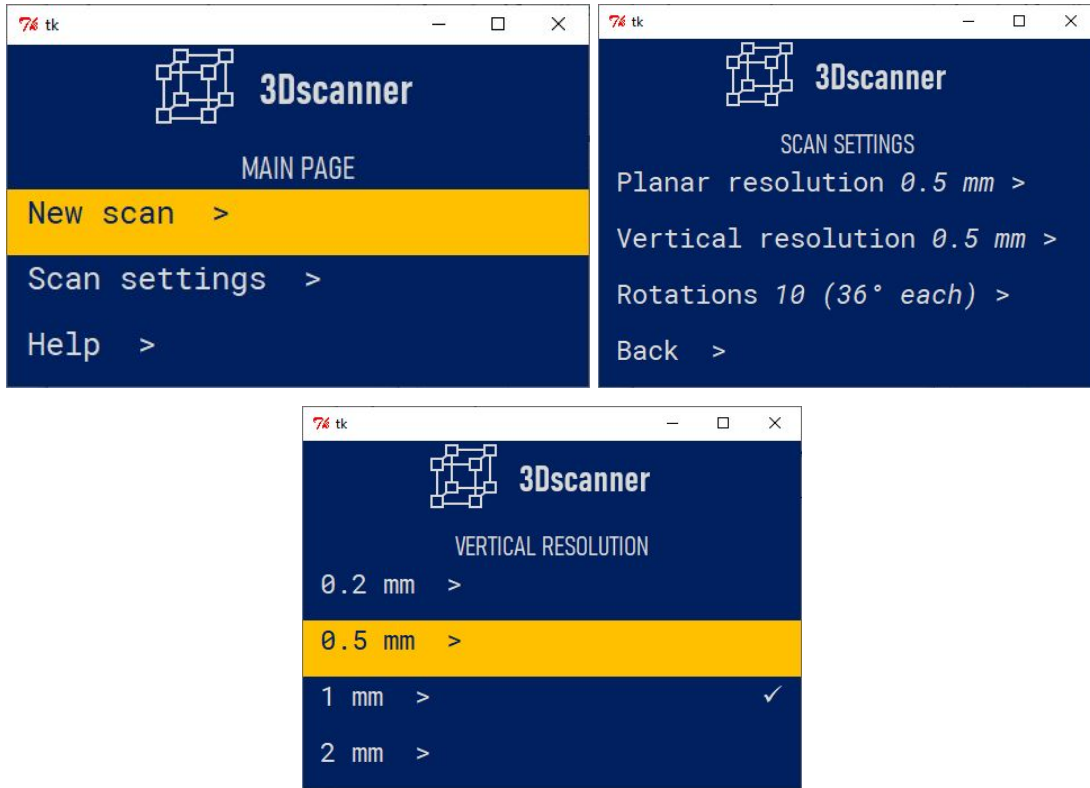


Figure 20: Sample screens from the menu UI. Top left: the main page; top right: the scan settings page; bottom: setting an individual scan parameter.

5.6.2 Progress Bar UI

The progress bar UI window tracks the progress of the scan (Figure 21). The denominator, or the number of total layers, is determined by the prescan routine (section 5.3.4). An interrupt routine increments the numerator every time a layer is finished processing. It is built using the `ProgressBar` widget in Tkinter, and accessed as a class called `progressBar`.

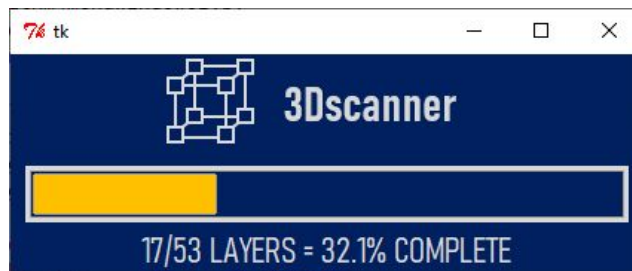


Figure 21: Example screen for the progress bar UI.

