University of Moratuwa

Department of Electronic & Telecommunication Engineering

EN2160 - Electronic Design Realization

3D mapping with LIDAR

(Design Documentation)

Name	Index
Perera . P. D. P	210469G
Rajapakshe S.D.D.Z	210508D

(This report is submitted as a partial fulfillment of module EN2160)

(Date : 2024/06/24)

Contents

Abstract

In this project, we undertake the development of a 3D mapping system using LiDAR technology, aimed at providing high-resolution spatial data for applications in various fields such as autonomous navigation, environmental monitoring, and many short ranged applications which requires 3D data. The core of our system is the TFmini-S LiDAR sensor, renowned for its accuracy and reliability, interfaced with an ATmega328P microcontroller to handle data acquisition and processing. By integrating this setup with a stepper motor controlled via an A4988 driver, we achieve precise angular movement, enabling comprehensive 3D scanning capabilities.

Our approach leverages the robustness of the TFmini-S LiDAR sensor to capture distance measurements with high precision. These measurements are synchronized with the stepper motor's rotational data to construct a detailed point cloud representation of the scanned environment. The ATmega328P microcontroller serves as the primary processing unit, managing the acquisition of LiDAR data, controlling the stepper motor, and ensuring the seamless operation of the system.

A significant aspect of this project involves the customization of the LiDAR sensor's communication protocol. By using the FT232RL module, we establish a direct communication link with the ATmega328P, bypassing the need for additional libraries and thereby optimizing the system's performance. This direct approach facilitates real-time data processing and enhances the overall efficiency of the 3D mapping system.

The collected LiDAR data is processed to generate accurate 3D maps, which can be utilized in various applications. The data is visualized using specialized software tools, allowing for detailed analysis and interpretation of the spatial information. This project not only demonstrates the feasibility of constructing a cost-effective 3D mapping system using readily available components but also highlights the potential of such systems in contributing to advancements in autonomous technologies and spatial data analysis.

In conclusion, our project successfully integrates LiDAR technology with precise motor control to create a functional 3D mapping system. The methodologies and technologies employed showcase the capability of low-cost hardware to perform high-precision tasks, providing a foundation for future developments in the field of 3D spatial mapping.

Chapter 1

Introduction

This document provides an in-depth exploration of the design specifics for our 3D mapping project, which integrates a LiDAR sensor, stepper motors, and a variety of supplementary components. The project's design decisions were thoroughly considered and are substantiated by specific criteria to ensure that the final product meets all required performance standards.

Our design journey commenced with the careful selection of appropriate components, each chosen for its ability to contribute to the overall functionality and efficiency of the system. Following the component selection, we engaged in an iterative process of conceptual design, during which various approaches were evaluated. Through rigorous testing and analysis, we ultimately refined our concepts to identify and finalize the most effective solution.

In the subsequent sections of this report, you will find detailed discussions on the selections and justifications for each component utilized in the project. These sections delve into the reasoning behind our choices, highlighting how each component contributes to the project's overall objectives. From the precision of the LiDAR sensor to the reliability of the stepper motors and the integration of supporting components, every aspect of the design has been crafted to ensure optimal performance and robustness of the 3D mapping system.

By documenting the comprehensive design details and justifications, this report serves as a testament to the thoughtful and systematic approach taken to bring this 3D mapping project to fruition.

Objective

The primary objective of our 3D mapping project is to develop a high-precision, reliable, and efficient 3D mapping system specifically designed for integration with Autonomous Mobile Robots (AMRs), drones, and other similar platforms. Our system aims to deliver accurate spatial data for applications requiring short-range measurements (6-8 meters) with a high data rate and precision, all at an affordable price compared to current market solutions.

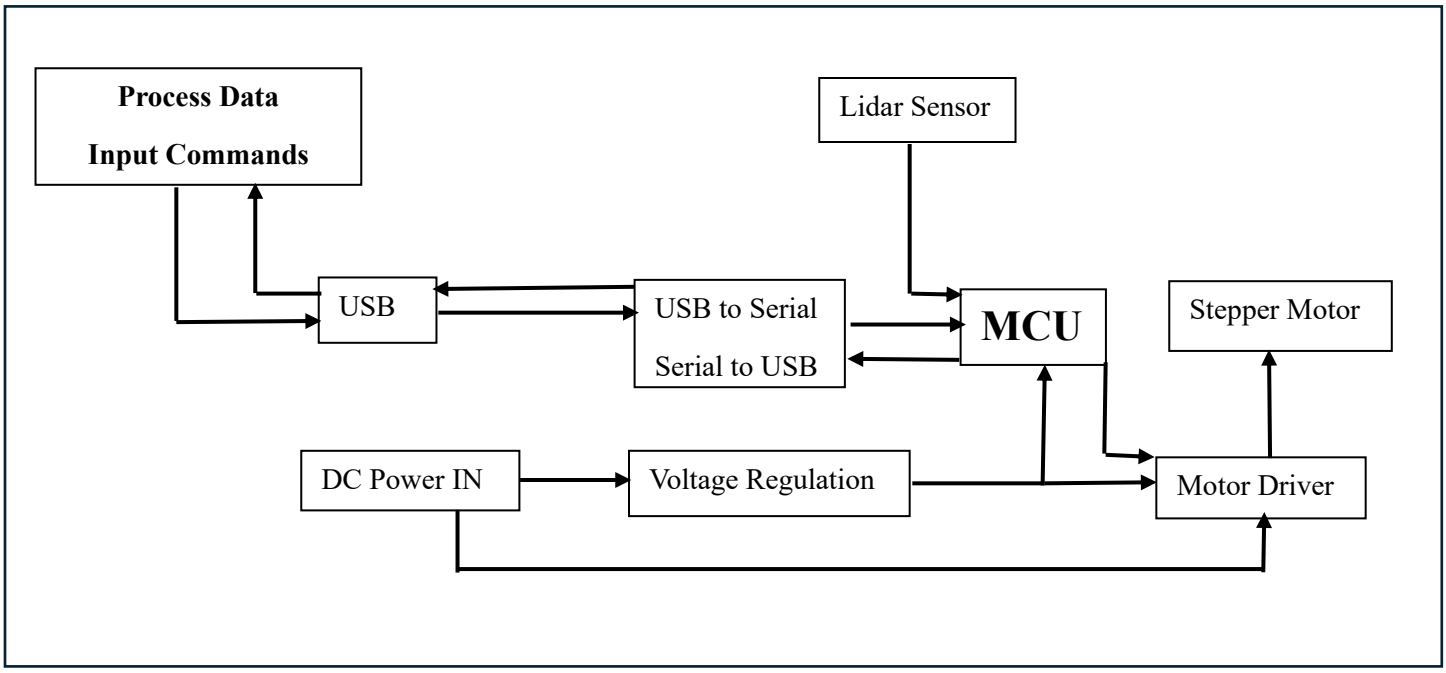
Our project is meticulously crafted to meet the diverse needs of our stakeholders, which include project team members, supervisors, end-users (such as robotics developers and drone operators), industry partners (robotics and drone manufacturers), regulatory authorities, the public, academic institutions, and competitors. We have thoroughly considered their interests, expectations, and potential contributions to ensure our product's success. Here’s how our product addresses their needs:

1. **Project Team Members and Supervisors:**
    - Ensuring a collaborative and transparent design process.
    - Delivering a technically robust solution that meets project goals.
2. **End-Users (Robotics Developers, Drone Operators):**
    - Providing accurate and reliable 3D spatial data for navigation and obstacle detection.
    - Ensuring user-friendly integration and operation with AMRs and drones.
3. **Industry Partners (Robotics and Drone Manufacturers):**
    - Delivering data that enhances the functionality and performance of their products.
    - Offering a scalable solution adaptable to various robotic and drone platforms.
4. **Regulatory Authorities:**
    - Complying with all relevant regulations and standards.
    - Ensuring the safety and reliability of the mapping system.
5. **The Public:**
    - Contributing to the advancement of autonomous systems and safety in public spaces.
    - Enhancing environmental monitoring and management.

6. **Academic Institutions:**
    - Providing a valuable tool for research and education in robotics and autonomous systems.
    - Supporting innovative projects and studies in spatial sciences and autonomous navigation.
7. **Competitors:**
    - Setting a benchmark for quality, performance, and affordability in the industry.
    - Encouraging healthy competition and continuous improvement.

By addressing these diverse stakeholder requirements, our 3D mapping project is poised to deliver exceptional value and utility, driving progress and innovation in the field of autonomous systems and spatial data collection.

Functional Block Diagram



Process Overview of 3D Mapping with LiDAR

1. Power Supply and Regulation

- **DC Power Input**: The system is powered by a DC supply in the range of 12V to 25V.
- **Voltage Regulation**: The input voltage is regulated to appropriate levels required by various components, ensuring stable operation.

2. LiDAR Data Acquisition

- **Sensor Activation**: The LiDAR sensor is activated and starts emitting laser pulses to measure distances.
- **Data Collection**: The LiDAR sensor collects distance measurements and sends this data to the MCU via a serial connection.

3. Data Processing

- **MCU Handling**: The MCU receives the raw distance data from the LiDAR sensor.
- **Coordinate Calculation**: The processed distance data is converted into coordinates for 3D mapping.

4. Stepper Motor Control

- **Motor Initialization**: The stepper motor is initialized by the MCU.
- **Angle Adjustment**: The MCU controls the stepper motor driver to rotate the motor to precise angles.
- **Scanning**: The stepper motor moves the LiDAR sensor to scan the environment, capturing distance data at various angles.

5. Data Integration and Mapping

- **Data Synchronization**: The distance data from the LiDAR sensor is synchronized with the angle data from the stepper motor.
- **3D Map Construction**: The synchronized data is used to construct a 3D map of the environment.

6. Communication and Command Input

- **USB Interface**: The MCU communicates with a computer via a USB-to-Serial converter.
- **Command Input**: Commands are sent from the computer to the MCU to control the scanning process, adjust settings, and retrieve data.

7. Output and Visualization

- **Data Transmission**: The processed and synchronized data is transmitted back to the computer.
- **Visualization**: The 3D map is visualized on the computer using appropriate software, allowing for analysis and interpretation.

This structured process outlines the flow from power input to data visualization, covering the key steps involved in creating a 3D map using LiDAR technology.

Chapter 2

PCB design

2.1.1 Overview

This document provides a comprehensive overview of the PCB design featuring key components including the A4988 motor driver, ATmega328P microcontroller, 0805 resistors and capacitors, L7805cv voltage regulator, FT232RL USB to Serial converter, male headers, JST connectors, USB ports, and other essential components. The design aims to create a robust and efficient control system for stepper motors, suitable for a variety of applications including 3D printers, CNC machines, and robotic platforms.

2.1.2 Key Components and Selection Rationale

1. **A4988 Motor Driver**
   - **Function:** The A4988 is a micro stepping motor driver with a built-in translator for easy operation. It is capable of driving bipolar stepper motors in full, half, quarter, eighth, and sixteenth step modes.
   - **Comparison and Selection Rationale:**
     - **Versus DRV8825:** The A4988 is generally more cost-effective and sufficient for applications requiring up to 2A per coil. Although the DRV8825 offers higher current capacity (up to 2.5A) and micro stepping options, the A4988's features are adequate for most standard applications.
     - **Versus TB6600:** The TB6600 is more powerful but significantly larger and more expensive, making the A4988 a better choice for compact and cost-sensitive designs.
   - **Features:**
     - Simple step and direction control interface
     - Adjustable current control from 0 to 2 A
     - Five different step resolutions
     - Over-temperature thermal shutdown, under-voltage lockout, and crossover-current protection
2. **ATmega328P Microcontroller**
   - **Function:** The ATmega328P is an 8-bit AVR microcontroller with 32KB of Flash memory, 2KB of SRAM, and 1KB of EEPROM. It is widely used in Arduino boards and provides a versatile platform for embedded applications.
   - **Comparison and Selection Rationale:**
     - **Versus ATmega32U4:** The ATmega328P is chosen for its widespread use, lower cost, and simplicity. While the ATmega32U4 offers USB support natively, using the ATmega328P with an external USB to serial converter (FT232RL) provides more flexibility and modularity.
     - **Versus PIC16F877A:** The ATmega328P has a more extensive community support, making development and troubleshooting easier.
   - **Features:**
     - 23 programmable I/O lines
     - 6-channel 10-bit ADC
     - Serial, SPI, and I2C communication interfaces
     - Low-power consumption with various power-saving modes

3. **FT232RL USB to Serial Converter**
   - **Function:** The FT232RL is a USB to serial UART interface, enabling USB connectivity for data transfer and communication between the microcontroller and a computer.
   - **Comparison and Selection Rationale:**
     - **Versus CP2102:** The FT232RL offers more configuration options and better driver support across various operating systems, making it a more reliable choice.
     - **Versus CH340G:** Although the CH340G is cheaper, the FT232RL is preferred for its superior performance and reliability.
   - **Features:**
     - USB 2.0 full-speed compatibility
     - Integrated clock generation
     - EEPROM for storing configuration settings
     - Provides UART interface with configurable baud rates
4. **0805 Resistors and Capacitors**
   - **Function:** Passive components for filtering, timing, and biasing circuits.
   - **Comparison and Selection Rationale:**
     - **Versus 0603 and 1206 packages:** The 0805 size is chosen as a balance between ease of handling and compact size. 0603 components are smaller but harder to solder manually, while 1206 components are larger and occupy more board space.
   - **Features:**
     - Standard 0805 package size
     - Wide range of values available
     - High reliability and stability
5. **L7805cv Voltage Regulator**
   - **Function:** A voltage regulator to provide a stable output voltage for the circuit, ensuring consistent operation of the microcontroller and other components.
   - **Comparison and Selection Rationale:**
     - **Versus LM7805:** The L7805cv offers a lower dropout voltage and better efficiency in compact designs.
     - **Versus AMS1117:** The L7805cv has better thermal performance and current handling capabilities.
   - **Features:**
     - Low dropout voltage
     - Overcurrent protection
     - Thermal shutdown capability
6. **Connectors and Headers**
   - **Male Headers:** Used for connecting the PCB to external components and modules, facilitating easy integration and prototyping.
   - **JST Connectors:** Compact connectors for secure and reliable connections, commonly used for power supply and sensor inputs.
   - **USB Ports:** Provides connectivity for programming and communication with external devices, such as computers and peripherals.
7. **TFmini-S LIDAR sensor**
   - The TFmini Lidar sensor is a compact, low-cost, and high-performance distance measurement sensor. It uses time-of-flight (ToF) technology to measure distances with high accuracy and reliability. Key features of the TFmini Lidar sensor include:

   - **Range**: Up to 12 meters
   - **Accuracy**: ±1% of the actual distance
   - **Resolution**: 5 mm
   - **Lightweight and compact**: Ideal for applications with space constraints
   - **Low power consumption**: Suitable for battery-powered devices
   - **Interface**: UART or I²C for easy integration with microcontrollers and other devices

2.1.3 Design consideration

- **Power Supply:** The PCB design includes considerations for efficient power distribution, ensuring that all components receive the necessary voltage and current without excessive heat generation or voltage drops.
- **Signal Integrity:** Careful layout of traces and placement of components to minimize noise and signal interference, crucial for the accurate operation of the microcontroller and motor driver.
- **Thermal Management:** Proper spacing and layout to allow heat dissipation from components like the A4988 motor driver and voltage regulator, preventing overheating and ensuring longevity of the components.
- **Footprint and Sizing:** Using standardized component sizes (e.g., 0805 for resistors and capacitors) to maintain a compact and manufacturable design, while ensuring ease of assembly and soldering.

2.1.4 Schematic design

The schematic design utilizes standard electronic symbols and conventions to ensure clarity and readability. Each component is represented with its conventional symbol, pin numbers, and labels. Key sections of the schematic include:

1. **Power Supply Section**
    - **L7805CV Voltage Regulator:** Provides a stable 5V supply to the circuit. The input is connected to an external power source, with decoupling capacitors (0805) to filter noise.
2. **Microcontroller Section**
    - **ATmega328P Microcontroller:** Acts as the central processing unit, interfacing with the LIDAR sensor and other peripherals. Connections include power, ground, crystal oscillator for clock, and I/O pins for communication.
3. **Motor Control Section**
    - **A4988 Motor Driver:** Controls the stepper motor for rotating the LIDAR sensor. The driver is connected to the microcontroller for step and direction control, with current-setting resistors (0805) and decoupling capacitors (0805) for stability.
4. **USB Communication Section**
    - **FT232RL USB to Serial Converter:** Facilitates USB communication between the microcontroller and an external computer. The converter is connected to the microcontroller's UART pins, with necessary supporting components like resistors and capacitors (0805).
5. **Connector Section**
    - **Male Headers and JST Connectors:** Used for interfacing with external components such as sensors, motors, and power supplies. Each connector is labeled with its function and pin numbers.
    - **USB Ports:** Provide connectivity for power and data transfer, connected to the FT232RL for serial communication.

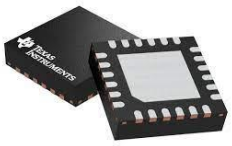


Fig: A4988 Motor Driver

Fig: TFmini-s

Fig: ATmega328p-au

Fig: FT232RL(USB to Serial)

Fig: L7805CV regulator

Fig: Micro USB (SMD)

Fig: JST Connectors

2.1.5 Reading the Schematic

- **Symbols and Labels:** Each component is represented by its standard symbol, with clear labels for pin numbers and functions.
- **Net Labels:** Wires are labeled with net names to indicate their connection points across different parts of the schematic.
- **Reference Designators:** Each component is assigned a reference designator (e.g., R1 for resistors, C1 for capacitors, U1 for ICs) to easily identify them in the Bill of Materials (BOM) and layout.
- **Decoupling Capacitors:** Placed near power pins of ICs to filter noise and ensure stable operation.
- **Bypass Capacitors:** Used on power lines to further filter and stabilize the power supply.

The schematic design for the 3D mapping with LIDAR project integrates critical components using standard conventions and symbols to ensure clarity and ease of understanding. The use of Altium Designer allows for precise layout and documentation, supporting effective implementation and troubleshooting of the system. This document serves as a clear guide for assembling and understanding the electronic design of the project.

