2D23D

Authors: Alex Patel, Chakara Owarang, Jeremy Leung: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— It cannot be overstated how much traditional 2D camera technology has shaped our progress and culture. However, 2D images do not effectively capture the details of an object when it is rotated around. This information is potentially crucial to archaeological archivists that want to preserve the object in its entirety, with the potential to 3D print and recreate these objects. Our goal is to be able to accurately map and scan the physical object into a digital three-dimensional representation for the purpose of archeological documentation. Our design goals are to design this device to be able to be used by a non-technical audience: easy to be set up, portable, and most of all, accurate in scanning. Currently, manual 3D modeling by users is time-ineffective, tedious, and costly, as well as prone to errors. Our project will uniquely address these issues in being cost-effective, user-friendly, portable, and accurate.**

***Index Terms*— Iterative Closest Point algorithm, point cloud data, pairwise registration, laser stripe triangulation, stepper motor**

1 INTRODUCTION

The main application use case of our project is to be able to scan archaeological objects for preservation in a 3D format. The goal is to be able to accurately map these objects into a widely usable 3D format for documentation and reproduction via 3D printing. Thus, our requirements will be focused around the accuracy of the scan. However, the device should also be easily usable and portable, and thus must require a reasonably fast scanning time. We thus aim to have these two accuracy requirements: 90% of non-occluded points must be within 2% of the longest axis to the ground truth model, and 100% of non-occluded points must be within 5% of the longest axis to the ground truth model. Other requirements will be covered in the Design Requirements section (2). Our basic approach will be to use a projected laser stripe along with a high resolution digital camera to scan the object atop a rotating platform. Our approach will also allow for combinations of multiple scans in case some angles of the object are hidden from the scan.

There are currently several other competing technologies which mostly involve either digital cameras or depth cameras. However, scans that combine multiple views from a digital camera (multi view stereo reconstruction) tend to have lower accuracy and are unable to detect concavities in the object accurately, and also requires very strict lighting setups to ensure the best scans. We also considered using depth cameras to give a 2D depth map which would tell us depth of each pixel scanned, but most depth cameras do not give very precise accuracy information, and depth cameras tend to be much less precise than using laser based approaches. We will discuss more details of the many different approaches we considered in depth in the Design Trade Studies section (4).

2 DESIGN REQUIREMENTS

a. Accuracy

Our first requirement is that of accuracy. We aim to satisfy this requirement: 90% of non-occluded points must be within 2% of the longest axis to the ground truth model, and 100% of non-occluded points must be within 5% of the longest axis to the ground truth model. To be able to test this, we will have two possible methods where we'll try both. The first method involves finding ground truth 3D models of common objects such as a solo cup or a coke bottle - however we may want to consider spray painting the coke bottle so that the transparency of the sides don't mess up the laser. The second method will be 3D printing 3D models that we find on the internet, then scanning these 3D printed models and comparing with the ground truth model. For the 3D printed models we will allow for an extra 1mm buffer since there is a fair bit of inaccuracy induced from 3D printing. We will compute the accuracy number by computing mesh vertex distances from each point in our constructed mesh to the surface of the ground truth mesh. These distances will be computed using the same principles as the Hausdorff distance, with the following method:

Algorithm: From every query point,

- the nearest vertex
- the nearest point on the edges and
- the nearest point on the triangle's surfaces

is calculated and the minimum distance out of these three is returned.

Algorithm 1 Definition of A_f

Execute A_g
while *Receive query for* $q \in X$ *from* A_g **do**
  Query $Challenger_f$ with q and receive response r return reverse(r) to A_g
end
When A_g outputs a guess b', output b' as the guess for A_f

These results must match our accuracy requirement as stated above.

b. Usability and Portability

Our next requirements involves usability and portability. The input object must be 5cm to 30cm along every axis, and have a maximum weight of 7kg. Thus, our platform will be tested with a 7kg load and we will see if the platform is able to withstand that weight without warping, as well as rotate the object without hindrance to rotational velocity. We also have other usability requirements such as the device being easy to setup and outputs a common 3D format that we will be able to input to a 3D printer. These requirements are easily evaluated and do not require any special quantitative tests. We will also test usability by doing user testing and evaluating survey responses.

c. Efficiency

We also have a requirement for efficiency. We will allow one minute total time for the scan including the rotation and the processing time. The rotation time should be well under 30 seconds, which gives more than 30 seconds for processing time, which will involve point cloud construction, filtering, and triangulation into a mesh. This will simply be measured by timing the process from pressing start to obtaining the 3D mesh. Note that the time for calibration will not be included since this is a one-time operation which will have amortized time cost across multiple successive scans.

Related to efficiency, we will also test software components we create against well-known open source library functions. This is due to our goal of implementing several components of the software pipeline to be optimized for the Nvidia Jetson GPU. We will first start off with open source code, but slowly replace components one by one with our optimized code. Thus, we must unit test all software components extensively, on real data and manufactured test cases. This includes filter routines, computer vision routines, mesh generation, and CUDA kernels. We will be testing for processing speed and correctness of these few components.

d. Affordability

Our final requirement is affordability. The whole system should cost less than $600. This is also to ensure that we are bridging the gap since commercial 3D scanners cost much more than $600.

3 ARCHITECTURE OVERVIEW

Allow us to start from the user story to link into our architecture design.

The user first presses the start button, and if the device is not calibrated, the program will prompt the user to insert the laser plane calibration object such as a checkerboard pattern and perform the calibration. Then, the user will start the scan and a point cloud will be generated. If additional scans are required, the user will rotate the object in a way that provides more information about the bottom or the concavities of the object, then these point clouds will be coalesced with pairwise registration. After a final point cloud is obtained, we will perform triangulation to output the final mesh.

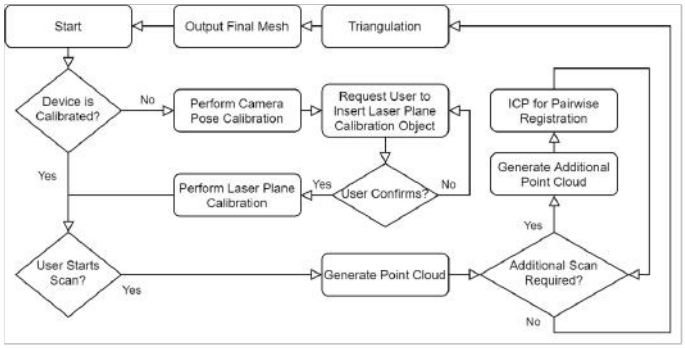

Figure 1: User Story Flowchart

This process leads into our software pipeline design, which directly corresponds with the user story. Note that the pipeline diagram above assumes the camera is already calibrated. We will use the camera image to perform laser detection, and for each image with the laser in it, we will generate Cartesian coordinate values based on how the laser warps around the object. We will also remove the background points and the points from the turntable. Combining this with rotational information we have from the stepper motor driver, we will be able to generate a point cloud, which will be passed through some noise reduction and outlier removal filtering. If multiple scans are required, then we will use pairwise registration to combine the two point clouds and output a final point cloud. Then, we will use triangulation on the point cloud to produce an output mesh. Triangulation basically involves forming triangles on the surface using close neighboring points, which produces a triangle mesh that can be easily 3D printed.