Chapter 6: Schedule and Division of Labor

6.1 Project Schedule

Figure 22 shows the actual Gantt chart that we worked from to ensure that we were on track through the whole process.

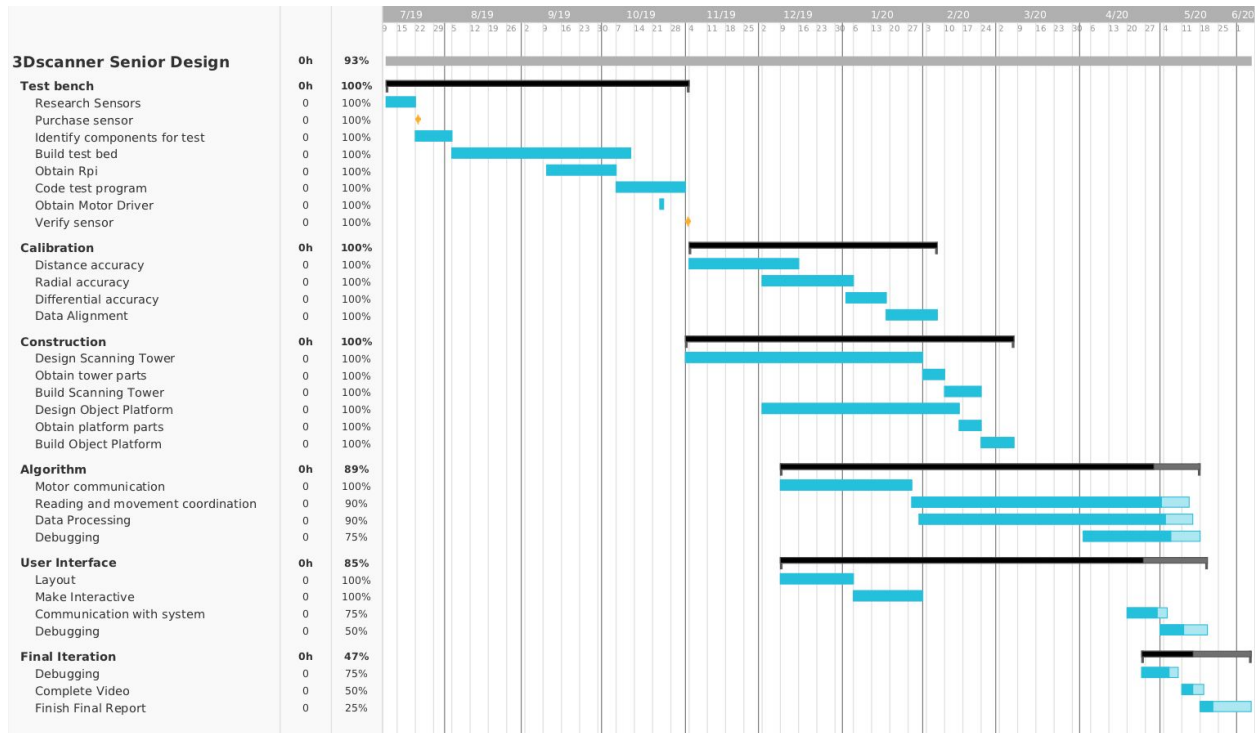


Figure 22: Full project Gantt chart

To highlight the most important parts of the schedule:

- Sensor Research and Choice.....July 2019
- Creation and Control of Test Bench.....Aug. 2019 - Oct. 2019
- Sensor Calibration.....Nov. 2019 - Jan. 2020
- Physical System Design/Construction.....Nov. 2019 - Feb. 2020
- Scanning Algorithm Creation/Refinement.....Dec. 2019 - April 2020
- User Interface Development.....Dec. 2019 - April 2020
- Final Debugging.....April 2020

6.2 Division of Labor

General tasks and responsibilities for this project were divided up among our members based on our respective backgrounds. Our division of labor is as follows:

- Thomas Heckman was in charge of the low-level software, microprocessors, and motor control
- Robert Kalb was in charge of designing and constructing the physical system as well as general project support, troubleshooting, and management
- Shreyes Nallan was in charge of the high-level software, data processing, STL generation, creation of visuals, and user interface

Chapter 7: Findings and Results

7.1 Findings

Once the scanner was finished and ready to take scans, we focused on scanning objects to create printable STL files. Through this process, we found out that lidar has two main limitations.

