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

7th July 2024

Contents

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.

Figure 1: ATmega 2560

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.

Figure 1. Pinout ATmega640/1280/2560

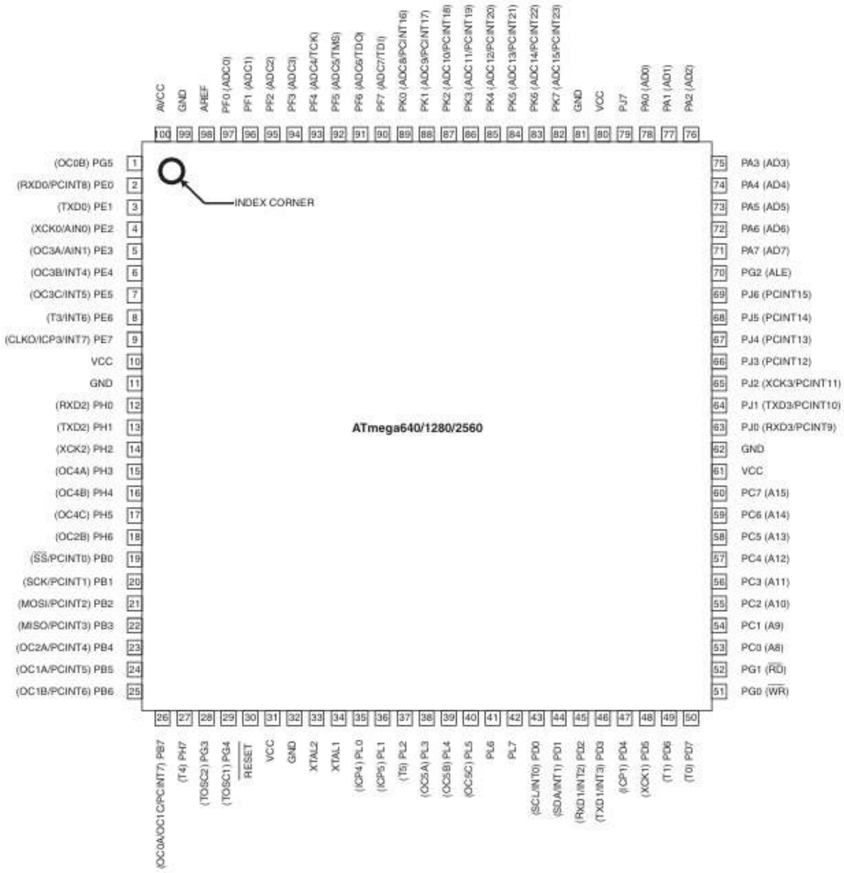


Figure 2: ATmega2560

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.

Figure 3: TOF

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

TCA9548A

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

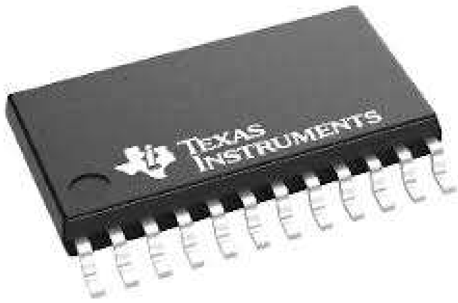


Figure 4: TCA9548A

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

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

Figure 5: CH340C

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

Nema 17HS4401

- The NEMA 17HS4401 stepper motor is renowned for its versatility and reliability in various applications requiring precise motion control

Figure 6: Nema 17

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·cm^2.
- **Weight:** 0.37 kg.

Reasons for Selection

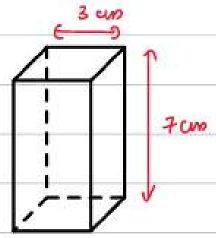
1. **Compact Size:** Suitable for applications with limited space, such as 3D printers and small CNC machines.
2. **Moderate Torque:** Provides adequate torque (0.35 Nm continuous, 0.5 Nm peak) for precise motion control and handling moderate loads.
3. **Precise Control:** 1.8° step angle and high subdivision 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 effectiveness, 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.

Calculation for Selection

Needed degree of freedom - 2

Required step Angle - 1.8°

Stepper Motor XZ-axis



Total moment of Inertia $\Rightarrow I_{1\text{-sheet}} \times 5$

$$= \frac{1}{12} m(a^2+b^2) \times 5$$

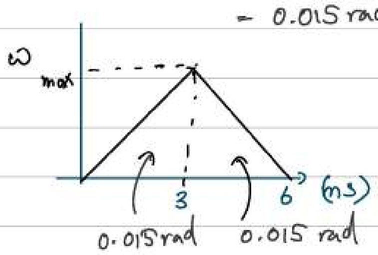
$$= \frac{5}{12} \times (10\times10^{-3}\ \text{kg})\left((3\times10^{-3})^2 + (7\times10^{-3})^2\right)$$

$$= \frac{5}{12} \times 10\times10^{-3} \times 58\times10^{-6}$$

$$= 2.4167\times10^{-7}\ \text{kg m}^2$$

$$2\pi \times \frac{0.9}{360} = 0.015\ \text{rad}$$

$$I_{total} = I + I_{rotor} = 2.4167\times10^{-7} + 54\times10^{-7} = 56.4167\times10^{-7}$$



$$\tau = I\alpha = 5.641\times10^{-6} \times 3.33\times10^{3} = 1.878\times10^{-2}\ \text{Nm}$$

$$\boxed{= 1.878\ \text{Ncm}}$$

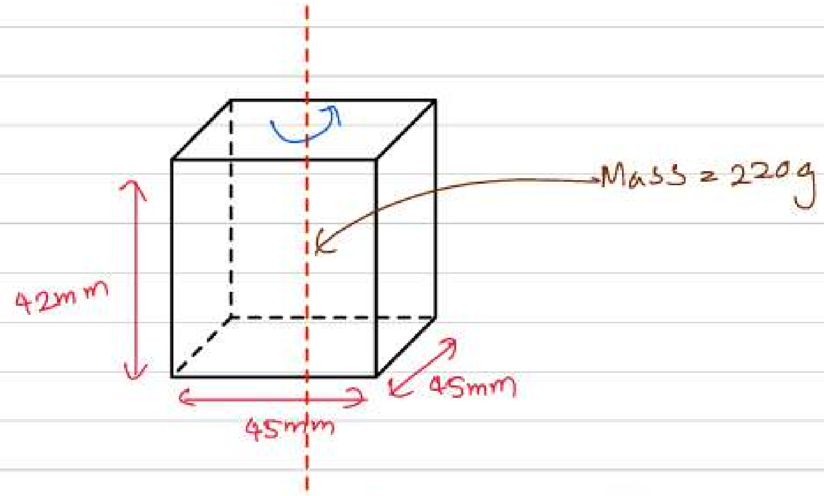
θ = Area

$$0.015 = \frac{1}{2} \times (3\times10^{-3}) \times \omega_{max}$$

$$\omega_{max} = 10\ \text{rad s}^{-1}$$

$$\alpha = \frac{\omega_{max}}{\text{time}} = \frac{10}{3\times10^{-3}} = 3.33\times10^{3}\ \text{rad s}^{-2}$$

Motor for XY plane → Needed step angle (1.8 deg)



$$I = \frac{1}{12} m \left((42\times10^{-3})^2 + (42\times10^{-3})^2 \right)$$

$$= \frac{1}{12} \times (280\times10^{-3}) \times 3528 \times 10^{-6}$$

$$= 8.232\times10^{-5}$$

$$I_{total} = I + I_{motor}$$

$$= 8.232\times10^{-5} + 5.4\times10^{-6}$$

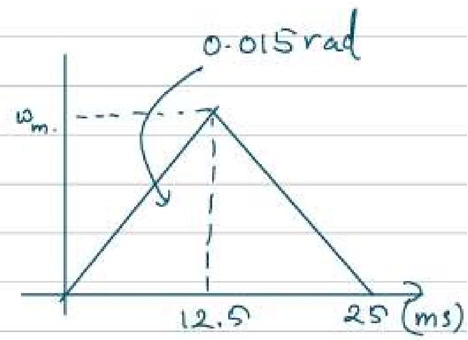
$$= 8.772\times10^{-5}\ kg\ m^2$$

$$\tau = I\alpha$$

$$= 8.772\times10^{-5} \times 192$$

$$= 1.68\times10^{-2}\ Nm$$

$$= 1.68\ Ncm$$



$$0.015 = \frac{1}{2} \times \omega_m \times 12.5\times10^{-3}$$

$$\omega_{max} = 2.4$$

$$\alpha = \frac{\omega_m}{t} = \frac{2.4}{12.5\times10^{-3}}$$

$$= 192\ rad\ s^{-2}$$

Electrical Specifications:

Series Model	Step Angle (deg)	Motor Length (mm)	Rated Current (A)	Phase Resistance (ohm)	Phase Inductance (mH)	Holding Torque (N.cm Min)	Detent Torque (N.cm Max)	Rotor Inertia (g.cm²)	Lead Wire (No.)	Motor Weight (g)
17HS2408	1.8	28	0.6	8	10	12	1.6	34	4	150
17HS3401	1.8	34	1.3	2.4	2.8	28	1.6	34	4	220
17HS3410	1.8	34	1.7	1.2	1.8	28	1.6	34	4	220
17HS3430	1.8	34	0.4	30	35	28	1.6	34	4	220
17HS3630	1.8	34	0.4	30	18	21	1.6	34	6	220
17HS3616	1.8	34	0.16	75	40	14	1.6	34	6	220
17HS4401	1.8	40	1.7	1.5	2.8	40	2.2	54	4	280
17HS4402	1.8	40	1.3	2.5	5.0	40	2.2	54	4	280
17HS4602	1.8	40	1.2	3.2	2.8	28	2.2	54	6	280
17HS4630	1.8	40	0.4	30	28	28	2.2	54	6	280
17HS8401	1.8	48	1.7	1.8	3.2	52	2.6	68	4	350
17HS8402	1.8	48	1.3	3.2	5.5	52	2.6	68	4	350
17HS8403	1.8	48	2.3	1.2	1.6	46	2.6	68	4	350
17HS8630	1.8	48	0.4	30	38	34	2.6	68	6	350

***Note:** We can manufacture products according to customer's requirements.

Figure 7: Nema17 Motor Comparision

Other constraints of choosing a suitable stepper motor were availability and price within budget. So we decided **Nema 17HS4401** as the best option for the 3D scanner considering all the factors.

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.

Figure 8: TB6600

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.

Figure 9: R-78CK-0.5

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.

Figure 10: oscillator

Specifications

- Model: HC49/4HSMX
- Frequency: 16 MHz
- Load Capacitance: 22 pF
- Shunt Capacitance: 7pF max
- Tolerance: 30 PPM
- Frequency Stability: 50 PPM
- Overtone Order: Fundamental

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

100µF Decoupling Capacitors

- 100µF decoupling capacitors were used to stabilize the power supply to the microcontroller and other components.

Figure 11: capacitor

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.

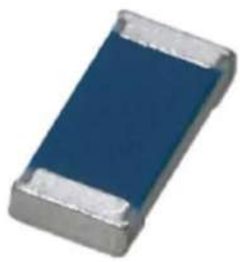


Figure 12: resistor

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**
2. **USB Port Circuit**
3. **I2C Multiplexer Circuit**
4. **Power Supply Circuit**

3.1 System Overview

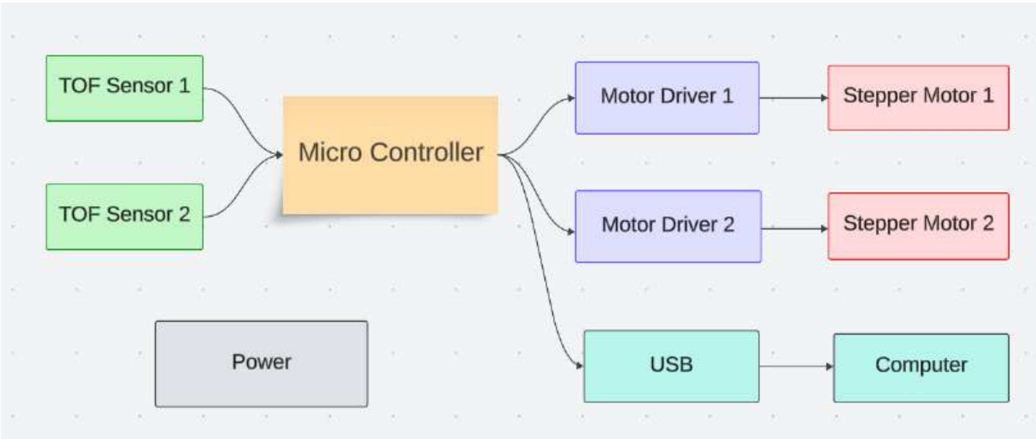


Figure 13: block diagram

The system consists of the following components and their interactions:

- **Microcontroller:**
  - Receives data from TOF sensors
  - Send control signals to stepper motors.
- **TOF Sensors:**
  - Measure distance to objects.
  - Send distance data to the microcontroller.
- **Stepper Motor Drivers:**
  - Receive signals from the microcontroller.
  - Control the rotation of stepper motors based on these signals.
- **Stepper Motors:**
  - Rotate according to commands given by the motor drivers.
- **USB Port:**
  - Transfers distance data from the microcontroller to the computer.

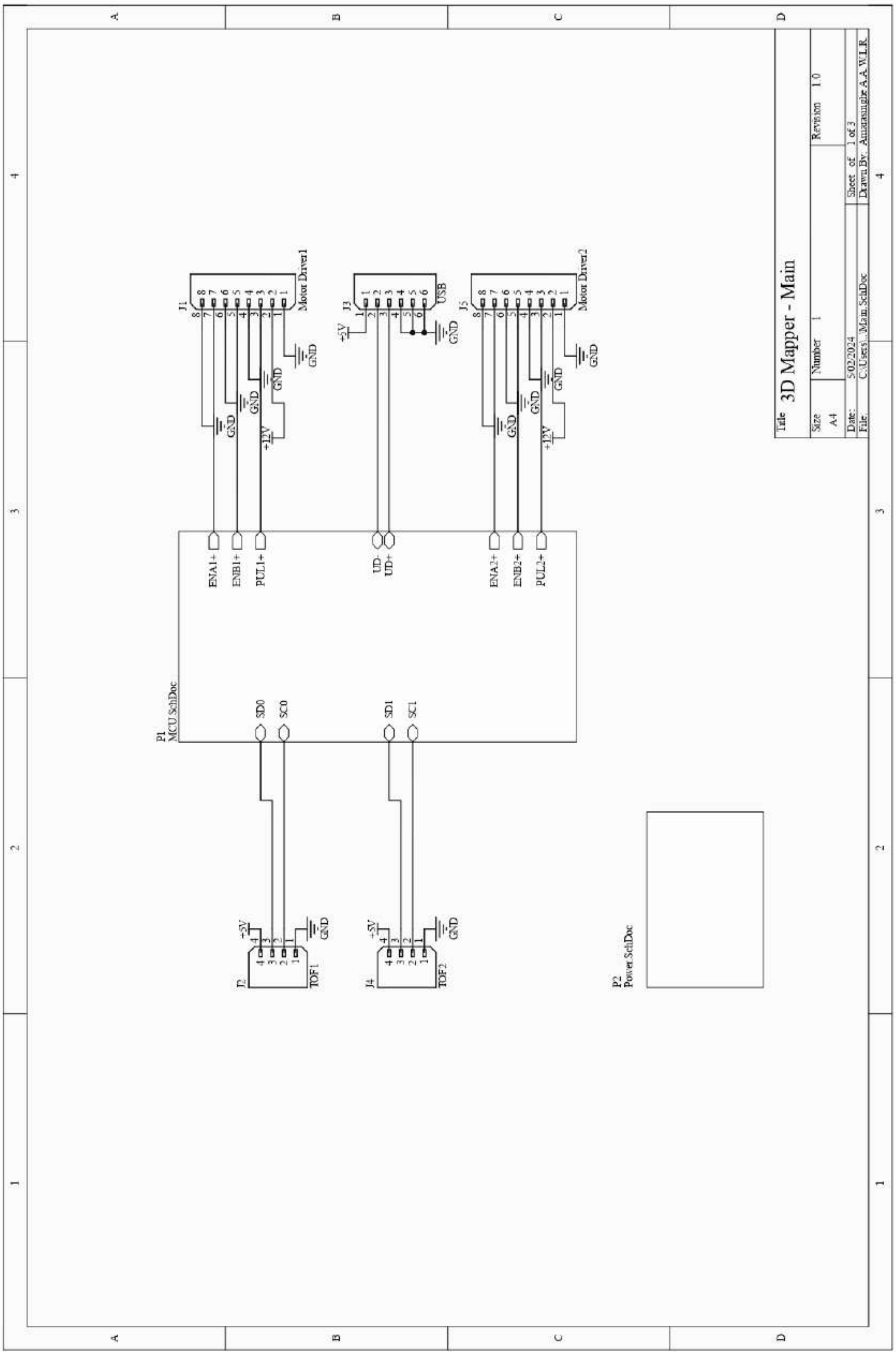


Figure 14: Main

3.2 Microcontroller Circuit

The microcontroller circuit is designed with the ATmega2560 as the core of the device.

- Microcontroller:
  - Used the ATmega2560, an 8-bit microcontroller.
  - Features 256K Bytes of In-System Programmable Flash memory.
  - Includes 54 digital I/O pins for versatile connectivity and control.
  - Ideal for memory-intensive applications.

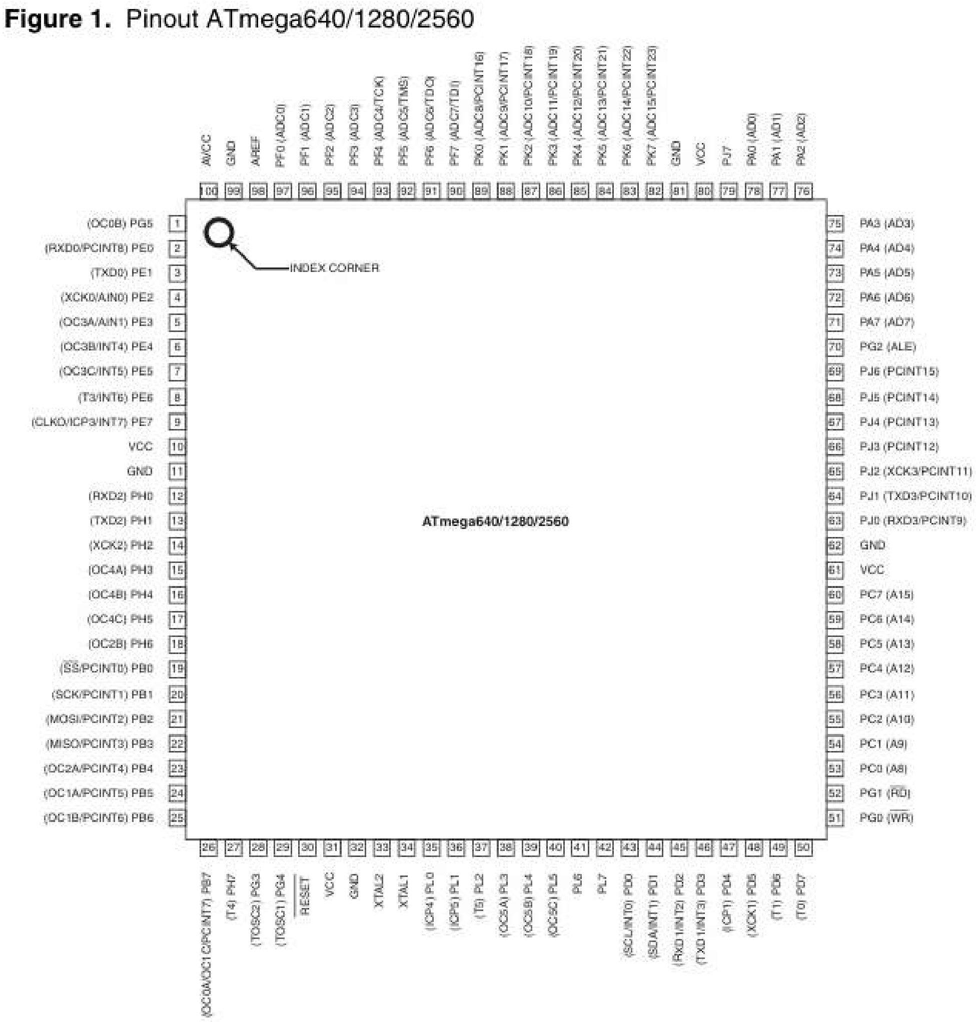


Figure 15: ATmega2560

- Connections:
  - CH340C chip is connected to the MCU for USB-to-serial communication.
  - I2C multiplexer is connected to the MCU for handling multiple I2C devices.
  - Two stepper motor drivers are connected to the MCU to provide ENABLE, DIRECTION, and PULL commands.

- Clocking:
  - Included a 16 MHz crystal oscillator for accurate clocking.
  - Stabilized the oscillator with capacitors to ensure precise frequency and reliable operation.

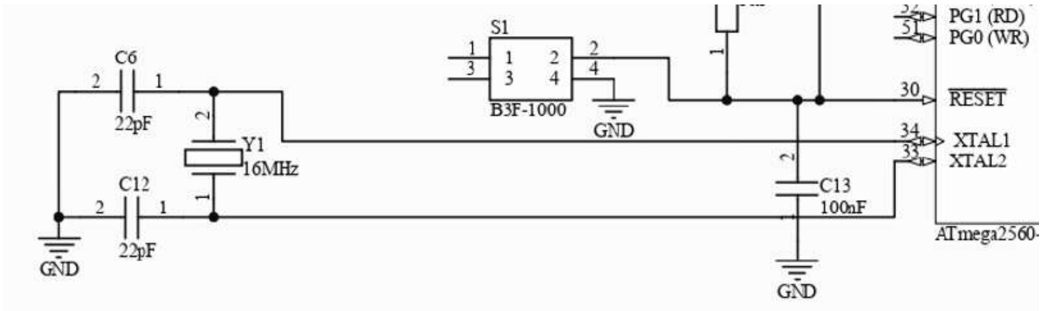


Figure 16: oscilator

- Noise Reduction & Stabilization:
  - Added decoupling capacitors to reduce noise and ensure stable operation.