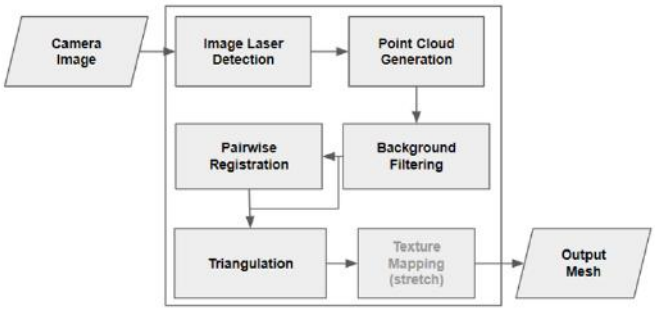

Figure 2: Software Pipeline

Based on this software pipeline, we can have a more comprehensive system specification diagram now, which includes components for the stepper motor, GPGPU, controllers, and more. Please refer to Figure 10: System Specification Diagram in Appendix A for our full system speci-

fication diagram.

The proposed integrated hardware platform includes a 4-core embedded system with programmable GPGPU, interfaces between the system and other components of our design, as well as logical connections between software elements. From the camera, data comes into the embedded system by USB. This data is processed by our core system controller process, which will either initiate the execution of GPU kernels and subroutines for 3D scanning using the image, or tell the motor controller to rotate a certain about to capture additional images, depending on the state of the controller within the process it is trying to accomplish. A software state machine within the core system controller process will match the behavior that can be found in Figure 1: User Story Flowchart based on user input and the current executing operation.

A motor controller process will be responsible for setting the GPIO pins to interface with the stepper motor through the motor driver, setting the rotation to a specific angle. A breadboard and wires will be used to interface between the motor driver and the stepper motor. The stepper motor angle will be communicated to the motor controller from the core system controller based on the stage of the scan. An additional process will be used to handle interrupts an processing related directly to user input. Having this additional indirection for user I/O helps enforce protection boundaries between our system and the outside world (only very specific types of input can make it through). Since we have 3 processes and 4 cores on the embedded system, each process can have a core dedicated to it to maximize performance. For the motor controller, we can change the kernel scheduling algorithm used to allow it to be prioritized for real-time deadlines.

Besides the physical components and the software processes, various GPGPU kernels and subroutines will exist within the system to carry out optimized execution of our required algorithms. A GPGPU kernel describes the execution of a single core in a many-core programming platform. Launching a GPGPU kernel first copies data from DRAM into the GPU's memory, sets up the scheduling and synchronization structures of the GPGPU to support the provided kernel, and begins execution of each execution core "simultaneously". The result is then copied back into DRAM. We will implement a majority of our optimized algorithms with compute kernels, including calibration, ray-plane intersection, iterative closest points for pairwise registration (single object multiple scans), and other geometric operations.

4 DESIGN TRADE STUDIES

In this section, we will discuss some design trade-offs and evaluation of different options for the design of several components of the project.

4.1 Scanning Sensor

The specific sensor setup we will use for our project required the most research and evaluation to compare many different possible methods. We started from our requirements to choose this sensor. In the most extreme case, to accurately capture an object whose longest axis is 5cm, in order to meet the requirement that 90% of reconstructed points are within 2% of that 5cm, we must have points within 1mm to the ground truth model. From this, if we consider the number of samples we need across the surface, we must have accurate samples within 1mm for each direction (X, Y along the surface). Since the surface itself is a continuous signal we are sampling from, we can compute the Nyquist sampling rate as being every 0.5mm along each direction of the surface.

Now considering the rotational mechanism, the largest radius of the object from the center of the rotating platform will be within 15cm. We need then a single rotation per data capture to be such that the amount of the surface rotated passed the sensor is less than or equal to 0.5mm. This gives us: $\frac{0.5mm}{150mm} = 0.0033$ radians of rotation per sample.

There are five main types of sensors for 3D scanning that we have extensively considered:

1. **Contact sensors:** These sensors are widely used in manufacturing. A Coordinate Measuring Machine (CMM) or similar may be used, which generally utilizes a probing arm to touch the sensor, and through angular rotations of the joints the coordinates of each probed area can be computed. This is a non-option for our application, both for price and the fact that we should not allow a large machine to touch timeless archeological artifacts.

2. **Time-of-flight sensors:** By recording the time between sending a beam of light and receiving a reflected signal, distance can be computed to a single point. The disadvantage of this approach is that we can only measure times so precisely, and the speed of light is very fast. With a timer that has 3.3 picosecond resolution, we are still not within sub-millimeter depth resolution, which is not reasonable for this project. Time-of-flight sensors in the domain of 3D scanning are more applicable to scanning large outdoor environments.

3. **Digital camera:** By taking multiple digital photographs from many perspectives around the object, computer vision techniques can be used to match features between pairs of images, and linear transforms can be computed to align such features. After feature alignment, and depth calculation, point clouds can be generated. Computer vision techniques to accomplish the above include Structure from Motion (SFM) and Multi-View Stereo reconstruction (MVS). This approach has a fundamental flaw: concavities in the object to be scanned cannot be resolved, since cameras do not capture raw depth data. Surface points

within the convex hull of an object cannot be easily distinguished from points on the convex hull. The digital camera methods also have very strict lighting requirements to produce accurate scans and will definitely suffer from accuracy compared to more precise approaches like the laser-based ones. This is an immediate elimination for our project, since archeological objects may have concavities, and we do not want to limit the scope of what type of objects can be captured, along with accuracy being our primary focus of the requirements.

4. **Structured/coded light depth sensor (RGB-D Camera):** The idea of such a sensor is to project light in specific patterns on the object, and compute depth by the distortion of the patterns. Such sensing devices have become incredibly popular in the 3D reconstruction research community with the consumer availability of the Microsoft Kinect. The original Microsoft Kinect only has 320 pixels wide of depth information for a single depth image. With the upper bound of 30cm across the surface of the object, this results in 30cm/320px = 0.094cm between each pixel, which does not meet our sensor requirement of being able to detect differences within 0.5mm (0.05 cm). The newer Microsoft Kinect v2 actually uses a time-of-flight sensor, and thus does not get measurements more accurate than 1mm depth resolution. Intel RealSense has recently released new product lines for consumer and developer structured light depth sensors that are very affordable. Most notably, for short range coded light depth sensing, the Intel RealSense SR305 offers 640480 pixel depth maps, which correlates to 30cm/640px = 0.047cm, which is within our requirements. However, Intel does not advertise any specific depth resolution for the device, and we cannot guarantee sub-millimeter depth accuracy - these depth cameras are more commonly used to scan a whole room or scan objects from a meter distance. Perhaps we can obtain a better accuracy figure after some extensive testing but it may not be valuable considering we can just build our own laser stripe triangulation sensor. Since this method only relies on the camera, lighting environment and material of the object can have much greater influence compared with laser triangulation. It also requires a significant algorithmic effort after data collection to reduce noise and correlate the views.

