Spring 2023

ND EE Senior Design: Final Report

LiDAR Object Mapping

Zach Jaromin, Jack Rourke, Garrett Schaefer, Robert Xu

UNIVERSITY OF NOTRE DAME

Table of Contents

1 Introduction

Inspiration:

Originally, the team's goal for senior design was to create an autonomous robot capable of mapping the floor of aquatic bodies. However, this project proved to be untenable. The degree to which waterproofing would be needed was a significant factor that persuaded group members to pursue another project. Additionally, the cost of an underwater sonar sensor proved to be prohibitively expensive.

However, the team was inspired by the idea of mapping the surface of an object. This development caused the team to focus the project on scanning small objects. Using LiDAR sensors, the team began to develop a plan to create a point cloud of an object which could then be mapped into an STL file.

The problem being solved:

Corporations spend millions of dollars designing products and then ensuring that manufactured results are up to the original design specifications. Using CAD software, nearly any part or product can be digitally modeled. However, this can often be a time consuming process and there is no guarantee that the engineer doing the modeling will complete his task perfectly. If even one dimension is off, the model will need to be updated, and any parts already sent to the printer will need to be scrapped. Thus, if there is already a prototype of a part and a company or individual would simply like to replicate it in CAD, they must undergo a time-consuming and error-prone process to model it in software. Additionally, there are currently incredibly costly systems in place to ensure parts meet the standards that they were designed with. Having an individual take measurements or employing a computer vision system can be unreliable and incredibly costly. If a cost-effective, simple solution to this problem could be implemented, it could increase the efficiency and accuracy of designing and manufacturing all manner of products.

High level description of our solution:

In order to solve this problem, a LiDAR scanner was created. This scanner employs a LiDAR array that continuously takes distance measurements. Additionally, this array moves to varying heights to record the full dimensions of a part. As the LiDAR array ascends, the part is placed on a 360-degree-rotating tray. Because the part is rotating and the LiDAR array is moving to variable heights, measurements can be taken at every point around a part. This data is then exported to an SD card, from which an STL file is generated.

Expectations and Results:

Our design exceeded our expectations, particularly when we selected the high resolution scan option. The highest resolution sensor gives very precise and accurate readings, while the other two sensors give somewhat unsatisfactory readings for the medium and low resolution scans. The lower resolution sensor's lack of accuracy proved to be a significant impediment to the quality of scans. These sensors were picked for their precision, each having a resolution greater than or equal to 1 mm. However, the accuracy of these sensors was not 1 mm, being closer to or greater than 5 mm. Thus, they were precise but not accurate, giving our scans lots of variance and making it difficult to create a consistent point cloud. Because of this, these sensors were not capable of producing effective scans in the way the team had originally hoped. While serviceable, these sensors did fail to live up to the expectations the group had for them.

The high quality sensor surpassed our expectation for the quality of scan it was able to generate. Because of its consistency in generating accurate, repeatable measurements, it was able to create a high-quality point cloud. These scans proved to be very reliable, repeatable, and high quality. The team is very pleased with the outcome of the scans from the high-resolution sensor; the only downside to this sensor is the slow scan rate (~2 Hz), leading to long scan times.

2 Detailed System Requirements

2.1 Precise Stepper Motor Control

- To create an accurate scan, the motor's position has to be accurate. Without this, the position the code thinks it is at will not be where it actually is, leading to error.
- Accurate stepper motor control is defined at 1.8-degree steps.

2.2 Functional LiDAR Sensors

- The sensors need to be able to return reliable measurements to create an accurate model of the objects.
- A reliable measurement is one that is within 1 mm of the actual distance value.

2.3 Stable Frame

- The frame needs to be large enough to have room for the sensor towers, object platform, and power supply cover.
- The towers have to be stable enough to not shake while the sensor is scanning an object.

2.4 STL File Creation

- Convert the data points from the LiDAR sensors into a 3D point cloud, which then gets rendered into a mesh.
- This mesh file is the STL file that can be 3D printed. The STL file will have a sufficient number of points to be 3D printed .

2.5 Data storage on a micro SD Card

- The micro SD card needs to be able to be written to.
- It also must have enough storage to ensure an entire scan can fit on it.

2.6 Wireless Communication

- The wireless portion of this project is required to start the scanning process.
- The user must be able to interact with the webpage, which has to communicate with the ESP32.
- The user will have the option to select a high, medium, or low quality of scan.

2.7 Overall Requirements

- The user will be able to receive an accurate scan of their object
- The time it takes to scan an object will be determined by the height of the object. However, a low-quality scan for a 5-inch-tall object should not take more than one hour, with a high-quality scan not taking more than ten.
- The user will have the option to select a high, medium, or low quality of scan.

3 Detailed project description

3.1 System theory of operation

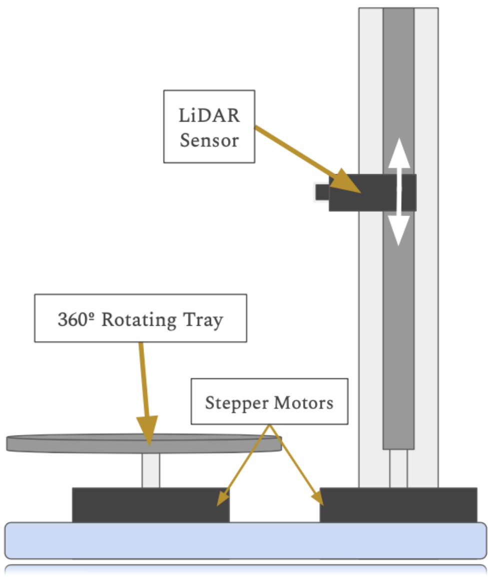


Figure 1: Initial Design of Our Project

The theory behind how the system works is that an object is placed on the central platform to be scanned. The user selects a quality of scan, which corresponds to a certain sensor. The platform begins to rotate. As it does, the selected sensor scans the object. This measurement, along with the rotational counter and height counter, is written to the micro SD card. The program rotates the platform 200 times, which corresponds to 360 degrees. Then, the tower the sensor is on gets raised up. This process is repeated until the sensor is fully above the object being scanned.

The program recognizes that the object has been fully scanned by checking what distance is being returned by the sensor. If more than halfway through a full rotation (100 steps of the platform) and all of the measurements recorded are greater than the defined max distance of an object, then the program stops scanning and resets the tower to the starting height. The max distance is defined roughly as the distance between the sensor and the tower across from it, or for the analog sensor, as returning 0 V.

The next step in the process is to remove the micro SD card from the board and plug it into a computer that has the STL file creation software on it. This software filters out any of the max points, saving them as no point being there. Also, the program linearly interpolates between horizontal scan planes by using upper and lower bounds to create two additional subplanes. As a result, we have more points to work with when generating a surface mesh. The mesh creation process is straightforward. All points, based on a neighbor estimation algorithm, are triangulated. This mesh is then exported as an STL file on the user's personal computer.

3.2 System Block diagram

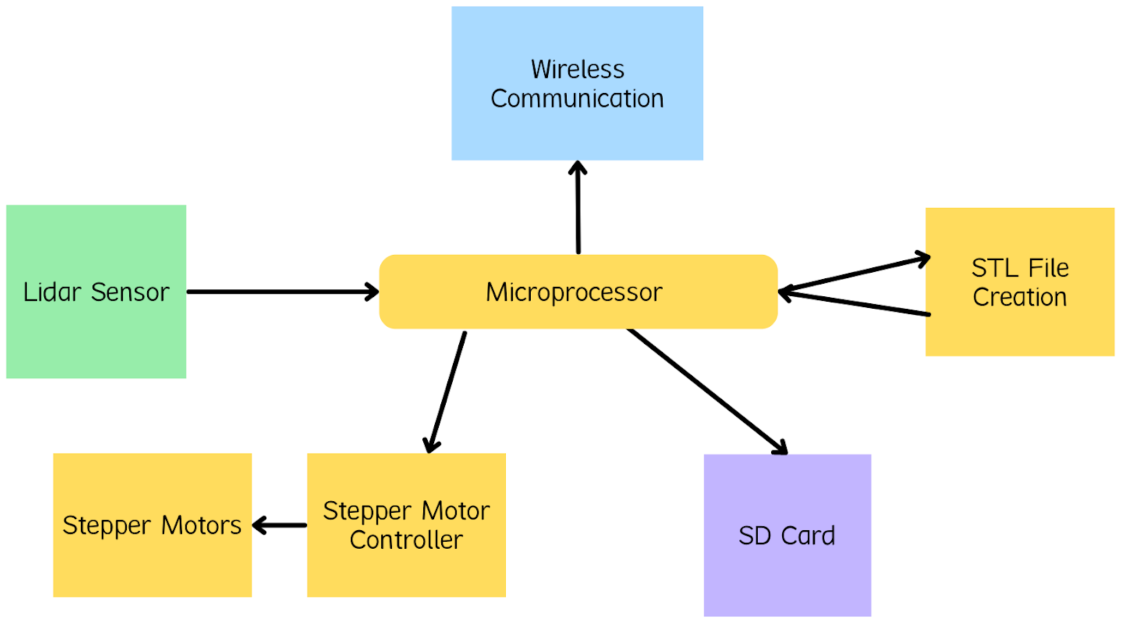


Figure 2: System Block Diagram

Overall system block diagram showing how it is divided into subsystems, and the interfaces between the subsystems

3.3 Subsystem 1: LiDAR Sensors

- Sensor 1: JRT U8XX 1mm Resolution Sensor

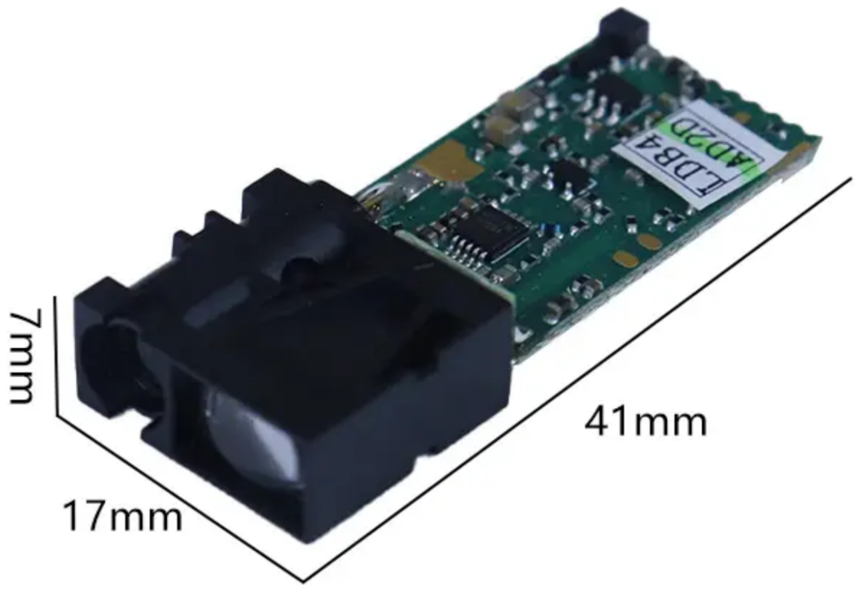


Figure 3: JRT Lidar Sensor

The JRT sensor communicates with the ESP32 via a UART interface. There are multiple UART interfaces within the ESP32, but we used the UART 2 (pins 16 and 17) and UART 0 (pins 1 and 3). These are TX and RX pins, and the TX pin of the ESP32 is connected to the RX pin of the sensor. The TX pin of the sensor is then connected to the RX pin of the ESP32.

