



Department of Electronic & Telecommunication
Engineering,
University of Moratuwa, Sri Lanka.

3D Mapping Device for Object Tracking Design Documentation - Group B

Submitted By:
210728C Wijewickrama W.K.D.D.D.
210031H Amarasinghe A.A.W.L.R.

Submitted in partial fulfillment of the requirements for the module

EN 2160 : Electronic Design Realization

20th June 2024

Contents

1	Introduction	3
2	Component Selection and Justifications	3
2.1	Microcontroller	3
2.2	Time-of-Flight (ToF) Sensors	4
2.3	I2C Multiplexer	5
2.4	USB Serial Communication	5
2.5	Stepper Motor	6
2.6	Stepper Motor Driver	6
2.7	Power Step-Down	7
2.8	Clock Oscillator	7
2.9	Decoupling Capacitors	8
2.10	Pull-up Resistors	8
3	Schematic Design	9
4	PCB Design	10
5	3D Model of Enclosure Design	12
5.1	CAD Tool: SolidWorks 2020	12
5.2	Material: PLA+	12
6	Setting Up Environment	14
6.1	Components	14
6.2	Software Requirements	14
7	Daily Log Entries	15
7.1	26 February – 3 March	15
7.2	4 March – 10 March	15
7.3	11 March – 17 March	15
7.4	18 March – 24 March	15
7.5	25 March – 31 March	16
7.6	1 April – 7 April	16
7.7	15 April – 21 April	16
7.8	22 April – 28 April	16
7.9	29 April – 5 May	16
7.10	6 May – 12 May	17
7.11	13 May – 15 June	17
8	Programming Process	18
8.1	Distance Measurement using TOF sensor (4 March - 10 March)	18
8.1.1	Arduino Code	18
8.1.2	Code Explanation	18
8.2	Stepper Motor Check (11 March - 17 March)	20
8.2.1	Arduino Code	20
8.2.2	Code Explanation	20
8.3	3D Cloud Generation (18 March - 24 March)	23
8.3.1	Python Code	23
8.3.2	Code Explanation	23
8.3.3	Import Libraries	23
8.4	2D Scan (25 March - 31 March)	25
8.4.1	Arduino Code	25

8.4.2	Code Explanation	26
8.5	3D scan (1 April - 7 April)	30
8.5.1	Arduino Code	30
8.5.2	Code Explanation	31
8.6	Serial Communication - Receiver (22 April - 28 April)	34
8.6.1	Python Code	34
8.6.2	Code Explanation	37
9	References	39
10	Cross Checked By	40

1 Introduction

The 3D Scanner device represents a cutting-edge solution for precise and efficient three-dimensional data acquisition in various applications. This design document comprehensively outlines the component selection process and daily progress of the project. It encapsulates the combined work of our multi-disciplinary team, leveraging expertise in optical engineering, electronics, software development, and human-centered design principles.

The device features a Time-of-Flight (ToF) sensor designed for precise measurement of distances and angles. It utilizes UART communication via ATMEGA 2560 microcontroller pins connected to a USB-to-Serial converter, enabling seamless interfacing with computers equipped with USB ports for data transmission and visualization. This setup ensures efficient handling of distance measurement data and robust communication capabilities, making the device suitable for applications requiring accurate spatial data acquisition and analysis.

2 Component Selection and Justifications

2.1 Microcontroller

ATMEGA 2560 MCU

The ATMEGA 2560 microcontroller unit (MCU) was selected as the central processing unit for the 3D surround scanner. This microcontroller was chosen for its adequate memory, processing speed, and overall reliability.

Specifications:

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
- Non-volatile Program and Data Memories
- JTAG (IEEE std. 1149.1 compliant) Interface
- 51/86 Programmable I/O Lines
- Temperature Range: -40°C to 85°C Industrial

Key Features:

- **Memory:** The ATMEGA 2560 offers 256 KB of Flash memory, 8 KB of SRAM, and 4 KB of EEPROM, providing sufficient space for handling the initial stages of data processing.
- **Processing Speed:** It operates at a clock speed of 16 MHz, which meets the demands of real-time data collection and processing required for the 3D mapper.
- **Reliability:** Known for its stable performance, the ATMEGA 2560 is widely used in various applications, making it a dependable choice for our project.

Reasons for Selection:

- **Versatility:** The ATMEGA 2560 supports a wide range of peripherals and interfaces, making it adaptable to various components used in the 3D mapper project.
- **Community Support:** The ATMEGA 2560 has extensive documentation and a large community of users, providing valuable resources and support for troubleshooting and development.

- **Cost-Effective:** Compared to other microcontrollers with similar capabilities, the ATMEGA 2560 offers a good balance between performance and cost, making it an economical choice for the project.
- **Ease of Programming:** The microcontroller is compatible with the Arduino platform, which simplifies programming and prototyping, allowing for faster development cycles.
- **Proven Track Record:** The ATMEGA 2560 has been used in numerous successful projects, indicating its reliability and robustness in real-world applications.

Despite these advantages, the memory capacity of the ATMEGA 2560 was insufficient to store the entire dataset generated during the scanning process. Therefore, the data readings had to be transmitted incrementally to a connected computer for further processing and storage.

2.2 Time-of-Flight (ToF) Sensors

VL53L0 ToF Sensors

Two Time-of-Flight sensors were selected for the 3D mapper to measure the distance of objects with high precision and speed.

Specifications

- **VDD:** Regulated 2.8 V output. Almost 150 mA available to power external components.
- **VIN:** The main 2.6 V to 5.5 V power supply connection. The SCL and SDA level shifters pull the I2C wire high to this level.
- **GND:** The ground (0 V) connection for power supply. I2C control source must also share a common ground with this board.
- **SDA:** Level-shifted I2C data line; HIGH is VIN, LOW is 0 V.
- **SCL:** Level-shifted I2C clock line; HIGH is VIN, LOW is 0 V.
- **XSHUT:** This pin is an active-low shutdown input; the board pulls it up to VDD to enable the sensor by default.

Reasons for Selection:

- **High Precision:** ToF sensors are capable of measuring distances accurately, which is crucial for creating a detailed 3D map.
- **Speed:** These sensors can take measurements rapidly, enabling real-time data collection and processing.
- **Range:** ToF sensors can cover a sufficient range suitable for the intended mapping application.

2.3 I2C Multiplexer

I2C Multiplexer

An I2C multiplexer was used to connect the two ToF sensors to the ATMEGA 2560.

Specifications:

- 1-to-8 Bidirectional Translating Switches
- I2C Bus and SMBus Compatible
- Allows Voltage-Level Translation Between 1.8-V, 2.5-V, 3.3-V, and 5-V Buses
- Operating Power-Supply Voltage Range of 1.65-V to 5.5-V
- Low RON Switches

Reasons for Selection:

- **Multiple I2C Devices:** The multiplexer allows multiple I2C devices to be connected to a single I2C bus, expanding the number of sensors that can be used simultaneously.
- **Simplifies Wiring:** Reduces the complexity of wiring by allowing easy addition of sensors without requiring additional I2C ports on the microcontroller.
- **Efficient Communication:** Ensures efficient communication between the microcontroller and the sensors, maintaining the integrity and speed of data transfer.

2.4 USB Serial Communication

CH340C Chip

The CH340C chip was chosen for USB serial communication between the ATMEGA 2560 and the connected computer.

Specifications:

- Full-speed USB device interface, compatible with USB V2.0
- Fully compatible with serial port applications
- Supports common MODEM contact signals
- Provides RS232, RS485, RS422 interfaces
- Supports 5V and 3.3V power supply voltages

Reasons for Selection:

- **Reliable Communication:** Provides a reliable interface for serial communication over USB, essential for transmitting data to the computer for further processing.
- **Compatibility:** Compatible with a wide range of operating systems, ensuring ease of integration and use.
- **Cost-Effective:** The CH340C is a cost-effective solution for USB to serial conversion, making it an economical choice for the project.

2.5 Stepper Motor

Nema17

The NEMA 17 stepper motor is renowned for its versatility and reliability in various applications requiring precise motion control.