5. **Laser point triangulation:** The principle of the single point laser sensor is that an emitting diode and corresponding CMOS sensor are located at slightly different angles of the device in comparison to the object, so depth can be computed by the location on the sensor the laser reflects to. Generally the position of the laser on the surface is controlled by a rotating (or pair of rotating) mirrors. We can assume that such a sensor is affordable, can easily measure with resolution of less than 0.5mm, and that we will not likely encounter any mechanical issues. However, the total number of distance measurements we are required to record is:

$$\frac{2\pi}{0.0033\text{rad}} \cdot \frac{300mm}{0.5mm} = 1142398 \text{ points}$$

Assuming the sensor has a sampling rate of about 10kHz (common for such a sensor), 1142398 points divided by 10000 points per second gives us 114.23 seconds theoretical minimum capture time with one sensor. From our timing requirement, assume that half of our time can be attributed to data collection (30 seconds). Then, with perfect parallelization, we could achieve our goals with $114.23s/30s = 3.80 \approx 4$ sensors collecting concurrently. However, with our budget, this is not achievable. Even if we had the budget, it is possible for systematic errors in, for example, the mounted angle of a sensor, to propagate throughout our data with no course for resolution. To add another set of sensors to mitigate this error, we would be even farther out of our budget. 8 sensors x $300 per sensor (low-end price) gives us $2400 for this sort of setup. Note these calculations are unrelated to any mechanical components, but are directly derived from required data points.

To make budget not an issue for the single point laser triangulation method, we could choose to adopt cheaper sensors, such as those with under 1kHz sampling rate. Performing the same calculations as above with 1kHz sampling rate shows us that we would require 39 sensors to meet our timing requirement, which is well out of the realm of possibility (and this is not accounting for error-reduction, which may require 78 sensors). If we did not purchase this amount of sensors, we would drastically under-perform for our timing requirement.

6. **Laser stripe triangulation:** Fortunately, there is an alternative to single-point laser triangulation. We may use a laser stripe depth sensor, which gets the depth for points along a fixed width stripe. This would improve our ability to meet our timing requirements significantly. Such devices are not easily available with high accuracy to consumers, but are usually intended for industry and manufacturing. Because of this, we would have the responsibility of constructing such a device ourselves. We have considered the risk of building our own sensor since none of us are experts in sensors and electronics. However, as long as we can find an affordable laser stripe with sufficient brightness, our laser stripe sensor should not suffer in accuracy. A laser stripe sensor consists of a projected linear laser source and a digital camera. Many laser stripe sensors use a CCD camera instead to avoid the projection brightness issue completely, but these CCD cameras tend to be too expensive and

would put an unnecessary strain on our budget, and we can do just as well with a digital camera.

After a calibration process to determine the intrinsic camera parameters as well as the exact angle and distance between the camera and the laser projector, linear transformations may be applied to map each point from screen space to world space coordinates. Because of the ease of achieving sub-millimeter accuracy, and the relative independence on lighting conditions and materials that photogrammetry is harmed by, we plan on constructing a laser stripe depth sensor with a digital camera.

After extensively considering the many available sensor options, we can see that contact, time-of-flight, and digital cameras are clearly not options for us to explore, and depth cameras cannot guarantee as much accuracy as laser stripe triangulation. Given that the main focus of our project is the accuracy of the scans, our sensor setup should be able to provide as much accuracy as possible so as to not cascade errors down the pipeline.

4.2 Platform Material

To fit our *Usability* requirement, the platform must be able to withstand an object with dimensions from 5cm to 30cm and weighing up to 7kg. The platform itself will be a circular disc with a diameter of approximately 38.1cm. We performed a rough estimation to compute the stress that the platform needs to be able to handle. From our maximum input object's mass of 7kg, the maximum gravitational force that it can exert on the platform is around $68.67N$. The lazy susan bearing our team might end up using has an inner diameter of around 19.5cm (or 0.0975m radius). This would give us an area of around $0.03m^2$ that would not have any support. We simplified the stress analysis of this design down but it should still give a good estimation of the stress the object would apply.

$$\text{Stress} = \frac{F}{A} = \frac{68.67N}{0.03m^2}$$

which is around $2300\frac{N}{m^2}$.

After getting the stress, we did more research on the material that would be able to handle this much stress, be cost-efficient, easily accessible, and easy to use (cut, coat, etc). We did some optimization based on cost and mass-to-stiffness ratio to narrow down the number of materials we had to do research on. Below is an image of the optimization graph. Note that we only looked into plastic and natural materials as they are easier to use and more easily accessible. The line in the second image is the optimization line.

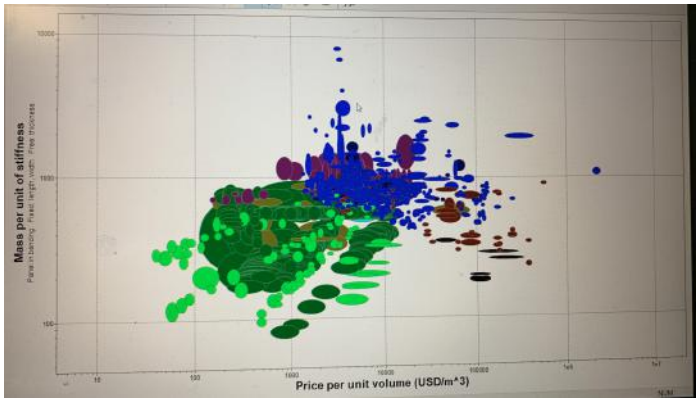

Figure 3: Material Optimization Graph

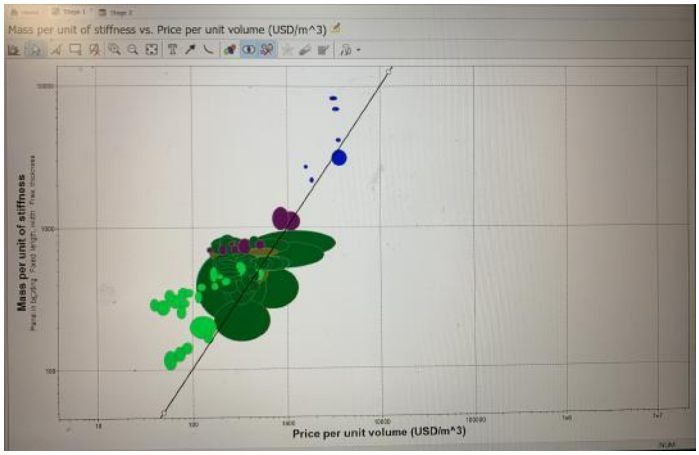

Figure 4: Optimized Material

After that, we narrowed the materials down more to 3 materials: plywood, Epoxy/hs carbon fiber, and balsa. Table 2: Material Tradeoffs Analysis in Appendix A shows the tradeoffs between different main properties that would affect our decision. Young's modulus, specific stiffness, and yield strength are mainly to see if the material would be able to handle the amount of stress the object would exert on it or not. The price per unit volume is to keep this within our project's constraint. The density is used to compute the mass of the platform (for computing the torque required and to stay within our Portability requirement).

From the table, we can see that carbon fiber is the strongest but is relatively expensive and might not suit our project well. Balsa is very light but is not as strong (even if the values here are still higher than the stress we computed, it might be because of the simplified stress analysis we did). Thus, our group decided to use plywood which is strong, inexpensive, easy-to-cut, and not too heavy. With plywood, the maximum mass our of platform would just be around $0.6kg$ (computed using density and dimensions of the platform).

4.3 Motor

The final part of the main platform design is to choose the right motor for the project. The main 2 motors we looked into to rotate the platform are the servo motor and the stepper motor. A servo motor is a motor coupled with a feedback sensor to facilitate with positioning for precise velocity and acceleration. A stepper motor, on the other hand, is a motor that divides a full rotation into a number of equal steps.