Selection Criteria:

This sensor was selected because it had the highest resolution we could find within our budget. Coming in at just under $50 with a 1 mm resolution and accuracy was by far the best available combination we found. The sensor was chosen for its high resolution and accuracy; however, because of its slow sampling rate (.4 - 4 Hz), we knew we would need to supplement it in order to take measurements more quickly.

We approached this problem in two ways. First, we purchased 2 JRT sensors so that we could station 2 sensors 180 degrees apart from each other. Unfortunately, one of them failed to work, so we had to come up with another sensor, discussed in the next section. The 180 degree separation cuts the amount we need to spin the turntable in half, which halves the time a scan will take. The second solution to the slow sampling rate was to include another sensor: the Sparkfun Vl53 ToF sensor.

Powering Scheme:

The JRT sensor requires 3.3 V to the VDD pin and 300 mA of current. Because this exceeds the limits of the ESP32, we used the power supply we purchased to power the device (Power Supply).

The power supply can supply 5 V, 12 V, and 24 V and 8+ A of current. We placed a voltage regulator on our board to step down the 5 V from the power supply to 3.3 V.

The JRT sensor also requires a GPIO pin to provide 3.3 V to an enable pin and a reset pin. This can easily be accomplished using any ESP32.

Subsystem Testing:

The JRT Sensor was tested using a breakout Dev board. The goal of the test was to get the sensor to print out distance measurements to the serial monitor. This way we could ensure the values were updating correctly and verify that they were reasonably accurate with a ruler.

- Sensor 2: Sparkfun Vl53 ToF sensor

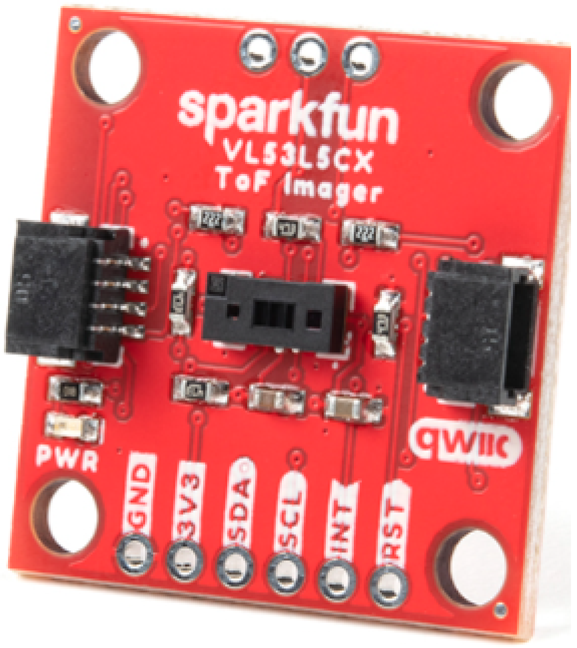


Figure 4: Sparkfun VI53 Sensor

As seen in the schematic above, the connection guide for the Sparkfun sensor is very simple. There are only four connections. Power is connected to 3.3 V via the ESP32. The ESP32 is able to provide ample current during all operations for the sensor as it only requires 50 mA. The ground connection on the sensor goes to a common ground. The SCL and SDA pins go to the respective pins on the ESP32 (22,21). The sensor communicates to the ESP32 through an I2C interface.

Selection Criteria:

This sensor was selected due to its affordable price ($25), high resolution (1 mm resolution with +-5 mm accuracy), and its high sampling rate (50 Hz). This sensor is intended to be used for a fast scan, which is why we sacrificed accuracy for the higher sampling rate. Additionally, the sensor still provides a very good distance measurement; it is just not nearly as consistent as the JRT sensor. Because of this, they complement each other well, which is why both are included in our design. Another important factor in the selection was the convenient power requirements of this sensor.

Powering Scheme:

The Sparkfun sensor requires 3.3 V to the VDD pin and 50 mA of current. This is within the limits of the ESP32, so no external power supply is necessary for this sensor. The 3.3 V comes from the VDD pin off the ESP32. The ground on the sensor is connected to ground on the ESP32.

Subsystem Testing: .

The Sparkfun Sensor was tested using a breakout Dev board. The goal of the test was to get the sensor to print out distance measurements to the serial monitor. This way we could ensure the values were updating correctly and verify that they were reasonably accurate with a ruler.

Sensor 3: Analog Distance Sensor

Selection Criteria:

The analog sensor was selected as a low-quality sensor for this project. This is because it is incredibly cost effective, coming in at under $15. Additionally, it has a sampling rate of over 50 Hz, making it the fastest scanning sensor included in this project.
Finally, this sensor has the unique property of being analog. This means that in theory it has infinite resolution. Additionally, it only requires one GPIO pin, as it functions by reading in a voltage and mapping that voltage onto a distance. Thus, the easy connectability and "perfect" accuracy made this an ideal sensor for the project.

Powering Scheme:

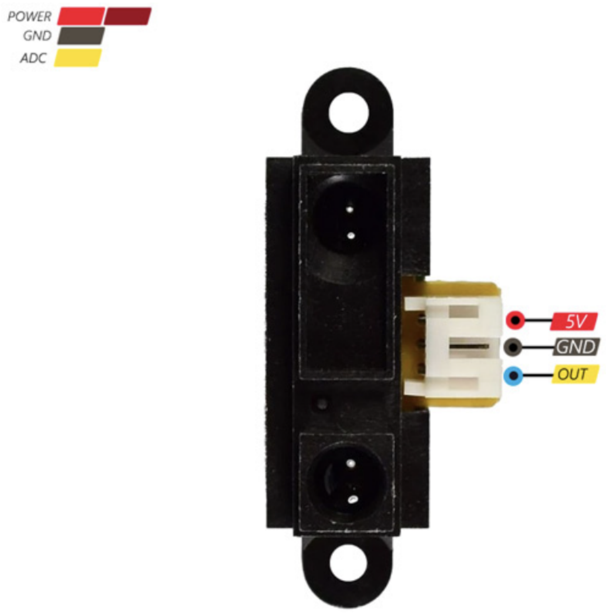


Figure 5: Analog Sensor

The powering scheme for this sensor is extremely basic. Power (5 V) comes directly from the power supply. The ground connection is hooked up to the common ground of the power supply and the ESP32. Finally, the OUT pin is connected to the GPIO pin of the ESP32 and a voltage is read in.

Subsystem Testing:

The Analog Sensor was tested using a breakout Dev board. The goal of the test was to get the sensor to print out distance measurements to the serial monitor. This way we could ensure the values were updating correctly and verify that they were reasonably accurate with a ruler. In order to validate the test results, the voltage being read from the sensor had to be mapped to a corresponding distance. This was executed by employing the figure pasted below.

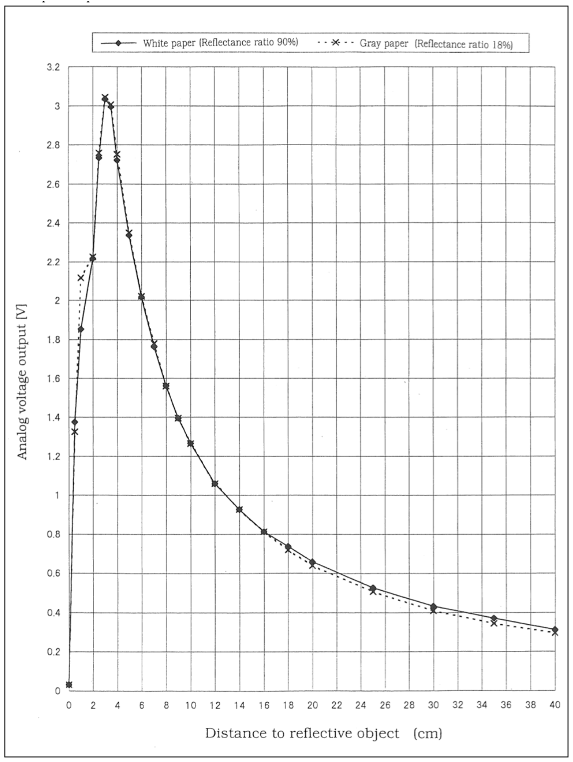


Figure 6: Analog Sensor Conversion Chart

Several different methods of mapping were experimented with. First, a polynomial regression was employed to mimic the shape of the curve. However, the team found that these results proved to be inaccurate and unstable. In lieu of this solution, the group opted to use a linear regression between sets of data points. This provided relatively stable data that could be used in a scan. However, it is necessary to note that this sensor heavily struggles with providing consistent data. 10-20% jumps in distance values are commonplace. Because of this, a filter was implemented to help maintain continuity in the data.