Specifications

- **Holding Torque:** 0.35 Nm at continuous current, 0.5 Nm at peak current.
- **Continuous Output Current:** 1.8 A.
- **Step Angle:** 1.8°.
- **Rotor Inertia:** 57 g·cm².
- **Weight:** 0.37 kg.

Reasons for Selection

1. **Compact Size:** Suitable for applications with limited space, such as 3D printers and small CNC machines.
2. **Moderate Torque:** Provides adequate torque (0.35 Nm continuous, 0.5 Nm peak) for precise motion control and handling moderate loads.
3. **Precise Control:** 1.8° step angle and high subdivision accuracy (1-32 steps) enable fine resolution and accurate positioning.
4. **Versatility:** Compatible with various driver modules and control systems, offering flexibility in design and integration.
5. **Cost-Effectiveness:** Balances performance and cost effectively, making it a practical choice for both industrial and DIY projects.
6. **Reliability:** Known for robust construction and durability, ensuring reliable performance in demanding environments.
7. **Availability:** Widely available from multiple manufacturers, providing easy access to parts and support.

2.6 Stepper Motor Driver

TB6600

The TB6600 stepper motor driver is a versatile choice for controlling stepper motors in various applications. It offers a range of features suitable for precise and powerful motor control, making it ideal for both industrial and DIY projects.

Specifications:

- **Model:** TB6600
- **Compatible Motors:** Nema 17/23/34 (42/57/86 Stepper Motor)
- **Control Signal:** 3.3VDC–24VDC
- **Subdivision Accuracy:** 1–32
- **Output Current:** 4A
- **Voltage:** 9VDC–40VDC

Reasons for Selection:

- **Motor Compatibility:** Compatible with Nema 17, 23, 34 motors.
- **Output Current:** Up to 4A for sufficient torque.
- **Voltage Range:** Supports 9VDC–40VDC for flexible power options.
- **Subdivision Accuracy:** Precision from 1–32 steps.
- **Control Signal:** Compatible with 3.3VDC–24VDC signals.
- **Reliability:** Known for robust performance in industrial applications.

2.7 Power Step-Down

R-78CK-0.5 12V-5V

A 12V-5V power step-down module was used to supply power to the ATMEGA 2560 and other components.

Specifications

- Efficiency up to 96%
- Pin-out compatible with LM78xx linears
- Compact package
- Wide input range (5V - 40V)
- Short circuit protection, thermal shutdown
- Low ripple and noise

Reasons for Selection:

- **Voltage Regulation:** Ensures stable voltage supply to the microcontroller and sensors, which is crucial for reliable operation.
- **Efficiency:** High efficiency in converting 12V input to 5V output, minimizing power loss and heat generation.
- **Compact Size:** The module is compact, making it easy to integrate into the overall design without adding significant bulk.

2.8 Clock Oscillator

16 MHz Crystal Oscillator

A 16 MHz crystal oscillator was used to provide the clock signal for the ATMEGA 2560.

Reasons for Selection:

- **Precision Timing:** Provides a stable and precise clock signal necessary for the accurate operation of the microcontroller.
- **Frequency Stability:** Ensures consistent performance of the microcontroller by maintaining a stable clock frequency.
- **Compatibility:** Specifically chosen to match the operational requirements of the ATMEGA 2560.

2.9 Decoupling Capacitors

100 μ F Decoupling Capacitors

100 μ F decoupling capacitors were used to stabilize the power supply to the microcontroller and other components.

Reasons for Selection:

- **Noise Reduction:** Help to filter out noise and prevent voltage spikes, ensuring a stable power supply.
- **Improved Performance:** Enhances the overall performance and reliability of the microcontroller by maintaining a clean power signal.
- **Protection:** Protects sensitive components from transient voltage fluctuations.

2.10 Pull-up Resistors

2.2k Resistors

2.2k resistors were used in the I2C lines to ensure proper signal levels.

Reasons for Selection:

- **Pull-up Function:** These resistors act as pull-up resistors for the I2C bus, ensuring that the lines are correctly biased and that signals are accurately interpreted.
- **Signal Integrity:** Maintains the integrity of the I2C signals, preventing erroneous data transmission.
- **Standard Value:** 2.2k ohms is a standard value for pull-up resistors in I2C communication, ensuring compatibility and reliable operation.

3 Schematic Design

The schematic design for the overall system was developed using the Altium Designer EDA platform. To ensure an organized and efficient design process, we adopted a hierarchical design methodology, emphasizing modularity and abstraction where we identified four main sub-parts:

1. Microcontroller Circuit (ATmega2560)

Included a 16 MHz crystal oscillator for clocking, decoupling capacitors for noise reduction, and necessary I/O configurations.

2. USB Port Circuit

Designed with the CH340C USB-to-serial converter to facilitate reliable data transfer to a computer.

3. I2C Multiplexer Circuit

The I2C multiplexer circuit is designed to get data from two TOF sensors simultaneously.

4. Power Supply Circuit

Designed to provide a stable 5V supply using the R-78CK-0.5 voltage regulator and necessary filtering capacitors.

This approach allowed for clear separation of different functional blocks, making the design more manageable and easier to troubleshoot. We planned to use Surface Mount Devices. During the design process, we referred to component selection platforms such as Mouser and LCSC. This enabled seamless integration of component footprints directly into our design, ensuring that all selected components were compatible with the SMD process.

4 PCB Design

The PCB layout integrates all schematic designs onto a single board, ensuring optimal component placement and efficient signal routing. Our design process prioritizes robustness, reliability, and ease of integration.

- Trace width: A trace width of 10 mil is chosen for signal traces, while a width of 1 mm is used for power components, ensuring adequate power delivery and signal integrity.
- PCB Size: 1650mil * 2615mil (4.20cm * 6.64cm)
- Component Placement: Components are strategically positioned to minimize trace lengths and optimize signal paths. Critical components such as the microcontroller and ICs are centrally located to enhance stability and performance. Decoupling capacitors are placed close to the ICs to filter out noise and provide a stable power supply.
- Use of vias: Vias of sufficient diameter are used to transition traces between the top and bottom layers of the PCB.