To figure out the motor used, we computed the torque required to rotate the platform with the maximum object size. From the density and dimensions of the platform, we computed that the plywood platform would weight around 0.64kg and carbon fiber would be around 1.2kg (We still accounted for the heaviest material and strongest material in case of a change in the future). From that we computed the moments of inertia which is around $0.024kgm^2$. For the input object, we used maximum size and different dimensions and shapes to cover most cases, the maximum moments of inertia computed is around $0.1575kgm^2$. Thus, the total maximum moments of inertia is less than $0.2kgm^2$. Refer to Table 3: Input Object Moments of Inertia in Appendix A for the full calculations. To get the torque, We also estimated the angular acceleration needed. We need at least a rotation of 0.0033 rad per step to capture enough data to meet our accuracy requirement. Assuming that 10% of the time requirement, which is 6s, can be used for data capturing (so that we have more buffer for the algorithmic part even if we anticipate 30s for data capturing), we would get that the angular velocity is around $3\frac{rad}{s}$. Assuming we want our motor to be able to reach that velocity fast enough (0.5s), we have an estimated acceleration of $2.094\frac{rad}{s^2}$. From here, the estimated torque needed to rotate the platform is around

$$\tau = I * \alpha = \mathbf{0.4188Nm.}$$

Since we need a high torque and from our algorithm we would need an accurate step, the stepper motor is preferred. The two stepper motor we looked into are the NEMA 17 and NEMA 23. NEMA 17 has a holding torque of 0.45Nm, and NEMA 23 has a holding torque of 1.26Nm. Even though NEMA 17 seems like it might be enough, in the computation, I neglected the friction which would drastically affect the torque the motor has to supply. Moreover, I also neglected the energy that would be lost through using the internal gear to rotate the platform. Since NEMA 23 is not that much more expensive, we believed NEMA 23 would fit our project best.

For the step driver, we just need one that can provide required current for NEMA 23. We decided to go with DM542T step driver since it could go up to 4.2A which is enough for NEMA 23. Moreover, it is relatively inexpensive and has 1/128 Micro-step Resolutions. This means that we can have more rotations per revolution and could achieve a smaller step angle (much smaller than the required 0.0033 rad if needed).

5 SYSTEM DESCRIPTION

The system consists of an integrated hardware platform, a laser line diode, a digital camera, the rotational platform, and the physical structure to hold up each component. The integrated hardware platform can be further broken down into a motor component (consisting of the software motor controller, the motor driver board, and the stepper motor itself), the algorithmic components (consisting of the GPGPU kernel routines on the embedded system), and the state control component (consisting of processes executing on the 4 core embedded system, determining the control flow of the system). Please see Figure 10: System Specification Diagram in Appendix A to see connections between components.

5.1 Sensor Setup

The sensor setup consists of a statically arranged laser line diode and digital camera. The laser line diode provides a red light at 650nm wavelength with ideally a consistent Gaussian intensity distribution across the line for the entire length of the line. The digital camera must have sufficient resolution to meet our accuracy requirement. We chose an 8 megapixel (the highest commercially available for reasonable price) camera to minimize the effective distance between each pixel and maximize our accuracy.

5.2 Mechanical Setup

The mechanical setup consists of all the components of the system, and determines how they are arranged in physical space with respect to each-other. Figure 5: Mechanical Setup (Front View) and Figure 6: Mechanical Setup (Side View) show how the components would be set up.

Figure 5: Mechanical Setup (Front View)

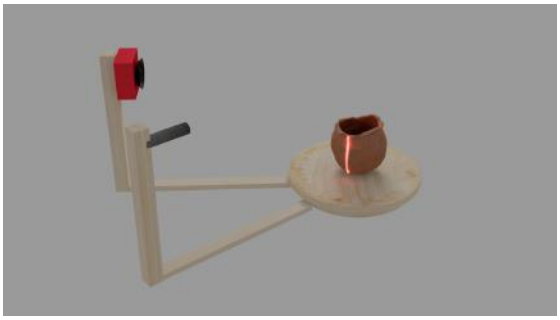

Figure 6: Mechanical Setup (Side View)

The rotational platform forms the physical basis of the device, with the camera and laser line diode positioned at a short distance (approximately 20cm) away pointing towards the center of the rotational platform. The platform would be mainly composed of a base, a motor, a gear on the motor's shaft, a lazy susan bearing to reduce friction, an internal gear, the top platform, and a high-friction surface. The high-friction surface here is to simply help reduce the chance of the object slipping off-center while the platform is rotating. The base here is to give the platform itself enough height so that the motor can be put under. The motor with a gear attached to the shaft will be inside the platform. The lazy susan bearing will be on top of the base, and the internal gear will be attached on top of the lazy susan bearing, and the top platform will be attached on the internal gear. The gear on the motor's shaft will be connected to the internal gear, and when the motor rotates, the platform will rotate with it. Figure 7: Platform Component Breakdown below shows a breakdown of how the platform itself would look like.

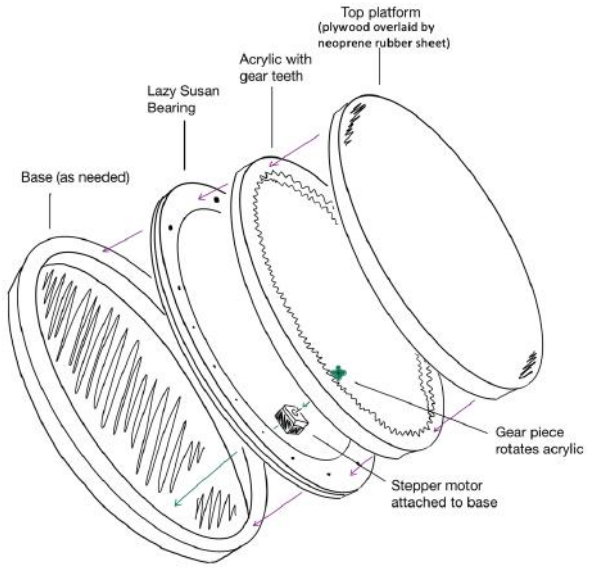

Figure 7: Platform Component Breakdown

From the design trade studies, we are using the following items for the components mentioned above:

- Motor: NEMA 23 Stepper Motor
- Gear on motor's shaft: 3D-printed to control the gear ratio and dimensions
- Lazy Susan Bearing: 10" Swivel Lazy Susan Bearing
- Internal Gear: Laser-cut Acrylic Plexiglass Sheet to control the gear ratio and dimensions
- Top Platform: Plywood 15" Circular Disc
- High-friction Surface: Neoprene Rubber Sheet

5.3 Hardware Setup

The integrated hardware platform consists of the NVIDIA Jetson Nano embedded platform, the DM542T motor driver, and USB/wires/breadboards to provide the interface to our external sensor, motor, and user. The NVIDIA Jetson Nano has the following notable specifications relating to our project:

- 128-core Maxwell GPU
- Quad-core ARM A57 @ 1.43 GHz CPU
- 4 GB 64-bit LPDDR4 25.6 GB/s Memory
- Video Encode/Decode at 30fps for 2x 4k (H.264/H.265)
- 4x USB 3.0 external ports
- GPIO pins for motor control

5.4 State Control Subsystem

A 4-core embedded system with programmable GPGPU will be the main computational platform for our device. Specifically, the NVIDIA Jetson Nano embedded system. This system will have 3 of our dedicated processes running on it while the device in in action:

- User I/O Controller: Enforces protection boundary between user behavior and device logic. Only reasonable commands will be passed from this controller to the other processes, and only intended output will pass from other processes to the user. This indirection layer helps prevent the user from accidentally (or intentionally) tampering with the device logic.
- Core System Controller: This process orchestrates the order of total functionality of the system. A software state machine is implemented to match the behavior of our user story flow chart (See Figure 1: User Story Flowchart). The Core System Controller interfaces with the user via the User I/O Controller (and indirectly USB), with the rotational platform via the Motor Subsystem, with the GPGPU via kernel launches (Algorithmic Subsystem), and with the camera via a USB interface on the NVIDIA Jetson Nano.