3.4 Subsystem 2: Stepper Motors

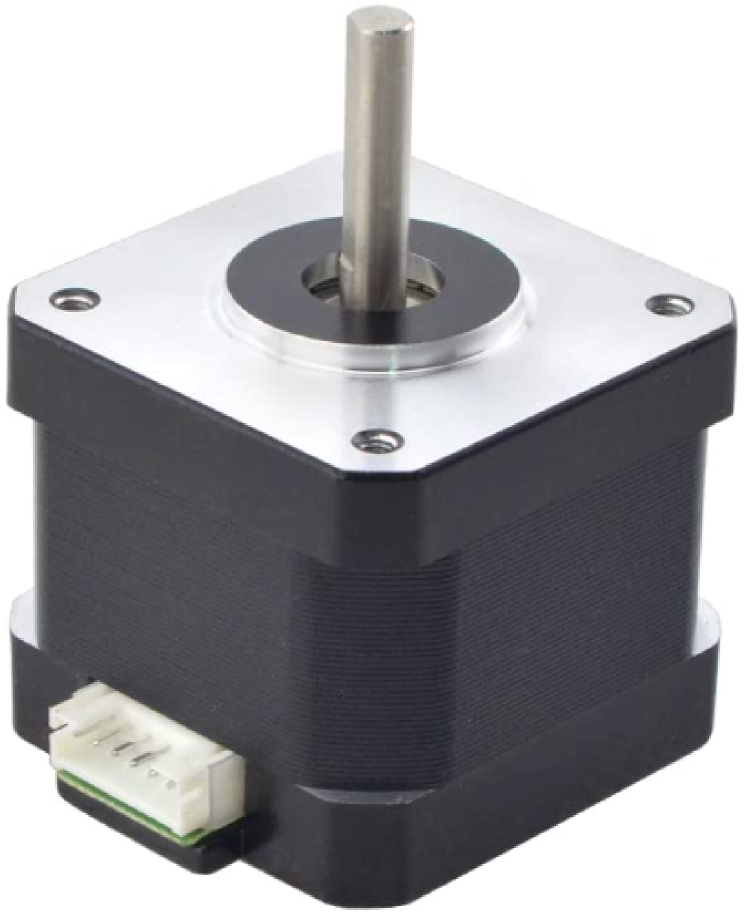


Figure 7: Stepper Motor

Selection Criteria:

We chose this stepper motor (Stepper Motor) due in part to its ready availability and low cost on Amazon. In terms of performance, we selected it because of its ability to be controlled in very precise movements (1.8 degrees per rotation). This accounts to 200 steps across a complete 360-degree rotation of the motor. This stepper motor interfaces quite nicely with the stepper motor controller we selected.

Powering Scheme:

The stepper motors are powered through the stepper motor controllers with 24 V from our power supply. The 24 V goes through a decoupling capacitor of 100 uF. The motors do not draw more than 1 A of current which is easily supplied by the power supply. There are no other power requirements for the motors as the stepper motor controller is responsible for the movement of the motors.

Essential Connections:

The stepper motors are directly connected to the stepper motor controllers. The motor controller provides power with a GPIO pin and then turns it back off. Each power cycle (High, Low) turns the motor 1.8 degrees, with 0.1125 degrees for each microstep.

Subsystem Testing:

To test to see if the motors were working properly, we used the most basic setup on the motor controllers and stepped the motor slowly. This allowed us to observe the current being drawn and make sure nothing was drawing more current than our project could supply. Secondly, we tested the motors moving continuously—or very close to it—to make sure they did not output too much heat. We did this because our stepper motor cases are 3D printed, so if the motors got very hot, there could be a chance to distort the 3D prints.

3.5 Subsystem 3: Motor Controller

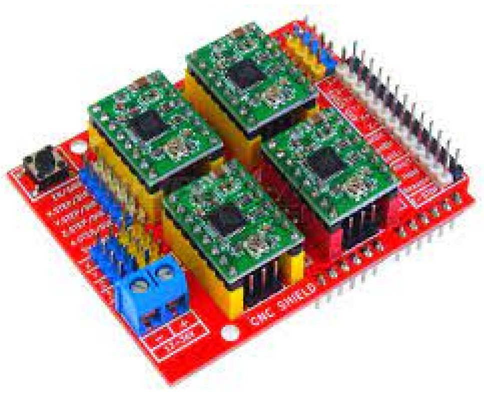


Figure 8: Arduino CNC Shield Expansion Board

Selection Criteria:

The cheapest motor controllers that could microstep over 1/16 of a step were utilized. Additionally, they had to have the ability to handle over one amp of current. The CNC shield expansion board included motor controllers with it that met these requirements, making this an easy selection.

Powering Scheme:

To power the motor controller board, there are two power connections. The first is a 5 V connection used for the VDD on the board. The second connection is 24 V used to power the actual motors through the motor controllers.

Essential Connections:

The essential connections for the motor controller include the power connections and the logic connections to the ESP32. The power connections are the 5 V to the board to act as the VDD supply, and the 24 V, which is used to power the motors. The reason we use 24 V to power the motors is because this lowers the amount of current running through the motor controller boards, which makes the system safer.

Each of the motors have their own direction and step connections to the ESP32. These connections are used to control which way the motors rotate, through the direction pin, and the amount of times they rotate, through the step pin.

Subsystem Testing:

To test if the motor controllers were working correctly, we tested in four phases. The first phase was powering the board and making sure the proper voltages were in the correct spots across the board. To do this, we hooked up the board solely to the power supply, and then we used a multimeter to check the voltages across the board.

Once we had the correct voltages on the board, we wrote a simple program to step one motor five times in one direction, then five times in the other direction. This test was to make sure the motor controllers could properly rotate a motor the correct number of revolutions and rotate the motors both directions.

The third test we conducted was making the program we used above into a while loop. This was to test how well the motor controllers could operate in a continuous manner. This way we would not have to worry about either the motors or the motor controllers becoming too warm or any other complications after a long run time. The last test we did was to use the same program, but expand it to all four of the motors. This demonstrated that we could operate all the motors at the same time without any power limitations. This last test occurred using the CNC Shield Expansion Board, enabling much simpler wiring to the motors.

3.6 Subsystem 4: micro SD Card Module: MEM2075

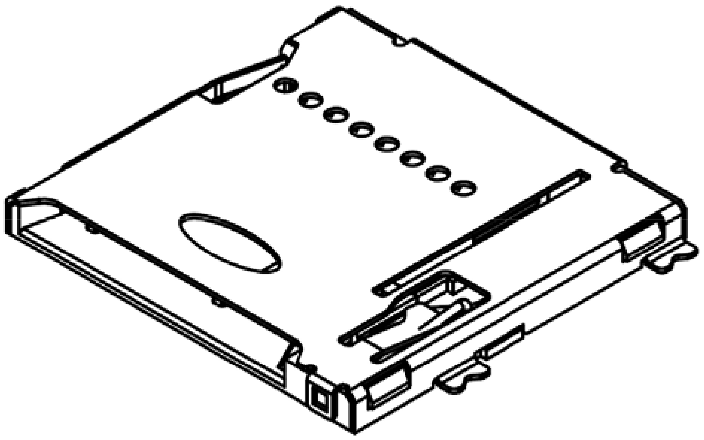


Figure 9: Model of our Micro SD Card Module

Selection Criteria:

We chose this micro SD module because it was readily available, it had good reviews by other customers, the data sheet had actually useful information, it was a push to insert and push to eject model, and it only required 3.3 V to work. The push-push model of the micro SD card was important because the breakout board version we used to test and get code working was not. It was very difficult to remove the micro SD card at times, so we knew we wanted to get a better version for our final project.

Powering Scheme:

The micro SD card module only needs a 3.3 V voltage source, which is readily available on our board in the form of a VDD trace.

Essential Connections:

The micro SD card uses a SPI interface to communicate with the microcontroller. It requires MISO, MOSI, CS, and CLK connections to communicate back and forth.

Subsystem Testing:

There were two parts of subsystem testing. One was to be able to write to the micro SD card, and the other was to do it using our actual micro SD module instead of the breakout board.

For the first subsystem test, the goal of the test was to get a working program to write onto the micro SD card. This was accomplished with arduino libraries: SPI.h and SD.h. These libraries were essential because they have pre-made functions that were easily implemented and allowed us to write to the micro SD card in under an hour. However, the one issue with these premade functions was that you could only implement character arrays into them. The next step in this initial testing was to find a way to write a variable, which represents a scan number, onto the card. To do this, we used a function called dtsorf(). This function takes in a numerical variable and converts it into a character array, which is the format the SD library requires for its functions.

The second subsystem test used the same code developed from the first test, but this time used the board we designed and built to write to the micro SD card. This was a test to see if our traces were connected correctly and that the micro SD module actually worked. This test went smoothly, and we were able to write to the micro SD card easily.

3.7 Subsystem 5: Wireless Communication

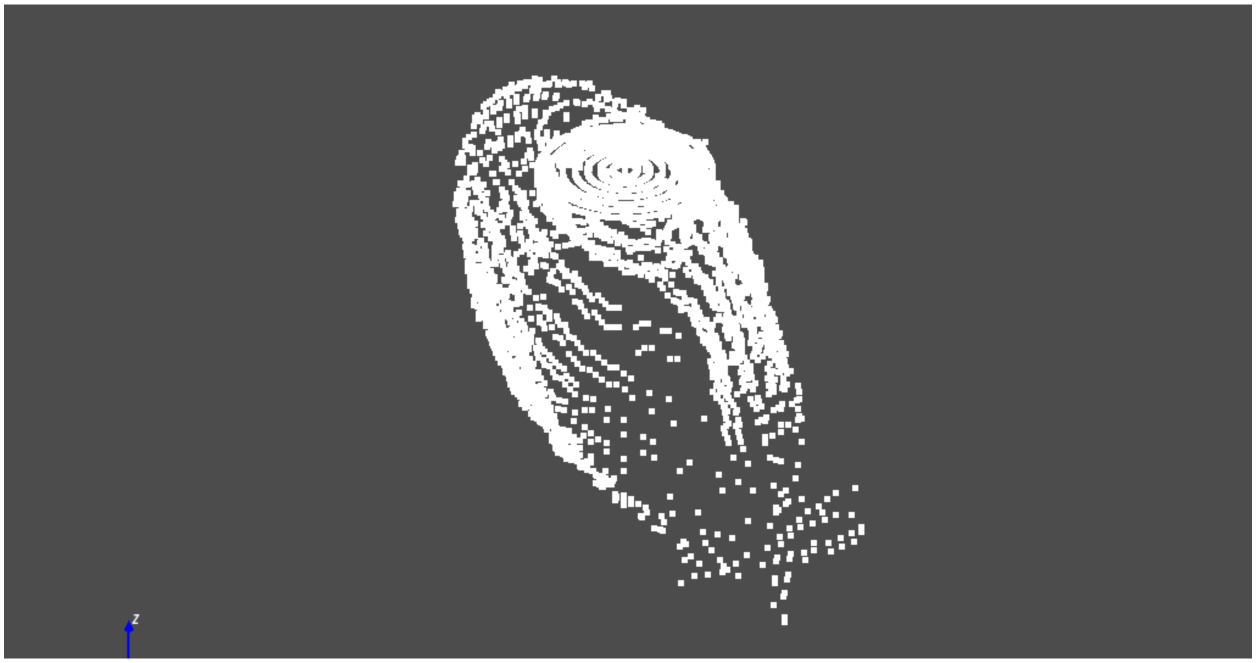
Essential Connections:

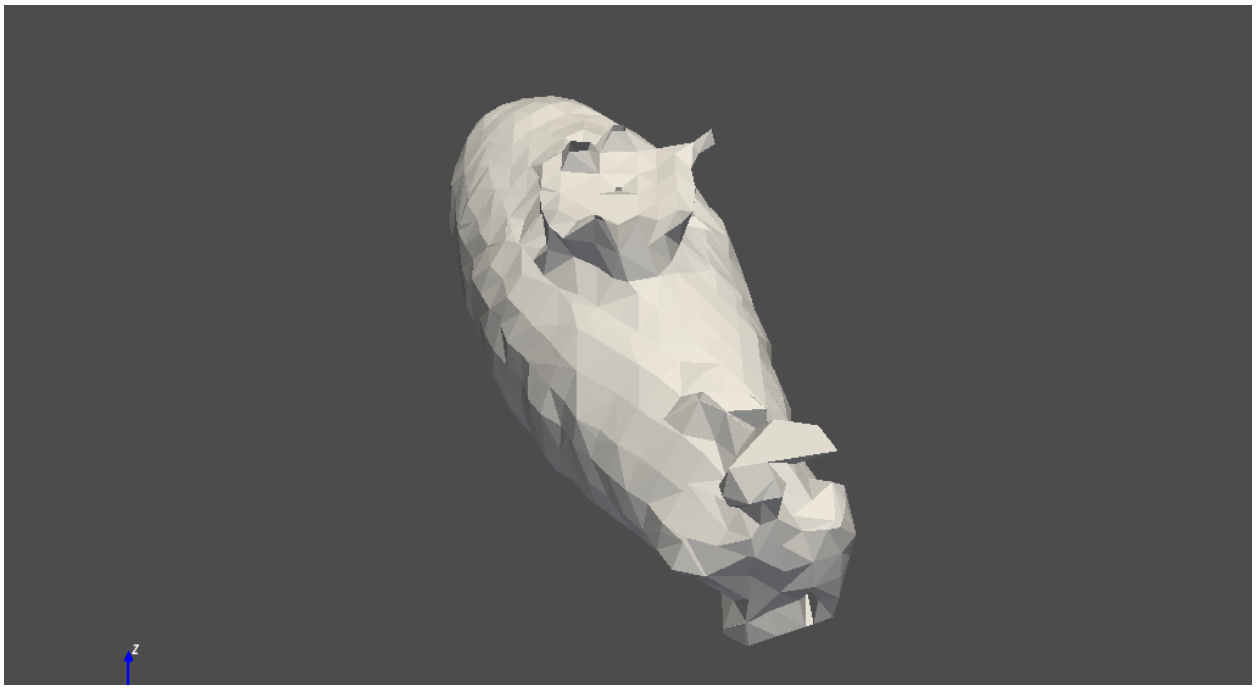
The ESP32 has an innate wireless communication capability. It utilizes a native wireless communication standard, which we used to connect to a website. In order to send commands to the system, users are required to connect to a specific network at a specific IP address.

Subsystem Testing:

To test our system, we simply powered our board and ran a script that configures our wireless ability. The script contains information regarding network authentication, IP addresses, and website structure. The device was able to successfully connect to the senior design network, and we verified its IP address to be the one we assigned. As for sending wireless commands, we designed an HTML and CSS-based website that allowed us to click buttons to write GPIO pins to high and low values. Changes in our commands were verified through the state of an LED.

3.8 Subsystem 6: STL File Creation





Selection Criteria:

In order to prove that our scanner is capable of replicating objects to scale, we needed a means of converting the data into a physically renderable format. Because the process of creating an STL file is computationally intense, we could not rely on the ESP32 to simultaneously handle the execution of the sensor scans, motor controllers, SD card writing, and data processing.

Powering Scheme:

For the STL file creation process, the Python program runs on the user's personal computer. There are no specific powering schemes needed for this subsystem, as it is not a device that is directly linked to our board.

Essential Connections:

The user needs to have a connection with the micro SD card, as he/she will need to access the scan file on it. Afterwards, they will need to ensure that the STL creation program is in the same directory as the scan file.

Subsystem Testing:

To test the system, we had to collect scan data first. For our demonstration day, we scanned a foam submarine. We took the distance measurements, filtered them, and interpolated between the values. For each sensor, we set a maximum allowable distance value so that if the value was detected by the program, it would be filtered out. Also, values that jump sharply (i.e. 1 percent) would be removed from the array. The system linearly interpolates between each horizontal scan plane; to do so, we first set consecutive planes as upper and lower planes, and the system then creates 2 additional subdivisions of points between them. This effectively quadruples the number of points that we have and allows us to generate a more refined surface mesh. To create the surface mesh, we used the PyVista library to triangulate between all the points. The function also allows the user to input the number of nearest numbers that it should use for a triangulation. Generally, estimation with more neighbors is better. However, using too many neighbors can prove to be problematic, especially for objects that have complex shapes and designs.

3.9 Subsystem 7: Microcontroller Board

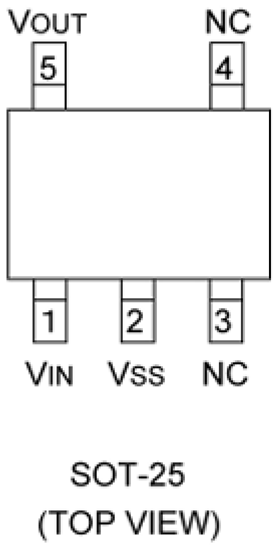


Figure 10: XC6201 Linear Voltage Regulator

Figure 11: Our Microcontroller

Selection Criteria:

There were two major components selected for our board. The first is the microcontroller and the second is our voltage regulators. For the microcontroller, we went with the ESP32-WROOM-32E. This was chosen because it had a large number of pins available to use. Secondly, it had built-in wifi capabilities. It was readily available online, and the datasheet was easily understandable. Lastly, it had a large amount of processing power for the size of the chip. At the time of selection, we did not know how much processing power we were going to need, and we wanted to be safe.

The second component chosen was the XC6201 linear voltage regulator. This was chosen because it had the correct voltage outputs and inputs for our project. The five pin layout means it could not be misplaced on the board during assembly. Also, the regulator was in stock and on the cheaper side, and it has a high efficiency rating.

Powering Scheme:

The board is powered through two 5 V connections from the power supply. Both of these 5 V inputs go through the aforementioned voltage regulator and are bumped down to 3.3 V each. One of the 3.3 V sources is used to power the microcontroller and everything connected to it. The other 3.3 V source is used to power the JRT LiDAR sensor because it requires 300 mA of current, which the ESP32 cannot supply.

Essential Connections:

The whole board is an essential connection because it handles everything that the project does. The ESP32 communicates with all of the sensors and motor controllers to ensure smooth operation of the project. The most important essential connections are the programmer header pins. These allow us to program and actually write to the ESP32.

Subsystem Testing:

The testing of the board was done gradually, testing different components at a time. The first test done was to ensure that there were no shorts on the board after production. This meant combing over the board with a multimeter and making sure all of the connections that were supposed to be connected were, and those that were not supposed to be connected were not. The next step was to power the board through our 5 V power supply and ensure that our voltage regulators were working. This entailed plugging in the board to the power supply and using the multimeter to confirm a 3.3 V reading where it was supposed to be. Once that was done, we wanted to make sure we could program the microcontroller. To test this, we made a simple program using a GPIO pin to light up an LED. The next step was to test the micro SD card module on the board. After we verified both of these components were working, we knew that the board assembly had gone smoothly and there were no issues with that part of the process.

3.10 Subsystem 8: Frame Assembly

The frame assembly parts were chosen based primarily on availability and cost. Parts needed include linear rods to guide the carriages holding the sensors, a linear screw, and a coupler that connects the motor to the linear screw. The rest of the major frame assembly parts were designed in Autodesk Inventor and 3D printed at the EIH.

3.11 Subsystem 9: Power Supply

Selection Criteria:

The selection criteria for the power supply was cost and the ability to provide 5 V and 24 V from a wall outlet. The 24 V connection is necessary to power the motors, while the 5 V is necessary for various sensors and the ESP32 board. Additionally, the power supply only needs to supply 1.5 A at any given time, which many power supply products are easily capable of.

Powering Scheme:

The power supply receives power from a wall outlet. Then it provides 5 V, 12 V and 24 V outputs.

Subsystem Testing:

This subsystem was simple to test. A multimeter was placed between the various outputs of the power supply and ground to measure the voltage. Because all the voltages corresponded to their expected values, no further testing was needed.

3.12 Interfaces

The frame interfaces with all our 3D-printed mounts for our motors and power supply. 3D-printed mounts also interface with our sensors to hold them. The sensor mounts interface with two linear rods (via bearings) and a screw shaft that is directly attached to a stepper motor. Our power system directly interfaces with our board and stepper motor controllers. The board interfaces with the sensors, stepper motor controllers, and SD card module. Additionally, the ESP32's wireless capability creates an interface between the user's personal computer and the microcontroller. Finally, the SD card module interfaces with the user's personal computer, as the STL creation program pulls the scan data from it.