The first limitation is that glossy black surfaces don't reflect infrared light very well - the beam gets absorbed or deflected, and doesn't get back to the sensor. We tried scanning the computer mouse shown in the first two pictures of Figure 23, and the third picture in the figure shows the resulting STL. Notice that it mainly captured the matte black and blue surfaces, and it only captured a little bit of data where it scanned the gloss black head on.



Figure 23: Scanning a computer mouse (top) and the resulting STL (bottom)

The second limitation is shown in Figure 24 - a low angle of incidence is important for getting good data. When we scanned across the surface of the blocks in the picture on the left,

we got no data because the angle of incidence was so large. In most cases, we can adjust the shape using the rotating platform to get those points, as shown in the second picture where we did see distance data along the surface of the block. However, for some shapes (like the steep curvature at the top of the computer mouse in Figure 23), this is impossible because the object doesn't rotate on any other axis during a scan, so these shapes are not particularly conducive to being scanned by LIDAR.

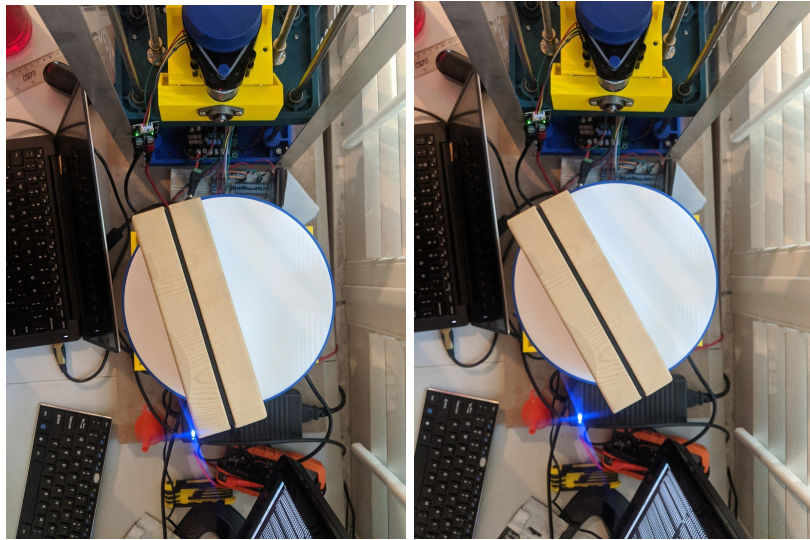
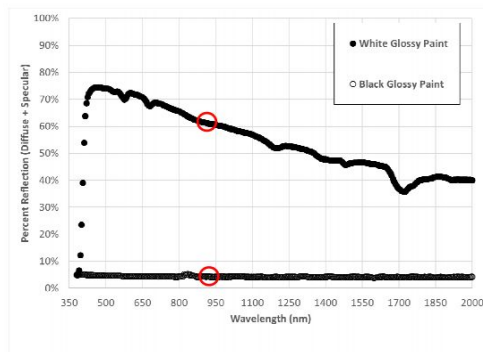


Figure 24: Scanning the blocks at originally (left) yielded no data because of the angle of incidence, but rotating them (right) gave the data we expected

These limitations were confirmed in a study presented by Christopher M. Seubert of the Ford Motor Company. Ford is experimenting with using LIDAR as a primary sensor for self-driving vehicles, so they tested and came to similar conclusions about the limitations of LIDAR. Figure 25 is a slide from the report confirming the poor reflectivity of near-infrared light on glossy black surfaces, and Figure 26 is a slide that confirms the dramatic effects that the angle of incidence has on the data captured by a LIDAR sensor.