- Motor Controller: This process rotates the stepper motor by angles determined and communicated by the core system controller. The motor subsystem description will go deeper into its behavior.

We chose to divide the computation work across three independent cores to maximize performance. The User I/O Controller rarely runs, but it should respond quickly to user inputs, and therefore it is left idle so that we can maximize the response time between an I/O device interrupt (new data from USB) and processing the I/O. The motor controller process on the other hand is real-time deadline driven, as it must continuously step the motor at specific angles to coordinate with the camera via the core system controller. Because of this, we plan on using a soft real-time scheduling algorithm for this process. We will also ask the kernel to separate these processes across the cores. The remaining core can be used for background processes for the OS.

5.5 Motor Subsystem

The motor subsystem is responsible for providing accurate control of the stepper motor turning the rotating platform during 3D scanning. This subsystem consists of both hardware and software components. The stepper motor itself accepts PWM wave modulation to control how many steps to take in which direction (or partial steps). Since such signals are difficult to generate manually, we are using a motor driver board DM542T to convert GPIO signals from the Nvidia Jetson Nano into waveform commands for the stepper motor. A breadboard with wires attached to the pins on both the driver and the stepper motor will transfer this signal to the motor. As noted before, the motor subsystem will have heavy communication with the state control subsystem to coordinate camera image captures. Both the camera latency and throughput, as well as the step amount and speed, have an effect on the details of this coordination.

5.6 Algorithmic Subsystem

The algorithmic subsystem is completely in software, and is mostly data-driven. Wheras the state control subsystem determines what happens across the system at any point in time by using the CPU of the embedded system with I/O and interrupts, the algorithmic subsystem utilizes the GPGPU to massively parallelize data-driven algorithmic code, such as image processing and geometric transformations. The algorithmic subsystem generally consists of GPGPU kernels which implement the required computations of our device, including calibration, computer vision, point-cloud generation, pairwise registration, and mesh triangulation, as well as accuracy computation for testing and validation. Algorithms and descriptions of their implementations are described in detail in the remaining subsections of the system description.

5.7 Image Laser Detection

This component of the algorithmic subsystem is responsible for detecting points on the laser line in a camera image. The basic idea is assume a model of the distribution of the laser line across its width as being a Gaussian distribution, such as in Figure 8: Laser Line Light Intensity Distribution.

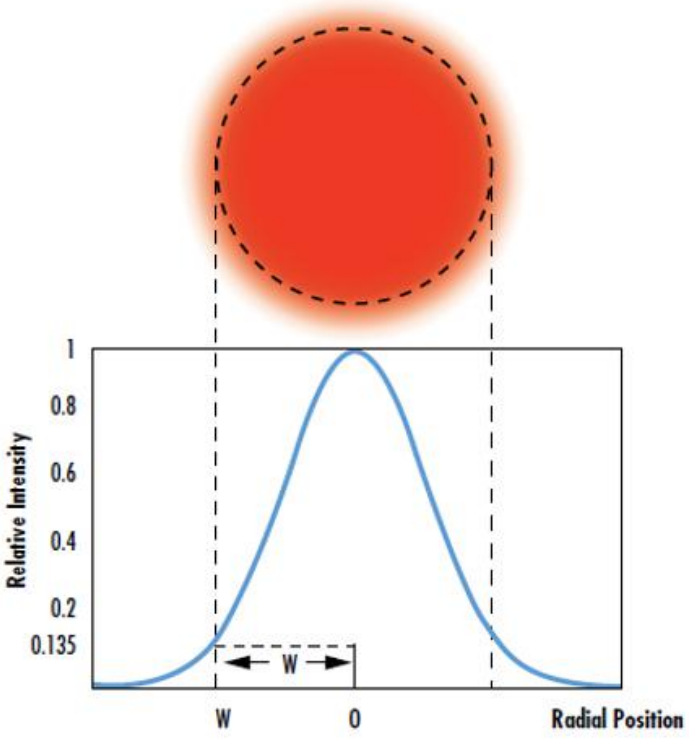

Figure 8: Laser Line Light Intensity Distribution

First we apply an intensity filter to extract higher values at pixels with color close to the wavelength of the laser light (650nm). Then we apply a horizontal Gaussian filter to enhance the Gaussian distribution of the laser intensity (since the laser line will be close to vertical in the camera image). For each row, the center of this Gaussian distribution is the horizontal location of the laser. Note that this suffers from the problem that only one point can be found for each row of the image. However, the case that a single row of the image has multiple parts of the laser line is a case of occlusion already, and these holes in the scan can be resolved by merging the results with an additional scan (single object multiple scans).

5.8 Camera and Scene Calibration

There are several components to calibration which must be performed prior to any 3D scanning. The general method to convert between a pixel and a world space coordinate is, for pixel u and coordinate p:

$$\lambda u = K(Rp + T)$$

where :

- λ is a scalar intrinsic to the camera and screen size in pixels
- K is a distortion matrix intrinsic to the camera lens
- R is the rotation matrix between world space and camera space coordinates.
- T is the translation matrix between world space and camera space coordinates.

Then to calibrate each of these parameters:

- Intrinsic Camera Calibration: this calibration resolves constants related to the camera lens and polynomial distortion that may have on the image verses real world space. This can be done by the mapping between known world-space points and known pixels. This calibration determines the K and λ parameters above.
- Extrinsic Camera Calibration: this calibration solves a system of linear equations to find the translation and rotation matrices for the transformation between camera space and world space. This system is made non-singular by having sufficiently many known mappings between camera space and world space (specific identifiable points on the turntable). Therefore, this requires an object on the turntable to act as these known points. We will use a CALTag checkerboard pattern pasted on the turntable surface to help perform this calibration. The CALTag pattern is especially useful since position can be determined even if some of the image is occluded. This calibration determines the R and T parameters above.

  We setup the linear equation as follows. Suppose

  $$u = \begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix}, \qquad p = \begin{bmatrix} p_x \\ p_y \\ 0 \end{bmatrix}$$

  Where u is in homogeneous pixel coordinates and p is on the turntable plane ($z = 0$), and

  $$\tilde{u} = K^{-1}u, \qquad R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \qquad T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}.$$

  From this we get that

  $$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \tilde{u} \times (\lambda\tilde{u}) = \tilde{u} \times (Rp + T).$$

  Then we solve for R and T from the following system:

  $$\tilde{u} \times \begin{bmatrix} r_{11}p_x + r_{12}p_y + T_x \\ r_{21}p_x + r_{22}p_y + T_y \\ r_{31}p_x + r_{32}p_y + T_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

  From this we group the unknowns in a single vector

  $$X = \begin{bmatrix} r_{11} & r_{21} & r_{31} & r_{12} & r_{22} & r_{32} & T_x & T_y & T_z \end{bmatrix}^T$$

  from which we can derive that

  $$\begin{bmatrix} 0 & -p_x & \tilde{(u)}_y p_x & 0 & -p_y & \tilde{(u)}_y p_y & 0 & -1 & \tilde{(u)}_y \\ p_x & 0 & -\tilde{(u)}_x p_x & p_y & 0 & -\tilde{(u)}_x p_y & 1 & 0 & -\tilde{(u)}_x \end{bmatrix} X = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