In final testing, all of these interfaces fit together seamlessly, giving the user control of all necessary subsystems and abstracting complexity away when at all possible. This led to a successful product that interfaced well with the user, with individual subsystems working together to accomplish the larger task at hand.

4 System Integration Testing

4.1 Description of Subsystem testings

The integration testing of the subsystems was carried out in multiple stages. We went through each subsystem, and if it worked closely with another subsystem, then we made sure those subsystems worked together before putting it all together.

The first pair of subsystems integrated together were the stepper motors and the motor controllers. This was a relatively straightforward subsystem pair to test because they cannot work as intended without each other. We tested this pair by rotating the motors at various rates and in both directions. This showed we had the ability to fully control the motors.

The next integrated set of subsystems we tested was the various sensors and the micro SD Card module. We wanted to make sure that we could write the various distance measurements we received onto the SD card. To do this test, we took measurements with all of the sensors we had then wrote those measurements in groups of three onto the SD card. This way, we could see how the sensors performed compared to each other. As we expected, the JRT sensor performed the best with both accuracy and consistency when compared to the other sensors.

The last set of subsystem integration testing was using wireless communication to turn the motors on. This test would demonstrate the ability of the user to interact with the website and cause the project to respond. All we did for this test was assign the enable pins of the motors to a button on the website. Once the user hit a button, the enable pin turned on, thus causing the motors to rotate.

4.2 Overall System Requirement Success

The final part of testing was to experiment with the overall integration of the system and make sure all design goals were met. First, the team completed a low quality scan of a 5 inch tall object in 4 minutes. The quality of this scan left much to be desired; however, the system was able to perform its basic function. Next, the system exceeded our expectations and completed a high quality scan of the same object in under 5 hours (this scan can be seen in the STL creation section of the report). In order to accomplish this, all of the other subsystem requirements were met (accurate lidar, precise motor control, STL creation, and wireless compatibility). Additionally, the system proved robust and functioned extremely well, even after multiple repeated scans. Because of these positive results, the team concluded that the integrated system was a success and met all original design requirements.

5 Users Manual/Installation Manual

5.1 How to install your product

This product is primarily business facing and already assembled. Because of this, the consumer application to this product is somewhat limited. However, a limited and basic installation process should still be followed.

First, the user should make sure all towers are securely fastened to the platform. If all mechanical connections are secure, the user should then move the platform to the location in which they would like to perform scans. While capable of moving, the user should ideally select a semi-permanent location for installation. Because the product comes pre-assembled, the installation for this project is straightforward.

5.2 How to setup your product

In order to set up the product, the user must follow the procedure to start a scan. First, the user must place an object on the turntable to be scanned. If the object exceeds the edges of the turntable, the scan quality may be compromised. Additionally, the user should be sure to fix the object to the center of the turntable, which is denoted by the drilled hole in the center. It is recommended that for lighter objects that may move about during operation of the scanner that the object to be scanned is fixed to the platform with a light dab of glue or double sided tape.

After the object being scanned has been correctly placed on the platform, the user should plug the power cord into a normal wall outlet. After powering the device from the wall, the user should then go to the IP address (**192.168.10.69**) in order to select the quality of scan. The user will then be able to select the quality of the scan they would like to perform, as well as an estimated time to complete that scan.

The user has three quality options: high, medium, and low. By clicking on the desired option, the scan will begin.

The scan will complete when the turntable stops spinning and the tower lowers itself back to the starting position. The user should unplug the system from the wall and remove the easily accessible SD card from its mounting location.

The user then can load the SD card into any computer and enter the file name of the SD card into the provided program. After doing this, the image of the point cloud will populate and an STL file will be generated.

5.3 How the user can tell if the product is working

The user will be able to determine if the scan is functioning properly by two ways. First, if the turntable is smoothly turning and the selected tower is smoothly raising, the product is functioning as intended. Additionally, upon retrieval of the SD card and after running the point mapping program, the user should be able to see a plot of the data points. If this plot looks nothing like the object, there is most likely an error with the sensor selected.

5.4 How the user can troubleshoot the product

If the product fails to function properly, first retry the scan. If the page to start the scan is not loading properly, it may be necessary to reset your wifi connection. If this does not rectify the problem, the user should contact an instructor at Basic Underwater Demolition School and ask for Jack Rourke, as he will need to dispatch a trained professional technician to diagnose and remedy the problem. He should be easily accessible at all times and more than happy to assist with any troubleshooting queries. In all seriousness, if there is a serious issue, such as a sensor or motor defect, it is highly unlikely anyone but a trained technician would be able to fix this problem (i.e. Jack Rourke).

6 To-Market Design Changes

There are several to-market design changes that could increase the utility of this product. First, the stepper motor controllers used in this project could be upgraded. The motor controls used currently offer microstepping to 1/16 of 1.8 degrees. However, there are several microcontrollers on the market that offer 1/128 or 1/256 microstepping. Additionally, the stepper motors could be upgraded to a 0.9-degree stepper motor. By combining these two changes, the user would achieve 32x more precise microstepping, allowing for more accurate and smoother scans.

In addition to the addition of higher quality motors and motor controllers, the biggest addition to this project would be the inclusion of a high quality LiDAR sensor. A 1 mm resolution LiDAR sensor that scans at a rate of greater than 270 Hz could be purchased for around $7,000 (https://www.aliexpress.us/item/2255800758376025.html?gatewayAdapt=glo2usa4itemAdapt&_randl_shipto=US). The inclusion of this sensor would not only increase the accuracy of the system, it would also reduce a scan that would currently take 10 hours to one that would take just under 5 minutes.

Additionally, if accuracy of the scan was a peak objective, then this sensor could be included: https://ouster.com/products/scanning-lidar/os1-sensor/. Coming in at just under 20,000 dollars, the sensor has the best resolution on the market in addition to being able to scan 5.2 million points per second. An inclusion of a sensor like this would enable this project to be used to almost instantly evaluate products for quality control.

Finally, the last to-market design changes would consist of a more professional mounting system and frame. Currently, the project is supported on a wooden tray; however, if a $20,000 sensor was added, it seems reasonable that the user would expect a higher quality construction. This would include molded or laser cut parts and a quality base made out of non-conductive materials.

If these overall changes were implemented, this project would be considerably more accurate and scan much faster. These improvements would make it viable for use in quality assurance—something a business might be willing to spend thousands of dollars on to automate.

7 Conclusions

Overall, the project was a success. The original design was executed well and ended up functioning as expected. The biggest takeaway the group had from this project was that a little bit of work up front goes a long way in the end. For instance, the team implemented a CNC Shield about two weeks before the final deadline due to connection issues with the stepper motors and controllers. Figuring out that this was a positive design improvement towards the beginning of the process would have greatly reduced the amount of time spent working with the stepper motors.

Another major takeaway from the project was that our group had a very successful division of work. Each member of the team was highly focused and proficient in a specific field. Because of this, the team was able to complete tasks efficiently and team members did not interfere with each other's work.

In conclusion, this project has equipped all team members with valuable experience in building a working project from scratch. Reading datasheets, finding valuable code libraries online, and scouring Stack Overflow saved the group many times. Special thanks for a successful project goes to Amazon One-Day Shipping, Digi-Key, and the soldering irons in Stinson-Remick Room 205. There were many pitfalls along the way, but the grit, bearing, and determination of all group members made this project possible.