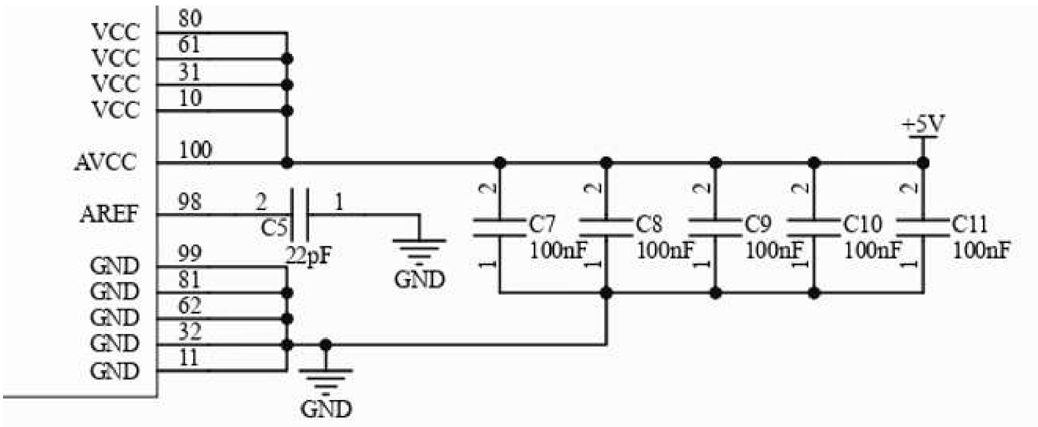


Figure 17: capacitors

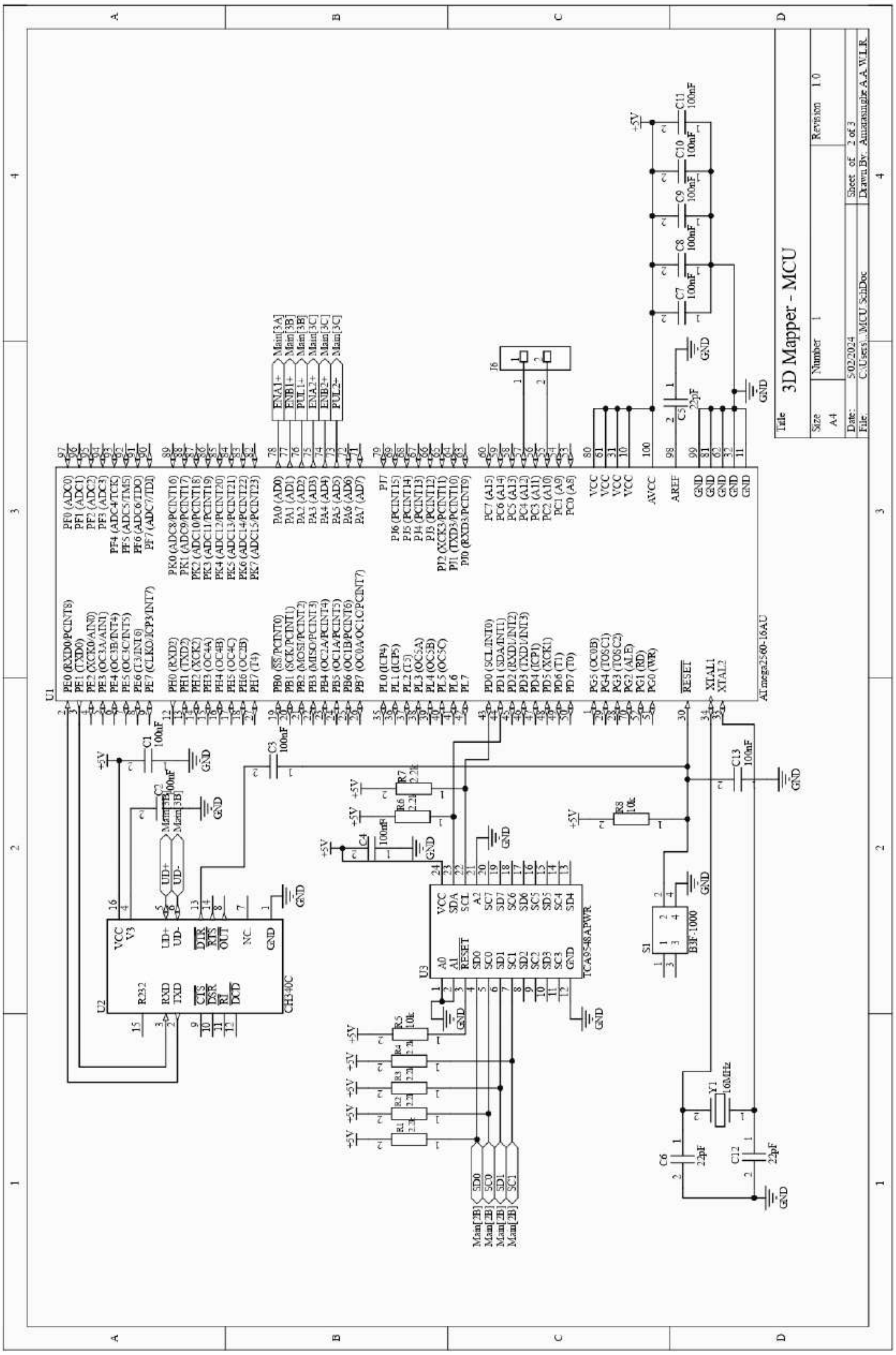


Figure 18: MCU circuit

3.3 USB Port Circuit

- Utilizes the CH340C USB-to-serial converter.
- Facilitates reliable data transfer to a computer.
- Proper decoupling capacitors for stable operation.
- Ensures compatibility with various operating systems through the CH340C driver.
- Provides stable and reliable communication between the microcontroller and the computer.
- Receiver and transmitter pins of CH340C chip is connected with transmitter and receiver of atmega 2560.

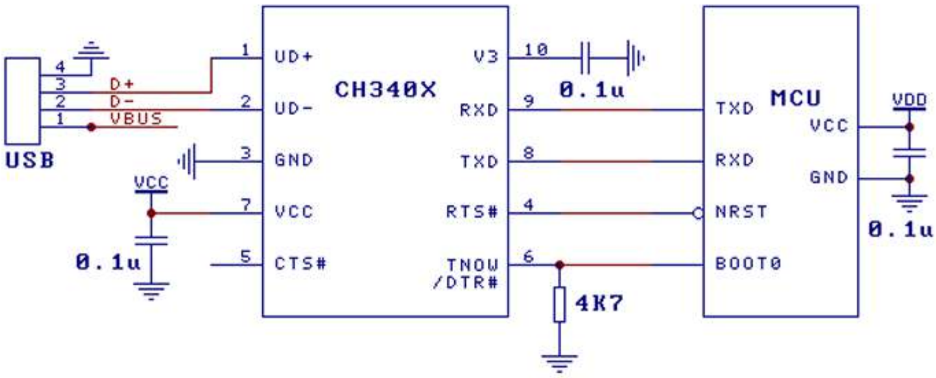


Figure 19: CH340

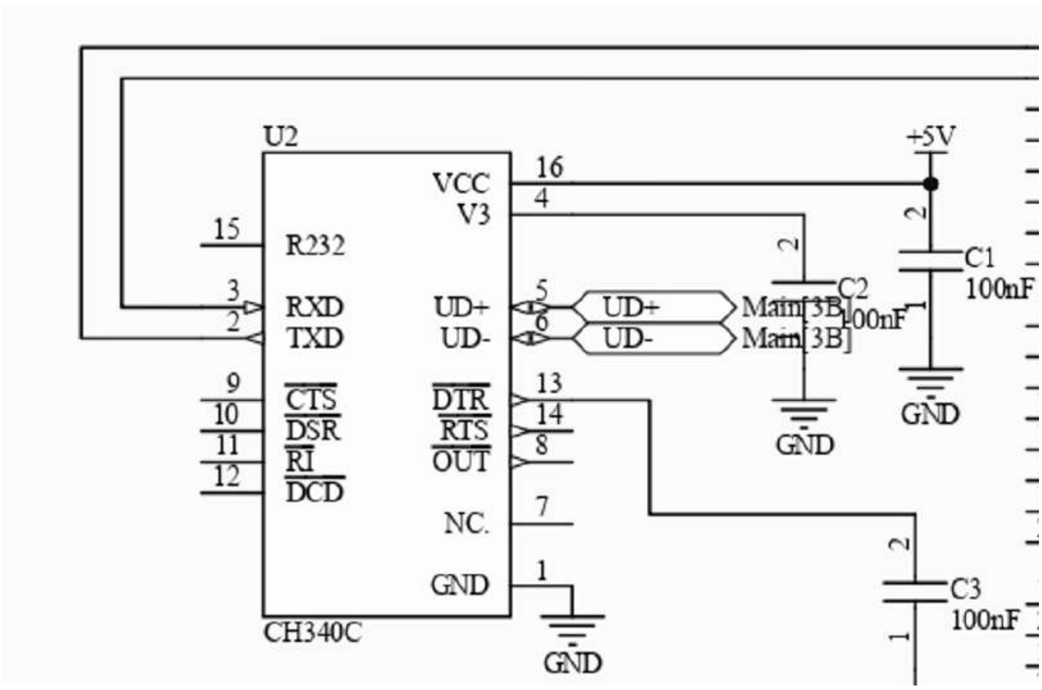


Figure 20: USB port circuit

3.4 I2C Multiplexer Circuit

- The I2C multiplexer circuit is designed to get data from two TOF sensors simultaneously.
- The multiplexer can support up to 8 I2C communication devices.
- Two TOF sensors are connected to the I2C multiplexer with SD0,SC0,SD1,SD1 ports.
- Pull-up resistors are used on the I2C lines to ensure proper communication:
  - Ensures that the SDA (data) and SCL (clock) lines are pulled to a high logic level when not actively driven low.
  - Helps to avoid floating states and improves the reliability of the communication.

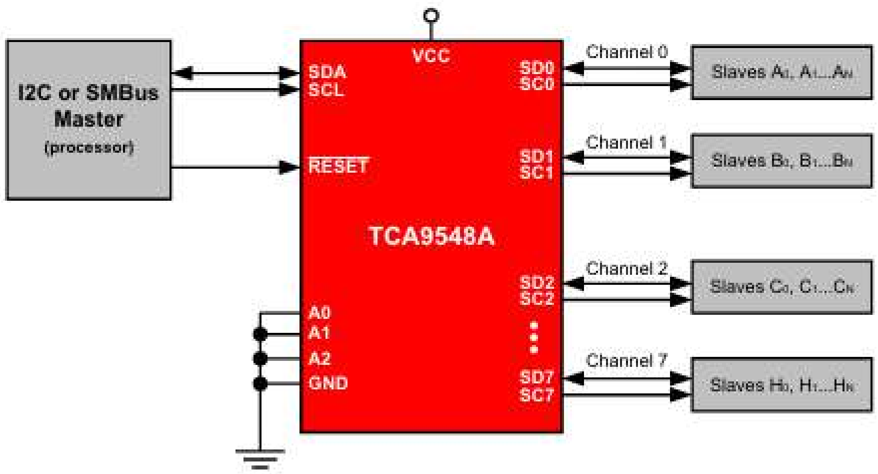


Figure 21: I2C multiplexer

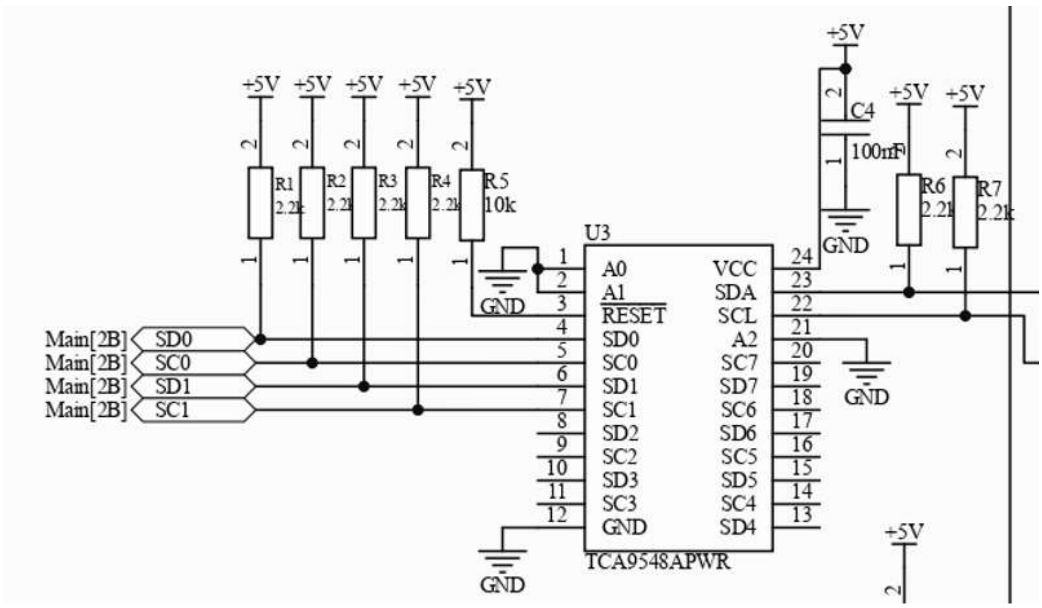


Figure 22: I2C multiplexer circuit

3.5 Power Supply Circuit

- Designed to provide a stable 5V supply from the 12V supply
- External 12V 10A SMPS Large Power Supply converts 230VAC to 12VDC and supplies to the power supply circuit
- R-78CK-0.5 ,non-isolated DC-DC converter converts 12V to 5V.

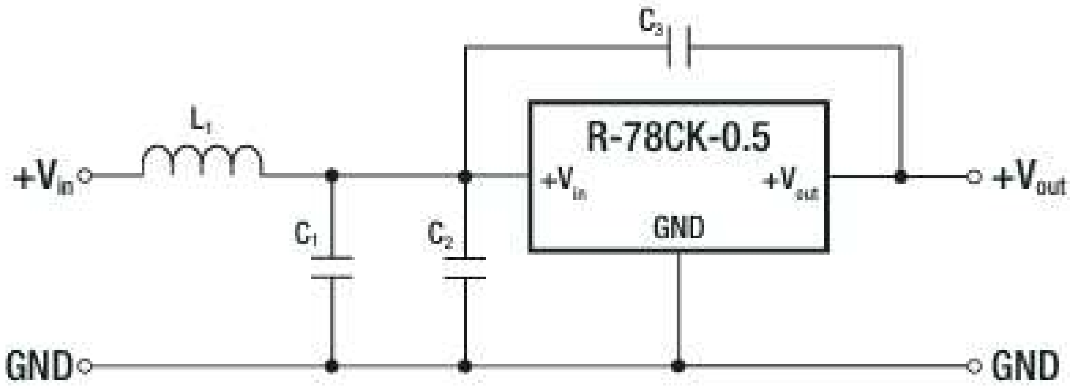


Figure 23: R-78CK-0.5

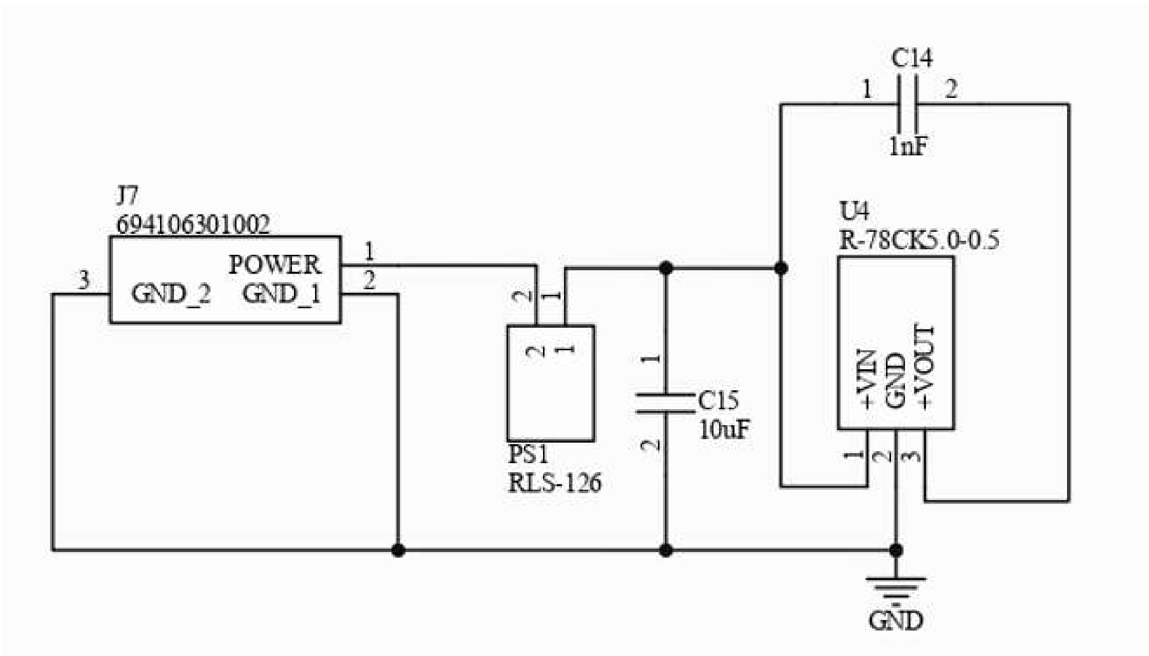


Figure 24: Power supply circuit

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.

3.6 Bill of Materials

Comment	Description	Designator	Footprint	LibRef	Quantity
VJ0805Y104JXXPW1BC	Capacitor	C1, C2, C3, C4, C7, C8, C9, C10, C11, C13	CAPC2012X90N	VJ0805Y104JXXPW1BC	10
0805ZA220JAT2A	Capacitor	C5, C6, C12, C15	CAPC2012X94N	0805ZA220JAT2A	4
C1206C102GBGACTU	Capacitor	C14	C1206	C1206C102GBGACTU	1
B8B-XH-A	Connector Header Through Hole 8 position 0.098 (2.50mm)	J1, J5	JST_B8B-XH-A	B8B-XH-A	2
B4B-EH-A	Connector Header Through Hole 4 position 0.098 (2.50mm)	J2, J4	JST_B4B-EH-A	B4B-EH-A	2
B6B-XH-A(LF)(SN)	Connector Header Through Hole 6 position 0.098 (2.50mm)	J3	JST_B6B-XH-A(LF)(SN)	B6B-XH-A(LF)(SN)	1
B2B-XH-AM(LF)(SN)	CONN HEADER VERT 2POS 2.5MM	J6	FP-B2B-XH-AM_LF_SN-MFG	CMP-17439-000037-1	1
694106301002	Connector	J7	694106301002_1	694106301002	1
RLS-126	Power Supply	PS1	RLS126	RLS-126	1
ERA-6APB222V	Resistor	R1, R2, R3, R4, R6, R7	ERA6AEB1020V	ERA-6APB222V	6
CMP0805AFX-1002ELF	Resistor	R5, R8	RESC2012X60N	CMP0805AFX-1002ELF	2
B3F-1000	Switch	S1	B3F1002	B3F-1000	1
ATmega2560-16AU	8-bit AVR Microcontroller, 4.5-5.5V, 16MHz, 256KB Flash, 4KB EEPROM, 8KB SRAM, 86 GPIO pins, 100-pin TQFP, Industrial Grade (-40°C to 85°C), Pb-Free	U1	100A_M	CMP-0095-00210-2	1
CH340C	USB to serial chip CH340	U2	SOIC127P600X180-16N	CH340C	1
TCA9548APWR	Integrated Circuit	U3	SOP65P640X120-24N	TCA9548APWR	1
R-78CK5.0-0.5	Power Supply	U4	R78CK5005	R-78CK5.0-0.5	1
LFXTAL027946Reel	Crystal or Oscillator	Y1	LFXTAL027946Reel	LFXTAL027946Reel	1

Figure 25: BOM

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 0.3 mm is chosen for signal traces, while a width of 0.4 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.