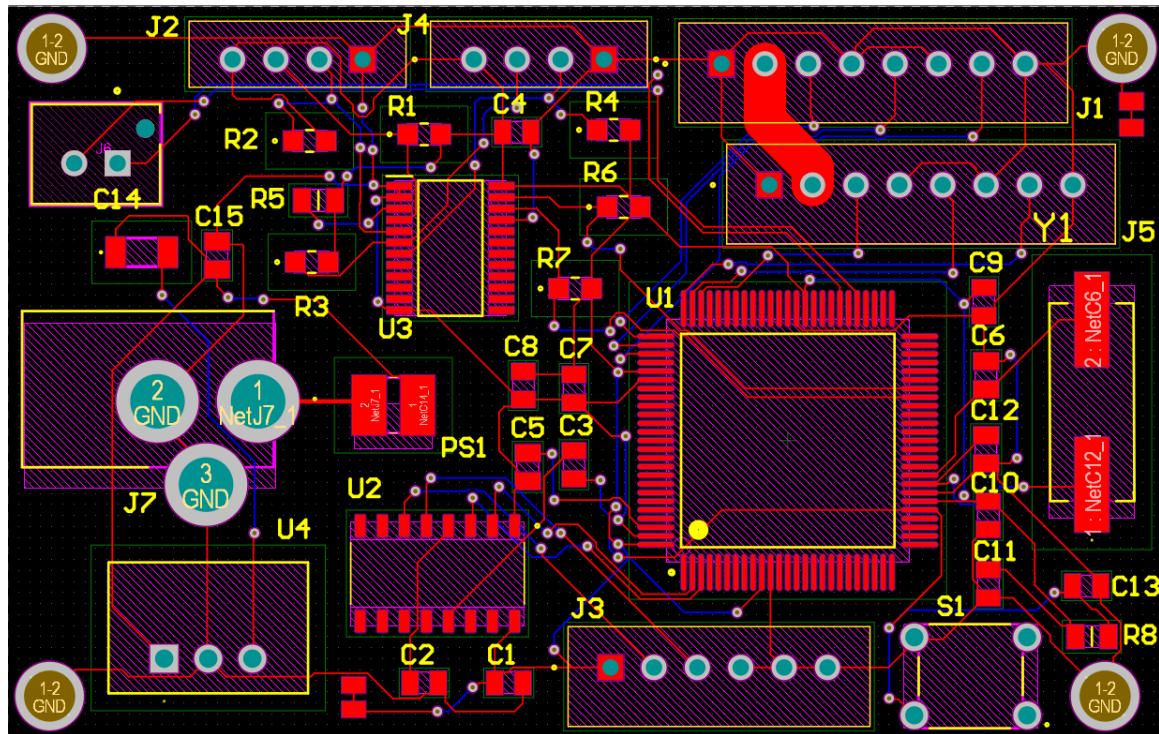


Figure 1: PCB

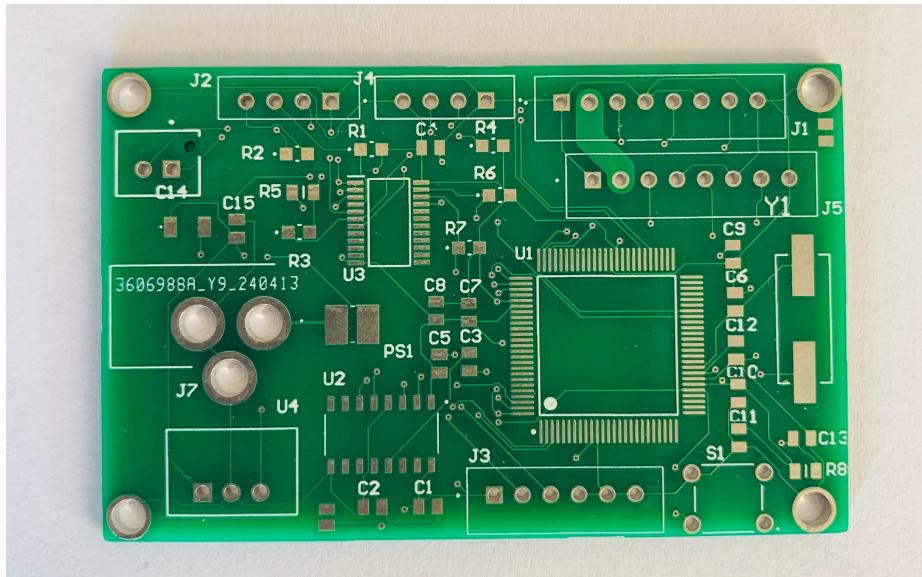


Figure 2: bare PCB

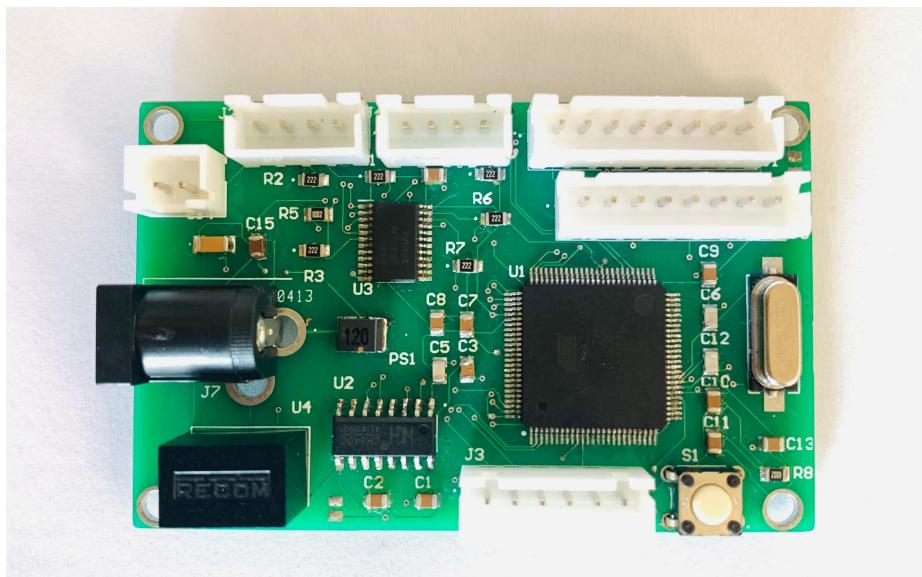


Figure 3: soldered PCB

5 3D Model of Enclosure Design

The 3D model for the enclosure of the Vibration Damping System is designed using SolidWorks 2020, a common and parametric 3D CAD tool known for its robust product data management capabilities. This software allows for seamless updates to the design, ensuring that modifications can be made without compromising the initial specifications.

5.1 CAD Tool: SolidWorks 2020

Description: SolidWorks 2020 is utilized for designing the enclosure due to its advanced parametric modeling features and comprehensive product data management. This choice allows for efficient design iterations and ensures that any changes can be easily managed and tracked.

Advantages:

- Parametric Design: Facilitates easy modifications by adjusting design parameters.
- Product Data Management: Ensures consistency and traceability of design changes, maintaining the integrity of the original specifications.

5.2 Material: PLA+

Description: The enclosure is made from PLA+ (Polylactic Acid Plus), a durable and eco-friendly plastic known for its enhanced properties compared to standard PLA. It provides the necessary strength while being lightweight.

Specifications:

1. Weight: The initial enclosure weighs 50g.
2. Draft Angles: The enclosure features 1-degree draft angles to facilitate easy removal from molds during manufacturing.
3. Thickness: The design maintains a minimum thickness of 3 mm, which is greater than the moldable thickness range for PLA+.



Figure 4: Enclosure



Figure 5: Enclosure

6 Setting Up Environment

6.1 Components

ToF Sensor

A Time-of-Flight sensor to measure distance and angles.

UART Communication Interface

UART pins on the ATMEGA 2560 connected to a USB-to-Serial converter (e.g., FTDI or CP2102) to interface with the computer.

Computer

A computer with USB ports to receive data from the microcontroller and perform visualization.

6.2 Software Requirements

Python

Install Python 3.10.1 or later on your computer.

PySerial

A Python library to handle serial communication.

Installation: `pip install pyserial`

Matplotlib

A Python library for creating static, animated, and interactive visualizations.

Installation: `pip install matplotlib`

NumPy

A Python library for numerical operations.

Installation: `pip install numpy`

3D Visualization Libraries

Matplotlib supports 3D plotting via `mpl_toolkits.mplot3d`.

Installation: `pip install matplotlib` (already mentioned)

7 Daily Log Entries

7.1 26 February – 3 March

Research Phase

- Conducted Research: Reviewed extensive literature on Time-of-Flight (ToF) sensors and analyzed existing 3D scanning technologies to understand the current landscape.
- Industry Standards: Assessed prevailing industry standards and cutting-edge technologies for ToF sensors and stepper motor control in 3D scanning applications.
- System Design Decision: Based on gathered insights, decided to focus on developing a 3D scanner integrating ToF sensors, ATmega2560 microcontroller, and stepper motors for precise and efficient 3D scanning.

7.2 4 March – 10 March

Conceptual Design Phase

- Brainstorming Sessions: Organized multiple brainstorming sessions to generate innovative ideas for accurately capturing 3D spatial data using ToF sensors and stepper motors.
- Concept Development: Developed several conceptual designs for the 3D scanner system, evaluating each design for functionality, precision, cost-effectiveness, and ease of implementation.

7.3 11 March – 17 March

Design Evaluation Phase

- Detailed Evaluation: Conducted a comprehensive evaluation of the conceptual designs, considering technical feasibility, accuracy in 3D data acquisition, ease of assembly, serviceability, and overall performance.
- Concept Selection: After rigorous analysis and discussions, selected the most viable design for further development and prototyping.

7.4 18 March – 24 March

Component Selection Phase and Feasibility Check

- Component Analysis: Evaluated various ToF sensors for accurate distance measurement and stepper motors for precise motion control. Reviewed options for ATmega2560 microcontrollers and PCB design components.
- Feasibility Check: Tested the integration of ToF sensors with ATmega2560 using appropriate communication protocols (e.g., I2C) to verify feasibility and performance.
- Equipment Finalization: Finalized component selection through collaborative discussions and technical assessments to ensure they meet the specific requirements of the 3D scanner project.
- Project Plan Development: Developed a detailed project plan outlining milestones, tasks, and a timeline to guide the development and testing phases of the 3D scanner prototype.