Forged by the sea,
LiDAR Team

8 Appendices

8.1 Schematics

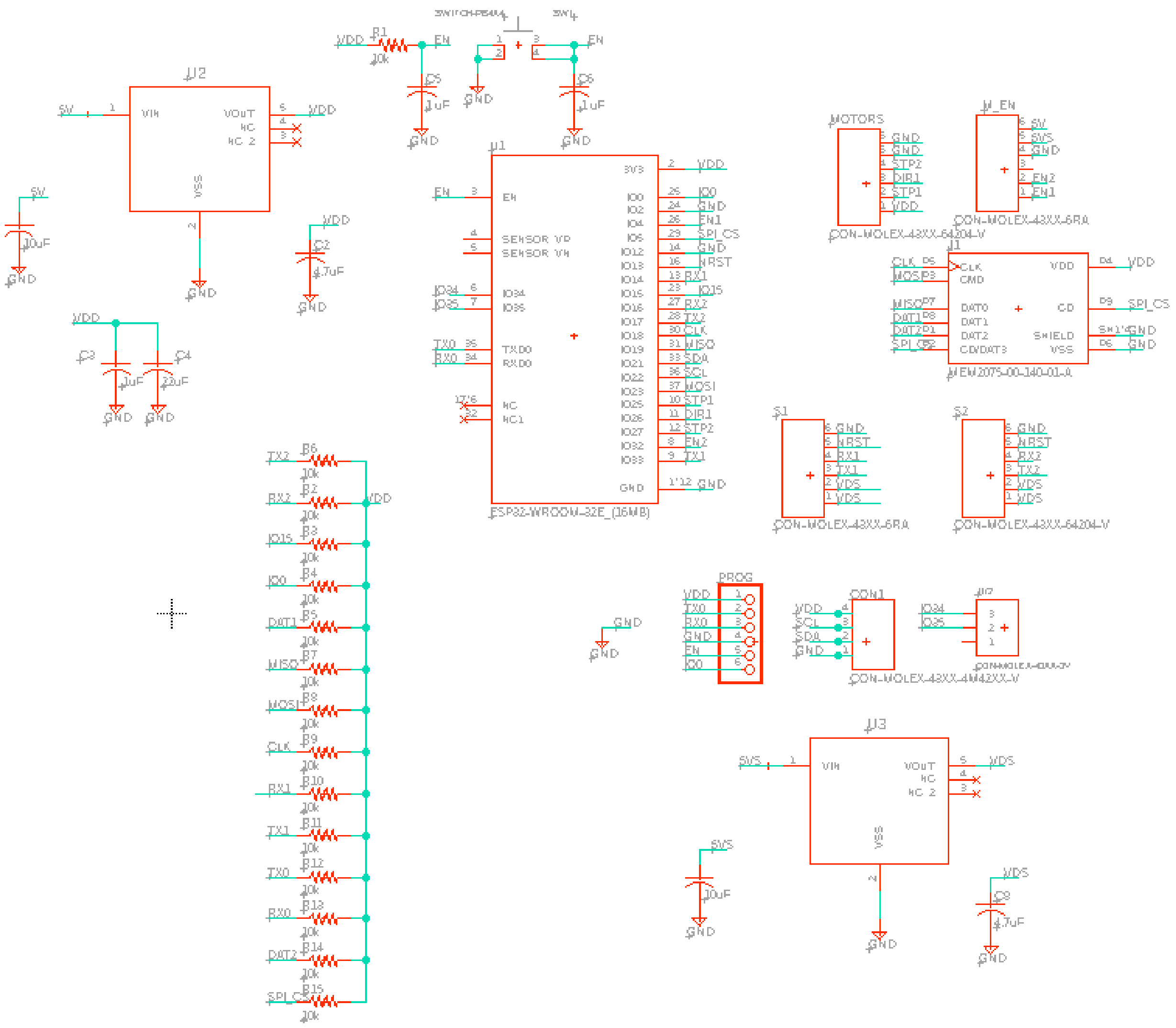


Figure 12: Board Schematic

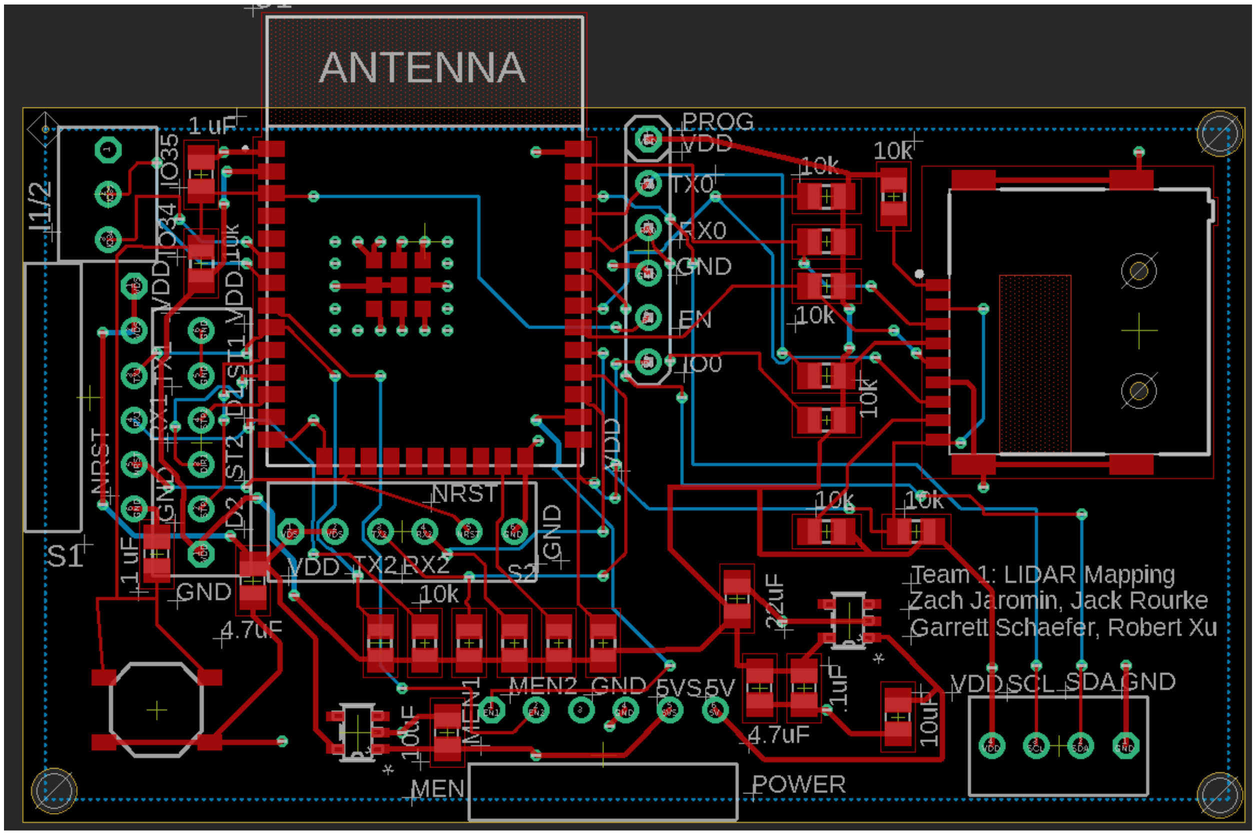


Figure 13: PCB Layout

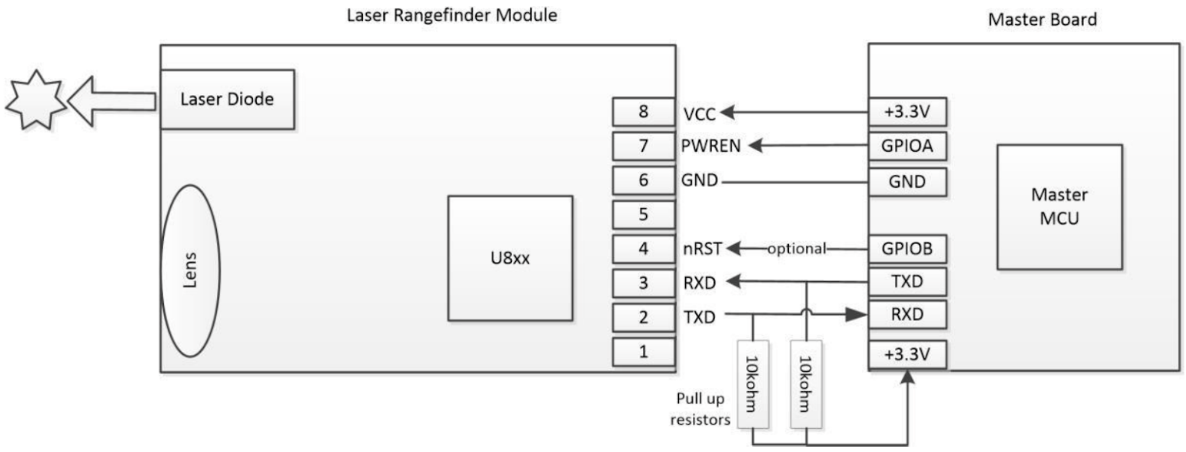


Figure 14: JRT Sensor Schematics

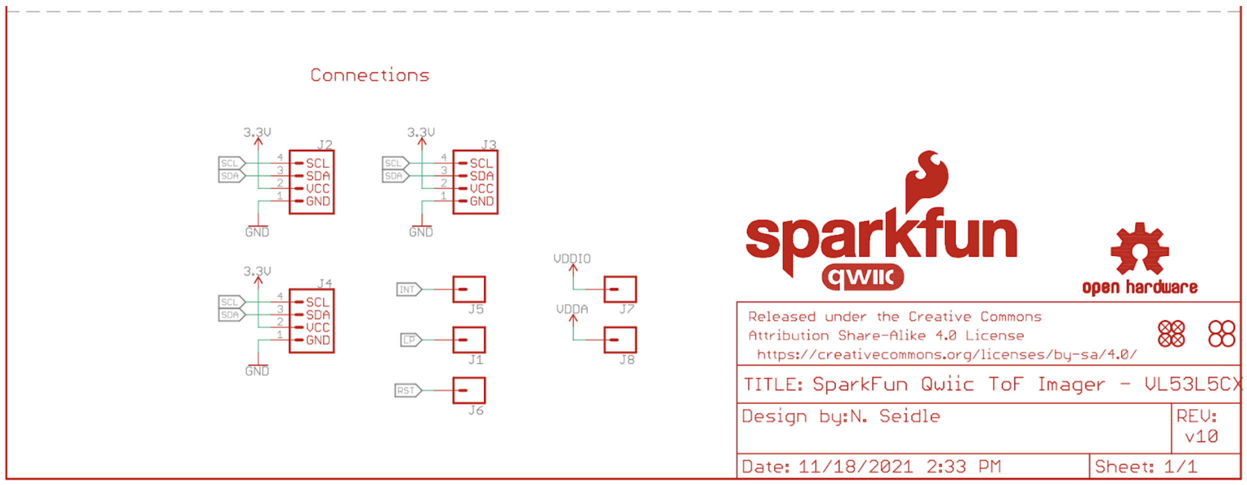

Connections
3.3V
J2
SCL
SDA
VCC
GND
J3
J4
GND
INT
J5
LP
J1
RST
J6
VDDIO
J7
VDDA
J8
sparkfun
qwiic
open hardware
Released under the Creative Commons Attribution Share-Alike 4.0 License https://creativecommons.org/licenses/by-sa/4.0/
TITLE: SparkFun Qwiic ToF Imager - VL53L5CX
Design by:N. Seidle
REV: v10
Date: 11/18/2021 2:33 PM
Sheet: 1/1