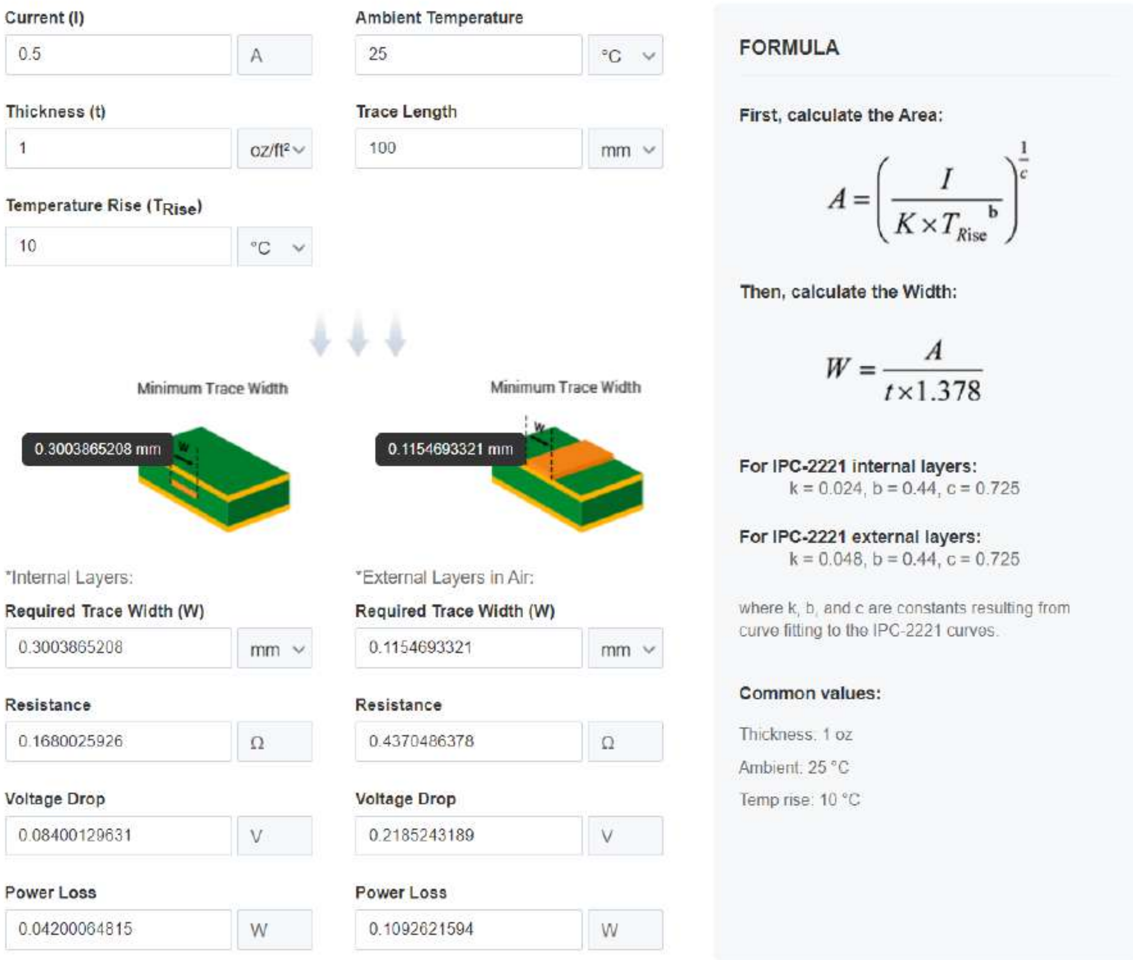


Figure 26: trace width calculation

4.1 PCB

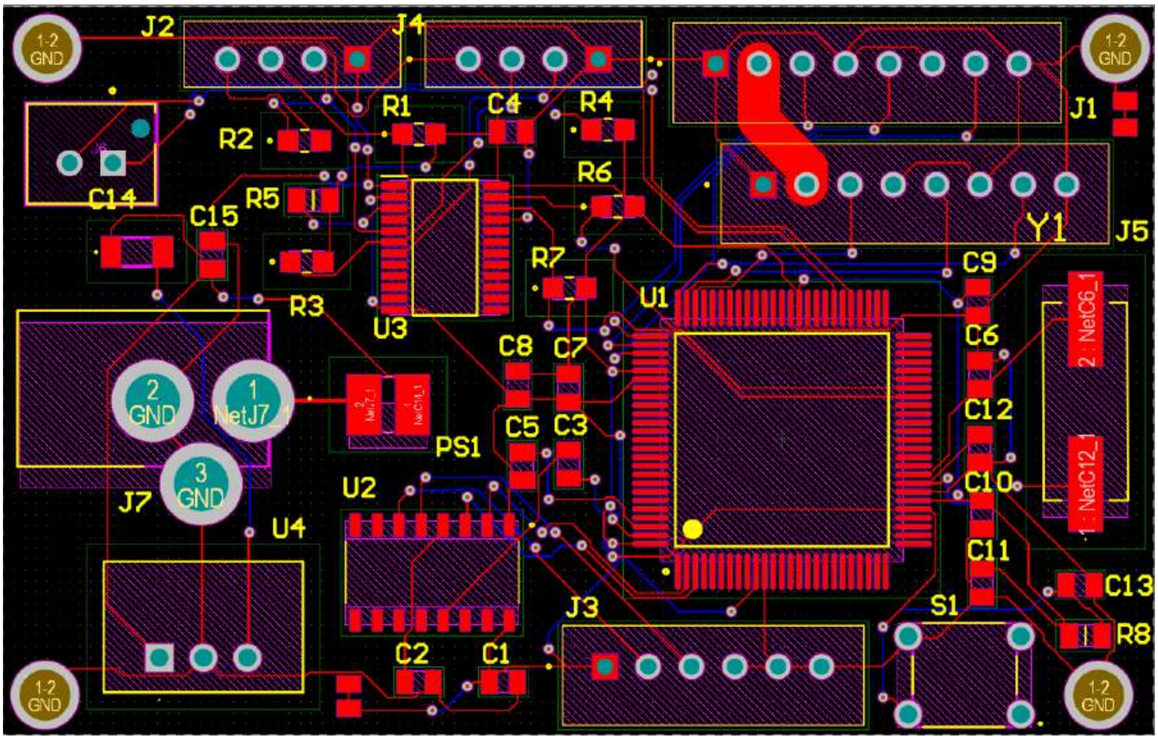


Figure 27: PCB

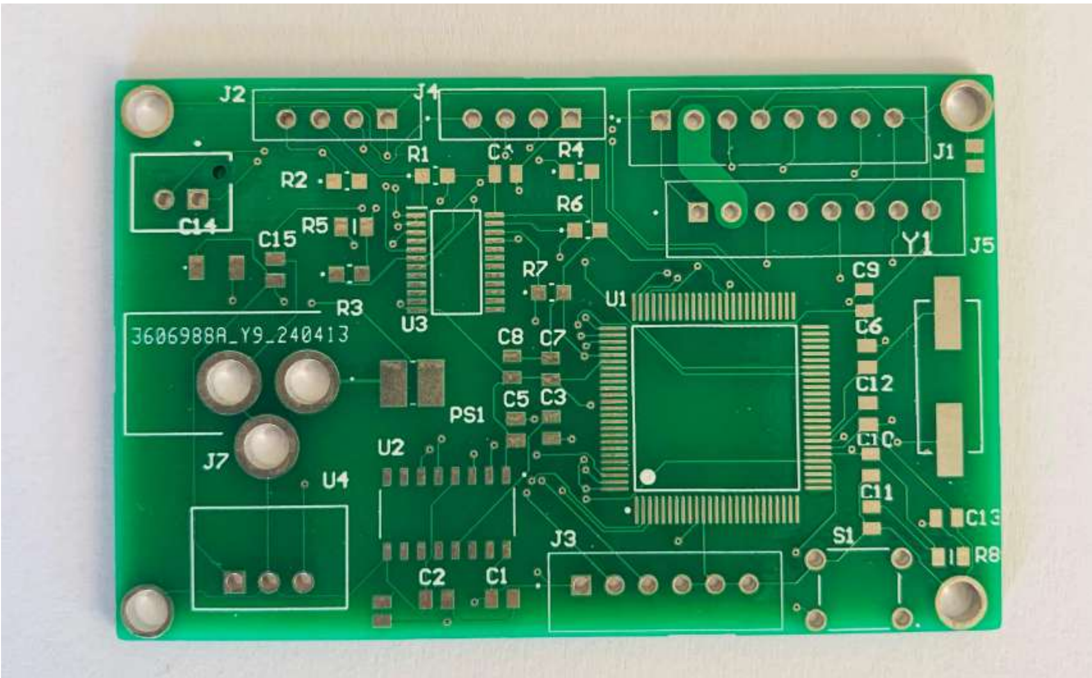


Figure 28: actual PCB

4.2 Top Layer

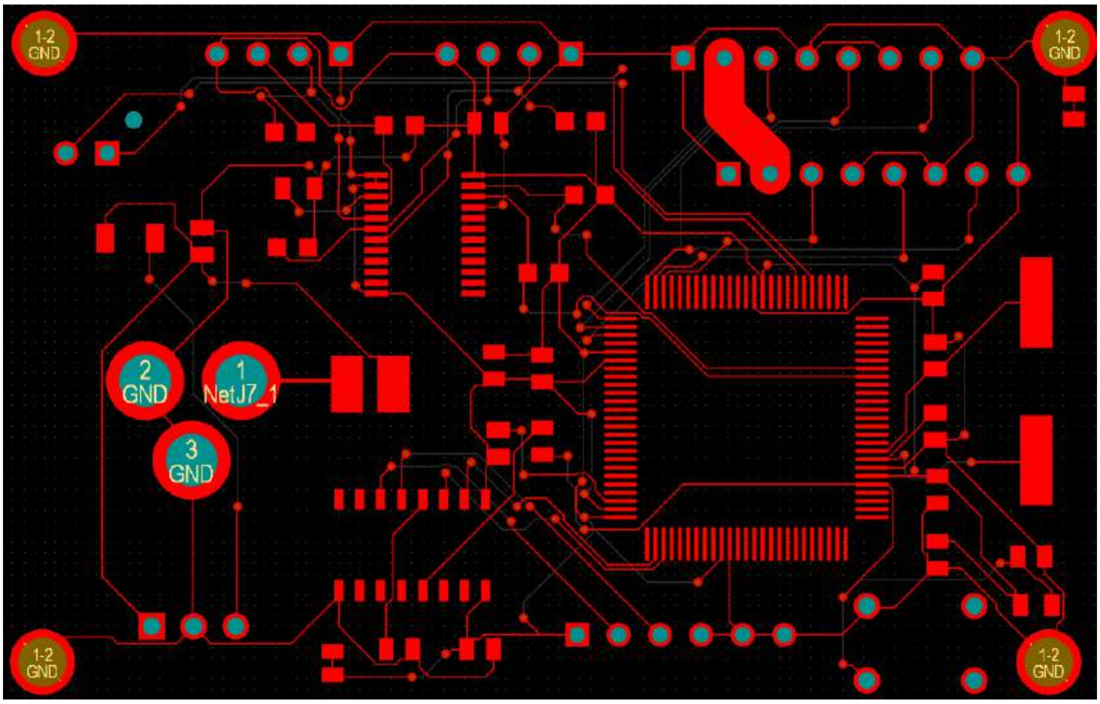


Figure 29: Top Layer

4.3 Bottom Layer

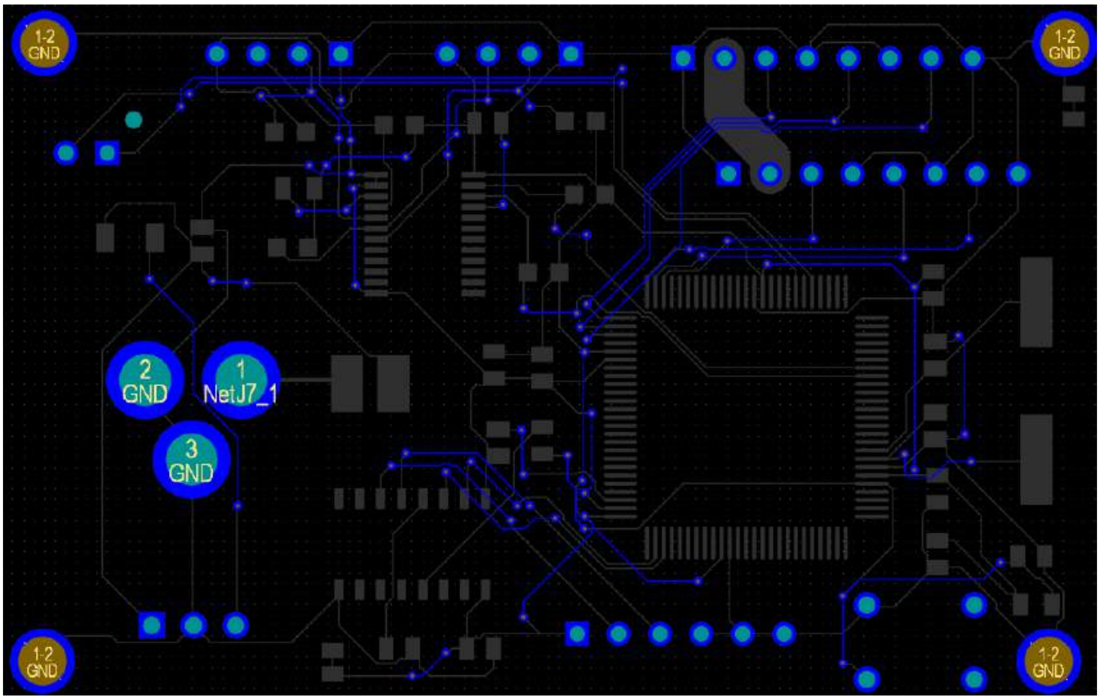


Figure 30: Bottom Layer

4.4 Overlay Layer

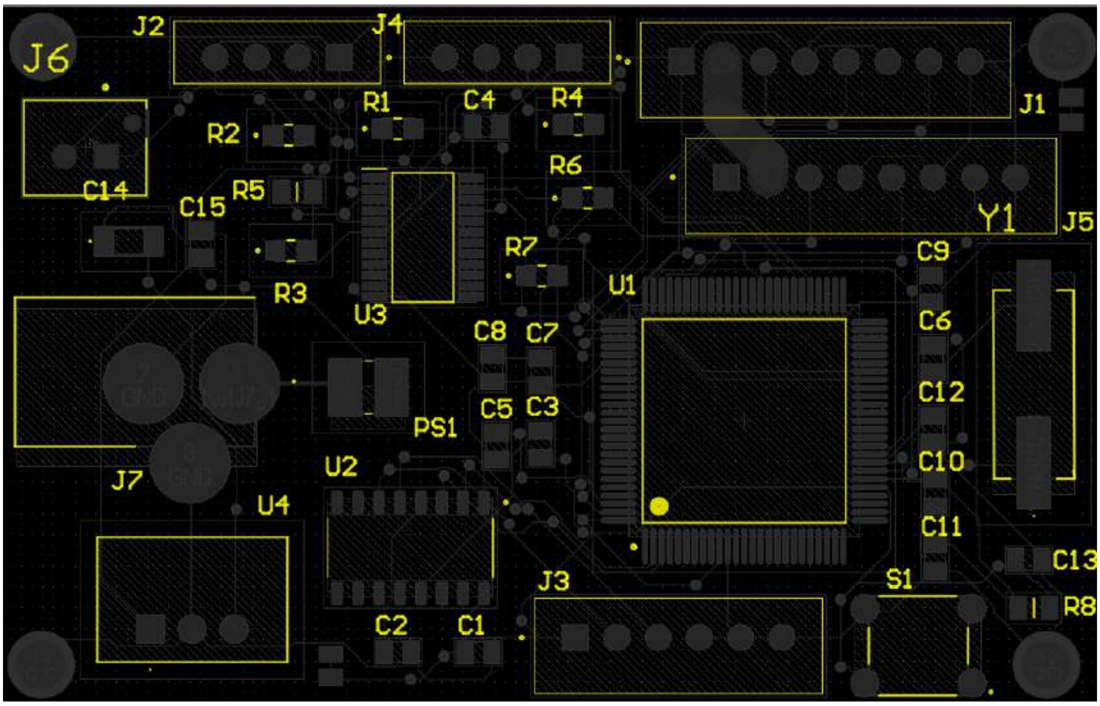


Figure 31: Overlay Layer

4.5 Drill Drawings

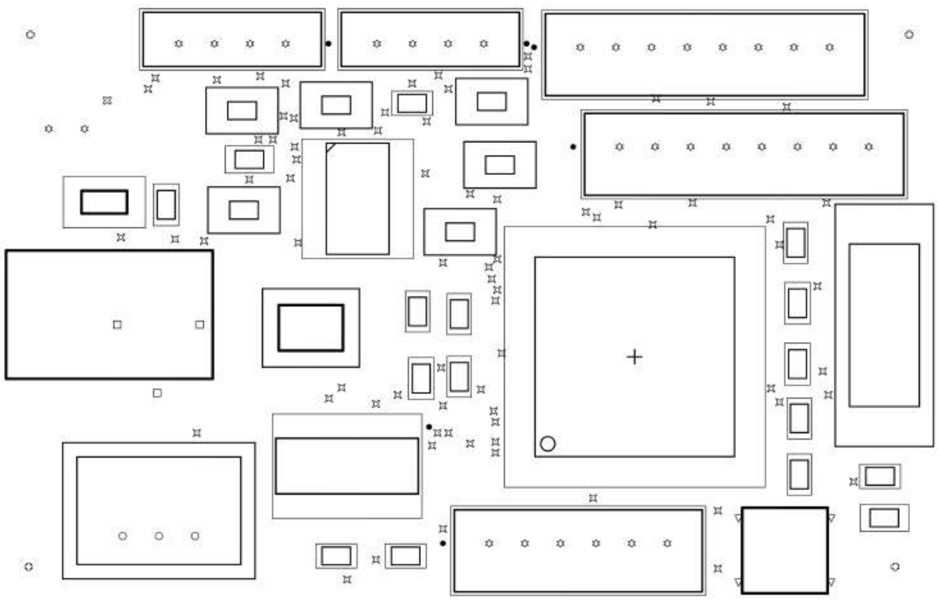


Figure 32: Drill Drawings

4.6 3D View

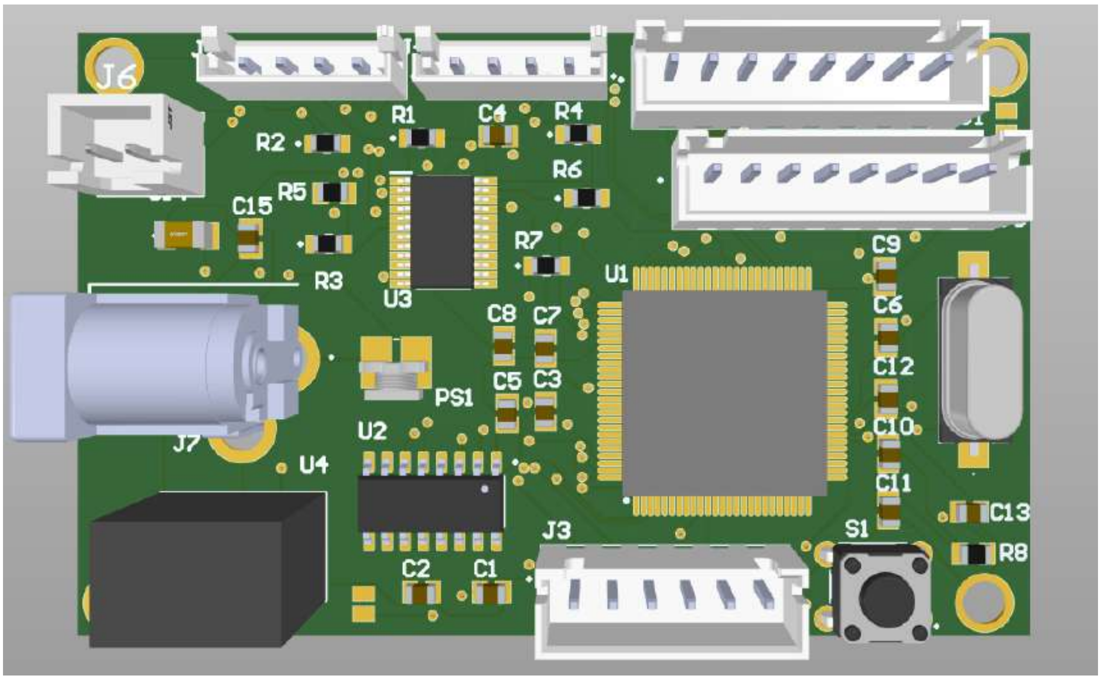


Figure 33: 3D View

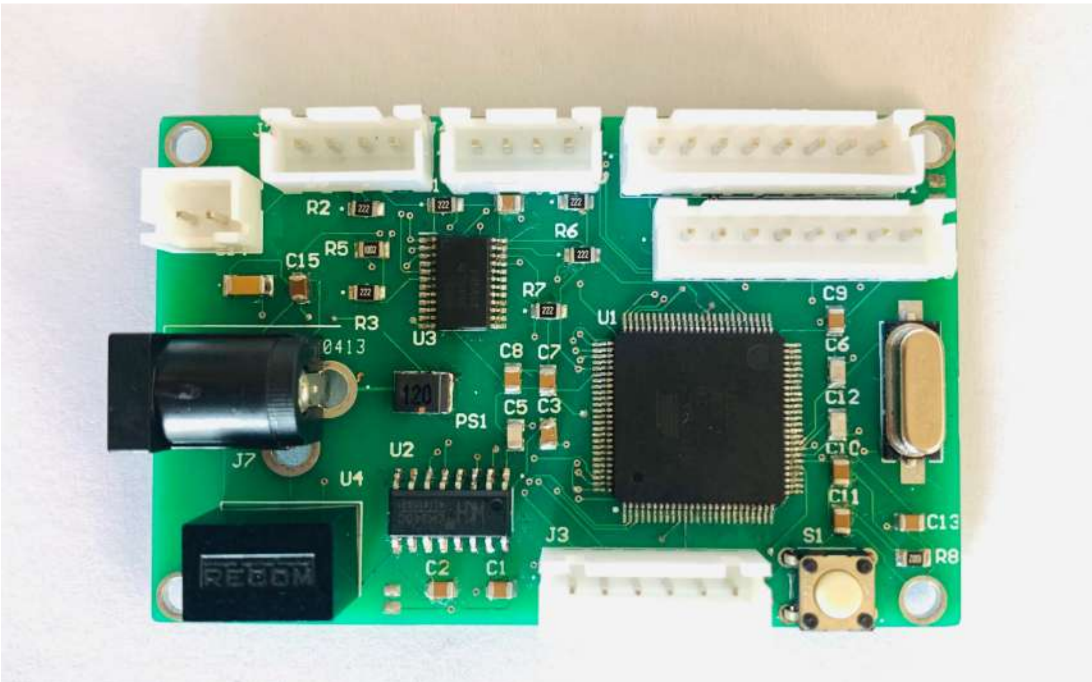


Figure 34: soldered PCB

5 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+.