Total Reflectivity of Paint

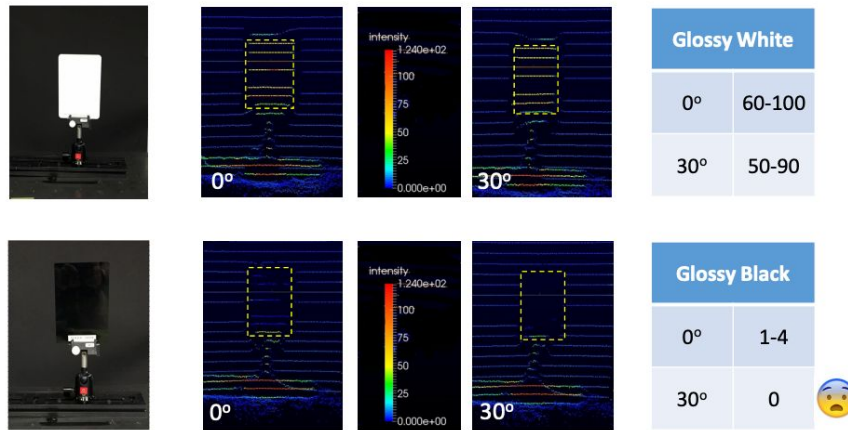


- Not all surfaces and colors reflect visible and IR light equally
- Basecoat color (and primer to a lesser extent) affect the reflectivity of the paint system
- LIDAR units emit and detect only 905 nm light
- 905 nm: Standard glossy white (62%); Standard glossy black (4%)



Figure 25: Slide from a presentation by the Ford Motor Company confirming the poor results yielded when scanning glossy black surfaces with LIDAR

LIDAR Visibility Data



LIDAR "intensity" is returned flux relative to a perfect diffuse reflector (error $\pm 30\%$)



Figure 26: Slide from a presentation by the Ford Motor Company confirming the poor results yielded when scanning a surface with high angle of incidence using LIDAR

7.2 Results

With everything said and done, our scanner is able to scan most objects! Figure 27 shows a can of Ice Cubes gum, whose shape works well with our scanner's capabilities and was the first object that we completed a full 3D scan of. The image in the middle of Figure 27 shows the STL output from our scanner, and the photo on the right shows the physical 3D print of that STL! As can be seen from the figures, the scan is a good representation of the object.

Unfortunately, our motor controller broke as soon as we got down to the last few steps, and one motor port wouldn't output any signal. As a result, this scan is not using dithering to smooth the surface. Since the scanner still retains full functionality and only loses performance by not dithering, we decided to temporarily sacrifice dithering for the sake of getting a scan to work. If, in a future iteration, we get dithering online, the scans we take can only increase in accuracy.