Figure 15: Sparkfun Sensor Schematic

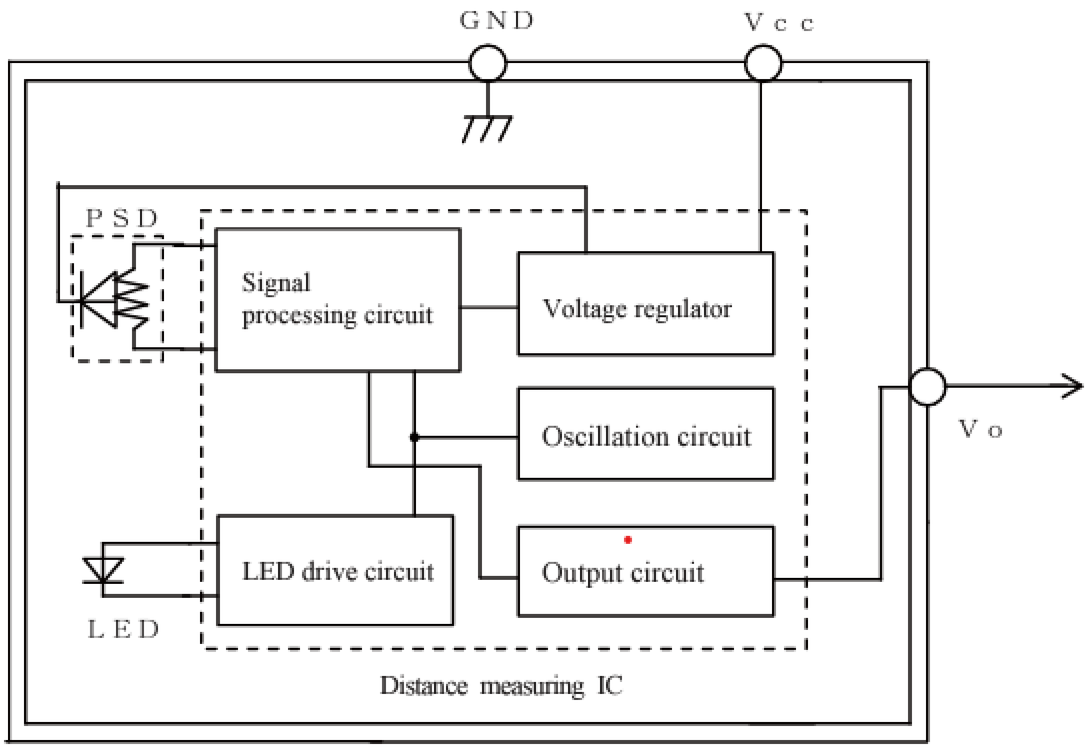


Figure 16: Analog Distance Sensor Block Diagram

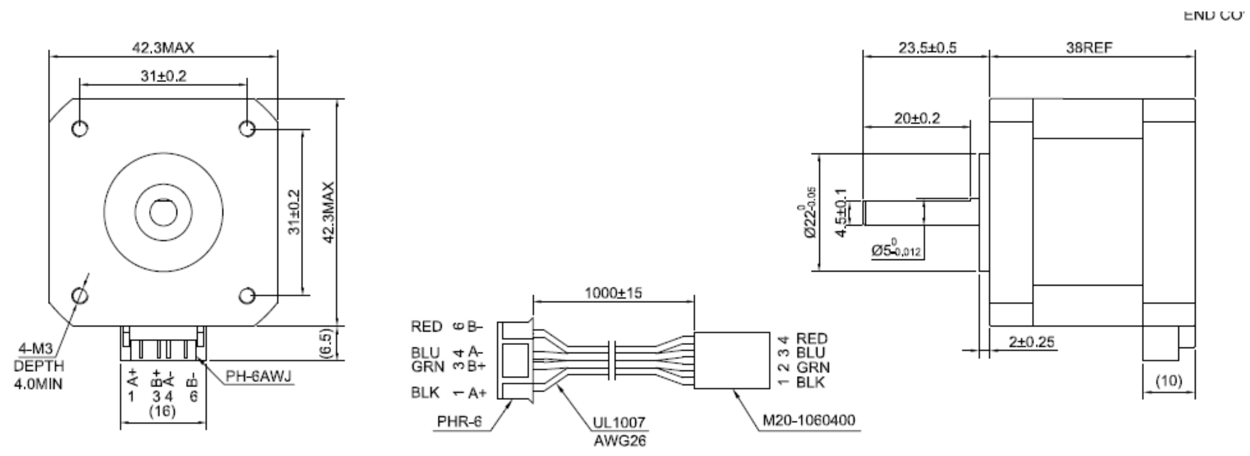


Figure 17: Stepper Motor Schematic

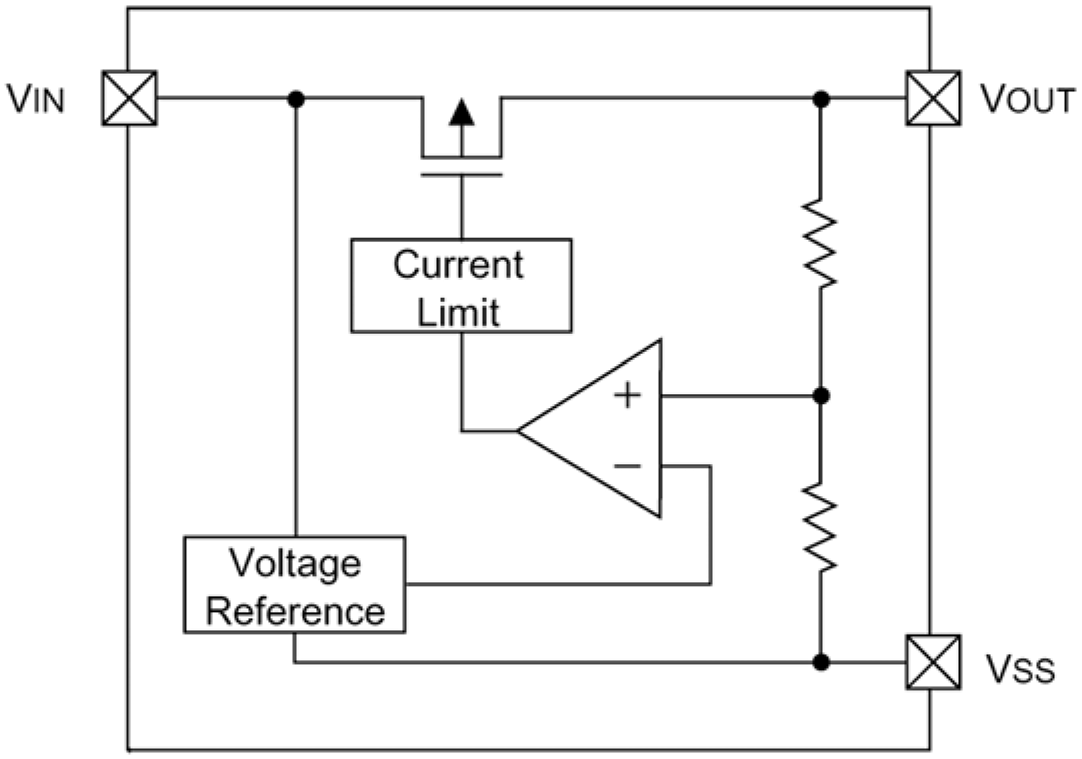


Figure 18: Voltage Regulator Schematic

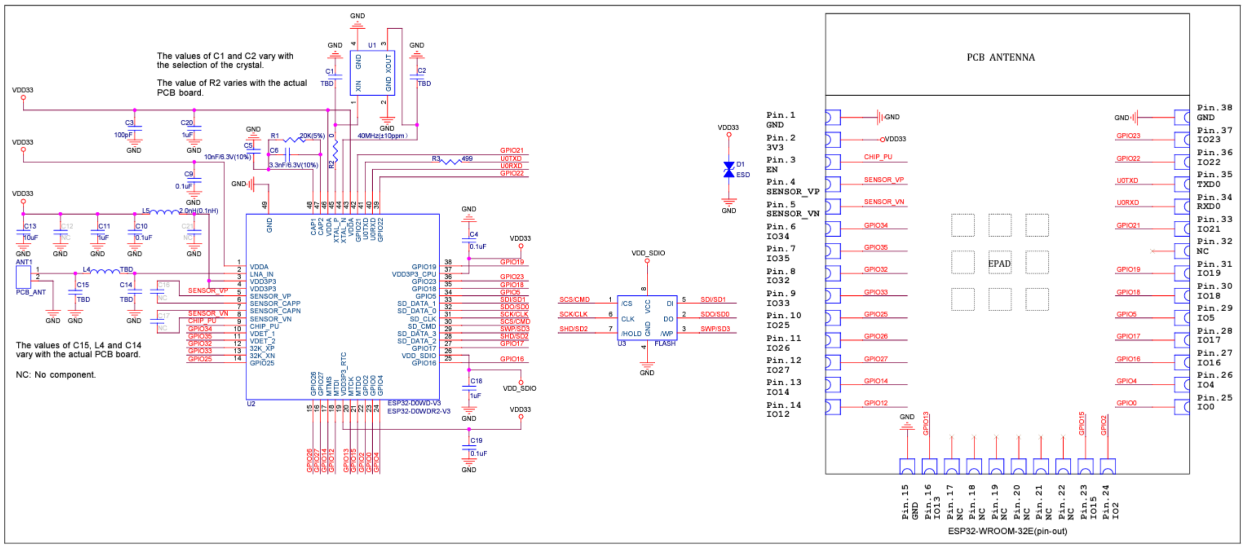


Figure 5: ESP32-WROOM-32E Schematics
PCB ANTENNA
ESP32-WROOM-32E(pin-out)
The values of C1 and C2 vary with the selection of the crystal.
The value of R2 varies with the actual PCB board.
The values of C15, L4 and C14 vary with the actual PCB board.
NC: No component.