2.1.6 Schematics

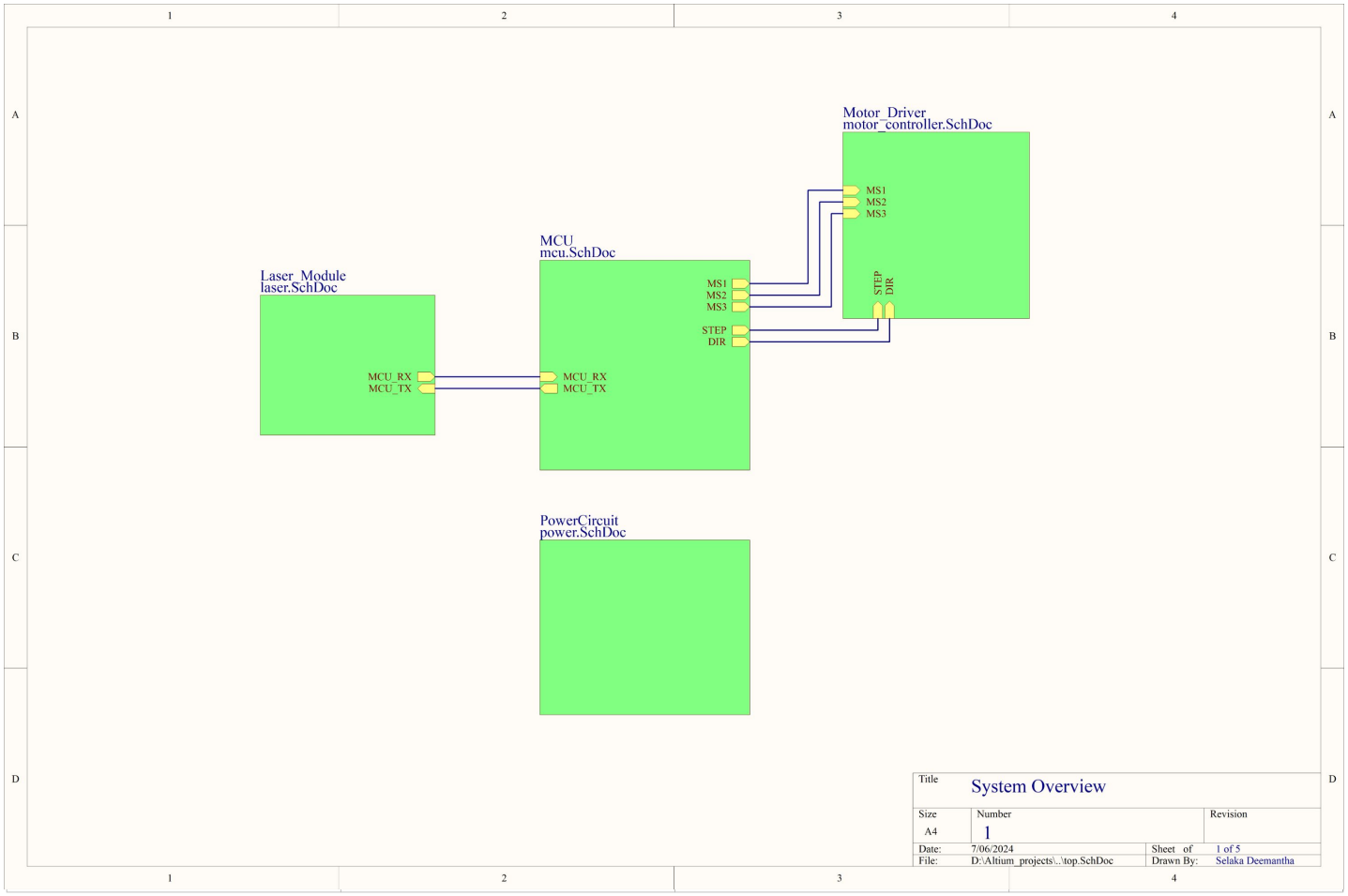


Fig: Hierarchy

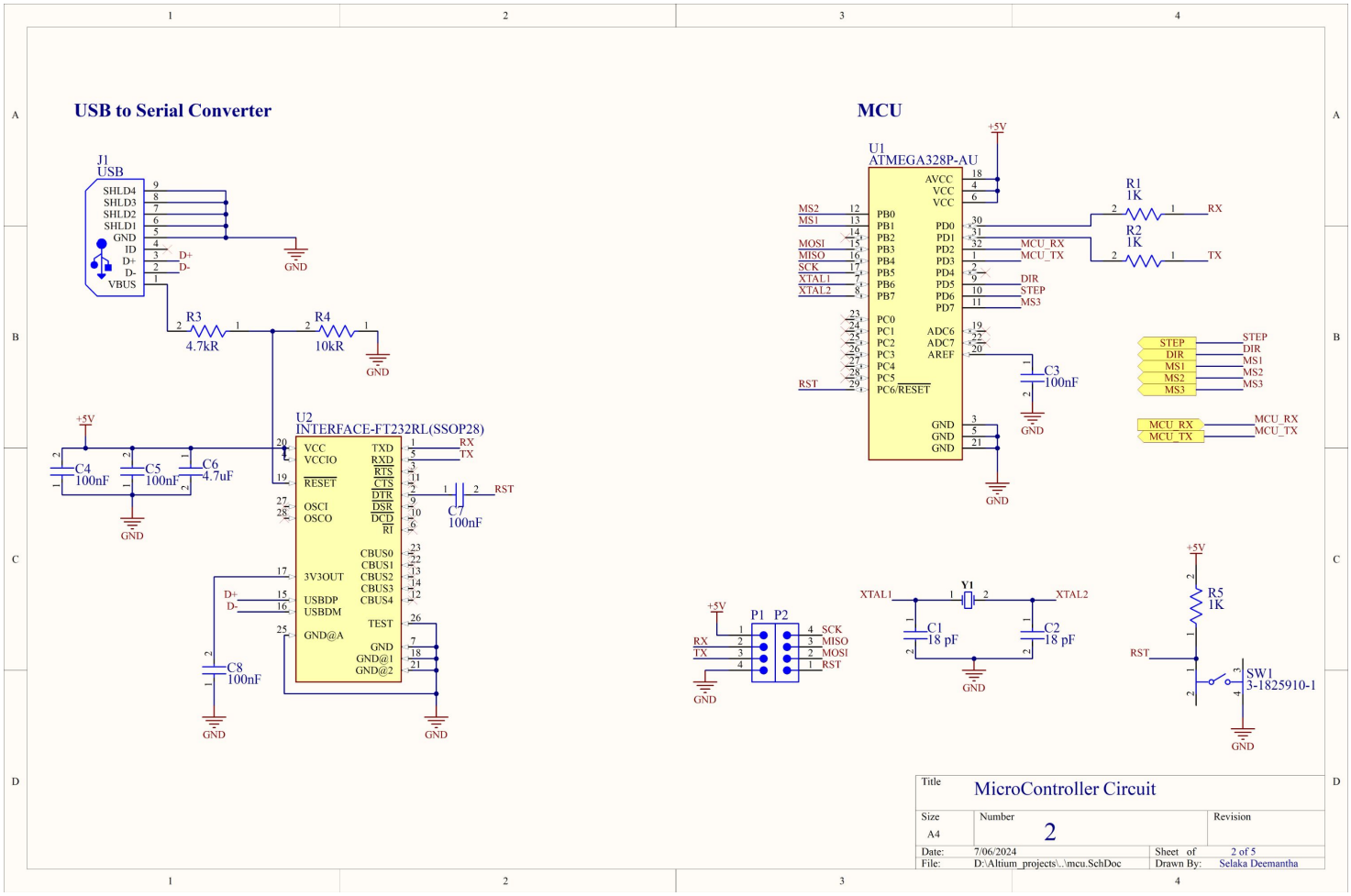


Fig: MCU

The microcontroller sheet includes the defined ports that connect to both the Lidar sensor and the stepper motor driver sheets. This hierarchical organization ensures clear and manageable schematics, enhancing the design process and debugging efficiency.

2.1.7 TFmini LIDAR sensor integration

The TFmini Lidar sensor is connected to the microcontroller through the following ports:

- **MCU_TX** (Microcontroller) to the **UART_RX** (TFmini Lidar)
- **MCU_RX** (Microcontroller) to the **UART_TX** (TFmini Lidar)

A4988 Motor Controller integration

The A4988 motor controller is connected to the microcontroller through the following ports:

- **STEP**(MCU) to the **STEP**(Motor_Driver) : for controlling the steps of the motor.
- **DIR**(MCU) to the **DIR**(motor_Driver) : for controlling the direction of the motor.
- **MS1, MS2, MS3** respectively to the **ms1, ms2, ms3**(motor_Driver) : for controlling the micro_stepping functionality of the motor Driver .

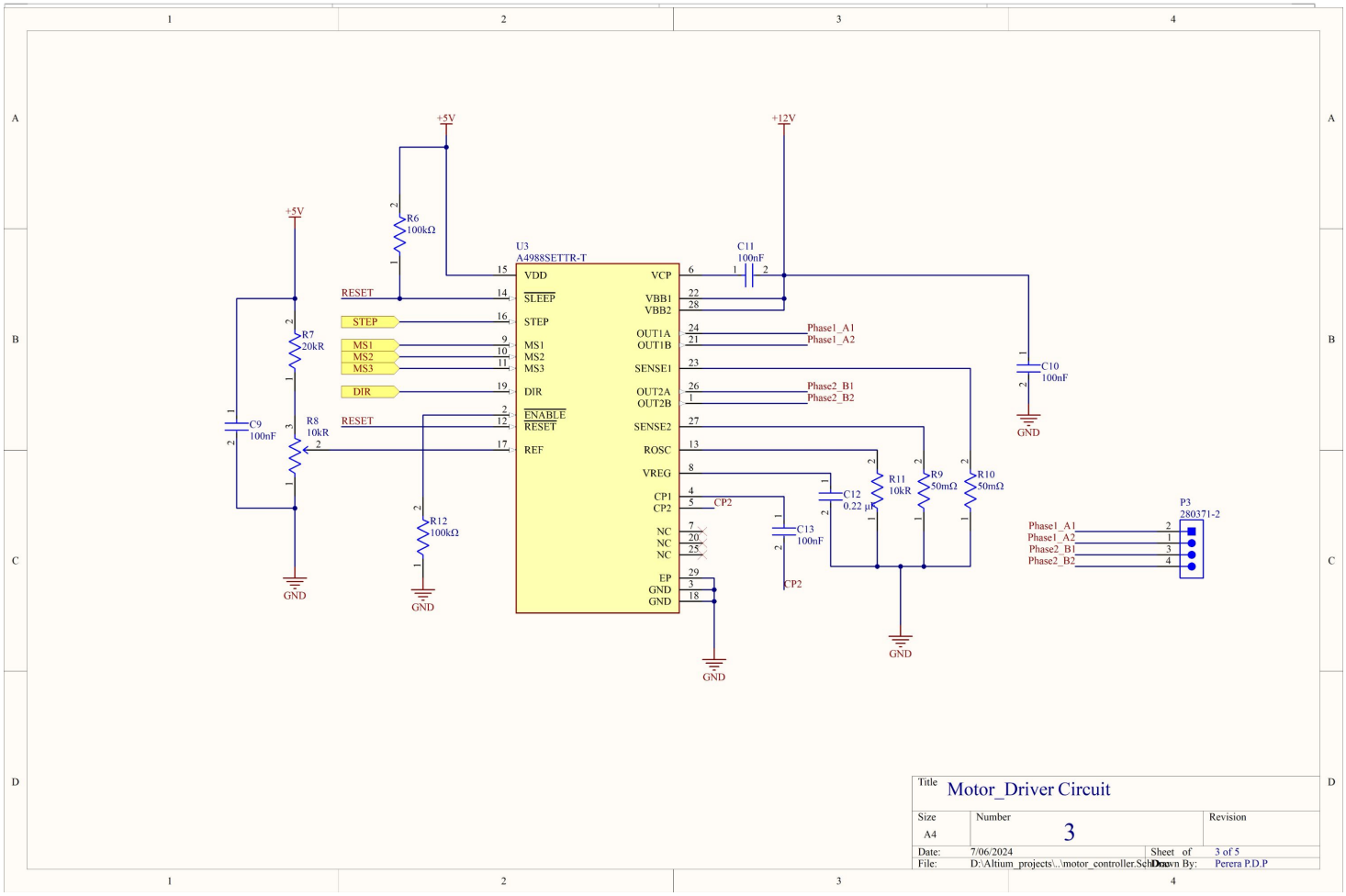


Fig: Motor Driver Circuit

- The four phase wires in the motor driver circuit sheet (A1, A2, B1, B2) provide the necessary current to the stepper motor's coils, enabling precise control of the motor's rotation and step movements.
- The Variable resistor (R8) allows for the controlling of the current flown to the Motor with a maximum of (2A).

- Below is the Power Regulator circuit, which provides a stable 5V output to power the entire system. This includes the TFmini Lidar sensor connection lines, where the regulated 5V is supplied to the VCC pin of the Lidar sensor, ensuring consistent and reliable operation. We also added a decoupling capacitor to the power lines to filter out any noise and stabilize the voltage supply. Additionally, the connection lines for UART communication (MCU_TX and MCU_RX) link the microcontroller to the Lidar sensor, allowing for precise distance measurement data exchange.

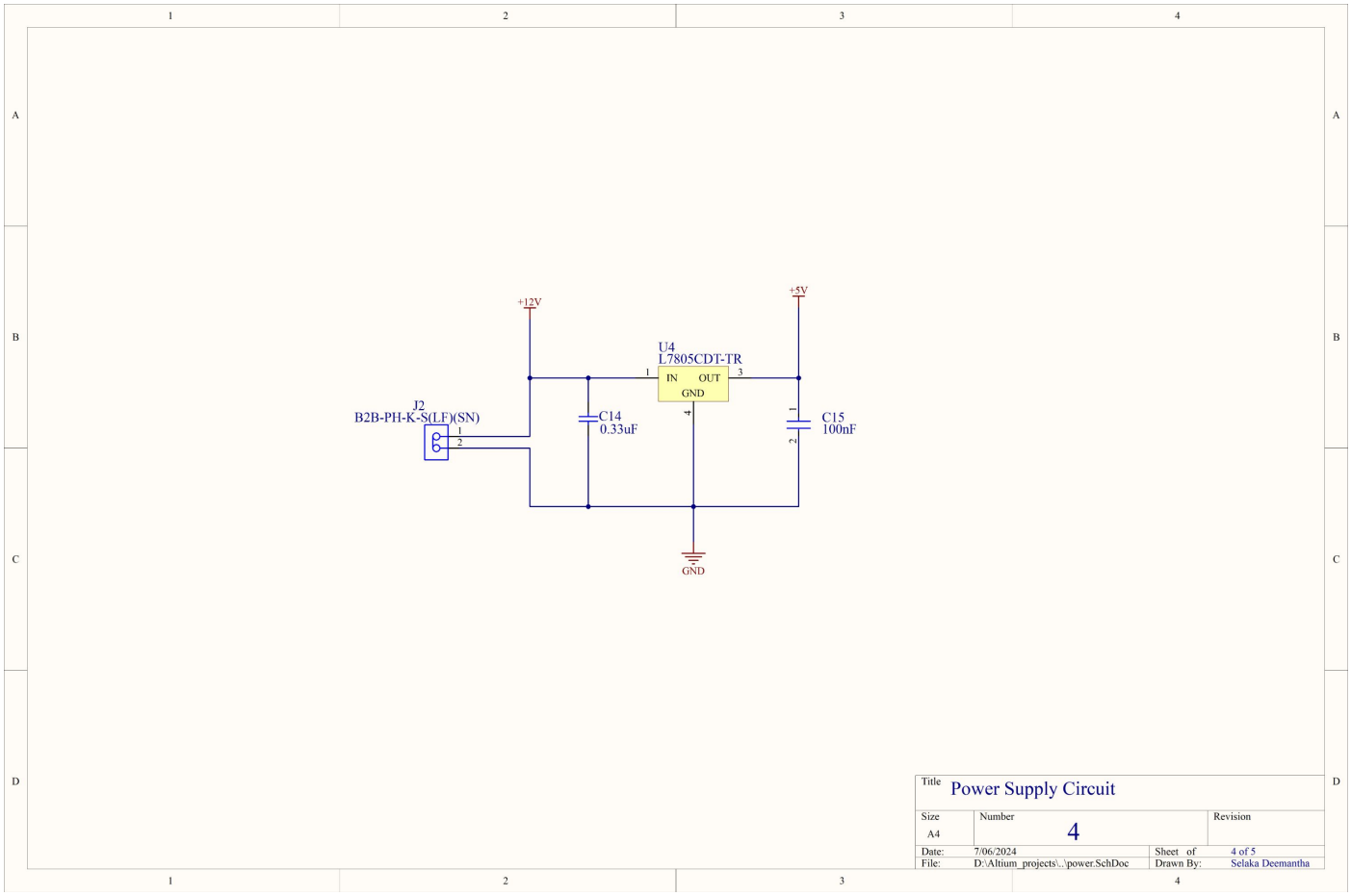


Fig: Power Supply Circuit

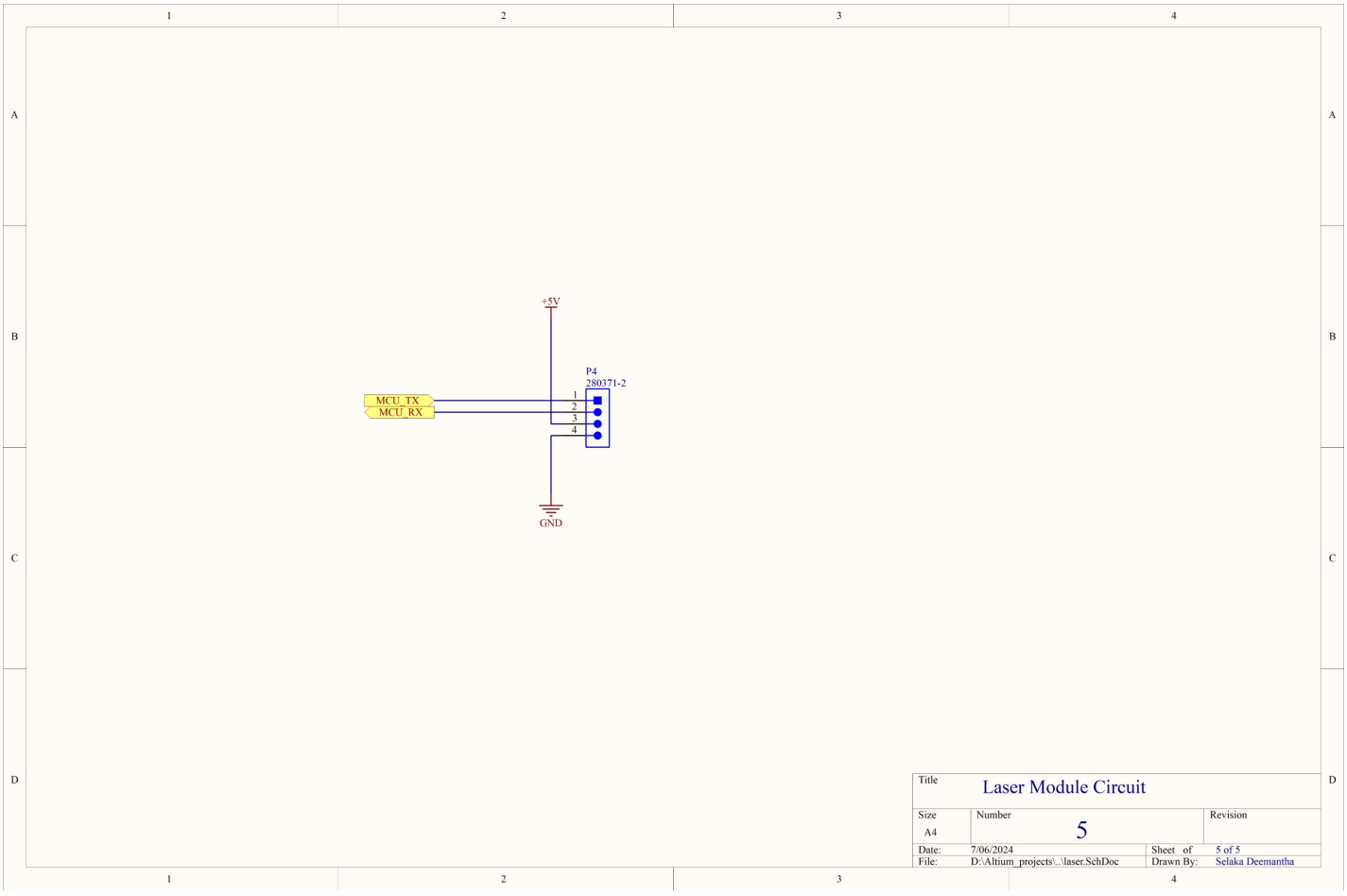


Fig: Laser module port

2.1.8 Component lists in each sheet with description and quantity

Description	Designator	Quantity
CAP CER 18PF 200V 10% NP0 0805	C1, C2	2
General Purpose Ceramic Capacitor, 0805, 100nF, 10%, X7R, 15%, 100V	C3, C4, C5, C7, C8	5
General Purpose Ceramic Capacitor, 0805, 4.7uF, 10%, X7R, 15%, 16V	C6	1
USB Connector, 30 V, 1 A, 0 to 50 degC, 5-Pin SMD, RoHS, Tape and Reel	J1	1
	P1, P2	2
	R1, R2, R5	3
RES Thick Film, 4.7kΩ, 1%, 0.125W, 100ppm/°C, 0805	R3	1
	R4	1
SWITCH TACTILE SPST-NO 0.05A 24V	SW1	1
AVR AVRÂ® ATmega Microcontroller IC 8-Bit 20MHz 32KB (16K x 16) FLASH 32-TQFP (7x7)	U1	1
USB-Ð¼Ð¾ÑÑ,, Ð¸Ð½Ñ‚ÐµÑ€Ñ„ÐµÐ¹Ñ USB Ðº UART USB 2.0 UART 28-SSOP	U2	1
Resistance Weld Thru-Hole Crystal, 16 MHz, -20 to 70 degC, 2-Pin THD, RoHS, Bulk	Y1	1

Fig: Information related to MCU sheet

Description	Designator	Quantity
General Purpose Ceramic Capacitor, 0805, 100nF, 10%, X7R, 0.15, 50V	C9, C10, C11, C13	4
CAP CER 0.22UF 50V X7R 0805	C12	1
Male Header, Pitch 2.54 mm, 1 x 4 Position, Height 12.8 mm, Tail Length 3.5 mm, -55 to 105 degC, RoHS, Bulk	P3	1
RES 100K OHM 1% 1/8W 0805	R6, R12	2
RES Thick Film, 20kΩ, 5%, 0.125W, 200ppm/°C, 0805	R7	1
TRIMMER 10K OHM 0.5W GW TOP ADJ	R8	1
RES 0.05 OHM 1% 1W 2512	R9, R10	2
RES Thick Film, 10kΩ, 1%, 0.125W, 100ppm/°C, 0805	R11	1
IC MTR DRVR BIPOLAR 3-5.5V 28QFN	U3	1

Fig: Information related to Motor Driver sheet

Description	Designator	Quantity
	C14	1
CAP CER 0.1UF 50V Z5U 0805	C15	1
Male Header, Pitch 2 mm, 1 x 2 Position, Height 6 mm, Tail Length 3.4 mm, -25 to 85 degC, RoHS, Bulk	J2	1
Positive Voltage Regulator, 3-Pin DPAK, Tape and Reel	U4	1

Fig: Information related to Power Circuit sheet

2.1.9 PCB Routing

PCB Routing Description

The PCB layout consists of two layers: the top layer and the bottom layer. We implemented careful routing strategies to ensure optimal performance and reliability of the circuit.