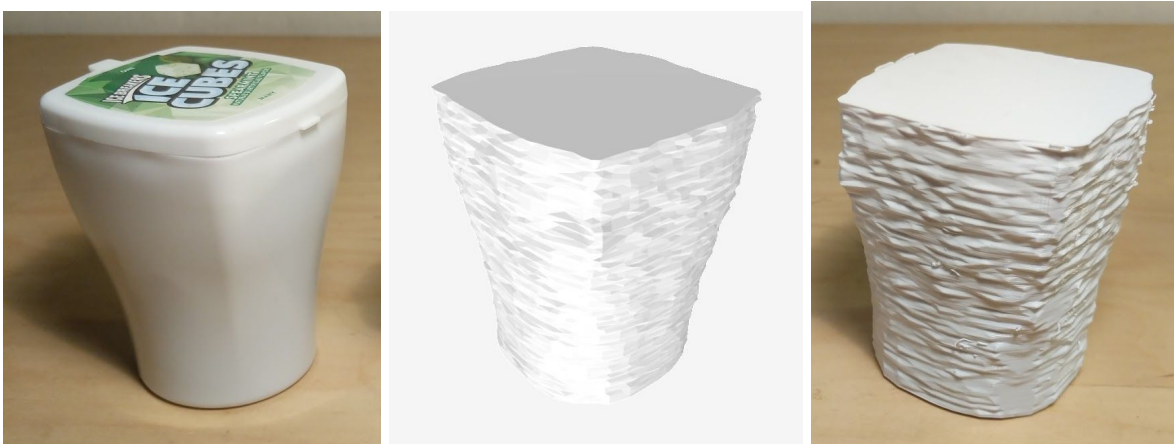


Figure 27: A can of Ice Cubes gum that was scanned (left), the STL output from our 3D scanner (middle), and the 3D print of the STL (right)

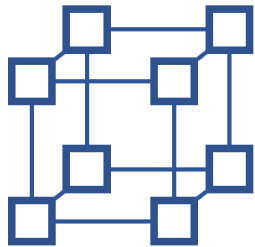
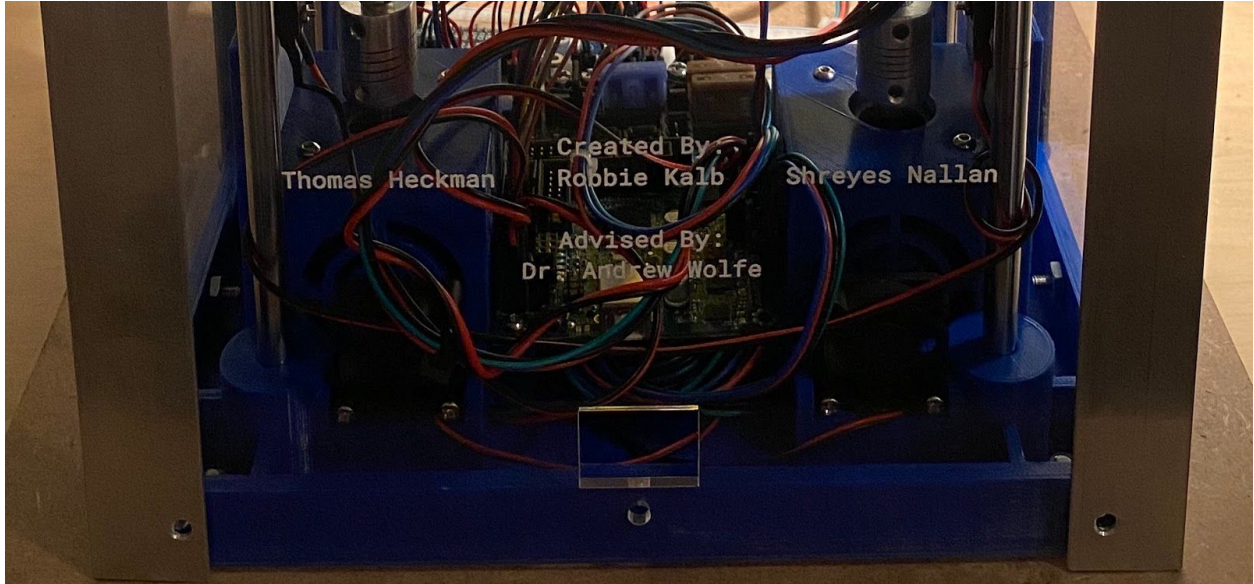
7.3 Conclusion

Through this project, we discovered a couple of important limitations of LIDAR in general, and we created a working 3D scanner. While dithering in our scanner can still be fixed to improve dimensional accuracy, our STL output and 3D print are clearly recognizable as the original object that was scanned. Any individual or small business can now use our scanner as is for precise and affordable scans to aid in rapid-iteration prototyping. This project is now complete; however, future senior design teams can extend our work to achieve even higher accuracies and scans of objects with glossy black or geometrically challenging surfaces.