Figure 19: ESP32-WROOM-32E Schematic

8.2 Code (Main File Only)

```
#include <Arduino.h>
#include <Wire.h>
#include "SparkFun_VL53L1X.h"
#include "Sharp.h"
#include <HardwareSerial.h>
#include <U8xLaser.h>
#include <iostream>
#include <motor.h>
#include <Wifi.h>
#include "SD_write.h"
#include "FS.h"
#include "SD.h"
#include "SPI.h"
#include "stdio.h"

// Motor Pin Definitions
#define C_RESET 13
#define STP_Pl 4
#define STP_SK 26
#define DIR_SK 33
#define DIR_A 14
```

```
#define STP_A 25
#define STP_CH 32
#define DIR_CH 27
#define REVOLUTIONS 200

// Analog Sensor Pin
const byte A_sensorPin = 35;

// Chinese Sensor UART Sensor
HardwareSerial *MySerial = &Serial2;
U8xLaser Laser1 = U8xLaser(*MySerial,C_RESET);
SFEVL53L1X distanceSensor;

// Wifi Setup
#define ANALOG_S 1
#define CHINESE 2
#define SPARK 3
const char* ssid = "SDNet";
const char* password = "CapstoneProject";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

String outputAnalogState = "off";
String outputChineseState = "off";
String outputSparkState = "off";

IPAddress local_IP(192,168,10,69);
IPAddress gateway(192,168,10,1);
IPAddress subnet(255,255,0,0);
IPAddress dns(192,168,1,1);

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
```

```
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 10000;

// SD Card
using namespace std;
void RunScan(int sensor_type);

void setup() {
  Wire.begin();

  Serial.begin(115200);
  delay(2000);
  uint32_t baudrate = 19200;
  Laser1.reset();
  Laser1.begin(baudrate);

  Serial.println("VL53L1X Qwiic Test");

  if (distanceSensor.begin() != 0) //Begin returns 0 on a good init
  {
    Serial.println("Sensor failed to begin. Please check wiring. Freezing...");
    while (1)
      ;
  }

  configMotors_raise(STP_CH,DIR_CH);
  configMotors_raise(STP_A,DIR_A);
  configMotors_raise(STP_SK, DIR_SK);
  configMotors_rotate(STP_Pl);


  Serial.println("Sensor online!");

    if (WiFi.config(local_IP,gateway,subnet,dns,dns) == false)
  {
    Serial.println("Configuration failed");
  }

  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid,password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();


  Serial.print("SD Card \n\r");
  // SD Card Setup
  if(!SD.begin(5)){
    Serial.println("Card Mount Failed");
    return;
  }
  uint8_t cardType = SD.cardType();

  if(cardType == CARD_NONE){
    Serial.println("No SD card attached");
    return;
  }

  Serial.print("SD Card Type: ");
  if(cardType == CARD_MMC){
    Serial.println("MMC");
  } else if(cardType == CARD_SD){
    Serial.println("SDSC");
  } else if(cardType == CARD_SDHC){
    Serial.println("SDHC");
  } else {
    Serial.println("UNKNOWN");
  }
  uint64_t cardSize = SD.cardSize() / (1024 * 1024);
  Serial.printf("SD Card Size: %lluMB\n", cardSize);

  deleteFile(SD,"/scan.txt");

}

void loop() {

  int sensor_type;

  WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                             // If a new client connects,
    currentTime = millis();
    previousTime = currentTime;
    Serial.println("New Client.");          // print a message out in the serial
port
    String currentLine = "";                // make a String to hold incoming
data from the client
    while (client.connected() && currentTime - previousTime <= timeoutTime) {  //
loop while the client's connected
      currentTime = millis();
      sensor_type = 0;
        if (client.available()) {             // if there's bytes to read from
the client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial monitor
        header += c;
        if (c == '\n') {                    // if the byte is a newline character
          // if the current line is blank, you got two newline characters in a
row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200
OK)
            // and a content-type so the client knows what's coming, then a blank
line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            // turns the GPIOs on and off
            if (header.indexOf("GET /26/on") >= 0) {
              sensor_type = CHINESE;
              outputAnalogState = "on";
              Serial.println("Analog Button");
            } else if (header.indexOf("GET /26/off") >= 0) {
              outputAnalogState = "off";
            } else if (header.indexOf("GET /27/on") >= 0) {
              sensor_type = SPARK;
              outputChineseState = "on";
              Serial.println("Chinese Button");
            }else if (header.indexOf("GET /27/off") >= 0) {
              outputChineseState = "off";
            }
            else if (header.indexOf("GET /4/on") >= 0) {
              sensor_type = ANALOG_S;
              outputSparkState = "on";
              Serial.println("Spark Button");
            }
            else if (header.indexOf("GET /4/off") >= 0) {
              outputSparkState = "off";
            }

            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");
            // CSS to style the on/off buttons
            // Feel free to change the background-color and font-size attributes
to fit your preferences
            client.println("<style>html { font-family: Helvetica; display:
inline-block; margin: 0px auto; text-align: center;}");
            client.println(".button { background-color: #4CAF50; border: none;
color: white; padding: 16px 40px;");
```

```
            client.println("text-decoration: none; font-size: 30px; margin: 2px;
cursor: pointer;}");
            client.println(".button2 {background-color: #555555;}");
            client.println(".pad {margin-bottom: 50px;}");
            client.println(".box {width: 200px; background-color: #f0f0f0;
padding-top: 1%; padding-bottom: 1%; padding-left: 2%; padding-right: 2%; margin:
auto; border-radius: 10px;}</style></head>");

            // Web Page Heading
            client.println("<body><h1>3D Object Scanner</h1>");

            // Box to enclose all shit
            client.println("<div class=\"box\">");

            // Display current state, and ON/OFF buttons for GPIO 26
            client.println("<p><b>High Quality Scan</b></p>");
            // If the output26State is off, it displays the ON button
            if (outputAnalogState =="off") {
              client.println("<p><a href=\"/26/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/26/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }

            // Display current state, and ON/OFF buttons for GPIO 27
            client.println("<p><b>Medium Quality Scan</b></p>");
            // If the output27State is off, it displays the ON button
            if (outputChineseState=="off") {
              client.println("<p><a href=\"/27/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/27/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }

            // Display current state, and ON/OFF buttons for LED on Pin  4
            client.println("<p><b>Low Quality Scan</b></p>");
            // If the output4State is off, it displays the ON button
            if (outputSparkState =="off") {
              client.println("<p><a href=\"/4/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/4/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }

            client.println("</div>");

            client.println("</body></html>");

            // The HTTP response ends with another blank line
            client.println();
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage
return character,
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");

    RunScan(sensor_type);
    sensor_type = 0;
  }
  delay (50);
}

void RunScan(int sensor_type){
  int M_dist = 0;
  int h_counter = 0;
  int r_counter = 0;
switch (sensor_type){
    case ANALOG_S:
        while(M_dist < (REVOLUTIONS/2)){
          unsigned vin = analogRead(A_sensorPin);
          float v = float(vin)/4096 * 3.3;
          float distance2 = getDist(v);
          if (distance2 == 1000){
            M_dist++;
            distance2 = 10000;
          }
          if (r_counter == REVOLUTIONS){
            digitalWrite(DIR_A,LOW);
            for (int i = 0; i < 1600; i++){
              raiseSensor(STP_A);
            }
            r_counter = 0;
            M_dist = 0;
            h_counter++;
          }
          else {
          rotatePlatform(STP_P1);
          r_counter++;
          }
          char result[8]; // Buffer big enough for 7-character float
            dtostrf(distance2, 6, 2, result); // Leave room for too large
numbers!
            appendFile(SD, "/scan.txt", result);
            appendFile(SD,"/scan.txt",", ");
            char result2[8]; // Buffer big enough for 7-character float
            dtostrf(h_counter, 6, 2, result2); // Leave room for too large
numbers!
            appendFile(SD,"/scan.txt",result2);
            appendFile(SD,"/scan.txt",", ");
            char result3[8]; // Buffer big enough for 7-character float
            dtostrf(r_counter, 6, 2, result3); // Leave room for too large
numbers!
```

```
            appendFile(SD,"/scan.txt",result3);
            appendFile(SD, "/scan.txt", "\r\n");
            delay(100);
        }
        lowerSensorAnalog(STP_A,DIR_A, h_counter);
      break;
    case CHINESE :

        while(M_dist < (REVOLUTIONS/2)){
         uint16_t dist = Laser1.measureSingle();
          if (dist >= 480){
            M_dist++;
            dist = 10000;
          }
          if (r_counter == REVOLUTIONS){
            digitalWrite(DIR_CH, LOW);
            for (int i = 0; i < 1600; i++){
              raiseSensor(STP_CH);
            }
            r_counter = 0;
            M_dist = 0;
            h_counter++;
          }
          else {
            rotatePlatform(STP_Pl);
          r_counter++;
          }
          char result[8]; // Buffer big enough for 7-character float
            dtostrf(dist, 6, 2, result); // Leave room for too large numbers!
            appendFile(SD, "/scan.txt", result);
            appendFile(SD,"/scan.txt",", ");
            char result2[8]; // Buffer big enough for 7-character float
            dtostrf(h_counter, 6, 2, result2); // Leave room for too large
numbers!
            appendFile(SD,"/scan.txt",result2);
            appendFile(SD,"/scan.txt",", ");
            char result3[8]; // Buffer big enough for 7-character float
            dtostrf(r_counter, 6, 2, result3); // Leave room for too large
numbers!
```

```
            appendFile(SD,"/scan.txt",result3);
            appendFile(SD, "/scan.txt", "\r\n");
            delay(550);
        }
        lowerSensorChinese(STP_CH,DIR_CH, h_counter);
      break;
    case SPARK :
       while(M_dist < (REVOLUTIONS/2)){
          distanceSensor.startRanging(); //Write configuration bytes to initiate
measurement
          while (!distanceSensor.checkForDataReady())
            {delay(1);}
          int distance = distanceSensor.getDistance(); //Get the result of the
measurement from the sensor
          distanceSensor.clearInterrupt();
          distanceSensor.stopRanging();
          if (distance >= 350){
            M_dist++;
            distance = 10000;
          }
          if (r_counter == REVOLUTIONS){
            Serial.println("Raise");
            digitalWrite(DIR_SK, LOW);
            for (int i = 0; i < 1600; i++){
              raiseSensor(STP_SK);
            }
            r_counter = 0;
            M_dist = 0;
            h_counter++;

          }
          else {
          rotatePlatform(STP_Pl);
          r_counter++;
          }
            Serial.print(distance);
            Serial.println();
            char result[8]; // Buffer big enough for 7-character float
            dtostrf(distance, 6, 2, result); // Leave room for too large numbers!
            appendFile(SD, "/scan.txt", result);
            appendFile(SD,"/scan.txt",", ");
            char result2[8]; // Buffer big enough for 7-character float
            dtostrf(h_counter, 6, 2, result2); // Leave room for too large
numbers!
            appendFile(SD,"/scan.txt",result2);
            appendFile(SD,"/scan.txt",", ");
            char result3[8]; // Buffer big enough for 7-character float
            dtostrf(r_counter, 6, 2, result3); // Leave room for too large
numbers!
            appendFile(SD,"/scan.txt",result3);
            appendFile(SD, "/scan.txt", "\r\n");
            delay(20);
        }
        lowerSensorSpark(STP_SK,DIR_SK, h_counter);
      break;
    default :
      delay(1);
  }
}
```

8.3 Data Sheets/Product Pages

Microcontroller
Linear Voltage Regulator
micro SD Card Module
JRT Sensor
Sparkfun Sensor
Analog Sensor