- **Top Layer and Bottom Layer**: The top layer primarily houses the signal traces and critical components, while the bottom layer is used for routing additional signal traces and power connections. This separation helps in managing signal integrity and reducing interference.
- **Wider Trace Widths for High-Current Paths**: For high-current carrying paths, such as the power supply lines and motor driver connections, we used wider trace widths. This ensures that the traces can handle the required current without excessive heating or voltage drops, enhancing the overall efficiency and safety of the circuit.
- **Minimized Distance for High Signal Integrity**: We minimized the distance between the microcontroller and the TFmini Lidar sensor for the UART communication lines (UART_TX and UART_RX). This reduces potential signal degradation and ensures reliable data transfer. Similar care was taken for other critical connections, including the stepper motor driver control lines (STEP, DIR, and EN) to maintain signal integrity and precise control.
- **Decoupling Capacitors**: Decoupling capacitors were strategically placed close to the power pins of critical components, such as the microcontroller and the Lidar sensor, to filter out noise and stabilize the voltage supply.

Fig: Top Layer

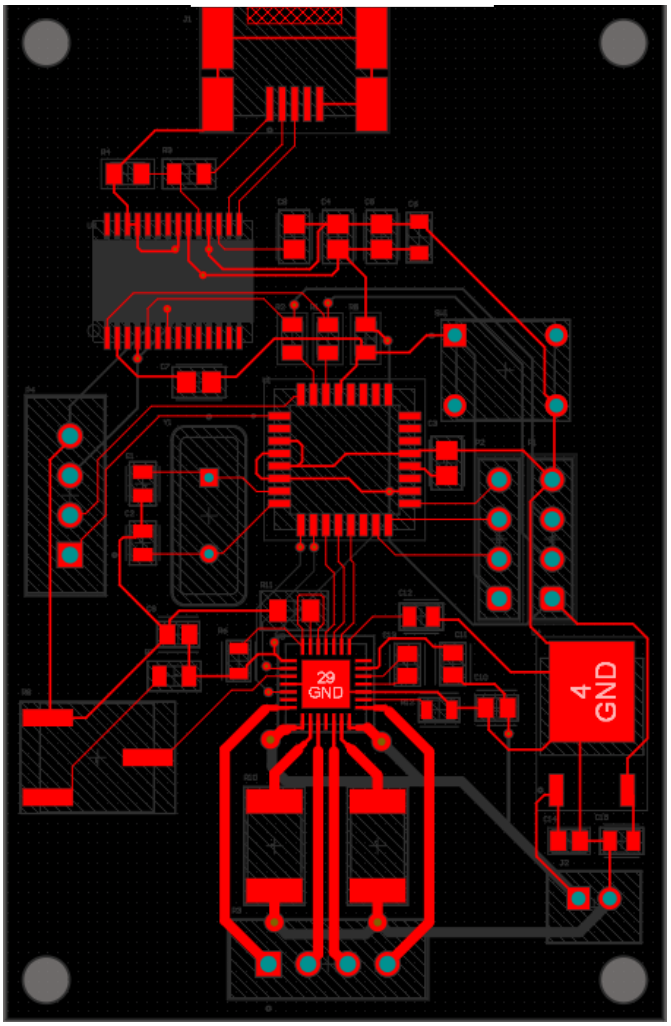
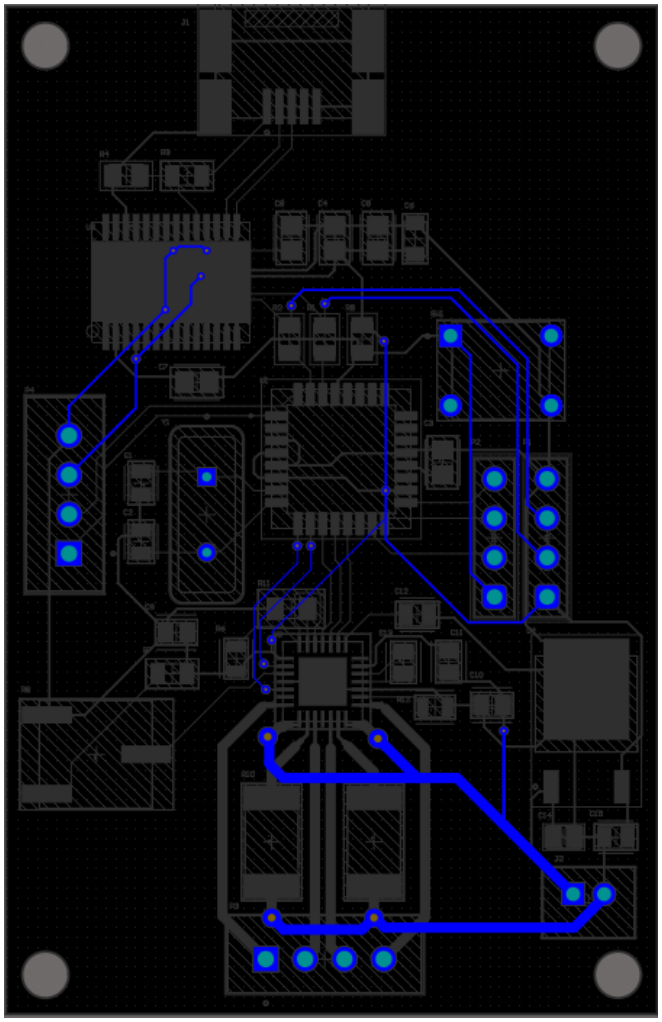
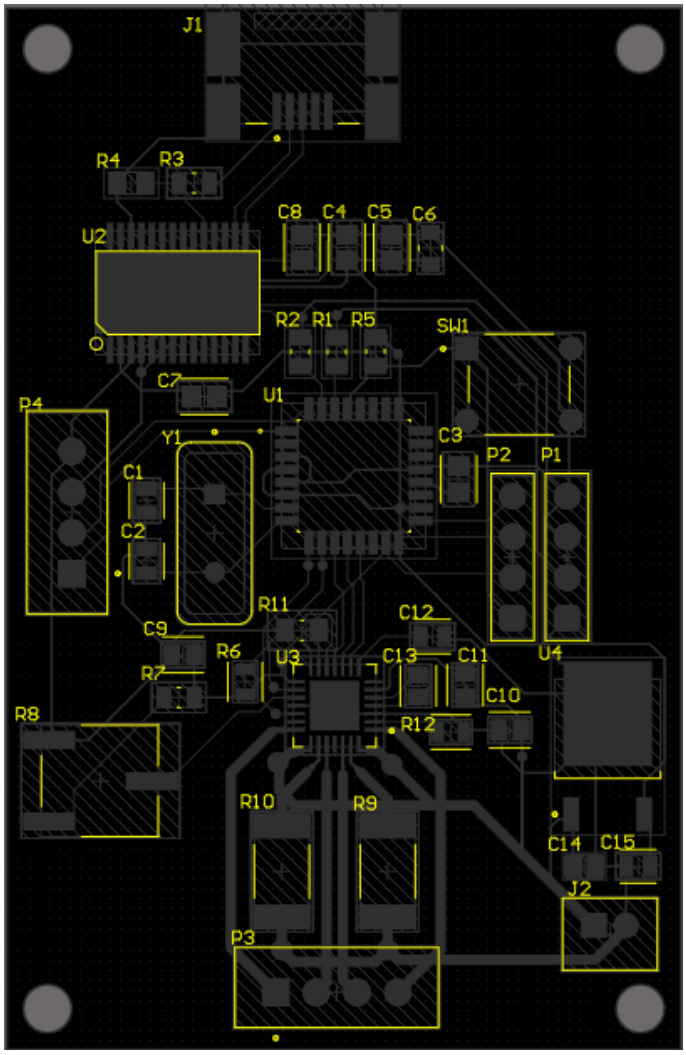
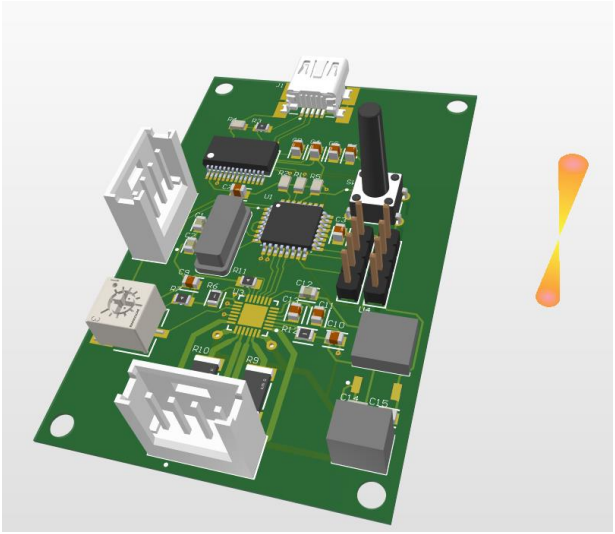


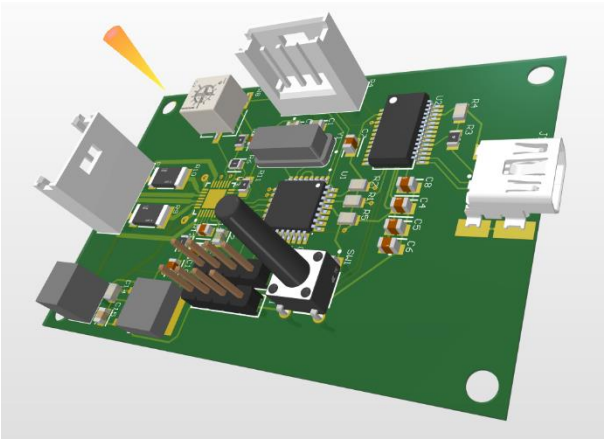
Fig: Bottom Layer



2.1.10 PCB Top overlay and the PCB layout







The PCB top overlay and layout in our project are crucial components designed to ensure clarity, functionality, and reliability. The top overlay provides a visual guide for component placement and labeling, essential for assembly and maintenance. It includes detailed markings such as component designators, outlines, and critical information for soldering and troubleshooting.

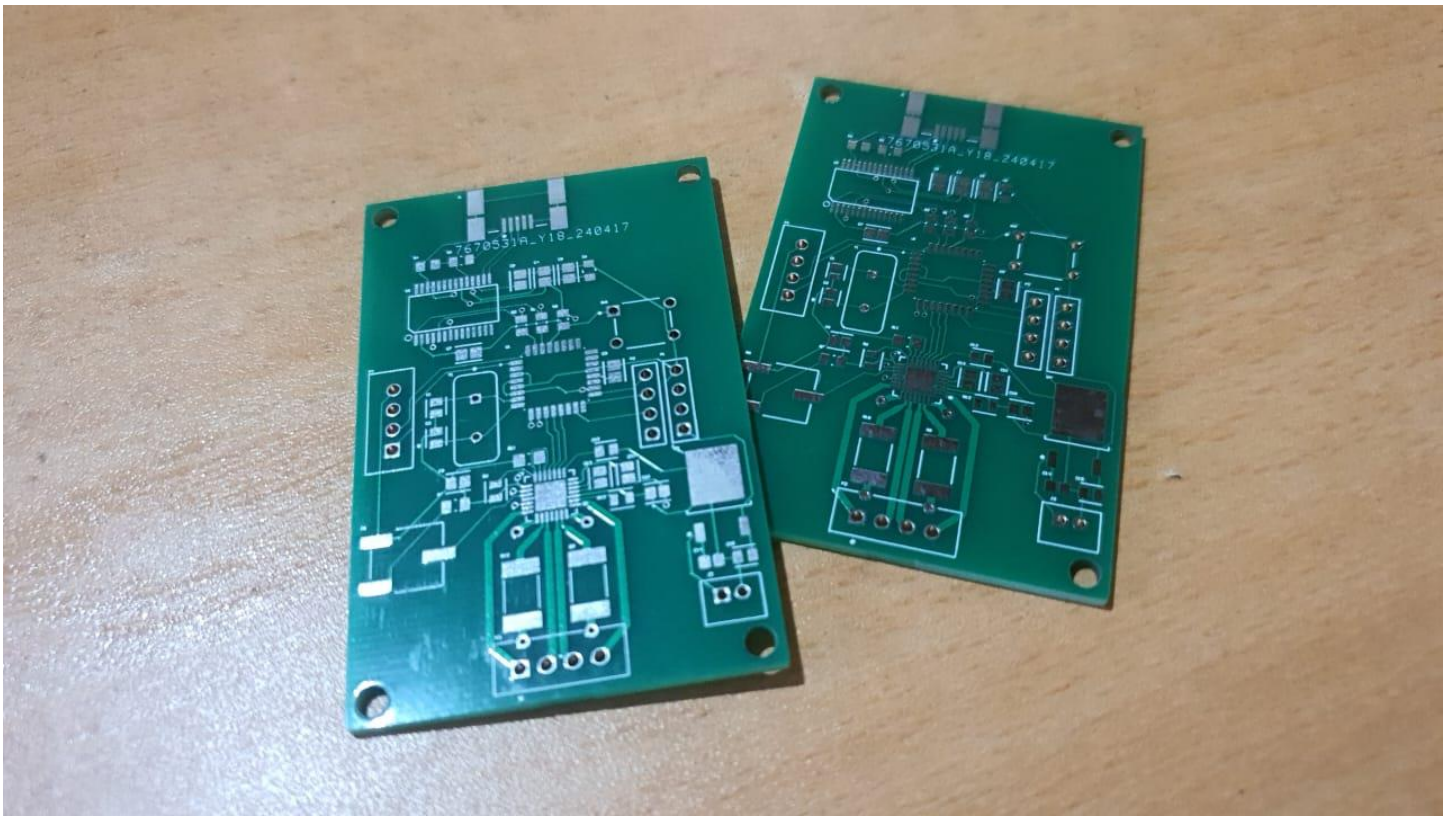
In our layout design, each component's placement is strategically optimized to minimize signal interference, enhance thermal management, and streamline the overall circuit flow. Careful attention is given to trace routing, ensuring signal integrity. This meticulous approach in PCB layout not only meets functional requirements but also facilitates efficient manufacturing and assembly processes.

2.1.11 PCB Layer stack

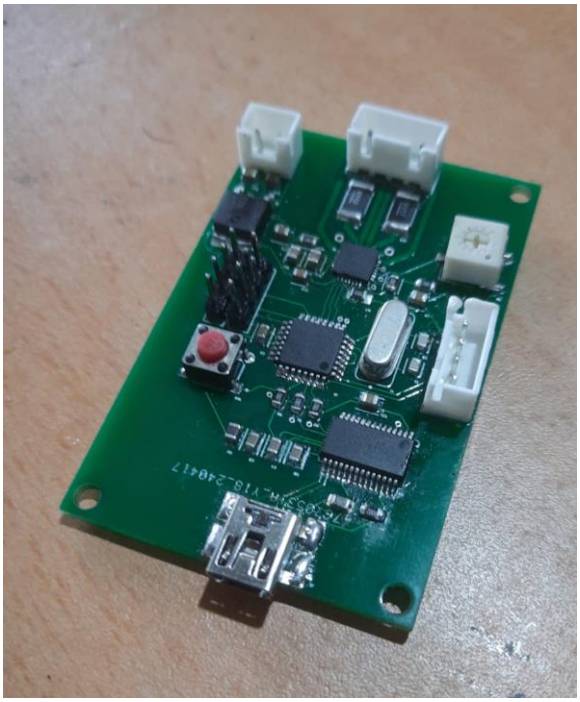
#	Name	Material	Type	Thickness	Weight	Dk
	Top Overlay		Overlay			
	Top Solder	Solder Resist	Solder Mask	0.4mil		3.5
1	Top Layer		Signal	1.4mil	1oz	
	Dielectric 1	FR-4	Dielectric	12.6mil		4.8
2	Bottom Layer		Signal	1.4mil	1oz	
	Bottom Solder	Solder Resist	Solder Mask	0.4mil		3.5
	Bottom Overlay		Overlay			

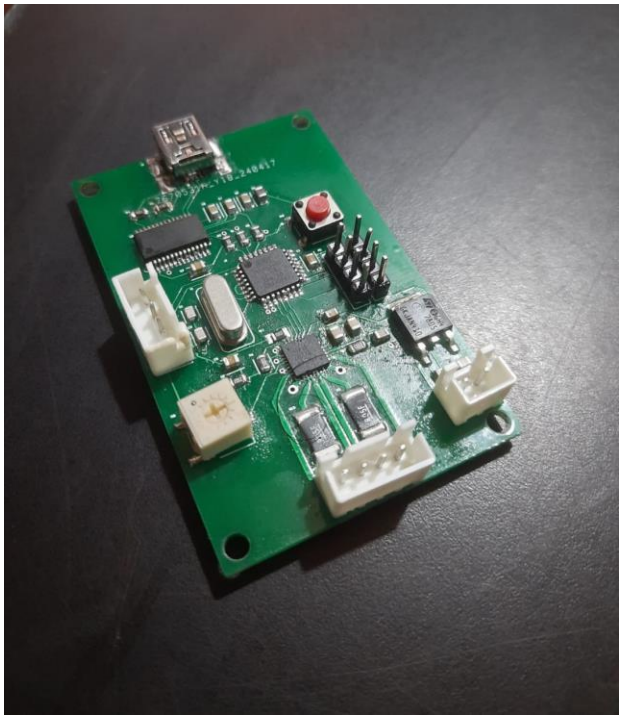
2.1.12 Bare PCB

We manufacture our PCBs using JLCPCB, with dimensions of 65mm x 42mm.



2.1.13 Soldered PCB





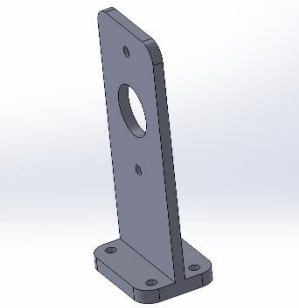
Chapter 3

SolidWorks Design

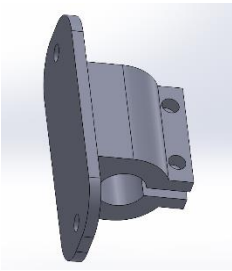
3.1.1 Introduction

Our project for 3D mapping with Lidar is designed in SolidWorks, featuring 10 functional parts alongside mechanical components. The overall design, selected from three concepts, is cylindrical with dimensions of 158mm x 170mm.