7.5 25 March – 31 March

Design Phase

- PCB Design: Designed the PCB for the 3D scanner's data acquisition system, ensuring compatibility with ToF sensors and ATmega2560.
- Prototype Design: Developed designs for the main unit of the 3D scanner and the mounting mechanism for ToF sensors using SolidWorks.

7.6 1 April – 7 April

PCB Finalization

- PCB Finalization: Completed the final design of the PCB tailored for the 3D scanner system and sent it for manufacturing at JLC PCB.

Initial Experiments

- Initial Experiments: Conducted preliminary experiments with ToF sensors to gather distance data and initiated basic integration tests with the ATmega2560 microcontroller.

7.7 15 April – 21 April

Component Arrival & Preparation

- Component Receipt: Received ToF sensors, ATmega2560 microcontroller, and other necessary components for the 3D scanner.
- Quality Inspection: Inspected components for quality assurance and compliance with design specifications.
- Organization: Organized components to streamline the assembly process of the 3D scanner.

7.8 22 April – 28 April

PCB Assembly & Soldering

- Component Placement: Started placing and soldering components on the PCB, focusing on ensuring precise connections for ToF sensors and microcontroller interfacing.
- Inspection: Used a microscope to inspect solder joints, verifying proper connections and functionality of critical components.
- Completion: Finished assembly and soldering of the PCB, followed by thorough cleaning to eliminate flux residues. Conducted final inspections to address any potential issues.

7.9 29 April – 5 May

Testing & Troubleshooting

- Initial Testing: Conducted initial power-up tests for the assembled 3D scanner system to validate basic functionality.
- Functional Testing: Performed detailed tests, including calibration of ToF sensors, validation of stepper motor control, and integration checks with the ATmega2560 microcontroller.
- Communication Verification: Ensured robust communication protocols between the 3D scanner's components, verifying data transmission integrity and system responsiveness.

7.10 6 May – 12 May

Final Adjustments

- Software and Hardware Adjustments: Implemented final refinements to both software algorithms (e.g., distance calculation, point cloud generation) and hardware configurations based on testing outcomes.

Enclosure Finalization

- 3D Printing and Integration: Printed the enclosure for the 3D scanner and integrated it with the assembled PCB and wiring, ensuring secure housing and optimal sensor positioning.
- Assembly: Verified proper fit and mounting of components within the enclosure, optimizing space utilization and ensuring durability in operational environments.

7.11 13 May – 15 June

Final Report Preparation

- Report Compilation: Compiled and finalized comprehensive project reports, including the Design Methodology Document and the Design Details Document. Detailed the entire development process, encountered challenges, and implemented solutions for the 3D scanner.

8 Programming Process

8.1 Distance Measurement using TOF sensor (4 March - 10 March)

8.1.1 Arduino Code

```

1 #include <Wire.h>
2 #include <Adafruit_VL53L0X.h>
3
4 Adafruit_VL53L0X lox = Adafruit_VL53L0X();
5
6 void setup() {
7     Serial.begin(115200);
8
9     if (!lox.begin()) {
10         Serial.println(F("Failed to boot VL53L0X"));
11         while (1);
12     }
13
14     Serial.println(F("VL53L0X test"));
15 }
16
17 void loop() {
18     VL53L0X_RangingMeasurementData_t measure;
19
20     lox.rangingTest(&measure, false);
21
22     if (measure.RangeStatus != 4) { // phase failures have incorrect data
23         Serial.print(F("Distance (mm): "));
24         Serial.println(measure.RangeMilliMeter);
25     } else {
26         Serial.println(F("Out of range"));
27     }
28
29     delay(100);
30 }
```

Listing 1: Arduino Code for VL53L0X Sensor

8.1.2 Code Explanation

Include Libraries

- `#include <Wire.h>`
 - `Wire.h`: Library to communicate with I2C devices.
- `#include <Adafruit_VL53L0X.h>`
 - `Adafruit_VL53L0X.h`: Library specific to the Adafruit VL53L0X sensor.

Create Sensor Object

- `Adafruit_VL53L0X lox = Adafruit_VL53L0X();`
 - Creates an instance of the VL53L0X sensor named `lox`.

Setup Function

```

1 void setup() {
2   Serial.begin(115200);
3
4   if (!lox.begin()) {
5     Serial.println(F("Failed to boot VL53L0X"));
6     while (1);
7   }
8
9   Serial.println(F("VL53L0X test"));
10 }
```

- Initializes serial communication at 115200 baud.
- Tries to initialize the sensor using `lox.begin()`.
- If the sensor fails to initialize, it prints an error message and enters an infinite loop.
- If initialization is successful, it prints a success message.

Loop Function

```

1 void loop() {
2   VL53L0X_RangingMeasurementData_t measure;
3
4   lox.rangingTest(&measure, false);
5
6   if (measure.RangeStatus != 4) {
7     Serial.print(F("Distance (mm): "));
8     Serial.println(measure.RangeMilliMeter);
9   } else {
10   Serial.println(F("Out of range"));
11 }
12
13 delay(100);
14 }
```

- Declares a variable `measure` of type `VL53L0X_RangingMeasurementData_t` to store the measurement data.
- Calls `lox.rangingTest(&measure, false)` to perform a ranging test and store the result in `measure`.
- Checks the `RangeStatus` to determine if the measurement is valid (status not equal to 4).
 - If valid, prints the measured distance in millimeters.
 - If not valid (out of range), prints an error message.
- Delays for 100 milliseconds before taking another measurement.

8.2 Stepper Motor Check (11 March - 17 March)

8.2.1 Arduino Code

```

1 #include <AccelStepper.h>
2 #include <Wire.h>
3 #include <Adafruit_VL53L0X.h>
4
5 // Define stepper motor connections
6 #define STEP_PIN 10
7 #define DIR_PIN 11
8
9 #define STEP_PIN_2 12
10 #define DIR_PIN_2 13
11
12 #define STEPS_PER_REVOLUTION 200
13 #define ANGLE_INCREMENT 1.8
14
15 // Create a stepper object
16 AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);
17 AccelStepper stepper2(AccelStepper::DRIVER, STEP_PIN_2, DIR_PIN_2);
18
19 void setup() {
20     // Set up the speed and acceleration of the stepper motor
21     stepper.setMaxSpeed(1000);
22     stepper.setAcceleration(500);
23
24     stepper2.setMaxSpeed(1000);
25     stepper2.setAcceleration(500);
26
27     // Set the motor to run continuously
28     stepper.moveTo(0);
29     stepper2.moveTo(0);
30 }
31
32 void loop() {
33     // Run the stepper motor
34     stepper.run();
35     stepper2.run();
36 }
```

Listing 2: Arduino Code for Two Stepper Motors with AccelStepper

8.2.2 Code Explanation

Include Libraries

- `#include <AccelStepper.h>`
 - `AccelStepper.h`: Library for controlling stepper motors.
- `#include <Wire.h>`
 - `Wire.h`: Library to communicate with I2C devices.
- `#include <Adafruit_VL53L0X.h>`
 - `Adafruit_VL53L0X.h`: Library specific to the Adafruit VL53L0X sensor.

Define Stepper Motor Connections

```

1 \item \texttt{\#define STEP\_PIN 10}
2 \item \texttt{\#define DIR\_PIN 11}
```

- Defines the step and direction pins for the first stepper motor.

```

1 \item \texttt{\#define STEP\_PIN\_2 12}
2 \item \texttt{\#define DIR\_PIN\_2 13}
```

- Defines the step and direction pins for the second stepper motor.

```

1 \item \texttt{\#define STEPS\_PER\_REVOLUTION 200}
2 \item \texttt{\#define ANGLE\_INCREMENT 1.8}
```

- Defines the number of steps per revolution and the angle increment.