  Here we are only considering a single point p. During calibration, we take multiple images, and so we actually have a set of points $\{p_i | i : 1...n\}$. We can use all such points to form a single system of the form $AX = 0$ where $A \in \mathbb{R}^{2n \times 9}$. Since vector X has 8 degrees of freedom, matrix A must have rank 8, meaning we need at least 4 non-collinear points. Finally we are left to solve the following optimization problem to compute R and T:

  $$\hat{X} = \underset{X}{\operatorname{argmin}} \, ||AX||, \qquad s.t. ||X|| = 1$$

- Axis of Rotation Calibration: this is computed in a similar manner to extrinsic camera calibration. The center position of the turntable is discovered, and the axis of rotation is assumed to be in the positive z direction. Note that this assumes that the 'up' direction of our camera image is the same angle as the up direction of the turntable.
- Plane of Laser Line Calibration: With laser line detection, points along the laser in the digital image can be identified. From this, a linear equation must be solved to determine the A, B, C, and D parameters of the plane of the laser line in world space, where the plane is $Ax + By + Cz + D = 0$. An additional calibration object is required here to completely define the laser plane, instead of simply defining a pencil of planes (all planes rotated around the axis of the turntable). The known object, which will be a checkerboard pattern, allows us to know where on the object the laser points are detected, and from that collect a set of non-collinear points, which allow us to solve the linear system.

5.9 Global Parameter Optimization

Calibration parameters will be solved for in the least-squared error sense, to match our recorded points. Global optimization will then be applied to all the parameters to reduce reconstruction error. Our implementation of optimization will consist of linear regression.

5.10 Point Cloud Generation

With all of the parameters calibrated, to generate the 3D point cloud of a scan we simply perform ray-plane intersection between the ray originating at the world space coordinate of the pixel with the laser line, whose direction is away from the camera world position and towards the laser plane in world space. This intersection point is in world space coordinates, so it must be rotated around the rotational axis by the reverse angle of the stepper motor rotation to get the corresponding point in object space.

All such points are computed and aggregated together to form the point cloud for a scan. See Figure 9: Ray-Plane Intersection for intuition.

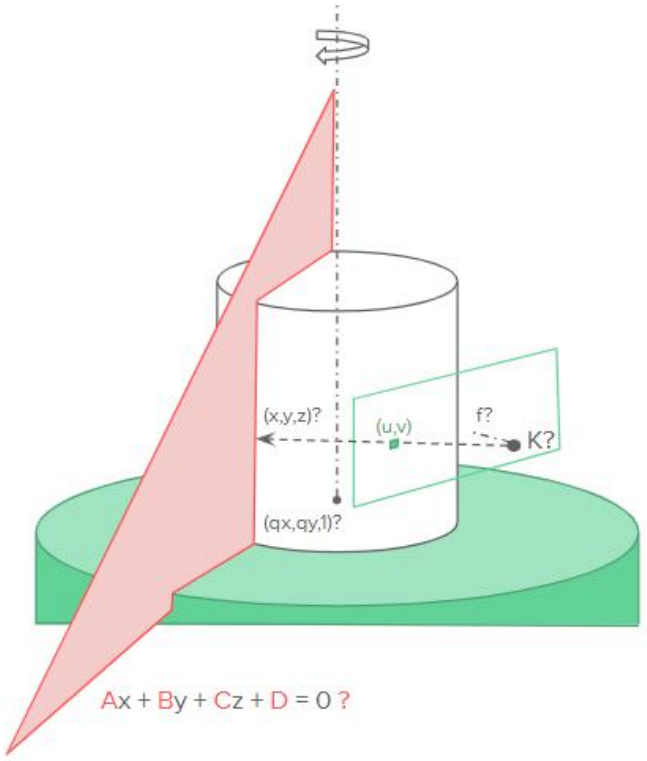

Figure 9: Ray-Plane Intersection

5.11 Point Cloud Processing

There are several components to processing the point cloud obtained after mapping the sensor data into world coordinates to generate the point cloud.

a. Outlier removal/noise reduction

We will first do outlier removal, and may or may not do noise reduction depending how bad the point cloud obtained is - we do not want to over-smooth the object since many archaeological objects tend to have weird shapes and contours and jagged edges. Point clouds obtained from 3D scanners with any methods, including laser projection, would regularly be contaminated with some level of noise and outliers. The first step for dealing with raw point cloud data obtained after conversion from laser projection to depth would be to discard outlier samples. Note that this step would come after removing the points in the background and the foreground points on the turntable. There are generally three types of outliers: sparse, isolated, and nonisolated. Sparse outliers have low local point density that are obvious erroneous measurements i.e. points that float outside of the rest of the data. Isolated outliers have high local point density but are separated from the rest of the point cloud, i.e. outlier clumps. Nonisolated outliers are the most tricky – they are outliers that are very close to the main point cloud and cannot be easily separated, akin to noise. We will focus on sparse and isolated outliers, and we can use a method that looks at average distance to k-nearest neighbors, then removes that point based on a threshold defined in practice. Let the average distance of point p_i be defined as:

$$d_i = 1/k \cdot \sum_{j=1}^{k} \text{dist}(p_i, q_j) \tag{1}$$

where q_j is the jth nearest neighbor to point p_i. Then, the local density function of p_i is defined as follows:

$$LD(p_i) = \frac{1}{k} \cdot \sum_{q_j \in kNN(p_i)} \exp\left(\frac{-\text{dist}(p_i, q_j)}{d_i}\right) \tag{2}$$

with d_i defined earlier in equation (1). Now we can define the probability that a point belongs to an outlier as:

$$P_{\text{outlier}}(p_i) = 1\text{–}LD(p_i)$$

We can then take this probability and if it is above a certain threshold, that point will be removed from the point cloud data (PCD). One reference paper uses $P_{\text{outlier}}(p_i) > 0.1 \cdot d_i$ as their threshold in practice which is dynamic based on d_i, we can determine this threshold through empirical testing by seeing accuracy numbers based on threshold values on multiple test samples.

b. Iterative Closest Point Algorithm (ICP)

We will use an ICP (Iterative Closest Point) algorithm to combine different scans. This is needed in the use case where the user wants to combine a scan from another angle since some part of the object was occluded in the original scan. The ICP algorithm determines the transformation between two point clouds from different angles of the object by using least squares to match duplicate points – these point clouds would be constructed by mapping the scanned pixel and depth to their corresponding 3D cartesian coordinates. Similar to gradient descent, ICP works best when starting at a good starting point to avoid being stuck at local minima and also save computation time.

c. Triangulation from Point Cloud to Mesh

The point cloud would likely be stored in a PCD (Point Cloud Data) file format. PCD files provide more flexibility and speed than other formats like STL/OBJ/PLY, and we can use the PCL (Point Cloud Library) in C++ to process this point cloud format as fast as possible (this library should be able to be run on the Jetson). The PCL library provides many useful functions such as estimating normals and performing triangulation (constructing a triangle mesh from XYZ points and normals). We think that implementing triangulation ourselves completely from scratch will be

out of scope for this project and also unnecessary, unless the PCL implementation is grossly inefficient. Since our data will just be a list of XYZ coordinates, we can easily convert this to the PCD format to be used with the PCL library (the PCD format is a list of XYZ coordinates with a few header lines in the beginning).