Ultimately, we managed to fulfill all of our original project objectives. We bought an \$80 LIDAR sensor off the shelf and built an easy-to-use, interconnected system around it that is able

to create three-dimensional digital representations of real-world objects using nothing but the position data obtained from the sensor.

And that, we think, is pretty special.



3Dscanner

References


Information from:


- “Laser Classes.” Laser Safety Facts, www.lasersafetyfacts.com/laserclasses.html.
- Liu, Xing, et al. “An Optimum Condition of the Dithering Signal against the Quantization Noise in ADC.” 2014 International Conference on Wireless Communication and Sensor Network, 2014, doi:10.1109/wcsn.2014.18.
- Seubert, Christopher M. *IR Reflectivity of Paint: Autonomy and CO2 Emissions*. Ford Motor Company, <https://detroitcc.org/wp-content/uploads/2018/07/IR-Reflectivity-of-Paint-Autonomy-and-CO2-Seubert.pdf>
- “YDLidar Downloads.” YDLIDAR, 2015, www.ydlidar.com/download.

Images from:

- “HE3D Open Source Ciclop DIY 3D Systems Scanner Kit.” Amazon.com, www.amazon.com/HE3D-Systems-Advanced-Injection-Plastics/dp/B076B9SXX8.
- “LIDAR Sensor Visualization.” Imveurope.com, <https://www.imveurope.com/news/photonic-vision-develops-lidar-sensor-video>.
- “Matter and Form 3D Scanner V2.” PCMag, 5 Oct. 2018, www.pcmag.com/feature/364076/matter-and-form-3d-scanner-v2-quickscan.
- “YDLidar X4 Sensor.” Amazon.com, https://www.amazon.com/LIDAR-053-YDLIDAR-Frequency-Sweeping-Positioning/dp/B07JGV5JTG/ref=pd_sbs_229_t_1/147-3536678-9630749?_encoding=UTF8&pd_rd_i=B07JGV5JTG&pd_rd_r=003f4cd7-8df2-49eb-a0de-f5e1b8c06c24&pd_rd_w=ThOxk&pd_rd_wg=rtIDX&pf_rd_p=5cfcfe89-300f-47d2-b1ad-a4e27203a02a&pf_rd_r=2D1MY7SN5CX64DJXG6NH&psc=1&refRID=2D1MY7SN5CX64DJXG6NH.

Appendix A: Senior Design Conference Slides



SCHOOL OF ENGINEERING




3Dscanner

Created by:
 Thomas Heckman, Robbie Kalb, and Shreyes Nallan


Advised by:
 Dr. Andrew Wolfe

www.ncsu.edu/engineering



SCHOOL OF ENGINEERING

Presentation Outline

- Background and Objectives
- Functional Block Diagram and Overview
- Project Outcomes
- Performance and Reliability
- Full Project Schedule
- Summary
- Bibliography and References

www.ncsu.edu/engineering




SCHOOL OF ENGINEERING

Original Objectives

In this project, we plan to design, prototype, and troubleshoot a low-cost, high-performance 3D scanner by April 2020 for less than \$2,000. By the end of this project, we hope to have:


- A fully functioning prototype
- A developed scanning algorithm
- A complete user-interface
- At least one object scanned and 3D printed


www.ncsu.edu/engineering



SCHOOL OF ENGINEERING

What is a 3D Scanner?

A 3D scanner is a device that creates digital, three-dimensional representations of real-world objects, which can then be edited, manipulated, and/or physically recreated by a 3D printer or CNC.



www.ncsu.edu/engineering



SCHOOL OF ENGINEERING

Background

Don't 3D scanners already exist?