Create Stepper Objects

```

1 AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);
```

- Creates an instance of the `AccelStepper` class for the first stepper motor.

```

1 AccelStepper stepper2(AccelStepper::DRIVER, STEP_PIN\_2, DIR_PIN\_2);
```

- Creates an instance of the `AccelStepper` class for the second stepper motor.

Setup Function

```

1 void setup() {
2   stepper.setMaxSpeed(1000);
3   stepper.setAcceleration(500);
4
5   stepper2.setMaxSpeed(1000);
6   stepper2.setAcceleration(500);
7
8   stepper.moveTo(0);
9   stepper2.moveTo(0);
10 }
```

- `void setup()`
 - Initializes the setup function.
- `stepper.setMaxSpeed(1000);`
 - Sets the maximum speed for the first stepper motor.
- `stepper.setAcceleration(500);`
 - Sets the acceleration for the first stepper motor.
- `stepper2.setMaxSpeed(1000);`
 - Sets the maximum speed for the second stepper motor.

- `stepper2.setAcceleration(500);`
 - Sets the acceleration for the second stepper motor.
- `stepper.moveTo(0);`
 - Moves the first stepper motor to position 0.
- `stepper2.moveTo(0);`
 - Moves the second stepper motor to position 0.

Loop Function

```
1 void loop() {  
2     stepper.run();  
3     stepper2.run();  
4 }
```

- `void loop()`
 - Initializes the loop function.
- `stepper.run();`
 - Runs the first stepper motor.
- `stepper2.run();`
 - Runs the second stepper motor.

8.3 3D Cloud Generation (18 March - 24 March)

8.3.1 Python Code

```

1 import ast
2 import numpy as np
3 import pyvista as pv
4
5 # Open the file in read mode
6 with open('example.txt', 'r') as file:
7     # Read the file content
8     content = file.read()
9
10 # Use ast.literal_eval to convert the string to a 2D list
11 points = np.array(ast.literal_eval(content))
12
13 # Create a PolyData object
14 cloud = pv.PolyData(points)
15
16 # Plot the point cloud
17 cloud.plot()
18
19 # Create a 3D Delaunay triangulation of the point cloud
20 volume = cloud.delaunay_3d(alpha=0.6)
21
22 # Extract the outer surface of the volume
23 shell = volume.extract_geometry()
24
25 # Plot the resulting mesh
26 shell.plot()

```

Listing 3: Python Code for 3D Delaunay Triangulation

8.3.2 Code Explanation

8.3.3 Import Libraries

- `import ast`
 - `ast`: Library for safely evaluating strings containing Python expressions.
- `import numpy as np`
 - `numpy`: Library for numerical operations on arrays.
- `import pyvista as pv`
 - `pyvista`: Library for 3D plotting and mesh analysis.

Open and Read File

```

1 with open('example.txt', 'r') as file:
2     content = file.read()

```

- Opens the file '`example.txt`' in read mode and reads its content.

Convert String to 2D List

```

1 points = np.array(ast.literal_eval(content))

```

- Safely evaluates the string content and convert it into a 2D list.
- Converts the 2D list to a NumPy array for further processing.

Create PolyData Object

```
1 cloud = pv.PolyData(points)
```

- Creates a PolyData object from the points array using PyVista.

Plot the Point Cloud

```
1 cloud.plot()
```

- Plots the point cloud using PyVista.

Create 3D Delaunay Triangulation

```
1 volume = cloud.delaunay_3d(alpha=0.6)
```

- Creates a 3D Delaunay triangulation of the point cloud with an alpha value of 0.6.

Extract Outer Surface

```
1 shell = volume.extract_geometry()
```

- Extracts the outer surface of the triangulated volume.

Plot the Resulting Mesh

```
1 shell.plot()
```

- Plots the resulting mesh of the outer surface using PyVista.

8.4 2D Scan (25 March - 31 March)

8.4.1 Arduino Code

```

1 #include <Wire.h>
2 #include <Adafruit_VL53L0X.h>
3 #include <Arduino.h>
4 #include "A4988.h"
5
6 #define STEPS_PER_REVOLUTION 200
7 #define ANGLE_INCREMENT 1.8
8
9 int Step = 10; //GPIO14---D5 of Nodemcu--Step of stepper motor driver
10 int Dire = 11; //GPIO2---D4 of Nodemcu--Direction of stepper motor driver
11 int Sleep = 14; //GPIO12---D6 of Nodemcu--Control Sleep Mode on A4988
12 int MS1 = 13; //GPIO13---D7 of Nodemcu--MS1 for A4988
13 int MS2 = 16; //GPIO16---D0 of Nodemcu--MS2 for A4988
14 int MS3 = 15; //GPIO15---D8 of Nodemcu--MS3 for A4988
15
16 //Motor Specs
17 const int spr = 200; //Steps per revolution
18 int RPM = 100; //Motor Speed in revolutions per minute
19 int Microsteps = 1; //Stepsize (1 for full steps, 2 for half steps, 4 for quarter
                     steps, etc)
20
21 //Providing parameters for motor control
22 A4988 stepper(spr, Dire, Step, MS1, MS2, MS3);
23
24 Adafruit_VL53L0X lox = Adafruit_VL53L0X();
25 //Stepper stepper(STEPS_PER_REVOLUTION, 2, 3, 4, 5); // Adjust pin numbers
               accordingly
26
27 const int num_measurements = STEPS_PER_REVOLUTION;
28 float measurements[num_measurements][2];
29
30 String arrayToString(int arr[], int size) {
31     String result = "{";
32     for (int i = 0; i < size; i++) {
33         result += String(arr[i]);
34         if (i < size - 1) {
35             result += ", ";
36         }
37     }
38     result += "}";
39     return result;
40 }
41
42 void setup() {
43     Serial.begin(9600);
44     Serial1.begin(9600); // Use Serial1 for communication on Arduino Mega
45
46     pinMode(Step, OUTPUT); //Step pin as output
47     pinMode(Dire, OUTPUT); //Direction pin as output
48     pinMode(Sleep, OUTPUT); //Set Sleep OUTPUT Control button as output
49     digitalWrite(Step, LOW); // Currently no stepper motor movement
50     digitalWrite(Dire, LOW);
51
52     // Set target motor RPM to and microstepping setting
53     //stepper.begin(RPM, Microsteps);
54
55     if (!lox.begin()) {
56         Serial.println(F("Failed to boot VL53L0X"));
57         while (1);
58     }
59
60     pinMode(13, OUTPUT); // Set pin 13 as an output
61
62 //    stepper.setSpeed(500); // Adjust the speed as needed

```

```

63     Serial.println(F("VL53L0X test with Stepper Motor"));
64 }
65
66 void loop() {
67     digitalWrite(12, HIGH);
68     digitalWrite(Sleep, HIGH); //A logic high allows normal operation of the A4988 by
69         // removing from sleep
70     stepper.rotate(360);
71
72     for (int i = 0; i < num_measurements; i++) {
73         // Rotate stepper motor by ANGLE_INCREMENT degrees
74         // stepper.step(ANGLE_INCREMENT);
75
76         // Take distance measurement
77         VL53L0X_RangingMeasurementData_t measure;
78         lox.rangingTest(&measure, false);
79
80         if (measure.RangeStatus != 4) { // phase failures have incorrect data
81             measurements[i][0] = i * ANGLE_INCREMENT;
82             measurements[i][1] = measure.RangeMilliMeter;
83         } else {
84             measurements[i][0] = i * ANGLE_INCREMENT;
85             measurements[i][1] = 10000;
86         }
87
88         // No delay or minimal delay between steps
89     }
90
91     // Print the 2D array after 360 degrees rotation
92     Serial.print("aaa[");
93     for (int i = 0; i < num_measurements; i++) {
94         Serial.print("[");
95         Serial.print(measurements[i][0]);
96         Serial.print(",");
97         Serial.print(measurements[i][1]);
98         Serial.print("],");
99     }
100    Serial.println("]");
101    //Serial.println(arrayToString());
102
103    // Reset the stepper motor to its initial position
104    // stepper.step(-360 * STEPS_PER_REVOLUTION / 360);
105    digitalWrite(13, LOW);
106
107    // Delay before starting a new measurement
108    delay(5000);
109}

```

Listing 4: Arduino Code for VL53L0X Sensor with Stepper Motors and A4988 Driver

8.4.2 Code Explanation

Import Libraries

- `#include <Wire.h>`
 - Provides I2C communication.
- `#include <Adafruit_VL53L0X.h>`
 - Provides functions for the VL53L0X sensor.
- `#include <Arduino.h>`
 - Provides core Arduino functions.