The triangulation algorithm works by maintaining a fringe list of points from which the mesh can be grown and slowly extending the mesh out until it covers all the points. There are many tunable parameters such as size of the neighborhood for searching points to connect, maximum edge length of the triangles, as well as maximum allowed difference between normals to connect that point, which helps deal with sharp edges and outlier points. The flow of the triangulation algorithm involves estimating the normals, combining that data with the XYZ point data, initializing the search tree and other objects, then using PCL's reconstruction method to obtain the triangle mesh object. The algorithm will output a `PolygonMesh` object which can be saved as an OBJ file using PCL, which is a common format for 3d printing (and tends to perform better than STL format).

6 PROJECT MANAGEMENT

6.1 Schedule

Our schedule has slightly changed since we decided to use the laser stripe triangulation sensor setup. Please refer to Figure 11: Gantt Chart (updated 03/02/2020) in Appendix A for the full Gantt Chart. Our original schedule was built around using the depth camera, but now the Gantt Chart is structured around using the laser stripe sensor.

We will aim to start writing some of the testing code and filtering code while we wait for parts, and assemble the platform and the whole setup either before or right after spring break depending when the parts arrive. Then, we will be able to write code to process the sensor data and convert it to point clouds, as well as code for triangulation and pairwise registration. We aim to be able to make a decent reconstruction by late March for the lab demo, and optimize our code and make necessary fixes for the final presentation in April.

6.2 Team Member Responsibilities

Our team divides work among each team member equally. The team deals with logistics, integration, and decision-making together but each member still has his main tasks assigned as follows:

Jeremy:

- Testing database construction
- Outlier removal and noise reduction
- Point cloud triangulation
- ICP for combining multiple scans

Chakara:

- Rotational mechanism
- Motor controller/driver
- Platform construction
- Optimization of software components for GPU

Alex:

- Laser stripe detection and mapping to world coordinates
- Point cloud generation from sensor data
- Camera calibration code
- Testing benchmark code

6.3 Risk Management

Our design still consists of some potential risks. We have several plans to mitigate these risks, which would require more implementation complexity on our part.

- **Part of the input object is obscured:**
  We plan on dealing with this issue by merging multiple scans of multiple angles of the objects to get more perspectives. We will be using pairwise registration to merge the point clouds generated from these scans.
- **Vibrational noise:**
  We can add an additional laser stripe for triangulation to correct for the noise. We can also modify the sensor setup to be disjoint from the platform if there is too much vibration induced from the stepper motor mechanism.
- **The stepper motor angle data is inaccurate:**
  There is a very low possibility this would happen but if it does, we can combine the computed angle from the motor with computer vision to get a more accurate angle. However, this may require placing some sort of calibration mat underneath the object to be able to track the angle rotated throughout the process.
- **The laser stripe doesn't have enough intensity:**
  We can buy a stronger laser since we have a buffer in our budget that amounts to around $200.
- **Potential holes in the point cloud:**
  We can merge multiple scans of the object, again using pairwise registration. We can also perform hole-filling mesh triangulation.

Item	Price ($)	Total Price Includes Shipping ($)	Description
Nema 23 Stepper Motor	23.99	23.99	Motor for rotational mechanism
15 Inch Wooden Circle by Woodpeckers	14.99	14.99	Plywood for platform
Lazy Susan Bearing	27.19	27.19	Reduce friction in rotational mechanism
Acrylic Plexiglass Sheet, Clear	9.98	9.98	To create internal gear
DM542T Step Driver	38.99	38.99	Step Driver for NEMA 23
Neoprene Rubber Sheet Rolls	14.80	14.80	To add friction to the platform
Adafruit Line Laser Diode (1057)	8.95	17.94	Projected Laser Stripe
Webcamera usb 8MP 5-50mm Varifocal Lens	76.00	76.00	Camera
Nvidia Jetson Nano Developer Kit	99.00	117.00	Embedded Systems
256GB EVO Select Memory Card	42.96	42.96	MicroSD card for NVIDIA Jetson
MicroUSB Cable (1995)	9.00	17.99	MicroUSB cable for NVIDIA Jetson

Table 1: Project Components (Purchase)

- **NVIDIA Jetson Nano and motor driver integration:**
  Although using NVIDIA Jetson Nano to control the motor step driver is possible, there is a possibility that it would not work since we have not found an article supporting the use of NVIDIA Jetson to control the DM542T motor driver model. If they can't integrate, then we plan on borrowing an Arduino from the Capstone course to control the motor driver instead.

6.4 Budget

Table 1: Project Components (Purchase) shows the project components that would be purchased. Note that some of the item names are shortened for readability.

This comes up with a total price of **$401.83**. We would have $198.17 left in our budget which would still need to be used to purchase wood for supporting our platform and for risk management.

We could potentially borrow an Arduino from the Capstone course in case our NVIDIA Jetson can't integrate with our motor step driver (refer to the risk management section). We could also potentially reuse one of our breadboards from previous classes if we need one to conenct the motor with the driver.

References

[1] Yan Cui et al. "3D shape scanning with a time-of-flight camera". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE. 2010, pp. 1173–1180.

[2] Steven M Seitz et al. "A comparison and evaluation of multi-view stereo reconstruction algorithms". In: *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*. Vol. 1. IEEE. 2006, pp. 519–528.

[3] Gabriel Taubin, Daniel Moreno, and Douglas Lanman. "3d scanning for personal 3d printing: build your own desktop 3d scanner". In: *ACM SIGGRAPH 2014 Studio*. 2014, pp. 1–66.

[4] Klaus Thoeni et al. "A Comparison of Multi-view 3D Reconstruction of a Rock Wall using Several Cameras and a Laser Scanner". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-5 (June 2014), pp. 573–580. DOI: 10.5194/isprsarchives-XL-5-573-2014.

[5] DroneBot Workshop. *Stepper Motors with Arduino – Bipolar & Unipolar*. https://dronebotworkshop.com/stepper-motors-with-arduino/. Feb. 2018.

[6] Michael Zollhöfer et al. "State of the Art on 3D Reconstruction with RGB-D Cameras". In: *Computer graphics forum*. Vol. 37. 2. Wiley Online Library. 2018, pp. 625–652.

Acknowledgement

We would like to thank the ECE department for providing the opportunity for us to work on this project. We would like to thank Professor Tamal Mukherjee, Professor Bill Nace, and Joe Zhao for giving us great feedback and advice and helping us mitigating our risks in our design decisions. We would also like to thank Maxwell Van Buskirk, Kevin Riordan, and Ian Suzuki for helping with software to choose materials and suggestions on the platform design.