Low Performance




High Cost




New Technology


www.ncsu.edu/engineering




SCHOOL OF ENGINEERING

Project Overview

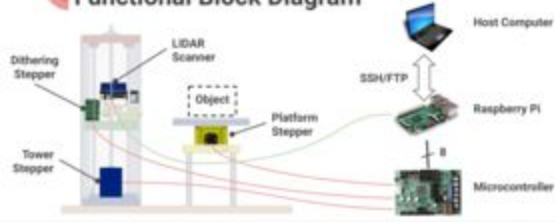
Use LIDAR to create a high-performance, low-cost 3D scanner that can match 3D printer resolution limits.





www.ncsu.edu/engineering



SCHOOL OF ENGINEERING

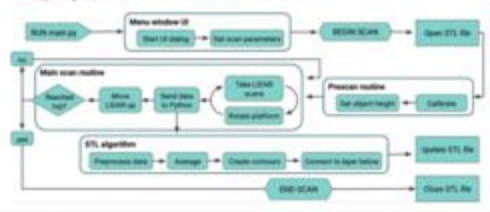
Functional Block Diagram




www.ncsu.edu/engineering



SCHOOL OF ENGINEERING

Operational Flowchart



www.ncsu.edu/engineering




Project Plan: Sensor and Calibration

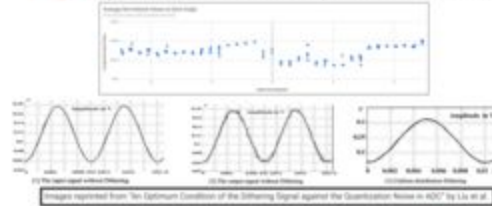


www.ncsu.edu/engineering

© 2020 North Carolina State University



Project Plan: Sensor Specs and Dithering



www.ncsu.edu/engineering

© 2020 North Carolina State University



Project Plan: Mechanical Design

Original Designs:



www.ncsu.edu/engineering

© 2020 North Carolina State University



Project Plan: Hardware and Software



www.ncsu.edu/engineering

© 2020 North Carolina State University



Project Plan: UI and Data Processing

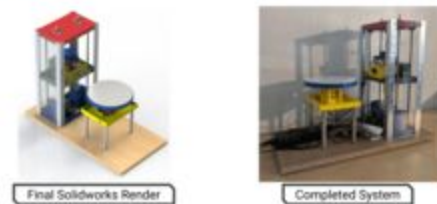


www.ncsu.edu/engineering

© 2020 North Carolina State University



Project Outcomes: Physical System

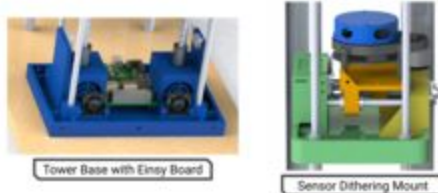


www.ncsu.edu/engineering

© 2020 North Carolina State University



Project Outcomes: Scanning Tower

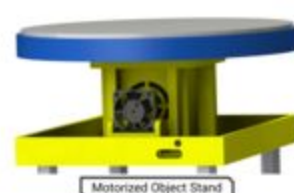


www.ncsu.edu/engineering

© 2020 North Carolina State University



Project Outcomes: Rotating Platform



www.ncsu.edu/engineering

© 2020 North Carolina State University



Project Outcomes: State Machine



Project Outcomes: Remote Control



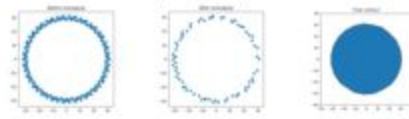
Project Outcomes: User Interface

- **Menu window:** sets scan parameters and begins the scan routine.
- **Progress bar:** tracks the progress of the scan. Integrated with the STL algorithm.
- Created in Tkinter (the Python GUI library)



Project Outcomes: Processing

- **Preprocess** point cloud: snap all points to the same axes
- **Average:** apply a uniform averaging grid
- **Create a 2D contour:** use the alphashape Python module
 - Find the most probable shape



Project Outcomes: STL File Generation

- **Connect** each layer to the layer below
 - Try to connect to nearest neighbor
 - Patch over holes
- **Print** triangles to STL file



Performance



Reliability



Gantt Chart