- `#include "A4988.h"`
 - Provides functions for controlling the A4988 stepper motor driver.

Define Constants and Variables

- `#define STEPS_PER_REVOLUTION 200`
 - Defines the number of steps per revolution for the stepper motor.
- `#define ANGLE_INCREMENT 1.8`
 - Defines the angle increment for each step.
- `int Step = 10;`
 - Defines the GPIO pin for the step signal.
- `int Dire = 11;`
 - Defines the GPIO pin for the direction signal.
- `int Sleep = 14;`
 - Defines the GPIO pin for the sleep signal.
- `int MS1 = 13;`
 - Defines the GPIO pin for the MS1 signal.
- `int MS2 = 16;`
 - Defines the GPIO pin for the MS2 signal.
- `int MS3 = 15;`
 - Defines the GPIO pin for the MS3 signal.

Motor Specifications and Setup

- `const int spr = 200;`
 - Sets the steps per revolution for the motor.
- `int RPM = 100;`
 - Sets the motor speed in revolutions per minute.
- `int Microsteps = 1;`
 - Sets the microstepping mode.
- `A4988 stepper(spr, Dire, Step, MS1, MS2, MS3);`
 - Initializes the A4988 stepper motor driver.
- `Adafruit_VL53L0X lux = Adafruit_VL53L0X();`
 - Initializes the VL53L0X sensor.

Setup Function

```

1 void setup() {
2   Serial.begin(9600);
3   Serial1.begin(9600); // Use Serial1 for communication on Arduino Mega
4
5   pinMode(Step, OUTPUT); //Step pin as output
6   pinMode(Dire, OUTPUT); //Direction pin as output
7   pinMode(Sleep, OUTPUT); //Set Sleep OUTPUT Control button as output
8   digitalWrite(Step, LOW); // Currently no stepper motor movement
9   digitalWrite(Dire, LOW);
10
11 // Set target motor RPM and microstepping setting
12 //stepper.begin(RPM, Microsteps);
13
14 if (!lox.begin()) {
15   Serial.println(F("Failed to boot VL53L0X"));
16   while (1);
17 }
18
19 pinMode(13, OUTPUT); // Set pin 13 as an output
20
21 // stepper.setSpeed(500); // Adjust the speed as needed
22
23 Serial.println(F("VL53L0X test with Stepper Motor"));
24 }
```

- Initializes serial communication.
- Sets pin modes for the stepper motor driver.
- Initializes the VL53L0X sensor.
- Prints a test message to the serial monitor.

Main Loop

```

1 void loop() {
2   digitalWrite(12, HIGH);
3   digitalWrite(Sleep, HIGH); //A logic high allows normal operation of the A4988 by
4   // removing from sleep
5   stepper.rotate(360);
6
7   for (int i = 0; i < num_measurements; i++) {
8     // Rotate stepper motor by ANGLE_INCREMENT degrees
9     // stepper.step(ANGLE_INCREMENT);
10
11    // Take distance measurement
12    VL53L0X_RangingMeasurementData_t measure;
13    lox.rangingTest(&measure, false);
14
15    if (measure.RangeStatus != 4) { // phase failures have incorrect data
16      measurements[i][0] = i * ANGLE_INCREMENT;
17      measurements[i][1] = measure.RangeMilliMeter;
18    } else {
19      measurements[i][0] = i * ANGLE_INCREMENT;
20      measurements[i][1] = 10000;
21    }
22
23    // No delay or minimal delay between steps
24  }
25
26  // Print the 2D array after 360 degrees rotation
27  Serial.print("aaa[");
28  for (int i = 0; i < num_measurements; i++) {
29    Serial.print(measurements[i][0]);
30    Serial.print(",");
31    Serial.print(measurements[i][1]);
32    Serial.print(" ");
33  }
34  Serial.println("]");
35}
```

```
28     Serial.print("[");
29     Serial.print(measurements[i][0]);
30     Serial.print(",");
31     Serial.print(measurements[i][1]);
32     Serial.print("],\"");
33 }
34 Serial.println("]");
35 //Serial.println(arrayToString());
36
37 // Reset the stepper motor to its initial position
38 // stepper.step(-360 * STEPS_PER_REVOLUTION / 360);
39 digitalWrite(13, LOW);
40
41 // Delay before starting a new measurement
42 delay(5000);
43 }
```

- Activates the A4988 driver.
- Rotates the stepper motor 360 degrees.
- Takes distance measurements with the VL53L0X sensor at each step.
- Stores measurements in a 2D array.
- Prints the measurements to the serial monitor in a JSON-like format.
- Resets the stepper motor.
- Delays before starting a new measurement cycle.

8.5 3D scan (1 April - 7 April)

8.5.1 Arduino Code

```

1 #include <Wire.h>
2 #include <Adafruit_VL53L0X.h>
3
4 #define STEP_PIN_1 8
5 #define DIR_PIN_1 9
6 #define ENA_PIN_1 10
7
8 #define STEP_PIN_2 4
9 #define DIR_PIN_2 3
10 #define ENA_PIN_2 2
11
12 #define XY_REV 200
13 #define XZ_REV 50
14
15 #define ANGLE_INCREMENT 1.8
16
17 Adafruit_VL53L0X lox = Adafruit_VL53L0X();
18
19 float measurements[XZ_REV][3];
20
21 void setup() {
22     Serial.begin(9600); // Initialize the serial monitor
23
24     if (!lox.begin()) {
25         Serial.println(F("Failed to boot VL53L0X"));
26         while (1);
27     }
28     pinMode(4,OUTPUT);
29     pinMode(3,OUTPUT);
30     pinMode(2,OUTPUT);
31
32     pinMode(7,OUTPUT);
33     pinMode(8,OUTPUT);
34     pinMode(9,OUTPUT);
35
36     digitalWrite(ENA_PIN_1, LOW);
37     digitalWrite(ENA_PIN_2, LOW);
38
39     Serial.println(F("VL53L0X test with Stepper Motor"));
40 }
41
42 void loop() {
43     // put your main code here, to run repeatedly:
44     for (int i=0; i < XY_REV; i++) {
45         if (i%2 == 0) {
46             digitalWrite(DIR_PIN_2,HIGH);
47         } else {
48             digitalWrite(DIR_PIN_2,LOW);
49         }
50         for (int j=0; j < XZ_REV; j++) {
51             VL53L0X_RangingMeasurementData_t measure;
52             lox.rangingTest(&measure, false);
53
54             if (i%2 == 0) {
55                 if (measure.RangeStatus != 4) { // phase failures have incorrect data
56                     measurements[j][0] = j * ANGLE_INCREMENT - 45;
57                     measurements[j][1] = i * ANGLE_INCREMENT;
58                     measurements[j][2] = measure.RangeMilliMeter;
59                 } else {
56                     measurements[j][0] = j * ANGLE_INCREMENT - 45;
57                     measurements[j][1] = i * ANGLE_INCREMENT;
58                     measurements[j][2] = 10000;
59                 }
60             } else {
61                 measurements[j][0] = j * ANGLE_INCREMENT - 45;
62                 measurements[j][1] = i * ANGLE_INCREMENT;
63                 measurements[j][2] = 10000;
64             }
65         }
66     }
67 }
```

```

65     if (measure.RangeStatus != 4) { // phase failures have incorrect data
66     measurements[XZ_REV - j - 1][0] = j * ANGLE_INCREMENT - 45;
67     measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
68     measurements[XZ_REV - j - 1][2] = measure.RangeMilliMeter;
69   } else {
70     measurements[XZ_REV - j - 1][0] = j * ANGLE_INCREMENT - 45;
71     measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
72     measurements[XZ_REV - j - 1][2] = 10000;
73   }
74 }
75   digitalWrite(STEP_PIN_2, HIGH);
76   digitalWrite(STEP_PIN_2, LOW);
77 }
78 Serial.print("aaa[");
79 for (int k = 0; k < XZ_REV; k++) {
80   Serial.print("[");
81   Serial.print(measurements[k][0]);
82   Serial.print(",");
83   Serial.print(measurements[k][1]);
84   Serial.print("],");
85   Serial.print(measurements[k][2]);
86   Serial.print("],");
87 }
88 Serial.println("]");
89 digitalWrite(STEP_PIN_1, HIGH);
90 digitalWrite(STEP_PIN_1, LOW);
91 }
92 }

```

Listing 5: Arduino Code for VL53L0X Sensor with Stepper Motors

8.5.2 Code Explanation

Import Libraries

- `#include <Wire.h>`
 - Provides I2C communication.
- `#include <Adafruit_VL53L0X.h>`
 - Provides functions for the VL53L0X sensor.