Appendix A

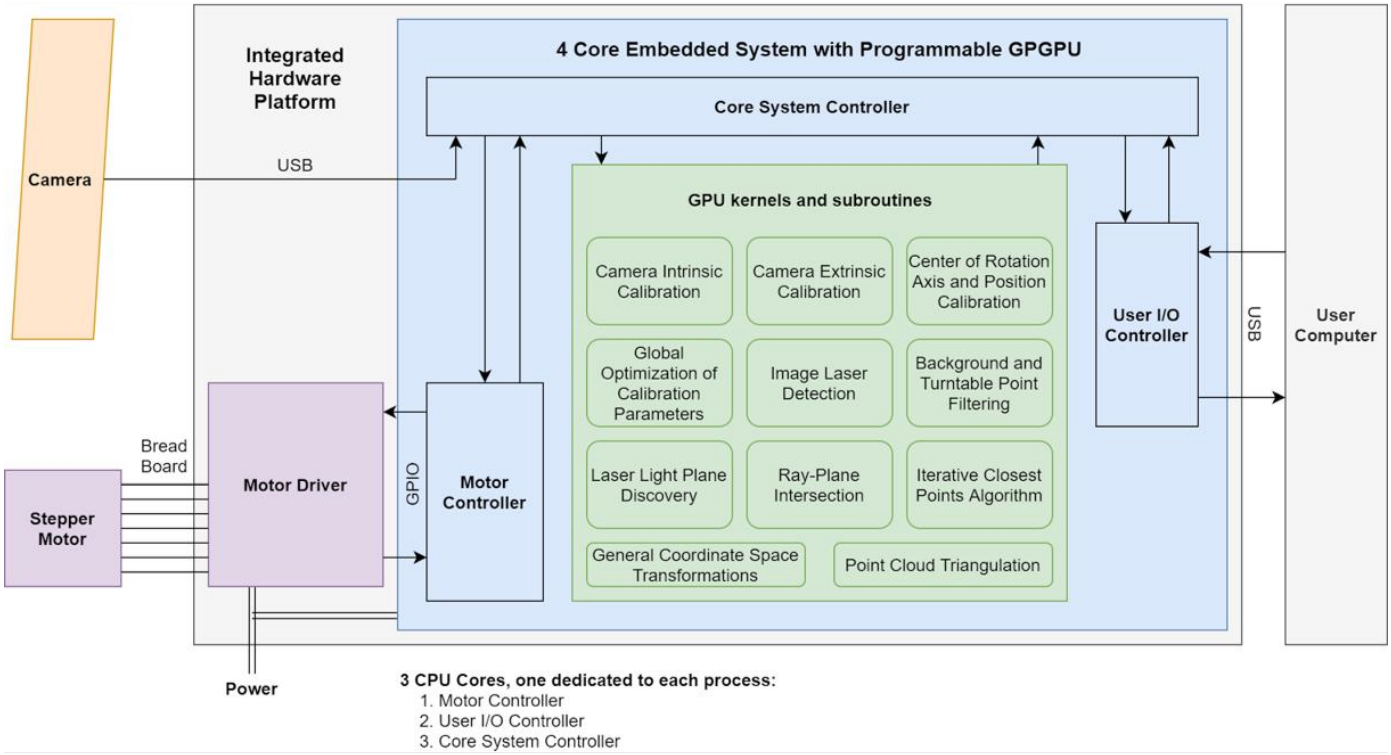

Figure 10: System Specification Diagram

Material	**Young's Modulus** (GPa)	**Specific Stiffness** $(\frac{MN.m}{kg})$	**Yield Strength** (MPa)	**Density** $(\frac{kg}{m^3})$	**Price per Unit Volume** $(\frac{\$}{m^3})$
Plywood	3-4.5	3.98-6.06	8.1-9.9	700-800	385-488
Carbon Fiber	58-64	38.2-42.4	533-774	1490-1540	26200-31400
Balsa	0.23-0.28	0.817-1.09	0.8-1.5	240-300	1610-3230

Table 2: Material Tradeoffs Analysis

Shape	**Dimensions** (m)	**Formula**	**Moments of Inertia** (kgm^2)
Cube	0.3*0.3*0.3	$\frac{1}{12}m(w^2+d^2)$	0.105
Solid Sphere	r = 0.3	$\frac{2}{5}mr^2$	0.063
Hollow Sphere	r = 0.3	$\frac{2}{3}mr^2$	0.105
Solid Cone	r = 0.3	$\frac{3}{10}mr^2$	0.04725
Hollow Cone	r = 0.3	$\frac{1}{2}mr^2$	0.07875
Solid Cylinder	r = 0.3	$\frac{1}{2}mr^2$	0.07875
Hollow Cylinder	r = 0.3	mr^2	0.105
Rod	L = 0.3	$\frac{1}{2}mL^2$	0.0525

Table 3: Input Object Moments of Inertia

Task Name	Start Date	End Date	Team Member	Status
Design				
Research algorithms	2/5	2/20	AP	Complete
Determine model components for algorithm	2/8	2/17	ALL	Complete
Design scanning platform layout	2/9	2/19	JL/CO	Complete
Design rotational mechanism	2/12	2/19	CO	Complete
Determine sensor setup	2/12	2/19	JL/CO	Complete
Determine risk factors and evaluation methods	2/15	2/19	ALL	Complete
Find specific parts for purchase and determine total cost	2/17	2/20	JL/CO	Complete
Create (and present) Design Review presentation	2/19	2/21	AP	Complete
Slack before Design Presentation	2/20	2/23	ALL	Complete
Design Document	2/26	3/1	ALL	Active
Implementation				
Order components	2/26	2/29	CO	Complete
Find/generate database of ground-truth models we can test on	2/26	3/3	JL	Active
Write testing benchmark code to determine accuracy of a generated model	2/26	3/5	AP	Active
Implement filtering of sensor data	3/5	3/16	JL	Upcoming
Assemble scanning capsule	3/5	3/16	CO	Upcoming
Wire up motor control circuit	3/5	3/16	CO	Upcoming
Implement rotational controller	3/5	3/20	CO	Upcoming
Spring break	3/7	3/14	ALL	Upcoming
Assemble the whole platform setup	3/16	3/20	ALL	Upcoming
Implement algorithm to construct global point cloud from sensor data	3/16	3/23	AP/JL	Upcoming
Early testing and analysis of sensor data	3/17	3/22	CO	Upcoming
Implement triangulation to form output mesh	3/22	3/27	AP/JL	Upcoming
Test and verify across many samples	3/25	3/27	ALL	Upcoming
Prepare an example reconstruction for in-lab demo	3/25	3/29	ALL	Upcoming
Make necessary fixes after in lab demo	3/30	4/10	ALL	Upcoming
Spring carnival	4/16	4/19	ALL	Upcoming
Create (and present) Final Presentation	4/20	4/24	ALL	Upcoming
Write final report	4/26	4/29	ALL	Upcoming
Slack before Final Report	4/29	5/3	ALL	Upcoming
Overall Progress				

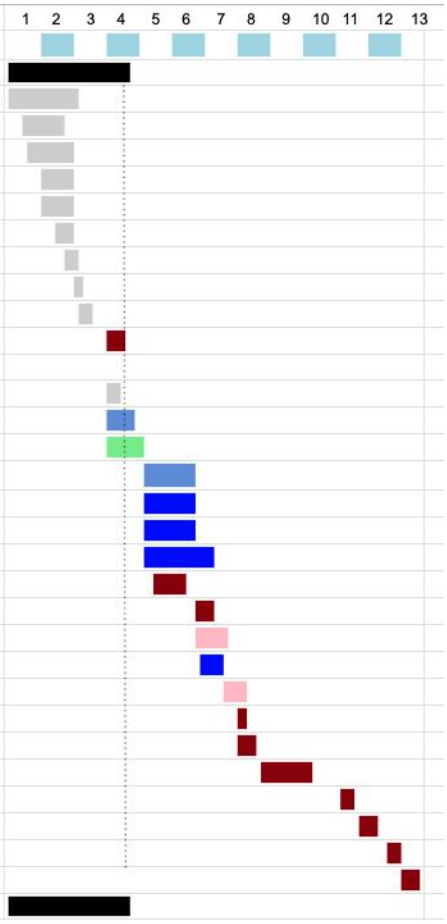

Figure 11: Gantt Chart (Updated 03/02/2020)