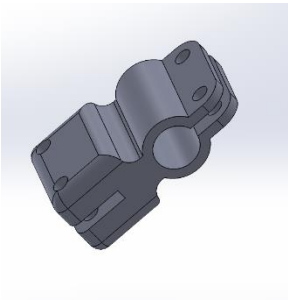
3.1.2 Main Functional parts



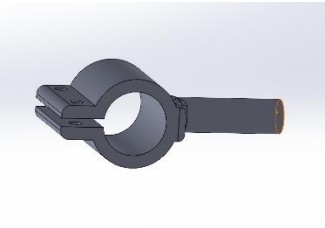
Lidar sensor mount



Lidar sensor Holder

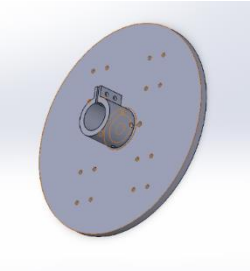


Shaft Holder 1

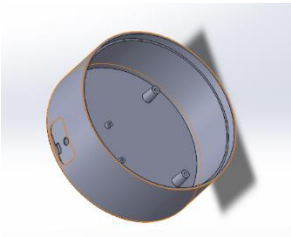


Shaft Holder 2

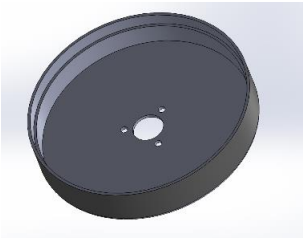
Lead Screw holder



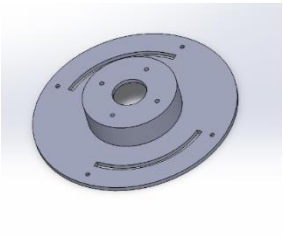
Rotating mount



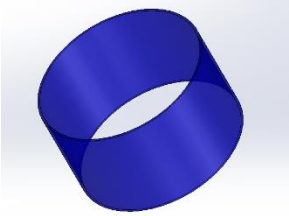
Base(main)



Top(main)



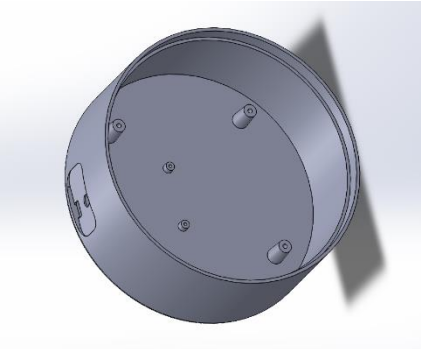
Stepper motor mount(main)

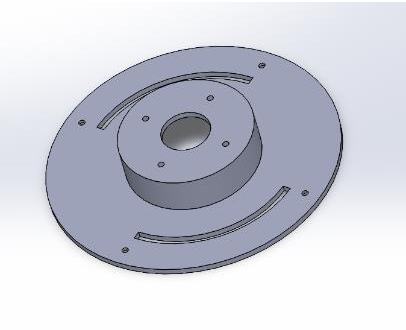


Outer covering

- **Lidar Sensor mount :** To hold the bearings and the shaft holding the LIDAR sensor .
- **Lidar Sensor Holder:** To hold the LIDAR sensor.
- **Shaft Holder 1, Shaft Holder 2 :** To hold the linear bearings and the shafts for motions.
- **Lead Screw Holder:** To Guide the path of the vertical motion along the lead screw.
- **Rotating mount:** most of the sub-assemblies are mounted on this, and this is the rotating part.
- **Base :** To hold the PCB and the Stepper motor mount .
- **Top:** Covering and Holding the Screw shaft stationary .
- **Stepper motor Mount :** To Hold the stepper motor .

3.1.3 Model Trees with main subparts

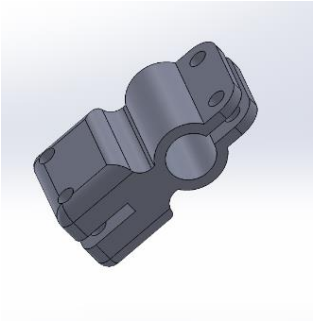


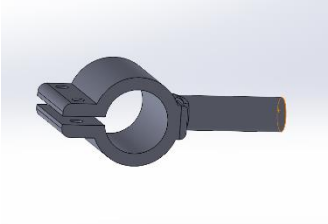


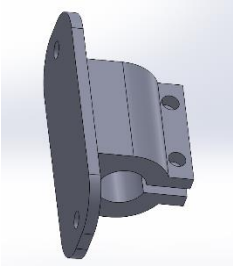
- Sensors
- Annotations
- Solid Bodies(1)
- Material <not specified>
- Front Plane
- Top Plane
- Right Plane
- Origin
- Boss-Extrude1
  - Sketch1
- Boss-Extrude2
  - Sketch2
- Boss-Extrude3
  - Sketch4
- Cut-Extrude2
  - Sketch6
- Cut-Extrude3
  - Sketch7
- Fillet1
- Boss-Extrude4
  - Sketch8
- Cut-Extrude4
  - Sketch9
- Plane1
- Cut-Extrude5
  - Sketch10
- Cut-Extrude7
  - Sketch14
- Cut-Extrude8
  - Sketch15

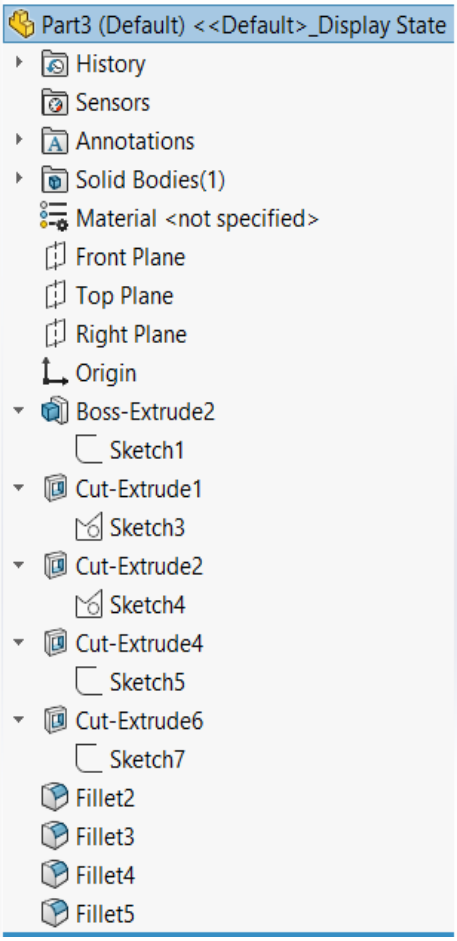
- Part15 (Default) <<Default>_Display
- History
- Sensors
- Annotations
- Solid Bodies(1)
- Material <not specified>
- Front Plane
- Top Plane
- Right Plane
- Origin
- Boss-Extrude1
  - Sketch1
- Cut-Extrude1
  - Sketch2
- Cut-Extrude2
  - Sketch3
- Boss-Extrude2
  - Sketch4
- Cut-Extrude3
  - Sketch5
- Fillet1

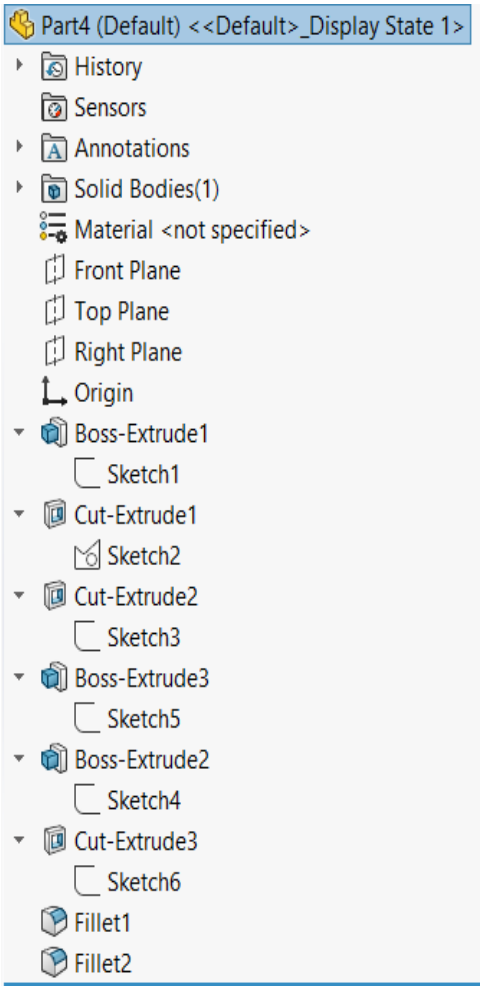
- Part18 (Default) <<Default>_Display
- History
- Sensors
- Annotations
- Solid Bodies(1)
- Material <not specified>
- Front Plane
- Top Plane
- Right Plane
- Origin
- Boss-Extrude1
  - Sketch1
- Boss-Extrude2
  - Sketch2
- Cut-Extrude1
  - Sketch3
- Plane2
- Cut-Extrude3
  - Sketch6
- Cut-Extrude5
  - Sketch7
- Cut-Extrude6
  - Sketch8

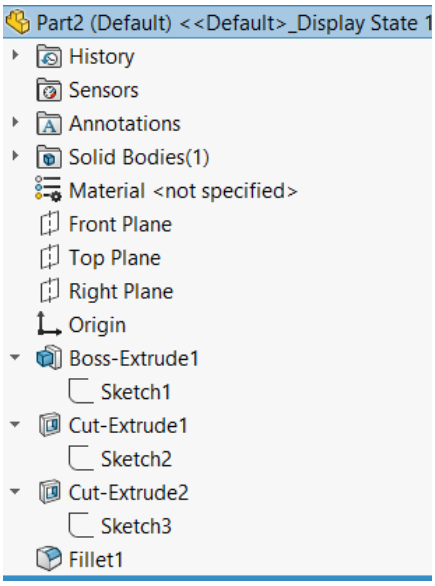








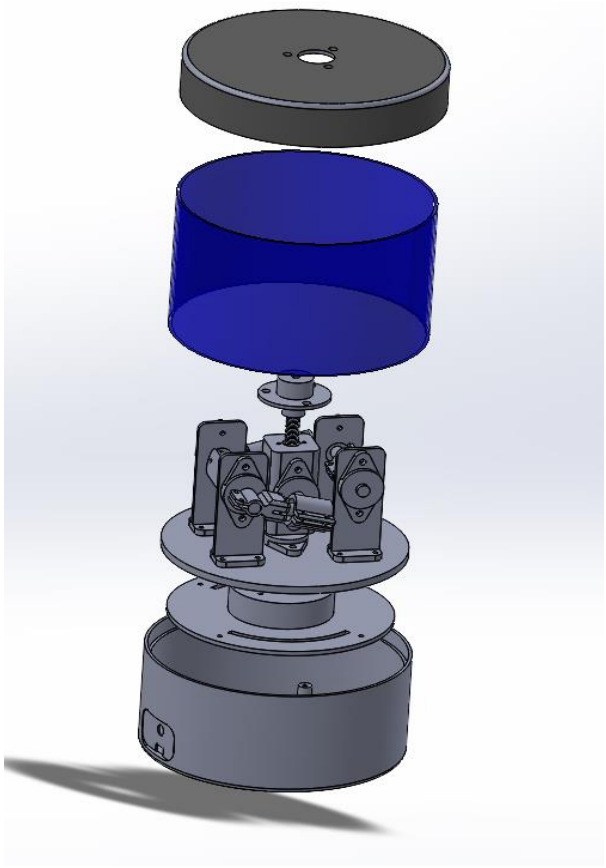


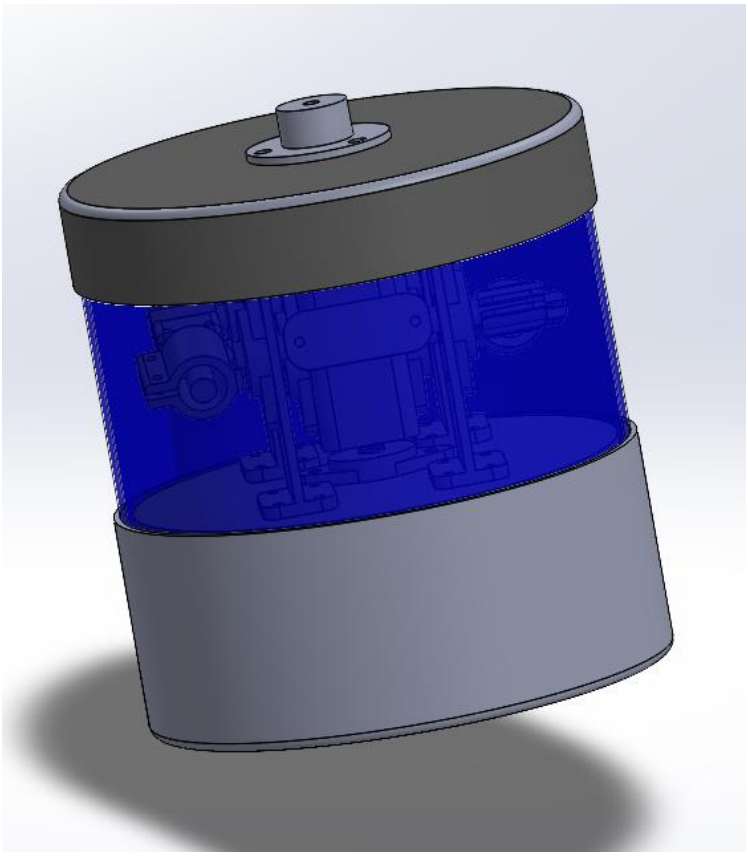


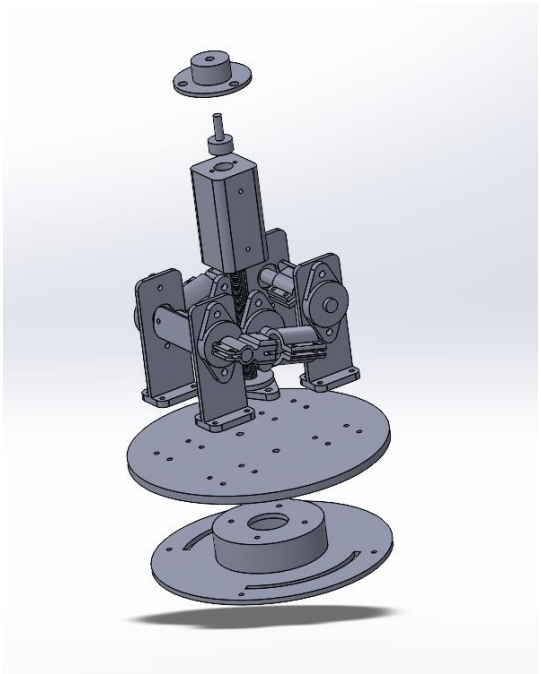
We've used SolidWorks to design precise and detailed components crucial for our LIDAR-based 3D mapping project. Each part is carefully engineered to ensure it fits perfectly and functions well within our mapping system. SolidWorks has allowed us to create everything from protective housings for electronics to mounts that ensure our sensors are perfectly aligned. These components, thanks to SolidWorks, turn our design ideas into reliable parts that are essential for the success of our 3D mapping efforts.

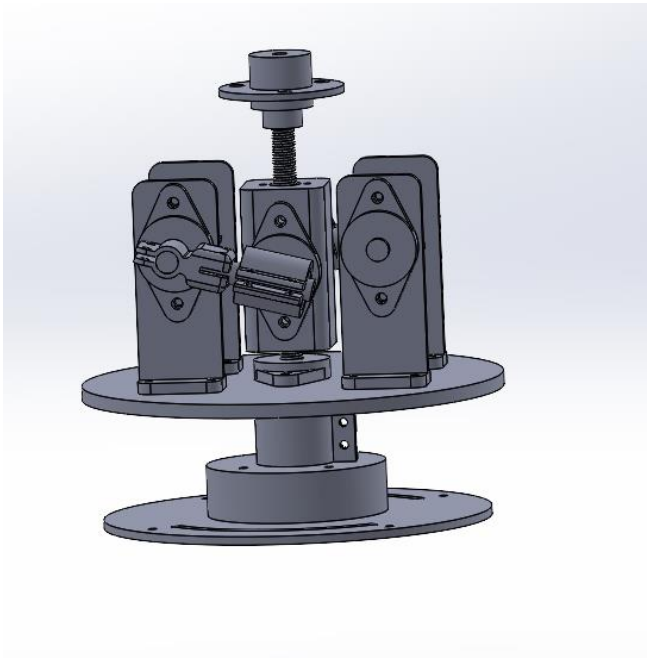
3.1.4 Overall Integration of the parts

We assembled all the designed parts in SolidWorks to visualize our final device for our 3D mapping project. This process allowed us to integrate each component, ensuring they fit together seamlessly and function as intended. Visualizing our device in SolidWorks provided a clear understanding of how each part interacts within the overall structure, from the sensor mounts to the protective housings. This digital assembly not only verified the compatibility and functionality of our design but also enabled us to make precise adjustments before manufacturing. It was a crucial step in refining our concept into a cohesive and effective solution for our 3D mapping endeavors.









3.1.5 Mold Designs for outer parts

The mold design for our 3D mapping LiDAR enclosure was created in SolidWorks with a focus on durability and precision.

Mold Design:

- Designed to securely house the TFmini-S LiDAR sensor.
- Ensured an unobstructed field of view for accurate 3D mapping.
- Included mounting holes, cable channels, and ventilation slots.
- Created a two-part mold with core and cavity for easy part ejection.
- Added draft angles to facilitate mold release.

Key Features and Benefits

- **Robust Protection:** Protects against environmental factors.
- **Ease of Manufacturing:** Facilitates efficient production.
- **Functional Design:** Optimizes performance and user convenience.