Define Constants and Variables

- `#define STEP_PIN_1 8`
 - Defines the GPIO pin for the step signal of the first stepper motor.
- `#define DIR_PIN_1 9`
 - Defines the GPIO pin for the direction signal of the first stepper motor.
- `#define ENA_PIN_1 10`
 - Defines the GPIO pin for the enable signal of the first stepper motor.
- `#define STEP_PIN_2 4`
 - Defines the GPIO pin for the step signal of the second stepper motor.
- `#define DIR_PIN_2 3`
 - Defines the GPIO pin for the direction signal of the second stepper motor.

- `#define ENA_PIN_2 2`
 - Defines the GPIO pin for the enable signal of the second stepper motor.
- `#define XY_REV 200`
 - Defines the number of steps per revolution for the XY plane.
- `#define XZ_REV 50`
 - Defines the number of steps per revolution for the XZ plane.
- `#define ANGLE_INCREMENT 1.8`
 - Defines the angle increment for each step.
- `Adafruit_VL53L0X lox = Adafruit_VL53L0X();`
 - Initializes the VL53L0X sensor.
- `float measurements[XZ_REV] [3];`
 - Initializes a 2D array to store the measurement data.

Setup Function

```

1 void setup() {
2   Serial.begin(9600); // Initialize the serial monitor
3
4   if (!lox.begin()) {
5     Serial.println(F("Failed to boot VL53L0X"));
6     while (1);
7   }
8   pinMode(4, OUTPUT);
9   pinMode(3, OUTPUT);
10  pinMode(2, OUTPUT);
11
12  pinMode(7, OUTPUT);
13  pinMode(8, OUTPUT);
14  pinMode(9, OUTPUT);
15
16  digitalWrite(ENA_PIN_1, LOW);
17  digitalWrite(ENA_PIN_2, LOW);
18
19  Serial.println(F("VL53L0X test with Stepper Motor"));
20 }
```

- Initializes serial communication.
- Initializes the VL53L0X sensor.
- Sets pin modes for the stepper motor driver.
- Enables the stepper motor drivers.
- Prints a test message to the serial monitor.

Main Loop

```

1 void loop() {
2     // put your main code here, to run repeatedly:
3     for (int i=0; i < XY_REV; i++) {
4         if (i%2 == 0) {
5             digitalWrite(DIR_PIN_2,HIGH);
6         } else {
7             digitalWrite(DIR_PIN_2,LOW);
8         }
9         for (int j=0; j < XZ_REV; j++) {
10            VL53L0X_RangingMeasurementData_t measure;
11            lox.rangingTest(&measure, false);
12
13            if (i%2 == 0) {
14                if (measure.RangeStatus != 4) { // phase failures have incorrect data
15                    measurements[j][0] = j * ANGLE_INCREMENT - 45;
16                    measurements[j][1] = i * ANGLE_INCREMENT;
17                    measurements[j][2] = measure.RangeMilliMeter;
18                } else {
19                    measurements[j][0] = j * ANGLE_INCREMENT - 45;
20                    measurements[j][1] = i * ANGLE_INCREMENT;
21                    measurements[j][2] = 10000;
22                }
23            } else {
24                if (measure.RangeStatus != 4) { // phase failures have incorrect data
25                    measurements[XZ_REV - j -1][0] = j * ANGLE_INCREMENT - 45;
26                    measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
27                    measurements[XZ_REV - j - 1][2] = measure.RangeMilliMeter;
28                } else {
29                    measurements[XZ_REV - j - 1][0] = j * ANGLE_INCREMENT - 45;
30                    measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
31                    measurements[XZ_REV - j - 1][2] = 10000;
32                }
33            }
34            digitalWrite(STEP_PIN_2, HIGH);
35            digitalWrite(STEP_PIN_2, LOW);
36        }
37        Serial.print("aaa[");
38        for (int k = 0; k < XZ_REV; k++) {
39            Serial.print("[");
40            Serial.print(measurements[k][0]);
41            Serial.print(",");
42            Serial.print(measurements[k][1]);
43            Serial.print(",");
44            Serial.print(measurements[k][2]);
45            Serial.print("],");
46        }
47        Serial.println("]");
48        digitalWrite(STEP_PIN_1, HIGH);
49        digitalWrite(STEP_PIN_1, LOW);
50    }
51 }

```

- Runs the main loop repeatedly.
- Controls the direction of the second stepper motor.
- Takes distance measurements with the VL53L0X sensor at each step.
- Stores measurements in a 2D array.
- Prints the measurements to the serial monitor in a JSON-like format.
- Steps the first stepper motor.

8.6 Serial Communication - Receiver (22 April - 28 April)

8.6.1 Python Code

```

1 import serial
2 import json
3 from math import cos, sin, pi
4 import matplotlib.pyplot as plt
5 import numpy as np
6 from mpl_toolkits.mplot3d import Axes3D
7
8 from datetime import datetime
9
10 # datetime object containing current date and time
11 now = datetime.now()
12
13 print("now =", now)
14
15 # dd/mm/YY H:M:S
16 dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
17
18 printed = False
19
20 def plot_lines(coordinates):
21     x_values, y_values = zip(*coordinates)
22     plt.plot(x_values, y_values, color='blue', linestyle='-', linewidth=2,
23               label='Line')
24
25     plt.title('Graph with Connected Line (No Dots)')
26     plt.xlabel('X-axis')
27     plt.ylabel('Y-axis')
28
29     # Set axis limits to always be 0 to 300
30     #plt.xlim(0, 300)
31     #plt.ylim(0, 300)
32
33     # Set the aspect ratio to be equal
34     plt.gca().set_aspect('equal', adjustable='box')
35
36     plt.grid(True)
37     plt.legend()
38     plt.show()
39
40 def plot_dots(coordinates):
41     x_values, y_values = zip(*coordinates)
42     plt.scatter(x_values, y_values, color='blue', marker='o')
43     plt.title('Graph with Dots')
44     plt.xlabel('X-axis')
45     plt.ylabel('Y-axis')
46
47     #plt.xlim(0, 300)
48     #plt.ylim(0, 300)
49
50     plt.gca().set_aspect('equal', adjustable='box')
51
52     plt.grid(True)
53     plt.show()
54
55 def plot_3d_surface(coordinates):
56     x_coords = [coordinate[0] for coordinate in coordinates]
57     y_coords = [coordinate[1] for coordinate in coordinates]
58     z_coords = [coordinate[2] for coordinate in coordinates]
59
60     # Convert coordinates to numpy arrays
61     x = np.array(x_coords)
62     y = np.array(y_coords)
63     z = np.array(z_coords)