5.3 Bottom Part

Figure 35: Enclosure

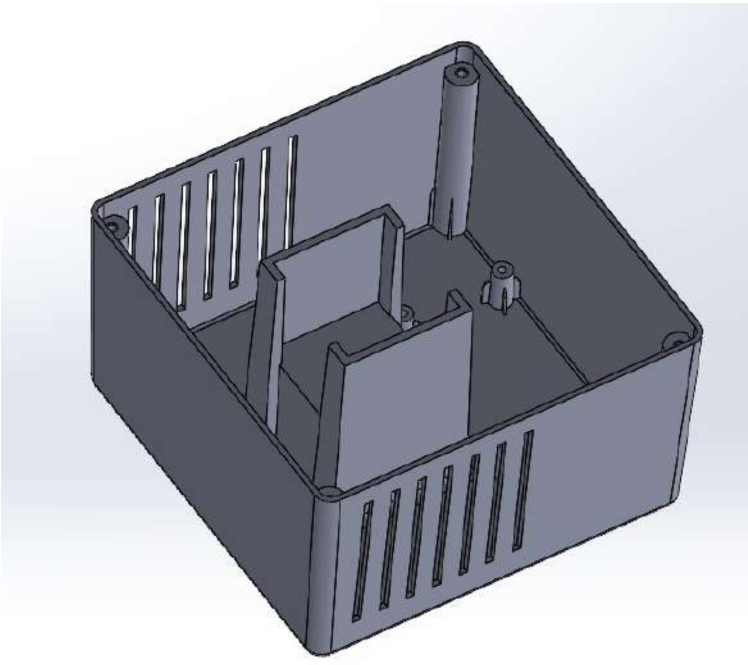


Figure 36: Side View

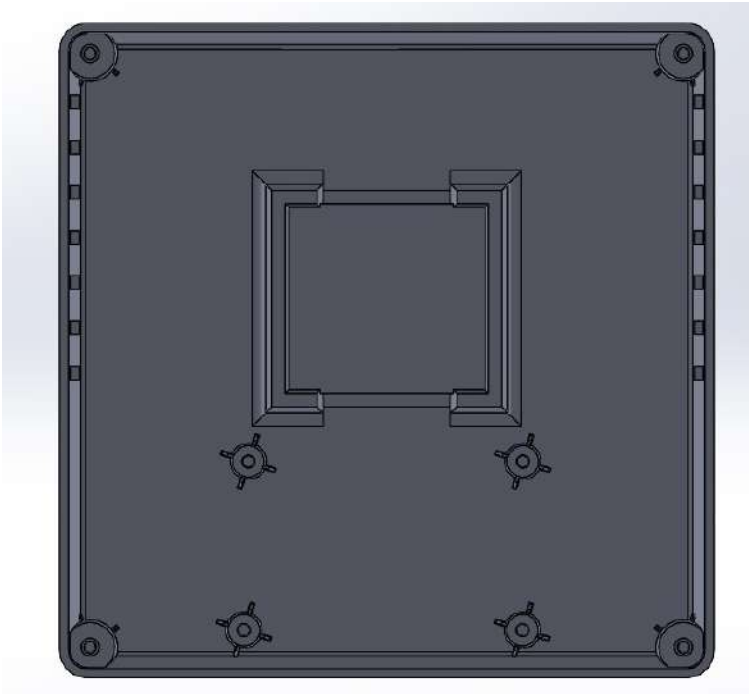


Figure 37: Top View

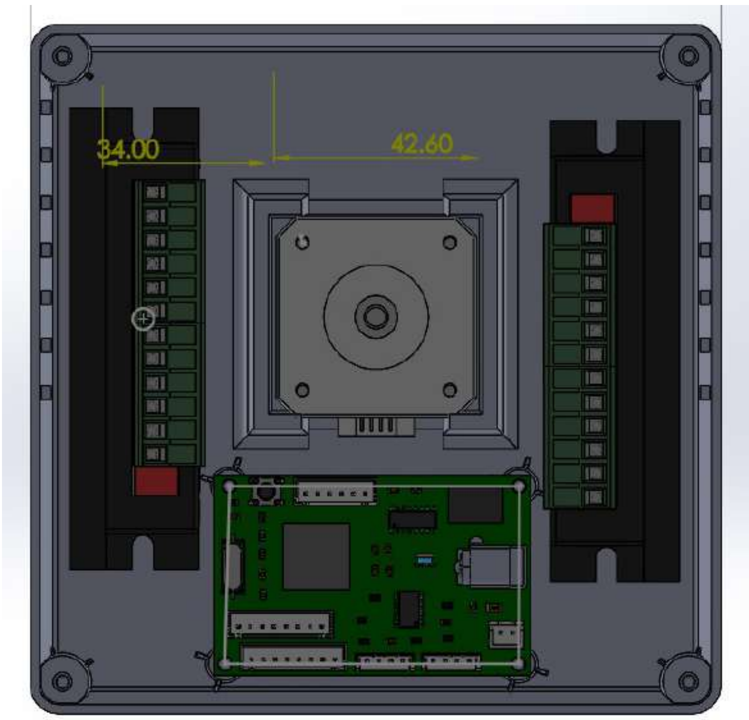


Figure 38: Top View with Components

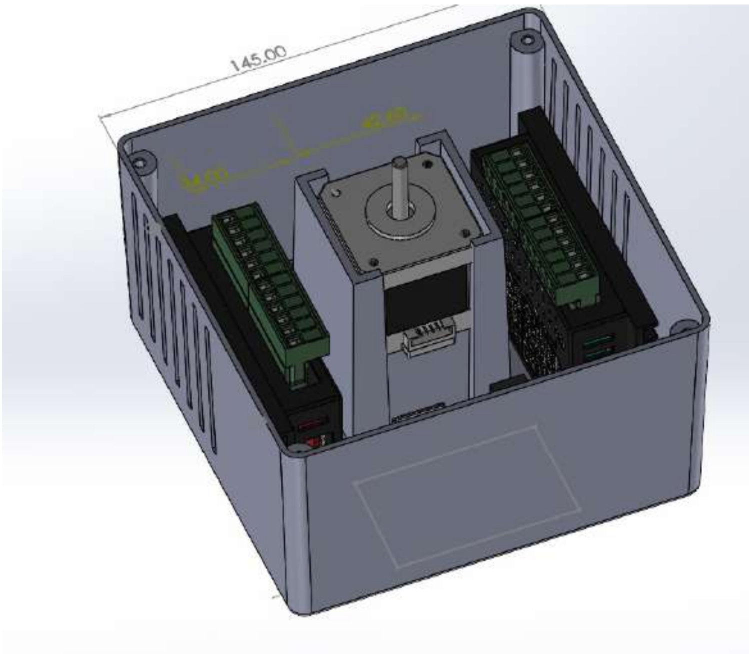


Figure 39: Side View with Components

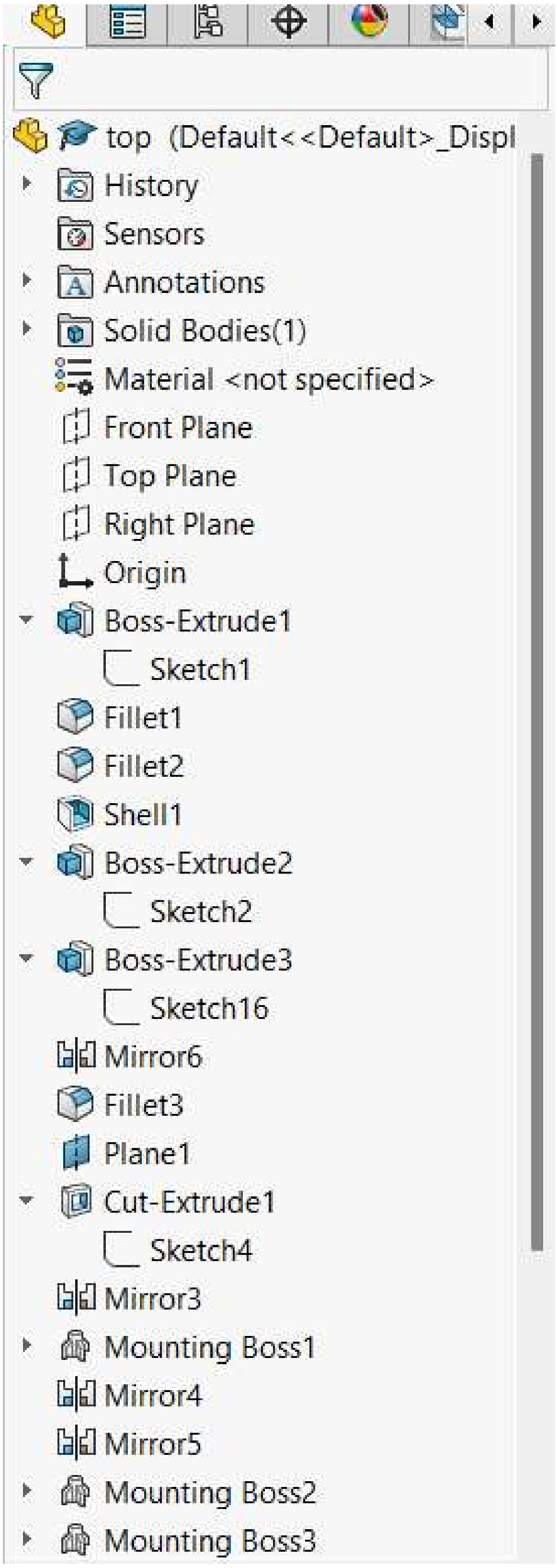


Figure 40: Model Tree - Bottom Part

5.4 Top Part

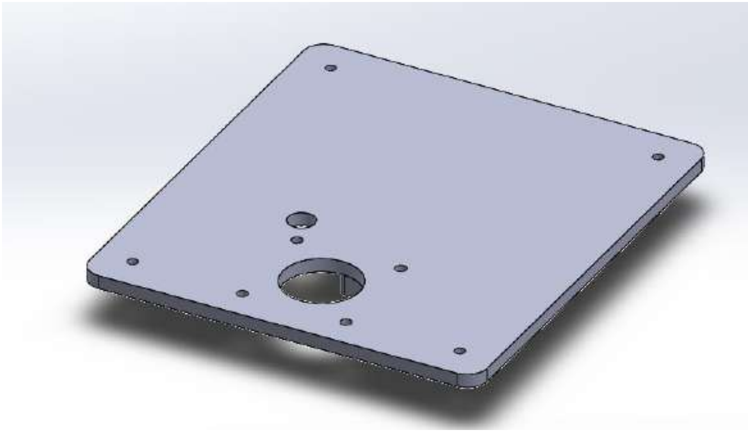


Figure 41: top view

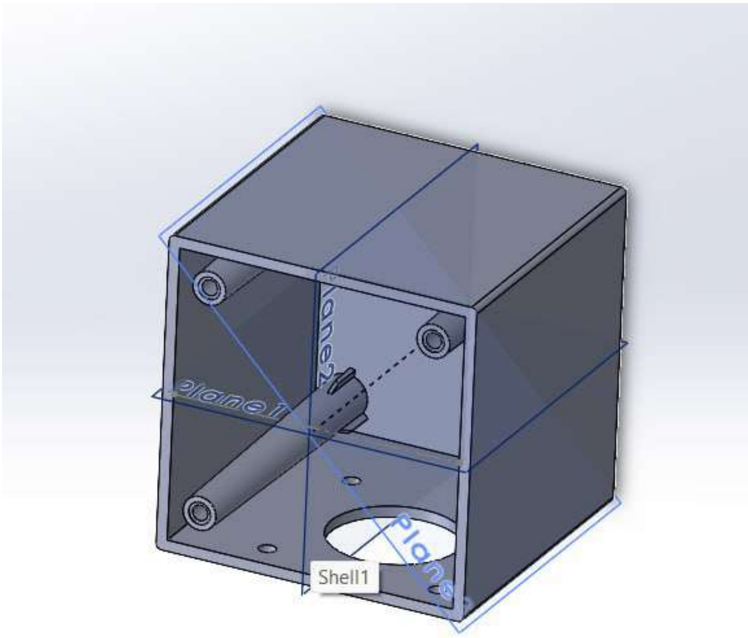


Figure 42: isometric view

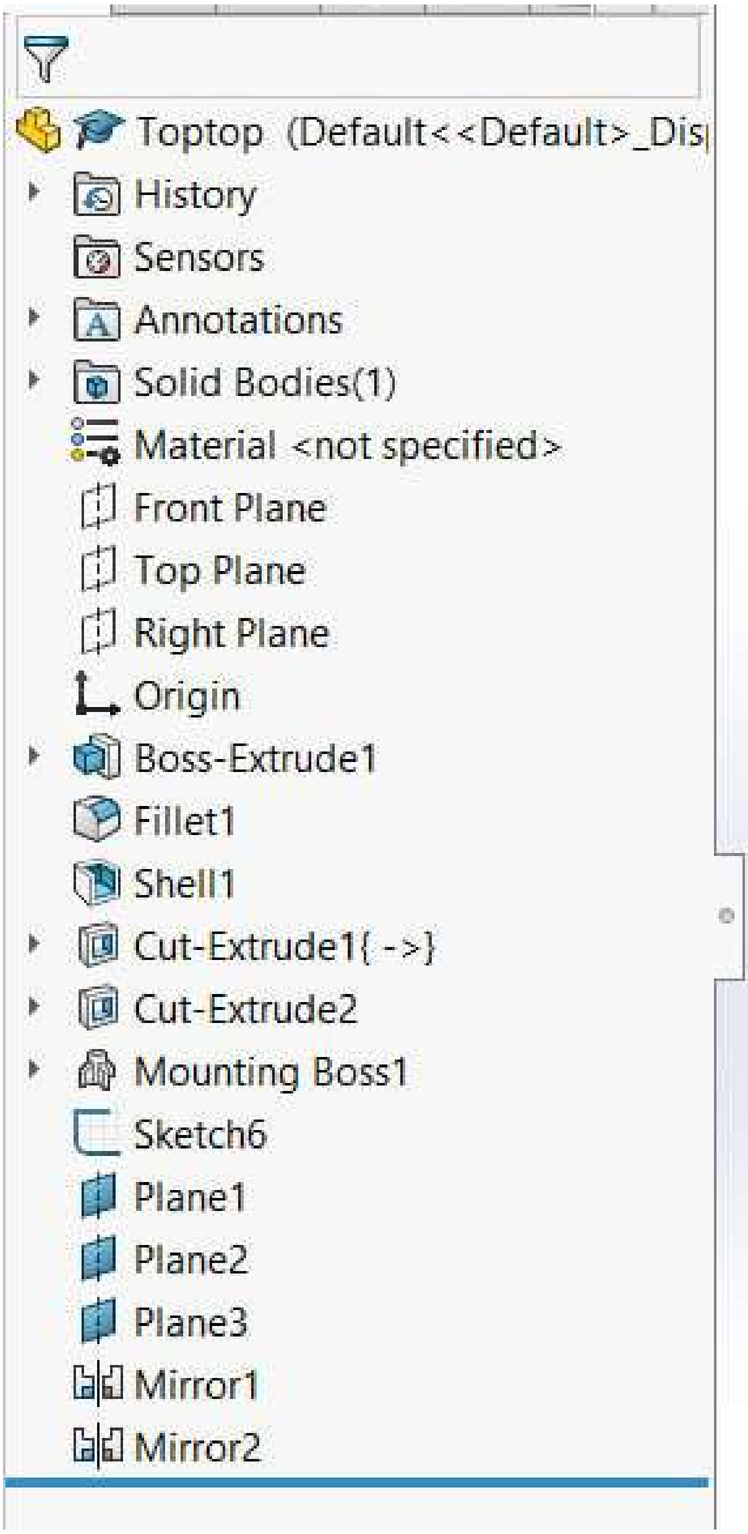


Figure 43: model tree - top part

5.5 Lid

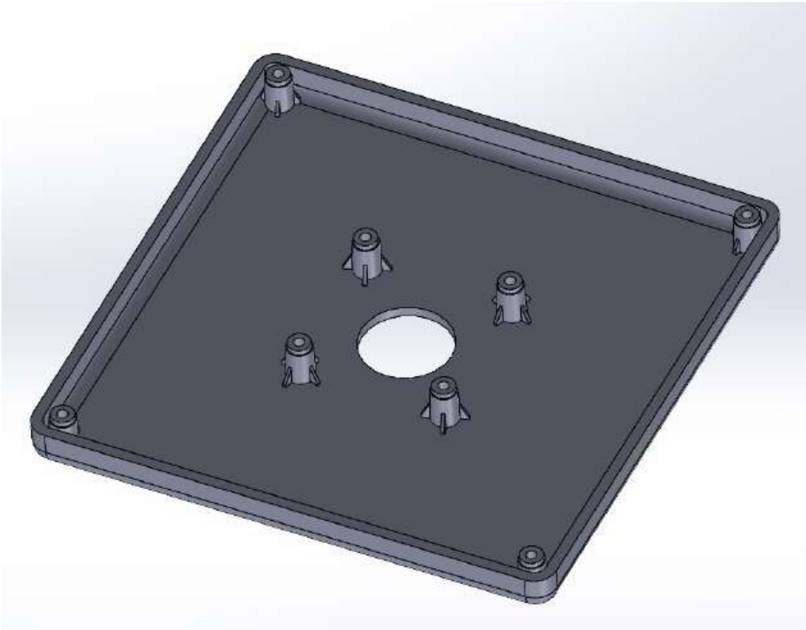


Figure 44: Side View

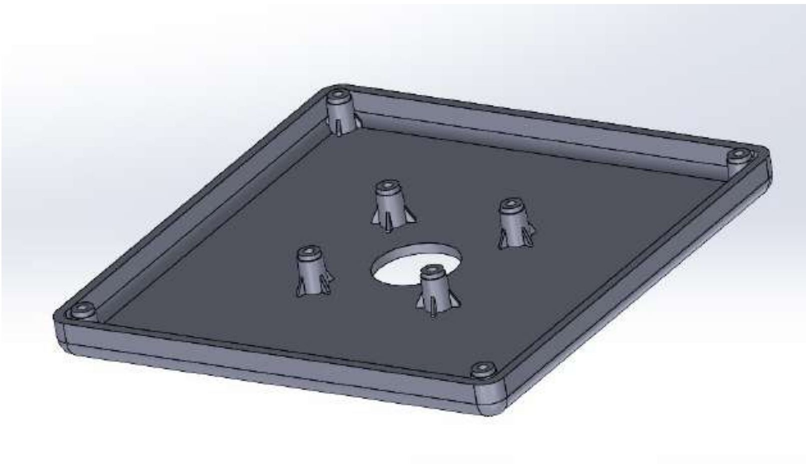


Figure 45: Inside Lid

Figure 46: Outside Lid

5.6 TOF holder

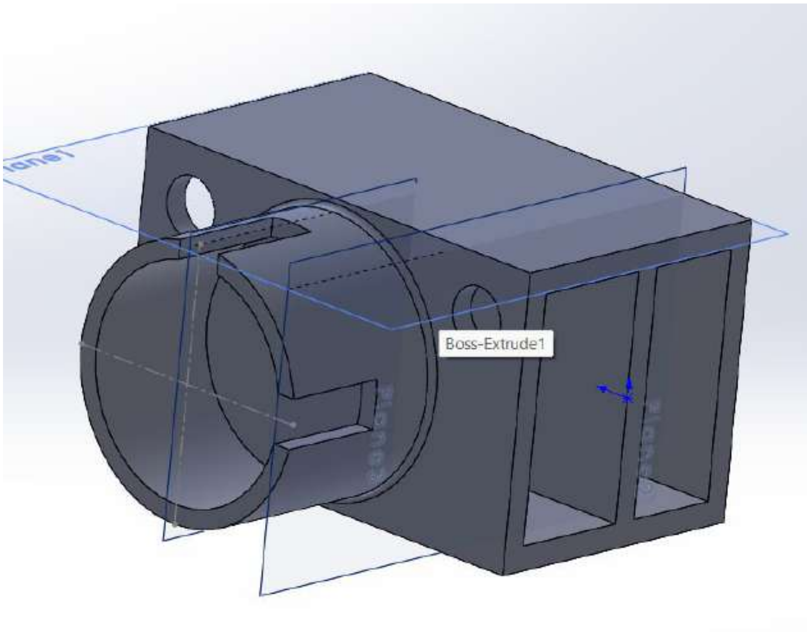


Figure 47: TOF holder

6 Detailed Programming Information

6.1 C++ Code for the Micro-controller

```cpp
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "VL53L0X.h"

#define STEP_PIN_1 24
#define DIR_PIN_1 23
#define ENA_PIN_1 22

#define STEP_PIN_2 27
#define DIR_PIN_2 26
#define ENA_PIN_2 25

#define XY_REV 200
#define XZ_REV 50

#define ANGLE_INCREMENT 1.8

#define F_CPU 16000000UL // Assuming a 16 MHz clock
#define BAUD 9600
#define MYUBRR F_CPU/16/BAUD-1

#define F_SCL 100000UL // SCL frequency
#define Prescaler 1
#define TWBR_val ((((F_CPU / F_SCL) / Prescaler) - 16) / 2)

float measurements[XZ_REV][3];

void uart_init(unsigned int ubrr) {
  // Set baud rate
  UBRR0H = (unsigned char)(ubrr >> 8);
  UBRR0L = (unsigned char)ubrr;
  // Enable receiver and transmitter
  UCSR0B = (1 << RXEN0) | (1 << TXEN0);
  // Set frame format: 8 data bits, 1 stop bit
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}

void uart_transmit(unsigned char data) {
  // Wait for empty transmit buffer
  while (!(UCSR0A & (1 << UDRE0)));
  // Put data into buffer, sends the data
  UDR0 = data;
}

void uart_print(const char *str) {
  while (*str) {
    uart_transmit(*str++);
  }
}

void i2c_init(void) {
  TWSR = 0x00;
  TWBR = (uint8_t)TWBR_val;
}

void i2c_start(void) {
  TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
  while (!(TWCR & (1 << TWINT)));
}

void i2c_stop(void) {
  TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
```

```
  while (TWCR & (1 << TWSTO));
}

void i2c_write(uint8_t data) {
  TWDR = data;
  TWCR = (1 << TWINT) | (1 << TWEN);
  while (!(TWCR & (1 << TWINT)));
}

uint8_t i2c_read_ack(void) {
  TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
  while (!(TWCR & (1 << TWINT)));
  return TWDR;
}

uint8_t i2c_read_nack(void) {
  TWCR = (1 << TWINT) | (1 << TWEN);
  while (!(TWCR & (1 << TWINT)));
  return TWDR;
}

void tof(uint8_t bus) {
  i2c_start();
  i2c_write(0x70 << 1); // TCA9548A address is 0x70
  i2c_write(1 << bus);  // send byte to select bus
  i2c_stop();
}

void setup() {
  uart_init(MYUBRR);  // Initialize the UART
  i2c_init();         // Initialize I2C

  tof(1);

  // Set pin modes using direct port manipulation
  DDRB |= (1 << DDB0) | (1 << DDB1) | (1 << DDB2); // STEP_PIN_1, DIR_PIN_1,
      ENA_PIN_1
  DDRB |= (1 << DDB3) | (1 << DDB4) | (1 << DDB5); // STEP_PIN_2, DIR_PIN_2,
      ENA_PIN_2

  // Enable pins
  PORTB &= ~(1 << PORTB2); // ENA_PIN_1 LOW
  PORTB &= ~(1 << PORTB5); // ENA_PIN_2 LOW

  // Move to the starting position
  PORTB &= ~(1 << PORTB4); // DIR_PIN_2 LOW
  for (int i = 0; i < XZ_REV / 2; i++) {
    PORTB |= (1 << PORTB3); // STEP_PIN_2 HIGH
    PORTB &= ~(1 << PORTB3); // STEP_PIN_2 LOW
    delayMicroseconds(50);
  }

  uart_print("VL53L0X test with Stepper Motor\n");
}

void loop() {
  for (int i = 0; i < XY_REV; i++) {
    if (i % 2 == 0) {
      PORTB |= (1 << PORTB4); // DIR_PIN_2 HIGH
    } else {
      PORTB &= ~(1 << PORTB4); // DIR_PIN_2 LOW
    }
    for (int j = 0; j < XZ_REV; j++) {
      i2c_start();
      i2c_write(0x29 << 1); // VL53L0X address
      i2c_write(0x00); // Register to read
      i2c_start();
      i2c_write((0x29 << 1) | 1); // Read mode
```

```
      uint8_t range = i2c_read_nack();
      i2c_stop();

      if (i % 2 == 0) {
        if (range != 255) {  // Check if range is valid
          measurements[j][0] = j * ANGLE_INCREMENT - 45;
          measurements[j][1] = i * ANGLE_INCREMENT;
          measurements[j][2] = range;
        } else {
          measurements[j][0] = j * ANGLE_INCREMENT - 45;
          measurements[j][1] = i * ANGLE_INCREMENT;
          measurements[j][2] = 10000;
        }
      } else {
        if (range != 255) {  // Check if range is valid
          measurements[XZ_REV - j - 1][0] = j * ANGLE_INCREMENT - 45;
          measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
          measurements[XZ_REV - j - 1][2] = range;
        } else {
          measurements[XZ_REV - j - 1][0] = j * ANGLE_INCREMENT - 45;
          measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
          measurements[XZ_REV - j - 1][2] = 10000;
        }
      }
      PORTB |= (1 << PORTB3); // STEP_PIN_2 HIGH
      PORTB &= ~(1 << PORTB3); // STEP_PIN_2 LOW
    }
    uart_print("aaa[");
    for (int k = 0; k < XZ_REV; k++) {
      uart_transmit('[');
      uart_print(measurements[k][0]);
      uart_transmit(',');
      uart_print(measurements[k][1]);
      uart_transmit(',');
      uart_print(measurements[k][2]);
      uart_transmit(']');
      if (k < XZ_REV - 1) uart_transmit(',');
    }
    uart_print("]\n");
    PORTB |= (1 << PORTB0); // STEP_PIN_1 HIGH
    PORTB &= ~(1 << PORTB0); // STEP_PIN_1 LOW
  }
}
```

6.2 C++ Code for vl53l0x.h library

To use the specific VL53L0X sensor, the manufacturer (Pololu Cooperation - A renowned company for sensor manufacturing) had implemented some specific bit sequences for I2C communication between the microcontroller and the sensor. Therefore they have recommended to use their own Open-Source and free to use pure C-AVR library to use that sensor. Therefore we implemented only that part as a usage of external library according to the recommendation of the manufacturer. But here we have included the complete source code for that sensor data reading.

```
// Copyright (c) 2017-2022 Pololu Corporation.

// For more information, see
// https://www.pololu.com/
// https://forum.pololu.com/

// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without
// restriction, including without limitation the rights to use,
// copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the
// Software is furnished to do so, subject to the following
// conditions:

// The above copyright notice and this permission notice shall be
// included in all copies or substantial portions of the Software.

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
// EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
// OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
// WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
// FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
// OTHER DEALINGS IN THE SOFTWARE.

// ================================================================

// Most of the functionality of this library is based on the VL53L0X API
// provided by ST (STSW-IMG005), and some of the explanatory comments are quoted
// or paraphrased from the API source code, API user manual (UM2039), and the
// VL53L0X datasheet.

#include "VL53L0X.h"
#include <Wire.h>

// Defines ////////////////////////////////////////////////////////////////////

// The Arduino two-wire interface uses a 7-bit number for the address,
// and sets the last bit correctly based on reads and writes
#define ADDRESS_DEFAULT 0b0101001

// Record the current time to check an upcoming timeout against
#define startTimeout() (timeout_start_ms = millis())

// Check if timeout is enabled (set to nonzero value) and has expired
#define checkTimeoutExpired() (io_timeout > 0 && ((uint16_t)(millis() -
    timeout_start_ms) > io_timeout))

// Decode VCSEL (vertical cavity surface emitting laser) pulse period in PCLKs
// from register value
// based on VL53L0X_decode_vcsel_period()
#define decodeVcselPeriod(reg_val)      (((reg_val) + 1) << 1)

// Encode VCSEL pulse period register value from period in PCLKs
// based on VL53L0X_encode_vcsel_period()
#define encodeVcselPeriod(period_pclks) (((period_pclks) >> 1) - 1)
```

```

// Calculate macro period in *nanoseconds* from VCSEL period in PCLKs
// based on VL53L0X_calc_macro_period_ps()
// PLL_period_ps = 1655; macro_period_vclks = 2304
#define calcMacroPeriod(vcsel_period_pclks) ((((uint32_t)2304 * (vcsel_period_pclks)
    * 1655) + 500) / 1000)

// Constructors ////////////////////////////////////////////////////////////////

VL53L0X::VL53L0X()
  : bus(&Wire)
  , address(ADDRESS_DEFAULT)
  , io_timeout(0) // no timeout
  , did_timeout(false)
{
}

// Public Methods //////////////////////////////////////////////////////////////

void VL53L0X::setAddress(uint8_t new_addr)
{
  writeReg(I2C_SLAVE_DEVICE_ADDRESS, new_addr & 0x7F);
  address = new_addr;
}

// Initialize sensor using sequence based on VL53L0X_DataInit(),
// VL53L0X_StaticInit(), and VL53L0X_PerformRefCalibration().
// This function does not perform reference SPAD calibration
// (VL53L0X_PerformRefSpadManagement()), since the API user manual says that it
// is performed by ST on the bare modules; it seems like that should work well
// enough unless a cover glass is added.
// If io_2v8 (optional) is true or not given, the sensor is configured for 2V8
// mode.
bool VL53L0X::init(bool io_2v8)
{
  // check model ID register (value specified in datasheet)
  if (readReg(IDENTIFICATION_MODEL_ID) != 0xEE) { return false; }

  // VL53L0X_DataInit() begin

  // sensor uses 1V8 mode for I/O by default; switch to 2V8 mode if necessary
  if (io_2v8)
  {
    writeReg(VHV_CONFIG_PAD_SCL_SDA__EXTSUP_HV,
      readReg(VHV_CONFIG_PAD_SCL_SDA__EXTSUP_HV) | 0x01); // set bit 0
  }

  // "Set I2C standard mode"
  writeReg(0x88, 0x00);

  writeReg(0x80, 0x01);
  writeReg(0xFF, 0x01);
  writeReg(0x00, 0x00);
  stop_variable = readReg(0x91);
  writeReg(0x00, 0x01);
  writeReg(0xFF, 0x00);
  writeReg(0x80, 0x00);

  // disable SIGNAL_RATE_MSRC (bit 1) and SIGNAL_RATE_PRE_RANGE (bit 4) limit checks
  writeReg(MSRC_CONFIG_CONTROL, readReg(MSRC_CONFIG_CONTROL) | 0x12);

  // set final range signal rate limit to 0.25 MCPS (million counts per second)
  setSignalRateLimit(0.25);

  writeReg(SYSTEM_SEQUENCE_CONFIG, 0xFF);

  // VL53L0X_DataInit() end

```

```
// VL53L0X_StaticInit() begin

uint8_t spad_count;
bool spad_type_is_aperture;
if (!getSpadInfo(&spad_count, &spad_type_is_aperture)) { return false; }

// The SPAD map (RefGoodSpadMap) is read by VL53L0X_get_info_from_device() in
// the API, but the same data seems to be more easily readable from
// GLOBAL_CONFIG_SPAD_ENABLES_REF_0 through _6, so read it from there
uint8_t ref_spad_map[6];
readMulti(GLOBAL_CONFIG_SPAD_ENABLES_REF_0, ref_spad_map, 6);

// -- VL53L0X_set_reference_spads() begin (assume NVM values are valid)

writeReg(0xFF, 0x01);
writeReg(DYNAMIC_SPAD_REF_EN_START_OFFSET, 0x00);
writeReg(DYNAMIC_SPAD_NUM_REQUESTED_REF_SPAD, 0x2C);
writeReg(0xFF, 0x00);
writeReg(GLOBAL_CONFIG_REF_EN_START_SELECT, 0xB4);

uint8_t first_spad_to_enable = spad_type_is_aperture ? 12 : 0; // 12 is the first
    aperture spad
uint8_t spads_enabled = 0;

for (uint8_t i = 0; i < 48; i++)
{
  if (i < first_spad_to_enable || spads_enabled == spad_count)
  {
    // This bit is lower than the first one that should be enabled, or
    // (reference_spad_count) bits have already been enabled, so zero this bit
    ref_spad_map[i / 8] &= ~(1 << (i % 8));
  }
  else if ((ref_spad_map[i / 8] >> (i % 8)) & 0x1)
  {
    spads_enabled++;
  }
}

writeMulti(GLOBAL_CONFIG_SPAD_ENABLES_REF_0, ref_spad_map, 6);

// -- VL53L0X_set_reference_spads() end

// -- VL53L0X_load_tuning_settings() begin
// DefaultTuningSettings from vl53l0x_tuning.h

writeReg(0xFF, 0x01);
writeReg(0x00, 0x00);

writeReg(0xFF, 0x00);
writeReg(0x09, 0x00);
writeReg(0x10, 0x00);
writeReg(0x11, 0x00);

writeReg(0x24, 0x01);
writeReg(0x25, 0xFF);
writeReg(0x75, 0x00);

writeReg(0xFF, 0x01);
writeReg(0x4E, 0x2C);
writeReg(0x48, 0x00);
writeReg(0x30, 0x20);

writeReg(0xFF, 0x00);
writeReg(0x30, 0x09);
writeReg(0x54, 0x00);
writeReg(0x31, 0x04);
writeReg(0x32, 0x03);
writeReg(0x40, 0x83);
```

```
    writeReg(0x46, 0x25);
    writeReg(0x60, 0x00);
    writeReg(0x27, 0x00);
    writeReg(0x50, 0x06);
    writeReg(0x51, 0x00);
    writeReg(0x52, 0x96);
    writeReg(0x56, 0x08);
    writeReg(0x57, 0x30);
    writeReg(0x61, 0x00);
    writeReg(0x62, 0x00);
    writeReg(0x64, 0x00);
    writeReg(0x65, 0x00);
    writeReg(0x66, 0xA0);

    writeReg(0xFF, 0x01);
    writeReg(0x22, 0x32);
    writeReg(0x47, 0x14);
    writeReg(0x49, 0xFF);
    writeReg(0x4A, 0x00);

    writeReg(0xFF, 0x00);
    writeReg(0x7A, 0x0A);
    writeReg(0x7B, 0x00);
    writeReg(0x78, 0x21);

    writeReg(0xFF, 0x01);
    writeReg(0x23, 0x34);
    writeReg(0x42, 0x00);
    writeReg(0x44, 0xFF);
    writeReg(0x45, 0x26);
    writeReg(0x46, 0x05);
    writeReg(0x40, 0x40);
    writeReg(0x0E, 0x06);
    writeReg(0x20, 0x1A);
    writeReg(0x43, 0x40);

    writeReg(0xFF, 0x00);
    writeReg(0x34, 0x03);
    writeReg(0x35, 0x44);

    writeReg(0xFF, 0x01);
    writeReg(0x31, 0x04);
    writeReg(0x4B, 0x09);
    writeReg(0x4C, 0x05);
    writeReg(0x4D, 0x04);

    writeReg(0xFF, 0x00);
    writeReg(0x44, 0x00);
    writeReg(0x45, 0x20);
    writeReg(0x47, 0x08);
    writeReg(0x48, 0x28);
    writeReg(0x67, 0x00);
    writeReg(0x70, 0x04);
    writeReg(0x71, 0x01);
    writeReg(0x72, 0xFE);
    writeReg(0x76, 0x00);
    writeReg(0x77, 0x00);

    writeReg(0xFF, 0x01);
    writeReg(0x0D, 0x01);

    writeReg(0xFF, 0x00);
    writeReg(0x80, 0x01);
    writeReg(0x01, 0xF8);

    writeReg(0xFF, 0x01);
    writeReg(0x8E, 0x01);
    writeReg(0x00, 0x01);
```

```
  writeReg(0xFF, 0x00);
  writeReg(0x80, 0x00);

  // -- VL53L0X_load_tuning_settings() end

  // "Set interrupt config to new sample ready"
  // -- VL53L0X_SetGpioConfig() begin

  writeReg(SYSTEM_INTERRUPT_CONFIG_GPIO, 0x04);
  writeReg(GPIO_HV_MUX_ACTIVE_HIGH, readReg(GPIO_HV_MUX_ACTIVE_HIGH) & ~0x10); //
      active low
  writeReg(SYSTEM_INTERRUPT_CLEAR, 0x01);

  // -- VL53L0X_SetGpioConfig() end

  measurement_timing_budget_us = getMeasurementTimingBudget();

  // "Disable MSRC and TCC by default"
  // MSRC = Minimum Signal Rate Check
  // TCC = Target CentreCheck
  // -- VL53L0X_SetSequenceStepEnable() begin

  writeReg(SYSTEM_SEQUENCE_CONFIG, 0xE8);

  // -- VL53L0X_SetSequenceStepEnable() end

  // "Recalculate timing budget"
  setMeasurementTimingBudget(measurement_timing_budget_us);

  // VL53L0X_StaticInit() end

  // VL53L0X_PerformRefCalibration() begin (VL53L0X_perform_ref_calibration())

  // -- VL53L0X_perform_vhv_calibration() begin

  writeReg(SYSTEM_SEQUENCE_CONFIG, 0x01);
  if (!performSingleRefCalibration(0x40)) { return false; }

  // -- VL53L0X_perform_vhv_calibration() end

  // -- VL53L0X_perform_phase_calibration() begin

  writeReg(SYSTEM_SEQUENCE_CONFIG, 0x02);
  if (!performSingleRefCalibration(0x00)) { return false; }

  // -- VL53L0X_perform_phase_calibration() end

  // "restore the previous Sequence Config"
  writeReg(SYSTEM_SEQUENCE_CONFIG, 0xE8);

  // VL53L0X_PerformRefCalibration() end

  return true;
}

// Write an 8-bit register
void VL53L0X::writeReg(uint8_t reg, uint8_t value)
{
  bus->beginTransmission(address);
  bus->write(reg);
  bus->write(value);
  last_status = bus->endTransmission();
}

// Write a 16-bit register
void VL53L0X::writeReg16Bit(uint8_t reg, uint16_t value)
{
  bus->beginTransmission(address);
```

```
  bus->write(reg);
  bus->write((uint8_t)(value >> 8)); // value high byte
  bus->write((uint8_t)(value)); // value low byte
  last_status = bus->endTransmission();
}

// Write a 32-bit register
void VL53L0X::writeReg32Bit(uint8_t reg, uint32_t value)
{
  bus->beginTransmission(address);
  bus->write(reg);
  bus->write((uint8_t)(value >> 24)); // value highest byte
  bus->write((uint8_t)(value >> 16));
  bus->write((uint8_t)(value >>  8));
  bus->write((uint8_t)(value));       // value lowest byte
  last_status = bus->endTransmission();
}

// Read an 8-bit register
uint8_t VL53L0X::readReg(uint8_t reg)
{
  uint8_t value;

  bus->beginTransmission(address);
  bus->write(reg);
  last_status = bus->endTransmission();

  bus->requestFrom(address, (uint8_t)1);
  value = bus->read();

  return value;
}

// Read a 16-bit register
uint16_t VL53L0X::readReg16Bit(uint8_t reg)
{
  uint16_t value;

  bus->beginTransmission(address);
  bus->write(reg);
  last_status = bus->endTransmission();

  bus->requestFrom(address, (uint8_t)2);
  value  = (uint16_t)bus->read() << 8; // value high byte
  value |=           bus->read();      // value low byte

  return value;
}

// Read a 32-bit register
uint32_t VL53L0X::readReg32Bit(uint8_t reg)
{
  uint32_t value;

  bus->beginTransmission(address);
  bus->write(reg);
  last_status = bus->endTransmission();

  bus->requestFrom(address, (uint8_t)4);
  value  = (uint32_t)bus->read() << 24; // value highest byte
  value |= (uint32_t)bus->read() << 16;
  value |= (uint16_t)bus->read() <<  8;
  value |=           bus->read();       // value lowest byte

  return value;
}

// Write an arbitrary number of bytes from the given array to the sensor,
```

```
// starting at the given register
void VL53L0X::writeMulti(uint8_t reg, uint8_t const * src, uint8_t count)
{
  bus->beginTransmission(address);
  bus->write(reg);

  while (count-- > 0)
  {
    bus->write(*(src++));
  }

  last_status = bus->endTransmission();
}

// Read an arbitrary number of bytes from the sensor, starting at the given
// register, into the given array
void VL53L0X::readMulti(uint8_t reg, uint8_t * dst, uint8_t count)
{
  bus->beginTransmission(address);
  bus->write(reg);
  last_status = bus->endTransmission();

  bus->requestFrom(address, count);

  while (count-- > 0)
  {
    *(dst++) = bus->read();
  }
}

// Set the return signal rate limit check value in units of MCPS (mega counts
// per second). "This represents the amplitude of the signal reflected from the
// target and detected by the device"; setting this limit presumably determines
// the minimum measurement necessary for the sensor to report a valid reading.
// Setting a lower limit increases the potential range of the sensor but also
// seems to increase the likelihood of getting an inaccurate reading because of
// unwanted reflections from objects other than the intended target.
// Defaults to 0.25 MCPS as initialized by the ST API and this library.
bool VL53L0X::setSignalRateLimit(float limit_Mcps)
{
  if (limit_Mcps < 0 || limit_Mcps > 511.99) { return false; }

  // Q9.7 fixed point format (9 integer bits, 7 fractional bits)
  writeReg16Bit(FINAL_RANGE_CONFIG_MIN_COUNT_RATE_RTN_LIMIT, limit_Mcps * (1 << 7));
  return true;
}

// Get the return signal rate limit check value in MCPS
float VL53L0X::getSignalRateLimit()
{
  return (float)readReg16Bit(FINAL_RANGE_CONFIG_MIN_COUNT_RATE_RTN_LIMIT) / (1 << 7);
}

// Set the measurement timing budget in microseconds, which is the time allowed
// for one measurement; the ST API and this library take care of splitting the
// timing budget among the sub-steps in the ranging sequence. A longer timing
// budget allows for more accurate measurements. Increasing the budget by a
// factor of N decreases the range measurement standard deviation by a factor of
// sqrt(N). Defaults to about 33 milliseconds; the minimum is 20 ms.
// based on VL53L0X_set_measurement_timing_budget_micro_seconds()
bool VL53L0X::setMeasurementTimingBudget(uint32_t budget_us)
{
  SequenceStepEnables enables;
  SequenceStepTimeouts timeouts;

  uint16_t const StartOverhead     = 1910;
  uint16_t const EndOverhead        = 960;
  uint16_t const MsrcOverhead       = 660;
```

```
  uint16_t const TccOverhead        = 590;
  uint16_t const DssOverhead        = 690;
  uint16_t const PreRangeOverhead   = 660;
  uint16_t const FinalRangeOverhead = 550;

  uint32_t used_budget_us = StartOverhead + EndOverhead;

  getSequenceStepEnables(&enables);
  getSequenceStepTimeouts(&enables, &timeouts);

  if (enables.tcc)
  {
    used_budget_us += (timeouts.msrc_dss_tcc_us + TccOverhead);
  }

  if (enables.dss)
  {
    used_budget_us += 2 * (timeouts.msrc_dss_tcc_us + DssOverhead);
  }
  else if (enables.msrc)
  {
    used_budget_us += (timeouts.msrc_dss_tcc_us + MsrcOverhead);
  }

  if (enables.pre_range)
  {
    used_budget_us += (timeouts.pre_range_us + PreRangeOverhead);
  }

  if (enables.final_range)
  {
    used_budget_us += FinalRangeOverhead;

    // "Note that the final range timeout is determined by the timing
    // budget and the sum of all other timeouts within the sequence.
    // If there is no room for the final range timeout, then an error
    // will be set. Otherwise the remaining time will be applied to
    // the final range."

    if (used_budget_us > budget_us)
    {
      // "Requested timeout too big."
      return false;
    }

    uint32_t final_range_timeout_us = budget_us - used_budget_us;

    // set_sequence_step_timeout() begin
    // (SequenceStepId == VL53L0X_SEQUENCESTEP_FINAL_RANGE)

    // "For the final range timeout, the pre-range timeout
    //  must be added. To do this both final and pre-range
    //  timeouts must be expressed in macro periods MClks
    //  because they have different vcsel periods."

    uint32_t final_range_timeout_mclks =
      timeoutMicrosecondsToMclks(final_range_timeout_us,
                                 timeouts.final_range_vcsel_period_pclks);

    if (enables.pre_range)
    {
      final_range_timeout_mclks += timeouts.pre_range_mclks;
    }

    writeReg16Bit(FINAL_RANGE_CONFIG_TIMEOUT_MACROP_HI,
      encodeTimeout(final_range_timeout_mclks));

    // set_sequence_step_timeout() end
```

```
    measurement_timing_budget_us = budget_us; // store for internal reuse
  }
  return true;
}

// Get the measurement timing budget in microseconds
// based on VL53L0X_get_measurement_timing_budget_micro_seconds()
// in us
uint32_t VL53L0X::getMeasurementTimingBudget()
{
  SequenceStepEnables enables;
  SequenceStepTimeouts timeouts;

  uint16_t const StartOverhead     = 1910;
  uint16_t const EndOverhead        = 960;
  uint16_t const MsrcOverhead       = 660;
  uint16_t const TccOverhead        = 590;
  uint16_t const DssOverhead        = 690;
  uint16_t const PreRangeOverhead   = 660;
  uint16_t const FinalRangeOverhead = 550;

  // "Start and end overhead times always present"
  uint32_t budget_us = StartOverhead + EndOverhead;

  getSequenceStepEnables(&enables);
  getSequenceStepTimeouts(&enables, &timeouts);

  if (enables.tcc)
  {
    budget_us += (timeouts.msrc_dss_tcc_us + TccOverhead);
  }

  if (enables.dss)
  {
    budget_us += 2 * (timeouts.msrc_dss_tcc_us + DssOverhead);
  }
  else if (enables.msrc)
  {
    budget_us += (timeouts.msrc_dss_tcc_us + MsrcOverhead);
  }

  if (enables.pre_range)
  {
    budget_us += (timeouts.pre_range_us + PreRangeOverhead);
  }

  if (enables.final_range)
  {
    budget_us += (timeouts.final_range_us + FinalRangeOverhead);
  }

  measurement_timing_budget_us = budget_us; // store for internal reuse
  return budget_us;
}

// Set the VCSEL (vertical cavity surface emitting laser) pulse period for the
// given period type (pre-range or final range) to the given value in PCLKs.
// Longer periods seem to increase the potential range of the sensor.
// Valid values are (even numbers only):
//  pre:  12 to 18 (initialized default: 14)
//  final: 8 to 14 (initialized default: 10)
// based on VL53L0X_set_vcsel_pulse_period()
bool VL53L0X::setVcselPulsePeriod(vcselPeriodType type, uint8_t period_pclks)
{
  uint8_t vcsel_period_reg = encodeVcselPeriod(period_pclks);

  SequenceStepEnables enables;
```

```
  SequenceStepTimeouts timeouts;

  getSequenceStepEnables(&enables);
  getSequenceStepTimeouts(&enables, &timeouts);

  // "Apply specific settings for the requested clock period"
  // "Re-calculate and apply timeouts, in macro periods"

  // "When the VCSEL period for the pre or final range is changed,
  // the corresponding timeout must be read from the device using
  // the current VCSEL period, then the new VCSEL period can be
  // applied. The timeout then must be written back to the device
  // using the new VCSEL period.
  //
  // For the MSRC timeout, the same applies - this timeout being
  // dependant on the pre-range vcsel period."


  if (type == VcselPeriodPreRange)
  {
    // "Set phase check limits"
    switch (period_pclks)
    {
      case 12:
        writeReg(PRE_RANGE_CONFIG_VALID_PHASE_HIGH, 0x18);
        break;

      case 14:
        writeReg(PRE_RANGE_CONFIG_VALID_PHASE_HIGH, 0x30);
        break;

      case 16:
        writeReg(PRE_RANGE_CONFIG_VALID_PHASE_HIGH, 0x40);
        break;

      case 18:
        writeReg(PRE_RANGE_CONFIG_VALID_PHASE_HIGH, 0x50);
        break;

      default:
        // invalid period
        return false;
    }
    writeReg(PRE_RANGE_CONFIG_VALID_PHASE_LOW, 0x08);

    // apply new VCSEL period
    writeReg(PRE_RANGE_CONFIG_VCSEL_PERIOD, vcsel_period_reg);

    // update timeouts

    // set_sequence_step_timeout() begin
    // (SequenceStepId == VL53L0X_SEQUENCESTEP_PRE_RANGE)

    uint16_t new_pre_range_timeout_mclks =
      timeoutMicrosecondsToMclks(timeouts.pre_range_us, period_pclks);

    writeReg16Bit(PRE_RANGE_CONFIG_TIMEOUT_MACROP_HI,
      encodeTimeout(new_pre_range_timeout_mclks));

    // set_sequence_step_timeout() end

    // set_sequence_step_timeout() begin
    // (SequenceStepId == VL53L0X_SEQUENCESTEP_MSRC)

    uint16_t new_msrc_timeout_mclks =
      timeoutMicrosecondsToMclks(timeouts.msrc_dss_tcc_us, period_pclks);

    writeReg(MSRC_CONFIG_TIMEOUT_MACROP,
```

```
      (new_msrc_timeout_mclks > 256) ? 255 : (new_msrc_timeout_mclks - 1));

    // set_sequence_step_timeout() end
  }
  else if (type == VcselPeriodFinalRange)
  {
    switch (period_pclks)
    {
      case 8:
        writeReg(FINAL_RANGE_CONFIG_VALID_PHASE_HIGH, 0x10);
        writeReg(FINAL_RANGE_CONFIG_VALID_PHASE_LOW,  0x08);
        writeReg(GLOBAL_CONFIG_VCSEL_WIDTH, 0x02);
        writeReg(ALGO_PHASECAL_CONFIG_TIMEOUT, 0x0C);
        writeReg(0xFF, 0x01);
        writeReg(ALGO_PHASECAL_LIM, 0x30);
        writeReg(0xFF, 0x00);
        break;

      case 10:
        writeReg(FINAL_RANGE_CONFIG_VALID_PHASE_HIGH, 0x28);
        writeReg(FINAL_RANGE_CONFIG_VALID_PHASE_LOW,  0x08);
        writeReg(GLOBAL_CONFIG_VCSEL_WIDTH, 0x03);
        writeReg(ALGO_PHASECAL_CONFIG_TIMEOUT, 0x09);
        writeReg(0xFF, 0x01);
        writeReg(ALGO_PHASECAL_LIM, 0x20);
        writeReg(0xFF, 0x00);
        break;

      case 12:
        writeReg(FINAL_RANGE_CONFIG_VALID_PHASE_HIGH, 0x38);
        writeReg(FINAL_RANGE_CONFIG_VALID_PHASE_LOW,  0x08);
        writeReg(GLOBAL_CONFIG_VCSEL_WIDTH, 0x03);
        writeReg(ALGO_PHASECAL_CONFIG_TIMEOUT, 0x08);
        writeReg(0xFF, 0x01);
        writeReg(ALGO_PHASECAL_LIM, 0x20);
        writeReg(0xFF, 0x00);
        break;

      case 14:
        writeReg(FINAL_RANGE_CONFIG_VALID_PHASE_HIGH, 0x48);
        writeReg(FINAL_RANGE_CONFIG_VALID_PHASE_LOW,  0x08);
        writeReg(GLOBAL_CONFIG_VCSEL_WIDTH, 0x03);
        writeReg(ALGO_PHASECAL_CONFIG_TIMEOUT, 0x07);
        writeReg(0xFF, 0x01);
        writeReg(ALGO_PHASECAL_LIM, 0x20);
        writeReg(0xFF, 0x00);
        break;

      default:
        // invalid period
        return false;
    }

    // apply new VCSEL period
    writeReg(FINAL_RANGE_CONFIG_VCSEL_PERIOD, vcsel_period_reg);

    // update timeouts

    // set_sequence_step_timeout() begin
    // (SequenceStepId == VL53L0X_SEQUENCESTEP_FINAL_RANGE)

    // "For the final range timeout, the pre-range timeout
    //  must be added. To do this both final and pre-range
    //  timeouts must be expressed in macro periods MClks
    //  because they have different vcsel periods."

    uint16_t new_final_range_timeout_mclks =
      timeoutMicrosecondsToMclks(timeouts.final_range_us, period_pclks);
```

```

    if (enables.pre_range)
    {
      new_final_range_timeout_mclks += timeouts.pre_range_mclks;
    }

    writeReg16Bit(FINAL_RANGE_CONFIG_TIMEOUT_MACROP_HI,
      encodeTimeout(new_final_range_timeout_mclks));

    // set_sequence_step_timeout end
  }
  else
  {
    // invalid type
    return false;
  }

  // "Finally, the timing budget must be re-applied"

  setMeasurementTimingBudget(measurement_timing_budget_us);

  // "Perform the phase calibration. This is needed after changing on vcsel period."
  // VL53L0X_perform_phase_calibration() begin

  uint8_t sequence_config = readReg(SYSTEM_SEQUENCE_CONFIG);
  writeReg(SYSTEM_SEQUENCE_CONFIG, 0x02);
  performSingleRefCalibration(0x0);
  writeReg(SYSTEM_SEQUENCE_CONFIG, sequence_config);

  // VL53L0X_perform_phase_calibration() end

  return true;
}

// Get the VCSEL pulse period in PCLKs for the given period type.
// based on VL53L0X_get_vcsel_pulse_period()
uint8_t VL53L0X::getVcselPulsePeriod(vcselPeriodType type)
{
  if (type == VcselPeriodPreRange)
  {
    return decodeVcselPeriod(readReg(PRE_RANGE_CONFIG_VCSEL_PERIOD));
  }
  else if (type == VcselPeriodFinalRange)
  {
    return decodeVcselPeriod(readReg(FINAL_RANGE_CONFIG_VCSEL_PERIOD));
  }
  else { return 255; }
}

// Start continuous ranging measurements. If period_ms (optional) is 0 or not
// given, continuous back-to-back mode is used (the sensor takes measurements as
// often as possible); otherwise, continuous timed mode is used, with the given
// inter-measurement period in milliseconds determining how often the sensor
// takes a measurement.
// based on VL53L0X_StartMeasurement()
void VL53L0X::startContinuous(uint32_t period_ms)
{
  writeReg(0x80, 0x01);
  writeReg(0xFF, 0x01);
  writeReg(0x00, 0x00);
  writeReg(0x91, stop_variable);
  writeReg(0x00, 0x01);
  writeReg(0xFF, 0x00);
  writeReg(0x80, 0x00);

  if (period_ms != 0)
  {
    // continuous timed mode
```

```

    // VL53L0X_SetInterMeasurementPeriodMilliSeconds() begin

    uint16_t osc_calibrate_val = readReg16Bit(OSC_CALIBRATE_VAL);

    if (osc_calibrate_val != 0)
    {
      period_ms *= osc_calibrate_val;
    }

    writeReg32Bit(SYSTEM_INTERMEASUREMENT_PERIOD, period_ms);

    // VL53L0X_SetInterMeasurementPeriodMilliSeconds() end

    writeReg(SYSRANGE_START, 0x04); // VL53L0X_REG_SYSRANGE_MODE_TIMED
  }
  else
  {
    // continuous back-to-back mode
    writeReg(SYSRANGE_START, 0x02); // VL53L0X_REG_SYSRANGE_MODE_BACKTOBACK
  }
}

// Stop continuous measurements
// based on VL53L0X_StopMeasurement()
void VL53L0X::stopContinuous()
{
  writeReg(SYSRANGE_START, 0x01); // VL53L0X_REG_SYSRANGE_MODE_SINGLESHOT

  writeReg(0xFF, 0x01);
  writeReg(0x00, 0x00);
  writeReg(0x91, 0x00);
  writeReg(0x00, 0x01);
  writeReg(0xFF, 0x00);
}

// Returns a range reading in millimeters when continuous mode is active
// (readRangeSingleMillimeters() also calls this function after starting a
// single-shot range measurement)
uint16_t VL53L0X::readRangeContinuousMillimeters()
{
  startTimeout();
  while ((readReg(RESULT_INTERRUPT_STATUS) & 0x07) == 0)
  {
    if (checkTimeoutExpired())
    {
      did_timeout = true;
      return 65535;
    }
  }

  // assumptions: Linearity Corrective Gain is 1000 (default);
  // fractional ranging is not enabled
  uint16_t range = readReg16Bit(RESULT_RANGE_STATUS + 10);

  writeReg(SYSTEM_INTERRUPT_CLEAR, 0x01);

  return range;
}

// Performs a single-shot range measurement and returns the reading in
// millimeters
// based on VL53L0X_PerformSingleRangingMeasurement()
uint16_t VL53L0X::readRangeSingleMillimeters()
{
  writeReg(0x80, 0x01);
  writeReg(0xFF, 0x01);
  writeReg(0x00, 0x00);
```

```
  writeReg(0x91, stop_variable);
  writeReg(0x00, 0x01);
  writeReg(0xFF, 0x00);
  writeReg(0x80, 0x00);

  writeReg(SYSRANGE_START, 0x01);

  // "Wait until start bit has been cleared"
  startTimeout();
  while (readReg(SYSRANGE_START) & 0x01)
  {
    if (checkTimeoutExpired())
    {
      did_timeout = true;
      return 65535;
    }
  }

  return readRangeContinuousMillimeters();
}

// Did a timeout occur in one of the read functions since the last call to
// timeoutOccurred()?
bool VL53L0X::timeoutOccurred()
{
  bool tmp = did_timeout;
  did_timeout = false;
  return tmp;
}

// Private Methods /////////////////////////////////////////////////////////////

// Get reference SPAD (single photon avalanche diode) count and type
// based on VL53L0X_get_info_from_device(),
// but only gets reference SPAD count and type
bool VL53L0X::getSpadInfo(uint8_t * count, bool * type_is_aperture)
{
  uint8_t tmp;

  writeReg(0x80, 0x01);
  writeReg(0xFF, 0x01);
  writeReg(0x00, 0x00);

  writeReg(0xFF, 0x06);
  writeReg(0x83, readReg(0x83) | 0x04);
  writeReg(0xFF, 0x07);
  writeReg(0x81, 0x01);

  writeReg(0x80, 0x01);

  writeReg(0x94, 0x6b);
  writeReg(0x83, 0x00);
  startTimeout();
  while (readReg(0x83) == 0x00)
  {
    if (checkTimeoutExpired()) { return false; }
  }
  writeReg(0x83, 0x01);
  tmp = readReg(0x92);

  *count = tmp & 0x7f;
  *type_is_aperture = (tmp >> 7) & 0x01;

  writeReg(0x81, 0x00);
  writeReg(0xFF, 0x06);
  writeReg(0x83, readReg(0x83)  & ~0x04);
  writeReg(0xFF, 0x01);
  writeReg(0x00, 0x01);
```

```cpp

  writeReg(0xFF, 0x00);
  writeReg(0x80, 0x00);

  return true;
}

// Get sequence step enables
// based on VL53L0X_GetSequenceStepEnables()
void VL53L0X::getSequenceStepEnables(SequenceStepEnables * enables)
{
  uint8_t sequence_config = readReg(SYSTEM_SEQUENCE_CONFIG);

  enables->tcc          = (sequence_config >> 4) & 0x1;
  enables->dss          = (sequence_config >> 3) & 0x1;
  enables->msrc         = (sequence_config >> 2) & 0x1;
  enables->pre_range    = (sequence_config >> 6) & 0x1;
  enables->final_range  = (sequence_config >> 7) & 0x1;
}

// Get sequence step timeouts
// based on get_sequence_step_timeout(),
// but gets all timeouts instead of just the requested one, and also stores
// intermediate values
void VL53L0X::getSequenceStepTimeouts(SequenceStepEnables const * enables,
    SequenceStepTimeouts * timeouts)
{
  timeouts->pre_range_vcsel_period_pclks = getVcselPulsePeriod(VcselPeriodPreRange);

  timeouts->msrc_dss_tcc_mclks = readReg(MSRC_CONFIG_TIMEOUT_MACROP) + 1;
  timeouts->msrc_dss_tcc_us =
    timeoutMclksToMicroseconds(timeouts->msrc_dss_tcc_mclks,
                               timeouts->pre_range_vcsel_period_pclks);

  timeouts->pre_range_mclks =
    decodeTimeout(readReg16Bit(PRE_RANGE_CONFIG_TIMEOUT_MACROP_HI));
  timeouts->pre_range_us =
    timeoutMclksToMicroseconds(timeouts->pre_range_mclks,
                               timeouts->pre_range_vcsel_period_pclks);

  timeouts->final_range_vcsel_period_pclks =
      getVcselPulsePeriod(VcselPeriodFinalRange);

  timeouts->final_range_mclks =
    decodeTimeout(readReg16Bit(FINAL_RANGE_CONFIG_TIMEOUT_MACROP_HI));

  if (enables->pre_range)
  {
    timeouts->final_range_mclks -= timeouts->pre_range_mclks;
  }

  timeouts->final_range_us =
    timeoutMclksToMicroseconds(timeouts->final_range_mclks,
                               timeouts->final_range_vcsel_period_pclks);
}

// Decode sequence step timeout in MCLKs from register value
// based on VL53L0X_decode_timeout()
// Note: the original function returned a uint32_t, but the return value is
// always stored in a uint16_t.
uint16_t VL53L0X::decodeTimeout(uint16_t reg_val)
{
  // format: "(LSByte * 2^MSByte) + 1"
  return (uint16_t)((reg_val & 0x00FF) <<
         (uint16_t)((reg_val & 0xFF00) >> 8)) + 1;
}

// Encode sequence step timeout register value from timeout in MCLKs
```

```
// based on VL53L0X_encode_timeout()
uint16_t VL53L0X::encodeTimeout(uint32_t timeout_mclks)
{
  // format: "(LSByte * 2^MSByte) + 1"

  uint32_t ls_byte = 0;
  uint16_t ms_byte = 0;

  if (timeout_mclks > 0)
  {
    ls_byte = timeout_mclks - 1;

    while ((ls_byte & 0xFFFFFF00) > 0)
    {
      ls_byte >>= 1;
      ms_byte++;
    }

    return (ms_byte << 8) | (ls_byte & 0xFF);
  }
  else { return 0; }
}

// Convert sequence step timeout from MCLKs to microseconds with given VCSEL period
    in PCLKs
// based on VL53L0X_calc_timeout_us()
uint32_t VL53L0X::timeoutMclksToMicroseconds(uint16_t timeout_period_mclks, uint8_t
    vcsel_period_pclks)
{
  uint32_t macro_period_ns = calcMacroPeriod(vcsel_period_pclks);

  return ((timeout_period_mclks * macro_period_ns) + 500) / 1000;
}

// Convert sequence step timeout from microseconds to MCLKs with given VCSEL period
    in PCLKs
// based on VL53L0X_calc_timeout_mclks()
uint32_t VL53L0X::timeoutMicrosecondsToMclks(uint32_t timeout_period_us, uint8_t
    vcsel_period_pclks)
{
  uint32_t macro_period_ns = calcMacroPeriod(vcsel_period_pclks);

  return (((timeout_period_us * 1000) + (macro_period_ns / 2)) / macro_period_ns);
}


// based on VL53L0X_perform_single_ref_calibration()
bool VL53L0X::performSingleRefCalibration(uint8_t vhv_init_byte)
{
  writeReg(SYSRANGE_START, 0x01 | vhv_init_byte); //
      VL53L0X_REG_SYSRANGE_MODE_START_STOP

  startTimeout();
  while ((readReg(RESULT_INTERRUPT_STATUS) & 0x07) == 0)
  {
    if (checkTimeoutExpired()) { return false; }
  }

  writeReg(SYSTEM_INTERRUPT_CLEAR, 0x01);

  writeReg(SYSRANGE_START, 0x00);

  return true;
}
```

6.3 C++ Code for plotting

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <ctime>
#include <json/json.h> // JSON library
#include <serial/serial.h> // Serial library
#include <gnuplot-iostream.h> // GNUplot library for plotting

#define PI 3.14159265358979323846

using namespace std;

// Structure to hold the 3D coordinates
struct Coordinate {
    float x;
    float y;
    float z;
};

// Function prototype for plotting 3D surface
void plot3DSurface(const vector<Coordinate>& coordinates);

int main() {
    // Open the serial port
    serial::Serial my_serial("COM8", 9600, serial::Timeout::simpleTimeout(1000));

    // Check if the serial port is open
    if (!my_serial.isOpen()) {
        cerr << "Failed to open serial port!" << endl;
        return 1;
    }

    vector<Coordinate> coordinates; // Vector to store the coordinates
    bool printed = false;
    int printed_lines = 0;
    bool can_update = false;

    // Loop to read data from the serial port
    while (!printed) {
        string line = my_serial.readline(); // Read a line from the serial port
        vector<Coordinate> temp_coords = str2list(line); // Convert the line to a
            list of coordinates

        // Check if the list is not empty
        if (!temp_coords.empty()) {
            // Check if the first coordinates are zero
            if (static_cast<int>(temp_coords[0].y) == 0 &&
                static_cast<int>(temp_coords[0].x) == 0) {
                can_update = true;
            }

            // Update the list of coordinates if allowed
            if (can_update) {
                coordinates.insert(coordinates.end(), temp_coords.begin(),
                    temp_coords.end());
                printed_lines++;

                // Once enough lines have been read, save the coordinates to a file
                    and plot the surface
                if (printed_lines >= 67) {
                    ofstream outfile("example.txt");
                    for (const auto& coord : coordinates) {
                        outfile << coord.x << " " << coord.y << " " << coord.z <<
                            "\n";
                    }
```

```
                    outfile.close();
                    plot3DSurface(coordinates);
                    printed = true;
                }
            }
        }
    }

    return 0;
}

// Function to convert a string to a list of coordinates
vector<Coordinate> str2list(const string& input_string) {
    string cleaned_string = input_string;
    cleaned_string.erase(remove(cleaned_string.begin(), cleaned_string.end(), '\r'),
        cleaned_string.end());
    cleaned_string.erase(remove(cleaned_string.begin(), cleaned_string.end(), '\n'),
        cleaned_string.end());

    vector<Coordinate> selected_dots;
    size_t pos = cleaned_string.find("aaa");

    // Check if the string contains the prefix "aaa"
    if (pos != string::npos) {
        string json_string = cleaned_string.substr(pos + 3);

        // Parse the JSON string
        Json::Value root;
        Json::CharReaderBuilder builder;
        builder["collectComments"] = false;
        JSONCPP_STRING errs;
        istringstream s(json_string);
        if (!parseFromStream(builder, s, &root, &errs)) {
            cerr << "Error decoding JSON: " << errs << endl;
            return {};
        }

        // Convert JSON data to a list of coordinates
        for (const auto& item : root) {
            float angle1 = item[0].asFloat() * (PI / 180);
            float angle2 = item[1].asFloat() * (PI / 180);
            float distance = item[2].asFloat();

            float x_temp = ((distance * cos(angle1) * cos(angle2)) + 2000) / 10;
            float y_temp = ((distance * cos(angle1) * sin(angle2)) + 2000) / 10;
            float z_temp = ((distance * sin(angle1)) + 2000) / 10;

            if (x_temp < 500 && y_temp < 500) {
                selected_dots.push_back({x_temp, y_temp, z_temp});
            }
        }
    }

    return selected_dots;
}

// Function to plot the 3D surface using GNUplot
void plot3DSurface(const vector<Coordinate>& coordinates) {
    Gnuplot gp;
    gp << "set title '3D Surface Plot'\n";
    gp << "set xlabel 'X-axis'\n";
    gp << "set ylabel 'Y-axis'\n";
    gp << "set zlabel 'Z-axis'\n";
    gp << "splot '-' with points pointtype 7 pointsize 1 notitle\n";

    // Send coordinates to GNUplot
    gp.send1d(boost::make_tuple(coordinates.begin(), coordinates.end(),
```

```
                [](const Coordinate& coord) { return make_tuple(coord.x, coord.y,
                    coord.z); }));
    gp << "e\n";
}
```

7 Photographs

7.1 Bare PCB

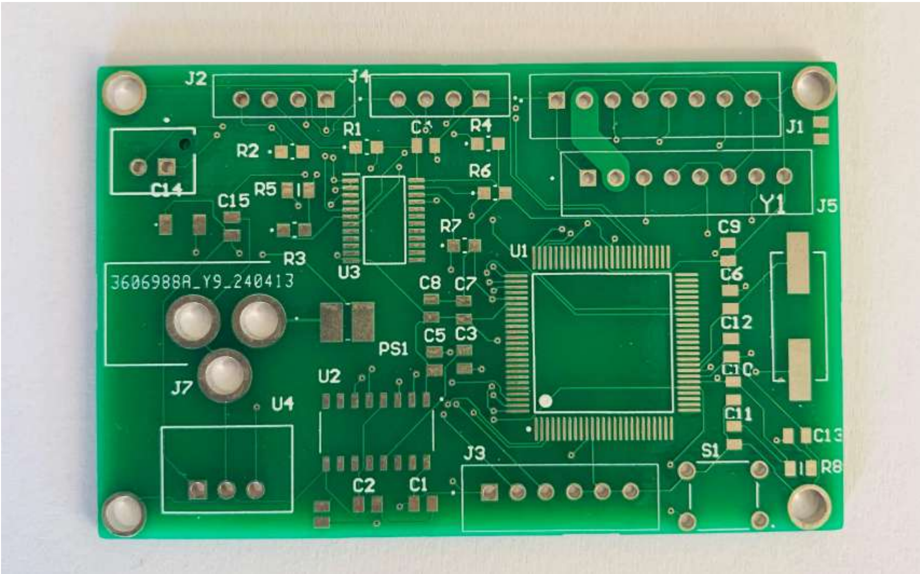


Figure 48: bare PCB

7.2 Soldered PCB

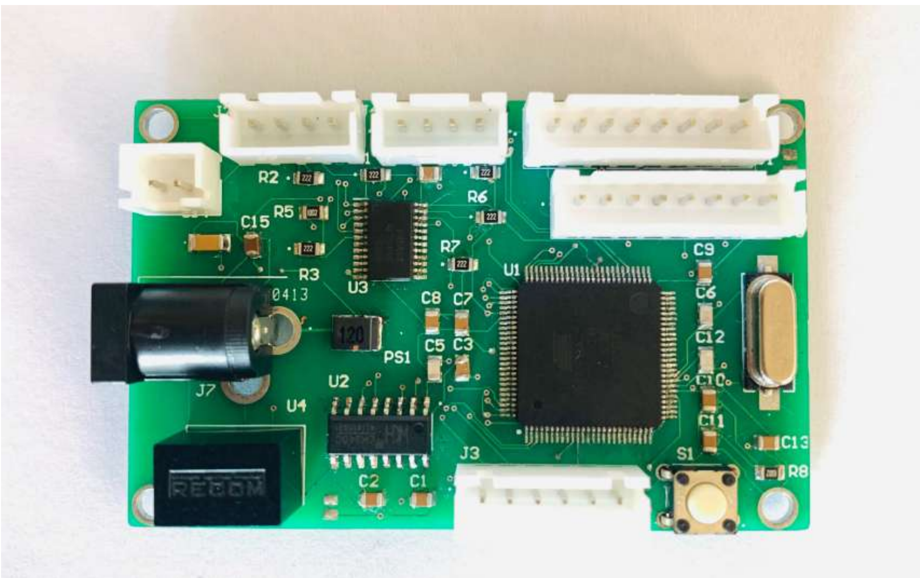


Figure 49: soldered PCB

7.3 Physically Built Enclosure

Figure 50: Enclosure

Figure 51: Enclosure

Figure 52: Enclosure

Figure 53: Enclosure

7.4 Testing

Figure 54: 12V supply

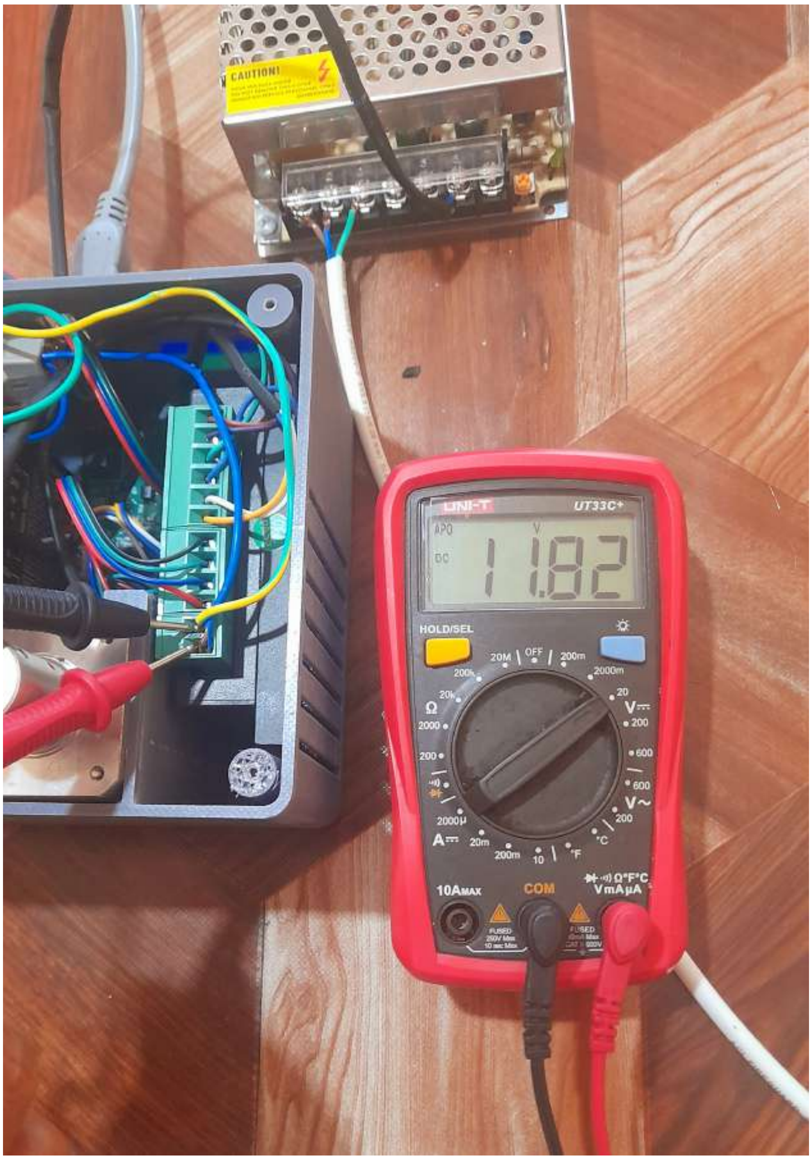


Figure 55: 12V supply for motor drivers

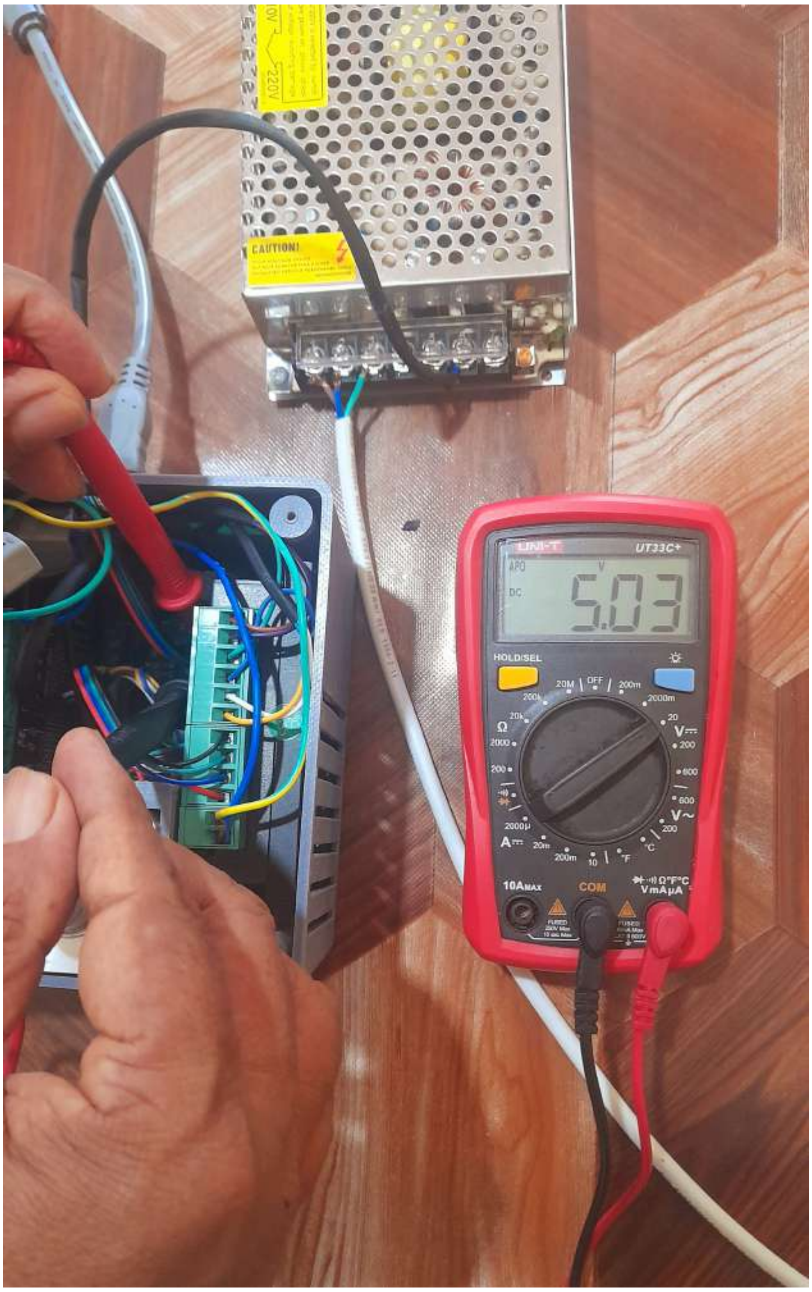


Figure 56: 5V supply for TOF sensors

7.5 System Integration

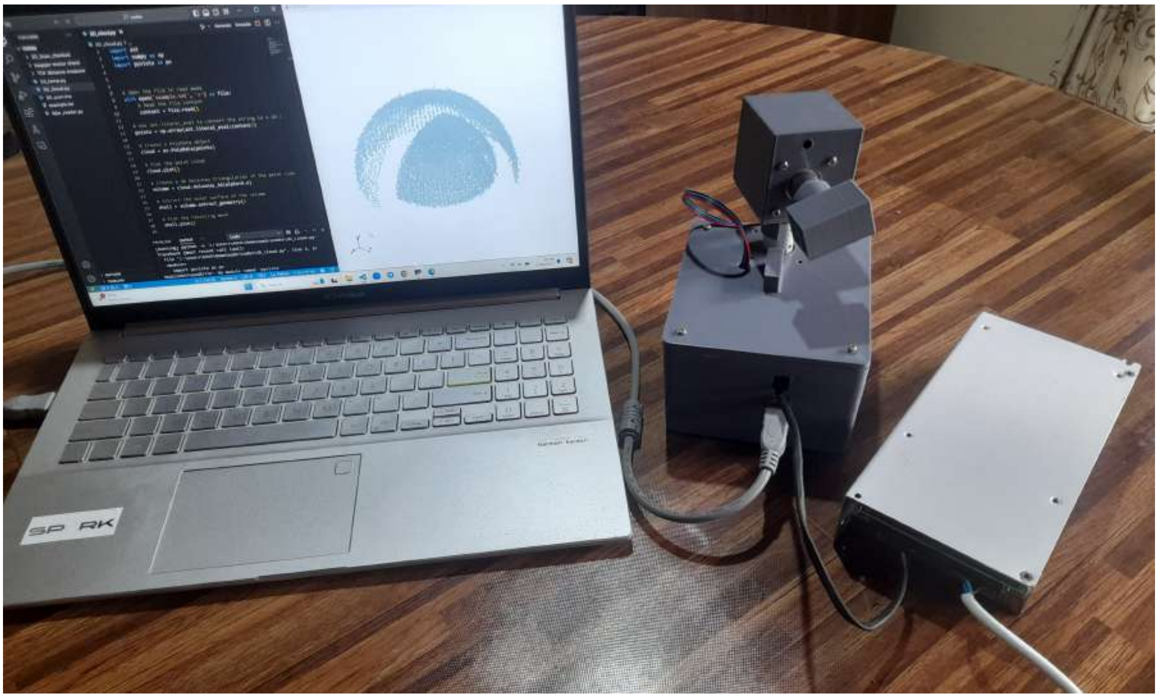


Figure 57: System Integration

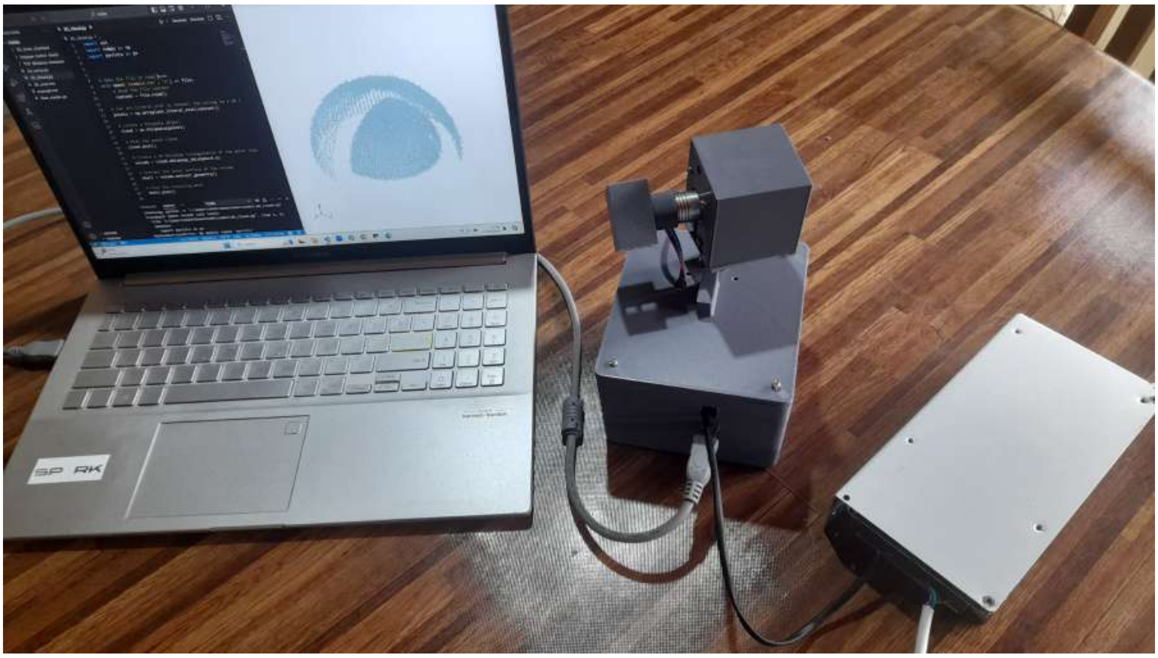


Figure 58: System Integration

Figure 59: System Integration

Figure 60: 3D scanner and power supply

Figure 61: System Integration

Figure 62: System Integration

Figure 63: TOF sensor 1

Figure 64: TOF sensor 2

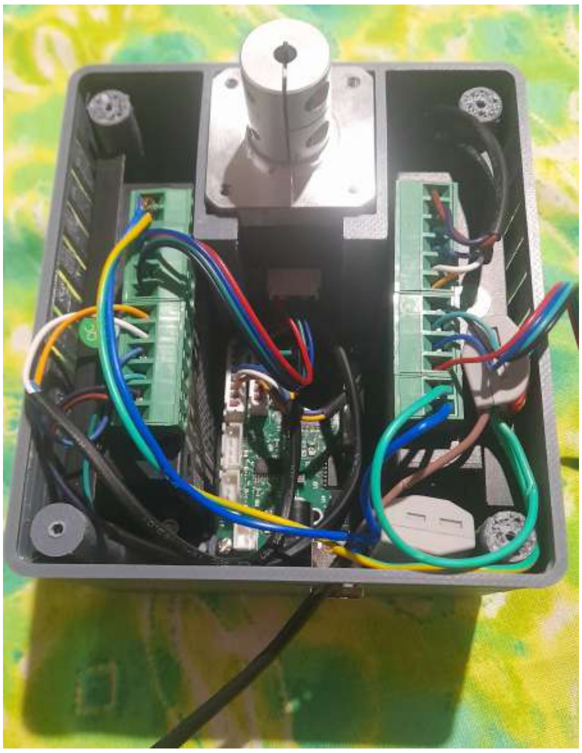


Figure 65: inside 3D scanner

Figure 66: inside 3D scanner

Figure 67: Power Supply

7.6 Simulation Results

We created a 3D point cloud using a test data set of a box. Video Link

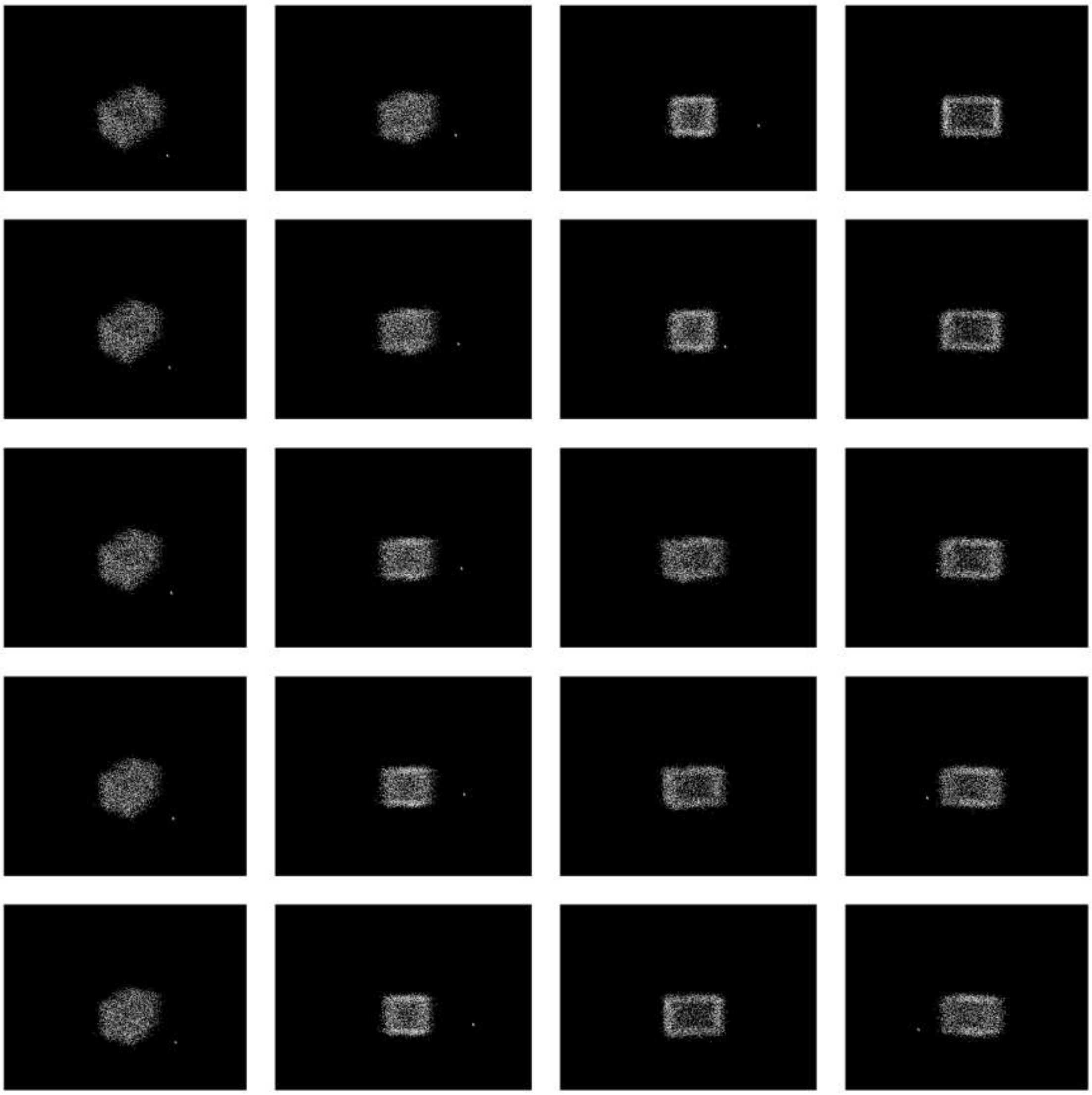


Figure 68: Plotting a box

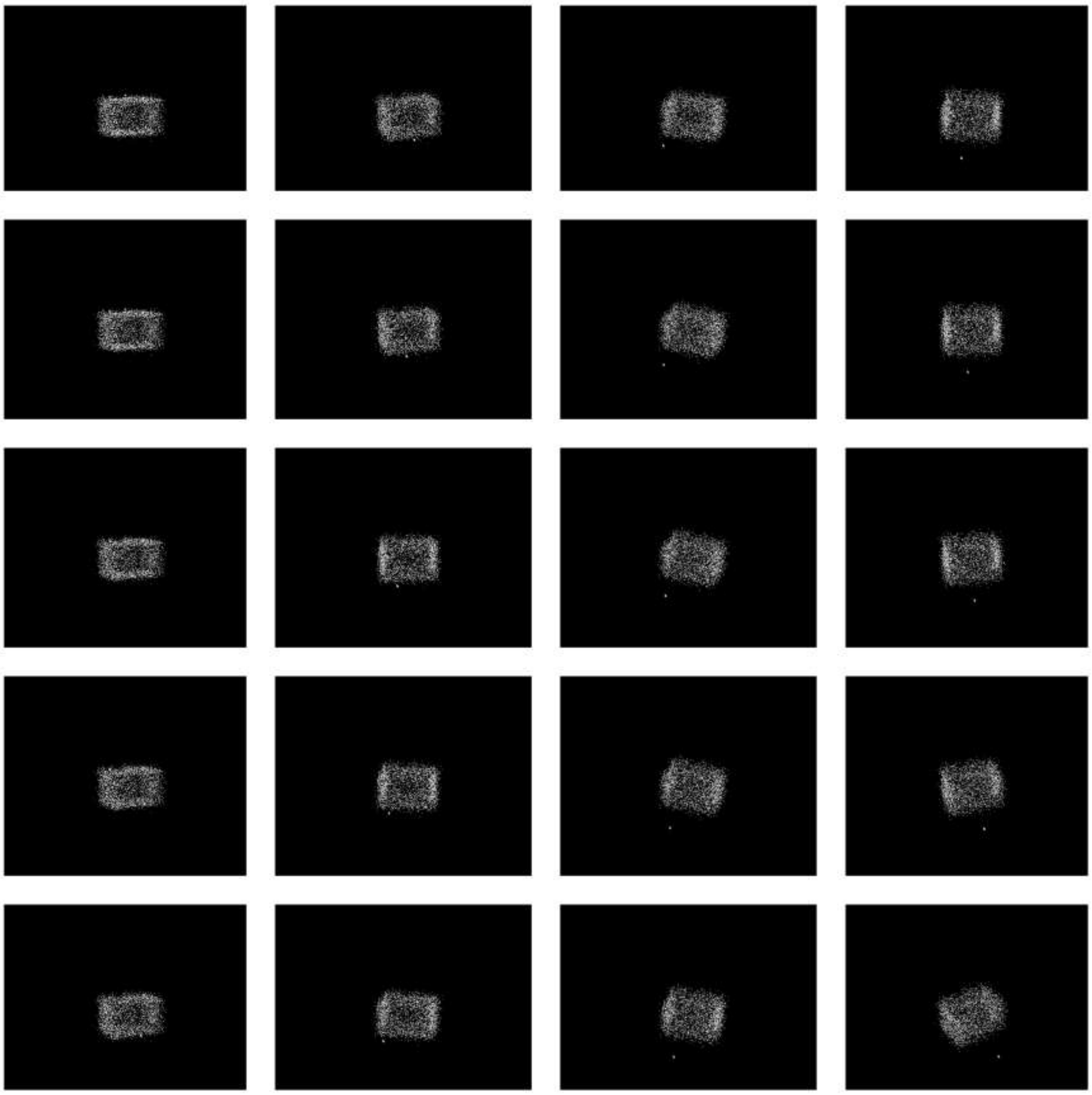


Figure 69: Plotting a box

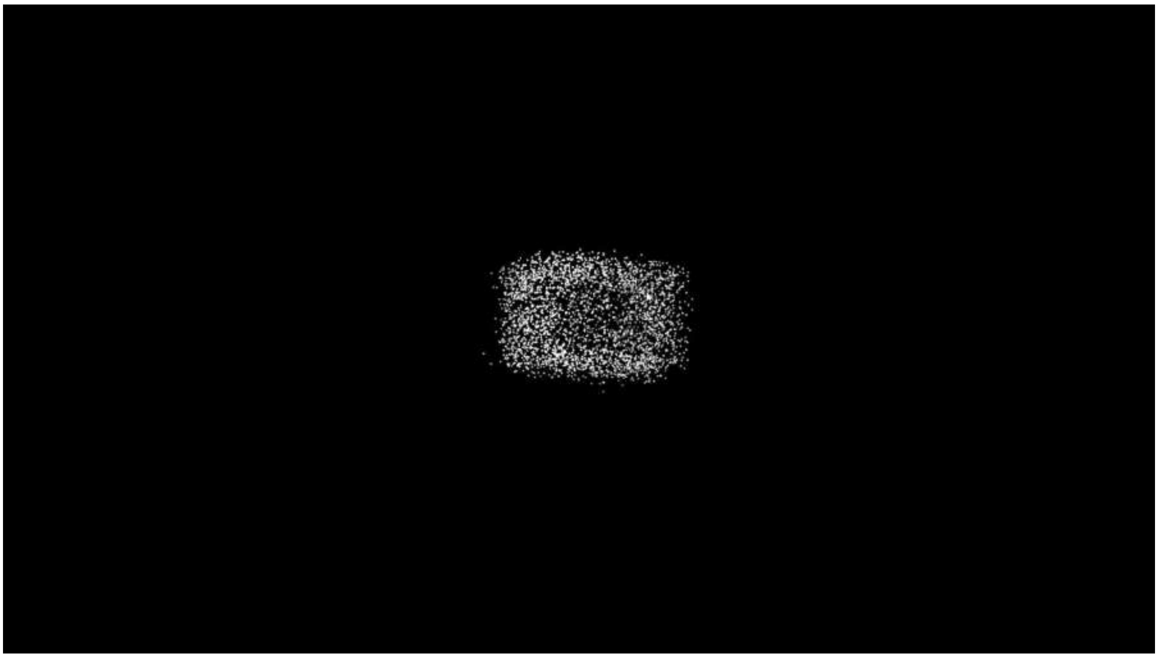


Figure 70: Plotting a box

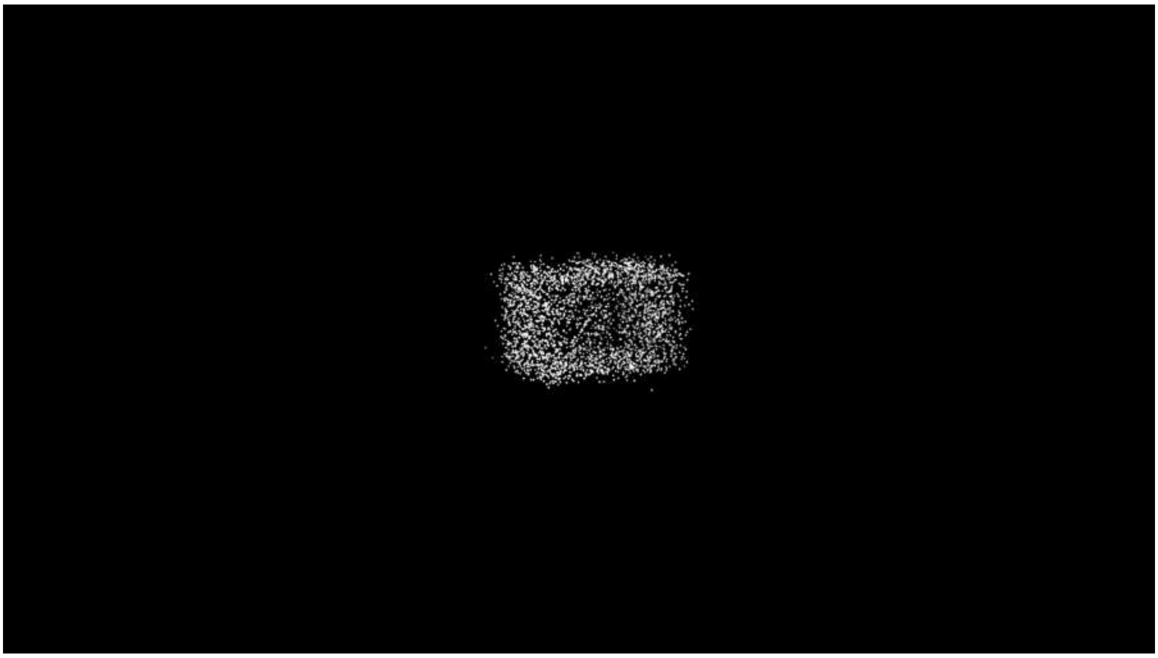


Figure 71: Plotting a box

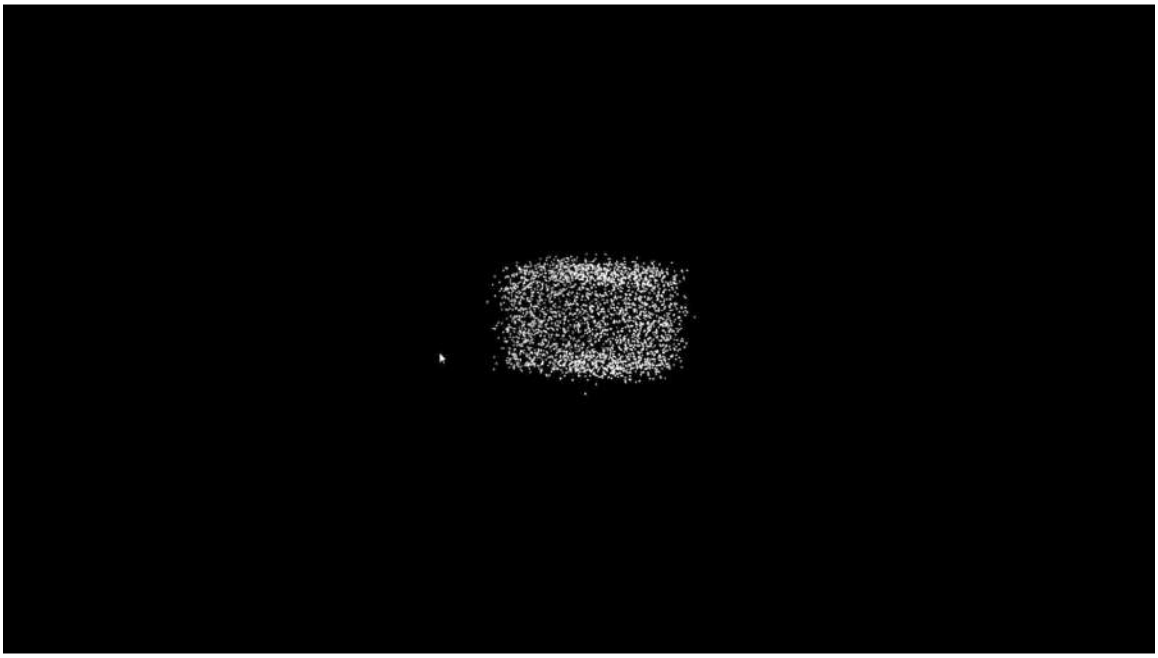


Figure 72: Plotting a box

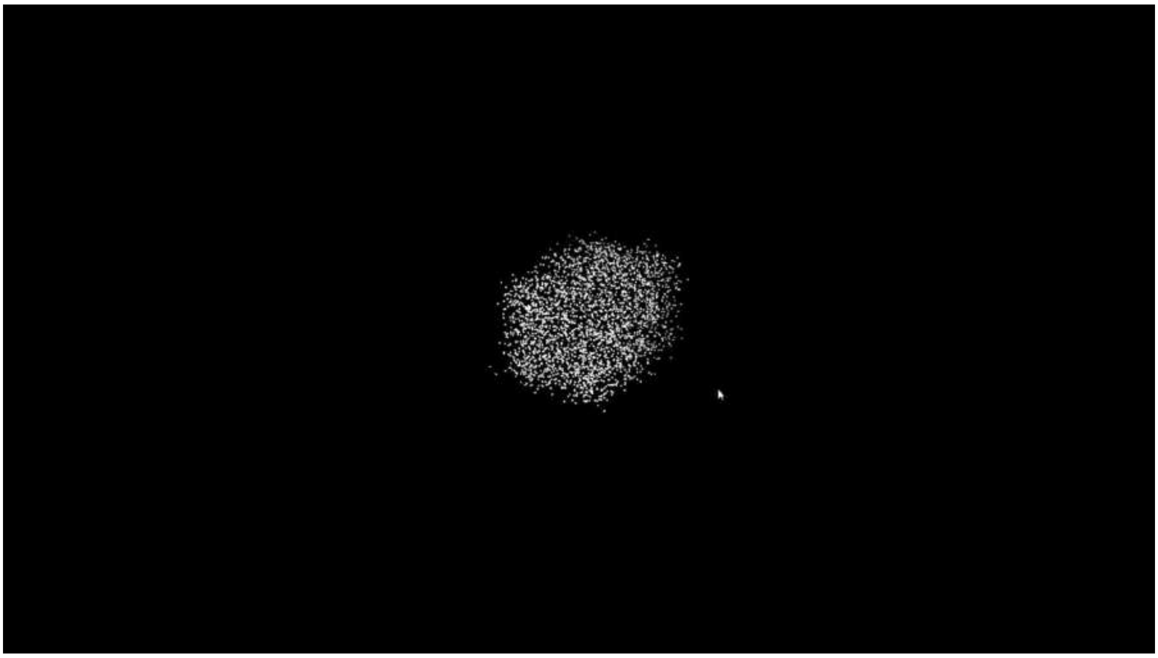


Figure 73: Plotting a box

8 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 Nuchter, "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

9 Signed Declaration

This document, detailing the design and development process of 3D mapping device has been thoroughly reviewed and cross-checked by an independant group to ensure accuracy and completeness.

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:

10 Appendix

10.1 Daily Log Entries

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

10.2 Datasheets

1. Atmega2560 Microcontroller
   https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega6
   datasheet.pdf
2. Nema 17 Stepper Motor
   https://pages.pbclinear.com/rs/909-BFY-775/images/Data-Sheet-Stepper-Motor-Support.
   pdf
3. TB6600-Micro Step Stepper-Motor-Driver
   https://www.watelectronics.com/tb6600-stepper-motor-driver-module/
4. TCA9548a I2C multiplexer
   https://www.ti.com/lit/ds/symlink/tca9548a.pdf
5. R_78CK_0_5-DC_DC Converter
   https://www.mouser.com/datasheet/2/468/R_78CK_0_5-3310288.pdf
6. CH340C USB to serial chip
   https://www.mpja.com/download/35227cpdata.pdf
7. Crystal Oscillator 16.000 MHz
   https://www.mouser.in/datasheet/2/741/LFXTAL027946Reel-995509.pdf

10.3 Wrong Programming Approach

This method is not suitable for Professional Engineers. Arduino and Python are not industry standards.

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

Arduino Code

```
#include <Wire.h>
#include <Adafruit_VL53L0X.h>

Adafruit_VL53L0X lox = Adafruit_VL53L0X();

void setup() {
  Serial.begin(115200);

  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1);
  }

  Serial.println(F("VL53L0X test"));
}

void loop() {
  VL53L0X_RangingMeasurementData_t measure;

  lox.rangingTest(&measure, false);

  if (measure.RangeStatus != 4) {  // phase failures have incorrect data
    Serial.print(F("Distance (mm): "));
    Serial.println(measure.RangeMilliMeter);
  } else {
    Serial.println(F("Out of range"));
  }

  delay(100);
}
```

Listing 1: Arduino Code for VL53L0X Sensor

Stepper Motor Check (11 March - 17 March)

Arduino Code

```
#include <AccelStepper.h>
#include <Wire.h>
#include <Adafruit_VL53L0X.h>

// Define stepper motor connections
#define STEP_PIN 10
#define DIR_PIN 11

#define STEP_PIN_2 12
#define DIR_PIN_2 13

#define STEPS_PER_REVOLUTION 200
#define ANGLE_INCREMENT 1.8

// Create a stepper object
AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);
AccelStepper stepper2(AccelStepper::DRIVER, STEP_PIN_2, DIR_PIN_2);

```

```
void setup() {
  // Set up the speed and acceleration of the stepper motor
  stepper.setMaxSpeed(1000);
  stepper.setAcceleration(500);

  stepper2.setMaxSpeed(1000);
  stepper2.setAcceleration(500);

  // Set the motor to run continuously
  stepper.moveTo(0);
  stepper2.moveTo(0);
}

void loop() {
  // Run the stepper motor
  stepper.run();
  stepper2.run();
}
```

Listing 2: Arduino Code for Two Stepper Motors with AccelStepper

3D Cloud Generation (18 March - 24 March)

Python Code

```
import ast
import numpy as np
import pyvista as pv

# Open the file in read mode
with open('example.txt', 'r') as file:
    # Read the file content
    content = file.read()

# Use ast.literal_eval to convert the string to a 2D list
points = np.array(ast.literal_eval(content))

# Create a PolyData object
cloud = pv.PolyData(points)

# Plot the point cloud
cloud.plot()

# Create a 3D Delaunay triangulation of the point cloud
volume = cloud.delaunay_3d(alpha=0.6)

# Extract the outer surface of the volume
shell = volume.extract_geometry()

# Plot the resulting mesh
shell.plot()
```

Listing 3: Python Code for 3D Delaunay Triangulation

2D Scan (25 March - 31 March)

Arduino Code

```
#include <Wire.h>
#include <Adafruit_VL53L0X.h>
#include <Arduino.h>
#include "A4988.h"

#define STEPS_PER_REVOLUTION 200
```

```
#define ANGLE_INCREMENT 1.8

int Step = 10; //GPIO14---D5 of Nodemcu--Step of stepper motor driver
int Dire  = 11; //GPIO2---D4 of Nodemcu--Direction of stepper motor driver
int Sleep = 14; //GPIO12---D6 of Nodemcu-Control Sleep Mode on A4988
int MS1 = 13; //GPIO13---D7 of Nodemcu--MS1 for A4988
int MS2 = 16; //GPIO16---D0 of Nodemcu--MS2 for A4988
int MS3 = 15; //GPIO15---D8 of Nodemcu--MS3 for A4988

//Motor Specs
const int spr = 200; //Steps per revolution
int RPM = 100; //Motor Speed in revolutions per minute
int Microsteps = 1; //Stepsize (1 for full steps, 2 for half steps, 4 for quarter
    steps, etc)

//Providing parameters for motor control
A4988 stepper(spr, Dire, Step, MS1, MS2, MS3);

Adafruit_VL53L0X lox = Adafruit_VL53L0X();
//Stepper stepper(STEPS_PER_REVOLUTION, 2, 3, 4, 5);  // Adjust pin numbers
    accordingly

const int num_measurements = STEPS_PER_REVOLUTION;
float measurements[num_measurements][2];

String arrayToString(int arr[], int size) {
  String result = "{";
  for (int i = 0; i < size; i++) {
    result += String(arr[i]);
    if (i < size - 1) {
      result += ", ";
    }
  }
  result += "}";
  return result;
}

void setup() {
  Serial.begin(9600);
  Serial1.begin(9600); // Use Serial1 for communication on Arduino Mega

  pinMode(Step, OUTPUT); //Step pin as output
  pinMode(Dire,  OUTPUT); //Direcction pin as output
  pinMode(Sleep,  OUTPUT); //Set Sleep OUTPUT Control button as output
  digitalWrite(Step, LOW); // Currently no stepper motor movement
  digitalWrite(Dire, LOW);

  // Set target motor RPM to and microstepping setting
  //stepper.begin(RPM, Microsteps);

  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1);
  }

  pinMode(13, OUTPUT);  // Set pin 13 as an output

//  stepper.setSpeed(500);  // Adjust the speed as needed

  Serial.println(F("VL53L0X test with Stepper Motor"));
}

void loop() {
  digitalWrite(12, HIGH);
  digitalWrite(Sleep, HIGH); //A logic high allows normal operation of the A4988 by
      removing from sleep
  stepper.rotate(360);

```

```
  for (int i = 0; i < num_measurements; i++) {
    // Rotate stepper motor by ANGLE_INCREMENT degrees
  //   stepper.step(ANGLE_INCREMENT);

    // Take distance measurement
    VL53L0X_RangingMeasurementData_t measure;
    lox.rangingTest(&measure, false);

    if (measure.RangeStatus != 4) {  // phase failures have incorrect data
      measurements[i][0] = i * ANGLE_INCREMENT;
      measurements[i][1] = measure.RangeMilliMeter;
    } else {
      measurements[i][0] = i * ANGLE_INCREMENT;
      measurements[i][1] = 10000;
    }

    // No delay or minimal delay between steps
  }

  // Print the 2D array after 360 degrees rotation
  Serial.print("aaa[");
  for (int i = 0; i < num_measurements; i++) {
    Serial.print("[");
    Serial.print(measurements[i][0]);
    Serial.print(",");
    Serial.print(measurements[i][1]);
    Serial.print("],");
  }
  Serial.println("]");
  //Serial.println(arrayToString());

  // Reset the stepper motor to its initial position
//  stepper.step(-360 * STEPS_PER_REVOLUTION / 360);
  digitalWrite(13, LOW);

  // Delay before starting a new measurement
  delay(5000);
}
```

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

3D scan (1 April - 7 April)

Arduino Code

```
#include <Wire.h>
#include <Adafruit_VL53L0X.h>

#define STEP_PIN_1 8
#define DIR_PIN_1 9
#define ENA_PIN_1 10

#define STEP_PIN_2 4
#define DIR_PIN_2 3
#define ENA_PIN_2 2

#define XY_REV 200
#define XZ_REV 50

#define ANGLE_INCREMENT 1.8

Adafruit_VL53L0X lox = Adafruit_VL53L0X();

float measurements[XZ_REV][3];

void setup() {
```

```
  Serial.begin(9600);  // Initialize the serial monitor

  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1);
  }
  pinMode(4,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(2,OUTPUT);

  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);

  digitalWrite(ENA_PIN_1, LOW);
  digitalWrite(ENA_PIN_2, LOW);

  Serial.println(F("VL53L0X test with Stepper Motor"));
}

void loop() {
  // put your main code here, to run repeatedly:
  for (int i=0; i < XY_REV; i++) {
    if (i%2 == 0) {
      digitalWrite(DIR_PIN_2,HIGH);
    } else {
      digitalWrite(DIR_PIN_2,LOW);
    }
    for (int j=0; j < XZ_REV; j++) {
      VL53L0X_RangingMeasurementData_t measure;
      lox.rangingTest(&measure, false);

      if (i%2 == 0) {
        if (measure.RangeStatus != 4) {  // phase failures have incorrect data
        measurements[j][0] = j * ANGLE_INCREMENT - 45;
        measurements[j][1] = i * ANGLE_INCREMENT;
        measurements[j][2] = measure.RangeMilliMeter;
      } else {
        measurements[j][0] = j * ANGLE_INCREMENT - 45;
        measurements[j][1] = i * ANGLE_INCREMENT;
        measurements[j][2] = 10000;
      }
      } else {
        if (measure.RangeStatus != 4) {  // phase failures have incorrect data
        measurements[XZ_REV - j -1][0] = j * ANGLE_INCREMENT - 45;
        measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
        measurements[XZ_REV - j - 1][2] = measure.RangeMilliMeter;
      } else {
        measurements[XZ_REV - j - 1][0] = j * ANGLE_INCREMENT - 45;
        measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
        measurements[XZ_REV - j - 1][2] = 10000;
      }
      }
      digitalWrite(STEP_PIN_2, HIGH);
      digitalWrite(STEP_PIN_2, LOW);
    }
    Serial.print("aaa[");
  for (int k = 0; k < XZ_REV; k++) {
    Serial.print("[");
    Serial.print(measurements[k][0]);
    Serial.print(",");
    Serial.print(measurements[k][1]);
    Serial.print("],");
    Serial.print(measurements[k][2]);
    Serial.print("],");
    }
  Serial.println("]");
  digitalWrite(STEP_PIN_1, HIGH);
```

```
  digitalWrite(STEP_PIN_1, LOW);
  }

}
```

Listing 5: Arduino Code for VL53L0X Sensor with Stepper Motors

Serial Communication - Receiver (22 April - 28 April)

Python Code

```python
import serial
import json
from math import cos, sin, pi
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

from datetime import datetime

# datetime object containing current date and time
now = datetime.now()

print("now =", now)

# dd/mm/YY H:M:S
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")

printed = False

def plot_lines(coordinates):
    x_values, y_values = zip(*coordinates)
    plt.plot(x_values, y_values, color='blue', linestyle='-', linewidth=2,
        label='Line')

    plt.title('Graph with Connected Line (No Dots)')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')

    # Set axis limits to always be 0 to 300
    #plt.xlim(0, 300)
    #plt.ylim(0, 300)

    # Set the aspect ratio to be equal
    plt.gca().set_aspect('equal', adjustable='box')

    plt.grid(True)
    plt.legend()
    plt.show()

def plot_dots(coordinates):
    x_values, y_values = zip(*coordinates)
    plt.scatter(x_values, y_values, color='blue', marker='o')
    plt.title('Graph with Dots')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')

    #plt.xlim(0, 300)
    #plt.ylim(0, 300)

    plt.gca().set_aspect('equal', adjustable='box')

    plt.grid(True)
    plt.show()


```

```
def plot_3d_surface(coordinates):
    x_coords = [coordinate[0] for coordinate in coordinates]
    y_coords = [coordinate[1] for coordinate in coordinates]
    z_coords = [coordinate[2] for coordinate in coordinates]

    # Convert coordinates to numpy arrays
    x = np.array(x_coords)
    y = np.array(y_coords)
    z = np.array(z_coords)

    # Create a 3D plot
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Plot the scatter plot
    ax.scatter(x, y, z, c='b', marker='o', s = 1)

    # Connect the points with lines
    for i in range(1, len(x)):
        ax.plot([x[i-1], x[i]], [y[i-1], y[i]], [z[i-1], z[i]], c='b')

    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')

    plt.show()

def plot_3d_wireframe(coordinates):
    x_coords = [coordinate[0] for coordinate in coordinates]
    y_coords = [coordinate[1] for coordinate in coordinates]
    z_coords = [coordinate[2] for coordinate in coordinates]

    # Convert coordinates to numpy arrays
    x = np.array(x_coords)
    y = np.array(y_coords)
    z = np.array(z_coords)

    # Create a 3D plot
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Plot the wireframe
    ax.plot_trisurf(x, y, z, linewidth=0, antialiased=False)

    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')

    plt.show()


def str2list(input_string):
    input_string = input_string.replace('\r', "")
    input_string = input_string.replace('\n', "")

    temp_buffer = []
    temp_text = ''
    selected_dots = []

    for stringData in input_string.split('\n'):
        temp_buffer.append(temp_text + stringData.split('\n')[0])
        temp_text = ''

        if temp_buffer[-1][:3] == "aaa":
            temp_text = ''
            build_temp = temp_buffer[-1][3:].rstrip('\n\r,')

            # Handle the case when the string ends with a trailing comma
```

```
            if build_temp[-2]:
                build_temp = build_temp[:-2]+build_temp[-1]

            try:
                nested_list = [list(map(float, innerList)) for innerList in
                    json.loads(build_temp)]
            except json.decoder.JSONDecodeError as e:
                print("Error decoding JSON:", e)
                print("Problematic data:", build_temp)
                return False

            for nested_item in nested_list:
                x_temp = ((nested_item[2] * cos(nested_item[0] * (pi / 180)) *
                    cos(nested_item[1] * (pi / 180))) + 2000) /10
                y_temp = ((nested_item[2] * cos(nested_item[0] * (pi / 180)) *
                    sin(nested_item[1] * (pi / 180))) + 2000) /10
                z_temp = ((nested_item[2] * sin(nested_item[0] * (pi / 180))) +
                    2000) /10

                if x_temp < 1000 and y_temp < 1000:
                    selected_dots.append([x_temp, y_temp,z_temp])
                    #print("Selected dots:", selected_dots)
            return selected_dots

    return False


class ReadLine:
    def init(self, s):
        self.buf = bytearray()
        self.s = s

    def readline(self):
        i = self.buf.find(b"\n")
        if i >= 0:
            r = self.buf[:i+1]
            self.buf = self.buf[i+1:]
            return r.decode("utf-8")

        while True:
            i = max(1, min(2048, self.s.in_waiting))
            data = self.s.read(i)
            i = data.find(b"\n")
            if i >= 0:
                r = self.buf + data[:i+1]
                self.buf[0:] = data[i+1:]
                return r.decode("utf-8")
            else:
                self.buf.extend(data)

ser = serial.Serial('COM3', 9600)
rl = ReadLine(ser)

can_update = False
list_for_3d = []

while not printed:
    line = rl.readline()
    #print("Received:", line)
    print_temp = str2list(line)
    if print_temp != False:
        #print(print_temp)
        if (int(print_temp[0][1]) == 0 and int(print_temp[0][0]) == 0):
            can_update = True
        if True:
            for i in print_temp:
                list_for_3d.append(i)
            print(len(list_for_3d)/50)
```

```
            if len(list_for_3d) >= 200*50:
                #plot_dots(print_temp)
                #plot_lines(print_temp)
                #print(len(list_for_3d))
                # Specify the file path
                file_path = "example" + dt_string + ".txt"

                # Open the file in write mode
                with open(file_path, 'w') as file:
                    # Write the text to the file
                    file.write(str(list_for_3d))

                print(f"Text saved to {file_path}")
                #print(list_for_3d)
                plot_3d_surface(list_for_3d)
                printed = True
```

Listing 6: Python Code for receiver