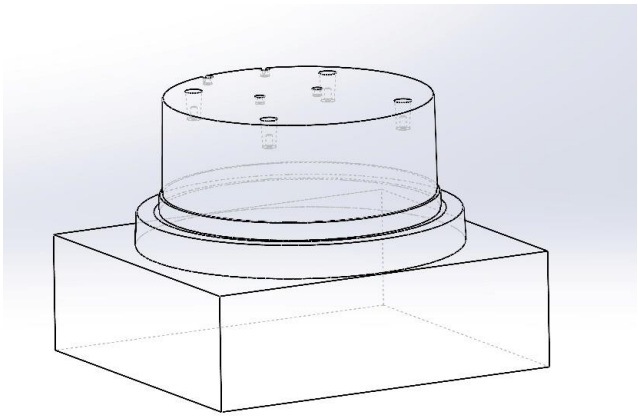


Fig : Bottom Core wire frame

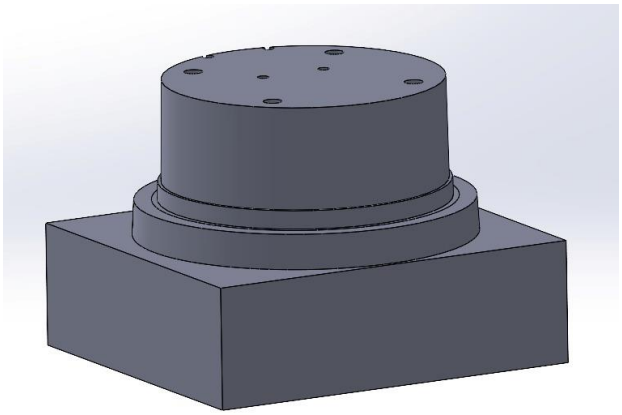


Fig : Bottom Core

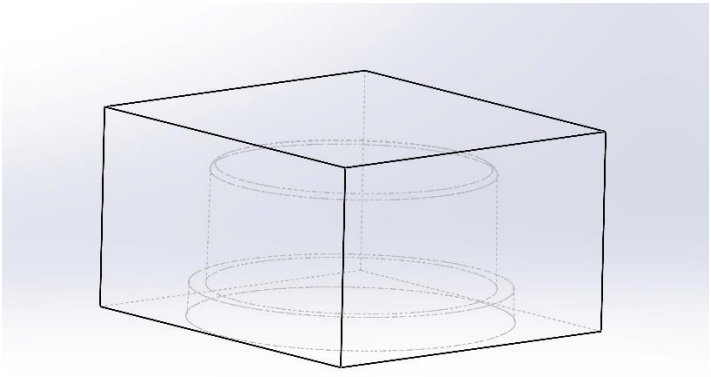


Fig : Bottom Cavity wire frame

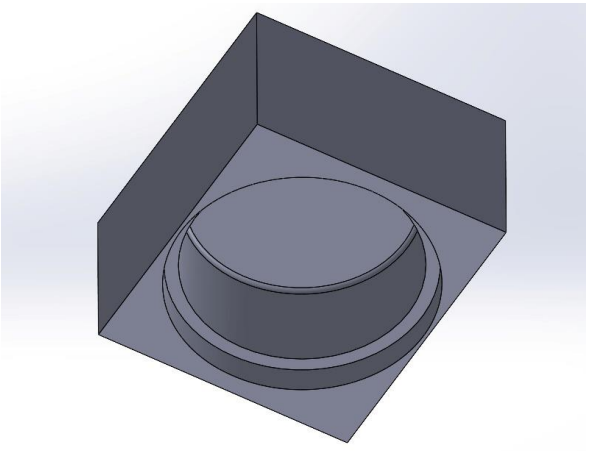


Fig : Bottom Cavity

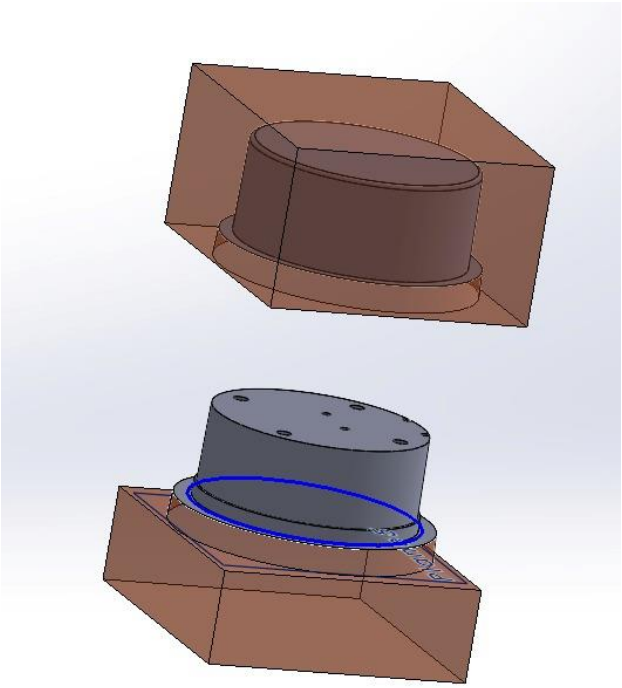


Fig : Bottom part mold

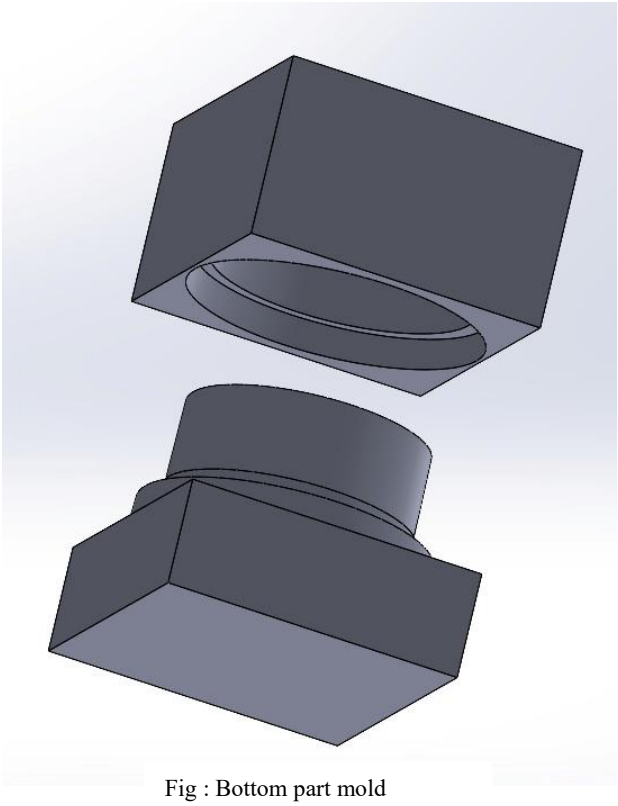


Fig : Bottom part mold

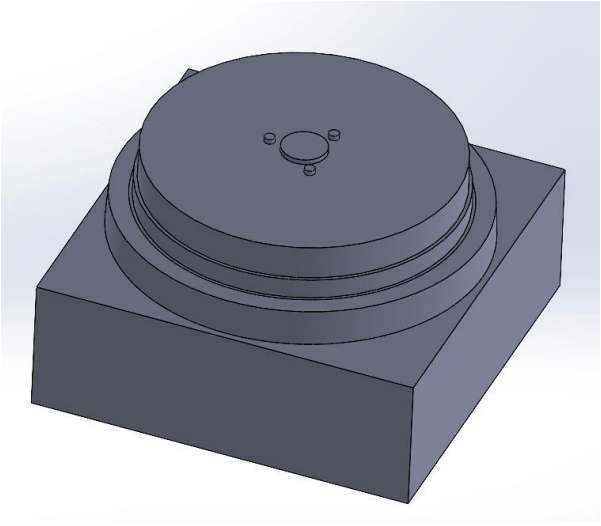


Fig : Top part core

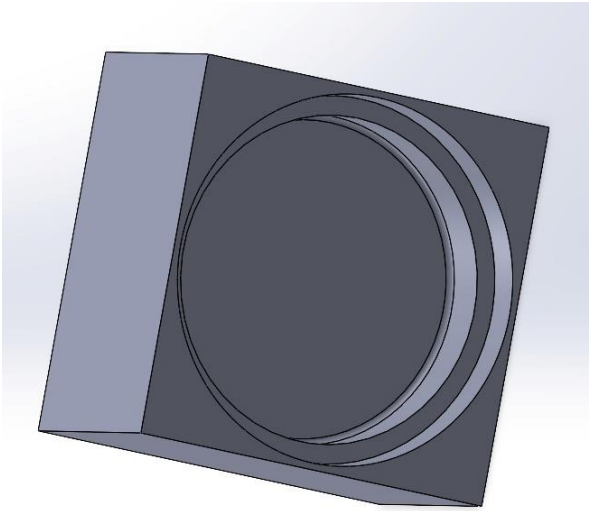


Fig : Top part core

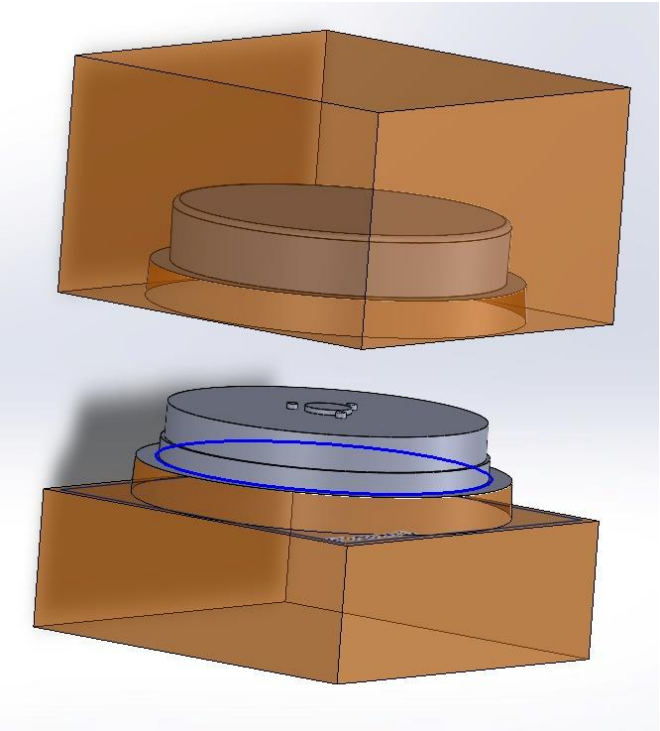


Fig : Top part mold

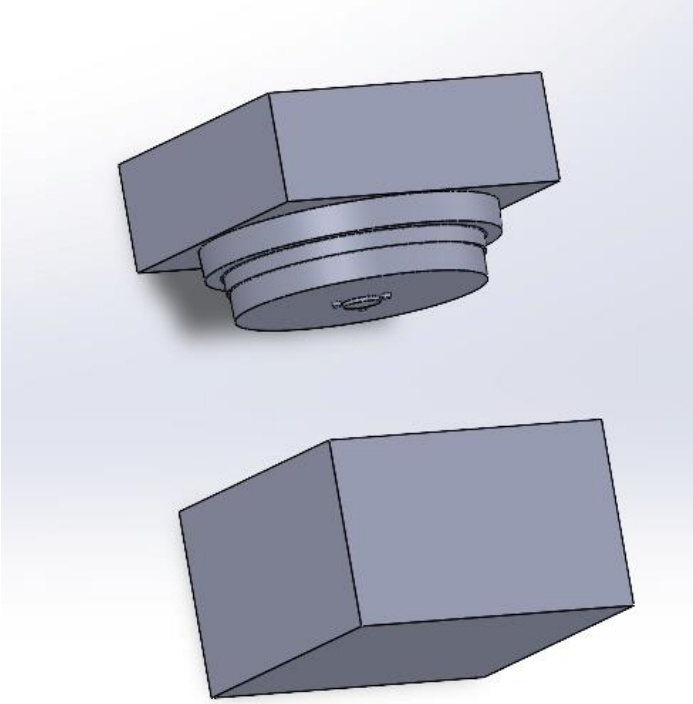


Fig : Top part mold

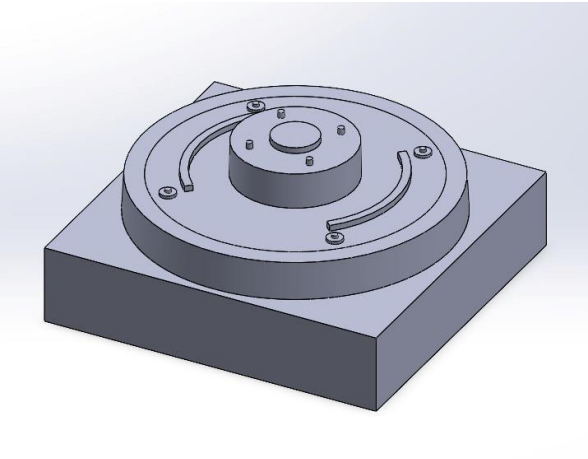


Fig : Stepper motor mount core

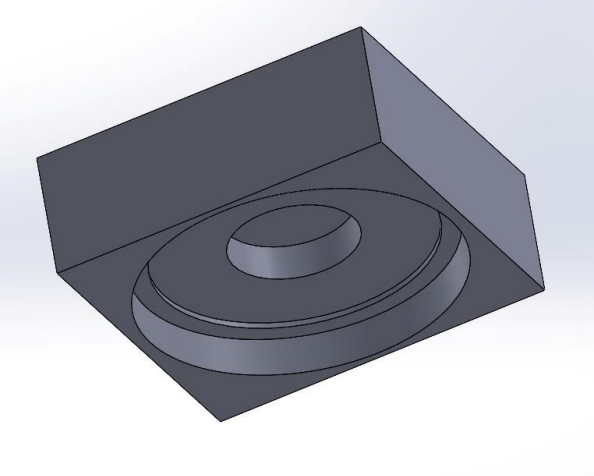
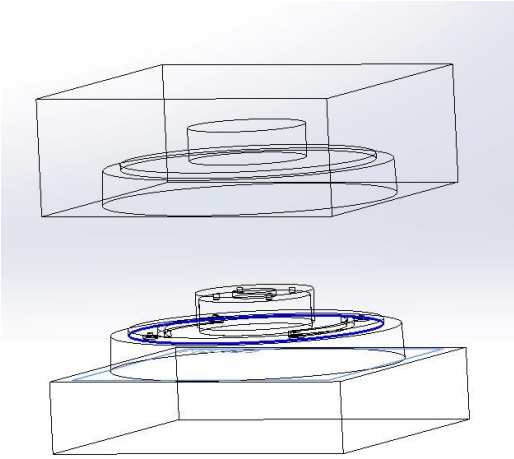


Fig: Stepper motor mount Cavity



Chapter 4

Mathematical modeling

4.1 Overview

The project "3D Mapping with LIDAR" involves using the TFmini-s LIDAR sensor to create a three-dimensional map of the environment. This process includes the sensor's rotational dynamics, data acquisition rates, and spatial data resolution. The key parameters are the total coverage angles (both horizontal and vertical), data points per revolution, data rate, accelerations at rotation end points, constant angular velocity, and points per unit radian.

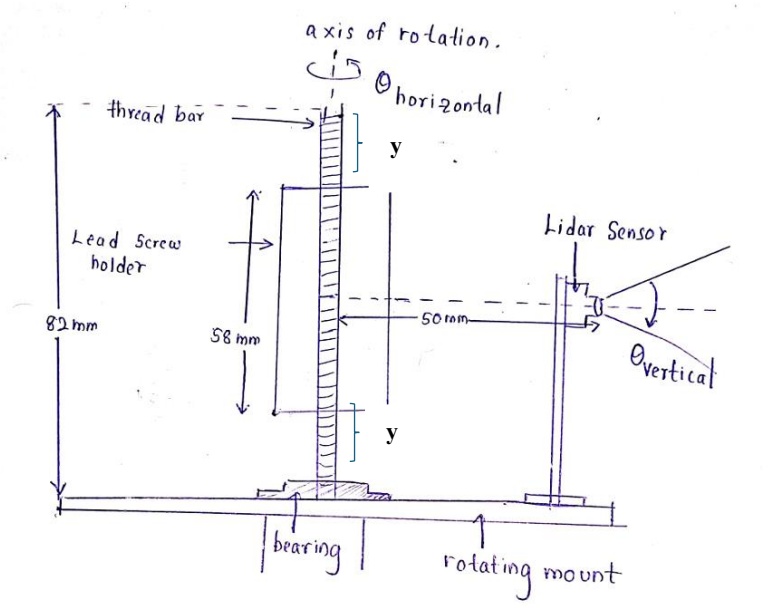


Fig: Annotated Sketch

4.2 Simple mathematical model of our device

- **Total Horizontal Coverage Angle**: This is the angular range within which the LIDAR sensor can scan horizontally. For 3D mapping, the sensor rotates to cover a 360-degree (2π radians) field

$$\theta_{horizontal} = 2\pi$$

- **Total Vertical Coverage Angle**: This is the angular range within which the LIDAR sensor can scan vertically. This range is less than the horizontal coverage.

$$\theta_{verical} = \tan^{-1}\left(\frac{\Delta y}{50}\right)$$

Δy: *pitch of the lead screw* (*The vertical distance moved when the rotating mount rotates one revolution*)

$$\Delta y = 1\ mm$$

$$y = 5.5\ mm$$

$$Vertical\ angle\ per\ one\ revolution = 1.145^{\circ}$$

$$Total\ revolutions\ possible\ in\ vertical\ motion = \frac{2\,y}{1}$$

$$\theta_{Total\ vertical\ coverage} = (Verticalangleperonerevolution)x(Totalrevolutionspossibleinverticalmotion)$$

$$\theta_{Total\ vertical\ coverage} = 12^{\circ} \quad (approx)$$

- **Data Points per Revolution**: The number of data points the LIDAR sensor collects in one full 360-degree horizontal rotation.

$$\omega_i = 0 \quad intial\ angluar\ velocity$$

$$\omega_o = 6\pi \quad final\ angular\ velocity$$

$$\theta_{displacement} = \pi\ angular\ dispalcment$$

We can use the motion equation to find the initial acceleration and final deceleration of the rotation, at these accelerating and decelerating times the data is not ploted or extracted by our implemented algorithms. We can find the initial and final accelerations using the below equation ,

$$\omega_o^2 = \omega_i^2 + 2\alpha\theta_{displacement}$$

$$\alpha = 18\pi \ \ angular\ acceleration$$

- **Data Rate**: The rate at which the sensor collects data, specified in points per second (Hz).

$$Rate\ of\ the\ Lidar\ sensor = 1000\ Hz$$

- **Points per Unit Radian of horizontal angle**: The density of data points per radian of rotation, determining the spatial resolution of the 3D map.

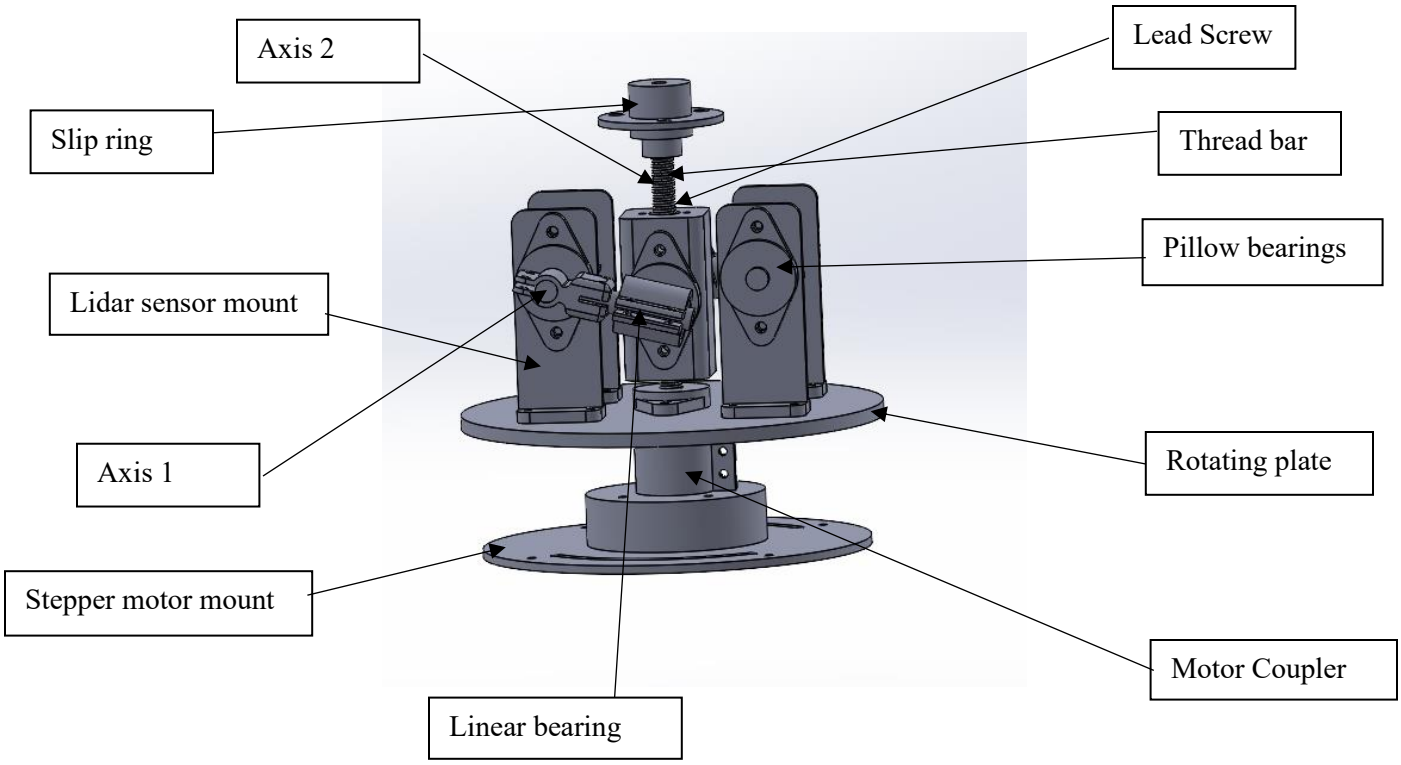
$$Data\ points\ per\ one\ revolution = \frac{Rate\ of\ the\ Lidar\ sensor}{\frac{\omega_o}{2\pi}}$$

$$\text{Points per Unit Radian of horizontal angle} = \frac{Data\ points\ per\ one\ revolution}{2\pi}$$

These parameters together provide a comprehensive mathematical model for understanding and optimizing the performance of the 3D mapping system using the TFmini-s LIDAR sensor. By considering both horizontal and vertical coverage angles, the model ensures accurate 3D spatial representation.

Mechanism Explanation

Our device consist of linear and rotary motions to achieve the distance information accurately, below is the overview of the mechanism of our device, We will be using the below figure to our explanations, which is the internal integrated parts of the device .



Mechanism Description

1. **Stationary Components:**
    - **Thread Bar:** Fixed in position with respect to the ground.
    - **Stepper Motor Mount:** Securely attached to ensure it does not move.
    - **Slip Ring:** Designed to remain stationary, providing a continuous electrical connection to the rotating components including Lidar sensor.
2. **Rotating Components:**
    - **Rotating Plate:** This plate is capable of rotating around a fixed axis, driven by the stepper motor. It forms the base for several other components.
    - **Pillow Bearings:** These bearings support the rotating plate and allow smooth rotational movement. They are mounted on the rotating plate.
    - **LiDAR Sensor Mount:** This mount holds the LiDAR sensor in place, ensuring it rotates along with the rotating plate to capture data from different angles.
    - **Linear Bearings:** These bearings are attached to the rotating plate and support the lead screw, allowing it to move linearly (vertically) as the plate rotates.
3. **Operational Details:**

    - **Vertical Movement (Axis 2):** As the rotating plate turns, the lead screw (thread bar) moves up or down along the vertical axis (Axis 2). This vertical motion is controlled by the rotational direction of the stepper motor, which dictates the direction of the lead screw's movement.
    - **Adjusting Length for Rotary Motion (Axis 1):** The linear bearing, integrated with the lead screw, adjusts the length of a steel bar. This adjustment allows the steel bar to move linearly in response to the rotation of the lead screw. Consequently, this enables the rotating plate to achieve rotary motion along the horizontal axis (Axis 1).

Chapter 5

Software Implementation

5.1.1 Introduction

In our project integrating Arduino with Open3D for 3D data processing and visualization, we aimed to leverage Arduino's microcontroller capabilities and Open3D's powerful tools for manipulating and visualizing 3D data. This setup allows us to process point cloud data from sensors like LIDAR and visualize it effectively, enhancing our 3D mapping capabilities.

5.1.2 What is Open3D

Introduction to Open3D and Its Functionalities

Open3D is a modern open-source library designed for processing and visualizing 3D data. It provides a comprehensive set of tools and algorithms that facilitate various tasks related to 3D point cloud processing, reconstruction, and visualization. Developed primarily in C++ with Python bindings, Open3D is known for its flexibility, ease of use, and performance across a wide range of applications in computer vision, robotics, augmented reality, and more.

Key Functionalities of Open3D Include:

1. **3D Data Manipulation:** Open3D offers powerful capabilities for manipulating 3D point cloud data. It includes functions for loading, saving, filtering, and transforming point clouds, enabling users to preprocess raw 3D data efficiently.
2. **Surface Reconstruction:** It provides algorithms for surface reconstruction from point clouds, allowing users to create meshes or surfaces from unstructured 3D data. This is essential for creating solid representations from scanned objects or environments.
3. **Visualization:** Open3D features advanced visualization tools for rendering 3D scenes, point clouds, and meshes. It supports interactive visualization in both desktop and Jupyter notebook environments, making it suitable for visualizing complex 3D datasets.
4. **Registration and Alignment:** The library includes algorithms for point cloud registration and alignment, facilitating the alignment of multiple scans or models into a unified coordinate system. This is crucial for tasks such as object recognition, scene reconstruction, and localization.
5. **Integration with Machine Learning:** Open3D integrates seamlessly with popular machine learning frameworks like TensorFlow and PyTorch. It provides utilities for data integration, preprocessing, and feature extraction from 3D data, enabling the application of machine learning techniques to 3D perception tasks.
6. **Python and C++ APIs:** With comprehensive Python and C++ APIs, Open3D supports rapid prototyping and development of 3D applications. Users can leverage high-level Python functionalities for quick experimentation or utilize the performance benefits of C++ for production-level implementations.

In summary, Open3D serves as a versatile toolkit for researchers, developers, and engineers working with 3D data. Its rich set of functionalities empowers users to explore, analyze, and manipulate 3D environments and objects effectively, contributing to advancements in fields ranging from robotics and autonomous systems to digital manufacturing and cultural heritage preservation.

5.1.3 Installation Guide

1. **Install Python and Required Packages:**

- First, we ensured Python was installed on our system from python.org.
- Using pip, we installed Open3D and NumPy to handle 3D data processing:

```
pip install open3d
pip install open3d
```

2. **Install pyserial for Serial Communication:**
   - To facilitate communication between Python and Arduino, we installed pyserial:

```
python -m pip install pyserial
```

3. **Set Up Microchip Studio IDE:**
   - We downloaded and installed the Microchip Studio from Microchip Studio official Software.
   - Within the IDE, we developed and uploaded our C++ program that defines sensor interactions and data transmission protocols.
   - The C++ scripts can be developed in a GCC C Executable Project .
4. **Develop Python Scripts for Integration:**
   - Using Open3D and pyserial libraries, we crafted Python scripts.
   - These scripts receive data from Arduino via serial communication, process 3D point cloud data, and utilize Open3D's visualization tools for rendering and analyzing the data.
5. **Testing and Debugging:**
   - Rigorous testing was conducted to ensure smooth communication between C++ program and Python scripts.
   - We verified data integrity and correctness of 3D visualizations using Open3D, ensuring accurate representation of the captured point cloud data.
6. **Optimization and Advanced Development:**
   - We optimized Python scripts for performance, leveraging Open3D's Python and C++ APIs for efficiency.
   - Additionally, we explored integrating machine learning frameworks with Open3D for enhanced data analysis capabilities and automation in our 3D mapping project.

Conclusion: Through this comprehensive setup process, combining Arduino for microcontroller programming with Open3D for robust 3D data processing and visualization, we've established a versatile platform for our project. This setup not only enables precise manipulation and visualization of 3D data but also supports future enhancements and applications in advanced mapping and spatial analysis tasks.

5.1.4 Setting up with the TFmini Software

The TFMini software provides robust capabilities for testing TFMini sensors. While it's a free tool, it's exclusively compatible with the Windows platform.

We can download this software from the official site of Benewake.

http://en.benewake.com/DataDownload/index_pid_20_lcid_22.html.

Prior Requirements are USB to Serial Convertor , TFmini-s LIDAR, and a micro USB to USB cable .

Fig: FT232RL

Fig: TFmini-s

Fig: Adapter

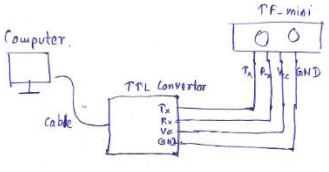


Fig: connection setup

5.1.5 Connection Setup

The connection can be carried out as shown above in (fig: Connection setup) ,

- Open the downloaded benewake software from the link given above.
- The program GUI will be displayed as below(Fig: GUI)
- Find the settings section and choose the product as TFminiS.
- Next, select the COM port to which the TFMini-S is Connected.
- Finally, Press the connect button .

Fig : Settings menu

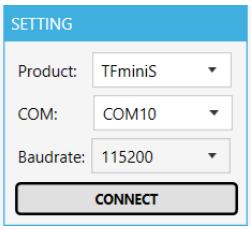


Fig : GUI

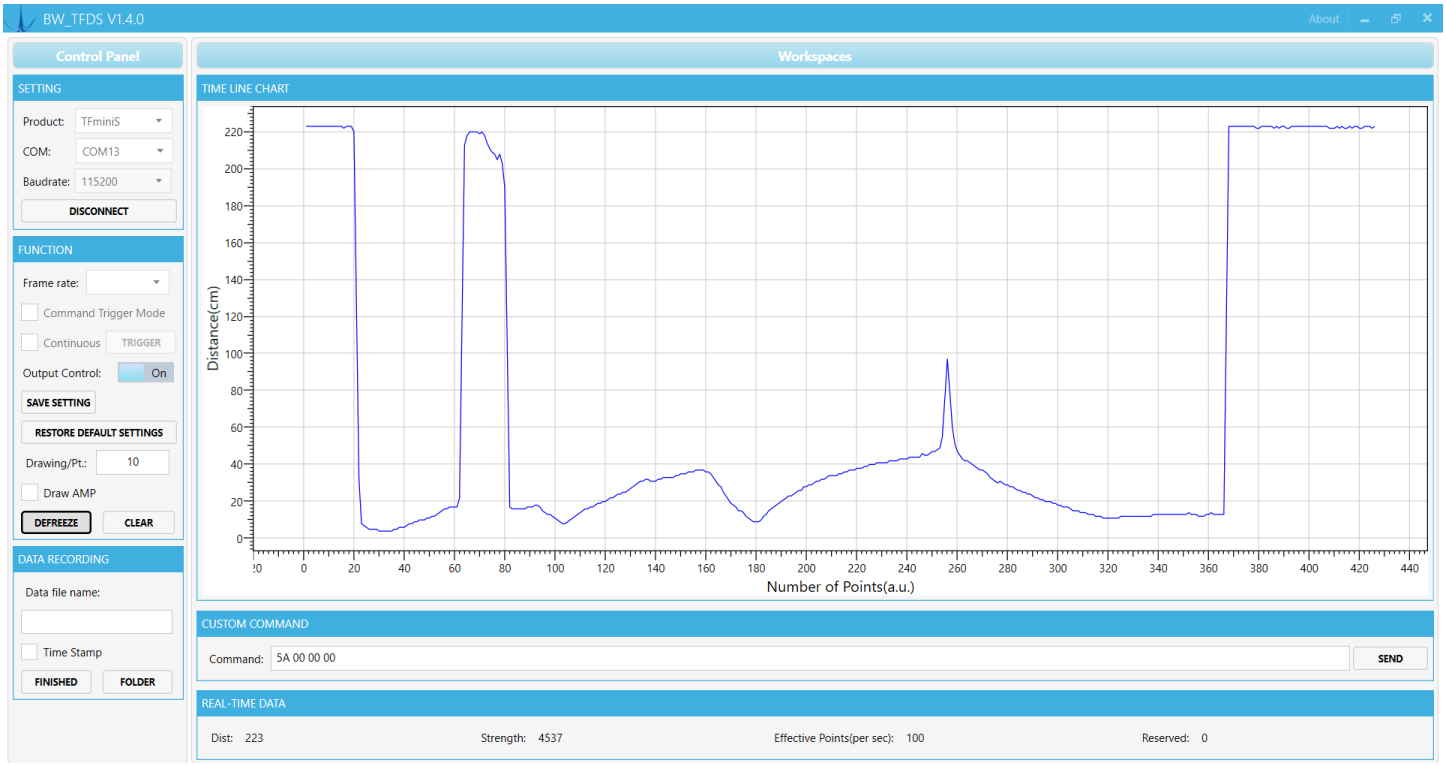
Once the device is connected, the program starts displaying a distance-over-time waveform in the “Time Line Chart” section. Below, in the “Real Time Data” section, you can see the current distance ,the number of effective data points per second , and the signal strength .

Connection Check: If no data appears in "TIME LINE CHART," check the line connection and sequence. A red indicator light inside the transmitting lens indicates successful power-on.

Distance Output Units: The default distance unit is centimeters (cm). If changed to millimeters (mm), the PC software may not recognize it, causing display issues. For example, 1 meter (1000 mm) will still show as 1000 cm in the software.

Pixhawk Format: To use Pixhawk format, select "Pix Mode" first. This will ensure correct data display in "TIME LINE CHART" and automatically change the distance unit to meters.

A graph shown as below should appear when the distance to the LIDAR is Varied,



5.1.6 TFmini Serial port communication and dataframe format

The TFmini-S sensor utilizes a serial port data communication protocol for transmitting data. This protocol is detailed in the following specifications:

- **Communication Interface**: The sensor communicates via a UART (Universal Asynchronous Receiver/Transmitter) interface, which is widely used for serial communication.
- **Default Baud Rate**: The default baud rate for data transmission is set at 115200. This rate determines the speed at which data is sent and received, measured in bits per second (bps).
- **Data Bit**: Each data packet consists of 8 bits. The data bit length indicates the number of bits in each segment of transmitted data, ensuring the accuracy and integrity of the information.
- **Stop Bit**: The protocol uses 1 stop bit. The stop bit signals the end of a data packet, allowing the receiving device to recognize the completion of the data transmission.
- **Parity Check**: There is no parity check used in this communication protocol. Parity check is a method of error detection, and in this case, the protocol relies on other means to ensure data accuracy.

Communication interface	UART
Default baud rate	115200
Data bit	8
Stop bit	1
Parity check	None

5.1.7 TFmini Dataframe format

The data structure of the TFmini-S sensor is organized into frames, each containing 9 bytes. These frames include crucial information such as the distance value, signal strength, chip temperature, and a checksum for error detection. The data format is in hexadecimal (HEX). The detailed breakdown is as follows:

- **Byte0 (0x59)**: Frame header, constant for each frame
- **Byte1 (0x59)**: Frame header, constant for each frame
- **Byte2 (Dist_L)**: Lower 8 bits of the distance value
- **Byte3 (Dist_H)**: Higher 8 bits of the distance value
- **Byte4 (Strength_L)**: Lower 8 bits of the signal strength
- **Byte5 (Strength_H)**: Higher 8 bits of the signal strength
- **Byte6 (Temp_L)**: Lower 8 bits of the chip temperature
- **Byte7 (Temp_H)**: Higher 8 bits of the chip temperature
- **Byte8 (Checksum)**: Lower 8 bits of the cumulative sum of the first 8 bytes

The frame headers (Byte0 and Byte1) are always 0x59, ensuring consistent identification of each frame. The checksum (Byte8) is used to verify the integrity of the data.

Byte 0-1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**
0x59 59	Dist_L	Dist_H	Strength_L	Strength_H	Temp_L	Temp_H	Checksum

In Summary ,

The TFmini-S sensor uses a UART interface for data communication, with a default baud rate of 115200 bps, 8 data bits, 1 stop bit, and no parity check. Its data frame format consists of 9 bytes in hexadecimal, including a constant frame header (0x59 59), lower and higher bits of the distance value, signal strength, chip temperature, and a checksum for data integrity. The consistent headers ensure frame identification, and the checksum verifies the accuracy of the transmitted data.

5.1.8 Parameter Configuration based on user requirements

Empower yourself with the TFmini-S's user-defined configuration feature, offering flexible solutions tailored to your needs. Easily adjust parameters such as data format and frame rate by employing relevant commands.

Key features are,

1. **Customizable Parameters**:
    - Tailor settings to meet specific requirements.
    - Commands adjust crucial aspects like data format and frame rate.
2. **Usage Recommendations**:
    - Optimize product configurations according to practical needs.
    - Avoid unnecessary commands to prevent unintended adjustments.
3. **Command Protocol**:
    - Adhere strictly to listed commands for accurate configurations.
    - Ensure commands are in line with manual instructions for optimal functionality.

5.1.9 Commands Manipulation

In handling multi-byte data or command frames, the TFmini-S employs a little endian format. For instance, a decimal value such as 1000 (represented as 0x03E8 in hexadecimal) would be transmitted and stored as follows: 0x5A 0x06 0x03 **0xE8 0x03** 0x4E. Commands are initiated as data instruction frames from a PC directed towards the LiDAR system. Conversely, responses are conveyed as data instruction frames from the LiDAR system back to the host computer or other connected terminals.

Command frame format

Byte0	Byte1	Byte2	Byte3 – ByteN-2	ByteN-1
Head	Length	ID	Payload	Checksum

Some of the available commands are Frame rate , Output format, Baud rate , Restore factory settings, Save settings. To satisfy our project requirements we used the command corresponding to increasing the frame rate. Below is the Frame rate (parameter) configuration .

Parameter	Command	Response	Remark	Default Setting
Frame rate	5A 06 03 LL HH SU	5A 06 03 LL HH SU	1-1000Hz	100Hz

Table : Frame Rate configuring command

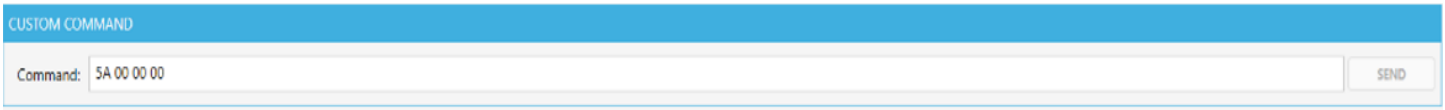
In summary,

The TFmini-S offers a user-defined configuration feature that allows for tailored adjustments to parameters like data format and frame rate through specific commands. Users can customize settings to meet precise operational needs while ensuring efficient product configurations. It is crucial to follow the command protocols outlined in the manual to achieve accurate configurations and avoid unintended adjustments. In handling data transmission, the TFmini-S utilizes a little endian format for multi-byte frames, ensuring compatibility and effective communication between the PC and LiDAR system. Commands are initiated as data instruction frames from the PC to the LiDAR, with responses transmitted back as data instruction frames to the host computer or other connected terminals. This structured approach enhances usability and flexibility in integrating the TFmini-S into various applications, supporting seamless parameter adjustments and operational efficiency.

5.1.10 Running a command

The software we initially installed can be used to input commands into TFmini-s, The input data is received from the Rx in the Lidar sensor. As the parameter configuration table shown above the response should follow after the command. Following the below steps we can enter commands to the configure the TFmini-s to our requirements in the range of product capabilities.

Step 1: Go to the custom command tab in the software GUI visible in the main interface as soon as opened.



Step 2: Enter the command in the space provided and hit **Send.**

Step 3: Observe the response, If it matches with command that was entered it successful.

Step 4: Now use the save settings command , below is the general parameter description of saving the settings

Parameter	Command	Response	Remark	Default setting
Save settings	5A 04 11 6F	5A 05 11 00 70 5A 05 11 01 71	Succeeded Failed	

Table : Save settings configuring command [10]

5.1.11 Troubleshooting

If the command is unsuccessful try the command Restore factory settings and redo the above procedure. Below is the command configurations of the Restoring factory settings.

Parameter	Command	Response	Remark	Default setting
Restore Factory settings	5A 04 10 6E	5A 05 10 00 6F 5A 05 10 01 70	Succeeded Failed	

Table : restore factory settings configuring command [10]

NOTE:

- **Update Rate Calculation**: The default update rate for the TFmini-S LiDAR is 100Hz. Custom update rates should follow the formula 1000/n (where n is a positive integer). Increasing the frame rate may lead to reduced data stability.
- **Baud Rate Compatibility**: Only standard baud rates are supported. For higher update rates, it's recommended to use a higher baud rate to ensure secure data transmission.
- **Parameter Modification**: Always execute the "save settings" command after modifying TFmini-S parameters. Wait for 1 second post-command execution to ensure changes take effect properly; failure to do so may result in settings not being applied.

5.1.12 Setting up and testing Open3D point clouds

To verify that Open3D is correctly installed in your Python environment, we can run the following test script in Python's IDLE or any Python interpreter

```
import open3d as o3d
import numpy as np

# Function to generate random points on a sphere of radius 1
def random_point_on_sphere(num_points):
    phi = np.random.uniform(0, 2 * np.pi, num_points)
    cos_theta = np.random.uniform(-1, 1, num_points)
    theta = np.arccos(cos_theta)
    x = np.sin(theta) * np.cos(phi)
    y = np.sin(theta) * np.sin(phi)
    z = np.cos(theta)
    points = np.vstack((x, y, z)).T
    return points

# Generate random points on a sphere
num_points = 1000
points = random_point_on_sphere(num_points)

# Create a point cloud object
pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(points)

# Visualize the point cloud
o3d.visualization.draw_geometries([pcd])
```

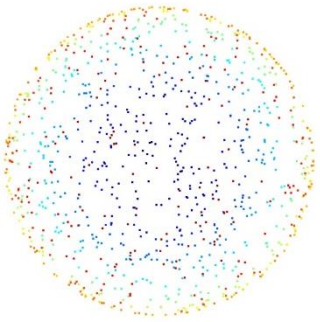


Fig : output

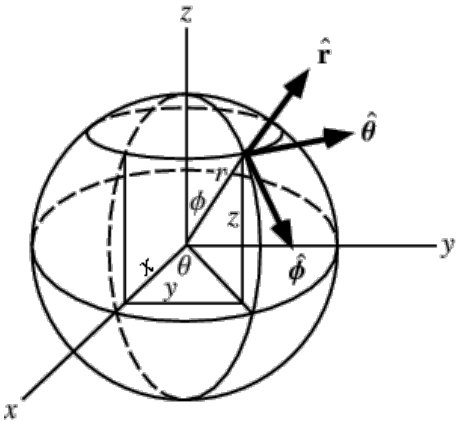


Fig : Spherical coordinates

The code leverages Open3D and NumPy to generate and visualize random points on the surface of a unit sphere. It defines a function, random_point_on_sphere, that calculates random spherical coordinates and converts them to Cartesian coordinates. By generating 1,000 such points, the code creates a dense point cloud representing the sphere's surface. This point cloud is then visualized using Open3D's draw_geometries function. The visualization shows a scatter plot of points uniformly distributed on the sphere, providing an intuitive view of the spherical distribution. This approach effectively demonstrates how to generate and visualize random 3D data

5.1.13 Detailed Program

```
/*
 * Original Code for Stepper Motor and LiDAR Control
 * ------------------------------------------------------------------------------
 * Contributors: Dineth Perera and Seleka Deemantha
 * The implementation is done without using any external libraries.
 * ------------------------------------------------------------------------------
 * Description: This program is designed to control a stepper motor using an A4988 driver
 * and a TFmini-S LiDAR sensor interfaced with an ATmega328P microcontroller. The program
 * includes functionality for motor acceleration, deceleration, and constant speed
rotation
 * in both clockwise and counterclockwise directions. Additionally, it processes LiDAR
sensor
 * data to measure distances.
 * ------------------------------------------------------------------------------
 * Stepper Motor Parameters:
 * - STEP_PIN (PD6) and DIR_PIN (PD5) control the motor's step and direction.
 * - MCU_RX (PD2) and MCU_TX (PD3) handle hardware serial communication.
 * - RX (PD0) and TX (PD1) handle FT232RL module communication.
 * - STEPS_PER_REVOLUTION defines the number of steps for a full revolution.
 * - FINAL_VELOCITY, DISPLACEMENT, and acceleration are used for calculating motor
dynamics.
 * - totalSteps calculates the total number of steps based on the displacement.
 * ------------------------------------------------------------------------------
 * Microstepping Configuration:
 * - MS_1 (PB1), MS_2 (PB0), and MS_3 (PD7) configure the microstepping mode.
 * ------------------------------------------------------------------------------
 * LiDAR Sensor Parameters:
 * - dist, strength, check, and uart array store the LiDAR measurement data.
 * - HEADER defines the frame header for the LiDAR data package.
 * ------------------------------------------------------------------------------
 * References : ATmega328P Datasheet
 *            : https://www.youtube.com/watch?v=dT0xxaG1DhM&list=PLD7F7ED1F3505D8D5
 *
 */


#define F_CPU 16000000UL  // Define CPU frequency for delay functions

#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>
#include <avr/interrupt.h>
#include <stdlib.h> // for itoa (integer to ASCII)
#include <math.h>

// Stepper motor parameters
#define STEP_PIN PD6  // STEP pin (Pin 10)
#define DIR_PIN PD5   // DIR pin (Pin 9)
#define MCU_RX PD2    // hardware serial port (Pin 32)
```

```
#define MCU_TX PD3    // hardware receiver port (Pin 1)
#define RX PD0        // from the FT232RL to MCU which is MCU's Rx and FT232Rl's
Transmitter (Pin 30)
#define TX PD1        // from the MCU to the FT232RL where FT232RL's Receiver and MCU Tx
(Pin 31)
#define MS_1 PB1
#define MS_2 PB0
#define MS_3 PD7

const int STEPS_PER_REVOLUTION = 200; // Number of steps per revolution
const float FINAL_VELOCITY = 6 * M_PI; // Final angular velocity in rad/s
const float DISPLACEMENT = M_PI;       // Displacement in radians

float acceleration = 0.0;  // Angular acceleration (to be calculated)
int totalSteps = 0;        // Total steps for the motor movement

// LiDAR sensor parameters
int dist = 0;       // Actual distance measurements of LiDAR
int strength = 0;   // Signal strength of LiDAR
int check = 0;      // Save check value
int uart[9];        // Save data measured by LiDAR
const int HEADER = 0x59;  // Frame header of data package

void acceleration_cw();
void rotation_cw();
void deceleration_cw();
void acceleration_ccw();
void rotation_ccw();
void deceleration_ccw();
void delay_T(int delay_time);
void Lidar_data();

void setup() {
    // Set pins as outputs/inputs
    DDRD |= (1 << STEP_PIN) | (1 << DIR_PIN); // STEP and DIR pins as output
    DDRD &= ~(1 << MCU_RX); // MCU_RX as input
    DDRD |= (1 << MCU_TX);  // MCU_TX as output
    DDRD &= ~(1 << RX);     // RX as input
    DDRD |= (1 << TX);      // TX as output

    // micro stepping pins
    DDRB |= (1 << MS_1);
    DDRB |= (1 << MS_2);
    DDRD |= (1 << MS_3);

    // initial the micro stepping pins are set to LOW logic
    PORTB &= ~(1 << MS_1);
    PORTB &= ~(1 << MS_2);
    PORTD &= ~(1 << MS_3);
```

```
    // Calculate angular acceleration
    acceleration = FINAL_VELOCITY * FINAL_VELOCITY / (2 * DISPLACEMENT);

    // Calculate total steps
    totalSteps = (DISPLACEMENT / (2 * M_PI)) * STEPS_PER_REVOLUTION;
}

void USART_Init(int ubrr){ // argument is the baudrate
//Referring to the Data sheet Atmega328P page[149]

    // Initialize serial communication for UART0
    UBRR0H = 0;
    UBRR0L = ubrr;  // Baud rate 9600 for 16MHz clock
    UCSR0B = (1 << RXEN0) | (1 << TXEN0); // Enable RX and TX
    UCSR0C = (1 << USBS0) | (1 << UCSZ00); // 8-bit data, 1 stop bit
}


int main() {
    setup();
    USART_Init(9600) ;
    while (1) {
        Lidar_data();

        acceleration_cw();
        rotation_cw();
        deceleration_cw();

        acceleration_ccw();
        rotation_ccw();
        deceleration_ccw();
    }
    return 0;
}

void acceleration_cw() {
    float currentVelocity = 0.1; // Start with a small initial velocity to avoid division
by zero
    float stepInterval;

    PORTB |= (1 << DIR_PIN); // Set DIR_PIN HIGH for clockwise direction
    for (int step = 0; step < totalSteps; step++) {
        stepInterval = 1.0 / (STEPS_PER_REVOLUTION * currentVelocity / (2 * M_PI));

        PORTB |= (1 << STEP_PIN); // Set STEP_PIN HIGH
        delay_T(ceil(stepInterval * 1e6)); // Adjust the pulse width if necessary
        PORTB &= ~(1 << STEP_PIN); // Set STEP_PIN LOW
        delay_T(ceil(stepInterval * 1e6));
```

```
        currentVelocity = sqrt(currentVelocity * currentVelocity + 2 * acceleration *
(1.0 / STEPS_PER_REVOLUTION));
    }
}

void rotation_cw() {
    int step_time = 1.0 / (STEPS_PER_REVOLUTION * FINAL_VELOCITY / (2 * M_PI));

    PORTB |= (1 << DIR_PIN); // Set DIR_PIN HIGH for clockwise direction
    for (int i = 0; i < 200 * 11; i++) {
        PORTB |= (1 << STEP_PIN); // Set STEP_PIN HIGH
        _delay_us(ceil((step_time / 2)*1e6)); // Adjust delay for step speed
        PORTB &= ~(1 << STEP_PIN); // Set STEP_PIN LOW
        _delay_us(ceil((step_time / 2)*1e6));
    }
}

void deceleration_cw() {
    float currentVelocity = FINAL_VELOCITY;
    float stepInterval;

    PORTB |= (1 << DIR_PIN); // Set DIR_PIN HIGH for clockwise direction
    for (int step = 0; step < totalSteps; step++) {
        stepInterval = 1.0 / (STEPS_PER_REVOLUTION * currentVelocity / (2 * M_PI));

        PORTB |= (1 << STEP_PIN); // Set STEP_PIN HIGH
        delay_T(ceil(stepInterval * 1e6)); // Adjust the pulse width if necessary
        PORTB &= ~(1 << STEP_PIN); // Set STEP_PIN LOW

        delay_T(ceil(stepInterval * 1e6));

        currentVelocity = sqrt(currentVelocity * currentVelocity - 2 * acceleration *
(1.0 / STEPS_PER_REVOLUTION));
    }
}

void acceleration_ccw() {
    float currentVelocity = 0.1; // Start with a small initial velocity to avoid division
by zero
    float stepInterval;

    PORTB &= ~(1 << DIR_PIN); // Set DIR_PIN LOW for counterclockwise direction
    for (int step = 0; step < totalSteps; step++) {
        stepInterval = 1.0 / (STEPS_PER_REVOLUTION * currentVelocity / (2 * M_PI));

        PORTB |= (1 << STEP_PIN); // Set STEP_PIN HIGH
        delay_T(ceil(stepInterval * 1e6)); // Adjust the pulse width if necessary
        PORTB &= ~(1 << STEP_PIN); // Set STEP_PIN LOW
        delay_T(ceil(stepInterval * 1e6));
```

```
        currentVelocity = sqrt(currentVelocity * currentVelocity + 2 * acceleration *
(1.0 / STEPS_PER_REVOLUTION));
    }
}

void rotation_ccw() {
    int step_time = 1.0 / (STEPS_PER_REVOLUTION * FINAL_VELOCITY / (2 * M_PI));

    PORTB &= ~(1 << DIR_PIN); // Set DIR_PIN LOW for counterclockwise direction
    for (int i = 0; i < 200 * 11; i++) {
        PORTB |= (1 << STEP_PIN); // Set STEP_PIN HIGH
        _delay_us(ceil((step_time / 2)*1e6)); // Adjust delay for step speed
        PORTB &= ~(1 << STEP_PIN); // Set STEP_PIN LOW
        _delay_us(ceil((step_time / 2)*1e6));
    }
}

void deceleration_ccw() {
    float currentVelocity = FINAL_VELOCITY;
    float stepInterval;

    PORTB &= ~(1 << DIR_PIN); // Set DIR_PIN LOW for counterclockwise direction
    for (int step = 0; step < totalSteps; step++) {
        stepInterval = 1.0 / (STEPS_PER_REVOLUTION * currentVelocity / (2 * M_PI));

        PORTB |= (1 << STEP_PIN); // Set STEP_PIN HIGH
        delay_T(ceil(stepInterval * 1e6)); // Adjust the pulse width if necessary
        PORTB &= ~(1 << STEP_PIN); // Set STEP_PIN LOW
        delay_T(ceil(stepInterval * 1e6));

        currentVelocity = sqrt(currentVelocity * currentVelocity - 2 * acceleration *
(1.0 / STEPS_PER_REVOLUTION));
    }
}

// delay function
void delay_T(int  delay_time){
    int Timer_Temp = 0 ;
    while (Timer_Temp < delay_time){
        Timer_Temp = Timer_Temp + 1 ;
    }
}

//uart communication
//Referring to the Data sheet Atmega328P page[152]
//Referring to the Data sheet Atmega328P page[151]

void Lidar_data() {
    if (UCSR0A & (1 << RXC0)) { // Check if serial port has data input
        if (UDR0 == HEADER) { // Check data package frame header 0x59
```

```
            uart[0] = HEADER;
            if (UDR0 == HEADER) { // Check data package frame header 0x59
                uart[1] = HEADER;
                for (int i = 2; i < 9; i++) { // Save data in array
                    uart[i] = UDR0;
                }
                check = uart[0] + uart[1] + uart[2] + uart[3] + uart[4] + uart[5] +
uart[6] + uart[7];
                if (uart[8] == (check & 0xff)) { // Verify received data as per protocol
                    dist = uart[2] + uart[3] * 256; // Calculate distance value

                    // Convert dist to a string
                    char distStr[10];
                    itoa(dist, distStr, 10);

                    // Send dist string over UART
                    for (int i = 0; distStr[i] != '\0'; i++) {
                        while (!(UCSR0A & (1 << UDRE0)));
                        UDR0 = distStr[i];
                    }
                }
            }
        }
    }
}
```

5.1.14 Program Explanation

This code is designed to control a stepper motor using an A4988 driver and a TFmini-S LiDAR sensor interfaced with an ATmega328P microcontroller. It manages motor acceleration, deceleration, and constant speed rotation in both clockwise and counterclockwise directions, while also processing LiDAR sensor data for distance measurements. The code does not rely on any external libraries.

Preprocessor Directives and Includes

- #define F_CPU 16000000UL sets the CPU frequency to 16 MHz, which is used for delay functions.
- The necessary libraries are included:
  - <avr/io.h> for input/output operations.
  - <util/delay.h> for delay functions.
  - <stdint.h> for standard integer types.
  - <avr/interrupt.h> for interrupt handling.
  - <stdlib.h> for general utilities (e.g., itoa function).
  - <math.h> for mathematical functions.

Stepper Motor Parameters

- #define STEP_PIN PD6 and #define DIR_PIN PD5 define the pins controlling the motor's step and direction, respectively.
- #define MCU_RX PD2 and #define MCU_TX PD3 define the pins for hardware serial communication.
- #define RX PD0 and #define TX PD1 define the pins for FT232RL module communication.

- #define MS_1 PB1, #define MS_2 PB0, and #define MS_3 PD7 define the pins for configuring the microstepping mode.
- const int STEPS_PER_REVOLUTION = 200 specifies the number of steps for a full revolution.
- const float FINAL_VELOCITY = 6 * M_PI sets the final angular velocity in radians per second.
- const float DISPLACEMENT = M_PI sets the displacement in radians.
- float acceleration = 0.0 is a placeholder for angular acceleration, to be calculated later.
- int totalSteps = 0 will hold the total number of steps for the motor movement.

LiDAR Sensor Parameters

- int dist = 0 stores the actual distance measurements from the LiDAR.
- int strength = 0 stores the signal strength of the LiDAR.
- int check = 0 is used to save the check value for data verification.
- int uart[9] is an array to store data measured by the LiDAR.
- const int HEADER = 0x59 defines the frame header for the LiDAR data package.

Function Prototypes

- void acceleration_cw(), void rotation_cw(), and void deceleration_cw() control the clockwise acceleration, rotation, and deceleration of the motor, respectively.
- void acceleration_ccw(), void rotation_ccw(), and void deceleration_ccw() control the counterclockwise acceleration, rotation, and deceleration of the motor, respectively.
- void delay_T(int delay_time) is a custom delay function.
- void Lidar_data() handles the LiDAR data processing.

Setup Function

The setup() function initializes the microcontroller's pins and calculates initial motor parameters:

- DDRD |= (1 << STEP_PIN) | (1 << DIR_PIN) sets the STEP and DIR pins as outputs.
- DDRD &= ~(1 << MCU_RX) sets MCU_RX as input.
- DDRD |= (1 << MCU_TX) sets MCU_TX as output.
- DDRD &= ~(1 << RX) sets RX as input.
- DDRD |= (1 << TX) sets TX as output.
- DDRB |= (1 << MS_1), DDRB |= (1 << MS_2), and DDRD |= (1 << MS_3) set the microstepping pins as outputs.
- PORTB &= ~(1 << MS_1), PORTB &= ~(1 << MS_2), and PORTD &= ~(1 << MS_3) initialize the microstepping pins to LOW logic.
- acceleration = FINAL_VELOCITY * FINAL_VELOCITY / (2 * DISPLACEMENT) calculates the angular acceleration.
- totalSteps = (DISPLACEMENT / (2 * M_PI)) * STEPS_PER_REVOLUTION calculates the total number of steps.

USART Initialization

The USART_Init(int ubrr) function initializes the serial communication for UART0:

- UBRR0H = 0 and UBRR0L = ubrr set the baud rate to 9600 for a 16 MHz clock.
- UCSR0B = (1 << RXEN0) | (1 << TXEN0) enables RX and TX.
- UCSR0C = (1 << USBS0) | (1 << UCSZ00) configures 8-bit data and 1 stop bit.

Main Function

The main() function runs the setup, initializes UART, and enters an infinite loop:

- setup() is called to initialize the hardware.
- USART_Init(9600) initializes the serial communication.
- In the infinite loop:
    - Lidar_data() processes LiDAR data.
    - acceleration_cw(), rotation_cw(), and deceleration_cw() handle the clockwise motor movement.
    - acceleration_ccw(), rotation_ccw(), and deceleration_ccw() handle the counterclockwise motor movement.

Motor Control Functions

- acceleration_cw() gradually increases the motor's speed in the clockwise direction by adjusting the step interval.
- rotation_cw() maintains a constant speed rotation in the clockwise direction.
- deceleration_cw() gradually decreases the motor's speed in the clockwise direction.
- acceleration_ccw() gradually increases the motor's speed in the counterclockwise direction.
- rotation_ccw() maintains a constant speed rotation in the counterclockwise direction.
- deceleration_ccw() gradually decreases the motor's speed in the counterclockwise direction.

Delay Function

The delay_T(int delay_time) function creates a delay by incrementing a temporary variable up to the specified delay time.

LiDAR Data Processing

The Lidar_data() function processes the data from the LiDAR sensor:

- It checks if the serial port has data and verifies the frame header.
- Data is stored in the uart array and a checksum is calculated.
- If the checksum matches, the distance value is calculated and sent over UART as a string.

5.1.15 How we setup the programming interface

Step 1: Download and Install Microchip Studio

1. **Visit the Microchip Studio website**:
    - Go to Microchip Studio website.
2. **Download Microchip Studio**:
    - Click on the "Download Microchip Studio" button.
    - Select the appropriate installer for your operating system.
3. **Install Microchip Studio**:
    - Run the downloaded installer.
    - Follow the on-screen instructions to complete the installation.
    - You may need to restart your computer after the installation.

Step 2: Set Up Microchip Studio

1. **Launch Microchip Studio**:
    - Open Microchip Studio from your Start menu or desktop shortcut.
2. **Create a New Project**:
    - Go to File > New > Project....
    - Select GCC C Executable Project and click OK.
    - Choose the AVR/GNU C Compiler.
    - Enter a name and location for your project, then click OK.
3. **Select the Device**:
    - In the Device Selection window, select ATmega328P and click OK.

Step 3: Write or Import Your C Code

1. **Add Your C Code**:
    - In the Solution Explorer pane, right-click on the Source Files folder.
    - Select Add > New Item....
    - Choose C File (.c) and name it .

Step 4: Build the Project

1. **Build the Project**:
    - Click on Build > Build Solution or press F7.
    - Check the Output pane for any compilation errors or warnings.

Step 5: Configure the Programmer

1. **Connect the Programmer**:
    - Connect your AVR programmer USBasp to the computer and to the ATmega328P-AU chip.
    - Ensured that our target board has the necessary connections specifically power, GND, MISO, MOSI, SCK, RESET.
2. **Set the Tool**:
    - Go to Tools > Device Programming.
    - Select your programmer from the Tool dropdown.
    - Select ATmega328P from the Device dropdown.
    - Click Apply.

Step 6: Upload the Code to the ATmega328P

1. **Upload the Code**:
    - In the Device Programming window, go to the Memories tab.
    - Click on the Program button to upload the compiled code to the ATmega328P.

Step 7: Debug the Code

1. **Start Debugging**:
    - Set breakpoints in your code by clicking in the left margin next to the line numbers.
    - Click on Debug > Start Debugging and Break or press Alt + F5.
    - The debugger will upload the code and halt at the first breakpoint.
2. **Use Debugging Tools**:
    - Use the Step Over (F10), Step Into (F11), and Step Out (Shift + F11) commands to navigate through your code.
    - Inspect variables and registers using the Watch and Registers windows.
    - Use the Output and Call Stack windows to get more information about the execution flow and errors.
3. **Stop Debugging**:
    - Click on Debug > Stop Debugging or press Shift + F5 to stop the debugging session.

Step 8: Final Testing

1. **Disconnect and Test**:
    - Once your code is uploaded and debugged, disconnect the programmer.
    - Power the ATmega328P-AU chip and test your stepper motor and LiDAR setup to ensure everything works as expected.

By following these steps, we were able to run and debug your C code on the AVR ATmega328P-AU chip using Microchip Studio.

5.1.16 Python code for Data extraction & Visualization

```
import open3d as o3d
import serial
import numpy as np

# Open the serial port
ser = serial.Serial('/dev/ttyUSB0', 115200)

def read_lidar_data(ser):
    """Reads data from the LiDAR sensor over the serial connection."""
    line = ser.readline().decode('utf-8').strip()
    if line:
        data = list(map(float, line.split(',')))
        if len(data) == 3:
            return data
    return None

def create_point_cloud(points):
    """Creates an Open3D point cloud from a list of points."""
    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(points)
    return pcd

def main():
    points = []

    while True:
        data = read_lidar_data(ser)
        if data:
            points.append(data)
            if len(points) % 100 == 0:  # Update visualization every 100 points
                pcd = create_point_cloud(points)
                o3d.visualization.draw_geometries([pcd])

if __name__ == "__main__":
    main()
```

5.1.17 Data Extraction Code Explanation

This code reads data from a LiDAR sensor and visualizes it as a point cloud using Open3D. Here's a detailed explanation of each part

1. Importing Libraries

- **open3d**: Used for 3D data processing and visualization.
- **serial**: Handles serial communication with the LiDAR sensor.
- **numpy**: Provides support for large, multi-dimensional arrays and matrices.

2. Opening the Serial Port

- **serial.Serial('/dev/ttyUSB0', 115200)**: Opens the serial port at /dev/ttyUSB0 with a baud rate of 115200. This connects the script to the LiDAR sensor.

3.Reading Data from the LiDAR Sensor

read_lidar_data(ser): Reads a line of data from the serial port.

- **ser.readline()**: Reads a line of data from the serial port.
- **.decode('utf-8')**: Decodes the byte string to a UTF-8 string.
- **.strip()**: Removes leading and trailing whitespace.
- **list(map(float, line.split(',')))**: Converts the comma-separated string into a list of floats.
- **if len(data) == 3**: Ensures that exactly three values are read (typically x, y, z coordinates).

4.Creating a Point Cloud

- **create_point_cloud(points)**: Converts a list of 3D points into an Open3D point cloud.

- **o3d.geometry.PointCloud()**: Initializes a point cloud object.
- **o3d.utility.Vector3dVector(points)**: Converts the list of points into a format that Open3D can use.

5.Main Function

- **main()**: The main function that runs the data acquisition and visualization loop.

- **points = []**: Initializes an empty list to store LiDAR data points.
- **while True**: Creates an infinite loop to continuously read data.
    - **data = read_lidar_data(ser)**: Reads a new data point from the LiDAR sensor.
    - **if data**: Checks if valid data was read.
        - **points.append(data)**: Adds the new data point to the list.
        - **if len(points) % 100 == 0**: Checks if the number of points is a multiple of 100.
            - **pcd = create_point_cloud(points)**: Creates a point cloud from the points.
            - **o3d.visualization.draw_geometries([pcd])**: Visualizes the point cloud.

6. Entry Point

- **if name == "main"**: Ensures that the main() function is called only when the script is run directly, not when it is imported as a module.

5.1.18 Conclusion

By following the outlined process, we can effectively analyze the data obtained from our 3D mapping device, LiDAR. The modified code ensures reliable communication between the Arduino and LiDAR sensor, capturing distance measurements and signal strengths accurately. This data is crucial for various applications such as autonomous navigation, environmental mapping, and object detection. With clear variable naming and structured code, the analysis becomes more straightforward, facilitating robust implementation and integration of LiDAR technology in advanced spatial mapping and robotic systems.

Chapter 6

Appendix

6.1.1 Daily Log Entries of our Project

1. Preferred Lidar Sensor Selection

Date: 2024/03/10

Task: Evaluate and select a Lidar sensor.
Details: We compared the VL53L0X and TFmini S3 sensors based on their specifications.
Outcome: We chose the TFmini S3 sensor because it has an operating range of 0.1m to 12m at 90% reflectivity, an accuracy of ±6cm (0.1-6m) and ±1% (6m-12m), a distance resolution of 5mm, and a frame rate of 100Hz. These features align well with our project requirements.

2. Preferred Motors Selection

Date: 2024/03/29

Task: Evaluate and select motors.
Details: We compared Servor motors and Stepper motors (Nema 17).
Outcome: We chose the Nema 17 Stepper motor due to its 2 phases, 200 steps/revolution with micro-stepping, a shaft load capacity of 20,000 hours at 1000 RPM, axial load capacity of 25 N, and holding torque of 0.83 Nm. Servo motors were not selected due to potential backlash introducing significant errors in output readings.

3. Conceptual Design Selection

Date: 2024/03/03 and 2024/03/07

Task: Develop and evaluate conceptual designs.
Details: We created three conceptual designs and documented them in Section 6 of the Design Document.
Outcome:
First Conceptual Design - Date: 2024/03/03
Second Conceptual Design - Date: 2024/03/07

4. Mechanism Selection

Date: 2024/03/18

Task: Select a mechanism based on evaluation criteria.
Details: We evaluated the three conceptual designs based on criteria such as complexity, weight, volume, and the ability to achieve 3D scanning with the selected motor.
Outcome: We selected the third conceptual design as it best met our evaluation criteria, providing a balanced solution in terms of complexity, weight, and functionality.

5. Preferred Microcontroller Selection and Mathematical modeling

Date: 2024/04/04

Task: Select a microcontroller for data retrieval and processing.
Details: We considered the data rate requirements of the Lidar scanner (up to 1000 frames per second, 9 bytes per frame, equating to 72,000 bps) and the default baud rate of the TFmini sensor (115200).
Outcome: We chose the Atmega328P-au microcontroller due to its compatibility with the data rates and lower power consumption compared to other options.

Mathematical Model: We started developing the mathematical model for our design, further refinements are to be done.

6. Preferred Motor Driver IC Selection and Lidar sensor testing with the software installation

Date: 2024/04/07

Task: Select a motor driver IC.
Details: We evaluated the TMC2208, A4988, and DRV8825.
Outcome: We selected the A4988 stepper driver for its ease of use with Arduino, micro-stepping capability for higher accuracy, low power consumption, and current control features.
We thoroughly tested the TFmini-S LiDAR sensor using the accompanying software provided by the manufacturer. This software allowed us to verify the sensor's performance, accuracy, and data output. Additionally, it enabled us to give commands to the LiDAR sensor, allowing us to customize its parameters to better suit our project needs. By running various tests and configurations, we ensured that the TFmini-S operated within the expected parameters and reliably provided the necessary distance measurements for our project. The successful testing confirmed that the sensor was suitable for integration into our 3D mapping system.

7. Preferred Serial to Parallel Converter IC Selection

Date: 2024/04/10

Task: Select a serial to parallel converter IC.
Details: We compared the FT232R and FT232RN USB UART ICs.
Outcome: We chose the FT232R due to availability constraints and its features such as:
Single chip USB to asynchronous serial data transfer interface.
No USB specific firmware programming required.
Integrated 1024 bit EEPROM, USB termination resistors, and clock generation.
Data transfer rates up to 3 Mbaud.
128 byte receive and 256 byte transmit buffer.
FTDI's royalty-free drivers, configurable CBUS I/O pins, and more.

8. Software Selection

Date: 2024/03/02

Task: Select software for programming and data visualization.
Details: We evaluated various software options for programming and 3D data processing.
Outcome: We chose Arduino for programming and Open3D for 3D data processing. Open3D is an open-source library with comprehensive tools for 3D data handling, robust visualization, geometry processing, various algorithms, and support for multiple file formats.

9. Power IC Selection and Stepper motor Working principle

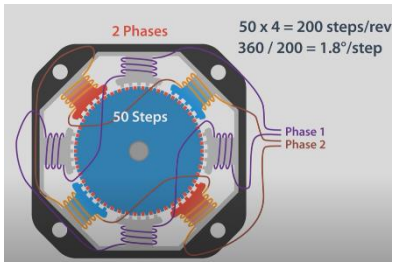
Date: 2024/04/13

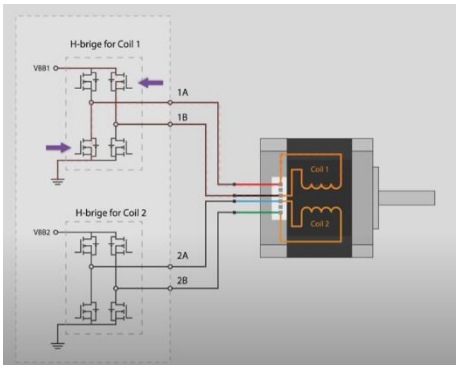
Task: Select a power IC.
Details: We compared the 78L05 Linear Voltage Regulator and the L7805CV 5-V Linear Regulator.
Outcome: We selected the L7805CV due to its specifications:
Fixed output voltage of 5V.
Maximum input voltage of 35V.
Maximum output current of 1.5A.
Line and load regulation of 100 mV.
Output voltage tolerance of 2%.
Power dissipation of 50°C/W junction to ambient and 5°C/W junction to case thermal resistance.
Operating temperature range of 0°C to 125°C.
Typical dropout voltage of 2V at 1A.
Quiescent current draw of 8 mA.

Stepper motor working principle:

We found that the NEMA 17 stepper motor operates on the principle of electromagnetism to achieve precise control of angular position without the need for feedback systems. The motor consists of a rotor, which is a permanent magnet, and a stator, which contains electromagnets. When electrical pulses are supplied to the motor's windings in a specific sequence, the electromagnetic fields generated in the stator interact with the permanent magnets in the rotor, causing the rotor to move in discrete steps.

We found that each pulse moves the rotor by a fixed angle, known as the step angle. For a typical NEMA 17 stepper motor, the step angle is 1.8 degrees, meaning the motor takes 200 steps to complete one full revolution (360 degrees). By controlling the sequence and frequency of these electrical pulses, the motor can achieve precise positioning and speed control.





10. Passive Component Selection

Date: 2024/04/15

Task: Select passive component sizes.
Details: We evaluated different SMD package sizes.
Outcome: We chose the 0805 SMD package size for its compatibility with our constraints and ease of handling.

11. Arduino Code implementation

Date : 2024/ 04/ 16

The program is designed to control a stepper motor using an A4988 driver and a TFmini-S LiDAR sensor interfaced with an ATmega328P microcontroller. It includes functionality for motor acceleration, deceleration, and constant speed rotation in both clockwise and counterclockwise directions. Additionally, it processes LiDAR sensor data to measure distances.

For the stepper motor, the stepPin (PD6) and dirPin (PD5) control the motor's step and direction, while MCU_Rx (PD2) and MCU_Tx (PD3) handle hardware serial communication. The stepsPerRevolution parameter defines the number of steps for a full revolution. The finalVelocity, displacement, and acceleration are used to calculate motor dynamics, and totalSteps calculates the total number of steps based on the displacement.

Microstepping configuration is handled by MS_1 (PB1), MS_2 (PB0), and MS_3 (PD7), which set the microstepping mode.

For the LiDAR sensor, variables dist, strength, check, and the uart array store the LiDAR measurement data. The HEADER constant defines the frame header for the LiDAR data package.

12. Started Register Level C program for our device

From Date : 2024/ 05/ 02 till 2024/05/05

Description: Today, I reviewed and worked on a C program designed to control a stepper motor using an A4988 driver and a TFmini-S LiDAR sensor interfaced with an ATmega328P microcontroller. The program includes functionality for motor acceleration, deceleration, and constant speed rotation in both clockwise and counterclockwise directions. Additionally, it processes LiDAR sensor data to measure distances. These were done referring to the Atmega328P – AU

Datasheet and following the videos,
youtube:https://www.youtube.com/watch?v=dT0xxaG1DhM&list=PLD7F7ED1F3505D8D5

Key Components and Parameters:

2. **Stepper Motor:**
    - **Control Pins:** STEP_PIN (PD6), DIR_PIN (PD5)
    - **Serial Communication Pins:** MCU_RX (PD2), MCU_TX (PD3)
    - **FT232RL Module Communication Pins:** RX (PD0), TX (PD1)
    - **Microstepping Pins:** MS_1 (PB1), MS_2 (PB0), MS_3 (PD7)
    - **Steps per Revolution:** 200
    - **Final Angular Velocity:** 6π rad/s
    - **Displacement:** π radians
    - **Calculated Angular Acceleration**
    - **Total Steps Based on Displacement**

3. **LiDAR Sensor:**
    - **Distance Measurement:** dist
    - **Signal Strength:** strength
    - **Frame Header:** HEADER (0x59)
    - **Data Array:** uart[9]

Implementation Notes:

- The program does not rely on any external libraries.
- The setup function initializes the pin configurations and calculates the motor dynamics.
- The main loop continuously reads LiDAR data and executes motor control sequences for both directions.
- Custom functions handle acceleration, constant rotation, and deceleration for both clockwise and counterclockwise directions.

13. Microchip Studio Installation

Date : : 2024/ 05/ 02

We installed Microchip Studio by downloading the latest version from the official Microchip Technology website. After completing the installation, I launched the IDE and created a new project for the ATmega328P-AU microcontroller. I selected the appropriate device and configured the project settings. Using the robust features of Microchip Studio, I wrote and compiled the C code necessary for our application. The IDE's comprehensive debugging tools and seamless integration with AVR programming hardware enabled me to efficiently program and test the ATmega328P-AU, ensuring that our code functioned correctly and met all project requirements.

6.1.2 Code Implemented using Arduino Syntax

We successfully implemented our code for 3D mapping using a LiDAR sensor with an Arduino, without utilizing any external libraries. Detailed information regarding this implementation can be found in section 6.1.1 of our Daily Log Entries, specifically in subsection 11 titled "Arduino Code Implementation."

```
/*
 * Original Code for 3D mapping with LiDAR
 * ----------------------------------------------------------------------------
 * Contributors: Dineth Perera and Seleka Deemantha
 * The implementation is done without using any external libraries.
 * ----------------------------------------------------------------------------
 * Description: This program is designed to control a stepper motor using an A4988 driver
 * and a TFmini-S LiDAR sensor interfaced with an ATmega328P microcontroller. The program
 * includes functionality for motor acceleration, deceleration, and constant speed rotation
 * in both clockwise and counterclockwise directions. Additionally, it processes LiDAR sensor
 * data to measure distances.
 * ----------------------------------------------------------------------------
 * Stepper Motor Parameters:
 * - stepPin (PD6) and dirPin (PD5) control the motor's step and direction.
 * - MCU_Rx (PD2) and MCU_Tx (PD3) handle hardware serial communication.
 * - stepsPerRevolution defines the number of steps for a full revolution.
 * - finalVelocity, displacement, and acceleration are used for calculating motor dynamics.
 * - totalSteps calculates the total number of steps based on the displacement.
 * ----------------------------------------------------------------------------
 * Microstepping Configuration:
 * - MS_1 (PB1), MS_2 (PB0), and MS_3 (PD7) configure the microstepping mode.
 * ----------------------------------------------------------------------------
 * LiDAR Sensor Parameters:
 * - dist, strength, check, and uart array store the LiDAR measurement data.
 * - HEADER defines the frame header for the LiDAR data package.
 * ----------------------------------------------------------------------------
 * References : Atmega328P - AU Datasheet
 *            : https://www.youtube.com/watch?v=dT0xxaG1DhM&list=PLD7F7ED1F3505D8D5
 *
 */

// Stepper motor parameters
const int stepPin = 10; // STEP pin  PD6
const int dirPin = 9;  // DIR pin PD5
const int MCU_Rx = 32 ; // hardware serial port  PD2
const int MCU_Tx= 1 ; // hardware receiver port  PD3
const int stepsPerRevolution = 200; // Number of steps per revolution
const float finalVelocity = 6 * PI; // Final angular velocity in rad/s
const float displacement = PI; // Displacement in radians
const float acceleration = finalVelocity * finalVelocity / (2 * displacement); // Angular
acceleration
int totalSteps = (displacement / (2 * PI)) * stepsPerRevolution;

// micro stepping
#define MS_1 13 // pin PB1
```

```
#define MS_2 12 // pin PB0
#define MS_3 11 // pin PD7


//Lidar sensor parameters
int dist;                     //actual distance measurements of LiDAR
int strength;                 //signal strength of LiDAR
int check;                    //save check value
int i;
int uart[9];                   //save data measured by LiDAR
const int HEADER = 0x59;      //frame header of data package



void setup() {
  Serial.begin (9600) ;
  // Set pins as outputs
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  pinMode(MCU_Rx, INPUT) ;
  pinMode(MCU_Tx, OUTPUT) ;

  //Microstepping
  pinMode(MS_1, OUTPUT ) ;
  pinMode(MS_2, OUTPUT) ;
  pinMode(MS_3 , OUTPUT) ;

  //Initially the microstepping is disabled
  digitalWrite(MS_1, LOW);
  digitalWrite(MS_2, LOW);
  digitalWrite(MS_3, LOW);
}


void loop() {
  Lidar_data() ;

  acceleration_cw();
  rotation_cw() ;
  deceleration_cw() ;

  acceleration_ccw();
  rotation_ccw() ;
  deceleration_ccw() ;


}

void acceleration_cw() {
```

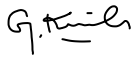
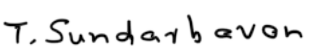
```
  float currentVelocity = 0.1; // Start with a small initial velocity to avoid division by
zero
  float stepInterval;

  digitalWrite(dirPin, HIGH); // clockwise direction
  for (int step = 0; step < totalSteps; step++) {
    // Calculate the step interval based on the current velocity
    stepInterval = 1.0 / (stepsPerRevolution * currentVelocity / (2 * PI));

    // Perform a step
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(stepInterval * 1e6); // Adjust the pulse width if necessary
    digitalWrite(stepPin, LOW);

    // Delay for the step interval
    delayMicroseconds(stepInterval * 1e6);

    // Update the current velocity using constant angular acceleration
    currentVelocity = sqrt(currentVelocity * currentVelocity + 2 * acceleration * (1.0 /
stepsPerRevolution));
  }
}
void rotation_cw(){
  int step_time = 1.0 / (stepsPerRevolution * finalVelocity / (2 * PI));
    // Rotate the motor in one direction
  digitalWrite(dirPin, HIGH);
  for (int i = 0; i < 200*11; i++) { // Change 200 to the number of steps for one rotation
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(step_time/2); // Adjust delay for step speed
    digitalWrite(stepPin, LOW);
    delayMicroseconds(step_time/2);
  }
}

void deceleration_cw() {
  float currentVelocity = finalVelocity;
  float stepInterval ;
  digitalWrite(dirPin, HIGH); // clockwise direction
  for (int step = 0; step < totalSteps; step++) {
    // Calculate the step interval based on the current velocity
    stepInterval = 1.0 / (stepsPerRevolution * currentVelocity / (2 * PI));

    // Perform a step
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(stepInterval * 1e6); // Adjust the pulse width if necessary
    digitalWrite(stepPin, LOW);
    // Delay for the step interval
    delayMicroseconds(stepInterval * 1e6);

    // Update the current velocity using constant angular acceleration
```

```
    currentVelocity = sqrt(currentVelocity * currentVelocity - 2 * acceleration * (1.0 /
stepsPerRevolution));
  }
}

void acceleration_ccw() {
  float currentVelocity = 0.1; // Start with a small initial velocity to avoid division by
zero
  float stepInterval;

  digitalWrite(dirPin, LOW); // clockwise direction
  for (int step = 0; step < totalSteps; step++) {
    // Calculate the step interval based on the current velocity
    stepInterval = 1.0 / (stepsPerRevolution * currentVelocity / (2 * PI));

    // Perform a step
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(stepInterval * 1e6); // Adjust the pulse width if necessary
    digitalWrite(stepPin, LOW);

    // Delay for the step interval
    delayMicroseconds(stepInterval * 1e6);

    // Update the current velocity using constant angular acceleration
    currentVelocity = sqrt(currentVelocity * currentVelocity + 2 * acceleration * (1.0 /
stepsPerRevolution));
  }
}

void rotation_ccw(){
  int step_time = 1.0 / (stepsPerRevolution * finalVelocity / (2 * PI));
    // Rotate the motor in one direction
  digitalWrite(dirPin, LOW);
  for (int i = 0; i < 200*11; i++) { // Change 200 to the number of steps for one rotation
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(step_time/2); // Adjust delay for step speed
    digitalWrite(stepPin, LOW);
    delayMicroseconds(step_time/2);
  }
}

void deceleration_ccw() {
  float currentVelocity = finalVelocity;
  float stepInterval ;
  digitalWrite(dirPin, HIGH); // clockwise direction
  for (int step = 0; step < totalSteps; step++) {
    // Calculate the step interval based on the current velocity
    stepInterval = 1.0 / (stepsPerRevolution * currentVelocity / (2 * PI));

    // Perform a step
```

```
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(stepInterval * 1e6); // Adjust the pulse width if necessary
    digitalWrite(stepPin, LOW);

    // Delay for the step interval
    delayMicroseconds(stepInterval * 1e6);

    // Update the current velocity using constant angular acceleration
    currentVelocity = sqrt(currentVelocity * currentVelocity - 2 * acceleration * (1.0 /
stepsPerRevolution));
  }
}

void Lidar_data() {
  if (Serial.available())                //check if serial port has data input
  {
    if (Serial.read() == HEADER)        //assess data package frame header 0x59
    {
      uart[0] = HEADER;
      if (Serial.read() == HEADER)      //assess data package frame header 0x59
      {
        uart[1] = HEADER;
        for (i = 2; i < 9; i++)         //save data in array
        {
          uart[i] = Serial.read();
        }
        check = uart[0] + uart[1] + uart[2] + uart[3] + uart[4] + uart[5] + uart[6] +
uart[7];
        if (uart[8] == (check & 0xff))        //verify the received data as per protocol
        {
          dist = uart[2] + uart[3] * 256;     //calculate distance value
          Serial.write(dist);                 //output measure distance value of LiDAR
        }
      }
    }
  }
```

The previously used and coded Arduino part of the project included a script that read data from the LiDAR sensor without utilizing any libraries. The script established serial communication with the LiDAR sensor, read the incoming data, processed it by splitting it into components (x, y, z coordinates), and converted these components from strings to floats for further use. The processed data was printed to the serial monitor for real-time observation and debugging.

Signed Declaration from the AMR group

1. 210019E - Abithan A. : 
2. 210498T - Priyankan V. : 
3. 210292G - Kirushanth G. : 
4. 210624E - Sundarbavan T. : 

References

[1]"Introduction To Lidar"

https://cloud.sdsc.edu/v1/AUTH_opentopography/www/shortcourses/17Utah/17Utah_Crosby_introLidar.pdf.

[2] " RGB and LiDAR Fusion-based 3D Semantic Mapping"

https://web.stanford.edu/class/cs231a/prev_projects_2022/CS231A_Final_Report__1_.pdf.

[3] "Datasheet of the RP Lidar"

https://www.generationrobots.com/media/rplidar-a1m8-360-degree-laser-scanner-development-kit-datasheet-1.pdf

[4] T. Ito, "LiDAR for Automated Driving Era," 2021 IEEE CPMT SymposiumJapan(ICSJ),2021,pp.37-40,doi: 10.1109/ICSJ52620.2021.9648873.

[5] "Atmega328P – AU Datasheet"

https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[6] "Nema 17 Stepper Motor datasheet"

https://pages.pbclinear.com/rs/909-BFY-775/images/Data-Sheet-Stepper-Motor-Support.pdf

[7] "FT232RL Chip Datasheet"

https://ftdichip.com/wp-content/uploads/2020/08/DS_FT232R.pdf

[8] " Datasheet of the A4988 Stepper Motor Driver"

https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf

[9] "TFmini Datasheet"

https://cdn.sparkfun.com/assets/8/a/f/a/c/16977-TFMini-S_-_Micro_LiDAR_Module-Product_Manual.pdf

[10] "AVR microcontroller programming"

https://www.youtube.com/watch?v=dT0xxaG1DhM&list=PLD7F7ED1F3505D8D5