```

```

64
65     # Create a 3D plot
66     fig = plt.figure()
67     ax = fig.add_subplot(111, projection='3d')
68
69     # Plot the scatter plot
70     ax.scatter(x, y, z, c='b', marker='o', s = 1)
71
72     # Connect the points with lines
73     for i in range(1, len(x)):
74         ax.plot([x[i-1], x[i]], [y[i-1], y[i]], [z[i-1], z[i]], c='b')
75
76     ax.set_xlabel('x')
77     ax.set_ylabel('y')
78     ax.set_zlabel('z')
79
80     plt.show()
81
82 def plot_3d_wireframe(coordinates):
83     x_coords = [coordinate[0] for coordinate in coordinates]
84     y_coords = [coordinate[1] for coordinate in coordinates]
85     z_coords = [coordinate[2] for coordinate in coordinates]
86
87     # Convert coordinates to numpy arrays
88     x = np.array(x_coords)
89     y = np.array(y_coords)
90     z = np.array(z_coords)
91
92     # Create a 3D plot
93     fig = plt.figure()
94     ax = fig.add_subplot(111, projection='3d')
95
96     # Plot the wireframe
97     ax.plot_trisurf(x, y, z, linewidth=0, antialiased=False)
98
99     ax.set_xlabel('x')
100    ax.set_ylabel('y')
101    ax.set_zlabel('z')
102
103    plt.show()
104
105
106 def str2list(input_string):
107     input_string = input_string.replace('\r', "")
108     input_string = input_string.replace('\n', "")
109
110     temp_buffer = []
111     temp_text = ''
112     selected_dots = []
113
114     for stringData in input_string.split('\n'):
115         temp_buffer.append(temp_text + stringData.split('\n')[0])
116         temp_text = ''
117
118         if temp_buffer[-1][:3] == "aaa":
119             temp_text = ''
120             build_temp = temp_buffer[-1][3:].rstrip('\n\r,')
121
122             # Handle the case when the string ends with a trailing comma
123             if build_temp[-2]:
124                 build_temp = build_temp[:-2]+build_temp[-1]
125
126             try:
127                 nested_list = [list(map(float, innerList)) for innerList in
128                               json.loads(build_temp)]]
129             except json.decoder.JSONDecodeError as e:
130                 print("Error decoding JSON:", e)
131                 print("Problematic data:", build_temp)

```

```

131     return False
132
133     for nested_item in nested_list:
134         x_temp = ((nested_item[2] * cos(nested_item[0] * (pi / 180)) *
135                     cos(nested_item[1] * (pi / 180))) + 2000) /10
136         y_temp = ((nested_item[2] * cos(nested_item[0] * (pi / 180)) *
137                     sin(nested_item[1] * (pi / 180))) + 2000) /10
138         z_temp = ((nested_item[2] * sin(nested_item[0] * (pi / 180))) +
139                     2000) /10
140
141         if x_temp < 1000 and y_temp < 1000:
142             selected_dots.append([x_temp, y_temp, z_temp])
143             #print("Selected dots:", selected_dots)
144
145     return selected_dots
146
147     return False
148
149
150
151 class ReadLine:
152     def __init__(self, s):
153         self.buf = bytearray()
154         self.s = s
155
156     def readline(self):
157         i = self.buf.find(b"\n")
158         if i >= 0:
159             r = self.buf[:i+1]
160             self.buf = self.buf[i+1:]
161             return r.decode("utf-8")
162
163         while True:
164             i = max(1, min(2048, self.s.in_waiting))
165             data = self.s.read(i)
166             i = data.find(b"\n")
167             if i >= 0:
168                 r = self.buf + data[:i+1]
169                 self.buf[0:] = data[i+1:]
170                 return r.decode("utf-8")
171             else:
172                 self.buf.extend(data)
173
174
175 ser = serial.Serial('COM3', 9600)
176 rl = ReadLine(ser)
177
178 can_update = False
179 list_for_3d = []
180
181 while not printed:
182     line = rl.readline()
183     #print("Received:", line)
184     print_temp = str2list(line)
185     if print_temp != False:
186         #print(print_temp)
187         if (int(print_temp[0][1]) == 0 and int(print_temp[0][0]) == 0):
188             can_update = True
189         if True:
190             for i in print_temp:
191                 list_for_3d.append(i)
192             print(len(list_for_3d)/50)
193             if len(list_for_3d) >= 200*50:
194                 #plot_dots(print_temp)
195                 #plot_lines(print_temp)
196                 #print(len(list_for_3d))
197                 # Specify the file path
198                 file_path = "example" + dt_string + ".txt"
199
200                 # Open the file in write mode
201                 with open(file_path, 'w') as file:

```

```

196         # Write the text to the file
197         file.write(str(list_for_3d))
198
199     print(f"Text saved to {file_path}")
200     #print(list_for_3d)
201     plot_3d_surface(list_for_3d)
202     printed = True

```

Listing 6: Python Code for receiver

8.6.2 Code Explanation

Import Libraries

- `import serial`
 - Provides serial communication capabilities.
- `import json`
 - Used for parsing JSON data.
- `from math import cos, sin, pi`
 - Provides trigonometric functions and the value of π .
- `import matplotlib.pyplot as plt`
 - Used for plotting graphs.
- `import numpy as np`
 - Provides support for large, multi-dimensional arrays and matrices.
- `from mpl_toolkits.mplot3d import Axes3D`
 - Enables 3D plotting.
- `from datetime import datetime`
 - Provides date and time functionalities.

Datetime Initialization

```

1 now = datetime.now()
2 dt_string = now.strftime("%d/%m/%Y %H:%M:%S")

```

Initializes the current date and time and formats it as a string.

Plotting Functions

Defines functions for plotting different types of graphs:

- `plot_lines(coordinates)`: Plots lines connecting the coordinates.
- `plot_dots(coordinates)`: Plots dots at the coordinates.
- `plot_3d_surface(coordinates)`: Plots a 3D surface.
- `plot_3d_wireframe(coordinates)`: Plots a 3D wireframe.

String to List Conversion

```

1 def str2list(input_string):
2     ...

```

Converts the input string from the serial data into a list of coordinates.

ReadLine Class

```

1 class ReadLine:
2     def __init__(self, s):
3         self.buf = bytearray()
4         self.s = s
5
6     def readline(self):
7         ...

```

Class for reading lines from the serial port.

Main Code Block

```

1 ser = serial.Serial('COM3', 9600)
2 rl = ReadLine(ser)
3 can_update = False
4 list_for_3d = []
5
6 while not printed:
7     line = rl.readline()
8     print_temp = str2list(line)
9     if print_temp != False:
10        if (int(print_temp[0][1]) == 0 and int(print_temp[0][0]) == 0):
11            can_update = True
12        if True:
13            for i in print_temp:
14                list_for_3d.append(i)
15        if len(list_for_3d) >= 200*50:
16            file_path = "example" + dt_string + ".txt"
17            with open(file_path, 'w') as file:
18                file.write(str(list_for_3d))
19            plot_3d_surface(list_for_3d)
20            printed = True

```

Establishes a serial connection and continuously reads data. When sufficient data is collected, it saves the data to a file and plots a 3D surface.

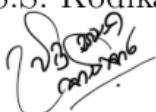
9 References

- Stefan May, David Droeschel, Dirk Holz, Christoph Wiesen, and Stefan Fuchs, "3D pose estimation and mapping with time-of-flight cameras," 2008. Available at: [here](#).
- Stefan May, David Droeschel, Stefan Fuchs, Dirk Holz, and Andreas Nüchter, "Robust 3D-Mapping with Time-of-Flight Cameras," *Fraunhofer IAIS*, 2009. Available at: [here](#)
- Nathan Larkin, Zeng Xi Pan, Stephen van Duin, and John Norrish, "3D mapping using a ToF camera for self programming an industrial robot," *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Wollongong, NSW, Australia, 2013, pp. 494-499. Available at: [here](#)
- Stefan May, David Droeschel, Dirk Holz, Stefan Fuchs, Ezio Malis, Andreas Nüchter, and Joachim Hertzberg, "Three-dimensional mapping with time-of-flight cameras," *Journal of Field Robotics*, 2009. Available at: [here](#)
- A. Kolb, E. Barth, and R. Koch, "ToF-sensors: New dimensions for realism and interactivity," in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, Anchorage, AK, USA, 2008. Available at: [here](#)

10 Cross Checked By

1. Name: U.S.S. Kodikara

Signature:



2. Name: J.N. Kodithuwakku

Signature:



3. Name: G.M.M. Sehara

Signature:



4. Name: E.R.N.H. Gunawardane

Signature:

